Introduction to

# JAVA™

# PROGRAMMING AND DATA STRUCTURES

COMPREHENSIVE VERSION

## Y. DANIEL LIANG

Pearson

11th Edition

# INTRODUCTION TO JAVA™

# PROGRAMMING AND DATA STRUCTURES

## COMPREHENSIVE VERSION

Eleventh Edition

**Y. Daniel Liang**

*Armstrong State University*

Pearson

330 Hudson Street, NY NY 10013

*To Samantha, Michael, and Michelle*

Senior Vice President Courseware Portfolio Management: Marcia J. Horton
Director, Portfolio Management: Engineering, Computer Science & Global Editions: Julian Partridge
Higher Ed Portfolio Management: Tracy Johnson (Dunkelberger)
Portfolio Management Assistant: Kristy Alaura
Managing Content Producer: Scott Disanno
Content Producer: Robert Engelhardt
Web Developer: Steve Wright
Rights and Permissions Manager: Ben Ferrini
Manufacturing Buyer, Higher Ed, Lake Side Communications Inc (LSC): Maura Zaldivar-Garcia
Inventory Manager: Ann Lam
Marketing Manager: Demetrius Hall
Product Marketing Manager: Bram Van Kempen
Marketing Assistant: Jon Bryant
Cover Designer: Marta Samsel
Cover Photography: Germano Poli/123RF.com
Full-Service Project Management: Shylaja Gattupalli, SPi Global

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on the appropriate page within text.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided "as is" without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services. The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

**Library of Congress Cataloging-in-Publication Data**

Names: Liang, Y. Daniel, author.
Title: Introduction to Java programming and data structures / Y. Daniel Liang, Armstrong State University.
Other titles: Introduction to Java programming
Description: Eleventh edition. Comprehensive version. | New York, NY : Pearson Education, 2017. | Revised edition of: Introduction to Java programming / Y. Daniel Liang, Armstrong Atlantic State University. Tenth edition. Comprehensive version. 2015. | Includes index.
Identifiers: LCCN 2017002082| ISBN 9780134670942 | ISBN 0134670949
Subjects: LCSH: Java (Computer program language)
Classification: LCC QA76.73.J38 L52 2017 | DDC 005.13/3--dc23 LC record available at https://lccn.loc.gov/2017002082

1–17

Pearson

ISBN-10: 0-13-467094-9
ISBN-13: 978-0-13-467094-2

# PREFACE

Dear Reader,

Many of you have provided feedback on earlier editions of this book, and your comments and suggestions have greatly improved the book. This edition has been substantially enhanced in presentation, organization, examples, exercises, and supplements.

fundamentals-first

The book is fundamentals first by introducing basic programming concepts and techniques before designing custom classes. The fundamental concepts and techniques of selection statements, loops, methods, and arrays are the foundation for programming. Building this strong foundation prepares students to learn object-oriented programming and advanced Java programming.

problem-driven

This book teaches programming in a problem-driven way that focuses on problem solving rather than syntax. We make introductory programming interesting by using thought-provoking problems in a broad context. The central thread of early chapters is on problem solving. Appropriate syntax and library are introduced to enable readers to write programs for solving the problems. To support the teaching of programming in a problem-driven way, the book provides a wide variety of problems at various levels of difficulty to motivate students. To appeal to students in all majors, the problems cover many application areas, including math, science, business, financial, gaming, animation, and multimedia.

data structures

The book seamlessly integrates programming, data structures, and algorithms into one text. It employs a practical approach to teach data structures. We first introduce how to use various data structures to develop efficient algorithms, and then show how to implement these data structures. Through implementation, students gain a deep understanding on the efficiency of data structures and on how and when to use certain data structures. Finally, we design and implement custom data structures for trees and graphs.

comprehensive version

brief version

AP Computer Science

The book is widely used in the introductory programming, data structures, and algorithms courses in the universities around the world. This *comprehensive version* covers fundamentals of programming, object-oriented programming, GUI programming, data structures, algorithms, concurrency, networking, database, and Web programming. It is designed to prepare students to become proficient Java programmers. A *brief version* (*Introduction to Java Programming*, Brief Version, Eleventh Edition) is available for a first course on programming, commonly known as CS1. The brief version contains the first 18 chapters of the comprehensive version. An AP version of the book is also available for high school students taking an AP Computer Science course.

examples and exercises

The best way to teach programming is *by example*, and the only way to learn programming is *by doing*. Basic concepts are explained by example and a large number of exercises with various levels of difficulty are provided for students to practice. For our programming courses, we assign programming exercises after each lecture.

Our goal is to produce a text that teaches problem solving and programming in a broad context using a wide variety of interesting examples. If you have any comments and suggestions for improving the book, please email me.

Sincerely,

Y. Daniel Liang
**y.daniel.liang@gmail.com**
www.cs.armstrong.edu/liang
**www.pearsonhighered.com/liang**

## ACM/IEEE Curricular 2013 and ABET Course Assessment

The new ACM/IEEE Computer Science Curricular 2013 defines the Body of Knowledge organized into 18 Knowledge Areas. To help instructors design the courses based on this book, we provide sample syllabi to identify the Knowledge Areas and Knowledge Units. The sample syllabi are for a three semester course sequence and serve as an example for institutional customization. The sample syllabi are accessible from the Instructor Resource Website.

Many of our users are from the ABET-accredited programs. A key component of the ABET accreditation is to identify the weakness through continuous course assessment against the course outcomes. We provide sample course outcomes for the courses and sample exams for measuring course outcomes on the Instructor Resource Website.

## What's New in This Edition?

This edition is completely revised in every detail to enhance clarity, presentation, content, examples, and exercises. The major improvements are as follows:

- The book's title is changed to Introduction to Java Programming and Data Structures with new enhancements on data structures. The book uses a practical approach to introduce design, implement, and use data structures and covers all topics in a typical data structures course. Additionally, it provides bonus chapters that cover advanced data structures such as 2-4 trees, B-trees, and red-black trees.
- Updated to the latest Java technology. Examples and exercises are improved and simplified by using the new features in Java 8.
- The default and static methods are introduced for interfaces in Chapter 13.
- The GUI chapters are updated to JavaFX 8. The examples are revised. The user interfaces in the examples and exercises are now resizable and displayed in the center of the window.
- Inner classes, anonymous inner classes, and lambda expressions are covered using practical examples in Chapter 15.
- More examples and exercises in the data structures chapters use lambda expressions to simplify coding. Method references are introduced along with the `Comparator` interface in Section 20.6.
- The `forEach` method is introduced in Chapter 20 as a simple alternative to the foreach loop for applying an action to each element in a collection.
- Use the default methods for interfaces in Java 8 to redesign and simplify `MyList`, `MyArrayList`, `MyLinkedList`, `Tree`, `BST`, `AVLTree`, `MyMap`, `MyHashMap`, `MySet`, `MyHashSet`, `Graph`, `UnweightedGraph`, and `WeightedGraph` in Chapters 24–29.
- Chapter 30 is brand new to introduce aggregate operations for collection streams.
- FXML and the Scene Builder visual tool are introduced in Chapter 31.
- The Companion Website has been redesigned with new interactive quiz, CheckPoint questions, animations, and live coding.
- More than 200 additional programming exercises with solutions are provided to the instructor on the Instructor Resource Website. These exercises are not printed in the text.

Please visit www.pearsonhighered.com/liang for a complete list of new features as well as correlations to the previous edition.

## Pedagogical Features

The book uses the following elements to help students get the most from the material:

- The **Objectives** at the beginning of each chapter list what students should learn from the chapter. This will help them determine whether they have met the objectives after completing the chapter.
- The **Introduction** opens the discussion with a thought-provoking question to motivate the reader to delve into the chapter.
- **Key Points** highlight the important concepts covered in each section.
- **Check Points** provide review questions to help students track their progress as they read through the chapter and evaluate their learning.
- **Problems and Case Studies**, carefully chosen and presented in an easy-to-follow style, teach problem solving and programming concepts. The book uses many small, simple, and stimulating examples to demonstrate important ideas.
- The **Chapter Summary** reviews the important subjects that students should understand and remember. It helps them reinforce the key concepts they have learned in the chapter.
- **Quizzes** are accessible online, grouped by sections, for students to do self-test on programming concepts and techniques.
- **Programming Exercises** are grouped by sections to provide students with opportunities to apply the new skills they have learned on their own. The level of difficulty is rated as easy (no asterisk), moderate (*), hard (**), or challenging (***). The trick of learning programming is practice, practice, and practice. To that end, the book provides a great many exercises. Additionally, more than 200 programming exercises with solutions are provided to the instructors on the Instructor Resource Website. These exercises are not printed in the text.
- **Notes**, **Tips**, **Cautions**, and **Design Guides** are inserted throughout the text to offer valuable advice and insight on important aspects of program development.

### Note

Provides additional information on the subject and reinforces important concepts.

### Tip

Teaches good programming style and practice.

### Caution

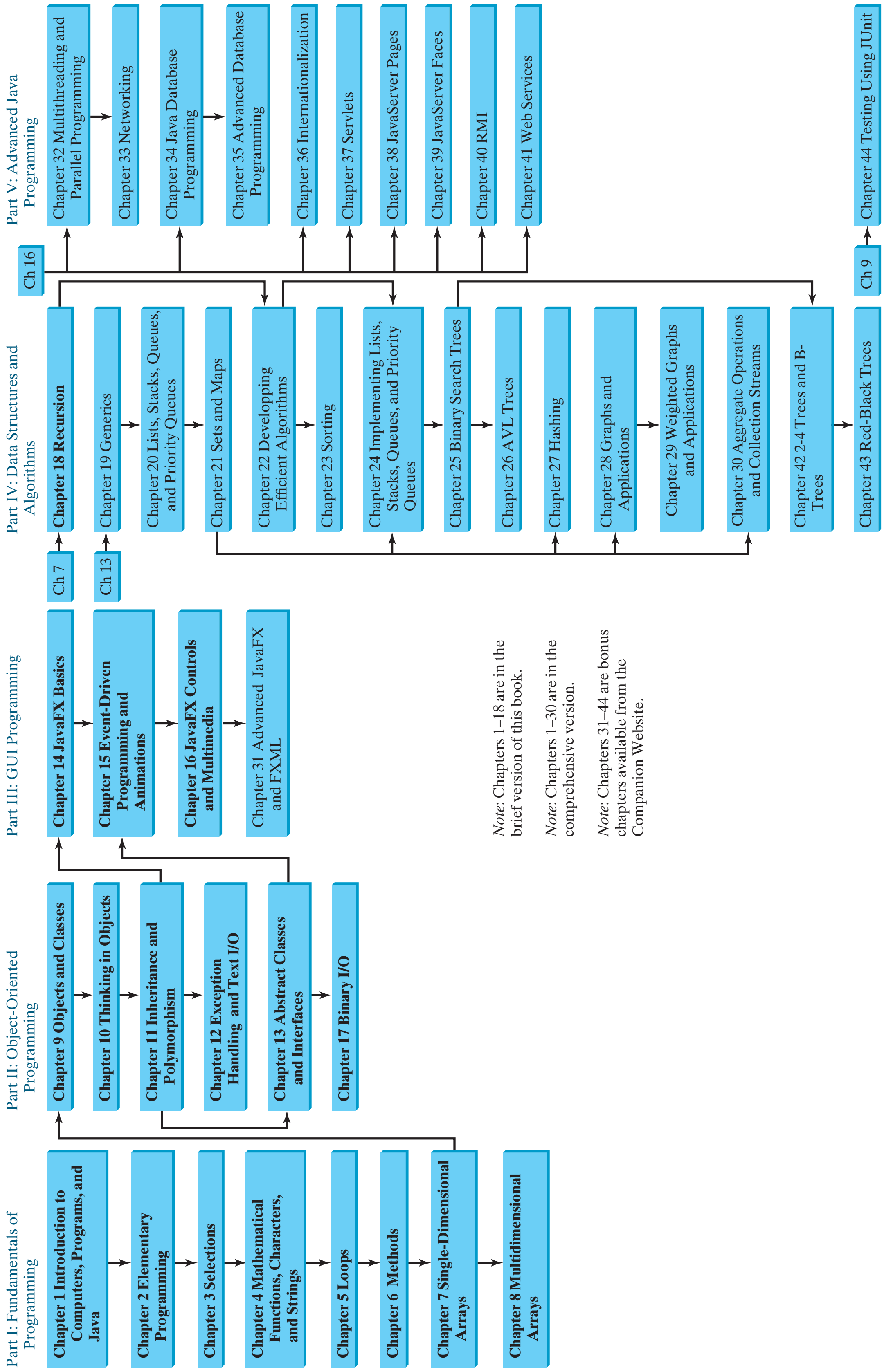
Helps students steer away from the pitfalls of programming errors.

### Design Guide

Provides guidelines for designing programs.

## Flexible Chapter Orderings

The book is designed to provide flexible chapter orderings to enable GUI, exception handling, recursion, generics, and the Java Collections Framework to be covered earlier or later. The diagram on the next page shows the chapter dependencies.


Part I: Fundamentals of Programming
Chapter 1 Introduction to Computers, Programs, and Java
Chapter 2 Elementary Programming
Chapter 3 Selections
Chapter 4 Mathematical Functions, Characters, and Strings
Chapter 5 Loops
Chapter 6 Methods
Chapter 7 Single-Dimensional Arrays
Chapter 8 Multidimensional Arrays
Part II: Object-Oriented Programming
Chapter 9 Objects and Classes
Chapter 10 Thinking in Objects
Chapter 11 Inheritance and Polymorphism
Chapter 12 Exception Handling and Text I/O
Chapter 13 Abstract Classes and Interfaces
Chapter 17 Binary I/O
Part III: GUI Programming
Chapter 14 JavaFX Basics
Chapter 15 Event-Driven Programming and Animations
Chapter 16 JavaFX Controls and Multimedia
Chapter 31 Advanced JavaFX and FXML
Part IV: Data Structures and Algorithms
Ch 7
Ch 13
Chapter 18 Recursion
Chapter 19 Generics
Chapter 20 Lists, Stacks, Queues, and Priority Queues
Chapter 21 Sets and Maps
Chapter 22 Developping Efficient Algorithms
Chapter 23 Sorting
Chapter 24 Implementing Lists, Stacks, Queues, and Priority Queues
Chapter 25 Binary Search Trees
Chapter 26 AVL Trees
Chapter 27 Hashing
Chapter 28 Graphs and Applications
Chapter 29 Weighted Graphs and Applications
Chapter 30 Aggregate Operations and Collection Streams
Chapter 42 2-4 Trees and B-Trees
Chapter 43 Red-Black Trees
Part V: Advanced Java Programming
Ch 16
Chapter 32 Multithreading and Parallel Programming
Chapter 33 Networking
Chapter 34 Java Database Programming
Chapter 35 Advanced Database Programming
Chapter 36 Internationalization
Chapter 37 Servlets
Chapter 38 JavaServer Pages
Chapter 39 JavaServer Faces
Chapter 40 RMI
Chapter 41 Web Services
Ch 9
Chapter 44 Testing Using JUnit

*Note*: Chapters 1–18 are in the brief version of this book.

