

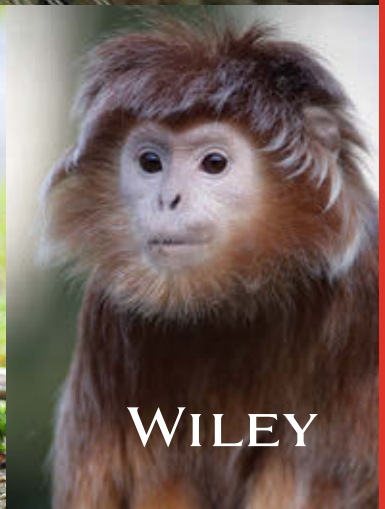
Cay Horstmann

Big Java

6/e

Early Objects

Includes Java 8 coverage



WILEY

Class Declaration

```
public class CashRegister
{
    private int itemCount;
    private double totalPrice;
}

public void addItem(double price)
{
    itemCount++;
    totalPrice = totalPrice + price;
}
...
}
```

Instance variables

Method

Selected Operators and Their Precedence

(See Appendix B for the complete list.)

[]	Array element access
++ -- !	Increment, decrement, Boolean <i>not</i>
* / %	Multiplication, division, remainder
+ -	Addition, subtraction
< <= > >=	Comparisons
== !=	Equal, not equal
&&	Boolean <i>and</i>
	Boolean <i>or</i>
=	Assignment

Conditional Statement

```
if (floor >= 13)
{
    actualFloor = floor - 1;
}
else if (floor >= 0)
{
    actualFloor = floor;
}
else
{
    System.out.println("Floor negative");
}
```

Condition

Executed when condition is true

Second condition (optional)

Executed when all conditions are false (optional)

Loop Statements

```
while (balance < TARGET)
{
    year++;
    balance = balance * (1 + rate / 100);
}
```

Condition

Executed while condition is true

```
for (int i = 0; i < 10; i++)
{
    System.out.println(i);
}
```

Initialization Condition Update

Variable and Constant Declarations

```
Type Name Initial value
int cansPerPack = 6;
final double CAN_VOLUME = 0.335;
```

Method Declaration

```
Modifiers Return type Parameter type and name
public static double cubeVolume(double sideLength)
{
    double volume = sideLength * sideLength * sideLength;
    return volume;
}
```

Exits method and returns result.

Mathematical Operations

Math.pow(x, y)	Raising to a power x^y
Math.sqrt(x)	Square root \sqrt{x}
Math.log10(x)	Decimal log $\log_{10}(x)$
Math.abs(x)	Absolute value $ x $
Math.sin(x)	Sine, cosine, tangent of x (x in radians)
Math.cos(x)	
Math.tan(x)	

String Operations

```
String s = "Hello";
int n = s.length(); // 5
char ch = s.charAt(1); // 'e'
String t = s.substring(1, 4); // "ell"
String u = s.toUpperCase(); // "HELLO"
if (u.equals("HELLO")) ... // Use equals, not ==
for (int i = 0; i < s.length(); i++)
{
    char ch = s.charAt(i);
    Process ch
}
```

Process ch

```
do
{
    System.out.print("Enter a positive integer: ");
    input = in.nextInt();
}
while (input <= 0);
```

Loop body executed at least once

Set to a new element in each iteration

```
for (double value : values)
{
    sum = sum + value;
}
```

An array or collection

Executed for each element



Big Java

6/e

Early Objects

Cay Horstmann

San Jose State University

WILEY

VICE PRESIDENT AND EXECUTIVE PUBLISHER	Laurie Rosatone
DIRECTOR	Don Fowley
EXECUTIVE EDITOR	Bryan Gambrel
EDITORIAL PROGRAM ASSISTANT	Jessy Moor
MARKETING MANAGER	Dan Sayre
SENIOR PRODUCT DESIGNER	Jennifer Welter
DESIGN DIRECTOR	Harry Nolan
SENIOR DESIGNER	Madelyn Lesure
SENIOR PHOTO EDITOR	Billy Ray
SENIOR CONTENT EDITOR	Karoline Luciano
SENIOR PRODUCTION EDITOR	Tim Lindner
PRODUCTION MANAGEMENT SERVICES	Cindy Johnson
COVER DESIGN	Madelyn Lesure
COVER PHOTOS	(tiger) Aprison Photography/Getty Images, Inc.; (rhino) irawansubingarphotography/Getty Images, Inc.; (bird) Nengloveyou/Shutterstock; (monkey) © Ehlers/iStockphoto.

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.

This book is printed on acid-free paper. ∞

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/citizenship.

Copyright © 2015 John Wiley & Sons, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, 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, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the Web at 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, or online at: www.wiley.com/go/permissions.

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 instructions and a free of charge return shipping label are available at: www.wiley.com/go/returnlabel. Outside of the United States, please contact your local representative.

ISBN 978-1-119-05628-7

ISBN-BRV 978-1-119-05644-7

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PREFACE

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 programming 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 instructions. 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. Additional Worked Examples are available online.

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
- 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

Practice makes perfect.

Of course, programming students need to be able to implement nontrivial programs, but they first need to have the confidence that they can succeed. This book contains a substantial number of self-check questions at the end of each section. “Practice It” pointers suggest exercises to try after each section. And additional practice opportunities, including automatically-graded programming exercises and skill-oriented multiple-choice questions, are available online.

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.

Visual features help the reader with navigation.

Focus on the essentials while being technically accurate.

An encyclopedic coverage is not helpful for a beginning programmer, but neither is the opposite—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 programs 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 programming in applied fields.

New to This Edition

Updated for Java 8

Java 8 introduces many exciting features, and this edition has been updated to take advantage of them. Interfaces can now have default and static methods, and lambda expressions make it easy to provide instances of interfaces with a single method. The chapter on interfaces and the sections that cover sorting have been updated to make these innovations optionally available. A new chapter covers the Java 8 stream library and its applications for “big data” processing.

In addition, Java 7 features such as the try-with-resources statement are now integrated into the text. Chapter 21 covers the utilities provided by the `Paths` and `Files` classes.

Interactive Learning

Additional interactive content is available that integrates with this text and immerses students in activities designed to foster in-depth learning. Students don’t just watch

animations and code traces, they work on generating them. The activities provide instant feedback to show students what they did right and where they need to study more. To find out more about how to make this content available in your course, visit <http://wiley.com/go/bjeo6interactivities>.

2. Consider the following code segment:

```

if (hour < 21)
{
    response = "Goodbye";
}
else
{
    response = "Goodnight";
}

```

Determine the value of `response` when `hour` has the values given in the table below.

Complete the second column. Press Enter to submit each entry.

hour	response
20	
22	
21	
3	

0 errors

6. Assume that weekdays are coded as 0 = Monday, 1 = Tuesday, ..., 4 = Friday, 5 = Saturday, 6 = Sunday. Rearrange the lines of code so that `weekday` is set to the next working day (Monday through Friday). Not all lines are useful.

Order the statements by dragging them into the left window. Use the guidelines for proper indenting and use of braces.

Code Snippet	Target Window
<code>if (weekday < 4)</code>	<code>if (weekday < 5)</code>
<code>{</code>	<code>weekday = 1;</code>
<code> weekday++;</code>	<code>weekday = 0;</code>
<code>}</code>	
<code>else</code>	
<code>{</code>	
<code>}</code>	

0 errors

InterActivities

1. In this activity, observe the inputs. They denote hours in "military time" between 0 and 23. For each input, click on the appropriate line inside the `if` statement.

Please click on the next line.

```

int hour = in.nextInt();
if (hour < 12)
{
    greeting = "Good morning";
}
else
{
    greeting = "Good afternoon";
}

```

hour	greeting
14	Good morning
18	

0 errors

[Start over](#)

“CodeCheck” is an innovative online service that students can use to work on programming problems. You can assign exercises that have already been prepared, and you can easily add your own. Visit <http://codecheck.it> to learn more and to try it out.

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 introduces a very simple subset of the UML notation. The discussion of polymorphism and inheritance is split into two chapters. 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

