



Cay Horstmann

Java Concepts

3/e

Late Objects

Includes Java 8 coverage

WILEY

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)	

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

Loop Statements

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

Executed while condition is true

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

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
}
```

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)

Class Declaration

```
public class BankAccount
{
    private double balance;
    private int transactions;
}
public BankAccount(double initialBalance)
{
    balance = initialBalance;
    transactions = 1;
}
public void deposit(double amount)
{
    balance = balance + amount;
    transactions++;
}
...
}
```

Instance variables

Constructor

Method

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

Loop body executed at least once

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

Set to a new element in each iteration

An array or collection

Executed for each element

Java Concepts

3/e

Late Objects

Cay Horstmann

San Jose State University



Java Concepts

3/e Late Objects

WILEY

PUBLISHER	Laurie Rosatone
EDITORIAL DIRECTOR	Don Fowley
DEVELOPMENTAL EDITOR	Cindy Johnson
ASSISTANT DEVELOPMENT EDITOR	Ryann Dannelly
EXECUTIVE MARKETING MANAGER	Dan Sayre
SENIOR PRODUCTION EDITOR	Laura Abrams
SENIOR CONTENT MANAGER	Valerie Zaborski
EDITORIAL ASSISTANT	Anna Pham
SENIOR DESIGNER	Tom Nery
SENIOR PHOTO EDITOR	Billy Ray
PRODUCTION MANAGEMENT	Cindy Johnson
COVER IMAGES	(tiger) © Aprison Photography/Getty Images, Inc.; (bird) © Nengloveyou/Shutterstock; (monkey) © Ehlers/iStockphoto; (rhino) © irawansubingarphotography/Getty Images, Inc.

This book was set in 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 © 2017, 2013, 2010 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, website 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, website <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. If you have chosen to adopt this textbook for use in your course, please accept this book as your complimentary desk copy. Outside of the United States, please contact your local representative.

ISBN 13: 978-1-119-32102-6

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

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:

Present fundamentals first.

The book takes a traditional route, first stressing control structures, methods, procedural decomposition, and arrays. Objects are used when appropriate in the early chapters. Students start designing and implementing their own classes in Chapter 8.

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 and Video 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
- 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 code completion questions 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 focus is on test-driven development, encouraging students to test their programs systematically.

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 sections on interfaces and sorting have been updated to make these innovations optionally available.

In addition, Java 7 features such as the try-with-resources statement are now integrated into the text.

Optional JavaFX Coverage

For those who prefer to use JavaFX instead of Swing, there is a new online resource that covers graphical user-interface programming with JavaFX.

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/bjlo2interactivities>.

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.

```

if (weekday < 4)
{
    weekday++;
}
else
{
}

```

```

if (weekday < 5)
    weekday = 1;
    weekday = 0;

```

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

This book is intended for a two-semester introduction to programming that may also include algorithms, data structures, and/or applications.

Part A: Fundamentals (Chapters 1–7)

The first seven chapters follow a traditional approach to basic programming concepts. Students learn about control structures, stepwise refinement, and arrays. Objects are used only for input/output and string processing. Input/output is covered in

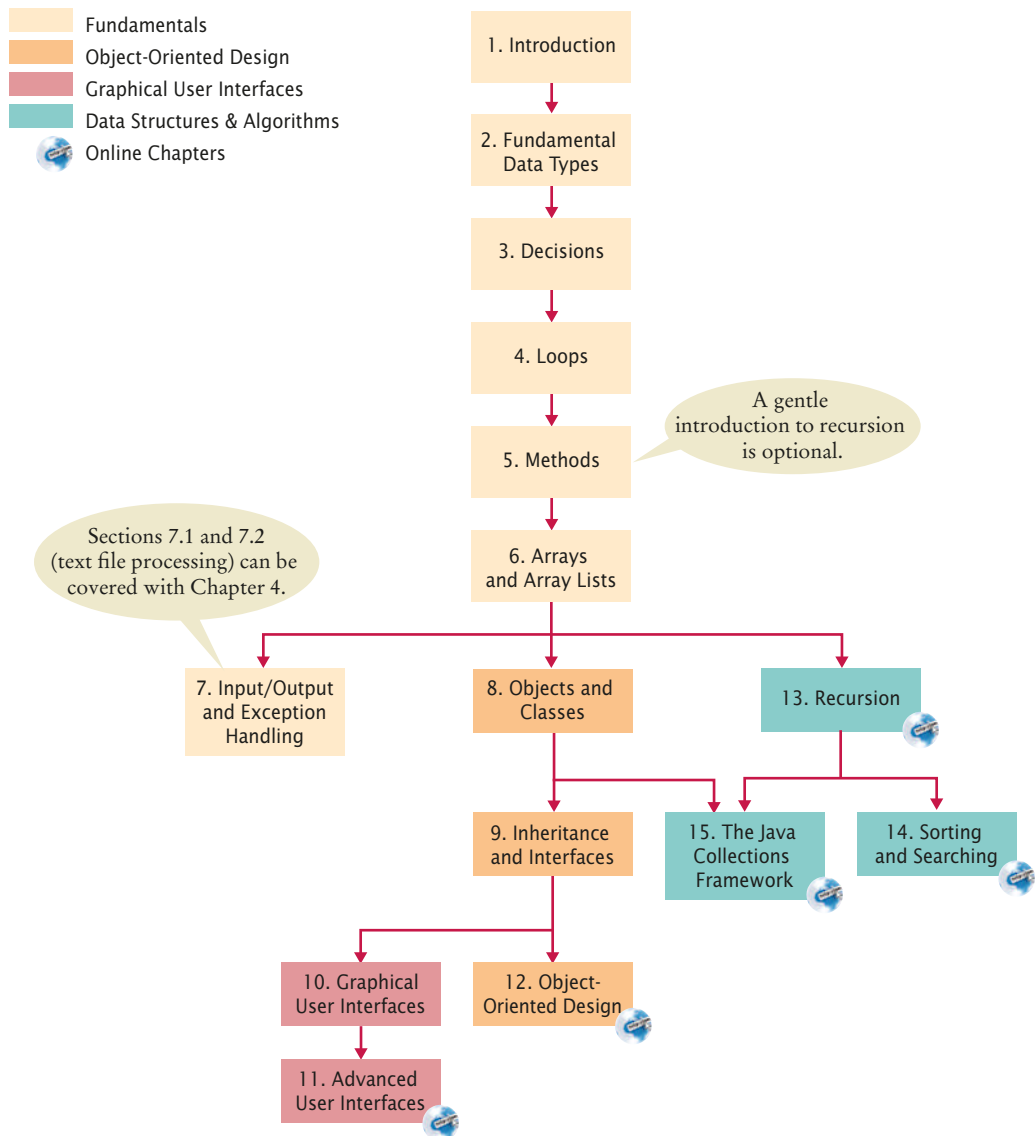


Figure 1 Chapter Dependencies

Chapter 7, but Sections 7.1 and 7.2 can be covered with Chapter 4; in that way, students can practice writing loops that process text files. Chapter 4 also provides an optional introduction to programming drawings that consist of lines, rectangles, and ovals, with an emphasis on reinforcing loops.

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

After students have gained a solid foundation, they are ready to tackle the implementation of classes in Chapter 8. Chapter 9 covers inheritance and interfaces. A simple methodology for object-oriented design is presented in Chapter 12. Object-oriented design may also be covered immediately after Chapter 9 by omitting the GUI versions of the sample programs. By the end of these chapters, students will be able to implement programs with multiple interacting classes.

Graphical user interfaces are presented in Chapters 10 and 11. The first of these chapters enables students to write programs with buttons, text components, and simple drawings. If you want to go deeper, you will find layout management and additional user-interface components in the second chapter. Online versions of these chapters cover JavaFX instead of Swing.

Part C: Data Structures and Algorithms (Chapters 13–15)

Chapters 13–15 cover algorithms and data structures at a level suitable for beginning students. 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. In Chapter 15, the Java Collections Framework is presented from the perspective of a library user, without revealing the implementations of lists and maps. You can cover this chapter anytime after Chapter 8. Chapters 11–15 are available in electronic form on the Web.

Any subset of these chapters can be incorporated into a custom print version of this text; ask your Wiley sales representative for details.

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 E–J are available on the Web.

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

Custom Book and eBook Options

Java Concepts may be ordered in both custom print and eBook formats. You can order a custom print version that includes your choice of chapters—including those from other Horstmann titles. Visit customselect.wiley.com to create your custom order.

Java Concepts is also available in an electronic eBook format with three key advantages:

- The price is significantly lower than for the printed book.
- The eBook contains all material in the printed book plus the web chapters and worked examples in one easy-to-browse format.
- You can customize the eBook to include your choice of chapters.

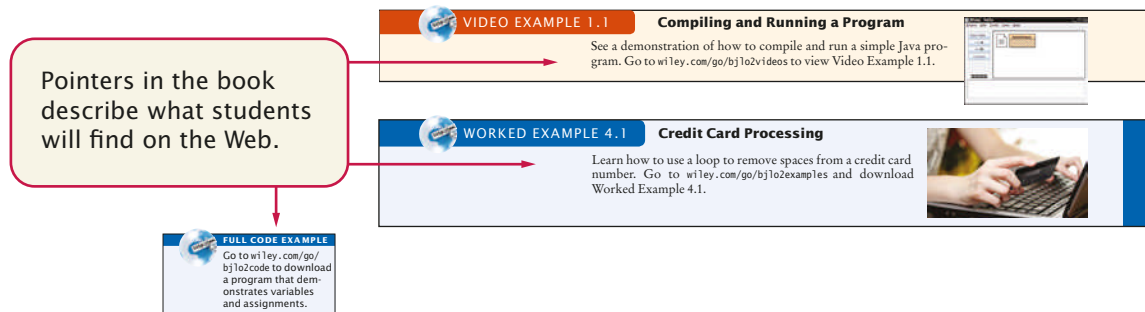
The interactive edition of *Java Concepts* adds even more value by integrating a wealth of interactive exercises into the eBook. See <http://wiley.com/go/bj1o2interactivities> to find out more about this new format.

Please contact your Wiley sales rep for more information about any of these options or check 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 to visit the online companion sites, which include

- Source code for all example programs in the book and its Worked Examples and Video Examples, plus additional example programs.
- Worked Examples that apply the problem-solving steps in the book to other realistic examples.
- Video Examples in which the author explains the steps he is taking and shows his work as he solves a programming problem.
- Lecture presentation slides (for instructors only).
- Solutions to all review and programming exercises (for instructors only).
- A test bank that focuses on skills, not just terminology (for instructors only). This extensive set of multiple-choice questions can be used with a word processor or imported into a course management system.
- “CodeCheck” assignments that allow students to work on programming problems presented in an innovative online service and receive immediate feedback. Instructors can assign exercises that have already been prepared, or easily add their own. Visit <http://codecheck.it> to learn more.



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.

150 Chapter 4 Loops

4.3 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.

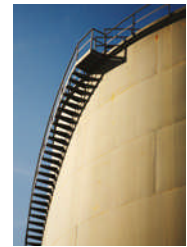
It often happens that you want to execute a sequence of statements a given number of times. You can use a `while` loop that is controlled by a counter, as in the following example:

