

**CAY HORSTMANN**

# **BIG JAVA**



# **Early Objects**

**Fifth Edition**

## 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
}
```

```
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



# Early Objects

Fifth Edition

# CAY HORSTMANN

San Jose State University

WILEY

PUBLISHER	Don Fowley
EXECUTIVE EDITOR	Beth Lang Golub
CONTENT MANAGER	Kevin Holm
EDITORIAL PROGRAM ASSISTANT	Katherine Willis
EXECUTIVE MARKETING MANAGER	Christopher Ruel
CREATIVE DIRECTOR	Harry Nolan
SENIOR DESIGNER	Madelyn Lesure
SENIOR PHOTO EDITOR	Lisa Gee
SENIOR CONTENT EDITOR	Wendy Ashenberg
SENIOR PRODUCT DESIGNER	Jenny Welter
EDITORIAL OPERATIONS MANAGER	Melissa Edwards
PRODUCTION EDITOR	Tim Lindner
PRODUCTION MANAGEMENT SERVICES	Cindy Johnson
COVER PHOTOS	(bird) © FLPA/John Holmes/Age Fotostock America, Inc.; (monkey) © S Sailer/A Sailer/Age Fotostock America, Inc.; (tiger) © Frans Lemmens/SuperStock
INTERIOR DESIGN	Maureen Eide

This book was set in Stempel Garamond by Publishing Services, and printed and bound by R.R. Donnelley & Sons Company.

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](http://www.wiley.com/go/citizenship).

Copyright © 2014 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 (Web site: [www.copyright.com](http://www.copyright.com)). Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, (201) 748-6011, fax (201) 748-6008, or online at: [www.wiley.com/go/permissions](http://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](http://www.wiley.com/go/returnlabel). If you have chosen to adopt this textbook for use in your course, please accept this book as your complimentary desk copy. Outside of the United States, please contact your local sales representative.

ISBN 978-1-118-43111-5 (Main Book)  
 ISBN 978-1-118-42297-7 (Binder-Ready Version)

Printed in the United States of America

10987654321



# 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. “Problem Solving” sections stress the importance of design and planning. “How To” guides help students with common programming tasks. Additional Worked Examples are available online.

## **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 lab 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.



*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.

## New to This Edition

### Problem Solving Strategies

This edition adds practical, step-by-step illustrations of techniques that can 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
- Adapting Algorithms
- Discovering Algorithms by Manipulating Physical Objects
- Patterns for Object Data
- Thinking Recursively
- Estimating the Running Time of an Algorithm

### Optional Science and Business Exercises

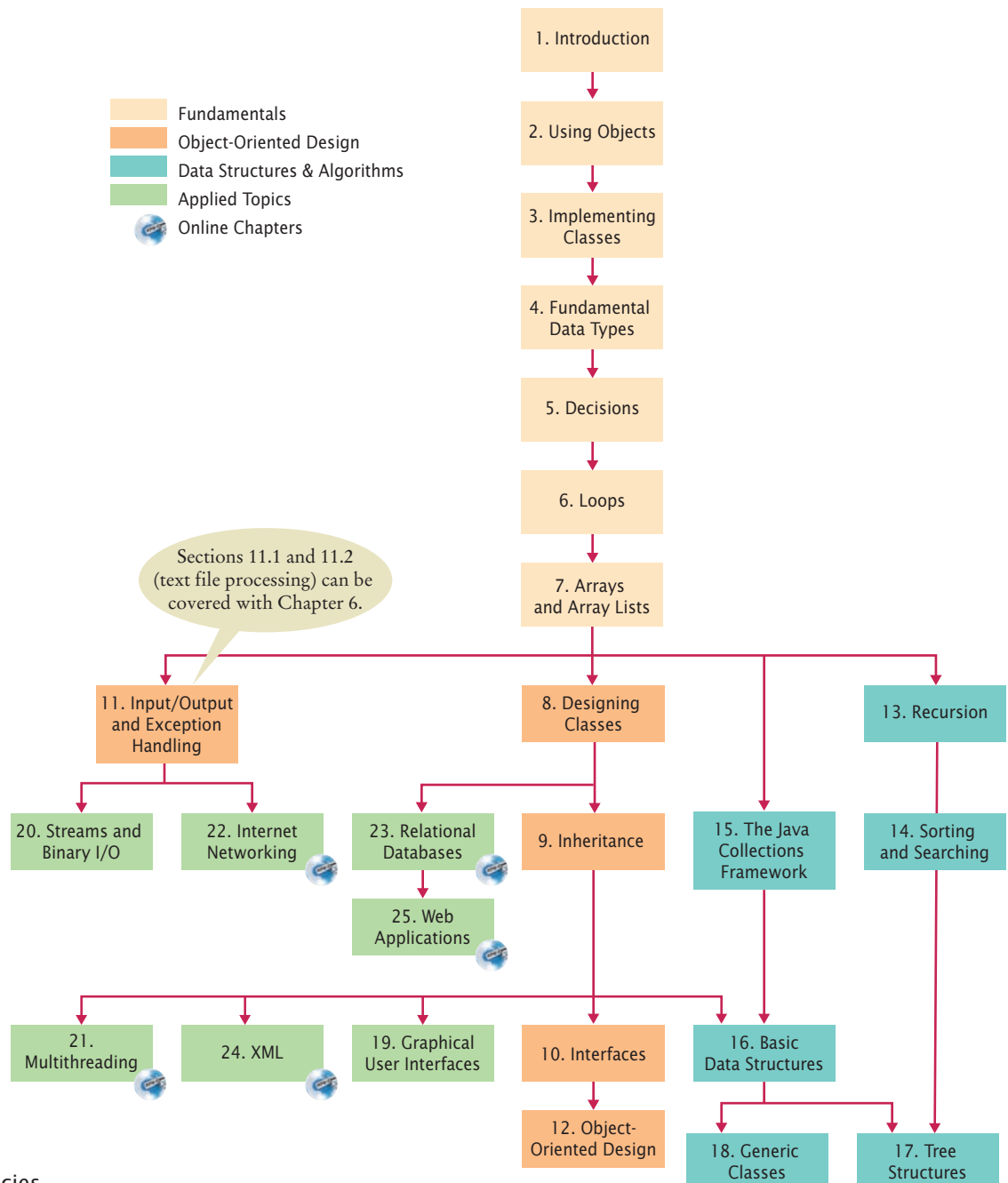
End-of-chapter exercises have been enhanced with problems from scientific and business domains. Designed to engage students, the exercises illustrate the value of programming in applied fields.

### New and Reorganized Topics

All chapters were revised and enhanced to respond to user feedback and improve the flow of topics. Loop algorithms are now introduced explicitly in Chapter 6. Additional array algorithms are presented in Chapter 7 and incorporated into the problem-solving sections. Chapter 8 is more clearly focused on the design of a single class, whereas Chapter 12 deals with relationships between classes. The coverage of data structures has been completely reorganized. Chapter 15 covers the use of existing data structures. The implementation of linked lists and stacks is now in Chapter 16, and a greatly enhanced Chapter 17 covers binary search trees, red-black trees, and trees whose nodes have more than two children. New example tables, photographs, and exercises appear throughout the book.