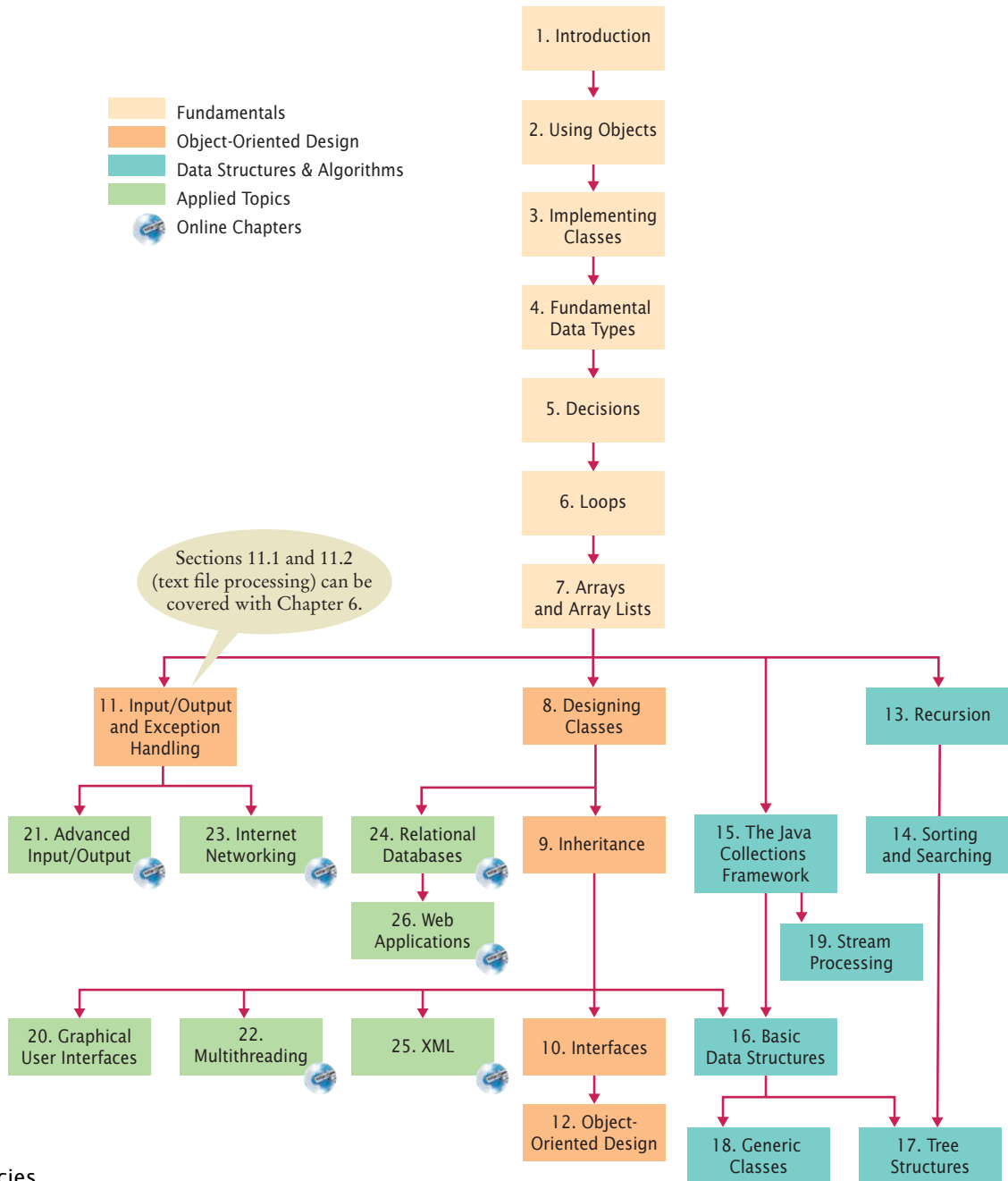


Figure 1
Chapter Dependencies

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 implement 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 20–26)



Chapters 20 through 26 cover Java programming techniques that definitely go beyond a first course in Java (21–26 are on the book's companion site). 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 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, multithreading, and those technologies that are of particular interest to server-side programming: networking, databases, XML, and web applications. The Internet has made it possible to 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 assignments. 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. Appendices F–J are available on the Web.

- | | |
|---|------------------------|
| A. The Basic Latin and Latin-1 Subsets of Unicode | F. Tool Summary |
| B. Java Operator Summary | G. Number Systems |
| C. Java Reserved Word Summary | H. UML Summary |
| D. The Java Library | I. Java Syntax Summary |
| E. Java Language Coding Guidelines | J. HTML Summary |

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, **margin notes** show where new concepts are introduced and provide an outline of key ideas.

Additional **full code examples** provides complete programs for students to run and modify.

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

Annotations explain required components and point to more information on common errors or best practices associated with the syntax.

250 Chapter 6 Loops

6.3 The for Loop

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

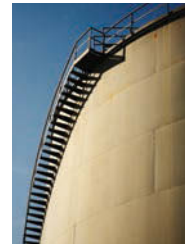
It often happens that you want to execute a sequence of statements a given number of times. You can use a `while` loop that is controlled by a counter, as in the following example:

```
int counter = 1; // Initialize the counter
while (counter <= 10) // Check the counter
{
    System.out.println(counter);
    counter++; // Update the counter
}
```

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

```
for (int counter = 1; counter <= 10; counter++)
{
    System.out.println(counter);
}
```

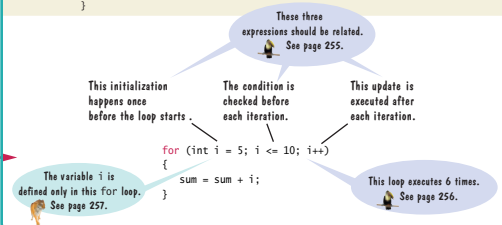
Some people call this loop *count-controlled*. In contrast, the `while` loop of the preceding section can be called an *event-controlled* loop because it executes until an event occurs; namely that the balance reaches the target. Another commonly used term for a count-controlled loop is *definite*. You know from the outset that the loop body will be executed a definite 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 *indefinite*.



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

Syntax 6.2 for Statement

Syntax `for (initialization; condition; update)`
`{`
`statements`
`}`



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.



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

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.

7.5 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 number of the month with the highest temperature. For example, according to 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 temperature (45.7 degrees Celsius) is July, and the program should display 7.



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).
- Incrementing a counter.

If you can't figure out what needs to go inside the loop, start by writing down the steps that

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.



WORKED EXAMPLE 6.1

Credit Card Processing

Learn how to use a loop to remove spaces from a credit card number. Go to wiley.com/go/bjoe6examples and download Worked Example 6.1.



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.

Table 1 Variable Declarations in Java

Variable Name	Comment
<code>int width = 20;</code>	Declares an integer variable and initializes it with 20.
<code>int perimeter = 4 * width;</code>	The initial value need not be a fixed value. (Of course, width must have been previously declared.)
<code>String greeting = "Hi!";</code>	This variable has the type <code>String</code> and is initialized with the string "Hi".
<code>height = 30;</code>	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.
<code>int width = "20";</code>	Error: You cannot initialize a number with the string "20". (Note the quotation marks.)
<code>int width;</code>	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1 on page 40.
<code>int width, height;</code>	Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement.

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

This means “compute the value of $width + 10$ 1 and store that value in the variable $width$ 2” (see Figure 4).
 In Java, it is not a problem that the variable $width$ is used on both sides of the $=$ symbol. Of course, in mathematics, the equation $width = width + 10$ has no solution.

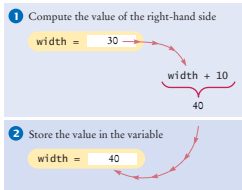
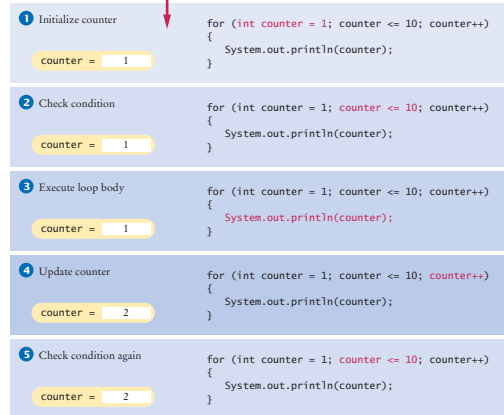


Figure 4
 Executing the Statement
 $width = width + 10$

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

Figure 3
 Execution of a
 for Loop



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

Self-check exercises at the end of each section are designed to make students think through the new material—and can spark discussion in lecture.



- Write the for loop of the Investment class as a while loop.
- How many numbers does this loop print?

```
for (int n = 10; n >= 0; n--)
{
    System.out.println(n);
}
```
- Write a for loop that prints all even numbers between 10 and 20 (inclusive).
- Write a for loop that computes the sum of the integers from 1 to n.

Practice It Now you can try these exercises at the end of the chapter: R6.4, R6.10, E6.8, E6.12.

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

• **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.

CANADA	CAD	1.09910	1.09910
CHINA	CNY	0.73769	1.60910
EURO	EUR	0.55644	1.05100
JAPAN	JPY	110.800	1.09910
SINGAPORE	SGD	1.3712	1.09910

• **Science P6.15** Radioactive decay of radioactive materials can be modeled by the equation $A = A_0 e^{-(\ln 2)t/b}$, where A is the amount of the material at time t , A_0 is the amount at time 0, and b 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.



section_1/Investment.java

```

1  /**
2  A class to monitor the growth of an investment that
3  accumulates interest at a fixed annual rate.
4  */
5  public class Investment
6  {
7      private double balance;
8      private double rate;
9      private int year;
10
11     /**
12     Constructs an Investment object from a starting balance and
13     interest rate.
14     @param aBalance the starting balance
15     @param aRate the interest rate in percent
16     */
17     public Investment(double aBalance, double aRate)
18     {
19         balance = aBalance;
20         rate = aRate;
21         year = 0;
22     }
23
24     /**
25     Keeps accumulating interest until a target balance has
26     been reached.
27     @param targetBalance the desired balance
28     */

```

Program listings are carefully designed for easy reading, going well beyond simple color coding. Methods are set off by a subtle outline.

