

Head First C#

Fourth Edition

WOULDN'T IT BE DREAMY IF
THERE WAS A C# BOOK THAT WAS
MORE FUN THAN MEMORIZING
A DICTIONARY? IT'S PROBABLY
NOTHING BUT A FANTASY...



Andrew Stellman
Jennifer Greene

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Head First C#

Fourth Edition

by Andrew Stellman and Jennifer Greene

Copyright © 2021 Jennifer Greene, Andrew Stellman. All rights reserved.

Printed in the United States of America.

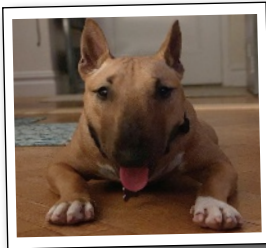
Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

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

Series Creators:	Kathy Sierra, Bert Bates
Cover Designer:	Ellie Volckhausen
Brain Image on Spine:	Eric Freeman
Editors:	Nicole Taché, Amanda Quinn
Proofreader:	Rachel Head
Indexer:	Potomac Indexing, LLC
Illustrator:	Jose Marzan
Page Viewers:	Greta the miniature bull terrier and Samosa the Pomeranian

Printing History:

November 2007: First Edition
May 2010: Second Edition
August 2013: Third Edition
December 2020: Fourth 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.

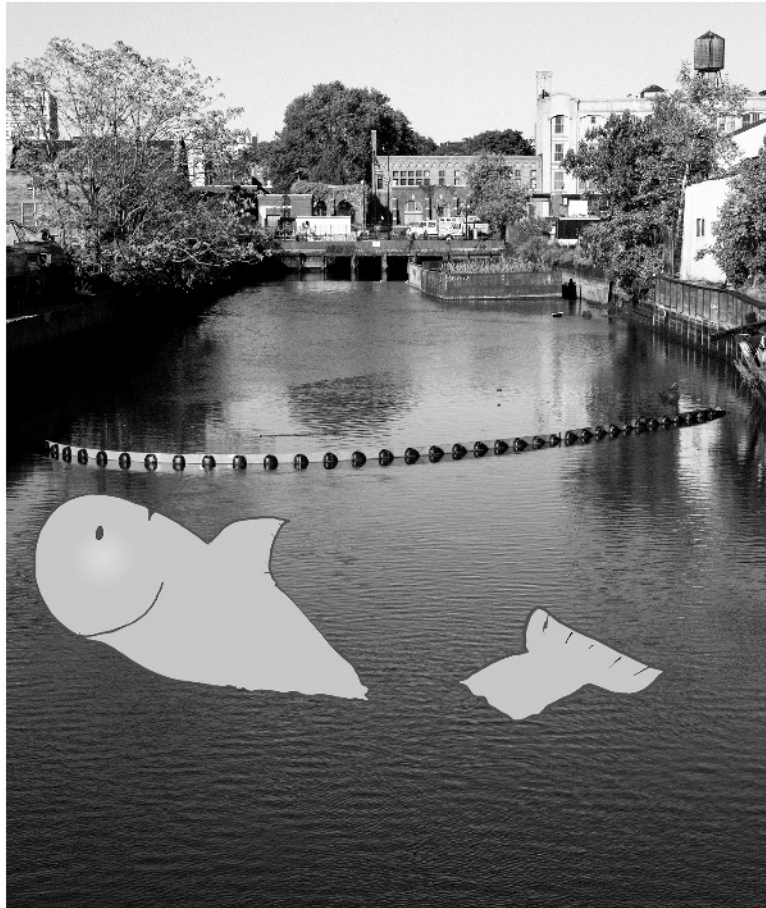
No bees, space aliens, or comic book heroes were harmed in the making of this book.

ISBN: 978-1-491-97670-8

[LSI]

[2020-12-18]

*This book is dedicated to the loving memory of Sludgie the Whale,
who swam to Brooklyn on April 17, 2007.*



*You were only in our canal for a day,
but you'll be in our hearts forever.*

THANKS FOR READING OUR BOOK!
WE REALLY LOVE WRITING ABOUT
THIS STUFF, AND WE HOPE YOU GET
A LOT OUT OF IT...

Andrew

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



...BECAUSE
WE KNOW YOU'RE
GOING TO HAVE
A GREAT TIME
LEARNING C#.

Jenny

Andrew Stellman, despite being raised a New Yorker, has lived in Minneapolis, Geneva, and Pittsburgh... *twice*, first when he graduated from Carnegie Mellon's School of Computer Science, and then again when he and Jenny were starting their consulting business and writing their first book for O'Reilly.

Andrew's first job after college was building software at a record company, EMI-Capitol Records—which actually made sense, as he went to LaGuardia High School of Music & Art and the Performing Arts to study cello and jazz bass guitar. He and Jenny first worked together at a company on Wall Street that built financial software, where he was managing a team of programmers. Over the years he's been a vice president at a major investment bank, architected large-scale real-time backend systems, managed large international software teams, and consulted for companies, schools, and organizations, including Microsoft, the National Bureau of Economic Research, and MIT. He's had the privilege of working with some pretty amazing programmers during that time, and likes to think that he's learned a few things from them.