```
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 4.2).

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

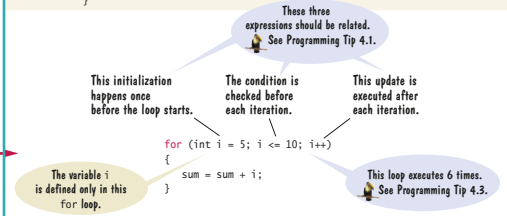
Some people call this loop *count-controlled*. In contrast, the `while` loop of the preceding section can be called an *event-controlled* loop because it executes until an event occurs; namely that the balance reaches the target. Another commonly used term for a count-controlled loop is *definite*. You 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 4.2 for Statement

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

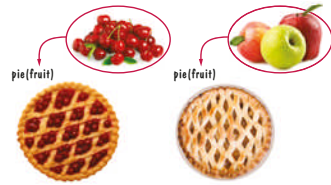


Like a variable in a computer program, a parking space has an identifier and a contents.



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.



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.

6.5 Problem Solving: Discovering Algorithms by Manipulating Physical Objects 277

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:



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.

HOW TO 1.1

Describing an Algorithm with Pseudocode

This is the first of many "How To" sections in this book that give you step-by-step procedures for carrying out important tasks in developing computer programs.

Before you are ready to write a program in Java, you need to develop an algorithm—a method for arriving at a solution for a particular problem. Describe the algorithm in pseudocode: a sequence of precise steps formulated in English.

For example, consider this problem: You have the choice of buying two cars. One is more fuel efficient than the other, but also more expensive. You know the price and fuel efficiency (in miles per gallon, mpg) of both cars. You plan to keep the car for ten years. Assume a price of \$4 per gallon of gas and usage of 15,000 miles per year. You will pay cash for the car and not worry about financing costs. Which car is the better deal?

Step 1 Determine the inputs and outputs.

In our sample problem, we have these inputs:

- `purchase price1` and `fuel efficiency1`, the price and fuel efficiency (in mpg) of the first car
- `purchase price2` and `fuel efficiency2`, the price and fuel efficiency of the second car

We simply want to know which car is the better buy. That is the desired output.

WORKED EXAMPLE 1.1

Writing an Algorithm for Tiling a Floor

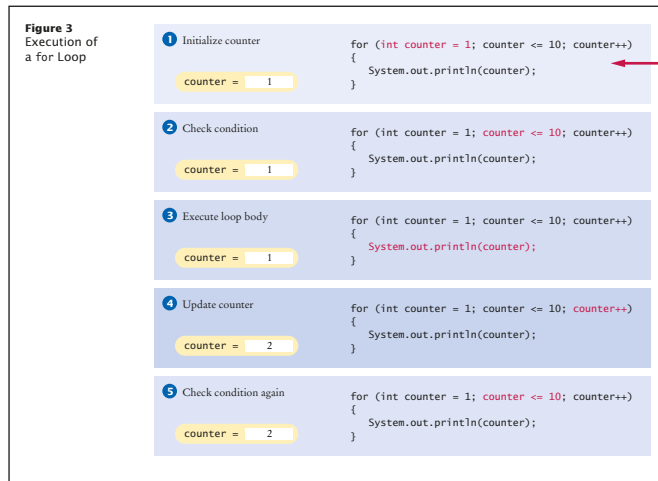
This Worked Example shows how to develop an algorithm for laying tile in an alternating pattern of colors.

Worked Examples and **Video 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 cans = 6;</code>	Declares an integer variable and initializes it with 6.
<code>int total = cans + bottles;</code>	The initial value need not be a constant. (Of course, <code>cans</code> and <code>bottles</code> must have been previously declared.)
<code>bottles = 1;</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 2.1.4.
<code>int bottles = "10";</code>	Error: You cannot initialize a number with a string.
<code>int bottles;</code>	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1 on page 37.
<code>int cans, bottles;</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.



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

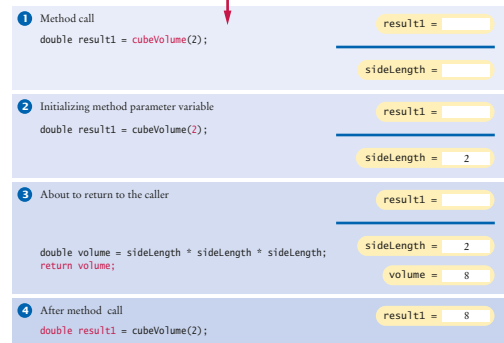


Figure 3 Parameter Passing

- The parameter variable `sideLength` of the `cubeVolume` method is created when the method is called. 1
- The parameter variable is initialized with the value of the argument that was passed in the call. In our case, `sideLength` is set to 2. 2
- The method computes the expression `sideLength * sideLength * sideLength`, which has the value 8. That value is stored in the variable `volume`. 3
- The method returns. All of its variables are removed. The return value is trans-

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.

