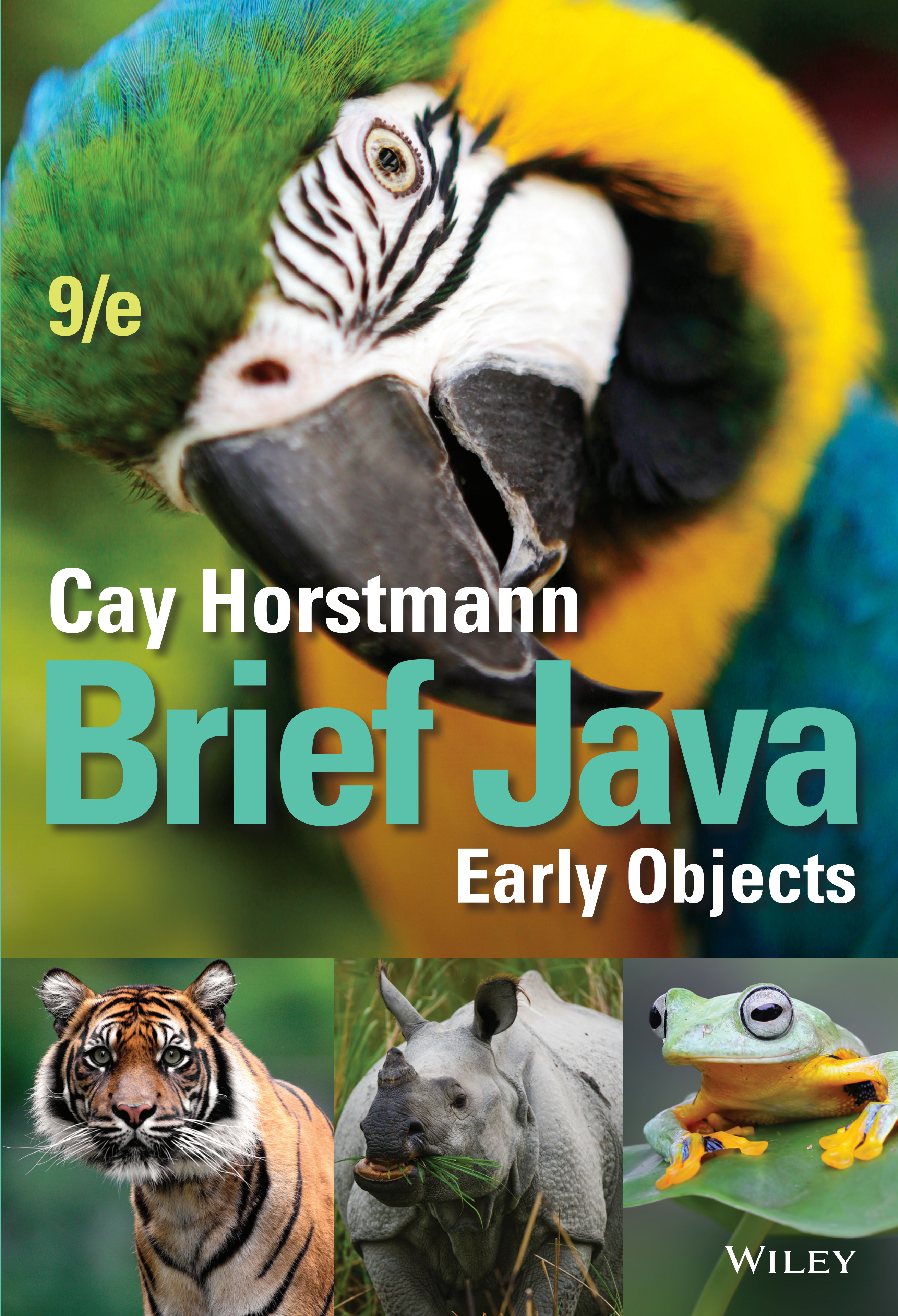


9/e

Cay Horstmann

Brief Java

Early Objects

WILEY

9/e

Brief Java

Early Objects

9/e

Brief Java

Early Objects

Cay Horstmann

San Jose State University

WILEY

VICE PRESIDENT AND EXECUTIVE PUBLISHER	Laurie Rosatone
EXECUTIVE EDITOR	Joanna Dingle
PROJECT MANAGER/DEVELOPMENT EDITOR	Cindy Johnson
EDITORIAL ASSISTANT	Crystal Franks
LEAD PRODUCT DESIGNER	Tom Kulesa
MARKETING MANAGER	Michael MacDougald
PRODUCTION MANAGER	Nichole Urban
PRODUCTION MANAGER	Nicole Repasky
PRODUCTION MANAGEMENT SERVICES	Cindy Johnson, Publishing Services
PHOTO EDITOR	Anindita Adiyal
COVER DESIGNER	Joanna Vieira
COVER PHOTOS	(tiger) © ArtMediaFactory/Shutterstock; (rhino) © GUDKOV ANDREY/Shutterstock; (bird) © Jeremy Woodhouse/Holly Wilmeth/Getty Images; (tree frog) © kuritafsheen/Getty Images.

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

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

This book is printed on acid-free paper. ∞

ePUB ISBN 978-1-119-49913-8

Printed in the United States of America.

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

10 9 8 7 6 5 4 3 2 1

PREFACE

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

Here are the key features of this book:

Start objects early, teach object orientation gradually.