*Note*: Chapters 1–30 are in the comprehensive version.

*Note*: Chapters 31–44 are bonus chapters available from the Companion Website.

## Organization of the Book

The chapters can be grouped into five parts that, taken together, form a comprehensive introduction to Java programming, data structures and algorithms, and database and Web programming. Because knowledge is cumulative, the early chapters provide the conceptual basis for understanding programming and guide students through simple examples and exercises; subsequent chapters progressively present Java programming in detail, culminating with the development of comprehensive Java applications. The appendixes contain a mixed bag of topics, including an introduction to number systems, bitwise operations, regular expressions, and enumerated types.

### Part I: Fundamentals of Programming (Chapters 1–8)

The first part of the book is a stepping stone, preparing you to embark on the journey of learning Java. You will begin to learn about Java (Chapter 1) and fundamental programming techniques with primitive data types, variables, constants, assignments, expressions, and operators (Chapter 2), selection statements (Chapter 3), mathematical functions, characters, and strings (Chapter 4), loops (Chapter 5), methods (Chapter 6), and arrays (Chapters 7–8). After Chapter 7, you can jump to Chapter 18 to learn how to write recursive methods for solving inherently recursive problems.

### Part II: Object-Oriented Programming (Chapters 9–13, and 17)

This part introduces object-oriented programming. Java is an object-oriented programming language that uses abstraction, encapsulation, inheritance, and polymorphism to provide great flexibility, modularity, and reusability in developing software. You will learn programming with objects and classes (Chapters 9–10), class inheritance (Chapter 11), polymorphism (Chapter 11), exception handling (Chapter 12), abstract classes (Chapter 13), and interfaces (Chapter 13). Text I/O is introduced in Chapter 12 and binary I/O is discussed in Chapter 17.

### Part III: GUI Programming (Chapters 14–16 and Bonus Chapter 31)

JavaFX is a new framework for developing Java GUI programs. It is not only useful for developing GUI programs, but also an excellent pedagogical tool for learning object-oriented programming. This part introduces Java GUI programming using JavaFX in Chapters 14–16. Major topics include GUI basics (Chapter 14), container panes (Chapter 14), drawing shapes (Chapter 14), event-driven programming (Chapter 15), animations (Chapter 15), and GUI controls (Chapter 16), and playing audio and video (Chapter 16). You will learn the architecture of JavaFX GUI programming and use the controls, shapes, panes, image, and video to develop useful applications. Chapter 31 covers advanced features in JavaFX.

### Part IV: Data Structures and Algorithms (Chapters 18–30 and Bonus Chapters 42–43)

This part covers the main subjects in a typical data structures and algorithms course. Chapter 18 introduces recursion to write methods for solving inherently recursive problems. Chapter 19 presents how generics can improve software reliability. Chapters 20 and 21 introduce the Java Collection Framework, which defines a set of useful API for data structures. Chapter 22 discusses measuring algorithm efficiency in order to choose an appropriate algorithm for applications. Chapter 23 describes classic sorting algorithms. You will learn how to implement several classic data structures lists, queues, and priority queues in Chapter 24. Chapters 25 and 26 introduce binary search trees and AVL trees. Chapter 27 presents hashing and implementing maps and sets using hashing. Chapters 28 and 29 introduce graph applications. Chapter 30 introduces aggregate operations for collection streams. The 2-4 trees, B-trees, and red-black trees are covered in Bonus Chapters 42–43.

### Part V: Advanced Java Programming (Chapters 32-41, 44)

This part of the book is devoted to advanced Java programming. Chapter 32 treats the use of multithreading to make programs more responsive and interactive and introduces parallel programming. Chapter 33 discusses how to write programs that talk with each other from different hosts over the Internet. Chapter 34 introduces the use of Java to develop database projects. Chapter 35 delves into advanced Java database programming. Chapter 36 covers the use of internationalization support to develop projects for international audiences. Chapters 37 and 38 introduce how to use Java servlets and JavaServer Pages to generate dynamic content from Web servers. Chapter 39 introduces modern Web application development using JavaServer Faces. Chapter 40 introduces remote method invocation and Chapter 41 discusses Web services. Chapter 44 introduces testing Java programs using JUnit.

### Appendixes

This part of the book covers a mixed bag of topics. Appendix A lists Java keywords. Appendix B gives tables of ASCII characters and their associated codes in decimal and in hex. Appendix C shows the operator precedence. Appendix D summarizes Java modifiers and their usage. Appendix E discusses special floating-point values. Appendix F introduces number systems and conversions among binary, decimal, and hex numbers. Finally, Appendix G introduces bitwise operations. Appendix H introduces regular expressions. Appendix I covers enumerated types.

## Java Development Tools

You can use a text editor, such as the Windows Notepad or WordPad, to create Java programs and to compile and run the programs from the command window. You can also use a Java development tool, such as NetBeans or Eclipse. These tools support an integrated development environment (IDE) for developing Java programs quickly. Editing, compiling, building, executing, and debugging programs are integrated in one graphical user interface. Using these tools effectively can greatly increase your programming productivity. NetBeans and Eclipse are easy to use if you follow the tutorials. Tutorials on NetBeans and Eclipse can be found in the supplements on the Companion Website www.pearsonhighered.com/liang.

IDE tutorials

## Student Resource Website

The Student Resource Website (www.pearsonhighered.com/liang) contains the following resources:

- Answers to CheckPoint questions
- Solutions to majority of even-numbered programming exercises
- Source code for the examples in the book
- Interactive quiz (organized by sections for each chapter)
- Supplements
- Debugging tips
- Video notes
- Algorithm animations
- Errata

## Supplements

The text covers the essential subjects. The supplements extend the text to introduce additional topics that might be of interest to readers. The supplements are available from the Companion Website.

# Instructor Resource Website

The Instructor Resource Website, accessible from www.pearsonhighered.com/liang, contains the following resources:

- Microsoft PowerPoint slides with interactive buttons to view full-color, syntax-highlighted source code and to run programs without leaving the slides.
- Solutions to majority of odd-numbered programming exercises.
- More than 200 additional programming exercises and 300 quizzes organized by chapters. These exercises and quizzes are available only to the instructors. Solutions to these exercises and quizzes are provided.
- Web-based quiz generator. (Instructors can choose chapters to generate quizzes from a large database of more than two thousand questions.)
- Sample exams. Most exams have four parts:
    - Multiple-choice questions or short-answer questions
    - Correct programming errors
    - Trace programs
    - Write programs
- Sample exams with ABET course assessment.
- Projects. In general, each project gives a description and asks students to analyze, design, and implement the project.

Some readers have requested the materials from the Instructor Resource Website. Please understand that these are for instructors only. Such requests will not be answered.

# Online Practice and Assessment with MyProgrammingLab

MyProgrammingLab™

MyProgrammingLab helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyProgrammingLab improves the programming competence of beginning students who often struggle with the basic concepts and paradigms of popular high-level programming languages.

A self-study and homework tool, a MyProgrammingLab course consists of hundreds of small practice problems organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of their code submissions and offers targeted hints that enable students to figure out what went wrong—and why. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code inputted by students for review.

MyProgrammingLab is offered to users of this book in partnership with Turing's Craft, the makers of the CodeLab interactive programming exercise system. For a full demonstration, to see feedback from instructors and students, or to get started using MyProgrammingLab in your course, visit www.myprogramminglab.com.

# Video Notes

VideoNote

We are excited about the new Video Notes feature that is found in this new edition. These videos provide additional help by presenting examples of key topics and showing how to solve problems completely from design through coding. Video Notes are available from www.pearsonhighered.com/liang.

Animation

## Algorithm Animations

We have provided numerous animations for algorithms. These are valuable pedagogical tools to demonstrate how algorithms work. Algorithm animations can be accessed from the Companion Website.

## Acknowledgments

I would like to thank Armstrong State University for enabling me to teach what I write and for supporting me in writing what I teach. Teaching is the source of inspiration for continuing to improve the book. I am grateful to the instructors and students who have offered comments, suggestions, corrections, and praise. My special thanks go to Stefan Andrei of Lamar University and William Bahn of University of Colorado Colorado Spring for their help to improve the data structures part of this book.

This book has been greatly enhanced thanks to outstanding reviews for this and previous editions. The reviewers are: Elizabeth Adams (James Madison University), Syed Ahmed (North Georgia College and State University), Omar Aldawud (Illinois Institute of Technology), Stefan Andrei (Lamar University), Yang Ang (University of Wollongong, Australia), Kevin Bierre (Rochester Institute of Technology), Aaron Braskin (Mira Costa High School), David Champion (DeVry Institute), James Chegwidden (Tarrant County College), Anup Dargar (University of North Dakota), Daryl Detrick (Warren Hills Regional High School), Charles Dierbach (Towson University), Frank Ducrest (University of Louisiana at Lafayette), Erica Eddy (University of Wisconsin at Parkside), Summer Ehresman (Center Grove High School), Deena Engel (New York University), Henry A. Etlinger (Rochester Institute of Technology), James Ten Eyck (Marist College), Myers Foreman (Lamar University), Olac Fuentes (University of Texas at El Paso), Edward F. Gehringer (North Carolina State University), Harold Grossman (Clemson University), Barbara Guillot (Louisiana State University), Stuart Hansen (University of Wisconsin, Parkside), Dan Harvey (Southern Oregon University), Ron Hofman (Red River College, Canada), Stephen Hughes (Roanoke College), Vladan Jovanovic (Georgia Southern University), Deborah Kabura Kariuki (Stony Point High School), Edwin Kay (Lehigh University), Larry King (University of Texas at Dallas), Nana Kofi (Langara College, Canada), George Koutsogiannakis (Illinois Institute of Technology), Roger Kraft (Purdue University at Calumet), Norman Krumpe (Miami University), Hong Lin (DeVry Institute), Dan Lipsa (Armstrong State University), James Madison (Rensselaer Polytechnic Institute), Frank Malinowski (Darton College), Tim Margush (University of Akron), Debbie Masada (Sun Microsystems), Blayne Mayfield (Oklahoma State University), John McGrath (J.P. McGrath Consulting), Hugh McGuire (Grand Valley State), Shyamal Mitra (University of Texas at Austin), Michel Mitri (James Madison University), Kenrick Mock (University of Alaska Anchorage), Frank Murgolo (California State University, Long Beach), Jun Ni (University of Iowa), Benjamin Nystuen (University of Colorado at Colorado Springs), Maureen Opkins (CA State University, Long Beach), Gavin Osborne (University of Saskatchewan), Kevin Parker (Idaho State University), Dale Parson (Kutztown University), Mark Pendergast (Florida Gulf Coast University), Richard Povinelli (Marquette University), Roger Priebe (University of Texas at Austin), Mary Ann Pumphrey (De Anza Junior College), Pat Roth (Southern Polytechnic State University), Amr Sabry (Indiana University), Ben Setzer (Kennesaw State University), Carolyn Schauble (Colorado State University), David Scuse (University of Manitoba), Ashraf Shirani (San Jose State University), Daniel Spiegel (Kutztown University), Joslyn A. Smith (Florida Atlantic University), Lixin Tao (Pace University), Ronald F. Taylor (Wright State University), Russ Tront (Simon Fraser University), Deborah Trytten (University of Oklahoma), Michael Verdicchio (Citadel), Kent Vidrine (George Washington University), and Bahram Zartoshty (California State University at Northridge).

It is a great pleasure, honor, and privilege to work with Pearson. I would like to thank Tracy Johnson and her colleagues Marcia Horton, Demetrius Hall, Yvonne Vannatta, Kristy Alaura, Carole Snyder, Scott Disanno, Bob Engelhardt, Shylaja Gattupalli, and their colleagues for organizing, producing, and promoting this project.

As always, I am indebted to my wife, Samantha, for her love, support, and encouragement.

# BRIEF CONTENTS

CHAPTER 31–44 are available from the Companion Website at *www.pearsonhighered.com/liang*

# CONTENTS

Chapter 31–44 are available from the Companion Website at www.pearsonhighered.com/liang

# VideoNotes

Locations of VideoNotes

http://www.pearsonhighered.com/liang

VideoNote

# Animations

CHAPTER

1

# Introduction to Computers, Programs, and Java™

## Objectives

- To understand computer basics, programs, and operating systems (§§1.2–1.4).
- To describe the relationship between Java and the World Wide Web (§1.5).
- To understand the meaning of Java language specification, API, JDK™, JRE™, and IDE (§1.6).
- To write a simple Java program (§1.7).
- To display output on the console (§1.7).
- To explain the basic syntax of a Java program (§1.7).
- To create, compile, and run Java programs (§1.8).
- To use sound Java programming style and document programs properly (§1.9).
- To explain the differences between syntax errors, runtime errors, and logic errors (§1.10).
- To develop Java programs using NetBeans™ (§1.11).
- To develop Java programs using Eclipse™ (§1.12).

## 1.1 Introduction

Key Point

*The central theme of this book is to learn how to solve problems by writing a program.*

what is programming?
programming
program

This book is about programming. So, what is programming? The term *programming* means to create (or develop) software, which is also called a *program.* In basic terms, software contains instructions that tell a computer—or a computerized device—what to do.

Software is all around you, even in devices you might not think would need it. Of course, you expect to find and use software on a personal computer, but software also plays a role in running airplanes, cars, cell phones, and even toasters. On a personal computer, you use word processors to write documents, web browsers to explore the Internet, and e-mail programs to send and receive messages. These programs are all examples of software. Software developers create software with the help of powerful tools called *programming languages.*

This book teaches you how to create programs by using the Java programming language. There are many programming languages, some of which are decades old. Each language was invented for a specific purpose—to build on the strengths of a previous language, for example, or to give the programmer a new and unique set of tools. Knowing there are so many programming languages available, it would be natural for you to wonder which one is best. However, in truth, there is no "best" language. Each one has its own strengths and weaknesses. Experienced programmers know one language might work well in some situations, whereas a different language may be more appropriate in other situations. For this reason, seasoned programmers try to master as many different programming languages as they can, giving them access to a vast arsenal of software-development tools.

