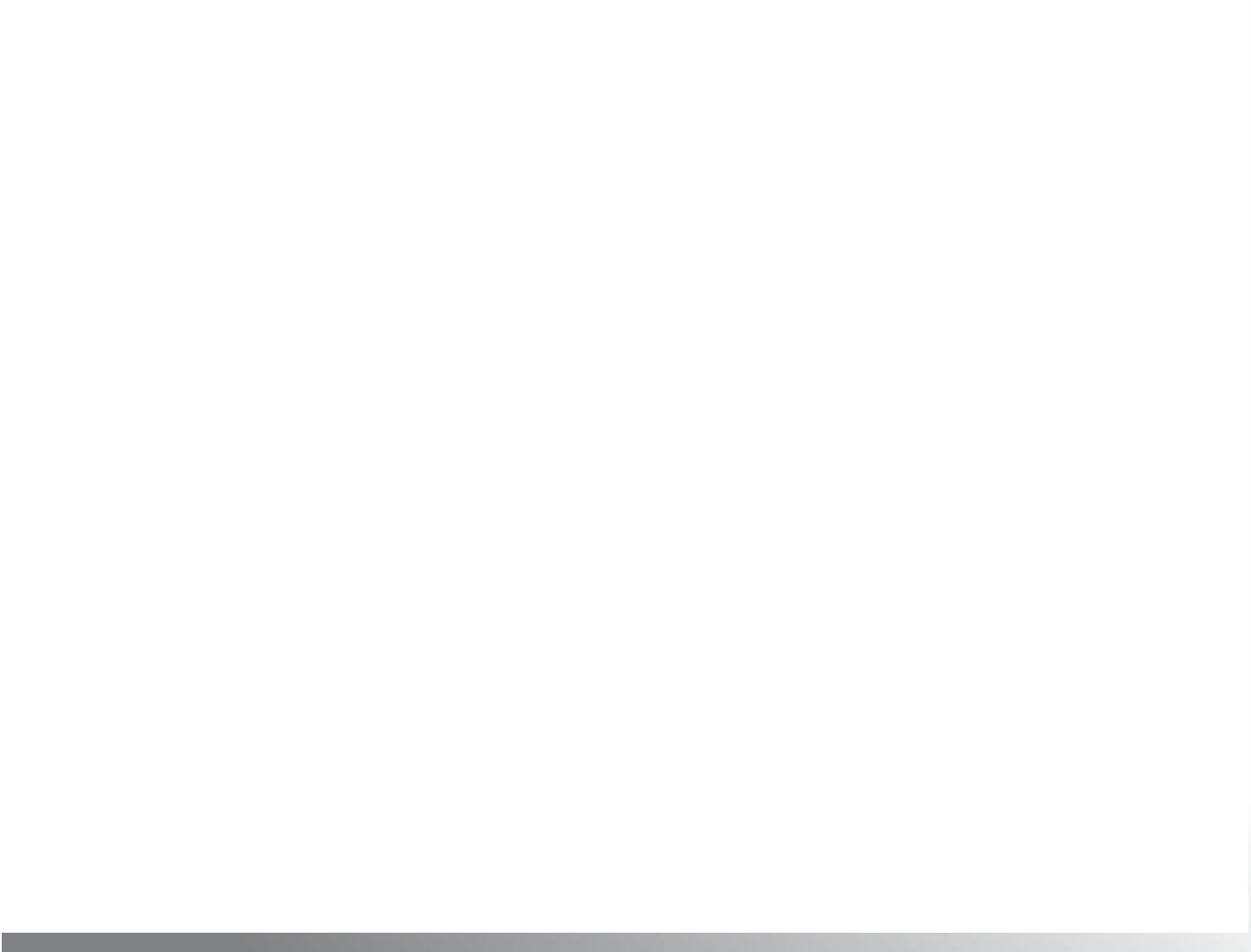


**7/e**

**Cay Horstmann**

**Big Java**

**Early Objects**



**7/e**

**Early Objects**

**Big Java**

**Cay Horstmann**

**San Jose State University**

VICE PRESIDENT AND EXECUTIVE PUBLISHER Laurie Rosatone

EXECUTIVE EDITOR Joanna Dingle PROJECT MANAGER/DEVELOPMENT EDITOR Cindy Johnson

EDITORIAL ASSISTANT Crystal Franks

LEAD PRODUCT DESIGNER Tom Kulesa

MARKETING MANAGER Michael MacDougald

PRODUCTION MANAGER Nichole Urban

PRODUCTION MANAGER Nicole Repasky

PRODUCTION MANAGEMENT SERVICES Cindy Johnson, Publishing Services PHOTO EDITOR Anindita Adiyal

COVER DESIGNER Joanna Vieira

COVER PHOTOS (tiger) © ArtMediaFactory/Shutterstock; (rhino) © GUDKOV ANDREY/Shutterstock; (bird) © Jeremy Woodhouse/Holly Wilmeth/ Getty Images; (tree frog) © kuritafsheen/Getty Images.

This book was set in 10.5/12 Stempel Garamond LT Std by Publishing Services, and printed and bound by Quad Graphics/Versailles. The cover was printed by Quad Graphics/Versailles.

Founded in 1807, John Wiley & Sons, Inc. has been a valued source of knowledge and understanding for more than 200 years, helping people around the world meet their needs and fulfill their aspirations. Our company is built on a foundation of principles that include responsibility to the communities we serve and where we live and work. In 2008, we launched a Corporate Citizenship Initiative, a global effort to address the environmental, social, economic, and ethical challenges we face in our business. Among the issues we are addressing are carbon impact, paper specifications and procurement, ethical conduct within our business and among our vendors, and community and charitable support. For more information, please visit our website: [www.wiley.com/go/](http://www.wiley.com/go/) citizenship.

This book is printed on acid-free paper. ∞

Copyright © 2019, 2016 John Wiley & Sons, Inc. All rights reserved. No part of this publication may be repro- duced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photo- copying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Dan- vers, MA 01923, (978) 750-8400, fax (978) 646-8600, or on the Web at [www.copyright.com.](http://www.copyright.com/) Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, (201) 748-6011, fax (201) 748-6008.

Evaluation copies are provided to qualified academics and professionals for review purposes only, for use in their courses during the next academic year. These copies are licensed and may not be sold or transferred to a third party. Upon completion of the review period, please return the evaluation copy to Wiley. Return instruc- tions and a free of charge return shipping label are available at: [www.wiley.com/go/returnlabel.](http://www.wiley.com/go/returnlabel) If you have chosen to adopt this textbook for use in your course, please accept this book as your complimentary desk copy. Outside of the United States, please contact your local representative.

ePUB ISBN 978-1-119-49909-1

Printed in the United States of America.

The inside back cover will contain printing identification and country of origin if omitted from this page. In addition, if the ISBN on the back cover differs from the ISBN on this page, the one on the back cover is correct.

10 9 8 7 6 5 4 3 2 1

This book is an introduction to Java and computer programming that focuses on the essentials—and on effective learning. The book is designed to serve a wide range of student interests and abilities and is suitable for a first course in programming for computer scientists, engineers, and students in other disciplines. No prior program- ming experience is required, and only a modest amount of high school algebra is needed.

Here are the key features of this book:

**Start objects early, teach object orientation gradually.**

In Chapter 2, students learn how to use objects and classes from the standard library. Chapter 3 shows the mechanics of implementing classes from a given specification. Students then use simple objects as they master branches, loops, and arrays. Object- oriented design starts in Chapter 8. This gradual approach allows students to use objects throughout their study of the core algorithmic topics, without teaching bad habits that must be un-learned later.

**Guidance and worked examples help students succeed.**

Beginning programmers often ask “How do I start? Now what do I do?” Of course, an activity as complex as programming cannot be reduced to cookbook-style instruc- tions. However, step-by-step guidance is immensely helpful for building confidence and providing an outline for the task at hand. “How To” guides help students with common programming tasks. Numerous Worked Examples demonstrate how to apply chapter concepts to interesting problems.

**Problem solving strategies are made explicit.**

Practical, step-by-step illustrations of techniques help students devise and evaluate solutions to programming problems. Introduced where they are most relevant, these strategies address barriers to success for many students. Strategies included are:

* Algorithm Design (with pseudocode)
* Tracing Objects
* First Do It By Hand (doing sample calculations by hand)
* Flowcharts
* Selecting Test Cases
* Hand-Tracing
* Storyboards

**Practice makes perfect.**

* Solve a Simpler Problem First
* Adapting Algorithms
* Discovering Algorithms by Manipulating Physical Objects
* Patterns for Object Data
* Thinking Recursively
* Estimating the Running Time of an Algorithm

Of course, programming students need to be able to implement nontrivial programs, but they first need to have the confidence that they can succeed. Each section con- tains numerous exercises that ask students to carry out progressively more complex tasks: trace code and understand its effects, produce program snippets from prepared parts, and complete simple programs. Additional review and programming problems are provided at the end of each chapter.

PREFACE

**iii**

**A visual approach motivates the reader and eases navigation.**

Photographs present visual analogies that explain the nature and behavior of computer concepts. Step-by- step figures illustrate complex program operations. Syntax boxes and example tables present a variety of typical and special cases in a compact format. It is easy to get the “lay of the land” by browsing the visuals, before focusing on the textual material.

© Terraxplorer/iStockphoto.

**Focus on the essentials while being**

**technically accurate.**

An encyclopedic coverage is not helpful for a begin- ning programmer, but neither is the opposite—

*Visual features help the reader with navigation.*

reducing the material to a list of simplistic bullet points. In this book, the essentials are presented in digestible chunks, with separate notes that go deeper into good practices or language features when the reader is ready for the additional information. You will not find artificial over-simplifications that give an illusion of knowledge.

**Reinforce sound engineering practices.**

A multitude of useful tips on software quality and common errors encourage the development of good programming habits. The optional testing track focuses on test-driven development, encouraging students to test their programs systematically.

**Provide an optional graphics track.**

Graphical shapes are splendid examples of objects. Many students enjoy writing pro- grams that create drawings or use graphical user interfaces. If desired, these topics can be integrated into the course by using the materials at the end of Chapters 2, 3, and 10.

**Engage with optional science and business exercises.**

End-of-chapter exercises are enhanced with problems from scientific and business domains. Designed to engage students, the exercises illustrate the value of program- ming in applied fields.

# New to This Edition

## Adapted to Java Versions 8 Through 11

This edition takes advantage of modern Java features when they are pedagogically sensible. I continue to use “pure” interfaces with only abstract methods. Default, static, and private interface methods are introduced in a Special Topic. Lambda expressions are optional for user interface callback, but they are used in the chapter on the stream library and its applications for “big data” processing.

The “diamond” syntax for generic classes is introduced as a Special Topic in Chap- ter 7 and used systematically starting with Chapter 15. Local type inference with the var keyword is described in a Special Topic.

Useful features such as the try-with-resources statement are integrated into the text. Chapter 21 covers the utilities provided by the Paths and Files classes.

## Interactive Learning

With this edition, interactive content is front and center. Immersive activities integrate with this text and engage students in activities designed to foster in-depth learning.

Students don’t just watch animations and code traces, they work on generating them. Live code samples invite the reader to experiment and to learn programming constructs first hand. The activities provide instant feedback to show students what they did right and where they need to study more.

# A Tour of the Book

The book can be naturally grouped into four parts, as illustrated by Figure 1 on page vi. The organization of chapters offers the same flexibility as the previous edition; dependencies among the chapters are also shown in the figure.

## Part A: Fundamentals (Chapters 1–7)

Chapter 1 contains a brief introduction to computer science and Java programming. Chapter 2 shows how to manipulate objects of predefined classes. In Chapter 3, you will build your own simple classes from given specifications. Fundamental data types, branches, loops, and arrays are covered in Chapters 4–7.

## Part B: Object-Oriented Design (Chapters 8–12)

Chapter 8 takes up the subject of class design in a systematic fashion, and it intro- duces a very simple subset of the UML notation. Chapter 9 covers inheritance and polymorphism, whereas Chapter 10 covers interfaces. Exception handling and basic file input/output are covered in Chapter 11. The exception hierarchy gives a useful example for inheritance. Chapter 12 contains an introduction to object-oriented design, including two significant case studies.