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

Guidance and worked examples help students succeed.

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

Problem solving strategies are made explicit.

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

- Algorithm Design (with pseudocode)
- Tracing Objects
- First Do It By Hand (doing sample calculations by hand)
- Flowcharts
- Selecting Test Cases
- Hand-Tracing
- Storyboards
- Solve a Simpler Problem First
- Adapting Algorithms
- Discovering Algorithms by Manipulating Physical Objects
- Patterns for Object Data
- Thinking Recursively
- Estimating the Running Time of an Algorithm

Practice makes perfect.

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

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.

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

Adapted to Java Versions 8 Through 11

This edition takes advantage of modern Java features when they are pedagogically sensible. I continue to use "pure" interfaces with only abstract methods. Default, static, and private interface methods are introduced in a Special Topic. Lambda expressions are optional for user interface callback.

The "diamond" syntax for generic classes is introduced as a Special Topic in Chapter 7 and used systematically starting with Chapter 15. Local type inference with the `var` keyword is described in a Special Topic. Useful features such as the `try`-with-resources statement are integrated into the text.

Interactive Learning

With this edition, interactive content is front and center. Immersive activities integrate with this text and engage students in activities designed to foster in-depth learning. Students don't just watch animations and code traces, they work on generating them. Live code samples invite the reader to experiment and to learn programming constructs first hand. The activities provide instant feedback to show students what they did right and where they need to study more.

A Tour of the Book

The book can be naturally grouped into three 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.

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,

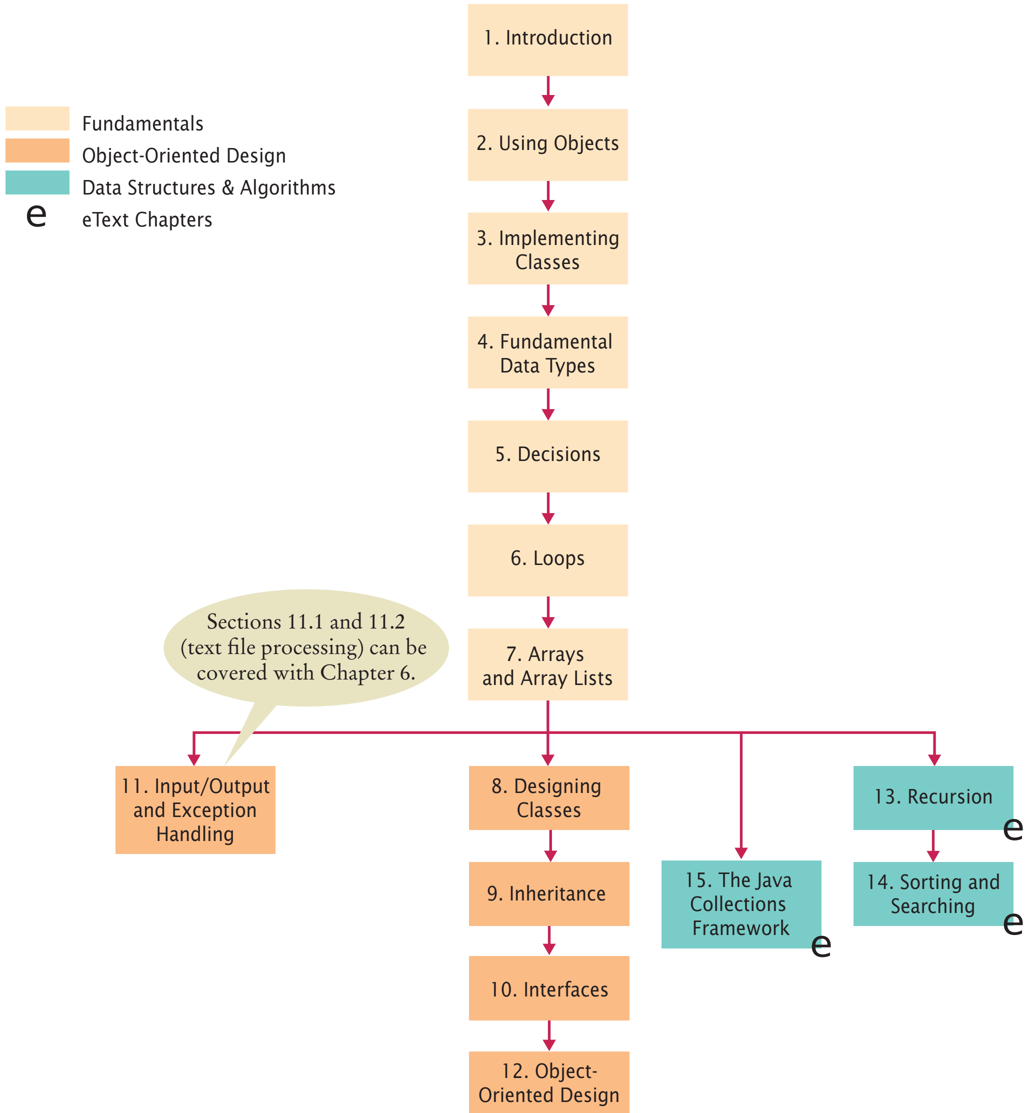


Figure 1
Chapter Dependencies

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. 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 13–15)

Chapters 13 through 15 (in the eText) contain an introduction to algorithms and data structures, covering recursion, sorting and searching, and the Java Collections Framework. 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.