If you learn to program using one language, you should find it easy to pick up other languages. The key is to learn how to solve problems using a programming approach. That is the main theme of this book.

You are about to begin an exciting journey: learning how to program. At the outset, it is helpful to review computer basics, programs, and operating systems (OSs). If you are already familiar with such terms as central processing unit (CPU), memory, disks, operating systems, and programming languages, you may skip Sections 1.2–1.4.

## 1.2 What Is a Computer?

Key Point

*A computer is an electronic device that stores and processes data.*

hardware
software

A computer includes both *hardware* and *software.* In general, hardware comprises the visible, physical elements of the computer, and software provides the invisible instructions that control the hardware and make it perform specific tasks. Knowing computer hardware isn't essential to learning a programming language, but it can help you better understand the effects that a program's instructions have on the computer and its components. This section introduces computer hardware components and their functions.

A computer consists of the following major hardware components (see Figure 1.1):

- A central processing unit (CPU)
- Memory (main memory)
- Storage devices (such as disks and CDs)
- Input devices (such as the mouse and the keyboard)
- Output devices (such as monitors and printers)
- Communication devices (such as modems and network interface cards (NIC))

bus

A computer's components are interconnected by a subsystem called a *bus.* You can think of a bus as a sort of system of roads running among the computer's components; data and power travel along the bus from one part of the computer to another. In personal computers,

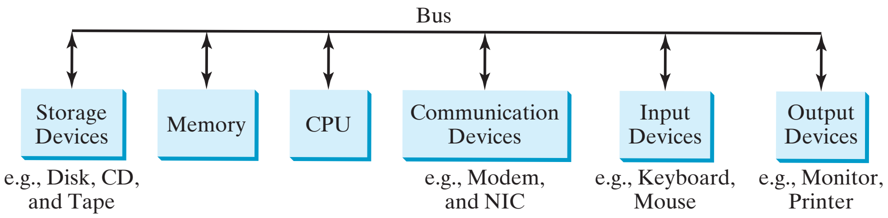


**Figure 1.1** A computer consists of a CPU, memory, storage devices, input devices, output devices, and communication devices.

the bus is built into the computer's *motherboard*, which is a circuit case that connects all of the parts of a computer together.

motherboard

## 1.2.1 Central Processing Unit

CPU

The *central processing unit (CPU)* is the computer's brain. It retrieves instructions from the memory and executes them. The CPU usually has two components: a *control unit* and an *arithmetic/logic unit.* The control unit controls and coordinates the actions of the other components. The arithmetic/logic unit performs numeric operations (addition, subtraction, multiplication, and division) and logical operations (comparisons).

Today's CPUs are built on small silicon semiconductor chips that contain millions of tiny electric switches, called *transistors*, for processing information.

speed

hertz

megahertz

gigahertz

Every computer has an internal clock that emits electronic pulses at a constant rate. These pulses are used to control and synchronize the pace of operations. A higher clock *speed* enables more instructions to be executed in a given period of time. The unit of measurement of clock speed is the *hertz* (*Hz*), with 1 Hz equaling 1 pulse per second. In the 1990s, computers measured clock speed in *megahertz* (*MHz*), but CPU speed has been improving continuously; the clock speed of a computer is now usually stated in *gigahertz (GHz)*. Intel's newest processors run at about 3 GHz.

core

CPUs were originally developed with only one core. The *core* is the part of the processor that performs the reading and executing of instructions. In order to increase the CPU processing power, chip manufacturers are now producing CPUs that contain multiple cores. A multicore CPU is a single component with two or more independent cores. Today's consumer computers typically have two, three, and even four separate cores. Soon, CPUs with dozens or even hundreds of cores will be affordable.

## 1.2.2 Bits and Bytes

Before we discuss memory, let's look at how information (data and programs) are stored in a computer.

bits

A computer is really nothing more than a series of switches. Each switch exists in two states: on or off. Storing information in a computer is simply a matter of setting a sequence of switches on or off. If the switch is on, its value is 1. If the switch is off, its value is 0. These 0s and 1s are interpreted as digits in the binary number system and are called *bits* (binary digits).

byte

The minimum storage unit in a computer is a *byte.* A byte is composed of eight bits. A small number such as 3 can be stored as a single byte. To store a number that cannot fit into a single byte, the computer uses several bytes.

encoding scheme

Data of various kinds, such as numbers and characters, are encoded as a series of bytes. As a programmer, you don't need to worry about the encoding and decoding of data, which the computer system performs automatically, based on the encoding scheme. An *encoding scheme* is a set of rules that govern how a computer translates characters and numbers into data with which the computer can actually work. Most schemes translate each character into a predetermined string of bits. In the popular ASCII encoding scheme, for example, the character `C` is represented as `01000011` in 1 byte.

A computer's storage capacity is measured in bytes and multiples of the byte, as follows:

kilobyte (KB)

megabyte (MB)

gigabyte (GB)

terabyte (TB)

- A *kilobyte (KB)* is about 1,000 bytes.
- A *megabyte (MB)* is about 1 million bytes.
- A *gigabyte (GB)* is about 1 billion bytes.
- A *terabyte (TB)* is about 1 trillion bytes.

A typical one-page word document might take 20 KB. Therefore, 1 MB can store 50 pages of documents, and 1 GB can store 50,000 pages of documents. A typical two-hour high-resolution movie might take 8 GB, so it would require 160 GB to store 20 movies.

## 1.2.3 Memory

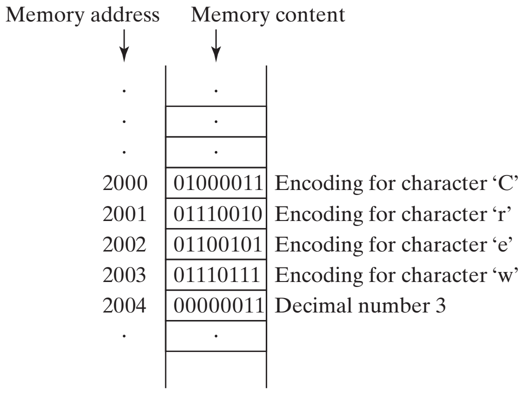
memory

A computer's *memory* consists of an ordered sequence of bytes for storing programs as well as data with which the program is working. You can think of memory as the computer's work area for executing a program. A program and its data must be moved into the computer's memory before they can be executed by the CPU.

unique address

RAM

Every byte in the memory has a *unique address*, as shown in Figure 1.2. The address is used to locate the byte for storing and retrieving the data. Since the bytes in the memory can be accessed in any order, the memory is also referred to as *random-access memory (RAM)*.



| | | |
|---|---|---|
| . | . | |
| . | . | |
| . | . | |
| 2000 | 01000011 | Encoding for character 'C' |
| 2001 | 01110010 | Encoding for character 'r' |
| 2002 | 01100101 | Encoding for character 'e' |
| 2003 | 01110111 | Encoding for character 'w' |
| 2004 | 00000011 | Decimal number 3 |
| . | . | |

**FIGURE 1.2** Memory stores data and program instructions in uniquely addressed memory locations.

Today's personal computers usually have at least 4 GB of RAM, but they more commonly have 6 to 8 GB installed. Generally speaking, the more RAM a computer has, the faster it can operate, but there are limits to this simple rule of thumb.

A memory byte is never empty, but its initial content may be meaningless to your program. The current content of a memory byte is lost whenever new information is placed in it.

Like the CPU, memory is built on silicon semiconductor chips that have millions of transistors embedded on their surface. Compared to CPU chips, memory chips are less complicated, slower, and less expensive.

## 1.2.4 Storage Devices

storage devices

A computer's memory (RAM) is a volatile form of data storage: Any information that has been saved in memory is lost when the system's power is turned off. Programs and data are permanently stored on *storage devices* and are moved, when the computer actually uses them, to memory, which operates at much faster speeds than permanent storage devices can.

There are three main types of storage devices:

- Magnetic disk drives
- Optical disc drives (CD and DVD)
- Universal serial bus (USB) flash drives

*Drives* are devices for operating a medium, such as disks and CDs. A storage medium physically stores data and program instructions. The drive reads data from the medium and writes data onto the medium. drive

### Disks

A computer usually has at least one hard disk drive. *Hard disks* are used for permanently storing data and programs. Newer computers have hard disks that can store from 500 GB to 1 TB of data. Hard disk drives are usually encased inside the computer, but removable hard disks are also available. hard disk

### CDs and DVDs

*CD* stands for compact disc. There are three types of CDs: CD-ROM, CD-R, and CD-RW. A CD-ROM is a prepressed disc. It was popular for distributing software, music, and video. Software, music, and video are now increasingly distributed on the Internet without using CDs. A *CD-R* (CD-Recordable) is a write-once medium. It can be used to record data once and read any number of times. A *CD-RW* (CD-ReWritable) can be used like a hard disk; that is, you can write data onto the disc, then overwrite that data with new data. A single CD can hold up to 700 MB. CD-ROM CD-R CD-RW

*DVD* stands for digital versatile disc or digital video disc. DVDs and CDs look alike, and you can use either to store data. A DVD can hold more information than a CD; a standard DVD's storage capacity is 4.7 GB. There are two types of DVDs: DVD-R (Recordable) and DVD-RW (ReWritable). DVD

### USB Flash Drives

*Universal serial bus (USB)* connectors allow the user to attach many kinds of peripheral devices to the computer. You can use an USB to connect a printer, digital camera, mouse, external hard disk drive, and other devices to the computer.

An USB *flash drive* is a device for storing and transporting data. A flash drive is small—about the size of a pack of gum. It acts like a portable hard drive that can be plugged into your computer's USB port. USB flash drives are currently available with up to 256 GB storage capacity.

## 1.2.5 Input and Output Devices

Input and output devices let the user communicate with the computer. The most common input devices are the *keyboard* and *mouse*. The most common output devices are *monitors* and *printers*.

### The Keyboard

A keyboard is a device for entering input. Compact keyboards are available without a numeric keypad.

*Function keys* are located across the top of the keyboard and are prefaced with the letter *F*. Their functions depend on the software currently being used. function key

A *modifier key* is a special key (such as the *Shift*, *Alt*, and *Ctrl* keys) that modifies the normal action of another key when the two are pressed simultaneously. modifier key numeric keypad

The *numeric keypad*, located on the right side of most keyboards, is a separate set of keys styled like a calculator to use for quickly entering numbers.

arrow keys

*Arrow keys,* located between the main keypad and the numeric keypad, are used to move the mouse pointer up, down, left, and right on the screen in many kinds of programs.

Insert key
Delete key
Page Up key
Page Down key

The *Insert*, *Delete*, *Page Up*, and *Page Down keys* are used in word processing and other programs for inserting text and objects, deleting text and objects, and moving up or down through a document one screen at a time.

### The Mouse

A *mouse* is a pointing device. It is used to move a graphical pointer (usually in the shape of an arrow) called a *cursor* around the screen, or to click on-screen objects (such as a button) to trigger them to perform an action.

### The Monitor

The *monitor* displays information (text and graphics). The screen resolution and dot pitch determine the quality of the display.

screen resolution
pixels

The *screen resolution* specifies the number of pixels in horizontal and vertical dimensions of the display device. *Pixels* (short for "picture elements") are tiny dots that form an image on the screen. A common resolution for a 17-inch screen, for example, is 1,024 pixels wide and 768 pixels high. The resolution can be set manually. The higher the resolution, the sharper and clearer the image is.

dot pitch

The *dot pitch* is the amount of space between pixels, measured in millimeters. The smaller the dot pitch, the sharper is the display.

## 1.2.6 Communication Devices

Computers can be networked through communication devices, such as a dial-up modem (*mo*dulator/*dem*odulator), a digital subscriber line (DSL) or cable modem, a wired network interface card, or a wireless adapter.

dial-up modem

- A *dial-up modem* uses a phone line to dial a phone number to connect to the Internet and can transfer data at a speed up to 56,000 bps (bits per second).

digital subscriber line (DSL)

- A *digital subscriber line (DSL)* connection also uses a standard phone line, but it can transfer data 20 times faster than a standard dial-up modem.

cable modem

- A *cable modem* uses the cable line maintained by the cable company and is generally faster than DSL.

network interface card (NIC)
local area network (LAN)
million bits per second (mbps)

- A *network interface card (NIC)* is a device that connects a computer to a *local area network (LAN)*. LANs are commonly used to connect computers within a limited area such as a school, a home, and an office. A high-speed NIC called *1000BaseT* can transfer data at 1,000 million bits per second (mbps).
- Wireless networking is now extremely popular in homes, businesses, and schools. Every laptop computer sold today is equipped with a wireless adapter that enables the computer to connect to the LAN and the Internet.

**Note**
Answers to the CheckPoint questions are available at **www.pearsonhighered.com/liang**. Choose this book and click Companion Website to select CheckPoint.

Check Point

**1.2.1** What are hardware and software?

**1.2.2** List the five major hardware components of a computer.

**1.2.3** What does the acronym CPU stand for? What unit is used to measure CPU speed?

**1.2.4** What is a bit? What is a byte?

**1.2.5** What is memory for? What does RAM stand for? Why is memory called RAM?

**1.2.6** What unit is used to measure memory size? What unit is used to measure disk size?

**1.2.7** What is the primary difference between memory and a storage device?

# 1.3 Programming Languages

*Computer programs, known as software, are instructions that tell a computer what to do.*

Key Point

Computers do not understand human languages, so programs must be written in a language a computer can use. There are hundreds of programming languages, and they were developed to make the programming process easier for people. However, all programs must be converted into the instructions the computer can execute.

## 1.3.1 Machine Language

machine language

A computer's native language, which differs among different types of computers, is its *machine language*—a set of built-in primitive instructions. These instructions are in the form of binary code, so if you want to give a computer an instruction in its native language, you have to enter the instruction as binary code. For example, to add two numbers, you might have to write an instruction in binary code as follows:

```
1101101010011010
```

## 1.3.2 Assembly Language

assembly language

Programming in machine language is a tedious process. Moreover, programs written in machine language are very difficult to read and modify. For this reason, *assembly language* was created in the early days of computing as an alternative to machine languages. Assembly language uses a short descriptive word, known as a *mnemonic*, to represent each of the machine-language instructions. For example, the mnemonic **add** typically means to add numbers, and **sub** means to subtract numbers. To add the numbers **2** and **3** and get the result, you might write an instruction in assembly code as follows:

```
add 2, 3, result
```

assembler

Assembly languages were developed to make programming easier. However, because the computer cannot execute assembly language, another program—called an *assembler*—is used to translate assembly-language programs into machine code, as shown in Figure 1.3.