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 program'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` variable is set to 80,000 and `status` is set to `MARRIED`. Then the `getTax` method is called. In lines 31 and 32 of `TaxReturn.java`, `tax1` and `tax2` are initialized to 0.

```

29 public double getTax()
30 {
31     double tax1 = 0;
32     double tax2 = 0;
33
34     if (status == SINGLE)
35     {
36         if (income <= RATE1_SINGLE_LIMIT)
37         {
38             tax1 = RATE1 * income;
39         }
40     }
41     else
42     {
43         tax1 = RATE1 * RATE1_SINGLE_LIMIT;
44         tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
45     }
46 }
    
```

Because `status` is not `SINGLE`, we move to the `else` branch of the outer `if` statement (line 46).

income	status	tax1	tax2
80000	MARRIED	0	0

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

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

Special Topic 11.2 **File Dialog Boxes**

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

The `JFileChooser` class has many options to fine-tune the display of the dialog box, but in its most basic form it is quite simple: Construct a file chooser object, then call the `showOpenDialog` or `showSaveDialog` method. Both methods show the same dialog box, but the button for selecting a file 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 file, or `JFileChooser.CANCEL_OPTION`, if the user canceled the selection. If a file was chosen, then you call the `getSelectedFile` method to obtain a `File` object that describes the file. Here is a complete example:

```

JFileChooser chooser = new JFileChooser();
    
```

Java 8 Notes provide detail about new features in Java 8.

Java 8 Note 10.4 **Lambda Expressions**

In the preceding section, you saw how to use interfaces for specifying variations in behavior. The `average` method needs to measure each object, and it does so by calling the `measure` method of the supplied `Measurer` object.

Unfortunately, the caller of the `average` method has to do a fair amount of work; namely, to define a class that implements the `Measurer` interface and to construct an object of that class. Java 8 has a convenient shortcut for these steps, provided that the interface has a *single abstract method*. Such an interface is called a *functional interface* because its purpose is to define a single function. The `Measurer` interface is an example of a functional interface.

To specify that single function, you can use a *lambda expression*, an expression that defines the parameters and return value of a method in a compact notation. Here is an example:

```

(Object obj) -> ((BankAccount) obj).getBalance()
    
```

This expression defines a function that, given an object, casts it to a `BankAccount` and returns the balance.

Computing & Society presents social and historical topics on computing—for interest and to fulfill the “historical and social context” requirements of the ACM/IEEE curriculum guidelines.

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 (electronic numerical integrator and computer), 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 has several computers—to control the engine, brakes, lights, and the radio.

The advent of ubiquitous computing changed many aspects of our lives. Factories used to employ people to do repetitive assembly tasks that are today carried out by computer-controlled robots, operated by a few people who know how to work with those computers. Books, music, and movies are nowadays often consumed on computers, and computers are almost always involved in their production. The book that you are reading right now could not have been written without computers.

This transit card contains a computer.

Acknowledgments

Many thanks to Bryan Gambrel, Don Fowley, Jenny Welter, Jessy Moor, Jennifer Lartz, Billy Ray, and Tim Lindner 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.

I am grateful to Jose Cordova, *The University of Louisiana at Monroe*, Suzanne Dietrich, *Arizona State University, West Campus*, Mike Domaratzki, *University of Manitoba*, Guy Helmer, *Iowa State University*, Peter Lutz, *Rochester Institute of Technology*, Carolyn Schauble, *Colorado State University*, Brent Seales, *University of Kentucky*, and Brent Wilson, *George Fox University* for their excellent contributions to the supplementary materials.

Many thanks to the individuals who reviewed the manuscript for this edition, made valuable suggestions, and brought an embarrassingly large number of errors and omissions to my attention. They include:

Robin Carr, *Drexel University*
 Gerald Cohen, *The Richard Stockton College of New Jersey*
 Aaron Keen, *California Polytechnic State University, San Luis Obispo*
 Aurelia Smith, *Columbus State University*
 Aakash Taneja, *The Richard Stockton College of New Jersey*
 Craig Tanis, *University of Tennessee at Chattanooga*
 Katherine Winters, *University of Tennessee at Chattanooga*

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

Eric Aaron, <i>Wesleyan University</i>	John Bundy, <i>DeVry University Chicago</i>	Lennie Cooper, <i>Miami Dade College</i>
James Agnew, <i>Anne Arundel Community College</i>	Robert P. Burton, <i>Brigham Young University</i>	Jose Cordova, <i>University of Louisiana, Monroe</i>
Tim Andersen, <i>Boise State University</i>	Frank Butt, <i>IBM</i>	Valentino Crespi, <i>California State University, Los Angeles</i>
Ivan Bajic, <i>San Diego State University</i>	Jerry Cain, <i>Stanford University</i>	Jim Cross, <i>Auburn University</i>
Greg Ballinger, <i>Miami Dade College</i>	Adam Cannon, <i>Columbia University</i>	Russell Deaton, <i>University of Arkansas</i>
Ted Bangay, <i>Sheridan Institute of Technology</i>	Michael Carney, <i>Finger Lakes Community College</i>	Geoffrey Decker, <i>Northern Illinois University</i>
Ian Barland, <i>Radford University</i>	Christopher Cassa, <i>Massachusetts Institute of Technology</i>	H. E. Dunsmore, <i>Purdue University</i>
George Basham, <i>Franklin University</i>	Nancy Chase, <i>Gonzaga University</i>	Robert Duvall, <i>Duke University</i>
Jon Beck, <i>Truman State University</i>	Dr. Suchindran S. Chatterjee, <i>Arizona State University</i>	Sherif Elfayoumy, <i>University of North Florida</i>
Sambit Bhattacharya, <i>Fayetteville State University</i>	Archana Chidanandan, <i>Rose-Hulman Institute of Technology</i>	Eman El-Sheikh, <i>University of West Florida</i>
Rick Birney, <i>Arizona State University</i>	Vincent Cicirello, <i>The Richard Stockton College of New Jersey</i>	Henry A. Etlinger, <i>Rochester Institute of Technology</i>
Paul Bladdek, <i>Edmonds Community College</i>	Teresa Cole, <i>Boise State University</i>	John Fendrich, <i>Bradley University</i>
Matt Boutell, <i>Rose-Hulman Institute of Technology</i>	Deborah Coleman, <i>Rochester Institute of Technology</i>	David Freer, <i>Miami Dade College</i>
Joseph Bowbeer, <i>Vizrea Corporation</i>	Tina Comston, <i>Franklin University</i>	John Fulton, <i>Franklin University</i>
Timothy A. Budd, <i>Oregon State University</i>		David Geary, <i>Sabreware, Inc.</i>
		Margaret Geroch, <i>Wheeling Jesuit University</i>

xiv Acknowledgments

- 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
Eileen Head, *Binghamton University*
Cecily Heiner, *University of Utah*
Guy Helmer, *Iowa State University*
Ed Holden, *Rochester Institute of Technology*
Brian Howard, *DePaul 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*
Mugdha Khaladkar, *New Jersey Institute of Technology*
Gary J. Koehler, *University of Florida*
Elliot Koffman, *Temple University*
Ronald Krawitz, *DeVry University*
Norm Krumpke, *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*
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*
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*
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*
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*

CONTENTS

PREFACE **iii**

SPECIAL FEATURES **xxiv**

1 INTRODUCTION **1**

- 1.1 Computer Programs **2**
- 1.2 The Anatomy of a Computer **3**
- 1.3 The Java Programming Language **6**
- 1.4 Becoming Familiar with Your Programming Environment **7**
- 1.5 Analyzing Your First Program **11**
- 1.6 Errors **14**
- 1.7 PROBLEM SOLVING: Algorithm Design **15**
 - The Algorithm Concept 16
 - An Algorithm for Solving an Investment Problem 17
 - Pseudocode 18
 - From Algorithms to Programs 18
 - HT1** Describing an Algorithm with Pseudocode 19
 - WE1** Writing an Algorithm for Tiling a Floor 21

2 USING OBJECTS **31**

- 2.1 Objects and Classes **32**
 - Using Objects 32
 - Classes 33
- 2.2 Variables **34**
 - Variable Declarations 34
 - Types 36
 - Names 37
 - Comments 38
 - Assignment 38
- 2.3 Calling Methods **41**
 - The Public Interface of a Class 41
 - Method Arguments 42
 - Return Values 43
 - Method Declarations 45
- 2.4 Constructing Objects **46**


2.5 Accessor and Mutator Methods **48**

2.6 The API Documentation **50**

- Browsing the API Documentation 50
- Packages 52

2.7 Implementing a Test Program **53**

ST1 Testing Classes in an Interactive Environment 54

WE1 How Many Days Have You Been Alive? 