Appendices

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

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

Interactive eText Designed for Programming Students

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

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

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

Web Resources

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

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

Walkthrough of the Learning Aids

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

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

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.

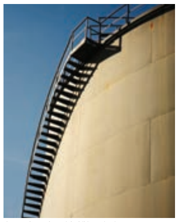
It often happens that you want to execute a sequence of statements a given number of times. You can use a while loop that is controlled by a counter, as in the following example:

```
int counter = 5; // Initialize the counter
while (counter <= 10) // Check the counter
{
   sum = sum + counter;
   counter++; // Update the counter
}
```

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

```
for (int counter = 5; counter <= 10; counter++)
{
   sum = sum + 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*.



© Enrico Fianchini/iStockphoto.

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

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

Syntax 6.2 for Statement

Syntax

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

These three expressions should be related. See Programming Tip 6.1.

This initialization happens once before the loop starts.

The condition is checked before each iteration.

This update is executed after each iteration.

```
for (int i = 5; i <= 10; i++)
{
   sum = sum + i;
}
```

The variable i is defined only in this for loop. See Special Topic 6.1.

This loop executes 6 times. See Programming Tip 6.3.

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

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

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

Memorable photos reinforce analogies and help students remember the concepts.

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

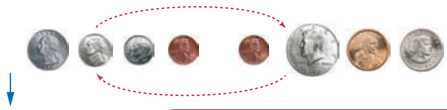
Problem Solving sections teach techniques for generating ideas and evaluating proposed solutions, often using pencil and paper or other artifacts. These sections emphasize that most of the planning and problem solving that makes students successful happens away from the computer.

7.5 Problem Solving: Discovering Algorithms by Manipulating Physical Objects 333

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

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

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



HOW TO 6.1

Writing a Loop

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

Problem Statement Read twelve temperature values (one for each month) and display the number of the month with the highest temperature. For example, according to `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.

© Stevegeer/iStockphoto.

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

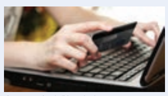
Every loop needs to do some kind of repetitive work, such as

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. See your eText or visit `wiley.com/go/bjeo7`.



© MorePixels/iStockphoto.

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

Table 1 Variable Declarations in Java

Variable Name	Comment
`int width = 20;`	Declares an integer variable and initializes it with 20.
`int perimeter = 4 * width;`	The initial value need not be a fixed value. (Of course, `width` must have been previously declared.)
`String greeting = "Hi!";`	This variable has the type `String` and is initialized with the string "Hi".
🚫 `height = 30;`	**Error:** The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.2.5.
🚫 `int width = "20";`	**Error:** You cannot initialize a number with the string "20". (Note the quotation marks.)
`int width;`	Declares an integer variable without initializing it. This can be a cause for errors—see Common Error 2.1.
`int width, height;`	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 Execution of a for Loop

1. Initialize counter — counter = 5

```
for (int counter = 5; counter <= 10; counter++)
{
   sum = sum + counter;
}
```

2. Check condition — counter = 5

```
for (int counter = 5; counter <= 10; counter++)
{
   sum = sum + counter;
}
```

3. Execute loop body — counter = 5

```
for (int counter = 5; counter <= 10; counter++)
{
   sum = sum + counter;
}
```

4. Update counter — counter = 6

```
for (int counter = 5; counter <= 10; counter++)
{
   sum = sum + counter;
}
```

5. Check condition again — counter = 6

```
for (int counter = 5; counter <= 10; counter++)
{
   sum = sum + counter;
}
```

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

sec01/ElevatorSimulation.java

```
import java.util.Scanner;

/**
   This program simulates an elevator panel that skips the 13th floor.
*/
public class ElevatorSimulation
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      System.out.print("Floor: ");
      int floor = in.nextInt();

      // Adjust floor if necessary

      int actualFloor;
      if (floor > 13)
```

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

The following program puts the if statement to work. This program asks for the desired floor and then prints out the actual floor.

sec01/ElevatorSimulation.java

```
import java.util.Scanner;

/**
   This program simulates an elevator panel that skips the 13th floor.
*/
public class ElevatorSimulation
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      System.out.print("Floor: ");
      int floor = in.nextInt();

      // Adjust floor if necessary

      int actualFloor;
      if (floor > 13)
      {
         actualFloor = floor - 1;
      }
      else
      {
         actualFloor = floor;
      }

      System.out.println("The elevator will travel to the actual floor "
         + actualFloor);
   }
}
```

Input

1 20

Run Reset

SELF CHECK

•• 8. Walk through the following code that swaps two elements in an array.

Press start to begin.

Start

```
int i = 1;
int j = 2;
// Good swap
double temp = a[i];
a[i] = a[j];
a[j] = temp;
// Bad swap
a[i] = a[j];
a[j] = a[i];
```

i	j	a[0]	a[1]	a[2]	a[3]	temp

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

•• **Business E6.17** *Currency conversion.* Write a program that first asks the user to type today's price for one dollar in Japanese yen, then reads U.S. dollar values and converts each to yen. Use 0 as a sentinel.

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

Technetium-99 is a radioisotope that is used in imaging of the brain. It has a half-life of 6 hours. Your program should display the relative amount A / A_0 in a patient body every hour for 24 hours after receiving a dose.

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

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.

© thomasd007/iStockphoto.

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

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
```

income	status	tax1	tax2
80000	MARRIED	0	0

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

```
34    if (status == SINGLE)
35    {
36       if (income <= RATE1_SINGLE_LIMIT)
37       {
38          tax1 = RATE1 * income;
39       }
40       else
41       {
```

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

Special Topic 11.2

File Dialog Boxes

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

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

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

Here is a complete example:

```
JFileChooser chooser = new JFileChooser();
Scanner in = null;
if (chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION)
{
   File selectedFile = chooser.getSelectedFile();
   in = new Scanner(selectedFile);
   . . .
}
```

Additional **full code examples** throughout the text provide complete programs for students to run and modify.