SELF CHECK


11. Write the for loop of the `InvestmentTable.java` program as a while loop.
12. How many numbers does this loop print?

```
for (int n = 10; n >= 0; n--)  
{  
    System.out.println(n);  
}
```
13. Write a for loop that prints all even numbers between 10 and 20 (inclusive).
14. Write a for loop that computes the sum of the integers from 1 to n.
15. How would you modify the for loop of the `InvestmentTable.java` program to print all balances until the investment has doubled?


Practice It Now you can try these exercises at the end of the chapter: R4.7, R4.13, E4.8, E4.16.

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

Science P6.13 Sounds can be represented by an array of "sample values" that describe the intensity of the sound at a point in time. The program `ch06/sound/SoundEffect.java` reads a sound file (in WAV format), calls a method process for processing the sample values, and saves the sound file. Your task is to implement the process method by introducing an echo. For each



Business P9.6 Implement a superclass `Appointment` and subclasses `Onetime`, `Daily`, and `Monthly`. An appointment has a description (for example, "see the dentist") and a date and time. Write a method `occursOn(int year, int month, int day)` that checks whether the appointment occurs on that date. For example, for a monthly appointment, you must check whether the day of the month matches. Then fill an array of `Appointment` objects with a mixture of appointments. Have the user enter a date and print out all appointments that occur on that date.




```
sec01/DoubleInvestment.java
1 /**
2  * This program computes the time required to double an investment.
3  */
4 public class DoubleInvestment
5 {
6     public static void main(String[] args)
7     {
8         final double RATE = 5;
9         final double INITIAL_BALANCE = 10000;
10        final double TARGET = 2 * INITIAL_BALANCE;
11
12        double balance = INITIAL_BALANCE;
13        int year = 0;
14
15        // Count the years required for the investment to double
16
17        while (balance < TARGET)
18        {
19            year++;
20            double interest = balance * RATE / 100;
21            balance = balance + interest;
22        }
23
24        System.out.println("The investment doubled after "
25            + year + " years.");
26    }
27 }
```

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 6.4 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 3.5 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_LIMET)
37         {
38             tax1 = RATE1 * income;
39         }
40     }
41     else
42     {
43         tax1 = RATE1 * RATE1_SINGLE_LIMET;
44         tax2 = RATE2 * (income - RATE1_SINGLE_LIMET);
45     }
46 }
    
