Preface

"Live in fragments no longer, only connect."
—Edgar Morgan Foster

Welcome to Java and *Java How to Program, Sixth Edition*! At Deitel & Associates, we write computer science textbooks and professional books. This book was a joy to create. To start, we put the fifth edition of *Java How to Program* under the microscope:

- We audited the presentation against the most recent ACM/IEEE curriculum recommendations and the Computer Science Advanced Placement Examination.
- All of the chapters have been significantly updated and upgraded.
- We changed to an early classes and objects pedagogy. Now students build their first reusable classes starting in Chapter 3.
- All of the GUI and graphics in the early chapters has been replaced by carefully paced optional sections in Chapters 3–10 with two special exercise sections in Chapters 11 and 12. Instructors have a broad choice of the amount of GUI and graphics to cover—from none, to a 10-section introductory sequence, to a deep treatment in Chapters 11, 12 and 22.
- We updated our object-oriented presentation to use the latest version of the *UML (Unified Modeling Language)—UML™ 2.0*—the industry-standard graphical language for modeling object-oriented systems.
- We replaced the optional elevator simulator case study from the previous edition with a new optional OOD/UML automated teller machine (ATM) case study in Chapters 1–8 and 10. The new case study is much simpler and more attractive for first and second programming courses.
- Several multi-section object-oriented programming case studies have been added.
- We incorporated key new features of Sun Microsystems' latest release of Java—the *Java 2 Platform, Standard Edition version 5.0* (*J2SE 5.0*).
- The design of the book has been completely revised. This new design uses color, fonts and various design elements to enhance a student's learning experience.

All of this has been carefully scrutinized by a team of 37 distinguished academic and industry reviewers.

We believe that this book and its support materials have everything instructors and students need for an informative, interesting, challenging and entertaining Java educational experience. In this Preface, we overview various conventions used in the book, such as syntax coloring the code examples, "code washing" and code highlighting. We discuss the software bundled with the book as well as the comprehensive suite of educational materials that help instructors maximize their students' learning experience, including the *Instructor's*

Resource CD, PowerPoint® Slide lecture notes, lab manual, companion Web site, course management systems, SafariX (Pearson Education's WebBook publications) and more.

As you read this book, if you have questions, send an e-mail to `deitel@deitel.com`; we will respond promptly. Please visit our Web site, `www.deitel.com`, regularly and be sure to sign up for the free *Deitel® Buzz Online* e-mail newsletter at `www.deitel.com/newsletter/subscribe.html`. We use the Web site and the newsletter to keep our readers and industry clients informed of the latest news on Deitel publications and services. Please check the Web site occasionally for errata, updates regarding the Java software, free downloads and other resources.

Features in *Java How to Program, 6/e*

This new edition contains many new and enhanced features including:

Updated for the Java 2 Platform Standard Edition 5.0 (J2SE 5.0)

We updated entire text to reflect the latest release of J2SE 5.0. New topics include:

- obtaining formatted input with class `Scanner`
- displaying formatted output with the `System.out` object's `printf` method
- using enhanced `for` statements to process array elements and collections
- declaring methods with variable-length argument lists ("varargs")
- using `enum` classes that declare sets of constants
- importing the `static` members of one class for use in another
- converting primitive-type values to type-wrapper objects and vice versa using autoboxing and auto-unboxing, respectively
- using generics to create general models of methods and classes that can be declared once, but used with many different data types
- using the generics-enhanced data structures of the Collections API
- using the Concurrency API to implement multithreaded applications
- using JDBC `RowSet`s to access data in a database

In addition, we carefully audited the manuscript against the *Java Language Specification* (available at `java.sun.com/docs/books/jls/index.html`). The programs you create as you study this text should work with any J2SE 5.0 compatible Java platform.

[*Note:* Sun Microsystems recently renamed J2SE from the **Java 2 Platform, Standard Edition 1.5.0** to the **Java 2 Platform, Standard Edition 5.0**. However, Sun decided not to replace all occurrences of 1.5.0 with 5.0 in the online Java documentation (available at `java.sun.com/j2se/1.5.0/docs/api/index.html`) and in the software installation directory (which is called `jdk1.5.0`). Sun's Web site accepts URLs that replace 1.5.0 with 5.0. For example, you can use the URL `java.sun.com/j2se/5.0/docs/api/index.html` to access the online documentation.]

New Interior Design

Working with the creative services team at Prentice Hall, we redesigned the interior styles for our *How to Program* Series. In response to reader requests, we now place the key terms and the index's page reference for each defining occurrence in **blue, bold style** text for easier reference. We emphasize on-screen components in the bold Helvetica font (e.g., the **File** menu) and emphasize Java program text in the `Lucida` font (for example, `int x = 5`).

Syntax Coloring
This book is presented in full color to show programs and their outputs as they typically appear on a computer screen. We syntax color all the Java code, as most Java integrated-development environments and code editors do. This greatly improves code readability—an especially important goal, given that this book contains 20,383 lines of code. Our syntax-coloring conventions are as follows:

```
comments appear in green
keywords appear in dark blue
errors and JSP scriptlet delimiters appear in red
constants and literal values appear in light blue
all other code appears in black
```

Code Highlighting
Extensive code highlighting makes it easy for readers to spot each program's featured segments and helps students review the material rapidly when preparing for exams or labs.

"Code Washing"
"Code washing" is our term for applying comments, using meaningful identifiers, applying uniform indentation conventions and using vertical spacing to highlight significant program units. This process results in programs that are easy to read and self-documenting. We have extensively "code washed" of all the source-code programs in the text and in the book's ancillaries. We have worked hard to make our code exemplary.

Early Classes and Objects Approach
Students are still introduced to the basic concepts and terminology of object technology in Chapter 1. In the previous edition, students began developing their customized classes and objects in Chapter 8, but in this edition, they start doing that in our completely new Chapter 3. Chapters 4–7 have been carefully rewritten from an "early classes and objects approach." For all intents and purposes, this new edition is object-oriented from the start and throughout the text. Moving the discussion of objects and classes to earlier chapters gets students "thinking about objects" immediately and mastering these concepts more thoroughly. Java is not trivial by any means, but it's fun to program with, and students can see immediate results. Students can get text-based and graphical programs running quickly by using Java's extensive class libraries of reusable components.

Carefully Tuned Treatment of Object-Oriented Programming in Chapters 8–10
We performed a high-precision upgrade of *Java How to Program, 5/e*. This edition is clearer and more accessible—especially if you are new to object-oriented programming (OOP). We completely rewrote the OOP chapters with an integrated case study in which we develop an employee payroll hierarchy, and we motivate interfaces with a payables hierarchy.

Case Studies
This book is loaded with case studies that span multiple sections and chapters. Often we build on a class introduced earlier in the book to demonstrate new programming concepts later in the book. Students learn the new concepts in the context of applications that they already know. These case studies include the development of the:

- `GradeBook` class in Chapters 3, 4, 5 and 7
- ATM application in the optional OOD/UML sections of Chapters 1–8 and 10
- polymorphic drawing program in the optional GUI and Graphics track in Chapters 3–12
- `Time` class in several sections of Chapter 8
- `Employee` payroll application in Chapter 9 and Chapter 10

Integrated `GradeBook` *Case Study*

To reinforce our early classes presentation, we present an integrated case study using classes and objects in Chapters 3–5 and 7. We incrementally build a `GradeBook` class that represents an instructor's grade book and performs various calculations based on a set of student grades—finding the average, finding the maximum and minimum, and printing a bar chart. Our goal is to familiarize students with the important concepts of objects and classes through a real-world example of a substantial class. We develop this class from the ground up, constructing methods from control statements and carefully developed algorithms, and adding instance variables and arrays as needed to enhance the functionality of the class.

GUI and Graphics Case Study (Optional)

The optional GUI and Graphics Case Study in Chapters 3–12 demonstrates techniques for adding visual elements to applications. It is designed for those who want to begin learning Java's powerful capabilities for creating graphical user interfaces (GUIs) and graphics. Each section introduces basic concepts and provides visual, graphical examples and complete source code. In the first few sections, we show how to create simple graphical applications. In the subsequent sections, we use the object-oriented programming concepts presented through Chapter 10 to create an application that draws a variety of shapes polymorphically. In Chapter 11, we add an event-driven GUI interface and in Chapter 12 we incorporate graphics features of Java 2D.

Unified Modeling Language (UML)—Using the UML 2.0 to Develop an Object-Oriented Design of an ATM

The Unified Modeling Language™ (UML) has become the preferred graphical modeling language for designing object-oriented systems. All the UML diagrams in the book comply with the new UML 2.0 specification. We use UML class diagrams to visually represent classes and their inheritance relationships, and we use UML activity diagrams to demonstrate the flow of control in each of Java's several control statements.

This *Sixth Edition* includes a new, optional (but highly recommended) case study on object-oriented design using the UML. The case study was reviewed by a distinguished team of OOD/UML academic and industry professionals, including leaders in the field from Rational (the creators of the UML and now a division of IBM) and the Object Management Group (responsible for maintaining and evolving the UML). In the case study, we design and fully implement the software for a simple automatic teller machine (ATM). The "Software Engineering Case Study" sections at the ends of Chapters 1–8 and 10 present a carefully paced introduction to object-oriented design using the UML. We introduce a concise, simplified subset of the UML 2.0, then guide the reader through a first design experience intended for the novice object-oriented designer/programmer. The case study is not an exercise; rather, it is an end-to-end learning experience that concludes with a detailed walkthrough of the complete Java code. The "Software Engineering Case Study" sections help students develop an object-oriented design to complement the object-oriented programming concepts they begin learning in Chapter 1 and implementing in Chapter 3. In the first of these sections at the end of Chapter 1, we introduce basic concepts and terminology of OOD. In the optional "Software Engineering Case Study" sections at the ends of Chapters 2–5, we consider more substantial issues, as we undertake a challenging problem with the techniques of OOD. We analyze a typical requirements document that specifies a system to be built, determine the objects needed to implement that system, determine the attributes these objects need to have, determine the behaviors these objects need to exhibit and specify how the objects must interact with one another to meet the system requirements. In Appendix J, we include a Java implementation of the object-oriented system that we designed in the earlier chapters. This case study will help prepare students for the kinds of substantial projects they will encounter in industry. We employ a carefully developed, incremental object-oriented design process to produce a UML model for our ATM system. From this design, we produce a substantial working Java implementation using key programming notions, including classes, objects, encapsulation, visibility, composition, inheritance and polymorphism.

