

# Head First Java™

Third Edition



Wouldn't it be dreamy  
if there was a Java book  
that was more stimulating  
than waiting in line at the  
DMV to renew your driver's  
license? It's probably just a  
fantasy...

Kathy Sierra  
Bert Bates  
Trisha Gee

Beijing • Boston • Farnham • Sebastopol • Tokyo

**O'REILLY®**

# Head First Java™

## Third Edition

by Kathy Sierra, Bert Bates, and Trisha Gee

Copyright © 2022 by Kathy Sierra and Bert Bates. 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 (*oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

**Editor for 1st and 2nd Editions:** Mike Loukides

**Editors for 3rd Edition:** Suzanne McQuade, Nicole Taché

**Cover Design:** Susan Thompson, based on a series design by Ellie Volckhausen

**Cover Illustration:** José Marzan, Jr.

**Production Editor:** Kristen Brown

**Original Interior Designers:** Kathy Sierra and Bert Bates

**3rd Edition Design Support:** Ron Bilodeau

**Java Whisperer:** Trisha Gee

**Series Advisors:** Eric Freeman, Elizabeth Robson

## Printing History:

May 2003: First Edition.

February 2005: Second Edition.

May 2022: Third Edition

(You might want to pick up a copy of *all* the editions...for your kids. Think eBay™)

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries. O'Reilly Media, Inc. is independent of Sun Microsystems.

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.

In other words, if you use anything in *Head First Java*™ to, say, run a nuclear power plant or air traffic control system, you're on your own.

978-149-191077-1

[LSI]

[2022-05-11]

# Table of Contents (summary)

Intro	xxi	
1	Breaking the Surface: <i>dive in: a quick dip</i>	1
2	A Trip to Objectville: <i>classes and objects</i>	27
3	Know Your Variables: <i>primitives and references</i>	49
4	How Objects Behave: <i>methods use instance variables</i>	71
5	Extra-Strength Methods: <i>writing a program</i>	95
6	Using the Java Library: <i>get to know the Java API</i>	125
7	Better Living in Objectville: <i>inheritance and polymorphism</i>	167
8	Serious Polymorphism: <i>interfaces and abstract classes</i>	199
9	Life and Death of an Object: <i>constructors and garbage collection</i>	237
10	Numbers Matter: <i>numbers and statics</i>	275
11	Data Structures: <i>collections and generics</i>	309
12	What, Not How: <i>lambdas and streams</i>	369
13	Risky Behavior: <i>exception handling</i>	421
14	A Very Graphic Story: <i>intro to GUI, event handling, and inner classes</i>	461
15	Work on Your Swing: <i>using swing</i>	509
16	Saving Objects (and Text): <i>serialization and file I/O</i>	539
17	Make a Connection: <i>networking and threads</i>	587
18	Dealing with Concurrency Issues: <i>race conditions and immutable data</i>	639
A	Appendix A: <i>final code kitchen</i>	673
B	Appendix B: <i>the top ten-ish topics that didn't make it into the rest of the book</i>	683
	Index	701

# Table of Contents (the real thing)



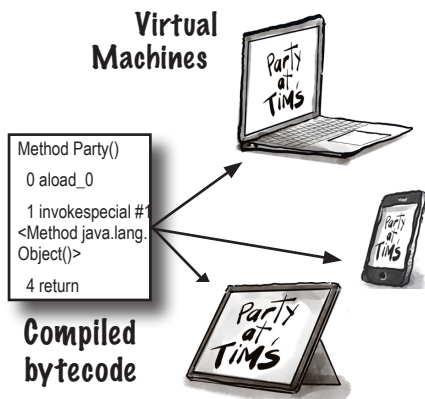
## Intro

**Your brain on Java.** Here *you* are trying to *learn* something, while here your *brain* is doing you a favor by making sure the learning doesn't *stick*. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how *do* you trick your brain into thinking that your life depends on knowing Java?

