Cengage

JAVA™ Programming

Tenth Edition

Joyce Farrell

Java Programming

Tenth Edition

Joyce Farrell

Cengage

Australia • Brazil • Canada • Mexico • Singapore • United Kingdom • United States

Cengage

***Java™ Programming,* Tenth Edition**
Joyce Farrell

SVP, Higher Education Product Management: Erin Joyner

VP, Product Management, Learning Experiences: Thais Alencar

Product Director: Mark Santee

Associate Product Manager: Tran Pham

Product Assistant: Ethan Wheel

Learning Designer: Mary Convertino

Senior Content Manager: Maria Garguilo

Associate Digital Delivery Quality Partner: David O'Connor

Technical Editor: John Freitas

Developmental Editor: Dan Seiter

VP, Product Marketing: Jason Sakos

Director, Product Marketing: April Danaë

Portfolio Marketing Manager: Mackenzie Paine

IP Analyst: Ann Hoffman

IP Project Manager: Integra Software Services

Production Service: Straive

Senior Designer: Erin Griffin

Cover Image Source: iStock.com/gremlin

Library of Congress Control Number: 2021925780
ISBN: 978-0-357-67342-3

Cengage
200 Pier 4 Boulevard
Boston, MA 02210
USA

Cengage is a leading provider of customized learning solutions with employees residing in nearly 40 different countries and sales in more than 125 countries around the world. Find your local representative at **www.cengage.com**.

To learn more about Cengage platforms and services, register or access your online learning solution, or purchase materials for your course, visit **www.cengage.com.**

Printed in the United States of America
Print Number: 01 Print Year: 2022

BRIEF CONTENTS

CONTENTS

CHAPTER 5

CHAPTER 6

CHAPTER 7

CHAPTER 12

CHAPTER 13

CHAPTER 14

PREFACE

Java Programming, Tenth Edition provides the beginning programmer with a guide to developing applications using the Java programming language. Java is popular among professional programmers because it is object-oriented, making complex problems easier to solve than in some other languages. Java is used for desktop computing, mobile computing, game development, Web development, and numerical computing.

This course assumes that you have little or no programming experience. It provides a solid background in good object-oriented programming techniques and introduces terminology using clear, familiar language. The programming examples are business examples; they do not assume a mathematical background beyond high school business math. In addition, the examples illustrate only one or two major points; they do not contain so many features that you become lost following irrelevant and extraneous details. Complete, working programs appear frequently in each chapter; these examples help students make the transition from the theoretical to the practical. The code presented in each chapter also can be downloaded from the Cengage website, so students easily can run the programs and experiment with changes to them.

The student using *Java Programming, Tenth Edition* builds applications from the bottom up rather than starting with existing objects. This facilitates a deeper understanding of the concepts used in object-oriented programming and engenders appreciation for the existing objects students use as their knowledge of the language advances. When students complete this course, they will know how to modify and create simple Java programs, and they will have the tools to create more complex examples. They also will have a fundamental knowledge of object-oriented programming, which will serve them well in advanced Java courses or in studying other object-oriented languages such as C++, C#, and Visual Basic.

Organization and Coverage

Java Programming, Tenth Edition presents Java programming concepts, enforcing good style, logical thinking, and the object-oriented paradigm. Objects are covered right from the beginning, earlier than in many other Java courses. You create your first Java program in Chapter 1. Chapters 2, 3, and 4 increase your understanding about how data, classes, objects, and methods interact in an object-oriented environment.

Chapters 5 and 6 explore input and repetition structures, which are the backbone of programming logic and essential to creating useful programs in any language. You learn the special considerations of string and array manipulation in Chapters 7 and 8.

Chapters 9 and 10 thoroughly cover inheritance, interfaces, and exception handling. Inheritance is the object-oriented concept that allows you to develop new objects quickly by adapting the features of existing objects, interfaces define common methods that must be implemented in all classes that use them, and exception handling is the object-oriented approach to handling errors. All of these are important concepts in object-oriented design. Chapter 11 provides information about handling files so you can store and retrieve program output.

Chapter 12 explains recursion, and Chapter 13 covers Java collections and generics. Both are important programming concepts, and Java provides excellent ways to implement and learn about them. Chapter 14 introduces GUI `Swing` components, which are used to create visually pleasing, user-friendly, interactive applications.

New in This Edition

The following features are new for the Tenth Edition:

- **Java:** All programs have been tested using Java 16.
- **Java help:** Instructions on searching for Java help have been updated to avoid using specific URLs because new Java versions are now being released twice a year.
- **Text blocks:** Chapter 2 introduces text blocks—a new feature since Java 13.
- **Methods:** Methods are covered thoroughly in Chapter 3, including topics such as overloading methods and avoiding ambiguity. In previous editions, the material was split between chapters.
- **Classes and objects:** Classes and objects are covered thoroughly in Chapter 4. In previous editions, the material was split between chapters.
- **The `switch` expression:** Chapter 5 includes the `switch` expression, which became a new feature in Java 14.
- **Arrays:** Chapter 8 covers beginning and advanced array concepts. In previous editions, this content was split between chapters.
- **Inheritance and interfaces:** Chapter 9 covers inheritance and interfaces. In previous editions, this content was split between chapters.
- **The `record` keyword:** Chapter 9 also introduces the `record` keyword, which allows simple classes to be developed more quickly because a constructor and methods to get and set fields are created automatically based on field definitions.
- **Recursion:** Chapter 12 is a new chapter on recursion. The chapter presents techniques to use to solve mathematical problems, manipulate strings, and create visual patterns using recursion.
- **Collections and generics:** Chapter 13 is a new chapter on collections and generics. The chapter covers the `Collection` and `List` interfaces, the `ArrayList` and `LinkedList` classes, `Iterators`, and generic classes and methods.