Database and Web-Applications Development with JDBC, Servlets and JSP
Chapter 25, Accessing Databases with JDBC, demonstrates how to build data-driven applications with the JDBC™ API. Chapter 26, Servlets, and Chapter 27, JavaServer Pages (JSP), expand our treatment of Internet and Web programming topics, giving readers everything they need to develop their own Web-based applications. Readers will learn how to build so-called *n*-tier applications, in which the functionality provided by each tier can be distributed to separate computers across the Internet or executed on the same computer. In particular, we build a three-tier Web-based survey application and a three-tier Web-based guest-book application. Each application's information is stored in the application's data tier—in this book, a database implemented with MySQL (a trial version is on the CD that accompanies this book). The user enters requests and receives responses at each application's client tier, which is typically a computer running a Web browser such as Microsoft Internet Explorer or Netscape. Web browsers, of course, know how to communicate with Web sites throughout the Internet. The middle tier contains both a Web server and one or more application-specific servlets (Java programs that extend the server to handle requests from client browsers) or JavaServer Pages (an extension of servlets that simplifies handling requests and formatting responses to client browsers). We use Apache's Tomcat Web server for these examples. Tomcat, which is the reference implementation for the servlets and JavaServer Pages technologies, is included on the CD that accompanies this book and is available free for download from `www.apache.org`. Tomcat communicates with client Web browsers across the Internet using the HyperText Transfer Protocol (HTTP)—the foundation of the World Wide Web. We discuss the crucial role of the Web server in Web programming and provide extensive examples demonstrating interactions between a Web browser and a Web server.

Teaching Approach

Java How to Program, 6/e contains a rich collection of examples, exercises and projects drawn from many fields to provide the student with a chance to solve interesting real-world problems. The book concentrates on the principles of good software engineering and stresses program clarity. We avoid arcane terminology and syntax specifications in favor of teaching by example. Our code examples have been tested on popular Java platforms. We are educators who teach leading-edge topics in industry classrooms worldwide. Dr. Harvey M. Deitel has 20 years of college teaching experience and 15 years of industry teaching experience. Paul Deitel has 12 years of industry teaching experience and is one of the world's most experienced Java corporate trainers, having taught about 100 Java courses at all levels since 1996 to government, industry, military and academic clients of Deitel & Associates.

Learning Java via the LIVE-CODE Approach

Java How to Program, 6/e, is loaded with LIVE-CODE examples—each new concept is presented in the context of a complete working Java application that is immediately followed by one or more sample executions showing the program's inputs and outputs. This style exemplifies the way we teach and write about programming. We call this method of teaching and writing the LIVE-CODE Approach. *We use programming languages to teach programming languages.* Reading the examples in the text is much like typing and running them on a computer. We provide all the source code for the book's examples on the accompanying CD and at `www.deitel.com`—making it easy for students to run each example as they study it.

World Wide Web Access

All of the source-code examples for *Java How to Program, 6/e,* (and our other publications) are available on the Internet as downloads from the following Web sites:

```
www.deitel.com
www.prenhall.com/deitel
```

Registration is quick and easy, and the downloads are free. We suggest that students download all the examples, then run each program as they read the corresponding text discussions. Making changes to the examples and immediately seeing the effects of those changes is a great way to enhance your Java learning experience.

Objectives

Each chapter begins with a statement of objectives. This lets students know what to expect and gives them an opportunity, after reading the chapter, to determine if they have met these objectives. This is a confidence builder and a source of positive reinforcement.

Quotations

The learning objectives are followed by quotations. Some are humorous, philosophical or offer interesting insights. We hope that you will enjoy relating the quotations to the chapter material. Many of the quotations are worth a second look after reading the chapter.

Outline

The chapter outline helps students approach the material in a top-down fashion, so they can anticipate what is to come, and set a comfortable and effective learning pace.

20,383 Lines of Code in 256 Example Programs (with Program Outputs)

Our LIVE-CODE programs range in size from just a few lines of code to substantial examples containing hundreds of lines of code. Each program is followed by a window containing the outputs produced when the program is run, so students can confirm that the programs run as expected. Relating outputs to the program statements that produce them is an excellent way to learn and to reinforce concepts. Our programs demonstrate the diverse features of Java. The code is syntax colored, with Java keywords, comments and other program text each appearing in different colors. This facilitates reading the code—students will especially appreciate the syntax coloring when they read the larger programs.

816 Illustrations/Figures

An abundance of charts, tables, line drawings, programs and program outputs is included. We model the flow of control in control statements with UML activity diagrams. UML class diagrams model the fields, constructors and methods of classes. We use additional types of UML diagrams throughout our optional OOD/UML ATM case study.

481 Programming Tips

We include programming tips to help students focus on important aspects of program development. We highlight these tips in the form of *Good Programming Practices*, *Common Programming Errors*, *Error-Prevention Tips, Look-and-Feel Observations*, *Performance Tips*, *Portability Tips* and *Software Engineering Observations*. These tips and practices represent the best we have gleaned from a combined six decades of programming and teaching experience. One of our students—a mathematics major—told us that she feels this approach is like the highlighting of axioms, theorems and corollaries in mathematics books; it provides a basis on which to build good software.

Good Programming Practices

Good Programming Practices *are tips for writing clear programs. These techniques help students produce programs that are more readable, self-documenting and easier to maintain.*

Common Programming Errors

Students who are new to programming (or a programming language) tend to make certain errors frequently. Focusing on these Common Programming Errors *reduces the likelihood that students will make the same mistakes and shortens long lines outside instructors' offices during office hours!*

Error-Prevention Tips

When we first designed this "tip type," we thought we would use it strictly to tell people how to test and debug Java programs. In fact, many of the tips describe aspects of Java that reduce the likelihood of "bugs" and thus simplify the testing and debugging processes.

Look-and-Feel Observations

Look-and-Feel Observations *highlight graphical user interface conventions. These observations help students design their own graphical user interfaces in conformance with industry norms.*

Performance Tips

In our experience, teaching students to write clear and understandable programs is by far the most important goal for a first programming course. But students want to write the programs that run the fastest, use the least memory, require the smallest number of keystrokes, or dazzle in other nifty ways. Students really care about performance. They want to know what they can do to "turbo charge" their programs. So we highlight opportunities for improving program performance—making programs run faster or minimizing the amount of memory that they occupy.

Portability Tips

One of Java's "claims to fame" is "universal" portability, so some programmers assume that an application written in Java will automatically be "perfectly" portable across all Java platforms. Unfortunately, this is not always the case. We include Portability Tips *to help students write portable code and to provide insights on how Java achieves its high degree of portability.*

Software Engineering Observations

The object-oriented programming paradigm requires a complete rethinking about the way we build software. Java is an effective language for performing good software engineering. The Software Engineering Observations *highlight architectural and design issues that affect the construction of software systems, especially large-scale systems. Much of what the student learns here will be useful in upper-level courses and in industry as the student begins to work with large, complex, real-world systems.*

Wrap-Up Section
New in this edition, each chapter ends with a brief "wrap-up" section that recaps the topics that were presented. Each section also helps the student transition to the next chapter.

Summary (1,303 Summary Bullets)
Each chapter ends with additional pedagogical devices. We present a thorough, bullet-list-style summary of the chapter. On average, there are 45 summary bullets per chapter. This helps the students review and reinforce key concepts.

Terminology (2,388 Terms)
We include an alphabetized list of the important terms defined in each chapter—again, for further reinforcement. There is an average of 82 terms per chapter. Each term also appears in the index, and the defining occurrence of each term is highlighted in the index with a **blue, bold** page number so the student can locate the definitions of terms quickly.

701 Self-Review Exercises and Answers (Count Includes Separate Parts)
Extensive self-review exercises and answers are included for self-study. This gives the student a chance to build confidence with the material and prepare for the regular exercises. We encourage students to do all the self-review exercises and check their answers.

874 Exercises (Count Includes Separate Parts)
Each chapter concludes with a set of exercises, including simple recall of important terminology and concepts; writing individual Java statements; writing small portions of Java methods and classes; writing complete Java methods, classes, applications and applets; and writing major term projects. The large number of exercises across a wide variety of areas enables instructors to tailor their courses to the unique needs of their classes and to vary course assignments each semester. Instructors can use these exercises to form homework assignments, short quizzes and/or major examinations. The solutions for the vast majority of the exercises are included on the *Instructor's Resource CD (IRCD),* which is *available only to instructors* through their Prentice Hall representatives. [**NOTE: Please do not write to us requesting the Instructor's CD. Distribution of this ancillary is limited strictly to college instructors teaching from the book. Instructors may obtain the solutions manual only from their Prentice Hall representatives.**] Students will have access to approximately half the exercises in the book in the free, Web-based *Cyber Classroom* which will be avail-

able in Spring 2005. For more information about the availability of the *Cyber Classroom*, please visit `www.deitel.com` or sign up for the free *Deitel® Buzz Online* e-mail newsletter at `www.deitel.com/newsletter/subscribe.html`.

Approximately 7150 Index Entries

We have included an extensive index at the back of the book. This helps students find terms or concepts by keyword. The Index is useful to people reading the book for the first time and is especially useful to practicing programmers who use the book as a reference.

"Double Indexing" of Java LIVE-CODE Examples

Java How to Program, 6/e has 256 live-code examples and 874 exercises (including parts). We have double indexed each of the live-code examples and most of the more substantial exercises. For every source-code program in the book, we indexed the figure caption both alphabetically and as a subindex item under "Examples." This makes it easier to find examples using particular features. The more substantial exercises are also indexed both alphabetically and as subindex items under "Exercises."