When he's not writing books, Andrew keeps himself busy writing useless (but fun) software, playing (and making) both music and video games, practicing krav maga, tai chi, and aikido, and owning a crazy Pomeranian.

Jennifer Greene studied philosophy in college but, like everyone else in the field, couldn't find a job doing it. Luckily, she's a great software engineer, so she started out working at an online service, and that's the first time she really got a good sense of what good software development looked like.

She moved to New York in 1998 to work on software quality at a financial software company. She's managed teams of developers, testers, and PMs on software projects in media and finance since then.

Jenny has traveled all over the world to work with different software teams and build all kinds of cool projects.

She loves traveling, watching Bollywood movies, reading the occasional comic book, playing video games, and hanging out with her huge Siberian cat, Sascha, and her miniature bull terrier, Greta.

Jenny and Andrew have been building software and writing about software engineering together since they first met in 1998. Their first book, *Applied Software Project Management*, was published by O'Reilly in 2005. Other Stellman and Greene books for O'Reilly include *Beautiful Teams* (2009), *Learning Agile* (2014), and their first book in the Head First series, *Head First PMP* (2007), now in its fourth edition.

They founded Stellman & Greene Consulting in 2003 to build a really neat software project for scientists studying herbicide exposure in Vietnam vets. In addition to building software and writing books, they've consulted for companies and spoken at conferences and meetings of software engineers, architects, and project managers.

Learn more about them on their website, *Building Better Software*: <https://www.stellman-greene.com>.

Follow @AndrewStellman and @JennyGreene on Twitter



Jenny and Andrew

Table of Contents (the summary)



Let's add some excitement to the game! The time elapsed since the game started will appear at the bottom of the window, constantly going up, and only stopping after the last animal is matched.

	Intro	xxxi
1	Start building with C#: <i>Building something great...fast!</i>	1
2	Dive into C#: <i>Statements, classes, and code</i>	49
	<i>Unity Lab 1: Explore C# with Unity</i>	87
3	Objects...get oriented: <i>Making code make sense</i>	117
4	Types and references: <i>Getting the reference</i>	155
	<i>Unity Lab 2: Write C# Code for Unity</i>	213
5	Encapsulation: <i>Keep your privates...private</i>	227
6	Inheritance: <i>Your object's family tree</i>	273
	<i>Unity Lab 3: GameObject Instances</i>	343
7	Interfaces, casting, and "is": <i>Making classes keep their promises</i>	355
8	Enums and collections: <i>Organizing your data</i>	405
	<i>Unity Lab 4: User Interfaces</i>	453
9	LINQ and lambdas: <i>Get control of your data</i>	467
10	Reading and writing files: <i>Save the last byte for me!</i>	529
	<i>Unity Lab 5: Raycasting</i>	577
11	Captain Amazing: <i>The Death of the Object</i>	587
12	Exception handling: <i>Putting out fires gets old</i>	623
	<i>Unity Lab 6: Scene Navigation</i>	651
	Downloadable exercise: Animal match boss battle	661
i	Visual Studio for Mac Learner's Guide	663
ii	Code Kata: <i>A learning guide for advanced and impatient readers</i>	725



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
And we know what your brain is thinking.	xxxii
Metacognition: thinking about thinking	xxxiii
Here's what WE did	xxxiv
Here's what YOU can do to bend your brain into submission	xxxv
README	xxxvi
The technical review team	xli
Acknowledgments	xli
O'Reilly online learning	xliv

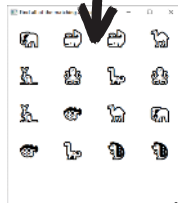




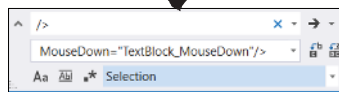
CREATE THE
PROJECT



DESIGN THE
WINDOW



WRITE C#
CODE



HANDLE MOUSE
CLICKS



ADD A GAME
TIMER

1

start building with C#

Build something great...fast!

Want to build great apps...right now?

With C#, you've got a **modern programming language** and a **valuable tool** at your fingertips. And with **Visual Studio**, you've got an **amazing development environment** with highly intuitive features that make coding as easy as possible. Not only is Visual Studio a great tool for writing code, it's also a **really valuable learning tool** for exploring C#. Sound appealing? Turn the page, and let's get coding.

Why you should learn C#	2
Visual Studio is a tool for writing code and exploring C#	3
Create your first project in Visual Studio	4
Let's build a game!	6
Here's how you'll build your game	7
Create a WPF project in Visual Studio	8
Use XAML to design your window	12
Design the window for your game	13
Set the window size and title with XAML properties	14
Add rows and columns to the XAML grid	16
Make the rows and columns equal size	17
Add a TextBlock control to your grid	18
Now you're ready to start writing code for your game	21
Generate a method to set up the game	22
Finish your SetUpGame method	24
Run your program	26
Add your new project to source control	30
The next step to build the game is handling mouse clicks	33
Make your TextBlocks respond to mouse clicks	34
Add the TextBlock_MouseDown code	37
Make the rest of the TextBlocks call the same MouseDown event handler	38
Finish the game by adding a timer	39
Add a timer to your game's code	40
Use the debugger to troubleshoot the exception	42
Add the rest of the code and finish the game	46
Update your code in source control	47

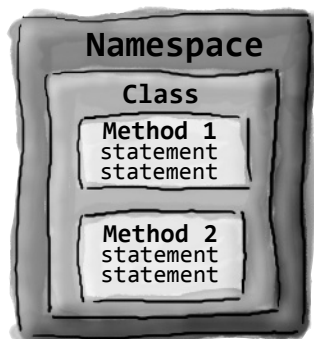
dive into C#

2

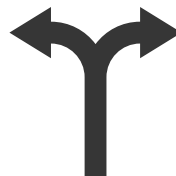
Statements, classes, and code**You're not just an IDE user. You're a developer.**

You can get a lot of work done using the IDE, but there's only so far it can take you.

