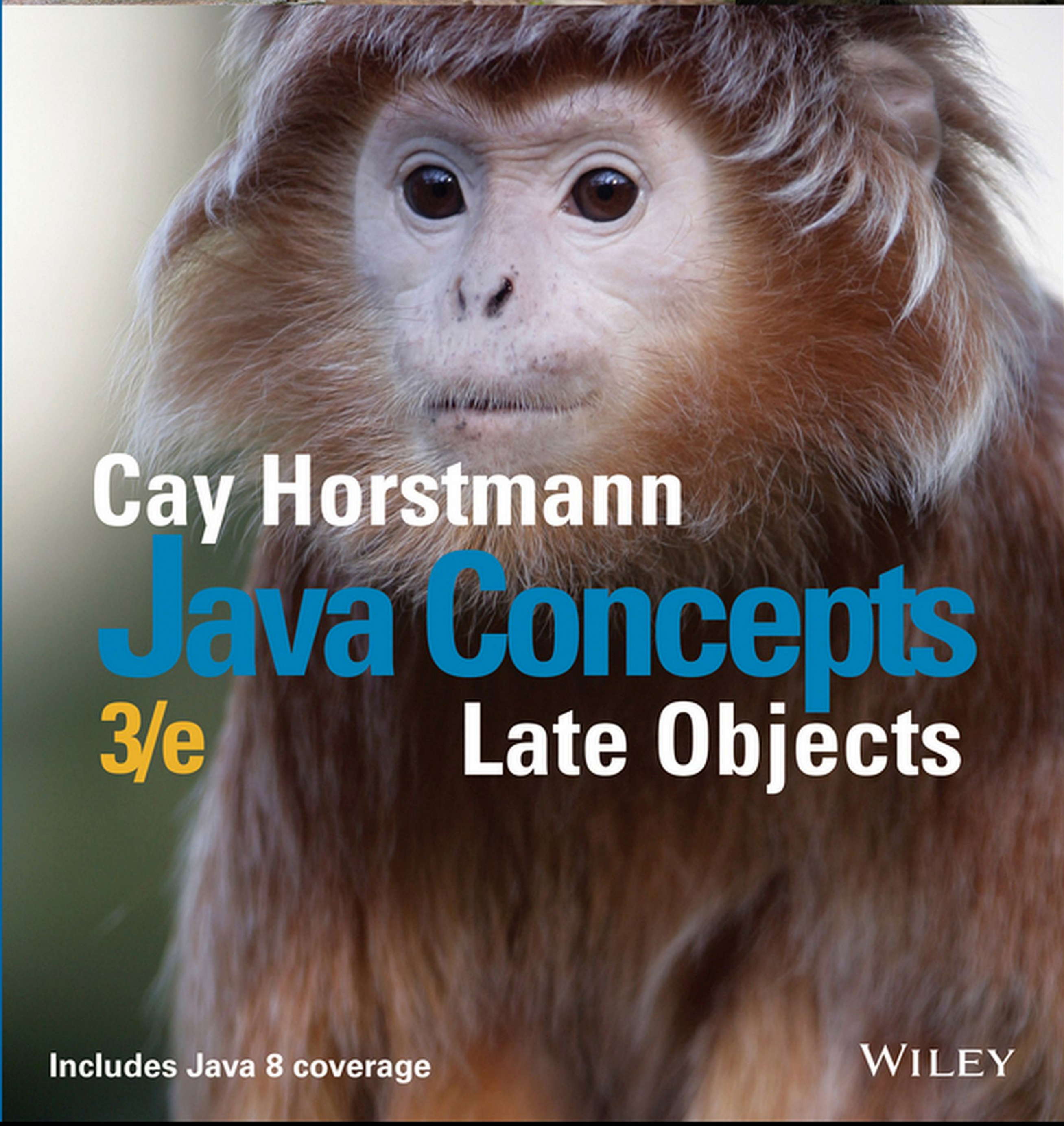


Cay Horstmann

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	$\lvert x \rvert$
`Math.sin(x)`, `Math.cos(x)`, `Math.tan(x)`	Sine, cosine, tangent of x (x in radians)	

Selected Operators and Their Precedence

(See Appendix B for the complete list.)

