Big Java
Big Java

Cay Horstmann
San Jose State University

John Wiley & Sons, Inc.
Preface

This book is an introductory text in computer science, focusing on programming principles and practices. Why should you choose this book for your first course in computer science? Here are the key reasons:

- I take a point of view that goes beyond language syntax and focuses on computer science concepts.
- I stress the object-oriented paradigm, starting with the first example—an object-oriented version of the traditional “Hello, World” program.
- I motivate students to master the practical aspects of programming, with lots of useful tips and a chapter on testing and debugging.
- I present a carefully selected subset of the Java library that is accessible to beginners and rich enough to create interesting programs.
- I use the standard Java language, library, and tools—not a specialized “training wheels” environment.
- In the final chapters of the book, I cover important techniques for server-side program development such as database programming, XML and Java Server Pages.

The Use of Java

This book is based on the Java programming language. I chose Java for four reasons:

- Object orientation
- Safety
- Simplicity
- Breadth of the standard library

At this point, the object-oriented point of view is the predominant paradigm for software design. I strongly believe that object orientation enables students to spend more time on the design of their programs and less time coding and debugging. In this book, I start out with objects and classes early. Students learn how to manipulate objects and build simple classes in Chapter 2.
I rarely use static methods other than `main`. As a result, students think in terms of objects from the outset—they don’t have to spend the second half of the course unlearning bad habits developed in the first half.

When designing classes, I strictly separate the classes from the test driver programs. (In fact, if you use an environment such as BlueJ, then you don’t need the test driver programs at all. This book doesn’t require that you use BlueJ or any other particular environment, but it works very well with BlueJ. Give it a try and you too may become a convert—my students were delighted when they were able to interact with their objects in an intuitive fashion.)

Another notable aspect of this book is that I cover interfaces before subclasses. This has a great advantage: Students see the power of polymorphism before having to worry about technicalities of extending classes.

Of course, there are many object-oriented programming languages besides Java. In principle, one can teach object-oriented programming using the C++ language. However, Java has a fundamental advantage over C++, namely its safety. Students can—and do—make an amazing number of errors when using C++, many of which lead to mysterious and irreproducible program behavior. When using C++, an instructor must spend a great deal of class time on safe programming habits, or students will end up with a well-deserved lack of confidence in their creations—hardly an ideal situation for a beginning course.

Another major advantage of Java is its simplicity. Although it is not a reasonable goal to cover all constructs of Java in the first course, instructors can master all of the syntax and semantics of the Java language and can answer student questions with complete confidence. In contrast, the C++ language is so complex that very few people can truthfully state that they understand all features of the language. Even though I have used C++ extensively for over a dozen years, I regularly get stumped by freshmen who show me a particularly baffling compiler error message. Simplicity is important, especially for a foundational course. It is not a good practice to choose as a foundational tool a programming language that students and instructors cannot master with confidence.

Finally, the standard Java library has sufficient breadth that it can be used for most courses in a computer science curriculum. Graphics, user interface construction, database access, multithreading, and network programming are all part of the standard library. Thus, the skills that students learn in the beginning course will serve them well throughout the curriculum. Again, C++ falls notably short in this regard. There are no standard toolkits for any of the above-mentioned programming domains in C++. The Java library subset that this book covers enables students to handle a wide variety of common programming tasks.

A Tour of the Book

The book can be naturally grouped into four parts. Figure 1 shows the dependencies among the chapters.

Part A

Chapters 1 through 8 cover the fundamentals of object-based programming: objects, methods, classes, variables, number types, strings, and control structures. Students learn how to build very simple classes in Chapter 2. Chapter 7 takes up the subject of class design in a more systematic fashion.
Preface

Figure 1
Chapter Dependencies
Starting with Chapter 7, I use a very small subset of the UML notation—just class diagrams and four arrow types for dependency, realization of interfaces, inheritance, and directed association. That small subset is useful for visualizing the relationships between classes, but it minimizes issues that beginners find complex, such as when to choose association, aggregation, or attributes.

I cover graphics very early, in Chapter 4, because many students enjoy writing programs that create drawings, and because rectangles, ellipses, and lines are good examples of objects. I use the “2D graphics” classes in the java.awt.geom package throughout, not the outdated procedural methods in the Graphics class. Calling g.drawRect(x,y,w,h) is not object-oriented. Manipulating geometric objects is both object-oriented and fun for the students. I use applets because students can program them with very little technical knowledge.