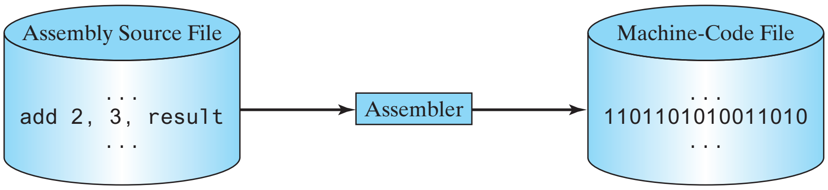


**FIGURE 1.3** An assembler translates assembly-language instructions into machine code.

Writing code in assembly language is easier than in machine language. However, it is still tedious to write code in assembly language. An instruction in assembly language essentially corresponds to an instruction in machine code. Writing in assembly language requires that you low-level language

know how the CPU works. Assembly language is referred to as a *low-level language*, because assembly language is close in nature to machine language and is machine dependent.

### 1.3.3 High-Level Language

high-level language

statement

In the 1950s, a new generation of programming languages known as *high-level languages* emerged. They are platform independent, which means that you can write a program in a high-level language and run it in different types of machines. High-level languages are similar to English and easy to learn and use. The instructions in a high-level programming language are called *statements*. Here, for example, is a high-level language statement that computes the area of a circle with a radius of **5**:

```
area = 5 * 5 * 3.14159;
```

There are many high-level programming languages, and each was designed for a specific purpose. Table 1.1 lists some popular ones.

**TABLE 1.1** Popular High-Level Programming Languages

| *Language* | *Description* |
|---|---|
| Ada | Named for Ada Lovelace, who worked on mechanical general-purpose computers. Developed for the Department of Defense and used mainly in defense projects. |
| BASIC | Beginner's All-purpose Symbolic Instruction Code. Designed to be learned and used easily by beginners. |
| C | Developed at Bell Laboratories. Combines the power of an assembly language with the ease of use and portability of a high-level language. |
| C++ | An object-oriented language, based on C |
| C# | Pronounced "C Sharp." An object-oriented programming language developed by Microsoft. |
| COBOL | COmmon Business Oriented Language. Used for business applications. |
| FORTRAN | FORmula TRANslation. Popular for scientific and mathematical applications. |
| Java | Developed by Sun Microsystems, now part of Oracle. An object-oriented programming language, widely used for developing platform-independent Internet applications. |
| JavaScript | A Web programming language developed by Netscape |
| Pascal | Named for Blaise Pascal, who pioneered calculating machines in the seventeenth century. A simple, structured, general-purpose language primarily for teaching programming. |
| Python | A simple general-purpose scripting language good for writing short programs. |
| Visual Basic | Visual Basic was developed by Microsoft. Enables the programmers to rapidly develop Windows-based applications. |

source program
source code

interpreter
compiler

A program written in a high-level language is called a *source program* or *source code*. Because a computer cannot execute a source program, a source program must be translated into machine code for execution. The translation can be done using another programming tool called an *interpreter* or a *compiler*.

- An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, then executes it right away, as shown in Figure 1.4a. Note a statement from the source code may be translated into several machine instructions.

- A compiler translates the entire source code into a machine-code file, and the machine-code file is then executed, as shown in Figure 1.4b.

Check Point

**1.3.1** What language does the CPU understand?

**1.3.2** What is an assembly language? What is an assembler?

**1.3.3** What is a high-level programming language? What is a source program?

**1.3.4** What is an interpreter? What is a compiler?

**1.3.5** What is the difference between an interpreted language and a compiled language?

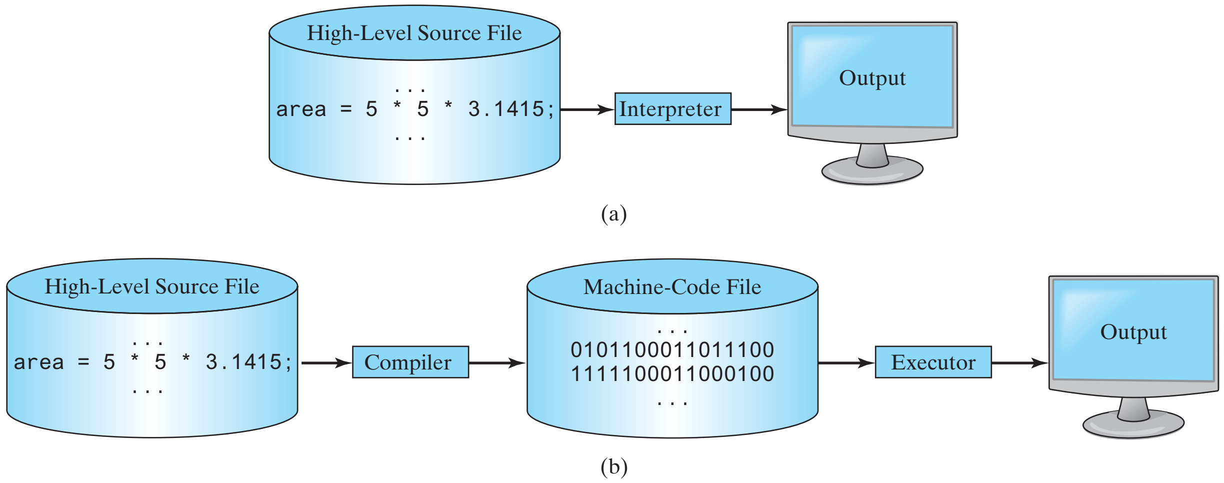


**Figure 1.4** (a) An interpreter translates and executes a program one statement at a time. (b) A compiler translates the entire source program into a machine-language file for execution.

## 1.4 Operating Systems

Key Point

*The* operating system (OS) *is the most important program that runs on a computer. The OS manages and controls a computer's activities.*

operating system (OS)

The popular *operating systems* for general-purpose computers are Microsoft Windows, Mac OS, and Linux. Application programs, such as a web browser or a word processor, cannot run unless an operating system is installed and running on the computer. Figure 1.5 shows the interrelationship of hardware, operating system, application software, and the user.

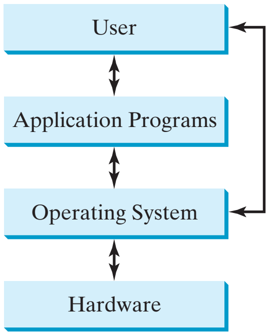


**Figure 1.5** Users and applications access the computer's hardware via the operating system.

The major tasks of an operating system are as follows:

- Controlling and monitoring system activities
- Allocating and assigning system resources
- Scheduling operations

### 1.4.1 Controlling and Monitoring System Activities

Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the monitor, keeping track of files and folders on storage devices, and controlling peripheral devices such as disk drives and printers. An operating system must also ensure different programs and users working at the same time do not interfere with each other. In addition, the OS is responsible for security, ensuring unauthorized users and programs are not allowed to access the system.

### 1.4.2 Allocating and Assigning System Resources

The operating system is responsible for determining what computer resources a program needs (such as CPU time, memory space, disks, and input and output devices) and for allocating and assigning them to run the program.

### 1.4.3 Scheduling Operations

multiprogramming
multithreading
multiprocessing

The OS is responsible for scheduling programs' activities to make efficient use of system resources. Many of today's operating systems support techniques such as *multiprogramming*, *multithreading*, and *multiprocessing* to increase system performance.

*Multiprogramming* allows multiple programs such as Microsoft Word, E-mail, and web browser to run simultaneously by sharing the same CPU. The CPU is much faster than the computer's other components. As a result, it is idle most of the time—for example, while waiting for data to be transferred from a disk or waiting for other system resources to respond. A multiprogramming OS takes advantage of this situation by allowing multiple programs to use the CPU when it would otherwise be idle. For example, multiprogramming enables you to use a word processor to edit a file at the same time as your web browser is downloading a file.

*Multithreading* allows a single program to execute multiple tasks at the same time. For instance, a word-processing program allows users to simultaneously edit text and save it to a disk. In this example, editing and saving are two tasks within the same program. These two tasks may run concurrently.

*Multiprocessing* is similar to multithreading. The difference is that multithreading is for running multithreads concurrently within one program, but multiprocessing is for running multiple programs concurrently using multiple processors.

Check Point

**1.4.1** What is an operating system? List some popular operating systems.

**1.4.2** What are the major responsibilities of an operating system?

**1.4.3** What are multiprogramming, multithreading, and multiprocessing?

## 1.5 Java, the World Wide Web, and Beyond

Key Point

*Java is a powerful and versatile programming language for developing software running on mobile devices, desktop computers, and servers.*

This book introduces Java programming. Java was developed by a team led by James Gosling at Sun Microsystems. Sun Microsystems was purchased by Oracle in 2010. Originally called *Oak,* Java was designed in 1991 for use in embedded chips in consumer electronic appliances.

In 1995, renamed *Java*, it was redesigned for developing web applications. For the history of Java, see www.java.com/en/javahistory/index.jsp.

Java has become enormously popular. Its rapid rise and wide acceptance can be traced to its design characteristics, particularly its promise that you can write a program once and run it anywhere. As stated by its designer, Java is *simple*, *object oriented*, *distributed*, *interpreted*, *robust*, *secure*, *architecture neutral*, *portable*, *high performance*, *multithreaded*, and *dynamic*. For the anatomy of Java characteristics, see liveexample.pearsoncmg.com/etc/JavaCharacteristics.pdf.

Java is a full-featured, general-purpose programming language that can be used to develop robust mission-critical applications. Today, it is employed not only for web programming but also for developing stand-alone applications across platforms on servers, desktop computers, and mobile devices. It was used to develop the code to communicate with and control the robotic rover on Mars. Many companies that once considered Java to be more hype than substance are now using it to create distributed applications accessed by customers and partners across the Internet. For every new project being developed today, companies are asking how they can use Java to make their work easier.

The World Wide Web is an electronic information repository that can be accessed on the Internet from anywhere in the world. The Internet, the Web's infrastructure, has been around for more than 40 years. The colorful World Wide Web and sophisticated web browsers are the major reason for the Internet's popularity.

Java initially became attractive because Java programs can run from a web browser. Such programs are called *applets.* Today applets are no longer allowed to run from a Web browser in the latest version of Java due to security issues. Java, however, is now very popular for developing applications on web servers. These applications process data, perform computations, and generate dynamic webpages. Many commercial Websites are developed using Java on the backend.

Java is a versatile programming language: You can use it to develop applications for desktop computers, servers, and small handheld devices. The software for Android cell phones is developed using Java.

Check Point

**1.5.1** Who invented Java? Which company owns Java now?

**1.5.2** What is a Java applet?

**1.5.3** What programming language does Android use?

## 1.6 The Java Language Specification, API, JDK, JRE, and IDE

Key Point

*Java syntax is defined in the Java language specification, and the Java library is defined in the Java application program interface (API). The JDK is the software for compiling and running Java programs. An IDE is an integrated development environment for rapidly developing programs.*

Computer languages have strict rules of usage. If you do not follow the rules when writing a program, the computer will not be able to understand it. The Java language specification and the Java API define the Java standards.

Java language specification

The *Java language specification* is a technical definition of the Java programming language's syntax and semantics. You can find the complete Java language specification at docs.oracle.com/javase/specs/.

API

library

The *application program interface (API)*, also known as *library*, contains predefined classes and interfaces for developing Java programs. The API is still expanding. You can view and download the latest version of the Java API at download.java.net/jdk8/docs/api/.

Java is a full-fledged and powerful language that can be used in many ways. It comes in three editions:

Java SE, EE, and ME

- *Java Standard Edition (Java SE)* to develop client-side applications. The applications can run on desktop.
- *Java Enterprise Edition (Java EE)* to develop server-side applications, such as Java servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF).
- *Java Micro Edition (Java ME)* to develop applications for mobile devices, such as cell phones.

Java Development Toolkit (JDK)

JDK 1.8 = JDK 8

This book uses Java SE to introduce Java programming. Java SE is the foundation upon which all other Java technology is based. There are many versions of Java SE. The latest, Java SE 8, is used in this book. Oracle releases each version with a *Java Development Toolkit (JDK).* For Java SE 8, the Java Development Toolkit is called *JDK 1.8* (also known as *Java 8* or *JDK 8*).

Java Runtime Environment (JRE)

Integrated development environment

The JDK consists of a set of separate programs, each invoked from a command line, for compiling, running, and testing Java programs. The program for running Java programs is known as *JRE (Java Runtime Environment)*. Instead of using the JDK, you can use a Java development tool (e.g., NetBeans, Eclipse, and TextPad)—software that provides an *integrated development environment (IDE)* for developing Java programs quickly. Editing, compiling, building, debugging, and online help are integrated in one graphical user interface. You simply enter source code in one window or open an existing file in a window, and then click a button or menu item or press a function key to compile and run the program.

Check Point

**1.6.1** What is the Java language specification?

**1.6.2** What does JDK stand for? What does JRE stand for?

**1.6.3** What does IDE stand for?

**1.6.4** Are tools like NetBeans and Eclipse different languages from Java, or are they dialects or extensions of Java?

## 1.7 A Simple Java Program

Key Point

*A Java program is executed from the* `main` *method in the class.*

what is a console?

console input

console output

Let's begin with a simple Java program that displays the message `Welcome to Java!` on the console. (The word *console* is an old computer term that refers to the text entry and display device of a computer. *Console input* means to receive input from the keyboard, and *console output* means to display output on the monitor.) The program is given in Listing 1.1.

**LISTING 1.1** `Welcome.java`

class

`main method`

display message

VideoNote

Your first Java program

```
public class Welcome {
  public static void main(String[] args) {
    // Display message Welcome to Java! on the console
    System.out.println("Welcome to Java!");
  }
}
```

```
Welcome to Java!
```

`line numbers`

Note the *line numbers* are for reference purposes only; they are not part of the program. So, don't type line numbers in your program.