Additionally, *Java Programming, Tenth Edition* includes the following features:

- **Objectives:** Each chapter begins with a list of objectives so you know the topics that will be presented in the chapter. In addition to providing a quick reference to topics covered, this feature provides a useful study aid.
- **You Do It:** In each chapter, step-by-step exercises help students create multiple working programs that emphasize the logic a programmer uses in choosing statements to include. These sections provide a means for students to achieve success on their own—even those in online or distance learning classes.
- **Notes:** These highlighted tips provide additional information—for example, an alternative method of performing a procedure, another term for a concept, background information about a technique, or a common error to avoid.
- **Emphasis on student research:** The student frequently is advised to use the Web to investigate Java classes, methods, and techniques. Computer languages evolve, and programming professionals must understand how to find the latest language improvements.
- **Figures:** Each chapter contains many figures. Code figures are most frequently 25 lines or fewer, illustrating one concept at a time. Frequent screenshots show exactly how program output appears. Callouts appear where needed to emphasize a point.
- **Color:** The code figures in each chapter contain all Java keywords in blue. This helps students identify keywords more easily, distinguishing them from programmer-selected names.
- **Files:** More than 200 student files can be downloaded from the Cengage website. Most files contain the code presented in the figures in each chapter; students can run the code for themselves, view the output, and make

changes to the code to observe the effects. Other files include debugging exercises that help students improve their programming skills.

- **Two Truths & a Lie:** A short quiz reviews almost every chapter section, with answers provided. This quiz contains three statements based on the preceding section of text—two statements are true, and one is false. Over the years, students have requested answers to problems, but we have hesitated to distribute them in case instructors want to use problems as assignments or test questions. These true-false quizzes provide students with immediate feedback as they read, without "giving away" answers to the multiple-choice questions and programming exercises.
- **Don't Do It:** This section at the end of each chapter summarizes common mistakes and pitfalls that plague new programmers while learning the current topic.
- **Summary:** Following each chapter is a summary that recaps the programming concepts and techniques covered in the chapter. This feature provides a concise means for students to check their understanding of the main points in each chapter.
- **Key Terms:** Each chapter includes a list of newly introduced vocabulary, shown in alphabetical order. The list of key terms provides a short review of the major concepts in the chapter.
- **Review Questions:** Each chapter includes 20 multiple-choice questions that serve as a review of chapter topics.
- **Programming Exercises:** Multiple programming exercises are included with each chapter. These challenge students to create complete Java programs that solve real-world problems.
- **Debugging Exercises:** Four debugging exercises are included with each chapter. These are programs that contain logic or syntax errors that the student must correct. Besides providing practice in deciphering error messages and thinking about correct logic, these exercises provide examples of complete and useful Java programs after the errors are repaired.
- **Game Zone:** Each chapter provides one or more exercises in which students can create interactive games using the programming techniques learned up to that point; 50 game programs are suggested in the course. The games are fun to create and play; writing them motivates students to master the necessary programming techniques. Students might exchange completed game programs with each other, suggesting improvements and discovering alternate ways to accomplish tasks.
- **Cases:** Each chapter contains two running case problems. These cases represent projects that continue to grow throughout a semester using concepts learned in each new chapter. Two cases allow instructors to assign different cases in alternate semesters or to divide students in a class into two case teams.
- **Glossary:** A glossary contains definitions for all key terms in the course.
- **Appendices:** This edition includes useful appendices on working with the Java platform, data representation, formatting output, generating random numbers, creating Javadoc comments, and JavaFX.
- **Quality:** Every program example, exercise, and game solution was tested by the author and then tested again by a quality assurance team.

MindTap Instructor Resources

MindTap activities for *Java Programming, Tenth Edition* are designed to help students master the skills they need in today's workforce. Research shows employers need critical thinkers, troubleshooters, and creative problem-solvers to stay relevant in our fast-paced, technology-driven world. MindTap helps you achieve this with assignments and activities that provide hands-on practice and real-life relevance. Students are guided through assignments that help them master basic knowledge and understanding before moving on to more challenging problems.

All MindTap activities and assignments are tied to defined unit learning objectives. MindTap provides the analytics and reporting so you can easily see where the class stands in terms of progress, engagement, and completion rates. Use

the content and learning path as is or pick and choose how our materials will wrap around yours. You control what the students see and when they see it. Learn more at *http://www.cengage.com/mindtap/*.

In addition to the readings, the *Java Programming, Tenth Edition* MindTap includes the following:

- **Gradeable assessments and activities:** All assessments and activities from the readings will be available as gradeable assignments within MindTap, including Review Questions, Game Zone, Case Problems, and Two Truths & a Lie.
- **Videos:** Each unit is accompanied by videos that help to explain important unit concepts and provide demos on how students can apply those concepts.
- **Coding Snippets:** These short, ungraded coding activities are embedded in the MindTap Reader and provide students an opportunity to practice new programming concepts "in the moment." The coding Snippets help transition the students from conceptual understanding to application of Java code.
- **Coding labs:** These assignments provide real-world application and encourage students to practice new coding skills in a complete online IDE. Guided feedback provides personalized and immediate feedback to students as they proceed through their coding assignments so that they can understand and correct errors in their code.
- **Interactive study aids:** Flashcards and PowerPoint lectures help users review main concepts from the units.

Instructor and Student Resources

Additional instructor and student resources for this product are available online. Instructor assets include an Instructor's Manual, Educator's Guide, PowerPoint® slides, Solution and Answer Guide, solutions, and a test bank powered by Cognero®. Student assets include data files. Sign up or sign in at *www.cengage.com* to search for and access this product and its online resources.