Visual Studio is one of the most advanced software development tools ever made, but a **powerful IDE** is only the beginning. It's time to **dig in to C# code**: how it's structured, how it works, and how you can take control of it...because there's no limit to what you can get your apps to do.



Let's take a closer look at the files for a console app	50
Two classes can be in the same namespace (and file!)	52
Statements are the building blocks for your apps	55
Your programs use variables to work with data	56
Generate a new method to work with variables	58
Add code that uses operators to your method	59
Use the debugger to watch your variables change	60
Use operators to work with variables	62
"if" statements make decisions	63
Loops perform an action over and over	64
Use code snippets to help write loops	67
Controls drive the mechanics of your user interfaces	71
Create a WPF app to experiment with controls	72
Add a TextBox control to your app	75
Add C# code to update the TextBlock	78
Add an event handler that only allows number input	79
Add sliders to the bottom row of the grid	83
Add C# code to make the rest of the controls work	84

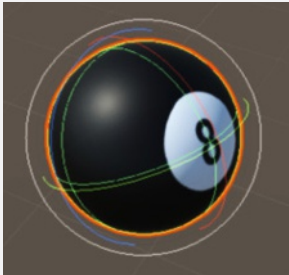
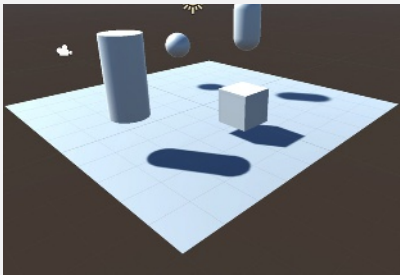


Unity Lab 1

Explore C# with Unity

Welcome to your first **Head First C# Unity Lab**. Writing code is a skill, and like any other skill, getting better at it takes **practice and experimentation**. Unity will be a really valuable tool for that. In this lab, you can begin practicing what you’ve learned about C# in Chapters 1 and 2.

Unity is a powerful tool for game design	88
Download Unity Hub	89
Use Unity Hub to create a new project	90
Take control of the Unity layout	91
Your scene is a 3D environment	92
Unity games are made with GameObjects	93
Use the Move Gizmo to move your GameObjects	94
The Inspector shows your GameObject’s components	95
Add a material to your Sphere GameObject	96
Rotate your sphere	99
Get creative!	102



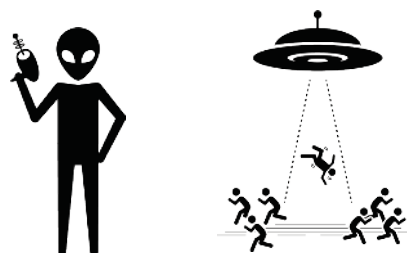
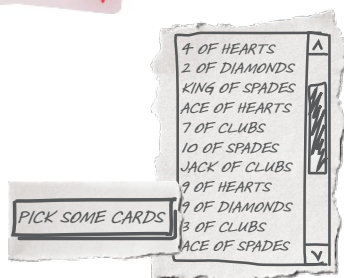
objects...get oriented!

Making code make sense

3

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—and really put some thought into how you design them—you end up with code that's *intuitive* to write, and easy to read and change.



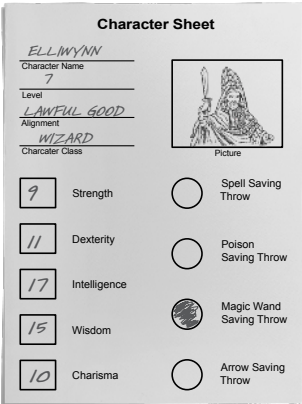
If code is useful, it gets reused	104
Some methods take parameters and return a value	105
Let's build a program that picks some cards	106
Create your PickRandomCards console app	107
Finish your PickSomeCards method	108
Your finished CardPicker class	110
Ana's working on her next game	113
Build a paper prototype for a classic game	116
Up next: build a WPF version of your card picking app	118
A StackPanel is a container that stacks other controls	119
Reuse your CardPicker class in a new WPF app	120
Use a Grid and StackPanel to lay out the main window	121
Lay out your Card Picker desktop app's window	122
Ana can use objects to solve her problem	126
You use a class to build an object	127
When you create a new object from a class, it's called an instance of that class	128
A better solution for Ana... brought to you by objects	129
An instance uses fields to keep track of things	133
Thanks for the memory	136
What's on your program's mind	137
Sometimes code can be difficult to read	138
Use intuitive class and method names	140
Build a class to work with some guys	146
There's an easier way to initialize objects with C#	148
Use the C# Interactive window to run C# code	154

types and references

Getting the reference

4

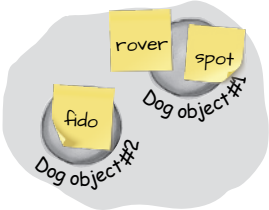
What would your apps be without data? Think about it for a minute. Without data, your programs are...well, it's actually hard to imagine writing code without data. You need **information** from your users, and you use that to look up or produce new information to give back to them. In fact, almost everything 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** and **references**, see how to work with data in your program, and even learn a few more things about **objects** (*guess what...objects are data, too!*).