Who is this book for?	xxvi
We know what you're thinking.	xxvii
Metacognition: thinking about thinking.	xxix
Here's what WE did	xxx
Here's what YOU can do to bend your brain into submission	xxxi
What you need for this book	xxxii
Last-minute things you need to know	xxxiii

# 1 Breaking the Surface

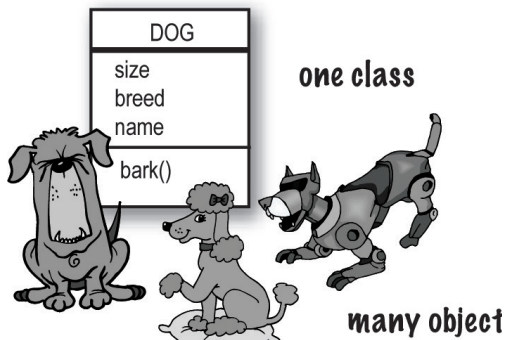
**Java takes you to new places.** From its humble release to the public as the (wimpy) version 1.02, Java seduced programmers with its friendly syntax, object-oriented features, memory management, and best of all—the promise of portability. We'll take a quick dip and write some code, compile it, and run it. We're talking syntax, loops, branching, and what makes Java so cool. Dive in.



The Way Java Works	2
What you'll do in Java	3
A Very Brief History of Java	4
Code structure in Java	7
Writing a class with a main()	9
Simple boolean tests	13
Conditional branching	15
Coding a Serious Business	16
Phrase-O-Matic	19
Exercises	20
Exercise Solutions	25

# 2 A Trip to Objectville

**I was told there would be objects.** In Chapter 1, we put all of our code in the main() method. That's not exactly object-oriented. So now we've got to leave that procedural world behind and start making some objects of our own. We'll look at what makes object-oriented (OO) development in Java so much fun. We'll look at the difference between a class and an object. We'll look at how objects can improve your life.

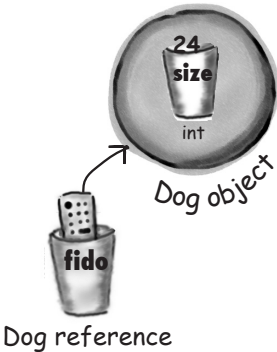


Chair Wars	28
Making your first object	36
Making and testing Movie objects	37
Quick! Get out of main!	38
Running the Guessing Game	40
Exercises	42
Exercise Solutions	47

# 3 Know Your Variables

Variables come in two flavors: primitive and reference.

There’s gotta be more to life than integers, Strings, and arrays. What if you have a PetOwner object with a Dog instance variable? Or a Car with an Engine? In this chapter we’ll unwrap the mysteries of Java types and look at what you can *declare* as a variable, what you can *put* in a variable, and what you can *do* with a variable. And we’ll finally see what life is truly like on the garbage-collectible heap.

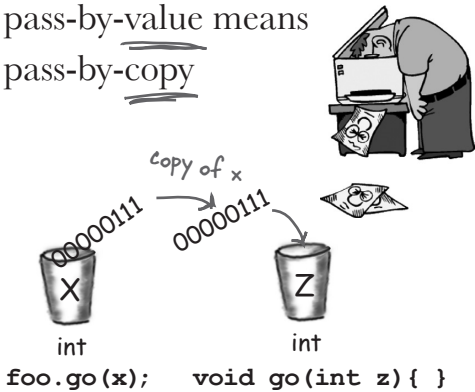


Declaring a variable	50
“I’d like a double mocha, no, make it an int.”	51
Back away from that keyword!	53
Controlling your Dog object	54
An object reference is just another variable value.	55
Life on the garbage-collectible heap	57
An array is like a tray of cups	59
A Dog example	62
Exercises	63
Exercise Solutions	68

# 4 How Objects Behave

State affects behavior, behavior affects state. We know that objects have **state** and **behavior**, represented by **instance variables** and **methods**. Now we’ll look at how state and behavior are *related*. An object’s behavior uses an object’s unique state. In other words, **methods use instance variable values**. Like, “if dog weight is less than 14 pounds, make yippy sound, else...” **Let’s go change some state!**

pass-by-value means  
pass-by-copy

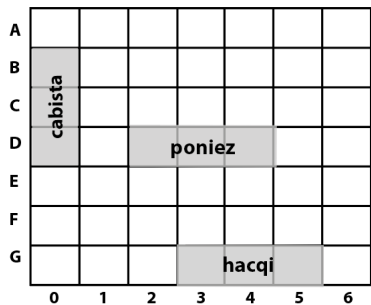


Remember: a class describes what an object knows and what an object does	72
The size affects the bark	73
You can send things to a method	74
You can get things <i>back</i> from a method.	75
You can send more than one thing to a method	76
Cool things you can do with parameters and return types	79
Encapsulation	80
How do objects in an array behave?	83
Declaring and initializing instance variables	84
Comparing variables (primitives or references)	86
Exercises	88
Exercise Solutions	93

