PREFACE

Java Programming: From the Ground Up begins with the fundamentals of programming, moves through the object-oriented paradigm, and concludes with an introduction to graphics and event-driven programming. The broad coverage of topics as well as the modularity of the text makes the book suitable for both introductory and intermediate-level programming courses. The text requires no prerequisites other than an enthusiasm for problem solving and a willingness to persevere.

KEY FEATURES OF THE TEXT

The style of this text is based on the following four principles:

1. **Fundamentals first**

   Our approach is neither "objects first" nor "objects late"; it's "fundamentals first." Our method is bottom up, starting with the basic concepts common to most programming languages: variables, selection, iteration, and methods. Once students understand the basic control structures, they can use them to build classes. Programming tools such as iteration, selection, and recursion are not the exclusive property of the object-oriented paradigm. Virtually every programming language, from Ada to ZPL, provides these tools. The text discusses these common features first before using them to build classes.

   Our experience in the classroom convinces us that this bottom-up approach is pedagogically sound and the best way to teach the material. Certainly, one learns how to use the tools of carpentry before building a house. We believe that the same principle applies to building classes. You might say that we present Java from the "grounds" up.

2. **Independent presentation of fundamental programming concepts, object-oriented concepts, GUIs, and event-driven paradigms**

   The text is modular. We first tackle basic programming structures, then the fundamentals of object-oriented programming, followed by graphics, GUIs, and events. The separation of graphics from basic programming structures is especially helpful to beginners, who when presented early with programs that mix fundamentals with GUI design, events, and OOP, have difficulty separating these concepts.

   Because the text is modular, it is appropriate for a variety of courses. For example, a course that teaches Java as a second language can proceed directly to "Part 2: Principles of Object-Oriented Programming." The basics common to most programming languages (selection, iteration, recursion, methods, arrays) are covered in Part 1 and not spread throughout the text. A student familiar with another language, such as C++, can easily find the Java counterpart to any fundamental control structure.

3. **Examples, examples, and more examples**

   Examples lead to understanding. Understanding leads to abstraction. Expecting students to immediately digest an abstraction that took a professional perhaps years to distill is unrealistic. Regardless of how clever or articulate the presentation, the practical teacher quickly resorts to examples so that the student can extract the general principles in context. Our text contains dozens of examples in the form of fully implemented programs. Moreover, our experience teaching introductory courses convinces

us that students rarely read examples spanning four or five pages. With that in mind, we have tried to keep our examples short, succinct, and occasionally entertaining.

4. **Independent and parallel presentation of related computer science topics**
   We present a variety of computer science topics that expand upon and enhance the study of a particular part of the Java toolbox. Optional "Bigger Picture" sections appear after the exercises of most chapters and are independent of each other. These optional segments provide an introduction to more advanced topics such as fractals, computer architecture, artificial intelligence, computer theory, bioinformatics, and trees.

PEDAGOGICAL FEATURES

Each chapter contains the following features:

1. **Objectives**—Each chapter begins with a list of concepts that the student will learn in that chapter.
2. **Just the Facts**—At the conclusion of each chapter, a summary of the fundamental ideas of the chapter can be reviewed at a glance.
3. **Bug Extermination**—At the end of each chapter is a short section on debugging with a summary of some commonly occurring bugs, and hints for how best to avoid them.
4. **Examples**—Examples permeate each chapter. Almost every numbered example is a standalone program. Many examples are dissected line by line. Each example follows the same easy-to-understand format: a problem description, a Java solution, typical output, and finally a discussion of the solution.
5. **Exercises**—Each chapter contains a variety of exercises and programming problems. The style and difficulty of the exercises and problems vary. There are:
   - crossword puzzles that test terminology,
   - short answer questions that check basic understanding,
   - debugging and tracing exercises that do not require a computer,
   - short programming problems that reinforce the concepts of the chapter, and
   - longer programming assignments that require some creativity and algorithm development.
6. **The Bigger Picture**—Following the exercises, a section entitled *The Bigger Picture* builds upon and extends the ideas covered in the chapter. Topics range from two's complement number representation, to the halting problem, to DNA sequencing. The material in *The Bigger Picture* sections is not prerequisite to any subsequent section of the text. Furthermore, one *Bigger Picture* segment does not depend upon another. Each stands entirely on its own. These sections may be included, assigned as supplemental reading, used in a closed lab setting, or skipped entirely, depending on the audience or time constraints. However, students who choose to tackle some or all of these sections will find a wealth of topics, each opening new roads of inquiry into computer science. The effort will provide students with a larger framework of ideas that extend beyond the study of programming.