Creating a reference is like writing a name on a sticky note and sticking it to the object. You're using it to label an object so you can refer to it later.



Owen could use our help!	156
Character sheets store different types of data on paper	157
A variable's type determines what kind of data it can store	158
C# has several types for storing integers	159
Let's talk about strings	161
A literal is a value written directly into your code	162
A variable is like a data to-go cup	165
Other types come in different sizes, too	166
10 pounds of data in a 5-pound bag	167
Casting lets you copy values that C# can't automatically convert to another type	168
C# does some conversion automatically	171
When you call a method, the arguments need to be compatible with the types of the parameters	172
Let's help Owen experiment with ability scores	176
Use the C# compiler to find the problematic line of code	178
Use reference variables to access your objects	186
Multiple references and their side effects	190
Objects use references to talk to each other	198
Arrays hold multiple values	200
Arrays can contain reference variables	201
null means a reference points to nothing	203
Welcome to Sloppy Joe's Budget House o' Discount Sandwiches!	208

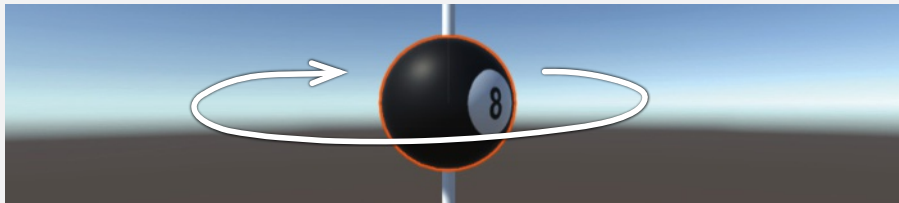


Unity Lab 2

Write C# Code for Unity

Unity isn't *just* a powerful, cross-platform engine and editor for building 2D and 3D games and simulations. It's also a **great way to get practice writing C# code**. In this lab, you'll get more practice writing C# code for a project in Unity.

C# scripts add behavior to your GameObjects	214
Add a C# script to your GameObject	215
Write C# code to rotate your sphere	216
Add a breakpoint and debug your game	218
Use the debugger to understand Time.deltaTime	219
Add a cylinder to show where the Y axis is	220
Add fields to your class for the rotation angle and speed	221
Use Debug.DrawRay to explore how 3D vectors work	222
Run the game to see the ray in the Scene view	223
Rotate your ball around a point in the scene	224
Use Unity to take a closer look at rotation and vectors	225
Get creative!	226



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 fields. In this chapter, you're going to learn about the power of **encapsulation**, a way of programming that helps you make code that's flexible, easy to use, and difficult to misuse. You'll **make your objects' data private**, and add **properties** to protect how that data is accessed.



SwordDamage
Roll
MagicMultiplier
FlamingDamage
Damage
CalculateDamage
SetMagic
SetFlaming



Let's help Owen roll for damage	228
Create a console app to calculate damage	229
Design the XAML for a WPF version of the damage calculator	231
The code-behind for the WPF damage calculator	232
Tabletop talk (or maybe...dice discussion?)	233
Let's try to fix that bug	234
Use Debug.WriteLine to print diagnostic information	235
It's easy to accidentally misuse your objects	238
Encapsulation means keeping some of the data in a class private	239
Use encapsulation to control access to your class's methods and fields	240
But is the RealName field REALLY protected?	241
Private fields and methods can only be accessed from instances of the same class	242
Why encapsulation? Think of an object as a black box...	247
Let's use encapsulation to improve the SwordDamage class	251
Encapsulation keeps your data safe	252
Write a console app to test the PaintballGun class	253
Properties make encapsulation easier	254
Modify your Main method to use the Bullets property	255
Auto-implemented properties simplify your code	256
Use a private setter to create a read-only property	257
What if we want to change the magazine size?	258
Use a constructor with parameters to initialize properties	259
Specify arguments when you use the new keyword	260



RealName: "Herb Jones"
Alias: "Dash Martin"
Password: "the crow flies at midnight"

inheritance

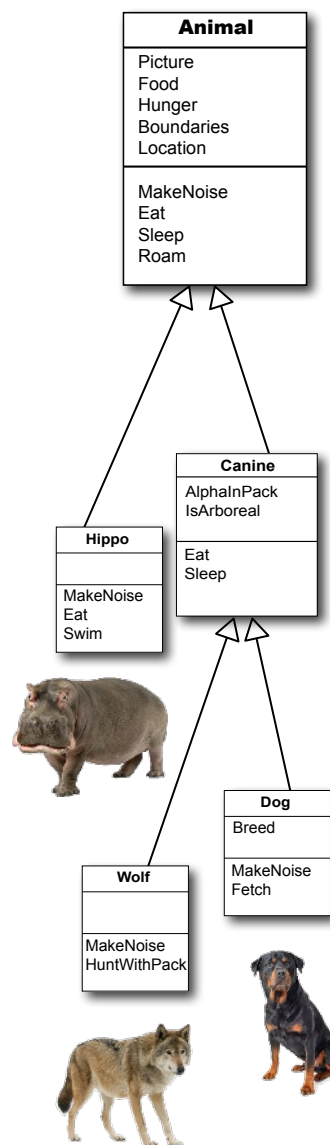
Your object's family tree

6

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

Ever run across a class that **almost** does exactly what you want **your** class to do?

