Through the power of practice and immediate personalized feedback, MyProgrammingLab improves your performance.

Learn more at www.myprogramminglab.com
Preface for Instructors

Welcome to the sixth edition of *Java: An Introduction to Problem Solving & Programming*. This book is designed for a first course in programming and computer science. It covers programming techniques, as well as the basics of the Java programming language. It is suitable for courses as short as one quarter or as long as a full academic year. No previous programming experience is required, nor is any mathematics, other than a little high school algebra. The book can also be used for a course designed to teach Java to students who have already had another programming course, in which case the first few chapters can be assigned as outside reading.

Changes in This Edition

The following list highlights how this sixth edition differs from the fifth edition:

- Updates have been made for Java version 7, including strings in *switch* statements and the use of type inference in generic instance creation.
- Additional case studies including unit testing, use of the *Comparable* interface, processing comma-separated value files, and others.
- Chapter 5 now begins with a simpler class to more gradually introduce how classes are constructed.
- Chapter 8 has been reorganized with a greater emphasis and more examples on polymorphism and interfaces.
- Chapter 2 describes how to create a Swing application using the *JFrame* class so thereafter students have the option of implementing graphics in applets or in an application.
- Chapter 12 includes an overview of the Java Collections Framework and examples using the *HashMap* and *HashSet* classes.
- A description of *System.out.printf* has been added to Chapter 2.
- A description of *Math.random* has been added to Chapter 6.
- Twenty new programming projects have been added.
- New VideoNotes added throughout the text to enhance student understanding of programming concepts and techniques.
Latest Java Coverage

All of the code in this book has been tested using a pre-release version of Oracle’s Java SE Development Kit (JDK), version 7.0. Any imported classes are standard and in the Java Class Library that is part of Java. No additional classes or specialized libraries are needed.

Flexibility

If you are an instructor, this book adapts to the way you teach, rather than making you adapt to the book. It does not tightly prescribe the sequence in which your course must cover topics. You can easily change the order in which you teach many chapters and sections. The particulars involved in rearranging material are explained in the dependency chart that follows this preface and in more detail in the “Prerequisites” section at the start of each chapter.

Early Graphics

Graphics supplement sections end each of the first ten chapters. This gives you the option of covering graphics and GUI programming from the start of your course. The graphics supplement sections emphasize applets but also cover GUIs built using the JFrame class. Any time after Chapter 8, you can move on to the main chapters on GUI programming (Chapters 13 through 15), which are now on the Web. Alternatively, you can continue through Chapter 10 with a mix of graphics and more traditional programming. Instructors who prefer to postpone the coverage of graphics can postpone or skip the graphics supplement sections.

Coverage of Problem-Solving and Programming Techniques

This book is designed to teach students basic problem-solving and programming techniques and is not simply a book about Java syntax. It contains numerous case studies, programming examples, and programming tips. Additionally, many sections explain important problem-solving and programming techniques, such as loop design techniques, debugging techniques, style techniques, abstract data types, and basic object-oriented programming techniques, including UML, event-driven programming, and generic programming using type parameters.

Early Introduction to Classes

Any course that really teaches Java must teach classes early, since everything in Java involves classes. A Java program is a class. The data type for strings of characters is a class. Even the behavior of the equals operator (==) depends on whether it is comparing objects from classes or simpler data items. Classes cannot be avoided, except by means of absurdly long and complicated “magic formulas.” This book introduces classes fairly early. Some exposure to using classes is given in Chapters 1 and 2. Chapter 5 covers how to define classes. All
of the basic information about classes, including inheritance, is presented by the end of Chapter 8 (even if you omit Chapter 7). However, some topics regarding classes, including inheritance, can be postponed until later in the course.

Although this book introduces classes early, it does not neglect traditional programming techniques, such as top-down design and loop design techniques. These older topics may no longer be glamorous, but they are information that all beginning students need.

**Generic Programming**

Students are introduced to type parameters when they cover lists in Chapter 12. The class `ArrayList` is presented as an example of how to use a class that has a type parameter. Students are then shown how to define their own classes that include a type parameter.

**Language Details and Sample Code**

This book teaches programming technique, rather than simply the Java language. However, neither students nor instructors would be satisfied with an introductory programming course that did not also teach the programming language. Until you calm students’ fears about language details, it is often impossible to focus their attention on bigger issues. For this reason, the book gives complete explanations of Java language features and lots of sample code. Programs are presented in their entirety, along with sample input and output. In many cases, in addition to the complete examples in the text, extra complete examples are available over the Internet.

