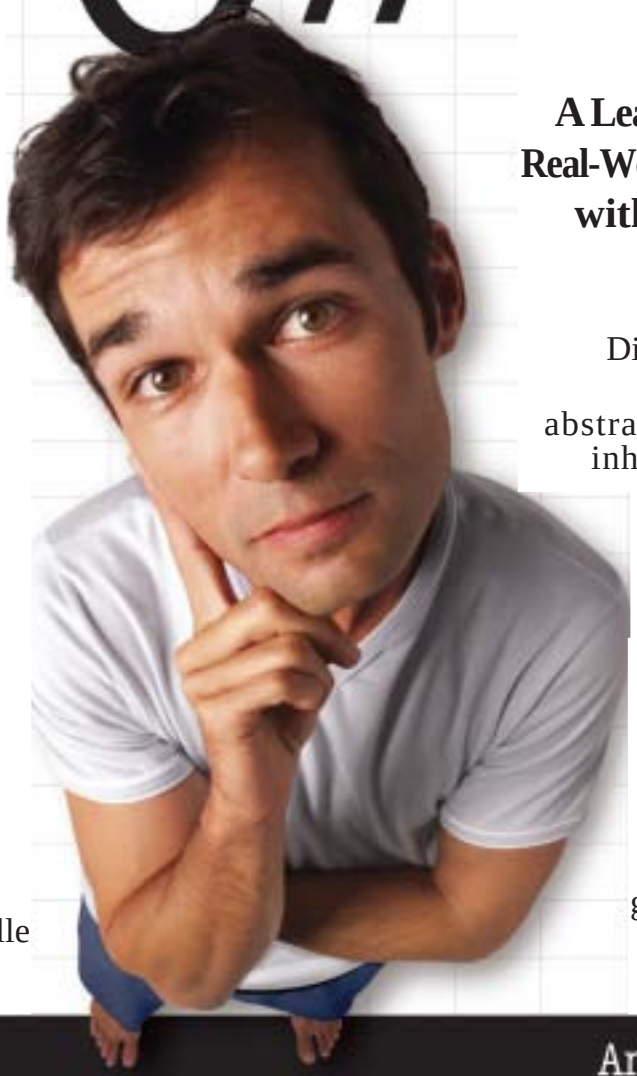
A Brain-Friendly Guide

Covers C# 3.0 and Visual Studio 2008

HeadFirst C#



Boss your data around with LINQ



A Learner's Guide to Real-World Programming with C# and .NET

Discover the secrets of abstraction and inheritance



Build a fully functional retro classic arcade game





Learn how extension methods helped Sue bend the rules in Objectville



See how Jim used generic collections to wrangle his data

Andrew Stellman & Jennifer Greene

O'REILLY®

Head First C#

by Andrew Stellman and Jennifer Greene

Copyright © 2008 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Series Creators: Kathy Sierra, Bert Bates

Series Editor: Brett D. McLaughlin

Design Editor: Louise Barr

Cover Designers: Louise Barr, Steve Fehler

Production Editor: Sanders Kleinfeld

Proofreader: Colleen Gorman

Indexer: Julie Hawks

Page Viewers: Quentin the whippet and Tequila the pomeranian

Printing History:

November 2007: First Edition.



The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. The *Head First* series designations, *Head First C#*, and related trade dress are trademarks of O'Reilly Media, Inc.

Microsoft, Windows, Visual Studio, MSDN, the .NET logo, Visual Basic and Visual C# are registered trademarks of Microsoft Corporation.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

No bees, space aliens, or comic book heroes were harmed in the making of this book.

ISBN-10: 0-596-51482-4

ISBN-13: 978-0-596-51482-2

[M]

Thanks for buying our book! We really love writing about this stuff, and we hope you get a kick out of reading it, ..

Andrew



This photo (and the photo of the Gowanus Canal) by Nisha Sondhe

Andrew Stellman, despite being raised a New Yorker, has lived ill Pittsburgh *twice*. The lir't tim.· was when he graduated from C"rnegie M1eU.n's School •f C...iput Scienc'e, a"d then again whe" he and J..nny starting their consulting business and writing th";f lil'>t Ioo,k foi O'Reilly.

When h.. "" wed back 10 his Iu"'"..tow his first job afi.er college " ;" ''' " programn at EMI-Capitol Records which .nmallv Ill"'!" sense.. since he went I' LaGuardia H ij; h Schoo1of Music .m d. Art and 1/ie Performing Arts to study cell ;'ud jazz bass guitar. II e and Jenny first ""rkec1together at u..« sa"".. financial software company, whei...he was managing a to;")]l of programmers. H.·'" had the privilege "f w king " id, some pretty .·mazing programmers ov rh•• years, and like '" think that he's learned a fi w things from the",.

When he's not writing books Andrew keeps himself busy writing useless (I fun ' software, playing m",i,'{biu video games even nlur,'). studying taiji and aikid,• having a girlfriend named Lis.. and owning a pomeranian.

Jennifer Greene studi..d philosophy in college but. like everyone else in the field. couldn't find a job doing it. I.uckily, she's a great sc,liware tes t,;,. so she started ..u, doing il at ,n, online servic... and that's the first time she really got a gr..1sen if what project management was.

Sl1e nu' d to Nev York ill 1998 ,,, test software at a financial software company. She managed a 1t'UI1,,1" te·.ters ·LI a ally cool startup th.u did artifi ial inr..lligenc and natural language processing.

Sinc IJ'"Il•• h..·; travele I all ,>),,.. ,he world L.. w 'lk " it h different software teams and build all kinds of cool projects.

Sl1e loves traveling, watching Bollywood movies reading the occasional "" mi., book, waiting li ,.. • he r Xbox ,., be repaired. drinking carloads of carbonat...1bev..rages, and owning a whippet.

Jenny and Andrew have been building software and writing about software engineering together since they first met in 1998 The... first book, *Applied Software Project Manage...ent*, was publ..hed by O'Reilly in 2005 They publ..hed their first book in the Head First series, *Head First PMP*, in 2007

They founded Stellman & Greene Consulting in 2003 to build a really neat software project for scientists studying herbicide exposur .n Vietna... .ets. When they're not building software or writing books, they do a lot of speaking at conferences and meetings of software engineers, architects and project managers.

Check out the... blog, *Building Better Software* http://www.stellman-greene.com

Table of Contents (Summary)

	Intro	xxix
1	Get productive with C#: *Visual Applications, in 10 minutes or less*	1
2	It's All Just Code: *Under the hood*	43
3	Objects Get Oriented: *Making code make sense*	85
4	Types and References: *It's 10:00. Do you know where your data is?*	123
	C# Lab 1: *A Day at the Races*	163
5	Encapsulation: *Keep your privates… private*	173
6	Inheritance: *Your object's family tree*	205
7	Interfaces and abstract classes: *Making classes keep their promises*	251
8	enums and collections: *Storing lots of data*	309
	C# Lab 2: *The Quest*	363
9	Reading and writing files: *Save the byte array, save the world*	385
10	Exception handling: *Putting Out Fires Gets Old*	439
11	events and delegates: *What Your Code Does When You're Not Looking*	483
12	Review and preview: *Knowledge, Power, and Building Cool Stuff*	515
13	Controls and graphics: *Make it pretty*	563
14	Captain Amazing: *The Death of the Object*	621
15	LINQ: *Get control of your data*	653
	C# Lab 3: *Invaders*	681

Table of Contents (the real thing)

Intro

Your brain on C#. You're sitting around trying to *learn* something, but your *brain* keeps telling you all that learning *isn't important*. Your brain's saying, "Better leave room for more important things, like which wild animals to avoid and whether nude archery is a bad idea." So how *do* you trick your brain into thinking that your life really depends on learning C#?

Who is this book for?	xxx
We know what you're thinking	xxxi
Metacognition	xxxiii
Bend your brain into submission	xxxv
What you need for this book	xxxvi
Read me	xxxii
The technical review team	xxxiv
Acknowledgments	xxxv

get productive with C#

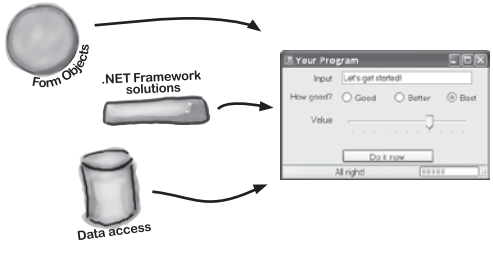
1 Visual Applications, in 10 minutes or less

Want to build great programs really fast?

With C#, you've got a **powerful programming language** and a **valuable tool** at your fingertips. With the **Visual Studio IDE**, you'll never have to spend hours writing obscure code to get a button working again. Even better, you'll be able to **focus on getting your work done**, rather than remembering which method parameter was for the *name* for a button, and which one was for its *label*. Sound appealing? Turn the page, and let's get programming.

Why you should learn C# 2
C# and the Visual Studio IDE make lots of things easy 3
Help the CEO go paperless 4
Get to know your users' needs before you startbuilding your program 5
Here's what you're going to build 6
What you do in Visual Studio… 8
What Visual Studio does for you… 8
Develop the user interface 12
Visual Studio, behind the scenes 14
Add to the auto-generated code 15
You can already run your application 16
We need a database to store our information 18
Creating the table for the Contact List 20
The blanks on contact card are columns in our People table 22
Finish building the table 25
Diagram your data so your application can access it 26
Insert your card data into the database 28
Connect your form to your database objects with a data source 30
Add database-driven controls to your form 32
Good apps are intuitive to use 34
How to turn YOUR application into EVERYONE'S application 37
Give your users the application 38
You're NOT done: test your installation 39
You built a complete data-driven application 40





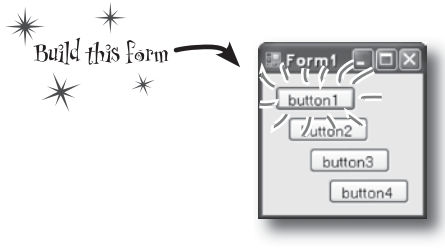
it's all just code

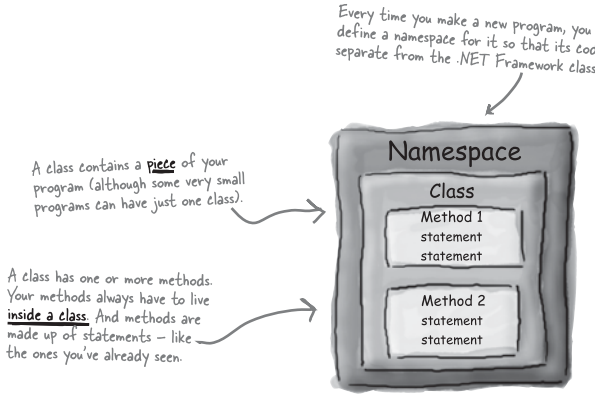
2 Under the Hood

You're a programmer, not just an IDE-user.

You can get a lot of work done using the IDE. But there's only so far it can take you. Sure, there are a lot of **repetitive tasks** that you do when you build an application. And the IDE is great at doing those things for you. But working with the IDE is *only the beginning*. You can get your programs to do so much more—and **writing C# code** is how you do it. Once you get the hang of coding, there's *nothing* your programs can't do.

When you're doing this… 44
…the IDE does this 45
Where programs come from 46
The IDE helps you code 48
When you change things in the IDE, you're also changing your code 50
Anatomy of a program 52
Your program knows where to start 54
You can change your program's entry point 56
Two classes can be in the same namespace 61
Your programs use variables to work with data 62
C# uses familiar math symbols 64
Loops perform an action over and over again 65
Time to start coding 66
if/else statements make decisions 67
Set up conditions and see if they're true 68







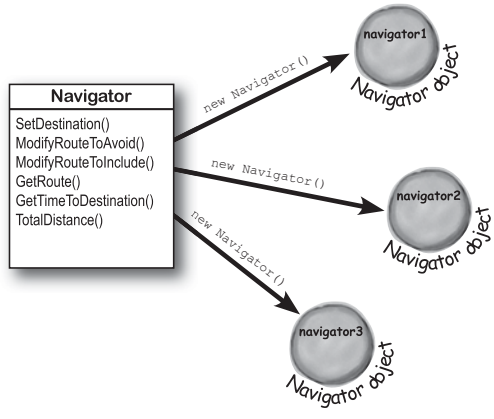
objects get oriented

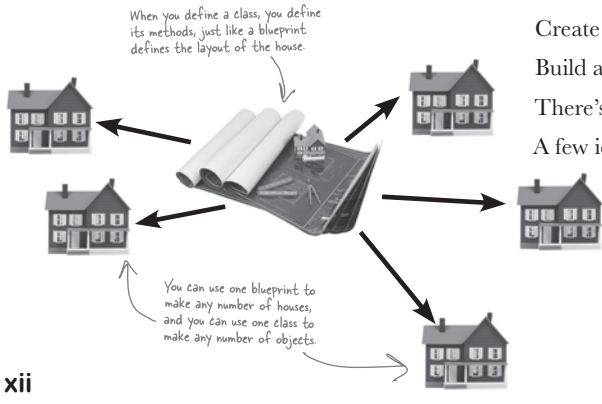
3 Making Code Make Sense

Every program you write solves a problem.

When you're building a program, it's always a good idea to start by thinking about what *problem* your program's supposed to solve. That's why ***objects*** are really useful. They let you structure your code based on the problem it's solving, so that you can spend your time *thinking about the problem* you need to work on rather than getting bogged down in the mechanics of writing code. When you use objects right, you end up with code that's *intuitive* to write, and easy to read and change.

How Mike thinks about his problems	86
How Mike's car navigation system thinks about his problems	87
Mike's Navigator class has methods to set and modify routes	88
Use what you've learned to build a simple application	89
Mike gets an idea	90
Mike can use objects to solve his problem	91
You use a class to build an object	92
When you create a new object from a class, it's called an instance of that class	93
A better solution… brought to you by objects!	94
An instance uses fields to keep track of things	98
Let's create some instances!	99
Thanks for the memory	100
What's on your program's mind	101
You can use class and method names to make your code intuitive	102
Give your classes a natural structure	104
Class diagrams help you organize your classes so they make sense	106
Build a class to work with some guys	110
Create a project for your guys	111
Build a form to interact with the guys	112
There's an even easier way to initialize objects	115
A few ideas for designing intuitive classes	116





4 types and references

It's 10:00. Do you know where your data is?

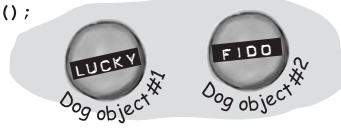
Data type, database, Lieutenant Commander Data… it's all important stuff. Without data, your programs are useless. You need **information** from your users, and you use that to look up or produce new information, to give back to them. In fact, almost everything you do in programming involves **working with data** in one way or another. In this chapter, you'll learn the ins and outs of C#'s **data types**, how to work with data in your program, and even figure out a few dirty secrets about **objects** (*psstt… objects are data, too*).

The variable's type determines what kind of data it can store 124
A variable is like a data to-go cup 126
10 pounds of data in a 5 pound bag 127
Even when a number is the right size, you can't just assign it to any variable 128
When you cast a value that's too big, C# will adjust it automatically 129
C# does some casting automatically 130
When you call a method, the variables must match the types of the parameters 131
Combining = with an operator 136
Objects are variables, too 137
Refer to your objects with reference variables 138
References are like labels for your object 139
If there aren't any more references, your object gets garbage collected 140
Multiple references and their side effects 142
Two references means TWO ways to change an object's data 147
A special case: arrays 148
Arrays can contain a bunch of reference variables, too 149
Welcome to Sloppy Joe's Budget House o' Discount Sandwiches! 150
Objects use references to talk to each other 152
Where no object has gone before 153


```
Dog fido;
Dog lucky = new Dog();
```

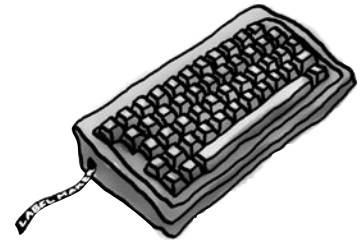



```
fido = new Dog();
```




```
lucky = null;
```



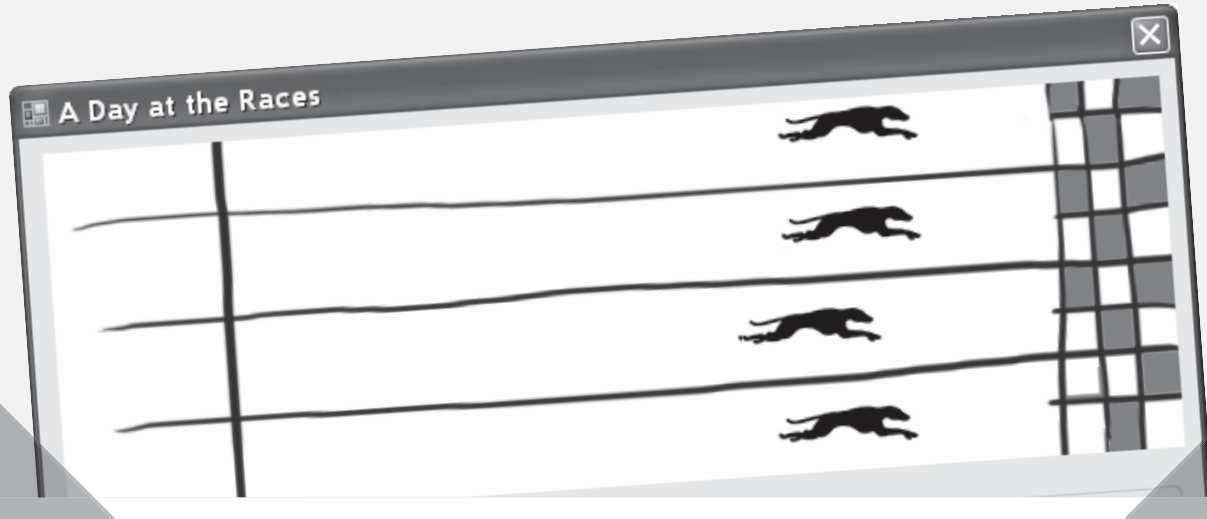


C# Lab 1

A Day at the Races

Joe, Bob, and Al love going to the track, but they're tired of losing all their money. They need you to build a simulator for them so they can figure out winners before they lay their money down. And, if you do a good job, they'll cut you in on their profits.

The Spec: Build a Racetrack Simulator 164
The Finished Product 172



encapsulation

5 Keep your privates... private

Ever wished for a little more privacy?

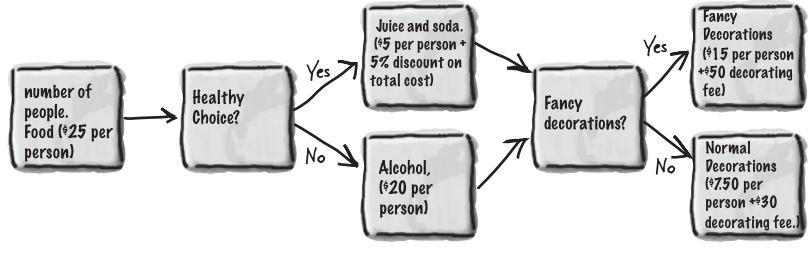
Sometimes your objects feel the same way. Just like you don't want anybody you don't trust reading your journal, or paging through your bank statements, good objects don't let ***other*** objects go poking around their properties. In this chapter, you're going to learn about the power of **encapsulation**. You'll **make your object's data private**, add methods to **protect how that data is accessed**.







Kathleen is an event planner	174
What does the estimator do?	175
Kathleen's Test Drive	180
Each option should be calculated individually	182
It's easy to accidentally misuse your objects	184
Encapsulation means keeping some of the data in a class private	185
Use encapsulation to control access to your class's methods and fields	186
But is the realName field REALLY protected?	187
Private fields and methods can only be accessed from inside the class	188
A few ideas for encapsulating classes	191
Encapsulation keeps your data pristine	192
Properties make encapsulation easier	193
Build an application to test the Farmer class	194
Use automatic properties to finish the class	195
What if we want to change the feed multiplier?	196
Use a constructor to initialize private fields	197



inheritance

6 Your object's family tree

Sometimes you *DO* want to be just like your parents.

Ever run across an object that ***almost*** does exactly what you want ***your*** object to do? Found yourself wishing that if you could just ***change a few things***, that object would be perfect? Well that's just one reason that **inheritance** is one of the most powerful concepts and techniques in the C# language. Before you're through this chapter, you'll learn how to **subclass** an object to get its behavior, but keep the **flexibility** to make changes to that behavior. You'll **avoid duplicate code**, **model the real world** more closely, and end up with code that's **easier to maintain**.







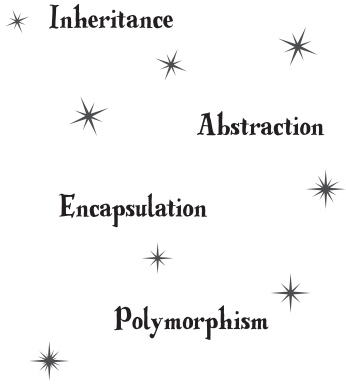
Kathleen does birthday parties, too 206
We need a BirthdayParty class 207
One more thing... can you add a $100 fee for parties over 12? 213
When your classes use inheritance, you only need to write your code once 214
Build up your class model by starting general and getting more specific 215
How would you design a zoo simulator? 216
Use inheritance to avoid duplicate code in subclasses 217
Different animals make different noises 218
Think about how to group the animals 219
Create the class hierarchy 220
Every subclass extends its base class 221
Use a colon to inherit from a base class 222
We know that inheritance adds the base class fields, properties, and methods to the subclass... 225
A subclass can override methods to change or replace methods it inherited 226
Any place where you can use a base class, you can use one of its subclasses instead 227
A subclass can access its base class using the base keyword 232
When a base class has a constructor, your subclass needs one too 233
Now you're ready to finish the job for Kathleen! 234
Build a beehive management system 239
First you'll build the basic system 240
Use inheritance to extend the bee management system 245

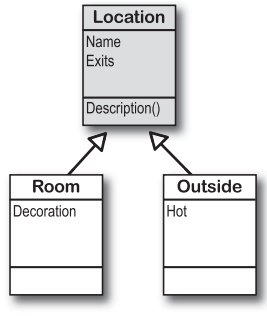
interfaces and abstract classes

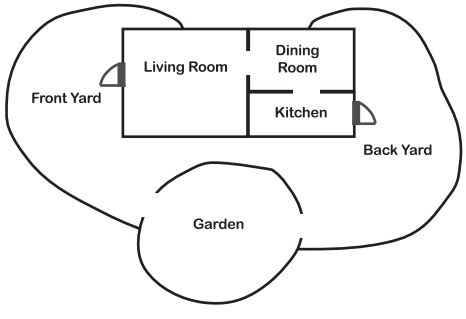
7 Making classes keep their promises

Actions speak louder than words.

Sometimes you need to group your objects together based on the **things they can do** rather than the classes they inherit from. That's where **interfaces** come in—they let you work with any class that can do the job. But with **great power comes great responsibility**, and any class that implements an interface must promise to **fulfill all of its obligations**... or the compiler will break their kneecaps, see?







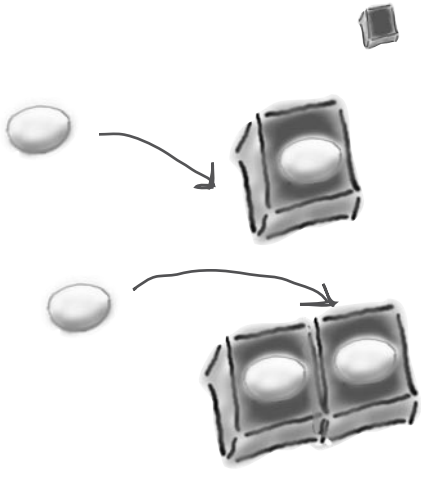
Let's get back to bee-sics	252
We can use inheritance to create classes for different types of bees	253
An interface tells a class that it must implement certain methods and properties	254
Use the interface keyword to define an interface	255
Get a little practice using interfaces	256
Now you can create an instance of NectarStinger that does both jobs	257
Classes that implement interfaces have to include ALL of the interface's methods	258
You can't instantiate an interface, but you can reference an interface	260
Interface references work just like object references	261
You can find out if a class implements a certain interface with "is"	262
Interfaces can inherit from other interfaces	263
The RoboBee 4000 can do a worker bee's job without using valuable honey	264
is tells you what an object implements, as tells the compiler how to treat your object	265
A CoffeeMaker is also an Appliance	266
Upcasting works with both objects and interfaces	267
Downcasting lets you turn your appliance back into a coffee maker	268
Upcasting and downcasting work with interfaces, too	269
There's more than just public and private	273
Access modifiers change scope	274
Some classes should never be instantiated	277
An abstract class is like a cross between a class and an interface	278
Some classes should never be instantiated	280
An abstract method doesn't have a body	281
Polymorphism means that one object can take many different forms	289

8 enums and collections

Storing lots of data

When it rains, it pours.

In the real world, you don't get to handle your data in tiny little bits and pieces. No, your data's going to come at you in **loads, piles and bunches**. You'll need some pretty powerful tools to organize all of it, and that's where **collections** come in. They let you **store, sort and manage** all the data that your programs need to pore through. That way you can think about writing programs to work with your data, and let the collections worry about keeping track of it for you.





Strings don't always work for storing categories of data 310
Enums let you enumerate a set of valid values 311
Enums let you represent numbers with names 312
We could use an array to create a deck of cards... 315
Arrays are hard to work with 316
Lists make it easy to store collections of... anything 317
Lists are more flexible than arrays 318
Lists shrink and grow dynamically 321
List objects can store any type 322
Collection initializers work just like object initializers 326
Let's create a list of Ducks 327
Lists are easy, but SORTING can be tricky 328
Two ways to sort your ducks 329
Use IComparer to tell your List how to sort 330
Create an instance of your comparer object 331
IComparer can do complex comparisons 332
Use a dictionary to store keys and values 335
The Dictionary Functionality Rundown 336
Your key and value can be different types, too 337
You can build your own overloaded methods 343
And yet MORE collection types... 355
A queue is FIFO — First In, First Out 356
A stack is LIFO — Last In, First Out 357

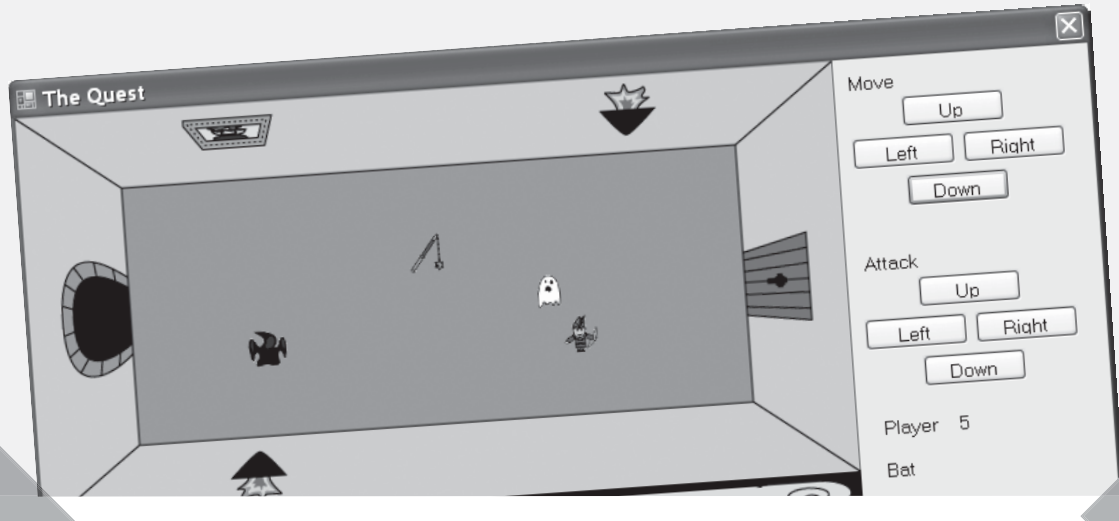
C# Lab 2

The Quest

Your job is to build an adventure game where a mighty adventurer is on a quest to defeat level after level of deadly enemies. You'll build a turn-based system, which means the player makes one move and then the enemies make one move. The player can move or attack, and then each enemy gets a chance to move and attack. The game keeps going until the player either defeats all the enemies on all seven levels or dies.

The spec: build an adventure game 364

The fun's just beginning! 484

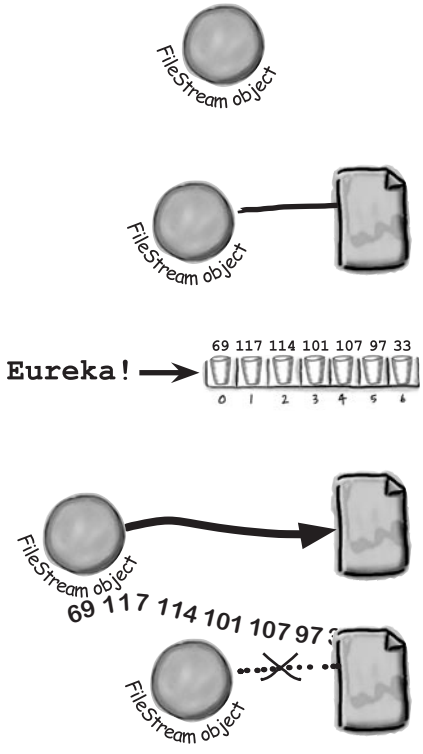


9 reading and writing files

Save the byte array, save the world

Sometimes it pays to be a little persistent.

So far, all of your programs have been pretty short-lived. They fire up, run for a while, and shut down. But that's not always enough, especially when you're dealing with important information. You need to be able to **save your work**. In this chapter, we'll look at how to **write data to a file**, and then how to **read that information back in** from a file. You'll learn about the .NET **stream classes**, and also take a look at the mysteries of **hexadecimal** and **binary**.



C# uses streams to read and write data 386
Different streams read and write different things 387
A FileStream writes bytes to a file 388
Reading and writing takes two objects 393
Data can go through more than one stream 394
Use built-in objects to pop up standard dialog boxes 397
Dialog boxes are objects, too 399
Use the built-in File and Directory classes to work with files and directories 400
Use File Dialogs to open and save files 403
IDisposable makes sure your objects are disposed properly 405
Avoid file system errors with using statements 406
Writing files usually involves making a lot of decisions 412
Use a switch statement to choose the right option 413
Add an overloaded Deck() constructor that reads a deck of cards in from a file 415
What happens to an object when it's serialized? 417
But what exactly IS an object's state? What needs to be saved? 418
When an object is serialized, all of the objects it refers to get serialized too... 419
Serialization lets you read or write a whole object all at once 420
If you want your class to be serializable, mark it with the [Serializable] attribute 421
.NET converts text to Unicode automatically 425
C# can use byte arrays to move data around 426
Use a BinaryWriter to write binary data 427
You can read and write serialized files manually, too 429
StreamReader and StreamWriter will do just fine 433

exception handling

10 Putting out fires gets old

Programmers aren't meant to be firefighters.

You've worked your tail off, waded through technical manuals and a few engaging Head First books, and you've reached the pinnacle of your profession: **master programmer**. But you're still getting pages from work because **your program crashes**, or **doesn't behave like it's supposed to**. Nothing pulls you out of the programming groove like having to fix a strange bug . . . but with **exception handling**, you can write code to **deal with problems** that come up. Better yet, you can even react to those problems, and **keep things running**.



Brian needs his excuses to be mobile 440

When your program throws an exception, .NET generates an Exception object. 444

Brian's code did something unexpected 446

All exception objects inherit from Exception 448

The debugger helps you track down and prevent exceptions in your code 449

Use the IDE's debugger to ferret out exactly what went wrong in the excuse manager 450

Uh-oh—the code's still got problems... 453

Handle exceptions with try and catch 455

What happens when a method you want to call is risky? 456

Use the debugger to follow the try/catch flow 458

If you have code that ALWAYS should run, use a finally block 460

Use the Exception object to get information about the problem 465

Use more than one catch block to handle multiple types of exceptions 466

One class throws an exception, another class catches the exception 467

Bees need an OutOfHoney exception 468

An easy way to avoid a lot of problems: using gives you try and finally for free 471

Exception avoidance: implement IDisposable to do your own clean up 472

The worst catch block EVER: comments 474

Temporary solutions are okay (temporarily) 475

A few simple ideas for exception handling 476

Brian finally gets his vacation... 481

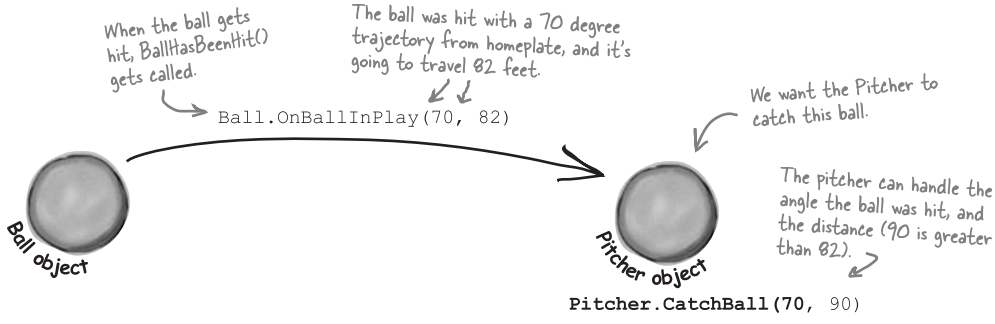
11 events and delegates

What your code does when you're not looking

Your objects are starting to think for themselves.

You can't always control what your objects are doing. Sometimes things... happen. And when they do, you want your objects to be smart enough to **respond to anything** that pops up. And that's what events are all about. One object *publishes* an event, other objects *subscibe*, and everyone works together to keep things moving. Which is great, until you've got too many objects responding to the same event. And that's when **callbacks** will come in handy.

Ever wish your objects could think for themselves?	484
But how does an object KNOW to respond?	484
When an EVENT occurs... objects listen	485
One object raises its event, others listen for it...	486
Then, the other objects handle the event	487
Connecting the dots	488
The IDE creates event handlers for you automatically	492
The forms you've been building all use events	498
Connecting event senders with event receivers	500
A delegate STANDS IN for an actual method	501
Delegates in action	502
Any object can subscribe to a public event...	505
Use a callback instead of an event to hook up exactly one object to a delegate	507
Callbacks use delegates, but NOT events	508



review and preview

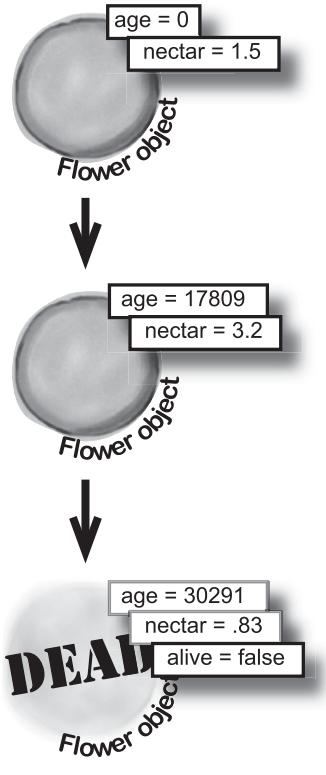
12 Knowledge, power, and building cool stuff

Learning's no good until you BUILD something.

Until you've actually written working code, it's hard to be sure if you really *get* some of the tougher concepts in C#. In this chapter, we're going to learn about some new odds and ends: **timers** and dealing with collections using **LINQ** (to name a couple). We're also going to build phase I of a **really complex application**, and make sure you've got a good handle on what you've already learned from earlier chapters. So buckle up... it's time to build some **cool software**.

You've come a long way, baby 516
We've also become beekeepers 517
The beehive simulator architecture 518
Building the beehive simulator 519
Life and death of a flower 523
Now we need a Bee class 524
Filling out the Hive class 532
The hive's Go() method 533
We're ready for the World 534
We're building a turn-based system 535
Giving the bees behavior 542
The main form tells the world to Go() 544
We can use World to get statistics 545
Timers fire events over and over again 546
The timer's using a delegate behind the scenes 547
Let's work with groups of bees 554
A collection collects... DATA 555
LINQ makes working with data in collections and databases easy 557

Life and death of a flower



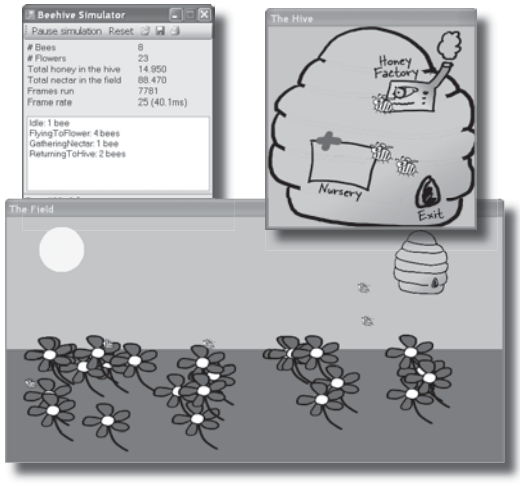
13 controls and graphics

Make it pretty

Sometimes you have to take graphics into your own hands.

We've spent a lot of time on relying on controls to handle everything visual in our applications. But sometimes that's not enough—like when you want to **animate a picture**. And once you get into animation, you'll end up **creating your own controls** for your .NET programs, maybe adding a little **double buffering**, and even **drawing directly onto your forms**. It all begins with the **Graphics** object, **Bitmap**s, and a determination to not accept the graphics status quo.

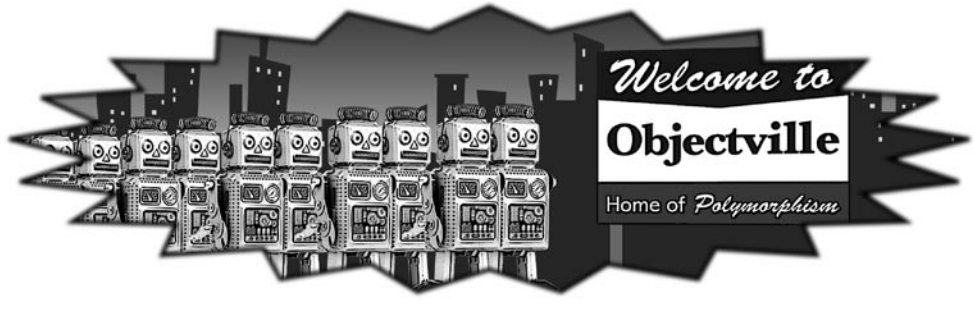
You've been using controls all along to interact with your programs	564
Form controls are just objects	565
Add a renderer to your architecture	568
Controls are well-suited for visual display elements	570
Build your first animated control	573
Your controls need to dispose their controls, too!	577
A UserControl is an easy way to build a control	578
Add the hive and field forms to the project	582
Build the Renderer	583
Let's take a closer look at those performance issues	590
You resized your Bitmaps using a Graphics object	592
Your image resources are stored in Bitmap objects	593
Use System.Drawing to TAKE CONTROL of graphics yourself	594
A 30-second tour of GDI+ graphics	595
Use graphics to draw a picture on a form	596
Graphics can fix our transparency problem...	601
Use the Paint event to make your graphics stick	602
A closer look at how forms and controls repaint themselves	605
Double buffering makes animation look a lot smoother	608
Double buffering is built into forms and controls	609
Use a Graphics object and an event handler for printing	614
PrintDocument works with the print dialog and print preview window objects	615



14 CAPTAIN AMAZING
THE DEATH OF THE OBJECT

Captain Amazing, Objectville's most amazing object pursues his arch-nemesis... 622
Your last chance to DO something... your object's finalizer 628
When EXACTLY does a finalizer run? 629
Dispose() works with using, finalizers work with garbage collection 630
Finalizers can't depend on stability 632
Make an object serialize itself in its Dispose() 633
Meanwhile, on the streets of Objectville... 636
A struct *looks* like an object... 637
..but *isn't* on the heap 637
Values get copied, references get assigned 638
Structs are value types; objects are reference types 639
The stack vs. the heap: more on memory 641
Captain Amazing... not so much 645
Extension methods add new behavior to EXISTING classes 646
Extending a fundamental type: string 648





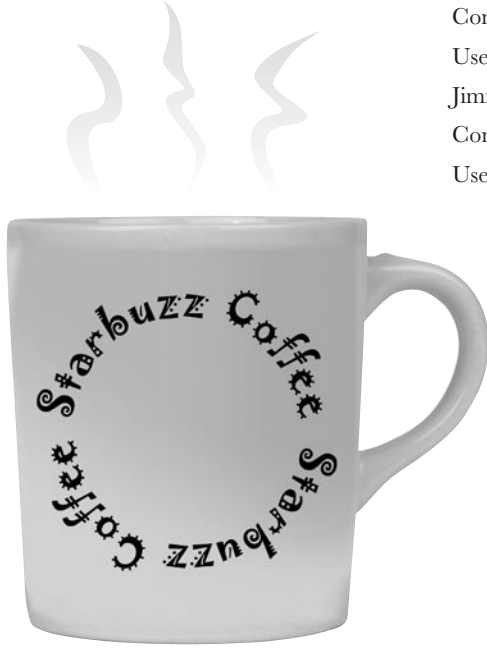
LINQ

15 Get control of your data

It's a data-driven world... you better know how to live in it.

Gone are the days when you could program for days, even weeks, without dealing with **loads of data**. But today, ***everything is about data***. In fact, you'll often have to work with data from **more than one place**... and in more than one format. Databases, XML, collections from other programs... it's all part of the job of a good C# programmer. And that's where **LINQ** comes in. LINQ not only lets you **query data** in a simple, intuitive way, but it lets you **group data**, and **merge data from different data sources**.

An easy project...	654
...but the data's all over the place	655
LINQ can pull data from multiple sources	656
.NET collections are already set up for LINQ	657
LINQ makes queries easy	658
LINQ is simple, but your queries don't have to be	659
LINQ is versatile	662
LINQ can combine your results into groups	667
Combine Jimmy's values into groups	668
Use join to combine two collections into one query	671
Jimmy saved a bunch of dough	672
Connect LINQ to a SQL database	674
Use a join query to connect Starbuzz and Objectville	678



C# Lab 3

Invaders

In this lab you'll pay homage to one of the most popular, revered and replicated icons in video game history, a game that needs no further introduction. It's time to build Invaders.

The grandfather of video games 682

And yet there's more to do... 701



leftovers

The top 5 things we wanted to include in this book

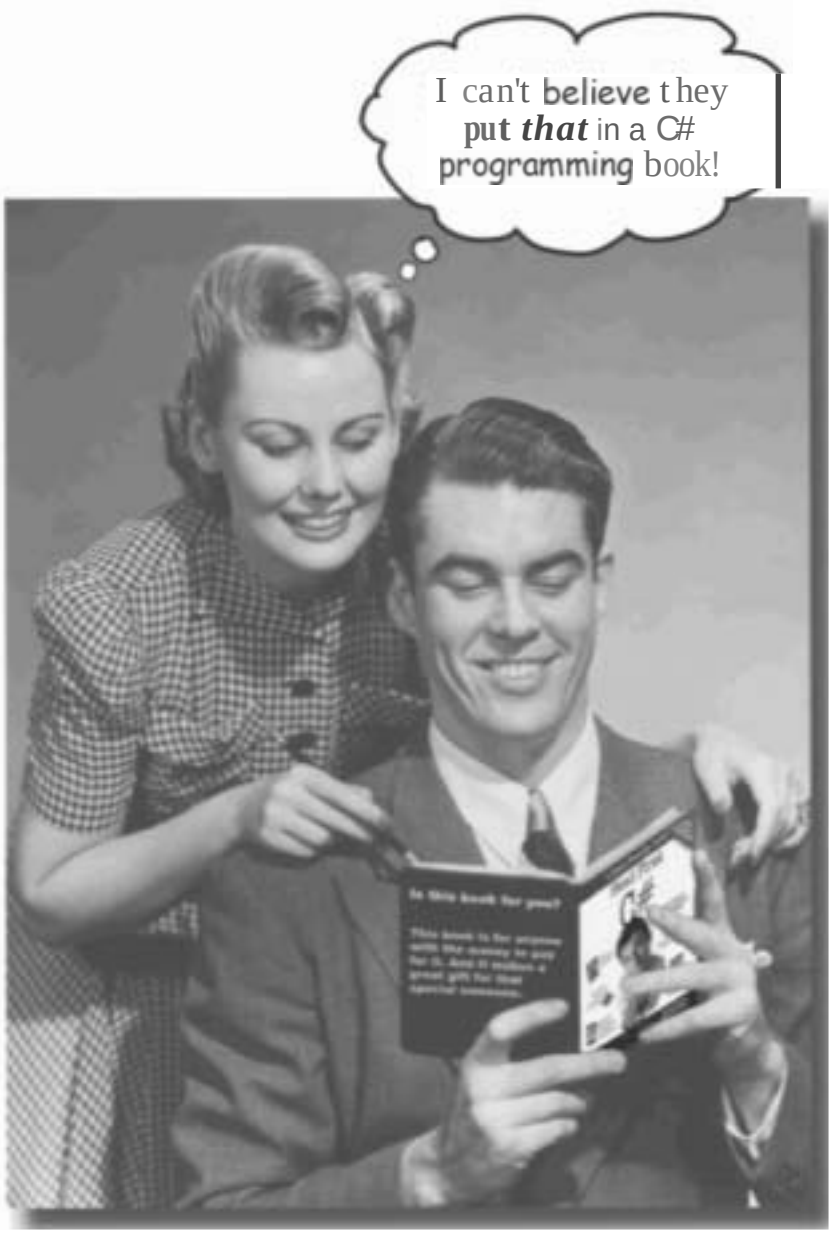
The fun's just beginning!

We've shown you a lot of great tools to build some really **powerful software** with C#. But there's no way that we could include **every single tool, technology or technique** in this book—there just aren't enough pages. We had to make some *really tough choices* about what to include and what to leave out. Here are some of the topics that didn't make the cut. But even though we couldn't get to them, we still think that they're **important and useful**, and we wanted to give you a small head start with them.

#1 LINQ to XML	704
#2 Refactoring	706
#3 Some of our favorite Toolbox components	708
#4 Console Applications	710
5 Windows Presentation Framework	712
Did you know that C# and the .NET Framework can...	714



Intro



In this section, we answer the burning question:
"So why DID they put that in a C# programming book?"

Who is this book for?

If you ";111 unsws-r "yes" (o all or these:

1. Do you want to learn C#?
2. Do you like to tlnker-do you een by doing. rather than just reading?
3. Do you prefer stimulating dinner party conversation 10dry, dull, academic lectures?

this book is for you.

Who should probably back away frolll this book?

If you can answer "yes" (o uny or these:

1. Does the idea of wrrtlng a lot of code make you bored and a little twitchy?
2. Are you a kick-butt C++ or Java programmer looking for a reference book?
3. Are you afraid 10 try something different? Would you rather have a root canal than mix stripes with plaid? Do you believe that a technical book can't be serious if C# concepts are anthropomorphIZed?



this book is not for you.

[Note from marketing: this book is for anyone with a credit card.]

We know what you're **thinking.**

"11,,,, can *this* be a serious C# programming book?"

"What's with .,11 Uhe graphics?"

"Can I actually *learn* it this way?"

And we know what your brai/lls **thinking.**

You.. b..ain crav ,ovelty. It's always searching, scanning, '....*ting* for something U'' L'I. II was huih that way, and it helps you stay aliv .

So wh•[d... ... y,ur brain don with all the ,'"lt;"..,•rdinary, Tlc"-Tl"" things you encounter? Everything it *can* to su'l} ,h.." , from interfering ",;th the h,ain" *real* job recording things th.I' ..*atter*. I, d...,,', I..otlo., saving the I..uring things; they oeve. make it pas II,,· ·',hi. is obviously n..l ;'"I"'n''''''' l;l1'-T

How does your brain *know* what's important? Sup!"' s· you're out for a day hik,' ."..,1a tiger jumps in front of you, what h"l'I"''' inside your head and body?

Neurons fire. Emoti,ms crank "I'. *Chemicals surge.*

.\ud that's h. ". vour brain knows...

This must be importantl Don't forget It!

Hur imagine you're at hom.., •r in a library. It's a safe, warm, tig....-free zon... You're studying. Getting readv for an ..xam. Or trying L.. I,-U'll som.. tough tec hnical topic vour 1oss thinks l, ill l..,l .., a week. 1,-.. davs a 11." 1nos1.

J.N'..... 1"'•l m, Yo..,r brain's trying I" .l.. y..., d big I.", ,u_lt-, trying ro make su Ih,,, rhis *obviously* II',,-imp..n.ull content doo-"n-I clutter "I' scarce r ources. Resources that ·re 1..:tter spent storing tl ally *big* things. Like tige Like the danger ..r fire. Like how ,..., sl uld never h,,,e posted tl "party" photos "n yu li1t Facebook page•

.And 110.·,..'s II<' simple way 10 tell your h,-"ill_"Hey brain. 110.","," you v..rv nuah, I"... "" m.'l1.... l..ow .1,,11this book is, ;" ..1how little r m registering •m 110.. Mll'•tional Richter scale right r" ow, I really *do* want you to kee" this stuff around."

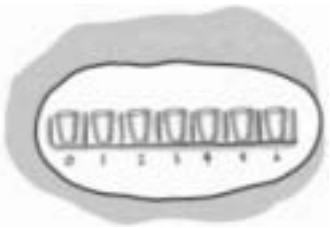




We think ofa "ljead First" reader as a learner.

So what does it take to *learn* something? **First, you** have to *get* it, then make sure **you don't** *forget* it. It's not **about** pushing facts into **your** head. Based on the latest **research** in **cognitive science, neurobiology, and educational psychology,** *learning* takes a lot more than **text** on a **page.** We know what turns your brain on.

Some of the Head **First learning** principle. :



Make **it visual.** Images are fa• more memorable t words alone, and make learning much more effective (up to 89% improv nt in recall and transfer studies). It also makes things more understandable. Put the **words within** O. **near** th. g.aphica lh..y relate to, rather than on the bottom or on another page, and learners will be up to *twice* as likely to solve problems related to the content

Use a cony., **sa**tlonal and pe.sonall"ed style. In recent studies, students performed up 10 40% better on post-learning tests if the content spoke directly to the reader, using a first-person, conversational style rather than taking a formal tone. Tel storie instead of lecturing. Use casual language. Don't take yourself too seriously. Which would *you* pay more attention to: a stimulating dinner party companion, or a lectur..1



Get the lea.ne. to think mo.e deeply. In other words, unles you actively flex your neurons, nothing much happens in your head. A reader has 10 be motivated, engaged, curiou•. and inspired 10 solv.. problems, draw conclusions, and generate new knowledge. And for that, you need challenges, exercises, and thought-provoking question.. and activities that involve both sides of the brain and multiple senses.

Get- and **keep—the** .eade.'. attention. We've all had the "I ,eally want to learn this but Ican't stay awake past page one" experience. Your brain pays att t,on to thing. that are out of the ordinary, interesting, strange, eye-catching, un pected. Learning a new, tough, technical topic doesn't have to be boring. Your brain will learn much more quickly if it's not.





TOUch **their** emotion**s.** We now kno.. that your ability to remember something i, largely dependent on its emotional content. You remember what you care about. You ...m..mber when you *feel* som..thing. No, ...te not talking heart-wrenching stories about a boy and his dog. We're talking emotions like surprise, curiosity, fun. •what the_?", and the feeling of'l Rule!" that comes ..hen you solve a puzzle, learn something everybody else thinks is hattl. or realize you know something that "I'm more technical than thou" Bob from engineering *doesn't.*



Metacogllitioll: thillkillg about thillkillg

II' you really want tu learn, and you want to learn Illon' '1ui"klyand more deeply, pay attention to how you pay all'-11';"". Think about how you think. Learn how you I,'am .



Most of us did not take courses on metacognition or learning theory whe-n we were growing up. \\",. were *expected* t« learn, hut rarelv *taught* I" learn.

lIu, we assume tha, if you're holding lhi, [>CH,k. you H-ally want t" I,'"m how to build programs in C#. And you probably don't want to spend a]", or time. If you want In use what you read inthis book, you need to *remember* what you read. And for th.u. yuii've go« t" *understand* it. 'I" get the m«"t fr«lIn Ihi, boo«ok. """ *any* boo«dk "" learning experi«11'>e, la k,-,,>sponsibility for your l""in. Your 1rain oil *this* content.

The trick is to g«-t your brain to see the new material you'r.. learning as Really Important. Crucialt" your w..ll-being. As il11p"1Ianl as a tiger. Otherwise, you're in for a constant battle, with your brain doing its best 10keep the n""- content from sticking.

So just how DO you get your **brain** to treat C# like it was a hungry tiger?

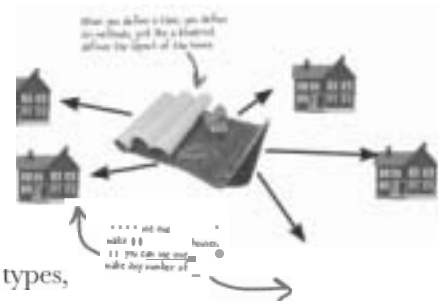
There's Ihe slow, tedious way, o« th,' faster, mol"" effective way. The slow way is about ,h""r repetition. You obviously know thal you *are* able 10learn and remember even the dullest of topics if you k,"'l' pounding the ';1ln,' thing i"I" your brain. With enough repetition, your brain says, "Thi, doesn'1*feel* imp'>rtant to him. hut he keeps I,>oking at the same lhing *over* and *over* and *over*, so [suppose il must be."

The faster wav is to do ***anything 'hat incr..ases brain activity,*** especially different *types* of brain activity. T h,' things on the previous pag« are a big part of Ih,· ,,,Iutiou. and they're all things that have been proven to help your brain work iu vour favor. Foi example, studie show thai]luu ing words *within* the pictures they descril>t. (as opposed to somewhere else in the page, like a caption or ill the body text) causes your hra;n to try 1" rnakes sense "I' h,,w Ihe words ;1Ilrl picture relate, and Ihis causes mon- neurons 10 lire. M1,,,-,. neurons firing = Ill'>re chance f«>r y«,til'l >rain I" *get* that Ihis is something worth paying illt,'nl;""t". and possibly recording.

.\ conversational style helps Because people tend to pay Ill«re attention when they >erceive that they're in it conversation, since they're expected 10follow along and hold lip their end . The amazing thing is, your brain doesn't necessarily *care* lh,1I the "conversation" is betwee11 you and it book! ()u the other hand, if the writing style is formal and dry, your hlili" perceives il Ill" same way you ,'xl" 'l i" lln' being lectured to while sitting ill it roo'1ll1',,1 of passive attendees. No need to stay awake.

But pictures and conversational style are just the beginning.

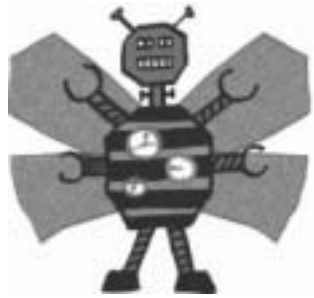
Here's what WE did:



\We used ***pictures***. because your brain i; tlln,»d for ,-i;lIa],. n, ,t text. As far as y("" hr"in's concerned, a 1';'·ll1e really i, worth a thousand words. ,\nd when text and pkturcs work together, we embedded the text *in* the pictures becausc your brain works more effectively when the text i, *within* the thing the te-xt refers 10. as opposed 10 in a caption or buried illll", Iext somewhere.

\We used ***redundancy***, saying the same thing in *different* ways and with different media types, and *multiple senses*, 1◇ increase the chance that the content gets coded iuto mon' than one area or VUU brain.

We uVed concepts and picture ill ***unexpected*** ways because your brain is tuned for novelty, and we used pi,'lUr<·, and ideas with at least *some* ***emotional*** *content*, because your brain is tlln,»d to pay attention to the biochemistry or emotions. '111at which cause you *(0 Jee* something i, more likely to be n-m-mb-red. even if that feeling i, nothing mol"' th.m a little ***humor***, ***surprise***, or ***illterest,***



We ",ed a personalized, ***conversational style***, because yot'r hrain i, 11I""d 10pay mor,' attention when it believe you're ill a conversation than if it thinks you're passively listening to a presentation. Y◇"r brain doe thi, even when you're *reading.*

\\',. included more than 80 ***activities***, because your l,rain is IIm;»dI,,, learn and remember ""»re \Ih"n you ***do*** things than when yc,\I *read* ;„ bc un things And we made Ihe exercises challenging-yc t-do-able, because that's "hat most people prefer.

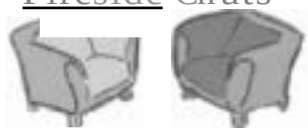
\We used ***multiple learning*** *s ty l**es***. Ibecause *you* might prefer step-by-step procedures, whih- someone else wants t,. under,tanrl the big picture first. and someone else just wants '" see an example. But regardless of your own learning prefere" cc·.*everyone* benefits ti,,,,, seeing the same cc,,,u·,,t represented inllllhil'l,· ways.



BUiLET POINTS

\\',. include content for *bolh sides of your brai"*. because lh,' more "r your hrain y(" engage, the 11"»re likely you are to learn and remember. and the longer you "an stay focused. Since working one side or the hr;,in often means giving th,' other side a chance 10rest, you can be III'»re productivc- at learning f('r a I,»nger period of Ii"ıe.

Fireside Chats



.\",1 we included ***stories*** and exercise ,hal present ***more than one poin1 of .·iew*** because your brain i; tuned to learn more deeply when it's forced to make evaluations and judgments

We included ***challenges***, with exercises. and by asking ***questions*** that d",,'t always have a straight answer, because y our brain is tum·d 10learn „„,,Iremembe \Ihei i, has (" *work* at something. T hink alII,ut il you, can't get your *body* in ;hape just by *watching* pe('11,· at the gym. Hilt we did o"r best 10make sure that when you're working hard, it's on the *right* things. 'I'hat ***y ou're IIot sp""di"g one extra dendrite*** processing a hard-to-understand ex.unple-. or parsing difficult, jargon-laden, ,>r,werly terse text.



\We used ***p eop le***. In stories, examples, pictures, etc., 1..:,-'"' st-. well, because *you're* a pers('", .And your brain pays more a'h'mion t" *people* than it does to *things.*



Here's what YOU can do to bend your brain into subllission

So, we did our part. T lie rest i, up to vou. These til" are i' starting pc,inl' listen to your brain and figure "n what works for you and what doesn't. Try new things.

Cut this out and stick it on r-....efrigerator

Slow down. The more **you** understand, the less **you** have to memorize.

Don't ju,t *read*. Stop and think. Whe" the book asks you a question, don't just ,kip to ,h,' answer. Imagine th"l someone really *is* asking the question. The more deeply you force your brain to thin k. the h..ue-r chance "on have „1' learning and remembering.

Do the exercises. Write your Own notes_

\\" p"l them in, hut if we did them for you that m ,uld h., like having some(,n,' else d(your workouts f(r you. .And don't jU'1 *look* at the exercises. **Use a** p..**ncil.** There's plenty "f evid..nce that physical activity *while* learning can increase the learning.

® **Read the "There are No Dumb Questions"**

That means all of them. They're not optional sidebars— ***they're part of the core content!*** 1),)[,', skip them.

Make this the last thing **you** read before bed. Or at least the last challenging thing.

Part of the learning (especially the transfer 'o long-term memory) happen *after* you put Ih.. book down. Your brain needs ,i",e Oil ii, own. 10 do more processing. If you put i" something new during that processing time, some "l' what you just learned will be lost.

5 Drink w ater. Lots of It.

Your brain works best in a nice· bath 01 fluid. Dehydration (" h id , can happen before you ever feel thirsty) decreases cognitive functi'II,

® Talk about it. Out loud.

Speaking activates a difb-n-nt part 01 the brain If you're trying '0 u"d"r'(a"d something, '" increase your chance of remembering it later, say i, ',ullo"d, B." h'r still, try 10 explain i, ou,f,,,,,1 (o someone else. You'll learn more quickly, and you might uncover ideas you hadn't known were there when you were reading about il.

7 Listen to your brain.

Pay attention to whether voui brain is getting overloaded, Ir yot liml yourself starting (" skim the surface or forge what you just read, it's time for a break. Once you go past a certain poinl, you won't learn faster by trying '0 shove more in, and you might ,·",.u hurl Ill,' process.

® Feel something.

Y" ur brain needs u know that this *matters.* (;,.(involved with the stories. Make up your own cap i,ons for Ih,' ph,tos. (: roaning over a bad j, ,k,' is .<till heiter than feeling nothing at all.

® Write e lot of softw.rel

There's only '>",' wav '" learn t" program: **writing a** 101 of ..**ode**. And that's what you're going to d" throughout this book. Coding is a skill, and the onlv way 10 get good at i, is to practice. We're going 10 give you a I", of practice: every chapter ha, exercises that pose a problem for you (o solve Don't jus skip ,,,er them—a lot of 11,,'learning happens whe" you solve ,|", exercises. \\',. included a solution to each exercise d, ,n'l Il,' afraid '0 p....k at th.. **solution** if you get stuck! (It's easy 10 get snagged on something small.) But try to solve the problem before you loc,k at the ",luti"n..\ml definitely get il working before you move on to the next part or the book.

What you need for this book:

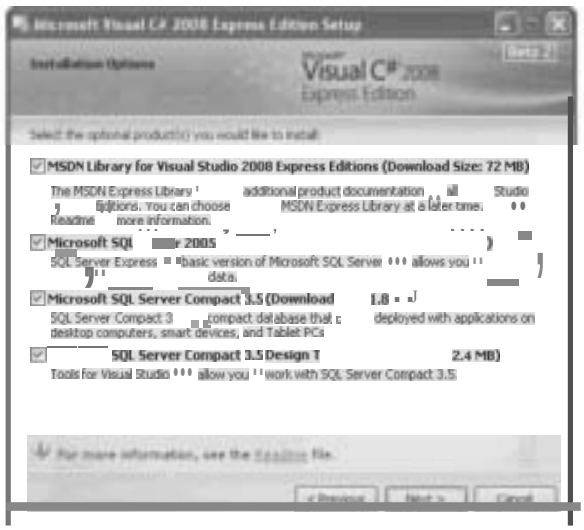
We wrote this book using Visual Studio 2008 Express Edition. That means we used C# 3.0 and the .NET Framework 3.5. The thing is, when we wrote this book, Visual Studio 2008 was still in beta, and it won't be released until after the first printing. But we know that every single exercise and piece of code in the book works 100% with the current beta version of Visual Studio 2008 Express. So we recommend that you download it from Microsoft's website—it's free to download and use.

SIDING UP VISUAL STUDIO 2008 EXPRESS EDITION

- Before you start, keep in mind that if you bought this book in the first few months after it was released, you're probably going to be using a beta version of Visual Studio 2008 Express Edition, so the usual beta version warnings apply. We did that because there have been some really amazing improvements to C# and .NET, and we wanted to make sure that you learned about them. And just so you know, we haven't had a single problem using the VS2008 beta.
- It's easy enough to download and install Visual C# 2008 Express Edition. Here's the link to the Visual Studio 2008 Express Edition download page:

  http://msdn2.microsoft.com/en-us/express/future/bb421473.aspx

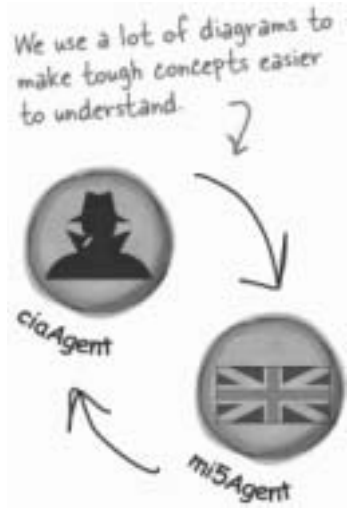
  Make sure that you choose all of the options when you install it.



If you absolutely must use an older version of Visual Studio C# or the .NET Framework, then please keep in mind that you'll come across topics in this book that won't be compatible with your version. The C# team at Microsoft has added some pretty cool features to the language. We'll give you warnings when we talk about any of these topics. But definitely keep in mind that if you're not using the latest version, there will be some code in this book that won't work.

- Download the installation package for Visual C# 2008 Express Edition. Make sure you do a complete installation. That should install everything that you need: the IDE (which you'll learn about), SQL Server Express Edition, and .NET Framework 3.5.
- Once you've got it installed, you'll have a new Start menu option: *Microsoft Visual C# 2008 Express Edition*. Click on it to bring up the IDE, and you're all set.

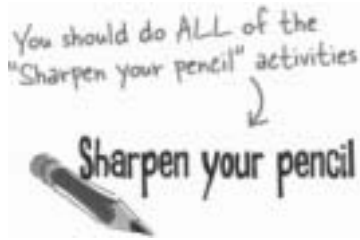
Read I\Ie



Thi, is a learning experience, not a reference book. We deliberately 'tnpped out everything that might get ill the way of learning whatev II is we're working on at that l'0im in Ihe book. And the first time through, you " d 10 begin at the beginning, because the book makes assumptions abour what you've ahready seen and learned.

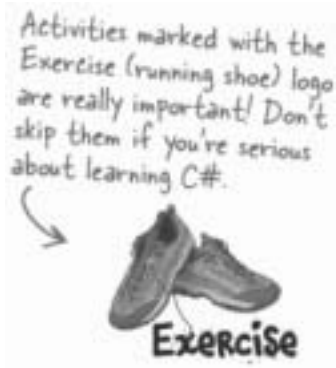
The activities are **NOT** optional.

The exercises and activities are nol add-ons; thev're part of the ('ore content of the book. Some of the", are to help with memory, some for tllld"r'tallding. and some to help you apply what you've learned. ***Don't skip the exercises.*** The crossword puzz]es are Ihe only things you don"t *have* 10 do, but they're good for giving your brain a chance 10 think about the words from a different ront" XI.



The redundancy **is** Intentional and important.

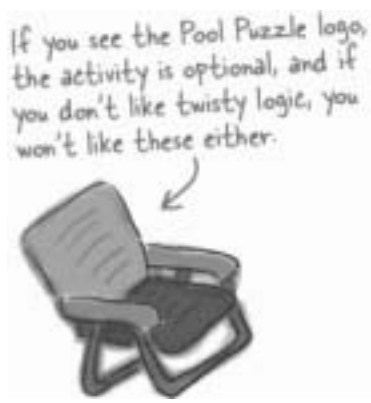
One di,tillct difference in a Head First book i, that we want vou to *really* get it. And we want you to fini,h the book remembering what you've learned. Most reference books don't have retention and recall as a goal, but this book i, aholl t *learning*, so you'll see some of the same concepts come np mOre than once.



Do all the exercisesl

The one big assumption that we made when we wrote thi, book is that you want to learn how to program in C#. So we know you want to get your hands dirty right away; and dig right into the code. We gave you a lot of opportunities to sharpen your , kill, by putting exercise in every chapter. We've labeled some of them "Do this!"—when you see that, it means that we'll walk you through a1\ of the step to solve a particular problem But when you see the Exercise logo with the runnmg shoes then we've left a big [>OItioll of the problem up to you to solve, and ı ve gave you the solution that we came "l' with. DOll't be afraid to peek at the solution—**it's** n,, 1 **cheating**! But you'U learn the most if lull try to solve the problem first

We've also placed all the exercise solutions' source code Oll the web so you can download it. You'll find it at `http://www.headfirstlabs.com/books/hfcsharp/`



The **"Brain** Power**"** exercises don't have answers.

For some of them, there i, no right answer, and for others, part of the learning experience of the Brain Power activities i, for you 10 decide if and when your answers are right. In some of the Brain Power ex..rcise you will find hillts to point you ill the right direction.

The technical review team



Not pictured (but just as awesome): Wayne Bradney, Dave Murdoch, and Bridgette Julie Landers







Technical Reviewers:

\\b .." we wrote this book. it had a bun..h "f "'istakes. issues. problems, typos, and terrible ..rilhn..tic ..rrors. Okay, i, wasn,., 'lui'" that bad. But we're 'lill r...ally grateful for the work that our chnical reviewers did for Ihe I" . ok. W. would have gone r" press with em including one or ,"n big ones had il not been for the "" ..I kick-ass review team EVER...

First .f "ll. I,., really want ,,, thank **Joe Albahari** for the enormous am".1111 ..f technical guidance. He reallv set us straight on a f,w reallv important things, and if i. '"T., ,'I for hi", you'd I", learning incor ·1stuff. \\'" also want '" thank U",. K..lln..r Ihi. is our third book thi 1",', reviewed for us. ."..1 sh.. made : huge differe,..e in the read, hilill, of the 10",.1 "'"h..·1. Thanks, Lisa! ,\..,1special thanks to **Jay** Hilyard and **Dan**i..1Kinnat'r f,r catching and fixing ,. whole 101"f . our mistakes, ;",.1**Aayam Singh** for actually going through and doing every ,,"'. "r these ,-xe,ises h..**fore** ,,,' fixe.1them and corrected their proble...s Aayam, you're really dedicated. Thanks!

Acknowledgments

*Our **editor:***

We want to thank our editor, Brett McLaughlin, for editing thi, book. He helped with a I"l "l' the narrative, and the Ulllli,- idea in Chapter 14 was ." „„„J·I.,ly hi" and we think it turned oW ,-,-ally well. Thanks, /lrett!



The O'Reilly team:



Lou Barr i, all amazing graphic designer who 'vell above and beyond Ul) thi, one, putting in unbelievable hours and coming "I' with some pretty amazing visuals. Ir yo>l see anything in Ihis boo,k lhat [,)oks fantasti,-. you can thank her (a",1her mad InDesign skillz) for it. She did all "I' lh" monster and alien graphics for the labs and the entire comic book. Thanks so Illud" Lou! You are 0 " 1' hero, an,1 you're awesome I" work with.



There an' so many people at O'Reilly we want to thank ,h,u we h"I'" we don't forge anyone. Firs "f all. the Head Firs team rocks—**Lauri**.. **Petrycki**, **Catherin**.. Nolan. Sand..**rs** K1e**infeld** (the mos super p"ɔchK,ioll editor ever!), Caitrin **McCullough**, and K..ith **McNamara**. Sp",ial thanks to ColI....n Gorn, an for 10,',. sharp 1'"ɔofread, Ron Bilod..**au** f(,r volunteering his time and preflighting expertise and Adam. Witw..r f(,,. offering one las sanity check- all "f whom helped get this book from production I" pres ill record time. And as alwavs, we I"v(· Mary T**res**..l..r. and tan'l wai, '" work with her again! And a big shout "u, h, " n, other friends and editors, Andy Oran, . **Isab**..1 Kunkle. and **Mik**.. H ..**ndrickson**. .\1\d ir ' 'ot1·1.· reading Ihi, book right n"w. then you ,-"n lh,Hlk the greatest publicitv team in the industry: **Marsee H**..non, Sara **P**eyton. **Mary** RotInan. **J.ssica Boyd**, and Ill,' rest " l' the folks at Sebastopol.

1 get productive with C#

Visual Applications, in 10 minutes or less



Want to build great programs really fast?

With C#, you've got a powerful programming language and a valuable tool at your fingertips. With the Visual Studio IDE, you'll never have to spend hours writing obscure code to get a button working again. Even better, you'll be able to **focus** on getting your work done, rather than remembering which method parameter was for the *name* for a button, and which one was for its *label*. Sound appealing? Turn the page, and let's get programming.

Why you should learn C#

C# and the .NET Framework make it easy for you to get to the business of writing code, and writing it fast. When you're working with C#, the IDE is your best friend and constant companion.

The IDE—or Visual Studio Integrated Development Environment—is an important part of working in C#. It's a program that helps you edit your code, manage your files, and publish your projects.

Here's what the IDE automates for you:

Every time you want to get started writing a program, or just to add a button to a form, your program needs a whole bunch of repetitive code.

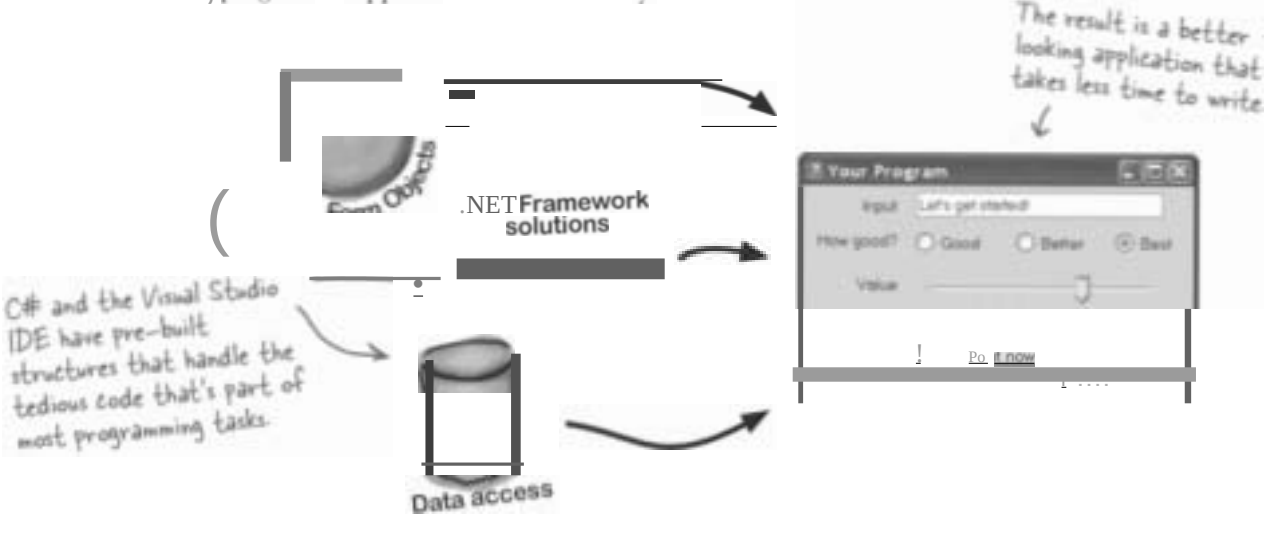
```
using System.Collections.Generic;
using System.Windows.Forms;
namespace A_New_Program
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

```
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(105, 56);
    this.button1.Name = "button1";
    this.button1.TabIndex = 0;
    this.button1.Text = "button1";
    this.button1.UseVisualStyleBackColor = true;
    this.button1.Click += new System.EventHandler(this.button1_Click);
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(292, 267);
    this.Controls.Add(this.button1);
    this.Name = "Form1";
    this.Text = "Form1";
    this.ResumeLayout(false);
}
```

It takes all this code just to draw a button on a form. Adding a few more visual elements to the form could take ten times as much code.

What you get with Visual Studio and C#...

With a language like C#, tuned for Windows programming, and the Visual Studio IDE, you can focus on what your program is supposed to do instead of the tedious code that's part of most programming tasks.



C# and the Visual Studio IDE have pre-built structures that handle the tedious code that's part of most programming tasks.

The result is a better looking application that takes less time to write.

C# and the Visual Studio IDE llIake lots of things easy

When you use C# and Yisual Studio, you gc1all of these great features, without having 10 do any extra work. Together, they let you:

1. Build a n application, FAST. Creating programs ill C# is a snap -n,e languag(is powertul and easy 1" learn, and the \ 'i, ,,a l Studio 11)E does a 1"1 of work for you automaticallv. You can leave mundane coding tasks 10 the Ill1E and focus on what y('Ur code ,],ould acc('""1'li,h.

2. Design a great looking user Inte rlace. The Form Designer in the Visual Studio IDE i, ou,' of the "a,i",t design 1,",1, 1" use out there. It doe s(much f(,r you Ih"t y('11'11 find that making stunning use-r interfaces i, "11" "f th(111'"1 satisfying parts of developing a C# "PI'Ii, -"ti"n. You can build full-featured professional programs without having ı" 'p""d hot"" writing a graphical user interface entirely from scratch.

3. Create and Interact with databases. The IDE includes a simple interface for building databases, and integrates seamlessly with SQL Server Express, as well as several other popular databas(systems.

4. Focus on solVing your REAL problems. The IDE does a lot foı you, but ***you*** are still in control "f wluu you build with C#, The Ill Ej"' 1lets \'"∪ focus »u y('Ur pl<ogram, y('Ur work (or fun'), and your customers. But the IIIE handles all the grunt work, such as:

   - Keeping track of all of your projects
   - Making il easv to edlr your project's code
   - Keeping track "f your project's graphics audio, icons, and olh er resources
   - Managing and interacting with databases

.\H Ihi, means you'll have all the time you would've spent doing thi' routine programming to I'UI into **building killer programs**



Help the CEO go paperless

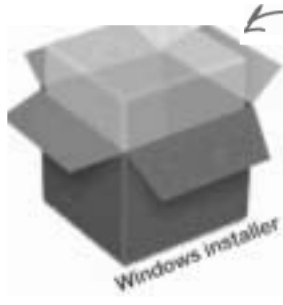
The <Objectville Paper Company just hired a new (:Ef). H,' loves hiking, "..lli·,·. and nature... and he's decided Ih,1 to help s:" e fores •. H,· wants ,.. become a paperless executive, starting " ilh his contacts. II,", heading to Aspen '" go ski for ,I", weeker..!. and exp ts ;, ",." address book program by ll.., time l..· gets bac k. Otherwi " ,11 ... i, won't b.. just the old (:E< I who,',]..oking for a job.



Get to know your users' needs before you start building your program

Before Ive can start writing the ;,tldress book application—or *any* application we need In ,;,k,-" minute am[think about **who's going** 10 b.. **using it**, and whal th..y n....d from the appli";lIi"n,

① The CEO needs 10 be able 10 lUll hi, address 1>oc,k program a, work and ull hi, laptop 1,»c,. He'll need all installer '" Ill.,k,·sure lh,u all of the right [il,-, get "UI" eac'h mac'hi", ",

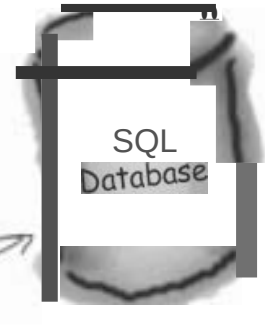


The CEO wants to be able to run his progra.. on his desktop and laptop, so an installer i. a must.

② The Objectville Paper company sales team wants 10 acces his address book, too. They r-an use his data It> h"ild mailing lists and get client leads for Ill"r" pape sales

The CEO figures a database would 1>e- the l>e-" way ,h.11 everyone in the company I" see his data, and then he can j'hl keep up with "1l" copy "r all hi, contacts.

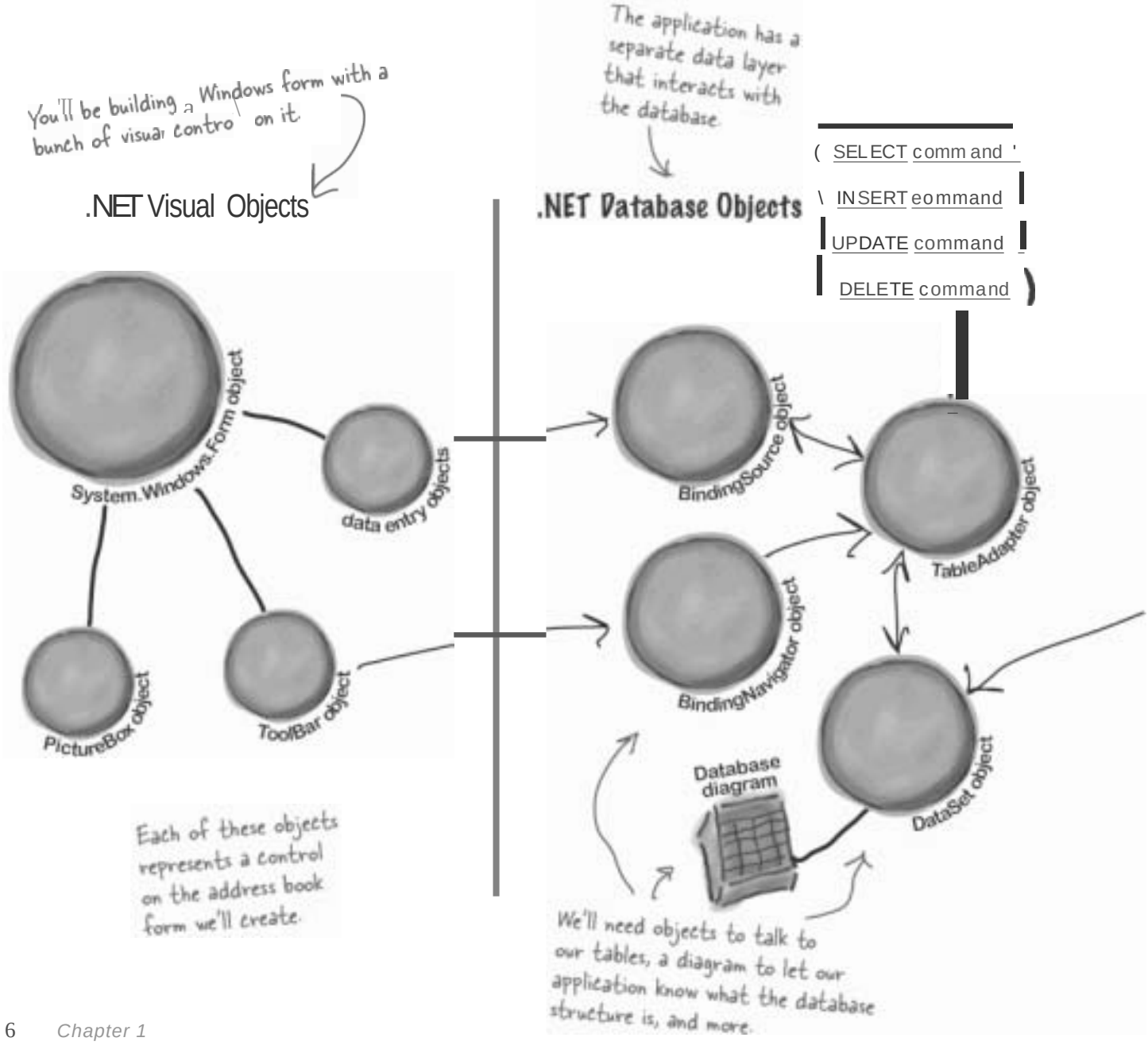
We already know that Visual C# makes working with databases easy. Having contacts in a database lets the CEO and the sales team all access the information, even though there's only one copy of the data.



Here's what you're goil1g to build

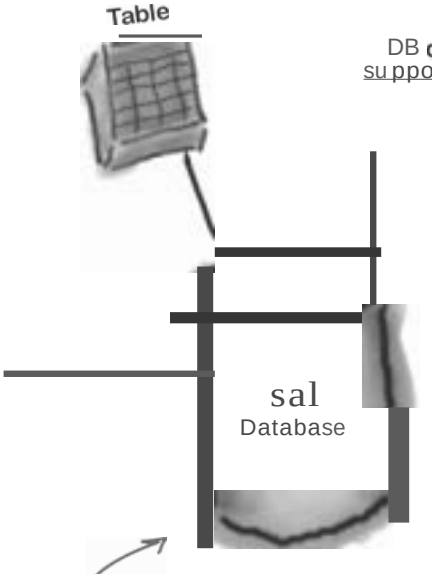
You're going I" need an application with a graphical user interface, objects 10talk to a database, the database itself, and an installer. II sounds likea I,,(..r work, but you'll build all of Ihis over Ih,' next few pages.

Here's th,- structure of Ih,- program we're going 10create:



The data is all stored in a table in a SQL Server Express database.

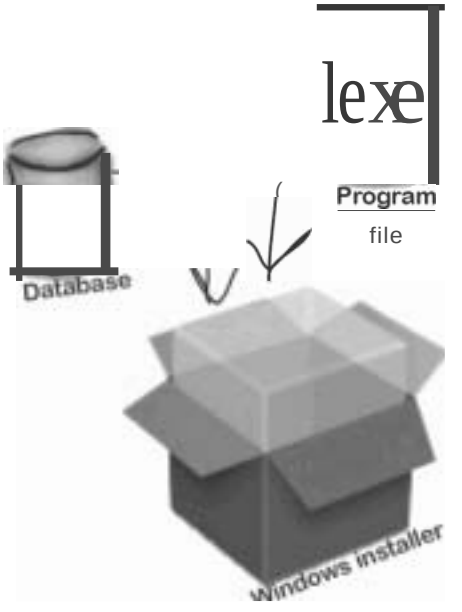
Data Storag,



Here's the database itself, which Visual Studio will help us create and maintain.

O..ce the program' built, it'll be packaged "\' into a Windows installer.

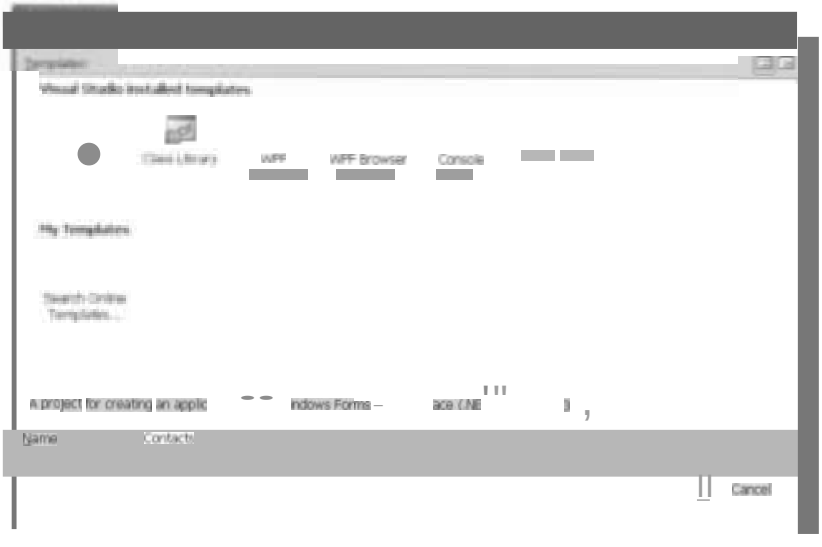
Deployment Packag,



The sales department will just need to point and click to install and the.. use his program

What you do in Visual Studio...

Go ahead and start Visual Studio, if you haven't already. Skip over the start page and select New Project from the **File** menu. Name your project "Contacts" and click OK.





Watch it!

Things may look a bit different in your IDE.

This is what the "New Project" window looks like in Visual Studio 2008 Express Edition. If you're using the Professional or Team Foundation edition, it might be a bit different. But don't worry, everything still works exactly the same.

What Visual Studio does for you...

As soon as you save the project, the IDE creates a Form1.cs, Form1.Designer.cs, and Program.cs file when you create a new project. It adds them to the Solution Explorer window and by default, puts those files in `My Documents\Visual Studio 2008\Projects\Contacts\`.

Make sure that you save your project as soon as you create it by selecting "Save All" from the File menu—that'll save all of the project files out to the folder. If you select "Save", it just saves the one you're working on.

This file contains the C# code that defines the behavior of the form.

This has the code that starts up the program and

The code that defines the form and its objects lives here.



Form1.cs

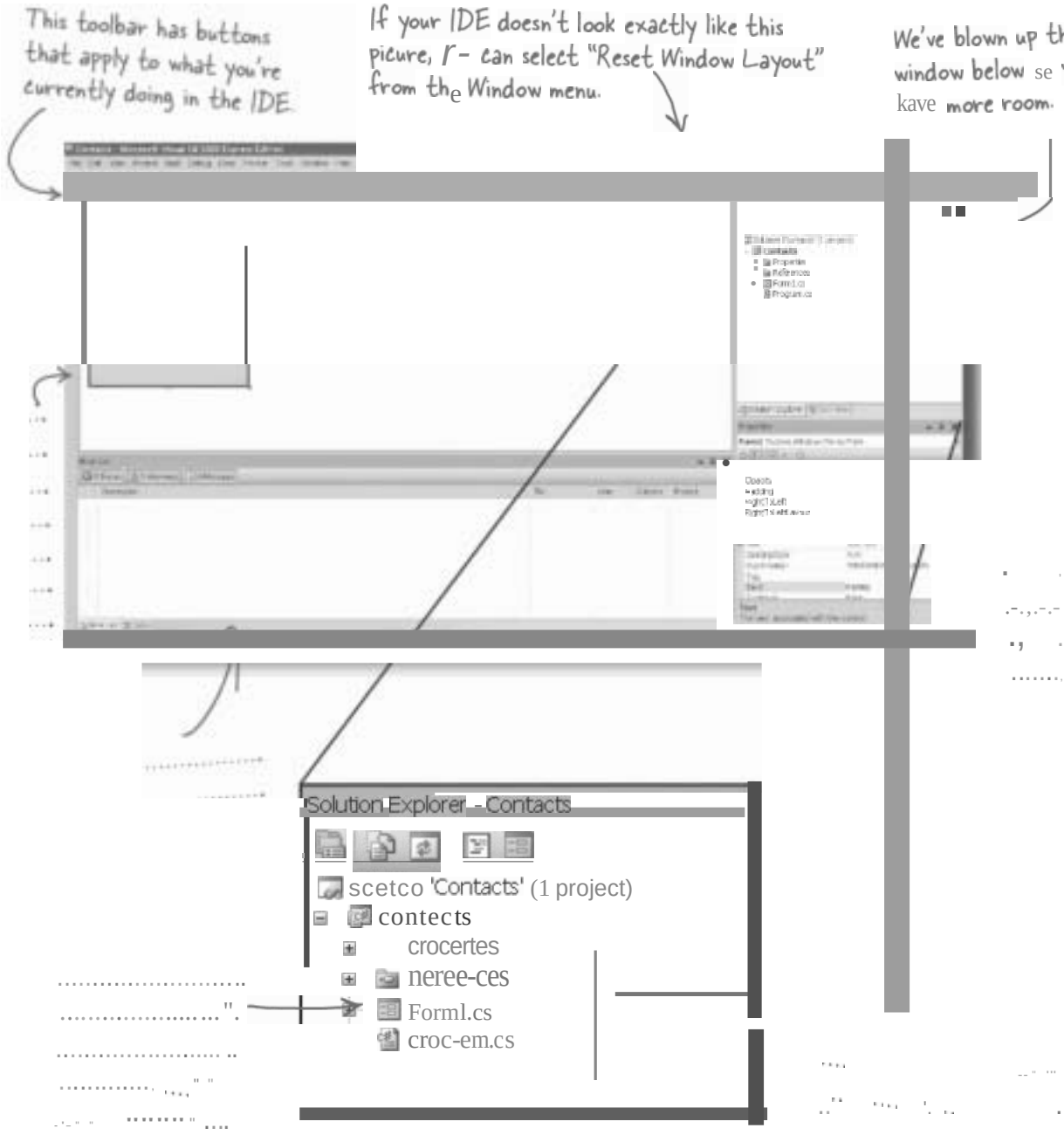
Program.cs

Form1.Designer.cs

Visual Studio creates all three of these files automatically

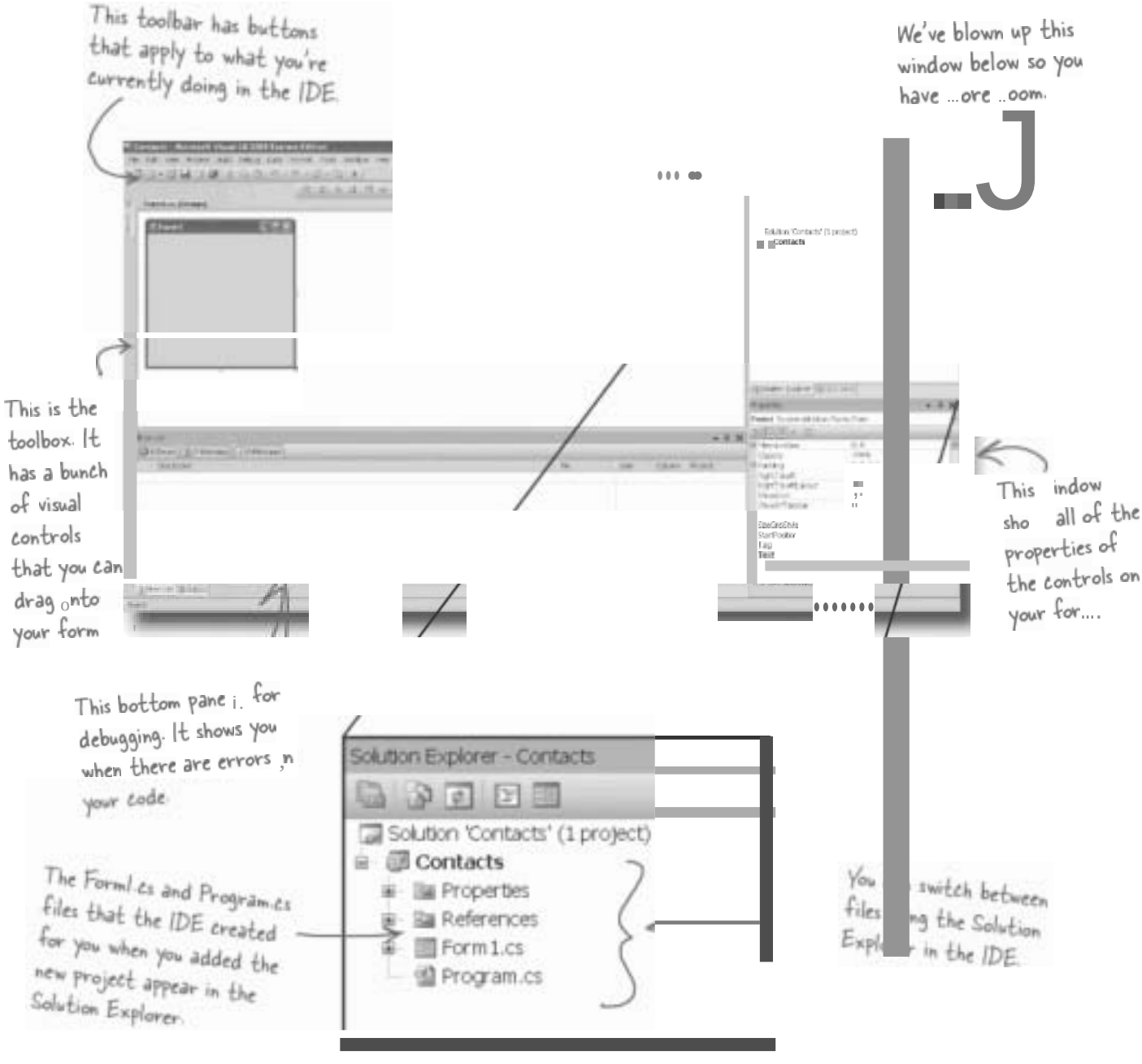
Sharpen your pencil

Below is what your screen probably looks like right now. You should be able to figure out what most of these windows and files are based on what you already know. In each of the blanks, try and fill in an annotation saying what that part of the IDE does. We've done one to get you started.



Sharpen your pencil Solution

We've filled in the annotations about the different sections of the Visual Studio C# IDE. You may have some different things written down, but you should have been able to figure out the basics of what each window and section of the IDE is used for.



there are no Dumb Questions

Q: So if tha IDE writes all this coda for me. is learning C# just a maner of learning how to use the IDE?

A: No. The IDE is great at automatically generating some code for you, but it can only do so much. There are some things it's really good at, like setting up good starting points for you, and automatically changing properties of controls on your forms But the hard part of programming—figuring what your program needs to do and making it do it—is something that no IDE can do for you Even though the Visue' Studio IDE is ooe ol the most advanced development environments out there, it can only go so far It's *you*—not the IDE.-who writes the actian coda. or the code that does the work

Q: I created a new prajact in Visual Studio, but whan I want mtc tha "Prejects" folder under My Documents, I didn't see it thera. What gives?

A: First of all, you must be using Visual Studio 2008-in 2005, this ccesnt happen When you first create a new project in Visual StudIO 2008, the IDE creates the project in your `Local Settings\Application Data\Temporary Projects` folder When you seve the project for the fllst rene. it Will prompt you for a new filename, and save It In the `My Doeuments\Vi"ma! Studio 2008\Projects` lolder, If you try 10 open a new project or close the temporary one, you'll be prompted to either save or discard the temporary project

Q: What if thalOE creates code I don't want in my project?

A: You can change it. The IDE is set up to create code based on the way the element you dragged or added is most commonly used. But sometimes that's not exactly what you wanted. Everything the IDE eces for you—every tine of code il creates, every file it eods-ccan be changed, either manually by editing the fites direct1y or through an easy-to-use interface in the IDE

Q: Is it OK that I downloaded and installed Visual Studio Express? Or do I need to use one of the versions of Visll sl Studio that isn't free in order to do everything in this book?

A: there's nothing in thiSbook that you can't do with the nee version of Visual Studio (which you can download from Microsoft's website). The main differences between Express and the other editions (Professional and Team FoundatlOnl aren't going to get in the way of writing C# and creating fully functional, compete applicafions

Q: Can I change the names of files tha IDE generates for me?

A: Absolutely, you can change any aspect of your program But the IOE is set up to name your files sensibly When you add a file to your project, the nererr e you cnccse enects the way the code is generated, and the code it generates will include that name In some cases, if you rename the file you'll either teve to go through and change other parts of the code, or live With the difference between the filename and the code Inside it SC Ce that's a bit of a pain, we recommend you don't change filenames uness you've got a really good reason to

Q: I'm looking at the IDE right now, bul my screen doesn't look like yours! It's mining some of ths windows, and others are in the wrong place. What gives?

A: If you click on the "Reset Window Layout" command under the "Window menu, the IDE will restore the default window layout for you Then your screen 1'1111 lock just like the ones in this chapter

Visual Studio will generate code you can use as a starting point for your applications.

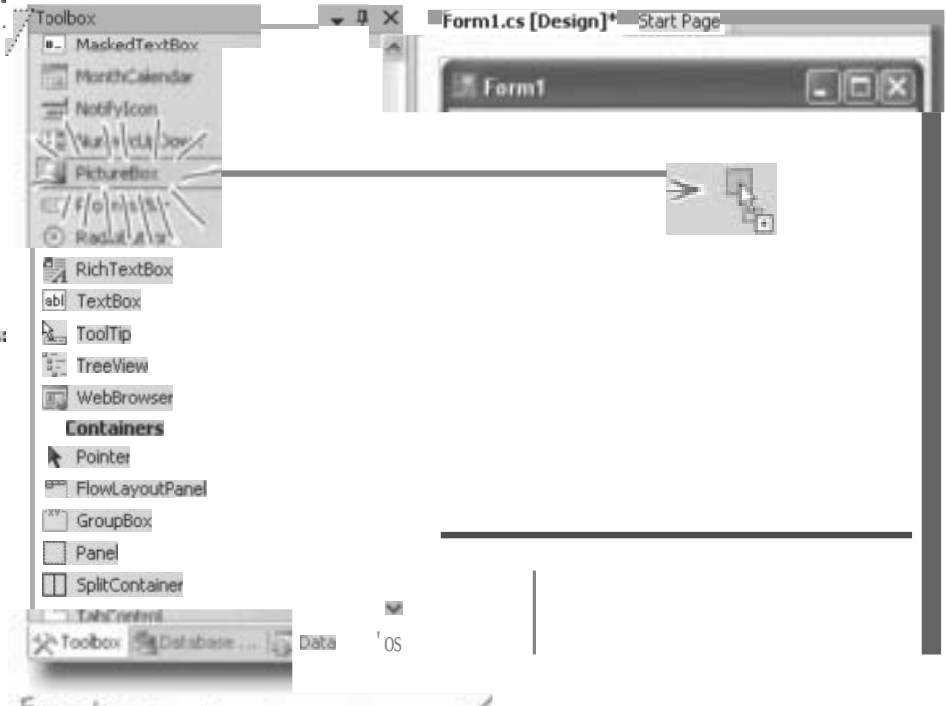
Making sure the application does what it's supposed to do IS still up to you.

Develop the user il1terface

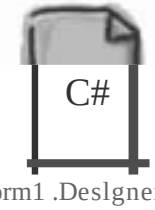
Adding controls and polishing the """1' interface is as easy as dragging and dropping with 1h,' Visual Studio IDE. Let's add a logo ı" the ı",-,n:

O Use the erctureaox control to add a picture.
Click on the PictureBox control in the Toolbox, and drag il ,>nl,)y<,U' f6)ı'111. In the background, the II)E added code to Formt . Designer . cs for a new picture control.

If you don't see the toolbox, try hovering over the word "Toolbox" that shows up in the upper left-hand corner 01the IDE. If Ir s notmere. select "Toolbox" from the View menu to make it appear.



Every time you make a change to a control's properties on the form, the code in Form1. Designer.cs is getting changed by the IDE.

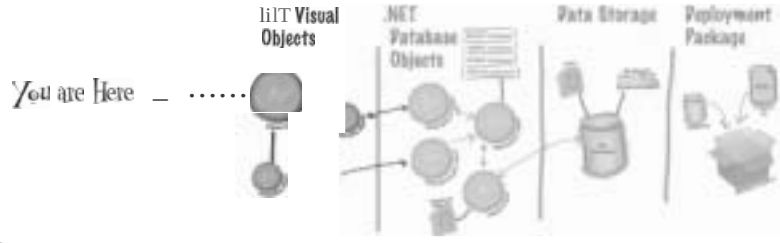


Form1 .Deslgner.cs



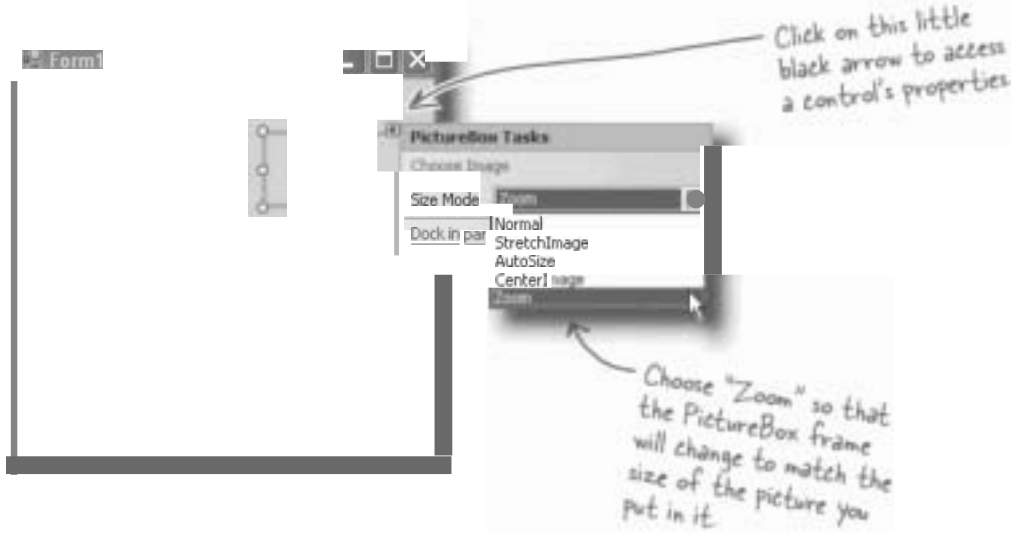
It's OK if you're not a pro at user interface design.

We'll talk a lot more about designing good user interfaces later on. For now, just get the logo ,,,,,I other "" llln ,l, on your form, and worry about *behavior,* We'll add some style ["1'-1'.



O Set the ptcturegcx to Zoo m mode.

Everv control on your form has properties that you can set. Cli,'k the little black arrow for a control to access these properties. Change· llw PictureBox's Size property to "Zoe,,,," to see how this works:



❸ Download the ObJectvUle Paper Company logo.

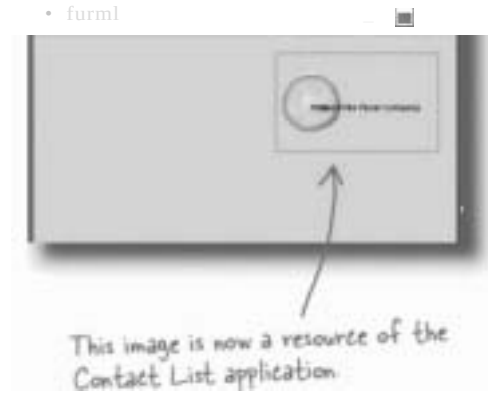
Download the Objectville 1"l"≯ Co. logo from Head First Labs (**http://www.headfirstlabs.com/books/hfcsharp**) and save il 10 your hard (hin·. Then click the PictureBox propertie arrow, and select Choose Image. Cli,'k Import, find your logo, and you're all set:

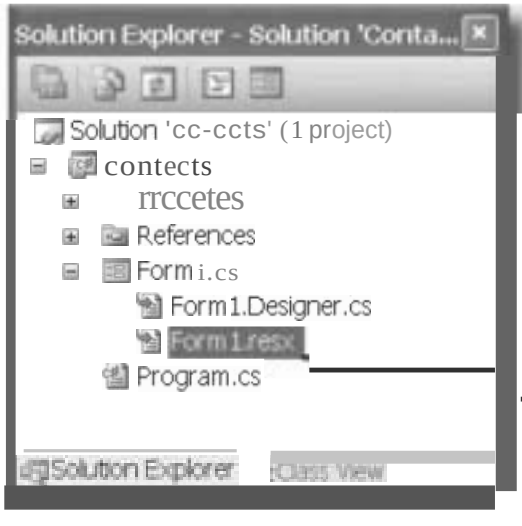


Visual Studio. benilld the seenes

Every time you do something in the Visual Studio IDE, the IDE is ***writing code for yo...*** When you created the logo and told Visual Sill'li,. 10 use the image yc.u d,wnloaded, \'isual Sludio created a resc'U1n' and associated it with)"ur application. A r..**sourc**.. i, anv graphics li i". ;lll1li" file, i" lIn. or otlll'1' kind of data file that gets bundled with your application. The graphic file gets integrated illl" the program, so that when it's installed oil anorlu-rrumpure-r; th,' graphic i, installed along with it and lht" PictureBox can use il.

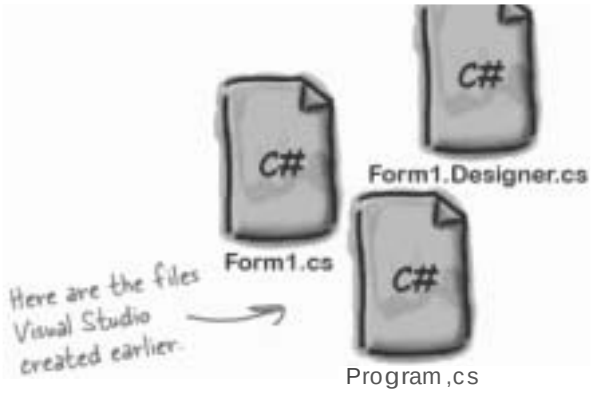
\When you dragged the PictureBox control olllu your fi,rill, II", II)E aUl,matically created a resource til,' called Ii,nn I.n'sx 10store that resourc and keep it in the project. Double-click oil Ihi, file. and you'll be able 10"..' the m-wlv imported image.



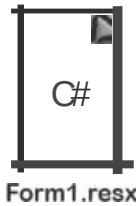


If you click on Form1.resx in the Solution Explorer, you can see the logo that you imported. That file is what links it to the PictureBox, and the IDE added code to do the linking.





r



When you imported the image, the IDE created this file for you. It contains all of the resources (graphics, video, audio and other stored data) associated with Form1.

Add to the auto-gel1erated code

The IDE creates lots or code for you, hur you'll ,Iill want to get inlo Ihi, coc1,-and add I" it. Let's set the logo ul' I" show an About message when the users double-click on it

Make sure you've got your I;,r", showing i" Ih" Il)I-:, and double-click "IIIIne PictureBox control. Yo" should see some code 1'"1' "I' that looks like this:




```
pUblic partial class Forml : Form
{
    publ Lic Form1()
    (
        InitializeComponent();

    private void p1ctllreBoxl_Click(object sender, EventArgs e)
    {
        Show("Contac  i st 1. .\nWrit en b : Your Name", "About");
    }
(
```

When you double-clicked on the Pict.reBox control, the IDE created thi. ..ethod It ill run every ti..e a ..ser clicks on the logo in the -e application

Thi····ethod name gives you a good idea about when it r..ns: when someone clicks on this PictureBox control.

When you double-click on the PictureBox, it will open this code .p with a c..rsor blinking right here Ignore any windows the IDE pops up as you type; it's trying to help you but we don't need that right now.

Type in this line of code. It causes a message box to popup with the text you provide. The box will be titled "About".

Once you've typed in the line of code, save it using the Save icon on the IDE toolbar or by selecting "Save" from the File menu. Get in the habit of doing "Save All" regularly!

there are no Dumb Questions

Q: **What's a** method?

A: A method is just a *named block* of *code* Well talk a lot more about methods In Chapter 2.

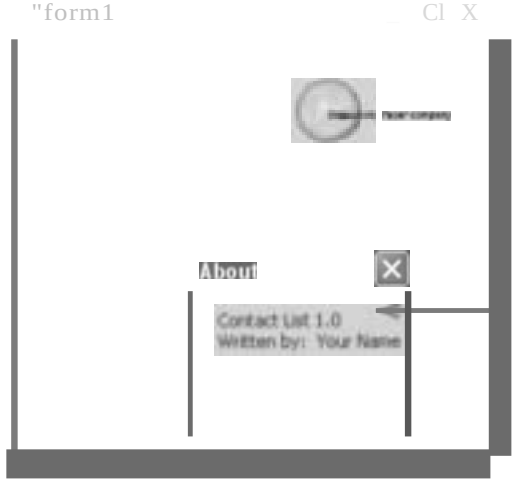
0_: **What does that** In thing do?

A: That's a line break. It tells C# to put "Coo tact List 1,0," 00 oce line, and then start a new line for "Written by:".

You call **already** rUll your applicatioll

Press th.. F5 kev on your keyboard, or click the green arrow h..tton on the toolbar ... , heck out what you've done s f.1f. This is called "Debugging", whic h just ,n...U,_ running your program ..sing the· IIIL) Y◇u •an stop debugging ,,, ...lecting "Stop Debugging" bum .ne Debug men" or clicking this toolbar button:

All three of these buttons work—and you didn't have to write any code to make them work.

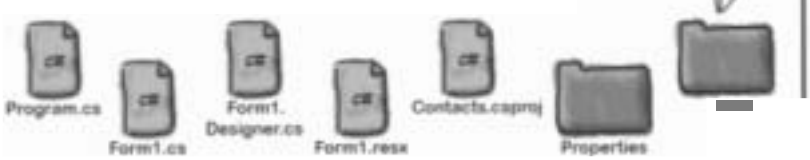


Clicking on the ope logo brings up the About box you just coded

Where are **my files?**

Wl u you rull your program, \ -" ,1..1Studio copies ••ll of your files til My Documents\Visua1 Studio 2008\Projects\Contacts\ Contacts\bin\debug. y,ot' can even hop ' er 1◇ that direc tory .....1run your progr..m by 0lou1 _ clicking ..n the .exe file the IIIE creates.

C# turns your program into a file that you can run, called an **executable**. You'll find it in here, in the debug folder.



This isn't a mistake; there are two levels of folders. The inner folder has the actual C# code files.

there are no Dumb Questions

Q: In Illy IDE, **the green arrow is marked as "Debug". Is that a problem?**

A: No. Debugging, at least for our purposes right now, just means running your application inside the IDE. We'll talk a lot more about debugging later, but for now, you can simply think about it as a way to run your program.

Q: I **don't see the** SlOp **Debugging** butlon on Illy **toolbar. What gives?**

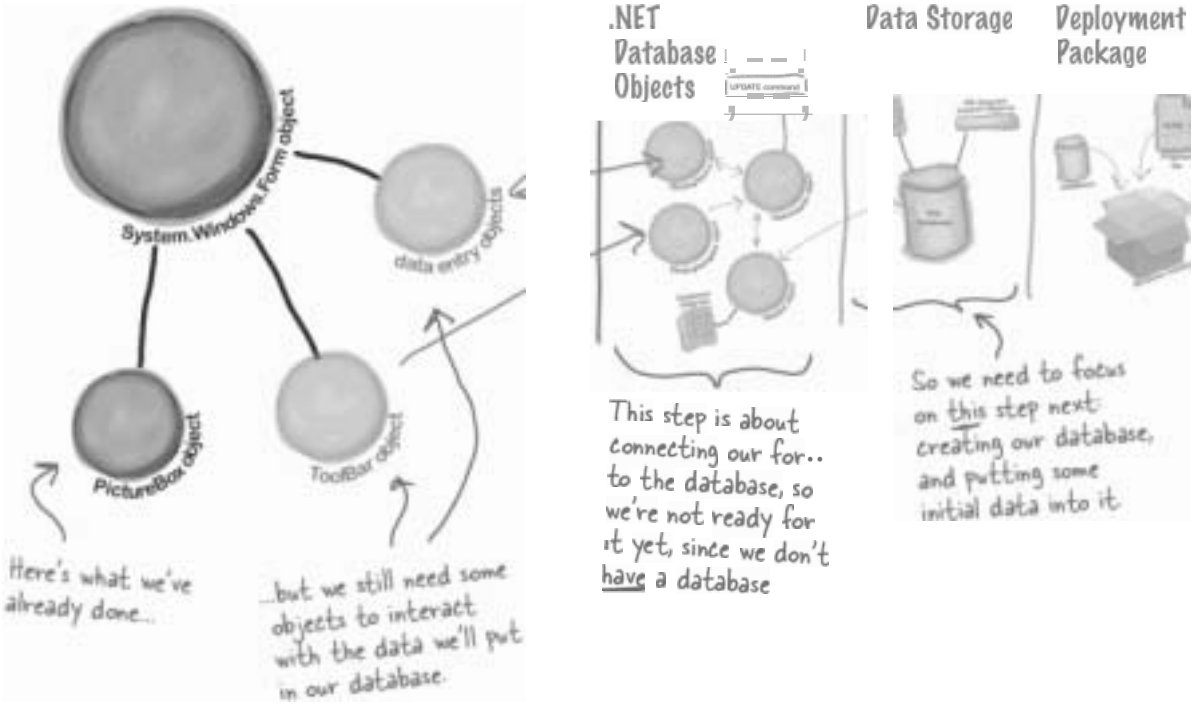
A: The Slop Debugging button only shows up in a special toolbar that **only** . how. up when your program is running. Try starting the application again, and see if it appears.

Here's what we've done so far

We've built a form and created a PictureBox object that pops up a message box when it's done. Next, we need to add all the other fields from the card, like the contact's name and phone number.

But, we'll store it all in a database. Visual Studio can connect fields directly to that database for us, which means we don't have to write with lots of database access code (which is good). But for that to work, we need to create our database so that the controls on the form can hook up to it. So we're going to jump from the .NET Visual Objects straight to the Data Storage section.

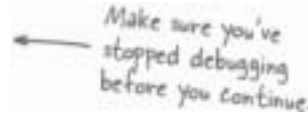




Visual Studio can generate code to connect your **form** to a **database, but** you need to have the **database** in place BEFORE generating that code.

We l1eed a database to store our il1forl\latiol1

Before we add lhe rest of the fields 10 the form. we need 10 create a database I" hook the form up 10. The ll lE can create lots "f tlie code I,,, "connecting our r"rm 10 our data, l,u I lve "",'<l I" define l]", database itself first.

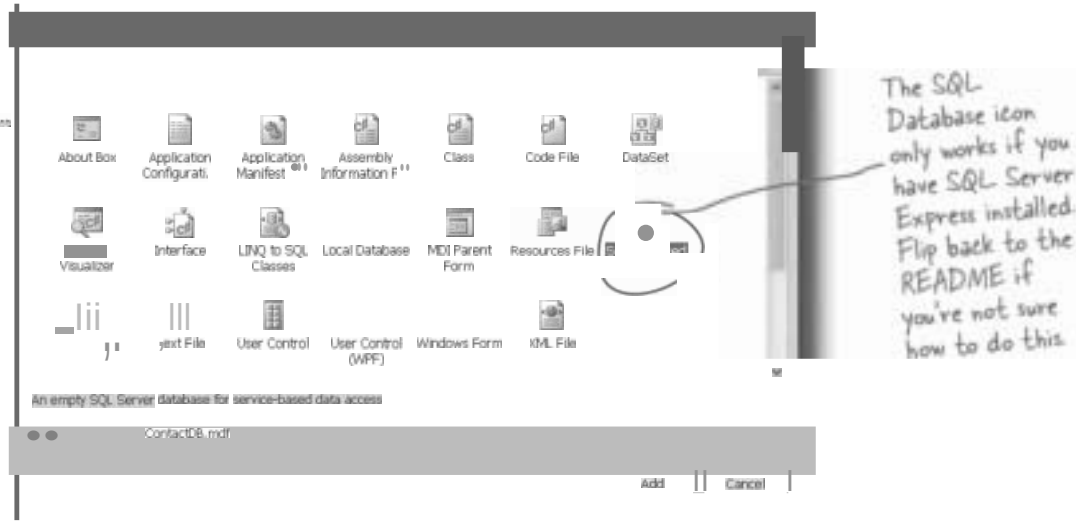


O Add a new Sa l database to your project.
I" the S"hlli,," Explorer, right_dick th.. **Contacts project**, select Add, and then choose New 1t"lll, Choose Ill,·SQL Database iron, and name- it ContactDB.mdf.



ContactDB.mdf

Pick the right icon for the version you're using Choose SOL Database If you're using Visuat Studio Express 2005 and service-Based Database if you're using 2008 .



O Cancel the Data Source Configuration Wizard.
For now, we want to skip configuring a data source, so click the Cancel button. We'll come back 10 Ihi, once we've set up c'UI' database structure.



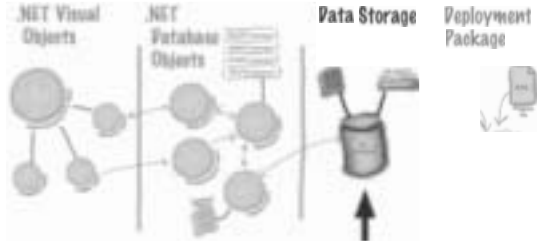
Watch it!

If **you're** not usln9 the Express edition, you'll see "Server Explorer" instead of "Database Explorer".

*The Visual Studio 2008 Professional and Team Foundation editions don't have a Database **Explorer** window. Instead, they have a **Server Explorer** window. which does everything the Database **Explorer** does. but also lets you explore **data** on your **network**.*

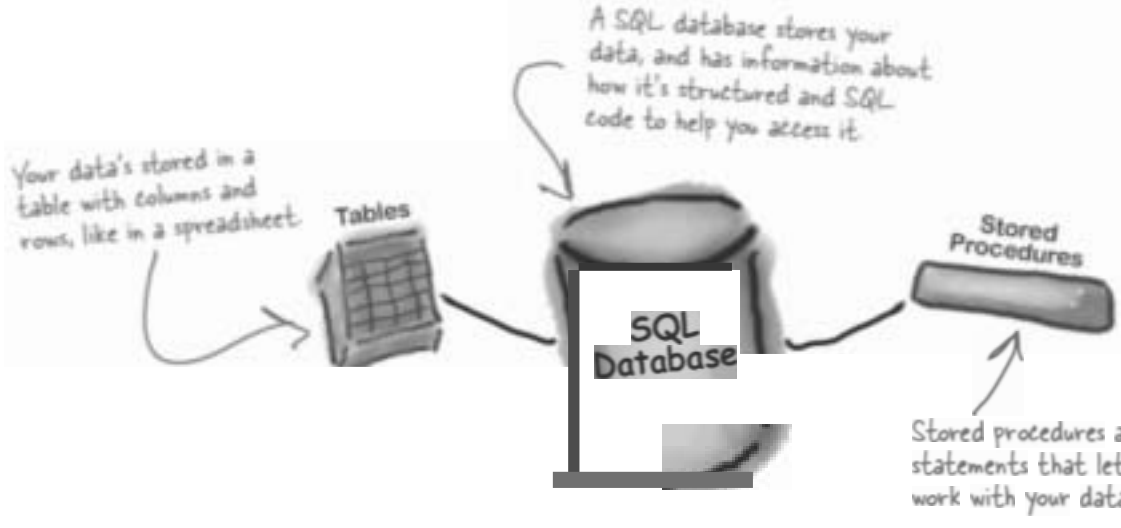
O View your database in the Solution Explorer.
(Jo 1" the Sollllion Ex!'lo,-er, and you'll see that ContactDB has been added I,) the file list. Double click ContactDB.mdf in the Solution Explorer and look at II,,· left side or your screen. T lie Toolbox has changed I" a Database Explorer.

Ihe IDE created a database



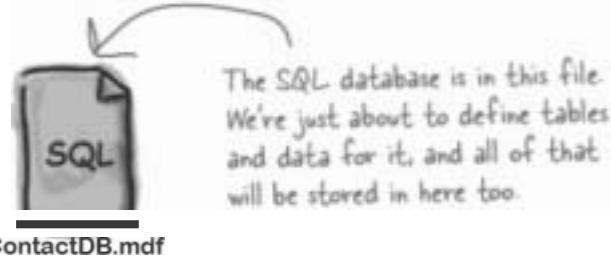
When you .''Id tlte IDE to add a new SQL database ,,' vour 1(1 oject lite IDE created ., '', database fo·r y("1_.\ **SQL** d**a**la**base** is .• system .li.u sto ss ,1..1' fi... you ill ;1[' organized, interrelated way. The 'ill: gives you ;,II th,· t"ols you need to m.•intain your ,l"" ,'',,1databases.

Data in a SQL database liv in tables. For ,now, you can think "f a t"Ible like a sp lsheet. II organizes y..ur infor n tion into ,-,>l,lll" d rows. TI,,· columns are tltr data , egories, like a cont..· name and phone numl...1· and 'h row is llor data fi... one c "" ,,.1 card.



SQL is its owl1 lal1guage

SQL stands for **Structure**d **Qu..ry Languag..** It's a programming language 1,,1' ;1(cessing data in d:..abases. It's got its ,,'' n syntax, keywor Is, i'',,1 structure_SQL ,',ode takes the for'', Or',.lal.. m .." I'' and **queries**, which access and retrieve the data. A SQL database c:,,, 1..,101 "Ior..d pr**oc**..d**u**r..... which are a bunch "r SQL statements and querie that you are stored in the d.•tabas· and ";III I,,· rllll ,,1 any time, The 11 11; generate SQL statements and stored I'r,,,;·,h,,-, for you ;"'lI"IL,lically iuh-t vour program access the d.,la in the databas·,

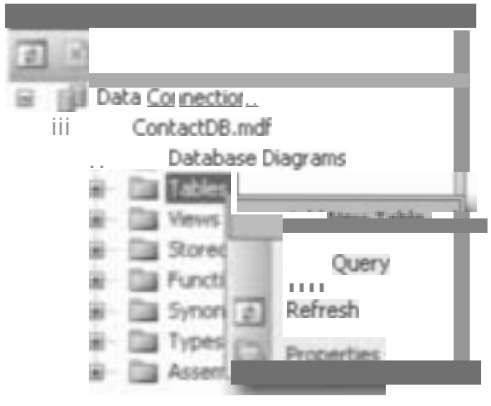


[note from marketing Can we get a plug for 'Head First SQL' in here?]

Creating the table for the Contact list

We have a database, ;",,1now ''' 'H',:cl,,, store information in it. But "Ur information actually has to go into a table, the data structure ,h,ıt ,Ll" Isases use to hold individual bit "I' data. I",r our applicatie'n. let' create a table called "People" '" store "II ,h.. contact information:

o Add a Lable to the ConLaetOB daLabase.
Right click on Tables in the Database Explore'r• and select Add New Table. This will •pen Ul' a window,- where you can define the coluIl"', in the table you just ,reated.



Now we need III add columns to our table. First, let's add a c••lumn called ContactID t•• " m new People table, so that each Contact rec-"rd has i" "" II unique ID.

o Add a ContaetlO column to the People table.
Type "ContactID" in the C"IUllll Name field, ,'n,1 select Ilil from the)),H, Type dropdown bc-x. Be sure In ,"', ·h,ıck the •Allow Nulls checkbox.

Finally, let's make ,hi, the primary key ..r our table. Highlight the Cont,actID co,l"""", you j",t created, and ,-li,l the Primary Key button. This tells the database that ,-," 'h ,-,ıtry " ill h,I\" a "IIi'I"'- primary key entry.



This is the Primary Key button. A primary key helps your database look up records quickly.

there are no Dumb Questions

Q_: What's a column again?

A: A column is one field of a table So in a People table, you might have a FirstName and LastName column - - - always have a data type too, like String or Date or Bool

Q: Why do we need this ContactID column

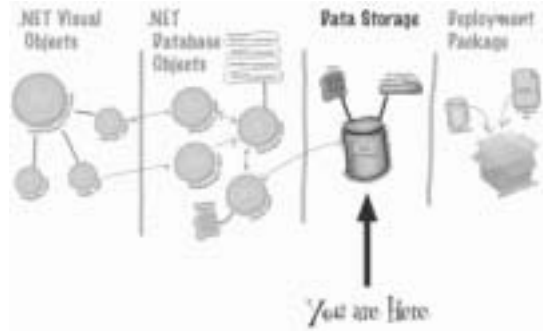
A: helps to have il unique ID for each record in most database tables Since we're storing contact information for individual people, we decided to create a column for that, and call it ContactID

Q_: What's that Int from OIU Type mean?

A: The data type tells the database what type of information to expect for • column Int stands for integer, which is just I whole number. So the ContactID column will have whole numbers in it

Q_: This is I lot 01.Iuff, Should I be getting I II of thi.?

A: No, it's OK if you don't understand everything right now Focus on the basic steps ilnd we'll spend a lot more time on databases in ee later chapters of the book And If you're dying 10 know more right away, you can always pick up *Head Fir I! SQL* to read along with this book

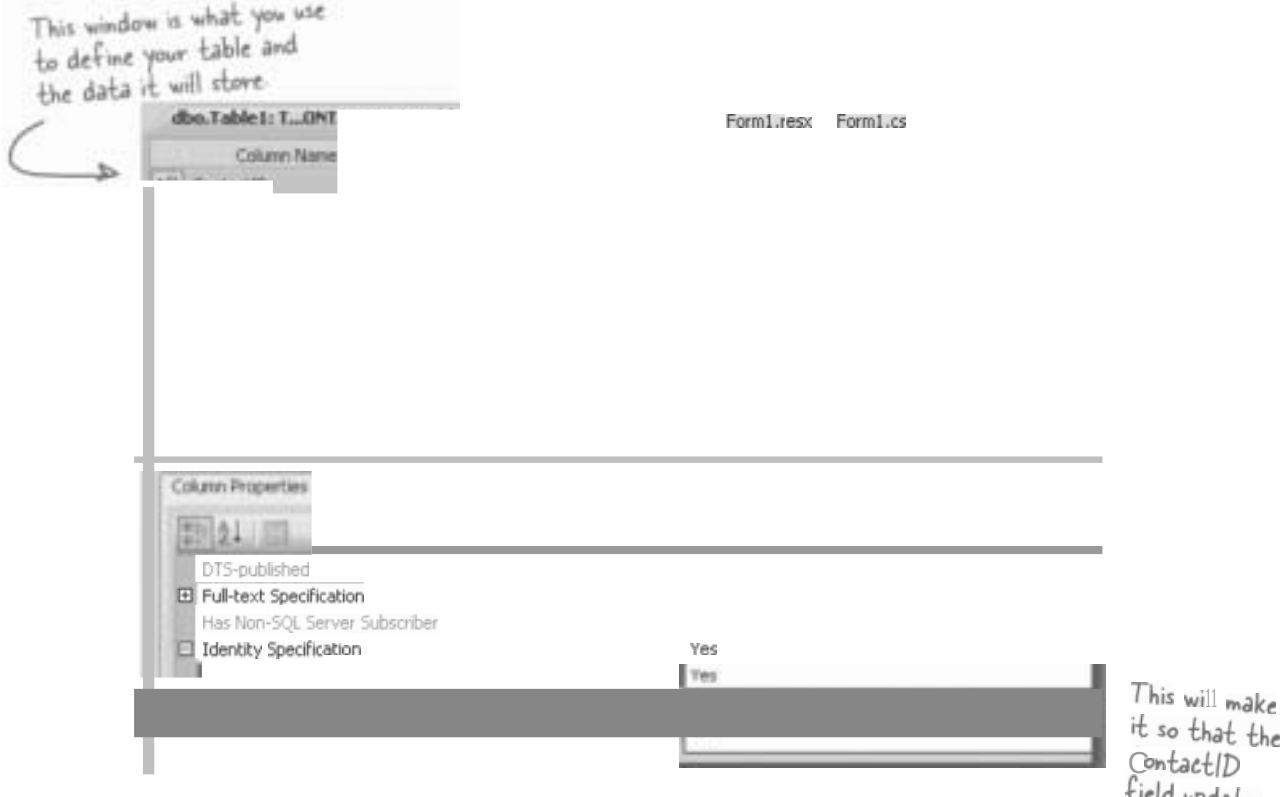


O Tell the database to auto-generate IDs.

Since ContactID is a number for the database, and not out' users, we can tell oul" database to handle creating and assigning IDs fi,r us automatically. That way, we don't hao- to worry about writing alll' code to do this.

l" the properties below your l"hl._ scroll down to Id.'nt;t) Specification. •lick the + hnlton, and select Yes next to the (Is Identity) property.

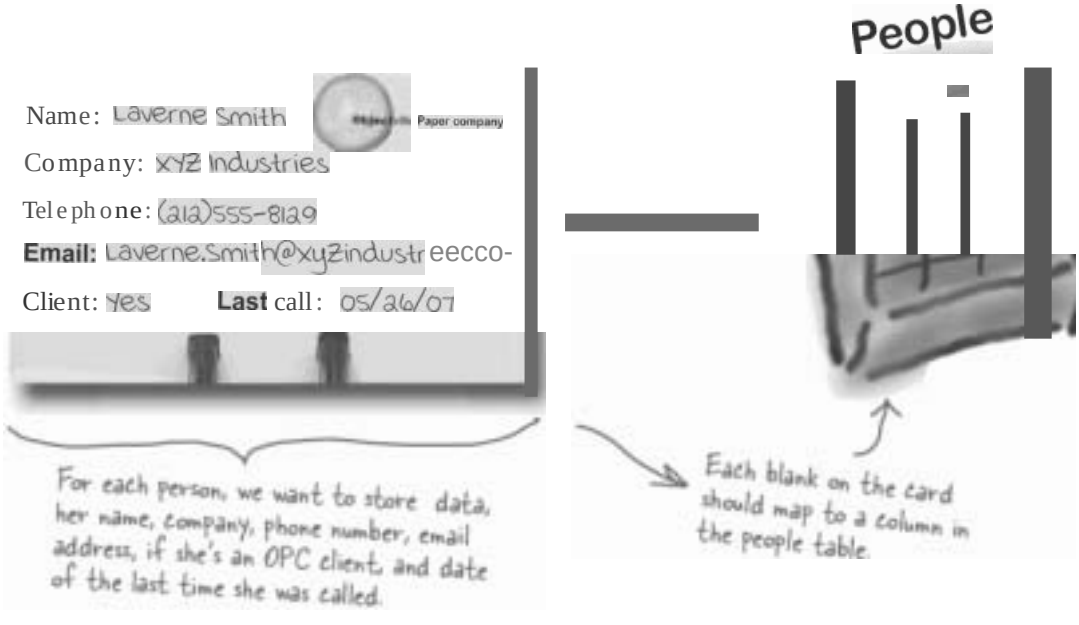
This window is what you use to define your table and the data it will store



This will make it so that the ContactID field updates automatically whenever a new record is added

Ihe hlaaks 011 eontaet card are columns ill our People table

Now Ih"t you've created a primary key for the table, you need to d"fi","ll of the fields your going to track i" the database. Each field on our "Til h'" contact card ,1",,,ld become a column in the People table.





What kinds of problems could result from having multiple rows stored for the same person?

WHO DOES WHAT?

Now that you've created a People table and a primary k,'y c◇lumn. you need (0 add c◇,hnnn, f◇r all of the data fields. Se◇ if you can work »m which data typ◇ g◇es "it h each "l' (h" c◇>Ill"""" in your table, and also match (he data typ◇ (0 tl,,' right description.

Column Name	Data Type	Description
i.,« Call	inl	This type stores a date and (i",,'
	bit	A Boolean true/false type
ContactID	nvarchar(50)	.\ string of letters. numbers and other characters with a maxi",u", length of 5◇
Client?	datetime	A whole number

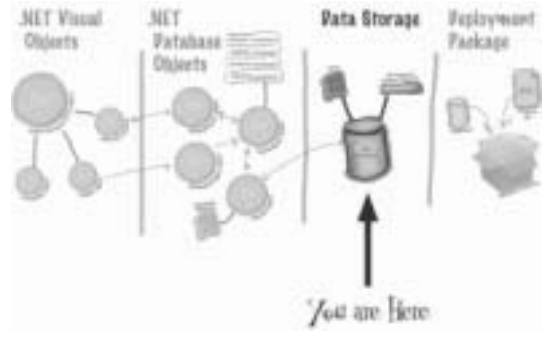
WHO DOES WHAT?

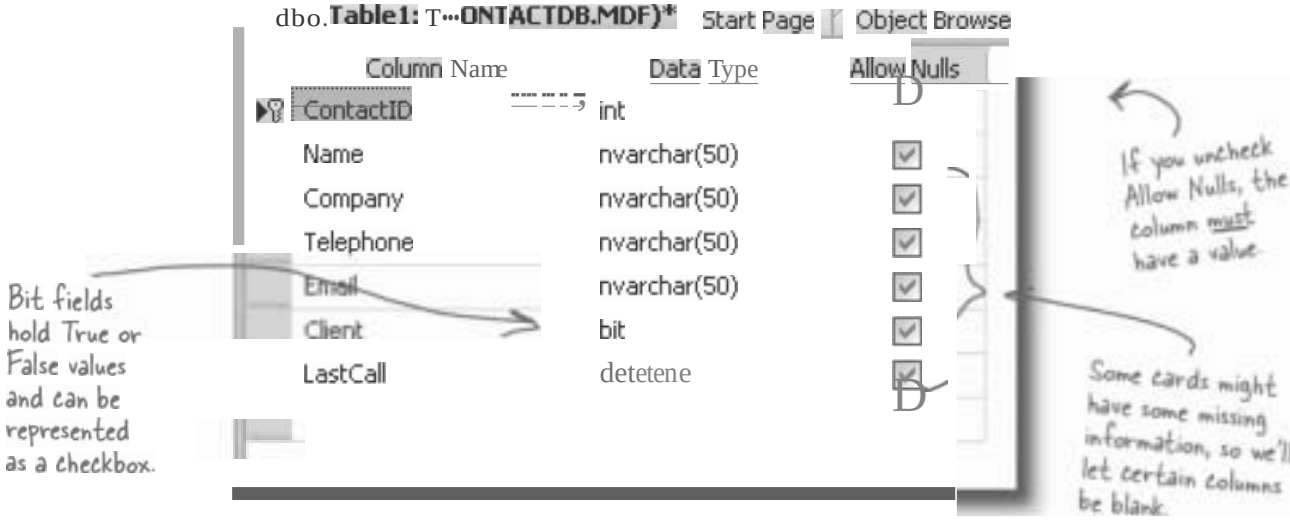
Now Ih"t you've created a People tahl(' aud a primary key column, you n('('d to add ,·"I" lllU f(,r all of Ih,' data fields. See il you can work '"'1 which data type goes with each "l' ,h,- columns in your tahl ,-. and also match the data type to the right description.

Column Name	Data Type	Description
Last Call		Thi, typ(stores a date aud lim"
Name		.\ Boolean true/false type
C,ntactID	bit	.\ string of letters. numbers and "lh,-r characters with a maxinuunb-ngrh of 50
Client?	nvarchar(50)	.\ whole "umh,-r
	datetime	

Fil1ish buildil1g the table

Go back to where you entered the ContactID column and add the "llwl' five co-hnuns frouu the contact card. 11, re's what your database table , h,»uld l,»ok lik,' "h" ll vou're done



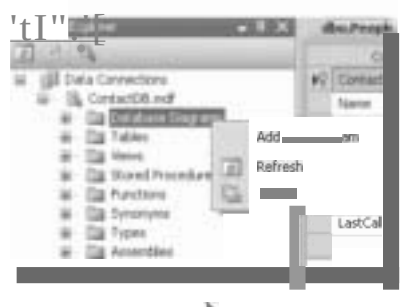


Click on the Save button on the toolbar 1n save your new table. Y" u'l) h,' asked fora nanu-. Call it "People" and ,-li,-k OK.



Diagram your data so your application can access it

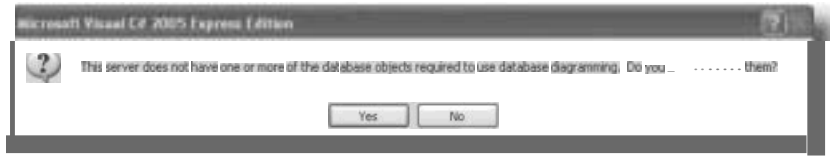
{Dnce you've created your database and tables, you have 1" let your application know about it. That's wher.. a database diagram comes ill. ,\ **database diagram** is a simple description Of your lahl,' lhal the Visua1Studio Ill E can usc to work with the table. It alsc lets the [llE aununatk-allv generate SQL statements 1" add, changc, and delete rows ill the table.



Re..e...ber, these options are all under ContactDB, so they all apply to that specific database

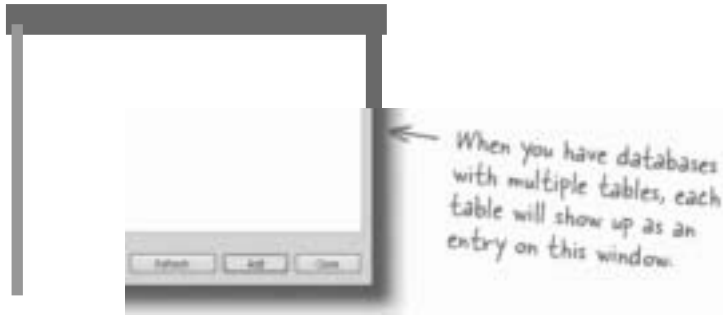
O Create a **new** database diagram.
(;" 10 II,,· Database Explorer window and right.di,-k '"' the Database 1Diagrams nod,'. Select .\dd New Diagram.

O **Let the** IDE **generate** access code.
Before you tell Ih" IDE about your specifi(table, il needs 10 'reate some basic st(,n'd pn)cedures for interacting with y(,ur database..Ju'l click Yes here, and let the IDE go 10work

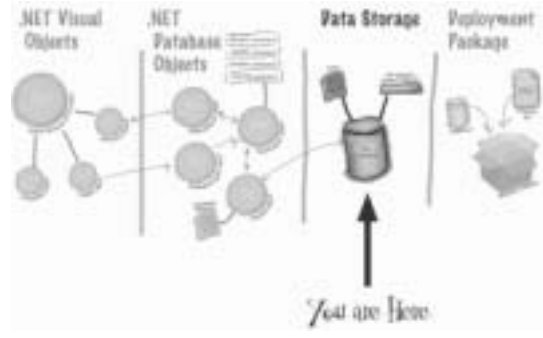


The IDE creates several stored procedures that allow your code to interact with the database you created

3 Select the tables you want to work with.
Select the People table from the window that pol" 51p, and dick .-\dd. Now the IDE is ready lo generate code specific 10 vour table.



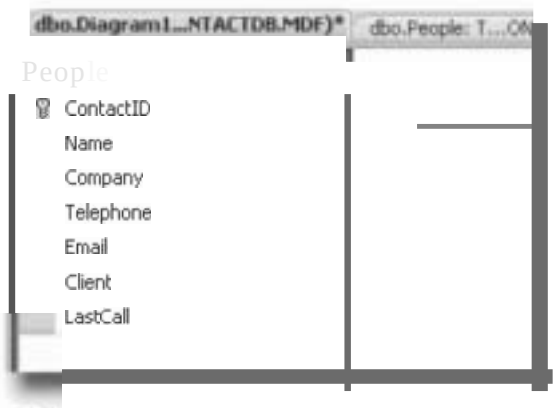
When you have databases with multiple tables, each table will show up as an entry on this window.



O Name *your* diagram PeopleDiagram.
Select File>Save Diagram. You'll be asked to name your new database diagram. Call it PeopleDiagram, and you're all set.

If you're using Visual Studio 2005, select File>Save All instead

The database diagram is shown here visually. It's a very simple representation of your table.



This is just a picture of the database design you've just done. It marks the ContactID field as your primary key and lists off all of the columns in the table.

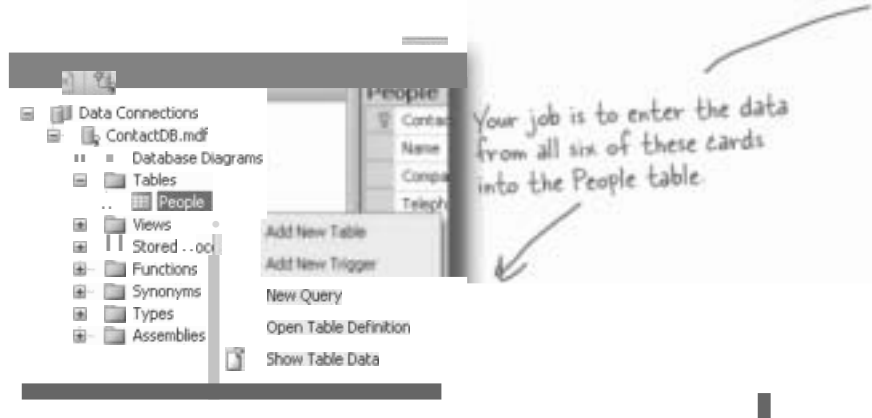
If you had any other tables in the database you wanted diagrammed, they would appear here, too.

A **database** diagram **describes** your **tables** to the Visual Studio IDE. The IDE then uses the **database** diagram to auto-generate code to **work** with your **database.**

Illsert your card data illto **the** database

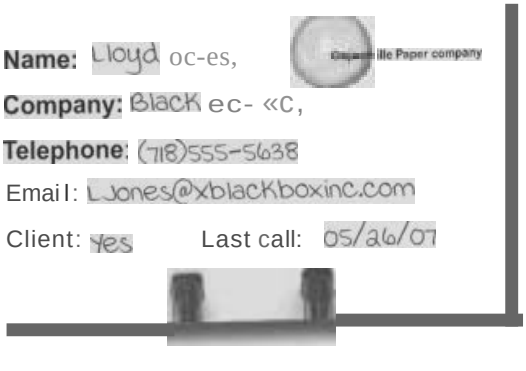
Now you're ready I" start entering cards into the database. Here ,t]"' sc'tm' "f the boss's conta :ts—we'll use th,se I" set lip the database with a few records.

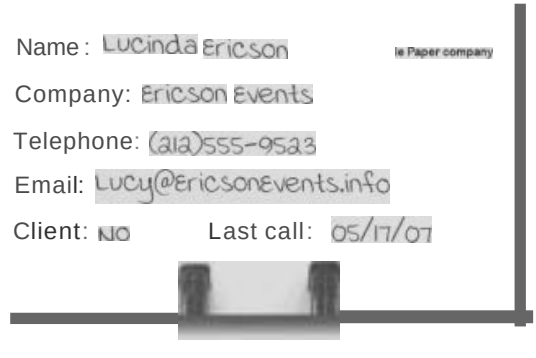
Expand '])ables and then right di,'k 011 the People Table ill the Database Explorer (or Server Explorer) ,,,I<I select Show Table Data.

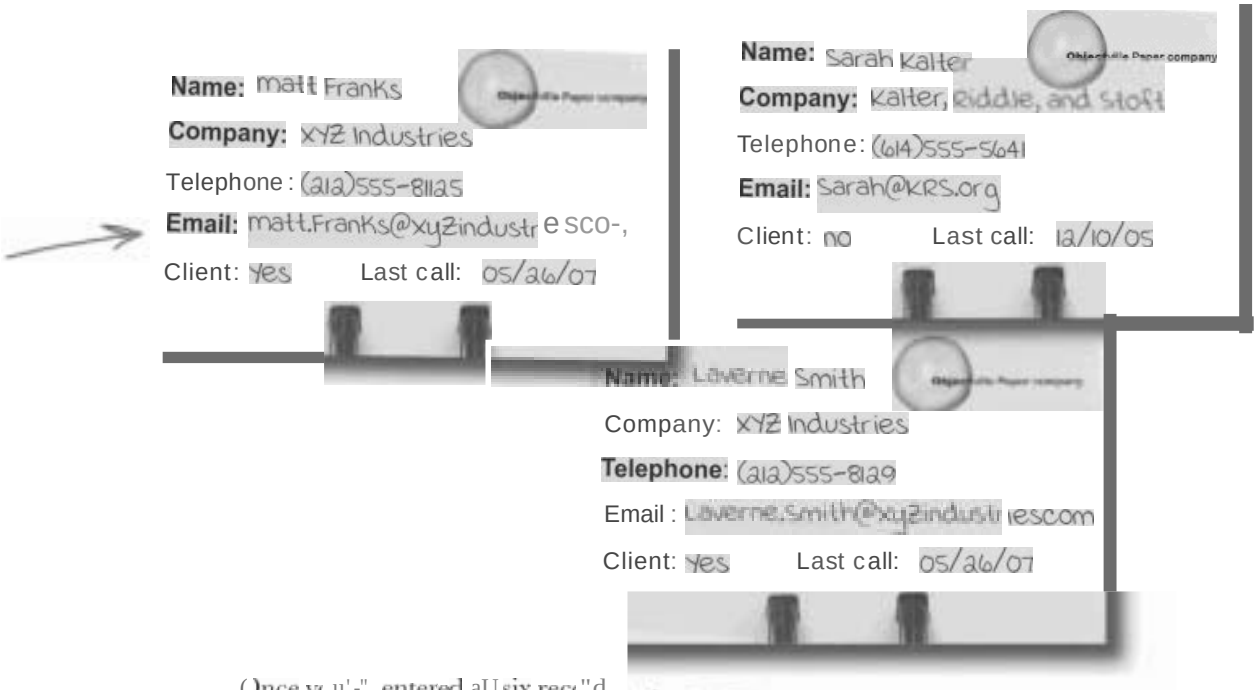


2 Once you see the Table grid ill the main window. X Cahead and add all of the data below. (You'll see all NULL values at first - j",t type over the", when you add your first row. .\I1d ignore the exclamation points that appear next !<>the data.) YOll don'l need (" lill in the ContactID column, that happens automatically.









(Once you've entered all six records, select Save All from the File menu again. That should save the records to the database.

"Save All" tells the IDE to save everything in your application. That's different from "Save", which just saves the file you're working on.

there are no Dumb Questions

Q: So what happened to the data after I entered it? Where did it go?

A: The IDE automatically stored the data you entered into the People table in your database. The table, its columns, the data types, and all of the data inside it is all stored in the Sql Server Express file, ContactDB.mdf. That file is stored as part of your project, and the IDE updates it just like it updates your code files when you change them.

Q: Okay, I entered these six records. Will they be part of my program forever?

A: Yes. they're as much a part of the program as the code that you write and the form that you're building. The difference is that instead of being compiled into an executable program, the ContactDR.mdf file is copied and stored along with the executable. When your application needs to access data, it reads and writes to ContactDB.mdf. In the program's output directory.

This file is actually a SQL database, and your program can use it with the code the IDE generated for you.

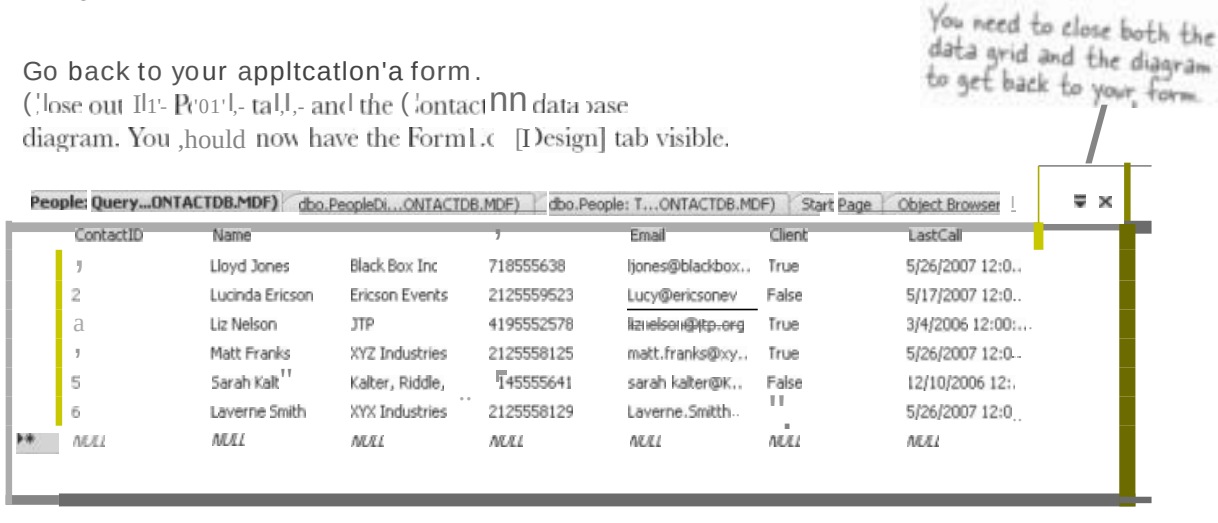


ContactDB.mdf

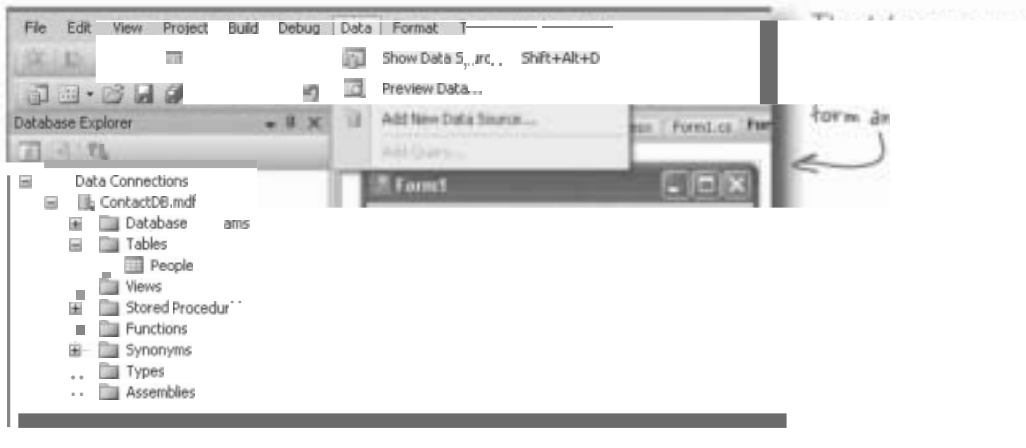
COl1l1ect your forlll to your database objects with a data source

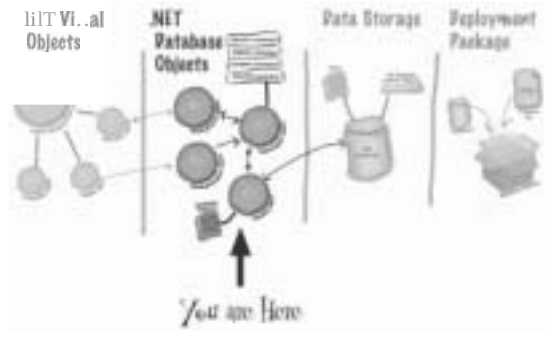
'\....re finally ready to build the .NET database "hj,·,-" that "ur form will use ·o lalk 1" your database. We need a **data source**, which i, really just a collection "f SQL statements y'111' program will use 1◇ talk '0 the ContactI HI database.

o Go back to your applticatlon'a form.
('lose out Il1'- P'01'l,- tal,l,- and l the (:ontacl nn data)ase diagram. You ,hould now have the Form1.c [Design] tab visible.



o Add **a** new data source to your application.
This should be easy by now. Click the l Iata nu-uu. and then select .\<1d New Data Source... from the drop dow n.





3 Configure your new data source.

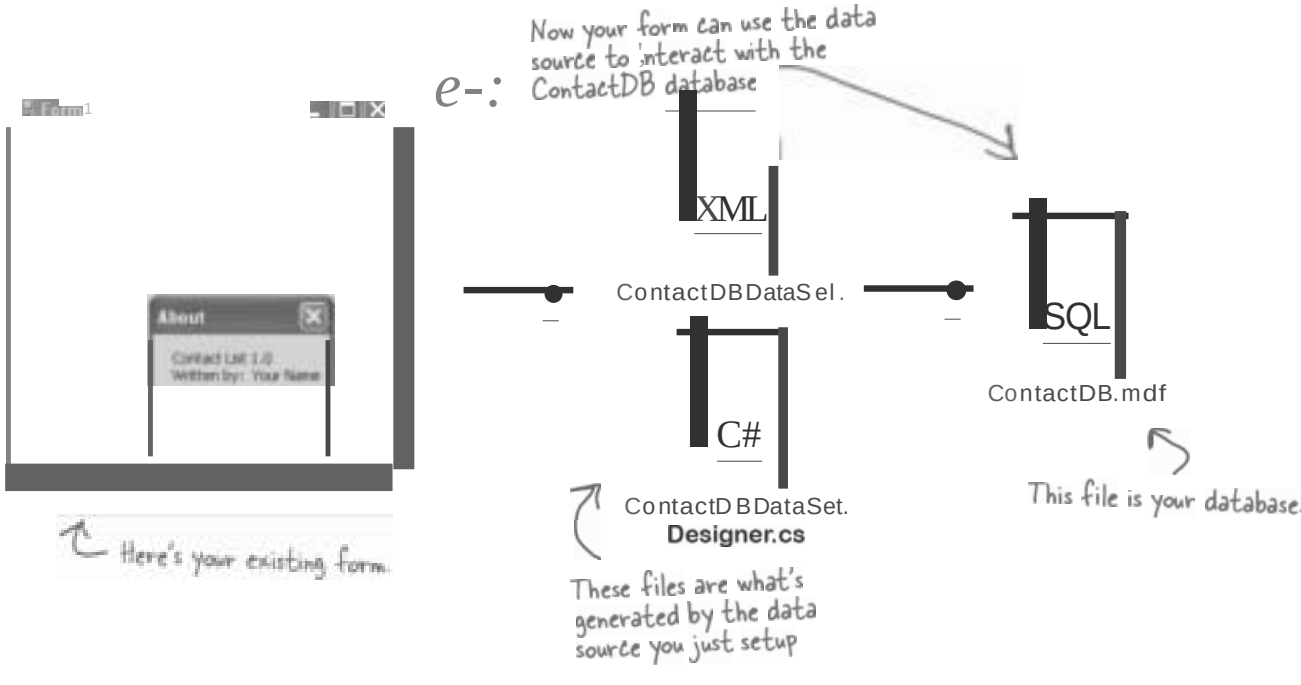
Now you need to setup I','Ur data source to u-c the ContactI11\ database. Here' what to do:

* Select Database and click the Next button.
- Click Next iuthc "Choose vour Data C"l1tl""li"tl'' screen.
- Make sure the Save the connection checkbox is checked ill the "Save the (Connection" scree·n that follow and click Next.

fr In th,' "Choose Your Objects" screen. click the Table checkbox.

- In the Dataset Name field, make sure it says "ContactDBDataSet" and click Finish,

These steps connect your new data source with the People table in the ContactDB database.



Add database-driven controls to your form

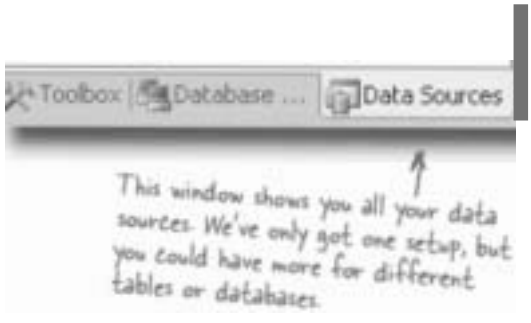
Now we can go back to our form, and add some more controls. But these aren't just any controls, they are controls that are *bound* to our database. That just means that a change in the database, like changing the columns in the People table, automatically changes the controls on the form. Or, if the controls change the data, then the matching column in the database changes.

Here's how to create several database-driven controls:

It took a little work, but now we're back to creating form objects that interact with our data storage.

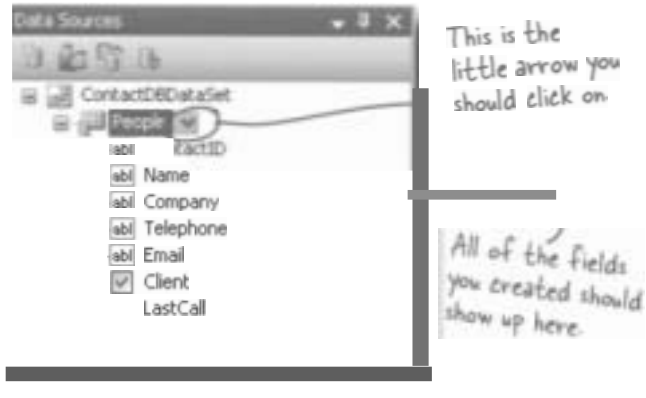
1. **Select the data source you want to use.**
   Select Show Data Sources from the Data menu. This will bring up the Data Sources window, showing the sources you have setup for your application.

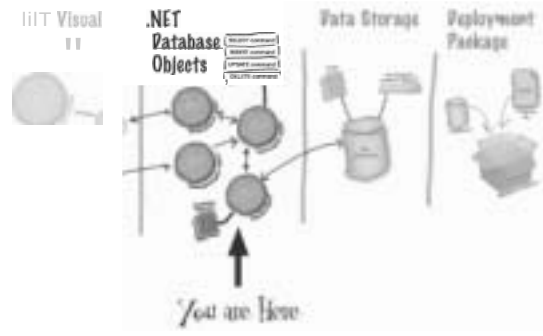
If you don't see this tab, select "Show Data Sources" from the Data menu.



You can also look to, and click on, the Data Sources tab along the bottom of your Database Explorer window.

2. **Select the People table.**
   Under the ContactDBDataSet, you should see the People table and all of the columns in it. If you don't see the columns, click the little arrow for the drop down menu, and select Details.





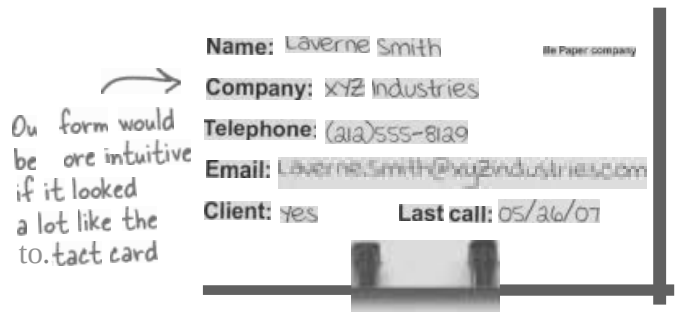
O Create controls that bind to the People table.
Drag and drop the People table onto your form. You should see controls appear for each column in your database. Don't worry too much about how they look right now; just make sure they all appear on the form.

If you accidentally click out of the form you're working on, you can always get back to it by clicking the "Form1.cs [Design]" tab, or opening Form1.cs from the Solution Explorer.



Good prograllls are illtuitive to use

Right now, the form works. 11m il doesn' look that great. Your application has to rio more than be functional. II should be easy (" use, With ,ill'l a few simple steps, you '-lll make the form look a 1"1 more like the paper cards we were using at ,I", beginning or Ihe chapter.



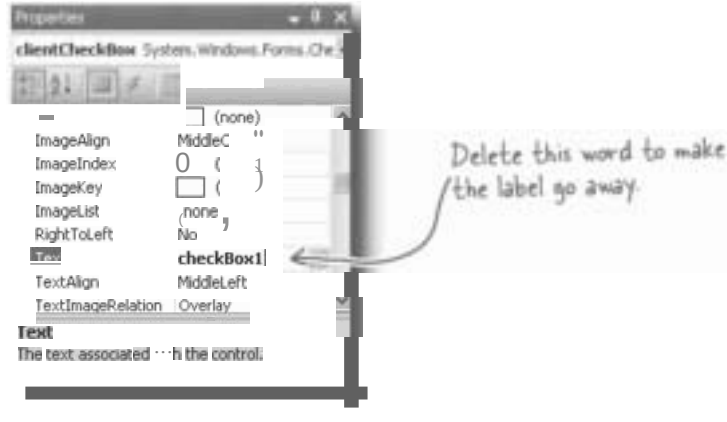
Line up your fields and labels,

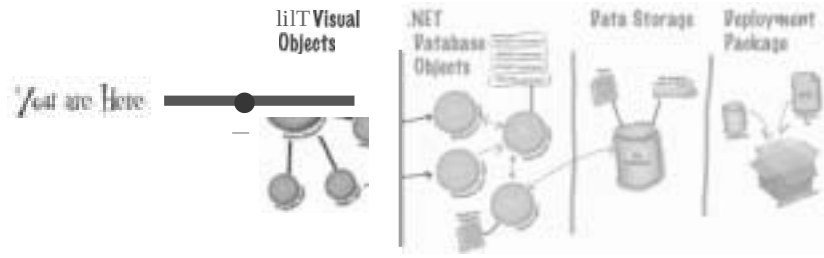
Line up your fields and labels along (h,- left ,·dge of the form. Your form will look lik,' ·ther applications, and make your users feel more comfortable using it.



2 Change the **Text** Property on the Client checkbox.

When you first drag the Ji,'ld, onto tl"'li'r,,, your Client (heckbox will have a label to the right that needs to be deleted. Right below the S"luti,,n Explorer, you'll see the properties "·ind,,,,: Scroll down 10 the Text pr"berty and delete the "checkbo»; I" label.



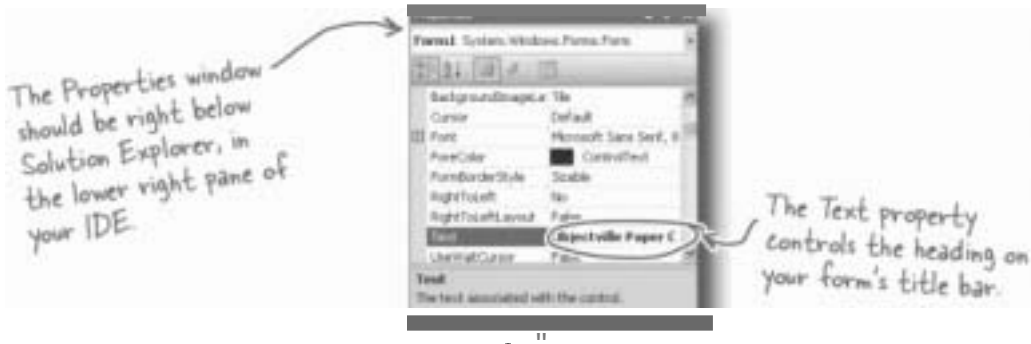


Make the application look professional.

You can change the name of the form by clicking on any space within the form, and finding the Text property in the Properties window of your IDE. Change the name of the form to "Objectville Paper Co. - Contact List."

You can also turn off the Maximize and Minimize buttons in this same window, by looking for the MaximizeBox and MinimizeBox properties. Set these both to False.

The reason you want to turn off the Maximize button is that maximizing your form won't change the positions of the controls, so it'll look weird.



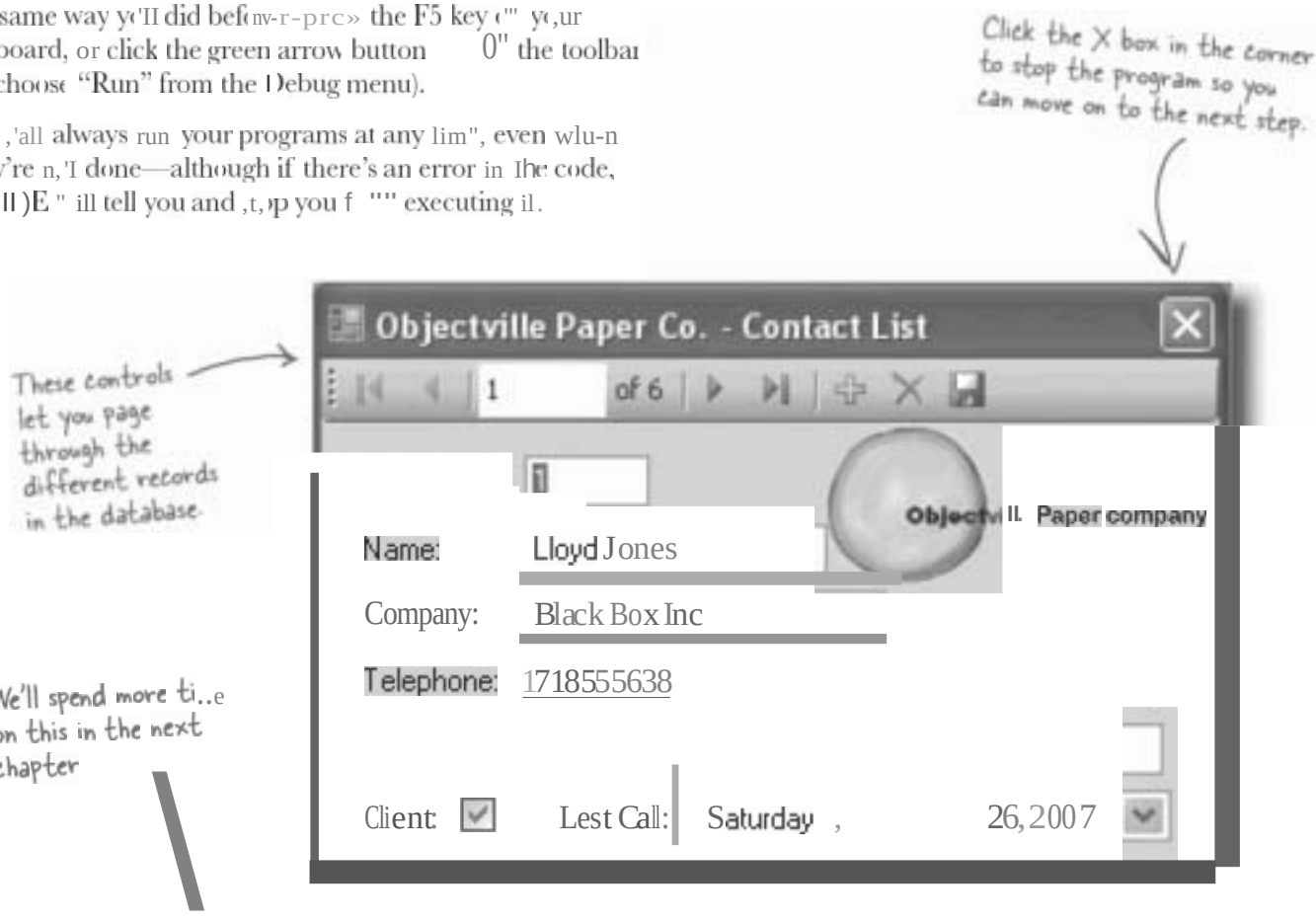
If you don't have a Properties window, you can turn lion by seiectmq It from the View croc-oown menu

A **good** application not only **works,** l,ut is easy to use. It's **always** a good idea to **make** sure it behaves as a typical user **would** expect it to.

fest drive

t)kay, jll,1 one more thing to d,)... rUIl yt,UI program and lll"k,' sure it works the way you think il should! J)" it Ih,· same way yt'II did befonv-r-prc» the F5 key (" yt,ur keyboard, or click the green arrow button 0" the toolbar (ur choose "Run" from the Debug menu).

You ,'all always run your programs at any lim", even wlu-n thev're n,'I done—although if there's an error in Ihe code, the II)E " ill tell you and ,t,)p you f "" executing il.



The IDE build, first, the. runs,

When you run "'Ill' program in the IDE il actually does tw(things. First il **builds** y'Ul' program, then il **execut..s** it. Thi, involves a fev distinct parts. It cornpil..**s** the code, 01' turns il i"lo all executabl(til,'. Then il places the compiled code, along with any resources amI other files, illl" a subdirectory underneath the bin folder.

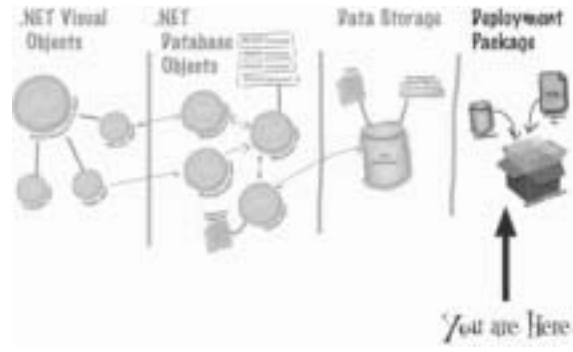
In Ihi, case, you'll find the executable and SQL database file in bini debug. Since it copies th,' database out each linne, anv changes vou make will be 10'1the next li,,"' you nm i",id,' the IDE. IIII(if ¸'ot'r"'l II", executable from Windows, it'll save your data ulllil you build again, at which point the Il1E will overwrite the SQL database with a new copy Ihal contains the data you set up from in,id,' the Database Expl"n'l,

Building your program overwrites the data in your database.

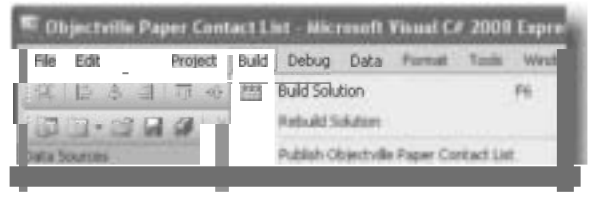
How to turn YOUR application into EVERYONE'S application

At this point, you've got a great program that only runs on your machine. That means that nobody else can use the app, pay you for it, see how great you are and hire you. And your boss and customer can't see the reports you're generating from the database.

C# makes it easy to take an application you've created, and deploy it. Deployment is taking an application and installing it on other machines. And with Visual C# 2010 Express, you can set up a deployment wizard in just a few steps.

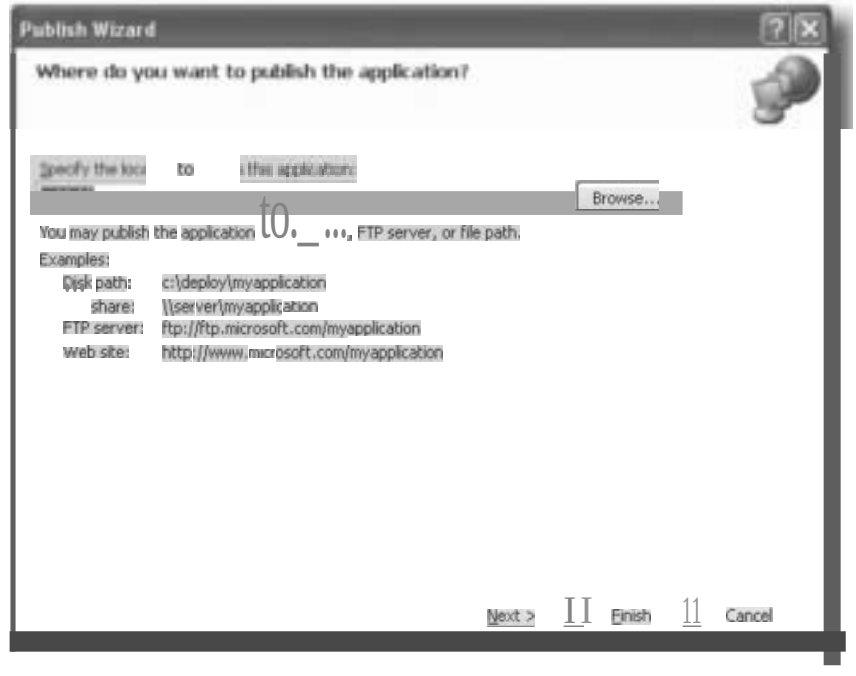


1 Select *Publish Contacts* from the Build menu.



Building the solution just copies the files to your local machine. Publish creates a Setup executable and a configuration file so that any machine could install your program.

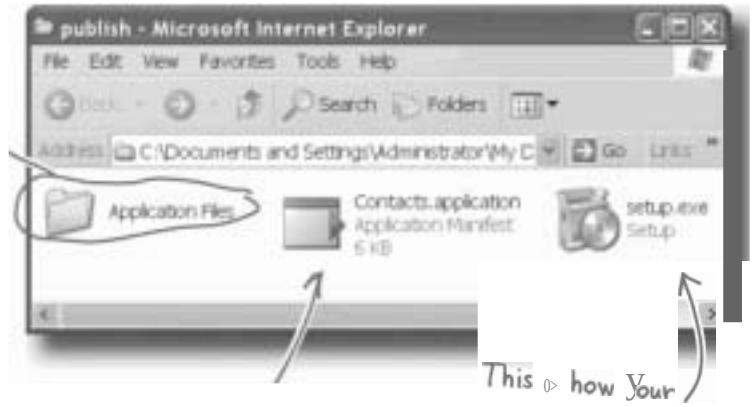
2 Just accept all the defaults in the Publish Wizard by clicking Finish. You could also package up your application and then show a folder with your Setup.exe file in it.



Give your users the application

Once you've created a deployment, you'll have a new folder called `publish`. That folder has several things in it, all used for installation. The most important for your users is `setup.exe`, a program that will let them install your program on their own computers.

This is where all of the supporting files for the installer are stored.



This is how your users will install the program on their computers!

This file tells the installer everything that needs to be included when the program is installed.

My secretary just told me that you've got the new contact database working already. Pack your bags—we've got room on the jet to Aspen for a go-getter like you!

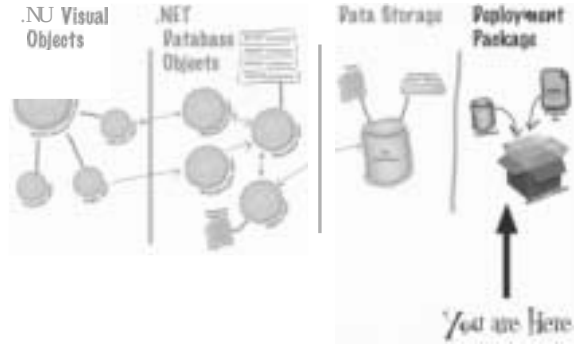


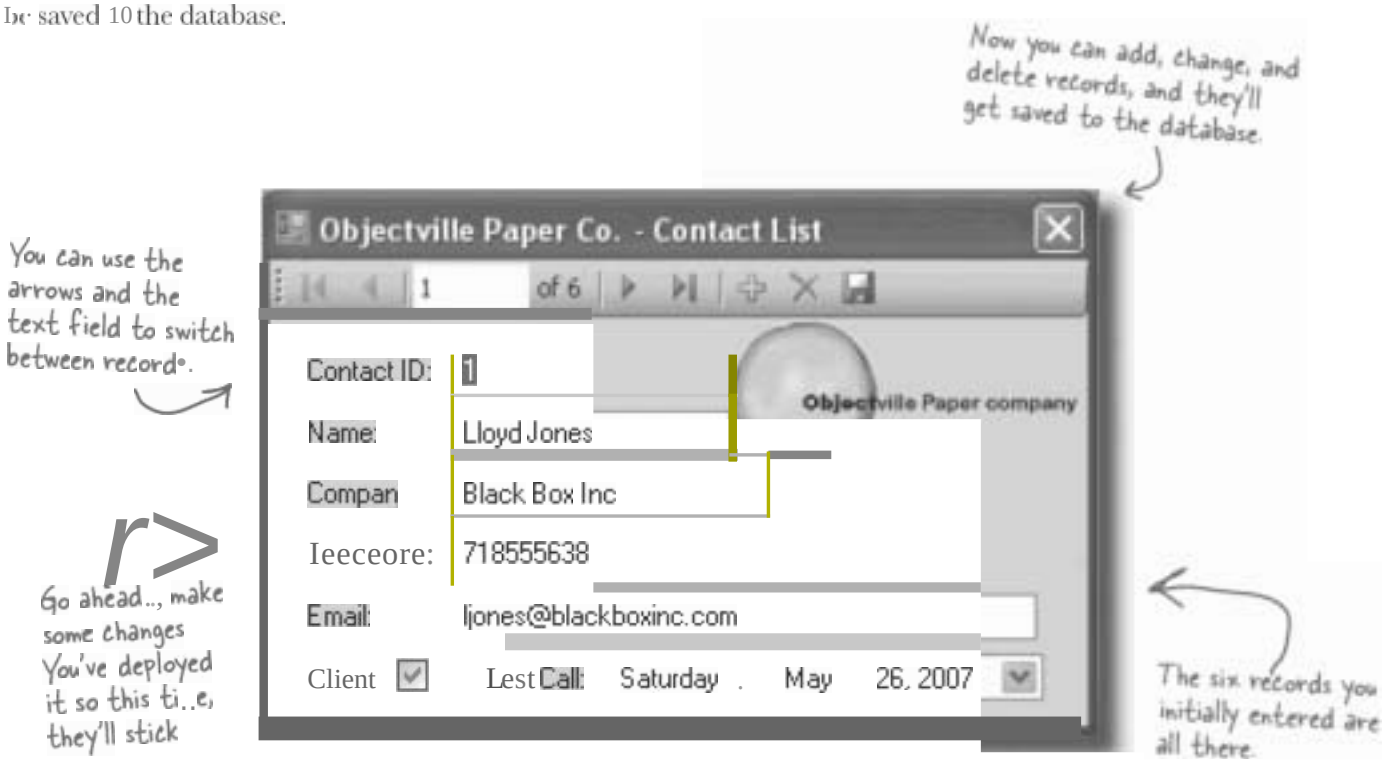
Sounds like the boss is pleased. Good job! There's just one more thing to do before you can jet off to the slopes, though...

You're NOr dOlle: test your illstallatioll

Before you pOl' ,h,- cork on any champagne bottles, you need I" 11"1 your deployment and installation. You wouldn't give anyone your program without running it lil' {. would you?

Close the \ 'isual Studio II IE. Click th.. setup program, and select a location lin your own computer to install the program. Now run it li'"ll there, and make sure il works tikI' you expect. You can add '111d change records. too, and they'll Ic· saved 10 the database.



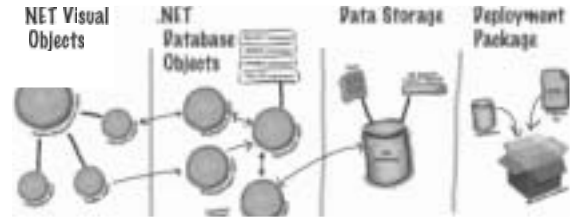


TEST EVERYTHING!

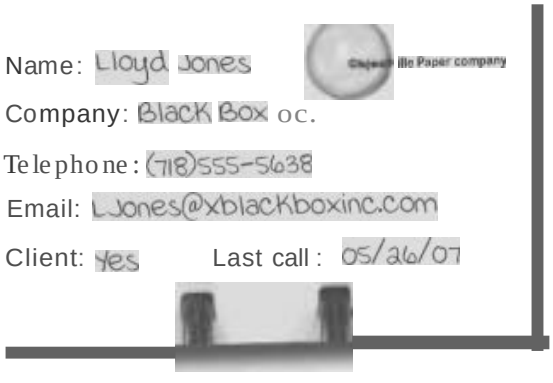
Test your program. test your Jeployment. test the Jata in your application.

You built a complete data-driven application

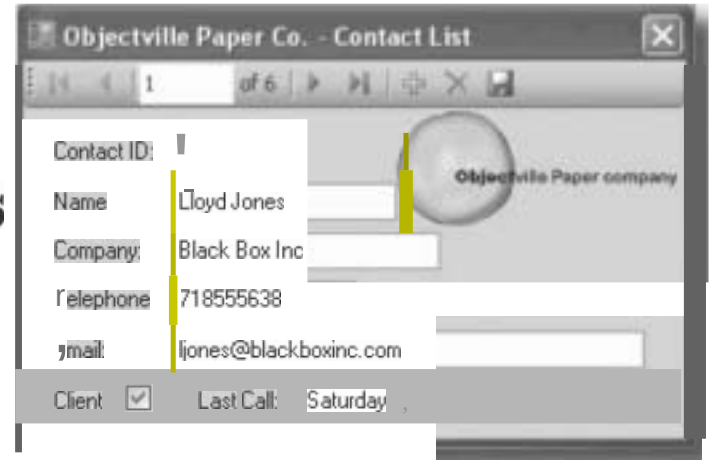
In Visual Studio 2005, you made it pretty easy to create a Windows application, create and design a database, and hook the two together. You even were able to build an installer with a few extra clicks.



From this



to this



in no time flat.

The power of Visual C# is that you can quickly get up and running, and then focus on your what your program's supposed to do... not lots of windows, buttons, and SQl access code.



CSharpcross

Take some time to sit back and exercise your C# vocabulary with this crossword; all of the solution words are from this chapter.



Across

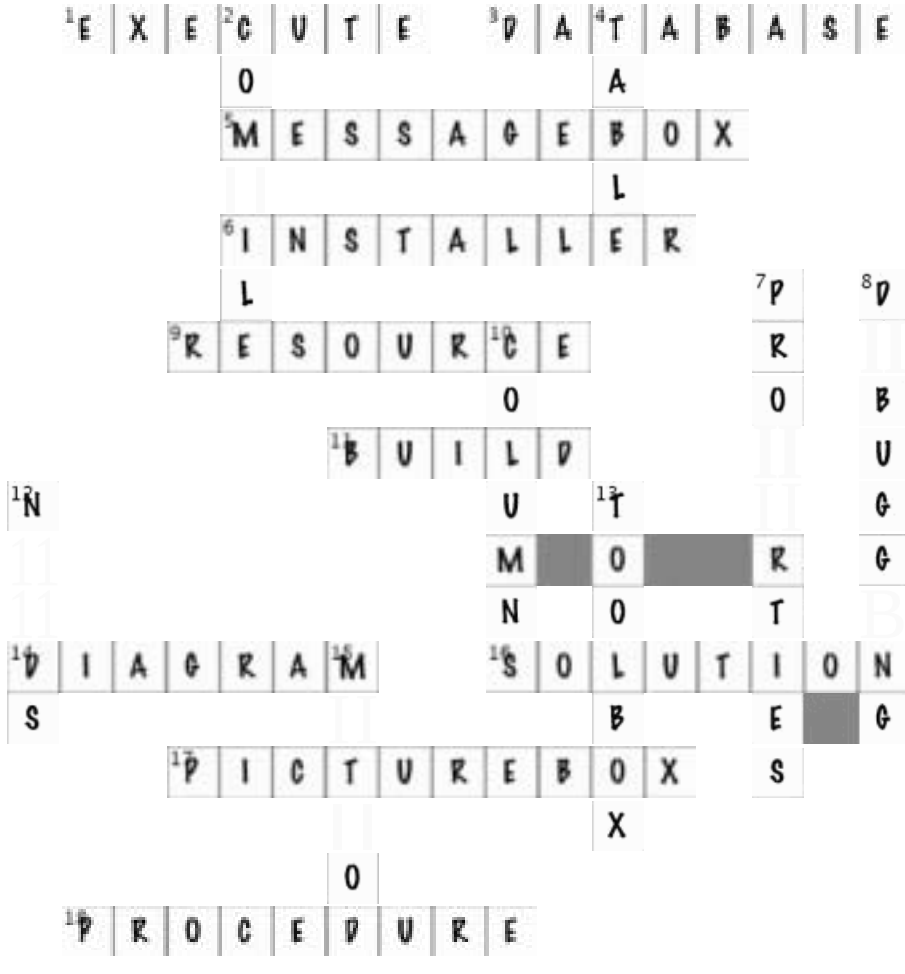
1. When you do this from inside the IDE it's called "debugging".
3. The _____ explorer is where you edit the contents 01 your SOL tables and bind them to your program
5. The "About" box in the Contact List program was one of these
6. You buikl one of these so you can deploy your program to another computer.
9 An image, sound, icon or file that's attached to your project .. • way thaI your objects can access easily
11. Before you can run your program, the IDE does this to create the executable and move files to the output directory.
14 The database give the IDE information about your database so it can generate SOL statements automatically.
16 The _____ explorer in the IDE is where you'll find the file• in your project
11. Drag one of these objects onto your form to display an image
18 A stored ''' . way for a SQL database to save queries and statements that you can reuse later

Down

2. What's happening when code is turned into an executable
4 A SOL database can use many of these to store its data
1. What you change to alter the appearance 01 behavior of objects on your form
8 What you're doing to your program when you run it from inside the IDE
10 Every row in • database contain several of these. and oi1 of them can have different data type•
12. Before you start building any application, always think about the users and their ___
13 You drag objects out of this and onto your form
1S. When you double-clicked on • visual control, the IDE created this for you and you added code to it.



CSharpcross Solution



Under the Hood



You're a programmer, not just an IDE-user.

You can get a lot of work done using the IDE. But there's only so far it can lake you. Sure, there are a lot of repetitive tasks that you do when you build an applicatioo And the IDE is great at doing those things for you But working with the IDE is only *the beginning.* You can gel your programs to do so much mce--ero **writing C#** code is how you do it Once you gel the hang of conrc. there's *nothing* your programs can't do

When you're doing this...

The IDE is a powerful tool, but that's all it is. a *tool* for you to use. Every time you change your project or drag and drop something in the IDE, it creates code automatically. It's really good at writing boilerplate code—code that can be reused easily without requiring much customization.

Let's look at what the IDE does in typical application development, when you're...

All of these tasks have to do with standard actions, and boilerplate code. Those are the things the IDE is great for helping with.



Make sure you always create a Windows Forms Application project—that tells the IDE to create an empty form and add it to your new project.

o **Creating a new Windows Application solution**

There are several kinds of application the IDE lets you build, but we'll be concentrating on Windows applications for now. Those are programs that have visual elements, like forms and buttons.

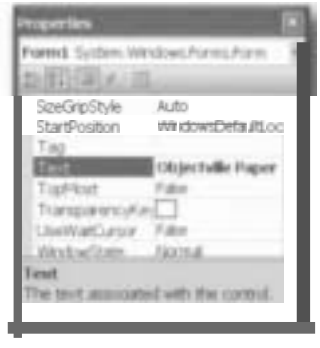
o **Dragging a button out of the toolbox and onto your form, and then double-clicking it**

Now you make things happen in your form. You'll learn a lot of terms, and explore various parts of the C# language. They're also a part of almost every C# application you'll write.



o **Setting a property on your form**

The **Properties** window in Visual Studio is a really powerful tool that lets you change attributes of just about everything in your program: all visual and functional properties for the controls on your form, attributes of your databases, and even options for your project itself.



The Properties window in the IDE is just a really easy way to edit a specific chunk of code in Form1.Designer.cs automatically. It would take a lot longer to do it by hand.

...the IDE does this

Every unn- you make a change intln- IDE, it makes a change 1◇ the ,:od1,·. which means it changes the files thar contain that code. Sometimes il jU'1 m<>difies a few lines, but other times il adds entire Jil," I" your !,,,,j,.,,!.

These files are created from a predefined template that contains the basic code to create and display a form.

O ... **the** IDE **creates the files** and folders for **the project.**

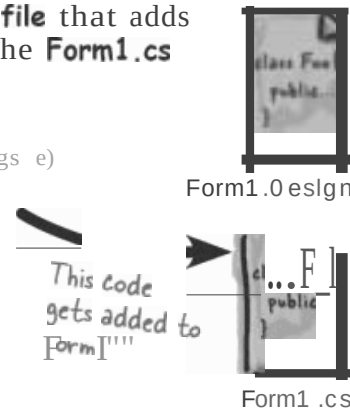


O ... **the** IDE adds code to **the** Forrnl.Designer.cs **file** that adds the button to the **form, and** then odds code to the **Form1.cs** file to **handle** the **button click.**

```
private void button1_Click(object sender, EventArgs e)
'
```

The IDE knows how to add an e..pty ..ethod to handle a button click. But it doesn't know what to put inside it—that's your job.

This code gets added to FormI''''



O ... the IDE opens the Forrn1.Designer.cs **file** and updates **a line** of code.

```
partial class Form1
'
    this.Text = "Contacts";
```

The IDE went into this file...

...and updated this line of code.



Where programs come from

A (:# program may xt.u't ollt as statements in a bunch of files, but nends up as a program running in your c(,mpu"·r. Here's how il gets there.

Every prograIM starts out as source code files



You've already ,,·,·nhow to edil a program, and how Ih(" 11DE saves vour program '0 files in a folder. Those files **are** your program—you can copy them '0 a new folder and open them til'. ami everything will be there: fc>rm,. resc,tl"'"", code, ami anything else yc'" added to your project.

You can Ihink of the IDE as a kind of fancy m- ,·dilor. It automatically does the mdvming fc" you, changes Ih,' colors of the keywords, matche III' brackets fc>r you, and "'"n suggests what "',rds might cc"!lt" next. But ill the end, all the nDE does is edit Ih,>files that contain \" "1' program.

The IDE bundles all of Ih,-Jil," for your program illl0 a solution by creating a solution (., In\ flk- and a folder that contains all or the other tu-, for the program. The sohnion file has a li'l <lr the project files (which "1,,1in .csproj) in the solution. and the project files contain lists or all the other Iii", associated with the program. I" this boc,k. you'll bc· building solutions that '>lly have one project in them. bUI you can easily add other projects to your soluti'" using the IDE's Solution Explc,n·r.

There's no reason you couldn't build your programs in Notepad, but it'd be a lot more ti··e-consu··ing.

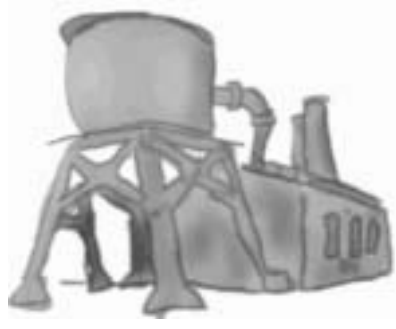
Ihe .NEf FraIMework gives you the right tools tor the Job



(:# is just a language by itself, i, can't actually *do* anything. And that's where the ,NET **Framework** comes ill. Remember tha Maximize button you turned "JI' for Ihe Contacts form? When you click Ihe Maximize burton on a window, there's code that tells the window how t" maximize itself and take up the whole "" "'"'!,1. That code is p"n "I' th,' .NET Framework. BUIIU"'. checkboxes, lists.. those are all pieces of the .NET framework. So are the inh'rnal bits that hooked your form up 10 the database. It's got tools h' draw graphics, read and writo· files, manage collection, or things... all sorts or I'"ols fc,r a 10, or jobs Ur.« programmers have 10 do every day.

The It",}; in the .NET Framework ar,-divided up into **namespaces**. You've seen these namespace before, al the top of your cc,,1· in the "using" lines. One namespac„' is called System.\\'indows.Fc,rm, - it's where vuur buttons, checkboxes, and fc',-,n' cc"n,: frc,m. Whenev',.]' you create a new \Vindows Forms Application project, Ihe II)E will add tl,,' necessary Iiies so that your project contains a fc'n Il. and those IiI," have th,' li",· `using System.Windows.Forms;` at the "'I'.

Build the progr. . . to create . " executable

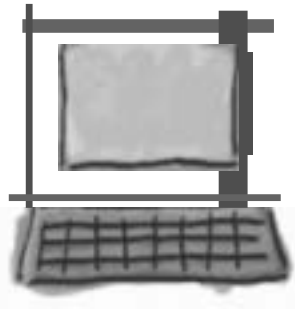


Wh..n \"11 select "Build Solution" from th.. Build menu. the IIl E **co**mpil.... your program. II does th by ""minl[.he **compil**..r. which is a tool Ih.n " "..d. your program's urce code ;"KI'llm_ it into an ..**x**..**cutabl**... The executable i file "" your disk .h", ends ill .exe– that's what you double-click I" 1'111 your 1"" '1<.un. \\1aen you huild the 1 gram, ;, creates th.. ex 11.,1 i..side the bin folder. which is insic the project folder. When y publish your sol" Ijun. it copies .It., ..xecutable LUKI any other files nc ury] into Ih." folder you on' publishing to.

When you ...lect "Start Debugging" from the n. bug m" I1". the UH: coml.'les ""ur program and nlll' the execut•ble. It's got some more advanced tools for d..b **ging** your program. which just means running ;. ,uKI being able to I "ır ·1ıre·,'k-1it s ,"u can figure out what's going u".

Your progr. . . rU"1i"lide the CLR

When you double.d i;k "" the execut•ble. Windows runs your program. But there's .m extra "layer" hetween Wind"ws and your program called Ihe Conun..n **Language** Runlim. . ..r CLR. Once upon a tim,'. "". so long ago (but ..efore C# was around), writing programs was harder, bec;,,,se you had to1deal w ilh hardware and low-level "'achi".>stuff. You nev knew exactly how some.".. was going to configure his computer. 'Il CLR •ften referre'! I" ." ,. **virtual** ma..hin.. —takes care of all that for you by doing ,. sort ..l' "translation' between your program ;",.1 \he computer running il



You don't really have to worry about the CLR much right now. It's enough to know it's there, and takes care of running your program for you automatically. You'll learn more about it as you go.

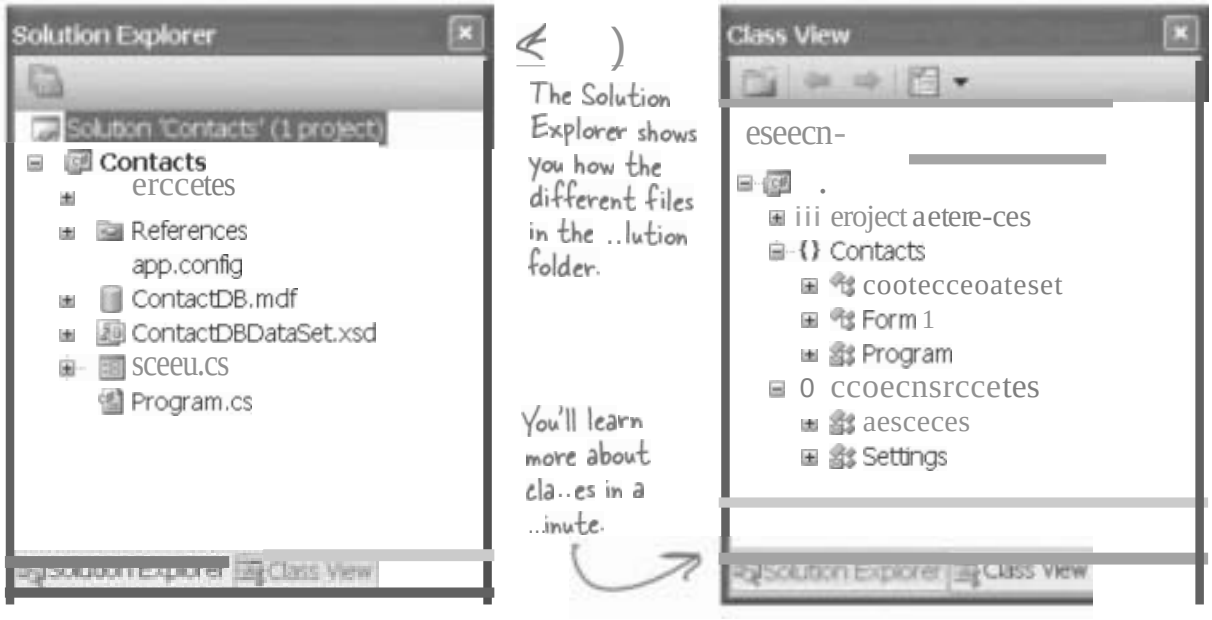
You'll learn about all sorts "r things the CLR does for you. For example, it tightly manages your computer's memory by figuring ....1 whe-n your program is finished with certain pieces of data and getting rid "r them li'r you. That's something programmers used to have to d.. themselves, and it's something th.,t you don't have to 1"·I,,,thered with. You ",,,,,'t kn.w it ut the time, but the (:LR will make your j.,10 "r1,"""ing (:# a whole 1.t easier.

Ihe IDE helps you code

You've already seen a few of the things Ihat the Il1E can do.
Let's take a closer look at some " f the tools it gives you.

O The Solution Explorer shows you everything in your project

You'll spend a 101 of time going back and forth between classes, ami Ilie easiest way 10 d" Ihal is to use Ih,' solutio explorer. It's got Iwo views: a Solutio Explorer ,;,-w (which shows you Ill,' files in your project) and a Class View (whicb ' h,ws you how vour cod logi..ally breaks down into classes).



O Use the tabs to switch between open files

Since your program is split up inlo mon-than one file, you'll usually have seve 1 code files open at once. When you do. each one will h,- ill its "" 11 Iall ill the l editor "" IDE displays an asterisk () next to a filename r it hasn' been saved yet.

Here's the form's resource file that I- added the Objectville Paper Company logo t,



When you're working on a form, you can have two tabs for it at the same time—one for the form designer, and one to view the form's code.

The IDe helps *you* write code

Did you notice little window popping up as you tvped code into the IDE? That's a feature called IntelliSense and it's really useful. One thing il d,ɔe, is show you possible ways I" """I'I" H your current Ii" e " f code. If you type MessageBox and lh,'" a period, it knows that there are thre e valid ways 10 complete that Ii"e:

```
MessageBox.
    Equals
    ReferenceEquals
    Show
```

The IDE knows that MessageBox has three methods called Equals, ReferenceEquals, and Show. If you type S, it selects Show. Press the tab or enter key to select it. That can be a real timesaver if you're typing a lot of really long ...ethod names.

[f you ,elect Show and typ((. the IDE's IntelliSense will show you information about how you can complete the line:

```
MessageBox.show(
▲ 3 of 21 ▼ DialogResult MessageBox.Show (string text, string cectco)
text: The text to display n the message box.
```

This means that there are 11 different ways that you can call the MessageBox's Show ..ethod (like ways to display different button. or icons).

The IDE also has shortcuts called **snipp..ts** Ihall"1 you type a" abbreviation 10 tell i, 10 fill iu the n"1 of the code. Here's a """ht[one: typ(mbo x and press the tab key twice, and rhe IIIE will fill in the MessageBox.Show method 1,1'you:

```
MessageBox.Show("Test");
```

When you use Start Debugging to run your program inside the IDE, the first thing it does ,i build your program. If it compiles then your program runs. If not, it won't run, and will show you errors in the Error List.

The Error List helps you troubleshoot compiler errors

If you haven't already discovered how easy il;' 10 make typos in a C# program, you'll find out very soon! Luckily, the IDE gives you a great tool for troubleshooting ,h,'nL " 'hell you hllild your ",lllliol1, any problems that keep il li')111 compiling will , h,ɔw ul' ill the Error List window at the 1",110111 "r the IDE:



Double-click "ll all error, and the IDE will jUll p I" the pr"hl,'", ill the code:

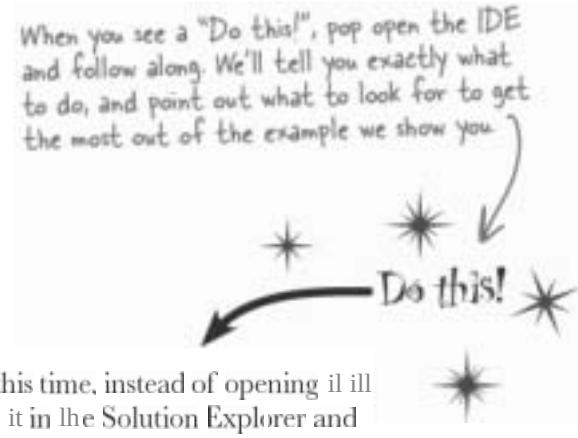
```
private void pictureBox1_Click(object sender, EventArgs e)
I
    MessageBox.Show{vcontact; List 1.0")
```

The IDE will show a red underscore where it finds you'..e ..issing a se..icolon,

Whell you challge thillgs ill the IDE, you're also challgillg your code

The Il IE is great at writing visual code for you. Hutdon't take our word li,r it, ()pI'n up Visual Studio, **cre**a le a n..w Wind ows Application proj..**ct**, and see for yourself.

When you see a "Do this!", pop open the IDE and follow along. We'll tell you exactly what to do, and point out what to look for to get the most out of the example we show you.



O **Open** up the designer code

Open the Forml .Designer.cs Iiie ill the IDE. But this time, instead of opening il ill ıh,' Form 1Designer, "p"n up its code by right-clicking on it in lhe Solution Explorer and selecting ..\'iew Code". 1ook f'ı' the Fa rmr class declaration:

```
public partial class Forml : Form
```

Notice how it's a Cartial Z!ass?

O **Open** up the form **designer** end Qdd Q PictureBox to your form

Get used t" working with more than one tab. (;" to the Solution Explorer ;I]\(I open up th,- Form designer hv double-clicking .tll Forml .es. **Drag** a n..W Pi,·tu....**Box** oulo a new form.

O f ind Qnd **expand** the designer-generated code for the PictureBox control

Then go back to the Formj •Designer.es tab in the IDE. Scroll down and I,",k for lhi, lim' in Ih,' code:

Click on the plus sign

```
⊞  Windows Form Designer generated code
```

Click on the + Oil the left-hand side or the line 10expand the code. Scroll dO"11 and find these lines:

```
//
// pictureBoxl
//
this.pictureBoxl.Location = new System.Drawing.Point(276, 28);
this.pictureBoxl.Name = "pictureBoxl";
this.pictureBoXl.Size = new System.Drawing.Size(lOO. 50);
this.pictureBoxl.Tablndex = 1;
this.pictureBox1.TabStop = false;
```

Don't worry if the .....bers in your code for the Location and Size lOe s are a little different than these...

Wait. wait! What did **that** say?

Scroll back "1' for a m;Ouit'. There ;1 is, at the top ,f the Windows Form Designer generated code section.

```
/// <summary>
11/ Required method for Designer support - do not modify
1/1 the contents of this method with the code editor.
/// <<summary>
```

Most com..ents only start with two slash (//) But the IDE eti..es adds these three-slash com..ents

There's nothing more attractive to a kid than a big sign that says. "]lon', touch thi,!" Come " u, you know you're tempted.. let's go modify Ih,- contents " f Ihat method with the ('0<1,-,-d il" ,.! Add II button to your **form,** and th..n **go** ahead and do thi",

- **Change the code** thot **sets the button1.Text property.** What do you think **it will** do to the **Properties** window in **the** IDE? (;ive it a shot_·t·what 1>"1]1<'ıs! Now go back to the Ii··m designer and check the Text property. Did it change?

- Stay in **the designer, and use the Properties window** to change **the Name** property to **something else.**
  See if you can lind a way 10 get the II IE 10 change Iht" Name property. It's ;1l 1!t.. Properti..s window at the very 1"1"' under "(Name)". What happened to the code? \\"h"I about Ih,' ('''IIIIIIe II I in the code?

  Change the code that sets **the Location property** to (0,0) end **the Size property** to make **the** button really big.
  Did il work?

- Go beck to **the designer,** and **change the** button's **BackColor** property to **something else.**
  Look closely at the Form I.Designer.cs code. \\','h" any Ii",", added?

You don't have to save the form or run the program to see the changes. Just make the change in the code editor, and then click on the designer tab to flip over to the form designer— the changes should show up immediately.

It's always easier to **use the** IDE to **change** you,' **form's Designer-generated code. But when** you **do,** any cl,ange you **make** in **the** IDE **ends** up **as** a **change** to your **projects' code.**

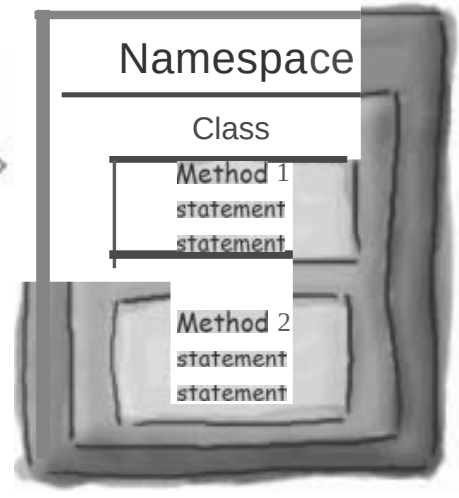
Al1afolllY of a progralll

Every (:# program's ('od,' is 'Iru,:tured in exactly Ihe same wav. .\11 programs use **namespaces**, eta**sses**, and m ..l**hods** 10make vour code ",,,i"r 1" manage.

A class contains a piece of your program (although some very small programs can have just one class).

A class has one or more methods. Your methods always have to live inside a class. And methods are made up of statements—like the ones you've already seen.

Every time you make a new program, you define a namespace for it so that its code is separate from the .NET Framework classes.



let'. take a eleser look at your code

(1p"{l up the code trom your Contact project's Formj•cs so w,. can go through il piece by piece.

O **The** code **file** starts by using **the** .NET Framework tools

You'll find a ,eI"f using lines at the lop of every program file. They tell C# which parts of the .NET Framework to use. If you use other classes Ih,u are irr other namespac-" '. then you'll add Uosi n g Ii"," for them I'o. Since forms often use a 1„1 "f dilli'rem toı,I, from Ih,' .NET Framework, Ihe 11)E automatically adds a hun('h of using lines I"'n il creates a form and adds lr to your project.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using Systern.Windows.Forms;
```

These **using** lines are at the top of every code file. They tell C# to use all of those .NET Framework classes. Each one tells your program that the classes in this particular .cs file will use all of the classes in one specific .NET Framework namespace.

2 C# programs are **organized** into classes

Every C# program is organized into **classes**. A class can do anything, but most classes do, one specific thing. When you created that program, the IDE added a class called Form1 that displays a form.

When you called your program Contacts, the IDE created a namespace for it called Contacts by adding the namespace keyword at the top of your code file. Everything inside its pair of curly brackets is part of the Contacts namespace.

```
namespace Contacts
{
  public partial class Form1 : Form
  {
```

This is a class called Form1. It contains all of the code to draw the form and the Toolbox controls on it. The IDE created it when you told it to create a new Windows Forms Application project.

3 Classes contain methods that perform actions

When a class needs to do something, it uses a **method**. A method takes an input, performs some actions, and sometimes produces an output. The way you pass input into a method is by using **parameters**. Methods can behave differently depending on what input they're given. Some methods produce output. When they do, it's called a **return value**. If you see the keyword `void` in front of a method, that means it doesn't return anything.

Look for the matching pairs of brackets. Every { is eventually paired up with a }. Some pairs can be inside others.

```
    public Form1()
    {
      InitializeComponent();
    }
```

This line calls a method named InitializeComponent(), which the IDE also created for you.

4 A statement **performs** one single action

When you added the MessageBox.Show() line to your program, you were adding a **statement**. Every method is made up of statements. When your program calls a method, it executes the first statement in that method, then the next, then the next, and so on. When the method runs out of statements or hits a `return` statement, it ends, and the program resumes after the statement that originally called the method.

This is a method called pictureBox1_Click() that gets called when the user clicks on the picture box.

This method has two parameters called sender and e.

```
    private void pictureBox1_Click(object sender, EventArgs e)
    {
      MessageBox.Show("Contact List 1.0", "About");
    }
  }
}
```

This is a statement. You already know what it does—it pops up a little message box window.

Your statement called the Show() method, which is part of the MessageBox class, which is inside the System.Windows.Forms namespace.

You statement passed two parameters to the Show() method. The first one was a string of text to display in the message box, and the second one was a string to display in its title bar.

Your program tells it where to start

When you created the new Windows Application solution, the IDE added a file called **Program.cs**. (You can see it in the Solution Explorer and double-click it. It's got a class called Program, and inside that class is a method called Main(). That method is the **entry point**, which means that it's the very first thing that's run in your program.

Every C# program can only have one entry point method, and it's always called Main(). That's how it knows where to start when you run it.

Here's some code the IDE built for you automatically in the last chapter. You'll find it in Program.cs.




```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Windows.Forms;

namespace Contacts
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

The namespace for all this code is Contacts. We'll talk about namespaces more in a few pages.

Lines that begin with slashes are comments, which you can add anywhere you want. The slashes tell C# to ignore them.

Every time you run your program, it starts here, at the entry point.

This statement creates and displays the Contacts form, and ends the program when the form is closed.

I do declare!

The first line of every class or method is called the declaration.

Remember, this is just a starting point for you to dig into the code. But before you do, you'll need to know what you're looking at.

o **C# and .NET have lots of built-in features.**

y" u"ll find lines like Ihi, "I Ih." [ol,,,f almost every C# clas Jil,'. System.Windcvs•Forms is ; **namespac**.. _T h.. using System.Windows.Forms Ii"."makes everything in th•• namespace available r" your program. In this case, thi,1 nam ,ı"" has lots "f visi..l elements ill il likt' buttons and forms.

Your programs will use more and more namespaces like this one as you learn about C# and .NET's other built-in features throughout the book.

o **The IDE chose Q namespace for your code.**

Here's th.. ,ıume -.. the IDE tTeated for you ;1 chose Contacts 1ı, 1n" your project's name..\11"f the code in yo..r program liv ill 110is namespace.

Namespaces let you use the same name in different program.. as long as those programs are..'t also •••• the same namespace

e **Your code is ste-ed in Q class.**

This particul.•r …ass is •alle1Program. Tl..· IDE created it •nd added ,I..· code that starts the program and brings up d... Cou'L..ts form.

You can have multiple classes in a single namespace

o This **code has one method, and** it contains **three** stetemeets.

.\ namespac h... , las ill iI, ,md clas have methods. Inside ..ach method is a 1."f staten Us. In this 1rogram, the statements handle starting "1' th.. Contacts form. Met1ı"I, ; where 110.. action happens every meth•• t **doe** nething.

e Each **program has** a **special** kind of **method** called the **entry** point,

Every C# program n1"..1 ha ve exactly one method called Main. 1',ı"" though your program has a lot "I""'Ih",t" only one can I", the first "„(" lh", gets executed, and that's your Mai n ","1h,ıl. C# chec ks every class ill your code for a method that reads static void Mai n (1. Then, when the I'"ogran is 1'1111, the flrsr statement ill this method gets executed, and everything else follows from that firs statement.

Every C# program must have exactly one method called Main. Tl,al method is the entry poi"t fo.· JCU' code.

Wlie" you run your code, the code in JCu.' Main() metho1 is executed FIRST.

You can ehanqe your progral\l's entry poillt

As long as your program has ,lll "lliry point, il doesn't matter which class your entry point method is ill, ,n whal that method does. Let's remove the `Main` method in `Program.ca`. and create a n..w r-ntrvpoint.



o Go back to `Program.cs` and change the name of the `Main` method to `NotMain`. Now **try to build and run** the program. What happens?

Write down what happened when you changed the method name, and why you think that happened.

o Now let' create a new entry l'0il1l_Add **a new class** called `AnotherClass.ca`. You add a class 10 your program by right-clicking on the project name in II,,· Solulion Explorer and selecting "Add>>Class.. ". Name yot'r class lil" `AnotherC 1ass.cs`. TI,,' 111E "ill add a class 10 your program called `AnotherClass`. Here's the file Ih,' IIlE added:

Right-click on the project in Properties and select "Add" and "Class..."

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;

namespace Contacts
{
    class AnotherClass
    {
    }
}
```

These four standard using lines were added to the file

This class is in the same Contacts namespace that the IDE added when you first created the Windows Application projec

The IDE automatically named the class based on the filename

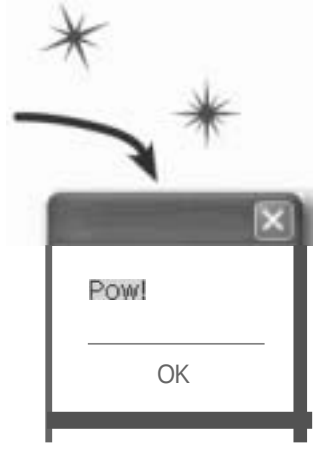
o Add a new using lin.. 10 the top of the file: `using System.Windows.Forms;` Don't forget to end th.. line with a semicolon!

4 Add this method to the **AnotherClass** class by typing it in between the curly brackets:

```
class AnotherClass
{
    public static void Main()
    {
        MessageBox.Show("Pow!");
    }
}
```

MessageBox is a class that lives in the System.Windows.Forms namespace, which is why you had to add the using line in step #3. Show() is a method that's part of the MessageBox class.

Now run it!



So what happened?

Instead "I' popping "1'Il,,' Contact application, your program now shows this messagebox. When you made the new Main() m-thod. you gave vuur program a new entry poi n!. Now the first thing the program does is Hm the statements ill that "",th,×] - "hid, means running that MessageBox.Show() statement. The",'s n<,[hing else in that method, so once you click the ( )K hlllioll. the program runs out Or statements 10 exerun and then il ends.

Figure >Ul how to fix y<,ur I'"'>gram so it P'ps lip Contacts again.

Hint: you only have to change two lines in two files to do it

Sharpen your pencil

Fill in the annotations So they describe th.. li"..., in th i, C# fil.. that they're pointing to, *We've* filled in th.. first one for you.

```
using System;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SomeNamespace
{
    class MyClass {
        public static void DoSomething()
        {
            MessageBox.Show("This is a message");
        }
    }
```

C# classes have these "using" lines to add methods from other namespaces

there are no Dumb Questions

Q: What's with all the curly brackets?

A: C# uses curly brackets (or 'braces') to group statements together into **blocks**. Curly brackets always come in pairs. You'll only see a closing curly bracket after you see an opening one. The IDE helps you match up curly brackets—just click on one, and you'll see it and its match get shaded darker.

Q: I don't quite get what the entry point is. Can you explain it one more time?

A: Your program has a whole lot of statements in it, but they're not all run at once. The program starts with the first statement in the program, executes it, and then goes on to the next one, and the next one, etc. Those statements are usually organized into a bunch of classes. So when you run your program, how does it know which statement to start with?

That's where the entry point comes in. The compiler will not build your code unless there is **exactly** one method called **Main()** which we call the entry point. The program starts running with the first statement in Main().

Q: How come I get **errors** in the Error List window when I **try** to run my **program? I thought that** only **happened** when I did "**Build** Solution."

A: Because the first thing that happens when you choose "Start Debugging" from the menu or press the toolbar button to start your program running is that it saves all the files in your solution and then tries to compile them. And when you compile your code—whether it's when you run it, or when you build the solution—if there are errors, the IDE will display them in the Error List instead of running your program.

Sharpen your pencil Solution

Fill in the annotations so they describe the lines in this C# file that they're pointing to. We've filled in the first one for you.

```
using System;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace SomeNamespace
{
    class Myclass
    {
        public static void DoSomething()
        {
            MessageBox.Show("This is a message");
        }
    }
```

C# classes have these "using" lines to add methods from other namespaces.

All of the code lives in classes, so the program needs a class here.

This is a method. Every method in the program does something. Methods are used to group statements together.

This is a statement. When it's executed, it pops up a little window with a message inside of it.

WHAT'S MY PURPOSE?

Match each "I' these fragments " I' code generated by th,' [JIE to what it does. (Some of these are n,·" take a guess and see if you got it right!)

```
partial class Forml

  this.BackColor = Color.DarkViolet;
```

Set properties for a label

```
/ /, This loop gets executed three times
```

Nothing—it's a comment tl",III!" programmer add..1 to explain th" code to anyone who's reading it

```
partial class Forml

  private void InitializeComponent I)
  !
```

I Disable the maxiinize icon ( ) i" the lill,' bar of the Fi,rm I window

```
number_of_pit_stopsLabel.Name
        - "number_of_pit_stopsLabel";
number of pH stopsLabel.Size
      -  • new System.Drawing.Size(135,  17);
numb..r_of_pit_stopsLabel.T..xt
        · "Number of pit stops:
```

.\ special kind " I' comment that the IDE use In explain what an entire block or code does

```
/// <summary>
/// Bring up th.. picture of Rover when
/// the button is clicked
/// </summary>
```

Chang(the background ""I"r " I the Ii,rm l winduw

```
partial class Forml
,

  this.MaximizeBox = false;
```

A block "I' code th"l executes whenever a program opens up a Form I window

WHAT'S MY PURPOSE?

Match each or lhese fragments or code generated by the 1l1lE Kl what it does.
lSom,· of these are new— take a guess and see if you got il right!)

```
partial class Forml

  this.BackColor - coler.Darkviolet;
```

Set properties for a label

```
// This loop gets executed three times
```

Nothing— it's a comment that lhe programmer added 10 explain the code 10 anyone who's reading it

```
partial class Forml

  ptivate void InitializeComponent I)
  !
```

Disable the maximize icon I) in the title hat' of the Fou-rnl window

```
number of_pit stopsLabel.Name
        = "number_of_pit_stopsLabel";
number_of_pit_stopsLabel.Size
        = new System.Drawing.Size(135, 17);
number of_plt stopsLabel.Text
      -  = "Number of pit stops:";
```

A special kind of comment that the lI)E use 1" explain what an entire block "I code does

```
/// <summary>
/// Bring up the picture of Rover when
/// the button is clicked
/// </summary>
```

Change lh.. background color of the Fo'1'1111 windo'\'

```
partial class Forml

  this.MaximizeBox = false;
```

.\ block "I' code that execute whenever a program opens up a Fo,m I1 window

Two classes can be in the same namespace

Take a look at these two class files from a program called PetFiler2. They've got three classes: a Dog class, a Cat class, and a Fish class. Since they're all in the same PetFiler2 namespace, statements in the Dog.Bark() method can call Cat.Meow() and Fish.Swim(). It doesn't matter how the various namespaces are divided up between files. They will all be combined when they're run.

SomeClasses.cs

```
namespace PetFiler2 {

    public class Dog {

        public void Bark() {
            // statements go here
        }
    }

    public partial class Cat {

        public void Meow() {
            // more statements
        }
    }
```

When a class is "public" it means every other class in the program can access its methods

MoreClasaes.cs

```
namespace PetFiler2 {

    public class Fish {

        public void Swim() {
            // statements
        }
    }

    public partial class Cat {

        public void Purr() {
            // statements
        }
    }
```

Since these classes are in the same namespace, they can all "see" each other—even though they're in different files. A class can span multiple files too, but you need to use the "partial" keyword when you declare it.

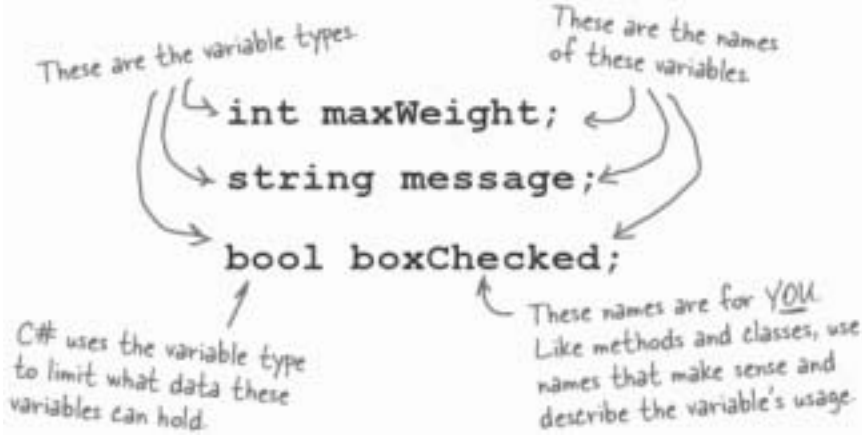
You can only split a class up into different files if you use the `partial` keyword. You probably won't do that in any of the code you write in this book, but the IDE used it to split your form up into two files, For...l.cs and For...] Designer.cs

Your prograllls use **variables** to work **with** data

When you get right down to it. every program is basically a dala cruncher. Sometimes th.. data is in the lorm of a do-mu-m. or an imag(in a video game, or an instant message. litH it's all .iU'1 data. And that's where **variabl**..s u >n,,' ili. .\ variable i, what your program uses 10 'I'ore data.

Declare your variable.

Whenever you d**eclare** a variable, you tell your program its *type* and its *name*. Once C# knows your variable's type, it'll keep your program from compiling if you make a mistake and try to do something that doesn't make sense, like subtract "Fido" from 48353



Yariable. vary

.\ varial>[,. i, equal 1'. different values at different time while youu program runs. In other words, a variable's ' -ahie ***v..ries***. (Which is why "variable" is such a good name.) This is really important, because that idea i, at the core of every proer-nn that you've written or will ever write. So if your program sets the variable myHeight equal to 63:

```
int myHeight - 63;
```

any ti",e myHeight appears ill tilt· code, C# will replace il with its value, 63. Then. later Oil. if you change iI' value 10 12:

```
myHeight • 12:
```

C# "ill replace myHeight "itlI12—but the variable i' 'Iill called myHeight.

Whenever your **progr'am needs to work with** number-s, text. **true/false** values, **or any other kind** of dala, you'll use ,'ariables to **keep track** 01 **them.**

You have to assign values to variables before you use them

Try typing these statements into a C# program:

```
int z;
MessageBox.Show("The answer is " + z);
```

Go ahead, give it a shot. You'll get all "IT" red and the IDE will refuse to compile your code. That's because the IDE checks each variable to make sure that you've assigned it a value before you use it. The easiest way to make sure you don't forget to assign your variables values is to combine the statement that declares a variable with a statement that assigns its value:

```
int xWeight = 25000;
string message = "Hi!";
bool oxChecked = true;
```

These values are assigned to the variables

Each declaration has a type, exactly like before

A few useful types

Every variable has a type that tells C# what kind of data it can hold. We'll go over a lot of detail about the different types in C# in Chapter 4. In the meantime, we'll concentrate on the three most popular types. `int` holds integers (or whole numbers), `string` holds text, and `bool` holds Boolean true/false values.

> **var-i-a-ble, adjective.**
> able to be changed or adapted
> *the drill's variable speed bit let Bob change the drill speed from slow to fast based on the job he had to do.*

If you write code that uses a variable that hasn't been assigned a value, your code won't compile. It's easy to avoid that error by combining your variable declaration and assignment into a single statement,

Once you've assigned a value to your variable, that value can change. So there's no disadvantage to assigning a variable an initial value when you declare it.

C# uses familiar math symbols

Once you've got some data stored in a variable, what can you do with it? Well, if it's a number, you'll probably want to add, subtract, multiply, or divide it. And that's where up..**rators** come in, You already know the basic """', Let's talk about a few ",", Here's a block of code that uses operators to do some simple math:

We declared a new int variable called number and set it to 15. Then we added 10 to it. After the second statement, number is equal to 25.

```
int number = 15;
number = number + 10;
number = 36 * 15;
number = 12 - (42 / 7),
number += 10;
number *= 3;
number = 71 / 3;
```

The third statement changes the value of number, setting it equal to 36 times 15, which is 540. Then it resets it again, setting it equal to 12 – (42 / 7), which is 6.

This operator is a little different. += ..eans take the value of number and add 10 to it. Since number i. currently equal to 6, adding 10 to it sets its value to 16.

The *= operator is si..ilar to +=, except it ..ultiplies the current value of number by 3, so it ends up set to 48

71 divided by 3 is 23.666666. Since number is an integer, it can only store whole numbers, so it gets rounded to 23.

```
int count = 0;
count ++;
count
```

You'll use int a lot for counting, and when you do the ++ and -- operators come in handy ++ incre..ents count by adding one to the value, and -- aeere..ents count by subtracting one from it, so it ends up equal to zero.