EXAMPLE CODE See special_topic_2 of your eText or companion code for a program that demonstrates how to use a file chooser.

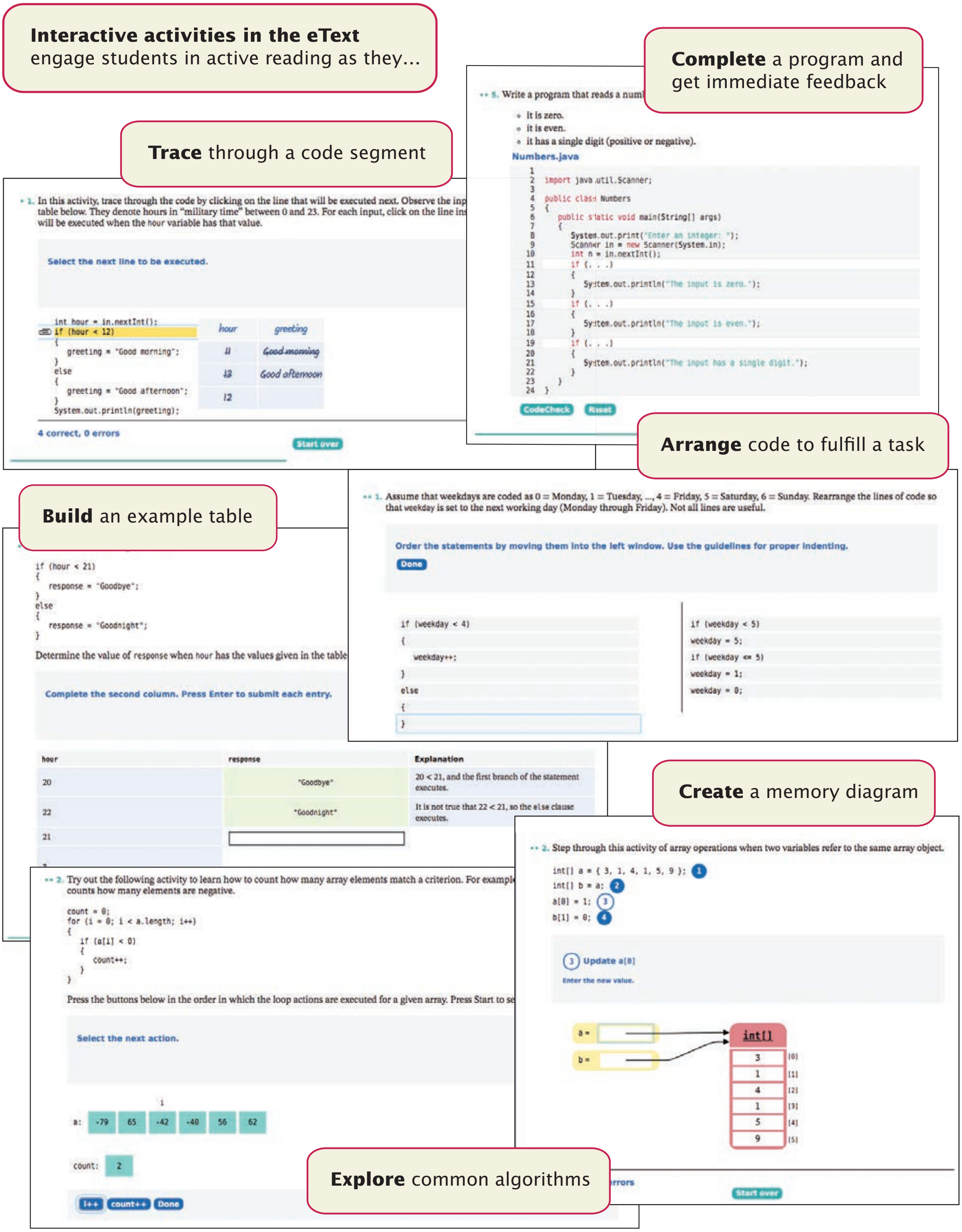
Computing & Society presents social and historical topics on computing—for interest and to fulfill the "historical and social context" requirements of the ACM/IEEE curriculum guidelines.