However, coverage of graphics is entirely optional. All material has been carefully presented so that you can skip all chapters that cover graphics and graphical user interfaces.

Chapter 8 covers testing and debugging, a subject that is unfortunately given short shrift in many textbooks.

As explained in Chapter 3, you can either use the JOptionPane class to read input from a dialog box (even in a console program), or you can use a BufferedReader. The latter forces you to tag the main method with throws IOException, which I used to find unacceptable until I reorganized all programs so that the main method is just a “throwaway” test driver class. I don’t find it problematic if methods in a test driver class throw exceptions. (By the end of the first semester, students will know how to catch exceptions.)

Part B

Chapters 9 through 16 cover inheritance, arrays, exceptions, streams and, optionally, graphical user interface (GUI) programming.

The discussion of inheritance is split into two chapters. Chapter 9 covers interfaces and polymorphism, whereas Chapter 11 covers inheritance. Introducing interfaces before inheritance pays off in several ways. Students immediately see polymorphism before getting bogged down with superclass construction. It becomes possible to discuss event-driven programming at an early stage. Students are naturally led to local inner classes for event handlers, a more robust technique than the “opportunistic” realization of event interfaces that one still finds in older textbooks.

GUI programming is split into two chapters. Chapter 10 covers event-driven programming, relying just on the notion of an interface introduced in Chapter 9. Chapter 12 covers GUI components and their layout. This chapter requires some knowledge of inheritance (extending frames and panels and invoking super.paintComponent). It is possible to cover both of these chapters together, either before or after Chapter 11.

Again, let me stress that coverage of graphics and graphical user interfaces is entirely optional. One alternative is to cover graphics and applets (which are quite simple to program) and skip GUIs and event handling.

I cover arrays and streams after inheritance. From an object-oriented point of view, inheritance is a crucial concept, and I find that it helps to introduce it as early as possible. However, if you prefer to cover arrays and streams earlier, you can simply switch the chapters without incurring any problems.
Preface

I prefer to cover array lists first before covering arrays. In my experience, students find the \texttt{get/set} syntax perfectly natural and have surprisingly little attachment to the \texttt{[]} operator. They aren’t even overly bothered by the cast required when using the \texttt{get} method. By using array lists, you avoid the unpleasantness of partially filled arrays altogether—it is no wonder that most professional programmers use array lists (or vectors) all the time and rarely resort to arrays. Of course, you need arrays for numbers, but lists of numbers aren’t all that common in object-oriented programs.

I highly recommend covering object streams and serialization, especially if the course involves significant programming projects. In my experience, students are delighted when they discover that they can store the entire state of their application with a single \texttt{writeObject} call and retrieve it again just as easily.

Part C

Chapters 17 through 20 contain an introduction to algorithms and data structures, covering recursion, sorting and searching, linked lists, binary trees, and hash tables. These topics are probably outside the scope of a one-semester course.

When discussing recursion, I find that the object-oriented point of view is very helpful. In my introductory examples, an object that solves a problem recursively constructs another object of the same class that solves a simpler problem. Having the other object do the simpler job is much more plausible to students than having a function call itself.

I place the data structures into the context of the standard Java collections library. Students will learn the essential abstractions of the standard library (such as iterators, sets, and maps) as well as the performance characteristics of the various collections. However, a detailed discussion of the implementation of data structures is beyond the scope of this book.

Part D

Chapters 21 through 25 cover advanced Java programming techniques that definitely go beyond a first course in Java. Although, as already mentioned, a comprehensive coverage of the Java library would span many volumes, many instructors suggested that a textbook should give students additional reference material of lasting value beyond their first course. Some institutions also teach a second-semester course that covers more practical programming aspects such as database and network programming, rather than the more traditional in-depth material on data structures and algorithms. Personally, I think this is a very viable approach. Thus, this book can be used in a two-semester course to give students an introduction into programming fundamentals and a broad coverage of applications. Alternatively, the material in the final chapters can be useful for student projects.