- **Instructor's Manual:** The Instructor's Manual includes additional instructional material to assist in class preparation, including sections such as Chapter Objectives, Complete List of Chapter Activities and Assessments, Key Terms, What's New In This Chapter, Chapter Outline, Discussion Questions, Suggested Usage for Lab Activities, Additional Activities and Assignments, and Additional Resources. A sample syllabus also is available.
- **PowerPoint presentations:** The PowerPoint slides can be used to guide classroom presentations, to make available to students for chapter review, or to print as classroom handouts.
- **Solution and Answer Guide:** Solutions to all end-of-chapter assignments are provided along with feedback.
- **Solutions:** Solutions to all programming exercises are available. If an input file is needed to run a programming exercise, it is included with the solution file.
- **Test bank:** Cengage Testing Powered by Cognero is a flexible, online system that allows you to:
  - Author, edit, and manage test bank content from multiple Cengage solutions.
  - Create multiple test versions in an instant.
  - Deliver tests from your LMS, your classroom, or wherever you want.
- **Educator's Guide:** The Educator's Guide contains a detailed outline of the corresponding MindTap course.
- **Transition Guide:** The Transition Guide outlines information on what has changed from the Ninth Edition.
- **Data files:** Data files necessary to complete some of the steps and projects in the course are available. The Data Files folder includes Java files that are provided for every program that appears in a figure in the text.

About the Author

Joyce Farrell has taught computer programming at McHenry County College, Crystal Lake, Illinois; the University of Wisconsin, Stevens Point, Wisconsin; and Harper College, Palatine, Illinois. Besides Java, she has written books on programming logic and design, C#, and C++ for Cengage.

Acknowledgments

I would like to thank all of the people who helped to make this project a reality, including Tran Pham, Associate Product Manager; Mary Convertino, Learning Designer; Maria Garguilo, Senior Content Manager; Dan Seiter, Developmental Editor, and John Freitas, Quality Assurance Tester. I am lucky to work with these professionals who are dedicated to producing high-quality instructional materials.

I am also grateful to the reviewers who provided comments and encouragement during this course's development, including Dr. Ross Foultz, Coastal Carolina University; and Dr. Carl M. Rebman, Jr., University of San Diego. Also, thank you to Charles W. Lively III, Ph.D. – Academic Faculty, Georgia Institute of Technology, who provided the appendix on JavaFX.

Thanks, too, to my husband, Geoff, for his constant support, advice, and encouragement. Finally, this project is dedicated to Norman Williams Peterson, who has brought a smile to my face every time I have seen him.

Joyce Farrell

Read This Before You Begin

The following information will help you as you prepare to complete this course.

To the User of the Data Files

To complete the steps and projects in this course, you need data files that have been created specifically for some of the exercises. Your instructor will provide the data files to you. You also can obtain the files electronically by signing up or signing in at *www.cengage.com* and then searching for and accessing this product and its online resources. Note that you can use a computer in your school lab or your own computer to complete the exercises.

Using Your Own Computer

To use your own computer to complete the steps and exercises, you need the following:

- **Software:** Java SE 16 or later, available from *www.oracle.com/technetwork/java/index.html*. Although almost all of the examples in this course will work with earlier versions of Java, a few require Java 16 or later. You also need a text editor, such as Notepad. A few exercises ask you to use a browser for research.
- **Hardware:** For operating system requirements (memory and disk space), see *http://java.com/en/download/help*.

Features

This text focuses on helping students become better programmers and understand Java program development through a variety of key features. In addition to Chapter Objectives, Summaries, and Key Terms, these useful features will help students regardless of their learning styles.

You Do It

These sections walk students through program development step by step.

Note

These notes provide additional information—for example, a common error to watch out for or background information on a topic.

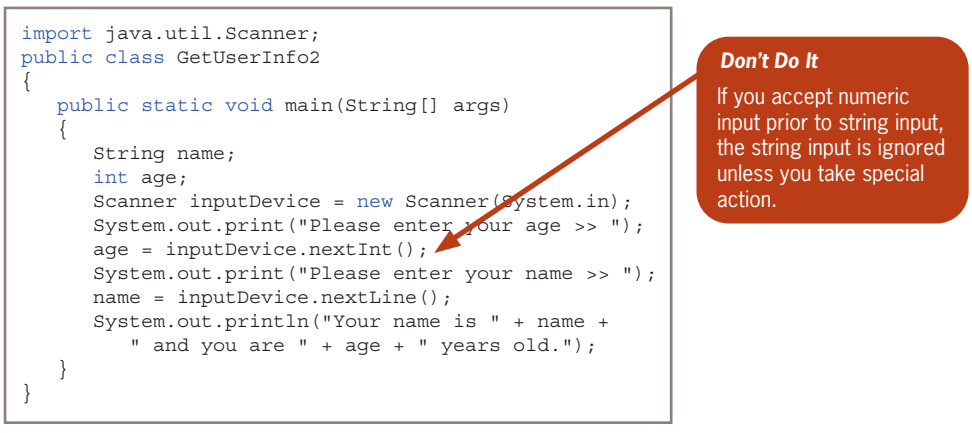
Two Truths & a Lie

These quizzes appear after almost every chapter section, with answers provided. Each quiz contains three statements based on the preceding section of text—two statements are true and one is false.

Answers give immediate feedback without "giving away" answers to the multiple-choice questions and programming problems later in the chapter. Students also have the option to take these quizzes in MindTap.