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


Interactive activities in the eText
engage students in active reading as they…
Trace through a code segment
1. In this activity, trace through the code by clicking on the line that will be executed next. Observe the inp
table below. They denote hours in "military time" between 0 and 23. For each input, click on the line ins
will be executed when the hour variable has that value.
Select the next line to be executed.
int hour = in.nextInt();
if (hour < 12)
greeting = "Good morning";
else
greeting = "Good afternoon";
System.out.println(greeting);
hour
greeting
11
Good morning
13
Good afternoon
12
4 correct, 0 errors
Start over
Complete a program and get immediate feedback
5. Write a program that reads a num
it is zero.
it is even.
it has a single digit (positive or negative).
Numbers.java
import java.util.Scanner;
public class Numbers
public static void main(String[] args)
System.out.print("Enter an integer: ");
Scanner in = new Scanner(System.in);
int n = in.nextInt();
if (. . .)
System.out.println("The input is zero.");
if (. . .)
System.out.println("The input is even.");
if (. . .)
System.out.println("The input has a single digit.");
CodeCheck
Reset
Arrange code to fulfill a task
1. 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 moving them into the left window. Use the guidelines for proper indenting.
Done
if (weekday < 4)
weekday++;
else
if (weekday < 5)
weekday = 5;
if (weekday <= 5)
weekday = 1;
weekday = 0;
Build an example table
if (hour < 21)
response = "Goodbye";
else
response = "Goodnight";
Determine the value of response when hour has the values given in the table
Complete the second column. Press Enter to submit each entry.
hour
response
Explanation
20
"Goodbye"
20 < 21, and the first branch of the statement executes.
22
"Goodnight"
It is not true that 22 < 21, so the else clause executes.
21
Create a memory diagram
2. Step through this activity of array operations when two variables refer to the same array object.
int[] a = { 3, 1, 4, 1, 5, 9 };
int[] b = a;
a[0] = 1;
b[1] = 0;
Update a[0]
Enter the new value.
int[]
Start over
2. Try out the following activity to learn how to count how many array elements match a criterion. For exampl
counts how many elements are negative.
count = 0;
for (i = 0; i < a.length; i++)
if (a[i] < 0)
count++;
Press the buttons below in the order in which the loop actions are executed for a given array. Press Start to se
Select the next action.
-79
65
-42
-40
56
62
count:
2
i++
count++
Done
Explore common algorithms

Acknowledgments

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

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

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

Radhouane Chouchane, *Columbus State University*
Sussan Einakian, *California Polytechnic State University*
Jon Hanrath, *Illinois Institute of Technology*
Brian King, *Bucknell University*
Kathleen O'Brien, *San Jose State University*
Eman Saleh, *University of Georgia*
William Wei, *New York Institute of Technology*

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

Eric Aaron, *Wesleyan University*
James Agnew, *Anne Arundel Community College*
Tim Andersen, *Boise State University*
Ivan Bajic, *San Diego State University*
Greg Ballinger, *Miami Dade College*
Ted Bangay, *Sheridan Institute of Technology*
Ian Barland, *Radford University*
George Basham, *Franklin University*
Jon Beck, *Truman State University*
Sambit Bhattacharya, *Fayetteville State University*
Rick Birney, *Arizona State University*
Paul Bladek, *Edmonds Community College*
Matt Boutell, *Rose-Hulman Institute of Technology*
Joseph Bowbeer, *Vizrea Corporation*
Timothy A. Budd, *Oregon State University*
John Bundy, *DeVry University Chicago*
Robert P. Burton, *Brigham Young University*
Frank Butt, *IBM*
Jerry Cain, *Stanford University*
Adam Cannon, *Columbia University*
Michael Carney, *Finger Lakes Community College*
Robin Carr, *Drexel University*
Christopher Cassa, *Massachusetts Institute of Technology*
Nancy Chase, *Gonzaga University*
Dr. Suchindran S. Chatterjee, *Arizona State University*
Archana Chidanandan, *Rose-Hulman Institute of Technology*
Vincent Cicirello, *The Richard Stockton College of New Jersey*
Gerald Cohen, *The Richard Stockton College of New Jersey*
Teresa Cole, *Boise State University*
Deborah Coleman, *Rochester Institute of Technology*
Tina Comston, *Franklin University*
Lennie Cooper, *Miami Dade College*
Jose Cordova, *University of Louisiana, Monroe*
Valentino Crespi, *California State University, Los Angeles*