When selecting topics for the latter part of the book, I purposefully included those technologies that are of particular interest to \textit{server-side programming}, such as networking, databases, XML, JavaServer Pages, and servlets, rather than including more advanced material on user interface construction. The Internet has made it possible to deploy many useful applications on servers, often accessed by nothing more than a browser. This new server-centric approach to application development was in part made possible by the Java language and libraries, and today, most of the industrial use of Java is in server-side programming.
Appendices

Appendix A1 contains a style guide for use with this book. I have found it highly beneficial to require a consistent style for all assignments. If this style guide conflicts with instructor sentiment or local customs, it can be modified. The style guide is available in electronic form for this purpose. Other appendices contain an overview over the parts of the standard library that this book covers, as well as quick references on HTML, UML, Java syntax, Unicode, the applet viewer, and javadoc.

The Pedagogical Structure

The beginning of each chapter has the customary overview of chapter objectives and motivational introduction. Throughout each chapter, margin notes show the places at which new concepts are introduced. The notes are summarized at the end of the chapter.

Throughout the chapters, there are five sets of notes to help your students, namely those entitled “Common Errors”, “Productivity Hints”, “Quality Tips”, “Advanced Topics”, and “Random Facts”. These notes are specially marked so that they don’t interrupt the flow of the main material. I expect that most instructors cover only a few of these notes in class and assign others for home reading. Some notes are quite short; others extend over a page. I decided to give each note the space that is needed for a full and convincing explanation, rather than attempting to fit them into one-paragraph “tips”.

- **Common Errors** describe the kinds of errors that students often make, with an explanation of why the errors occur, and what to do about them. Most students quickly discover the Common Errors sections and read them on their own.

- **Quality Tips** explain good programming practices. Since most of them require an initial investment of effort, these notes carefully motivate the reason behind the advice, and explain why the effort will be repaid later.

- **Productivity Hints** teach students how to use their tools more effectively. Many beginning students put little thought into their use of computers and software. They are often unfamiliar with tricks of the trade such as keyboard shortcuts, global search and replace, or automation of common tasks with scripts.

- **Advanced Topics** cover nonessential or more difficult material. Some of these topics introduce alternative syntactical constructions that are not necessarily technically advanced. In many cases, the book uses one particular language construct but explains alternatives as Advanced Topics. Instructors and students should feel free to use those constructs in their own programs if they prefer them. It has, however, been my experience that many students are grateful for the “keep it simple” approach, because it greatly reduces the number of gratuitous decisions they have to make.

- **Random Facts** provide historical and social information on computing, as required to fulfill the “historical and social context” requirements of the ACM curriculum guidelines, as well as capsule reviews of advanced computer science topics. Many students will read the Random Facts on their own while pretending to follow the lecture.
Preface

New to this edition is a set of HOWTO sections, inspired by the Linux HOWTO guides. These sections are intended to answer the common student question: “Now what do I do?”, by giving step-by-step instructions for common tasks.

Web Resources

Additional resources are found on the book’s web site at http://www.wiley.com/college/horstmann. These resources include:

- Solutions to selected exercises (accessible to students)
- Solutions to all exercises (for instructors only)
- A test bank
- A laboratory manual
- A list of frequently asked questions
- Help with common compilers
- Presentation slides for lectures
- Discussion boards for instructors and students
- Source code for all examples in the book
- The programming style guide in electronic form, so you can modify it to suit local preferences
- A “crash course in C++” that takes students rapidly from the material covered in this book to C++ programming

Acknowledgments

Many thanks to Paul Crockett, Bill Zobrist, Katherine Hepburn and Lisa Gee at John Wiley & Sons and Jerome Colburn, Lori Martinsek, and the team at Publication Services for their hard work and support for this book project.