Line 1 defines a class. Every Java program must have at least one class. Each class has a name. By convention, *class names* start with an uppercase letter. In this example, the class name is `Welcome`.

class name

Line 2 defines the `main` method. The program is executed from the `main` method. A class may contain several methods. The `main` method is the entry point where the program begins execution.

main method

A method is a construct that contains statements. The `main` method in this program contains the `System.out.println` statement. This statement displays the string `Welcome to Java!` on the console (line 4). *String* is a programming term meaning a sequence of characters. A string must be enclosed in double quotation marks. Every statement in Java ends with a semicolon (`;`), known as the *statement terminator.*

string

statement terminator

*Reserved words,* or *keywords*, have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word `class`, it understands that the word after `class` is the name for the class. Other reserved words in this program are `public`, `static`, and `void`.

reserved word

keyword

Line 3 is a *comment* that documents what the program is and how it is constructed. Comments help programmers to communicate and understand the program. They are not programming statements, and thus are ignored by the compiler. In Java, comments are preceded by two slashes (`//`) on a line, called a *line comment,* or enclosed between `/*` and `*/` on one or several lines, called a *block comment* or *paragraph comment*. When the compiler sees `//`, it ignores all text after `//` on the same line. When it sees `/*`, it scans for the next `*/` and ignores any text between `/*` and `*/`. Here are examples of comments:

comment

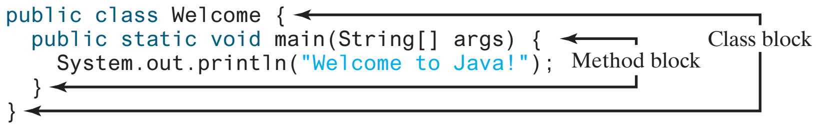
line comment

block comment

```
// This application program displays Welcome to Java!
/* This application program displays Welcome to Java! */
/* This application program
   displays Welcome to Java! */
```

A pair of braces in a program forms a *block* that groups the program's components. In Java, each block begins with an opening brace (`{`) and ends with a closing brace (`}`). Every class has a *class block* that groups the data and methods of the class. Similarly, every method has a *method block* that groups the statements in the method. Blocks can be *nested*, meaning that one block can be placed within another, as shown in the following code:

block



```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

## Tip

An opening brace must be matched by a closing brace. Whenever you type an opening brace, immediately type a closing brace to prevent the missing-brace error. Most Java IDEs automatically insert the closing brace for each opening brace.

match braces

## Caution

Java source programs are case sensitive. It would be wrong, for example, to replace `main` in the program with `Main`.

case sensitive

You have seen several special characters (e.g., `{ }`, `//`, `;`) in the program. They are used in almost every program. Table 1.2 summarizes their uses.

special characters

The most common errors you will make as you learn to program will be syntax errors. Like any programming language, Java has its own syntax, and you need to write code that conforms

common errors

**Table 1.2** Special Characters

| *Character* | *Name* | *Description* |
|---|---|---|
| `{}` | Opening and closing braces | Denote a block to enclose statements. |
| `()` | Opening and closing parentheses | Used with methods. |
| `[]` | Opening and closing brackets | Denote an array. |
| `//` | Double slashes | Precede a comment line. |
| `" "` | Opening and closing quotation marks | Enclose a string (i.e., sequence of characters). |
| `;` | Semicolon | Mark the end of a statement. |

syntax rules

to the *syntax rules*. If your program violates a rule—for example, if the semicolon is missing, a brace is missing, a quotation mark is missing, or a word is misspelled—the Java compiler will report syntax errors. Try to compile the program with these errors and see what the compiler reports.

**Note**

You are probably wondering why the `main` method is defined this way and why `System.out.println(...)` is used to display a message on the console. *For the time being, simply accept that this is how things are done.* Your questions will be fully answered in subsequent chapters.

The program in Listing 1.1 displays one message. Once you understand the program, it is easy to extend it to display more messages. For example, you can rewrite the program to display three messages, as shown in Listing 1.2.

## Listing 1.2 WelcomeWithThreeMessages.java

class
main method
display message

```java
public class WelcomeWithThreeMessages {
  public static void main(String[] args) {
    System.out.println("Programming is fun!");
    System.out.println("Fundamentals First");
    System.out.println("Problem Driven");
  }
}
```

```
Programming is fun!
Fundamentals First
Problem Driven
```

Further, you can perform mathematical computations and display the result on the console. Listing 1.3 gives an example of evaluating $\frac{10.5 + 2 \times 3}{45 - 3.5}$.

## Listing 1.3 ComputeExpression.java

class
main method
compute expression

```java
public class ComputeExpression {
  public static void main(String[] args) {
    System.out.print("(10.5 + 2 * 3) / (45 - 3.5) = ");
    System.out.println((10.5 + 2 * 3) / (45 - 3.5));
  }
}
```

```
(10.5 + 2 * 3) / (45 - 3.5) = 0.39759036144578314
```

The print method in line 3

print vs. println

```
System.out.print("(10.5 + 2 * 3) / (45 – 3.5) = ");
```

is identical to the println method except that println moves to the beginning of the next line after displaying the string, but print does not advance to the next line when completed.

The multiplication operator in Java is *. As you can see, it is a straightforward process to translate an arithmetic expression to a Java expression. We will discuss Java expressions further in Chapter 2.

Check Point

**1.7.1** What is a keyword? List some Java keywords.

**1.7.2** Is Java case sensitive? What is the case for Java keywords?

**1.7.3** What is a comment? Is the comment ignored by the compiler? How do you denote a comment line and a comment paragraph?

**1.7.4** What is the statement to display a string on the console?

**1.7.5** Show the output of the following code:

```
public class Test {
  public static void main(String[] args) {
    System.out.println("3.5 * 4 / 2 – 2.5 is ");
    System.out.println(3.5 * 4 / 2 – 2.5);
  }
}
```

## 1.8 Creating, Compiling, and Executing a Java Program

*You save a Java program in a .java file and compile it into a .class file. The .class file is executed by the Java Virtual Machine (JVM).*

Key Point

You have to create your program and compile it before it can be executed. This process is repetitive, as shown in Figure 1.6. If your program has compile errors, you have to modify the program to fix them, then recompile it. If your program has runtime errors or does not produce the correct result, you have to modify the program, recompile it, and execute it again.

You can use any text editor or IDE to create and edit a Java source-code file. This section demonstrates how to create, compile, and run Java programs from a command window. Sections 1.11 and 1.12 will introduce developing Java programs using NetBeans and Eclipse. From the command window, you can use a text editor such as Notepad to create the Java source-code file, as shown in Figure 1.7.

command window

**Note**

The source file must end with the extension .java and must have the same exact name as the public class name. For example, the file for the source code in Listing 1.1 should be named **Welcome.java**, since the public class name is Welcome.

file name Welcome.java,

A Java compiler translates a Java source file into a Java bytecode file. The following command compiles **Welcome.java**:

compile

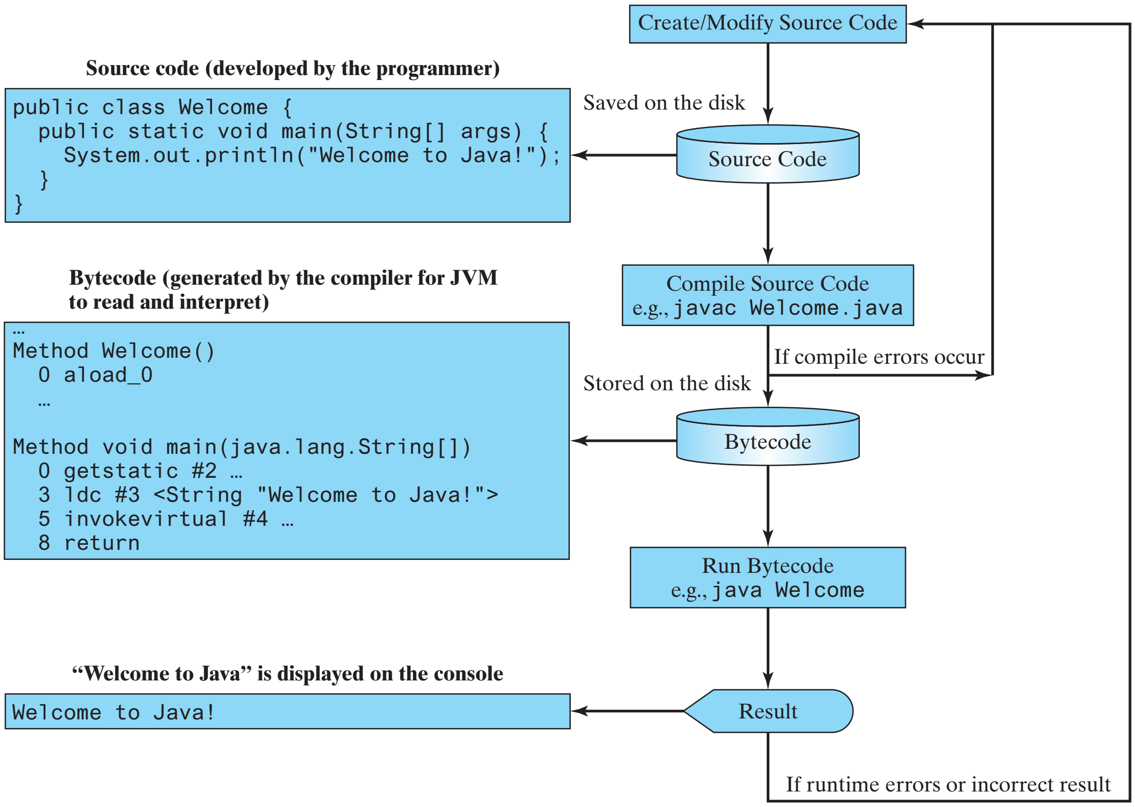
```
javac Welcome.java
```

**Note**

You must first install and configure the JDK before you can compile and run programs. See Supplement I.B, Installing and Configuring JDK 8, for how to install the JDK and set up the environment to compile and run Java programs. If you have trouble compiling and running programs, see Supplement I.C, Compiling and Running Java from the Command Window. This supplement also explains how to use basic DOS commands and how to use Windows Notepad to create and edit files. All the supplements are accessible from the Companion Website.

Supplement I.B

Supplement I.C



**Source code (developed by the programmer)**

```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

**Bytecode (generated by the compiler for JVM to read and interpret)**

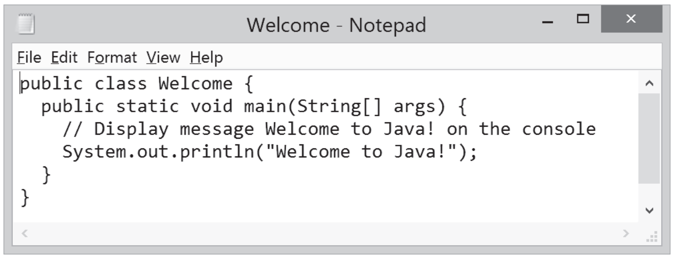
```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

**"Welcome to Java" is displayed on the console**

```
Welcome to Java!
```

**Figure 1.6** The Java program-development process consists of repeatedly creating/modifying source code, compiling, and executing programs.



```
public class Welcome {
  public static void main(String[] args) {
    // Display message Welcome to Java! on the console
    System.out.println("Welcome to Java!");
  }
}
```

**Figure 1.7** You can create a Java source file using Windows Notepad.

.class bytecode file

bytecode

Java Virtual Machine (JVM)

If there aren't any syntax errors, the *compiler* generates a bytecode file with a `.class` extension. Thus, the preceding command generates a file named **Welcome.class**, as shown in Figure 1.8a. The Java language is a high-level language, but Java bytecode is a low-level language. The *bytecode* is similar to machine instructions but is architecture neutral and can run on any platform that has a *Java Virtual Machine (JVM)*, as shown in Figure 1.8b. Rather than a physical machine, the virtual machine is a program that interprets Java bytecode. This is one of Java's primary advantages: *Java bytecode can run on a variety of hardware platforms and operating systems.* Java source code is compiled into Java bytecode, and Java bytecode is interpreted by the JVM. Your Java code may use the code in the Java library. The JVM executes your code along with the code in the library.

interpret bytecode

To execute a Java program is to run the program's bytecode. You can execute the bytecode on any platform with a JVM, which is an interpreter. It translates the individual instructions in the bytecode into the target machine language code one at a time, rather than the whole program as a single unit. Each step is executed immediately after it is translated.

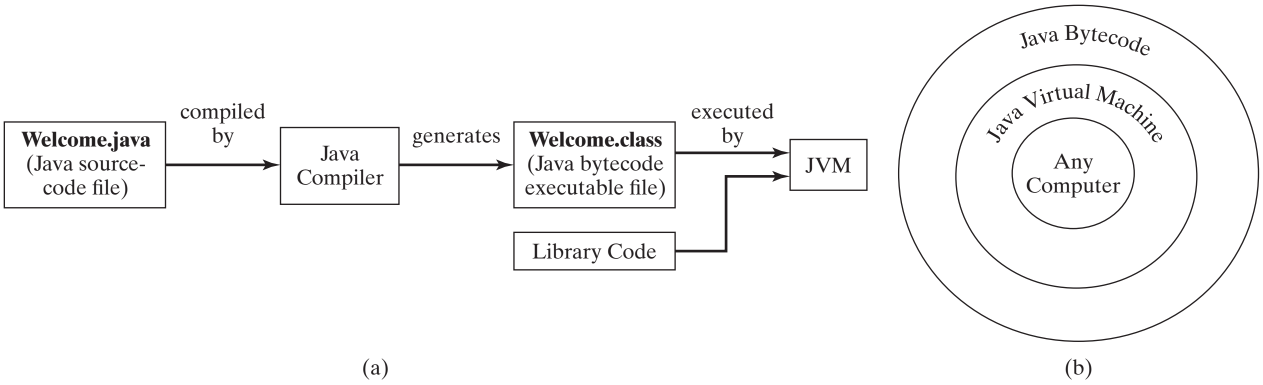


**FIGURE 1.8** (a) Java source code is translated into bytecode. (b) Java bytecode can be executed on any computer with a Java Virtual Machine.

The following command runs the bytecode for Listing 1.1:

run