Don't Do It Icon

The Don't Do It icon illustrates how NOT to do something—for example, having a dead code path in a program. These icons provide a visual jolt to the student, emphasizing that particular practices are NOT to be emulated and making students more likely to recognize problems in existing code.



Don't Do It

These sections at the end of each chapter list advice for avoiding common programming errors.

Assessment

Review Questions

Review Questions test student comprehension of the major ideas and techniques presented. Twenty questions follow each chapter.

Programming Exercises

Programming Exercises provide opportunities to practice concepts. These exercises allow students to explore each major programming concept presented in the chapter. Additional coding labs and snippets are available in MindTap.

Debugging Exercises

Debugging Exercises are included with each chapter because examining programs critically and closely is a crucial programming skill. Students and instructors can download these exercises at *www.cengage.com*.

Game Zone

Game Zone exercises are included at the end of each chapter. Students can create games as an additional entertaining way to understand key programming concepts.

Case Problems

Case Problems provide opportunities to build more detailed programs that continue to incorporate increasing functionality throughout the course.

CHAPTER 1

Creating Java Programs

Learning Objectives

When you complete this chapter, you will be able to:

- **1.1** Define basic programming terminology
- **1.2** Compare procedural and object-oriented programming
- **1.3** Describe the features of the Java programming language
- **1.4** Analyze a Java application that produces console output
- **1.5** Compile a Java class and correct syntax errors
- **1.6** Run a Java application and correct logic errors
- **1.7** Add comments to a Java class
- **1.8** Create a Java application that produces GUI output
- **1.9** Identify and consult resources to help develop Java programming skills

1.1 Learning Programming Terminology

You see many computers every day. There might be a laptop on your desk, and there also are computers in your phone, in your car, and perhaps in your thermostat, washing machine, and vacuum cleaner. When you learn computer terminology and how to program a computer, you learn a bit about how these devices work, you develop your critical thinking skills, and you learn to communicate more clearly. You will reap all these benefits as you work through this course.

Computer systems consist of both hardware and software.

- Computer equipment, such as a monitor or keyboard, is **hardware**.
- Programs are **software**. A **computer program** (or simply, **program**) is a set of instructions that you write to tell a computer what to do.

Software can be divided into two broad categories:

- A program that performs a task for a user (such as calculating and producing paychecks, word processing, or playing a game) is **application software**. Programs that are application software are called *applications*, or *apps* for short.
- A program that manages the computer itself (such as Windows or Linux) is **system software**.

The **logic** behind any computer program, whether it is an application or system program, determines the exact order of instructions needed to produce desired results. Much of this course describes how to develop the logic for application software.

You can write computer programs in high- or low-level programming languages:

- A **high-level programming language** such as Java, Visual Basic, C++, or C# allows you to use English-like, easy-to-remember terms such as *read*, *write*, and *add*.
- A **low-level programming language** corresponds closely to a computer's circuitry and is not as easily read or understood. Because they correspond to circuitry, low-level languages must be customized for every type of machine on which a program runs.

All computer programs, even high-level language programs, ultimately are converted to the lowest-level language, which is machine language. **Machine language**, or **machine code**, is the most basic set of instructions that a computer can execute. Each type of processor (the internal hardware that handles computer instructions) has its own set of machine language instructions. Programmers often describe machine language using 1s and 0s to represent the on-and-off circuitry of computer systems.

Note

The system that uses only 1s and 0s is the *binary numbering system*. Appendix B describes the binary system in detail. Later in this chapter, you will learn that *bytecode* is the name for the binary code created when Java programs are converted to machine language.

Every programming language has its own **syntax**, or rules about how language elements are combined correctly to produce usable statements. For example, depending on the specific high-level language, you might use the verb *print* or *write* to produce output. All languages have a specific, limited vocabulary (the language's **keywords**) and a specific set of rules for using that vocabulary. When you are learning a computer programming language, such as Java, C++, or Visual Basic, you are learning the vocabulary and syntax for that language.

Using a programming language, programmers write a series of **program statements**, which are similar to English sentences. The statements carry out the program's tasks. Program statements are also known as **commands** because they are orders to the computer, such as *Output this word* or *Add these two numbers*.