**Self-Test Questions**

Self-test questions are spread throughout each chapter. These questions have a wide range of difficulty levels. Some require only a one-word answer, whereas others require the reader to write an entire, nontrivial program. Complete answers for all the self-test questions, including those requiring full programs, are given at the end of each chapter.

**Exercises and Programming Projects**

 Completely new exercises appear at the end of each chapter. Since only you, and not your students, will have access to their answers, these exercises are suitable for homework. Some could be expanded into programming projects. However, each chapter also contains other programming projects, several of which are new to this edition.

**Support Material**

The following support materials are available on the Internet at www.pearsonhighered.com/irc:
For instructors only:
- Solutions to most exercises and programming projects
- PowerPoint slides
- Lab Manual with associated code.

Instructors should click on the registration link and follow instructions to receive a password. If you encounter any problems, please contact your local Pearson Sales Representative. For the name and number of your sales representative, go to pearsonhighered.com/replocator.

For students:
- Source code for programs in the book and for extra examples
- Student lab manual
- VideoNotes: video solutions to programming examples and exercises.

Visit www.pearsonhighered.com/savitch to access the student resources.

Online Practice and Assessment with MyProgrammingLab

MyProgrammingLab helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyProgrammingLab improves the programming competence of beginning students who often struggle with the basic concepts and paradigms of popular high-level programming languages.

A self-study and homework tool, a MyProgrammingLab course consists of hundreds of small practice problems organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of their code submissions and offers targeted hints that enable students to figure out what went wrong—and why. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code inputted by students for review.

MyProgrammingLab is offered to users of this book in partnership with Turing’s Craft, the makers of the CodeLab interactive programming exercise system. For a full demonstration, to see feedback from instructors and students, or to get started using MyProgrammingLab in your course, visit www.myprogramminglab.com.

VideoNotes

VideoNotes are Pearson’s new visual tool designed for teaching students key programming concepts and techniques. These short step-by-step videos demonstrate how to solve problems from design through coding. VideoNotes allow for self-placed instruction with easy navigation including the ability to select, play, rewind, fast-forward, and stop within each VideoNote exercise.

Margin icons in your textbook let you know when a VideoNote video is available for a particular concept or homework problem.

**Integrated Development Environment Resource Kits**

Professors who adopt this text can order it for students with a kit containing seven popular Java IDEs (the most recent JDK from Oracle, Eclipse, NetBeans, jGRASP, DrJava, BlueJ, and TextPad). The kit also includes access to a website containing written and video tutorials for getting started in each IDE. For ordering information, please contact your campus Pearson Education representative or visit www.pearsonhighered.com.

**Contact Us**

Your comments, suggestions, questions, and corrections are always welcome. Please e-mail them to savitch.programming.java@gmail.com.
Preface for Students

This book is designed to teach you the Java programming language and, even more importantly, to teach you basic programming techniques. It requires no previous programming experience and no mathematics other than some simple high school algebra. However, to get the full benefit of the book, you should have Java available on your computer, so that you can practice with the examples and techniques given. The latest version of Java is preferable, but a version as early as 5 will do.

If You Have Programmed Before

You need no previous programming experience to use this book. It was designed for beginners. If you happen to have had experience with some other programming language, do not assume that Java is the same as the programming language(s) you are accustomed to using. All languages are different, and the differences, even if small, are large enough to give you problems. Browse the first four chapters, reading at least the Recap portions. By the time you reach Chapter 5, it would be best to read the entire chapter.

If you have programmed before in either C or C++, the transition to Java can be both comfortable and troublesome. At first glance, Java may seem almost the same as C or C++. However, Java is very different from these languages, and you need to be aware of the differences. Appendix 6 compares Java and C++ to help you see what the differences are.

Obtaining a Copy of Java

Appendix 1 provides links to sites for downloading Java compilers and programming environments. For beginners, we recommend Oracle's Java JDK for your Java compiler and related software and TextPad as a simple editor environment for writing Java code. When downloading the Java JDK, be sure to obtain the latest version available.

Support Materials for Students

- Source code for programs in the book and for extra examples
- Student lab manual
- VideoNotes: video solutions to programming examples and exercises.