## Part C: Data Structures and Algorithms (Chapters 13–19)

Chapters 13 through 19 contain an introduction to algorithms and data structures, covering recursion, sorting and searching, linked lists, binary trees, and hash tables. These topics may be outside the scope of a one-semester course, but can be covered as desired after Chapter 7 (see Figure 1). Recursion, in Chapter 13, starts with simple examples and progresses to meaningful applications that would be difficult to imple- ment iteratively. Chapter 14 covers quadratic sorting algorithms as well as merge sort, with an informal introduction to big-Oh notation. Each data structure is presented in the context of the standard Java collections library. You will learn the essential abstractions of the standard library (such as iterators, sets, and maps) as well as the performance characteristics of the various collections. Chapter 18 introduces Java generics. This chapter is suitable for advanced students who want to implement their own generic classes and methods. Finally, Chapter 19 introduces the Java 8 streams library and shows how it can be used to analyze complex real-world data.

## Part D: Applied Topics (Chapters 20–25)

Chapters 20 through 25 cover Java programming techniques that definitely go beyond a first course in Java (21–25 are in the eText). Although, as already mentioned, a comprehensive coverage of the Java library would span many volumes, many instructors prefer that a textbook should give students additional reference material valuable beyond their first course. Some institutions also teach a second-semester course that covers more practical programming aspects such as database and network

6. Iteration

16. Basic Data Structures

17. Tree Structures

18. Generic Classes

programming, rather than the more traditional in-depth material on data structures and algorithms. This book can be used in a two-semester course to give students an introduction to programming fundamentals and broad coverage of applications. Alternatively, the material in the final chapters can be useful for student projects. The applied topics include graphical user-interface design, advanced file handling, multi- threading, and those technologies that are of particular interest to server-side pro- gramming: networking, databases, and XML. The Internet has made it possible to

1. Introduction

Fundamentals

e

2. Using Objects

Object-Oriented Design

Data Structures & Algorithms Applied Topics

3. Implementing Classes

eText Chapters

4. Fundamental Data Types

5. Decisions

Sections 11.1 and 11.2 (text 1le processing) can be covered with Chapter 6.

21. Advanced Input/Output e

23. Internet Networking e

24. Relational Databases

e

15. The Java Collections Framework

22.

Multithreading e

25. XML

e

10. Interfaces

9. Inheritance

14. Sorting and Searching

13. Recursion

8. Designing Classes

11. Input/Output and Exception

Handling

7. Arrays and Array Lists

6. Loops

12. Object- Oriented Design

20. Graphical User Interfaces

19. Stream Processing

**Figure 1** Chapter Dependencies

deploy many useful applications on servers, often accessed by nothing more than a browser. This server-centric approach to application development was in part made possible by the Java language and libraries, and today, much of the industrial use of Java is in server-side programming.

## Appendices

Many instructors find it highly beneficial to require a consistent style for all assign- ments. If the style guide in Appendix E conflicts with instructor sentiment or local customs, however, it is available in electronic form so that it can be modified. Appen- dices F–J are available in the eText.

* 1. The Basic Latin and Latin-1 Subsets of Unicode
  2. Java Operator Summary
  3. Java Reserved Word Summary
  4. The Java Library
  5. Java Language Coding Guidelines
  6. Tool Summary
  7. Number Systems
  8. UML Summary
  9. Java Syntax Summary
  10. HTML Summary

## Interactive eText Designed for Programming Students

Available online through wiley.com, vitalsource.com, or at your local bookstore, the enhanced eText features integrated student coding activities that foster in-depth learning. Designed by Cay Horstmann, these activities provide instant feedback to show students what they did right and where they need to study more. Students do more than just watch animations and code traces; they work on generating them right in the eText environment. For a preview of these activities, check out [http://wiley.](http://wiley/) com/college/sc/horstmann.

Customized formats are also available in both print and digital formats and pro- vide your students with curated content based on your unique syllabus.

Please contact your Wiley sales rep for more information about any of these options.

## Web Resources

This book is complemented by a complete suite of online resources. Go to www.wiley. com/go/bjeo7 to visit the online companion sites, which include

* Source code for all example programs in the book and its Worked Examples, plus additional example programs.
* Worked Examples that apply the problem-solving steps in the book to other realistic examples.
* Lecture presentation slides (for instructors only).
* Solutions to all review and programming exercises (for instructors only).
* A test bank that focuses on skills, not just terminology (for instructors only). This extensive set of multiple-choice questions can be used with a word processor or imported into a course management system.
* CodeCheck®, an innovative online service that allows instructors to design their own automatically graded programming exercises.

# Walkthrough of the Learning Aids

The pedagogical elements in this book work together to focus on and reinforce key concepts and fundamental principles of programming, with additional tips and detail organized to support and deepen these fundamentals. In addition to traditional features, such as chapter objectives and a wealth of exercises, each chapter contains elements geared to today’s visual learner.



Throughout each chapter,

6.3 The for Loop

6.3 The for Loop **183**

**margin notes** show where new concepts are introduced

and provide an outline of key ideas.

The for loop is used when a value runs from a starting point to an ending point with a constant increment or decrement.

It often happens that you want to execute a sequence of statements a given number of times. You can use a while loop that is controlled by a counter, as in the following example:

int counter = 5; // Initialize the counter

while (counter <= 10) // Check the counter

{

sum = sum + counter;

counter++; // Update the counter

}

Because this loop type is so common, there is a spe- cial form for it, called the for loop (see Syntax 6.2).

for (int counter = 5; counter <= 10; counter++)

{

sum = sum + counter;

}

Annotated **syntax boxes** provide a quick, visual overview of new language constructs.

Some people call this loop *count-controlled*. In con- trast, the while loop of the preceding section can be called an *event-controlled* loop because it exe- cutes until an event occurs; namely that the balance reaches the target. Another commonly used term for a count-controlled loop is *de)nite*. You know from the outset that the loop body will be executed a de1- nite number of times; ten times in our example. In contrast, you do not know how many iterations it takes to accumulate a target balance. Such a loop is called *inde)nite*.

Syntax 6.2 for Statement

© Enrico Fianchini/iStockphoto.

*You can visualize the* for *loop as an orderly sequence of steps.*

*Syntax*

for (*initialization*; *condition*; *update*)

{

**Annotations** explain required

*statements*

}

*These three expressions should be related.*

*See Programming Tip 6.1.*

components and point to more information on common errors

*This initialization*

*happens once*

*before the loop starts.*

*The condition is*

*checked before each iteration.*

*This update is*

*executed after each iteration.*

or best practices associated with the syntax.

*The variable* i *is defined only in this* for *loop.*

*See Special Topic 6.1.*

for (int i = 5; i <= 10; i++)

{

sum = sum + i;

}

*This loop executes 6 times. See Programming Tip 6.3.*

*Like a variable in a computer program, a parking space has an identifier and a contents.*

**Analogies** to everyday objects are used to explain the nature and behavior of concepts such as variables, data types, loops, and more.

**Memorable photos** reinforce analogies and help students remember the concepts.

**Problem Solving sections** teach techniques for generating ideas and evaluating proposed solutions, often using pencil and paper or other artifacts. These sections emphasize that most of the planning and problem solving that makes students successful happens away from the computer.

*In the same way that there can be a street named “Main Street” in diferent cities, a Java program can have multiple variables with the same name.*

* 1. Problem Solving: Discovering Algorithms by Manipulating Physical Objects **333**

Now how does that help us with our problem, switching the first and the second half of the array?

Let’s put the first coin into place, by swapping it with the fifth coin. However, as Java programmers, we will say that we swap the coins in positions 0 and 4:

Next, we swap the coins in positions 1 and 5:

**HOW TO 6.1**

**Writing a Loop**

This How To walks you through the process of implementing a loop statement. We will illustrate the steps with the following example problem.

**Problem Statement** Read twelve temperature values (one for each month) and display the num- ber of the month with the highest temperature. For example, according to [http://worldclimate.com,](http://worldclimate.com/) the average maximum temperatures for Death Valley are (in order by month, in degrees Celsius):

18.2 22.6 26.4 31.1 36.6 42.2

45.7 44.5 40.2 33.1 24.2 17.6

In this case, the month with the highest tempera- ture (45.7 degrees Celsius) is July, and the program should display 7.

© Stevegeer/iStockphoto.

**How To guides** give step-by-step guidance for common programming tasks, emphasizing planning and testing. They answer the beginner’s question, “Now what do I do?” and integrate key concepts into a problem-solving sequence.

**Step 1** Decide what work must be done *inside* the loop.

Every loop needs to do some kind of repetitive work, such as

* Reading another item.

• Updating a value (such as a bank balance or total).

**WORKED EXAMPLE 6.1**

• Incrementing a counter.

If you can’t 1gure out what needs to go inside the loop, start by writing down the steps that

**Credit Card Processing**

Learn how to use a loop to remove spaces from a credit card number. See your eText or visit wiley.com/go/bjeo7.

you would take if you solved the problem by hand. For example, with the temperature reading problem, you might write

© MorePixels/iStockphoto.

**Worked Examples** apply the steps in the How To to a different example, showing how they can be used to plan, implement, and test

a solution to another programming problem.



**Example tables** support beginners with multiple, concrete examples. These tables point out common errors and present another quick reference to the section’s topic.

Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement.

int width, height;

Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1.

int width;

**Error:** You cannot initialize a number with the string “20”. (Note the quotation marks.)

int width = "20";

**Error:** The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.2.5.

height = 30;

String greeting = "Hi!";

int perimeter = 4 \* width;

Declares an integer variable and initializes it with 20.

int width = 20;

Comment

Variable Name

Table 1 Variable Declarations in Java

This variable has the type String and is initialized with the string “Hi”.

The initial value need not be a fixed value. (Of course, width

must have been previously declared.)

**Progressive figures** trace code segments to help students visualize the program Now. Color is used consistently to make variables and other elements easily recognizable.

**Figure 3** Execution of a for Loop

* + 1. Initialize counter

counter =

5

* + 1. Check condition

counter =

5

for (int counter = 5; counter <= 10; counter++)

{

sum = sum + counter;

}

for (int counter = 5; counter <= 10; counter++)

{

sum = sum + counter;

}

* + 1. Execute loop body

counter =

5

for (int counter = 5; counter <= 10; counter++)

{

sum = sum + counter;

}

* + 1. Update counter

for (int counter = 5; counter <= 10; counter++)

{

**sec01/ElevatorSimulation.java**

counter =

sum = sum + counter;

6

}

import java.util.Scanner;

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**10**

**11**

**12**

**13**

**14**

**15**

**16**

**17**

/\*\*

This program simulates an elevator panel that skips the 13th floor.

* + 1. Check condition again

counter =

6

for (int counter = 5; counter <= 10; counter++)

{

sum = sum + counter;

}

\*/

public class ElevatorSimulation

{

public static void main(String[] args)

{

Scanner in = new Scanner(System.in); System.out.print("Floor: ");

int floor = in.nextInt();

// Adjust floor if necessary

int actualFloor; if (floor > 13)

The for loop neatly groups the initialization, condition, and update expressions together. However, it is important to realize that these expressions are not executed together (see Figure 3).

* The initialization is executed once, before the loop is entered. **1**
* The condition is checked before each iteration. **2 5**

• The update is executed after each iteration. **4**

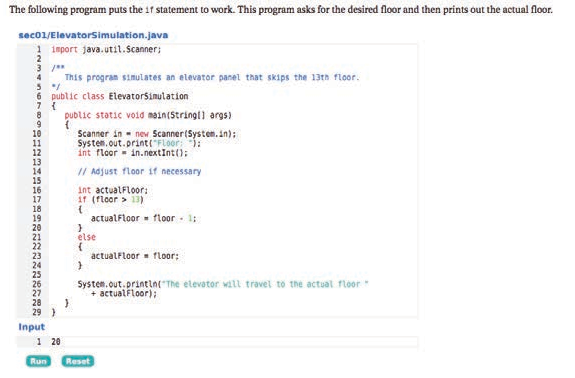
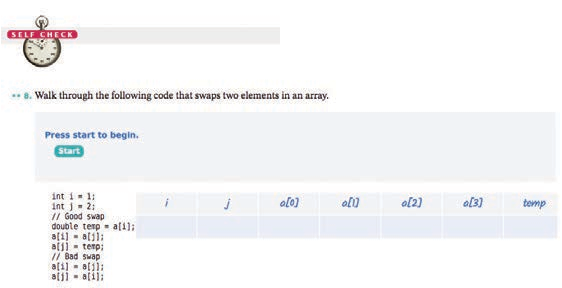
**Program listings** are carefully designed for easy reading, going well beyond simple color coding. Students can run and change the same programs right in the eText.

**Self-check exercises** in the eText are designed to engage students with the new material and check understanding before they continue to the next topic.