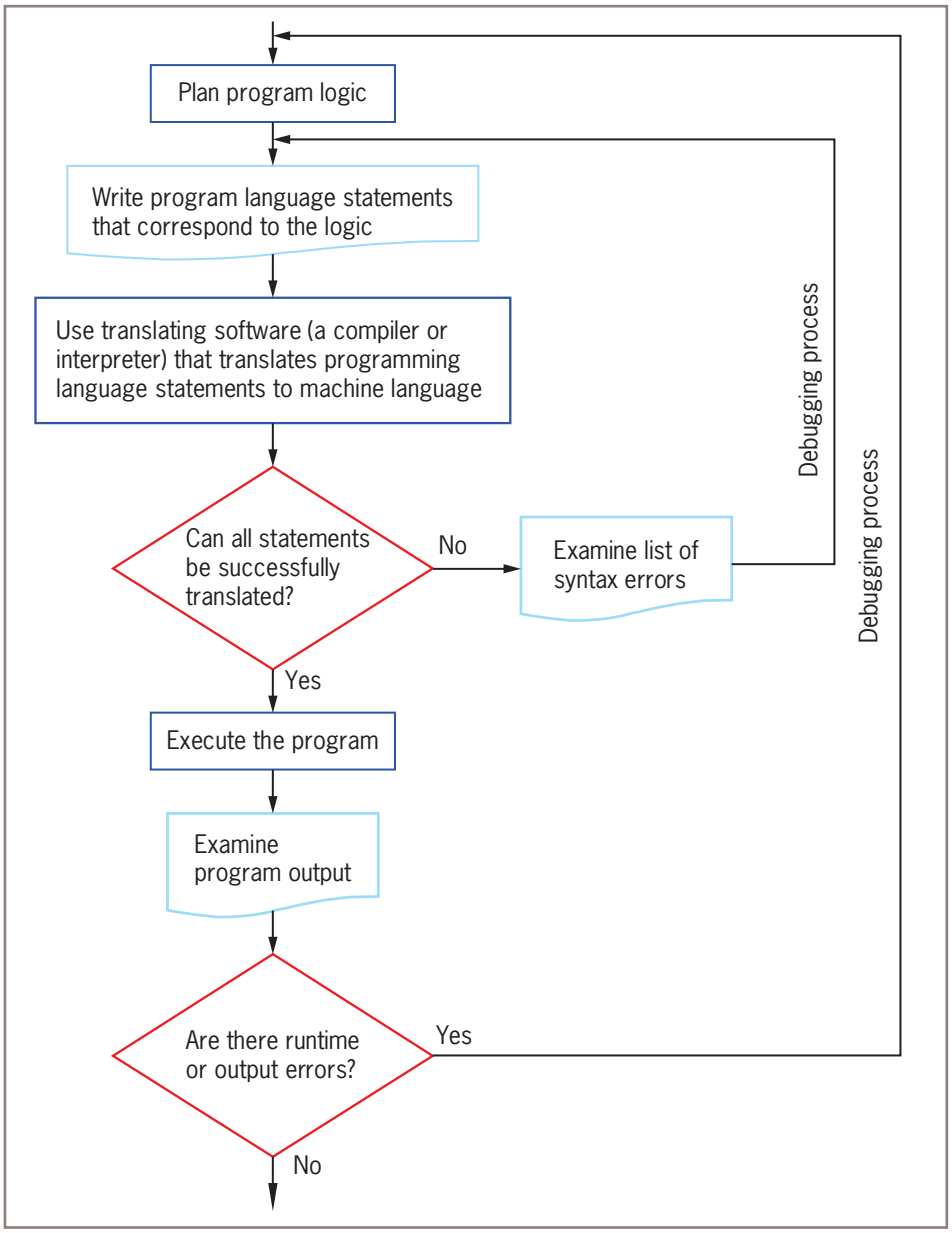
After the program statements are written in a high-level programming language, a computer program called a **compiler** or **interpreter** translates the statements into machine language. A compiler translates an entire program before carrying out any statements, or **executing** them, whereas an interpreter translates one program statement at a time, executing a statement as soon as it is translated.

Note

Whether you use a compiler or interpreter often depends on the programming language you use. For example, C++ is a compiled language, and Visual Basic is an interpreted language. Each type of translator has its supporters; programs written in compiled languages execute more quickly, whereas programs written in interpreted languages can be easier to develop and debug. Java uses the best of both technologies: a compiler to translate your programming statements and an interpreter to read the compiled code line by line when the program executes (also called **at run time**).

Compilers and interpreters issue one or more error messages each time they encounter an invalid program statement—that is, a statement containing a **syntax error**, or misuse of the language. Examples of syntax errors include misspelling a keyword or omitting a word that a statement requires. When a syntax error is detected, the programmer can correct the error and attempt another translation. Repairing all syntax errors is the first part of the process of **debugging** a program—freeing the program of all flaws or errors, also known as **bugs**. **Figure 1-1** illustrates the steps a programmer takes while developing an executable program. You will learn more about debugging Java programs later in this chapter.

Figure 1-1 The program development process



As Figure 1-1 shows, you might write a program that compiles successfully (that is, it contains no syntax errors), but it still might not be a correct program because it might contain one or more logic errors. A **logic error** is a bug that allows a program to run, but that causes it to operate incorrectly. Correct logic requires that all the right commands be issued in the appropriate order. Examples of logic errors include multiplying two values when you meant to divide them or producing output prior to obtaining the appropriate input. When you develop a program of any significant size, you should plan its logic before you write any program statements.

Correcting logic errors is much more difficult than correcting syntax errors. Syntax errors are discovered by the language translator when you compile a program, but a program can be free of syntax errors and execute while still

retaining logic errors. Sometimes you can find logic errors by carefully examining the structure of your program (when a group of programmers do this together, it is called a *structured walkthrough*), but sometimes you can identify logic errors only when you examine a program's output. For example, if you know an employee's paycheck should contain the value $4,000, but when you examine a payroll program's output you see that it holds $40, then a logic error has occurred. Perhaps an incorrect calculation was performed, or maybe the hours-worked value was output by mistake instead of the net pay value. When output is incorrect, the programmer must carefully examine all the statements within the program, revise or move the offending statements, and translate and test the program again.

Note

Just because a program produces correct output does not mean it is free from logic errors. For example, suppose that a program should multiply two values entered by the user, that the user enters two 2s, and that the output is 4. The program might actually be adding the values by mistake. The programmer would discover the logic error only by entering different values, such as 5 and 7, and examining the result.

Note

Programmers call some logic errors **semantic errors**. For example, if you misspell a programming language word, you commit a syntax error, but if you use a correct word in the wrong context, you commit a semantic error.

Two Truths & a Lie | Learning Programming Terminology

In each "Two Truths & a Lie" section, two of the numbered statements are true, and one is false. Identify the false statement and explain why it is false.

1. Unlike a low-level programming language, a high-level programming language allows you to use a vocabulary of reasonable terms instead of the sequences of on-and-off switches that perform the corresponding tasks.
2. A syntax error occurs when you violate the rules of a language; locating and repairing all syntax errors is part of the process of debugging a program.
3. Logic errors are fairly easy to find because the software that translates a program finds all the logic errors for you.