Visit www.pearsonhighered.com/savitch to access the student resources.
Learning Aids

Each chapter contains several features to help you learn the material:

- The opening overview includes a brief table of contents, chapter objectives and prerequisites, and a paragraph or two about what you will study.
- Recaps concisely summarize major aspects of Java syntax and other important concepts.
- FAQs, or “frequently asked questions,” answer questions that other students have asked.
- Remembers highlight important ideas you should keep in mind.
- Programming Tips suggest ways to improve your programming skills.
- Gotchas identify potential mistakes you could make—and should avoid—while programming.
- Asides provide short commentaries on relevant issues.
- Self-Test Questions test your knowledge throughout, with answers given at the end of each chapter. One of the best ways to practice what you are learning is to do the self-test questions before you look at the answers.
- A summary of important concepts appears at the end of each chapter.

Online Practice with MyProgrammingLab

A self-study and practice tool, a MyProgrammingLab course consists of hundreds of small practice problems organized around the structure of this textbook. The system automatically detects errors in the logic and syntax of your code submissions and offers targeted hints that enable you to figure out what went wrong—and why. Visit www.myprogramminglab.com for more information.

VideoNotes

These short step-by-step videos demonstrate how to solve problems from design through coding. VideoNotes allow for self-placed instruction with easy navigation including the ability to select, play, rewind, fast-forward, and stop within each VideoNote exercise. Margin icons in your textbook let you know when a VideoNote video is available for a particular concept or homework problem.

This Text Is Also a Reference Book

In addition to using this book as a textbook, you can and should use it as a reference. When you need to check a point that you have forgotten or that you hear mentioned by somebody but have not yet learned yourself, just look in the index. Many index entries give a page number for a “recap.” Turn to that page. It will contain a short, highlighted entry giving all the essential points...
on that topic. You can do this to check details of the Java language as well as
details on programming techniques.

Recap sections in every chapter give you a quick summary of the main
points in that chapter. Also, a summary of important concepts appears at the
end of each chapter. You can use these features to review the chapter or to
check details of the Java language.
Acknowledgments

We thank the many people who have made this sixth edition possible, including everyone who has contributed to the first five editions. We begin by recognizing and thanking the people involved in the development of this new edition. The comments and suggestions of the following reviewers were invaluable and are greatly appreciated. In alphabetical order, they are:

Asa Ben-Hur—Colorado State University
Joan Boone—University of North Carolina at Chapel Hill
Dennis Brylow—Temple University
Billie Goldstein—Temple University
Helen H. Hu—Westminster College
Tammy VanDeGrift—University of Portland

Many other reviewers took the time to read drafts of earlier editions of the book. Their advice continues to benefit this new edition. Thank you once again to:

Gerald Baumgartner—Louisiana State University
Jim Buffenbarger—Idaho State University
Robert P. Burton—Brigham Young University
Mary Elaine Califf—Illinois State University
Steve Cater—Kettering University
Martin Chelten—Moorpark Community College
Ashraful A. Chowdhury—Georgia Perimeter College
Ping-Chu Chu—Fayetteville State University
Michael Clancy—University of California, Berkeley
Tom Cortina—State University of New York at Stony Brook
Prasun Dewan—University of North Carolina
Laird Dornan—Sun Microsystems, Inc.
H. E. Dunsmore—Purdue University, Lafayette
Adel Elmaghraby—University of Louisville
Ed Gellenbeck—Central Washington University
Adrian German—Indiana University
Gobi Gopinath—Suffolk County Community College
Le Gruenwald—University of Oklahoma
Gopal Gupta—University of Texas, Dallas
Ricci Heishman—North Virginia Community College
Robert Herrmann—Sun Microsystems, Inc., Java Soft
Chris Hoffmann—University of Massachusetts, Amherst
ACKNOWLEDGMENTS

Robert Holloway—University of Wisconsin, Madison
Charles Hoot—Oklahoma City University
Lily Hou—Carnegie Mellon University
Richard A. Johnson—Missouri State University
Rob Kelly—State University of New York at Stony Brook
Michele Kleckner—Elon College
Stan Kwasny—Washington University
Anthony Larrain—Depaul University
Mike Litman—Western Illinois University
Y. Annie Liu—State University of New York at Stony Brook
Michael Long—California State University
Blayne Mayfield—Oklahoma State University
Drew McDermott—Yale University
Gerald H. Meyer—LaGuardia Community College
John Motil—California State University, Northridge
Michael Olan—Stockton State
Richard Ord—University of California, San Diego
James Roberts—Carnegie Mellon University
Alan Saleski—Loyola University Chicago
Dolly Samson—Hawaii Pacific University
Nan C. Schaller—Rochester Institute of Technology
Arijit Sengupta—Raj Sion College of Business, Wright State University
Ryan Shoemaker—Sun Microsystems, Inc.
Liuba Shrira—Brandeis University
Ken Slonneger—University of Iowa
Donald E. Smith—Rutgers University
Peter Spoerri—Fairfield University
Howard Straubing—Boston College
Navabi Tadayon—Arizona State University
Boyd Trolinger—Butte College
Tom Van Drunen—Wheaton College
Subramanian Vijayarangam—University of Massachusetts, Lowell
Stephen F. Weiss—University of North Carolina, Chapel Hill
Richard Whitehouse—Arizona State University
Michael Young—University of Oregon