# A Tour of the Book

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



**Figure 1**  
Chapter  
Dependencies

## 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 inheritance. Chapter 12 contains an introduction to object-oriented design, including two significant case studies.

## Part C: Data Structures and Algorithms (Chapters 13–18)

Chapters 13 through 18 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.

## Part D: Applied Topics (Chapters 19–25)



Chapters 19 through 25 cover Java programming techniques that definitely go beyond a first course in Java (21–25 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 I conflicts with instructor sentiment or local customs, however, it is available in electronic form so that it can be modified.

- |   |                                    |
|---|------------------------------------|
| 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 Language Coding Guidelines |
| E. Java Syntax Summary                            | J. HTML Summary                    |

## Custom Book and eBook Options

*Big Java* may be ordered as a custom print or eBook that includes your choice of chapters—including those from other Horstmann titles. Visit [customselect.wiley.com](http://customselect.wiley.com) to create your custom book order.

To order the Wiley Select Edition of *Big Java* with all 25 chapters in the printed book, specify ISBN 978-1-119-93670-1 when you order books.

*Big Java* is available in a variety of eBook formats at prices that are significantly lower than the printed book. Please contact your Wiley sales rep for more information or check [www.wiley.com/college/horstmann](http://www.wiley.com/college/horstmann) for available versions.

## Web Resources

This book is complemented by a complete suite of online resources. Go to [www.wiley.com/college/horstmann](http://www.wiley.com/college/horstmann) to visit the online companion sites, which include

- “CodeCheck,” a new online service currently in development by Cay Horstmann that students can use to check their homework assignments and to work on additional practice problems. Visit <http://horstmann.com/codecheck> to learn more and to try it out.
- Source code for all example programs in the book and in online examples.
- Worked Examples that apply the problem-solving steps in the book to other realistic examples.
- Animations of key concepts.
- Lab exercises that apply chapter concepts (with solutions for instructors only).
- Lecture presentation slides (for instructors only).
- Solutions to all review and programming exercises (for instructors only).
- A test bank that focuses on skills, not just terminology (for instructors only). This extensive set of multiple-choice questions can be used with a word processor or imported into a course management system.

Pointers in the book describe what students will find on the Web.

The diagram shows a text box on the left with two red arrows pointing to two example boxes on the right. The top box is titled 'WORKED EXAMPLE 6.3 A Sample Debugging Session' and contains text about finding bugs in an algorithm. The bottom box is titled 'FULL CODE EXAMPLE' and contains text about downloading code to demonstrate variables and assignments. The background of the right side features a stylized graphic of the word 'communication'.

**WORKED EXAMPLE 6.3 A Sample Debugging Session**  
Learn how to find bugs in an algorithm for counting the syllables of a word. Go to [wiley.com/go/javaexamples](http://wiley.com/go/javaexamples) and download Worked Example 6.3.

**FULL CODE EXAMPLE**  
Go to [wiley.com/go/javacode](http://wiley.com/go/javacode) to download a program that demonstrates variables and assignments.



# A 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.

254 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).

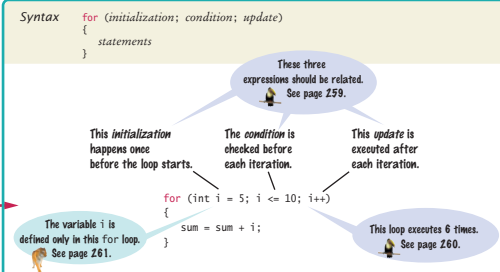
```
for (int counter = 1; counter <= 10; counter++)
{
    System.out.println(counter);
}
```

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



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.

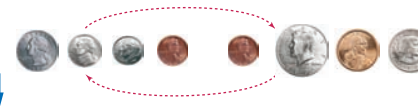
**Problem Solving sections** teach techniques for generating ideas and evaluating proposed solutions, often using pencil and paper or other artifacts. These sections emphasize that most of the planning and problem solving that makes students successful happens away from the computer.

7.5 Problem Solving: Discovering Algorithms by Manipulating Physical Objects 339

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](http://worldclimate.com), the average maximum temperatures for Death Valley are (in order by month, in degrees Celsius):

18.2 22.6 26.4 31.1 36.6 42.2 45.7 44.5 40.2 33.1 24.2 17.6  
In this case, the month with the highest 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/javaexamples](http://wiley.com/go/javaexamples) and download Worked Example 6.1.



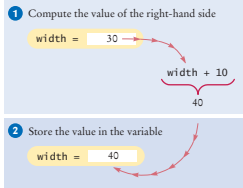
**Worked Examples** apply the steps in the How To to a different example, showing how they can be used to plan, implement, and test a solution to another programming problem.

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, <code>width</code> 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>	<b>Error:</b> 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>	<b>Error:</b> 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 42.
<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.

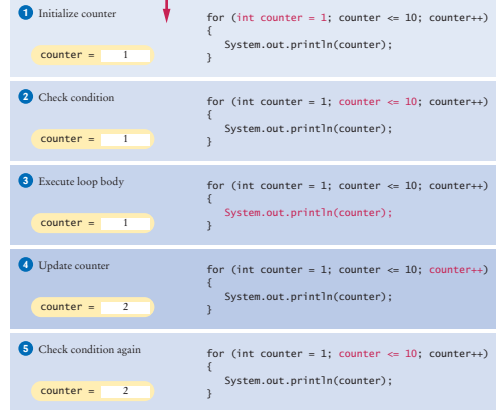
This means “compute the value of  $width + 10$  ① and store that value in the variable  $width$  ②” (see Figure 4).  
 In Java, it is not a problem that the variable  $width$  is used on both sides of the = symbol. Of course, in mathematics, the equation  $width = width + 10$  has no solution.



**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

Students can view **animations** of key concepts on the Web.



**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	1.099712	1.08889
CHINA	CNY	0.13769	0.6910
EURO	EUR	0.66849	0.6100
JAPAN	JPY	109.800	10.800
SINGAPORE	SGD	1.3712	1.2680

• **Science P6.15** Radioactive decay of radioactive materials can be modeled by the equation  $A = A_0 e^{-t(\log 2/b)}$ , 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` instance variable is set to 80,000 and `status` is set to `MARRIED`. Then the `getTax` method is called. In lines 31 and 32 of `TaxReturn.java`, `tax1` and `tax2` are initialized to 0.


```

29 public double getTax()
30 {
31     double tax1 = 0;
32     double tax2 = 0;
33 }
    
```

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

```

34     if (status == SINGLE)
35     {
36         if (income <= RATE1_SINGLE_LIMIT)
37         {
38             tax1 = RATE1 * income;
39         }
40     }
41     else
42     {
43         tax1 = RATE1 * RATE1_SINGLE_LIMIT;
44         tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
45     }
    
```




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

income	status	tax1	tax2
80000	MARRIED	0	0

**Special Topics** present optional topics and provide additional explanation of others. New features of Java 7 are also covered in these notes.