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

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

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);
}
```

Loop body executed at least once

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

Set to a new element in each iteration

An array or collection

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

Executed for each element

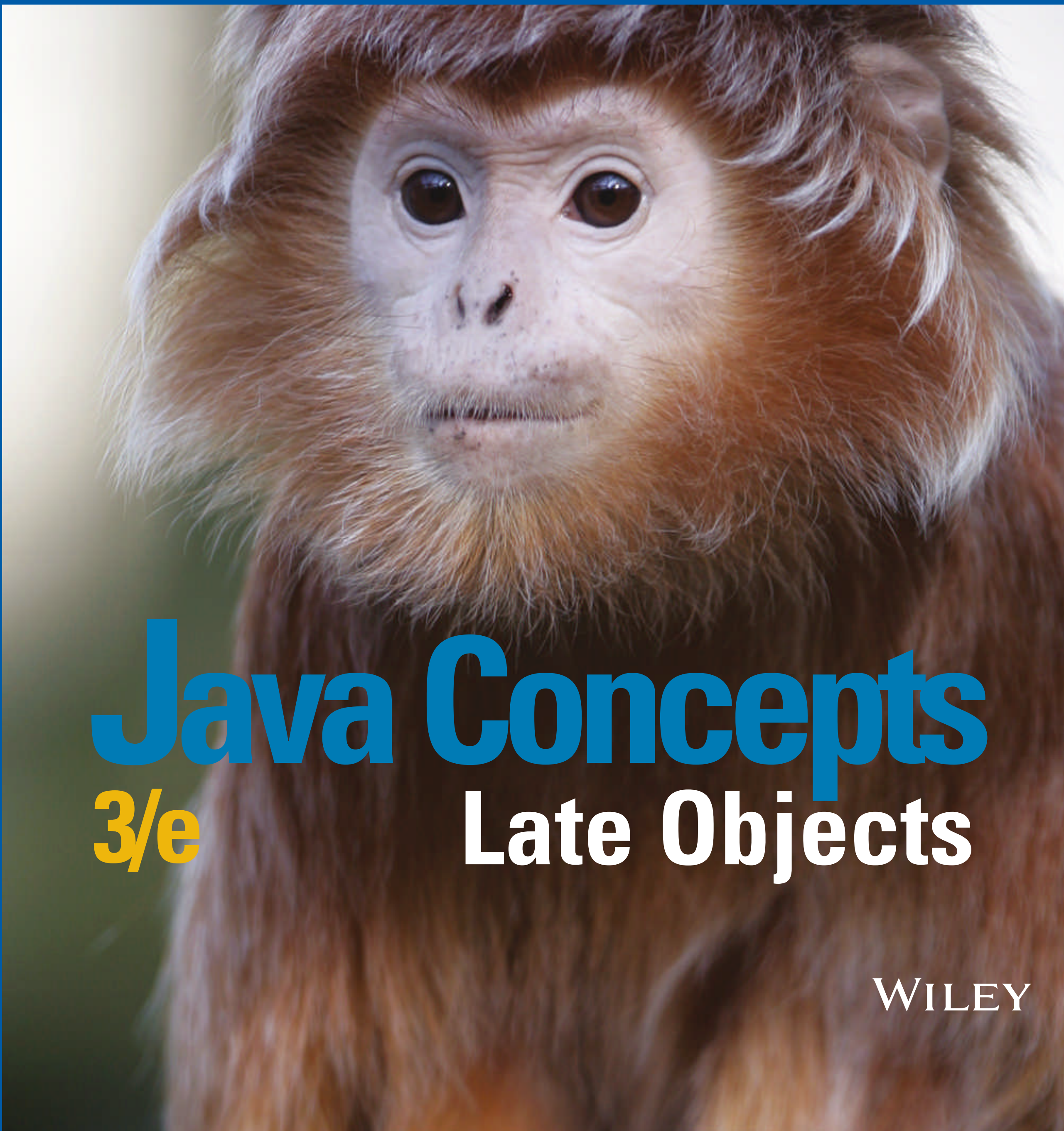
Java Concepts

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.

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.