Found yourself thinking that if you could just **change a few things**, that class would be perfect? With **inheritance**, you can **extend** an existing class so your new class gets all of its behavior—with the **flexibility** to make changes to that behavior so you can tailor it however you want. Inheritance is one of the most powerful concepts and techniques in the C# language: with it you'll **avoid duplicate code**, **model the real world** more closely, and end up with apps that are **easier to maintain** and **less prone to bugs**.



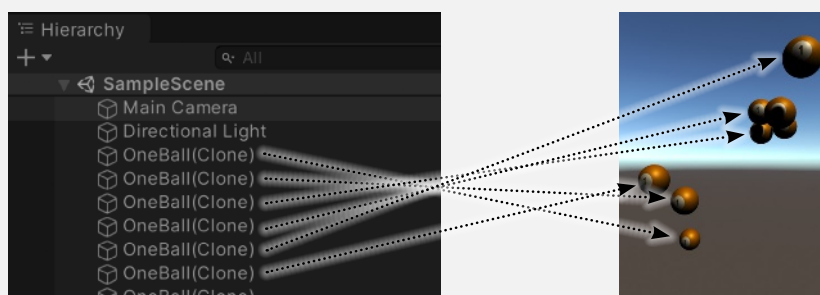
Calculate damage for MORE weapons	274
Use a switch statement to match several candidates	275
One more thing...can we calculate damage for a dagger? and a mace?	277
When your classes use inheritance, you only need to write your code once	278
Build up your class model by starting general and getting more specific	279
How would you design a zoo simulator?	280
Every subclass extends its base class	285
Use a colon to extend a base class	290
A subclass can override methods to change or replace members it inherited	292
Some members are only implemented in a subclass	297
Use the debugger to understand how overriding works	298
Build an app to explore virtual and override	300
A subclass can hide methods in the base class	302
Use the override and virtual keywords to inherit behavior	304
When a base class has a constructor, your subclass needs to call it	307
It's time to finish the job for Owen	309
Build a beehive management system	316
The Queen class: how she manages the worker bees	318
The UI: add the XAML for the main window	319
Feedback drives your Beehive Management game	328
Some classes should never be instantiated	332
An abstract class is an intentionally incomplete class	334
Like we said, some classes should never be instantiated	336
An abstract method doesn't have a body	337
Abstract properties work just like abstract methods	338

Unity Lab 3

GameObject Instances

C# is an object-oriented language, and since these Head First C# Unity Labs are all **about getting practice writing C# code**, it makes sense that these labs will focus on creating objects.

Let's build a game in Unity!	344
Create a new material inside the Materials folder	345
Spawn a billiard ball at a random point in the scene	346
Use the debugger to understand Random.value	347
Turn your GameObject into a prefab	348
Create a script to control the game	349
Attach the script to the Main Camera	350
Press Play to run your code	351
Use the Inspector to work with GameObject instances	352
Use physics to keep balls from overlapping	353
Get creative!	354



interfaces, casting, and “is”

Making classes keep their promises

7

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 its kneecaps, see?

DEFEND
THE HIVE AT ALL
COSTS.

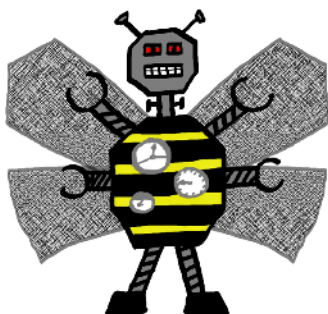


Queen object

YES,
MA'AM!



HiveDefender object



The beehive is under attack!	356
We can use casting to call the DefendHive method...	357
An interface defines methods and properties that a class must implement...	358
Get a little practice using interfaces	360
You can't instantiate an interface, but you can reference an interface	366
Interface references are ordinary object references	369
The RoboBee 4000 can do a worker bee's job without using valuable honey	370
The IWorker's Job property is a hack	374
Use “is” to check the type of an object	375
Use “is” to access methods in a subclass	376
What if we want different animals to swim or hunt in packs?	378
Use interfaces to work with classes that do the same job	379
Safely navigate your class hierarchy with “is”	380
C# has another tool for safe type conversion: the “as” keyword	381
Use upcasting and downcasting to move up and down a class hierarchy	382
Upcasting turns your CoffeeMaker into an Appliance	384
Upcasting and downcasting work with interfaces, too	386
Downcasting turns your Appliance back into a CoffeeMaker	385
Interfaces can inherit from other interfaces	388
Interfaces can have static members	395
Default implementations give bodies to interface methods	396
Add a ScareAdults method with a default implementation	397
Data binding updates WPF controls automatically	399
Modify the Beehive Management System to use data binding	400
Polymorphism means that one object can take many different forms	403



enums and collections

8 Organizing your data

When it rains, it pours.