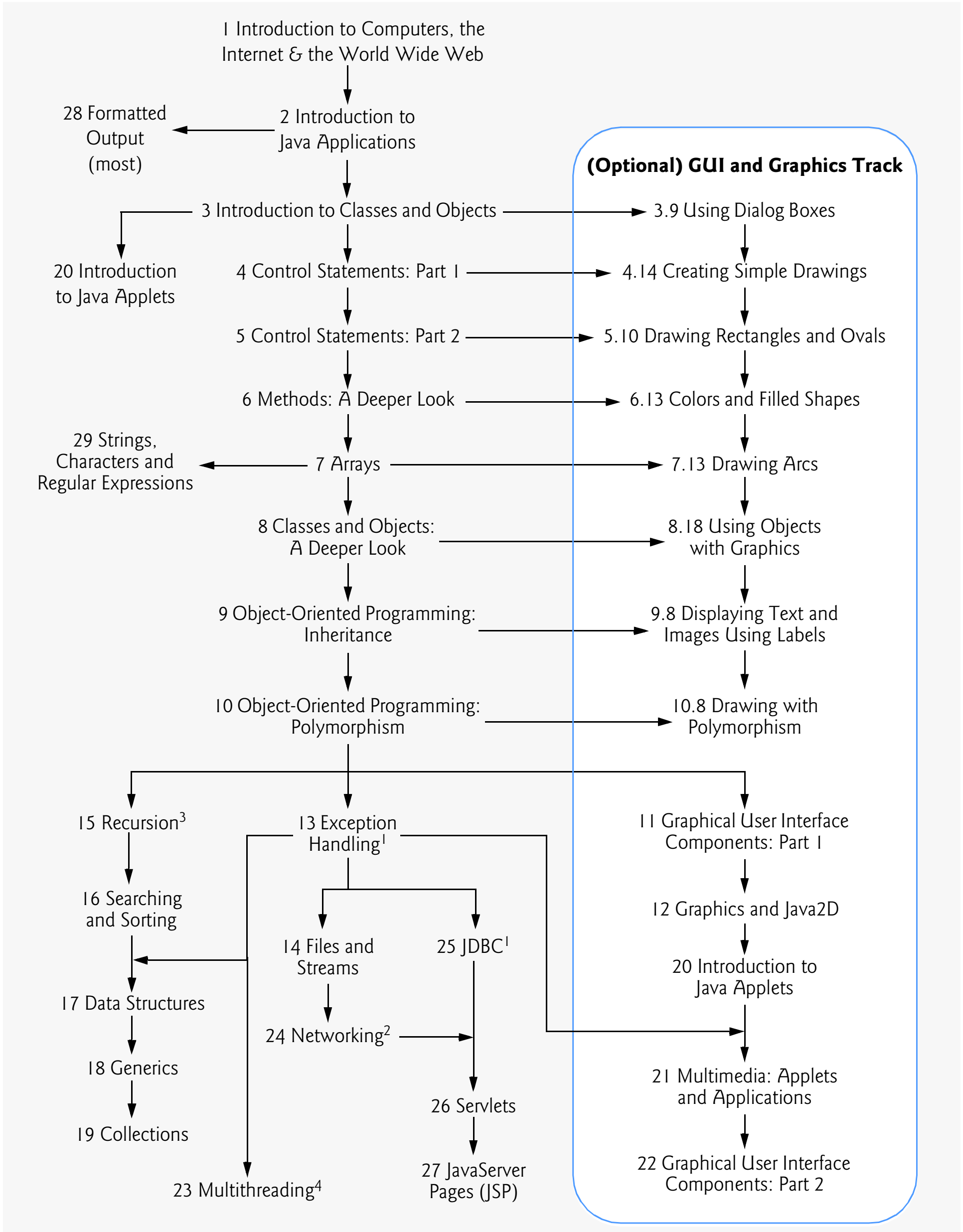
Tour of the Book

In this section, we take a tour of the many capabilities of Java that we explore in *Java How to Program, 6/e*. Figure 1 illustrates the dependencies among the chapters. We recommend studying these topics in the order indicated by the arrows, though other orders are possible. This book is widely used in all levels of Java programming courses. Search the Web for "syllabus," "Java" and "Deitel" to find syllabi used with recent editions this book.

Chapter 1—Introduction to Computers, the Internet and the Web—discusses what computers are, how they work and how they are programmed. The chapter gives a brief history of the development of programming languages from machine languages, to assembly languages and to high-level languages. The origin of the Java programming language is discussed. The chapter includes an introduction to a typical Java programming environment. Chapter 1 walks readers through a "test drive" of a typical Java application. This chapter also introduces basic object technology concepts and terminology and the Unified Modeling Language.

Chapter 2—Introduction to Java Applications—provides a lightweight introduction to programming applications in the Java programming language. The chapter introduces nonprogrammers to basic programming concepts and constructs. The programs in this chapter illustrate how to display data on the screen and how to obtain data from the user at the keyboard. This chapter introduces J2SE 5.0's new `Scanner` class, which greatly simplifies obtaining user input. This chapter also introduces some of J2SE 5.0's new formatted output capabilities with method `System.out.printf`. Chapter 2 ends with detailed treatments of decision making and arithmetic operations.

Chapter 3—Introduction to Classes and Objects—introduces classes, objects, methods, constructors and instance variables using five simple real-world examples. The first four of these begin our **case study on developing a grade-book class** that instructors can use to maintain student test scores. The first example presents a `GradeBook` class with one method that simply displays a welcome message when it is called. We show how to create an object of that class and call the method so that it displays the welcome message. The second example modifies the first by allowing the method to receive a course name as


1 Introduction to Computers, the Internet & the World Wide Web
2 Introduction to Java Applications
28 Formatted Output (most)
(Optional) GUI and Graphics Track
3 Introduction to Classes and Objects
3.9 Using Dialog Boxes
20 Introduction to Java Applets
4 Control Statements: Part 1
4.14 Creating Simple Drawings
5 Control Statements: Part 2
5.10 Drawing Rectangles and Ovals
6 Methods: A Deeper Look
6.13 Colors and Filled Shapes
29 Strings, Characters and Regular Expressions
7 Arrays
7.13 Drawing Arcs
8 Classes and Objects: A Deeper Look
8.18 Using Objects with Graphics
9 Object-Oriented Programming: Inheritance
9.8 Displaying Text and Images Using Labels
10 Object-Oriented Programming: Polymorphism
10.8 Drawing with Polymorphism
15 Recursion3
13 Exception Handling1
11 Graphical User Interface Components: Part 1
16 Searching and Sorting
12 Graphics and Java2D
14 Files and Streams
25 JDBC1
20 Introduction to Java Applets
17 Data Structures
24 Networking2
18 Generics
21 Multimedia: Applets and Applications
26 Servlets
19 Collections
27 JavaServer Pages (JSP)
22 Graphical User Interface Components: Part 2
23 Multithreading4

1. Dependent on Chapter 11 for GUI used in one example.
2. Dependent on Chapter 20 for one example that uses an applet. The two large case studies at the end of this chapter each depend on Chapter 22 for GUI and Chapter 23 for multithreading.
3. Dependent on Chapters 11 and 12 for GUI and graphics used in one example.
4. Dependent on Chapter 11 for GUI used in one example. Dependent on Chapters 18 and 19 for one example.

Fig. 1 | Flowchart illustrating the dependencies among chapters in *Java How to Program, 6/e*.

an argument and by displaying the name as part of the welcome message. The third example illustrates storing the course name in a `GradeBook` object. For this version of the class, we also show how to set the course name and obtain the course name using methods. The fourth example demonstrates how the data in a `GradeBook` object can be initialized when the object is created—the initialization is performed by the class's constructor. The last example introduces floating-point numbers in the context of a bank account class that maintains a customer's balance. The chapter describes how to declare a class and use it to create an object and then discusses how to declare methods in a class to implement the class's behaviors, how to declare instance variables in a class to implement the class's attributes and how to call an object's methods to make them perform their tasks. Chapter 3 explains the differences between instance variables of a class and local variables of a method, how to use a constructor to ensure that an object's data is initialized when the object is created, and the differences between primitive and reference types.

Chapter 4—Control Statements: Part 1—focuses on the program-development process. The chapter discusses how to take a problem statement and develop a working Java program from it, including performing intermediate steps in pseudocode. The chapter introduces some primitive types and simple control statements for decision making (`if` and `if...else`) and repetition (`while`). We examine counter-controlled and sentinel-controlled repetition using the **`GradeBook` class** from Chapter 3, and introduce Java's increment, decrement and assignment operators. The chapter includes **two enhanced versions of the `GradeBook` class**, each based on Chapter 3's final version. These versions each include a method that uses control statements to calculate the average of a set of student grades. In the first version, the method uses counter-controlled repetition to input 10 student grades from the user, then determines the average grade. In the second version, the method uses sentinel-controlled repetition to input an arbitrary number of grades from the user, then calculates the average of the grades that were entered.The chapter uses simple UML activity diagrams to show the flow of control through each of the control statements.

Chapter 5—Control Statements: Part 2—continues the discussions of Java control statements with examples of the `for` repetition statement, the `do...while` repetition statement, the `switch` selection statement, the `break` statement and the `continue` statement. We create an **enhanced version of class `GradeBook`** that uses a `switch` statement to count the number of A, B, C, D and F grade equivalents in a set of numeric grades entered by the user. This version uses sentinel-controlled repetition to input the grades. While reading the grades from the user, a method modifies instance variables that keep track of the sum of the grades entered and the number of grades entered, as well as the count of grades in each letter grade category. Other methods of the class then use these instance variables to perform the averaging calculation and display a summary report based on the grades entered. The chapter also discusses logical operators.

Chapter 6—Methods: A Deeper Look—takes a deeper look inside objects and their methods. We discuss class-library methods and examine more closely how students can build their own methods. We present our first example of a method with multiple parameters. A portion of the chapter focuses on developing a game playing application that uses random number generation to simulate the rolling of dice. This application divides its required work into small, reusable methods. The techniques presented in Chapter 6 are essential to the production of properly organized programs, especially the larger programs that system programmers and application programmers are likely to develop. The topic of method overloading (i.e., allowing multiple methods to have the same name provided they have different "signatures") is motivated and explained clearly. We introduce the method call stack to explain how Java is able to keep track of which method is currently executing, how local variables of methods are maintained in memory and how a method knows where to return after it completes execution. Additional chapter topics include `static` methods, `static` fields, class `Math`, enumerations and the scope of declarations.

Chapter 7—Arrays—explains how to process lists and tables of values. Arrays in Java are objects, further evidence of Java's commitment to almost 100% object orientation. We discuss the structuring data in arrays of data items of the same type. The chapter presents numerous examples of both one-dimensional arrays and two-dimensional arrays. The examples investigate common array manipulations, printing bar charts, passing arrays to methods and an introduction to the field of survey data analysis (with simple statistics). The chapter includes a case study that simulates the shuffling and dealing of playing cards in a game playing application. Also, the chapter includes the **final two `GradeBook` case study sections**, in which we use arrays to store student grades for the duration of a program's execution. Previous versions of the class process a set of grades entered by the user, but do not maintain the individual grade values in instance variables of the class. Thus, repeat calculations require the user to reenter the same grades. In this chapter, we use arrays to enable an object of the `GradeBook` class to maintain a set of grades in memory, thus eliminating the need to repeatedly input the same set of grades. The first version of the class in this chapter stores the grades in a one-dimensional array and can produce a report containing the average of the grades, the minimum and maximum grades and a bar chart representing the grade distribution. The second version of the class in this chapter (i.e., the final version in the case study) uses a two-dimensional array to store the grades of a number of students on multiple exams in a semester. This version can calculate each student's semester average, as well as the minimum and maximum grades across all grades received for the semester. The class also produces a bar chart displaying the overall grade distribution for the semester. This chapter introduces J2SE 5.0's new enhanced `for` statement to traverse the elements of an array. Variable-length argument lists (new in J2SE 5.0) are also demonstrated.

Chapter 8—Objects: A Deeper Look—begins a deeper discussion of objects and classes. The chapter represents a wonderful opportunity for teaching data abstraction the "right way"—through a language (Java) expressly devoted to implementing new types. Building on the concepts introduced in Chapters 3–7, the chapter focuses on the essence and terminology of classes and objects. In the context of the **`Time` class case study**, Chapter 8 discusses implementing Java classes, accessing class members, enforcing information hiding with access modifiers, separating interface from implementation, using access methods and utility methods and initializing objects with constructors. We discuss declaring and using constants, composition, the `this` reference, `static` class members and examples of popular abstract data types such as stacks and queues. We introduce the `package` statement and discuss how to create reusable packages, and present J2SE 5.0's new `static` import and `enum` capabilities. Java's `enum`s are class types that can have methods, constructors and fields.