```

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




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.

Special Topic 7.2 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 9.3 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 Don Fowley, Graig Donini, Brad Franklin, Dan Sayre, Anna Pham, Laura Abrams, and Billy Ray at John Wiley & Sons 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, *University of Louisiana at Monroe*, Suzanne Dietrich, *Arizona State University, West Campus*, Byron Hoy, *Stockton University*, Brent Wilson, *George Fox University*, and David Woolbright, *Columbus State University*, for their contributions to the supplemental material.

Every new edition builds on the suggestions and experiences of new and prior reviewers, contributors, and users. Many thanks to the individuals who provided feedback, reviewed the manuscript, made valuable suggestions and contributions, and brought errors and omissions to my attention. They include:

- | | |
|--|--|
| Lynn Aaron, <i>SUNY Rockland
Community College</i> | Haiyan Cheng, <i>Willamette University</i> |
| Karen Arlien, <i>Bismarck State College</i> | Chakib Chraibi, <i>Barry University</i> |
| Jay Asundi, <i>University of Texas, Dallas</i> | Ta-Tao Chuang, <i>Gonzaga University</i> |
| Eugene Backlin, <i>DePaul University</i> | Vincent Cicirello, <i>Richard Stockton
College</i> |
| William C. Barge, <i>Trine University</i> | Mark Clement, <i>Brigham Young
University</i> |
| Bruce J. Barton, <i>Suffolk County
Community College</i> | Gerald Cohen, <i>St. Joseph's College</i> |
| Sanjiv K. Bhatia, <i>University of Missouri,
St. Louis</i> | Ralph Conrad, <i>San Antonio College</i> |
| Anna Bieszcza, <i>California State
University, Channel Islands</i> | Dave Cook, <i>Stephen F. Austin State
University</i> |
| Jackie Bird, <i>Northwestern University</i> | Rebecca Crellin, <i>Community College of
Allegheny County</i> |
| Eric Bishop, <i>Northland Pioneer College</i> | Leslie Damon, <i>Vermont Technical
College</i> |
| Paul Bladek, <i>Edmonds Community
College</i> | Geoffrey D. Decker, <i>Northern Illinois
University</i> |
| Paul Logasa Bogen II, <i>Texas A&M
University</i> | Khaled Deeb, <i>Barry University, School
of Adult and Continuing Education</i> |
| Irene Bruno, <i>George Mason University</i> | Akshaye Dhawan, <i>Ursinus College</i> |
| Paolo Bucci, <i>Ohio State University</i> | Julius Dichter, <i>University of Bridgeport</i> |
| Joe Burgin, <i>College of Southern
Maryland</i> | Mike Domaratzki, <i>University of
Manitoba</i> |
| Robert P. Burton, <i>Brigham Young
University</i> | Philip Dorin, <i>Loyola Marymount
University</i> |
| Leonello Calabresi, <i>University of
Maryland University College</i> | Anthony J. Dos Reis, <i>SUNY New Paltz</i> |
| Martine Ceberio, <i>University of Texas,
El Paso</i> | Elizabeth Drake, <i>Santa Fe College</i> |
| Uday Chakraborty, <i>University of
Missouri, St. Louis</i> | Tom Duffy, <i>Norwalk Community
College</i> |
| Suchindran Chatterjee, <i>Arizona State
University</i> | Michael Eckmann, <i>Skidmore College</i> |
| Xuemin Chen, <i>Texas Southern
University</i> | Sander Eller, <i>California State
Polytechnic University, Pomona</i> |
| | Amita Engineer, <i>Valencia Community
College</i> |

- Dave Evans, *Pasadena Community College*
James Factor, *Alverno College*
Chris Fietkiewicz, *Case Western Reserve University*
Terrell Foty, *Portland Community College*
Valerie Frear, *Daytona State College*
Zhenguang Gao, *Framingham State University*
Ryan Garlick, *University of North Texas*
Aaron Garrett, *Jacksonville State University*
Stephen Gilbert, *Orange Coast College*
Rick Giles, *Acadia University*
Peter van der Goes, *Rose State College*
Billie Goldstein, *Temple University*
Michael Gourley, *University of Central Oklahoma*
Grigoriy Grinberg, *Montgomery College*
Linwu Gu, *Indiana University*
Sylvain Guinepain, *University of Oklahoma, Norman*
Bruce Haft, *Glendale Community College*
Nancy Harris, *James Madison University*
Allan M. Hart, *Minnesota State University, Mankato*
Ric Heishman, *George Mason University*
Guy Helmer, *Iowa State University*
Katherin Herbert, *Montclair State University*
Rodney Hoffman, *Occidental College*
May Hou, *Norfolk State University*
John Houlihan, *Loyola University*
Andree Jacobson, *University of New Mexico*
Eric Jiang, *University of San Diego*
Christopher M. Johnson, *Guilford College*
Jonathan Kapleau, *New Jersey Institute of Technology*
Amitava Karmaker, *University of Wisconsin, Stout*
Rajkumar Kempaiah, *College of Mount Saint Vincent*
Mugdha Khaladkar, *New Jersey Institute of Technology*
Richard Kick, *Newbury Park High School*
Julie King, *Sullivan University, Lexington*
Samuel Kohn, *Touro College*
April Kontostathis, *Ursinus College*
Ron Krawitz, *DeVry University*
Nat Kumaresan, *Georgia Perimeter College*
Debbie Lamprecht, *Texas Tech University*
Jian Lin, *Eastern Connecticut State University*
Hunter Lloyd, *Montana State University*
Cheng Luo, *Coppin State University*
Kelvin Lwin, *University of California, Merced*
Frank Malinowski, *Dalton College*
John S. Mallozzi, *Iona College*
Khaled Mansour, *Washtenaw Community College*
Kenneth Martin, *University of North Florida*
Deborah Mathews, *J. Sargeant Reynolds Community College*
Louis Mazzucco, *State University of New York at Cobleskill and Excelsior College*
Drew McDermott, *Yale University*
Patricia McDermott-Wells, *Florida International University*
Hugh McGuire, *Grand Valley State University*
Michael L. Mick, *Purdue University, Calumet*
Jeanne Milostan, *University of California, Merced*
Sandeep Mitra, *SUNY Brockport*
Michel Mitri, *James Madison University*
Kenrick Mock, *University of Alaska Anchorage*
Namdar Mogharreban, *Southern Illinois University*
Jose-Arturo Mora-Soto, *University of Madrid*
Shamsi Moussavi, *Massbay Community College*

- Nannette Napier, *Georgia Gwinnett College*
 Tony Tuan Nguyen, *De Anza College*
 Michael Ondrasek, *Wright State University*
 K. Palaniappan, *University of Missouri*
 James Papademas, *Oakton Community College*
 Gary Parker, *Connecticut College*
 Jody Paul, *Metropolitan State College of Denver*
 Mark Pendergast, *Florida Gulf Coast University*
 James T. Pepe, *Bentley University*
 Jeff Pittges, *Radford University*
 Tom Plunkett, *Virginia Tech*
 Linda L. Preece, *Southern Illinois University*
 Vijay Ramachandran, *Colgate University*
 Craig Reinhart, *California Lutheran University*
 Jonathan Robinson, *Touro College*