In the real world, you don't receive 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 **enums** and collections come in. Enums are types that let you define valid values to categorize your data. Collections are special objects that store many values, letting you **store, sort, and manage** all the data that your programs need to pore through. That way, you can spend your time thinking 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	406
Enums let you work with a set of valid values	407
Enums let you represent numbers with names	408
We could use an array to create a deck of cards...	411
Lists make it easy to store collections of...anything	413
Lists are more flexible than arrays	414
Let's build an app to store shoes	417
Generic collections can store any type	420
Collection initializers are similar to object initializers	426
Let's create a List of Ducks	427
Lists are easy, but SORTING can be tricky	428
Comparable<Duck> helps your list sort its ducks	429
Use IComparer to tell your List how to sort	430
Create an instance of your comparer object	431
Overriding a ToString method lets an object describe itself	435
Update your foreach loops to let your Ducks and Cards write themselves to the console	436
You can upcast an entire list using IEnumerable<T>	440
Use a Dictionary to store keys and values	442
The Dictionary functionality rundown	443
Build a program that uses a dictionary	444
And yet MORE collection types...	445
A queue is FIFO—first in, first out	446
A stack is LIFO—last in, first out	447
Downloadable exercise: Two Decks	452



The rarely-played duke of oxen card

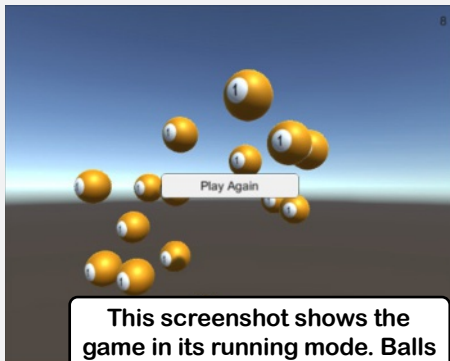


Unity Lab 4

User Interfaces

In the last Unity Lab you started to build a game, using a prefab to create GameObject instances that appear at random points in 3D space and fly in circles. This Unity Lab picks up where the last one left off, allowing you to apply what you've learned about interfaces in C# and more.

Add a score that goes up when the player clicks a ball	454
Add two different modes to your game	455
Add game mode to your game	456
Add a UI to your game	458
Set up the Text that will display the score in the UI	459
Add a button that calls a method to start the game	460
Make the Play Again button and Score Text work	461
Finish the code for the game	462
Get creative!	466



This screenshot shows the game in its running mode. Balls are added and the player can click on them to score.



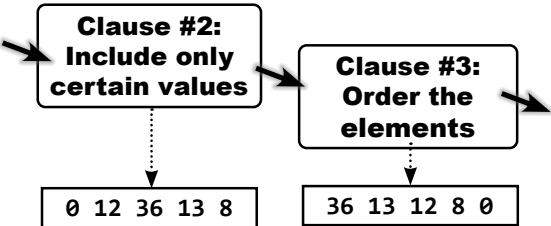
When the last ball is added, the game switches to its Game Over mode. The Play Again button pops up and no more balls get added.

LINQ and lambdas

Get control of your data

9

You're ready for a whole new world of app development. Using WinForms to build Windows Desktop apps is a great way to learn important C# concepts, but there's *so much more* you can do with your programs. In this chapter, you'll use **XAML** to design your Windows Store apps, you'll learn how to **build pages to fit any device**, **integrate** your data into your pages with **data binding**, and use Visual Studio to cut through the mystery of XAML pages by exploring the objects created by your XAML code.



Jimmy's a Captain Amazing super-fan...	468
Use LINQ to query your collections	470
LINQ works with any IEnumerable<T>	472
LINQ's query syntax	475
LINQ works with objects	477
Use a LINQ query to finish the app for Jimmy	478
The var keyword lets C# figure out variable types for you	480
LINQ queries aren't run until you access their results	487
Use a group query to separate your sequence into groups	488
Use join queries to merge data from two sequences	491
Use the new keyword to create anonymous types	492
Add a unit test project to Jimmy's comic collection app	502
Write your first unit test	503
Write a unit test for the GetReviews method	505
Write unit tests to handle edge cases and weird data	506
Use the => operator to create lambda expressions	508
A lambda test drive	509
Refactor a clown with lambdas	510
Use the ?: operator to make your lambdas make choices	513
Lambda expressions and LINQ	514
LINQ queries can be written as chained LINQ methods	515
Use the => operator to create switch expressions	517
Explore the Enumerable class	521
Create an enumerable sequence by hand	522
Use yield return to create your own sequences	523
Use yield return to refactor ManualSportSequence	524
Downloadable exercise: Go Fish!	528

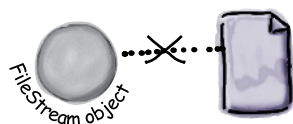
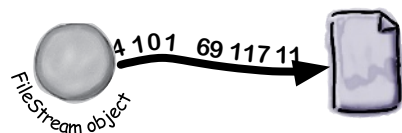
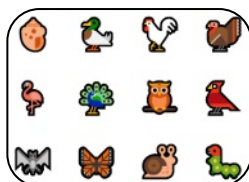
reading and writing files

Save the last byte for me!