WE2 Working with Pictures 

2.8 Object References **55**

2.9 Graphical Applications **59**

Frame Windows 59

Drawing on a Component 60

Displaying a Component in a Frame 63

2.10 Ellipses, Lines, Text, and Color **64**

Ellipses and Circles 64

Lines 65

Drawing Text 65

Colors 66

3 IMPLEMENTING CLASSES **79**

3.1 Instance Variables and Encapsulation **80**

Instance Variables 80

The Methods of the Counter Class 82

Encapsulation 82

3.2 Specifying the Public Interface of a Class **84**

Specifying Methods 84

Specifying Constructors 85

Using the Public Interface 87

Commenting the Public Interface 87


3.3 Providing the Class Implementation **91**

Providing Instance Variables 91

Providing Constructors 92

Providing Methods 93



HT1 Implementing a Class 96

WE1 Making a Simple Menu 


3.4 Unit Testing **100**

- 3.5 PROBLEM SOLVING Tracing Objects **103**
- 3.6 Local Variables **105**
- 3.7 The this Reference **107**
 - ST1 Calling One Constructor from Another 110
- 3.8 Shape Classes **110**
 - HT2 Drawing Graphical Shapes 114

4 FUNDAMENTAL DATA TYPES **129**





- 4.1 Numbers **130**
 - Number Types 130
 - Constants 132
 - ST1 Big Numbers 136
- 4.2 Arithmetic **137**
 - Arithmetic Operators 137
 - Increment and Decrement 138
 - Integer Division and Remainder 138
 - Powers and Roots 139
 - Converting Floating-Point Numbers to Integers 140
 - J81 Avoiding Negative Remainders 143
 - ST2 Combining Assignment and Arithmetic 143
 - ST3 Instance Methods and Static Methods 143
- 4.3 Input and Output **145**
 - Reading Input 145
 - Formatted Output 146
 - HT1 Carrying Out Computations 149
 - WE1 Computing the Volume and Surface Area of a Pyramid 
- 4.4 PROBLEM SOLVING First Do it By Hand **152**
 - WE2 Computing Travel Time 
- 4.5 Strings **154**
 - The String Type 154
 - Concatenation 155
 - String Input 155
 - Escape Sequences 156
 - Strings and Characters 156
 - Substrings 157
 - ST4 Using Dialog Boxes for Input and Output 160

5 DECISIONS **177**

- 5.1 The if Statement **178**
 - ST1 The Conditional Operator 182
- 5.2 Comparing Values **183**
 - Relational Operators 184
 - Comparing Floating-Point Numbers 185
 - Comparing Strings 186
 - Comparing Objects 187
 - Testing for null 187
 - HT1 Implementing an if Statement 190
 - WE1 Extracting the Middle 
- 5.3 Multiple Alternatives **193**
 - ST2 The switch Statement 196
- 5.4 Nested Branches **196**
 - ST3 Block Scope 201
 - ST4 Enumeration Types 203
- 5.5 PROBLEM SOLVING Flowcharts **203**
- 5.6 PROBLEM SOLVING Selecting Test Cases **206**
 - ST5 Logging 208
- 5.7 Boolean Variables and Operators **209**
 - ST6 Short-Circuit Evaluation of Boolean Operators 213
 - ST7 De Morgan's Law 213
- 5.8 APPLICATION Input Validation **214**

6 LOOPS **237**

- 6.1 The while Loop **238**
- 6.2 PROBLEM SOLVING Hand-Tracing **245**
- 6.3 The for Loop **250**
 - ST1 Variables Declared in a for Loop Header 257
- 6.4 The do Loop **258**
- 6.5 APPLICATION Processing Sentinel Values **259**
 - ST2 Redirection of Input and Output 262
 - ST3 The “Loop and a Half” Problem 262
 - ST4 The break and continue Statements 263
- 6.6 PROBLEM SOLVING Storyboards **265**
- 6.7 Common Loop Algorithms **268**
 - Sum and Average Value 268
 - Counting Matches 268

- Finding the First Match 269
 - Prompting Until a Match is Found 270
 - Maximum and Minimum 270
 - Comparing Adjacent Values 271
 - HT1** Writing a Loop 272
 - WE1** Credit Card Processing 
 - 6.8** Nested Loops **275**
 - WE2** Manipulating the Pixels in an Image 
 - 6.9** **APPLICATION** Random Numbers and Simulations **279**
 - Generating Random Numbers 279
 - The Monte Carlo Method 281
 - 6.10** **Using a Debugger** **282**
 - HT2** Debugging 285
 - WE3** A Sample Debugging Session 
- 7** **ARRAYS AND ARRAY LISTS** **307**
- 7.1** Arrays **308**
 - Declaring and Using Arrays 308
 - Array References 311
 - Using Arrays with Methods 312
 - Partially Filled Arrays 312
 - ST1** Methods with a Variable Number of Arguments 315
 - 7.2** The Enhanced for Loop **317**
 - 7.3** Common Array Algorithms **318**
 - Filling 318
 - Sum and Average Value 319
 - Maximum and Minimum 319
 - Element Separators 319
 - Linear Search 320
 - Removing an Element 320
 - Inserting an Element 321
 - Swapping Elements 322
 - Copying Arrays 323
 - Reading Input 324
 - ST2** Sorting with the Java Library 327
 - 7.4** **PROBLEM SOLVING** Adapting Algorithms **327**
 - HT1** Working with Arrays 330
 - WE1** Rolling the Dice 
 - 7.5** **PROBLEM SOLVING** Discovering Algorithms by Manipulating Physical Objects **332**
 - 7.6** Two-Dimensional Arrays **336**
 - Declaring Two-Dimensional Arrays 336
 - Accessing Elements 337
 - Locating Neighboring Elements 338
 - Accessing Rows and Columns 338
 - WE2** A World Population Table 
 - ST3** Two-Dimensional Arrays with Variable Row Lengths 341
 - ST4** Multidimensional Arrays 343
 - 7.7** Array Lists **343**
 - Declaring and Using Array Lists 344
 - Using the Enhanced for Loop with Array Lists 345
 - Copying Array Lists 346
 - Wrappers and Auto-boxing 347
 - Using Array Algorithms with Array Lists 348
 - Storing Input Values in an Array List 348
 - Removing Matches 348
 - Choosing Between Array Lists and Arrays 349
 - ST5** The Diamond Syntax 352
 - 7.8** **Regression Testing** **352**
- 8** **DESIGNING CLASSES** **375**
- 8.1** Discovering Classes **376**
 - 8.2** Designing Good Methods **377**
 - Providing a Cohesive Public Interface 377
 - Minimizing Dependencies 378
 - Separating Accessors and Mutators 379
 - Minimizing Side Effects 380
 - ST1** Call by Value and Call by Reference 382
 - 8.3** **PROBLEM SOLVING** Patterns for Object Data **386**
 - Keeping a Total 386
 - Counting Events 387
 - Collecting Values 387
 - Managing Properties of an Object 388
 - Modeling Objects with Distinct States 388
 - Describing the Position of an Object 389
 - 8.4** Static Variables and Methods **391**
 - ST2** Alternative Forms of Instance and Static Variable Initialization 394
 - ST3** Static Imports 395
 - 8.5** **PROBLEM SOLVING** Solve a Simpler Problem First **395**

8.6 Packages 400
 Organizing Related Classes into Packages 400
 Importing Packages 401
 Package Names 401
 Packages and Source Files 402
ST4 Package Access 403
HT1 Programming with Packages 404

8.7 Unit Test Frameworks 405

9 INHERITANCE 423

9.1 Inheritance Hierarchies 424
9.2 Implementing Subclasses 428
9.3 Overriding Methods 433
ST1 Calling the Superclass Constructor 438
9.4 Polymorphism 439
ST2 Dynamic Method Lookup and the Implicit Parameter 442
ST3 Abstract Classes 443
ST4 Final Methods and Classes 444
ST5 Protected Access 444
HT1 Developing an Inheritance Hierarchy 445
WE1 Implementing an Employee Hierarchy for Payroll Processing

9.5 Object: The Cosmic Superclass 450
 Overriding the toString Method 450
 The equals Method 452
 The instanceof Operator 453
ST6 Inheritance and the toString Method 455
ST7 Inheritance and the equals Method 456

10 INTERFACES 465

10.1 Using Interfaces for Algorithm Reuse 466
 Discovering an Interface Type 466
 Declaring an Interface Type 467
 Implementing an Interface Type 469
 Comparing Interfaces and Inheritance 471
ST1 Constants in Interfaces 473
J81 Static Methods in Interfaces 473
J82 Default Methods 473
J83 Conflicting Default Methods 474

10.2 Working with Interface Variables 475
 Converting from Classes to Interfaces 475
 Invoking Methods on Interface Variables 476
 Casting from Interfaces to Classes 476
WE1 Investigating Number Sequences

10.3 The Comparable Interface 477
ST2 The clone Method and the Cloneable Interface 479

10.4 Using Interfaces for Callbacks 482
J84 Lambda Expressions 485
ST3 Generic Interface Types 486

10.5 Inner Classes 487
ST4 Anonymous Classes 488

10.6 Mock Objects 489

10.7 Event Handling 490
 Listening to Events 491
 Using Inner Classes for Listeners 493
J85 Lambda Expressions for Event Handling 496

10.8 Building Applications with Buttons 496


10.9 Processing Timer Events 499

10.10 Mouse Events 502
ST5 Keyboard Events 506
ST6 Event Adapters 506

11 INPUT/OUTPUT AND EXCEPTION HANDLING 519

11.1 Reading and Writing Text Files 520
ST1 Reading Web Pages 523
ST2 File Dialog Boxes 523
ST3 Character Encodings 524

11.2 Text Input and Output 525
 Reading Words 525
 Reading Characters 526
 Classifying Characters 526
 Reading Lines 527
 Scanning a String 528
 Converting Strings to Numbers 528
 Avoiding Errors When Reading Numbers 529
 Mixing Number, Word, and Line Input 529
 Formatting Output 530
ST4 Regular Expressions 532
ST5 Reading an Entire File 533

11.3 Command Line Arguments 533**HT1** Processing Text Files 536**WE1** Analyzing Baby Names **11.4 Exception Handling 540**

Throwing Exceptions 540

Catching Exceptions 542

Checked Exceptions 543

Closing Resources 545

Designing Your Own Exception Types 546

ST6 Assertions 549**ST7** The try/finally Statement 549**11.5 APPLICATION Handling Input Errors 549****12 OBJECT-ORIENTED DESIGN 565****12.1 Classes and Their Responsibilities 566**

Discovering Classes 566

The CRC Card Method 567

12.2 Relationships Between Classes 569

Dependency 569

Aggregation 570

Inheritance 571

HT1 Using CRC Cards and UML Diagrams in Program Design 572**ST1** Attributes and Methods in UML Diagrams 573**ST2** Multiplicities 574**ST3** Aggregation, Association, and Composition 574**12.3 APPLICATION Printing an Invoice 575**