**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.

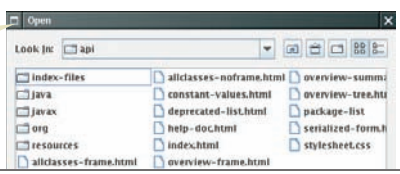
For better placement of the dialog box on the screen, you can specify the user-interface component over which to pop up the dialog box. If you don't care where the dialog box pops up, you can simply pass `null`. The `showOpenDialog` and `showSaveDialog` methods return either `JFileChooser.APPROVE_OPTION`, 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();
Scanner in = null;
if (chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
{
    File selectedFile = chooser.getSelectedFile();
    in = new Scanner(selectedFile);
}
    
```

**FULL CODE EXAMPLE**


Go to [wiley.com/go/javacode](http://wiley.com/go/javacode) to download a program that demonstrates how to use a file chooser.



Call with `showOpenDialog` method


**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.

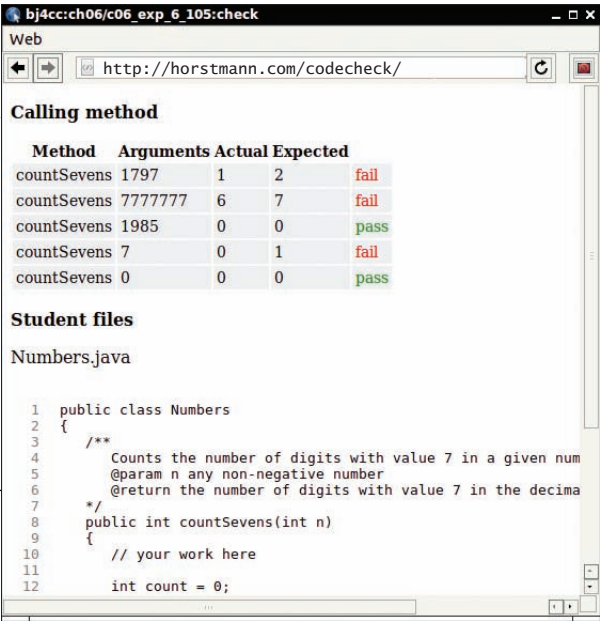
The advent of ubiquitous computing changed many aspects of our lives. Factories used to employ people to do repetitive assembly tasks that are today carried out by computer-controlled robots, operated by a few people who know how to work with those computers. Books, music, and movies are nowadays often 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.*

Web Resources

**CodeCheck** “CodeCheck” is a new online service currently in development by Cay Horstmann that students can use to check their homework and to work on additional practice problems. Visit <http://horstmann.com/codecheck> to learn more and to try it out.



**Test Bank** Instructors can use quiz and test questions designed to exercise students’ code reading and writing skills.

10) What is displayed after executing the given code snippet?

```

int[] mymarks = new int[10];
int total = 0;
Scanner in = new Scanner(System.in);
for (int cnt = 1; cnt <= 10; cnt++)
{
    System.out.print("Enter the marks: ");
    mymarks[cnt] = in.nextInt();
    total = total + mymarks[cnt];
}
System.out.println(total);
    
```

a) The code snippet displays the total marks of all ten subjects.  
 b) The for loop causes a run-time time error on the first iteration.  
 c) The code snippet causes a bounds error.  
 d) The code snippet displays zero.

1.1) Consider the following Card class.

```

public class Card
{
    private String name;