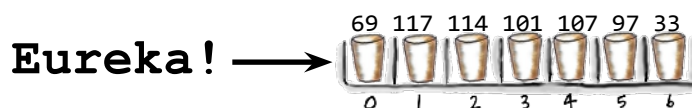
10

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**.



.NET uses streams to read and write data	530
Different streams read and write different things	531
A FileStream reads and writes bytes in a file	532
Write text to a file in three simple steps	533
The Swindler launches another diabolical plan	534
Use a StreamReader to read a file	537
Data can go through more than one stream	538
Use the static File and Directory classes to work with files and directories	542
IDisposable makes sure objects are closed properly	545
Use a MemoryStream to stream data to memory	547
What happens to an object when it's serialized?	553
But what exactly IS an object's state? What needs to be saved?	554
Use JsonSerializer to serialize your objects	556
JSON only includes data, not specific C# types	559
Next up: we'll take a deep dive into our data	561
C# strings are encoded with Unicode	563
Visual Studio works really well with Unicode	565
.NET uses Unicode to store characters and text	566
C# can use byte arrays to move data around	568
Use a BinaryWriter to write binary data	569
Use BinaryReader to read the data back in	570
A hex dump lets you see the bytes in your files	572
Use Stream.Read to read bytes from a stream	574
Modify your hex dumper to use command-line arguments	575
Downloadable exercise: Hide and Seek	576

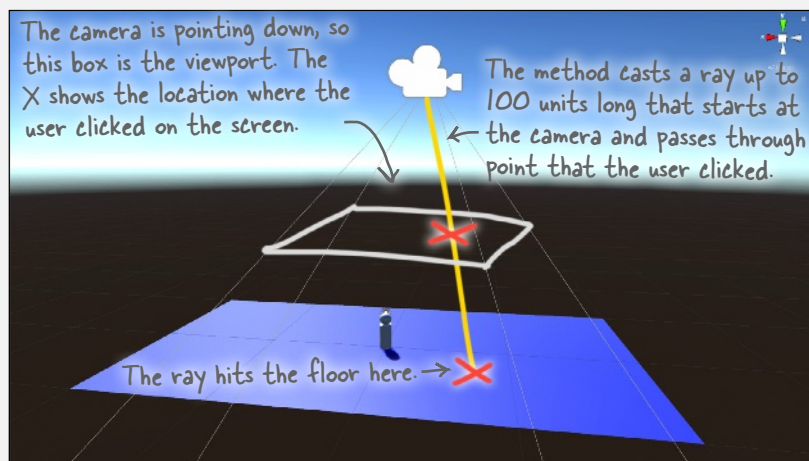


Unity Lab 5

Raycasting

When you set up a scene in Unity, you're creating a virtual 3D world for the characters in your game to move around in. But in most games, things aren't directly controlled by the player. So how do these objects find their way around a scene? In this lab, we'll look at how C# can help.

Create a new Unity project and start to set up the scene	578
Set up the camera	579
Create a GameObject for the player	580
Introducing Unity's navigation system	581
Set up the NavMesh	582
Make your player automatically navigate the play area	583



CAPTAIN AMAZING

THE DEATH OF THE OBJECT

Head First C#	
Four bucks	Chapter 11



The life and death of an object	590
Use the GC class (with caution) to force garbage collection	591
Your last chance to DO something...your object's finalizer	592
When EXACTLY does a finalizer run?	593
Finalizers can't depend on other objects	595
A struct looks like an object...	599
Values get copied; references get assigned	600
Structs are value types; objects are reference types	601
The stack vs. the heap: more on memory	603
Use out parameters to make a method return more than one value	606
Pass by reference using the ref modifier	607
Use optional parameters to set default values	608
A null reference doesn't refer to any object	609
Non-nullable reference types help you avoid NREs	610
The null-coalescing operator ?? helps with nulls	611
Nullable value types can be null...and handled safely	612
"Captain" Amazing...not so much	613
Extension methods add new behavior to EXISTING classes	617
Extending a fundamental type: string	619



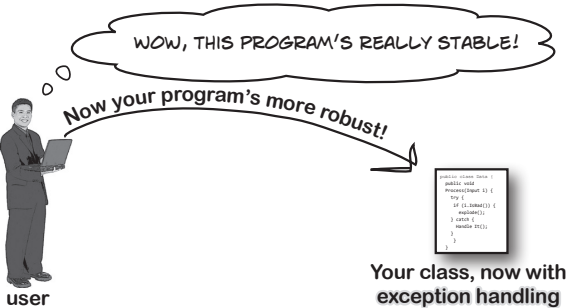
12

exception handling

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. But you're still getting panicked phone calls in the middle of the night 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 plan for those problems, and **keep things running** when they happen.