 Chaman Lal Sabharwal, *Missouri University of Science & Technology*
 Katherine Salch, *Illinois Central College*
 Namita Sarawagi, *Rhode Island College*
 Ben Schafer, *University of Northern Iowa*
 Walter Schilling, *Milwaukee School of Engineering*
 Jeffrey Paul Scott, *Blackhawk Technical College*
 Amon Seagull, *NOVA Southeastern University*
 Brent Seales, *University of Kentucky*
 Linda Seiter, *John Carroll University*
 Kevin Seppi, *Brigham Young University*
 Ricky J. Sethi, *UCLA, USC ISI, and DeVry University*
 Narasimha Shashidhar, *Sam Houston State University*
 Ali Shaykhian, *Florida Institute of Technology*
 Lal Shimpi, *Saint Augustine's College*
 Victor Shtern, *Boston University*
 Rahul Simha, *George Washington University*
 Jeff Six, *University of Delaware*
 Donald W. Smith, *Columbia College*
 Derek Snow, *University of Southern Alabama*
 Peter Spoerri, *Fairfield University*
 David R. Stampf, *Suffolk County Community College*
 Peter Stanchev, *Kettering University*
 Ryan Stansifer, *Florida Institute of Technology*
 Stu Steiner, *Eastern Washington University*
 Robert Strader, *Stephen F. Austin State University*
 David Stucki, *Otterbein University*
 Ashok Subramanian, *University of Missouri, St Louis*
 Jeremy Suing, *University of Nebraska, Lincoln*
 Dave Sullivan, *Boston University*
 Vaidy Sunderam, *Emory University*
 Hong Sung, *University of Central Oklahoma*
 Monica Sweat, *Georgia Tech University*
 Joseph Szurek, *University of Pittsburgh, Greensburg*
 Jack Tan, *University of Wisconsin*
 Cynthia Tanner, *West Virginia University*
 Russell Tessier, *University of Massachusetts, Amherst*
 Krishnaprasad Thirunarayan, *Wright State University*
 Mark Thomas, *University of Cincinnati Blue Ash*
 Megan Thomas, *California State University, Stanislaus*
 Timothy Urness, *Drake University*
 Eliana Valenzuela-Andrade, *University of Puerto Rico at Arecibo*
 Tammy VanDeGrift, *University of Portland*
 Philip Ventura, *Broward College*
 David R. Vineyard, *Kettering University*
 Qi Wang, *Northwest Vista College*
 Jonathan Weissman, *Finger Lakes Community College*
 Reginald White, *Black Hawk Community College*
 Ying Xie, *Kennesaw State University*
 Arthur Yanushka, *Christian Brothers University*

xviii Acknowledgments

Chen Ye, *University of Illinois, Chicago*
Wook-Sung Yoo, *Fairfield University*
Steve Zale, *Middlesex County College*
Bahram Zartoshty, *California State
University, Northridge*
Frank Zeng, *Indiana Wesleyan
University*

Hairong Zhao, *Purdue University
Calumet*
Stephen Zilora, *Rochester Institute of
Technology*

And a special thank you to our class testers:



Eugene Backlin and the students of DePaul University, Loop
Debra M. Duke and the students of J. Sargeant Reynolds Community College
Gerald Gordon and the students of DePaul University, Loop
Mike Gourley and the students of the University of Central Oklahoma
Mohammad Morovati and the students of the College of DuPage
Mutsumi Nakamura and the students of Arizona State University
George Novacky and the students of the University of Pittsburgh
Darrin Rothe and the students of the Milwaukee School of Engineering
Paige Rutner and the students of Georgia Southern University
Narasimha Shashidhar and the students of Sam Houston State University
Mark Sherriff and the students of the University of Virginia
Frank Zeng and the students of Indiana Wesleyan University

CONTENTS

PREFACE **v**

SPECIAL FEATURES **xxvi**

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**
 - VE1** Compiling and Running a Program 
- 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
 - VE2** Dividing Household Expenses 

2 FUNDAMENTAL DATA TYPES **31**

- 2.1 Variables **32**
 - Variable Declarations 32
 - Number Types 34
 - Variable Names 35
 - The Assignment Statement 36
 - Constants 37
 - Comments 37
 - ST1** Numeric Types in Java 41
 - ST2** Big Numbers 42
- 2.2 Arithmetic **43**
 - Arithmetic Operators 43
 - Increment and Decrement 43
 - Integer Division and Remainder 44
 - Powers and Roots 45

Converting Floating-Point Numbers to Integers 46

J81 Avoiding Negative Remainders 49

ST3 Combining Assignment and Arithmetic 49


VE1 Using Integer Division 

2.3 Input and Output **50**


Reading Input 50

Formatted Output 52

HT1 Carrying Out Computations 56

WE1 Computing the Cost of Stamps 

2.4 PROBLEM SOLVING First Do it By Hand **59**

WE2 Computing Travel Time 

2.5 Strings **61**

The String Type 61

Concatenation 61

String Input 62

Escape Sequences 62

Strings and Characters 63

Substrings 63

ST4 Instance Methods and Static Methods 66

ST5 Using Dialog Boxes for Input and Output 67

VE2 Computing Distances on Earth 

3 DECISIONS **83**


3.1 The if Statement **84**

ST1 The Conditional Operator 89

3.2 Comparing Numbers and Strings **90**

ST2 Lexicographic Ordering of Strings 94

HT1 Implementing an if Statement 95

WE1 Extracting the Middle 

3.3 Multiple Alternatives **98**

ST3 The switch Statement 101

3.4 Nested Branches **102**

ST4 Enumeration Types 107

VE1 Computing the Plural of an English Word 





3.5 PROBLEM SOLVING Flowcharts **107**










3.6 PROBLEM SOLVING Test Cases **110**

ST5 Logging 112

3.7 Boolean Variables and Operators **113**

xx Contents





- ST6 Short-Circuit Evaluation of Boolean Operators 116
 - ST7 De Morgan's Law 117
 - 3.8 APPLICATION Input Validation 118
 - VE2 The Genetic Code 
- 4 LOOPS 142**
 - 4.1 The while Loop 142
 - 4.2 PROBLEM SOLVING Hand-Tracing 149
 - 4.3 The for Loop 152
 - 4.4 The do Loop 158
 - 4.5 APPLICATION Processing Sentinel Values 160
 - ST1 The “Loop and a Half” Problem and the break Statement 162
 - ST2 Redirection of Input and Output 163
 - VE1 Evaluating a Cell Phone Plan 
 - 4.6 PROBLEM SOLVING Storyboards 164
 - 4.7 Common Loop Algorithms 167
 - Sum and Average Value 167
 - Counting Matches 167
 - Finding the First Match 168
 - Prompting Until a Match is Found 169
 - Maximum and Minimum 169
 - Comparing Adjacent Values 170
 - HT1 Writing a Loop 171
 - WE1 Credit Card Processing 
 - 4.8 Nested Loops 174
 - WE2 Manipulating the Pixels in an Image 
 - 4.9 PROBLEM SOLVING Solve a Simpler Problem First 178
 - 4.10 APPLICATION Random Numbers and Simulations 182
 - Generating Random Numbers 182
 - Simulating Die Tosses 183
 - The Monte Carlo Method 184
 - ST3 Drawing Graphical Shapes 186
 - VE1 Drawing a Spiral 
- 5 METHODS 211**
 - 5.1 Methods as Black Boxes 212
 - 5.2 Implementing Methods 214
 - 5.3 Parameter Passing 217
 - 5.4 Return Values 220
 - HT1 Implementing a Method 222
 - WE1 Generating Random Passwords 