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:

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.

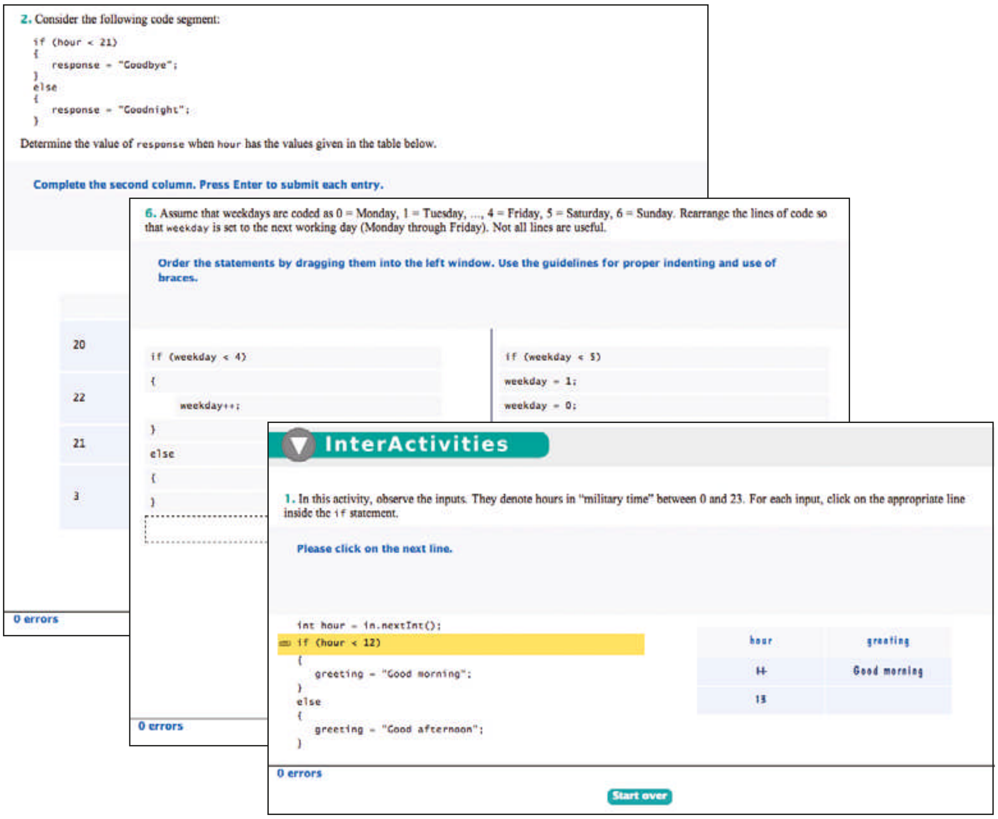
In addition, Java 7 features such as the `try`-with-resources statement are now integrated into the text.

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`.



"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

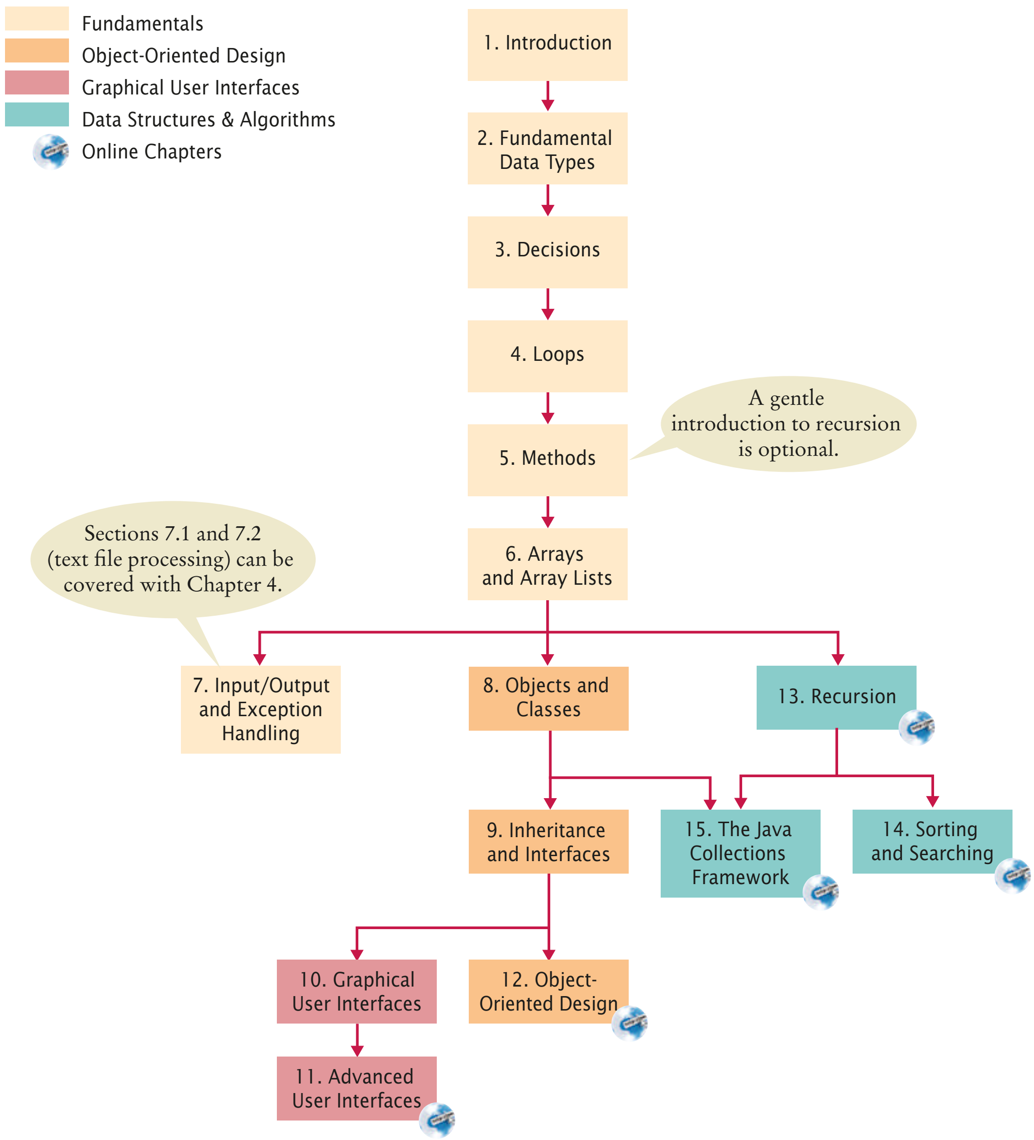


Figure 1 Chapter Dependencies

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.

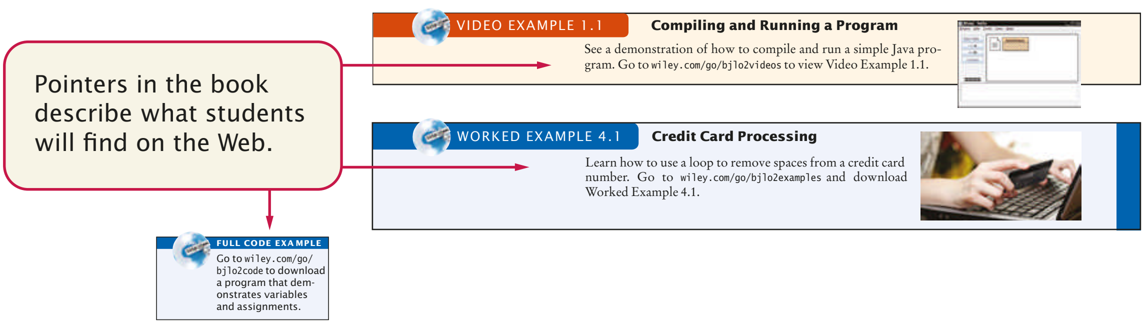