# 5 Extra-Strength Methods

**Let’s put some muscle in our methods.** You dabbled with variables, played with a few objects, and wrote a little code. But you need more tools. Like **operators**. And **loops**. Might be useful to **generate random numbers**. And **turn a String into an int**, yeah, that would be cool. And why don’t we learn it all by *building* something real, to see what it’s like to write (and test) a program from scratch. **Maybe a game**, like Sink a Startup (similar to Battleship).

We’re gonna build the Sink a Startup game



Let’s build a Battleship-style game: “Sink a Startup”	96
Developing a Class	99
Writing the method implementations	101
Writing test code for the SimpleStartup class	102
The checkYourself() method	104
Prep code for the SimpleStartupGame class	108
The game’s main() method	110
Let’s play	113
More about for loops	114
The enhanced for loop	116
Casting primitives	117
Exercises	118
Exercise Solutions	122

# 6 Using the Java Library

**Java ships with hundreds of prebuilt classes.** You don’t have to reinvent the wheel if you know how to find what you need from the Java library, commonly known as the **Java API**. *You’ve got better things to do.* If you’re going to write code, you might as well write *only* the parts that are custom for your application. The core Java library is a giant pile of classes just waiting for you to use like building blocks.

“Good to know there’s an ArrayList in the java.util package. But by myself, how would I have figured that out?”

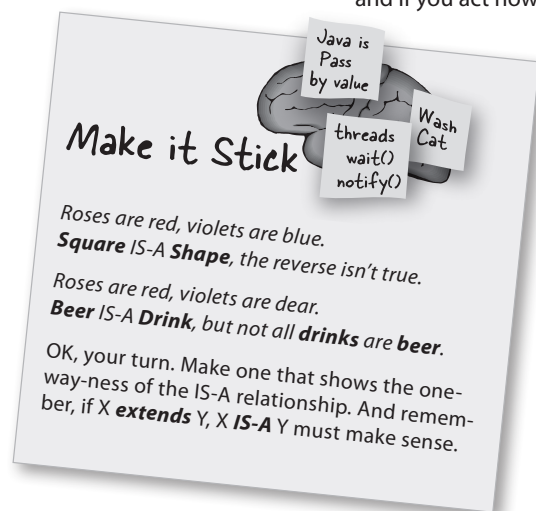


- Julia, 31, hand model

In our last chapter, we left you with the cliff-hanger. A bug.	126
Wake up and smell the library	132
Some things you can do with ArrayList	133
Comparing ArrayList to a regular array	137
Let’s build the REAL game: “Sink a Startup”	140
Prep code for the real StartupBust class	144
The final version of the Startup class	150
Super Powerful Boolean Expressions	151
Using the Library (the Java API)	154
Exercises	163
Exercise Solutions	165

# 7 Better Living in Objectville

**Plan your programs with the future in mind.** What if you could write code that someone *else* could extend, **easily**? What if you could write code that was flexible, for those pesky last-minute spec changes? When you get on the Polymorphism Plan, you'll learn the 5 steps to better class design, the 3 tricks to polymorphism, the 8 ways to make flexible code, and if you act now—a bonus lesson on the 4 tips for exploiting inheritance.