Last but not least, we thank the many students in classes at the University of California, San Diego (UCSD), who were kind enough to help correct preliminary versions of this text, as well as the instructors who class-tested these drafts. In particular, we extend a special thanks to Carole McNamee of California State University, Sacramento, and to Paul Kube of UCSD. These student comments and the detailed feedback and class testing of earlier editions of the book were a tremendous help in shaping the final book.

W. S.
K. M.
Dependency Chart

This chart shows the prerequisites for the chapters in the book. If there is a line between two boxes, the material in the higher box should be covered before the material in the lower box. Minor variations to this chart are discussed in the “Prerequisites” section at the start of each chapter. These variations usually provide more, rather than less, flexibility than what is shown on the chart.

* Note that some sections of these chapters can be covered sooner. Those sections are given in this chart.
** These chapters contain sections that can be covered sooner. See the chapter’s “Prerequisites” section for full details.

Recaps
Summarize Java syntax and other important concepts.

Remembers
Highlight important ideas that students should keep in mind.

Features of This Text

Recap

Bytes and Memory Locations
A computer’s main memory is divided into numbered units called bytes. The number of a byte is called its address. Each byte can hold eight binary digits, or bits, of a piece of data that is too large to fit into a single byte.

RECAP

Syntactic Variables
When you see something in this book like Type, Variable_1, or Variable_2 used to describe Java syntax, these words do not literally appear in your Java code. They are syntactic variables, which are a kind of blank that you fill in with the category that they describe. For example, Type can be replaced by int, double, char, or any other type name. Variable_1 and Variable_2 can each be replaced by any variable name.

Programming Tips
Give students helpful advice about programming in Java.

PROGRAMMING TIP

Initialize Variables
A variable that has been declared, but that has not yet been given a value by an assignment statement (or in some other way), is said to be uninitialized. If the variable is a variable of a class type, it literally has no value. If the variable has a primitive type, it likely has some default value. However, your program will be clearer if you explicitly give the variable a value, even if you are simply assigning the default value. (The exact details on default values have been known to change and should not be counted on.)

One easy way to ensure that you do not have an uninitialized variable is to initialize it within the declaration. Simply combine the declaration and an assignment statement, as in the following examples:

```java
int count = 0;
double taxRate = 0.075;
char grade = 'A';
int balance = 1000, newBalance;
```

Note that you can initialize some variables and not initialize others in a declaration. Sometimes the compiler may complain that you have failed to initialize a variable. In most cases, that will indeed be true. Occasionally, though, the compiler is mistaken in giving this advice. However, the compiler will not compile your program until you convince it that the variable in question is initialized. To make the compiler happy, initialize the variable when you declare it, even if the variable will be given another value before it is used for anything. In such cases, you cannot argue with the compiler.

Gotchas
Identify potential mistakes in programming that students might make and should avoid.

GOTCHA

Hidden Errors
Just because your program compiles and runs without any errors and even produces reasonable-looking output does not mean that your program is correct. You should always run your program with some test data that gives predictable output. To do this, choose some data for which you can compute the correct results, either by using pencil and paper, by looking up the answer, or by some other means. Even this testing does not guarantee that your program is correct, but the more testing you do, the more confidence you can have in your program.

FAQs
Provide students answers to frequently asked questions within the context of the chapter.

FAQ

Why just 0s and 1s?
Computers use 0s and 1s because it is easy to make an electrical device that has only two stable states. However, when you are programming, you normally need not be concerned about the encoding of data as 0s and 1s. You can program as if the computer directly stored numbers, letters, or strings of characters in memory.

There is nothing special about calling the states zero and one. We could just as well use any two names, such as A and B or true and false. The important thing is that the underlying physical device has two stable states, such as on and off or high voltage and low voltage. Calling these two states zero and one is simply a convention, but it’s one that is almost universally followed.
So far we've tested our programs by running them, typing in some input, and visually checking the results to see if the output is what we expected. This is fine for small programs but is generally insufficient for large programs. In a large program there are usually so many combinations of interacting inputs that it would take too much time to manually verify the correct result for all inputs. Additionally, it is possible that code changes result in unintended side effects. For example, a fix for one error might introduce a different error. One way to attack this problem is to write unit tests.