THE CONTENTS

The text is divided into four parts:

1. The Fundamental Tools; 2. Principles of Object-Oriented Programming; 3. More Java Classes; and 4. Basic Graphics, GUIs, and Event-Driven Programming

Part 1: The Fundamental Tools

Part 1 consists of the standard programming constructs that exist in most programming languages: storage and control structures.

1. **Introduction to Computers and Java**
   Chapter 1 is a brief introduction to the hardware and software of a computer system. The chapter includes a discussion of programming languages, compilers, and the Java Virtual Machine.
2. **Expressions and Data Types**
   Chapter 2 begins with a few applications that display string output and moves gradually to examples that evaluate expressions. The chapter includes an introduction to the primitive data types: int, double, char, and boolean.
3. **Variables and Assignment**
   Variables are introduced in this chapter. Specifically, Chapter 3 addresses three questions:
   - How does an application obtain storage for data?
   - How does an application store data?
   - How does an application utilize stored data?

   Java's Scanner class is used for interactive input.
4. **Selection and Decision: if Statements**
   Chapter 4 covers selection via
   - the if statement,
   - the if-else statement, and
   - the switch statement.

   The chapter also includes a discussion of nested if statements.
5. **Repetition**
   Repetition is first introduced with the while statement, then the do-while statement, and finally the for loop. The chapter explains the stylistic differences among the loops and when each type of loop may be appropriate. There is a discussion of common errors that may lead to infinite loops or loops that are "off by one." The chapter includes examples of applications with nested loops.
6. **Methods**
   Methods are introduced as "black boxes" that accept input and return a value. Here, we present a number of methods from Java's Math class. The bulk of the chapter deals with "home grown" methods. Because we have not yet introduced classes and objects, all methods are static.
7. **Arrays and Lists: One Name for Many Data**
   This chapter covers arrays and array instantiation. Here, we first introduce the concept of a reference. The chapter includes an introduction to sorting and searching. After discussing two-dimensional arrays, the chapter concludes with a case study: *The Fifteen Puzzle.* The case study uses most of the concepts introduced in Part 1.
8. **Recursion**
   Recursion is the final topic of Part 1. The chapter begins with a simple example that does no more than print a message. Subsequent examples grow in complexity, leading to a discussion of tail recursion versus "classic" recursion as well as the Quicksort algorithm. A final case study, *The Design of an Anagram Generator,* ties the concepts together. The chapter emphasizes *recursive thinking.*

Part 2: Principles of Object-Oriented Programming

The heart of Part 2 is the object-oriented paradigm. With the tools of Part 1 mastered, students can concentrate on the principles of object-oriented programming. The concepts of Parts 1 and 2 are not in any way tied to building GUIs or event-driven programming. No side trips to loop-land or "by-the-ways" are necessary. Part 2 is comprised of the following chapters:

9. **Objects and Classes I: Encapsulation, Strings, and Things**
   Chapter 9 introduces encapsulation, classes, and objects. This first introduction to classes and objects is accomplished with examples of several Java classes, including:
   - Random
   - String
   - StringBuilder
   - File
   - DecimalFormat

   Here, students learn how to use text files for simple I/O.
10. **Objects and Classes II: Writing Your Own Classes**
    In Chapter 9, students learn about objects and classes by using a few prepackaged classes. In this chapter students learn how to write their own classes. The chapter discusses encapsulation and information hiding and gives meaning to a few mysterious words, such as public and static, that have been used in previous chapters. A final case study builds a simple audio player, which we dub a *myPod*.
11. **Designing with Classes and Objects**
    The sole topic of Chapter 11 is program design. This chapter consists of a single case study: an interactive poker game. We formulate a methodology for determining the appropriate classes and objects and how these objects interact. Our focus here is not the syntax, semantics, or mechanics of Java but problem solving and object-oriented design.
12. **Inheritance**
    We introduce inheritance as the second principle of object-oriented programming. Here, we contrast inheritance and composition. We also discuss the Object class and those Object methods inherited by all classes. The chapter includes a discussion of abstract classes and interfaces.
13. **Polymorphism**
    The final chapter of Part 2 is a discussion of polymorphism. If inheritance emphasizes the "sameness" of classes in a hierarchy, then polymorphism underscores the differences. The chapter discusses dynamic binding, using polymorphism with interfaces, and polymorphism as it relates to Object.

Part 3: More Java Classes

Part 3 is the most technical section of the text. Here, we examine the wrapper classes, exception classes, stream classes, and classes for random access files. We also introduce generics and several elementary data structures such as stacks, queues, and linked lists. Part 3 ends with a discussion of the Java Collections Framework.