Chapter 9—Object-Oriented Programming: Inheritance—introduces one of the most fundamental capabilities of object-oriented programming languages—inheritance—which is a form of software reusability in which new classes are developed quickly and easily by absorbing the capabilities of existing classes and adding appropriate new capabilities. In the context of an **`Employee` hierarchy**, this substantially revised chapter presents a five-example sequence demonstrating `private` data, `protected` data and software reuse via inheritance. We begin by demonstrating a class with `private` instance variables and `public` methods to manipulate that data. Next, we implement a second class with additional capabilities. To do this, we duplicate much of the first example's code. The third example begins our discussion of inheritance and software reuse—we use the class from the first example as a superclass and inherit its data and functionality into a new subclass. This example introduces the inheritance mechanism and demonstrates that a subclass cannot access its superclass's `private` members directly. This motivates our fourth example, in which we introduce `protected` data in the superclass and demonstrate that the subclass can indeed access the `protected` data inherited from the superclass. The last example in the sequence demonstrates proper software engineering by defining the superclass's data as `private` and using the superclass's `public` methods (that were inherited by the subclass) to manipulate the superclass's `private` data in the subclass. The chapter discusses the notions of superclasses and subclasses, direct and indirect superclasses, use of constructors in superclasses and subclasses, and software engineering with inheritance. The chapter also compares inheritance (*is a* relationships) with composition (*has a* relationships).

Chapter 10—Object-Oriented Programming: Polymorphism—deals with another fundamental capability of object-oriented programming: polymorphic behavior. The completely revised Chapter 10 builds on the inheritance concepts presented in Chapter 9 and focuses on the relationships among classes in a class hierarchy, and the powerful processing capabilities that these relationships enable. A feature of this chapter is its two polymorphism case studies—a **payroll system using an abstract class `Employee`** and an **accounts payable system using an interface `Payable`**. Both case studies expand on the **`Employee` hierarchy** introduced in Chapter 9. The first case study processes an array of variables that contain references to `Employee` objects. All the objects referenced by the array elements have a common abstract superclass `Employee` containing the set of methods common to every class in the hierarchy. The case study demonstrates that when a method is invoked via a superclass reference, the subclass-specific version of that method is invoked. The case study also shows how a program that processes objects polymorphically can perform type-specific processing by determining the type of the object currently being processed via operator `instanceof`. This chapter distinguishes between abstract classes and concrete classes, and introduces interfaces—Java's replacement for the feature of C++ called multiple inheritance—in the context of the second payroll case study.

Chapter 11—Graphical User Interface Components: Part 1—introduces several of Java's Swing components for creating programs with user-friendly graphical user interfaces (GUIs). These platform-independent GUI components are written entirely in Java, providing them with great flexibility. Swing components can be customized to look like the computer platform on which the program executes, or they can use the standard Java look-and-feel to provide an identical user interface on all computer platforms. GUI development is a huge topic, so we divided it into two chapters. These chapters cover the material in sufficient depth to enable you to build rich user interfaces. The chapter illustrates GUI principles, the `javax.swing` hierarchy, labels, buttons, lists, textfields, combo boxes, checkboxes, radio buttons, panels, handling mouse events, handling keyboard events and layout managers used to position components. We present the powerful concept of nested classes that help hide implementation details. Then, the chapter demonstrates our first

GUI-based applications as part of a more complete introduction to event handling. In these examples, we use nested classes to define the event handlers for several GUI components.

Chapter 12—Graphics and Java 2D—presents Java's graphical capabilities. We discuss graphics contexts and graphics objects; drawing strings, characters and bytes; color and font control; screen manipulation; and paint modes and drawing lines, rectangles, rounded rectangles, three-dimensional rectangles, ovals, arcs and polygons. We introduce the Java 2D API, which provides powerful two-dimensional graphics capabilities. Figure 12.29 and Fig. 12.30 are examples of how easy it is to use the Java 2D API to create complex graphics effects such as textures and gradients.

Chapter 13—Exception Handling—is one of the most important chapters in the book from the standpoint of building "mission-critical" or "business-critical" applications. Programmers need to be concerned with, "What happens when the component I call on to do a job experiences difficulty? How will that component signal that it had a problem?" To use a Java component, you need to know not only how that component behaves when "things go well," but also what exceptions that component "throws" when "things go poorly." The chapter distinguishes between `Exception`s and rather serious system `Error`s, and distinguishes further between checked `Exception`s (which the compiler requires an application to catch or declare) and unchecked `Exception`s (which an application is not required to catch or declare). The chapter discusses the elements of exception handling, including `try` blocks, `catch` blocks, `finally` blocks and the `throw` statement. The chapter also introduces Java's chained-exception facility. The material in this chapter is crucial to many of the examples in the remainder of the book.

Chapter 14—Files and Streams—deals with input/output that is accomplished through streams of data directed to and from files. The chapter begins with an introduction to the data hierarchy from bits, to bytes, to fields, to records, to files. Next, Java's view of files and streams is presented as well as the differences between text files and binary files. We show how programs pass data to secondary storage devices, like disks, and how programs retrieve data already stored on those devices. We demonstrate manipulating both sequential- and random-access files. We discuss class `File` which programs use to obtain information about files and directories. The chapter explains how objects can be input and output in their entirety by using object serialization. We also introduce the `JFileChooser` dialog, which enables users of a program to easily select files. Many of the stream processing techniques demonstrated in this chapter can also be used to create applications that communicate with one another across networks (Chapter 24), such as the Internet.

[Note: The next five chapters form the core of a nice Java-specific data structures course.]
Chapter 15—Recursion (New Chapter)—discusses recursive methods—i.e., methods that call themselves. We discuss the notions of base cases, recursion steps and infinite recursion. We present some popular recursion examples, including factorials, the Fibonacci series, string permutations, The Towers of Hanoi and fractals. We explain how recursion works "under the hood," including the order in which method activation records are pushed on or popped off the program execution stack. We compare and contrast recursive methods and iterative methods, and explain when to choose each. We introduce recursive backtracking. The chapter exercises include recursive approaches to: raising an integer to an integer power, visualizing recursion, greatest common divisor, palindromes, the Eight Queens problem (recursive backtracking), printing an array, minimum value in an array, star fractal, maze traversal (recursive backtracking), and generating mazes. The recursive

examples and exercises of Chapter 16 and Chapter 17 include merge sort, linear search, binary search, quicksort, binary tree insert, (preorder, inorder and postorder) binary tree traversals, and various linked list manipulations.

Chapter 16—Sorting and Searching (**New Chapter**)—discusses two of the most important classes of algorithms in Computer Science. We consider a variety of specific algorithms for each and compare them with regard to their memory consumption and processor consumption (introducing Big O notation, which indicates how hard an algorithm may have to work to solve a problem). Searching data involves determining whether a value (referred to as the search key) is present in the data and, if so, finding the value's location. In the examples and exercises of this chapter, we discuss a variety of searching algorithms, including: linear search, binary search, recursive linear search and recursive binary search; Chapter 17 discusses linear list search and the binary tree search; Chapter 19 discusses the `binarySearch` method of class `Collections`. Sorting places data in ascending or descending order based on one or more sort keys. Chapter 16 discusses through examples and exercises the selection sort, insertion sort, recursive merge sort, bubble sort, bucket sort and the recursive quicksort; Chapter 17 discusses the binary tree sort and Chapter 19 discusses the `sort` method of class `Collections` and the `SortedSet` collection. Chapter 16 also discusses the notion of loop invariants.

Chapter 17—Data Structures—discusses the techniques used to create and manipulate dynamic data structures, such as linked lists, stacks, queues and trees. For each type of data structure, we present examples with sample outputs. Although it is valuable to know how these classes are implemented, Java programmers will quickly discover that most of the data structures they need are available in class libraries, such as Java's own `java.util` that we discuss in Chapter 19. This chapter also introduces the J2SE 5.0's boxing capability, which simplifies converting between primitive-type values and type-wrapper objects. The chapter offers a rich selection of exercises, including a special section on building your own compiler that will enable you to compile high-level language programs into machine language code that will execute on the Simpletron simulator in the exercises of Chapter 7.

Chapter 18—Generics (**New Chapter**)—presents one of J2SE 5.0's new features: generics. The examples demonstrate generic methods and generic classes which enable programmers to specify, with a single method declaration, an entire range of related methods or, with a single class declaration, an entire range of related types, respectively. The chapter discusses how this new feature achieves legacy compatibility with earlier versions, such as through raw types. This chapter also introduces wildcards, a powerful generics concept that enables programmers to represent an "unknown type."

Chapter 19—Collections—discusses the `java.util` classes of the Java Collections Framework that provide predefined implementations of many of the data structures discussed in Chapter 17. Collections provide Java programmers with a standard set of data structures for storing and retrieving data and a standard set of algorithms (i.e., procedures) that allow programmers to manipulate the data (such as searching for particular data items and sorting data into ascending or descending order). The chapter examples demonstrate collections, such as linked lists, trees, maps and sets, and algorithms for searching, sorting, finding the maximum value, finding the minimum value and so on. We also demonstrate how generics are used in the collections API.

Chapter 20—Introduction to Applets—introduces Java applets, which are Java programs designed to be transported over the Internet and executed in Web browsers, such as

Microsoft Internet Explorer, Netscape, Opera and Mozilla Firefox. The chapter shows several of the demonstration applets supplied with the JDK. We then write Java applets that perform tasks similar to the programs of Chapter 2, and we explain the similarities and differences between applets and applications. This chapter can be covered early in the book—after Chapter 3. Applets are GUIs and graphical, so for a deeper understanding of Chapter 20, cover it after Chapters 11 and 12 (these prerequisites are not absolutely necessary). The next chapter and Chapter 24 present additional interesting applets.

Chapter 21—Multimedia: Images, Animation and Audio—presents some of Java's capabilities that make programs come alive through multimedia. The chapter discusses images and image manipulation, audio, animation and video. We present an image-map application with the icons from the programming tips shown earlier in the Preface and that appear throughout the book. As the user moves the mouse pointer across the icons, the tip type is displayed. Once you have read this chapter, you will be eager to try out all these techniques, so we have included many exercises to challenge and entertain you.

Chapter 22—Graphical User Interface Components: Part 2—continues the Swing discussion started in Chapter 11. Through its programs, tables and line drawings, the chapter illustrates GUI design principles, sliders, windows, menus, pop-up menus, changing the look-and-feel, multiple-document interfaces, tabbed panes and using advanced layout managers.