```
java Welcome
```

Figure 1.9 shows the `javac` command for compiling **Welcome.java**. The compiler generates the **Welcome.class** file, and this file is executed using the `java` command.

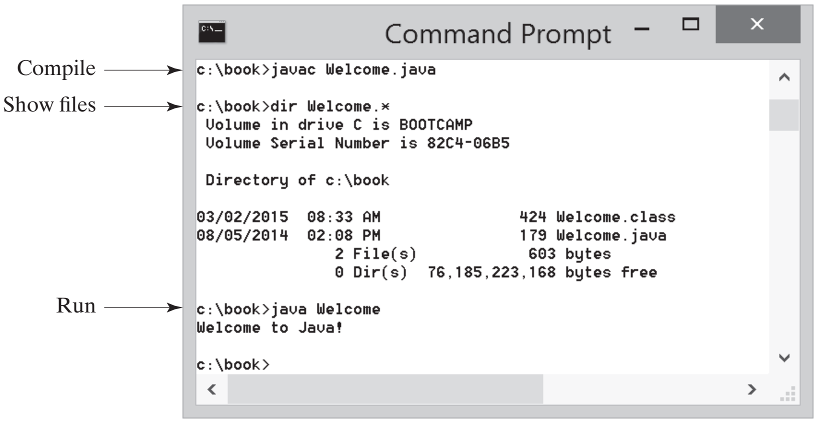
`javac` command

`java` command

### Note

For simplicity and consistency, all source-code and class files used in this book are placed under **c:\book** unless specified otherwise.

c:\book



```
c:\book>javac Welcome.java

c:\book>dir Welcome.*
 Volume in drive C is BOOTCAMP
 Volume Serial Number is 82C4-06B5

 Directory of c:\book

03/02/2015  08:33 AM               424 Welcome.class
08/05/2014  02:08 PM               179 Welcome.java
               2 File(s)            603 bytes
               0 Dir(s)  76,185,223,168 bytes free

c:\book>java Welcome
Welcome to Java!

c:\book>
```

**VideoNote**
Compile and run a Java program

**FIGURE 1.9** The output of Listing 1.1 displays the message "Welcome to Java!"

### Caution

Do not use the extension `.class` in the command line when executing the program. Use `java ClassName` to run the program. If you use `java ClassName.class` in the command line, the system will attempt to fetch `ClassName.class.class`.

java ClassName

### Tip

If you execute a class file that does not exist, a `NoClassDefFoundError` will occur. If you execute a class file that does not have a `main` method or you mistype the `main` method (e.g., by typing `Main` instead of `main`), a `NoSuchMethodError` will occur.

NoClassDefFoundError

NoSuchMethodError

**Note**

When executing a Java program, the JVM first loads the bytecode of the class to memory using a program called the *class loader*. If your program uses other classes, the class loader dynamically loads them just before they are needed. After a class is loaded, the JVM uses a program called the *bytecode verifier* to check the validity of the bytecode and to ensure that the bytecode does not violate Java's security restrictions. Java enforces strict security to make sure Java class files are not tampered with and do not harm your computer.

class loader

bytecode verifier

**Pedagogical Note**

Your instructor may require you to use packages for organizing programs. For example, you may place all programs in this chapter in a package named *chapter1*. For instructions on how to use packages, see Supplement I.F, Using Packages to Organize the Classes in the Text.

use package

Check Point

**1.8.1** What is the Java source filename extension, and what is the Java bytecode filename extension?
**1.8.2** What are the input and output of a Java compiler?
**1.8.3** What is the command to compile a Java program?
**1.8.4** What is the command to run a Java program?
**1.8.5** What is the JVM?
**1.8.6** Can Java run on any machine? What is needed to run Java on a computer?
**1.8.7** If a `NoClassDefFoundError` occurs when you run a program, what is the cause of the error?
**1.8.8** If a `NoSuchMethodError` occurs when you run a program, what is the cause of the error?

## 1.9 Programming Style and Documentation

Key Point

*Good programming style and proper documentation make a program easy to read and help programmers prevent errors.*

*Programming style* deals with what programs look like. A program can compile and run properly even if written on only one line, but writing it all on one line would be bad programming style because it would be hard to read. *Documentation* is the body of explanatory remarks and comments pertaining to a program. Programming style and documentation are as important as coding. Good programming style and appropriate documentation reduce the chance of errors and make programs easy to read. This section gives several guidelines. For more detailed guidelines, see Supplement I.D, Java Coding Style Guidelines, on the Companion Website.

programming style

documentation

### 1.9.1 Appropriate Comments and Comment Styles

Include a summary at the beginning of the program that explains what the program does, its key features, and any unique techniques it uses. In a long program, you should also include comments that introduce each major step and explain anything that is difficult to read. It is important to make comments concise so that they do not crowd the program or make it difficult to read.

In addition to line comments (beginning with `//`) and block comments (beginning with `/*`), Java supports comments of a special type, referred to as *javadoc comments*. javadoc comments begin with `/**` and end with `*/`. They can be extracted into an HTML file using the JDK's `javadoc` command. For more information, see Supplement III.Y, javadoc Comments, on the Companion Website.

javadoc comment

Use javadoc comments (`/** . . . */`) for commenting on an entire class or an entire method. These comments must precede the class or the method header in order to be extracted into a javadoc HTML file. For commenting on steps inside a method, use line comments (`//`). To see an example of a javadoc HTML file, check out liveexample.pearsoncmg.com/javadoc/Exercise1.html. Its corresponding Java code is shown in liveexample.pearsoncmg.com/javadoc/Exercise1.txt.

## 1.9.2 Proper Indentation and Spacing

A consistent indentation style makes programs clear and easy to read, debug, and maintain. *Indentation* is used to illustrate the structural relationships between a program's components or statements. Java can read the program even if all of the statements are on the same long line, but humans find it easier to read and maintain code that is aligned properly. Indent each subcomponent or statement at least *two* spaces more than the construct within which it is nested.

indent code

A single space should be added on both sides of a binary operator, as shown in (a), rather in (b).

```
System.out.println(3 + 4 * 4);
```

(a) Good style

```
System.out.println(3+4*4);
```

(b) Bad style

## 1.9.3 Block Styles

A *block* is a group of statements surrounded by braces. There are two popular styles, *next-line* style and *end-of-line* style, as shown below.

```
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("Block Styles");
  }
}
```

Next-line style

```
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```

End-of-line style

The next-line style aligns braces vertically and makes programs easy to read, whereas the end-of-line style saves space and may help avoid some subtle programming errors. Both are acceptable block styles. The choice depends on personal or organizational preference. You should use a block style consistently—mixing styles is not recommended. This book uses the *end-of-line* style to be consistent with the Java API source code.

Check Point

**1.9.1** Reformat the following program according to the programming style and documentation guidelines. Use the end-of-line brace style.

```
public class Test
{
  // Main method
  public static void main(String[] args) {
  /** Display output */
  System.out.println("Welcome to Java");
  }
}
```

## 1.10 Programming Errors

Key Point

*Programming errors can be categorized into three types: syntax errors, runtime errors, and logic errors.*

### 1.10.1 Syntax Errors

syntax errors

compile errors

Errors that are detected by the compiler are called *syntax errors* or *compile errors*. Syntax errors result from errors in code construction, such as mistyping a keyword, omitting some necessary punctuation, or using an opening brace without a corresponding closing brace. These errors are usually easy to detect because the compiler tells you where they are and what caused them. For example, the program in Listing 1.4 has a syntax error, as shown in Figure 1.10.

**Listing 1.4** ShowSyntaxErrors.java

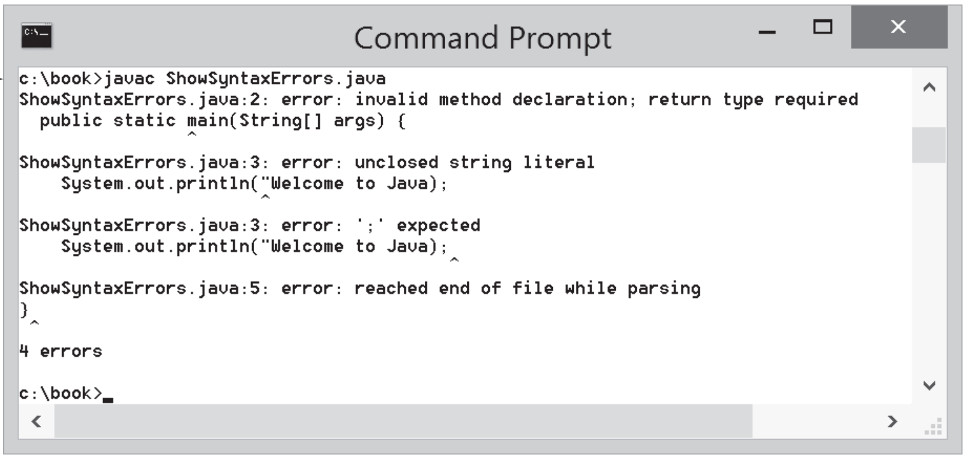
```
1  public class ShowSyntaxErrors {
2    public static main(String[] args) {
3      System.out.println("Welcome to Java);
4    }
5  }
```

Four errors are reported, but the program actually has two errors:

- The keyword `void` is missing before `main` in line 2.
- The string `Welcome to Java` should be closed with a closing quotation mark in line 3.

Since a single error will often display many lines of compile errors, it is a good practice to fix errors from the top line and work downward. Fixing errors that occur earlier in the program may also fix additional errors that occur later.

Compile



```
c:\book>javac ShowSyntaxErrors.java
ShowSyntaxErrors.java:2: error: invalid method declaration; return type required
  public static main(String[] args) {
                ^
ShowSyntaxErrors.java:3: error: unclosed string literal
    System.out.println("Welcome to Java);
                       ^
ShowSyntaxErrors.java:3: error: ';' expected
    System.out.println("Welcome to Java);
                                         ^
ShowSyntaxErrors.java:5: error: reached end of file while parsing
}
 ^
4 errors

c:\book>_
```

**Figure 1.10** The compiler reports syntax errors.

fix syntax errors

**Tip**

If you don't know how to correct an error, compare your program closely, character by character, with similar examples in the text. In the first few weeks of this course, you will probably spend a lot of time fixing syntax errors. Soon you will be familiar with Java syntax, and can quickly fix syntax errors.

### 1.10.2 Runtime Errors

runtime errors

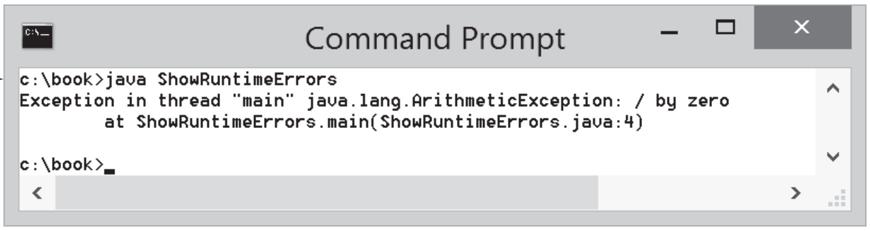
*Runtime errors* are errors that cause a program to terminate abnormally. They occur while a program is running if the environment detects an operation that is impossible to carry out. Input mistakes typically cause runtime errors. An *input error* occurs when the program is waiting for the user to enter a value, but the user enters a value that the program cannot handle. For instance, if the program expects to read in a number, but instead the user enters a string, this causes data-type errors to occur in the program.

Another example of runtime errors is division by zero. This happens when the divisor is zero for integer divisions. For instance, the program in Listing 1.5 would cause a runtime error, as shown in Figure 1.11.

**LISTING 1.5** ShowRuntimeErrors.java

```java
public class ShowRuntimeErrors {
  public static void main(String[] args) {
    System.out.println(1 / 0);
  }
}
```

runtime error



Run

```
c:\book>java ShowRuntimeErrors
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ShowRuntimeErrors.main(ShowRuntimeErrors.java:4)

c:\book>_
```

**FIGURE 1.11** The runtime error causes the program to terminate abnormally.

### 1.10.3 Logic Errors

logic errors

*Logic errors* occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons. For example, suppose you wrote the program in Listing 1.6 to convert Celsius `35` degrees to a Fahrenheit degree:

**LISTING 1.6** ShowLogicErrors.java

```java
public class ShowLogicErrors {
  public static void main(String[] args) {
    System.out.print("Celsius 35 is Fahrenheit degree ");
    System.out.println((9 / 5) * 35 + 32);
  }
}
```

```
Celsius 35 is Fahrenheit degree 67
```

You will get Fahrenheit `67` degrees, which is wrong. It should be `95.0`. In Java, the division for integers is the quotient—the fractional part is truncated—so in Java `9 / 5` is `1`. To get the correct result, you need to use `9.0 / 5`, which results in `1.8`.

In general, syntax errors are easy to find and easy to correct because the compiler gives indications as to where the errors came from and why they are wrong. Runtime errors are not difficult to find, either, since the reasons and locations for the errors are displayed on the console when the program aborts. Finding logic errors, on the other hand, can be very challenging. In the upcoming chapters, you will learn the techniques of tracing programs and finding logic errors.

## 1.10.4 Common Errors

Missing a closing brace, missing a semicolon, missing quotation marks for strings, and misspelling names are common errors for new programmers.

### Common Error 1: Missing Braces

The braces are used to denote a block in the program. Each opening brace must be matched by a closing brace. A common error is missing the closing brace. To avoid this error, type a closing brace whenever an opening brace is typed, as shown in the following example:

```
public class Welcome {

}
```

← Type this closing brace right away to match the opening brace.

If you use an IDE such as NetBeans and Eclipse, the IDE automatically inserts a closing brace for each opening brace typed.

### Common Error 2: Missing Semicolons

Each statement ends with a statement terminator (;). Often, a new programmer forgets to place a statement terminator for the last statement in a block, as shown in the following example:

```
public static void main(String[] args) {
  System.out.println("Programming is fun!");
  System.out.println("Fundamentals First");
  System.out.println("Problem Driven")
}
```

↑ Missing a semicolon

### Common Error 3: Missing Quotation Marks

A string must be placed inside the quotation marks. Often, a new programmer forgets to place a quotation mark at the end of a string, as shown in the following example:

```
System.out.println("Problem Driven);
```

↑ Missing a quotation mark

If you use an IDE such as NetBeans and Eclipse, the IDE automatically inserts a closing quotation mark for each opening quotation mark typed.

### Common Error 4: Misspelling Names

Java is case sensitive. Misspelling names is a common error for new programmers. For example, the word `main` is misspelled as `Main` and `String` is misspelled as `string` in the following code:

```
public class Test {
  public static void Main(string[] args) {
    System.out.println((10.5 + 2 * 3) / (45 – 3.5));
  }
}
```

Check Point

**1.10.1** What are syntax errors (compile errors), runtime errors, and logic errors?

**1.10.2** Give examples of syntax errors, runtime errors, and logic errors.

**1.10.3** If you forget to put a closing quotation mark on a string, what kind error of will be raised?

**1.10.4** If your program needs to read integers, but the user entered strings, an error would occur when running this program. What kind of error is this?

