GLOBAL EDITION

Introduction to Java™ Programming

Brief Version

ELEVENTH EDITION

Y. Daniel Liang

Pearson

Digital Resources for Students

Your new textbook provides 12-month access to digital resources that may include VideoNotes (step-by-step video tutorials on programming concepts), source code, web chapters, quizzes, and more. Refer to the preface in the textbook for a detailed list of resources.

Follow the instructions below to register for the Companion Website for Daniel Liang's *Introduction to Java™ Programming, Brief Version,* Eleventh Edition, Global Edition.

1. Go to www.pearsonglobaleditions.com/liang
2. Enter the title of your textbook or browse by author name.
3. Click Companion Website.
4. Click Register and follow the on-screen instructions to create a login name and password.

ISSLJB-FLUFF-ALIEN-PAREU-BEGUN-OOSSE

Use the login name and password you created during registration to start using the digital resources that accompany your textbook.

IMPORTANT:

This prepaid subscription does not include access to MyProgrammingLab, which is available at www.myprogramminglab.com for purchase.

This access code can only be used once. This subscription is valid for 12 months upon activation and is not transferable. If the access code has already been revealed it may no longer be valid.

For technical support go to https://support.pearson.com/getsupport

INTRODUCTION TO JAVA™ PROGRAMMING

BRIEF VERSION

Eleventh Edition

Global 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*
Assistant Acquisitions Editor, Global Edition: *Aditee Agarwal*
Assistant Project Editor, Global Edition: *Shaoni Mukherjee*
Manager, Media Production, Global Edition: *Vikram Kumar*
Senior Manufacturing Controller, Production, Global Edition: *Jerry Kataria*
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: *Lumina Datamatics*
Cover Image: *Eduardo Rocha/ shutterstock.com*
Full-Service Project Management: *Shylaja Gattupalli*, SPi Global

Java™ and Netbeans™ screenshots ©2017 by Oracle Corporation, all rights reserved. Reprinted with permission. 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.

Pearson Education Limited
KAO Two
KAO Park
Harlow
CM17 9NA
United Kingdom

and Associated Companies throughout the world

Visit us on the World Wide Web at: www.pearsonglobaleditions.com

© Pearson Education Limited 2019

The rights of Y. Daniel Liang to be identified as the author of this work have been asserted by him in accordance with the Copyright, Designs and Patents Act 1988.

Authorized adaptation from the United States edition, entitled Introduction to Java Programming, Brief Version, 11th Edition, ISBN 978-0-13-461103-7 by Y. Daniel Liang, published by Pearson Education © 2018.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the publisher or a license permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, Saffron House, 6–10 Kirby Street, London EC1N 8TS.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

10 9 8 7 6 5 4 3 2 1

Typeset by SPi Global.
Printed and bound by Vivar in Malaysia

ISBN 10: 1-292-22203-4
ISBN 13: 978-1-292-22203-5

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.

brief version

comprehensive version

This book is widely used in the introductory programming courses in the universities around the world. The book is a *brief version* of Introduction to Java Programming and Data Structures, *Comprehensive Version*, Eleventh Edition, Global Edition. This version is designed for an introductory programming course, commonly known as CS1. It contains the first eighteen chapters in the comprehensive version and covers fundamentals of programming, object-oriented programming, GUI programming, exception handling, I/O, and recursion. The comprehensive version has additional twenty-six chapters that cover data structures, algorithms, concurrency, parallel programming, networking, internationalization, advanced GUI, database, and Web programming.

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 on and suggestions for improving the book, please email me.

Sincerely,

Y. Daniel Liang
y.daniel.liang@gmail.com

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 Center.

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 Center.

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:

- 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.
- 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 in the Companion Website. These exercises are not printed in the text.

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 representative problems to give the reader an overview of what to expect from 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 Center. 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.