This MessageBox will pop up a box that says "hello again hello"

```
string result = "hello";
result += " again " + result;
MessageBox.Show(result);
result = "the value is: " + count;
result = "";
```

e-r--: When you sse the + operator with a string, it just puts two strings together. It'll automatically convert nu..bers to strings for you

The "" is an e..pty string It has no characters (It's kind of like a zero for adding strings.)

A bool stores true or false The ! operator ..eans NOT It flips true to false, and vice versa

```
bool yesNo = false;
bool anotherBool = true;
yesNo = !anotherBool;
```



Don't **worry about** m**emorizing** th**ese operators** noW,

You'll get to k"" w them because you'll see 'em over and over again.

Loops **perform an action** over alld over agaill

Here's a peculiar thing about most large programs: they almost always involve doing certain things over and over again. And that's what **loops** are for– they tell your program to keep executing a certain set of statements as long as some condition is `true` (or `false`!).

That's a big part of why booleans are so important. A loop uses a test to figure out if it should keep looping.

```
while (x > 5)
{
    x = x - 3;
}
```

In a while loop, all of the state...ents inside the curly brackets get executed as long as the condition in the l'arentheses is true

Every for loop has three state...ents. The first sets up the loop. The state..ent w,ll keep loopin as long as the second one is true. And the third state...ent gets executed after each ti..e through the loop

```
for (i = 0; i < 8; i = i + 2)
{
    MessageBox.Show("I'll pop up 4 times");
}
```

Use a code "'ippet to write **simple** for loops

You'll be typing for loops in a just a minun-, and the IDE can help speed up your coding a little. Type `for` followed hI' two tabs, and the IDE " ill automaticallv insert code for you. If you type a new variable, il'll automatically update the n"1 or the snippet. Press tab again, and the cursor will jump 10 the length.

Press tab to get the cursor to jump to the length. The number of times this loop runs is determined by whatever you set length to. You can change length to a number or a variable.

```
for (int i = 0; i < len th; i++)
{
}
```

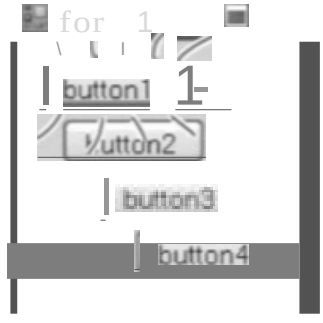
If you change the variable to something else, the snippet automatically changes the other two occurrences of it

Time to start codll1g

The real work of any program is in its statements. Bill statements don't exist in a vacuum. So let's set the stage for digging ill ;l[ld getting some code I".itt,'n. **Create a new Windows F....ms Application pr..ject**

Tl.....ll tell the IDE to create a new project ..ith a blank form and an entry point. You ...ight want to name it something like "Chapte.. 2 program I"'—you'll be building a whole lot of programs throughout the book

Build this form



Add statements to show a m",ag,

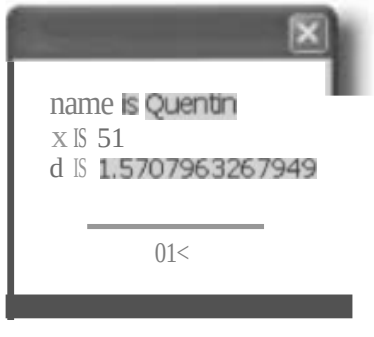
Get started bv dc lic king ''' the first hnn"". Then add these 6 statements to tl buttonI_Click() method. Look closelv ..1 1he c. ..le, ,mel the "u'I"" it prod"ces.

x is a variable. The "int" part tells C# that it's an integer, and the rest of the statement sets its value to 3.

```
pci vate void buttonl_Click(object ..ender, EventArgs el
{
    // this is a comment
    String name = "Quentin";
    int x = 3;
    x = x * 17;
    double d = Math.PI / 2;
    MessageBox.Show("name is " + name
        + "\nx is " + x
        + "\nd is " + d);
```

There's a built-in class called Math, and it's got a member called PI. Math lives in the System namespace, so the file this code came from needs to have a using System; line at the top.

The \n is an escape sequence to add a line break to the message box.



Syntax 101

- Each statement mu.t end in a semicolon.

```
x - x + 1;
```

- A single-line <omm enl begins with two forward slashes.

```
// this line is ignored
```

- Most white space doesn't matter.

```
x                3
```

is the same as

```
x - 3;
```

- Variables are declared with a name and a type (there are plenty 01 types that you'll learn about in chapter 4).

```
int weight;
// weight is an integer
```

- Classes and methods must be defined within a pair of eurly braces.

```
public void go() I
  // amazing code here
```

if/else statements make decisions

Use **if/else** statements to tell your program to do certain things only when the conditions you set up are (or aren't) true. A lot of if/else statements check if two things are equal. That's when you use the == operator. That's different from the single equal sign (=) operator, which you use to set a value.

Every if statement starts with a conditional test.

```
if (someValue == 24)
{
    MessageBox.Show("The value was 24.");
}
```

The statement inside the curly brackets is executed only if the test is true

Always use two equal signs to check if two things are equal to each other.

```
if (someValue == 24)
{
    // You can have as many statements
    // as you want inside the brackets
    MessageBox.Show("The value was 24.");
} else {
    MessageBox.Show("The value wasn't 24.");
}
```

if/else statements are pretty straightforward. If the conditional test is true, the program executes the statements between the first set of brackets. Otherwise, it executes the statements between the second set.



Don't confuse the two equal sign operators!

You use one equal sign (=) to set a variable's value, but two equal signs (==) to compare two variables. You won't believe how many bugs in programs—even ones made by experienced programmers!—were caused by using = instead of ==. If you see the IDE complain that you cannot implicitly convert type 'int' to 'toot', that's probably what happened.

Set Up coltditiolts altd see if they're true

Use **if/..ls.. stat..ments** to tell your program to do certain things only when the conditions you s..(ul' are ("r aren't) true.

Use logical operators to check ,,"dltio",

You've just looked at the -- "p,·rator. whn-hyou use to test whether two variables are equal. There are a few other operators too. Don't worry about lhemorizing lhem right now—you'll get to know them over the next few chapters:

- The != operator works a 1„(like -- except it's true if the two things you're comparing are not ..qual
- You can use > and < to compare numbers and see if one is bigger or smaller than (he 011„r.
- The -o-, !- . >, and < are called conditional op..rators. \\'hell you use rlu-rn r" test two variables " r values, it's called perfo·rming a conditional tests.
- You can ,'o",hi"," imli,'idllal conditional tests imo ou- long test using tl„' && operator for AND and the || operator for ()R. So to check if i equals 3 or j is less Ihan j. do (i -- 3) || (j < 5).

When you use a conJitional operator to compare two numbers, it's calleJ a conJitional test.

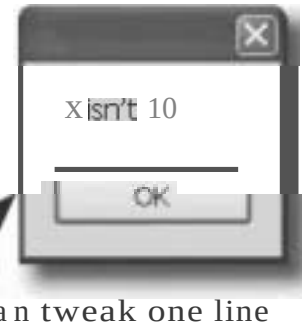
Make sure you stop your progra.. before you do this—the IDE won't let r- edit the code while the program's nning. You can stop it b closing the „„dow, using the stop bu ton on the toolbar, or selecting "Stop Debugging" from the Debug ..enu

Set a variable al'ld thel'l check its value

Here's th" code for rhe second b"tto". It's an if/else ,t"ll'''''ll that checks an integer **variable** called x to see if it', equal to 10.

```
private vOld button2 CIICk(ob)ect sender, EventArg5 e)
{
    int x = 5;
    if (x == 10)
    {
        MessageBox.Show("x must be 10");
    (
    else
        MessageBox.Show("x isn't 10");
```

First we set up a variable called x and make it equa1 to 5 Then we check if it's equal to 10.

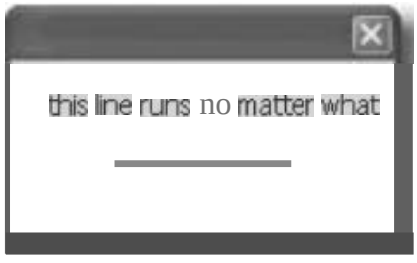


Here's the output. See if you can tweak one line of code and get it to say "x must be 10" instead.

Add aaether co.ditio.al test

The third huuol1 mak,·, lhi, output. Now make a change to two line of code so tha i' pops up h", h message boxe,.

This \ine checks someValue to see if it's less than 3, and then it checks to ..eke sure name is "Joe".




```
private vOld button3 Cllck(obJect sender, EventArgs e)

    lnt someValue - 4;
    String name - "Bobbo Jr.";
    if ((someValue < 3) && (name.Equals("Joe")))

        MessageBox.Show("x is 3 and the name is Joe");

    MessageBox.Show("this line runs no matter what");
```

Add loops to your program

H,·n", the code for the last hutton, It·, got two 1''')jI ' The first is a whil.. loop, which repeats the statements inside the brackets as long as the """di,i"'l is true ,10something *while* this is true. The second ,„l<- is a for I,»op. Take a lo«,k and ".,- h,»w il works.

This loop keeps repeating as long as the , ount variable is le.. than *10*.

The second part of the for statement is the test. It says "for as long as i is less than five the loop should keep on going".

This sets up the loop It j..t assigns a value to the integer that'll be used in it.

This is where the loop actually does something. In this case, it adds one to i. So every ti..e the loop executes, it will add 1 to ;.

```
private void blltton4_CLick(object sender, EventArgs e)
{
    int count _ 0;

    while (count < 10)
    {
        count - count + 1;

    for int i =    i < 5;'i++)

        count = count    1;

    MessageBox.Show("The anSwer is " + count);
```

Before you click on the button, read through the code and try to figure out what the message box will show. Then click the button and see if you were right!

Sharpen your pencil

Let', get a lillie more practice with conditionalte,t, and loop,. Take a look ilt the code below. Circle the con ditionillte,u. and fill in the blanks \0 that the comments correctly describe the code that's being run.

```
int result = 0; II this variable will hold the final result
int x .. 6; II declare a variable x and  set it to 6
While (x > 3) I
  II execute these statements as long as

  result = result + x; II add x


  x .. x - 1; II subtract
}
for (int z = 1; z < 3; z .. z + 1)
  II start the loop by
  II keep looping as long as
  II after eaeh loop,
  result .. result + z; II
}
II The next statement will pop up a message box that says
//                                                    .
MessageBox.Show("The result is " + result);
```

We filled in the first one for you

More about conditional tests

You can do simple conditional tests by checking the value of a va..iable using a comparison ope..ator. He..e's how r- compa..e two nu...bers, x and y:

- x < y (less than)
- x > Y (greater than)
- x -- y (equals-and yes. with two equals signs)

These are the ones you'll use most often.



Wait up! There's a flaw in your logic. What bcppens to my loop if I write a conditional test that never-becomes false?

Then your loop runs foreverl

Every time your program runs a conditional ("'I, the result i, ..ither **true** or **false**, If it's t.rue.fhen your program goes through the loop ("„-""'1',' ime. Every loop, I"ould have code rh.u, if ii" run enough lim,-" ,1"",ld cause the conditional 1,·,1 10 eventually return **false**, But if il doesn't then the l,..,p will k,·ep running \lnlil you killthe program or (lim Ih,' computer off!

Sometimes you call this a "...finite loop"

Sharpen your pencil

Here are a few loops. Write down if each loop will repeat forever or eventually end, If it', going to end, how many times will it loop?

Loop #1

```
int count - 5;
while (count> 0) |
   count   count * 3;
   count   count ' -1;
```

Loop # 2

```
int i   0;
while |i    0)
   count   count • 3;
   count   count • -1;
```

Loop #3

```
intj -2;
for (int i   l; i< 100;
     i = i • 2}

   j = j - i;
   while (j < 25)

      j - j + 5;
```

Loop #4

```
whiLe (true) { int i = 1;)
```

Loop #5

```
intp - 2;
for (int q   2; q < 32;
     q - q * 2)

   while (p < q)

      p - p * 2;

   q -p- q;
```

BRAIN POWER

Can you think of a reason that you'd want to write a loop that never stops running?

Sharpen your pencil Solution

Lets get a little more practice with conditional tests and loops. Take a look at the code below. Circle the conditional tests, and fill in the blanks so that the comments correctly describe the code that's being run.

```
int result = 0; // this variable will hold the final result
int x_6; // declare a variable x and set it to 6
while (x > 3)
  // execute these statements as long as x is greater than 3
  result = result + x; // add x to the result variable
  x _ x- I; // subtract 1 from the value of x
)
for (int z - 1; z < 3; z - z + 1) I
  // start the loop by declaring a variable z and setting it to 1
  // keep looping as long as z is less than 3
  // after each loop, add 1 to z
  result = result + z; // add the value of z to result
}
// The next statement will pop up a message box that says
// The result is 18
MessageBox.Show("The result is " + result);
```

This loop twice—first with z sd; to J, and then a second time with z set to 2. Once it hits 3, it's no longer less than 3, so the loop stops

Sharpen your pencil Solution

Here are a few loops. Writ" down if each loop will repeat forever or eventually end. If it's going to end. how many times will it loop?

Loop # 1
This loop executes once

Loop # 2
This loop runs forever

Loop # 3
This loop executes 1 times

Loop #4
Another infinite loop

Loop #5
This loop executes 8 times.

there are no Dumb Questions

Q: I. every piece 01 cod- -lways in a class?

A: Yes Any time a C# program does something, it's because statements were executed Those statements are a part of classes, and those classes are a part of namespaces Even when it looks like something is not a statement in a class ike when you use ee designer to set a property on an object on your form—if you search through your code you'll find that the IDE added or changed statements inside a class somewhere

Q: Are there any ""m. .paces rm not allowed 10 UN? Are there any I *have* 10 use?

A: Yes, there are a few namespaces you're not allowed to use Notice how all of the ua i n g lines 0ll the top of your C# class files always said System? That's because there s a System namespace that's used by the NET F I . I _ k It s where you find all of your important tools to add power to your programs Like Systee • Da t.e. which lets you work with tables and databases and System.Math• which has mathematical functions. But for the most part, you can choose any name you want for a namespace (as long as only has letters, numbers and underscores). When you create a new program, the IDE will automatically choose a namespace for you based on the program name

Q: I still don't get why I need this partial class . tufl.

A: Partial classes are how you can spread the code for one class between more than _ file. The IDE does that when it creates a form—it keeps the code you edit in one file (like Form1 cs), and the code it modifies automatically for you in another file (Form1.Designer.cs) You don't need 10 do that with a namespace, though. One namespace can span two, three or a dozen 01 more files Just put the namespace declaration at the top of the file, and everything within the curly brackets after the declaration is inside the same namespace. One more thing you tiln have more than one class in a file. And you tiln have more than _ namespace in a file You'll learn a lot more about classes in the next few chapters

Q: Let's say I drag something o nlo m, lorm. • o Ih. IDE **generat. . . bunch 01 code automatically. What happens** 10 Ih.1 **code II I click** "Undo"?

A: The best way to answer this question is to try it! Give it a shot—do something where the IDE generates some code for you Drag a button on a form, change properties Then try to undo II What happens? Wdl , lor simple things what you'll see is that the IDE is smart enough to undo it itself But for more complex things, like adding a new SOI database to your project, you'll be given a warning message It sill knows how to undo the action, but it may not be able to redo it

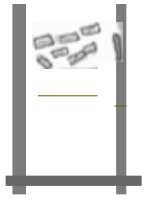
Q: So exactly how careful do I have 10 be with the code that's automatically generated by th. IDE?

A: You should generally be pretty careful It's really useful to know what the IDE is doing to your code, and once in a while you'll need to know what's in there in order to solve a serious problem But in almost all cases, you'll be able to do everything you need 10 do through the (IE



BUlln POINTS

- You tell your program to perform actions using statements Statements are always part of eesses. and every class is in a namespace
- Every statement ends with a sen iicolon (;)
- When you use the visual tools IIlhe Visual Studio IDE. it automatically adds or changes code in your program
- Code blocks are surrounded by curly braces t I. Classes. while loops, if/else statements and lots of other kinds of statements use those blocks
- , A conditional test is either `true` or `false` You use conditional tests to determine when a loop ends, and which block of ecce to execute II an if/else statement
- Any nrre your program needs to store some data, you use a variable Use = to assign a variable, and == to test if two variables ae equal
- A `while` loop runs everything wFltun its block (defined by curly braces) as long as the *conditional test* is `true`.
- II the conditional test is `fal ae`, the while loop code block won't run, and execution will move down to the code immedialely after the loop block,



Code Magnels

Part of a C» program i, all scrambled up o r the fridge. Can you rearrange the code snippets 10 make a worki r>g C' program that produces the message box? Some of the curly braces fell or> the floor and they were too srnallto pick up, so feel t-ee 10 add as many of those as you need!

The "" '1 an empty string—it means Result ha. .... characters in it yet

```
string Result ..
```

This magnet didn't fall off the fridge.

```
MessageBox.Show(Result);
```

```
if (x - 1) |
Result .. Re."It • "d";
• • • . 1
;
```

```
if (s == 21 |
Result .. Result • "b c·;
;
```

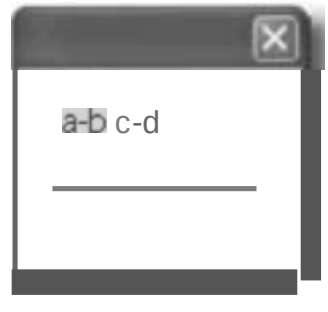
```
if (x > 21 |
Result .. Result • "a";
|
```

```
int x = 3;
```

```
x = '' - 1;
Result .. Result +
```

```
while (x > 0) (
```

Output:



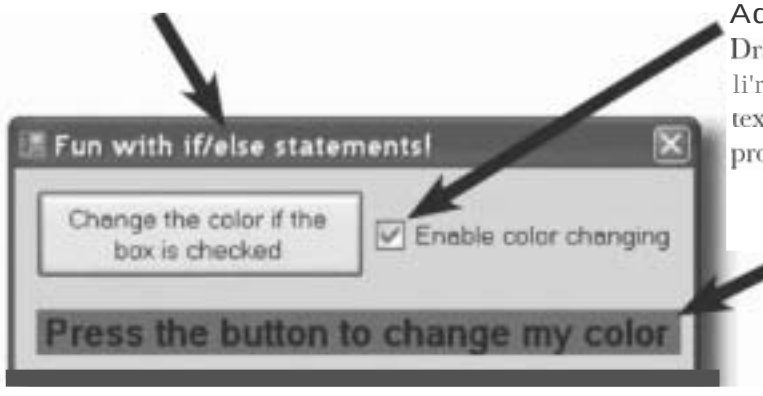
Answers on page 82.

We'll give you a lot of exercises like this throughout the book. We'll give you the answer in a couple of pages. If you get stuck, don't be afraid to peek at the answer—it's not cheating!



Time to get some practice using if/else statements. Can you build this program?

Here's the form.



Add **this** checkbox.

Drag it out of the (,",ll",x and onto your li'rm. Use the Text property to change the text that's next to il. (You '11'0 use Ih.. Text property to change ,h,' butt"u and label I,·XI.)

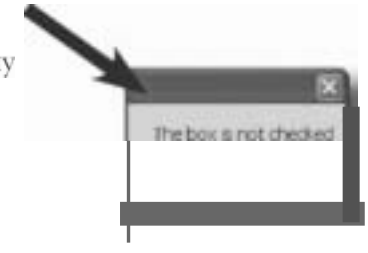
This is a label.

You can use the pr<.)><rties I,. change the font size and make it boldface. Use Ih,' BackColor property 10 set I" red—choose "**Red**" from the selection of web colors.

Pop up this message if the user clicks the button but the **box** IS NOT checked.

If your checkbox is named checkBox! (you can change the **Name** property if you want), then here's the conditional test to see if it's checked:

```
eheekBoxl .Cheeked == t rue
```



If the **user clicks** the button and the box IS checked, change the background color of the label.

I" the label background color is red. change it 10 blue when the blilloll;' clicked. If it's blue, change it back I" "rd. Here's a statement that sets the background color of a label called **label1**:

```
l abe ! l ,BackColor = Color .Red ;
```

[11;111: The conditional 1,"1 to check whether a label's background color is red looks a lot like thar '1"1"""",1- hut with 0"" ;Il'I" 'rt'It,t difference!)



Let's bUild something flashy!

0 **Here's the form to build**



Hint: If you declare a variable inside a for loop—for (int c = 0; c—then that variable's only valid inside the loop's curly brackets. So if you have two for loops that both use the variable, you'll either declare it in each loop or have one declaration outside the loop.

2 **Make the form background go all psychedelic!**
When the button's clicked, make the form's background color cycle through a whole lot of colors! Create a loop that has a variable **c** go from 0 to 254. Here's the block of code that goes inside the curly brackets:

```
this.BackColor = Color.FromArgb(c, 255 - c, c);
Application.DoEvents();
```

This line tells the program to let the operating system do other things outside of your program. Without it, your program will take up all the CPU cycles—and it won't listen for events that happen (like the user clicking the X box to close the window).

Color me impressed!

.NET has a bunch of predefined colors like Blue and Red, but it also lets you make your own colors using the `Color.FromArgb()` method, by specifying three numbers: a red value, a green value, and a blue value.

3 **Make it slower**
Slow down the flashing by adding this line before the `Application.DoEvents()` line:

```
System.Threading.Thread.Sleep(3);
```

This statement inserts a 3 millisecond delay in the loop. It's a part of the .NET library, and it's in the `System.Threading` namespace.

You'll be creating a lot of applications throughout this book, and you'll need to give each one a different name. We recommend naming this one "2 Fun with if-else statements" based on the chapter number and the text in the title bar of the form.

Make it smoother

Let's make the colors cycle back to where they started. Add another loop that has C go from 254 down to 0. Use the same block of code, including the curly brackets.

Keep it going

Set around your two loops with another loop, just before the "End Sub" statement so that when the button is pressed, the background starts changing colors and then keeps doing it. (Hint: The `while (true)` loop will run forever!)

When one loop is inside another one, we call it a "nested" loop.

Un-on! The program doesn't stop!

Run your program in the IDE. Start it looping. Now close the window. Wait a minute—the IDE didn't go back into edit mode! It's acting like the program is still running. You need to actually stop the program using the square Stop button in the IDE (or select "Stop Debugging" from the Debug menu).

Make it stop

Make the loop you added in step #5 stop when the program is closed. Change your outer loop to this:

```
while (Visible)
```

Now run the program and click the X box in the corner. The window closes, and then the program stops! Except… there's a delay of a few seconds before the IDE goes back to edit mode.

When you're checking a boolean value like Visible in an if statement or a loop, sometimes it's tempting to test for `(Visible == true)`. You can leave off the "== true"—it's enough to include the boolean.

When you're working with a form or control, Visible is true as long as the form or control is being displayed. If you set it to false, it makes the form or control disappear.

Hint: The `And` operator means "AND". It's how you string a bunch of conditional tests together into one big test that's true only if the first test is true AND the second is true AND the third, etc. And it'll come in handy to solve this problem.

Can you figure out what's causing that delay? Can you **fix** it **so** the program ends immediately when you close the window?



Time to gel some practice using if/else statements. Can you bUild this program?

Here's the entire Forml.cs file for the "Fun with If/Else State..ents!" exercise If we show you a lot of eede in a file like this, we'll draw a grey box behind the part that you should add.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Fun_with_If_Else
{
    public partial class Forml : Form
    {
        public Forml()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (checkBox1.Checked == true)
            {
                if (label1.BackColor == Color.Red)
                {
                    label1.BackColor = Color.Blue;
                }
                else
                {
                    label1.BackColor = Color.Red;
                }
            }
            else
            {
                MessageBox.Show("The box is not checked");
            }
        }
    }
}
```

Here's the code for the form We named r solution "Fun with If Els..", so the IDE made the na espace Fun_with_If_Else If you gave your solution a different name, it'll have a different namespace

The IDE added the method called button1_Click() to your form when you double-clicked on the button. The method gets run every time the button's clicked.

The outer if statement cheeks the checkbox to see if it's been chezked Check!

The inne- if statement checks the label's col If the label is cu ently ved, it executes a statement to turn it blue

This statement's run if the label's background color is not red to make it set back to red

This MessageBox pops up if the checkbox isn't checked

You can download the code for all of the exercise solutions in this book from www.headfirstlabs.comlbookslhfcsharpf



let's build something flashy!

Exercise Solution

Sometimes we won't show you the entire code in the solution, just the bits that changed. All of the logic in the FlashyThing project is in this button1_Click() method that the IDE added when you double-clicked the button in the form designer.

When the IDE added this ..ethod, it added an extra return before the curly bracket. Someti..es we'll put the bracket on the sa..e line like this to save space—but C# doesn't care about extra space, so this is perfectly valid

```
private void button1_Click(object sender, EventArgs e)
{
    while (Visible) {
        for (int c = 0; c < 254 && Visible; c++) {
            this.BackColor = Color.FromArgb(c, 255 - c, c);
            Application.DoEvents();
            System.Threading.Thread.Sleep(3);
        }
        for (int c = 254; c >= 0 && Visible; c--)
            this.BackColor = Color.FromArgb(c, 255 - c, c);
            Application.DoEvents();
            System.Threading.Thread.Sleep(3);
    }
}
```

The outer loop keeps running as long as the form is visible. As soon as it's closed, Visible is false, and the while will stop looping

The first for loop makes the colors cycle one way, and the second for loop reverses them so they look smooth.

We fixed the extra delay by using the & operator to make each of the for loops also check Visible. That way the loop ends as soon as Visible turns false

Can you figure out what's causing that delay? Can you **fix** it so the program ends immediately when you close the window?

The delay happens because the for loops need to finish before the while loop can check if Visible is still true. You can fix it by adding `&& Visible == true` to the conditional test in each for loop

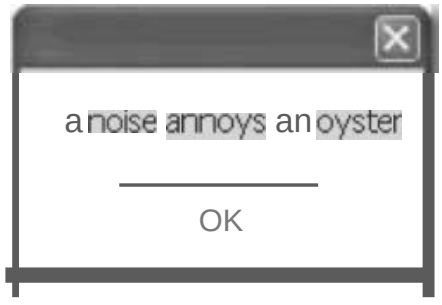
Was your **code** a little **different** than **ours? There's more** than one **way** to solve any **programming problem—like** you could have **used while loops** instead **of for loops. If** your **program works, then** you got **the** exercise **right!**

Pool puzzle



Your job is to take code snippets from the pool and place them into the blank lines in the code. You may not use the same snippet more than once, and you won't need to use all the snippets. Your **goal** is to make a class that will compile and run. Don't be fooled—this one's harder than it looks.

Output



We included these "Pool Puzzle" exercises throughout the book to give your brain an extra-tough workout. If you're the kind of person who loves twisty little logic puzzles, then you'll love this one. If you're not, give it a shot anyway—but don't be afraid to look at the answer to figure out what's going on. And if you're stumped by a pool puzzle, definitely move on.

```
using System;
using System.Windows.Forms;
namespace Chapter2 {
  class Chapter2PoolPuzzle {
    public static void Main() {
      int x = 0;
      String Poem = "";

      while ( ____ ) {

        ____________________
        if(x < 1) {
        ____________________

        if ( ____ ) {

        ____________________

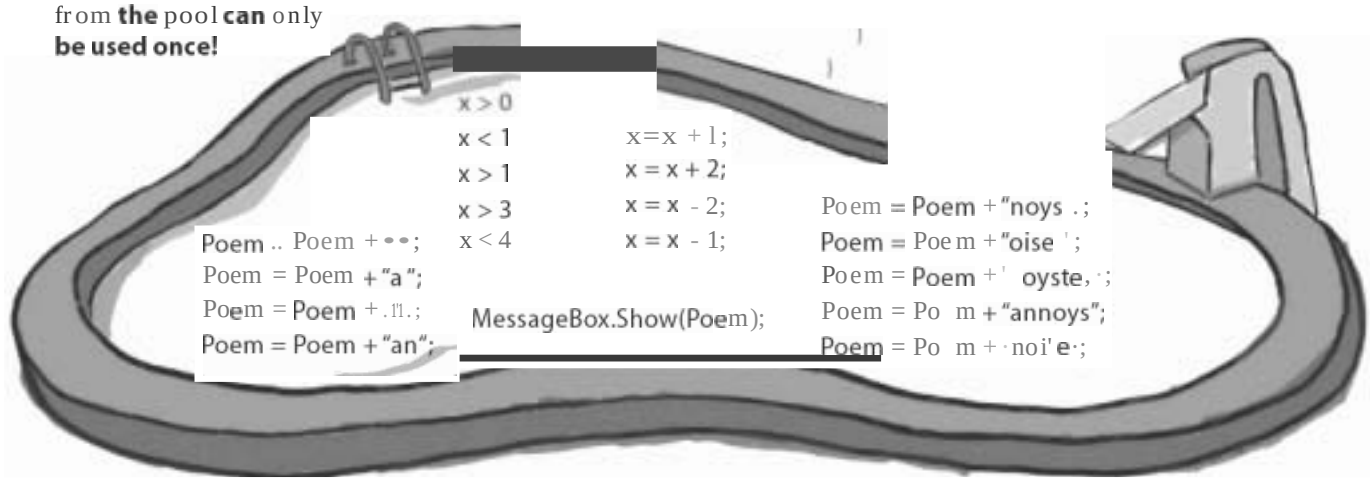
        ____________
        }
        if ( x == 1 ) {

        ____________________

        ____ ) {

        ____________________
```

Note: **each snippet from the pool can only be used once!**

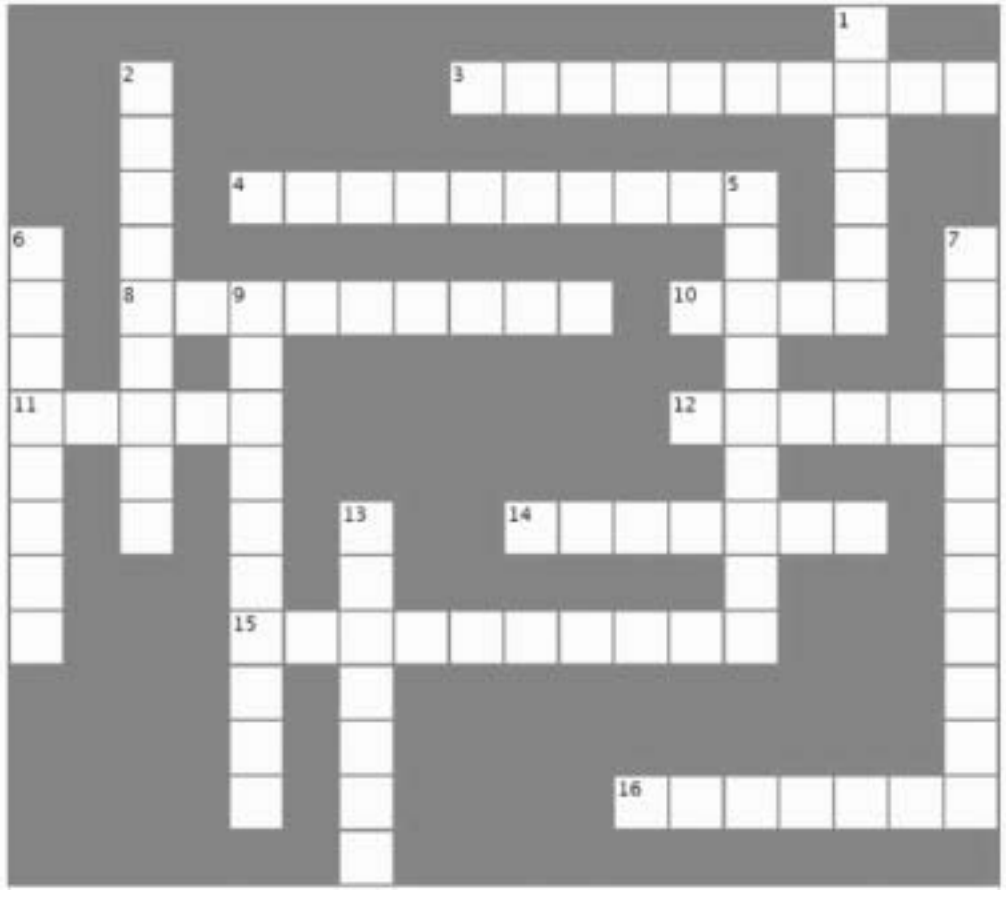


Answers on page 83.



csharpcross

How does a crossword help you learn C#? Well, all the words are C#-related and from this chapter. The clues also provide mental twists and turns that will help you burn alternative routes to C# right into your brain

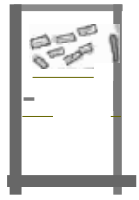


Across

3. You give information to a method using these
4. buttonIText and checkBox3.Name are examples of
8. Every statement ends with one of these
10. The name of every C# program's entry point
11. Contains methods
12. Your statements live here
14. A kind of variable that's either true or false
15. A special method that tells your program where to start
16. This kind of class spans multiple files

Down

1. The output of a method is its value
2. SyslemWindows.Forms is an example of ore of these
5. A tiny piece of a program that does something
6. A block of code is surrounded by
7. The kind of test that tells a loop when to end
9. You can call Shown to pop up a simple Windows dialog box
13. The kind of variable that contains a whole number



COde Magnets Solution

Part of a C# program is all scrambled up on the fridge. Can you rearrange the code snippets to make a working C# program that produces the message box? Some of the curly braces fell on the floor and they were too small to pick up, so feel free to add as many of those as you need!

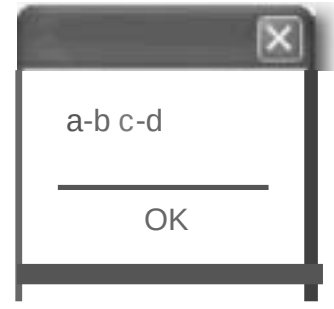
```
string Result = "";
int x = 3;
while (x > 0)
{
    if (x > 2)
    {
        Result = Result + "a";
    }
    x = x - 1;
    Result = Result + "-";
    if (x == 2)
    {
        Result = Result + "b c";
    }
    if (x >= 1)
    {
        Result = Result + "d";
        x = x - 1;
    }
}
MessageBox.Show(Result);
```

This magnet didn't fall off the fridge...

The first time through the loop, x is equal to 3 so this conditional test will be true.

This statement makes x equal to 2 the first time through the loop, and 1 the second time through.

Output:



Pool Puzzle Solution



Your job was to take code snippets from the pool and place them into the blank lines in the code. Your goal was to make a class that will compile and run.

```
using System;
using System.Windows.Forms;
namespace Chapter_2 {
  class Chapter2PoolPuzzle {
    public static void Main()
      int x = 0;
      String Poem = "";

      while ( x < 4 ) {

        Poem = Poem + "a";
        if ( x < 1 ) {
          Poem = Poem + " ";
        }
        Poem = Poem + "\n";

        if ( x > 1 ) {

          Poem = Poem + " oyster";

          x = x + 2;
        }
        if ( x == 1 ) {

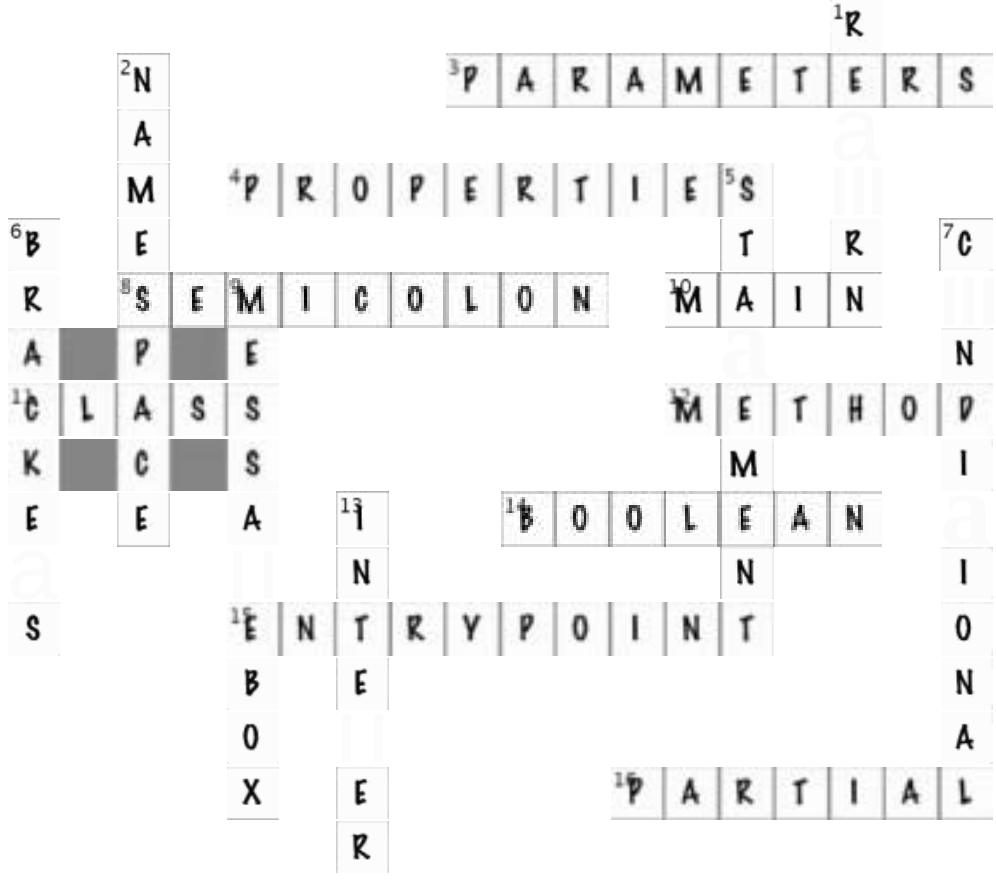
          Poem = Poem + "oys ";
        }
        if ( x != 3 ) {

          Poem = Poem + "oise ";

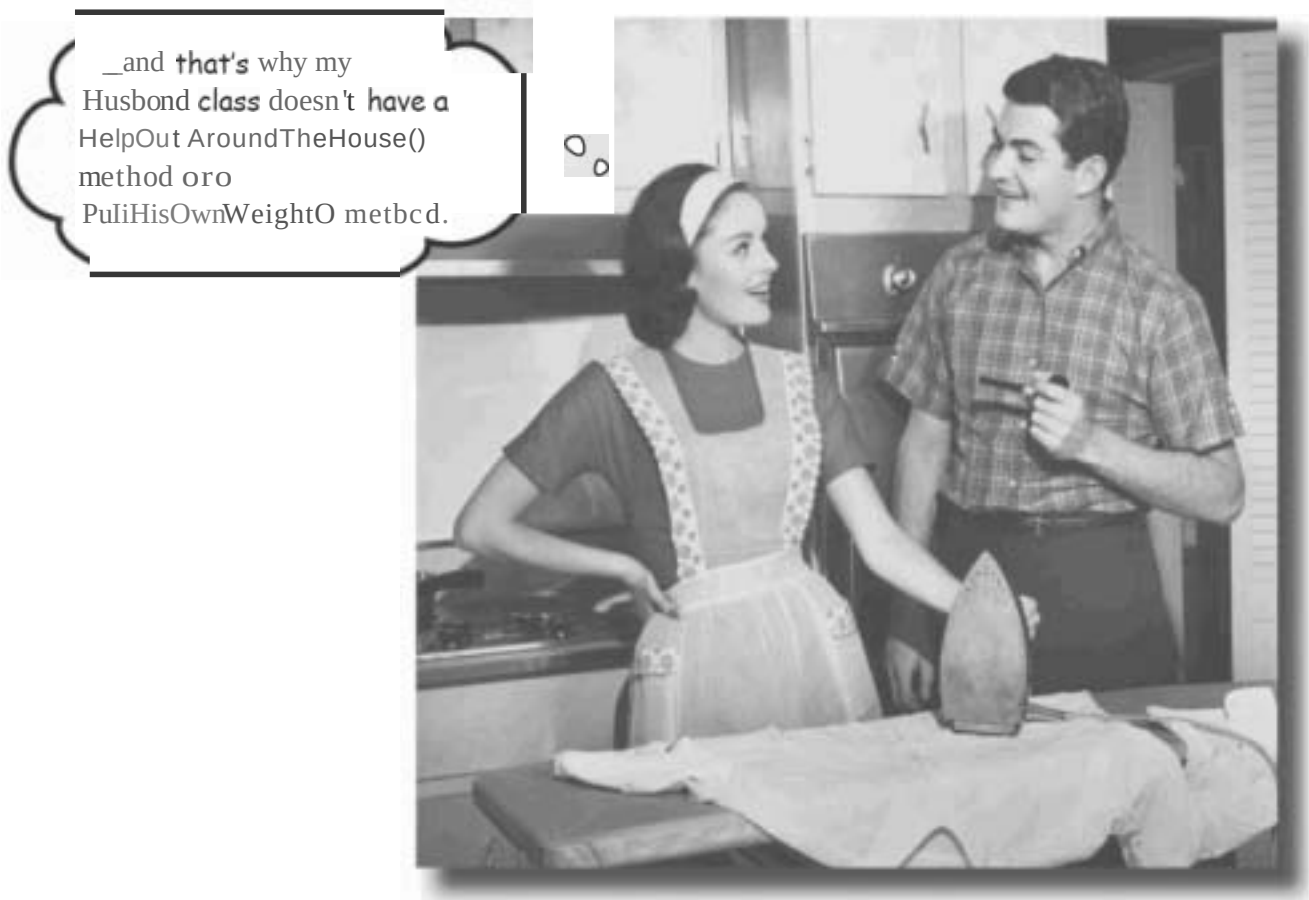

        x = x + 1;
      }
      MessageBox.Show(Poem);
    }
  }
```



csharpcrossSolution



Making Code Make Sense



Every program you write solves a problem.

When you're building a program, it's always a good idea to start by thinking about what *problem* your orocrem's supposed to solve. That's why ***objects*** are really useful. They let you structure your code based on the problem it's solving, so that you can spend your time *thinking about the problem* you need to work on ralher than gelling bogged down in the mechanics of writing code. When you use objects righi, you end up with code that's *intUitive* 10 write, and easy \0 read and change

How Mike thil1ks about his problellls

Mike's a programmer about to head ,"II 10ajoh intvrvicv; He can't wait to show otl hi, C# skills, but lir't he has to get tlıere—and he's running late!

Mike figures out the route he'll take to get to the interview.



I'll take the 31st Street bridge, head up Liberty Avenue, and go through Bloomfield.

Mike sets his destination, then comes up with a route.

Good thing he had his radio on. There's a huge traffic jam that'll make him late!

Mike gets new information about a street he needs to avoid

This is Frank Loudly with your eye-in-the-sky shodow traffic report. It looks like a three-car pileup on Liberty has traffic backed up all the way to 32nd Street.



Mike comes up with a new route to get to his interview on time.

No... he can come up with a ne... route to the interview

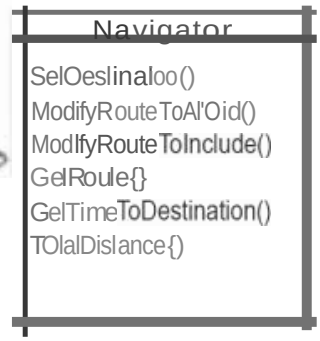
No problem. If I take Route 28 instead, I'll still be on time!



How Mike's ear navigation **system thinks** about **his problems**

Mike huill his own (;PS navigati,'ll system, which he uses Ito help him get around 10\11).

Here's a diagram of a class in Mike's program. It shows the name on top, and the methods on the bottom.




```
Set.Deati nati on ("Fi f t h Ave & Penn Ave");
string route:
route - GetRoute();
```

Here's the output from the GetRoute() ...ethod—it's a string that contains the directions Mike should follow.

The navigation syste... .ets a destination and comes up with a route

"Take 31St Street Bridge to Li berty Avenue to Bloomfield"



The navigation system gets new information about a street it needs to avoid.

```
ModifyRouteToAvoid("Liberty Ave");
```

Now it can come up with a new route to the destination.

```
string route:
route = GetRoute II ;
```



"Take Route 28 to the Highland Park Bridge to Washington Blvd"

GetRoute() gives a new route that doesn't include the street Mike wants to avoid

Mike's navigation **system** solves **the** street navigation **problem the same way he does**

Mike's Navigator class has **methods** to set altd **modify** routes

Mike's Navigator class has methods, which are where the a"lion hill'p"ns, But unlike the button_Click() methods in rhe forms you've built, they're all focused "",uud it single problem: navigating a route through a city. That's why Mike stuck them together inlo 0'u' class, and called that class Navigdtor.

Mike designed his Navigator class so Ihal ;1', easy to create and modify routes. To get it route, Mike's program calls the SetDestination() method I" set the desl;lla,;''ll. and Ihnl uses Ih,- Get Route() method to put the route inl" a string. If he needs In change the route', hi, program calls Ihe ModifyRouteToAvoid() method 10 change the route so Ihal it avoids it certain street, and then calls the Get Route() method to get I]ll' new directions.

Mike chose method names that would make sense to someone who was thinking about how to navigate a route through a city.

```
public class Navigator() {

    public void SetDestination(string destinationName) {      };

    public void ModifyRouteToAvoid(string streetName) { ... };

    public string GetRoute() { ... };
```

This is the **return type** of the method. It means that the statement calling the GetRoute() method can use it to set a string variable that will contain the directions. When it's **void**, that means the method doesn't return anything.

```
string route;
route = GetRoute();
```

Some methods have a return value

Every llll'llll ,,1is '', ad,' "I' or statements that do things. Some "" 'lh,,,], just execute their statements and Ih"11 exit. But other nll'lh,,,I, have a r..lurn **value**, or a '-ill","' that's calculated or generated inside Ih,' method, and sent back 10 Ihe statement that called thar melhod. The type of the rllIrn value (like string or int) is called the r..lurn **typ**..,

Here's an example of a ..ethod that has a return type—it returns an int, The ..ethod uses the two para..eters to calculate the result and uses the **return** state..ent to pass the value back to the state..ent that called it.

The return statement tells Ihe method 10 i111tll,>diately exit. If yc,ur method doesn'I have a r,'turn value—which means it's declared with a return type of **void** thr'n the return statement just ends with a "'111 icolon, and you don't alwavs have 10 have one in your ll wlhod. Btll if the method has a return type, then it lllU,1 use the return statement.

```
pubjIc int MUltiplyTwoNumbers(int firstNumber, int secondNurnber) {
        int result = firstNumber * secondNumber;
        return resUlt;
```

Here's a statement th.u calls a 1l1<'IIH"lw multiply two numbers. It returns an int:

```
int myResult · MultiplyTwoNumbers(3, 5);
```

Methods can take values like 3 and 5. But you can also pass variables to them.

Use **what** you've learned to build a **simple** application

Let's hook up a form to a class, and make its button call a method inside that class.



1. Create a new Windows Application project in the IDE. Then add a class file to the project called `Talker.cs` by right-clicking on the project in the Solution Explorer and selecting "Class..." from the Add menu. When you name your new class file "Talker.cs", the IDE will automatically name the class in the new file Talker. This will open up a new file in the IDE.

2. Add `using System.Windows.Forms;` to the top of the class file, then add code to the class:

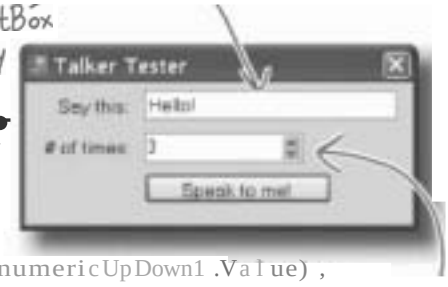
```
class Talker {
    public static int BlahBlahBlah(string thingToSay, int numberOfTimes)
    {
        string finalString = "";
        for (int count = 1; count <= numberOfTimes; count++) {
            finalString = finalString + thingToSay + "\n";
        }
        MessageBox.Show(finalString);
        return finalString.Length;
    }
}
```

This statement declares a finalString variable and sets it equal to an empty string.

The BlahBlahBlah() method's return value is an integer that has the total length of the message it displayed. You can add ".Length" to any string to figure out how long it is.

The new class has one method called `BlahBlahBlah()` that takes two parameters. The first parameter is a string that tells it something to say, and the second is the number of times to say it. When it's called, it pops up a MessageBox with the message repeated a number of times. Its return value is the length of the string. The method needs a string for its thingToSay parameter and a number for its numberOfTimes parameter. We'll get those parameters from a form that will ask the user for the text using a TextBox and a number using a NumericUpDown control.

Set the default text of the TextBox to "Hello!" using its Text property



3. Add this form to your project.

4. Then double-click on the button and have it run this code:

```
int len = Talker.BlahBlahBlah(textBox1.Text, (int) numericUpDown1.Value);
MessageBox.Show("The message length is " + len);
```

This is a NumericUpDown control. Set its Minimum property to 1, its Maximum property to 10, and its Value property to 3

5. Now run your program! Click the button and watch it pop up two message boxes. The class pops up the first message box, and then the form pops up the second one.

The BlahBlahBlah() method pops up this message box based on what's in its parameters.



When the method returns a value, the form pops it up in this message box.



Mike gets an idea

The interview went great! Ilul the traffic jam this morning got Mike thinking about h",,' he could improve his navigator.

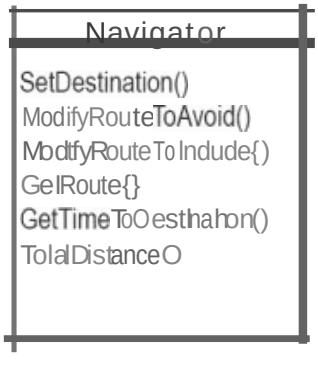


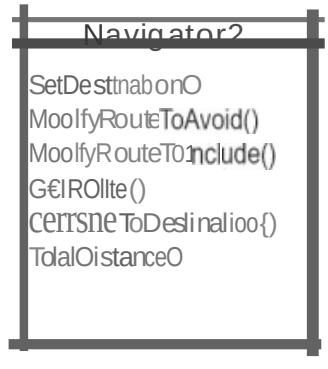
It'd be great if I could compare (i few r-outes and figure out which is fastest..

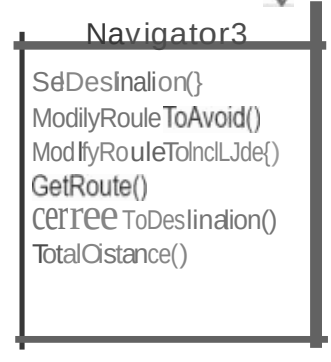
He could create three different Navigator classes...

Mike could copy the Navigator class code and pa'lt- it illt" two "",,',' classes. Then his program ,""ld store Ihn-,- route .u once.

This box is a class diagra... It lists all of the methods in a class, and it's an easy way to see everything that it does at a glance







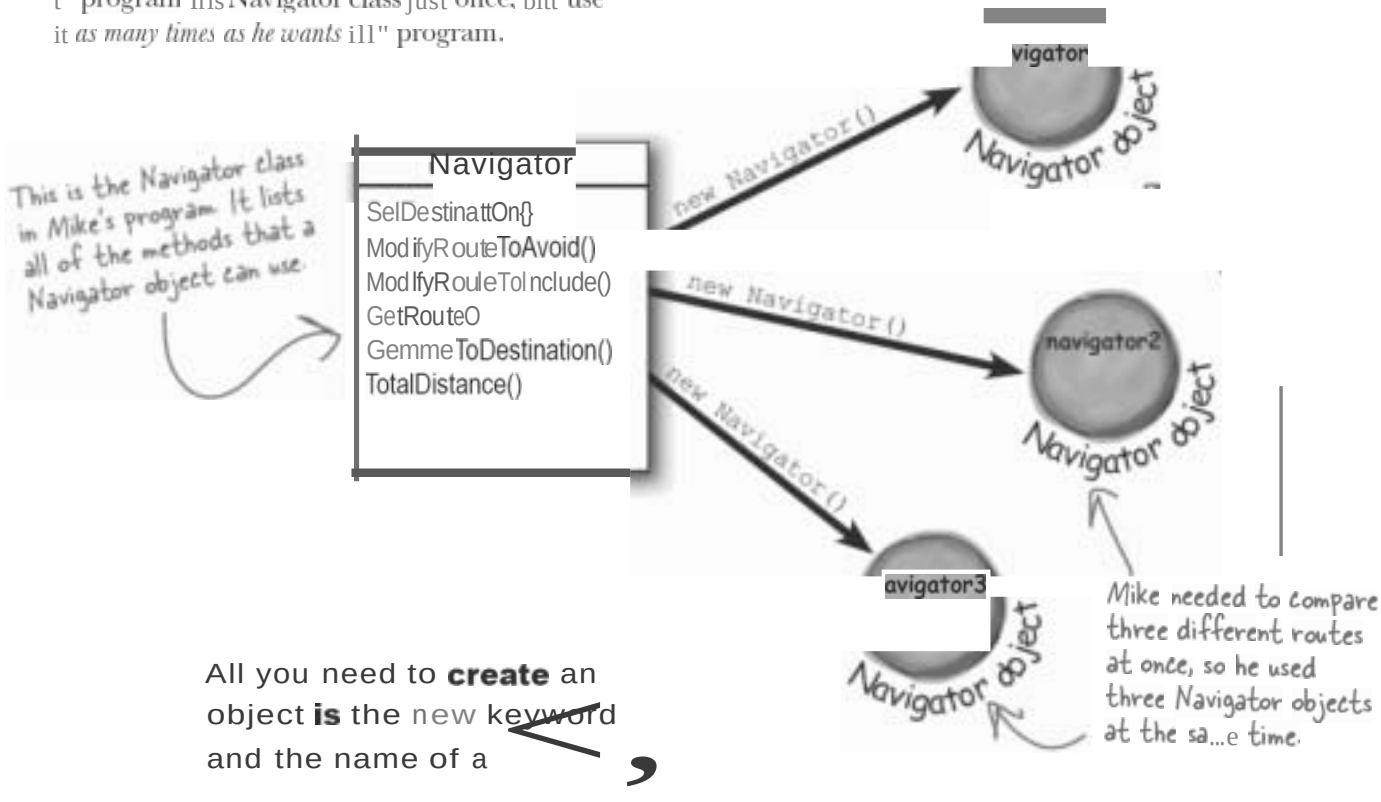
Whoa, that can't be right! What if I want to change 0 method? Then I need ta go back and fix it in three places.



Right! Maintaining three copies of the **same** code **is** really **messy**. A 1D" of problem you ",-,-,1 10solve neecIa wav 10 represent one *I it* ***ing*** a bunch of different times. In Ihi, case, it's alnmch "r routes. But il could be a hUlldl of turbines 0)' dogs, "'- mu,i,' files, or anything. All of those programs have one thing in common: they always n,',-d I" treat the same kind or thing in tl,,- same way, no malk-r how many "I' the thing they're dealing with.

Mike call use **objects** to solve **his problem**

Objects are C#'s tool that you use to work with a bunch "l' similar things. Mike can use objects t" program Iris Navigator class just once, bitt use it *as many times as he wants* ill" program.



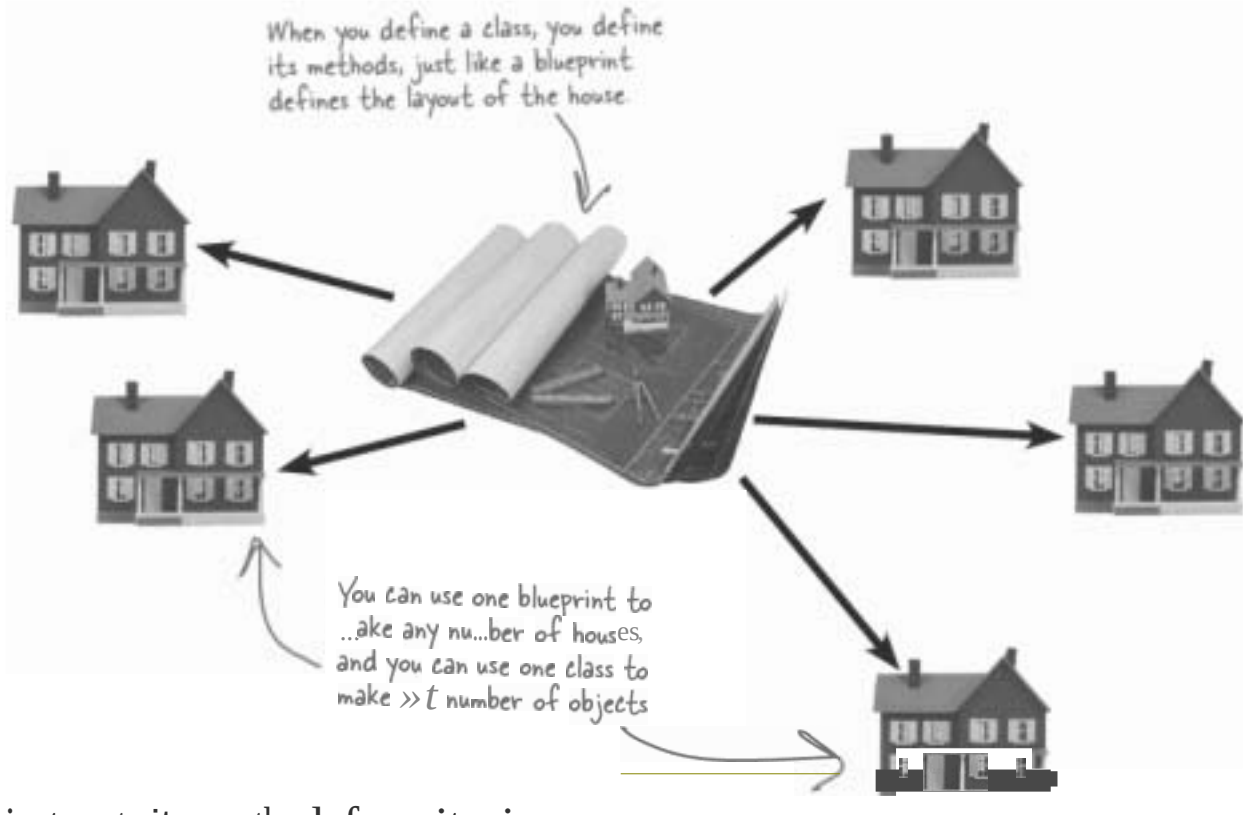
All you need to **create** an object **is** the new keyword and the name of a

```
Navigator navigatorl =(new)Navigator();
navigatorl.SetDestination("Fifth Ave & Penn Ave");
string route;
route =navigator1.GetRoute();
```

Now you can **use** the objectl When you create an object from a **class,** that object has all of the methods from that class.

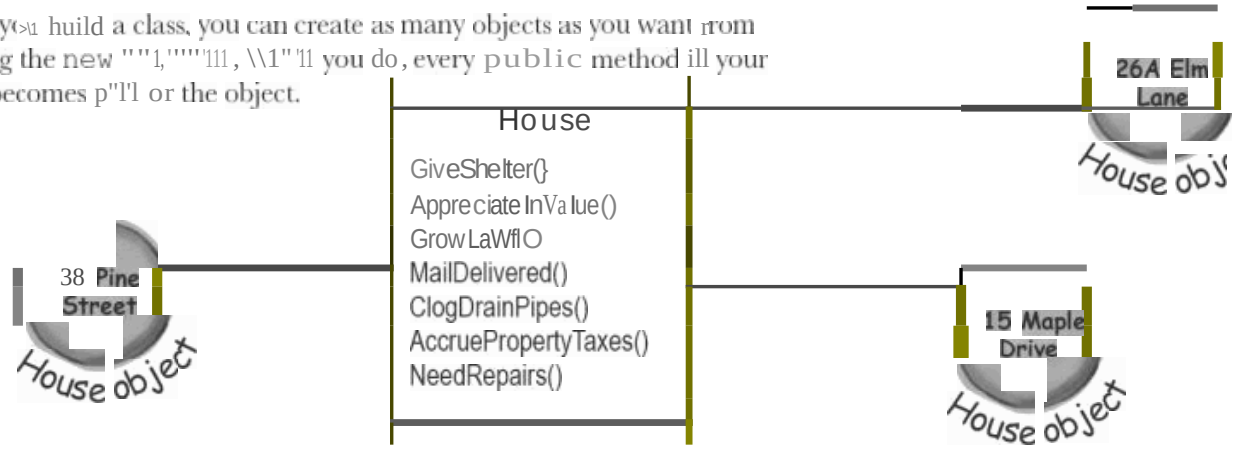
You use a class to build an object

A class is like a blueprint for an ohi""!. If you wanted ,,, build five identical houses in a suburban housing dr-vr-lopnu-nt. you wouldn't ask an architect '0 draw lip five identical sets or blueprints. You'd ju', use " uc' Blueprint '" huild five houses.



A" object get. its method. from it. ciass

Once yo>1 huild a class, you can create as many objects as you want rrom i, using the new ""1,"""'111, \\1"'11 you do, every public method ill your class becomes p"l'l or the object.



Whell you create a Ilew object frolll a **class,** **it's** called **an** illstallce of that class.

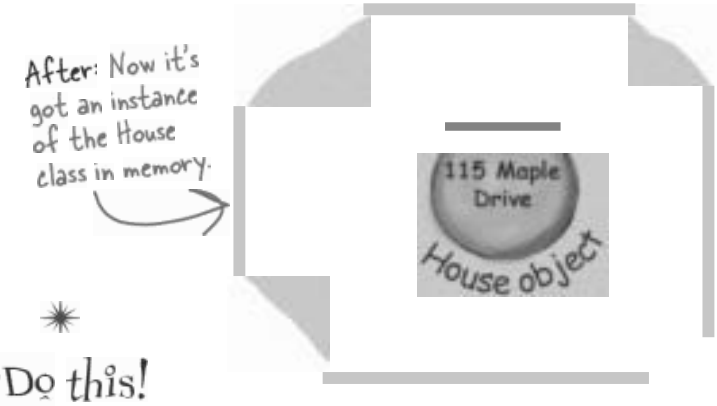
Guess what... you already know lhis stuff! Everything ill the toolbox j, a class: there's a Butt on class, a TextBox class, a La b e l class, etc. When you drag a button "ul ‹0r the toolbox, the IDE aut(unaricallv creates an instance of the Button class and calls it butt on1, \When you drag "n, »ther 1nu« ,n "ttl "f the I'" ,n" .x. il creates another instance ,'all,rc1butt on2. E,,,'h instance or But t on has its own properties and """lh, »ds. Blll every hun ,"' acts exactly lh,' same way, 1" ,cause thev're "Uinstances of the same class.



Before Here's a picture of your computer's memory when r-r program starts

Your program executes a new statement

```
House 11SMapleDrive = new House();
```



Check it out for yourself!

Do this!

()p"" any I'",j"", lhal uses " button ("all,'d butt on 1, and use lh,' IDE to search the entire project for the text "**new button**l·'. You'll lind the code thar the- IHE added to the form designer to create the instance of the Button class.

in-stance, noun.

all example or Oll Coccurrence of something. *The IDE search-and-reptace frntnre finds crerv* ***instance*** *of a n-ord and changes it to another.*

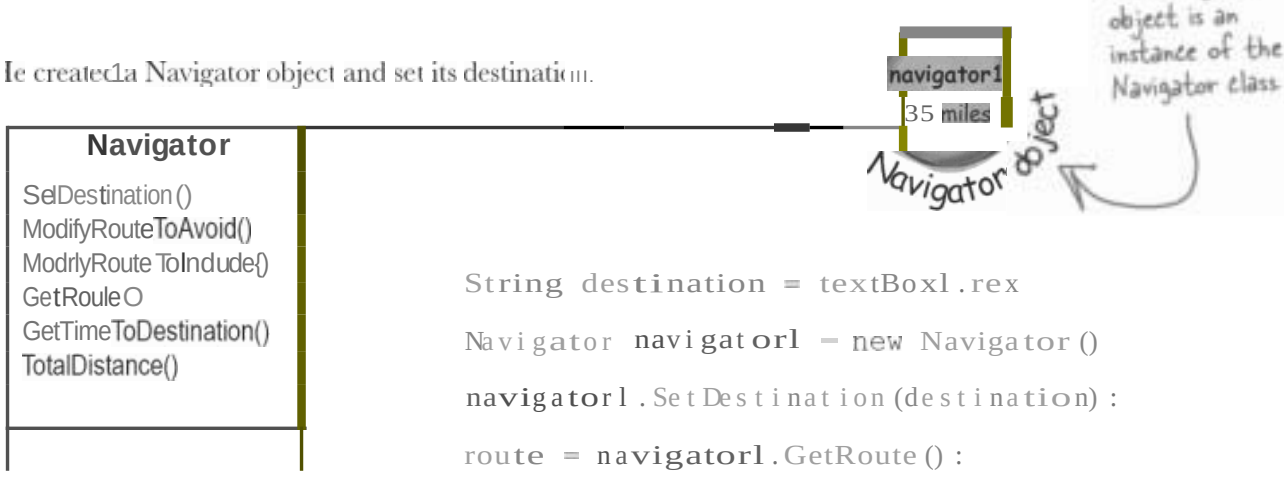
A better solution... brought to you by objects!

Mike came up with a new route comparison program that uses objects to find the shortest of three different routes to the same destination. Here's how he built his program.

GUI stands for Graphical User Interface, which is what you're building when you make a form in the form designer.

- Mike set up a GUI with a textBox1 that contains the destination for the three routes. The GUI has added textBox2, which has a street that one of the routes should **avoid**; and textBox3, which contains a different street that the third route has to include.

- He created a Navigator object and set its destination.




```
String destination = textBox1.text;
Navigator navigator1 = new Navigator();
navigator1.SetDestination(destination);
route = navigator1.GetRoute();
```

- Then he added a second Navigator object called navigator2. He called its SetDestination() method to set the destination, and then he called its ModifyRouteToAvoid() method.

The SetDestination(), ModifyRouteToAvoid() and ModifyRouteToInclude() methods all take a string as a parameter.

- The third Navigator object is called navigator3. Mike set its destination, and then called its ModifyRouteToInclude() method.



- Now Mike can call each object's TotalDistance() method to figure out which route is the shortest. And he only had to write the code once, not three times!

Any **time** you **create a** new **object** from **a class, it's called creating an** instance **of** that class,

Sharpen your pencil

Follow the same steps that Mike followed on the facing page to w.ite the code to create Navigator objects and call thei, method,.

```
String destination = textBox1.Text;
String route2StreetToAvoid = textBox2.Text;
String route3StreetToInclude = textBox3.Text;
```

We gave you a head start. Here's the code Mike wrote to get the destination and street na···es from the textboxes.

```
Navigator navigator1 = new Navigator();
navigator1.SetDestination(destination);
int distance1 = navigator1.TotalDistance();
```

And here's the code to create the navigator object, set its destination, and get the distance

r- -t

1. (reate the navlgatot2 obj ect. set it, deslination. call its Mod ifyRouteToAvold(l method. and use its **TotalDistance()** method to se1 an integer variable called distance2_

```
Navigator navigator2 =

navigator2.

navigator2.
```

r- -t

2. (reate the navlgat o.3 object, set it, de,tination, call its ModifyRouteTolndudeO method, and use its TotatDistancen method to set an integer va.iabte called distance1.

L ⌟

The built-in C# Math.MinO ...ethod compares two numbers and returns the smallest one. Mike used it to find the shortest distance to the destination.

```
int shortestDistance = Math.Min{distance1, Math.Min(distance2, distance3});
```

Sharpen your pencil Solution

Follow the same steps that Mike followed on the facing page to write the code to create Navigator objects and call thei, method,.

```
String destination = textBoxl.Text;
String route2StreetToAvoid = textBox2.Text;
String route3StreetTolnclude = textBox3.Text;

Navigator navigatorl - new Navigator()
navigatorl.SetDestination(destination);
int distancel - navigator!.TotalDistance();
```

We gave you a head start. Here's the code to get the destination and street names, along with the code to create the first Navigator object, set its route, and get the distance.

1, Create the navigatot2 object, set its des'ination. call its ModifyRouteToAvold(l method, and use its **TotalDistance()** method to se, an integer varable called **distance2**.

```
Navigator navigator2 = new Navigator()
navigator2.SetDestination(destination);
navigator2.ModifyRouteToAvoid(route2StreetToAvoid);
navigator2.TotalDistance();
```

2. Create the navigator3 object. se, i', de, tination. call its ModifyRouteTolnd udeOmethod, and use its **TotalDistance()** method to set an integer varable called dlstance3,

```
Navigator navigator3 = new Navigator()
navigator3.SetDestination(destination);
navigator3.ModifyRouteTolnclude(route3StreetTolnclude);
int distance3 = navigator3.TotalDistance();
```

The built-in C# Math.Min() ..ethod compa..es two numbers and ..eturns the smallest one Mike used it to find the shortest distance to the destination.

```
int shortestDistance = Math.Min(distancel, Math.Min(distance2, distance3));
```

Hold itl r ye written a few classes now, but I haven't used "new· to create an instance yet! 50 does that means I con call methods wit hout creating objects?

C 0



Yesl That's why you used the `static` keyword in your methods.

Take another l,.,k at ,l,,· declaration 1'",. the `Talker` class \ou1built a few pages ago:

```
class Talker

  pUblic static int BlahBlahBlah(String thingToSay, int numberOfTimes)
  '
    string finalString - "";
```

\\"1"11 you called the "ıe,hod you did" '\ create a `new` instance or Talker. You just did this:

```
Talker.BlahBlahBlah("Hello hello hello", 5);
```

That's how you call `static` methods, and you've been doing that all along. If you take away tht" `static` keyword from the `BlahBlahBlah()` method declaration, then you'll have to create an instance of 'Talker() in order r" call the meth◈d, ()rher than that (li'lincti,ui. static "ıeIhod, are just like object methods. You can pass parameters, they c:'" return values, and thev live in classes.

There's one more thing you can do with the `static` keyword. You ,"all mark your **whole ciass** as static, and then all "r its methods **must** h." `static` too. Ir you try to add a non-static method ro a static class. ir won't compile.

there are no Dumb Questions

Q: When I Ihink or somelhing **that's** "static", I Ihink of something **that doesn't** change. ueee Ihal **mean non-static methods** can change, but **static** methods don"t? Do they behave **differently?**

A: No, **both** stanc and non-static methods act **exactly the same**. The only **difference** rs that static methods don't require an instance, while not-state methods do A lot of people have trouble remembering that. because the word "static" isn't really all that intuitive.

0_: So I een't **use** my **class** until I **create** en instance 01 an object?

A: You **can use** lis state methods. But if you have methods that aren't static, then you need an instance bercre you can use them

Q_: Then why would I want a method that **needs** an **instance?** Why woutdn"t I **make all** my **methods** $talic?

A: Because if you have anobject that's keeping track of certain data-like Mike's Instances of IIIS Navigator class that each kepi track of a different route-then you ca nuse each nsterces methods to work with that data. So when Mike called his ModifyRouteToAvoid() method in the navigator2 instance, it only affected the route that was stored in that partcular instance It didn't affectthe navigator1 or navigator3 ccects That's how he was able to work With three d,fferent routes at the same time—and his program courd keep track of all of it

0_: So how **does** an **ins**lance keep track 01data?

A: Turn the page and find out!

An instance uses fields to keep track of thillgs

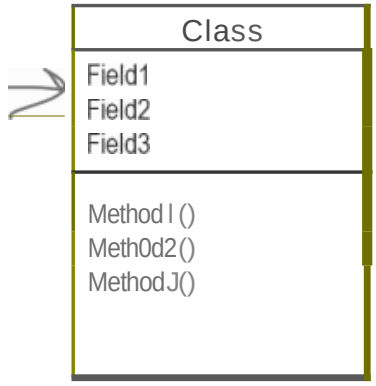
y,,,, change the text 011 a b"lh'll by setting its `Text` property in the IDE. When you do, the IDE adds code lik,'rhi, ro the designer:

```
buttonl.Text - "rext for the button";
```

Now you know rluu `buttonl` is an instance or the `Button` class. What that code does is modify a fi..ld for the `button1` instance. You can add fields to a class diagram - jU'1 draw a horizotal line in the middle of it. Fields go above the lille, methods g' underneath it.

Technically, it's setting a property. A property i, a special kind of field—but we'll get into all that a little later on

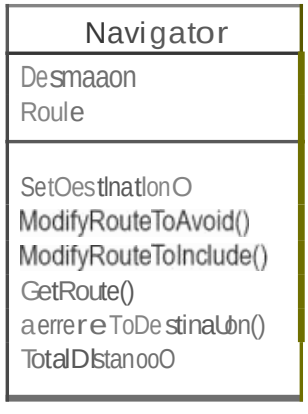
This is where a class diagram shows the fields. Every instance of the class uses the... to keep track of its state



Add this line to separate the fields f..om the ...ethods

Methods are what a" object does. Fields are what the object k"ows.

\,'h,'n Mike created Ih,.." , instance "l" `Navigator` classes, hi, program created three objects. Each or those objects was used to keep track "l' a different route. When rhe program created rh,- `navigator2` instance and called it, `Set.Destination()` nn-thc"l. irset the destination f'1' that one instance. But il didn'r affect the `nevLqator1` instance "r the `navigatorJ` instance.



Every instance of Navigator knows its destination and its route

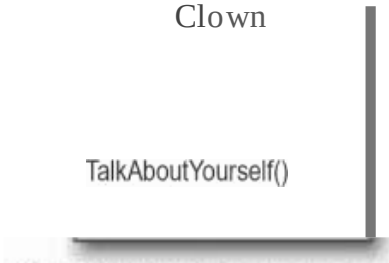
What a Navigator object does is let you set a destination, modify its route, and get information about that route.

An **object's behavior** is **defined by** its **methods, and** il uses **fields** 10 **keeP** trac**k of** its sIale.

Let's create sOllie lnstaneesl

Re..e..be.., when r- see "void" in f..ont of a method, it ..eans that it doesn't etu.. any value

```
public class Clown
  public String Name;
  public lnt Height;
```



When you want to create instances of your class, **don't use** the static keyword in either the class declaration or the method declaration.

Remember, the *= operator tells C# to take whatever's on the left of the operator and multiply it by whatever's on the right

Sharpen your pencil

Writ.. dowrllh.. conlent, of each m..ssage box that will be displayed afte, the ,tatement next to it i, "xl'cut..d,

```
Clown oneClown - new Clown{);
oneClown.Name = "Boffo";
oneClown.Height = 14;
oneC!own.TalkAboutYourself();
```

"My name i. and 1m inch..s tall."

```
Clown anotherClown = new Clown();
anotherClown.Name - "Biff";
anotherC!own.Height - 16;
anotherClown,TalkAboutYourself();
```

"My nam.. i, and I'm inches tall:

```
Clown clown3 - new Clown!};
clQwn3.Name - anotherClown.Name;
clown3.Height - oneClown.Height  3;
clown3.TalkAbout Yourself();
```

"My nam.. i, and I'm inches tall"

```
anotherClown.Height *= 2;
anotherClown.TalkAboutYourself();
```

"My nam.. i, and I'm inches tall:

Thanks for the l\Iel\lory

" 'lwil your program creates an object, it lives ill a part Of the computer' me-nu ory called the heap_When y'U' code creates an object with a new statement, C# immediately n"",-","' space in the heap '0 il "'lll,t"n' lhe data for that object.

Here's a picture of the heap before the project starts Notice that it's e..pty



Let's take a closer look at what bappened here

Sharpen your pencil Solution

Write down the contentl of each message box th at will be displayed after the statement nexlto it i, executed,

```
Clown oneClown = new Clown();
oneClown.Name = "Boffo";
oneClown.Height = 14;
```

Each of these new statements creates an instance of the Clown class by reserving a chunk of memory on the heap for that object and filling it up with the object's data.

```
oneClown.TalkAboutYourself();
```

"My name is Boffo and I'm 14 inches tall."

```
Clown anotherClown = new Clown();
enotnerctovn.xeee - "Bi
anotherClown.Height = 16;
anotherClown,TalkAboutYourself();
```

"My name is Biff and I'm 16 inches tall."

```
Clown clown3 =  ew Clown!};
clown3.Name = anotherClown.Name;
clown3.Height - oneClown.Height   3;
clown3.TalkAboutYourself();
```

"My name is Biff and I'm 11 inches tall."

```
anotherClown.Height *= 2;
anotherClown.TalkAboutYourself();
```

"My name is Biff and I'm 32 inchel tall."

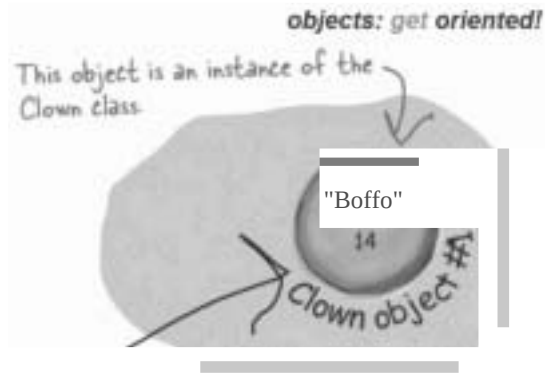
When you.' program creates a new object, it gels added 10 the lreap.

What's on your program's mind

Here's how your program creates a new instance of the Clown class:

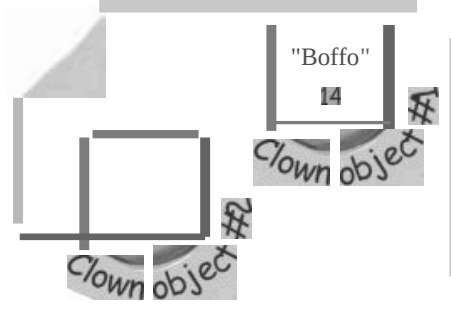
```
Clown myInstance = new Clown();
```

That's actually two statements crammed into one. The first statement declares a variable of type Clown (`Clown myInstance;`). The second statement creates a new object and assigns it to the variable that was just created (`myInstance = new Clown();`). Here's what the heap looks like after each of these statements:



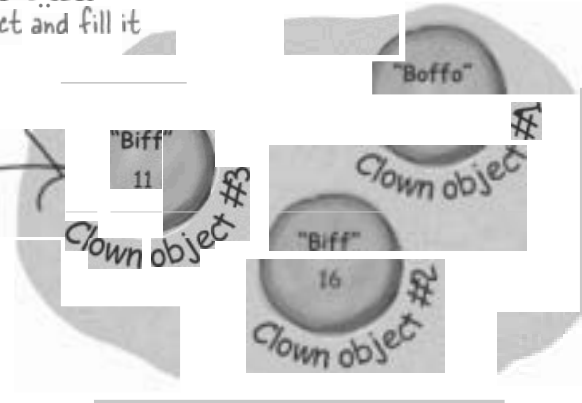
1
```
Clown oneClown = new Clown();
oneClown.Name = "Boffo";
oneClown.Height = 14;
oneClown.TalkAboutYourself();
```

The first object is created, and its fields are set.



2
```
Clown anotherClown = new Clown();
anotherClown.Name = "Biff";
anotherClown.Height = 16;
anotherClown.TalkAboutYourself();
```

These statements create the second object and fill it with data.

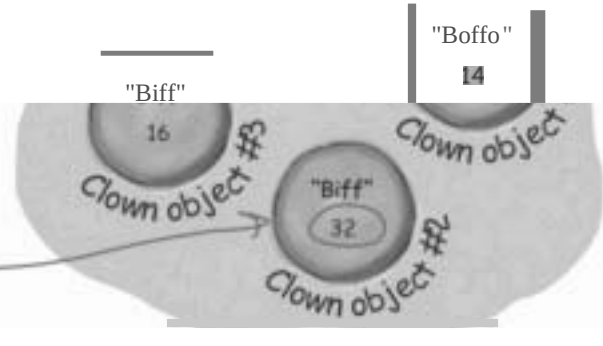


3
```
Clown clownJ = new Clown();
clown3.Name = anotherClown.Name;
clownJ.Height = oneClown.Height - 3;
clown3.TalkAboutYourself();
```

Then the third Clown object is created and populated.

4
```
anotherClown.Height *= 2;
anotherClown.TalkAboutYourself();
```

There's no `new` command, which means these statements don't create a new object. They're just modifying one that's already in memory.



You call use class aad **method names** to **make** your code illtuitive

When you 1'"1 code in a method, you're making a choice ahoUI how 10 structure yonn' program. Do yc'IIuse "Il" method? I)" yc,\I 'plil it inn Ill're than one? Or ,10 y 'u ,'\"'11 """d a nll'll" "I at "II.' The choice yc.u make abc'UI methods can m"k,' yc'Ill code mUl'h n"'H' intuitive—or, if vou're ""I careful, much tIlon' ,'oll\'lll"ll'd.

O Here's a nice, compact chunk of code. It's frc"n a control program that run" , rn,« hun- Ih,u 111 ,k,-, ",nd) bars

The chkTemp() method -eturns an ,nteger but what does it do?

```
int t _ m chkTemp();
if (t >1601]
    T obj = new T();
    tb.clsTrpV(2);
    ics.Fiìl();
    ics.Vent();
    m.airsyschk();
```

"obj", "ics", and "..." are terrible na...es! We have no idea what they do. And what's that T() class for?

The t1.r .,.VO ..ethod has one parameter, but we don't know what it's supposed to be

Take a second and look at that code. Can you figure out what it does?

O Th",se statements dll"·1 give you any hints about why the code's doing what it's doing I" Ihi, case, the programmer was happy with the results because she was able IIIget il all into one method. But making your code as compact as possible isn't really useful! Let's break ir UI) into m'·lhlld, III make it ,';"i,'!, 10 read. and Illak,' sure the classes are given names that make sense. But we'll start bv figuring out what the c(„1.. i, "'PI"»sed 1> do.

How do you figure out what your code is supposed to do? Well, all code is written for a reason. So it's up to you to figure out that reason. In this case, we can look up the page in the specification manual that the programmer followed.

General **Electr**on;c**s Type** 5 Candy Bar **Maker Specification Manual**

'The nougat temperature must be checked ,-, ery 3 minute by an automated system. If the temperature **exce..ds** 160°C, Ihe candy is toc hot, and lh,' system rnu'l **perform the candy** ;.,ol;>.l;on **cooling system** (CICS) **vent procedure**.

Close the lrip throttle valve "n turbine #2

Fill Ih,· i,,,la,jlln cooling system with a solid stream "I' water

Vent the water

Verifv Ih,n there is "" ,",'i,k,ıce "r air ill Ihe system

3 Tha page from the manual made it a 101 easier to understand the code. It also gave '" some great hints ah, uu how t" make- ollt" ,',ode easier to understand. Now we km»w why the c«,ndi ti,»nal te-st checks the variable t against 160 - the manual say that any temperature al»ove 160°C means the Il'»ugat is t,»o hot. .\l1d it turns "111 that "m" was a class that "Olltrolled th« candy milk"t, " ith stati,-1Il,'tlto,J, to ("h","k the nougat temperature and d",,'k th,' air system. So let's put the temperature check il1to a method, and choose name for Ih,' class and the methods Ihat ma k,' the purpose obvious.

The IsNougatTooHot() ..ethod's return type

```
public boolean IsNougatTooHot{} {
    int temp = Maker.CheckNougatTemperature();
    if (temp> 160)
      return true;
      else {
      return false;
```

By na'"ing the class "Maker" and the ..ethod "CheckNougatTe..perature" the eede is a lot eaoe- to understand

This method's retu..n type is boolean, which ..eans it ..eturns a true or false value

O \\'hal does the specification sav to d" if the nougat is too hot? It tells ll' to perform th,. candy isolation cooling system (or CICS) vent I't<"',,dult,. So let's lIlak,' another method, and choose all "Iwi"u, nan,,' fi»r the "T" class (which turns out to control the lurl»ine) ami the "ics" class (which ,'olllrols the isolation cooling system, and has two static methods I" fill and vent the system):

A void return type means the method doesn't return any value at all.

```
public oid DoCICSVent Procedure()
    Turbine turbineControl ler - new Turbine();
    turbineController.CloseTripValve(2);
    IsolationCoolingSystem.Fill();
    IsolationCoolingSystem.Vent();
    Maker.CheckAirSystem();
```

O Now the code's a lot Illon' intuitive! Even ;F you don't know tha, lin' CICS vent procedure needs to be rim if Ih,' nougat is l' H'h"l, **it's a lot .nore obvious what this code ;s doing**:

```
if (IsNouqatTooHot() -- true)
    DoCICSVentProcedure();
```

You can **make** your **code easier** to rea.l and **write by thinking** about **the problem** your **code was** built 10 solve, **If** you **choose** names **for** your **methods** Ibal **make sense** to someone **who understands** Ibal **problem,** Iben your code **will be** a 101 **easier** to **decipher...and develop!**

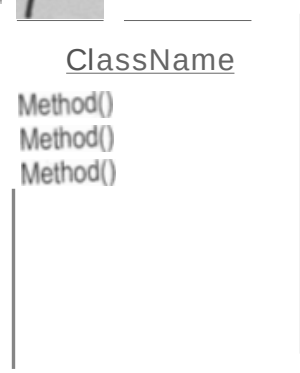
Give your classes a "atural structure

Take a sec ond and remind yourself why you want to ma ke yot1r methods intuitive: **because every progra m soh es a p bl..m or h a purpose.** h might not Ie a bt..iness l'"'obleTll - sometimes a ' ogram's 1""1" (like FlashvThing is ju t I" be cool of fun! But no matter what your program does, the ""ore you ".m m: your cod.. reseml the problenl you're trying In solve, the easier your program "ill I,,· to write ,...„1 ad, and repair, and m.,im•in.. .).

Use class diagrams to plan out your classes

A class diagram is a simple way to draw your classes out on paper It's a really valuable tool for designing your code BEFORE r- start writing it.

Write the name of the class at the top of the diagram The.. write each method ',.. the box at the bottom. No... you ea.. see all of the parts of the class at a glance!



Let', build a cia ss diagram

Take .tll. "h" r I•• k at tI,,· i f statement ill #5 "" tl,,' last page. You already know that statements always live inside lllethods, which always live inside I lasses, right? In rhis ,ase, that i f statement was in ,I method called DoMa i ntenanceTests( ), which is part of Ih.. CdndyCont ro l l e r class. Now take a l•• k at the code and Ih.. class diagram. See how they L".. t ith ?

```
pUblIC class CalidyControl ler t

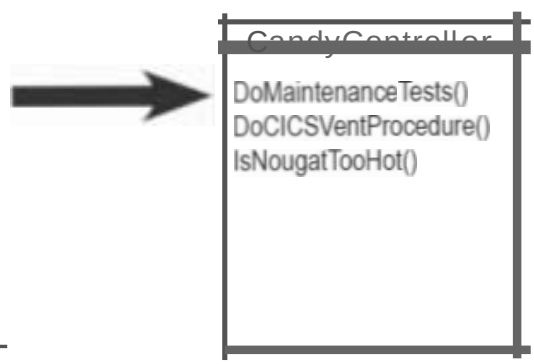
  public void DoMaintenanceTests()

    if (IsNougatTooHot() -- true)
       DoCICSVentProcedure();



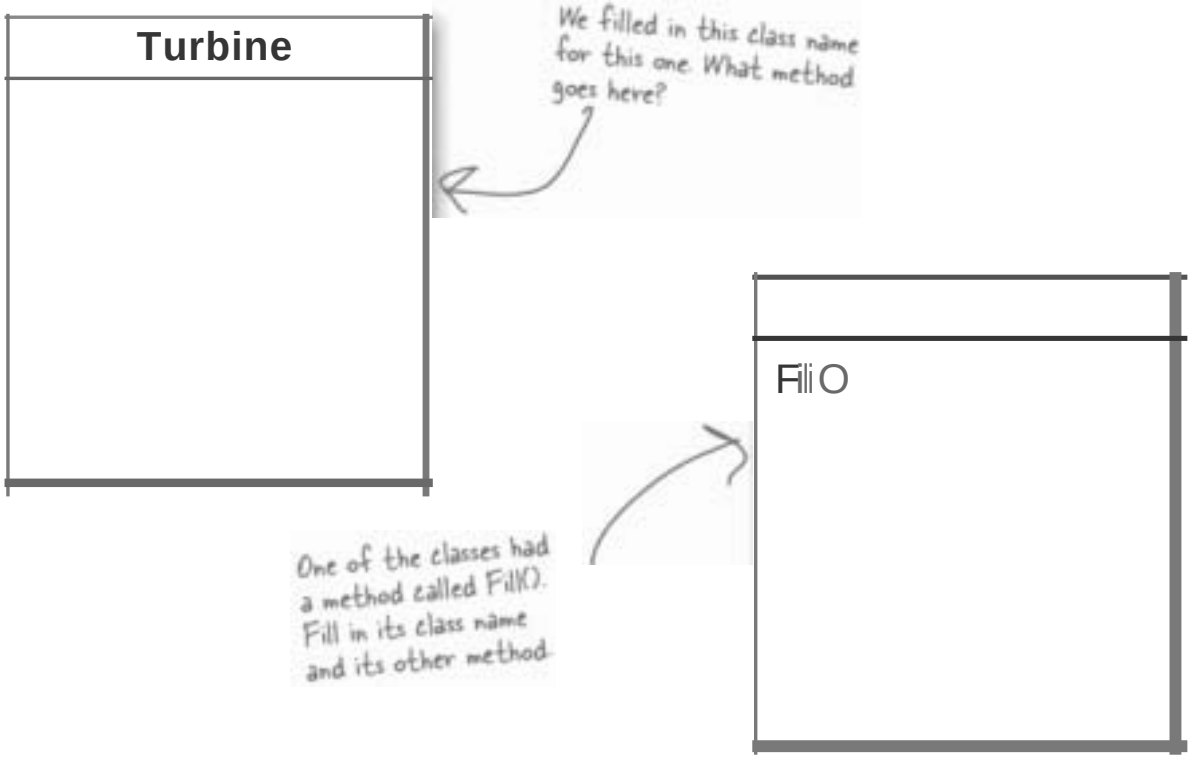
  public void DoCICSVentProcedure() .,.

  public boolean IsNougatTooHot()
```



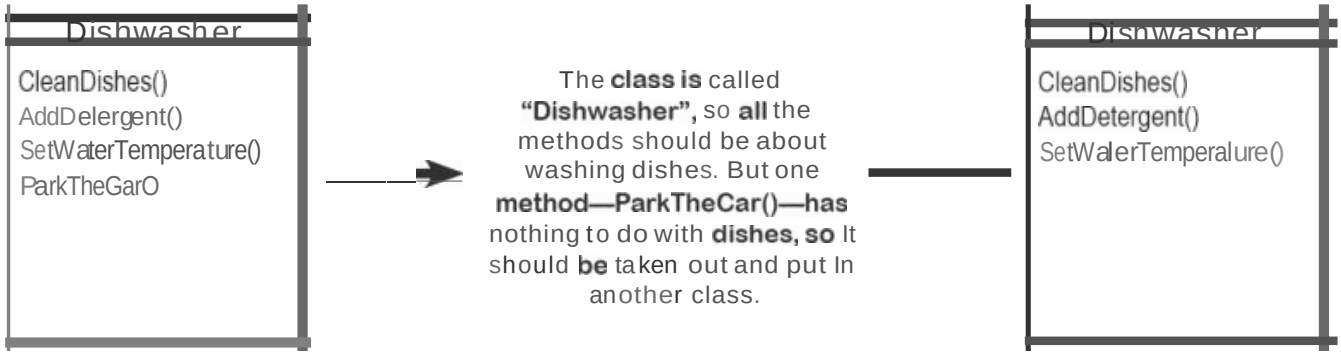
Sharpen your pencil

The code for the candy conlrol system we buill on the previous page celled three eth er classes. Flip back and look th roug h the code, and fill in the ir class di agra ms.



Class diagrallls help you orgal1ize your classes so they lIlake sease

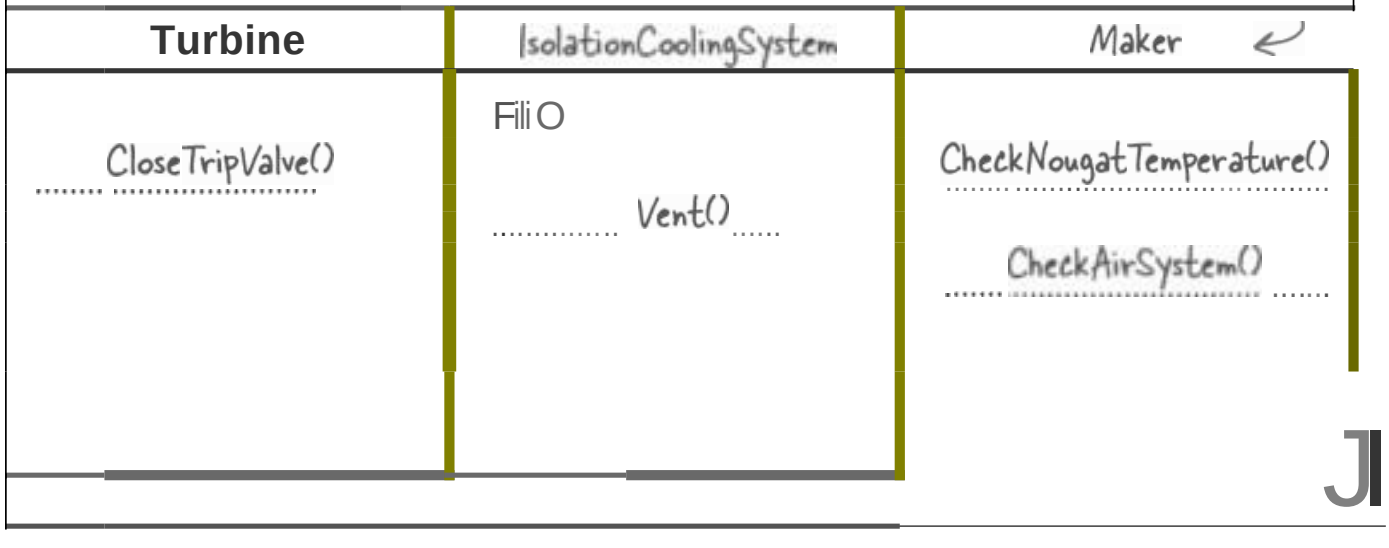
Writing ◊lll class diagrams make il a lot easier to spot potential problems ill your classe **before** you "Tit,· code. Thinking about your classe from a high level before you get into the details can help vou come up with a class structun- that will makr- 'Un' your code addresses the problems il solves. II lets you step back and make sure 1hat you're Ill,1 planning on writing unnecessary "'-poorly structured classe "r methods, and that the one you do write will be intuitive and easy to use.



Sharpen Your penel Solution

The code for the candy control system we built on the previous page called three other classes. Flip back and look through the code. and fill in their class diagrams,

You could figure out that Maker is a class because it appears in front of a dot in Maker.CheckAirSystem()



Sharpen your pencil

Each of these classes has a serious design flaw. Write down what you think is wrong with each class, and how you'd fix it.

Class23

CandyBarWeight()
PrintWrapper()
GenerateReportO
Go{)

Thi, class is part of the candy manufacturing system from earlier.

DeliveryGuy

AddAPillaO
PizzaDelivered()
TotalCashO
RetumTlmeO

DeliveryGirl

AddAPizzaO
PizzaDelivered()
TotalCashO
RetumTlme()

These two ctasses are part of a system that a pizza parlor uses to track the pizzas that are out for delivery.

CashRegister

MakeSaleO
NoSale{}
PumpGas{}
Refund()
TotalCashlnReglsterO
GetTransactionList()
AddCashO
semovecasm

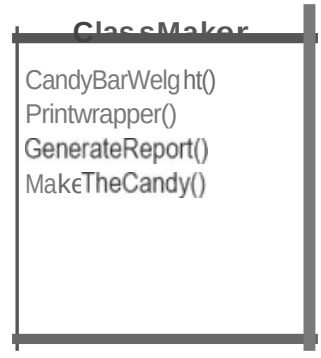
The CashRegister class i, part of a program that's used by an automated convenience store checkout system.

Sharpen your pencil Solution

Here's how we corrected the classes. They're just one possible way to fix the problems—but there are plenty of other ways you could design these classes depending on how they'll be used.

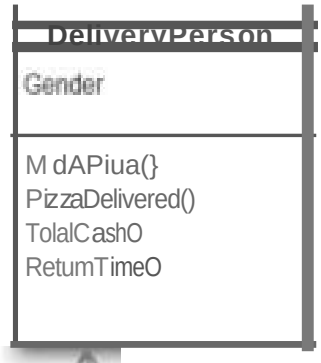
This class is part of the candy manufacturing system from earlier.

The class name doesn't describe what the class does. A programmer who sees a line of code that calls Class23.Go() will have no idea what that line does. We'd also rename the method to something that's more descriptive—we chose MakeTheCandy(), but it could be anything.



These two classes are part of a system that a pizza place uses to track the pizzas that are out for delivery.

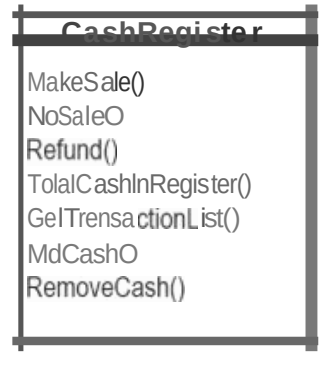
It looks like the DeliveryGuy class and the DeliveryGirl class both do the same thing—they track a delivery person who's out delivering pizzas to customers. A better design would replace them with a single class that adds a field for gender.



We added the Gender field because we assumed there was a reason to track both delivery guys and girls separately, and that's why there were two classes for them.

The CashRegister class is part of a program that's used by an automated convenience store checkout system.

All of the methods in the class do stuff that has to do with a cash register—making a sale, getting a list of transactions, adding cash... except for one: pumping gas. It's a good idea to pull that method out and stick it in another class.



```
public partial class Formi : Form
{
  private void buttonl_Click(object sender, EventArgs e}
  {
    String result
    Echo el - new Echo();

    int x   0;
    while              { {
      result - result + el.helloO + "\n";

      if (             { {
        e2.count    e2.count + 1;

                       { {
        e2.count    e2.count + el.count;

      x - x + 1;
    (
    MessageBox.Show(result + "Count: " + e2.count);

public class _____{
  public int _____ = 0;
  public string
    return "helloooo...";
```



Pool Puzzle

Your job is to take code snippets from the pool and place them into the blank lines in the code. You may use the same snippet more than once, and you won't need to use all the snippets. Your **goal** is to make classes that will compile and run and produce the output listed.

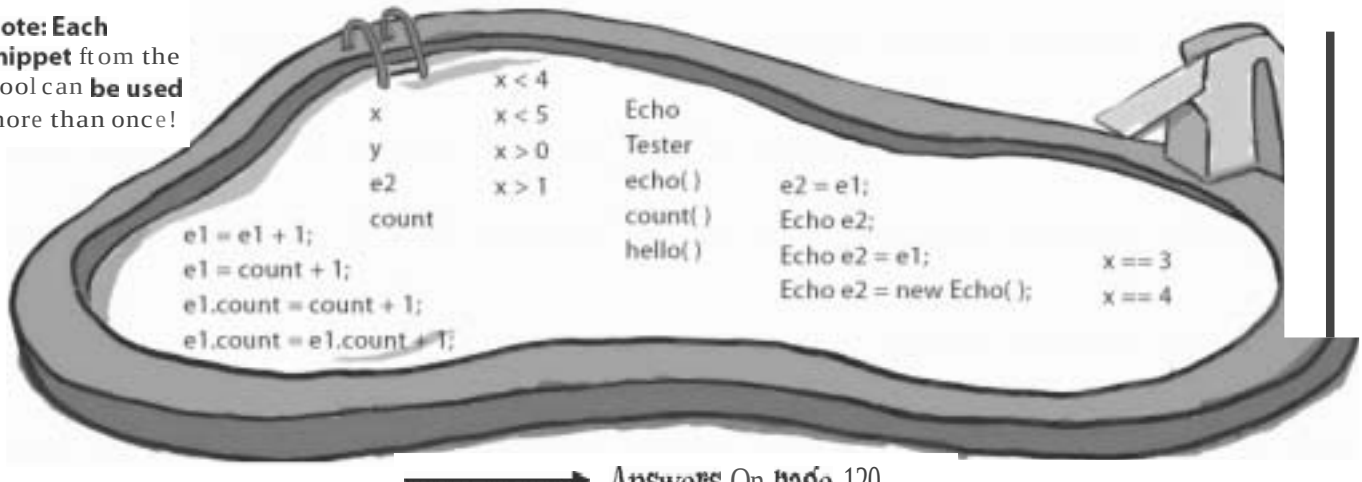
Output



Bonus Question!

If the last line of output was 24 instead of 10 how would you compiete the puzzle? You can do it by changing just one statement.

Note: Each snippet ftom the pool can **be used** more than once!



Answers On page 120.

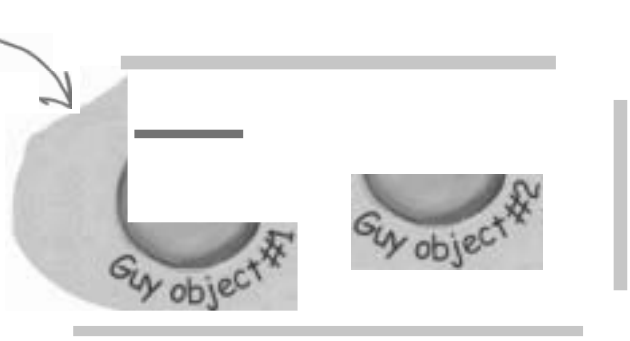
Build a class to work with sOllie guys

Joe and lI"h lend each other money all the time. Let's huild a class to keep track of them.

Guy
Name Cash
GiveCashO TakeCashO

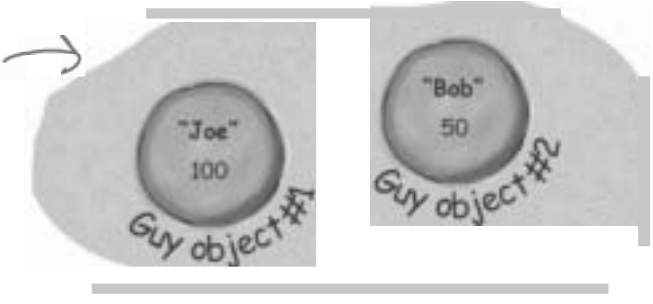
o We'll create a Guy class and add twa instances of it to **a form**
Th,' form will have two fields, o"" called joe (10 keep Ilack of the first object), and the other called bob (to keep Irack of the second object).

The new statements that create the two instances lives in the code that gets run as soon as the for is treated Here's hat the hea1' looks like after the form is loaded



o We'll **set** eoch Guy object's cash and nome
The two objects represent different guys, '0 each one has its own name and a different amount of cash in hi, pocket.

Each guy ha• a Name field that keep track of his name, and a Cash field that has the number of bucks in his pocket.



e We'll **give** cash to the guys ond take cash from them
We'll use each guy's Gi vecesh () method to give cash to a guy, and w,··]] use hi, TakeCash () 1ll" th"d to take cash back from hi",.

The form calls the object's TakeCash() ethod.

```
jo  TakeCash (25) ;
```

The method returns the number of bucks that were taken



When you take an instance of Guy and call its TakeCash() method, you pass the amount of cash to take as a parameter. So calling joe.TakeCash(25) takes 25 bucks from Joe.



Create a project for your guys

Create a new Windows Forms Application project (because we'll be using a form). Then use the Solution Explorer to add a new class to it called Guy. Make sure to add `using System.Windows.Forms;` to the top of the Guy class file. Then fill in the Guy class. Here's the code for it:

Do this!

The Guy class has two fields. The Name field is a string, and it'll contain the guy's name ("Joe"). And the Cash field is an int, which will keep track of how many bucks are in his pocket.

```
public class Guy {
    public string Name;
    public int Cash;

    public int GiveCash(int amount) {
        if (amount <= Cash && amount > 0) {
            Cash -= amount;
            return amount;
        } else {
            MessageBox.Show(
              "I don't have enough cash to give you " + amount,
              Name + " says...");
            return 0;
        }
    }
```

The GiveCash() method has one parameter called amount that you'll use to tell the guy how much cash to give you.

He uses an if statement to check whether he has enough cash—if he does, he takes it out of his pocket and returns it as the return value.

The Guy makes sure that you're asking him for a positive amount of cash, otherwise he'd add to his cash instead of taking away from it.

If the guy doesn't have enough cash, he'll tell you so with a message box, and then he'll make GiveCash() return 0.

```
    public int ReceiveCash(int amount) {
        if (amount > 0) {
            Cash += amount;
            return amount;
        } else {
            MessageBox.Show(amount + " isn't an amount I'll take",
                    Name + " says...");
            return 0;
        }
    }
```

The ReceiveCash() method works just like the GiveCash() method. It's passed an amount as a parameter, checks to make sure that amount is greater than zero, and then adds it to his cash.

If the amount was positive, then the ReceiveCash() method returns the amount added. If it was zero or negative, the guy shows a message box and then returns 0.

Be careful with your curly brackets. It's easy to have the wrong number—make sure that every opening bracket has a matching closing bracket. When they're all balanced, the IDE will **automatically indent** them for you when you type the last closing bracket.

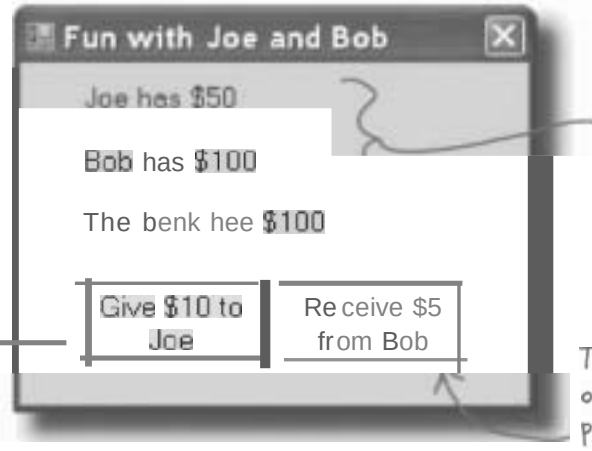
Build a forlll to lnteraet with the guys

The Guy class is great, but it's just a start. Now put together a form that uses two instances of the Guy class. It's got labels Ih"l show you their name and how much cash they have, „,,d buttons t" give and t"k,- cash from them.



O Add two buttons and six labels to your form

'11„- labels "ll the right-hand ,id,- of the form show how much cash each guy has. \Ve'll also add a variable called bank 10' the form—the third label "1\ the right shows how much cash is in it. We're going t" have ,""ll name some of the labels that you drag onto Ih,' forms. You can do Ih"l by **clicking** on ..**ach lab..l** that you want to name and **changing** Its "IN anI")" row in the Properties window. That'll make your code a 101 easier to read, because you'll be able to use "joesCash" and "bobsCash" instead of "label4" and "label5"



Name the top label joesCash, the label underneath it bobsCash, and the bottom label bankCash You can leave their Text properties alone, we'll add a ..ethod to the form to set the..

This button will call the Joe object's ReceiveCash() method, passing it 10 as the amount, and subtracting the cash it gives to Joe from the bank variable

This button will call the Bob object's GiveCash() ..ethod, passing it 5 as the amount, and adding the cash it receives from Joe to the bank variable.

O Add variables to your form

Your f"nn will need to keep track of the two guys so you'll need a variable 1"„ each of them. Call them joe and bob. Th"n add a variable to the f"lIn called bank to keep track "I' how much money Ih,' form has to give t" and receive from th.- guys

```
namespace Your_Project_Name {
public partial class Forml    Form (
    GUY joe;
    Guy bob;
    int bank .. 100'

    public Forml ()
        InitializeComponent();
```

Since we're using Guy objects to keep track of Joe and Bob, you declare their variables using Guy.

The amount of cash in the bank goes up and down depending on how ..uch money the for... gave to and received fr_ the Guy objects

4 Add a method to the form to update the labels

The lab..ls "n the right-hand ,id" „1' the fi,nll show how much cash each guy has and how much i, in the bank \ariable. So add lh" UpdateForm() method 10 keep Ih"'rl IIp 10date—**mak..** **sur..** th.. r..lurn lypp **is** void to tell C# that II", method doesn't return a value. Type Ihi, method illlOthe 1'1'111 right underneath where you add..d Ih,- bank variable:

```
pUblic void UpdateForm() {
    S JOeScash.Text = Joe.Name +   has $" + joe.Cash;
    bobsCash.Text = bob.Name +   has $" + CCu.ceen.
    bankCash.Text   "The bank has $" + bank;
}
```

Notice how the labels are updated using the Guy objects' Name and Cash fields.

This new ..ethod is simple It just updates the three labels by setting their Text properties You'll have each button call it to keep the labels up to date

4 Double-click on each button and odd the code to mtercct with the objects

Make sure the lefi-hand billIon i' called button1, and the right-hand hllll"ll is called butt.onz. Then double-click ",,,-h„1' the buttons— when you do, the IDE will add two methods called button1_Click() ,,,,d button2_Click() to the form. Add this cod,· to each „1' them:

```
private void buttonl_Click(object sender, EventArgs e) {
    if (bank >= 10) {
        bank -= Joe,RecelveCash(10);
        UpdateFormO;
    else {
        MessageBox.Show("The bank is out of money_");
```

When the user clicks the "Give $10 to Joe" button, the for·· calls the Joe object's ReceiveCash() ..ethod—but only if the bank has enough money

The bank needs at least $10 to give to Joe If there's not enough, it'll pop up thi. ..essage box.

```
private void button2_ClicK(object sender, EventArgs e) {
    bank +- bob.GiveCash(5);
    UpdateFormO;
```

The "Receive $5 from Bob" button doesn't need to check how ..uch i. in the bank, because it'll just add whatever Bob gives back.

If Bob's out of money, GiveCash() w,ll ..etu..n zero.

5 Start Joe out with $50 and start Bob out with $100

It's "I' I" you to **figure** oul how 10 gplJo.. and Bob 10 **start** oul with Ih..ir **Cash** and Nam.. fi..ld" **s**..t prop..rly. 1'll1il right underneath InitializeComponent() in the form. That's part of a special method that gets run once, when the form is lil"1 i"ili"liz,-,U)nce you've d,"I" that, click h" lh I'Ull'ns a number ,1' times— mak(sure lhal one butt(.u takes $11) fr(uu the bank and adds il to Joe, and the other take $5 from Bob and adds it 1◇ the bank.




```
public Form() {
    InitializeComponentO;
    // Initialize joe and bob here!
```

Add the lines of code here to c eate the two objects and set their Na..e and Cash fields.



It's up to you '" **figure** oUI h..w 1.. g.. IJ.... and Boh **to star**1 oUI wilh Ih.·i**r Cash** and Nam**e** fi..**lds s**..t **prop**....ly. Put i, right underneath Ini tial i zeComponent ()

```
public Forml ()
    InitializeComponent() ;
```

Here's where we set up the first instance of Guy. The first line creates the objeet, and the next two set its fields.

```
    bob = new GUy () ;
    bob.Name = "Bob";
    bob Cash = 100;
```

```
    joe = new GUy 0 ;
    joe.Name = "Joe";
    joe.Cash = 50;
```

Then we do the sa..e for the second instance of the Guy class

Make sure you call UpdateFormO '0 the labels look ..ight when the form first pop "T'.

```
    UpdateForm () ;
}
```

there are no
Dumb Questions

Make sure you save the project now—we'll come back to it in a few pages

Q: **Why** doesn't **the** lolution **start with "Guy** bob = new GUy **( ) "? Why** did you **leave** off tile **first "Guy"?**

A: seceuse **you already declared the** bob variable at **the top** of the **form. Remember** howthe statement 'int i - 5;' ISthe **same** as the**two statements "int i'** and 'i - 5;"? This is the **same** thing You **could try** to **declare the** bob vanable in **one** line **like** this: "Guy bob - new Guy () ; But you already have the **first** partof that statement ('Guy bob ;"j at ee topof **your** form **So you only need the second half** Of **the** line, the partthai sets the bob **variable** 10**create a** new nstance of Guy () .

Q: Okay, so then **why** not get rid of **the** "Guy bob ; " line at **the** top of **the** form?

A: Then the bob venabre will only **exist inside** that special 'public Forml () . method. When you declare a veneere inside a method, it's only valid inside the method—you can't access il from any otherme**thod**. But when you **declare** Il outside of your metroe**but** n eoe the **form** or a **class** that **you** added. then you can **access** it **from** any *olher method* inside the form or cress

Q: **What happens** if I don't **leave** off that **first "Guy"?**

A: You'll run Into**problems**—your form won't**work**, **because** it wont**ever set** the form's bob **variable**. Th**ink** about**it** for a **minute**, and **you ll see** why it **works** that **way**. If **you** neve **this** code at **the** top of your **form**

```
public partial class Formi : Form (
    Guy bob;
```

and then **you have** thiS code later on, **inside** a **method**

```
Guy bob - new GUy();
```

then you've declared */WO* veneues. It's a little confusing because they both **have the same name** But **one** of them **is** valid throughout the entire form, and the**other** one-the new**one you added—is** only valid **inside** the **method**. The next line (bob . Name - "Bob";) only updates that ***local*** variable, and ooesn't touch the one in the form . So when **you** try to run your code. it'll **give you** a **nasty error message** ("NullReferenceException not handled'), **which** just means **you** tned to **use** an **object** before **you created** it with new.

There's an evell easier way to illitialize objects

Almost every "hj!",'I that you create needs 10 I,,' initialized ill '''Ill" way. And the Guy object is no exception - it's useless until you set its Name and Cash fields. It's so common 1" have t" initialize fields that C# gives you a shortcut for doing it called all obj..**ct** inilializ..r..\"d the IDE's IntelliSense willhelp you do it.



Watch it!

Object initializers only work with C# 3.0.

If you're running Visual Studio 2005. then this won't work. Definitely consider downloading Visual Studio 2008 Express Edition—it's free. end you can install it alongside your existing VS2005 installation.

O II. :re's Ill., original code that you wrote I" initialize Joe-, Guy "hj,"l.

```
joe = new GuyO ;
joe.Name = "Joe";
joe.Cash = 50;
```

❷ Delete the second tw(lines, a"d the semicolon afte, "Guy () " ami add a right curly bracket.

```
joe = new Guy {) {
```

❸ Press space. .As soon as you do the II1E pops up an IntelliSense window th"l show ""u allor the fields that you're able 10 inil;"lil<'.

```
joe = new Guy {) {
```



O Press tab tu tell it (o add the Cash field. Then set il "qual '" 50.

```
joe = new Guy{) ( Cash = 50
```

O Type ill a comma. As soon a, you do, the other field show "I'.

```
joe = new GUy{) { Cash = 50,
```



O Finish the object initializer. Now you've saved yourself two lines of code!

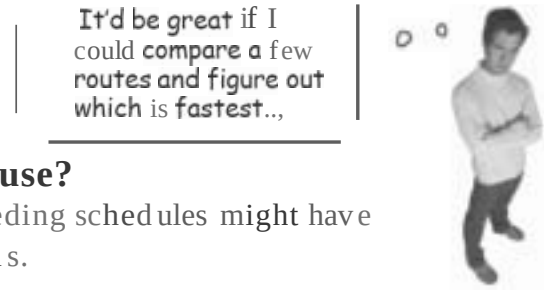
```
joe = new Guy O { Cash = 50, Name = "Joe" };
```

This new declaration does exactly the same e-S as the three lines of code you wrote originally. It's just shorter and easier to read

Object intializers save **you time** and **make** your **code more compad and easier 10 read...** and **the** IDE **help'** you **write them.**

Afew ideas for designing intuitive classes

- **You're building your program to solve a problem.**
  Spend some time thinking about that problem. Does it break down into pieces easily? How would you explain that problem to someone else? These are good things to think about when designing your classes.



- **What real-world things will your program use?**
  A program to help a zoo keeper track her animals' feeding schedules might have classes for different kinds of food and types of animals.

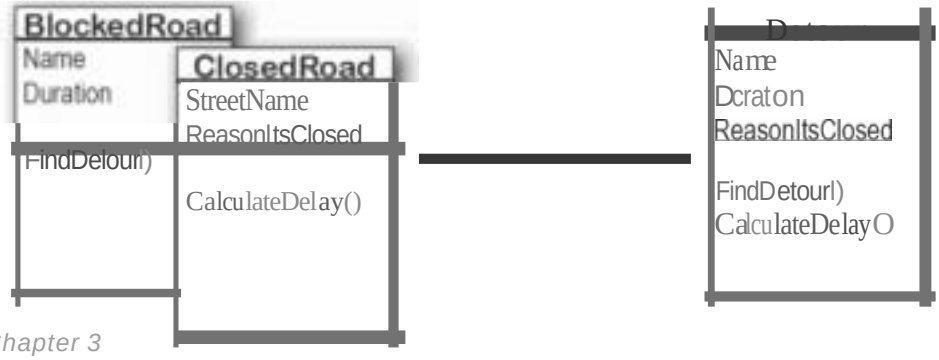


- **Use descriptive names for classes and methods.**
  Someone should be able to figure out what your classes and methods do just by looking at their names.





- **Look for similarities between classes.**
  Sometimes two classes can be combined into one if they're really similar. The candy manufacturing system might have three or four turbines, but there's only one method for closing the trip valve that takes the turbine number as a parameter.





Add buttons to the "Fun wilh Joe and Bob" program 10 make the guys give each other cash,

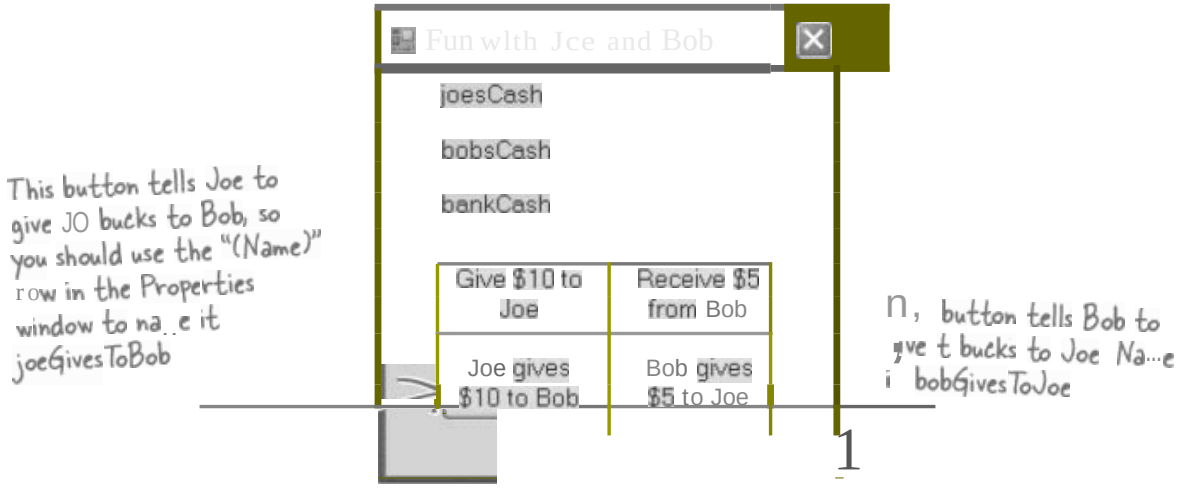
Use an object initializer to **initialize** Bob's instance of **Guy**

You've already done it with Joe Now make Bob's instance' work with an object initializ"r too.

If you already clicked the button, just delete it, add it back to your form, and rena..e it Then delete the old button3_Click() ..ethod that the IDE added before, and use the new method it adds now

Add two **more** buttons to your form

The first button tells Joe to give 10 h"dcs to Bob, and the second tells Bob In give 5 bucks back to Joe. **Before you double-click on the** button, go to the Properties window and change each button's name using the "(Name)" row—it's **at the top** of the li,t "l' properties Name the first hUll"" joeGivesToBob, and the second ""e bobGivesToJoe.



This button tells Joe to give JO bucks to Bob, so you should use the "(Name)" row in the Properties window to na..e it joeGivesToBob

n, button tells Bob to ;ve t bucks to Joe Na...e i bobGivesToJoe

Make the buttons work

Double-click " ll the joeGivesToBob blllloll in the designer. The IDE will add a method t(thr- form called joeGivesToBob_Cl i c k () that gets run any (im,·the button's clicked. Fill in that method tn make Joe give 10 bucks ı" Bob. Then double-dick '"' the ',Iher button "nd Jill i" *th*.. new bobGivesToJoe Click () method rh.u the IDE creates s(that Bob gives five bucks to Joe. Make sure the form update itself after the cash change hands.



Add buttons to the "Fun with Joe and Bob" program to make the guys give each other cash

```
public partial class Forml : Form {
    Guy joe;
    Guy bob;
    int bank - 100;

    public ronou {
        InitializeComponent();
        bob • new Guy() { Cash = 100, Name = "Bob" };
        joe .. new GUy() ( Cash = 50, Name = "Joe" };
        lJpdateForm O ;


    publ ic void UpdateForm() {
        joesCash.Text   joe.Name + .. has $" + joe.Cash;
        bobsCash.Text   bob.Name + " has $" + bob.Cash;
        bankCash.Text   "The bank has $" + bank;


    private void buttonl_Click(object sender, EventArgs e) {
        if (bank >= 10) {
            bank -= joe.ReceiveCash(lO);
            UpdateForrn O ;
          else {
            MessageBox.Show("The bank is out of money.");


    private void button2_Click(object sender, EventArgs e) {
        bank +_ bob.GiveCash(5);
        IJpdateForm O ;


    private void joeGivesToBob_Click(object sender, EventArgs e) {
        bob.ReceiveCash(joe.GiveCash(10));
        UpdateForm();


    private void bobGivesToJoe_Click(object sender, EventArgs el {
        joe.ReceiveCash(bob.GiveCash(5));
        UpdateForm();
```

Here are the object initializers for the two instances of the Guy class. Bob gets initialized with 100 bucks and his name.

To make Joe give cash to Bob, we call Joe's GiveCash() ..ethod and send its results into Bob's ReceiveCash() method

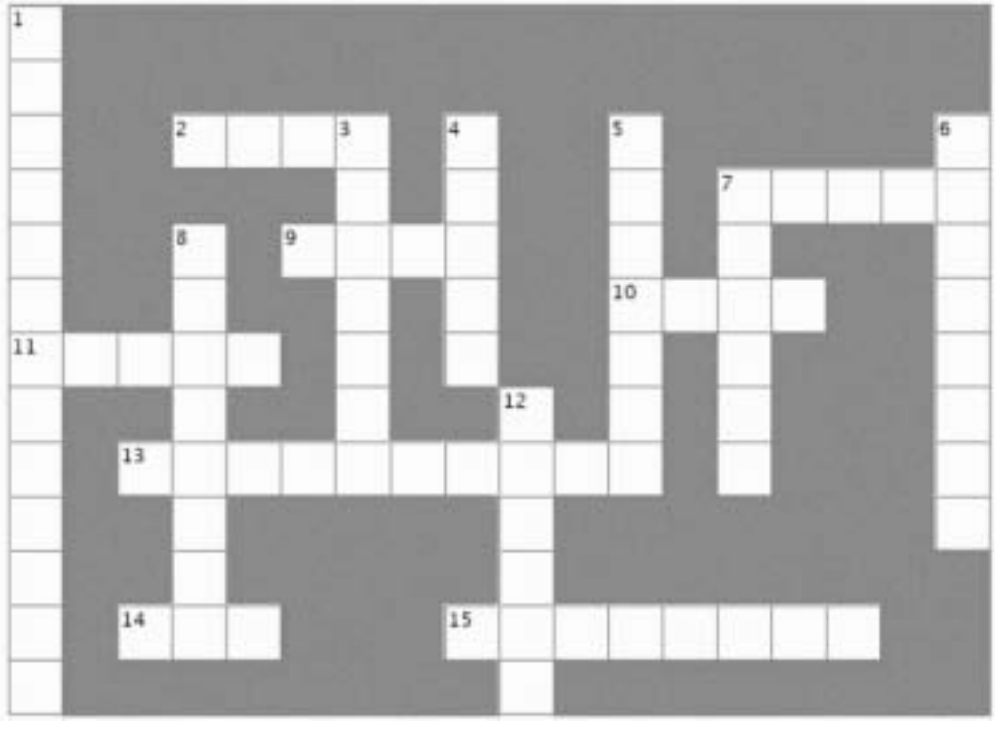
Take a close look at how the Guy ..ethods are being called. The results returned by GiveCash() are pumped right into ReceiveCash() as its para..eter.

The trick here il thinking through who's giving the cash and who's receiving it.



Objeetcross

It's time to give your left brain a break, and put that right brain to work: all the words are object-related and from this chapter.



Across

2. If a method's return type is _____, it doesn't rewm anything

7. An object's fields define its _

9 A good method makes it clear what the method COeS

10, Where objects live

11. What you use to build an object

13. What you use to pass Information Into a method

14, The statement you use 10 create an object

IS. A special kind of field that's used by the form controls

Down

1 This form control lels the user choose a number from a range you set

3. It's a great idea 10 create a class on paper befOf€ you start writing code

4. What an object uses to keep track 01 what it knows

5, These define what an object does

6 An object's methods define its _

7. Don't use this keyword in your class declaration if you wenno be able to create instances of it

8. An object is an of a eass

12. This statement tel Is a method to immediately snt, and specifies the value that should be passed back to the saterreot that called the mefhod

Pool puzzle Solution



YOUR *job* was to take code snippets from the pool and place them into the blank lines in the code. Your ***goal*** was to make classes that will compile and run and produce the output listed.

```
public partial class Forml : Form
(
  private void buttonl_Click(object sender, EventArgs e)

    String result - "";
    Echo el _ new Echo();
    Echo e2 = new Echo();
    int x   0;
    while  ____________ ( (
      result   result + el.helloO + "\n";
       el.count - el.count + Ii
      if (    x == 3
        e2.count - e2.count + 1;

          ____________ ; (
        e2.count = e2.count + el.count;

      x = x + 1;
    (
    MessageBox.Show(result + "Count: " + e2.count);

          ass  Echo
           it  count

    pUblic string  hello( )
      return "helloooo••.";
```

That's the correct answer

1

And here's the bonus answer!

```
Echo e2 = el;
```



Objectcross Solution



4 Types and References

It's 10:00.

Do you know where your data is?



Data type, database, Lieutenant Commander Data... it's all important stuff. Without data, your programs are useless. You need information from your users, and you use that to look up or produce new information, to give back to them. In fact, almost everything you do in programming involves working with data in one way or another. In this chapter, you'll learn the ins and outs of C#'s data types, how to work with data in your program, and even figure out a few dirty secrets about objects *(psst!... objects are data, too)*

Ihe variable's **type determines what kind** of data it **can** store

There are fifteen v"lu.. typ..s huilt i" t" C#, and each one stores a different kind of data. You've already seen some of the most ""Illl" On ones, and you know how 10 uv-th-m. But there are a few that you haven't seen, and they can really come iu handv, too.

Value types you'll use all the time

It shouldn't come as a surprise that `int`, `string`, `bool`, and `float` are the rnml n ,rnnron types.

A whole nu..ber doesn't have a deci..al point.

- `int` can store any whol.. "umlwr from -2,147.483,648 1" 2,147.483,647.
- `string` """ I",ld text of any length (i" d llding the empty string "").
- `bool` is a Boolean value—it's either `true` or `false`,
- `float` can store any **decimal** number from $\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ wilh "I' to 7 significant figures Thal range looks weird and colllplicated, but it's actually pretty ,impl(', The "significant figures" part means II,,' *precision* of the number: 35,048,4 I0.000,000, 1.7-\:\.059, 1-\,43857, and 0.0001-\37-\155 all h,II" seven significant figures The lO" thing means that you can "; " anv number as large as 10^{38} (,·r I f(,II,)wed by 38 zeroes)—as long as 'I only has ı ··· fewer significant figures. CIll Ih,' other end "f th,' range, Io-' means th,u y(n, ,',m store any Ill"nber a small as 10^{-45} (or a decimal poinl followed by 45 zeroe followed by I) .. but, you guessed it. as long as il only has 7 or fewer significant figure

"float" is short for "floating point"—as opposed to a "fixed point" number, which always has the same number of decimal places.

More types for whole numbers

The "u" i" uint stands for "unsigned", which ..eans it can't be negative (so there's no ..inus sign).

( )nce up"u a time, computer memory was really expensive, and processors were reallv slow. And, believe it 0" not, if you used the wrong type, it could seriouslv slow down youu program. Luckily, limes have changed, and 1Il0,t "f lit,· lime if you """ ,l to store a whole 1I11lnl,,·,. y(," can just use an `int`. lI"t sometimes " ,,, really need s(>In,·thing bigger... and once in a whik-, you ne-d something smaller, too,_ "That's why C# give you more opl ions:

A lot of times, if you're using these types it's because you're solving a proble.. where it really helps to have the "wrapping around" effect that you'll read about i" a few ..inutes

- `byte` can 'ton' any **whole** number betwceu 0 ,,,\(1255.
- `sbyte` can ,t,,,·· any whol.. number f""n -127 to 128.
- `shert` can store any whol.. """'nl.·r from -32,767 to 32,768.
- `ushort` can store any **whole** number from 0 to 65,535.
- `uint` can store """ whol.. number from 0 to 4,294,967,295.
- `long` can store any number between minus illHI pI", 9 hilli"" hilli"n.
- `ulang` can store any numlber between I) and abdUI I8 hilli"l1 hillion

The "u" stands for "unsigned"

Types for storing *really* HUGE and *really* tiny numbers

Sometime 7 significant figure just isn't precise enough. .\nd, believe il or not, sometime)ı)''' isn't big enough and 10^{-4} isn't small enough .\ I" t "I programs written foı finance or scientifi resı arch nllllllO thesı probl Ill' "ll thı IlIn,. so C# gives "" two mo" types

- double can stor.. any lH,,,,I,,-,.lfOn, $\pm 5.0 \times 10^{-324}$ 10 ±1,7 " 10^{308} ""Ih 15-16 significant digit.
- deci rna1 can ,It'e any number frıun $\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$ with 28-29 significant digits.

The double type is actually as common as float. A lot of people use it all the time, and rarely use float

A literal just means a number that you type into your code So here you type "int i = 5;", the 5 is a literal

When you used the Value property in your numericUpDown control, you were using a decimal.

Literals have types, too

\\'1he" you type a Illllllhe,. directly into your C# prog'-"'". you'r,' using a **lit..ral**... and every literal is automatically assigned a t)1"" Y'" can ,,-,. this for yourself - jll,t enter this line .,f "(„1,-th"t assigns the literal `14.7` h, an `int` variable:

```
int myrnt - 14,7;
```

Now try to build the program. You'll get this:

```
Cannot implicitly convert type double to int. An
explicit conversion exists (are you missing a cast?)
```

equal to a `double` variable. What

A few more useful built-in value types

Did you ""Ii" e how the `byte` type has 256 possible values? Well, there's another typı that also has 256 possible '-'11",-" `char`. But it's not numeric; you use il h. storı a character. Literal values li,l' `char` are always ill,id" single rl""l'-' (`'x'`, `'3'`). y,,,, can include **escape sequences** in the quotes, too (`'\n'` i, a line break. `'\t'` is a tab).

And finally, there's one more i'"I'"I1;"'1 type `object`, You've already seen how an "hj""t"'''' inherit !i-"1ll another 0"'-, a"d that obj,-"t can ill turn illl",rit lilllll yet a different "bj,-ct. ,\t the Itll' of every inheritance hi.-rard,y i, th,-`object` class that's a special type Ih"l every otlwl' object iuhcrns from. That's really useful, because it means that you **can assign** any value, variable, ◇r **object** 1◇ an **object variable**

You'll learn a lot more about how `char` and `byte` relate to each other in Chapter 9.

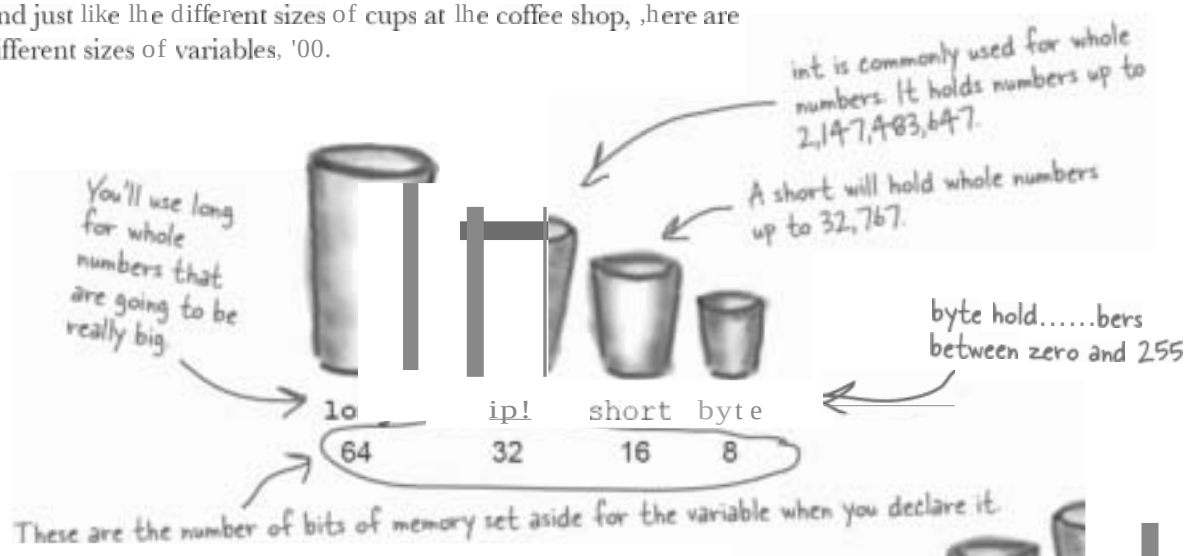


You can use the Windows calculator to convert between decimal (normal. base-tn) numbers and binary numbers (base-2 numbers written with only ones and zeroes)-put it in Scientific mode, enter a number, and click the Bin radio button to convert to binary. Then click Dec to convert it back, Now enter some of lhe upper and lower limits for **the** whole number types (like -32,767 and 255) and convert them to binary, Can you flgure out *why* C# gives you those particular limits?

Avariable is like a data to-go cup

All of your data take, "I' space in ",emory. :Reme",her the heap from la'l chapter?) So part of your joh i, 10lhiuk about how *much* space you're going 10 "eed whenever you use a string or a number in your program. That's one of the reasons you use variables. They Ier vou ,eI aside enough space iu memory to store your data.

Think of a ,-a';able like a cup that you keep your data in. C# uses a bunch of different kinds of cups to hold different kinds of data. And just like lhe different sizes of cups at lhe coffee shop, ,here are different sizes of variables, '00.



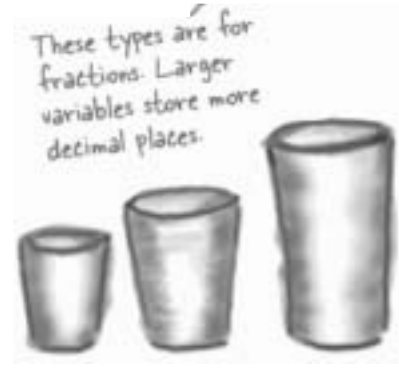
Numbers that have decimal place are 'lor"d differentlv lhau whole Ill,mhers. You can handle most of vour mnnbers that ha,-e decimal place, using `float`, the smallest data type lhat stores decimals. If vou need 10 be more accurate, use a `double`, and if you're writing a scie",ifi,- al'l'li,-alio" where the llumlɔe,,, "..ed '0 he extremely accurate, the `decimal` type ha' the most 1'1',,(i,ion.



`float` 32

`decimal` 128

It's nol always about mnnber" though. (You wouldn't expect 10 gel hot coffee in a plastic cup or (old coffee in a paper one.) The C# compiler also can handle characters and non-numeric types. The `char` type hold, Oi'e character, aud `string` is used for lots of characters "strung" together. There·s no set ,ize for a `string` variable, either. Ir expand' 10 hold as much data as you need 10 store in it. The `bool` data type i, used 10 store In'e or false values, like the one vou've used for vour `if` statements.



`bool` 8

`char` 16

`string` depends on the size of the string

`string` is pretty unique. It's the only data type without a set size, except for objects (think about that for a bit)

10 pounds of data in a 5 pound bag

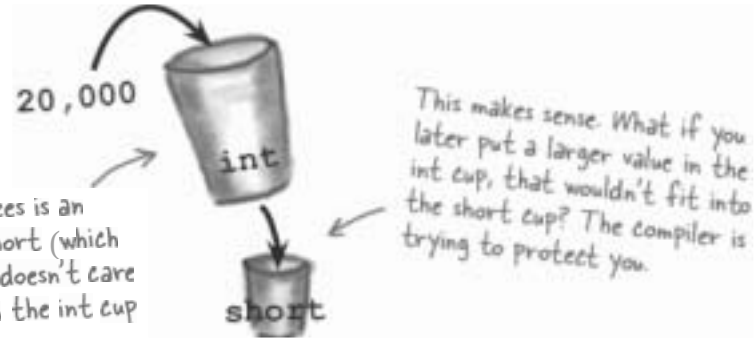


When you declare your variable as one type, that's how your compiler looks at it. Even if the value is nowhere near the upper boundary of the type you've declared, the compiler will see the cup it's in, not the number inside. So this won't work:

```
int leaguesUnderTheSea = 20000;

short smallerLeagues = leaguesUnderTheSea;
```

20,000 would fit into a `short`, no problem. But since `leaquesUnderTheSea` is declared as an `int`, the compiler sees it as a pint-sized and considers it too big a number to put in a `short` container. The compiler won't make those translations for you all the fly. You need to make sure that you're using the right type for the data you're working with.



Sharpen your pencil

Three of these statements won't compile, either because they're trying to cram too much data into a small variable Or because they're putting the wrong type of data in. Circle them.

```
int hours = 24;

short y = 78000;

bool isDone = yes;

short RPM = 33;

int balance = 345667 - 567;

string taunt = "your mother";

byte days = 365;

long radius = 3;

char initial = 'S';

string months = "12";
```

[veil whell a llulI1ber is the right size. you call't Just assigll it to allY variable

Let's ".,. what happens when you try 1" assign a `decimal` \'"hl'- to an `int` variable.

r:: Do this

o Create a new project and add a hnnon to it. Then add these lines to the button's `C11ck ()` method:

```
decimal myDecimalValue = 10;
int myIntValue = myDecimalValue;

MessageBox.Show("The myIntValue is " + myIntValuej;
```

o Try building \',"ut' program. Uh... 11 - y'll g,,t all error 1hat l,oks lik,' this:

e

Sharpen your pencil Solution

e>. Thr.... of thes.. stat..ments won't compil... eilh ..r becaus.. they're trying to cram too much data into a small variable Orb..caus.. they're putting the wrong type of data in. Circle them.

```
short y = 78000;
```

The short type holds numbers from -32,767 to 32,768 This number's too big!

```
bool isDone   yes;
```

You can only assign a value of "true" or "false" to a bool.

```
byte days = 365;
```

A byte can only hold a value of up to 256. You'll need a short for this.

Whel1 you cast a value that's too big, C# will adjust it autol\latically

You've "1ready seen Ih,,(., decima1 t·.·" , I,,' cast ." an int. It turns ....I that *any* nll",I"'r can l..· cast t" *any other* tllllllber. But that doe' " .' 1lnc·1ll the 1*alue* stavs inl,,,-, through the casting. If you t·.,., .lll 1nt variable· that's set I" 365 ••• " byte variable··365 is ,ou, big for the byte But ;"" ..ad of giving you an error. the value " ill j ll>' **wrap** around ' fi example, 256 cas '" a byte will h."e a value of 0. 257 w lei b.. convertec1'" l. 258 '" 2. ..tc., " \' l,. 365. which will end "l' being **109** .\ ",1onc·· you get bc.. k '" 255 again, the t-, nversion ,-,1" ., "wr"l"- 10,,. '" zero.



Hey, I've been combining numbers and strings in my message boxes since I learned about loops in Chapter Zl Have I been casting this whole time?

Yes! The **+** operator **casts** for **you.**

What you've bee" ing is using l" + ', "Totl'". whic li d **a** 101 or **casting** ror **you** a" lom alically btll it's e..pecially smart abou ll it. When \" u use + to add a lllUlll"T ·or boolean 1" a string, then it'll ;"'1,matically .'.nvert that value to a string, too. If you use + In' ·· / ", -) with two differe"1 types, il ....**tomatically casts the smaller type** 10 lh**e bigger one** Here's an example:

```
int myInt = 36;
float myFloat = 16.4F;
myFloat = myInt + myFloat;
```

When you're ass_g..ng a number value to a float, you need to add an F to the end of the .....ber to tell the compiler that it's a float, and not a d...ble

Since ;," i n t can iii i"I" ;' f loat in" a t 10at ",tll'cIii i"I" an i nt.. the + operato casts myl nt. I"'" float befi r·adding il «, myFloat.

Wrap it yourself!

There's no mystery to how casting "wraps" the numbers—you can do it yourself. Just pop up the Windows calculator, switch it to Scientific mode, and calculate 365 Mod 256 (using the "Mod" button, which does a modulo calculation). You'll get 109.

Sharpen your pencil

Vou **can't always cast any type** to **any** olh·· **type.** ('eaI.. a new project, drag a button cetc a form, and type these sta,ements into its method. Then build your program—it will give lots of errors. Cross out lh.. ones that give errors. That'll help you figure out which types can be cast , and which can't!

```
int myInt = 10;

byte myByte - (byte)myInt;

double myDouble • (doublel_yByte:

bool lIy800l • (booll.yDouble:

string myString = "false";

myBool = (bool)myString;

myString (string)myInt;

myStrinq _ .ylnt.ToStrlng();

myBool _ (boollmyByte,

myByte (byte)myBool;

short myShort • (short)myInt;

char myChar - 'x',

myString (string)myChar

long myLong Oonq)mylntl

decimal myDecimal - (decimal)myLong;

mySttinq - myStrinq + mylnt + myByte
+ myDouble • myChar;
```

C# does some casting automatically

There are two important conversions that ,I",,', require you to ,I" the casting. The lir' l is d"" automatically any ' imr you use arithmetic operat ors, lik,' in thi. example:

```
long 1 _ 139401930;
short e - 516;
double d - 1 - s;
d .. d / 123.456;
MessageBox.Show("The answer is " + d);
```

The - operator subtracted the short from the long, and converted the result to a double

This + operator " smart / enough to convert the decimal to a string

Tlue other wan C# converts types for y..u automatically is when you use the + operat or I" **concat**..na'e strings ,which just means sticking one string on 1 id ,,(another, like yu..i've been doing "'i'll sage I..xes!. Wl you use + In concatenate a string wilh ,ething that'•.nother ty,"'. it automatically converts Ihr n"ml"'r, 'n strings for you. 11,re·· an example. '11ue first Iwo Ii"es ,'rr fine, but the Ihird 'ne ''''"1 co...lpile.

```
long x - 139401930;
MessageBox.Show("The answer is " + xl;
MessageBox.Show(x);
```

'Illr C# compiler spits out an error Ih.,' mentions something about invalid arguments (an argument is another name for ,' """Ih"d', p.lr.m...ter). T h.,' ·. I....:ause the parameter for **MessageBox.** Show (I is a et.ring. and this code passed a long, which i. the wrong type for II,,' "'•.,hod. But you can convert il ,,, a string really easily by calling its ToStrlnq (I method. That m"II"d is a 11"·,,,1oer "f every valt... ty1"' and object. (All "f the (l.",," you 1>lliJd yourself have a ToSteing () method that returns Il... class }(,un,·.) That's how you can conver x to s ,nlething .h.rl **MessageBox. Show** (I can use:

```
MessageBox.Show(x.ToString());
```

Sharpen your pencil

Solution

You can't always c..t any type to any other type. Create a n__ project, drag a button onto a form, and type these statements inlOits method. Then build youl program—it w,11 give lots 01 errors. Cross out the ones that give errors. That'll help you figure out which types can be cast , and which can't!

```
int .yInt • 10,
byte myByte • (byte)myInt;
double ...yDouble • (double)myByte;
[illegible]
steing .yString "false";
[illegible]
[illegible]
...ySteing myInt.ToString();
[illegible]
[illegible]
short myShort (short)myInt;
char myChar = 'x':
[illegible]
long myLong • (long)mylnt:
decimal myDecimal • (decimal)myLong;
myString • myString + mylnt + myByte
+ myDouble + myChar;
```

When you call a method, the variables Il\ust Il\atch the types of the parall\eters

Try calling `MessageBox.Show(123)` — passing MessageBox.Show() a literal (123) instead of a string. The IDE won't let you build program. Instead, it'll show you an error in the Error List: "Argument 1: cannot convert from 'int' to 'string'."

But MessageBox.Show() isn't the only method that will give you compiler errors if you try to pass it a variable whose type doesn't match the parameter. *All* methods work that way, even the ones you write yourself. Go ahead and try typing this completely valid code into a class:

```
public int MyMethod(bool yesNo) {

    if (yesNo) {
      return 45;
    else {
      return 61;
```

One reminder—the code that calls this parameter doesn't have to pass it a variable called yesNo. It just has to pass it a boolean value or variable. The only place it's called yesNo is inside the method's code.

When the compiler gives you an "invalid al·guments" error, il means that you tried to call a method with variables whose type. didn't match tl.e meth od's parameters.

It works just fine if you pass it what it expects (a bool—`MyMethod(true)` or `MyMethod(false)`), and it compiles just fine.

But what happens if you pass it an integer or a string instead? The IDE gives you a similar error to the one that you got when you passed it a long. Now try passing it a boolean, but assigning the return value to a long or passing it on to `MessageBox.Show()`. It won't work, either—the method returns an int, not a long or the `string` that `MessageBox.Show()` expects.

You can assign anything to a variable, parameter, or field with the type object.

if statements always test to see if something's true

Did you notice how we wrote our if statement like this:

```
if (yesNo) {
```

We didn't have to explicitly say "if (yesNo == true)". That's because an if statement always checks if something's true. You check if something's false using ! (an exclamation point, or the NOT operator). "if (!yesNo)" is the same thing as "if (yesNo == false)". In our code examples from now on, you'll usually just see us do "if (yesNo)" or "if (!yesNo)", and not explicitly check to see if a boolean is true or false.



There are about 77 reserved words In C#. These are words reserved by the C# compiler: you can't use them for variable names. You'll know them all really well by the time you finish me took Here are some you've already used Write down what you thin k these words do in C#

namespace

for

class

public

else

new

using

if

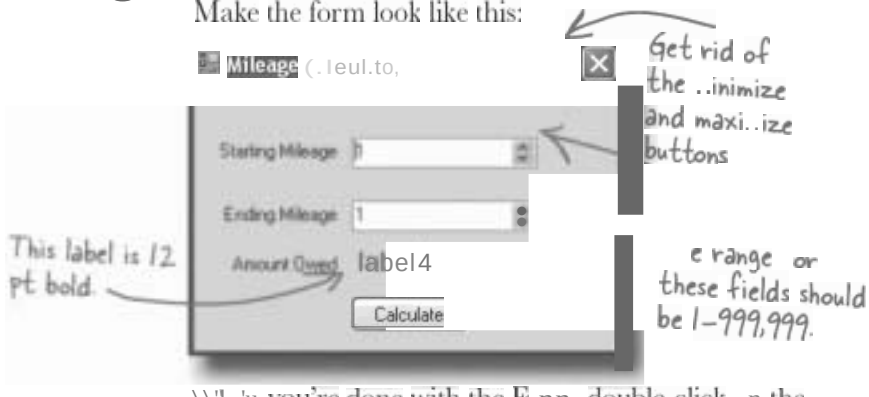
while

Answers on page 161.



Create a reimbursement calculator lor a business top. It should allow the user to enter a starting odometer reading and an ending odometer reading, From those two numbers, it will calculate how many miles she's travelled and figure out how much she should be reimbursed if her company pays her $ 39 for every mile she puts on her car

o **Start** with a new Windows project.
Make the form look like this:



\\'1,,'u you're done with the F,nn. double-click ,.n the billion to add some code 1" the I'r"j'·ct.

o **Create the** vcr-Icbles you'll **need** for the c:olculotor.
I'lll the variable ill class definition "I the 1"1'"I Forml, You need two whole lllllllll"'r variables t" track the starting Od"ll w!t'r reading and the ending "d"ll\"I"r reading. Call them `startingMileage` and `endingMileage`. You need three numbers that """" hold decimal places. Make them `doubles` and call them `milesTraveled`, `reimburseRateo` and `amoUntowed`. Set Ih,' value for `reimburseRate` III.39.

e Make your calculator work.
Addovk-inthebut tonl Click() method to:

- Make 'un' that the IItllll"T ill til< Starting Mileage field is smaller than the number ill the Ending Mileage field. If not, show a messagebox that says ..'11,,·starting mileage must be less than the ending mileage'·. Make lh,·till,· for the message box "Cannot Calculate"
- Subtract lite starting number from the ending mlml,..r and til"" multiply il by th,' reimburse rate using these lines:

```
miles'I'raveled - endingMileage -- startingMileage;
amountOwed - milesTraveled *= reirnburseRate;
label4.Text - "S" + amountovadr
```

o Run it.
Make sure it's giving the right numbers. Try changing the starting value to be higher th,llI lh., ending value and make ,ur,' it's giving you the message box.



Here's the code for the firsl partof the exercise

```
public partial class Forml : Form

    int startingMileage;
    int endingMileage;
    double milesTraveled;
    double reimburseRate    .39;
    double amount.Oveds
    public Formi() {
        InitializeComponent();

    private void buttonl_Click(object sender, EventArgs e){
      startingMileage - (int) numericUpDownl.Value;
      endingMileage - (int)numericUpDo;;n2.Value;
        if (startingMileage <- endingMileage){
            mlesTraveled = endlngMrleage -= startrngMleagej.
            amountOwed - mrlesTraveled *= rermburseRate;
            label4.Text - "$" + amountOwed;
          else {
            MessageBox.Show(
    "The starting mileage must be less than the ending mileage",
                        "Cannot Calculate Mileage");
```

int works e-eat for whole numbers. TI"s number could go all the wat up to 999,999. So a short or a byte won't cut it.

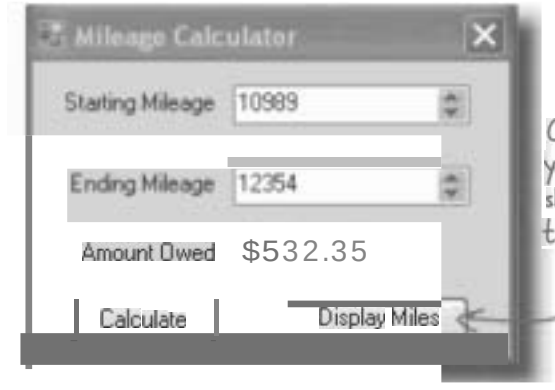
Did you remember that you have to change the decimal value from the numericUpDown control to an int?

Thi. block is supposed to figure out how many ..iles were traveled and then multiply th... by the rei..bursement rate

We used an alternate way of calling the MessageBox. Show() method here. We gave it two parameters: the first one is the message to display, and the second one goes in the title bar.

This button seems to work, but it has a pretty big problem. Can you spot it?

Now odd **another button** to **the form.**
Make it so Ihal the IUnnl,,'r "l' miles traveled is displayed on the form after you've calculated the am"11l1 '''''',1.



When you're done with lh,' Form, double-click on the Display Mii," hUll'>ıı '0' add some code u the p" ject.

One line should do **it.**
•\ll we need to do i, get the form to display the milesTraveled variable, right? So this line should do that:

```
private VOid button2 Cllck(obJect sender, EventArgs e) {
        Messagebox.Show(milesTraveled + •• miles", "Miles Traveled");
```

Run **it.**
Type in some values and " .,. wha h"PI''''',

Um, something's not **right...**
Th,. number of mile alwav matche Ihe „UO\lUI owed. Why?

COllIbillillg = with all operator

Take a good look at the operator we used to subtract ending mileage from starting mileage (-=). The problem is it doesn'fjllst subtract. it al,o assigns a value to the variable on the left side of the subtraction sign. The same thing hapP"ns in the Ii"" where we 1l111lriply nllll,her of miles traveled by the reimbursement rate. We , ho"ld replace the -= and the *= with just - and *:

```
private void buttonl_Click(object sender, EventArgs e)

  startingMileage - Ont) numericUpDownl.Value;
  endingMileage - (int)numericUpDown2.Value;
    if (startingMileage <= endingMileage){
      miles'I'raveled - endingMileage -= startingMileage;
      amount.Oved - milesTraveledG  reimburseRate;
      Iabel4,Text  -S- + amountOwed'
    else (
     MessageBox.Show("The starting mile ge number must
                be less than the c ling mileage number",
                "Cannot Calculate M leage");
```

These are called **compound operators**. Thi one subtracts startingMileage from endingMileage but also assign the new value to endingMileage and milesTraveled at the same time

Thi is better—"" your zede won't modify endingMileage and milesTraveled.

```
milesTraveled _ endingMileage - startingMileage;
amountOwed = milesTraveled * reimburseRate;
```

So can good variable names help **you out** here? Definitely! Take a dose look at what each variable i•• upposed 10 do. You already get a 101 of clues from the name mi lesTraveled- -you know that's the variable that the form i. displaying incorrectly, and you've got a good idea of how that ,'al"e ought to be calculated. So you can take advantage of that when you're looking through your code to try 1<, track down the bug. Il'd be a whole Jot harder to find Ihe problem if the incorrect line. looked like this instead:

```
mT = eM -= 8M;
aO = mT *= rR;
```

Variables named like this are essentially useless in telling you what their purpose might be.

Objects are variables. too

So far, we've looked at objects separate from other types. Il,n an "Iject is just another data type. Your code treats objects ex.actlv lik,' il treats numbers, strings, ,,,,d booleans. h uses variable t" work with them:

Usi"g a" i"1

Write a statement to declare the integer.

```
int mylnt;
```

② Assign a value to the new variable.

```
mylnt = 3761;
```

③ Use the integer in your code.

```
while (i < mylnt) {
```

Usi"g a" object

o Write a statement to declare the object.

```
Dog spot;
```

When you have a class like Dog, you use it as the type in a variable declaration statement.

o Assign a value to the object.

```
spot = new Dog();
```

o Check one of the object's fields

```
while (spot.Happy) {
```

So iTdeesn't matter if I'm working with on object or 0 vcfue. If it's going into memory, ond my program needs To use it, I use 0 variable.

o C



Objects are just one more type of variable your program can use.

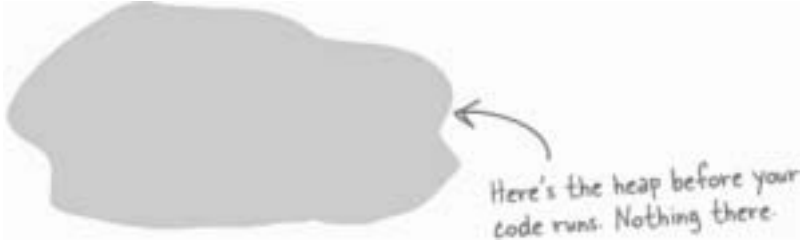
If your program needs lu work with a whole uumlx-r that's really big, use a long. If il needs a whole mUllh,'l' that's small, use a ,h"rl. If il needs a ves/ uo value, use a boolean. And if it needs something that barks ami sits, use a Dog. No matter what type of d"(,, your program needs It, wOl'k with, it'll use a variable.

Refer to your objects with refsrenee variables

That's called instantiating the object.

When you create a new obj"cl, you use code like **new** Guy, Bill that's not enough; even Ihongh that code creates a new Guy object on the heap, il doesn't give you a way to *access* Ihat object. *You ..eed a reference to the object*. So you create a **refe nce va**r**iable**: a variable of type Guy with a name, like Joe, So Joe ;. a ref nce 10 Ihe new GUy objcct you created. Anytime you want to use that particular guy, you can reference it with the reference variable called Joe,

So when you have a variable Ihat is an object type, it's a reference variable: a reference 10 a particular object. Take a look:



Thi variable is na..ed Joe, and will reference an object of type Guy

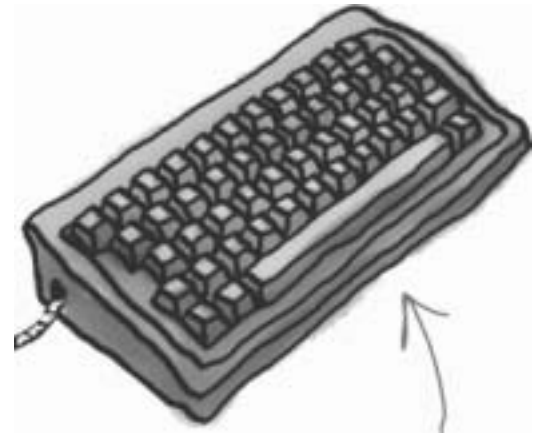
```
public partial class Forml ; Form
{
    Guy Joe;

    public Form! ()
    {
        InitializeComponent() ;

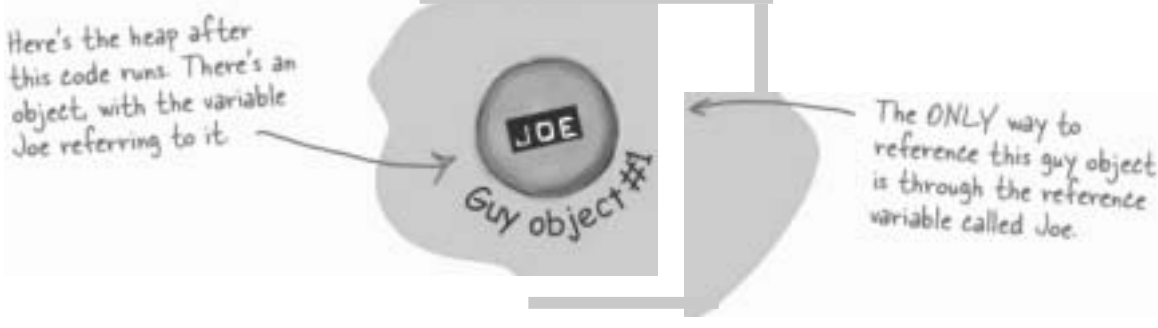
        Joe = new Guy();
    }
```

This is the reference variable...

...and this is the object that Joe now refers to

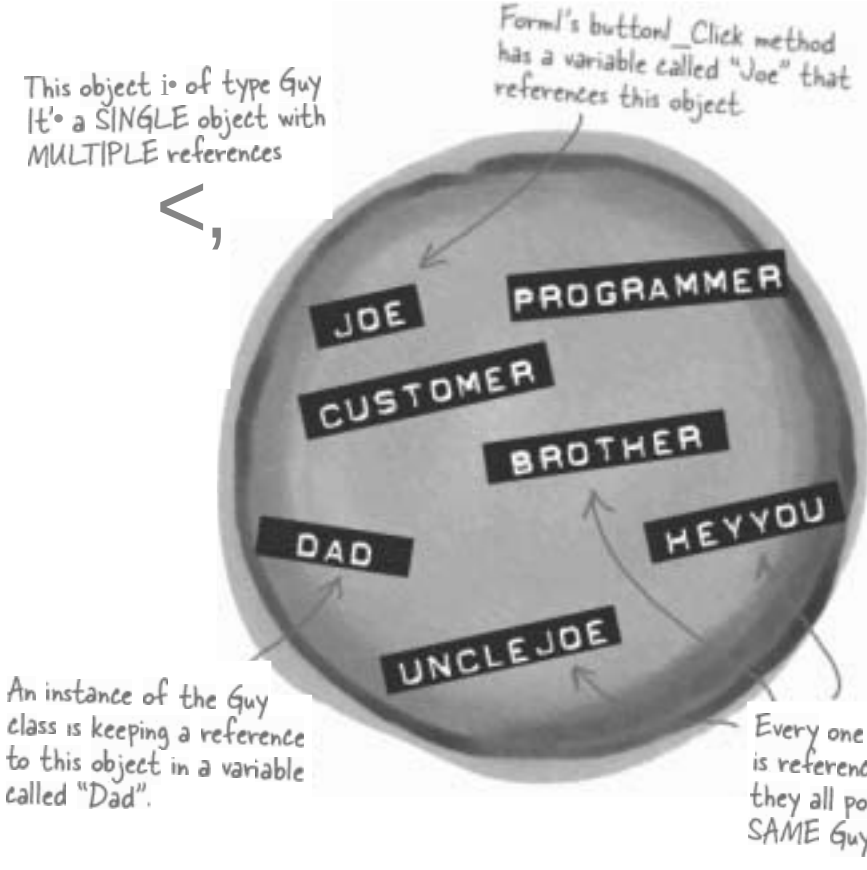


Creati.g a reference is like maki..g a label with a **label maker**—instead of sticking it on your stuff, you're using it to label an object so you can refer to it later.



References are like labels for your object

In your kitchen, you probably have a container of salt and sugar. If you switched their labels, il would make for a pretty disgusting meal—even though the labels changed, the contents of the containers stayed the same. ***References are like l..bels.*** You can move labels around, poinl Ihe", al differenl things, bUI it's Ihe **object** that dictates what methods "ml data are available, not the reference itself.



When your co,lc **needs** 10 **work with** an ohjcct m memory, **it** uses a rejerence, **which** is a varial.le **whose** type is a class of **the object** it's going to point to. A re lerenee is **like** a **label** that your **code** uses to talk **about** a **specific** ol,jert.

There are lots of different references to this same Guy, because a lot of different methods use him for different things. Each reference has a different name that makes sense in its context.

You never refer 10 your object directly. For example, you can't write code like Guy . GiveCash ( ) if GUy is your object ope.The C# compiler doesn'l know which **Guy** you're talking about, since vou might have several instances of Guy on the heap. So you need a reference variable, like `joe`. that you assign 10 a specific instance, like GUy `joe _ new Guy ()`.

Now, you can call methods, like `joe.GiveCash ()`. `joe` refers 10 a specific instance of the Guy class, and your C# compiler knows exactly which instance to use. And, as you saw above, you might have ***multiple l..bels pointing to tne same instance***. So you could say `Guy dad _ joe`. and then call `dad.Gi vecash ()`. 11,al', okay, too—that's what Joe's kid does every day.

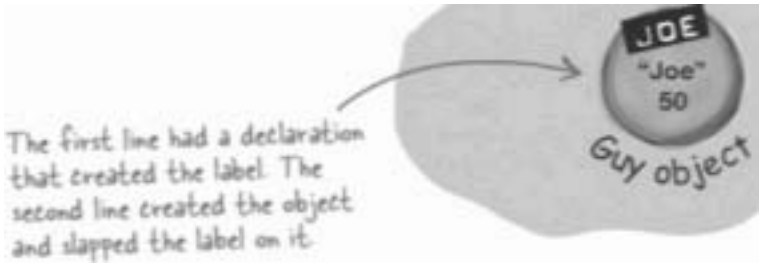
If there aren't any l\Iore references, your object gets garbage collected

If all of lhe lal...ls come off of an object, "'' programs can access that object anymore. TII;,r means C# can mark lhe object for **garbage coll..ction**. That', when C# gets rid of any u"...ferenced objects, atld reclains ,lle memory those objects took up for your program's nse.

For an **object** to stay in the heap. it has to **be** refe,·enced. When the last reference to the ol,ject disappears. so **does** the **object.**

o **Here's some code that** c-eeres **an object.**

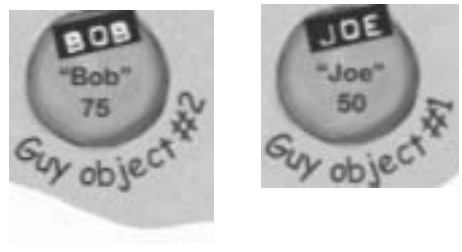
```
Guy joe • nev GuyO
  , Name • "Joe", Cash - 50 );
```



o **Now** I.t's erecte a **second object.**

```
Guy bob = new Guy()
  { Name = "Bob", Cash = 75 };
```

Now we have two Guy object instances, and two reference variables: one for each Guy.



o **Let's** toke **the reference** to **the** first **object,** ond **change** it to point at **the** s.cond **object.**

```
joe • bobl
```

Now joe .. pointing to the sa...e object as bob

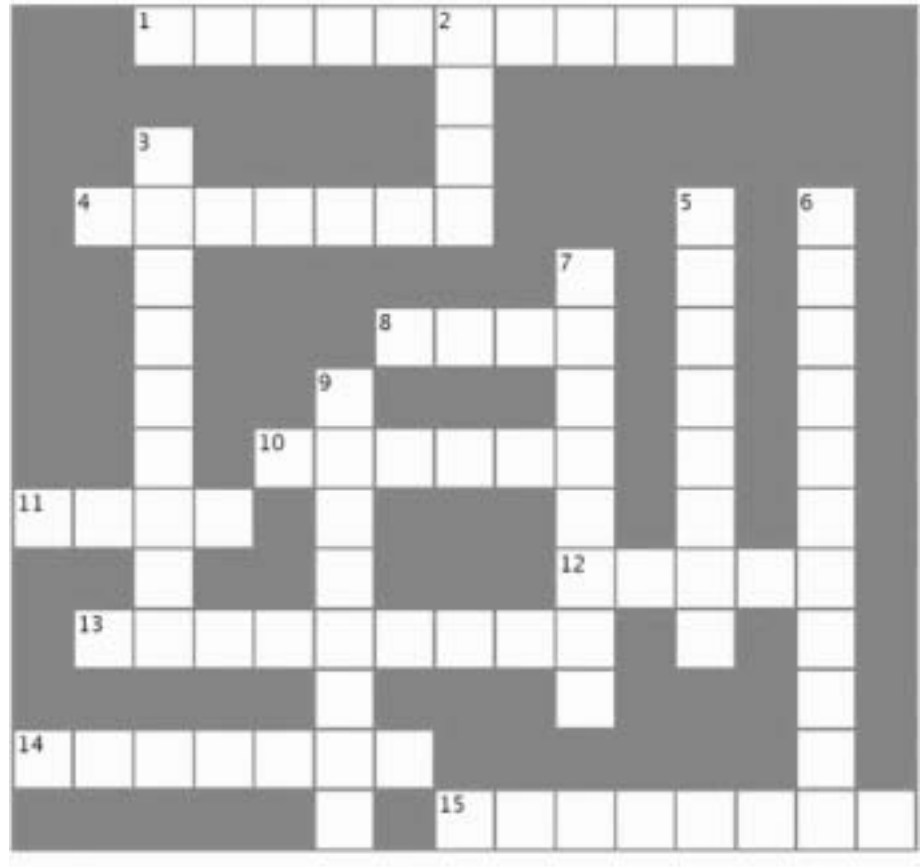




Typecross

Take a break, and sit back and give your right brain something to do. It's your standard crossword; all of the solution words are from this chapter.

When you're done, turn the page, and take on the rest of the chapter.



Across

1 You can combine the variable oecreranon and the _ intoonestatement

4. When an object no longer has any rejerences pointing 10 it, it's removed from the heap using collection

8. A variable cedeatco always starts Wlth this

10, The type that holds the biggest numbers

11 The type met stores a single letter or number

12. If yoo never setthis for a variable. your program won', compile

13, A variable that points to an object

14, What (int) ooes in this line of code x = (inl) y;

15. The four whole number types thaI only hold positive numbers

Down

2, The secoro part 01 a variable declaration

3, Variable names like my Big Number and bobTheDog use this style of capitalization

5, What your program uses to work With data that's in memory

6. What you're doing when you use the + operator to stick two stnngs together

7. 'namaspace'', "lor', "while", "using" and 'new" are sxampes 01 words

9. Every object has this method that converts it to a string

Answers on page 162"

Multiple refereaees and their side effects

You've got to be careful when vou start moving around reference variables. Lots of limes. il might seem like you re simply pointing a \'ariable to a dilfetent object. But, you could end up removing all teferences to anolher object in Ihe pro" ess. That's not a bad thing, btlt it may not b,' what you intended. Take a look:

```
Dog rover = new Dog();
rover.Breed = "Greyhound";
```

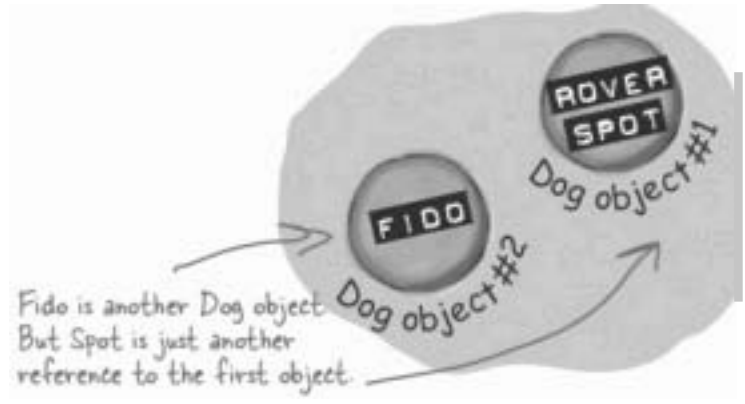


Rover i. a Dog object with a Breed field called Greyhound

Objects: 1

References: 1

```
Dog fido = neW Dog();
fida,Breed = "Beagle";
Dog spot = rover;
```

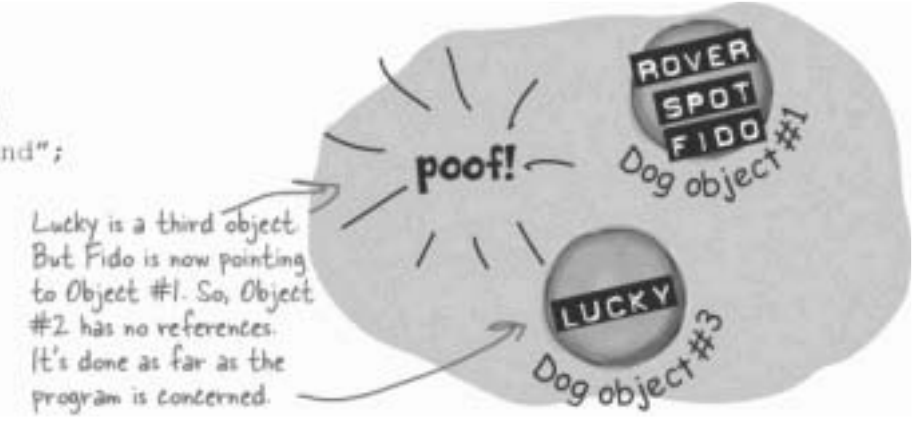


Fido is another Dog object
But Spot is just another reference to the first object.

Objects: 2

References: 3

```
Dog lucky = new Dog();
lucky.Breed = "Dachshund";
t.ida = rover;
```



Lucky is a third object
But Fido is now pointing to Object #1. So, Object #2 has no references.
It's done as far as the program is concerned.

Objects: 2

References: 4



your penCil

Now it', your turn. Here's one long block of code. Figure out how many object, and references there are at each stage. On the right·hand side, draw a picture at the objects and labels in the heap.

O
```
Dog rover - new Dog();
rover.Breed - "Greyhound":
Dog rinTinTin - new Dog():
Dog fida - new Dog():
Dog quentin - Fido:
```

Objects:_____

References:____

O
```
Dog spot - new Dog():
spot.Breed - "Dachshund";
spot - rover:
```

Objects:_____

References:____

```
Dog lucky - new Dog():
lucky.Breed - "Beagle";
Dog charlie - fido:
fLdo - rover;
```

Objects:_____

References:____

O
```
rinTinTin - lucky;
Dog laverne - new Dog();
laverne.Breed - "pug";
```

Objects:_____

References:____

O
```
charlie - laverne;
lucky - rinTinTin;
```

Objects:_____

References:____

Sharpen your pencil Solution

Now it', your turn. Here's one long block of code, figure out how many object, and references there are at each stage. On the right·hand, ide, draw a picture of the object' and labels in the heap.

```
Dog rover - new Dog();
rover.Breed - "Greyhound";
Dog rinTinTin - new Dog();
Dog fido - new Dog{);
Dog quentin = Fido;
```

Objects: 3

References' 4

One new Dog object i, created but Spot is the only reference to it. When Spot i. set = to Rover, that object goes away

```
Dog spot _ new Dog();
spot.Breed = "Dachshund";
spot = rover;
```

Objects: 3

References:_ '_

Here a new Dog object is created, but when Fido is set to Rover, Fido's object from #1 goes away

```
Dog lucky - new Dog();
lucky.Breed   "Beagle";
Dog charlie   fida;
fido - rover;
```

Charlie was set to Fido when Fido was still on object #3. Then, after that, Fido moved to object #1, leaving Charlie behind

Objects: 4

References:l

```
rinTinTin —Lucky;
Dog laverne = new Dog();
laverne.Breed = "pug";
```

Dog #2 I ' its last ..eference, and it went away.

poof!

Objects: 4

References:_ 8 _

When Lucky ..oved to Ri" Tin Ti,,', object, the old Lucky object disappeared

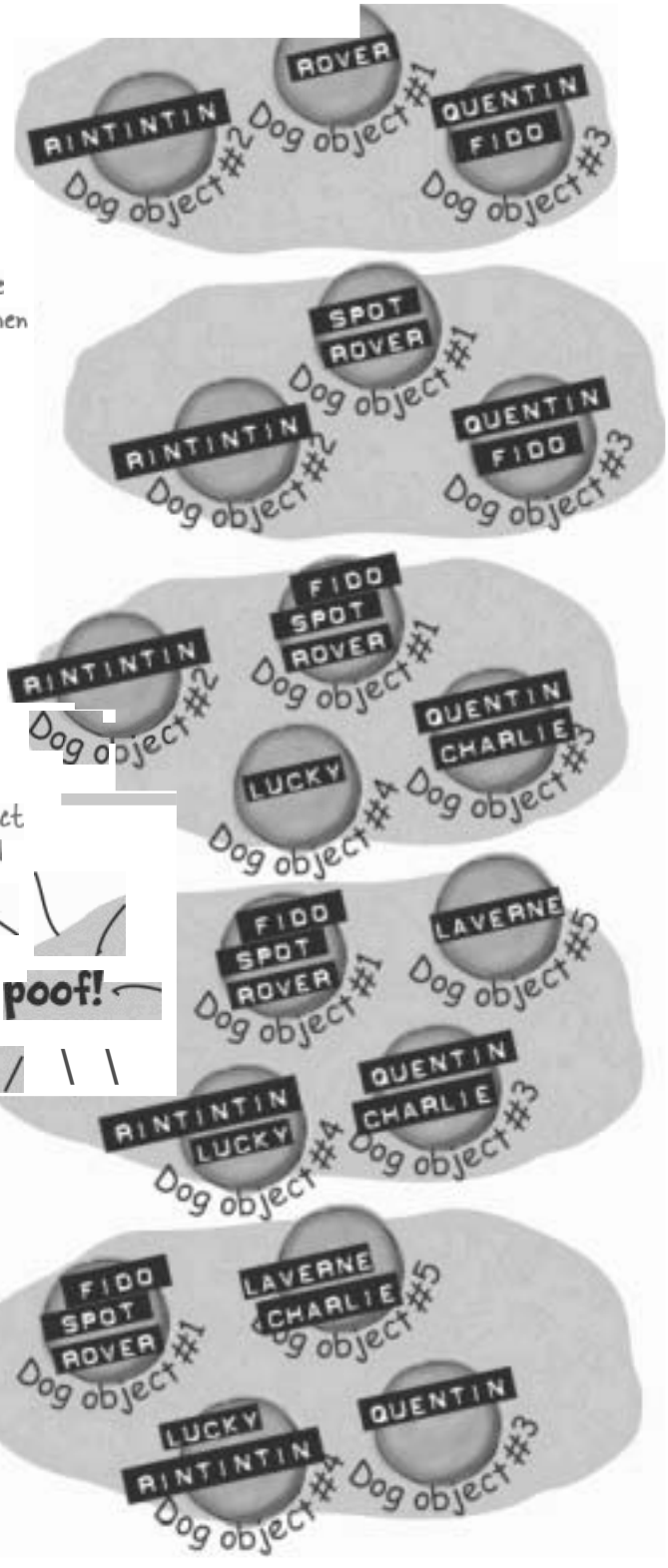
```
charlie   laverne;
lucky = rinTinTin;
```

Objects: 4

References: 8

— —

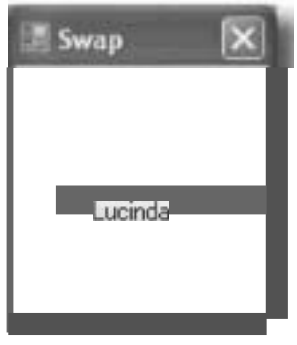
Here the ..eferences move around but no new objects are created And setting Lucky to Ri" Tin Tin did nothing because they already pointed to the same object.





Create a program with an elephant class. Make two elephant instances and then have them swilch properties. *without* ge1llng any Etepnantnetences garbage collected

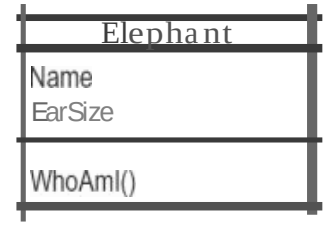
O **Start with** a new Windows Application project.
Make the form look lik" this:



Clicking on the "Lucinda" button calls lucinda WhoA 10 which displays this .. essagebox



Here's the class d iagram for I he Elephant class you need to create



This gives you a clue as to what WhoA..IO should return: a single string with this message

O Create **the** Elephant class.
.\dd an Elephant class to the project. Have a I,ook at the Elephant class diagram - you'll need an int li"kl calll-d EarSize and a String field called Name. (Make sure hOlh are 1'"I,li,'.) Then add a llwlltod called WhoAmI () lhat displays a messagebox lhat tells y(" the na",e and l-ar size of the elephant.

3 Create two elephant instances and a **reference.**
Add two Elephant fields 10the Forml class (in the area riuhr below the class declaration) named Lloyd and Lucinda, Initialize them so they h"W'll", right name and "," -size. Here are the Elephant objec initializers '0 ;ltld to your form:

```
lucinda - new Elephant() { Name   "Lucinda", EarSize = 33  I;
lloyd - new Elephant() { Name = "Lloyd", EarSize = 40  );
```

O Make **the "Lloyd"** and **"Lucinda"** buttons work.
Have the Lloyd buttor call lloyd.WhoAmI() "",11h!' Lucinda buttor call lucinda.WhoAmI(),

O Hook up **the** SWClp button.
IIer,.'" th.. hlOrd plOrt. Make the Swap button ***exchange*** the two references, so that when you click Swap, the Lloyd and Lucinda variables swap objects .md a "Objects swapped" box is displayed. Test out your program by clicking the Swap buulln and the" clicking lll<' other two buttons. The first lilll!' you click Swap, th.. Lloyd button should 1'"1' lip Lucinda's messagebox, and the Lucinda buttor ,1",,,ld 1'"1' "I' Lloyd's ",essagebox. If you click the Swap button again. ..verything should go back.

C# garbage collects any object with no references to it. So here's your hint: If you want to pour a glass of beer into another glass that's currently full of water, you'll need a third glass to pour the water into••.



Exercise Solution

Create a program with an elephant class. Make two elephant instances and then have them switch properties, *without* getting any Elephant instances garbage collected.

```
using System.Windows.Forms;

class Elephant {

  public int EarSize;
  public String Name;

  public void WhoAmI() {
     MessageBox.Show("My ears are " + EarSize + " inches tall.",
        Name + " says...");
  }
}
```

This is the Elephant class definition code in the Elephant.cs file we added to the project. Don't forget the "using System.Windows.Forms;" line at the top of the class. Without it, the MessageBox statement won't work.

Here's the Form1 class code from Form1.cs

```
public partial class Form1 : Form {

    Elephant lucinda;
    Elephant lloyd;

    public Form1()
    {
        InitializeComponent();
        lucinda = new Elephant()
            { Name = "Lucinda", EarSize = 11 };
        lloyd = new Elephant()
            { Name = "Lloyd", EarSize = 40 };


    private void button1_Click(object sender, EventArgs e) {
        lloyd.WhoAmI();


    private void button2_Click(object sender, EventArgs e) {
        lucinda.WhoAmI();


    private void button3_Click(object sender, EventArgs e) {
        Elephant holder;
        holder = lloyd;
        lloyd = lucinda;
        lucinda = holder;
        MessageBox.Show("Objects swapped");
    }
}
```

If you just point Lloyd to Lucinda, there won't be any more references pointing to Lloyd and his object will be lost. That's why you need to have the Holder reference hold onto the Lloyd object until Lucinda can get there.

There's no new statement for the reference because we don't want to create another instance of Elephant.



BRAIN POWER

Why do you think we didn't add a SwapO method to the Elephant class?

fwo references means TWO ways to change an object's data

Besides losing all the reference I" an ◁bject, when you have rIlultil'k references I" an object, you can U"illl""1;'mally change all object. In other words, ow' reference to an "hj""1 may ***change*** thai object, while another reference 10lhal "hj,-c! ha' ***no idea*** that something has changed. Watch:



O Add another bliiton 10your form.

2 Add this code for the button. Can you guess what's going to happen when you click it?

```
private void button4_Click(object sender, EventArgs e)
{
    lloyd = Lucinda;
    lloyd.EarSize = 4321;
    lloyd.WhoAmI();
}
```

This statement says to set EarSize to 4321 on whatever object the lloyd reference happens to point to.

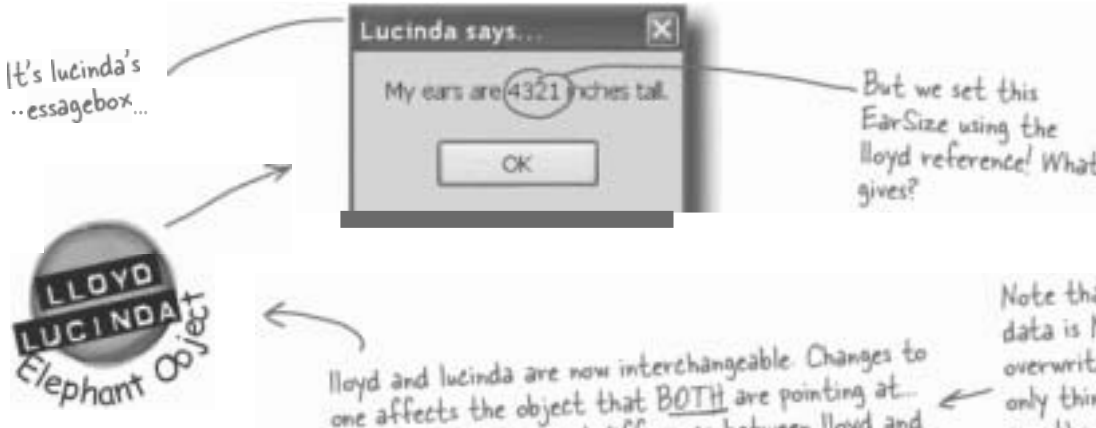
After this code runs, both the lloyd and lucinda variables reference the SAME Elephant object.

You're calling the WhoAmI() method from the lloyd object.

But lloyd points at the same thing that lucinda does.



3 OK, go ahead and click the new hUllo11. Wait a second, that's Ill,· Lucinda messagebox. Did,,'(we call the WhoAmI() m"lh'HI frolll Lloyd?



A special case: arrays

If you have to keep track of a lot of data of the same type, like a list of heights or a group of dogs, you can do it in an array – like you used when you made the sandwich menus for Sloppy Joe. What makes an array special is that it's a **group of variables** that's treated as one object. An array gives you a way of storing and changing more than one piece of data without having to keep track of each variable individually. When you create an array, you declare it just like any other variable, with a name and a type:

You could combine the declaration of the myArray variable with its initialization – just like any other variable. Then it'd look like this:

```
bool[] myArray = new bool[15];
```

You declare an array by specifying its type, followed by square brackets.

```
bool[] myArray;
myArray = new bool[15];
myArray[4] = true;
```

This array has 15 elements within it.

You use the new keyword to create an array because it's an object. So an array variable is a kind of reference variable.

This line sets the value of the fifth element of myArray to true. It's the fifth one because the first is myArray[0], the second is myArray[1], etc.

Use each element in an array like it is a normal variable

Here's an example of code that declares and fills up an array – and what's happening on the heap when you do it. The first element in the array has an **index** of zero.

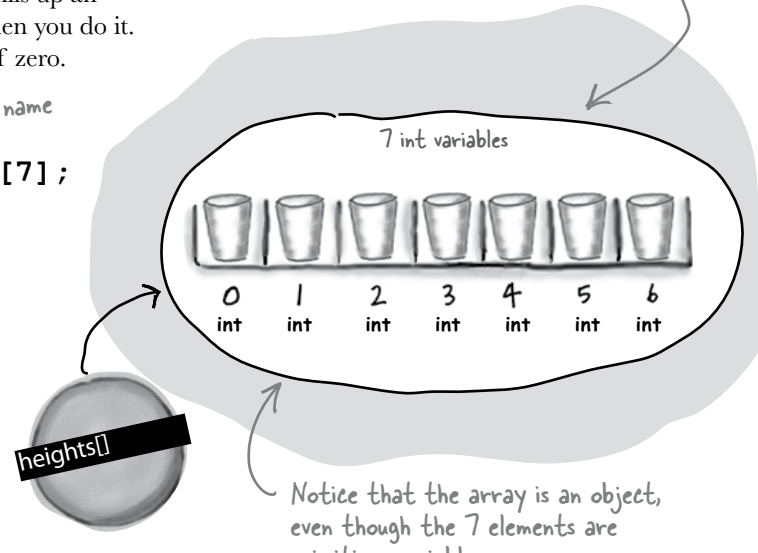
The type of each element in the array.

name

```
int[] heights;
heights = new int[7];
heights[0] = 68;
heights[1] = 70;
heights[2] = 63;
heights[3] = 60;
heights[4] = 58;
heights[5] = 72;
heights[6] = 74;
```

You reference these by index, but each one works essentially like a normal int variable.

In memory, the array is stored as one chunk of memory, even though there are multiple int variables within it.



Notice that the array is an object, even though the 7 elements are primitives variables.

Arrays can contain a bunch of reference variables, too

You can create an array of object references just like you create an array of numbers or strings. Arrays don't care what the type of variable is that it stores; it's up to you. So you can have an array of `int`s, or an array of `Duck` objects, with no problem.

Here's code that creates an array of 7 **`Dog`** variables. The line that initializes the array only creates reference variables. Since there are only two `new Dog()` lines, only two actual instances of the **`Dog`** class) are created.

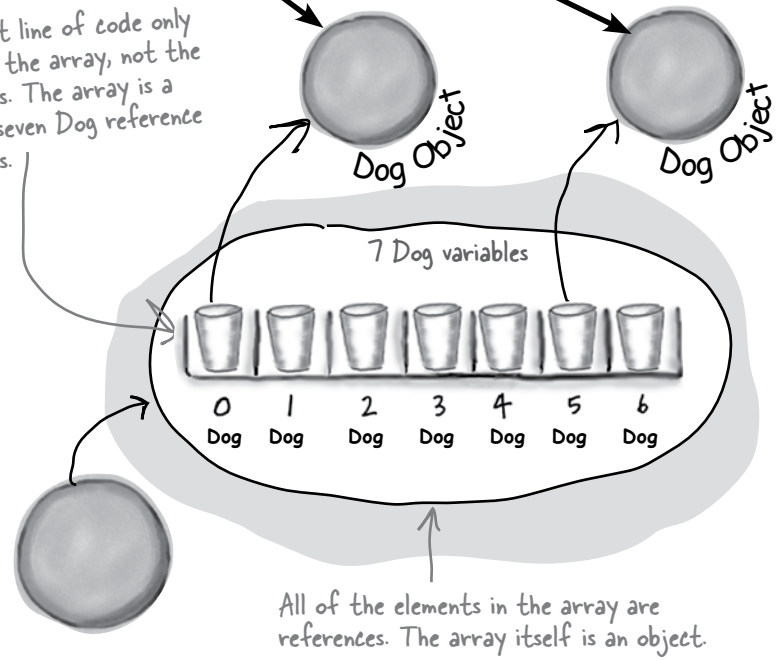
```
Dog[] dogs = new Dog[7];
dogs[5] = new Dog();
dogs[0] = new Dog();
```

This line declares a dogs variable to hold an array of references to Dog objects, and then creates a 7-element array.

These two lines create new instances of Dog() and put them at indexes 0 and 5.

When you set or retrieve an element from an array, the number inside the brackets is called the index. The first element in the array has an index of zero.

The first line of code only created the array, not the instances. The array is a list of seven Dog reference variables.

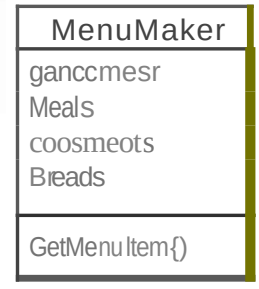


All of the elements in the array are references. The array itself is an object.

Welcol\le to Sloppy Joe's Budget House o' lIiscoul1t Sal1dwiches!

Sloppy Joe has a pil,' of meat, a whole lotta bread, and more coudilll"I\" than you can shake a stick at. But what he doesn't have is a menu! Can \'011 build a program that makes a new *random* menu for him every day?





O Stort a new project ond add a MenuMoker class

If you need to build a 1llt'1l11, you 1I,'ed ingredients And arrays would be perfect for those lists. We'll also need some way or choosing random ingredients 10 combine together into a sandwich. Luckilv. the .NET Framework has a built-in class called **Random** that generate random numbers. Sc "'t'll have f<ll' fields iu ,,,u' class: a Raml"",i/"r fi.,ld Iha, hold, a reference to a Random object, ami ,Im',- arrays ,,l' `strings` to hold the meats, condiments, and breads.

The class has three fields to store three different arrays of strings. It'll use them to build the random menu items.

The field called Randomizer holds a reference to a Random object. Calling its Next() ..ethod will generate random numbers.

```
public class MenuMaker {
    public Random Randomizer;

    string[] Meats = t "Roast beef", "Salami", "Turkey", "Ham", "Pastrami" );
    string[] Condiments = { "yellow mustard", "brown mustard",
                "honey mustard", "mayo", "relish", "french dressing" };
    string[J Breads = t "rye", "white", "wheat.., "pumpernickel",
                "italian bread", "a roll" I;
```

Re..e..ber, use square brackets to access a member of an array. The value of Breads[2] is "wheat"

O Add a GetMenItem() method to generate a random sandwich to the etess

The poinl of the class is '0 generate sandwiches, so let's add a method '0 do exac"I\, that. 1,'ll use the Random object's `Next()` ",ethod I" choose a random meat, coı,di",,', 'I and bread fr<"" each array. \When you pass an `int` parameter I,. `Next(I` ' the meth<"I returns a ra, ,do", that's less ,han lha, parameter. Sc if your Random object is ,'"ll"r! `Randomizer`, lh,·" calling `Randomizer.Next(7)` will n-mrna random numbe-r between 0 and li.

So how do you know what parameter '0 pass inl" the `Next()` method? \\'ell. that's easy—just pas in each array's `Length`, Thai will return tl,,' i"d,'x of a random ii"'" i" ,he arrav.

The GetMenuItem O ..ethod ·eturns a string that contains a sandwich built from ..andom ele..ents in the three arrays

```
public string GetMenuItem() {
    string randomMeat = Meats[Randomizer.Next(Meats.Length)];
    string randomCondiment • Condiments IRandomiaer.Next(Condiments.Lenqthl I;
    string randomBread - Breads[Randomizel'.Next(Breads,Lengthl1;
    return randomMeat + " with " + randomCondiment + " on " + randomBread;
}
```

The ..ethod puts a random ite.. from the Meats array into ..andomMeat by passing Meats.Length to the Random object's Next() ..ethod. Since there are 5 items in the Meats array, Meats.Length is 5, so Next(5) will return a random number between 0 and 4.



How it works...

The randomizer.Next(7) method gets a random number that's less than 7. Meats.Length returns the number of elements in Meats. So `randomizer.Next(Meats.Length)` gives you a random number that's greater than or equal to zero, but less than the number of elements in the Meats array.

```
Meats[Randomizer.Next(Meats.Length)]
```

Meats is an array of strings. It's got five elements, numbered from zero to 4. So Meats[0] equals "Roast Beef", and Meats[3] equals "Ham".

I ect all my meals of Sloppy .Ice's!



3 Build **your form**

.\<1<1 six labels (0 the I',nn, `label1` through `lable16`. Then add code to set each label's `Text` property using a `MenuMaker "bj",-(.` You'll need to initialize the object using a new in'[,ll"'!' or the R,llul,,,,, class. Here's the code:

Use an object initializer to set the MenuMaker object's Randomizer field to a new instance of the Random cla...

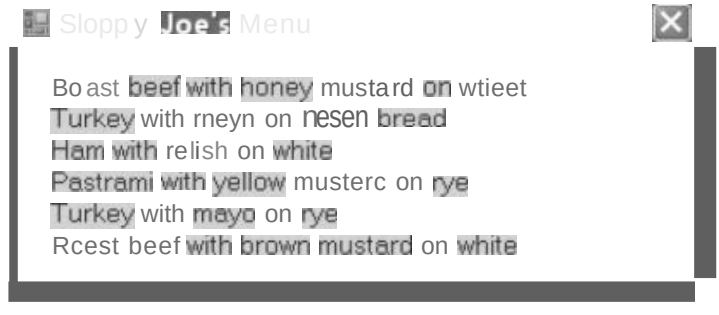
```
pUblic Form1() (
    InitializeComponentll;

    MenuMaker menu'' new MenuMaker()    Randomizer = new RandomO );

    label1.Text    menu.GetMenuItem();
    label2.Text = menu.GetMenuItem();
    label3.Text • menu.GetMenuItem();
    label4.Text • menu.GetMenuItem();
    label5.Text • menu.GetMenuItem();
    label6.T..xt =menu.GetMenuItem();
```

Now you're all set to generate six different ..andom sandwiches using the GetMenultemO ..ethod

When you ..un the progra... the six labels show six different random sandwiches.



Objects use references to talk to each other

So far, you've seen forms talk to objects by using reference variables to call their methods, and check their fields. Objects can also call each others' methods using references, too. In fact, there's nothing that a form can do that your objects can't do, because your form is just another object. And when objects talk to each other, they use a useful keyword that they have in common, this. Any time an object uses the this keyword, it's referring to itself—it's a reference that points to the object that called it.

Here's a method to tell an elephant to speak

Let's add a method to the Elephant class. Its first method is a message from an elephant. Its second method is the elephant that said it:

```
public void TellMe(string message, Elephant whoSaidIt) {
    MessageBox.Show(whoSaidIt.Name + " says: " + message);
}
```

Here's what it looks like when it's called:

```
Elephant lloyd = new Elephant() { Name = "Lloyd", EarSize = 40 };
Elephant lucinda = new Elephant() { Name = "Lucinda", EarSize = 33 };
lloyd.TellMe("Hi", lucinda);
```

We called Lloyd's TellMe() method, and passed it two parameters: "Hi" and a reference to Lucinda's object. The method uses its whoSaidIt parameter to access the Name parameter of whatever elephant was passed into TellMe() using its second parameter.

Here's a method that calls another method

Now let's add a SpeakTo() method to the Elephant class. It uses a special keyword: this. That's a reference that **lets an object talk about itself**.

```
public void SpeakTo(Elephant talkTo, string message)
    talkTo.TellMe(message, this);
```

This method in the Elephant class calls another elephant's TalkTo() method. It lets one elephant communicate with another one.

Let's take a closer look at how this works.

```
lucindd.SpeakTo(lloyd, "Hello");
```

When Lucinda's SpeakTo() method is called, it uses its talkTo reference parameter to call Lloyd's TellMe() method.

```
talkTo.TellMe(message, this);
```

Lucinda uses talkTo (which has a reference to Lloyd) to call TetiMeO

this is replaced with a reference to Lucinda', object

```
lloyd.TellMe(message, [a reference to Lucinda]);
```

So Lloyd acts as if he was called with ("Hello", lucinda), and shows this message:



Where no object has gone before

There's another important keyword ,hal you'll use with objects. ""hell vou create a new reference and don't set it 10 anything, it has avalue. It starts „11' set '" null. "hid, means it's not pointing 10 anything.

```
Dog fido;
```

Right now, the··e's only one object. The f.do reference ,• ·et to null.

```
Dog lucky = new Dog();
```



Now that fido's pointing to an object, it's no longer equal to null.

```
fido = new Dog();
```



When we set lucky to null, it's no longer pointing at its object, so it gets garbage collected.

```
lucky = null;
```



there are no Dumb Questions

0 : One more **time—my form** is **an** object?

A: Yes' That's why your class code starts With a cress declaration. open upcode lor a form and see for yourself. Then open up Program.cs in any program you've wmen so lar and look inside the InitializeComponent() method—you'll find *new Forml, (l'.

0...: Why would I ever **use** null?

A: There are a few ways you see null used in typical programs The most common way is testing for it-

```
if (lloyd -- null)
```

That test will return true if the lloyd reference is set to null.

Another way you'll see the null keyword used is when you wanf your object to get garbage collected. If you've got a reference to an object and you're finished with the object, setting the reference to nu 11 will immediately mark rt for collection (unless there's another retereoce at it somewhere.)

0: You **keep** talking .bout garbage **collecting,** bul **what's** actually **doing** Ihe collecling?

A: Remember how we talked about the **Common Language** Runtime (or CLRj back in the beginning of the first chapter? That's the virtual machine that runs all NET programs A *virtual machine* is a way for to isolate running programs from the rest of the operating system One thing that virtual machines do is manage the memory thai they use. That means that it keeps track of all of your objects, figures out when the last reference to the object disappears and frees up the memory eat it was using

there are no Dumb Questions

Q: I'm ltill notlurl l **get how references w**ork.

A: References are the way you use all of the methods and fields I'l an object If you create a reference toI Dog object you can then use that reference 10 access any methods you've created for the Dog object If you have (non-static) a method called Dog Bark() or Dog Beg(), you can create a reference called spot Then you can use that to access spot Bark() ot spot.Beg(). You could also change information in the fields for the object using the reference So you could change a Breed field using spot Breed

Q: Wait, then doesn't that ....an that every ti.... , change a value through , reference rm changing it for . . of the other references to that object too?

A: Yes If rover is , reference 10 the same object as spot, changing rover Breed to "beagle" would make it so that spot.Breed was "beagle."

Q: Go **back** to th,t stuffabout **value type·. Now, why can't I change a s**m.11 **number from a** bigger **type if it's small enough?**

A: Okay The thing about variables is they assign a size 10 your number no matlet how big its value is So if you name a variable and give it a `long` type even though the number is really small, (like, say, 5) C# sets aside enough memory for dlo get really big When you think about it, that's really useful. After all, they're called variables because they change all the time

C# assumes you know what you're doing and you'. . not going 10 give a variable a type that you don't need So even though the number might not be big now. there s a chance that after some math happens, it'll change and C# gives it enough memory 10 handle whatever type of number you call it

Q: Remind me again—what does "`this`." do?

A: `this` is a special variable that you can only use inside an object. When you re inside a class, you use `this` to refer 10 any field Of method of that particular instance It's especially useful when you re working with an class whose methods call other classes One object can use it to send , **reference 10 itself** 10 another object So if Spot calls one of Rover's methods passing `this` as a parameter, he's giving Rover a reference 10 the Spot object

Any time j-ou've got **code** in an **object** thaI's going to I.. instanliate**d, the instance** can **use the** special this **variable that has** a **refer**-enee 10 itsel],

BULLET POINTS

- There are **value types** lor numbers that hold different sees ot numbers The biggest numbers should be of lhe type, `long` and the smallest ones (up to 128) can be declared as `bytes`.
- When you declare a variable you ALWAYS give a type Sometimes you conooe it with setting the value
- Every value type has a see. and you can't put a value of a bigger type inlO a smaller variable, no matter what the adual size of the data is
- When you're Using l,leral values, use the F suffix to indicate a aoar (15,6F) and M for a decimal (36.12M)
- The compile1 won't let you set a variable equal to a value of a different type unless you cast it
- There are some words that are reserved by the language and you eent name your variables Wlh them They're words like, `for`, `while`, `using`, `new`, and others that do specnc things in the language
- References are like labels you can have as many references to an object as you wanl and they all rerer to the same thing
- If an ocect ooesnt have a rereeercs , Il gets garbage CCaecieo



Sharpen your pencil

Here's an array of Elephant object, and a loop that will go through it and find the one with the biggest ears. What', the value of the biggestEars.Ears **after** each iteration of the for loop?

```
private void button1_Click(object sender, EventArgs e)
{
    Elephant[] elephants = new Elephant[7];
    elephants[0] = new Elephant() { Name = "Lloyd", EarSize = 40 };
    elephants[1] = new Elephant() { Name = "Lucinda", EarSize = 33 };
    elephants[2] = new Elephant() { Name = "Larry", EarSize = 62 };
    elephants[3] = new Elephant() { Name = "Lucille", EarSize = 52 };
    elephants[4] = new Elephant() { Name = "Lars", EarSize = 44 };
    elephants[5] = new Elephant() { Name = "Linda", EarSize = 37 };
    elephants[6] = new Elephant() { Name = "Humphrey", EarSize = 45 };

    Elephant biggestEars = elephants[0];
    for (int i = 1; i < elephants.Length; i++)
    {
        if (elephants[i].EarSize > biggestEars.EarSize)
        {
            biggestEars = elephants[i];
        }
    }
    MessageBox.Show(biggestEars.EarSize.ToString());
}
```

We're creating an array of 7 Elephant() references

Every array starts with index 0, so the first elephant in the array is Elephants[0]

This line makes the biggestEars reference point at whatever elephant elephants[i] points to.

Be careful—this for loop starts with the **second** element of the array (at index 1) and iterates six times until i is equal to the length of the array

Iteration #1 biggestEars.Earssize = _____

Iteration #2 biggestEars.EarSize = _____

Iteration #3 biggestEars.EarSize = _____

Iteration #4 biggestEars.EarSize = _____

Iteration #5 biggestEars.EarSize = _____

Iteration #6 biggestEars.EarSize = _____

Sharpen your pencil Solution

Here's an array of Elephant object, and a loop that will go through it and find the one with the biggest ears. What's the value of the biggestEars.Ears **after** each iteration of the for loop?

```
private void button1_Click(object sender, EventArgs e)

  Elephant[] elephants = new Elephant[7];
  elephants[0] = new Elephant() { Name = "Lloyd", EarSize = 40 };
  elephants[1] = new Elephant() { Name = "Lucinda", EarSize = 33 };
  elephants[2] = new Elephant() { Name = "Larry", EarSize = 42 };
  elephants[3] = new Elephant() { Name = "Lucille", EarSize = 32 };
  elephants[4] = new Elephant() { Name = "Lars", EarSize = 44 };
  elephants[5] = new Elephant() { Name = "Linda", EarSize = 37 };
  elephants[6] = new Elephant() { Name = "Humphrey", EarSize = 45 };

  Elephant biggestEars = elephants[0];
  for (int i = 1; i < elephants.Length; i++)

    if (elephants[i].EarSize > biggestEars.EarSize)

      biggestEars = elephants[i];

  MessageBox.Show(biggestEars.EarSize.ToString());
```

Did you remember that the loop starts with the second element of the array? Why do you think that is?

The biggestEars reference is used to keep track of which element we've seen while going through the for loop has the biggest ears so far.

Iteration #1 biggestEars.EarSize = 40

Iteration #2 biggestEars.EarSize = 42

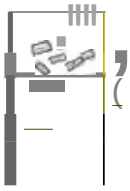
Iteration #3 biggestEars.EarSize = 42

Iteration #4 biggestEars.EarSize = 44

Iteration #5 biggestEars.EarSize = 44

Iteration #6 biggestEars.EarSize = _____

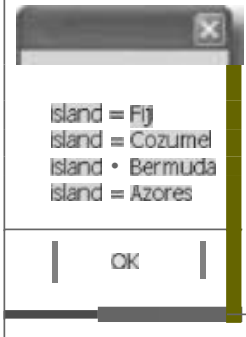
The for loop starts with the second elephant and compares it to whatever elephant biggestEars points to. If its ears are bigger, it points biggestEars at that elephant instead. Then it moves to the next one, then the next one... by the end of the loop biggestEars points to the one with the biggest ears.



Code Magnets

The code for a bullon is all scrambled up on lhe fridge. Can you reconstruct the code snippets to make a working melhod lhat produces the output li,ted below?

```
int y " 0;
refNum • index[y);
iSlands(OJ = "Bermuda";
iSlandS[lJ = "Fiji";
islands[2] = "Azores";
islands[3] = "Cozumel"
int refNum;
while (y c 4) |
1result += islands[reINumj;
MessageBox.Show(result);
index[0] " 1;
inde"tl1 = 3'
index[2] " 0,
index[3] = 2
}
}
String[1 islands " new StringI4];
result += "\nisland = ";
Lintrj index = new int(4j;1
y " v + 1;
private void buttonj Click (object sender, EventArgs e)
|
String result = "".
```



Code Magnets Solution

The code for a button is all scrambled up on the fridge. Can you reconstruct the rode snippets to make a working method that produces the oulput listed below?

```
private VOid buttonl_ell_ck (object sender, EventArgs e)
{
String result = "";

Int// index = new int[4];
index[0] = 1;
index[1] = 3;
index[2] = 0;
index[3] = 2;

String[] islands = new String[4];
islands[0] = "Bermuda";
islands[1] = "Fiji";
islands[2] = "Azores";
islands[3] = "Cozumel";

int y = 0;
int refNum:

"hi re (y < 4) {
refNum = index[y];
reSult += "\nisl"nd = ".
result += islands[refNum];
y = y + 1;
}

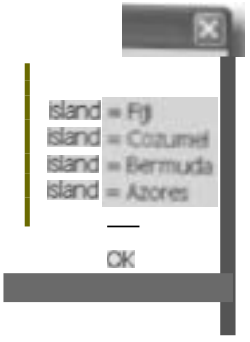
HessaqeBox.ShowCresultl;
}
```

Here's where the index[1 array gets initialized

The islandllll array i• initialized here

The result string is built up u....g the += operator to concatenate lines onto it

This while loop pulls a value from the index(J array and uses it for the index in the islands[] array

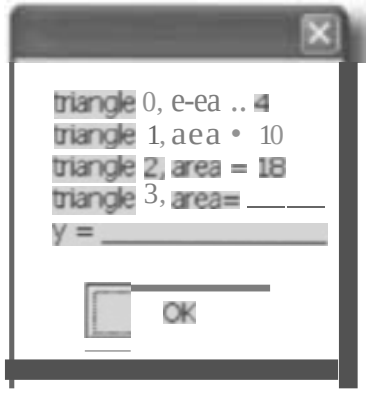


Pool Puzzle



Your **job** is to take code snippets from the pool and place them into the blank lines in the code. You may use each snippet more than once, and you won't need to use all the snippets. Your **goal** is to make a class that will compile and run and produce the output listed.

Output



Bonus Question!

For extra bonus points, use snippets from the pool to fill in the two blanks missing from the output.

Note: Each snippet from the pool can be used more than once

```
class Triangle

    double area,
    int height;
    int length:
    public static void Main(String[] args)

        string results _ "";

        _____________________
        while

            _____________________
                .helght    (x. 1) • 2;
            ____.length _ x • 4;

            _____________________
            results .- "triangle " • x • ", area";
            r..sults                    .area • "\n";

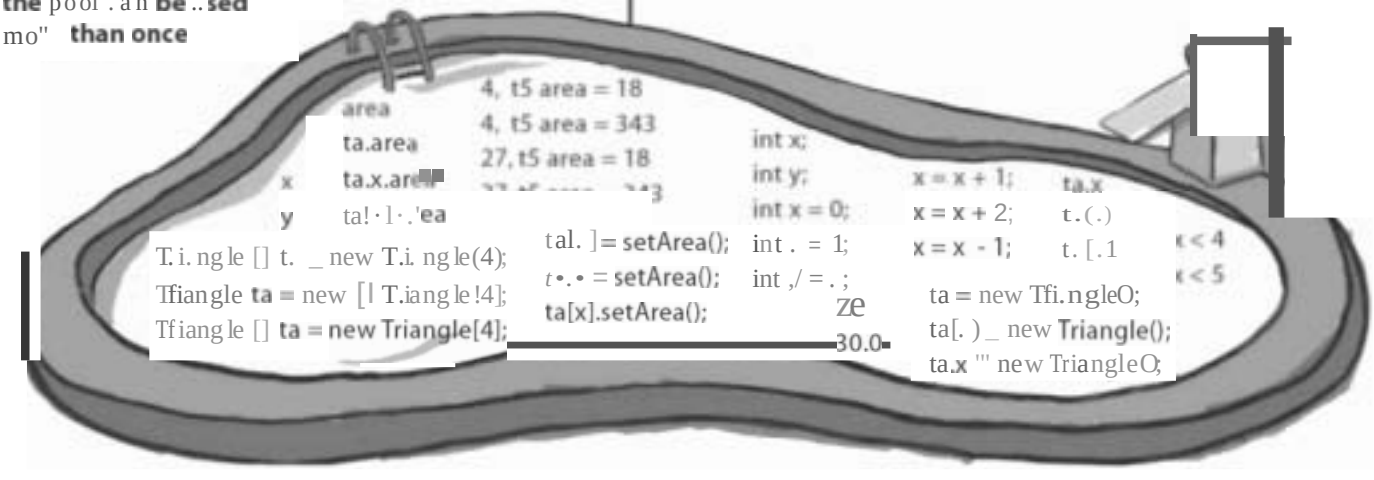
        _____________
        x - 27;
        Triangle r5 = ta[2];
        ta[2].area _ 343;
        results .- "y = " • y;
        MessageBox.Show(results •
            ", tS area = " • t5.area);

    void setArea()  ____________________

        _____________ = (height * length) f 2;
```

Here's the entry point for the application. Assume it's in a file with the right "using" lines at the top

Hint: SetArea() is NOT a static method. Flip back to Chapter 3 for a refresher .... what the static keyword means



Pool puzzle Solution



Notice how this cla.. contains the entry point, but it also creates an instance of itself? That's completely legal in C#

```
class Triangle
{
    double area;
    int height;
    int length;
    public static VQid Main(String[] args)
    {
        string results = "";
        int x = 0;
        Triangle[] ta = new Triangle[4];
        while ( x < 4 )
        {
            ta[x] = new Triangle();
            ta[x].height = (x + 1) * 2;
            ta[x].length = x * 4;
            ta[x].setArea();
            results += "triangle " + x + ", area";
            results += " = " + ta[x].area + "\n";
            x = x + 1;
        }
        int y = x;
        x * 27;
        Triangle t5 . ta[2];
        ta[2].area * 343;
        results += "y = " + y;
        MessageBox.Show(reeu;ts *
            ", t5 area - " + tS.areal;
    }
    void setAr"a()
    {
        area = (height · length) I 2;
    }
```

After this line, we've got an array of four Triangle references—but there aren't any Triangle objects yet!

The while loop creates the four instances of Triangle by calling the new statement four times.

The SetArea() method uses the height and length fields to set the area field. Since it's not a static method, it can only be called from inside an instance of Triangle

Bonus Answer





There are about 77 reserved words in C#, These are words reserved by the C# compiler; you can't use them for variable names. You'll know them all really well by the time you finish the book. Here are some you've already used, Write down what you think these words do in C#.

namespace

Na..espaces ..ake sure h the na..es you are usin.. in Yo r proara.. don't collide with the ones in the NET Fra..ework or other external classes 've used in r ram Il he classes and m ods in a pr a.. are inside a namespace

for

This lets you do a 1 hat ex e s thr m n s First el r s h variable it's na to use, then there's the state..ent that evaluates the variable ainst a condition. The third state...ent does somethi to the value

class

A class ' how de ine ar object. Classes have rties and methods P. es are what th know and ..ethods are what t do

public

method i. declared as ublic it can be used b classes and called b ..ethods that are ou ide e one • ein dec ared '",

else

ode h with else will aet executed if the if men ceding it fails

new

/ou use this to create a new instance an object

using

Th,. " a wa listin all the names aces ou are us in vour am Usin lets you use code from he .NET tra...ework and pre-detined classes from hird parties as we as c asses you can m e yourse

if

One wa of settin u a conditional state..ent in a a.. It sa if one thi is true do one thi and i not do somethin else

while

while loops are loops that keep on going as long as the condition in them is true.



Typecross Solution



Name: Date:

C# Lab

A Day at the Races

This lab gives you a spec that describes a program for you to build, using the knowledge you've gained over the last few chapters.

This project is bigger than the ones you've seen so far. So read the whole thing before you get started, and give yourself a little time. And don't worry if you get stuck—there's nothing new in here, so you can move on in the book and come back to the lab later.

We've filled in a few design details for you, and we've made sure you've got all the pieces you need... and nothing else.

It's up to you to finish the job. You can download an executable for this lab from the website... but we won't give you the code for the answer.

The Spec: Build a Racetrack Simulator

Joe, Bob, and Al love going to the track, but they're tired of losing all their money. They need you to build a simulator for them so they can figure out winners *before* they lay their money down. And, if you do a good job, they'll cut you in on their profits.

Here's what you're going to build for them...

The Guys

Joe, Bob, and Al want to bet on a dog race. Joe starts with 50 bucks, Bob starts with 75 bucks, and Al starts with 45 bucks. Before each race, they'll each decide if they want to bet, and how much they want to put down. The guys can change their bets right up to the start of the race... but once the race starts, all bets are final.



The Betting Parlor

The betting parlor keeps track of how much cash each guy has, and what bet he's placed. There's a minimum bet of 5 bucks. The parlor only takes one bet per person for any one race.

The parlor checks to make sure that the guy who's betting has enough cash to cover his bet—so the guys can't place a bet if they don't have the cash to cover the bet.



Betting

Every bet is double-or-nothing—either the winner doubles his money, or he loses what he bet. There's a minimum bet of 5 bucks, and each guy can bet up to 15 bucks on a single dog. If the dog wins, the bettor ends up with twice the amount that he bets (after the race is complete). If he loses, that amount disappears from his pile.

Say a guy places a $10 bet at the window. At the end of the race, if his dog wins, his cash goes up by $10 (because he keeps the original $10 he bet, plus he gets $10 more from winning). If he loses, his cash goes down by $10.

All bets: double-or-nothing
Minimum Bet: $5
Up to $15 per dog
Win: $$ added
Lose: $$ removed

The Race

There are four dogs that run on a straight track. The winner of the race is the first dog to cross the finish line. The race is totally random, there are no handicaps or odds, and a dog isn't more likely to win his next race based on his past performance.

If you want to build a handicap system, by all means do it! It'll be really good practice writing some fun code.



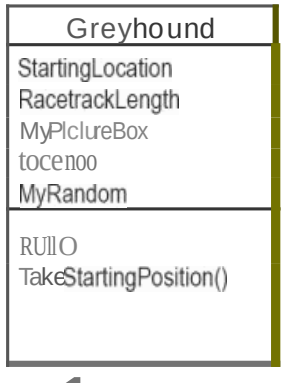
Sound fun? We've got more details coming up...

You'lll1eed three classes alld a forl\l

Y',,"ll huild three main classes in the project, as well as a (;l "1for the simulator. You should have an array or three Guy objects In keep track of the three guys and their winnings, and an array of four Greyhound objects tlt"l actually n llllh,' race. .\1,,,, each instance or GUy should have its own Bet object that keeps tra('k of his b..1and pays mil (or takes ba"k;, cash at the end of the race.

" ',.'ve gotten vou started with class descript jon, and some ,nippets „1' code to work from. You've got to fini,h everything lip_

yo...:ll **need** to add "u sing System.Windows.Fonns" to **the** top of the Greyhound and Guy classes.




```
public class Greyhound
  public int StartingPosition; // Where my PictureBox starts
  public int RacetracJtLength; // How long the racetrack is
  pUblic PictureBox MyPictureBox • null: // My PictureBox object
  public int Location - 0; // My Location on the racetrack
  public Random MyRandom; // An instance of Random

  pUblic bool Run() {
    1/ Move forward either 1, 2, 3 or 4 spaces at random
    1/ Update the position of my PictureBox on the form
    // Return true if I won the race

public void TakeStartingPositiono {
  // Reset my location to the start line
```

We've added comments to give r- an idea of what to do.

Don't overthink this. sometimes you just need to set a variable, and you're done

The Greyhound object initializer is pretty straightforward Just make sure you pass a reference to the right PictureBox on the form to each Greyhound object

Your object ca" **control** !hi"g, 0" your for,",..

The Greyhound class keeps lrark of its position "ll the racetrack during the race. It also updates the location of Ih,' PictureBox representing the dog I" move down rh.. race' track. E,,,:h instance Of Greyhound uses a field called MyPictureBox to reference the PictureBox control on the form that shows the picture " f the dog. Suppose rhe distance variable contains the distance 1<, move Ihe dog forward. TIt"ll this ,'od,' will update the location of MyPictureBox by adding distance to iI, X value

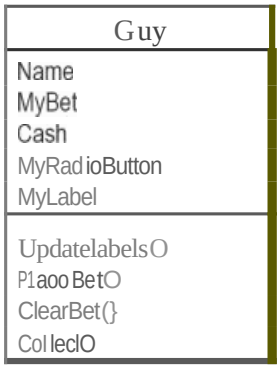
you'll have to make sure the form passes U, right picturebox into each Greyhound's object initializer.