The interactive edition of *Java Concepts* adds even more value by integrating a wealth of interactive exercises into the eBook. See http://wiley.com/go/bjlo2interactivities to find out more about this new format.

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

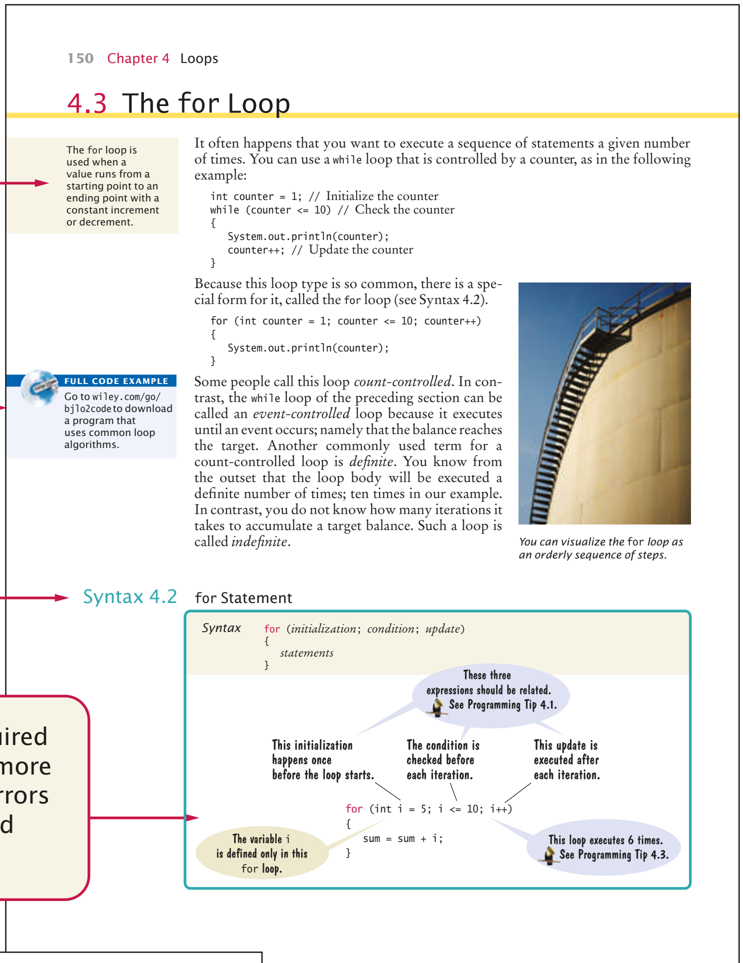
The pedagogical elements in this book work together to focus on and reinforce key concepts and fundamental principles of programming, with additional tips and detail organized to support and deepen these fundamentals. In addition to traditional features, such as chapter objectives and a wealth of exercises, 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);
}
```