Requirements 575

CRC Cards 576

UML Diagrams 578

Method Documentation 579

Implementation 581

WE1 Simulating an Automatic Teller Machine **13 RECURSION 593****13.1 Triangle Numbers 594****HT1** Thinking Recursively 599**WE1** Finding Files **13.2 Recursive Helper Methods 602****13.3 The Efficiency of Recursion 604****13.4 Permutations 609****13.5 Mutual Recursion 614****13.6 Backtracking 620****WE2** Towers of Hanoi **14 SORTING AND SEARCHING 635****14.1 Selection Sort 636****14.2 Profiling the Selection Sort Algorithm 639****14.3 Analyzing the Performance of the Selection Sort Algorithm 642****ST1** Oh, Omega, and Theta 644**ST2** Insertion Sort 645**14.4 Merge Sort 647****14.5 Analyzing the Merge Sort Algorithm 650****ST3** The Quicksort Algorithm 652**14.6 Searching 654**

Linear Search 654

Binary Search 655

14.7 PROBLEM SOLVING Estimating the Running Time of an Algorithm 659

Linear Time 659

Quadratic Time 660

The Triangle Pattern 661

Logarithmic Time 662

14.8 Sorting and Searching in the Java Library 664






Sorting 664


Binary Search 664

Comparing Objects 665

ST4 The Comparator Interface 666**J81** Comparators with Lambda Expressions 667**WE1** Enhancing the Insertion Sort Algorithm **15 THE JAVA COLLECTIONS FRAMEWORK 677****15.1 An Overview of the Collections Framework 678**



xx Contents

- 15.2 Linked Lists 681**
 - The Structure of Linked Lists 681
 - The `LinkedList` Class of the Java Collections Framework 682
 - List Iterators 683
 - 15.3 Sets 687**
 - Choosing a Set Implementation 687
 - Working with Sets 688
 - 15.4 Maps 692**
 - J81** Updating Map Entries 694
 - HT1** Choosing a Collection 694
 - WE1** Word Frequency 
 - ST1** Hash Functions 696
 - 15.5 Stacks, Queues, and Priority Queues 698**
 - Stacks 698
 - Queues 699
 - Priority Queues 699
 - 15.6 Stack and Queue Applications 701**
 - Balancing Parentheses 701
 - Evaluating Reverse Polish Expressions 702
 - Evaluating Algebraic Expressions 703
 - Backtracking 706
 - WE2** Simulating a Queue of Waiting Customers 
 - ST2** Reverse Polish Notation 709
- 16 BASIC DATA STRUCTURES 721**
- 16.1 Implementing Linked Lists 722**
 - The Node Class 722
 - Adding and Removing the First Element 723
 - The Iterator Class 724
 - Advancing an Iterator 725
 - Removing an Element 726
 - Adding an Element 728
 - Setting an Element to a Different Value 729
 - Efficiency of Linked List Operations 729
 - ST1** Static Classes 736
 - WE1** Implementing a Doubly-Linked List 
 - 16.2 Implementing Array Lists 737**
 - Getting and Setting Elements 737
 - Removing or Adding Elements 738
 - Growing the Internal Array 739
 - 16.3 Implementing Stacks and Queues 741**
 - Stacks as Linked Lists 741
 - Stacks as Arrays 743
 - Queues as Linked Lists 743
 - Queues as Circular Arrays 744
 - 16.4 Implementing a Hash Table 747**
 - Hash Codes 747
 - Hash Tables 747
 - Finding an Element 749
 - Adding and Removing Elements 749
 - Iterating over a Hash Table 750
 - ST2** Open Addressing 755
- 17 TREE STRUCTURES 765**
- 17.1 Basic Tree Concepts 766**
 - 17.2 Binary Trees 770**
 - Binary Tree Examples 770
 - Balanced Trees 772
 - A Binary Tree Implementation 773
 - WE1** Building a Huffman Tree 
 - 17.3 Binary Search Trees 775**
 - The Binary Search Property 775
 - Insertion 776
 - Removal 778
 - Efficiency of the Operations 780
 - 17.4 Tree Traversal 784**
 - Inorder Traversal 784
 - Preorder and Postorder Traversals 785
 - The Visitor Pattern 786
 - Depth-First and Breadth-First Search 787
 - Tree Iterators 789
 - 17.5 Red-Black Trees 790**
 - Basic Properties of Red-Black Trees 790
 - Insertion 792
 - Removal 793
 - WE2** Implementing a Red-Black Tree 
 - 17.6 Heaps 797**
 - 17.7 The Heapsort Algorithm 808**
- 18 GENERIC CLASSES 823**
- 18.1 Generic Classes and Type Parameters 824**
 - 18.2 Implementing Generic Types 825**

- 18.3 Generic Methods **829**
- 18.4 Constraining Type Parameters **831**
 - ST1 Wildcard Types 834
- 18.5 Type Erasure **835**
 - ST2 Reflection 838
 - WE1 Making a Generic Binary Search Tree Class 

19

 STREAM PROCESSING **845**

- 19.1 The Stream Concept **846**
- 19.2 Producing Streams **848**
- 19.3 Collecting Results **850**
 - ST1 Infinite Streams 851
- 19.4 Transforming Streams **852**
- 19.5 Lambda Expressions **855**
 - ST2 Method and Constructor References 857
 - ST3 Higher-Order Functions 858
 - ST4 Higher-Order Functions and Comparators 859
- 19.6 The Optional Type **859**
- 19.7 Other Terminal Operations **862**
- 19.8 Primitive-Type Streams **863**
 - Creating Primitive-Type Streams 864
 - Mapping a Primitive-Type Stream 864
 - Processing Primitive-Type Streams 864
- 19.9 Grouping Results **866**
- 19.10 Common Algorithms Revisited **868**
 - Filling 868
 - Sum, Average, Maximum, and Minimum 869
 - Counting Matches 869
 - Element Separators 869
 - Linear Search 870
 - Comparing Adjacent Values 870
 - HT1 Working with Streams 871
 - WE1 Word Properties 
 - WE2 A Movie Database 

20

 GRAPHICAL USER INTERFACES **883**

- 20.1 Layout Management **884**
 - Using Layout Managers 884

- Achieving Complex Layouts 885
- Using Inheritance to Customize Frames 886
- ST1 Adding the main Method to the Frame Class 888


20.2

 Processing Text Input **888**

- Text Fields 888
- Text Areas 891

20.3

 Choices **894**

- Radio Buttons 894
- Check Boxes 895
- Combo Boxes 896
- HT1 Laying Out a User Interface 901
- WE1 Programming a Working Calculator 


20.4

 Menus **905**

20.5

 Exploring the Swing Documentation **911**

21

 ADVANCED INPUT/OUTPUT (WEB ONLY) 

- 21.1 Readers, Writers, and Input/Output Streams
- 21.2 Binary Input and Output
- 21.3 Random Access
- 21.4 Object Input and Output Streams
 - HT1 Choosing a File Format
- 21.5 File and Directory Operations
 - Paths
 - Creating and Deleting Files and Directories
 - Useful File Operations
 - Visiting Directories


22

 MULTITHREADING (WEB ONLY) 


- 22.1 Running Threads
 - ST1 Thread Pools
- 22.2 Terminating Threads
- 22.3 Race Conditions
- 22.4 Synchronizing Object Access
- 22.5 Avoiding Deadlocks
 - ST2 Object Locks and Synchronized Methods
 - ST3 The Java Memory Model
- 22.6 APPLICATION Algorithm Animation

23 INTERNET NETWORKING
(WEB ONLY) 

- 23.1 The Internet Protocol
- 23.2 Application Level Protocols
- 23.3 A Client Program
- 23.4 A Server Program
 - HT1** Designing Client/Server Programs
- 23.5 URL Connections

24 RELATIONAL DATABASES
(WEB ONLY) 

- 24.1 Organizing Database Information
 - Database Tables
 - Linking Tables
 - Implementing Multi-Valued Relationships
 - ST1** Primary Keys and Indexes
- 24.2 Queries
 - Simple Queries
 - Selecting Columns
 - Selecting Subsets
 - Calculations
 - Joins
 - Updating and Deleting Data
- 24.3 Installing a Database
- 24.4 Database Programming in Java
 - Connecting to the Database
 - Executing SQL Statements
 - Analyzing Query Results
 - Result Set Metadata
- 24.5 **APPLICATION** Entering an Invoice
 - ST2** Transactions
 - ST3** Object-Relational Mapping
 - WE1** Programming a Bank Database






25 XML (WEB 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
- 25.2 Parsing XML Documents

- 25.3 Creating XML Documents
 - HT2** Writing an XML Document
 - ST1** Grammars, Parsers, and Compilers
- 25.4 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

26 WEB APPLICATIONS
(WEB ONLY) 


- 26.1 The Architecture of a Web Application
- 26.2 The Architecture of a JSF Application
 - JSF Pages
 - Managed Beans
 - Separation of Presentation and Business Logic
 - Deploying a JSF Application
 - ST1** Session State and Cookies
- 26.3 JavaBeans Components
- 26.4 Navigation Between Pages
 - HT1** Designing a Managed Bean
- 26.5 JSF Components
- 26.6 **APPLICATION** A Three-Tier Application
 - ST2** AJAX