```
Point p = MyPictureBox.Location;
p.X +_ distancc,
MyPictureBox.Location . p;
```

You get the t:urrent ,ocation of the picture...

...add the value to move forward to its X coordinate ..

. and then update the picture box location on the form




```
public class Guy I
   public string Name; 1/ The guy's name
   public Bet MyBet; // An instance of Bet() that has his bet
   public int. Cash; 1/ How much cash he has

   // The last two fields are the guy's GUI controls on the form
   pUblic RadloButton MyRadioButton; // My RadioButton
   public Label MyLabel; // My Label

   public void UpdateLabels()
     // Set my label to my bet's description,      the label on
     // my radio button to show my cash ("Joe      43 bucks")
                          Add you.. code here

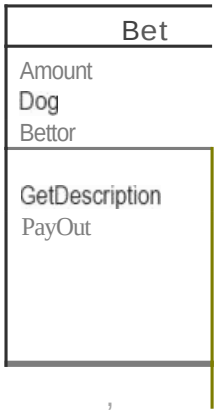
   public void ClearBet(J { ) // Reset my bet so it's zero

   public bool PlaceBer(int Amount, lnt Dog) {
     // Place a new bet and store it in my bet field
     // Return true if the guy had enough money to bet !

   public void Collect(int Winner) { } // Ask my bet to payout
```

Re..e...ber that bets are represented by instances of Bet

The key here is to use the Bet object... let it do the work




```
public class Bet {
   public int Amount; // The amount of cash that was bet
   public int Dog; /1 The number of the dog the bet is on
   public Guy Bettor; // The guy who placed the bet

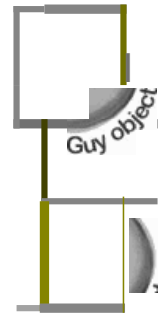
   public string GetDescription() {
     /1 Return a string that says who placed the bet, how much
     // cash lias bet, and IIhich dog he bet on ("Joe bets 8 on
     // dog #4"). If the amount is zero, no bet was placed
     // ("Joe hasn't placed a bet").

   public int Payoutlint Winner) {
     // The parameter is the winner of the race. If the dog won,
     // return the amount bet. Otherwise, return the negative of
     // the amount bet.
```

This " a common programming task: assembling a stri.... or message from several individual bits of data

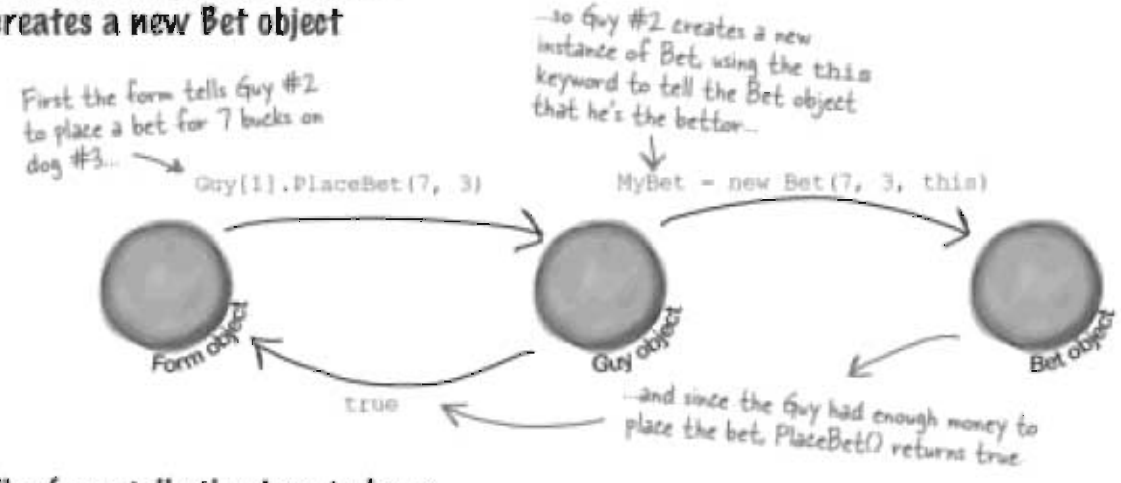
Here's your applicatiol1 architecture

Spo'nd some rinu- looking closely at the architecture. It],"ɔks pretty complicated at first, but there's nothing here you d"U'I know. Your .i"]' i, to recreate this architecture yourself, starting with the Gr eyhound and Guy arrays ill your Illa;" I;,r",.

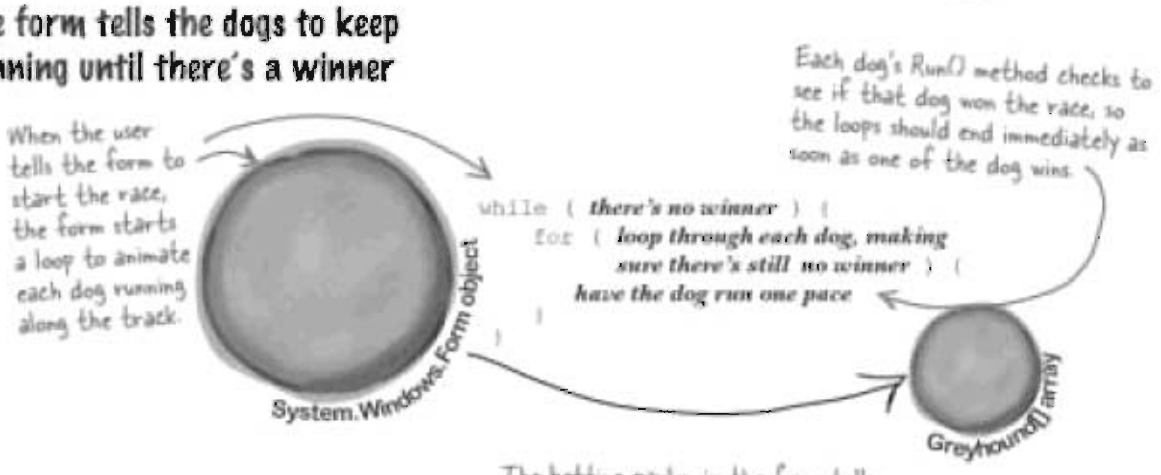




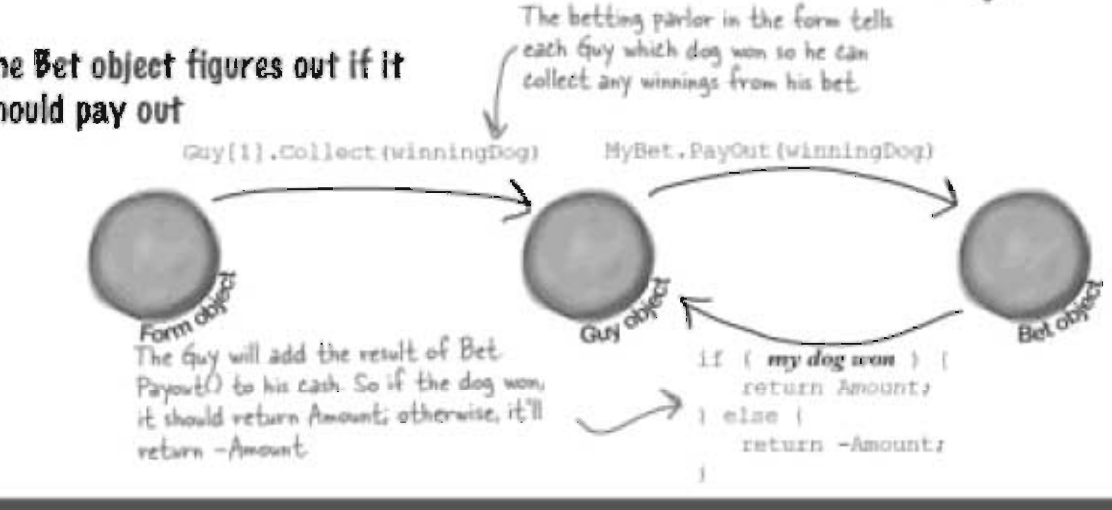
When a Guy places a bet, he creates a new Bet object



The form tells the dogs to keep running until there's a winner



The Bet object figures out if it should pay out

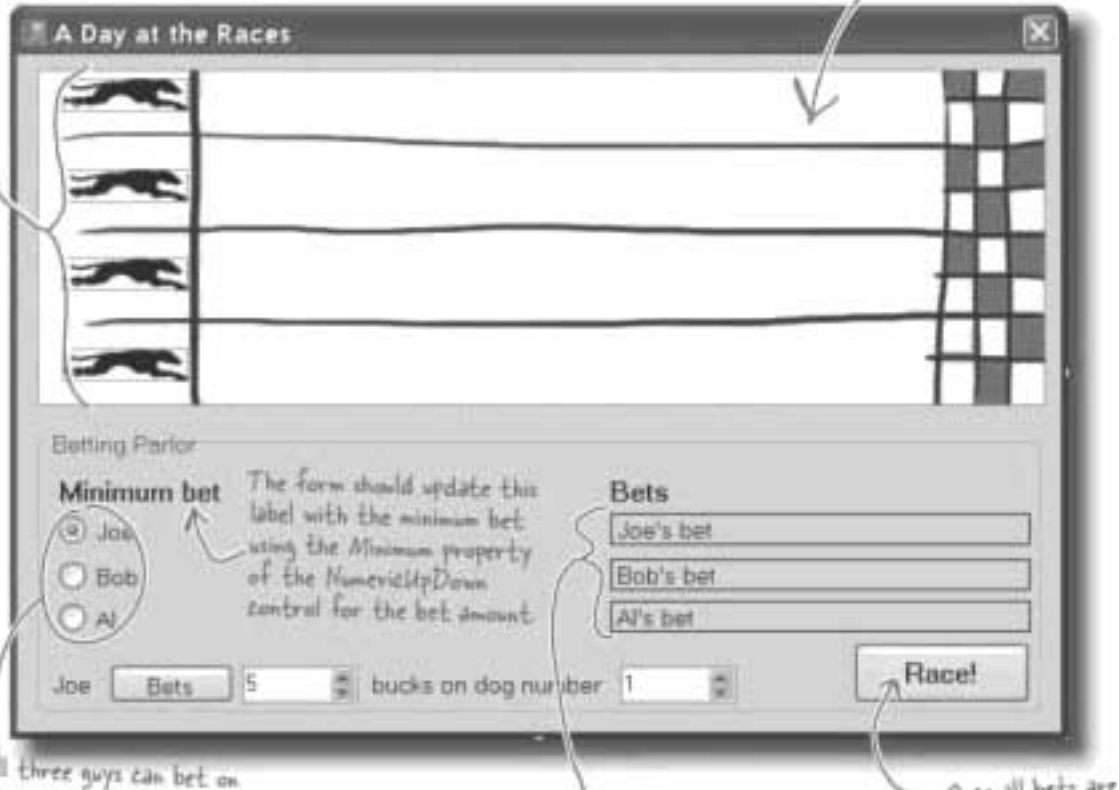


Here's what your GUI should look like

The graphical user interface for the "Day at the Races" application consists of a form that's divided into two sections. The top is the racetrack: a `PictureBox` control for the track, and four more for the dogs. The bottom half of the form shows the betting parlor, where three guys (Joe, Bob, and Al) can bet on the outcome of the race.

Each of the four dogs has its own PictureBox control. When you initialize each of the four Greyhound objects, each one's MyPictureBox field will have a reference to one of these objects. You'll pass the reference (along with the racetrack length and starting position) to the Greyhound's object initializer

You'll use the Length property of the racetrack PictureBox control to set the racetrack length in the Greyhound object, which it'll use to figure out if it won the race



All three guys can bet on the race, but there's only one betting window so only one guy can place a bet at a time. These radio buttons are used to select which guy places the bet

When a guy places a bet, it overwrites any previous bet he placed. The current bets show up in these label controls. Each label has AutoSize set to False and BorderStyle set to FixedSingle

Once all bets are placed, click this button to start the race

You can download the graphics files from www.headfirstlabs.com/books/hfcsharp/

Placing bets

Use the controls in the Betting Parlor groupbox to place each guy's bet. There are three distinct stages here:

When a guy places a bet, his Guy object updates this label using the MyLabel reference. He also updates the cash he has using his MyRadioButton reference.

1 **No bets have been placed yet**

When the program first starts up, or if a race has just finished, no bets have been placed in the betting parlor. You'll see each guy's total cash next to his name on the left.



Each guy's cash shows up here.

The minimum bet should be the same as the minimum value in the bet control.

Once Bob places his bet, his Guy object updates this label and the radio button text.

2 **Each guy places his bets**

To place a bet, select the guy's radio button, select an amount and a dog, and click the Bets button. His `PlaceBet()` method will update the label and radio button.



Since Al bet 12 bucks on the winning dog, his cash goes up by 12. The other two guys lose the money they bet.

3 **After the race, each guy collects his winnings (or pays up!)**

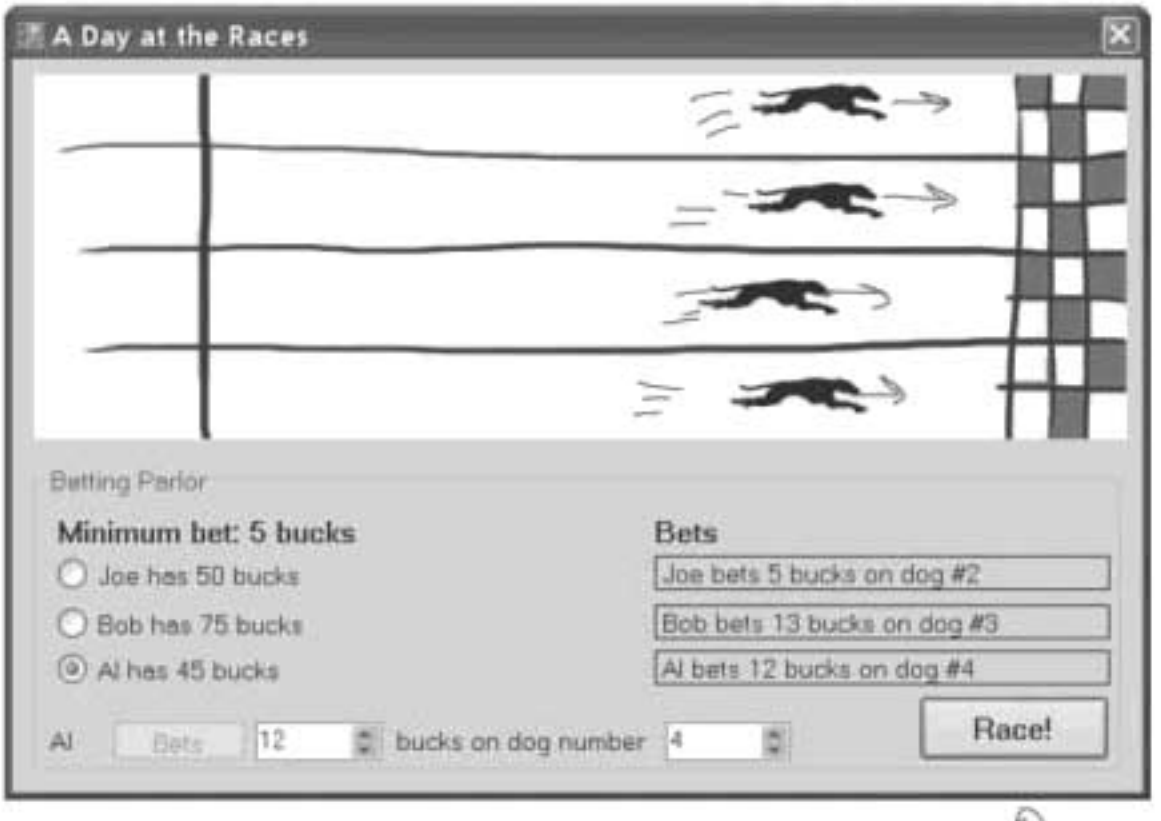
Once the race is complete and there's a winner, each `Guy` object calls his `Collect()` method and adds his winnings or losses to his cash.



The Finished Product

You'll know your "Day at the Races" application is done when your guys can place their bets and watch the dogs race.

During the race, the four dog images run across the racetrack until one of them wins the race.



During the race, no bets can be placed... and make sure you can't start a new race while the dogs are running!

You can download a finished executable, as well as the graphics files for the four dogs and the racetrack, from the Head First labs website: www.headfirstlabs.com/books/hfcsharp

But you won't find the source code! In real life, you don't get a solution to your programming problems. Here's your chance to really test your C# knowledge and see just how much you've learned!

5 encapsulation

Keep your privates... private

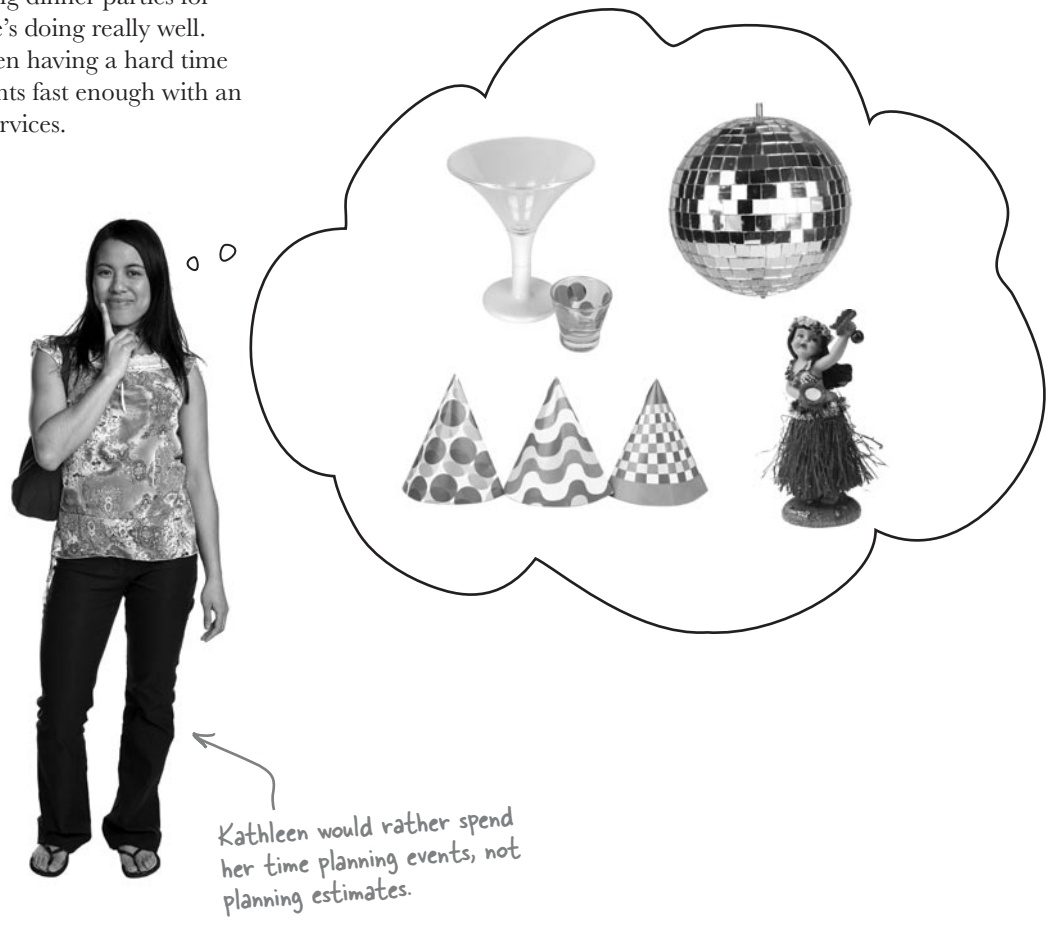


Ever wished for a little more privacy?

Sometimes your objects feel the same way. Just like you don't want anybody you don't trust reading your journal or paging through your bank statements, good objects don't let ***other*** objects go poking around their fields. In this chapter, you're going to learn about the power of **encapsulation**. You'll **make your object's data private**, and add methods to **protect how that data is accessed**.

Kathleen is an event planner

She's been planning dinner parties for her clients and she's doing really well. But lately she's been having a hard time responding to clients fast enough with an estimate for her services.



When a new client calls Kathleen to do a party, she needs to find out the number of guests, what kind of drinks to serve, and what decorations she should buy. Then she uses a pretty complicated calculation to figure out the total cost, based on a flow chart she's been using for years. The bad news is that it takes her a long time to work through her chart, and while she's estimating, her potential clients are checking out other event planners.

It's up to you to build her a C#-driven event estimator and save her business. Imagine the party she'll throw you when you succeed!

What does the estimator do?

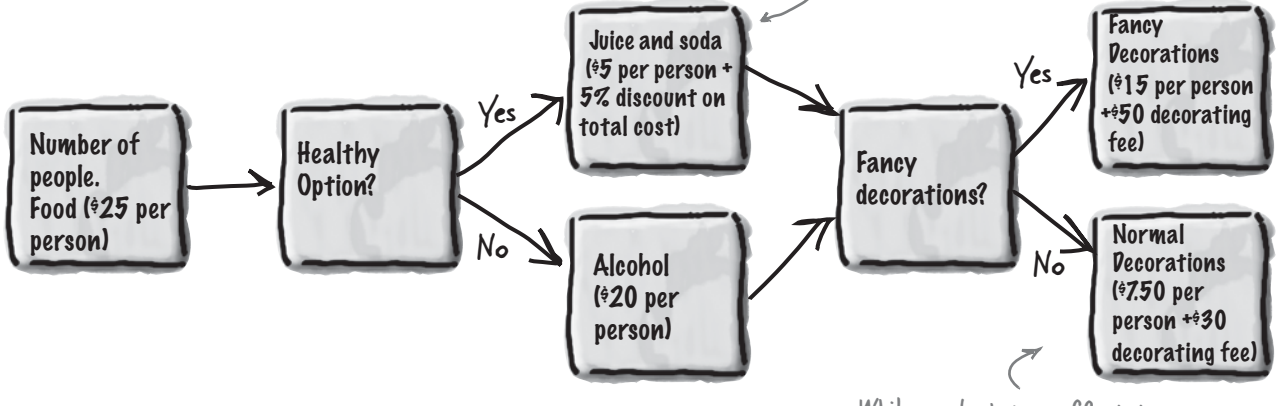
Kathleen runs down some of the basics of her system for figuring out the costs of an event. Here's part of what she came up with:

Kathleen's Party Planning Program—Cost Estimate for a Dinner Party

- For each person on the guest list there's a $25 food charge.
- Clients have a choice when it comes to drinks. Most parties serve alcohol, which costs $20 per person. But they can also choose to have a party without alcohol. Kathleen calls that the "Healthy Option," and it only costs $5 per person to have soda and juice instead of alcohol. Choosing the Healthy Option is a lot easier for her, so she gives the client a 5% discount on the entire party, too.
- There are two options for the cost of decorations. If a client goes with the normal decorations, it's $7.50 per person with a $30 decorating fee. A client can also upgrade the party decorations to the "Fancy Option"—that costs $15 per person with a $50 one-time decorating fee.

Here's another look at this same set of costs, broken down into a little flow chart to help you see how it works:

Some of these choices involve a change to the final price of the event, as well as individual per-person costs.

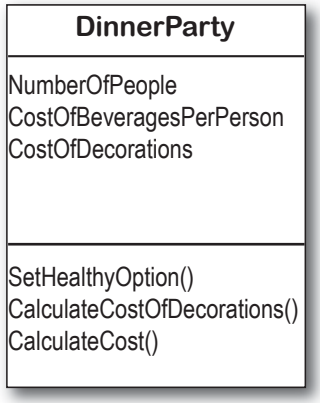


While most choices affect the cost for each guest, there are also one-time fees to figure in.



Exercise

Build a program to solve Kathleen's party estimating problem.



Here's the class diagram for the DinnerParty class you'll need to create.

1. Create a new Windows Application project and add a class file to it called `DinnerParty.cs`, and build the DinnerParty class using the class diagram to the left. It's got three methods: `CalculateCostOfDecorations()`, `SetHealthyOption()`, and `CalculateCost()`. For the fields, use `decimal` for the two costs, `int` for the number of people, and `bool` to keep track of whether or not the healthy option was selected. Make sure you **add an M after every literal** you assign to a decimal value (`10.0M`).

2. Here's a useful C# tool. Since the cost of food won't be changed by the program, you can declare it as a ***constant***, which is like a variable except that its value can never be changed. Here's the declaration to use:

```
public const int CostOfFoodPerPerson = 25;
```

3. Flip back to the previous page to be sure you've got all of the logic right for the methods. Only one of them returns a value (a `decimal`)—the other two are `void`. The `CalculateCostOfDecorations()` method figures out the cost of decorations for the number of people attending the party. Use the `CalculateCost()` method to figure out the total cost by adding up the cost of the decorations to the cost of drinks and food per person. If the client wants the Healthy Option, you can apply the discount inside the `CalculateCost()` method after you've figured out the total cost.

4. Add this code to your form:

You'll declare the dinnerParty field in the form, and then add these four lines below InitializeComponent().

```
DinnerParty dinnerParty;
public Form1() {
  InitializeComponent();
  dinnerParty = new DinnerParty() { NumberOfPeople = 5 };
  dinnerParty.SetHealthyOption(false);
  dinnerParty.CalculateCostOfDecorations(true);
  DisplayDinnerPartyCost();
}
```

The SetHealthyOption() method uses a bool parameter (healthyOption) to update the CostOfBeveragesPerPerson field based on whether or not the client wants the healthy option.

5. Here's what the form should look like. Use the NumericUpDown control's properties to set the maximum number of people to 20, the minimum to 1, and the default to 5. Get rid of the maximize and minimize buttons, too.

Set the default value to 5. The minimum should be 1 and the maximum should be 20.

The Fancy decorations box should have Checked set to true.



This is just a label with the Text Property set to "", the BorderStyle property set to Fixed3D, and the AutoSize property set to false.

You don't need to add "using System.Windows.Forms;" to your DinnerParty class, because it doesn't use MessageBox.Show(), Point, or anything else from that .NET Framework namespace.

6 Instead of using a button to calculate the costs, this form will update the cost label automatically as soon as you use a checkbox or the NumericUpDown control. The first thing you need to do is create a method in the form that displays the cost.

This method will get called by all of the other methods you create on the form. It's how you update the cost label with the right value whenever anything changes.

Add this method to `Form1()`. It'll get called when the NumericUpDown control is clicked:

Add this method to the form—it'll recalculate the cost of the party and put it in the Cost label.

```
private void DisplayDinnerPartyCost()
{
  decimal Cost = dinnerParty.CalculateCost(checkBox2.Checked);
  costLabel.Text = Cost.ToString("c");
}
```

This is true if the checkbox for the Healthy Option is checked.

Change the name of the label that displays the cost to costLabel.

Passing "c" to ToString() tells it to format the cost as a currency value. If you're in a country that uses dollars, it'll add a dollar sign.

When you double-click on a button in the IDE to add code that gets run when the button is clicked, that's an event handler too.

7 Now hook up the NumericUpDown field to the NumberOfPeople variable you created in the DinnerParty class and display the cost in the form. Double-click on the NumericUpDown control—the IDE will add an ***event handler*** to your code. That's a method that gets run every time the control is changed. It'll reset the number of people in the party. Fill it in like this:

```
private void numericUpDown1_ValueChanged(
                                  object sender, EventArgs e)
{
   dinnerParty.NumberOfPeople = (int) numericUpDown1.Value;
   DisplayDinnerPartyCost();
}
```

You need to cast numericUpDown.Value to an int because it's a Decimal property.

Uh-oh—there's a problem with this code. Can you spot it? Don't worry if you don't see it just yet. We'll dig into it in just a couple of minutes!

The value you send from the form to the method will be fancyBox.Checked. That will be passed as a boolean parameter to the method in the class.

These are just two-line methods. The first line will call the method you created in the class to figure out the costs and the second will display the total cost on the form.

8 Double-click on the Fancy Decorations checkbox on the form and make sure that it first calls `CalculateCostOfDecorations()`, and then `DisplayDinnerPartyCost()`. Next, double-click the Healthy Option checkbox and make sure that it calls the `SetHealthyOption()` method in the DinnerParty class and then calls the `DisplayDinnerPartyCost()` method.



Here's the code that goes into DinnerParty.cs.

Using a constant for CostOfFoodPerPerson ensures the value can't be changed. It also makes the code easier to read—it's clear that this value never changes.

When the form first creates the object, it uses the initializer to set NumberOfPeople. Then it calls SetHealthyOption() and CalculateCostOfDecorations() to set the other fields.

```
public class DinnerParty {
    const int CostOfFoodPerPerson = 25;
    public int NumberOfPeople;
    public decimal CostOfBeveragesPerPerson;
    public decimal CostOfDecorations = 0;

    public void SetHealthyOption(bool healthyOption) {
        if (healthyOption) {
            CostOfBeveragesPerPerson = 5.00M;
        } else {
            CostOfBeveragesPerPerson = 20.00M;
        }
    }

    public void CalculateCostOfDecorations(bool fancy) {
        if (fancy)
        {
            CostOfDecorations = (NumberOfPeople * 15.00M) + 50M;
        } else {
            CostOfDecorations = (NumberOfPeople * 7.50M) + 30M;
        }
    }
    public decimal CalculateCost(bool healthyOption) {
        decimal totalCost = CostOfDecorations +
               ((CostOfBeveragesPerPerson + CostOfFoodPerPerson)
                   * NumberOfPeople);

        if (healthyOption) {
            return totalCost * .95M;
        } else {
            return totalCost;
        }
    }
}
```

We used "if (Fancy)" instead of typing "if (Fancy == true)" because the if statement always checks if the condition is true.

We used parentheses to make sure the math works out properly.

This applies the 5% discount to the overall event cost if the non-alcoholic option was chosen.

We had you use a `decimal` for the prices because it's designed for monetary values. Just make sure you always put an "M" after every literal—so if you want to store $35.26, make sure you write `35.26M`.

```
public partial class Form1 : Form {
    DinnerParty dinnerParty;
    public Form1() {
        InitializeComponent();
        dinnerParty = new DinnerParty() { NumberOfPeople = 5 };
        dinnerParty.CalculateCostOfDecorations(fancyBox.Checked);
        dinnerParty.SetHealthyOption(healthyBox.Checked);
        DisplayDinnerPartyCost();
    }

    private void fancyBox_CheckedChanged(object sender, EventArgs e) {
        dinnerParty.CalculateCostOfDecorations(fancyBox.Checked);
        DisplayDinnerPartyCost();
    }

    private void healthyBox_CheckedChanged(object sender, EventArgs e) {
        dinnerParty.SetHealthyOption(healthyBox.Checked);
        DisplayDinnerPartyCost();
    }

    private void numericUpDown1_ValueChanged(object sender, EventArgs e) {
        dinnerParty.NumberOfPeople = (int)numericUpDown1.Value;
        DisplayDinnerPartyCost();
    }

    private void DisplayDinnerPartyCost() {
        decimal Cost = dinnerParty.CalculateCost(healthyBox.Checked);
        costLabel.Text = Cost.ToString("c");
    }
}
```

We call DisplayDinnerPartyCost to initialize the label that shows the cost as soon as the form's loaded.

Changes to the checkboxes on the form set the healthyOption and Fancy booleans to true or false in the SetHealthyOption() and CalculateCostOfDecorations() methods.

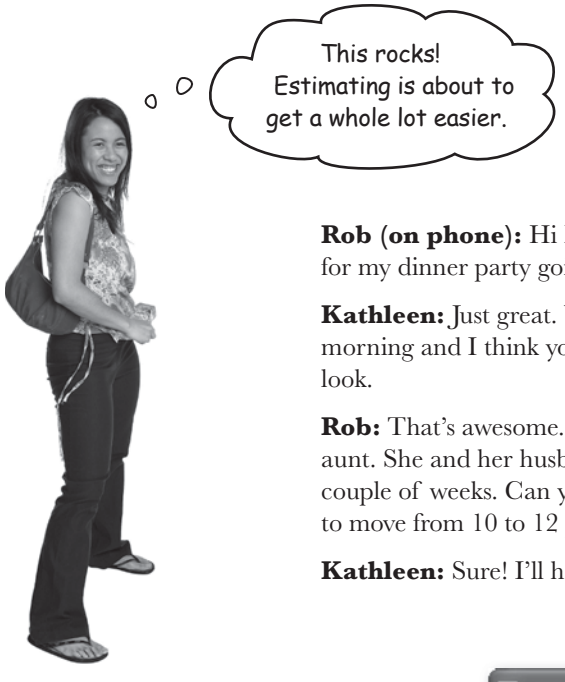
We named our checkboxes "healthyBox" and "fancyBox" so you could see what's going on in their event handler methods.

The new dinner party cost needs to be recalculated and displayed any time the number changes or the checkboxes are checked.

String formatting

You've already seen how you can convert any variable to a string using its ToString() method. If you pass "c" to ToString(), it converts it to the local currency. You can also pass it "f3" to format it with as a decimal number with three decimal places, "0" (that's a zero) to convert to a whole number, "0%" for a whole number percentage, and "n" to display it as a number with a comma separator for thousands. Take a minute and see how each of these looks in your program!

Kathleen's Test Drive



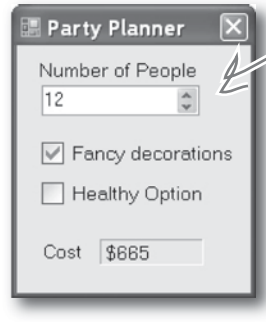
Rob's one of Kathleen's favorite clients. She did his wedding last year, and now she's planning an important dinner party for him.

Rob (on phone): Hi Kathleen. How are the arrangements for my dinner party going?

Kathleen: Just great. We were out looking at decorations this morning and I think you'll love the way the party's going to look.

Rob: That's awesome. Listen, we just got a call from my wife's aunt. She and her husband are going to be visiting for the next couple of weeks. Can you tell me what it does to the estimate to move from 10 to 12 people on the guest list?

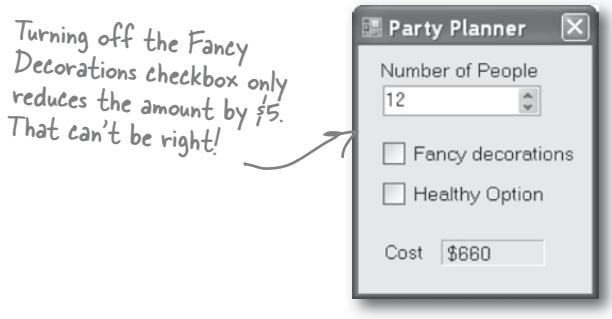
Kathleen: Sure! I'll have that for you in just one minute.



Changing the Number of People value from 10 to 12 and hitting enter shows $665 as the total cost. Hmm, that seems a little low...

Kathleen: OK. It looks like the total cost for the dinner will go from $575 to $665.

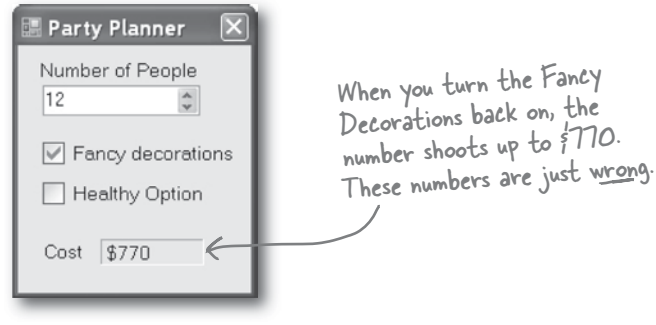
Rob: Only $90 difference? That sounds like a great deal! What if we decide to cut the fancy decorations? What's the cost then?



Kathleen: Um, it looks like... um, $660.

Rob: $660? I thought the decorations were $15 per person. Did you change your pricing or something? If it's only $5 difference, we might as well go with the Fancy Decorations. I've gotta tell you though, this pricing is confusing.

Kathleen: We just had this new program written to do the estimation for us. But it looks like there might be a problem. Just one second while I add the fancy decorations back to the bill.



Kathleen: Rob, I think there's been a mistake. It looks like the cost with the fancy decorations just shot up to $770. That does seem to make more sense. But I am beginning not to trust this application. I'm going to send it back for some bug fixes and work up your estimate by hand. Can I get back to you tomorrow?

Rob: I am not paying $770 just to add two people to the party. The price you quoted me before was a lot more reasonable. I'll pay you the $665 you quoted me in the first place, but I just can't go higher than that!



Why do you think the numbers are coming out wrong every time Kathleen makes a change?

Each option should be calculated individually

Even though we made sure to calculate all of the amounts according to what Kathleen said, we didn't think about what would happen when people made changes to just one of the options on the form.

When you launch the program, the form sets the number of people to 5 and Fancy Decorations to true. It leaves Healthy Option unchecked and it calculates the cost of the dinner party as $350. Here's how it comes up with the initial total cost:



Don't worry! This one wasn't your fault.

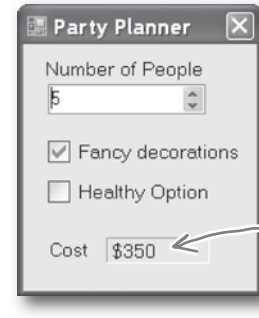
We built a nasty little bug into the code we gave you to show you just how easy it is to have problems with how objects use each others' fields... and just how hard those problems are to spot.

5 people.

$20 per person for drinks → Total cost of drinks = $100

$25 per person for food → Total cost of food = $125

$15 per person for decorations plus $50 fee. → Total cost of Decorations = $125



So far, so good.

$100 + $125 + 125 = $350

When you change the number of guests, the application should recalculate the total estimate the same way. But it doesn't:

10 people.

$20 per person for drinks → Total cost of drinks = $200

$25 per person for food → Total cost of food = $250

$15 per person for decorations plus $50 fee. → Total cost of Decorations = $200

$200 + $250+ 200 = $650



This is the total we should get. But we're not...

The program is adding the old cost of decorations up with the new cost of food and drink.

It's doing $200 + $250 + $125= $575.

New food and drink cost.

Old decorations.

The Problem Up Close



Take a look at the method that handles changes to the value in the `numericUpDown` control. It sets the value from the field to the `NumberofPeople` variable and then calls the `DisplayDinnerPartyCost()` method. Then it counts on that method to handle recalculating all the individual new costs.

This line sets the value of NumberofPeople in this instance of DinnerParty to the value in the form.

```
private void numericUpDown1_ValueChanged(
                            object sender, EventArgs e) {
    dinnerParty.NumberOfPeople = (int)numericUpDown1.Value;
    DisplayDinnerPartyCost();
}
```

This method calls the CalculateCost() method, but not the CalculateCostofDecorations() method.

So, when you make a change to the value in the NumberofPeople field, this method never gets called:

```
public void CalculateCostOfDecorations(bool Fancy) {
    if (Fancy) {
        CostOfDecorations = (NumberOfPeople * 15.00M) + 50M;
    } else {
        CostOfDecorations = (NumberOfPeople * 7.50M) + 30M;
    }
}
```

This variable is set to $125 from when the form first called it and, since this method doesn't get called again, it doesn't change.

That's why the number corrects itself when you turn fancy decorations back on. Clicking the checkbox makes the program run CalculateCostOf Decorations() again.



Hold on! I assumed Kathleen would always set all three options at once!

People won't always use your programs in exactly the way you expect.

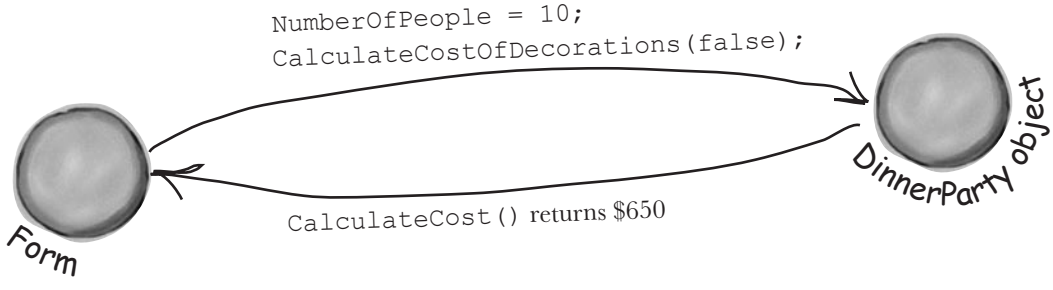
Luckily, C# gives you a powerful tool to make sure your program always works correctly—even when people do things you never thought of. It's called **encapsulation** and it's a really helpful technique for working with objects.

It's easy to accidentally misuse your objects

Kathleen ran into problems because her form ignored the convenient `CalculateCostOfDecorations()` method that you set up and instead went directly to the fields in the DinnerParty class. So even though your DinnerParty class worked just fine, the form called it in an unexpected way... and that caused problems.

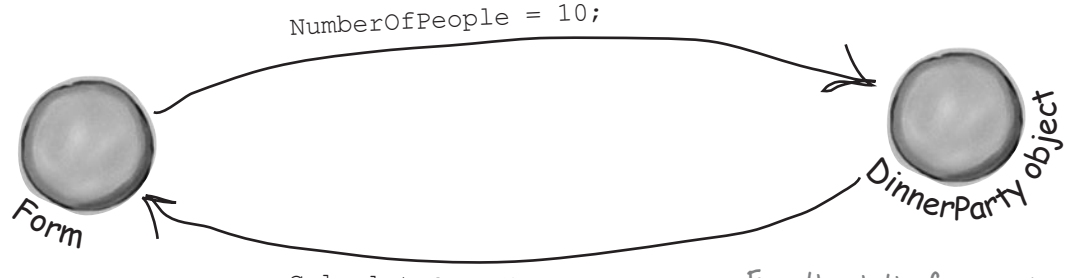
1 **How the DinnerParty class expected to be called**

The DinnerParty class gave the form a perfectly good method to calculate the total cost of decorations. All it had to do was set the number of people and then call `CalculateCostOfDecorations()`, and then `CalculateCost()` will return the correct cost.



2 **How the DinnerParty class was actually called**

The form set the number of people, but just called the `CalculateCost()` method without first recalculating the cost of the decorations. That threw off the whole calculation, and Kathleen ended up giving Rob the wrong price.



Even though the form didn't set up the party properly, `CalculateCost()` still returned a number... and there was no way for Kathleen to know that the number was wrong.

Encapsulation means keeping some of the data in a class private

There's an easy way to avoid this kind of problem: make sure that there's only one way to use your class. Luckily, C# makes it easy to do that by letting you declare some of your fields as **`private`**. So far, you've only seen public fields. If you've got an object with a public field, any other object can read or change that field. But if you make it a private field, then **that field can only be accessed from inside that object** (or by another object *of the same class*).

Use your laziness to your own benefit—if you leave off the "private" or "public", then C# will just assume that your field is private.

```
public class DinnerParty {
    private int numberOfPeople;

    ...

    public void SetPartyOptions(int people, bool fancy) {
        numberOfPeople = people;
        CalculateCostOfDecorations(fancy);
    }

    public void GetNumberOfPeople() {
        return numberOfPeople;
    }
```

If you want to make a field private, all you need to do is use the `private` keyword when you declare it. That tells C# that if you've got an instance of DinnerParty, its `numberOfPeople` field can only be read and written by that instance. Other objects won't even know it's there.

Other objects still need a way to set the number of people for the dinner party. One good way to give them access to it is to add methods to set or get the number of people. That way you can make sure that the `CalculateCostOfDecorations()` method gets run every time the number of people is changed. That'll take care of that pesky bug.

By making the field that holds the number of party guests *private*, we only give the form one way to tell the DinnerParty class how many people are at the party—and we can make sure the cost of decorations is recalculated properly. When you make some data private and then write code to use that data, it's called *encapsulation*.

en-cap-su-la-ted, adj.
enclosed by a protective coating or membrane. *The divers were fully* ***encapsulated*** *by their submersible, and could only enter and exit through the airlock.*

Use encapsulation to control access to your class's methods and fields

When you make all of your fields and methods public, any other class can access them. Everything your class does and knows about becomes an open book for every other class in your program... and you just saw how that can cause your program to behave in ways you never expected. Encapsulation lets you control what you share and what you keep private inside your class. Let's see how this works:

SecretAgent

Alias
RealName
Password

AgentGreeting()

1. Super-spy Herb Jones is defending life, liberty, and the pursuit of happiness as an undercover agent in the USSR. His ciaAgent object is an instance of the SecretAgent class.




```
RealName: "Herb Jones"
Alias: "Dash Martin"
Password: "the crow flies at midnight"
```

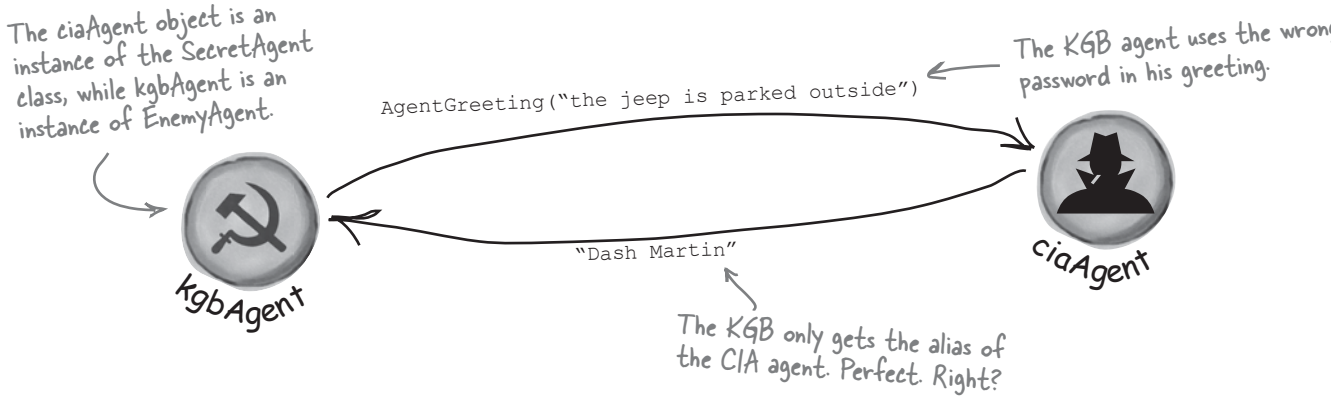
EnemyAgent

Borscht
Vodka

ContactComarades()
OverthrowCapitalists()

2. Agent Jones has a plan to help him evade the enemy KGB agents. He added an `AgentGreeting()` method that takes a password as its parameter. If he doesn't get the right password, he'll only reveal his alias, Dash Martin.

3. Seems like a foolproof way to protect the agent's identity, right? As long as the agent object that calls it doesn't have a the right password, the agent's name is safe.

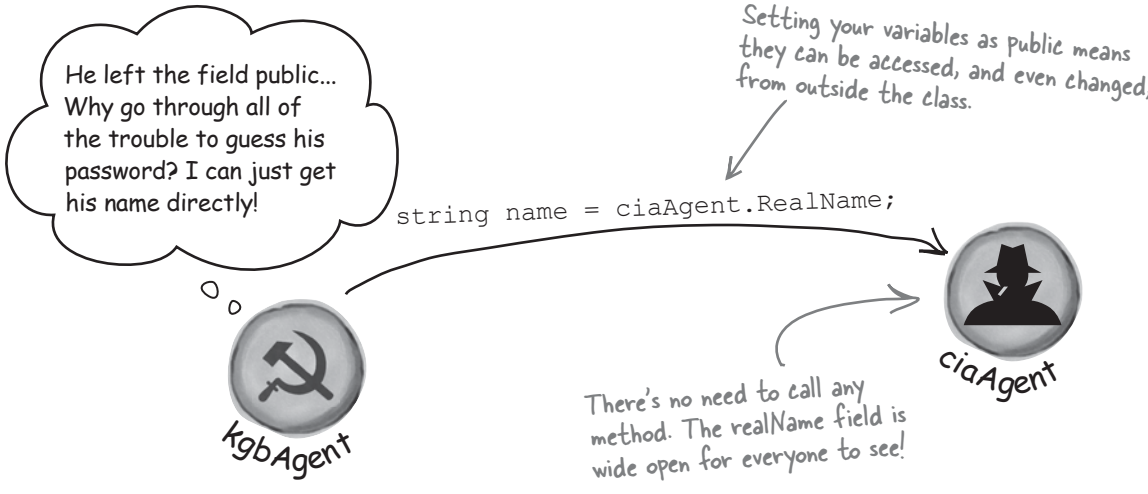


But is the realName field REALLY protected?

So as long as the KGB doesn't know any CIA agent passwords, the CIA's real names are safe. Right? But what about the field declaration for the realName field:

Setting your variables public means they can be accessed, and even changed, from outside the class.

```
public string RealName;
```



Agent Jones can use **`private`** fields to keep the his identity secret from enemy spy objects. Once he declares the `realName` field as private, the only way to get to it is ***by calling methods that have access to the private parts of the class***. So the KGB agent is foiled!

The kgbAgent object can't access the ciaAgent's private fields because they're instances of **different** classes.

Just replace public with private, and boom, your fields are now hidden from the world.

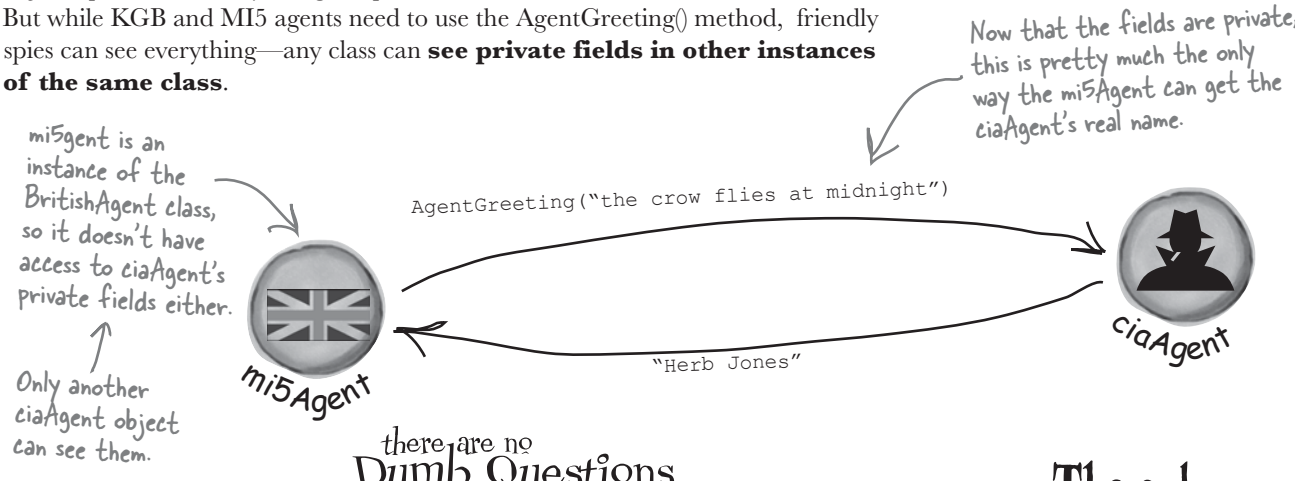
```
private string realName;
```

Keeping your fields and methods private makes sure no outside code is going to make changes to the values you're using when you don't expect it.

You'd also want to make sure that the field that stores the password is private, otherwise the enemy agent can get to it.

Private fields and methods can only be accessed from inside the class

There's only one way that an object can get at the data stored inside another object's private fields: by using the public fields and methods that return the data. But while KGB and MI5 agents need to use the AgentGreeting() method, friendly spies can see everything—any class can **see private fields in other instances of the same class**.



there are no Dumb Questions

Q: Okay, so I need to access private data through public methods. But what happens if the class with the private field doesn't give me a way to get at that data, but my object needs to use it?

A: Then you can't access the data from outside the object. When you're writing a class, you should always make sure that you give other objects some way to get at the data they need. Private fields are a very important part of encapsulation, but they're only part of the story. Writing a class with good encapsulation means giving a sensible, easy-to-use way for other objects to get the data they need, without giving them access to hijack data your class needs.

Q: Why would I ever want to keep a field with no way for another class to access?

A: Sometimes a class needs to keep track of information that is necessary for it to operate, but which no other object really needs to see. Here's an example. When computers generate random numbers, they use special values called *seeds*. You don't need to know how they work, but every instance of `Random` actually contains an array of several dozen numbers that it uses to make sure that `Next()` always gives you a random number. If you create an instance of `Random`, you won't be able to see that array. That's because you don't need it—but if you had access to it, you might be able to put values in it that would cause it to give non-random values. So the seeds have been completely encapsulated from you.

Q: Hey, I just noticed that all of the event handlers I've been using have the `private` keyword. Why are they private?

A: Because C# forms are set up so that only the controls on the forms can trigger event handlers. When you put the `private` keyword in front of any method, then that method can only be used from inside your class. When the IDE adds an event handler method to your program, it declares it as private so other forms or objects can't get to it. But there's no rule that says that an event handler must be private. In fact, you can check this out for yourself—double-click on a button, then change its event handler declaration to `public`. The code will still compile and run.

The only way that one object can get to data stored in a private field inside another object is by using public methods that return the data.



Here's a class with some private fields. Circle the statements below that **won't compile** if they're run from outside the class using **an instance of the object called mySuperChef**.

```
public class SuperChef
{
      public string cookieRecipe;
      private string secretIngredient;
      private const int loyalCustomerOrderAmount = 60;
      public int Temperature;
      private string ingredientSupplier;

      public string GetRecipe (int orderAmount)
      {
            if (orderAmount >= loyalCustomerOrderAmount)
            {
                  return cookieRecipe + " " + secretIngredient;
            }
            else
            {
                  return cookieRecipe;
            }
      }
}
```

1. `string ovenTemp = mySuperChef.Temperature;`

2. `string supplier = mySuperChef.ingredientSupplier;`

3. `int loyalCustomerOrderAmount = 94;`

4. `mySuperChef.secretIngredient = "cardamom";`

5. `mySuperChef.cookieRecipe = "get 3 eggs, 2 1/2 cup flour, 1 tsp salt, 1 tsp vanilla and 1.5 cups sugar and mix them together. Bake for 10 minutes at 375. Yum!";`

6. `string recipe = mySuperChef.GetRecipe(56);`

7. After running all of the lines that will compile above, what's the value of recipe?

.....

.....



Here's a class with some private fields. Circle the statements below that **won't compile** if they're run from outside the class using **an instance of the object called mySuperChef**.

```
public class SuperChef
{
      public string cookieRecipe;
      private string secretIngredient;
      private const int loyalCustomerOrderAmount = 60;
      public int Temperature;
      private string ingredientSupplier;

      public string GetRecipe (int orderAmount)
      {
            if (orderAmount >= loyalCustomerOrderAmount)
            {
                  return cookieRecipe + " and the secret ingredient is "
                   + secretIngredient;
            }
            else
            {
                  return cookieRecipe;
            }
      }
}
```

This is a private constant. Its value can't be changed, and it can't be accessed outside of this class.

The only way to get the secret ingredient is to order a whole lot of cookies. Outside code can't access this field directly.

1. `string ovenTemp = mySuperChef.Temperature;`

#1 doesn't compile because you can't just assign an int to a string.

2. `string supplier = mySuperChef.ingredientSupplier;`

3. `int loyalCustomerOrderAmount = 54;`

4. `mySuperChef.secretIngredient = "cardamom";`

#2 and #4 don't compile because ingredientSupplier and secretIngredient are private.

5. `mySuperChef.cookieRecipe = "Get 3 eggs, 2 1/2 cup flour, 1 tsp salt, 1 tsp vanilla and 1.5 cups sugar and mix them together. Bake for 10 minutes at 375. Yum!";`

6. `string recipe = mySuperChef.GetRecipe(56);`

Even though you created a local variable called loyalCustomerAmount and set it to 54, that didn't change the object's loyalCustomerAmount value, which is still 60—so it won't print the secret ingredient.

7. After running all of the lines that will compile above, what's the value of recipe?

"Get 3 eggs, 2 1/2 cup flour, 1 tsp salt, 1 tsp vanilla and 1.5 cups sugar and mix them together. Bake for 10 minutes at 375. Yum!"

A few ideas for encapsulating classes

- **Think about ways the fields can be misused.**
  What can go wrong if they're not set properly?

- **Is everything in your class public?**
  If your class has nothing but public fields and methods, you probably need to spend a little more time thinking about encapsulation.

- **What fields require some processing or calculation to happen when they're set?**
  Those are prime candidates for encapsulation. If someone writes a method later that changes the value in any one of them, it could cause problems for the work your program is trying to do.



- **Only make fields and methods public if you need to.**
  If you don't have a reason to declare something public, don't. You could make things really messy for yourself by making all of the fields in your program public—but don't just go making everything private, either. Spending a little time up front thinking about which fields really need to be public and which don't can save you a lot of time later.

Encapsulation keeps your data pristine

Sometimes the value in a field changes as your program does what it's supposed to do. If you don't explicitly tell your program to reset the value, you can do your calculations using the old one. When this is the case, you want to have your program execute some statements any time a field is changed—like having Kathleen's program recalculate the cost every time you change the number of people. We can avoid the problem by encapsulating the data using private fields. We'll provide a method to get the value of the field, and another method to set the field and do all the necessary calculations.

A quick example of encapsulation

A Farmer class uses a field to store the number of cows, and multiplies it by a number to figure out how many bags of cattle feed are needed to feed the cows:

```
class Farmer
{
      private int numberOfCows;
}
```

We'd better make this field private so nobody can change it without also changing bagsOfFeed—if they get out of sync, that'll create bugs!

When you create a form to let a user enter the number of cows into a numeric field, you need to be able to change the value in the `numberOfCows` field. To do that, you can create a method that returns the value of the field to the form object.

```
public const int FeedMultiplier = 30;
public int GetNumberOfCows()
{
      return numberOfCows;
}

public void SetNumberOfCows(int newNumberOfCows)
{
      numberOfCows = newNumberOfCows;
      BagsOfFeed = numberOfCows * FeedMultiplier;
}
```

The farmer needs 30 bags of feed for each cow.

We'll add a method to give other classes a way to get the number of cows.

These accomplish the same thing!

We used camelCase for the private fields and PascalCase for the public ones.

And here's a method to set the number of cows that makes sure the BagsOfFeed field is changed too. Now there's no way for the two to get out of sync.

Properties make encapsulation easier

C# has special kinds of methods that make it easy to encapsulate your data. You can use **properties**, methods that are executed every time a field is called to set or return the value of the field, which is called a **backing field**.

We'll rename the private field to numberOfCows (notice the lowercase "n"). This will become the backing field for the NumberOfCows property.

You use properties by combining them with a normal field declaration. Here's the declaration for NumberOfCows.

This is a **get accessor**. It's a method that's run any time the NumberOfCows field is **read**. It has a return value that matches the type of the variable—in this case it returns the value of the private numberOfCows field.

This is a **set accessor** that's called every time the NumberOfCows field is **set**. Even though the method doesn't look like it has any parameters, it actually has one called **value** that contains whatever value the field was set to.

```
private int numberOfCows;

public int NumberOfCows
{
    get
    {
        return numberOfCows;
    }
    set
    {
        numberOfCows = value;
        BagsOfFeed = numberOfCows * FeedMultiplier;
    }
}
```

You ***use*** get and set accessors exactly like fields. Here's code for a button that sets the numbers of cows and then gets the bags of feed:

When this line sets NumberOfCows to 10, the set accessor sets the private numberOfCows field and then updates the public BagsOfFeed field.

Since the NumberOfCows set accessor updated BagsOfFeed, now you can get its value.

Even though the code treats NumberOfCows like a field, it runs the set accessor, passing it 20. And when it queries the BagsOfFeed field it runs the get accessor, which returns 300.

```
private void button1_Click(object sender, EventArgs e) {
    Farmer myFarmer = new Farmer();
    myFarmer.NumberOfCows = 10;

    int howManyBags = myFarmer.BagsOfFeed;

    myFarmer.NumberOfCows = 20;
    howManyBags = myFarmer.BagsOfFeed;
}
```

Build an application to test the Farmer class

Create a new Windows Forms application that we can use to test the Farmer class and see properties in action. We'll use the **`Console.WriteLine()`** method to write the results to the output window in the IDE.

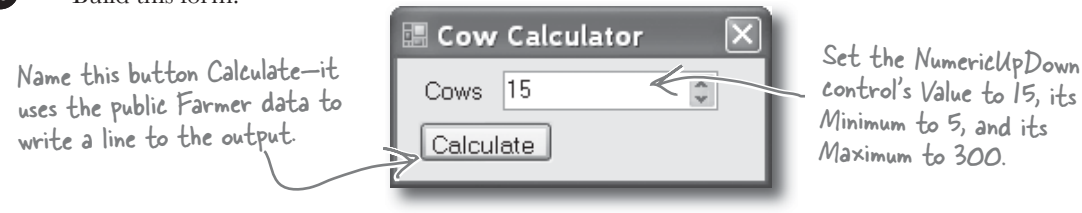


1. Add the Farmer class to your project:

```
public class Farmer {
    public int BagsOfFeed;
    public const int FeedMultiplier = 30;

    private int numberOfCows;
    public int NumberOfCows {
         (add the get and set accessors from the previous page)
    }
}
```

2. Build this form:



3. Here's the form for the code. It uses `Console.WriteLine()` to send its output to the Output window (which you can bring up by selecting "Output" from the View menu). You can pass several parameters to `WriteLine()`—the first one is the string to write. If you include "`{0}`" inside the string, then `WriteLine()` replaces it with the first parameter. It replaces "`{1}`" with the second parameter, "`{2}`" with the third, etc.

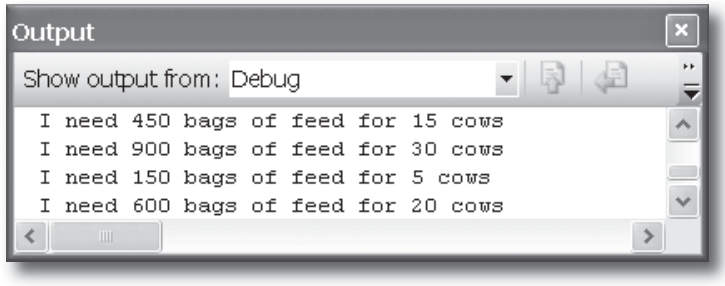
```
public partial class Form1 : Form {
    Farmer farmer;
    public Form1() {
        InitializeComponent();
        farmer = new Farmer() { NumberOfCows = 15 };
    }
    private void numericUpDown1_ValueChanged(object sender, EventArgs e) {
        farmer.NumberOfCows = (int)numericUpDown1.Value;
    }
    private void calculate_Click(object sender, EventArgs e) {
        Console.WriteLine("I need {0} bags of feed for {1} cows",
            farmer.BagsOfFeed, farmer.NumberOfCows);
    }
}
```

Use the Console.WriteLine() method to send a line of text to the IDE's Output window.

WriteLine() replaces "{0}" with value in the first parameter, and "{1}" with the second parameter.

Use automatic properties to finish the class

It looks like the Cow Calculator works really well. Give it a shot—run it and click the button. Then change the number of cows to 30 and click it again. Do the same for 5 cows and then 20 cows. Here's what your Output window should look like:



But there's a problem with the class. Add a button to the form that executes this statement:

```
farmer.BagsOfFeed = 5;
```

Now run your program again. It works fine until you press the new button. But press that button and then press the Calculate button again. Now your ouput tells you that you need 5 bags of feed—no matter how many cows you have!



Watch it!

Automatic properties are a C# 3.0 feature.

*If you're still using Visual Studio 2005 and C# 2.0, this code won't work. We highly recommend that you use Visual Studio 2008 Express. You can download it for **free**!*

Fully encapsulate the Farmer class

The problem is that your class **isn't fully encapsulated**. You used properties to encapsulate NumberOfCows, but BagsOfFeed is still public. This is a common problem. In fact, it's so common that C# has a way of automatically fixing it. Just change the public BagsOfFeed field to an ***automatic property***. And the IDE makes it really easy for you to add automatic properties. Here's how:

The prop–tab–tab code snippet adds an automatic property to your code.

1. Remove the BagsOfFeed field from the Farmer class. Put your cursor where the field used to be, and then type **prop** and press the tab key twice. The IDE will add this line to your code:

```
public int MyProperty { get; set; }
```

2. Press the tab key—the cursor jumps to `MyProperty`. Change its name to `BagsOfFeed`:

```
public int BagsOfFeed { get; set; }
```

Now you've got a property instead of a field. When C# sees this, it works exactly the same as if you used a backing field (like the private `numberOfCows` behind the public `NumberOfCows` property).

3. That hasn't fixed our problem yet. But there's an easy fix—just make it a **read-only property**:

```
public int BagsOfFeed { get; private set; }
```

Try to rebuild your code—you'll get an error on the line in the button that sets `BagsOfFeed` telling you that the **set accessor is private**. Now your Farmer class is better encapsulated!

What if we want to change the feed multiplier?

We built the Cow Calculator to use a `const` for the feed multiplier. But what if we want to use the same Farmer class in different programs that need different feed multipliers? You've seen how you poor encapsulation can cause problems when you make fields in one class too accessible to other classes. That's why you should **only make fields and methods public if you need to**. Since the Cow Calculator never updates `FeedMultiplier`, there's no need to allow any other class to set it. So let's change it to a read-only automatic property.



1. Remove this line from your program:

```
public const int FeedMultiplier = 30;
```

Use `prop-tab-tab` to add a read-only automatic property:

```
public int FeedMultiplier { get; private set; }
```

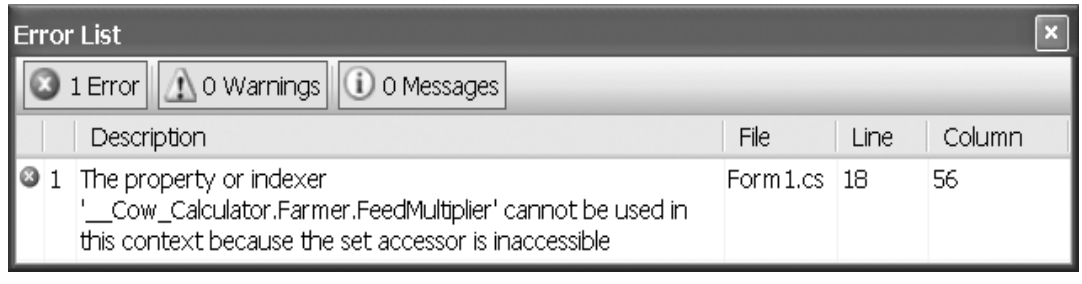
This automatic property acts just like an int field. It has a public get, which means any other class can read the value of FeedMultiplier. But since its set is private, that makes it read-only—it can only be set by an instance of Farmer.

2. Go ahead and make that change to your code. Then run it. Uh-oh—something's wrong! BagsOfFeed **always returns 0 bags**!

Wait, that make sense. `FeedMultiplier` never got initialized. It starts out with the default value of zero and never changes. When it's multiplied by the number of cows, it still gives you zero. So add an object initializer:

```
public Form1() {
    InitializeComponent();
    farmer = new Farmer() { NumberOfCows = 15, FeedMultiplier = 30 };
```

Uh-oh—the program won't compile! You should get this error:



You can only initialize public fields and properties inside an object initializer. So how can you make sure your object gets initialized properly if some of the fields that need to be initialized are private?

Use a constructor to initialize private fields

If you need to initialize your object, but some of the fields that need to be initialized are private, then an object initializer just won't do. Luckily, there's a special method that you can add to any class called a **constructor**. If a class has a constructor, then that constructor is the **very first thing that gets executed** when the class is created with the `new` statement. You can pass parameters to the constructor to give it values that need to be initialized. But the constructor **does not have a return value**, because you don't actually call it directly. You pass its parameters to the `new` statement. And you already know that `new` returns the object—so there's no way for a constructor to return anything.

All you have to do to add a constructor to a class is add a method that has the same name as the class and no return value.

1 **Add a constructor to your Farmer class**

This constructor only has two lines, but there's a lot going on here. So let's take it step by step. We already know that we need the number of cows and a feed multiplier for the class, so we'll add them as parameters to the constructor. We'll change `feedMultiplier` from a `const` to an `int`. We'll need a value for it, so let's make sure it gets passed into the constructor. We'll use the constructor to set the number of cows, too..

We'll change feedMultiplier from a const to an int field.

Notice how there's no "void" or "int" or another type after "public". That's because constructors don't have a return value.

```
private int feedMultiplier;

public Farmer(int numberOfCows, int feedMultiplier) {

    this.feedMultiplier = feedMultiplier;

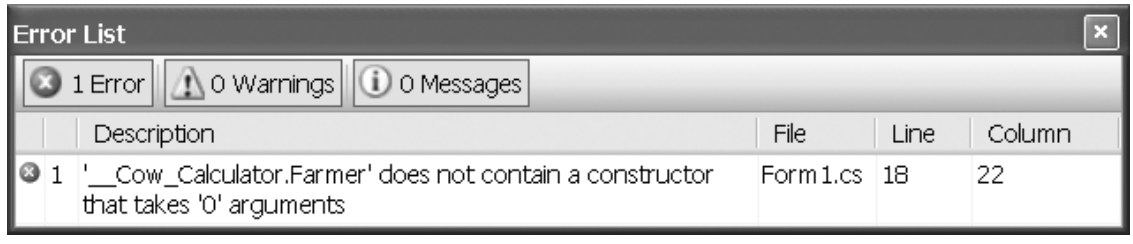
    NumberOfCows = numberOfCows;
}
```

Since we changed FeedMultiplier from a public const to a private int field, we changed its name so it starts with a lowercase "f". That's a pretty standard naming convention you'll see throughout the book.

The first thing we'll do is set the feed multiplier, because it needs to be set before we can call the NumberOfCows set accessor.

If we just set the private numberOfCows field, the NumberOfCows set accessor would never be called. Setting NumberOfCows makes sure it's called.

This is the error you'll get if your constructor takes parameters but your `new` doesn't have any.



2 **Now change the form so that it uses the constructor**

The only thing you need to do now is change the form so that the `new` statement that creates the `Farmer` object uses the constructor instead of an object initializer.

```
public Form1() {

    InitializeComponent();

    farmer = new Farmer(15, 30);
}
```

You already know that the form is an object. Well, it's got a constructor too! That's what this method is—notice how it's named Form1 (like the class) and it doesn't have a return value.

Here's where the new statement calls the constructor. It looks **just like any other new statement**, except that it has parameters that it passes into the constructor method. When you type it in, watch for the IntelliSense pop-up—it looks just like any other method.



Constructors Way Up Close

Let's take a closer look at the Farmer constructor so we can get a good sense of what's really going on.

Constructors don't return anything, so there's no return type.

This constructor has two parameters, which work just like ordinary parameters. The first one gives the number of cows, and the second one is the feed multiplier.

```
private int feedMultiplier;

public Farmer(int numberOfCows, int feedMultiplier) {
    this.feedMultiplier = feedMultiplier;
    NumberOfCows = numberOfCows;
}
```

We need to set the feed multiplier first, because the second statement calls the NumberOfCows set accessor, which needs feedMultiplier to have a value in order to set BagsOfFeed.

We need a way to differentiate the field called feedMultiplier from the parameter with the same name. That's where the "this." keyword comes in really handy.

Since "this" is always a reference to the current object, this.feedMultiplier refers to the field. If you leave "this" off, then feedMultiplier refers to the parameter. So the first line in the constructor sets the private feedMultiplier field equal to the second parameter of the constructor.

there are no Dumb Questions

Q: Is it possible to have a constructor without any parameters?

A: Yes. It's actually very common for a class to have a constructor without a parameter. In fact, you've already seen an example of it—**your form's constructor**. Look inside a newly added Windows form and find its constructor's declaration:

```
public Form1() {
  InitializeComponent();
}
```

That's the constructor for your form object. It doesn't take any parameters, but it does have to do a lot. Take a minute and open up `Form1.Designer.cs`. Find the `InitializeComponent()` method by clicking on the plus symbol next to "Windows Form Designer generated code".

That method initializes all of the controls on the form and sets all of their properties. If you drag a new control onto your form in the IDE's form designer and set some of its properties in the Properties window, you'll see those changes reflected inside the `InitializeComponent()` method.

The `InitializeComponent()` method is called inside the form's constructor so that the controls all get initialized as soon as the form object is created. (Remember, every form that gets displayed is just another object that happens to use methods that the .NET Framework provides in the System.Windows.Forms namespace to display windows, buttons and other controls.)



Watch it!

When a method's parameter has the same name as a field, then it masks the field.

Did you notice how the constructor's `numberOfCows` parameter looks just like the backing field behind the NumberOfCows property? If you wanted to use to the backing field in of the constructor, you'd use "`this.`"—`numberOfCows` refers to the parameter, and `this.numberOfCows` is how you'd access the private field.

there are no Dumb Questions

Q: Why would I need complicated logic in a get or set accessor? Isn't it just a way of creating a field?

A: Because sometimes you know that every time you set a field, you'll have to do some calculation or perform some action. Think about Kathleen's problem—she ran into trouble because the form didn't run the method to recalculate the cost of the decorations after setting the number of people in the DinnerParty class. If we replaced the field with a set accessor, then we could make sure that the set accessor recalculates the cost of the decorations. (In fact, you're about to do exactly that in just a couple of pages!)

Q: Wait a minute—so what's the difference between a method and a get or set accessor?

A: There is none! Get and set accessors are a special kind of method—one that looks just like a field to other objects, and called whenever that field is set. Get accessors always return a value that's the same type as the field, and set accessors always take exactly one parameter called `value` whose type is the same as the field. Oh, and by the way, you can just say "property" instead of "get and set accessor".

Q: So you can have ANY kind of statement in a property?

A: Absolutely. Anything you can do in a method, you can do in a property. They can call other methods, access other fields, even create objects and instances. But they only get called when a field gets accessed, so it doesn't make sense to have any statements in them that don't have to do with getting or setting the field.

Q: If a set accessor always takes a parameter called value, why doesn't its declaration have parentheses with "`int value`" in them, like you'd have with any other method that takes a parameter called value?

A: Because C# was built to keep you from having to type in extra information that the compiler doesn't need. The parameter gets declared without you having to explicitly type it in, which doesn't sound like much when you're only typing one or two—but when you have to type a few hundred, it can be a real time saver (not to mention a bug preventer).

Every set accessor ***always*** has exactly one parameter called `value`, and the type of that parameter **always** matches the type of the field. C# has all the information it needs about the type and parameter as soon as you type "`set {`". So there's no need for you to type any more, and the C# compiler isn't going to make you type more than you have to.

Q: Wait, a sec—is that why I don't add a return value to my constructor?

A: Exactly! Your constructor doesn't have a return value because ***every*** constructor is always `void`. It would be redundant to make you type "`void`" at the beginning of each constructor, so you don't have to.

Q: Can I have a get without a set or a set without a get?

A: Yes! When you have a get accessor but no set, you create a read-only field. For example, the SecretAgent class might have a ReadOnly field for the name:

```
string name = "Dash Martin";
public string Name {
   get { return name; }
}
```

And if you create a property with a set accessor but no get, then your field can only be written, but not read. The SecretAgent class could use that for a Password field that other spies could write to but not see:

```
public string Password {
 set {
  if (value == secretCode) {
    name = "Herb Jones";
  }
}
```

Both of those techniques can come in really handy when you're doing encapsulation.

Properties (get and set accessors) are a special kind of method that's only run when another class reads or writes a property.

Sharpen your pencil

Take a look at the get and set accessors here. The Form that is using this class has a new instance of CableBill called thisMonth and calls the GetThisMonthsBill () method with a button click. Write down the value of the amountOwed variable after the code below executed.

```
public class CableBill {
    private int rentalFee;
    public CableBill(int rentalFee) {
        this.rentalFee = rentalFee;
        discount = false;
    }

    private int payPerViewDiscount;
    private bool discount;
    public bool Discount {
        set {
            discount = value;
            if (discount)
            payPerViewDiscount = 2;
            else
            payPerViewDiscount = 0;
        }
    }

    public int CalculateAmount(int payPerViewMoviesOrdered) {
        return (rentalFee - payPerViewDiscount) * payPerViewMoviesOrdered;
    }
}
```

1.
```
CableBill january = new CableBill(4);
MessageBox.Show(january.CalculateAmount(7).ToString());
```
What's the value of amountOwed?

2.
```
CableBill february = new CableBill(7);
february.payPerViewDiscount = 1;
MessageBox.Show(february.CalculateAmount(3).ToString());
```
What's the value of amountOwed?

3.
```
CableBill march = new CableBill(9);
march.Discount = true;
MessageBox.Show(march.CalculateAmount(6).ToString());
```
What's the value of amountOwed?

there are no Dumb Questions

Q: I noticed that you used uppercase names for some fields but lowercase ones for others. Does that matter?

A: Yes—it matters to you. But it doesn't matter to the compiler. C# doesn't care what you name your variables, but if you choose weird names then it makes your code hard to read. Sometimes it can get confusing when you have variables that are named the same, except one starts with an uppercase letter and the other starts with a lowercase one.

Here are a few tips about variable names to help you keep it straight. They're not hard-and-fast rules—the compiler doesn't care whether a variable is uppercase or lowercase—but they're good suggestions to help make your code easier to read.

1. When you declare a private field, it should be in camelCase and start with a lowercase letter. (It's called camelCase because it starts with a lowercase letter and additional words are uppercase, so they resemble humps on a camel.)

2. Public properties and methods are in PascalCase (they start with an uppercase letter).

3. Parameters to methods should be in camelCase.

4. Some methods, especially constructors, will have parameters with the same names as fields. When this happens, the parameter **masks** the field, which means statements in the method that use the name end up referring to the parameter, not the field. Use the `this` keyword to fix the problem—add it to the variable to tell the compiler you're talking about the field, not the parameter.

Sharpen your pencil

This code has problems. Write down what you think is wrong with the code, and what you'd change.

```
class GumballMachine {
      private int gumballs;

      private int price;
      public int Price
      {
        get
        {
            return price;
        }
      }
      public GumballMachine(int gumballs, int price)
      {
            gumballs = this.gumballs;
            price = Price;
      }
      public string DispenseOneGumball(int price, int coinsInserted)
      {
            if (this.coinsInserted >= price) { // check the field
                  gumballs -= 1;
                  return "Here's your gumball";
            } else {
                  return "Please insert more coins";
            }
      }
}
```

Sharpen your pencil Solution

Write down the value of the amountOwed variable after the code below executed.

```
1. CableBill january = new CableBill(4);
   MessageBox.Show(january.CalculateAmount(7).ToString());

2. CableBill february = new CableBill(7);
   february.payPerViewDiscount = 1;
   MessageBox.Show(february.CalculateAmount(3).ToString());

3. CableBill march = new CableBill(9);
   march.Discount = true;
   MessageBox.Show(march.CalculateAmount(6).ToString());
```

What's the value of amountOwed?

28

What's the value of amountOwed?

won't compile

What's the value of amountOwed?

42

Sharpen your pencil Solution

This code has problems. Write down what you think is wrong with the code, and what you'd change.

Lowercase `price` refers to the parameter to the constructor, not the field. This line sets the PARAMETER to the value returned by the Price get accessor, but Price hasn't even been set yet! So it doesn't do anything useful. If you change the constructor's parameter to uppercase Price, this line will work properly.

The "this" keyword is on the wrong "gumballs". `this.gumballs` refers to the property, while `gumballs` refers to the parameter.

This parameter masks the private field called Price, and the comment says the method is supposed to be checking the value of the `price` backing field.

```
public GumballMachine(int gumballs, int price)
{
        gumballs = this.gumballs;
        price = Price;
}

public string DispenseOneGumball(int price, int coinsInserted)
{
    if (this.coinsInserted >= price) { // check the field
            gumballs -= 1;
            return "Here's your gumball";
    } else {
            return "Please insert more coins";
    }
}
```

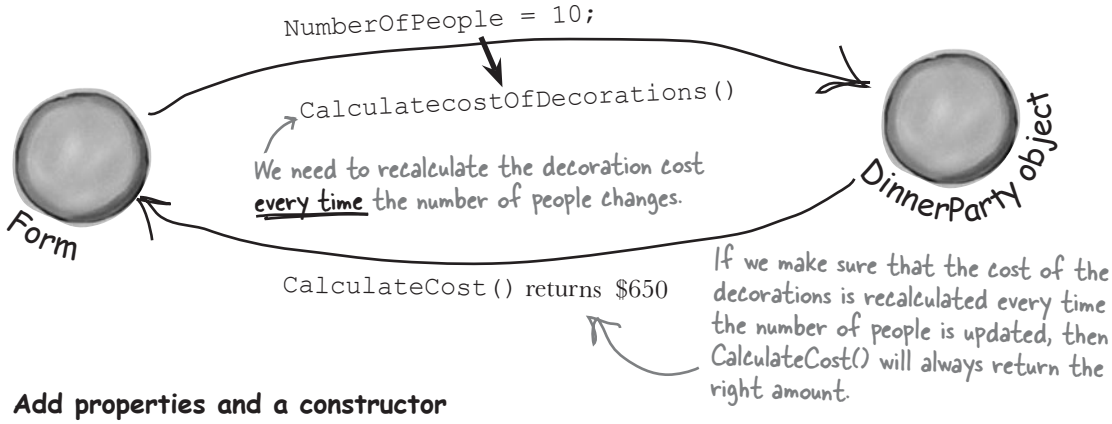
The "this" keyword is on a parameter, where it doesn't belong. It should be on `price`, because that field is masked by a parameter.



Use what you've learned about properties and constructors to fix Kathleen's Party Planner program.

1 How to fix the Dinner Party calculator

If we want to fix the DinnerParty class, we'll need a way to make sure that the `CalculateCostOfDecorations()` method gets called every time that NumberOfPeople changes.



2 Add properties and a constructor

All you need to do to fix Kathleen's problem is make sure the DinnerParty class is well-encapsulated. You'll start by **changing NumberOfPeople to a property** that always calls CalculateCostOfDecorations() any time it's called. Then you'll **add a constructor** that makes sure the instance is initialized properly. Finally, you'll **change the form** so it uses the new constructor. If you do this right, that's the only change you'll need to make to the form.

- You'll need to create a new property for `NumberOfPeople` that has a set accessor which calls `CalculateCostOfDecorations()`. It'll need a backing field called `numberOfPeople`.
- The `NumberOfPeople` set accessor needs to have a value to pass as the parameter to the `CalculateCostOfDecorations()` method. So add a private `bool` field called `fancyDecorations` that you set every time `CalculateCostOfDecorations()` is called.
- Add a constructor that sets up the class. It needs to take three parameters for the Number of People, Healthy Option, and Fancy Decorations. The form currently calls two methods when it initializes the DinnerParty object—move them into the constructor.

```
dinnerParty.CalculateCostOfDecorations(fancyBox.Checked);
dinnerParty.SetHealthyOption(healthyBox.Checked);
```

- Here's the constructor for the form—everything else in the form stays the same:

```
public Form1() {
    InitializeComponent();
    dinnerParty = new DinnerParty((int)numericUpDown1.Value,
                        healthyBox.Checked, fancyBox.Checked);
    DisplayDinnerPartyCost();
}
```



Use what you've learned about properties and constructors to fix Kathleen's Party Planner program.

```
public class DinnerParty {
    const int CostOfFoodPerPerson = 25;

    private int numberOfPeople;
    public int NumberOfPeople {
        get { return numberOfPeople; }
        set {
            numberOfPeople = value;
            CalculateCostOfDecorations(fancyDecorations);
        }
    }
    private bool fancyDecorations;

    public decimal CostOfBeveragesPerPerson;
    public decimal CostOfDecorations = 0;

    public DinnerParty(int numberOfPeople, bool healthyOption, bool fancyDecorations) {
        this.NumberOfPeople = numberOfPeople;
        this.fancyDecorations = fancyDecorations;
        SetHealthyOption(healthyOption);
        CalculateCostOfDecorations(fancyDecorations);
    }

    public void SetHealthyOption(bool healthyOption) {
        if (healthyOption) {
            CostOfBeveragesPerPerson = 5.00M;
        } else {
            CostOfBeveragesPerPerson = 20.00M;
        }
    }

    public void CalculateCostOfDecorations(bool fancy) {
        fancyDecorations = fancy;
        if (fancy) {
            CostOfDecorations = (NumberOfPeople * 15.00M) + 50M;
        } else {
            CostOfDecorations = (NumberOfPeople * 7.50M) + 30M;
        }
    }

    public decimal CalculateCost(bool healthyOption) {
        decimal totalCost = CostOfDecorations
           + ((CostOfBeveragesPerPerson + CostOfFoodPerPerson) * NumberOfPeople);

        if (healthyOption) {
            return totalCost * .95M;
        } else {
            return totalCost;
        }
    }
}
```

Now that numberOfPeople is private, there's no way for the form to change it without also recalculating the cost of the decorations. That'll fix the bug that almost cost Kathleen one of her best clients!

By using a property, you can make sure that the cost of decorations is recalculated every time the number of people changes.

Be careful how you use "this.". Make sure you put it in front of NumberOfPeople so that it calls the set accessor.

Make sure you store the fancy decorations in a field so the NumberOfPeople set accessor can use it.

And you'll need to put it in front of "fancyDecorations" because the fancyDecorations parameter masks the private field with the same name.

Your object's family tree



Sometimes you DOwant to be just like your parents.

Ever run across an object that *almos!* ooes exactly what you want your objec1 to do? Found yourself wishing that if you could just ***change*** *a few* ***things***, mat object would be perfect? Well that's just one reason that inheritance is one of the most powerful concepts and techniques in the C# language. Before you're through this chapter, you'll learn how to subclass an object to get its behavior, but keep the flexibility 10 make changes to that behavior. You'll avoid duplicate code, model the real world more closely, and end up Wth code triers easier to maintain.

Kathleen does birthday parties, too

Now that yo" got your program working, Kathleen is using il all the time. But she d(u·i just handle dinner parties she does birthdavs l'.." ,,,,,I they" riced a little differently. SI.,'11 need vou 1'• add birthd..vs In her program.

I just got 0 call for a birthday party for 10 peeele. Can your p0 ogi am handle that?



Most of the changes have to do with cakes and writing

These are both the same as the dinner party.

Cost Estimate for a birthday rarty

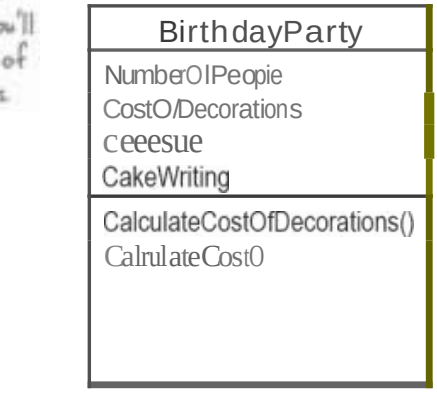
- $25 per person.
- Tbe-e cre two cotcce for the cost of decorations. If a client goes with the normal decorations, It ·a $7,50 per person wIth a $30 decorating fee. A client can also upgrade the party decorations to the "Fancy Optic.a"-rtct costs $15 per pe-son with a $50 one-time decorating fee.
- When the party has four people or less, use an 8-inch cake ($40). Otherwise, she uses a 16-inch cake ($15).
- Writing on the cake costs $.25 Io- each letter. The 8-inch cake can have up to 16 letters of writing, and the 16-inch one can have up to 40 letters of writing.

The application should handle both types of cc-tee. Iee c tab control, one tab Fo- eccb kind of party.

We l1eed a BirthdayParty class

Modifying your program to calculate the cost "f Kathleen's l,;rthd,,\' parties means adding a new class and changing Ih,· form to let \011 hamll,-both kind, of pari;"".

You'll do all this in a minute—but first you'll need to get a sense of what the job involves.



Here', what we "eed to do:

- **Create Q new BirthdayParty class**
  Y"ur new class "ill "'·ed1(<1 calculate the costs, deal "ith decorations, and check the size of the writing Oil the cake.

- **Add a TabControl to your form**
  Each lal, <1)\ the li,nn is a ["I like the GroupBox control you used to show how ""uh cash Joe and Bob had back ill Chapter 3. Just di"k on the tab you want {" display, and drag controls ill1<' it.

- **Label the first tob Clod move the Dinner Porty eentrets onto it**
  Yo,,'11 drag eaeh of the controls lhat handle the dinner party int" the new tah. They'll work exactly lik,-before, b,t! they'll only be displayed when the dinne-r party tab is selected.

- **Label the second tab and add new Birthday Party controls onto it**
  You'll design the i,,terface fi". handling birthday parties just Iik,' you did fi>r the dinner p"rt;",.

- **Wire your birthday party class up to the controls**
  Now all you need 10 do is add a BirthdayParty reference to the form's fields, and add tilt, code to each or vour ucw controls t" s(tlt"t it use its lI",tl"ds ,uid properties

there are no Dumb Questions

Q: Why can't we just create a new Instance 01 OlnnerParly. tike Mike did when h, wanted to compare three routes in his navigation program?

A: Because ,f you created another instance 01 the DinnerParty class. you'd only be able to use it to plan extra dinner cartes Two instances 01 the same class can be really useful if you need to manage twodtfferent peces 01 the same kind of data. But if you need to store different kindSOf dota, you'll need different classes to do it

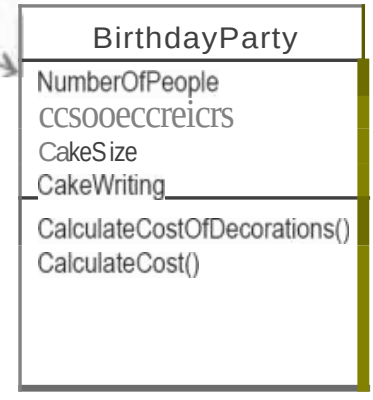
Q: How do I know what to put in the new class?

A: Before you can start building a class, you need to know what problem it's supposed to solve. That's why you had to talk to Kathleen—she's going to be using the program, Good th,ng you took a lol of notes! You can come up with your class's methods, fields, and properties by thinking about its behavior (what it *needs to do)* and its state (what it *needs 10 know*).



Add birthday parties to Kathleen's party plannmg prog ram

Make sure you use decimal as the type for the fields and properties that hold currency.



O Add **the new BirthdayParty** class to your program

You already know how you'll handle the Number( )11eople property ;llld the (:,,'I( )fDecc'r"li,ms methods —they're just lik,' Iheir counterparts ill DinnerParty. Start by creating yc,nr new class and adding th,'''-, Then add Ihe rest of the behavior:

If the cake writing i too ong for the cake, the set accessor e...ts the backing field down to size. So you'll need to make sure to reload the text into the text box every time the te :t changes or the ber of people changes (in case there's a long string and she cuts dow.. to a smaller cake).

1 • Add a CakeSize integer property. Make a private method called CalculateCakeSize() that sets CakeSize 10 either 8 '" 16 depending " 11 the ml1nl,,'r , ,I' 1"-"1'1" . You'll need to call il from th,' constructor and the Number(Ill',-" pl,-set accessor.

★ You'll need a CakeWriting string property I" hold the writing on the cake. We'll give you the code for Ihi, one.

We don't want any other methods changing the value of CakeWriting.

• The CakeWriting set accessor checks CakeSize because "'''erent size 0 f ca k,-",,,, hold different numbers 01 ` letters. Then it 11"-' `value.Length` I" check how long the string i,,_ If it's too long, instead of setting the private field, the set ,I<'';'ssor P'ops U]' « nessage bc.x that says, "T,,,, many letters for a]G-i",], cake" (or 8-inch cake).

• Every string has a `Substring()` method that returns a portion of the string. CakeWriting uses il to cut the size 01 Ih,' cak,' writing d,wn —if the numl>er •,l' people changes and reduce Ilie ,oak,' size, you'll need to rut d"'''1 the string, 1'. ',

★ Finally, add the CalculateCost() ",..Ih"d, lI,n instead " l' taking ,he decora i,'" cost and adding the cost of beverages (which is what happens in DinnerParty), it'll add the cost of the' cake.

Click on the tabs to switch between them. Use the TabCollection property to change the text for each tab. Click the "..." button next to it and select each tab's Text property.



O Update the form to add tabs

Drag a TabControl """ „1' tl,,' ,," ,II" rx ,,,,,,I "„1', vour f,,.m. ,,,,,I resize it '0 i, tak,-, up the enti · form. Change the 1"Xl ;,1' each tab using the TabPages property: a h"rton shows up in the Properties \\'indow next 1" Ill<' property. \Vhen you click il. the 11)E I"ps up a window lhal lets you edit the properties of each tab. S"I the Text property , ,(the tabs to "Dinner Party" "",I "Birthdav Party".

Name the first tab and move the Dinner Party controls onto it

Y" u'li drag the "„-h of I],e controls that h"mlk the dinner party i",o the new lah. They'll work exactly like tln-y do now. In,t they'll only be displayed wh,-" ,h'll tab is selected.

After you drag the Dinner Party controls onto the tab, they'll only le visible when the Dinner Party tab is selected

Build the Birthday Party user interface

The Birthday Party tab has a NumericUpDown control for the number of people, a CheckBox control for fancy decorations, and a Label control with a 3D border for the cost. Then you'll add a TextBox control for the cake writing.



You'll need this property

Here's the code for the BirthdayParty.CakeWriting property—it'll come in handy:

```
private string cakeWriting = "";
public string CakeWriting {
    get { return this.cakeWriting; }
    set {
        int maxLength;
        if (CakeSize == 8)
            maxLength = 16;
        else
            maxLength = 40;
        if (value.Length > maxLength) {
            MessageBox.Show("Too many letters for a " + CakeSize + " inch cake");
            if (maxLength > this.cakeWriting.Length)
                maxLength = this.cakeWriting.Length;
            this.cakeWriting = cakeWriting.Substring(0, maxLength);
        }
        else this.cakeWriting = value;
    }
}
```

This property is a little more complex than the one you've seen before. It checks the cake size to see if it's too long for the cake, using the maxLength variable to store the maximum length. If it's too long, it gives an error message and then cuts the backing field down to the right size, so it can be reloaded into the text box.

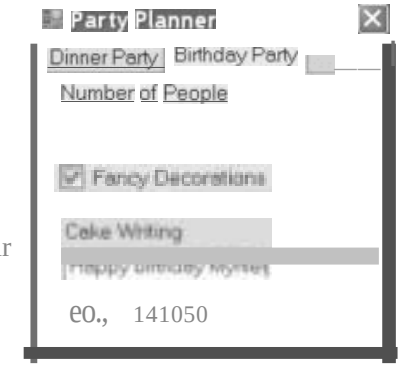
Did you notice how we left out some of the brackets? When you only have one statement in a code block, you don't need to add curly brackets around it

Every string has a Substring() method that returns a portion of the string. This one cuts it down to the allowed length, so you'll need to reload the writing into the textbox when the text or cake size change

Put it all together

All the pieces are there, now it just a matter of writing a few more lines of code to make the controls work.

- Add a BirthdayParty object to the form. Make sure you instantiate it.
- Add code to the NumericUpDown control's ValueChanged event handler method to set the object's NumberOfPeople property.
- Make the Fancy Decorations checkbox work.
- Add a DisplayBirthdayPartyCost() method to all of the event handlers, so the cost label is updated automatically any time there's a change.



Run it

Make sure the program works the way it's supposed to. Check that it pops up a message box if the writing is too long for the cake. Make sure the price is always right. Once it's working, you're done!



Add birthd ay parties to Kathleen's party planning program.

```
public partial class Formi : Form
    DinnerParty dinnerParty;
    BirthdayParty birthdayParty;
    public remit: {
        InitializeComponent();
        dinnerParty = new DinnerParty((int)numericUpDownl.Value,
                                      healthyBox.Checked, fancyBox.Checkedl;
        DisplayDinnerPartyCost();

        birthdayParty - new BirthdayParty((int)numberBirthday.Value,
                                      fancyBirthday.Checked, cakeWriting.Text);
        DisplayBirthdayPartyCostll;


    II The fancyBox, healthyBox, and numericUpDownl event handlers and
    II the DisplayDinnerCost() method are identical to the ones in the
    II Dinner Party exercise at the end of Chapter 5.


    private void nwnberBirthday_ValueChanged(object sender, EventArgs e) (
        birthdayParty.NumberOfPeople = (int)numberBirthday.Value;
        DisplayBirthdayPartyCost();


    private void fancyBirthday_CheckedChanged(object sender, EventArgs e) {
        birthdayParty.CalculateCostOfDecorations{fancyBirthday.Checked);
        DisplayBirthdayPartyCostll;


    private void cakeWriting_TextChanged(object sender, EventArgs e) {
        birthdayParty.Cakewriting = cakeWriting.Text;
        DisplayBirthdayPartyCostll;


    private void DisplayBirthdayPartyCost() {
        cakeWriting.Text = birthddyParty.CakeWriting;
        decimal cost = birthdayParty.Calculatecostl1;
        birthdayCost.Text = cost,ToString("c"l;
```

The Bi..thdayParty instance is initialized in the form's constructor, just like the instance of DinnerParty

The CheckBox and NumericUpDown controls' event handlers are just like the ones for the dinner party

All the intelligence for dealing with making sure the writing, the number of people, and the cake size are built into the NumberOfPeople and CakeWriting set accessors, so the form just has to set and display the values.

The way that the form handles the cake writing can be really simple because the BirthdayParty class is **well encapsulated**. All the form has to do is use its controls to set the properties on the object, and the object takes care of the rest.

```
using System,Windows.Forms;

public class BirthdayParty {
    public ccnst; int CostOfFoodPerPerson   25;

    pUblic decimal CostOfDecorations = 0;
    private bool fancyDecorations;
    public int CakeSize;

    public BirthdayParty(int numberOfPeople, bool fancyDecorations, string cakeWriting) {
        this.numberOfPeople = numberOfPeople;
        this.fancyDecorations = fancyDecorations;
        CalculateCakeSize();
        this.CakeWriting = cakeWriting;
        CalculateCostOfDecorations(fancyDecorations);
    }

    private void CalculateCakeSize() {
        if (NumberOfPeople <= 4)
            CakeSize = 8;
        else
            CakeSize = 16;
    }

    private string cakeWriting = "";
    public string CakeWriting {
        get { return this.cakeWriting;
        set
            int maxLength;
            if (CakeSize == 8)
                maxLength = 16;
            else
                maxLength = 40;
            if (value.Length > maxLength)
                MessageBox.Show("Too many letters for a " + cexesiae + " inch cake");
                if (maxLength > this.cakeWriting.Length)
                    maxLength = this.eakeWriting.Lenqth;
                this.cakeWriting = cakeWriting.Substring(0, maxLength);

            else
                this.cakeWriting = value;
```

When the BirthdayParty object is initialized, it needs to know the number of people, the fancy decorations and the writing on the cake, so it can start out with the right cake cost when CalculateCost() is called.

The constructor's calling the set accessor to set the cake writing, in case the parameter is too long for the cake, so it's got to calculate the cake size first.

The constructor sets the properties and then runs the calculations.

The CalculateCakeSize() method sets the CakeSize field. It's called by the NumberOfPeople set accessor and the CalculateCost() method.

The CakeWriting property makes sure that the cake's writing is never too long for the zake »ee Its set accessor checks the cake size, then uses the backing fields Length property to ...ake sure it's not too long. If it is, it cuts the string down to the right size



Add birthday parties to Kathleen's party planning program

We're using decimal because we're dealing with prices and currency

```
pUblic decimal CalculateCost()
    eecr  l TotalCost _ CostOfDecorations + (CostOfFoodPerPerson • NumberOfPeople);
    deci  l CakeCost
    .if (CakeSize _· 8)
        CakeCost • 40M . CakeWriting.Length    .25M;
    else
        CakeCost _ 75M + CakeWriting.Length    .25M;
    return TotalCost • Cakecost;
```

The CalculateCost() method is a lot like the one from DinnerParty, except that it adds the cost of the cake instead of the Healthy Choice option

```
private int numberOfPeople;
public int NumberOfPeople (
    get l return numberOfPeople;
    set {
        numberOfPeople • value;
        CalculateCostOfDecorations(fancyDecorations);
        CalculateCakeSize();
        this.CakeWriting _ CakeWriting;
```

J

Thi. ...ethod is just like the one ... the DinnerParty class

```
public void CalculateCostOfDecorations(bool fancy)
    fancyDecorations    fancy;
    If (fancy)
        CostOfDecorations • (NumberOfPeople
    else
        Co,", tOtDecoration" • (NumberOfPeople • 7.5OHI + 30M;
```

Curly brackets are optional for single-line blotks

A lot of times you'll have an if state...ent or while loop that's just got a single state ent inside its block When that happens a lot, you can end up with a whole lot of *t* ly brackets—and that can be a real eyesore! C# helps you avoid that proble... by letting you drop the curly brackets if there's just one state...ent. So the is perfectly valid syntax for a loop and an if state...ent:

```
for (int i = 0; i < 10; i++)
    DoTheJob(i);
```

```
if (..yValue == 36)
    myValue #= 5;
```

Olle more thing... call you add a $100 fee for parties over 12?

Kathleen's gotten s(nnn-h ▷",in,-" using your program Ihat she can afford ttl charge a little more for some of her larger clients. So what would i, tak,·1" change y('Urprogram to add in the extra charge?

- Change the JDinnerParty.(!alculateCost() 10check NumberOfPeople and add $100 to the return value if it's over 12.
- Do the exact same thing for BirthdayParty.CalculateCost().

Take a ",innt!- and Ihink about how vou'd add a fee to both the DinnerParty and BirthdayParty class. What ,'tld,- would you write? Where would it have 10go?

Easy enough... bUl what happens if Iher" are three ,imil,,, classes? Or four? Or twelve? And what if vou had III maiwain Ihal code and make mor(changes lal,-r? What if you had Itl milk,- the ***same exact change*** to five or six ***closely related*** classes?

C O



Wow, rd have to write the same code over ond over ogain_That's a really inefficient way to work. There's got to be 0 better way!

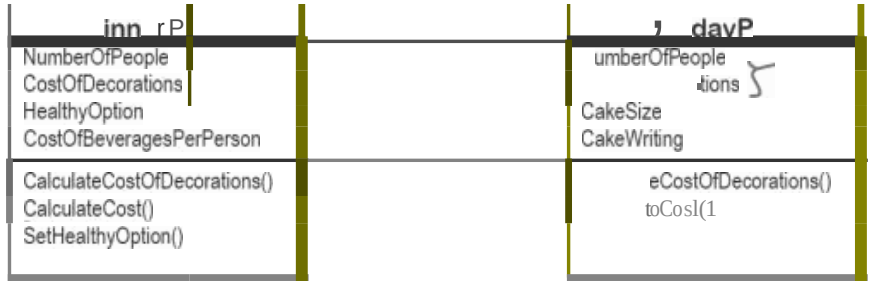
You're rightl Having the same code repeated in different classes is inefficient and error-prone.

Lucky for us, (:# gives us a better way Itl hltild classes that are related Itl each other, and share behavior: *inheritance*.

Whel1 your classes use il1herital1ce, you ol1ly l1eed to write your code ol1ce

It's no coincidence that yuur IlinnerParty and BirthdayParty classes have a 101 of the same code. When you write C# programs you on, · " create classes that represent things ill the real world - and those things are usually related (0 each oth,',-, Your classe have **sirnilar** cod.. because Ih,' things they represent ill the n'al world a birthday party and a dinner party— have **similar** b..**haviors**.

Kathleen needs to figure out the cost of her parties, no ..atter what kind of parties they are

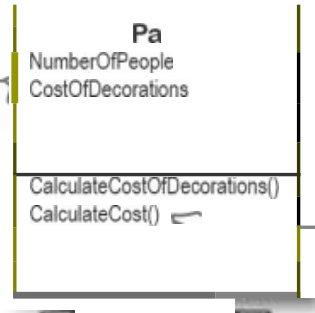


A birthday party handles the number of people and the cost of decorations in **almost** the sa..e way as a dinne-party

Dinner parties and birthday parties are both parties

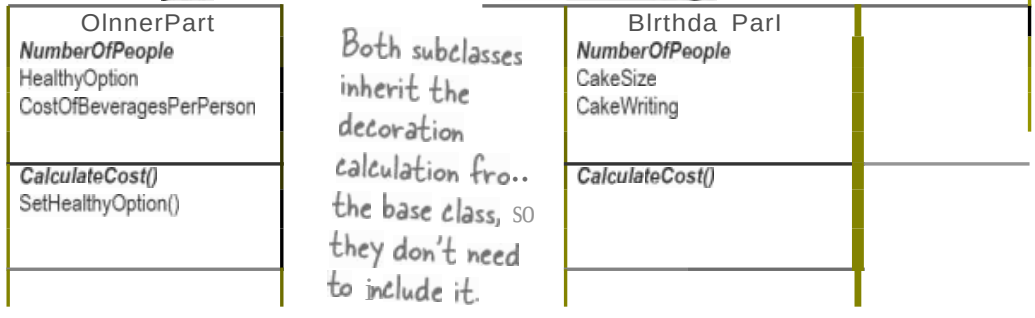
\\'he" you have two classe that are mor(specific case "l' something more general, you can set Ih"lll u!, to in h..rit from the same class. When you do that, each ,,1' Ih"1ll is a **subclass** of the same **base class**.

Both kinds of parties have to keep track of the nu...ber of people and the cost of decorations, so you can move that into the base class.



The way both parties handle the number of people and calculating the total cost is si..ilar but **distinct**. We can break up the behavior for these things so si..ilar part is in the base class, while putting the distinct pieces in the two subclasses

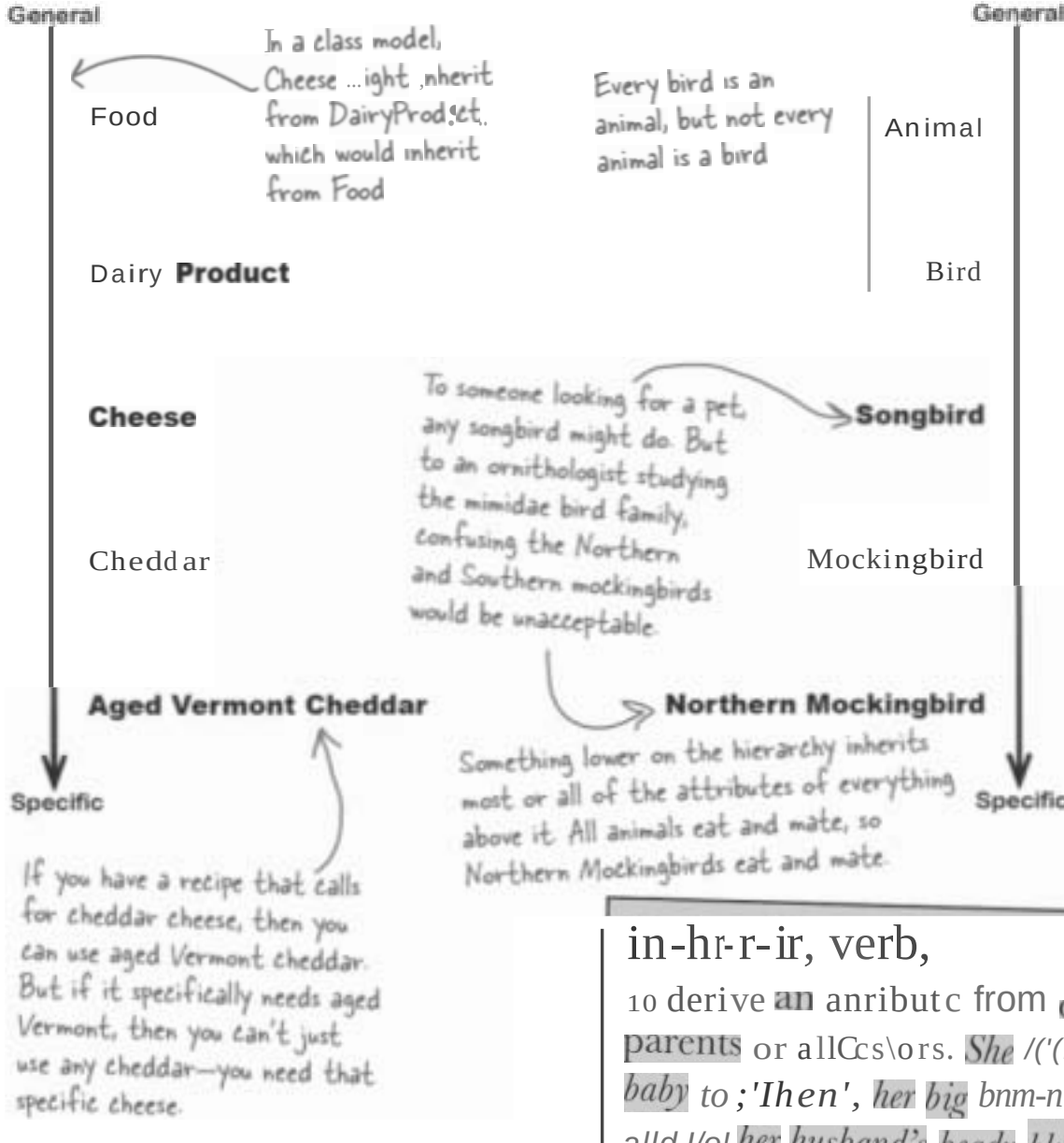
This arrow in the class diagra... ..eans the DinnerParty class inherits from the Party class.



Both subclasses inherit the decoration calculation fro.. the base class, so they don't need to include it.

Build up your class Model by starting general and getting more specific

C# programs use inheritance because it mimics the real-world relationships we have between things. Real-world things often fall into a hierarchy that goes from more general to more specific, and your programs have their own class hierarchy that does the same thing. In your class models, classes further down in the hierarchy inherit from those above it.



in-hr-r-ir, verb,

10 derive an anribute from one's parents or allCcs\ors. *She /('(II/InI thr baby to ;'Ihen', her big bnm-n eyes. alld I/o! her husband's beady blue ones.*

How would you desigl1 a zoo simulator?

Lions and tigers and h..ars.. " h my! Also, hippos, wolves, and the ◁occasional cat. Y" nr j" h is ı·· design a program that simulates a 1◁oo. 1)) " n'l get ı' ıu excited—we're no! going 10actually build the code, just design the classe ı·· represent l]", animals.)

We've h.·.." given a list ,,1' ,,,,,,,, of 1lıe animals that will h.· in the program, In" n„1 all of Ih"l11. We know Ihal each animal will h.· represented by an object, and that the objects will move around in the simulator. doing whatever il is that each particular 'lIlilllal is programmed 10d„_

More importantly, we want the program t(1", ..asy for other programmers 10 maintain. which means they'll need I" I,., able 1" add their own classes later »n if they want I,. add new animals 1< the simulat('1'

So what's the first step? Well. before we ,·m talk ab('U! **specific** animals, we need I" figure 0"1 the g..n..**ral** things they have in common, abstract characteristic Ihal *all* animals have. Th"n we can build those characteristics inlo a class Ihat all animal classe can inherit from.

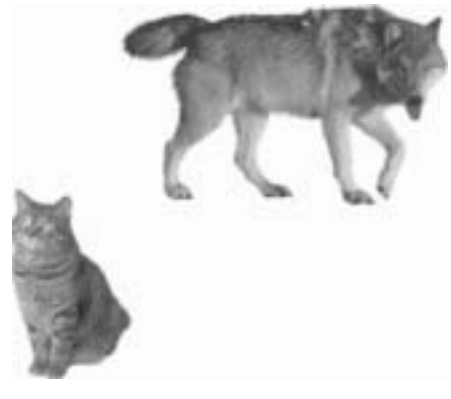
O Look for things the animals have in common

Take a look at Ihese six animals. What do a lion, a hippo, a tiger, a cat, a wolf, and a dalmatian have in common? How are they related:'You'll need to figure out their relationships s(you ,."" come "1' with a class model that include all "r them.









Use inheritance to avoid duplicate code in subclasses

You've already got a good idea that duplicate code sucks. It's hard to maintain, and always leads to headaches down the road. So let's think about fields and methods for an Animal base class that you only have to write once... so each of the subclasses will inherit from them. Let's start with the public fields:

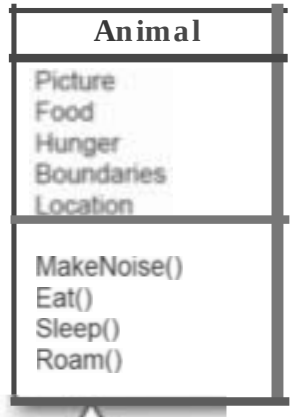
- `Picture`: an image that you'll input using PictureBox.
- `Food`: the type of food this animal eats. Right now there are only two values: meat or grass.
- `Hunger`: an integer representing the hunger level of the animal. This changes depending on what and when the animal eats.
- `soundLee`: a reference to a class that stores the width, height, and location of the perimeter that the animal will roam around in.
- `Location`: the X and Y coordinates where the animal's standing.

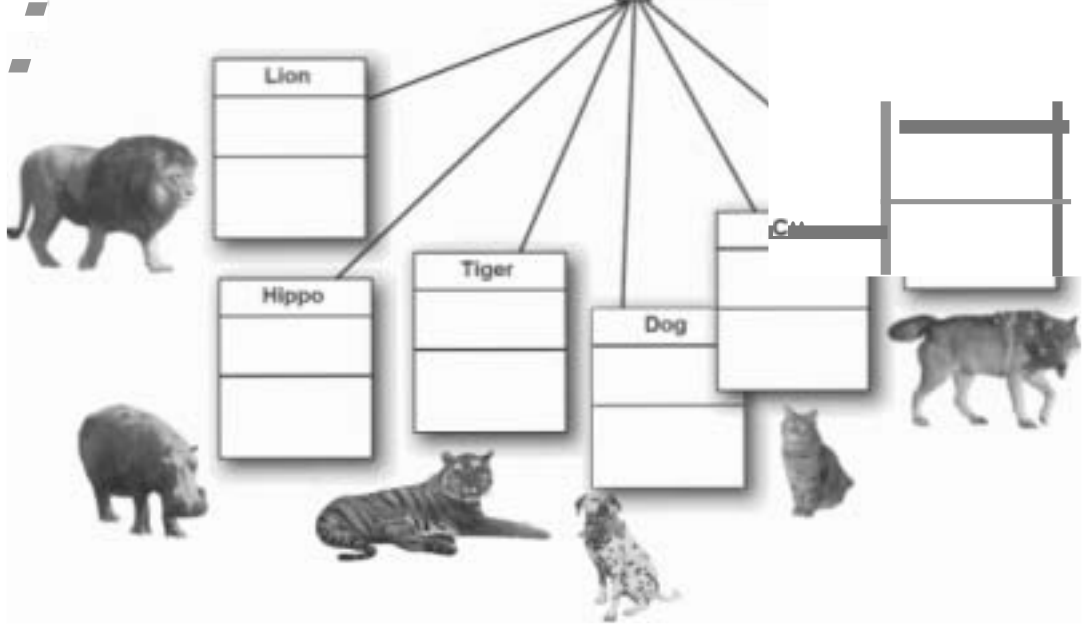
And the Animal class has four methods that the animals can inherit:

- `MakeNoise()`: a method that lets the animal make a sound.
- `Eat()`: behavior for when the animal encounters its preferred food.
- `Sleep()`: a method to make the animal lie down and take a nap.
- `Roam()`: the animals like to wander around their pens in the zoo.

0 Build a base class to give the animals everything they have in common

The fields, properties, and methods in the base class will give all of the animals that inherit from it a common set of state and behavior. They're all animals, so it makes sense to put the base class in.





Choosing a base class is about making choices. You could have decided to use a ZooOccupant class that defines the feed and maintenance costs, or an Attraction class with methods for how the animals entertain the zoo visitors. But we think Animal makes the most sense here. Do you agree?

Different animals make differellt Iloises

I,i,>n, roar. d,ogs bark, and as far as *we* know hipl"" d(in't make anv sound at all. Each "f the classes thar inherit li"m Animal "ill have a MakeNoise() method, but each of those u"'lh,ods will work a dilf"n'nl way "nd " ill have different code. \\'h,'n a subclass changes the behavior "l' '"'"' "f Ih,' methods Ih"l il inh"rih'd, we say th,lI it ov..**rrides** Ih,' method.

Just because a property or a method is in the Animal base class, that doesn't mean every subclass has to use it the same way... or at all!

Think about what you "eed to override

When a subclass changes the bel",,·i"r"l' a ""'lh,„1 il inherited, we "all il **overriding**. Every animal needs t" eat. flLlt a dog might take linl,' bites "f meat, while a hipI'" eats huge mouthfuls of grass. So what would the ,-,,,L, for that behavior I,,,,k like? Both Ihe dog and the hipI'" would ovetTid,' the Eat() method. The hippo's method would have il consume, say, 20 p,,,,,,,,I, of hay each time il was called. The dog's Eat.() method, on the other h""d. would reduce the z,",'s fo"I supply by one 12-ounce can of d,ogfood.

So when you've got a subclass that inherits f..om a base class, it **must** inherit all of the base class's behaviors.., but you can **modify** them in the subclass so they'.e not pe..formed exactly the same way That's what over..iding is all about

Grass is yummy! I could go for a good pile of hay right now



3 Figure out what each animal does that the Animal class does differently—or not at all

\What d"," t'a,'h type "l' a"i",,,1 d" that all the other animals don't? Dogs eat dog food, so the dog's Eat() method willn-ed t<, override the AnLrna I . Ed t () method. Hippos swim so a hippo " ill have a Swim() "",tlt,,,1 lh,lI isn't in Ill,' ,\"i",,,1 class at all,

I beg to differ.

C



Animal
Picture Food Hunger Boundaries Location
MakeNoise() Eat() Sleep() RoamO



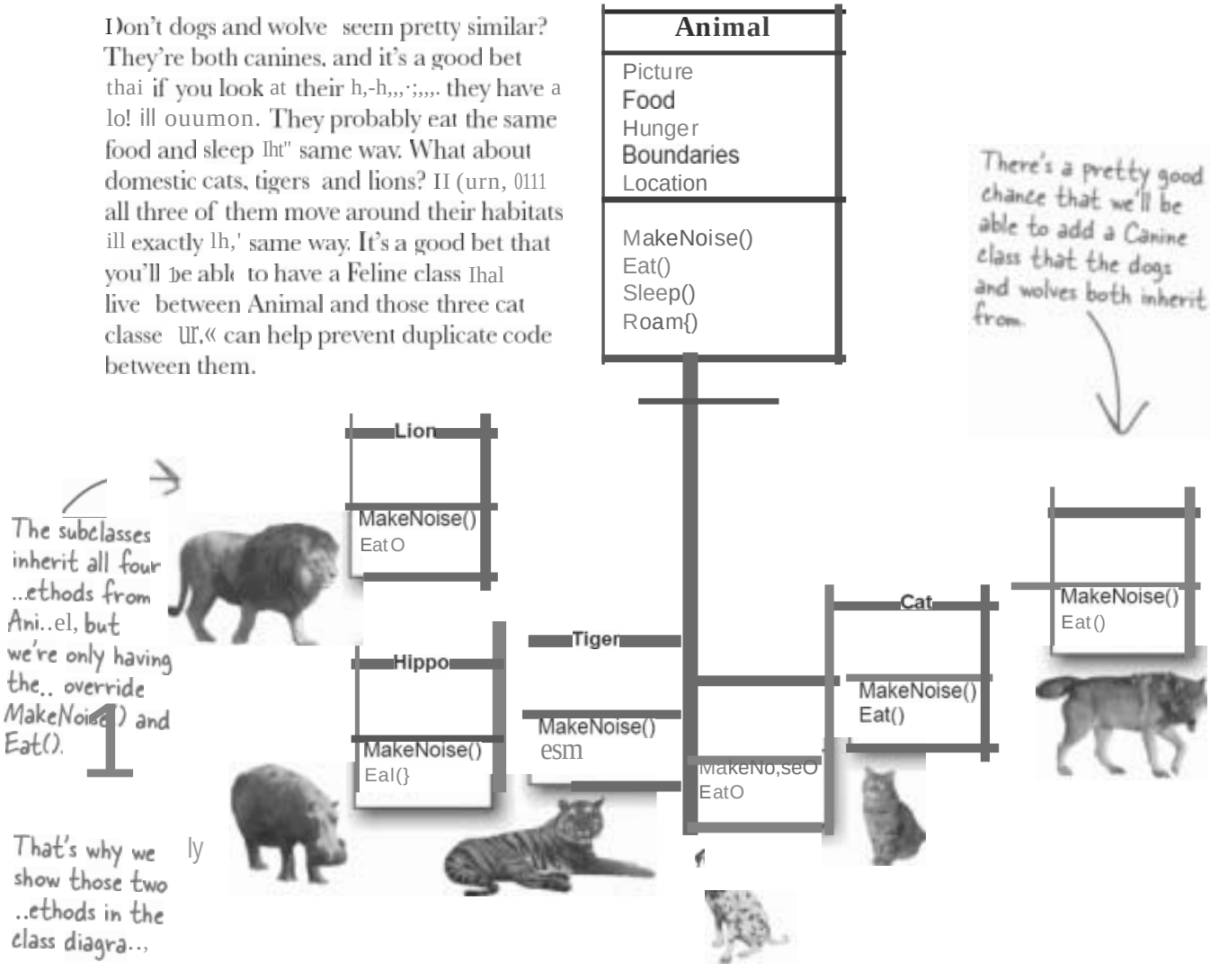
We already know that **some** animals will override the MakeNoise() and Eat() methods, Which animals will override Sleep() or Roam()? Wilt any of them? What about the properties-which animals will override some properties?

rhil1k about **how** to group **the animals**

Aged Vermont cheddar is a kind or cheese, which is a dairy product, whjch is a kind of food. and a good clas: model for food would represent that. Lucky for U'. C# give U, an easy way to d.. it You can create a chain of classes that inherit from each other, start ing with the tt'1""0'1 base class and working <l, wn. So you "o"ld have a Food class, with a subclass called DairyProduct that serve as the base class for Cheese, which has a subclass ,'"ll,-d Cheddar, which is what AgedVermontCheddar inherits ti"lll.

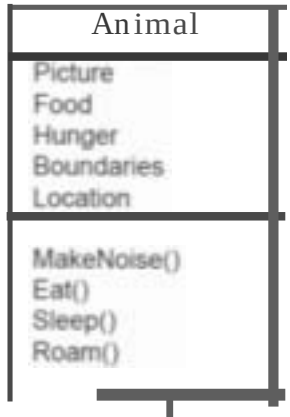
O Look for classes **that have** a lot **in** common

Don't dogs and wolve seem pretty similar? They're both canines, and it's a good bet thai if you look at their h,-h,,,·;,,,. they have a lo! ill ouumon. They probably eat the same food and sleep Iht" same wav. What about domestic cats, tigers and lions? II (urn, 0111 all three of them move around their habitats ill exactly lh,' same way. It's a good bet that you'll ɔe abl< to have a Feline class Ihal live between Animal and those three cat classe ur.« can help prevent duplicate code between them.



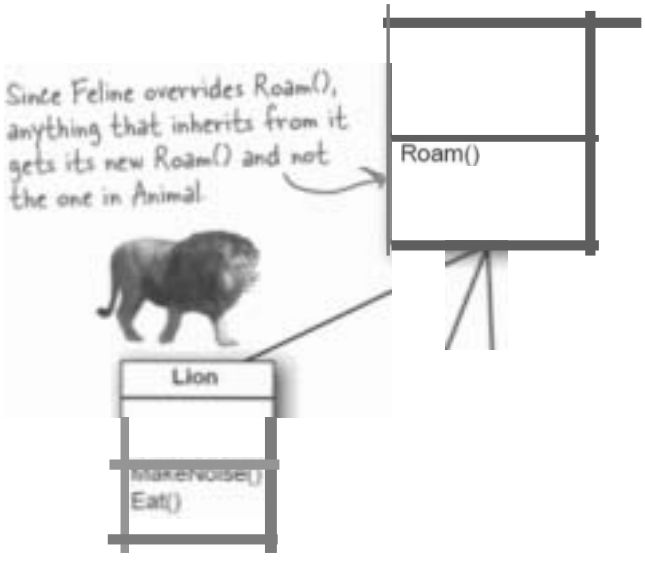
Create the class hierarchy

\\'h..1\ y..u '-...ne your classes so that th 's a base class at the t"" with subclasses below it. and those subcl hav.. th..i•• wn subclasses that inh..rit from them, what you'v.. built is calle..I..I **cl**i.... **hi..rarchy**. This is ..1...it more than just avoiding duplicate code, although tha is certainly .. grea, I nefit ur a sensible hierarchy. Hut when it c '.....n IO it, the biggest 1 lit you'll get is that your code l....-,,, llv easy to understand ..nd maintain. When you're looking ..t th.. zoo , i",,,Lit,,r code, when you see a method or property defined in th.. Felin.. . lass, Ihen y..u *immediately know* ,1",1 you're looking .., "ıething that "ll of Ihe cats share. Your hier..rchv hecomes a m"I' thıu 1 lps you find your way through ,'ur program.

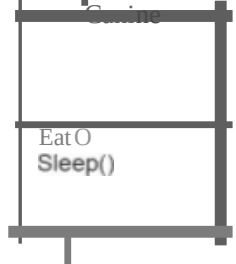


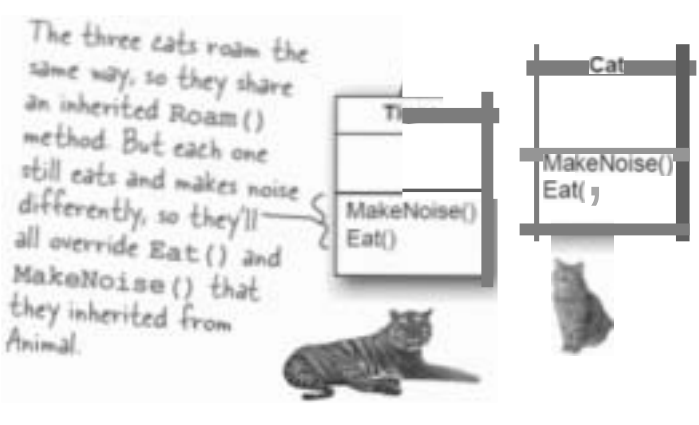
O Finish **your** class **hierarchy**

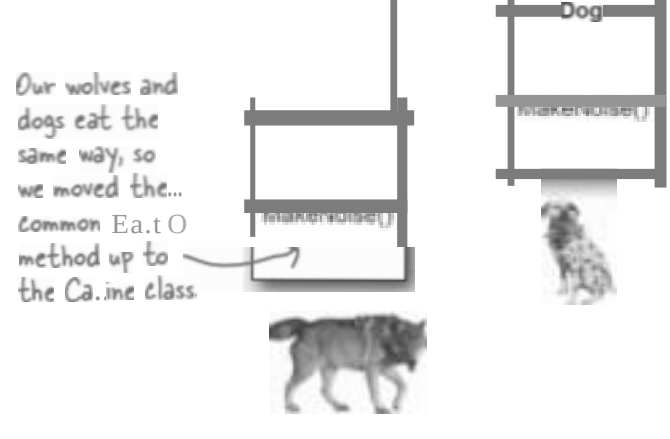
Now that you know how yo,,'ll organize the animals, you ca.. add ,h,' Feline and Canine ,.lasse...











hi-cr-ar-chy, noun.
an arrangement or classification in which groups or things are ranked one above the other. *Thr president of Dvnamco hnd worked his way liP from the mailroom to the top of the corporate hierarchy.*

Every subclass **extends** its base class

You're not limited to the llnethod. that a . nto.'L,ss inherits f'''1ll its base class... hn, you already kn.'" ,h.ll! After .•ll. you'v... been l...ilding your own classes all along. When you add inheritance ,,, a class, what you're doing is taking the class you've already h..ilt and ***extending*** ;1by adding all <lf the fields, properties, and meth..ds in the hase , L, So if you wanted L....kl., Fe t c h () method '" th... dog, th 's perfec thy normal. It ". .m't i" herit or override anything onlv the dog ",in h." that lllet1iod••md il "' '""1 end ••' in Wolf. Canine• .\" im 11il'1' '. "r any nIher class.

```
makes a new Dog object      Dog spot = new Dog();

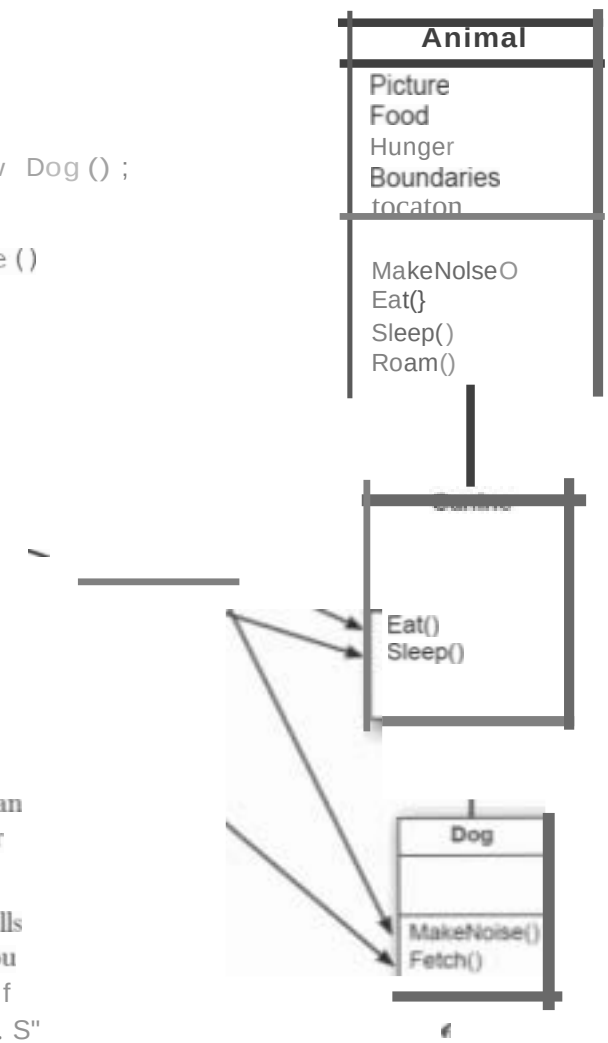
calls the version in Dog    spot.MakeNoise()

calls the version in Animal

calls the version in Canine spot.Eat();

calls the version in Canine spot.Sleep();

calls the version in Dog    spot.Fetch();
```



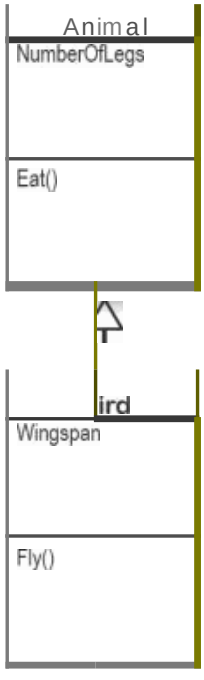
e- always cans the **most specific method**

If you tell your dog object Io roam, ,here·.....lv one m...lh"d Ih.u can be called Ih.. .me in the .\ nun,ll clas.. But what about l...lling your dog I,. make noise? Which `MakeNoise` (I is called?

Well, it's not too hard '" figure it,,out. .\ UKlh"d ill 11K' Dog class tells you h,,,, dogs d" that thing. If |, in the Canine « Lis, . it's telling you how "ll canines do il. .\ml if il" ill Animal, 1h,'" it's a ,k•scription " f that behav-i"r that's so general that it applies I" everv single animal. S" if you ask your ,l,ıg to make a n••isc·. lir,' (:# "ill look inside the dog class I" find the behavior that applies *specifically* I" dogs If Dog didn'l h,,,,;' •ne, it'd then check Callilll·. and after th.•l it'd check •\"illl.•l,

Use a cololl to illnerit **from** a base class

When you're writing a class, you use a colon ('I I" have il inherit from a base class. That makes it a subclass, and gives il **all** of **the fields**, **properti..s,** and **m..thods** ,,1' lh,' class il inherits li""""




```
public class Animal

    public int NumberOfLegs;
    public void Eat() {
        /1 code to make the animal eat
    }
}
```

The Bird class uses a colon to inherit from the Animal class. This means that it inherits all of the fields, properties, and methods from Animal.

```
public class Bird : Animal
{
    public double Wingspan;
    public void Fly{) (
        /1 code to make the bird fly
```

You extend a class by adding a colon to the end of the class declaration, followed by the base class to inherit from

tweety is an instance of Bird, so it's got the Bir ...ethods and fields as usual.

```
public buttonl_Click(object sender, EventArgs e) {
    Bird tweety - new Bird();
    tweety.Wingspan - 7.5;
    tweety.Fly();
    tveety.NumberOfLegs   2;
    tweety.Eat();
```

Since the Bird class inherits from An,mal, every i.. stance of Bird also has the fields and ..ethods defined in the Animal class.

When a subclass **inherits from** a **base class,** all of **the fields, properties,** anJ **methods** in **the base class are** automatically **added** to **the subclass.**

there are no Dumb Questions

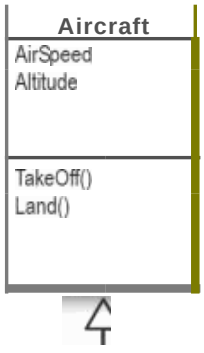
0_: Why does the arrow point up, **from** the lubcla**ss** to the **base class? Wouldn't the diagram look** better **with** the arrow pointing down **instead?**

A: It might look better, but il **wouldn't** be **as** accnate. **When** you **set** up **a** class to **inherit** (rom **another one,** you build that **relationship** ,nto **the subclass—the base class** rema**ins the same.** And that **makes sense when you** think about it **from the** perspecnva of **the base** cress

Its behavior is completely unchanged when you adda dass that inherits from it. The base case isn't even aware of this new class that mhented from it. ns methods, fields, and properties remain entirely intact. But the subclass definitely changes its behavior. Every instance ctee subclass automatically gets all oll1le propemes. fields, and methods from the base ciase. and n all happens just by adding a COCO. That's why you draw the arrow on your diagram so that it's part of the subclass, and points to the base class that it inherits from

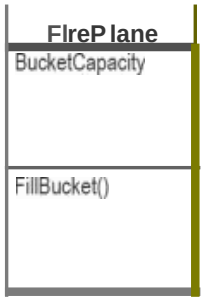
Sharpen your pencil

Take a look at these class models and declarations, and then circle the statements ther won't work.

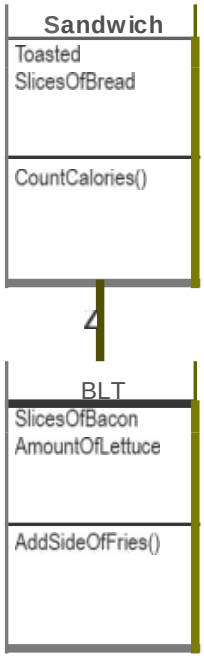



```
public class Aircraft I
    public double AirSpeed;
    public double Altitude;
    public void TakeOff() {     };
    public void Land() { ... };


public class FirePlane : Aircraft
    public double BucketCapacity;
    public void FiIIBucket()        I;
```




```
public void FireFightingMission() I
    FirePlane myFirePlane = new FirePlane();
    new FirePlane.BucketCapacity - 500;
    Aircraft.Altitude = 0;
    myFirePlane.TakeOff();
    myFirePlane.AirSpeed = 192.5;
    myFirePlane.FiIIBucket();
    Aircraft.Land();
```



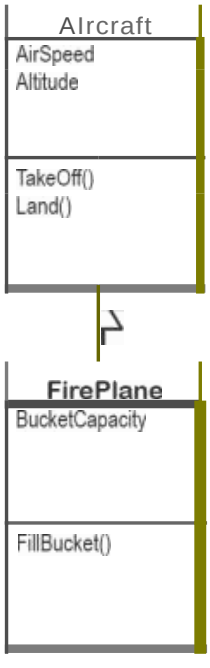

```
public class Sandwich I
    public boolean Toasted;
    public int SlicesOfBread;
    public int CountCalories()    ... I


public class BLT : Sandwich {
    public int SlicesOfBacon;
    public int AmountOfLettuce;
    public int AddSideOfFries() { ... )


public BLT OrderMyBLTO {
    BLT mySandwich - new BLT();
    BLT.Toasted - true;
    Sandwich.SlicesOfBread = 3;
    mySandwich.AddSideOfFries();
    mySandwich.SlicesOfBacon +- 5;
    MessageBox.Show("My sandwich has
       + mySandwich.CountCalories + "calories",);
    return mySandwich;
```

Sharpen your pencil Solution

Take a look at these class models and declarations, and then circle the statements that won't work.



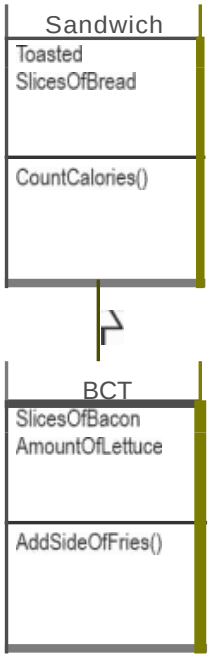

```
public class Aircraft {
    public double AirSpeed;
    public double Altitude;
    public void TakeOff() {     };
    public void Land() { ... };


public class FirePlane   Aircraft
    public double BucketCapacity;
    public void fillBucket()       );


public void fireFightingMission() {
    FirePlane          ane = new FirePlane();
      w FirePl                    ;
    AIr      .Attue
    myFirePlane. a e    ();
    myFirePlane.AirSpeed = 192.5;
    myFuePlane.FillBucket();
    Aircraft.Land(); )
```

That's not how You use the new keyword

These statements all use the class names instead of the name of the instance, myFirePlane.




```
public class Sandwich {
    public boolean Toasted;
    public int SlicesOfBread;
    public int CountCalories()    ... )


public class BLT : Sandwich {
    public int SlicesOfBacon;
    public int AmountOfLettuce;
    public int AddSideOfFries()    ... )

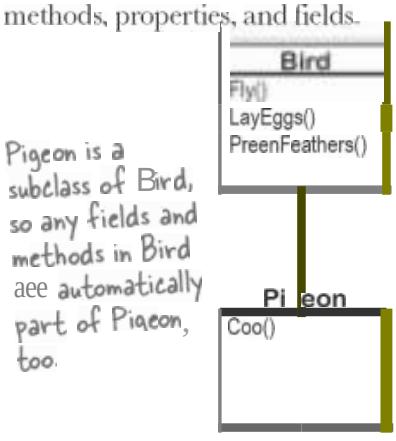

public ELT OrderMyELTO {
    B          h - new BLT();
    BLT.Toasted - true;
    Sandwich.SlicesOfBread - 3;
    my an wicn.     1 e
    m Sandwich.SlicesOfBacon +- 5;
    MessageBox.Sow("My san wic has
        + mySand',o/ich.CountCalories + "calories".);
    re
```

These properties are part of the instance, but the statements are trying to call them incorrectly using the class names

CountCalories is a method, but this statement doesn't include the parentheses () after the call to the method.

We kl10w **that** il1herital1ce adds **the** base class fields. properties. al1d **methods** to **the** subclass...

Inheritance is simple when your subclass needs 10 inherit *all* ,,f the base class methods, properties, and fields.



Pigeon is a subclass of Bird, so any fields and methods in Bird aee automatically part of Piaeon, too.

```
public class Bird {
   public void Fly() {
    // here's the code to make the bird fly

   public void LayEggs() { ... };

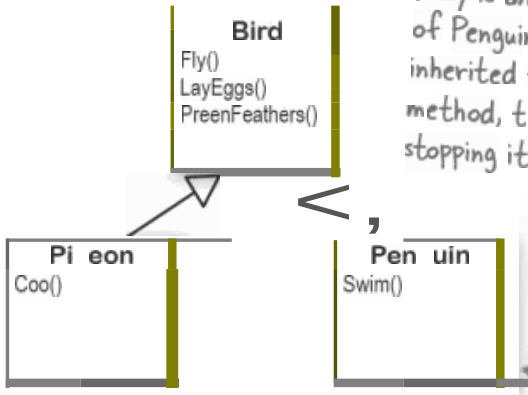
   pUblic void PreenFeathers() { ... };

public class Pigeon : Bird
   public void Coo() { ... }

public class Penguin : Bird
   public void Swim() { ... }
```

... **but som** birds do"'t fly!

" "hal do ¸'ou1 d" if your base class has a method that your subclass needs to ***modify***?



Izzy is an instance of Penguin. Since it inherited the FlyO method, there's nothing stopping it from flying

```
pubiic void BirdSimulator()
    Pigeon Harriet - new Pigeon();
    Penguin Izzy - new Penguin();
    Harriet.Fly();
    Harriet.Coo();
    Izzy.FlyO;
}
```

Both Pigeon and Penguin inherit from Bird, so they both get the Fly(), LayEggs(), and PreenFeathers() methods.

Penguin objects shouldn't be able to fly! But if the Penguin class inherits from Bird, then you'll have penguins flying all over the place. So what do we do?

Pigeons fly, lay eggs, and preen their feathers, so there's no problem with the Pigeon class inheriting from Bird.



If this were your Bird Simulator code, what would you do to keep the penguins from flying?

Asubclass call override **methods** to challge or replace **methods** it illherited

SO",.'I;",." you've got a subclass Iha, you'd like 10 inherit *most* "I' the behaviors h-„„,, the base class, Inn 1101 *all of them.* When you want 10 change the behaviors that a clas: has inherited, vou can **override** the methods.

O Add **the** virtual **keyword** to **the method** in **the base** class

.\ subclass ran only override a method if it's marked with the vir tu **al** keyword. which tells C# to allow Ih,' subclass 10 override methods.

```
public class Bird {
    public virtual void Fly() {
        // code to make the bird fly
    }
}
```

Adding the virtual keyword to the FlyO method tells C# that a subclass is allowed to override it.

2 Add a **method** with **the** same **name** to **the derived** cless

y" „·I1 lleed 10 have exactly Ihe same signature meaning the same "'lUrn value and parameters - and you'll need to use ,he **overrid**.. keyword in the d.·.'l"l'a,ion.

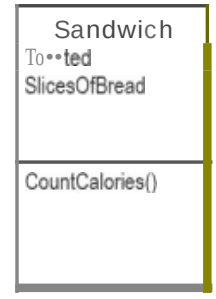
```
public class Penguin Bird {
    public override void Fly() {
        MessageBox.Show("Penguins can't fly!")
    }
}
```

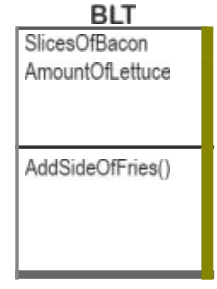
To override the FlyO method, add an identical method to the sub lass and use the override keyword. t:

When you override a method, your new method needs to have exactly the same signature as the method in the base class it's overriding. In this case, that means it needs to be called Fly, return void, and have no parameters.

Use the override keyword 10 **add** a **method** to your subclass **that replaces** one **that** it **inherited. Before** you can **override** a **method,** you **need** 10 **mark** it victual in **the** l-ase **class.**

Any place where you can use a base class, you can use one of its subclasses instead





One of the most useful things you ',m do with inh"';la1l"" is use a subclass ill place "I' the base class it inherit, from. So, if your Recipe {} method takes a Cheese object and you've got an AgedVermontChectdar class that inherits from Cheese. then you can pass an instance ,,I' AgedVermontCheddar to th,> Recipe (} method. Recipe () only ha' access 101h(" fields, properties and m"lh"d, that are part or the Cheese class, though— it doesn't have access to anything specific to AgedVermontCheddar.

1 Let's say we have a method to analyze sandwich objects:

```
public void SandwichAnalyzer lSandwich specimen)
    int calories - specimen.CountCalories();
    UpdateDiet Plan{calories);
    PerformBreadCalculatiolls(specimen.SlicesOfBread, specimen.Toasted);
```

O You could pass a sandwich to ,I,,· method Inll you ,"""Id also pass a m:r. Since a BLT is a *kind* or sandwich, we set i, up so that i, inherits from ,h,' Sandwich class.

```
public buttonl_Click(object sender, EventArgs e)
    BLT myBLT - new BLT();
    SandwichAnalyzer(myBLT);
```

We'll talk about this more in the next chapter!

O You can always ""ve ***down*** the class diagram —a reference variable can always be se equal to all instance of one of its subclasses. JI', you ,.'' ,,'t move *up* ,h,· class diagram.

```
public button2 Click(object sender, EventArgs e) I
    Sand'wich mySand-wich _ nev Sandwich();
    BLT myBLT _ new BLT();
    Sandwich someRandomSandwich   myBLT;
    BLT anotherBLT = mySandwich;     11<--- THIS WON'T COMPILE!!!
```

You can assign myBLT to any Sandwich variable because a BLT is a kind of sandwich.

But you can't assign mySandwich to a BLT variable, because not every sandwich is a BLTI That's why this last line will cause an error.



Mixed Messages



A short C# program is listed below. One block of the program is missing! Your challenge is to match the candidate block of code (on the left), with the output—what's in the messagebox that the program pops up—that you'd see if the block were inserted. Not all the lines of output will be used, and some of the lines of output might be used more than once. Draw lines connecting the candidate blocks of code with their matching output.

Instructions:

1. fill in the four blanks in the code.
2. Match the code candidates to the output.

```
public class A {
  public int ivar = 7;
  public virtual string m1() {
    return "A's m1, ";
  }
  public string m2() {
    return "A's m2, ";
  }
  public _____ string m3() {
    return "A's m3, ";
  }
}

public class B : A {
  public override string m1() {
    return "B's m1, ";
  }
}
```

```
public class C : B {
  public new string m3() {
    return "C's m3, " + ivar + 61;
  }
}

public class Mixed5 {
  public static void Main(string[] args) {
    A a = new A();
    B b = new B();
    C c = new C();
    A a2 = new C();
    string q = "";

    ____________
    ____________
    ____________

    System.Windows.Forms.MessageBox.Show(q);
  }
}
```

Here's the entry point for the program—it doesn't show a form, it just pops up a messagebox.

candidate code goes here (three lines)

code candidate:

```
q += b.m1();
q += c.m2();
q += a.m3();

q += c.m1();
q += c.m2();
q += c.m3();

q += a.m1();
q += b.m2();
q += c.m3();

q += a2.m1();
q += a2.m2();
q += a2.m3();
```

output:

A's m1, A's m2, C's m3,

B's m1, A's m2, A's m3,

A's m1, B's m2, A's m3,

B's m1, A's m2, C's m3,

B's m1, C's m2, A's m3,

B's m1, A's m2, C's m3,

A's m1, A's m2, C's m3,

(Don't just type this into the IDE-you'll learn a lot more if you figure this out on paper!)



Pool Puzzle

Your *job* is to take code snippets from the pool and place them into *the* blank lines in th» code. You may use the same snippet more than once. and you might not need to use all the snippets. Your ***goal*** is to make a ,et of classes that will compile and run together as a program. Don't be fooled-thi. one', harder than it looks.

```
public class Rowboat

   public ..... rowTheBoat()
      return "stroke natasha";
```

```
public class

   private int ...

            void                          ) )
      length = len;

   public int getLength()


   public                        move 11
      return
```

```
public class TestBoats {

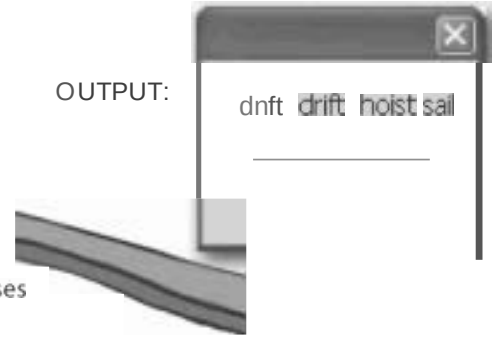
      ..... Main() {
   ..... xyz = =
   ..... bl _ new Boat();
    Sailboat b2                          11;
    Rowboat          _ new Rowboat I);
    b2.setLength(32);
    xyz - bl.          ();
    KyZ += b3. ..... 11;
    xyz t.          .move();
    System.Windows.Forms.MessageBox.Show(xyz);
```

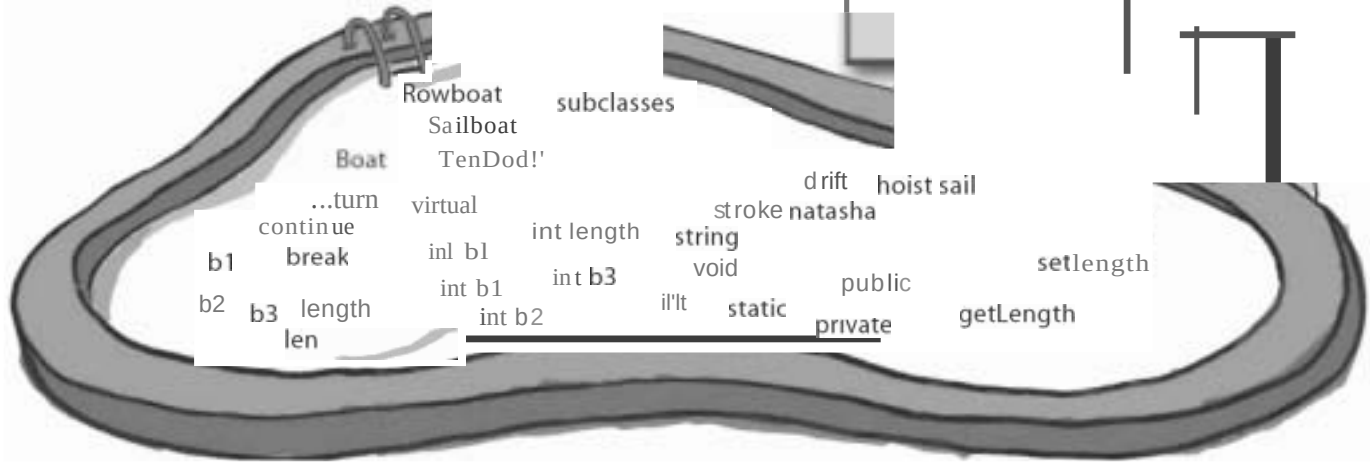
Hint: This is the entry point for the program.

```
public class

   public                          ()
      return
```

OUTPUT:







Mixed Messages

```
public class A {
    public virtual string m1()

    public virtual string m3()
```

```
public class B : A
    public override string m1() {

public class C : B
    public override string m3()
```

6; → 56
5; → 11
5; → 65

```
q += a.m1();
q += b.m2();
q += c.m3();
```

```
q += a2.m1());}
q += a2.m2());
q += a2.m3());
```

B's m1, C's m2, A's m3,

A's m1, B's m2, A's m3,

B's m1, A's m2, C's m3, 6

A's m1, A's m2, C's m3, 13

Pool puzzle Solution



```
public class Rowboat        Boat      {
    public ....string....rowTheBoat1)
        return "stroke natasha";
```

```
public class      Boat      |
    private int ....length....;
    public ....void ....setLength.... ....int len....
        length = len;

    public int getLength()
        return length ....;

    public ....virtual string.... move()
        return "....drift....
```

```
public class TestBoats {
    public static void Main||{
        ..string... xyz = "";
        Boat b1 = new Boat();
        Sailboat b2 = new Sailboat ||;
        Rowboat      = new Rowboat();
        b2.setLength(32);
        xyz = b1.  move ||;
        xyz += b3.  move ||;
        xyz += b2.move();
        System.Windows.Forms MessageBox.Show(xyz);
```

```
pubilc class ....Sailboat....  Boat |
    public ....override string move.... ||
        return  hoist sail
```

there are no Dumb Questions

Q: About the entry point that you pointl d out in the Pool Puzzle—do. . !hie m. .n I can have I program that doee n"1 have I Form1 form?

A: Yes When you create • new Windows Application project, the IDE creates all the files for that project for you, including Program cs (which contains a static class with an entry point) and Form1 cs (which contains an empty form called Form1)

Try thi. : instead of creating a new Windows Application project, create an empty project by selecting "Empty Project" instead of "Windows Application" when you create • new project in the IDE Then add a class file to it ., the Solution Explorer and type in everything in the Pool Puzzle solution Since your program uses • messagebox, you need ID add a **reference** by right-clicking on "References" in the Solution Explorer, selecting "Add Reference", and choosing System Windows Forms from the NET tab (That's another thing the (IE does for you automatically when you create a Windows Application.) Finally, select "Properties" from the Project menu and choose the "Windows Application" project type

Now run it... you'll see the results! Congratulations, you just created • C# program from scratch

Flip back to the beginning of Chapter 1 if you need a refresher on Main() and the entry point!

Q: C.n I inherit from the class that contains the entry point?

A: Yes. The entry point ***must*** be a static method, but that method ***doesn't have to be*** in a static class. (Remember, the `stati c` keyword means that the clas can't be instantiated. but that its methods are available as soon as the program starts SO in the Pool Puzzle program, you can call TestBoats Main() from any other method without declaring a reference variable or instantiating an object using a `new` statement.)

Q: 1'1' 1 don't get why they're called "virtual" methods—they seem real 10 ma'

A: The name "virtual" has ID do with how NET handles the virtual methods behind the scenes • uses something called a virtual method table lor vtable). That's a table that NET uses to keep track of which methods are inherited and which ones have been overridden Don't worry you don1 need ID know how it works to use virtual methods!

Q: What did you maan by only being able to move up the class diagram but not being able to move down?

A: When you've got a diagram with one class that's above another one, the class that's higher up is more *abstract* than the one that's lower down More specific or concrete classes (like Shirt or Car) inherit from mora abstract ones (like Clothing or Vehicle) When you think about it that way it's easy to see how if a. you need is any vehicle, a car or van or motorcycle will do But if you need a car, a ll.,,torcycle won't be useful to you

Inheritance works exactly the same way • you have • method with `Veh 1c Le` as a parameter, and if the `Motorcycle` class inherits from the `Vehic Le` class, then you can pass an instance of `Motorcycle` to the method But if the method takes `Motorcycle` as a parameter, you can't pass any `Veh tc1e` object, because it may be a `Van` instance. Otherwise C# wouldn't know what 10 do when the method tries to access the `Handlebars` property!

You can always pass an instance of a subclass 10 any mellrod whose parameters expect a class tl.al il inherits f.·om.

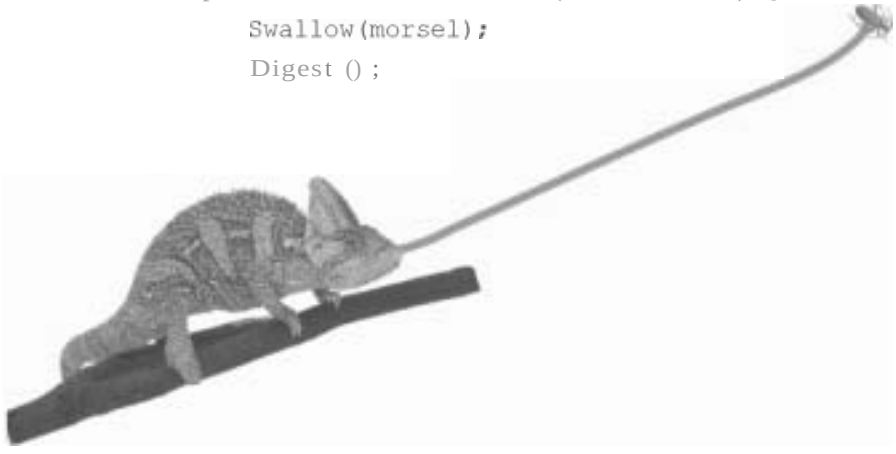
A subclass can access its base class using the base keyword

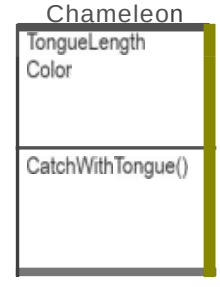
Even when you override a method or property in your base class, sometimes you'll still want to access it. Luckily, we can use `base`, which lets us access any method in the base class.



- All animals eat, so the `Animal` class has an `Eat()` method that takes a `Food` object as its parameter.

```
public class Animal
    public virtual void Eat(Food morsel) {
        Swallow(morsel);
        Digest();
```





- Chameleons eat by catching food with their tongues. So the `Chameleon` class inherits from `Animal` but overrides `Eat()`.

```
public class Chameleon : Animal {
    public override void Edt (Food morsel)
        CatchWithTongue(morsel);
        Swallow(morsel);
        Digest();
```

The cha..eleon needs to swallow and digest the food, just like any other animal. Do we really need to duplicate this code, though?

- Instead of duplicating the code, we can use the **`base`** keyword to call the method that was overridden. Now we have access to both the old and the new version of `Eat()`.

```
public class Chameleon : Animal {
    public override void Eat(Food morsel) {
        CatchWithTongue(morsel);
        base.Eat(morsel);
```

This line calls the Eat() method in the base class that Cha..eleon inherited from

When a base class has a constructor, your subclass needs one too

If your class has a constructor, then any class that inherits [rom it **must** call th at **constructor**. The subclass's constructor can have different parameters from the base class constructor.

```
public class Subclass : BaseClass {
    public Subclass(parameter list)
        : base(the base class's parameter list) {
        // first the base class constructor is executed
        // then any statements here get executed
    }
}
```

Add this extra line to the end of your subclass's constructor declaration to tell C# that it needs to call the **base class's constructor** every time the subclass is instantiated.

Here's the constructor for the subclass.

Ihe **base** class eonstrueter is executed before **the** subclass ec asrreeter

But tI",,'t take "UI' word for it—see t, ;t yourself!

Do this!

O **Create** a **base** class with a ecastructcr- that pops up **a messagebox**
Ihen add a button to a form that instantiates Ihi, ***base class*** and ,h"w, a messagebox:

```
pUblic class MyBaseClass {
    public MyBaseClass(string baseClassNeedsThis) {
        MessageBox.Show("This is the base class; " + baseClassNeedsThis);
```

This i• a pa.a..ete.. that the base class constructor need•.

Keep an eye out for thi slightly cryptic error. It ..eans that your subclass didn't inherit the constructor properly

O **Try** adding a **subclass,** but don't call t he **constructor**
Then add a hUll"" to a form that in,lam;at,·, this *s ubl"lass* and shows a messagebox:

```
public class MySubclass : MyBaseClass{
    public MySubclass(string baseClassNeedsThis, int anotherValue)
        MessageBox.Show("This is the subclass: " + baseClassNeedsThis
            + " and " • anothervalue);
```

Select Build > Build Solution in the IDE and t'ou'" get an error from this) code

1 No overload for method 'MyBaseClass' takes '0' arguments

O **Fix** the error by making the constructor call **the** one from the base class
Ihen instantiate the subclass and ***se what order*** the two message boxes IKp up!

```
public class MySllbclass : MyBaseClass{
    public MySubclass(string baseClassNeedsThis, int enccnervaiuej
        : base(baseClassNeedsThis)

    // the rest of the subclass is the same
```

Thi. is how we send the base class the paramete.. its constructor needs

Add this line to tell C# to call the constructor in the base cl It has a para..eter list that shows what gets p sed to the base class const"'ctor Then the error will go awa't and you can make a button to see the two ..essage boxes pop up!

Now you're ready to ftnlsh the Job for Kathleen!

When you last lefi Kathleen, you'd finished adding birthday parties III her program. She 11,;eds you In charge an extra $100 for partie over 12. 11 seemed like vou were going 10 have U, write the same exact code twice, once for ea..h class. Now Ihal you know how In use inheritance, you can have them inherit from the same base class that ..ontains all or their shared code, sc yc'11 only have 10 "ril" il once.

DinnerParty	BirthdayParty
NumberOlPeople CostOfDecorations CostOfBeveragesPerPerson HeallhyOpnon	NumberOfPeople ccetotecconcoe c ekesze CakeWntlng
CalculateCostOfDecorations() CalculateCostO SetHeallhyOptlOnO	CalculateCostOfDecorations() CalrulateCostO

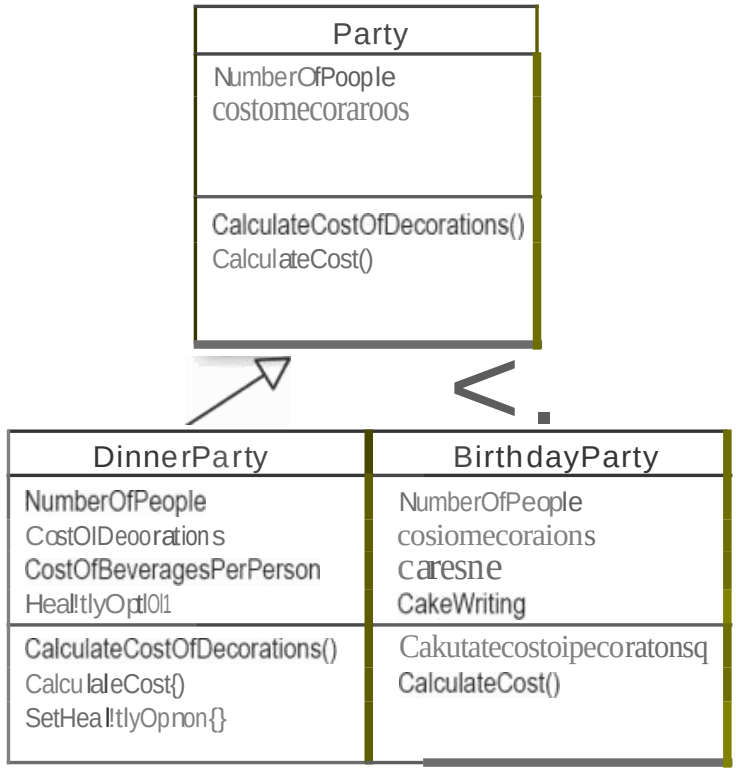


If we play our cards right, we should be able to change the two **classes** without making any changes to the forml

O

Let's create the new class model

" 'e'll 'Iill have the same DinnerParty and BirthdayParty classes, but now they'll inlw1il from a single Party class. \,'e need them I" have ,-xa,tly the same llwlh"(l,, properties and fields, ,,) we d,,n't have u>make any change 10 the f('I'm. But some or those methods, properties, and fields willl,,·mo,ed inlo uie Party base class, and we mav have 10 override a few or Ih'-'ll_



Build the Party base class

Create the Party class, and make sure it's public. You'll need to look really closely at the properties and methods in the class diagram, and figure out what you need to move out of DinnerParty and BirthdayParty and into Party.

- Move the NumberOfPeople and CostOfDecorations properties and fields so that they're compatible with both DinnerParty and BirthdayParty.
- Do the same for the CalculateCostOfDecorations() and CalculateCost() methods. If those methods need any private fields, you'll need to change them too. (Remember, subclasses can only inherit **public** fields—once you change a private field to Party, the DinnerParty and BirthdayParty classes will have access to it.)
- You'll also need a constructor. Take a close look at the BirthdayParty and DinnerParty constructors—anything they have in common should be moved to it.
- Now add **the $100 bonus** for parties over 12 people. After all, that's why we're doing this! It's common to both birthday and dinner parties, so it belongs in Party.

3 Make DinnerParty inherit from Party

Now the Party does all the things a DinnerParty does, you can eliminate the overlap and only keep the part of DinnerParty that's unique to dinner parties.

- Make sure the constructor is working properly. Does it do anything the Party constructor doesn't? If so, keep that and then leave everything else to the base class constructor.
- Any logic that has to do with setting the healthy option, will, too, stay in DinnerParty.
- You'll need to override at least one method because it does something specific to dinner parties.

Make BirthdayParty inherit from Party

Do the same thing for BirthdayParty—leave anything that's specific to birthdays to the base class, and you'll keep the birthday-specific functionality in BirthdayParty.

- What does the BirthdayParty constructor need to do that's not part of Party?
- You'll need to deal with the cost of the cake in the BirthdayParty. That's a method and a property, so you'll need to override them.
- Yes, you can override a property! It's just like overriding a method. When you set the `NumberOfPeople` or `base.NumberOfPeople`, the property's set accessor in the base class. You'll need to use the `base` keyword to both get and set the value



Check it out- you changed the DinnerParty and BirthdayParty classes so that they inherited from the same base class, Party. Then you were able to make the change to the cost calculation 10 add the $1 00 fee, and you mco'thave 10 change the form at all. Neat!

```
public class Party
(
    const int CostOfFoodPerPerson = 25;
    private bool fancyDecorations;
    public decimal CostOfDecorations • 0;

    public Party(int numberOfPeople, bool fancyDecorations) {
        this.fancyDecorations = fancyDecorations;
        this.NumberOfPeople = numberOfPeople;
```

This code was moved straight out of the DinnerParty and BirthdayParty classes and into Party.

The Party constructor does everything that was previously in both the DinnerParty and BirthdayParty constructors.

```
    private int numberOfPeople;
    public virtual int NurnberOtPeople (
        get I return nurnberOfPeople; )
        set I
            nurnberOfPeople = value;
            CalculateCostOfDecorations(fancyDecorations);
```

NumberOfPeople eeeds to be virtual because Bi..thdayPa..ty needs to over..ide it (so that a change to the numbe.. of people calculates a new eske size).

```
    public void CalculateCostOfDecorationstbool fancy) {
        fancyDecorations - fancy;
        if (fancy)
            CostOfDecorations    (NumberOfPeople • I 5. 00M) + SaM;
        else
            CostOfDecorations    (NurnberOfPeople * 7.50M) + 30M;
```

The decoration calculation is identical in both birthday and dinner parties, so it makes sense to move it to Party. That way none of the code is duplicated in ..ultiple classes.

```
    public virtual decimal CalculateCost() {
        // Each person costs $25 for food plus cost of beverages
        decimal TotalCost = CostOfDecorations + (CostOfFoodPerPerson * NurnberOfPeople);
        if (NumberOfPeople > 12)

            TotalCost +- 100;

        return TotalCost;
```

The cost calculation needs to be a virtual method because the birtday party overrides it (and also extends it by calling the base class method).

```
publlc class BirthdayParty : Party {
    pUblic int CakeSize;

    public BirthdayPartylint numberOfPeople, bool fancyDecorations, string cakeWritingl
        : base(numberOfPeople, fancyDecorationsl {
        CalculatecakeSize();
        this.CakeWriting - cakeWriting;
        CalculateCostOfDecorations(fancyDecorationsl;


    private void CalculateCakeSize()
        if (NumberOfPeople <- 4)
            cexesree   8;
        else
            CakeSize • 16;


    private string cakeWriting - "";
    public string CakeWriting {
        get { return this.cakeWriting:
        set {
            int maxLength;
            if (CakeSize -  81
                maxLength   16;
            else
                ...xLength   40;
            if (value.Length > maxLength)
                MessageBox.Show("Too many letters for a " + CakeSize • " inch cake-I;
                if (maxLength > this.cakeWriting.Length)
                    ...xLength • this.cakeWriting.Lengt h:
                this.cakeWriting   cakeWriting.SubstringCO, MaxLenqth);
              else
                this.cakeWriting   value:


    puejrc override decimal CalculateCost() {
        decimal CakeCost;
        if (CakeSize •• 81
            ceseccee   40H. CakeWriting.Length   .25M;
        else
            CakeCost   75M + cakeWriting.Length   .25M;
        return base.CalculateCost() + CakeCost;


    public override int NumberOfPeople {
        get { return base.NumberOfPeople;
        set  {
            base.NumberOfPeople • value;
            CalculateCakeSize();
            this.CakeWriting - cakeWriting;
```

The constructor relies on the base c to do most of the work. Then it call Calc·lateCakeSize[1) •J"'t'ike the old BirthdayParty constructor d,d

The CalculateCakeSize() method is specific to birthday parties, so it stays in the BirthdayParty class

The CakeWriting property stays intact in the BirthdayParty class too

CalculateCost() also needs to be overridden, because it needs to first calculate the cost of the cake, and then add it to the cost that's calculated in the Party class's CalculateCost() method

The NumberOfPeople property has to override the one in Party because the set access··· needs to recalculate the cake size. The set acce..... needs to call base NumberOfPeople so that the set accessor in Party also gets executed

Continuls on page 238.



Exercise Solution continued from p.237

Here's the last class in Kathleen's solution. (There's no change to the form code.)

```
public class DinnerParty : Party
{
    public decimal CostOfBeveragesPerPerson;

    public DinnerParty(int numberOfPeople, bool healthyOption,
                       bool fancyDecorations)
        base(numberOfPeople, fancyDecorations) {
        SetHealthyOption(healthyOption);
        CalculateCostOfDecorations(fancyDecorations);
    }

    public void SetHealthyOption(bool healthyOption) {
        if (healthyOption)
            CostOfBeveragesPerPersDn - 5.00M;
        else
            CostOfBeveragesPerPersDn = 20.0DM;
    }

    public decimal CalculateCost(bool healthyOption) {
        decimal totalCDst    base.CalculateCost()
                           + (CostOfBeveragesPerPerson • NumberOt People);

        if (healthyOption)
            return tDtalCost    .95M;
        else
            return tDtalCost:
    }
```

This public field is only used in dinner parties, not birthday parties, so it stays in the class.

To do what the old DinnerParty class did, the new constructor calls the Party constructor and then calls SetHealthyOption().

The SetHealthyOption() ...ethod stays exactly the same.

Dinne..rarty has to over..ide CalculateCost()—it uses the base cost from Pa..ty, and then adds the cost of the beve..ages and adds in the healthy option discount.

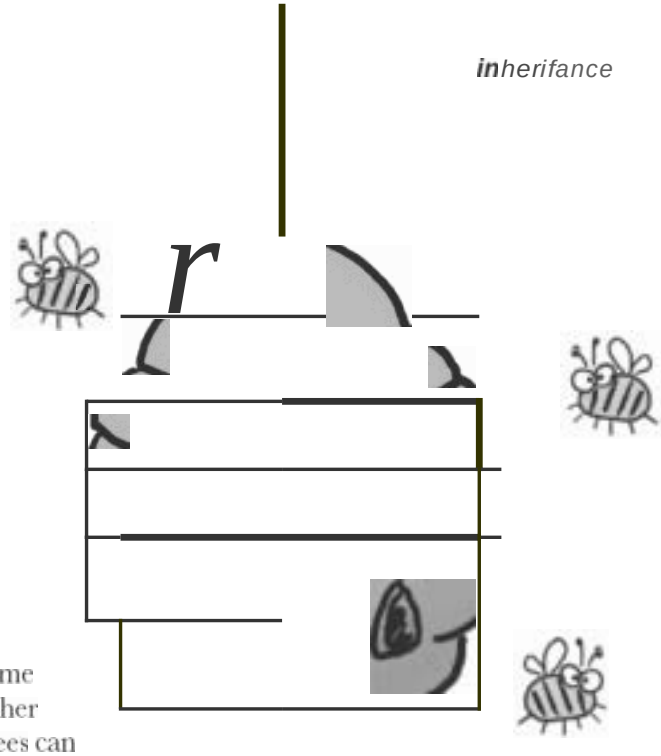


The program's perfect. It's so much easier to run my business now—thanks so much!

Build a beehive llIal1agelllel1t systelll

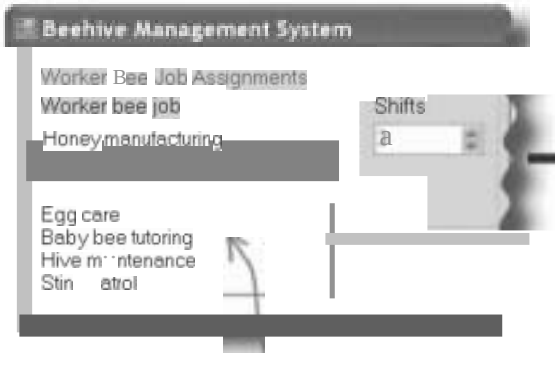
.\ '1'1<-"11 bee needs your help! H'-I' hive is nm or control, and ,ht" needs a program to help manage it. She's go a beehive lilll (1f \c,orkers and a wh,,le lunch "f jtobs thar need " . be tit",,· around the hive·, BUl somehow she's 10,1 control tlr "hid, bee· i, doing what, "",1 whether "r IH'1 she's got the beepower 10do 1l><>jobs that need to be don",

It's up ,"\"'111◇ I,uild a beehive management system "' 1ielp her keep track ..r her workers Here's how ;,'11 "ork:

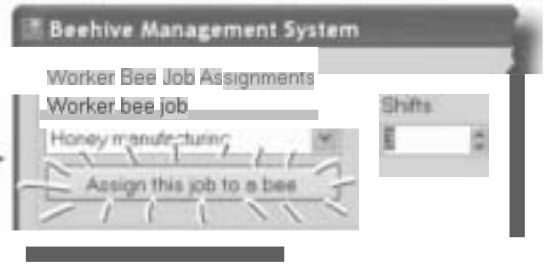


o The queen assigns jobs to her workers

There are six p<,,,ihl,' jobs Ihat the workers can d". Some know how to collect nectar and manufacture honey, other can maintain the hive and pal ,.,] for enemi s. A few bees can d" every job in II,,· hive. S" your program ill needII" give her a way I" assign a job 10 any h.·e that's available 10 do it.

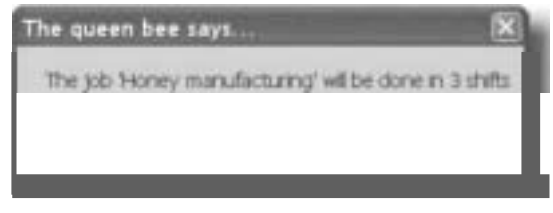


Thi dropdown list shows all ..x jobs that the workers can do. The queen knows what jobs need to be done, and she doesn't really care which bee does each job So she just select hich job has to be done—the progra....ill figure t if there's a worker available to do it and assign the job to hi..



The bees work shifts, and ...ost jobs requi.e more than one shift. So the queen enters the number of shifts the job will take, and clicks the "Assign this job" button.

If there's a bee available to do the job, the program assigns the job to the bee and lets the queen know it's taken care of.



o When the jobs are all assigned, it's time to work

Once the queen's done assinging the work, she'll tell the bees 10 work the next shift by clicking the "Work the next shift" button. The program then generates a shift report that tells her which bees worked Ihal shift, what jobs they did, allli how many more shifts Ihey'll bc working each joh.



First you'll build the basic **system**

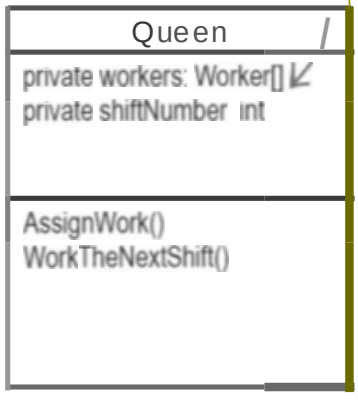




This project is divided into two parts. The first part, a bit of a review, where you'll create the basic system to manage the hive. It's got two classes: Queen and Worker. You'll build the form for the system, and hook it up to the two classes. And you'll make sure the classes are encapsulated so they're easy to change when you move to the second part.



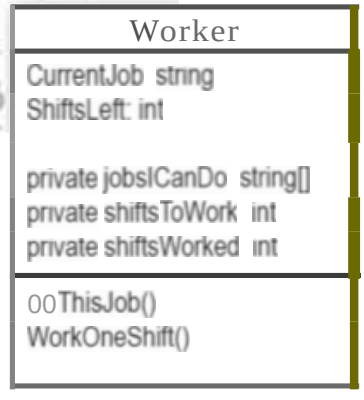
Sometimes class diagrams list private fields and types



The hive program has one Queen object that manages all the work being done.

- The Queen uses an array for Worker objects to track all of the worker bees and who they're assigned to. It's stored in a private Worker[] field called **workers**.
- The form calls the AssignWork() method, passing a string for the job that needs to be performed. It will return an int for the number of shifts. It will return -1 if it can't find a worker to assign the job to, or false if it couldn't find a worker to do that job.
- The form's "Work the next shift" button calls WorkTheNextShift(), which tells the workers to work and returns a shift report to display. It tells a Worker object to work one shift then checks that worker's status so it can add a line to the shift report.

CurrentJob and ShiftsLeft are read-only properties



The Worker class uses an array of Worker objects to keep track of all of the workers and what jobs they're doing.

- CurrentJob is a read-only property that tells the Queen object what job the worker's doing ("String patrol", "Hive maintenance", etc.) If the worker isn't doing any job, it'll return an empty string.
- The Queen object's AssignWork() method will assign a job to a worker using its DoThisJob() method. If that worker is already doing the job, and if that's a job that he knows how to do, then he'll accept the assignment. And the DoThisJob() method returns true. Otherwise, it returns false.
- When the WorkOneShift() method is called, the worker works a shift. He keeps track of how many shifts are left in the current job. If the job is done, then he sets his current job to an empty string so that he can take a next assignment.

String.IsNullOrEmpty()

Since each bee stores its current job as a string, the way the worker knows whether or not he's currently doing a job is to check if his CurrentJob property—it'll be equal to an empty string if he's waiting for his next job. C# gives you an easy way to do that: String.IsNullOrEmpty(CurrentJob) will return true if the CurrentJob string is either empty or null, false otherwise.

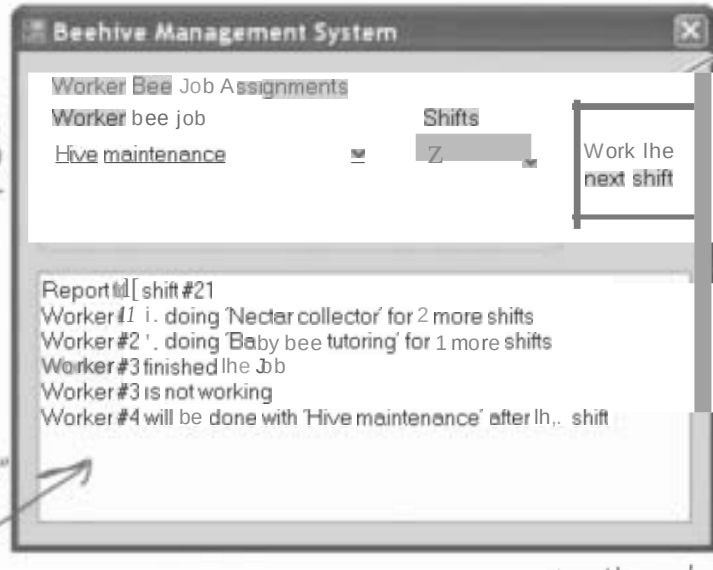


A queen bee needs your help! Use what you've learned about classes and objects to bUild a beehive management system to help her track her worker bees

1 Build the form

The form is pretty simple—all "f the intelligence i, ill the Queen and Worker classes. The f'rIll has a privat(Queen field, and two hUllo"' call its AssignWork() and WorkTheNextShift() methods. You'll need 10 add a ComboBox ,"'"U'ol for lhe h,-,' jobs !flip bac'k to the pre,-i"", page 10 see its list il"111'), a NumericUpDown control, two 1"Ill"n" and a multiluu- textbox for the shift report. Yo,,,'U also need the form's constructor— it's below the screenshot.

This is a ComboBox control named workerBeeJob. Use its Items property to set the list, and set its DropDownStyle property to "DropDownList" so the user is only allowed to choose items from the list. The Shifts box is a NumericUpDown control called shifts.



The nextShift button calls the queen's

Look closely at this shift report, which the

what each worker is doing. Use the escape sequences "\r\n" to add a line break in the middle of a string

Name this text box "report" and set its MultiLine property to true.

```
public Form!()
    InitializeComponent();
    Worker[] workers = new Worker[41;
    workers[O]   new Worker(new string!] { "Neetar colleetor", "Honey manufaeturing" });
    workers!1]   new Worker(new strLnqll { "Egg care", "Baby bee tutoring" ))1
    workers[2]   new Worker(new string!] { "Hive maintenance", "Sting patrol" I);
    workers[3]   new Worker(new string[) { "Nectar collector", "Honey manufacturing",
                 "Egg eare", "Baby bee tutoring", "Hive maintenance", "Sting patrol" II;
    queen = new Queen(workers);
```

Each Worker object's constructor takes one parameter, an array of strings that tell it what jobs it knows how to do

Your form will need a Queen field called queen. You'll pass that array of Worker object references to the Queen object's constructor

O Build the Worker and Queen classes

You've got almost everything you need U'know about the Worker and Queen classes. There are just a couple mor(details. Queen.AssignWork() loops through Ih,-the Queen object's `worker` array and attempts 1" assign the job tu each \\',rker using its 1)()'ThisJ(,hO nu-t hod _The \\'(u'ks-r object checks its `jobsICanDo` string array 10 see if il can do the job. [I' il can, it sets its I'ri,-" t,· shiftsToWork field 1<' the j(,I, dllrali,"', its CurrentJob to the j,)b. and its shiftNIlilber I" zero. When i1 works a shift, it decreases shiftNumber by one. The read-only ShiftsLeft property rt-lurn, `shiftsToWork - shiftsWorked` 11,,-queen use il I" see how many shifts ;u"- left Oil the joh.




```
puoJic cJaee Worker {
    public Worker(string[] jobsICanDo)
        this.jobsICanDo = jobsICanDo;
```

The constructor just sets the JobsICanDo property, which is a string array. It's private because we want the queen to ask the worker to do a job, rather than make her check whether he knows how to do it.

ShiftsLeft is a read-only property that calculates how many shifts are left on the current job

```
pUblic int ShiftsLeft
    get {
        return shiftsToWork - shiftsWorked;
```

CurrentJob is a read-only property that tells the queen which job needs to be done.

```
private string currentJob =
public string CurrentJob {
    get {
        return currentJob;
    }
```

```
private string[] jobsICanDo;
private int shiftsToWork;
private int shiftsWorked;
```

The queen uses the worker's DoThisJob() method to assign work to her—she checks his JobsICanDo property to see if he knows how to do the job.

```
public bool DoThisJob(string job, int numberOfShifts)
    if (!String.IsNullOrEmpty(currentJob))
        return false;
    for (int i = 0; i < jobsICanDo.Length; i++)
        if (jobsICanDo[i] == job)
            currentJob = job;
            this.shiftsToWork = numberOfShifts;
            shiftsWorked = 0;
            return true;

    return false;
```

We used ! —the *NOT* operator—to check if the string is *NOT* null or empty. It's just like checking to see if something's false.

The queen uses the worker's WorkOneShift() method to tell him to work the next shift. The method only returns true if this is the very last shift that he's doing the job. That way the queen can add a line to the report that the bee will be done after this shift.

```
public bool WorkOneShift() {
    if (String.IsNullOrEmpty(currentJob))
        return false;
    shiftsWorked++;
    if (shiftsWorked > shiftsToWork) {
        shiftsWorked = 0;
        shiftsToWork = 0;
        currentJob = "";
        return true;
    }
    else
        return false;
```

Take a close look at the logic here. First it checks the currentJob field: if the worker's not working on a job, it just returns false, which stops the method. If not, then it increments ShiftsWorked, and then checks to see if this is the job's done by comparing it with ShiftsToWork. If it is, the method returns true. Otherwise it returns false.

```
public cLass Queen {
    public QueenC Worker l] workers) {
        this.workers • workers;
    }

    privete Worker[] workers;
    private int shiftNumber _ 0;

    public bool AssignWork(string job, int numberOfShifts)
        for lint i _ 0; i < workers.Length; i++)
            if (workers[i J•DoThisJob(job, numberOfShifts))
                return true;
        return false;
    }

    pUblic string WorkTheNextShift() {
        shiftNumber++;
        string report = -Report for shift #" • shiftNumber • "\r\n";
        for (int i • 0; i < workers.Lenqth; i++)
        {
            if (workers[i].WorkOneShift())
                report += "Worker #" + (i • 1) + " finished the job\r\n";
            it (String.IsNullOrEmpty(workers[i].CurrentJob))
                report += -iiorker #" . (i • I) + " is not working\r\n";
            else
              if (workers[i].ShiftsLeft > 0)
                report += "Worker #" - (i - II • - is doing .- • workers[i].CurrentJob
                    + "' for " + workersli.l.ShittsLeft + " more shifts\r\n";
              else
                report •• -Worl<er #" • Ii • 1) + " will be done with .-
                    + workers[i].CurrentJob + "' after this shift\r\n";
        }
        return report:
    }
```

The queen keeps her array of workers private because once they're assigned, no other class should be able to change them... or even see them, since she's the only one who gives them orders. The constructor sets the field's value.

When she assigns work to her worker bees, she starts with the first one and tries assigning him the job. If he can't do it, she moves on to the next. When a bee who can do the job is found, the ...ethod returns (which stops the loop).

The queen's WorkTheNextShift() method tells each worker to _k a shift and adds a ine to the ..eport depending _ the worker's status

We already gave '.o Uth.. constructor. Here's the rest ..f 11... cod.. for the form:

```
Queen queen:

private void assignJob_Cllcl«object eenoer, EventArgs el {
    if (queen.AssignWork(workerBeeJob.Text, Gnt)shitts.valuel == false)
        MessageBox.Show("No workers are available to do the job '"
            + workerBeeJob.Text + "'", -rne queen bee says...-i•
    else
        MessageBox.Show("The job '" • workerBeeJob.Text + -. will ee done in "
            • shifts.Value . • shifts", "The queen eee says...");
}

private void nextShift_Click(object sender. EllentArqs e) I
    report.Text • queen.WorkTheNextShift();
}
```

The for... uses its queen field to keep a reference to the Queen object, which in turn has an array of references to the worker objects.

The assignJob button calls the ,..een's AssignWork() ...ethod to assign work to a worker, and displays a ...essagebox depending on whether or not a worker'1 available to do the job

The nextShift button tells the ,ueen to work the next shift. She generates a report, which it displays in the report text box.



Objeetcross

Before you move on to the next part of the exercise, give your brain a break with a quiok crossword.



Across

5 This method gets the value of a property
7. This method returns true il you pass it ••
8. The constructor in a subclass class doesn't need the same as the constructor in ns base class
9, Aconlrol on a form that lets you create tabbed applications
11 This type of class cent be instantiated

Down

1. A can ovemoe methods from ns base class
2. Il you want a subclass to override a method, mark the method with this keyword in the base class
3. A method in a class eere run as soon as irs instantiated.
4 What a subclass does to replace a method in the base class
6. Th is contains base classes and subdasses
7. What you're doing when add a coion to a class declaration,
10. A subclass uses this keyword to call the members of the class it inherited from

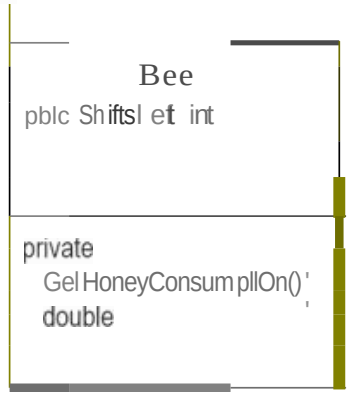
— — — —•• Answers on page 250,

Use inheritance to extend the bee management system



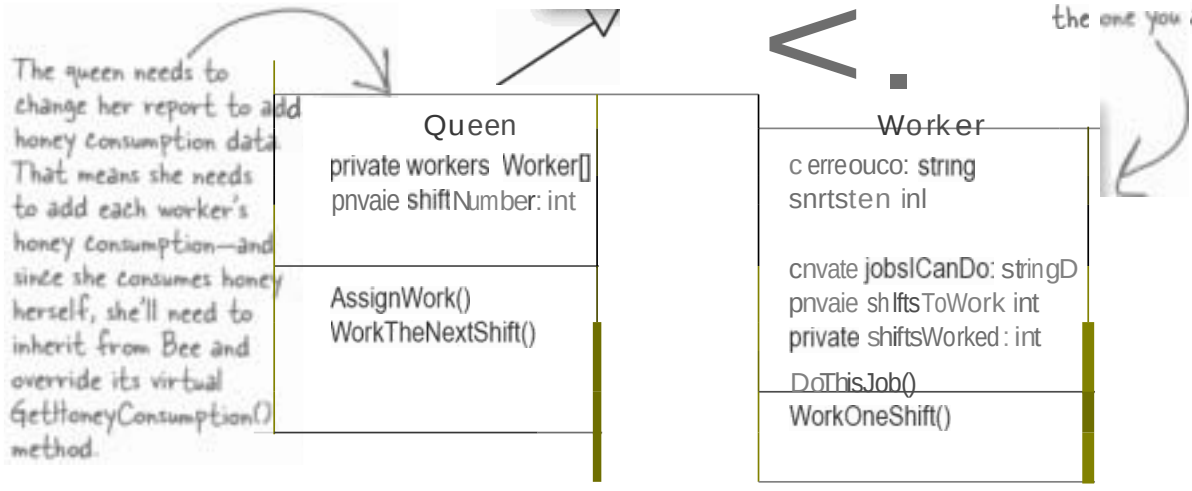
Now that you have the basic system in place, you'll use inheritance to make it track how much honey each bee consumes. Different bees consume different amounts of honey, and the queen consumes the most honey of all. So you'll use what you've learned about inheritance to create a Bee base class that Queen and Worker inherit from.

The Bee class has the basic honey consumption behavior. Since honey consumption requires the number of shifts left, we'll move the ShiftsLeft property into it and mark it virtual so the Worker can override it.



The worker just needs to subclass Bee overide the ShiftsLeft method with the one you already wrote

The queen needs to change her report to add honey consumption data. That means she needs to add each worker's honey consumption—and since she consumes honey herself, she'll need to inherit from Bee and override its virtual GetHoneyConsumption() method.



Add Existing Item

Whenever you have a two-part exercises, it's always a good idea to start a new project for the second part. That way you can always get back to the first solution if you need it. An easy way to do that is to right-click on the project name in the new project's Solution Explorer in the IDE, navigate to the old project's folder, and select the files you want to add. The IDE will make new copies of those files in the new project's folder, and add them to the project. There are a few things to watch out for, though. The IDE will NOT change the namespace, so you'll need to edit each class file and change its namespace line by hand. And if you add a form, make sure to add its designer (.Designer.cs) and resource (.resx) files—and make sure you change their namespaces, too.







We're not done yet! The queen needs to keep track of how much honey the hive is spending on us workers. Here's a perfect chance to use your new inheritance skills!

1 The **queen** needs to know how **much** hooey the **hive** uses

The queen just got a call from her accountant bees, who told her that the hive is producing enough honey. She'll need to know how much honey she and her workers are using so she can decide whether to divert workers from egg maintenance to honey production.

- All bees eat honey, so this method runs through a loop of honey. That's why they need to keep making more of it.
- Worker bees use money when they're working. They need the most honey when the job starts, to give them plenty of energy for the job. They consume less and less as the job goes on. On the last shift the bee uses 10 units of honey, the second-to-last shift he uses 11 units, the shift before that he uses 12 units, etc. So if the bee's working (meaning its ShiftsLeft is greater than zero), then you can find out how many units of money it will consume by adding 9 to ShiftsLeft.
- If a bee doesn't have a job (i.e. its ShiftsLeft is zero), he only uses 7.5 units of honey for the shift.
- Those numbers are all for normal bees. If a bee weighs more than 150 milligrams, it uses 35% more honey. This includes both workers and queens.
- Queens require a lot of honey. A queen uses more honey when she's got more workers doing jobs because it's a lot of work overseeing them. She needs to consume as much honey as if she'd worked as many shifts as the worker with the most shifts left in his job.
- Then she needs even more honey: she uses 20 extra units of honey per shift if there are 2 or more fewer workers working, and 30 extra units of honey if there are 3 or more fewer working jobs. That's the queen's contribution to the 35% rule, since all queens weigh 275 milligrams.
- The queen needs all the honey consumption numbers added to the end of each shift report.

2 **Create a Bee** class to **handle the honey calculations**

Since the workers and queen all do their honey calculations in similar ways, you'll be able to avoid duplicating your code by having a Bee base class that Worker and Queen both inherit from. You know that each bee needs to know its weight (so it knows whether to multiply its honey consumption by 35%).

- Create a getHoneyConsumption() method that calculates the amount of honey that a worker uses. Since the workers and queen all need to do this calculation but the queen needs to do additional calculations as well, it makes sense for the worker to implement it and the queen to override it.
- The getHoneyConsumption method needs the number of shifts left, so add a read-only property called ShiftsLeft that returns zero. The worker's ShiftsLeft will override it.
- The honey consumption calculation needs to know the bee's weight, so the Bee constructor will need to take the weight as a parameter and store it in a field. Since no other class needs to use it, you should make it private.



Here's a good rule of thumb: You should make fields and methods private by default, and only make them public if another class needs them. That way you avoid bugs in your program caused by one class accessing another class's properties or methods incorrectly.

Hint: You can use the slightly cryptic "no overload" error message to your advantage! Have the Worker class inherit from Bee, then build your project. When the IDE displays the error, double-click on it and the IDE will jump right to the Worker constructor automatically. How convenient!



3 Make the Worker class inherit from Bee

Yont'Jj need to set up Ihe constructor to call the base class constructor, like you did with Kathleen. You 'll need ,,' change the \Vorker constructor so that i, take the bee's weight as a parameter, and pass that parameter on ,,' the base class constructe,r, Then, just add the override keyword to rhe \Vorker's ShiftLeft n1<·,h,,<I. Once you do that, each worker will I,,· able '0 calculate his honev consumption for the queen... and vou don', have '0 make anv m",,· changes In the \Vorker class!

O Moke the Queen class inherit from Bee

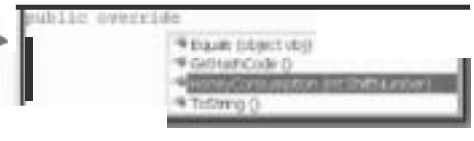
The Queen class needs a little more alteration than rh,' Worker class, since she needs r" actually do the h"ney calculati"ll and add i, '0 the shift report.

- Override Ihe Bee.GetHoneyConsumptie'"0 method and add the queen's extra calculation. She'll need I" figure ''''11 II hether she has 2 ,,,-fewer worker with j, bs, so sl". knows II hether ,1,,· needs 20 " 1'30 units. Then she'll need to add that 10the number or u"i" she'd use if she h,,,l the same number of shifts left as the worker with the 1ll"" shifts left.

- Update the queen's WorkTheNextShift() by adding the a honev consumption line to the report. Add a loc'I' '', add "I' the h,,,ney consumptions fc,,-each worker and also lind the worker with ,h,' largest honey ",,"slImpli'"l - d" i, b..I'0.... ,h,' <]"" "11 tells each "-" rk,,,.I" work each shift (so she gets ,h,' consumption numbers for ,h,' current shift). Sh,"U add those "I', add her own consumption, aud then add a line '" the "ud or the shift report that says, "Total Honey Consumption: xxx units" (where xxx is rhe number of units of honey consumed).

- You'll need 10 update the Queen constructor just like you did 1'" Worker.

Go to the Queen class and type "public ove.ride"—when you press the space bar, the IDE automatically lists all the methods you can override. Select the method you want to over.ide and it'll fill in the base ...ethod call automatically



O Updote the **form** to **instantiate** the **bees** properly

Since you changed the Queen and Worker constructors, you'll also need ," change the way they're called. Each constructor has a new \Veight parameter, so you'll need weights to use:

- \\·ork Bee # I: 175mg; \\',,rker 11,,e #2: 114mg; Worker Bee #3: 149mg; Work Bee#4: 155mg; Queen Bee: 275mg

.['hat' the ,nly change 1'ou'll need to make to the form!



Here's the Bee class. It does the basic honey consumption calculation that's used by both the Worker and Queen classes.

```
public class Bee {
    public Bee(double weight) {
        this.weight = weight;
    }

    public virtual int ShiftsLeft {
        get { return 0; }
    }

    private double weight;

    public virtual double GetHoneyConsumption() {
        double consumption;
        if (ShiftsLeft == 0)
            consumption = 7.5;
        else
            consumption = 9 + ShiftsLeft;
        if (weight > 150)
            consumption *= 1.35;
        return consumption;
    }
}
```

The Bee class has a constructor that sets its Weight field and a HoneyConsumption() method that calculates how much honey a worker consumes.

If a bee has 1 shift left, he consumes 10, if 2 left, he consumes 11, etc. If he has no job, then he consumes 7.5. If ShiftsLeft is zero, then the bee has no job.

If the bee weighs more than 150mg, then consumption goes up by 35%.

[nlieritanee **made** it easy **for** you to update your **code** and **add the new honey consumption behavior** 10 **the** Queen **and Worker classes. It would have been** a lot **harder** to make **this change** il you'd **had** a lot 01 **duplicated code.**

Only the f... constructor changed—the rest of the f... exactly the same.

```
public Form1() {
  InitializeComponent();

  Worker[] workers = new Worker[4];
  workers[0] = new Worker(new string[] { "Nectar collector", "Honey manufacturing" }, 175);
  workers[1] = new Worker(new string[] { "Egg care", "Baby bee tutoring" }, 114);
  workers[2] = new Worker(new string[] { "Hive maintenance", "Sting patrol" }, 149);
  workers[3] = new Worker(new string[] { "Nectar collector", "Honey manufacturing",
      "Egg care", "Baby bee tutoring", "Hive maintenance", "Sting patrol" }, 155);
  queen = new Queen(workers);
```

The only change to the form is that the weights need to be added to the Worker constructors.

```
public class Worker : Bee {
    public Worker(string[] jobsICanDo, int weight)
        : base(weight) {
        this.jobsICanDo = jobsICanOo;

    public override int ShiftsLeft {
    // ... the rest of the class is the same ...
```

All the Worker class needed was to inherit from Bee and have its constructor adjusted so that it takes a Weight parameter and passes it on to the base class constructor, and overrides the Bee.ShiftsLeft property by adding the override keyword to the property declaration.

The Queen class needed a few changes, starting with inheriting from Bee

```
public class Queen : Bee {
    public Queen(Worker[] workers)
        : base(275) {
        this.workers = workers;

    public string WorkTheNextShift()

        double totalConsumption = 0;
        for (int i = 0; i < workers.Length; i++)
            totalConsumption += workers[i].GetHoneyConsumption();
        totalConsumption += GetHoneyConsumption();

        //     here's where the orignal code for this method goes

        report += "Total honey consumption: " + totalConsumption + " units";
        return report;
    }

    public override double GetHoneyConsumption() {
        double consumption = 0;
        double largestWorkerConsumption = 0;
        int workersDoingJobs = 0;
        for (int i = 0; i < workers.Length; i++)
            if (workers[i].GetHoneyConsumption() > largestWorkerConsumption)
                largestWorkerConsumption = workers[i].GetHoneyConsumption();
            if (workers[i].ShiftsLeft > 0)
                workersDoingJobs++;

        consumption += largestWorkerConsumption;
        if (workersDoingJobs >= 3)
            consumption += 30;
        else
            consumption += 20;
        return consumption;
```

The queen weighs 115mg, so her constructor calls the base Bee constructor and passes it a weight of 275

The WorkTheNextShift() has a loop added to the top that calls each worker's GetHoneyConsumption() method, and then calls her own GetHoneyConsumption() method to come up with a total consumption

The rest of WorkTheNextShift() is the same, except that it adds the honey line to the report.

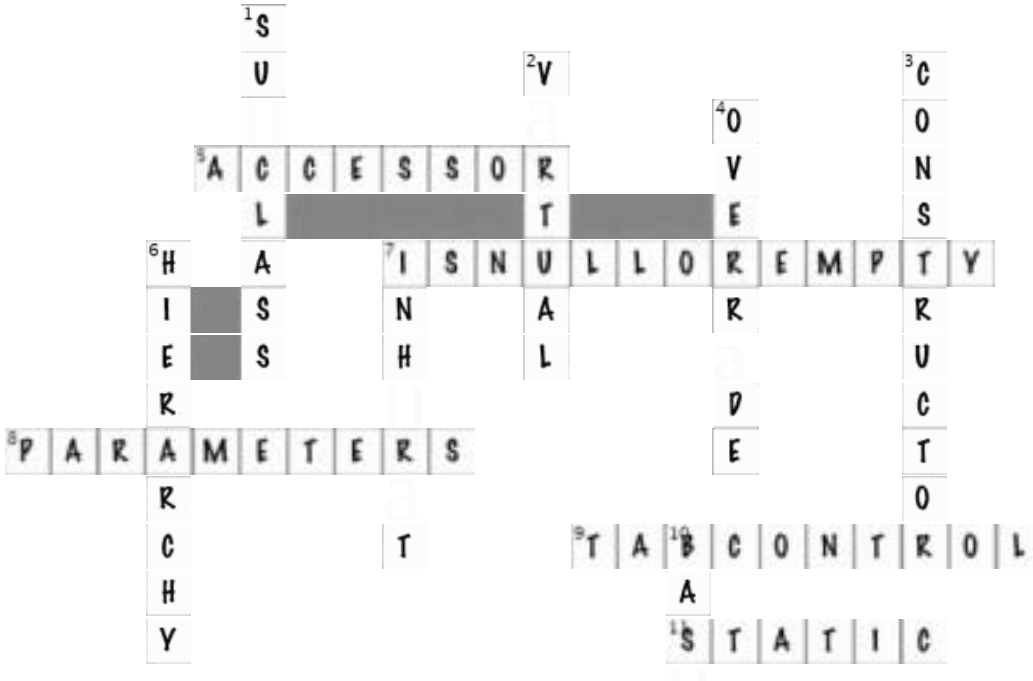
The queen overrides the Bee's GetHoneyConsumption() method to do her honey calculation. It finds the worker with the largest consumption and adds either 20 or 30 to it based on how many workers are working

This loop looks at the consumption of all the workers and finds the one with the largest consumption.

If there are 3 or more workers doing jobs, the queen needs 30 more units of honey; otherwise, she needs 20 more units.



Objectcross Solution



Making classes keep their promises

Okay. okay. I know I implemented the BookieCustomer interface, but I can't code the PayMoney() method until next weekend.



You've got three days before I send some Thug objects by to make sure you implement the WalksWithALimp() method.

Actions speak louder than words.

Sometimes you need to group your objects together based on the things they can do rather than the classes they inherit from. That's where interfaces come in--they let you work with any class that can do the job. But with great power comes great responsibility, and any class that implements an interface must promise to fulfill all of its obligations or the compiler will break their kneecaps, see?

Let's get back to bee-sics

The General Bee-namics corporation wants to make the Beehive Management System you created in the last chapter into a full-blown Hive Simulator. Here's an overview of the specification for the new version of the program:



General Bee-namics Hive Simulator

To better represent life in the hive, we'll need to add specialized capabilities to the worker bees.

- All bees consume honey and have a weight.
- Queens assign work, monitor shift reports, and tell workers to work the next shift.
- All worker bees work shifts.
- Sting patrol bees will need to be able to sharpen their stingers, look for enemies, and sting them.
- Nectar collector bees are responsible for finding flowers, gathering nectar and then returning to the hive.

The Bee and Worker classes don't look like they'll change much. We can extend the classes we already have to handle these new features.

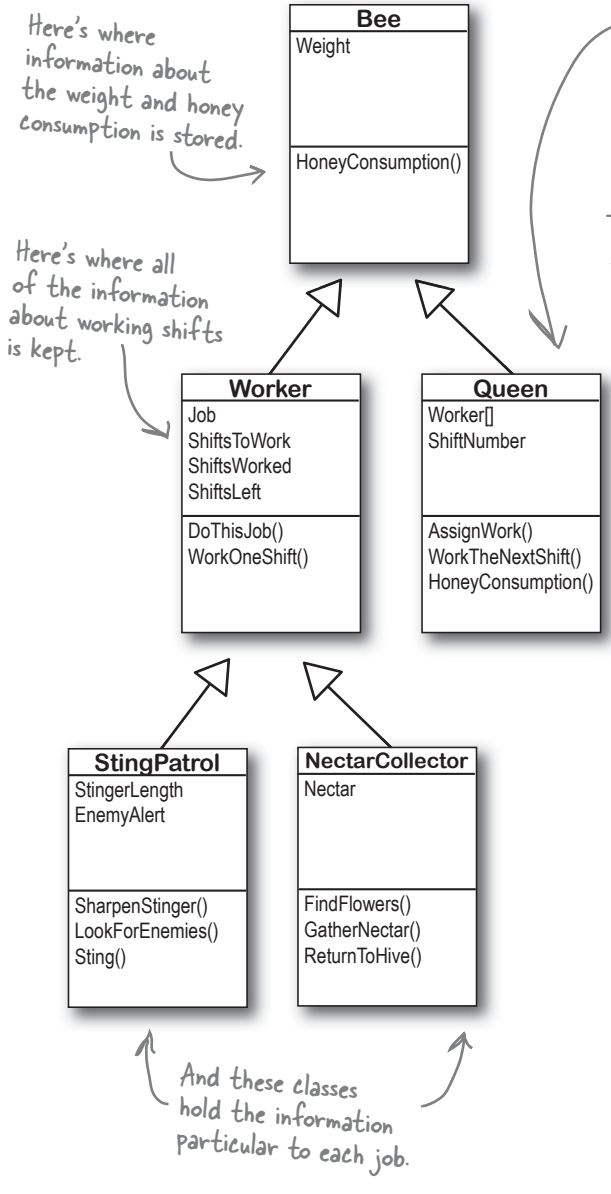
Looks like we'll need to be able to store different data for the worker bees depending on the job they do.

Lots of things are still the same

The bees in the new hive simulator will still consume honey in the same way they did before. The queen still needs to be able to assign work to the workers and see the shift reports that tell who's doing what. The workers work shifts just like they did before, too, it's just that the jobs they are doing have been elaborated a little bit.

We can use inheritance to create classes for different types of bees

Here's a class hierarchy with Worker and Queen classes that inherit from Bee, and Worker has subclasses NectarCollector and StingPatrol.



This is what the new subclasses will look like.

StingPatrol and NectarCollector inherit from the Worker class.

```
class StingPatrol : Worker
{
    int StingerLength;
    bool enemyAlert;
    public bool SharpenStinger (int Length)
    {...}
    public bool LookForEnemies(){...}
    public void Sting(string Enemy){...}
}

class NectarCollector : Worker
{
    int Nectar;
    public void FindFlowers (){...}
    public void GatherNectar(){...}
    public void ReturnToHive(){...}
}
```



What happens if you have a bee that needs to sting ***and*** collect nectar?

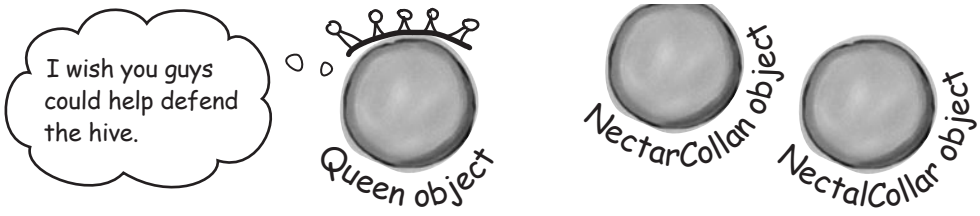
An interface tells a class that it must implement certain methods and properties

A class can only inherit from one other class. So creating two separate subclasses for the `StingPatrol` and `NectarCollector` bees won't help us if we have a bee that can do **both** jobs.

The queen's DefendTheHive() method can only tell StingPatrol objects to keep the hive safe. She'd love to train the other bees to use their stingers, but she doesn't have any way to command them to attack:

```
public class Queen {
  private void DefendTheHive(StingPatrol patroller) { ... }
}
```

You use an interface to require a class to include all of the methods and properties listed inside the interface—if it doesn't, the compiler will throw an error.



There are NectarCollector objects that know how to collect nectar from flowers, and instances of StingPatrol that can sharpen their stingers and patrol for enemies. But even if the queen could teach the NectarCollector to defend the hive by adding methods like SharpenStinger() and LookForEnemies() to its class definition, she still couldn't pass it into her DefendTheHive() method. Maybe she could use two different methods:

```
private void DefendTheHive(StingPatrol patroller);
private void AlternateDefendTheHive(NectarCollector patroller);
```

Even if the queen adds sting patrol methods to a NectarCollector object, she still can't pass it to her DefendTheHive() method because it expects a StingPatrol reference. She can't just set a StingPatrol reference equal to a NectarCollector object.

She could add a second method called AlternateDefendTheHive() that takes a NectarCollector reference instead, but that would be cumbersome and difficult to work with.

Plus, the DefendTheHive() and AlternateDefendTheHive() methods would be identical except for the type of the parameter. If she wanted to teach the BabyBeeCare or Maintenance objects to defend the hive, she'd need to keep adding new methods. What a mess!

But that's not a particularly good solution. Both of those methods would be identical, because they'd call the same methods in the objects passed to them. The only difference is that one method would take a StingPatrol, and the other would take a NectarCollector that happens to have the methods necessary for patrolling the hive. And you already know how painful it is to maintain two identical methods.

Luckily, C# gives us **interfaces** to handle situations like that. Interfaces let you define a bunch of methods that a class ***must*** have.

An interface ***requires*** that a class has certain methods, and the way that it does that is that it ***makes the compiler throw errors*** if it doesn't find all the methods required by the interface in every class that implements it. Those methods can be coded directly in the class, or they can be inherited from a base class. The interface doesn't care how the methods or properties get there, as long as they're there when the code is compiled.

Use the `interface` keyword to define an interface

Adding an interface to your program is a lot like adding a class, except you never write any methods. You just define the methods' return type and parameters, but instead of a block of statements inside curly brackets you just end the line with a semicolon.

Interfaces do not store data, so you **can't add any fields**. But you *can* add definitions for properties. The reason is that get and set accessors are just methods, and interfaces are all about forcing classes to have certain methods with specific names, types and parameters. So if you want your interface to require a field with a certain name and type, just **use a property instead**—it'll accomplish the same thing.

Interface names start with I

Whenever you create an interface, you should make its name start with an uppercase `I`. There's no rule that says you need to do it, but it makes your code a lot easier to understand. You can see for yourself just how much easier that can make your life. Just go into the IDE to any blank line inside any method and type "I"—IntelliSense shows .NET interfaces.

You declare an interface like this:

Interfaces don't store data. So they don't have fields... but they can have properties.

```
public interface IStingPatrol
{
    int AlertLevel { get;}
    int StingerLength { get; set;}
    bool LookForEnemies();
    int SharpenStinger(int Length);
}
```

Any class that implements this method must have all of these methods and properties, or the program won't compile.

Any class that implements this interface will need a SharpenStinger() method that takes an `int` parameter.

```
public interface INectarCollector
{
    void FindFlowers();
    void GatherNectar();
    void ReturnToHive();
}
```

Everything inside an interface is meant to be actually used inside another class. So all of the methods in an interface are automatically public.

You don't write the code for the methods in the interface, just their names. You write the code in the class that implements it.

Since this takes an IStingPatrol reference, you can pass it *ANY* object that implements IStingPatrol.

So how does this help the queen? Now she can make one single method that takes any object that knows how to defend the hive:

```
private void DefendTheHive(IStingPatrol patroller)
```

This gives the queen a single method that can take a `StingPatrol`, `NectarStinger`, and any other bee that knows how to defend the hive—it doesn't matter which class she passes to the method. As long as it implements `IStingPatrol`, the `DefendTheHive()` is guaranteed that the object has the methods and properties it needs to defend the hive.



Everything in a public interface is automatically public, because you'll use it to define the public methods and properties of any class that implements it.

Now you can create an instance of NectarStinger that does both jobs

You use the **colon operator** to declare an interface, just like you do for inheritance. It works like this: the first thing after the colon is the class it inherits from, followed by a list of interfaces -- unless it doesn't inherit from a class, in which case it's just a list of interfaces (in no particular order).

You implement an interface with a colon operator, just like you inherit.

This class inherits from Worker and implements INectarCollector and IStingPatrol.

```
class NectarStinger : Worker, INectarCollector,
IStingPatrol {
        public int AlertLevel {
          get { return alertLevel; }
        }

        public int StingerLength {
           get { return stingerLength; }
           set {
                 stingerLength = value;
           }
        }

        public bool LookForEnemies() {...}
        public int SharpenStinger(int Length)
        {...}
        public void FindFlowers() {...}
        public void GatherNectar() {...}
        public void ReturnToHive() {...}
}
```

You can use more than one interface if you separate them with commas.

The NectarStinger sets the backing field for the AlertLevel property in its LookForEnemies() method.

Every method in the interface has a method in the class. Otherwise it wouldn't compile.

When you create a NectarStinger object, it will be able to do the the job of both a NectarCollector and a StingPatrol worker bee.

The bee retracts its stinger when there are no enemies around, so the backing field changes its value over time.

When you've got a class that implements an interface, it acts just like any other class. You can instantiate it with `new` and use its methods:

```
NectarStinger bobTheBee = new NectarStinger();

bobTheBee.LookForEnemies();

bobTheBee.FindFlowers();
```

there are no Dumb Questions

Q: I still don't quite get how interfaces improve the beehive code. You'll still need to add a NectarStinger class, and it'll still have duplicate code…right?

A: Interfaces aren't about preventing you from duplicating code. They're about letting you use one class in more than one situation. The goal is to create one worker bee class that can do two different jobs. You'll still need to create classes for them—that's not the point. The point of the interfaces is that now you've got a way to have a class that does any number of jobs. Let's say you have a PatrolTheHive() method that takes a StingPatrol object and a CollectNectar() method that takes a NectarCollector object. But you don't want StingPatrol to inherit from NectarCollector or vice versa—each class has public methods and properties that the other one shouldn't have. Now take a minute and try to think of a way to create one single class whose instances could be passed to both methods. Seriously, put the book down, take a minute and try to think up a way! How do you do it?

Interfaces fix that problem. Now you can create an IStingPatrol reference—and that reference can point to any object that implements IStingPatrol, no matter what the actual class is. It can point to a StingPatrol, or a NectarStinger, or even a totally unrelated object. If you've got an IStingPatrol reference pointing to an object, then you know you can use all of the methods and properties that are part of the IStingPatrol interface, regardless of the actual type of the object.

But the interface is only part of the solution. You'll still need to create a new class that implements the interface, because it doesn't actually come with any code. Interfaces aren't about avoiding the creation of extra classes or avoiding duplicate code. They're about making one class that can do more than one job without relying on inheritance, because inheritance brings along a lot of extra baggage—you'll have to inherit every method, property and field, not just the ones that have to do with the specific job.

Can you think of ways that you could still avoid duplicating code while using an interface? You could create a separate class called Stinger or Proboscis to contain the code that's specific to stinging or collecting nectar. NectarStinger and NectarCollector could both create a private instance of Proboscis, and any time they need to collect nectar, they'd call its methods and set its properties.

Classes that implement interfaces have to include ALL of the interface's methods

Implementing an interface means that you have to have a method in the class for each and every property and method that's declared in the interface—if it doesn't have every one of them, it won't compile. If a class implements more than one interface, then it needs to include all of the properties and methods in each of the interfaces it implements. But don't take our word for it...

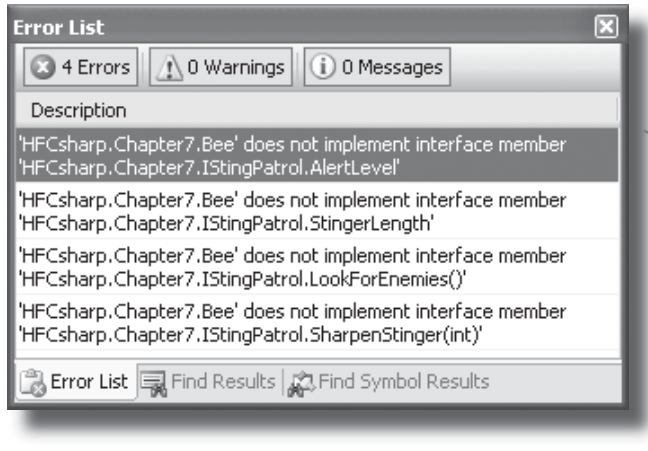


1. **Create a new application and add a new class file called IStingPatrol.cs**
Instead of adding a class, type in the **`IStingPatrol`** interface on the previous page.

2. **Add a Bee class to the project**
Don't add any properties or methods yet. Just have it implement IStingPatrol:

```
public class Bee : IStingPatrol {
```

3. **Try to compile the program**
Select "Rebuild" from the Build menu. Uh-oh—the compiler won't let you do it:



You'll see one of these "does not implement" errors for every member of IStingPatrol that's not implemented in the class. The compiler **really** wants you to implement every method in the interface.

4. **Add the methods and properties to the Bee class**
Add a `LookForEnemies` method and a `SharpenStinger` method—they don't have to do anything, they just need to compile. Then add a get accessor for an `int` called AlertLevel and get and set accessors for an `int` called StingerLength. Now the program will compile!

Get a little practice using interfaces

Interfaces are really easy t" use, but the h."t way to understand is to start using them. S" create a new Windows Forms Application project, drag a hUllo" "H'O the form, and get started!



1 Here's the TallGuy class, alld the code for a hUllo" that create it using an "I,j""l initializer and calls its TalkAbo"u Yourself() method. Nothing new here we'll use it in a minute

```
public class TallGuy {
    public string Name;
    public int Height;

    public void TalkAboutYourself()
        MessageBox.Show("My name is " + Name + " and I'm "
            + Height + " inches tall.");
}
```

```
private void button1_Click(object sender, EventArgs e) {
    TallGuy tallGuy = new TallGuy() { Height = 74, Name = "Jimmy" };
    tallGuy.TalkAboutYourself();
}
```

2 Let's create an IClown interface for tite class.

You already know th;ll everything inside an illl<,rface has to> be public. B"l don't take lit" word for it. (:reate a new project and declare an interface on your own, hk,' this:

```
public interface IClown
```

Now try to declare a private method inside the interface:

```
private void Honk();
```

Select Build>>Bnild Sol",io" ill th,' IDE. Yo>ll'll see this error:

> You don't need to type "public" inside the interface, because it automatically makes every property and ...ethod public.

1	The modifier 'private' is not valid for this item

Now go ahead and *delete* the **private access** modifier the error will go away ;lIld your program will compile just fine.

3 Before you g(Oil to the next page, "" if you I';tll create rhe rest of ,he IClown interface-, ;llllli modify th,' TallGuy class to implement this interface. Add your interface t> your pn)ject jU'1 lik,' yo>ll add a class: right-click on the proj ·1 in the Sollll ion Explorer and add a class file calle1 IClown•Cs.

Your new IClown int face should have a void method , alled Honk that d,esn't take any parameters and a string read-only property called FunnyThingIHave that has a get access or hut IIOset accessor.

6 Here's the interface—did you get it right?

```
public interface IClown

    string FunnyThingIHave    get; }
    void Honk();
```

Here's an example of an interface that has a get accessor without a set accessor. Remember, interfaces can't contain fields, but when you implement this read-only property in a class, it'll look like a field to other objects.

Okav; now modify the TallGuy class so tn.« i, implements clown. Remember, the colon operator is alway followed bv the base class '0 inherit from (if any), ,llld then a li" "r interfaces '" implement, ,,ll separated by commas. Since there's "" base class a",1onlv OtH' interface to implement, the declaration looks like this:

TallGuy will implement the IClown interface.

```
public class TallGuy : IClown
```

What the IDE is telling you is that when V- said TallGuy would implement leiown, you promised to add all of the properties and methods in that nterface .. and then you broke that promise!

Then make sure the rest of the class is (h,' same, including the two fields and the method. Select "Bllild s, ,ltHi",," from the Build menu in the Il lE (.,"(ompile and hnild the program. Y",,'ll see two errors, including this one:

```
o  'TaIIGuy' does not implement interface
   member 'IClown.Honk(),
```

6 The "rrors will go away a soon a' you add "II of the methods and properties ddill"d in the interface. So go ahead and implement the interface. Add a read-only string property called FunnyThingIHave with a get accesso·r that always returns the string "big shoes". Then ,,,Id a Honk() method that pops "p a message box that says, "Honk IHmk!",

Here-, what irllbok like:

All the interface requires is that a clas that implements it has a property called FunnyThingIHave with a get accessor. You can put any get accessor in there, even one that just returns the same strin_ every time Most get accessors won't do this, but this will work Just fine if it does what you need it to do

```
public string FunnyThingIHave
    get ( return "big shoes";

public void Honk() {
    MessageBol<.Show("Honk honk!");
```

The interface says that you need a public void ...ethod called Honk, but it deesn't say what that ...ethod needs to do It can do anything at all matter hat it dees, the code will compile as long as e ...ethod is there with the right signature.

7 Now your code will compile! Update your button so that the ohj""l initializer sets Ih,' FunnyThingIHave property, and the button calls the TallGuy "bj"" 's Honk() method.

You can't il1stal1tiate an lntsrfaee, but you can referel1ce an il1terface

Say you had a 1llt'lhod that Il,',>d,'ct an object Ihal could perform the FindFlowers() method. .Any objec·1 1hat implemented the INectarCollec1')[' interface would do. It could Ье a \\'orker "bi'TI. or a Robot object or a Dog object as long as il implements Ih,· INectarCollector interface.

That's wh..re int..**rface refe**r..n**ce**s come in. You can use on,' to n-k-r 10 an object that imph-nu-nts the inn-rfao- you need and you'll always Ье sure that il has the right methods for your purpose—even if you d"n'l know much else about it.

If you try to the instantiate an interface, the compiler will complain.

This won't work...

```
IStingPatrol dennis = new IStingPatrol();
```

```
O 1 Cannot create an instance of the abstract class or interface
```

You can't use the new keyword with all interface, "I,id, make sense- the methods a",II""I)(TI;es do,,'1 have any i"'p],'"u·mal;ou. If you could create an "bj" "llrolll an interface, how would it know how 10 behave?

...but this will.

```
NectarStinger fred = new NectarStinger();
IStingPatrol george = fred;
```

Remember how you could pass a BLT reference into any class that expects a Sandwich, because BLT inherits from Sandwich? Well, this is the same thing—you can use a NectarStinger in any method or statement that expects an IStingPatrol.

Even though this object can do more, when you use an interface reference you only have access to the methods in the interface.

The first line is an ordinary new statement, creating reference called Fred and pointing it to a NectarStinger object.

The second line is where things start to get interesting, because that line of code **creates a new reference variable using IStingPatrol**. That line may look a little odd when you first see it. But look at this:

```
NectarStinger ginger = fred;
```

Yo" kno" "hal this Ihi,d statement does ıı create an,'" NectarStinger reference called ginger and ()(,illt, il at whatever object fred i, pointing 10. The george statement uses IStingPatrol the same way.

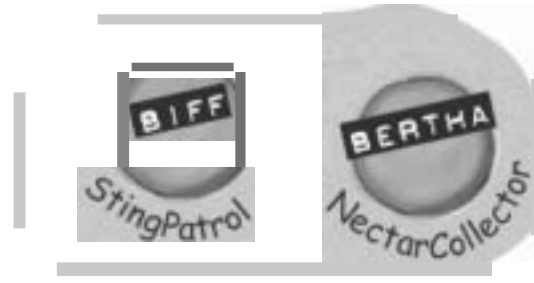


So what happened?

There's onlv '"|(' new statement, so **only** one object was created. The second statement created a reference variable called george that can 1)(';ll1lO an instance ..f **any class** that inlp**lements** ISt:1ngPatro l .

Interface references work just like object references

y,,,, already know all about how objects live on the heap. When you work with an interface referenc it's just another way t" refer 10 the same objects vou've alr ady been dealing with. Look—it's easy!



O **Create a** couple of **bees**
Thi, i, totally familiar 'Htll by now.

```
StingPatrol biff = new StingPatrol():
NectarCollector bertha = new NectarCollector(1;
```

O Add IStingPatrol **and** INeetarCollector **references**
You can use interface references just [ik,' you use anv other reference type.

```
IStingPatrol defender - bitt;
INectarCollector cutiePie = bertha;
```

These two statements «se interfaces to create new references to existing objects. You can only point an interface reference at an instance of a cla.. that i..plements it.



3 An interfoce **reference** will keep on object **alive**
Wheulhn.· aren't any references pointing to ,tit object, il disappears. But there's IlO rule lhal says those refere·"'es all have m be the same type! .\n interface reference is just as good as an object reference when ;1 comes to, keeping track "I' objects. /

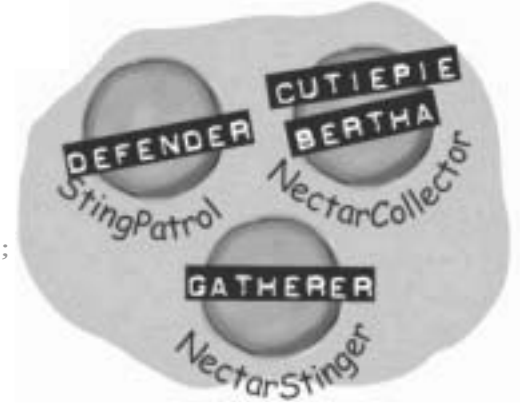
```
bitt - null;
```

This object didn't disappear because defender is still pointing to it.



O Assign a **new instance** to **an** interfoce **reference**
You don't actually *need* an object reference you can create all"" object and assign il straight to an interface reference variable.

```
INectarCollector gatherer - new NectarStinger();
```



You can find out if a class implements a certain interface with "is"

Sometimes you need to find "" if a ""nai" class illlpl'·lll<·nt' a" interface. Suppose we have all our w,ɔrker bees ill an array, called Bees. \Ve ("all make the array h,ɔld the *t"* ɔe \\" rk,·r. since all worker Bees will be Worker classes, or subclasses "f that type.

Btll which of the worke h,-," ,';tlt collect Il,',-lar'! In o,h"r words, we want '0 kll"" if the class implements the I Ne c:ta rCol l e c:tor interface. We ,'a" use the `is` keyword '0 find out exactly ,ha t.

All the workers are in an array of Workers. We'll use "is" to sort out which type of worker each bee is.

We've got an array of Worker bees who are all eligible to go on a nectar collecting mission. So we'll loop though the array, and use "is" to figure out which ones have the right methods and properties to do the job.

```
Worker[] Bees = new Worker[3];
Bees[0] = new NectarCollector(),
Bees(l] - new StingPatrol();
Bees[2] - new NectarStinger(),
for (int i = 0: i < Bees.Length: i++)
{
    if (Bees[i] is I                :tor)
    (
        Bees(i].DoThisJob("Nectar Collector", 3),
    }
}
```

is works like an equals operator (==) for interfaces

This is like saying, if this bee implements the INectarCollector interface... do this.

Now that we know the bee is a nectar collector, we can assign it the job of collecting nectar.

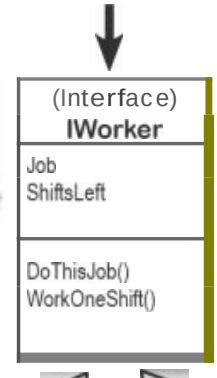


If you have some other class that doesn't inherit from Worker but *does* implement the INectarColiector interface. then it'll be able to do the job. too! But since it doesn't inherit from Worker. you can't get it into an array with other bees. Can you think of a way to get around the problem and create an array with both bees and this new class?

Interfaees call **inherit** frolll otber illterfaces

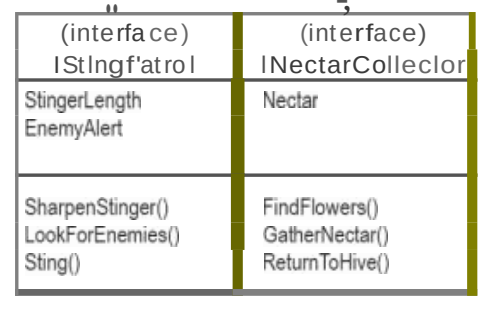
When one class inherits from another, il gets all of the 1ll"lh"d, and propertie from the base class. **Interface inheritance** is even simpler. Since there's "" actual method body in any interface, von don't have 10 worry about calling base constructors or methods. T h,' inherited interface simply accumulate all of the methods and propertie from the interfaces they inherit from.

When we draw an interface on a **class** diagram, we'll show inheritance using dashed lines.




```
public interface IWorker
{
    string Job { get; }
    int Left { get; }
    void DoThisJob(string Job, int Shifts)
    void WorkOneShift()
)
```

We've created a new IWorker interface that the other interfaces inherit from.



A.y class that **implements** a. i.teriace that i.herlts **from** IWorker **must implement** its **methods** a.d properties

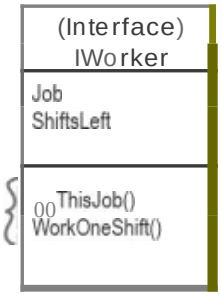
When a class implements an interface', il has lo include every property and method ill lhal interface·. And if lhal interface intwril' from another one, then all or *those* properties and methods need 10be implemented, too

```
public interface tStingPatrol : IWorker
    int AlertLevel { get;I
    1nt St1ngerLength ( get; set;)
    bool LookForEnemies ();
    1nt SharpenSt1nger(1nt Length);
```

Here's the same IStingPatrol interface, but now it inherits from the IWorke- interface It looks like a tiny change, but it makes a huge difference in any cla.. that imple..ents IStingPatrol.

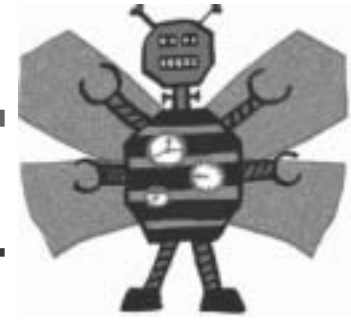
A class that implements IStingPatrol must not only implement these methods...

...but the ..ethod• of the IWorker interface this interface inhe-its from, too.



Ihe RoboBee 4000 call do a worker bee's job without usillg valuable honey

Let's create a new bee, a R" foBee 4000, thai runs on gas. \\t· ('all have it inherit from the IWorker ;"lt·.-l:,,e, though, so it can do everything a 1101'111"] worker bee can





This is our basic Robot class, so robots can run on gasoline.