FULL CODE EXAMPLE
Go to wiley.com/go/bjlo2code to download a program that uses common loop algorithms.

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
}
```

These three expressions should be related. See Programming Tip 4.1.

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;
}
```

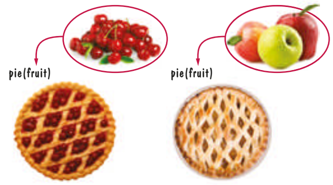
The variable i is defined only in this for loop.

This loop executes 6 times. See Programming Tip 4.3.

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

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

Memorable photos reinforce analogies and help students remember the concepts.



A recipe for a fruit pie may say to use any kind of fruit. Here, "fruit" is an example of a parameter variable. Apples and cherries are examples of arguments.

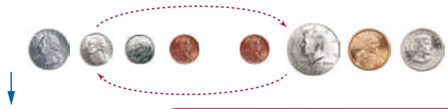
Problem Solving sections teach techniques for generating ideas and evaluating proposed solutions, often using pencil and paper or other artifacts. These sections emphasize that most of the planning and problem solving that makes students successful happens away from the computer.

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?

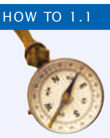
Let's put the first coin into place, by swapping it with the fifth coin. However, as Java programmers, we will say that we swap the coins in positions 0 and 4:

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



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 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.

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.

