Head First Java

- Learn how threads can change your life
- Avoid embarrassing OO mistakes
- Fool around in the Java Library
- Make Java concepts stick to your brain
- Bend your mind around 42 Java puzzles
- Make attractive and useful GUIs

Kathy Sierra & Bert Bates
# Table of Contents (summary)

| Intro                                           | xxi |
| 1 Breaking the Surface: *a quick dip*           | 1   |
| 2 A Trip to Objectville: *yes, there will be objects* | 27  |
| 3 Know Your Variables: *primitives and references* | 49  |
| 4 How Objects Behave: *object state affects method behavior* | 71  |
| 5 Extra-Strength Methods: *flow control, operations, and more* | 95  |
| 6 Using the Java Library: *so you don’t have to write it all yourself* | 125 |
| 7 Better Living in Objectville: *planning for the future* | 165 |
| 8 Serious Polymorphism: *exploiting abstract classes and interfaces* | 197 |
| 9 Life and Death of an Object: *constructors and memory management* | 235 |
| 10 Numbers Matter: *math, formatting, wrappers, and statics* | 273 |
| 11 Risky Behavior: *exception handling*         | 315 |
| 12 A Very Graphic Story: *intro to GUI, event handling, and inner classes* | 353 |
| 13 Work on Your Swing: *layout managers and components* | 399 |
| 14 Saving Objects: *serialization and I/O*      | 429 |
| 15 Make a Connection: *networking sockets and multithreading* | 471 |
| 16 Data Structures: *collections and generics*  | 529 |
| 17 Release Your Code: *packaging and deployment*| 581 |
| 18 Distributed Computing: *RMI with a dash of servlets, EJB, and Jini* | 607 |
| A Appendix A: *Final code kitchen*              | 649 |
| B Appendix B: *Top Ten Things that didn’t make it into the rest of the book* | 659 |
| Index                                           | 677 |

# Table of Contents (the full version)

## 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? xxii
- What your brain is thinking xxiii
- Metacognition xxv
- Bend your brain into submission xxvii
- What you need for this book xxviii
- Technical editors xxx
- Acknowledgements xxxi
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
- Code structure in Java
- Anatomy of a class
- The main() method
- Loops
- Conditional branching (if tests)
- Coding the “99 bottles of beer” app
- Phrase-o-matic
- Fireside chat: compiler vs. JVM
- Exercises and puzzles

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 (Brad the OO guy vs. Larry the procedural guy)
- Inheritance (an introduction)
- Overriding methods (an introduction)
- What's in a class? (methods, instance variables)
- Making your first object
- Using main()
- Guessing Game code
- Exercises and puzzles
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.

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!
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 Dot Com (similar to Battleship).

Building the Sink a Dot Com game
Starting with the Simple Dot Com game (a simpler version)
Writing prepcode (pseudocode for the game)
Test code for Simple Dot Com
Coding the Simple Dot Com game
Final code for Simple Dot Com
Generating random numbers with Math.random()
Ready-bake code for getting user input from the command-line
Looping with for loops
Casting primitives from a large size to a smaller size
Converting a String to an int with Integer.parseInt()
Exercises and puzzles

6
Using the Java Library
Java ships with hundreds of pre-built 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

Analyzing the bug in the Simple Dot Com Game
ArrayList (taking advantage of the Java API)
Fixing the DotCom class code
Building the real game (Sink a Dot Com)
Prepcode for the real game
Code for the real game
boolean expressions
Using the library (Java API)
Using packages (import statements, fully-qualified names)
Using the HTML API docs and reference books
Exercises and puzzles
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.

Make it Stick

Roses are red, violets are blue.
Square IS-A Shape, the reverse isn't true.

Roses are red, violets are dear.
Beer IS-A Drink, but not all drinks are beer.

OK, your turn. Make one that shows the one-way-ness of the IS-A relationship. And remember, if X extends Y, X IS-A Y must make sense.