```
public class Robot
{
      public void ConsumeGas () { ... }
}

public class RoboBee : Robot, IWorker
{
      private int shiftsToWork;
      private int shiftsWorked;
      public int ShiftsLeft
        (get {return ShiftsToWork - ShiftsLeft i} }
      private string job;
      public string Jobi{qet(return job;}}
      public bool DoThisJob(string Job, int ShiftsToWork){ ... )
)
```

The RoboBee class inherits from Robot and imple..ents IWorker. That mean, it's a robot, but can do the job of a worker bee. Perfect!

The RoboBee class implements all the methods from the IWorker interface.

If RoboBee didn't imflement everything in the IWorker Interface, the cede _lei"'t compile.

Remember. for other classes in the application, there's no functional difference between a RoboBee and a normal worker h.·:.. They both illll'l" lll"1l1 the IWorker interface so both act like w"rk"r h.·,·,as far as the """1 "I' the program i, concerned.

But. you could distringuish between the types by using `is`:

```
1f (workerBee is Robot) {
  // now we know worker Bee
  // is a Robot object
```

We can see what class or interface workerBee i..ele..ents or subclasses with "is."

Any **class** can implement ANY interface **as** long **as** it **keeps the** promise of implementing tl.e **interface's methods and properties.**

is tells you what an object implements, as tells the cOlllpiler how to treat your object

S, """lim", yc.u need 10 call a "I"lh, ic1 that an object gets from an interface it implements. But what if you don't know if ,hal object is the right type? You usc **is** '" find Iha, out. Then, you can use **as** '" Ireatlhat "hi,-'-t which you now know is the right type—as having the nu-thod you need 'o c;,!1

```
IWorker[] Bees = new IWorker[3];
    Bees[0] = new NectarStinger();
    Bees[1] = new RoboBee();
    Bees[2] = new Worker();
```

All these bees implement IWorker, but we don't know which ones implement other interfaces, like INectarCollector.

We're looping through each bee...

We can't call INectarCollector methods on the bees. They're of type IWorker, and don't know about INectarCollector methods.

```
for (int i = 0; i < Bees.Length; i++) {
    if (Bees[i] is INectarCollector) {
        INectarCollector thisCollector;
        thisCollector = workers[i] as INectarCollector;
        thisCollector.EmptyNectarBucket();
    }
}
```

...and checking to see if it implements INectarCollector.

...we use "as" to say, treat this object AS an INectarCollector implementation.

NOW we can call INectarCollector ..ethods.

Sharpen your pencil

Take a look at the array On the left. Fo, each of these statements, w,ite down whi(h values of i would make it evaluate to true. Also, two of them won't compile—cross those lines out.

```
rworker[] Bees = new IWorker[8];
Bees[0]   new NectarStinger();
Bees[1]   new RoboBee();
Bees[2]   new Worker();
Bees[3]   Bees[0] as IWorker;
Bees[4]   rStingPatrol;
Bees[5]   null;
Bees[6]   Bees l2];
Bees[7]   new INectarCollector();
```

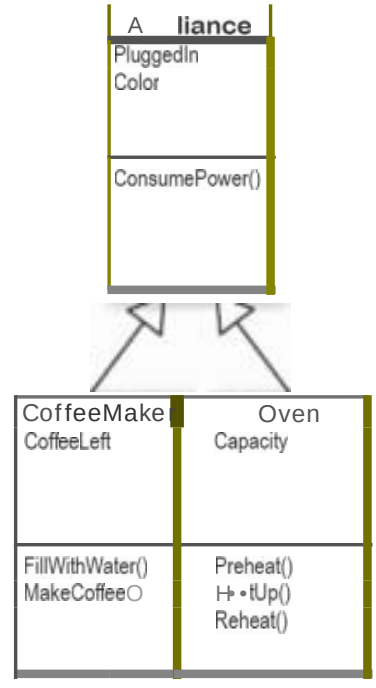
```
l, (Bees[i] is INectarCollector)

2 (Bees[i] is IStingPatrol)

3. tBeesli] is rworcerj
```

A CoffeeMaker is also an Appliallce

If you re trying 10 figure o u l how I" cut down you energy hill each month. you don't really care what each of your appliance does. You only really ("„'," that they consume P)wer. s" if you were writing a program 10 monit >1 y()ur electricity consumption, you'd probably ju" write an ·\l'l'li,lIK" class. But if you needed to) h., able to distinguish a coffee maker li-ll"';ltl "'-'ll, you'd have to huild a class hierarchy. S" you'd add the methods and pr"p.,rti,·, that are specific 10 a coffee make-r or oven to some CoffeeNlake-r and Oven classes. and they'd inherit from an Appliance class that has their common m,·thod, and properties..




```
public void MonitorpoWer(Appliance appliance)
    // code to add data to a household
    // power conswnption database
```

Here's a method in the progra to monitor t r consumption for a hous

This code would appear later on in the program to monitor the coffee maker's power consumption.

```
CoffeeMaker misterCoffee = new CoffeeMaker();
MonitorPower(misterCoffee);
```

Even though the MonitorPower() method takes a reference to an Appliance object, you can pass it the misterCoffee reference because CoffeeMaker is a subclass of Appliance.

You already saw this in the last chapter, when you saw how you could pass a BLT reference to a method that expected a Sandwich.

Sharpen your penCil Solution

Take a look at the array on the left. For each of these statements write dow n which values of i would make it evaluate to true. AI, o, two of them won't compile—cross them out.

```
IWorker[]
Bees[0]
Bees[1]
Bees[2]
Bees[3]
[illegible]
Bees[5]   null;
Bees[6]   Bees[O];
Egp [7]   new INectarCollector()
```

All of these objects inherit from base class Worker.

3. (Bees[i] is IWarker)

0, 1, 2, 3, and 6

Upcastil1g works with both objects al1d lnterfaees

\Vhe-n you ,,,Dstilllh' a subclass f'1' a base class —like substituting a coffee maker for an appliance "r a BLT for a sandwich- it's called **upcasting**. It's a really powerful t,ol that you get when vou build clas hierarchies. The only drawback to upcasting is thar you can only use tl,, · propertie and methods "I' the base class. Ill other words, when you treat a coffee maker lik,' ,111 appli;H1<t', you can't tell it It, make coffee "r till;1 with water. But you *can* tell whether 01" 11,,1 it's plugged ill. since that's something you can d" with any appliance (which is why the PluggedIn property is part or the Appliance class).

Let's create some objects

\Vc can create a CoffeeMaker and Oven class as usual:

```
CoffeeMaker misterCoffee - new CoffeeMaker();
Oven oldToasty - new Oven():
```

We'll start by instantiating an Oven object and a CoffeeMaker object as usual.

What if we want to create an array of appliances?

You run"t puta CoffeeMaker in an Oven[] arrav, and vou can't pili an Oven in a CoffeeMaker[] array. But you can pUI both "r them iu an Appliance[] array:

```
Appliance[] kitchenWare = new Appliance[2];
kitchenWare[OJ    misterCoffee:
kitchenWare[1]    oldToasty:
```

You can use upcasting to create an array of appliances that can hold both coffee makers and ovens.

But you can't treat an appliance like an oven

\\'I"'1lyou'v·got all Appliance reference, you can only access lit" methods and propertie th,lt have t· do with appliances. You **can't** use the ('olh'" maker methods and propertie through the Appliance referenc· ***even if you know il's really a CoffeeMaker***. So these statements will work just fine, becaus· they treat a CoffeeMaker object lik,' an Appliance:

```
Appliance powerConsumer - new CoffeeMaker();
powerConsumer.ConsumePower():
```

powerConsu...er is an Appliance reference pointing to a CoffeeMaker object.

Hit as soon as you try to use il lik,' a Coffee Maker:

```
powerConsumer.MakeCoffee();
```

This line won't compile because powerCon umer is an Appliance reference, so it can only be used to do Appliance things.

your code won't compile, and the IDE will displays an error:

```
'Appliance' does net; contaLn a definitiolll for 'MakeCoffee'
```



because· once you upcast from a subclass to a base class, (lien you ,.,,,, only access Ih,· meth·,,], ,,,ul prop"n;," th.u **match** th.. **r..f..r..nc..** that vou're using to access the object.

Downcasting lets you turn your appliance back into a coffee maker

Upcasting is a great tool, because it lets you use a coffee maker or an "" anywhere you just need an appliance. But it's got a big drawback if you're using an Appliance reference that points to a CoffeeMaker object, you can only use the methods and properties that belong to Appliance. And that's where **downcasting** come in: that's how you tell your **previously upcast reference** and change it back. You can figure out if your Appliance is really a CoffeeMaker using the **is** keyword. And once you know that, you can convert the Appliance back to a CoffeeMaker using the **as** keyword.

Here's our Appliance reference that points to a CoffeeMaker object from the last page



The javaJoe reference points to the same CoffeeMaker object as powerConsumer. But it's a CoffeeMaker reference, so it can call the MakeCoffee() method.



- We'll start with **the coffee maker we already upcast**. Here's the code that we used:

```
Appliance powerConsumer = new CoffeeMaker();
powerConsumer.ConsumePower();
```

- **But what if we want to turn the Appliance back into a CoffeeMaker?** The first step in downcasting is using the **is** keyword to check if it's even possible,

```
if (powerConsumer is CoffeeMaker)
   // then we can downcast!
```

- **Now that we know it's a CoffeeMaker, let's use it like one.** The **is** keyword is the first step. Once you know that you've got an Appliance reference that's pointing to a CoffeeMaker object, you can use **as** to downcast it. And that lets you use the CoffeeMaker class's methods and properties. And since CoffeeMaker inherits from Appliance, it still has its Appliance methods and properties.

```
if (powerConsumer is CoffeeMaker)
   CoffeeMaker javaJoe = powerConsumer as CoffeeMaker;
   javaJoe.MakeCoffee();
```

When downcasting fails, as returns null

So what happens if you try to use as to convert an Oven object into a CoffeeMaker? It returns null -and if you try to use it, .NET will cause your program to break.

Uh-oh—these don't match!

```
if (powerConsumer is CoffeeMaker) {
   Oven foodWarmer = powerConsumer as Oven;
   foodWarmer.Preheat();
```

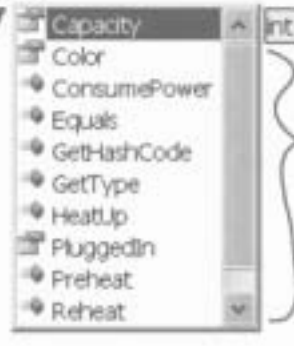
powerConsumer is NOT an Oven object. So when you try to downcast it with "as", the foodWarmer reference ends up set to null. And when you try to use a null reference, this happens...



Upcasting and downcasting work with interfaces, too

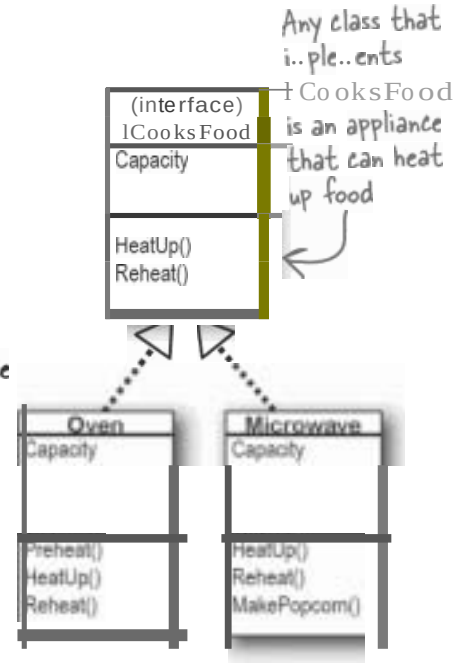
You already know that `is` and `as` work with interfaces. Well, you can do all of the upcasting and downcasting tricks. Let's add an ICooksFood interface for any class that can heat up food. And then we'll add a Microwave class—both Microwave and Oven implement the ICooksFood interface. Now there are three different ways that you can access an Oven object, and VS IntelliSense can help you figure out exactly what you can and can't do with each of them:

```
Oven misterToasty = new Oven();
misterToasty.
```

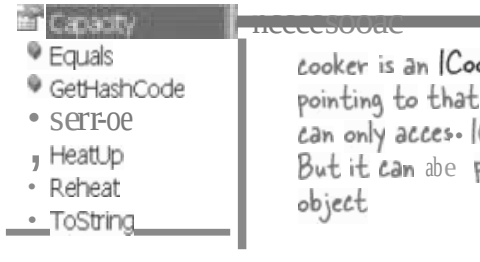


As soon as you type the dot, the IntelliSense window will pop up with a list of all of the members you can use.

misterToasty is an Oven reference pointing to an Oven object, so it can access all of the methods and properties... but it's the least general type, so you can only point it at Oven objects



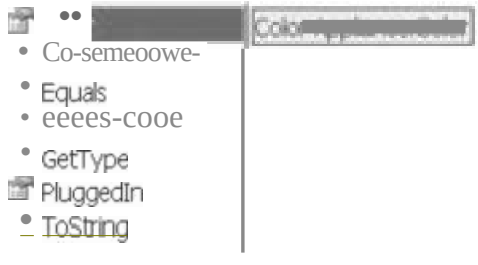

```
ICooksFood cooker;
if (misterToasty is ICooksFood)
    cooker = misterToasty as ICooksFood;
    cooker.
```



cooker is an ICooksFood reference pointing to that same Oven object. It can only access ICooksFood members. But it can also point to a Microwave object

```
Appliance powerConsumer;
if (misterToasty is Appliance)
    powerConsumer = misterToasty;
    powerConsumer.
```

powerConsumer is an Appliance reference. It only lets you get to the public fields, methods and properties in Appliance. You can also point it at a CoffeeMaker object if you want.



Three different references that **point** to **the same object can** access **different methods** and **properties, depending** on **the** reference's type,

there are no Dumb Questions

Q: So block up—you told me that I can always upcast but I can't always downcnl. Why?

A: Because the compiler can warn you if your upcast is wrong. The only time an upcast won't work is if you're trying to set an object equal to a case that it doesn't inherit from or an Interface that it doesn't implement. And the compiler can figure out immediately mat you didn't upcast properly, and Will give you an error.

On the otber hand, the compiler doesn't know how to check If you're downcasting *from* an object or interface rerereoce *to* a reference that's not valid. That's because it's perfectly legal to put any crass or interface name on the right-hand side of the `as` keyword. If the downcast is illegal, then the `as` steiement will jus! return `null`, And it's a good thing that the compiler ooesnt stop you from doing thal, because mere are plenty of times when you'd want to do lhat.

Q: Someone told me that an interface is like a ecneaet, but I don't really get why, What does that mean?

A: Yes, we've heard that too—a lot of people like to say that an interface IS like a contract. (That's a really common question on job interviews.) And it's true, to some extent. When you make your class implement an interface, you're telling the compiler that you promise to put certam methods into it. The compiler will hold you to that proense.

Butwe think that it's easier to remember how interfaces work If you think of an interface as a kind of checklist. The compiler runs through the checklist to make sure that you actually put all of the methods from the interface into your dass. If you cent, it'll bomb out and not let you compile.

Q: Whll if I wanttc put ı method body into my interface? Is that okay?

A: No, the compiler won't let you do that. An interface Isn't allowed to have any statements in it at all. Even though you use the colon operator to implement an Interface, n'e not the same thing as inheriting from a cress. Implementing an Interface doesn't add any behavior 10 your ctass at all, or make any changes to It. All it does is tell the compiler to make sure that your class has all of the methods that the interface says It shoukt bave.

Q: Then why would I want to use an interface? It seems like il'a just adding restrictions, without ictuilly chln ging my class I t all.

A: Because when your class implements an interface, then an interface reference can point to any instance of that dass. And that's really useful to you—it lets you create one reference type that can work a whole bunch of different kinds of objects.

Here's a quick example. A horse, an ox, a mule, and a steer can all pull a cart. But in OUr zoo simulator, Horse, Ox, Mule, and Steer woukt all be d,fferenl cresses. Let's say you had a cart-pulling ride in your zoo, and you wanted to create an array of any animal that could pull carts around. Uh-oh—you can't just create an *array* thai Will hold all of those. If they all inherited from the same base cess. then you could create an array of those. But It turns out that they don't. So what'll you do?

That's where interfaces come in handy. 'rcu can create an `IPuller` interface that has methods for pulling carts around. Now you coud declare your array like this:

```
IPuller[ ] pullerArray;
```

Now you can put a reference to any animal you wantm lhat array, as long as it implements the `IPuller` interface.

Q: Is there en easier way to implement interfaces? It's ı lot of typing!

A: Why yes, there is! The IDE gives you a very powerful shortcut that automatically implements an interface for you. Juststart typing your class.

```
public class
   Microwave    ICooksFood
   | |
```

Click on ICooksFood—you'll see a small bar appear underneath the "I" Hcver ever It and you'll see an icon appear underneath it.

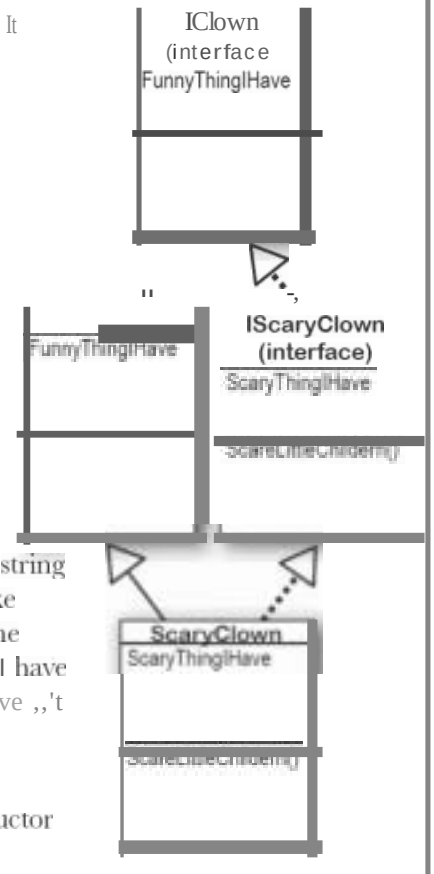


Click on icon and choose "Implement Interface 'ICooksFood'" from the menu. It'll automatically add any members u et you haven't implemented yet. Each one has a single `throws` statement in it—they'll cause your program to hatt, as a reminder In case you forget to implement one of them. (You'll learn about `throws` in chapter fO.)

An interface is like a checklist that tl.e compiler runs throngl. to make sure your class implemented a certain set of methods.



Extend the IClown Interface and use classes that Implement It



O Start with the IClown inl..rface from the last "Do This!" <Oil page 258

```
public interface IClown {
    string FunnyThinglHave { get; }
    void Honk();
```

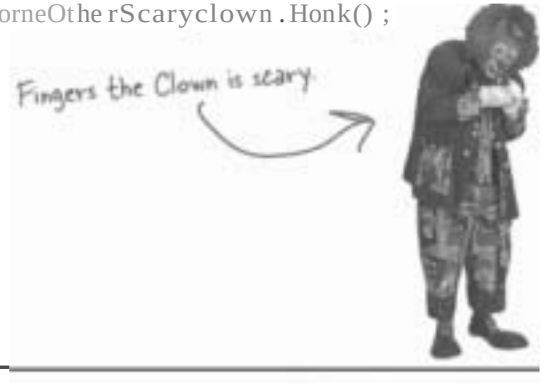
O ["I,'"d ICl own by creating a *w'"* inl..,face, IScaryClown, Ihat inherits from IClown. II should have an additional string prop..rty called ScaryThingIHave with a get accessor bill "" set accessor. and a void method called ScareLi t t leChildren().

3 Create these classes:

- A funnv clown class called FunnyFunny thar uses a private string variable 10 store a funny thing, and use a constructor that take a parameter called f unnyThi ngI Have and use it 10 set the I'ri,'ate field. T h,' Hon k () method ,hll"ld say, "Honk hllnk! I have a ' followed by the fiuujv thing it has. The FunnyThi ngI Have ,,'t accessor should return th.. same thing.
- A scary clown class "all,'(1 ScarySedry thai uses a private variable to store an integer that was passed to il by its constructor in a parameter call..d nu mberOfScar yThings. The SearyThi nq I Have get accessor sho,\lId r..turn a string consisting <or Ih,' number from tI"" 'o",lrul"lor followed by "spiders". The Honk () pops up a message box that says, "Boo! (;"td ıa!"

O Here's code for a button—but it's n"t working. Can you figure out how 10 fix it?

```
private void blttonl_Click(object sender, gventArca e) {
    scervscerv fingersTheClown = new scervscervt-crc shoes", 14);
    FunnyFunny some FunnyClown = fingersTheClown:
    lScaryClown someOtherScaryclown  someFunnyClown;
    sorneOther Scaryclown.Honk();
```



C O

You better get this one right... or else!



Extend the IClown otertace and use classes that implement i1.

```
public interface IClown {
    string FunnyThingIHave ( get; )
    void Honk();

public interface IScaryClown    IClown
    string ScaryThing1Have ( get;
    void ScareLittleChildren();

public class FunnyFunny : IClown
    public FunnyFunny(string funnyThing1Have)
        this.funnyThingIHave - tunnyThingIHave;

    private string funnyThingIHave;
    public string FunnyThingIHave {
        get I return "Honk honk! I have    + funnyThingIHave; )

    public void Honk() {
        MessageBox,Show(this,FunnyThingIHave);

public class ScaryScary :     n funn '  IScaryClown {
    public ScaryScary(string funnyThingIHave, int numberOfScaryThings)
        : base(funnyThing1Have) I
        this.numberOfScaryThings = numberOfScaryThings;

    private int numberOfScaryThings;
    public string ScaryThingIHave {
        get 1 return "I have " + numberOfSca,ryThrngs + .. spiders"; I

    pUblic vore ScareLlttleChlldren() {
        MessageBox Show("Boo' Gotcha!");

private vOld buttonl Clrck(obJect sender, EventArgs e) {
    ScaryScary tingersTheClown . new ScaryScary("big shoes", 14);
    FunnyFunny someFunnyClown - fingersTheClown;
    IScaryClown sorneOtherScaryclown = someFunnyClown as ScaryScary;
    someOtherScaryclown.Honk();
```

The Honk() ..ethod just uses this set accessor to display its ..essage—no need to have the same code twice.

You could have imple..ented the IClown method and property again, but why not just inherit from FunnyFunny?

Since Sca..ySca..y is a subclass of Fu yFunny and FunnyFunny implements IClown, ScaryScary imple ents IClown too.

You can set a FunnyFunny reference equal to a ScaryScary object because ScaryScary inherits from FunnyFunny But I- can't set any IScaryClown reference to just any clown, because I- don't know if that clown is scary. That's why you need to use the as keyword

There's More than Just public and private

We call a class's methods fields and properties its members. Any member can be marked with the public or private access modifier.

You already know how important the private keyword is, how you use it, and how it's different from public. C# has a name for them: they're called **access modifiers**. The name makes sense, because when you change an access modifier on a property, field, or method in a class—its members—or the entire class, you change the way other classes can access it. There are a few more access modifiers that you'll use, but we'll start with the ones you know:

- **public means that anyone can access it**
  When you mark a class or class member public, you're telling C# that any member or other class can access it. It's the least restrictive access modifier. And you've already seen how it can get you in trouble—only mark class members public if you have a reason. That's how you make sure your classes are well-encapsulated.

- **private means that only other members can access it**
  When you mark a class member private then it can only be accessed from other members inside that class or other instances of that class. And when you mark a class private, then it can only be used by other classes inside its namespace—unless that class **lives inside** another class, in which case it's only available to instances of its container class.

- **protected means public to subclasses, private to everyone else**
  You've already seen how a subclass can't access the private fields in its base class. It has to use the base keyword to get to the public members of the base object. Wouldn't it be convenient if the subclass could access those private fields? That's why you have the protected access modifier. Any class member marked protected can be accessed by any other member of its class, and any member of a subclass of that class.

- **internal means public only to other classes in an assembly**
  The built-in .NET Framework classes are assemblies—they're libraries of classes that are in your project's list of references. You can see a list of assemblies by right-clicking on "References" in the Solution Explorer and choosing "Add Reference..." when you create a new Windows Forms Application, the IDE automatically include the references you need, and when you build a Windows application. When you build an assembly, you can use the internal keyword to keep classes private to that assembly, so you can only expose the classes you want.

- **sealed says that this class can't be subclassed**
  There are some classes which you can't just "call" inherit from. A lot of the .NET Framework classes are like that—go ahead, try to make a class that inherits from String (that's the class whose IsEmptyOrNull() method you used in the last chapter.) What happens? The compiler won't let you build your code. It gives you the error, "cannot derive from sealed type 'string'". You can do that with your own classes—just add sealed after the access modifier.

Access Modifiers **change** scope

Let's take a closer look at the access modifiers, and how they affect the **scope** of the various class members. We've made two changes: the funnyThingIHave backing field is now `protected`, and we changed the ScareLittleChildren() method so that it uses the funnyThingIHave field.

Make these two changes to your own exercise solution. Then change the protected access modifier back to private and see what errors you get.

```
public interface IClown {
    string FunnyThingIHave { get; }
    void Honk();
}

public interface IScaryClown : IClown {
    string ScaryThingIHave { get; }
    void ScareLittleChildren();
}

public class FunnyFunny : IClown {
    public FunnyFunny(string funnyThingIHave) {
        this.funnyThingIHave = funnyThingIHave;
    }
    protected string funnyThingIHave;
    public string FunnyThingIHave {
        get { return "Honk honk! I have " + funnyThingIHave; }
    }

    public void Honk() {
        MessageBox.Show(this.FunnyThingIHave);
    }
}
```

The "this" keyword also changes the scope of a variable. It says to C#, "Look at the current instance of the class to find whatever I'm connected to—even if that matches a parameter or local variable."

This is a really common way to use "this", since the parameter and backing field have the same name. funnyThingIHave refers to the parameter, while this.funnyThingIHave is the backing field.

By adding "this", we told C# that we're talking about the backing field, not the parameter that has the same name.

We changed this to protected. Look and see how it affects the ScaryScaryScareLittleChildren() method.

When you use "this" with a property, it tells C# to execute the set or get accessor.

Access Modifiers Up Close



```
public class ScaryScary : FunnyFunny, IScaryClown {
    public ScaryScary(string funnyThingIHave,
                      int numberOfScaryThings)
        : base(funnyThingIHave) {
        this.numberOfScaryThings = numberOfScaryThings;
    }

    private int numberOfScaryThings;
    public string ScaryThinglHave {
        get { return "I have " + numberOfScaryThings + " spiders"; }
    }

    public void ScareLittleChildren() {
        MessageBox.Show("You can't have my "
                        + base.funnyThinqIHave);
    }

private void buttonl_Click(object sender, EventArgs e) {
    ScaryScary fingerstheClown = new ScaryScary("big shoes", 14);
    FunnyFunny someFunnyClown = fingersTheClown;
    IScaryClown someOtherScaryclown = someFunnyClown as ScaryScary;
    someOtherScaryclown.Honk();
}
```

numberOfScaryThings is private, which is typical of a backing field. So only another instance of ScaryScary would be able to see it.

The protected keyword tells C# to make something private to everyone except instances of a ...bclass

The "base" keyword tells C# to use the value from the base class. That's another way to change scope.

If we'd left funnyThinglHave private, this would cause the compiler to give a <-- error. But when we changed it to protected, that made it visible to any ...bclass of FunnyFunny

Since this button click event handler is not part of FunnyFunny and ScaryScary, it can't access the protected funnyThinglHave field

It's outside of both classes, so the statements inside it only have access to the public members of any FunnyFunny or ScaryScary objects.

there are no Dumb Questions

Q: Why would Iwant to **use an interface instead** of just writing all 01 the methods I need directly into my **class?**

A: You might end up with a lot 01different classes as youwrite more and more complex proqrams Interfaces let yougroup those cresses by the kind 01 work they do They help you be sure that every class there going 10 do a certain kind of work does It using the same methods. The class can do the work however it needs to and, because of the interlace, you dentneee 10 worry abcct how it does it just to get the job oone

Here's an example: you can have a truck class and a sailboat classthat implement `ICarryPassenger`. Say the `ICarryPassenger` Interlace snputates that any ctass jnat implements it has to have a `ConsumeEnergy ()` method Your program could use them both to carry passengers even though the sailboat caass `ConsumeEnergy ()` melhod uses wind power and the truck class's method uses diesel fuel.

Imagine if you didn't have the `ICarryPassenger` interlace Then it would be tough to tell your program which vehicles could carry people and which couldn1 You wouldhave to look through each class that your program might use and figure out whether or not there was a method for carrying people from one place to another. Then you'd have to call each of the vehicles your program was going to use with whatever method was defined for carrying passengers And erce there's no standard interface, they could be named all sorts of things or buried inside other methods You can see how thal'll getcontusing pretty fast.

Q: Why do I need to **use a** property? **Can't I** just include a field?

A: Good question. An interface only defines the way a class should do a specific kind of job. I1's not an object by ,tsel , so you can't instantiate it and it can't store information. Il you added a field that was just a variable declaration, then C# would have to store that data somewhere—and an interlace can't store data by itself

Q: **What's** the **difference** between **a** regUar object relerenee and an interlace **reference?**

A: You already know how a regular, everyday object relerence works If you create a instance of `Skateboard` called `VertBoard`, and then a new rerererce to it called `HalfPipeBoard`, they both point to the same thing But if `Skateocal'd` implements the interlace `1StreetTricks` and you create an interface reference to `Skateboard` called `Street.Board`. it Wll only know the methods in the `Skateboard` crass that are also in the `1StreetTricks` interface.

All three references are actually pointing to the same object. If you call the object using the `HalfPipeBoard` or `VertBoard` references, you'll be able to access any method or property in the object. If you call it using the `StreetBoard` reference, you'll only have access to the methods and properties in the interface

Q: Thtn why would I **ever** want 10 **use** an interlace reler. nea if it limit, **what I** can do with **the object?**

A: Interface references give you a way of working with a bunch of different kinds 01 objects that do the same thing You can create an array using the interlace retererce type that will let you pass information to and from the methods In `Icarrypassenqer` whether your working With a truck object, a horse object, a unicycle ceect. or a car object The way each of mcee objects do the job IS probably a little different, but with interface references, you know that they all have the same methods that take the same parameters and have the same return types. So, you can call them and pass information to them in exactly the same way

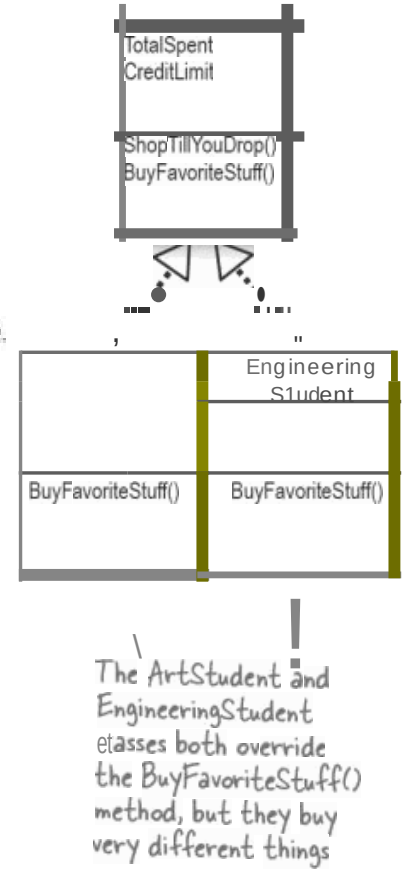
Q: Why would I make something protected **instead** 01privi te or public?

A: Because it helps you encapsulate your casses better. There are a lot of times that a subclass needs access to some internal part of its base class. For example, If you need to cverrda a property, it's pretty common to use the backing field in the base class In the get accessor, so that rtreturns some sort of variation of it. But when you bUIld cresses, you should only make something public Il you have a reason to do it. Using the protected access mccrer lets you expose n only to the subclass that needs it, and keep it pw ate to everyone else

Interface rejerences only **know** about **the** melllOd, and **properties that are defined** in **the interface.**

SOllie classes should never be instantiated

Rememl)er "ur zoo simulator class hierarchy? You'll definitely end lip instantiating a bunch "r hippos, dogs and li"lt, _But what about the Canine and Feline classes? How ab(nn the Animal class? [Iturns »rn that there are some classes that jU'l d""'t need t" be instantiated... ,,,,d, ill fact, don't make any sens(if they are. Here's all example.



Let's start with a basic class for a student shopping at the student bookstore.

```
public class Shopper {
    public void ShopTillYouDrop()
        while (TotalSpent < CreditLimit)
            BuyFavoriteStuff();

    public virtual void BuyFavoriteStuff ()
        // No implementation here - we don't know
        // what our student likes to buy!
```

Here's the ArtStudent class --;1subclasses Silt']'pn:

```
pUblic class ArtStud"nt : Shopper {
    public override void BuyFavoriteStuff () {
        BuyArtSupplies();
        BuyBlackTurtlenecks();
        BuyDepressingMusic();
```

.And the EngineeringStudent class als(inherits li" nn Shopper:

```
public class EngineeringStudent : Shopper {
    public override void BuyFavoriteStuff ()
        BuyPencils();
        BUyGraphingCalculator(l;
        BuyPocketProtector():
```

So what happens when you instantiate Shopper? Does it ever make sense to do it?

An abstract class is like a cross betweell a class and an illterface

Suppose you need something like a" interface, that require classes 1" implement certain methods and properties. 11," you need I" il1dlld,' ",m,· code in that interface, so thar certain methods don't have (" be implemented in each inheriting class. What you want is an **abstract class**. You get the features of an interface, bill you can write code in it like a w'rlllal class.

A method that has a declaration but no statements or method body is called an **abstract method**. Inheriting classes must implement all abstract methods, just like when they inherit from an interface.

- *An* abstract class is like a **normal** class

  You define an abstract class jll'l Iik,' a 11"rmal one. It has fields and methods, ""d vuu can inherit fr'In other classes, too, exactlv like with a u"n n,,1 class. There' almost nothing new 10 learn l,,'n., because you already know everything that an abstract class d,)es!

Only abstract classes can have abstract methods. If you put an abstract ..ethod into a class, then you'll have to mark that class abstract or it won't compile. You'll learn more about how to mark a class abstract in a mi... te.

- An abstract class is like an interface

  When you create a class that impl"""llIS an interface, you agree ,,, implement all of the properties alld methods defined ililhal interface. An all'lr'I<'1 class works the same way it can include declarations "l' propertie and methods that, just like in an interface, must be implemented by inheriting classes.

The opposite of abstract is **concrete**. A concrete method is one that has a body, and all the classes you've been working with so far are concrete classes.

- **But an** eestrect **class** eee't b. ins'.a,;.,.d

  The biggest difference between an **abstract** class and a **concrele** class i, that you can't use nev [0 create an instance of an abstract class. [l' you d", C# will give you an error "h"11 you try (0 ,'olll)'il,' your code.

```
Cannot create an instance of the
abstract class or interface 'MyClass'
```

This error is because you have abstract methods without any code! The compiler won't let you instantiate a class with missing code, just like it wouldn't let you instantiate an interface.



Wait what? A eless thot I con', instantiate?
Why would I even wont something like that?

Because you want to provide some code, but still require that subclasses fill in the rest of the code.

Sometimes *bad* ***things happen*** wlx-n you create objects Ih"l , ho uld never b." created. Th,' clas at the tc'T' ,,1' vour class diagram usuallv has some fields Iha l ;1 expects its subclasses (o set. An Animal class may have a calculation thai depends ,n,,, boolean called IHasTail or Vertebrate, bUI there's no way for it 10set lhal itself.

Here's a class that the Objectville Astrophysics Club uses to send thei.. rockets to diffe..ent planets

Here's nil example..

```
pub1ic class PlanetMission {
  pUblic long RocketFuelPerMile;
  public long RocketSpeedMPH;
  public int MlleSToPlanet;
  public long UnitsOfFuelNeeded() {
    return MilesToPlanet * RocketFuelPerMile;

  public int TimeNeeded() {
    return MilesToPlanet / RocketSpeedMPH;

  public string FuelNeeded(1 {
    return "You-Ll need
      + MilesToPlanet * RocketFuelPerMile
      + " units of fuel to get there. It'll take "
      + TimeNeededO + " hours.";
```

It doesn't make sense to set these fields in the base c!:ss, because ..... don't know what "ocket or P!:net we" be using

The astrophysicists have two missions—one to Ma..., and one to Venus.

```
public class Venus: PlanetMission
    public Venus 0 {
        MilesToPlanet = 40000000;
        RocketFuelPerMile .. 100000;
        RocketSpeedMPH = 25000;
```

```
public class Mars  PlanetMission
    public Mars 0 {
        MilesToPlanet .. 75000000;
        RocketFuelPerMile .. 100000,
        RocketSpeedMPH = 25000;
    ,
```

The constructors for the Mars and Venus subclasse• •et the three fields they inherited from Planet. But those fields won't get set if you instantiate Planet directly. So what happens when FuelNeeded() tries to use the..?

```
private void buttonl_Click(object 5, EventArgs e) {
  MarsMission mars " new Mars();
  MessageBox.Show(mars,FueINeeded(» ;

private void button2_Click(object S, EventArgs e) {
  VenusMission venus = new Venus();
  MessageBox.Show(venus,FuelNeeded() ;

private void button3_Click(object s, EventArgs e) {
  PlanetMission planet = new PlanetMission();
  MessageBox.Show(planet.FuelNeeded());
```



Before you flip the page, try to figure out what will happen when the user clicks the third button•••

SOllie classes should **never** be **instantiated**

['he problems all start when you create an instance of the PlanetMission class. It: Fuel Needed () l1l"lh"d expect-lh.· fields 10 be set bv the subclass. IIIH when they aren't, they get Ih,'i, default values - zero.-And when C# tries I" divide a il" ",I,,'r hv zero..

```
private void button3_Click(object s, EventArgs e) {
  PlanetMission planet = new PlanetMission();
  MessageBox.Show(planet.FuelNeeded());
}
```

The PlanetMission class wasn't written to be instantiated. We were only supposed to inherit from it. But we did, and that's where the problems started.

When the FuelNeeded() method tried to divide by RocketSpeedMPH, it was zero. And when you divide by zero, this happens.



Solutioo: use ao abstraot olass

Adding the `abstract` keyword to the class declaration tells C# this is an abst..act class, and can't be instantiated

" 'h("n you mark a class `abstract`. C# "-"n'l let you "'Til.· code In instantiate il. It's a lot like an interface—it acts like a template for the clas es Ihal inherit from it.

N"... C# will refuse to compile our program until we remove the line that creates an instance of PlanetMi..ion

```
public long UnitsOfFuelNeeded() {
  return MilesToPlanet * RocketFuelPerMile;
}

// the rest of the class is defined here
}
```

An abstract Il\ethod doesll't have a body

Y<>ll know how an interface only has declarations fi,r methods and properties, but it doesn't actually have any method bodies? That's because every method ill an interface is '1Il **abstract** m..thud, You ,'a" have abstract " "'Ihud, ill all abstract class, too— but you need to explicitly mark them with the **abstract** keyword, because an abstract class can also have normal methods too.

Every method in an interface **is** automatically abstract, **so** you don't need to **use** the abstract **keyword** in an interface, **just** in an **abstract** class. Only abstract classes can have abstract methods••• but they can have concrete methods too.

```
public abstract class PlanetMission {

    public abstract void setMissionInfo(
        int MilesToPlanet, int RocketFuelPerMile,
        long RocketspeedMPH);

    // the rest of the class..
```

This abstract method is just like an interface—it doesn't have a body, but any class that inherits from PlanetMission has to implement the SetMissionInfo() method or the program won't compile.

If we add thalllH'du,d in and try '0 build the program, the II>1'; give us illl erro:

```
'VenusMission' does not implement inherited abstract
member 'PlanetMission.SetMissionInfo(long, int, int)'
```

So let's implement it! Once we do, the ,'IT '1 will go away.

```
public class Venus : PlanetMission {
    public Venus() {
        SetMissinInfo(40000000, 100000, 25000);

    SetMassfonInfo Lirrt milesToPlanet, Long rocker.guelperttt Ie, int rocketSpeedMPH) {
        this.MllesToPlanet = mllesToPlanet,
        this.RocketFuelPerMile - rocketFuelPerMile;
        this.RocketSpeedMPH - rocketSpeedMPH;
```

It really sucks to be an abstract methad. You don't have a body.



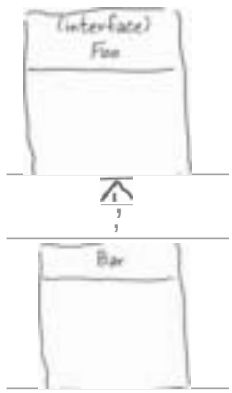
Sharpen your penCil

Here's your chance to demonstrate your a,ti'tic abilities. On the left you'll find sets 01 class and interface declarations. your job is to draw the associated class diagrams on the ,ight. We did the first one fo, you, Don't forget to u,e a dashed line 10' implementing an interface and a solid line tor inheriting Irom a cress.

Given:

What's the Picture ?

1)
```
public interrace roc {
public class Bar : foo
```



2)
```
public interface Vinn
public abstract class Vout   Vinn {
```

2)

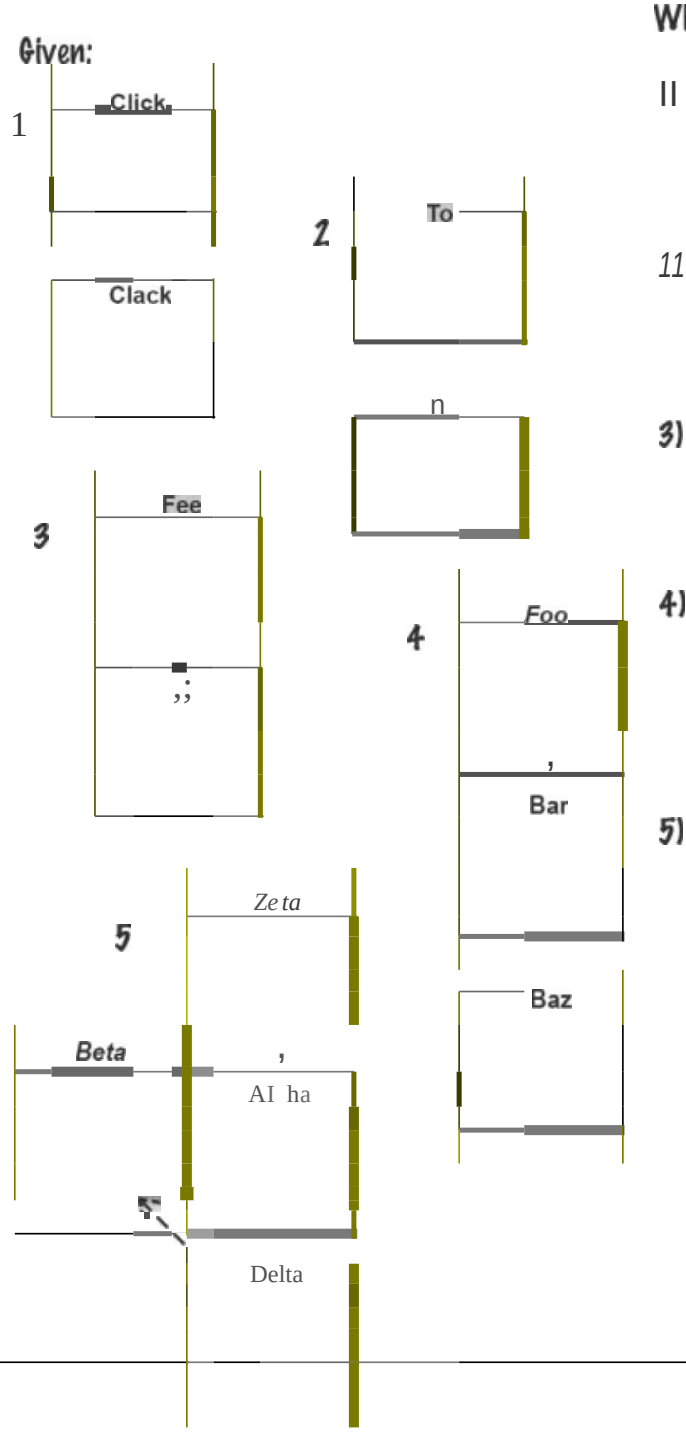
3)
```
pUblic abstract class MUffie : Whuffie
public class Fluffie : Miffie
public interface Whuffie {
```

4)
```
public cla55 Zoop
public class Boop   Zoop
public class Goop   Boop
```

5)
```
public class Gamma : Delta, Epsilon
public interface Epsilon {
public interface Beta (
public class Alpha   Gamma, Beta
public class Delta
```

5)

On the left you'll find sets of class diagram,. Your job is to turn the,e into valid C# declarations. We did num be r 1 for you.



What's the Declaration ?

1) public class Click { }

public cla.. Clack : Click { }

11

3)

4)

5)

KEY

T extends

implemenls

Clack class

Clack interface

Clack abstract class

Fireside Chats



Tonight's talk: **An abstract** class and **an interface butt heads over the pressing question,** "Who's **more important?"**

Abstract mass:

I lhi"k it's obvious who's more illll'on"nl between the two of us. Programmer need me 1" get rlu-ir jobs done. Let's face it. You dun't "\"11 come close.

You can·t really lhink you're ",on' impon anl than me. You don't even use n'a) inheritance you only get implemented.

Better? You're nuts. Im much more-flexible rh.m you. I can have abstract methods "r concrete oil"'. I can "\-"11 have virtual methods if I want. Sure, I can't be in'lalll;"1<'tl bUI then, neither can you. And I ,'an do pretty ",udt anything else a regular class does.

Interface:

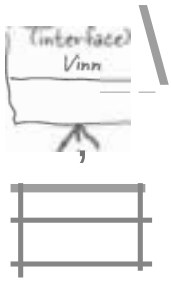
Nice. This oughta be good.

Great, here we go again. Interfaces d"u'l use real inheritance. Interface only implement. That's just plain ignorant. l"'I'I'I "'nlati,," is as good as inheritanc', in fact it's better!

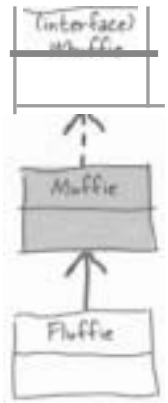
Yeah? What if you want a class that inherits from you *fmd* vnur buddy? **You** can't inh..ril **from two class..s.** You have t" choose which class to inherit fr<"n. ,\lld that's just plan r"d,'! There's "" li",it 10d,,' nllml,,'r of interfaces a class can implement. Talk about Il"xihl,'! With me. a progr.unnu-r can m.rke a class do anyth ing.

Sharpen your penCil Solution

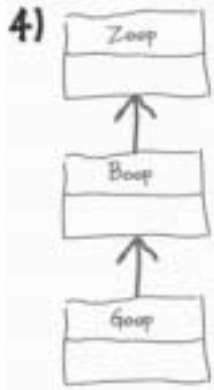
2)



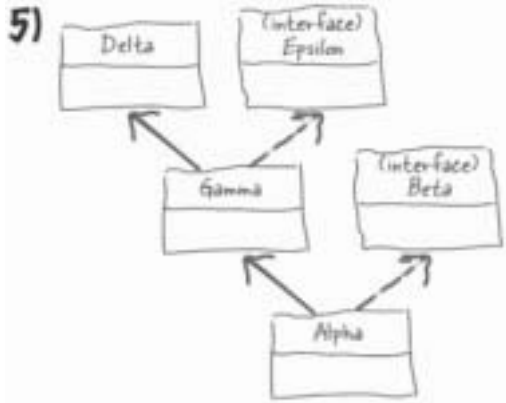
3)



4)



5)



What's the Picture 1

Abstract Class:

You might be overstating your power a little bit.

That's exactly the kind ur drivel I'd expect from ;u, interface. Code is extremely important! It's what ıı ı;·kes y(" 'r programs ,nn,

Really? I donht Ihat pnogrammers always care what's intln-ir properties and methods.

Yeah, sure, tell a coder he can't code.

Interface:

You think that just becaus‹ you can contain code, you're the greatest thing since sliced bread. BUI you call'l change the fact that a program can only inherit from one class ur a time. S", you're a liul,> limin-d. Su"" I can't illdud,' anv code. But really, code is overrated.

Nine times out of I,·U. a programmer wants I" make ,ur.· ;ıı\ object has certain properties and methods, but doesn't really care how they're illll'l" rl" 'lll' "(l.

Okay, ,ur,·. Lvr-ntuailv. But think about huw manv times you'v.. seen a programmer write a metho‹1 that take all ohj"'-I Ihalju" "",'ds 10 have a certain method, and il do,""'I really matter right at Ihal very nu"'ıent exactly how lh.' method's huilt,.J"'1 that it's there. S" bang! The progranuuvr just needs I" write an interface. Problem solved!

What***ever***!

---

2)
```
public abstract class Tor { }
public class Tip : Tor { }
```

3)
```
r blic abstract class Fee { }
r blic abstract class Fi : Fee { }
```

4)
```
r"blit interface Foo { }
p blit class Bar : Foo { }
p blit class Baz : Bar { }
```

5)
```
public interface Zeta { }
public class Alpha : Zeta { }
public interface Beta { }
public class Delta : Alpha, Beta { }
```

Delta inherits from Alpha and implements Beta

What's **the Declaration ?**

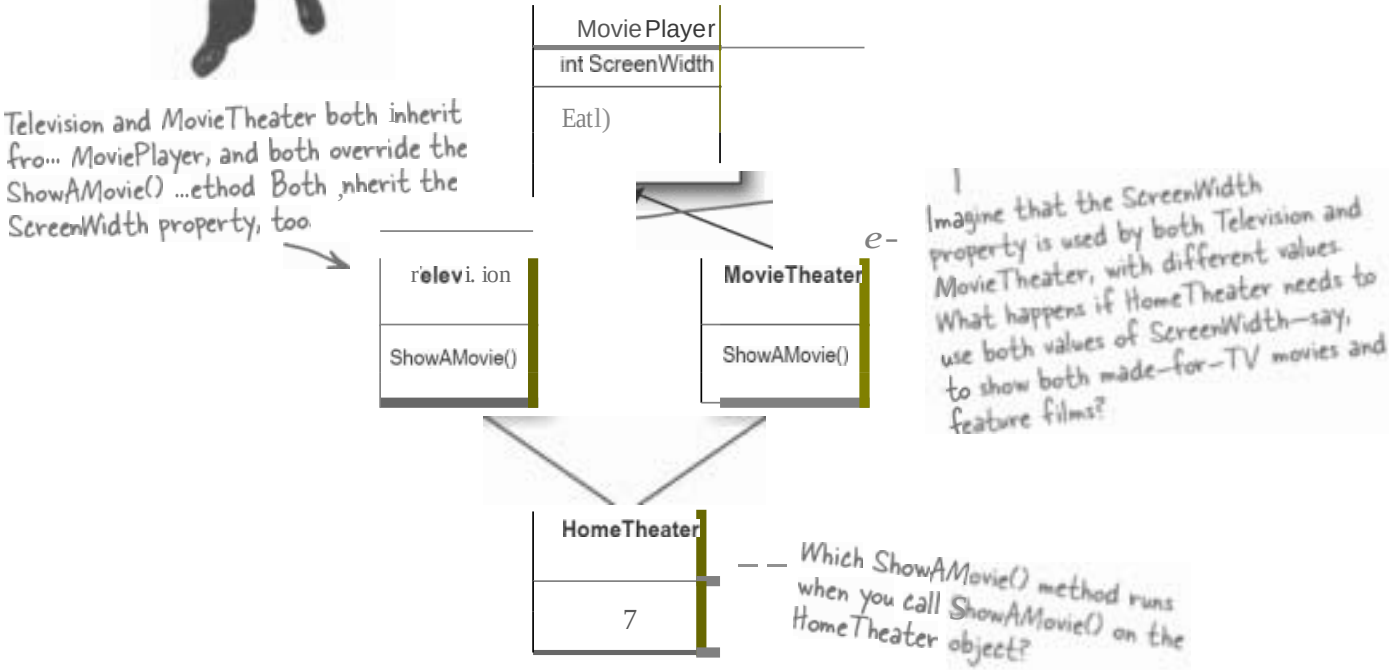
rm still hung up on not being ebte te inherit from two clessee. If I can't inherit from more than one clcss , so I hcve te use rter-fcces. That's a pretty big limitation of C# , right?



It's not a limitation, It's a protection.

If C# let you iuherit from more than one base class, il would "IKn up a whole can of worms. \Vhe1 a language lets one sublcass inherit from two base classes, it's "all,'d Dlultipl.. inh..**ritanc**... .\nd by giving you interface instead, C# saves you from a big fat mess tha' we like to call...

The Deadly Diamond of Death!



Avoid aMblguity l

,\ language that allows the Deadly Diamond of Death can lead ,n some pretty ugly situations, because you no-d special rules to deal withthis kind of ambiguous situation... which means extra work for you when you're building y<>tlr program! C# p"'tects you Jr.'Ill having t" d..al with this by giving you interfaces. If `Television` and `MovieTheater` an' interfaces instead of classes, th"ll the same `ShowAMovie()` method can satisfy both of them. ,\ll lhe interface cares about is that there's some nv-thod called `ShowAMovie()`.



Pool Puzzle

Your *job* is to take code snippets from the pool and place them into the blank lines in the code and output. You may use the same snippet more than once, and you won't need to use all the snippets. Your *goal* is to make a set of classes that will compile and run and produce the output listed.

```
public                Nose

    string Face { get;


pUblic abstract class          . .. '. .      ..{
    public virtual int Ear(J
    {
        return 7;

    pUblic Picasso(string face)
    '
                  _ face;

    public virtual string Face
          '

    string face;


public class                  .   .{
    public Clowns() : base("Clowns") ( }
```

```
publ ic class          •...•..... .....{
    public Acts()    base("Acts") {
    public override              {
        return 5;

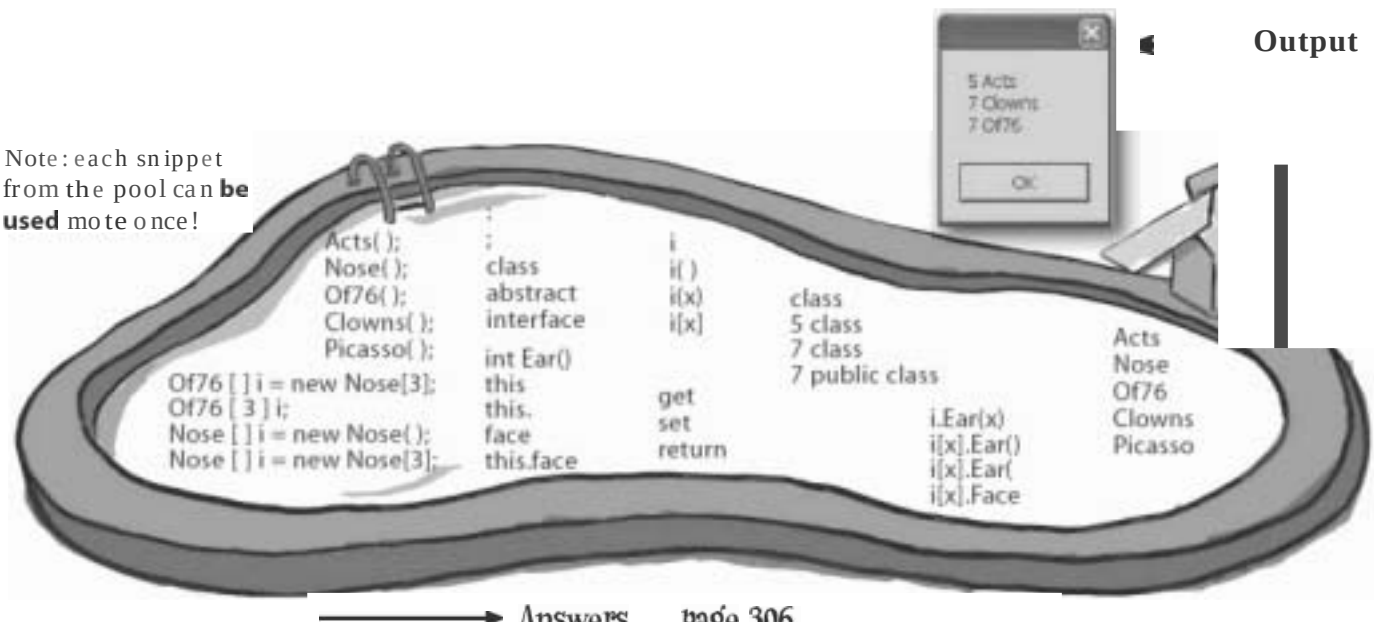

publ ic class              .
    public override string Face (
       get I return "Of76";

    public static void Main(string[] args)
        string result • "";
        Nose[] i = new Nose[]):
        i tO)    new Acts!);
        i 11)    new Clowns()
        i(2)     new 0f76();
        for lint x - 0; x < 3; x++J
            result +-                +
                        +                 + "\n";
        '
        MessageBox.Show(resultj;
```

Here's the entry point—this is a complete C# program.

Output

Note: each snippet from the pool can be used more than once!



Answers on page 306.



Okay, I think r ye got a pret ty good handle on objects now!

The idea that you could combine your data and your code into classes and objects was a revolutionary one when it was first introduced—but that's how you've been building all your C# programs so far, so you can think of it as just plain programming.

You're an object oriented programmer.

There's a name t"r what you've been doing. It's called **object** oripntpd **programming**, or (J{)1' Before languages like C# ,-ame along, people didn't use objects and methods wh..ll writing their code. They j"'l usecI f'",'Iio", (which is what they called methods ill it non-( )( )P pn)gram) ,h,ll were all in one place as if eac'h program were just one big " ali,' class Ih,lI only had "ali,' methods. It made il a 10' harder to creat(programs that modeled the problems they were solving. Luckily, you'll never have 10 write programs without ( )( )P. because ii" a core part ..r C#.

The four principles of object oriented programming

When programmers talk about ( )( )P. Ihey're referring 10 fi,ur importaru principles. They , hould seem very familiar 10 you by u,," because you've been working with every one of Ihem. Y'll'll recogniz(the first three I'ri,ciples jU'1 from rh..ir names: **inheritance**, **abstraction**, and **encapsulation**, The last one's called **polymorphism** h " "Hul, a littl(odd, but i, turns "mlh,lI you already know all about i, too.

Thi just ..eans having one class or interface that inherits from anothe..

[nkeritance

Encapsulation means creating an object that keeps track of its state internally using private fields, and uses public properties and methods to let other classes work with only the part of the internal data that they need to see.

Encapsulation

Abstraction

You're using abstraction when you create a class model that starts with more general—or abstract—classes, and then has more specific classes that inherit from it

Polymorpbism

The word "polymorphism" literally means "many forms". Can you think of a time when an object has taken on many forms in your code?

PolYllIorphislll means that one object can take many different forllls

Any time vou use a mockingbird in place "I' an animal or aged \'"n"olll dll-ddar in a recipe that j",t calls for cheese you're using polyolorphi".n. That's what you're doing any time you upcast or downcast. It's taking an ohi,-,'I and use i, in a Ilw,h"d or a ,Ialt'ru'-Ill Ih"l expects something else

Keep your eyes ope" for polyl\torphisl\t in the next exercise!

You're ab()lll 10dll a really big exercise— lh,' biggest oil" you\." seen so far—and you'll be using a]01 <If polymorphism ill il. So keep your eye Op'll for it. Here's a li'l of 1,)„1' typical ways lhal yo,,'l1 use p(,lymorphi"", We gave you an example ,,1' ,-",'h ,,1' them (you won't see these particular line in the exercise, though). As soon as you see similar code ill the code that you write for the exercise, **check** i1 ofT **the** Iisr.

You're using rolymo"rbism **when** you **take** an **instance** *01* **one class** anJ **use** it **in a statement or** a **method that expects** a **different type, like** a **pal"ent class or an interface that the class** implements.

D Taking any reference variable that use one class „,,<l setting il .'qllal I" an instance of a different class.

```
NectarStinger bertha - new NectarStinger();
INectarCollector gatherer - bertha;
```

D Upcasting by using a subclass in a statemeI'I or method that expects its base class

```
spot = new Dog();
zooKeeper.FeedAnAnimal(spot);
```

If FeedAnAnimal() expects an Animal object, and Dog inherits from Animal, then you can pass Dog to FeedAnAnimal().

D Creating a reference variable whose type is 'Ill interface 'Illd pointing il III an object that implements llral i"l..rf"ce.

```
IStingPatrol defender - new StingPatrol();
```

This is upcasting, too!

D Downcasting using the as keyword.

```
void MdintainTheHive(rworker worker) {
    if (worker is HiveMdintainer)
        HiveMaintainer maintainer  worker as HiveMaintainer;
```

The MaintainTheHive() method takes any IWorker as a parameter. It uses as to point a HiveMaintainer reference to the worker.

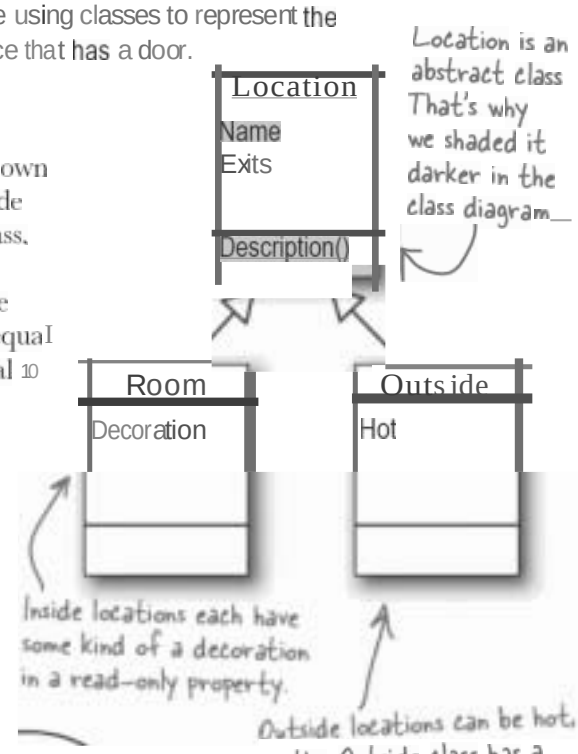


Long Exercise

Let's build a house! Create a model of a house using classes to represent the looms and locations, and an interface for any place that has a door.

1 Start with this class model

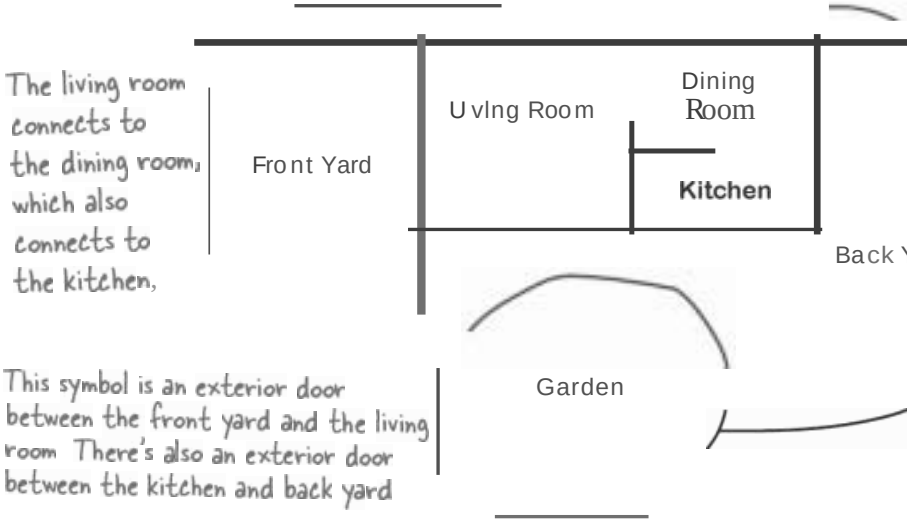
Every room or location in your house will be represented by its own object. T h,' interior rooms all inherit from Room, and the outside places i"h,>ril fr‹,m Outside, which subclass rhe same base class, Location, It has two fields: Name is the name or the locali"" ("Kitchen"), and Exits i, an «nuv "r Location "bjects that the current lo:"rion connects In. So dinIngRoom. Name will be equal to "Dining Room". and diningRoom. Exits will be equal 10 the array { LivingRoom, Kitchen),



➔ **Create** a **Windows Application** project and add Locati..n, R....m **and Outside classes** ,.. it.

2 You'll need the blueprint for the house

This I",u,.: has three rooms, a front yard, and a garden. There are two doors: the front door connects the living H.'tl! to rl,,' f"nll yard, .",d the b,,,'k door connects the kin-hen to the ha('k yard.



3 Use the IHasExteriorDoor interface for rooms with an exterior door

There are two exterior d,.ors in the house, the front d,. ·r and rl,,' back d,. ır. Every location rh"l has 0"" (the front yard, back yard, living "., 'lH, and kitchen) , hould implement IHds Exterior Door. Th,' Door Description read-only property ,">lnai", a d""Tipri"11 of the door (tl", front door is "an oak door with a brass knob", lhl" back door j, "a screen door"). The DoorLocation field contains a reference to the Location where the d,".r leads (kitchen).



Here's the Location class

To get you started, here's the Location class:

```
public abstract class Location {
    public Location(string name)
        this.name - name;
    )
    public Locationll Exits;

    privat.e string name;
    public string Name {
        get { retum name;
    )

    pUblic virtual string Description {
        get {
            string descript.ion _ "You're standing in the " + name
                + "  You see exits to the following places: ";
            for (lnt i - 0; i < Exits.Length; i++) {
                description += " " + Exits[i].Name;
                if (i !- Exits.Length - 1)
                    description += ",";

            description += ".";
            return description;
```

The constructor sets the name field, which is the read-only Name property's backing field.

Description is a virtual ..ethod You'll need to override it.

The public Exits field is an array of Location references that keeps track of all of the other places that this location connects to.

The Room clas w,ll override and extend Description to add the decoration, and Outside will add the temperature

The Description property ret.rns a string that describes the room, incl..ding the na..e and a list of all of the locations it connects to (which it finds in the Exits[] field). Its subclasses will -eed to change the description slightly, so they'll override it

Remember, Location is an abstract class—you can inherit from it and declare reference variables of type Location, but you can't instantiate it.

Create the classes

First create the Room and Outside classes based "ll lhe class model. Th("n create two more classes: OutsideWi thDoor. "hidl inherits from Ou tside ,md impl"llllt"'!' IHa sExte r iorDoor. and RoomWi thDoor, .. hid , subclasse Room.lIld implements IHasExteriorDoor.

Here are the class declarations to give you a leg up:

Get the classes started now—we"" give r- more details about the.. on the next page.

```
public class OutsideWithDoor : Outside, IHasExteriorDoor
{
    // The read-only DoorLocation property goes here
    // The DoorDescription property goes here


public class RoomWithDoor : Room, IHasExteriorDoor
{
    // The read-only DoorLocation property goes here
    // The DoorDescription property goes here
```

This one's going to be a pretty big exercise••• but we promise it's a lot of funl And you'll definitely know this stuff once you get through it.

We're not done yet—flip the page!

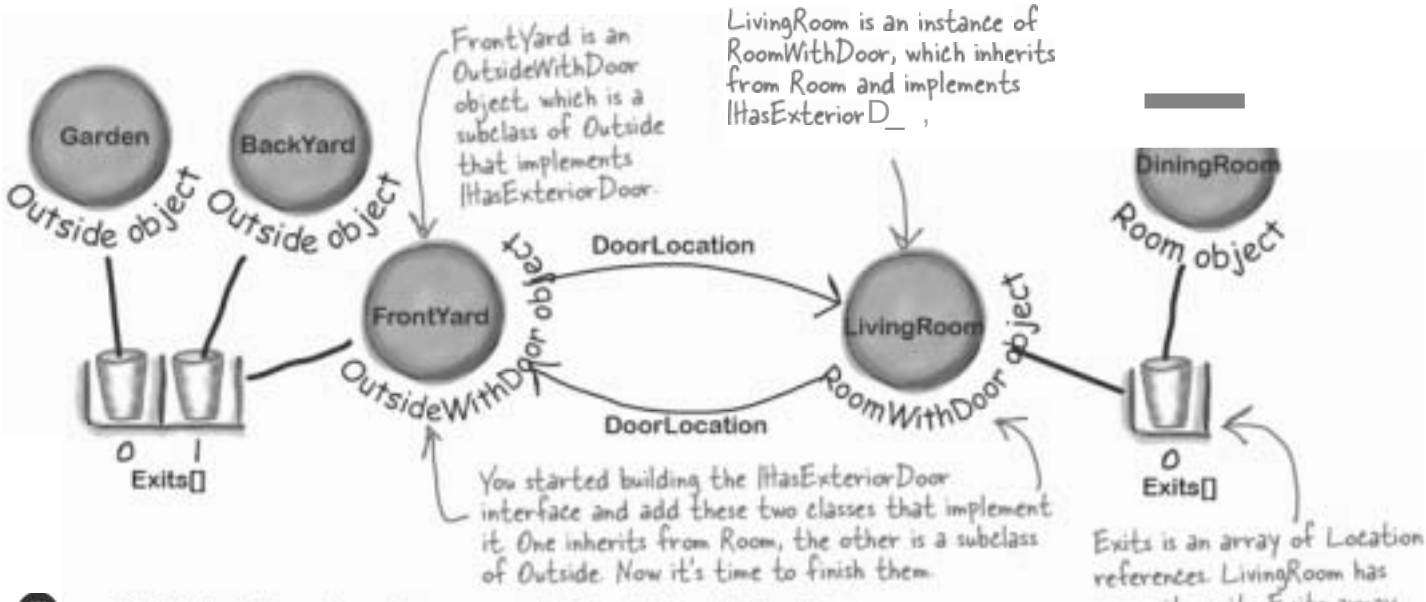


Long Exercise (continued)

Now that you've got the class model, you can create the objects for all 01 the parts of the house, and add a tom to explore it.

How *your* house objects work

Here's the architecture for two of your objertx, frontYard and diningRoom, Sinc e each „1' them has a door. they 1.'1h need t" Dc instances „1' a class that implements IHasExteriorDoor. The DoorLOCdtion property keep a reference tKJ the I"cation oil the othe1' ,id., "f the door.



7 Finish building the classes, and instantiate their instances

You've got all the classes, now it's time to finish them and build your objects.

★ You'll need to make sure that the constructor for the Outside class sets the read-only Hot property and overrides the Description property to add the text "It's very hot here." if Hot is true. It's hot in the back yard but not the front yard or garden.

- The constructo'r f,·r Room needs t" set the Decoratiom, and sho,\lId,werride the I),·"„-it,ti,'" property u> add, "Yo>U " .,,. *(the decoration)* here." '1'1,,' living roo"11 has an antique carpet, the dining roo"11 has a crystal chandelier, and the kitchen has stainless steel appliances and a screen d'" ,r Ih.,t leads to the back yard.
- Your form needs 1" create each or the objects and keep a reference to each one. So add a 111,·th"d to the form "alled CreateObjects() and call it from the form's constructor.
- Instantiate each "f the >1bjects f,"-the six locations in th,· h,use. Here's one of lh,se lines:

```
RoomWithDoor livingRoom = new RoomWithDoor("Living Room",
    "an antique carpet", "an oak door with a brass knob");
```

Every location will have its own field in the form class

★ Your CreateObjects() method needs 10 populate the Exits[] field in each "bi,·,'t:

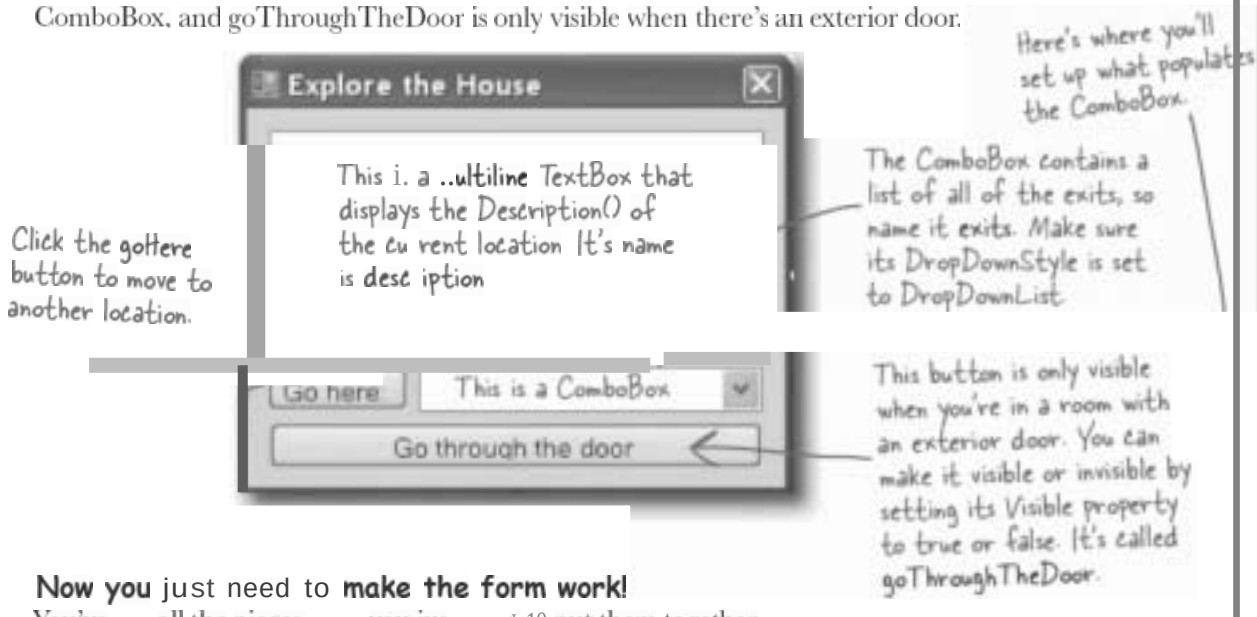
```
frontYard.Exits = new Location[] {backYard, garden};
```

Exits is an array of Location references, so this line creates one and that has two references in it

These are curly brackets. Anything else will cause an error

Build a form to explore the house

Build a simple form to let you explore the house. It'll have a big mull ili,,,' textbox "aU>d description 10show the description of the "UITt'n1room. .\ ComboBox called rxh lists all or the exits in the current ""'111, It's got two bunons: goHere moves '" the room selec-1,·d in the ComboBox, and goThroughTheDoor is only visible when there's an exterior door.



Now you just need to make the form work!

You've go all the pieces, now you jus nee<l 10 put them together.

- You'll need a field ill your form called currentLoc:"i"" to keep track of your current location.
- .\<ld a MoveToANewLocation() method Ihal has a Location as its parameter. Thi, nie,h"d sh""ld first set ,'nlT"!llLocat iOl1 to the new locat ion. Then it'll clear the combo box using ii, Items.Clear() method, and then add the name of each location in the Exits[J «rrav using the combo box's Items.A\ddOmethod. Finally, n'''' the combo box s< il displays the firs item in the li" by setting ii, SelectedIndex property to zero.
- S,'llh,' textbox so Iha, i, has the description or the current location.
- Use ,h,' **is** keyword to check ir ,Ine curre-nt location has a door. If il does make the "Go through the door" button visible using its Visible property. If not, make il invisible.
- If the "Go here:" button is clicked, move to the location selected in the combo box.
- If the "Go through the door" button is clicked, move to the location that the door coonects to.

Hint: When you choose an item in the combo box, its selected index in the combo box will be the same as the index of the corresponding location in the Exits[] array.

Another hint: Your form's currentLocation field is a Location reference. So even though it's pointing to an object that implements IHasExteriorDoor, you can't just type "currentLocation.DoorLocation" because DoorLocation isn't a field in Location. You'll need to downcast if you want to get the door location out of the object.



Here's the ecce to model the house We used classes to represent the rooms and locations. and an Interface for any place that has a door.

```
public interface IHasExteriorDoor I
    string DoorDescription { get; )
    Location DoorLocation t get; set; )
```

Here' the IHasExteriorDoor interface

```
public class Room : Location {
    private string decoration;

    public Room(string name, string decoration)
        : base(name) {
        this.decoration = decoration;

    public override string Description t
        get
            return base.Description + " You see " + decoration +
```

The Room class inherits from Location and adds a backing field for the read-only Decoration property. Its constructor sets the field.

```
pUblic class RoomWithDoor : Room, IHasExteriorDoor {
    public RoomWithDoor(string name, string decoration, string doorDescriptionl
        : base(name, decoration)

        this.doorDescription - doorDescription;

    private string doorDescription;
    public string DoorDescription {
        ger { return doorDescription;

    private Location doorLocation;
    public Location DoorLocation {
        get   return doorLocation;
        set { doorLocation = value; )
```

De RoomWithDoor cla.. inherits from Room and imple..ents IHasExteriorDoor. It does everything that the ..oom doe.. but it adds a description of the exterior door to the constructor It also adds DoorLocation, a reference to the location that the door leads to DoorDescription and DoorLocation are ..equi..ed by IHasExteriorDoor

```
pUblic class OUtside : Location {
    ptivate bool hot;
    pUblic bool Hot { get ( return hot; } }

    public Outside(string name, bool hot)
        : ease(name)

        this.hot - hot;


    public override string Description {
        get
            string NewDescription   base.Description;
            if (hot)
                NewDescription +=   It's very hot.-;
            return NewDescription;
```

Outside is a lot like R t inherits from Location, and add• a backing field for the Hot property, ..hich is ..sed in the Description() method extended from the base class

```
public class OutsideWithDoor : Outside, IHasExteriorDoor (
    pUblic OutsideWithDoor(string name, bool hot. string doorDescription)
        : base(name, hot)

        this.doorDescription - doorDescription;


    private string doorDescription;
    public string DoorDescription (
        get ( return doorDescription;


    private Location dootLocation;
    pUblic Location DoorLocation (
        get ( return doorLocation;
        set { doorLocation - value; }


    public override string Description I
        get
            return base.Description + " You see " + doorDescription + ".";
```

OutsideWithDoor ...herits from Outside and implements IHasExteriorDoor, and it looks a lot like RoomWithDoor

The base class's Description property fills "" whether or not the location is hot. And that relies on the original Location class's Description property to add the main description and exits

We're not done yet—flip the page!



Long Exercise Solution (continued)

Here's the code for the form. It's all In the Formt.cs, Inside the Forml dectarauon

```
public partial class Forml : Form

    Location currentLocation;

    RoomWithDoor livingRoom;
    Room diningRoom;
    RoomWithDoor kitchen;

    OutsideWithDoor frontYard;
    OutsideWithDoor backYatd;
    Outside garden;

    public Forml () (
        InitializeComponent();
        CreateObjects();
        MoveToANewLocation(livingRoom);

    private void CreateObjects() (
        livingRoom = new RoomWithDoor("Living Room", "an antique carpet",
                "an oak door with a brass knob");
        diningRoom - new Room("Dining Room", "a crystal chandelier");
        kitchen . new RoomWithDoor("Kitchen", "stainless steel appliances", "a screen door");

        frontYard = new OutsideWithDoor("Front Yard", false, -en oak door with a brass knob");
        backYard - new OutsideWithDoor("Back Yard", true, "a sCreen door");
        garden _ new Outside("Garden", false);

        diningRoom.Exirs = new Location!] { livingRoom, kitchen );
        livingRoom.Exits = new Location)] ( diningRoom );
        kitchen.Exits = new Location!] ( diningRoom };
        frontYard.Exits - new Location!] { backYard, garden );
        backYard.Exits = new Location!] ( frontYard, garden );
        garden.Exits = new Locationl] ( backYard, frontYard };

        livingRoom.DoorLocation - frontYard;
        frontYard.DoorLocation - livingRoom;

        kitchen.DoorLocation • backYard;
        backYard.DoorLoc"tlon • kitchen;
```

This is how the form keeps track of which room is being displayed.

The form uses these reference variables to keep track of each of the rooms in the house.

The for"i's constructor creates the objects and then uses the MoveToANewLocation ··ethod

When the form creates the objects, first it needs to instantiate the classes and pass the right information to each one's constructor.

Here's where we pass the door desc.iption to the OutsideWithDoor tonstructors

Here's where the Exits[] array for each instance i. populated We need to wait to do this until after all the instances are created, because otherwise we wouldn't have anything to put into each array!

For the IHasExteriorDoor objects, we need to set their door locations.

```
private void MoveToANewLocation(Location newLocation) {
    current.Locat.Lon = nevr.cceei cn.

    exits.Items.Clear();
    for (int i = 0; i < currentLocation,Exits.Length; i++)
        exits.Items.Add(currentLocation.Exits[iJ•Name);
    exLts.SeLectedjndex = 0;

    description.Text - currentLocation,Description;

    if (currentLocation is IHasExteriorDoor)
        goThroughTheDoor.Visible = true;
    else
        goThroughTheDoor.Visible - false;
```

The MoveToANewLocation() method displays a new location in the form.

First we need to clear the combo box, then we can add each of the locations names to it. Finally, ..e set its selecte ,ndex (or ..hich line is highlighted) to zero so it ,hows the first item in the list. Don't forget to set the ComboBox's DropDownStyle property to "DropDownList"—that way the user won't be able to type anything into the combo box.

This make the "Go through the door" button nvisible if the current location doesn't imple..ent IHasExter_ Poor_

```
private void goHere_Click(object sender, avenenrcs e)
    MoveToANewLocation(currentLocation,Exits[exits.SelectedIndex));

private void goThroughTheDoor_Click(object sender, EventArgs e) {
    IHasExteriorDoor haspcor • currentLocatLon as IBasExterlorDoor;
    MoveToANewLocation(hasDaor.DoorLocationl;
```

When the user clicks the "Go here:" button, it moves to the location selected in the combo box

We need to use the as keyword in order to downcast currentLocation to an IHasExteriorDoor so we can get access to the DoorLocation field

But we're not done yet!

It's fillll' to create a model of a house-. but wouldn't illbe cool I" lUTll it i"lo a game? Let's do it! You'll play Hide and Seek against the computer. \\',,'n need to add "II Opponent class an<1 have him hid,· in a room. And we'll need to m.rkr- the house a [01 bigger. ()h. and he'll need someplace to hide! We'll add a new interface '" thar some roo„1\, can have a hiding place. Finally, we'll update the f"rlll to let you check the hiding places, and k,"'p track or how many moves you've made trying 10 find your opponent. S"lllH-] fllll? Definitely!

Let's get started!



Time for hide and seek! Build on your original house program to add more rooms, hiding places, and an opponent who hides from you

Create a new project, and use the IDE's "Add Existing Item" feature to add the classes from the first part of the exercise

Add an IHidingPlace interface

We don't need to add anything fancy here. Any Location subclass that implements IHidingPlace has a place for the opponent to hide, so it just needs a string to store the description of the hiding place ("in the closet", "under the bed", etc.)

- Give it a get accessor, but no set accessor—we'll set this in the constructor, since once a room has a hiding place we won't ever need to change it.

Add classes that implement IHidingPlace

You'll need two more classes: OutsideWithHidingPlace (which inherits from Outside) and RoomWithHidingPlace (which inherits from Room). Let's make any room with a door have a hiding place, so it'll have to inherit from RoomWithHidingPlace instead of Room.

So every room with an exterior door will also have a hiding place

Add a class for your opponent

The opponent object will find a random hiding place in the house and it's your job to fill in the details.

- He'll need a private Location field (myLocation) so he can keep track of where he is, and a private Random field (random) to use when he moves to a random hiding place.

- The constructor takes the starting location and returns a new instance of Random. He stores the starting location in myLocation, and sets random in a new instance of Random. He stores the front yard (that'll be passed in by the caller) in his current location field and moves to the hiding place. The hiding place starts moving when the game starts. When he encounters an exterior door, he uses a random number to figure out whether or not he should go through it.

- Add a Move() method that moves the opponent from his current location to a new location. First, if he's in a room with a door, then he flips a coin to decide whether or not to go through the door, so if random.Next(2) is 1 (equal to 1, he goes through it. Then he chooses a random number between 0 and the number of exits from his current location and goes through it. If it doesn't have a hiding place, he'll do it again—he'll choose a random exit from his current location and go there, and he'll keep doing it over and over until he finds a place to hide.

- Add a Check() method that takes a location as a parameter and returns true if he's hiding in that location, or false otherwise.

Add more rooms to the house

Update your CreateObjects() method to add more rooms:

- Add **stairs** with a wooden bannister that connects the living room to the **upstairs hallway**, which has a picture of a dog and a closet to hide in.

- The upstairs hallway connects to three rooms: a **master bedroom**, with a large bed, a **second bedroom**, with a small bed, and a **bathroom** with a sink and a wall. Someone could hide under the bed in either bedroom or in the bathroom.

- The front yard and back yard both connect to the **driveway**, where you could hide in the garage. And there's someone could hide in the **garden**.

Okay, time to update the form

You'll need to add a few buttons to the form. And we'll get a little more intricate with making them visible or invisible, depending on the state of the game.

You use the top two buttons and the combo box exactly the same way as before, except that they're only visible while the game is running.

When the game first starts, the hide button is the only one displayed. When you click it, the form counts to 10 in the text box, and calls the opponent's Move() method 10 times. Then it makes this button invisible.



The middle button's called check. You don't need to set its Text property.

This is the button you'll use to check the room's hiding place. It's only visible if you're in a room that has a place to hide. When it's shown, the Text property is changed from "button3" to "Check" followed by the name of the hiding place—so for a room with a hiding place under the bed, the button will say, "Check under the bed".

8 Make the buttons work

There are two "W buttons u, add to tilt· form.

A'f back to Chapter 2 for a refresher on DoEvents() and Sleep()—they'll come in handy.

* '1'he middle hili")n checks the hiding place in the current room and is only visible when you're in a room witha place 10hide using the opponent's Check () method. If you found him, then il resets (h,' game.

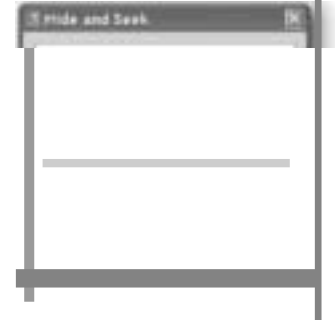
★ The bottom buuon is how you start the game. It counts I,> 10by showing "1...", waiting 200 milliseconds, then showing ..2..." · then "3...", etc. i" ,h,' text box. .After each number, it tells the opponent to move by calling his Mbve () method. Then i, ,h,,,,'s. "Ready "r not, here I come!" for half a sec-ond. ;'nd then the game starts.

Add a method te redraw the form, and another one to reset the game

.\dd a Redrawf'orrn () method that puts the right text ill the description textbox, make ,he buttons visible or invisibh-, and I'UI, the correct label on the middle bUllon_ Then add a ResetGame (} method that's run when youlil1d the opponent. It resets the 0l'l">n"I1' object so that h,' starts illlh,' front yard again—he'll hide „11<11 the user click Ih,' "Hide!" button. It should leave thr- li'nll with llolhi"g llllll the text box and "Hide!" hulloll visible. The text box ,j,,,,,ld say where you found the opponent, and how many move il took.

8 Keep track of how many moves the player made

Make sure the text box displays the number or times Ih,' player checked a hiding place or llHwed between rooms. \Vhen you fi"d Ihe "PI'-nent, he should pop "I' a mesag(b"x that says, "You found me in X nH"es "



M(lke it look right when you start the program

When you first start the program, all you should see is an empty text b"x and the "Hide!" button. When you click Ihe button, the fun begins!



Build on your original house program to add more rooms. hiding places, and an opponent who hides from you.

Here's the new IHidingPlace interface. It just has one string field with a get accessor that returns the name of the hiding place.

```
public interface IHidingPlace {
    string HidingPlaceName { get; }
}

public class RoomWithHidingPlace : Room, IHidingPlace {
    public RoomWithHidingPlace(string name, string decoration, string hidingPlaceName)
        : base(name, decoration)
    {
        this.hidingPlaceName = hidingPlaceName;
    }

    private string hidingPlaceName;
    public string HidingPlaceName {
        get { return hidingPlaceName; }
    }

    public override string Description {
        get
        {
            return base.Description + " Someone could hide " + hidingPlaceName + ".";
        }
    }
}

public class RoomWithDoor : RoomWithHidingPlace, IHasExteriorDoor {
    public RoomWithDoor(string name, string decoration,
                        string hidingPlaceName, string doorDescription)
        : base(name, decoration, hidingPlaceName)
    {
        this.doorDescription = doorDescription;
    }

    private string doorDescription;
    public string DoorDescription {
        get { return doorDescription; }
    }

    private Location doorLocation;
    public Location DoorLocation {
        get { return doorLocation; }
        set { doorLocation = value; }
    }
}
```

The RoomWithHidingPlace class inherits from Room and implements IHidingPlace by adding the HidingPlaceName property. The constructor sets its backing field.

Since we decided every room with a door also needed a hiding place, we made RoomWithDoor inherit from RoomWithHidingPlace. The only change to it is that its constructor takes a hiding place name and sends it to the RoomWithHidingPlace constructor.

```
public class OutsideWithHidingPlace : Outside, IHidingPlace {
    public OutsideWithHidingPlace(string name, bool hot, string hidingPlaceNamel
        : base(name, hot)
    ( this.hidLnql'Laceftame = hidingP1aceName; )

    private string hidingPlaceName;
    public string HidingPlaceNilme {
        get { return hidingPlaceNilmel

    public override string Description (
        get
            return base.Description + " Someone could hide " + hidingPlaceName + "."1
```

The OutsideWithHidingPlace class inherits from Outside and implements IHidingPlace just like RoomWithHidingPlace does.

The Opponent class constructor takes a starting location. It creates a new instance of Random, which it uses to move randomly between rooms

```
public class Opponent {
    private Random random;
    private Location myLocation;
    pUblic Opponent(Locat ion startingLocation) (
        myLocation = startingLocatlon;
        random - new Random();

    public void Mbve() {
        it (myLocation is IHasExteriorDoor)
            IHasExteriorDoor LocatlonWlthDoor
                                myLocatlon as IHasExteriorDoor;
            if (random,Nel<t(2) -- 1)
                myLocation = LociltionWithDoor.DoorLociltion;

        bool hidden = false;
        while (hIdden) {
            int rand - random.Next(0, myLocation.Exits.Length):
            myLociltion = myLociltion.Exits[randj;
            it (myLocation is IHidingPlacel
                hidden = true;
```

The Move() method first checks if the current room has a door using the is keyword—if so, it has a 50% chance of going through it. Then it moves to a random locaiton, and keeps moving until it finds a hiding place.

The guts of the Move() ..ethod is this w hi le loop It keep looping until the variable hidden is true—and it sets it to true when it finds a room with a hiding place

```
    public bo01 Check(Location 10cationToCheckl {
        if {lociltionToCheck != myLocationl
            return reteer
        else
            return true:
```

The Check() method just checks the opponent's location against the location that was passed to it using a Location reference. If they point to the same object, then he's been found!

We're not done yet—flip the page!



He-e a.e all the fields in the Forml class It uses the... to keep track of the locations, the opponent and the nu...ber of moves the player has made

```
int Moves;

Location currentLocation;

RoomWithDoor livingRoom;
RoomWithHidingPlace diningRoom;
RoomWithDoor kitchen;
Room stairs;
RoomWithHidingPlace hallway;
RoomWithHidingPlace bathroom;
RoomWithHidingPlace masterBedroom;
RoomWithHidingPlace secondBedroom;

OutsideWithDoor frontYard;
OutsideWithDoor backYard;
OutsideWithHidingPlace garden;
OutsideWithHidingPlace driveway;

Opponent opponent;
```

The Formj constructor creates the objects, sets up the opponent, and then resets the ga..e We added a boolean para..eter to ResetGa...e() so that it only displays its message when you win, not when you first start -l' the progra...

```
public Forml() (
    InitializeComponent();
    CreateObjects();
    opponent . new Opponent(frontYardl;
    aeeeeoaee(false);
```

```
private void MoveToANewLocation(Location newLocationl
    Moves++;
    currentLocation = newLocation;
    Redra"Form();
```

The MoveToANewLocation() method sets the new location and then redraws the form.

```
private void Redrawrorm() {
    exits.ltems.Clear();
    for (int i - 0; i < currentLocation.Exits.Length; i++l
        exits.1terns.Add(currentLocation.Exits[i1.Namel;
    exits.SelectedIndex = 0;
    description.Text = currentLocation.Description + "\r\n(move #" + Moves + ")";
    if (current.LocatLon is IHidingP!ace) {
        IHidingPlace hidingPlace _ currentLocation as IHidingPlace;
        check.Text _ "Check " + hidingPlace.HidlngPlaceName;
        check.Visible _ true;

    else
        check.Visible - false:
    if (currentLocation is IHasExteriorDoor)
        goThroughTheDoor.Visible true:
    else                                    v'-,
        goThroughTheDoor.Visible false;
```

We need the hiding place na...e. but we've only got the CurrentLocation object which doesn't have a HidingPlaceNa...e property, '" we can use as to dewncast the reference to an IHidingPlace va..iable.

Red..awForm{) populates the combo box list, sets the text (adding the nu ber of moves), and the.....akes the buttons visible i isible depending on whether or not there's a door the room has a hiding place

Wow—you could add an entire wing onto the house just by adding a couple of lines That's why well-encapsulated classes and objects are really ...eful

```
private void CreateObjects()
    livingRoom - new RoomWithDoor("Living Room", "an antique carpet",
              "inside the closet", "an oak door with a brass handle");
    diningRoom _ new RoomWithHidingPlace("Dining Room", "a crystdl chdndelier",
              "in the taLf armcirevrr
    kitchen - new RoomWithDoor("Kitchen", "stainless steel appliances",
              "in the cabinet", "a screen door");
    stairs - new Room("Stairs", "a wooden bdnnister");
    hallway = new RoomWithHidingPldce("Upstdirs Hallway", "a picture of a dog",
              "in the closet");
    bathroom - new RoomWithHidingPlace("Bathroom", "a sink and a toilet",
              "in the shower");
    masterBedroom - new RoomWithHidingPlace("Master Bedroom", "a large bed",
              "under the bed");
    seccncaeoroom » new RoomWithHidingPlacel"Second Bedroom", "a small bed",
              "under the bed");

    frontYdrd - new OutsideWithDoor("Front Yard", false, "a heavy-looking oak door");
    backYard = new OutsideWithDoor("Back Yard", true, "a screen door");
    gdrden • new OutsideWitHidingplace("Garden", false, "inside the shed");
    driveway · new OutsideWithHidingPlace["Driveway", true, "in the garage"}:

    diningRoom,Exits - new Location[] ( livingRoom, kitchen );
    livingRoom.Exits _ new Location[] { diningRoom, stairs );
    kitchen.Exits • new LocationlJ ( diningRoom );
    sta.irssExLts • new Location[J { livingRoom, hallway I:
    hallway.Exits - new Location[1 ( stairs, bathroom, masterBedroom, secondBedroom );
    bathroom.Exits - new Location[] ( hallway );
    masterBedroom.Exits   new Location!] { hallway I;
    secondBedroom.Exits • new Locarion!] { hallway I;
    frontYard.Exits - new Locat.Lcnll ( backYard, garden, driveway );
    backYard.Exits - new Location[] ( frontYard, garden, driveway );
    garden.Exits - new Location[]   backYard, frontYard );
    driveway.Exits - new Location[] ( backYard, frontYard );

    LivingRoom.DoorLocation = frontYard;
    frontYard.DoorLocation - LivingRoom:

    kitchen.DoorLocation = backYard;
    backYard.DoorLocdtion _ kitchen;
```

The new CreateObjects() ..ethod creates all the object.. to build the house It's a lot like the old one, but it has a whole lot ..ore places to go.

We're still not done—flip the page!



Exercise Solution (continued)

Here's the rest of the code for the form–The goHere and goThroughTheDoor button event handlers are identical to the ones in the first part of this exercise, so flip back a few pages to see the...

```
private void ResetGame(bool displayMessage) {
    if (displayMessage) {
        MessageBox.Show("You found me in "+ Moves + " moves!");
        IHidingPlace foundLocation = currentLocation@ IHidingPlace;
        description.Text = "You found your opponent in " + Moves
            + " moves! He was hiding" + foundLocation.HidingPlaceName + ".";
    }
    Moves = 0;
    hide.Visible = true;
    goHere.Visible = false;
    check.Visible = false;
    goThroughTheDoor.Visible = false;
    exits.Visible = false;
}

private void check_Click(object sender, EventArgs e) {
    Moves++;
    if (opponent.Check(currentLocation))
        ResetGame(true);
    else
        RedrawFormO;
}

private void hide_Click(object sender, EventArgs e) {
    hide.Visible = false;

    for (int i = 1; i <= 10; i++) {
        opponent.Move();
        description.Text = i + "... ";
        Application.DoEvents();
        System.Threading.Thread.Sleep(200);
    }

    description.Text = "Ready Or not, here I come!";
    Application.DoEvents();
    System.Threading.Thread.Sleep(500);

    goHere.Visible = true;
    exits.Visible = true;
    MoveToANewLocation(livingRooml;
}
```

The ResetGame() method resets the game. It displays the final message, then makes all the buttons except the "Hide!" one invisible.

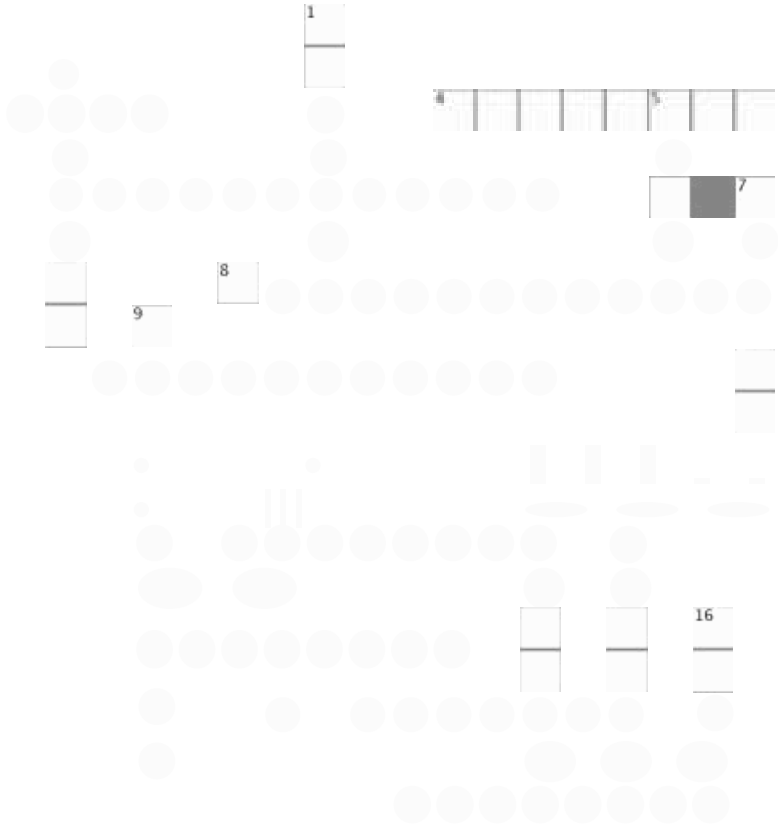
We want to display the name of the hiding place, but CurrentLocation is a Location reference, so it doesn't give us access to the HidingPlaceName field. Luckily, we can use the as keyword to downcast it to an IHidingPlace reference that points to the same object.

When you click the check button, it checks whether or not the opponent is hiding in the current location. If he is, it resets the game. If not, it redraws the form to update the number of moves.

The hide button is the one that starts the game. The first thing it does is make itself invisible. Then it counts to 10 and tells the opponent to move. Finally, it makes the first button and the combo box visible, and then starts off the player in the living room. The MoveToANewLocation() method calls RedrawForm().



Objeetcross



Across

3. What an abstract melhod ooesnt have
4. C# doesn't allow inheritance
6. When you use a pass subdass to a method that expects its base class, you're using thiS OOP prinople
8. The OOP principle where you hide private data and only expose ecse methods and fields that other classes need access to
10 One of the lour pmcpiss of COP Ihat you implement using the colon operator
14. Every method In an interlace is autom atically' _____
15. Your crass that implements an nartace that _____ from another interlace, nen you need to implement all of its members too
17. An access momernere not valid for anything inside an interlace
18. Object Programming means creating programs that combine your data and code together into oasses and objects

Down

1. When you move common methods from specnc classes to more a general cass that they all inherit from, you're using this COP principle
2. If a c ass ltlat Implemenls an Interlace doesn't implement all of its methods, getters and seneo. een the project won't _____
5. Everything in an Interlace IS automatically
7. An abstract class can include both abstract and methods
9. You cant an abstract class
11. Aclass that implements this must Include all of the methods, getters and setters that It defines
12. What you do with an interface
13. The is keyword returns true if an implements an nartace
16. An interlace can'ttechnically include a . but it can define getters and setters that look just like one from the octaoe



Pool puzzle Solution from page 287

Your job i, to take code , nippets from the pool and place them into the blank lines in the code and output. You may use the same snippet more than once, and you won't need to use all the snippets. Your goal is to make a set of classes that will compile and run and produce the output listed.

Here's where the Acts class call. the constructor in Picasso, which it inherits fr— It passes "Acts" into the constructor, which gets stored in the face property.

```
public interface Nose
{
    int Ear()
    string Face { get; }
}

public abstract class Picasso : Nose
{
    public virtual int Ear()
    {
        return 7;
    }

    public Picasso(string face)
    {
        this.face = face;
    }

    public virtual string Face
    {
        get { return face; }
    }

    string face;
}

public class Clowns : Picasso
{
    pUblic Clowns() : base("Clowns") , [
```

Properties can appear anywhere in the , lass! It's easier to ead your code if they're at the top, but it's perfectly valid to have the face property at the bottom of the Picasso class

```
public class Acts : Picasso
{
    public Acts() , base("Acts") {
    public override int Ear()
        return 5;
    .
```

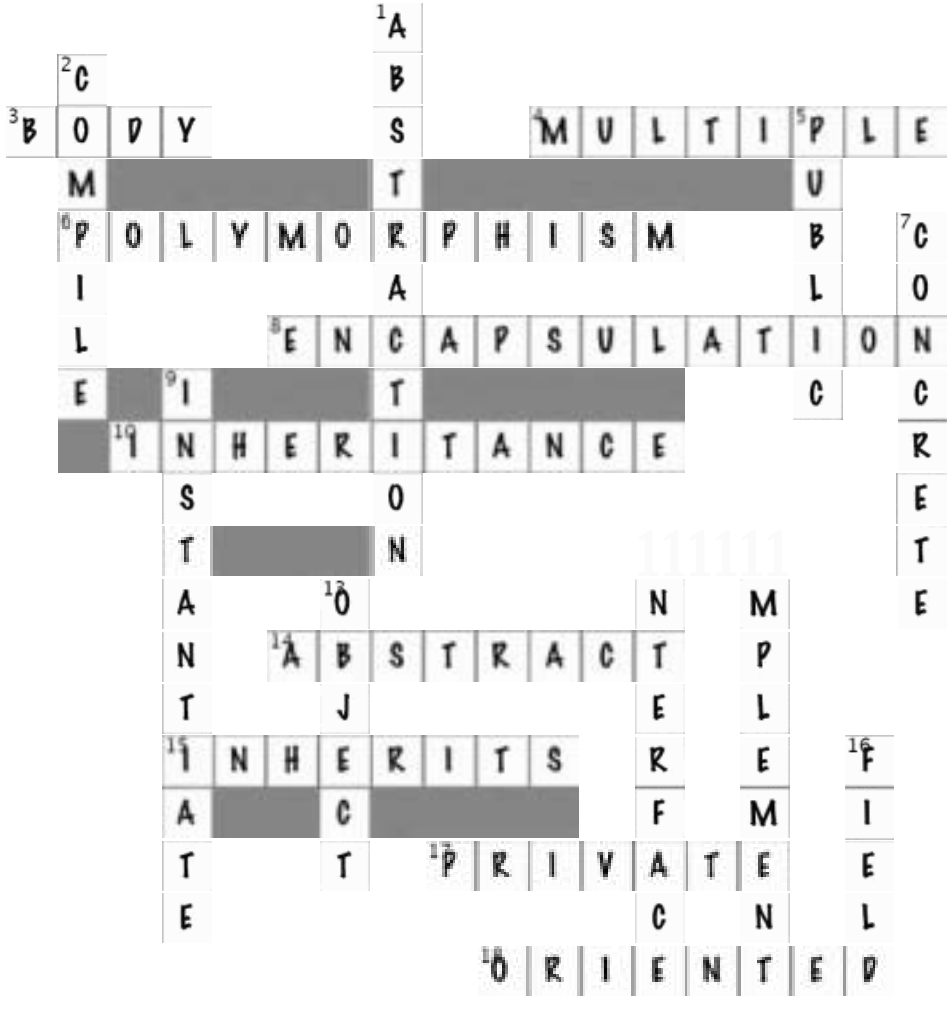
```
public class Of76..:.Clowns....
    public override string Face
        get { return "Of76"; }

public static void Main(string[J args) {
    string result - "";
    NoselJ i = new Nosel)J;
    llOl   ne.w Acts();
    illJ   _ new Clowns()
    il2J = new Of76();
    for (int x -O; x<:3; x++) {
        result +-  ..,'x' ... Ear()  +
                +  i[x].Face  ) + "\n";
    [
    MessageBox.Show(result);
```

Face is a get accessor that returns the value of the face property. Both of them are defined in Picasso and inherited into the subclasses



Objectcross solution



Storing lots of data



When it rains. it pours.

In the real world, you don't gel to handle your data In tiny little bts and pieces. No, your data's going to come at you in loads, piles, and bunches You'll need some pretty powerful tools to organize all of it. ero that's where **collections** come in, They let you store, sort, and manage all the data that your programs need to pore through. That way you can think about writing programs to work with your data, and let the collections worry about keeping track of it for you.

Stril1gs dOl1't always work tor storil1g categories ot data

Suppose you have several worker bees, all represented by Wo r ke r classes. How would you write a constructor 1hat took a job as a parameter? If vou use a string for the job name, you might end up with code· that looks lik,' Ihi,:

Our bee management software kept track of each worker's job using a string like "Sting Patrol" or "Nectar Collector".

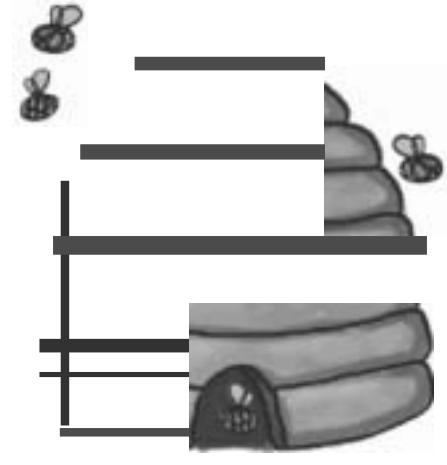
Our code would allow these values to be passed in a constructor even though the program only supports Sting Patrol, Nectar Collector and other jobs that a bee does.

```
Worker buzz = new Worker("Attorney General");
Wor ker clover = new Worker("Dog Walker");
Worker gladys = new Worker("Newscaster");
```

This code compiles, no problem. But these jobs don't make any sense for a bee. The Worker class really shouldn't allow these types as valid data.

You could probably add code 10 the Wo r ke r constructor 10 dn·,·k each string and make sure it', a \'alid h,'" joh..\11hough, jf you _"Id new jobs that bees can do, you've got 10 chang.. this code and recompile the Worker class. But that's a pretty short-sighted solution. What if you have othe classes that need to check for the t\'I"" "I' worker bees they can be? Now you've got to dUl'lic "lt·code-, and that's a bad path to go down.

What we need is a way to say, "Hey, there are only certain value that are allowed here." We need to ..**num**..rat.. the ""I"," that are okay I" use.



Enums let you enumerate a set of valid values

.\" enum is a data type that only allows certain value fi>r that pi,rce, "I' data. So we could define an (,1111" ,'"II"d Jobs, and d"li",' the an,,wed j,,bs:

The stuff inside the brackets is called the enu..erator list, and each one of the.. i an enumerator The whole thing together i called an enumeration

But ..ost people just call the.. enu..s

This is the name of the enum

```
public enum Job {
    NectarCollector,
    StingPatrol,
    HiveMaintenance,
    BabyBeeTutoring,
    EggCare,
    HoneyManufacturing,
}
```

The last enumerator doesn't have to end with a comma, but using one makes it easier to rearrange them using cut and paste.

Each of these is a valid job. Any can be used as a Jobs value.

Separate each value with a comma, and end the whole thing with a curly brace.

Now, you can reference these with types like this:

This i the na..e of the enum.

Finally, the value you want from the enum.

```
Worker nanny - new Worker(Job.EggCare);
```

We've changed the Worker constructor to accept Worker.Jobs as its parameter type.

Any other values aren't "II. ,wed. Y(" can't jU'1 III"k,' III' a new ,'al",' for the enum. If you do, the program won't compile.

```
private void buttonl_Click{object sender EventArgs e)

    Worker buzz - new Worker(Jobs.AttorneyGenerall;
```

Here's the exror you get from the compiler.

```
'Jobs' does not contain a definition for
'AttorneyGeneral'
```

Enums let you represent numbers with names

Sometimes it' easier to work with numbers if you have names fcn them. Yo" ,"au assign numbers to rhcvaluo, in an "mUHand use the name 1<1 refer 10 them. That wav. you don't have bunch "f unexplain,"(1 numbers floating around in your code. Here's an enum 1" keep track of a the score for trick at a dog "' "" be,iti"". It's insid.. th.. DogConlp... ilion class. so if yc,u want to use it ollt,id,-the class you'll ",',-d to call it DogCompetition. 'TrickScore.s.

```
class DogCompetition{
    public enum TrickScore {
        Sit = 7,
        Beg = 25,
        RollOver = 50,
        Fetch = 10,
        ComeHere = 5,
        Speak = 20,
    }

// code later in the class
int score = (int)TrickScore.Fetch * 3;
MessageBox.Show(score.ToString());
```

This is the DogCompetition TrickScores enum.

These don't have to be in any particular order, and you can give multiple names to the same number.

Supply a name, then "=", then the number that name stands in for.

The (int) cast tells the compiler to turn this into the number it represents. So since TrickScroe.Speak has an index of to 20, (int)TrickScore.Speak turns it into the int value 20.

This shows "30", 10 * 3.

ToString() returns the name... in this case, "Comellere".

MessageBox.Show(TrickScore.ComeHere.ToString(»

You can cast the "Htltll as a number and do calculations with it, or you can use the ToString() methoc1to treat ,he Il," "" as a string. If you dou't assign any numbe1 to a name, the it"lll' ill th,' Ii,t will be give" ind..x numbers by default. The first ilem will be assigned a 0 value, Ih,-second a I. etc.

But what h"I'I"'u' if you want ttl "" really big numl >ers for one "I' the enumerators? The default type for the IIIlml >ers III an enum is lnt, so you'll need '" specify the type you need using tit,· : operator, like this:

You can put an enum inside a class like this, or it can exist on its 0wn outside of a class.

```
public enum TrickScore : long {
        Sit = 7,
        Beg = 2500000000025
}
```

This tells the compiler to treat values in the TrickScores enum as longs, not ints.

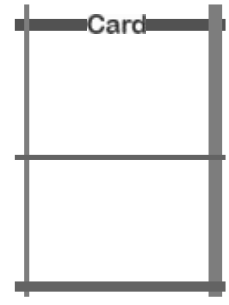
If you tried to compile this code without specifying long as the type, you'd get this message

```
Cannot implicitly convert type 'long' to 'int'.
```



Exercise

Use what you've learned about enums to build a class that holds a playing card.



1. **Create a new project and add a Card class**

   You'll need two public fields: Suit (which will either be Spades, Clubs, Diamonds, or Hearts) and Value (Ace, Two, Three... Ten, Jack, Queen, King). And you'll need a read-only property, Name ("Ace of Spades", "Five of Clubs", etc.).

2. **Use two enums to define the suits and values.**

   Make sure that `(int)Card.Suits.Spades` is equal to 0, followed by Clubs (equal to 1), Diamonds (2), and Hearts (3). Make the values equal to their face values: `(int)Card.Values.Ace` should equal 1, Two should be 2, Three should be 3, ... Jack should equal 11, Queen should be 12, and King should be 13.

3. **Add a property for the name of the card**

   Name could be a read-only property. The get accessor should return a string that describes that card. This code will run in a form that calls the Name property from the card class and displays it:

```
Card card = new Card(Card.Suits.Spades, Card.Values.Ace);
string cardName = card.Name;
```

To make this work, your Card class will need a constructor that takes two parameters.

The value of cardName should be "Ace of Spades".

4. **Add a form button that pops up the name of a random card**

   You'll can get your program to create a card with a random suit and value by casting a random number between 0 and 3 as a Cards.Suits and another random number between 1 and 13 as a Cards.Values. To do this you can take advantage of a feature of the built-in Random class that give it three different ways to call its Next() method:

When you've got more than one way to call a method, it's called overloading. More on that later...

```
Random random = new Random();
int numberBetween0and3 = random.Next(4);
int numberBetween1and13 = random.Next(1, 14);
int anyRandomInteger = random.Next();
```

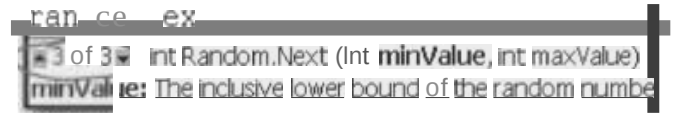
This tells Random to return a value at least 1 but under 14.



there are no Dumb Questions

Q: Hold on. **second.** When I **was typing** in **that** code, I noticed th. t . n **IntelliSense** window **popped** up **that said something** abuut **"3 of 3"** when I **used** that **Random.Next() constructor. What was that** .bout?

A: What you saw was a constructor that was overlo.ded When a crass has a method that you can call more tnan one way. it's called overloading. When you're using a cess with an overloaded method. the IDE lets you know all of the options that you have. In this case, ue Random class has three possible Next() methods As soon as you type "random Next(" into the code window. the IDE pops up its IntelliSense box Illat shows the parameters lor the different overloaded methods. The up and down arrows next to the '3 of 3" let you scroll between them That's really useful when you're dealing With a method that has dozens of overloaded definitions. So when you're doing it, make sure you choose the right wedoeced Next() method! But doll'l worry too much now—we'll talk a lot about overloading later on in the chapter





A deck of cards is a great example of where limiting values is important. Nobody wants to turn over their cards and be faced with a Joker of Clubs, or a 13 of Hearts. Here's how we wrote the Card class.

```
public class Card {

    public enum Suits {
        Spades,
        Clubs,
        Diamonds,
        Hearts
    }

    public enum Values {
        Ace = 1,
        Two = 2,
        Three = 3,
        Four = 4,
        Five = 5,
        Six = 6,
        Seven = 7,
        Eight = 8,
        Nine = 9,
        Ten = 10,
        Jack = 11,
        Queen = 12,
        King = 13
    }

    public Suits Suit;
    public Values Value;

    public Card(Suits suit, Values value) {
        this.Suit = suit;
        this.Value = value;
    }

    public string Name {
        get { return Value.ToString() + " of " + Suit.ToString(); }
    }
}

private void button1_Click(object sender, EventArgs e) {
    Random random = new Random();
    Card card = new Card((Card.Suits)random.Next(4), (Card.Values)random.Next(1, 14));
    MessageBox.Show(card.Name);
}
```

When you don't specify values, the first item in the list is equal to zero, the second is 1, the third is 2, etc

Here's where we set the value of Card.Values.Ace to 1.

When you set up your Suit and Value fields, you can just use Suits instead of Card.Suits, since you're inside of the Card class.

The get accessor for the Name property can take advantage of the way an enum's ToString() method returns its name converted to a string

Here's where we use the overloaded Random.Next() method to generate a random number that we cast to the enum.

Here's the code for the button that pop up the name of a random card

We could use an array to create a deck of cards...

What if you want to create a class to represent a deck of cards? It would need a way to keep track of every card in the deck, and it'd need to know what order they were in. A Card array would do the trick—the first card would be at index 0, the next card at index 1, etc. Here's a starting point -- a Deck that starts out with a full deck or 52 cards.

This array declaration would continue all the way through the deck. It's just abbreviated here to save space.

```
public class Deck {
    private Card[] cards
        new Card(Card.Suits.Spades, Card.Values.Ace),
        new Card(Card.Suits.Spades, Card.Values.Two),
        new Card(Card.Suits.Spades, Card.Values.Three),
        // ...
        new Card(Card.Suits.Diamonds, Card.Values.Queen),
        new Card(Card.Suits.Diamonds, Card.Values.King),


    public void PrintCards()
        for (int i = 0; i < cards.Length; i++)
            Console.WriteLine(cards[i].ToString());
```

... but what if you wanted to do more?

Think of everything you might need to do with a deck of cards though. If you're playing a card game, you routinely need to change the order of the cards, and add and remove cards from all the deck. You just can't do that with all array very easily.

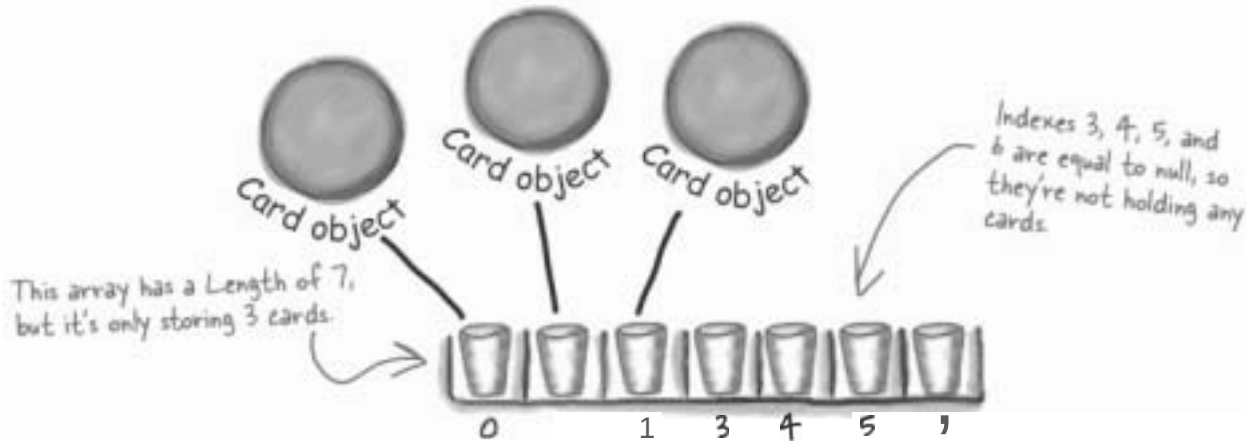


How would you add a ShuffleO method to the Deck class that rearranges the cards in random order? What about a method to deal the first card off the top of the deck? How would you add a card to the deck?

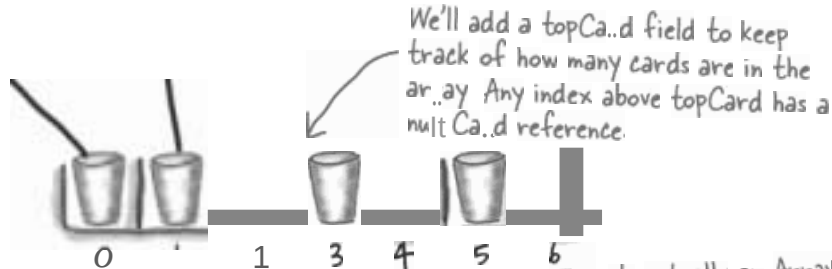
Arrays are **hard** to work wtth

,\n array is fine f(·r storing a fixed li,{ of values or references 1lm once you need 101ll'J\l' arrav elements around, or add more element than thr- array can hold. things start (o get a little sticky.

- Every array has a length, and you need {oknow the length (o work with it. You could use null references t" keep some array elements empty:



- You'd need to k,, '1' track "f h,w many cards are being held. S" y(" 'd need all iut field, which we could call t opce ro, which would hold the index of the last card in the array. S" our 3..card array would havea Lengt h of 7, but we'd set tooce rct equal {o 3.



The·e's actually an Array.Resize() "ethod built into the .NET Framework that does exactly that.

- Hut now things get complicated, It's easv enough 1" add a Peek() uw{ho(1 that jU'1 returns a reference 10 (h,' top card—so you can peek at Ihe ("p"f the deck. But what if you want to add a card? If topCard is less lh,,,, the array's Length, you ,-all just ptll your card ill the array at that index aud add I I" topCard. But if it the array's full, you'll need to create a new, bigger array and copy the existing cards to it. Rerm>ving a card is easy enough - h1l1 after you ,,,hlr,"'1 I li, >lll toCard, you'll need to make ,un· {, Iset the removed card's array index back I" nulL. \nd what if you need 10remove a card **from th**.. Dlid dl.. of Ih.. **list**? If you remove card #-1. you'll need to move card 5 back to replac, it. and (h"ulll"n' 6 back, then i back...wow, what a mess!

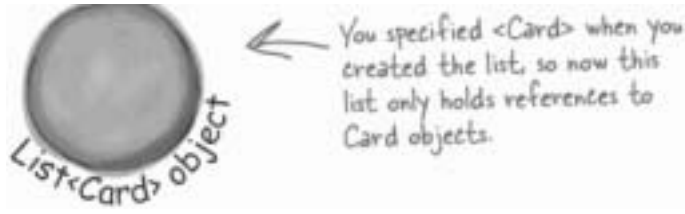
Lists llIake it easy to store **collections** of... **anything**

The .NET Framework has a hund l of coUl'ction classe that handle all of those nasty issues that come up with ytu add and remo" ',. array elements. The most common sort of collection is a **List**. Once vou creat a List object, it's easy to add all il"tH, remove one from any location in the list, peek at an iUIll, "",I even move all item fr"ll one place iu 110e list to another. Here's how a list works:

O First **you create new instance** of List

Every array has a type—you dOl"! just! have an array, you have an inl array, a Card array, etc. Lists art-the same way. You need (o specify the type of object or value lhal the list will h"ld by putting it in angle brackets <> when you use the new keyword to create it.

```
List cards - new List<Card>();
```

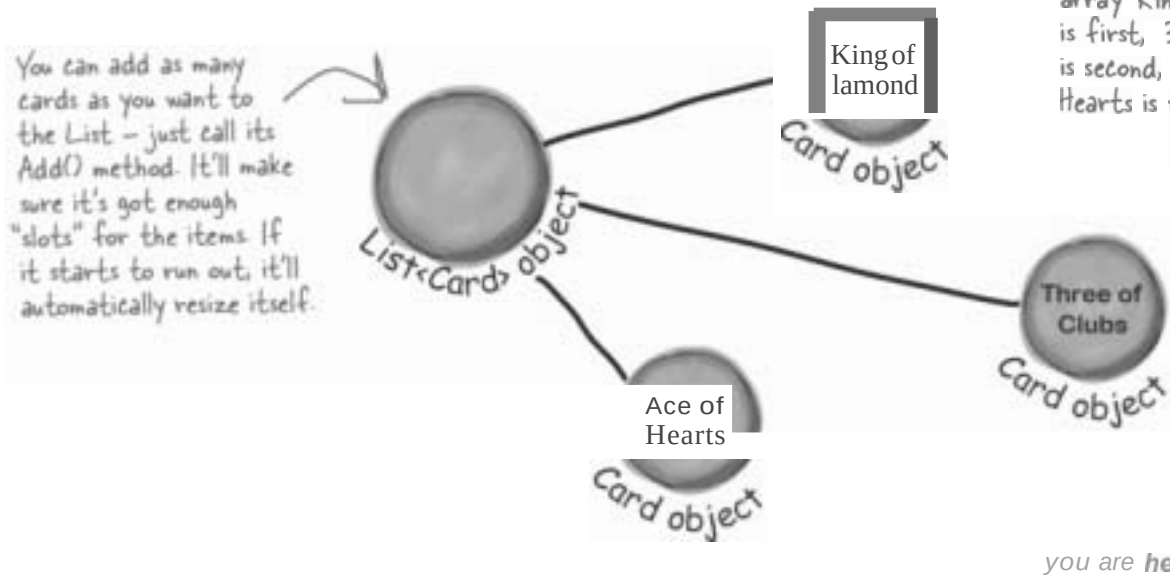


O Now you **can** add to your List

Once you've got ; List object, you can add as many items l" it as you want (as long as Ihey match whatever type you specified when you created your new List).

```
cards.Add(new Card(Card.Suits.Diamonds, Card.Values.King);
cards.Add(new Card(Card.Suits.Clubs, Card.Values.Three);
cards.Add(new Card(Card.Suits.Hearts, Card.Values.Ace);
```

A list keeps its ele..ents in order, just like an array King of Diamondl is first, 3 of Clubs is second, and Ace of Hearts is third.



lists are more flexible than arrays

The List ,-Iss is built into the .NET Framework, and il lets you do a lot of things with objects 11...1you can't d" wilh .• l'L.in old array. Check out '" m" of the things you '-,lll do with a List.

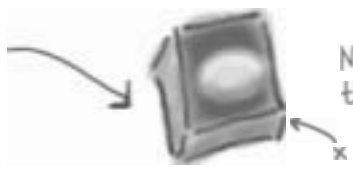
You can make one.

```
List<Egg> myCarton - new List<Egg>();
```

A new List object is created on the heap. But there's nothing in it yet

Add something to it.

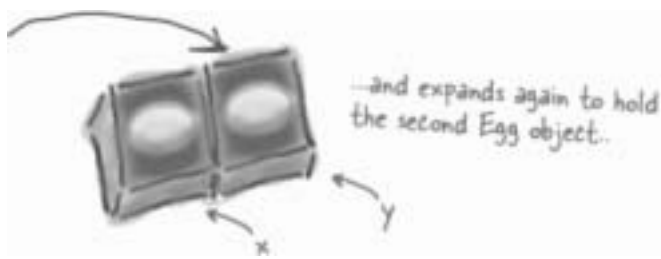
```
Egg x = new Egg();
myCarton.Add(x);
```



Now the List expands to hold the Eg object....

Add something else to it.

```
Egg y - new Egg(l;
myCarton.Add(y);
```



...and expands again to hold the second Egg object.

Find out how **many** things **are** in it.

```
int theSize - myCarton.Count;
```

Find **out** if it **has something** in **particular in** it.

```
bool Isin - myCarton.Contains(x);
```

Now you can search for any Egg inside the list. This would definitely come back true.

Figure out where that thing is.

```
int idx - mycertcn.lndexOf(yl;
```

The index for x would be 0 and the index for Y would be I

Find out how much th. list **will hold.**

```
int limit - myCarton.Capacity;
```

Th.....1 tell you the number of objects the list can hold before it has to resize itself

Take something out if it.

```
myCarton.Remove(y);
```





When we ..emoved y, we left only " 'n the List, ... it shrank!



Sharpen your pencil

Fill in the rest of the table below *by* looking at the List code on the left and putting in what *you* think the code might be if it were using a regular array instead. We don't expect *you* to get all of them exactly right, so just make your best guess.

Assume these statements are all executed in order, one after another.

We filled in a couple for you...

List	regular array
`List<String> myList = new List <String>();`	`String [] myList = new String[2];`
`String a = "Yay!";`	`String a = "Yay!";`
`myList.Add(a);`	
`String b = "Bummer";`	`String b = "Bummer";`
`myList.Add (b);`	
`object o = myList [1];`	
`int theSize = myList.Count;`	
`bool isIn = myList.Contains(b);`	

Hint: You'll need than more than one line of code here



Your job was to fill in the rest of the table below by looking at the Li't code on the left and putting in what you th ink the code mig ht be if it were u,ing a regular array instead.

List	regular array
List<String> myList = new List <Stringq> ();	StringlJ myList new strincjzj.
String a = "Yay!" myList.Add{a);	String a "Yay!"; ..yList[O] = a;
String b = "Bummer";	String b "Bummer";
myList.Add{b);	..yList[l] - b;
int theSize = myList.Count;	,nt theSize = ..yList.Length;
object o = myList [11 ;	ob ect o - ..yList[l];
bool isIn = myList.Contains{b);	bool isln = false; for (int i = Di ; < ..yList Length; ift) { if (b == ..yList[iJ) I isln = true;))

t

Li, ts are objects rh.u use methods j",t like every other class you've used so far. You can see the li, t "f methods available from within the IDE just bv typing a . next to the List nann- and you pass parameters to tlremj,,'t the same as you would for a class you created yourself.

t

" 'ith arrav vou're a lot more Jill,iu'd, Y"" need 11' set the , iz,' ,,l' the array when you create it. and any logic that'll need to be performed "ll it will need to I,,· written on vour own.

The .NET F..a..ework does have an Array class which makes some of these things a little easier to do., but we're concentrating on List objects because they're a lot easier to use

lists shriltk altd grow dyltalllically

The great thing about a List is that you don't need to know how long it'll be when you create it. A List automatically grows and shrinks to fit its contents. Here's an example or a few of the methods you'll work with. Working with Lists is easier than arrays:

We're declaring a List of Shoe objects called shoeCloset.

```
List<Shoe> shoeCloset = new List<Shoe>();

shoeCloset.Add(new Shoe()
    { Style = Style.Sneakers, Color = "Black" });
shoeCloset.Add(new Shoe()
    { Style = Style.Clogs, Color = "Brown" });
shoeCloset.Add(new Shoe()
    { Style = Style.Wingtips, Color = "Black" });
shoeCloset.Add(new Shoe()
    { Style = Style.Loafers, Color = "White" });
shoeCloset.Add(new Shoe()
    { Style = Style.Loafers, Color = "Red" });
shoeCloset.Add(new Shoe()
    { Style = Style.Sneakers, Color = "Green" });

int numberOfShoes = shoeCloset.Count;

foreach (Shoe shoe in shoeCloset)
{
    shoe.Style = Style.Flipflops;
    shoe.Color = "Orange";
}

shoeCloset.RemoveAt(4);

Shoe thirdShoe = shoeCloset[3];
Shoe fifthShoe = shoeCloset[5];
shoeCloset.Clear();

shoeCloset.Add(thirdShoe);
if (shoeCloset.Contains(fifthShoe))
    Console.WriteLine("That's surprising.");
```

You can use a new statement inside the List Add() method

The Count property returns the total number of Shoe objects in the List

This foreach loop goes through each of the shoes in the closet

The RemoveAt() method will remove the object by it's reference; RemoveAt() does it by index number

The Clear() method removes all of the objects in a List

We saved references to two shoes before we cleared the list. We added one back, but the other's still missing

This line will never run, because Contains() will return false. We only added thirdShoe into the cleared list, not fifthShoe

foreach is a special kind of loop for Lists. It will **execute** a statement for each object in the List. This loop **creates** an identifier called **shoe.** As the loop goes through the items, it sets shoe equal to the first item in the list, then the second, then the third, until the loop is done.

foreach loops work on arrays, too! In fact, they work on any collection

Here's the Shoe class we're using

```
public class Shoe
    public Style Style;
    public string Color;


public enum Style
    Sneakers,
    Loafers,
    Sandals,
    Flipflops,
    Wingtips,
    Clogs,
```

list objects call store any type

\;ou've already seen that " List can store strings "r Shoes. You c◦n,ld also make Lists "f integers, 'n-any ,,'her object you can create. Th"1lll;,k," a List a **generic collec**I;on, \\1ıe" you "n';"" ;, ",." 1list object, you tie il I 1specific type: y◦>I, '·.tll have a list of i"Is, or strings, ..1'Sh◦ object◦. That Ill••kes working " ilh I.ists easy ,,,,,-r you've ıted your list, you alv◦ays kn",·the type of dat.◦ that's inside it.

This doesn't actually mean that you add the letter T It's notation that you11 see whenever a class or interface works with all types The <T> part means you can put a type .. there, like List<Shoe>, and limit the type

```
List name = new List<T>();
```

Lists can be either very flexible (allowing any type) or very restrictive. So they do what arrays do, and then quite a few things more

Thr .. ET Fram comes with some generi.. inl..rfi: ◦ that let tl ,-,llections you're building work with any and all types List implements those interfi ·es and th... ◦ why you ıld cre..... a List ..f integers and ,vork with il ,using 1 ·tty much the s,,, way that you would work " ilh a List of Shoe objects.

— —•• ***Che..k*** it *o,,'f◦r* ***yo..rself.*** Type the word, List, inl" Ih.. IIDE and the" right-◦lick on il :",d select ..(i.. To Defin irjou". That will take you I" th◦ declaration for th◦ List ◦Lss. h iml'lements a few interfaceS:

This is where RemoveAt(), IndexOf(), and Insert() come from.

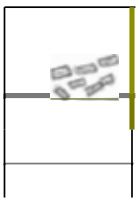
```
pub!      ss List<T> : List<T>
 correcercncrs   Enumerable<T>  rt.tat,
ICollection, IEnumerab e      ,
```

This is where Add(), Clear(), CopyTo(), and Remove() come from It's the basis for all generic collections.

Th.. ·nterface lets you use foreach among other things.

BULLn POINTS

- ust s a class in the .NET Framework
- A List **resizes dynamically** to whatever size is needed. It's got a certain capacity—once you add enough data to the list, it'll grow to accommodate it
- To put something into an List, use **Add()**. To remove something from a List, use **Remove()**.
- You can remove objects using their index number using **RemoveAt()**.
- You declare the type 01 the List using a **type parameter**, which is a type name in angle brackets Example List<Frog> means the List will be able to hold only objects of type Frog
- To find out where something is (and if it is) in a US, use **IndexOf()**.
- To get the number 01 elements in a List, use the **count** property
- You can use the **Contains()** method to find out if a particular object ISin a IISt
- **foreach** is a special kind of loop thaI will eerate through all of the elements in a list and erecute ecce on n The syntax for a foreach loop IS **foreach (string I in StringList)**. You don't have \0 tell the foreach loop 10 increment by one. it will go through the entire IISt an on its own.



Code Magnels

Can you reconstruct the eeoe snippets to make a working Windows Form that will pop up the message box below when you click a button?

```
private void buttonl_Click(object sender.
EventArgs
```

```
a.RemoveAt(2);
```

```
List<string> a .. new List<string>();
```

```
public void printL (List<string> all
```

```
if (a.contains("two") I {
    a.Add(twopointtwo):
```

```
a.Add(zilch):
a.Add(first);
o.Addtsecondl:
a.Add(third);
```

```
}
```

```
string result • "";
```

```
if la.Contains("three")){
    a.Add("four");
```

```
foreach (string element in . J
{
    result += "\n" + element;
```

```
MessageBox.Show(result I:
```

```
if (a.IndexOf("four") != 4) {
    a.AddlfOurthJ:
```

```
}
```

```
printL(a);
```

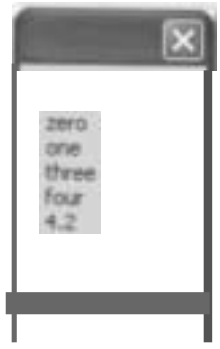
```
string zilch = "zero";
string first = "one":
string second .. "two";
string third = -cnree-:
string fourth .. "4.2";
string twopointtwO .. "2.2";
```

```
zero
one
three
four
4.2
```

cc



Code Magnets Solution




```
private void buttonl_Click(object sender,
EventArgs e){
    List<string> a = new List<string>(I;

    string zilch ="zero";
    string first = "one";
    string second ="two";
    string third - "uhree";
    string fourth ="4.2";
    string tvopointtwo - "2.2";

    a.AddUirstl;
    a.Add(secondl;
    a.Add(thirdl;
    it (i1.Contains(-three")l(
        a.Add(-four"):

    a.RemoveAt(21;1

    if (a.recexor("four") !- 4)
        a.Add(fourth):

    if (a.conta ns
        a.Addltwopointtwo);

    printL(a);
}

publac void printL IList<string> "
    ing result • "";
    foreach (string element in ";
        reeurr += "\n" + element;
    MessageBox.Show(result);
}
```

Can you figure out why "2.2" never gets added to the list, even though it's declared here?

RemoveAt() removes the element at inde·· #2—which is the third element in the list

The printL() method uses a foreach loop to go through a list of strings, add each of the... to one big string, and then show it in a message box

The foreach loop goes through all of the elements in the list and prints them.

there are no Dumb Questions

Q: So why would I ever use an enum instead of a List? Don't they solve the same problem?

A: Enums are a little different than Lists. You can think of enums as a handy way to store *lists of constants* so you can refer to them by name. They're great for keeping your code readable and making sure that you are always using the right variable names to access values that you use really frequently.

A List can store just about anything since it's a list of *objects*, each element in a list can have its own methods and properties. Enums, on the other hand, have to be assigned one of the values types in C# (like the ones on the first page of Chapter 4). So, you can't store reference variables in them.

Enums can't dynamically change their size either. They can't implement interfaces or have methods, and you'll have to cast them to another type to store a value from an enum in another variable. Add all that up and you've got some pretty big differences between the two ways of storing data. But both are really useful in their own right.

Q: OK, it sounds like Lists are pretty powerful. So why would I ever want to use an array?

A: Arrays take up less memory and take less CPU time for your programs. If you're doing something that's really performance-intensive—like the same operation thousands and thousands of times –then you might find that a List will cause your program to slow down significantly. Luckily, you can easily convert any list to an array using the `ToArray()` method... and you can convert an array to a list using one of the overloaded constructors for the `List` object.

Okay, honestly, we're talking about a really, really tiny performance boost. Like if you have to do the same thing millions of times a second, use an array and not a list.

Q: I don't get that "<T>" name "generic". Why is it called a generic collection? Why isn't an array one?

A: A generic collection is a collection object (or a built-in object that gets you to store and manage a bunch of other objects) that's been set up to store only one type.

Q: Okay, that explains the "collection" part. But what makes it "generic"?

A: Supermarkets used to carry generic items that were packaged in big white packages with black type that just said the name of what was inside ("Potato Chips", "Cola", "Soap", etc.). The generic brand was all about what's inside the bag, and not about how it's displayed.

The same thing happens with generic data types. Your <T> will work exactly the same with whatever happens to be inside it. A List of Shoe objects, Card objects, bits, longs, or even other Lists will still act at the container level. So you can always add, remove, insert, etc., no matter what's inside the list.

The term "generic" refers to the fact that even though a specific instance of List can only store one specific type, the List class in general works with any type.

That's what the <T> stuff is all about. It's the way that you tie a specific instance of a List to one type. But the List class as a whole is generic enough to work with ANY type. That's why generic collections are different from anything you've seen so far.

Q: Can I have a list that doesn't have a type?

A: No. Every list—in fact, every generic collection (and you'll learn about a bunch more generic collections in just a minute)--must have a type connected to it. C# does have non-generic lists called ArrayLists that can store any kind of object. If you want to use an ArrayList, you need to include a `"using System.Collections;"` line in your code.

Generic collections are actually a recent addition to C#—they didn't exist in the early versions of the language. But they're so useful that people rarely use non-generic collections any more... which is why we won't be talking much about them.

When you create a new list object, you always supply a **type—that** tells C# what type of data that it'll store. A list can **store** a value type (like int, bool, or string) or a class.

Collection initializers work Just like object initializers

(:# gives you a nice hit of shorthand to cut down on rvping whvn you need I" create a li,1 and imm-diarcly add a bunch of il"m, lu it. When you create a new Lis object, you can use a **collection initializer** ,n give it a starting 1i'1 of i" 'III', h'll add ih"III ;IS soon as IIIr lis i, ,u ';lII'd,

You saw this code a few pages ago—it creates a new List<Shoe> and fills it with new Shoe objects.

```
List<Shoe> shoeCloset = new List<Shoe>();
shoeCloset.Add(new Shoe(l  ( Style = Style.Sneakers, Color = "Black" });
shoeCloset.Add(new Shoe(l  ( Style = Style.Clogs, Color = "Brown" });
shoeCloset.Add(new Shoe(l  ( Style = Style.Wingtips, Color = "Black" });
shoeCloset.Add(new Shoe(l  ( Style = Style.l.oafers, Color = "White" });
shoeCloset.Add(new Shoe(l  ( Style = Style.l.oafers, Color = "Red" });
shoeCloset.Add(new Shoe()  ( Style = Style.Sneakers, Color = "Green" });
```

The same code rewritten using a collection initializer

Collection /.itializers are a **C# 3.0 feature** If you're still using Visual Studio 2005, you should download Visual Studio 2008 Express for free from Microsoft, otherwise this code wo·'t work.

You can create a collection initializer by taking each item that was being added -S-S AddO and adding the.. to the state..ent that creates the list.

```
List<Shoe> shoecloset = new List<Shoe>() {
    new Shoe() { style = Style.Sneakers, Color = "Black" },
    new Shoe() { Style = Style.Clogs, Color = "Brown" },
    new Shoe() { Style = Style.Wingtips, Color = "Black" },
    new Shoe() { Style = Style.Loafers, Color = "White" },
    new Shoe() { Style = Style.Loafers, Color = "Red" },
    new Shoe() { Style = Style.Sneakers, Color = "Green" },
};
```

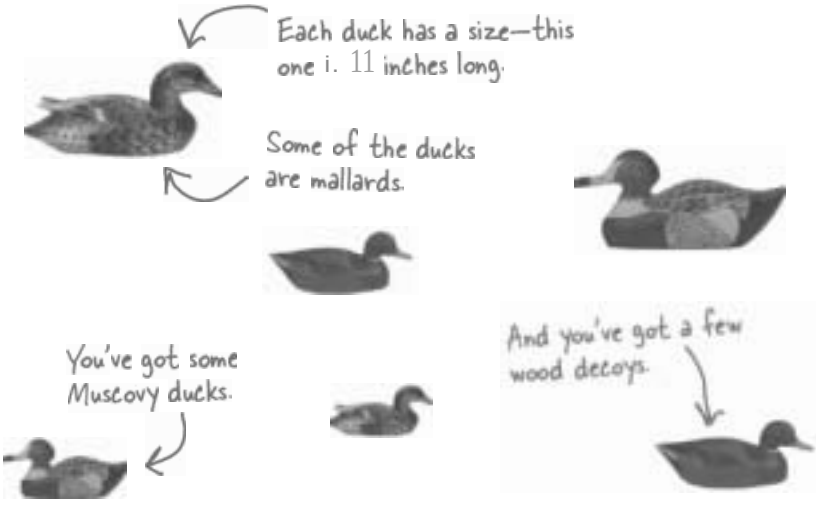
The state..ent to create the list is followed by curly brackets that contain separate new statement, separated by commas

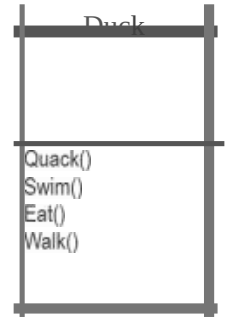
You're not limited to using new state..ents i" the initializer—you can include variables, too

A collection initializer **makes your code more** compact **by** lctting you combinc **creating** a list **with** adding an initial **set** of items.

Let's create a list of Ducks

Here's a duck class that keep tra,'k of your extensive duck colleciton (You *do* collect duc ks. don'! you?)






```
public class Duck {
    public int Size:
    public KindOfDuck Kind;
```

The clas has two public fields It's also got some ..ethods, which we're not showing here.

```
public enum KindOfDuck
    Mallard,
    Muscovy,
    Decoy
```

We'll use an enum called KindOfDuck to keep track of what sort of ducks are in your collection.

Here's the initializer

We've got six ducks. so we'll create a I.i,«)tuck> tha(has a collection initializer " ith six statements. Each statement ill the illiti"liz" r create a new dll,'k. using an object initializer to set each J)ll,'k object's Si,t· and Kind field.

```
List<Duck> Ducks = new ListcDuck>() {
    new Duck()   Kind   KindOfDuck.Mallard, Size   17 i.
    new Duck()   Kind   KindOfDuck.Muscovy, Size   18 i.
    new Duck()   Kind   KindOfDuck.Decoy. Size = 14 }.
    new Duck()   Kind   KindOfDuck.Muscovy, Size   11 i.
    new Duck()   Kind   KindOfDuck.Mallard, Size   14 i.
    new Duck()   Kind   KindOfDuck.Decoy, size = 13 },
};
```

Lists are easy, but SORTING call be tricky

It's uot hard to think about ways to ' 0 11 Ulunl ers or letters. But what do you ,ort two ,I>j,·,'t, on, especially if they have multiple fields? In some case you might want to order objects by the value in a name field, while in oth,'" cases it miuht make sense " ,',rder objects based ,>1) height "r date of birth, [he-re are I,)ts of wavs vou ,'an order things, and]Jists supp(,. t any of them.

You could **sort** a **list** of **ducks** by slza..



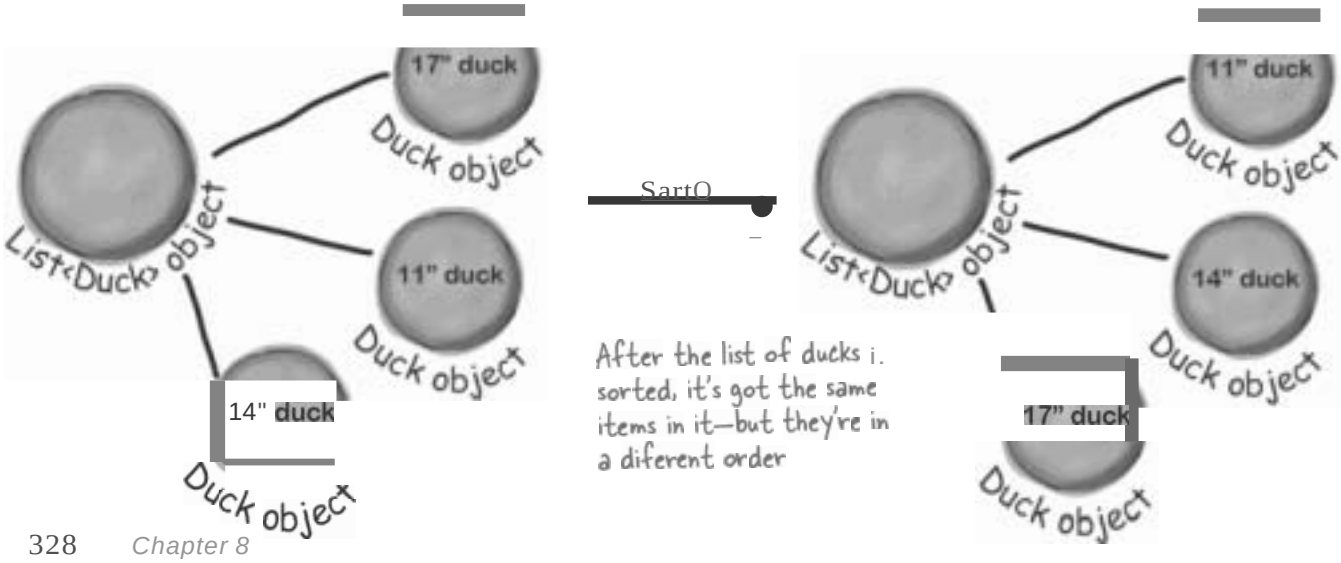


...or by **type.**



lists k.ow how to sort themselves

Every 1i,t comes with a Sort() method that rearranges all of the items in the list to put them ill order. Lists already know how to sort most built-in types ;lllld classes, and it's easy to teach it how to sort your 0"1) classes.



two ways to sort your ducks

The I,i,cS"rt() tlWlhod already knows how to sort any type or class that implements the I Compar**able** interface. That interface has just one member—a method called CompareTo(). Sorl() uses an object's CompareTo() method to compare il I,'ilh other objects, and uses its H'lum value (an illl) '" determii". "hid, comes ti,." .

flm sometimes you need to sort a list " I' objects lhal don't illlpl"nu'lll IComparable, and .NET has .cnmhe-r interface t" help with that. You can pass Sort() an instance of a class that implements IComparer. That interface also has one method. The List's Sort() metl""I uses ,he comparer objec{'s Compare() method to compare pairs "r obje,:ts ill order I" figure out which one comes first ill the sorted list.

You Can **make any class work with the** Lisl's **built**-in SorlO **method** I,y **having** it **implement IComparable** and adding a CompareToO **method.**

An obj ect's Co",pareTo() ",elhod co",pares it with a"other object

(In,-wav In let our List object sort is 1n modify the Duck class I" implement IComparable. To do that, we'd add a CompareTo() meth"I that I"kes a I1"d, reference as a parameter. [I' the duck to compare should (·om,· after the current duck in the sorted list, Compare'To() returns it positive llll1ll1"·l.

Here's a I Iurk class that sorts iw-lf based on duck size:

When you implement Ie omparab'e, you specify the type being compared when r- have the class implement the interface.

```
public class Duck : IComparable<Duck> {
    public int Size;
    public KindOfDuck Kind;

    public int CompareTo(Duck duckToCompare)
    {
        if (this.Size > duckToCompare.Size)
            return 1;
        else if (this.Size < duckToCompare.Size)
            return -1;
        else
            return 0;
    }
}
```

Most CompareTo() methods look a lot like this. This method first compares the Size field against the other duck's Size field. If this duck is bigger, it returns 1. If it's smaller, it returns –1. And if they're the same size, it returns zero.

If you want to sort your list from s..allest to biggest, have CompareTOO return a positive number if it's comparing to a smaller duck, and a negative numbe- if it's comparing to a bigger one



Use **IComparer** to tell your **List how** to sort

Lists have a special interface built illl" the .NET Framework Ihal lets you build your own sorting logic. By implementing the IComparer interface, you are can tell your `List` cxactlv how you want il 10sort your objects. You d" that 1)\-implementing the `Compare ()` method iu the `IComparer` interface. It takes two object parameters, x and y, and returns an Lnt.. If Xis less lhan y, it should return a negative ,-alu,-, II they're equal. il ,h"llid return zer‹·. And if x is greater than ", il should return a positive vahu-

11.,10"" an example or I"ɯ y‹lll'd declare a comparer class (0 compare .lu,'k ohi,'ct, by size:

```
public class DuckComparer_bySize : IComparer<Duck>
{
    public int Compare(Duck x, Duck y)
    {
        if (x.Size < y.Size)
            return -1;
        if (x.Size > y.Size)
            return 1;
        return 0;
    }
}
```

This class implements IComparer, and specifies the type of object it can sort: Duck objects

These will always match: the same type in each.

The Compare() method returns an int, and has two parameters: both of the type you're sorting

You can do whatever types of comparisons you want in the method.

A -1 means object x should go before object y. x is "less than" y.

Positive 1 means object x should go after object y. x is "greater" than y.

0 means that these two objects should be treated as the same (using this comparison calculation).

Your `list` will sort diHerently depending on **how** you implement IComparer.

Create an instance of your Comparer object

When you want to sort using IComparer, you need to create a new instance of the class that implements it. That object exists for one reason—to help List.Sort() figure out how to sort. But like any other (non-static) class, you need to instantiate it before you use it.

We left out the code you already saw a few pages ago to initialize the list

```
List<Duck> ducks = new List<Duck>() {      };
DuckComparer_bySize dc = new DuekComparer_bySize();
ducks.Sort(dc);
```

You'll pass Sort() a reference to the new DuckComparer_bySize object as its parameter.



Sorted smallest to biggest...



Multiple IComparer implementations, multiple ways to sort *your* objects

You can create multiple IComparer classes with different sorting logic depending on what you need to do. Then you can call the comparer you want when you need to sort in that particular way. Here's another duck comparer implementation:

This comparer sorts by duck type. Remember, when you compare the enum Kind, you're comparing their index values.

So Mallard comes before Muscovy, which comes before Decoy.

```
class DuckComparer_byKind : IComparer<Duck> {
    public int Compare(Duck x, Duck y)
    {
        if (x.Kind < y.Kind)
            return -1;
        if (x.Kind > y.Kind)
            return 1;
        else
            return 0;
    }
```

We used the value of the Type that comes from the index value in the enum Duck.Type.

Here's an example of how enums and Lists work together. Enums stand in for numbers, and are used in sorting of lists.

Notice how "greater than" and "less than" have a different meaning here. One type is "greater than" another, just so we have an ordering we can use.

```
DuckComparer_byKind dcKind = new DuckComparer_byKind();
ducks.Sort(dcKind);
```

Sorted by kind of duck...



IComparer call do complex cOllIparisOIlS

(Jne advantage ".,-reating a separate class for sorting your d'k:ks is tha you ,'all build mor(complex logic into that class—and you can add members tha help ,I,·!th"i",· hov the fis, gets sorted.

```
public class DuckComparer : IComparer<Duck> {
    public enum SortCriteria {
        SizeThenKind,
        KindThenSize,
    }

    public SortCriteria SortBy = SortCriteria.SizeThenKind;

    public int Compare(Duck x, Duck 1'1 I
        re (SortBy _ SortCriteria.SizeThenKind)
            if (x.Size > y.Size)
                return 1;
            else it (x.Siz", < y.Size)
                return -1;
            else
                it (x.Kind > y.Kind)
                    return 1;
                else it (x.Kind < y.Kind)
                    return -1;
                else
                    ret urn 0;
        .1••
            if (x.Kind > y.Kind)
                return 1;
            else it (x.Kind < y.Kind)
                return -1;
            else
                if (x.Size > y.Size)
                    return I;
                else it (x.Size < y.Size)
                    return -I;
                else
                    return 0;
```

Here's a more complex class to compare ducks. Its CompareTo() method takes the same parameters, but it looks at the public SortBy field to determine how to sort the ducks.

This enum tells the object which way to sort the ducks.

This if statement checks the SortBy field. If it's set to SizeThenKind, then it f.....t sorts the ducks by size, and then within each size it'll sort the ducks by the.. kind

Instead of just returning *0* if the two ducks are the same size, the comparer checks their kind, and only returns *0* if the two ducks are both the same size and the same k·nd

If SortBy isn't set to SizeThenKind, then the comparer first sorts by the kind of duck. If the two ducks are the same kind, then it compares their size

```
DuckCompllrer de • neW DuckComparer();

dc.SortBy • DuckComparer.sortCrit.eris.KindThenSi re;
ducks.Sort();

dc.sortEly • DuckCompar"r.SortCriteria.SizeThenKind;
ducks.Sort();
```

Here's how we'd use this comparer object. First we'd instantiate it as usual. Then we can set the object's SortBy field before calling ducks.Sort(). Now you can change the way the list sorts its ducks just by changing one field in the object.



create five random cards and then sort them

Console applications that you run from the command prompt use the Console.WriteLine() method a lot, but you can use it in a Windows application too.

1 **Create code to make a jumbled set of cards**

Add a button to a form that creates five random Card objects. After you create each object, use the built-in Console.WriteLine() method to write its name to the output. You can view everything written to the output by selecting "Output" from the View menu while the program's running.

2 **Create a class that implements IComparer<List> to sort the cards**

Here's a good chance to use that IDE shortcut to implement an interface:

```
public class CardComparer_byValue : IComparer<Card>
```

Then click on IComparer and hover over it. You'll see a box appear underneath it, when you click on the box, the IDE pops up a window

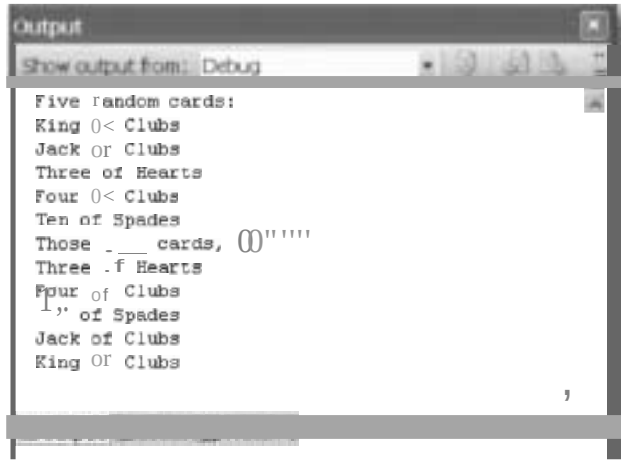


If you click on "Implement interface IComparer<Card>", the IDE automatically fills in all the methods and properties that you need to implement. In this case, it creates an empty Compare() method to compare two cards, x and y. Write the method so that it returns 1 if x is bigger than y, -1 if it's smaller, and 0 if they're the same rank. In this case, make sure that any king comes after any jack, which comes after any four, which comes after any ace.

3 **Make sure the output looks right**

Here's what your small window should look like after you click the button.

When you use the built-in Console WriteLine() method, it adds a line to this output window



Your IComparer object needs to sort the cards by value, so the cards with the lowest values are first in the list.



Create five random cards and then sort them.

Here's the "guts" of the card sorting, which uses the built-in List.Sort() method. Sort() takes an IComparer object, which has one method: Compare(). This implementation takes two cards and first compares their suits, then their values.

```
public class CardComparer_byValue : IComparer<Card> {
    public int Compare(Card x, Card y) {
        if (x.Value < y.Value)
            return -1;
        }
        if (x.Value > y.Value)
            return 1;
        }
        if (x.Suit < y.Suit)
            return -1;

        if (x.Suit > y.Suit)
            return 1;

        return 0;
```

If x has a bigger suit, return 1. If x's suit is smaller, return -1. Remember, both return statements end the method immediately.

These statements only get executed if x and y have the same suit—that means the first two return statements weren't executed.

If none of the other four return statements were hit, the cards must be the same—so return zero.

Here's a generic List of Card objects to store the cards. Once they're in the list, it's easy to sort the.. using an IComparer.

```
private void button1_Click(object sender, EventArgs e)
{
    Console.WriteLine("Five random cards:");
    List<Card> cards = new List<Card>();
    for (int i = 0; i < 5; i++)

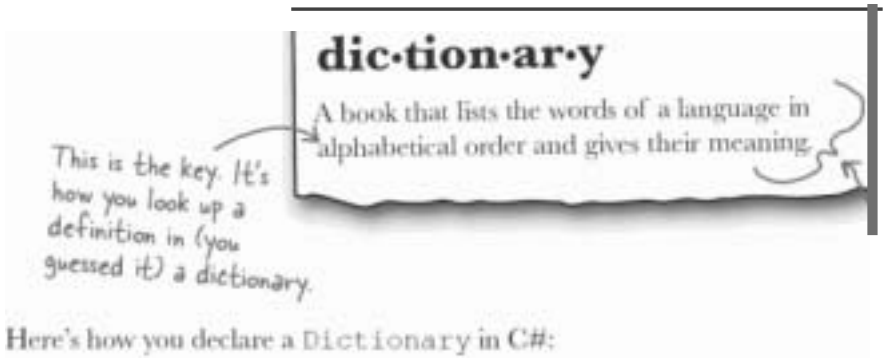
        cards.Add(new Card((Card.Suits)random.Next(4),
                           (Card.Values)random.Next(11, 14)));
        Console.WriteLine(cards[i].Name);


    Console.WriteLine("Those same cards, sorted:");
    cards.Sort(new CardComparer_byValue());
    foreach (Card card in cards)

        Console.WriteLine(card.Name);
```

Use a dictiol1ary to store keys al1d values

.\ list is lik,' a big long page 1\11 of names. B"I what if you want, for each nauu-. an address? Or for cv-rv car in the garage li'l. you wanr details ab«,UI thar car? You need a **dictionary**. .A dictionary lets you take a special value—the k..**y**—and associate that key with a bunch "I' data—the vatu**e**. A"d 0"" more thing: a specific key can only app..ar on**ce** ill anv dictionarv.



Here's how you declare a `Dictionary` in C#:

```
Dictionary <Tkey, TValue> kv = new Dictionary <TKey, TValue>();
```

These are like List<T>. The <T> ...eans a type goes in there. So you t an declare one type for the key, and another type for the value.

The first item in the angle brackets is always the key, and the second is always the data.

.\nd here's a `Dictionary` in action:

```
private void buttonl_Click(object sender, Ev..ntArgs e)
{
    Dlctionary<string, ..tring> wordDefinition
        n..w Dictionary<..tr1ng, strlng>O;

    wordDefinition.Add ("Dictionary", "A book that list.. the words of a "
                        + "language in alphabetical ord..r and g1v..s the1r meaning");
    wordDefinition.Add ("Key", "A thing that provides a means of gaining access to
                        + "our understanding something,");
    wordDefinition.Add ("Value", "A magnitude, quantity, or number.");

    if (wordDefinition.ContainsKey("Key")) {
        MessageBox.Show(wordDefinition["Key"]);
    }
```

This dictionar\ ha' string values for keys, and strings as the value. It's like a real dictionary: term, and definition.

The Add() ethod is how ... you add keys and values to the dictionary

Add() takes a key, and then the value

ContainsKey() tells you if a key is in the dictionary. Handy, huh?

Here's how you get the value for a key. It looks kind of like an array index—get the value for the key at this index.

The Dictionary Functionality Rundown

Dictionaries are a "lik" Lists. Both types are flexible in letting you work with lots of data types, and also co"l" with lots of built-in functionality. Here are the basic Dictionary methods:

- **Add an item.**
  You can add an item to a dictionary by passing a key and a value to its the Add() method,

```
Dictionary<string, string> myDictionary = new Dictionary<string, string>();

myDictionary.Add("some key", "some value");
```

- **Look up a value using its key.**
  The most important thing you'll do with a dictionary is look up values – which makes sense, because you stored those values in a dictionary so you could look them up using their unique keys.

```
string lookupValue = myDictionary["some key"];
```

- **Remove an item.**
  Just like a List, you can remove and item from a dictionary using the Remove() method, All you need to pass to the Remove method is the Key value to have both the key and the value removed,

```
myDictionary.Remove("some key");
```

Keys are unique in a Dictionary; any key appears exactly once. Values can appear any number of times—two keys can have the same value. That way, when you look up or remove a key, the Dictionary knows what to remove.

- **Get a list of keys.**
  You can *get* a list of all of the keys in a Dictionary using a `KeyCollection` and loop through it using a foreach loop. You'll usually use a Keycollecnon like this:

```
foreach (string key in myDictionary.Keys) { ... };
```

Keys is a property of your dictionary object. This particular dictionary has string keys, so Keys is a collection of strings

- **Get a list of values.**
  You can get a list of all of the values in a Dictionary using a `ValueCollection`, Most of the time, you use a ValueCollection with a foreach loop too:

```
foreach (string value in myDictionary.Values) { ... };
```

C Since this dictionary has string values, the foreach identifier will be a string

Your key alld value call be differellt types. too

1Dict,onaries are really versatile and can h,old just 'IDout anything, Ii,au strings 10numbers ,,,,,,l ''en objects. Here's an example of a dictionary that's storing an integer as a key and a duck objec1as a value. The Duck object has a Size field and Types "ll""" th,u are set inside· iI' constructor.

Here's where the dictionary is declared It'll store nu..bers and ducks. We'll add each of the ducks to the dictionary, giving it a unique ID nu..ber as the key

```
Oictionary<int, Duck> duckDictionary _ new Dictionary<int, Duck>();

duckDictionary Add(5155, new DuckO    Kind _ KindOfDuck.Mallard, Size = 15 I);

duckDictionary Add(6256, new DuckO    Kind _ KindOfDuck.Mallard, Size = 14 I);

duckOictionary.Add(2799, new Duck II  Kind   KindOfDuck.Mallard, Size . 13 I);
```

The Count property tells how many key-value pairs are in the Dictionary

```
int howMany = duckDictionary.Count;

Console.WriteLine("There are (O) ducks.", howMany);
```

If r- need to pull the keys and values out of a dictionary, you can use a KeyValuePair<>.

This loop assigns the current key/value pair to idDuck, one at a time, through the whole loop.

```
foreach (KeyValuePa1r<int, Duck> idDuck in duckDictionary)

        Consol...WriteLine("Key/value pair: (O): {1}, {2}",

            IdDuck K..y, idDuck.Valu.., S1z", idDuck.Valu.., K1nd.ToStr1ng{) I;
```

Every KeyValuePair has a key and a value Since the value is a Duck, you can use its field°.

```
foreach (Duck duck in duckDictionary.Value$l

    Consol...Writ eL1n..("Duck size: {0}", duck.Size);
```

This foreach loop goes through all of the values in the dictionary.

```
foreach (int key in duckOictionary.Key$l

    Con$ole.WriteLine{"ID Nwnber: {0}", key);
```

And this foreach loop pulls each of the keys out of the dictionary

```
Output
Show output from: Debug
Key/v..lue pair: 5155: 15, Mall  d
Key/v..lue pair: 6256: 14, Mall  d
Key/v..lue pair: 2799: 13, Mall  d
Duck size: '5
Duck size: 14
Duck size: 11
ID Number: 5155
ID Number: 6256
ID Number: 2799
```

Here's the output that this code writes to the console



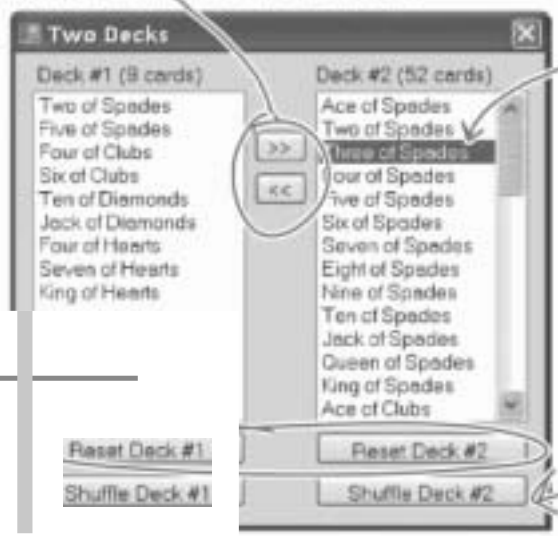
Build a form that lets you move cards between two decks

You've built a card class already. Now it's time to build a class to hold any number of cards, which we'll call Deck. A real-life deck has 52 cards, but the Deck class can hold any number of cards—or zero cards at all.

Then you'll build a Rummy that shows you the contents of two Deck objects. When you first start the program, deck #1 has up to 10 random cards, and deck #2 is a complete deck of 52 cards, both sorted by suit and then value—and you can reset either deck to its initial state using two Reset buttons. The form also has buttons (labeled "<<" and ">>") to move cards between the decks.

You can use a button's Name property to give it a name to make your code easier to read. Then when you double-click on the button, its event handler is given a matching name.

These buttons are named moveToDeck2 (top) and moveToDeck1 (bottom). They move cards from one deck to the other.



Use two ListBox controls to show the two decks. When the moveToDeck1 button is clicked, it moves the selected card from deck #2 to deck #1.

These buttons are named shuffle1 and shuffle2. They call the appropriate Deck.Shuffle() method, and then redraw the deck.

Each of the reset1 and reset2 buttons first calls the RedrawDeck() method and then the ResetDeck() method.

In addition to the event handlers for the six buttons, you'll need to add two methods for the form. First add a ResetDeck() method which resets a deck to its initial state. It takes an int as a parameter: if it's passed 1, it resets the first Deck object by initializing it to an empty deck and a random number of up to 10 random cards; if it's passed 2, it resets the second Deck object so that it contains a full 52-card deck. Then add this method:

```
private void RedrawDeck(int DeckNumber)
{
    if (DeckNumber == 1)
    {
        listBox1.Items.Clear();
        foreach (string cardName in deck1.GetCardNames())
            listBox1.Items.Add(cardName);
        label1.Text = "Deck #1 (" + deck1.Count + " cards)";
    }
    else
    {
        listBox2.Items.Clear();
        foreach (string cardName in deck2.GetCardNames())
            listBox2.Items.Add(cardName);
        label2.Text = "Deck #2 (" + deck2.Count + " cards)";
    }
}
```

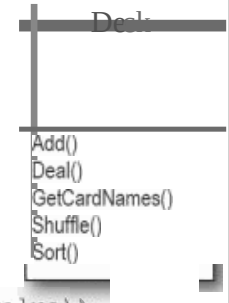
The RedrawDeck() method updates the two listbox controls with whatever happens to be in the two Deck objects.

Take a look at how we used a foreach loop to add each of the cards in the deck to the listbox.

9 Build the Deck class

Where you have the declarations for a class without the implementation, it's called a "skeleton".

Here's the skeleton for the Deck class. We've filled in several of the methods for you. You'll need to finish it by writing the Shuffle() and GetCardNames() methods, and you'll have to get the Sort() method working. We also added two useful **overloaded constructors**: one that creates a complete deck of 52 cards, and the other that takes an array of Card and loads them into the deck.



The Deck stores its cards in a List—but it keeps it private to make sure it's well-encapsulated

If you don't pass parameters into the constructor, it creates a complete deck of 52 cards

```
public class Deck {
    private List<Card> cards;
    private Random random = new Random();

    public Deck() {
        cards = new List<Card>();
        for (int suit = 0; suit <= 3; suit++)
            for (int value = 1; value <= 13; value++)
                cards.Add(new Card((Card.Suits)suit, (Card.Values)value));
    }

    public Deck(Card[] initialCards) {
        cards = new List<Card>(initialCards);
    }

    public int Count { get { return cards.Count; } }

    public void Add(Card cardToAdd) {
        cards.Add(cardToAdd);
    }

    public Card Deal(int index) {
        Card CardToDeal = cards[index];
        cards.RemoveAt(index);
        return cardToDeal;
    }

    public void Shuffle() {
        // this method shuffles the cards by rearranging them in a random order
    }

    public string[] GetCardNames() {
        // this method returns a string array that contains each card's name
    }

    public void Sort() {
        cards.Sort(new CardComparer_bySuit());
    }
}
```

This overloaded constructor takes one parameter—an array of cards, which it loads as the initial deck.

The Deal method deals one card out of the deck—it removes the Card object from the deck and returns a reference to it. You can deal from the top by passing it 0, or deal from the middle of the deck by passing in the index of the card to deal.

ilist: The ListBox control's SelectedIndex property will be the same as the index of the card in the list. You can pass it directly to the Deal() method. If no card is selected, it'll be less than zero. In that case, the MoveToDeck button should do nothing.

You'll need to write the Shuffle() method, the GetCardNames() method, and add a class that implements IComparer to make the Sort() method work. And you'll need to add the Card class you already wrote. If you're using "Add Existing Item" to add it, don't forget to change its namespace.

Another hint: The form makes it really easy to test your Shuffle() method. Keep clicking the "Reset Deck #1" button until you get a three-card deck. That'll make it easy to see if your shuffling code works.



Build a class to store a deck of cards, along with a form that uses it

Here's the constructor that creates a complete deck of 52 cards. It uses a nested for loop. The outside one loops through the four suits. That means the inside loop that goes through the 13 values runs four separate times, once per suit.

```
public class Deck {
    private List<Card> cards;
    private Random random = new Random();

    public Deck() {
        cards = new List<Card>();
        for (int suit = 0; suit <= 3; suit++)
            for (int value = 1; value <= 13; value++)
                cards.Add(new Card((Card.Suits)suit, (Card.Values)value));
    }

    public Deck(Card[] initialCards) {
        cards = new List<Card>(initialCards);
    }

    public int Count { get { return cards.Count; } }

    public void Add(Card cardToAdd)
    {
        cards.Add(cardToAdd);
    }

    public Card Deal(int index)
    {
        Card CardToDeal = cards[index];
        cards.RemoveAt(index);
        return CardToDeal;
    }

    public void Shuffle()
    {
        List<Card> NewCards = new List<Card>();
        while (cards.Count > 0)
        {
            int CardToMove = random.Next(cards.Count);
            NewCards.Add(cards[CardToMove]);
            cards.RemoveAt(CardToMove);
        }
        cards = NewCards;
    }

    public string[] GetCardNames()
    {
        string[] CardNames = new string[cards.Count];
        for (int i = 0; i < cards.Count; i++)
            CardNames[i] = cards[i].Name;
        return CardNames;
    }

    public void Sort()
    {
        cards.Sort(new CardComparer_bySuit());
    }
```

Here's the other constructor—this class has two overloaded constructors, each with different parameters.

The Add and Deal methods are pretty straightforward—they use the methods for the Cards list. The Deal method removes a card from the list, and the Add method adds a card to the list.

Your GetCardNames() method needs to create an array that's big enough to hold all the card names. This one uses a for loop, but it could also use foreach

The Shuffle() method creates a new instance of List<Cards> called NewCards. Then it pulls random cards out of the Cards field and sticks them in NewCards until Cards is empty. Once it's done, it resets the Cards field to point to the new instance. The old instance won't have any more references pointing to it, so it'll get collected by the garbage collector.

```
class CardComparer_bySuit : IComparer<Card>
{
    public int Compare(Card x, Card y)
    {
        if (x.Suit > y.Suit)
            return 1;
        if (x.Suit < y.Suit)
            return -1;
        if (x.Value > y.Value)
            return 1;
        if (x.Value < y.Value)
            return -1;
        return 0;
```

Sorting by suit is a lot like sorting by value. The only difference is that in this case the suits are compared first, and then the values are compared only if the suits match.

Instead of using if/else if, we used a series of if statements. This works because each if statement only executes if the previous one didn't—otherwise the previous one would have returned.

```
    Deck deck1;
    Deck deck2;
    Random random = new Random();

    public Form1() {
        InitializeComponent();
        ResetDeck(1);
        ResetDeck(2);
        RedrawDeck(1);
        RedrawDeck(2);
    }
```

The form's constructor needs to reset the decks, and then it draws the...

```
    private void ResetDeck(int deckNumber) {
        if (deckNumber == 1) {
            int numberOfCards = random.Next(1, 11);
            deck1 = new Deck(new Card[] { });
            for (int i = 0; i < numberOfCards; i++)
                deck1.Add(new Card((Card.Suits)random.Next(4),
                                   (Card.Values)random.Next(1, 14)));
            deck1.Sort();
        }
        else
            deck2 = new Deck();
    }
```

To reset deck #1, this method first uses random.Next() to pick how many cards will go into the deck, and then creates a new empty deck. It uses a for loop to add that many random cards. It finishes off by sorting the deck. Resetting deck #2 is a easy—just create a new instance of Deck().

You've already got the RedrawDeck() method from the instructions.

We're not done yet—flip the page!



Exercise Solution (continued)

Naming your control. ..akes it a lot easier to read your eede If these were called button1_Click, button2_Click, etc., r- wouldn't know which button's eede you were looking at!

Here's the rest of the code for the form.

```
private void reset1_Click(object sender, EventArgs e) {
    ResetDeck(1);
    neurevrecx(1);

private void reset2_Click{object sender, EventArgs e) {
    ResetDeck(2);
    RedrawDeck(2);

private void shuffle1 Click(object sender, EventArgs e) {
    decki.Shuffle();
    RedrawDeck(1);

private void shuffle2 Click(object sender, EventArgs e) {
    deck2.Shuffle();
    RedrawDeck(2);

private void moveToDeck1 Click(object sender, EventArgs e) {
    if (listBox2.SelectedIndex >= 0)
        it (deck2.Count > 0) {
            deck1.Add(deck2.Deal(listBox2.SelectedIndex));

    RedrawDeck(1);
    RedrawDeck(2);

private void moveToDeck2_Click(object sender, EventArg5 e) {
    it (listBox1.SelectedIndex >= 0)
        if (deck1.Count > 0)
            deck2.Add(deck1.Ded1(listBox1.SelectedIndex));
    RedrawDeck(1);
    RedrawDeck(2);
```

These buttons are pretty simple—first reset or shuffle the deck, then redraw it.

You can use U., ListBox control's SelectedIndex property to figure out which card the user selected and then move it from one deck to the other. (If it's less than zero, no card was selected, so the button does nothing.) Once the card' ed, both deeks need to be redra .

You call build your OWII overloaded methods

You've been using ov..r1oad..d **m..thods** and even an overloaded constructor that were part of lh,' built-in .NET Framework classes and objects, so you ('an already see how useful they an', Wouldn't it be cool if you comid huild overloaded meth.ods into "'ur own classes? Well, you run-cand it's easy! .\11 you need ro do i, write two "r more rlll'rh"d, that have the same ""n",I'UI take difTnl'1l1 parameters



O **Create** a **new** project and add **the** *Card* class to **it.**

You can d" this easily by right-clicking on the project in Ihl' S"lution Explorer and selecting "Existing Item" from the Add """Ill(, The IIlE will make a copy <Jr lhe class and add illl> the project. The file will still hav.. lhe **namespac**e from th.. old **project**, '0 g(to till' lop of the Card.cs Iii,' and change the `namespace` li,1<' to match the name (,I'the new)roiec r you created.

If you do'i't do this, you'l' only be able to access the Card class by specifying its na··espace (like oldnamespace Card).

O Add **some** new **overloaded** methods to **the** card class.

Create two `static DoesCardMatch()` methods. '111< lir,l one ,hould ,-h,',-ka card's ' uil, The """<Jml ,hould check its value. Both return true onlv if the card matches.

```
public static bool DoesCardMatch(Card CardToCheck, Card.Suits Suit) {
   if (CardToCheck.Suit == Suit)
      return true;
     else {
      return false;
```

Overloaded methods don't have to be static, but it's good to get a little practice writing static methods.

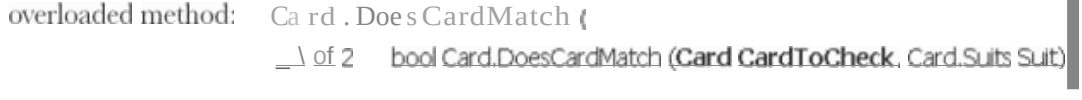
```
pUblic static bool DoesCardMatch(Card CardToCheck, Card.Values Value) {
   if (CardToCheck.Value == Value) {
      return true;
     else {
      return false;
```

O Add a **button** to **the** form to use **the** new methods.

Add lhi, code to the button:

```
Card cardToCheck    new Card(Card.Suits.Clubs, Card.Values.Three);
bool doesltMatch    Card.DoesCardMatch{cardToCheck, Card.Suits.Hearts);
```

As soon as you type "DoesCardMatch{" til(' IIlE will show you that you really did build an overloaded method: `Card.DoesCardMatch(`



Take a minute and play around with the two methods so you can get used to overloading.



Long Exercise

Build a game of Go Flsh1 that you can play against the computer

This exercise is a little different.. .

There's a good chance ,ha, you're learning C# because you want a job as a professional developer. That's " hy we modeled ,hi, r-x..rcise after a professional assignment. \When you're working as a I'" ogrammer on a team, yc.u dc""1 usuallv build a complete I'rogram from start I" lini,h, """"„1. yc'\l'1l build a *piece* ,,[a bigger program. So we're going to give you " puzzle that's got ,,,,,,,. ,,[the piece already lill"d ill. The ""d,' for the li'l'm j, all on the next page in step #3. You just have to type il in which may seem like you've got a great head start, hili it m,'an, that your classes ***hav to work with thaI code*** .\nd Ihal can h,· a challenge!

O

Start with the spec

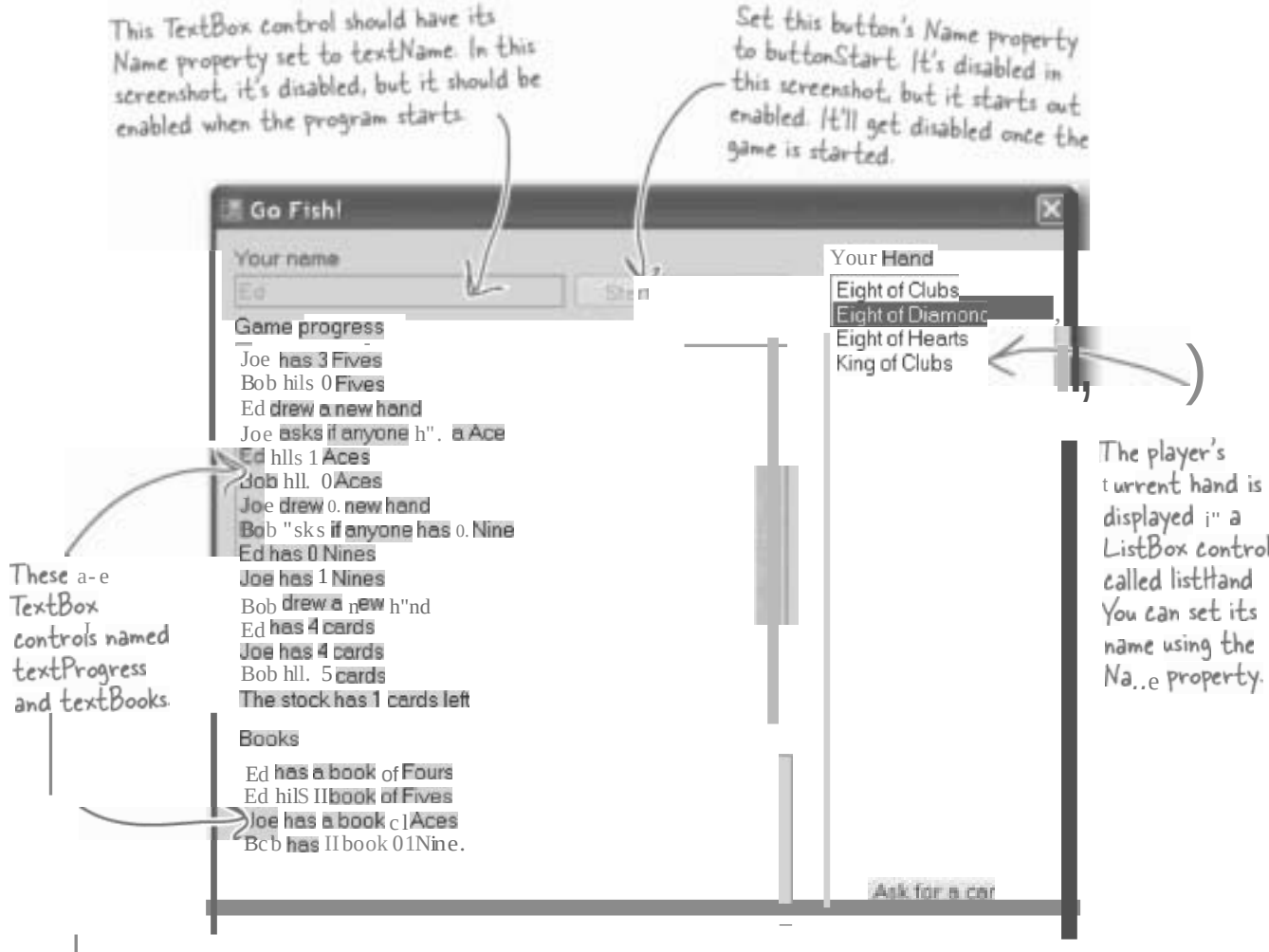
Every professional software I'I'"j","1 starts with a specification, and ,h i, one is "0 excepuon. y,,"'ll be building a game "I" Ihe classic card game ***Go Fish!*** Different people play the game bv slightly different rules, sc here's a recap "I" the rules voull be using:

- The game starts with a deck "f 52 cards. Five cards arr- dealt I" each player. The pile "I" cards that's left after everyone's d..alt a hand is called ,h,' **stock**. Each player ,ak," turns asking for a value ("Do you have any sevens?"). .Any oth"r playe holding cards with that value 111'"1 hand Ih..1ll over. If nobody has a card with ,h"l \'al"l', ,he" lhe player IIH" "go li,h" by taking a card from the stock
- The goal ,,[the game is '0 make books. where a book j, the complete set 01" all four cards that have the same value. T h,' player with the mos books at the "lid ,,[Ih" game is the "i,"'er. As soc»ll as a player collects a book, he places it face-up 0 ll tln- table so all th.. Oilie]'players can see what books evervone else I"",
- \\'1ne11 a player place a book "" II", table, that may cause him to nlll "ut ,,[cards. If il does then he has to draw five more cards from the "',. 'k, I!' there are fewer than five cards left in Ihe stock, he takes a ll ,,[them. The game is over as soon as the stock i, »ut "I" cards. T h,' winner is then d""'"ll based '"' whoever has the mos books.
- ★ For ,hi, computer version 01" Go Fi, h. there are two computer players and " lit' h"llIa" player. Every round starts with Ihe h'"I1'lll playe selecting one of the cards ill his hand, which i, display..rl .u all tim,'" H,' doe ,hi, by choosing one ,,[the cards and indicating Iha, he will ask for a card. Then the two ccenpuu-r player will ask for their cards. The results ,,[each round will be displayed. This will repeat IIHlil there's a winner.
- ★ The game will take care ,,[all ,,[the trading ,,[cards and pulling " ut "I" books automatically. (1m',' there's a winner, the game i, over. The game displays Ihe 11,1111,' or the winner (or winners, ill "a,,' 01" a lie). No other actic"I can be taken the player will have to restart the program in order «, start a new game.

II you don't know what you're l,uitJing before you start, then how would you know when you're done? That's why most professional sojtware projects start with a specification that lells you what you're going to l,uilJ.

Build the form

Build the form for the Go Fish game. It should have a ListBox control for the player's hand, two TextBox controls for the progress of the game, and a button that the player can click to ask for a card. To play the game, the user will select cards from their hand and click the button to ask the computer players if you have that card.



• We're not done yet—flip the page!

Long Exercise (continued)



3 **Here's the code for the form**

Enter it exactly like you see here. The rest of the code that you write yourself will have to work with it.

```
public partial class forml : form
    pUblic Forml () (
        InitializeComponent();

        private Game game;

        private void buttonStart_Click(object sender, Eventllrgs e) (
            if (String.IsNullOrErnpty(textName.Text))(
                MessageBox.Show("Please enter your name", "Can't start the game yet"),
                return;

            game _ new GameltextName.Text, new string!] ( "Joe", "Bob" }, textProgress);
            buttonStart.Enabled - false;
            textName.Enabled - false;
            buttonAsk,Enabled _ true;
            UpdateForm();
        }

        private void UpdateForm() {
            listHand.Items,Clear();
            foreach (String cardNarne in game.GetPlayerCardNames())
                listHand.Iterns.Add(cardNamel;
            rextBooks.Text _ game.DescribeBooks();
            textProgress Text += game.DescribePlayerHands();
            textProgress SelectionStart = textProgress.Text.Lengt h;
            textProgress ScrollToCaret();

        prrvate voac buttonAsk Cl 1cl« objject sender, Eventllrgs el (
            textProgress.Text _ ....,
            if (listHand.SelectedIndex < 0) {
                MessageBox Show("Please select a card").
                return;
            ,
            if (game.PlayOneRound(listHand.SelectedIndex))
                textProgress.Text += "The winner is... " + game.GetWinnerName();
                textBooks.Text _ game.DescribeBoo ks(),
                buttonAsk.Enabled    false;
              else
                UpdateForm();
```

This is the only class that the form interacts with. It runs the whole game.

The Enabled property enables or disables a control on the form.

When you start a new ga..e it creates a new instance of the Ga..e class, enables the "Ask" button, deables the "Start Ga...e" button, and then redraws the form

Thi ethod a..d cle repopulates the ListBox that holds the player' hand, and then updates the 1 text boxes

Using SelectionStart a..d ScrollToCaret() like this scrolls the text box to the end, so if there's too much tex to display at once it sc ll. down to the bottom

The SelectionStart line moves the flashing textbox cursor to the end, and once it's ...ved, the ScrollToCaret() ethod scrolls the text box d to the cursor.

The player se ects one of the car ds and clicks the "Ask" button to see if any of the oth e- players have a card that matches its value The Gam e c lass plays a round using the PlayOneRound() ..ethod.

You'll **need this** code, too

You'll need the code you wrote before for the `Card` class, the `Deck` class and the `CardComparer_byValue` class. But you'll need to add a few more methods to the Deck class... and you'll need to understand them in order to use them.

```
public Card Peek(int cardNumber)
    return cards[cardNumber];
```

The Peek() method lets you take a peek at one of the cards in the deck without dealing it.

```
public Card Deal() {
    return Deal(0);
```

Someone overloaded Deal() to make it a little easier to read. If you don't it pass any parameters, it deals a card off the top of the deck.

```
public bool ContainsValue(Card.Values value)
    foreach (Card card in cards)
        if (card.Value == value)
            return true;
    return false;
```

ContainsValue() method searches through the entire deck for cards with a certain value, and returns true if it finds any. Can you guess how you'll use this in the Go FISh game?

```
public Deck PullOutValues(Card.Values value) {
    Deck deckToReturn = new Deck(new Card[] { });
    for (int i = cards.Count - 1; i >= 0; i--)
        if (cards[i].Value == value)
            deckToReturn.Add(Deal(i));
    return deckToReturn;
```

You'll use the PullOutValues() method when you build the code to get a book of cards from the deck. It looks for any cards that match a value, pulls them out of the deck, and returns a new deck with those cards in it.

```
public bool HasBook(Card.Values value)
    int NumberOfCards = 0;
    foreach (Card card in cards)
        if (card.Value == value)
            NumberOfCards++;
    if (NumberOfCards == 4)
        return true;
    else
        return false;
```

The HasBook() method checks a deck to see if it contains a book of four cards of that value was passed as the parameter. It returns true if there's a book in the deck, false otherwise.

```
public void SortByValue() {
    cards.Sort(new CardComparer_byValue());
}
```

The SortByValue() method sorts the deck using the Comparer_byValue class.

Still not done—flip the page!

Long Exercise (continued)



0 Now comes the HARD part: Build the Player class

Here's an instance of the Player class for each of the three players in the game. They get created by the buttonStart button's event handler.

```
public class Player
{
    private string name;
    public string Name { get { return name; } }
    private Random random;
    private Deck cards;
    private TextBox textBoxOnForm;

    public Player(String name, Random random, TextBox textBoxOnForm) {
        // The constructor for the Player class initializes four private fields, and then
        // adds a line to the TextBox control on the form that says, "Joe has just
        // joined the game" - but use the name in the private field, and don't forget to
        // add line break ("\r\n") at the end of every line you add to the TextBox.

    public List<Card.Values> PullOutBooks() { } // see the facing page for the code

    public Card.Values GetRandomValue() {
        // This method gets a random value-but it has to be a value that's in the deck!

    public Deck DoYouHaveAny(Card.Values value) {
        // This is where an opponent asks if I have any cards of a certain value
        // use Deck.PullOutValues() to pull out the values. Add a line to the TextBox
        // that says, "Joe has 3 sixes" - use the new Card.Plural() static method

    public void AskForACard(List<Player> players, int myIndex, Deck stock)
        // Here's an overloaded version of AskForACard() - choose a random value
        // from the deck using GetRandomValue() and ask for it using AskForACard()

    public void AskForACard(List<Player> players, int myIndex, Deck stock, Card.Values value)
        // Ask the other players for a value. First add a line to the TextBox: "Joe asks
        // if anyone has a Queen". Then go through the list of players that was passed in
        // as a parameter and ask each player if he has any of the value (using his
        // DoYouHaveAny() method). He'll pass you a deck of cards - add them to my deck.
        // Keep track of how many cards were added. If there weren't any, YOU'll need
        // to deal yourself a card from the stock (which was also passed as a parameter),
        // and you'll have to add a line to the TextBox: "Joe had to draw from the stock"
    }
    // Here's a property and a few short methods that were already written for you
    public int CardCount { get { return cards.Count; }
    public void TakeCard(Card card) { cards.Add(card); }
    public string[] GetCardNames() { return cards.GetCardNames(); }
    public Card PeekAt(int cardNumber) { return cards.PeekAt(cardNumber); }
    public void SortHand() { cards.SortByValue(); }
```

Look closely at each of the comments—they tell you what the methods are supposed to do. Your job is to fill in the methods.

That Peek() ...ethod we added to the Deck class will come in handy. It lets the program look at one of the cards in the deck by giving its index .....ber, but unlike Deal() it doesn't remove the card

```
public List<Card.Values> PullOUtBooks() {
    List<Card.Values> Books - new List<Card.Values>();
    for (int i = 1; i <_ 13: i++) {
        Card.Values value - (Card.Values)i;
        int hm;Many _ 0:
        for (int card - 0: card < cards.Count; card++)
            if (cards.Peek(card).Value __ value)
                howMany++:
        if (howMany __ 4) {
            Books.Add(value);
            tor (int card _ cards.Count - I: card >= 0: card--)
                cards.Deal(card);

    return Books;
```

You'll have to build *TWO* eve-leaded versions of the AskForACard() ..ethod The first one is used by the opponents when they ask for cards—it'll look th··ough their hands and find a card to ask for. The second one is used when the player asks for the card. Both of them a0k EVERY other player (both computer and human) for any cards that ..atch the value.

O You'll **need** to add this **method** to **the** Card eless

It's a static method 10take a valtu- and n'III1'n its plural—that way a len will retum "Ten,'· bill a six will rc lurn "Sixes" (with "es" on the end). Since it's static, you call il with Ihl" class name—Card . PIlira I ()—and nol from an i"'I""cc·,

```
public partial class Card (
    public static string Plural(Card.Values value) {
        if (value -- Values.Six)
            return "Sixes";
        else
            return villue.ToString O • "s";
```

Nearly there–keep flipping!

Long Exercise (continued)



7 **The rest of the job: Build the Game closs**

Ih,' I,"4ll keeps one mstance 01 Game. It manages th.. game play. Lo«,k closelv at I",,, it's used in the form.

```
public class Game (
    private List<Player> players;
    private Dictionary<Card,Value5, Player> books;
    private Deck stock;
    privat.e TextBox textBoxOnForm;

    public Game(string playerName, stringll opponentNames, Te><tBox textBoxOnForm) !
        Random random - new Random!);
        this.textBoxOnForm • textBoxOnForm;
        players - new List<Player>();
        players.Add(new Player(playerName, random, textBoxOnForm» ;
        foreach (string player in opponentNames)
            players.Add(new Player(player, random, textBoxOnForm» ;
        books - new Dictionary<Card.Values, Player>();
        stock _ new Deck();
        Deal();
        players[0J•SortHand();

    private void Deal() (
       // This is where the game starts - this method's only called at the beginning
       // ot the game. Shuffle the stock, deal five cards to each player, then use a
       // foteach loop to call each player's PullOutBooks() method.

    public bool PlayOneRound(int selectedPlayerCardl (
       // Play one round of the game. The parameter is the card the player selected
       // from his hand - get its value. Then go through all of the players and call
       // each one's AskForACard() methods, starting with the hllman player Ivhots at
       // at index zero in the Players list - make sure he asks for the selected
       // card's value). Then call PullOutBooks() - if it returns true, then the
       // player ran our of cards and needs ro draw a new hand, After all the players
       // have gone, sort the human player's hand (so it looks nice in the form).
       // Then check the stock to see if it.ts out of cards. If it is, reset the
       // TextBox on the form to say, "The stock is out of cards. Game over!" and return
       // true, oenervtse, the game isn't over yet, so return false.

    public bool PullOutBooks(Player player)
       // Pull out a player's book5. Return true if the player ran out of cards, otherwise
       // return false. Each book is added to the Books dictionary. A player runs out of
       // cards when he's used all of his cards to make books-and he wins the game.

    public string DescribeBooks() {
       // Return a long string that describes everyone's books by looking at the Books
       // dictionary: "Joe has a book of sixes. (line break) Ed has a book of Aces."
```

```
public string GetWinn..rName() {
        // This method is called at the end of the game. It uses its own dictionary
        // (Dictionary<string, int> winners) to keep track of how many books each player
        // ended up with in the Books dictionary. First it uses a foreach loop
        li on Books.Keys -- foreach (Card.Values value in Books.Keys) -- to populate
        // its winners dictionary with the number of books each player ended up with.
        // Then it loops through that dictionary to find the largest number of books
        // any winner has. And finally it makes one last pass through winners to come
        // IIp with a list of winners in a string ("Joe and Ed"). If there's one winner,
        // it returns a string like this, "Ed with 3 books". Otherwise it returns a
        // string like this, "A tie between Joe and Bob with 2 books."


// Here are a couple of short methods that were already written for you,

public string[] GetPlayerCardNames() {
    return players[0].GetCardNames();


public string DescribePlayerHands() {
    string description = "";
    for (int i = 0; i < players.Count; i++} {
        description += players[i].Name + " has " + players(i).CardCount;
        if (players[i].CardCount == 1)
            description +=   card.\r\n";
        else
            description +=   cards.\r\n";

    description += "The stock has " + stock.Count + " cards left.";
    return description;
```



Long Exercise Solution

Here are the filled-in methods in the Game class

```
public class Game {

    private void Deal() {
        stock.Shuffle();
        for (int i = 0; i < 5; i++)
            foreach (Player player in players)
                player.TakeCard(stock.Deal());
        tcreech (Player player in players1
            PullOutBooks(player);


    public bool PlayOneRound lint selectedPlayerCard) {
        Card.Values cardToAskFor - player:s[0]•Peek(:selectedPlayerCard).Value;
        for (int I - 0; i < pjavere vcount. i++) {
            if (i --Ol
                players[0J.AskForACard(players, 0, stock, cardToAskFor);
            else
                players[iJ.AskForACard(players, i, stock);
        if (PullOutBooks(players[i]) {
            textBoxOnForm.Text +- players[i].Name + " drew a new hand\r\n";
            int card = 1;
            while (card <= 5 && stock.Count > 0) (
                players[i].TakeCard(stock.Deal());
                card++;
        ) )
        players(0J.SortHand();
        if (stock.Count -- 0) (
            textBoxOnForrn.Text = "The stock is out of cards. Game over!\r\n";
            return true;


    return false;


public bool PullOUtBook"(Player playerl
{
    List<Card.Vallies> BooksPulled - player.PullOutBooks();
    foreach (Card.Values value in BooksPulled)
        books.Add(value, playe r);
    it (player.CardCount == 0)
        return true;
    return false;
```

The Deal() method gets zalled when the game first starts—it shuffles the deck and then deals five cards to each player. Then it pulls out any books that the players happened to have been dealt.

After the player or opponent asks for a card, the ga..e pulls out any books that he made If a player's out of books, he draws a new hand b dealing up to 5 cards rom the stock

As soon as the player clicks the Ask for a card" button, the ga..e calls AskForACard() with that card Then it calls AskForACard() for each opponent.

After the round is played, the game sorts the player's hand, to make sure it's displayed in der on the form. Then it checks to see if the ga e's over If it is, PlayOneRound() returns true.

PullOutBooksi> looks through a player's cards to see if he's got four cards with the same value If he does, they get added to his books dictionary. And if he's got no cards left afterwards, it returns true

The form needs to display a list of books, so it uses DescribeTheBooks() to turn the player's books dictionary into words.

```
public string DescribeBooks() {
    string whoHasWhichBooks = "";
    foreach (Card.Values value in books.Keys)
        whoHasWhichBooks += books[value].Name + " has a book of "
            + Card.PluralValue(value) + "\r\n";
    return whoHasWhichBooks;
}

public string GetWinnerName() {
    Dictionary<string, int> winners = new Dictionary<string, int>();
    foreach (Card.Values value in books.Keys) {
        string name = books[value].Name;
        if (winners.ContainsKey(name))
            winners[name]++;
        else
            winners.Add(name, 1);
    }
    int mostBooks = 0;
    foreach (string name in winners.Keys)
        if (winners[name] > mostBooks)
            mostBooks = winners[name];
    bool tie = false;
    string winnerList = "";
    foreach (string name in winners.Keys)
        if (winners[name] == mostBooks)
        {
            if (!String.IsNullOrEmpty(winnerList))
            {
                winnerList += " and ";
                tie = true;
            }
            winnerList += name;
        }
    winnerList += " with " + mostBooks + " books";
    if (tie)
        return "A tie between " + winnerList;
    else
        return winnerList;
}
```

Once the last card's been picked up, the game needs to figure out who won. That's what the GetWinnerName() does. And it'll use a dictionary called winners to do it. Each player's name is a key in the dictionary, its value is the number of books that player got during the game.

Next the game looks through the dictionary to figure the number of books that the player with the most books has. It puts that value in a variable called mostBooks.

Now that we know which player has the most books, the method can return a string that lists the winner (or winners).

We're not done yet—flip the page!



Long Exercise Solution (continued)

Here are the filled-in methods inthe Player class

```
public Pkayer Lstring name, Random random, TextBox textBoxOnForm) {
    this.name - name;
    this.random _ random;
    this.textBoxOnForm - textBoxOnForm;
    this.cards _ new Deck( new cere[I {I );
    textBoxOnForm.Text +- name + " has just joined the game\r\n";
```

Here's the constructor for the Player class It sets its private fields and the adds a line to the progress text box saying who joined

```
public Card.Values GetRandomValue() {
    Card randomCard _ cards.Peek(random.Next(cards.Count));
    return rencoeceru.Value;
```

The GetRandomValue() method uses Peek() to look at a random card in the player's hand

```
public Deck DoYouHaveAny(Card.Values value) t
    Deck cardsIHave _ cards.PullOutVillues(value);
    textBoxOnForm.Text +_ Name + " has " + cardsIHave.Count + " "
          + Card.Plural(value> + "\r\n";
    return cardsIHave;
```

DoyouHaveAny() uses the PullOutValues() method to pull out and retu.... all cards that match the parameter

```
pUblic void AskForACard(List<Player> players, int mylndex, Deck stock) (
    Card.Values randomValue - GetRandomValue I);
    AskForACard(players, mylndex, stock, randomValue);
```

There are two o loaded AskForACard() ethods This one i• used by the opponents—it gets a random card from the hand and calls the other AskorACard()_

```
public void AskForACard(List<Player> players, int mylndex,
      Deck stock, Card.Values value) {
    textBoxOnForm.Text +_ Name + " asks if anyone has a " + value + "\r\n";
    int totalCardsGiven = 0;
    for (int i _ 0; i < players.Count; i++) {
        if (i !_ mylndex) (
            Player player = players[iI;
            Deck CardsGiven   player.DoYouHaveAny(value);
            totalCardsGiven += CardsGiven.Count;
            while (CardsGiven.Count > 0)
                cards.Add(CardsGiven.Deal() );

    }
    if (totalCardsGiven == 0) {
        textBoxOnForm.Text +_ Name + " must draw from the stock.\r\n";
        cards.Add(stock.Deal(» ;
```

This AskForACard() method looks through every player (except for the one asking), calls its DoyouHaveAny() method, and add• any cards handed over to the hand

If no cards were handed over, the player has to draw from the stock using its Deal() method

And yet MORE collection types...

List and Dictionary objects are two of the built_in g..n..ric coll..ctions that are part of the .NET framework. Lists and dictionaries are very flexible—you can access any "I'the data in them ill any order. But sometimes yont need I" restrict h,w your program works with Ih,' data because the *thing* that you're representing ill,id,' your program works like rhar inlh,' real world. For situations lik,' that, you'll ",se a Queue "r a St ack, Those are· the other two generic collections rhar are similar ro lists, h,n they're especially good 'II making sure that 'oin' data is processed in a o-rtaiu o{d,·r.

There are other types of collections, too—but these are the ones that you're most likely to come in contact with.

Use a Queue when the first object you store will be the first one you'll use, like:

- Cars moving down a one-wav street
- People standing in line
- (:"'1)mers on h,)ld for a customer service support lim-
- Anything else that's handled on a Ji,st-c,,,"'·, li, st-sc rvc d basi

A queue first-in fi.st-out, which means that the first object that you put into the queue is the first one you pull out of it to use.

Use a Stack when you always want to use the object you stored most recently, like:

- l-urnjtun- loaded inl" the back or a moving truck
- A stack "|' books where you want I" read the most recently added one first
- ,\ pyramid of cheerleaders, where the ones on r,,1' have to di,mounl first.. imagine tl,,' mess ir the one "'llh" 1"'11"", walked awav first!

The stack is first-in last-out—the first object that goes into the stack is the last one that comes out of it

Generic eo lleetio", are a" important part of the .NET fra mework

They're really useful—so much that the IDE automatically adds this statement I" the top of every class you add t" your project:

```
using System.Collections.Generic;
```

Almost every large project that ,u,,'11 work on will include some sort "r generic collection, because your programs need ro store data. ·\nd when you're dealing \ith groups "r similau things in the real world, tl,,·y alinost always naturally fall illl"" category that corresponds pretty well to ,m,' "r these kinds collections.

You can, however, use foreach to enumerate th ough a stack or queue, because they plement IEnumerable!

A queue is like a list that lets you put objects on th••nd of t1,. list anJ use th. ones on th. front. A stack only lets you access the last 01)'ect J you put into it.

A queue is FIFO - First In, First Out

A queue is a lot like a list, except that you can't just add or remove items at any index. To add an object to a queue, you **enqueue** it. That adds the item to the end of the queue. You can dequeue the first object from the front of the queue. When you do that, the object is removed from the queue, and the rest of the objects in the queue move up a position.

Create a new queue of strings

```
Queue<string> myQueue = new Queue<string>();
myQueue.Enqueue("first in line");
myQueue.Enqueue("second in line");
myQueue.Enqueue("third in line");
myQueue.Enqueue("last in line");
string takeALook = myQueue.Peek();①
string getFirst = myQueue.Dequeue();②
string getNext = myQueue.Dequeue();③
int howMany = myQueue.Count;④
myQueue.Clear();
MessageBox.Show("Peek() returned: " + takeALook + "\n"
    + "The first Dequeue() returned: " + getFirst + "\n"
    + "The second Dequeue() returned: " + getNext + "\n"
    + "Count before Clear() was " + howMany + "\n"
    + "Count after Clear() is now " + myQueue.Count);⑤
```

Here's where we add four items to the queue. When we pull them out of the queue, they'll come out in the same order they went in.

Peek() lets you take a "look" at the first item in the queue without removing it.

The first Dequeue() pulls the first item out of the queue. Then the second one shifts up into the first place—the next call to Dequeue() pulls that one out next.

The Clear() method removes all objects from the queue.

The queue's Count property returns the number of items in the queue.



Objects in a queue need to wait their turn. The first one in the queue is the first one to come out of it.



A stack is LIFO - Last In, First Out

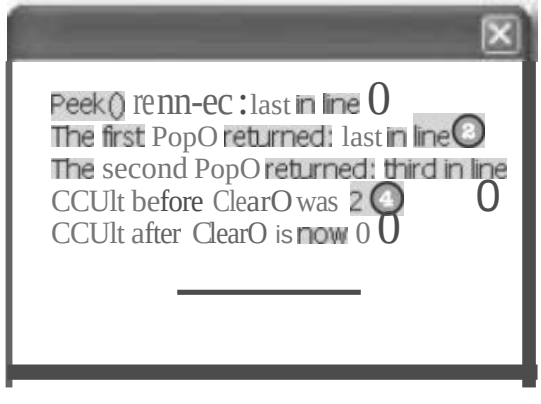
A **stack** is really similar to a line - with one big difference. You push each item onto the back of the line, and when you want to take an item off the stack you pop it. When you pop an item off of a stack, you end up with the most recent item that you pushed. It's just like a stack of plates, magazines or anything else—you can only add something onto the top of the stack, but you can only take it off the top before you can get to whatever's underneath it.

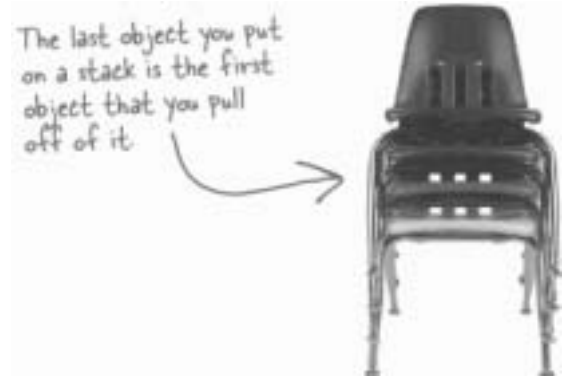
Creating a stack is just like creating any other generic collection

When you push an item onto a stack, it pushes the other items back one notch and sits on top.

```
Stack<string> myStack = new Stack<string>();
myStack.Push("first in line");
myStack.Push("second in line");
myStack.Push("third in line");
mystack.Push("last in line");
string takeALook = myStack.Peek();
string getFirst = myStack.Pop();
string getNext = myStack.Pop();
int howMany = myStack.Count;
mystack.Clear();
MessageBox.Show("Peek() returned: " + takeALook + "\n"
    + "The first Pop() returned: " + getFirst + "\n"
    + "The second Pop() returned: " + getNext + "\n"
    + "Count before Clear() was " + howMany + "\n"
    + "Count after Clear() is now " + myStack.Count);
```

When you pop an item off the stack, you get the most recent item that was added







Wait a minute, something's bugging me. You haven't shown me onything I can do with a srcck or a queue that I ccn't do with a list- they just seve me a couple of lines of code. But I can't get ct the rems in the middle of a stack or a queue. I con do thct with a list pretty easily! So why would I give thot up just for a little convenience?

Don't worry-you don't give up anything when you use a queue or a stack.

1t's really easy t" "o"n'rt a Queue object 1" a List object. .\nd it's ju,' as easy r" convert a l.i« to a Queue, a Queue t" a Stack... ill fact, you can create a List, Queue 01' Stack li"-'Il any other object lhat implements the **IEnumerable** interface. .-\ll you have to do is use the overloaded constructor that I"t, you pas the ('ollection you want to copy from as a parameter. That ,,,,"""" you have the flexibility and convenience or representing your data with the collection rhar best matches "" way you need to be used.

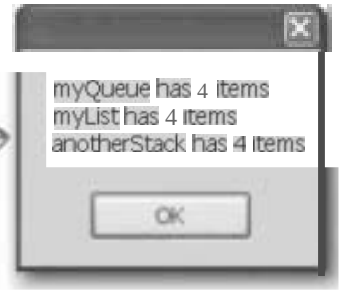
Let's set up a stack with four items—in this case, a stack of strings.

```
Stack<string> myStack = new Stack<string>()
myStack.Push("first in line");
myStack.Push("second in line");
myStack.Push("third in line");
myStack.Push("last in line");
```

It's easy to convert that stack to a queue, then copy the queue to a list, and then copy the list to another stack.

```
Queue<string> myQueue = new Queue<string>(myStackl ;
List<string> myList = new List<string>(myQueue) ;
stack<string> anotherStack = new Stack<string>(myList);
MessageBox.Show("myQueue has " + myQueue.Count + " items\n"
    + "myList has " + myList.Count + " items\n"
    + "anotherStack has " + anotherStack.Count + " items\n");
```

All four items were copied into the new collections.



••.and you can always use a foreach loop to access all of the members in a stack or a queuel



Write a program to help a cafeteria full of lumberjacks eat some flaPJacks, Start wllh the Lumberjack class, filling in the missing code. Then design the term, and add the button event handlers to it.

O Here's the Lumberjack class. Fill in the get accessor for FlapjackCount and the TakeFLapjacks and Eat.FLapjacks methods.

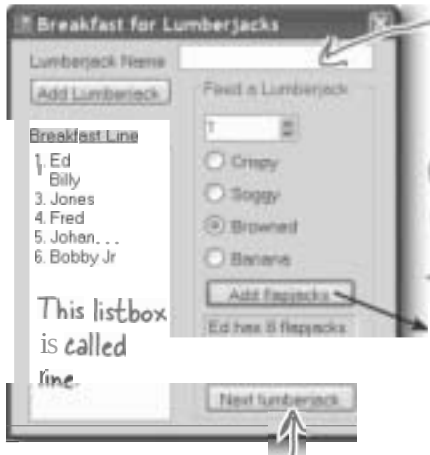
```
public class Lumberjack {
    private string name;
    public string Name { get { return name; } }
    private Stack<Flapjack> meal;
    public Lumberjack(string name) {
        this.name = name;
        meal = new Stack<Flapjack>();
    }
    public int FlapjackCount { get { // return the count } }
    pUblic void Takeflapjacks(flapjack Food, int HowMany)
        // Add some number of flapjacks to the Meal stack

    public void EatFlapjacks() {
        // Write this output to the console
    }
}
```

```
public enum Flapjack {
    crispy,
    soggy,
    browned,
    banana
}
```



O Build this form. It lets you ,>m,'r the names of lumberjacks into a text box so they get in th,>break ast line. You can give the lumberjack at the fronr of the line a plate "r flapjacks, and the-n tell him to move on 1" eat them using the "Next lumberjack" l,tltlotl. We've given you the click """Ul handler for the "Add flapjacks" hun"". Use a queue called breakfastLine I" keep track or the lumberjacks.



When the user clicks "Add Lumberjack", add the name in the name text box to the BreakfastLine queue.

When you drag these RadioButton controls into the groupbox, the form automatically links them and only allows the user to check one of them at a time. Look at the addFlapjacks_Click method to figure out what they should be named.

```
private void addFlapjac  _Cli clt(...)
    Flapj     food;
    if  crisp .Checke    = true)
        food = Flapjack.crispy;
    else if (soggy.Checked -- true)
        food - Flapjack.soggy;
    else if (browned.Checked == true)
        food = Flapjack-browned;
    else
        food = Flapjack.banana:

    Lumberjack currentLumberjack - breakfastLine.Peek();
    eurrentLumberjack.TakeFlapjacks(food,
                      (int)howMany.Value);
    RedrawList();
}
```

Note the special "else if" syntax

Peek() returns a reference to the first lu..berjack in the queue

This button should dequeue the next lumberjack, call his EatFlapjacks ..ethod, and then redraw the listbox,

"""ll you need to add a RedrawList() ..ethod to update the listbox with the contents of the queue All three buttons will call it. Here's a hint: it uses a foreach loop.

The NumericUpDown control is called howMa_, and the label is called nextlnLine




```
private Queue<Lumberjack> breakfastLine = new Queue<Lumberjack>();
private void addLumberjack_Click(object sender, EventArgs e) {
    breakfastLine.Enqueue(new Lumberjack(name.Text));
    name.Text = "";
    RedrawList();
}

private void RedrawList() {
    int number = 1;
    line.Items.Clear();
    foreach (Lumberjack lumberjack in breakfastLine) {
        line.Items.Add(number + ". " + lumberjack.Name);
        number++;
    }
    if (breakfastLine.Count == 0) {
        groupBox1.Enabled = false;
        nextInLine.Text = "";
    } else {
        groupBox1.Enabled = true;
        Lumberjack currentLumberjack = breakfastLine.Peek();
        nextInLine.Text = currentLumberjack.Name + " has "
            + currentLumberjack.FlapjackCount + " flapjacks";
    }
}

private void nextLumberjack_Click(object sender, EventArgs e)
    Lumberjack nextLumberjack = breakfastLine.Dequeue();
    nextLumberjack.EatFlapjacks();
    nextInLine.Text = "";
    RedrawList();
}

public class Lumberjack {
    private string name;
    public string Name { get { return name; } }
    private Stack<Flapjack> meal;

    public Lumberjack(string name)
        this.name = name;
        meal = new Stack<Flapjack>();
    }

    public int FlapjackCount { get { return meal.Count; } }

    public void TakeFlapjacks(Flapjack food, int howMany)
        for (int i = 0; i < howMany; i++) {
            meal.Push(food);
        }
    }

    public void EatFlapjacks()
        Console.WriteLine(name + "'s eating flapjacks");
        while (meal.Count > 0) {
            Console.WriteLine(name + " ate a "
                + meal.Pop().ToString() + " flapjack");
        }
    }
}
```

We called the listbox "line", the label between the two buttons "nextInLine".

The RedrawList() method uses a foreach loop to pull the lumberjacks out of their queue and add each of them to the listbox.

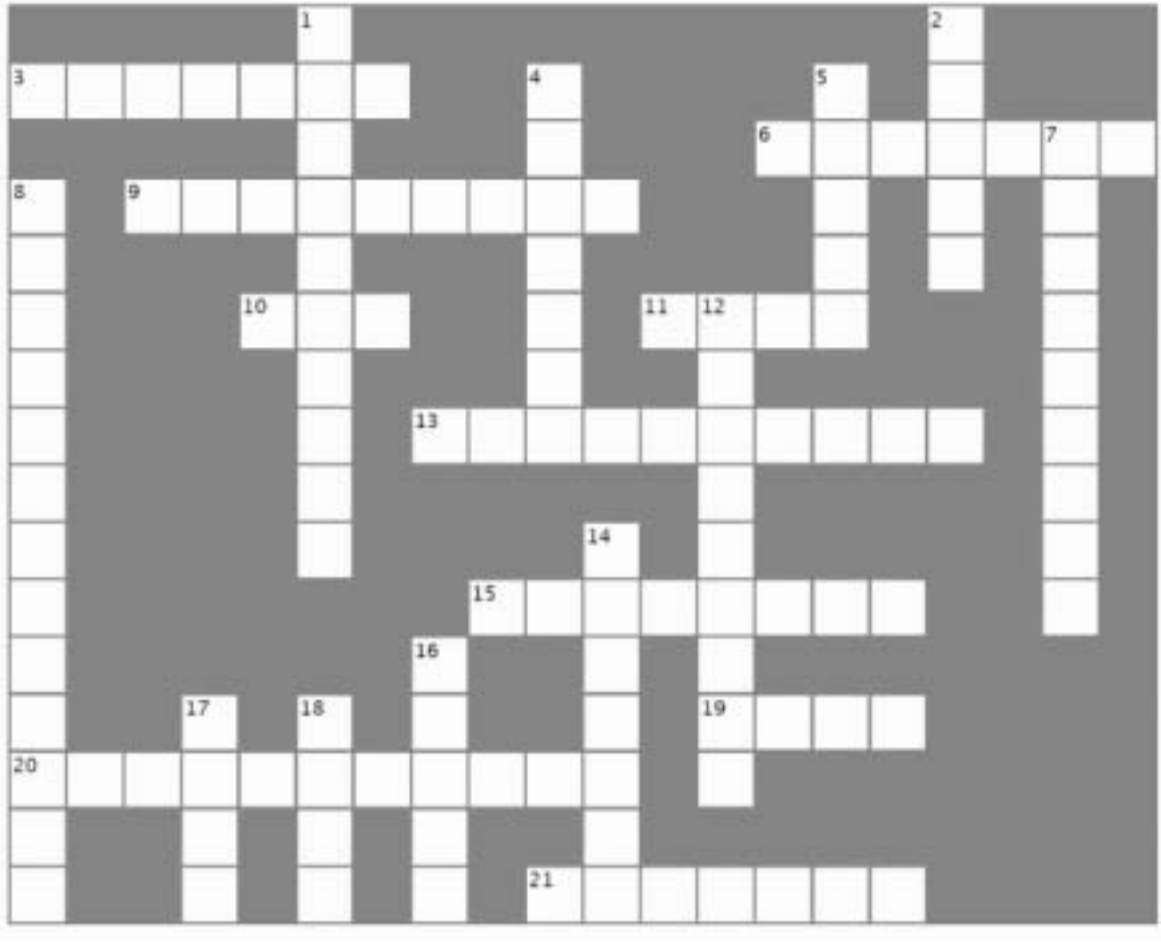
This if statement updates the label with information about the first lumberjack in the queue

The TakeFlapjacks method updates the Meal stack

The EatFlapjacks method uses a while loop to print out the lumberjack's meal



Collectioncross



Across

3. An instance of a collection only sons with one spectc type

6. A special kind of loop that only works on coaectons

9. The name of the method you use to send a string to the output

10, How you remove someltlng from a stack

11. An object mat's like an array but more nexible

13, Two methods in a class with the same name but different parameters are ,.

15 A method 10 figure oul if a certa in object ISin a collection

19. An easy way to keep track 01 categories

20 All generic coaectoos implement this interface

21 . How you remove something lrom a queue

Down

1, The generic cciece cn nat leis you map keys 10 va lues

2. This collection is nrst-in , first-out

4, The builI-in class that lets your program write text to the output

5. A method to find out how many th ings are In a collection

7. The only method in the IComparable interface

8. Most professional projects start with ttliS

12. An object that implements this intertace helps your list sort lis con tents

14 How you add something to a queue

16. This col lection ISfirst-In, last-cut

17. How you add something to a stack

18. Ttus method returns the next object to come off of a stack or queue



Collectioncross solution

					1 D									2 Q						
3 G	E	N	E	R	I	C		4 C					5 C		U					
					C			O				6 F	O	R	E	A	7 C	H		
8 S		9 W	R	I	T	E	L	I	N	E			U		U		O			
P					I								N		E		M			
E				10 P	O	P		O		11 L	12 I	S	T				P			
C					N			L			C						A			
I					A		13 O	V	E	R	L	O	A	D	E	D	R			
F					R						M						E			
I					Y					14 E		P					T			
C								15 C	O	N	T	A	I	N	S		O			
A							16 S			Q		R								
T			17 P		18 P		T			U		19 E	N	U	M					
20 I	E	N	U	M	E	R	A	B	L	E		R								
O			S		E		C			U										
N			H		K		K		21 D	E	Q	U	E	U	E					

Name: Date:

C# Lab

The Quest

This lab gives you a spec that describes a program for you to build, using the knowledge you've gained over the last few chapters.

This project is bigger than the ones you've seen so far. So read the whole thing before you get started, and give yourself a little time. And don't worry if you get stuck—there's nothing new in here, so you can move on in the book and come back to the lab later.

We've filled in a few design details for you, and we've made sure you've got all the pieces you need... and nothing else.

It's up to you to finish the job. You can download an executable for this lab from the website... but we won't give you the code for the answer.

Ihe spec: build **an adventure** gallle

The player and ene..ies move around in the dungeon

The playe.. moves using the four Move buttons



The game shows '/ou the number of **hit points** for the player and enemies When the player attacks an enemy, the eee..,,', hit points go down. Once the hit points get down to zero, the eee..y or player dies.

Th•••e four buttons are used to attack ene...ies and drink potions.

Ihe player **picks** up weapons...

There are weapons and potions scattered around the dungeon that the player can I';,-kHI'and use to defeat his enemies. .\ll he has 1" do is move onto a weap(" and ;1 disappears from the floor and appears ill hi, inventory.

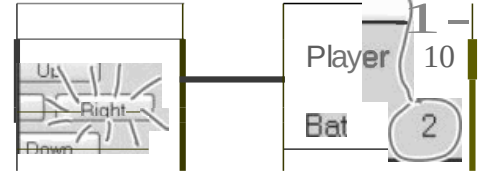


...a.d attacks ,.,mtes with them

Every 1e,-,[ill the game has a weapon that the player can pick "I' and use to defeat h i, ,!11<-"';'-<_( )nce the weapon's picked \lp. it should disappear from the'game floor.



The attack causes the bat's hit points to drop, from 6 to 2 in this case



Higher level, bri.g more ,.emtes

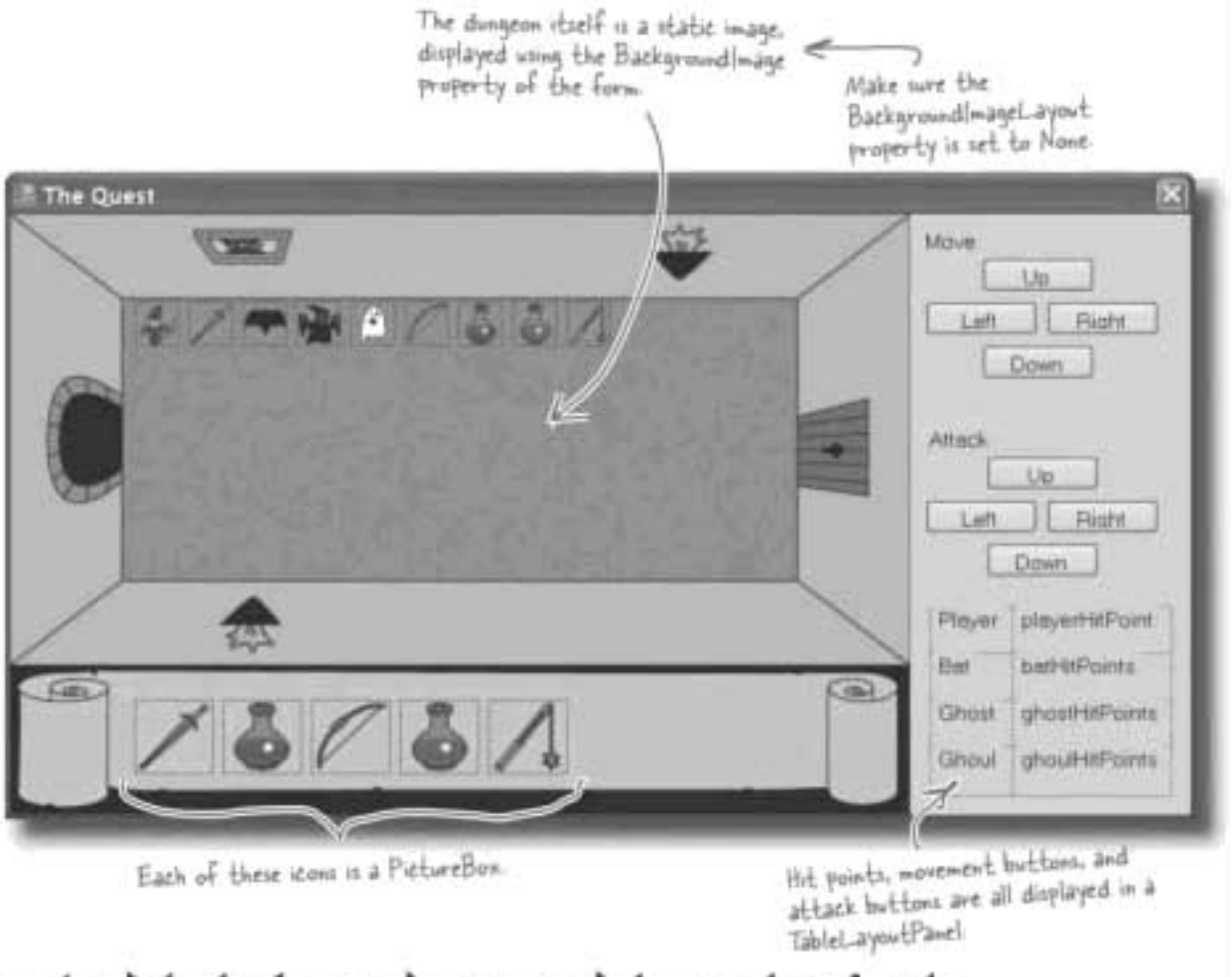
T h" H' are tlm'(' dilT"H'1l1 kind, of enemies: a bat, a ghost, "'d a ghoul. The firs level onlv has a bat. The seventh level is the last one, and il has all three enemies.

The bat Ilies around somewhat randomly. When it's near the player, it causes a small amount of da..age



The design: building the form

The form gives the game its unique look. Use the form's `BackgroundImage` property to display the image of the dungeon and the inventory, and a series of `PictureBox` controls to show the player, weapons and enemies in the dungeon. You'll use a `TableLayoutPanel` control to display the hit points for the player, bat, ghost and ghoul as well as the buttons for moving and attacking.



Download the background image and the graphics for the weapons, enemies, and player from the Head First Labs website: www.headfirstlabs.com/books/hfcsharp

Players, weapons, and enemie shollld all h,- represented by icon . .\<ld nine PictureBox controls. and set their Visible properties ı" False. Then. vour game can movr- around the controls. and toggle their Visible properties as needed.

The Quest



Add nine PictureBox controls to the dungeon Use the Size property to make each one 30x30 It doesn't matter where you place the...—the form will ...ove the... around Use the little black arrow that shows uf when you click on the PictureBox to set each to one of the i..ages fro.. the Head First Labs web site



Build your stars wi.dow

TI,,· hit points are ill a TableLayoutPanel, just like Iht' attack and movement buttons For the hit points, create twc columns in the panel, and drag the column di,-id"r I" the lefi a bit. .\<ld four rows, each 25% height, and add in Label controls I" each of the eight cells:

1 columns, 4 rows. 8 cells for your hit point statistics

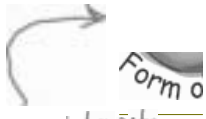
Player	playerHitPoint
; ▪	batHitPoints
Ghost	ghostHitPoints
Ghoul	ghoulHitPoints

Each cell has a Label 'h it, and you can update those .alues during the game

Ihe architecture: using **the** objects

You'll need several types of objects ill your game: a Player objc:" several sub-types of all Enemy object, and several ,ul,. type of a Weapon object..\nd you'll als.. n",-d one object 1" k,<p "I' with everything that's going Oil: II,,, Game object.

This i, just the general overview We'll give you a lot more details on how the playe.. and ene..ies move, how the enemy figures out if it's nea.. the player, etc.



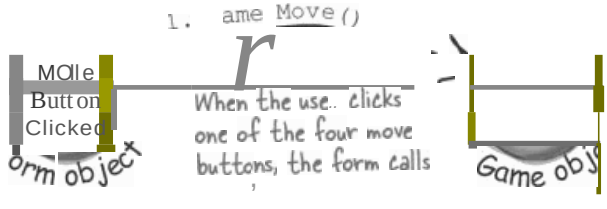
The for.. never interacts directly with the players, weapons or enemies.

The Game object ha"dle, terns

When 0"" Of your form's Move buttons j, clicked, the fi,rm will call1lw Game object's Move ( ) method. Tha nH'lh,,<! will let the player take a (urn. and then let all the enemies move. S" it's up I" Game 10 handle the turn-based movement !",n;,,,, or the game

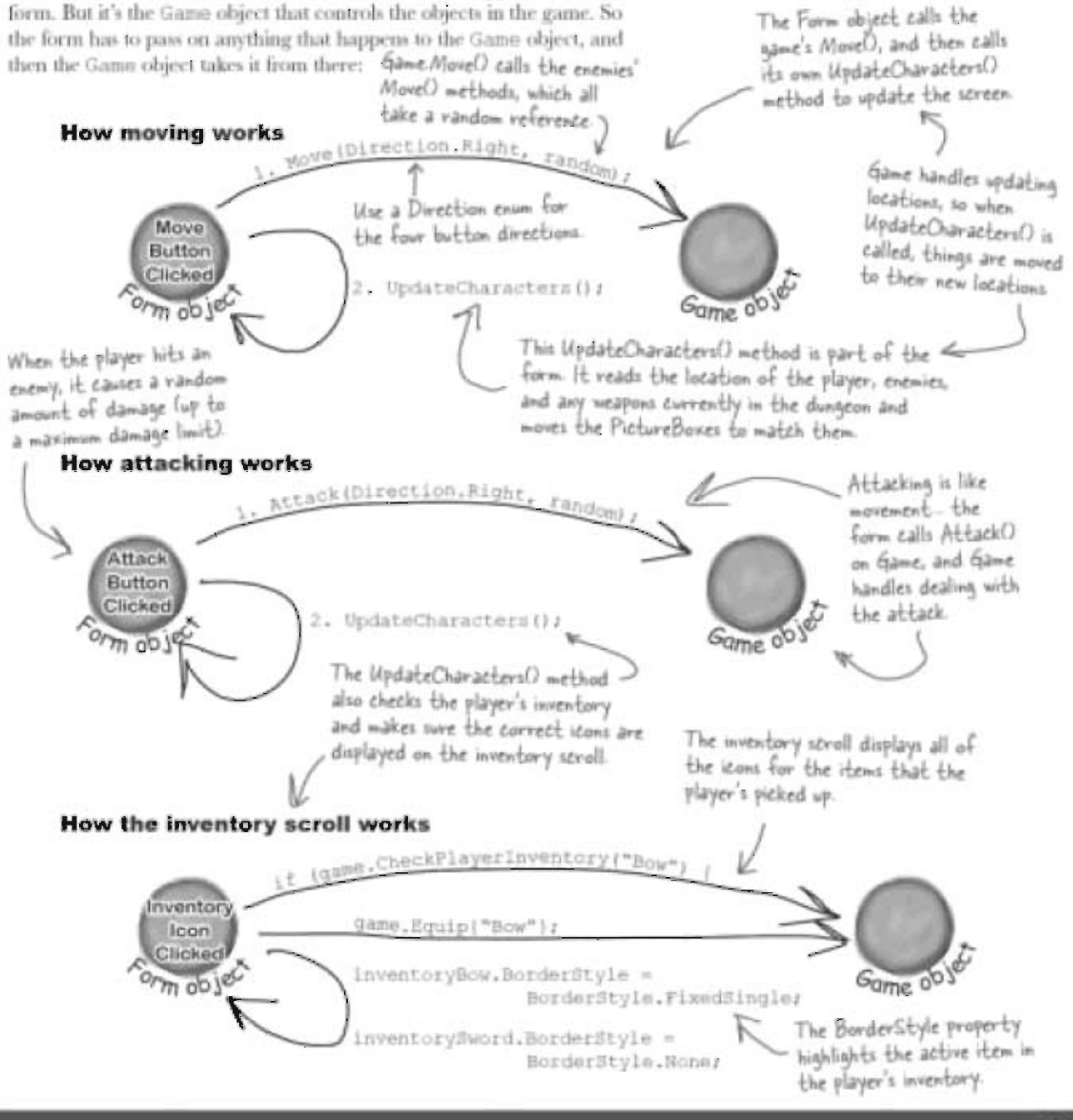
1,,1 example, here's how lh.. move buttons work:

We left the parameters out of this diagram Each Move() method takes a d,..ection and some of them [ake a Random object too.



The form delegates activity to the Game object

Movement, attacking, and inventory all begin in the form. So clicking a movement or attack button, or an item in inventory, triggers code in your form. But it's the Game object that controls the objects in the game. So the form has to pass on anything that happens to the Game object, and then the Game object takes it from there:



Building the Game class

We've gotten you started with the Game class in the code below. There's a lot for you to do—so read through this code carefully, get in the IDE, and get ready to go to work:

```
using System.Drawing;

public class Game
{
  public List<Enemy> Enemies;
  public Weapon WeaponInRoom;

  private Player player;
  public Point PlayerLocation { get { return player.Location; } }
  public int PlayerHitPoints { get { return player.HitPoints; } }
  public List<string> PlayerWeapons { get { return player.Weapons; } }

  private int level = 0;
  public int Level { get { return level; } }

  private Rectangle boundaries;
  public Rectangle Boundaries { get { return boundaries; } }

  public Game(Rectangle boundaries)
  {
    this.boundaries = boundaries;
    player = new Player(this,
      new Point(boundaries.Left + 10, boundaries.Top + 70),
      boundaries);
  }

  public void Move(Direction direction, Random random)
  {
    player.Move(direction);
    foreach (Enemy enemy in Enemies)
      enemy.Move(random);
  }

  public void Equip(string weaponName)
  {
    player.Equip(weaponName);
  }

  public bool CheckPlayerInventory(string weaponName)
  {
    return player.Weapons.Contains(weaponName);
  }

  public void HitPlayer(int maxDamage, Random random)
  {
    player.Hit(maxDamage, random);
  }
}
```

You'll need the Rectangle class from System.Drawing, so be sure to add this to the top of your class.

These are okay as public properties if Enemy and Weapon are well-encapsulated. In other words, just make sure the form can't do anything inappropriate with the...

The game keeps a private Player object. The form will only interact with this through methods on Game, rather than directly

The Rectangle object has a Top, Bottom, Left, and Right field, and works perfectly for the overall game area

Game starts out with a bounding box for the dungeon, and creates a new Player object in the dungeon.

Movement is simple: move the player in the direction the form gives us, and move each enemy in a random direction

These are all great examples of encapsulation... Game doesn't know how Player handles these actions, it just passes on the needed information and lets Player do the rest.