    public Card()
    {
        name = "";
    }

    public Card(String n)
    {
        name = n;
    }

    public String getName()
    {
        return name;
    }

    public boolean isExpired()
    {
        return false;
    }

    public String format()
    {
        return "Card holder: " + name;
    }
}
    
```

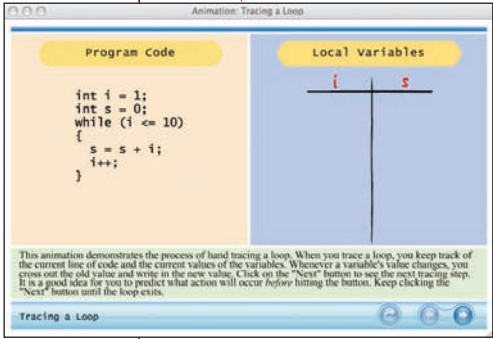
Use this class as a superclass to implement a hierarchy of related classes:

<u>Class</u>	<u>Data</u>
IDCard	ID number
CallingCard	Card number, PIN
DriverLicense	Expiration year

Write declarations for each of the subclasses. For each subclass, supply private instance variables. Leave the bodies of the constructors and the format methods blank for now.

**Lab Exercises** These multi-part exercises ask students to apply chapter concepts. They can serve as “warm-ups” in the lab or to provide additional practice.

**Animations** Students can play and replay dynamic explanations of concepts and program flow.





# Acknowledgments

Many thanks to Beth Lang Golub, Don Fowley, Elizabeth Mills, Katherine Willis, Jenny Welter, Wendy Ashenberg, Lisa Gee, Kevin Holm, 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 Suchindran Chatterjee, *Arizona State University*, Jose Cordova, *University of Louisiana*, Udayan Das, *DeVry University*, James Johnson, Aaron Keen, *California Polytechnic State University San Luis Obispo*, Norm Krumpe, *Miami University Ohio*, Kathy Liszka, *University of Akron*, Kathleen O'Brien, *San Jose State University*, Donald Smith, *Columbia College*, Mark Thomas, *University of Cincinnati*, Laurie White, *Mercer University*, Brent Wilson, *George Fox University*, and David Woolbright, *Columbus State 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:

Eric Aaron, <i>Wesleyan University</i>	Guy Helmer, <i>Iowa State University</i>	Bill Mongan, <i>Drexel University</i>
James Agnew, <i>Anne Arundel Community College</i>	Ed Holden, <i>Rochester Institute of Technology</i>	George Novacky, <i>University of Pittsburgh</i>
Greg Ballinger, <i>Miami Dade College</i>	Steven Janke, <i>Colorado College</i>	Mimi Opkins, <i>California State University Long Beach</i>
Jon Beck, <i>Truman State University</i>	Mark Jones, <i>Lock Haven University of Pennsylvania</i>	Derek Pao, <i>City University of Hong Kong</i>
Matt Boutell, <i>Rose-Hulman Institute of Technology</i>	Dr. Mustafa Kamal, <i>University of Central Missouri</i>	Katherine Salch, <i>Illinois Central College</i>
John Bundy, <i>DeVry University Chicago</i>	Gary J. Koehler, <i>University of Florida</i>	Javad Shakib, <i>DeVry University</i>
Michael Carney, <i>Finger Lakes Community College</i>	Ronald Krawitz, <i>DeVry University</i>	Charlie Shu, <i>Franklin University</i>
Christopher Cassa, <i>Massachusetts Institute of Technology</i>	Norm Krumpe, <i>Miami University Ohio</i>	Joslyn A. Smith, <i>Florida International University</i>
Dr. Suchindran S. Chatterjee, <i>Arizona State University</i>	Jim Leone, <i>Rochester Institute of Technology</i>	Robert Strader, <i>Stephen F. Austin State University</i>
Tina Comston, <i>Franklin University</i>	Kevin Lillis, <i>St. Ambrose University</i>	Jonathan S. Weissman, <i>Finger Lakes Community College</i>
Lennie Cooper, <i>Miami Dade College</i>	Darren Lim, <i>Siena College</i>	Katherine H. Winters, <i>University of Tennessee Chattanooga</i>
Sherif Elfayoumy, <i>University of North Florida</i>	Hong Lin, <i>DeVry University</i>	Tom Wulf, <i>University of Cincinnati</i>
Henry A Etlinger, <i>Rochester Institute of Technology</i>	Kuber Maharjan, <i>Purdue University College of Technology at Columbus</i>	Qi Yu, <i>Rochester Institute of Technology</i>
	Patricia McDermott-Wells, <i>Florida International University</i>	

## xiv Acknowledgments

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:

- Tim Andersen, *Boise State University*  
Ivan Bajic, *San Diego State University*  
Ted Bangay, *Sheridan Institute of Technology*  
Ian Barland, *Radford University*  
George Basham, *Franklin University*  
Sambit Bhattacharya, *Fayetteville State University*  
Rick Birney, *Arizona State University*  
Paul Bladek, *Edmonds Community College*  
Joseph Bowbeer, *Vizrea Corporation*  
Timothy A. Budd, *Oregon State University*  
Robert P. Burton, *Brigham Young University*  
Frank Butt, *IBM*  
Jerry Cain, *Stanford University*  
Adam Cannon, *Columbia University*  
Nancy Chase, *Gonzaga University*  
Archana Chidanandan, *Rose-Hulman Institute of Technology*  
Vincent Cicirello, *The Richard Stockton College of New Jersey*  
Teresa Cole, *Boise State University*  
Deborah Coleman, *Rochester Institute of Technology*  
Jose Cordova, *University of Louisiana, Monroe*  
Valentino Crespi, *California State University, Los Angeles*  
Jim Cross, *Auburn University*  
Russell Deaton, *University of Arkansas*  
Geoffrey Decker, *Northern Illinois University*  
H. E. Dunsmore, *Purdue University*  
Robert Duvall, *Duke University*  
Eman El-Sheikh, *University of West Florida*  
John Fendrich, *Bradley University*  
David Freer, *Miami Dade College*  
John Fulton, *Franklin University*  
David Geary, *Sabreware, Inc.*  
Margaret Geroch, *Wheeling Jesuit University*  
Ahmad Ghafarian, *North Georgia College & State University*  
Rick Giles, *Acadia University*  
Stacey Grasso, *College of San Mateo*  
Jianchao Han, *California State University, Dominguez Hills*  
Lisa Hansen, *Western New England College*  
Elliotte Harold  
Eileen Head, *Binghamton University*  