The false statement is #3. A language translator finds syntax errors, but logic errors can still exist in a program that is free of syntax errors.

1.2 Comparing Procedural and Object-Oriented Programming Concepts

All computer programmers must deal with syntax errors and logical errors in much the same way, but they might take different approaches to the entire programming process. Procedural programming and object-oriented programming describe two different approaches to writing computer programs.

Procedural Programming

Procedural programming is a style of programming in which operations are executed one after another in sequence.

The typical procedural program defines and uses named computer memory locations; each of these named locations that can hold data is called a **variable**. For example, data might be read from an input device and stored in a location the programmer has named `rateOfPay`. The variable value might be used in an arithmetic statement, used as the basis for a decision, sent to an output device, or have other operations performed with it. The data stored in a variable can change, or vary, during a program's execution.

For convenience, the individual operations used in a computer program are often grouped into logical units called **procedures**. For example, a series of four or five comparisons and calculations that together determine a person's federal withholding tax value might be grouped as a procedure named `calculateFederalWithholding()`. (As a convention, this course will show parentheses following every procedure name.) As a procedural program executes its statements, it can sometimes pause to **call a procedure**. When a program calls a procedure, the current logic is temporarily suspended so that the procedure's commands can execute. A single procedural program might contain any number of procedure calls. Procedures are also called *modules*, *methods*, *functions*, and *subroutines*. Users of different programming languages tend to use different terms. As you will learn later in this chapter, Java programmers most frequently use the term *method*.

Object-Oriented Programming

Object-oriented programming is an extension of procedural programming in which you take a slightly different approach to writing computer programs. Writing **object-oriented programs** involves the following:

- Creating classes, which are blueprints for objects
- Creating objects, which are specific instances of those classes
- Creating applications that manipulate or use those objects

Note

Programmers use *OO* as an abbreviation for *object-oriented*; it is pronounced *oh oh*. Object-oriented programming is abbreviated *OOP*, and pronounced to rhyme with *soup*.

Originally, object-oriented programming was used most frequently for two major types of applications:

- **Computer simulations**, which attempt to mimic real-world activities so that their processes can be improved or so that users can better understand how the real-world processes operate
- **Graphical user interfaces (GUIs)**, pronounced *gooeys*, which allow users to interact with a program in a graphical environment

Thinking about objects in these two types of applications makes sense. For example, a city might want to develop a program that simulates traffic patterns and controls traffic signals to help prevent tie-ups. Programmers would create classes for objects such as cars and pedestrians that contain their own data and rules for behavior. For example, each car has a speed and a method for changing that speed. The specific instances of cars could be set in motion to create a simulation of a real city at rush hour.

Creating a GUI environment for users is also a natural use for object orientation. It is easy to think of the components a user manipulates on a computer screen, such as buttons and scroll bars, as similar to real-world objects. Each GUI object contains data—for example, a button on a screen has a specific size and color. Each object also contains behaviors—for example, each button can be clicked and reacts in a specific way when clicked. Some people consider the term *object-oriented programming* to be synonymous with GUI programming, but object-oriented programming means more. Although many GUI programs are object oriented, not all object-oriented programs use GUI objects. Modern businesses use object-oriented design techniques when developing all sorts of business applications, whether they are GUI applications or not. Early in this course, you will learn object-oriented techniques that are appropriate for any program type; in the last chapters, you will apply what you have learned about those techniques specifically to GUI applications.

Understanding object-oriented programming requires grasping three basic concepts:

- Encapsulation as it applies to classes as objects
- Inheritance
- Polymorphism

Understanding Classes, Objects, and Encapsulation

In object-oriented terminology, a **class** is a group or collection of objects with common properties. In the same way that a blueprint exists before any houses are built from it, and a recipe exists before any cookies are baked from it, a class definition exists before any objects are created from it. A **class definition** describes what attributes its objects will have and what those objects will be able to do. **Attributes** are the characteristics that define an object; they are **properties** of the object. When you learn a programming language such as Java, you learn to work with two types of classes: those that have already been developed by the language's creators and your own new, customized classes.

An **object** is a specific, concrete **instance** of a class. Creating an instance is called **instantiation**. You can create objects from classes that you write and from classes written by other programmers, including Java's creators. The values contained in an object's properties often differentiate instances of the same class from one another. For example, the class `Automobile` describes what `Automobile` objects are like. Some properties of the `Automobile` class are make, model, year, and color. Each `Automobile` object possesses the same attributes, but not necessarily the same values for those attributes. One `Automobile` might be a 2014 white Honda Civic, and another might be a 2021 red Chevrolet Camaro. Similarly, your dog has the properties of all `Dog`s, including a breed, name, age, and whether the dog's shots are current. The values of the properties of an object are referred to as the object's **state**. In other words, you can think of objects as roughly equivalent to nouns (words that describe a person, place, or thing), and of their attributes as similar to adjectives that describe the nouns.

When you understand an object's class, you understand the characteristics of the object. If your friend purchases an `Automobile`, you know it has a model name, and if your friend gets a `Dog`, you know the dog has a breed. Knowing what attributes exist for classes allows you to ask appropriate questions about the states or values of those attributes. For example, you might ask how many miles the car gets per gallon, but you would not ask whether the car has had shots. Similarly, in a GUI operating environment, you expect each component to have specific, consistent attributes and methods, such as a window having a title bar and a close button, because each component gains these properties as a member of the general class of GUI components. **Figure 1-2** shows the relationship of some `Dog` objects to the `Dog` class.