- 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-39**
- APPENDIX F** TOOL SUMMARY 
- APPENDIX G** NUMBER SYSTEMS 
- APPENDIX H** UML SUMMARY 
- APPENDIX I** JAVA SYNTAX SUMMARY 
- APPENDIX J** HTML SUMMARY 








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





ALPHABETICAL LIST OF SYNTAX BOXES



















Arrays	309
Array Lists	343
Assignment	39
Calling a Superclass Method	433
Cast	141
Catching Exceptions	542
Class Declaration	87
Comparisons	184
Constant Declaration	134
Constructor with Superclass Initializer	438
Declaring a Generic Class	826
Declaring a Generic Method	830
Declaring an Interface	468
for Statement	250
if Statement	180
Implementing an Interface	469
Importing a Class from a Package	52
Input Statement	145
Instance Variable Declaration	81
Java Program	12
Lambda Expressions	855
Object Construction	47
Package Specification	401
Subclass Declaration	430
The Enhanced for Loop	318
The instanceof Operator	453
The throws Clause	545
The try-with-resources Statement	545
Throwing an Exception	540
Two-Dimensional Array Declaration	337
while Statement	239
Variable Declaration	35





CHAPTER	 Common Errors	 How Tos and Worked Examples
1 Introduction	Omitting Semicolons 13 Misspelling Words 15	Describing an Algorithm with Pseudocode 19 Writing an Algorithm for Tiling a Floor 21
2 Using Objects	Using Undeclared or Uninitialized Variables 40 Confusing Variable Declarations and Assignment Statements 40 Trying to Invoke a Constructor Like a Method 48	How Many Days Have You Been Alive?  Working with Pictures 
3 Implementing Classes	Declaring a Constructor as void 90 Ignoring Parameter Variables 96 Duplicating Instance Variables in Local Variables 106 Providing Unnecessary Instance Variables 106 Forgetting to Initialize Object References in a Constructor 107	Implementing a Class 96 Making a Simple Menu  Drawing Graphical Shapes 114
4 Fundamental Data Types	Unintended Integer Division 142 Unbalanced Parentheses 142	Carrying out Computations 149 Computing the Volume and Surface Area of a Pyramid  Computing Travel Time 
5 Decisions	A Semicolon After the if Condition 182 Using == to Compare Strings 189 The Dangling else Problem 201 Combining Multiple Relational Operators 212 Confusing && and Conditions 212	Implementing an if Statement 190 Extracting the Middle 
6 Loops	Don't Think "Are We There Yet?" 243 Infinite Loops 244 Off-by-One Errors 244	Writing a Loop 272 Credit Card Processing  Manipulating the Pixels in an Image  Debugging 285 A Sample Debugging Session 

 Programming Tips		 Special Topics and Java 8 Notes 		 Computing & Society	
Backup Copies	10			Computers Are Everywhere	5
Choose Descriptive Variable Names	41	Testing Classes in an Interactive Environment	54	Computer Monopoly	58
Learn By Trying	45				
Don't Memorize—Use Online Help	53				
The javadoc Utility	90	Calling One Constructor from Another	110	Electronic Voting Machines	102
Do Not Use Magic Numbers	137	Big Numbers	136	The Pentium Floating-Point Bug	144
Spaces in Expressions	143	● Avoiding Negative Remainders	143	International Alphabets and Unicode	161
Reading Exception Reports	160	Combining Assignment and Arithmetic	143		
		Instance Methods and Static Methods	143		
		Using Dialog Boxes for Input and Output	160		
Brace Layout	181	The Conditional Operator	182	Denver's Luggage Handling System	192
Always Use Braces	181	The switch Statement	196	Artificial Intelligence	217
Tabs	182	Block Scope	201		
Avoid Duplication in Branches	183	Enumeration Types	203		
Hand-Tracing	200	Logging	208		
Make a Schedule and Make Time for Unexpected Problems	208	Short-Circuit Evaluation of Boolean Operators	213		
		De Morgan's Law	213		
Use for Loops for Their Intended Purpose Only	255	Variables Declared in a for Loop Header	257	Digital Piracy	249
Choose Loop Bounds That Match Your Task	256	Redirection of Input and Output	262	The First Bug	287
Count Iterations	256	The Loop-and-a-Half Problem	262		
Flowcharts for Loops	259	The break and continue Statements	263		

CHAPTER	 Common Errors	 How Tos and Worked Examples
7 Arrays and Array Lists	Bounds Errors 314 Uninitialized and Unfilled Arrays 314 Underestimating the Size of a Data Set 327 Length and Size 352	Working with Arrays 330 Rolling the Dice  A World Population Table 
8 Designing Classes	Trying to Access Instance Variables in Static Methods 394 Confusing Dots 403	Programming with Packages 404
9 Inheritance	Replicating Instance Variables from the Superclass 432 Confusing Super- and Subclasses 432 Accidental Overloading 437 Forgetting to Use super When Invoking a Superclass Method 437 Don't Use Type Tests 454	Developing an Inheritance Hierarchy 445 Implementing an Employee Hierarchy for Payroll Processing 
10 Interfaces	Forgetting to Declare Implementing Methods as Public 472 Trying to Instantiate an Interface 472 Modifying Parameter Types in the Implementing Method 495 Trying to Call Listener Methods 495 Forgetting to Attach a Listener 498 Forgetting to Repaint 502	Investigating Number Sequences 
11 Input/Output and Exception Handling	Backslashes in File Names 523 Constructing a Scanner with a String 523	Processing Text Files 536 Analyzing Baby Names 

 Programming Tips	 Special Topics and Java 8 Notes 	 Computing & Society
Use Arrays for Sequences of Related Items 314 Make Parallel Arrays into Arrays of Objects 314 Batch Files and Shell Scripts 354	Methods with a Variable Number of Arguments 315 Sorting with the Java Library 327 Two-Dimensional Arrays with Variable Row Lengths 341 Multidimensional Arrays 343 The Diamond Syntax 352	Computer Viruses 316 The Therac-25 Incidents 355
Consistency 381 Minimize the Use of Static Methods 393	Call by Value and Call by Reference 382 Alternative Forms of Instance and Static Variable Initialization 394 Static Imports 395 Package Access 403	Personal Computing 407
Use a Single Class for Variation in Values, Inheritance for Variation in Behavior 428	Calling the Superclass Constructor 438 Dynamic Method Lookup and the Implicit Parameter 442 Abstract Classes 443 Final Methods and Classes 444 Protected Access 444 Inheritance and the toString Method 455 Inheritance and the equals Method 456	Who Controls the Internet? 456
Comparing Integers and Floating-Point Numbers 478 Don't Use a Container as a Listener 499	Constants in Interfaces 473 • Static Methods in Interfaces 473 • Default Methods 473 • Conflicting Default Methods 474 The clone Method and the Cloneable Interface 479 • Lambda Expressions 485 Generic Interface Types 486 Anonymous Classes 488 • Lambda Expressions for Event Handling 496 Keyboard Events 506 Event Adapters 506	Open Source and Free Software 507
Throw Early, Catch Late 548 Do Not Squelch Exceptions 548 Do Throw Specific Exceptions 548	Reading Web Pages 523 File Dialog Boxes 523 Character Encodings 524 Regular Expressions 532 Reading an Entire File 533 Assertions 549 The try/finally Statement 549	Encryption Algorithms 539 The Ariane Rocket Incident 554

CHAPTER	 Common Errors	 How Tos and Worked Examples
12 Object-Oriented Design		Using CRC Cards and UML Diagrams in Program Design 572 Simulating an Automatic Teller Machine 
13 Recursion	Infinite Recursion 598 Tracing Through Recursive Methods 598	Thinking Recursively 599  Finding Files  Towers of Hanoi 
14 Sorting and Searching	The compareTo Method Can Return Any Integer, Not Just -1, 0, and 1 666	Enhancing the Insertion Sort Algorithm 
15 The Java Collections Framework		Choosing a Collection 694  Word Frequency  Simulating a Queue of Waiting Customers 
16 Basic Data Structures		Implementing a Doubly-Linked List 
17 Tree Structures		Building a Huffman Tree  Implementing a Red-Black Tree 
18 Generic Classes	Genericity and Inheritance 833 The Array Store Exception 833 Using Generic Types in a Static Context 838	Making a Generic Binary Search Tree Class 
19 Stream Processing	Don't Use a Terminated Stream 854 Optional Results Without Values 861 Don't Apply Mutations in Parallel Stream Operations 863	Working with Streams 871  Word Properties  A Movie Database 
20 Graphical User Interfaces	By Default, Components Have Zero Width and Height 887	Laying Out a User Interface 901 Programming a Working Calculator 

 Programming Tips		 Special Topics and Java 8 Notes 		 Computing & Society	
		Attributes and Methods in UML Diagrams	573	Databases and Privacy	586
		Multiplicities	574		
		Aggregation, Association, and Composition	574		
				The Limits of Computation	612
		Oh, Omega, and Theta	644	The First Programmer	658
		Insertion Sort	645		
		The Quicksort Algorithm	652		
		The Comparator Interface	666		
		● Comparators with Lambda Expressions	667		
Use Interface References to Manipulate Data Structures	691	● Updating Map Entries	694	Standardization	686
		Hash Functions	696		
		Reverse Polish Notation	709		
		Static Classes	736		
		Open Addressing	755		
		Wildcard Types	834		
		Reflection	838		
One Stream Operation Per Line	851	Infinite Streams	851		
Keep Lambda Expressions Short	856	Method and Constructor References	857		
		Higher-Order Functions	858		
		Higher-Order Functions and Comparators	859		
Use a GUI Builder	904	Adding the main Method to the Frame Class	888		