**•• Business E6.17** *Currency conversion*. Write a program

that first asks the user to type today’s price for one dollar in Japanese yen, then reads U.S. dollar values and converts each to yen. Use 0 as a sentinel.

Optional **science and business exercises** engage students with realistic applications of Java.



* **Science P6.15** Radioactive decay of radioactive materials can be modeled by the equation *A* = *A*0e*-t* (log 2/*h*), where *A* is the amount of the material at time *t*, *A*0 is the amount at time 0, and *h* is the half-life.

Technetium-99 is a radioisotope that is used in imaging of the brain. It has a half-life of 6 hours. Your program should display the relative amount *A* / *A*0 in a patient body every hour for 24 hours after receiving a dose.

**Common Errors** describe the kinds of errors that students often make, with an explanation of why the errors occur, and what to do about them.



**Common Error 7.4 Length and Size**

Unfortunately, the Java syntax for determining the number of elements in an array, an array list, and a string is not at all consistent. It is a common error to confuse these. You just have to remember the correct syntax for every data type.

|  |  |
| --- | --- |
| Data Type | Number of Elements |
| Array | a.length |
| Array list | a.size() |
| String | a.length() |

**Programming Tips** explain good programming practices, and encourage students to be more productive with tips and techniques such as hand-tracing.

**Programming Tip 5.5 Hand-Tracing**

A very useful technique for understanding whether a program works correctly is called *hand-tracing*. You simulate the pro- gram’s activity on a sheet of paper. You can use this method with pseudocode or Java code.

Get an index card, a cocktail napkin, or whatever sheet of paper is within reach. Make a column for each variable. Have the program code ready. Use a marker, such as a paper clip, to mark the current statement. In your mind, execute statements one at a time. Every time the value of a variable changes, cross out the old value and write the new value below the old one.

For example, let’s trace the getTax method with the data from the program run above. When the TaxReturn object is constructed, the income instance variable is set to 80,000 and status is set to

MARRIED. Then the getTax method is called. In lines 31 and 32 of Tax- Return.java, tax1 and tax2 are initialized to 0.

**29** public double getTax()

**30** {

© thomasd007/iStockphoto.

*Hand-tracing helps you understand whether a program works correctly.*

1. double tax1 = 0;
2. double tax2 = 0;

**33**

Because status is not SINGLE, we move to the else

branch of the outer if statement (line 46).

**34** if (status == SINGLE)

**35** {

**36** if (income <= RATE1\_SINGLE\_LIMIT)

**37** {

**38** tax1 = RATE1 \* income;

**39** }

**40** else

**41** {

**42** tax1 = RATE1 \* RATE1\_SINGLE\_LIMIT;

**43** tax2 = RATE2 \* (income - RATE1\_SINGLE\_LIMIT);

*income status tax1 tax2*

*80000 0*

*0*

*MARRIED*

**Special Topics** present optional topics and provide additional explanation of others.

Additional **full code examples**

**Special Topic 11.2 File Dialog Boxes**

In a program with a graphical user interface, you will want to use a 1le dialog box (such as the one shown in the 1gure below) whenever the users of your program need to pick a 1le. The JFileChooser class implements a 1le dialog box for the Swing user-interface toolkit.

The JFileChooser class has many options to 1ne-tune the display of the dialog box, but in its most basic form it is quite simple: Construct a 1le chooser object; then call the showOpenDialog or showSaveDialog method. Both methods show the same dialog box, but the button for select- ing a 1le is labeled “Open” or “Save”, depending on which method you call.

For better placement of the dialog box on the screen, you can specify the user-interface component over which to pop up the dialog box. If you don’t care where the dialog box pops up, you can simply pass null. The showOpenDialog and showSaveDialog methods return either JFileChooser.APPROVE\_OPTION, if the user has chosen a 1le, or JFileChooser.CANCEL\_OPTION, if the user canceled the selection. If a 1le was chosen, then you call the getSelectedFile method to obtain a File object that describes the 1le.

Here is a complete example:

JFileChooser chooser = new JFileChooser(); Scanner in = null;

if (chooser.showOpenDialog(null) == JFileChooser.APPROVE\_OPTION)

{

File selectedFile = chooser.getSelectedFile(); in = new Scanner(selectedFile);

throughout the text provide complete programs for students to run and modify.

}

**EXAMPLE CODE** See special\_topic\_2 of your eText or companion code for a program that demonstrates how to use a file chooser.

*Computing & Society 1.1* Computers Are Everywhere



When computers The advent of ubiqui-

**Computing & Society** presents social and historical topics on computing—for interest and to fulfill the “historical and social context” requirements of the

were first invented tous computing changed in the 1940s, a computer filled an many aspects of our entire room. The photo below shows lives. Factories used the ENIAC (*e*lectronic *n*umerical *i*nte- to employ people to grator *a*nd *c*omputer), completed in do repetitive assembly 1946 at the University of Pennsylvania. tasks that are today car- The ENIAC was used by the military ried out by computer- to compute the trajectories of projec- controlled robots, oper- tiles. Nowadays, computing facilities ated by a few people of search engines, Internet shops, and who know how to work social networks fill huge buildings with those computers. called data centers. At the other end of Books, music, and mov- the spectrum, computers are all around ies are nowadays often us. Your cell phone has a computer consumed on com- inside, as do many credit cards and fare puters, and comput- cards for public transit. A modern car ers are almost always

*This transit card contains a computer.*

ACM/IEEE curriculum guidelines.

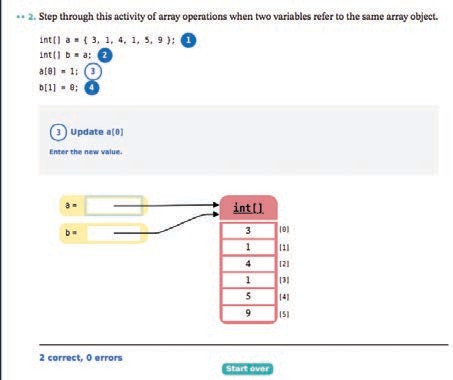
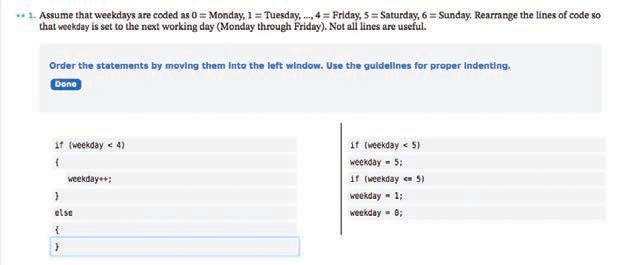
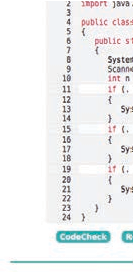
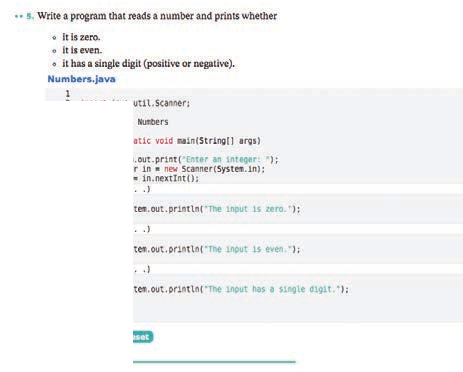
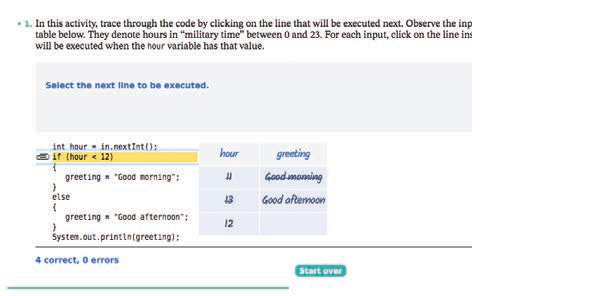
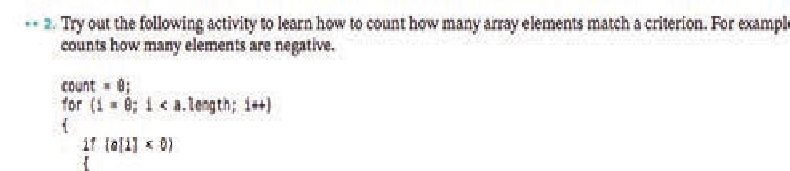
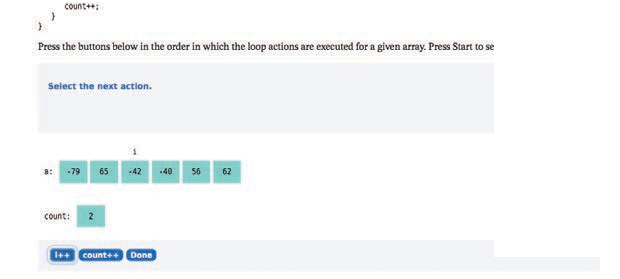
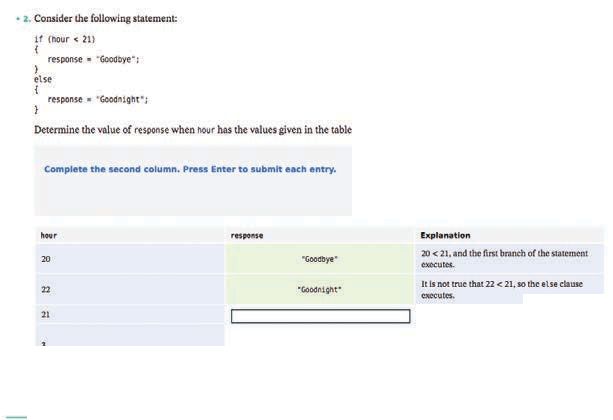
has several computers––to control the involved in their production. The engine, brakes, lights, and the radio. book that you are reading right now

could not have been written without computers.

**Interactive activities in the eText**

engage students in active reading as they…

**Complete** a program and get immediate feedback



**Trace** through a code segment

**Arrange** code to fulfill a task

**Build** an example table

**Create** a memory diagram

**Explore** common algorithms

Acknowledgments **xiii**

# Acknowledgments

Many thanks to Joanna Dingle, Crystal Franks, Graig Donini, and Michael Mac- Dougald at John Wiley & Sons, and Vickie Piercey at Publishing Services for their help with this project. An especially deep acknowledgment and thanks goes to Cindy Johnson for her hard work, sound judgment, and amazing attention to detail.

Special thanks to Stephen Gilbert, *Orange Coast College*, for his excellent help with the interactive exercises.

Many thanks to the individuals who worked through the many new activities in this edition, reviewed the manuscript, made valuable suggestions, and brought errors and omissions to my attention. They include:

Radhouane Chouchane, *Columbus State University* Sussan Einakian, *California Polytechnic State University* Jon Hanrath, *Illinois Institute of Technology*

Brian King, *Bucknell University*

Kathleen O’Brien, *San Jose State University*

Eman Saleh, *University of Georgia*

William Wei, *New York Institute of Technology*

Each new edition builds on the suggestions and experiences of prior reviewers, con- tributors, and users. I am grateful for the invaluable contributions these individuals have made:

Eric Aaron, *Wesleyan University*

James Agnew, *Anne Arundel Community College*

Tim Andersen, *Boise State University*

Ivan Bajic, *San Diego State University*

Greg Ballinger, *Miami Dade College*

Ted Bangay, *Sheridan Institute of Technology*

Ian Barland, *Radford University* George Basham, *Franklin University* Jon Beck, *Truman State University* Sambit Bhattacharya, *Fayetteville*

*State University*

Rick Birney, *Arizona State University*

Paul Bladek, *Edmonds Community College*

Matt Boutell, *Rose-Hulman Institute of Technology*

Joseph Bowbeer, *Vizrea Corporation*

Timothy A. Budd, *Oregon State University*

John Bundy, *DeVry University Chicago*

Robert P. Burton, *Brigham Young University*

Frank Butt, *IBM*

Jerry Cain, *Stanford University*

Adam Cannon, *Columbia University*

Michael Carney, *Finger Lakes Community College*

Robin Carr, *Drexel University*

Christopher Cassa, *Massachusetts Institute of Technology*

Nancy Chase, *Gonzaga University*

Dr. Suchindran S. Chatterjee,

*Arizona State University*

Archana Chidanandan, *Rose- Hulman Institute of Technology*

Vincent Cicirello, *The Richard Stockton College of New Jersey*

Gerald Cohen, *The Richard Stockton College of New Jersey*

Teresa Cole, *Boise State University*

Deborah Coleman, *Rochester Institute of Technology*

Tina Comston, *Franklin University* Lennie Cooper, *Miami Dade College* Jose Cordova, *University of*