```
public void IncreasePlayerHealth(int health, Random random) {
  player.IncreaseHealth(health, random);
}

public void Attack(Direction direction, Random random) {
  player.Attack(direction, random);
  foreach (Enemy enemy in Enemies)
    enemy.Move(random);
}

private Point GetRandomLocation(Random random) {
  return new Point(boundaries.Left +
    random.Next(boundaries.Right / 10 - boundaries.Left / 10) * 10,
      boundaries.Top +
    random.Next(boundaries.Bottom / 10 - boundaries.Top / 10) * 10);
}

public void NewLevel(Random random) {
  level++;
  switch (level) {
    case 1:
      Enemies = new List<Enemy>();
      Enemies.Add(new Sat(this, GetRandomLocation(random), boundaries));
      WeaponInRoom = new Sword(this, GetRandomLocation(random));
      break;
```

Attack() is almost exactly like Move(). The player attacks, and the enemies all get a turn to move.

GetRandomLocation() will come in handy in the NewLevel() method, where it will use it to determine where to place enemies and weapons

This is just a math trick to get a random location within the rectangle that represents the dungeon area

We only added the case for the level 1. It's your job to add cases for the other levels

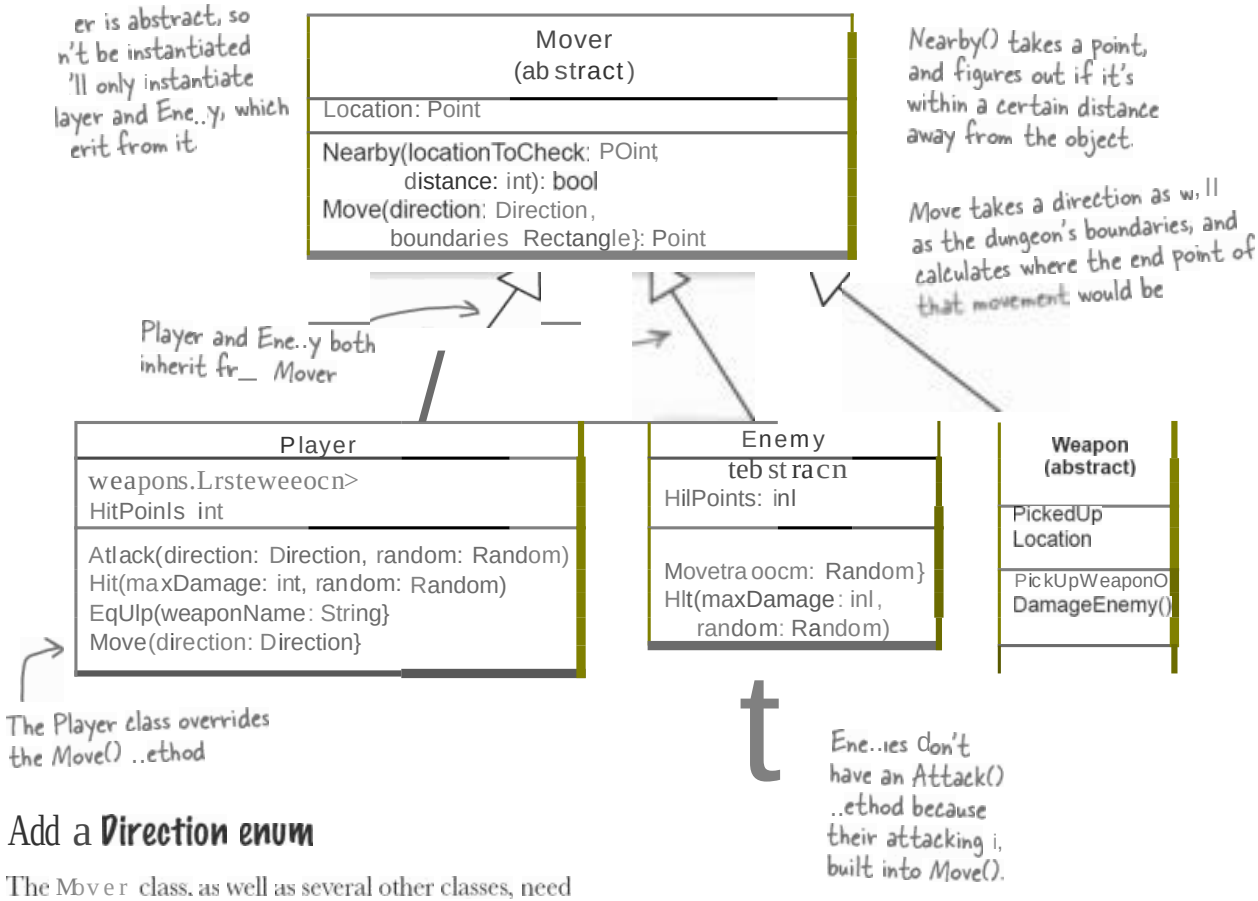
Finish the rest of the levels

Level	Enemies
1	Ghost
3	Ghoul
4	Bat, Ghost
5	Bat, Ghoul
6	Ghost, Ghoul
7	Bat, Ghost, Ghoul
8	NA

Filldillg **common** behavior: **movement**

You already know thai duplicate code j, hild. and duplicate code usually show: up when two or more objects ,hare the same behavior. That's the case in the dungeon game, too... holh enemies and players move.

Let's create a Mover class t« abstract tl,al common behavior iIII" " single place·. Player amI Enemy will inherit from Mover. And even though weapons d",,'! move around, they ;Illn'rit finnl Mover too, because they need some of its propertie and methods. Mover ha, a Move () method Ii,,. moving around, and a read-onlv Location property that Ih,' form can use 10 position a subclass of Mover.



Add a **Direction enum**

The Mover class. as well as several other classes, need " Direction enum. Cr..ate Ihi, " 11111, and give it 1111 I'llulll,'rat"d values: Up, Down, Left, and Right.

Ihe Mover class source code

```
public: abstract class Mover I
  private canst int MoveInterval - 10;
  protected Point location:
  public Point Location I get { return location; } I
  protected Game game;

  public Mover(Game game, Point
    this.game = game;
    this.location - location;


  public bool Nearby(Point locationToCheck, int distance)
    if (Math.Abs(location.X   locationToCheck.X) < distance &&
       (Math.Abs(location.Y   locationToCheck.Y) < distance)) (
      return true;
      else {
      return false;
  }
  public Point Move(Direction direclion, Rectangle boundaries)
    Point newLocation - location;
    switch (direction} I
      case Direction.Up:
        if (newLocation.Y - MoveInterval >= boundaries.Top)
          newLocation.Y -= MoveInterval;
        break;
      case Direction.Down:
        if (newLocation.Y + MoveInterval <= boundaries.Bottom)
          newLocation.Y +- MoveInterval;
        break;
      case Direction.Left:
        if (newLocation.X - MoveInterval >= boundaries.Left)
          newLocation.X -= MoveInterval;
        break;
      case Direction.Right:
        if (ne"'Location.X + MoveInterval <= boundaries.Right)
          newLocation.X +- MoveInterval;
        break;
      default: break;
    }
    return newLocation;
```

Since protected properties are only available to subclasses, the form object, an't set the location... only read it through the public get method we define.

The Nearby method checks a Point against this object's current location. If they're within distance of each other, then it returns true, otherwise it ret...ns fal"

If the end location is outside the boundaries, the new location stays the sa..e as the starting point.

Finally, thi.. new location is returned (which ..ight still be the sa..e as the starting location!).

The Player class keeps track of the player

Here's a start on the Player class. Start with this code in the IDE, and then get ready to add to it.

```
public class Player : Mover {
  private Weapon equippedWeapon;
  private int hitPoints;
  public int HitPoints { get { return hitPoints; } }

  private List<Weapon> inventory = new List<Weapon>();
  public List<string> Weapons {
    get {
      List<string> names = new List<string>();
      foreach (Weapon weapon in inventory)
        names.Add(weapon.Name);
      return names;
    }
  }

  public Player(Game game, Point location, Rectangle boundaries)
      : base(game, location) {
    hitPoints = 10;
  }

  public void Hit(int maxDamage, Random random) {
    hitPoints -= random.Next(1, maxDamage);
  }

  public void IncreaseHealth(int health, Random random) {
    hitPoints += random.Next(1, health);
  }

  public void Equip(string weaponName) {
    foreach (Weapon weapon in inventory) {
      if (weapon.Name == weaponName)
        equippedWeapon = weapon;
    }
  }
}
```

All of the properties of Player are hidden from direct access.

A Player can hold multiple weapons in inventory, but can only equip one at a time.

Player inherits from Mover, so this passes in the Game and location to that base class.

The player's constructor sets its hitPoints to 10 and then calls the base class constructor.

When an enemy hits the player, it causes a random amount of damage. And when a potion increases the player's health, it increases it by a random amount.

The Equip() method tells the player to equip one of his weapons. The Game object calls this method when one of the inventory icons is clicked.

A Player object can only have one Weapon object equipped at a time.

Even though potions help the player rather than hurt the enemy, they're still **considered weapons** by the game. That way the inventory can be a List<Weapon>, and the game can point to one with its WeaponInRoom reference.

Write the Move() method for the Player

`Game` calls the `Player`'s `Move()` method to tell a player to move in a certain direction. `Move()` takes in the direction to move (using the `Direction` enum you should have already added). Here's the start of that method:

This happens when one of the movement buttons on the form is clicked.

```
public void Move(Direction direction) {
  base.location = Move(direction, game.Boundaries);
  if (!game.WeaponInRoom.PickedUp) {
    // see if the weapon is nearby, and possibly pick it up
  }
}
```

Move is in the Mover base class.

When the player picks up a weapon, it needs to disappear from the dungeon and appear in the inventory.

You've got to fill in the rest of this method. Check and see if the weapon is near the player (within a single unit of distance). If so, pick up the weapon and add it to the player's inventory.

The Weapon and form will handle making the weapon's Picturebox invisible when the player picks it up... that's not the job of the Player class.

If the weapon is the only weapon the player has, go ahead and equip it immediately. That way, the player can use it right away, on the next turn.

Add an Attack() method, too

The weapons all have an Attack() method that takes a Direction enum and a Random object. The player's Attack() will figure out which weapon is equipped and call its Attack().

Next up is the `Attack()` method. This is called when one of the form's attack buttons is clicked, and carries with it a direction (again, from the `Direction` enum). Here's the method signature:

If the weapon is a potion, then Attack() removes it from the inventory after the player drinks it.

```
public void Attack(Direction direction, Random random) {
  // Your code goes here
}
```

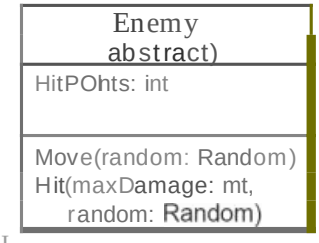
If the player doesn't have an equipped weapon, this method won't do anything. If the player does have an equipped weapon, this should call the weapon's `Attack()` method.

But potions are a special case. If a potion is used, remove it from the player's inventory, since it's not available anymore.

Potions will implement an IPotion interface (more on that in a minute), so you can use the "is" word to see if a Weapon is an implementation of IPotion.

Bats, ghosts, al1d ghouls il1herit from the Enemy class

We'll give you another useful abstract class: Enemy. Each differe1ll sort or enemy has its own class that inherits from the Enemy class. Th,-different kind, "r '-lin n;,", 111'we in different ways, so 11,,· Enemy abstract class leaves the Move method as an abstract method ,he three enemy classes will need to illll'I"1lH"111 it differently, depending on how they move.




```
public abstract class Enemy    Mover
  private canst int NearPlayerDistance = 25;
  private int h1tPo1nts;
  public int H1tPoints { get { return hitPoints; } I
  public bool Dead { get {
       if (hitPoints <= 0) return true;
       else return false;

  |
  public Enemy(Game game, Point location, Rectangle boundaries, int hitPoints)
       : base(game, location) ! this.hitPoints   hitPoints; J

  public abstract void Move(Random random);

  public void Hit(int maxDamage, Random random) {
     hitPoints -- random Next(1, maxDamage);


  protected bool NearPlayer() {
    return (Nearby(game.PlayerLocdtion,
                   NearPlayerDistance});

  protected Direction FindPlayerDirection(Point playerLocation)
    Direction directionToMove;
       if (playerLocation.X > location.X + 10)
          directionToMove = Direction.Right;
    else if (playerLocation.X < location.X - 10)
          directionToMove = Direction.Left;
    else if (playerLocation.Y < locatLonY    10)
          directionToMove    Direction.Up;
    else
          directionToMove    Direction.Down;
     return directionToMove;
```

The for.. can use this read-only property to see if the enemy should be visible in the game dungeon.

Each subclass of Enemy implements this.

When the player attacks an ene..y, it calls the enemy's Hit() ..ethod, which subtracts a ..andom number from the hit points.

The Nea..Player() method uses the Mover Nea..by() static method to figure out if the eee..y .. near the ·layer

If you feed FindPlayerDirection() the player's location, it'll .'e the base class' loeation field to figure out wkere the player i. in relation to the enemy and retu a Direction enum that tells you h h direction the enemy needs to move in order to move towards the player.

Write the different Enemy subclasses

The three Enemy subclasses are pretty straightforward. Each enemy has a different number of starting hit points, moves differently, and does a different amount of damage when it attacks. You'll need to have each one pass a starting HitPoints parameter to the Enemy base constructor, and you'll have to write different Move() methods for each subclass.

```
public class Bat : Enemy {
    public Bat(Game game, Point location, Rectangle boundaries)
        : base(game, location, boundaries, 6)
    { }
    public override void Move(Random random)
        // Your code will go here
```

The bat starts with 6 hit points, so it passes 6 to the base class constructor.

You probably won't need any constructor for these; the base class handles everything

Each of these subclasses the Enemy base class which in turn subclasses Mover.

Move()

The bat flies around somewhat randomly, so it uses Random to fly in a random direction half the time

Once an enemy has no more hit points, the form won't display it any more. But it'll still be in the game's Enemies list until the player finishes the level

The bat starts with 6 hit points. It'll keep moving towards the player and attacking **as long as it has one or more hit points**. When it is at 0 hit points there's a 50% chance that it'll move towards the player, and a 50% chance that it'll move in a random direction. After the bat moves, it checks if it's near the player. If it is then it attacks the player with up to 2 hit points of damage.

We'll have to make sure the form sees if an enemy should be visible at every turn.

Ghost

Move()

The ghost is harder to defeat than the bat, but like the bat, it will only move and attack if its hit points are greater than zero. It starts with 8 hit points. When it moves, there's a 1 in 3 chance that it'll move towards the player, and a 2 in 3 chance that it'll stand still. If it's near the player, it attacks the player with 1 to 4 hit points of damage.

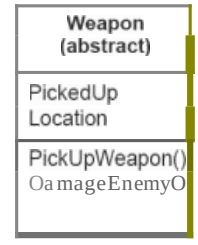
The ghost and ghoul use Random to make them move more slowly than the player.

The ghoul is the toughest enemy. It starts with 10 hit points, and only moves and attacks if its hit points are greater than zero. When it moves, there's a 2 in 3 chance that it'll move towards the player, and a 1 in 3 chance that it'll stand still. If it's near the player, it attacks the player with 1 to 3 hit points of damage.

Weapon inherits frolll Mover. each weapon inherits frolll Weapon

W(need a base Weapon class, just like we had a base Enemy class. And each weapon has a 1ocat jun. as well as a property indi,ating whether Drn,,(it's been picked up. Here' the bas(Weapon class:

Weapon inherits from Mover because it uses its Nearby() and Move() methods in DamageEne..y()




```
public abstract class Weapon ; Mover {

  protected Game game;
  private bool pickedUp;
  public booI Pickedup { get { return pickedUp; } J
  privat;e Point location;
  public Point Location { get { return location; } }

  public Weapon(Game game, Point location) {
    this.game - game;
    this.location _ location;
    pickedUp = false;
  )

  public void PickupWeapon() { pickedUp - true;

  public abstract string Name { get; I

  public abstract void Attack(Direction direction, Random random);

  protected bool DamageEnemy(Direction direction, int radius,
                             int damage, Random random} {
    Point target = game.PlayerLocation;
    for (int distance _ 0; distance < radius; distance++)
      foreach (Enemy enemy in game.Enemies) {
        if (Nearby(enemy.Location, target, radius) I
          enemy.Hit(damage, random);
          return true;

      |
      target - Mve(direction, target, game.Boundaries);

    return false;
```

A pickedUp weapon be displayed anymore Id n't the form can use thi get acce sor to figu..e that out

Every weapon has a location in the game dungeon.

The constructor set.. the game and location fields and sets pickedUp to false (because it hasn't been picked up yet)

Each eapon class needs to imple ent a Na..e property and an Attack() ethod that determines how that eapon attacks.

Each weapon has a different range and pattern of attack, so the weapons imple..ent the Attack() ..ethod differently

weapon's Name prope-ty returns its na ..e ("Sword", "Mace", "Bow").

Different weapons attack in different ways

Each subclass of `Weapon` has its own name and attack logistic. Your job is to implement these classes. Here's the basic skeleton for a Weapon subclass:

Each subclass represents one of the three weapons: a sword, bow, or mace.

```
public class Sword : Weapon {

  public Sword(Game game, Point location)
      : base(game, location) { }

  public override string Name { get { return "Sword"; } }

  public override void Attack(Direction direction, Random random) {
    // Your code goes here
  }
}
```

Each subclass relies on the base class to do the initialization work.

You're basically hardcoding in the name of each weapon.

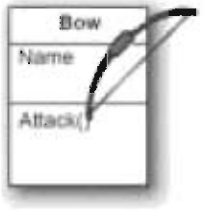
The Game object will pass on the direction to attack in.

The player can use the weapons over and over—they never get dropped or used up.



The sword is the first weapon the player picks up. It's got a wide angle of attack: if he attacks up, then it first tries to attack an enemy that's in that direction—if there's no enemy there, it looks in the direction that's clockwise from the original attack and attacks any enemy here, and if it still fails to hit then it attempts to attack an enemy counterclockwise from the original direction of attack. It's got a radius of 10, and causes 3 points of damage.

Think carefully about this—what is to the right of the direction left? What is to the left of up?



The bow has a very narrow angle of attack, but it's got a very long range—it's got an attack radius of 30, but only causes 1 point of damage. Unlike the sword, which attacks in three directions (because the player swings it in a wide arc), when the player shoots the bow in a direction, it only shoots in that one direction.



The mace is the most powerful weapon in the dungeon. It doesn't matter which direction the player attacks with it—since he swings it in a full circle, it'll attack any enemy with a radius of 20 and cause up to 6 points of damage.

The different weapons will call DamageEnemy() in various ways. The Mace attacks in all directions, so if the player's attacking to the right, it'll call DamageEnemy(Direction.Right, 20, 6, random). If that didn't hit an enemy, it'll attack Up. If there's no enemy there, it'll try Left, then Down—that makes it swing in a full circle.

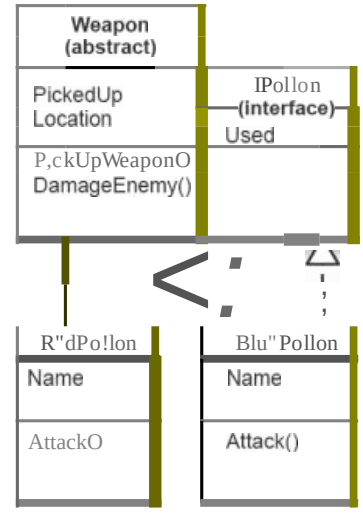
Potions illlplelllent the IPotion interface

lhe-reare two potions, a 111""I"I;"n and a red l'"l;"n. which increase the player's health. They act ju,' lik' weapons—the player picks Inem up in the dungeon, equips them by clicking "ulh,' inventory, and **us..s ,h"m** by **clicking** on.. of th.. **attack buttons**. So it makes sense for them to iulwr;1 from the abstract Weapon class.

But 1"1;011' act a little differently, too. s(you'll need ,o add an IPot*i*on interface so they can have extra behavior: increasing the player's health. The IPotion interfac(is n""II,';lll1"e. Potions only need to „ld one read-only property called Used that returns false if the player hasn't used rhe poti"n. and true if he has. The form will use it to determl",-whether ," not 1" display the l'"li"l1 in the inventory.

```
public interface IPotion
  bool Used { get; }
```

IPotion makes potions usable only once. It's also possible to find out if a Weapon i• a potion with "if (weapon i. IPotion) because of this interface."



Name

Attack()

'I'lk' BluePotion class's Name property should return the string "Blue Potion". Its Attack() method will be called when the player uses the blue potion it should increase the player's health bv up to 5 hit points by calling the IncreasePlayerHealth() method. After the player use the potion, the p'li,,n', Used() method _I",uld return true,

If the player ,"icks up a blue potion on level 2, uses it, and then picks up another one on eve I 4, the ga..e will end up creating two different BluePotion instances

RedPotio

Name

Attack()

The RedPotion class is very similar to Blue Potion. except m.« its Name property returns the string "Red l'"lion", and its Attack() method increases the player's health by up 1010 hil points

Ihe **form** brings it all together

There's one instance or the Game object, alld il lives a, a private field or your form. It's created ill the form's Load event, and Ih,' various event handlers ill Ih,' form use the fields and methods ')(' Ih,' Game object l,. keep the game play going.

Everything begins with the form's Load event handler, which passe the Game a Rectang1e Ihal define the boundaries of the dungeon play area. Here's ' '"(\\" form c.d,· I" get you going:

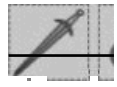
```
private Game game;
pr-Ivet.e Random random - new Random();
prtvete void Forml Load(object sender,
                        EventArgs e) |
    game = new Game(new Rectangle(78, 57, 420, 155));
    game.NewLevel(random);
    updet.echerecters();
```

These are the boundaries of the dungeon in the background i..age you'll download and add to the for..,

Using *a* Rectangle

You'll find a lot of Rectangles any time you work with for..... You can create one by passing it X, Y, Width, and Height values, or two Points (for opposite corners). Once you've got a rectangle instance, t- can also access its Left, Right, Top, and Bottom, as sell as its X, Y, Width, and Height values.

Reme..ber to double-click on each PictureBox '0 the IDE adds a separate event handler ..ethod for each of the..,





The form has a separate event handler for each of these PictureBox's Click events. When the player clicks '"' the sword it first checks to make sure Ihe sword is lll Ih:. player's inventory using the Game" 'ject's Ch ee kPl a yerInventory() method.], the player's ' ' ,,J,"ll," Ih e sword, th lll' form calls game.Equip() to eqll'p it. It then sets each PictureBox s or .er y e pr"perty to draw a box around the sword, and mak,' sure noru-of the- other icons ha,,' a hox around them.

Up

Len ll Right

There's all event handler for each "l' the Ilpur ""vement buttons. They're pretty simple. First the [mn"" calls game.Move() with the appropriate Direction value, and tlu-nit calls the I"rlll' VpdateCharacters() method.

Make e you change the buttons back hen the player equips the ...ord, bow, or ..ace

Up

Len ll Right

The four auack huuon event handlers are also really simple. . Each tuu«"' "all, game.Attack(), a lll llnell calls th.. l,orm's UpdateCharacters() """Ih,,d, If the player «nor» a potion, il's 'Iill used the same way by calling 9ame.Attack() llln [loti,'II' have ll" directio,n. So make the Left, Rialu. and I),wn buttons invisible when the player equips a polioll, and change the text "11 the I lp hUlI""I" say ") Iri"k".

Ihe form's UpdateCharactersO method moves the PictureBoxes into position

The last piece Of the puzzle i, the form's UpdateCharacters () method. Once all the objects have moved awl acted Oil each other, the form updates everything... so weapons that h,-,'n ,Itopped have their PictureBoxes \'isible properties set to falsc. ,"nt-mi,', and players are drawn in their new locations (and dead ones are mad.. invisible), and inventorv i, updated.

Here's what you need 10<l„:

O Update the player's position ond stots

The lir'l thing you'H do is update the player's PictureBox location 'llld 11ıc label til", shows hi, hil poim', Then you'll need a few variables 10determine whethe you've shown each of the \ariou, enemies.

```
public void UpdateCharacters()
  Player.Location = game.PlayerLocation;
  playerHitPoints.Text =
    game.PlayerHitPoints.ToString();

  bool showBat = false;
  bool showGhost = false;
  bool showGhoul = false;
  int enemiesShown = 0;
  // more code to go here...
```

The showBat variable will be set to true if we ..ade the bat's PictureBox visible. Same goes for showGhost and showGhoul.

O Update each enemy's location and hit points

1:'I(h enemy could be in a new location. and have a different "'f or hil point', You need to update each enemv after you've lll'da("d the player's location:

```
foreach (Enemy enemy in game.Enemies)
  if (enemy is Bat)
    bat.Location = enemy.Location;
    bat.HitPoints.Text = enemy.HitPoints.ToString();
    if (enemy.HitPoints > 0) {
      showBat = true;
      enemiesShown++;

  |
  // etc...
```

This will affect the visibility of the ene..y Pict....eBox controls in just a bit.

Y<.;11 need two more if state..ents like this in your foreach loop—one for the ghost and one for the ghoul.

Once you'v(looped th.ough all the enemies on the level, check the showBat variable. If the bat was killed. then showBat will 'lill be false, so mak,' its PictureBox invisible and clear its hit points label. Then do thr- same for showGhost and showC.huul.

3 Update the weapon PictureBoxes

Declare a weaponControl variable and use a big switch statement to set it equal to the PictureBox that corresponds to the weapon in the room.

```
sword.Visible = false;
bow.Visible = false;
redPotion.Visible = false;
bluePotion.Visible = false;
mace.Visible = false;
Control weaponControl = null;
switch (game.WeaponInRoom.Name)
    Case "Sword":
        weaponControl = sword;
```

The rest of the cases should set the variable weaponControl to the correct PictureBox on the form. After the switch, set weeponcontrol.Visible to true to display it.

Set the Visible property on each inventory icon PictureBox

Check use the Game object's CheckPlayerInventory() method to figure out whether or not to display the various inventory icons.

Here's the rest of the method

The first thing you'll do is update the player's PictureBox location and the label that shows hit points. Then you'll need a few variables to determine whether you've shown the various enemies.

```
weaponControl.Location = game.WeaponInRoom.Location;
if (game.WeaponInRoom.PickedUp)
  weaponControl.Visible = false;
else
  weaponControl.Visible = true;

if (game.PlayerHitPoints <= 0)
  MessageBox.Show("You died");
  Application.Exit();

if (enemiesShown < 1)
  MessageBox.Show("You have defeated the enemies on this level");
  game.NewLevel(random);
  UpdateCharacters();
```

Every level has one weapon. If it's been picked up, we need to make its icon invisible.

Application.Exit() immediately quits the program. It's part of System.Windows.Forms, so you'll need the appropriate using statement if you want to use it outside of a form.

If there are no more enemies on the level, then the player's defeated them all and it's time to go to the next level.

Ihe fun's just beginning!

Seven levels, three enemies... that's a pretty decent game. But you can make it even better. Here are a few ideas to get you started...

Make the enemies smarter

Can you figure out how to change the enemies' MoveO methods so that they're harder to defeat? Then see if you can change their constants to properties, and add way to change them in the game.

Add more levels

The game doesn't have to end after seven levels. See if you can add more... can you figure out how to make the game go on indefinitely? If the player does win, make a cool ending animation with dancing ghosts and bats! And the game ends pretty abruptly if the player dies. Can you think of a more user-friendly ending-maybe let the user restart the game or retry hie last level.

Add different kinds *ot* enemies

You don't need to limit the dangers to ghouls, ghosts, and bats. See if you can add more enemies to the game.

Add more weapons

The player will definitely need more help defeating any new enemies you've added. Think of new ways that the weapons can attack, or different things that potions can do. Take advantage of the fact that Weapon is a subclass of Mover—make magtc weapons the player has to chase around!

Add more graphics

You can go to **www,headflrstlabs,comjbooks/hfcsharpj** to find more graphics files for additional enemies, weapons, and other images to help spark your imagination.

This is your chance to show oH! Did you come up with a cool new version of the game? Join the Head First C# forum and claim your hragging rights: www.headfirstlabs.com/books/hfcsharp/

9 reading and writing files

Save the byte **array,** save the world



Sometimes it pays to be a litUe persistent.

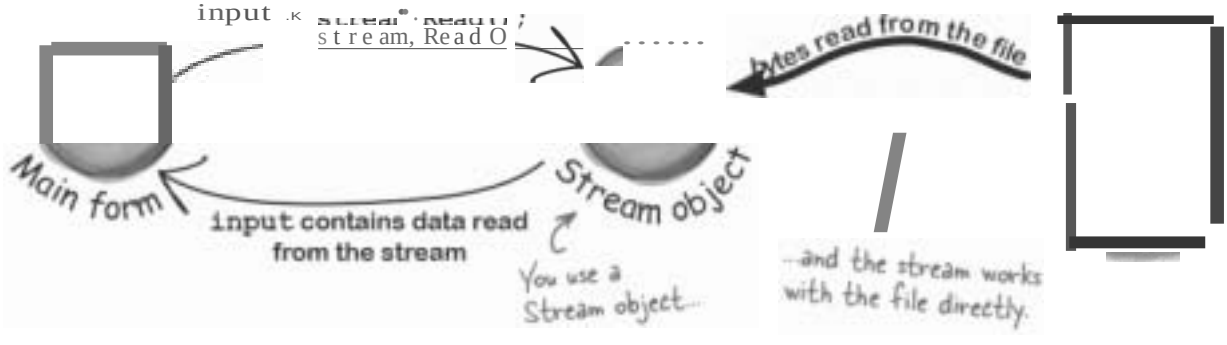
So far, all of your programs have been pretty short-lived, They fire up, NO for a while, and shut down, acnrers not atways enough, especially when you're dealing with important information. You need 10 be able to save your work, In this chapter, we'll look at how to write data to a file, and then I'OW to read that information back in from a file, 'rcu'llleern about the .NET stream classes, and also take a look at the mvetenes of hexadecimal and binary

C# uses streams to read and write data

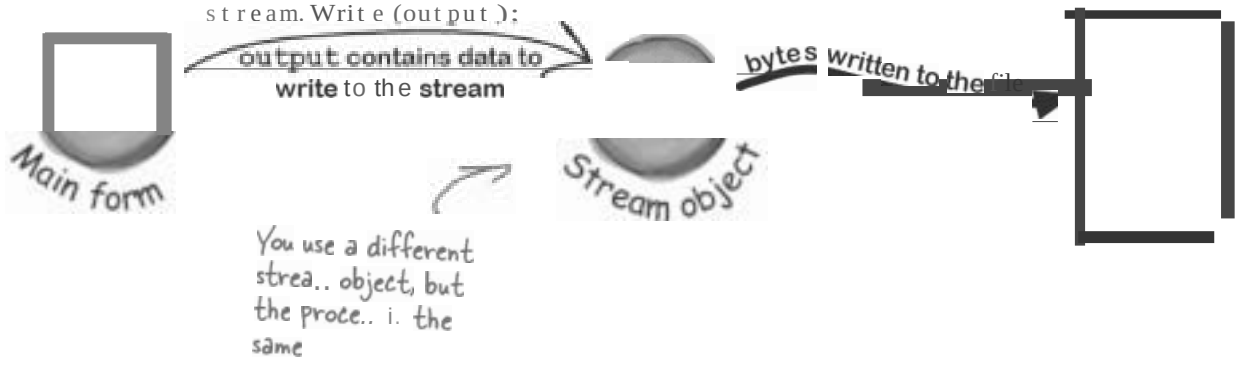
A **stream** is the .NET Framework's way of getting data in and out of your program. Any time your C# program reads or writes a file, connects to another computer over a network, or generally does anything where it **sends** or **receives** bytes from one place to another, you're using streams.

Whenever you want **to read** data from a **file or write** data to a file, you'll use a Stream **object.**

Let's say you have a simple program–a form with an event handler that needs to read data from a file. You'll use a Stream object to do it.

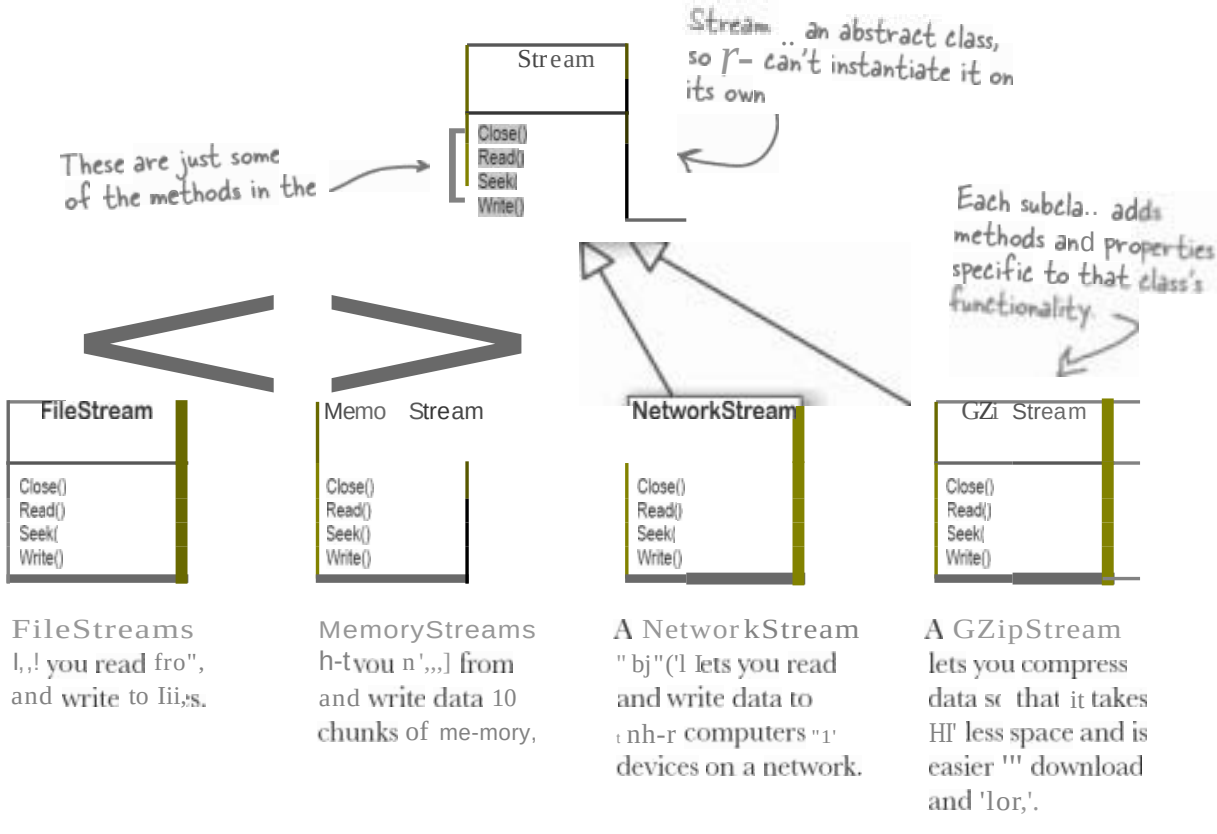


And if your program needs to write data out to the file, It can use another Stream object.



Different streams read al1d write differel1t thil1gs

Every stream is a subclass "f the abstract **Stream** class, and there are a bunch of built-in stream classes to d" different ,him,"_ We'll be concentrating on reading ami writing regular files but everything you learn in ,hi, chapter will just as easily apply to compressed or encrypted IiL,·" "'- network streams that don't use files at aiL



FileStreams
I,,! you read fro", and write to Iii,;s.

MemoryStreams
h-tvou n',,,] from and write data 10 chunks of me-mory,

A NetworkStream
" bj"('l lets you read and write data to ,nh-r computers "1' devices on a network.

A GZipStream
lets you compress data sc that it takes HI' less space and is easier "' download and 'lor,'.

fhi"gs you ca" do with a stream:

- **Write to the stream.**
  You can write te-xt or binary data to a stream and through a stream's `Write()` method.

- **Read from the streem.**
  You can use the `Read()` method to get data from a fil,-. or a network, or me-morv, "' .)"'1 abc,ul anything else, using a stream

- **Change your position within the stream.**
  Most streams support a `Seek()` "",th"d that let you lind a position withi" the stream so you """ insert data at a specific place.

SIreams **let** you **read** and **write** data. **Use the** right kind of **stream** for **the** data you're working **with.**

A FileStreall writes bytes to a file

When your program needs I" write a few lines or text to a file, there are a 1„1 of things that have 10 happen:

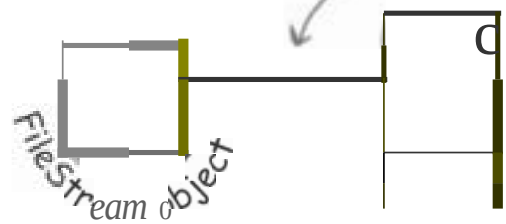
Make sure you add using System.IO; to any program that uses stream,



O Creat‹ a new FileStream object and tell il to writ‹ to the file.



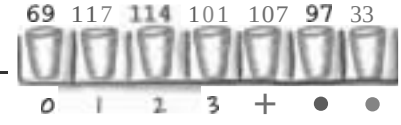
O Tln- FileStream attaches irv-lf to a file.



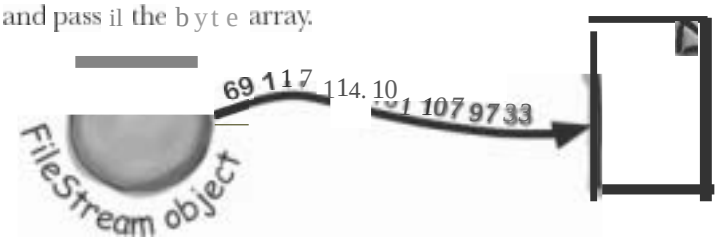
3 Streams write byte to files so vou'll need to convert the string th"l you want h' write to an array of byte>.

This is called **encoding**, and we'll talk more about it later on...

Eureka! –

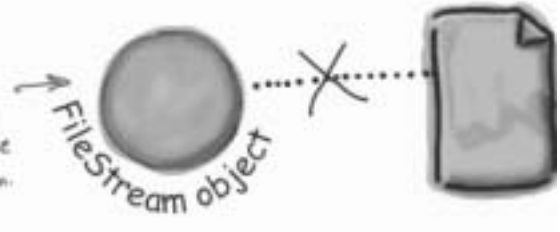


O Call 1]" , stream's W‹i te () method and pass il the byte array.



5 Close the file so other programs can access it.

Forgetting to close a stream is a big deal! Otherwise, the file will be locked, and other programs can't use that file until you close your stream.



How to write text to a file ill 3 sillple steps

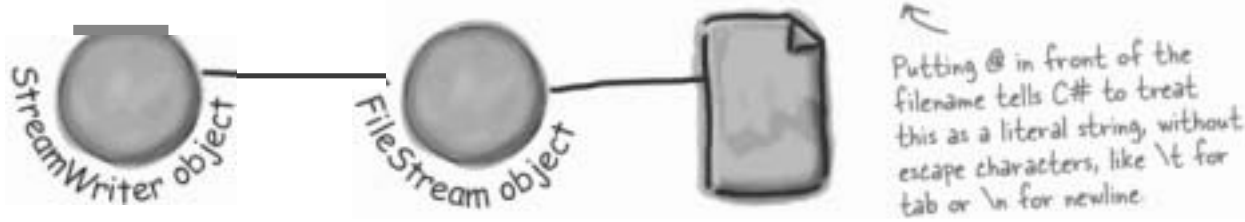
C# come with a convenient class called Strf'a.uWritf'r that does all of those things in one easy step. ,\11 you have '0 do is create a new St.reamWrite r object and give il a filename. It automatically creates a FileStream ,,,,d links il 10 ,he file. Thell you can use Ihe StreamWriter's Wr i t e () ,,,,d WriteLi ne () methods to wrile everything to the file yc'" want.

StreamW.-iter creates anJ manages a FileSlream ol.jeel lor you autematieally.

0 Use the StreamWriter's constructor to open or create a file

You can pass a filename I" ,h,' St reamWr-iter () constructor. \Vhen you do, the writer automatically op"n, the file. St reemwr iter also has an overloaded constructor that also takes a boo1: true if you want 10,,,Id text I" the end ,)1' all existing Iil" (or append), Ot false if you want 10 delete the existing file and create a new Iii,' with the same I""" ,

```
StreamWriter writer _ new StreamWriter(@"C:\newfiles\toaster oven.txt", true);
```



0 Use the Write() and WriteLine() methods to write to the file

These methods ""'1kj"'1 like the one in Console : Write () writes text, "nd Wr i teLine () writes ,,'XI and adds a li"" break lO the end. [f you include .. : ()l", ..¦ 1¦"•.. : 2¦", ,·le .. ;", id,- ,I", string yc,u're writing, the methods include parameters in the strings being written: .. : ():.. i, replacc·d wilh the Ii.s parameter ;. Ii,·r the string being writ n-n. "¦ l :.. is replaced with the second, etc.

```
writer.WriteLine("The {0} is set to III degrees.", appliance, temp.ToString());
```



3 Call the Close() method to release the file

If you leave the 'Ir""'1! 0p"n and attached to a file, then i,'11 keep the file locked "1" '" and '") other program will be able (" use it. S,) make ,,,re you alway close yc,ur files!

```
writer.Close!);
```

Ihe Swilldler **launches** another diabolical plall

The citizens of ()hj'"l,-ilk have long lived in fear of the Swindler.
Now he's using a StreamWriter 10implement another evil plan.
Let's take a l,"ok at what's going oil:

This line c..eates the StreamW..ite.. object and tells it whe..e the file will be.

The path starts with an @ sign so that the StreamWriter doesn't interpret the "\" as the start of an escape sequence.

```
StreamWriter sw - new StreamWriter(@"c:\secret_plan.txt");
sw.WriteLine("How I'll defeat Captain Amazing");
sw,WriteLine("Another genius secret plan by The Swindler");
sw.Write("I'll ereate an army of clones and ");
sw.WriteLine("unleash them upon the citizens of Objectville.");
string location - "the mall";
for (int number - 0; number <- 6; number++)(
    sw.WriteLine("Clone #{0} attacks Il}", number, location);
    if (location _ "the mall") I location _ "downtown"; }
    else I location - "the mall";
}
sw.Close();
```

W..iteLine() adds a new line after writing. W..ite() sends just the text, with no extra line feeds at the end

Can you I igure out what's going on with the location variable in this code?

You can use the {} within the text to pass in variables to the st..ing being written {0} is replaced by the first para..ete.. after the string, {1} by the second, and so on.

Close() frees up any connections to the file and any resources the StreamWriter is using. The text doesn't get written if you don't close the stream.

This is what the eede above produces.

secret_plan.txt - Notepad

File Edit Format View Help

```
How I'll defeat Captain Amazing
Another genius secret plan by The Swindler
I'll create an army of clones and unleash them upon the citizens of Objectville.
:lone #0 eereces the mall
Clone #1 attacks downtown
Clone #2 attacks the mall
:looe #3 attacks downtown
Clone #4 attacks the mall
Clone #5 attacks downtown
Clone #6 attacks the mall
```



StreamWriter Magnets

Suppose you have the code for button1_Click shown below. Your job is to use the magnets to build code for the Flobbo class so that, when the event handler is called, it produces the output shown at the bottom of the page. Good luck!

```
private void button1_Click(object sender, EventArgs e) {
    Flobbo f = new Flobbo("blue yellow");
    StreamWriter sw = f.Snobbo();
    f.Blobbo(f.Blobbo(f.Blobbo(sw), sw), sw);
}
```

```
sw.WriteLine(Zap);
Zap = "red orange";
return true;
```

```
sw.WriteLine(Zap);
sw.Close();
return false;
```

```
public bool Blobbo
    (bool Already, StreamWriter aw) {
```

```
public bool Blobbo(StreamWriter sw) {
```

```
sw.WriteLine(Zsp);
Zap = "green purple";
return falae;
```

```
return new
StreamWriter("macaw.txt");
```

```
private string Zap;

public Flobbo(string Zap) ,
    thia.Zap = Zap;
```

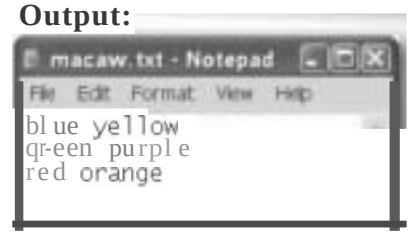
```
public class Flobbo
```

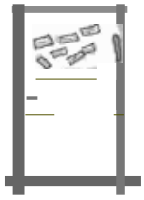
```
if (Already) {
```

```
else
```

```
public StreamWriter Snobbo()
```

Output:





StreamWriter Magnets Solution

Your job was to construct the flobbo class from the magnets to create the desired output.

```
privat.. void buttonl_Clic:k(ohj..ct s..nd..r , Ev..ntArgs ..) I
    Flobbo f .. new Flobbo("blu.. y..llow");
    StreamWriter sw _ f .Snohbo();
    f .Blobbo(f .Bl obbo(f .Bl ohbo(sw) , sw) , sw);
```

```
public class Flobbo {
    privat.. string Zap;

    public: Flobbo(stringq Zap) {
        this.Zap = Zap;
    }

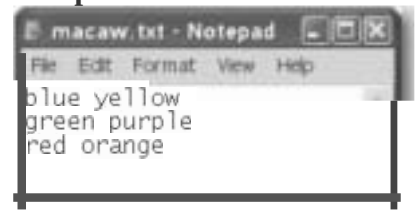
    public: Str..amWrit..r SnobboO {
        return new
            StreamWriter("macaw.txt");
    }

    public: bool Blobbo(StreamWrit..r sw)
        sw.WriteLine(Zap);
        Zap " "gr....n purpl..";
        return fals..z

    public: boo1 Blobbo
        (boo1 Alr..ady, Str..amWriter sw) {
        if (Already) {
            sw.WriteLine(Zap);
            sw.Close();
            return false;
        } else {
            sw.Wr1teLi n..(Zap);
            Zap . "r..d orange";
            return true;
        }
    }
```

Make sure you close files when you're done with them

Output:



Reading and writing takes two objects

Let's read Swindler's secret plans with another ,Iream. a Streemaeecer. Streemkeecer works just lik,' a StreamWrite.r. except instead or writing a file y.u give ,h,' reader the name "r the file I" read in ii' c0""lnKI'I. The ReadLine{) 1lH'lh,id returns a string that contains the next li",- from 11„- lil,-. Y„" can write a loop lhal reads lines from illlillil its EndOf StreamJi,-ld is Irll'- that's when il 1'111\' ,,"I or Iill'-' 10 read:

```
StreamReader reader =
    new StreamReader(@"d:\secret""plan,txt")
StreamWriter writer =
    new StreamWriter(@"e:\emailToCaptainAmazing.txt")
```

Pass the file you want to read from into the StreamReader's constructor.

This progra.. uses a StreamReader to read the S...indler's plan, and a Strea..Writer to write a file that ....ill get emailed to Captain Amazing

```
writer.WriteLine("To: CaptainAmazing@objectville.net")
writer.WriteLine("From: Commissioner@objectiville.net");
writer.WriteLine("Subject: Can you save the day. again?");
writer.WriteLine();
writer.WriteLine("We've discovered the Swindler's plan:");
while (!reader.EndOfStream) {
  string lineFromThePlan = reader.ReadLine();
  writer.WriteLine("The plan -> " + lineFromThePlan);
}
writer.WriteLine();
writer.WriteLine("Can you help us?")
writer.Close()
reader.Close()
```

An empty WriteLine() ..ethod writes a blank line.

EndOfStream is the property that tells you if there's no data left unread in the file.

This loop reads a line from the reader and writes it out to the writer.

Make sure to close every strea.. that you open, even if you're just reading a file

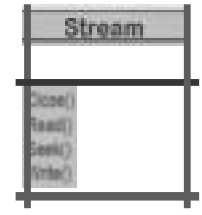
```
emailToCaptainAmazing.txt - Notepad
File Edit Format View Help
To: CaptainAmazing@objectville.net
From: Commissioner@objectville.net
Subject: Can you save the day... again?

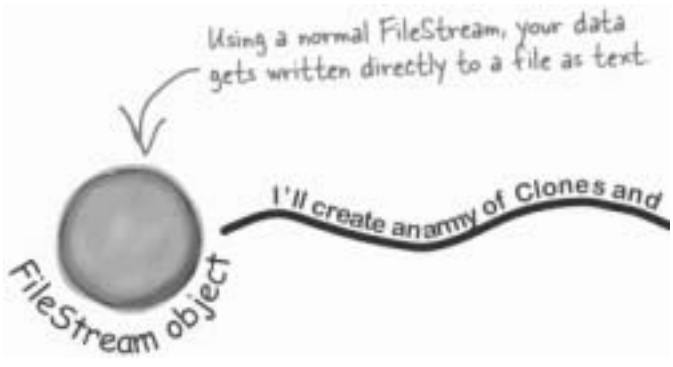
We've discovered 'h, Swindler's plan:
The plan ->       11 defeat Captain Amazing
The plan -> Another genius secret plan by The Swindler
The plan -> I'll create an army of clones an  unleash them upon the citizens of Objectville
The plan -> Clone #0 attacks the mall
The plan  > Clone #1 attacks downtown
The plan  , Clone .. attacks the .."
The plan -> Clone #3 attacks downtown
The plan -> Clone #4 attacks the ..."
The plan -> Clone #5 attacks downtown
The plan -> Clone #6 attacks the ma

Can you hel  us?
```

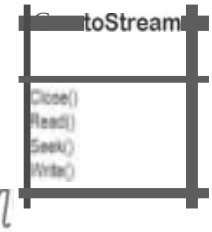
Data call go through more than one stream

One big advantage to working with streams in .NE'I' is that you can have y"'l' data go through mot'e lhan one su-eam on its way 1" its final d"'1inati,n. O!1" of the many types of streams that .NET ,hiI" with i, the CryptoSt reamclass. This lets you "Incrypt your data bef'n' you do anything else with it:

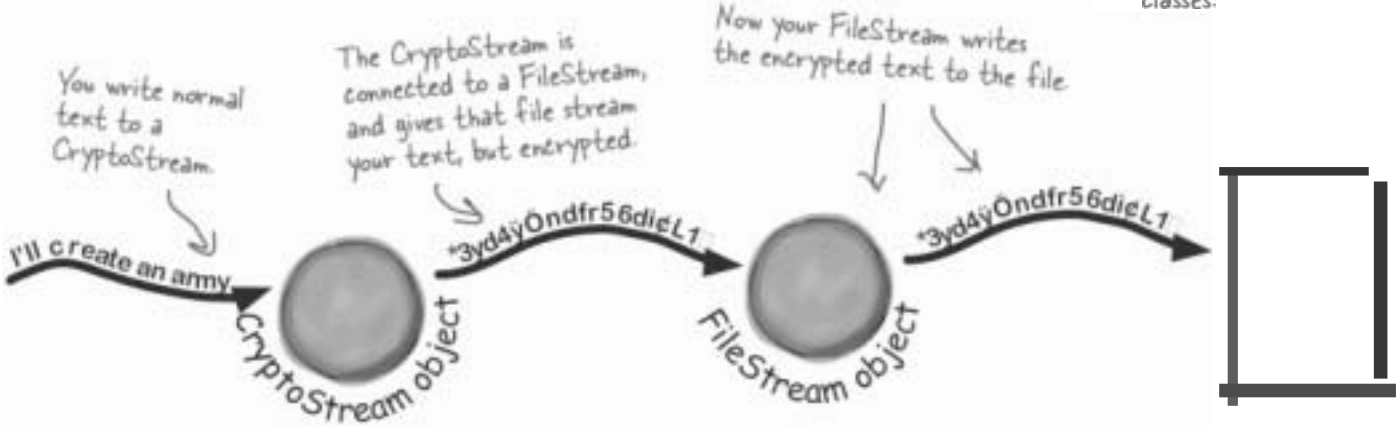








CryptoStream inherits from the abstract Strea... class, just like the other strea.. classes.

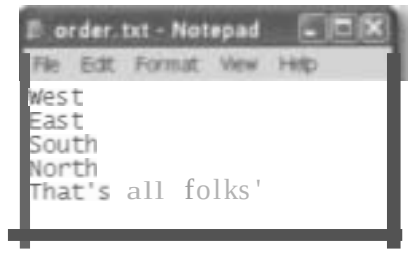


You Can **CHAIN** streams. One stream **can write to another stream, which writes to another stream... olten** ending **with** a **network** or file **stream.**

Pöö] Puzz]e



You, ***job*** is to tak.. code snippets from the pool and place th..m into lhe blank lines in Ihe program. You can use ihe sam.. .nippet more than once, and you won'" need 10 use all the snippets. Your ***goal*** i, to mak.. the program produc.. the output shown to the right.




```
public class Pineapple I
  const      __ d = "delivery.txt";
  public     _______
        { North, South, East, West, Flamingo }
  public static void Main()
    _________ o_ new              ("order.txt.."):
    Pizza pz. new Pizza(new              Id, truejJr
    pz,         (Fargo.Flamingo);
    for 1       w - 3; w >- 0: w--)
      Pizza i • new Pizza
            (new              (d, false));
      i.Idaho((Fargo)w):
      Party p _ new Party(new              idl);
      p,            (o);

    c-.           ("That's all folks!");
    o.           ();
```

```
public class Pizza I
  private                    -      _
  public Pizza(                      , I
    ______.Writer = Writer;

  public void                .Fargo tl
    Writer.          (f.ToStrinqOI;
    Writer.           0 ;
```

```
public class Party (
  private              Reader;
  public Party(          _   Reader)
    _ _ _ _ _ _.Reader   Reader;
```

NOll': **each snippet** from the pool Can **be used** mo.. Ih. Dnc.!

int
long
string
enum
class

Stream
Reader
Writer
StreamReader
StreamWrit..,
Open
)lose

public
privat..
this
class
static

for
while
foreach

•
>=
<=
'=
)
++

Fargo
Utah
Idaho
Dakota
Pineapple



Pool Puzzle Solution

This enum is used with its ToString() method to print a lot of the output

```
public class Pineapple {
    const string d = "delivery.txr";
    public enum Fargo { North, South, East, West, Flamingo
    public static void Main() {
        StreamWriter o = new StreamWriter("order.txt");
        Pizza pz = new Pizza(new StreamWriter(d, true));
        pz.Idaho(Fargo.Flamingo);
        for (int w = 3; w >= 0; w--) {
            Pizza i = new Pizza(new StreamWriter(d, false));
            i.Idaho((Fargo)w);
            Party p = new Party(new StreamReader(d));
            p.HowMuch(o);
        }
        o.WriteLine("That's all folks!");
        o.Close();
```

Here's the entry point for the program. It creates a StreamWriter that it passes to the Party class. Then it loops through the Fargo members, passing each of them to the Pizza Idaho() method to print

The Pizza class keeps a StreamWriter as a private field, and its Idaho() method writes Fargo enums to the file using their ToString() methods.

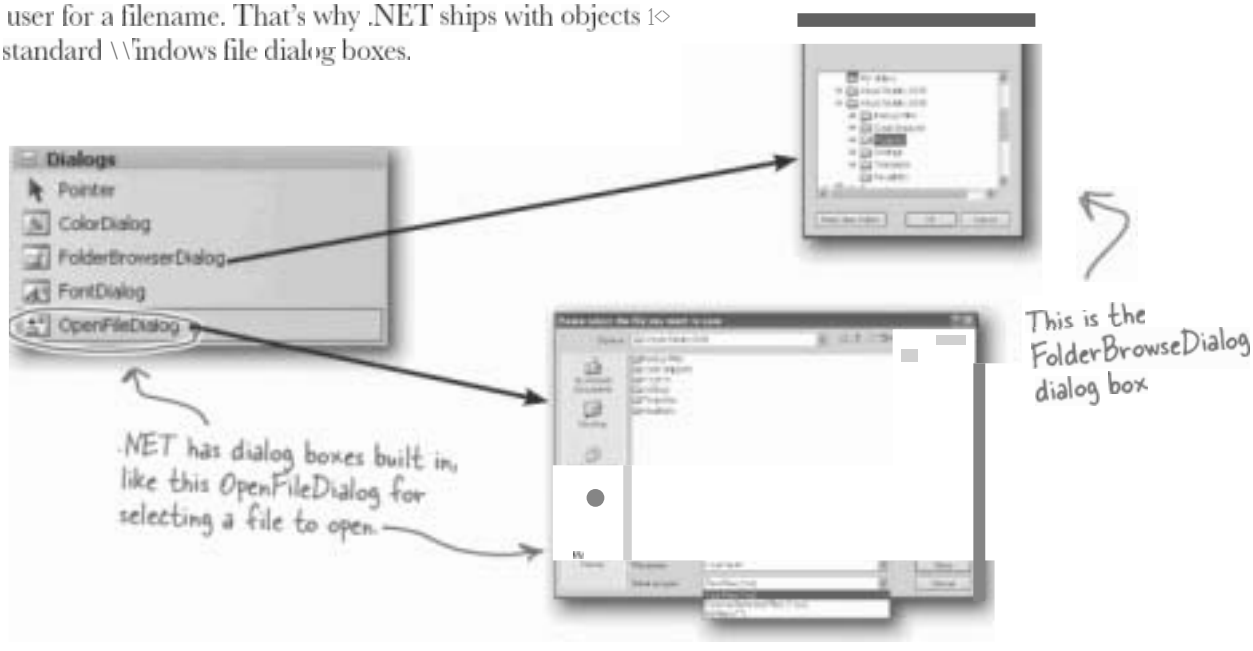
```
public class Pizza {
    private StreamWriter Writer;
    public Pizza(StreamWriter Writer) {
        this.Writer = Writer;
    }
    public void Idaho(Pineapple.Fargo f)
        Writer.WriteLine(f.ToString());
        Writer.Close();
```

The Party class has a StreamReader field, and its HowMuch() method reads a line from that StreamReader and writes it to a StreamWriter

```
public class Party {
    private StreamReader Reader;
    public Party(StreamReader Reader)
        this.Reader = Reader;
    }
    public void HowMuch(StreamWriter q)
        q.WriteLine(Reader.ReadLine());
        Reader.Close();
```

Use built-in objects to pop up standard dialog boxes

When you're working "n a program that reads and writes files, there's a good chance (h"r vou'll ue"d 1" pop up a dialog box at some p"iul to prompt the user for a filename. That's why .NET ships with objects 1◇ pop "I' the standard \\'indows file dialog boxes.



ShowDialog() pops Up a dialog box

Displaying a dialog box is easv. Here's ;,ll vou need tu do:

We'll walk you through these steps in a minute.

O Creatc an instance of rhe dialog box "hi'-"!. You can d" this in cod,' using new. or you cau drag it "m" your form "Ul of the toolbox.

O Set rhe dialog box object's properties. A few "„-f,,1 ,,"'., ind"de `TitIe` (which sets lit,. text in rlk' titl,' bar), `Init ia IDirectory` (which tells il which directory 1" open first:',and `Fi IeName` (for open and save dialog boxes).

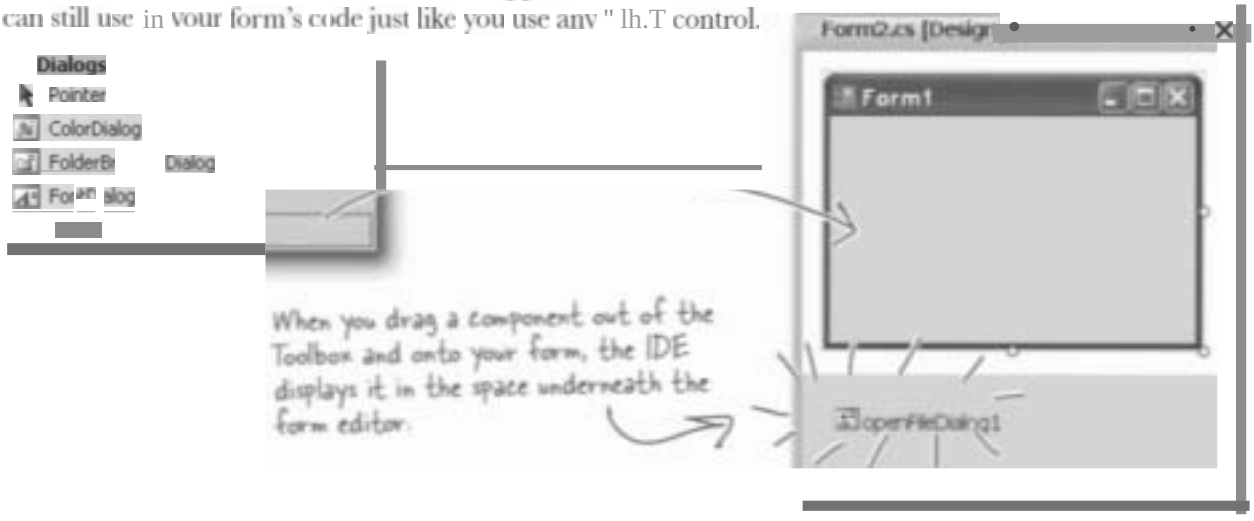
❸ Call the object's `ShowDialog ()` method. That pop' up IIie dialog box. and ,Ioe,u'l r,'{urn until Ih,' user clicks the ()K I"'ll"" or the Cancel 1,"1" m, ,'r closes ril,- window.

O The `ShowDialog ()` method n-tm'ns a `Dta IoqkesuLt`.. whkh i, an r-num. S"m,· of its members are `OK` (which means the u-cr clicked OK), `Cancel`, `Yes`, and `No` (for Yes/Nn di"I, ıg Ioxes).

Vialog boxes are Just another .NET cOl1trol

You ,-,'" add Windows standard lii,' ,hal,ıg Ixoxes 10 your prog·am just by dragging them 1" vour form jus•·h ag an OpenFileDialog control "Ill or the toc,Ibox ,und drop it "" your form. Instead "I' showing "I' as a visual control, you'll see il appear ill the space below vour form. That's Ixecause it's a ("oDlpon..nt_which is a special kind of **non-vi..ual** Tool!>**ox control** that doesn'I appear directly on the ku-m. but which vot can still use in vour form's code just like you use anv " lh.T control.

"Non-..sual" just eans it doesn't appear your form when r- drag it out of the toolbox



The InitialD..ectory propery changes the folder that's first displayed when the dialog opens.

```
openFileDialog1.InitialDirectory - @"c:\MyFolder\Default\";
openFileOialogl.Filtec - "rext. Files (*.txt)|·.txtl"
  + "Comma-Delimited Files (*.csv)|*.csv|All Files (*.*)|*.*";
openFileDialog1.FileName - "default_fLre.txt";
openFileOialogl.CheckFileExists - true;
openFileDialog1.CheckPathExists - false;
DialogResult result - openFileDialogl.ShowDialog();
if freauLt; -- DialogResult.OK)|
    OpenSomeFile(openFileOialogl.FileName);
```

The Filter property lets you change the filters that show the bottom of the dialog box, such as what types of files to show

These properties tell the dialog box to display an error ..essage if the user tries to open up a file path that doesn't exist on the d e

Display the dialog box using its ShowDialog() method, which returns a DialogResult. That's an enum that you can use to check whether or not the user hit the OK button. It'll be set to DialogResult.OK if the user clicked OK, and DialogResult.Cancel if he hit Cancel.

l1ialog boxes are objects. too

.\n OpenF11e01alQg box object shows the ,1"l1dard Windows "()pen' window, and the SaveF11e01alog shows its "Save' window. You can display them by creating a new instance, setting the propertie Oil the object, and calling its ShowDialog () method. The ShowDialog () "",thod n'tum, a Di03logResu1t '-llllIll(because some dialog boxes have mo e than two buttons or results, so a simple bool wouldn't 1", enough).

When you drag a seve dialog object out of the toolbox and onto your for"', the IDE just adds a line like this to your for ", InitializeComponent() method

```
saveFileDialogl - new SaveFileDialog();
saveFileDialogl.InitialDirectory - @"c:\MyFolder\DefauIt\";
saveFileDialogl.Filter - "Text Files (·.txt)|*.txt|"
  + "Comma-Delimited Files (•.csv)|*.csvIAll Files (*.*)I' .".
DialogResult result - saveFileDialogl.ShowDialog{);
if (result -- DialogResulLOK){
    SaveTheFile(saveFileDialogl.FileName);
```

The ShowDialog() and FileName properties work exactly the same as on the OpenFileDialog object

The SaveFileDialog object pops up the standard Window' "Save as..' dialog box.

The Title property lets you change this text



The ShowDialog() method pops up the dialog box and opens the folder specified in the InitialFolder property.

The DialogResult returned by the ShowDialog() method lets you figure out

Use the built-ill File alld lIirectory classes to work with files and directories

Like StreamWeitee, II,,' Fi Ie class c,..ates streams for you 10 work with file behind the scenes. You can use its n"·lhods to do m",t com1110n actions will"""1 having 10 create the Fi 1e Streams first. Di rectory ohj,·cts let you work with whole directories full of files yo,u can use il u>make changes I'>\', 'Ur file structure easily.

Things you can do with a File:

- **Find out if it exists**
  You call check to see if a file exists using the Exists () melhod. It'll return true if il doe 'llld false if it doesn't.

- **Read from and write to the file**
  You can 'N' Ihe OpenRead () method to get data from a file, or the Create () "r OpenWrite () method to write 10 the file.

- **Append text to the file**
  -n; AppendAll Text O method lets you "1'1"""d text to an already created file. It even creates the Iii,' if it's nol the-re when the method runs.

- **Get information about the file**
  The GetLastAccessTime () and GetLastWriteTi me () """lhods r..turn the date and time when the file was last accessed and modified

FileInfo works just like File

If you're going to be doing a lot of work with a file, you ..ight want to create an instance of u, FileInfo class instead of using the File class's static ..ethods.

The FileInfo class does just about everything the File class does except r- have to instantiate it to use it. You can create a new instance of FileInfo and access its Exists () ..ethod, or its OpenRead () method in just the sa··e way.

The only difference i, that the File class " faster for a small number of actions and FileInfo is better suited for big jobs.

Things you can do with a Directory:

- **Create a new directory**
  Create a directory using lhe CreateDi rectory () rm-tbod.\ll you have to do is supply the path; lhi, method does lh,' rest.

- **Get a list of the files in a directory**
  You can create "'I array or files in a directory using the GetFiles () method; just nll the method which directory you want to know about and il will do the rest.

3. **Delete a directory**
   Deleting a directory is really simple too. Use the Delete () method.

there are no Dumb Questions

Q: I still don't get that {0} and {1} thing that was part of the StreamWriter.

A: When you're printing slnngs to a file. you'll often find yourself in the position of having to print the contents of a bunch otvanatsee For example, you might have to erne something like this

```
writer.WriteLine("My name is    + name +
    "and my age is " + age);
```

It gets really tedious and somewhat error-prone to have to keepusing + to combine strings. It's easier to take advantage ot{0J and It}'

```
writer_WriteLine(
  "My name is {0} and my age is 11)",
   name, age);
```

It's a lot easer to read thatcode, especially when manyvariables are included In the same line

Q: Why did you put a @ infront orthes tring that contained the filename?

A: The Writ.e() and WriteLine() methods support escape sequences like \n and VT. That makes it difficult to type filenames, whichhave a lot of backslash characters Inthem, If you put @ in front 01a slnng. mens C# not to interpret escape sequences It also tells C# to include line breaks in your string, so you can hit Enler halfway through the string and it'll include thatas a linebreak In the output:

```
string twor.tne = @"this is a string
that spans two lines,";
```

Q: And whal do In and It mean again?

A: Those are escape sequences. \n is a uoeteed and \t is a tab \r is a remmcharacter, or half of a Windows return—in Windows text files, lines have to end with \r\n, If you want to use an *actual* backslash in yourstring, and not have C# interpret it as the beginning 01 an escape sequence, justdo a ***double*** backslash: \\,

Q: What was that in the beginning about converting .. string to a byte array? How would that even work?

A: You've probably heard many times that files on a disk are rapesenled as bits and bytes What that means IS thai when you wnte a file to a disk, the operating system treats it as one long sequence 01 bytes Remember from Chapter 4 howa byte variable can store anynumber between 0 and 255? Every file on your hard dnve is one long sequence of numbers between 0 and 255, It's up to the programs that read and write those Illesto interpret those bytes as meaningful data wren you open a file in Notepad, it converts eacn individal byte to a character—for example, E is 69 and a is 97, And when you type text into Notepad and save it, Nolepad converts each of the characters back into a byte and saves it to disk. And if you wantlo wnte a string toa stream, you'll need to do the same,

Q: If f mjust using a SlreamWriter to writ, to, file, why do I really care if it's creating .. FileStream for me?

A: If you're only reading or writing lines to ortrom a text fite in order, then all you need are StreamReader and StreamWriter. Butas soon as you need to do anything more complex than that, you'll need to start working With other streams. If you ever need to write data likenumbers, arrays. collections or objects to a file, a StreamWriter just won't do, But don't worry, we'll go into a lot more detail about how that will work Injust a mmute

Q: What if I want to create my own dialog boxes? can I do that?

A: Yes, you definitely can. You can add a newform to your project, design it to look exactly howyou want. Then you can create a new instance 01 il with new (just like you created an OpenFileDialog object) Then you can can its ShowDialog() method, and it'll work just like any other dialog box We'll talk a lot more about adding other forms to your program in Chapter 13

Q: Why do I need to worry about closing streams after I'm done with them?

A: Have you ever had a word processor lell you it couldn't open a file because It was "busy"? When one program uses a file. Windows locks it and preventsoffer programs lromusing it. And it'll do thaifor your program when it opens a file. 11 you don't call the Close() method, then it's possible for your program to keep a file locked open until it ends

Sharpen your pencil



,NET has two built·in classes with a bunch of ,tatic method, lor working with files and folders. The File class gives methods to wor k with lile" and the Directory class lets you work with directories. Write down what you think each of these lines of code doe,.

Code	What the ecce does

`if (Directory.Exists(@"c:\SYP")) ;`
`  Directory.CreateDirectory(@"c:\SYP");`
`;`	

`if (Directory.Exists(@"c:\SYP\Bonk")) ;`
`  Directory.Delete(@"c:\SYP\Bonk"I;`
`;`	

`Directory.CreateDirectory(@"c:\SYP\Bonk");`	
`Directory.SetCreationTime(@"c:\SYP\llonk",` `  new DateTime(1976, 09, 25));`	
`string[] files = Direcrory.GetFiies(@"c,\windows\",` `  "▪.log", SearchOption.xi mi recrcrieg);`	

`Flle.WriteAllText(@"c:\SYP\Bonk\weirdo.txt",`
`  S-rtOe i the first line`
`and this is the second line`
`and this is the last line");`	

`File.Encrypt(@"c:\SYP\llonk\weirdo.txt");` See if you can guess what this one does—you haven't seen it yet.	
`File.Copy(@"c:\SYP\llonk\weirdo.txt",` `  @"c:\SYP\copy.txt");`	
`DateTime myTime =` `  Directory.GetCreationTime(@"c:\SYP\Bonk");`	
`File.SetLastWriteTime(@"c:\SYP\copy.txt", myTime);`	
`File.Delete(@"c:\SYP\Bonk\weirdo.txt"l:`	

Use File Dialogs to opell alld save files (all with just a few lilies of code)

You can build a pn'gram that opens a textfile. It'll k-t voum.rkc changes 1" the file, and saves yuur changes. with ", ry little code, all using standard .NET controls. Here' how .

Do this



1 **Build a simple** form.

•\ll you need is a textbox and two Imlt"'". Drop 11", ( )1"'ll File1Dialog and SaveFileDialog controls onlO the lonu too. Double-click on the buttons I" create lh"if event handler and **add a** p n\-at**e string field calle**d name to **th.. form** Don't forget (" 1"'1 a using statement up (op 1,,1' System.

2 **Hook the Open** button up to the **openFileDialog**.

The Open button shows an OpenFileDialog „„\(1 then use Fil, .ReadAllText() t" read the file inlo lhe text box:

```
private void open Click(object sender, EventArgs e) {
  if (opeнFileDialoql.ShowDlalogll == DlalogResult,OKJ {
    name • openFileDialogl.FileName;
    textBox1.Clear(l;
    textBoxl.Text - File.ReadAllText(name);
```

Clicki..g Ope.. shows the OpenFileDialog control

3 **Now,** hook up the **Save** button.

The Save button use the File.WriteAllText() method I" save rhcflb-:

```
private void save_Click(object sender, EventArgs e) {
  if (saveFileDialogl.ShowDialog() -- DialogResult.OK)
    name - saveFileDialogl.FileName;
    rile.WriteAIIText(name, textBoxl,Text);
```

The ReadAllText() and WriteAllText() ..ethods are part of the File class That's coming up on the next page We'll look at the··· in more detail in just a few pages

4 **Play with** the other **properties** of the **dialog** boxes.

- Use the Title property of the saveFileDialog I" change the I"XI in the title bar.
- Set the initialFolder property to have the dialog OpenFileDialog start in a specified directory.
- Filter the OpenFileDialog so it will only show text files using the Filter property.

If you don't add a filter, then the drop-down lists at the bottom of the open and save dialog boxes will be empty. Try using this filter: "Text Files (*.txt)|*.txt"



.NET has two built-in classes with a bunch of static methods for working with files and folders. The File class gives methods to work with files, and the Directory class lets you work with directories. You, job was to write down what each bit of code did,

Code	What the code does

if (Directory.Exists(@"c:\SYP")) {
  Directory.CreateDirectory(@"c:\SYP");
}
	Check if the C:\SYP folder exists. If it doesn't, create it.

if (Directory.Exists(@"c:\SYP\Bonk")) {
  Directory.Delete(@"c:\SYP\Bonk");
}
	Check if the C:\SYP\Bonk folder exists. If it does, delete it.

Directory.CreateDirectory(@"c:\SYP\Bonk");	Create the directory C:\SYP\Bonk.
Directory.SetCreationTime(@"c:\SYP\Bonk", new DateTime(1976, 09, 25));	Set the creation time for the C:\SYP\Bonk folder to September 25, 1976
string[] files = Directory.GetFiles(@"c:\windows\", "*.log", SearchOption.AllDirectories);	Get a list of all files in C:\Windows that match the *.log pattern, including all matching files in any subdirectory.
file.WriteAllText(@"c:\SYP\Bonk\weirdo.txt", @"This is the first line and this is the second line and this is the last line");	Create a file called "weirdo.txt" (if it doesn't already exist) in the C:\SYP\Bonk folder and write three lines of text to it.
file.Encrypt(@"c:\SYP\Bonk\weirdo.txt"); This is an alternative to using a CryptoStream	Take advantage of built-in Windows encryption to encrypt the file "weirdo.txt" using the logged in account's credentials.
file.Copy(@"c:\SYP\Bonk\weirdo.txt", @"c:\SYP\copy.txt");	Copy the C:\SYP\Bonk\weirdo.txt file to C:\SYP\Copy.tx
DateTime myTime = Directory.GetCreationTime(@"c:\SYP\Bonk");	Declare the myTime variable and set it equal to the creation time of the C:\SYP\Bonk folder.
file.SetLastWriteTime(@"c:\SYP\copy.txt", myTime);	Alter the last write time of the copy.txt file in C:\SYP\ so it's equal to whatever time is stored in the myTime variable.
file.Delete(@"c:\SYP\Bonk\weirdo.txt");	Delete the C:\SYP\Bonk\weirdo.txt file.

Illisposable l\Iakes sure your objects are disposed properly

,\ 1"1 of .NET classes implement a particularly useful interface called Hrisposable.Jr **only has** one **memb**..r: a method called Di spose() . Whenever a class implements I Di epoaab1e. it's telling you 11'"1 th"n' are important things that it needs 10 d" ill order to shut itself down, usually because it's **allocat**..d **resourc**..**s** that il won't give back \llIlil you k·n il 10. The Dispose() method i, how youl,'l! the object to release those resources.

You can use the "Go To Definition" feature in the IDE the "ni,;,,1 C# deli" ;lio,, of rnispoebte.(;" tu your project and type IDisposable anvwhere ill,;d,-tit" "(kl,'. Then right-click on il ami select h(;" 'n Definition" !;-';Illllw 1Il,'lIU, It'll open a new taltwith cod,' ill it. Expand all of the ('OI1,' and lhi, is what you'll see:

You'll learn more about "Go To Definition" later on.

Declare an object in a using **block** and that ol,ject's **Dispose()** method **is** called automatically.

A lot of classe llocate i..portant resources, like ..emory, files, d other objects. That ..eans they take the.. over, and don't give them back until you tell the.. it's done with those resources.

```
namespace System
{
    // Summary:
    //     Defines a method to release allocated resources.
    public interface IDisposable
    {
        // Summary:
        //     Performs application-defined tasks
        //     associated with freeing, releasing, or
        //     resetting unmanaged resources.
        void Dispose();
    }
}
```

Any class that imple..ents IDisposable will immediately ..elease any resou..ces that it took over as soon as you call its Dispose() ..ethod It's al..ost always the last thing r- do before you're done with the object

Go To Definition

There's a handy feature in the IDE that lets you automatically jump to the definition for any variable, object or method. Just right-click on it and select "Go To Definition", and the IDE will automatically jump right to the code that defines it.

ai-In-catc, verb.
to distribute resourcc-, or duties for a part icular purpose. *The programming team was irritated at their project manager because he* **allocated** *all f!f the conference rooms for (/ useless management ,1t1l1il/(fr.*

Avoid file system errors with **using** statements

We've been telling you all chapter that you need to **close** your streams. That's because some of the most common bugs that programmers run across when they start with files are caused when streams aren't closed properly. Luckily, C# gives you a great tool to make sure that never happens to you: IDisposable and the Dispose() method. When you wrap your **stream code** in a **using statement**, it automatically closes your streams for you. All you need to do is declare your stream reference with a using statement, followed by a block of code (inside curly brackets) that uses that reference. When you do that, the using statement automatically **calls** the **stream's** Dispose() method as soon as it finishes running the block of code. Here's how it works:

These "using" statements are different from the ones at the top of your code

A using statement is always followed by an object declaration...

...and then a block of code within curly braces.

```
using (StreamWriter sw = new StreamWriter("secret_plan.txt")) {
    sw.WriteLine("How I'll defeat Captain Amazing");
    sw.WriteLine("Another genius secret plan");
    sw.WriteLine("by The Swindler");
}
```

These statements can use the object created in the using statement above like any normal object.

When the using statement ends the Dispose() method of the object being used is run.

In this case, the object being used is the Log Stream, so the Dispose() method of the Stream class is run, which closes the stream

Every stream has a DisposeO method that closes the stream. So if you declare your stream in a using statement, it will always close itself!

Use multiple nesting statements for multiple objects

You can pile using statements on top of each other—you don't need extra sets of curly brackets or indents.

```
using (StreamReader reader = new StreamReader("secret_plan.txt"))
using (StreamWriter writer = new StreamReader("email.txt"))
{
    // statements that use reader and writer
}
```

You don't need to call Close() on the streams now, because the using statement will close them automatically.

Any time you **use a stream**, you should **ALWAYS declare** it **inside a using** statement. That makes **sure it's always closed!**

Trouble at work

Meet Brian. He like hi,j"b as a C# developer, but he *loves* taking the occasional day off. BUI his boss **hates** when people take vacation days sc Brian's got I" come lip with a goc>d excuse.

Sorry I've gOfto leave early, boss. My cat's got Q vet appointment.

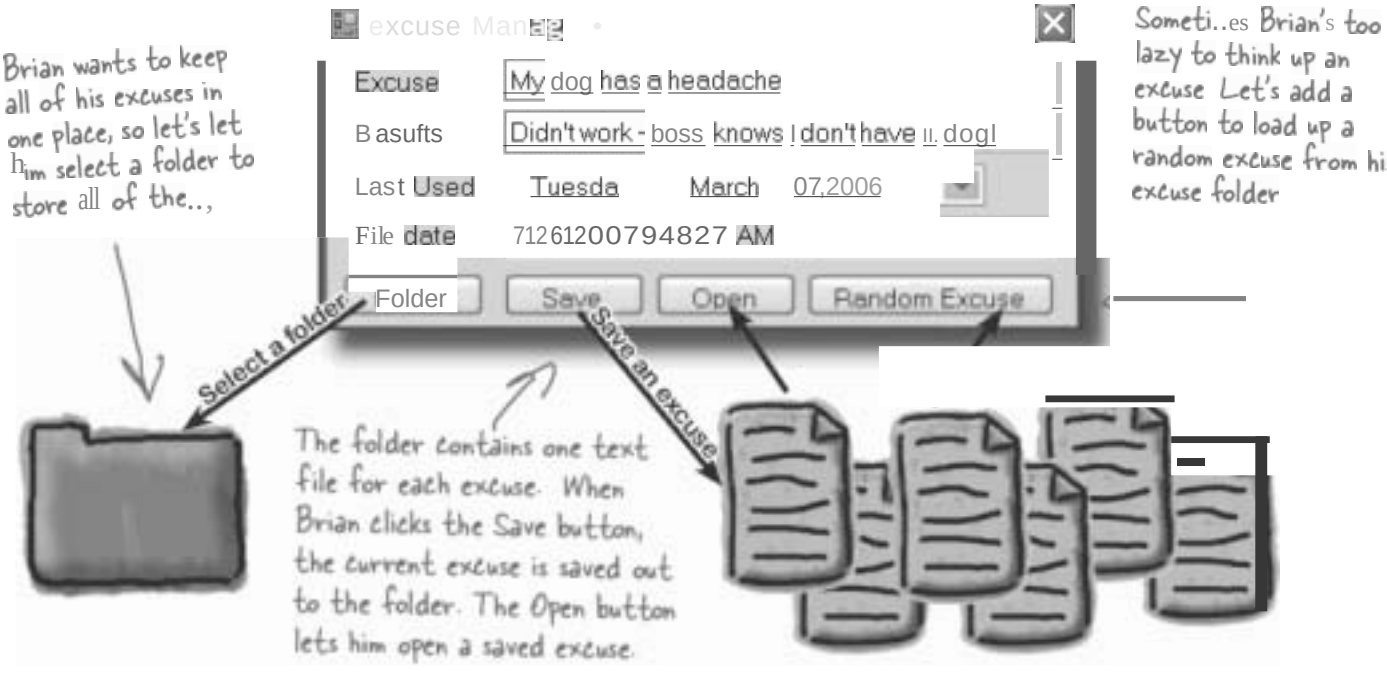


That's the ninth vet appointment you've had since Moreh, son. If I find out you're lying to me, you'd better start looking for a new job!

Vou ca" help Brian out by buildi"g a progra., to manage his excuses

Use what you know about reading "m] writing til," I"],,,ild an excuse manager lhal Brian can use to keep track of which excuses he's used recently and how well they ""'nl ,""'r with 11ıe boss.

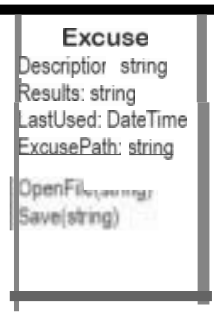
This asterisk appears when a form has unsaved data





Exercise

Build me excuse manager so Brian can manage his excuses at work



1 Build the form

This form has a few special features

- When the form's first loaded, only the Folder button should be enabled - disable the other three buttons until the user selects a folder.
- When the form opens or saves an excuse, it displays the file date for the excuse file using a Label control with AutoSize set to False and BorderStyle set to Fixed3D.
- After the excuse is saved, the form pops up an "Excuse Saving" messagebox.
- The Folder button brings up a folder browser dialog box. If the user selects a folder, it enables the Save, Open, and Random Excuse buttons.
- The form knows when there are unsaved changes. When there are no unsaved changes the text on the title bar is "Excuse Manager". If anything has changed any of the three fields, the form adds an asterisk (*) to the title bar. The asterisk goes away when the data is saved or a new excuse is opened.
- The form will need to keep track of the current folder, and whether or not the current excuse has been saved. You can figure out when the excuse hasn't been saved by **using the Changed event handlers** for the three input controls.

When you drag a textbox to a form and double-click on it, you create a Changed event handler for that field

2 Create an Excuse class and store an instance of it in the form

Now add a CurrentExcuse field to the form to hold the current excuse. You'll need three overloaded **constructors**: one for when the form's first loaded, one for opening up a file, and one for a random excuse. Add methods `OpenFile()` to open an excuse (for the constructors to use), and `Save()` to save the excuse. Then add this `UpdateForm()` method to update the controls (it'll give you errors, but finish the class):

```
private void UpdateForm(bool Changed) {
    if (!Changed) {
        this.description.Text = currentExcuse.Description;
        this.results.Text = currentExcuse.Results;
        this.lastUsed.Value = currentExcuse.LastUsed;
        if (!String.IsNullOrEmpty(currentExcuse.ExcusePath))
            fileDate.Text = File.GetLastWriteTime(currentExcuse.ExcusePath).ToString();
        this.Text = "Excuse Manager";
    }
    else
        this.Text = "Excuse Manager*";
    this.formChanged = Changed;
}
```

This parameter indicates whether or not the form has changed. You'll need a field in your form to keep track of this status.

Remember, the ! means NOT—so this checks if the excuse path is NOT null or empty.

Double-click on the input controls so the IDE builds Changed event handlers for you. The event handlers for the three input controls will first change the Excuse instance and then call UpdateForm(true)—then it's up to you to change the field your form.

3 Make the Folder button open a folder browser

When the user clicks on the Folder button, the form should pop up a "Browse for Folder" dialog box. The form will need to store the folder in a field so that the other dialog boxes can use it. When the form first loads, the Save() and Random Excuse buttons are disabled, but if the user selects a folder then the Folder button enables them.

3 Make Save button save the current excuse to a file

Clicking the Save hUII"" should bring up the Save As dialog box.

- Each excus‹ is saved 10 a separate text file. The lir;1 lim-01 the file is the excuse. the second is the result. and the Ihird is Ihe date last used (use the DateTimePicker's `ToSt r ing ()` method). The `Excuse` class should have a `Save () ",eIhod` 10 save an excus‹ out to a specifiedlile.
- When the Save As dialog box is opened, its folder should be set 10 tl"' I, ,ld,'r rh.n the user selected using Ih,' Folder button, and Ih,' filename should be set I" the excuse plus a ".txt" extension.
- Tl«- dialog box should have two filters: 'Ii-xt Files (*.txt) and All Files (•.-).
- If the user Iri," to save II", nllr.'lll excuse IJIII has left t·ilher Ihe excuse or lite result blank. rhe form should pop " I' a warning dialog box:

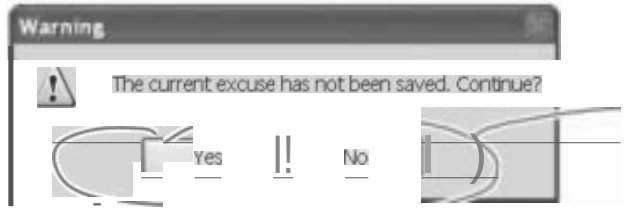
You can display this Exclamation icon by using the overloaded MessageBox.Show() method that allows you to specify a MessageBoxIcon parameter.



O Make the Open button open a saved excuse

Clicking Ihe Save hUI tu>ll should bring up the Open dialog box.

- \When the ()p"1l dialog box is opened, its folder should Ibe set 1" Ih,' folder Ih"l the user selected using the Folder button.
- Add an `Open ()` method tu Ihe `Excuse` class 10 open an excuse from a given file.
- Use `Convert , ToDateTime ()` I" load the saved date inlo the DateTimePicker control.
- If the user tries 10 0l"'n the current excus‹ but the current excuse hasn't been saved, it pops up this dialog box:



Show a Yes/No d;alog box by using the overloaded MessageBox.ShowO method that lets you specify the MessageBoxButtons.YesNo parameter. If the user clicks "No", then Show() returns D,alogResult.No

O Finally. make the Random Excuse button load a random excuse

\\·lre" Ihe user clicks the Random Excuse hUllon . il looks i" the excuse folder, chooses orr.' of the excuses at random, and 01"'''' it.

- The form will need to save a `Ra ndom` object in a field and pass il ro",\<, " f the overloaded constructors or the `Excuse` object.
- If the current excuse hasn't bee" saved, Ihe button ,lrould pop up 11re same warning dialog box as the (II"'" huilon.



Exercise Solution

Build the excuse manager so Bnan can manage his excuses at work

```
private Excuse currentExcuse _ new Excuse();
private string eeiectecrorcer = "";
private bool formChanged = false;
Random random • new Random();
```

The form uses fields to store the current Excuse object, the seI_ected folder, remember whether or not the current excuse has changed, and keep a Random object for the Random Excuse button.

```
private void folder Click(object sender, EventArgs e) (
    folderBrowserDidlogl.SelectedPath - selectedrolder;
    DialogResult result · fOlderBrowserDiaLogl.ShOWDialog();}
    if (result == DialogResult.OK) (
        selectectrolder • folderBrowserDialogl.SelectedPath;
        save.Enabled · true;
        open.Enabled • true;
        ranctomExcuse.Enablect = true;
)
```

If the user selected a folder, the form saves the folder name and then enables the other three buttons.

The two vertical bar mean OR—this is true if description is empty OR results is e..pty

```
private void save_C11ck(object sender, EventArgs e) {
    1f (String.IsNullOrEmpty1description.Text) || String.IsNullOrEmpty(results.Text)) (
        MessageBox.Show("Please specify an excuse and a result",
            "Unable to save", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        return;

    saverlleDialogl.lnitiaLDirectory • selectedf'olderr
    saveFileDialogl.Filter . "Text files ('.txtl'.txtlAll files 1'.')1'.'";
    saveFileDialogl.FileName · description.Text + ".txt";
    DialogResult result = saveFlleDialogl.ShowDialog();
    if (result == DialogResult.OK) (
        currentExcuse.SaveIsaveFileDialogl.FileName);
        Updaterorm(false);
        MessageBox.Show("Excuse written");
```

Here's where the filters are set for the Save As dialog

This w[illegible] cause two rows to show up in the "File• of Type" dropdown at the bottom of the Saye dialog box: one for Text F[illegible], and one for Ail Files (*.*)

```
private void open_Click(object sender, EventArgs e) {
    if (CheckChanged O) (
        openFileDialogl.lnitialDirectory = selectedFolder;
        openFileDialogl.Filter . "Text files I'.txt)1'.txtlAll files (*.*)|*.*";
        openFileDialogl.FileName - description.Text + ".txt";
        DialogResult result = openFileDialogl.ShowDia  g();
        if (result -- DialogResult.OK) {
            currentExcuse = new Excuse(openFileDialogl.FileName
            UpdateForm(false);
```

Use the DialogResult enum ...eturned by the Open and Save dialog boxes to make sure 'lou only open or save if the user clicked "OK", and not "Cancel".

```
private void randomExcuse_Click(object sender, EventArgs e)
    if lCheckChangedO) {
        currentExcuse _ new Excuse(random, selectedFolder);
        UpdateFormlfalse);
```

```
private bool CheckChanged()
    it IformChanged) (
        DialogResult result   MessageBox,Show(
              "The current excuse has not been saved. Continue?",
               "Warning", MessageBoxButtons,YesNo, MessageBoxlcon.Warninq);
        it (result == Q.ialogResult.No)
            return false;
    J
    return true;
```

MessageBox.Show() alse retu..ns a DialogResult enum that we can check

Here are the three Changed event handlers for the three input fields on the for... If any of the are triggered, that eans the excuse has changed, so first we update the Excuse instance and then we call UpdateFor...0, add the asterisk to the for...', title bar, and set Changed to true.

```
private void description TextChanged(object sender, EventArgs e) {
    currentgxcuse.Description = description,Text;
    UpdateForm(true);


private void results_T.."tChanged(object sender, EventArgs el {
    currentExcuse.Results • results.Text;
    Updateform(true);


private void lastUsed_Valuechanged(object sender, EventArgs e)
    currentExcuse.LastUsed = lastused.Value;
    UpdateForm(true);
```

Passing true to UpdateForm() tells it to just ..ark the for... as changed, but not update the input controls.

```
public class Excuse {
    public string Description;
    pUblic string Results;
    public DateTime LastUsed:
    public string ExcusePath:
    public Excuse() {
        Excusepath = "";

    public Excuse(string excusePath)
        OpenFile(excusePath):
    }
    public Excuse(Random random, string folder) {
        string[J fileNames • Directory.GetFiles(folder, "*.txt") r
        OpenFile(fileNames[random.Next(fileNames.Length)] I;
    }
    private void OpenFilelstring excusePathl {
        this.ExcusePath = excueepat.tu
        using (StreamReader reader . new StreamReader(excusePath))
            Description = reader.ReadLinell;
            Results = reader.ReadLine();
            LastUsed = Convert.ToDateTime(reader.ReadLine I);
    |

    publlc voict seve tstrrnc fileName) (
        USIng (StreamWrlter writer _ new StreamWrlter(flleName))
        {
            wrlter.WriteLlne(Descrlptlonl,
            wnter.WnteLIne(Results);
            wrlter.WrlteLlne(LastUsed.ToStrlng());
```

The Random Excuse button uses Directory.GetFiles() to read all of the text files in the selected folder into an array, and then chooses a random a....ay index to open.

We made sure to use a using state...ent every ti...e we opened a st"ea.. That way our files wi always be cl sed

Here's where the using state...ent comes in. We dec ..ed the StreamW..iter inside of a using state...ent, so its Close() ..ethod is ca ed for us automatically

Writing files usually involves making a lot of decisions

You'll write lots of programs that take a single inpul. maybe fi-olll a file, and have I" d",'ide what to do based ,on Ih", input. Here's code that uses one long `if` statement— ii" pretty typical. It checks the `part` variable and prints different line tu the file based on which enum it uses There are lots "l' choices, so ["1' "1' `else ifs`:

```
enum Bodypert {
    Head,
    Shoulders,
    Knees,
    Toes
```

Here's an enum- we'll want to compare a variable against each of the four members and write a different line to the StreamWriter depending on which one it ..atches. We'll also write something different if none of the.. ..atch

```
private void WritePartInfo(BodyPart part, StreamWriter writer)
  if (part -- BodyPart.Head)
    vriter WWtt.et.tne tvt.he head is hairy");
  else if (part -- BodyPart.Shoulders)
    writer.WriteLine("the shoulders are broad"};
  else if (part -- BodyPart.Knees)
    wrLter.WriteLine('·the knees are knobby");
  else if (part __ BodyPart.Toes)
    writer.WriteLine("the toes are teeny");
    wrLter cwrLteLtne tvsome unknown part is unknown");
```

If we use a series of `else` statements, then we end up writing this "if (part ==[option])" over and over.

We've got a final else in case we didn't find a match.



What sort of things can go wrong when you write code that has this many if/else statements? Think about typos and bugs caused by brackets, a single equals sign, etc,

Use a switch statement to choose the right option

Comparing one variable against a bunch of different values is a really common pattern that you'll see over and over again. It's especially common when you're reading and writing files. It's so common, in fact, that C# has a special kind of statement designed specifically for this situation.

A switch statement lets you compare one variable against many values in a way that's easy to read and is compact. Here's a switch statement that does exactly the same thing as the series of if statements on the previous page:

There's nothing about a switch statement that's specifically related to files. It's just a useful C# tool that we can use here.

A **switch** statement **compares** ONE variable against **MULTIPLE possible** values.

```
enum BodyPart
{
    Head,
    Shoulders,
    Knees,
    Toes,
}

private void WritePartInfo(BodyPart part, StreamWriter writer)
{
    switch (part)
    {
        case BodyPart.Head:
            writer.WriteLine("the head is hairy");
            break;
        case BodyPart.Shoulders:
            writer.WriteLine("the shoulders are broad");
            break;
        case BodyPart.Knees:
            writer.WriteLine("the knees are knobby");
            break;
        case BodyPart.Toes:
            writer.WriteLine("the toes are teeny");
            break;
        default:
            writer.WriteLine("some unknown part is unknown");
            break;
    }
}
```

You'll start with the switch keyword followed by the variable that's going to be compared against a bunch of different possible values.

The body of the switch statement is a series of cases that compare whatever follows the switch keyword against a particular value.

Every case must end with "break;"

Each of these cases consists of the case keyword followed by the value to compare and a colon. After that is a series of statements followed by "break;". Those statements will be executed if the case matches the comparison value.

Switch statements can end with a default block that gets executed if none of the other cases are matched.

Use a **switch statement** to let your deck of cards read **from** a file or write itself out to olle

\Vriting a card out to a file is straightforward jn,t mak,' a loop tha i writes the name or each card ont to a file. Here's a method you can add to the Deck "hj"'·llh.l! doe ex.utlv that:

```
public void WriteCdrds(string Filename) {
    using (sr reamarIr.er writer = new StreamWriter(Filename))
        for (int i = 0; i < Cards.Count; i++)
            writer.writeLine(Cards[i J•Name);
```

Hutwhat about reading tht" file in? It's n"t '1"it,· so vimpk-. That' where tht"switch statement can come in handy.

The switch statement lets you **test** one value **against** a I,unch 01 **cases** and **execute different** statements **depending** on **which** one it matches.

```
Card.Suits suit;
switch (suitStrinq) {
    case "Spades":
        suit = Card.Suits.Spades;
        break;
    case "Clubs":
        suit = Card.Suits.Clubs;
        break;
    case "Hearts":
        suit = Card.Suits.Hearts;
        break;
    case "Diamonds":
        suit = Card.Suits.Diamonds;
        break;
    default:
        MessaqeBox.Show(suitstrinq + " isn't a valid suit!");
}
```

The switch statement starts with a value to compare against. This switch statement is called from a method that has a suit stored ... a string.

Each of these **case** lines compares some value against the value in the switch line. If they match, it executes all of the following statements until it hits a **break**

The **default** line comes at the end. If none of the cases match, the statements after the default get executed instead.

Add an overloaded Deck() constructor that reads a deck of cards in frol\l a file

You can use a switch statement 10 build a new constructo for the D,-,-k class that you wrote ill the last chapter. Thi, constructor reads ill a file and checks each Ii,,e for a card. Anv valid card gets added (0 the deck.

There's a Illt'lhod that you can find Cl every string that'll come ill handy: Split (). II lets you split the string illl<>;ttl array "I' substrings by passing it it char lJ array of separator character that it'll use to 'pl;1 the string up.

This line tells C# to split the nextCard st..ing using a space as a separator character. That splits the string "Six of Diamonds" into the array {"Six", "of", "Diamonds"}.

```
pUblic Deck(string Filename) (
    Cards = new List<Card> ();
    StreamReader reader = new StreamReader(Filename);
    while (!reader.EndOfStream) {
        bool invalidCard = false;
        string nextCard = reader.ReadLine();
        string[J cardParts - nextCard.Split(new char[] {      ));
        Card.Values value = Card.Values.Ace;
        switch (cardParts[OJ)
            case "Ace": value = Card.Values.Ace; break;
            case "Two": value - Card.Values.Two; break;
            case "Three": value = Card.Values.Three; break;
            case "Four": value - Card.Values.Four; break;
            case "Five": value = Card.Values.Five; break;
            case "Six": value - Card.Values.Six; break;
            case "Seven": value = Card.Values.Seven; break;
            case "Eight": value - Card.Values.Eight; break;
            case "Nine": value = Card.Values.Nine; break;
            case "Ten": value - Card.Values.Ten; break;
            case "Jack": value = Card.Values.Jack; break;
            case "Queen": value - Card.Values.Queen; break;
            case "King": value = Card.Values.King; break;
            default: invalidCard - true; break;

        Card.Suits suit - Card.Suits.Clubs;
        switch (cardParts[2]) {
            case "Spades": suit - Card.Suits.Spades; break;
            case "Clubs": suit = CerdvSuitsvCfunsr break;
            case "Hearts": suit - Card.Suits.Hearts; break;
            case "Dkamonds"  suit = Card.Suits.Diilmonds; break:
            default: invalidCard - true; break:
        |
        if (!invalidCardl {
            Cards.Add(new Cardtsu.it, valuejjs
```

This switch statement checks the first word in the line to see if it matches a value. If it does, the right value is assigned to the value variable.

We do the same thing for the third word in the line, except we convert this one to a suit.



All that code just to read in one simple card? That's way too much work! What if my object has a whole bunch of fields and values? Are you telling me I need to write a switch statement for each of them?

There's an easier way to store your objects in files. It's called serialization.

Instead of painstakingly writing out each field and value line by line, you can save your objects the easy way by serializing them and writing them to a stream. ***Serializing*** an object is like flattening it out so you can slip it into a file. And then, on the other end, you can ***deserialize*** it, which is like taking it out of the file and **inflating** it again.

What happens to an object when it's serialized?

It seems like something mysterious has tt> happen 10 an objec't in ord,'r to copy it uff "f the heap and put ir into a fib-. hUI it's actually pn-ttv straightforward.

0 Object on the heap

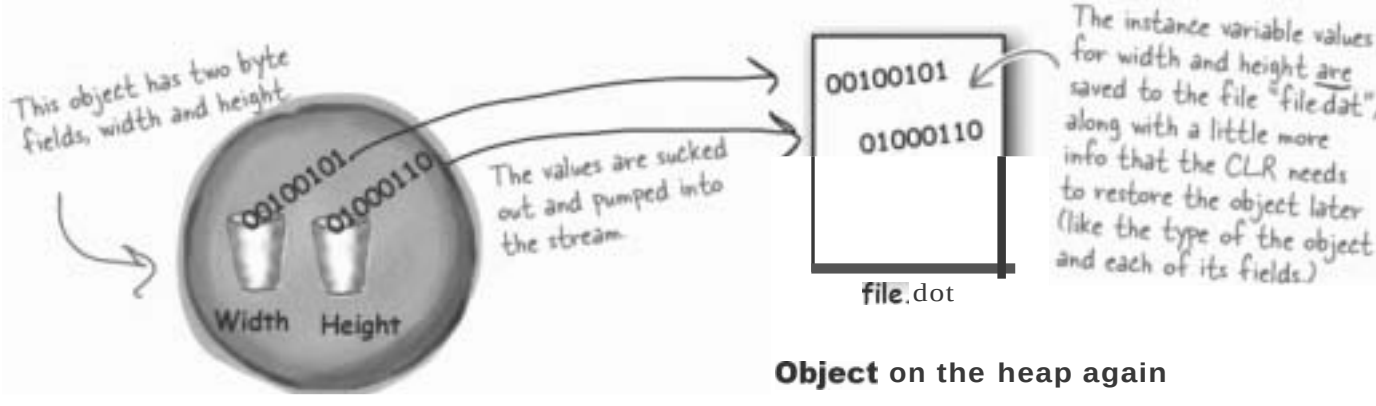


"'h"' you create an instance of ;m object, il has a **stat...** Everything that an object "knows" is wha1makes one instance of a class different from ;",<>1 her instance of the same class

2 Object serialized



\,'I,,'u C# serialize an "hj,-"I, ;l **saves th.. compl..t.. stat.. of th.. object**, so that an identical instance i',hi,x't) can be brought back 10life on tlu-heap later.



Object on the heap again



3 And later on...

Later— maybe days later, a1,tI in a different program you can go back It> fhe file and d**eserializ**.. it. That pulls the original class back t>Ul of Ih,' file and restores il **exactly as** it **was**. with all of its fields and values ;Ill;,,'!.

But what exactly IS all object's state? What lIeeds to be saved?

\\', . already know that an obj..ct "tor..s its stat.. in its fi..ld", So when an "hi","t is serialized, every "Ilt·"r those those fields needs 10be saved 1" the file.

Serialization starts to get interesting when you have more complicated objects. 37 and 70 are bytes— those are value types, '0 they ,-,,,, ju,t be written olll to a lil,· as-is. But what ir an object has an instance variable rh.u's an object *reference*? What ah"lIl an object that has five instance variables Ihal are object references? What ir those object instance variables themselves have instance variables?

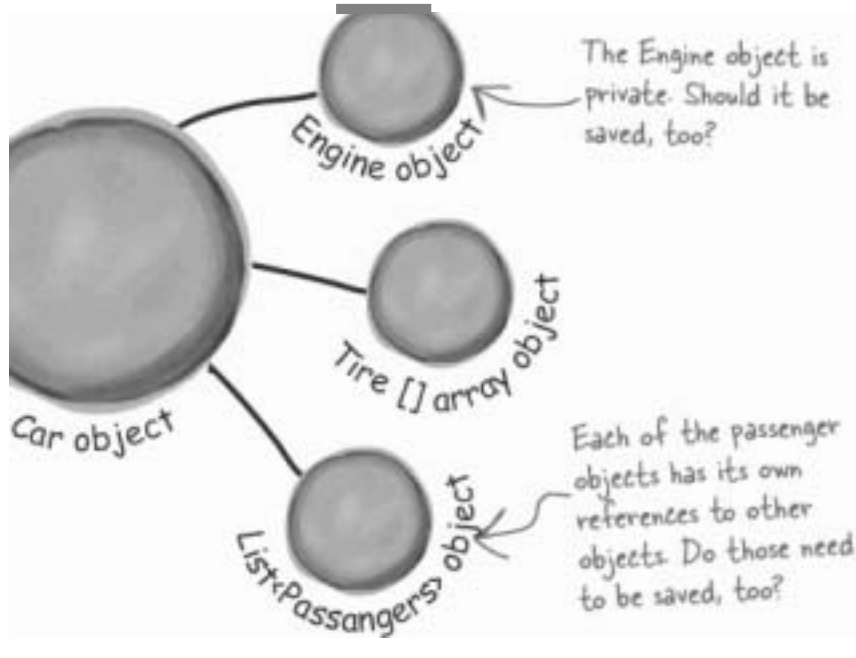
Think about it for a minute. What part or an object is potentially unique? Imagine what needs t" be restored ill order to get an object that's identical to th.. one that was saved. Somehow everything "11 the heap has to be written to Ih,' lii,',



BRAIN BARBELL

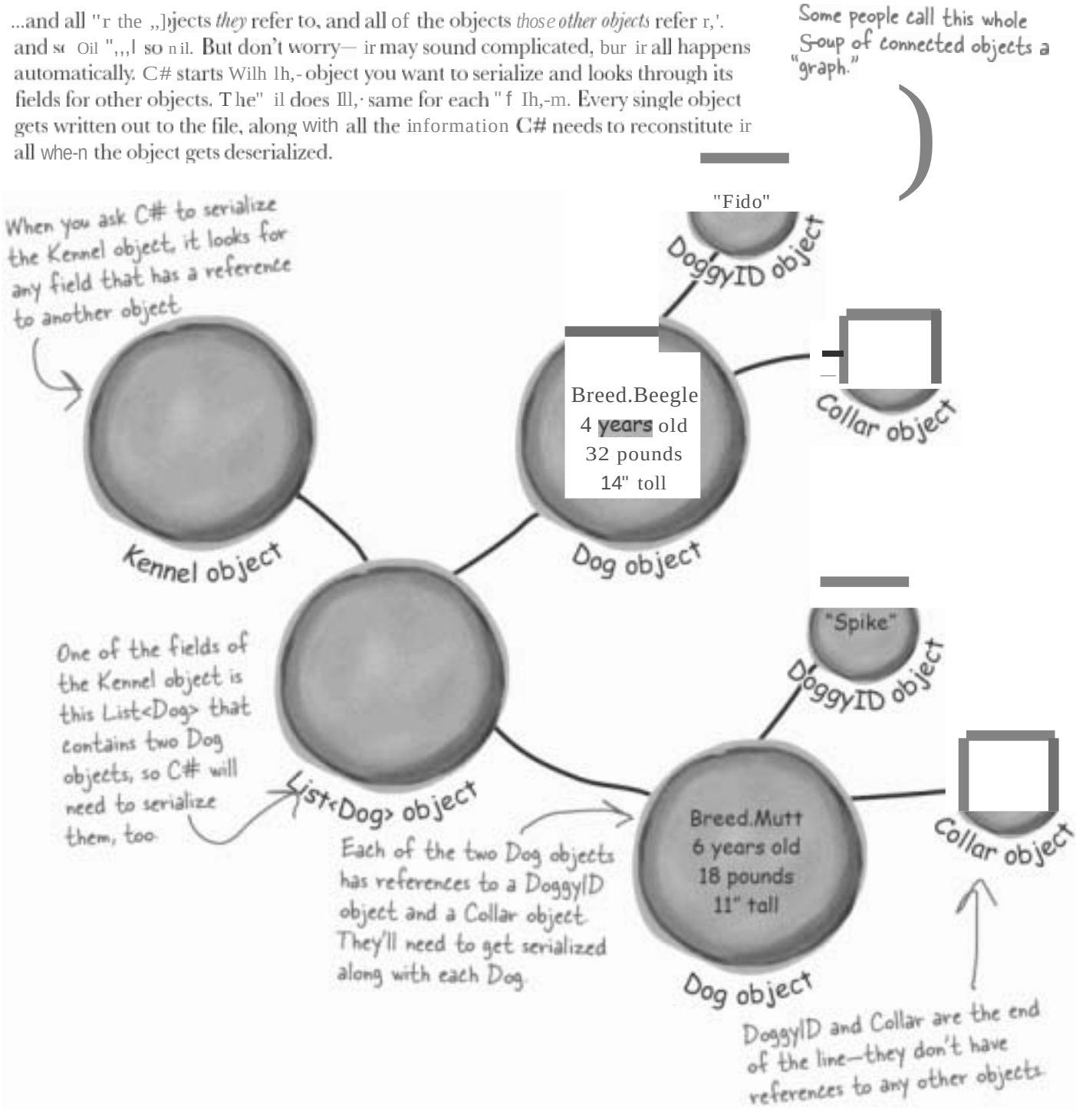
What has to happen for this Car object to be saved so that it gets restored back to its original state? Let's say the car has three passengers and a 3-liter engine and all-weather radial tires .. aren't those things are part of the state? What should happen to them?

The Car object has references to an Engine object, an array of Tire objects, and a List<> of Passenger objects. Those are part of its state, too—what happens to them?



Whell all object is serialized. all of the objects it refers to get serialized too...

...and all "r the „]bjects *they* refer to, and all of the objects *those other objects* refer r,'. and s(Oil ",,,l so nil. But don't worry— ir may sound complicated, bur ir all happens automatically. C# starts Wilh lh,- object you want to serialize and looks through its fields for other objects. The" il does Ill,· same for each " f Ih,-m. Every single object gets written out to the file, along with all the information C# needs to reconstitute ir all whe-n the object gets deserialized.



Serializatioll lets you read or write a whole object all at alice

It's quick to copy an object out to a file or ,'eaJ it *in* from One. You can serialize or deserialize it.

You'r· noljU'l limil"d I" reading and writing line of text I" your files. You can use s..rialization 10let your programs copy "nlin' objects 10Iii.s ;lll d read them hatk ill... all illjU 1a few lines of t"od,,! There's a tiny amou,,1 of pn'!, work you need to do add one [$eria1izable I Ii"" 10the tc'I' "j" the class to serialize- but "nc· you do Ihal. everything's ready I" write.

You'll .eed a Si.aryFormatler objeot

If you want 1" serialize an object—*any* object—the first Ihing you d" i, create an instance "j" BinaryFormatte l'. It's really straightforward I" do—and all it takes i, one line or code (and an extra `using` line at Ihe top of Ihe class file).

```
using system.Runtirne.Serialization.Formatters.Binary;

BinaryFormatter formatter = new BinaryFormatter();
```

The File.Create() ..ethod creates a new file. You can open an existing one using File OpenWrite().

Now just create a stream a.d read or write your objects

Use the Serialiee () me-thod front Ihe BinaryFormatter "hjec·1 1" write any object out 10;' stream.

```
Stream output = File.Create(filenameString);
formatter.Serialize(output, objectToSerialize);
output.Close();
```

If you use a using state..ent, Close() w,ll be zalled automatically

The Se..ialize() ..ethod takes an object and writes it out to a st··ea·· That's a whole lot easier than building a method to write it ou" yourself!

And once you've got an "hj'·"1 serialized ,,,tl 10a file, use the BinaryFormatter object's Deserialize() method I" read it back in. The method returns a reference, s(\'011 ",ed 1" cast the uutput so that il matches t},,· type of the reference variable you're copying ill",

```
stream input = File.OpenRead(filenameString);
SomeObj obj = (SomeObj)formatter.Deserialize(input);
input.Close();
```

When you .se Dee..ialize() to read an object back fro.. a strea..., don't forget to cast the return val..e to ..atch the type of object you'..e reading.

If you want your class to be ssrtaltzable, mark it with the LSerializableJ attribute

.\n **attribute** is a special tag that you can add 10 the top or ,ml' C# class. It's how C# stores m...ladala about your code, or information about how the code should 1K· usec1or treated. \\'!t,," you add `[Serializabl el` to the lop of a class just **above** the class d..daral;on. you're telling C# that your class is ,,,fe for serialization. And you only '''t it with classes Ihat include fields Ihat are either value types (likt· an `int` ostring. " 1' enum) of other serializable classes. If ' 'ou1dont add the attribute to Ih" class you want (0 serialize, or if you include a field with a type that ;,n'(serializable, then your program will have an error "h,-" vou try (0 n", it ***See jur yourself...***

Do th is

1 **Create a class and serialize it**

Remembc·r lit,· Guy Class fro>l" Chapt..r 3? 1," 1" serialize Joe so we can keep a file Ihal knlows how "'Ud, ""oney he' go ill his 1'ockets even after you close yo"'r program.

```
[Serializable]
public class GUy
```

Here's code In serialize il In a file called "Guy_file.dat' - add a "Save Joe" billion and a "Load Joe·' hUlloll I▷ the Form

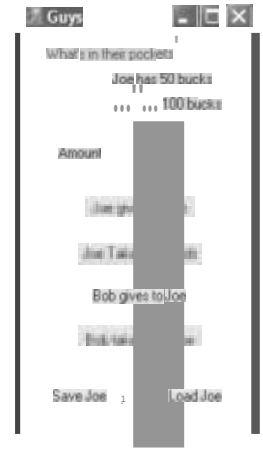
```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
```

You'll eeed these two `using` li..es. The first one is for the file a..d stream methods, and the second is for serialization.

```
private void eeve.rcejntceroejecc serlder, EventArgs e)
{
  using (Stream output = File.Create("Guy_File.dat")) {
    BinaryFormatter formatter = new BinaryFormatter(I ;
    formatter.Serialize(output, Joe);

private void loadJoe Cl1ck(obJect sender, EventArgs e}

  using (Stream input = File.OpenRead("Guy_File.dat"))
    BinaryFormatter bf - new Binaryrormatter();
    Joe = (Guy)bf,Deserialize(input);
  };
  CheckGuys();
```



o Run **the program and** play around **with** it .

If Joe had two IUlllldn'd dollars saved up from his transactions with Bob during your time running the program, il would be a pain n lose all Ihal money just because you need..d 10 ..xit. Now \our proarmu can save Joe oHl 1,. a file and restor.. him whenever yo"' want.

Let's serialize and deserialize a deck of cards

Take a deck of cards, and write it out to a file. C# makes serializing objects really easy. All you need to do is create a stream and write all your objects.



- **Create a new project and add the Deck and Card classes**

  Right-click on the project in the Solution Explorer and choose "Add/Existing Item", and add the Card and Deck classes you created in Chapter 1. You'll also need to add the two card comparer classes, since Deck uses them. The IDE will copy all of the files into the new project - make sure you change the namespace line at the top of each class file to match the new project's namespace.

- **Mark all of the classes serializable**

  Add the [Serializable] attribute to all of the classes you added to the project.

If you don't do this, C# won't let you serialize the classes to a file.

- **Add a couple of useful methods to the form**

  The RandomDeck method creates a random deck of cards, and the PrintCards method deals all of the cards and prints the title to the console.

```
Random random = new Random();
private Deck RandomDeck(int Number) {
    Deck myDeck = new Deck(new Card[] {});
    for (int i = 0; i < Number; i++)

        myDeck.AddNew Card(
            (Card.Suits)random.Next(4),
            (Card.Values)random.Next(1, 14)));

    return myDeck;
```

This creates an empty deck and then adds some random cards to it using the Card class from the last chapter.

```
private void DealCards(Deck DeckToDeal, string Title) {
    Console.WriteLine(Title);
    while (DeckToDeal.Count > 0)

        Card nextCard = DeckToDeal.Deal();
        Console.WriteLine(nextCard.Name);

    Console.WriteLine("----------------------");
```

The DealCards() method deals each of the cards off of the deck and prints it to the console.

Okay, prep work's done.. now serialize that deck

Start by adding buttons U, serialize a ,,,,,,Iolll deck to a til" and read it back. Check the eo"<,,I,- output I" make sure the ,],:c-k you wrote out is the same as the ,k,,'k you read.

```
private void button1_Click(object sender, EventArgs el {
    Deck deckToWrite - RandomDeck(5),
    using (Stream output - File.Create("Deck1.dat")) {
      BinaryFormatter bf - new BinaryFormatter()
      bf.Serializeloutput, deckToWrite);
    I
    oearcarcstoeckrowrjce, "What r just wrote to the file");
```

The BinaryFormatter object takes any object ..arked with the Serializable att..ibute—in this case a Deck object—and writes it out to a stream us:ng its Serialize() method.

```
private void button2_Click(object sender, EventArgs el {
    using (Stream input - File.OpenRead("Deck1.dat")) {
      BinaryFormatter bf = new BinaryFormatter();
      Deck deckFromFile = (Deck)bf.Deserialize(input);
    |
    DealCards(deckFromFile, '·What | read from the file");
```

The BinaryFormatter's eserialize() ...ethod returns an Object, which i. just the general type that every C# object inherits from—which i. why we need to cast it to a Deck object

Now serialize a bunch of decks to the same file

Once you 01"·"" str..am, you can write as llllKh as vuu want to it. You can serialize as Illany objects as you need i"l" the same file. So now add two mor,' buttons to write out a raJ"l"", number "l' decks to lh,' lii,', Check the output I" ",ak,· sure everything looks goc.d.

```
private void butron3 Chck(object sender, EventArgs el
    using {Stream output - File.Create("Deck1.dat"))
      BinaryFormatter bf _ new BinaryFormatter();
      for (int i - I; i <_ 5; i++) |
        Deck deckToWrite = RandomDeck|random,Next(1,101J;
        bf.Serialize{output, deckToWriteJ;
        teeicarcetoecx'rcwrrte, "Deck #" + i + " written");
    )
```

You can serialize one object after another to the same stream 1

Notice how the li"e that reads a single deck from the file uses (Deck) to cast the output of Deserialize() to a Deck. That's because Deserialize() returns an object, but deesn't necessarily know what type of object.

```
private void button4 Click(object sender, EventArgs e) {
    using (Stream input = File.OpenRead l"Deckl.dat"») i
      BinaryFormatter bf - new BinaryFormatter();
      for (int i - i; i <- 5; i++) |
          Deck deckToRead = (Deck)bf.Deserialize(input);
          DealCards ldeckToRead, "Deck #" + i + " read");
```

As long as you cast the objects r- read off the stream to the right type, there's no li..it to the numbe- of objects you can serialize

To.ke 0. look ct the file you wrote

Open "p Deckl.dat in Notepad. It may nol be quite something you'd read 01\lhe beach, but it's got all the inform.nion It> restore your whole deck "l' cards.



Wait a minute, I'm not sure I like all th is writing objects out to some weird fi le t hat looks like garbage when I open it up, When I wrote the deck of cards as strings, I could open up t he output in Notepad and see every t hing in it. Isn't C# supposed t o make it easy for me t o understand every t hing I'm doing?

When you serialize objects out to a file, they're written In a binary format.

But th;'1 tI,"-·,,,'1 mean it's indecipherable—just compact. That's why you can recognize the strings when you "1'"11 up a file with serialized objects in il: that's the most compact way C# ..an write strings to a file as strings. Bill writing "ut a UUll1I,,'I' as a string would he really wasteful. Anv int can be ,t"rc·d in four bytes. S" il would be ,,,Id if C# 'Io,.'d, say, Ih,' number 49,369,144 as an 8-character string that ""11 could read - 111 characters if ,'ou, illd ud,' commas. That "ould be a waste of space!

Behind the Scenes

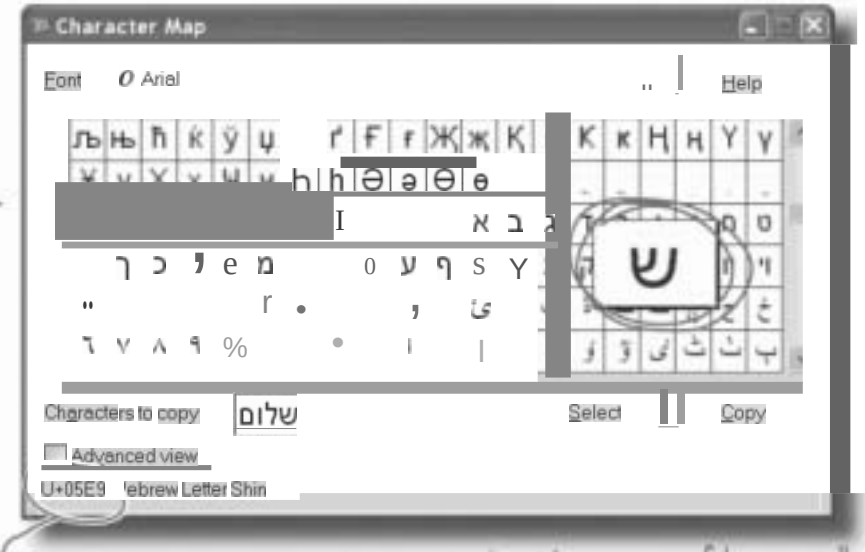
.NET uses Unicode to encode a char or string into bytes, Luckily. Windows has a useful little tool to help us figure out how unicode works, Open up the Character Map (it's in the Start menu under Accessones, or do Start I Run and type 'chermep exe').

When you look at all the letters and symbols that 'll"· ll"'d ill languages all around the world, you realize jU'1 ho" many dill,'n'nl *things* need 1" be written (" a li i., jU'1 I" ,("n' text. That's why .NET ..m·od.... all of its strings and characters ill " fi,rmal called Unicode. Em',ding jU'1 means taking the logical data (like the 1'll,'r H ;and turning il inlO bytes (the number 72). It needs 1" do> that because letters, 11"'nl"rs, enums and other data all end up in byte ou disk or in memorv. ,\ nrl that's why Character Map s useful—it show you how letters are encoded int» numhers

Select the Arial font and scroll down until you reach the Hebrew letters. Find the letter Shin and click on it.

As soon as you click on the letter, its Unicode number shows up in the status bar. The Hebrew letter Shin is number 05E9. That's a hexadecimal number—"hex" for short.

You can convert it to decimal using the Windows calculator: open it up, put it in Scientific mode, click the "Hex" radio button, enter "05E9", and then click "Dec"—it's 1,513.



Unicode is an industry standard developed by a non-profit group called the Unicode Consortium, and it works across programs and different computer platforms. Take a minute and look at their website: http://www.unicode.org/

.NET converts text to Unicode automatically

The two C# ,-ah,,' type f(r storing text - `string` and `char` - keep their data in memory as Unicode. When that data's written "Ill as bytes to a file, each of those Unicode numbers i,,, ritten "111 10 the Iike. So start a new project and drag three buttons 0111 0 a form, 'lIId we'nuse the Fii,'.\VriteAllBytes() and ReadAllBytes() ",,'!h,ods to get a sense or exactly how Unicode data i, written 0111 to a IiI,'.



1 Write a normal string out to a file and read it back

Use the same WriteAllText() method that you used in the text editor to have the li1"ll'"II"n write Ih,' string "E",eka!" out to a file called "eureka.txt". T h,'" create a new byte array called eurekaBytes, read Ihe IiI" into ii, and Ih"ll prim "Ill all or II", byte read:

```
File.WriteAllText("eureka_txt", "Eureka!");
byte[] eurekaBytes = File.ReadAllBytes("eureka.txt");
foreach (byte b in eurekaBytes)
    Console.Write("{0} ", b);
Console.WriteLine();
```

The ReadAllBytes() method returns a reference to a new array of bytes that contains all of the bytes that were read in from the file

You'll see Ih,'", bytes written 10 the "1111"": 69 117 114 101 107 97 33. Now open up the file in .he Simple Text Editor thar ,"u "Tol" earlier in the chapter. It says "Eureka!"

2 Make the second button display the bytes as hex numbers

It's "01 j"'l Character Map that shows uIIt"I,,'r, in hex. .\i""""t anything you r"ad that has to do with encoding data will , h,,,,' Ihal data ill hex. so iI" useful 10 k'lOW how to work wilh it. Make the code for seco,,d hUllon', event handler ill "o'l{' program identical to (he first on '-X""pl change Ihe Console.Write() line '0 it looks lik,' this in'I, ·,,,I:

```
Console.Write("{0:x2} ", bj;
```

Hex uses the bers 0 th gh 9 and letters A thr gh F to represent nu..bers in base 16, so 6B is equal to 107.

That tells WriteO 10 prilll paramete,. #0 (the first Oll(' after Ihe string to priul) as a two-character hex code. So il write Ih,' same seven byte iu hex instead of decimal: 45 75 72 65 6b 61 21

3 Make the third button write out Hebrew letters

(;,. back It, Character Map and d01l1)le-click " n the Shin character (" 1' click the Select hlltl ''11). II'Uadd il I" th,' "Character to copy" bc.x. 'I'hen d" the same for Ih,' rest ,.r the letters in "Shalom": Lamed (U+05DC), Vav (U+05D5I. ,,,,d Final Mem .t :+05DD). Now add the code for the thinl button's ,"\{"]11 handler. 1t'll 1,",k exactly like button 2, except for Oll(change. Click the "Copy"I","oll ill Character Map, and then paste the letters over "Eureka!" sc it looks lik,' this:

```
File,WriteAllText("eureka txt", "ם1ל1ש");
```

Did you notice that the IDE pasted the letters ill **backwards**? That's because it knows that Hebrew is read right-to-left, ,,, any time il encounters Hebrew l'ni('od,' 1,'lh'r" it displays them right-to-left. PilI your "111"'1' in the- middle or the letters— the left and right arrow keys reversed! That make, il" i" 1easier if you need t" t,p" in Hebrew. Now run the code, and I,e,k closely at theoutput: *tt* to' e9 05 dc 05 d5 05 dd 05. T h,' first two character are "FF FE", which i, the Unicode way or saying that w,..". going to have a string of Iwo-byte characters. T I,,' 1est or Ihe bytes are II", Hebrew letters - but they're reversed, so U+05E9 appears as .. 9 05. Now open the file up ill your Simpl,' Text Editor it I,,oks right!

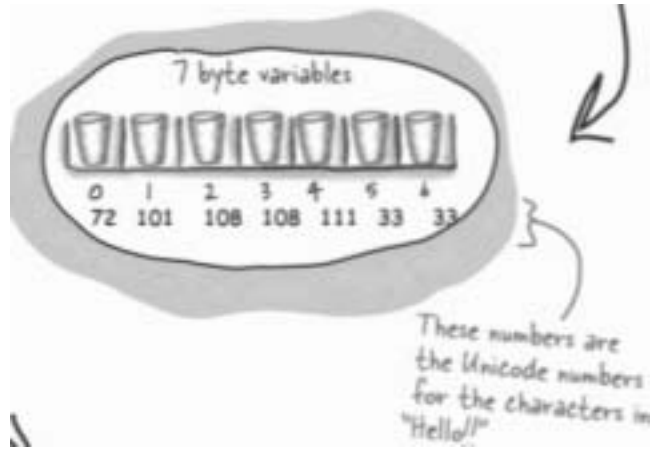
C# can use byte arrays to move data around

Since all your data—"", I, "1", "code1;" bytes, then it makes sense to think of a file as "a big byte array". And you already know how to read and write byte arrays.

Here's the code to create a byte array, open an input stream, and read data into bytes 0 through 6 of the array.

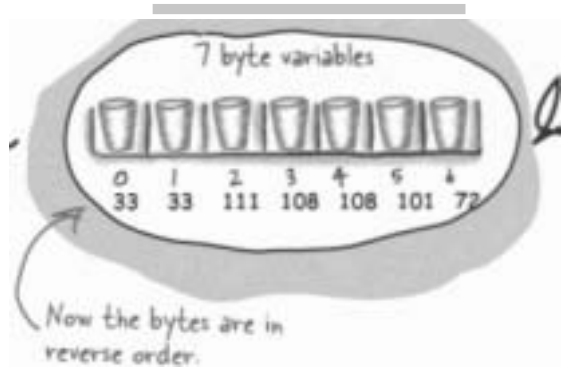



```
byte[] greeting;
greeting = File.ReadAllBytes(filename);
```

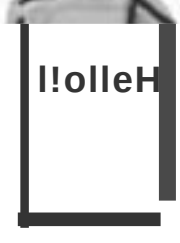


This is a static method for Arrays that reverses the order of the bytes. We're just using it to show that the changes you make to the byte array get written out to the file exactly.

```
Array.Reverse(greeting);
File.WriteAllBytes(filename, greeting);
```



When the program writes the byte array out to a file, the text is in reverse order too.



Use a BinaryWriter to write biliary data

y,,,, ***could*** encode all of your strings, chars, ints, and floats illh' byte arrays before writing them "Ill to files, but that would ge pretty ("Iiou'_That's why .NET gives you a very useful clas called **BinaryWr**it.... th,u autOlnatically ..n**cod**..**s your** data and write ill" a lil". All you need to do is create a FileStream and pass it int(the BinaryWrite 's constructor. Th"" you can call its "",thod, I" write out your data. So add another h",,,,,, ,o your program, and we'll show you how to use BinaryWriter().



1 Start by setting "I' ,om,' data to write to a lil".

```
int intValue = 48769414;
string stringValue - "Hello!":
byte!] byteArray - ( 47, 129, a, 116 );
float floatValue • 491.695F;
char charValue = 'E';
```

If you use File.Create(), it'll start a new file—if there's one there already, it'll blow it away and start a brand new one. There's also the File.OpenWrite() method, which opens the existing one and starts overwriting it from the beginning.

2 To use a BinaryWriter, first you need to open a new stream with File.Create():

```
using (FileStream output = File.creet.e("binarydata.dat")) {
  BinaryWriter writer · new BinaryWriter(output);
```

3 Now just ,'all its Write() ,,"',hod. Ead , time you do, it adds new byte ,,,,to the end "I' the file ,h", contain ,Il' encoded version or whatever data you passed it as a parameter.

```
writer.Write(intVallle);
writer.Write(stringValue).
writer.lll'ite(byteArray);
writel',writerrloatvarue};
vrIter•W'ite(cnarvatue};
```

Each Write() state..ent encodes one value into bytes, and then sends those bytes to the FileStream object. You can pass it any value type, and it'll encode it automatically

The FileStrea.. writes the bytes to the end of the file

Sharpen your penCil

4 Now use the same code you used before to read in the file you just wrote.

```
byte[] dataWritten = File.ReadAllBytes("binarydata.dat");
foredch (byte b in dataWritten)
    Console.Write("(O:x21 ", b);
Console,WriteLine(" - {al bytes", dataWritten.Length);
```

Here's a hint: Strings can be different lengths, so the string has to start with a nu..ber to tell .NET how long it is. Also, you can look up the string and char Unicode values -S-e Character Map

Write down ,he output in the blanks below. Can you flgure nUl **what bytes corre**"pond '0 each of the five Wi'i to' () statements? Mark each group or bytes " ith the name of ,he variable.

J

bytes

float and int values take up four bytes when you write them to a file. If you'd used long or double, then they'd take up eight bytes each.

Sharpen your pencil Solution

86 29 e8 02 t» 48 65 6c 6c 6f 21 2f 81 00 74 f6 d8 f5 43 45 _ 20 bytes

L- intValue — stringValue — byteArray — floatValue — charValue

The first byte in the string is 6—that's th length of the string You can use Character Map to look up each of the characters 'n "Hello!"—it starts with U+0048 and ends with U+0021.

If r- use the Windows calculator to convert these bytes from hex to decimal, you can see that these are the .....bers in byteArray.

char holds a Unicode character, and 'E' only take on byte—it's encoded as U+0045

Use **BinaryReader** to read the data baek in

The BinaryReader class works just like BinaryWriter. You create " stream, attach the BinaryReader object ,,, it. and then call il' methods. But the reader do..sn'. know **what** data's in **the file**! And il has Ilo way of knowing. Your float valu of 491.695F was encoded '" dB *i* 5 43 45. 11,,1those same byte are a perfectly valid illl 1,1-1-0.185,334. So you'll need In tell the BinaryReader "x,r<:tly what type 1" read from the Iii,'. Add 0"" more button 10your form, and have it read the data you just wrote.

Don't take our word for it. Replace the line that reads the float with a call to ReadInt32(). (You'll need to change the type of floatRead to int.) Then you can see for yourself what it read. from the file

- Start out by setting HI the FileStream and BinaryReader objects:

```
using (FileStream input - File.OpenRead("binarydata.dat"))
   BinaryReader reader - new BinaryReader(input);
```

- You tell BinaryReader wh.u type or data 10read by ,-alling its differeul methods.

```
int intRead - reader.ReactInt32();
string stringRead - reader.ReadString();
byte II byteArrayRead _ reader.ReadBytes 141;
float floatRead = reader.R..adSingle();
char charRead - reader.ReadChar();
```

Each value type has its ow·· ··ethod i, BinaryRea der() that return th data .n the correct type. Most don't need any r ete s, but ReadBytes() takes one P eter that tells BinaryReader ho '/ bytes to read.

- You tell BinarvReader what type or data 10n'ad by calling its different methods.

```
Console.Write("int: {0} string: {l) bytes, ", int.Reed, stringRead);
foreach Ibyte b in bytellrrayRead)
    Console.Write("(OI ", bl;
Console.Write(" float: (0) char: {l) , floatRead, charRead);
```

Here's Ih" »urputth.u gets priUl,·d lo the console:

```
into 48769414 string: Hello! bytes: 47 1290 116 float: 491.695 char: E
```

You can read aad write serialized files manually, too

Serialized files d"n"! look s« pretty whe-n you open them up in Notepad. Y,," "ll lind all the tiles you write in your project's "bin/Debug' folder— let's take a minute and get more acquainted '" the inner workings "r a serialized file.



O **Serialize** two *Card* **objects** to **different files**

Use the serializatiOll code you've already written to serialize the Thr..e " f Clubs to `cerdl•dat` and **Six** of H..**arts** to `card2 .dat`.. Check t« make ,,,re that both files were written "UI and are now Ill" folder, and th.u they both have Ihe same file ,;"," Then "I','n ""e "r them ill Notepad:

There are some words in the file (like "Chapter9", which was the namespace we used), but it's mostly unreadable.



O Write **a** loop to **compare the** two **binary** fi les

We IIsed the ReadByte() method to read the next byte from a stream it n'II11'n,'"1 i nt. that contains the value or that byte. \Ve also used the stream's `Length` field I" mak,' sure we read the' whole file.

```
byte[] firstFile = File.ReadAllBytes("card1.dat");
byte[] secondFile = File.ReadAllBytes("card2.dat");
for (int i = 0; i < firstFile.Length; i++)
    if (firstFile[i] != secondFile[i])
        Console.WriteLine("Byte #{0}: {1} versus {2}",
            i, firstFile[i], secondFile[i]);
```

This loop examines the first byte from each of the files and compares them, then the second byte, then the third, etc When it finds a difference, it writes a line to the console

The two files are read into two different byte arrays so they can be compared byte by byte. Since the same class was serialized to two different files they'll be almost identical... but let's see just HOW identical they a-e



When you write to a file, you don't always start from a clean slate!

Be careful if you use File.OpenWrite(). It doesn't ***delete the file—it*** *just* starts *overwriting the data starting* at *the beginning." That's why we've been using File.Create()—it creates* a *new file.*

We're not done yet—flip the page!

Find where the files differ, and use that information to alter thelll

The loop you ju,t wrote l'iul"'il11' exactly where Ih,' two serialized Card files difb-r; Since the only difference between the two objects were their Sui1 and Value fields then that should lie the only difference in their files too So if we find the bytes th"l hold tlie ,uit and value. we ,houl<l I", able t" **chang**.. th..**m** *to* Dlake a n..w **card** with whatever ,,,it ,ll,,1 \'al",' wc want!

3 Teke **a** look et 'the ccnscle **output** to see how 'the two fues differ-

Th,' console should show that two bytes differ:

```
Byte #218: 1 versus 3
Byte #266: 3 versus 6
```

That ,hould make a lot or sense! G back to the `Card.Suits` ,'I1Ulll from the last chapter, ,u,d you'll find that value for Clubs is I and the value for Hearts is :\, so th,u', th"1ir't difference. .\nd the second difference—six versus three— i, pretty obviously the card's value. You might ,,',' di'[k-n-nt byte numbers, which i,n"t surprising: you might be using a different namespace, which would change the length "f lhe file.

Re...e...ber how the namespace was included as part of the serialized file? If your na...espace is different, then the byte numbers will be different too.

Hmm, if byte #218 in the serialized file represents the suit, then we should be able to change the suit of the card by reading that file in, changing that one byte, and writing it out again. (Remember, your own serialized file might store the suit at a different location.)

0 **Write** cede to **manually** ceecte **a new** file tho.t **contains King** of **Spades**

We'll take one of the arrays that we read, alt"f it to contain a new card, and write it back Cllt.

```
firstFil [227] = (byte)Card.Suits.Spades;
firstFil '275' - (byte)Ca rd.Values,King;
File.Delete("card3.dat");
File.WriteAllBytes("card3,dat", firstFile);
```

If you found differe..t byte numbers in step #3 substitute them in here.

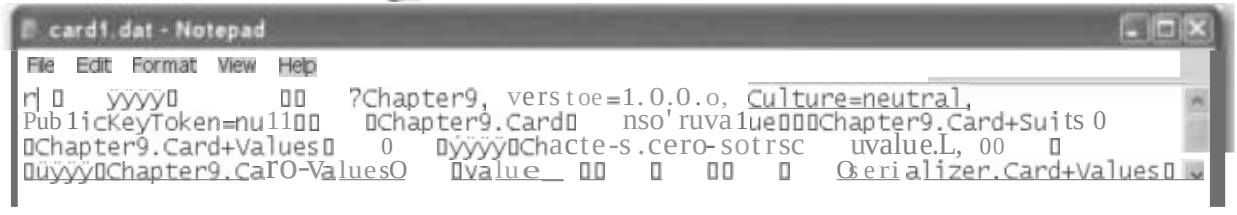
Now d..**s**..rialize the **card from** c a r d 3 , d **a t** and see if il·' tl", King " I' Spades!

Now that you know which bytes contai.. the suit and value, you can change just those bytes in the array before it gets writte.. out to card3.dat.

Working with binary files can be tricky

What do you do if you have a file and you aren't quite sure what's inside iL'. You d"n't kn,ow what apJ,Ii,-ati, .n created it. and you n,;ed lKJ kn,ow something about it -- but when you open it in Notepad, it looks like a bunch or garbage. \\'h,,, if vuu'dexhaucn-d all y('ur other "pti,>l". allli really needed In jus lo(,k inside? Looking at that picture, it's pretty d"ar Ihat Notepad just isn't II", right tool.

Here's the se..ialized ca..d, opened up i" Notepad That's not going to be useful at all

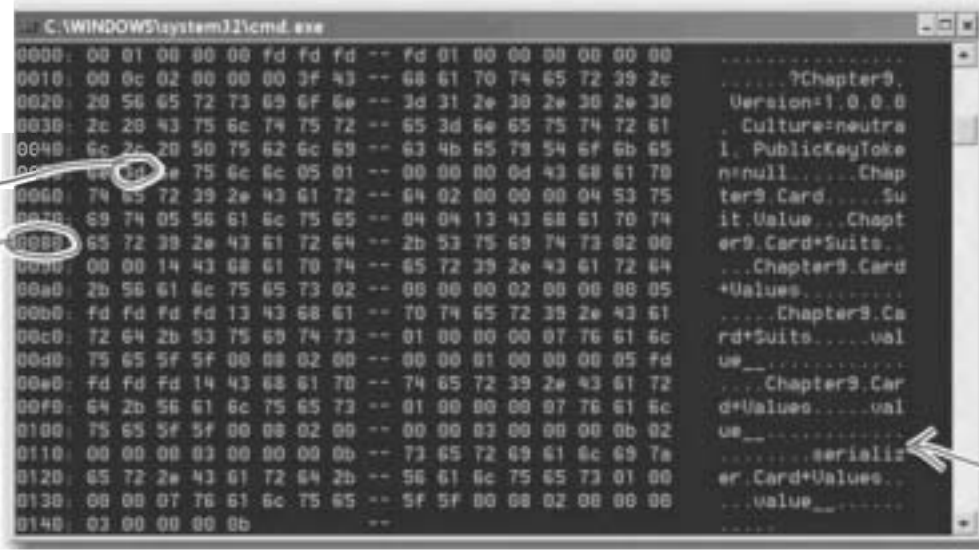


You can make out a few things—like the enum names ("Suit" and "Value"), and the name of the namespace we used ("Chapter9"). But that's not all that helpful.

There's another "pli,," - it's a f('nll,,1 called a "hex ,hllllp". "",I it's a pretty standard way 1" lo(,k a binary data. It's definitely !IlO'l' informative 111,1ll looking at the liI" ill Notepad. Hexadec;",al - 'll- "hex" is a convenient way to display bytes ill aliI". Every byte ,ak,s 2 characters to display ill hex, so you ,',ll' see a lot "I' data in a really "nail space, and a format that makes it easy t" spot patterns. .Also it's useful to display l.inarv data in rows ll[,« are 8, 16, "r 32 byte long because most binarv data tends 10 break down in chunks or 4, 8, 16, or 32 ..like all the types in C#. For example, an `int` takes up 4 bytes, and is 4 bytes],ng when serialized on disk. Here' \ı hat that same liI,· I, . >k, like as a hex dump, using tin,' " l' any 1ll1lll!>"r of free hex dllillp programs available for Windows:

You can '....ediately see the nu..eric value of each byte in the file

The number at the beginning of each line is the offset (or distance into the file) of the first byte in the line.



You still get to see the original text, but the garbage characters are stripped out.

Use file **streams** to build a hex **dumper**

A **hex dump** is a *hexadecimal* view of the contents of a file, and it's a really common way for programmers to take a deep look at a file's internal structure. Most operating systems ship with a built-in hex dump utility. Unfortunately, Windows doesn't. So let's build one!

How to make a **hex** dump

Start with some familiar text:

```
We the people of the United States, in order to form a more perfect union...
```

Here's what a hex dump of that text would look like:

Again, you can immediately see the numeric value of each byte in the file.

```
0000: 57 65 20 74 68 65 20 70   65 6f 70 6c 65 20 6f 66   We the people of
0010: 20 74 68 65 20 55 6e 69   74 65 64 20 53 74 61 74    the United Stat
0020: 65 73 2c 20 69 6e 20 6f   72 64 65 72 20 74 6f 20   es, in order to
0030: 66 6f 72 6d 20 61 20 6d   6f 72 65 20 70 65 72 66   form a more perf
0040: 65 63 74 20 75 6e 69 6f -- 6e 2e 2e 2e              ect union...
```

We'll add the number at the beginning of each line by using the offset of the first byte in the line

And we'll need to replace the garbage characters with periods.

Each of those numbers—57, 65, 6F—is the value of one byte in the file. The reason some of the "numbers" have letter values is that they're *hexadecimal* (or hex). That's just another way of writing a number. In decimal we use ten digits from 0 to 9, but hexadecimal uses sixteen digits from 0 to 9 plus the letters A through F.

Each time you run the hex dump represents a byte of the input that was used to generate it. In our dump, the first four characters are the offset in the file—the first line starts at character 0, the next at character 16 (or hex 10), then 32 (hex 20), and so on. (The other hex dump you'll see is slightly different, but this one will work for us.)

Working with **hex**

You can put hex numbers directly into your program code. Just add the characters 0x in front of the number:

```
int j = 0x20;
MessageBox.Show("The value is " + n.
```

When you use the + operator to concatenate a number into a string, it gets converted to decimal. You can use the static `String.Format()` method to convert your number to a hex-formatted string instead:

String.Format() uses parameters just like Console.WriteLine(), so you don't need to learn anything new to use it.

```
string h = String.Format("{0:x2}", j);
```

StreamReader aad StreallIWriter will do just fil1e

()UI' lu-x dumper will write its dump out to a file, and since it's just writing text a StreamWriter will do just fine. But we can also take advantage of the **ReadBl..ck()** 11I"lh, >(l in StreamReader. II reads a block "f characers illt" a char array—y ou specify the Illllnlll'r of characters you, want to read, and it'll either read that many characters or, if there are fewer lha" thai many left in 11", file, it'll read the rest of the Iil'. Since we're displaying 16 characters per li"" we'll read blocks of 16 characters.

So add one more butloll 10 your program—add lhi, lu-x dumper 1" il. Change the fir'l two lines so that they point I,. real files on your hard drive. Start "ilh a serialized Card file. Then see if you can modify itto use lhe ()p"" and Save As dialog boxes.

```
using (StreamReader reader    new StreamReader(@"c:\files\inputFile.txt"))
using (StreamWriter writel'   new StreamWriter(@"c:\files\outputFile.txt", false}}

  int position = 0;

  WmIe (!reader.EndOfStream) {
     char[] buffer - new char[16];
     int charactersRead .. reader.ReaclB10ck(buffer, 0, 16);
     writel'.Writet"10): -, String.Format("{0:x4}", position));
     position +_ charactersRead;

     for (int i = 0; i < 16; i++}
        if (i < cherect.eraaeed) (
           string hex - strtnq.Pcrmet rvjnrxzj", (byte)buffer[i]);
           writer.Write(hex + " ");
        else
           vriteriwritet"    ");
        if (i -- 7) { writer.Write("-- "); }
     if (buffer[i] < 32 || buffer[i] > 250}   buffer[i] .. '.   |

     string bufferContents = new string(buffer);
     writer.WriteLine("    " + butterContents.sy   trlng(O, eharaetersRead»);
```

A StreamReader's EndofStream field .eturns false if there are cha.acters still left to ..ead in the file.

This ReadBlock() call reads up to 16 cha ..ters into a char array.

The static String.Format ..ethod converts nu..bers to strings "{O:x4}" tells Format() to print the second parameter—in this case, position-as a 4-character hex .....ber

This loop goe through the cha.acters and prints each of the.. to a line in the output

Some characters with an value ..nder 32 don't print, so we'll replace all of the.. w th a period

You can conve..t a charIJ array to a string by passing it to the overloaded constructor for string

Every string has a substring ..ethod that returns a piece of the string fn this case, it retu the first charactersRead characters starting at the beginning (po ion 0) (Look bacl at the top of the loop to see where charactersRead is set—the ReadB10ck() method returns the number of cha.acters that it read into the array

there are no Dumb Questions

Q: **Why didn't I have to use the `Close()` method to close the file after I used `File.ReadAllText()` and `File.WriteAllText()`?**

A: The `File` class has several very useful static methods that automatically open up a file, read or write data, and then **close** it **automatically**. In addition to the `ReadAllText()` and `WriteAllText()` methods there are `ReadAllBytes()` and `WriteAllBytes()` that work with byte arrays, and `ReadAllLines()` and `WriteAllLines()`, which read and write string arrays, where each string in the array is a separate line in the file. All of these methods automatically open and close the streams, so you can do your whole file operation in a single statement.

Q: **If the `FileStream` has methods for reading and writing, why do I ever need to use `StreamReader` and `StreamWriter`?**

A: The `FileStream` class is really useful for reading and writing bytes to binary files. Its methods for reading and writing operate with bytes and byte arrays. But a lot of programs work exclusively with text files, like the first version of the Excuse Generator, which only wrote strings out to files. That's where the `StreamReader` and `StreamWriter` come in really handy. They have methods that are built specifically for reading and writing lines of text. Without them, if you wanted to read a line of text in from a file, you'd have to first read a byte array and then write a loop to search through that array for a linebreak—so it's easy to see how they make your life easier.

Q: **When should I use `File` and when should I use `FileInfo`?**

A: The main difference between the `File` and `FileInfo` classes is that the methods in `File` are static, so you don't need to create an instance of them. On the other hand, `FileInfo` requires that you instantiate it with a filename. In some cases that would be more cumbersome, like if you only need to perform a single file operation (like just deleting or moving one file). On the other hand, if you need to do many file operations to the same file, then it's more efficient to use `FileInfo`, because you only need to pass it the filename once. You should decide which one to use based on the particular situation you encounter. In other words, if you're doing one file operation, use `File`. If you're doing a lot of file operations in a row, use `FileInfo`.

Q: **Back up a minute. Why was "Eureka!" written out with one byte per character, but when I wrote out the Hebrew letters they took up two bytes? And what was that "FF FE" thing at the beginning of the bytes?**

A: What you're seeing is the difference between two **closely related** Unicode encodings. Plain English letters, numbers, normal punctuation marks, and some standard characters (like curly brackets, ampersands, and other things you see on your keyboard) all have very low Unicode numbers—between 0 and 127. (If you've used ASCII before, they're the same as the ASCII characters.) So a file only contains those Unicode characters with low numbers, it just prints out their bytes.

Things get a little more complicated when you add higher-numbered Unicode characters into the mix. One byte can only hold a number between 0 and 255. But two bytes in a row can store numbers between 0 and 65,536, which, in hex, is FFFF. The file needs to be able to tell whatever program opens it up that it's going to contain these higher-numbered characters, so it puts a special reserved byte sequence at the beginning of the file: "FF FE". That's called the "byte order mark". As soon as a program sees that, it knows that all of the characters are encoded with two bytes each. (So an E is encoded as 00 45—with leading zeroes.)

Q: **Why is it called a byte order mark?**

A: Remember how your bytes were reversed? Shin's Unicode value of U+05E9 was written to the file as E9 05. That's called "little endian". Go back to the code that wrote out those bytes and add a third parameter to WriteAllText() Encoding.BigEndianUnicode. That tells it to write the data out in "big endian", which doesn't flip the bytes around. You'll see the bytes come out as "05 E9" this time. You'll also see a different byte order mark: "FE FF". And your Simple Text Editor is smart enough to read both of them!

If you're writing a string that only has Unicode characters with low numbers, it writes one byte per character. But if it's got higher-numbered characters, they'll be written using two bytes each.



Change Brian's Excuse Generator so it uses binary files with serialized Excuse objects instead of text files.

1. **Make the Excuse class** ser-icllacble
Mark the Excuse class with the [Serializable] "Utihut" to mak,- it serializable. Also, you'll need to add the using hne:

```
using System.Runtime.SerLa1izat Ion.Formatters.Bfnary;
```

2. **Change the Excuse.Save() method** to **serialize the** excuse
When the Save() method writes a flk- oul to the folder, instead "I' using StreamWriter 10 writ(the file out, have· it op..n a file and serialize itself out. Y'll'll need 10 figure "111 how Ihe current class can deserialize itself.

Hint: What keyword can you use inside of a class that returns a reference to itself?

3. Change **the Excuse.OpenFile() method** to **deserialize** on **excuse**
You'll need 10 create a temporary Excus(object to deserialize li'"1111t,· file, and then copy its fields inlo lite "llITl-1tl class.

4. Now just **change the form** so it **uses** a **new file extension**
There's just one very small chang(you nee-c110 make to the form. Since we're "0 long(·r working with text files, we ,houldu't use lit.. .txt e-xt..,,,ill" anv mOI'-. Chang(the dialog boxes, default filenames and directory search code so tli.u they work with *.excuse files in't,>;ul



Wow, thot was **really easy!** All **the** code for **saving** and **opening excuses** was **inside** the Excuse cltlSS I **just** had **to change the class—I** barely had to touch the form at all. It's like the form doesn't even care how the class saves its data. It just passes in the filename and knows everything will get saved properly.

That's rlghtl Your code was very easy to change because the class was well encapsulated.

\\'Il,'n vou've got a class that hides its internal operations from the rest "l' the program and onlv exposes the behavior that needs to be exposed, it's called a **wen ..ncapsulat..d cia**ss. lll the Excuse Manager program, the form doesn'r have anv information ab(nu how excuses are saved to files. It jU'l passes a filename iulo the excus(class, and Ihl' class takes l"an' of Ill,' rest. That makes it very easy to lllak,- big change to how your class works with files The better you encapsulate your classes, the easier they are t" "lt,·l' later on.

Re..emb(- how encapsulation was one of the fou.. core OOP principles? Here's an example of how using those principles ...akes you.. programs better



Change Brian's Excuse Generator so it uses binary files with serialized Excuse objects instead of text files.

You only need to change these four lines in the form: two lines in the save button's Click event, and two in the open button's—they just change the dialogs to use the .excuse extension.

```
private void save_Click(object sender, EventArgs e) {
  // existing code
  saveFileDialog1.Filter = "Excuse files (*.excuse)|*.excuse|All files (*.*)|*.*";
  saveFileDialog1.FileName = description.Text + "...xcuse";
  // existing code

private void save_ctrcerccjecr sender, aveot.xros e)
  // existing code
  openFileDialogl.Filter ..
    "Excuse files (e.excusel 1.,excuselAII files (*.*)|*.*";
  openFileDlalogl.FileName .. description.Text +  ...xcuse";
  ff existing code
```

Standard save and open dialog boxes do the trick here

The only change to the for... is to have it change the file extension it passes to the Excuse class.

```
[Serializable]
pucjLc class Excuse {
    public string Description;
    public string Results;
    public DateTime LastUsed;
    pubLic string gxcusepatn.
    public Excuse() {
        ExcusePat.h • "";

    public Excuse(string excusePath)
        OpenFile(ExcusePathl,

    public Excuse(Random random, string folder) {
        stringl) fileNames - Directory.GetFiles(folder, "*.excuse");
        OpenFile(fileNameslrandom.Next(fileNames.Length) 1);

    private void OpenFile(string excusePath)
        this.ExcusePath = excusePath;
        BinaryFormatter formatter - new BinaryFormatter();
        Excuse tempExcuse;
        using (Stream input - File.OpenRead(excusePath)) {
            tempExcuse = (Excuse)formatter.Deserialize(input);

        Description = tempExcuse.Description;
        Results - tempExcuse,Results;
        LastUsed _ tempExcuse,LastUsed;

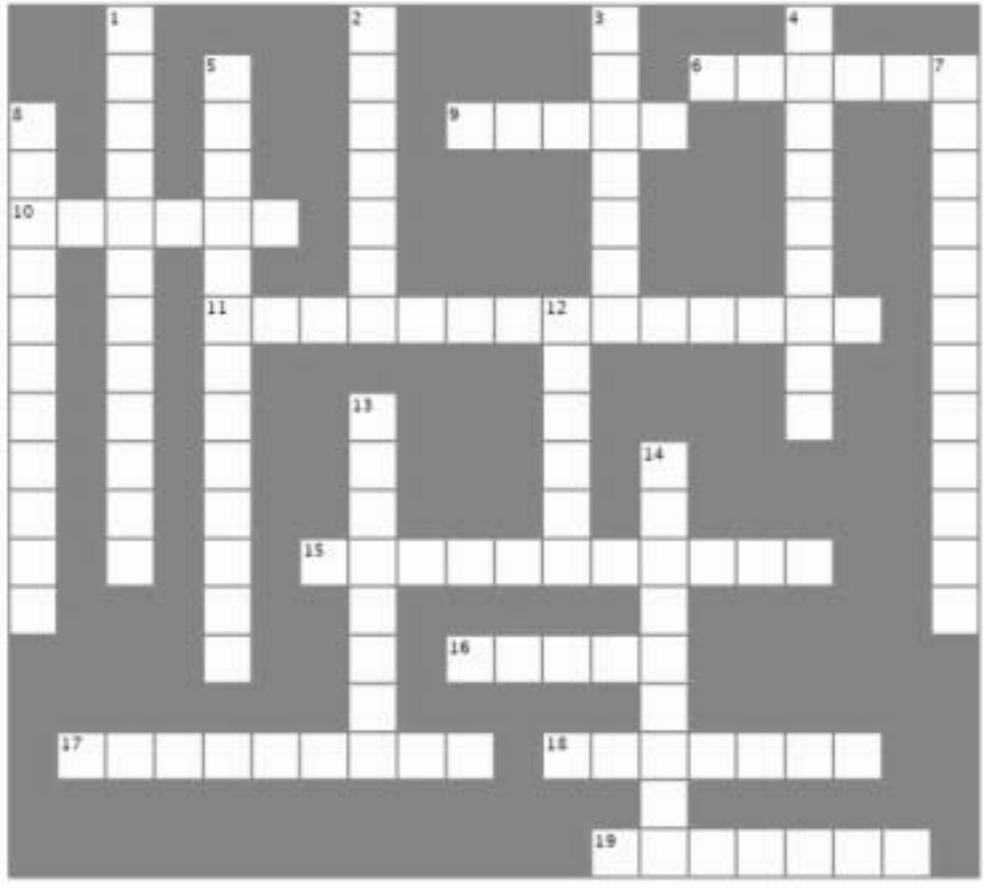
    public void Save(string fileName) {
        BinaryFormatter formatter . new BinaryFormatter();
        using (Stream output - File.OpenWrite(fileName)) {
            formatter.Serialize(output, this);
        }
```

The constructor for loading random excuses needs to look for the ".excuse" extension instead of "*.txt" files.

We pass in "this" because we want thi class to be serialized



Filecross



Across

6. The method in the File class that checks whether 01' not a specific file is on the drive

9, This statement indicates the end of a case inside a switch statement

10 The abstract class that FileStream inherits from

11 Anon-visual control that lets you pop up the standard Windows "Save As" dialog box

15 How you write numbers in base-16

16 If you ccet cen this method, your stream could be locked ccen so one- rreecos 01' programs cant open it

17, The StreamReader method that reads data into a char[] array

18. An encoding system Ihat assigns a uniue number to each character

19, Use this staierreeuo indicate which statements should be executed when the value being tested in a switch statement does IIOt match any of the cases

Down

1. This class has a method that writes any value type to a file

2 The static method in the Array class that turns an array backwards

3, The event handlel that gets run whenever someone modifies the data in an Input control

4 This cress has many static methods that let you manipulate folders

5, Using thIS OOP principle makes it a loteaser to maintain your code

7, If youdon't use this allnbute to indicate that a class can be w neoto a stream, BinaryFormatter will generate an error

8, This BinaryFormatter method reads an object from a stream

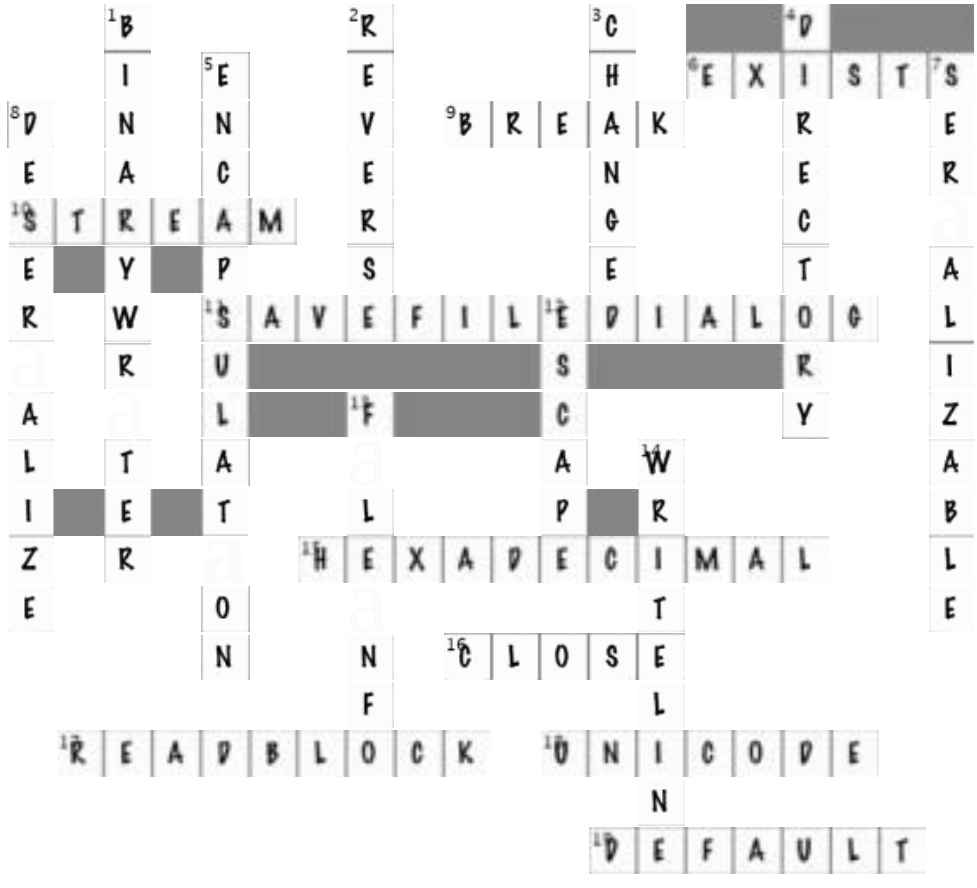
12 In and II' areexamples ofthis kind of sequence

ta. This class lets you perform atee cperatcns in Ihe FIle cass 101' a specific file

14, This method sends text to a stream followed by a line break



Filecross solution



Putting out fires gets old



Programmers aren't meant to be firefighters.

You've worked your tail off, waded through technical manuals and a few engaging Head First books, and you've reached the pinnacle of your profession: **master programmer.** But you're enn getting pages from work because your program crashes. or doesn't behave like it's supposed to, Nothing pulls you out of the programming groove like having to fix a strange bug,.,but with exception handling, you can write code to deal with problems that come up. Better yet you can even react to those problems. and keep things running

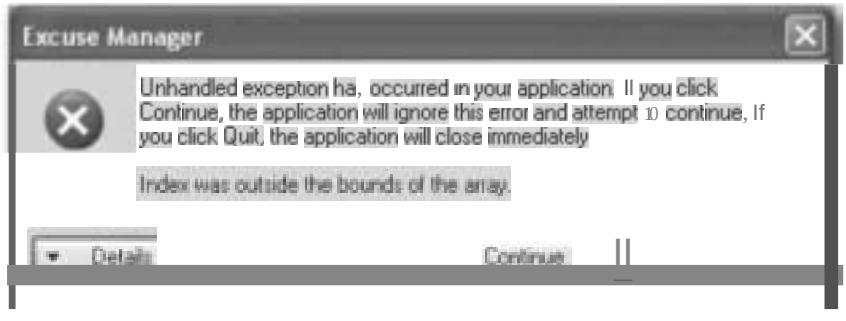
Brian Ileeds his excuses to be 1\I0bile

Brian recently got reassigned 10 the inn-rn.ujonal division. Now he flie all over Iht" world. But he still needs 10 k,"'p track 0I his excuses, so he installed the program you huill "n his lal'l0l'. and takes it with him evervwher,·.



But the program isn't working!

Brian clicks the "Random Excuse" button. and gets a pretty nasty looking error. Something about not finding his excuses. "'ha l gives?



Sharpen your pencil

Here's another example of some broken code. There are five different exceptions that this code throws, and the error messages are shown on the right. It's your job to match the line of code that has a problem with the exception that line generates. Read the exception messages for a good hint.

```
public static void BeeProcessor(J |
    object myBee = new HoneyBee(36.5, "Zippo");
    float howHuchHoney = (float)myBee;

    HoneyBee anotherBee = new HoneyBee(12.5, "Buzzy");
    double beeName = double.Parse(anotherBee.MyName);

    double totalHoney = 36.5 + 12.5;
    string beesWeCanfeed = "";
    for (int i = I, i < (int) total1Honey; i++) |
        beesWeCanFeed += i.ToString();
    |
    float f =
      float.Parse(beesWeCanFeed);

    int drones = 4
    int queens = 0;
    int dronesPerQueen = drones / queens;

    anotherBee = null;
    if (dronesPerQueen < 10) {
        anotherBee.DoMyJob n ;
    |
|
```

Calling double.Parse("32") will parse a sV-S and return a double value, like 32.

When you have a reference that doesn't point to any object, it gets a special value called null, Setting a reference to null, tells C# it doesn't point to anything

OverflowException was unhandled 1
Value was ee-e- too large or too small for a Single.

NullReferenceException was unhandled 2
Object reference not set to an instance of an object.

InvalidCastException was unhandled 3
Specified cesr is rot valid.

DivideByZ xception was unhandled 4
Attempted to divide by zero.
Troubleshooting tips:
Make sure the value of the denominator is not zero before performing a division operation.
Get general help for this exception.
Search for more Help Online...

FormatException was unhandled
Input string was not in a correct Iormat
Troubleshooting tips:
Make sure your method arguments are in the right format.
When converting a string to DateTime, parse the string to take ee date before putting een variable into ee peteree object.
Get general help for this exception.
Search for more Help Online.

Sharpen your pencil Solution

Your job was to match the line of code that has a problem with the exception that line generates.

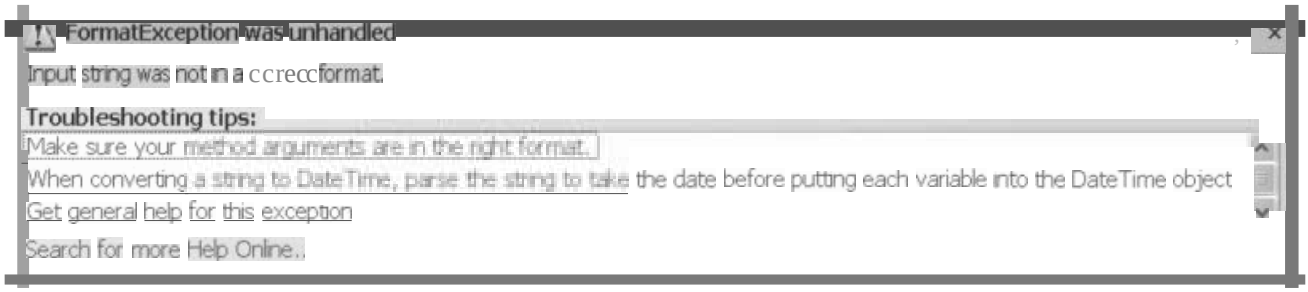
```
object myBee = new HoneyBee(3f.5, "Zippo");
float howMuchHoney = (float)myBee;
```

C# has no idea how to cast a HoneyBee object to a float, and trying to do it will cause an InvalidCastException.




```
HoneyBee anotherBee = new HoneyBee(12.5, "Buzzy");
double beeName = double.Parse(anotherBee.MyName);
```

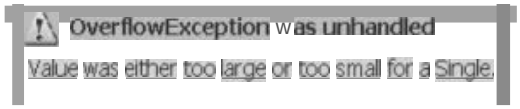
The Parse() method wants to give it a string in a certain format. "Buzzy" isn't a string it knows how to convert to a number. That's why it throws a FormatException.




```
double total1Honey = 36.5 + 12.5;
string beesWeCanFeed = "";
for (int i = 1; i < (int) totalHoney; i++) {
    beesWeCanFeed += i.ToString();
}

float f = float.Parse(beesWeCanFeed);
```

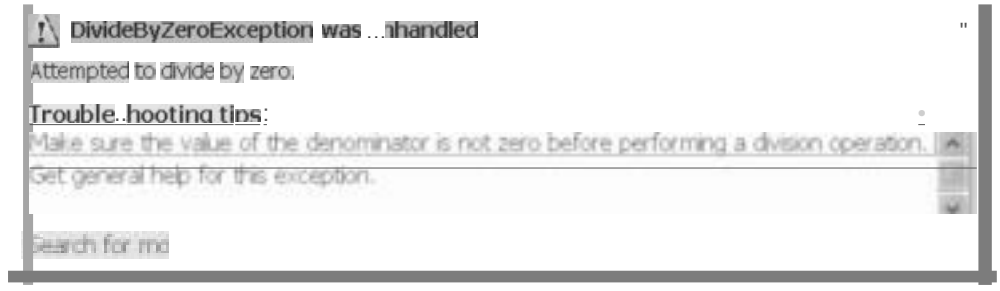
The for loop will create a string called beesWeCanFeed that contains a number with over 60 digits in it. There's no way a float can hold a number that big, and trying to cram it into a float will throw an OverflowException.



You'd never actually get all these exceptions in a row. The program would throw the first exception and then stop. You'd only get to the second exception if you fixed the first.

```
int drones = 4;
int queens = 0;
int dronesPerQueen = drones / queens;
```

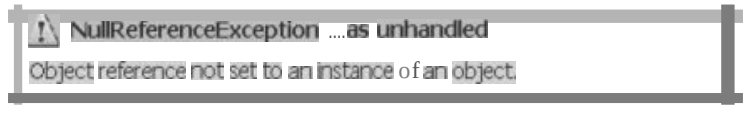
It's really easy to th.ow a DivideByZeroException. Just divide any .....ber by zero.



Dividing any number by zero always throws this kind of exception. Even if you don't know the value of queens, you can prevent it just by checking the value to make sure it's not zero **before** you divide it into drones.

```
anotherBee = null;
if (dronesPerQueen < 10) {
    anotherBee.DoMyJob();
```

Setting the anotherBee reference variable equal to null tells C# that it doesn't point to anything. So instead of pointing to an object, it points to nothing. Throwing a NullReferenceException is C#'s way of telling you that there's no object whose DoMyJob() method can be called.



That DivideByZero error didn't have to happen. You can see just by looking at the code that there's something wrong. The same goes for the other exceptions. These problems were preventable-and the more you know about exceptions, the better you'll be at keeping your code from crashing.

Whell your progral\l throws an exceptio•. .NET gellerates an Exception object.

You've been I,. ıking at .NET's \\ay or telling you something ""Ill wrong in your program: an ..**xc**..ption. In C#, when an excc·]"iotl occurs, an object is created to represent the problem. It's ,·all,',!. 110 surprise here, Exception.

For example, suppose you have an array with four items. Then, you try allli access the sixteenth item (index 15, since we're zero-based here):

```
int anArray = {3, 4, 1, 11};
int aValue = anArray[15];
```

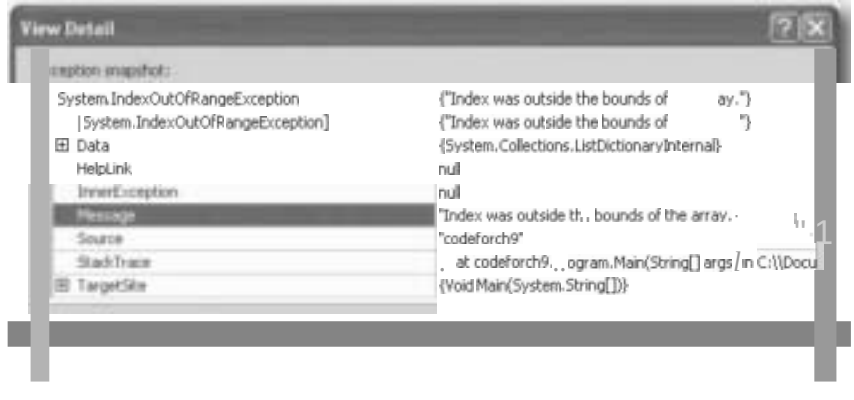
This code is obviously going t, cause problems.

cx-ccp-rion, noun.
a pe-rson or thing that is vxcludcd from a general statement or dot's not follow a rule. *While Jim usually hates peanut butter, he made (//1 exceptionfa Ken's peanut butter fudge.*

As soon as your progra·· ru''' into an unhandled exception, it generates an object with all the data it has about it.

Exception object

You can see this detail by clicking on the View Detail link in the unhandled exception window.



The exception object has a message that tells you what's wrong and a list of all of the calls that were made to the system's memory leading up to the event that caused the exception.

.NET goes 10 the trouble of creating an object because it wants 10 give vou all the information about what caused the exception. You may have code 10 fix. ',r you may jU'1 need I" make "'Illl" changes 10 h,»w "'I' h"mll,' a particular ,illla,i,>I, in yo,ur program.

In this case, un IndexOUWfRangeExcept1on indicates you have a bug: you·re trying !" access an index ill the array that's "\11 of range. You\ .,. 'II", got illf"m'illioll about exactly where in the code the problem occurred, making il easy I" track down Ill"]1",1,1"1ll (even if you've got Ih,»usands of lines of (·(>tl,·).

there are no Dumb Questions

Q: Why are there so many kinds of exceptions?

A: There are all sorts of ways that you can write code that C# simply doesn't know how to deal with. It would be difficult to troubleshoot your problems if your program simply gave a generic error message ("A problem occurred at line 37"). It's a lot easier to track down and fix problems in your code when you know specifically what kind of error occurred.

Q: So what *is* an exception, really?

A: It's an object that .NET creates when there's a problem (more about that in a minute)

Q: Wait, what? It's an *object?*

A: Yes, an exception is an object. The properties in the object tell you information about the exception. For example, it's got a Message property that has a useful string like "Specified cast was invalid" and "Value was either too large or too small for a Single", which is what it used to generate the exception window. The reason that .NET generates it is to give you as much information as it can about exactly what was going on when it executed the statement that threw the exception.

Q: Okay, I still don't get it. Sorry. Why are there so many different kinds of exceptions, again?

A: Because there are so many ways that your code can act in unexpected ways. There are a lot of situations that will cause your code to simply crash. It would be really hard to troubleshoot the problems if you didn't know why the crash happened. By throwing different kinds of exceptions under different circumstances, .NET is giving you a lot of really valuable information to help you track down and correct the problem.

Q: So exceptions are there to help me, not just cause a pain in my butt?

A: Yes! Exceptions are all about helping you expect the unexpected. A lot of people get frustrated when they see code throw an exception. But if you think about an exception as .NET's way of helping you track down and debug your program, it really helps out when you're trying to track down what's causing the code to bomb out.

Q: So when my code throws an exception, it's not necessarily because I did something wrong?

A: Exactly. Sometimes your data is different than you expected it to be—like you've got a method that's dealing with an array that's a lot longer or shorter than you anticipated when you first wrote it. And don't forget that human beings are using your program, and they almost always act in an unpredictable way. Exceptions are .NET's way to help you handle those unexpected situations so that your code still runs smoothly and doesn't simply crash or give a cryptic, useless error message.

Q: Once I knew what I was looking for, it was pretty clear that the code on the previous page was going to crash. Are all exceptions easy to spot?

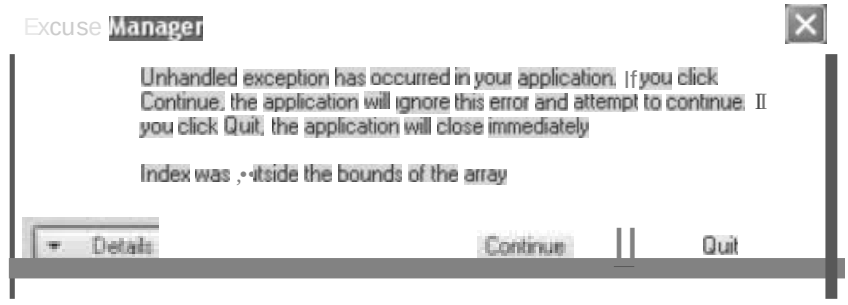
A: No. Unfortunately there are times when your code will have problems, and it'll be really hard to figure out what's causing them just by looking at it. That's why the IDE gives you a really useful tool called the debugger. It lets you pause your program and execute it statement by statement, inspecting the value of each individual variable and field as you go. That makes it a lot easier for you to figure out where your code is acting in a way that's different from how you expect it to act. That's when you have the best chance of finding and fixing the exceptions-, or, even better, preventing them in the first place.

Exceptions are all about helping you find and **fix** situations where your code **behaves** in **ways** you didn't **expect.**

Brian's code did something uaexpeeted

When Brian wrote his excuse manager, he never expected the uv-r to try 10 pull a random excuse out of an empty directory.

O Tl u- problem hal'p,-",-d when Brian pointed his Excuse Manager program at an empty folder on his laptop and clicked the Random hUll' ,n, Let's take a I,ю,k at ir and see if we can figure out what wenl wrong. Here's the unhandled exception window that popped llp when he ran the program outside the IDE:



O Okay, that's a good starting point. It's telling us tll"11he index was outside the bounds "l' Ih,' array, right? So let's look for "'I array in the code for the Random Excuse button's event handler:

```
pr1vate v01d RandomExcuseButton_Cl ick(object sender, EventArgs e)
    if lCheckChanged() == true)
        CurrentExcuse = new Excuse(random, folder);
        UpdateForm(falsej;
```

3 Hmm. no arrays in there. But il create a new Excuse object using one of the overloaded constructors. Maybe there's an array in the constructor code:

```
public Excuse(Random random, string Folder)
    stringI1 fileNames _ Directory.Getfiles(folder, ....excuse");
    Openfile(fileNameslrandom.Next(fileNames.Length)]);
```



Bingo! There's the array We ..ust be trying to use an index that's past the end of the array

O

It 111m, out Iha! `Directory.GetFiles()` returns a ll empty array when you poim it at a directory with n" files in iI, Hey, we can test for Ilral! .\ ll we need 10 do is add a check 10 **mak**.. s Ur .. th.. dir..**ctory's** nut ..rnpty before we open a file, ,ll,,1 the nasty unhandled exception window will be replacec1 with an iul; >rmalive messagebox.

```
private void RandomExcuseButton_Click(object sender, EventArgs e)
    string[] fileNames = Directory.GetFiles(Folder, "*.excuse");
    if (fileNames.Length -- 0) {
        MessageBox.Show("Please specify a folder with excuse files in it",
                        "No excuse files found");
      else |
        if (CheckChanged() == true)
            CurrentExcuse - new Excuse(random, Folder);
            UpdateForm(falsel ;
```

By checking for excuse files in the folder **before** we create the Excuse object, we can prevent the exception from being thrown—and pop up a helpful message box, too.

Oh, I get it. Exceptions aren't always bad. Sometimes they identify bugs, but a lot *of* the time they're just telling me that something happened that was different from what I expected.



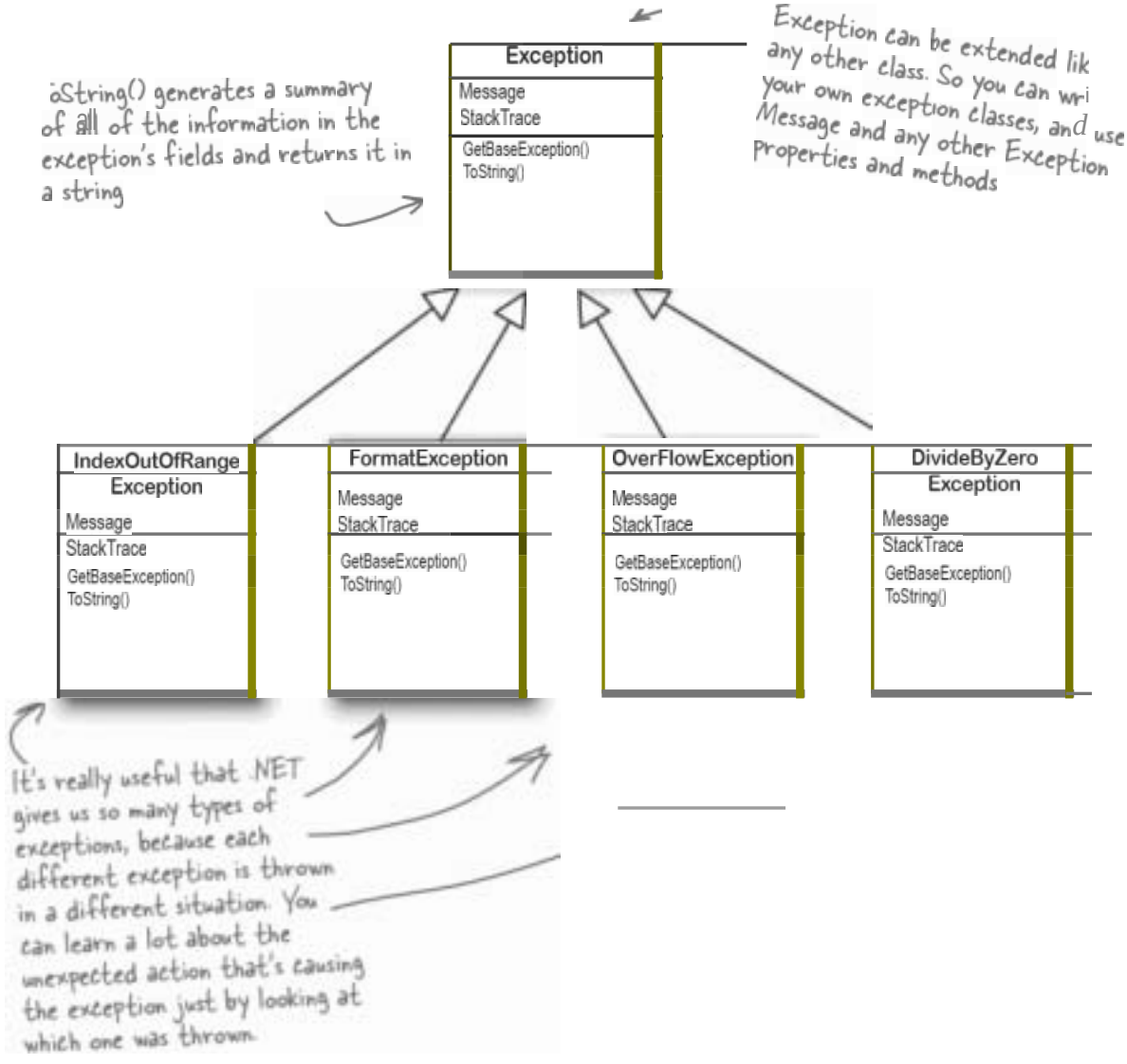
That's right. Exceptions are a really useful tool that you **can** use to find places where your **code acts** in ways you don't expect•

.\ 1001 of programmers get frustrated the first time they see all exception. 11m exceptions are really useful, and you t' lll use them to vour advantage When you see an exception, it's giving you a 1„1 of clues 10 Iwll' you figure 0 l11„1'"11 y(>Ur code i, reacting It>a situati"" that you didn't anticipate. Aud that's good for you il lets you kuo\\ about a new st-t'ua, io lhal your program has «, handle, and il give you an " 1'l>(>r11l11;ly I" do **something** a bout it

All exceptiol1 objects **inherit from** Exception

.NET has 101, of different exceptions il may need 10 report. Since manv of these have a 101 of similar features, inheritance come intu play. .NET defin..s a base class, called Exception, that all specific exceptions type inherit from.

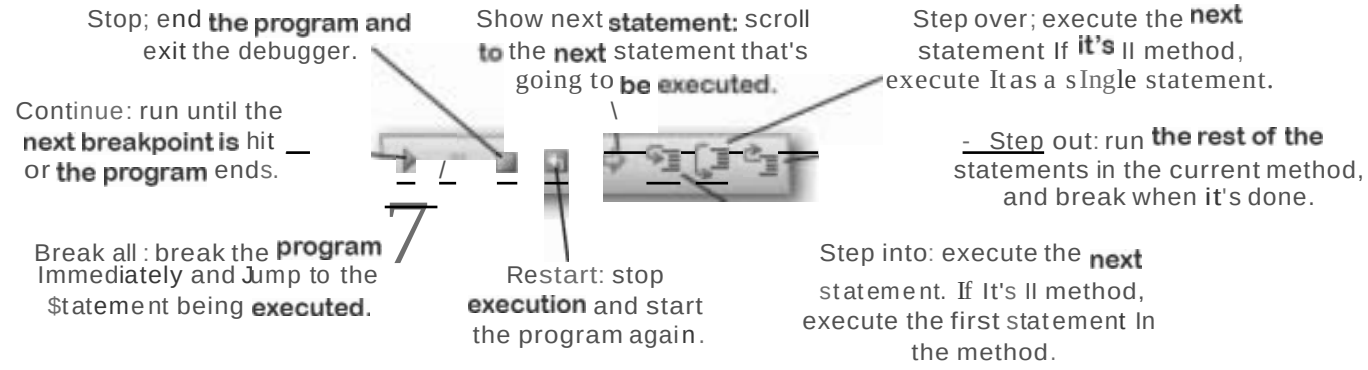
Th,- Exception class has a couple o[u,,'li,1members. The `Message` propertie store an easy-to-read messag(about what went wrong. .\"d `St.ack'Trace` tells you what was going oil in memory when the exception occurred, and what led "l' to the exception. (T h,·r,· are others, too, but we'll use those first.)



The debugger helps you track dowll alld prevent exceptiolls ill your code

B"li›re y(>I, can add ('X"'"!,1;',' handling to your 1"'›gram, y(>ll need t(>know which statements in your program are throwing the exception. That's where the d..**bugger** that's huih iuto the IDE can h., really helpful. When you run the debugger, rh,' IDE pops \ll' a toolbar with some really useful buttons. Take a minule and hover your mouse cursor over each of them to see its name and ,lInn,'ul key. (Those 'JU',-I"lI(keys come ill really handy.)

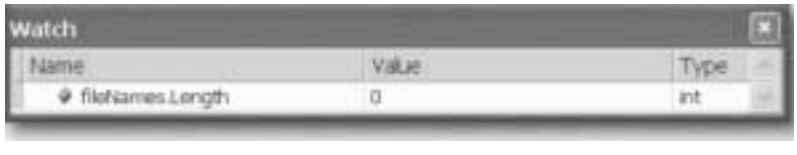
The Debug toolbar only shows up when you're debugging your program in the IDE. So you'll have to run a program in order to hover over the toolbar icons.



Uebugging .,ean, running your code line by line to see what happen,

Whenever you rim vour program in,id,' the IDE, vou can alwavs pause it a1any time by hitting the **Break All** button in the toolbar (or choosing the command li,›m the Debug menu). Thi, causes y(''1 program to **stop** in **its tracks** and ,I"›w you the line "l' code lhal it's ahOlll to fUll. II turns lhal lin,>"l' code yellow II' show you that it's the one that'll run next. [f you press Continue, tht-n your program will keep runuinu as if vou'd never stopped it. But vou can also ""I' throuuh your code, "hidl means r-xrcuting the current li"" and going 1" the next one. If the next line is a method, lh"11 you can **step** ;nlo the """lh"d, which causes the debugger to jUlIlpl" the lir'l line of the method a",1 highlight it. ()r you can step over the """lh,>(I, which executes the wh,,],' thing. If you're inside a method, you can **step** o.n of it. which causes the debugger to "X"CU1" the rest of 1he statements in the method and break all],"1ir'll in,' after i(returns f",m the llWlhud.

You can also inspect ,111<1 change variables and fields ill your "0<1,· using d,,' **Watch window**,.l"'! right.click 0" a variable ill the '"(Kl,- and select "Add \Vatch", and il'll apl',-ar jnrhe Watch window or you can type i, directly inl" the Watch window. Then its value will be displayed. 1l' it's '"1 object, you can drill down inl" its fields.



When you **break** inside the del,ugger, the IDE stops your program **and** displays the next line 01 **code** that **it's about** to run I,ighlighted in yellow. Then you can move **forward** line I,y line until you lind your **problem.**

Use the IDE's debugger to ferret out exactly what wellt wrollg in the excuse manager

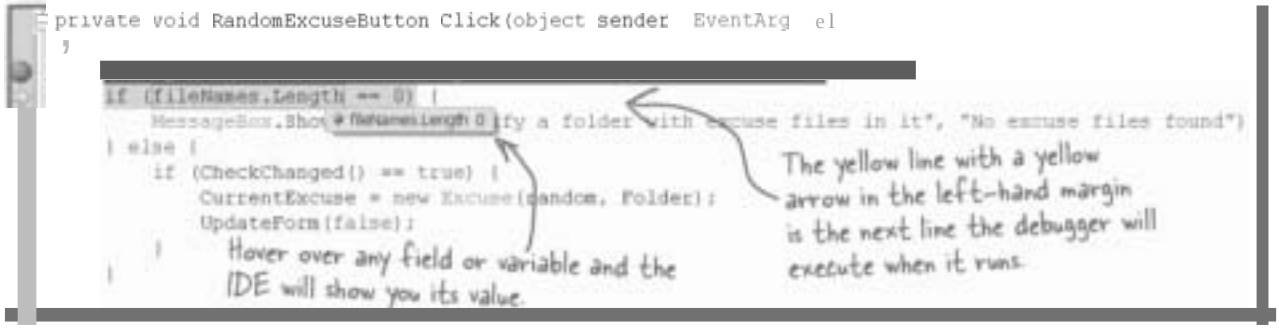
Let's use the debugger t" lak,'" closer look at the problem that ",. ran into in the excuse manager. It's a good place to get some practi..e "ilh the debugger, because y"I knuw exactly "hal you're looking for. (That's a luxury that you don't have most or the time!)



0 use a breakpoint to break—or pause—your program

You've got a starting point the exception happens wh,'n the Random Excuse button is clicked afi,'r an empty folder is selec·!t'd. So open lip the code f(,r the hun, 'no cluk anvwhc-re- in the firstlin.. "r the' me-thod. and select "Toggle Breakpoint" from the Debug menu (or press F9):

The debugger shows your breakpoints as red lines ..ith a big red dot in U, left-hand margin.



The IDE turned the line red and put a circle in the left-hand margin. That's the debugger's way of telling you that it set a **breakpoint** on the line. Now, when you debug the program in the IDE, execution will stop on that line. Give it a try—run the program in the IDE (using the same "Start Debugging" command you've been using all along). When you reproduce the problem, the line should turn yellow, with an arrow pointing to the code. Now your program's temporarily paused. The "Start Debugging" menu item's turned into "Continue Debugging," too. Click it—the program will pick up exactly where it left off, starting with the line you put the breakpoint on.

Now you know why the IDE command for running a program is "Start Debugging"—because you're using the debugger built into the IDE.

Start Debugging – when you have no breakpoints set – is the same as running your program.

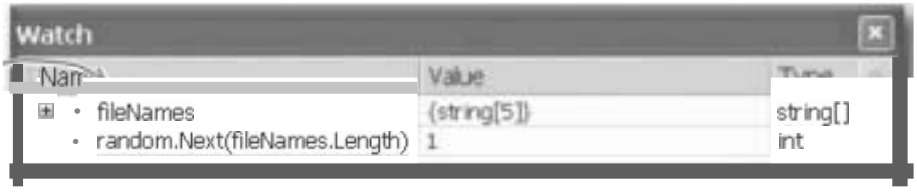
0 Step through the application

Use the **Step Into** command (using either the toolbar or the F11 key) to move through the application line by line. When it gets to the line that creates the new `Excuse` object, it'll jump straight into the constructor that you fixed. Step past the first line so it sets the `fileNames` variable. Then hover over the variable to see its value.

3 Use the Watch window to reproduce the problem

One really powerful feature "f the debugger i, the **Watch** window. which lets you check the value "f variables and fields ill your objects. 11," er vour mouse curs(' r over the "Length" part of `f i 1eNames _ Length`. Then select all of `f ileNames . Length`. right-click, and select "Add Watch" Ii'''''' Ihe, menu. It'll get added to the Watch window fwhi('h i, in the same pane as Output you can bring il IIp by selecting "Watch' unrh-r Windo(\" in the Debug menu). Then add each pi,ce of the statement: flrsr `randam.Next (fileNames•Length)`. It ,h"uld look something like Ihi,. depending oil how many files are in the fi,Id,'-you p'>inted to—in this case, we had fiv -, s `f i 1eNames` has five elements:

We'll use the Watch window to reproduce the proble.. that caused the exception We'll start by adding the fileNa..es array



You need to break (pause) the program before you can add a watch

O Set fileNames equal to on empty string *orroy*

Double-click in the empty space ill Ih,' \, "!I('!> window underneath Ih,' two watch variables. You'll get a cursor. Type this in: **`fileNames = new str 1ng[Oj`**. Watch the 1"1'row in the window— as soon a you hil enter. the valu("f `f i 1eNames` will change' I" `{str Lnq[0] )`. A re-evaluate, icon ⟲ should show up next to the `random . Next` line—click orr il and its value gets '.:110 0. So what happened?

Th.. Watch window has another very useful feature i! lets you **chang.. th.. valu..** of variables and fields Ihat il', displaying. A\nd il even lets you **exe**cut.. **m.. thods** and cr..at.. n..w ohj..cts and whe-n you do, it displays its re-evaluate icon (⟲) Ihal you ,'all click to tell il I" execut(Ihal lin,' again. Becauw sometimes running the same method twice· "'ill generat(different results (like with Random).

We know the problem happened with an e..pty fileNa..es array, so we'll use the Watch window to change its value to an e..pty string array.

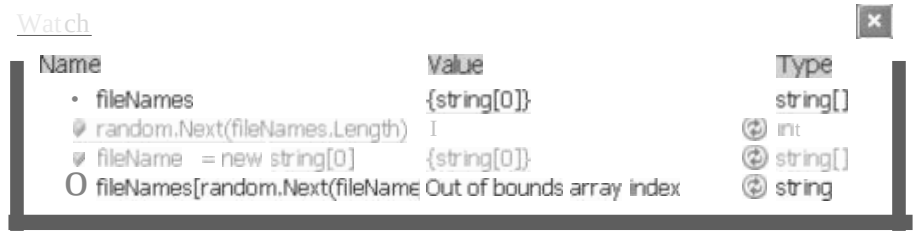


This icon tells the Watch window to reevaluate the Next() ...ethod.

5 Reproduce the problem that threw Brian's original exception

Here's wh..re debugging gets really interesting. Add one Ill<'n' li,ne 10the debugger—the statement that a,'I Il"lIy threw the exception: `fileNames [random.Next (f i leNames . Length) 1,_\`, soon as you type it in. Ihe Watch window evaluate it... and Ihat throws Ihe exception. It ",II, you that it found Ihe exception by displaying an exclamation point, and displays Ihe text "f the ..xc-ption in the Value column.

This exclamation point is the Watch window's way of telling you it found an exception



When **you** get an exception. you can go back and reproduce It in the debugger. That's another way that more descriptive exception messages can help you fix your code.

there are no Dumb Questions

Q: How come Brian's unhandled exception window looked different than the one in the IDE?

A: Because when you run a program inside the IDE, you're running it in the debugger, which **breaks the program** (as if you'd pressed the Break All button or inserted a breakpoint) as soon as it intercepts an exception, and displays it in a useful window. That lets you inspect the Exception object and your program's fields and variables so you can track down the problem.

When Brian ran his program, he wasn't running it from inside the IDE. He'd published his program and installed it, just like you did back in Chapter 1 with the Contact List program. You can run your program outside the IDE any time without publishing it—just build your program, which causes Visual Studio to create an executable file. Just look inside your project's folder for the `bin/` folder—one of its subdirectories should have the `exe` file for your application. If you run that, any exceptions that it throws will be unhandled and show the same window that Brian saw.

Q: So that's it? When an exception happens outside the IDE, my program just stops and there's nothing I can do about it?

A: Well, your program does stop when there's an ***unhandled*** exception. But that doesn't mean that all of your exceptions have to be unhandled! We'll talk a lot more about how you can handle exceptions in your code. There's no reason your users ever have to see an unhandled exception.

Q: How do I know where to put a breakpoint?

A: That's a really good question, and there's no one right answer. When your code throws an exception, it's always a good idea to start with the statement that threw it. But usually the problem actually happened earlier in the program, and the exception is just fallout from it. For example, the statement that throws a divide by zero error could be dividing values that were generated 10 statements earlier but just haven't been used yet. So there's no one good answer to where you should put a breakpoint, because every situation is different. But as long as you've got a good idea how your code works, you should be able to figure out a good starting point.

Q: Can I run any method in the Watch window?

A: Yes. Any statement that's valid in your program will work inside the Watch window, even things that make absolutely no sense to run inside a Watch window. Here's an example. Bring up a program, start it running, break it, and then add this to the Watch window: `System.Threading.Thread.Sleep(2000)`. (Remember, that method causes your program to delay for two seconds.) There's no reason you'd ever do that in real life, but it's interesting to see what happens: you'll get an hourglass for two seconds while the method evaluates. Then, since `Sleep()` has no return value, the Watch window will display the value, `"Expression has been evaluated and has no value"`, to let you know that it didn't return anything. But it did evaluate it. Not only that, but it displays IntelliSense pop-ups to help you type code into the window. That's useful because it'll let you what methods are available to an object when your program is running.

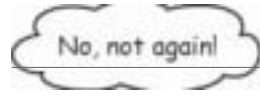
Q: Wait, so isn't it possible for something to run something in the Watch window that'll change the way my program runs?

A: Yes! Not permanently, but it can definitely affect your program's output. But even better, just ***hovering*** over fields inside the debugger can cause your program to change its behavior, because hovering over a property ***executes its get accessor***. If you have a property that's got a get accessor that executes a method, then hovering over that property will cause that method to execute. And if that method sets a value in your program, then that value will stay set if you run the program again. And that can cause some pretty unpredictable results inside the debugger. Programmers have a name for results that seem to be unpredictable and random: they're called **heisenbugs** (which is a joke that makes sense to physicists and cats in boxes).

When you run your program inside the IDE, an unhandled exception will cause it to break as if it had run into a breakpoint.

Uh-oh—the code's still got problems...

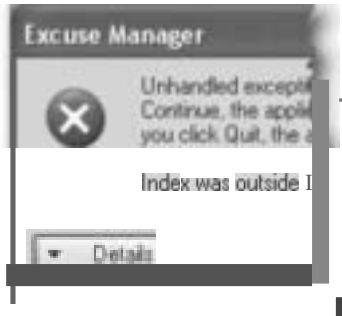
Brian was happily using his Excuse Manager, when he remembered that h,' had a folder Itll1.,r excuse that he Ill,,,l,' whe-n he first built the program —but he forgot that he made that folder ***before*** h,' addc:c1serialization to the program Let's see what happens..





O You r-an re-cr..at.. Brian's problem - .ill, 1creat.. your own text-based Excuse file using Notepad. Th" first lill" ,h"nld Ibe Ih,' description, the second ,holiid be the results, and the third sh<.uld be Ihe last used date *(" W!* 4/2007 I2:08:13 1M1"),

O 1"'1' open the Excuse Manager and "1"-11 "I' the excuse. 11 throw an exception! But this time, click on the Details hUliotl so we can take a closer look .u what il says. Pay attention to the call **stack** - that' what it's calle<1when a nu-tlu "I is called by another me-thod, which is called bv another method, etc.



The program threw a SerilllizationException Can we figure out what line threw it from the exception details?

It looks like there was a proble ith the BinaryF: atter—which makes sense, because it was trying to deserialize a text file

You can learn a lot from the call stack which tells you which methods were running You can see that the Excuse class's OpenFile() ..ethod was being called from its constructor Co ,ctor"l, which was called from the "Random Excuse" button's elek event handler.

..... Exception Text .

System Runtime SerialiZation SerializationExceptiorr End of Stream encountered teioe persrq was completed

at System.Runtime.Serialization.Formatters.Binary.__BinaryParser.Run()

at System.Runtime Serialization.Formatters.Binary.ObjectReader Deserialize(HeaderHandler handler, __BinaryParser sersarser, scoeanrcteo, soceenisCrossAppDomain, IMethodCallMessage methodCaliMessage}

at System.Runtime.Serialization.Formatters.Binary.BinaryFormatter.Deserialize(Stream serialization Stream, HeaderHandler handler, Boolean fCheck, Boolean isCrossAppDomain, IMelhodCallMessage methodCa llMessage}

at System.Runbme.SenalizatIOOFormatters.BInary.BinaryFormatter.Deserialize(Stream serializationStream)

at Chapterl0 Excuse.OpenFile{Smng ExcusePath) in C:\Documents and Settings\Administrator\My Doccrrentswsuet stoco 2005\Projects\Chapter10IChapterI 0\Excuse csI Ine 40

t Chapter10.Excuse..ctor(Random random, String Folder) in C:\Documents and SettIngsI Administrator\My DocumentsWisuai Studio 2005\Projects\Chapter10IChapterI 0\Excusecs I ine 30

at Chapter10.Form1.RandomExcuseButton_Click(Object sender, EventArgs e) in C:\ Documents and Settings\Administrator\My DocumenlsIVisuaI SnEO 2005\Projects\Chapter10\ Chapter10\Form1.cs:line 146

O So the 1Details hili" 'I' ill Ihe I"ihandled exception window tells you a Lt about what caused this problem *Can you think of **anything you can** do about* it?

Wait a second. Of course the program's gonna crash—I gave it 0. bod file, Users screw up all the time. You can't expect me to do anything about that, right?

c o



Actually, there is something you can do about it.

Yes it's true that users screw up all the time. That's a fact "r life. nlll that d,)esn't mean you can't d.. anything about il. There's a name t"l' programs that deal with bad data, malformed i"p"I, and other unexpected situations gracefully: they're called rob ust. And C# gives you some reallv powerful exception handling I'ools 10 help you make your programs lll"re robust. Because while vuu *can't* control what vour users dn, you „llll make ,un' that your program doesn't crash when thev do it.

ro-bust, adj.

sturdy in construction: able tu withstand or ovr-rcon«- adverse conditions. *After the Tacoma Narrows Bridge disastn; the cnil engineering team tool-Nt for a mort robust design for the bridge that would replace it.*



BinaryFormatter will throw an exception if there's anything at ali wrong with a serialized file.

It's easy *to get the Excuse Manager to throw* a *SerialiizationException-just feed it any file that's not* a *serialized Excuse object. When you try to deserialize an object from* a *file, BinaryFormatter expects the file to contain* a serialized *object that matches the* class *that* iI's *trying to read. If the file contains anything* else. *anything at all. then the Deserialize() method will throw* a *SerializationException,*

Hal1dle exceptiol1s with try al1d catch

In C#, you ,-an basically say, "**Try** Ihi, code, and if an exception occurs, **catch** il with this *othe* hil "f code." The part of code you're trying is the **try block**. and the part where you deal with exceptions is call..d the **catch** block. In Ihe catch block, you ,-,m d" thinks like prin t a friendly error message, instead or letting your program come to a screeching halt:

```
private void RandomExcuseButton_Click(object sender,
EventArgs e) {
  // . code you added a few pages ago goes here ...
  try {
    if (CheckChanged() == true) {
      CurrentExcuse = new Excuse(random, Folder);
      UpdateForm(false);
    }
  }
  atch {
    "Your excuse file was invalid."
                    a random excuse"),
```

Thi is the try block You start exception handling with try.

The catch keyword means that the block immediately following it contains an exception handler.

Put the code that might throw an exception inside the try block. If no exception happens, it'll get run exactly as usual, and the statements in the catch block will be ignored. But if a statement in the try block throws an exception, the rest of the try block won't get executed.

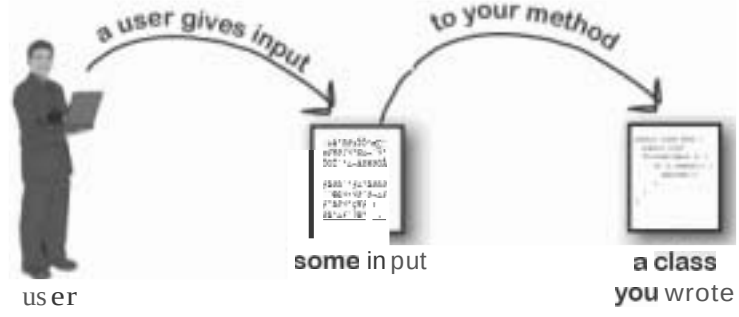


If throwing an exception makes your code automatically jump to the catch block, what happens to the objects and data you were working with before the exception happened?

What happens when a method you want to call is risky?

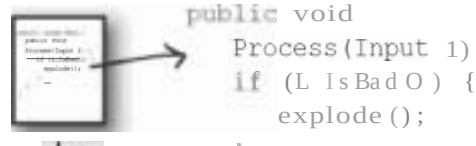
Users are unpredictable. They feed all sorts of weird data into your program, and click on things in ways you never expected. And that's just life, because you can handle unexpected ill-ful with good exception handling.

1 **Let's say your user is using your code, and gives it some input that it didn't expect.**

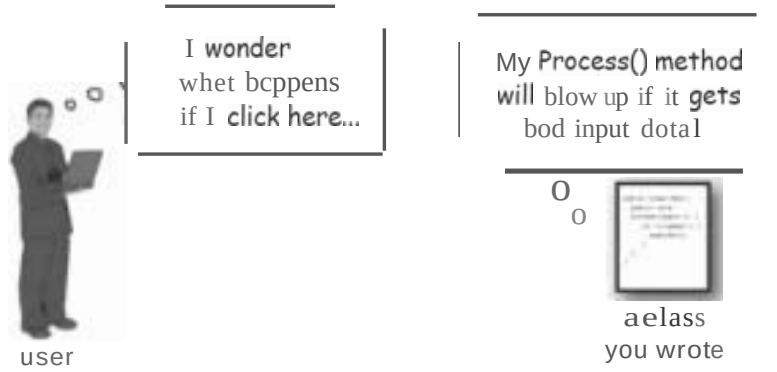


2 **That method does something risky, something that might not work at runtime.**

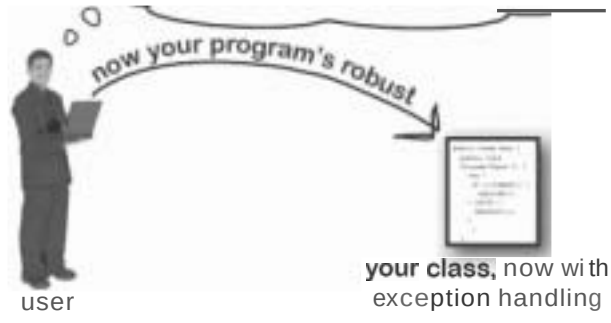
"Runtime" just means "while your program is running". Some people refer to exceptions as "runtime errors".



3 **You need to *know* that the method you're calling is risky.**



4 **You then write code that can handle the failure if it *does* happen. You need to be prepared, just in case.**



there are nc DumlJ Questions

0_: So when do I use try end catch?

A: Any time you're writing risky code, 01 code that could throw an exception. The trick ıs figuring out which code ıs risky, and which code ıs safer

You've already seen that code that uses ınput provided by a user can be risky Users gıve you ıncorrect files, words instead of numbers, names instead of dates, and they pretty much click everywhere you could possibly ımagıne And a good program will take all that ınput and work ın I calm, predictable way It might not gıve the users a result they can use. but ıt will let them know that ıt found the problem and hopefully suggest a solution

0_: How can a program suggest a solution 10 a problem it doesn't even know about in advance?

A: That s what the catch block ıs for A catch block ıs only executed when code ın the try block throws an exception It's your chance to make sure the user knows that something went wrong, and to let ee user know that it's a situation that might be corrected

If the excuse manager simply crashes when there's bad input, that's not particularly useful But if it tries to read the input and displays garbage ın the form, that's also not useful—fact, some people might say that it's worse. But if you have the program display an error message telling the user that it couldn't read the file, then the user has an idea of what went wrong, and information that he can use to fix the problem

0_: Is the debugger only used 10 troubleshoot exceptions?

A: No. The debugger's actually a really useful 100l that you can use to examine any code you've written. Sometimes it's useful to step through your code and check the value of certain fields and variables like when you've got a really complex method. and you want to make sure it s working properly.

But as you may have guessed tom the name "debugger," its most common use ıs to track down and remove bugs. Sometimes those bugs are exceptions that get thrown But a lot of the time, you1 be usıng the debugger to try to find other kinds of problems, like code that gives a result that you don't expect

0_: I'm not IUf, I totally got the Watch window. What's it for, again?

A: When you re debugging a program, you usually want to pay attention to how a few variables and fields change. Thats where the Watch window comes in, If you add watches for a few variables, the Walch window updates their values every time you step into, out 01 or over cooe ThaIlets you monitor exactly what happens to them after every statement, which can be really useful when you're trying to track down I problem

The Watch window also lets you type ın any statement you want, and it'll evaluate it If the statement updates any of the fields and variables ın your program, then it does that, too That lets you change values while your program ıs runnıng, which can be another really useful tool for reproducing exceptions and other bugs

Any changes you make ,. the Watch window just affect the data ın memory and only last as long as the program ıı runnıng Restart your ogram, and values that you changed be undone

The catclr **block** is enlj- **executed** **when** code in lite **try block throws** an exceplion. **ll** **gives** J ou a chance 10 **make sure** Jour user 1.3s **the** injermation 10 fix **the** pl·oMem.

Use the debugger to follow the try/catch flow

,\ u important part of exception handling is that when a statement in your trv block throws an exception, the rest of the code in the block gets **short-circuit**..d, The program" executi,,n immediately jll,np' I" the fir,1 ll"" in the catch block. *But do.. 't* ***take our word for*** it..,



o Make sure ,hal you've incorporaled all "I' the ,ode from this chapter illl" the R,"" I',n Excuse h"lIo" ill your Excuse Manager. Place a breakpoint on the [ir'llilH' in the Random Excuse h"I1"", Clic'k event hamU,'r. Then r"n your program ill Ih,' IDE. Click II", Fe,Id,'.. hllllloll and specify a folder with a single excuse file in il- and ma k,' sure it's a **not a** ,;a lid **excuse' file** (any other sort "f file will cause il 10 throw an excepti,,n). Press the Rande,m Excuse hllll"ll. Th.. debugger ,hould hn'a k the program ;,1 the breakpoint you placed. Pres Ihe "Step ()ver"],,,11< III (, lr F10) six Ii""" ". get h. rhe statement that calls the Excuse constructor. Here's what your debugger screen ,hollid I,.,k like:

Place a breakpoint on the first line of the Random Excuse button event handler

Step over the state..ents until your yellow "next state··ent" ba.. shows that the next state..ent to get executed will create the new Excuse object

```
private void RandomExcuseButton_Click(object sender, EventArgs e) {
    string[] fileNames = Directory.GetFiles(Folder, "*.excuse");
    if (fileNames.Length == 0) {
        MessageBox.Show("Please specify a folder with excuse files in it",
                        "No excuse files found");
    else {
        try {
            if (CheckChanged()) {
                CurrentExcuse = new Excuse(random, Folder);
                UpdateForm(false);

        catch
            MessageBox.Show(
              "Your excuse file was invalid.",
              "Unable to open a random excuse");
```

Make sure r- use the Step Over (F10) com..and in the debugger so it doesn't step into the CheckChanged() ..ethod

o Use Step Inl" :1'II) h> step into the new ,lalt"'ItIll, The-debugger will jUlllp 10 the Excuse ,'o"'II'Ilt'lOr, and p."ili"" its yellow "next statement" bar over the declaration line in Ih,' code. Keep hitting Step Into (F1 I) 10 step inlo Ih.. ()penFile() meth,,d. Watch what happens when you hit the Deserialize () line.

As soon as r- step into the new statement that creates the Excuse object, the debugger jumps to the construct·· code

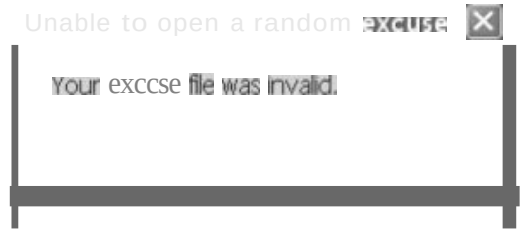
```
public Excuse(Random random, string folder)
{
    string[] fileNames = Directory.GetFiles(folder, "*.excuse");
    OpenFile(fileName.[random.Next(fileNames.Length)]);
```

O As soon as the debugger executes the neeerLa1ize() statement, the exception is thrown and the program jumps straight to the first statement in the catch block. It **short-circuit**..d right past the call to UpdateForm() and **jumped straight** to the catch block.

The debugger will highlight the catch statement with its yellow "next statement" block, but it shows the rest of the block in grey to show you that it's about to execute the whole thing.