Table 1 Variable Declarations in Java

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

Figure 3 Execution of a for Loop

1 Initialize counter

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

counter = 1

2 Check condition

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

counter = 1

3 Execute loop body

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

counter = 1

4 Update counter

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

counter = 2

5 Check condition again

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

counter = 2

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

1 Method call

```
double result1 = cubeVolume(2);
```

result1 =

sideLength =

2 Initializing method parameter variable

```
double result1 = cubeVolume(2);
```

result1 =

sideLength = 2

3 About to return to the caller

```
double volume = sideLength * sideLength * sideLength;
return volume;
```

result1 =

sideLength = 2

volume = 8

4 After method call

```
double result1 = cubeVolume(2);
```

result1 = 8

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

```
/**
   This program computes the time required to double an investment.
*/
public class DoubleInvestment
{
   public static void main(String[] args)
   {
      final double RATE = 5;
      final double INITIAL_BALANCE = 10000;
      final double TARGET = 2 * INITIAL_BALANCE;

      double balance = INITIAL_BALANCE;
      int year = 0;

      // Count the years required for the investment to double

      while (balance < TARGET)
      {
         year++;
         double interest = balance * RATE / 100;
         balance = balance + interest;
      }

      System.out.println("The investment doubled after "
         + year + " years.");
   }
}
```

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.