Chapter 23—Multithreading—introduces Java's capabilities that enable programmers to implement applications that can perform tasks concurrently. We introduce threads, and present a thread state transition diagram. We overview thread priorities and thread scheduling. We then present a simple example that creates three threads. Next, we introduce the producer/consumer relationship in which a producer thread places data in a shared location and a consumer thread then reads that data. In this context, we use a sequence of examples to illustrate problems that may arise when multiple threads manipulate shared data. The first example demonstrates these problems. We then introduce features of J2SE 5.0's new concurrency API and use them to "synchronize" access to the shared data to ensure correct operation. Next, we enhance the example to use a shared circular buffer in which the producer can place up to three values before the consumer attempts to read a value. The last example in the sequence reimplements the circular buffer example using J2SE 5.0's `ArrayBlockingQueue` class, which automatically synchronizes access to its contents. This is significant, because programming concurrent applications is a difficult and error-prone undertaking. The last example in the chapter demonstrates the producer/consumer relationship using the pre-J2SE 5.0 synchronization techniques. Though we present several of J2SE 5.0's lower-level synchronization features in this chapter for those who are interested, the vast majority of programmers should use the predefined classes like `ArrayBlockingQueue` to provide synchronized access to data in multithreaded applications.

[Note: The next four chapters form the core of a nice introduction to distributed computing in Java.]

Chapter 24—Networking—focuses on Java programs that communicate over computer networks. This chapter presents Java's lowest-level networking capabilities. The chapter examples illustrate an applet interacting with the browser in which it executes, creating a mini Web browser, communicating between two Java programs using streams-based sockets and communicating between two Java programs using packets of data. A key feature of the chapter is the implementation of a collaborative client/server Tic-Tac-Toe game in which two clients play Tic-Tac-Toe against each other arbitrated by a multithreaded server—great stuff! The capstone example in the chapter is the `DeitelMessenger` case study, which simulates many of today's popular instant-messaging applications that enable computer users to communicate with friends, relatives and coworkers over the Internet. This 788-line, multithreaded, client/server case study uses most of the programming techniques presented up to this point in the book.

Chapter 25—Accessing Databases with JDBC™—discusses the JDBC API, which enables applications to access data in databases. Today's most popular database systems are relational databases. The examples in this chapter use the MySQL database management system (DBMS), which is included on the CD that accompanies the book. In the context of a database that contains information about many of our publications, this chapter introduces JDBC and uses it to connect to a MySQL database, then to manipulate its content. We discuss the Structured Query Language (SQL) and how it can be used to extract information from, insert information into and update data in a database. The examples enable you to obtain data from the database with SQL queries and display the data in a `JTable` GUI component. In addition, we discuss J2SE 5.0's `RowSets`, which simplify connecting to a database and accessing its data. In the next two chapters, we use the JDBC techniques in Chapter 25 to build data-driven three-tier Web applications.

Chapter 26—Servlets—discusses servlets—Java programs that extend the functionality of Web servers. Servlets are effective for developing Web-based solutions that interact with databases on behalf of clients, dynamically generate custom content to be displayed by browsers and maintain unique session information for each client. The Java Servlet API allows developers to add functionality to Web servers for handling client requests. Servlets are also reusable across Web servers and across platforms. This chapter demonstrates the Web's request/response mechanism (primarily with HTTP get and post requests), redirecting requests to other resources and interacting with databases through JDBC. The chapter features a three-tier client/server application that tracks users' responses to a survey. For readers not familiar with developing Web pages, the CD includes PDFs of three chapters from our book *Internet & World Wide Web How to Program, Third Edition*—Introduction to XHTML: Part 1, Introduction to XHTML: Part 2 and Cascading Style Sheets (CSS). These chapters are also useful in Chapter 27.

Chapter 27—JavaServer Pages (JSP)—introduces an extension of servlet technology called JavaServer Pages (JSP). JSPs enable delivery of dynamically generated Web content, and are frequently used by Web designers and others who are not familiar with Java programming. JSPs may contain Java code in the form of scriptlets. Each JSP is compiled into a Java servlet—this normally occurs the first time a JSP is requested by a client. Subsequent client requests are fulfilled by the compiled servlet. This chapter features a three-tier client/server guest-book application that stores guest information in a database.

Chapter 28—Formatted Output (**New Chapter**)**—**presents the powerful formatting capabilities of `printf`, which is new to J2SE 5.0. We discuss `printf`'s capabilities such as rounding floating point values to a given number of decimal places, aligning columns of numbers, right justification and left justification, insertion of literal information, forcing a plus sign, printing leading zeros, using exponential notation, using octal and hexadecimal numbers, controlling field widths and precisions, and displaying dates and times in various formats. We also demonstrate formatting output with class `Formatter`.

Chapter 29—Strings, Characters and Regular Expressions—deals with processing words, sentences, characters and groups of characters. We present classes `String`, `StringBuffer`, `Character` and `StringTokenizer`. The chapter explores Java's API for regular expressions, which enables programs to search strings for sequences of characters that match specified patterns.

Appendix A—Operator Precedence Chart—lists each of the Java operators and indicates their relative precedence and associativity.

Appendix B—ASCII Character Set—lists the characters of the ASCII (American Standard Code for Information Interchange) character set and indicates the character code value for each. Java uses the Unicode character set with 16-bit characters for representing all of the characters in the world's "commercially significant" languages. Unicode includes ASCII as a subset.

Appendix C—Keywords and Reserved Words—lists all keywords and reserved words defined in the *Java Programming Language Specification*.

Appendix D—Primitive Types—lists all primitive types defined in the *Java Programming Language Specification*.

Appendix E (On CD)—Number Systems—discusses the binary (base 2), decimal (base 10), octal (base 8) and hexadecimal (base 16) number systems.

Appendix F (On CD)—Unicode®—discusses the Unicode character set, which enables Java to display information in many languages. The appendix provides a sample Java program that displays "Welcome to Unicode" in several languages—English, Chinese, Cyrillic, French, German, Japanese, Portuguese and Spanish.

Appendix G—Using the Java API Documentation—introduces the Java API documentation, which provides easy-to-use information on Java's built-in packages. **This appendix is important for Java novices.** You should search the Java API documentation regularly to learn more about the Java API classes used in this book and additional predefined classes—Java provides thousands of them that perform all kinds of useful tasks. Reusing these classes enables you to develop applications faster. The appendix discusses the information you can find, how this information is organized, and how to navigate the Java API documentation online, including viewing a package, a class and a method. The appendix also demonstrates how to find a particular package, class or method from the index page.

Appendix H (On CD)—Creating HTML Documentation with `javadoc`—introduces the `javadoc` documentation generation tool. Sun Microsystems uses the `javadoc` tool to produce the Java API documentation that is presented in Appendix G. The example takes the reader through the `javadoc` documentation process. First, we introduce the comment style and tags that `javadoc` recognizes and uses to create documentation. Next, we discuss the commands and options used to run the utility. Finally, we examine the source files `javadoc` uses and the HTML files `javadoc` creates.

Appendix I (On CD)—Bit Manipulation—discusses Java's bit-manipulation operators and class `Bitset`. It also includes an extensive discussion of bit-manipulation operators, such as bitwise AND, bitwise inclusive OR, bitwise exclusive OR, left shift, signed right shift, unsigned right shift and complement. The appendix also discusses class `BitSet`, which enables the creation of bit-array-like objects for setting and getting individual bit values.

Appendix J (On CD)—ATM Case Study Code—contains the implementation of our case study on object-oriented design with the UML. This appendix is discussed in the overview of the case study (presented shortly).

Appendix K (On CD)—Labeled `break` and `continue` Statements—introduces specialized versions of the `break` and `continue` statements presented in Chapter 5. These versions are for use with nested repetition statements.

Appendix L (On CD)—UML 2: Additional Diagram Types—Overviews the UML 2 diagram types that are not found in the OOD/UML Case Study.

Appendix M (On CD)—Design Patterns. In their book, *Design Patterns, Elements of Reusable Object-Oriented Software*, the "Gang of Four" (E. Gamma, R. Helm, R. Johnson, and J. Vlissides) described 23 design patterns that provide proven architectures for building object-oriented software systems. We briefly discuss 18 of their patterns—creational patterns address issues related to object creation; structural patterns organize classes and objects in a system; and behavioral patters offer strategies for modeling how objects collaborate with one another. We also discuss concurrency patterns, which are useful in multithreaded systems, and architectural patterns, which help designers assign functionality to subsystems. We mention how Java API packages take advantage of design patterns and explain how certain programs in this book use design patterns. Design patterns are used mostly in the J2EE (Java 2 Platform, Enterprise Edition) community, where systems tend to be large and complex.

Appendix N (On CD)—Using the Debugger—demonstrates key features of the J2SE Development Kit (JDK) version 5.0's built-in debugger, which allows a programmer to monitor the execution of applications to locate and remove logic errors. The appendix presents step-by-step instructions, so students learn how to use the debugger in a hands-on manner.

A Tour of the Optional Case Study on Object-Oriented Design with the UML

In this section we tour the book's optional case study of object-oriented design with the UML. This tour previews the contents of the nine "Software Engineering Case Study" sections (in Chapters 1–8 and 10). After completing this case study, the reader will be thoroughly familiar with an object-oriented design and implementation for a significant Java application.

The design presented in the ATM case study was developed at Deitel & Associates, Inc. and scrutinized by industry professionals. We crafted this design to meet the requirements of introductory course sequences. Surely real ATM systems used by banks around the world are based on more sophisticated designs that take into consideration many more issues than we have addressed here. Our primary goal throughout the design process, however, was to create a simple design that would be clear to OOD and UML novices, while still demonstrating key OOD concepts and the related UML modeling techniques.

Section 1.16—Software Engineering Case Study: Introduction to Object Technology and the UML—introduces the object-oriented design case study with the UML. The section introduces the basic concepts and terminology of object technology, including classes, objects, encapsulation, inheritance and polymorphism. We discuss the history of the UML. This is the only required section of the case study.

Section 2.9—(Optional) Software Engineering Case Study: Examining the Requirements—discusses a *requirements document* that specifies the requirements for a system that we will design and implement—the software for a simple automated teller machine (ATM). We investigate the structure and behavior of object-oriented systems in general. We discuss how the UML will facilitate the design process in subsequent "Software Engineering Case Study" sections by providing several additional types of diagrams to model our system. We include a list of URLs and book references on object-oriented design with the UML. We discuss the interaction between the ATM system specified by the requirements document and its user. Specifically, we investigate the scenarios that may occur between the user and the system itself—these are called *use cases*. We model these interactions, using *use case diagrams* of the UML.

Section 3.9—(Optional) Software Engineering Case Study: Identifying the Classes in the Requirements Documents—begins to design the ATM system. We identify its classes, or "building blocks," by extracting the nouns and noun phrases from the requirements document. We arrange these classes into a UML class diagram that describes the class structure of our simulation. The class diagram also describes relationships, known as *associations*, among classes.