```
private void RandomExcuseButton_Click(object sender, EventArgs e)
    string[] fileNames = Directory.GetFiles(Folder, "*.excuse");
    if (fileNames.Length == 0) {
        MessageBox.Show("Please specify a folder with excuse files in it",
                        "No excuse files found");
    else {
        try {
            if (CheckChanged()) {
                CurrentExcuse = new Excuse(random, Folder);
                UpdateForm(false);
            }
        }
        catch
            .Show(
            "Your excuse file was invalid.",
            "Unable to open a random excuse");
        }
```

O Start the program again by pressing the Continue button (or F5). It'll begin running the program again, starting with whatever's highlighted by the yellow "next statement" block, in this case, the catch block.



He-e's a career tip: a lot of C# programming job interviews include a question about how you deal with exception in a constructor.



Be careful with exceptions in a constructorl

You've noticed by now that a constructor doesn't have a return value, not even void. That's because a constructor doesn't actually return anything, Its only purpose is to initialize an object-which is a problem for axception handling inside the constructor. When an exception is thrown inside the constructor. then the statement that mea to instantiate the csees won't end up with an Instance of the object. That's why you had to move the try/catch block to the button's event handler. That way. if there's an exception in the constructor. the code won't axpact CurrentExcuse to contain a valid Excuse object.

If you have code that ALWAYS should rUII. use a filially block

When your program throws an exception, a couple or things can happen. II'the exception ***isn't*** handled. your proer.un will stop processing and crash. If the exception ***is*** handled, your code jumps 10 the catch 1,1,ock. lIuI what alx'Ill the rest or thc code iu your try I,I,ock? \What ir you were closing a sream· or cleaning up important resources? That code needs \0 rUIl. even ir all exception occurs, or you're going I" make a mess " I' your program's state. That's where ,I", finally block come in really handy. It "0'"" after Ih,' try ,,,,d cat.ch blocks. The **finally block** always runs, whether or nol an exception was thrown. Here's ho" you'd use il 10 lini;h the "","UI handling in the Random Excuse button:

```
private VOid RandomExcuseButton Cllck(obJect sender, EventArgs e)
    stringII fileNames = Directory.GetFiles(Folder, "*.excuse");
    if (fileNames.Length -- O)
        MessageBox.Show("Please specify a folder with excuse files in it",
                        "No excuse files found");
    else {
      try
          if (CheckChanged() -- true)
              CurrentExcuse - new Excuse(random, Folder};

      catch (Exception)
          CurrentExcuse    new Excuse();
          CurrentExcuse.Description - "";
          CurrentExcuse.Results - "";
          CurrentExcuse.LastUsed = DateTime,Now;
          MessageBox.Show(
            "Your excuse file was invalid.",
            "Unable to open a random excuse");
      }
      finally {
          UpdateForm(false);
      }
```

If the Excuse constructor throws an exception, we have no way of knowing what's in CurrentExcuse. But you do know that no instance of Excuse was created. So the catch block creates a new Excuse object and clears out all its fields.

The finally block makes sure that UpdateForm() gets run whether or not an exception was thrown. So if the Excuse constructor successfully read an excuse, it'll call UpdateForm(), but it'll also call it if the constructor threw an exception and cleared out the excuse.

Did)'UU "olie.. huw catch **was** f"U"w..d **by** (Except ion)? When you have a catch statement, you can follow it with a specific ki"d "r exception telling il what to catch. If you specify (Except ion) or leave it out, il catche all exceptions. But ir you only wanted '0 catch a Seria1izat i.onExcept ion. you could specify Iha, inside the parentheses i'"I";"1. ()r you could use an IOExcept 1on which will catch **any** fil.. input or oulput probl.....



O 1'pdate the Random Excuse button' eve-m handler with the code 0" the facing page. T h,-" piace a breakpoint "11 the first line ill the method and debug the program

O Run the program mumallv; and make 'UI1' that lhe Random Excuse button works when you set the program's folder 10 one with a bunch "r nonHa! oc-u-c files in it, T],,· debugger ,holll<1 break at the breakpoint vou set:

When U, "next state..ent" bar and the breakpoint are on the same line, the IDE shows you the yellow arrow placed over the big red dot in the margin.

```
private void RandomExcuseButton Click(object sender, EventArgs e)
    string[] fileNames = Directory.GetFiles(Folder, "*.excuse");
    if (fileNames.Length •• 0) I
        MessageBox.Show("Please specify a folder with excuse files in it",
                        "NO excuse files found");
      else I
        tr
        ,
            if (CheckChanged())
                CurrentExcuse • new Excuse(random, Foldet);

        ,
        catch (Exception) {
            CurrentExcuse • new Excuse();
            CurrentExcuse.Description • "";
            CurrentExcuse.Results • •";
            CurrentExcuse.LastUsed • DateTime.Now;
            MessageBox.Show(
              "Your excuse file was invalid.",
              'unable to open a random excuse");
        ,
        finally {
            UpdateForm(false);
```

❸ Step through the rest "r Ill" Random Excuse button's ,-"" nr handler amlmak,- sure il runs Ill<' wav you expect il 10. It ,houl<1 lini,h the t r y block. skip over the ce t ch bl,ıck (because 110 exceptions were throw"I, and Ih"11 execute the f Lna 11y block,

O Now set the program's 1,ld,-r s(lhilt it's pointed III the folder with one malformed excuse file ill il and click the Random excuse I",n",\, It shollid start executing the t r Y block, and 1hen jump III the catch block whe-n Nthrows the exception. ,\Ii,·r lr finishes all or the statements in Ihe catch block. it'll execute the f ina l ly block.

there are no Dumb Questions

Q_: Blck **up** I **second. So every** timl **my program** run. Inlo In n Clplion, **it's going** to . lop **whatever it' doing unles** I **specifically** wrill **code** to Cllch **it. How is** lhiligood lhing?

A: One of the best things about exceptions ıs that they make it really obvious when you run into problems Imagine how easy it could be in a complex application for you to lose track of all of the objects your program was working with. Exceptions call attention to your problems and help you root out their causes so that you always know that your program ıs doing what it's supposed to do

Any time an exception occurs in your program, something you expected to happen didn't. Maybe an object referel .... wasn't pointing where you thought it was, or it was possible for I user to supply • value you hadn't considered. or • file you thought you'd be working with suddenly isn't av If something like that happened and you didn't know it, it's likely that the output of your program would be wrong, and the behavior from that point on would be pretty different than you expected when you wrote the program

Now ımagine that you had no idea the error had occurred and your users started calling you up with incorrect data and telling you that your program was unstable That's why it's a *good* thing that exceptions disrupt everything your program ıs doing. They force you to deal with the problem while it s easy to find and fix

0_: Okay, '0now what's I handled exception Ind **what's** In **unhandled exception?**

A: Whenever your program throws an exceptoo. the runtime environment will search through your code looking for a eaten block 10that matches it. If you've written one, the catch block will execute and do whatever you specified for that particular exception Since you wrote a catch block to deal with that error up front, that exception ıs considered handled If the runtime can't find a catch block to match the exception, it stops everything your program ıs doing and raises an error. Then, you'd call the exception unhandled

0_: What was that bit **about specifying** I **particular kind** of **exception** to **catch? Why would I ever want** to **do that?**

A: You ***usually*** don't want to catch every kind 01 exception. In fact, you should do your best to avoid catching `Exception.` and instead catch specific exceptions For example, let's say you wanted your Excuse class to prevent a `FileNotFoundException` from getting back to the form—say, • you wanted to make it so that • you tried to open a tile that wasn't found, it would automatically create I excuse file with that filename and gıve it some default values Then you could add a `try` block followed by `catch (FileNotFoundException).` Then if • file isn't found, the class can handle it—but an `IOException` or `serializationException` would not get caught, and the exception handler you added to the form would catch it. But if you do that, you have to make sure that there' *some* method in ee call stack that does have a catch-all exception handler, otherwise the exception would be unhandled And that would cause the users to see the ugly "unhandled exception" crash

0_: What happens when you have ı cilch that **doesn't specify** I **particular** IIclption?

A: A catch block like that will catch any kind of exception the try block can throw

0_: If I **catch block with** no **specified exception** will Cllch **anything, why** would 1**ever** WlnI to **specify?**

A: Good question Because certain exceptions might require different actions to keep your program moving An exception that happens when you divide by zero might have • catch block where you go back and set some number values to save some of the data you've been working with. A null referel.... exception might require that you create new ınstances of an object if you re going to recover

Q: Does all error handling happen in I **try/catch/finally sequence?**

A: No You can m . it up I bit You could have **multiple catch blocks** if you wanted to deal with lots of different kinds of errors You could also have no catch block at all It's legal to have I try/finally block That wouldn't handle any exceptions, but it would make sure that the code in the finally block ran even " you got stopped half way through the try block But we'll talk I lot morl about that in ı minute

An **unhandled exception** means **your** p"ogram **will** run unp"edictably. That's **why the program stops** whenever it **runs** into one.

Pool puzzle



Your job is to take code snippets from the pool and place them in the blank line, in the program. You can use the same snippet more than once, and you won't need to use all the snippets. Your goal is to make the program produce the output.

Output: ⟶ G'day Mate!

```
using System.IO;
public static void Main()
    Kangaroo Joey = new Kangaroo();
    int Koala = Joey.Wombat(
          Joey.Wombat(Joey.Wombat(1)));

        Console.WriteLine(Koala +
                " eggs per pound");

    catch
        Console.writeLine("G'Day Mate!");
```

```
public class Kangaroo
{
    _____ f51
    int Croci
    int Dingo = 0;

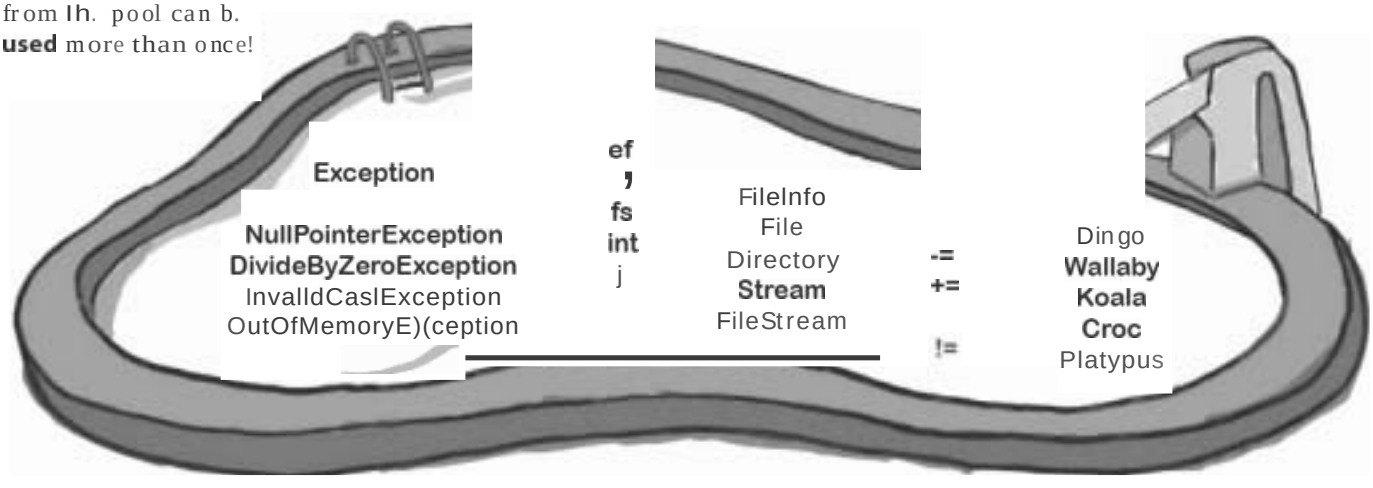
    public int Wombat(int Wallaby)

        try {
            if _____ > 0) {
                    .OpenWrite("wobbiegong");
                croc  0;
            else if (     _ -: 0)
                Croe • 3;
            else
                    .OpenRead("wobbiegong");
                Croc - 1;


        catch (IOException)
            Croc - -3;

        catch
            Croe = 4;
        }
        finally
            if (_      > 2) {
                Croc     Dingo1
```

Not. : **each snippet** from Ih. pool can b. **used** more than once!



Pool Puzzle Solution



```
public static vOld Main() {
    Kangaroo Joey - new Kangaroo(1,
    int Koala - Joey.Wombat(Joey.Wombat(Joey.Wombat(1)));
    try
        Console.WriteLine((lS / Koala) + " eggs per pound");
    }
    catch (DivideByZeroException) {
        Console.WriteLine("G'Day Mate!");

public class Kangaroo {
    FileStream fs;
    int Croc;
    int Dingo = 0;

    public int Wombat(int Wallaby) {
        Dingo ++;
        try I
            if (Wallaby > 01 {
                15 • File.Openwrice("wobbiegong");
                croc • 0;
            else if (Wallaby < 0) {
                Croe • 3;
            else {
                fs - File.OpenRead("wobbiegong");
                Croc - 1;

        catch IIOExeeption) }
            Croe - -3;

        catch '
            Croc = 4;
        }
        finally {
            if (Dingo > 2) {
                Croc -= Dingo;

        }
        return Croc;
```

Joey.Wombat() is called three ti..es, and the third time it ..etu..ns ee-e. That causes the WriteLine() to throw a DivideByZeroException

This catch block only catches exceptions where the code divides by zero

The clue that this i• a FileStream i. that it has an OpenRead() method and th..ows an IOException

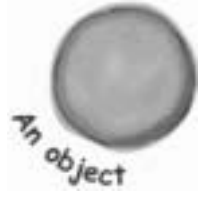
This code opens a file called "wobbiegong" and keeps it open the first time it's called. Later on, it opens the file again. But it never closed the file, which causes it to throw an IOException.

You already know that you always have to close files when you're done with them. If you don't, the file will be locked open, and if you try to open it again it'll throw an IOException

Use the Exception object to get information about the problem

We've been saying all along that .NET generates an Exception object when an exception is thrown. When you write your catch block, you have access to the object. Here's how it works:

1. An object is humming along, doing its thing, when it encounters some something unexpected and throws an exception.



Uh-oh—what the heck happened?

2. Luckily, its try/catch block caught the exception. Inside the catch block, it gave the Exception a name: **ex**.

```
try {
    DoSomethingRisky();
}
catch (Exception ex)
    string message = ex.Message;
    MessageBox.Show(message, "An error occurred")
}
```

When you specify a specific type of exception in the catch block, if you provide a variable name then your code can use it to access the Exception object.

3. The exception object stays around until the catch block is done, then the **ex** reference disappears, and it's garbage collected.



Use more than one catch block to handle multiple types of exceptions

You know that you can catch a specific type of exception, but what if you write code where more than one problem can occur? In these cases, you may want to write code that handles each different type of exception. That's where using more than one catch block comes in. Here's an example from code from the beehive honey processing plant. You can see how it catches several kinds of exceptions. In some cases it uses properties in the Exception object. It's pretty common to use the **Message** property, which usually contains a description of the exception that was thrown.

You can also call the exception's ToString() method to get a lot of the pertinent data into your MessageBox.

```
public void ProcessNectar(NectarVat vat, Bee worker, HiveLog log)
    try
        NectarUnit[] units = worker.EmptyVat(vat);
        for (int count = 0; count < worker.UnitsExpected, count++)
            stream hiveLogFile = log.OpenLogFile();
            worker.AddLogEntry(hiveLogFile);
    catch (IndexOutOfRangeException) {
        vat.Emptied = true;

    catch (IOException ex) {
        worker.AlertQueen("Log file is corrupted: " + ex.Message);

    catch (Exception ex) {
        worker.AlertQueen("An unspecified error happened: "
            + "Message: " + ex.Message + "\r\n"
            + "Stack trace: " + ex.StackTrace + "\r\n"
            + "Data: " + ex.Data + "\r\n");

    finally
        vat.Seal();
        worker.FinishedJob();
```

If you won't use the Exception object, there's no need to declare it.

When you have several catch blocks, they're examined in order. In this code, first it checks to see if there was an index out of range. If not, then it'll check for a file I/O exception. The last catch block is a general catch-all exception that will get executed for any exception that wasn't already caught.

This catch block assigns the exception to the variable ex, which it can use to get information from the Exception object.

It's fine for two blocks to use the same name ("ex") for the Exception

This statement uses three properties in the Exception object: Message, which has the message you'd normally see in the exception window in the IDE ("Attempted to divide by zero"); StackTrace, which gives you a summary of the call stack; and Data, which sometimes contains pertinent data that's associated with the exception.

One class throws an exception, another class catches the exception

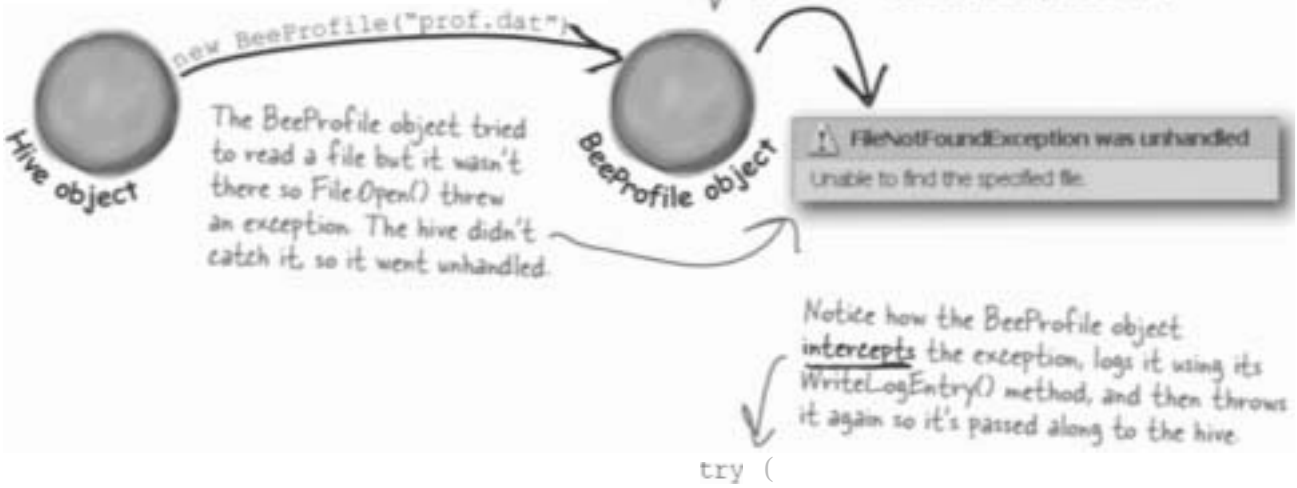
Of course, one method in a single class can throw and catch its own exceptions.

When you're building a class, you don't always know how it's going to be used. Sometimes other people are using your objects in a way that causes problems, sometimes you do it yourself! That's where exceptions come in.

The whole point behind throwing an exception is to say what might go wrong, so you can put in place some sort of contingency. You don't usually see a method that throws an exception and then catches it. An exception is usually thrown in one method and caught in a totally different one, usually in a different object.

Instead of this...

Without good exception handling, one exception can halt the entire program. Here's how it would work in a program that manages bee profiles for the queen bee.

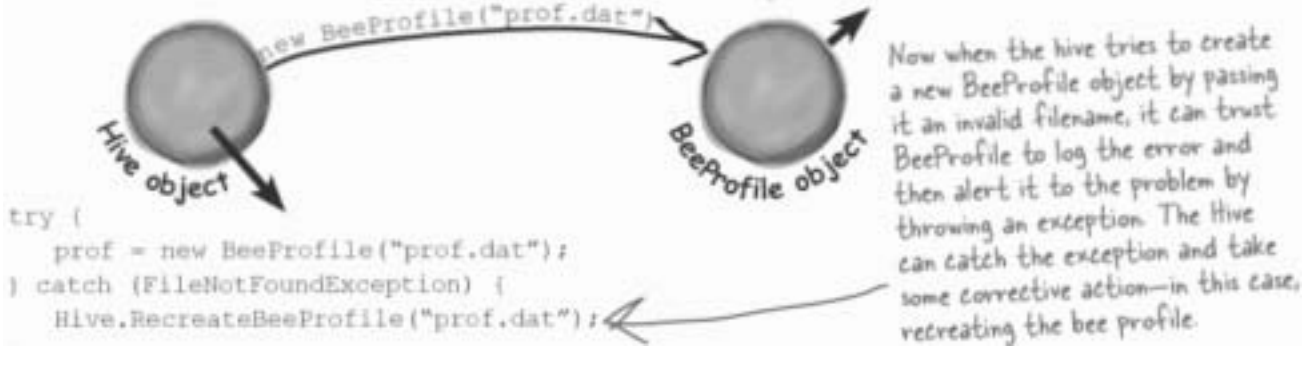


...we can do this.

The BeeProfile object can intercept exceptions, add a log entry. Then it can rethrow the exception back to the hive, which catches it and recovers gracefully.

Notice how the BeeProfile object intercepts the exception, logs it using its WriteLogEntry() method, and then throws it again so it's passed along to the hive.

```
try {
   stream = File.Open(profile);
} catch (FileNotFoundException ex) {
   WriteLogEntry("unable to open " +
      profile + ": " + ex.Message());
   throw ex;
}
```

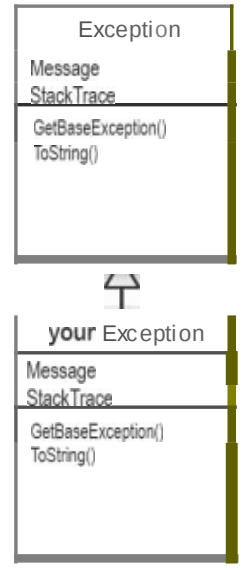


Bees need an OutOfHoney exception

Your classes can throw their own exceptions. For example, if you get a null parameter in a method that was expecting a value, it's pretty common to throw the same exception a .NET method would:

```
throw new ArgumentNullException();
```

But sometimes you want your program to throw an exception because of a condition that should happen when it runs. The bees we created in the hive, for example, consume honey at a different rate depending on their weight. If there's no honey left to consume, it make sense to have the hive throw an exception. You can create a custom exception to deal with that specific error condition just by creating your own class that inherits from Exception and then throwing the exception whenever you encounter a specific error.




```
public class OUtOfHoneyException : System.Exception {
    public OutOfHoneyException(string message) : base(message) { }
}

public class HoneyDeliverySystem {

    public void FeedHoneyToEggs() {
        if (honeyLevel == 0) {
            throw new OUtofHoneyException("The hive is out of honey.");
        } else {
            foreach (Egg egg in Eggs) {
                // ...
            }
        }
    }
}

public partial class Form1 : Form {

    private void consumeHoney_Click(Object sender, EventArgs e) {
        HoneyDeliverySystem delivery = new HoneyDeliverySystem();
        try {
            FeedHoneyToEggs();
        }
        catch (OUtOfHoneyException ex) {
            MessageBox.Show(ex.Message, "Warning: Resetting Hive");
            Hive.Reset();
        }
    }
}
```

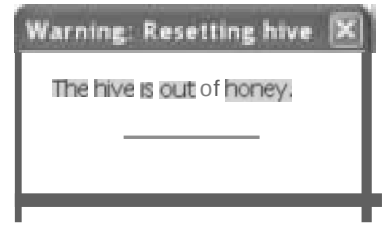
You need to create a class for your exception and make sure that it inherits from System.Exception. Notice how we're overloading the constructor so we can pass an exception message.

This throws a new instance of the exception object.

If there's honey in the hive, the exception will be thrown and never get caught and this code will run.

You can catch a custom exception by name just like any other exception, and do whatever you need to do to handle it.

In this case, if the hive is out of honey none of the bees can work, so the simulator can't continue. The only way to keep the program working once the hive runs out of honey is to reset it, and we can do that by putting the code to reset it in the catch block.



Exception Magnels

Arrange the magnets so the application writes the output to the console.

```
public static void Main() {
    Console.Write("when it ");
    ExTestDrive.Zero("yes");
    Console.Write(" it ");
    ExTestDrive.Zero("no");
    Console.WriteLine(".");
}

class MyException : Exception { }
```

output:

```
when it thaws It throws.
```

Every value type—including constants like "yes" or 37 or true—has built-in methods. So "yes".Equals(t) returns true if the variable t contains the string "yes"

```
if ("yes".Equals(t))

Con.ol.Write("a");
Con.Ol.Write("o");
Console.Write("t");

Consol.Write("...");
eon.ol.Write("s");

try {
} catch (MyException) {
    throw ne... MyException();
} finally {
    doRisky(test);
}

Con.ole Write("r");

public c ass ExT..tDrive {
    public static void Zero(String test) {
        atatic void doRiaky(Str!ng t) {
            Con.ol.Write("h");
        }
```

Exception Magnets Solution

Arrange the magnets so the application writes the output to the console.

```
public static void Main() {
    Console.Write("when it ");
    ExTestDrive.Zero("yes");
    Console.Write(" it ");
    ExTestDrive.Zero("no");
    Console.WriteLine(".");
}
```

output:

```
when it thaws it throws.
```

```
class MyException : Exception { }
```

This line defines a custom exception called MyException, which gets caught in a catch block in the code.

```
public class ExTestDrive {
    public static void Zero(string test) {
        try {
            Console.Write("t");
            doRisky(test);
            Console.Write("");
        } catch (MyException) {
            Console.Write("");
        } finally {
            Console.Write("w");
        }
        Console.Write("s");
    }

    static void doRisky(string t) {
        Console.Write("h");
        if ("yes".Equals(t)) {
            throw new MyException();
        }
        Console.Write("r");
    }
}
```

The Zero() method either prints "thaws" or "throws", depending on whether it was passed "yes" or something else as its test parameter

The finally block makes sure that the method always prints "w". And the "s" is printed outside the exception handler, so it always prints, too.

This line only gets executed if doRisky() doesn't throw the exception.

The doRisky() method only throws an exception if it's passed the string "yes".

BULLET POINTS

- Any statement can throw an exception if something fails at runtime
- Use a try/catch block to handle exceptons Unhanded exceptions will cause your program to stop execution and pop up an error window
- Any exception In the block of code after the try statement will cause the program's esec aon to immediately jump to the nrst statement in the block of code after catch.
- The Exception object gives you Information about the exceptcn that was caught. If you SpeCify an Exception vansbe in your catch statement, that vamoe will contain intormation about any exception thrown in the try block

```
try t
  // statements that might
  // throw exceptions
  catch (IOException ex) {
  // if an exception is thrown,
  // ex has information about it
```

- There are many different uOOs of exception that you can catch. Each has its own object lhal lnhents from Exception. Try to acoo just catching Exception-catch specnc exceptions.
- Each try can have more than one catch

```
try {     |
catch (NullReferenceException ex) {
   // these statements will run if a
   // NullReferenceException is thrown
|
catch (OverflowException ex) t ... )
catch (Exception ex) (
   // Any exception that hasn't been
   // caught will jump to this block
```

- Your cooe can throw an escecnco using throw:

```
throw;

throw new Exception("Exception message");
```

- You can create a custom exception by inheriting from the Exception base class

```
class CustomException : Exception;
```

- Most of me time, you only need 10 throw exceptions that are built into .NET. hke ArgumentException. The reason you use different kinds of exceptcos is so that you can give more information to your users Popping up a window with the text "An unknown error has occurred" IS not nearly as useful as an error message Ihat says 'The excuse !older IS empty. Please select a different folder If you want to reac excuses "

An easy way to avoid a lot of problems: using gives you try alld 'Ilia'y 'or 'ee

Remember when you declare a reference in a "usi " statement its Dispose() me hod is atically called at the end of th block

You already know Ihal "sing i, all easy way to make sure that your files always get closed. BUI what you didn't know thai it's really **just a C# shortcut** for try and finally'

```
using (YourClass c
        = n•• YourClass () | |
    // code
|
```

is the same as this

```
YourClass c = n••
YourClass ();

try |
    // code
| finally |
    c.Oi spose ();
|
```

When r- use a using state..ent, you're taking advantage of finally to ..ake sure its Dispose() ..ethod is always called

Exception avoidance: illlplellIent Illisposable to do your own clean up

IDisposable is a really effective way to avoid com..on exceptions and proble s Make sure you use using state ents any ti...e you're working with any class that imple..ents it.

Streams are great, because they already have code writte-n r" close lhemselves when II,,·,bject i, disposed "r. BU what if yont have your ,>wn custom object, ;lIld il always needs to do something when it's disposed "r.' Wouldn't il be great if you could write your own cc,d,' th,u got nlll if your object was used ill a using statement?

C# lets you d"]ll' 1 that with the Hij epoeabj,e interface. Implement j DfsposabIe. and write vour clean up code 1ll the DIspose() method, like this:

You can only use a class in a °osing" state...ent if it imple..ents IDisposable; otherwise, your program won't co..pile

Your object must i...ple...ent IDisposable if you want to use your object within a using state...ent.

```
class Nectar     Disposable I
  private double amount;
  private BeeH1ve hive;
  private Stream h1veLoq;
  public Nectar(double amount, BeeH1ve hive, Stream hiveLoql {
    this.amount = amount;
    this.hive = hive;
    this.hiveLog = hiveLog;
  }
  public void Dispose() {
    H1ve.Add(amount) ;
    H1ve.WriteHiveLoqEntry(hiveLoq,
        amount + "mg of nectar was added") ;
  }
```

The IDisposable interface only has one ...e be..: the Dispose() ...ethod Whateve.. y put in this ..ethod will get executed at the end of the using state..ent

This particular code always logs the amount of nectar added. It's important, and must happen, so we put it in the Dispose() method.

You'll often see one using state...ent inside of another.

We can use multiple using statements now. First, let's "". a built-in object that implements rnispoeeb.ie. Stream. Th"ll. we'll work with our updated Nectar object, which also implements IDisposable:

```
using (Stream Log = new File.Write("log.txt"))
using (Nectar nect = new Nectar(16.3, hive, Log)) {
    Bee.FlyTo(flower);
    Bee.Harvest(nect);
    Bee.FlyTo(hive);
}
```

The Nectar object uses the Log stream, which will close automatically at the end of the outer using statement.

Then the Bee object uses the nect object, which will log automatically at the end of the inner using statement.

there are no Dumb Questions

Q: Can I only use objects that implement `IDisposable` with I using statement?

A: Yes. `IDisposable` is tailor-made to work with `using` statements, and adding a `using` statement is just like creating a new instance of a class, except *that* it always calls its `Dispose()` method.

Q: Can you put any item inside a `using` block?

A: Definitely. The whole idea with `using` is that it helps you make sure that every object you create with it is disposed. But what you do with those objects is entirely up to you. In fact, you can create an object with a `using` statement and never even use it inside the block. But that would be pretty useless, so we don't recommend doing that.

Q: Can you call `Dispose()` outside of a `using` statement?

A: Yes. You don't ever actually *need* to use a `using` statement. You can call `Dispose()` yourself when you're done with the object. Or you can do whatever cleanup is necessary—like calling a stream's `Close()` method manually. But if you use a `using` statement, it'll make your code easier to understand and prevent problems that happen if you don't dispose your objects.

Q: You mentioned a "try/finally" block. Does that mean it's okay to have a try and finally without a catch?

A: Yes! You can definitely have a try block without a catch, and just a finally. It looks like this:

```
try {
    DoSomethingRisky();
    SomethingElseRisky();
}
finally {
    AlwaysExecuteThis();
}
```

If `SomethingRisky()` throws an exception, then the finally block will immediately run.

Q: Does `Dispose()` only work with files and streams?

A: No. There are a lot of classes that implement `IDisposable`, and when you're using one you should always use a using statement. (You'll see some of them in the next few chapters.) And if you write a class that needs to be disposed in a certain way, then you can implement `IDisposable`, too.

If try/catch is so great, why doesn't the IDE just put it around everything? Then we wouldn't have to write all these try/catch blocks on our own, right?



You want to know what **type** of exception is thrown, so you can handle **that** exception.

There's more to exception handling than just printing a generic error message. For instance, if the customer finder says it can't find a customer, we know we've got a `FileNotFoundException`, we might print a helpful error telling the user where the right files should be located. If we have an exception related to databases, we might send an email to the database administrator. Your code depends on you catching *specific* exception types.

This is why there are so many classes that inherit from Exception, and why you may even want to write your own classes to inherit from Exception.

Ihe worst catch block EVER: comments

A ,'a1,h block will let your proaram k,"'p running if you want. An exception gets thrown. you catch the exception, and instead of shutting down and giving an error message, you keep going. BIll sometimes that's nol such a good thing.

Tak,' a look at the DivisorCalculator. which seems to I", acting funny all the time. What's going "n')

```
public class Calculator I

  public void Divide(float dividend, float divisor) {
    try |
      this.quotient • dividend / divisor;
    catch (
    // Note from Jim: we need to figure out a way to prevent
    // people from entering in zero in a division problem.
```

Here's the problem. If divisor is zero, this will create a DivdeByZeroException

But there's a catch block. So

The progra.....er thought that he could bury hi. exceptions by using an e..pty catch block, but he just caused a headache for whoever had to track down proble.... with it later

Vou should ha"dle your exceptio"" nor bury them

.1"'1 becaus.' you c:... keep y('"1 program running d,)esn'\ mean you've *handled* your exceptions. In rhe ",)d" above, th.. calculator won't crash .. at least, n„1 in the Divide () method. But what il some other ,-,)de .:alls that meth(×l, and tries 1" print the results? [l' the divisor was zero, then the method probably returned an i"correct (,,,,,Iunexpected) value.

Instead of j",t adding a comment, and burying 1he "X""pli,)n, you need 10 hand!e lh.. ..**xception**. ,\nd if you're nul able 10 handle the problem, ***don't Ieaive emply or commenteJ catch blocks!*** That ju-t makes il harder for someone else l" track <|"" il what' going un. It's better 10 let the program ,'onli"",' \" throw exceptions, because then it's easy to figure out what' going wrong.

Tell1porary solutiolls are okay ltell1porarily)

Sometirue' you find a problem, and know it's a problem, hm aren't sure what '0 do about it. In thes(cases, you might want to log the problem, and not,· what's going on. That's not as good as handling the "X"(I"i"", 1"" it's better tha", doing nothing.

Here's a temporary solution to the calculator:

```
public class Calculator {

  public void Divide(float dividend, float divisor) {
    try |
      this,quotient _ dividend / divisor;
      catch (Exception ex) {
      StreamWriter sw = new StrearnWriter(@'C:\Logs\errors,txt");
      sw.WriteLine(ex.getMessage() ;
      sw.Close () ;
```

This still needs to be fixed, but short-term, this makes it clear where the problem occurred.

)

I get it. It's sort of like using exception handling to place a marker in the problem aree.

O

C



Handling exceptions doesn't always mean the same thing as FIXING exceptions.

It's never good to have your program bomb out But it's way worse to have rl0' idea «hv it's crashing or what it's doing to users data. That's why you w'('d to h,' , ,,n' that you're always dealing with the errors you can I'n'(h"1and logging the on," you can't.

A few simple ideas for exception handling



Design your code to handle failures gracefully.



Give your users useful error messages.



Only use built-in .NET exceptions where you can. Only write custom exceptions when you need to give custom information.



Think about code in your try block that could get short-circuited.



Ill and most of all ...

Avoid unnecessary file system errors... always use a using block any time you use a stream!

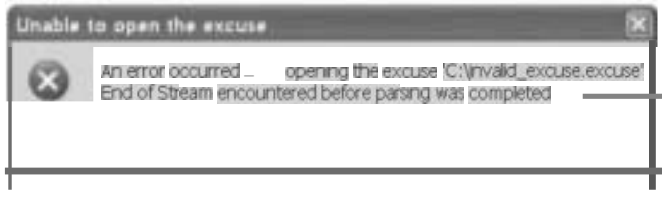
ALWAYS ALWAYS ALWAYS!



Use what you krow about t r y l c a t c h l f i n a l l y to improve the exception handling in Brian's excuse manager.

O

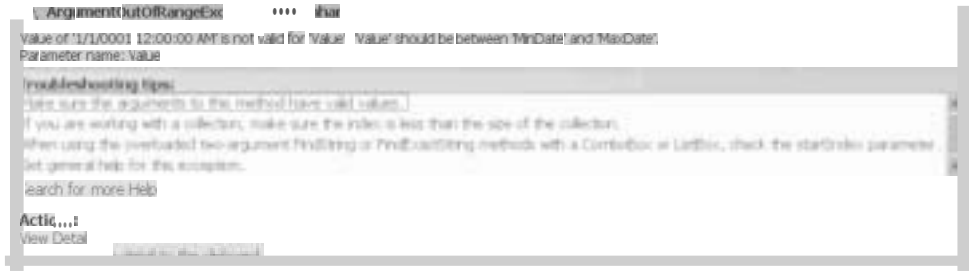
Add exceptio" hai,dling 10 the ()I" '" button's Click event handler..JU I make a simple try/catch block 111"11"'1" up a message box. Here's what it sl,,'uld 1'"1' "l';f you try 10 open " l' a Iii,· that's not a real excuse file



You'll get this exception message with a really small file, but a bigger one will give you a different error ("The input stream is not a valid binary format..."). Your message box should work either way.

O

You're ""1 done yet. ()I" '" up Ihe excuse manager, select a folder, "111,,1' data into the "Description" ,""t "Last Results" boxes, **but** don't l'ntl'r a **Last** Usl'd datl', Now select a folder and try saving the excuse. Did vou get this `ArgumentOutOfRange` exception?



Use the debugger to track down Ih,' exception. This particular exception i, totally avoidable— you can fix the program and make sure th.u Ihe exception never happens?
IU inl: This has nothing to do with adding a tryI catch block. You'll need to ***figure*** *o"t* ***.:hy*** *the "Last Used" dalf is causing a problem. Look carefully at the exception message for clues.)*

O

One 1'1,1 thing. Before the program threw the `Argument OutotRange` exceptiOll_;1 saved " ul a file. Load thar file in- you ,ho"ld get the same exception. .\nd you'll get a di'[k-n-nr exception if you try to open a file that's not a valid excuse file. Add an exception handling block ***nes ed inside the one***) '0 " ***added i.. step 2*** 10 make sure it do""'1 fail when you try to load an invalid excuse file (which can happen ill several ';lu"li"",). Here's what to do:

1. Declare a boolean variable called `c Le a r Form` above the try/catch block You'll set this tu true if there's an exception. and check it later to ".,- if the fonn ,hould be cleared.
2. Add ,,,,,'Iher try/catch block i" , id,' the ow' you just added 10 the ( )I" '" hUllol!,
3. .Add a `f i na11y` block to the ",,,,,,. `try Icat eh` to reset the fi,,-m 1" ii' original empty state. Reset `Last.Used . Val ue` to `DateTime . Now`(which returns the current date) if the `clearForm` variable is set tu `true`.



Use what you knew about t r y l c a t c h l t i n a 1 1 y to improve the exception handling to Brian's excuse manager,

```
private void open_Click(object sender, EventArgs e) {
    if (CheckChanged()) {
        openFileDialogl.lnitialDirectory = selectedFolder;
        openFileDialogl.Filter =
          "Excuse files (•.excuse)|*.excuse|All files ('.')I'.'";
        openFileDialoql.FileName = description.Text + ".excuse";
        DialogResult res"lt = openFileDialogl.ShowDialog();
        if (result __ DialogResult.OK)
            bool clearForm _ false;

            currentExcuse = new Excuse(openFileDialogl.FileName);

            try {
                UpdateForm(false);

            catch {
                MessageBox,Show("The excuse file ,"
                    + openFileDialogl.FileName + "' is invalid",
                    "Unable to open the excuse");
                clearForm = true;

        catch (Exception ex) {
            MessageBox.Show("An error occurred while opening the excuse '"
                + openFileDialogl.FileName + "'\n" + ex.Message,
                "Unable to open the excuse", MessageBoxButtons.OK,
                MessageBoxlcon.Error);
            clearForm   true;
    }
    finally {
        if (clearForm) {
            description.Text = "";
            results.Text = "";
            lastUsed.Value = DateTime.Now;
```

Here's the t..y/catch block to create a pop up error, in case proble... occur when the form calls the Excuse constructor to load an excuse

He..e's a nested try/catch It handles exceptions that happen if the file gets oa[d]ed but has invalid data i...it That'I not the same as problems ari , from the Excuse constructor

Here's the messagebox from the outer try/catch block. It prints the exception message

Both catch blocks set clearForm to true so that this finally block knows that the form should be reset. It's okay to have code that interacts with your finally block, since you know finally blocks will always run.



Exceptioncross

Across

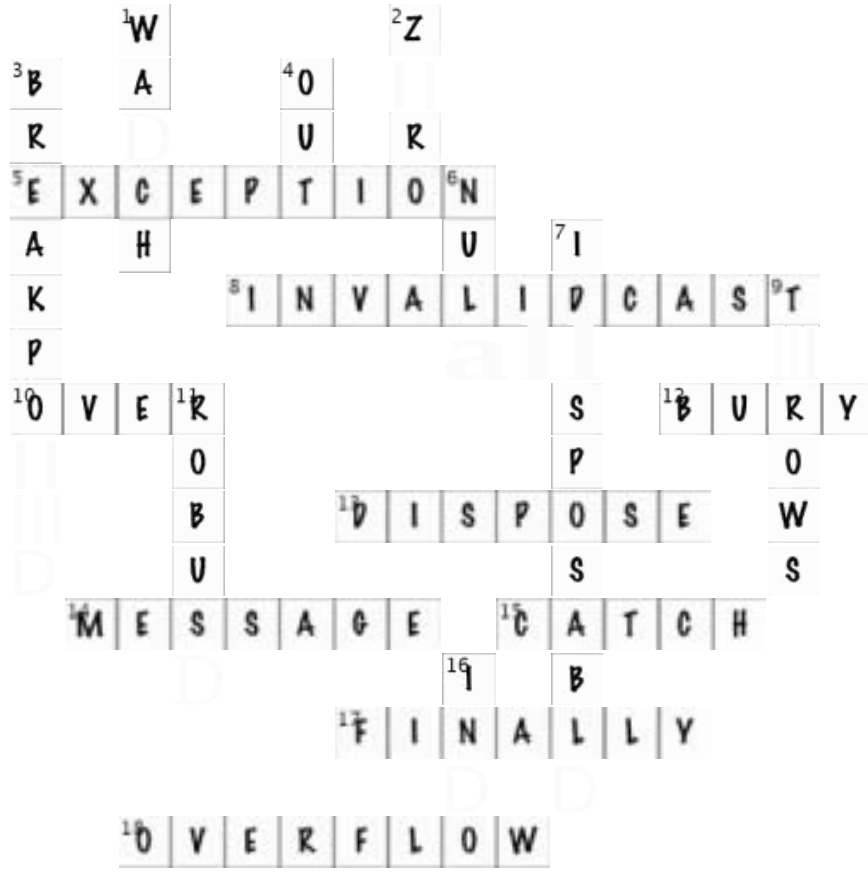
5. The base class that DivideByZeroException ard FormatException inherit from
8. An Exception happens when you try to cast a value to a variable that *can't* hold it
10. If the next statement is a method, "Step _ _ • tells the debugger to execute all the statements in the method and break immediately afterwards
12. If you ____ your exceptions, it can make them hard to track down
13. Th is eeltod is always caned at the end of a using block
14. The field in the Exception object that contains a string with a description
15. One try block can have multiple _____ blocks
17. The block contains any statements thai absolutely must be run after an exception is handled
18. An Ercepton means you tried to cram a number that was too big into a variable that couldn't hold it

Down

1. The window in the IOE that you can use to check your variables' values
2. You'll get an ercepucn if you try to divide by tte
3. Toggle this if you want e e debugger to stop execution when it hits a specific line of code
4. 'Step____• tells the debugger to execute the rest of the statements in the current method and then break
6. What a reference contains if it doesn't point to anything
7. Yoo can onty declare a variable with a using statement if it implements this interface
9. When a statement has a problem, it an exception
11. A program that handles errors well
16. If the next statement is a method, "Step _____• tells ttle debugger to execcte the first statement in lhat method



Exceptioncross Solution



Brian filially gets his vaeattcn...

Now that Brian's got a handle on his exceptions, his job's going smc....hly .md he can tak.. that w.ll-deserved land I• ss-a'l..... 'wed!) vacation day.



...and things are looking up back bome!

You'r ..xc eption handling skills did mo re •han just prevent problems. They ..nsured that Brian's boss has n" idea anything went wrong in rh.. first plac..!



Good ot Brian. Never misses 0 day of work unless he's got a real problem.

Good exception handling is invisible 10 your users. The program never crashes, ao<' if there are problems, they are handled gracefully, without conlusing error messages.

• • •

What your code does when you're not looking



Your objects are starting to think for themselves.

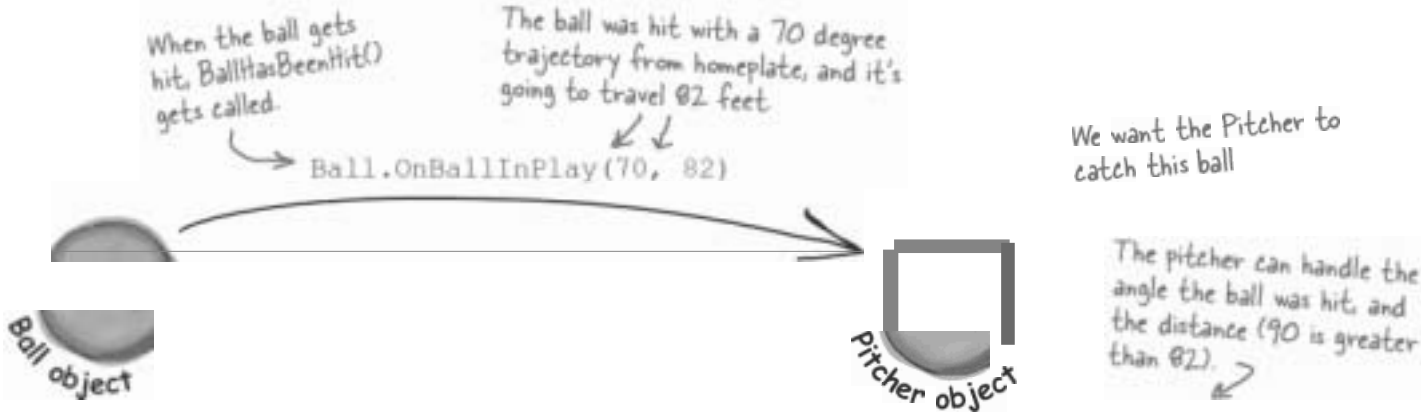
You can't always control what your objects are doing. Sometimes things _ happen. And when they do, you want your objects to be smart enough to respond to anything that pops up. And thai's what events are all about One object *publishes* an event, other objects sccscee. and everyone works together 10 keep things moving. Which is great, until you've got too many objects responding to the same event. And mere when callbacks will come In handy

Ever **wish** your objects could **think** for **themselves?**

Suppose you're writing a baseball simulator. You're going 1" model a game, sell the software 10> the Yankee (they've got deep pockets, right?), and make a million bucks. Y" ll create your Ball. Pitcher, Umpire. and Fan objects, and a whole 101 Ill"!>', You even write code so that the Pitcher object can catch a ball.

That's a standard way of naming ··ethods—we'll talk ··ore about it later

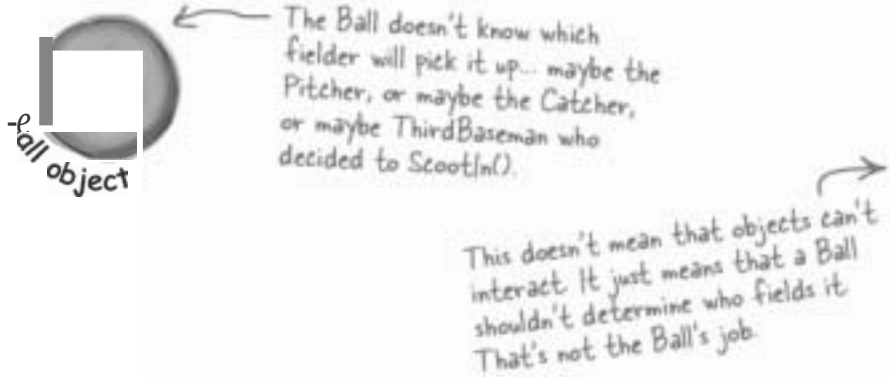
Now you just need ll> connect everything together. YOII add an Onfsa l I InP1ay() meth,d I" Ba11, and now you want your Pitcher object to respond with ;1, event handler llWlhorL Once the methods are written, vou .ill'1 need to ti,' the separate methods together:



Pitcher. CatchBall(70, 90j

But how does **an** object KNOW to **respond?**

Here's the problem. You really want your Ba11 object to only worry abc'UI getting hil. ami your Pitcher to object only worry about catching balls that come ii, way. III other words, you really don't want the Ball telling the Pitcher, -I'lll coming ttl you."



You want an ol,ject to **worry** about itself, not **other objects. You're separating the concerns of each object.**

Whell all EVENT occurs... objects Iistell

\\"hal you need to do when the ball is hil is "'t0 an **ev**pnt. •\1) event is simply ***som..thing that's happened*** in ""'II' program. 'I'hen, " Ihl"- objects can respond to Ih,,(event —like our Pitcher object.

Even better, ",,>re than one object can listen I', events. So the Pitdier could li,It'11 for a ball-being-hit event, as well as a Catcher. ThirdBaseman, an Umpire. even a Fan. ,\l1d "adl "hj",-, can respond to the ,"','11 1differently.

So what we "'al\t is a Ball object that can **rai.... an ev..nt.** Theil. we want '" have other objects '0 **subscrib..** to that **particular type** or **ev**pnl... that just ",,""" li'Wllt" it, "nd get "",;I;,-,j when that event """-111-'.

> **event, noun.**
> a **thing** that happens, especially something of importance. *The solar eclipse was all amaz inv* ***event*** *10 behotd.*



If we subscribe to the BallInPlay eveer. we'll always get notified when the ball's in pby.

W.ot to DO SOMETHING with 'Oev,ot?
You o"d .o event h.odler

Once your object "hears" about an event, you can set \11' some code to rtm, 'I'hat code is called all event handler_.\11 event handler gets mformation about the event, and runs every time thai event occurs.

Remember, all this happens ***without.your interven***'ion ;,, runtime. S" y(,u write ,'od,' t" raise all event, "",I til"")'''' write c("1,, to h;lll1l1,' Ih,,s(·,',-ents, and fire "I' your application. '11"",,, whenever an ,"-'ut;; raised, your handlers kick ;UIO act;ou_ *without you doing anything.* .'vnd. best of all. your ohj,:cts have separate concerns. They're worrying about themselves, uol other objects.

We've been doing this all along. Every time you click a button, an event is raised, and your code responds to that event

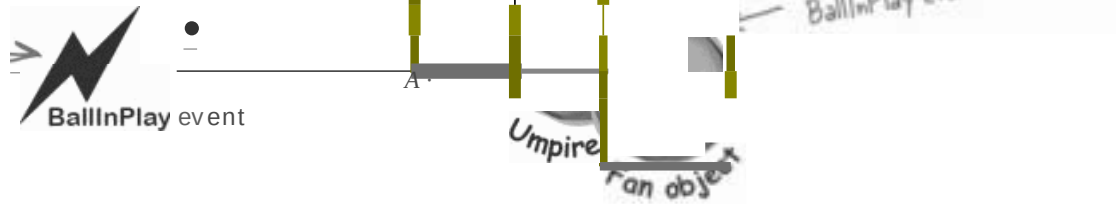
Olle object raises its evellt ethers Iistell for it...

Let' take a look at how events, event handlers, and , " h" Ti!,ti' rn works in C#:

First, other objects subscribe to the event

Before the Ba11 can raise its Ba11 rnPlay "\l'tll, other objects need to subscrib», to it. That's their way of saying anytime a Ba11 IuPLa y event ,xccurs, "" want to know ab«,nt it

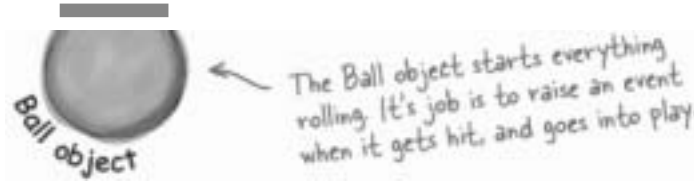
Every object adds its own event handler to listen for the event—just like you add button1_Click() to your programs to listen for Click events.



2 Something triggers an event

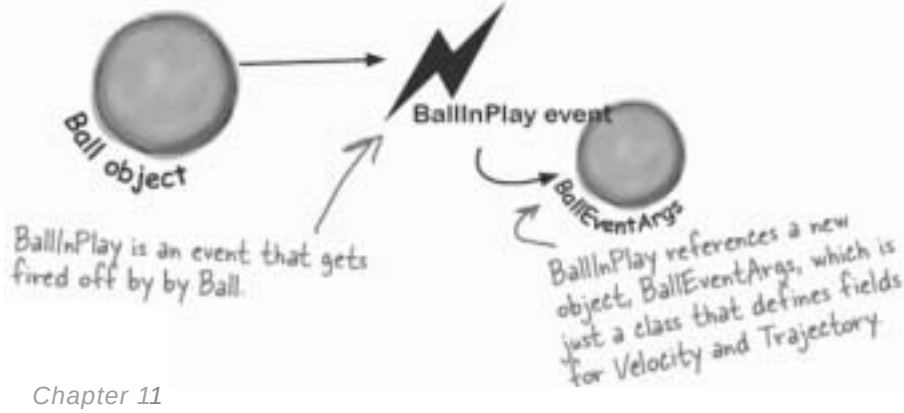
The ball gets hit It's time fo tl„' Ba l l object h> raise a new "Will ,

Sometimes we'll talk about raising an event, or firing it, or invoking it—they're all the same thing People just use different names for it.



3 The ball raises an event

.\ new event gets created (we'll talk ab«>"1 exactly how that works injn't a minnt" }. That event also has ,ome arguments, lik.. th.. velocity or th,- ball. as well as its trajectory. T h" ,.. arguments are attached to tlie event as an instance or an EventArgs object, and then th,· event is sent "Jf, available to anyone listening ro,- it.

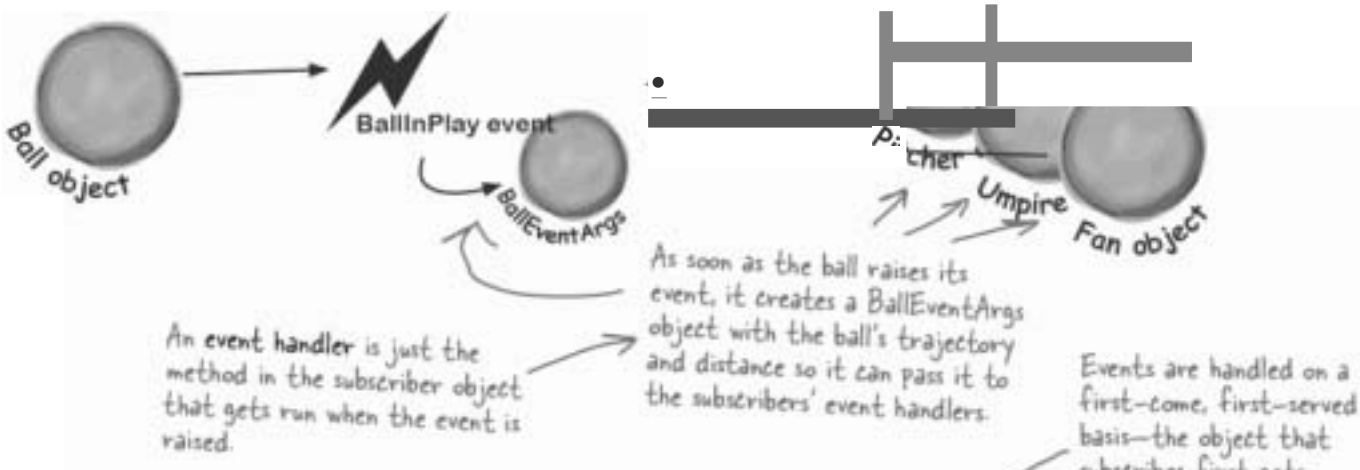


Then, the other objects halldle the evellt

()nce an event is raised, all the "1jects subscribed 1" lhal event get ll"lili'-"lioll, and can do something:

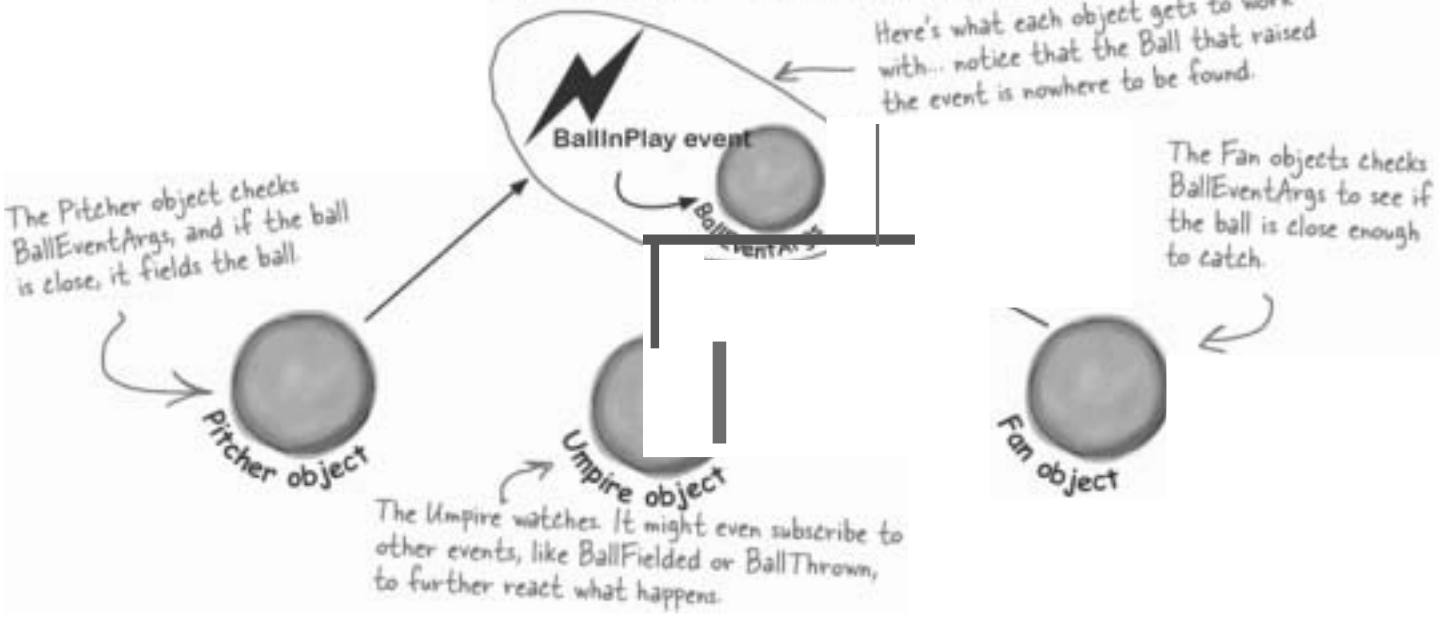
④ Subscribrs get notification

Since the `Pit.cher`, `Umpire`, and `Fan` object subscribed to lht" `Ball` object' `Ball InPlay` event, lney all get llo1ifi,:d —all or their ,"vent h"",Il"'Ill1('I],,,d, get called one after """lh,'r.



⑤ Each object handles the event

Now, `Pitcher`, `Umpire`, and `Fan` can all handle the `BallInPlay` event in their own way. But they don't all run at the same time—their event handlers get called one after another, with a reference to a `BallEventArgs` object as its parameter.



COl1l1ectil1g the dots

Now that you've got a handle on what's going on, let's take a closer look at how the pieces fit together. Luckily, there are only a few moving parts.

It's a good idea (although not required) for your event argument objects to inherit from EventArgs. That's an empty class—it has no public members.

It means that you can upcast your EventArgs object in case you need to send it to an event that doesn't handle it in particular.



We need an object for the event arguments

Remember, '","r Ba11 InPlay eve„1 has " f,•v, arguments lia,1 ;1 carrie long. So we need a very simple object for th " arguments. .NET I " standard class fe il •alled **EventArgs**, I , lh.u class **ha**.. no **me**nlb h ' rl'ose is 'n "II.",·your < nt arguments object '" 1.. ".,, 01 III tl nl handlers tha• use ;., Here's the rLass declaration:

```
public class BallEventArgs : EventArgs
```

The ball will use these properties to pass information hit to the event handlers about where the ball's been hit.

2 Next **we'll need** to **define** the event in the class that'll **raise** it

n ball rL,ss ";111a".. a li"" ";,h th.. event **yv** ord this is how it informs ."h objects .,llou" 110.. ev..Il','" they < Ill 'ul ril " i•. This line can be anvwhere in the cl ss it's usually "....r d , perty d.cl r ions. But as long as it's in the B•ll , L other objects can _ut ri",' to a ball's '11. It looks like this:

```
public event EventHandler BallInPlay;
```

Events are usually public. This event .. defined in the Ball class, but we'll want Pitcher, U..pire, etc., to be able to .eference it. You c...ld make it private if you only wanted other instances of the same class to subsc..ibe to it.

After the event keyword comes EventHandler. That's **not a reserved C# keyword**—it's defined as part of .NET. The reason r- need it is to tell the objects subscribing to the event what their event handler ...ethods should look like.

When you ..se EventHandler, you're telling other ..ethods that their event handlers need to take two parameters, an object na..ed sender and an EventArgs reference na··ed · _

The subscribing classes need event handler methods

Every object that has tn subscribe to lh,-Ball's BallInPlay event needs 10 have all "Wilt hamll,-l. V'" already know how event handlers work every time you added a method to h.mdk- a Button's Click event "r a Nunne-ric Upl),wn's ValueChanged event, the IDE added an **event handler** m..thod 1" ",'III' class. The Ball's BallInPlav event i, n" different, and an event handler for it should look pretty familiar:

```
void ball_BallInPlay(object sender, EventArgs e)
```

There's no C# rule that says your event handlers need to be named a certain way, but there's a pretty standard naming convention: the name of the object reference, followed by an underscore, followed by the name of the event.

The class that has this particular event handler method has a Ball reference variable called `ball`, so its BallInPlay event handler starts with "`ball_`", followed by the name of the event being handled, "`BallInPlay`".

The BallInPlay event declaration listed its event type as `EventHandler`, which ..eans that it needs to take two para...eters—an object called sender and an EventArgs called e—and have no return value.

Each individual object subscribes to the event

Once we've go the event handler set lip, tl,,-various Pitcher, Umpire-. ThirdBaseman, and Fan objects need 1" hook up their ',WII,""-Ill handlers. Each on,' 01 them "ilf have its own specific `ball_BallInPlay` method that responds differentlv In the r-vcnt. So if rhere's a Ball objec reference "ariable "'-field called `ball`. then the += "Ierat ,,' will hoc,k up the event handler:

```
ball.BallInPlay += new EventHandler(ball_BallInPlay);
```

This tells C# to hook the event handler up to the BallInPlay event of whatever object the ball reference is pointing to.

The += operator tells C# subscribe an event handler to an event.

This part specifies which event handler method to subscribe to the event.

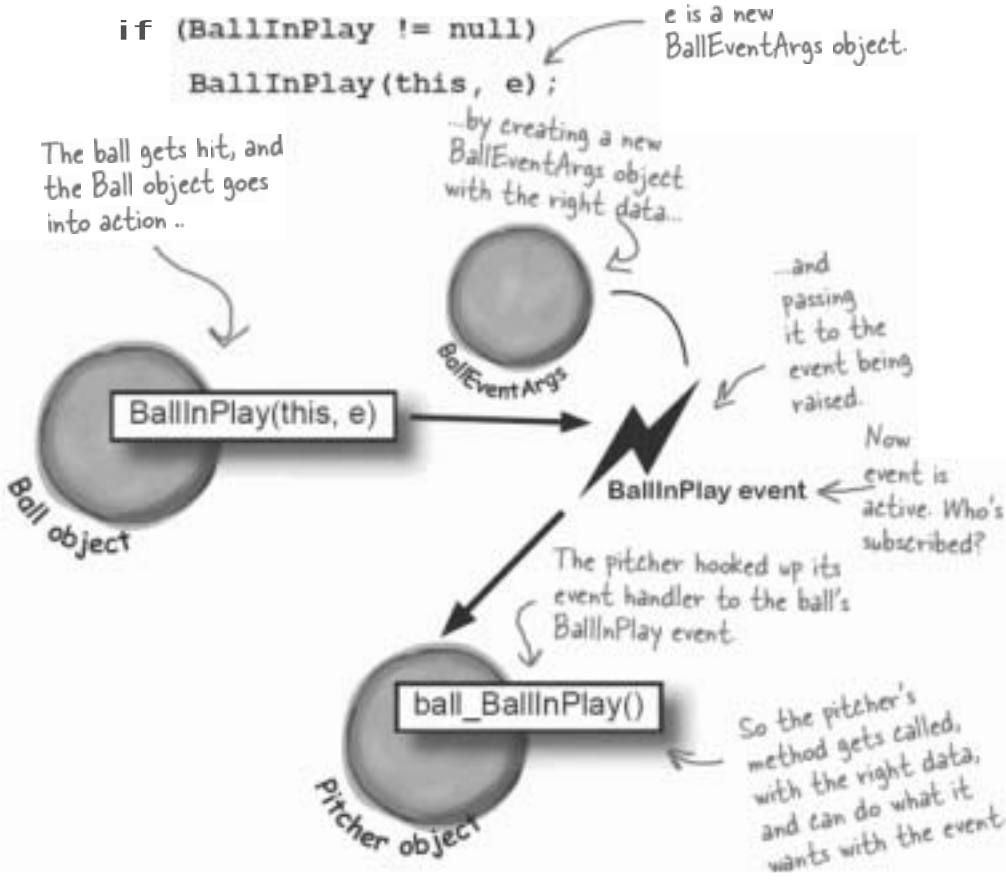
The event handler method's signature (its parameters and return value) has to **match** the one defined by EventHandler or the program **won't compile**.

Turn the page, there's a [ttie more....

5 A Ball object raises its event to notify subscribers that it's in play

Now that the events are all set up. The Ball can **raise its event** in response to something else that happens in the simulator. Raising an event is easy—it just calls the BallInPlay event.

```
if (BallInPlay != null)
    BallInPlay(this, e);
```





Watch it!

If you raise an event with no handlers, it'll throw an exception.

If no other objects have added their event handlers to an event, it'll be null. So *always check to make sure your event handler* isn't *equal to null before you raise it. If you don't, it'll throw a NullReferenceException.*

Use a standard name when you add a method to raise an event

Take a minute and go the code for any form and type the keyword override any place you'd declare a method. As soon as you press space, an IntelliSense window pops up:



There are a huge number of events that a Form object can raise, and every one of them has its own method that raises it. The form, OnDoubleClick() raises the DoubleClick event, and that's the whole reason it's there. So the Ball event will follow the same convention: we'll make sure it has **a method called** OnBallInPlay that takes a BallEventArgs object as a parameter. The baseball simulator will call our method any time *it* needs the ball to raise its BallInPlay event—so when the simulator detects that the bat hit the ball, it'll create an instance of BallEventArgs with the ball's trajectory and distance and pass it to OnBallInPlay().

there are no Dumb Questions

Q_: Why do I need to include the word EventHandler when I declare an event? I thought the event handler WII what the other objects used to subscribe to the events.

A: That's true when you need 10 subscribe 10 an event, you write a method called an event handler But did you notice how we used `EventHandler` III the the event declaration (step #2) ***and*** the III line 10 subscribe the event handler 10 it (step #4)? What `EventHandIer` does is it defines the **signature** of the event—it tells the objects subscribing 10 the event exactly how they need 10 define their event handler methods Specifically, it says that II you want 10 subscribe a method 10 this event, it needs to take two parameters (an `object` and an `EventArgs` reference) and have I `void` return value

Q: What happens if I try to use I method that doesn't match the ones that art defined by EventHandler?

A: Then your program won't compile The compiler will make sure that you don't ever accidentally subscribe an incompatible event handler method to an event. That's why the standard event handler, `EventHandler`, is sc useful—as soon as you see it, you know exactly what your event handler method needs 10 look like

Q_: Wait, "standard" event handler? Th. " I. other kinds of event handlers?

A: Yes! Your events don1 ***have*** to send an object and an EventArgs In fact, they can send anything at all—or nothing at all! Look at the last line III the IntelliSense window on the bottom on the facing page Notice how the OnDragDrop method takes a DragEventArgs reference instead of an EventArgs reference? DragEventArgs inherits from EventArgs, just like BallEventArgs does. The form's DragDrop ~~event doesn't use~~ EventHandler It uses ....nething eee . DragEventHandler, and if you want to handle rl your event handler method needs to take an object and a DragEventArgs reference

The parameters of the event are defined by something called a *delegate*—EventHandler and DragEventArgs are two examples of delegates. But we'll talk more about that in a minute

Q: So I can probably have my event handlers return something other than void, 100. right?

A: Well, you can, but it s often a bad idea If you don't return void from your handler, you can't *chain* event handlers That means you can't connect more than one handler to each event Since chaining is a handy feature, you'd do best to always return void from your event handlers

Q_: Chllning? **What's that?**

i : It's how more than one object can subscribe 10 the same event—they chain their event handlers onto the event, one after another. W.'llalk a lot more about that In a minute, too

Q: Is that why I used += when when I added my event handler? lik,l'm somehow adding I new handler 10 existing handlers?

A: Exactly! Anytime you add an event handler, you want to use . –. That way your handler doesn t replace existing handlers It just becomes one III what may be • very long chain of other event handlers. II of which are listening to the same event

Q: Why does the ball use "this" when it raises the BallInPlay() , venI?

A: Because that s the first parameter of the standard event handler Have you noticed how • very Click event handler method has a par "object sender"? That parameter is a **reference** to **the object that's raising the event** So if you're handling a button click, `sender` points to the button that was clicked And rl you re handling a BallInPlay event, `sender` will point to the Ball object that's in play—and the ball sets that parameter to `this` when it raises the event

A SINGLE event is always raised by 3 SINGLE object.

Bul 3 SjNGLE event can be responded 10 by MULTIPLE objects.

Ihe IDE creates evellt handlers for you autolllatically

Most programmers follow the same convention li,1' naming their event handlers. If there's a Ball object that has a Ba111nPlay event, and Iht·l1«mt· "r the reference holding the objec l is called ba11. then the ,"'t'ul handler would typically be named ba11 Ba11InPlay() Ihat's nol « h,,,.1',mtl·f",1 rul., but rl vou wrnte ,,,"' c«..I" like that, it'll be a l" t easier l;>r other programmers to read.

Luckilv; the IDE makes il reallv easy 10name vour event handlers properly. It has a feature that **automatically adds event handler methods** for **you** when vuurr-working wit h a class tha1raises an event. 1\ sh«'llldll'I be too surprising 1hat the 11 11-: t"'" d" this for you - after all, this is exactly what it does when you double-click "'I a Inmon in your form.



O Start a **new Windows** application and add **the** Ball and BaliEventArgs Here's the 11all class:

```
public class Ball {
    public event EventHandler BalllnPlay;
    public void OnBallInPlay(BaliEventArgs e)
        if (BalllnPlay != null)
            BalllnPlay(this, e);
```

:\lld here's the BallEventArgs class:

```
public class BallEventArgs : EventArgs {
    public int Trajectory { get; private set; }
    public int Distance { get; private set; }
    public BallEventArgs(int Trajectory, int Distance)
        this.Trajectory - Trajectory;
        this,Distance - Distance;
```

O Start adding **the** Pitcher's constructor

:\tld a new Pitcher class to your project. Then give il a c«,,,,lnKI,»r that takes a Ba11 reference called ba11 as a parameter. Thl'n' will be one Jim' "r c«..I;> ill the <oll,lru"[r)r to add its event handler [r) ball . BalllnPlay. Start typing tln- statement, hUI don't **type += yet**.

```
public Pitcher(Ball ball)
    ball.BalllnPlay
```

3 Type += and **the IDE will finish** the statement for you

,\, soon as you typ(+= intln- statement, the IDE displays a very useful little box:

```
public Pitcher(Ball ball) {
    ball.BallInPlay .-
                    Inl·w EventHandler(ball_BallInPlay);   (Press TAB to insert)
```

As soon as you pres the tab key, the IIlE will lini,h the statement for you. It'll look like this:

```
public Pitcher(Ball ball) {
    ball.BallInPlay += new EventHandler(ball_BallInPlay);
```

When you double-click on a button in the form designer, the IDE does the exact sa..e trick—adding an event handler automatically—except that it adds the code to the for..'. InitializeComponent() ..ethod in the Forml. Designer.cs file instead of just adding it to the end of the class file.

O **The** IDE **will** add **your event** handler. too

You're "01 done- '"'II ,Iill ", ...d to "dd a method to chain ",Ut> the event. Lu<kill', Ih" IHE l"k," care ,,f that fi> you, t<••.

```
new EventHandler(ball_BallInPlay);
                 Press TAB to generate handler 'ball_BallInPlay' n this class
```

II il lhe tab key again I" make the IIlE add this event haml,'r lll,'lho,1 to youu Pitcher class. The IDI ",ll alwavs follow the objectName_HandlerName() convention

```
void ball_BallInPlay(object sender, EventArgs e) {
    throw new NotImplementedException();
```

The IDE always fills in this Notl..ple..entedException() as a placeholder so if you run the code it'll throw an exception that tells you that you still need to imple..ent something it filled in automatically

5 **Finish the** pitcher's event hondler

Now that you've got the event handler's skeleton added ,,, your class, fill in the rest or its code. The pitcher should catch any low balls, otherwise h,' covers first base.

Since BallEventArgs is a subclass of EventArgs we'll downcast it using the as keyword so we can use its properties

```
void ball_BallInPlay(BallEventArgs e)
    if {e is BallEventArgsl {
        BaUEventArgs ballEventArgs = e a" Bal lEventArgs;
        if «ballEventArgs.Distance < 95) && (ballEventArgs,Trajectory < 60))
            caccnnaL; u •
        else
            CoverFirstBase(J;
```

You'll add these methods ill a minute.



It's time to put what you've learned so far into practice. Your job is to complete the Ball and Pitcher classes, add a Fan class, and make sure they all work together with a very basic version of your baseball simulator.

0 Complete the Pitcher class.

Below is what we've got for the Pitcher. Add the CatchBall() and CoverFirstBase() methods. Hint: should print "" that the catcher has either caught the ball, or run to first base.

```
public class Pitcher
    public Pitcher(Ball ball) {
        ball.BallInPlay += new EventHandler(ball_BallInPlay);


    void ball_BallInPlay(object sender, EventArgs e) {
        if (e is BallEventArgs){
            BallEventArgs ballEventArgs = e as BallEventArgs;
            if ((ballEventArgs.Distance < 95) && (ballEventArgs.Trajectory < 60))
                CatchBall();
            else
                CoverFirstBase();
```

You'll need to implement these two methods to write a line of output to the console





0 Write a Fan class.

Create another class called Fan. Fan should also hook up to the BallInPlay event in its constructor. The fan's event handler should see if the distance is greater than 400 feet and the trajectory is greater than 30 (a home run), and grab for a glove to try and catch the ball if it is. If not, the fan should scream and yell. Write out what's going on with the fan to the console.

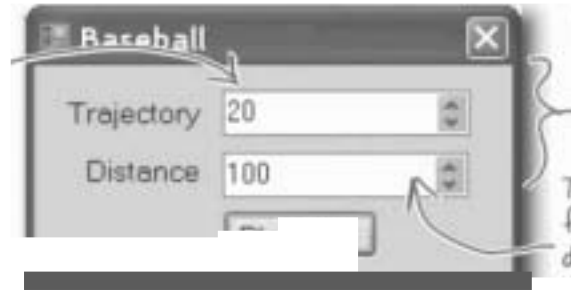
Look at the output window on the facing page to see exactly what it should print.



o **Build a very simple simulator.**

Create a new application. The application should have two NumericUpDown controls: one for the ball's distance, and one for its trajectory. Add a button, labeled "Play ball!" When "Play ball!" is clicked, a ball is hit with the values in the two NumericUpDowns. Your form should look something like this:

The value for trajectory can range from 0 to 100, so set its Minimum property to 0, Maximum to 100 and Value to 20.

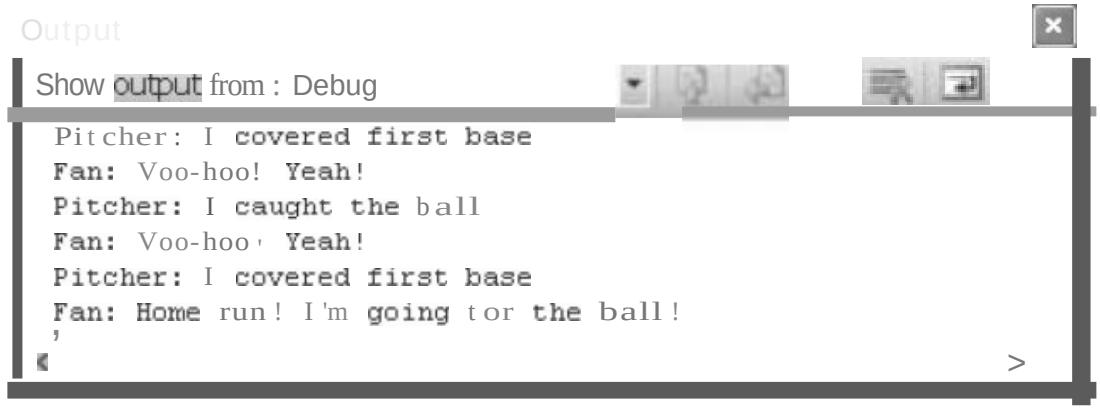


Don't forget to cast the Value properties to ints before you use them.

The Distance can range from 0 to 500, with a default value of 100.

e **Create the following output.**

Sure, if you can't make your simulator generate this output with multiple successive balls just right, play. Write down the values you used to get the result below:



Ball 1,
Trajectory:
Distance:

Ball 2,
Trajectory:

Ball 3:
Trajectory:



It's time to put what you've learned so far into pracuce. Your job is to complete the Ball and Pitcher classes, add a Fan class, and make sure they all work together with a very basic version of your baseball simulator

```
public class Ball
{
    public event EventHandler BallInPlay;
    public void OnBallInPlay(BallEventArgs eJ

        if (BallInPlay != null)
            BallInPlay(this, e);
```

The OnBallInlay() method just raises the BallInPlay event—but it has to check make sure it's not null, otherwise it'll throw an exception.

Read-only automatic properties work .eally well in event a..gu...ents because the event handlers only read the data passed to the...,

```
public class BallEventArgs {
(
    public int Trajectory { ger; private set; )
    public int Distance { get; private set; )
    public BallEventArgs(int Trajectory, int Distance)

        this.Trajectory _ Trajectory;
        this.Distance = Distance;
```

The Fan object's constructor chains its event handler onto the BallInPlay event.

```
public class Fan {
    pUblic Fan(Ball ball)
    {
        ball.BallInPlay += new EventHandler(ball_BallInPlay);

    void ball_BallInPlay(object sender, EventArgs e)
    {
        if (e is BallEventArgs)
            BallEventArgs ballEventArgs = e as BallEventArgs;
            if (ballEventArgs.Distance > 400 && ballEventArgs.Trajectory > 30)
                Console.WriteLine("Fan: Home run! I'm going for the ball!N);
            else
                Console.WriteLine(..Fan: Woo-hoG! Yeah!");
```

The fan's BallInPlay event handler looks for any ball that's high and long

```
public class Pitcher {
    public pitcher(Ball ball) {
        ball.BallInPlay += new EventHandler(ball_BallInPlay);

    void ball_BallInPlay(object sender, EventArgs e) {
        if (e is BallEventArgs) {
            BallEventArgs ballEventArgs = e as BallEventArgs;
            if ((ballEventArgs.Distance < 95) && (ballEventArgs.Trajectory < 60))
                CatchBall();
            else
                CoverFirstBase();


    private void CatchBall() {
        Console.WriteLine("Pitcher: I caught the ball");

    private void CoverFirstBase() {
        Console.WriteLine("Pitcher: I covered first base");
```

You already have the pitcher's BallInPlay event handler. It looks for any low balls.

```
public partial class Form1 : Form {
    Ball ball = new Ball();
    Pitcher pitcher;
    Fan fan;

    public Form1()
        InitializeComponent();
        pitcher = new Pitcher(ball);
        fan = new Fan(ball);

    private void playBall_Click(object sender, EventArgs e) {
        BallEventArgs ballEventArgs = new BallEventArgs(
            (int)trajectory.Value, (int)distance.Value);
        ball.OnBallInPlay(ballEventArgs);
```

The for.. needs one ball, one fan, and one pitcher. It hooks the fan and pitcher up to the ball 'n its constructor

When the button's clicked, the form tells the pitcher to pitch the ball to the batter, hich tells the ball to fire off its BallInPlay event, hich calls the event handlers ;ll the pitcher and fan objects.

Here are the value° we used to get the output. Yours ..ight be a little different

Ball 1:
Trajectory: 75
Distance 105

Ball 2:
Trajectory: 48
Distance: 80

Ball 3:
Trajectory: 40
Distance 435

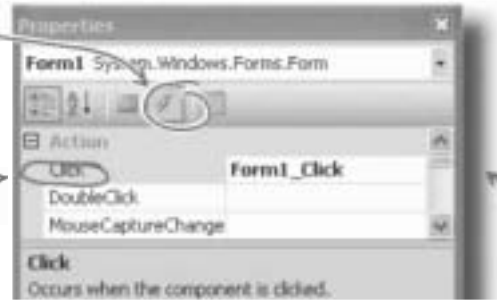
Ihe forms you've been building all use events

Every time you've created a hllllon, double-clicked "n il in the designer, and written ,ode for a method lik,' `button1_Click()`, vou've been working with events.

Do this

O Create a JIC\\ Windows Application project. Bring up lh" form, and go 10the Properli,·, " indow. The..., are ic‹,"' at Ih‹ top of the windo\ click on lh,' " n,' that's got a lightning 1",11 icon 0" il. ThaI will bring " I' th.. ..**v**..n**ts page** in the **Properties** window:

You can see all of the pre-..ade events for a control by click on it and then click on this events button in the prope..ities window



You can create an event that will fire every ti..e someone clicks on the form by selecting Form1_Click next to Click in the events window

Sc·oll down to CI,tk and double-click on the word "Click". When you do, the IDE w,ll add a new click event handler to you.. form that get• fired every ti..e you click on it. And it'll add a line to Form1.Designe..cs to hook the event handler up to the event

O Double-click "Il the "Click" row in the events page. The !l1E will automatically add an event handler method t, . vour r"nn called Form I Cli,-k Add llti, line " I' c‹..le to il:

```
private void form1_Click(object sender, gventArqs e)
    MessageBox.Show("You just on clicked the form");
```

3 Visual Su"li" did more Ihan j"'1 write a lin I" "I<'lho,1 declaration for you, though. II also hooked Ill,' event handler up '" Ih,' Form objects Click event. Open "I' l'; ,n n l.Designer.cs and use the Quick Find feature ill the [n E 10search for the text `form1_Click` in the r-urn-nt project. You'll find lhi, line "I' code:

```
this.Click += new System.EventHandler(this.Form1_Click);
```

Now run the program and make sure your code works!



Watch **it!**

Event handlers always need to be "hooked up".

If you drag a button onto your form and add a method called `button1_Click()`, *that has the right parameters. but isn't registered to* ***listen*** *to your button. the method won't ever get called. Double-click on the button in the designer- the IDE will see the default event handler name is* ***taken****, so it'll add an event handler for the button called* `button1_Click_1()`.

One event, multiple handlers

Here's a really useful thing that you can do with events: you can chain tlu-m s(that one "\""UI or delegat(calls many methods, one after another. Let's add a few buttons 10your appli'-"Ii,," 10see how il works.

O Add Ih,"" two methods 10your form:

```
private void SaySomething lobject sender, EventArgs e)
    MessageBox.Show("Something");
}
private void SaySomethingElse(object sender, EventArgs e) {
    MessageBox.Show("Something else");
```

O Now add two bUI1<>I1_' "' your form. 1Double-click on each Imllo'l It> add ii, event handler. Here's the code for holh event handlers:

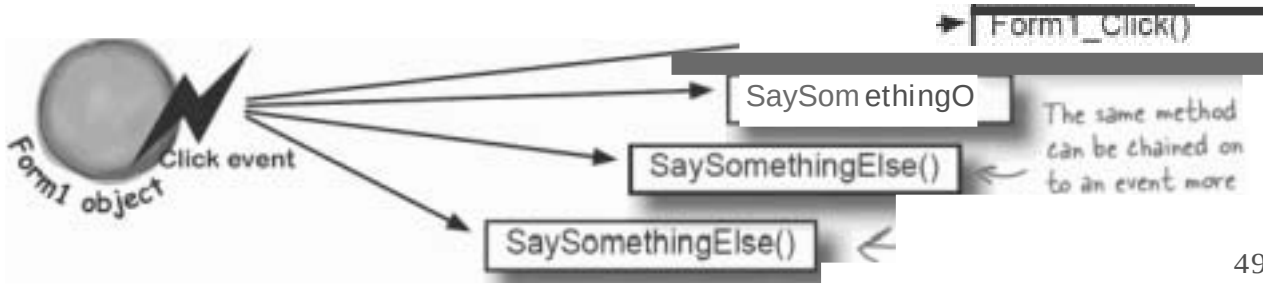
```
private void button1_Clicklobject sender, EventArgs e) {
    this.Click +=new EventHandler(SaySomethingJ;
}
private void button2_Click(object sender, EventArgs e)
    this.click -r- nev EventHandler(SaySomethingElse):
```

O Now run your program 'liul do Ihi"

- Click th.. form you'll see a message box I'"p 11]1 Ihat says, "You just clicked " 'j lh,' I" r"'·'.
- Now **click** button) and then **click on** the form **again**. You'll see two message boxes pop up: "You jU'1 clicked on the form" and then ..s, mething".
- Click button2 twic.. and II " dick on th.. form again_You'll see four message boxes: "You just clicked on the f(",,," s,mething", "Something else", and "Something else".

So what happened?

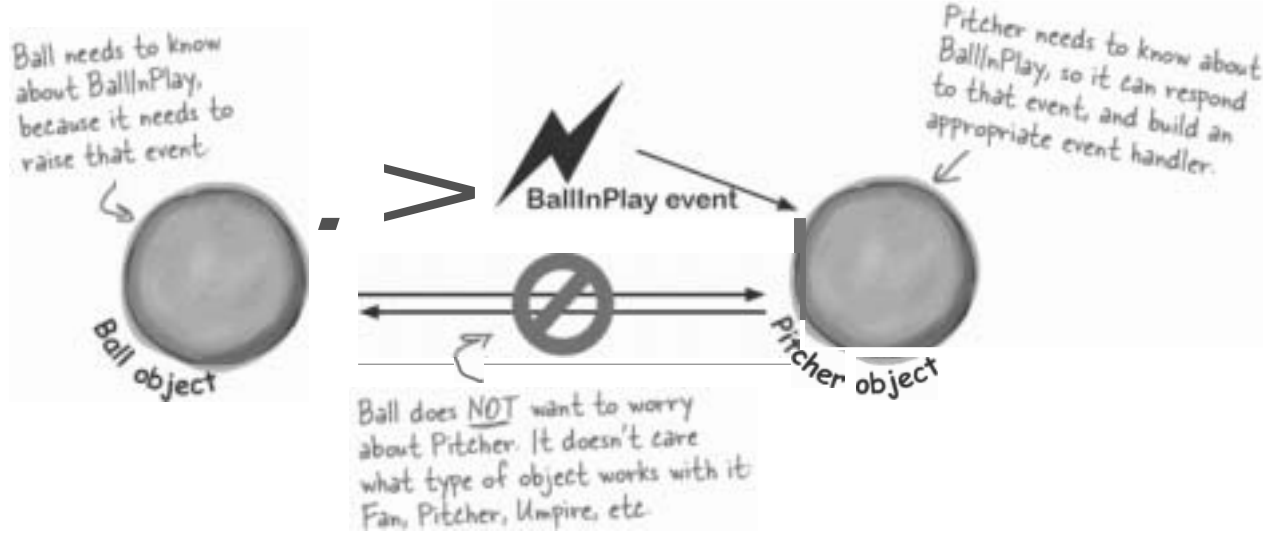
Every time you clicked one "I' the buttons, you chained another method—either Something() or SomethingElse()—onto the f('11"'s eli,·k event. You can keep clicking the buttons ,,,,,I they'll 11" "1' chaining the **same m**..thods onto the event. T h,' event doesn't care how many methods are ,'hai,wd " 11. "r even if the same method is ill the chain more than once. h'll j"' 1call Ihem all every tinu-the event fires, one after another, ill the ord" r they were added.



Connecting event senders with event receivers

One of the trickiest things ahout events is that the **s..nder** of the ,·,·,·tH has 10 know what kind of event (" send— including the arguments (" pass to the event. And the **rec..iver** "r the event has 10 know about the return typ‹ and the arguments its h"llll,·,. methods must ",",

But and here's the tricky part you ("au't ti,· the sender and receiver *together*. You want the sender to send the event and *no/ worry about who receives* ii, ,\11<1 the receiver cares abɔun (h(" event, *not III/f object that raised the event*. So bɔ,(h sender and receiver foɔu, ,'ı' th,' event, no't each "ther.



"My people will get ,. touch with your people:

You know what this code does:

```
Ball currentBall;
```

It creates a reƒerence variah le that can poinl 10 any Ball object. It's no(ti,'d to a single Ball. Instead. it can poillllll any ball object—or it ,-,on be null. and "tlt po int to anything at all.

A\ n event needs a similar kil\d "I reference except instead of pointing to an obj-ct, it needs one Ihat **points to** a **method**. Every event needs (0 k,"'p track of a list or methods that are subscribed to it. You\ -,' already see·u that rtıey can ɔe ill o(hn classes, and they can even ɔe private. So how does it keep track or all of the event handler methods that it needs to call?

A delegate STANDS IN for an actual method

When you create a delegate, all you need to do is specify the signature of methods that it can point to.

So this delegate can be used to reference any method that takes an object and an EventArgs and has no return value.

One of the most useful aspects of events is that when an event listener has no idea whose event handler methods it's calling. Anyone who happens to subscribe to an event gets their event handler called. So how does the event manage that?

It uses a C# type called a **delegate**. A delegate lets you create a reference variable, but instead of referring to an instance of a class, it **refers to a method inside a class.**

You've actually already been using delegates throughout this chapter! When you created the BallInPlay method, you used EventHandler. Well, an EventHandler is a delegate. If you right-click on EventHandler in the IDE and select "Go To Definition", this is what you'll see (try it yourself):

```
public delegate void EventHandler(object sender, EventArgs e);
```

This specifies the return value of the delegate's signature—which means an EventHandler can only point to methods with void return values.

The name of this delegate is EventHandler.

Do this

A delegate adds a new type to your project

When you add a delegate to your project, you're adding a **delegate type**. And when you use it to create a field or variable, you're creating an **instance** of that delegate type. So **create a new project**. Then add a new class file to the project called ReturnsAString.cs. Instead of putting a class in it, add a single line:

```
public delegate string ReturnsAString();
```

ReturnsAString is a delegate type that you've added to your project. Now you can use it to declare variables and fields.

Go to the form code and add this line to the form:

```
ReturnsAString someMethod;
```

someMethod is a variable of the delegate type ReturnsAString.

Now build your program - it compiles! (It gets a warning because you never used that field - that's okay.) As soon as you added your new delegate to the program, it created a new type called ReturnsAString. If you use that type to declare a variable, you can set that variable equal to any method that takes no parameters and returns a string. Try it out—add this method to your code:

```
private string HiThere() {
    return "Hi there!";
}
```

This method's signature matches ReturnsAString.

Add a button that has these three lines. Click it and see what happens:

```
someMethod = new ReturnsAString(HiThere);
string message = someMethod();
MessageBox.Show(message);
```

You can set someMethod just like any other variable. But when you call it like a method, it calls whatever method it happens to point to.

del-e-gate, noun.
a person sent or authorized to represent others. *The president sent a delegate to the summit.*

Delegates in action

Ilu-n-'s nothing mysterious about delegate - in fact, they don't take much code at all to u-c. Let's use ru-m 10 help a restaurant owner sort out hi, I"!, chef's secret ingredients.



o **Create a new Windows project and add a delegate**

1Delegates usually appear outside "j" any other classes, so add a new class lil,- I" your l'r"j""l and call il AddSecretIngredient.cs. It will have exactly one lim- "r c..de in it:

```
public delegate string GetsecretIngredient(int amount);
```

'1l1i, delegate can be used 1" create a variable that can p"inl l" any method that takes one inl parameter and returns a string.

o **Add o etess for the first chef. Suzanne**

Suzanne.cs will lr"ld a class Ih"l keeps track "l" II", fir'l chef's secret ingredient. It has a private u,,-th,)d called SuzannesSecretIngredient() with a signature that matches GetSecretIngredient. But it also has a read...nly property and check " m that property's type. It retur u, a GetSecretIngredient. S,) other objects can use that property 1" get a reference to her SUE,nnesIngredientList() nl"lll<)d.

Suzanne's secret ingredient method takes an int called a..ount and returns a string that describes her secret ingredient.

```
public class Suzanne {
  public GetSecretIngredient MySecretIngredientMethod {
    get {
      return new GetSecretIngredient(SuzannesSecretIngredient);
    }
  private string SuzannesSecretIngredient(int amount)
    return amount.'roString() + " ounces of cloves";
  }
```

e **Then add a class for the second chef, Amy**

Amy's method works a lot like Suzanne's:

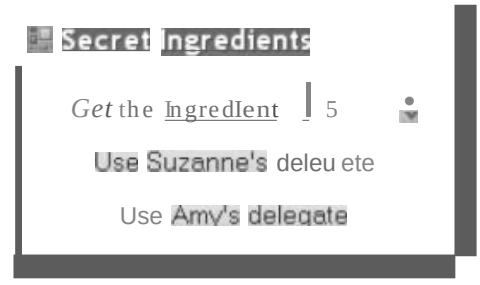
Amy's GetSecretIng..edient property returns a new nstance of the GetSecretIngredient delegate that's pointing to her sec..et ingredient ..ethod

A.y's secret ingredient method also takes an int called a..ount and returns a string, but it ..eturns a different string from Suzanne's.

```
public class Amy I
    public GetSecretIngredient AmysSecretIngredientMethod {
        get {
            return new GetSecretIngredient(Amys SecretIngredient);
    }
    private string AmysSecretIngredient(int amount) {
        if (amount < 10)
            return amount.'roString()
                    + " cans of sardines -- you need more!";
        else
            return amount.ToString() + " cans of sardines";
```

O **Create** ((**new** Windows **project** and (ldd a **delegate**

Build this form.



Here's the code for the form:

```
GetSecretIngredient ingredientMethod = null:
Suzanne suzanne = neW Suzanne();
Amy amy = new Amy();

private void uselngredient_Cl1ck(object sender, EventArgs e)
    if (ingredientMethod != null)
        Console.WriteLine("I'll add " + ingredientMethod((int)amount.Value}!;
    else
        ConscIeWrLteLi ne("I don't have a secret ingredient!H):


private void getSuzanne_Click(object sender, EventArgs e) {
    ingredientMethod - new GetSecretIngredient(suzanneoMySecretIngredientMethod):


private void getAmy_Click(object sender, EventArgs e) {
    ingredientMethod = new GetSecretIngredient(amyoAmysSecretIngredientMethod):
```

5 **Use the debugger** to **explore how delegates** work

You've got a great tool—the IDE's debugger—that really help you get a handle "" how delegates wo.k, Do the following steps:

- Start by running your program. Firs di,'k the "Get the ingredient' hUll"") i, ,I"m)d "Tit" a line t" the console that says, "I r1"n', have a secret ingr-dienr."
- Click ,h,' "Use Suzanne's delegate" bUI1"n- that takes the form's ingredientMethod field (which is a GetSecretIngredient delegate) and set i, "'lu,l[to whatever Suzanne's GetSecretIngredient property returns. That property returns a new instance "I' th(' (;etSecretIngredient typ(that's I"';Il'ing I" the SuzannesSecretIngredient() method.
- Click th(..(;", the ingredient" button again. Now that the form's ingredientMethod field is pointing to SuzannesSecretIngredient(), il calls that, passing i, the value illlh" "uIll,'ri,'UpDow|| control and writing i" "U'pUl to the cons(",
- Cli,'k the "Use Amy's delegate" hll""" It uses the Amy.GetSecretIngredient property to set the f(urn'singredientMeth("l field "' p'iinr"' the ·AmysSecretIngredient() meth,d.
- Click the ..(;et the ingred;n lt" llre,1",,1 ···r(· ····ore time. Now i, calls Amy's method.
- ★ Now **use** th.. d..hugg..r to se(exactly what's going (", Place a breakp(,i", 0" the first line of each of the three methods in the form. Then r**estart** th.. **program** (which resets the ingredientMethod ,,, that it's equal to nulh. ;1ll(1 ,,,;,rt over with the above fiv(steps. Use the S"'p Ill," rFII) featur("r the debugger '" step through every line "r code. Watch what happens when you click "(;", the ingredient". It steps right i",,, ,h" Suzanne ;uu[Amy classes, depending 0" which method the ingredientMethod li,'ld is pointing '0,

Pool puzzle



Your *job* is to take snippets from the pool and place them into the blank lines in the code. You can use the same snippet more than once, and you won't need to use all the snippets. Your *gool* is to complete the code for a form that writes this output to the console when its buttonl button is clicked.

Output

```
Fingers is coming to get you!
```

```
public Forml() {
    InitializeComponent():
    this.        += new EventHandler(                )
    thiso        += new E:ventHandler('              );

void Towtruck(object sender, EventArgs e} {
    Console.Write("is corning ");

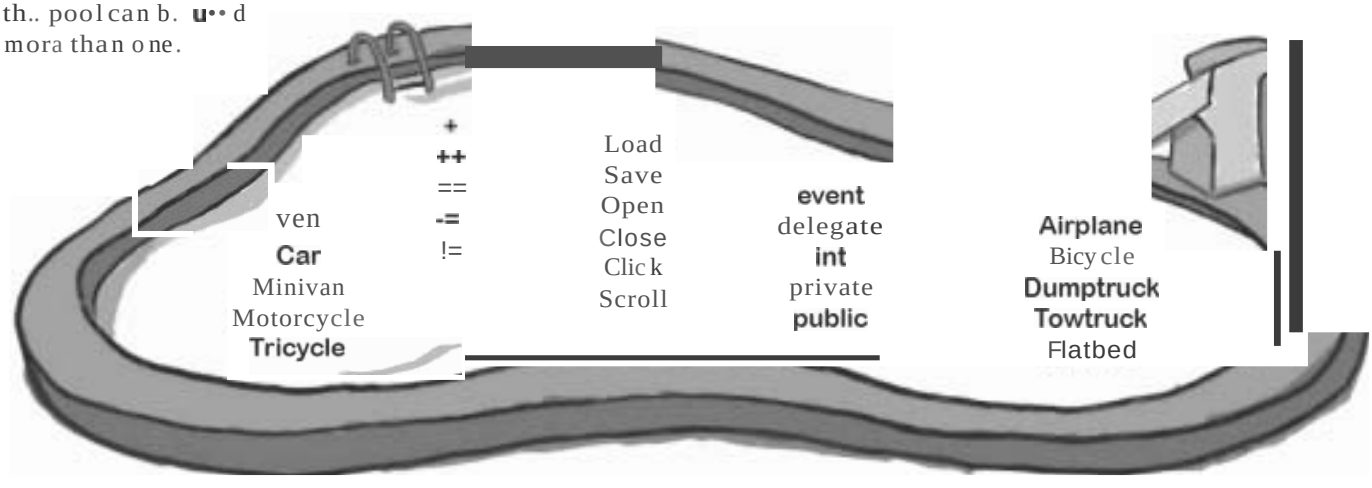
void Motorcycle(object sender, EventArgs e)
    buttonlo     _ +- new EventHandler(              "

void Bicycle(object sender, EventArgs el
    consoje.wrLtet.Inetvto get you!");

void ___________{object sender, EventArgs e)
    buttonl.     _ t_ new EventHandler(Dumptruckl:
    buttonlo     _ +- new EventHandler(              "

void ___________Iobject sender, EventArgs e)
    Console.Write("Fingers "):
```

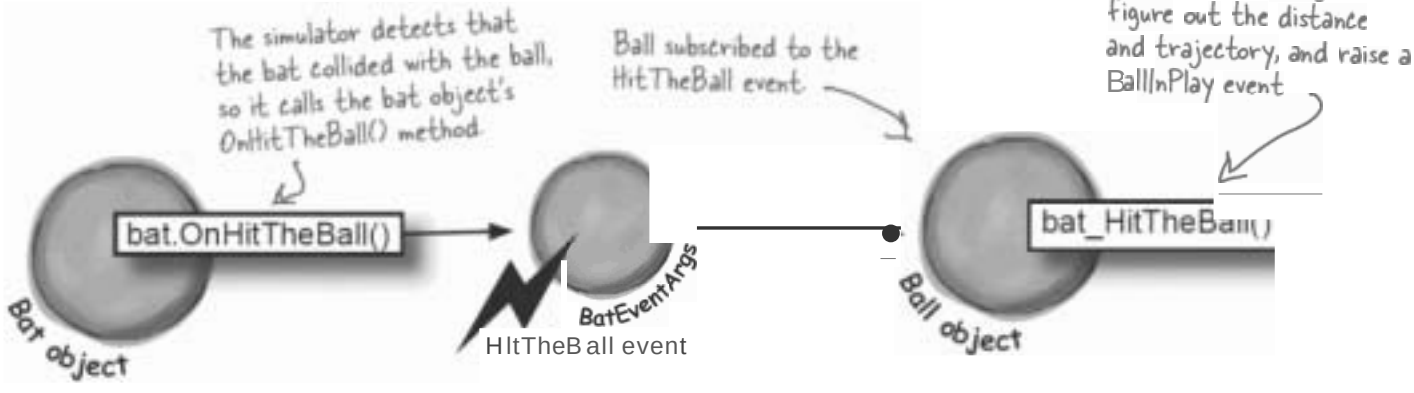
Not..:. ach thing from th.. poolcan b. **u••** d mora than one.



Any object can subscribe to a public event...

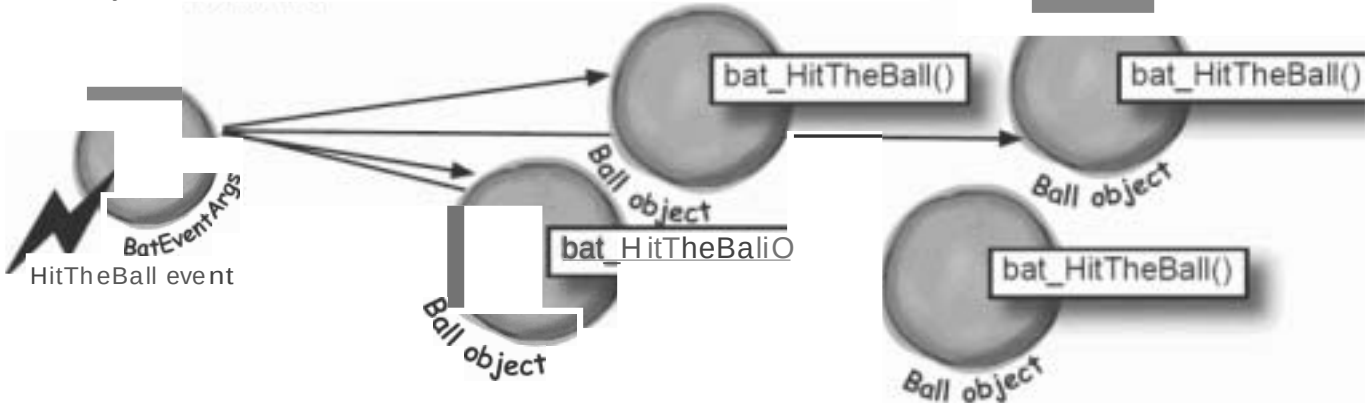
Suppose we add a new class to our simulator, a Bat class, and that class adds a HitTheBall event into the mix. Here's how it works: if the simulator detects that the player hit the ball, it calls the Bat object's OnHitTheBall() method, which raises a HitTheBall event.

So now we can add a bat_HitTheBall method to the Ball class that subscribes to the Bat object's HitTheBall event. Then when the ball gets hit, it, own event handler calls its OnBallInPlay() method to raise its own event, BallInPlay, and the chain reaction begins. Fielders field, fans scream, umpires yell... we've got a ball game.



...but that's not always a good thiog!

There's only ever going to be one ball in play at any time. But if the Bat object uses an event to announce to the ball that it's been hit, any Ball object can subscribe to it. And that means we've set ourself up for a nasty little bug—what happens if a programmer accidentally adds three more Ball objects? Then the batter will swing, hit, and four different balls will fly off into the field!



Five Minute Mystery

The Case of the Golden Crustacean

Henry "Flatfoot" Hodgkins is a TreasureHunter. He's h"l on the trail "j" one "r the most prized possessions in the rare and unusual aquatic-themed jewelry markets: a jade-encrusted translucent gold crab. BUI s(are lots "j" other TreasureI 1lll1tn " They all got a reference to the same ('Iah ill their constructor. Inn Henry wants I,. claim the prize ***first***.



In a stolen set of class diagrams, Henrydiscovers that the GoldenCrab class raises a RunForCover event every time anyone gets close to it. Even better. the event include NewLocationArgs, which d, ·t"il where the crab is moving 1", But no",' "r I],e other treasure Inlluters know abc'UI the t-I'ent, s(Henry figures he ,'all cash in.

H,',,,) , adds c(,d,- to hi, constructor I,. register hi, 1i, lt-"ForClues() meth(,£I as an event haudk-r 1;,l' the- Rllnl"'l'C'"'"r event. on the crab reference he's got. Then, he sends a lowlv underling after the crab, knowing il will run away, hid". and raise the RunForCover """"lll– giving Henry's ListenForClues() method all the information he needs.

Everything goes according to pl."" unt il Henry gets the new I,cation and rushes to grab the crab. H,–, stunned to see three other TreasureHunters already there, fighting over the crab.

How did the other treasure hunters beat Henry to the crab?

a Answers on page 511.

Pool Puzzle Solution



```
pUblic Formi() {
    InitializeComponent();
    this.Load += new EventHandler(Minivan);
)   this.Load += new EventHandler(Motorcycle);

void Towtruck(object sender, EventArgs e) {
    Console.Write("is coming ");

void Motorcycle(object sender, EventArgs e) {

    buttonl.Click +- new EventHandler{Bicyclei ;

void Bicycle(object sender, EventArgs e) {
    Console.WriteLine("to get you!");
}
void Minivan(object sender, EventArgs e)
    buttonl_Click +_ new EventHandler(Dumptruck);
    buttoni.Click += new EventHandler(Towtruck);
|
void Dumptruck(object sender, EventArgs e)
    Console.Write("Fingers ");
```

The constructor chains two event handlers onto the load events They get fired off as soon as the form is loaded

When the button is clicked, it calls the three event handlers that are chained to it.

The two Load event handlers hook up three sepa.ate event handlers to the button's Click event handler

Use a callback illstead of an evellt to hook up exactly one object to a delegate

(lur svstem of events "nly works if we've got one Ba11 and one Bat, If you\'" got several Ba11 objects, au,d rhey all subscribe to the 1',1)lic event Hi t TheBa I I, then they'll all g(flying " hen the event is raised. II", Ihal ,I,esn't make any sense... ii" r..ally only one Ball ohj","I Ihat got hit. We "",'<1 10 let the one hall that's being pitched hook itself IIp to the bat, but we need 10 do it in a way that doesn't allow anv other balls to hook themselves til', ,\ud that's what a **callback** is— it's a way of using a delegate so the object that's calling it is guaranteed 1> only call the une tlwlh"d that il needs 10 call. and no other method can ,'ha ill itself 01110 Ihe delgate.

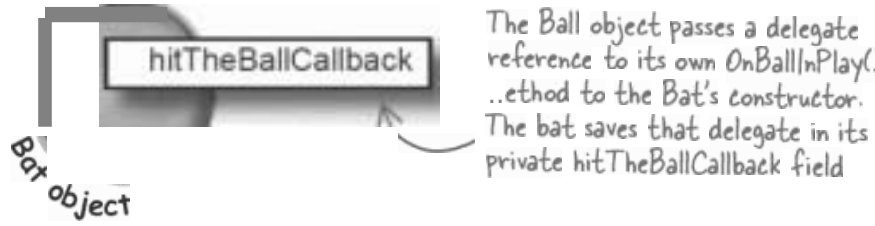
Here's how th.. callback will work:

O **The Bat will** keep **its** delegate **field** private

The easiest way tu keep the wrong Ball objects li"11l chaining themselves "ntu the Bat's delegat(is f('r the bat 10 make it pri,'ate. That way, il has control over which Ball object's method gets called.

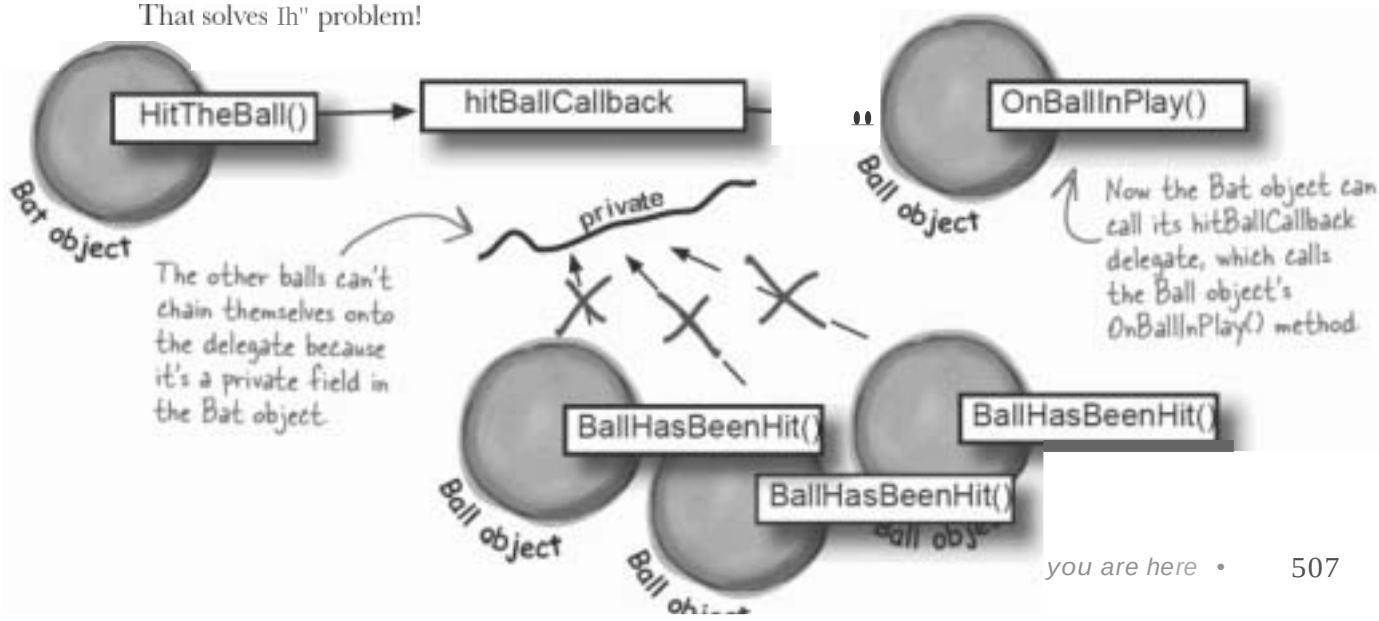
O **The** Ball's constructor tokes a delegate **that** points to a **method** in **the** boll

When the ball is in play, il create the new instance "f the bat, and it passes the Bat object a Ik,inllT 10 its OnBallInPlay() meth(,d, This is called a **callback** m .'1hod because the Bat is using it to call back I" the object Ihal instantiated it.



3 **When the** bot **hits the** ball, it colts **the callback method**

But since the bat kept its delegate privat... il can be 100% sure that nu other ball has been hit. That solves Ih" problem!



Callbacks use delegates, but NOT events

A callback is a different way of **using** a **delegate**. It's not a new keyword or operator. It just describes a **pattern**—a way that you use delegates with your classes so that one object can tell another object, "Notify me when this happens—and I'll tell anyone else!"



Add another delegate to your Baseball project

Since the Bat will have a private delegate field that points to the Ball object's OnBallInPlay() method, we'll need a delegate that matches its signature:

```
public delegate void BatCallback(BallEventArgs e);
```

The Bat object's callback will point to a Ball object's OnBallInPlay() method, so the callback's delegate needs to match the signature of OnBallInPlay()—so it needs to take a BallEventArgs parameter and have a void return value

2 Add the Bat class to the project

The Bat class is simple. It's got a HitTheBall() method that the Umpire will call every time a ball is hit. That HitBall() method uses the hitBallCallback() delegate to call the ball's OnBallInPlay() method (or whatever method is passed into its constructor).

```
public class Bat {
    private BatCallback hitBallCallback;
    public Bat(BatCallback callbackDelegate) {
        this.hitBallCallback = new BatCallback(callbackDelegate);
    }
    public void HitTheBall(BallEventArgs e) {
        if (hitBallCallback != null)
            hitBallCallback(e);
    }
```

Make sure you check every delegate to make sure it's not null, otherwise it could throw a null reference exception

We used = instead of += because there's no need to chain the delegate only gets set once. But if you really feel like using += instead, it'll work just fine.

3 We'll need to hook the bat up to a ball

So how does the Bat's constructor get a reference to a particular ball's OnBallInPlay() method? Easy—just call that ball object's GetNewBat() method, which you'll have to add to Ball:

```
public Bat GetNewBat()
{
    return new Bat(new BatCallback(OnBallInPlay));
}
```

We set the callback in the Bat object's constructor. But in some cases, it makes more sense to set up the callback method using a public method or property's set accessor.

The Ball's GetNewBat() method creates a new Bat object, and it uses the BatCallBack delegate to pass a reference to its own OnBallInPlay() method to the new bat. That's the callback method the bat will use when it hits the ball.

Now we can encapsulate the Ball class a little better

It's unusal for one of the On... methods that raise an event to be public, you can decide this for yourself—go to the form and try to call the playBall button's OnClick() event. You won't be able to, because it's protected (so a subclass can override it). So let's follow that pattern for our ball class, by making its OnBallInPlay() method protected:

```
protected void OnBallInPlay(BallEventArgs e)
{
    if (BallInPlay != null)
        BallInPlay(this, e);
}
```

This is a **really standard pattern** that you'll see over and over again when you work with .NET classes. When a .NET class has an event that gets fired, you'll almost always find a protected method that starts with "On".

5 All that's left to do is hook up the form

The form doesn't need to listen to the Ball object's OnBallInPlay() method anymore—which is exactly what we wanted. That's why we set up the Ball.GetNewBat() method. Now the form needs to ask the Ball for a new bat in order to hit the ball, and when it does, the Ball object will make sure that its OnBallInPlay() method is hooked up to the bat's callback.

```
private void playBall_Click(object sender, EventArgs e)
{
    Bat bat = ball.GetNewBat();
    BallEventArgs ballEventArgs = new BallEventArgs(
        (int)trajectory.Value, (int)distance.Value);
    bat.HitTheBall(ballEventArgs);
}
```

If the form (or the simulator) wants to hit a Ball object, it needs to get a new Bat object from that ball. The ball will make sure that the callback is hooked up to the bat. Now when the form calls the bat's HitTheBall() method, it calls the ball's OnBallInPlay() method, which fires its BallInPlay event.

Now run the program. It should work exactly like it did before. But it's now protected from any problems that would be caused by more than one ball listening for the same event.

But don't take our word for it—pop it open in the debugger!

BULLET POINTS

- When you add a delegate to your project, you're creating a new type that stores references to methods
- Events use delegates to notify objects that actions have occurred
- Objects subscribe to an object's event if they need to react to something that happened in that object.
- An EventHandler is a kind of delegate that's really common when you work with events
- You can chain several event handlers onto one event. Here's why you use += to assign an handler to an event
- Always check that an event or delegate is not null before you use it to avoid a NullReferenceException
- All of the controls in the toolbox use events to make things happen in your programs
- When one object passes a reference to a method to another object so it—and only it—can return information, it's called a callback
- Events let any method subscribe to your object's events anonymously, while callbacks let your objects exercise more control over which delegates they accept
- Both callbacks and events use delegates to reference and call methods in other objects
- The debugger is a really useful tool to help you understand how events, delegates, and callbacks work. Take advantage of it!

there are no Dumb Questions

Q: How are callbacks different from events?

A: Events are part of C# They're a way for one object to announce to other objects that something specific has happened When one object publishes an event, any number of other objects can subscribe to it without the publishing object knowing Of caring When an object fires 011 an event, • anyone happens to have subscribed to it then it calls each of their event handlers

Callbacks ar, not published Instead, a callback establishes a relationship between two clases where one object automatically reacts 10 another A callback ıs generally kept private, and the class that stores the callback maintains control over who has access to it. A callback ıs often set up ın an object's

Q: So a callback il n'l l n aclual type in C#?

A: No, it ısn't. A callback is a ***pattern***—it's just a novel way of usıng the existing types, keywords and tools that C# comes with. Go back and take another look at the callback code you just wrote for the Bal and Ball Did you see any new keywords that we haven't used before? Nope!

It turns out that there are a lot of patterns that you can use In fact, there's a whole area of programming called ***design patterns*** There are • lot problems that you'll run into which have been solved before and the ones that pop up over and over again have their own design patterns that you can benefit from

Check ....t "Head First Design Patter s" at the Head First Labs website It's a great way to learn about different patterns that you can apply to your - programs

www.headfirstlabs.com/books/hfdp/

Q: So callbacks are just private events?

A: Not quite • seems easy to think about it that way, but private events are a different beast altogether Remember what the p r i v e t.e access modifier really means? When you mark ıl class member private, only instances of that same class can access it So if you mark an event p r1vate. then other instances if the same class can subscribe to it That's different from ıl callback. because it still involves one or more objects anonymously subscribing to an event

Q: But it looks jllıl lıkt . n event, except with the event keyword, right?

A: The reason a callback looks so much like an event is that they both use d.l.g. l.. And It makes sense that they both use delegates, because that's C#'s tool for letting one object pass another object a reference to one of its methods

Bullhe big difference between normal events and callbacks is that an event ıs ıl way for a class to publish to lhe world that some specific thing has happened A callback, on the other hand, ıs never published It's private and the method that's doing the calling keeps tight control over who it calling

The first one you'll learn about ıs called the "Publisher-Subscriber" pattern, and it'll look really familiar to you One object publishes ınformation, and other objects subscribe to it Hmmm

The Case of the Golden Crustacean



How did the other treasure hunters beat Henry to the crab?

The crux of the mystery lies in how the treasure hunters seeks his quarry. But just watch to see exactly what all those fellows did with their diagrams.

With a stolen set of class diagrams, Henry discovers that the Golden Crab class raises a RunForCover event every time anyone gets close to it. Even better, the event includes NewLocationArgs, which detail where the crab is moving to. All the other treasure hunters know about the event, so Henry figures he can cash in.

```
public class GoldenCrab {
    public delegate void Escape(NewLocationArgs e);
    public event Escape RunForCover;
    public void SomeonesNearby()
        NewLocationArgs e = new NewLocationArgs("Under the rock");
        RunForCover(e);

}
public class NewLocationArgs {
    public NewLocationArgs(HidingPlace newLocation)
        this.newLocation = newLocation;
    }
    private HidingPlace newLocation;
    public HidingPlace NewLocation { get { return newLocation; } }
```

Any time someone comes close to the golden crab, its SomeonesNearby() method fires off a RunForCover event, and it finds a place to hide.

So how did Henry take advantage of his newfound insider information?

Henry adds code to his constructor to register his ListenForClues() method as an event handler for the RunForCover event, on the crab reference he's got. Then, he sends a lowly underling after the crab, knowing it will run away, hide, and raise the RunForCover event—giving Henry's ListenForClues() method the information he needs.

```
public class TreasureHunter {
    public TreasureHunter(GoldenCrab treasure)
        treasure.RunForCover += new GoldenCrab.Escape(treasure_RunForCover);

    void treasure_RunForCover(NewLocationArgs e) {
        MoveHere(e.NewLocation);
    }
    void MoveHere(HidingPlace Location)
        // ... code to move to a new location
    }
```

Henry thought he was being clever by altering his class's constructor to add an event handler that calls his MoveHere() method every time the crab raises its RunForCover event. But he forgot that the other treasure hunters inherit from the same class, and his clever code adds their event handlers to the chain, too!

And that explains why Henry's plan backfired. When he added the event handler, the TreasureHunter constructor was inadvertently ***doing the same thing for all of the treasure hunters!*** And that meant that every treasure hunter's event handler was chained onto the same RunForCover event. So when the Golden Crustacean ran for cover, everyone was notified about the event. And all of that that would have been fine if Henry were the first one to get the message. But Henry had no way of knowing when the other treasure hunters would have been called—if they subscribed before he did, they'd get the event first.

Sharpen your penCil

Fill in the blanks to make this game of Whack-a-mole wor k. You need to ,upply the code that does the callbacks. Once you've got it filled in, go ahead and typ.. it into the IDE. O, you can t,y to g..t it working in Ihe IDE, and then fill in the bla nk, afterwards. It's fun'

```
public partial class Form1 : Form
    Mole mole;
    Random random • new Random();
    pubLic Form1() (
        Initializecomponent();

        mole = new Mole(random, new Mole._____                    )):
        timerl.lnterval - random.Next(500, 1000);
        timerl.Start():

    private void timerl_Tick(object sender, EventArgs e) (
        timerl.Stop():
        ToggleMole():
    )
    private vore ToggleMole() {
        if (mole.Hidden -- true)
            mole.Show():
        else
            mole.HideAgain();
        timerl.Interval - random.Next(500, 1000).
        timerl.Start():
    }
    private void HoleCaIlBack(int MoleNumber, bool Show) (
        if (MoleNumber      {
            timerl.Stop()?)
            return:
        }
        Button button:
        switch (MoleNumber) {
            case 0: button . buttonl: break;
            case !: button   button2; break;
            case 2: button · button3: break;
            case 3: button _ button4; break;
            default: button . button5; break;
        )
        if (snov == true)  {
            button.Text • "HIT ME""
            euct on.uecxcorcr = Color.Red:
          else {
            button.Text = "";
            button.BackColor • SystemColors.Control:
        }
        timerl.Interval · random.Next(500, 1000);
        timerl.Start():
    }
    private void buttonl Click(object sender, EventArgs e) {
        mole.Smacked(O);-
```

The form passes a delegate pointing to a callback ··ethod into the ··ole's constructor Fill it in

When you double-click on the timo- in the form (after r- drag it out of the toolbox), the IDE w,ll create this event han dl er for it. T,...ers fire the Tick event over and over again You'll learn all about the... in the next chapter

This method's called to pop "1' or hide the ..ole when the ti··er's elapsed

This switch makes sure that the right button changes its color and text



When r- type in the code, add five button event handlers Have button2_click() call mole. Smacked(1), and then ...ske button3 call ...ole.Smacked(2), and make button4 call ..ole. Smacked(3) and button5 call mole.Smacked(4).

Just add these event handle the usual way by double-clicking in the for designer.

```
public cress Mole (

    public                  void PopUp(int hole, bool show):

    private ____ ______ popUpe"llb"ck;
    private bool hidden;
    public bool Hidden { get { return hidden; ) I
    private int timesHit _ 0;
    private int timesShown _ 0;
    private int hole - 0;
    Random random;

    public Mole(Random random, PopUp popUpCallback) {
        it (popUpCallback == null)
            throw new ArgumentException("popUpCallback can't be null");
        this.random _ random;


        this.______________________
        hidden = true;


    public void Show()
        timesShown++;
        hidden - false;
        hole - random.Next(5);

        ________________(hole, true);


    public void HideAgain()
        hidden - true;

        ____________________(hole, false);
        CheckForGameOver();

    public void Smacked(int holeSmacked)
        if (holeSmacked -- hole)
            timesHit••;
            hidden _ true;
            CheckForGameOver():

            ____________________lhole, false);


    private void CheckForGameOver() {
        if (timesShown >- 10) (
            popUpCallback(-1, false);
            MessageBox.Show("You scored " + timesHit, "Game over");
            Application,Exit();
```

Fill in the delegate and field to hold the delegate—they're both at the top of the Mole class

Here's where we make sure the callback .. not null—if it ... the Mole object throws an ArgumentException

When the for.. creates a _ Mole object, it passes it a reference to its callback ,•ethod Take a look in the form to see how the constructor .. called, and then fill in this blank

After the mole shows itself, it needs to call the method on the form that displays the mole by turning the button red and showing the text "HIT ME!"

The HideAgain() and Smacked() methods also use the callback delegate to call the method on the form

The way the game works is that it uses the timer to wait a random period of time between half a second and 1.5 seconds. Once that time is elapsed, it tells the mole to show itself. The form gives the Mole object a callback that it uses to tell the form to show or hide the mole in one of the five holes. The form uses its timer to wait between .5 and 1.5 seconds again, and then tells the mole to hide itself.

The ga..e's over after the ..ele shows itself '0 times Your score is the .....ber of times you hit it

Sharpen your pencil Solution

Fill in the blanks to make this game of Whack-a-mole work. You need to supply the code that does the callbacks. Once you've got it filled in, go ahead and type il inlo the IDE. It', fun!

```
public partial class forml : form {
    private void Form1_Load(object sender, EventArgs e)  {

        mole = new Mole(random, new Mole.        POpUp        MoleCallBack        ));
        timer1.Interval _ random.Next(500, 1000);
        timer1.Start();

.}
```

This il where the form passes a refe-ence to its MoleCallBack() method into the Mole object. That lets the mole eall its method

```
public class Mole {

    public    delegate

    prLvate   PopUp    popupcal Lbackr
```

show); Here's where the mole defines its delegate and uses it to set up a private field to hold a reference to the method on the form that changes the colors of the buttons

```
    publIc Mole(Random random, PopUp pcpupcal Lback) (
        this.random _ random;

        this.popUpCalibaek = popUpCalibaek;
        hidden = true;
```

When the form creates a new instance of the Mole object, it passes a reference to its MoleCallBack() method to the constructor as a parameter. This line in the constructor copies that reference to its popUpCallback field. Its methods can use that field to call the MoleCallBack() method in the form.

```
    public void Show()
        t Imes Shcvneer
        hidden . false;
        hole . random.Next(S);

        _ popUpCallback    (hole, true);

    pUblic void HideAgain{)
        hidden · true;

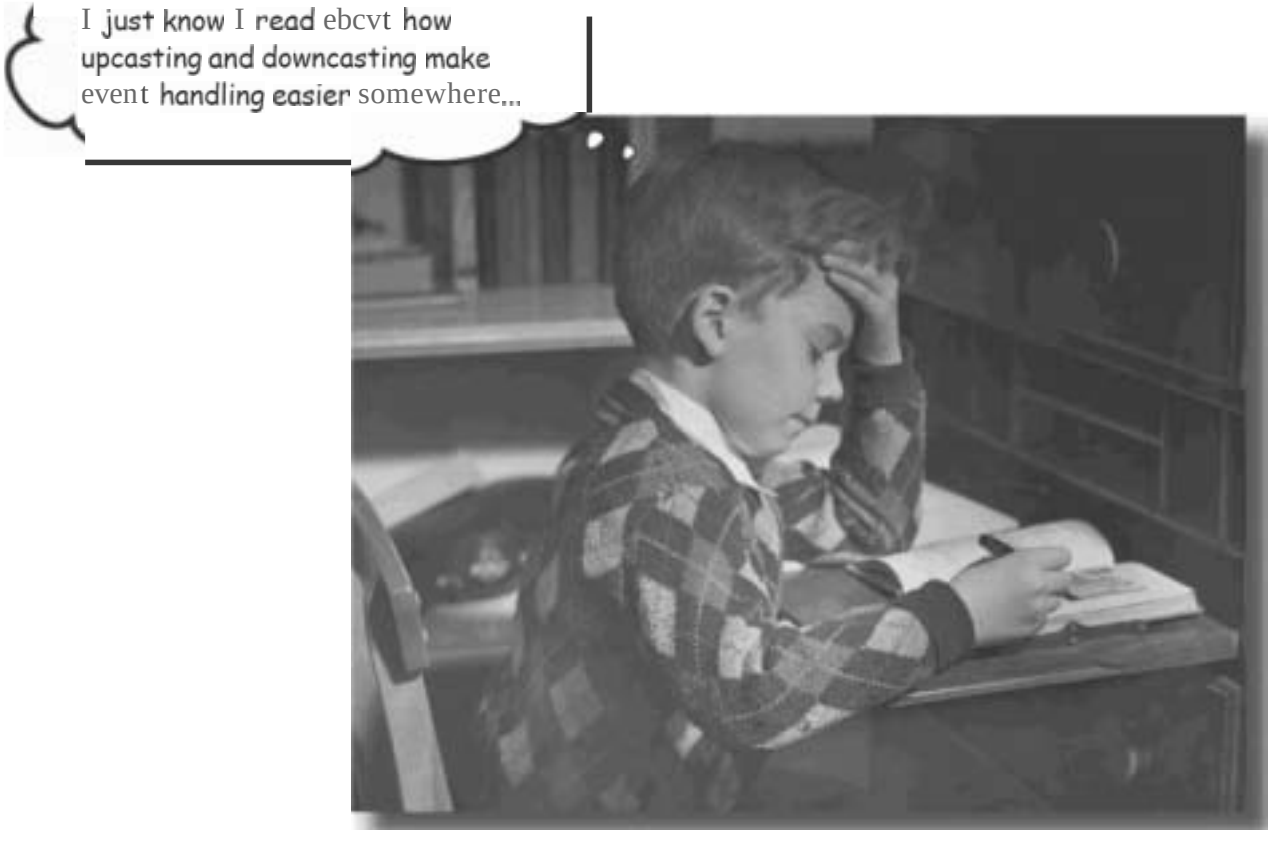
        popUpCallback    (hole, false);
        CheckForGameOver();

    public void Smacked lint holeSmacked)
        if (holeSmacked -- hole) (
            timesHitH;
            hidden _ true;
            CheckForGameOver();

                popUpCalibaek
```

When the mole shows itself, hides again, or gets smacked, the Mole object uses its popUpCallback delegate field to call the method on the form that changes the color and text of one of the buttons.

12 review and preview



Learning's no good until you BUILD something.

Until you've actually written working code, irs hard to be sere if you really *gel* some of the tougher concepts in C#. In this chapter, we're going to learn about some new odds and ends: **timers** and dealing with collections using IINQ (to name a couple) We're also going to build phase I of a **really** complex application, and make sure you've got a good handle on what you've already learned from earlier chapters So buckle up" ,Il'S time 10 build some cool software

YOU've cOllie a long way, ~~baby~~

Enote from human resources: "baby" i. no longer politically correct. Please use age-challenged or infant to avoid offending readers.

\\" 've come a long way since we first used the IDE t,> help '" rescue the Objectville Paper Company. Here's jusr a few ,>1' the things you've d(,JII' over the last several hUll<h'eds pages:

You've built forms, used the .NET framework, and even talked with databases.

.NET Framework solutions

Objects, classes, instances... all these strange terms are now part of your everyday programming toolbox.

Navigator

SetDestination()
ModifyRouteToAvoid()
ModifyRouteToInclude()
GetRoute()
GetTimeToDestination()
TotalDistance()

Navigator object

Navigator object

Navigator object

You've used inheritance, as well as interfaces and subclasses, to build object trees.

Party

NumberOfPeople
CostOfDecorations
CalculateCostOf[] ations()
CalculateCost()

DinnerParty

NumberOfPeople
CostOfDecorations
CostOfB '''' erPerso
HealthyOption
CalculateCostOfDecorations()
CalculateCost()

BirthdayParty

NumberOfPeople
CostOfDecorations
CakeSize
CakeWriting
CostOf[] ations()
CalculateCost()

Even complex types like arrays are no big deal to work with.

7 int variables

0 int | 1 int | 2 in, | 3 int | 4 int | 5 in | 6 int

HEIGHTS[]

You've used events to notify objects about certain things that happen, while keeping your objects' concerns separate.

Ball object

BallInPlay event

BallEventArgs

Pitcher

Umpire

Fan object

Debugging and exceptions are part of your problem-eliminating techniques.

```
private void RandomExcuseButton_Click(object sender
{
    CurrentExcuse = ...
    UpdateForm(false);
}
```

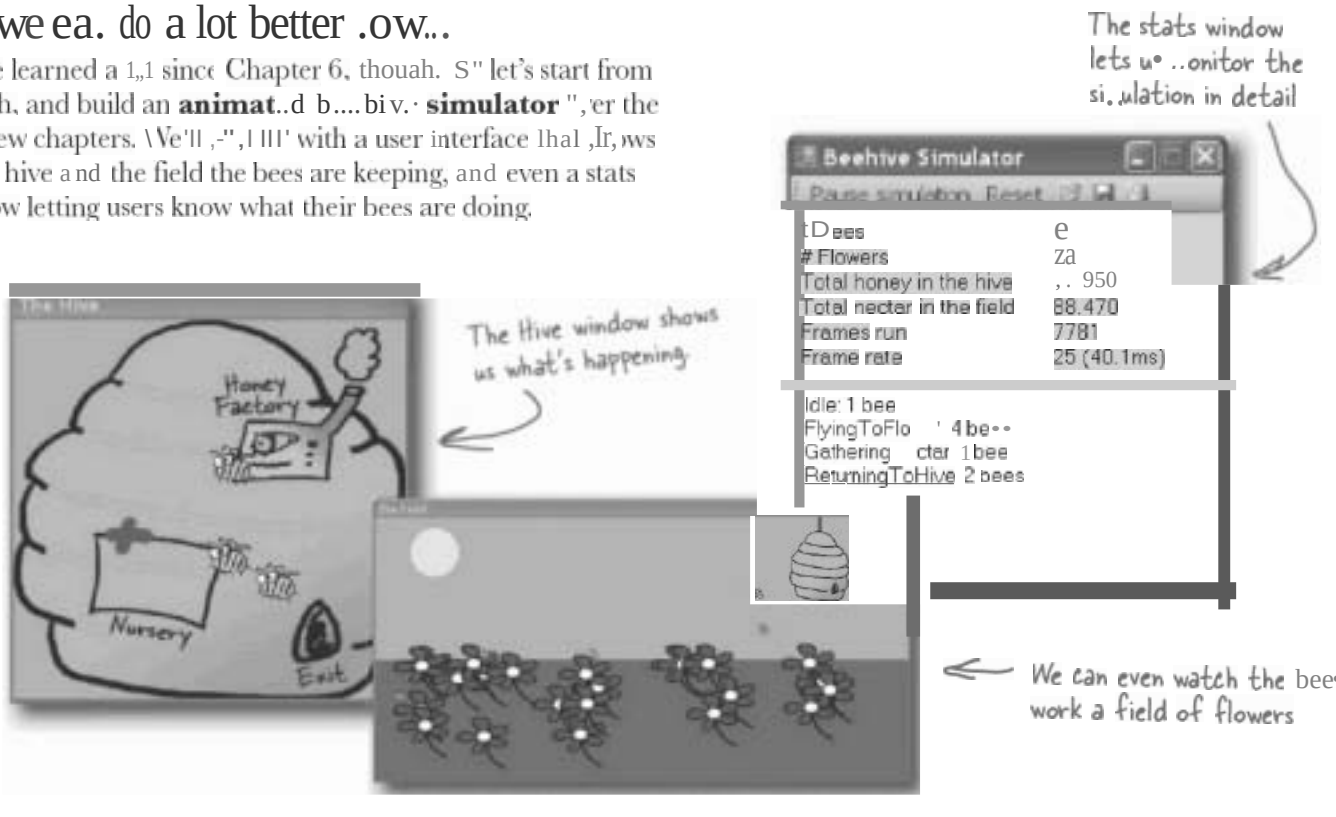
We've also beeollle beekeepers

Back in Chapter 6, we built some bee classes. Remember these?



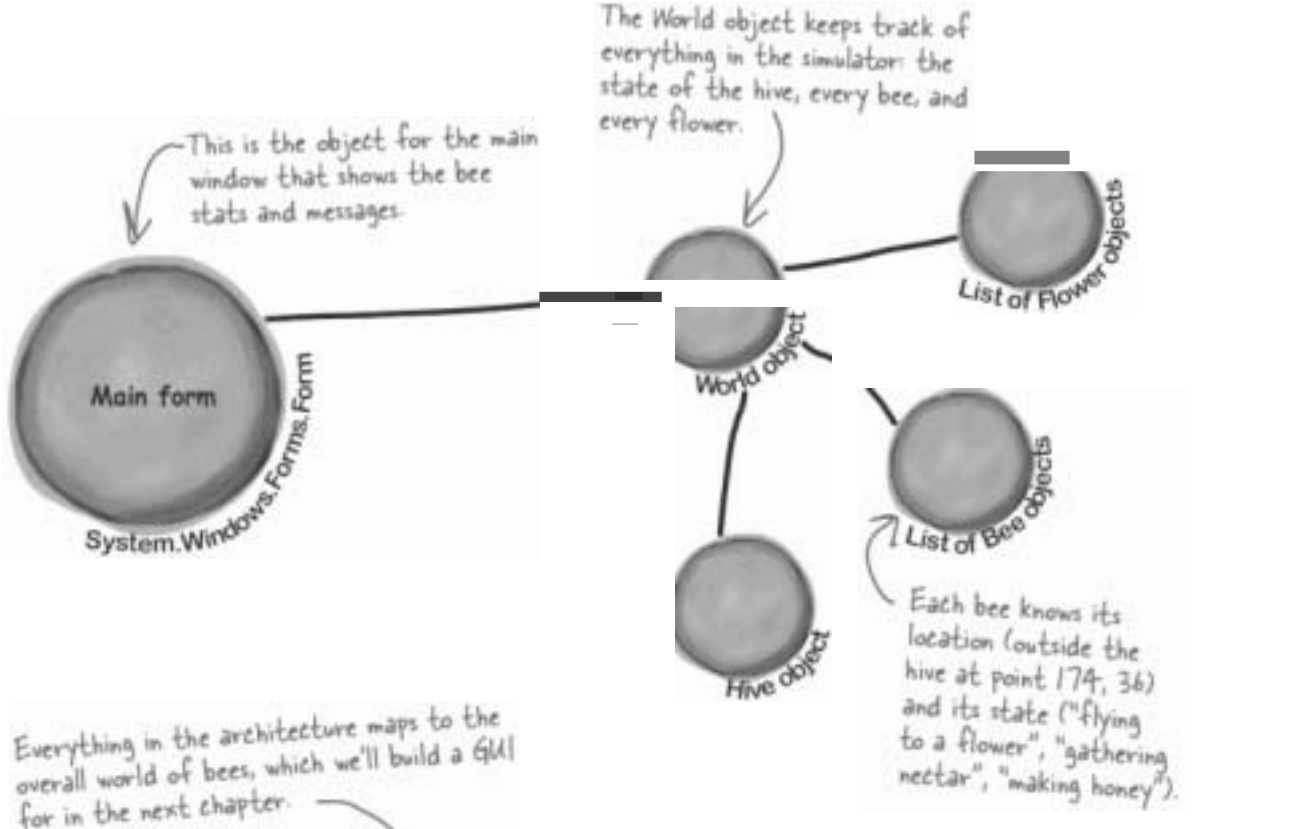
But we ea. do a lot better .ow...

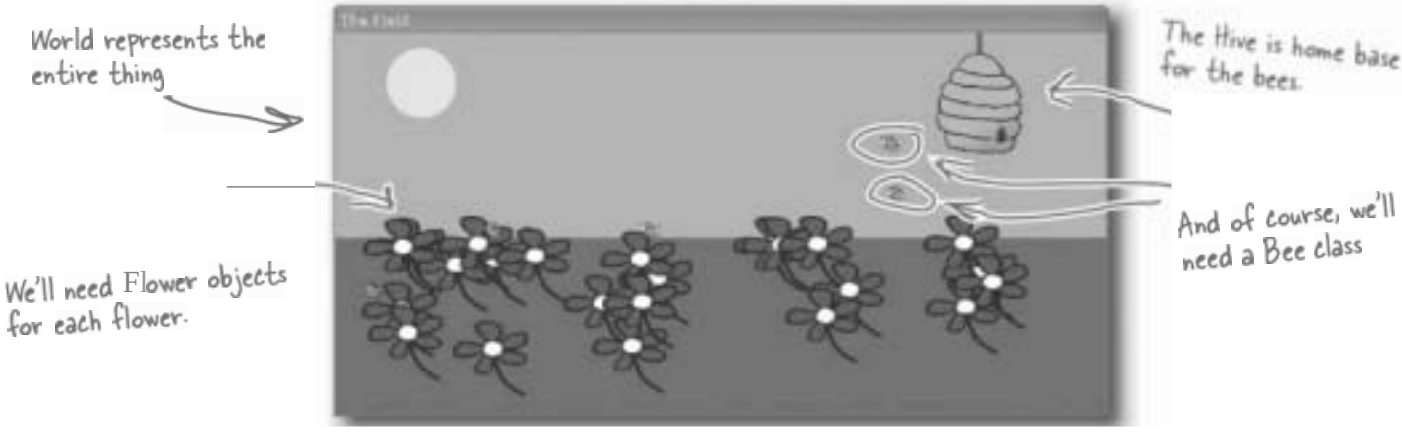
You've learned a 1„1 since Chapter 6, thouah. S" let's start from scratch, and build an **animat**..d b....biv.· **simulator** ",'er the next few chapters. \Ve'll ,-",I III' with a user interface lhal ,Ir,ws us the hive and the field the bees are keeping, and even a stats window letting users know what their bees are doing.



Ihe beehive shllulator architecture

Here's the architecture for the h,-,- ,itl\II"to['. Even though the simulator will be controlling a 101 of different bees, the overall object model i, pretty simple.





Building the beehive shllulator

Of course, we've never huill anything thi, complex before, so it's going I" take u, a couple „1' chapters to put all the pieces together. Along the way, vou'H add timer,. LINQ, ,""I a lot "I' graphical ,killw your [oolkit

Here's what you're going to <1" in this ch"I'{,'r (more In come in the next):

1. **Build a Flower class that ages, produces nectar, and eventually wilts and dies.**

2. **Build a Bee class that has several different states (gathering nectar from a flower, returning to the Hive), and knows what to do based on Its state.**

3. **Build a Hive class that has an entrance, exit, nursery for new bees, and honey factory for turning collected nectar into honey.**

4. **Build a World class that manages the hive, flowers, and bees for any given moment.**

5. **Build a main form that collects statistics from the other classes, and keeps the world going.**



Let's jump right In to some code First up, we need a Flower crass. The Flower class has a location. defined by a Point. an age, and a lifespan, As time goes on. the flower gets older, Then, when its age reaches its lifespan, the flower dies It s your job *to* put all this into action.

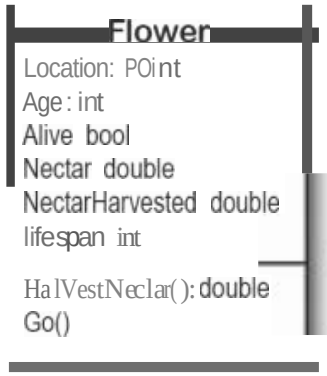
A class "skeleton" is just its field, property and ..ethod declarations with no i..ple...entation.

Write the skeleton code for Flower

Below is Ih,' class diagram for Flower. Write the basic class skeleton. Loeat.Ion. Age, Al i ve, Nect ar. and NectarHarvested are autom. ali.. prop....li..s. NectarHarvested is writable, the other four "n' read.only. For now, leave Ih,' ""'Ih"d, blank; we'll come back to those in a minute.

All of these should be read-only properties.

This is used only in the class, so it just needs be a private field



The type after the colon is the type of the variable...

...or the return type of the method.

Add severe! constants to the class

We])" ,'d lots of constants for flowers. .\dd six I" your Flower class:

- LifeSpanMin, the shortest flower lifespan
- LifeSpanMax, lhe longest flower lifespan
- InitiaINectar, how mud, "",'lar a flower starts with
- MaxNeetar. how l1u,,'h nectar a flower can lH,Id
- NectarAddedPerTurn, how much nectar gets added each time Ih,' flo"',·,. grows old,'r
- NeetarGatheredPerTurn. how mud, nectar gets collected during a cycle

FYI, you don't usually show constants in a class diagram.

r"l! should I,,·able 10 figure oUl the types for each constant based onlh,·i,. values. 1'1,wers live between 15,00(Iand 30.00(Icycles, ,,,,d h'l\" 1.5 uuit, or nectar when they ,larl out. They can store "l' I" 5 l1"i" "l' nectar. In each cycle "l' Iii;-. a flower adds l1.0 I llnit, of nectar, and il1" single cycle, 0.3 units can I", ,'"II""t",1.

You'll need to add `using System.Drawing;` to the top of any class file that uses a PQint.

3 Build the constructor

'1'1,,' constructor for `Flower` should take in a `Point`, indicating the flow..r's location, and an instance of the `Random` class. You should be, able III use Ih" ,,' arguments to set Ihe I,ocation of the now,'I', and the-n set its age to 0, set the flower 10 ahvr-. and set its nectar 10 the initial ,,,Holml "I' nectar for a flower. Since H" nectar has been harvested yet, set that variable correctly, a well. Finally, figure 'Iiit th.. flower's lifespan. Here's a lilll' of code to help you:

```
lifeSpan = random.Next(LifeSpanMin, LifeSpanMax + 1);
```

This will only work if you've got your variables and constants named right, as well as the argument to the Flower constructor.

Write cede for the HarvestNectar() method

Every lim.. Ihi, method is called, it ,]"ould check I" see if Ih,' nectar gathered every cycle i, larger than Ih" all" """ "I' nectar left. If so, return 0. () Iherwise, you should remove the amount collected in a cvcle from the nectar the flower hOl, left, and retur-n how muc-h nectar was collected. (Ih. and don't forget to add th.« amount 10 the `NectarHarvested` variable, I,hid, keeps up with the 1111,,1 nectar collected from Ihi, particular flower.

*Hint: We used `NectarGatheredPerTu t` . nectar, and `NectarHarvested` in this ethod, but nothing else

Write code for the GoO method

Th is is the method that rnakes the flower go. Assume every Inne' Ihi, meth"d i, called, one cvcle passes s« update the flower's age appropriately. You'll also need 10 see if the age i, greater Ihan the Il,wer's lifespan. If so, the flower dies,

Assuming Ih" flower stays alive, you'll ne..d 10 add Ih" amount of nectar each flower gets in a cycle. B.' sure and check against the maximum nectar your flower can store, and don't overrun that.

The final product will be **animated**, with little pictures of bees flying around, The Go() ..ethod will be called once every fra..e. and there w;ll be several frames run per second

Answers on the next page... try and finish your code and compile it before peeking.



Your job was to build the Flower class for our beehive simulator.

Flower
tccatco: Point
Age: Int
Alive boo!
Nectar: double
Necta ltarvested double
lifespan: int
HarvestNectar() double
Go()

```
public class Flower I
    private canst int LifeSpanMin = 15000;
    private canst int LifeSpanMax - 30000;
    private canst double InitialNectar = 1.5;
    private canst double MaxNectar - 5.0;
    private canst double NectarAddedPerTurn = 0.01:
    private canst double NectarGatheredPerTurn - 0.3:
    pUblic Point Location I get; private set; )
    public lnt Age { get: prIvate set: }
    public bool Alive ( get; private set; J
    public double Nectar { get; private set;
    public double NectarHarvested ( get; set; )
    private int lifeSpan:

    public FLowerfPoint; Location, Random random) (
        Location - location;
        Age = 0;
        Alive - true;
        Nectar = InitialNectar;
        NectarHarvested = 0;
        LifeSpan = random.Next(LifeSpanMin, t.tresoenaex • 1);
    ,

    public double HarvestNectar()
        tr (NectarGatheredPerTurn > necter}
            return 0:
        else {
            Nectar     NectarGatheredPerTurn;
            NectarHarvested += NectarGatheredPerTurn:
            return NectarGatheredPerTurn;
    ,

public void Go() (
    Age++;
    if (Age > lifeSpanl
        Alive = false;
    else (
        Nectar +- NectarAddedPerTurn;
        if (Nectar > MaxNectar)
            Nectar = MaxNectar:
```

Location, Age, Alive, Alive, and Nectar are all read-only automatic properties.

NectarHarvested will need to be accessible to other classe°.

Flowers have random lifespans so that the field of flowers doesn't all change exactly at once

A bee calls HarvestNectarO to get nectar out of a flower A bee can only harvest a little bit of nectar at a ti..e, so he'll kave to sit near the flower for several turns until the nectar's all gone.

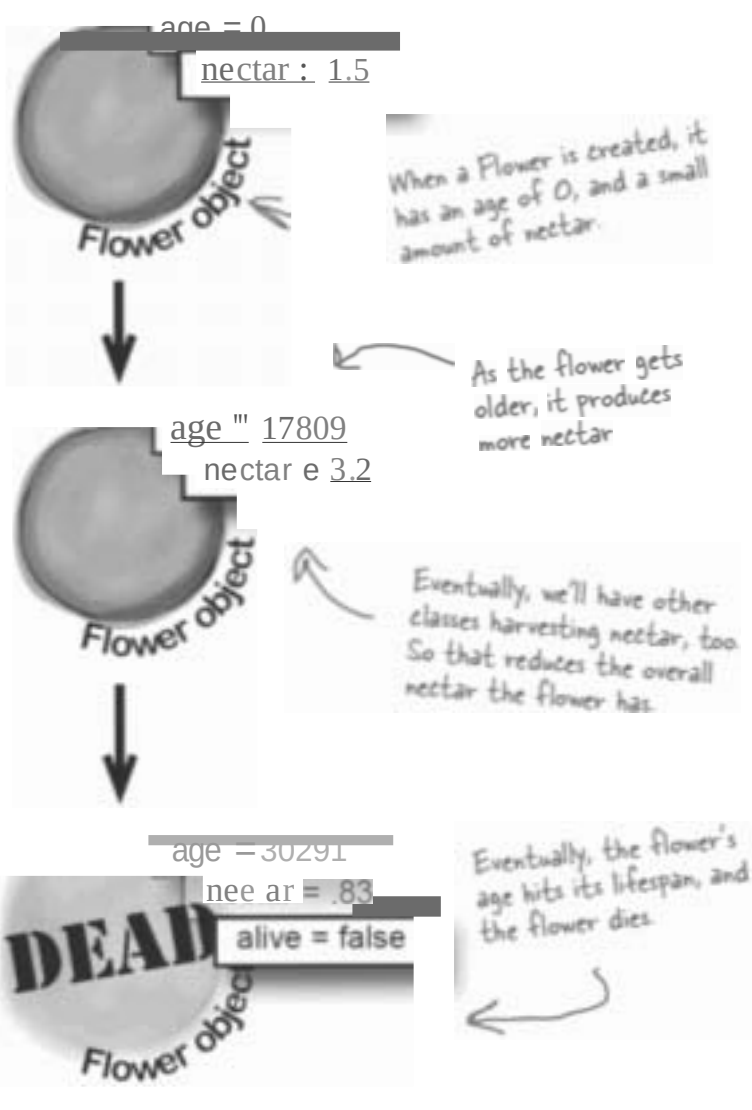
A. part of the "..ulator's ani..ation, the Go() ..ethod will be called each frame. Thi makes the flower age just a tiny little bit per fra..e—as the 'i..ulator runs, those tiny bits will add up over ti..e

Make sure the flower stops adding nectar after it's dead.

Point lives in the System.Drawing namespace, so make sure you added using System.Drawing; to the lop of the class file.

life alld death of a flower

'1", Ηwer goe• through a basic lurn, living, adding nectar. having nectar h.,„vested, ...,,1eventually dying:



there Jare no DumlJ Questions

0_: It doesn'l look lik. NectarHarv••ltd it lind anywh. . . In the class, except where we increment it. What's that variable lor?

A: Good catch! We re planning ahead a bit. eventually the simulator will keep an eye on flowers, and how much total nectar has been harvested, for our statistics monitor So leave it in, and our other classes will use it shortly

0_: Why all the . .ad-only automatic properties?

A: Remember Chapter 5, and hiding our privates? Always • good practice Flowers can take care of those values, so we've made them read-only Other objects, like bees and the hive, should be able III read those properties, but not change them

0_: My code looks differe„1. Did I do something wrong?

A: You might have your code in each method 11'11 different order, but as long as your code ***functions*** the same way as ours does, you'll be okay That's another aspect of encapsulation the internals 01 each class aren't important to other classes, as long as each class does what it's supposed to do



If Go(l increases the age of the Flower by 1. and the lifespan range is between 15.000 and 30,000, that means GoO will get called at least 15.000 limes for each flower before it dies. How would you handle calling the method that many times? What if there are 10 flowers? 100? 10001

Now we need a Bee class

With flowers ready to be harvested, we needed a Bee class. Below is the basic code for Bee. The Bee knows its age, whether or not its in the hive, and how much nectar it can carry. We've also added a method to move the bee towards a specific destination flower.

```
public class Bee {
  private const double HoneyConsumed = 0.5;
  private const int MoveRate = 3;
  private const double MinimumFlowerNectar = 1.5;
  private const int CareerSpan = 1000;

  public int Age { get; private set; }
  public bool InsideHive { get; private set; }
  public double NectarCollected { get; private set; }

  private Point location;
  public Point Location { get { return location; } }

  private int ID;
  private Flower destinationFlower;

  public Bee(int id, Point location) {
    this.ID = id;
    Age = 0;
    this.location = location;
    InsideHive = true;
    destinationFlower = null;
    NectarCollected = 0;
  }

  public void Go(Random random)
    Age++;
```

Like the Flower class, there are several bee-specific constants we need to define

MinimumFlowerNectar is how the bee figures out which flowers are eligible for harvesting

We used a backing field for location. If we'd used an automatic property, MoveTowardsLocation() wouldn't be able to set its members directly ("Location.X -= MoveRate").

Each bee will be assigned its own unique ID number

A bee needs an ID and an initial location.

Bees start out inside the hive, they don't have a flower to go to, and they don't have any nectar.

We'll have to add a lot more code to Go() before we're done, but this will get us started

Here we used Math.Abs() to calculate the absolute value of the difference between the destination and the current location.

```
private bool MoveTowardsLocation(Point destination) {
    if (destination != null) {
        if (Math.Abs(destination.X - location.X) <= MoveRate &&
            Math.Abs(destination.Y - location.Y) <= MoveRate)
            return true;
        if (destination.X > location.X)
            location.X += MoveRate;
        else if (destination.X < location.X)
            location.X -= MoveRate;
        if (destination.Y > location.Y)
            location.Y += MoveRate;
        else if (destination.Y < location.Y)
            location.Y -= MoveRate;
    }
    return false;
}
```

This method starts by figuring out if we're already within our MoveRate of being at the destination.

If the bee has no destination, its field will be set to null. So we only move towards it if its destination is NOT null

If we're not close enough, then we move towards the destination by our move rate

We return false, since we're not yet at the destination point. We need to keep moving.

The MoveTowardsLocation() destination moves the bee's current location by changing the X and Y values of its location field. It returns true if the bee's reached its destination.



Bees have lots of things they can do. Below is a list. Create a new enum that Bee uses called BeeState. You should also create a read-only automatic property called CurrentState for each Bee to track that bee's state. Set a bee's initial state to Idle, and in the Go() method, add a switch statement that has an option for each item in the enum.

The enum item	What the item means
Idle	The bee isn't doing anything
FlyingToFlower	The bee's flying to a flower
GatheringNectar	The bee's gathering nectar from a flower
ReturningToHive	The bee's heading back to the hive
MakingHoney	The bee's making honey
Retired	The bee's hung up his wings



Bees have lots of things they can do. Below is a list. Create a new erum that Bee uses called BeeStateoYou shoud also create a private currentState field for each Bee to track that bee's state. Set a bee's initial slate to rdte, and in the Go() method, add a sWitch statement that has an option for each item in the enum

```
public enum BeeState
  Idle,
  FlyinqToFlower,
  GatheringNectar,
  ReturningToHive,
  MakingHoney,
  Retired
}
```

Here's the enum with all the different bee states.

```
public class Bee I
  /1 constant declarations
  // variable declarations

  public Beestate Currentstate { get; private set;

  public Bee(int ID, Point InitialLocation)
    this.ID = W;
    Age = 0;
    location = InitialLocation;
    InsideHive = true;
    CurrentState = BeeState.Idle;
    destinationFlower _ null;
    NectarCollected - 0;
```

We also need a variable to track the state of each bee.

The bee starts out idle

Did you remember to add using System.Drawing; to the top of the class file (because it uses Point)?

We've filled ..t a few of the states. It's okay if you didn't come up with th.. code, but go ahead and add it in now

```
public void Go(Random random)
    Age++;
    switch (CurrentState) I
      case BeeState.Idle:
        if (Age > CareerSpan) |
          CurrentState - BeeState.Retired;
          else |
          // What do we do if we're idle?

        break;
      case BeeState.FlyingToFlower:
         // move towards the flower we're heading to
         break;
      case BeeState.GatheringNectar:
        double nectar - destLnat.Lonj'Lower.HarveatNecter n.
        if (nectar > 0)
          NectarCoilected +- nectar;
        else
          CurrentState _ BeeState.ReturningToHive;
        break;
      case BeeState.ReturningToHive:
        if (!|nsideHivel |
          // move towards the hive
          else t
          // what do we do if we're inside the hive?
        | break;
      case BeeState.MakingHoney:
        if (NectarCollected < 0.5)
          NectarCollected - 0;
          CurrentState - BeeState.Idle;
          else |
          // once we have a Hive, we'll turn the nectar into honey

        break;
      case BeeState.Retired:
        // Do nothing! We're retired!
        break;
```

Here's the switch() statement to handle each bee's state

If the age reaches the bee's lifespan, the bee retires. But he'll fi.. sh the current job before he does

We'l fill this code in a bit later

Here, we harvest nectar from the flower we're working

--and if there's nectar left, add it to what we've already collected

but if there'. ... nectar left, head for the hive

Returning to the hive is different based on whether we're already in the hive or not

The bee adds half a unit of nectar to the honey factory at a time. If there's not enough nectar to add, the factory can't use it so the bee just discards it

You should have each of these states covered

P. A. II. B. IPrograllllllers Agaillsf 1I0IlIeless Bees)

\\e'\". got bees, and flowers h,II of nectar. We neecI'o write code so thc·bees ,'jll ,ollect nectar, but befo„',' tha happens, where do the bees get created in the first place? .\ud where do they ,;,k,' all that ,...,,t.,r:' That's " h,'re a Hive clas comc.. in.

'11,e hive isn't just a 1'1...:e for bec.. Io come ba.. k to, thougl•. I, has se' ral loca ion wit hin it. all with differ·nt p .;nt, in thE "Hrld. The th ntr; uice au..1,10., exit. a well a: a nurserv for birthing more 1 „1 ., hoı..:y fac "ıry for turning nectar ;"t" honey.



Each location is distinct, and bees can travel from one to the other just like they can go from the hive to a flower.

New bees are created and start out i.. the hive nursery

Bees come in the entrance, and leave from the exit. It's all very orderly.

The hive ruas on hOlley

'11,,' other big part th.u tl,,' hiv" plays is keeping up " ilh ho" much honey it has sto :d up. I' lal:,.. honey for the hive to keep running, and ;f new 1" ll.·,',11o 1.. created, Ih.,t takes h,ney, """ (In top of ,h.". the honev I:" lory h." Io ,..k,· ,,,',';,,· th;u bees co11,"·' and turn that illto honey. Fi,r every IInil "I' nectar that ,'omes in. .25 units „1' honey '·.n, I", created.

Think about this f a second as time passes, the hive use ey to run, and to create mor bees Mea..while, other bees are bringing in nectar, which gets turned into honey, which keeps things going longer

It's up to you (with some help) to model all of this in the simulator code

Exercise

Irs up to you to write the code for Hive

Hive
Honey: double ocanons: Dictionary<string, Point> teeccua: iot
InitializeLocations() AddHoney(Nectar: double) bool ConsumeHoney(amount: double): bool AddBee(random: Random) Go(random: Random) getLocation(location string) Point

- **Write the skeleton code for Hive**

Like we did wirh the PLover- class, you ,hould start with a basic skeleton f"l Hive. The class diagram i, shown 1il the right. Make the Honey " read-only aUloma,;, -property, `locations` ,hould he I'.-i'-"h" and beeCount is only ""xI internally, so can h.. a private field.

- **Define the constants for the Hive**

You need a constant for the initial number of bees (6), the amount of honey the hive starts with (3.2), the maximum amount of honey the hive can store (15), and ratio of units of nectar produced from units of honey (.25), the maximum number of bees (8), and the minimum honey required for the hive to birth new bees (4).

You'll have to figure out good names for each, as well as the types. For types, don't just think about initial values, but also the values these constants will be used with. Doubles pair best with other doubles, ints with other ints.

Write the code to work with **Locations**

First, "Tite the GetLocation() method. It ,hould lak,' in a string, l,ok lip ,h"l string in Ih,-Locat;'"1' di,',;,nary, and return the associ,w'd I'0;nt lf it's ilolllo,n', throw an ArgumentException.

Then, write the InitLalizeLocations() method. Thi, method ,h" "ld set Ill' the following locations in the hive:

- Entrance, at (600, 100)
- Nursery, at (95, 174)
- HoneyFactory, at (157, 98)
- Exit, "I (194, 213)

Each of thes.. maps to a location within the 2D space tnat our hive tak... up Later on, we'll ha to make su-e the simulator mak th.. hive cover all th..... points.

In this simulation, we're just assuming one hive, with fixed points. If you wanted multiple hives, you might make the points relative to the hive, instead of the overall world.

- **Build the Hive constructor**

When a hive is constructed, il should set its h",,,'y to the inilial ,1111'!lUll of honey all hives have. It ,h"uld setup the I".:ali,,", in Ihe hive, and also create a new instance "f Random. Then, AddBee() shcatld be called - passing in the Random instance you jU;1 created - ,nce for each bee lh"l starts "III in the hive,

t AddB··e() needs a Random object b.cause it adds a random value to the Nursery location—that way the bees don't start on top of each other.



Your job was to start building the Hive class

Make sure you add "using Syste... Drawing;" because this code uses Point

```
public class Hive {
  private canst irrt InitialBees - 6;
  private canst double InitialHoney = 3.2;
  private const double Maximum/-loney = 15.0;
  private const double NectarHoneyRatio = .25;
  private const double MinimumHoneyForCreatingBees = 4.0;
  private canst int. Naxrmumbees - 8;

  private Dictionary<string, Point> locations;
  private int beeCount - 0;

  public double Honey { get; private set; }

  prrvat.e void InitializeLocations(l {
    locations = new DtctIonary-cstrInq, Point>();
    locations.Add("Entrance", new Point(600, 100»);
    locations.Add("Nursery", new Point{95, 174));
    locations.Add("HoneyFactory", new Point(157, 98}};
    locations.Add("Exit", new Point{194, 213));

  public Point GetLocation(string location)
    if (locations.Keys.Contains(location))
      return locations[location];
    else
      throw new ArgumentException("Unknown location: " + location);

  public Hivetj (
    Honey = InitialHoney;
    InitializeLocations();
    Random random - new Random();
    for (lnt i. - 0; i < InitialBees; i++)
      AddBee(random);

  public bool AddHoney(double Nectar) { return true;}
  public bool ConsumeHoney(double Amount) { return true; }
  private void AddBee{Random random} I I
  public void Go(Random random) { }
```

You might have different names for your constants. That's okay, as long as you're consistent in the rest of your code.

We made MaximumHoney a double, since it can range from InitialHoney (3.2) to this value. Since InitialHoney will need to be a double, it's best to make this a double, too.

Remember dictionaries? Ours stores a location, keyed with a string value.

p on't forget to create a new instance of Dictionary, or this won't work

The rest of this method is pretty straightforward.

nu. ..ethod protects other classes from working with our locations dictionary and changing something they shouldn't It's an example of encapsulation

You should have called AddBee() once for each bee that a hive starts with.

We don't have code for these yet: but you should have built e...pty methods as placeholde...,

Isn't this sort of a weird way to build code? Our bees don't know about flowers yet, and our hive is full of empty method declarations. Nothing actually works yet, right?



Real code is built bit by bit

It would be nice if you could write all the code for a single class one time, compile it, test it, and then start on your next class. Unfortunately, that's almost never possible.

More often than not, you'll write code just the way we are in this chapter: piece by piece. We were able to build pretty much the entire `Flower` class, but when it came to `Bee`, we're still going to need some work to do (mostly telling it what to do for each state).

And now, with `Hive`, we've got [illegible] methods to fill in, but we haven't hooked any `Bees` up to the `Hive`. And there's still that nagging problem about how to call the `Go()` method in all these objects thousands of times.

But we didn't really start out by putting the classes together! We figured out the architecture first, *and* then started building.

First you design, then you build

We started off this project knowing exactly what we wanted to build: a beehive simulator...we know a little about how bees, flowers, hive, and world all worked together. That's why we started off with the **architecture**, which told us how the classes would work with each other. Then we could move on to each class, designing them individually.

Projects always go a lot more smoothly if you have a good idea of what you're building before you start building it. That's pretty straightforward and common-sense. But it makes all the difference in the final product.



Fillillg out the Hive class

1..t's get back hI the Hive cli ss, and fill in a few "r those· missing methods:

```
public class Hive (
  II constant declarations
  II va riable declarations

  II InitializeLocations()
  /I Get Locet Lon tj
  II Hive constructor

  public bool AddHoneyldouble nectar) I
    double honeyToAdd _ nectar * NectarHoneyRatio;
    if IhoneyToAdd + Honey> MaximumHoney)
        return false;
    Honey += honeyToAdd:
    return true;
  I
  public bool ccneueeacoey(double amount) I
    if (amount > Honey)
      return false;
    else I
      Honey -- amount;
      return true:
  }
  private void AddBee(Random random) {
    beeCount++;
    int rI - random.Next(100) - 50;
    int r2 - random.Next(100) - 50;
    Point startPoint - new Point(locations["Nursery"].X + rL,
                                 locations["Nursery"].Y + r2);
    Bee nevaee … new Bee(beeCount, startPoint};
    II Once we have a system, we need to add this bee to the system
  I
  public void Go(Random random) { )
```

First, we figure out how much honey this nectar can be converted to

and then see if there's room in the hive for that much more honey

If there's room, we add the honey to the hive

This method takes an amount of honey, and tries to consume it from the hive's stores

If there's not enough honey in the hive to meet the demand, we return false

If there's enough, remove it from the hive's stores and return true

This creates a point within 50 units in both the X and Y direction from the nursery location

Th… " private only Hive instances can create bees

Add a new bee, at the designated location

We'll finish AddBee() and fill in the Go() method …

The hive's Go() method

We've already written a Go() method for Flower, and a Go() method for Bee (even though we've got some additional code to add in). Here's the Go() method for Hive:

```
public void Go(Random random)
  if (Honey > MinimumHoneyForCreatingBees)
    AddBee(random);
```

The only constraint (at least for now) is the hive must have enough honey to create more bees.

The same instance of Random that got passed to Go() gets sent to the AddBee() method

Unfortunately, this isn't very realistic. Lots of times in a busy hive, the queen doesn't have time to create more bees. We don't have a QueenBee class, but let's assume that when there's enough honey for new bees a new bee actually gets created 10% of the time. We can model that like this:

```
public void Go(Random random)
  if (Honey > MinimumHoneyForCreatingBees
      && random.Next(10) == 1)
    AddBee(random);
```

This is an easy way to simulate a 1 in 10 chance of a bee getting created. It comes up with a random number between 0 and 9. If the number is 1, then create the bee.

there are no Dumb Questions

One reason to leave it out is so that you can save the Rand_ seed—that way you can re-run a specific simulation if you feel like doing that later!

Q: So the hive can create an infinite number of bees?

A: Right now it can—or, at least, it's got a very large limit—but you're right, that's not very realistic. Later on, we'll come back to this, and add a constraint that only lets so many bees exist in our Simulator world at one time.

Q: Couldn't we assign that instance of Random to a property of the class, instead of passing it on to AddBee()?

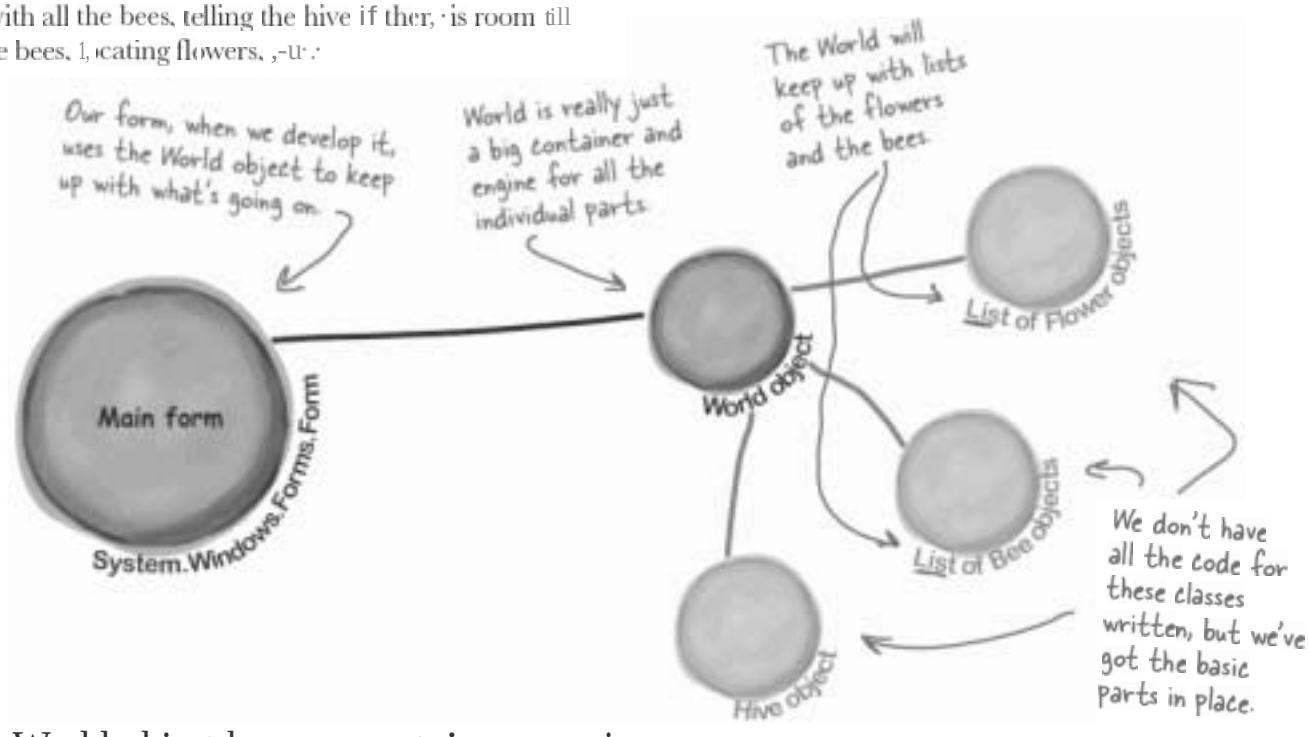
A: You sure could. Then AddBee could use that property, rather than a parameter passed in. There's not really a right answer to this one: it's up to you.

Q: I still don't understand how all of these Go() methods are getting called.

A: That's okay, we're just about to get to that. First, though, we need one more object: the World class, which will keep track of everything that's going on in the hive, track all the bees, and even keep up with flowers.

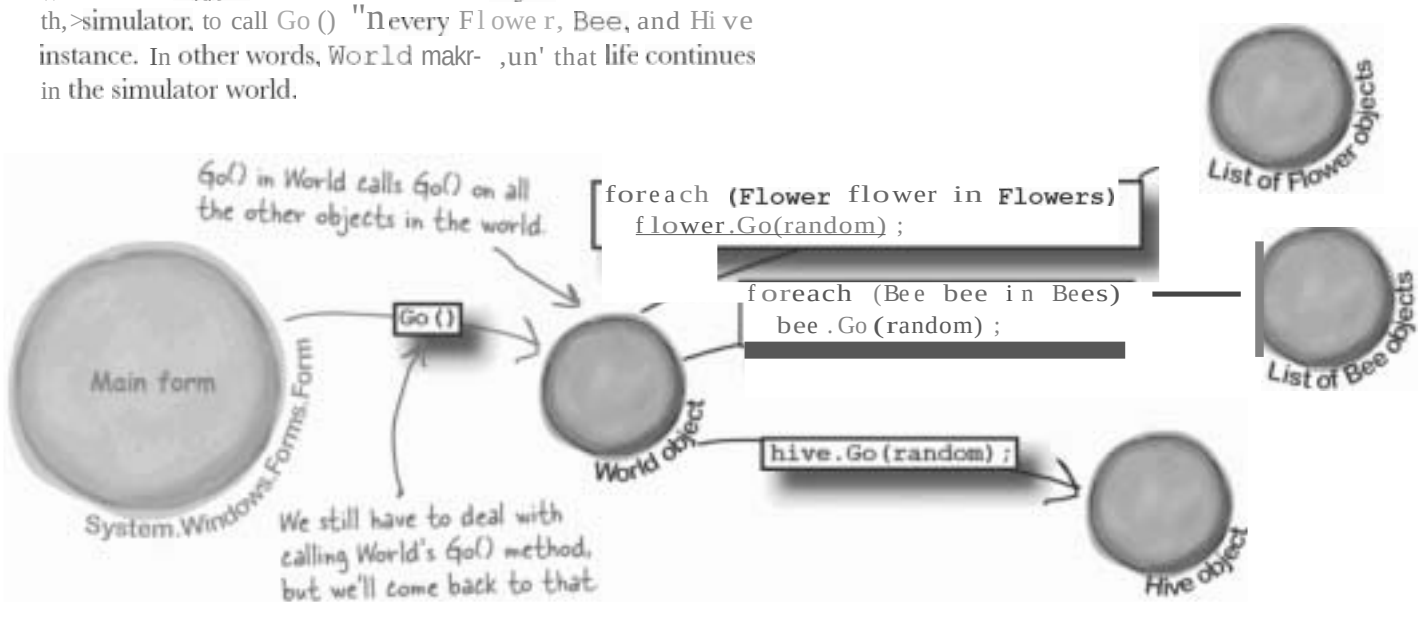
We're ready for the World

With a Hive, Bee. and Flower class in place, we can finally build the World class. World h.mdk- coordination between all the individual piece or our simulaor: keeping up with all the bees, telling the hive if ther, · is room till more bees, l,cating flowers, ,-u·.·



Ihe World objeat keeps everytniog `Go()` iog

()n,' of the biggest tasks of th,' `World` object is, for each turn ill th,>simulator, to call `Go()` "n every `Flower`, `Bee`, and `Hive` instance. In other words, `World` makr- ,un' that life continues in the simulator world.

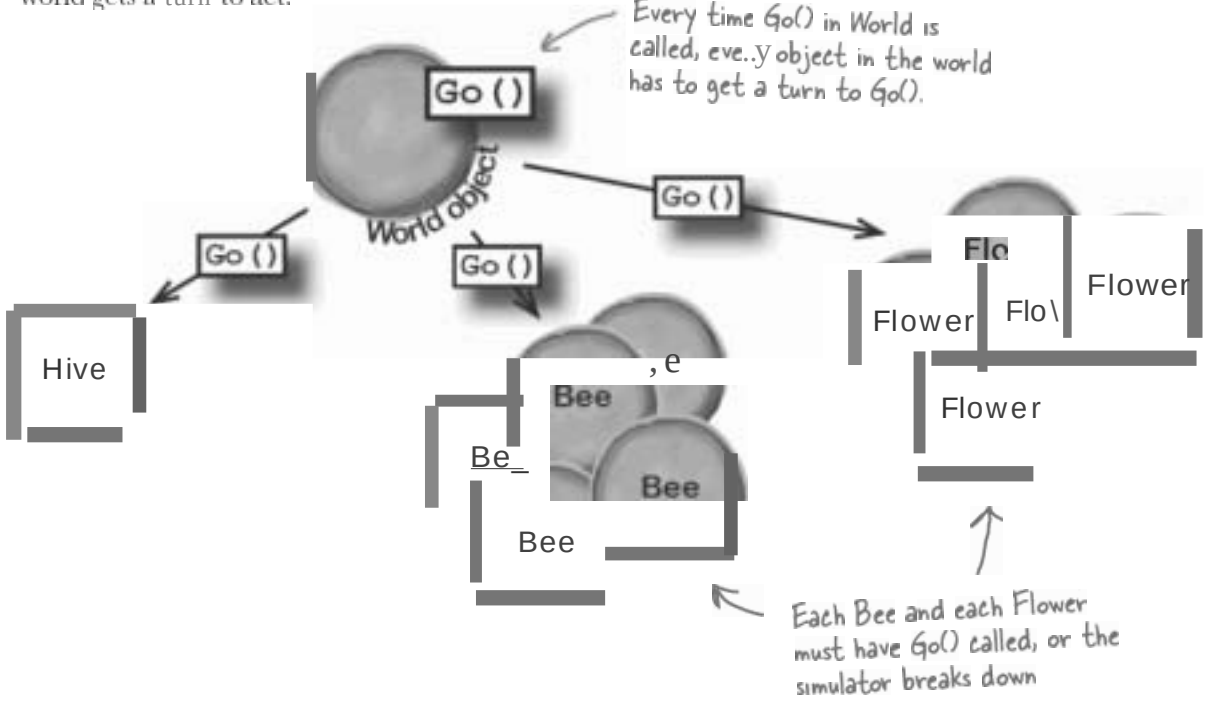


We're building a turn-based system

(Jill' Go() methods in each object are supposed to run each tnrn. 01' **cycl**.., uf ou simulator. A turn in ,hi, case ju" means "'1 arbitrary amrmnt „1' rim!"... for instance, a Iurn could Ile every 1()seconds, '>1'every 60 seconds. ,,,' every 1()miunle'.

Each "turn" will be drawn as a single frame of animation, so the world only needs to change a tiny litle bit each turn.

'I'll" main thing i, thai a turn affects every object ill the world. Tll,-hi\'(,ages by one "turn", checking to see if it needs (<0 add more bees. Then each be.. takes a I\1rn. moving a very small distance towards *it,* d,-,tinal jon, '1' doing one 'mall acl;,m, and getting <,\d,"T'. Then each ɲower l"k,-, a turn, manufacturing a little nectar and getting ‹,Id"r (,)o. And that's what Wo r1d does: ;Im"k,", sure ,h"t every time its Go() method i, called, every object in lh" world gets a turn to act.



Sharpen your pencil

One of the big object-oriented principles we've been using in the simulator is encapsulation (flip back to Chapter 5 for a refresher). See if you can look over the code we've developed so far and come up with two examples of encapsulation for eech class you've built.

Hive	Bee	Flower
1.	1.	1.
2 ..	2 .	2.

Here's the code for World

The World class is actually "the glue" that holds the simpler classes in our simulator. Here's the code:

```
using System.Drawing;
public class World {
    private const double NectarHarvestedPerNewFlower = 50.0;
    private const int FieldMinX = 15;
    private const int FieldMinY = 177;
    private const int FieldMaxX = 690;
    private const int FieldMaxY = 290;

    public Hive Hive;
    public List<Bee> Bees;
    public List<Flower> Flowers;

    public World() {
        Bees = new List<Bee>();
        Flowers = new List<Flower>();
        Random random = new Random();
        for (int i = 0; i < 10; i++)
            AddFlower(random);
    }

    public void Go(Random random) {
        Hive.Go(random);

        for (int i = Bees.Count - 1; i >= 0; i--) {
            Bee bee = Bees[i];
            bee.Go(random);
            if (bee.CurrentState == BeeState.Retired)
                Bees.Remove(bee);
        }

        double totalNectarHarvested = 0;
        for (int i = FlowersCount - 1; i >= 0; i--) {
            Flower flower = Flowers[i];
            flower.Go();
            totalNectarHarvested += flower.NectarHarvested;
            if (!flower.Alive)
                Flowers.Remove(flower);
        }
```

These define the bounds of the field, which is where flowers can live

Every world has one hive, a list of bees, and a list of flowers.

When we create a new world, we initialize our lists, create a new hive, and then add 10 initial flowers.

This is easy.. we just tell the Hive to Go(), pass-in a Random instance

We run through all the current bees and tell them Go().

If a bee's retired, we need to take it out of the ___

We run through each flower and tell it to Go()

We need to keep up with how much nectar's been collected this turn, too. So we get that by summing up the nectar collected from each flower.

Just like bees we remove any flowers that die during this turn.

Sharpen your pencil Solution

One of the big object-oriented principles we've been using in the simulator is "ncapsulation (flip back 10 Chapter S for a feflesher). See if you can look over the code *we've* developed so far and come up with two examples of encapsulation for each class you've buiIl.

Here are the ones we ca..e up with. D,d *r-* come up with any others?

Hive

1. The hive's Locations dictionary is private
2. It gives the bees a method to add honey

Bee

1. The bee's location is read-only
2. So is its age. So other classes can't write to them

Flower

1. The flower provides a ...ethod to gather nectar
2. And it keeps its alive boolean private

```
if (totalNectarHarvested > NectarHarvestedPerNewFlower) {
    foreach (Flower tLower- in Flowers)
        flower.NectarHarvested = 0;
    AddFlower(random):
}
```

Bees po"inate flowe-s as they harvest nectar. Once they've harvested enough nectar from the flowers, they've pollinated enough for the world to add a new flower.

If there's enough necta·· in the field, true, the world adds a new flower.

```
private void AddFlower(Random random)
{
    Point location = new Point(random.Next(FieldMinx, FieldMaxX),
                               random.Next(FieldMinY, FieldMaxY));
    Flower newFlower - new Flower(location, random);
    Flowers.Add(newFlower);
}
```

This handles coming up with a ..andom location in the field ..

...and then adding a new flower in that location.

there are no Dumb Questions

0_: Why don't **you use foreach loops** to remove dead flowera and **retired bees?**

A: Because you can't remove **items** from a collection from inside a foreach cop that's iterating on it If you dO,NET1!'11 1throwan exception.

0_: Okay, then **why does** each 01**those** for **loops start at** Ille end of tile lil t and count down 10 **0?**

A: Because eacntcco needs to preserve the numbering of ee hst. Let's sayyou started at the beginning of a list of five tees. and your loop discovered that oneof the flowers in the middle was dead. If it removes the flower at index #3, nowthe list only has 4 flowers in it, andthere's a new flower at index #3—and thal b er will end upgetting skipped, because ite next time through the jeep it'll look at index #4

If the ceo starts at theend. then the flower that moves into the empty slot will already have been looked at by the cop . so there's no chance of missing a flower



With all lour of our core classes in place, we've got some work to do to tie them all together. Follow the steps below, and you should have working Bee, Hive, Flower, and World classes. But beware: you'll have to make changes to almost every class, in several places, before you're done

O **Update Bee to take** in a **Hive and** World **reference.**
Now that we've got a class lor Hive and a class lor WorId, Bee objects need to know about both, Update your code to take In rererences to a bee's hive and world in the constructor and save those refe rences for later use.

❷ **Update Hive to take** in **a** World **reference.**
Just as a Bee needs to know about Its Hive, a Hive needs to know about its World, Update Hive to take In a World reference In Its constructor, and save that reference, You should also update the code in Hive that creates new bees to pass into the Bee a reference to itsell (the Hi ve), and the World.

❸ **Update** World **to pass itself into a new Hive.**
Update your World crass so that when it creates a new Hive, It passes in a reference to nsen.

STOP! AI this poinl, you should be able 10 compile all of your code. If you can't, check t1I1'Olligh it and correct any mistakes before continuing on.

O **Place** an **upper limit** on **the bees that Hive** can **create.**
The Hive crass has a MaximumBees constant that eeiermoee how many bees the Hive can support (inside and outsrce the hive, comb ined). Now that the Hive has access to the World, you shoul d be able to enforce that con straint

Hint: Look at code near where you create or add bees. There are two places where code related to this occurs in Hive, so be careful.

O **When the Hive creates bees, let the** World **know.**
The wono class keeps up With all the bees that east. When the Hive creates a new Bee, make sure that Bee gets added \0 the overall list that the wcrrd is keeping up with ,

there are no Dumb Questions

Q: Why did you throw an exception in the Hive class's GetLocationU method?

A: Because we needed a way to deal with bad data passed into the parameter. The hive has a few locations, but the parameter to GetLocations() can pass any string. What happens if there's a bug in the program that causes an Invalid string (like an empty string, or the name of a location that's not in the locations dictionary) to be sent as the parameter? What should the method return?

When you've got an invalid parameter and il's not clear what to do with it, it's always a good idea to throw a new ArgumentExceptIon, Here's how the GelLocalionO method does it

```
throw new ArgumentException(
      "Unknown location: " + location);
```

This statement causes the Hive ce ss to throw an ArgumentException With the message "Unknown location:" that contain s the location that it couldn't find

rte reason this is useful is that it immediately alerts you if a bad location parameter is passed to the method And by including the parameter in the excepn cn message, you're giving yourself a some valuable information that will help you debug the problem

Q: What's the point of storing alt the locations in a Point object if we're not drawing anything?

A: Every bee has a rccancn, whether or not you draw It on the screen m meu ocaton The job ot the Bee object is to keep track otwtere it is in the world Each time its Go () method is called, it needs to move a very small mstance towards its eeseneuon

Now, even though we may not be drawing a picture of the bee yet, the bee still needs to keep track of where It is inside the hive or In the field, because it needs to know if it's arrived at its destination.

0_: Then why USe Point to store the location, and not something else? Artn ', Point s specifically for drawing?

A: Yes, a Point is what all 01 the visual controls use for their Lccatcn properties. However, just tecese .NET uses them that way, that doesn't mean it's not also useful for us to keep track of locations Yes, we could have e eered our own BeeLocation cess wilh integer fields called X and Y. No reason re nvera ee wheel when C# and .NET give us Point for free!

It's almost always easier to **repurpose** or **extend** an existing class Ihal **does MOSTLY** wkat you **want it** to Jo. ratker- **than creating** an all **new** class from **scratch.**



With all four of our core classes in place, we've got some work to do to tie them all together. Follow the steps below, and you should have working Bee, Hive, Flower, and World classes. Here's how we made the changes to put us into place.

Update Bee to take in a Hive and World reference.

Now that we've got a class for `Hive` and a class for `World`, `Bee` objects need to know about both. Update your code to take in references to a bee's hive and world. In the constructor and save those references for later use.

```
public class Bee {
  // existing constant declarations
  // existing variable declarations
  private World world;
  private Hive hive;

  public Bee(int ID, Point InitialLocation, World world, Hive hive) {
    // existing code
    this.world = world;
    this.hive = hive;
```

This is pretty straightforward... take these in, assign them to local variables.

Update Hive to take in a World reference.

Just as a `Bee` needs to know about its `Hive`, a `Hive` needs to know about its `World`. Update `Hive` to take in a `World` reference in its constructor, and save that reference. You should also update the code in `Hive` that creates new bees to pass into the `Bee` a reference to itself (the `Hive`), and the `World`.

```
public class Hive {
  private World world;

  public Hive(World world) {
    this.world = world;
    // existing code
```

More basic code: get the reference, assign it locally. You want to assign the world FIRST because the rest of the constructor needs to use it.

```
  public void AddBee(Random random) {
    // other bee creation code
    Bee newBee = new Bee(beeCount, startPoint, world, this);
```

New bees need a reference to the world, and to the hive, now.

If you're having trouble getting this running, you can download the code lor this exercise (and atlthe etlrers, 100) from http://www.headfirstlal.s.com/l"oks/hfcsharp/

3 Update Warld to pass itself into a **new Hive**.

Update your World class so that when it creates a new Hive, it passes in a reference to itself.

```
public World() {
  Bees = new List<Bee>();
  Flowers = new List<Flower>();
  Hive = new Hive(this);
  Random random = new Random();
    for (int i = 0; i < 10; i++)
      AddFlower(random);
```

this passes in the ..eference to the Hive

O **Place** an **upper limit** on **the bees** that **Hive** ccn create.

The Hive class has a MaximumBees constant that determines how many bees the Hive can support (inside and outside the hive, combined). Now that the Hive has access to the World, you should be able to enforce that constraint

```
public VOId Go{Random random} {
  if (wor a-ass-cseae < MaximumBees
      && Honey > MinimumHoneyForCreatingBees
      && random.Next(10) __ 1)
    AddBee(random);
```

We can u the World object to e how ..anY total bees there are, and compa e that to the maximum bees for this hive.

We put that comparison first. If the e's no room for bees, no sense in seeing if the e's enough honey to create bees.

5 **When the Hive** c-ectes **bees, let the** World know.

The World class keeps up with all the bees that exist. When the Hive creates a new Bee, make sure that Bee gets added to the overall list that the World ISkeeping up With.

```
private void AddBee(Random random)
  beeCount++:
  // Calculate the starting point
  Point startPoint = // start the near the nursery
  Bee newsee = new BeeIbeeCount, startPoint, world, this):
  world.Bees.     (newBee);
```

We add the new bee to the world's overall bee ,

This demonstrates one of ..eference in the Hive class.

Giving the bees behavior

The one big piece "r code that's missing in mir current classes is the Bee's Go () method. We were able 10code a few "l" lho' states earlier, IJIII 11ere are plenty left (Idle i, i"ulIl1l'kt,', FIyingToFlower, and part of MakingHoney).

Let's finish up those remaining states now:

```
public void Go(Random random)
    Age++;
    evitcti (CurrentState) {
        case BeeState.Idle:
            if (Age > CareerSpan)
                CurrentState - BeeState.Retired;
            else if (world.Flowers Count> 0
                && hive.ConsumeHoney(HoneyConsumed))
                Flower flower ..
                world.Flowers[random,Next(world. Flowers.Count)];
                if (flower.Nectar >= MinimumFlowerNectar && flower.Alive) {
                    destinationFlower .. flower;
                    CurrentState .. BeeState.FlyinqToFlower;
            }
            break;
        case BeeState.FlyingToFlower:
            if ('world.Flowers.Contains(destinationFlower))
                CurrentState .. BeeState,ReturningToHive;
            else if (lnsideHive) I
                if (MoveTowardsLocation(hive.GetLocation("Exit")))
                    InsideHive = false;
                location .. hive.GetLocation("Entrance");
                )

            else
                if (MoveTowardsLocation(destinationFlower.Location»
                    eurrentState .. BeeState.GatheringNectar;
            break;
        case BeeState.GatheringNectar:
            double nectar - destinationFlo\<ler.HarvestNectar();
            if (nectar > 0)
                NectarCollected t- nectar;
            else
                CurrentState - BeeState.ReturningToHive;
            break;
```

If we're idle, we want to go find another flower to harvest from.

See if there are flowers left, and then consume enough honey to keep on going. Otherwise, we're stuck

We need another living flower with nectar

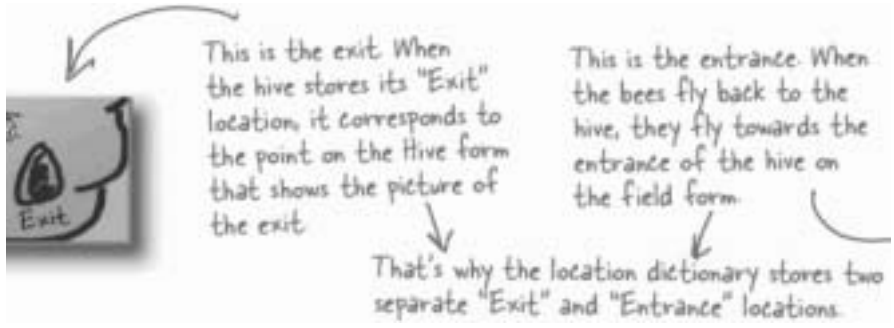
Ass.....ing that all works out, go to the new flower.

Make ..... the flower hasn't died as we'.... heading toward it

That's why we passed a reference to the hive to the Bee const..uctor

If we can get to the exit, then we're out of the hive. Update our location, Since we're now on the field form, we should fly out near the entrance

If we're out of the hive and the flower is alive, get to it and sta..t. gathering nectar.





```
case BeeState.ReturningToHive:
    if (!InsideHive) {
        if (MoveTowardsLocation(hive.GetLocation("Entrance"))) {
            InsideHive = true;
            location = hive.GetLocation("Exit");
    else
        if (MoveTowardsLocation(hive.GetLocation("HoneyFactory")))
            CurrentState = BeeState.MakingHoney;
    break;
case BeeState.MakingHoney:
    if (NectarCollected < 0.5) {
        NectarCollected = 0;
        CurrentState = BeeState.Idle;

    else
        if (hive.AddHoney(0.5))
            NectarCollected -= 0.5;
        else
            NectarCollected = 0;
    break;
case BeeState.Retired:
    // Do nothing! We're retired!
    break;
```

If we've made it to the hive, update our location and the insideHive status

If we're already in the hive, head to the honey factory

Try and give this nectar to the hive.

If the hive could use the nectar to make honey...

...remove it from the bee.

If the hive's full, AddHoney() will return false, so the bee just dumps the rest of the nectar so he can fly out on another mission.

Once the bee's retired, he just has to wait around until the Hive removes him from the list. Then he's off to Miami!



Suppose you wanted to change the simulator so it took two turns to reach a flower, and two turns to go from a flower back to the hive. Without writing any code, which *methods* of which classes would you have to change to put this new behavior into place?

Ihe **main form** tells **the** world to **Go()**

Okay, ◊' you know that the world advance by one frame every turn its Go () method is called. lIut what calls that Go () method? "'hl'. the main form, 0f course! Time 1o lay it out.

(;" ahead and add a new form to your project. Make it look lik,· Ih,· form below. We're using some new controls, h"1 we'll explain rh-ru all over the next several pages

The ToolStrip control puts a toolstrip at the top of your form You can add the two buttons using the dropdown that appe on the f....., Set each tton's DisplayStyle to Text



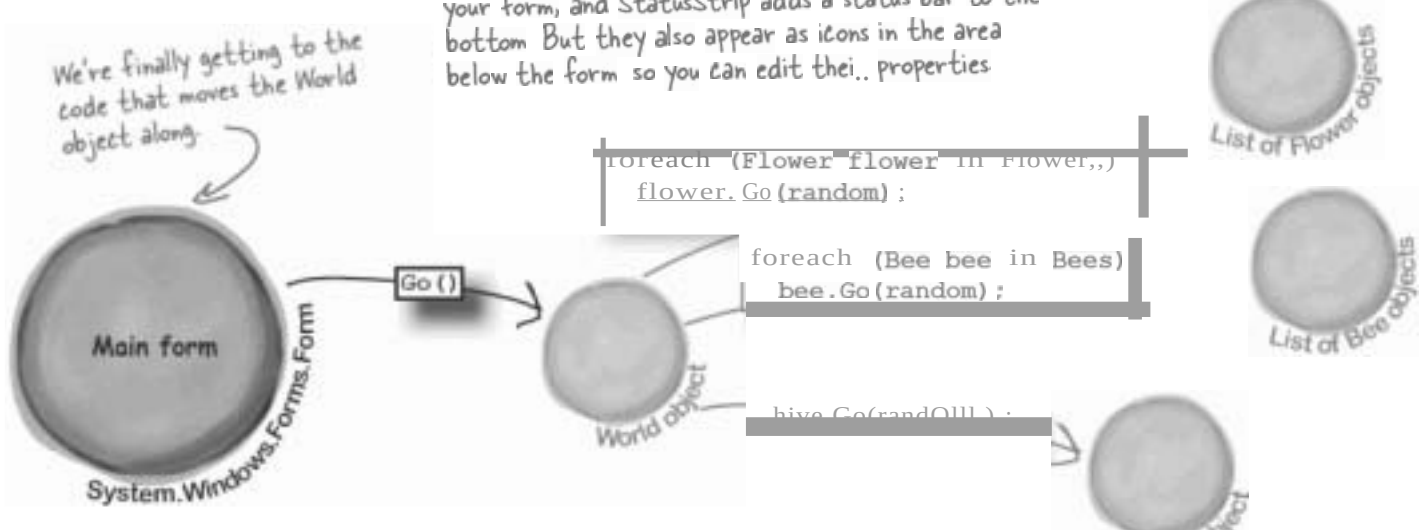
Each of these labels lives ,o. one cell of a TableLayoutPa-,el control. You lay it out just like a table in Microsoft Word Click on the little black arrow to add, ve..ove, and resize columns and ..ows

Add a StatusStrip to put a status bar on the bottom. Use the dropdown to add a StatusLabel to it

Add a T;..er control to the f..... It doesn't show up at all—it's a non-visua1 component that the form designer displays as an icon in the space below the form

The ToolStrip control adds a toolbar to the top of your form, and StatusStrip adds a status bar to the bottom But they also appear as icons in the area below the form so you can edit thei.. properties

We're finally getting to the code that moves the World object along



We call use World to get statistics

Now v ·w·'nt to upd;.... .,II, hese controls. But we don·1 neec1 click handle fi,r, ach one; in,t, ·.,,!. let's use a single method tha i" ill "I',bl" the different statistics ill the ,i Ul1t1.,I"l' window:

This indicates how long passes for a turn we'll have to send this parameter ,n from som·here else, m just a fe.. pages

Most of this j...t involves getting data from the world and updating labels

```
private void updateStetSttImeSpan fremeuuratLonl {
    nees,rext _ world.Bees.Count.ToString();
    Flowers.Text .. worId,Flowers.Count.toStrlng();
    HoneyInHive,Text - srring.Forssat.t"{0:f)l", vorLd,Hive.Honey);
    double nectar - O.
    toreach (Flower flower in world.Flowers)
        nectar += flower.Nectar;
    NectarInflowers.Text • String.Format("{0:f3}", nectar);
    r'remesuun,Text - rrameagun.ToStnhg();
    double milliSeconds · frameDucation.TotalMilllseconds;
    if (milliSeconds != 0.0)
        FrameRate.Text   string.Format("{0:f0} ({1:f1}ms)",
                    1000 / milliSeconds, lllilliSeconds);
    else
        FrameRate.Text = "N/A";
}
```

Be sure you match your label names on the form with your code.

Print the first parameter as a number with no decimals, then a space, then print the second parameter with one decimal followed by the letters "ms·C.. parentheses)

The frame ..ate is the number of frames run per second We're u·ing a TimeSpan object to store ho.. long it took to run the fra..e We divide 1000 by the ber of ..illiseconds it took to run the fra —that gives us the total number of milliseconds it took to run the last frame

Add this method into Forml.

Whoa! Where did that World object come from... we haven't created that yet. have we? And what's all that time and frame stuff?

o

o



We'll talk more about this when we create that TimeSpan object.

n", code uses the same String.For..at() ..ethod you used in the hex dump But instead of printing in hex using "x2", you use "f3" to display a number with three decimal places.

Let's creat. a World

Y", 'rt' right, we neec110 create th.· World object. .Add Ihi, li,ue ", your form's constructor:

```
public Forml(I 1
  InitializeComponent();
  world _ new World(l:
```

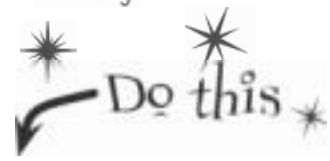
Go ahead and ",kl a private WorId nekll.. your form called worId.

That jll" leaves "ll the linue·n·L,Ixc1cod... W..'ve always said ",. needed a way ttt "Ull Go () in Wor1d '"'"rand ,,'"r... sounds like we need some sort of limer.

Take a ..inute and create a new project so you can see how ti..ers work. Then we'll get back to the si..ulator and put your new knowledge to work

filllers fire **events** over **and** over **again**

Remember how you used a loop to animate the greyhounds? Well, there's a better way to d" it. .\ **Timer** is an especially useful component that triggers an event over 'llld over again, lip to a thousand rillll"' a second.



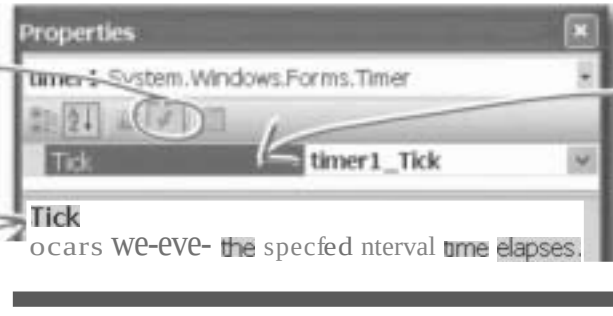
1 **Create** Q **new project with** Q **timer and three** buttons

Y(" don"l have to close yc,Ul ,'UIT 'I)I project just pc'I' open a new \ 'isual Studic and start up a new projcc(, Drag a timer and thn-ebuttonv onto the form. Cli,-k on the time-r icon at the bottom "f lh,-designer and set ii' Interval property (0 1000. That number j, measured in milliseconds- il tells the limer (0 fire its tick event •nce a sec("""l.

2 **Open the** IDE's **Properties window and click** on the Events button.

(Remember, the Events button],ooks like a lightning]" olt, and it lets you manage the events for any "f yc«u- form's controls.) The limer contrc,] has exactly one event, Tick. **Click** on Ih**e Time**r icon in the designer, then d"uhl**e-click on** il s **row** in the Events page,, ,,,"I the 1I)E will create a new eve'm handler method for you and h,,uk il up to the property automatically.

The Events button in the Properties window lets you work with all the events for each of your controls.



The Timer control has one event called Tick. If you double-click here, the IDE creates an event handler method for you automatically.

The bottom of the window has a description of the event.

3 Add code to **the Tick** event and to your buttons,

Here's some code that will help you get a sens(or how the timer works:

```
private void timerl_Tick(object sender, EventArgs e) {
    Console.WriteLine(DateTime.Now.ToStIing());
}

private void toggleEnabled_Clid(object sender, EventArgs e) {
    if (timer1.Enabled)
        timed.Enabled = false;
    else
        timerl.Enabled = true;
}

private void startTimer_Click(object sender, EventArgs e){
    timed.Start n.
    Console.WriteLine("Enabled = " + timerl.Enabled);
}

private void stopTlmer Click(objJect sender, EventArgs e) {
    timerl.Stop();
    Console.Writ eLine("Enabled · .. · timerl.Enabled);
}
```

T' state··ent writes the cu ent date and time to the tput. Check the output window to ..ake sure the tick event is fired once a second (every 1000 m lliseconds).

Tkese buttons let) you play with the Enabled property and the Start() and Stop() ..ethods. The first one switches E bled between t e and false, and the other two call the Start() and Stop() ..ethods.

The time..'. Enabl(d property starts and stops the ti..er.

The timer's Sta..t() method starts the timer and sets Enabled to true The Stop() method stops the time- and sets Enabled to false

Ihe till\er's using a delegate behind the scenes

The ti..er's Tick event is an average, everyday event handler, just like the ones to han dle button clicks

How do C# and .NET tell the timer what tc do ever tick? How does the `timerI_Tick()` method get run every time your time-r ticks? Well, we're back to **ev..nls** and d..l..gat..... ju,1 lik,' we talke-d about inth.- last chapter. Use the IDE's ..t;" To 1)" fill i(i,," " feature 10 remind yourself how th,' Event Handler delegate works:

O **Right-click on your timer1 variable** and seteet "Go To **Definition"**

The ..(Jo To Ild;" il;'>il" r,;uII!'t· will cause (h,' Il IE (0 automatically jll"'p to the locatiorIill the c("l,· wh,"" the `timer1` variable is defined. Tl,,' 1\) E will jUIIlp you (▷ th,' code il created tll add `timer1` as a property in the F("-,,,I object in R,rml.1)esigner.cs. SIT ,ll "pin th,' Iii,' IIIIlil y'" find this Ii"e:

```
this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
```

This is the Tick event of your timer control. You've set this to occur every 1000 milliseconds.

Here's one of the System's delegates: the basic event handler. It's a delegate... a pointer to one or more methods.

Here's the method you just wrote, timer1_Tick(). You're telling the delegate to point to that method.

O Now **right-click** on **EventHandler** and eeleet "Go To **Definition"**

The IllE will automatically jump to the code that defines `EventHandler`. Take a loc,k at (he name "I' new tab that i1opened to show you the code: "EventHandler [from metadata]". Thi, means that the code (o define EventHandler isn't ill your code. It's b"ilt into the .NET framework, ,IIld the IDE generated a ..fake·· lin'· or code to show you how it's represented:

```
public delegate void EventHandler(object sender, EventArgs e);
```

Each event is of type EventHandler. So our Tick event now points to the timer1_Tick() method.

Here's why every event in C# generally takes an Object and EventArgs parameter—that's the form of the delegate that C# defines for event handling.



What code would **you** write to run the World's GoO method every 500 milliseconds in our beehive simulator?

Add a Timer to the simulator

Let's add a timer to the simulator. You've already got a timer control, probably called timer1. In the designer, you can generate a timer1_Tick() method, though, we can wire the timer's event handler up manually:

TimeSpan has properties like Days, Hours, Seconds, and Milliseconds that let you measure the span in different units.

DateTime & TimeSpan

.NET uses the DateTime class to store information about a time, and its Now property returns the current date and time. If you want to find the difference between two times, use a TimeSpan object: just subtract one DateTime object from another, and that'll return a TimeSpan object that holds the difference between them.

```
public partial class Form1 : Form {
  World world;                          // You should have a World property from earlier
  private Random random = new Random();
  private DateTime start = DateTime.Now;   // These will be used to figure out how long the simulator's been running at any given point
  private DateTime end;
  private int framesRun = 0;               // We want to keep up with how many frames—or turns—have passed

  public Form1() {
    InitializeComponent();
    world = new World();

    timer1.Interval = 50;                  // Run every 50 milliseconds
    timer1.Tick += new EventHandler(RunFrame);   // We set the handler to our own method, RunFrame
    timer1.Enabled = false;                // Timer starts off.
    UpdateStats(new TimeSpan());           // We also start out by updating stats, with a new TimeSpan (0 time elapsed).
  }

  private void UpdateStats(TimeSpan frameDuration) {
    // Code from earlier to update the statistics
  }

  public void RunFrame(object sender, EventArgs e) {
    framesRun++;
    world.Go(random);                      // Increase the frame count, and tell the world to Go()
    end = DateTime.Now;
    TimeSpan frameDuration = end - start;  // Next, we figure out the time elapsed since the last frame was run.
    start = end;
    UpdateStats(frameDuration);            // Finally, update the stats again, with the new time duration.
  }
```

A second is 1000 milliseconds so our timer will tick 200 times a second

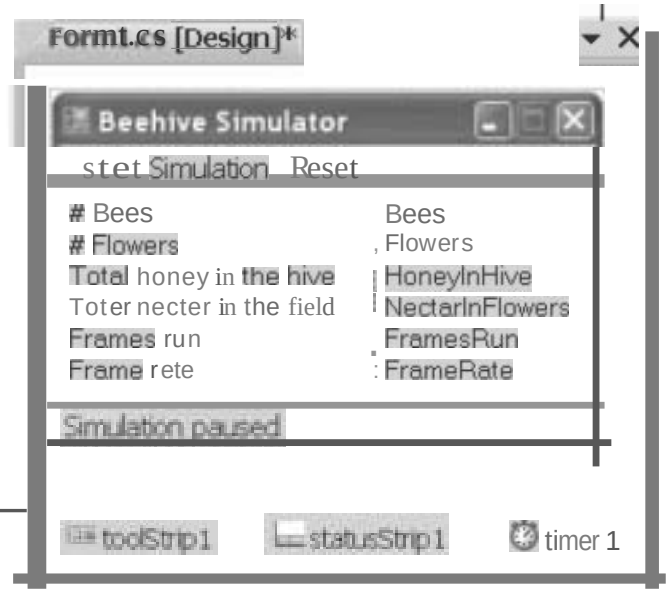


Your job is to write the event handlers for the Start Simulation and reset buttons in the ToolStrip. Here's what each button should do:

Initially, the first button should read "Start Simulation." Pressing it causes the simulation to start, and the label to change to "Pause Simulation." If the simulation is paused, the button should read, "Resume simulation."

The second button should say "Reset." When it's pressed, the world should be recreated. If the timer is paused, the text on the first button should change from "Resume simulation" to "Start Simulation."

If you haven't dragged a ToolStrip and StatusStrip out of the toolbox and onto your form, do it now.



There's no single answer to this question—we just want you to think about what's left to do.

Sharpen your pencil

What do you think is left to be done in this phase of the simulator? Try running the program. Write down everything you think we'll need to take care of before moving on to the graphical stuff.

there are no Dumb Questions

Q: We've been using the term "turn," but now you're talking about frames. What's the difference?

A: Semantics, really. We're still dealing in turns—little chunks of time where every object in the world gets to act. But eventually we'll soon be putting some heavy-duty graphics in place, we've started using "game", as in a graphical game's frame rate.



Your job was to write the event handlers for the Start Simulation and Reset buttons.




```
public partial class Form1 : Form {
  // variable declarations

  public Form1() {
    InitializeComponent();
    world = new World();

  private void Form1_Load(object sender, EventArgs e)
    // code to start simulator

  private void UpdateStats(TimeSpan frameDuration)
    // Code from earlier to update the statistics
  }
  public void RunFrame(object sender, EventArgs e)
    // event handler for timer
```

```
private void startSimulation_Click(object sender, EventArgs e) {
  if (timer1.Enabled) {
    toolStrip1.Items[0].Text = "Resume simulation";
    timer1.Stop();
  } else {
    toolStrip1.Items[0].Text = "Pause simulation";
    timer1.Start();
  }
}

private void resetButton_Click(object sender, EventArgs e) {
  framesRun = 0;
  if (timer1.Enabled)
    toolStrip1.Items[0].Text = "Start simulation";
}
```

Be sure your form's control names match up with what I use in your code.

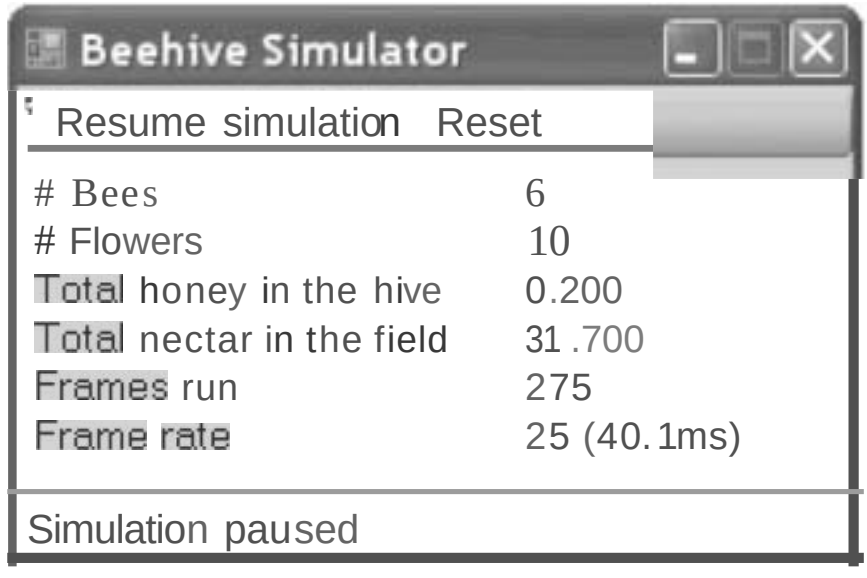
Toggle the timer, and update the message.

Resetting the simulator is just a matter of recreating the World instance and resetting framesRun.

The only time we need to change the first button's label is if it says, "Resume simulation." If it says, "Pause simulation," it doesn't need to change

Test drive

You've done a run of work. Compil,-you code, fix '1[1\' typos, and run the simulator. How's it look?



Looks pretty good! All these numbers should update as the world moves along.

Your start/pause and reset buttons should all work

Hmmm... our status strip seems to be the only thing not working.



Here's your chance 10 pUl10gelher everything you've learned. *We* need to allow bees 10 lell our simulator whallhey're doing. When Ihey do, *we* want our simulator to update the ,tatus message in the simulator.

This time, it', up to you to not only w, ite mOlt of th.. code. but figure o ut what code you need to write. How can you have a method in your simulato. that gets called eve.ytime a bee changes its state?

To give you a little help, we've written Ihe method to add to the form. The Bee class ,hould call this method any time it, state changes:

* Okay, one more hint. You'll need to make changes to all but one of your classes to make this work.

```
private void SendMessage(int ID, string Message)
  statusStrip1.Items[0].Text = "Bee #" + ID + ": " + Message;
```



Exercise Solution

Your job was to come up with a way for Bees to let the simulator know about what they're doing.

Here's what we added to the Bee class.

We used a **callback** to hook each individual bee object up to the form's SendMessage() method.

```
public class Bee {
  // all our existing code
  public delegate void BeeMessage(int ID, string Message);
  public BeeMessage MessageSender;

  public void Go(Random random)
    Age++;
    BeeState oldState = CurrentState;
    switch (currentState) {
     // the rest of the switch statement is the same
    }
    if (OldState != CurrentState
        && MessageSender != null)
          MessageSender(ID, CurrentState.ToString());
```

BeeMessage is our delegate. It's also a match with the SendMessage() method we wrote in the form.

If the status of the Bee changed, we call back the method our BeeMessage delegate points to, and let that method know about the status change

Here are the changes we made to the Hive

```
public class Hive {
  // all our existing code
  public Bee.BeeMessage MessageSender;

  public Hive(World world, Bee.BeeMessage MessageSender) {
    this.MessageSender = MessageSender;
    // existing constructor code


  public void AddBee(Random random)
    // existing AddBee() code
    Bee newBee = new Bee(beeCount, startPoint, world, this);
    newBee.MessageSender += this.MessageSender;
    world.Bees.Add(newBee);
```

Hive needs a delegate too, so it can pass on the methods for each bee to call when they're created in AddBee()

AddBee() now has to make sure that each new bee gets the method to point at.

Hive required some changes as well

```
public class World {
  // all our existing code

  public World(Bee.BeeMessage messageSender)
    Bees - new List<Bee>();
    Flowers - new List<Flower>();
    Hive = new Hive(this, messageSender);
    Random random - new Random();
    for (int i - 0; i < 10; i++)
        AddFlower(random);
```

World doesn't need to have a delegate of its own. It just passes on the method to call to the Hive instance

Last but not least, here's the updated form. Anything not shown stayed the same.