*Louisiana, Monroe*

Valentino Crespi, *California State University, Los Angeles*

Jim Cross, *Auburn University*

Russell Deaton, *University of Arkansas*

Geoffrey Decker, *Northern Illinois University*

Suzanne Dietrich, *Arizona State University,West Campus*

Mike Domaratzki, *University of Manitoba*

H. E. Dunsmore, *Purdue University* Robert Duvall, *Duke University* Sherif Elfayoumy, *University of*

*North Florida*

Eman El-Sheikh, *University of West Florida*

Henry A. Etlinger, *Rochester Institute of Technology*

John Fendrich, *Bradley University* David Freer, *Miami Dade College* John Fulton, *Franklin University* David Geary, *Sabreware, Inc.*

Margaret Geroch, *Wheeling Jesuit University*

Ahmad Ghafarian, *North Georgia College & State University*

Rick Giles, *Acadia University* Stacey Grasso, *College of San Mateo* Jianchao Han, *California State*

*University, Dominguez Hills*

Lisa Hansen, *Western New England College*

Elliotte Harold

**xiv** Acknowledgments

Eileen Head, *Binghamton University*

Cecily Heiner, *University of Utah* Guy Helmer, *Iowa State University* Ed Holden, *Rochester Institute*

*of Technology*

Brian Howard, *Depauw University* Lubomir Ivanov, *Iona College* Norman Jacobson, *University of*

*California, Irvine*

Steven Janke, *Colorado College* Curt Jones, *Bloomsburg University* Mark Jones, *Lock Haven University*

*of Pennsylvania*

Dr. Mustafa Kamal, *University of Central Missouri*

Aaron Keen, *California Polytechnic State University, San Luis Obispo*

Mugdha Khaladkar, *New Jersey Institute of Technology*

Gary J. Koehler, *University of Florida*

Elliot Koffman, *Temple University* Ronald Krawitz, *DeVry University* Norm Krumpe, *Miami University*

*Ohio*

Jim Leone, *Rochester Institute of Technology*

Kevin Lillis, *St. Ambrose University*

Darren Lim, *Siena College*

Hong Lin, *DeVry University* Kathy Liszka, *University of Akron* Hunter Lloyd, *Montana State*

*University*

Youmin Lu, *Bloomsburg University*

Peter Lutz, *Rochester Institute of Technology*

Kuber Maharjan, *Purdue University College of Technology at Columbus*

John S. Mallozzi, *Iona College*

John Martin, *North Dakota State University*

Jeanna Matthews, *Clarkson University*

Patricia McDermott-Wells, *Florida International University*

Scott McElfresh, *Carnegie Mellon University*

Joan McGrory, *Christian Brothers University*

Carolyn Miller, *North Carolina State University*

Sandeep R. Mitra, *State University of New York, Brockport*

Teng Moh, *San Jose State University* Bill Mongan, *Drexel University* John Moore, *The Citadel*

Jose-Arturo Mora-Soto, Jesica Rivero-Espinosa, and Julio-Angel Cano-Romero, *University*

*of Madrid*

Faye Navabi, *Arizona State University*

Parviz Partow-Navid, *California State University, Los Angeles*

George Novacky, *University of Pittsburgh*

Kevin O’Gorman, *California Polytechnic State University, San Luis Obispo*

Michael Olan, *Richard Stockton College*

Mimi Opkins, *California State University Long Beach*

Derek Pao, *City University of Hong Kong*

Kevin Parker, *Idaho State University*

Jim Perry, *Ulster County Community College*

Cornel Pokorny, *California Polytechnic State University, San Luis Obispo*

Roger Priebe, *University of Texas, Austin*

C. Robert Putnam, *California State University, Northridge*

Kai Qian, *Southern Polytechnic State University*

Cyndi Rader, *Colorado School of Mines*

Neil Rankin, *Worcester Polytechnic Institute*

Brad Rippe, *Fullerton College*

Pedro I. Rivera Vega, *University of Puerto Rico, Mayaguez* Daniel Rogers, *SUNY Brockport*

Chaman Lal Sabharwal, *Missouri University of Science and Technology*

Katherine Salch, *Illinois Central College*

John Santore, *Bridgewater State College*

Javad Shakib, *DeVry University*

Carolyn Schauble, *Colorado State University*

Brent Seales, *University of Kentucky* Christian Shin, *SUNY Geneseo* Charlie Shu, *Franklin University*

Jeffrey Six, *University of Delaware*

Don Slater, *Carnegie Mellon University*

Ken Slonneger, *University of Iowa*

Aurelia Smith, *Columbus State University*

Donald Smith, *Columbia College*

Joslyn A. Smith, *Florida International University*

Stephanie Smullen, *University of Tennessee, Chattanooga*

Robert Strader, *Stephen F. Austin State University*

Monica Sweat, *Georgia Institute of Technology*

Peter Stanchev, *Kettering University*

Aakash Taneja, *The Richard Stockton College of New Jersey*

Craig Tanis, *University of Tennessee at Chattanooga*

Shannon Tauro, *University of California, Irvine*

Ron Taylor, *Wright State University*

Russell Tessier, *University of Massachusetts, Amherst*

Jonathan L. Tolstedt, *North Dakota State University*

David Vineyard, *Kettering University*

Joseph Vybihal, *McGill University* Xiaoming Wei, *Iona College* Jonathan S. Weissman, *Finger Lakes*

*Community College*

Todd Whittaker, *Franklin University*

Robert Willhoft, *Roberts Wesleyan College*

Brent Wilson, *George Fox University*

Katherine Winters, *University of Tennessee at Chattanooga*

Lea Wittie, *Bucknell University*

David Womack, *University of Texas at San Antonio*

David Woolbright, *Columbus State University*

Tom Wulf, *University of Cincinnati*

Catherine Wyman, *DeVry University*

Arthur Yanushka, *Christian Brothers University*

Qi Yu, *Rochester Institute of Technology*

Salih Yurttas, *Texas A&M University*

PREFACE **iii**

SPECIAL FEATURES **xxiv**

## INTRODUCTION **1**

**1**

* 1. Computer Programs **2**
  2. The Anatomy of a Computer **3**
  3. The Java Programming Language **5**
  4. Becoming Familiar with Your Programming Environment **7**
  5. Analyzing Your First Program **11**
  6. Errors **13**
  7. PROBLEM SOLVING Algorithm Design **15**

The Algorithm Concept 15 An Algorithm for Solving an Investment Problem 16

Pseudocode 17

From Algorithms to Programs 18

**HT1** Describing an Algorithm with Pseudocode 18

**WE1** Writing an Algorithm for Tiling a Floor 20

## USING OBJECTS **23**

**2**

* 1. Objects and Classes **24**

Using Objects 24

Classes 25

* 1. Variables **26**

Variable Declarations 26

Types 28

Names 29

Comments 30

Assignment 30

**ST1** Variable Type Inference 33

* 1. Calling Methods **33**

The Public Interface of a Class 33 Method Arguments 34

Return Values 35

Method Declarations 36

* 1. Constructing Objects **38**
  2. Accessor and Mutator Methods **40**
  3. The API Documentation **41** Browsing the API Documentation 41 Packages 43
  4. Implementing a Test Program **44**

**ST2** Testing Classes in an Interactive Environment 45

**WE1** How Many Days Have You Been Alive? 46

**WE2** Working with Pictures 46

* 1. Object References **46**
  2. Graphical Applications **49**

Frame Windows 50

Drawing on a Component 51 Displaying a Component in a Frame 53

* 1. Ellipses, Lines, Text, and Color **54**

Ellipses and Circles 54 Lines 55

Drawing Text 56

Colors 56

## IMPLEMENTING CLASSES **61**

**3**

* 1. Instance Variables and Encapsulation **62**

Instance Variables 62

The Methods of the Counter Class 64 Encapsulation 64

* 1. Specifying the Public Interface of a Class **66**

Specifying Methods 66

Specifying Constructors 67 Using the Public Interface 69

Commenting the Public Interface 69

* 1. Providing the Class Implementation **72**

Providing Instance Variables 72 Providing Constructors 73

Providing Methods 75

**HT1** Implementing a Class 78

**WE1** Making a Simple Menu 81

CONTENTS

**xv**

**xvi** Contents

* 1. Unit Testing **81**
  2. PROBLEM SOLVING Tracing Objects **84**
  3. Local Variables **86**
  4. The this Reference **88**

**ST1** Calling One Constructor from Another 90

* 1. Shape Classes **90**

**HT2** Drawing Graphical Shapes 94

## FUNDAMENTAL DATA TYPES **99**

**4**

* 1. Numbers **100** Number Types 100 Constants 102

**ST1** Big Numbers 106

* 1. Arithmetic **107**

Arithmetic Operators 107 Increment and Decrement 107 Integer Division and Remainder 108 Powers and Roots 109

Converting Floating-Point Numbers to Integers 110

**ST2** Avoiding Negative Remainders 112

**ST3** Combining Assignment and Arithmetic 113

**ST4** Instance Methods and Static Methods 113

* 1. Input and Output **114**

Reading Input 114

Formatted Output 115

**HT1** Carrying Out Computations 118

**WE1** Computing the Volume and Surface Area of a Pyramid 121

* 1. PROBLEM SOLVING First Do it By Hand **121**

**WE2** Computing Travel Time 122

* 1. Strings **122**

The String Type 122

Concatenation 123

String Input 124

Escape Sequences 124

Strings and Characters 124

Substrings 125

**ST5** Using Dialog Boxes for Input and Output 128

DECISIONS **131**

* 1. The if Statement **132**

**5**

**ST1** The Conditional Operator 137

* 1. Comparing Values **137**

Relational Operators 138

Comparing Floating-Point Numbers 139 Comparing Strings 140

Comparing Objects 141 Testing for null 141

**HT1** Implementing an if Statement 143

**WE1** Extracting the Middle 146

* 1. Multiple Alternatives **146**

**ST2** The switch Statement 148

* 1. Nested Branches **149**

**ST3** Block Scope 154

**ST4** Enumeration Types 155

* 1. PROBLEM SOLVING Flowcharts **156**
  2. PROBLEM SOLVING Selecting Test Cases **159**

**ST5** Logging 161

* 1. Boolean Variables and Operators **161**

**ST6** Short-Circuit Evaluation of Boolean Operators 165

**ST7** De Morgan’s Law 165

* 1. APPLICATION Input Validation **166**

## LOOPS 171

**6**

* 1. The while Loop **172**
  2. PROBLEM SOLVING Hand-Tracing **179**
  3. The for Loop **183**

**ST1** Variables Declared in a for Loop Header 189

* 1. The do Loop **190**
  2. APPLICATION Processing Sentinel Values **192**

**ST2** Redirection of Input and Output 194

**ST3** The “Loop and a Half” Problem 194

**ST4** The break and continue Statements 195

* 1. PROBLEM SOLVING Storyboards **197**
  2. Common Loop Algorithms **199** Sum and Average Value 199 Counting Matches 200

Finding the First Match 200 Prompting Until a Match is Found 201 Maximum and Minimum 201 Comparing Adjacent Values 202

**HT1** Writing a Loop 203

**WE1** Credit Card Processing 206

* 1. Nested Loops **206**

**WE2** Manipulating the Pixels in an Image 209

* 1. APPLICATION Random Numbers and Simulations **209**

Generating Random Numbers 210 The Monte Carlo Method 211

* 1. Using a Debugger **213**

**HT2** Debugging 215

**WE3** A Sample Debugging Session 217

## ARRAYS AND ARRAY LISTS **221**

**7**

* 1. Arrays **222**

Declaring and Using Arrays 222 Array References 225

Using Arrays with Methods 226 Partially Filled Arrays 226

**ST1** Methods with a Variable Number of Arguments 229

* 1. The Enhanced for Loop **230**
  2. Common Array Algorithms **232**

Filling 232

Sum and Average Value 232 Maximum and Minimum 232 Element Separators 232

Linear Search 233 Removing an Element 234 Inserting an Element 234 Swapping Elements 236

Copying Arrays 237

Reading Input 238

**ST2** Sorting with the Java Library 240

* 1. PROBLEM SOLVING Adapting Algorithms **240**

**HT1** Working with Arrays 242

**WE1** Rolling the Dice 245

* 1. PROBLEM SOLVING Discovering Algorithms by Manipulating Physical Objects **245**
  2. Two-Dimensional Arrays **248** Declaring Two-Dimensional Arrays 248 Accessing Elements 249