Note

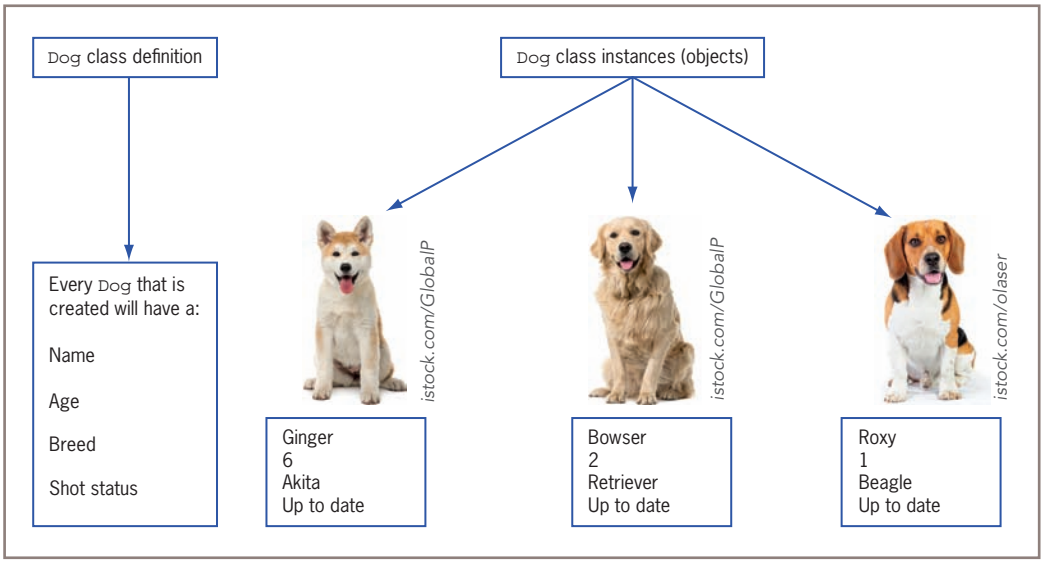
By convention, programmers using Java begin their class names with an uppercase letter. Thus, the class that defines the attributes and methods of an automobile probably would be named `Automobile`, and the class for dogs probably would be named `Dog`. This convention, however, is not required to produce a workable program.

Besides defining properties, classes define methods their objects can use. A **method** is a self-contained block of program code that carries out some action, similar to a procedure in a procedural program. An `Automobile`, for example, might have methods for moving forward, moving backward, and determining the status of its gas tank. Similarly, a `Dog` might have methods for walking, eating, and determining its name, and a program's GUI components might have methods for maximizing and minimizing them as well as determining their size. In other words, if objects are similar to nouns, then methods are similar to verbs.

In object-oriented classes, attributes and methods are encapsulated into objects. **Encapsulation** refers to two closely related object-oriented notions:

- Encapsulation is the enclosure of data and methods within an object. Encapsulation allows you to treat all of an object's methods and data as a single entity. Just as an actual dog contains all of its attributes and abilities, so would a program's `Dog` object.

Figure 1-2 Dog class definition and some objects created from it



- Encapsulation also refers to the concealment of an object's data and methods from outside sources. Concealing data is sometimes called *information hiding*, and concealing how methods work is *implementation hiding*; you will learn more about both terms as you learn more about classes and objects. Encapsulation lets you hide specific object attributes and methods from outside sources and provides the security that keeps data and methods safe from inadvertent changes.

If an object's methods are well written, the user can be unaware of the low-level details of how the methods are executed, and the user must simply understand the interface or interaction between the method and the object. For example, if you can fill your `Automobile` with gasoline, it is because you understand the interface between the gas pump nozzle and the vehicle's gas tank opening. You don't need to understand how the pump works mechanically or where the gas tank is located inside your vehicle. If you can read your speedometer, it does not matter how the displayed value is calculated. As a matter of fact, if someone produces a superior, more accurate speed-determining device and inserts it in your `Automobile`, you don't have to know or care how it operates, as long as your interface remains the same. The same principles apply to well-constructed classes used in object-oriented programs—programs that use classes only need to work with interfaces.

Understanding Inheritance and Polymorphism

An important feature of object-oriented program design that differentiates it from procedural program design is **inheritance**—the ability to create classes that share the attributes and methods of existing classes, but with more specific features. For example, `Automobile` is a class, and all `Automobile` objects share many traits and abilities. `Convertible` is a class that inherits from the `Automobile` class; a `Convertible` is a type of `Automobile` that has and can do everything a "plain" `Automobile` does—but with an added ability to lower its top. (In turn, `Automobile` inherits from the `Vehicle` class.) `Convertible` is not an object—it is a class. A specific `Convertible` is an object—for example, `my1967BlueMustangConvertible`.

Inheritance helps you understand real-world objects. For example, the first time you encounter a convertible, you already understand how the ignition, brakes, door locks, and other systems work because you realize that a convertible is a type of automobile. Therefore, you need to be concerned only with the attributes and methods that are "new" with a convertible. The advantages in programming are the same—you can build new classes based on existing classes and concentrate on the specialized features you are adding.

A final important concept in object-oriented terminology (that does not exist in procedural programming terminology) is **polymorphism**. Literally, polymorphism means *many forms*—it describes the feature of languages that allows the

same word or symbol to be interpreted correctly in different situations based on the context. For example, although the classes `Automobile`, `Sailboat`, and `Airplane` all inherit from `Vehicle`, methods such as `turn()` and `stop()` work differently for instances of those classes. The advantages of polymorphism will become more apparent when you begin to create GUI applications containing features such as windows, buttons, and menu bars. In a GUI application, it is convenient to remember one method name, such as `setColor()` or `setHeight()`, and have it work correctly no matter what type of object you are modifying.