14. **More Java Classes: Wrappers and Exceptions**
    Chapter 14 begins with a discussion of the wrapper classes. The chapter includes a discussion of auto-boxing and unboxing. The remainder of the chapter is

devoted to Java's Exception hierarchy. The chapter explains the throw-catch mechanism, the finally block, checked and unchecked exceptions, the throws clause, and how to create an Exception class.

15. **Stream I/O and Random Access Files**
    By far the most technical chapter of the text, Chapter 15 is a selective discussion of some of the Byte Stream and Character Stream classes as well as the connection between the Byte Stream hierarchy and the Character Stream hierarchy. The chapter contrasts text and binary files, gives examples of binary file I/O, and discusses object serialization. Random access files are also covered in this chapter.
16. **Data Structures and Generics**
    Chapter 16 begins with an introduction to Java's ArrayList class and generics. This leads to a discussion of several elementary data structures: stacks, queues, and linked lists. An implementation for each type of data structure is discussed.
17. **The Java Collections Framework**
    By examining the implementations of several classes in the Java Collections Framework, this chapter demonstrates how choosing the "wrong" class an lead to an inefficient application.

Part 4: Basic Graphics, GUIs, and Event-Driven Programming

Part 4 introduces graphics, graphical user interfaces, and event-driven programming.

18. **Graphics: AWT and Swing**
    Chapter 18 discusses Swing and AWT. The chapter emphasizes frame layout and discusses several layout managers. Here, we explain how to arrange graphical components within a window. We also include an introduction to the Graphics class.
19. **Event-Driven Programming**
    Event-driven programming is discussed in terms of the delegation event model. Applications that include buttons, labels, text fields, text areas, dialog boxes, checkboxes, radio buttons, mouse events, and menus fill out the rest of the chapter.
20. **A Case Study: Video Poker, Revisited**
    Chapter 20 revisits the case study of Chapter 11. Here the focus is on the design and implementation of a GUI for the text-based poker game developed in Chapter 11. The objective of this chapter is an understanding of the design principle that entails the separation of the data model from the interface, or more simply, the model from the view.

Appendix A: Java Keywords

Appendix B: The ASCII Character Set

Appendix C: Operator Precedence

Appendix D: Javadoc

This appendix describes how to use Sun's Javadoc tool to automatically generate documentation from Java source files.

Appendix E: Packages

Appendix E focuses on the use of packages to better organize large-scale applications with many classes.

TO THE INSTRUCTOR

How to Use This Book

This book is flexible and is designed to serve several audiences:

- For a college-level introduction to programming in Java, Parts 1 and 2 can be used alone or followed by Part 4 with selections from Part 3, depending on the pace and focus of the course. In a first course, we would omit the chapter on Stream classes (Chapter 15). Basic text file I/O is covered in Chapter 9.
- A course for students who already know a programming language can begin with Part 2 and refer to Part 1 as needed. This same approach could be used by an instructor who prefers "objects early."
- For high school students in an AP course, Parts 1 and 2 and selections from Part 3 cover the required Java topics. Chapter 15 can be skipped entirely.

Recursion appears as Chapter 8 at the conclusion of Part 1, prior to our introduction to object-oriented programming. We present recursion independent of object-oriented programming because recursion is a fundamental concept of program control independent of the programming paradigm. Although recursion appears at the end of Part 1, the topic can be delayed until the end of Part 2, or skipped entirely. Any example or exercise in the book that requires recursion is explicitly marked **(R)** so that an instructor can choose whether or not to assign it.

Arrays are storage structures common to most programming languages. Consequently, we have included the topic of arrays in Part 1. On the other hand, Java arrays are objects. The book is structured so that arrays (Chapter 7) can be covered at the end of Part 1, or delayed until after Chapter 9, *Objects and Classes I: Encapsulation, Strings, and Things*. Chapter 7 includes a discussion of two-dimensional arrays. These sections can be postponed without loss of continuity.

Simple data structures (stacks, queues, and linked lists) and the Java Collections Framework are covered at the end of Part 3 because the implementation of data structures is heavily dependent on the object-oriented paradigm.