Locating Neighboring Elements 250 Accessing Rows and Columns 251

Two-Dimensional Array Parameters 252

**WE2** A World Population Table 253

**ST3** Two-Dimensional Arrays with Variable Row Lengths 254

**ST4** Multidimensional Arrays 255

* 1. Array Lists **255**

Declaring and Using Array Lists 255 Using the Enhanced for Loop with Array Lists 258

Copying Array Lists 259 Wrappers and Auto-boxing 259

Using Array Algorithms with Array Lists 260 Storing Input Values in an Array List 261 Removing Matches 261

Choosing Between Array Lists and Arrays 262

**ST5** The Diamond Syntax 264

* 1. Regression Testing **264**

## DESIGNING CLASSES **271**

**8**

* 1. Discovering Classes **272**
  2. Designing Good Methods **273** Providing a Cohesive Public Interface 273 Minimizing Dependencies 274 Separating Accessors and Mutators 275 Minimizing Side Effects 276

**ST1** Call by Value and Call by Reference 278

* 1. PROBLEM SOLVING Patterns for Object Data **282**

Keeping a Total 282 Counting Events 283

Collecting Values 283

Managing Properties of an Object 284 Modeling Objects with Distinct States 284 Describing the Position of an Object 285

* 1. Static Variables and Methods **286**

**ST2** Alternative Forms of Instance and Static Variable Initialization 289

**ST3** Static Imports 290

* 1. PROBLEM SOLVING Solve a Simpler Problem First **291**
  2. Packages **295**

Organizing Related Classes into Packages 295 Importing Packages 296

Package Names 297

Packages and Source Files 297

**ST4** Package Access 298

**HT1** Programming with Packages 299

* 1. Unit Test Frameworks **300**

## INHERITANCE **305**

**9**

* 1. Inheritance Hierarchies **306**
  2. Implementing Subclasses **310**
  3. Overriding Methods **314**

**ST1** Calling the Superclass Constructor 318

* 1. Polymorphism **319**

**ST2** Dynamic Method Lookup and the Implicit Parameter 322

**ST3** Abstract Classes 323

**ST4** Final Methods and Classes 324

**ST5** Protected Access 324

**HT1** Developing an Inheritance Hierarchy 325

**WE1** Implementing an Employee Hierarchy for Payroll Processing 330

* 1. Object: The Cosmic Superclass **330**

Overriding the toString Method 330

The equals Method 332

The instanceof Operator 333

**ST6** Inheritance and the toString Method 335

**ST7** Inheritance and the equals Method 336

INTERFACES **339**

**10**

10.1 Using Interfaces for Algorithm Reuse **340**

Discovering an Interface Type 340

Declaring an Interface Type 341

Implementing an Interface Type 343

Comparing Interfaces and Inheritance 345

**ST1** Constants in Interfaces 346

**ST2** Nonabstract Interface Methods 347

* 1. The Comparable Interface **350**

**ST3** The clone Method and the Cloneable

Interface 352

* 1. Using Interfaces for Callbacks **355**

**ST4** Lambda Expressions 358

**ST5** Generic Interface Types 360

* 1. Inner Classes **360**
  2. Mock Objects **361**
  3. Event Handling **363**

Listening to Events 363

Using Inner Classes for Listeners 365

* 1. Building Applications with Buttons **368**
  2. Processing Timer Events **371**
  3. Mouse Events **374 ST6** Keyboard Events 377 **ST7** Event Adapters 378

## INPUT/OUTPUT AND EXCEPTION HANDLING **383**

**11**

* 1. Reading and Writing Text Files **384**

**ST1** Reading Web Pages 387 **ST2** File Dialog Boxes 387 **ST3** Character Encodings 388

* 1. Text Input and Output **389**

Reading Words 389

Reading Characters 390

Classifying Characters 390

Reading Lines 390 Scanning a String 392

Converting Strings to Numbers 392 Avoiding Errors When Reading Numbers 392 Mixing Number, Word, and Line Input 393 Formatting Output 394

**ST4** Regular Expressions 395

**ST5** Reading an Entire File 396

* 1. Command Line Arguments **396**

**HT1** Processing Text Files 399

**WE1** Analyzing Baby Names 403

* 1. Exception Handling **403**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10.2 | Working with Interface Variables | **348** | Throwing Exceptions | 403 |
|  | Converting from Classes to Interfaces | 348 | Catching Exceptions | 405 |

Invoking Methods on Interface Variables 349 Casting from Interfaces to Classes 349

**WE1** Investigating Number Sequences 350

Checked Exceptions 407

Closing Resources 409

Designing Your Own Exception Types 410

**ST6** Assertions 411

**ST7** The try/finally Statement 412

* 1. APPLICATION Handling Input Errors **412**

## OBJECT-ORIENTED DESIGN **419**

**12**

* 1. Classes and Their Responsibilities **420**

Discovering Classes 420 The CRC Card Method 421

* 1. Relationships Between Classes **423**

Dependency 423

Aggregation 424

Inheritance 425

**HT1** Using CRC Cards and UML Diagrams in Program Design 426

**ST1** Attributes and Methods in UML Diagrams 426

**ST2** Multiplicities 427

**ST3** Aggregation, Association, and Composition 427

* 1. APPLICATION Printing an Invoice **428**

Requirements 429

CRC Cards 429

UML Diagrams 432

Method Documentation 432

Implementation 434

**WE1** Simulating an Automatic Teller Machine 439

RECURSION **443**

**13**

* 1. Triangle Numbers **444 HT1** Thinking Recursively 448 **WE1** Finding Files 452
  2. Recursive Helper Methods **452**
  3. The Efficiency of Recursion **453**
  4. Permutations **459**
  5. Mutual Recursion **463**
  6. Backtracking **469**

**WE2** Towers of Hanoi 475

## SORTING AND SEARCHING **477**

* 1. Selection Sort **478**

**14**

* 1. Profiling the Selection Sort Algorithm **481**
  2. Analyzing the Performance of the Selection Sort Algorithm **484 ST1** Oh, Omega, and Theta 486

**ST2** Insertion Sort 487

* 1. Merge Sort **488**
  2. Analyzing the Merge Sort Algorithm **491**

**ST3** The Quicksort Algorithm 493

* 1. Searching **495** Linear Search 495 Binary Search 497
  2. PROBLEM SOLVING Estimating the Running Time of an Algorithm **500**

Linear Time 500

Quadratic Time 501

The Triangle Pattern 502 Logarithmic Time 503

* 1. Sorting and Searching in the Java Library **504**

Sorting 504

Binary Search 505

Comparing Objects 505

**ST4** The Comparator Interface 506

**ST5** Comparators with Lambda Expressions 507

**WE1** Enhancing the Insertion Sort Algorithm 507

## THE JAVA COLLECTIONS FRAMEWORK **511**

**15**

* 1. An Overview of the Collections Framework **512**
  2. Linked Lists **514**

The Structure of Linked Lists 515

The LinkedList Class of the Java Collections Framework 516

List Iterators 516

* 1. Sets **520**

Choosing a Set Implementation 520 Working with Sets 522

* 1. Maps **525**

**ST1** Updating Map Entries 527 **HT1** Choosing a Collection 527 **WE1** Word Frequency 528

**ST2** Hash Functions 529

* 1. Stacks, Queues, and Priority Queues **531**

Stacks 531

Queues 532

Priority Queues 533

* 1. Stack and Queue Applications **534**

Balancing Parentheses 534

Evaluating Reverse Polish Expressions 535 Evaluating Algebraic Expressions 537 Backtracking 540

**ST3** Reverse Polish Notation 542

**WE2** Simulating a Queue of Waiting Customers 543

## BASIC DATA STRUCTURES **545**

**16**

* 1. Implementing Linked Lists **546**

The Node Class 546

Adding and Removing the First Element 547 The Iterator Class 548

Advancing an Iterator 549 Removing an Element 550 Adding an Element 552

Setting an Element to a Different Value 553 Efficiency of Linked List Operations 553 **ST1** Static Classes 560

**WE1** Implementing a Doubly-Linked List 560

* 1. Implementing Array Lists **560** Getting and Setting Elements 560 Removing or Adding Elements 562 Growing the Internal Array 563
  2. Implementing Stacks and Queues **564**

Stacks as Linked Lists 565 Stacks as Arrays 566 Queues as Linked Lists 567

Queues as Circular Arrays 568

* 1. Implementing a Hash Table **570**

Hash Codes 570

Hash Tables 570 Finding an Element 572

Adding and Removing Elements 572

Iterating over a Hash Table 573

**ST2** Open Addressing 578

## TREE STRUCTURES **581**

**17**

* 1. Basic Tree Concepts **582**
  2. Binary Trees **585**

Binary Tree Examples 586

Balanced Trees 588

A Binary Tree Implementation 589

**WE1** Building a Huffman Tree 590

* 1. Binary Search Trees **590**

The Binary Search Property 591

Insertion 592

Removal 594

Efficiency of the Operations 595

* 1. Tree Traversal **599**

Inorder Traversal 599

Preorder and Postorder Traversals 601

The Visitor Pattern 602

Depth-First and Breadth-First Search 603

Tree Iterators 604

* 1. Red-Black Trees **605**

Basic Properties of Red-Black Trees 605

Insertion 607

Removal 608

**WE2** Implementing a Red-Black Tree 611

* 1. Heaps **612**
  2. The Heapsort Algorithm **622**

## GENERIC CLASSES **629**

**18**

* 1. Generic Classes and Type Parameters **630**
  2. Implementing Generic Types **631**
  3. Generic Methods **634**
  4. Constraining Type Parameters **636**

**ST1** Wildcard Types 638

* 1. Type Erasure **639**

**ST2** Reflection 642

**WE1** Making a Generic Binary Search Tree Class 643

## STREAM PROCESSING **645**

**19**

* 1. The Stream Concept **646**
  2. Producing Streams **648**
  3. Collecting Results **649**

**ST1** Infinite Streams 651

* 1. Transforming Streams **652**
  2. Lambda Expressions **654**

**ST2** Method and Constructor References 656

**ST3** Higher-Order Functions 657

**ST4** Higher-Order Functions and Comparators 658

* 1. The Optional Type **659**
  2. Other Terminal Operations **661**
  3. Primitive-Type Streams **663** Creating Primitive-Type Streams 663 Mapping a Primitive-Type Stream 663 Processing Primitive-Type Streams 664
  4. Grouping Results **665**
  5. Common Algorithms Revisited **667**

Filling 667

Sum, Average, Maximum, and Minimum 668 Counting Matches 668

Element Separators 668

Linear Search 669

Comparing Adjacent Values 669 **HT1** Working with Streams 670 **WE1** Word Properties 672

**WE2** A Movie Database 673

## GRAPHICAL USER INTERFACES **675**

**20**

20.1 Layout Management **676** Using Layout Managers 676 Achieving Complex Layouts 677

Using Inheritance to Customize Frames 678

**ST1** Adding the main Method to the Frame Class 679

Check Boxes 687

Combo Boxes 687

**HT1** Laying Out a User Interface 692

**WE1** Programming a Working Calculator 694

* 1. Menus **695**
  2. Exploring the Swing Documentation **702**

## ADVANCED INPUT/OUTPUT\* (ETEXT ONLY)

**21**

* 1. Readers, Writers, and Input/Output Streams
  2. Binary Input and Output
  3. Random Access
  4. Object Input and Output Streams

**HT1** Choosing a File Format

* 1. File and Directory Operations

Paths

Creating and Deleting Files and Directories Useful File Operations

Visiting Directories

## MULTITHREADING\* (ETEXT ONLY)

**22**

* 1. Running Threads

**ST1** Thread Pools

* 1. Terminating Threads
  2. Race Conditions
  3. Synchronizing Object Access
  4. Avoiding Deadlocks

**ST2** Object Locks and Synchronized Methods

**ST3** The Java Memory Model

* 1. APPLICATION Algorithm Animation

## INTERNET NETWORKING\* (ETEXT ONLY)

**23**

23.1 The Internet Protocol

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 20.2 | Processing | Text Input | **680** | 23.2 | Application Level Protocols |
|  | Text Fields | 680 |  | 23.3 | A Client Program |
| 20.3 | Text Areas  Choices | 682  **685** |  | 23.4 | A Server Program  **HT1** Designing Client/Server Programs |

Radio Buttons 685

23.5 URL Connections

\*See your eText or visit [www.wiley.com/go/bjeo7.](http://www.wiley.com/go/bjeo7)