Cecily Heiner, *University of Utah*  
Brian Howard, *Depauw University*  
Lubomir Ivanov, *Iona College*  
Norman Jacobson, *University of California, Irvine*  
Curt Jones, *Bloomsburg University*  
Aaron Keen, *California Polytechnic State University, San Luis Obispo*  
Mugdha Khaladkar, *New Jersey Institute of Technology*  
Elliot Koffman, *Temple University*  
Kathy Liszka, *University of Akron*  
Hunter Lloyd, *Montana State University*  
Youmin Lu, *Bloomsburg University*  
John S. Mallozzi, *Iona College*  
John Martin, *North Dakota State University*  
Jeanna Matthews, *Clarkson 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*  
John Moore, *The Citadel*  
Jose-Arturo Mora-Soto, Jessica Rivero-Espinosa, and Julio-Angel Cano-Romero, *University of Madrid*  
Faye Navabi, *Arizona State University*  
Parviz Partow-Navid, *California State University, Los Angeles*  
Kevin O’Gorman, *California Polytechnic State University, San Luis Obispo*  
Michael Olan, *Richard Stockton College*  
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*  
John Santore, *Bridgewater State College*  
Carolyn Schauble, *Colorado State University*  
Brent Seales, *University of Kentucky*  
Christian Shin, *SUNY Geneseo*  
Jeffrey Six, *University of Delaware*  
Don Slater, *Carnegie Mellon University*  
Ken Slonneger, *University of Iowa*  
Donald Smith, *Columbia College*  
Stephanie Smullen, *University of Tennessee, Chattanooga*  
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*  
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*  
Catherine Wyman, *DeVry University*  
Arthur Yanushka, *Christian Brothers University*  
Salih Yurttas, *Texas A&M University*

# CONTENTS

PREFACE	iii
SPECIAL FEATURES	xxii

## CHAPTER 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	8
1.5	Analyzing Your First Program	12
1.6	Errors	15
1.7	Problem Solving: Algorithm Design	16

## CHAPTER 2 USING OBJECTS 33

2.1	Objects and Classes	34
2.2	Variables	36
2.3	Calling Methods	43
2.4	Constructing Objects	48
2.5	Accessor and Mutator Methods	50
2.6	The API Documentation	52
2.7	Implementing a Test Program	55
2.8	Object References	57
2.9	Graphical Applications	61
2.10	Ellipses, Lines, Text, and Color	66

## CHAPTER 3 IMPLEMENTING CLASSES 81

3.1	Instance Variables and Encapsulation	82
3.2	Specifying the Public Interface of a Class	86
3.3	Providing the Class Implementation	93
3.4	Unit Testing	102
3.5	Problem Solving: Tracing Objects	105
3.6	Local Variables	107
3.7	The this Reference	109
3.8	Shape Classes	112

**CHAPTER 4** FUNDAMENTAL DATA TYPES **131**

- 4.1 Numbers **132**
- 4.2 Arithmetic **139**
- 4.3 Input and Output **147**
- 4.4 Problem Solving: First Do it By Hand **154**
- 4.5 Strings **156**

**CHAPTER 5** DECISIONS **179**

- 5.1 The if Statement **180**
- 5.2 Comparing Values **186**
- 5.3 Multiple Alternatives **196**
- 5.4 Nested Branches **200**
- 5.5 Problem Solving: Flowcharts **207**
- 5.6 Problem Solving: Selecting Test Cases **210**
- 5.7 Boolean Variables and Operators **213**
- 5.8 Application: Input Validation **218**

**CHAPTER 6** LOOPS **241**

- 6.1 The while Loop **242**
- 6.2 Problem Solving: Hand-Tracing **249**
- 6.3 The for Loop **254**
- 6.4 The do Loop **262**
- 6.5 Application: Processing Sentinel Values **263**
- 6.6 Problem Solving: Storyboards **269**
- 6.7 Common Loop Algorithms **272**
- 6.8 Nested Loops **279**
- 6.9 Application: Random Numbers and Simulations **283**
- 6.10 Using a Debugger **286**

**CHAPTER 7** ARRAYS AND ARRAY LISTS **311**

- 7.1 Arrays **312**
- 7.2 The Enhanced for Loop **321**
- 7.3 Common Array Algorithms **322**
- 7.4 Problem Solving: Adapting Algorithms **331**
- 7.5 Problem Solving: Discovering Algorithms by Manipulating Physical Objects **336**
- 7.6 Two-Dimensional Arrays **340**

- 7.7 Array Lists 347
- 7.8 Regression Testing 356

## CHAPTER 8 DESIGNING CLASSES 379

- 8.1 Discovering Classes 380
- 8.2 Designing Good Methods 381
- 8.3 Problem Solving: Patterns for Object Data 390
- 8.4 Static Variables and Methods 395
- 8.5 Packages 400
- 8.6 Unit Test Frameworks 407

## CHAPTER 9 INHERITANCE 421

- 9.1 Inheritance Hierarchies 422
- 9.2 Implementing Subclasses 426
- 9.3 Overriding Methods 431
- 9.4 Polymorphism 437
- 9.5 Object: The Cosmic Superclass 448

## CHAPTER 10 INTERFACES 463

- 10.1 Using Interfaces for Algorithm Reuse 464
- 10.2 Working with Interface Variables 471
- 10.3 The Comparable Interface 473
- 10.4 Using Interfaces for Callbacks 477
- 10.5 Inner Classes 481
- 10.6 Mock Objects 483
- 10.7 Event Handling 484
- 10.8 Building Applications with Buttons 490
- 10.9 Processing Timer Events 494
- 10.10 Mouse Events 497

## CHAPTER 11 INPUT/OUTPUT AND EXCEPTION HANDLING 513

- 11.1 Reading and Writing Text Files 514
- 11.2 Text Input and Output 519
- 11.3 Command Line Arguments 527
- 11.4 Exception Handling 534
- 11.5 Application: Handling Input Errors 545



**CHAPTER 12** OBJECT-ORIENTED DESIGN **559**

- 12.1 Classes and Their Responsibilities **560**
- 12.2 Relationships Between Classes **563**
- 12.3 Application: Printing an Invoice **569**

**CHAPTER 13** RECURSION **587**

- 13.1 Triangle Numbers **588**
- 13.2 Recursive Helper Methods **596**
- 13.3 The Efficiency of Recursion **598**
- 13.4 Permutations **603**
- 13.5 Mutual Recursion **608**
- 13.6 Backtracking **614**

**CHAPTER 14** SORTING AND SEARCHING **629**

- 14.1 Selection Sort **630**
- 14.2 Profiling the Selection Sort Algorithm **633**
- 14.3 Analyzing the Performance of the Selection Sort Algorithm **636**