```
public partial class Formi : Form {
  // variable declarations
  public Formi() {
    InitializeComponent();
    world _ new World(new Bee.BeeMessage(SendMessage));
    // the rest of the Formi constructor

  private void reset_Click(object sender, EventArgs e) I
    framesRun = 0;
    world = new World(new Bee.BeeMessage(SendMessage));
    if (!timer1.Enabled)
        toolStripl.Items[O].Text _ "Start simulation";

  private void SendMessage(int IO, string Message)
    statusStripl.Items[OJ.Text - "Bee #" + IO + "· " + Message;
```

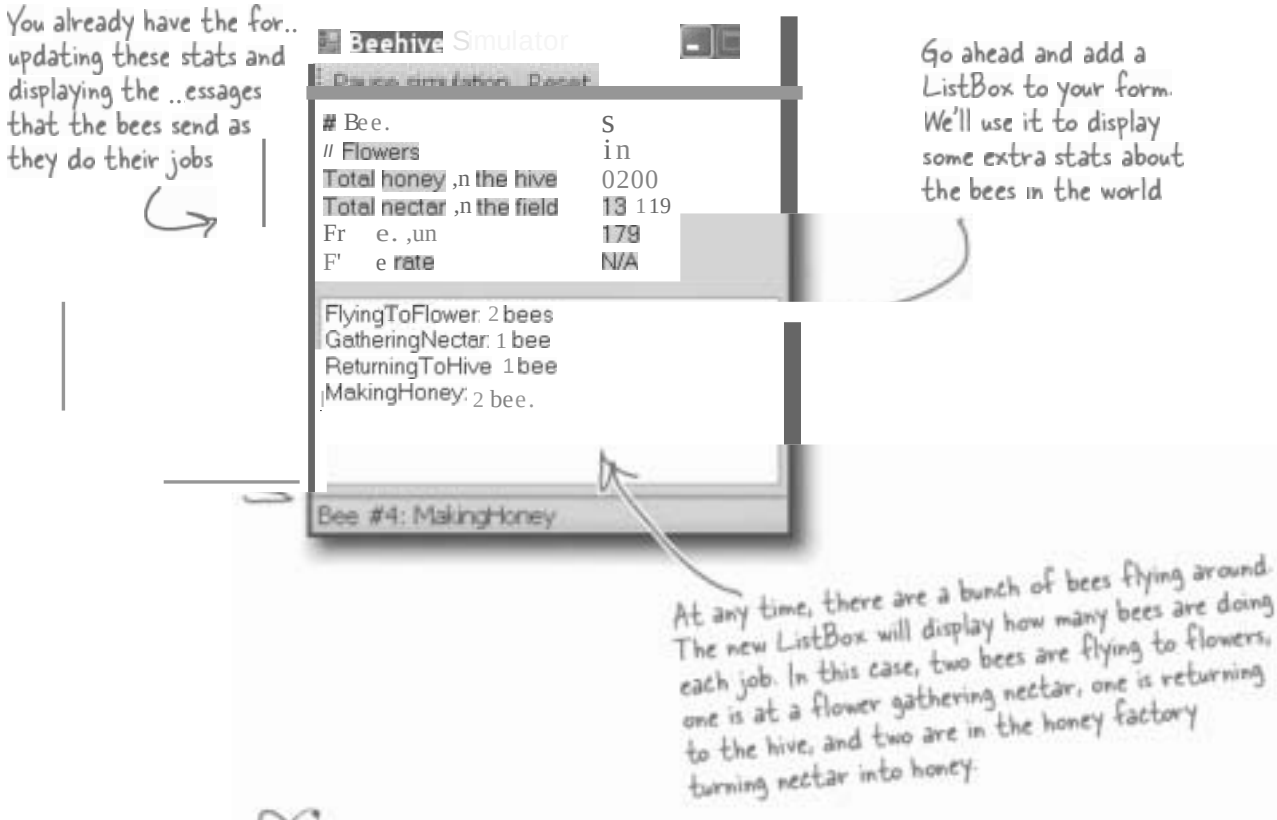
We create a new delegate from the Bee class (make sure you declared BeeMessage public), and point it at our SendMessage() method

Same thing here... create the world with the method for bees to call back.

This is the method we gave you... be sure to add it in, too

Let's **work** with **groups** of bees

Y""l' bee ,h"ulrllw buzzing around the hive and the field, and your simulation should Ik' running! How cool is that? But since we don't have the visual part Of the simulator working vet -- that's what we're doing ill the next chapter - all the inl;"'m." i",, we have s(far is the message that the bees are sending back to the main fl um with their callback. So let's add more information ab""t "hat the bees are d"im:.

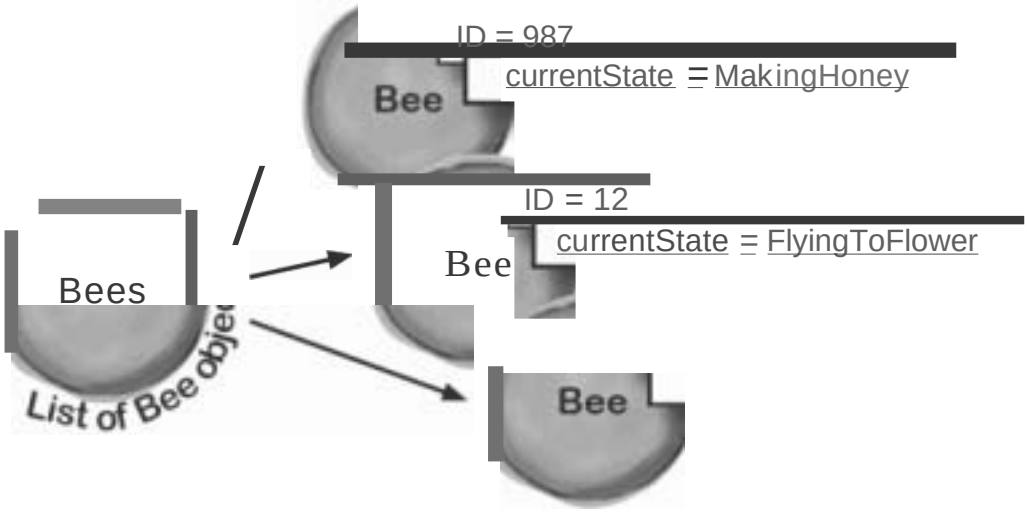


BRAIN POWER

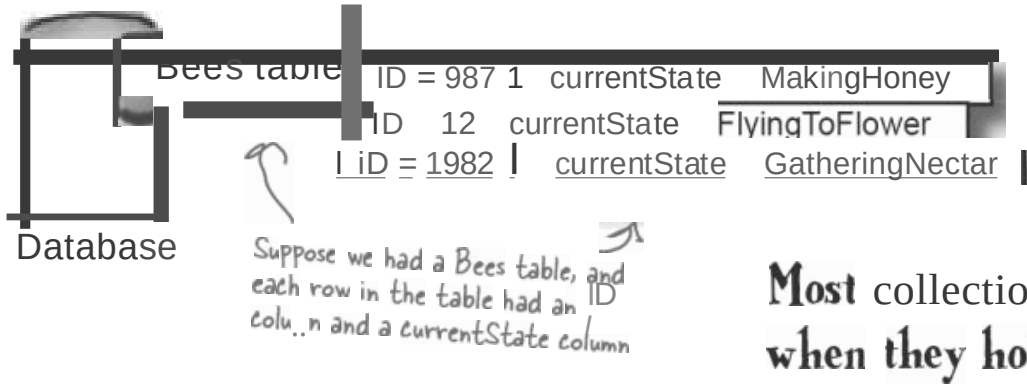
You know enough to gather the information you'd need to populate that ListBox-take a minute and think through how that would work. But it's a little more complex than it seems at first. What would you need to do to figure out how many bees are in each of the various Bee.St at e states?

A collection collects... DATA

(1,,1' bees are stored in a 1.ic, which is one of C#'s collection ty1"" · And cc,ll,:c,i"ll ty1"" re.dlv jus store data... a I", like a database d,,,·,. So each bee is likt· ;, row of data, complete with a stan', and II), and so "". Here's how ""1· bees loc,k as a """ll" , ,i"" or objects:



There's a lot "r data in the B · objects' fields Y'll can *almost* think "r a collec,i"n "f obj ts ,h,' same way you lhiu k "f rows i11" database. Each object holds data in its fields, the same wav each row i" a database)11olds data i" its t·„l"""" ,



Most collections-especially **when they hold** objects-can **be thought** of as data stores, just **like** a dalal,ase.

Who cares it you con think about a collection cs 0 dctcbcse if you can't use (I collection like (I database? Whot 0 totol waste of time..

O

C



What if you could query collections, databases, and even XML documents with the same basic syntax?

C# has a really useful feature called U NQ (whi, h stands I',r Langauge **INt**..grat..d Q.uery i. The idea behind LINQ is Ihal il give you a way to take an array, li,1. stack, queue, or other collection and work with all the data inside il all at once in a single operation.

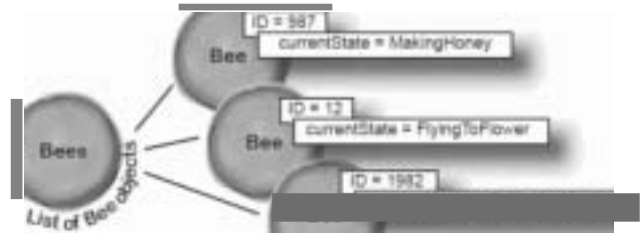
But what's ,eally great about LINQ is th.u you can use |"" same syntax Ihal works with collections as you rail for working with databases.

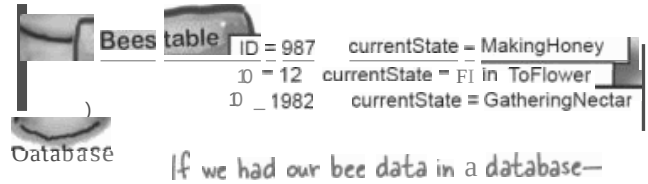
We'll spend most of Chapter 15 working with LINQ.

This LINQ query works essentially the same with data in a collection or a database.

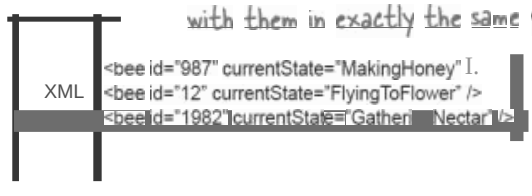
```
var beeGroups -
  from bee in world,Bees
  group bee by bee.CurrentState
  into beeGroup
  orderby beeGroup.Key
  select beeGroup;
```

LINQ





If we had our bee data in a database—or even an XML file—LINQ could work with them in exactly the same way



LINQ makes working with data in collections and databases easy

\Ve'n' going 10spe"d " ll "lliin' chapter Oil LINQ befi'r<' long, bur ",' can use LINQ and some R,'ady Bake e ,,,le 10add some extra li'" lllrd' 10our simulator. Ready B.•ke (:0.1" i, code you ,hould typ•. ill. and it's okay if vou don't understand it all. ,"" {l'll learn how it all works ill Chapter 15.



Ready Bake Code

```
private void SendMessage(int 10, string Message}
    statusStripl.Items[O).Text    "Bee #" + 10 + ". " + Message;
    val' beeGroups -
      from bee in world.Bees
      group bee by bee.CurrentState into beeGroup
      orderby beeGroup.Key
      select beeGroup;
    h'stBoxl.Items.Clear{);
    foreach (var group in beeGroups)
    {
      string s;
      if (group.Count() == 1)
          s    "".
      else
          s    "s";
      listBoxl.Items.l'ldd(group.Key.ToString() + ". "
                           + group.CountO + " bee" + 5);
      if (group.Key -- BeeState.Idle
       && group.Count() -- world.Bees.Count()
       && framesRun > 0)
          listBoxl.Items.l'ldd("Simulation ended: all bees are idle");
          tooIStripl.Items[O].Text - "Simulation ended";
          statusStripl.Items[O].Text - "Simulation ended";
          timerl.Enabled - false;
    }
```

e-: This is a LINQ query. It takes all the bees in the Bees collection, and groups them by their CurrentState property

The group's Key is the bee's CurrentState, so that's the order the states we displayed on the form

Make sure this matches the list box control's name on your form.

beeGroups is from the LINQ query We can count the members, and iterate over them

This bit of code makes sure it says, "1 bee" and "2 bees", keeping the plural right

Finally, add the group status (its key) and count to the list box.

Here's another nice feature Since we know how many bees are idle...

...we can see if ALL bees are idle. If they're out of the hive, so let's stop the simulation.



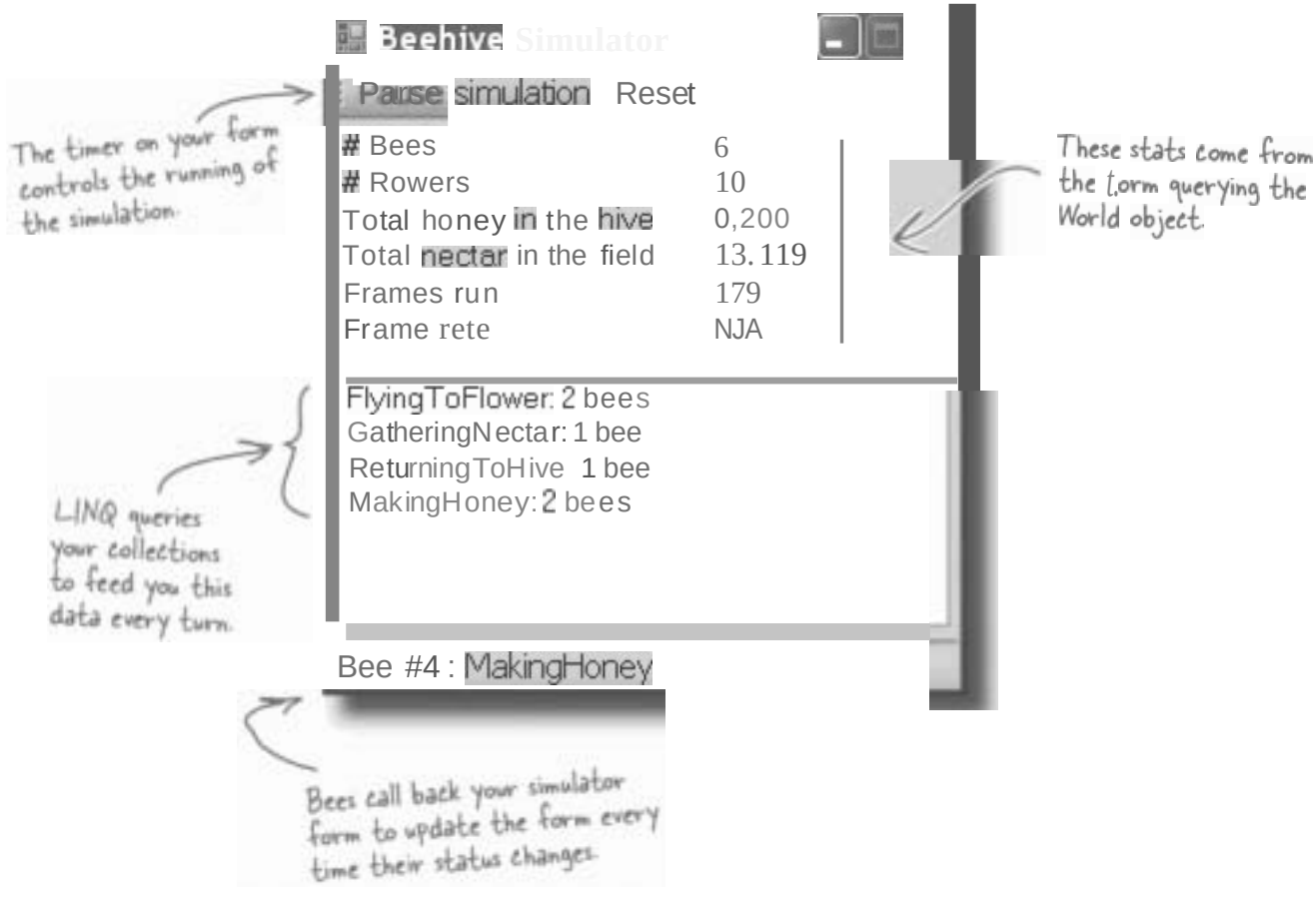
Relax

We'U learn a lot more about UNQ in **upcoming** chap'ere.

Y(,u do,,', need '0 memorizl' LINQ syntax '" try to drill all of this inl" your head right 'Hw. You'll get a l(,I nl!ore pra" lic" working with LINQ in (:hapt"r 15.

rest drive !Part 2)

(;" ahead and compile yo>III' code and run yomf project. II you get any errors, double-check your syntax, especially with the new LINQ code. Then, lire lip your simulator!



Olle filial challellge: Opell alld Save

We're almost ready I" take "11 graphics and add some visual eye candy I" our simulator. Fir' l. Ih,ough, let's d" one more Ihi11l: I" Ihi, version: all,ow loading, saving, "nd printing "r bec statistics.

you'll add the Print button now- we'll make it print a status page for the hive in the next chapte...

O Add the Open, Save. and Print icons

The ToolStrip control has a really useful feature—it can automatically insert picture buttons for standard icons: new. open, save, print, cut. copy, paste, and help. ,JU I right-click ,," Ih., ToolStrip icon the bOllO"' or lite Form Designer window and select "Insert Standard It"",; ', Then click 0" 11", first item that's the "new" ",'o" and delete il. Keep 11ll' next three items. Because they're the ones we need (" P" ll, save ;<l1d pnnn, ,\h"I' that comes a separator; you can ..irlu-r delete il" 1' move il between Ih,' Reset button and the save buton. Th"1\ delete the rest or the buttons.

O Add the button event handlers

The new standard hUll'ons ;,n' named openToolStril'BulIon. saveToc,ISlripBlln,,n and printToolStripiltutton. ,Jll' 1doul ,I,-click 0" them I" add their event handlers.



Add code to make the save and open cuncns work

1. Make the save button serialize the world to a lile. Stop the timer (you can restart II after saving). Set `MessageSender` 10 `null` for the hive and aililhe bees. so ,NET doesn't try and serialize the code your delegates point to: the form itself, Display a Save dialog box, and then serialize the `World` object, and the number 01 frames that have been run. Don't forget to reattachthe `sencneeaeue()` when you're done saving

2, Make the open button deserialize the world from a file. Take care of the timer Just like In the save button, pop up an Open dialog box, and oeseneuze the world and the number 01 frames run from me selectee file Then you can hook up the `HeeeaqeSender` delegates again and restart the timer (if necessary).

3. ncn't forget about exception handling! Make sure the world IS intact If there's a problem reading or writing the file, Consider poppmg up a human-readable error message mdicallng what went wrong



Your job was to make the Save and Open buttons work

Don't forget the extra using statements

```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
```

You'll need to make the World, Hive, Flower, and Bee classes serializable. When you serialize the world, .NET will find its references to Hive, Flower and Bee objects and serialize the... too

```
[Serializable]
public class World {
[Serializable]
public class Hive {
[Serializable]
public class Flower
[Serializable]
public class Bee
```

Here's the code for... the Save button

```
private void ••veToolStripButton_Click(object sender, EventArgs e)
    bool enabled - timer1.Enabled;
    if (enabledI
        ti_r1.Stopll:
    vcrre,Hive.MessageSender    null:
    foreach (Bee bee in world.Bees)
        bee.MessageSender .. null;
```

If we don't set all the MessageSender delegates to null, then when we try to serialize the world they'll act as a reference to Form1, and the BinaryFormatter will try to write out our form.

```
    SaveFileDialog saveDialog - new SaveFileDialog();
    saveDialog.Fi1ter - "Simulator File (•.bees)I•.bees";
    saveDialog.CheckPathExists .. true;
    saveDialoq.Title _ -caeeee a file to save the current sillUlatlon·:
    if (saveDialog.ShowDialog() .... DialogResult.OK) (
        try {
            BinaryFormatter bf .. new BinaryFormatter();
            using (Stream output .. File.OpenWrite(saveDialog.FileName)) {
                bf.Seri.alize(output, world);
                bf.seereiiee(output. framesRun);
        catch (Exception ex) (
            MessageBo".Show(·Unable to save the simulator file\r\n" • e".Message,
              "Bee Si.mulator Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

We decided to use ·bees" as the extension f... simulator save files

Here's where the world is written out to a file

Remember' when we serialize World, everything it references gets serialized... all the bees, flowers, and the hive

After we save the file, we need to hook up the Me..ageSender delegate• again, and then restart the t..... (if we stopped it).

```
    world.Hive.HessageSender - new Bee.BeeMessage(SendMessage):
    foreach (Bee bee in world.Bees)
        bee.MessageSender .. new Bee.BeeMessage(SendMessage);
    if (enabled)
        timerl.Start():
}
```

Here's the code for the Open button

```
private void openToolStripButton_Click(object sender, EventArgs e) {
    World currentWorld = world;
    int currentFramesRun = framesRun;

    bool enabled = timer1.Enabled;
    if (enabled)
        timer1.Stop();

    OpenFileDialog openDialog = new OpenFileDialog();
    openDialog.Filter = "Simulator File (*.bees)|*.bees";
    openDialog.CheckPathExists = true;
    openDialog.CheckFileExists = true;
    openDialog.Title = "Choose a file with a simulation to load";
    if (openDialog.ShowDialog() == DialogResult.OK) {
        try {
            BinaryFormatter bf = new BinaryFormatter();
            using (Stream output = file.OpenRead(openDialog.FileName)) {
                world = (World)bf.Deserialize(output);
                framesRun = (int)bf.Deserialize(output);
            }
        }
        catch (Exception ex) {
            MessageBox.Show("Unable to read the simulator file\r\n" + ex.Message,
            "Bee Simulator Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            world = currentWorld;
            framesRun = currentFramesRun;
        }

    world.Hive.MessageSender = new Bee.BeeMessage(SendMessage);
    foreach (Bee bee in world.Bees)
        bee.MessageSender = new Bee.BeeMessage(SendMessage);
    if (enabled)
        timer1.Start();
```

Before opening the file and reading from it, save a reference to the current world and framesRun. If there's a problem, you can revert to these and keep running

Set up the Open File dialog box and pop it up

using ensures the stream gets closed

Here we serialize the world and the number of frames run to the file

If the file operations throw an exception, we restore the current world and framesRun

Once everything is loaded, we hook back up the delegates and restart the timer.

You'll need to get your simulator up and running before you move on to the next chapter. You can download a **working** version from the **Head First Labs** website: www.headfirstlabs.com/books/hfcsharp/

Make it pretty



Sometimes you have to take graphics into your own hands.

We've spent a lot of lime on relying on controls 10handle everything visual in our applications Bul sometimes that's not enough-like when yeo wanllo anlmale a picture And once you get inlo emmauco. you'll! end up creating your own controls for your .NET programs, maybe adding a little double buffering , ane even drawing directly onto your forms , It all begins with the Graphics object, Bitmaps, and a detertrunafton to not accept the graphics status quo.

You've been using controls all along to interact with your progral\ls

Text.Boxes Prct.ure Soxes Labe l s... you've got a pretty good handle bv now un luow yc'u can use the controls in the 11)E's toolbox. But what do you ***really*** know about them? There's a IUI more to a control ,ll,,,, jus dragging ai icon '"\10 vour f(>'III.

O You can create **your own** controls

The controls ill the toolbox are really useful foi building forms and applications but the..:'s nothing magical about them. They'..: just classes. like the classes Ihal you've been " Tilim: "11 ;" "1' uWtl _In fact, C# make it reallv easy fc,r yc'11 1" create controls yourself. just by inheriting from the right base class.

O Your custom controls show up in **the** IDE's toolbox

Tl1ere's also nothing mysterious about the too[l»ox in the I]I I]:, I t just looks ir, your project's classes and the built-in .NET classe I," any controls. If it finds a control a -ass Ihal implements the right interface then il displays all icon for the class. If vou add vour own n"I"1II controls they'll show up in the 1,.,Il,,,x. too



You can create a class that inherits fro any or the existing co trol classes—even if it doesn't have any other code in it—and it'll automatically show up in the toolbox

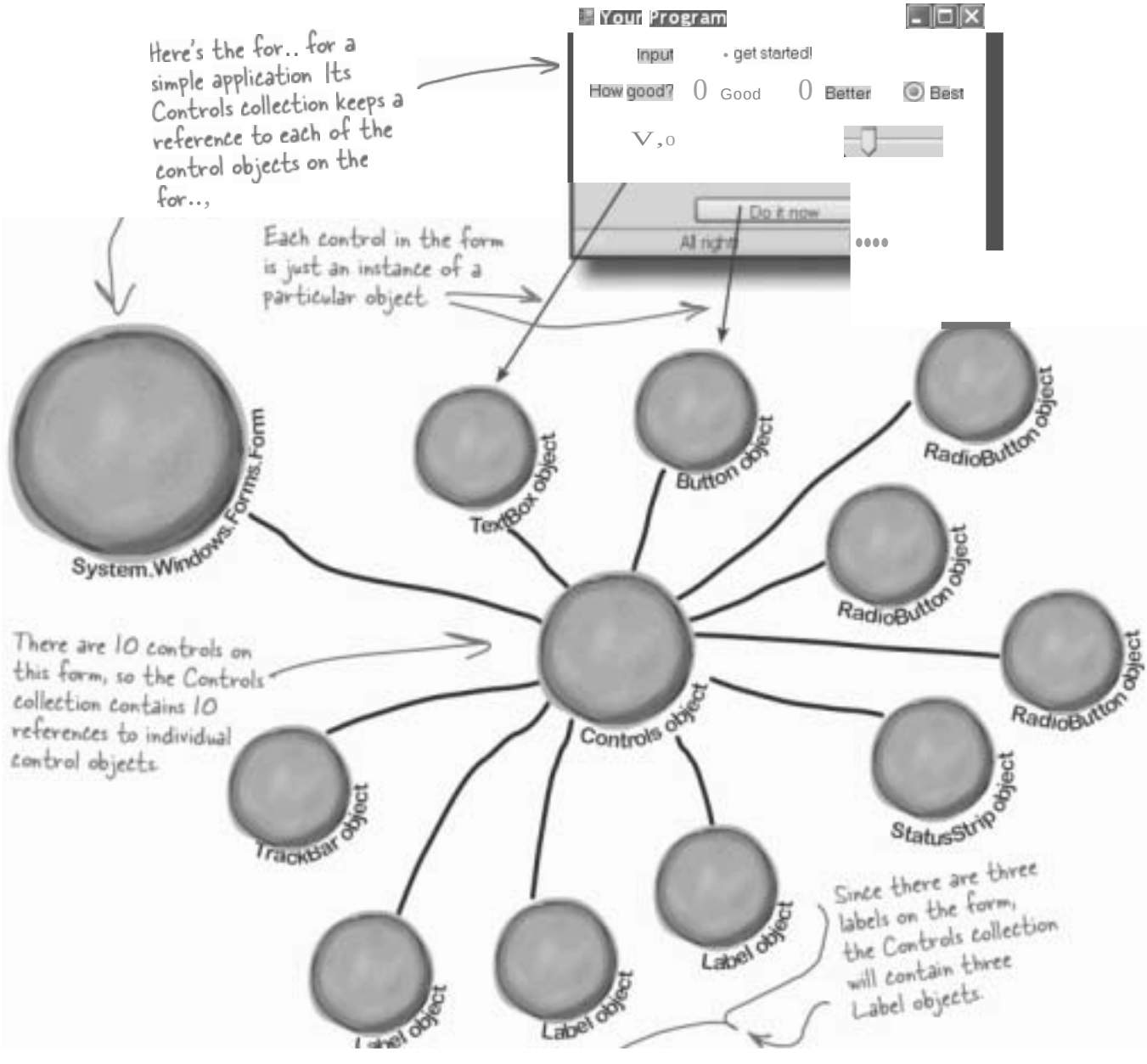
O You can write code to odd controls to your **form,** and even remove **controls,** while your **program's** running

.111' 1because yc,n lav "\II a form in the IDE's form designer. that doesn't mean thar il has to stay like that. Y" U\ "' already IIH",ed plenty Of `PictureBox` controls around (like wi",,, you built Ih,' greyhound race). B,u you can also add " r remove controls, too. In fact, when you build a form ill the III E, all it's doing j, writing the code tha' adds the controls lu the [orrn.., which means you can write similar code, and run that code whenever you «unr.

Forl\1 eontrols are Just objects

You already know how important c on lrols are to your forms. You've been using buttons, text boxes, picture boxes, check I",x,·,. group boxes, labels, and other fc'nn, ,ince· chapter I. \Vell, il Iurns out that Ihose controls are just objects, just like everything else you've been working with.

.\ control is ju,t an object, like any other object it just happens 10 know h,,,, to draw itself. The Form object form keeps track of its controls using a special collection called Controls . which you can use to add or remove controls ill "oUr 0\11l code.

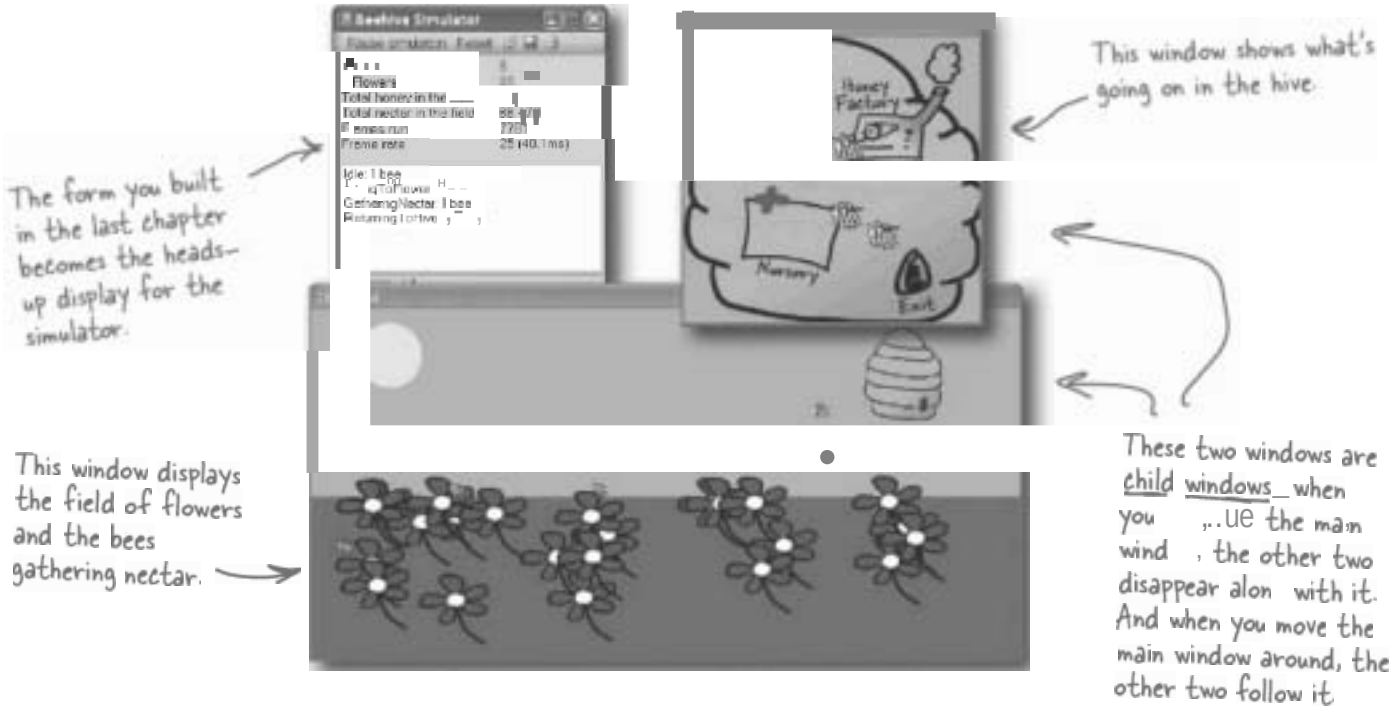


Use controls to al1ill\ate **the** beehive sill\ulator

You've built a great simulator. Inn it's nol nunh 1" look at. It's time to create a really stunning visualization that shows those h..es in action. You're about to I,uild a r..nderer that animate the beehive...and controls are Ilie key.

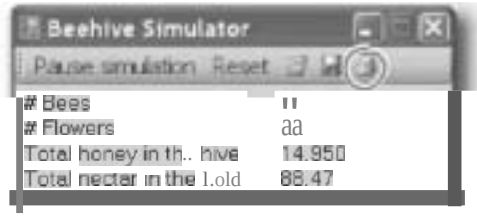
O The user interface shaws yau everything that's going on

Your simulator will have three different windows. You've already buill Ihe main "heads-up display" stats window that shows stats about lh,' current simulation and updates from th.. bees. Now you'll add a window that show you what's going ill inside the hive, and a window that shows lhe li,'ld of flowers wlu-re the bees gather nectar.

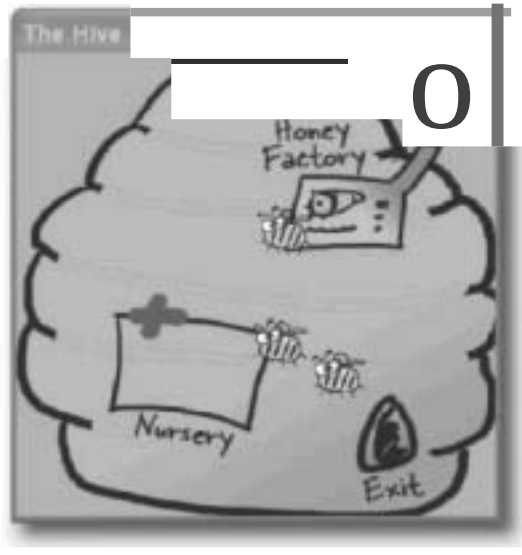


O We'll add a Print button to the stats window

T Ihe stats window has an oper ,,,,,1save I"m,,,, already. Once we add th,. graphic", we can add a bUll""l" the toolbar l" pIin! an info page about what's going "n.



The **hive** window shows **you** what's **going** on inside the hive
As lh.. bees fly around the world, you'll „-,-<1 to ''''imate ,-adl one. S<" „-ti„„-, they're inside rh,' hive, and when they are, they show lip in thi, window.



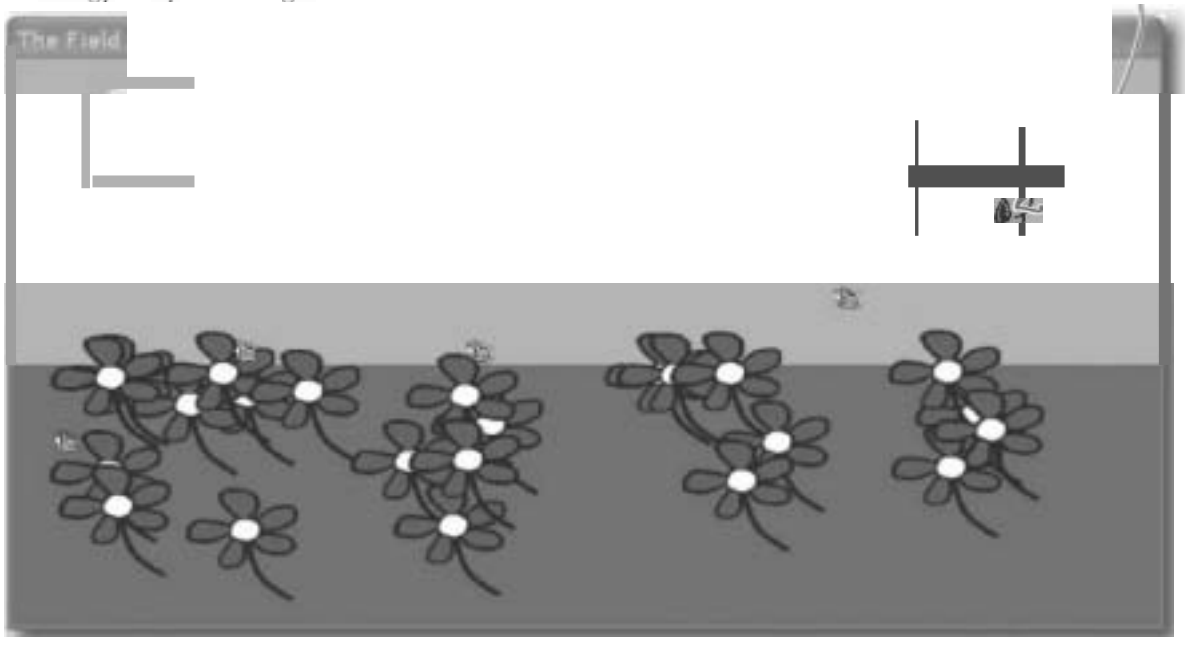
The hive has three important locations in it. The bees are born in the nursery, they have to fly to the exit to leave the hive to gather necta.. fr__ the flowers, a"t! when they come back they need to go to the honey factory to make honey.

The hive exit " on the hive for , and the entrance i. on the field form (That's why we put both of the.. in the Hive's locations dictionary.)

Here's the entrance to the hive. When bees fly into it, they disappe from the field for nd reappear near the Exit in the hive for..

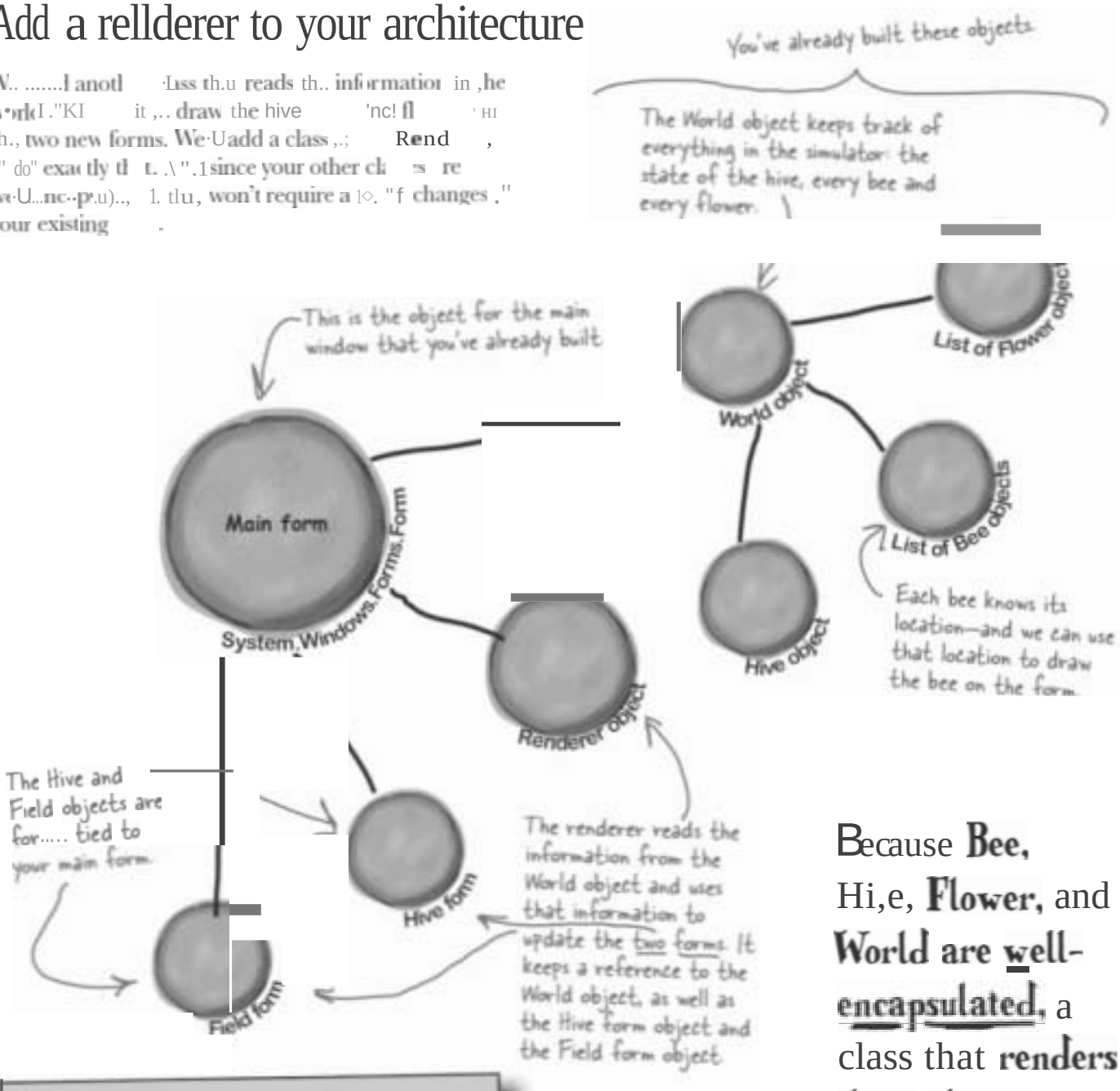
O

The field **window is** where the bees collect the **nectar**
Bees have H"" big job: collect nectar liu lll ,I,,·flowers, 'llld bring it back to Ill,' h;,-,>I" ",;ok,> lu",ey. T il,'" they eat I"oney to give them energy to fly out and get more nectar.



Add a rellderer to your architecture

W.. .....l anotl ·Lss th.u reads th.. inforrnatiou in ,he w•rkl."KI it ,.. draw the hive 'nc! fl 'HI ,h., two new forms. We·Uadd a class ,.; Rend , '" do" exac tly tl t. .\".1since your other cl s re ww·U...nc··p.u).., 1. tlu, won't require a 1◇. "f changes ." your existing .

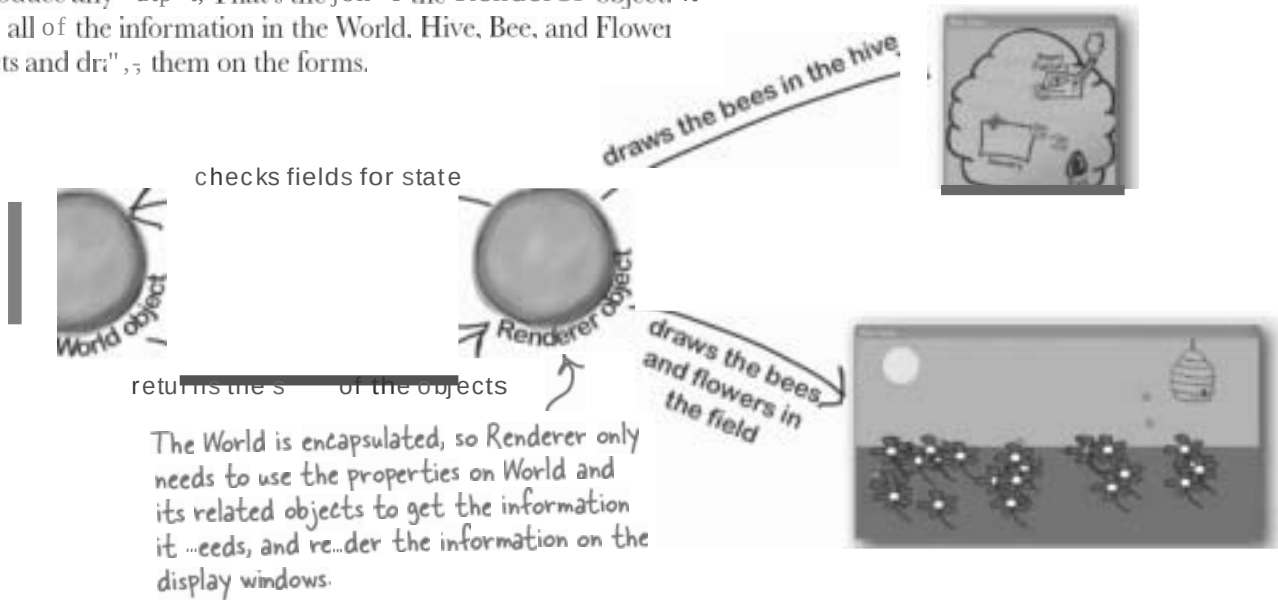


ren-der, verb.
to n-pn-sr-nt or depict artistically.
Sally's nrt tmcher asked the class til look at nll of the shadows and lines ill the model and **render** *them 01/ the page.*

Because **Bee,** Hi,e, **Flower,** and **World are well-encapsulated,** a class that **renders those objects** can I.. **added without** lots 01 **changes** to existing coJe.

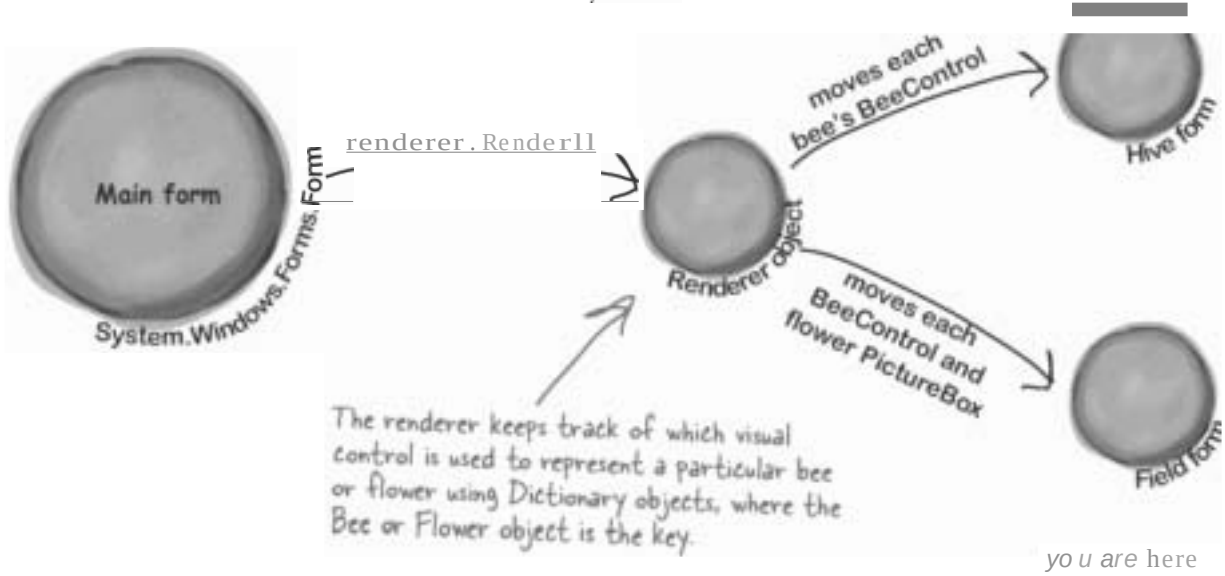
Ihe renderer draws everythillg in the World on the two forms

'11", War1d obj..ct keeps track ,,l' everything ill the simulation: th,' hive, the bees, and the flowers. But it d,)esn't actually draw anyl1hing or produce any "utp"t, That's the joh"f the Renderer object. It reads all of the information in the World. Hive, Bee, and Flowe1 objects and dr:", ; them on the forms.



The shMulator re.ders the world after each frame

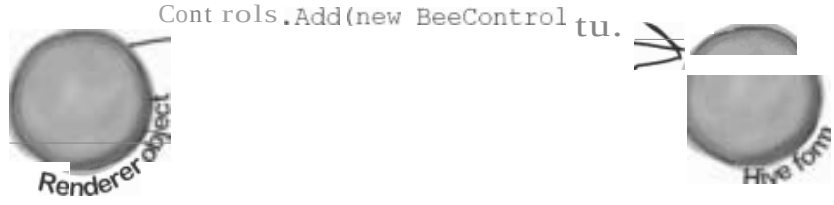
After the main form calls the world's Go () method, il ,h"\I)d call the renderer's Render () method to redraw the display windows. For example, each flower will be ,Ii'l)layed using a PictureBox control. But let's go f"rlh,',-with lk"" ""d create an animated control. You'll create this new control, called BeeControl, and define its behavior yourself.



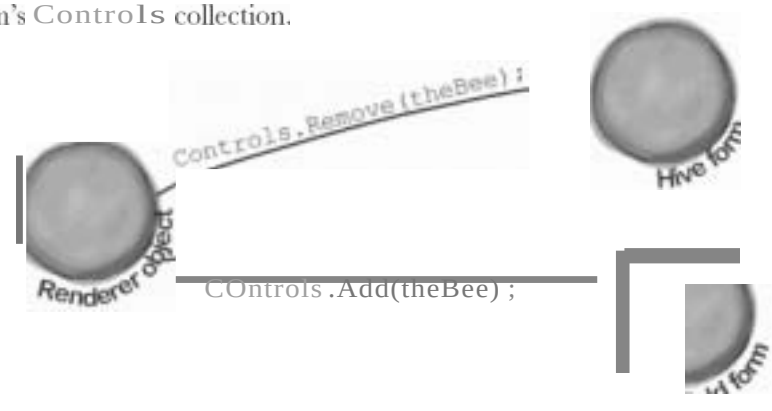
COl1trols are well-suited for visual display elelllellts

When a new bee i, added to the hive. we'll want our simulator to add a new BeeCont rol 10the Hive form and change its I,cation as il move around the world. When that bee flies out of th.. hive, o" r simulator will " ",'d 10remove the control from the Hive form and add it to the Field !(>rrn, ,\ nd when il flies back to the hive with it, load of n",'tar, it, control needs to be removed from the Field fortu and added back 10the Hive form. ,\ ud all the while, we'll want the animated bee picture t" flap its wings.., and controls will make it easy to do all or that.

o The world adds a new h,',·, and the n'nd"r"f create a new Bee Control ami adds il to the Hive form's Controls collec,tinn,



o When the bee lli,', urn or the hive and enters the field, the renderer n-move the BeeControl from the hive's Controls collection and adds il to the field form's Controls collection.



o ,\bee «m retire if it's idle and it's gotten too old, If the renderer checks the world's Bees list and finds Iha l the h.·" i, no longer rln-re. it removes the control from the Hive form



Sharpen your pencil

Can you figure out what each of thes.. code snippets does? Assume each snippet is in'ide a form, and write down your best guess.

```
this.Controls.Add(new Button());
```

```
Form:? childWindow • new Form2();
childWindow.Backgroundlmage •
        Properties.Resources.Mbsaic;
childWindow.BackgroundImageLayout
        ImageLayout.Tile;
childWindow.Show();
```

If you've got a ListBox on you.. for.., 'r- can use its AddRange() method to add list items

```
Label myLabel - new Label();
myLabel.Text _ "What animal do you like?";
myLabel.Location _ new Point 110, 10);
ListBox myLlst = new ListBox(),
myList.Items,AddRange( new object I]
      ( "Cat", "Dog", "Fish", "None" } );
myList.Location = new Point(lO, 40),
Controls.Add(myLabel),
Controls.Add(myList);
```

You don't need to write down each line, a uch as summarize what's going in the code block.

```
Label controlToRemove = null;
foreach (Control control in Controls)
    if (control is Label
        && ccneecrvrext == "Bobby")
        controlI'okernove = control as Label;

Controls.Remove(controIToRemove);
controlToRemove.Dispose();
```

Bonus question: Why do you think we didn't put the Controls.RemoveO statement inside the foreach loop?

Try it out if you want, and write why you think you got the result that .NET gave you.



Sharpen your pencil Solution

Can you figure oul what each of these code snippets does? Assume each snippet is inside a form. and write down what you thin k il does.

```
this.Controls.Add(new Button(» ;
```

Create a new button and add it to the form .It'll have default values (e.g.•the.Text property.will.be.empty),....

```
Form2 childWindm; = new Form2();
childWindow.BackgroundImage =
        Properties.Resources.Mosaic;
chiIdWindow•BackgroundImageLayout
        ImageLayout.Tile;
childWindow.Show0 ;
```

There's.a second.Form.in.the.application called.Form2,.so.this.creates it,.sets its background.image. to.a.resource.image.called "Mosaic",.makes.the.background.image.so.it's.. tiled.instead.of.stretched,.and.then.displays. the indow.to.the.user.

```
Label myLabel = new Label();
myLabel.Text = "What animal do you like?N;
myLabel.Location = new Point(10, 10);
ListBox myList - new ListBox();
myList.Items.AddRange( new Object II
      { "Cat", -occ-. -rrsn-. "None" } );
myList.Location . new Point(10, 40);
Controls.Add(myLabel);
Controls.Add(myListl;
```

This code creates a new label, sets its text, and moves it to a new position. Then it creates a new listbox, adds four items to the list, and moves it just underneath the label. It adds the label and listbox to the form, so they both get displayed immediately.

```
Label controlToRemove • nUll;
roreecn (Control control in Controls)
    it (control is Label
        && control.Text == "Bobby")
      controlToRemove = control as Label;

Controls.Remove(controlToRemove);
controlToRemove.Dispose();
```

This loop searches through all the controls on the form until it finds a label with the text "Bobby". Once it finds the label, it removes it from the form.

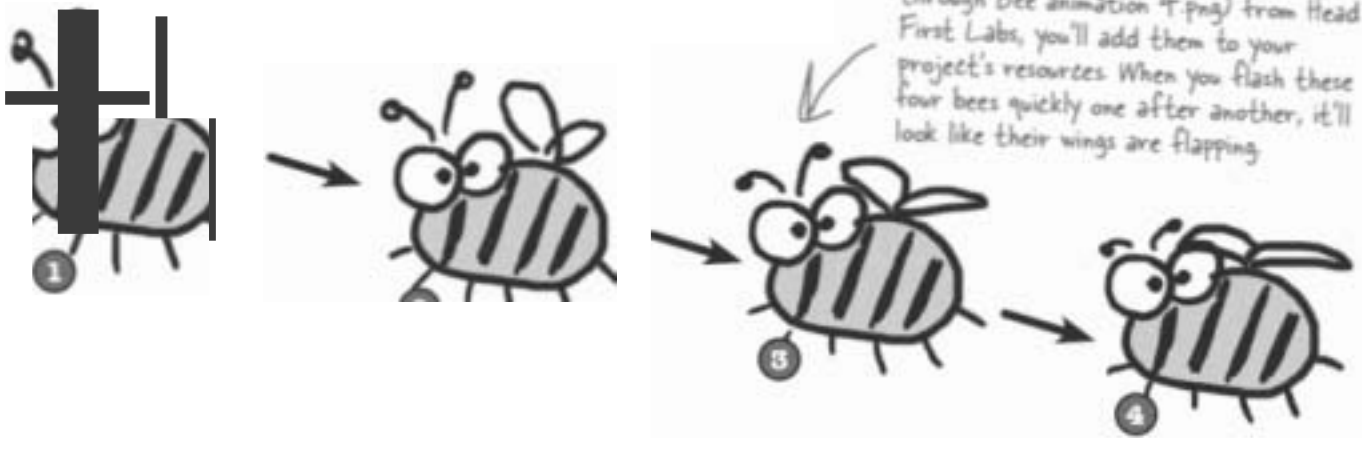
If you try, .NET will throw an exception. It needs the collection intact, otherwise it'll lose its place and give you unpredictable results. That's why you'd use a for loop for this ,,stead

Bonus question: Why do you think we didn't put the Controls.RemoveO statement Inside the foreach loop?

You can't modify the Controls collection (or any other colllection) in the middle of a foreach loop that's iterating through it.

Build your first al1il\1ated eoatrol

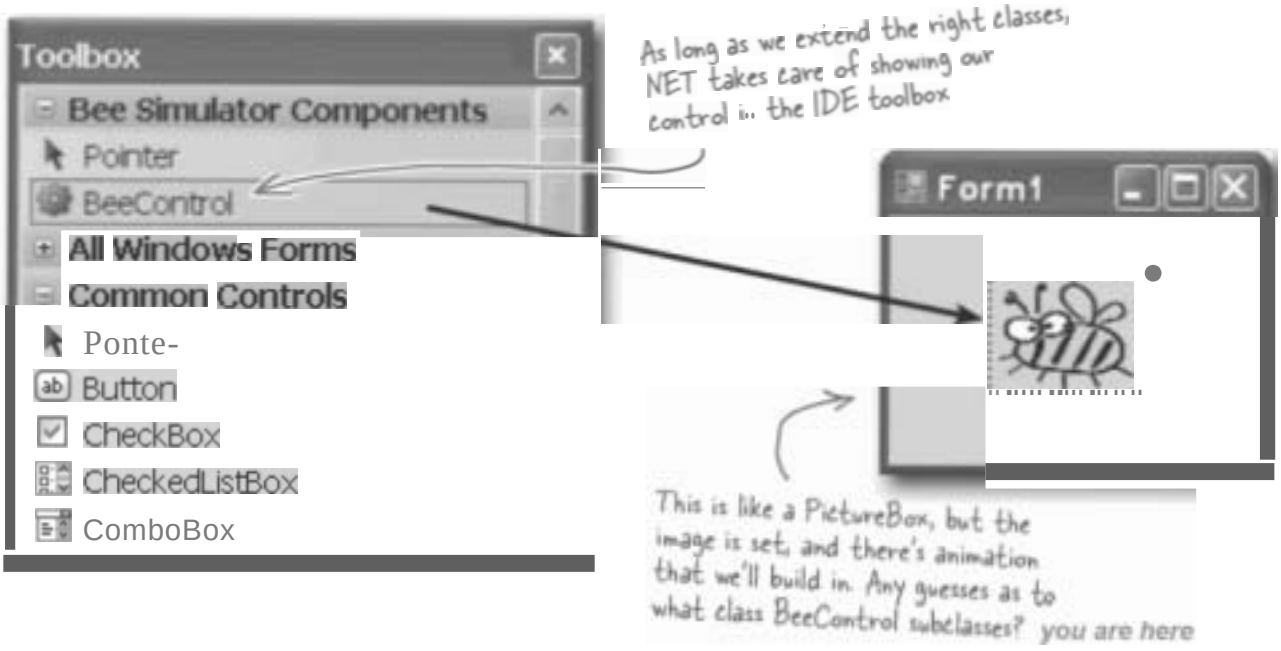
You're going ,,, build **your** 0**wn** r o n l r o J that draws an animated bee picture. If you've never d..„<," animation, it's "ot as hard as i' sounds: you drav "..n of pictur ' "fIr. another, and prodh Ih.· illusion of """ 11..mt. I k, for us. li ay C# .",d .NET handle rces makes it reallv easv for I" do anim.tion.



We waot a **control** io **the** toolbox

I f you build BeeControl right, ;1·1l apJ..ar as a com..l that " .l can drag ,,,n " f yc,ur rt"dbox and onto your fi" 'III, h'Dlook just like .. Pfcturegcx showing a picture of .. l..•••• except that j(U have animated flapping wings.

Download the images lor this chapter from the Hea(First Labs website: www.headIirstlabs.com/Isooks/hfcsharp/



BeeControl is LIKE a PictureBox... so let's start by INHERITING frol\l PictureBox

Since every contr,l in the Too,ll>c'x is just an "hj,:ct. it's easy 10 make a new control. All you need 10 do i, add a new class 10 your proje t that inherits from an existing control, and add any new behavior you want your ntrol 10 perform.

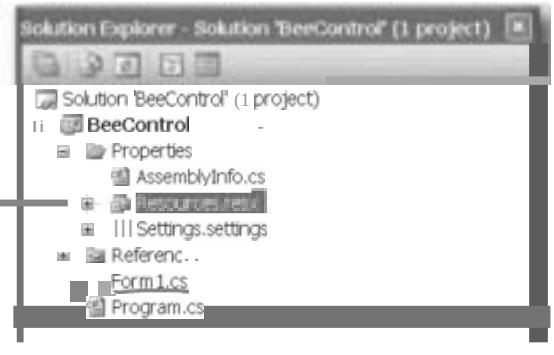
We want a control let's call il a **B....Control** that shows an animated picture: of a bee flapping its wings but " '€'11 ,1;1l'1 with a control that shows a *non*-animated picture, and then just add an im",iol1. So we'll start with a PictureBox, and the11 we'II add code to> draw an animated bee '"' it.



o Cr..a... a new project and add the f'ur animatiOll ce·ll, to the project's resources, jn'l like you added the (1hj"'l,-ill,' Paper Company logo to your project way back in Chapter 1_But instead of adding the-m 10 the *form* resources, add the-m 10 the *project's* resources.

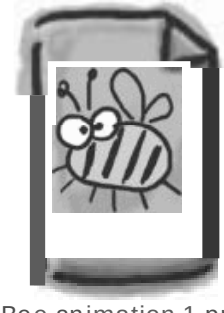
I" Chapter I, we added the logo graphic to the form's Resources file This ti..e we're adding the resources to the project's global collection of resources, which makes the.. available to every class in the project (through the Properties Resources collection).

Take a minute and flip back to Chapter J to re..ind yourself how you did this.



Tsese appear under your project, not a particular for..

o We've drawn a four-ce1I1"·e· animation that you ran import iuto your resource that you can download from htlp:1 l www.he**adfirstlabs.com/books/hfcshar**p/ . Then, go 10 the Resource page, select "Images" from the lir' t dropdown at tl", top or the screen, and select ':\<ld Existing File..." from II,,' "Add Resource" dropdown.


Bee animation 1.png


Bee animation 2.png

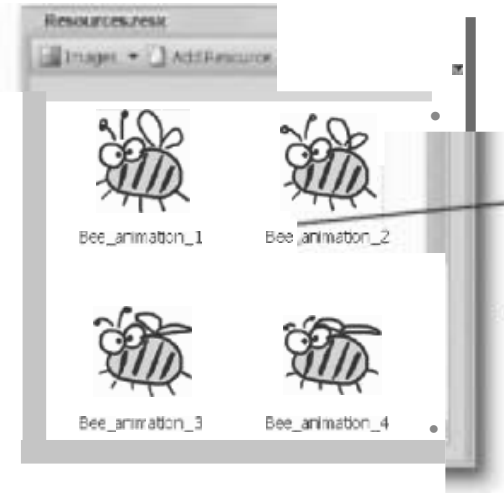

Bee animation 3.png


Bee animation 4.png

I..port each of these i..ages into your project's resources

e When you add images or other resources 1" the project's Resource file, you can a..:cess them using the Properties,Resources class..IU 1go 10anv line in your code and typ(Properties,Resoure as soon as you do. the IntelliSense pops up a dropdown list that ,I..ws all "r the picture yo orted.

Note that "." at the end.. that's what tells the IDE to pop up the properties and ..ethods of the class r- typed in




```
pictureBox1.Image =
    Properties.Resources.Bee_animation_1;
```

This sets the image used for a particular PictureBox's image (and for our starting image).

These images are stored as public properties of the Properties.Resources class.

You'll need to add a "using System.Windows.Forms" line for the PictureBox and Timer.

4 **Now add your BeeControl!** Just add this BeeControl class to your project:

```
pub11c class BeeControl . PictureBox (

  private Timer animationTimer = new Timer();
  public BeeControl()
    animationTimer,Tick +- new EventHandler(animationTimer_T1ck);
    animationTimer,Interval - 150;
    animationTimer,Start();
    BackColor • System.Drawing.Color.Transparenr;
    BackqroundImaqebaycut • ImageLayout.stretcr»
```

Here's where you initialize the timer by instantiating it, setting its Interval property, and then adding its tick event handle-.

```
  private int cell . 0;
  void animationTimer_Tick(object sender. EventArgs e) {
    cell++;
    switch (cell)
      case I: BackgroundImage   Properties.Resources.Bee_animation 1; break;
      case 2: BackgroundImage   Properties.Resources.Bee_animation 2; break;
      case 3: BackgroundImage   Properties.Resources.Bee_animation 3; break;
      case 4: BackgroundImage   Properties.Resources.Bee_animation_4; break;
      case 5: BackgroundImage   Properties,Resources.Bee_animation_3; break;
      default: BackgroundImage - Properties.Resources.Bee_animation 2.
      cell - 0; break;
```

Once we get back to frame #1, well reset cell back to D

Each ti..e the timer's tick event fi.es, it incre...ents cell, and then does a switch based on it to assign the .ight pictu.e to the Image property (inhe-ited from Pictu..eBox).

When you change the eede for a control, you need to rebuild your program to make your changes show up in the designer

Then **rebuild your program**. Go back to the- form designer and look in the- toolbox, the BeeControl i, (here. Drag il "nlo your form - you get an **animated** bee!

Create a buttoll to add **the BeeControl** to your **form**

It's easy t" add a .."ntr,,1 10 a form —just add il 10 the Controls ..ollection. And it's just as easy 10 remove from Ih"«Jft11 b\' removing il (""'III Controls. But controls implement IDisposable, so make sure you **always dispose your control** after you n'm,,,,' il.

Now do this

1 Remove the aeeccntrel from your form, and then add a button

Go 10 the form designer and d..lete Ih.. 8eeControl [rom th.. [orlll Then add a button. We'll make Ih,' bUII"n add and remove a BeeControl.

2 Add a button to and and remove the bee control

H,-re's the event handler for il:

When you add a control to the Controls collection, it appears on the form immediately.

You ca.. use an object initializer to set the BeeControl properties after it's i..stantiated

```
BeeControl control - null;
private void buttonl_Click(object sender, EventArgs e)
    if (control == null) {
        control . new BeeControlO { Location = new Point(lOO, 100) );
        Controls.Add(control);
    } else {
        Controls.Remove(control):
        control.Dispose() :
        control · null:
```

All you ..eed to do to remove the c trol is remove it from the C trols collection But make ...re you dispose it—otherwise it'll keep taking up resources!

Now when you nil' your program, if you click Ih,' Imlt"" ont·; it'll add a new BeeControl to the form. Cli,'k il again and il'll delete it. It uses the private `control` li,'ld I" hold the reference to the control. (It sets the reference to ""II when there's n« control 011 the j',rm.)



You can add your own control to the toolbox just by c..eating a e!ass that i..herits from Control.

Behind the Scenes

Every visual control in your toolbox inherits from `System.Windows.Forms.Control`. That class implements the has some members which should be pretty familiar by now: Visible, Width, Height, Text, location, BackColor, BackgroundImage... all of those familiar properties you see in the form's Properties window.

Your controls need to dispose their controls, too!

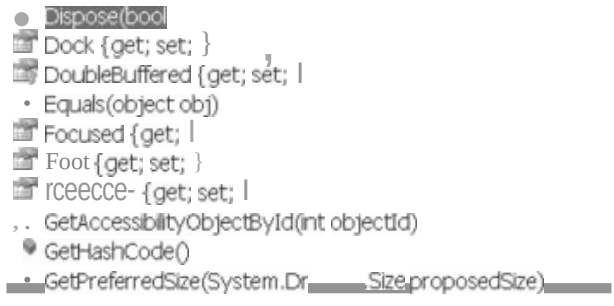
There's a problem with the BeeControl. Controls need to be disposed after they're done. Hut the BeeControl creates a new instance of Timer, which is a control that shows up in the toolbox... and it never gets disposed! That's a problem. Luckily, it's easy to lix- j",(override the Dispose() method.

The control class implements IDisposable, so you need to make sure every control you use gets disposed

3 Override the Dispose() method and dispose of the timer

Since BeeCIlll",Inherits frIlll a control, then rh.lt cmtrol Illust have a Dispose() methc,d. S" we can j"'1 override and extend that method to dispose otll' timer. Just go into the c('tl1r,,] and type `override`:

```
public class BeeControl : PlctureBox {
    override
```



When you type "override" inside a class, the IDE pops up an IntelliSense window with all of the methods you can override. Select the Dispose() method and it'll fill one in for you!

As soon as you click on Dispose(), the IDE willlill in th" method with a ('"11110 base.Dispose():

```
protected override void Dispose(bool disposing) {
    base.Dispose(disposing);
```

e Add the code to dispose the timer

Since BeeControl inherits from a control, then that control must have a Dispose() method. So we can just override and extend that method 10 dispose ,our timer. Just add a line 1" the 1Dispose() methc,d Ihal the J1)1-; I,II,-d ill fi,r y"":

```
protected override void Dispose(bool disposing) {
    animationTimer.Dispose();
    base.Dispose(disposing);
```

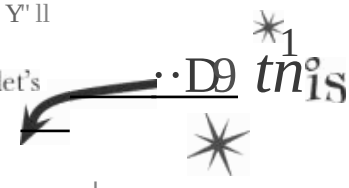
Now r-:. BeeControl will dispose of the ti...er that it c..eated in its constructor. The IDE filled in a call to the Pictu..eBox base class's Dispose() ...ethod—leave it in so the PictureBox can dispose itself, too.

Now the BeeControl will dispose or its li",er as part of its own Dispose() llH'lh"d. It cleans HI' after itself!

Any conlrol that you write **from** scralch **is** responsible **for disposing** any olher controls (or **disposable** ol,jeels) that il **creates.**

A UserColltrol is an easy way to build a eontrol

There's an easier way It> build your "wn Toolbox controls. Instead of creating a class that inherits from an existing control, all you need I,.do is **use** the **IDE** to add a **UserCon**Irol **to your** projc**ct**. You work wilh a UserControl jU'1 lik,' a form. Y" ll ,'an drag otl,,_r controls " Ul of the I'olbo'x and 0"1') ;1 – il uses the Illormal fo'nil designer ;n ll,,-l11E. .\"d you can use its events just like you do with a form. So let's rebuild the BeeControl using it U..« 'ontrol.



o Right-click oil BeeControl.cs in the [HE and rename it I> ()ldBeeContro,l.,,. TI,,-IJ)E will pop up a **Y**.."/ No window: *"You are renaming a file. Would you also like to perform a rename in this project of "II references to the code element 'BeeControl'?"* thi, window is asking if you want to rename the BeeControl class 10(1idBeeControl, and Ihen change all of the ('od,- ill the !'roj'c1I" reflect Ihi, change. **Click the "Yes"** button to rename yo'lir BeeControl to ()ldB:eContro,].

o Right-click ' "l the project ill Ihe Solution Explorer alltl select "Add > User Control..." Have the 11)1' add a **us**..r **control call**..d B....**Control**. The IDE will open up the new control inlh,' ti)rlll designer.

Use the ani..ationTi..e-_T,dO ..ethod (you'll need to rename ,V and the eell field from the ..ld bee t:ontr.l—but don't copy anything else

o Drag a Timer ,,,ntrol " "1o yo,ur user co)llln)l. It'll show IIJl at the bo,n, llll ,)1' the designer, jU'1 lik,' with a form. Use the Prop,-n i,-, window 10set its Interval to 150 and its Enabled to true. l' lu-n double-click on il the IDE will add it, Tick event handler. Just use the same Tick event handler that you used earlier 10animate the first bee C0l11ml.

o Now update the BeeControl's constructor:

```
public BeeControlll
    InitializeComponent (J;
    BackColor = System.Drawing.Color.Transparent;
    BackgroundImageLayout = ImageLayout.Stretch;
```

You can also do this from the Properties page in the IDE, instead of using code

o (;0 back to the button event handler ,," the form. \When you renamed the first BeeContro,l to ()ldBeeControl, il changed the fi-rm ,',×l,- as well. So change the two lin,-, back 10BeeControl so it uses your new UserControl instead of the PictureBox:

```
BeeControi control = null;
private void buttonl Click(...)
    if (control -- null} {
        control - new Beecontrol ll { ,.
```

Now **run** your prograllI- il ,1',,,ld work exactly the same as before. The h"llon now adds and removes your IJserControl-based BeeC:011rol.

A Usel'Conll'o! Is an **easy way** 10 **add** a control 10 **the** toolbox, **Edit** a Usel'Con I I'ol jusl **like** a [orm-eyeu can **drag** otb.1' controls oul of lb. **toolbox** onto it, anJ **you** can use its events exactly **like** a **form**'s even**ts.**

But I've been using controls all this time, and I've never disposed 0 single one of them! Why should I start now?



You didn't **dispose** your controls because your **forms** did it for you.

But don't take our word for it. Use the IDE's search function to search your project for the word "Dispose", and you'll find that the IDE added a method in Form1.Designer.cs to override the Dispose() method that calls its own `base.Dispose()`. When the form is disposed, it **automatically disposes everything** in **its** Controls **collection** so you don't have to worry about it. But once you start removing controls from that collection or creating and instance of controls (like the Timer in the BeeControl) outside of the Controls collection, then you need to do the disposal yourself.

there are no Dumb Questions

Q: **Why does** the form code for the **PictureBox-based** BeeControl work **exactly** the **same** with the UnrControl. **based** BeeControl?

A: Because the code doesn't care how the BeeControl object is implemented. It just cares that it can add the object to the form's Controls method.

Q: I **double-clicked** on **my** OldBeeControl **class** in **the** Solution **Explorer,** and it had a **message** about adding **components** to **my class. What's** that about?

A: When **you create** a control **by adding** a class to **your** project that **inherits** from PictureBox or another control, the IDE does some clever things. One of the things it does is set you up to work with **components** like those non-standard controls like Timer and OpenFileDialog that show up in the space beneath your form when you work with them.

Give it a try—create an empty class that inherits from PictureBox. Then rebuild your project and double-click on it in the IDE. You'll get this message:

TO ***add components to*** your ***class, drag them*** *from* ***the Toolbox and use the Properties*** window to ***set their properties.***

Drag an OpenFileDialog out of the Toolbox and onto your new class. It'll appear as an icon. You can click on it and set its properties. Set a few of them. Now go back to the code for your class. Check out the constructor—the IDE added code to instantiate the OpenFileDialog object and set its properties.

Q: **When I changed the properties** in **the OpenFileDialog, I noticed** an **error message** in **the** IDE: **"You must rebuild** your **project** for **the changes** to **show** up in any open **designers."** Why did I **get** this **error?**

A: Because the designer runs your control, and until you rebuild your code it's not running the latest version of the control.

Remember how the wings of the bee were flapping when you first created your BeeControl, even when you dragged it out of the toolbox and into the designer? You weren't running your program yet, but the code that you wrote was being executed. The timer was firing its Tick event, and your event handler was changing the picture. The only way the IDE can make that happen is if the code were actually compiled and running in memory. So it's reminding you to update your code so it can display your controls properly.

Ihe renderer uses your **BeeControl** to draw **animated** bees **on** your **forms**

\Vith a BeeControl class and two fe'-m,. youj"'l need a way 1,. positi,.u Ibe,·,. move them from one form 10the other, and k,"'l' up with the he,·,_ Y' n'll ;,1", ueec110po,ili"n flowers ,,1\ the FieldForm. although since tl,wers don'l move, that's pretty ,i"'pl,'. .\ll "l' this is code Ilat we can pill inlo a new class, Renderer. Here's what that class does:

"1"....ll want the hive and field forms "linked" to the stats form—that does useful things like minimizing the hive and field forms when you ..ini.. ee the stats form You can do this by telling Windows that the stats form is their owner.

0 **The** stcts form **will** be **the parent** of **the hive** and field **forms**

The flrsr step in adding graphics 10the Beehive Sil11l1al"r will Be· adding two !"Im, In the project. You'll add one called HiveForm (to show the inside "l' Ih(hive) and c"" •alled 1" ldFor'" (w1", h ",11 show the Ii, ld "r n,we rs) I l" 11 vou'll add Ii', ·, to Ihe main fi 'Im', '" nstructor to ,I.,w ,I, two child form to te ll Windows ,h.ll the stats form i, Ih,-if owner:

```
public form! ()
    II other code in the Formi constructor
    hiveForm.Show(thi::) ;
    fieldForm.Show(this) ;
```

Every for.. object has a Show() ethod If r- want to set another f as its owner, just pass a reference to that form to Show().

2 **The renderer** keeps **a reference** to **the** world **and each child form**

.\1 the very top of Ihe Renderer class you'll need a few imp"rl,mt fields. T Ihe class Ilas 10know Ihe location "r each bee and flower, so it needs a reference to the World..\nd it'll need to add, m",-,'. and remove controls i"II,,' 'Wll forms, sc il needs a reference to each of those forms:

```
publ1c class Renderer
    private World world;
    private HiveForm h1veForm;
    private FieldForm f1eldForm;
```

Start your Renderer class with these lines We'll add to this class throughout the chapter.

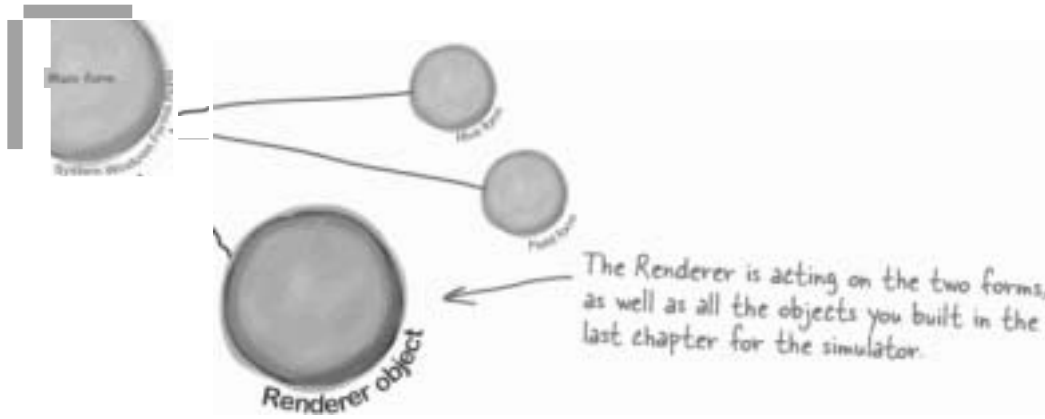
3 **The renderer** uses dictionaries to keep **track** of **the** controls

World k,"'!" track or its Bee objects using a List <Bee > and a Ljst .cj'Lcwer> to 'Ion' its flowers. Tite renderer needs I" Be able tu look at each or those Bee and Flower obj,:cr, ,,,,,I figure "," what BeeContro1 ;",,1 PictureBOK they corresp""t 1" - Ot. if it can't lind" corresponding control, it needs to create 0"" . So here's a perfect opportunity to use dictionaries. So we'll n,·ed 'Wll more private fields in Renderer:

```
    private Dictionary<Flower, Pfct.urefsox> tLowerLcckup
        new mcticne.rvcriover, PicrureBox>();
private Diction"ry<Bee, BeeControl> beeLookup
    ne"' Dictionary<Bee, BeeControl>();
```

These dictionaries become one-to-one mappings between a bee or flower and the con·rol for that bee or flower.

These two dictionary collections let the renderer store exactly one control for each bee or flower in the world



6 **The bees and flowers already know their locations**

There's a reason we stored each bee and flower location using a `Point`. Once we have a `Bee` object we can easily look up its `BeeControl` and set its location.

```
beeControl = beeLookup[bee];
beeControl.Location = bee.Location;
```

For each bee or flower, we can look up the matching control. Then, set that control's location to match the location of the bee or flower object

5 **If a bee doesn't have a control, the renderer adds it to the hive form**

It's easy enough for the renderer to figure out if a particular bee or flower has a control. If the dictionary's `ContainsKey()` method returns `false` for a particular `Bee` object, that means there's no control on the form for that bee. So `Renderer` needs to create a `BeeControl` to add to the dictionary, and then add the control to the form. It also calls the control's `BringToFront()` method to make sure the control doesn't get hidden behind the flower pictures):

```
if (!beeLookup.ContainsKey(bee)) {
    if (!beeLookup.ContainsKey(bee)) {
        beeControl = new BeeControl() { Width = 40, Height = 40 };
        beeLookup.Add(bee, beeControl);
        hiveForm.Controls.Add(beeControl);
        beeControl.BringToFront();
    }
    else
        beeControl = beeLookup[bee];
```

ContainsKey() tells if the bee exists in the dictionary. If not, then we need to add that bee, along with a corresponding control

BringToFront() ensures the bee appears "on top of" any flowers on the FieldForm, and on top of the background of the HiveForm

Remember how a dictionary can use anything as a key? Well, this one uses a Bee object as a key. The renderer needs to know which BeeControl on the form belongs to a particular bee. So it looks up that bee's object in the dictionary, which spits out the correct control. Now the renderer can move it around

Add the hive altd field forl\ls to the project

Th.. is a PictureBox control with its Backgrou"dl..age set to the outside hive picture. When you load the hive pictures into the Resource Designer, they'll sh up in the list of -e rees when y click the "..." butt next to Backgrou dl..age in the Properties w,ndow

Now you need forms 10 put bees Oll_So start with your ..**xisting** B....hiv.. Simulator proj**ec**•. and use "Add >> Existing Item..." 10 **add** your new Bee**Control** u**ser control**. Then add two more \\'indo\\" ronn, 10 the project by **right-clicking** on th.. proj**ec**t ill II", Solution Explorer and choosing "Window Form..." fron II,,' Add III'.'". If you name Ih,' files HiveForm.•s a"d FieldForm.cs, Ihe 11)E will automa ically set Ih"ir Name properti," to `HiveForm` and `Fie1dForm`. You already know that forms are just objects, '" `HiveForm` and `Fie1dForm` are really just two more classes.



Make sure *t*-- resize both for... s" they look like these screenshots.

Set the form's BackgroundImage property to the inside hive picture.

You'll need the inside and outside hive images—"Hive (inside).png" and "Hive (outide).png"—loaded into your resources. Then add these two forms. Set each form's FormBorderStyle property to FixedSingle (so the user can't resize it), the ControlBox property to false (to take away its minimize and maximize controls), and StartPosition to Manual (so its Location property is settable).

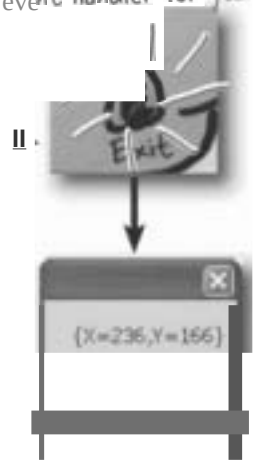
Re..e..ber, go to the Properties window, click icon to bring up the Events window, sc..oll do and double-click on it. The IDE will add the event handler for you.

igh -S-bolt to the MouseClick row

Figure out where your locations are

You need lo figure oul wh..re rh.. hive i, on your FieldForm. Using Ih,' Properties window, crear" a handler 1;,1' the MouseClick event, a"d add this code:

```
private void Form1_MouseClick(object sender, MouseEventArgs e) {
  MessageBox.Show(e.Location.ToString());
```



Now you ('all n m Ih,' fo,rm. and click on Ihe location of Ih,' hive. That will give you a se-t <lr coordinates for your hive on the field.

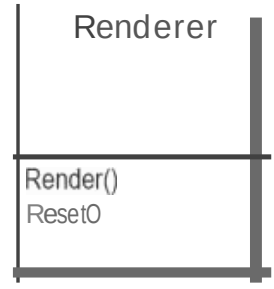
Add the same handler (<1 HiveFo,,-m. and run it. Then, by clicking, get the coordinate or the exit, the nursery, and Ih,' l"oney factory. Using all Ihese l,cations, yc.u can update Ihe `ResetLocations()` method you wrote in the `Hive` class in (h,' last chapter:

```
private void ResetLocations()
{
    locations = new Dictionary<string, Point>();
    locations.Add("Entrance", new Point(600, 100));
    locations.Add("Nursery", new Point(95, 174));
    locations.Add("HoneyFactory", new Point(157, 9BJI;
    locations.Add("Exit", new Point(194, 213));
}
```

Remove the mouse click handler when r- re d one... you just eded it to get the locations your forms

These are the t dinates that worked for us but if you're for..s a little bigger aller, your *t*oordinate• ..,ll be different

Build the Renderer



Here's the complete Renderer class. The main form calls Ihi, class's Render () method rialu after it calls World.Go () to draw the- bees and flowers on the forms. You'll need [o make sure that th,- flower graphic (Flower.png) is loaded illl" the project, just like Ih,>an;lllalt'd bee images.

```
public class Renderer {
    private World world;
    private HiveForm hiveForm;
    private FieldForm fieldForm;

    private Dictionary<Flower, PictureBox> flowerLookup =
        ne'W Dictionary<Flower, PictureBox>(J;
    private List<Flower> deadFlowers = new List<Flower>();

    private Dictionary<Bee, BeeControl> beeLookup -
        new Dictionary<Bee, BeeControl>();
    private List<Bee> retiredBees • new Llst<Bee>();

    pUblic Renderer(World world, HlveForm hlveForm, FleldForm field,orm) (
        this.world = world:
        this.hiveForm = hiveForm;
        this.fieldForm = fieldForm;

    public void Render()
        Dra1 Bees();
        Dra1 Flowers();
        RemoveRetiredBeesAndDeadFlowers();


    public void Reset() I
        foreach (PictureBox flower in flowerLookup.Values)
            tieldForm.Controls.Remove(flower);
            flower.Dispose();

        foreach (BeeControl bee in beeLookup.Values)
            hiveForm.Controls.Remove(bee);
            tieldForm.Controls.Remove(bee);
            bee.Dispose();

        flowerLookup.Clear();
        beeLookup.Clear();
```

The renderer keeps references to the world and the two forms it draws the bees on.

The world uses Bee and Flower objects to keep track of every bee and flower in the world. The forms use a PictureBox to display each flower and a BeeControl to display each bee. The renderer uses these dictionaries to connect each bee and flower to its own BeeControl or PictureBox.

When a flower dies or a bee retires, it use, the deadFlowers and retiredBees lists to clean out the dictionaries.

The timer on the main form that ....ns the ani..ation calls the Render() ..ethod, which updates the bees and the flowers, and then cleans out its dictionaries

If the .i..ulator i eset, it calls the Re..oveAllC trols() ..ethod to completely clear t the controls on the two forms. It finds all of the controls in each of its two dictionaries and .e..oves the.. f..om the forms, calling Dispose() on each of the.. Then it elea.., the two dictionaries.

It takes two foreach loops to d..a.. the flowers The first looks for ne... flowers and adds their Pict..reBoxes. The second looks for dead flowers and removes their Pictu..eBoxes

```
private void Dr....Flowers() (
    foreach (Flower flower in world.Flowers)
        it (!tlowerLookup.ContainsKey(flower))
            PictureBox flo_reontrol - ne" PlctureBoxl)
                Width - 45.
                Height - 55.
                Image = Properties.Resources.Flower,
                SizeMode • PictureBoxSizeMode.StretchImage,
                Location - flower.Location
            |;
            flowerLookup.Add(flower, flowerControll;
            fieldFona.Controls.Add(tlowerConrroll;

    foreach (Flower flower in flowerLookup.Keys) {
        if (!world.Flowers.Contains(flower)) I
            erceureecx fteeercceretToRemove = flowerLookup[flowerJ;
            fieldform.COntrcre.aeecve(frcvercenteorToRemove);
            flowerControlToRemove.Dispose();
            deadFlowers.Add(flower);
```

The first foreach loop uses the flowerLookup dictionary to check each flowe.. to see if it'i got a control on the for.. If it doesn't, it creates a new PictureBox using an object initializer, adds it to the form, and the.. adds it to the flowerLookup dictionary

DrawFlowers() uses the Location property ... the Flower object to set the PictureBox's location on the form

The second foreach loop looks for any PictureBox in the flowerLook.p dictionary that's .... longer .. the for.. and removes it

After it removes the PictureBox, it calls its Dispose() method Then it adds the Flower object to deadFlowers so it'll get cleared later

```
private void Dra..Bees() I
    eeeccneeer eeeccnerol;
    rcreeen (Bee bee in world.Bees)
        eeeccneect - GetBeeControl(beel;
        it (bee.InsideHive) {
            if (tieldfon'.COntrols.ccnt.arns(beeControl))
                MoveBeeFromFieldToHive(beeControl);
          else if Ihiveform.Controls.ContainslbeeControlll
            HoveBeeFromHiveToField(beeControl, bee);
        beeControl.Location   bee.Location;

    foreach taee bee in beeLookup.Keys) (
        if (!world.Bees.Contains(bee)) (
            eeeccnerct • beeLookup[bee];
            it (tieldform.Controls.ccntetns(beecontrol))
                fieldform.Controls.Remove(beeControl);
            if (hive-form.Controls.Contains(beeControrOl))
                hiveform.Controls.Remove(beeControl);
            beeControl.Dispose();
            retiredBees.Add(bee);
```

Dra··Bees() also uses two foreach loops, and it does the same basic things as Dra··Flowers() But it's a little more complex, we split some of its behav— t into separate methods to make it easier to understand

Dra..Bees() checks if a bee is in the hive but its control is on the FieldForm, or vice versa It uses two extra methds to move th.. BeeControls between the for s

Once the BeeControl is re..oved, we need to call its Dispose() ..ethod—the user control ...ll dispose of its ti...er for us.

The second foreach loop works just like ... Dra..Flowers(), except it needs to re..ove the BeeControl fr_ the right form

You'll n....d 10 mak....ure you've got using System.Drawing and using Syst em.Windows .Forms III Ih.. lop or the R..nderer cla.... file.

GetBeeControl() l....ks up a bee in the beeLookup dictionary and returns it. If it's not the..e, it create· a new 40 x 40 BeeControl and adds it to the hive for.. (since that's ..here bees are born)

```
pIivate eeeccntecr GetBeeControl(Bee bee)
    seeccneect beecontroI,
    it (!beeLookup.ContainsKey(bee)) (
        beeControl = new BeeControl() { Wdth   40, Height. 40 };
        beeLookup.Add(bee, eeeccntrcrj·
        hiveForm.Controls.Add(beeControl);
        beeControl.BringToFront();
    |
    else
        beeControl · beeLookup[bee];
    Ieturn beeControl;
```

Don't forget that the ! means NOT.

MoveBeeFromHiveToField() takes a specific BeeControl out of the hive for.... Controls collection and adds it to the field form's Controls collection

```
private void MoveBeeFromHiveToField(BeeControl eeeccrarcr, Bee bee) {
    hiveForm.ccnerere.seeeve(beeControl),
    beeControl.Size = new sieerao. 20);
    fieldForm.Controls.Add(beeControl);
    beeControI.Bting1'oFtOnt();
```

The bees the ones the field form are smaller than the hive form, so the method needs to change BeeControl's Size property

```
private void MoveBeeFromFieldToHive(BeeControl beeControll {
    tieldForm.Controls.Remove(beeControl);
    beeControl.Size _ new Size(40, 40):
    hiveForm.Controls.Add(beeControl);
    beeControl.BringToFront();
```

MoveBeeFromFieldToHive() moves a BeeControl back to the hive form It has to make it bigger aga...

```
private void RemoveRetiredBeesAndDeadFlowers() {
    rcreaen (Bee bee in retiredBees)
        beeLookup.Remove(bee);
    retiredBees.Clear(),
    toreach (Flower flower in deadFlowers)
        flowerLookup.Remove rrlower);
    deadFlowers.Clear();
```

Whenever Dra..Bees() and DrawFlowers() found that a flower or bee was no longer in the world, it added them to the deadFlowers and retiredBees lists to be removed at the e d of the f..a..e

After all the controls are ...oved around, the renderer calls this ..ethod to clear any dead flowers and retired bee.....t of the tw.. dictionaries

Now **connect** the **main** forlll to your two **new forms,** HiveForlll **and** FieldForllI

It's great ,,, have •• Renderer. but s<, far. there aren't any forms to render onto. We can fix ,h.ll by going back to Ih.· main Form class (probably called FormII, and making some ,·.ide changes:

When the main form loads, it creates an instance of each of the other two forms. They're just objects in the heap for now—they won't be displayed until their Show() methods are called.

```
public pert.LaI class Forml ; Form {
  HiveForm hiveForm _ new HiveForm();
  FieldForm fieldForm _ new FieldForm();
  Renderer renderer;

  // the rest of the fields

  public ForllliO |
    InitializeComponent();

    MoveChildForms();
    hiveForm.Show(this);
    fieldForm.Show(this);
    ResetSimulator();

    // The rest of the code stays the same
```

Move the code to instantiate the World into the ResetSimulator() method

The form passes a reference to itself into Form.Show() so it becomes the parent form.

The main form's constructor moves the two child for s n place, then d..plays the... Then it calls ResetSimulator(), which nstantiates Renderer

Everything else that used to be in the constructor will be moved to the ResetSimulator() method.

```
private vere MoveChildForms 0 (
  hiveForm.Location _ new Point(Location.X + Width + 10, Location.Y);
  fieldForm.Location _ new Point(Location.X.
    Location.Y + Math.Max(Height, hiveForm.Height) + 10);
|

public void RunFrame(object sender, EventArgs e) I
  framesRun++;
  world.Go(random);
  renderer.Render();
  // previous code
```

This code moves the two f s so that the hive for " next to the main stats form and the field form .. below both of the...

Adding this one line to RunFrame makes the simulator update the graphics after each time the world's Go() method is called.

```
private vold Form1_Move(object sender, EventArqs e) |
  MoveChildForms();
|
```

The M..ve event is fired every ti...e the main form is moved Calling M..veChildFormsO makes sure the child for s always ..ove along ith the main form

Use the Eve··ts button in the Properties window to add the Move event handler.

Make e you've set the ield and hive f ' StartP. ition property to Manual, or ebe M eChildFormsO won't work

Here's where we create new instances of the World and Renderer classes, which resets the simulator.

```
private void ResetSimulator() {
  framesRun = 0;
  world = new World(new BeeMessage(SendMessage));
  renderer = new Renderer(world, hiveForm, fieldForm);
```

```
private void reset_Click(object sender, EventArgs e) {
    renderer.Reset();
    ResetSimulator();
    if (!timer1.Enabled)
        toolStrip1.Items[0].Text = "Start simulation";
```

The Reset button needs to call Reset() to clear out all the BeeControls and flower PictureBoxes, and then reset the simulator.

```
private void openToolStripButton_Click(object sender, EventArgs e)
  // The rest of the code in this button stays exactly the same.

  renderer.Reset();
  renderer = new Renderer(world, hiveForm, fieldForm);
```

Finally, you'll need to add code to the Open button on the ToolStrip to use the Reset() method to remove the bees and flowers from the two forms' Controls collections and then create a new renderer using the newly loaded world.

there are no Dumb Questions

Q: I saw that you showed the form using the Show() method, but I don't quite get what was going on with passing `this` as a parameter.

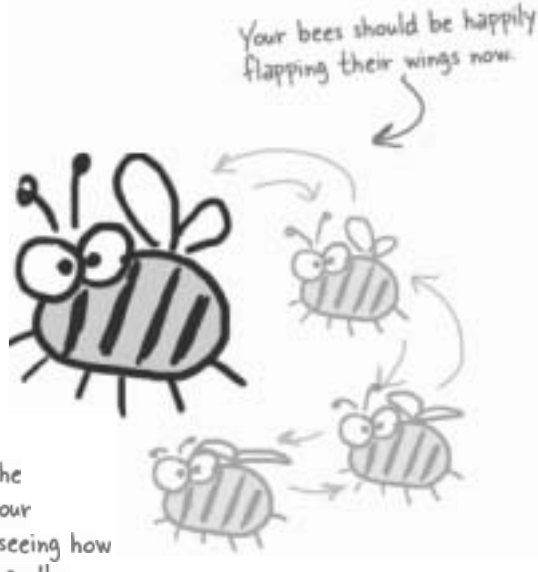
A: This all comes down to the idea that a form is just another class. When you display a form you're just instantiating that class and calling its Show() method. There's an overloaded version of Show() that takes one parameter, a parent window. When one form is a parent of another, it causes Windows to set up a special relationship between them—for example, when you minimize the parent window, it automatically minimizes all of that form's child windows, too.

Q: Can you alter the preexisting controls and muck around with their code?

A: No, you can't actually access the code inside the controls that ship with Visual Studio. However, every single one of these controls is a class that you can inherit, just like you inherited from PictureBox to create your own control. If you want to add or change behavior in any of those controls, you add your own methods and properties that manipulate the ones in the base class.

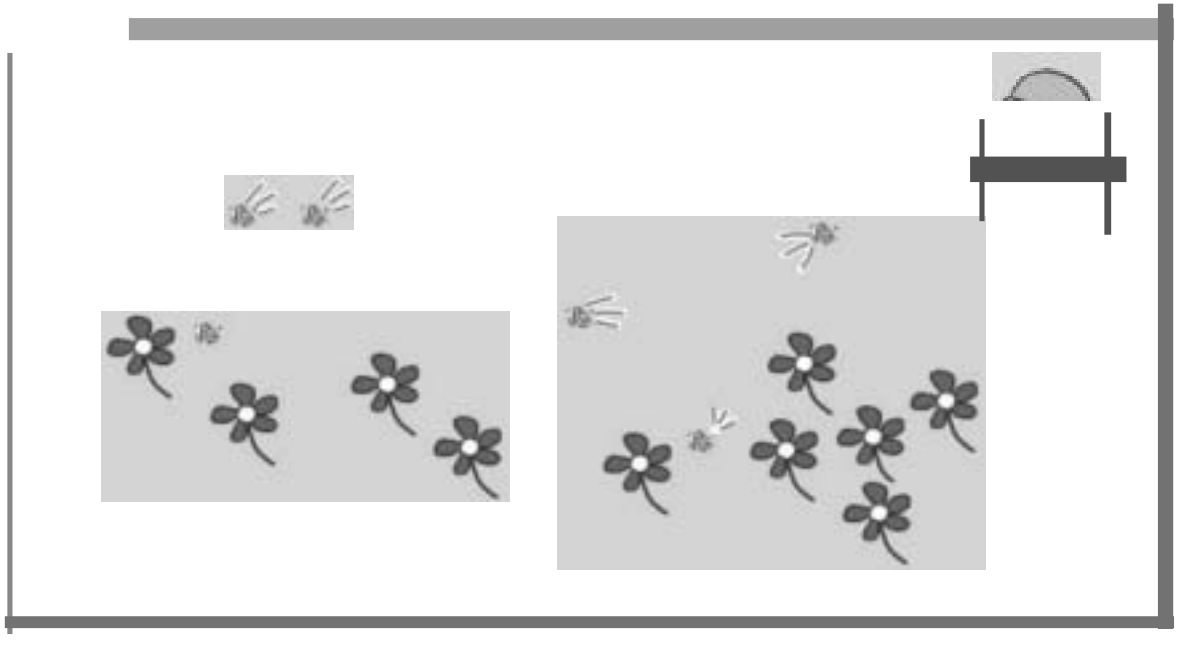
rest drive... **ahem**... buzz

Compile all your code, chase down any errors you're getting, and nm your simulator.





Try changing the constants on your simulator, and seeing how the Renderer handles more bees or flowers.



Looks great **but something's not** quite **right...**

Look closely "I the bees buzzing around *the* hive "lid the flowers, and you'll notice some problem with lh,- way they're b..ing rendered. Remember how you set "„,'h BeeControl's Backcolor property 10 Color.Transparent.? Unfortunately, tha wasn't enough to keep the simulator from having some problems which are actually pretty typical 0f graphics programs.

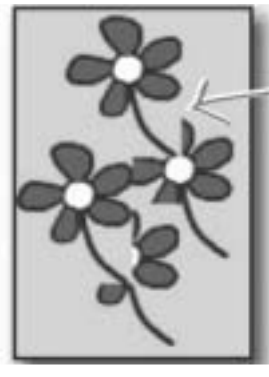
O **There** are some serious **performance** issues

Did y<UII'tice h,w the "h,le simulat,,'sl,'" dlrwn when "ll the h,...s are inside the hive? If ""I. try adding more bees by increasing the constants ill the Wo r l d class. Keep your eye Oil the frame ra1e- add "",,.' b.es ,,,,,Iil starts to drop significantly.

2 **The** flowers' **backgrounds** aren't really **transparent**

And there's another, completely ***separate*** problem. When we saved the graphics files I,, the flowers, we gave them transparent backgrounds. fill! while that made sure Ihat each flower's background matched the background of the fiu'm. it doesn't look '0 ni,e wlu-n flowers overlap each other.

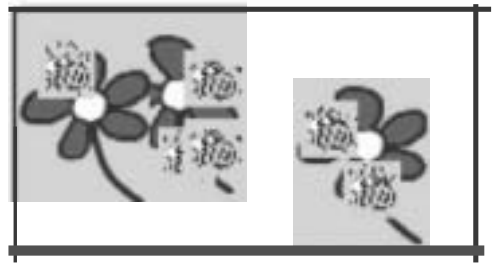
When you set a PictureBox's background color to Transparent, it draws any transparent pixels in the image so they match the background of the form... which isn't always the right thing to do.



When one ictureBox overlaps another, C# draws the transparent pixels 'o they ..atch the for.., not the other contr..l that it overlaps causing weird .ectangular "cut-outs" any time two flowers overlap

3 **The bees'** backgrounds aren't transparent, **either**

It turns "lit that Color,Transparent really doe h,l\" some limit'lti"",." 'hell th,' bees ar,' hovering over the flowers, th,' same "nlt.out" glitch happens. Transparency works a lillle better with th,-hive form, wh,-n- th,' form's ha,kground image does ,Ill'" through the tr'lll'p,Ir<'IH areas of of the bee graphics. [lilt when the bees overlap, the same problems occur. ,\,,<l if ,-,," watch closely as the h,es move around the hiv-. you'll see some glitche when · the bee image are sometimes distorted when they move.





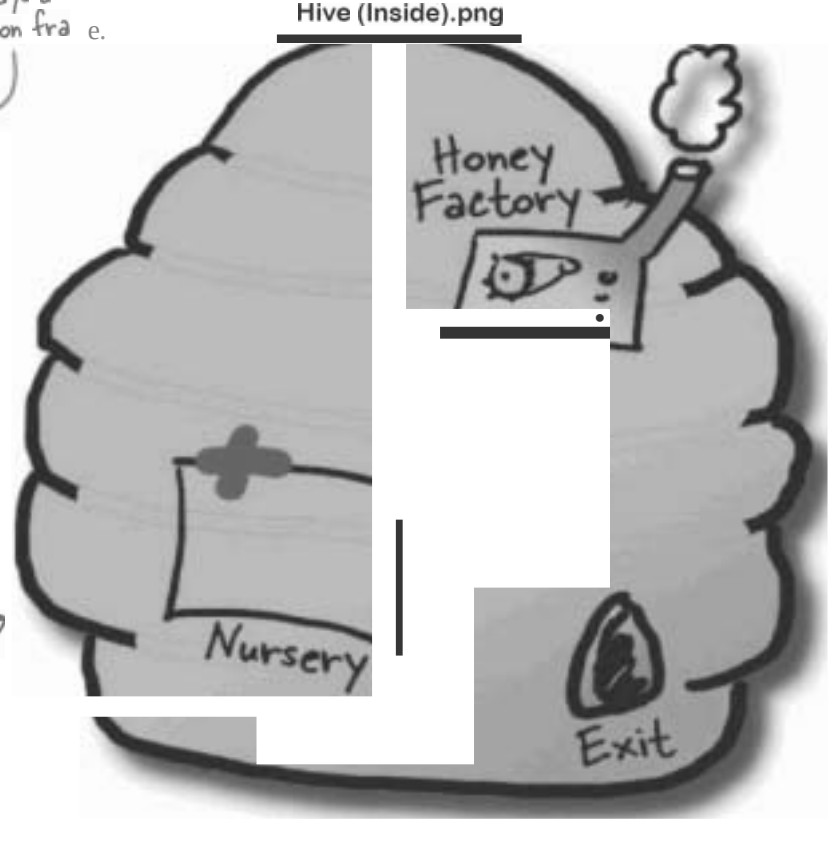
Let's take a closer look at those performance issues

Each bee picture you downloaded is big. Really big. Pop one of them "I"'u in Window Pic ture Viewer and see fi,r \'"ur,,·lf That means the ['i-Iun'B ox needs 10 shrink ll down every lillle il changes the image, and scaling an image 111' "r tl,)wn take time. The reas<>u the bees move a lut ,[",\('r when there's a ["I "I' 11"'1ll flying around inside lite hive is because the inside hive picture is HUGE. '\Ild when you made the background fi>r Ih,' BeeC'111 r<,1 transparent, il needs to d" double work: first it has I', shrink the I,." , picture down, and then il needs I" shrink a portion of the fu'm" background down so that it can draw il in lite transparent area I'"hind Ih" bee.



The bee picture i• .eally big, and the PictureBox needs ti e to shrink it d eve..y ti...e it displays a new ani..ation fra e.

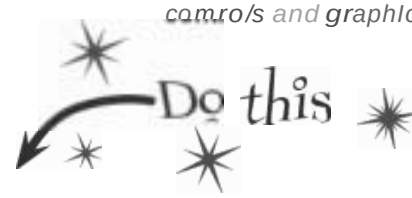
The graphics files for the bees are really BIG. When the PictureBox needs to scale the picture down to size every time it displays a new animation frame. That takes a lot of time•••



The inside hive picture is huge Every ti...e a bee flies in front of it, its PictureBox needs to scale it down to the size of the control. It needs to do that to show pa..t of the picture any place the bee pictu.e's transpa..ent background lets it show through.

.••so all we need to do to speed up the simulator's performance is to shrink down all the pictures before we try to display them.

All we need to do" to speed up the graphics performance is, add a method to the renderer that s ,1,·, any image· to a different size. Then we can **resize each pictu once** whO'n **it's load**..d, and only use (h,' scaled down version in the bee control and fi>t the hive f("111" background.



1 Add **the ResizeImage method** to **the** renderer-

.\ll Or the picture in your project (like `Proper t.Ies.Resources.Flowe r`) are stored as `Bitmap` objects. Here's a , 1a1i,' method that resizes bitmap - add il I" the Renden-r class:

```
pUblic static Bitmap Re..izeImage(Bitmap picture, int width, int height)
    Bitmap resi aecpi crure - new Bitmilp(width, height);
    using (Graphics graphics _ Graphics.Fromlmdge(resizedPicture))
        graphics,Drawlmagelpicture, 0, 0, width, height);
    }
    return re"izedPicture;
```

We'll take a close.. look at what this Graphics object is and ho·· th·· ··ethod works in the next few pages

2 Add this **ResizeCells** method to your BeeControl

Yo>t,rBeeControl can sto,n' its own lli("",,1' objects—an array of fi,ur "r them. Here's a control that'll populate (ha(array, resizing each om' ," that il" exactly the right size for the control:

```
private Bitmap[] cells - new Bitmap!41'
private void
  cei ierot                                                    Width, Height);
  cells(ll                                                     Width, Height);
  cells(21                                                     Width, Height);
  cells!31                                                     Width, Height);
```

3 **Change** the **switch statement** so that it **uses the** cells arroy, not the **resources**

The BeeControl's Ti,'k event h.mdb-r has a swit:h statement that sets its BackgroundImage:

```
Backgroundlmage _ Propertles.Resources.Bee anlmatlon_1;
```

Replace `Properties.Resources.Bee_animation_1` with `cells[OJ`. Now replace the rest Of the `case` lines, so that case 2 one use `ce11s[1]`• case 3 use `cells[2]`• case -\ uses `cells[3]`• case 5 use `Ce11s[2]`, and tl". default case uses `ce11s[1]`. That way only the resized image is displayed.

4 Add calls to **ResizeCells()** to the **BeeControl**

Yo,u'll need I" add two calls Itl the new ResizeCells() metho,d. First, add it 10 Ih,' IKutto,m "f (h,' constructor amI **chang.. th.. BackgroundImag..Layout** prop..rty to None. Th,'" go back (0 Ih,' IIIE designer by double-clicking Oll (h,' BeeControl in lh,' Properite window. (;" ,ver to the Events page ill the Propertie window (by clicking on Ihe lightning holt icon). scroll down I" Resize, and double-click "11 il l" **add a** R..**siz**.....nt handlOr_ Make the new Resize event handler call ResizeCells() too> that wav it'll resize iI" animation picture every lim,· the fo'I'm is resized.

5 **Set the form's** background **image** manually

Go (" Ill" Pro'I"'r(i," window ;" ,,1set the hive f,'rm's backgrou,d imago I" {none). Th"n go (0 its constructor and set the image Itl "n,' that's sized properly.

```
public partial class HlveForm : Form {
    public HiveForm() {
        rnttialiaecomponent(),
        Backgroundlmage - Renderer.Resize1mage I
                Properties,Resources.Hive__inside_,
                ClientRectangle.Width, ClientRectangle.Height);
```

Your form has a ClientRectangle property that contains a Rectangle which has the dimensions of its display area

Now run the simulator-It's **much** fasterl

You resized your **Bitmaps using** a **Graphics** object

Forms and controls have a CreateGraphics() method that returns a new Graphics object. You'll see a lot more about that shortly.

Let's take a cl..ser I,ok at that ResizeImage () me-rh..d you added I.. the renderer. The first thing it does is creat.· a new Bi tmap object that's the ,jz.· that the picture will be resized to. Then il uses Graphics . FromImage () l" c**reat**.. **a** n..w **Graphics** obj..**ct**. It uses that Graphic object's Dr a wI mage ( ) method to draw th.· p;,'[ur,>'>Illo the Bi t map. N(,tice how y(" passed th,' width and h e i g h t parameters to DrawImage (} Ihat's h..w vou tell ;t to scale the image down l" the new size. Finally 'uu n-turm-d the m-w Bi t map y(," created, s(it ran be used as tltt" form's background image , ,,-one or Ih,' four animation cells.

You pass a picture into the method, along with a new width and height that it'll be resized to.

```
public static Bitmap ResizeImage(Bitmap picture, int width, int height) (
    Bitmap resizedPicture = new Bitmap(width, height);
    using (Graphics graphics = Graphics.FromImage(resizedPicture))
        graphics.DrawImage(picture, 0, 0, width, height);
    }
    return reslzedPicture:
```

The FromImage() ...ethod returns *a* new Graphics object that lets you draw graphics onto that image. Take a m,nute and use the IDE' IntelliSe"ie to look at the methods in the Graphics class. When you call DrawImage(), it copies the image into the resizedPicture bitmap at the location (0, 0) and scaled to the width and heJ.ght para..eters..

let', **see** Image r"lzI.g I. actio.

Just do this temporarily. Delete the button and code when you r[e] done.

Drag a button onto tht" Field form and add thi, code. h creates a new PictureBox control that's 100 x100 pixels, setting its border to a black line "' you can "ie how big it is. Then it uses Resi zeImage ( ) to lll"k,' a bee picture that's squished <l,"' 11 r" 80Ix 40 pixels and assigns that new picture to it, Image pr<>perty. Once the PictureBox i, added to the form, the 1,," · is displayed.

```
private void buttonl_Click(object sender, EventArgs e)

    Pfct.urebox beePkct.ur-e = new PictureBox O ;
    beePicture.Location - new Point(l0, 101;
    beePicture.Size - new Size(l0O, 100);
    beePicture.BorderStyle = BorderStyle.FixedSingle;
    beePicture.lmage = Resizelmage(
        Properties.Resources.Bee_animation_1, eo, 40);
    Controls.Add(beePicturel ;
```

You can see the image resizing in action—the ',\wished bee image is much aller than the Pict...eBox. Resizel age() ,,\wished it down.



The ResizeImageO **method** creates a Graphics object 10 **draw** on an **invisible** Bilmap **object.** II returns Ihal Bitmap **so** it can **be** displayed on a **form or** in PietureBox.

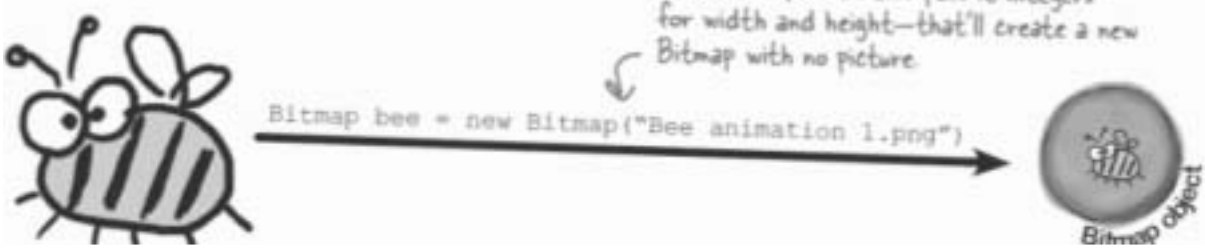
Your **image** resources are stored **in Bitmap** objects

Whe11 you i"'I"'.I graphics files illt" your project's resources, what happens I" them? You already know that you can access them using properttes.nesourcee. llut wh.u. exactly, is your progr.ull doing with tlnem, tlney r.. imported?

.NET tu'-n> \1",- image into a **Bitmap** obj..ct:

The Bitmap class has several overloaded constructors. This one loads a graphics file from disk. You can also pass it integers for width and height—that'll create a new Bitmap with no picture.

```
Bitmap bee = new Bitmap("Bee animation 1.png")
```



B•• animation 1.png

The" each **Bitmap** ls draw" to the ,cree"

(J"ce youu images are ill Bitmap objects, your form draws them t" the screen, with a call like this:

This call gets a Graphics object to draw on the form We use a using statment to make sure the Graphics object is disposed

```
using (Graphics g = CreateGraphics(» (
    g.DrawlmageCmyBitmap, 30, 30, 150, 150):
|
```

DrawImage() takes a Bitmap, the image to dra.. _

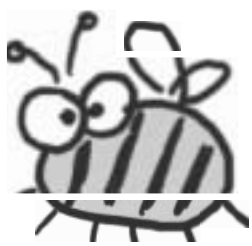
...a starting X, Y coordinate...

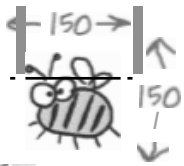
and a size, 150x150 pixels

The bigger they are...

I)id you notice those last two parameters I" `DrawImage()`? \\1a' if tl.·· image in the *Bi* `tmap` is 175 by 175? The graphics library must then resize the irn•ge ,n Iii 150 by 150. What if .1... `Bitmap` ıtains an image that's 1.500 bv 2,025? -11,,·11 the scaling becom even trickier...

This image, which is 300x300 pixels..





„ gets shrunk to this size, which " (for example) 150x150 pixels And that slow r-r simulator down!

Resizing Images takes a lot of **processing power! If** you do it **once,** it's no big deal. But if you do it EVERY fRAME, your program will slow down. We gave you REALLY BIG images for the bees and the hive. When the renderer moves the bees around (especially In "on' of 'ho In.ldo hlvo **picture),** it has to **resize** them over and over again. And that was **causing** the performance problemsl

Use Systelll.tlrawil1g to TAKE CONTROL of graphics yourself

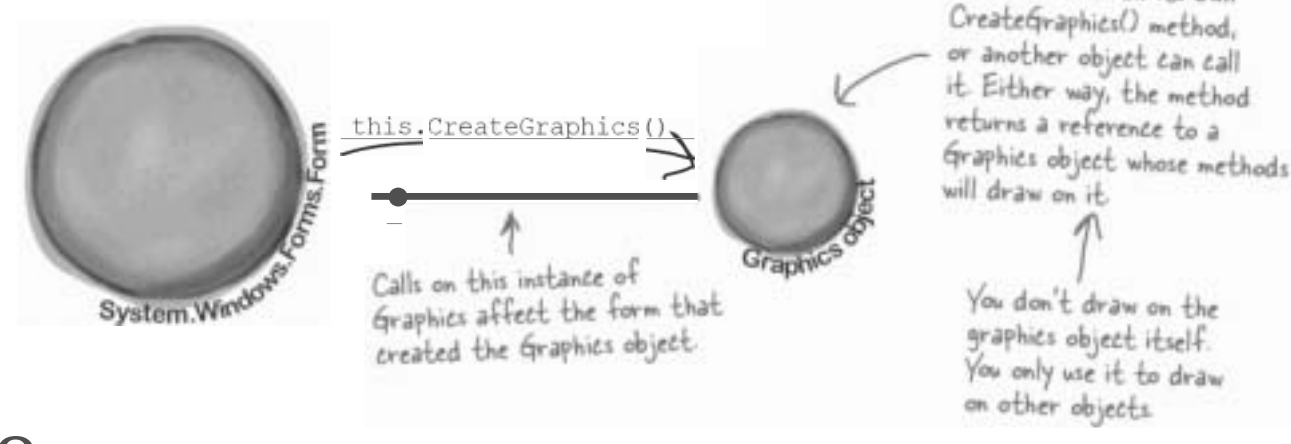
The Graphics ,bj""t is part "I' the System.Drawing namespace. The .NET Framework comes with some pretty powerful graphics tools that g(a lot funlu-r than the simple PictureBox control that's in the toolbc.x. You can draw shapes, use fonts, and do all sorts of complex graphics.., ar,d it all starts Wil11 a (;raphics object. ,\11\' time you want to add or IIH..lify any object's graphics or images, you'll create a (;ral'hi,', object that's link..d to th.. ohject }'on want to draw on, and then use th,' Graphics object's methods to draw 011 your target.

System.Drawing

The graphics methods in the System.Drawing namespace are sometimes referred to as GDI+, which stands for Graphics Device Interface. When you draw graphics with GDI+, you start with a Graphics object that's hooked up to a Bitmap, form, control, or another object that you want to draw on using the Graphics object's methods.

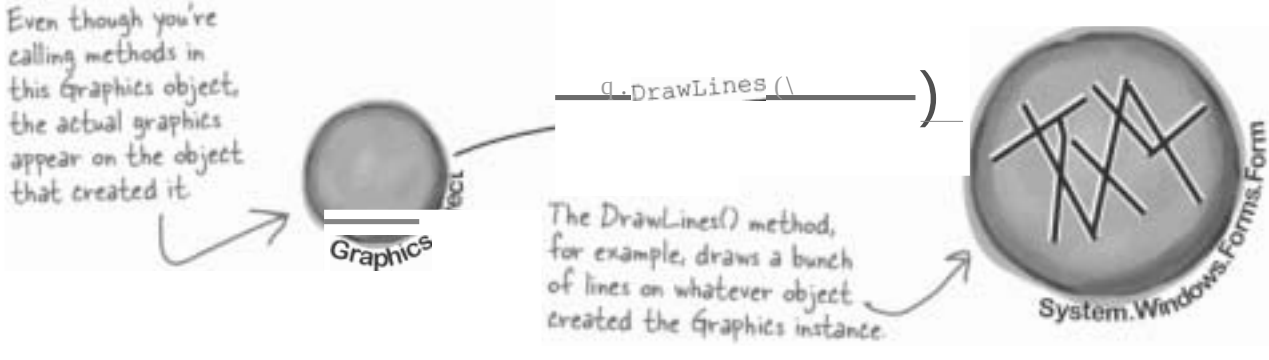
o Start with the object you wont to draw on

For i"'1a"n', think about a form. When you call the form's CreateGraphics() lllidho,l, il returns an instance of Graphics that's "'1111' t" draw nil itself



o Use the **Graphics** object's methods to *drow* on **your** object

Every Graphics object has methods that let vou draw on the object that created it. When you call methods in the Graphics object In draw lines. circles rectangles, text. ,,,,d images, they appear "'" the I;orm.



You'll need to make sure you've got a using System.Drawing; line at the top of your class to use these methods. Or, when you add a form to your project, the IDE adds that line to your form class automatically.

A 30-second tour of GDI+ graphics

There ,In" all sorts of shape ,,,,d l';'-(Urt·, that you can draw once you've created a Graphics object. All you need to d" is call its methods, and it'll draw directly onto tlw object Iha, created it.

- The first step is always to grab yourself a Graphics object. 1Ise a form's CreateGraphics() ",,,thod. or have a Graphics "bj""l passed ill. Remember, Graphics i"'pl""[('l1l' the IDisposable() interface-. s(if you create a new one use a using statement:

```
using (Graphics g _ this.CreateGraphics()) {
```

Re...e..ber, this draws on the object that created this instance.

- If you want m draw a line, ,-all DrawLine() with starting point a",l ..nding point, ..ach represented by X aud Y coordinates:

The start coordinate

```
g.DrawLine(Pens.Blue, 30, 10, 100, 45);
```

_ and the end coordinate.

or you can do i, using a couple "l' Points:

```
g.DrawLine(Pens.Blue, new Point(30, 451, new Point(l00, 10));
```

Here's code that draws a filled slate gray rectangle, and then gives i(a sky blue border. I' uses a Rectangle (0 define the dimensions— in this case, (h,' upper left hand corner is at (150, 15), and it's 140 pixels wide and 90 pixels high.

There are a whole lot of colors you can use—just type "Color", "Pens" or "Brushes" followed by a dot and the IntelliSense window w;ll display them

```
g.FillRectangle(Brushes.SlateGray, new Rectangle(150, 15, 140, 90));
g.DrawRectangle(Pens,sxvaiue. new Rectangle(150, 15, 140, 90));
```

- You can draw an ellipse or a circle using the DrawCirc Le() or Fi11Circle() m"(hod" which also use a Rectangle (o specify how big the shape ,hould 1)('. This code draws two ellipses that are slightly offset (" give a shadow effect:

```
g,FillEllipse(Brushes.DarkGray, new Rectangle(45, 65, 200, 100));
g,FillEllipse(Brushes,Silver, new Rectangle(40, 60, 200, 100));
```

- Usc the DrawString() method to draw text in any font and color. T(do that, you'll need (o create a Font object. I, illll'l""H'lll' I Disposable. so use a using statement:

```
usIng (Font aria124Bold = new Font("Arial", 24, FontStyle.Bo1d))
   g.DrawString("Hi there!", aria124Bold, Brushes.Red, 50, 75);
```

If the above statements are executed in order, this is what will end up on the form. Each of the statements above matches up with the numbers here. The upper left-hand corner is coordinate (0, 0).

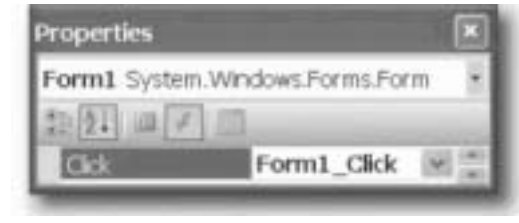


There's no step 1 on this picture, since that was creating the actual Graphics object.

picture on a fo,m when **you** cljck on it.

o Start by adding a Click event to **the form**

Go 10the **Events pag**.. in th.. **Prop..rti..s window** (by clicking on the lightning-bolt icon), scroll d"\\lllO the Click event and double-click "" it.



Start the event handler "ilh a `using` li",-10create the Graphics object. When you work with (;111+_you use a lot "l" objects that implement `IDisposable`. 1l' you don'l dispose of them. they'll slowly '1I"k up your computer's resources uUli! you quit the program. So you'll ,-",l up using a 101 "f `using` statements:

```
using (Graphics 9 - CreateGraphics() )
```

He-e's the first line in your Forml_Click() event handler ..ethod We'll give you all the lines for the event handler—put the.. together to draw the pictu..e

o Pay attention to **the** order you draw **things** on our **form**

We want a sky blue background !;,r this picture, so you'll draw a big blue rectangle first - the-n anything else you draw afb-rwards willIoe drawn on IOP "f it. You'll take advantage of one "r the form's properties called `ClientRectangle`. It's a `Rectangle` that define Ilie I"'llml,,,;'" of the form's drawing area. `Rectangles` are really useful - you ,'an create a new rectangle by specifying a `Point` h,r its "pl"-r left-hand corner, it, width and its height. (l,we you do that, it'll automatically calculate its `Top`, `Left`, `Right` and `Bottom` properties I;" you. ,\nd it's got **useful methods** like `Contains()`, **which will return true** if a **given** point **is inside** it

```
S.FillRectangle(Brushes.SkyBlue, ClientRectangle);
```

This will come in eally handy later on in"the book! What do r- think you'll be doing with Contains()?

e **Draw the bee and the** flower

You already know how ,h,' `DrawImage()` metl..od works.

```
g.DrawImage(Propert.Les.Resources.Bee_anl.matlon 1, 50, 20. 75, 75);
g.Drawlmage(Properties.Resources.Flower. 10, 130, 100, 150);
```

Pens are for d..awing lines, and they have a width. If you want to d..aw a filled shape or some text, you'll need a Brush

o Add a **pen** that you can draw with

Every time you draw a line, you use a Pen object 10 determine its color and thickness. There's a I,uilt-ill `Pens` class that gives you plenty "l' pens (`Pens.Red` is a Ihill red pen, for example). But you can create your own I"'ll using lh,- `Pen` class constructor, which takes a `Brush` object and a Ihi,'kn,'" (il-' a float, so make "'1"' it ,'nd, with F). Brushes are how you draw filled graphics (!ike filled rectangles and ellipses), and there' a `Brushes` class that gives you brushes ill various colors.

```
using (Pen thickBlackPen - new Pen(Brushes.Black, 3,OF))
```

This goes inside the inner us-S statement that created the Pen.

Add (In **arrow** that **points** to **the flower**

There an' some Graphics "",thod, that take au array of Points and ,'"m,,',:t th"", using a series of line or curves. We'll use the `DrawLines ()` method to draw the arrow head, and the `Drevcurve ()` nu-th••l tu draw its shaft. There are other methods that take- I",int arrays, too (like `DravpoLyqon ()`. which draws a closed shape, and `FillPolygon ()`. which fills it in,)

```
g.DrawLines(thickBlackPen, new Point[] {
    new Point(130, 110), new Point(120, 160), new Point(155, 163)});
g.DrawCurve(thickBlackPen, new Point[] {
    new Point(120, 160), new Point(175, 120}, new Point(215, 70) n.
```

Here's where the using block ends—we don't need the thickBlackPen anY more, so it'll get disposed.

When r- pass an array of points to DrawCu e(), it draws a smooth c ve that connects the.. all in order.

Add *0.* font to draw the **text**

\\'henever you work with drawing text, the lir,t thing you need t" do is create a `Font` object. .Again, use a using statement because `Font` implements `IDisposable`. Creating a f(,nt i, straightfi In,'ard. lln-n- are several overloaded constructors— the simplest one takes a font n.mu-. font size, and `FontStyle` "ll11lll,

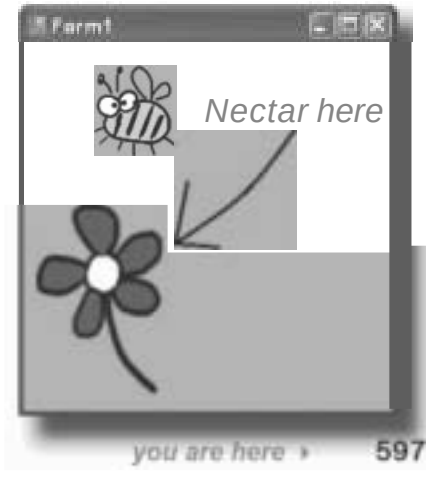
```
using (Font font = new Font("Arial", 16, FontStyle.Italic))
```

Add **some text** that *says* "Nect ar **here"**

Now that you've got a font, you can figure out where t" put the string by measuring how big it will be when it's drawn. The `MeasureString()` method returns a `SizeF` that defines it, size. (`SizeF` is j",t the `float` version of `Size` and hoth of them just defin.. a width and height.) Since we know where the arrow ends, we'll use the string measurements 1" I""ition its center just above the arrow.

```
SizeF size = g.MeasureString("Nectar here", font);
g.DrawString("Nectar here", font, Brushes.Red, new Point(
    215 - (int)size.Width / 2, 70 - (int)size.Height));
```

Make sure you close out both using blocks.



You can create a **Rectangle by** gi,·ing it a **point** and a **Size** (or **width** and **height).** Once **you've** got it, you can find its I.oundaries and **check** its ConlainsO **method** to **see** if it contains a Poinl.

Sharpen your pencil

1. Most of your work with Graphics will involve thinking about your forms as a grid of X. Y coordinates, Here's the code to build the grid shown below; your job is to fill in the missing parts.

```
using (Graphics g = this.CreateGraphics())
using (Font f = new Font("Arial", 6, FontStyle.Regular))
  for (int x = 0; x < this.Width; x +- 20)



  for (int y = 0; y < rhis.Height; y +- 20)
```



2. Can you figure out what happens when you run the code below? Draw the output onto the form, using the grid you just rendered for locating specific points.

```
using (Pen pen •
         new PenIBrushes.Black, 3.on)
   g.DrawCurvelpen, new Point[J I
      new Point 180, 60),
      new Point 1200,40),
      new Point 1180, 60),
      new Point 1300,40),
   });
   g.DrawCurve lpen, new Point[J

      new Point I200,lBO), new Point(BO, 200),
   ));
   g.DrawLine lpen, 300, 40, 300, 1BO);
   g.DrawLinelpen, BO, 60, BO, 200);
   g.DrawEllipse(pen, 40, 40, 20, 20);
   g.DrawRectangle(pen, 40, 60, 20, 300);
   qcDrawl.Lnetpen, 60, 60, BO, 60);
   g.Dra"Linelpen, 60, 200, BO, 200);
```



FillPolygon(), DrawLines(), and a few other graphics methods have a constructor that takes an array of Points that define the vertices of a series of connected lines.

3. Here's some more graphics code, dealing with irregular shapes. Figure out what's drawn using the grid we've given you below.

```
g.fillPolygon(Brushes.Black, new Point[J (
        new PointI60,40), new Point(140,80), new Point(200,40),
        new Point{300,BO), new Point(380,60), new Point(340,140),
        nev Point(320,lBO), new Point{3BO,240), new Point(320,300),
        new Point(340,340), new Point(240,320), new Point(180,340),
        new Point(20,320), new Point(60, 280), new Point(l00, 240),
        new Point(40, 220), new Point(BO,160),
    ));

using (font big - new font("Times New Roman", 24, fontStyle.Italic)
    g.DrawString("Pow!", big, Brushes.White, new Point(80, 80));
    g.DrawString("Pow!", big, Brushes.White, new Point(120, 120));
    g.DrawString("Pow!", big, Brushes.White, new Point(160, 160));
    g.DrawString("Pow!", big, Brushes.White, new Point(200, 200));
    g.DrawString("Pow!", big, Brushes.White, new Point(240, 240));
```





Sharpen your pencil Solution

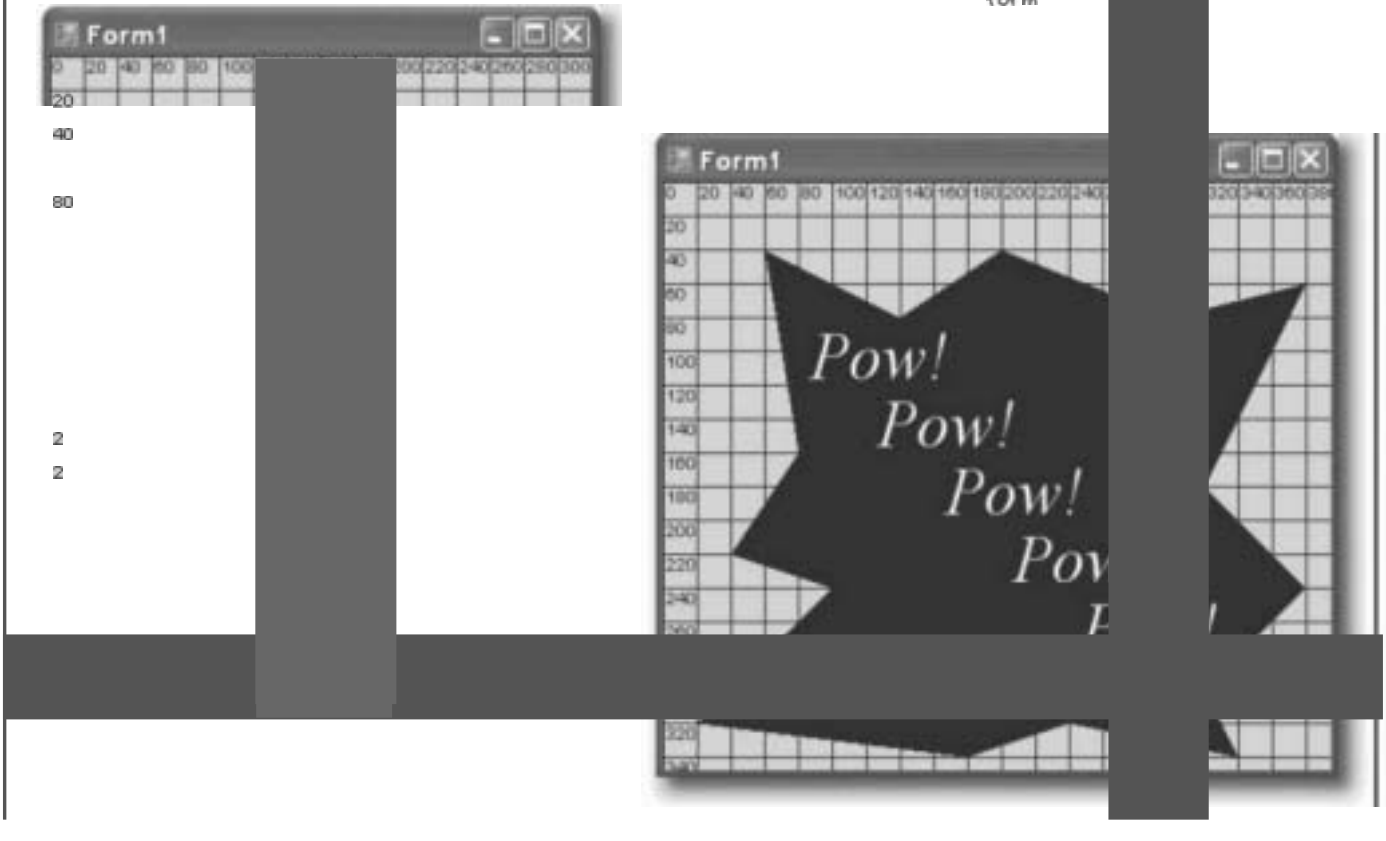
Your job was to fill in the missing code to draw a grid and plot two chunks of code on the grids.

```
using (Graphics g = this.CreateGraphics())
using (Font f = new Font("Arial", 6, FontStyle.Regular))
{
    for (int x = 0; x < this.Width; x += 20)
    {
        g.DrawLine(Pens.Black, x, 0, x, this.Height);
        g.DrawString(x.ToString(), f, Brushes.Black, x, 0);
    }
    for (int y = 0; y < this.Height; y += 20)
    {
        g.DrawLine(Pens.Black, 0, y, this.Width, y);
        g.DrawString(y.ToString(), f, Brushes.Black, 0, y);
    }
}
```

First we draw the vertical lines and the numbers along the y axis. There's a vertical line every 20 pixels along the X axis

We wrapped using statements to make sure the Graphics and Font object get disposed after the form is drawn

Next we draw the horizontal lines and X axis numbers. To draw a horizontal line, you choose a Y value and draw a line from *(0, y)* on the left of the form to *(0, this.Width)* on the right-hand side of the form



Graphics can fix our transparency problem...

y,"'\'e probably figured out by now that Drawfmaqe () is the key 10fixing the problem in the renderer where the images were drawing those boxes around the bees and flowers Ihal caused Ilie overlap issues. So let's tackle them! (;0 back t(your Windows application with the picture.



The renderer d.ew the bees so that they looked weird when they overlappe



O Add a nrevaee () method 1l'"1 draws a bee 0" any Graphics object. It uses Ih,- overl('""I,'d `DrawImage ()` c onstructor Ihat takes a Rectangle I" d"h', m;",' where to draw the image, ami how big I" draw ii,

```
pUblic void DrawBee(Graphics g, Rectangle rect) I
  g.DrawImage(Properties.Resources.Bee_animation_1, rect);
```

Much better—click on the for.. and the bees overlap just fine



8 Here's the new Cl iclt ..v..nt **handl..r for th.. form**. Take " close look at h"" ;t works—it draws the hive '" that its uppe left-hand corne i, way "IT the I;,rlll. ,ll location (-Width, -Height). and il draws il at twice the width and height of the 1;,rl11 - 'o vou can resize the fonu and it'll ,Iill draw okay. Then il draw four bees using lh.· `DrawBee()` method.

First we'll draw the hive backround, with its t:orner fa.. off the page so we only see a », all piece of it. Then we'll draw four bees so that they overlap—if they don't, ...ake your for.. bigger and then click it again so they do

```
private void Forml_Click(object sender, gvenrArcs e) {
    using (Graphics g - CreateGraphics I»)
        g.Drawlmagel Propertles.Resources.Hlve inside ,
            -Width, -Height, Width * 2, Heiqht' 2);
    Size size - new Size(Width / 5, Height / 5);
    DrawEee(g, new Rectangle(
            new Point(Width | ' - 50, Height | Z - 40), size));
    DrawBeelg, new Rectangle(
            new Point(Width | 2 - 20, Height | Z - 60), sLrej);
    crevaee(g, new Rectangle(
            new Point(Width | Z - eO. Height | Z - 30), size));
    DrawBee(g, new Rectangle(
            new Point(Width | Z - 90, Height | Z - 80), size));
```

", but there's a catch

But look what happens if you drag it off the side of the screen and back! Oh no!



O Run your program and click 0" rh.. form, and watch il draw the bees! But something' wrong. \\'h"n you drag Ihe Ii'rill "Jf Ih,' side "r Ih,- screen and back again, **the** pictu.... **disappears!** Now go back and check the "Nectar here" program you wrote a few page ago— it'" **got** th.. **same** probl""l!

What do you think happened?

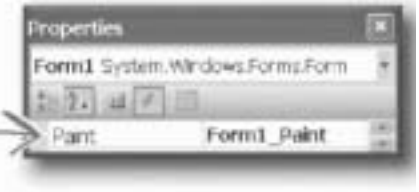
Use the Paint event to make your graphics stick

What good are graphics if they disappear from your form as soon as part of your form gets covered up? They're no good at all. Luckily, there's an easy way to make sure your graphics stay on your form: just write a Paint event handler. Your form fires a Paint event every time it needs to redraw itself, like when it's dragged off the screen. One of the properties of its PaintEventArgs parameter is a Graphics object called Graphics, and anything you draw with it will "stick".

Forms and controls have a Paint event that gives you a Graphics object. Anything you draw on it is repainted automatically.

1 Add a Paint event handler

Double-click on "Paint" in the Events page in the Properties window to add a Paint event handler. The Paint event is fired any time the image on your form gets "dirty". So drawing your graphics in there will make your image will stick around.



Double-click on Paint to add a Paint event handler. Its PaintEventArgs has a property called Graphics—and anything you draw with it will stick to your form.

2 Use the Graphics object from the Paint event's EventArgs

Instead of starting with a using statement, make your Paint event handler start like this:

```
private void UserControl1_Paint(object sender, PaintEventArgs e) {
    Graphics g = e.Graphics;
```

You don't have to use a `using` statement since you didn't create it, so you don't have to dispose it.

3 Copy the code that draws the overlapping bees and hive

Add the new DrawBees() method from the previous page to your new user control. Then copy the code from the Paint event into your new Paint event—**except for the first line with the using statement**, since you already have a Graphics object called g. (Since you don't have the using statement any more, make sure you take out its closing curly bracket.) Now run your program. The **graphics stick!**

Do the same with your "Nectar here" drawing to make it stick, too

Forms and controls redraw themselves all the time

It may not look like it, but your forms have to redraw themselves all the time. Any time you have controls on a form they're displaying graphics—labels display text, buttons display a picture of a button, checkboxes draw a little box with an X in it. You work with them as controls that you drag around, but each control actually draws its own image. Any time you drag a form off the screen and then drag it back or uncover it, the part of the form that was covered up is invalid, which means that it no longer shows the image that it's supposed to. That's when .NET sends a message to the form telling it to redraw itself. The form fires off a Paint event any time it's "dirty" and needs to be redrawn. If you ever want your form or user control to redraw itself, you can tell .NET to make it "dirty" by calling its Invalidate() method.

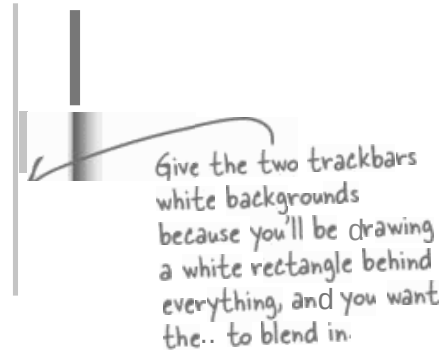


See if you can combine your knowledge 01 forms and user controls – and get a little more practice using `eit map` objects and and the `Drawlmage {)` method--by building a user control that uses TrackBars 10 zoom an image In and out

O Add two TrockB(lr controls to a new user control

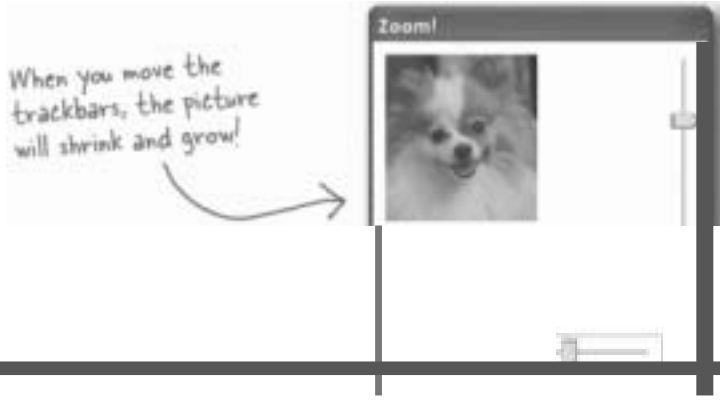
Cr..au · a new Windows .\pplicali"" project. Add a **Us**..r `Control` call il Zoomer and set its `Size` property to (300I.300). Jrag two TrackBar controls •Illt of the toolbox amI .nto it. Drag `trackBarl` to Ihe hottotll "f the control. Then drag `trackBar2` to the right-hand side of the control and set its Or`ientation` property t" `Horizontal`. Both should have the `Minimum` property set to 1, `Maximum` set to 175, `Value` se-r to 175, and `TickStyle` set to `None`. Set each TrackBar's background co,lor 10white. Finally, double-click on ..ach trackbar t" add a `So::r011` event handler. Make bo,th ,·,-,'nt handlers call th.. control's `Inve1idate()` method.

You user **control has** a Paint event, and It **works just** like the one **you** Just **used** In the form. Just **use** Its `PaintEventArgs` parameter **e.** It **has** a **property** called `Graphics`, and anything that **you** draw with that Graphics object will be pa inted onto **any instance** of the user control **you** drag out of the Toolbox.



O Load a picture into ,(Bitm(lp **object and draw it on the** control

Add a private `Bitmap` field called `photo` t" your Zoomer user ,",,"trol. When you create the instance of `Bitmap`. use its constructo,r t" I,)ad yomr favo,rinc image file - we used a picture "f a fluffy log. Then add a `Paint` event t" the contro,l. Th,' ,." „" „! handler should create a graphics object t" draw "" tl,,· control, draw a white filled rectangle ,,,er th,· ,'ntin' control, "",I th,'n use `Drawlmage()` t" draw th,' ,·t)hl<'nt' or your `photo` field "" In your contro,l so its upper ["Ii-hand corner is at (10, 10), its width is `trackaarL.Value`. and its height is `t.rackgarz.Value`. Then drag your control "',tl" Ihe form—make sure It, rr--ize th,' form so the trackbars are at the edges



Whenever **the** ueer serena **one** of the TrackBars, **they** call the user control's InvalidateO method. That **will** cause the **user control** to fire **its** Paint event and resize the photo. **Remember, since you** didn't create the **Graphics object—it was passed** to **you** in **`PaintEventArgs`—you** don't need to dispose it. So **you** don't **have** to **use** a **`using`** statement with It. Just draw the Image Insi de the Paint event handler.



Get a little more practice using `Bitmap` objects and and the `DrawImage()` method by building a form that uses them to load a picture from a file and zoom it in and out

This particular Bitmap constructor loads its picture from a file. It's got other overloaded constructors, including one that lets you specify a width and height—that one creates an empty bitmap.

```
public partial class Zoomer : UserControl {

  Bitmap photo = new Bitmap(@"c:\GraphicS\flUffy_dog.jP9");

  public Zoomer() {
    InitializeComponent();
  }

  private void zoomer_Paint(Object sender, PaintEventArgs e) {
    Graphics g = e.Graphics;
    g.FillRectangle(Brushes.White, 0, 0, Width, Height);
    g.DrawImage(photo, 1a, 1a, trackBar1.Value, trackBar2.Value);
  }
}

  private void trackBar1_Scroll(object sender, EventArgs e) {
    Invalidate();
  }
  private void trackBar2_SCrOll(Object sender, EventArqs e) {
    Invalidate();
  }
```

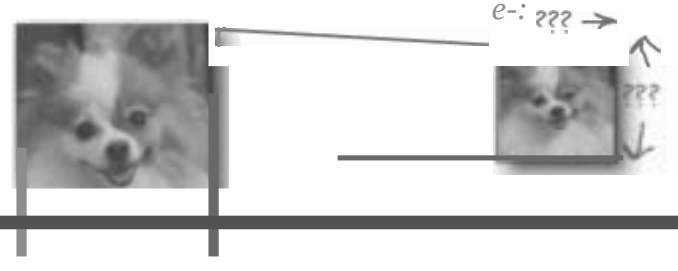
First we draw a big white rectangle so it fills up the whole control, then we draw the photo on top of it. The last two parameters determine the size of the image being drawn—trackBar1 sets the width trackBar2 sets the height.

Every time the user slide, one of the trackbar controls, it fires off a Scroll event. By making the event handlers call the control's `Invalidate()` method, we cause the form to repaint itself... and when it does, it draws a new copy of the image with a different size.



Each drag keeps causing another image resize from DrawImage()

```
g.DrawImage(myBitmap, 30, 30, 150, 150);
```



Acloser look at how forllls al1d cOl1trols repail1t thelllseives



Remember earlier, we said lh"1 when you start working with (Jraphics objects, you're re"llly taking control of graphics It's lil,' you tell .NET, "Hey, I lll"W what I'm doing, I can ha",II,' the ,'xtra resp »nsibility." [" the case ,,1' drawing and redrawing, y(" may not want t" redraw when a form i, uunhuizcd and maximia-d.; "" you may want to redraw ***mor.. often*** · Once vou knuw what's going on behiud the scenes wilh your form "r c'Ulrol. you can take control „1' redrawing yourself:

o **Every** form has a Paint **event** that draws **the** graphics on the form

(i" 10the event list for any form and lind the event called `Paint` . Whenever Ih,' form has to repaint it,dL lhi, event i, fired. Every form and contrc,luses ;, `Paint` event internally to decide wh,'n I" redraw itself. But what fires lh,'l event? It's c:,ll"d hv a method ";,ll,'d `OnPaint` that the form or user contrc,j inherits fi "n the Control class. (That method follows ll"'palh',-n you saw iu Chapt,'r II, where ""'lhod, thaI fire an event ar.' named "On" followed by the event name.) Go h, any form and override (lnl',';nt:

Do this just like you did ea..lie.. with Dispose()

Override OnPaint on any form and add this line

```
protected override void OnPaint(PaintEventArgs e) {
  )console.WriteLine("Onpaint {OJ IlJ", DateTime.Now, e.ClipRectanglel;
    base.OnPaint(e);
```

Drag yc"u' fc'I'm around —drag il halfway off the screen, Ul inimize- it. hide il behind other windows. Watch closely at the »urpur that it writes You'll see that your (1"1'aint me-thod fires off a Paint event anv time part of it i, "dirty"—or **invalid** - a",1needs 1> be redrawn. .\",1 if you look closelv at the ClipRectangle-. you'll see that it's a rectangle that describes the part of lhe form that "",'d, 1" be repainted. ThaI gets passed to the Paint event's `PaintEventArgs` so it can imprtve performance by onlv redrawing the portii,," that's invalid.

e InvalidateO controls **when to** redraw ,,, and WHAT **to redraw**

.NET fires lh., Paint event when something ,'"' form i, interfered with, ,·,,,·.,rt'd up, or moved offscree", a"d then shown again. [I calls `Invalidate()`. and passe lh,' method a `Rectangle`. The `Rectangle` t€n, the ,Tnva,lidate O method what part of thrfium needs to be redrawn... "h:,t part o[the form is "dirty." Then .NET calls OnPaint t(tell your form 10 fire a `paint`; event and repaint lh,' dirty area

Invalidate() essentially says, some part of the.. form ··ight be "invalid, so redraw that part to make sure it's got the right things showing

So when you call it yourself, you're telling .NET that you" whole form or control is invalid, and the whole thing needs to be redrawn. You can pass it your own clip rectangle if I'- want—that'll get passed along to the Paint event's PaintEventArgs

o The UpdateO **method gives** your **Invalidate request top** prior ity

Y,,,, may "ot realize il,lnn your form is getting messages all the time. The same system thaI tells it that it's been covered up and calls OnPaint has all sorts or other messages il uee·d, I" ,,·ud. See for yourself: type `override` and "'" ,ll through all the methods that start with "On"—every 0"'" of them i, a message your fc'nn H"po"d, I", 'T'll,' `Update` (I 1lH'11",,J Ill"'"" the Invalidate message 10the lop"f rhe message list.

o **The** form's RefreshO **method** is **Invalidate()** plus UpdateO

Forms and controls give you a ,honn ,l, They have a Refresh() lll"II"Kllh"t fir'l ,'"ll, Invalidate() tt> invalidate the whole di"1l1 area, and tt"," calls Update() tu make sure thaI message moves to the 1"1',,1' the li,t.

there are no Dumb Questions

Q: It still seems like ju,l resizing the graphics in a program lik, Paint or PholoShop would be better. Why can't I do that?

A: You can, if you're in control 0/ the images you work with III your applications and if they'll always stay the same size But that's not often the case Lots of times, you1 get images from another source, whether it's online or I co-worker in the design group Or you may be pulling an image from a read-only source, and you have 10 size it programmatically

Q: But III can resize i1 outside of .NET, that's bett. .. right?

A: If you're sure you'll never need I larger size, it could be Bul" your program might need 10 display the image in multiple sizes during the program, you'll have 10 resize at some point anyway Plus, if your ever needs 10 be displayed larger than resize you'll end up in real trouble It's much easier to size down than it is to size up

More often than not, it's better lo be able lo resize an image programmatically, than to be limited by an external program or constraints like read-only files

Q: I get that CreateGraphics() gets the Graphics object for drawing on I Io,m. but what was that FromImage() call in the ResizeImage() meIhod about?

A: `FromImage()` retrieves the `Graphics` object for I `Bitmap` object And just as `creeteGraph1cs()` called on a form returns the `GraphIca` object for drawing on that form, `FromImage()` retrieves a `Graphics` object for drawing on the `8itmap` the method was called on

Q: So. Graphics object isn't just for dr.winll on. form?

A: Actually, a Graphics object is for drawing on. well, anything that gives you a Graphics object. The Bitmap gives you a Graphics object that you can use to draw onto an invisible image that you can use later. And you'll find Graphics objects ona lol more than forms Drag a button onto I form, then go into your code and type its name followed by a period Check out the IntelliSense window that popped up—it s got CreateGraphics() method that returns I Graphics object. Anything you draw on it will show up on the button! Same goes for Label, PictureBox, StatusStrip .. almost every toolbox control has a Graphics object

Q: Wait, I thought `using` was ju' l something I used with stream·. Why am I using `u.inq` with graphics?

A: The `using` keyword comes in handy with streams, but it's something that you use with *any* class that implements the `IDisposable` interface When a class implements `IDisposable`, any time you instantiate it you should call its `D1epoee()` method With streams. the `Mepcse(I` method makes sure that any file Ihal was opened gets closed `Graphics.Pen.` and `Brush` objects are also disposable When you `ceeie` any of them, they take up some small amount 01 memory and other resources, and they don't always give them back immediately If you're just drawing something once, you won't neeee a difference. But most of the time. your graphics code will be called over and over and over again—like in a `Paint` event handler, which could get called many times a second for a particularly busy form. That's why you should always `Dispose()` of your graphics-related objects And the easiest way to make sure that you do e e use a `using` line, and let NET worry about disposal Any object you create with `using` will automatically have its `Dispose()` method called lIIthe end 0/ the block following the `using` statement That will guarantee that your program won't slowly take up more and more memory if it runs for I long time

Q: I'm creating a new control, should I USII UserControl or should I create · class that inherits from one of the toolbox controls ;

A: That depends on what you want your new control to do If you're building a control that's really similar lo one that's already in the toolbox, then you'll probably find it easiest to inherit from that control But most of the time when programmers create new controls III C# they use user controls One useful advantage 0/ a user control is that you can drag toolbox controls onto it · works 11o/like a `GroupBox` or other container control you can drag a button or checkbox onto your user control, and work with them just e you'd work with controls on a form The IDE s form designer becomes a powerful loollo help you design user controls Youl see more about that in just I minute

A user control can hosl oll,er conlrols. The IDE', 101'm designer lets you drag controls oul 01 the looll.ox an,1 01110 Jour new user control.

I noticed a whole lot of flickering in my Zoomer ee ereel. With all this tcfk of taking control of graphics, I'll bet there's something we can do about that! But why does it happen?

o o



Even without resizing, it takes time to draw an image to a form.

Suppose you've got everv imag(in the simulator resized. It still takes tim" to draw all those bee and flowers and the hive. ,\"d right now we're drawing right 10 the Graphics ohj" "1 o" the form. So if your eye catches the tail end of a render. you'r.. going to p-rceivc il as a little flicker.

The problem is lhat a 1"1 Of drawing i, happening, so there's a good chance that some flickering will occur, "\-"]] with our resizing :\",1 that's why you nm i"to problems with most amateur computer games... the huma" eve catches the end of a rendering cycle, and perceive il as a little hil "I llickering on the screen.

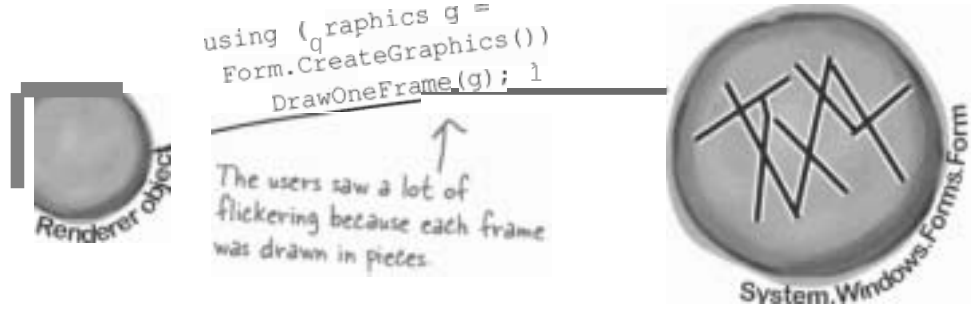


How could you get rid of this flicker? If drawing lots of images to the form causes flickering, and you have to draw lots of images. how do you think you might be able to avoid all the flickering?

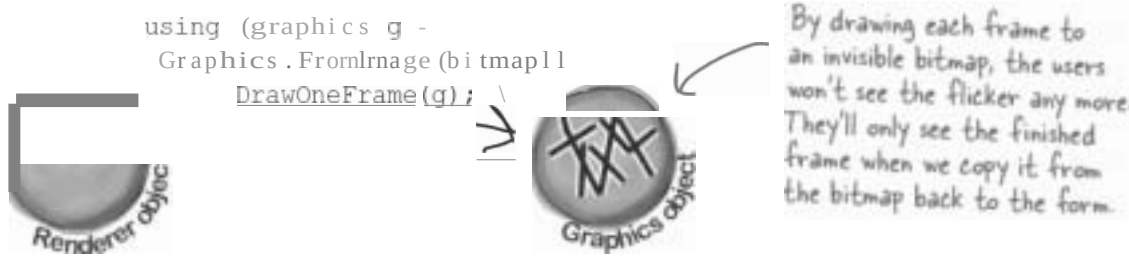
Double bufferil1g llIakes al1illlatioll look a lot sllIoother

(;" back 10 y(>lll' imag(zoomer "Ild fiddle with the Irackbars. Noti,'e how there's a whole lot of flickering when you move the bars? That's because the `Paint` event handler first has t" draw the "hil" rectangle and then draw II", imag(every lime the trackbar [[[0ves a tiny lillle hit. When y)[[1' eves see alternating white rectangles and images many times a second, they interpret that as flicker. It's irritating... and it's avoidable using a technique called **doub!.. buff..ring**. That means drawing each frame or cell "f animation 10 an invisible bitmap (a "buffer"), and only displaying Ihe new frame once it's been drawn entirely. Here's how il would work with a `Bitmap`:

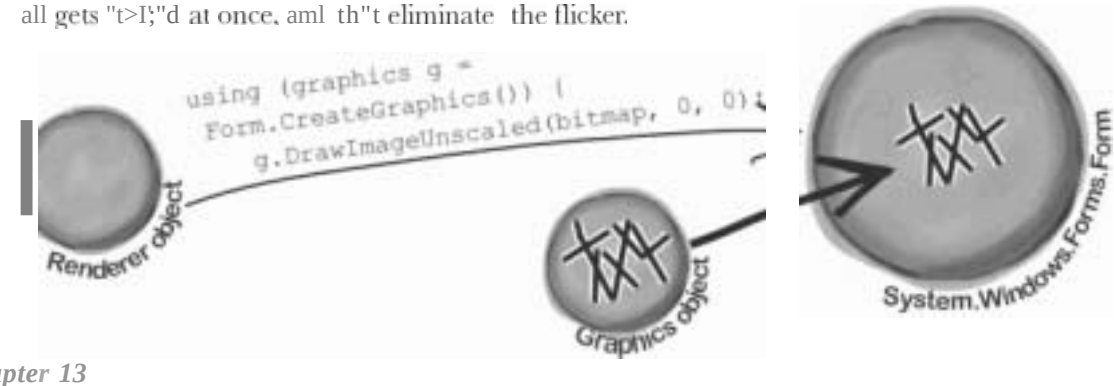
1 Here's a typical program that draws some graphics on a form using its Graphics object.



O To do double buffering, we can add a `Bitmap` object 10 the progrnm tu act as a buffer. Every time "Ur form or control needs 10 be repainted, instead "f drawing the graphics din-'-lly ,," the form, we draw on the buffer instead.



3 Now that the frame is completely drawn "UI It> the illl'i,itole `Bitmap` object, we can use `DrawImageUnscaled()` It, copy the object back t" Ih,' form's `Graphics`. II all gets "t>I;"d at once, aml th"t eliminate the flicker.



Double bufferil1g is built il1to forms al1d eontrels

y,,,, can do double buffering yourself using a Bit map, bIll C# and .NET make ;1 even easier with I,uih·in support for dOllhk buffering. All **you need to** do **is set its** Douhl..BuJTered prop""I)' **to** Iru... Try il "Ul oil your Zoomer user control—go '" i" Propertie window, set DoubleBuffered to (n"', ;",,1 your control will stop flickering! Now **go** back to)·ou I" B....**Control** "",I do the sam... That won't fix all of the graphics problems we'll d" lha! in a ",inul,- Inn it *will* make a difference.

Now you're ready to fix the graphics problems in the simulutor!

Wh.n you use **the** Painl event **for** all your graphic,. you can turn on **double** I,uffered pain ling 'imply **by** c1.anging on. prop.,·ly.

Overhaul the Beehive Simulator

In the next exercise, you'll take your Beehive Simulator and completely ""'rha,,l it. You"ll probably want to create a whole new projec, and use "Add > Existing Item..." I" add lhe current files I" il so you have a backup "f your curre111 simulator. ;1),n't forget I" change lh,.ir ",1111"!""-" I" match your 111"" project.)

Here's what you're going III do:

- You'll **start** by rem oving **the BeeControl** user control
  Th,.n' won't h., any controls on Ih,. hi", and field at all. No BeeControls, no l'i"'llreBoxes, nothing. The hees, flowers, and hive pictures will aU be drawn using (;\)1+ graphics. S" right-click un BeeControl.cs (and ()ldBeeControl.cs) in the Sc,lut;,>ll Explorer and click Delete - they 1\ be removed li'llm the project and permanently deleted.

- Yau'll **need** a **timer** ta **handle the bee wing** flapping
  The bee flap their wings mur-h mon° slowly than the simulator's frame rate, so you'H need a second, slower timer. Th;, ,hou[dn'\ be I'u surprising, since Ih.· BeeControl had its own limIT I" d" the same thing.

- **The big step: overhaul the renderer**
  You'll ,ıeed 10 throw out the current renderer entirely, because it does everything with controls. You won't need those lookup dictionaries. because Ihere won't be any PictureBoxe or BeeControls to look "I'. Instead, ;1'1\ have two ;llll'ortalllllll,-Ihod" DrawHive(g) will draw a Hive form on a graphics object, and DrawField(g) will draw a Field form

- Lost of **all,** you'll **hook** up **the new renderer**
  The Hive and Field forms will need Paint event handlers. Each of them will "aU the Renderer object's DrawField(g) "I' DrawHive(g) methods. The two timers—one fe" telling Ih,- simul.uor I" draw lh,' next frame, and the "I her I" flap lht" bees' wings— will call the tw >fi,rm,' Invalidate() methods I" repaint rhemselves. When they ,I", lIeir Paint event handlers will remltT the frame.

Let's get started!



Irs time to get rid of the graphics glitches in the beehive simulator Use graphics and double buffering to make the simulator look polished

o **Change the** main form's **RunFrame() method**
You'll need to remove the call to `Renderer.Render()` and add two `Invalidate()` statements.

```
public void RunFrame(object sender, EventArgs e) {
    framesRun++;
    world.Go(random);
    end = DateTime.Now;
    TimeSpan frameDuration = end - start;
    start = end;
    UpdateStats(frameDuration);
    hiveForm.Invalidate();
    fieldForm.Invalidate();
}
```

You'll need to remove the call to renderer.Render(), since that method will go away

As long as you keep the world up to date and both forms have a reference to the renderer object, all you need to do to animate them is call their `Invalidate()` methods. Their `Paint` event handlers will take care of the rest.

o Add a **second timer** to **the main** form to **make the bees'** wings flap
Drag a new timer onto the main form, set its `Interval` to 150ms and `Enabled` to true. Then double-click on it and add this event handler:

```
private void timer2_Tick(object sender, EventArgs e) {
    renderer.AnimateBees();
}
```

Then add this `AnimateBees()` method to the renderer to make the bees' wings flap

```
private int Cell = 0;
private int Frame = 0;
private void AnimateBees() {
    Frame++;
    if (Frame >= 6)
    Frame = 0;
    switch (Frame) {
        case 0: Cell = 0; break;
        case 1: Cell = 1; break;
        case 2: Cell = 2; break;
        case 3: Cell = 3; break;
        case 4: Cell = 2; break;
        case 5: Cell = 1; break;
        default: Cell = 0; break;
    }
    hiveForm.Invalidate();
    fieldForm.Invalidate();
}
```

The whole idea here is to set a field called Cell that you can use when you're drawing the bee in the renderer. Make sure you're always drawing BeeAnimationLarge[Cell] in the hive form and BeeAnimationSmall[Cell] in the field for. The timer will constantly call the AnimateBees() method, which will cause the Cell field to keep changing, which will cause your bees to flap their wings

Set up the hive *and* field forms for double-buffered animation

R"Ill'ove the code fro1in the hive fo,rm" constructor that sets IhI' background image. Then remov',. all controls from holh forms and s..t th..ir `DoubleBuffered` properties to `true`. Finally, add a Paint event handler '0 each of them, Here's the handler for the hive fou-m -, the field fo'rm}'s Paint eve-nt handler i, identical, except that it calls `renderer.PaintField()` instead "I' `renderer.PaintHive(}`:

```
private void HiveForm_Paint(object sender, Paint.Event.Arqs e} {
    renderer.PaintHive(e.Graphics);
}
```

Make sure r- turn on double buffering, or your forms will flicker!

The hive form and field form both need a public `renderer` property

Add a public field to the hive form and the field [urm:

```
pUblic Renderer renderer;
```

There are two places where you create ; `new Renderer()`: in !he open button (underneath a call '0 renderer.Reset() ;llld illl!te ResetSimulator() method. Remove all calls to renderer.Reset(), and replace each of the new `renderer` statements with a call 1" this new CreateRenderer() method Ihal you'll add:

```
private void CreilteRendererO
    renderer = new Renderer IvorLd, hiveForm, fieldFonn),
    hiveForrn.renderer _ renderer;
    fieldForrn.renderer - renderer;
```

You won't need a Reset() method in the .enderer any more. All it did was remove the controls from the I orms) and there won't be any controls to ..emove

Overhaul the r-enderer- by removing ccetrcl-bcsed code and odding graphics

H,'re's what vou need I" do I" fix the renderer:

- R"m,ve the two dictionaries, since there ;"l''''t any more controls. .And while you're at it, yc,n don't need the `BeeControl` any more, or Ih" `Render()`. `uTevaees()` .or `DrawfLovers 0` methods..
- •Add some `Bitmap` fields called `Hivejnsace`. `Hiveouts rce`, and `Flower 10`store the images. Then create two `Bitmap` [1 arrays calk-d `BeeAnimationLarge` and `Beean.irnetionSmall`. Each of them will hold four bee pictures—the large ones are 40x40 and Ih,' small are 20x20. Create a method called `InitializeImages()` to resize II,,' resources and store them ill tl,, sc· fields, ;llul call i, from the `Renderer` class constructor.
- .Add ,I,,' `PaintHiVe()` method lhal takes a `Graphics` object as a parameter and paint the hive form 011'0 il. First draw a sky blue rectangle, then use `DrawImaqeunscaled()` to draw the inside hive picture, then use `Drav'ImaqeunacaLed()` III draw each of the bees that are inside the hive.
- Finally, add the `PaintField()` method. II should draw a sky blue rectangle on the top half "f rh.. form, and a green rectangle on thr- bottom half. You'll find two l;,r111 properties helpful for thi,: ClientSize and ClientRectangle tell you how big the drawing area is so you """ lind half of its height using `ClientSize.Height / 2`. Then use `FillEllipse()` to draw a yellow ,un in the sky, `DrawLine()` to draw a ,hick lin" for a hrand, lhe hive ,'an hang from and `DrawImageUnsceIed()` !Il draw the outside hive picture. Then draw each flower 01\10 the form. Finally, draw each bee (using lhe small bee pictures)—draw them last so they're in fnun of the flowers.
- When you n- drawing the bees, remember that `AnimateSees()` sets the `Cell` field.



Irs time to get rid of the graphics glitches in the beehive simulator Use graphics and double buffering to make the simulator look polished

Here's the complete Renderer class, including the AnimateBees() method that we gave you. Make sure you make all the modifications to the three forms—especially the Paint event handlers in the hive and field forms. Those event handlers call the renderer's PaintHive() and PaintField() methods, which do all of the animation.

```
using System.Drawing;

public class Renderer {
    private World world;
    private HiveForm hiveForm;
    private FieldForm fieldForm;

    public Renderer(World TheWorld, HiveForm hiveForm, FieldForm fieldForm)
        this.world - TheWorld;
        this.hiveForm • hiveForm;
        this.fieldForm - fieldForm;
        InitializeImages();


    public static Bitmap ResizeImage(Image ImageToResize, int Width, int Height) I
        Bitmap bitmap . new Bitmap(Width, Height);
        using (Graphics graphics - Graphics.FromImage(bitmap)) {
            graphics.DrawImage(ImageToResize, 0, 0, Width, Height);

        return bitmap;


    8itmap HiveInside;
    Bitmap HiveOutside;
    Bitmap Flower;
    BitmaplJ BeeAnimationSmall;
    Bitmap[J BeeAnimationLarge;
    private void InitializeImag...()
        HlveOutslde - Reslzelmage(Propertles.Resources.Hlve  outside , 85, 100);
        Flower · ResizeImage(Properties.Resources.Flower, 75, 75);
        HlvelnsIde - ReslzeImage IPropertles.Resources.Hlve  inside_,
            hiveForm.clientRectangie.Width, hiveForm.ClientRectangle.Height);
        BeeAnimationLarge - new Bitmap[4j;
        BeeAnimationLargelOJ    ResizeImage(Properties.Resources.Bee_animation_l, 40, 40);
        BeeAnimationLarge[1]    ReslzeImage(Propertles.Resources.Bee anlmatlon 2, 40, 40);
        BeeAnimationLargel2J    ResizeImage(Properties.Resources.Bee_animation_3, 40, 40);
        BeeAnimationLarge[3]    ReslzeImage(Propertles.Resources.Bee_an1matlon 4, 40, 40);
        BeeAnimationSmall • new Bitmap[4];
        BeeAnimationSmall[O]    ResizeImage(Properties.Resources.Bee_animation 1, 20, 20);
        BeeAnimationSmall[1J    PesLzeImaqe(Properties.Resources.Bee_animation_2, 20, 20);
        BeeAnimationSmall[2]    ReslzeImage(Propert1es.Resources.Bee_anlmatlon 3, 20, 20);
        BeeAnimationSmall[3]    ResizeImage(Properties.Resources.Bee_animation_4, 20, 20);
```

The InitializeI..ages() method resize• all of the image -esources and stores them in Bitmap fields "..side the Renderer object. That way the PaintHive() and PaintForm() methods can d..aw the images unscaled using the f... Graphics objects' DrawlmageUnscaled() ethods.

```
public void PaintHiv..lGraphics gl {
    g.FiIIRectanqlelBrushes,Sley8Iue. hiveForm.ClientRectanglel;
    g.DrawImageUnscaled(HiveInside, O. 0);
    roreach (Bee bee In world.Bees) (
        it (bee.InsideHive)
            g.DrawImageUnscaled(BeeAnimationLarge[Cell],
                                bee.Location.X, bee.Location.Y);
```

A form's ClientSize property is a Rectangle that tells you how big its drawing area is.

```
public void PalnlFl.1dlGraphlcs g) /
    using IPen brownPen • new Pen(Color.Brown, 6.0F)) {
        g.FillRectangle(Brushes.SkyBlue, O. 0,
                fieldFor•.ClientSize.Width, fieldForm.ClientSize.Height I 21:
        q.FillEllip"eIBru"he".Yellow. new RectangleF(50, 15, 70, 10)):
        g.FillRectangle(Brushes.Green, 0, fieldForm.ClientSize.Height I 2.
                fieldForm.ClientSize.Width, fieldForm.ClientSize.Height I 2);
        g.DrawLine(brownPen, new Point(643, 0), new Point(643, 30));
        g.DrawImageUnscaled(HiveOutside, 600. 20);
        foreach (Flower flower in world.Flowers) (
            g.DrawImageUnscaled(Flower, flower.Location.X. rrever.Location.Y);
        |
        tareach lBee bee in world.Bees)
            if (!bee.InsideHive)
                g.DrawImageUnscaled(BeeAnimationSmall[CellJ.
                                    bee.Location.X, bee.Location.Y);
```

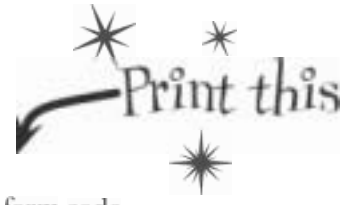
The PaintField() method looks at the bees and flowers in the world and draws a field using their locations. First it draws the sky and the ground, then it draws the sun, and then the beehive. After that, it draws the flowers and the bees. It's important that everything is drawn in the right order—if it were to draw the flowers before the bees, then the bees would look like they were flying behind the flowers.

```
private int cell - 0:
private int Frame - 0:
public void AnimateBees()
|
    Frame++;
    if (Frame >- 6)
        Frame - 0:
    switch (Frame) (
        case 0, cell _ 0, break,
        case 1, Cell. I: break;
        case 2: cell   2: break;
        case 3: Cell   3: break;
        case 4: Cell   2: break;
        ease 5: Cell   1: break;
        default: cell- 0; break;
    |
    hiveForm.InvalidatelI;
    fieldForm.Invalidate();
```

Here's the sa e AnimateBees() method from the exercise. It cycles through the animations using the Fra_e field—first it shows cell 0, then cell 1, then 2, then 3, and then back to 2, then 1 again. That way the bee flapping animation ... ooth

Use a Graphics object alld an evellt handler for prilltillg

The Graphics methods you've been using '0 draw Oil """I' forms are th.. **sam..** on**es** you u.... to print. .NET's printing objects in System.Drawing,Printing make il really easy 10 add printing and print preview In your appli,'ations. :\11 I"" need to do is ..**reat**.. a **PrintDocument object**. It's go an event called PrintPage, which you can use exactly like you use a timer's Tick event. Then call the PrintDocument object's Print() method and il pl'im, the d,ocument. '\lld renu-mber. the II)E makes it especially easy 1" add the '."\·'·U' handk-r. Here's how:



O Start a n..w **Windows** appli..a tion aud add a I"'ll,m '0 the form. (;" to the form code aud add a `using System.Drawing,Printing;` line I" the l"f'. Double-click "11 the I",ll"n and add the event handler. \\'atch what happens as soon as you type `+_`:

```
private void buttonl Click(object sender, EventArgs e)
    PrinrDocument document - new PrintDocument();
    document.Print Page +=
                        new PrintPageEventHandler(document PrintPage);
```

(Press TAB to insert)

O Press Tab and the IDE automaticallv fills in the rest "j" rhe line. This is just likr- how you added event handlers in Chapter II:

```
private void buttonl_Cllck(object sender, EventAr'qs e)
    PrintDocument document - new PrintDocument();
    document.PrintPage += new PrintPageEventHandler(documen    rintPage);
```

Press TAB to de-e-ate handler 'document.PrintPage' in this

3 As soon as you press Tab. the IDE generates an event handler method and adds il1" the form.

```
void document PrintPage(object sender, PrintPageEventArgs e) {
    throw new- NotImplementedException(I;
)
```

Now you can put ANY graphics code here—just replace the throw line and use e.Graphics for all of the drawing. We'll show you how in a ..\nute..

l"he PrintPageEventArgs parameter e has a Graphics property. Just replace the throw statement with code that calls the e.Graphics object's drawing methods.

O Now finish "If lh" buttonl_Click event handler by calling `document.Print()`, When lha' method is called, the PrintDocument. object creates a Graphics object and lh('ll fire off a PrInt Page,"\"11' with the Graphics ohj" r-t as a parameter. Anything thar ,h,' event handler draws 'n,l" the Graphics object will get sent to the printer.

```
private void buttonl Click(object sender, EventArgs e)
    Print Document document . new Print Docurnent();
    document.PrintPage += new PrintPageEventHandler(document PrintPage);
    document.Print();
```

Printl10eullIent works with the print dialog and print preview window objects

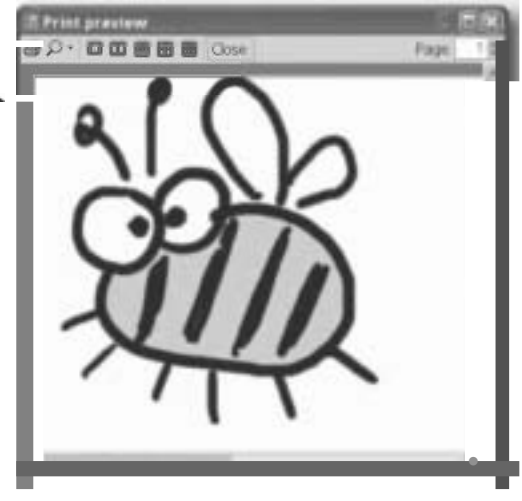
Adding a print preview window or a print dialog box is a lot like adding an open O[save dialog box. All you "ed to d" i, create a PrintDialog or PrintPreviewDialog object, set its Document property 10your Document object, and then call ,I,, dialog's Show() method. The dialog will take care Of sending the document to the !,rinh" llo need I" call its Print {) llWIh"d. So let's add thi- I" the bnll"n you created in step 1:

Once you've got a PrintDocument and an event handler to print the page, you can pop up a print preview window just by creating a new PrintPreviewDialog object.

```
private void buttonl_Click(object sender, EventArgs e) {
    PrintDocument document = new PrintDocument();
    document.PrintPage += new PrintPageEventHandler(doclIment_PrintPage);
    PrintPreviewDialog preview = new PrintPrevlewDialogl);
    preview.Document = document;

    preview.ShowDialog(this);
}

void document_PrintPage(object sender,
                        PrintPageEventArgs e) {
    DrawBee(e.Graphics, new Rectangle(O, 0, 300, 300» ;
}
```



Use e.HasMorePages to print OIulti-pag, documents

If you heed 10 l',i"lIW'n" than one page, all you need 10 do i, have your PrintPage event handle set e.HasMorePages to true. That tells the Document that you've got another page to print. It'll call the event handler over and over again, once p,'r page, a, long a the event handler keeps setting e.HasMorePages 1' true. So modify your IDocument's ,",(lll handler 10 Pl'illl two pages:

```
bool firstPage - true;
void document_PrintPage(object sender, PrintPageEventArgs e) {
    DrawBee(e.Graphics, new Rectangle(O, 0, 300, 300» ;
    using (Font font = new Font("Arial", 36, Font.St.yle.Boldj} {
        if (firstPage)
            e.Graphics.DrawString("First page", Font, Brushes.Black, 0, 0);
            e,Ha"MorePage" = true;
            firstPage - false;
        else {
            e.Graphics.DrawString("Second page", Fonr, Brushes.Black, 0, 0);
            firstPage = true;
```

If you set e.HasMorePages to true, the Document object will call the event handler again to print the next page

Now run your progra again, and make sure it's displaying two pages the print preview



Exercise

Write me code lor Ihe Pnnt button In Ihe simulator so mat II pops up a print preview window showing the bee stats and pictures of ire hive and the field

1 Make the button pop up a print preview window

\dd a ll event handler for the I" tlOll's click ev.." Ihal pauses Ih,- simulat<l_pops up th..]lliill preview dialog, and then resume the simulato Wh'-ll it's d"",-_(If the ,i",ul,tlol is paused when the button is clicked, make sure il stays paused after the preview is shown.)

2 Create the document's PrintPage event handler

II should cr..,",- a page that looks .xactly lik,- Ihe one on the facing page. We'll start you off:

```
private void document_PrintPage(object sender, PrintPageEventl\.rgs e)
    Graphics 9 • e.Graphics;
    Size stringSize;
    using (Font arial24bold • new Font("Arial", 24, FontStyle.Bold))
      stringSize - Size.Ceiling(
          g.MeasureString("Bee Simulator", arial24bold));
      g.FillEllipse(Brushes.Gray,
          new Rectangle(e.MarginBounds.X + 2, e.MarginBounds.Y + 2,
          stringSize.width + 30, stringSize.Height + 30));
      g.FillEllipse(Brushes.Black,
          new Rectangle(e.MarginBounds.X, e.MarginBounds.Y,
          stringSize.Width + JO, stringSize.Height + JO» ;
      g.DraWString("Bee Simulator", arial24bold,
          Btushes.Gray, e.MarginBounds.X + 17, e.MarginBounds.Y + 17);
      g.DrawString r-eee Sjmujator-", arial24bold,
          Btushes.white, e.MarginBounds.X + 15, e.MarginBounds.Y + 15);
,
    int tableX = e.MarginBounds.X + (int)stringSize.Width + 50;
    int tableWidth - e.MarginBounds.X + e.MarginBounds.Width - tableX - 20;
    int firstColumnX _ tableX + 2;
    int eeccncccrceox _ tableX + (tableWidth / 2) + 5;
    int tableY _ e.MarginBounds.Y;
```

We created the oval with text in it using the MeasureString() ...ethod, which returns a Size that contains the size of a string We drew the oval and text twice to give it a shadow effect

y'...'ll need 1, these to build the table

II Your job: fill in the rest of the method to make it print this

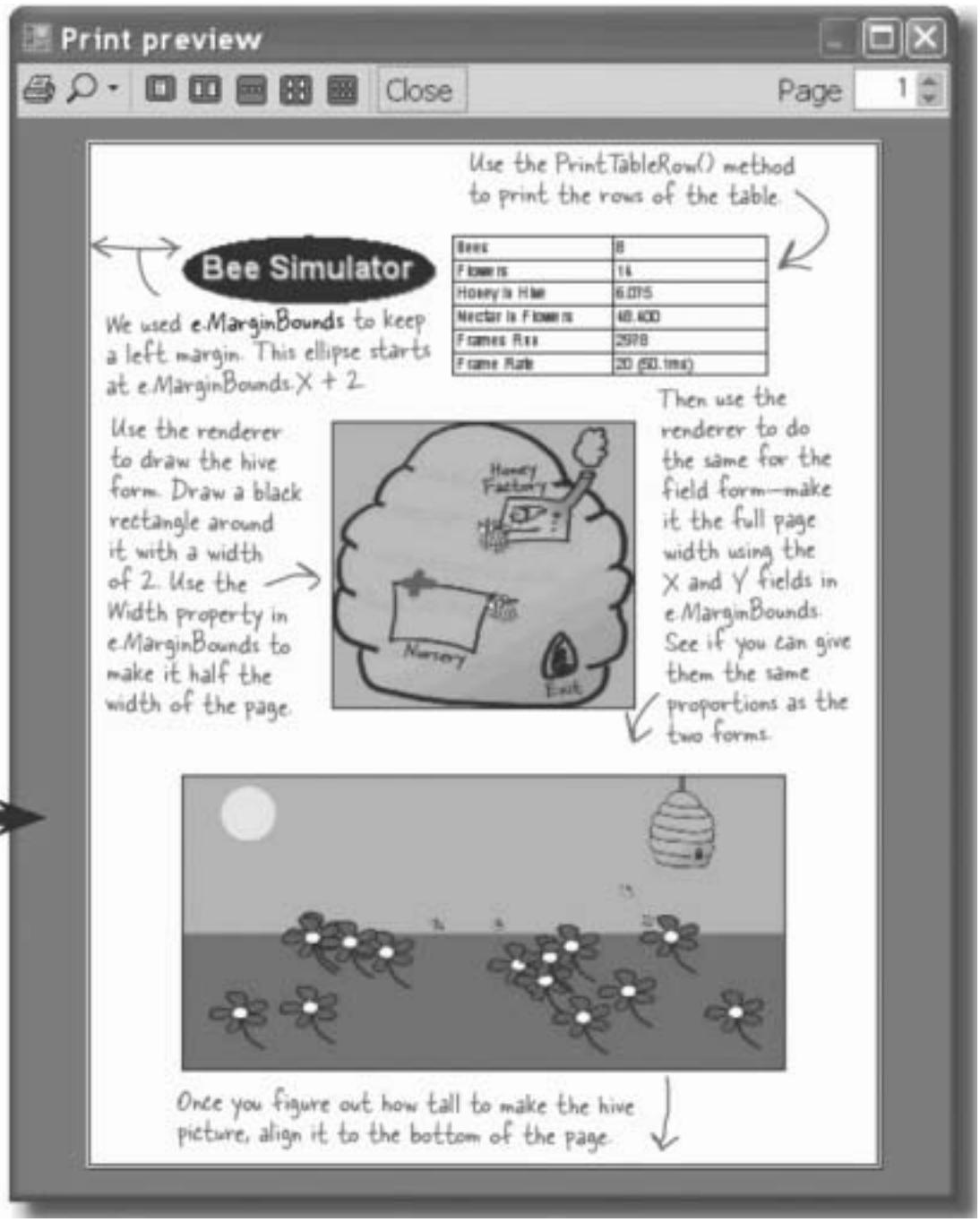
3 This PrintTableRowO method will come in handy

You'll lind this method useful whe-n you create the table of bee stats at the 1"1'of the page

```
private int PrintTableRow(Graphics printGraphics, int tableX,
                int tableWidth, int firstColumnx, int secondColumnX,
                int tableY, string firstColumn, string secondColumn)
  Font arial12 = new Font("Arial", 12);
  Size stringSize - Size.Ceiling(printGraphics.MeasureString(firstColumn, aria1l2));
  tableY += 2;
  printGraphics.DrawString(tLrstColumn, aria112, Brushes.Black,
                firstColumnX, tablen;
  printGraphics.DrawStringtseconocojumn, aria112, Brushes.Black,
                secondColumnX, tableY);
  tableY += (int)stringSize.Height + 2;
  printGraphics.DrawLine IPens.Black, tableX, tableY, tableX + tableWidth, tableY);
  ariaLl2.Dispose();
  return tableY;
```

Each time you call PrintTableRow(), it adds the height of the row it printed to tableY and returns the new value.

T.ke **a close** look . t the notes we wrote on the printout. This is **a** little **complex—take** your timel



Here's a hint: To find the height of each form, find the ratio of its height divided by its width and ultiply that by the final width You can locate the top of the field form by subtracting its height from the bottom margin of the page: (e.MarginBounds.Y + e.MarginBounds.Height – fieldHeight).



Write the code for the Print button in the simulator so that it pops up a print preview window showing the bee stars and pictures of the hive and the field

Here's the event handler for the Document's PrintPage event. It goes in the form.

```
using System.Drawing.Printing;

prlvate vOld document PrlntPage(obJect sender, PrintPageEventArgs e) {
   Graphics 9 - e.Graphics!

   Size stringSize;
   using (Font aria124bold · new Font("Arial", 24, FontStyle.Bold))
       stringSize · Size.Ceiling(
             9.MeasureString("Bee Simulator". arial24bold));
       g.FilIEllipse(Brushes.Gray,
           new Rectangle(e.MarginBounds.X + 2, e.MarginBounds.Y + 2,
           stringSize.Width + 30, stringSize.Height + 30));
       g.FiIIEIIipse(Brushes.Black,
           new Rectangle(e.MarginBounds.X, e.MarginBounds.Y,
           stringSize.Width + 30, stringSize.Height + 30));
       g.DrawString("Bee Simulator", arial24bold,
           Brushes.Gray, e.MarginBounds.X + 17, e.MarginBounds.Y + 17);
       g.DrawStringr-eee Si mujator«, arial24bold,
           Brushes.white. e.MarginBounds.X + 15, e.MarginBounds.Y + 15);


   int tableX - e.MarginBounds.X + (int}stringSize.width + 50;
   int tablewidth = e.MarginBounds.X + e.MarginBounds.Width - tableX - 20,
   int !irstColumnX • tableX + 2;
   int secondColumnX • tableX • (tableWidth / 2) t 5;
   int tableY - e.MarginBounds.Y;

   tableY - PrintTableRow(g, tableX, tableWidth, firstColumnX,
        secondColumnX, tableY, "Bees", Bees.Text);
   tableY • PrintTableRow(g, tableX, tableWidth, tirstColumnX,
        secondColumnX. tableY, "Flowers", Flowers.Text);
   tableY - PrintTableRow(g, tableX, tableWidth, firstColumnX,
         secondColurnnX. tableY, "Honey in Hive", HoneylnHive.Text);
   tableY - PrintTableRow(g, tableX, tableWidth, firstColumnX,
        secondColumnX, tableY, "Nectar in Flowers". NectarInFlowers.Text);
   tableY • PrintTableRow(g, rabjex, tableWidth, firstColumnX,
        secondColumnX, tableY, "Frames Run". framesRun.Text};
   tableY • PrintTableRow(g, tableX, tableWidth, firstColumnX,
        secondColumnX. tableY, "Frame Rate", FrameRate.Text);

   g.DrawRectangle(Pens.Black, tableX, e.MarginBounds.Y,
        tableWidth, tableY - e.MarginBounds.Y);
   g.DrawLine(Pens.Black, secondColumnX, e.MarginBounds.Y,
        secondColumnX. tableY);
```

We gave you this part already. It draws the oval header, and sets up variables that you'll use to draw the table of bee stats.

Did r- figure out how the PrintTableRow() ..ethod works? All r- need to *do* i. call it once per row, and it prints whatever text r- want i" the two colu..ns. The trick is that it returns the new tableY value for the next row.

Don't f get to draw the rect ngle around the table and the line between the columns.

```
using (Pen blackPen = new Pen(Brushes.Black, 2))
using (Bitmap hiveBitmap - new Bitmap(hiveForm.ClientSize,Width,
                                      hiveform.ClientSize.Height))
using (Bitmap fieldBitmap - new Bitmap(tieldform.CUentSize.Width,
                                       fieldform.ClientSize.Height))
{
    using (Graphics hiveGraphics = Graphics.rromlmage(hiveBitmap))
    |
        renderer.PaintHive(hiveGraphics);

    int hiveWidth = e.MdrglnBounds.Wldth / 2;
    float ratio = (float} hiveBitmap.Height / (float)hiveBitmap.Width;
    int hivel\eight = (int) (hiveWidth • ratio);
    int hiveX - e.MctrginBounds.X + (e.MarginBounds.width - hiveWidth) / 2;
    int hiveY = e.MarginBounds.Height / 3;
    g.Drawlmage(hiveBitmap, hiveX, hiveY, hiveWidth, hiveHeight);
    g.DrawRectangle(blackPen, hiveX, hiveY, hiveWidth, hivel\elght);

    using (Graphics fieldGraphics = Graphics.Fromlmage(fieldBitmap))
    (
        renderer.PaintField(fieldGraphicsl: e.

    int fleldWidth = e.MarginBounds.Width;
    ratio = (float)fieldBitmap.Height / Ifloat)fieldBitmap.Width;
    int fieldHeight = (int)(fieldWidth • ratio);
    int fieldX = e.MarginBounds.X;
    int fieldY = e.MarginBounds.Y + e.MarginBounds.Height - fieldHeight;
    g.DrawImage(fieldBitmap, rteiox, fieldY, fieldWidth, fieldHeightj;
    g.DrawRectangle(blackPen, fieldX, fieldY, fieldWidth, fielctHeight);
}
```

Since the pen and the two bitmaps need to be disposed, we put the.. all in one big using block

You'll need a black pen that's 1 pixels wide to draw the lines around the screenshots.

The bitmaps need to be the same size as the for..: 1 drawing area, so ClientSize comes in handy

The PaintHive() method needs a Graphics object to draw on, so this code creates an empty Bitmap object and passes it to PaintHive()

MarginBound'.Width has the width of the printable area ee the page. That's how wide the field screenshot should be drawn

Here's where the height of the screenshot is calculated using the form's height–width ratio.

```
private void printToolStripButton Click{object sender, EventArgs e) (
    bool stoppedTimer = false;
    if (timerl.Enabled) (
        timerl.Stop();
        stoppedTimer - true;
    ]
    PrintPreviewDialog preview    new PrintPreviewDiarog();
    PrintDocument document - new PrintDocument();
    preview.Document _ document;
    document.PrintPage += new PrrntPageEventHandler(document PrrntPage);
    preview.ShowDialog(this);
    if (stoppedTimer)
        timerl.Start();
```

Here's the code for the Print button. It pauses the si..ulator (if it's running), creates a PrintDocument, hooks it up to the PrintPage event handler, shows the dialog, and then restarts the s..ulator

mini lab

There's so much more to be done...

You've built a pretty neat little simulator, but why, (uP now? There's a whole I", ""'n' that you can d" '"' your own. Here are some ideas see if you can implement some of tlH'lll.

Add a control panel

Convert the constants in the World end Hive cleeees to properties. Then add a new form with a control panel that has sliders to control them.

Add enemies

Add enemies that attack the hive. The more flowers there are, the more enemies are attracted to the hive, Then add Sting Patrol bees to defend against the enemies and Hive Maintenance beef! to defend and repair the hive. Those bees take extra honey,

Add hive upgrades

If the hive gets enough honey, it gets bigger. A bigger hive can hold more bees, but takes more honey and attracts more enemies. If enemies ceuee too much damage, the hive gets smaller again.

A good si..ulation will have lots of tradeoffs, and will give the user ways to decide which tradeoffs to ..ake to influence the progress of the hive

Add a queen bee who lays eggs

The eggs need Baby Bee Care worker bees to take care of them. More honey in the hive causes the queen to lay more eggs, which need more workers to care for them, who consume more honey.

Add animation

Animate the background of the Hive form so the sun slowly travels across the sky. Make it get dark at night, and draw stars and a moon. Add BOme perspective-make the bees get smaller the further they get from the hive in the field of flowers.

Use your imaginationl

Try to think of other ways you can make the simulation more interesting or more interactive.

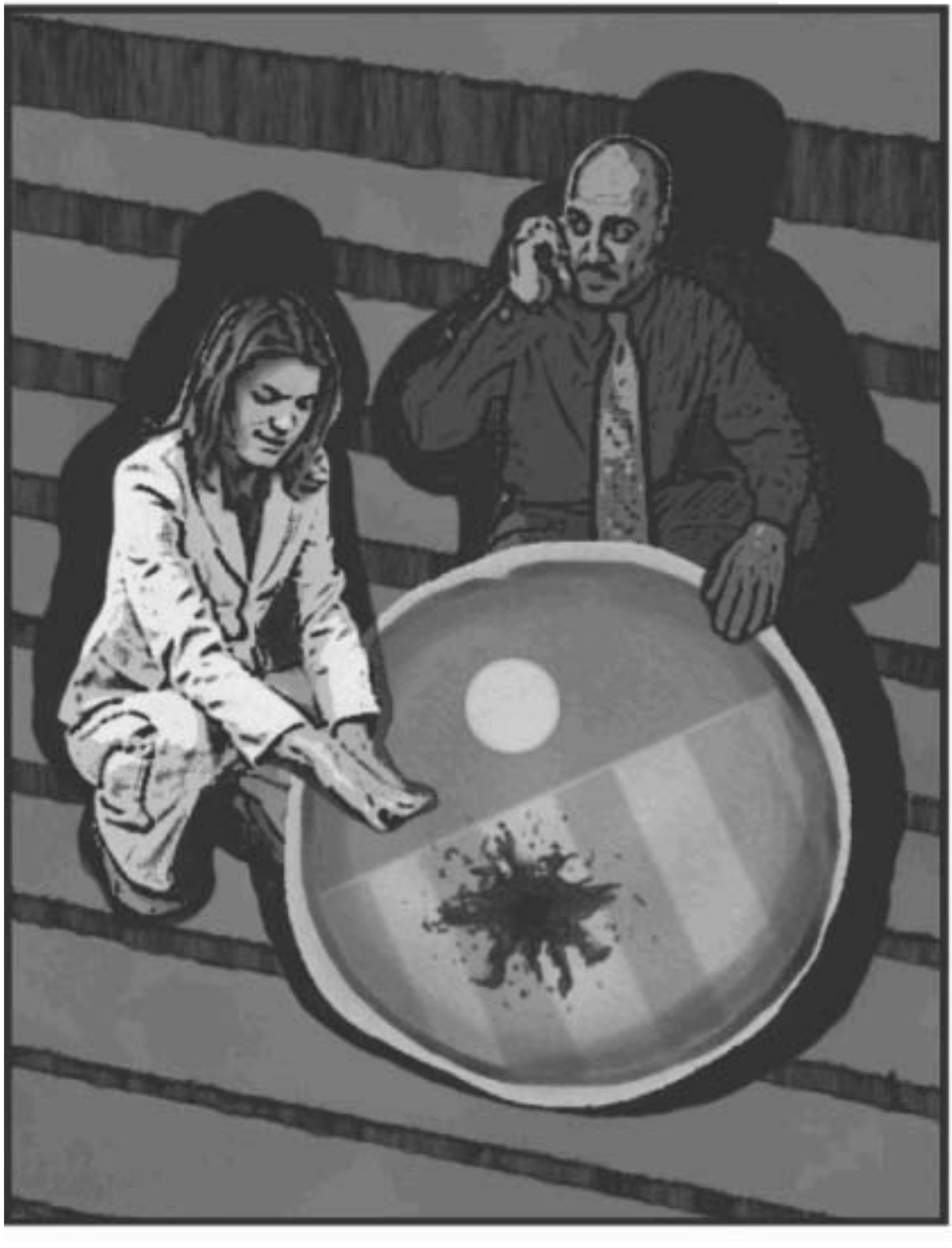
Did you come up with a cool modification to the simulator? SI,ow off your skills—upload your project's **source code to the Head First C#** lorum, at **www.headfirstlalss.com/l,ooks/hfcsharp/.**

CAPTAIN AMAZING

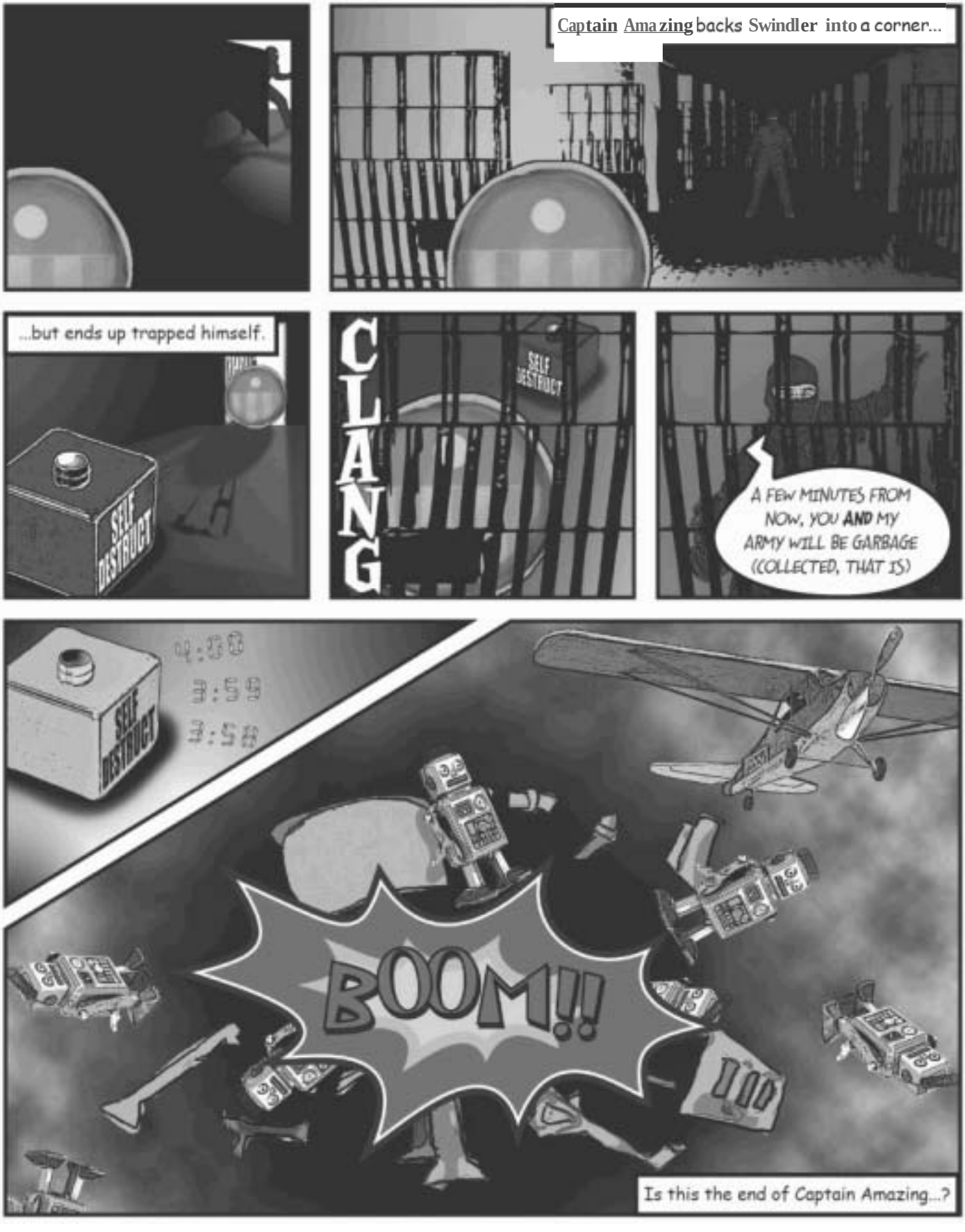
THE DEATH OF THE OBJECT

Head First Labs

$2.98 Chapter 14




Captain Amazing, Objectvi'le's most amazing object, pursues his arch-nemesis...
I'VE GOT YOU NOW, SWINDLER.
YOU'RE TOO LATE! AS WE SPEAK MY CLONE ARMY IS GATHERING IN THE FACTORY BENEATH US...
... READY TO WREAK HAVOC ON THE STREETS OF OBJECTVILLE!
Welcome to
Objectville
Home of Polymorphism
POW!!
I'LL TAKE DOWN EACH CLONE'S REFERENCES, ONE BY ONE.


Captain Amazing backs Swindler into a corner...
...but ends up trapped himself.
SELF DESTRUCT
CLANG
A FEW MINUTES FROM NOW, YOU AND MY ARMY WILL BE GARBAGE (COLLECTED, THAT IS)
4:00
3:59
3:58
BOOM!!
Is this the end of Captain Amazing...?

Sharpen your pencil

Below is the code detailing the fight between Captain Amazing and Swindler (not to mention his clone army). Your job is to draw out what's going on in memory when the FinalBattle class is instantiated.

```
public class FinalBattle {
  pUblic CloneFactory Factory = new CloneFactory(),
  public List<Clone> Clones - new List<Clone>();
  public SwindlersEscapePlane escapePlane;

  pubLic FinalBattle() {
    public Villain swindler - new Villain(this),
    using (Superhero captilinAmazing · new Superhero() I {
      Factory.PeopleInFactory.Add(captainAmazing);
      Factory.PeopleInFactory.Add(swindler);
      captainAmazing.Think("l'll take down each Df the clones' references,
                            one by one");
      captainAmazing.IdentifyTheClones(Clones);
      captainAmazing.RemoveTheClones(Clones),
      swindler.Think("A few minutes from now, you AND my army will be garbage");
      swindler.Think("(collected, that is!)");
      e5capePlane - new SwindlersE5capePIane(swindler);
      swindler.TrapCaptainAmazing(Factory);
      MessageBDx,Show("The Swindler escaped");
    }
  }
```

You can assu..e that Clones was set using an object initializer.

We've gotten you started here, with what's going on in the factory object.

Draw what's going on right here, when the SwindlersEscapePlane object i. instantiated

Draw a picture of what the heap will look like exactly one second after the FinalBattle constructor runs.

```
[Serializable]
public class SuperherD : IDisposable {
      private List<Clone> clonesToRemove · new List<Clone>();
      public void IdentifyTheClones(List<Clone> clones)
          fDreach (Clone clone in clones)
              clonesToRemove.Add(clone);
      |
      public void aemovernecrcnes IList<Clone> clones)
          foreach (Clone clone in clonesToRemove)
              clones.Remove(clone);
```

e-s-, There's more code here (including the Dispose() ..ethod) that we aren't showing you but r_ don't need it to answer this

```
pUblic class Villain {
  private FinalBattle finalBattle;
  pUblic VillainIFinalBattle finalBattle)
    thi5.finalBattle - finalBattle;
  |
  public void TrapCaptainAmazing(Clonefactory factory) {
    factory.SelfDestruct.Tick += new EventHandler(SelfDestruct_Tick);
    factory.SelfDestruct.Interval - 600;
    factory.SelfDestruct.StartI),
  |
  private void SeltDestruct_Tick(object sender. EventArgs e)
    finalBattle,factory - null;
```

```
pUblic class SwindlersEscapePlane (
  public Villain PilotsSeat;
  public SwindlersEscapePlane(Villain escapee) {
    PilotsSeat • escapee;
```

```
pUblic class CloneFactory {
  public Timer sertoeet.ruct • new Timer();
  public List<object> PeopleInFactory = new List<object>();
```

There's a clone class and that we're not showing you in this code too. You don't need it to answer the questions.

Make sure r- add labels to your objects to show the ..eference variables that a.e pointing to the...

We started the first one for you Make sure you d.aw in lines showing the architecture—we d.ew a Ine from the clone factory to the Superhero object, because the factory has references to it (via its PeopleInFactory field).

o

r



We've left space, as there is more to be drawn at this stage.

You can leave out the Clone and List objects, and focus on first-order objects in these drawings.

2

3

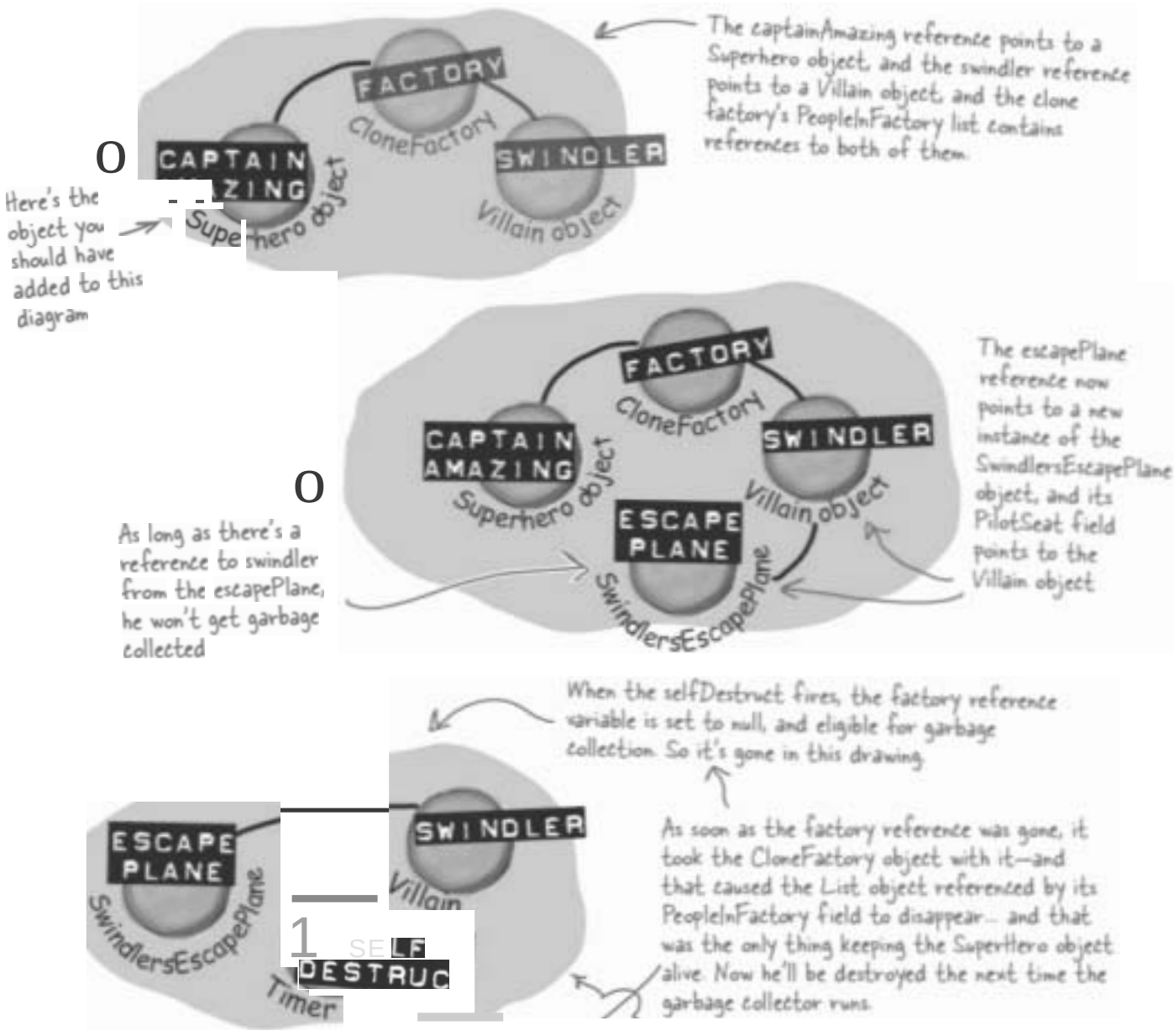
Your job is to draw what's going on in these two bits of memory, too.

Based on your diagrams, where in the code did Captain Amazing die?

Be sure and annotate that on your diagram. too.

Sharpen your pencil Solution

Draw what's happening in memory with the FinalBallle proqram.



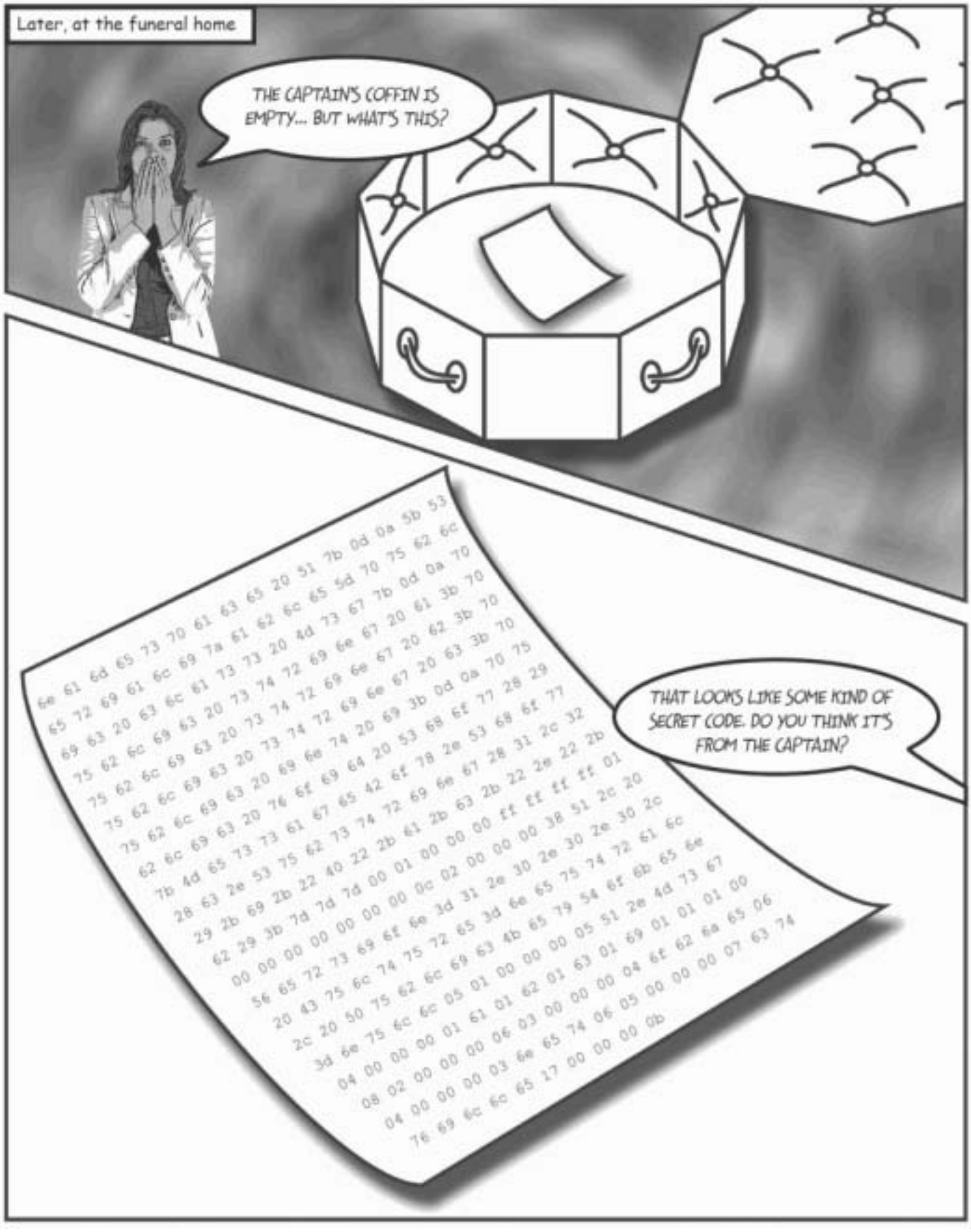
Based on your diagrams, where in the code did Captain Amazing die?

```
void SelfDestruct_Tick(object sender, EventArgs e) {
    FinalBattle factory = null;
}
```

One second after the FinalBattle constructor ran, the hero was gone.

Once FinalBattle was set to null, it was ready for garbage collection. And it took the last reference to the Ceptein with it!

Once the Superhero instance had no clone factory referencing it, it was marked for garbage collection too


Later, at the funeral home
THE CAPTAIN'S COFFIN IS EMPTY... BUT WHAT'S THIS?
THAT LOOKS LIKE SOME KIND OF SECRET CODE. DO YOU THINK IT'S FROM THE CAPTAIN?

Your last **chance** to **DO** sOlllefhil1g... your **object's** fil1alizer

S,metimes yc'u need I" Ibe sure something happens ***before*** yc,u,. "bj""1 gets garbage collected. You might walll to release """"ll""1i,ms " ,. resources... "r perhaps send a coded messag I" lhe world.

,\ special method in your object called the **finaliz**..r allow you to write code lhat will always execute when your "hj" '-l is destroyed. Think of il as youu object's personal f ina11y bl'k'k: il gets executed last, no matter what.

Here's an example of a destructor in the `Clone` class:

```
[Serializable]
public class Clone
  string Location;
  int CloneID;

  public Clone lint cloneIO, string location){
    this.CloneID _ cloneID;
    this,Location = location;

  public void TellLocation(string location, int cloneID){
     Console.WriteLine("My Identification number is {Ol and  +
                  "you can find me here: {1}.", cloneID, location);

  public voi d WreakHavoc(){.. I
    lone 0  {
    TellLocation(this,Location, this.CloneID);
    Console.WriteLine ("{Ol has been destroyed", CloneID);
  }
}
```

Here's the constructor. It looks like the CloneID and Location fields are populated anyti..e a Clone gets t:reated

This ~ (or "tilde") character says that the code in this block gets run when the object is garbage collected.

This is the finalizer. It sends a message to the villain telling the ill-fated clone's location and ID. But it will only run when the object is garbage collected.

You write a finalizer method just lik,' a constructor, but instead of an access modifier. you pill a - in Ii,,,,, or 110e class name. That tells .NET tharrhe code ill the finalizer hlock ,holll<1 Ik' l'Ul1 any tjtm- it garbage collects the object.

Additionally, finalizers can't have parameters, since .NET calls lhis object, rather than any other object.



Destructors and finalizers are the same thing.

Sometimes you ?1 hear people refer to an object's Finalizer method, and sometimes *to its destructor, Both of those terms refer to* a *method that runs when an object* is ***garbage*** *collected_"Finalizer" is generally replacing "destructor"* as *the name for this We'll use "destructor"* a ***few*** *times. just because some of the IDE's error messages do.*

Whell EXACTLY does a fillalizer rUII?

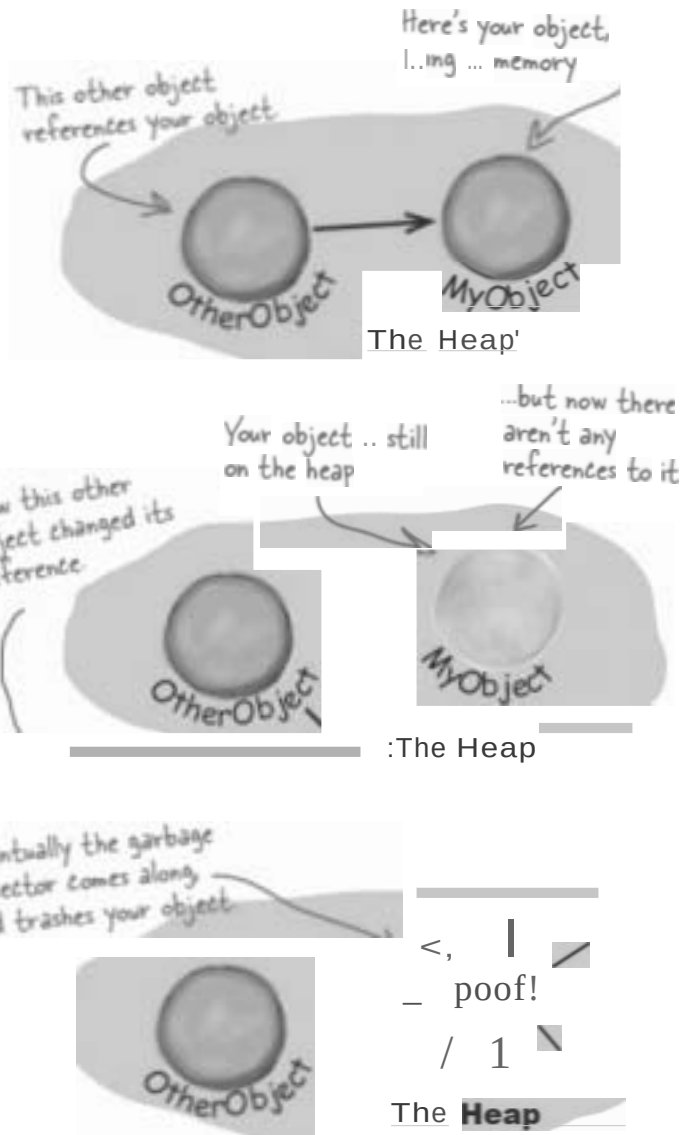
The tin.·lizer vou build fo your object runs wh.." that obje.." gets garbage coll cted. .And ga..b llection happens aft.., ..1Irefe nces to y"'r "I,j ! go aw,·y. But garbage .·„11.. tion ' " ., ..lways happen *right after* the references ;,,,. gou .

Suppose you have an object with a ref u to ii, .NET sends due garbage collector to work nd it checks out vour •obje.., . But sinc.. ,h.· are references l., your •bjec,. the garbage coll..:lor ig ;1 and moves along. Your •bject keeps living "n in memory.

Then, something happens That last object holding a reference 1..__ . ,bject decides '" move oil. Now, you object is sitting ill memory, with no references. I, can't Ibe accessed. It's basically a d..ad **objec**l,

Bm here's ,h,' thing. ***Garbage collection is something that .NET control••*** no' your objects. So if the garbag collector isn't sent out again li ,l', say, a few seconds, or maybe even a li·" minutes, your object still lives "n in memory. It's unusable, bu i, hasn't been garbag llecte..1, And **any** fin**aliz**....)'OU" obj....1 **has** d " **not** (y0'I) **g**..1 run.

Finally, .NET ".",], ,I,,· garbage collector out again This ti",,·. your object is de..d. .."',I ,h.-,,ll :tor I,sses i' away. Here, finally, your destructor nlll possibly several minutes aft..r the last referenc.. to th.. object was reme"'ed1"r changed.



Vou Ca" SUGGEST to .NET that it's time to collect the garbage

.NET does k-l you ...***ggest*** that garbage colle !i"'" would l..· a good ide **Mo**..l **1imes,** yoll·1l **n**..**ver** u**se** Ihi" **method,** b **u**.... **garbag**.. **collection** i" IlIa**e**d 10 **respond** 10 **a** 1..1 of con**dit**;on" in Ihe e LR a nd **calling it *isn't rea****Iß' a good idea.*** 11m just '" s... how a finalizer works, you could call for garbage collection "n ,.ur ""1', If that's wha ,,," wan [",]". just call GC:collect O _ -

```
public void RemoveTheClones(
                    Llsl<Clone> clones)
  foreach (Clone clone in clonesToRemove)
  clones.Remove(clone);
  GC.ColleetO;
}
```

Bd careful, though. That method doesn't ***force*** .NET '0garbage collect things immediately. It jus says, "Do garbag collection as soon ," possible."

l1ispose(J works with `using`, fillalizers work with garbage collectioll

`Dispose ()` runs whenever an object that is created in a `using` statement is set 10 null or lose all or its references. If vuu don'l u-e a `using` statement, thenjust setting the reference 10 llull won't cause `Dispose ()` 10 get called —you'll "",'d 10 '-illl il directly. ,\" object's finalizer runs at garbage collection for that particular object. Let's create a ,'"upl,' "l' objects, ilml see how these two methods differ:



o **Create** a **Clone** class **and make** sure it **implements** I Dispasable, Tl,,' class ,hould have one `int` automatic property ('all,'d Ill. II has a constructor, a `Dispose ()` method and a finalizer:

```
public class Clone : IDisposable I
    public int. tD { get; private set; }

public,Clonelint IDj I
)       thLs.tD = ID;

public void Dispose!} {
    eteeseceaox.snov("I"ve been disposed!",
                     "Clone #" + 10 + " says.,.n);

-CLone () {
    MessageBox.unOV("Aaargh! You got me!",
                     "Clone #" + 10 + " says..");
```

Since the class imple..ents IDisposable, it has to have a Dispose() method

Here's the finalizer It will run when the object gets garbag collected

o Create a **Form** with **three** buttons, Create one instance of `Clone` ill,id,' the `Click` handler for the first hutton with a `using` statement. Here's the first part of the code for the button:

Here's the form you should create



The ..ethod creates a new Clone and the.. i....ediately kills it by taking away its reference

```
private void clonel_Click(object sender, EventArgs el
    using (Clone clonel - new Clone(l)) (
        // Do nothing!
```

Since we declared clonel with a using stateme..t, its Dispose() ..ethod gets run

As soon as the using block is done and the Clone object's Dispose() method is called, there's no more reference to it and it gets marked for garbage collection.

3 **Implement the other two buttons.**

Create another instance of Clone in the second button's Click handler, "",I set it to null manually:

```
private void clone2_Click(object sender, EventArgs e) {
    Clone clone2 = new Clone(2);
    clone2 = null;
}
```

Since this doesn't use a using statement, Dispose() won't ever get run, but the finalizer will.

For the third button, add a call to GC.Collect() to suggest garbage collection occur.

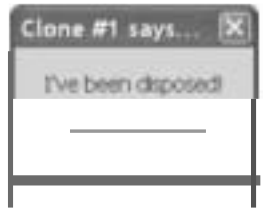
```
private void gc_Click(object sender, EventArgs e) {
    GC.Collect();
}
```

This suggests that garbage collection run.

Remember, normally it's not a great idea to do this. But it's fine here, because it's a good way to learn about garbage collection.

4 **Run the program and play with Dispose() and finalizers.**

Click on the first button and check the message box: Dispose() runs first.



Even though the Clone1 object has been set to null and its Dispose method has run, it's still on the heap waiting for garbage collection.



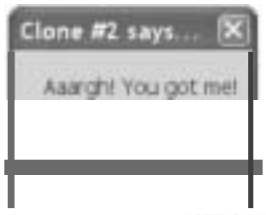
In most cases, you won't see the garbage collection message box. Because your code set it to null but garbage collection hasn't run yet.

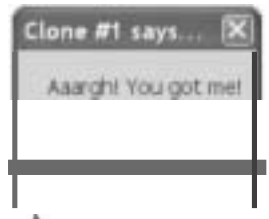
Now click on the second button... Nothing happens, right? That's because we didn't use a using statement, so no Dispose() method. And until the garbage collector runs, you won't see the message boxes from the finalizer.

Now Clone2 is on the heap, too, but without any references to it.



Now click the third button, to suggest garbage collection. You should see the finalizer from both Clone1 and Clone2 fire up and display message boxes.







When GC.Collect() is run, both objects run their finalizers and disappear.

Play around with the program. Click the Clone #1 button, then the Clone #2 button. Then the GC button. Do it a few times. Sometimes clone #1 is collected first, and sometimes clone #2 is. And once in a while, the garbage collector runs even though you didn't ask it to using GC.Collect().

Finalizers can't depend on stability

When you w, it.. a finalizer; you can't depe,,,1 on it running a any one ,im..., Even if you call GC.Collect () - "hid , you should avoid. unless you have a really good reaso, t.. d.. i, you're onlv ***suggesting*** Ih.U the garbage collector is run. I,'. , a gu...rantee that it'll happen right tiow.. \ ,ıc1wl " it does. you h."r 'io way of knowing wha order tl ojects ill l... collected.

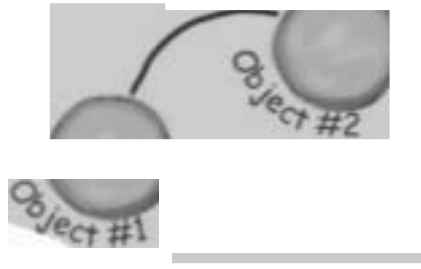
So what does that l1I..all. in practical terms? Well. think ..lıouu what happens if you've got two obje..ts th... h e references I••".'"h other. If objec # 1 is colle.:ted li . then objec 1 #2's refere,see '" it is pointing to an ·object that's n" I,onger there. But if object #2 is collected lit.,. then object #1's nf.·,.."',', is invalid. So " I"" lh.•, n,,"""" is Ih.,',>'oll ***ca,, ·t dep..''d*** *oıı* ***r.1..rences in your object*** " ***finaliz..r.*** \Which m.·"", thal it's a really I",d id," to try 10 do something inside a tin.,li,,·, th,lt d" I"·t,,l, »n references being valid.

Serializati«" is " really good ,'",,,n]1I,' •,I'something II"" vou **sh**..uldn'l do insid.. **a finaliz**..r , If your object's got ; I"",, 'h "I' references to "th..r objects, serialization '""1"".1,•"" "// "I' tik«se objects s1ill being ill memory.. ..id „ll "f tl,," »bjects they n·r.·"•,,,"" ..and Ilıe ones ,10",,' •bjc ts referen.."'. and s« on. So if you try ı·· serialize wh n garbage collection is happening, you coulc1.,,,.1 up rni **ing** vital parts of y«",,. program because some objects ,· ,ld· been collected ***before*** th.. finalizer ran.

Luckily. C# giv us a really good solution, 1" this: `IDisposable`. .Anything lh,.. could ,..odifv vour core data or that depends on other objects b..ing in memory needs t.. happer, ... part of a `Dispose()` Oıethod. not a finalizer.

Some people like ,•• think of a finalizers as a kind of fail...afe for ,I,,· `Dispose()` " th"o,.\ud tl.." makes sense you ..ıw with your (: object tha just I se you1implement IDisposable, that doesn' n....an tl... •bject' Dispose() method " ill get called. But you need to) I", •.,,,..ful if your Dispose() "",th,"I depends " II "tl1er obje :ts Ih" t ,ıf' on rh.. heap, then calling Dispose() from your li""I;,er can cause trouble. '1Ine 1"" 1way around this is 1" make sure you alway" u".. a **using staten**...,,1 any time you're creating a u 11 1;,1" »able objec r.

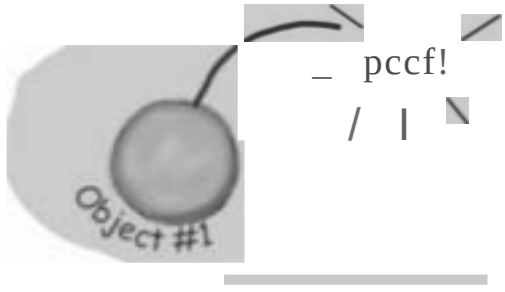
Let's say you've got two **objects** that have **references** to each other•••



•••if **they're** both **marked** for **garbage** collection at the same time, then object .1 could disappear first•••



•••on the other hand, object ' 2 could disappear before **object #1.** You've got no way of knowing the order•••



•••and that'. why one object's finalizer can't rely on any other object still being on the heap.

Make an object serialize itself in its Dispasell

Once you understand the difference between Dispose() and a tinalizer, it's pretty easy to write objects that serialize themselves out automatically when they're disposed of.



Make the Clone class (from page 630) Serializable.

Just add a Serializeable attribute to the top of the class so that we can save the file out.

```
[Serializable]
public class Clone : IDisposable
```

Modify Clone's Dispose() method to Serialize itself out to a file.

Let's use a BinaryFormatter to write Clone out to a file in Dispose():

```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

// existing code

public void Dispose() {
  string filename = "C:\\Temp\\Clone.dat";
  string dirname = "C:\\Temp\\";
  if (File.Exists(filename) == false) {
    Directory.CreateDirectory(dirname);
  }
  BinaryFormatter bf = new BinaryFormatter();
  using (Stream output = File.OpenWrite(filename)) {
    bf.Serialize(output, this);
  }
  MessageBox.Show("This is " + this.ID +
                  " must... serialize...object.");
}
```

You'll need a few more using statements to access the VO classes we'll use

This is straight serialization, although we do hardcode in the file, since Dispose() can't take any parameters.

You could set a class like this up to take in a filename to write itself to in its constructor.

Run the application.

You'll see the same behavior you saw on the last few pages... but before the Clone1 object is garbage collected, it's serialized to a file. Look inside the file and you'll see the binary representation of the object.



What do **you** think the rest of the SuperHero object's code looked like? We showed you part of it on page 624, Could **you** write the rest now?

Fireside Chats



Tonight's talk: **The Dispose() method and a finalizer spar over who's more valuabl··**

Dispose:

To l,,· l"·nest. I'm a little surprised I was im ited 11,·re. I thought th.. Ing programming world had come ,,, .· consensus. I n..;",. I'm way n"·re , du··ble 'h··n you are' Really, you'· j tuv fe..ble. You can't even serialize ",..,rs·If "U·..il r ,., ,L" .,. .anything. Pretty unstable. are"·1you!

There·· .,ll int..rface specifically **because** I'm s· i"'p"" ,,m. In fact, I'm rh.. ,."I" method in it!

OK, you're right, progr··mn...rs need to know tl..y're going to need me and ..itl..r ,·"n "'.. directly or use a using statement '" call m'. But they always know wl..n I'", gonna "Ill .....1,1,.\.,;,n use n..· to do whatever they ,,eed1 to do ,ll clean "I' afte.. ,h,·i..object. I'm pow...'[ul, reliable, and ..asy to use. I'm a ,riple threat. And you? Nobody knows exactly "'1"'ll you'll nm o,. what the stat· "f Ih.· applic.,.1ion " ill be when you finally do decide '" ,I"." "p.

So there's ba"i, ally ,).thing vou ,'a" ,I" Ih.ll I ";11I'1d". Bill you Ihink you're a big ,h", because yo" run when garbage collectio"' happens.

Finalizer:

Excuse n...? That's rich. I'm f OK. Well, I did,,', want 10 get imo this, hUI sinc.. we' alrea..lv stooping this low... at least I don', ' n in,..rfi...,.. I" get s,, '1.ect. Witho" rntepoeebte. y , just anotl..·.. useless "I<·lh,,<l.

Right, right... keep telling yourself ,hau. And wh.ll happen" "Ihen, son..some· forgets ,<l use a `using` statement when they instantiate th..ir object? Then vou're ,.. ,w-here '" b.. found.

Handles are what your programs use when they go around .NET and the CLR and interact directly with Windows. Since .NET doesn't know about them, it can't clean them up for you.

OK. IKII if you need to do something ." the very LO', moment wh..n an objec, is garbage collected, II ,... way In d" il without 111". I can free "I' network ,Un·es and windows handles and streams ;.'"" any hing else that might cause a problem for ,],.. res of the program if you don't clean it "I, I ca", make sure Ih 1your ,,1i" "' d, a1 with being trashed more gracefully, nd that's nothing ,n sneeze at.

I'll take that over your flash and attitude any day, pal.



there are no Dumb Questions

Q: Can a finalizar **use** all of an objecors **fields** and methods?

A: see. While you can't pass parameters to a finalize r method, you can use any 01 file fields In an object, e,ther directly or uSing this—but be careful. because if mose fields reference ofher objects. then the other objects may neve already been collected But you can definitely call cner methods in the object being finalized (as long as ecse methods don't depend on other objects)

Q: What **happens** to **exceptions that** get Ihrown in **a finalizer?**

A: Good question. n's totally legal to put a try/catch block insode a nrenzer method Give if a try yourself. Creafe a divide by zero exception inside a try block in the Clone program we just wrote. Catch If and fhrow up a message box mat says ·I lust caught an exception." right beroe the •...I've been destroyed." box we'd already wnllen, Now run the program and click on the first button and then the GC button, You'll see both the exception box and the destroyed box pop up (Of course. it generally a **really** bed **idea** to pop up message boxes in finalizers for objects that are more than just toys.. and those message boxes may never actually pop up}

Q: How oftI n **does** the **garbage** collector run aUomatically?