CHAPTER	 Common Errors	 How Tos and Worked Examples 
21 Advanced Input/Output (WEB ONLY) 	Negative byte Values 	Choosing a File Format 
22 Multithreading (WEB ONLY) 	Calling await Without Calling signalAll  Calling signalAll Without Locking the Object 	
23 Internet Networking (WEB ONLY) 		Designing Client/Server Programs 
24 Relational Databases (WEB ONLY) 	Joining Tables Without Specifying a Link Condition  Constructing Queries from Arbitrary Strings 	Programming a Bank Database 
25 XML (WEB ONLY) 	XML Elements Describe Objects, Not Classes 	Designing an XML Document Format  Writing an XML Document  Writing a DTD 
26 Web Applications (WEB ONLY) 		Designing a Managed Bean 

 Programming Tips	 Special Topics and Java 8 Notes 	 Computing & Society
Use the Runnable Interface Check for Thread Interruptions in the run Method of a Thread	Thread Pools Object Locks and Synchronized Methods The Java Memory Model	
Use High-Level Libraries		
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	Primary Keys and Indexes Transactions Object-Relational Mapping	
Prefer XML Elements over Attributes Avoid Children with Mixed Elements and Text	Grammars, Parsers, and Compilers Schema Languages Other XML Technologies	
	Session State and Cookies AJAX	

CHAPTER 1

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

CHAPTER CONTENTS

- 1.1 COMPUTER PROGRAMS** 2
- 1.2 THE ANATOMY OF A COMPUTER** 3
 - C&S** Computers Are Everywhere 5
- 1.3 THE JAVA PROGRAMMING LANGUAGE** 6
- 1.4 BECOMING FAMILIAR WITH YOUR PROGRAMMING ENVIRONMENT** 7
 - PT1** Backup Copies 10
- 1.5 ANALYZING YOUR FIRST PROGRAM** 11
 - SYN** Java Program 12
 - CE1** Omitting Semicolons 13
- 1.6 ERRORS** 14
 - CE2** Misspelling Words 15
- 1.7 PROBLEM SOLVING: ALGORITHM DESIGN** 15
 - HT1** Describing an Algorithm with Pseudocode 19
 - WE1** Writing an Algorithm for Tiling a Floor 21



© JanPietruszka/iStockphoto.



© JanPietruszka/iStockphoto.

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

Computers execute very basic instructions in rapid succession.

A computer program is a sequence of instructions and decisions.

Programming is the act of designing and implementing 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.

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), interacts 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 collectively called the **hardware**. The programs the computer executes are called the **software**.

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 **programming**. In this book, you will learn how to program a computer—that is, how to direct the computer to execute tasks.

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 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.



1. What is required to play music on a computer?
2. Why is a CD player less flexible than a computer?
3. What does a computer user need to know about programming in order to play a video game?

1.2 The Anatomy of a Computer

To understand the programming process, you need to have a rudimentary understanding 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.

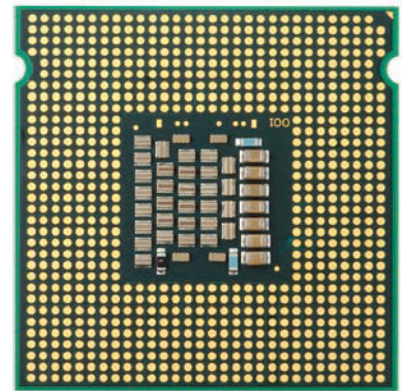
At the heart of the computer lies the **central processing unit (CPU)** (see Figure 1). The inside wiring of the CPU is enormously complicated. For example, the Intel Core processor (a popular CPU for personal computers at the time of this writing) is composed of several hundred million structural elements,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

The CPU performs program control and data processing. That is, the CPU locates and executes the program instructions; it carries out arithmetic operations such as addition, subtraction, multiplication, and division; it fetches data from external memory or devices and places processed data into storage.

There are two kinds of storage. Primary storage, 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

The central processing unit (CPU) performs program control and data processing.

Storage devices include memory and secondary storage.



© Amorphis/iStockphoto.

Figure 1 Central Processing Unit



PhotoDisc, Inc./Getty Images, Inc.

Figure 2 A Hard Disk

electricity. A hard disk consists of rotating platters, which are coated with a magnetic 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 computer 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 computer 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 secondary storage or elsewhere on the network. When a program is started, its instructions 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 analyzes the nature of these inputs and then executes the next appropriate instruction.

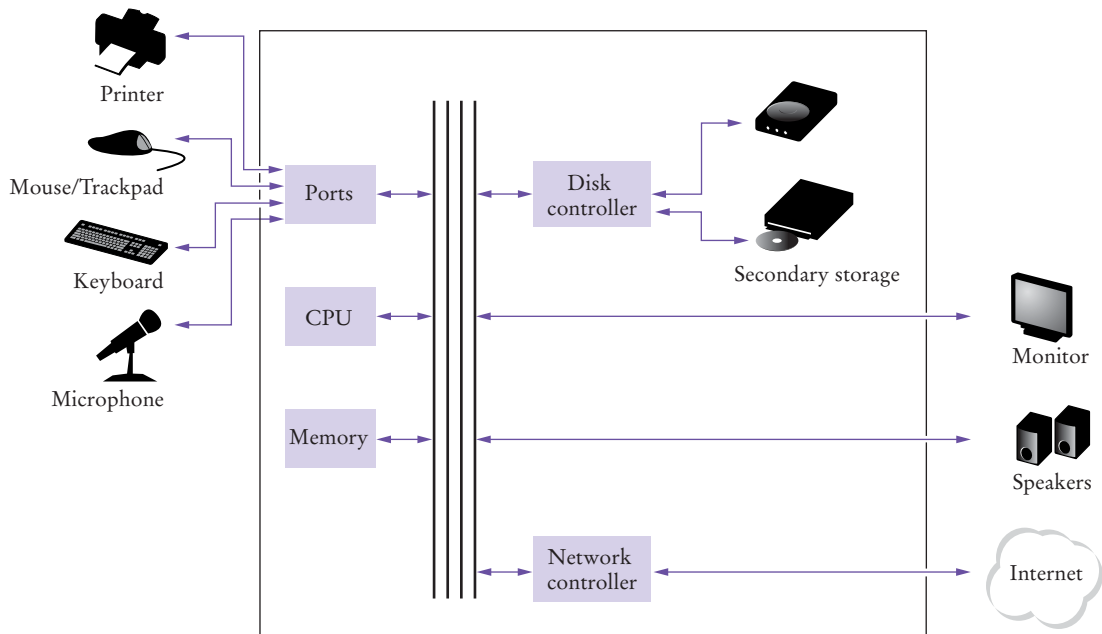


Figure 3 Schematic Design of a Personal Computer



4. Where is a program stored when it is not currently running?
5. Which part of the computer carries out arithmetic operations, such as addition and multiplication?
6. A modern smartphone is a computer, comparable to a desktop computer. Which components of a smartphone correspond to those shown in Figure 3?

Practice It Now you can try these exercises at the end of the chapter: R1.2, R1.3.



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 (electronic numerical integrator and computer), 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 has several computers—to control the engine, brakes, lights, and the radio.

The advent of ubiquitous computing changed many aspects of our lives. Factories used to employ people to do repetitive assembly tasks that are today carried out by computer-controlled robots, operated by a few people who know how to work with those computers. Books, music, and movies nowadays are often consumed on computers, and computers are almost always involved in their production. The book that you are reading right now could not have



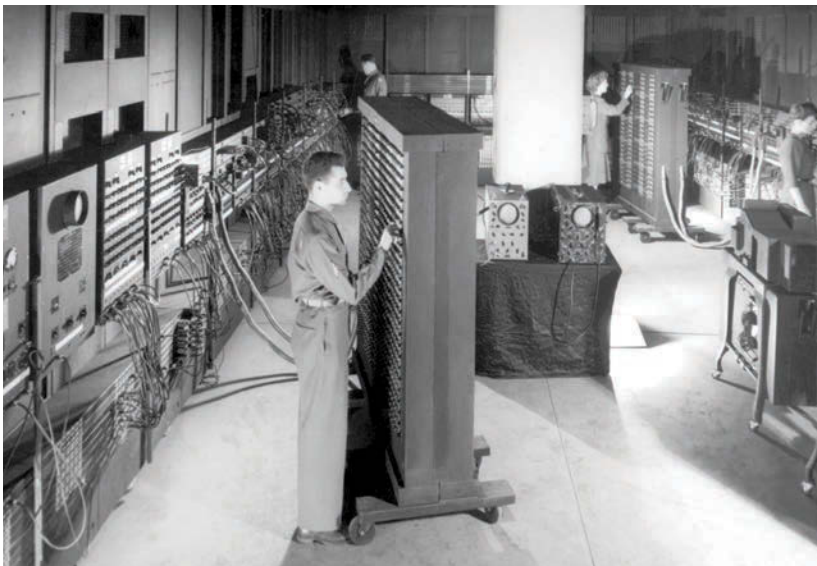
© Maurice Savage/Alamy Limited.

This transit card contains a computer.

been written without computers.

Knowing about computers and how to program them has become an essential skill in many careers. Engineers 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 networks to share videos showing abuse by repressive regimes, and this information 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 computing fundamentals that will make you a more informed citizen and, perhaps, a computing professional.