Jim Cross, *Auburn University*
Russell Deaton, *University of Arkansas*
Geoffrey Decker, *Northern Illinois University*
Suzanne Dietrich, *Arizona State University, West Campus*
Mike Domaratzki, *University of Manitoba*
H. E. Dunsmore, *Purdue University*
Robert Duvall, *Duke University*
Sherif Elfayoumy, *University of North Florida*
Eman El-Sheikh, *University of West Florida*
Henry A. Etlinger, *Rochester Institute of Technology*
John Fendrich, *Bradley University*
David Freer, *Miami Dade College*
John Fulton, *Franklin University*
David Geary, *Sabreware, Inc.*
Margaret Geroch, *Wheeling Jesuit University*
Ahmad Ghafarian, *North Georgia College & State University*
Rick Giles, *Acadia University*
Stacey Grasso, *College of San Mateo*
Jianchao Han, *California State University, Dominguez Hills*
Lisa Hansen, *Western New England College*
Elliotte Harold
Eileen Head, *Binghamton University*
Cecily Heiner, *University of Utah*
Guy Helmer, *Iowa State University*
Ed Holden, *Rochester Institute of Technology*
Brian Howard, *Depauw University*
Lubomir Ivanov, *Iona College*
Norman Jacobson, *University of California, Irvine*
Steven Janke, *Colorado College*
Curt Jones, *Bloomsburg University*
Mark Jones, *Lock Haven University of Pennsylvania*
Dr. Mustafa Kamal, *University of Central Missouri*
Aaron Keen, *California Polytechnic State University, San Luis Obispo*
Mugdha Khaladkar, *New Jersey Institute of Technology*
Gary J. Koehler, *University of Florida*
Elliot Koffman, *Temple University*
Ronald Krawitz, *DeVry University*
Norm Krumpe, *Miami University Ohio*
Jim Leone, *Rochester Institute of Technology*
Kevin Lillis, *St. Ambrose University*
Darren Lim, *Siena College*
Hong Lin, *DeVry University*
Kathy Liszka, *University of Akron*
Hunter Lloyd, *Montana State University*
Youmin Lu, *Bloomsburg University*
Peter Lutz, *Rochester Institute of Technology*
Kuber Maharjan, *Purdue University College of Technology at Columbus*
John S. Mallozzi, *Iona College*
John Martin, *North Dakota State University*
Jeanna Matthews, *Clarkson University*
Patricia McDermott-Wells, *Florida International University*
Scott McElfresh, *Carnegie Mellon University*
Joan McGrory, *Christian Brothers University*
Carolyn Miller, *North Carolina State University*
Sandeep R. Mitra, *State University of New York, Brockport*
Teng Moh, *San Jose State University*
Bill Mongan, *Drexel University*
John Moore, *The Citadel*
Jose-Arturo Mora-Soto, Jesica Rivero-Espinosa, and Julio-Angel Cano-Romero, *University of Madrid*
Faye Navabi, *Arizona State University*
Parviz Partow-Navid, *California State University, Los Angeles*
George Novacky, *University of Pittsburgh*

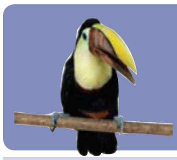
Kevin O'Gorman, *California Polytechnic State University, San Luis Obispo*
Michael Olan, *Richard Stockton College*
Mimi Opkins, *California State University Long Beach*
Derek Pao, *City University of Hong Kong*
Kevin Parker, *Idaho State University*
Jim Perry, *Ulster County Community College*
Cornel Pokorny, *California Polytechnic State University, San Luis Obispo*
Roger Priebe, *University of Texas, Austin*
C. Robert Putnam, *California State University, Northridge*
Kai Qian, *Southern Polytechnic State University*
Cyndi Rader, *Colorado School of Mines*
Neil Rankin, *Worcester Polytechnic Institute*
Brad Rippe, *Fullerton College*
Pedro I. Rivera Vega, *University of Puerto Rico, Mayaguez*
Daniel Rogers, *SUNY Brockport*
Chaman Lal Sabharwal, *Missouri University of Science and Technology*
Katherine Salch, *Illinois Central College*
John Santore, *Bridgewater State College*
Javad Shakib, *DeVry University*
Carolyn Schauble, *Colorado State University*
Brent Seales, *University of Kentucky*
Christian Shin, *SUNY Geneseo*
Charlie Shu, *Franklin University*
Jeffrey Six, *University of Delaware*
Don Slater, *Carnegie Mellon University*
Ken Slonneger, *University of Iowa*
Aurelia Smith, *Columbus State University*
Donald Smith, *Columbia College*
Joslyn A. Smith, *Florida International University*
Stephanie Smullen, *University of Tennessee, Chattanooga*
Robert Strader, *Stephen F. Austin State University*
Monica Sweat, *Georgia Institute of Technology*
Peter Stanchev, *Kettering University*
Aakash Taneja, *The Richard Stockton College of New Jersey*
Craig Tanis, *University of Tennessee at Chattanooga*
Shannon Tauro, *University of California, Irvine*
Ron Taylor, *Wright State University*
Russell Tessier, *University of Massachusetts, Amherst*
Jonathan L. Tolstedt, *North Dakota State University*
David Vineyard, *Kettering University*
Joseph Vybihal, *McGill University*
Xiaoming Wei, *Iona College*
Jonathan S. Weissman, *Finger Lakes Community College*
Todd Whittaker, *Franklin University*
Robert Willhoft, *Roberts Wesleyan College*
Brent Wilson, *George Fox University*
Katherine Winters, *University of Tennessee at Chattanooga*
Lea Wittie, *Bucknell University*
David Womack, *University of Texas at San Antonio*
David Woolbright, *Columbus State University*
Tom Wulf, *University of Cincinnati*
Catherine Wyman, *DeVry University*
Arthur Yanushka, *Christian Brothers University*
Qi Yu, *Rochester Institute of Technology*
Salih Yurttas, *Texas A&M University*

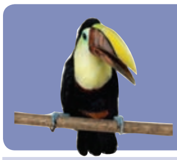
CONTENTS

*See your eText or visit www.wiley.com/go/bjeo7.

*See your eText or visit www.wiley.com/go/bjeo7.

ALPHABETICAL LIST OF SYNTAX BOXES





*See your eText or visit www.wiley.com/go/bjeo7.

*See your eText or visit `www.wiley.com/go/bjeo7`.

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

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.

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.

A computer program is a sequence of instructions and decisions.

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.

Programming is the act of designing and implementing computer programs.