Unit testing is a methodology in which the programmer tests the correctness of individual units of code. A unit is often a method but it could be a class or other group of code. The collection of unit tests becomes the test suite. Each test is generally automated so that human input is not required. Automation is important because it is desirable to have tests that run often and quickly. This makes it possible to run the tests repeatedly, perhaps once a day or every time code is changed, to make sure that everything is still working. The process of running tests repeatedly is called regression testing.

Let's start with a simple test case for the Species class in Listing 5.19. Our first test might be to verify that the name, initial population, and growth rate is correctly set in the setSpecies method. We can accomplish this by creating
Programming Examples
Provide more examples of Java programs that solve specific problems.

PROGRAMMING EXAMPLE
Nested Loops

The body of a loop can contain any sort of statements. In particular, you can have a loop statement within the body of a larger loop statement. For example, the program in Listing 4.4 uses a while loop to compute the average of a list of nonnegative scores. The program asks the user to enter all the scores followed by a negative sentinel value to mark the end of the data. This while loop is placed inside a do-while loop so that the user can repeat the entire process for another exam, and another, until the user wishes to end the program.

Self-Test Questions
Provide students with the opportunity to practice skills learned in the chapter. Answers at the end of each chapter give immediate feedback.

SELF-TEST QUESTIONS

28. Given the class Species as defined in Listing 5.19, why does the following program cause an error message?

```java
public class SpeciesEqualsDemo
{
    public static void main(String[] args)
    {
        Species s1, s2; s1.setSpecies("Klingon ox", 10, 15);
        s2.setSpecies("Klingon ox", 10, 15);
        if (s1 == s2)
            System.out.println("Match with ==.");
        else
            System.out.println("Do Notmatchwith ==.");
    }
}
```

29. After correcting the program in the previous question, what output does the program produce?

30. What is the biggest difference between a parameter of a primitive type and a parameter of a class type?

31. Given the class Species, as defined in Listing 5.19, and the class

Asides
Give short commentary on relevant topics.

ASIDE Use of the Terms Parameter and Argument

Our use of the terms parameter and argument is consistent with common usage. We use parameter to describe the definition of the data type and variable inside the header of a method and argument to describe items passed into a method when it is invoked. However, people often use these terms interchangeably. Some people use the term parameter both for what we call a formal parameter and for what we call an argument. Other people use the term argument both for what we call a formal parameter and for what we call an argument. When you see the term parameter or argument in other books, you must figure out its exact meaning from the context.
Brief Table of Contents

Chapter 1  Introduction to Computers and Java  1
Chapter 2  Basic Computation  47
Chapter 3  Flow of Control: Branching  137
Chapter 4  Flow of Control: Loops  195
Chapter 5  Defining Classes and Methods  261
Chapter 6  More About Objects and Methods  373
Chapter 7  Arrays  479
Chapter 8  Inheritance, Polymorphism and Inheritance  575
Chapter 9  Exception Handling  657
Chapter 10  Streams and File I/O  725
Chapter 11  Recursion  799
Chapter 12  Dynamic Data Structures and Generics  847
Appendices
  1 Getting Java  917
  2 Running Applets  918

BRIEF TABLE OF CONTENTS

3 Protected and Package Modifiers 920
4 The DecimalFormat Class 921
5 javadoc 925
6 Differences between C++ and Java 928
7 Unicode Character Codes 932

Index 933

The following chapters and appendices, along with an index to their contents, are on the book's website:

Chapter 13 Window Interfaces Using Swing

Chapter 14 Applets and HTML

Chapter 15 More Swing

Appendices
8 The Iterator Interface
9 Cloning
**Table of Contents**

**Chapter 1  Introduction to Computers and Java  1**

1.1 COMPUTER BASICS  2  
Hardware and Memory  3  
Programs  6  
Programming Languages, Compilers, and Interpreters  7  
Java Bytecode  9  
Class Loader  11  

1.2 A SIP OF JAVA  12  
History of the Java Language  12  
Applications and Applets  13  
A First Java Application Program  14  
Writing, Compiling, and Running a Java Program  19  

1.3 PROGRAMMING BASICS  21  
Object-Oriented Programming  21  
Algorithms  25  
Testing and Debugging  27  
Software Reuse  28  