 - 5.5 Methods Without Return Values 224
 - 5.6 PROBLEM SOLVING Reusable Methods 225
 - 5.7 PROBLEM SOLVING Stepwise Refinement 229
 - WE2 Calculating a Course Grade 
 - 5.8 Variable Scope 236
 - VE1 Debugging 
 - 5.9 Recursive Methods (Optional) 240
 - HT2 Thinking Recursively 243
 - VE2 Fully Justified Text 
- 6 ARRAYS AND ARRAY LISTS 261**
 - 6.1 Arrays 262
 - Declaring and Using Arrays 262
 - Array References 265
 - Partially Filled Arrays 266
 - 6.2 The Enhanced for Loop 269
 - 6.3 Common Array Algorithms 270
 - Filling 270
 - Sum and Average Value 271
 - Maximum and Minimum 271
 - Element Separators 271
 - Linear Search 272
 - Removing an Element 272
 - Inserting an Element 273
 - Swapping Elements 274
 - Copying Arrays 275
 - Reading Input 276
 - ST1 Sorting with the Java Library 279
 - ST2 Binary Search 279
 - 6.4 Using Arrays with Methods 280
 - ST3 Methods with a Variable Number of Parameters 284
 - 6.5 PROBLEM SOLVING Adapting Algorithms 284
 - HT1 Working with Arrays 287
 - WE1 Rolling the Dice 

- 6.6 PROBLEM SOLVING Discovering Algorithms by Manipulating Physical Objects **291**
 - VE1** Removing Duplicates from an Array 
 - 6.7 Two-Dimensional Arrays **294**
 - Declaring Two-Dimensional Arrays 295
 - Accessing Elements 295
 - Locating Neighboring Elements 296
 - Computing Row and Column Totals 297
 - Two-Dimensional Array Parameters 298
 - WE2** A World Population Table 
 - ST4** Two-Dimensional Arrays with Variable Row Lengths 300
 - ST5** Multidimensional Arrays 301
 - 6.8 Array Lists **301**
 - Declaring and Using Array Lists 302
 - Using the Enhanced for Loop with Array Lists 304
 - Copying Array Lists 305
 - Array Lists and Methods 305
 - Wrappers and Auto-boxing 305
 - Using Array Algorithms with Array Lists 307
 - Storing Input Values in an Array List 307
 - Removing Matches 307
 - Choosing Between Array Lists and Arrays 308
 - ST6** The Diamond Syntax 311
 - VE2** Game of Life 
- 7 INPUT/OUTPUT AND EXCEPTION HANDLING 331**
- 7.1 Reading and Writing Text Files **332**
 - ST1** Reading Web Pages 335
 - ST2** File Dialog Boxes 335
 - ST3** Reading and Writing Binary Data 336
 - 7.2 Text Input and Output **337**
 - Reading Words 337
 - Reading Characters 338
 - Classifying Characters 338
 - Reading Lines 339
 - Scanning a String 340
 - Converting Strings to Numbers 340
 - Avoiding Errors When Reading Numbers 340
 - Mixing Number, Word, and Line Input 341
 - Formatting Output 342
 - ST4** Regular Expressions 344
 - ST5** Reading an Entire File 344
 - VE1** Computing a Document's Readability 
 - 7.3 Command Line Arguments **345**
 - HT1** Processing Text Files 348
 - WE1** Analyzing Baby Names 
 - 7.4 Exception Handling **352**
 - Throwing Exceptions 352
 - Catching Exceptions 354
 - Checked Exceptions 355
 - Closing Resources 357
 - ST6** Assertions 360
 - ST7** The try/finally Statement 360
 - 7.5 APPLICATION Handling Input Errors **361**
 - VE2** Detecting Accounting Fraud 
- 8 OBJECTS AND CLASSES 375**
- 8.1 Object-Oriented Programming **376**
 - 8.2 Implementing a Simple Class **378**
 - 8.3 Specifying the Public Interface of a Class **381**
 - ST1** The javadoc Utility 384
 - 8.4 Designing the Data Representation **385**
 - 8.5 Implementing Instance Methods **386**
 - 8.6 Constructors **389**
 - ST2** Overloading 393
 - 8.7 Testing a Class **393**
 - HT1** Implementing a Class 395
 - WE1** Implementing a Menu Class 
 - VE1** Paying Off a Loan 
 - 8.8 Problem Solving: Tracing Objects **399**
 - 8.9 Object References **403**
 - Shared References 403
 - The null Reference 405
 - The this Reference 405
 - ST3** Calling One Constructor from Another 408
 - 8.10 Static Variables and Methods **408**
 - 8.11 PROBLEM SOLVING Patterns for Object Data **410**
 - Keeping a Total 411
 - Counting Events 411
 - Collecting Values 412
 - Managing Properties of an Object 413
 - Modeling Objects with Distinct States 413
 - Describing the Position of an Object 414
 - VE2** Modeling a Robot Escaping from a Maze 



8.12 Packages 417

- Organizing Related Classes into Packages 417
- Importing Packages 418
- Package Names 419
- Packages and Source Files 419
- ST4** Package Access 421
- HT2** Programming with Packages 421

9 INHERITANCE AND INTERFACES 437

- 9.1 Inheritance Hierarchies 438**
- 9.2 Implementing Subclasses 442**
- 9.3 Overriding Methods 446**
 - ST1** Calling the Superclass Constructor 451
- 9.4 Polymorphism 452**
 - ST2** Dynamic Method Lookup and the Implicit Parameter 455
 - ST3** Abstract Classes 456
 - ST4** Final Methods and Classes 457
 - ST5** Protected Access 458
 - HT1** Developing an Inheritance Hierarchy 458
 - WE1** Implementing an Employee Hierarchy for Payroll Processing 
 - VE1** Building a Discussion Board 
- 9.5 Object: The Cosmic Superclass 463**
 - Overriding the toString Method 464
 - The equals Method 465
 - The instanceof Operator 466
 - ST6** Inheritance and the toString Method 468
 - ST7** Inheritance and the equals Method 469
- 9.6 Interface Types 470**
 - Defining an Interface 470
 - Implementing an Interface 472
 - The Comparable Interface 474
 - ST8** Constants in Interfaces 476
 - ST9** Generic Interface Types 476
 - J81** Static Methods in Interfaces 477
 - J82** Default Methods 477
 - ST10** Function Objects 478
 - J83** Lambda Expressions 479
 - WE2** Investigating Number Sequences 
 - VE2** Drawing Geometric Shapes 

10 GRAPHICAL USER INTERFACES 493

- 10.1 Frame Windows 494**
 - Displaying a Frame 494
 - Adding User-Interface Components to a Frame 495
 - Using Inheritance to Customize Frames 497
 - ST1** Adding the main Method to the Frame Class 498
- 10.2 Events and Event Handling 498**
 - Listening to Events 499
 - Using Inner Classes for Listeners 501
 - APPLICATION** Showing Growth of an Investment 503
 - ST2** Local Inner Classes 507
 - ST3** Anonymous Inner Classes 508
 - J81** Lambda Expressions for Event Handling 509
- 10.3 Processing Text Input 509**
 - Text Fields 509
 - Text Areas 511
- 10.4 Creating Drawings 515**
 - Drawing on a Component 515
 - Ovals, Lines, Text, and Color 517
 - APPLICATION** Visualizing the Growth of an Investment 520
 - HT1** Drawing Graphical Shapes 525
 - WE1** Coding a Bar Chart Creator 
 - VE1** Solving Crossword Puzzles 

11 ADVANCED USER INTERFACES (WEB ONLY) 

- 11.1 Layout Management**
- 11.2 Choices**
 - Radio Buttons
 - Check Boxes
 - Combo Boxes
 - HT1** Laying Out a User Interface
 - WE1** Programming a Working Calculator
- 11.3 Menus**
- 11.4 Exploring the Swing Documentation**
- 11.5 Using Timer Events for Animations**

11.6 Mouse Events