Chapter Dependency Chart

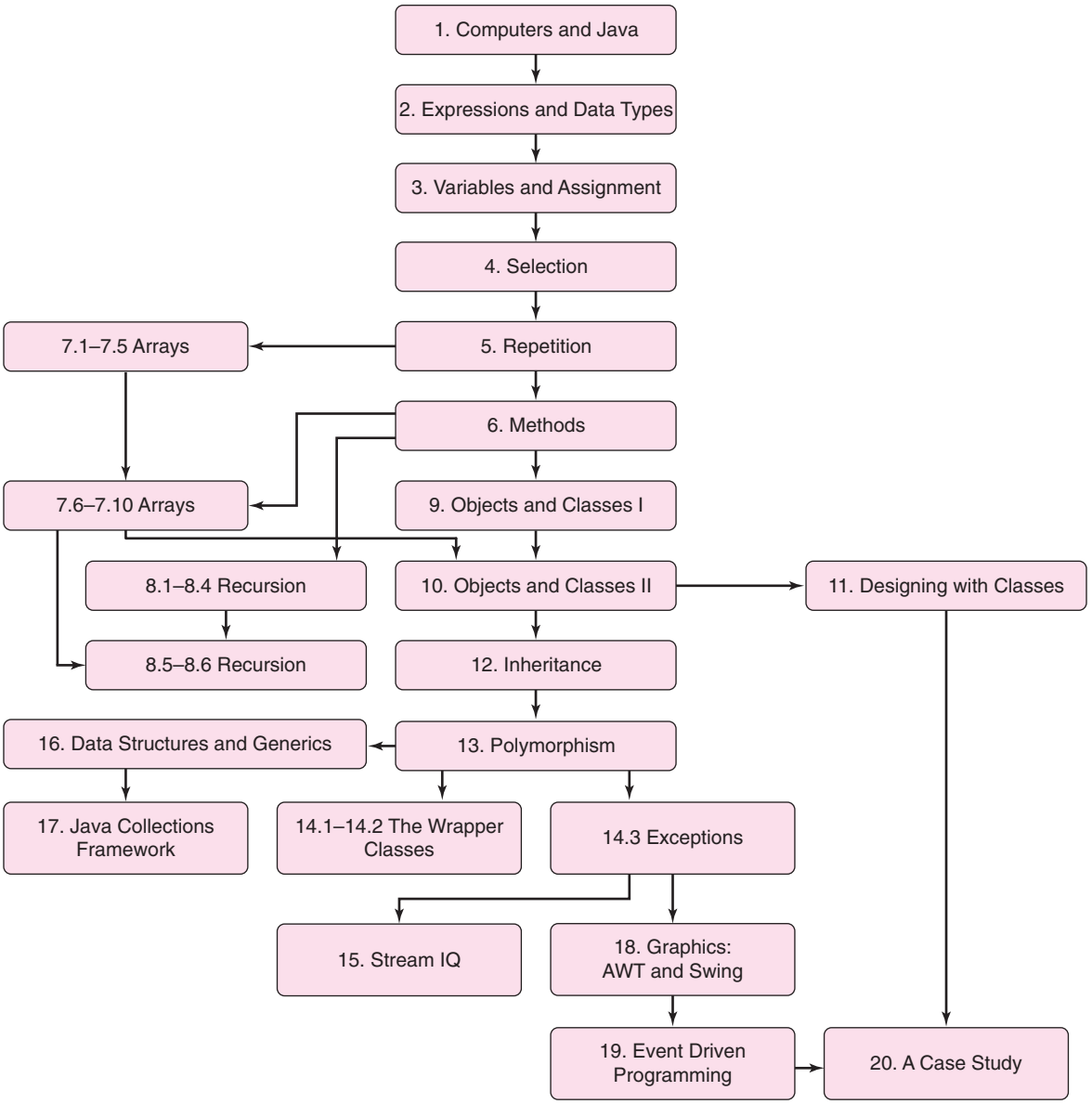
The following chart gives general chapter prerequisites. The chart can be used to configure many different types of courses. Although Chapters 1 through 6 are shown as prerequisite to Chapter 9, for those instructors eager to start with objects, a course might begin with Chapters 1–3, skip to 9, and cover the material in 4–6 as needed.

Online Resources

Online resources to accompany *Java Programming* are available on the text's website at www.mhhe.com/bravaco. Some of those resources include:

- Code and data for all program examples in the text
- Lecture PowerPoint slides
- An image library of all line art in the text
- An instructor's manual containing solutions to exercises

To access these resources, contact your McGraw-Hill representative.



TO THE STUDENT

You are about to study Java, a popular object-oriented programming language. There are many reasons why you may be studying Java:

- **Knowledge of Java and computer programming is required in your discipline (business, information technology, science, etc.).**

  Programming is a useful tool. Even if you do not become a programmer yourself, this text will provide you with an appreciation for what a programmer does. Long after you have forgotten the details in this book, the principles that you have learned will allow you to communicate better with programmers.