- 14.4 Merge Sort **641**
- 14.5 Analyzing the Merge Sort Algorithm **644**
- 14.6 Searching **648**
- 14.7 Problem Solving: Estimating the Running Time of an Algorithm **653**
- 14.8 Sorting and Searching in the Java Library **658**

**CHAPTER 15** THE JAVA COLLECTIONS FRAMEWORK **671**

- 15.1 An Overview of the Collections Framework **672**
- 15.2 Linked Lists **675**
- 15.3 Sets **681**
- 15.4 Maps **686**
- 15.5 Stacks, Queues, and Priority Queues **692**
- 15.6 Stack and Queue Applications **695**

**CHAPTER 16** BASIC DATA STRUCTURES **715**

- 16.1 Implementing Linked Lists **716**
- 16.2 Implementing Array Lists **731**
- 16.3 Implementing Stacks and Queues **735**
- 16.4 Implementing a Hash Table **741**

**CHAPTER 17** TREE STRUCTURES **761**

- 17.1 Basic Tree Concepts **762**
- 17.2 Binary Trees **766**
- 17.3 Binary Search Trees **771**
- 17.4 Tree Traversal **780**
- 17.5 Red-Black Trees **786**
- 17.6 Heaps **793**
- 17.7 The Heapsort Algorithm **804**

**CHAPTER 18** GENERIC CLASSES **819**

- 18.1 Generic Classes and Type Parameters **820**
- 18.2 Implementing Generic Types **821**
- 18.3 Generic Methods **825**
- 18.4 Constraining Type Parameters **827**
- 18.5 Type Erasure **831**

**CHAPTER 19** GRAPHICAL USER INTERFACES **841**

- 19.1 Layout Management **842**
- 19.2 Processing Text Input **846**
- 19.3 Choices **852**
- 19.4 Menus **863**
- 19.5 Exploring the Swing Documentation **869**

**CHAPTER 20** STREAMS AND BINARY INPUT/OUTPUT **881**

- 20.1 Readers, Writers, and Streams **882**
- 20.2 Binary Input and Output **883**
- 20.3 Random Access **887**
- 20.4 Object Streams **893**

**CHAPTER 21** MULTITHREADING (WEB ONLY)

- 21.1 Running Threads
- 21.2 Terminating Threads
- 21.3 Race Conditions
- 21.4 Synchronizing Object Access
- 21.5 Avoiding Deadlocks
- 21.6 Application: Algorithm Animation

**CHAPTER 22** INTERNET NETWORKING (WEB ONLY) 

- 22.1 The Internet Protocol
- 22.2 Application Level Protocols
- 22.3 A Client Program
- 22.4 A Server Program
- 22.5 URL Connections

**CHAPTER 23** RELATIONAL DATABASES (WEB ONLY) 

- 23.1 Organizing Database Information
- 23.2 Queries
- 23.3 Installing a Database
- 23.4 Database Programming in Java
- 23.5 Application: Entering an Invoice

**CHAPTER 24** XML (WEB ONLY) 

- 24.1 XML Tags and Documents
- 24.2 Parsing XML Documents
- 24.3 Creating XML Documents
- 24.4 Validating XML Documents

**CHAPTER 25** WEB APPLICATIONS (WEB ONLY) 

- 25.1 The Architecture of a Web Application
- 25.2 The Architecture of a JSF Application
- 25.3 JavaBeans Components
- 25.4 Navigation Between Pages
- 25.5 JSF Components
- 25.6 A Three-Tier Application

**APPENDICES**

- 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 SYNTAX SUMMARY **A-53**
- APPENDIX F** TOOL SUMMARY **A-64**
- APPENDIX G** NUMBER SYSTEMS **A-68**
- APPENDIX H** UML SUMMARY **A-76**

**APPENDIX I** JAVA LANGUAGE CODING GUIDELINES **A-79**

**APPENDIX J** HTML SUMMARY **A-86**









GLOSSARY **G-1**



INDEX **I-1**

CREDITS **C-1**

## **ALPHABETICAL LIST OF** SYNTAX BOXES

Arrays 313  
 Array Lists 347  
 Assignment 41  
 Calling a Superclass Method 431  
 Cast 143  
 Catching Exceptions 536  
 Class Declaration 89  
 Comparisons 187  
 Constant Declaration 136  
 Constructor with Superclass Initializer 436  
 Declaring a Generic Class 822  
 Declaring a Generic Method 826  
 Declaring an Interface 465  
 for Statement 254  
 if Statement 182  
 Implementing an Interface 467  
 Importing a Class from a Package 54  
 Input Statement 147  
 Instance Variable Declaration 83  
 Java Program 13  
 Object Construction 49  
 Package Specification 402  
 Subclass Declaration 428  
 The Enhanced for Loop 322  
 The finally Clause 540  
 The instanceof Operator 451  
 The throws Clause 539  
 Throwing an Exception 534  
 Two-Dimensional Array Declaration 341  
 while Statement 243  
 Variable Declaration 37

CHAPTER	 Common Errors	 How Tos and Worked Examples
<b>1</b> Introduction	Omitting Semicolons 14 Misspelling Words 16	Describing an Algorithm with Pseudocode 20 Writing an Algorithm for Tiling a Floor 22
<b>2</b> Using Objects	Using Undeclared or Uninitialized Variables 42 Confusing Variable Declarations and Assignment Statements 42 Trying to Invoke a Constructor Like a Method 50	How Many Days Have You Been Alive?  Working with Pictures 
<b>3</b> Implementing Classes	Declaring a Constructor as void 92 Ignoring Parameter Variables 98 Duplicating Instance Variables in Local Variables 108 Providing Unnecessary Instance Variables 108 Forgetting to Initialize Object References in a Constructor 109	Implementing a Class 98 Making a Simple Menu  Drawing Graphical Shapes 116
<b>4</b> Fundamental Data Types	Unintended Integer Division 144 Unbalanced Parentheses 144	Carrying out Computations 151 Computing the Volume and Surface Area of a Pyramid  Computing Travel Time 
<b>5</b> Decisions	A Semicolon After the if Condition 184 Using == to Compare Strings 192 The Dangling else Problem 204 Combining Multiple Relational Operators 216 Confusing && and    Conditions 216	Implementing an if Statement 193 Extracting the Middle 

 <b>Programming Tips</b>		 <b>Special Topics</b>		 <b>Computing &amp; Society</b>	
Backup Copies	11			Computers Are Everywhere	5
Choose Descriptive Variable Names	43	Testing Classes in an Interactive Environment	56	Computer Monopoly	60
Learn By Trying	47				
Don't Memorize—Use Online Help	55				
The javadoc Utility	92	Calling One Constructor from Another	112	Electronic Voting Machines	104