1.4 GRAPHICS SUPPLEMENT  30  
A Sample Graphics Applet  30  
Size and Position of Figures  32  
Drawing Ovals and Circles  34  
Drawing Arcs  35  
Running an Applet  37  

**Chapter 2  Basic Computation  47**  

2.1 VARIABLES AND EXPRESSIONS  48  
Variables  49  
Data Types  51  

# Table of Contents

Java Identifiers 53  
Assignment Statements 55  
Simple Input 58  
Simple Screen Output 60  
Constants 60  
Named Constants 62  
Assignment Compatibilities 63  
Type Casting 65  
Arithmetic Operators 68  
Parentheses and Precedence Rules 71  
Specialized Assignment Operators 72  
*Case Study*: Vending Machine Change 74  
Increment and Decrement Operators 79  
More About the Increment and Decrement Operators 80

## 2.2 The Class String 81

String Constants and Variables 81  
Concatenation of Strings 82  
String Methods 83  
String Processing 85  
Escape Characters 88  
The Unicode Character Set 89

## 2.3 Keyboard and Screen I/O 91

Screen Output 91  
Keyboard Input 94  
Other Input Delimiters *(Optional)* 99  
Formatted Output with printf *(Optional)* 101

## 2.4 Documentation and Style 103

Meaningful Variable Names 103  
Comments 104  
Indentation 107  
Using Named Constants 107

## 2.5 Graphics Supplement 109

Style Rules Applied to a Graphics Applet 110  
Creating a Java GUI Application with the JFrame Class 110  
Introducing the Class JOptionPane 113  
Reading Input as Other Numeric Types 123

## Chapter 3  Flow of Control: Branching  137

### 3.1 THE if-else STATEMENT  138
- The Basic if-else Statement  139
- Boolean Expressions  145
- Comparing Strings  150
- Nested if-else Statements  155
- Multibranch if-else Statements  157
  - Programming Example: Assigning Letter Grades  159
  - Case Study: Body Mass Index  162
  - The Conditional Operator (Optional)  165
  - The exit Method  165

### 3.2 THE TYPE boolean  166
- Boolean Variables  167
- Precedence Rules  168
- Input and Output of Boolean Values  171

### 3.3 THE switch STATEMENT  173
- Enumerations  179

### 3.4 GRAPHICS SUPPLEMENT  180
- Specifying a Drawing Color  181
- A Dialog Box for a Yes-or-No Question  184

## Chapter 4  Flow of Control: Loops  195

### 4.1 JAVA LOOP STATEMENTS  196
- The while Statement  197
- The do-while Statement  200
  - Programming Example: Bug Infestation  205
  - Programming Example: Nested Loops  211
- The for Statement  213
- Declaring Variables within a for Statement  219
- Using a Comma in a for Statement (Optional)  220
- The for-each Statement  222

---

4.2 Programming with Loops  222
The Loop Body  223
Initializing Statements  224
Controlling the Number of Loop Iterations  225
Case Study: Using a Boolean Variable to End a Loop  227
Programming Example: Spending Spree  229
The break Statement and continue Statement in Loops
(Optional)  232
Loop Bugs  235
Tracing Variables  237
Assertion Checks  239

4.3 Graphics Supplement  241
Programming Example: A Multiface Applet  241
The drawstring Method  247

Chapter 5  Defining Classes and Methods  261

5.1 Class and Method Definitions  263
Class Files and Separate Compilation  265
Programming Example: Implementing a Dog Class  265
Instance Variables  266
Methods  269
Defining void Methods  272
Defining Methods That Return a Value  273
Programming Example: First Try at Implementing a Species Class  278
The Keyword this  282
Local Variables  284
Blocks  286
Parameters of a Primitive Type  287

5.2 Information Hiding and Encapsulation  293
Information Hiding  294
Precondition and Postcondition Comments  294
The public and private Modifiers  296
Programming Example: A Demonstration of Why Instance Variables Should Be Private  299
Programming Example: Another Implementation of a Class of Rectangles  300
Accessor Methods and Mutator Methods  302

Programming Example: A Purchase Class  306
Methods Calling Methods  310
Encapsulation  316
Automatic Documentation with javadoc  319
UML Class Diagrams  320

5.3 OBJECTS AND REFERENCES  321
Variables of a Class Type  322
Defining an equals Method for a Class  327
Programming Example: A Species Class  331
Boolean-Valued Methods  334
Case Study: Unit Testing  336
Parameters of a Class Type  338
Programming Example: Class-Type Parameters Versus
  Primitive-Type Parameters  342