- **You hope to secure an interesting job.**

  Proficiency in Java is a marketable skill. Many interactive websites are written using Java. There is much to learn and Java's learning curve is steep, but greater proficiency comes with experience.
- **You are beginning a college major in computer science.**

  Unlike introductory courses in other sciences such as chemistry and physics, a first course in computer science is generally *not* an overview of the discipline but an intense introduction to programming and the *tools* of the discipline. While there are breadth-first courses that provide an overview of computer science, these courses are rare, and most computer science programs have retained the tradition of teaching programming first.

Java may very well be the first of many programming languages that you will learn. A good first language is one with a rich set of features that enables you to learn other languages quickly. A good first language is one powerful enough to implement sophisticated algorithms without tedious effort. A good language gives you enough power to easily implement an abstract concept.

There is no best first language, but there are many good ones such as Scheme, C, C++, C#, Visual Basic, Python, and of course, Java. Each language has its fans as well as its detractors. Java, like any programming language, has its strengths and weaknesses as a first language.

Strengths:

- Internet friendly
- Platform independent
- Reliable
- Secure
- Sophisticated GUI and event-driven paradigm
- Designed from the ground up as an object-oriented language
- Widely used
- Has huge collection of object libraries allowing fast, efficient reuse of code

Weaknesses:

- Huge collection of object libraries is intimidating to beginners.
- Steep learning curve, especially for GUI and event-driven models.
- Slow execution relative to standard compiled languages.

There is no perfect choice, but Java is certainly a good one. Thousands of people consider Java their "native" programming language, and Java will not likely disappear soon from industry or the classroom. Java is an excellent first language.

The only way to become fluent in Java is to write programs. You can and should listen to lectures; you can and should read the text. And, unquestionably, you must do the exercises. With practice and perseverance, you can become a skilled and successful programmer and have a bit of fun along the way. Enjoy your journey.

Electronic Textbook Option

This text is offered through CourseSmart for both instructors and students. CourseSmart is an online resource where students can purchase the complete text online at almost half the cost of a traditional text. Purchasing the eTextbook allows students to take advantage of CourseSmart's web tools for learning, which include full text search, notes and

highlighting, and email tools for sharing notes between classmates. To learn more about CourseSmart options, contact your sales representative or visit www.CourseSmart.com.

ACKNOWLEDGMENTS

Many people have contributed to the development of this book. We owe a debt of gratitude to our reviewers, who graciously gave of their time and expertise:

Suzanne Balik, North Carolina State University
Julia I. Couto, Georgia Gwinnet College
Jeanne Douglas, University of Vermont
William E. Duncan, Louisiana State University
H. E. Dunsmore, Purdue University
Joseph D. Hurley, Texas A & M University
Dennis Kellermeier, Wright State University
Lorrie Lehman, University of North Carolina, Charlotte
Kathy Liszka, University of Akron
Mark Llewellyn, University of Central Florida
Hunter Lloyd, Montana State University
Blayne E. Mayfield, Oklahoma State University
Robert J. McGlinn, Southern Illinois University, Carbondale
Rodrigo A. Obando, Columbus State University
Kevin O'Gorman, California Polytechnic Institute of Technology
Rayno D. Niemi, Rochester Institute of Technology
Juan Pavón, Facultad de Informática
Cyndi Rader, Colorado School of Mines
Michael D. Scott, University of Texas at Austin
Harish Sethu, Drexel University
Monica Sweat, Georgia Institute of Technology
Bahram Zartoshty, California State University, Northridge

We also wish to thank the members of the academic administration at Stonehill College for their encouragement and support, especially Provost and Academic Vice President Katie Conboy, Dean Karen Talentino, and Dean Joseph Favazza.

Colleagues, friends, and students who helped us along our way include Ryan Amari, Tanya Berger-Wolf, Jennifer Burge, Kathy Conroy, Robert Dugan, Matthew Fuller, Thomas Gariepy, Michael Haney, Andrew Harmon, Matthew Hinds, Antonio "Thumbs" Martinez, Nan Mulford, Elizabeth Patterson, Annemarie Ryan, Bonnie Troupe, and Thomas Wall.

Our gratitude goes to our students at Stonehill College and to the participants in our NSF Java workshops. You have contributed to this book in ways great and small.

Our editorial, production, and marketing staff helped this book take shape and we thank them all: Alan Apt, Carrie Burger, Kevin Campbell, Bonnie Coakley, Edwin Durbin, Tammy Juran, Melissa Leick, Rebecca Olson, Curt Reynolds, Brenda Rolwes, Michael Ryder, Raghu Srinivasan, and most especially Lora Kalb-Neyens, who patiently guided us throughout the creation of this book.

Lastly, we thank our families, Kathryn Kalinak and Emily Bravaco, and Andrea, Zosh, Yair, and Yona Simonson, for their love, their encouragement, and their endless patience without which this book would not have been possible.