Note

When you see a plus sign (+) between two numbers, you understand they are being added. When you see it carved in a tree between two names, you understand that the names are linked romantically. Because the symbol has diverse meanings based on context, it is polymorphic. Later in this course, you will learn more about inheritance and polymorphism and how they are implemented in Java. Using Java, you can write either procedural or object-oriented programs. In this course, you will learn about how to do both.

Two Truths & a Lie | Comparing Procedural and Object-Oriented Programming Concepts

1. An instance of a class is a created object that possesses the attributes and methods described in the class definition.
2. Encapsulation protects data by hiding it within an object.
3. Polymorphism is the ability to create classes that share the attributes and methods of existing classes, but with more specific features.

The false statement is #3. Inheritance is the ability to create classes that share the attributes and methods of existing classes, but with more specific features; polymorphism describes the ability to use one term to cause multiple actions.

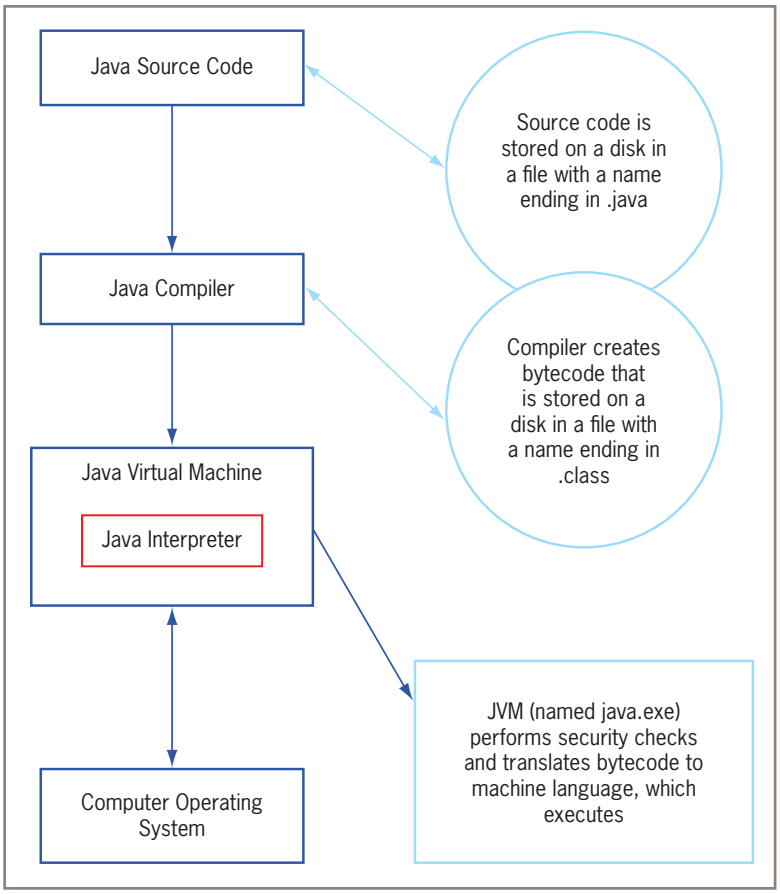
1.3 Features of the Java Programming Language

Java was developed by Sun Microsystems as an object-oriented language for general-purpose business applications and for interactive, World Wide Web-based Internet applications. (Sun was later acquired by Oracle Corporation.) Some of the advantages that make Java a popular language are its security features and the fact that it is **architecturally neutral**. In other words, unlike many other languages, you can use Java to write a program that runs on any operating system (such as Windows, macOS, or Linux) or any device (such as PCs, phones, and tablet computers).

Java can be run on a wide variety of computers and devices because it does not execute instructions on a computer directly. Instead, Java runs on a hypothetical computer known as the **Java Virtual Machine (JVM)**. When programmers call the JVM *hypothetical*, they mean it is not a physical entity created from hardware, but is composed only of software.

Figure 1-3 shows the Java environment. Programming statements written in a high-level programming language are **source code**. When you write a Java program, you first construct the source code using a plain text editor such as Notepad, or you can use a development environment such as Eclipse, NetBeans, or JDeveloper. A **development environment** is a set of tools that help you write programs by providing such features as displaying a language's keywords in color.

Figure 1-3 The Java environment



When you write a Java program, the following steps take place:

- The Java source code statements you write are saved in a file.
- The Java compiler converts the source code into a binary program of **bytecode**.
- A program called the **Java interpreter** then checks the bytecode and communicates with the operating system, executing the bytecode instructions line by line within the JVM.

Because the Java program is isolated from the operating system, it is also insulated from the particular hardware on which it is run. Because of this insulation, the JVM provides security against intruders accessing your computer's hardware through the operating system. Therefore, Java is more secure than other languages. Another advantage provided by the JVM means less work for programmers—when using other programming languages, software vendors usually have to produce multiple versions of the same product (a Windows version, Macintosh version, UNIX version, Linux version, and so on) so all users can run the program. With Java, one program version runs on all these platforms. **"Write once, run anywhere" (WORA)** is the slogan developed by Sun Microsystems to describe the ability of one Java program version to work correctly on multiple platforms.

Java also is simpler to use than many other object-oriented languages. Java is modeled after C++. Although neither language is easy to read or understand on first exposure, Java does eliminate some of the most difficult-to-understand features in C++, such as pointers and multiple inheritance.

You can write two types of Java applications:

- **Console applications**, which support character or text output to a computer screen
- **Windowed applications**, which create a GUI with elements such as menus, toolbars, and dialog boxes