**1.10.5** Suppose you write a program for computing the perimeter of a rectangle and you mistakenly write your program so it computes the area of a rectangle. What kind of error is this?

**1.10.6** Identify and fix the errors in the following code:

```
public class Welcome {
  public void Main(String[] args) {
    System.out.println('Welcome to Java!);
  }
)
```

**Note**

Section 1.8 introduced developing programs from the command line. Many of our readers also use an IDE. The following two sections introduce two most popular Java IDEs: NetBeans and Eclipse. These two sections may be skipped.

## 1.11 Developing Java Programs Using NetBeans

*You can edit, compile, run, and debug Java Programs using NetBeans.*

Key Point

VideoNote

NetBeans brief tutorial

NetBeans and Eclipse are two free popular integrated development environments for developing Java programs. They are easy to learn if you follow simple instructions. We recommend that you use either one for developing Java programs. This section gives the essential instructions to guide new users to create a project, create a class, compile, and run a class in NetBeans. The use of Eclipse will be introduced in the next section. For instructions on downloading and installing latest version of NetBeans, see Supplement II.B.

### 1.11.1 Creating a Java Project

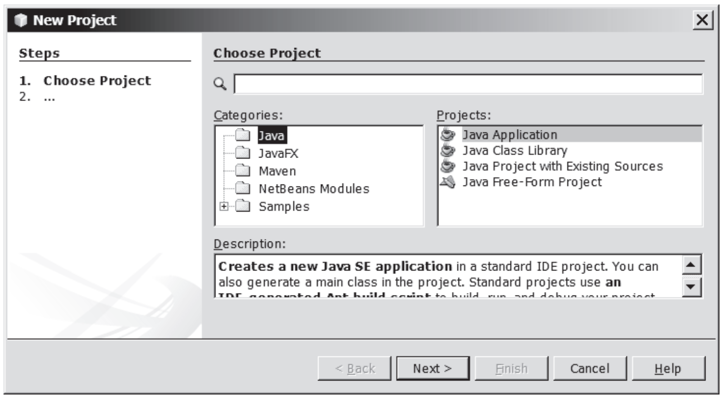
Before you can create Java programs, you need to first create a project. A project is like a folder to hold Java programs and all supporting files. You need to create a project only once. Here are the steps to create a Java project:

1. Choose *File*, *New Project* to display the New Project dialog box, as shown in Figure 1.12.

2. Select Java in the Categories section and Java Application in the Projects section, and then click *Next* to display the New Java Application dialog box, as shown in Figure 1.13.

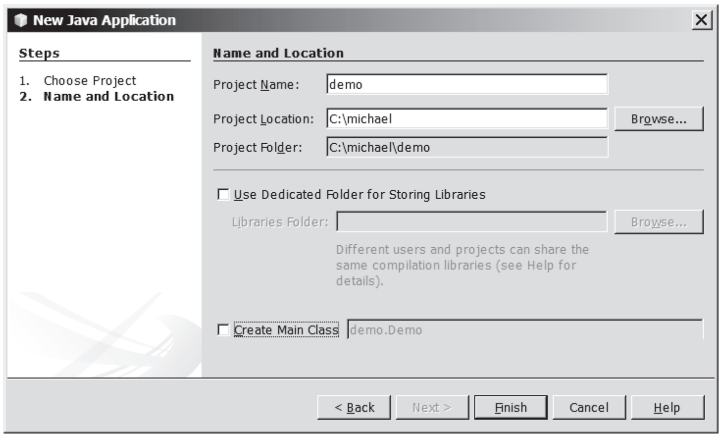
3. Type `demo` in the Project Name field and `c:\michael` in Project Location field. Uncheck *Use Dedicated Folder for Storing Libraries* and uncheck *Create Main Class*.

4. Click *Finish* to create the project, as shown in Figure 1.14.



**Creates a new Java SE application** in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

**FIGURE 1.12** The New Project dialog is used to create a new project and specify a project type. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.



New Java Application

Steps

1. Choose Project
2. **Name and Location**

**Name and Location**

Project Name: demo

Project Location: C:\michael

Project Folder: C:\michael\demo

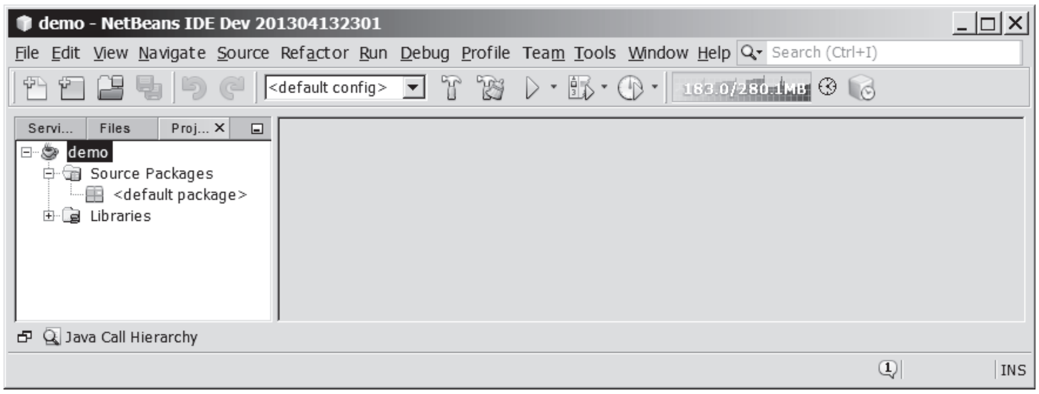
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☐ Create Main Class demo.Demo

**FIGURE 1.13** The New Java Application dialog is for specifying a project name and location. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.



demo - NetBeans IDE Dev 201304132301

<default config>

183.0/280.1MB

demo
Source Packages
<default package>
Libraries

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Servi... Files Proj...
Java Call Hierarchy
INS

**FIGURE 1.14** A New Java project named demo is created. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

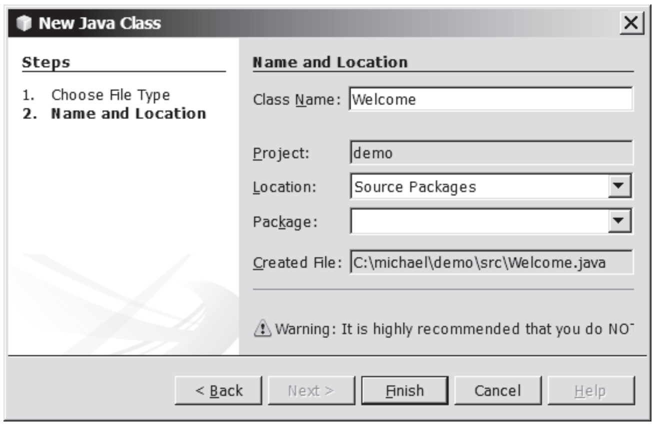
### 1.11.2 Creating a Java Class

After a project is created, you can create Java programs in the project using the following steps:

1. Right-click the demo node in the project pane to display a context menu. Choose *New*, *Java Class* to display the New Java Class dialog box, as shown in Figure 1.15.
2. Type `Welcome` in the Class Name field and select the Source Packages in the Location field. Leave the Package field blank. This will create a class in the default package.
3. Click *Finish* to create the Welcome class. The source-code file **Welcome.java** is placed under the <default package> node.
4. Modify the code in the Welcome class to match Listing 1.1 in the text, as shown in Figure 1.16.

### 1.11.3 Compiling and Running a Class

To run **Welcome.java**, right-click **Welcome.java** to display a context menu and choose *Run File*, or simply press Shift + F6. The output is displayed in the Output pane, as shown in Figure 1.16. The *Run File* command automatically compiles the program if the program has been changed.



Class Name: Welcome

Project: demo

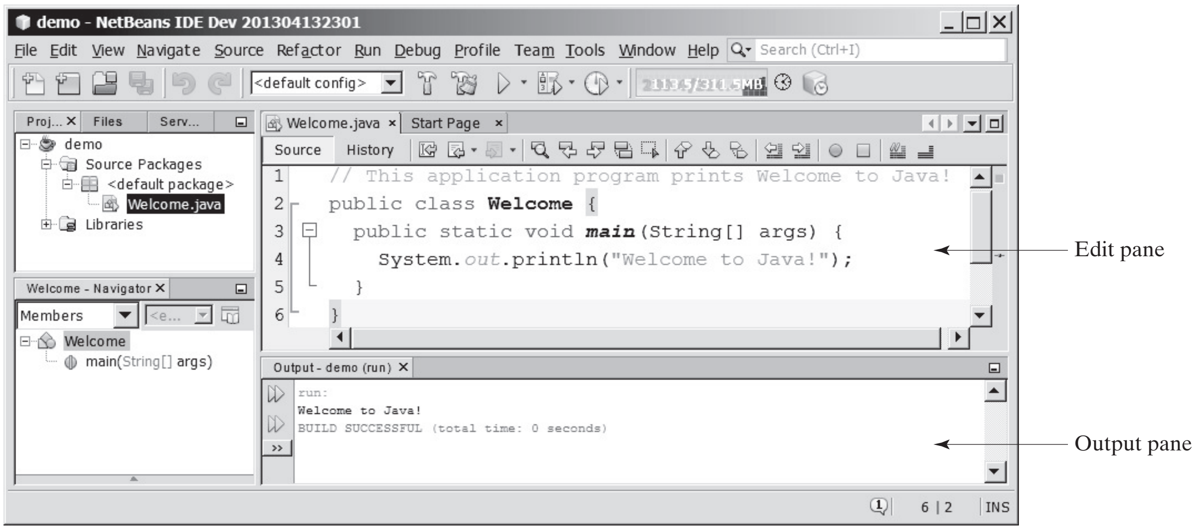
Location: Source Packages

Package:

Created File: C:\michael\demo\src\Welcome.java

Warning: It is highly recommended that you do NO

**FIGURE 1.15** The New Java Class dialog box is used to create a new Java class. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.



```
// This application program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