- ST1** Keyboard Events
- ST2** Event Adapters
- WE2** Adding Mouse and Keyboard Support to the Bar Chart Creator
- VE1** Designing a Baby Naming Program

12 OBJECT-ORIENTED DESIGN (WEB ONLY) **12.1** Classes and Their Responsibilities

- Discovering Classes
- The CRC Card Method
- Cohesion

12.2 Relationships Between Classes

- Dependency
- Aggregation
- Inheritance
- HT1** Using CRC Cards and UML Diagrams in Program Design
- ST1** Attributes and Methods in UML Diagrams
- ST2** Multiplicities
- ST3** Aggregation, Association, and Composition

12.3 APPLICATION Printing an Invoice

- Requirements
- CRC Cards
- UML Diagrams
- Method Documentation
- Implementation
- WE1** Simulating an Automatic Teller Machine

13 RECURSION (WEB ONLY) **13.1** Triangle Numbers

- HT1** Thinking Recursively
- WE1** Finding Files

13.2 Recursive Helper Methods**13.3** The Efficiency of Recursion**13.4** Permutations**13.5** Mutual Recursion**13.6** Backtracking

- WE2** Towers of Hanoi

14 SORTING AND SEARCHING (WEB ONLY) **14.1** Selection Sort**14.2** Profiling the Selection Sort Algorithm**14.3** Analyzing the Performance of the Selection Sort Algorithm

- ST1** Oh, Omega, and Theta
- ST2** Insertion Sort

14.4 Merge Sort**14.5** Analyzing the Merge Sort Algorithm

- ST3** The Quicksort Algorithm

14.6 Searching


- Linear Search
- Binary Search

14.7 PROBLEM SOLVING Estimating the Running Time of an Algorithm

- Linear Time
- Quadratic Time
- The Triangle Pattern
- Logarithmic Time

14.8 Sorting and Searching in the Java Library

- Sorting
- Binary Search
- Comparing Objects
- ST4** The Comparator Interface
- J81** Comparators with Lambda Expressions
- WE1** Enhancing the Insertion Sort Algorithm

15 THE JAVA COLLECTIONS FRAMEWORK (WEB ONLY) **15.1** An Overview of the Collections Framework**15.2** Linked Lists

- The Structure of Linked Lists
- The `LinkedList` Class of the Java Collections Framework
- List Iterators

15.3 Sets

- Choosing a Set Implementation
- Working with Sets

15.4 Maps

J81 Updating Map Entries

HT1 Choosing a Collection

WE1 Word Frequency

ST1 Hash Functions

15.5 Stacks, Queues, and Priority Queues

Stacks

Queues

Priority Queues

15.6 Stack and Queue Applications

Balancing Parentheses

Evaluating Reverse Polish Expressions

Evaluating Algebraic Expressions

Backtracking

WE2 Simulating a Queue of Waiting Customers

VE1 Building a Table of Contents


ST2 Reverse Polish Notation

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 TOOL SUMMARY 

APPENDIX F NUMBER SYSTEMS 

APPENDIX G UML SUMMARY 

APPENDIX H JAVA SYNTAX SUMMARY 

APPENDIX I HTML SUMMARY 











GLOSSARY **G-1**





INDEX **I-1**

CREDITS **C-1**

ALPHABETICAL LIST OF SYNTAX BOXES

Arrays	263
Array Lists	302
Assignment	36
Cast	46
Catching Exceptions	354
Comparisons	91
Constant Declaration	37
Constructor with Superclass Initializer	452
Constructors	390
for Statement	154
if Statement	86
Input Statement	51
Instance Methods	387
Instance Variable Declaration	379
Interface Types	471
Java Program	12
Package Specification	419
Static Method Declaration	215
Subclass Declaration	444
The Enhanced for Loop	270
The instanceof Operator	467
The throws Clause	357
The try-with-resources Statement	357
Throwing an Exception	352
Two-Dimensional Array Declaration	295
Variable Declaration	33
while Statement	142

CHAPTER	 Common Errors	 How Tos and Worked Examples
1 Introduction	Omitting Semicolons 13 Misspelling Words 15	Compiling and Running a Program Video Describing an Algorithm with Pseudocode 19 Writing an Algorithm for Tiling a Floor  Dividing Household Expenses Video
2 Fundamental Data Types	Using Undeclared or Uninitialized Variables 39 Overflow 40 Roundoff Errors 40 Unintended Integer Division 48 Unbalanced Parentheses 48	Using Integer Division Video Carrying out Computations 56 Computing the Cost of Stamps  Computing Travel Time  Computing Distances on Earth Video
3 Decisions	A Semicolon After the if Condition 88 Exact Comparison of Floating-Point Numbers 93 Using == to Compare Strings 94 The Dangling else Problem 106 Combining Multiple Relational Operators 115 Confusing && and Conditions 116	Implementing an if Statement 95 Extracting the Middle  Computing the Plural of an English Word Video The Genetic Code Video
4 Loops	Don't Think "Are We There Yet?" 146 Infinite Loops 147 Off-by-One Errors 147	Evaluating a Cell Phone Plan Video Writing a Loop 171 Credit Card Processing  Manipulating the Pixels in an Image  Drawing a Spiral Video
5 Methods	Trying to Modify Arguments 219 Missing Return Value 222	Implementing a Method 222 Generating Random Passwords  Calculating a Course Grade  Debugging Video Thinking Recursively 243 Fully Justified Text Video

 Programming Tips		 Special Topics and Java 8 Notes 		 Computing & Society	
Backup Copies	10			Computers Are Everywhere	5
Choose Descriptive Variable Names	40	Numeric Types in Java	41	The Pentium Floating-Point Bug	50
Do Not Use Magic Numbers	41	Big Numbers	42	International Alphabets and Unicode	68
Spaces in Expressions	49	● Avoiding Negative Remainders	49		
Use the API Documentation	55	Combining Assignment and Arithmetic	49		
		Instance Methods and Static Methods	66		
		Using Dialog Boxes for Input and Output	67		
Brace Layout	88	The Conditional Operator	89	The Denver Airport Luggage System	97
Always Use Braces	88	Lexicographic Ordering of Strings	94	Artificial Intelligence	121
Tabs	89	The switch Statement	101		
Avoid Duplication in Branches	90	Enumeration Types	107		
Hand-Tracing	105	Logging	112		
Make a Schedule and Make Time for Unexpected Problems	111	Short-Circuit Evaluation of Boolean Operators	116		
		De Morgan's Law	117		
Use for Loops for Their Intended Purpose Only	157	The Loop-and-a-Half Problem and the break Statement	162	The First Bug	148
Choose Loop Bounds That Match Your Task	157	Redirection of Input and Output	163	Digital Piracy	188
Count Iterations	158	Drawing Graphical Shapes	186		
Flowcharts for Loops	159				
Method Comments	217			Personal Computing	228
Do Not Modify Parameter Variables	219				
Keep Methods Short	234				
Tracing Methods	234				
Stubs	235				
Using a Debugger	239				

CHAPTER



Common Errors

How Tos
and
Worked Examples**6** Arrays and Array Lists

Bounds Errors	267
Uninitialized Arrays	267
Underestimating the Size of a Data Set	279
Length and Size	311

Working with Arrays	287
Rolling the Dice	
Removing Duplicates from an Array	Video
A World Population Table	
Game of Life	Video

7 Input/Output and Exception Handling

Backslashes in File Names	335
Constructing a Scanner with a String	335