I am very grateful to the many individuals who reviewed the manuscript, made valuable suggestions and brought an embarrassingly large number of errors and omissions to my attention. They include:

Sven Anderson, University of North Dakota, Robert Burton, Brigham Young University, Bruce Ellinbogen, University of Michigan-Dearborn, John Franco, University of Cincinnati, Rick Giles, Acadia University, John Gray, University of Hartford, Joann Houlihan, John Hopkins University, Richard Kick, Hinsdale Central High School, Michael Kölling, University of Southern Denmark, Miroslaw Majewski, Zayed University, Blaine Mayfield, Oklahoma State University, Hugh McGuire, University of California-Santa Barbara, Jim Miller, Bradley University, Jim Miller, University of Kansas, Don Needham, US Naval Academy, Ben Nystin, University of Colorado at Colorado Springs, Hugh O’Brien, University of California-Santa Barbara, Kathleen O’Brien, West Valley College, Richard Pattis, Carnegie Mellon University, Pete Peterson, Texas
A&M University, Sarah Pham, SGI, Stuart Reges, University of Arizona, Jim Roberts, Carnegie Mellon University, John Rose, University of South Carolina-Columbia, Kenneth Slonneger, University of Iowa, and Monica Sweck, University of Florida.

Finally, as always, my gratitude goes to my family—Hui-Chen, Thomas and Nina—for their never-ending encouragement and patience.
Contents

Preface v

Chapter 1 Introduction 1
  1.1 What Is a Computer? 2
  1.2 What Is Programming? 2
  1.3 The Anatomy of a Computer 3
  1.4 Translating Human-Readable Programs to Machine Code 8
  1.5 Programming Languages 10
  1.6 The Java Programming Language 11
  1.7 Becoming Familiar with Your Computer 13
  1.8 Compiling a Simple Program 17
  1.9 Errors 24
  1.10 The Compilation Process 26

Chapter 2 An Introduction to Objects and Classes 33
  2.1 Using and Constructing Objects 34
  2.2 Object Variables 37
  2.3 Defining a Class 42
  2.4 Testing a Class 44
  2.5 Instance Fields 47
  2.6 Constructors 49
  2.7 Designing the Public Interface of a Class 51
  2.8 Commenting the Public Interface 54
  2.9 Specifying the Implementation of a Class 58
  2.10 Variable Types 66
  2.11 Explicit and Implicit Method Parameters 67

Chapter 3 Fundamental Data Types 77
  3.1 Number Types 78
  3.2 Assignment 84
  3.3 Constants 88
  3.4 Arithmetic and Mathematical Functions 92
  3.5 Calling Static Methods 98
  3.6 Type Conversion 100
  3.7 Strings 107

xiii
CONTENTS

3.8 Reading Input  110
3.9 Characters  116
3.10 Comparing Primitive Types and Objects  119

►Chapter 4 Applets and Graphics  133
4.1 Why Applets?  135
4.2 A Brief Introduction to HTML  136
4.3 A Simple Applet  140
4.4 Graphical Shapes  145
4.5 Colors  147
4.6 Fonts  148
4.7 Drawing Complex Shapes  154
4.8 Reading Text Input  164
4.9 Comparing Visual and Numerical Information  167
4.10 Coordinate Transformations  171

►Chapter 5 Decisions  185
5.1 The if Statement  186
5.2 Comparing Values  191
5.3 Multiple Alternatives  197
5.4 Using Boolean Expressions  210

►Chapter 6 Iteration  227
6.1 while Loops  228
6.2 for Loops  236
6.3 Nested Loops  244
6.4 Processing Input  246
6.5 Random Numbers and Simulations  262

►Chapter 7 Designing Classes  281
7.1 Choosing Classes  282
7.2 Cohesion and Coupling  283
7.3 Accessor and Mutator Methods  286
7.4 Side Effects  287
7.5 Preconditions and Postconditions  293
7.6 Static Methods  297
7.7 Static Fields  299
7.8 Scope  304
7.9 Packages  310

►Chapter 8 Testing and Debugging  329
8.1 Unit Tests  330
8.2 Test Case Evaluation  337
## Contents

8.3 Regression Testing and Test Coverage 340  
8.4 Program Traces, Logging, and Assertions 342  
8.5 The Debugger 345  
8.6 A Sample Debugging Session 348  

**Chapter 9 Interfaces and Polymorphism 363**  
9.1 Developing Reusable Solutions 364  
9.2 Converting between Types 370  
9.3 Polymorphism 372  
9.4 Using a Strategy Interface for Improving Reusability 373  
9.5 Processing Timer Events 381  

**Chapter 10 Event Handling 395**  
10.1 Events, Event Listeners, and Event Sources 396  
10.2 Processing Mouse Input 398  
10.3 Processing Text Input 404  
10.4 Multiple Buttons with Similar Behavior 409  
10.5 Frame Windows 418  
10.6 Text Components 419  