**xxii** Contents

## RELATIONAL DATABASES\* (ETEXT ONLY)

**24**

* 1. Organizing Database Information

Database Tables Linking Tables

Implementing Multi-Valued Relationships

**ST1** Primary Keys and Indexes

* 1. Queries

Simple Queries Selecting Columns Selecting Subsets Calculations

Joins

Updating and Deleting Data

* 1. Installing a Database
  2. Database Programming in Java Connecting to the Database Executing SQL Statements Analyzing Query Results

Result Set Metadata

* 1. APPLICATION Entering an Invoice

**ST2** Transactions

**ST3** Object-Relational Mapping

**WE1** Programming a Bank Database

## XML\* (ETEXT ONLY)

**25**

* 1. XML Tags and Documents

Advantages of XML

Differences Between XML and HTML The Structure of an XML Document

**HT1** Designing an XML Document Format

* 1. Parsing XML Documents
  2. Creating XML Documents

**HT2** Writing an XML Document

**ST1** Grammars, Parsers, and Compilers

* 1. Validating XML Documents

Document Type Definitions

Specifying a DTD in an XML Document Parsing and Validation

**HT3** Writing a DTD

**ST2** Schema Languages

**ST3** Other XML Technologies

APPENDIX A THE BASIC LATIN AND LATIN-1 SUBSETS OF UNICODE **A-1**

APPENDIX B JAVA OPERATOR SUMMARY **A-5** APPENDIX C JAVA RESERVED WORD SUMMARY **A-7** APPENDIX D THE JAVA LIBRARY **A-9**

APPENDIX E JAVA LANGUAGE CODING

GUIDELINES **A-38**

APPENDIX F TOOL SUMMARY (ETEXT ONLY) APPENDIX G NUMBER SYSTEMS (ETEXT ONLY) APPENDIX H UML SUMMARY (ETEXT ONLY)

APPENDIX I JAVA SYNTAX SUMMARY (ETEXT ONLY)

APPENDIX J HTML SUMMARY (ETEXT ONLY)

GLOSSARY **G-1** INDEX **I-1** CREDITS **C-1**

QUICK REFERENCE **C-3**

\*See your eText or visit [www.wiley.com/go/bjeo7.](http://www.wiley.com/go/bjeo7)

Arrays 223

**ALPHABETICAL LIST OF**

## SYNTAX BOXES

Array Lists 256

Assignment 31

Calling a Superclass Method 315

Cast 110

Catching Exceptions 406

Class Declaration 68

Comparisons 138

Constant Declaration 104

Constructor with Superclass Initializer 318

Declaring a Generic Class 632

Declaring a Generic Method 635

Declaring an Interface 342

for Statement 183

if Statement 134

Implementing an Interface 343

Importing a Class from a Package 43

Input Statement 115

Instance Variable Declaration 63

Java Program 12

Lambda Expressions 655

Object Construction 39

Package Specification 296

Subclass Declaration 311

The Enhanced for Loop 231

The instanceof Operator 334

The throws Clause 408

The try-with-resources Statement 409

Throwing an Exception 403

Two-Dimensional Array Declaration 249

while Statement 173

Variable Declaration 27

CHAPTER

## Common Errors

How Tos

and



## Worked Examples

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | Introduction | Omitting Semicolons Misspelling Words | 13  14 | Describing an Algorithm with Pseudocode Writing an Algorithm for  Tiling a Floor | 18  20 |  |
| **2** | Using Objects | Using Undeclared or  Uninitialized Variables 32  Confusing Variable Declarations  and Assignment Statements 32  Trying to Invoke a Constructor Like  a Method 40 | | How Many Days Have You Been Alive?  Working with Pictures | 46  46 |  |
| **3** | Implementing Classes | Declaring a Constructor as void  Ignoring Parameter Variables Duplicating Instance Variables in Local Variables  Providing Unnecessary Instance Variables  Forgetting to Initialize Object References in a Constructor | 72  77 | Implementing a Class Making a Simple Menu | 78  81 |  |
|  |  |  | Drawing Graphical Shapes | 94 |
|  |  | 86 |  |  |
|  |  | 87 |  |  |
|  |  | 87 |  |  |
| **4** | Fundamental | Unintended Integer Division | 111 | Carrying out Computations Computing the Volume and Surface Area of a Pyramid  Computing Travel Time | 118 |  |
|  | Data Types | Unbalanced Parentheses | 112 | 121 |
|  |  |  |  | 122 |
| **5** | Decisions | A Semicolon After the  if Condition 134  Using == to Compare Strings 143  The Dangling else Problem 152  Combining Multiple  Relational Operators 164  Confusing && and || Conditions 164 | | Implementing an  if Statement  Extracting the Middle | 143  146 |  |
| **6** | Loops | Don’t Think “Are We There Yet?” 177 Infinite Loops 177 | | Writing a Loop  Credit Card Processing Manipulating the Pixels in an Image  Debugging  A Sample Debugging Session | 203  206 |  |
|  |  | Off-by-One Errors 178 | |  |
|  |  |  | | 209 |
|  |  |  | | 215 |
|  |  |  | | 217 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Backup Copies | 10 |  | | Computers Are Everywhere | 5 |  |
|  | Choose Descriptive Variable Names  Learn By Trying  Don’t Memorize—Use Online Help | 32  37  43 | Variable Type Inference  Testing Classes in an Interactive Environment | 33  45 | Computer Monopoly | 49 |  |
|  | The javadoc Utility | 72 | Calling One Constructor from Another | 90 | Electronic Voting | 83 |  |
|  | Do Not Use Magic Numbers | 106 | Big Numbers Avoiding Negative Remainders  Combining Assignment and Arithmetic Instance Methods and  Static Methods  Using Dialog Boxes for Input and Output | 106  112  113  113  128 | Bugs in Silicon International Alphabets and Unicode | 114  128 |  |
| Spaces in Expressions | 112 |
| Reading Exception Reports | 127 |
|  | Brace Layout Always Use Braces Tabs  Avoid Duplication in Branches Hand-Tracing  Make a Schedule and Make Time for Unexpected Problems | 135  135  136  136  153  160 | The Conditional Operator The switch Statement Block Scope  Enumeration Types Logging  Short-Circuit Evaluation of Boolean Operators  De Morgan’s Law | 137  148  154  155  161  165  165 | Dysfunctional Computerized Systems  Artificial Intelligence | 145  168 |  |
|  | Use for Loops for Their Intended Purpose Only Choose Loop Bounds That  Match Your Task  Count Iterations Flowcharts for Loops | 188  188  189  191 | Variables Declared in a  for Loop Header  Redirection of Input and Output The Loop-and-a-Half Problem The break and continue  Statements | 189  194  194  195 | Digital Piracy | 182 |  |
| The First Bug | 217 |



Programming Tips

Special Topics

Computing & Society

CHAPTER

## Common Errors

How Tos

and



## Worked Examples

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **7** Arrays and Array Lists | Bounds Errors Uninitialized and Unfilled Arrays  Underestimating the Size of a Data Set Length and Size | 227 | Working with Arrays Rolling the Dice | 242  245 |  |
|  | 227 | A World Population Table | 253 |
|  | 240 |  |  |
|  | 264 |  |  |
| **8** Designing Classes | Trying to Access Instance Variables in Static Methods Confusing Dots | 288 | Programming with Packages | 299 |  |
|  | 298 |  |  |
| **9** Inheritance | Replicating Instance Variables from the Superclass Confusing Super- and  Subclasses  Accidental Overloading Forgetting to Use super When Invoking a  Superclass Method  Don’t Use Type Tests | 313 | Developing an Inheritance Hierarchy | 325 |  |
|  |  | Implementing an |  |
|  | 313 | Employee Hierarchy for |  |
|  | 317 | Payroll Processing | 330 |
|  | 318 |  |  |
|  | 335 |  |  |
| **10** Interfaces | Forgetting to Declare Implementing Methods  as Public 346  Trying to Instantiate an Interface 346 Modifying Parameter Types  in the Implementing Method 367  Trying to Call Listener Methods 368  Forgetting to Attach a Listener 371  Forgetting to Repaint 373 | | Investigating Number Sequences | 350 |  |
| **11** Input/Output and | Backslashes in File Names Constructing a Scanner with a String | 386 | Processing Text Files | 399 |  |
| Exception Handling | 386 | Analyzing Baby Names | 403 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Use Arrays for Sequences of Related Items  Make Parallel Arrays into Arrays of Objects  Batch Files and Shell Scripts | 228  228  266 | Methods with a Variable Number of Arguments Sorting with the Java Library  Two-Dimensional Arrays with Variable Row Lengths Multidimensional Arrays  The Diamond Syntax | 229  240  254  255  264 | Computer Viruses Liability for Software Malfunction | 229  267 |  |
|  | Consistency Minimize the Use of Static Methods | 277  289 | Call by Value and Call  by Reference 278  Alternative Forms of Instance  and Static Variable Initialization 289  Static Imports 290  Package Access 298 | | Personal Computing | 302 |  |
|  | Use a Single Class for Variation in Values, Inheritance for Variation in Behavior | 309 | Calling the Superclass Constructor  Dynamic Method Lookup and the Implicit Parameter Abstract Classes  Final Methods and Classes Protected Access Inheritance and the  toString Method  Inheritance and the  equals Method | 318  322  323  324  324  335  336 | Who Controls the Internet? | 337 |  |
|  | Comparing Integers and Floating- Point Numbers 351 | | Constants in Interfaces Nonabstract Interface Methods The clone Method and the  Cloneable Interface  Lambda Expressions Generic Interface Types Keyboard Events  Event Adapters | 346  347  352  358  360  377  378 | Open Source and Free Software | 379 |  |
|  | Throw Early, Catch Late  Do Not Squelch Exceptions Do Throw Specific Exceptions | 411  411  411 | Reading Web Pages File Dialog Boxes Character Encodings Regular Expressions Reading an Entire File Assertions  The try/finally Statement | 387  387  388  395  396  411  412 | Encryption Algorithms  The Ariane Rocket Incident | 402  417 |  |



Programming Tips

Special Topics

Computing & Society

CHAPTER

## Common Errors

How Tos

and



## Worked Examples

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **12** Object-Oriented Design | |  | | Using CRC Cards and UML Diagrams in Program Design  Simulating an Automatic Teller Machine | 426  439 |  |
| **13** | Recursion | Infinite Recursion  Tracing Through Recursive Methods | 447  447 | Thinking Recursively Finding Files  Towers of Hanoi | 448  452  475 |  |
| **14** Sorting and Searching | | The compareTo Method Can Return Any Integer,  Not Just –1, 0, and 1 | 506 | Enhancing the Insertion Sort Algorithm | 507 |  |
| **15** | The Java Collections Framework |  | | Choosing a Collection Word Frequency  Simulating a Queue of Waiting Customers | 527  528  543 |  |
| **16** Basic Data Structures | |  | | Implementing a Doubly- Linked List | 560 |  |
| **17** | Tree Structures |  | | Building a Huffman Tree Implementing a Red-Black Tree | 590  611 |  |
| **18** | Generic Classes | Genericity and Inheritance The Array Store Exception  Using Generic Types in a Static Context | 637  638  642 | Making a Generic Binary Search Tree Class | 643 |  |
| **19** | Stream Processing | Don’t Use a Terminated Stream  Optional Results Without Values Don’t Apply Mutations in Parallel Stream Operations | 654  660  662 | Working with Streams Word Properties  A Movie Database | 670  672  673 |  |
| **20** | Graphical User Interfaces | By Default, Components Have Zero Width and Height | 679 | Laying Out a User Interface Programming a Working Calculator | 692  694 |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | Attributes and Methods in UML Diagrams Multiplicities  Aggregation, Association, and Composition | 426  427  427 | Electronic Privacy 439 |  |
|  |  | |  | | The Limits of Computation 461 |  |
|  |  | | Oh, Omega, and Theta Insertion Sort  The Quicksort Algorithm The Comparator Interface  Comparators with Lambda Expressions | 486 | The First Programmer 499 |  |
| 487 |  |
| 493 |  |
| 506 |  |
| 507 |  |
|  | Use Interface References to Manipulate Data Structures | 524 | Updating Map Entries | 527 | Standardization 519 |  |
| Hash Functions | 529 |  |
| Reverse Polish Notation | 542 |  |
|  |  | | Static Classes | 560 |  |  |
| Open Addressing | 578 |
|  |  | |  | |  |  |
|  |  | | Wildcard Types | 638 |  |  |
| ReNection | 642 |
|  | One Stream Operation Per Line | 651 | Infinite Streams  Method and Constructor References  Higher-Order Functions Higher-Order Functions and Comparators | 651 |  |  |
| Keep Lambda Expressions Short | 656 |  |
|  |  | 656 |
|  |  | 657 |
|  |  | 658 |
|  | Use a GUI Builder | 694 | Adding the main Method to the Frame Class | 679 |  |  |