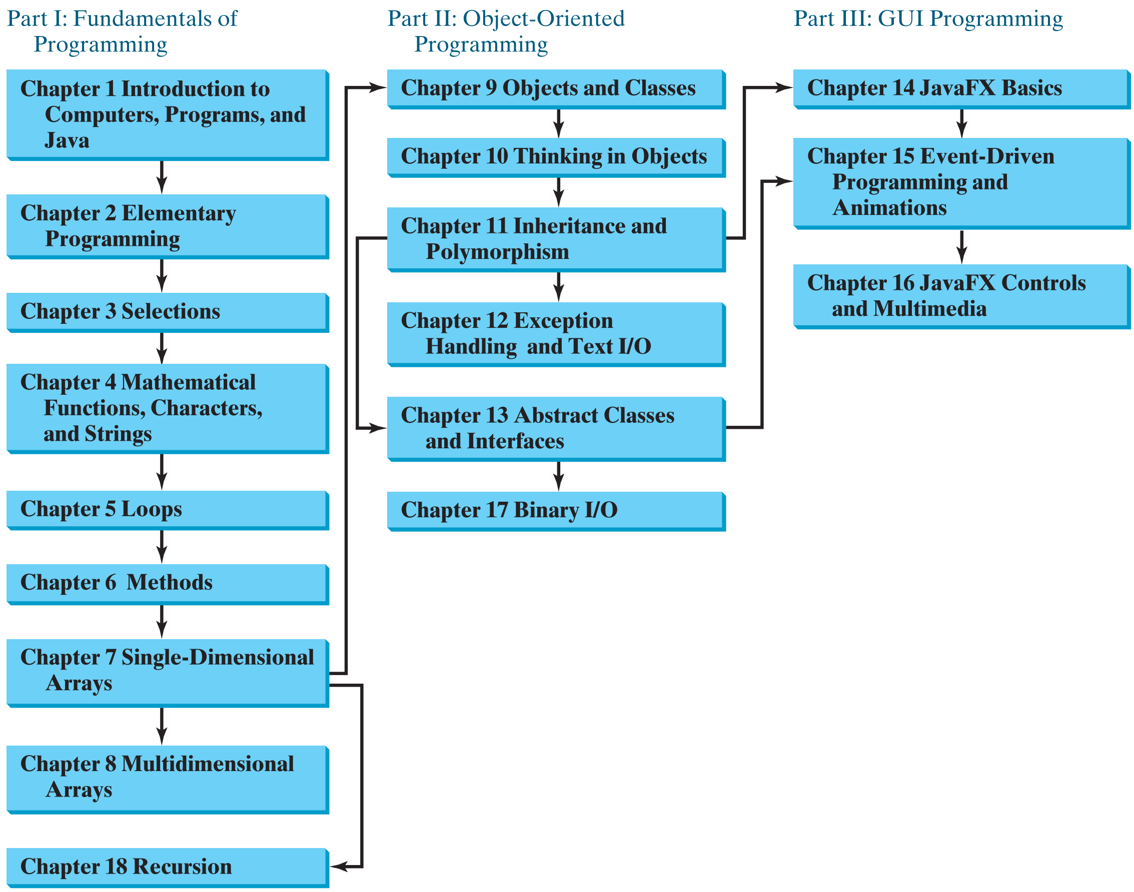
Tip
Teaches good programming style and practice.

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, and recursion to be covered earlier or later. The diagram on the next page shows the chapter dependencies.



Organization of the Book

The chapters in this brief version can be grouped into three parts that, taken together, form a solid introduction to Java 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, 18)

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)

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.

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 at www.pearsonglobaleditions.com/Liang.

IDE tutorials

Student Resources

The Companion Website (www.pearsonglobaleditions.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

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 Resources

The Companion Website, accessible from www.pearsonglobaleditions.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 a 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 Center. Please understand that these are for instructors only. Such requests will not be answered.

MyProgrammingLab™

Online Practice and Assessment with 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.pearsonglobaleditions.com/Liang.

Algorithm Animations

Animation

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, bug reports, and praise.

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.

Acknowledgments for the Global Edition

Pearson would like to thank and acknowledge Yvan Maillot (Univresite Haute-Alsace) and Steven Yuwono (National University of Singapore) for contributing to this Global Edition, and Arif Ahmed (National Institute of Technology, Silchar), Annette Bieniusa (University of Kaiserslautern), Shaligram Prajapat (Devi Ahilya Vishwavidyalaya, Indore), and Ram Gopal Raj (University of Malaya) for reviewing this Global Edition.

CONTENTS

This page intentionally left blank

VideoNotes

Locations of VideoNotes

www.pearsonglobaleditions.com/Liang

VideoNote

Animations

This page intentionally left blank

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,

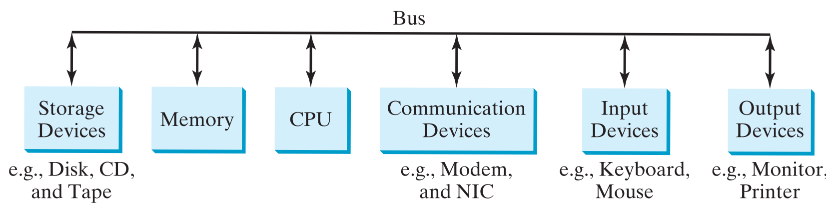


FIGURE 1.1 A computer consists of a CPU, memory, storage devices, input devices, output devices, and communication devices.

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)*.

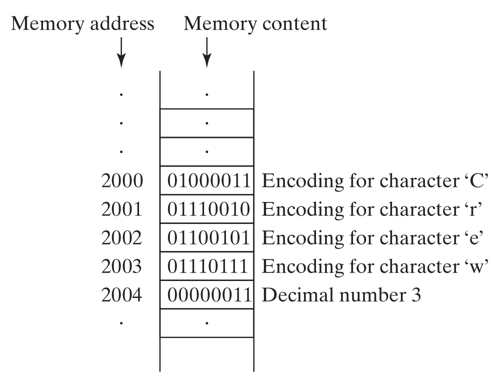


FIGURE 1.2 Memory stores data and program instructions in uniquely addressed memory locations.

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.pearsonglobaleditions.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.

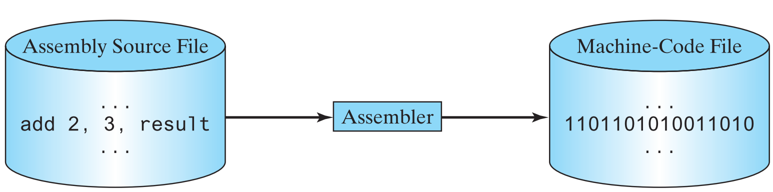


FIGURE 1.3 An assembler translates assembly-language instructions into machine code.

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.