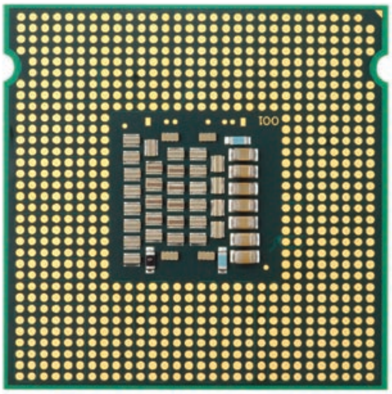
The act of designing and implementing computer programs is called **programming**. In this book, you will learn how to program a computer—that is, how to direct the computer to execute tasks.

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.

1.2 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*.



© Amorphis/iStockphoto.

Figure 1 Central Processing Unit

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

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.

Storage devices include memory and secondary storage.

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

© PhotoDisc, Inc./Getty Images, Inc.

Figure 2 A Hard Disk

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

To interact with a human user, a computer requires peripheral devices. The 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.

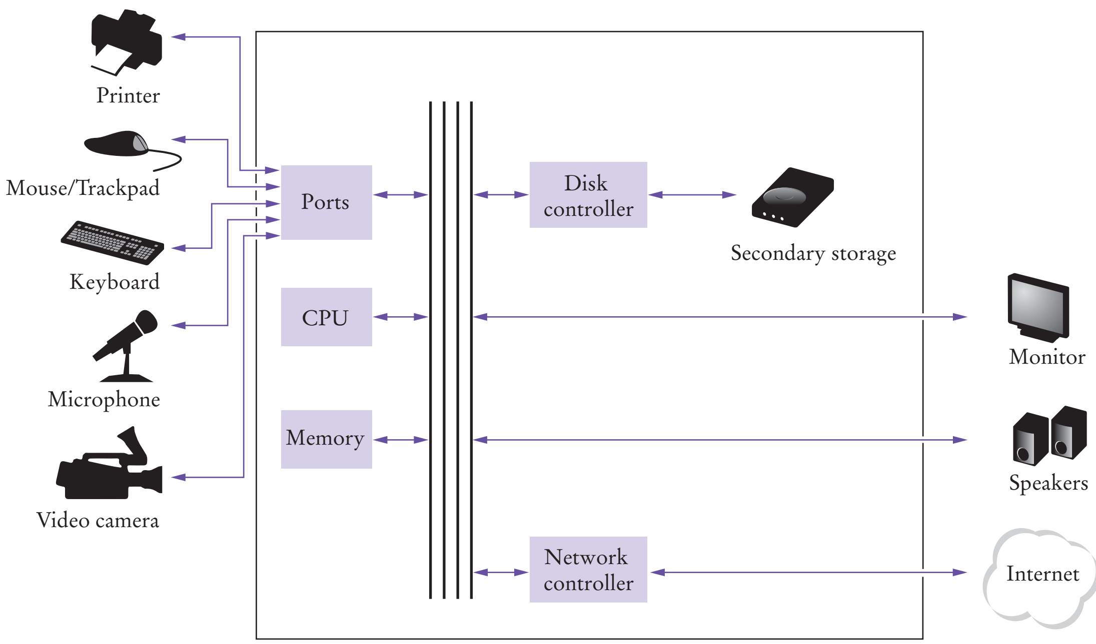


Figure 3 Schematic Design of a Personal Computer

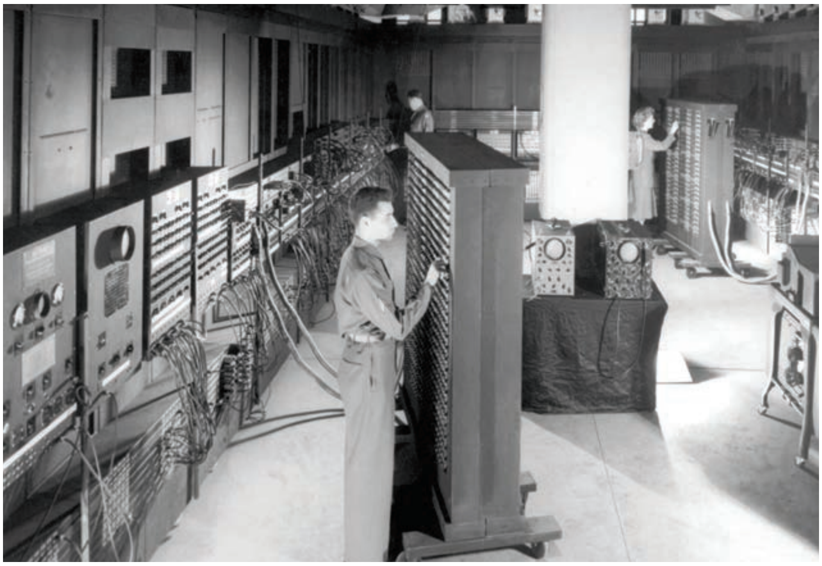
Computing & Society 1.1 Computers Are Everywhere

When computers were first invented in the 1940s, a computer filled an entire room. The photo below shows the ENIAC (*e*lectronic *n*umerical *i*ntegrator *a*nd *c*omputer), completed in 1946 at the University of Pennsylvania. The ENIAC was used by the military to compute the trajectories of projectiles. Nowadays, computing facilities of search engines, Internet shops, and social networks fill huge buildings called data centers. At the other end of the spectrum, computers are all around us. Your cell phone has a computer inside, as do many credit cards and fare cards for public transit. A modern car 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 been written without computers.