```
run:
Welcome to Java!
BUILD SUCCESSFUL (total time: 0 seconds)
```

**FIGURE 1.16** You can edit a program and run it in NetBeans. *Source*: Copyright © 1995–2016 Oracle and/or its affiliates. All rights reserved. Used with permission.

## 1.12 Developing Java Programs Using Eclipse

*You can edit, compile, run, and debug Java Programs using Eclipse.*

Key Point

The preceding section introduced developing Java programs using NetBeans. You can also use Eclipse to develop Java programs. This section gives the essential instructions to guide new users to create a project, create a class, and compile/run a class in Eclipse. For instructions on downloading and installing latest version of Eclipse, see Supplement II.D.

### 1.12.1 Creating a Java Project

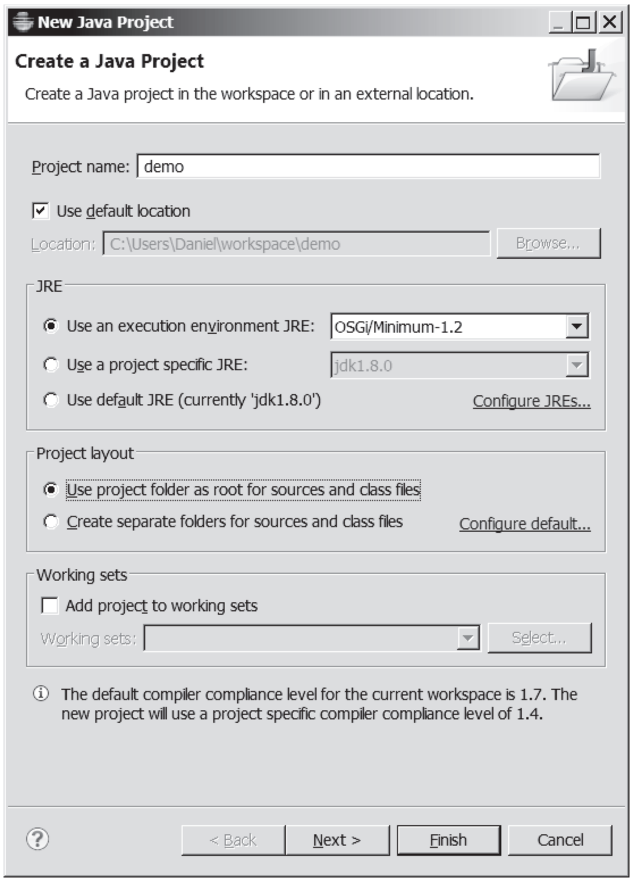
VideoNote

Eclipse brief tutorial

Before creating Java programs in Eclipse, you need to first create a project to hold all files. Here are the steps to create a Java project in Eclipse:

1. Choose *File*, *New*, *Java Project* to display the New Project wizard, as shown in Figure 1.17.

2. Type `demo` in the Project name field. As you type, the Location field is automatically set by default. You may customize the location for your project.


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: demo

☑ Use default location

Location: C:\Users\Daniel\workspace\demo

JRE

● Use an execution environment JRE: OSGi/Minimum-1.2

○ Use a project specific JRE: jdk1.8.0

○ Use default JRE (currently 'jdk1.8.0')

Project layout

● Use project folder as root for sources and class files

○ Create separate folders for sources and class files

Working sets

□ Add project to working sets

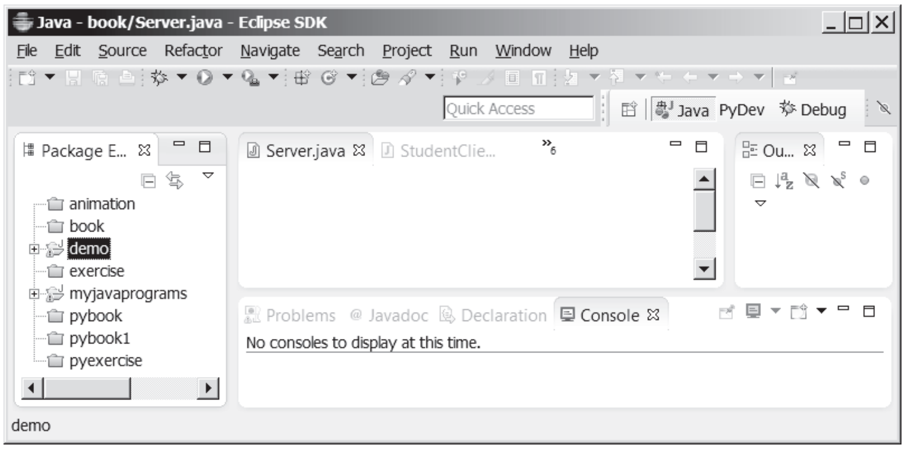
Working sets:

The default compiler compliance level for the current workspace is 1.7. The new project will use a project specific compiler compliance level of 1.4.

**Figure 1.17** The New Java Project dialog is for specifying a project name and the properties. *Source*: Eclipse Foundation, Inc.

3. Make sure you selected the options *Use project folder as root for sources and class files* so the .java and .class files are in the same folder for easy access.

4. Click *Finish* to create the project, as shown in Figure 1.18.


Java - book/Server.java - Eclipse SDK

No consoles to display at this time.

**Figure 1.18** A New Java project named demo is created. *Source*: Eclipse Foundation, Inc.

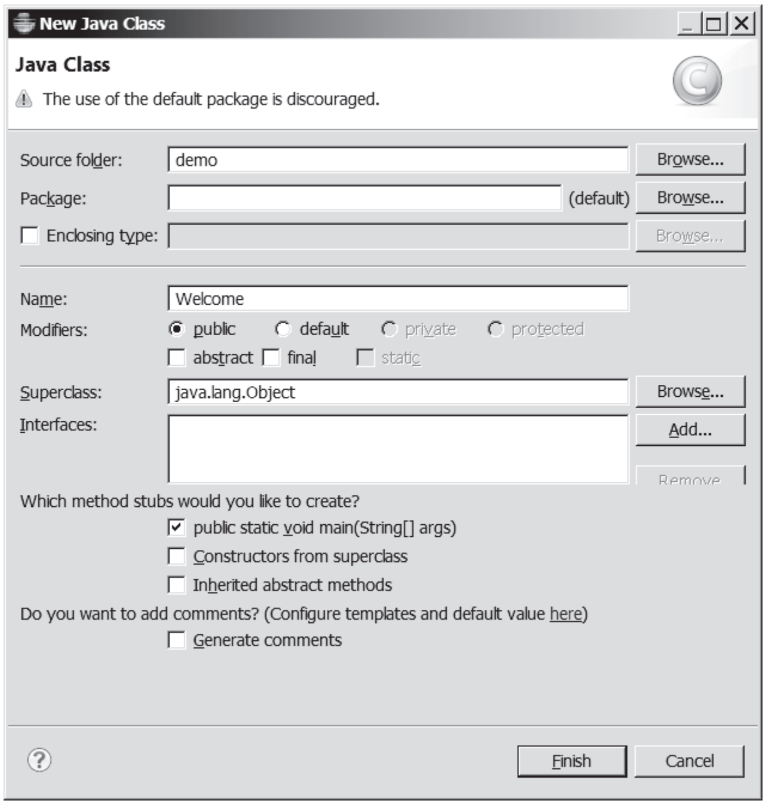
## 1.12.2 Creating a Java Class

After a project is created, you can create Java programs in the project using the following steps:

1. Choose *File*, *New*, *Class* to display the New Java Class wizard.
2. Type `Welcome` in the Name field.
3. Check the option *public static void main(String[] args)*.
4. Click *Finish* to generate the template for the source code **Welcome.java**, as shown in Figure 1.19.

## 1.12.3 Compiling and Running a Class

To run the program, right-click the class in the project to display a context menu. Choose *Run*, *Java Application* in the context menu to run the class. The output is displayed in the Console pane, as shown in Figure 1.20. The *Run* command automatically compiles the program if the program has been changed.



New Java Class

Java Class

The use of the default package is discouraged.

Source folder: demo

Package: (default)

□ Enclosing type:

Name: Welcome

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

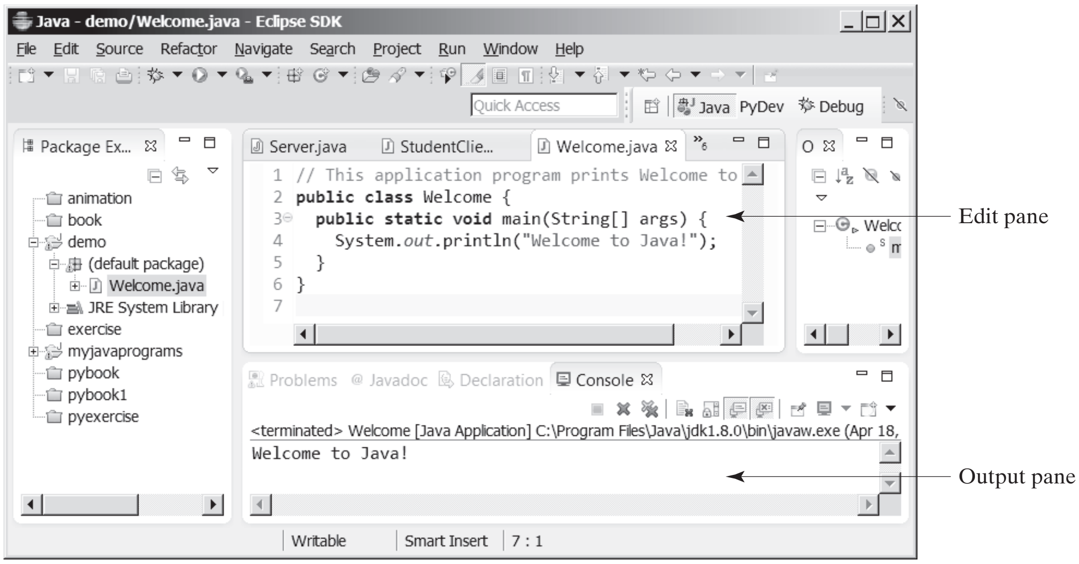
□ Constructors from superclass

□ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

**FIGURE 1.19** The New Java Class dialog box is used to create a new Java class. *Source*: Eclipse Foundation, Inc.



```
// This application program prints Welcome to
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

```
<terminated> Welcome [Java Application] C:\Program Files\Java\jdk1.8.0\bin\javaw.exe (Apr 18,
Welcome to Java!
```

**Figure 1.20** You can edit a program and run it in Eclipse. *Source*: Eclipse Foundation, Inc.

## Key Terms

| | |
|---|---|
| Application Program Interface (API) | 11 |
| assembler | 7 |
| assembly language | 7 |
| bit | 3 |
| block | 13 |
| block comment | 13 |
| bus | 2 |
| byte | 3 |
| bytecode | 16 |
| bytecode verifier | 18 |
| cable modem | 6 |
| central processing unit (CPU) | 3 |
| class loader | 18 |
| comment | 13 |
| compiler | 8 |
| console | 12 |
| dot pitch | 6 |
| DSL (digital subscriber line) | 6 |
| encoding scheme | 3 |
| hardware | 2 |
| high-level language | 8 |
| integrated development environment (IDE) | 12 |
| interpreter | 8 |
| `java` command | 17 |
| Java Development Toolkit (JDK) | 12 |
| Java language specification | 11 |
| Java Runtime Environment (JRE) | 12 |
| Java Virtual Machine (JVM) | 16 |
| `javac` command | 17 |
| keyword (or reserved word) | 13 |
| library | 11 |
| line comment | 13 |
| logic error | 21 |
| low-level language | 8 |
| machine language | 7 |
| `main` method | 13 |
| memory | 4 |
| dial-up modem | 6 |
| motherboard | 3 |
| network interface card (NIC) | 6 |
| operating system (OS) | 9 |
| pixel | 6 |
| program | 2 |
| programming | 2 |
| runtime error | 21 |
| screen resolution | 6 |
| software | 2 |
| source code | 8 |
| source program | 8 |
| statement | 8 |
| statement terminator | 13 |
| storage devices | 4 |
| syntax error | 20 |

Supplement I.A

### Note

The above terms are defined in this chapter. Supplement I.A, Glossary, lists all the key terms and descriptions in the book, organized by chapters.

## Chapter Summary

1. A computer is an electronic device that stores and processes data.
2. A computer includes both *hardware* and *software*.
3. Hardware is the physical aspect of the computer that can be touched.
4. Computer *programs*, known as *software*, are the invisible instructions that control the hardware and make it perform tasks.
5. Computer *programming* is the writing of instructions (i.e., code) for computers to perform.
6. The *central processing unit (CPU)* is a computer's brain. It retrieves instructions from *memory* and executes them.
7. Computers use zeros and ones because digital devices have two stable states, referred to by convention as zero and one.
8. A *bit* is a binary digit 0 or 1.
9. A *byte* is a sequence of 8 bits.
10. A kilobyte is about 1,000 bytes, a megabyte about 1 million bytes, a gigabyte about 1 billion bytes, and a terabyte about 1,000 gigabytes.
11. Memory stores data and program instructions for the CPU to execute.
12. A memory unit is an ordered sequence of bytes.
13. Memory is volatile, because information is lost when the power is turned off.
14. Programs and data are permanently stored on *storage devices* and are moved to memory when the computer actually uses them.
15. The *machine language* is a set of primitive instructions built into every computer.
16. *Assembly language* is a *low-level programming language* in which a mnemonic is used to represent each machine-language instruction.
17. *High-level languages* are English-like and easy to learn and program.
18. A program written in a high-level language is called a *source program.*
19. A *compiler* is a software program that translates the source program into a *machine-language program.*
20. The *operating system (OS)* is a program that manages and controls a computer's activities.
21. Java is platform independent, meaning you can write a program once and run it on any computer.
22. The Java source file name must match the public class name in the program. Java source-code files must end with the `.java` extension.
23. Every class is compiled into a separate bytecode file that has the same name as the class and ends with the `.class` extension.
24. To compile a Java source-code file from the command line, use the `javac` command.

25. To run a Java class from the command line, use the `java` command.

26. Every Java program is a set of class definitions. The keyword `class` introduces a class definition. The contents of the class are included in a *block*.

27. A block begins with an opening brace (`{`) and ends with a closing brace (`}`).

28. Methods are contained in a class. To run a Java program, the program must have a `main` method. The `main` method is the entry point where the program starts when it is executed.

29. Every *statement* in Java ends with a semicolon (`;`), known as the *statement terminator*.

30. *Reserved words,* or *keywords,* have a specific meaning to the compiler and cannot be used for other purposes in the program.

31. In Java, comments are preceded by two slashes (`//`) on a line, called a *line comment,* or enclosed between `/*` and `*/` on one or several lines, called a *block comment* or *paragraph comment*. Comments are ignored by the compiler.

32. Java source programs are case sensitive.

33. Programming errors can be categorized into three types: *syntax errors*, *runtime errors*, and *logic errors*. Errors reported by a compiler are called syntax errors or *compile errors.* Runtime errors are errors that cause a program to terminate abnormally. Logic errors occur when a program does not perform the way it was intended to.

## Quiz

Answer the quiz for this chapter at www.pearsonhighered.com/liang. Choose this book and click Companion Website to select Quiz.

MyProgrammingLab™

## Programming Exercises

### Pedagogical Note

We cannot stress enough the importance of learning programming through exercises. For this reason, the book provides a large number of programming exercises at various levels of difficulty. The problems cover many application areas, including math, science, business, financial, gaming, animation, and multimedia. Solutions to most even-numbered programming exercises are on the Companion Website. Solutions to most odd-numbered programming exercises are on the Instructor Resource Website. The level of difficulty is rated easy (no star), moderate (*), hard (**), or challenging (***).

level of difficulty

1.1 (*Display three messages*) Write a program that displays `Welcome to Java`, `Welcome to Computer Science`, and `Programming is fun`.

1.2 (*Display five messages*) Write a program that displays `Welcome to Java` five times.

*1.3 (*Display a pattern*) Write a program that displays the following pattern:

```
    J     A     V     V     A
    J    A A     V   V     A A
J   J   AAAAA     V V     AAAAA
 J J   A     A     V     A     A
```

**1.4** (*Print a table*) Write a program that displays the following table:

```
a       a^2     a^3
1       1       1
2       4       8
3       9       27
4       16      64
```

**1.5** (*Compute expressions*) Write a program that displays the result of

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}.$$

**1.6** (*Summation of a series*) Write a program that displays the result of

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9.$$

**1.7** (*Approximate $\pi$*) $\pi$ can be computed using the following formula:

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \ldots\right)$$

Write a program that displays the result of $4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11}\right)$ and $4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13}\right)$. Use 1.0 instead of 1 in your program.

**1.8** (*Area and perimeter of a circle*) Write a program that displays the area and perimeter of a circle that has a radius of 5.5 using the following formulas:

$$perimeter = 2 \times radius \times \pi$$
$$area = radius \times radius \times \pi$$

**1.9** (*Area and perimeter of a rectangle*) Write a program that displays the area and perimeter of a rectangle with a width of 4.5 and a height of 7.9 using the following formula:

$$area = width \times height$$

**1.10** (*Average speed in miles*) Assume that a runner runs 14 kilometers in 45 minutes and 30 seconds. Write a program that displays the average speed in miles per hour. (Note 1 mile is equal to 1.6 kilometers.)

***1.11** (*Population projection*) The U.S. Census Bureau projects population based on the following assumptions:

- One birth every 7 seconds
- One death every 13 seconds
- One new immigrant every 45 seconds

Write a program to display the population for each of the next five years. Assume that the current population is 312,032,486, and one year has 365 days. *Hint*: In Java, if two integers perform division, the result is an integer. The fractional part is truncated. For example, 5 / 4 is 1 (not 1.25) and 10 / 4 is 2 (not 2.5). To get an accurate result with the fractional part, one of the values involved in the division must be a number with a decimal point. For example, 5.0 / 4 is 1.25 and 10 / 4.0 is 2.5.

**1.12** (*Average speed in kilometers*) Assume that a runner runs 24 miles in 1 hour, 40 minutes, and 35 seconds. Write a program that displays the average speed in kilometers per hour. (Note 1 mile is equal to 1.6 kilometers.)

***1.13** (*Algebra: solve* 2 × 2 *linear equations*) You can use Cramer's rule to solve the following 2 × 2 system of linear equation provided that *ad* – *bc* is not 0:

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \qquad x = \frac{ed - bf}{ad - bc} \qquad y = \frac{af - ec}{ad - bc}$$

Write a program that solves the following equation and displays the value for *x* and *y*: (Hint: replace the symbols in the formula with numbers to compute *x* and *y*. This exercise can be done in Chapter 1 without using materials in later chapters.)

$$\begin{aligned} 3.4x + 50.2y &= 44.5 \\ 2.1x + .55y &= 5.9 \end{aligned}$$

**Note**

More than 200 additional programming exercises with solutions are provided to the instructors on the Instructor Resource Website.