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       {
42          tax1 = RATE1 * RATE1_SINGLE_LIMIT;
43          tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
```

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.

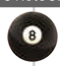
The `JFileChooser` class has many options to fine-tune the display of the dialog box, but in its most basic form it is quite simple: Construct a file chooser object; then call the `showOpenDialog` or `showSaveDialog` method. Both methods show the same dialog box, but the button for selecting a file is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

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

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*

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

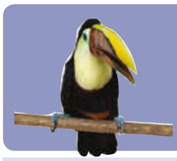
ALPHABETICAL LIST OF SYNTAX BOXES

Available online at `www.wiley.com/college/horstmann`.



Available online at `www.wiley.com/college/horstmann`.

Available online at `www.wiley.com/college/horstmann`.



Available online at `www.wiley.com/college/horstmann`.

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

Available online at `www.wiley.com/college/horstmann`.

Available online at www.wiley.com/college/horstmann.

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.

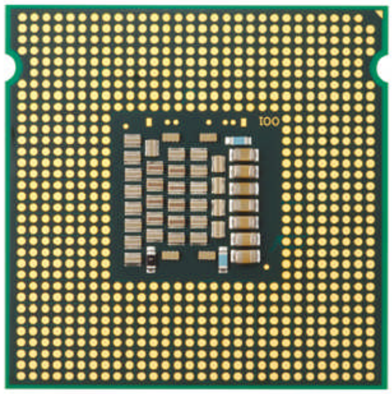
SELF CHECK

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*.



© 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)

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.

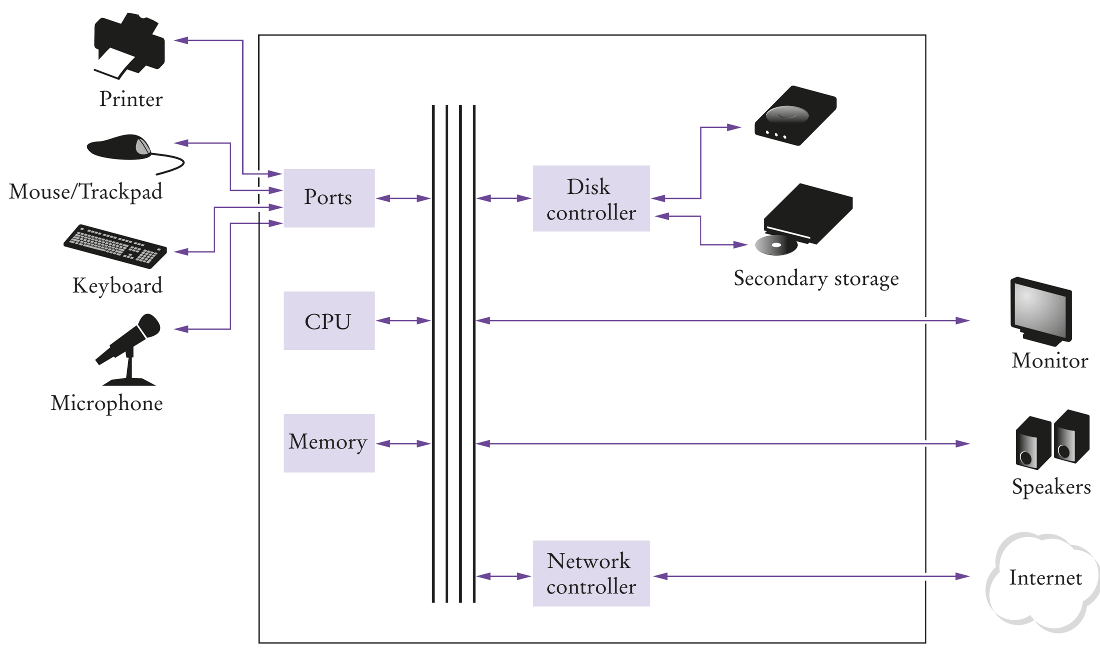


Figure 3 Schematic Design of a Personal Computer

SELF CHECK

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