© UPPA/Photoshot.

The ENIAC

1.3 The Java Programming Language



James Sullivan/Getty Images

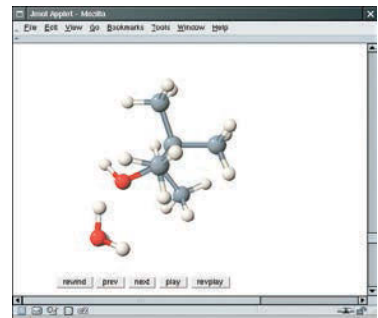
James Gosling

Java was originally designed for programming consumer devices, but it was first successfully used to write Internet applets.

In order to write a computer program, you need to provide a sequence of instructions that the CPU can execute. A computer program consists of a large number of simple CPU instructions, and it is tedious and error-prone to specify them one by one. For that reason, **high-level programming languages** have been created. In a high-level language, you specify the actions that your program should carry out. A **compiler** translates the high-level instructions into the more detailed instructions (called **machine code**) required by the CPU. Many different programming languages have been designed for different purposes.

In 1991, a group led by James Gosling and Patrick Naughton at Sun Microsystems designed a programming language, code-named “Green”, for use in consumer devices, such as intelligent television “set-top” boxes. The language was designed to be simple, secure, and usable for many different processor types. No customer was ever found for this technology.

Gosling recounts that in 1994 the team realized, “We could write a really cool browser. It was one of the few things in the client/server mainstream that needed some of the weird things we’d done: architecture neutral, real-time, reliable, secure.” Java was introduced to an enthusiastic crowd at the SunWorld exhibition in 1995, together with a browser that ran **applets**—Java code that can be located anywhere on the Internet. The figure at right shows a typical example of an applet.



An Applet for Visualizing Molecules

Since then, Java has grown at a phenomenal rate. Programmers have embraced the language because it is easier to use than its closest rival, C++. In addition, Java has a rich **library** that makes it possible to write portable programs that can bypass proprietary operating systems—a feature that was eagerly sought by those who wanted to be independent of those proprietary systems and was bitterly fought by their vendors. A “micro edition” and an “enterprise edition” of the Java library allow Java programmers to target hardware ranging from smart cards to the largest Internet servers.

Because Java was designed for the Internet, it has two attributes that make it very suitable for beginners: safety and portability.

Java was designed to be safe and portable, benefiting both Internet users and students.

Table 1 Java Versions (since Version 1.0 in 1996)

Version	Year	Important New Features	Version	Year	Important New Features
1.1	1997	Inner classes	5	2004	Generic classes, enhanced for loop, auto-boxing, enumerations, annotations
1.2	1998	Swing, Collections framework	6	2006	Library improvements
1.3	2000	Performance enhancements	7	2011	Small language changes and library improvements
1.4	2002	Assertions, XML support	8	2014	Function expressions, streams, new date/time library

Java was designed so that anyone can execute programs in their browser without fear. The safety features of the Java language ensure that a program is terminated if it tries to do something unsafe. Having a safe environment is also helpful for anyone learning Java. When you make an error that results in unsafe behavior, your program is terminated and you receive an accurate error report.

The other benefit of Java is portability. The same Java program will run, without change, on Windows, UNIX, Linux, or Macintosh. In order to achieve portability, the Java compiler does not translate Java programs directly into CPU instructions. Instead, compiled Java programs contain instructions for the Java **virtual machine**, a program that simulates a real CPU. Portability is another benefit for the beginning student. You do not have to learn how to write programs for different platforms.

At this time, Java is firmly established as one of the most important languages for general-purpose programming as well as for computer science instruction. However, although Java is a good language for beginners, it is not perfect, for three reasons.

Because Java was not specifically designed for students, no thought was given to making it really simple to write basic programs. A certain amount of technical machinery is necessary to write even the simplest programs. This is not a problem for professional programmers, but it can be a nuisance for beginning students. As you learn how to program in Java, there will be times when you will be asked to be satisfied with a preliminary explanation and wait for more complete detail in a later chapter.

Java has been extended many times during its life—see Table 1. In this book, we assume that you have Java version 7 or later.

Finally, you cannot hope to learn all of Java in one course. The Java language itself is relatively simple, but Java contains a vast set of *library packages* that are required to write useful programs. There are packages for graphics, user-interface design, cryptography, networking, sound, database storage, and many other purposes. Even expert Java programmers cannot hope to know the contents of all of the packages—they just use those that they need for particular projects.

Using this book, you should expect to learn a good deal about the Java language and about the most important packages. Keep in mind that the central goal of this book is not to make you memorize Java minutiae, but to teach you how to think about programming.

Java programs are distributed as instructions for a virtual machine, making them platform-independent.

Java has a very large library. Focus on learning those parts of the library that you need for your programming projects.



7. What are the two most important benefits of the Java language?
8. How long does it take to learn the entire Java library?

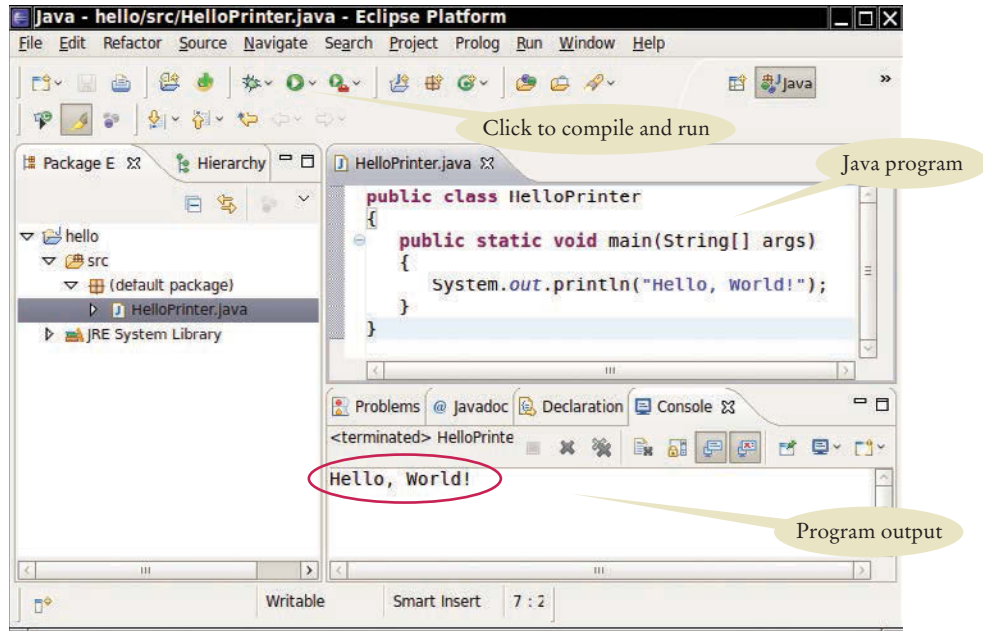
Practice It Now you can try this exercise at the end of the chapter: R1.5.

1.4 Becoming Familiar with Your Programming Environment

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

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.

Figure 4
Running the
HelloPrinter
Program in an
Integrated
Development
Environment



Step 1 Start the Java development environment.

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

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 tradition. 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.

An editor is a program for entering and modifying text, such as a Java program.

Figure 5
Running the HelloPrinter
Program in a Console Window

```

Terminal
File Edit View Terminal Tabs Help
~$ cd BigJava/ch01/hello
~/BigJava/ch01/hello$ javac HelloPrinter.java
~/BigJava/ch01/hello$ java HelloPrinter
Hello, World!
~/BigJava/ch01/hello$

```

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 on page 15.

Step 3 Run the program.

The process for running a program depends greatly on your programming environment. 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 Figures 4 and 5).

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

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 implementations 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 compiler and virtual machine explicitly.

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

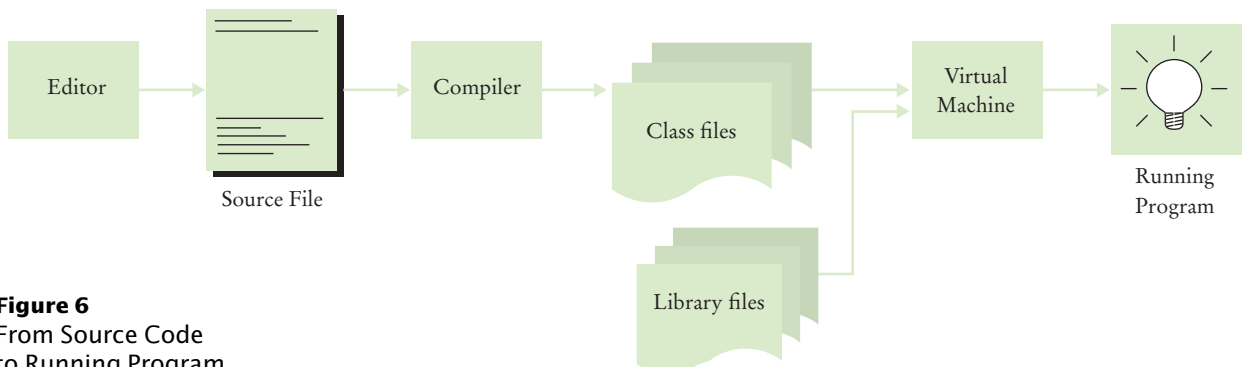


Figure 6
From Source Code
to Running Program