Do Not Use Magic Numbers	139	Big Numbers	138	The Pentium Floating-Point Bug	146
Spaces in Expressions	145	Combining Assignment and Arithmetic	145	International Alphabets and Unicode	163
Reading Exception Reports	162	Instance Methods and Static Methods	145		
		Using Dialog Boxes for Input and Output	162		
Brace Layout	184	The Conditional Operator	185	Denver's Luggage Handling System	195
Always Use Braces	184	The switch Statement	199	Artificial Intelligence	221
Tabs	185	Block Scope	205		
Avoid Duplication in Branches	186	Enumeration Types	206		
Hand-Tracing	203	Logging	212		
Make a Schedule and Make Time for Unexpected Problems	212	Short-Circuit Evaluation of Boolean Operators	217		
		De Morgan's Law	217		



## CHAPTER



## Common Errors

How Tos  
and  
Worked Examples**6** Loops

Don't Think "Are We There Yet?"	247
Infinite Loops	248
Off-by-One Errors	248

Writing a Loop	276
Credit Card Processing	
Manipulating the Pixels in an Image	
Debugging	289
A Sample Debugging Session	

**7** Arrays and Array Lists

Bounds Errors	318
Uninitialized and Unfilled Arrays	318
Underestimating the Size of a Data Set	331
Length and Size	356

Working with Arrays	334
Rolling the Dice	
A World Population Table	

**8** Designing Classes




Trying to Access Instance Variables in Static Methods	398
Confusing Dots	403

Programming with Packages	404
---------------------------	-----

**9** Inheritance

Replicating Instance Variables from the Superclass	430
Confusing Super- and Subclasses	430
Accidental Overloading	435
Forgetting to Use super When Invoking a Superclass Method	435
Don't Use Type Tests	452

Developing an Inheritance Hierarchy	443
Implementing an Employee Hierarchy for Payroll Processing	

 <b>Programming Tips</b>	 <b>Special Topics</b>	 <b>Computing &amp; Society</b>
Use for Loops for Their Intended Purpose Only 259 Choose Loop Bounds That Match Your Task 260 Count Iterations 260 Flowcharts for Loops 263	Variables Declared in a for Loop Header 261 Redirection of Input and Output 266 The Loop-and-a-Half Problem 266 The break and continue Statements 267	Software Piracy 253 The First Bug 291
Use Arrays for Sequences of Related Items 318 Make Parallel Arrays into Arrays of Objects 318 Batch Files and Shell Scripts 358	Methods with a Variable Number of Arguments 319 Sorting with the Java Library 331 Two-Dimensional Arrays with Variable Row Lengths 345 Multidimensional Arrays 347 The Diamond Syntax in Java 7 356	Computer Viruses 320 The Therac-25 Incidents 359
Consistency 385 Minimize the Use of Static Methods 397	Call by Value and Call by Reference 386 Static Imports 398 Alternative Forms of Instance and Static Variable Initialization 399 Package Access 404	Personal Computing 406
Use a Single Class for Variation in Values, Inheritance for Variation in Behavior 426	Calling the Superclass Constructor 436 Dynamic Method Lookup and the Implicit Parameter 440 Abstract Classes 441 Final Methods and Classes 442 Protected Access 442 Inheritance and the toString Method 453 Inheritance and the equals Method 454	Who Controls the Internet? 454

## CHAPTER



## Common Errors

How Tos  
and  
Worked Examples**10** Interfaces

Forgetting to Declare Implementing Methods as Public 470

Trying to Instantiate an Interface 470

Modifying Parameter Types in the Implementing Method 489

Trying to Call Listener Methods 490

Forgetting to Attach a Listener 493

Forgetting to Repaint 496

Investigating Number Sequences

**11** Input/Output and Exception Handling

Backslashes in File Names 517

Constructing a Scanner with a String 517

Processing Text Files 530

Analyzing Baby Names

**12** Object-Oriented Design

Using CRC Cards and UML Diagrams in Program Design 566

Simulating an Automatic Teller Machine

**13** Recursion

Infinite Recursion 592




Tracing Through Recursive Methods 592

Thinking Recursively 593

Finding Files

Towers of Hanoi



 <b>Programming Tips</b>	 <b>Special Topics</b>	 <b>Computing &amp; Society</b>
<p>Don't Use a Container as a Listener 493</p>	<p>Constants in Interfaces 470                      The clone Method and the Cloneable Interface 475                      Anonymous Classes 482                      Keyboard Events 500                      Event Adapters 501</p>	<p>Open Source and Free Software 502</p>
<p>Throw Early, Catch Late 542                      Do Not Squelch Exceptions 542                      Do Not Use catch and finally in the Same try Statement 542                      Do Throw Specific Exceptions 543</p>	<p>Reading Web Pages 517                      File Dialog Boxes 517                      Character Encodings 518                      Regular Expressions 526                      Assertions 543                      Automatic Resource Management in Java 7 544</p>	<p>Encryption Algorithms 533                      The Ariane Rocket Incident 544</p>
	<p>Attributes and Methods in UML Diagrams 567                      Multiplicities 568                      Aggregation, Association, and Composition 568</p>	<p>Databases and Privacy 580</p>
		<p>The Limits of Computation 606</p>




## CHAPTER



## Common Errors

How Tos  
and  
Worked Examples

<b>14</b> Sorting and Searching	The compareTo Method Can Return Any Integer, Not Just -1, 0, and 1 660	Enhancing the Insertion Sort Algorithm 859
<b>15</b> The Java Collections Framework		Choosing a Collection Word Frequency Simulating a Queue of Waiting Customers 688
<b>16</b> Basic Data Structures		Implementing a Doubly-Linked List 859