Section 4.14—(Optional) Software Engineering Case Study: Identifying Class Attributes—focuses on the attributes of the classes discussed in Section 3.9. A class contains both *attributes* (data) and *operations* (behaviors). As we see in later sections, changes in an object's attributes often affect the object's behavior. To determine the attributes for the classes in our case study, we extract the adjectives describing the nouns and noun phrases (which defined our classes) from the requirements document, then place the attributes in the class diagram we created in Section 3.9.

Section 5.10—(Optional) Software Engineering Case Study: Identifying Objects' States and Activities—discusses how an object, at any given time, occupies a specific condition called a *state*. A *state transition* occurs when that object receives a message to change state. The UML provides the *state machine diagram*, which identifies the set of possible states that an object may occupy and models that object's state transitions. An object also has an *activity*—the work it performs in its lifetime. The UML provides the *activity diagram*—a flowchart that models an object's activity. In this section, we use both types of diagrams to begin modeling specific behavioral aspects of our ATM system, such as how the ATM carries out a withdrawal transaction and how the ATM responds when the user is authenticated.

Section 6.13—(Optional) Software Engineering Case Study: Identifying Class Operations—identifies the operations, or services, of our classes. We extract from the requirements document the verbs and verb phrases that specify the operations for each class. We then modify the class diagram of Section 3.9 to include each operation with its associated class. At this point in the case study, we will have gathered all information possible from the requirements document. However, as future chapters introduce such topics as inheritance, we will modify our classes and diagrams.

Section 7.10—(Optional) Software Engineering Case Study: Collaboration Among Objects—provides a "rough sketch" of the model for our ATM system. In this section, we see how it works. We investigate the behavior of the simulation by discussing *collaborations*—messages that objects send to each other to communicate. The class operations that we discovered in Section 6.13 turn out to be the collaborations among the objects in our system. We determine the collaborations, then collect them into a *communication diagram*—the UML diagram for modeling collaborations. This diagram reveals which objects collaborate and when. We present a communication diagram of the collaborations among objects to perform an ATM balance inquiry. We then present the UML *sequence diagram* for modeling interactions in a system. This diagram emphasizes the chronological ordering of messages. A sequence diagram models how objects in the system interact to carry out withdrawal and deposit transactions.

Section 8.19—(Optional) Software Engineering Case Study: Starting to Program the Classes of the ATM System—takes a break from designing the behavior of our system. We begin the implementation process to emphasize the material discussed in Chapter 8. Using the UML class diagram of Section 3.9 and the attributes and operations discussed in Section 4.14 and Section 6.13, we show how to implement a class in Java from a design. We do not implement all classes—because we have not completed the design process. Working from our UML diagrams, we create code for the `Withdrawal` class.

Section 10.8—(Optional) Software Engineering Case Study: Incorporating Inheritance into the ATM System—continues our discussion of object-oriented programming. We consider inheritance—classes sharing common characteristics may inherit attributes and operations from a "base" class. In this section, we investigate how our ATM system can benefit from using inheritance. We document our discoveries in a class diagram that models inheritance relationships—the UML refers to these relationships as *generalizations*. We modify the class diagram of Section 3.9 by using inheritance to group classes with similar characteristics. This section concludes the design of the model portion of our simulation. We implement this model as Java code in Appendix J.

Appendix J (On CD)—ATM Case Study Code—The majority of the case study involved designing the model (i.e., the data and logic) of the ATM system. In this appendix, we implement that model in Java. Using all the UML diagrams we created, we present the Java classes necessary to implement the model. We apply the concepts of object-oriented design with the UML and object-oriented programming in Java that you learned in the chapters. By the end of this appendix, students will have completed the design and implementation of a real-world system, and should now feel confident tackling larger systems, such as those that professional software engineers build.

Appendix L—UML 2 Additional Diagrams—Overviews the UML 2 diagram types not found in the OOD/UML Case Study.

A Tour of the Optional GUI and Graphics Case Study

In this section, we tour the book's optional 10-section case study on creating graphics and graphical user interfaces (GUIs) in Java. The goal of this case study is to create a polymorphic drawing application in which the user can select a shape to draw, select the characteristics of the shape (such as the color of the shape and whether it is filled with color or hollow) and use the mouse to draw the shape. This integrated case study builds gradually toward that goal, with the reader implementing polymorphic drawing in Chapter 10, adding an event-driven GUI in Chapter 11 and enhancing the drawing capabilities in Chapter 12 with Java 2D. This tour previews the topics covered in each section of the case study. After completing this case study, students will be able to create their own simple graphical applications.

Section 3.9—Using Dialog Boxes—introduces graphical user interfaces and demonstrates handling input and output with dialog boxes. We use predefined `JOptionPane` dialogs to display information and read text in an application.

Section 4.14—Creating Simple Drawings—introduces Java's graphics capabilities. First, we describe Java's coordinate system, then we cover drawing lines and creating a window that displays drawings.

Section 5.10—Rectangles and Ovals—explains how to use the graphics methods that draw rectangles and ovals.

Section 6.13—Colors and Filled Shapes—discusses how computers represent colors, how to use colors in graphics, and how to fill oval or rectangular regions with a solid color.

Section 7.13—Drawing Arcs—describes how Java specifies angles; then we demonstrate drawing arcs (i.e., sections of an oval) by defining an oval and angular positions along the oval.

Section 8.18—Using Objects with Graphics—describes how to use objects to represent shapes. We create classes to represent each shape type, store these objects in arrays and retrieve the shapes each time we need to draw them.

Section 9.8—Displaying Text and Images Using Labels—explores creating labels and attaching them to the application window. Applications use labels to display information to the user. Labels in Java can display text, an image or both.

Section 10.8—Drawing with Polymorphism—focuses on the common characteristics of the classes created in Section 8.18. In this section, we examine these similarities and redesign the individual shape classes so that they inherit their common functionality from a "base" class and can be processed polymorphically.

Exercise 11.18—Expanding the Interface—asks the reader to apply the event-driven GUI programming techniques they learn in Chapter 11. Readers create a GUI that allows users to select the shape to draw and its color, then draw the shape with a mouse. The application stores and manipulates shapes using the classes created in Section 10.8.

Exercise 12.31—Adding Java2D—asks the reader to use Java 2D's more elaborate two-dimensional drawing capabilities to create a robust drawing application in which users select a shape's line thickness and fill options (solid colors or gradients that transition between colors). To do this, the reader enhances the shape classes in Exercise 11.18 to store information about each shape's Java 2D features.

Software Included with *Java How to Program, 6/e*

A number of for-sale Java development tools are available, but you do not need any to get started with Java. We wrote *Java How to Program, 6/e* using only the new Java 2 Standard Edition Development Kit (JDK), version 5.0. The current JDK version can be downloaded from Sun's Java Web site `java.sun.com/j2se/downloads/index.html`. This site also contains the JDK documentation downloads.

The CD that accompanies *Java How to Program, 6/e*, contains several Java editors, including BlueJ Version 1.3.5, JCreator Lite Version 3.10 (Windows only), jEdit Version 4.1 and jGRASP Version 1.7.0. The CD also contains the NetBeans™ Version 3.6 Integrated Development Environment (IDE). Windows and Linux versions of MySQL® 4.0.20 and MySQL® Connector/J Version 3.0.14 are provided for the database processing performed in Chapters 25–27. Finally, Apache Tomcat Version 5.0.25 is provided for use with servlets and JavaServer Pages in Chapters 26–27. If you have questions about using this software, please read the documentation on the CD, or read our *Dive Into*™ Series publications, which are available with the resources for *Java How to Program, 6/e* at `www.deitel.com/books/downloads.html`. The free *Dive-Into*™ Series publications help students and instructors familiarize themselves with various Java development tools. These publications include: *Dive Into*™ *NetBeans*, *Dive Into*™ *Eclipse*, *Dive Into*™ *JBuilder*, *Dive Into*™ *jEdit*, *Dive Into*™ *jCreator*, *Dive Into*™ *jGRASP* and *Dive Into*™ *BlueJ*.

The CD also contains the book's examples and an HTML Web page with links to the Deitel & Associates, Inc. Web site and the Prentice Hall Web site. This Web page can be loaded to a Web browser to afford quick access to all the resources.

Teaching Resources for *Java How to Program, 6/e*

Java How to Program, 6/e, has extensive resources for instructors. The *Instructor's Resource CD (IRCD)* contains the *Solutions Manual* with solutions to the vast majority of the end-of-chapter exercises, a *Test Item File* of multiple-choice questions (approximately two per book section) and PowerPoint slides containing all the code and figures in the text, plus bulleted items that summarize the key points in the text. Instructors can customize the slides.

Prentice Hall's *Companion Web Site* (`www.prenhall.com/deitel`) for *Java How to Program, 6/e* offers resources for students and instructors. For instructors, the *Companion Web Site* offers a *Syllabus Manager*, which helps instructors plan courses interactively and create online syllabi.

Chapter-specific resources available for students on the *Companion Web Site* include:

- Chapter objectives
- Highlights (e.g., chapter summary)
- Outline
- Tips (e.g., *Common Programming Errors*, *Error-Prevention Tips, Good Programming Practices*, *Portability Tips*, *Performance Tips* and *Software Engineering Observations*)
- Online Study Guide—contains additional short-answer self-review exercises (e.g., true/false) with answers and provides immediate feedback to the student

Students can track their results and course performance on quizzes using the *Student Profile* feature, which records and manages all feedback and results from tests taken on the *Companion Web Site*. To access the *Companion Web Site*, visit `www.prenhall.com/deitel`.

Java in the Lab

Java in the Lab, Lab Manual to Accompany Java How to Program, 6/e (ISBN 0-13-149497-X) complements *Java How to Program, 6/e* and *Small Java How to Program 6/e* with hands-on lab assignments designed to reinforce students' understanding of lecture material. This lab manual is designed for closed laboratories—regularly scheduled classes supervised by an instructor. Closed laboratories provide an excellent learning environment, because students can use concepts presented in class to solve carefully designed lab problems. Instructors are better able to gauge the students' understanding of the material by monitoring the students' progress in lab. This lab manual also can be used for open laboratories, homework and for self-study. *Java How to Program, 6/e*, and the lab manual are available individually or together in a value pack (ISBN 0-13-167380-7).

Every chapter in the lab manual is divided into *Prelab Activities*, *Lab Exercises* and *Postlab Activities*. Each chapter contains objectives that introduce the lab's key topics and an assignment checklist that allows students to mark which exercises the instructor has assigned. The lab manual pages are perforated, so students can submit their answers (if required).

Solutions to the lab manual's *Prelab Activities*, *Lab Exercises* and *Postlab Activities* are available in electronic form. Instructors can obtain these materials from their regular Prentice Hall representatives; the solutions are not available to students.