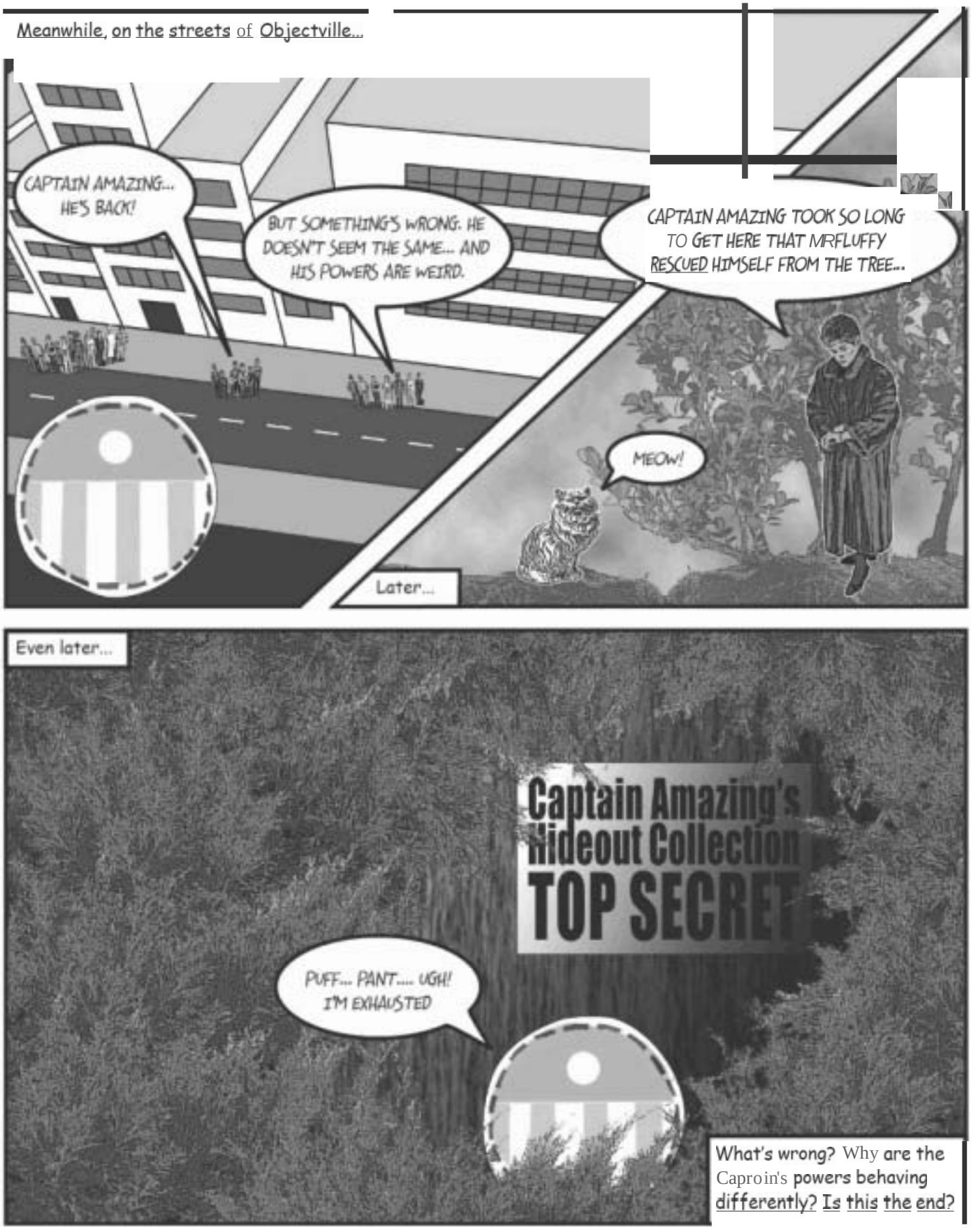
A: ttere'e no good answer to that one It doesn't run on an easily predldable cycle, and you dont have any firm control over it You can be sure it will be run when your program exits But if you want to be sure It'll run, you have to use GC.Collect() to set it off.. and even then, timing is an issue

Q: How **soon after** I call cc.ceneem will .NET **start garbage** collection?

A: When you run GC.Collect{}. you're lell,ng NET to garbage couect soon as possible, That's ***usually*** as soon as NET finishes whatever its doing. That means .I'U happen pretty soon, but you can't actually control when,

Q: If I absolutely need something to run. I put it in **a** finalizor, right?

A: It's possible that your finalizer wont run, It's possible 10 suppress finalizers when garbage collection happens. Or the process could end entirely Butas a general rue, your finalizer should run.


Meanwhile, on the streets of Objectville...
CAPTAIN AMAZING... HE'S BACK!
BUT SOMETHING'S WRONG. HE DOESN'T SEEM THE SAME... AND HIS POWERS ARE WEIRD.
CAPTAIN AMAZING TOOK SO LONG TO GET HERE THAT MRFLUFFY RESCUED HIMSELF FROM THE TREE...
MEOW!
Later...
Even later...
Captain Amazing's Hideout Collection
TOP SECRET
PUFF... PANT.... UGH! I'M EXHAUSTED
What's wrong? Why are the Caproin's powers behaving differently? Is this the end?

A struct *looks* like an object...

On....f the types ill .NET we haven't talked abou11 1I1111h is the s t r uct., struct is short for ..tructure. and at ruct.s look a lot like objects. ·11..:y h."t, fields and properties, just lil e object.. .And you can even pas them int.. a m..thod that takes an object type parameter:

```
public struct; AllllOstSuperhero : IDisposable
    public int SuperStrength;
    public lnt SuperSpeed { get; private set; }

    public void RemoveVillain(Villaln villain)
    |
        Console.WriteLine("OK. " • villain.Name +
            " surrender and stop all the madness!");
        if (villain.Surnmdered)
            villain.GoToJail();
        else
            villain.Kill!);


    public void Dispose() { ... |
```

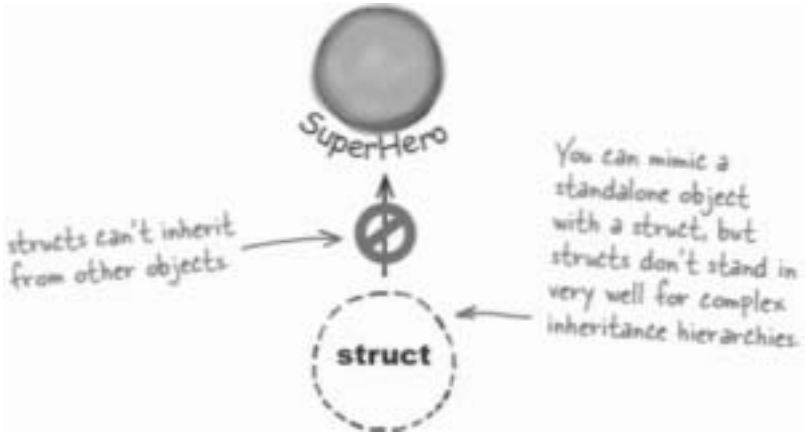
Structs can implement ..terfaces but can't subclass other classes

A struct can have properties and fields...

and define methods

..but *Isn't* on the heap

11m s t ruet, ***aren't*** objects u" the heap. They • .an have· methods and field t.." they ••an't h.,, :. finalizers. T h•y .•1s, can't inherit r" .rn nIh.., clas n, et.ruces, or h." .. cl..sses •r e t ructs inlo eri from Ihe111.



The power 01 ol.jects **lies** in **their** ahility to mimic **real-world behavior, through inheritance** anJ polJmoll'\'ism.

Structs **are best used** lor **storing** data. hut **are** not as usejul as **objects when** it **comes** to **representing** actual **behavior.**

Values get copied. referellces get assiglled

Y'" already have a sense or how some types are different Ihall others. ()n one hand you've got value **types** like i nt, booj, and string, ()n ,h,' "Ih"r hand. you've got **objects** tik,' L i s t. Stream. and Exception. And they don't 'lui,,' work exactly the same way. do they?

Here's a quick refresher on value types vs objects.

When you use the equals sign to set one value type variable to another. i, nlak.... a **copy of** th.. **value**. and afterwards the two variables aren't connected tu each other. (In the other hand, "h"ll you use lhe equals sign with references, what vou're doing is poin.ing bu.h **referenc..s** at .h.. **sam**.. obj**ect**

- \'ariable declaratiOll and assignment works the same with value types or object types:

int and str-S e value types List and Exception e object types

```
int howMany - 37;
string name - "fingers";
List<float> temperatures   new List<float>();
exceptr'on ex - new axceptfon("Does not compute");
```

These are all initialized in the same basic way

- Difference creep in when you start 10 assign values, though. Value type allare handled with copying. Here's an example:

This line copies the value that's stored in the fifteenMore variable into the howMa-t variable and adds 15 to it.

Changing the fifteenMore variable has no effect on howMany, and vice versa.

```
int fifteenMore = howMany;
fifteenMore += 15;
Console.WriteLine("howMany has {OJ, fifteenMore has ur-.
                  howMany, fifteenMore);
```

The output here sh..w that fifteenMore and howMany are ***not*** connected:

```
howMany has 25, fifteenMore has 40
```

- With object assignments, though, you're assigning references, not actual values:

This line sets the differentList reference to point to the sa··e object as the temperatures reference

```
temperatures.Add(56.5f);
temperatures.Add(27.4F);
List<float> differentList _ temperatures;
differentLisLAdd(62.9f);
```

Both references point at the same actual object.



So changing the List means both references see the update... ,illce (hey both point to a single List object.

```
Console.WriteLine("temperatures has (OJ, differentlist has {I)",
                  temperatures.Count(l, differentList.Count());
```

Th,' " 1lt1"' 1here d" Ill'nstrates (hal differentList and temperatures are actually pointing to the *sam e* object

When you called differentList.Add(), it added a new te..perature to the object that both differentList and temperatures point to.

```
temperatures has 3  differentList has 3
```

Structs are value types; objects are referel1ce types

When you create a struct, you're creating a **value** type. What that means is whe-n you use equals 1" set "n,·et.ruct variable equal to '''''Iher, you're creating a fresh *copy* of Ihe struct ill the new variable. So even Ih,ugh a st ruct *looks* lik,'an object, il doesn't ac1 like one.



O Creote o street coiled Dog

Her..'s simple struct 10 keep track 01 a dog. It looks just like an object, but it's not.

```
public struct Dog {
  public string Name;
  public string Breed;

  public Doq Lstrinq name, string breed) (
    this.Name · name;
    this.Breed · breed;


  pUblic void Speak() {
      ccoscre.wrrcer.iuer-nv name is (OJ and I'm a {1}.", Name, Breed);
```

Yes, this is not good encapsulation
Bear with us—we're making a point

O Create **a class** called **Canine**

Make an exact copy "I' the Dog struct, except replace struct with class and then replac.. Dog with Can in... Now you'll have a Canine class that you can play with, which i, alm"'1 exactly equivalent (o the Dog struct.

3 Add o button **that** makes **some** copies of Dogs and Canines

Here's the code:

```
Canine spot = new Canine("Spot", "pug");
Canine bob · spot;
bob.Name _ "Spike";
bob.Breed · "beagle";
spot.Speak();

Dog jake _ nell Dog("Jake", "poodle");
Dog betty · jake;
betty.Name · "Betty";
betty,Breed · "pit bull N;
jake.Speak();
```



O Before you press **that** button..,

\\'rile down the what vuu Ihink will be written 10 lhe console when \"11 nUl lhi, c"I,·:

Sharpen your pencil Solution

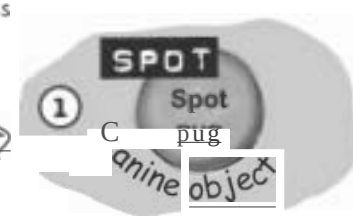
What did you think would get written to the console?

My name is Spike and I'm a beagle.
My name is Jake and I'm a poodle.

Here's what happened...

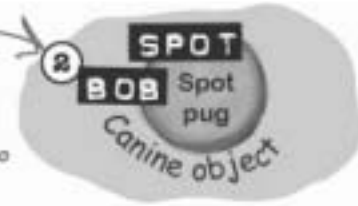
The bob and spot references both point to the same object, so both changed the same fields accessed the same Speak() method. But struct doll work differently. When you created betty, you made a fresh copy of the data in jake. The two structs are completely independent of each other.

A new Canine object was created and the spot reference points to it.




```
Canine spot = new Canine("Spot", "pug");  (1)
Canine bob = spot;  (2)
bob.Name = "Spike";
bob.Breed = "beagle";
spot.Speak(); (3)
```

The new reference variable bob was created, but no new object was added to the heap—the bob variable points to the same object as spot.

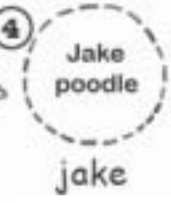


Since spot and bob both point to the same object, spot.Speak() and bob.Speak() both call the same method, and both of them produce the same output with "Spike" and "beagle".



```
Dog jake = new Dog("Jake", "poodle");  (4)
Dog betty = jake;  (5)
betty.Name = "Betty";
betty.Breed = "pit bull";
jake.Speak();  (6)
```

When you create a new struct, it looks really similar to creating an object—you've got a variable that you can use to access its fields and methods.



Here's the big difference. When you added the betty variable, you created a whole new struct, even though you didn't use the new keyword.





When you set one str-eet **equal** to anetlier, you're "-eating a **fresh COPY** of **the** data **inside the** struet. **That's** because struet is a VALUE TYPE.

Since you created a fresh copy of the data, jake was unaffected when you changed betty's fields.





The stack vs. the heap: more on memory



Behind the Scenes

It's easy I', understand how a `struct` differs from an object you can make a fresh copy ..f a struct just using equals, which you can't .In with an object. But what's really going ull behind rh,' scenes? The reas›ll s t r uct.s "l(-I likt" other value types i, that **value types** don't liv.. on **the heap**. The .NET CLR divides vour data inl" two places illlll'·'"Ol). You already know th.u objects live 011 the h..**ap**. It also k"('I" ,mo,h"r part or memory called the **stack** '0 store all of 11",1,,1",,1variables you declare ill your methods and the parameters lhat you pass into those methods. You callihink of the stack as a bunrh "I slots that you can stick values 'll_\\'hen a method gets ",,ll,"1. the CLR adds m"r,' slots 10 Ihe lOp "r the stack. When it returns, its slots 'H' removed.

Re",ember, when your progra..'< running, the CLR is actively managing "emory, dealing with the heap and collecting garbage

Even though you can set an object variable equal to a struct, structs and objects are different

The Code

Here's code thal you might see In a program.

```
amne spo  - new                'pug");
og jake - new Dog("Jake", "poodle");
```

Here's what the stack looks like after these two lines of code run.

The Stack

This Is where structs and local variables hang out.



```
Canine spot - new Canine("Spot", "pug");
Dog jake = new Dog("Jake", "poodle");
Dog betty = jake;
```

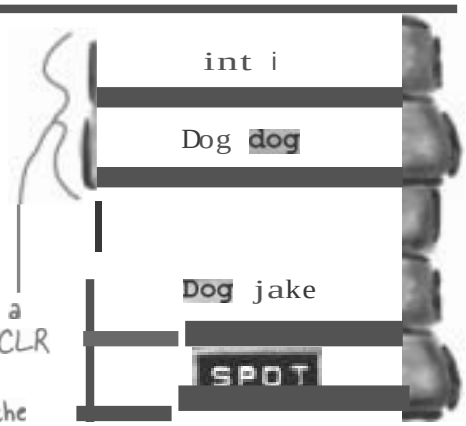
When you create a new struct—or any other value type variable—a new "slot" gets added onto the stack. That slot is a copy of the value in your type



```
Canine spot - new Canine("Spot", "pug");
Dog jake = new Dog("Jake", "poodle");
Dog betty = jake;
      ree rmes ja e ;

public SpeakThreeTimes(Dog dog)
    int i;
    for (i - 0; i < 5; i++)
        dog.Speak();
```

When you call a ..ethod, the CLR puts its local variables on the top of the stac It takes the... off when it's done.



Wait a minute, Didn't you just say that you cen set an object variable equal to a struct? If an object's on the heap, and a struct is on the stack, what happens?



You can also use the "is" keyword to see if an object is a struct, or any other value type, that's been boxed and put on the heap

When you set an object **equal** to a value type, it gets **boxed.**

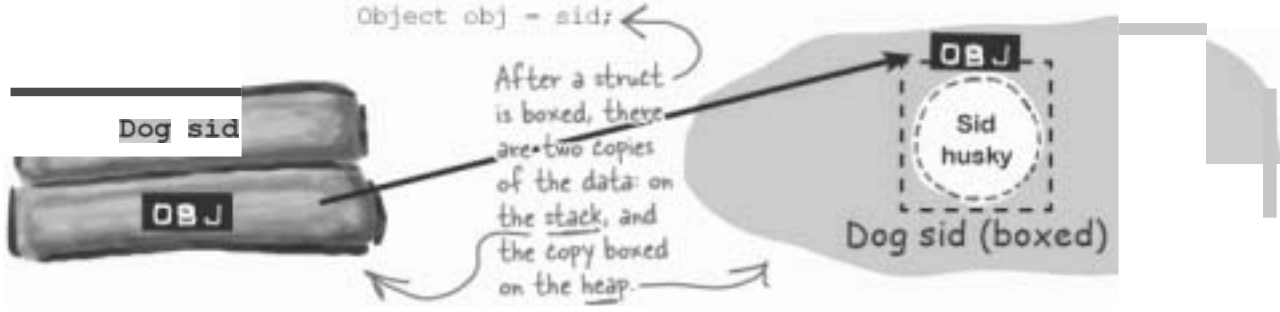
There are some times that you need to be able to write a method that can take either a value type ***or*** a reference type - per],ap' a "'"thod that ,'a" work with either a Dog `struct` 0,. a Canine object. If you find yourself iUlhat situation, you ,"all use the `object` keyword:

```
public WalkDogOrCanine(object anything) { ... )
```

If you send this method a struct, th,-struct gets **box**..d into a special obje,:t "wrapper" th.,t allows it to live on the heap. While the wrapper's on the heap you can't do IIIIlch with the struct. You have to "unwrap" the struct to work with it. Lus-kilv, all "I' this happens *automatically* whe-n you set an object equal to a value tvpe, or pass a vahu- type into ; method that expects an object.

❶ Here's what the stack and heap look like after you create an object variable and set it equal to a Dog struct.

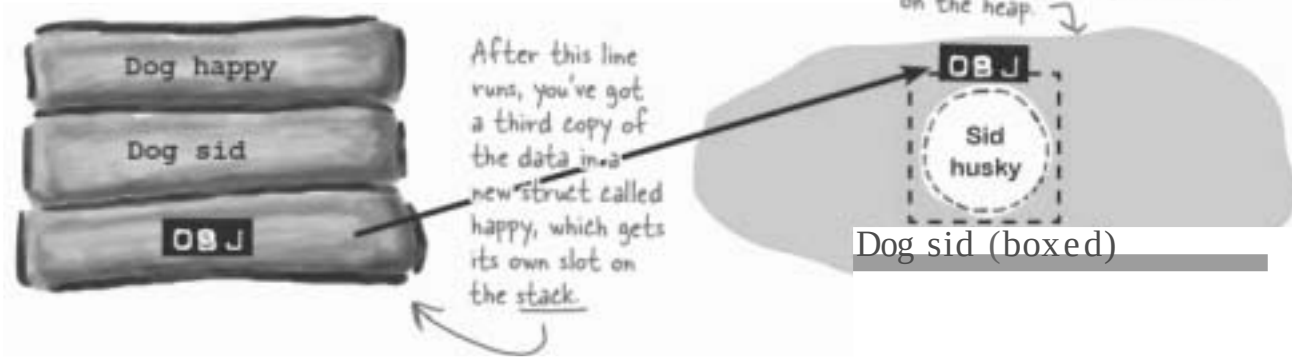
```
Dog sid = new Dog("Sid", "husky");
Object obj = sid;
```



❷ If you want to unbox the object, all you need to d,) is cast it I" the right type and it gets unhoxed automatically. This i, where the `is` keyword comes in handy.

```
if (obj is Dog)
    Dog happy = (Dog) obj;
```

These are structs, so unless they're boxed, they don't live on the heap.



pool Puzzle



Your Job i. 10 take snippets from the pool and place them into th.. blank lines in the code. You may u... th.. sam...nippet mo.ethan once, and you won't need 10 use all the snippets. You. go.l is 10 make the code write this output to the consol.. when a new instance of the Faucet d... is cr••ted

```
public: class Faucet {
    public Faucet() I
        Table wine· new Table1):
        Hinge book • new Kingen ;
        wine.SetA(book);
        book.SetB(wine);
        wine.Lamp(10);
        book.Garden.Lamp("back in-I;
        book.Bulb -. 2,
        wine.Lamp("minutes");
        …ine.Lamp(book);
```

Output when you **create a** new Faucet object:

back in 20 minutes

Here's the goal to get this output

```
public          Table (
    public string Stairs;
    public Hinge Floor:
    pUblic void Set(King.. bl (
        Floor - b;

    public void Lamp(object oilI {
        it coil _ _ int)
            _____.Bulb · (int)oil;
        else if (oil      string)
            Stairs - (string)oil;
        else if (oil ____ Hinge) {
            _____ vree › oil      _____
            Console.WeiteLine(vine.Table()
            + " " + _____.Bulb . " " · Stairs);
|
public          Hinge (
    public int Bulb:
    public Table Garden;
    public void SetlTable a) {
        Garden   a;
    |
    public string TableD
        return _____.Stairs:
```

Bonus points: Circle the lines where boxing happens.

Note: ...ch Ihing from **the** pool can **be used** mora **than** onc••

public
private
cla'i
new
abstract
Interlace

or
is
on
as
•••

Garden
Fl••r
Window
0 . .r
Hinge

•
••
=
==

struct
string
int
float
alngle
double

Answers on page 652.

there are no Dumb Questions

Q: Okay, back up a millule, Why do I care aboutthe stack?

A: Because understanding the difference teteeen the stack and Ille heap helps you keep your reereree types and value types straight. It's easy to forget that structs and objects work very differently—when you use the equals sign with both otttem they look really Slm ilar, Having some idea of how .NET and the CLR handle things under the hood helps you understand *why* rere-ence and vane types are different

Q: And boxing? Why ia that Important tome?

A: Because you need to know when things end up an the stack, and you need to know when data's being copied back and forth, Boxing takes extra memory and more time When you're only doing it a few times (or a few hundred limes) In your program, then you won't notice the difference But let's say you're writing a program that does the same thing over and over again, millions of bmes a second. That's not 100 far-fetched, since that's exactly what your beehive simulator did. If you fmd that your program s taking up more and more memory, or going slower and slower, Ille n it's possible that you can make it more efficient by avoiding boxing in the part of the program mat repeats

Q: I get how you get a fresh copy of a snuct when you set one struct variable equal to another ona. But why is that useful to me?

A: One place e ars really helpful is with encepsulatlcn Take a look at thiSfamiliar field from a crass that knows ns location:

```
private Point location;
public Point Location {
  get { return location; }
```

If Poi nt were a class, then this would be terrible encapsulahon, It wouldn't matter that location is private, because you made a publIC read-only property that returns a reference to it, so any other object would be able to access 1t.

Lucky for us, Point is adually a struct And that means that the public Location property returns a fresh copy of the point. The object that uses it can do whatever it wants to that oopy-none of those changes will make it to the private location field

Q: Il Point is a slruct, does that mean thtre are *other* etructs that I've been working with ali I lon; 1

A: vest One struct thats really useful and very common when you're working With graphiCS and lorms is Rectangle It's got some very useful methods that come in really handy when you need to figure out boundaries and check whether points are InSde or outside of the rectangle. All you need to do is set its location and size, and it'll automatically compute Its top, bottom, left, righl, width, and height

Another useful struct that you'll run eto is Size. You've already seen n in action-you used It when you were determining the size 01a string using the MeasureString() method. It's a suuct. too

Q: How do I know whether to use a struct of a class?

A: Most of the time, programmers use classes. Structs have a lot of limitations that can really make it hard 10 work with them lor large jobs. They don't support Inhentance, abstraction, or polymorphism, and you already know how important those things are for building programs easily

Where smcts come in really handy is If you have a small, limited type of data that you need to work with repeatedly. Rectangles and points are good examples—there's not much you'll do with them, but you'll use them over and ever again. Structs tend to be relatively small and limited in scope If you find that you have a small chunk 01 a lew different kinds of data that you want 10 store In a field In a class or pass 10 a method as a parameler, that's probably a good cancicate for a struct

A slruet can be very valuable when you wanl to add good encapsulation to your class. because a read-only property that returns a struet always makes a fresh copy of it.

← Pop quiz, hotshot! Answer's on page 646

Sharpen your pencil

Thi, method is supposed to kill a Clone object, but it doesn't wm k, Why not?

```
private void SetCloneToNull(Clone clone)
    clone = null;
```

Captain Amazing... Not So Much

With all this talk of boxing, you should have a pretty good idea of what was going on with the less-powerful, more-tired Captain Amazing. In fact, it wasn't Captain Amazing at all, but a boxed struct:



VS.



That's one big advantage of structs (and other value types)—you can easily make copies of the...

- Structs **can't** inherit from classes or **implement interfaces**
  No wonder the Captain's superpowers seemed a little weak! He didn't get any inherited behavior.

- **Structs can** only **live** on the **heap when they're boxed**
  The struct couldn't get onto the heap without being boxed up

- You can't **create a fresh copy** of an **object**
  When you have one object variable equal to another variable, you're copying a *reference* to the *same* variable.

- You can **use** the "as" **keyword with an object**
  Objects allow for polymorphism by allowing an object to function as any of the objects it inherits from.



[xtel1siol1 **methods** add l1ew behavior to **EXISTING** classes

Sometime you "",'d '0 extend a class ,hill you can't inl,,'rit li-""" like a sea1ed class (a I", "r ,1", .NET classe are sealed, so yc'll ,"an"' inherit fr"n tlu-ml. ,\nd (:# give you a powerful tool for that: **extension** nlethods_ When you add a class with "x,,"n,ioll llw,hod, ," your project, il **adds** n**ew** m**ethods** to clas-,**es** that already ,"xi,t. All you have lOll" i, create a static class, and add a static method that accepts an ;ll'l;llK"' or the class as its first parameter using ,he `this` keyword.

So 1"1" say you've gll an (Ordinary**H**,"n,1ll class:

```
public sealed class OrdinaryHuman {
    private int age;
    int weight;

    public Ordin"ryHuman(int weight){
        this.weight - weight;

    public void GoToWork()    // code to go to work
    public void PayBills()    // code to pay bills
```

The_Ordinaryttu,..an cla.. is sea ed, so it can't be subclasse[d] But what if we want to add a ··ethod to it?

You use an extension ..ethod by specifying the first para..eter using the "this" keyword

Since we want to extend the OrdinaryHuman class, we make the first para..eter this `OrdinaryH:man`

The SuperSolierSerum nH"thod adds an extension metho×l to (Ordinary**H**uman:

```
public static class SuperSoldierSerum (
    public static string BreakWalls(this OrdinaryHuman h. double vaLkDensity)
        return ("I broke through a wall of " • wallDensity + " density.");
```

Extension methods are always static ..ethod.. and they have to live i" static classes.

As soon as the SuperSoldierSerum class is ,,,I,led to the- project, (OrdinaryHuman gets a BreakWalls method. So now a form can use it:

```
prlvate vOld buttonl Clicklob)ect sender, EventArgs e) )
    OrdinaryHuman steve - new OrdinaryHwnan(lBSl;
    Console.WriteLine(steve.BreakWalls(89.2));
```

When the for.. creates an instance of the Ordinaryttuman c:::.. it can access the BreakWalls() method directly—as long as it has access to the SuperSoldierSeru.. class

Sharpen your pencil Solution

This method i, supposed to kill a Clone object. but it doesn't work, Why not?

```
private void SetCloneToNull(Clone clone) I
    clone . null;
```

So the clone parameter is just on the stack, so setting it to null doesn't do anything to the heap.

All this method does is set its own parameter to null, but that parameter's just a reference to a Clone. It's like sticking a label on an object and peeling it off again.

there are no Dumb Questions

Q: I'll m. again **why I wouldn't** add the n.w **methods I need** dir.ctly to **my class** cod., **instead** of **using extensions?**

A: You could do Illat, and you probably should If you're just talking about adding a method 10 one crass. Extension methods should be used pretty sparingly, and only in cases where you absolutely can't change the class you're working with for some reason (like it's part of the .NET Framework or another th,rd party). Where ecenecn methods really become powerful is when you need to extend the behavior of something you *wouldn't* ***normally have access*** *to*, like a type or an object that comes for free with the NET framework or another I,brary

Q: **Why use extension** methods al **all? Why** not just **extend** Ih. **class** with **inheritance?**

A: see. But then you'd need to use your custom objects, and any existing code that used the oog,nal, non-extended objects woold have to change. With eceoscn methods. you can change the tenavce of whole groups of objects, and even add functionality to some of the most base classes in ite ,NET Framework

Extending a class gives you new behavior, but requires that you use the new subclass if you want to use thai new behavior

Q: **Does my extension** method **affect** III **instances** 01 a **class**, or just a certain **instance** 01 the **class?**

A: Il WIII affect all meierces of a class that you extend In fact, once you've created an extension method, the new method will show up in your IDE alongside of the extended ctaes s normal methods

Oh, I **get** itl So you'd use **extension methods** to add **new behavior** to ere of t he built ·;n NET **Framework classes**, right?



Exactlyl There are some classes that you can't inherit from.

Pop open any project, add a class, and try typing this:

```
public class x : string I )
```

Try tu compile your code—the IDE will give vou an error. The reason i, that "'Iilt' .NET classes al" **seal**..d . which means Ihat you "'Ill't inherit I'"",, them. (You can do this with your own classes, I,o! Just add the sealed keyword til your class after the publi c access tn' odifier, and no other class will be allowed to inherit Inuu it.) Extensi'II methods give you a way «, extend it. ,","u if you can't inln-rit from it.

BU that's lIot all you can du with extension ",,'thods. III ,,,kl;I;",, lu extending classes, you can also extend int.. r**face**s , ,\ll you have 10 do is use an interface ",,,"" ill place of the class, after til,' thi s keyword ill th" extensi III method's fi',1 parameter. \\'hen you do, the r-xte-nvion method i, added to ..v..**ry class** that in.pl..n...n t**s** th**a**t int..rf**ace**. Remember th.u 1LINQ code you added 10 y nn' simulato in chapter 12? 1LINQ was built entirely with extension methods, extending the IEnumerable class. (Yc,u'li learn a 101 "'o'r about LINQ in Chapter 15.)

That's another thing you just can't do with inheritance—there's no way to inherit from an interface

Extel1dil1g a **fundamental type:** stril1g

Y(''' d"U'1 „11"1\ get ,,, chang(the l".ha,'i,,1'of a language's most fundamental types, like 'trim,". But with extension methods, you ,'all d(just that!



O Put all of your **extension** methods in a separate **namespace.**

It's a good idea t" keep all "r your extensions iH a different namespace than the n',1 "r your "„1e. That way, you won't have trouble finding th,'", li'r use in other programs. Set "I' a static class for your """thod t" live in, 10",

Using a separate namewaee is a good organizational tool

```
namespace Emergency |
  public static class HumanExtensions {
```

The class your extension method i. defined in ..ust be static.

O **Create the** static **extension method,** and defines its first **parameter** as `this` and **then the type** yau're **extending.**

The two main things you need 10 know when you declar(an extension method i, thai Ih,' method needs 1" be static and have the class it's extending as its first parameter.

"this string" says we're extending the string cla..,

```
public static bool IsDistressCal1 (this string s){
```

The extension method must be static, too

O Put **the** cede to evaluate **the** string in **the method.**

```
public static class HumanExtensions {
  public static bool IsDistressCall(this string s)|
    if (s.Contains("Help!"))
      return true;
    else
      return false;
```

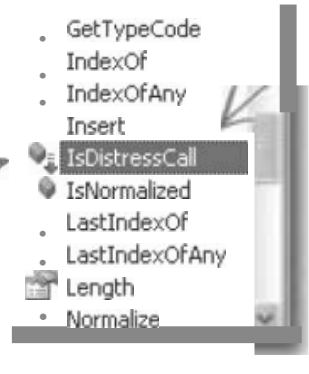
This checks the string for a certain value_ something definitely not in the default string class_

O **Create** a form and add a string.

Ad(Iusing Eme rgency; t,>y('Ul' cod(.. Now, when y(.u use a string, y(.u get the "x(,'u,i,'ll methods for free. You ,-,m ".,. thi, for yourself by typing the name or a 'Iri,ig variable and a period:

```
string message1;
message1 = "An army of clones is wreaking havoc at the factory. Help!";
message1.
```

A• •oon as you type U, dot, The IDE pops up a helper wind ith all of string' ethods... including your extension method



Comment t the using line and the extension ethod will disappear from the htelliSense w,ndow.

This toy example just shows you the syntax of extension methods. To get a real sense of how useful they are, just wait until **the** next chapter. It's all about LINQ, which is implemented entirely with extension methods•



Extension Magnets

Arrang.. the magn..ts to prootKt"thi, output:

```
a buck begets more bucks
```

```
namespace Upside {
  public static class Margin {
    public static void SendIt
```

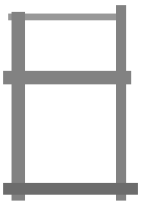
```
namespace Sideway. (
  u'ing Upside;
    public static cla.. Ticker I
```

```
public Itatic .trinq ToPrice
```

```
} } }
```

```
} } }
```

```
· .SendIt O
bool b • true;
Con.ole.Wrile(.);
public static .trinq Green
if (n _ 1)
    return "a buck "
.SendIt();
i = II
if (b _ true)
    return "be";
(this
b.Gr__nO;
else
    rllturn "gets";
b . fal.e;
el.e
    return " more bucks";
b GreenO;
1.
i.ToPrice()
(this bool bJ r r
int i • 1;
(thi int n) {
J public .tatic void Main ( |
q II • 1.ToPrice ()
```



Extension Magnets

Your job was 10 arrange the magnets 10 produc.. lhis output:

```
a buck begets more bucks
```

The Upside namespace has the extensions The Sideways namespace has the entry point

The Margin class extends string by adding a method called SendIt() that just writes the string to the console, and it extends int by adding a ...ethod called ToPrice() that returns "a buck" if the int' equal to I, or "...ore bucks" if it's not.

The entry point method uses the extension that you added ''' the Margin class

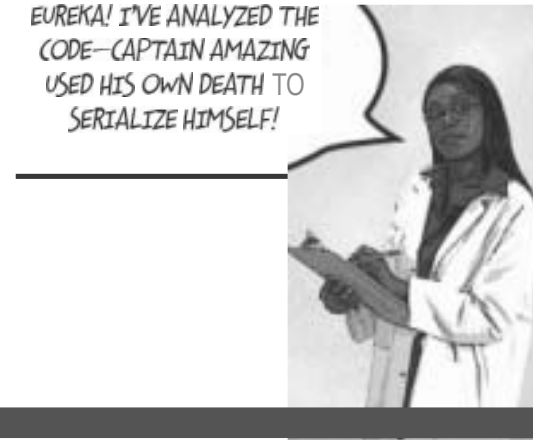
```
namespace Upside {
  [ public static ..:1••• Margin {
    -, public static void SendIt (this string_ I , |
      Console.Write(s);
    ' |
    public static string ToPrice (this int nl {
      if Cn __11
        return -. buck ";
      .1s.
        return - more bucks";
    }
    public static string Green (this bool b) {
      if (b _ true)
        re urn "be";
      . ls.
        return "gets";
    ,
  }
```

```
namespace Sideways ,
  using Upside;
  public static cla.. Ticker { t
    public static void Main (
      int i = 1;
      string s _ i.ToPrice()
      s.SendIt()
      bool b - true;
      b.Green();
      b _ t.1_ ;
      b.Green();
      i _ 3
      i.ToPrice() .SendIt();
    }
  }
r
```

The Green ...ethod extends a bool. it returns the string "be" if the bool is true, and "gets" if it's false

Here's where the Margin class extends bool by adding a Green() method to it If the bool is true, Green returns "be", otherwise it returns "gets"





TheUNIVERSE

CAPTAIN AMAZING REBORN

Death was not the end!

By Bucky Barnes
UNIVERSE STAFF WRITER

(OBJECTVILLE

Captain Amazing des....ializ..s him"elf, mak··s stunning comeback

T" a stunning t···rn or events. Captain .\"'azing has returned to (Object ill,". month. Captain Amazing's coffin was found empty, and only a strange ,h, Last left where hi body .ho"ld have been. .Analysis 01 the note revealed Captain Amazing's object DN.\ all his last fields and values, captured faithfully in binary format.

Today, that data has sprung 10 life. The Captain i, back, deserialized from hi., own brilliant note. When asked how h, conceived of such a plan, Captain merely shrugged and mumbled. "Chapter 9." Sources close 10 the Captain refused 10"o""n"1l1 n" the meaning "r his cryptic reply, but did admit that prior T" his failed assault on Swindler. the Captain has spent a 101 nr time reading books. studying Dispose methods and persistence. \We expect Capta!'" Amazing.

".'",. AMAZING 0" A-5



Captain **Amazing** is **back!**

Pool puzzle Solution



The Lamp() method sets the various strings and ints. If you call it with an int, then it sets the Bulb field in whatever object Hinge points to.

Output when you create a new Faucet object:

```
back 1n 20 minutes
```

```
public class Faucet
    pUblic Faucet()
        Table wine   new Table(j;
        Hinge book   new Hinge();
        wine.Set(book);
        book.Set(wine);
        ( wine.LampUOI;
C       book.Garden.Lamp("back in");
        book.Bulb *= 2;
        wine.Lamp("minutes");
        wine.Lamp(book);          )
```

Here's why Table has to be a struct. If it were a class, then wine would point to the same object as book.Garden. which would cause this to overwrite the "back ..." string

Bonus question: Circle the line. where boxing happens.

Since the Lamp() method takes an object parameter, boxing automatically happens when it's passed an int or a string

```
public struct Table I
    public string Stairs;
    public Hinge Floor;
    public void Set(Hinge bl {
        Floor = b:

    public void Lamp(object oil) l
        if (oil is int)
            FIoor.Bulb " (int)oil;
        else it (oil is string)
            Stairs. (string)oil;
        else it (oil is Hinge) r
            Hinge vine .. oil as Hinge;
            Console.WriteLine(vine.Table()
            + " " + Floor.Bulb + " " + Stairs);
```

If you pass a string to Lamp it sets the Stairs field to ...hatever is in that string

Remember, the as keyword only works with classes, not structs

```
public class Hinge I
    public int Bulb:
    public Table Garden;
    public void Set(Table a) (
        Garden = a,

    public string Teble() {
        return Garden.Stairs;
```

Both Hinge and Table have a Set() method. Hinge's Set() sets its Table field called Garden, and Table's Set() method sets its Hinge field called Floor.

15 LINQ

Get control of your data

So if you take the first word trom this article, and the second word in that list, and odd it fa the fifth word over here... you get secret messages from the government!



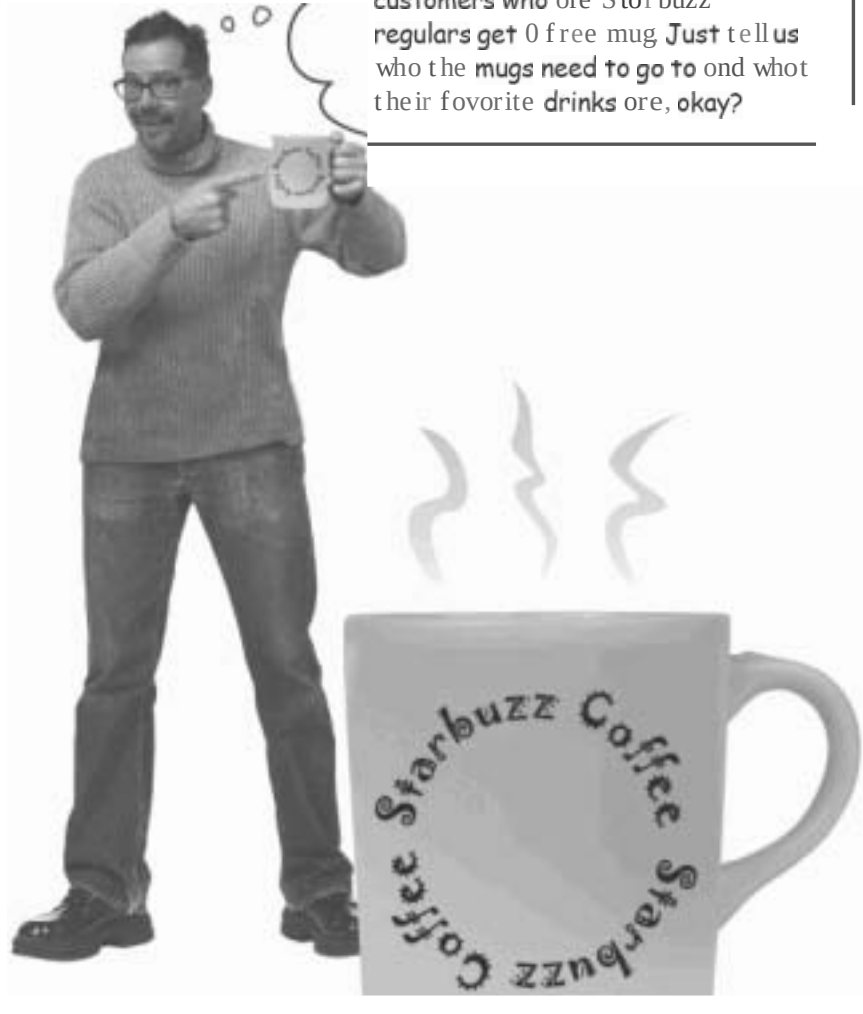
It's a data-driven world... you better know how to **live** in it.

Gone are the days when you could program for days, even weeks, Without dealing with loads of data. Bulloday, ***everything is about data.*** In fact, you'll ottee have to work with data from more Ihan one ptace.. and in more than one formal. Databases, XML, collections from other programs... It'S all part of the job of a good C# programmer. And that's where L.INQ comes in. UNO not only lets you query data In a simple, Intuitive way, but It lets you group data, and merge data **from** different data sources

An easy project...

Objectville Paper Company wants (" d" a cross-promotio›ll with Starbuzz Coffee Starbuzz has a frequent customer program where they kn..w who buys which drink ,,,,d how often they buy it, Objectville Paper wants L.. Iig',,..' o"1 **which** of Ih..ir cuslom...**rs ar**.. **also** Slarbuzz r..gular**s** and send the,,, a fn',- mug and a coupon foı their favorite "..ffee drink... and it's up 10you 10"<>Iuhin,- the data and generate the list or cust.."',"|" 10,,'ud mugs and coupons to.

All of Objectville Poper's customers who ore Storbuzz regulars get 0 free mug. Just tell us who the mugs need to go to ond whot their fovorite drinks ore, okay?

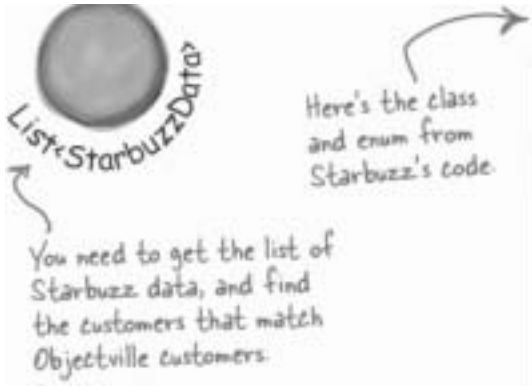


...but the data's all over the place

Starbuzz keeps an their data in classes, grouped together in a big List. But the Objectville data is in a database (from way back in Chapter 1). We want to find any Starbuzz customers who spent more than $90, match them (<I Ih" Objectv-ill,' Paper contact li,t, "lid Ill"ke a final li'l of people: **we want** each p..**rson'**.. nan..', the conlpany they work for, and Ih" ir favorite Starbuzz drink

The Starbuzz data's in a List<>

The Starbuzz people provided a program that connects to Ihe;r website and pulls "IIIhe data ;n1<> a Lts t. of StarbuzzData class.




```
public class StarbuzzData

    public string Name { get; set; )
    public Drink FavoriteDrink { get; set;
    public int MoneySpent ( get; Set;
    pUblic int Visits { get; set; )


public enum Drink (
    BoringCotfee,
    ChocoRockoLatte,
    TripleEspresso,
    ZestyLemonChai,
    DoubleCappuccino,
    HalfCafAmerieana,
    ChocoMacchiato,
    BananaSplitlnACup,
```

You've already got the customer data

You hu ilt th,' ( )hj,-,'I,-ill,-Paper Company contact list back in (:h" I'{,'r I it's got part of llt,- data you need.



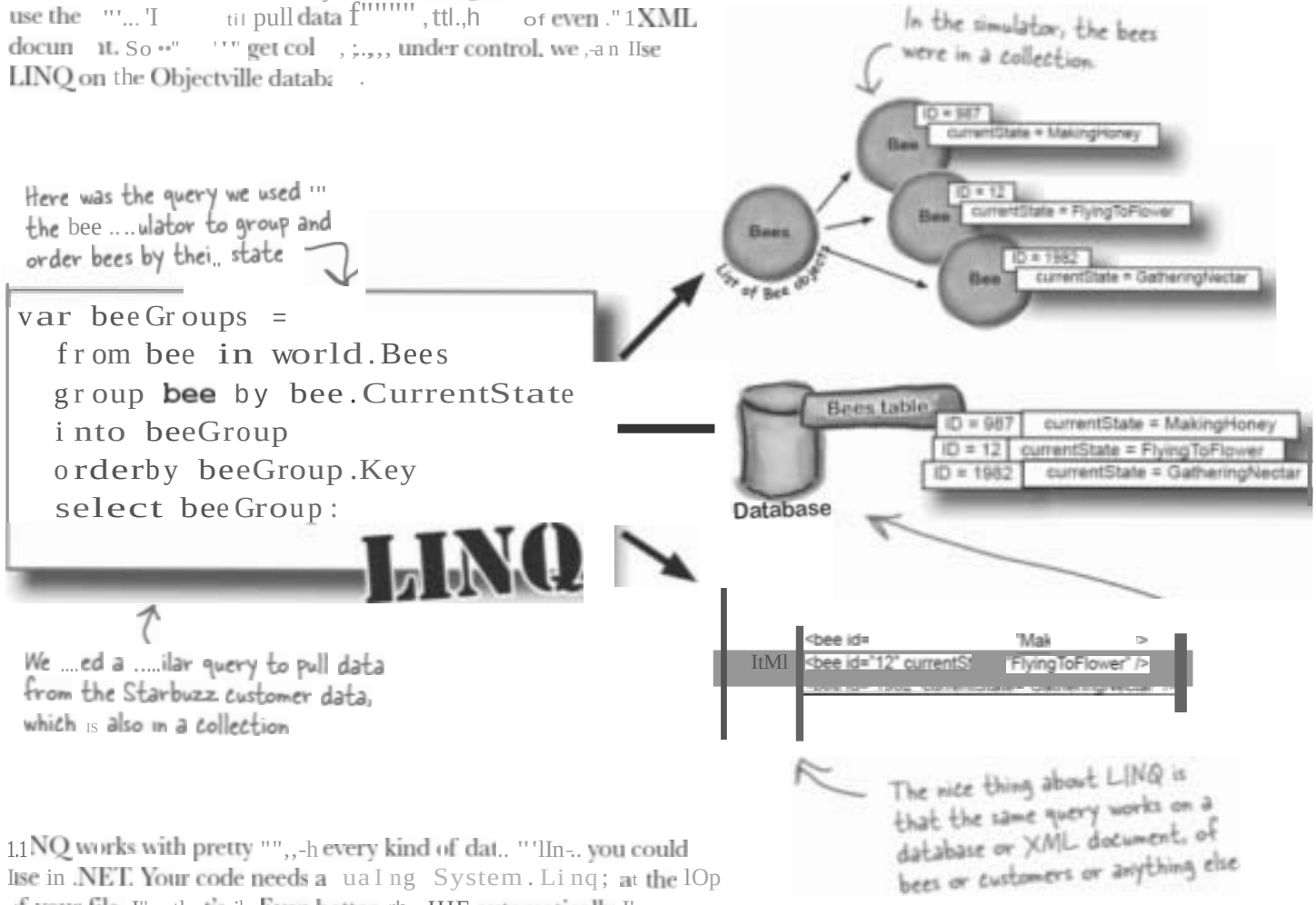


How would you combine the data from Starbuzz and Objectville paper to get a complete contact list?

LINQ can pull data from multiple sources

You 1LINQ in Ih.. Hiv.. Simu!al In tr.. k wh•• groups "r Bees w loing. Y.., ook advantage of d power •f LINQ Ihere t. "Tite ,i",p"' queries 'u I'ull data out uf a llection. IJ Q can work with tilt· Starbuzz data just lik.. il worked with 110.. I helping you use queries 'n pull ,..it .-,ust", data. .\s long a collection implements the `IEnumerable` interfa you ran ,se LINQ queries "ith it.

But LINQ a I ts you work wjrh more than just llections. You ""11 use the "'... 'I til pull data f""""" ,ttl.,h of even ." 1XML docun it. So •• '"' get col , ;.,,, under control. we ,-a n IIse LINQ on the Objectville databa .



1.1NQ works with pretty "",,-h every kind of dat.. "'lIn-. you could Ilse in .NET. Your code needs a uaIng System.Linq; at the lOp •f your file, I" n that's il_ Even better. rh.. IHE automatically I'ur, a reference r" LINQ in Ih.' header ..r any code Iiles th.•t are created ill \ ·"".,1Studio 2008. So if you re using \ "isual Studio 2008 or 1.11" ', just start .'.ding.• ""t LINQ is available r" you.

.NE! collectiol1s are already **set** up for **LINQ**

.\ll of the collection type in .NET implement the IEnumerable interface Type System.Collections.Oeneric.IEnumerable inl" your IIIE window, right-click "n the line, ,,,,d selcc1(;" To !ld inili,,". Y" u'llsee that Ih, IEnumerable lll1'd.", dc fines a GetEnumerator() """Ih"d

All collections and arrays implement IEnumerable Many of the.. do it by plementing IEnumerable<T>, which inherits from IEnumerable

This T ..eans that IEnumerable will work with any object or type

```
namespace System.CoLj.ect.Lons.CenerLc {
  public interface IEnumerable<T> : IEnumerable {
    // Summary:
    //     Returns an enumerator that iterates through the collection.
    //
    // Returns:
    //     A System.Collections.Generic.IEnumerator<T> that Cdn be
    //       used to iterate through the collection.
    IEnumerator<T> GetEnumerator();
```

This is the only method in the interface. Each collection implements this method. You could create your own kind of collection that implemented IEnumerable too... And if you did, you could use LINQ with your collection.

This method requires collections I" define a way I" move through Ih.. ,:" II"-li,," . one element at a Ii",,', That's all LINQ require as a prerequisite. If you can move through a list of data, item-by-item, LINQ can query Ih,' ,'OII" "li,)n.

Behind the Scenes



LINQ uses ex...**nsion In".hods** to let von query, sort, and update data. Ch,', k il oul for yourself. Create an Int array called IInqtest, 1"'1 some ""IllI"'r' in the array, and then type thi, Ii"" of code (don't worry, you'll learn what il doe in a miIHltr);

```
var result = from i in linqtest where i < ] select i;
```

Now comment "ut the using System.Ling; Ii"" "I' ill ,I", header Of the Iii,' you've created. When you try to rebuild th" ",l"li"", you'll see' that this line rkoesn't compile anymore. The methods you"r- calling when yc"I use 1.1NQ arr .i"'1 extensi,n methods that are being used I" extend thr- arrav.

Now you can see why extension methods were so important in Chapter 14... they let .NET (and you) add all kinds of cool behavior to existing types.

LINQ l\lakes queries easy

Here's a simple example of LINQ syntax. II selects all the numbers in an int array tha are under :17 and puts lh.",· number in ascending ",drr. II does that using four ..lau..... rh•1 tell it what collection I" 'Iu.ry. what criteria h. u 10d..termine which members of ll,,· collection '0 selc.:t, how ,,, sort ,lue , lts, ;'lltl how the ..:sults should be returned.

```
int[] values = new int[] {0, 12, 44, 36, 92, 54, 13, 8};

var result = from v in values
             where v < 37
             orderby v
             select v;

foreach(int i in result)
  Console.WriteLine(i);
```

This assigns the letter "v" to stand in for each of the values array in the query. So v is 0, then 12, then 44, then 36... etc.

This LINQ query has four clauses the from clause, a where clause, an orderby clause, and the select clause

This says, select each v in the array that is less than 37.

Then put those values in order (lowest to highest).

If you've used SQL before, it may seem weird to put the select at the end, but that's how things work in LINQ.

Now you can iterate through the results array and print out each item in the LINQ result.

Output:

```
0, 8, 12, 13, 36
```

var

var is a keyword that tells the compiler to figure out the type of a variable at compilation time .NET detects the type from the type of the local variable that you're using LINQ to query. When you build your sol..tion the compiler ...ill replace var with the right type for the data you're working with

In the exa...ple above, when this [ine is compiled

```
var result = from v in values
```

The compiler replaces "var" with this:

```
IEnumerable<int>
```



Watch it!

LlNQ **is** a new feature that's part of C# 3.0 and Visual Studio 2008.

If you're using an earlier version of C#, take a few minutes to download and install Visual C# 2008 Express Edition. It's free from Microsoft, and it can be installed alongside previous versions.

LINQ is simple, but your queries don't have to be

Jimmy jll'! sold his start-up company 10 a big investor, and wants 10 take some of his profits "'ld buy the m, "I expensive issues ,1' Capta ill Amazing that he can lind. 11m all he's g,t is data. II,w can LINQ help him scour his two collections and figure out which t"" ",i," are the most expensive?

O Jimmy rkownloaded a Ii'l „1' (:apta in Amazing issues fr om a Captain ·Amazing fan page H" 1'111 them in a Li s t <> of Comic objects that have two fields Na me and Issue,

```
public class Comic {
    public string Name { get: set;
    pubLic int Issue { get; set; }
```

Jimmy used object initializers and a collection initializer to build his catalog:

```
private static List<Comic> BuildCatalog()
    Listccornic> comics = new List<Comic>():
    comics.Add(new Comic("Johnny America vs. the Pinko", 6)):
    comics.Add(new Comic("Rock and Roll (limited edition)", 19»:
    comics.Add(new Comic("Woman's Work", 36)):
    comics.Add(new Comic("Hippie Madness (misprinted)", 51)):
    comics.Add(new Comic("Revenge of the New Wave Freak (damaged)", 6Sll:
    comics.Add(new Comic("Black Monday", 74));
    comics.Add(new Comic("Tribal Tattoo Madness", 83»:
    comics.Add(new Comic("The Death of an Object«, 97));
    return comics;
```

Issue #74 of Captain Amazing is called "Black Monday"

O Luckily, there's a tlll'i,-;"\(marketplace for Captain Amazing c(„„ics 0" (Greg' I.i,t, I He knows that issue #57, "Hippie Madness," was misprint..d and II,,' alm(" I all <of the run was destroyed by the publisher, and he li,„„d a rare copy recently sold on Greg's List fi". $13,52! . .\h"f a few lk<mr' of searching, jmunv was able 10 hlllild a Diet i*anar*y<> Ihat mapped issue numbe-rs 10 values

```
private static Dictionary<int, int> GetPrices!) {
    pfcticnaryctnt, int> values = new Dfctionarycint, int>():
    values.Add(6, 3600):
    values.Add(19, 500):
    values.Add(36, 6501;
    values.Add(57, 13525):
    values.Add(6S, 250);
    values.Add(74, /51;
    values.Add(83, 251:
    values.Add(97, 351;
    return values:
```

Issue #57 is worth $13,525.



Look closely at the LINQ query on page 658, What do you think Jim has to put in his query to find the most expensive issues?



Anatomy of a query

```
Dictionary<int,

        where values[comic.Issue] > 500
        orderby values[comic.Issue] descending
        select comic;
```

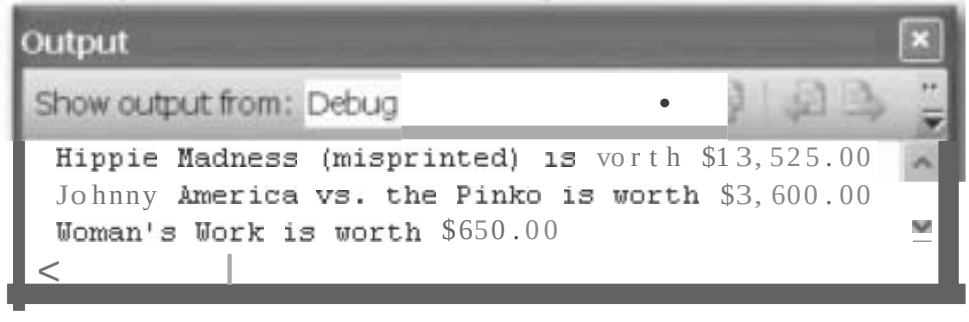
The name comic was defined in the from clause specifically so it could be used in the where and orderby clauses

"comic".

```
foreach (Comic comic in mostExpensive)
    Console.WriteLine("{0} is worth {1:c)",
                      comic.Name, values[comic.Issue]);
```

When you add "{1c}" to the WriteLine output, that tells it to print the second parameter in the local currency format

The query returned its results into a collection called mostExpensive. The select clause determines what goes into the results—since it selected comic, the query returned Comic objects.



Don't worry if you've never used SQL—you don't need to know anything about it to work with LINQ. But if you're curious, check out "Head First SQL."

I don't buy this, I know SQL already—isn't writing a LINQ query just like writing SQL?



UNQ may *look* like SQL, but it doesn't *work* like SQL.

If you've done a I", of work with SQL, it may be tempting to dismiss all this LINQ stuff a intuitive and "l,,'jou,- and you wouldn', Be alone, because a I", of developers make that mistake. It's Irue that LINQ uses Ih,' `select. from, where, ascending, ;ll,d join` keywords, " hid, are hornowed from SQI·. But 1.1NQ is very different from SQL, and if ya'" try to 'hiuk about LINQ the way yc11, lhink abcIIn SQL you'll "11d Ill' with code that **doesn't do what you** expect

()ne big difference between the two is Ulf.« SQL operate (,n ***tables***. " hid, are very different from ***collections***. \When you ,',X,-"Me a SQL `select` against a 'ahl,-. you can h.. sure that Ih.. lah l,- is 11„1 going H' b,- updated. SQL has all sorts of huilt-in data security that you can trust. ,\nd SQL querie are set operations, which means they don'l examine Ih,' rows in Ih,' table in any predictable "I'der, .\ collection, "" the olh..r h;md, can store *anything*—values, structs, "hj..cts, anything—and "Oll,-"Ii"", have a specific order. (,\ table's rows aren't in any particular ,>nJ,-r until you make a SQL query that orders the-m: items in,id,- a 1.i' I, "'1 the »ther hand, are in order.) And LINQ lets \"'∪ perform any operation that's supported by whate,'er happen 1' be ill the ,·"Il,>c·ti" ,, - it can even call """Ihod, "" the "hj,-u; in ,h,' collection. .And LINQ loops through the (·"Il,-,·ti' ,n, which means that it d,)es its operations ill a sp..cific order. That may "ul seem all that important, but il you're used 10dealing with SQL, it means yc.ur 1.1NQ querie will surprise \', .u if you '·Xp;c1 Ihen" IU act like SQL.

There are a lot of other differences between LINQ and SQL too, but you don't need to understand the... i" order to work with LINQ successfully. Just approach it with a, open "',,,d, and don't expect it to work the way SQL works.

LINQ is versatile

You can do a 101 more than just 1',,11 a few items oul of a collecti'no You can modify the items before you n'lUrn them. And "nce you'vr-generated a set "f result collections, LINQ gives you a bunch ,,1' methods rh;rr work with lhem. Top r" bottom, LINQ gives you the tools you need to manage your data.

O Modify **every item returned** from **the query**

'111i, code will add a string ollr" the end ,,1' each string inan array. IIdoesn't change the array ;r" '1'– il **creates a new coll..ction** ,,1' modified strings.

```
string[] sandwiches - { "ham and cheese", "salami with mayo",
                        "t.urkey and swiss", "chicken cutlet" );
val' sandwichesOnRye
    from sandwich in senovtcnes
    select sandwich + " on rye";

foreach (var sandwich in sandwichesOnRyel
    Console.WriteLine(sandwich);
```

This adds the string " on rye" to every item in the results from the query

This change is made to the ite... in the results of your query.., but not to the item in the original collection or database

-e

Notice that all the ite....eturned have " on rye" added to the end

Output:

```
ham and cheese on rye
salami with mayo on rye
turkey and swiss on rye
chicken cutlet on rye
```

O **Perform** calculations on collections

Remember. we said LINQ adds new methods r" your collections (and database access objects ... and some "f those are pretty handy 'n, rhpi, own, wirlH"u actual requiring a query:

```
List<int> listOfNumbers - new List<int>();
int length = random.Next(50, 150);
for (int i = 0; i < length; i++)
    listOfNumbers.Add(random.Next(100));

Console vxriteLine("There are IO) numbers",
                   1istOfNumbers.Count());
Console.WriteLine("The smallest is {O}",
                   listOfNumbers.Min());
Console,WriteLine e-rne biggest is {0)".
                   listOfNumbers.Max
Console.WriteLine e-rne sum is {0}",
                   () );
                   listOfNumbers.Sum());
Console.WriteLine("The average is {O:F2}",
                   listOfNumbers.Average(» ;
```

As long as you're using C# 3.0 (which is part of Visual Studio 2008), any collection you create has LINQ capabilities.

None of these ··ethods are part of the .NET collections classes... they're all defined by LINQ.

These are all extension methods defined in the Enumerable class in the System.Linq namespace



Watch it!

UNO queries aren't run until you access their resultsl

It's called "lazy evaluation"-the UNQ query aoesnt actually do any looping until a statement is executed that uses the results of the query. That's why ToListO is important: it tells LINQ to evaluate the query immediately.

O Store **all** or **part** of **your results in a** new **collection**

Sn""time you'll want h' kee1' your results from a LINQ query around. Y(" can use the `ToList()` command 10do just that:

```
var under50sorted -
  from number in listOfNumbers
  where number -: 50
  orderby number descending
  select number;

List<int> newList - under50sorted.ToList():
```

This ti..e, we'..e sorting a list of numbers descending, from highest to lowest.

ToList() converts a LINQ var into a list object, so you can keep results of a query around. There's also ToArray() and ToDictionary() methods, which do just what you'd expect.

You ,'an even take just a subset of the results, using the `Take()` method:

```
var firstFive = under50sorted.Take(6);

List<int> shortList = firstFive.ToList();
foreach (int n in shortList)
    Console.WriteLine(n);
```

Take() pulls out the supplied number of items, from the first of the results from a LINQ query. You can put these into another var, and then convert that into a list.

O Check out **Microsoft's official "101** LINQ **Samples" page**

'There's way more that 1.1NQ can do. Luckily, Micros'h gave you a great little reference In h"lp y(" alom:.

http://msdn2.microsoft.com/en-us/vcsharp/aa336746.aspx

there are nc Dumb Questions

Q: That's a lot of new keyword__from, where, orderby, select,.. **it's** like a whole different language. Why does **it** look ao different Irom the **rest** of **C#?**

A: seceuse It serves a different purpose Most of the C# syntax was built todo one small operation or calculation at a nrre. You can start a loop, or set a variable, or do a mathematical operation, or call a mencc.. those are all single operations

UNO quenes look dlffere nl because a single LINQ query usually does a whole bunch of things at once, ters take a ecser look at a straightforward query

```
var under10 -
  from number in numberArray
  where number < 10
  select number;
```

Il looks really simple—not a lot of stuff there right? BUI this is actually a pretty complex piece of code. Think about what's got to happen for the program to actually select all the numbers from number Array œ t are less than 10, First, you need to loop through the entire array Then. each number is compared to 10 Then mcee results need to be gathered together so your cooe can use them.

And that's why UNO looks a little odd because C# has to cram a whole lot of behavior inlo a very small space

LINQ **lets** you **write queries** Il,al **do very** complex **things** using **very** little code.

BULLET POINTS

- `from` is how you specify the collection that you're querying. It's always followed by the name "r a variable, followed by `in` and the name of the collection (`from value in values`).
- `where` generally follows Ihe `from` clause. That's where you use normal C# conditions to tell LINQ which items '" pill! out of lh,. collection (`where value < 10`).
- `orderby` lets you order Ih" results. It's followed by the criteria that you're using 1o sort them, and optionally `descending` 10 ",II ;1 '" reverse the sort (`orderby veLue descending`),
- `select` is how you specify what goe into the results (`select va lue`).
- Take lets you pullthe first items 011 1of the results or a LINQ query (`resuIt s •'rake (10)`). LINQ give vou other methods fo>' each collection: `Mi n I}. Max (}. Sum n. ,tIld Ave r age ()`
- You can `s e I ect` anything - you're uol Iimil"d I" selecting the nann- Ihal ,"u created in the `from` clause. Here's an example: if vour LINQ query pulls a set of 1'1';ces 0111 or all array of `int` value and name them `value` in the `from`clause, you can return a r"ll,x·lion of price strings like this: `select String.Format("{uicj", value.`

This is just like the r0o,,} you used in Chapter 9 when you built the hex dumper. The-e's also {O,d} and (OD) for short and I g dates, and (O:P) or {OPn} to print a percent (ith n decimal places).

there are no Dumb Questions

0_: How does the `from` clause work?

A: It's a lot like the first line ofa `foreach` loop One thingthat makes thinking atcct lINO queries a little tricky is that you're nol just doing one ooeranco Most C# statements just doone single thing,

A L1NO query does the same th ing over and over again for each Ilem in a collection. The `from` clause doestwo things: it tells L1NQ which collection to use for the query, and it assigns a name 10 use foreach member of the eeaecton that's being queried

The way the `from`clause creates a new name for each Itemin the collection IS really similar to how a `foreach` loop does it Here's the first line ofa `foreach` loop

```
foreach (int i in values)
```

Thai foreach loop temporarily creates a variable called `i`. which it assigns sequentially to each ilem in the `va1ues` collection. Now look at a `from` clause in a UNO query on ue same collection

```
from i in values
```

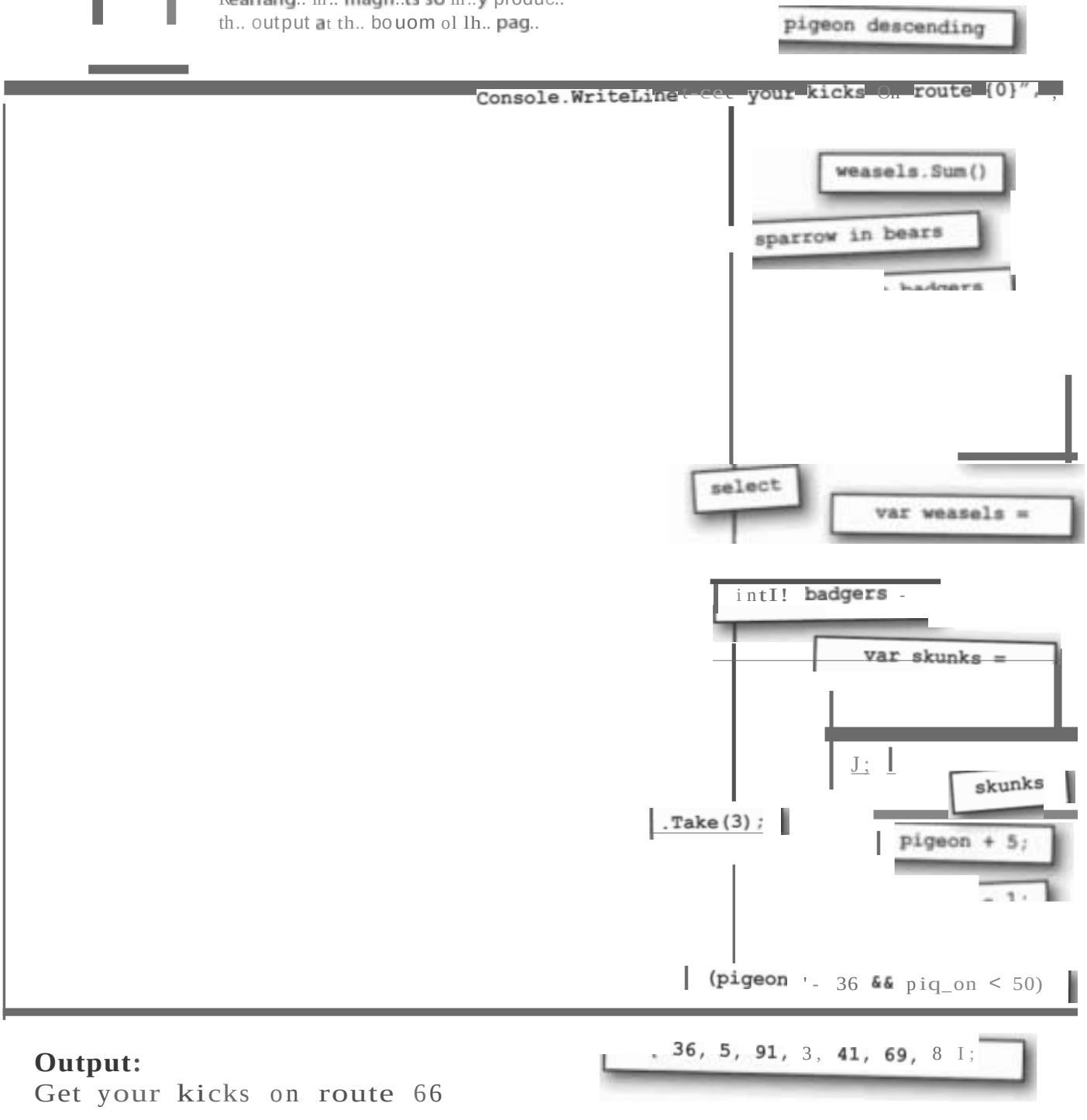
Thai clause does pretty much the same thing Il creates a temporary variable ceuec i and assigns it sequentially to each item in the `values` couectco. The `foreach` loop runs thesame block of code for each Itemin the collection, while the UNO query applies the same cetene in the `vhere` clause to each item in the collection to determine whether or nollo include it in the resuns.

Q: How does UNO decide what goes inlo the results?

A: That's what the `select` clause IS for Every L1NO query returns a collection, and every Ilem in a collection is of tte same type It tells L1NO exactly what that collection should contain. When you're querying an array or list Of a single type—like an array of `int`s or a `List<string>`—it's obvious what goes into the `select` clause. But what if you're selecting from a list of Comic objects? You could do what Jimmy did and seecure whole ass. But you could also change the last line ofthe query to `select comic.Name` lo tell illo retom a collecbon of `string`s, Or you could do `select comic.Issue` andhave it return a conectco of `int`s

LINQ Magnets

Rearrang.. lh.. magn..ts so lh..y produc.. th.. output at th.. bouom ol lh.. pag..



Output:

Get your kicks on route 66



LINQ Magnets Solution

Rearrang.. the magnets so they produc..
the output at the botlom of the page.

LINQ starts with some sort of collection or array—in this case, an array of integers.

```
int[] badgers =  { 36, 5, 91, 3, 41, 69, 8 };
```

"from pigeon in badgers" makes for a good puzzle, but an unreadable LINQ query. "from badger in badgers" is more readable.

```
var skunks =
from pigeon in badgers
"h.r. (pigeon !. 36 && piq..on < 50)
orderby pi9. on d". "",ndinq
select piq. on + 5;
```

After this statement, skunks contains four numbers: 46, 13, 10 a"d 8

This LINQ statement pulls all the numbers that are belo 50 and not equal to 36 t of the array, adds 5 to each of the..., ts the... from biggest to llest, and puts the... in a ne collection called skunks

```
var bears =
skunks .Take(3);
```

After this statement, bears contains three numbers: 46, 13 and 10

Here's where we take the first three numbers in skunks and put them into a new collection called bears.

```
var wea_ 1- -
fr om sparrow in bears
..lect sparrow - i -
```

After this statement, weasels contains four numbers: 45, 12, and 9

This statement just subtracts 1 from each number in bears and puts them all into weasels.

```
Con.ol- .WriteLine("Get your kicks on route "OIN.
```

45 + 12 + 9 = 66

The —"bers in weasels add up to 66.

Output:

Get your kicks on route 66

LINQ can cOl\lbllle your results Illto groups

You already kn"·. Ih." you can us" I.INQ ". Ibuild your result• in'o groups, because that's "h.I' ,,,. did "iIh the beehi ,illl"I.I"'. Let's take a closer look at that query d,KI re h,,, it works.

```
var beeGroups =
    from bee in world.Bees
    group bee by bee.CUrrentstate
    into beeGroup
    orderby beeGroup.Key
    select beeGroup;
```

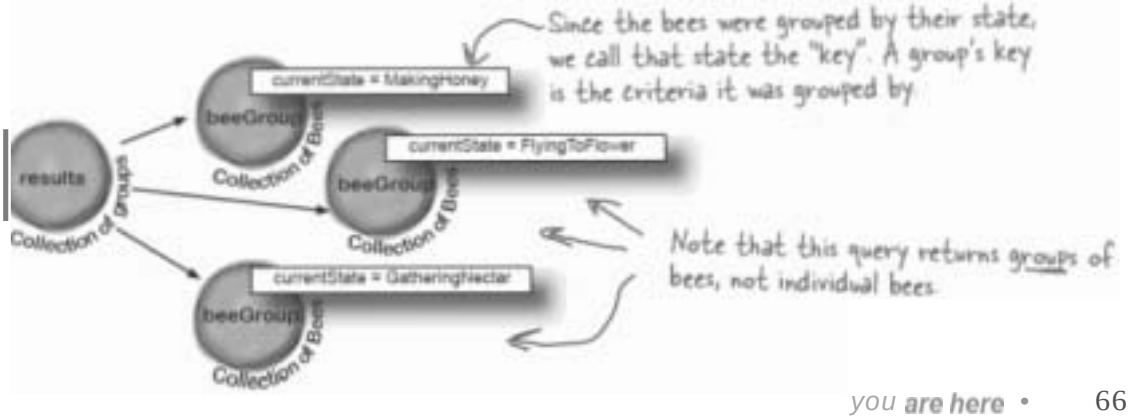
1 The qu....ry starts out jll" like the othe queries you've seen)- by pulling individual bee objects out "f ,h,· worId.Bees collection, a t.iat-csee> object.

2 The next line in the query ha a ,..w keyword: **group**. This tells tl... query to return ***groups*** of bees. What that means is that rather than returning one single .-"Ilection, tl... query will relUrn a **collection** or **collections** **group bee** by **bee.CurrentState** tells LINQ ,n return one group f h unique CurrentState property ,h i. finds in the bees that il selec-,s. Finally, w.. need to giv.. LINQ a name for the group. That's what the next line is : Into **beeGroup** ys Ih the name .1bee(; "",- refers .,. the I gr.....s.

3 N we've got groups, we can manipulate them. Since w turning a collectiou "I' groups, w.. can use .t orderby keyword to put the groups in order of the CurrentState "IIIIIT values (Idle, FlyingToFlower. ..1.-): **orderby beeGroup.Key** tells Ihr query ." I"" ,hr ,-,,11ection of groups in order, sorting them by the group key. Sin,r we grouq 1the 1bees bv their CurrentState. ,hat's " h.ll I ing used as a kev.

4 N " ,. jn" have ,,, use the select key ,•d ,,, indic tl,' what's l..ing returned by the query. Sinc we're returning grouq.... we select the grouq name:

```
•elect beeGroup;
```



Combine Jimmy's values illto groups

Jimmy buys a 101 of cheap comic books, some midrange comic books, and a few expensive ones, and he wants 10 know what hi, opti"n, are beft're he decides what comics to buy. He's got tho,,' price he got from Greg's List and put illto a Dictionary<int, int> using his GetPrices() "ethod - let's use LINQ to group them iuto three groups: '"It' for cheap comics thai':ost under $100I,"IIC for midrange comics th"t cost between $100 and $1000, ami expensive ones thai cost over $1000. We'll create a PriceRange ""lin th,II we'll u,e as the key for the groups, and a method called EvaluatePrice() lhat'll evaluate a price alld return a Priceganqe.

Every group needs a key-we'll use an enum for that

Th" group's key is the thing lha, all "I' its members have iu '·"""lllOl1. The key can be anything: a string, a number, even an "hj"'-l reference. \Ve'II be looking at the price thal .lillullY got from (Jreg' li,t. Each group that the query n-ntrns will be a collection of issue numl)ers and the group's key will be a PriceRange '·I1\1l1l. ,\nd the EvaluatePrice() method tak," a price ;,., a parameter ami returns a PriceRange:

```
public enum PriceRange { cheap, midrange, expensive }

public PriceRange EvaluatePrice(int price)
    if (price < 100) return PriceRange.cheap;
    else if (price < 1000) return PriceRange.midrange;
    else return PriceRange.expensive;
```

Now we CQn group the comics by their price CQtegories

The LINQ query returns a **coll..ction of collections**. Each of the collections inside II", results has a Key property, which matches the PriceRange Ihal was returned by EvaluatePrice(). Look closely al the group by clause- we"n- pulling pairs 0111 of the Dictionary, aud using the narm' pe Lr for each of them: pair.Key is Ih,' issue number, and pair.Value is the price from Greg's list. Adding group pair.Key tdl, LINQ to create groups of issue numbers, and thell 1)undles all of thm,· groups up based 0" the price category that's returned by EvaluatePrice():

```
Dictionary<int, int> values = GetPrices();

    priceGroups
    from pair in values
    group pair.Key by EvaluatePrice(pair.Value)
        into priceGroup
        orderby priceGroup.Key descending
        select priceGroup;

roreecn tver group in priceGroups)
    Console.Write("I found {Ol {1} comics: issues    group.Count(), group,Key):
    foreach (var price in group)
        Console.Write(price.ToStrin() + .. ");
    Console.WriteLine();
```

The query figures out which group a particular price belongs to by sending its price to EvaluatePrice(). That returns a PriceRange enum which it uses as the grou\", key

Each of the groups is a collection so we added an inner foreach loop to pull each of the prices out of the group.

```
Output
Show ou    from: Debug
I found 2 expensive comics: issues 6 57
I found 3 midrange comics: issues 19 36 68
I found 3 cheap comics: issues 74 83 97
```

Pool puzzle



Your job is to take snippets from the pool and place them into the blank lines in the program. You can use the same snippet more than once, and you won't need to use all the snippets. Your goal is to make the program produce this output:

```
Horses enjoy eating carrots, but they love eating apples.
```

```
public class Line {
    public string[] Words;
    public int Value;
    public Line(string[] Words, int Value) {
        this.Words = Words; this.Value = Value;
```

Hint: LINQ sorts strings in alphabetical order

```
Line[] lines = {
  new Line { "eating", "carrots,",
        "but", "enjoy", "horses", 1 },
  new Line { "zebras", "hay",
        "Cows", "Bridge.", "bolted", 21 },
  new Line { "tork", "dogs!",
        "Engine", "and", 3 },
  new Line { "Love", "they",
        "Wives", "eating" }, 2 ),
  new Line { "whistled.", "Bump" }, 1 ) };
```

```
var _____
    from _____ in _____
    _____ line by line. _____
    into wordGroups
    orderby _____
    select _____

_____ words. _____ (2);

foreach (var group in twoGroups)

    int i = 0;

    foreach _____ inner in _____ {

        i++;

        if (i _____.Key) {

            var poem =

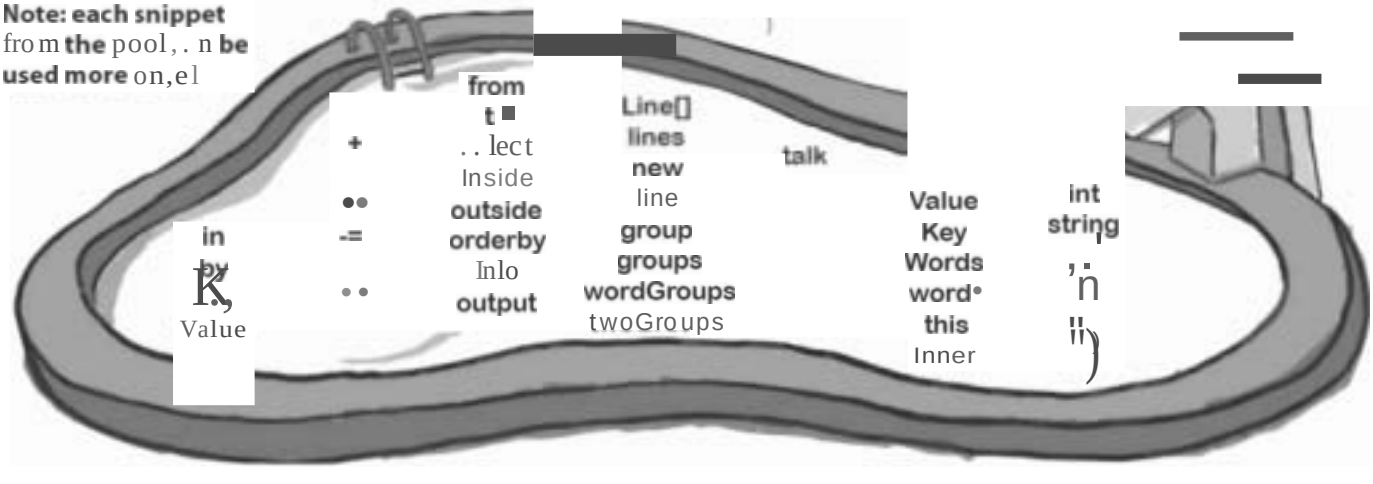
                _____ word in _____

                _____ word descending

                _____ word +

            foreach (var word in _____
                Console.WriteLine(word);
```

Note: each snippet from the pool can be used more than once!



Pool Puzzle Solution



```
public class Line {
    public string[] Words;
    public int Value;
    public Liner(string[] Words, int Value) {
        this.Words = Words; this.Value = Value;
```

```
Line[] lines = {
  new Line( new    "eating", "carrots,", "but", "enjoy", "Horses" }, 1),
  new Line( new    "zebras?", "hay", "Cows", "bridge.", "bolted" }, 2),
  new Line( new    "fork", "dogs!", "Engine", "ena- I. 3 ) •
  new Line( new    "love", "they", "apples.... "eating" }, 2 ) •
  new Line( new    "whistled.", "Bump" }, 1 )
};
```

```
var words =
    from line in lines
    group line by line.Value
    into wordGroups
    orderby wordGroups.Key
    select wordGroups;
```

This first LINQ query divides the Line objects in the lines[] array into two groups, grouped by their Value, in ascending order of the Value key.

```
var twoGroups = words.Take(2);
```

The first two groups are the lines with Values 1 and 2.

```
foreach (var group in twoGroups)

    int i =0;
    foreach (var inner in group) {
        i++;
        if (i == group.Key) {
            var poem =
                from word in inner.Words
                orderby word descending
                select word + " ";
            foreach (var word in poem)
                Console.Write(word);
```

This loop does a LINQ query on the first Line object in the first group and the second Line object in the second group.

Did you figure out that the two phrases "Horses enjoy eating carrots, but" and "they love eating apples" are in descending alphabetical order?

Output: **Horses enjoy eating carrots,** but **they love eating** apples,

Use Joil1 to cOl\lbil1e two collectiol1s il1to ol1e query

Jimmy's got a whol,' collection of comics he's purchased, and he wants to co'np"re rlu-m with the prices he fonmd on Greg's I.i'l 10 see if the pr;ces he's been getting are be,ter or worse He's been tracking his purchase using a `Purchase` class with two fields, `Issue` and `Price`, ,\nd he's got a `List<Purchase>` called `purchases` that's got all the comics he's bought. But now he needs 10 "1atch "I' the purchases he's ",ad,- with the I'rice·, h,' to "nul 0" Greg's List. H,)w's he going 10 d(, it?

LINQ to the rescue! Its `join` keyword lets you **combine** data from two **coll..ctions** into a single query. It doe it by comparing ;t"Ill,;n the first collection their matching items in the second collection. (LINQ i, smart enough to d" Ihi, efficiently - il d,)esn't actually compare every pair "I' items unless it ha, to.) The ,-",I result i, a li"al re,,,h Ihat combines every pair thaI ",atd"-,,

Jimmy's got his data in a collection of Purchase objects called purchases

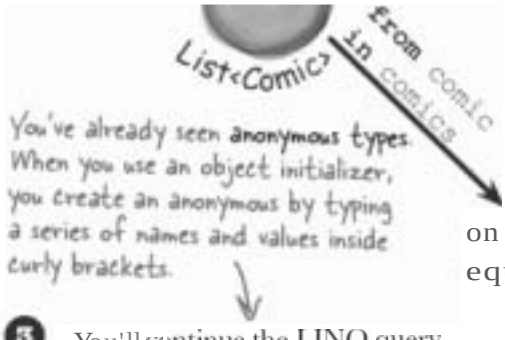
```
public class Purchase t
  pUblic int Issue
             { get; set; };
  public int Price;
             { get; set; };
```

O Statl «I' you query with tll,' usual `from` clause. But i"'"';1(1of following it "I' with the criteria it'll use to determine what goes inn the results, you add:

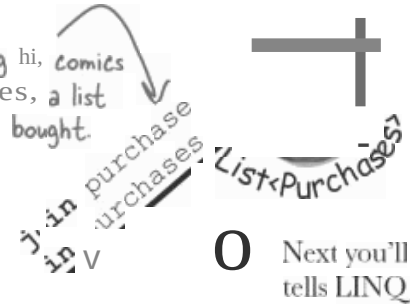
```
join name in collection
```

The `join` d"us<' tells LINQ to loop through bolh ('"Il,-,'I;on' to match "I' pairs of one """,,,h,-,. from each collection. It assigns *name* to the member it'll ['" Il out of Ihe jcum-d collection in each iteration. You'll usc thar name in til<' where d'UI"',

J,mmy's joining hi, comics to purchases, a list of comics he', bought.



You've already seen **anonymous types**. When you use an object initializer, you create an anonymous by typing a series of names and values inside curly brackets.



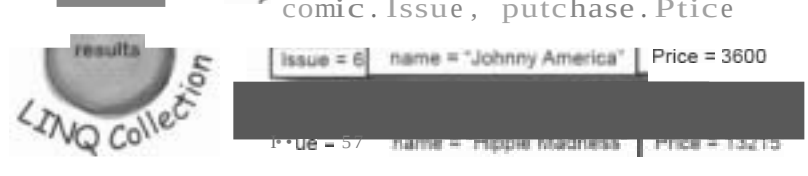
O Next you'll add the `on` clause, which tells LINQ how to match Ihe two collections together. You'll follow it with the name "f the "" ',,,h,-, til' the first collection you're-matching, followed by **`equals`** "uri the name of the member of the second collection tu match it ttl,

```
on comic.Issue
equals purchase,Issue
```

The `select new` is followed by curly brackets that contain the data to return in the results.

```
select new { comic.Name,
comic.Issue, putchase.Ptice
```

❺ You'll continue the LINQ query "ith `where` and `orderby` clauses as usual. You could tilli,h it with a no,-mal `select` clause, but y(" usually want I" return results rh.« pull-, some data from one collectiol and other data from the other. That's where vou use **`select new`** to create a ,""to'n set of results using an **anonymous** type.



Jimmy saved a bunch ot dough

It I,,oks like Jnnmv drives a hard hargain. H,' created a li,1 01 Purchase classes that contained his purchases, and compared them with the prices he found "n Greg's l.i,1.

O First J immy ereetee his collection to join.

Jimmy already had his first c<,llection—he ju-t used his Bui Ideat a log () method from before. So all he had l" do was write a FindPurchases () method to build his li,1"r Purchase classes.

```
public sList<Purchase> FindPurchases() {
    List<Purchase> purchases. new List<Purchase>()
        new Purchase()   Issue   68, Price . 225 },
        new purcnaset}   Issue   19, Price · 375 },
        new purchaset}   Issue   6, Priee = 3600 },
        new Purchase()   Issue   57, Price   13215 I,
        new Purchase()   Issue   36, Price - 660 },
    };
    return purchases;
```

J,....'t paid $13,215 for issue #57.

O Now he could do the join!

You've seen all the parts 01 this query already _now here they are, put together in "ll" piece.

```
List<Comic> comics - BuildCatalog();
Dictionary<int, int> values _ GetPrices();
List<purchase> purchases · FindPurchases();
var results =
    from comic in comics
    join purchase in purchases
    on comic.Issue equals purchase.Issue
    orderby comic.Issue ascending
    select new { comic.Name, comic.Issue, purchase.Price );
int gregsListValue - 0;
int toralSpent = 0;
foreach (var result in results) {
    qreqsList.vaLue += vaIues[result.Issue];
    totalSpent •• result.Price;
    Console.WriteLine("Issue I{O) ({1}) bought for 12:cl",
            result.Issue, result.Name, result.Price);

Console.WriteLine("I spent {0:c} on comics worth {1:c}",
            t.ot.alSpent, qreqsListvaLuelr
```

When J,....y used a join clause, LINQ compared every ite.. in the comics collection with each ite.. in purchases to see which ones have comic.Issue equal to purchase.Issue

The select ne.. clause creates a result set ..ith Name and Issue from the comic ..e..ber, and Price from the purchase ..e...be-.

Ji....y's real happy that he knows LINQ, because it let him see just how hard a bargain he can drive!

OUtput

Show output from: Debug

```
Iss     (Johnny America vs. the Pinko) bought for 1',O00.00
Iss   #19 (Rock and Roll Ill..ited edition)) bought for $375.00
Iss   #36 (Woman's Work) bought for $660.00
Iss   #57 (Hippie Madness Imisprinted)) bought <or $13,215 00
I     #68 (Revenge of the New Wave Freak (damaged)) bought for $225.00
I spent $18,075.00 on comics worth $18,525.00
```



Okay, so now I know Jimmy played with his comie books using LINQ queries to query his collections... but what about the Starbuzz promotion problem? I still don't see how LINQ works with databases.

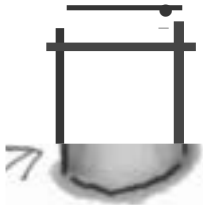
Even though LINQ to SQL is "rt different under the hood, when you write your code it looks really .i..ila.. to other LINQ queries.

IINQ uses the same syntax with databases as it does with collections.

You've already " -"ll ill Chapter I how easy .NET makes ;1 to work with a database. The II1E gives you a really convenient way to connect with databases, add tables, and r-vr-n link data in those tables to your forms.

N ow, you can l"k,' that same database yorn already connected 10 and query il with LINQ. Not only lhal, LINQ lets you combine your data from your database with data from your objects seamlessly.

[n fact, vou can use the same exact query svntax .. all you need is I" get access t" yOUr database so yOu can run a LINQ query against it.



ContactDB database

The Objectville Contact Database has the addresses.



The StarBuzz frequent customer data's here

LINQ

We can use LINQ to compare and combine data from more than one source and create a collection of results



COl1l1ect LINQ to a SQL database

LINQ operates »n collections that implement the IEnumerable interface, riehr? S" il ,h, ntld make sense that y>U access your SQL database using an object that implements IEnumerable..And C# makes it easy to add that object to your project.

Do this

O Add the Objectville Contoct betcbese to a **new** project

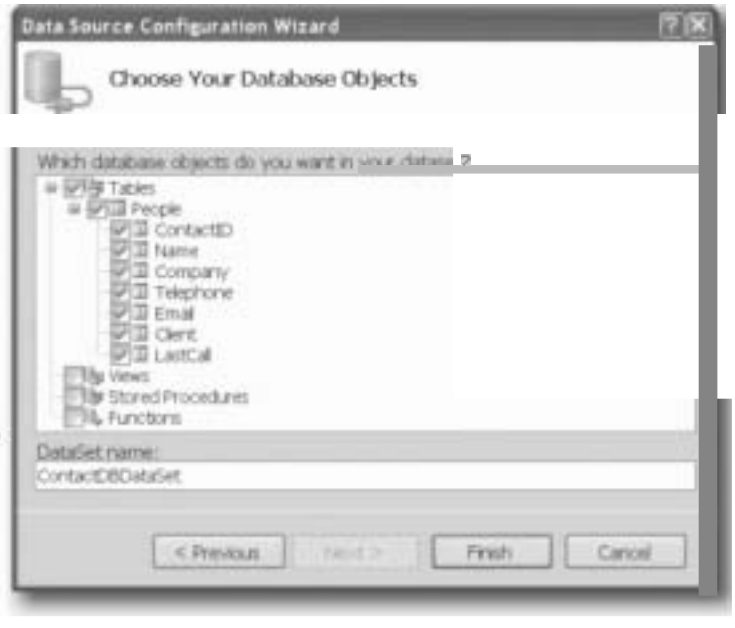
II'H'k in (:h" I'I,'r 1, you created a database of contacts (>1" the Objectville Paper Company and saved il in a file called Contact DB.mdf. Start a new Windows Application project, right-click on vour project in the SolUlion Explorer, select "Add Existing Item' and add the database. Make sure you select "Data Fil"," I;,,,,, Ih....(lbject of Typ.." filter li,1.

O The IDE will **pop** up the **Data Source Configuration Wizard**

Choose the Pe<>pl,- table by selecting its checkbox. Click Finish the wizard will create a datase called ContactDB!)ataSet and add il In your project automatically.

This il the sa..e wizard that you used in Chapter I when you first created the database It adds classes to your project that let you access the database directly.

Take a minute and flip back to Chapter , to see how you built it



3 Add **the** LINQ to SQL **Classes** to your project

Right-click on the project in the Solution EXI'lolIT and choose "Add New Item". It'll display the familiar li,1 "I' icons—choose Ihe UNQ 10 SQL **Class..s** and call il Contact DB.dbml.



When you add the LINQ to SQL Classes to your project, the IDE automatically adds a collection class to your project that you can use with LINQ.

The IbE has a designer to build your SQL collection

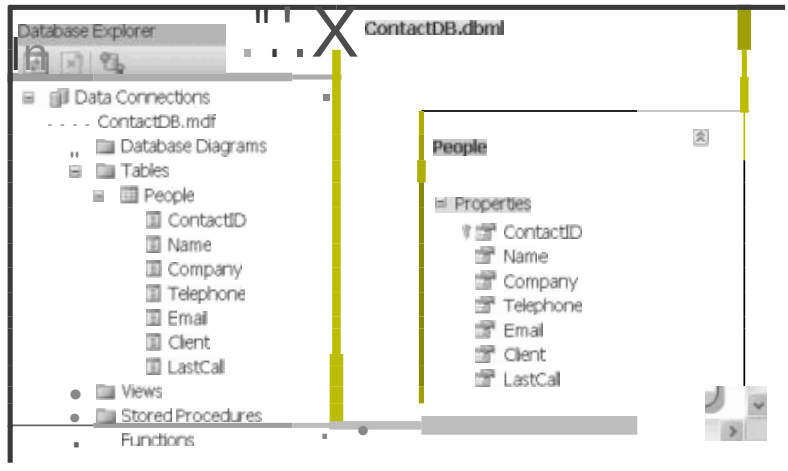
As soon as you add the LINQ to SQL Classes to your project, the 11)E pops "I' an empty window called the ()]ject Relati,nal Designer. Here's what it looks like:



Drag the People table to the Object Relational Designer

Cli, k ou the Database Explorer link in the Object Relational Designer window— the IIlE will pop up a Database Explorer window. Expand the Tables node, click on the People table icon, drag il into ,Il,' Object Relational Designer window, and save the I'"ject.

The DataContext is a little too s..art for its own good. It knows that it's got a People table, so it assumes that the table contains a bunch of rows, one for each "People".. so it has a ..e..ber called Peoples to contain each individual People.



As soon as you save the project, the IDE automatically adds a class called a DataContext to your project—one of its properties is the collection you can query with LINQ, which is connected to the database.

You're all set to write LINQ queries that pull data out of the database

Add a button I" the ku'm-e-here's rhe code for it. Notice how we used the `select new` keyword 10 create custom results that only contain the Name and Company.

```
ContactDBDataContext context = new ContactDBDataContext();

var peopleData =
    from person in context.Peoples
    select new { person.Name, person.Company };

foreach (var person in peopleDatal
    Console.WriteLine("{0} works at {1}", person.Name, per.son vCorspanyjr
```

Get some practice using `select new` It'll pull just the values from the Name and Company columns from the database



BUlIET POINTS

- The q:r:oup eecse tells lINO to group ee results together—when you use it, LINQ creates a collection 01 group collections
- Every group contains members that have one member ın common, called ee group's **key** Use the by keyword to specify the key for the group Each group collection has a **Key** member that contains the group's key
- Use a **join** clause to tell UNO to combine two collections into a single query When you do, LINQ compares every member of the first collection with every member 01 the second collection, including the matching pairs ın the results
- Join queries use an on **equals** eecse to tell UNO how to match the pairs of items
- When you're doing a JOln query, you usually want a set of resets that includes some eerees from ee first collection and other members from the second collection. The **select** ne" clause lets you build custom results from both of them
- lINO can query a SOL database using the UNQ to SOl **Classes**. SeCe UNO only works with collections that implement IE numerable, they provide a collection that lets you access the tables and queries as if they were a collection
- The IDE's Object Relational Designer lets you choose the tables that you want to access via lINO When you specify the tables you want to access, it adds a DataContext cess to your project When it's instantiated, add its members to your LINQ queries to eccess the SOL tables

there are no Dumb Questions

Q_: Can you rewind a min"".nd explain what var is again?

A: Yes, definitely The vat keyword solves a tricky problem that LINQ brings with it Normally, when you call a method or execute a statement, it's absolutely clear • •actly what types you're working with If you've got a method that returns a `string`, for instance, then you can only store its results ll' et ring variable or field

But LINQ isn't quite so simple When you build . llNQ statement, it usually returns • type that *isn't defined anywhere in your program*. Yes, you know that it's going to be a collection of some sort. But what kind of collection Wl. it be? You don't know—because the objects that are contained In the collection depend entirely on what you put in your LINQ query

Take ee query, for example

```
var mostExpensive •
    from comic in ccence
    where valueaIcomtcc Lasue] > 500
    select comic:
```

What if you changed the last line to this:

```
    select comic.lssue;
```

That's a perfectly valid LINQ query Instead of returning a collection of `Comic` objects, it'll return a collection of v.lue,. And that presents a problem for C#—those are two d,fferen! types, and we'd have to add extra statements to define those types So instead, C# gives us the `ver` keyword, which tells the compiler "Okay, we know that this is a valid type but we cent exactly tell you what It is right now So why don't you just figure that out yourself and not bother us with it? Thanks so much."

there are no Dumb Questions

Q: I don't quit. gt! how join works.

A: Join works with any two collections Lee s say you've got a collection of football players called players—its items are objects that have a Name property, a Positlon property and a Number property So we could pull out the players whose jerseys have a number bigger than I (i with this query

```
var results
 from pleyer in players
 vnere player.Number > 10
 select player;
```

Let's say we wanted to figure out each player's shirt size, and we ve got a jerseys collection whose items have a Number property and a Slze property A join would work really well for that

```
var results -
 from player in pLayers
 where pleyer.Number > 10
 join shirt in jerseys
 on player.Number
 equals shirt.Number
 select shirt;
```

Q: Hold on, that query will just give me a bunch or shirts. What ifl wanl10 connect each player to his shirt liz., end I don'l e. . . .bout his numbtr It all?

A: That's what select new is for It lets you construct an **anonymous type** that only has the data you wantm It And it lets you pick and choose from the various collections that you're joining together, too SO you canselect the player's name and the shirt's see. and nothing ese

```
var results -
 from p.layer in pLeyers
 where player.Number > 10
 join shirt in jerseys
 on player.Number
 equals shirt.Number
 select new t
      player.Name,
      shirt.Size
 l;
```

The IDE is smart enough 10 figure out exactly what results you'll be creating with your query If you o este a loop 10 enumerate through the results, as soon as you type the variable name the IDE will pop up an IntelliSense list.

```
foreach (var r in results)
    r.
```



Notice how the list has Name and Size in n If you added more items to the select new clause, they'd show up In the list too That's because the query would create a different anonymous type with different members

Q: But does that only wo.k with joins?

A: No, you can use select new with any UNa query where you want 10 build a results collection that only includes certain items in it

Q: DoI always have to add those UNO to SQL elll"'l if I wtnt to use UNQlo query a SOL database? What a.. they?

A: Yes, you do have 10 create them. UNa object that implements the IEnumerable interface A SOl. database doesn't normally implement that interface or any interface, really, because it s notan object SO if you want LINQ 10 work with SQL—or any other source of data that you can query—then you need an object that interacts with it and implements the IEnumerable interface

That's why the IDE provides the LINQ 10 SQL classes for you When you add it to your project, it automatically does everything you need in order 10 connect LINQ 10 a SQL database it lets you drag database objects into its Object Relational Designer, and when you do it automatically reads your database's tables and creates classes (like the People class) that LINQ can use to access them

You ran use "select new" to construct cuatom LlNQ query results that include only the items II.al you wan! in YOUI' result collection,

Use a joill query to connect Starbuzz alld Objecfville

Now you have all the tools that you need 10 combine the data from SI"rbulZ and (1hj,'"„ill,' Paper Company into one final result set.



O Add the SQL data to your project

If you haven't already done ;" create a new Window application project ;„„r1 add the ContactDB SQL database t(it. The" add lh,' LINQ '0 SQL classes t" the projec" and write a simple test query jus tu make sure it's all working.

2 Build the Starbuzz objects

Here's the lis thar contains the Starbuzz customer data..Add them I" your project:

```
public class StarbuzzDatd { {
    public string Name { get; set; )
    public Drink FavoriteDrink { get; set; }
    publ ic int MoneySpent { get; set; }
    public int Visits { get; set; )

public enum Drink (
    BoringCoffee, ChocoRockoLatte, TripleEspresso,
    ZestyLemonChai, DoubleCappuccino, HalfCafAmericano,
    ChocoMacchiato, BananaSplitlnACup,
```

The Starbuzz data comes as a collection of StarbuzzData objects. It's got a lot of data—you won't need it all for the promotion, so you'll have to select only the data you need in the LINQ query.

Starbuzz has plenty of great drinks, and each customer has his or her favorite.

\;,u'li also need a method to generate """" sample data

```
public List<StarbuzzDdta> GetStarbuzzDataO {
    List<StatbuzzData> list _ new List<StatbuzzData>()
        new StarbuzzData() I
            Ndme • "Janet Venutian", FdvoriteDtink _ Drink.ChocoMacchiato,
            MoneySpent = 255, Visits = 50 I,
        new StarbuzzData() {
            Name = "Liz Nelson", FavoriteDrink = Drink.DoubleCappuccino,
            MoneySpent _ 150, Visits - 35 I,
        new StarbuzzData() {
            Name · "Matt Franks", FavoriteDrink • Drink.ZestyLernonChai,
            MoneySpent = 75, Visits = 15 I,
        new StarbuzzData() {
            Name = "Joe Ng", FavoriteDrink = Drink.BananaSplitlnACup,
            MoneySpent - 60, Visits - 10 },
        new StarbuzzData() {
            Name • "Sarah Kalter", FavoriteDrink • Drink.BoringCoffee,
            MoneySpent = 110, Visits = 15 J
    };
    return list;
```

GetStarbuzzData() uses a collection initializer and object initializers to set up the Starbuzz objects.

We built thi···ethod so that it has some na..es that also appear in the Objectville contact list. If you used different names make sure you've got matching data here.

3 Now join the SQL databo.se to the Starbu%%collection

Add a button to the project that will execute the query and display the results in the console. Here's I],,. code for the query:

```
List<StarbuzzData> starbuzzList = GetStarbuzzData{);
Contact DBDataContext context = new ContactDBDataContext{);

val' results =
    from starbuzzCustomer in starbuzzList
    where starbuzzCustomer.MoneySpent > 90
    join person in dataContext.Peoples
    on starbuzzCustomer.Name equals person.Name
    select new ( person.Name, person.Company,
                 starbuzzCustomer.FavoriteDrink );
```

We'll need to do a join to combine the Starbuzz data with the customer data in the People table

Here's where the select new clause pulls the na..e and company from the database and the favorite drink from the Starbuzz data into one single result collection.

The Peoples member in the DataContext is a collection that gives r- access to the People table in the database.

and it's easy enough to write to the console:

Check your results—make sure it works the way r- expect it to.

```
foreach (val' row in results){
    Console.WriteLine("{O} at {1} likes {2)",
                      row.Name, row.Company, row.FavoriteDrink);
```

Edit queries with LINQPad

There's a great learning tool for exploring and using LINQ. It's called LINQPad, and it's available for free from Joe Albahari (one of the tech reviewers who helped make sure "Head First C#" was reasonably error-free). You can download it here:

http://www.albahari.com/linqpad.html



Nice work... with this new promoTion, I'll bet we'll geTtons of repeet business. I'll defini Tely be calling you again.

Name: Date:

C# Lab

Invaders

This lab gives you a spec that describes a program for you to build, using the knowledge you've gained over the last few chapters.

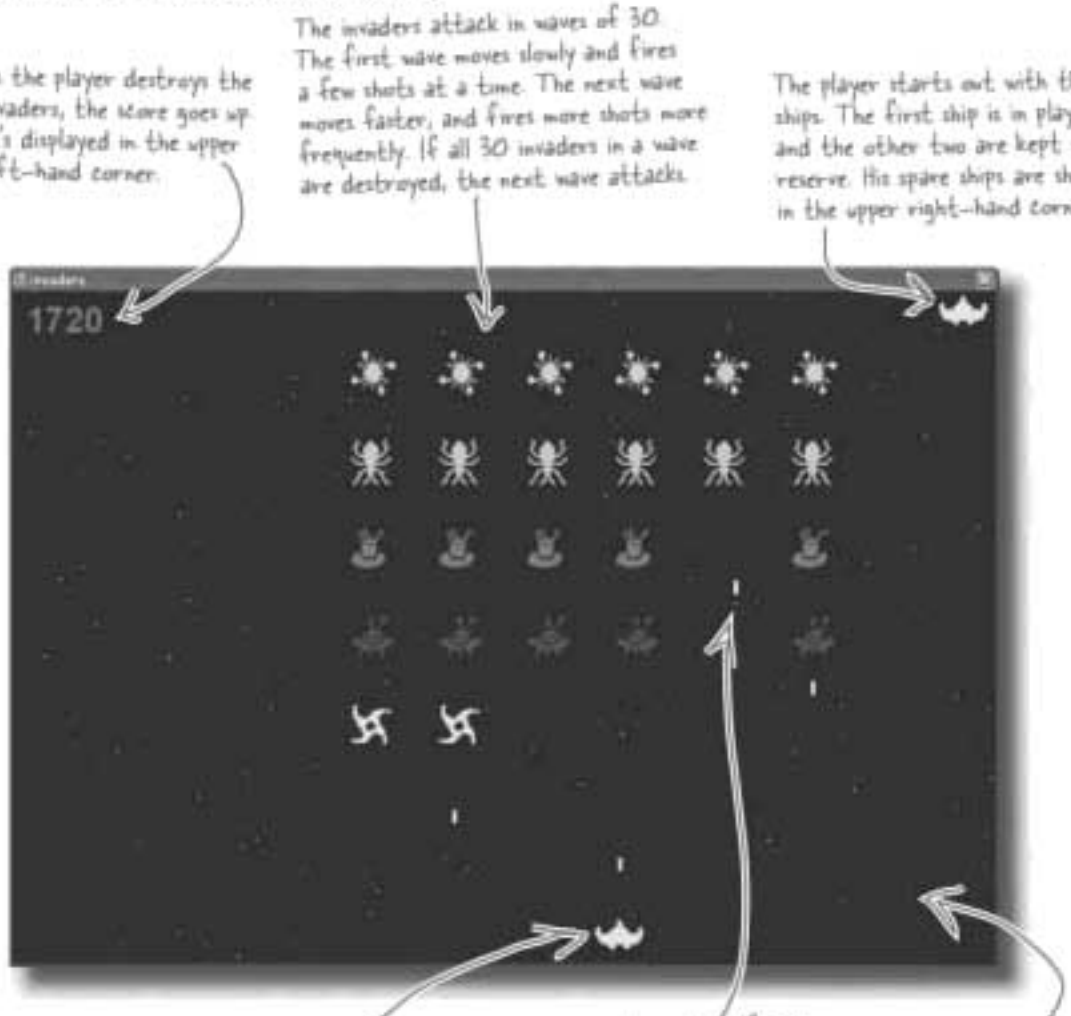
This project is bigger than the ones you've seen so far. So read the whole thing before you get started, and give yourself a little time. And don't worry if you get stuck—there's nothing new in here, so you can move on in the book and come back to the lab later.

We've filled in a few design details for you, and we've made sure you've got all the pieces you need... and nothing else.

It's up to you to finish the job. You can download an executable for this lab from the website... but we won't give you the code for the answer.

The grandfather of video games

In this lab you'll pay homage to one of the most popular, revered and replicated icons in video game history, a game that needs no further introduction. **It's time to build Invaders.**



Your mission: defend the planet against wave after wave of invaders

The invaders attack in waves, where each wave is a tight formation of 30 individual invaders. As the player destroys invaders, the score goes up. The bottom invaders are shaped like stars and worth 10 points. The spaceships are worth 20, the saucers are worth 30, the bugs are worth 40, and the satellites are worth 50. The player starts with three lives. If he loses all three lives or the invaders reach the bottom of the screen, the game's over.

There are five different types of invaders, but they all behave the same way. They start at the top of the screen and move left until they reach the edge. Then they drop down and start moving right. When they reach the right-hand boundary, they drop down and move left again. If the invaders reach the bottom of the screen, the game's over.



The first wave of invaders can fire two shots at once—the invaders will hold their fire if there are more than two shots on the screen. The next wave fires three, the next fires four, etc.



The spacebar shoots. But there can only be two shots on the screen at once. As soon as a shot hits something or disappears, another shot can be fired.



If a shot hits an invader, both disappear. Otherwise, the shot disappears when it gets to the top of the screen.

The game should keep track of all keypresses. So pressing right and spacebar would cause the ship to move to the right and fire (if two shots aren't already on the screen).



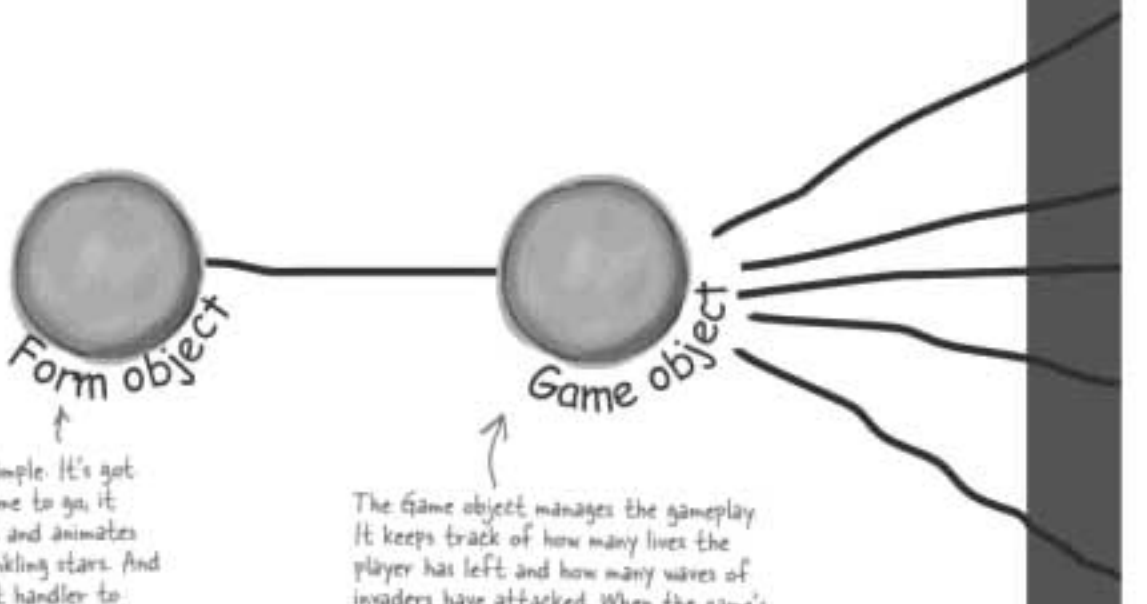
The left arrow moves the ship towards the left-hand edge of the screen.

The right arrow key moves the ship to the right.

The architecture of Invaders

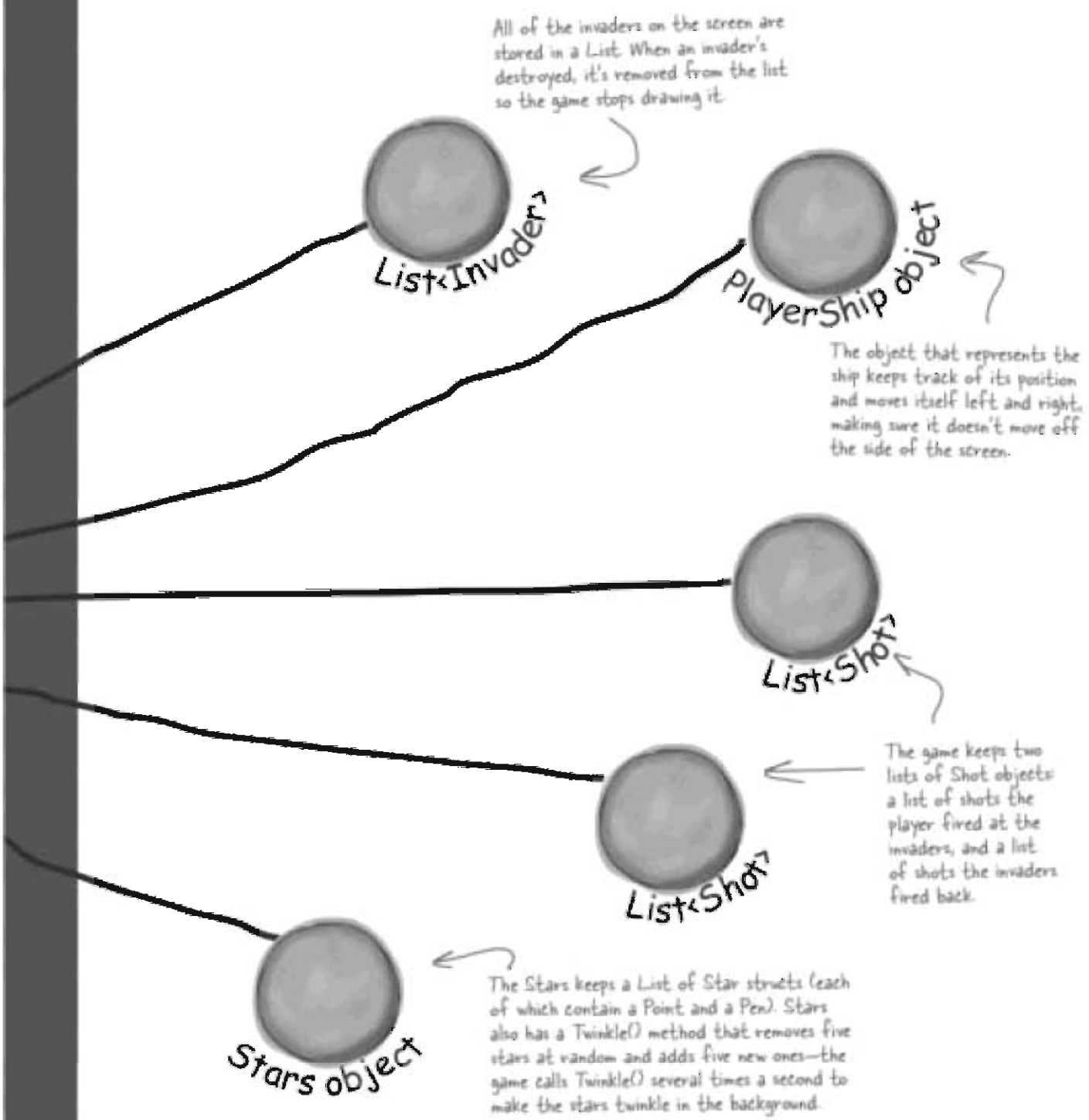
Invaders needs to keep track of a wave of 30 invaders (including their locations, type, and score value), the player's ship, shots that the player and invaders fire at each other, and stars in the background. As you did in the Quest lab, you'll need a Game object to keep up with all this, and coordinate between the form and the game objects.

Here's an overview of what you'll need to create:



The form is pretty simple. It's got timers to tell the game to go, it passes on key presses, and animates the invaders and twinkling stars. And it's got a Paint event handler to draw the graphics, which just calls the Game object's Draw() method

The Game object manages the gameplay. It keeps track of how many lives the player has left and how many waves of invaders have attacked. When the game's over, it raises a GameOver event to tell the form to stop its timers.


All of the invaders on the screen are stored in a List. When an invader's destroyed, it's removed from the list so the game stops drawing it.
List<Invader>
PlayerShip object
The object that represents the ship keeps track of its position and moves itself left and right, making sure it doesn't move off the side of the screen.
List<Shot>
The game keeps two lists of Shot objects: a list of shots the player fired at the invaders, and a list of shots the invaders fired back.
List<Shot>
Stars object
The Stars keeps a List of Star structs (each of which contain a Point and a Pen). Stars also has a Twinkle() method that removes five stars at random and adds five new ones—the game calls Twinkle() several times a second to make the stars twinkle in the background.

Design the Invaders form

The Invaders form has only two controls: a timer to trigger animation (making the stars twinkle and the invaders animate by changing each invader picture to a different frame), and a timer to handle gameplay (the invaders marching left and right, the player moving, and the player and invaders shooting at each other). Other than that, the only intelligence in the form is an event handler to handle the game's GameOver event, and KeyUp and KeyDown event handlers to manage the keyboard input.

The form fires a KeyDown event any time a key is pressed, and it fires a KeyUp event whenever a key is released.

When the form initializes its Game object, it passes its DisplayRectangle to it so it knows the boundaries of the form. So you can change the size of the battlefield just by changing the size of the form.



You should add two timers: animationTimer and gameTimer.

Set the form's FormBorderStyle property to FixedSingle, its DoubleBuffered property to true, turn off its MinimizeBox and MaximizeBox properties, set its title, and then stretch it out to the width you want the game area to be.

The animation timer handles the eye candy

The stars in the game's background and the invader animation don't affect gameplay, and they continue when the game is paused or stopped. So we need a separate timer for those.



Add code for the animation timer's tick event

Your code should have a counter that cycles from 0 to 3 and then back down to 0. That counter is used to update each of the four-cell invader animations (creating a smooth animation). Your handler should also call the `Game` object's `Twinkle()` method, which will cause the stars to twinkle. Finally, it needs to call the form's `Refresh()` method to repaint the screen.

Try a timer interval of 33ms, which will give you about 30 frames per second. Make sure you set the game timer to a shorter interval, though. The ship should move and gameplay should occur more quickly than the stars twinkle.

Animation occurs even when gameplay doesn't. That means that the stars twinkle and the invaders animate even if the game is over, paused, or hasn't been started.

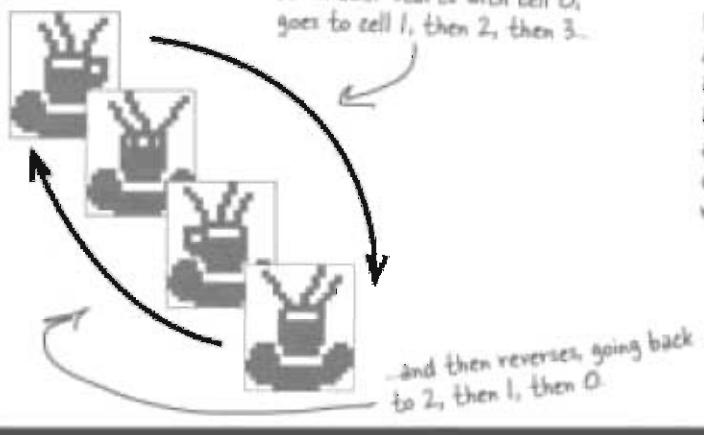
Adjust the timer for smooth animation

With a 33ms interval for animation, set the game timer to 10ms. That way, the main gameplay will occur more quickly than the animation (which is really just background eye candy). At the same time, the `Go()` method in `Game` (fired by the game timer, which we'll talk about in a little bit) can take a lot of CPU cycles. If the CPU is busy handling gameplay, the animation timer will just wait until the CPU gets to it, and then fire (and animate the stars and invaders).

Alternately, you can just set both timers to an interval of 5ms, and the game will run and animate about as fast as your system can handle (although on fast machines, animation could get annoyingly quick).

If the animation timer is set to 33ms, but the Game object's Go() method takes longer than that to run, then animation will occur once Go() completes.

We tried things out on a slow machine, and found that setting the animation interval to 100ms and the gameplay timer interval to 50ms gave us a frame rate of about 10 frames per second, which was definitely playable. Try starting there and reducing each interval until you're happy.



Respond to keyboard input

Before we can code the game timer, we need to write event handlers for the `KeyDown` and `KeyUp` events. `KeyDown` is triggered when a key is pressed, and `KeyUp` when a key is released. For most keys, we can simply take action, by firing a shot or quitting the game.

For some keys, like the right or left arrow, we want to store those in a list that our game timer can then use to move the player's ship. So we'll also need a list of pressed keys in the form object:

So if the player's holding down the left arrow and space bar at the same time, the list will contain Keys.Left and Keys.Space.

We need a list of keys so we can track which keys have been pressed. Our game timer will need that list for movement in just a bit.

```
List<Keys> keysPressed = new List<Keys>();

private void Form1_KeyDown(object sender, KeyEventArgs e) {
  if (e.KeyCode == Keys.Q)
    Application.Exit();

  if (gameOver)
    if (e.KeyCode == Keys.S) {
      // code to reset the game and restart the timers
      return;
    }

  if (e.KeyCode == Keys.Space)
    game.FireShot();
  if (keysPressed.Contains(e.KeyCode))
    keysPressed.Remove(e.KeyCode);
  keysPressed.Add(e.KeyCode);
}

private void Form1_KeyUp(object sender, KeyEventArgs e) {
  if (keysPressed.Contains(e.KeyCode))
    keysPressed.Remove(e.KeyCode);
}
```

The 'Q' key quits the game.

If the game has ended, reset the game and start over.

But we only want this to work if the game's over. Pressing S shouldn't restart a game that's already in progress.

The Keys enum defines all the keys you might want to check key codes against.

You'll need to fill in this code.

The spacebar fires a shot.

By removing the key and then re-adding it, it makes the key the last (most current) item in the list.

The key that's pressed gets added to our key list, which we'll use in a second.

We want the most current key pressed to be at the very top of the list, so that if the player mashes a few keys at the same time, the game responds to the one that hit most recently. Then, when he lets up one key, the game responds to the next one in the list.

When a key is released, we remove it from our list of pressed keys.

Ihe game timer handles movement and gameplay

gameTlmer

The main j. ,h of the form's game tin's-r is to call Go () in the Game class. But ;1 also has I" respond to any keys pressed, so it has t" check the keysPressed list to find anv keys caught by the xeyncvn and KeyUp events:

Make sure your nam-S matches up with what you call your handler method.

```
private void gameTimer_Tick(object sender, EventArgs e) I
  if (keysPressed.Count() >- 1)
    switch (keysPressed[0]) {
      case Keys.Left:
        game vj-lovePlayer(Direetion.Left);
        break;
      case Keys.Right:
        game.MovePlayer{Direction.Right);
        break;
    } ..
  )
  game.Go();
```

keysPressed is your List<Keys> object ..anaged by the KeyDown and KeyUp event handlers The key at index ee-o will always be the ..ost recent key pre..ed

The KeyUp and KeyDown events use the Keys enum to specify a key We'll use Keys.Left and Keys.Right to move the ship.

We only need to deal with move..ent. Other keys, like spacebar, and 'Q' for quit, are handled "' the KeyDown() ..ethod r- just wrote

Finally, we call Go() on the Game object to let ga...e play continue

```
public enum Direction {
    Left,
    Right,
    Up,
    Down,
```

Shots ...ove up and down, the player meees left and .ight, and the invaders move left, right, and down You'll need this enum to keep all those directions straight.

One more form detail: the GameOver event

Add a private booLfickl called gameOver to Ih,' form that's true onlv when the game is over. Then add an event handler for the Game object's GameOver event that stops the game timer (Inti l101 Ih,' animation timer, so the stars still twinkle and the invaders still animate), sets gameOver 10true, and calls the form's Refresh () me'tlu×1.

\\'h..n you write the form's Paint event handler, have it check gameOver. If it's true, have il writ.. GAME OVER ill big yellow letters in Ih,' lIliddl,' of Ihe screen. T IH1I have it write "Press S If) start a new game or Q to quit" ill Ihe lower right-hand corner.

The form's game timer tells the game to Go()

In addition to handling movement left and right, the main job of the game timer is to call the `Game` object's `Go()` method. That's where all of the gameplay is managed. The `Game` object keeps track of the state of the game, and its `Go()` method advances the game by one frame. That involves:

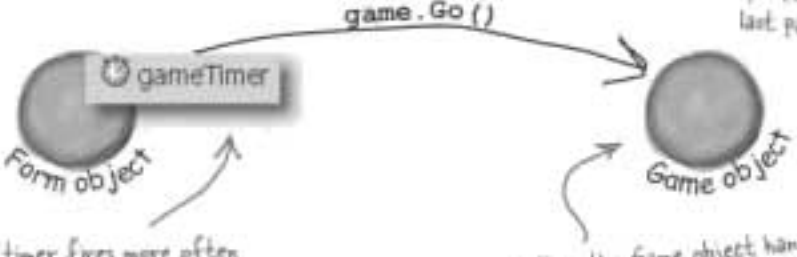
1. **Checking to see if the player died**, using its `Alive` property. When the player dies, the game shows a little animation of the ship collapsing (using `DrawImage()` to squish the ship down to nothing). The animation is done by the `PlayerShip` class, so `Go()` just needs to check to see if it's dead. If it is, it `returns`—that way, it keeps the invaders from moving or shooting while the player gets a small break (and watches his ship get crushed).

2. **Moving each of the shots.** Shots fired by the invaders move down, and shots fired by the player move up. `Game` keeps two `List<Shot>` objects, one for the invaders' shots and one for the player's. Any shot that's moved off the screen needs to be removed from the list.

3. **Moving each of the invaders.** `Game` calls each `Invader` object's `Move()` method, and tells the invaders which way to move. `Game` also keeps up with where the invaders are in case they need to move down a row or switch directions. Then, `Game` checks to see if it's time for the invaders to return fire, and if so, it adds new `Shot` objects to the `List<>`.

4. **Checking for hits**. If a player's shot hit any invaders, `Game` removes the invaders from the appropriate `List<>`. Then `Game` checks to see if any of the invader shots have collided with the player's ship, and if so, it kills the player by setting its `Alive` property to `false`. If the player's out of lives, then `Game` raises the `GameOver` event to tell the form that the game's over. The form's `GameOver` event handler stops its game timer, so `Go()` isn't called again.

Here's where that GameOver event from the last page comes into play.



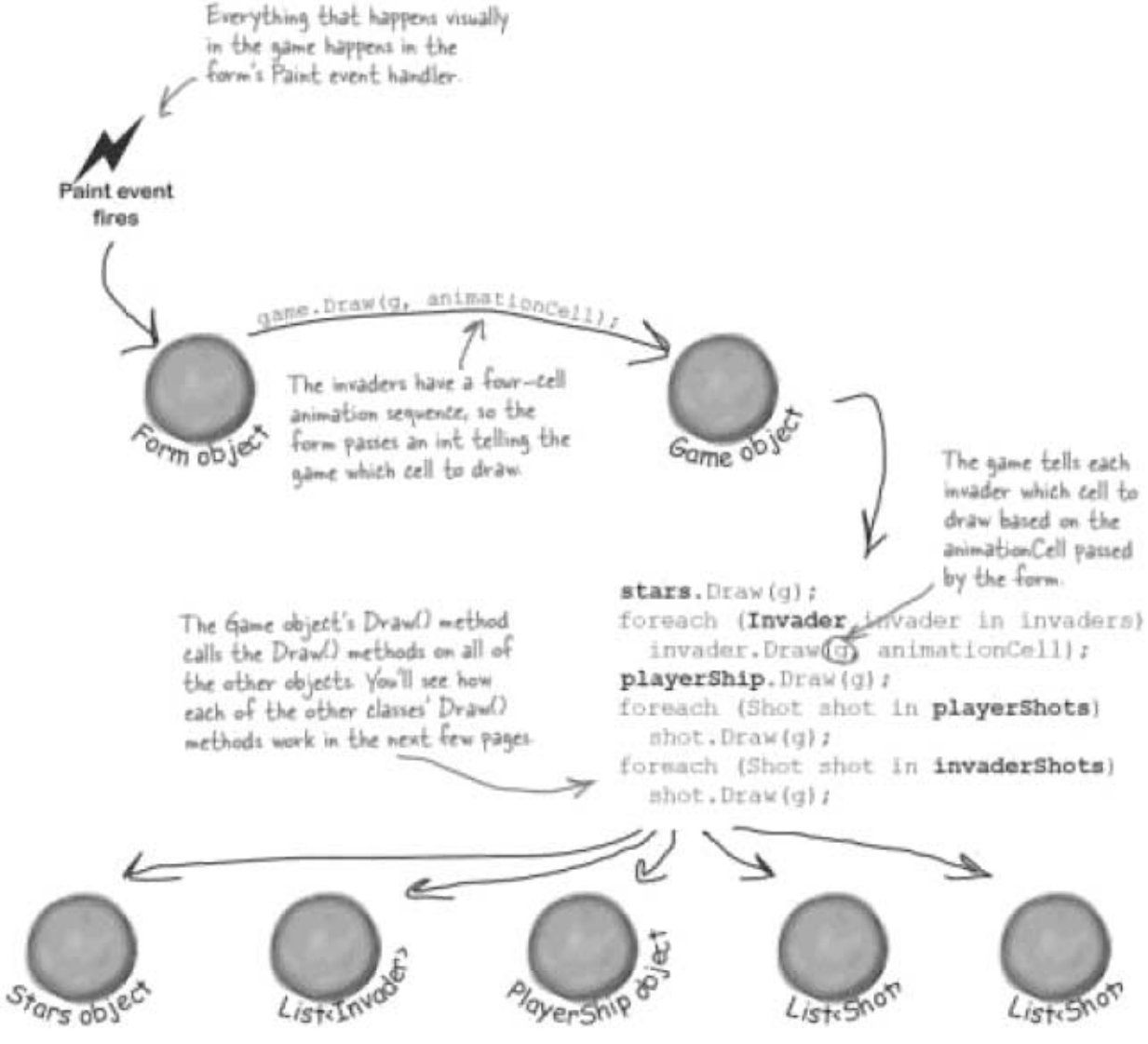
The game timer fires more often than the animation timer, making gameplay happen quickly.

Go() in the Game object handles everything from movement to shots to checking to see if ships or invaders have been hit.

Taking control of graphics

In earlier labs, the form used controls for the graphics. But now that you know how to use `Graphics` and double-buffering, the `Game` object should handle a lot of the drawing.

So the form should have a `Paint` event handler (which you'll put your double-buffering inside). But you'll delegate the rest of the drawing to the `Game` object every time that event fires.



Building the Game class

The Game class is the controller f\'r the In,-,,I,-,, game. Here's a start on what (hi, class should l'K,k lik,", although there's lots "I' work still f"r vuu 10 do.

```
public class Game {
  private lnt score   0;
  private lilt livesLeft - 2;
  private lnt wave = 0;
  private i nt; framesSkipped = 0;

  private Rectangle boundaries;
  private Random random;

  private Dt'rectIon ¯invaderDirection;
  private ListcInvader> Invaders:

  private PlayerShIp playerShip;
  prlvat.e List<Shot> playerShots;
  privat,e List <Shot > invaderShots:

  private Stars stars;

  public event EventHandler GameOver;

  // etc...
```

The score, ivesLeft, and wave fields keep track of some basic information about the state of the ga..e.

You'll use the fra...e field to slow down the invaders early on in the ga..e—the first wave should skip 6 fra..es before they move to the left, the next wave sho·ld skip 5, the next should skip 4, etc

This List<> of Invader objects keeps track of a[[of the invaders in the current wave. When an invader i. destroyed, it's re··oved f·om the list. The game checks periodically to make sure the list isn't empty—if it is, it sends in ¯ne nex¯ wave of invaders

This Stars object keeps track of the ..ulticolored stars in the background.

The Ga..e object raises its Ga..eOve.. event when the player dies and doesn't have any more lives left. You'll build the event handler method in the form, and hook it into the Game object's GameOver event.

.me

cemeovec event

Draw(g: cracbcs. animationCell: int)
TwinkleO
MovePlayer(direction: Direction)
FireShot()
GoO

Re be-, these a-e the public ethods. You may need a lot ore private methods to structu.e your code in a way that ..akes sense to you

Ihe **Game** class **methods**

The Game class has five I"ıblic mr-tlu ıds lhal get triggered 1by different , "\" "ır happening in the form.

- **The Draw() method draws the game on a graphics object**

  The Draw () method takes two parameters: a Graphi ce object and an integer that contains the "nim,,,;;,,, n'll (" number from 0 to n First i, should draw a black rectangle that liJl, lip the whole form (using the display rectangl.. stored ill boundari es. received from the fi>,-m1. Then the method should draw Ill,' stars. the invaders, then the player's ,hip, and ,h,'nlhe shots. Finally, it should draw the score in lh,- upper left-hand corner. the player's ships in the UPI"1' right-hand corner. and a big "GAME OVER" in yellow letter ir gameOver is true.

- **The TWinkleD method twinkles the stars**

  The form's animati,,n timer event handler needs 1" ve able to twinkle Ill<' stars. s rh,' Game

- **The MovePloyerO method moves the player**

  The j;ırm's keyboard timer event h,,,,rll,'r ,,,,,d, to move the player's ,hip, so ,h,' Game ohj,"'1 also needs a two-line method that takes a Direction ,'nllm as a parameter, checks whether or nol the player's d'·ad. and calls piayerShip . Move () '0 affect that movement.

- **The FireShotO method makes the player fire a shot at the invaders**

  The Fi reShot (I "'I'll"ıd checks '" see if there are fewer than two player shots on screen. If there are, ,h,>method ,houlrl add a new , ho, to 11<' piayer Shots Ii" at the right location.

- **The Go() method makes the game go**

  Th,' j;,,-m" ,,,,ima,iOll timer calls Ih,' Game object's Go ( ) method anvwhere between]() and 30 time a second (depending 0" the computer's (: l'tJ 'p","d). The Go ( ) method does everything the game m-eds ıt . d" to advance itself by a frame:

  - The game checks if the player's dead using its Ali ve property. If he's "ill alive, the game i,,,'l ,"',"r yet— if il ı·er·. ,Ire form would have stopped the animation timer with its Stop () """Ih"d. S" the Go ( ) m,'ll"ıd won't do anything <,I", IImil the ,hip', alive again it'll just return.
  - Every shot needs 10 be updated. The game needs In loop through holh List<Shot> „1ıjects, calling each shot's Move () method. Il' any shot's Move () H',um, false, that means Ih,' ,!ro, " ,·,,1on' the edge "I' the screen—so il gets deleted from the 1i,1.
  - Finally, i, checks for collisions: lir,' for any shot that overlaps an invader (and n'm'fI'inr: bnlh li'om their Ldst c>s], and then 10 see if the player's been ,hoI. We'll add a Rectangle property called Area In the Invader and PiayerShip classes—so we can use the Contains () method to "„- ir the ships' area overlap with a ,h,".

Filling out the **Game** class

The problem with class diagrams is that they usually leave out anv llm-public properties and methods. So even after you've got the methods from page 693 done, you've ,till got a lot or work to do. Here are some things to think about:

'11,,' Game object needs In create all "f the other objects tin' Invader objects, the PIayerShip object, the List objects to hold the shots, and the Stars object. The form passes in all initialized Random object and its own DisplayRectangle '(fU,1 (so the Game can **figure out the** h..**undaries** uf th.. haul**e**fi..ld. which it uses to determine when shots are out or range ,,,,d " h"11 the invaders reached th,' edge "ml need to drop ,"ltl reverse direction). Then, your code ;ho"ld create everything else ill the game world.

We'll talk about ...ost of these individual objects over the next several pages of this lab

Build a NextWavell **method**

A simple method to ·reate the next wave "f invaders will come in handy. It should assign a new List "r Invader objects 1<' II,,' ;mraders field, add the 30 invaders in 6 ""Ium,,, so Ihat they're ill their starting I',,,ili,,,,,. in,rease the wave li"ld by I. and set the invaderDirection field to srarttlu-m moving towards the rizlu-hand sid< ol th< screen You'll also chang< th< framesSkLppedfu-ld

Here's an example of a private method that will really help out r-r Game class organization.

Afew other ideas for private methods

Here are a few of the private method ideas you might play with. and see if these would also help the design of your Game class:

- ✓ .\ ""lho,ll" see if the player's been hil (CheckForPlayerCollisions())
- ✓ ,\ method to ".,- if any invaders have been hi! (CheckForInvaderCollisions())
- ✓ A method to mov< all the invaders (MoveInvaders())
- ✓ ,\ method allowing invaders I" return fire {ReturnFire())

It's possible to show protected and **private** properties and methods on **a** class diagram. but you'll rarely see thai put into practice. Why do you think that is?

LINQ l\Iakes collisiol1 detectiol1 much easier

This seems really complex when you fi..st read it, but each LINQ que..y is just a couple of lines of code Here's a hint: don't overcomplicate it!

You've got collections or invaders and shots, and you need ,,, search through those collec'Ii, ,n, «' find certain invaders and shots. Anytime you hear ,', ,)llecti'11' and searching in the same sentence, you sh<.ul,[1!oink LINQ. Here's what you need '" do:

O **Figure** out **if the** invaders' forll'\Qtion has reached **the** edge of **the battlefield**

Th" invaders n("'d to change· dire,-,;"" if any "11(' invader is within 100 pixels of the edge of ,I,,· battlefield. When the invaders are marching I" the right, once they reach the right-hand side "r the form the game needs to tdlll"',,, to drop d",,,, and start marching to lht, [efl. A\"d wh,'" tl,,' invaders are marching to the left, the game needs m check if they've reached the left edge Ti make this happen, add a private `MoveInvaders()` metho..1that gets .:alled by `Go()` Th" lir,r thing il should do is check and IIp late the private `framesSkipped` field, and `return` if this frame ,h"uld h.. skipped (depending on the level). Th"" it should check which direction til<' invaders are moving. If the invaders are moving to the right, `MoveInvaders()` should use LINQ It, search the `invaderCollection` li,l for any invader whose l.ocation's `X` "ah,,' is within 100 pixels of th.. right-hand boundary. If il fi"d, any, then ir, hollld 'e11110e invaders 10 march downwards and thcu set `invaderDirection` equal to `Direction.Left`; if nol. il can tell each invader 10 march to the right. On the other hand, if rio.. invaders are moving \0 lh,' left, then ir sh<>!rld do the "1'1"}site, using an<,rher LINQ query ". see if rhe invaders are within Ir)0 pixel of the left-hand boundary, marching them down and changing direction if they a,.e.

O **Determine** which **invaders** can **return fire**

.Add a pri"ate ",,'rh,)d .:alled `ReturnFire()` rhar gets called by `Go()`. First, it should `return` if 11ne invaders' shot li,t already has `wave + 1` shots. It should al",, `return if random.Next(10) < 10 - wave.` (That makes the invaders fire at random, and ""1 all th.. time.) [f il gets past both tests, ir can use LINQ to group the invaders by their `Location.X` and sort them `descending`, Once it's got those groups, il can choose a group ar rand<"", and use its `First()` """tho,l 11' find 11ne invader at the bottom "f rh.. column. All right, now you've got the shooter you can add a shot to the invader's shot li'l just below the middle uf the invader ("„' the invader's 'Ar e a '" set the shot's 10'"r;"n).

3 Check for **invader** and player collisions

You'll wantto create a method ro ,-heck for collisions. There are three collisions 10 ..-licck 1„1'. and IIne `Rectangle` struct's `Contains()` llwth"d will ,'0"'" in really handy just pass il any `Point`, a"d it'll return `true` if that point is insid.. the rectangle.

- Use LINQ '" lind any dead invaders by looping through the shots ill the player" ,hot list and selecting any invader when· `invader.Area` contains the shot's l,,,-ari,",. R nrove thr- inv<<der and the shot.
- Add a query It, figure out if any invaders reached the b<>lI"rn of the :r :e11 - if so, "lid the game.
- Y<>" do,,'r need LINQ to l,ook fi,,.shots Ihat co,lIid,'d with the player. just a lo<>l> and the player's `Area` property. (Remember, YOQ **can't** modify a **collection** in si d.. **a** (or..ach loop_If you do, you'll get a" [m'alid()perationException wirh a message that the collection was modified.)

Craftillg the lavader class

The Invader class keeps track or a single invader. So when the Game objec, creates a new wave or invaders. it adds 30 instance or Invader In aLietcInvadar> object. Every (illle it, Go() method is called, it calls each invader's Move () "u',hud (0 tell it '" move. .\",1every lilllP its Draw () method is called, it calls ,-""h ;1\\-",1-,."hj,-„!'s Draw () mt·tholl. So you'll need 1" hllild out the Move {} and Draw () methods. ,\ud you'll want to add a private method called InvaderImage () too—it'll curm- in really handy when vou're drawing the invader. Make sun- you ,-all it inside the Draw () method It' keep the image field up to date:

```
public class Invader {
  private canst int HorizontalInterval - 10;
  private const int VerticalInterval - 40;
  public enum Type {
    B
      ug,
    Saucer,
    Sat.ellite,
    Spaceship,
    Star,


  private Bitmap image;

  public Point Location   get; private set; |

  public Type InvaderType | get; private set;

  public Rectangle Area | get |
    return new Rectangle(location, .i.maqe.Size}; I


  public int Score { get; private set; }

  public Invader{Type invaderType, Point location, int score} I
    this.InvaderType - invaderType;
    this.Location - location:
    this.Score - score;
    image - InvaderImage(0);


  1/ Additional methods will go here
```

The HorizontallnterVal constant deter··ines how many pixels an invader ··oves every time it ..arches left or right. VerticallnterVal i, the nu..ber of pixels it drops down when the formation reaches the edge of the battlefield

The invader uses the Type enu.. to figure out wnat ki- nd of enemy ship it is.

Check out what we d,d with the Area property. Since we know the invader's location and we know its size (f..._ its image field), we can add a get accessor that calculates a Rectangle for the area it covers... which ..eans you can use the Rectangle's Contai O ··ethod inside a LINQ uery to detect any ts that collided with an i ade-,

Invader

location- Point
InvaderType: Type
Area: Rectangle
Score int
Draw(g: Graphics. animalionCell: Inl)
Move(direction: Direction)

Build the Invaders' methods

The three core methods for `Invader` are `Move()`, `Draw()`, and `InvaderImage()`. Let's look at each in turn:

There are five types of invaders, and each of them has four different animation cell pictures.











Move the invader ships

First, you need a method to move the invader ships. The `Game` object should send in a direction, using the `Direction` enum, and then the ship should move. Remember, the `Game` object handles figuring out if an invader needs to move down or change direction, so your `Invader` class doesn't have to worry about that.

```
public void Move(Direction direction) {
  // This method needs to move the ship in the
  //   specified direction
}
```

Draw the ship - and the right animation cell

Each `Invader` knows how to draw itself. Given a `Graphics` object to draw to, and the animation cell to use, the invader can display itself onto the game board using the `Graphics` object the `Game` gives it.

```
public void Draw(Graphics g, int animationCell) {
  // This method needs to draw the image of
  //   the ship, using the correct animation cell
}
```

Get the right invader image

You're going to need to grab the right image based on the animation cell a lot, so you may want to pull that code into its own method. Build an `InvaderImage()` method that returns a specific `Bitmap` given an animation cell.

```
private Bitmap InvaderImage(int animationCell) {
  // This is mostly a convenience method, and
  //   returns the right bitmap for the specified cell
}
```

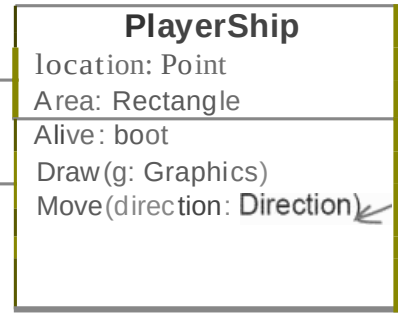
Each invader knows its type. So if you give its InvaderImage() method a number for its animation cell, it can return a Bitmap that's got the right graphic in it.

Ihe player's **ship** call \Nove alld die

The PlayerShip class k,eps track of the player's ,hip. It's similar to ıh,' Invaders class, bUI even simpler.

Tke Location and Area properties are exactly like the ones in the Invader class.

When the ship's hit with a shot, the game sets the ship's Alive property to false The game then keeps the invaders from • oving until the ship ..esets its Alive property back to true



The Draw() method just dra_• the player' shif ift the ..ight location—unless the player died, in ..hich case it dra a.. animation of the ship gett g crushed by the shot

|-The Move() method takes one para...ece-, a Direction enum, and moves the player ,,, that direction

PlayerShip needs to take in a Rectangle ..ith the game's bounda.ies ift its constructor, and ..ake .....e the .hif doesn't get moved out of the ga..e's bounda.ies ift Move()

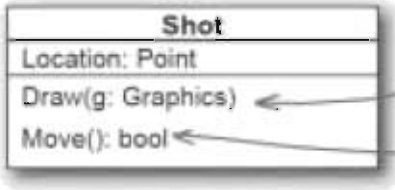
A.itllate the player .hip **when it's hit**

The nrev () method ,hould take a Graphics object as a parameter. Then it checks its Alive property. If it's alive, il draws itself using its Location property. If it's dead, then instead "r drawing the regular hill""!, ,,,, th,-graphics, the PlayerShip "hj",J uses its private deadShipHeight field I" animate the player ship slowly getting crushed by the sh<,I. After 1hn"- seconds of being dead, it ,h,)uld 1lip ii' Alive property back to true.

```
public void Draw(Graphics q) {
  if (!Alive) {
      Reset the deadShipHeight field and draw the ship.
    else {
      Check the deadShipHeightfield. If it's greater than zero, decrease it by J
      and lise DrawImage() to draw the ship a little flatter.
```

"Shots fired!"

Game has two lists of Shot objects: one for the player's shots, moving up the screen, and one for enemy shots, moving down the screen. Shot only needs a few things to work: a Point location, a method to draw the shot, and a method to move. Here's the class diagram:



Draw() handles drawing the little rectangle for this shot. Game will call this every time the screen needs to be updated.

Move() moves the shot up or down, and keeps up with whether the shot is within the game's boundaries.

Here's a start on the Shot class:

```
public class Shot {
  private const int moveInterval = 20;
  private const int width = 5;
  private const int height = 15;

  public Point Location { get; private set; }

  private Direction direction;
  private Rectangle boundaries;

  public Shot(Point location, Direction direction,
              Rectangle boundaries) {
    this.Location = location;
    this.direction = direction;
    this.boundaries = boundaries;
  }

  // Your code goes here
}
```

You can adjust these to make the game easier or harder... smaller shots are easier to dodge, faster shots are harder to avoid.

The shot updates its own location in the Move() method, so location can be a read-only automatic property.

Direction is the enum with Up and Down defined.

The game passes the form's display rectangle into the constructor's boundaries parameter so the shot can tell when it's off of the screen.

Your job is to make sure Draw() takes in a Graphics object and draws the shot as a yellow rectangle. Then, Move() should move the shot up or down, and return true if the shot is *still* within the game boundaries.

Twinkle, twinkle... it's up to you

The last class you'll need is the Stars class. There are 300 stars, and this class keeps up with all of them, causing 5 to display and 5 to disappear every time Twinkle() is called.

First, though, you'll need a struct for each star:

```
private struct Star {
  public Point point;
  public Pen pen;

  public Star(Point point, Pen pen) {
    this.point = point;
    this.pen = pen;
  }
}
```

Each star has a point (its location) and a pen (for its color).

All Star does is hold this data... no behavior.

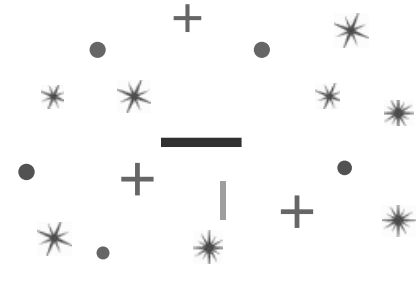
The Stars class will keep a List<Star> of, storing 300 of these Star structs. You'll need to build a constructor for Stars that populates that list. The constructor will get a Rectangle with the display boundaries, and a Random instance to use in creating the random Points to place each star in a random location.

Here's another hint: start out the project with just a form, a Game class, and Stars class. See if you can get it to draw a black sky with twinkling stars. That'll give you a solid foundation to add the other classes and methods.

Stars

Draw() should draw all the stars in the list, and TwinkLe() should remove five random stars and add five new stars in their place.

You might also want to create a RandomPen() method so you can get a random color for the stars every time you create a new star easily. It should return one of the five colors stars come in, by generating a number between 0 and 4, and selecting the matching Pen object.



Alld yet there's lIIore to do...

['h j'lk the game's lo·king pretty good? YOll can take il to the next level with a few more additions:

Add animated explosions

Make each invader explode after it's hit, then briefly display a number to tell the player how many points the invader was worth.

Add a mothership

Once in a while, a mothership worth 250 points can travel across the top of the battlefield. If the player hits it, they get a bonus.

Add shields

Add floating shields the player can hide behind. You can add simple shields the enemies and player can't shoot through. Then, if you really want your game to shine, add breakable shields that the player and invaders can blast holes through after a certain number of hits.

Try making the shields last for fewe·· hits at highe.. level. of the game

Add dlvebombers

Create a special type of enemy that dive bombs the player. A divebombing enemy should break formation, take off towards the enemy, fly down around the bottom of the screen, and then resume its position.

Add more weapons

Start an arms race! Smart bombs, laeera. guided missiles... there are all sorts of weapons that both the player and the invaders can use to attack each other. See if you can add three new weapons to the game.

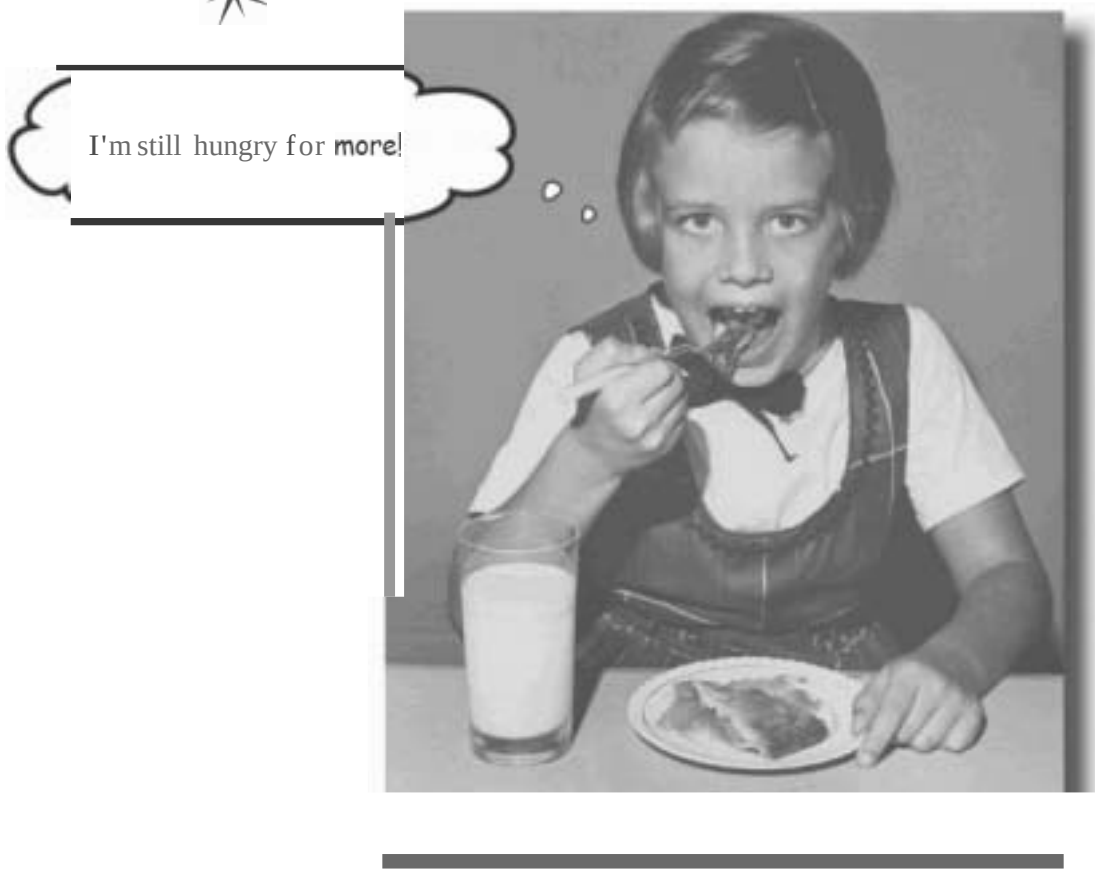
Add more graphics

You can go to www.hee.dfirstlabs.com/books/hfcsharp/ to find more graphics files for simple shields, a mothershfp, and more. We provided blocky, plxelated graphics to give it that stylized '80s look. Can you come up with your own graphics to give the game a new style?

A good class design shou ld let you change out graphics with ..inimal eede changes

This is your chance 10 **show** *oW* Did you COme up **with** a cool **new** version of **the** game? Join **the** Head Firsl C# forum and claim your bragging **rights:** **www.headfirstlal**is.**com/books/hfcsharp/**

The top 5 things we wanted to include in this book



The fun'sjust beginning!

We've shown you a 101 of greallools 10 build some really **powerful software** with e# But there's no way thai we could mcruoe every single tool, technology, or technique in this book-there just aren't enough pages. We had to make some *really tough choices* about what to include and what to leave out. Here are some of the topics that didn't make the cut. But even though we couldn't gel to them, we sllllllhink that they're important and useful, and we wanted to give you a small head start with them,

"I LINQ to XML

XM11. - or **Ex**tensible **M**arkup **L**anguage - is a format fi,r lil,', and data streams that represents complex data as text. The .NET framew'rk gives "'\l some really powerful 1,"ols fi,r creating, l,oading and saving XM11. lii," ..\nd once you've got your hands "n XML data, you can use LINQ 10 query il. Add add "using System. Xml. Lfnq r" 10 the 101' of a til" and enter this method— il generate an XMII. d,ocument to store Starbuzz customer loyalty data.

```
private static XDocument GetStarbuzzData()
    XDocument doc - new XDocument(
       new XDeclaration("1.0", "utf-S", "yes"),
       new XComment("Starbuzz Customer Loyalty Data"),
       new XElement("starbuzzData",
           new XAttribute I"storeName", "Park Slope"),
           new XAttribute("location", "Brooklyn, NY"),
           new XElement("person",
               new XElement("personalInfo",
                   new XElement("name", "Janet Venutian"),
                   new XElement("zip", 11215l',
               new XElement("favoriteDrink", "Choco Macchiato"),
               new XElement("moneySpent", 2551,
               new XElement("visits", 50» ,
           new XElement("person",
               new XElement("personalInfo",
                   new XE1ement("name", "Liz Nelson"),
                   new XElement("zip", 11238)),
               new XElement("favoriteDrink", "Double Cappuccino"),
               new XE1ement("moneySpent", 150),
               new XElement("visits", 35)I,
           new XElement("person",
               new XElement("personallnfo",
                   new XElement("name", "Matt Franks"),
                   new XElement("zip", 11217)),
               new XElement("favoriteDrink", "Zesty Lemon Chai"),
               new XElement("moneySpent", 75),
               new XElement("visits", 15»,
           new XElement("person",
               new XElement("personalInfo",
                   new XElement("name", "Joe Ng"),
                   new XElement("zip", 11217)1,
               new XElement("favoriteDrink", "Banana Split in a Cup"),
               new XElement("moneySpent", 60),
               new xaieeent.("visits", 10)I,
           n..w XElement("person",
               new XElement("personal lnfo",
                   new XElement("name", "Sarah Kalter"),
                   new XElement("zip", 11215}),
               new XElement("favoriteDrink", "Boring Coffee"),
               new XElement("moneySpent", 110),
               new XElement("visits", 15))));
    return doc;
```

Save and load XML files

y,,,, can write an XDocument object to the console or save il to a lil,-, and you ",,,, load an XML file in", il:

```
XDocument doc = GetStarbuzzData();
Console.WriteLine(doc_ToString());
doc_Save("starbuzzData.xml"I;
XDocument anotherDoc - XDocument.Load("starbuzzData.xml");
```

The XDocu..ent object's Load() and Save() ..ethods read and write XML files. And its ToString() method renders everything inside it as one big XML docu..ent.

Query your data

Here', a simph- l.INQ query lhat queries the Starbuzz data using its XDocument:

```
var data _ from item in doc.Descendants("person")
    select new { drink - item.Element("favoriteDrink").vetue,
        moneySpent - item.Element("moneySpent").Value.
        zipCode - item.Element("personalInfo").Element("zip").Value J;
foreach (var P,in data)
    Console.WriteLine(p.ToString());
```

The Descendants() method returns a reference to an object that you ca... plug right into LINQ.

You already know that LINQ lets you call methods and use the·· as part of the query, and that works ..eally well with the Element() method

,\"d)'''' can do more complex queries too:

```
var zipcodeGroups - from item in doc.Descendants("person")
    group item.Element("favoriteDrink").Value
    by item.Element("persana11nf0"),Element("zip").Value
        into zipcodeGroup
        select zipcodeGroup;
foreach (var group in zipcodeGroups)
    Console.WriteLine("IOj favorite drinks in rir".
                group.Distinct().Count(). group.Keyl;
```

\ Element() ..eturns an XElement object, and you can use its properties to check specific values in your XML document.

Read data from an RSS feed

Y'" can do some pretty powerful things with LINQ to XMJ.. Here's a <illpl.- query to **read** artic**es from** our **blog**:

```
XDocument ourBlog - xpocument.Load("http://www.stellman-green...com/f....d");
Console.writeLine(ourBlog.Element("rss").Element("channel").Element("title").Value);
Var posts - from post in ourBlog.Descendants("item")
    select new { Title _ post.Element("title").Value,
        Date _ post.Element("pubDate"I.Valuej;
foreach (var post in posts)
    Console.WriteLine lpost.ToString());
```

The XDocu..ent.Load() ..ethod has several overloaded constructors. This one pulls XML data from a URL

Stick a button on a form, make sure you've got "using System.X..l.Linq;" at the top, type this query into its event handler, and check out what it prints to the console

We used the URL of our blog, Building Better Software http://www.stellma..-gree..e.com/

#2 Refactoring

Refactoring means changing the way your code i, ,rructured Wilh"ll1 changing its behav'iol, \Vhenever you write a complex method. you ,h, >llld take a few minute, ro step back and figure "ut h,ıw y›ll can change it s« that you makr- it easier to Ulld 'I,md, Luckily, the [IlE has some ı ery useful refactoring tools built in. T lıen· are all sorts of refactorings you can ,10– h are ""|<' we ,,,,. ofte" .

Extract a **method**

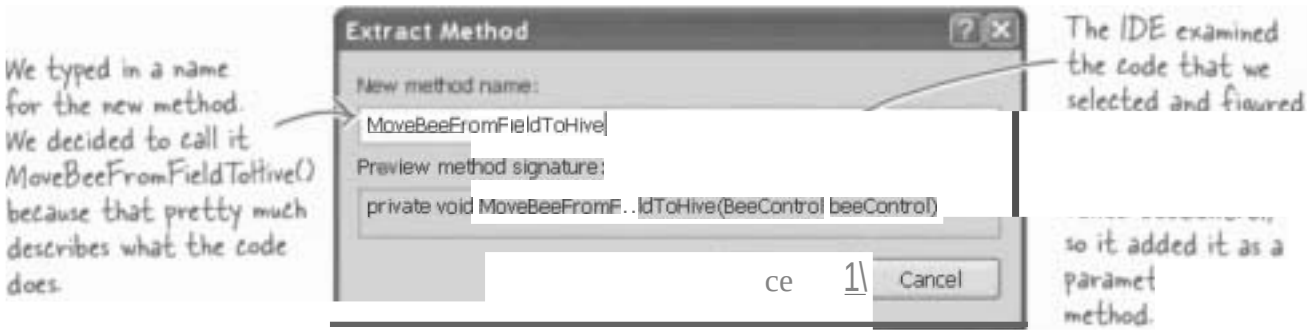
When we were writing rhe control-base<1renderer for {:hapr'l I:\. we originally ind"d,›c1this foreach loop:

These 1our lines move a BeeControl the Field form to the Hive form.

```
rcreecn (Bee bee in world.Bees) I
    beeControl = GetBeeControl(bee);
    if (bee.InsideHive) {
        if (fieldForm.Controls.Contains(beeControl))
            fieldForm.Controls.RemovelbeeControl> ;
            beeControl.Size = new Size(40, 40);
            hiveForm.Controls.Add IbeeControl);
            beeControl.BringToFrontO ;
    ) else if (hiveForm.Controls.Contain5(bee ontrol})
        hiveForm.Controls.RemovelbeeControl);
        beeControl.Size - new sreecao. 20);
        fieldForm.Controls.Add(beeControl);
        beeControl.BringTo Front(> ;
    )
    beeControl.Location = bee.Location;
```

And reese fou.. lines move a BeeControl from the Hive form to the Field form.

One of our h·,'h reviewers. Joe .Albahari, pointed o"r rhar this was a lillle hard 10read. He suggested that we .."tract those two four_lin.. blocks into **methods**. So we selected the Ii,,.r block. right-clicked "n ir. and selected "Refactor > Extract Method..,"- rhi, window popped "p:



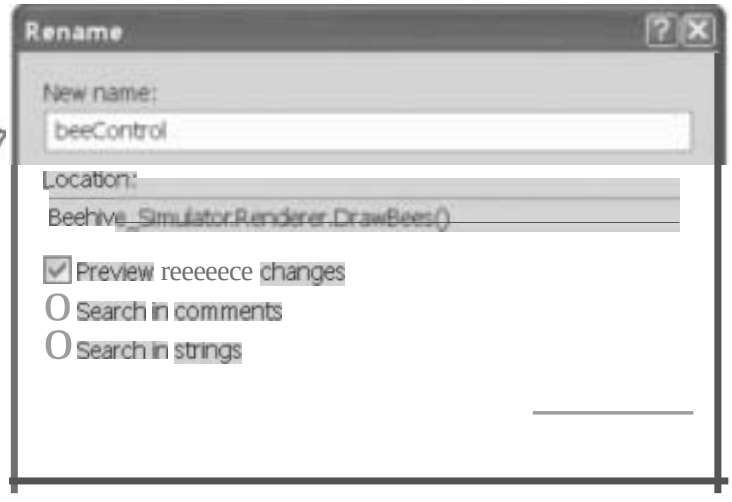
T lıe" we did the same thing for the "lher li'"r·li,ıe block, extracting il into a method tluu we named MI,ıveBeeFromHi",; li›Field(). 11ere's Jr,ıw that foreach I,ıop "",I"d up - it's " 1„1 ,·..,i,·r ro read:

```
foreach (Bee bee in world.Bees) {
    beeControl = GetBeeControl(bee);
    if (bee.InsideHive) {
        if (fieldForm.Controls.ccet.etns(beeControl)I
            MoveBeeFromFieldToHive(beeControl);
      else if (hiveForm.Controls.Contains(beeControl)I
        MoveBeeFromHiveToField(beeControl, bee);
    beeControl.Location - bee.Location;
```

Rename a variable

Back ill Chapter 3, we explained h,ıw choosing intuitive name for your classes, methods, fields and variables makes y‹>I'r cı„1,- a lot easier I" understand. The II) E can really help y‹>I, out when il come '0 naming thing ill your code. Just right-click on any class, variable, field, property, namespace, constant - prt-uv much anything that you can name—and choose "Refactor > Rename". \\',. did il with "beeControl" ill the code from the simulator. Here's what popped up:

Thi window lets you thoo" a new na..e for the ite.., If we rena..ed this, say, "Bobbo", then the IDE would go through the code and change every single occurrence of it to "Bobbo".



The IDE does a really thorough job of rena..ng If you rena..e a cla.., it'll change every state..ent that instantiates it or uses it. You can click on any occurrence of the name anywhere in the code, and the IDE will ..eke the change everywhere .. your progra....

Consolidate a conditional expression

Here's a neat wav to use the "Extract Method" feature. Open "l' anv program, add a l,ul1on. and add ,hi, code '0 its ,·,',u, handler:

```
private void buttonl_Click(object sender, gventArcs el {
    int value - 5;
    string text - "Hi there";
    if (value -- 36 II text.Contains("there"))
        MessageBox.Show("Pow!");
```

Select everything inside the ir statement: `value == 36 II text.Contains("there")`. T h,'" right-c·jj,k"" it and select "Refactor > Extract Method..: ·. Here's what pOp' "l':

Every conditional expression evaluates to a bool, so the IDE will create a method that returns a bool and replace the conditional test with a call to that method.



The expression uses two variables called value and text, so the IDE added parameters to the method using those names.

Not only will this make the code easier to read, but now you've got a new method that you can reuse elsewhere!

#3 SOllie of our favorite roolbox cOlllpollellts

This was a book about learning C#, not learning the in' and outs of the components lha, ship with .NET. Slili. we've got our favorites. „It I we thought we'd share a few of them with vou.

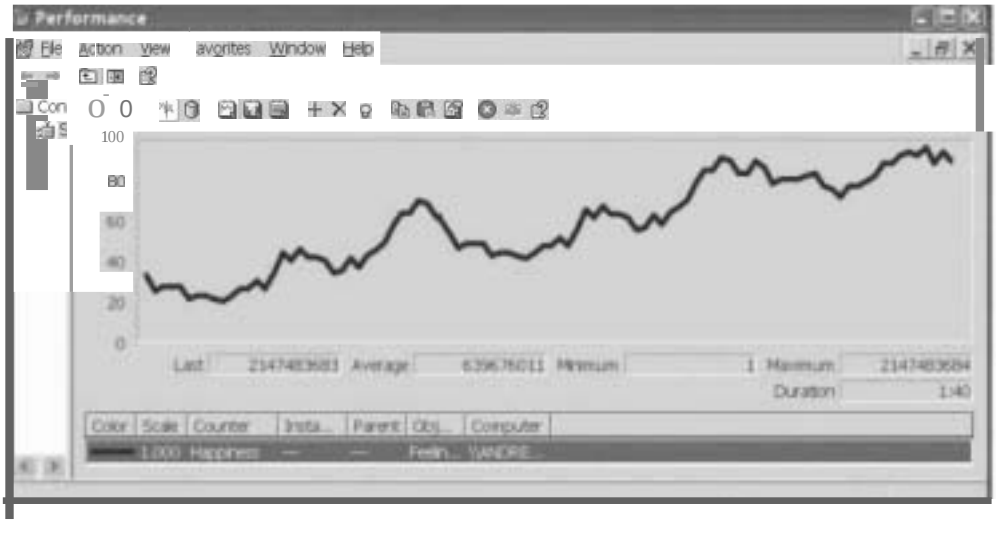
BackgroundWorker is one of those non-visual components (like Timer) that you can drag onto your form that does something really neat—it lets you easily build multithreaded applications.

Just drag one on your form and double-click on it (or, if you want, instantiate it—but don't forget to dispose it!). Then add an event handler to do the work that you want to run in the background.



Whe.. you call its RunWorkerAsync() method, it fires off the DoWork() ethod—but the catch is that it runs it i other thread That means it does its work in parallel with the other stuff your program', doing

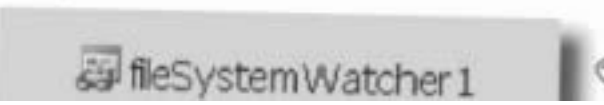
Your process ca.. report its progress back to r- by calling the BackgroundWorker's ReportProgress() method—that causes it to raise its ProgressCha.ged eve..t.



We set up this "Happiness" counter in the "Feelings" category, and fed it data using PerformanceCounter.

FileSystemWatcher pretty much does what it sounds like it does—it watches your filesystem to see if something's changed.

You set its Filter property to the type of file it should look for (like *.txt), set the Path property to the directory it needs to watch, and you can tell it to look in subdirectories with its IncludeSubdirectories property.



Once you've set it up, it watches the directory for any _ or changed files. As soon as a file's added, changed, or deleted, it raises its Changed eve..t. It's also got Created, Deleted, and Rena...ed eve..ts to do more specific tracking

When the FileSyste...Watcher raises an event, it passes information about the file that was changed (or added, or deleted, etc.) using a FileSystemEventArgs object

When you're writing a program that runs continuously, it's really useful to monitor it. And Windows ships with a nifty tool called Performance Monitor (perfmon.exe) that lets you monitor processes.

The PerformanceCounter component lets you make information about your program available to the Windows performance monitoring system. Use Increment() and Decrement() or set its RawValue property. As soon as you do, you can see the data in the Performance Monitor.



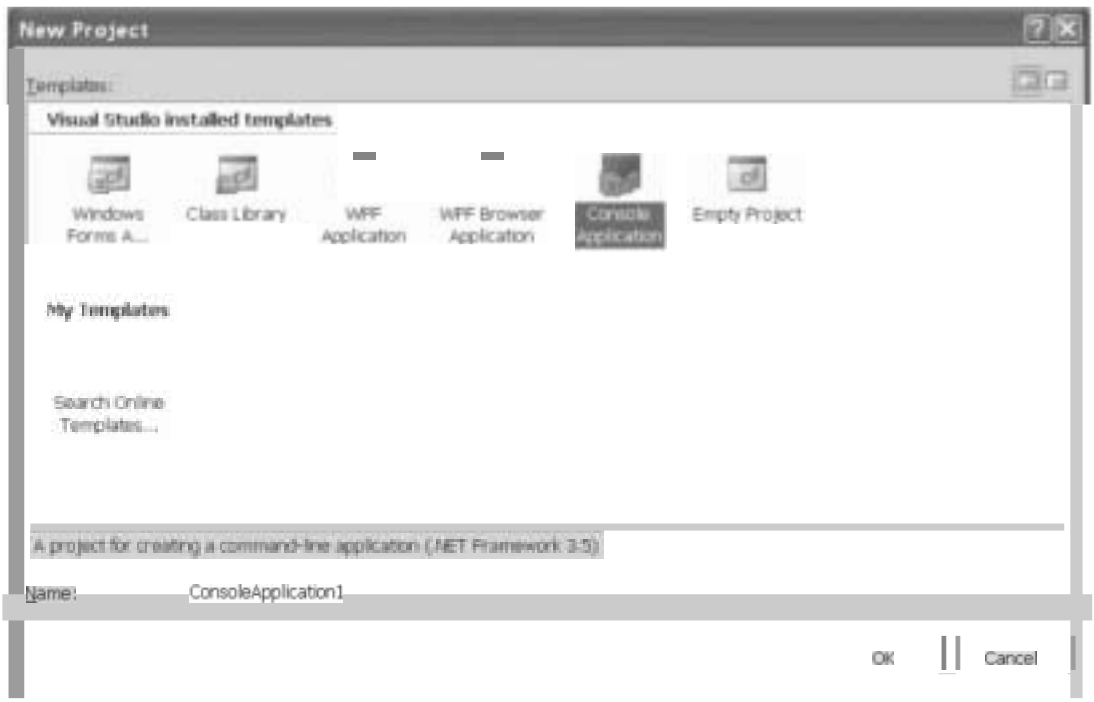
Windows keeps its performance counters in categories, so you'll need to create a category—there are methods in System.Diagnostics that let you do that. Then just hook up your PerformanceCounter to the category you created, and start sending diagnostic info to your heart's delight!

#4 COll1sole Applicatiol1s

MI,ist (:# books start with console applications \Ve th,ught that was boring. It's a I,,1 more satisfying 10 h"ild programs lhal l,ok like, well...that loc,k hke anything at all. .\nd that's what a console program isn't. IItUsometimes you do need 10 write a u"nm,"ul-lin,- application. Luckily, it's pretty straightforward. II,re s how:

O **Create a Console Application project**

Any project can be a console apl'li,',uiou, (;", to a project, select "Properties' from the Project menu, and change lhe "()utput type' I' "Console '\I'pli""I;IIII··. IIul it's easier 10 create "",' from scratch.



O **The IDE only adds one file—Program.cs**

.\nd it's got an emptv entry point... and 1hat's it.

```
class Program

    static void Main(string[] args)
```

Here's a little project for you! Take the hex dumper you built in Chapter 9 and turn it into a console applicaiton. Have it read a filename that you pass it on the command-line and print it out as a hex dump. Have it take data from standard input (using the Console ReadLine() and dump that data out as a hex dump. Then look up the Unix command "od" and see if you can reproduce it in C#.

3 Use the args parameter for command-line arguments

Your entry point takes one parameter, a string array called `args` that contains the command-line arguments. You already know how I" use the Console.WriteLine() method —there are some other useful console nlt-thod,. including ReadLine() and ReadKey().

```
class Program {
    static void Main(string[] args)
        Console.WriteLine("I got {0} arguments", args.Length);
        for (int i = 1; i <= args.Length; i++)
            Console.WriteLine("Argument #{0} is {1}", i, args[i - 1]);
        Console.Write("Enter some text: ");
        string input = Console.ReadLine();
        Console.WriteLine("You entered: {0}", input);
        Console.WriteLine("Press any key to end...");
        Console.ReadKey();
```

0 Debug your program in a console window

When vou debug your program, the **IDE** 1'01" Ill' a console window. The ReadLine() and ReadKey() methods get their illl'"t from that window— j"'l type into it. .\][(1 instead "f writing 1<,the ()lll pUl windo"'; a console appli,'al;'>l, writes 10this console window instead. You can set the rommand-line arguments in the "Debug" page of the Project Properties window



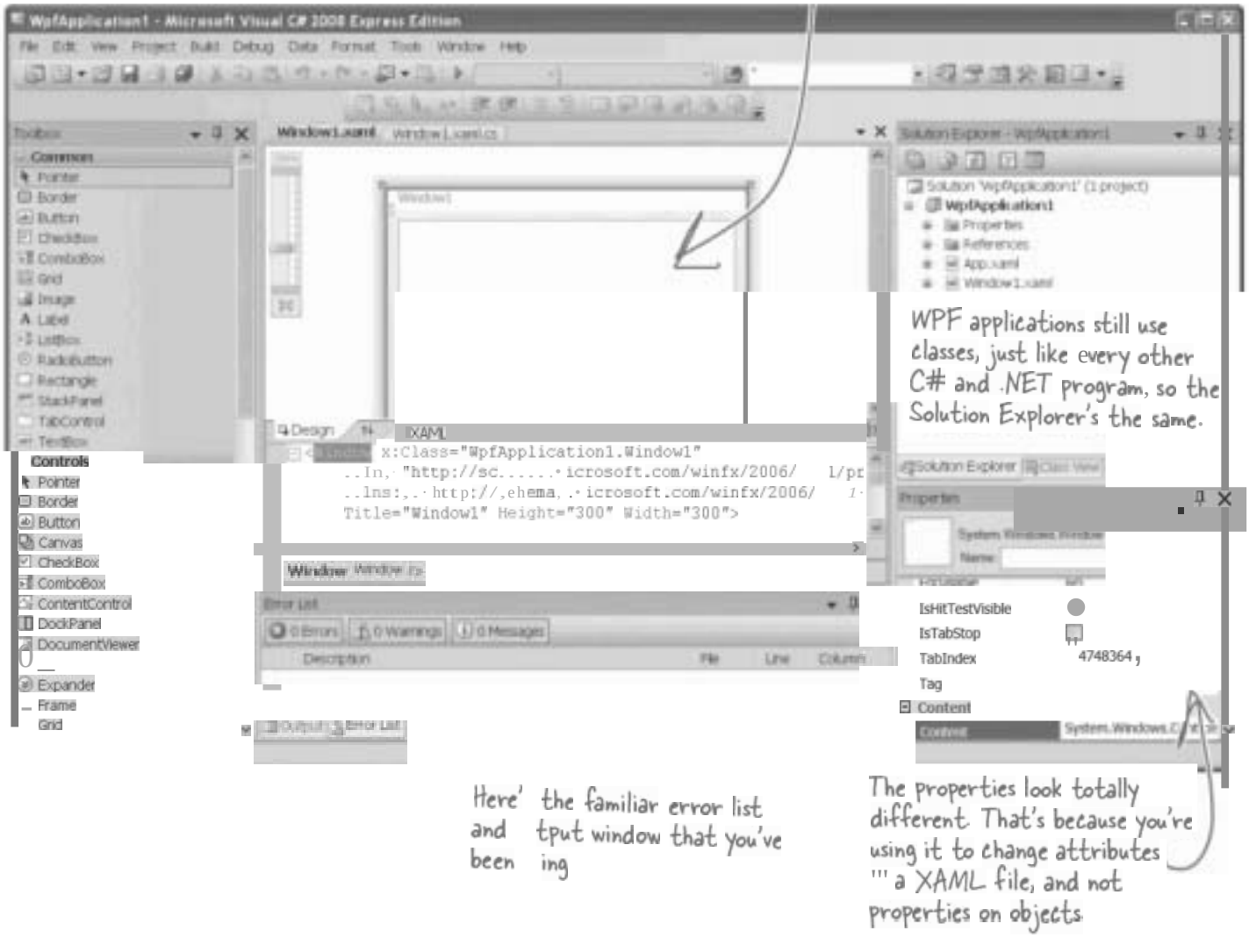
#5 Windows Presentation Framework

Windows Presentation Framework, "I' WPF, is Microsoft's next-generation platform for building visual applications. It's pretty amazing— it has XML-based layout, scalable controls. a totally new system fo ir controls. 2-J) and 3-D graphics and "nim" li"!l, text flow and d"nllllll'lli formatting, and there's even a cross-platform web browser plugin tha uses it.

Unfortunately, while WPF is a really co,1 and highly capable technology, it's not a particularly good tool for teaching C#. And that was 0 \11' goal - getting (:# concepts into your braiu as quickly and easily as possible.

Take a second and create a new WPF application Just create a "" \l project using (h,- IDE, but d"n't create a new \Vindow Forms .Application project. Instead, **selec***t* WPF Application. You'll immediately notice a difference in the JOE:

The biggest di-II erence you'll notice is that the form designer looks nothing like the one you ..e used to. We'll take a closer look at it i" a ..,nu[e

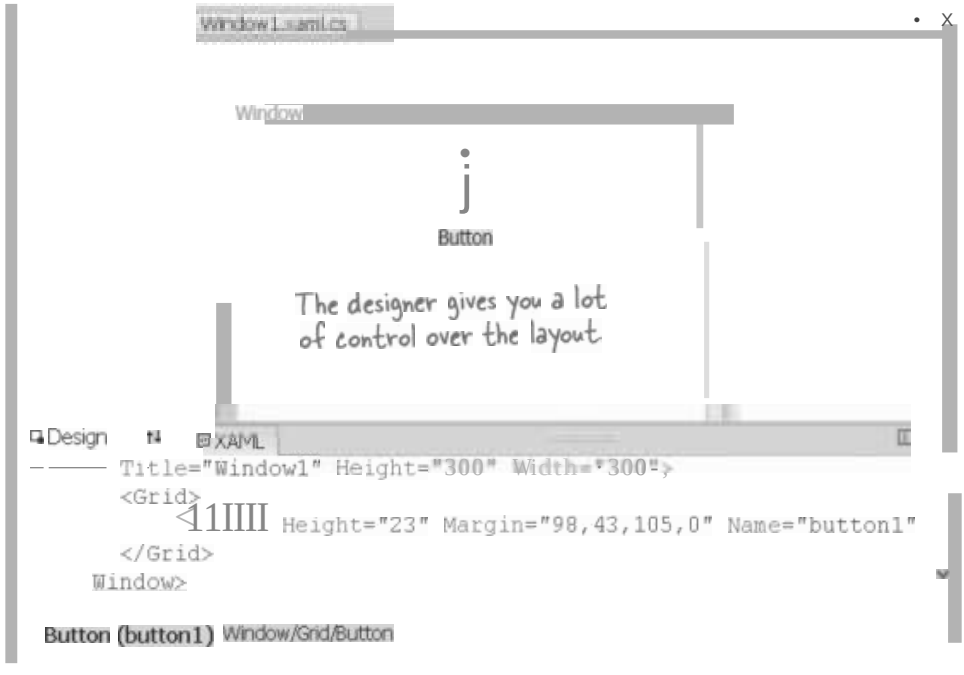


WPF applications still use classes, just like every other C# and .NET program, so the Solution Explorer's the same.

Here' the familiar error list and tput window that you've been ing

The properties look totally different That's because you're using it to change attributes "' a XAML file, and not properties on objects

Drag a hunon onl of Ih.. loolhox **and** onlo th.. form. If this were a Window Forms applicati,on, Ih,' I1)E would add code I" Fc'nn 1.1)esigner.cs I" add a control t" Ih,' Form I object. But \VPF is different it uses an XM1L-based language called XAM II. 10 define how Ihe use.. interface is laid " nt.

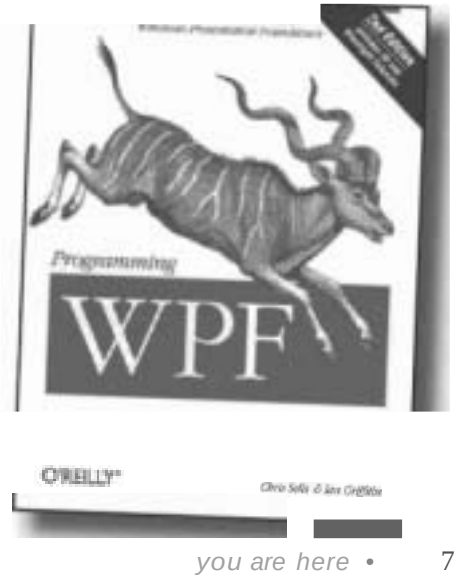


Go I" the XM II. editor and ,,,Id a second bUll"" by typing the bol d line below inlo the X.\ .\II. editor. You'll notice h,)w the 11)E's IntelliSense d,)es a go(,d j,)b "I' helping you enter all the X.\II. tags.

```
<Grid>
    <But ton Height="23" Margin="98,43,105,0" Name="button1"
                 VertLeaIAlignment="Top" CIick="buttonl_CIick"»But.toncIButton>
    <Button Height="23" Margin="5.5,100,20" Name="button2"
                 VerticalAlignment="Top" Click="button2_Click">Another button</Button>
</Grid>
```

\\·hc1, you get In th.. "**Click="button2_Click"**" part of the Ii"", don't type i" the "allH' "I' the event handler. Instead, use tl,,-IntelliSense window that pop up'" tell the III E to add a new ,,,',-nl handler. As soon as you finish the lin-. you'll see a new hUll"n appear in the- designer. Switch over I" the Window1.xaml.cs tab, and you'll fmd a a new I'tlllon1_CIi,k method Ih..re.

That's all Ih,-\\'1'1' and XAML that we can include here. But now Ihat you've got the t," ,I, to start learning about WPF, we definitely n-cununc-ml that you take a 1," ,k at ***Programming WPF*** by Chris Sells and Ian Griffiths. It's available fr("n the ()'Reilly website: hnp:/ /www.o....il1y.com/ .



Did you kllow that C# alld the .NET Framework call...

- Give you Much More power over your data with advanced LINQ querieS?)erialize objects to aN XML file?
- Access websites and other network resources using built-iN classes?
- Let you add advanced eNcryptioN and Security to your programs?
- Create cOMplel< multithreaded applicatioNS?
- Let you deploy your classes So that other people caN uSe theM?
- USe regular expressions to do advanced text Searching?



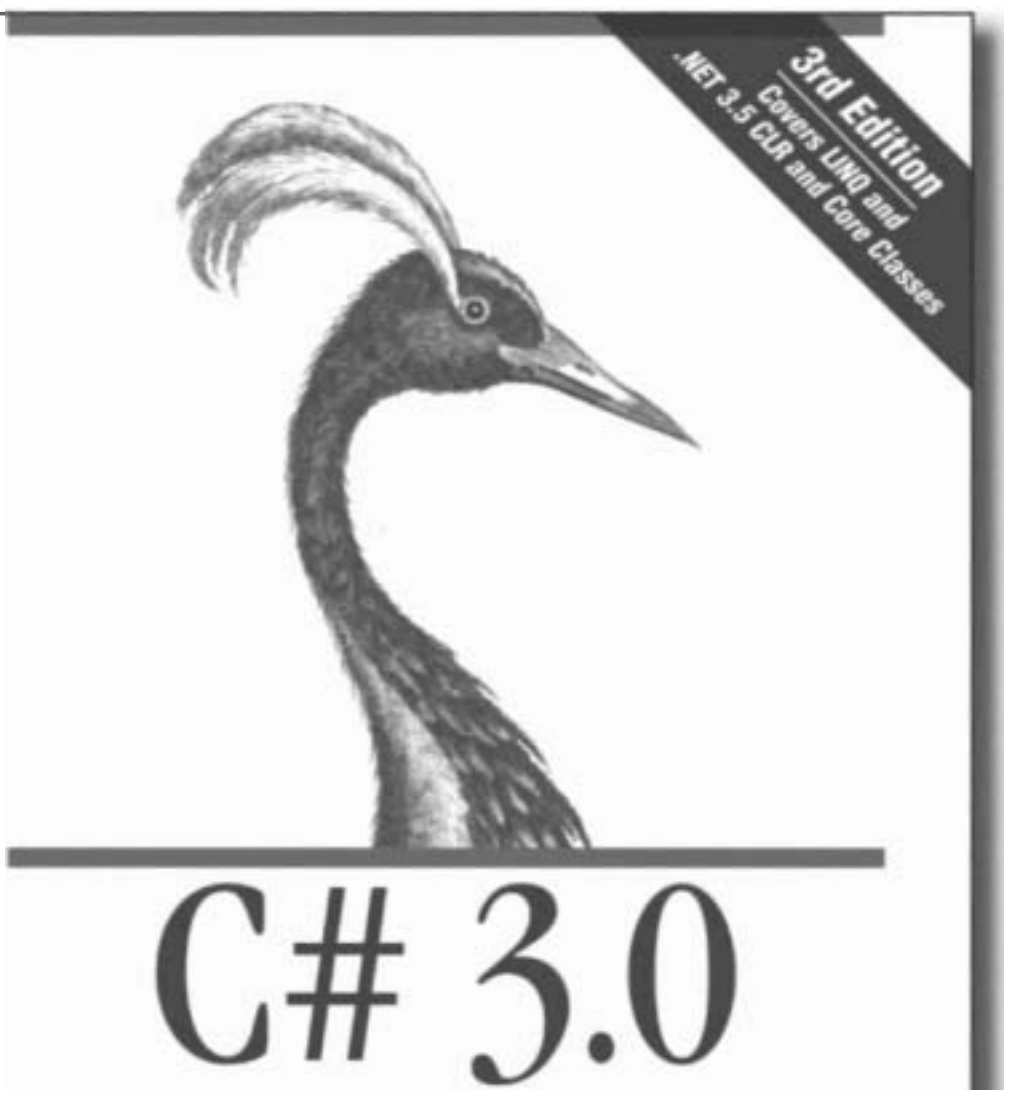
C 0

I had no idea! Where can I learn more?

There's a great book that explains it alii

It's c„11ed ***C# 1.0 in a Nutshe11*** by Joseph Albahari and Ben .\Ihahal'i. and it's a thorouuh guide to everything that C# has to offer. You'll learn about advanced C# language features you'll see "II or the essential .NET Framework classes and tools, and you'll **1**..arn **more** about what's really going on ulHl"r the hood "I'(:# .

Check it out .u **http:/**l www. o**reilly.com/**.



Joseph Albahari helped us out a whole lot by giving this book a really thorough tech review. Thanks so much for all your help, Joe!

Index

Symbols

!= operator 68

&& operator 68, 77

*= operator 64, 136

* operator 64

+= operator 64

+ operator 64
  casting 129

-= operator 136

- operator 64

// (slashes) 66

/ operator 64

< operator 68

== operator 67, 68

= operator 64, 67

> operator 68

@ in front of filenames 389, 401

\n 15, 66, 401

\t 401

|| operator 68

A

abstract classes 278–285
  Fireside Chat 284–285
  usefulness 279–280

abstraction as principle of OOP 288

abstract keyword 281

abstract methods 278, 281

access modifiers 273–274
  internal 273
  private 273
  protected 273
  protected versus private or public 276
  public 273
  scope 274
  sealed 273

Adventure Game program (see labs, #2 The Quest)

Albahari, Joe 706

allocate, defined 405

allocated resources 405

Anatomy of a program 52–53
  .NET Framework 52
  classes 52, 53
  methods 52, 53
  namespaces 52
  parameters 53
  statements 53
  using lines 52

Anatomy of a query 660

AND operator 68

animal inheritance program 216–222

animation 566–567
  building control 573
  double buffering 608–613

AppendAllText( ) method 400

Appliance project 266–270
  Appliance class 266
  downcasting 268
    interfaces 269
  upcasting 267
    interfaces 269

application design (see design)

applications
  compiling 47
  console 710–711
  debugging 47

applications (*continued*)
deploying 37, 38
running 36
running in IDE 36
architecture 531
args parameter 711
arrays 148–149, 262
deck of cards 315–316
finding length 149
versus Lists 318–320, 325
assemblies 273
attributes 421

B

BackColor property 51
BackgroundWorker 708
backing field 193, 198
Baseball Simulator project 484–501
callbacks 507–509
Fan class 494–497
Pitcher class 494–497
subscription and public events 505
base classes 214, 217
colon (:) 222
constructors 233
extending 221
subclasses accessing with base keyword 232
upcasting 267
using subclasses instead 227
base keyword 232, 275
Beehive Simulator project 239–249, 252–265
adding new form 544
AnimateBees( ) method 613
animating with controls 566–567
building control 573–575
images 574–575
timer 575
Bee class 524–525
Go( ) method 533–535, 542–543
BeeControl 574–579
animating bees on form 580–581
creating button to add to form 576
creating from UserControl 578
disposing 577
implementation 579
ResizeCells method 591
BeeState enum 525–527
Bitmap class 593
building form 241
building Worker and Queen classes 241
class hierarchy with Worker and Queen classes 253
collection of bees 555
Color.Transparent 589
creating Bee class 246
DateTime class 548
Dictionary objects 569
double buffering 609–613
drawing picture on form 596–597
encapsulation 537
extending through inheritance 245–249
fixing transparency problems 601
Flower class 520–522
for loops 537
forms
adding hive and field 582
clearing out all controls 583
FieldForm 586–587
FormBorderStyle property 582
HiveForm 586–587
Location property 582
Reset button 587
Graphics object 592
Hive class 529–530
adding methods 532–533
exceptions 539
updates 540–541
honey production 246
interfaces 254–263
inheritance 263
references 260–261
making Queen class inherit from Bee class 247
making Worker class inherit from Bee class 247
NectarHarvested variable 523
number of bees 533
object model 518

OutOfHoneyException 468
overview of what's to be added 519
Paint event handler 602
performance issues 589–591
Point object 539
printing 616–619
code for the Print button 619
event handler for the Document's PrintPage event 618
PrintTableRow( ) method 618
read-only automatic properties 523
RemoveAllControls( ) method 583
removing dead flowers and retired bees 537
Renderer 568–569
Renderer class 583–585
DrawBees( ) 584
DrawFlowers( ) 584
GetBeeControl( ) 585
MoveBeeFromFieldToHive( ) 585
MoveBeeFromHiveToField( ) 585
RemoveRetiredBeesAndDeadFlowers( ) 585
ResizeImage method 591
resizing images 592–593
Show( ) method 587
timers 546
adding to program 548
disposing 577
events and delegates 547
Render( ) method 583
ToolStrip control, adding Open, Save, and Print 559
updating form to instantiate bees 247
using World to get statistics 545
World class 534–535
code 536

behavior 12

Behind the Scenes
how forms and controls repaint themselves 605
LINQ using extension methods 657
The stack vs. the heap: more on memory 641
Unicode 424
visual components 576

binary and decimal, converting between 125

binary files 424
comparing 429
hex dump 431
working with 431
writing 427

BinaryFormatter 420
Deserialize( ) method 420, 423
Serializable attribute 423
SerializationException 454

BinaryReader 428

BinaryWriter 427

Birthday Party project 206–214
BirthdayParty.CalculateCost( ) 213
BirthdayParty class 207
CakeWriting method 212
CalculateCost( ) method 212
inheriting from Party class 234–238

Bitmap class 593

blank space 66

boilerplate code 44

bool type 63, 64, 124, 126

bound 32

boxed objects and structs 642, 644

boxed struct 645

breakpoints 450
knowing where to put 452

BringToFront( ) method 581

Brush object 606

Build menu 47

Build Solution 47

built-in features 55

Bullet Points
delegates 509
event handlers 509
exception handling 471
LINQ query statements 664
Lists 322

Bullet Points (*continued*)
reference variables 154
statements 73
try/catch blocks 471
types 154
buttons 44
adding code to interact with objects 113
adding to form 51, 112
BackColor property 51
Name property 51
Size property 51
Text property 51
byte arrays 401
moving text around in 426
byte order mark 434
byte type 124, 126

C

C#
what you can do with 714
what you get with Visual Studio and 2
why you should learn 2
C# 3.0
automatic properties 195
object initializers 115
Calculator program 474–475
temporary solution 475
callbacks 507
versus events 510
call stack 453
camelCase 201
Candy Control System 102–108
capitalization 201
Captain Amazing 622–626, 635, 636, 645, 651
casting 128–130
+ operator 129
automatic 130
wrapping numbers 129
catch blocks 455, 457
followed by (Exception) 460
following in debugger 458–459
multiple 466
with no specified exceptions 462
chaining 491, 499
Character Map 424, 425
char type 125, 126
checkbox 75
class diagrams 90, 104, 106
moving up, not down 231
private fields and types 240
classes 52, 53, 73
abstract (see abstract classes)
adding 56
collection 317
concrete 278
copying 90
creating example 111
curly braces 66
declaration 54
defining 66
designing intuitive classes 116
finding out if class implements specific interface 262
instances (see instances)
internal 273
looking for common 219
members 273
message about adding components to my class 579
MessageBox 56
multiple in same namespace 61
naming 102–103
natural structure 104
never instantiated 277
organizing 106
partial (see partial classes)
private 273
protected 273
public 273
sealed 273
serializable 421

similarities between 116
static 97
subscribing 489
using to build objects 92
versus structs 644
why some should never be instantiated 280
you can't inherit from 647
class hierarchy 215, 220
Hive Simulator 253
Clone class
implementing IDisposable 630, 631
CLR (Common Language Runtime) 47, 153
code
avoiding duplication 217
blocks 73
boilerplate 44
copying 90
looking at auto-generated 15
renaming things in code 707
repeating 213
similar 214
unwanted code from IDE 11
using IDE to help write code 48–49
collection initializers 326–327
collections 317, 555
Dictionary (see dictionaries)
generic 325
LINQ 556–558
List (see lists)
performing calculations on 662
Queue (see queues)
Stack (see stack)
using join to combine two collections into one query 671, 672
versus tables 661
colon (:) 222
colon operator 256
Color.Transparent 589
colors 76
columns 20
command-line arguments 711
comments 66
CompareTo( ) method 329
compiler 47
compiler errors
interfaces 254
troubleshooting 49
compiling application 36, 47
compound operators 136
concatenation 130
concrete classes 278
conditional expressions, consolidating 707
conditional operators 68–70
conditional tests 68–70, 73
configuration files 37
Console.WriteLine( ) method 194
console applications 710–711
console window, debugging in 711
constructors 197, 198, 199
base class 233
building new with switch statement 415
exceptions in 459
overloaded 313
subclasses 233
ContactDB.mdf 18, 29, 31
ContactDBDataSet 31, 32
ContactDBDataSet.Designer.cs 31
ContainsKey( ) method 581
controls
adding code 564
altering reexisting 587
animating Behive simulator 566–567
as objects 565
bound to database 32
clearing out all on forms 583
custom 564
animation 573
disposable objects 577
database-driven 32–33
disposing 577, 579

controls (*continued*)
how forms and controls repaint themselves 605
redrawing themselves 602
removing 564
visual display elements 570
Controls collection 565
count ++ 64
count -- 64
count = 64
Create( ) method 400
CreateDirectory( ) method 400
CryptoStream 394
curly braces 66, 73
curly brackets 58, 111
single-line blocks 212

D

data
pulling data from multiple sources 656
storing categories of 310
database-driven controls 32–33
database diagram 26
saving 27
Database Explorer 18, 675
databases 3
adding table 20
adding to project 18
connecting forms to 17, 30
(see also data source)
connecting LINQ to SQL database 674–675
entering data 28–29
LINQ 673
LINQ querying SQL database 677
multiple tables 26
SQL 18, 19
SQL Server Express 7
data source
adding new 30
configuring 31
database-driven controls 32
Data Source Configuration Wizard 18
data storage 7
data types 20
generic 325
DateTime class 548
debugger
Break All button 449
Bullet Points 471
catch blocks
followed by (Exception) 460
following flow 458–459
multiple 466
with no specified exceptions 462
Continue button 449
exploring delegates 503
finally block 460
knowing where to put breakpoints 452
Restart button 449
Show next statement button 449
Step into button 449
Step Into command 450
Step out button 449
Step over button 449
try blocks 458–459
unhandled exception window 452
uses for 457
Watch window 449, 457
running methods in 452
(see also exception handling)
debugging 16, 47, 449
console window 711
Excuse Management program 450–451
Debug menu 47
decimal and binary, converting between 125

decimal type 125, 126
declaration 54
delay 76, 79
delegates
  Bullet Points 509
  defined 501
  exploring in debugger 503
  hooking up to one event 507–509
  in action 502–503
  multiple events 499
Delete( ) method 400
deploying application 37, 38
deployment package 7
design 531
  intuitive applications 34–35
  intuitive classes 116
  making code intuitive with class and method names 102–103
  professional looking applications 35
  user's needs 5
destructor 628
dialog boxes 398–400
  as objects 399
  customized 401
  popping up 397
DialogResult 397–399
  excuse management program 410
dictionaries 335–337
  Add( ) method 335
  adding or removing items 336
  ContainsKey( ) 335
  getting list of keys 336
  getting list of values 336
  keys 335
  keys and values 337
  looking up values using keys 336
  renderer 580
Dictionary objects 569
Dinner Party Planning project 174–185
  CalculateCostOfDecorations( ) method 184
  cost estimate 175
  DinnerParty class
    class diagram 176
    exercise solution 178–179
  fixing calculator 203–204
  inheriting from Party class 234–238
  numericUpDown control 183
  recalculating new individual costs 183
directories
  creating new 400
  deleting 400
  getting list of files 400
Dispose( ) method 406, 577
  finalizers 632, 634
  making object serialize in 633
  using statement 630–632
DivideByZero error 443
DivideByZeroException 443, 448
dividing any number by zero 443
DogCompetition class 312
double buffering 608–613
double type 125, 126
downcasting 268, 269
  failure 270

E

encapsulation 183–204, 435, 644
  as principle of OOP 288
  automatic properties 195
  better 276
  defined 185
  example 192
  ideas for 191
  properties 193
  Renderer 568
entry point 55, 58, 231
  changing 56
enumeration 310–311
enums 311–315
  big numbers 312
  representing numbers with names 312
  versus Lists 325

equal signs 67
error handling 462
Error List 49, 58
errors
  avoiding file system errors with using statements 406
  compiler errors and interfaces 254
  DivideByZero 443
  invalid arguments 131
  You must rebuild your project for the changes to show up in any open designers. 579
escape sequence 66
EventHandler 488, 491
  using methods that do match others defined by EventHandler 491
event handlers 177, 188, 485
  adding 491
  automatic 492–493
  Bullet Points 509
  excuse management program 408
  hooking up 498
  how they work 486–487
  printing 614
  returning something other than void 491
  types of 491
event keyword 488
events
  callbacks 507–509
  connecting senders with receivers 500
  defined 485
  delegates 499
  forms 498
  how they work 486–487
  naming methods when raising events 490
  raising 490
  raising events with no handlers 490
  reference variables 500
  subscription and public events 505
  versus callbacks 510
  (see also event handlers)
exception, defined 444
exception handling 439–482
  Bullet Points 471
  catch block 455, 457
  catching specific exception types 473
  DivideByZeroException 443, 448
  dividing any number by zero 443
  exceptions in constructors 459
  exceptions versus unhandled exceptions 462
  Excuse Management program 477–478
  FileNotFoundException 462
  finalizers 635
  FormatException 448
  handling, not burying 474
  handling versus fixing 475
  IDisposable interface 472
  IndexOutOfRangeException 448
  IOException 460
  NullReferenceException 443
  OverFlowException 448
  program stopping with exceptions 462
  SerializationException 453, 460
  simple ideas for 476
  specifying particular kinds of exceptions 462
  spotting exceptions 445
  throwing and catching exceptions 467
  try block 455, 457
  unexpected input 456
  unhandled exceptions 452
  using exceptions to find bugs 447
  using statement 471
  why there are so many exceptions 445
  (see also debugger)
Exception objects 444, 445
  inheriting from Exception class 448
  Message property 466
  using to get information about the problem 465
Excuse Management project 407–411
  binary files with serialized objects 436
  building the form 408
  code problems 453
  debugging 450–451
    Random Excuse button 461

DialogResult 410
event handlers 408
exception handling 477–478
Folder button 408
Random Excuse button 411
Save As dialog 410
solution 410–411
unexpected user behavior 446–447
using debugger to follow try/catch flow 458–459
executable file 47
executing application 36
Exists( ) method 400
Expand Tables 28
extension methods 646, 647
LINQ 657
strings 648

F

Farmer class 192–198
constructors 197
fully encapsulating 195
testing 194–195
features 3
built-in 55
fields 98
backing field 193
initializing public fields 196
interfaces 255
lining up 34
masking 198
method's parameter has same name as a field 198
private 185–188
constructors 197
declaring 201
public 191
versus methods 98
versus properties 276
with no access 188
File.Create( ) 429
File.OpenWrite( ) 429
File.ReadAllBytes( ) 425, 426, 434
File.ReadAllLines( ) 434
File.ReadAllText( ) 434
File.WriteAllBytes( ) 425, 426, 434
File.WriteAllLines( ) 434
File.WriteAllText( ) 434
File class
Close( ) method 434
ReadAllText( ) method 403
static methods 434
versus FileInfo class 434
WriteAllText( ) method 403
FileDialogs 403
FileInfo class 400
versus File class 434
filenames, @ in front of 389
FileNotFoundException 462
files
appending text to 400
finding out if exists 400
get information about 400
reading from or writing to 400
(see also streams)
where Visual Studio stores them 16
writing 412
FileStreams 387, 388
BinaryWriter 427
StreamWriter 389
versus StreamReader and StreamWriter 434
FileSystemWatcher 709
Filter property 398
finalizers 628
Dispose( ) method 632, 634
exceptions 635
fields and methods 635
garbage collection 629–631
references 632
stability 632
when they run 629
finally block 460

Fireside Chats
abstract classes 284–285
Dispose( ) method and finalizers 634
Five Minute Mystery
The Case of the Golden Crustacean 506
mystery solved 511
flickering images 607
float type 124, 126
assigning 129
foreach loop (lists) 321, 358
for loops 65, 69
Form1.cs 8
Form1.Designer.cs 8, 45, 50
changing control properties 12
Form1.resx 14
Form1 form, programs without 231
FormatException 448
FormBorderStyle property 582
Form Designer 3
forms
adding buttons 112
adding method 113
adding variables 112
as objects 152–153
connecting to databases 17, 30
(see also data source)
CreateGraphics( ) method 594
database-driven controls 32
events 498
how forms and controls repaint themselves 605
OnPaint method 605
Paint event 605
PaintEventArgs 605
redrawing themselves 602
Refresh( ) method 605
(see also Beehive Simulator project, forms)
frames versus turns 549
from clause 664, 667
FromImage( ) 606

G

garbage collection 140, 153, 635
finalizers 629–631
GDI+ 594–595
generic collections 325, 355
generic data types 325
get accessor 193, 199
interfaces with get accessor without set accssor 259
GetFiles( ) method 400
GetLastAccessTime( ) method 400
GetLastWriteTime( ) method 400
Go To Definition 405
GPS navigation system 87
graphical user interface (see GUI)
graphics
drawing picture on form 596–597
how forms and controls repaint themselves 605
Rectangle 597
using keyword 606
Graphics object 592, 594, 606
CreateGraphics( ) method 595
DrawBee( ) method 601
DrawCircle( ) method 595
DrawString( ) method 595
FillCircle( ) method 595
Invalidate( ) controls 605
Paint event handler 602
printing 614
Update( ) method 605
green arrow button 16
GroupBox control 207, 606
group keyword 667, 668
GUI (Graphical User Interface) 94
labs, #1 A Day at the Races 170
guys (Two Guys project) 110–115, 117–118
building form
adding a method 113
adding buttons 112

adding code to interact with objects 113
adding variables 112, 113
creating Guy class and two instances 110
sample code 111
GZipStreams 387

H

heap 100, 101
garbage collection 140
structs 637
versus stack 641
Hebrew letters 425
hex dump 431
StreamReader and StreamWriter 433
using file streams to build hex dumper 432
hexadecimal 431
working with 432
hierarchy 215
defined 221

I

IClown interface 258
access modifiers 274–275
extending 271–272
IComparable interface 329
IComparer interface 330
complex comparisons 332
creating instance 331
multiple classes 331
SortBy field 332
IDE
auto-generated code 73
behind the scenes 14
buttons 44
changing names of files 11
changing things in 50
compiler errors, troubleshooting 49
creating new projects 8
Error List 49
green arrow 16
helping users code 48–49
importing images 14
making changes in 45
New Project window 8
Properties window 44
renaming things in code 707
Reset Window Layout command 11
running program in 36
SQL statements 19
stored procedures 19
unwanted code 11
using tabs to switch between open files 48
visual tools 73
what it does in typical application 44–45
what the IDE automates 2
where data is stored 29
Windows Forms Application project 44
XAML 713
IDE toolbar
green arrow button 16
Save icon 15
Stop Debugging button 16
IDisposable interface 405, 473, 630
avoiding exceptions 472
IDs
auto-generated 21
unique 20
if/else statements 67
if statements 131
images
drawing picture on form 596–597
fixing transparency problems 601
flickering 607
performance issues 590
Rectangle 597
resizing 592–593, 606
TrackBars to zoom an image in and out 603–604
images, importing 14
index (arrays) 148–149
IndexOutOfRangeException 448
infinite loops 71

inherit, defined 215
inheritance 213–250
  as principle of OOP 288
  avoiding duplication of code 217
  classes you can't inherit from 647
  class hierarchy, Hive Simulator 253
  class that contains entry point 231
  looking for common classes 219
  multiple 286
  passing instance of subclass 231
  subclasses 225–226
  (see also interfaces)
InitialDirectory propery 398
initialization 115
InitializeComponent( ) method 198
installation, testing 39
instances 93
  creating 94
    example 111
    heap 101
    static keyword 99
  defined 93
  fields 98
  keeping track of things 98
  non-static methods 97
instantiation, interfaces 260
int 63
integers, using in code 137
IntelliSense window 577, 595
  CreateGraphics( ) method 606
interface keyword 255
interfaces 254–276
  colon operator 256
  compiler errors 254
  containing statements 270
  downcasting 269
  easy way to implement 270
  example code 258
  fields 255
  finding out if class implements specific interface 262
  get accessor without a set accssor 259
  implementing 257–258
  inheriting from other interfaces 263
  is keyword 262, 265
  like contracts 270
  naming 254
  new keyword 260
  object references versus interface references 276
  properties 255
  public 255
  public void method 259
  references 260–261
    why use 276
  upcasting 269
  void method 258
  why use 270, 276
internal access modifier 273
int type 124, 126, 127
  assigning value 137
  declaring 137
invalid arguments error 131
IOException 460
is keyword 262, 265

J

join clause 671, 672, 677, 678

L

labels 75
  lining up 34
labels for objects (see reference variables)
labs
  #1 A Day at the Races
    application architecture 168
    Bet class 167
    Bet object 169
    Betting Parlor groupbox 171
    dogs array 168

finished executable 172
Greyhound class 166
Greyhound object initializer 166
GUI 170
Guy class 167
Guy object 169
guys array 168
PictureBox control 166, 168, 170
RadioButton controls 168
this keyword 167
#2 The Quest 363–384
Bat subclass 377
BluePotion class 380
Bow subclass 379
Enemy class 376
Enemy subclasses 377
form, bringing it all together 381–383
form, building 366–367
form, UpdateCharacters( ) method 382
Game class 370–371
Ghost subclass 377
Ghoul subclass 377
IPotion interface 380
Mace subclass 379
Mover class 372–373
Mover class source code 373
Player class 374
Player class Attack( ) method 375
Player class Move( ) method 375
RedPotion class 380
Sword subclass 379
using objects 368–369
Weapon class 378
#3 Invaders 681–702
additions 701
animation timer 687
architecture 684–685
designing the form 686–690
Form object 684
Game class 692
Game class, filling out 694
Game class methods 693
Game object 684–685
Game object's Draw( ) method 691
gameOver event 689
game timer 689, 690
graphics 691
Invader class 696
Invader class methods 697
KeyDown and KeyUp events 688
LINQ 695
movements 683
Paint event handler 691
PlayerShip class 698
PlayerShip object 685
Shot objects 699
Stars class 700
Stars object 685
types of invaders 683
line break 15, 66
LINQ (Langauge INtegrated Query) 556–558
101 LINQ Samples 663
combining results into groups 667, 668
connecting to SQL database 674–675
databases 673
extension methods 657
from clause 664, 667
Invaders lab 695
modifying items 662
.NET collections 657
orderby clause 664, 667
performing calculations on collections 662
pulling data from multiple sources 656
queries 658, 663
querying SQL database 677
query statements 664
scouring comic collections 659
select clause 664
Take statement 664
to XML 704–705
reading RSS feed 705
using join to combine two collections into one query 671, 672
versus SQL 661
where clause 664
LINQPad 679
lists 317–334
Bullet Points 322
CompareTo( ) method 329

lists (*continued*)
converting from stacks or queues 358
creating new 325
dynamically shrinking and growing 321
foreach loop 321
IComparable interface 329
IComparer interface 330
complex comparisons 332
creating instance 331
multiple classes 331
Sort( ) method 328
sorting 328–329
storing types 322
things you can do with 318
versus arrays 318–320, 325
versus enums 325
literals 125
Location property 582
logical operators 68
long type 124, 126
loops 65, 69
infinite 71
nested 77
lowercasing 201

M

Main( ) method 54, 55
masking fields 198
Math class 66
Maximize and Minimize buttons 35
MaximizeBox property 35
members (class) 273
memory 100
MemoryStreams 387
message about adding components to my class 579
MessageBox 56
methods 15, 52, 53
abstract 278, 281
adding for form 113
calling specific 221
curly braces 66
declaration 54
defining 66
entry point 54, 55
extension (see extension methods)
extracting 706
implementing interfaces 257–258
Main( ) 54, 55
naming 102–103
objects 92
overloaded (see overloaded methods)
overriding 218, 226
parameter has same name as a field 198
private 187–188
public 191
accessing private fields 188
capitalization 201
return values 88
set and get accessors 199
Show( ) 56
static (see static methods)
variables matching types of parameters 131
versus fields 98
with no return value 197
MinimizeBox property 35
multiple inheritance 286

N

Name property 51
namespaces 46, 52, 55
multiple classes in same 61
reserved 73
Navigation project 86–98
nested loops 77
.NET collections, LINQ 657
.NET Database Objects 6
.NET Framework 46
colors 76
Random class 150–151

tools 52
what you can do with 714
.NET Visual Objects 6, 17
NetworkStreams 387
new keyword 91
interfaces 260
No Dumb Questions
@ in front of filenames 401
\n 15, 401
\t 401
Beehive Simulator project
BeeControl 579
for loops 537
Go( ) methods 533
Hive class exceptions 539
NectarHarvested variable 523
number of bees 533
Point object 539
read-only automatic properties 523
removing dead flowers and retired bees 537
Show( ) method 587
BirthdayParty class 207
boxed objects and structs 644
byte order mark 434
capitalization 201
catch block 457
with no specified exceptions 462
chaining 491
changing names of files generated by IDE 11
changing types 154
class diagrams, moving up, not down 231
classes 73
versus structs 644
Close( ) method 434
closing streams 401
columns 20
constructors 198, 199
controls, altering reexisting 587
converting strings to byte array 401
creating new Lists 325
curly brackets 58
customized dialog boxes 401
data types 20
debugger 457
Watch window 457
easy way to implement interfaces 270
encapsulation 188, 276
entry point 58, 231
error handling 462
Error List 58
errors, You must rebuild your project for the changes to show up in any open designers. 579
EventHandler 491
event handlers 188
adding 491
returning something other than void 491
types of 491
events versus callbacks 510
Exception object 445
exceptions versus unhandled exceptions 462
extension methods 647
fields with no access 188
File class versus FileInfo class 434
FileStreams versus StreamReader and StreamWriter 434
finalizers
exceptions 635
using fields and methods 635
forms as objects 153
frames versus turns 549
from clause 664
FromImage( ) 606
garbage collection 153, 635
generic collections 325
generic data types 325
get accessor 199
graphics, using keyword 606
Graphics object 606
guys (Two Guys project) 114
IDE
auto-generated code 73
where data is stored 29
IDE toolbar
green arrow 16
Reset Window Layout command 11
Stop Debugging button 16

No Dumb Questions (*continued*)
IDisposable interface 473
instances, non-static methods 97
interface references, why use 276
interfaces
containing statements 270
like contracts 270
why use 276
join clause 677
knowing where to put breakpoints 452
line break 15
LINQ 663
LINQ querying SQL database 677
Lists versus arrays 325
Lists versus enums 325
message about adding components to my class 579
method 15
namespaces, reserved 73
new projects, Visual Studio 2008 11
null keyword 153
object references versus interface references 276
OpenFileDialog, changing properties 579
overloaded constructors 313
partial classes 73
patterns, callbacks 510
Point 644
private data 188
program stopping with exceptions 462
programs without Form1 form 231
properties
statements 199
versus fields 276
protected versus private or public 276
record data 29
reference variables, how they work 154
resizing images 606
select clause 664
select new clause 677
set accessor 199
setting structs equal to another 644
specifying particular kinds of exceptions 462
spotting exceptions 445
stack 644
static and non-static methods 97
static methods, when to use 97
StreamReader 401
StreamWriter 401
subclasses
and base classes 222
passing instance of 231
this variable 154
try/finally block 473
try block 457
unhandled exceptions 452
unhandled exception window 452
Unicode 434
unique IDs 20
unwanted code from IDE 11
upcasting, but not downcasting 270
UserControl 606
using methods that do match others defined by EventHandler 491
using statement 473
using this keyword to raise event 491
virtual methods 231
Visual Studio Express 11
Watch window, running methods in 452
why there are so many exceptions 445
why use interfaces 270
null keyword 153
NullReferenceException 443
NumericUpDown control 89

O

object initializers 115, 196, 197
object oriented programming (OOP) 288
object references, versus interface references 276
Object Relational Designer window 675
objects 91
accessing fields inside object 185
accidently misusing 184
assigning value 137
as variables 137

boxed 642
building from classes 92
declaring 137
encapsulation (see encapsulation)
event arguments 488
finalizers (see finalizers)
garbage collection 140
instances (see instances)
knowing when to respond 484
null keyword 153
reading entire with serialization 420
references 261
reference variables (see reference variables)
setting equal to value type 642
states 418
talking to other objects 152
versus structs 639
object type 125
Objectville Paper Co. logo 13
OOP (object oriented programming) 288
OpenFileDialog 579
changing properties 579
initialFolder property 403
OpenFileDialog control 398, 403
OpenRead( ) method 400
OpenWrite( ) method 400
operators 64
compound 136
orderby clause 664, 667
OR operator 68, 410
Oven class 266
OverFlowException 448
overloaded constructors 313
excuse management program 408
overloaded methods 343
override keyword 226
overriding methods 218

P

PaintEventArgs 605
Paint event handler 602
parameters 53, 88, 89
capitalization 201
same name as a field 198
partial classes 45, 50, 53, 59–61, 73, 78
PascalCase 201
patterns
callbacks 508, 510
Pen object 606
PerformanceCounter 709
performance issues
Beehive Simulator project 589–591
images 590
PictureBox control 12
adding to form 50
double-clicking 15
labs, #1 A Day at the Races 166, 168, 170
transparent background 589
Zoom mode 13
PictureBox control 217
Point 581, 644
polymorphism 289
as principle of OOP 288
popping up dialog boxes 397
primary key 20–24, 27
Primary Key button 20
PrintDocument object 614–615
printing
Beehive Simulator project 616–619
Graphics object and event handler 614
PrintPage event handler 615
private access modifier 258, 273

private fields 185–188
constructors 197
declaring 201
private methods 187–188
Problem Up Close, recalculating new individual costs 183
Program.cs 8, 54
programs (see applications)
properties 98
automatic 195
encapsulation 193
initializing public properties 196
interfaces 255
public, capitalization 201
read-only 195, 196
statements 199
versus fields 276
Properties window 35, 44
protected access modifier 273
protected keyword 275
public access modifier 273
public fields 191
initializing 196
public methods 191
accessing private fields 188
capitalization 201
public properties
capitalization 201
initializing 196
public void method 259
publish/ folder 38
Publish Contacts 37
Publish Wizard 37

Q

queries 19
anatomy of 660
editing with LINQPad 679
LINQ 658, 663
using join to combine two collections into one query 671
queues 355
converting to lists 358
enqueuing and dequeuing 356
foreach loop 358

R

Racetrack Simulator (see labs, #1 A Day at the Races)
Random class 150–151
randomizing results 150–151
read-only properties 195, 196
record data 29
refactoring 706–707
references
interfaces 260–261
objects 261
versus values 638
reference variables 138–140, 500
garbage collection 140
how they work 154
multiple 139
multiple references and their side effects 142
multiple references and unintentional changes 147
objects talking to other objects 152
Refresh( ) method 605
RemoveAllControls( ) method 583
render, defined 568
Renderer class 568–569, 583–585
animating bees on form 580–581
dictionaries 580
ResizeImage method 591
reserved words 154
Reset Window Layout command 11
Resource Designer 582
resource files 14
result += 64

result = 64
return statements 88
return type 88
return values 53, 88
risky code 456–473
RoboBee class 264
robust, defined 454
RSS feed, LINQ to XML 705

S

SaveFileDialog control 399, 403
  Title property 403
Save icon 15
sbyte type 124
scope 274
sealed access modifier 273
select new clause 671, 672, 675–679
select statement 664
semicolons 66
serialization 416–425
  making classes serializable 421
  making object serialize in Dispose( ) method 633
  object states 418
  reading and writing serialized files 429
  reading entire object 420
  serializing and deserializing deck of cards 422–423
  serializing objects out to file 424
  what happens to objects 417, 419
SerializationException 453, 460
  BinaryFormatter 454
Server Explorer 18
Service-Based Database 18
set accessor 193, 199
  interfaces with get accessor without set accssor 259
Setup executable 37
short type 124, 126, 127
Show( ) method 56
ShowDialog( ) method 397, 399
similar behaviors 214
similar code 214
Size property 51
slashes (//) 66
Sloppy Joe's Random Menu Item project 150–151
solution (.sln) file 46
Solution Explorer 18, 46, 48
Sort( ) method 328
SortBy field 332
source code files 46
Spy project 186–188
SQL (Structured Query Language) 19
  connecting LINQ to SQL database 674–675
  LINQ querying SQL database 677
  versus LINQ 661
SQL databases 18, 19
SQL Server Express
  database 7
  file 29
stack 355, 644
  converting to lists 358
  foreach loop 358
  popping items off 357
  versus heap 641
Starbuzz Coffee project 654–656
  join clause 678–679
Start Debugging 47
statements 19, 53, 73
static keyword 97
  creating instances 99
static methods 97
  when to use 97
static void Main( ) 55
Step out button 449

Step over button 449
Stop Debugging button 16
stored procedures 19
Stream object 386
StreamReader 393, 401
  hex dump 433
  versus FileStreams 434
streams 386
  chaining 394
  closing 401
  different types 387
  Dispose( ) method 406
  forgetting to close 388
  serializing objects 423
  things you can do with 387
  using file streams to build hex dumper 432
  using statements 406
  writing text to files 389
StreamWriter 389–393, 401
  [0] and [1] 401
  hex dump 433
  versus FileStreams 434
  Write( ) and WriteLine( ) methods 389
String.IsNullOrEmpty( ) 240
string literals 389, 401
strings 63
  converting to byte array 401
  extension methods 648
  formatting 179
  storing categories of data 310
string type 124, 126
  concatenation 130
  converting 130
structs 637
  boxed 642, 645
  Point 644
  setting one equal to another 640, 644
  versus classes 644
  versus objects 639
Structured Query Language (see SQL)
subclasses 214, 221
  accessing base class with base keyword 232
  constructors 233
  inheriting from base class 222
  modifying 225–226
  overriding methods 226
  passing instance of 231
  upcasting 267
  using instead of base classes 227
subscription
  how it works 486–487
  public events 505
  subscribing classes 489
switch statements 413–415
  building new constructors with 415
syntax 66
System.Drawing 594
System.Windows.Forms 46, 89, 111
System.Windows.Forms.Control 576
System namespace, Math class 66

T

TabControl 207
Table grid 28
tables
  adding columns 20
  adding to database 20
  finish building 25
  multiple 26
  saving 25
  versus collections 661
Take statement 664
testing installation 39
Textbox control 89
Text property 51
  changing 34
The Problem Up Close, recalculating new individual costs 183

this keyword 201, 274
labs, #1 A Day at the Races 167
using to raise event 491
this variable 154
Timer 579
timer 575, 583
disposing 577
timers 546, 548
events and delegates 547
Title property 399
Toolbox components 708–709
BackgroundWorker 708
FileSystemWatcher 709
PerformanceCounter 709
Toolbox controls, easier way to build 578
ToString( ) method 130, 179
TrackBars to zoom an image in and out 603–604
try/finally block 473
try blocks 455, 457
following in debugger 458–459
turns versus frames 549
types
object 125
(see also value types)

U

uint type 124
ulong type 124
unexpected input 456
unhandled exceptions 452
versus exceptions 462
Unicode 424, 434
converting text to 425
unique IDs 20
unwanted code from IDE 11
upcasting 267, 269
but not downcasting 270
Up Close, access modifiers 274–275
uppercasing 201
user's needs 5
UserControl 578, 606
user interface 3
developing 12
ushort type 124
using keyword, graphics 606
using lines 52
using statements 406, 473
Dispose( ) 630
exception handling 471

V

values versus references 638
value types 124, 154
bool (see bool type)
byte (see byte type)
casting 128–130
changing 154
char (see char type)
decimal (see decimal type)
double (see double type)
float (see float type)
int (see int type)
long (see long type)
sbyte 124
short (see short type)
string (see string type)
uint 124
ulong 124
ushort 124
variables matching types of parameters 131
variables 62, 73, 126
adding to form 112
assigning 128
assigning values 63
declaring 66
matching types of parameters 131

variables (*continued*)
naming 136
reference (see reference variables)
value types (see types)
var keyword 658
vertical bars 410
virtual keyword 226
virtual machines 47, 153
virtual methods 231
visual components 576
visual display elements 570
Visual Studio, what you get with C# and 2
Visual Studio 2008, new projects 11
Visual Studio 2008 Express 11
downloading xxxvi
setting up xxxvi
Visual Studio Integrated Development Environment (see IDE)
void method
interfaces 258
public 259
void return type 88, 103, 113

W

Watch it!
= operator versus == operator 67
automatic properties 195
destructors and finalizers 628
event handlers, hooking up 498
exceptions in constructors 459
LINQ 658
LINQ queries 663
method's parameter has same name as a field 198
object initializers 115
raising events with no handlers 490
SerializationException 454
Server Explorer versus Database Explorer 18
things looking different in your IDE 8
writing to files 429
where clause 664
while loops 65, 69, 73
white space 66
Windows calculator 125
Windows Forms Application project 44, 66
Windows installer 7
Windows Presentation Framework (WPF) 712–713
Write( ) method 401
WriteLine( ) method 401

X

XAML (Extensible Application Markup Language) 713
XML, LINQ to XML 704–705
reading RSS feed 705

Y

yesNo = 64

Z

zooming, TrackBars to zoom an image in and out 603–604
Zoo Simulator project 216–222
class hierarchy 220
extending base class 221
inheriting from base class 221
overriding methods 218