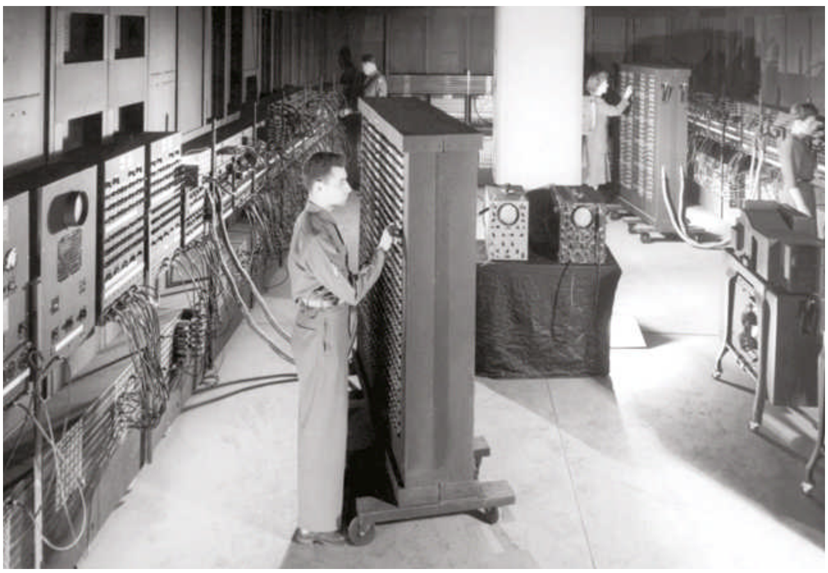
When computers were first invented in the 1940s, a computer filled an entire room. The photo below shows the ENIAC (*e*lectronic *n*umerical *i*ntegrator *a*nd *c*omputer), completed in 1946 at the University of Pennsylvania. The ENIAC was used by the military to compute the trajectories of projectiles. 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.

This transit card contains a computer.

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.

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

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



The ENIAC

1.3 The Java Programming Language

James Sullivan/Getty Images.

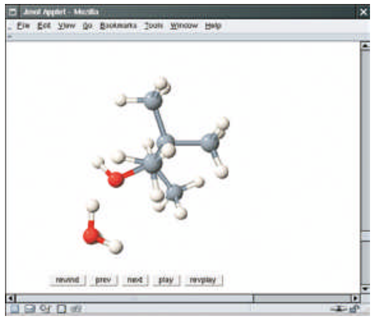
James Gosling

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.

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.

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.

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.

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

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

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

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

SELF CHECK

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.

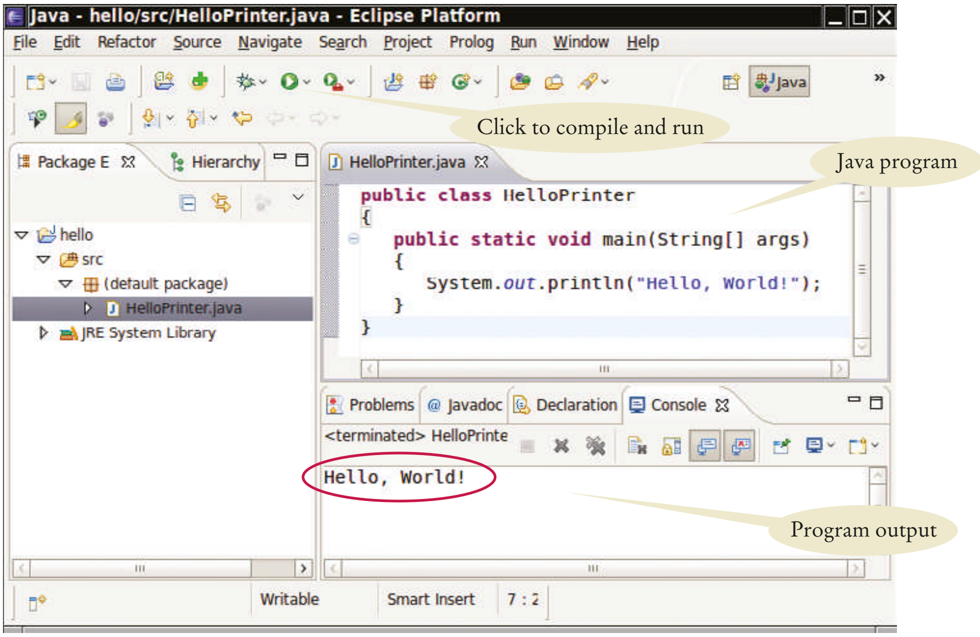


Figure 4 Running the `HelloPrinter` Program in an Integrated Development Environment

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.

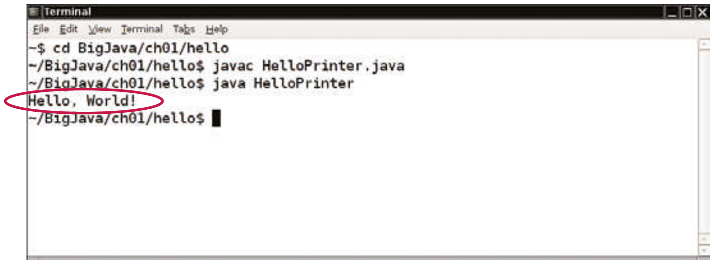


Figure 5
Running the `HelloPrinter` Program in a Console Window

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

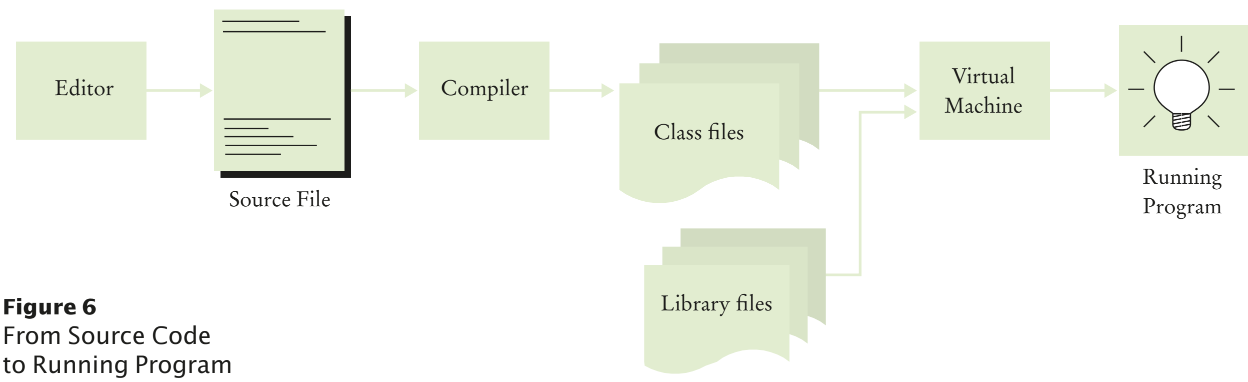


Figure 6
From Source Code to Running Program