Programming Tips

Special Topics

Computing & Society

CHAPTER

## Common Errors

How Tos

and



## Worked Examples

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **21** | Advanced Input/Output\* (eTEXT ONLY) | Negative byte Values | W714 | Choosing a File Format | W723 |  |
|  | **22** | Multithreading\*  (eTEXT ONLY) | Calling await Without Calling signalAll  Calling signalAll Without Locking the Object | W755 W755 |  | |  |
|  | **23** Internet Networking\*  (eTEXT ONLY) | |  | | Designing Client/Server Programs | W786 |  |
|  | **24** Relational Databases\*  (eTEXT ONLY) | | Joining Tables Without Specifying a Link Condition Constructing Queries from  Arbitrary Strings | W809 W820 | Programming a Bank Database | W831 |  |
|  | **25** | XML\*  (eTEXT ONLY) | XML Elements Describe Objects,  Not Classes W853 | | Designing an XML Document Format  Writing an XML Document Writing a DTD | W845 W859 W869 |  |

\*See your eText or visit [www.wiley.com/go/bjeo7.](http://www.wiley.com/go/bjeo7)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | |  | |  |  |
|  | Use the Runnable Interface W740 Check for Thread Interruptions  in the run Method of a Thread W742 | | Thread Pools Object Locks and  Synchronized Methods  The Java Memory Model | W740  W755 W756 |  |  |
|  | Use High-Level Libraries | W790 |  | |  |  |
|  | Stick with the Standard Avoid Unnecessary Data Replication  Don’t Replicate Columns in a Table  Don’t Hardwire Database Connection Parameters into Your Program  Let the Database Do the Work | W802 | Primary Keys and Indexes | W803 |  |  |
|  | Transactions | W829 |
| W802 | Object-Relational Mapping | W830 |
| W803 |  |  |
| W820 |  |  |
| W821 |  |  |
|  | Prefer XML Elements over Attributes  Avoid Children with Mixed Elements and Text | W847 | Grammars, Parsers, and Compilers Schema Languages  Other XML Technologies | W860 |  |  |
|  | W871 |
| W848 | W872 |



Programming Tips

Special Topics

Computing & Society

\*See your eText or visit [www.wiley.com/go/bjeo7.](http://www.wiley.com/go/bjeo7)

**1**

CHAPTER

INTRODUCTION

CHAPTER GOALS

To learn about computers and programming

To compile and run your first Java program

To recognize compile-time and run-time errors

To describe an algorithm with pseudocode

© JanPietruszka/iStockphoto.

CHAPTER CONTENTS

* 1. **COMPUTER PROGRAMS** 2
  2. **THE ANATOMY OF A COMPUTER** 3

**C&S** Computers Are Everywhere 5

* 1. **THE JAVA PROGRAMMING LANGUAGE** 5
  2. **BECOMING FAMILIAR WITH YOUR PROGRAMMING ENVIRONMENT** 7

**PT1** Backup Copies 10

* 1. **ANALYZING YOUR FIRST PROGRAM** 11

**SYN** Java Program 12

**CE1** Omitting Semicolons 13

* 1. **ERRORS** 13

**CE2** Misspelling Words 14

* 1. **PROBLEM SOLVING: ALGORITHM DESIGN** 15

**HT1** Describing an Algorithm with Pseudocode 18

**WE1** Writing an Algorithm for Tiling a Floor 20



**1**



Just as you gather tools, study a project, and make a plan for tackling it, in this chapter you will gather up the basics you need to start learning to program. After a brief introduction to computer hardware, software, and programming in general, you will learn how to write and run your first Java program. You will also learn how to diagnose and fix programming errors, and how to use pseudocode to describe an algorithm—a step-by-step description of how to solve a problem—as you plan your computer programs.

# 1.1 Computer Programs

You have probably used a computer for work or fun. Many people use computers for everyday tasks such as electronic banking or writing a term paper. Computers are good for such tasks. They can handle repetitive chores, such as totaling up numbers or placing words on a page, without getting bored or exhausted.

Computers execute very basic instructions in rapid succession.

The flexibility of a computer is quite an amazing phenomenon. The same machine can balance your checkbook, lay out your term paper, and play a game. In contrast, other machines carry out a much narrower range of tasks; a car drives and a toaster toasts. Computers can carry out a wide range of tasks because they execute different programs, each of which directs the computer to work on a specific task.

The computer itself is a machine that stores data (numbers, words, pictures), inter- acts with devices (the monitor, the sound system, the printer), and executes programs. A **computer program** tells a computer, in minute detail, the sequence of steps that are needed to fulfill a task. The physical computer and peripheral devices are collec- tively called the **hardware**. The programs the computer executes are called the **software**.

A computer program is a sequence

of instructions and decisions.

Today’s computer programs are so sophisticated that it is hard to believe that they are composed of extremely primitive instructions. A typical instruction may be one of the following:

* Put a red dot at a given screen position.
* Add up two numbers.
* If this value is negative, continue the program at a certain instruction.

The computer user has the illusion of smooth interaction because a program contains a huge number of such instructions, and because the computer can execute them at great speed.

The act of designing and implementing computer programs is called **program- ming**. In this book, you will learn how to program a computer—that is, how to direct the computer to execute tasks.

Programming is the act of designing and implementing

computer programs.

To write a computer game with motion and sound effects or a word processor that supports fancy fonts and pictures is a complex task that requires a team of many highly-skilled programmers. Your first programming efforts will be more mundane. The concepts and skills you learn in this book form an important foundation, and you should not be disappointed if your first programs do not rival the sophisticated software that is familiar to you. Actually, you will find that there is an immense thrill even in simple programming tasks. It is an amazing experience to see the computer

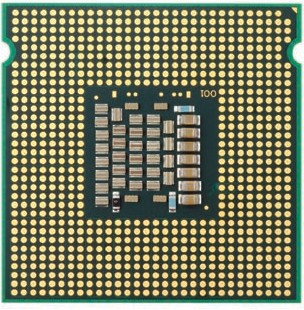
**2**

1.2 The Anatomy of a Computer **3**

precisely and quickly carry out a task that would take you hours of drudgery, to make small changes in a program that lead to immediate improvements, and to see the computer become an extension of your mental powers.

# The Anatomy of a Computer

To understand the programming process, you need to have a rudimentary under- standing of the building blocks that make up a computer. We will look at a personal computer. Larger computers have faster, larger, or more powerful components, but they have fundamentally the same design.

At the heart of the computer lies the **central processing unit (CPU)** (see Figure 1). The inside wiring of the CPU is enormously complicated. For example, the Intel Core processor (a popular CPU for personal computers at the time of this writing) is composed of several hundred million structural elements, called *transistors*.

The CPU performs program control and data processing. That is, the CPU locates and exe- cutes the program instructions; it carries out arithmetic operations such as addition, subtrac- tion, multiplication, and division; it fetches data

The central processing unit (CPU) performs program control and

data processing.

from external memory or devices and places processed data into storage.

There are two kinds of storage. Primary stor-

Storage devices include memory and secondary storage.

© Amorphis/iStockphoto.

**Figure 1** Central Processing Unit

age, or memory, is made from electronic circuits that can store data, provided they are supplied with electric power. **Secondary storage**, usually a **hard disk** (see Figure 2) or a solid-state drive, provides slower and less expensive storage that persists without electricity. A hard disk consists of rotating platters, which are coated with a magnetic



© PhotoDisc, Inc./Getty Images, Inc.

**Figure 2** A Hard Disk

material. A solid-state drive uses electronic components that can retain information without power, and without moving parts.

To interact with a human user, a computer requires peripheral devices. The com- puter transmits information (called *output*) to the user through a display screen, speakers, and printers. The user can enter information (called *input*) for the computer by using a keyboard or a pointing device such as a mouse.

Some computers are self-contained units, whereas others are interconnected through **networks**. Through the network cabling, the computer can read data and programs from central storage locations or send data to other computers. To the user of a networked computer, it may not even be obvious which data reside on the com- puter itself and which are transmitted through the network.

Figure 3 gives a schematic overview of the architecture of a personal computer. Program instructions and data (such as text, numbers, audio, or video) reside in sec- ondary storage or elsewhere on the network. When a program is started, its instruc- tions are brought into memory, where the CPU can read them. The CPU reads and executes one instruction at a time. As directed by these instructions, the CPU reads data, modifies it, and writes it back to memory or secondary storage. Some program instructions will cause the CPU to place dots on the display screen or printer or to vibrate the speaker. As these actions happen many times over and at great speed, the human user will perceive images and sound. Some program instructions read user input from the keyboard, mouse, touch sensor, or microphone. The program ana- lyzes the nature of these inputs and then executes the next appropriate instruction.



Monitor



Speakers

Internet

**Figure 3** Schematic Design of a Personal Computer



Printer

Mouse/Trackpad

Secondary storage

Keyboard

Microphone

Video camera

Network controller

Memory

CPU

Disk controller

Ports

* 1. The Java Programming Language **5**

*Computing & Society 1.1* Computers Are Everywhere



When computers were first invented in the

1940s, a computer filled an entire room. The photo below shows the ENIAC (*e*lectronic *n*umerical *i*ntegrator *a*nd *c*omputer), completed in 1946 at the University of Pennsylvania. The ENIAC was used by the military to compute the trajectories of projectiles. Nowadays, computing facilities of search engines, Internet shops, and social networks fill huge buildings called data centers. At the other end of the spectrum, computers are all around us. Your cell phone has a computer inside, as do many credit cards and fare cards for public transit. A modern car

The advent of ubiqui- tous computing changed many aspects of our lives. Factories used to employ people to do repetitive assembly tasks that are today car- ried out by computer- controlled robots, oper- ated by a few people who know how to work with those com- puters. Books, music, and movies nowadays are often consumed on computers, and computers are almost

© Maurice Savage/Alamy Limited.

*This transit card contains a computer.*

has several computers––to control the engine, brakes, lights, and the radio.

© UPPA/Photoshot.

*The ENIAC*

always involved in their production. The book that you are reading right

now could not have been written with- out computers.

Knowing about computers and how to program them has become an essential skill in many careers. Engi- neers design computer-controlled cars and medical equipment that preserve lives. Computer scientists develop programs that help people come together to support social causes. For example, activists used social net- works to share videos showing abuse by repressive regimes, and this infor- mation was instrumental in changing public opinion.

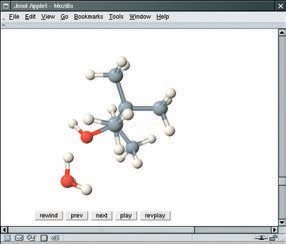
As computers, large and small, become ever more embedded in our everyday lives, it is increasingly important for everyone to understand how they work, and how to work with them. As you use this book to learn how to program a computer, you will develop a good understanding of com- puting fundamentals that will make you a more informed citizen and, per- haps, a computing professional.

# 1.3 Bahasa Pemrograman Java

Untuk menulis program komputer, Anda perlu memberikan urutan instruksi yang dapat dieksekusi oleh CPU. Sebuah program komputer terdiri dari sejumlah besar program instruksi CPU sederhana, dan membosankan juga rawan untuk kesalahan untuk menentukannya satu persatu. Karena alasan itu, bahasa pemrograman tingkat tinggi telah dibuat. Dalam bahasa tingkat tinggi, Anda menentukan tindakan yang harus dilakukan programnya. Penyusun menerjemahkan instruksi tingkat tinggi ke dalam instruksi yang lebih rinci (disebut kode mesin) yang dibutuhkan oleh CPU. Banyak bahasa pemrograman yang berbeda telah dirancang untuk tujuan yang berbeda.

 Pada tahun 1991, sebuah kelompok yang dipimpin oleh James Gosling dan Patrick Naughton di Sun Microsystems merancang sebuah bahasa pemrograman, berkode nama “Green”, untuk kegunaan di perangkat konsumen, seperti “set-top” televisi cerdas. Bahasanya dirancang untuk sederhana, aman, dan dapat digunakan oleh berbagai tipe prosesor yang berbeda. Tidak ada pelanggan yang pernah ditemukan untuk teknologi ini.