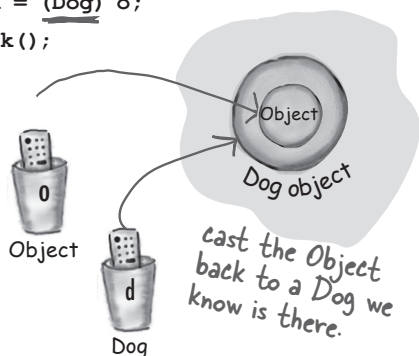


Chair Wars Revisited...	168
Understanding inheritance	170
Let's design the inheritance tree for an Animal simulation program	172
Looking for more inheritance opportunities	175
Using IS-A and HAS-A	179
How do you know if you've got your inheritance right?	181
When designing with inheritance, are you using or abusing?	183
Keeping the contract: rules for overriding	192
Overloading a method	193
Exercises	194
Exercise Solutions	197

# 8 Serious Polymorphism

**Inheritance is just the beginning.** To exploit polymorphism, we need interfaces. We need to go beyond simple inheritance to flexibility you can get only by designing and coding to interfaces. What's an interface? A 100% abstract class. What's an abstract class? A class that can't be instantiated. What's that good for? Read the chapter...

```
Object o = al.get(id);
Dog d = (Dog) o;
d.bark();
```

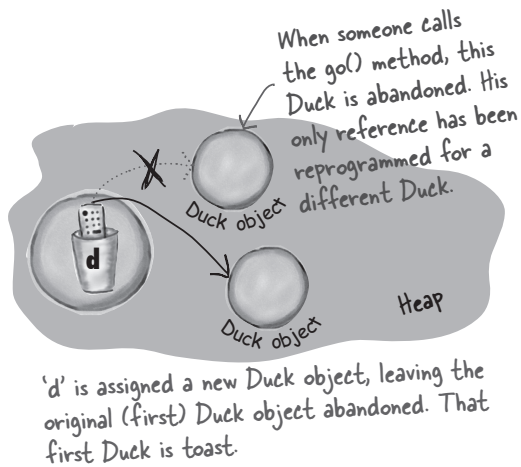


Did we forget about something when we designed this?	200
The compiler won't let you instantiate an abstract class	203
Abstract vs. Concrete	204
You MUST implement all abstract methods	206
Polymorphism in action	208
Why not make a class generic enough to take anything?	210
When a Dog won't act like a Dog	214
Let's explore some design options	221
Making and Implementing the Pet interface	227
Invoking the superclass version of a method	230
Exercises	232
Exercise Solutions	235



# 9 Life and Death of an Object

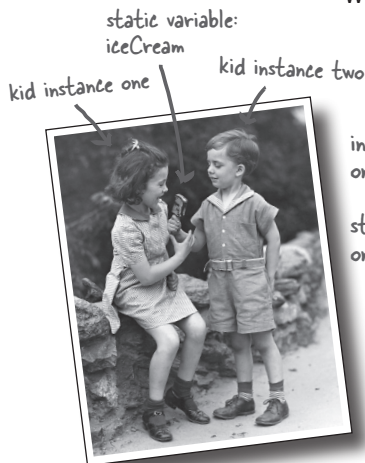
**Objects are born and objects die.** You're in charge. You decide when and how to *construct* them. You decide when to *abandon* them. The **Garbage Collector (gc)** reclaims the memory. We'll look at how objects are created, where they live, and how to keep or abandon them efficiently. That means we'll talk about the heap, the stack, scope, constructors, super constructors, null references, and gc eligibility.



The Stack and the Heap: where things live	238
Methods are stacked	239
What about local variables that are objects?	240
The miracle of object creation	242
Construct a Duck	244
Doesn't the compiler always make a no-arg constructor for you?	248
Nanoreview: four things to remember about constructors	251
The role of superclass constructors in an object's life	253
Can the child exist before the parents?	256
What about reference variables?	262
I don't like where this is headed.	263
Exercises	268
Exercise Solutions	272

# 10 Numbers Matter

**Static variables are shared by all instances of a class.**



**Do the Math.** The Java API has methods for absolute value, rounding, min/max, etc. But what about formatting? You might want numbers to print exactly two decimal points, or with commas in all the right places. And you might want to print and manipulate dates, too. And what about parsing a String into a number? Or turning a number into a String? We'll start by learning what it means for a variable or method to be *static*.