5.4 GRAPHICS SUPPLEMENT  346
The Graphics Class  346
Programming Example: Multiple Faces, but with a Helping
  Method  348
The init Method  352
Adding Labels to an Applet  352

Chapter 6  More About Objects and Methods  373

6.1 CONSTRUCTORS  375
Defining Constructors  375
Calling Methods from Constructors  384
Calling a Constructor from Other Constructors (Optional)  387

6.2 STATIC VARIABLES AND STATIC METHODS  389
Static Variables  389
Static Methods  390
Dividing the Task of a main Method into Subtasks  397
Adding a main Method to a Class  398
The Math Class  400
Wrapper Classes  403

6.3 WRITING METHODS  409
Case Study: Formatting Output  409
Decomposition  415

Arguments for the Method main 507
Array Assignment and Equality 508
Methods That Return Arrays 511

7.3 PROGRAMMING WITH ARRAYS AND CLASSES 515
Programming Example: A Specialized List Class 515
Partially Filled Arrays 523

7.4 SORTING AND SEARCHING ARRAYS 525
Selection Sort 525
Other Sorting Algorithms 529
Searching an Array 531

7.5 MULTIDIMENSIONAL ARRAYS 532
Multidimensional-Array Basics 533
Multidimensional-Array Parameters and Returned Values 536
Java's Representation of Multidimensional Arrays 539
Ragged Arrays (Optional) 540
Programming Example: Employee Time Records 542

7.6 GRAPHICS SUPPLEMENT 548
Text Areas and Text Fields 548
Programming Example: A Question-and-Answer Applet 548
The Classes JTextArea and JTextField 551
Drawing Polygons 553

Chapter 8 Inheritance, Polymorphism and Interfaces 575

8.1 INHERITANCE BASICS 576
Derived Classes 578
Overriding Method Definitions 582
Overriding Versus Overloading 583
The final Modifier 583
Private Instance Variables and Private Methods of a Base Class 584
UML Inheritance Diagrams 586

8.2 PROGRAMMING WITH INHERITANCE 589
Constructors in Derived Classes 589
The this Method—Again 591
Calling an Overridden Method 591

Programming Example: A Derived Class of a Derived Class  592
Another Way to Define the equals Methods in Undergraduate  597
Type Compatibility  597
The Class Object  602
A Better equals Method  604

8.3 POLYMORPHISM  606
Dynamic Binding and Inheritance  606
Dynamic Binding with toString  609

8.4 INTERFACES AND ABSTRACT CLASSES  611
Class Interfaces  611
Java Interfaces  612
Implementing an Interface  613
An Interface as a Type  615
Extending an Interface  618
Case Study: Character Graphics  619
Case Study: The Comparable Interface  632
Abstract Classes  636

8.5 GRAPHICS SUPPLEMENT  638
The Class JApplet  639
The Class JFrame  639
Window Events and Window Listeners  642
The ActionListener Interface  644
What to Do Next  644

Chapter 9 Exception Handling  657

9.1 BASIC EXCEPTION HANDLING  658
Exceptions in Java  659
Predefined Exception Classes  669

9.2 DEFINING YOUR OWN EXCEPTION CLASSES  671

9.3 MORE ABOUT EXCEPTION CLASSES  681
Declaring Exceptions (Passing the Buck)  681
Kinds of Exceptions  684
Errors  686
Multiple Throws and Catches  687
The finally Block  693

Table of Contents

Rethrowing an Exception (Optional) 694
Case Study: A Line-Oriented Calculator 695

9.4 Graphics Supplement 707
Exceptions in GUIs 707
Programming Example: A JFrame GUI Using Exceptions 707

Chapter 10 Streams and File I/O 725

10.1 An Overview of Streams and File I/O 727
The Concept of a Stream 727
Why Use Files for I/O? 728
Text Files and Binary Files 728

10.2 Text-File I/O 730
Creating a Text File 730
Appending to a Text File 736
Reading from a Text File 738

10.3 Techniques for Any File 741
The Class File 741
Programming Example: Reading a File Name from the Keyboard 741
Using Path Names 743
Methods of the Class File 744
Defining a Method to Open a Stream 746
Case Study: Processing a Comma-Separated Values File 748

10.4 Basic Binary-File I/O 751
Creating a Binary File 751
Writing Primitive Values to a Binary File 753
Writing Strings to a Binary File 756
Some Details About writeUTF 757
Reading from a Binary File 759
The Class EOFException 764
Programming Example: Processing a File of Binary Data 766