Your hex dumper reads a filename from the command line 624

When your program throws an exception, the CLR generates an Exception object 628

All Exception objects inherit from System.Exception 629

There are some files you just can't dump 632

What happens when a method you want to call is risky? 633

Handle exceptions with try and catch 634

Use the debugger to follow the try/catch flow 635

If you have code that ALWAYS needs to run, use a finally block 636

Catch-all exceptions handle System.Exception 637

Use the right exception for the situation 642

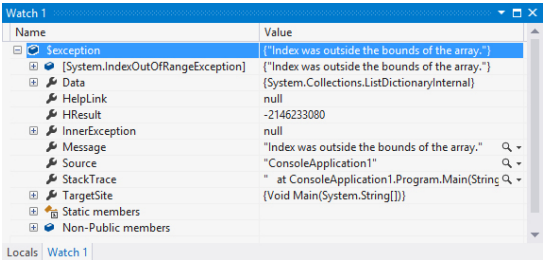
Exception filters help you create precise handlers 646

The worst catch block EVER: catch-all plus comments 648

Temporary solutions are OK (temporarily) 649



```
int[] anArray = {3, 4, 1, 11};
int aValue = anArray[15];
```

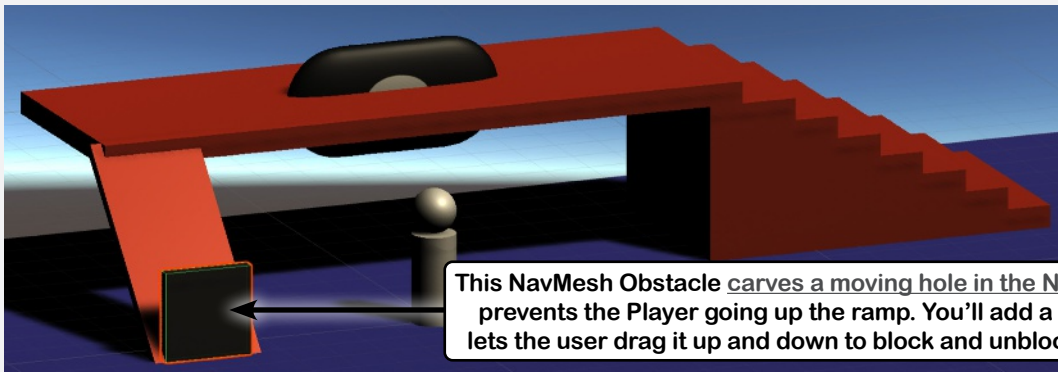


Unity Lab 6

Scene Navigation

In the last Unity Lab, you created a scene with a floor (a plane) and a player (a sphere nested under a cylinder), and you used a NavMesh, a NavMesh Agent, and raycasting to get your player to follow your mouse clicks around the scene. In this lab, you'll add to the scene with the help of C#.

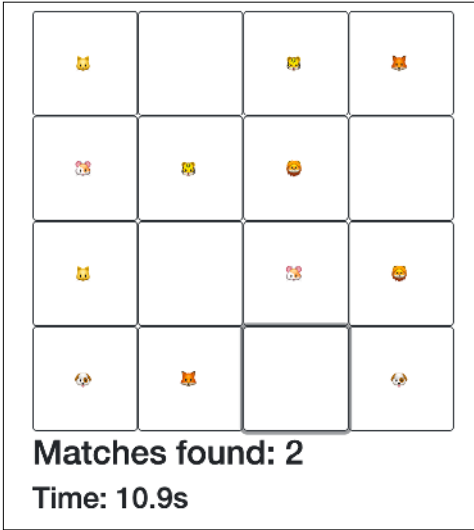
Let's pick up where the last Unity Lab left off	652
Add a platform to your scene	653
Use bake options to make the platform walkable	654
Include the stairs and ramp in your NavMesh	655
Fix height problems in the NavMesh	657
Add a NavMesh Obstacle	658
Add a script to move the obstacle up and down	659
Get creative!	660



This NavMesh Obstacle carves a moving hole in the NavMesh that prevents the Player going up the ramp. You'll add a script that lets the user drag it up and down to block and unblock the ramp.

appendix i: ASP.NET Core Blazor projects

Visual Studio for Mac Learner's Guide



Why you should learn C#	664
Create your first project in Visual Studio for Mac	666
Let's build a game!	670
Create a Blazor WebAssembly App in Visual Studio	672
Run your Blazor web app in a browser	674
Start writing code for your game	676
Finish creating your emoji list and display it in the app	680
Shuffle the animals so they're in a random order	682
You're running your game in the debugger	684
Add your new project to source control	688
Add C# code to handle mouse clicks	689
Add click event handlers to your buttons	690
Test your event handler	692
Use the debugger to troubleshoot the problem	693
Track down the bug that's causing the problem	696
Add code to reset the game when the player wins	698
Add a timer to your game's code	702
Clean up the navigation menu	704
Controls drive the mechanics of your user interfaces	706
Create a new Blazor WebAssembly App project	707
Add a page with a slider control	708
Add text input to your app	710
Add color and date pickers to your app	713
Build a Blazor version of your card picking game	714
The page is laid out with rows and columns	716
The slider uses data binding to update a variable	717
Welcome to Sloppy Joe's Budget House o'Discount Sandwiches!	720



appendix ii: Code Kata

A learning guide for advanced and impatient readers