© Maurice Savage/Alamy Limited.

This transit card contains a computer.



© UPPA/Photoshot.

The ENIAC

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.

1.3 The Java Programming Language

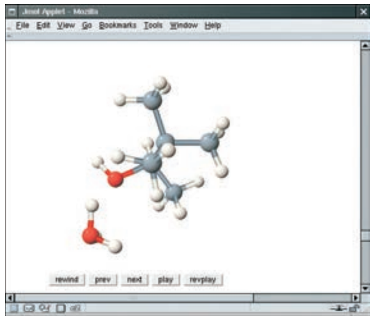
In order to write a computer program, you need to provide a sequence of instructions that the CPU can execute. A computer program consists of a large number of simple CPU instructions, and it is tedious and error-prone to specify them one by one. For that reason, **high-level programming languages**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

James Gosling

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

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



An Applet for Visualizing Molecules

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

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	6	2006	Library improvements
1.2	1998	Swing, Collections framework	7	2011	Small language changes and library improvements
1.3	2000	Performance enhancements	8	2014	Function expressions, streams, new date/time library
1.4	2002	Assertions, XML support	9	2017	Modules
5	2004	Generic classes, enhanced `for` loop, auto-boxing, enumerations, annotations	10, 11	2018	Versions with incremental improvements are released every six months

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.

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.

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

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 8 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.

1.4 Becoming Familiar with Your Programming Environment

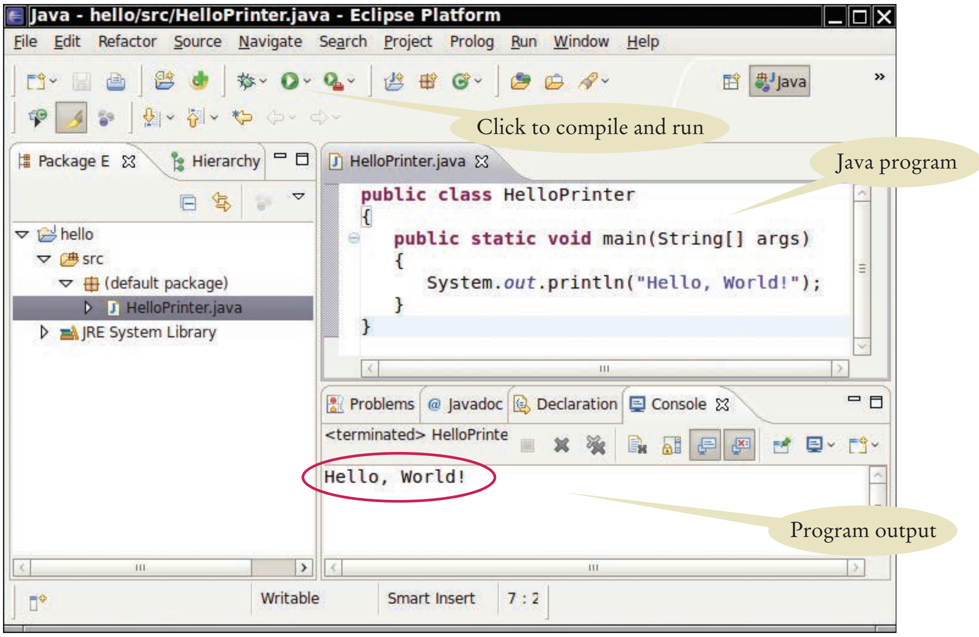
Set aside time to become familiar with the programming environment that you will use for your class work.

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.

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.

Figure 4 Running the `HelloPrinter` Program in an Integrated Development Environment



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

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.

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

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

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

Step 3 Run the program.

The process for running a program depends greatly on your programming 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 Figure 4 and Figure 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

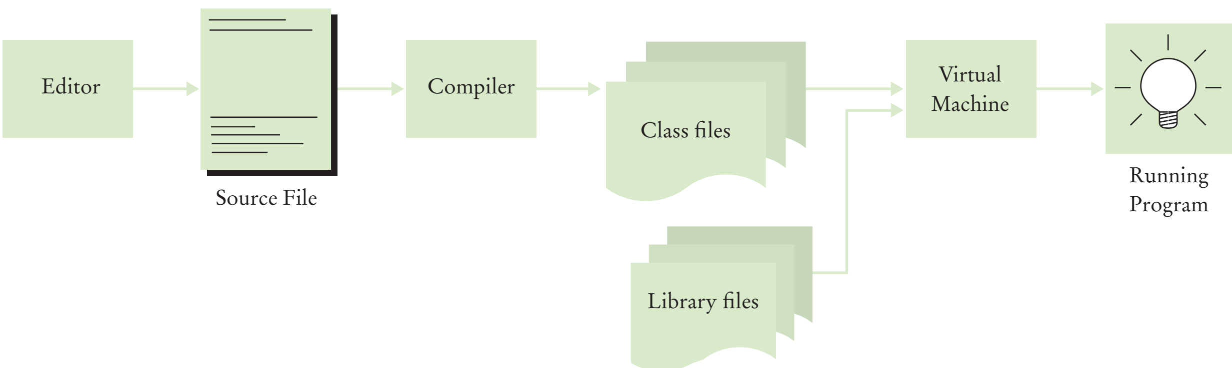


Figure 6 From Source Code to Running Program