Prelab Activities

Prelab Activities are intended to be completed by students after studying each chapter of *Java How to Program, 6/e*. *Prelab Activities* test students' understanding of the textbook material and prepare students for the programming exercises in the lab session. The exercises focus on important terminology and programming concepts and are effective for self-review. *Prelab Activities* include *Matching Exercises*, *Fill-in-the-Blank Exercises*, *Short-Answer Questions*, *Programming-Output Exercises* (determine what short code segments do without actually running the program) and *Correct-the-Code Exercises* (identify and correct all errors in short code segments).

Lab Exercises

The most important section in each chapter is the *Lab Exercises*. These teach students how to apply the material learned in *Java How to Program, 6/e*, and prepare them for writing Java programs. Each lab contains one or more lab exercises and a debugging problem. The *Lab Exercises* contain the following:

- *Lab Objectives* highlight specific concepts on which the lab exercise focuses.
- *Problem Descriptions* provide the details of the exercise and hints to help students implement the program.
- *Sample Outputs* illustrate the desired program behavior, which further clarifies the problem descriptions and aids the students with writing programs.
- *Program Templates* take complete Java programs and replace key lines of code with comments describing the missing code.
- *Problem-Solving Tips* highlight key issues that students need to consider when solving the lab exercises.
- *Follow-Up Questions and Activities* ask students to modify solutions to lab exercises, write new programs that are similar to their lab-exercise solutions or explain the implementation choices that were made when solving lab exercises.
- *Debugging Problems* consist of blocks of code that contain syntax errors and/or logic errors. These alert students to the types of errors they are likely to encounter while programming.

Postlab Activities

Professors typically assign *Postlab Activities* to reinforce key concepts or to provide students with more programming experience outside the lab. *Postlab Activities* test the students' understanding of the *Prelab* and *Lab Exercise* material, and ask students to apply their knowledge to creating programs from scratch. The section provides two types of programming activities: coding exercises and programming challenges. Coding exercises are short and serve as review after the *Prelab Activities* and *Lab Exercises* have been completed. The coding exercises ask students to write programs or program segments using key concepts from the textbook. *Programming Challenges* allow students to apply their knowledge to substantial programming exercises. Hints, sample outputs and pseudocode are provided to aid students with these problems. Students who successfully complete the *Programming Challenges* for a chapter have mastered the chapter material. Answers to the programming challenges are available at `www.deitel.com/books/downloads.html`.

OneKey, CourseCompass^{SM}, WebCT™ and by Blackboard™

OneKey is Prentice Hall's exclusive new resource that gives instructors and students access to the best online teaching and learning tools through one convenient Web site. OneKey enables instructors to prepare their courses effectively, present their courses more dramatically and assess students easily. An abundance of searchable presentation material together with practice activities and test questions—all organized by chapter or topic—helps to simplify course preparation.

Selected content from the Deitels' introductory programming language *How to Program* series, including *Java How to Program, 6/e*, is available to integrate into various popular course management systems, including CourseCompass, Blackboard and WebCT. Course management systems help faculty create, manage and use sophisticated Web-based educational tools and programs. Instructors can save hours of inputting data by using Deitel course-management-systems content. [*Note:* The e-Book included with OneKey contains the entire text of *Java How to Program, 6/e*.]

Blackboard, CourseCompass and WebCT offer:

- **Features to create and customize an online course**, such as areas to post course information (e.g., policies, syllabi, announcements, assignments, grades, performance evaluations and progress tracking), class and student management tools, a gradebook, reporting tools, page tracking, a calendar and assignments.
- **Communication tools** to help create and maintain interpersonal relationships between students and instructors, including chat rooms, whiteboards, document sharing, bulletin boards and private e-mail.
- **Flexible testing tools** that allow an instructor to create online quizzes and tests from questions directly linked to the text, and that grade and track results effectively. All tests can be inputted into the gradebook for efficient course management. WebCT also allows instructors to administer timed online quizzes.
- **Support materials** for instructors are available in print and online formats.

In addition to the types of tools found in Blackboard and WebCT, CourseCompass from Prentice Hall includes:

- **CourseCompass course home page**, which makes the course as easy to navigate as a book. An expandable table of contents allows instructors to view course content at a glance and to link to any section.
- **Hosting on Prentice Hall's centralized servers**, which allows course administrators to avoid separate licensing fees or server-space issues. Access to Prentice Hall technical support is available.
- **"How Do I" online-support sections** are available for users who need help personalizing course sites, including step-by-step instructions for adding PowerPoint slides, video and more.
- **Instructor Quick Start Guide** helps instructors create online courses using a simple, step-by-step process.

To view free online demonstrations and learn more about these Course Management Systems, which support Deitel content, visit the following Web sites:

- Blackboard: `www.blackboard.com` and `www.prenhall.com/blackboard`

- WebCT: `www.webct.com` and `www.prenhall.com/webct`
- CourseCompass: `www.coursecompass.com` and `www.prenhall.com/coursecompass`

Java 2 Multimedia Cyber Classroom, 6/e Through OneKey

Java How to Program, 6/e and *Small Java How to Program, 6/e* will now include a free, Web-based interactive multimedia accompaniment to the book—*The Java 2 Multimedia Cyber Classroom, 6/e*—available in spring 2005 through OneKey. Our Web-based *Cyber Classroom* will include audio walkthroughs of code examples in the text, solutions to about half of the exercises in the book and more. For more information about the new Web-based *Cyber Classroom* and its availability through OneKey, please visit our Web site at `www.deitel.com` or sign up for the free *Deitel® Buzz Online* e-mail newsletter at `www.deitel.com/newsletter/subscribe.html`.

Students who use our *Cyber Classrooms* tell us that they like the interactivity and that the *Cyber Classroom* is a powerful reference tool. Professors tell us that their students enjoy using the *Cyber Classroom* and consequently spend more time on the courses, mastering more of the material than in textbook-only courses. For a complete list of our current CD-ROM-based *Cyber Classrooms*, see the *Deitel® Series* page at the beginning of this book, the product listing and ordering information at the end of this book, or visit `www.deitel.com`, `www.prenhall.com/deitel` or `www.InformIT.com/deitel`.

PearsonChoices

Today's students have increasing demands on their time and money, and they need to be resourceful about how, when and where they study. Pearson/Prentice Hall, a division of Pearson Education, has responded to that need by creating PearsonChoices to allow faculty and students to choose from a variety of textbook formats and prices.

Small Java How to Program 6/e

Small Java How to Program, 6/e is our new 10-chapter alternative print edition to *Java How to Program, 6/e*. *Small Java How to Program, 6/e* is focused on first-semester Computer Science (CS1) programming courses and is priced lower than our 29-chapter *Java How to Program, 6/e* and other competing texts in the CS1 market. The following is a chapter-level table of contents for the book. [*Note*: This book includes Chapters 1–10 of *Java How to Program, 6/e* and their corresponding optional GUI and graphics case study sections, but does not include the OOD/UML optional automated teller machine (ATM) case study.]

Chapters in Both Small Java How to Program, 6/e _and_ Java How to Program, 6/e

Chapter 1—Introduction to Computers, the Internet and the Web
Chapter 2—Introduction to Java Applications
Chapter 3—Introduction to Classes and Objects
Chapter 4—Control Statements: Part 1
Chapter 5—Control Statements Part 2
Chapter 6—Methods: A Deeper Look
Chapter 7—Arrays
Chapter 8—Classes and Objects: A Deeper Look
Chapter 9—Object-Oriented Programming: Inheritance
Chapter 10—Object-Oriented Programming: Polymorphism

SafariX WebBooks

SafariX Textbooks Online is a new service for college students looking to save money on required or recommended textbooks for academic courses. This secure WebBooks platform creates a new option in the higher education market; an additional choice for students alongside conventional textbooks and online learning services. Pearson provides students with a WebBook at 50% of the cost of its conventional print equivalent.

SafariX WebBooks are viewed through a Web browser connected to the Internet. No special plug-ins are required and no applications need to be downloaded. Students simply log in, purchase access and begin studying. With SafariX Textbooks Online students can search the text, make notes online, print out reading assignments that incorporate their professors' lecture notes and bookmark important passages they want to review later. They can navigate easily to a page number, reading assignment, or chapter. The Table of Contents of each WebBook appears in the left hand column alongside the text.

We are pleased to offer students the *Java How to Program, 6/e* SafariX WebBook available for January 2005 classes. Visit `www.pearsonchoices.com` for more information. Other Deitel titles available as SafariX WebBooks include *Small Java How to Program, 6/e* and *Simply C++: An Application-Driven Tutorial Approach*. Visit `www.safarix.com/tour.html` for more information.

Computer Science AP Courses

Java How to Program, 6/e, is a suitable textbook for teaching AP Computer-Science classes and for preparing students to take the corresponding exams. While writing this book, we carefully reviewed the syllabi for the AP Computer Science A and AB exams, to ensure that *Java How to Program, 6/e,* covers the vast majority of the information required for the exams. Instructors and students interested in preparing for these exams should visit:

```
www.deitel.com/books/jHTP6/Java_AP_Exam.html
```

This site is dedicated to the use of *Java How to Program, 6/e*, in the Computer Science AP curriculum. We will update this site regularly with additional information about the exams. For detailed information on the Computer Science AP curriculum, please visit

```
apcentral.collegeboard.com
```

DEITEL® Buzz Online Free E-mail Newsletter

Our free e-mail newsletter, the *Deitel® Buzz Online*, includes commentary on industry trends and developments, links to free articles and resources from our published books and upcoming publications, product-release schedules, errata, challenges, anecdotes, information on our corporate instructor-led training courses and more. It's also our way to notify our readers proactively about issues related to *Java How to Program, 6/e*. To subscribe, visit

```
www.deitel.com/newsletter/subscribe.html
```

Acknowledgments

One of the great pleasures of writing a textbook is acknowledging the efforts of many people whose names may not appear on the cover, but whose hard work, cooperation, friendship and understanding were crucial to the production of the book. Many people at Deitel & Associates, Inc. devoted long hours to this project.