MATH methods: as close as you'll ever get to a <i>global</i> method	276
The difference between regular (non-static) and static methods	277
Initializing a static variable	283
Math methods	288
Wrapping a primitive	290
Autoboxing works almost everywhere	292
Turning a primitive number into a String	295
Number formatting	296
The format specifier	300
Exercise	306
Exercise Solutions	308

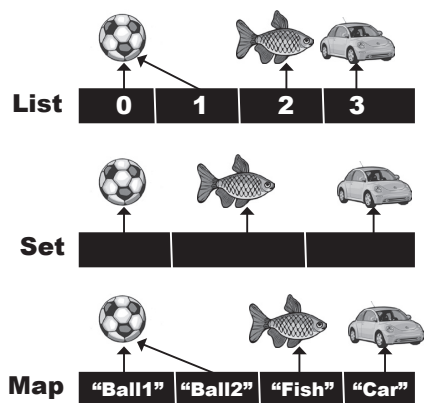


# 11 Data Structures

## Sorting is a snap in Java

your data without having to write

**Sorting is a snap in Java.** You have all the tools for collecting and manipulating your data without having to write your own sort algorithms. The Java Collections Framework has a data structure that should work for virtually anything you'll ever need to do. Want to keep a list that you can easily keep adding to? Want to find something by name? Want to create a list that automatically takes out all the duplicates? Sort your co-workers by the number of times they've stabbed you in the back?



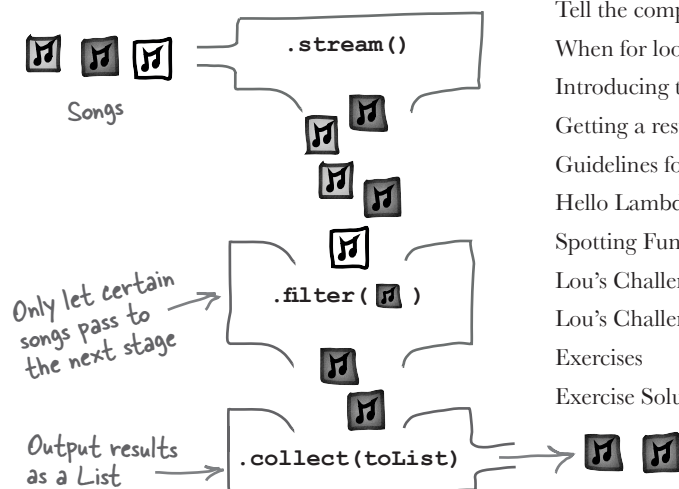
Exploring the java.util API, List and Collections	314
Generics means more type-safety	320
Revisiting the sort() method	327
The new, improved, comparable Song class	330
Sorting using only Comparators	336
Updating the Jukebox code with Lambdas	342
Using a HashSet instead of ArrayList	347
What you MUST know about TreeSet...	353
We've seen Lists and Sets, now we'll use a Map	355
Finally, back to generics	358
Exercise Solutions	364

## 12 Lambdas and Streams: What, Not How

What if...you didn't need to tell the computer HOW to do something? In this chapter we'll look at the Streams API. You'll see how behav-

## What if...you didn't need to tell the computer HOW to do something?

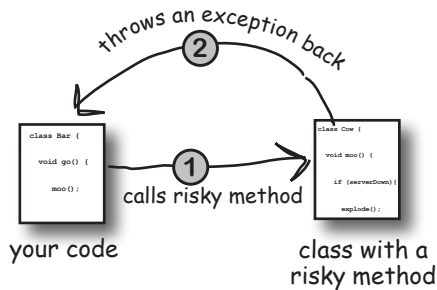
In this chapter we'll look at the Streams API. You'll see how helpful lambda expressions can be when you're using streams, and you'll learn how to use the Streams API to query and transform the data in a collection.



Tell the computer WHAT you want	370
When for loops go wrong	372
Introducing the Streams API	375
Getting a result from a Stream	378
Guidelines for working with streams	384
Hello Lambda, my (not so) old friend	368
Spotting Functional Interfaces	396
Lou's Challenge #1: Find all the "rock" songs	400
Lou's Challenge #2: List all the genres	404
Exercises	415
Exercise Solutions	417

# 13 Risky Behavior

**Stuff happens.** The file isn't there. The server is down. No matter how good a programmer you are, you can't control *everything*. When you write a risky method, you need code to handle the bad things that might happen. But how do you *know* when a method is risky? Where do you put the code to *handle* the **exceptional** situation? In *this* chapter, we're going to build a MIDI Music Player that uses the risky JavaSound API, so we better find out.