James Gosling

Gosling menceritakan pada tahun 1994 tim menyadari “Kita bisa menulis browser yang sangat keren. Itu adalah salah satu dari sedikit hal dalam arus utama klien/server yang membutuhkan beberapa hal aneh yang telah kami lakukan: arsitektur netral, waktu nyata, andal, aman.” Java diperkenalkan kepada orang banyak yang antusias di pameran SunWorld pada tahun 1995, bersama dengan browser yang menjalankan applet—kode Java yang dapat ditemukan di mana saja di Internet. Gambar di sebelah kanan menunjukkan contoh khas applet.

Sebuah applet untuk memvisualisasikan molekul

Java pada awalnya dirancang untuk pemrograman perangkat konsumen, tapi pertama berhasil digunakan untuk menulis applet internet.

Sejak saat itu, Java telah berkembang dengan kecepatan yang fenomenal. Programmer telah menggunakan bahasa ini karena lebih mudah digunakan daripada saingan terdekatnya, C++. Selain itu, Java memiliki perpustakaan yang kaya yang memungkinkan untuk menulis program portabel yang dapat melewati sistem operasi berpemilik—fitur yang sangat dicari oleh mereka yang ingin menjadi independen dari sistem berpemilik tersebut dan diperjuangkan dengan sengit oleh vendor mereka.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Table 1 Versi Java (sejak versi 1.0 pada tahun 1996) | | | | | |
| Versi | Tahun | Fitur Penting Baru | Versi | Tahun | Fitur Penting Baru |
| 1.1 | 1997 | Inner classes | 6 | 2006 | Peningkatan library |
| 1.2 | 1998 | Swing, Collections framework | 7 | 2011 | Perubahan kecil pada bahasa dan peningkatan library |
| 1.3 | 2000 | Peningkatan performa | 8 | 2014 | Function expressions, streams, library tanggal/waktu baru |
| 1.4 | 2002 | Dukungan assertions dan XML | 9 | 2017 | Modul |
| 5 | 2004 | Generic classes, peningkatan for loop, auto-boxing, pencacahan, anotasi | 10, 11 | 2018 | Versi dengan peningkatan tambahan dirilis setiap enam bulan |

Karena Java dirancang untuk internet, ia mempunyai dua atrbut yang membuatnya sangat cocok untuk pemula: keamanan dan portabilitas.

Java dirancang untuk agar aman dan portabel, menguntungkan kedua pengguna internet dan pelajar.

Java dirancang sehingga siapa pun dapat menjalankan program di browser mereka tanpa takut. Fitur keamanan bahasa Java memastikan program dihentikan apabila mencoba untuk melakukan sesuatu membahayakan. Memiliki lingkungan yang aman juga membantu bagi siapa pun yang mempelajari Java. Di saat Anda melakukan error yang mengakibatkan perilaku membahayakan, program Anda dihentikan dan Anda menerima laporan eror yang akurat.

Manfaat lain dari Java adalah portabilitas. Program Java yang sama akan jalan, tanpa perubahan, pada Windows, UNIX, Linux, atau Macintosh. Untuk mencapai portabilitas, penyusun Java tidak menerjemahkan program-program Java langsung ke dalam instruksi CPU. Melainkan, program Java yang tersusun memuat instruksi untuk **mesin virtual** Java, sebuah program yang menyimulasikan CPU nyata. Portabilitas adalah manfaat lain untuk murid pemula. Anda tidak harus belajar bagaimana cara menulis program untuk platform yang berbeda.

Program Java didistribusikan sebagai instruksi untuk mesin virtual, membuatnya mandiri platform.

Saat ini, Java telah ditetapkan sebagai salah satu bahasa yang paling penting untuk pemrograman umum serta untuk instruksi ilmu komputer. Namun, meskipun Java adalah bahasa yang baik untuk pemula, tapi tidak sempurna, karena tiga alasan.

Karena Java tidak dirancang khusus untuk siswa, tidak ada pemikiran yang diberikan untuk membuatnya sangat sederhana untuk menulis program dasar. Sejumlah mesin teknis diperlukan untuk menulis bahkan program yang paling sederhana. Hal ini bukan masalah bagi programmer profesional, tetapi bisa menjadi gangguan bagi pelajar pemula. Saat Anda mempelajari caranya untuk memprogram di Java, akan ada saatnya Anda akan diminta untuk puas dengan penjelasan awal dan menunggu detail yang lebih lengkap di bab selanjutnya.

Java telah diperpanjang berkali-kali selama masa pakainya—lihat Table 1. Dalam buku ini, kami menganggap bahwa Anda memiliki Java versi 8 atau yang lebih baru.

Pada akhirnya, Anda tidak dapat berharap untuk mempelajari keseluruhan dari Java dalam satu pembelajaran. Bahasa Java itu sendiri relatif sederhana, tapi Java berisi sekumpulan *library packages* yang diperlukan untuk menulis program yang berguna. Ada paket untuk grafik, desain antarmuka pengguna, kriptografi, jaringan, suara, penyimpanan basis data, dan banyak kegunaan lainnya. Bahkan programmer Java yang ahli sekalipun tidak dapat berharap untuk mengetahui isi semua paket—mereka hanya menggunakan yang mereka perlukan untuk proyek tertentu.

Dengan menggunakan buku ini, Anda diharapkan untuk belajar banyak tentang bahasa Java dan tentang paket yang paling penting. Camkan selalu bahwa tujuan utama dari buku ini bukan untuk Anda mengingat detail kecil Java, tapi untuk mengajari Anda cara berpikir tentang pemrograman.

Java memiliki library yang sangat besar. Fokuslah mempelajari bagian library yang diperlukan untuk proyek pemrogramanmu

# Becoming Familiar with Your

Programming Environment

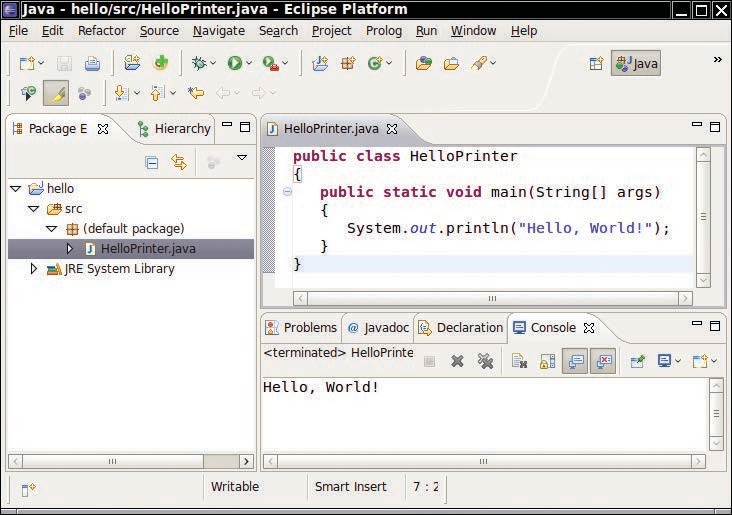
Many students find that the tools they need as programmers are very different from the software with which they are familiar. You should spend some time making yourself familiar with your programming environment. Because computer systems vary widely, this book can only give an outline of the steps you need to follow. It is a good idea to participate in a hands-on lab, or to ask a knowledgeable friend to give you a tour.

Set aside time to become familiar with the programming environment that you will use for your class work.

**Step 1** Start the Java development environment.

Computer systems differ greatly in this regard. On many computers there is an **inte- grated development environment** in which you can write and test your programs.

**Figure 4** Running the HelloPrinter



Click to compile and run

Java program

Program output

Program in an Integrated Development Environment

On other computers you first launch an **editor**, a program that functions like a word processor, in which you can enter your Java instructions; you then open a *console window* and type commands to execute your program. You need to find out how to get started with your environment.

An editor is a program for entering and modifying

text, such as a Java program.

**Step 2** Write a simple program.

The traditional choice for the very first program in a new programming language is a program that displays a simple greeting: “Hello, World!”. Let us follow that tradi- tion. Here is the “Hello, World!” program in Java:

public class HelloPrinter

{

public static void main(String[] args)

{

System.out.println("Hello, World!");

}

}

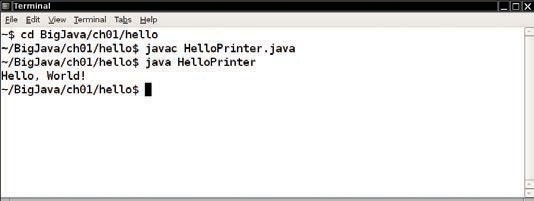
We will examine this program in the next section.

No matter which programming environment you use, you begin your activity by typing the program statements into an editor window.

Create a new file and call it HelloPrinter.java, using the steps that are appropriate for your environment. (If your environment requires that you supply a project name in addition to the file name, use the name hello for the project.) Enter the program instructions *exactly* as they are given above. Alternatively, locate the electronic copy in this book’s companion code and paste it into your editor.

Java is case sensitive. You must be careful about distinguishing between upper- and lowercase letters.

As you write this program, pay careful attention to the various symbols, and keep in mind that Java is **case sensitive**. You must enter upper- and lowercase letters exactly as they appear in the program listing. You cannot type MAIN or PrintLn. If you are not careful, you will run into problems—see Common Error 1.2.



**Figure 5** Running the HelloPrinter Program in a Console Window

**Step 3**

Run the program.

The process for running a program depends greatly on your programming environ- ment. You may have to click a button or enter some commands. When you run the test program, the message

Hello, World!

will appear somewhere on the screen (see Figure 4 and Figure 5).

In order to run your program, the Java compiler translates your **source files** (that is, the statements that you wrote) into *class files*. (A class file contains instructions for the Java virtual machine.) After the compiler has translated your **source code** into virtual machine instructions, the virtual machine executes them. During execution, the virtual machine accesses a library of pre-written code, including the implementa- tions of the System and PrintStream classes that are necessary for displaying the program’s output. Figure 6 summarizes the process of creating and running a Java program. In some programming environments, the compiler and virtual machine are essentially invisible to the programmer—they are automatically executed whenever you ask to run a Java program. In other environments, you need to launch the com- piler and virtual machine explicitly.

The Java compiler translates source code into class files that contain instructions for the

Java virtual machine.

**Step 4** Organize your work.

As a programmer, you write programs, try them out, and improve them. You store your programs in **files**. Files are stored in **folders** or **directories**. A folder can contain

Virtual Machine

Source File

Compiler

Editor

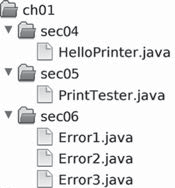
Running Program

**Figure 6** From Source Code to Running Program

Class 1les

Library 1les

files as well as other folders, which themselves can contain more files and folders (see Figure 7). This hierarchy can be quite large, and you need not be concerned with all of its branches. However, you should create folders for organizing your work. It is a good idea to make a separate folder for your pro-

gramming coursework. Inside that folder, make a separate folder for each program.

Some programming environments place your programs into a default location if you don’t spec- ify a folder yourself. In that case, you need to find out where those files are located.

Be sure that you understand where your files are located in the folder hierarchy. This information is essential when you submit files for grading, and for

making *backup copies* (see Programming Tip 1.1).

**Figure 7** A Folder Hierarchy

**Programming Tip 1.1 Backup Copies**

You will spend many hours creating and improving Java programs. It is easy to delete a file by accident, and occasionally files are lost because of a computer malfunction. Retyping the contents of lost files is frustrating and time-consuming. It is therefore crucially important that you learn how to safeguard files and get in the habit of doing so *before* disaster strikes. Backing up files on a memory

Develop a strategy for keeping backup copies of your work before disaster strikes.



© Tatiana Popova/iStockphoto.

stick is an easy and convenient storage method for many people. Another increasingly popular form of backup is Internet file storage.

Here are a few pointers to keep in mind:

* *Back up often.* Backing up a file takes only a few seconds, and you will hate yourself if you have to spend many hours recreating work that you could have saved easily. I recommend that you back up your work once every thirty minutes.
* *Rotate backups.* Use more than one directory for backups, and rotate them. That is, first back up onto the first directory. Then back up onto the second directory. Then use the third, and then go back to the first. That way you always have three recent backups. If your recent changes made matters worse, you can then go back to the older version.
* *Pay attention to the backup direction.* Backing up involves copying files from one place to another. It is important that you do this right—that is, copy from your work location to the backup location. If you do it the wrong way, you will overwrite a newer file with an older version.
* *Check your backups once in a while*. Double-check that your backups are where you think they are. There is nothing more frustrating than to find out that the backups are not there when you need them.
* *Relax, then restore.* When you lose a file and need to restore it from a backup, you are likely to be in an unhappy, nervous state. Take a deep breath and think through the recovery process before you start. It is not uncommon for an agitated computer user to wipe out the last backup when trying to restore a damaged file.