<b>17</b> Tree Structures		Building a Huffman Tree Implementing a Red-Black Tree 859
<b>18</b> Generic Classes	Genericity and Inheritance The Array Store Exception Using Generic Types in a Static Context 829 829 834	Making a Generic Binary Search Tree Class 859
<b>19</b> Graphical User Interfaces	By Default, Components have Zero Width and Height 845	Laying Out a User Interface Programming a Working Calculator 859
<b>20</b> Streams and Binary Input/Output	Negative byte Values 887	Choosing a File Format 896

 Programming Tips	 Special Topics	 Computing & Society
	Oh, Omega, and Theta 638 Insertion Sort 639 The Quicksort Algorithm 646 The Parameterized Comparable Interface 660 The Comparator Interface 661	The First Programmer 652
Use Interface References to Manipulate Data Structures 685	Hash Functions 690 Reverse Polish Notation 703	Standardization 680
	Static Classes 730 Open Addressing 749	
	Wildcard Types 830 Reflection 834	
Use a GUI Builder 862	Adding the main Method to the Frame Class 846	



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

 Programming Tips	 Special Topics	 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
  - Computing & Society 1.1: Computers Are Everywhere* 5
- 1.3 THE JAVA PROGRAMMING LANGUAGE** 6
- 1.4 BECOMING FAMILIAR WITH YOUR PROGRAMMING ENVIRONMENT** 8
  - Programming Tip 1.1: Backup Copies* 11
- 1.5 ANALYZING YOUR FIRST PROGRAM** 12
  - Syntax 1.1: Java Program* 13
  - Common Error 1.1: Omitting Semicolons* 14

- 1.6 ERRORS** 15
  - Common Error 1.2: Misspelling Words* 16
- 1.7 PROBLEM SOLVING: ALGORITHM DESIGN** 16
  - How To 1.1: Describing an Algorithm with Pseudocode* 20
  - Worked Example 1.1: Writing an Algorithm for Tiling a Floor* 22



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?

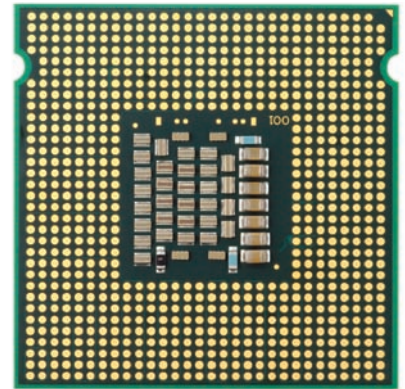
## 1.2 The Anatomy of a Computer

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 3). 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)



**Figure 1** Central Processing Unit

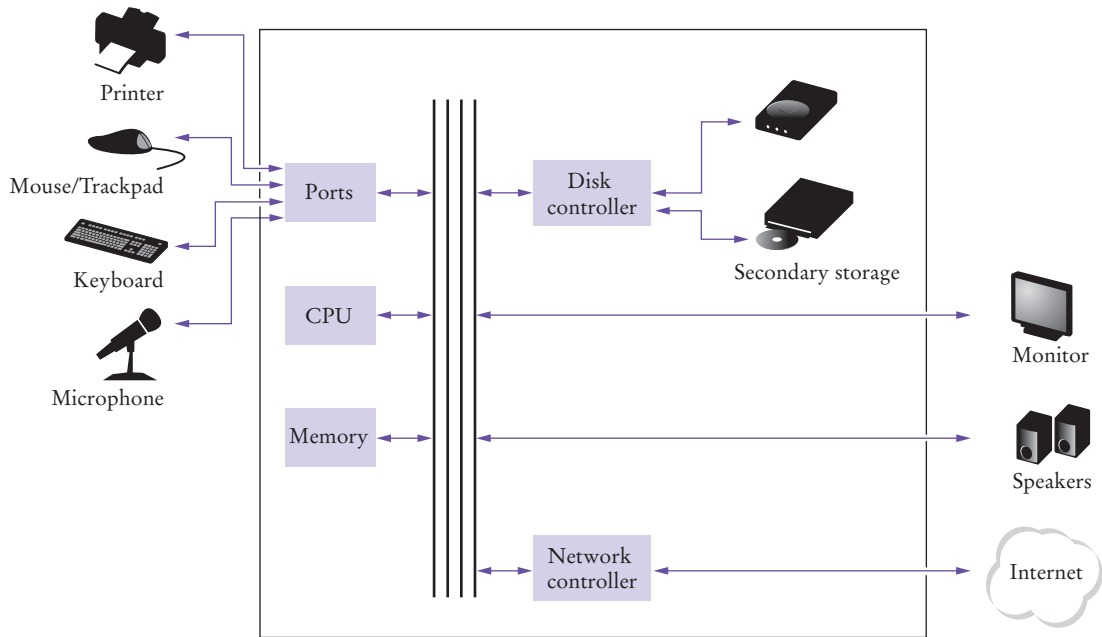
The central processing unit (CPU) performs program control and data processing.

Storage devices include memory and secondary storage.



**Figure 2** A Hard Disk





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

or a solid-state drive, provides slower and less expensive storage that persists without electricity. A hard disk consists of rotating platters, which are coated with a magnetic 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.



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

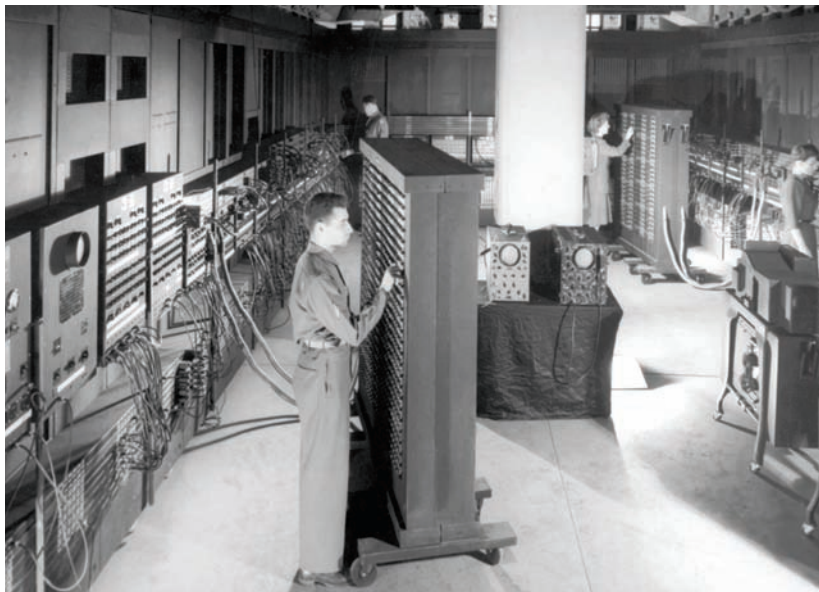


*This transit card contains a computer.*

could not have 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.



*The ENIAC*

## 1.3 The Java Programming Language

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.

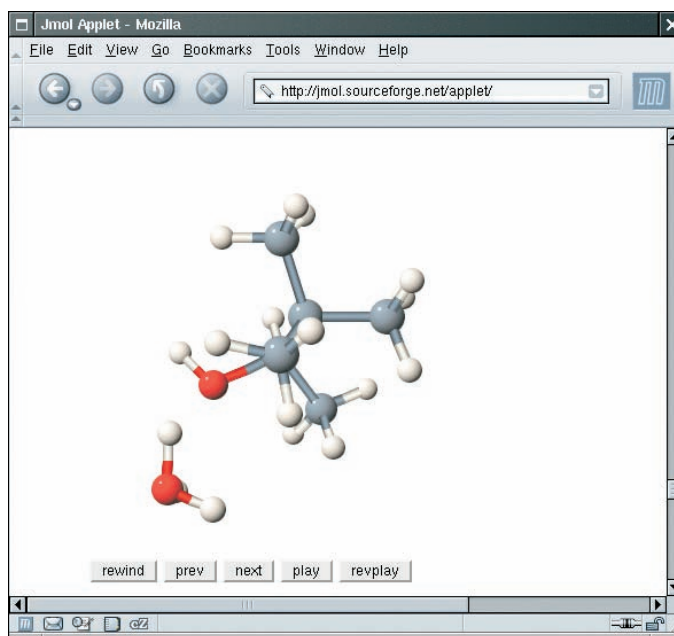
In 1991, a group led by James Gosling and Patrick Naughton at Sun Microsystems designed a programming language, code-named “Green”, for use in consumer devices, such as intelligent television “set-top” boxes. The language was designed to be simple, secure, and usable for many different processor types. 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. Figure 4 shows a typical example of an applet.



*James Gosling*

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



**Figure 4** An Applet for Visualizing Molecules Running in a Browser Window (<http://jmol.sourceforge.net/>)

Version	Year	Important New Features
1.0	1996	
1.1	1997	Inner classes
1.2	1998	Swing, Collections framework
1.3	2000	Performance enhancements
1.4	2002	Assertions, XML support
5	2004	Generic classes, enhanced for loop, auto-boxing, enumerations, annotations
6	2006	Library improvements
7	2011	Small language changes and library improvements

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 and cell phones to the largest Internet servers.

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

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

You can run a Java program in your 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.

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

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 5 or later.

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

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.



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 some 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.

**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:

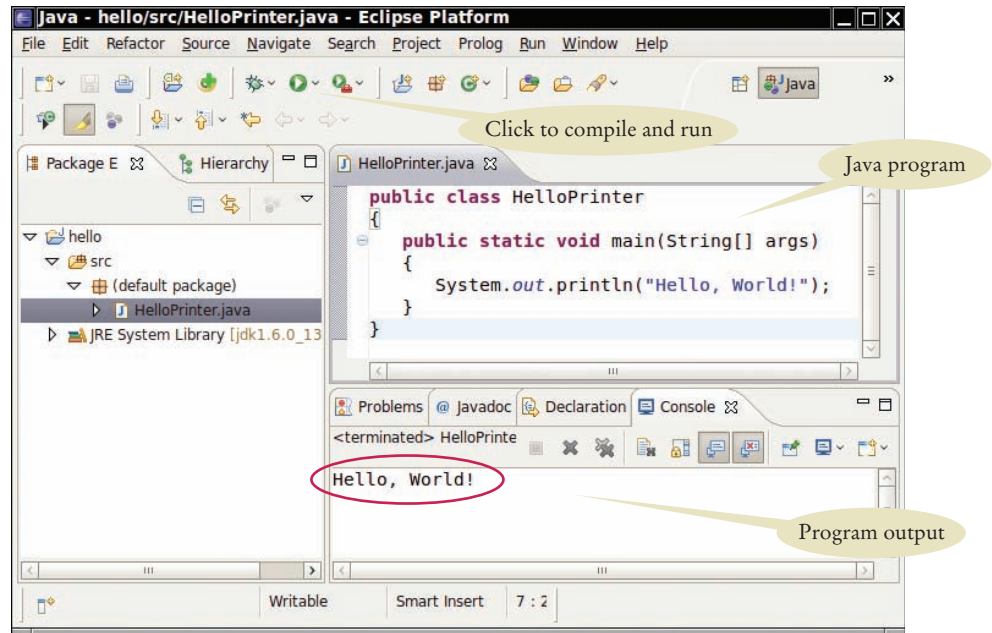
```
public class HelloPrinter
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

We will examine this program in the next section.

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



**Figure 5**  
Running the  
HelloPrinter  
Program in an  
Integrated  
Development  
Environment



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

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

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

As you write this program, pay careful attention to the various symbols, and keep in mind that Java is **case sensitive**. You must enter upper- and lowercase letters exactly as they appear in the program listing. You cannot type `MAIN` or `PrintLn`. If you are not careful, you will run into problems—see Common Error 1.2 on page 16.

**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 5 and 6).



**Figure 6**  
Running the HelloPrinter  
Program in a Console Window