10.5 Binary-File I/O with Objects and Arrays 771
Binary-File I/O with Objects of a Class 771
Some Details of Serialization 775
Array Objects in Binary Files 776

<table>
<thead>
<tr>
<th>Chapter 11</th>
<th>Recursion</th>
<th>799</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>11.1 THE BASICS OF RECURSION</strong></td>
<td>800</td>
<td></td>
</tr>
<tr>
<td>Case Study: Digits to Words</td>
<td>803</td>
<td></td>
</tr>
<tr>
<td>How Recursion Works</td>
<td>808</td>
<td></td>
</tr>
<tr>
<td>Infinite Recursion</td>
<td>812</td>
<td></td>
</tr>
<tr>
<td>Recursive Methods Versus Iterative Methods</td>
<td>814</td>
<td></td>
</tr>
<tr>
<td>Recursive Methods That Return a Value</td>
<td>816</td>
<td></td>
</tr>
<tr>
<td><strong>11.2 PROGRAMMING WITH RECURSION</strong></td>
<td>820</td>
<td></td>
</tr>
<tr>
<td>Programming Example: Insisting That User Input Be Correct</td>
<td>820</td>
<td></td>
</tr>
<tr>
<td>Case Study: Binary Search</td>
<td>822</td>
<td></td>
</tr>
<tr>
<td>Programming Example: Merge Sort—A Recursive Sorting Method</td>
<td>830</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 12</th>
<th>Dynamic Data Structures and Generics</th>
<th>847</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>12.1 ARRAY-BASED DATA STRUCTURES</strong></td>
<td>849</td>
<td></td>
</tr>
<tr>
<td>The Class ArrayList</td>
<td>850</td>
<td></td>
</tr>
<tr>
<td>Creating an Instance of ArrayList</td>
<td>850</td>
<td></td>
</tr>
<tr>
<td>Using the Methods of ArrayList</td>
<td>852</td>
<td></td>
</tr>
<tr>
<td>Programming Example: A To-Do List</td>
<td>856</td>
<td></td>
</tr>
<tr>
<td>Parameterized Classes and Generic Data Types</td>
<td>859</td>
<td></td>
</tr>
<tr>
<td><strong>12.2 THE JAVA COLLECTIONS FRAMEWORK</strong></td>
<td>859</td>
<td></td>
</tr>
<tr>
<td>The Collection Interface</td>
<td>860</td>
<td></td>
</tr>
<tr>
<td>The Class HashSet</td>
<td>861</td>
<td></td>
</tr>
<tr>
<td>The Map Interface</td>
<td>862</td>
<td></td>
</tr>
<tr>
<td>The Class HashMap</td>
<td>862</td>
<td></td>
</tr>
<tr>
<td><strong>12.3 LINKED DATA STRUCTURES</strong></td>
<td>865</td>
<td></td>
</tr>
<tr>
<td>The Class LinkedList</td>
<td>865</td>
<td></td>
</tr>
<tr>
<td>Linked Lists</td>
<td>866</td>
<td></td>
</tr>
<tr>
<td>Implementing the Operations of a Linked List</td>
<td>869</td>
<td></td>
</tr>
<tr>
<td>A Privacy Leak</td>
<td>876</td>
<td></td>
</tr>
<tr>
<td>Inner Classes</td>
<td>877</td>
<td></td>
</tr>
<tr>
<td>Node Inner Classes</td>
<td>878</td>
<td></td>
</tr>
<tr>
<td>Iterators</td>
<td>878</td>
<td></td>
</tr>
</tbody>
</table>
The Java Iterator Interface 890
Exception Handling with Linked Lists 890
Variations on a Linked List 892
Other Linked Data Structures 894

12.4 GENERICS 895
The Basics 895
Programming Example: A Generic Linked List 898

APPENDICES
1 Getting Java 917
2 Running Applets 918
3 Protected and Package Modifiers 920
4 The DecimalFormat Class 921
   Other Pattern Symbols 922
5 Javadoc 925
   Commenting Classes for Use within javadoc 925
   Running javadoc 926
6 Differences Between C++ and Java 928
   Primitive Types 928
   Strings 928
   Flow of Control 928
   Testing for Equality 929
   main Method (Function) and Other Methods 929
   Files and Including Files 929
   Class and Method (Function) Definitions 930
   No Pointer Types in Java 930
   Method (Function) Parameters 930
   Arrays 930
   Garbage Collection 931
   Other Comparisons 931
7 Unicode Character Codes 932

INDEX 933