Let's make a Music Machine	422
First we need a Sequencer	424
An exception is an object...of type Exception	428
Flow control in try/catch blocks	432
Did we mention that a method can throw more than one exception?	435
Multiple catch blocks must be ordered from smallest to biggest	438
Ducking (by declaring) only delays the inevitable	442
Code Kitchen	445
Version 1: Your very first sound player app	448
Version 2: Using command-line args to experiment with sounds	452
Exercises	454
Exercise Solutions	457

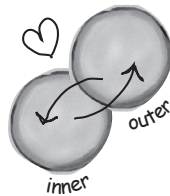
# 14 A Very Graphic Story

**Face it, you need to make GUIs.** Even if you believe that for the rest of your life you'll write only server-side code, sooner or later you'll need to write tools, and you'll want a graphical interface. We'll spend two chapters on GUIs and learn more language features including **Event Handling** and **Inner Classes**. We'll put a button on the screen, we'll paint on the screen, we'll display a JPEG image, and we'll even do some animation.

```
class MyOuter {
    class MyInner {
        void go() {
        }
    }
}
```

The outer and inner objects are now intimately linked.

These two objects on the heap have a special bond. The inner can use the outer's variables (and vice versa).



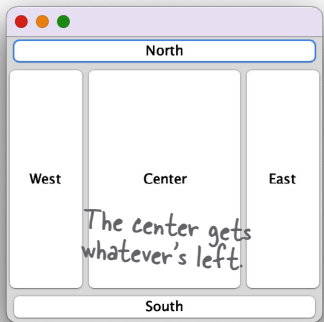
It all starts with a window	462
Getting a user event	465
Listeners, Sources, and Events	469
Make your own drawing widget	472
Fun things to do in <code>paintComponent()</code>	473
GUI layouts: putting more than one widget on a frame	478
Inner class to the rescue!	484
lambdas to the rescue! (again)	490
Using an inner class for animation	492
An easier way to make messages/events	498
Exercises	502
Exercise Solutions	507

# 15 Work on Your Swing

**Swing is easy.** Unless you actually *care* where everything goes. Swing code *looks* easy, but then compile it, run it, look at it, and think, “hey, *that’s* not supposed to go *there*.” The thing that makes it *easy* to *code* is the thing that makes it *hard* to *control*—the **Layout Manager**. But with a little work, you can get layout managers to submit to your will. In this chapter, we’ll work on our Swing and learn more about widgets.

Components in the east and west get their preferred width.

Things in the north and south get their preferred height.

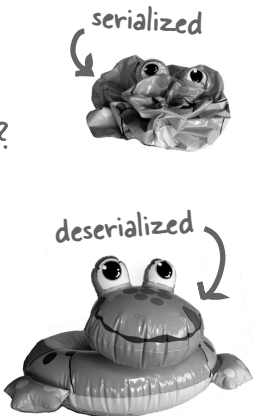


Swing components	510
Layout Managers	511
The Big Three layout managers: border, flow, and box.	513
Playing with Swing components	523
Code Kitchen	526
Making the BeatBox	529
Exercises	534
Exercise Solutions	537

# 16 Saving Objects (and Text)

**Objects can be flattened and inflated.** Objects have state and behavior. Behavior lives in the class, but *state* lives within each individual *object*. If your program needs to save state, *you can do it the hard way*, interrogating each object, painstakingly writing the value of each instance variable. Or, **you can do it the easy OO way**—you simply freeze-dry the object (serialize it) and reconstitute (deserialize) it to get it back.

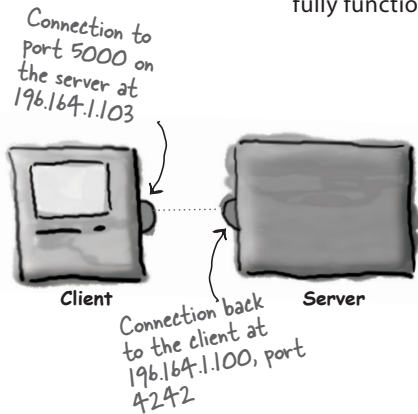
Any questions?