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();
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 objects and variables live
Methods on the stack
Where local variables live
Where instance variables live
The miracle of object creation
Constructors (the code that runs when you say new)
Initializing the state of a new Duck
Overloaded constructors
Superclass constructors (constructor chaining)
Invoking overloaded constructors using this()
Life of an object
Garbage Collection (and making objects eligible)
Exercises and puzzles

Numbers Matter

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 class (do you really need an instance of it?)
static methods
static variables
Constants (static final variables)
Math methods (random(), round(), abs(), etc.)
Wrapper classes (Integer, Boolean, Character, etc.)
Autoboxing
Number formatting
Date formatting and manipulation
Static imports
Exercises and puzzles
11 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.

```
class Cow {
  void moo() {
    if (serverDown) {
      explode();
    }
  }
}
```

```
class Bar {
  void go() {
    moo();
  }
  int stuff() {
    x.beep();
  }
}
```

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

The compiler guarantees (it checks) that you’re aware of the risks.

- Making a music machine (the BeatBox) 316
- What if you need to call risky code? 319
- Exceptions say “something bad may have happened...” 320
- The compiler guarantees (it checks) that you’re aware of the risks 321
- Catching exceptions using a `try/catch` (skateboarder) 322
- Flow control in `try/catch` blocks 326
- The `finally` block (no matter what happens, turn off the oven!) 327
- Catching multiple exceptions (the order matters) 329
- Declaring an exception (just duck it) 335
- Handle or declare law 337
- Code Kitchen (making sounds) 339
- Exercises and puzzles 348

12 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.

- Your first GUI 355
- Getting a user event 357
- Implement a listener interface 358
- Getting a button’s `ActionEvent` 360
- Putting graphics on a GUI 363
- Fun with `paintComponent()` 365
- The `Graphics2D` object 366
- Putting more than one button on a screen 370
- Inner classes to the rescue (make your listener an inner class) 376
- Animation (move it, paint it, move it, paint it, move it, paint it...) 382
- Code Kitchen (painting graphics with the beat of the music) 386
- Exercises and puzzles 394

Exercises and puzzles 348
13

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.

Swing Components 400
Layout Managers (they control size and placement) 401
Three Layout Managers (border, flow, box) 403
BorderLayout (cares about five regions) 404
FlowLayout (cares about the order and preferred size) 408
BoxLayout (like flow, but can stack components vertically) 411
JTextField (for single-line user input) 413
JTextArea (for multi-line, scrolling text) 414
JCheckBox (is it selected?) 416
JList (a scrollable, selectable list) 417
Code Kitchen (The Big One - building the BeatBox chat client) 418
Exercises and puzzles 424

14

Saving Objects

**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.

Saving object state 431
Writing a serialized object to a file 432
Java input and output streams (connections and chains) 433
Object serialization 434
Implementing the Serializable interface 437
Using transient variables 439
Deserializing an object 441
Writing to a text file 447
java.io.File 452
Reading from a text file 454
Splitting a String into tokens with split() 458
CodeKitchen 462
Exercises and puzzles 466
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?
Release Your Code

It’s time to let go. You wrote your code. You tested your code. You refined your code. You told everyone you know that if you never saw a line of code again, that’d be fine. But in the end, you’ve created a work of art. The thing actually runs! But now what? In these final two chapters, we’ll explore how to organize, package, and deploy your Java code. We’ll look at local, semi-local, and remote deployment options including executable jars, Java Web Start, RMI, and Servlets. Relax. Some of the coolest things in Java are easier than you think.

Deployment options 582
Keep your source code and class files separate 584
Making an executable JAR (Java ARchives) 585
Running an executable JAR 586
Put your classes in a package! 587
Packages must have a matching directory structure 589
Compiling and running with packages 590
Compiling with -d 591
Making an executable JAR (with packages) 592
Java Web Start (JWS) for deployment from the web 597
How to make and deploy a JWS application 600
Exercises and puzzles 601

Data Structures

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?

Collections 533
Sorting an ArrayList with Collections.sort() 534
Generics and type-safety 540
Sorting things that implement the Comparable interface 547
Sorting things with a custom Comparator 552
The collection API—lists, sets, and maps 557
Avoiding duplicates with HashSet 559
Overriding hashCode() and equals() 560
HashMap 567
Using wildcards for polymorphism 574
Exercises and puzzles 576
Distributed Computing

Being remote doesn’t have to be a bad thing. Sure, things are easier when all the parts of your application are in one place, in one heap, with one JVM to rule them all. But that’s not always possible. Or desirable. What if your application handles powerful computations? What if your app needs data from a secure database? In this chapter, we’ll learn to use Java’s amazingly simple Remote Method Invocation (RMI). We’ll also take a quick peek at Servlets, Enterprise Java Beans (EJB), and Jini.

Appendix B

The Top Ten Things that didn’t make it into 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.

Index