- Andrew B. Goldberg is a recent graduate of Amherst College, where he earned a degree in Computer Science. Andrew is a co-author with us of *Internet and World Wide Web How to Program, 3/e* and contributed to *Operating Systems, Third Edition*. Andrew's contributions to *Java How to Program, 6/e* included updating Chapters 3–10 based on the new early-classes presentation of the book and other content revisions. He co-designed and co-authored the new, optional OOD/UML ATM case study. He updated Appendix M, Design Patterns and co-authored Appendix L, UML 2: Additional Diagram Types and Appendix N, Using the Debugger.
- Jeff Listfield is a Computer Science graduate of Harvard College. Jeff co-authored *C# How to Program*, *C# A Programmer's Introduction*, *C# for Experienced Programmers* and *Simply Java Programming*. He has also contributed to *Perl How to Program* and *Operating Systems, Third Edition*. Jeff contributed to Chapter 11, GUI: Part 1; Chapter 16, Searching and Sorting; Chapter 22, GUI: Part 2; Chapter 23, Multithreading; Chapter 24, Networking; and Chapter 29, Strings, Characters and Regular Expressions.
- Su Zhang holds B.Sc. and a M.Sc. degrees in Computer Science from McGill University. She is co-author with us of *Java Web Services for Experienced Programmers* and *Simply Java Programming*. She has also contributed to other Deitel publications, including *Advanced Java 2 Platform How to Program*, *Python How to Program* and *Operating Systems, Third Edition*. Su contributed to the sections on the new features of J2SE 5.0 in Chapters 7 and 8. She contributed to Chapter 18, Generics; Chapter 19, Collections; Chapter 25, Manipulating Database with JDBC; Chapter 26, Servlets; Chapter 27, JavaServer Pages; Chapter 28, Formatted Output; Appendix F, Unicode®; Appendix G, Using Java API Documentation; Appendix H, Creating HTML Documentation with `javadoc`; and Appendix I, Bit Manipulation. Su also converted much of the code from *Java How to Program, 5/e* to J2SE 5.0.
- Cheryl Yaeger graduated from Boston University in three years with a bachelor's degree in Computer Science. Cheryl has co-authored various Deitel & Associates, Inc. publications, including *C# How to Program*, *C#: A Programmer's Introduction*, *C# for Experienced Programmers*, *Visual Basic .NET for Experienced Programmers*, *Simply Visual Basic .NET 2003* and *Simply C#*. Cheryl has also contributed to other Deitel & Associates publications including *Perl How to Program*, *Wireless Internet and Mobile Business How to Program*, *Internet and World Wide Web How to Program, 3/e*, *Visual Basic .NET How to Program, 2/e* and *Simply Java Programming*. Cheryl contributed to Chapter 12, Graphics; Chapter 13, Exceptions; Chapter 14, Files and Streams; Chapter 15, Recursion; and Chapter 21, Multimedia.
- Jing Hu, a participant in the Deitel & Associates, Inc. Summer Internship Program, is a sophomore at Cornell University studying Computer Science. He contributed to the new optional GUI and Graphics Case Study and the instructor's manual for Chapters 3–8. He also contributed to the sections on preconditions/postconditions, assertions, invariants, and the discussion of video in Chapter 21, Multimedia.

- Sin Han Lo, a participant in the Deitel & Associates, Inc. Summer Internship Program in 2003, is a recent graduate of Wellesley College where she studied Computer Science and Economics. She contributed to Chapter 15, Recursion, and designed the fractal example in that chapter.
- John Paul Casiello, a participant in the Deitel & Associates, Inc. Summer Internship Program in 2003, is a Computer Science student at Northeastern University. He contributed to Chapter 16, Searching and Sorting.
- Barbara Deitel, Chief Financial Officer at Deitel & Associates, Inc. researched the quotes at the beginning of each chapter and applied copyedits to the book.
- Abbey Deitel, President of Deitel & Associates, Inc., is an Industrial Management graduate of Carnegie Mellon University. She co-authored the Preface. She also suggested the theme and bug names for the cover of the book.
- Christi Kelsey, a Management Information Systems graduate of Purdue University, contributed to Chapter 1, the Preface and Appendix N, Using the Debugger. She edited the Index and paged the entire manuscript.

We are fortunate to have worked on this project with the talented and dedicated team of publishing professionals at Prentice Hall. We especially appreciate the extraordinary efforts of our Computer Science Editor, Kate Hargett and her boss and our mentor in publishing—Marcia Horton, Editorial Director of Prentice Hall's Engineering and Computer Science Division. Jennifer Cappello did an extraordinary job recruiting the review team and managing the review process. Vince O'Brien, Tom Manshreck and John Lovell did a marvelous job managing the production of the book. The talents of Carole Anson, Paul Belfanti and Geoffrey Cassar are evident in the re-design of the book's interior and the new cover art, and Sarah Parker managed the publication of the book's extensive ancillary package.

We sincerely appreciate the efforts of our fifth edition post-publication reviewers and our sixth edition reviewers:

Academic Reviewers
Karen Arlien, Bismarck State College
Ben Blake, Cleveland State University
Walt Bunch, Chapman University
Marita Ellixson, Eglin AFB/University of Arkansas
Ephrem Eyob, Virginia State University
Bjorn Foss, Florida Metropolitan University
Bill Freitas, The Lawrenceville School
Joe Kasprzyk, Salem State College
Brian Larson, Modesto Junior College
Roberto Lopez-Herrejon, University of Texas at Austin
Dean Mellas,Cerritos College
David Messier, Eastern University
Andy Novobilski, University of Tennessee, Chattanooga
Richard Ord, University of California, San Diego
Gavin Osborne, Saskatchewan Institute of Applied Science & Technology
Donna Reese, Mississippi State University
Craig Slinkman, University of Texas at Arlington

Sreedhar Thota, Western Iowa Tech Community College
Mahendran Velauthapillai, Georgetown University
Loran Walker, Lawrence Technological University
Stephen Weiss, University of North Carolina at Chapel Hill

Industry Reviewers
Butch Anton, Wi-Tech Consulting
Jonathan Bruce, Sun Microsystems, Inc. (JCP Specification Lead for JDBC)
Gilad Bracha, Sun Microsystems, Inc. (JCP Specification Lead for Generics)
Michael Develle, Independent Consultant
Jonathan Gadzik, Independent Consultant
Brian Goetz, Quiotix Corporation (JCP Concurrency Utilities Specification Expert Group Member)
Anne Horton, AT&T Bell Laboratories
James Huddleston, Independent Consultant
Peter Jones, Sun Microsystems, Inc.
Doug Kohlert, Sun Microsystems, Inc.
Earl LaBatt, Altaworks Corp./ University of New Hampshire
Paul Monday, Sun Microsystems, Inc.
Bill O'Farrell, IBM
Cameron Skinner, Embarcadero Technologies, Inc.
Brandon Taylor, Sun Microsystems, Inc.
Karen Tegtmeyer, Independent Consultant

OOD/UML Optional Case Study Reviewers
Sinan Si Alhir, Independent Consultant
Gene Ames, Star HRG
Jan Bergandy, University of Massachusetts at Dartmouth
Marita Ellixson, Eglin AFB/University of Arkansas
Jonathan Gadzik, Independent Consultant
Thomas Harder, ITT ESI, Inc.
James Huddleston, Independent Consultant
Terrell Hull, Independent Consultant
Kenneth Hussey, IBM
Joe Kasprzyk, Salem State College
Dan McCracken, City College of New York
Paul Monday, Sun Microsystems, Inc.
Davyd Norris, Rational Software
Cameron Skinner, Embarcadero Technologies, Inc.
Craig Slinkman, University of Texas at Arlington
Steve Tockey, Construx Software

Under a tight time schedule, they scrutinized every aspect of the text and made countless suggestions for improving the accuracy and completeness of the presentation.

Well, there you have it! Java is a powerful programming language that will help you write programs quickly and effectively. Java scales nicely into the realm of enterprise-systems development to help organizations build their critical information systems. As you read the book, we would sincerely appreciate your comments, criticisms, corrections and suggestions for improving the text. Please address all correspondence to:

```
deitel@deitel.com
```

We will respond promptly, and we will post corrections and clarifications on our Web site:

```
www.deitel.com
```

We hope you enjoy learning with *Java How to Program, Sixth Edition* as much as we enjoyed writing it!

Dr. Harvey M. Deitel
Paul J. Deitel

About the Authors

Dr. Harvey M. Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 43 years experience in the computing field, including extensive industry and academic experience. Dr. Deitel earned B.S. and M.S. degrees from the Massachusetts Institute of Technology and a Ph.D. from Boston University. He worked on the pioneering virtual-memory operating-systems projects at IBM and MIT that developed techniques now widely implemented in systems such as UNIX, Linux and Windows XP. He has 20 years of college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., with his son, Paul J. Deitel. He and Paul are the co-authors of several dozen books and multimedia packages and they are writing many more. With translations published in Japanese, German, Russian, Spanish, Traditional Chinese, Simplified Chinese, Korean, French, Polish, Italian, Portuguese, Greek, Urdu and Turkish, the Deitels' texts have earned international recognition. Dr. Deitel has delivered hundreds of professional seminars to major corporations, academic institutions, government organizations and the military.

Paul J. Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of the MIT's Sloan School of Management, where he studied Information Technology. Through Deitel & Associates, Inc., he has delivered Java, C, C++, Internet and World Wide Web courses to industry clients, including IBM, Sun Microsystems, Dell, Lucent Technologies, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, Compaq, White Sands Missile Range, Rogue Wave Software, Boeing, Stratus, Cambridge Technology Partners, Open Environment Corporation, One Wave, Hyperion Software, Adra Systems, Entergy, CableData Systems and many other organizations. Paul is one of the most experienced Java corporate trainers having taught about 100 professional Java training courses. He has also lectured on C++ and Java for the Boston Chapter of the Association for Computing Machinery. He and his father, Dr. Harvey M. Deitel, are the world's best-selling Computer Science textbook authors.

About Deitel & Associates, Inc.

Deitel & Associates, Inc., is an internationally recognized corporate training and content-creation organization specializing in computer programming languages, Internet/World Wide Web software technology and object technology education. The company provides instructor-led courses on major programming languages and platforms, such as Java, Advanced Java, C, C++, .NET programming languages, XML, Perl, Python; object technology; and Internet and World Wide Web programming. The founders of Deitel & Associates, Inc., are Dr. Harvey M. Deitel and Paul J. Deitel. The company's clients include many of the world's largest computer companies, government agencies, branches of the military and business organizations. Through its 28-year publishing partnership with Prentice Hall, Deitel & Associates, Inc. publishes leading-edge programming textbooks, professional books, interactive multimedia *Cyber Classrooms*, *Complete Training Courses*, Web-based training courses and course management systems e-content for popular CMSs such as WebCT, Blackboard and Pearson's CourseCompass. Deitel & Associates, Inc., and the authors can be reached via e-mail at:

`deitel@deitel.com`

To learn more about Deitel & Associates, Inc., its publications and its worldwide *DIVE INTO*™ Series Corporate Training curriculum, see the last few pages of this book or visit:

`www.deitel.com`

and subscribe to the free *Deitel*® *Buzz Online* e-mail newsletter at:

`www.deitel.com/newsletter/subscribe.html`

Individuals wishing to purchase Deitel books, Cyber Classrooms, Complete Training Courses and Web-based training courses can do so through:

`www.deitel.com/books/index.html`

Bulk orders by corporations and academic institutions should be placed directly with Prentice Hall. See the last few pages of this book for worldwide ordering details.