Computing a Document's Readability	Video
Processing Text Files	348
Analyzing Baby Names	
Detecting Accounting Fraud	Video

8 Objects and Classes





Trying to Call a Constructor	392
Declaring a Constructor as void	393
Forgetting to Initialize Object References in a Constructor	407
Confusing Dots	420

Implementing a Class	395
Implementing a Menu Class	
Paying Off a Loan	Video
Modeling a Robot Escaping from a Maze	Video
Programming with Packages	421

9 Inheritance and Interfaces

Replicating Instance Variables from the Superclass	445
Confusing Super- and Subclasses	446
Accidental Overloading	450
Forgetting to Use super When Invoking a Superclass Method	451
Don't Use Type Tests	468
Forgetting to Declare Implementing Methods as Public	475

Developing an Inheritance Hierarchy	458
Implementing an Employee Hierarchy for Payroll Processing	
Building a Discussion Board	Video
Investigating Number Sequences	
Drawing Geometric Shapes	Video

 Programming Tips	 Special Topics and Java 8 Notes 	 Computing & Society
Use Arrays for Sequences of Related Items 268 Reading Exception Reports 286	Sorting with the Java Library 279 Binary Search 279 Methods with a Variable Number of Parameters 284 Two-Dimensional Arrays with Variable Row Lengths 300 Multidimensional Arrays 301 The Diamond Syntax 311	Computer Viruses 268
Throw Early, Catch Late 359 Do Not Squelch Exceptions 359 Do Throw Specific Exceptions 360	Reading Web Pages 335 File Dialog Boxes 335 Reading and Writing Binary Data 336 Regular Expressions 344 Reading an Entire File 344 Assertions 360 The try/finally Statement 360	Encryption Algorithms 351 The Ariane Rocket Incident 361
All Instance Variables Should Be Private; Most Methods Should Be Public 388	The javadoc Utility 384 Overloading 393 Calling One Constructor from Another 408 Package Access 421	Open Source and Free Software 402 Electronic Voting Machines 416
Use a Single Class for Variation in Values, Inheritance for Variation in Behavior 442 Comparing Integers and Floating-Point Numbers 475	Calling the Superclass Constructor 451 Dynamic Method Lookup and the Implicit Parameter 455 Abstract Classes 456 Final Methods and Classes 457 Protected Access 458 Inheritance and the toString Method 468 Inheritance and the equals Method 469 Constants in Interfaces 476 Generic Interface Types 476 ● Static Methods in Interfaces 477 ● Default Methods 477 Function Objects 478 ● Lambda Expressions 479	Who Controls the Internet? 481


CHAPTER







Common Errors

How Tos
and
Worked Examples**10** Graphical User Interfaces



Modifying Parameter Types in the Implementing Method 506
 Forgetting to Attach a Listener 506
 Forgetting to Repaint 524
 By Default, Components Have Zero Width and Height 525


Drawing Graphical Shapes 525
 Coding a Bar Chart Creator
 Solving Crossword Puzzles  Video



11 Advanced User Interfaces(WEB ONLY) 




Laying Out a User Interface 
 Programming a Working Calculator 
 Adding Mouse and Keyboard Support to the Bar Chart Creator 
 Designing a Baby Naming Program 

12 Object-Oriented Design(WEB ONLY) 


Using CRC Cards and UML Diagrams in Program Design 
 Simulating an Automatic Teller Machine 


13 Recursion(WEB ONLY) 


Infinite Recursion 
 Tracing Through Recursive Methods 





Thinking Recursively 
 Finding Files 
 Towers of Hanoi 
















14 Sorting and Searching(WEB ONLY) 

The compareTo Method Can Return Any Integer, Not Just -1, 0, and 1 

Enhancing the Insertion Sort Algorithm 

15 The Java Collections Framework(WEB ONLY) 

Choosing a Collection 
 Word Frequency 
 Simulating a Queue of Waiting Customers 
 Building a Table of Contents 

 Programming Tips	 Special Topics and Java 8 Notes 	 Computing & Society
Don't Use a Frame as a Listener 506	Adding the main Method to the Frame Class 498 Local Inner Classes 507 Anonymous Inner Classes 508 ● Lambda Expressions for Event Handling 509	
Use a GUI Builder 	Keyboard Events  Event Adapters 	
Make Parallel Arrays into Arrays of Objects 	Attributes and Methods in UML Diagrams  Multiplicities  Aggregation, Association, and Composition 	Databases and Privacy 
		The Limits of Computation 
	Oh, Omega, and Theta  Insertion Sort  The Quicksort Algorithm  The Comparator Interface  ● Comparators with Lambda Expressions 	The First Programmer 
Use Interface References to Manipulate Data Structures 	● Updating Map Entries  Hash Functions  Reverse Polish Notation 	Standardization 

CHAPTER 1

INTRODUCTION



© JanPietruszka/iStockphoto.

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
 - VE1** Compiling and Running a Program 🌐
- 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
 - VE2** Dividing Household Expenses 🌐



© 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.

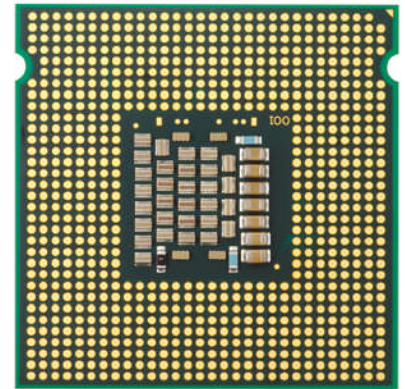
At the heart of the computer lies the **central processing unit (CPU)** (see Figure 1). The inside wiring of the CPU is enormously complicated. For example, the Intel Core processor (a popular CPU for personal computers at the time of this writing) is composed of several hundred million structural elements,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

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

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.

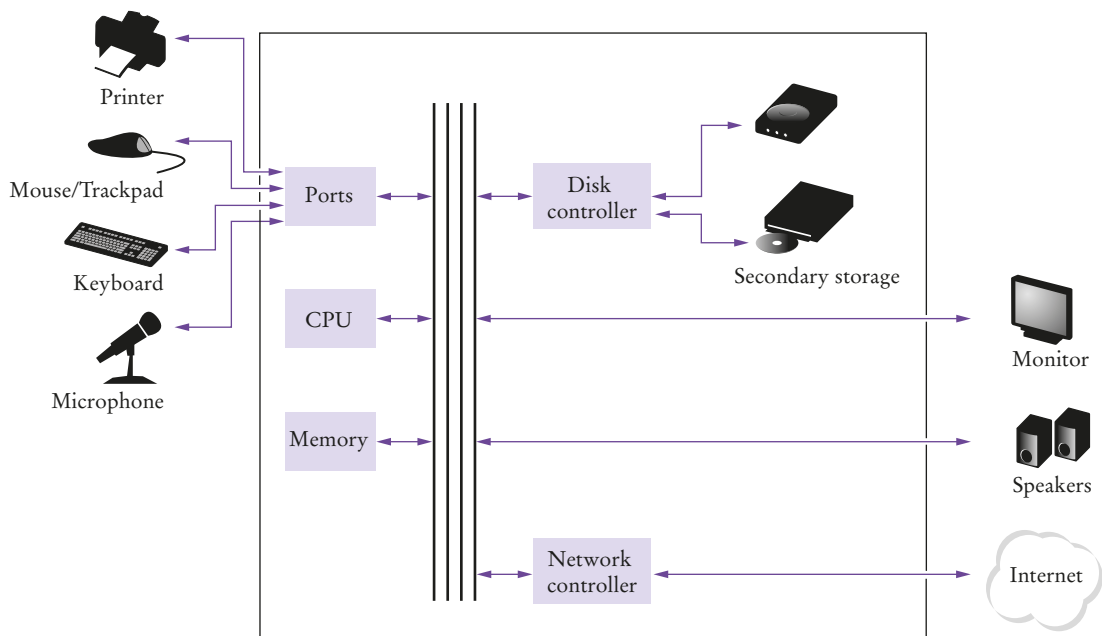


Figure 3 Schematic Design of a Personal Computer



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



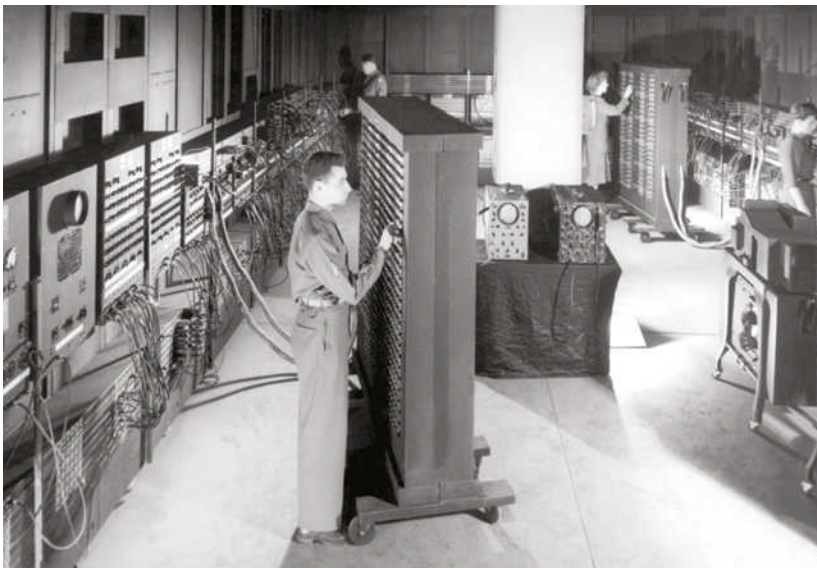
© Maurice Savage/Alamy Limited.

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.

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

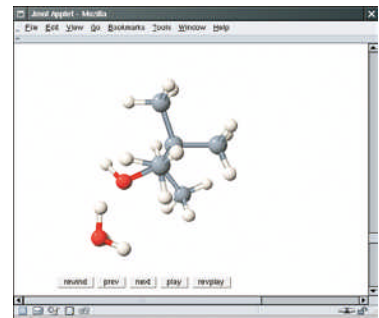
In order to write a computer program, you need to provide a sequence of instructions that the CPU can execute. A computer program consists of a large number of simple CPU instructions, and it is tedious and error-prone to specify them one by one. For that reason, **high-level programming languages** 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.

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.



An Applet for Visualizing Molecules

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?

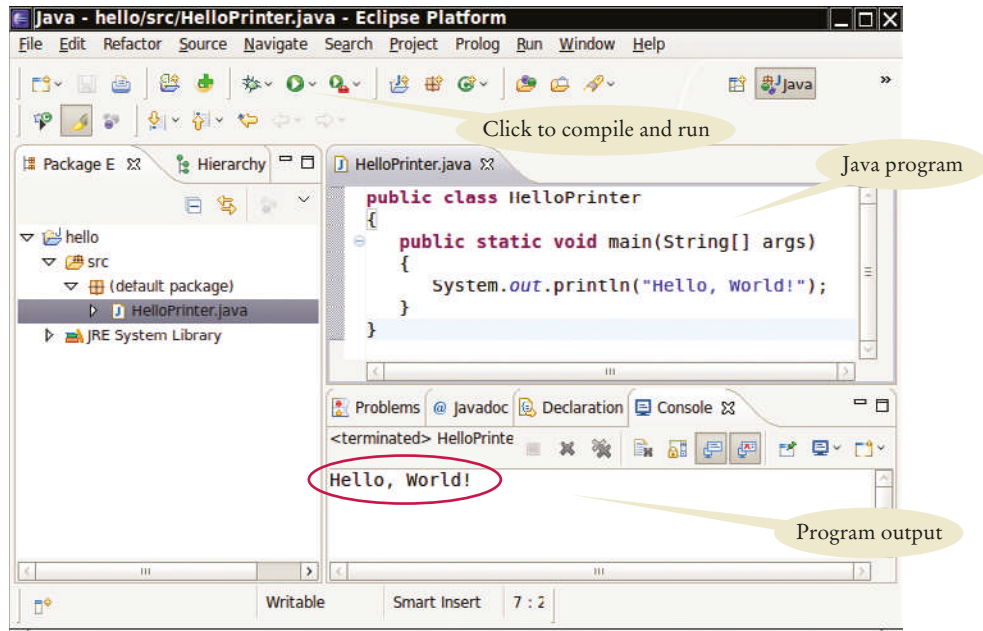
Practice It Now you can try this exercise at the end of the chapter: R1.5.

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.

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

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.

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

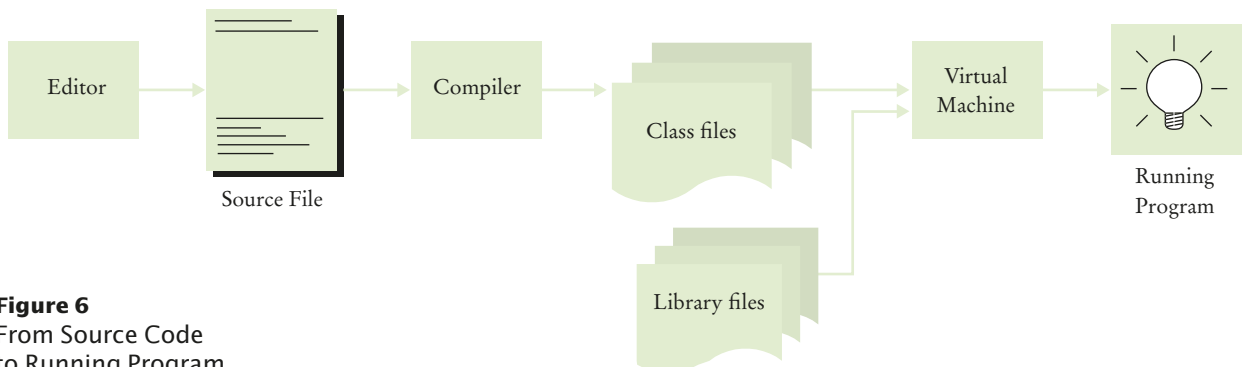


Figure 6
From Source Code
to Running Program