**Chapter 11 Inheritance 429**  
11.1 An Introduction to Inheritance 430  
11.2 Inheritance Hierarchies 434  
11.3 Inheriting Instance Fields and Methods 436  
11.4 Subclass Construction 442  
11.5 Converting from Subclasses to Superclasses 443  
11.6 Access Control 451  
11.7 **Object**: The Cosmic Superclass 454  

**Chapter 12 Graphical User Interfaces 473**  
12.1 Using Inheritance to Customize Panels 474  
12.2 Layout Management 480  
12.3 Using Inheritance to Customize Frames 483  
12.4 Choices 489  
12.5 Menus 501  
12.6 Exploring the Swing Documentation 507  

**Chapter 13 Array Lists and Arrays 521**  
13.1 Array Lists 522  
13.2 Simple Array List Algorithms 526  
13.3 Storing Numbers in Array Lists 530  
13.4 Declaring and Accessing Arrays 530  
13.5 Copying Arrays 534
13.6 Partially Filled Arrays 537
13.7 Two-Dimensional Arrays 544

►Chapter 14 Exception Handling 557
14.1 Throwing Exceptions 558
14.2 Checked Exceptions 561
14.3 Designing Your Own Exception Types 565
14.4 Catching Exceptions 565
14.5 The finally Clause 568
14.6 A Complete Example 570

►Chapter 15 Streams 581
15.1 Streams, Readers, and Writers 582
15.2 Reading and Writing Text Files 585
15.3 File Dialogs 586
15.4 An Encryption Program 587
15.5 Command Line Arguments 592
15.6 Object Streams 595
15.7 Random Access 599

►Chapter 16 System Design 615
16.1 The Software Life Cycle 616
16.2 Discovering Classes 621
16.3 Relationships between Classes 624
16.4 Example: Printing an Invoice 629
16.5 Example: An Automatic Teller Machine 641

►Chapter 17 Recursion 667
17.1 Triangle Numbers 668
17.2 Permutations 672
17.3 Recursive Helper Methods 682
17.4 Mutual Recursions 683
17.5 The Efficiency of Recursion 689

►Chapter 18 Sorting and Searching 703
18.1 Selection Sort 704
18.2 Profiling the Selection Sort Algorithm 707
18.3 Analyzing the Performance of the Selection Sort Algorithm 710
18.4 Merge Sort 712
18.5 Analyzing the Merge Sort Algorithm 715
18.6 Searching 722
18.7 Binary Search 724
18.8 Searching and Sorting Real Data 726
Contents

▶ Chapter 19 An Introduction to Data Structures  737
  19.1 Using Linked Lists  738
  19.2 Implementing Linked Lists  742
  19.3 Abstract and Concrete Data Types  754
  19.4 Stacks and Queues  757

▶ Chapter 20 Advanced Data Structures  767
  20.1 Sets  768
  20.2 Maps  772
  20.3 Hash Tables  775
  20.4 Computing Hash Codes  782
  20.5 Binary Search Trees  787
  20.6 Using Tree Sets and Tree Maps  798

▶ Chapter 21 Multithreading  809
  21.1 Thread Basics  810
  21.2 Synchronization  819
  21.3 Avoiding Deadlocks  827
  21.4 An Application of Threads: Algorithm Animation  835

▶ Chapter 22 Internet Networking  851
  22.1 The Internet Protocol  852
  22.2 Application Level Protocols  856
  22.3 A Client Program  859
  22.4 A Server Program  862
  22.5 URL Connections  870
  22.6 Posting Form Data  874

▶ Chapter 23 Relational Databases  885
  23.1 Organizing Database Information  886
  23.2 Queries  894
  23.3 Installing a Database  903
  23.4 Database Programming in Java  908
  23.5 Case Study: A Bank Database  916
  23.6 Advanced Database Concepts  928

▶ Chapter 24 XML  941
  24.1 XML Tags and Documents  942
  24.2 Parsing XML Documents  953
  24.3 Creating XML Documents  970
  24.4 Document Type Definitions  977
  24.5 Parsing with Document Type Definitions  987