Writing a serialized object to a file	542
If you want your class to be serializable, implement Serializable	547
Deserialization: restoring an object	551
Version ID: A Big Serialization Gotcha	556
Writing a String to a Text File	559
Reading from a Text File	566
Quiz Card Player (code outline)	567
Path, Paths, and Files (messing with directories)	573
Finally, a closer look at finally	574
Saving a BeatBox pattern	579
Exercises	580
Exercise Solutions	584

# 17 Make a Connection

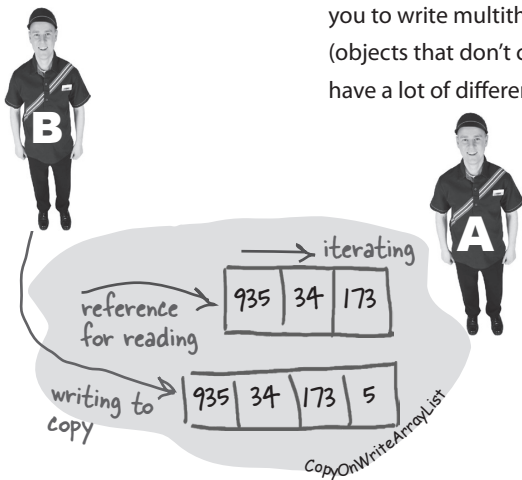
**Connect with the outside world.** It's easy. All the low-level networking details are taken care of by classes in the `java.net` library. One of Java's best features is that sending and receiving data over a network is really just I/O with a slightly different connection stream at the end of the chain. In this chapter we'll make client sockets. We'll make server sockets. We'll make clients and servers. Before the chapter's done, you'll have a fully functional, multithreaded chat client. Did we just say *multithreaded*?



Connecting, Sending, and Receiving	590
The DailyAdviceClient	598
Writing a simple server application	601
Java has multiple threads but only one Thread class	610
The three states of a new thread	616
Putting a thread to sleep	622
Making and starting two threads (or more!)	626
Closing time at the thread pool	629
New and improved SimpleChatClient	632
Exercises	631
Exercise Solutions	636

# 18 Dealing with Concurrency Issues

**Doing two or more things at once is hard.** Writing multithreaded code is easy. Writing multithreaded code that works the way you expect can be much harder. In this final chapter, we're going to show you some of the things that can go wrong when two or more threads are working at the same time. You'll learn about some of the tools in `java.util.concurrent` that can help you to write multithreaded code that works correctly. You'll learn how to create immutable objects (objects that don't change) that are safe for multiple threads to use. By the end of the chapter, you'll have a lot of different tools in your toolkit for working with concurrency.

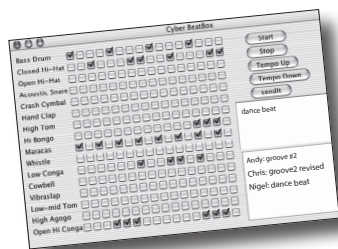


The Ryan and Monica problem, in code	642
Using an object's lock	647
The dreaded "Lost Update" problem	650
Make the <code>increment()</code> method atomic. Synchronize it!	652
Deadlock, a deadly side of synchronization	654
Compare-and-swap with atomic variables	656
Using immutable objects	659
More problems with shared data	662
Use a thread-safe data structure	664
Exercises	668
Exercise Solutions	670

# A

## Appendix A

**Final Code Kitchen.** All the code for the full client-server chat beat box. Your chance to be a rock star.



Final BeatBox client program	674
Final BeatBox server program	681

# B

## Appendix B

**The top ten-ish topics that didn't make it into the rest of the book.** We can't send you out into the world just yet. We have a few more things for you, but this *is* the end of the book. And this time we really mean it.

#11 JShell (Java REPL)	684
#10 Packages	685
#9 Immutability in Strings and Wrappers	688
#8 Access levels and access modifiers (who sees what)	689
#7 Varargs	691
#6 Annotations	692
#5 Lambdas and Maps	693
#4 Parallel Streams	695
#3 Enumerations (also called enumerated types or enums)	696
#2 Local Variable Type Inference (var)	698
#1 Records	699

# i Index