CENGAGE Learning

国外信息科学与技术经典图书系列

Fundamentals of JAVA Programming

(Fifth Edition)

JAVA编程

（第五版）

〔美〕Joyce Farrell 著

（英文影印版）

科学出版社

国外信息科学与技术经典图书系列

JAVA 编程（英文影印版）

（第五版）

Fundamentals of JAVA Programming

（Fifth Edition）

〔美〕 Joyce Farrell 著

科 学 出 版 社

北 京

图字：01-2011-6335

内 容 简 介

本书为入门级程序员提供了用 JAVA 编程语言开发应用程序的方法。JAVA 语言深受专业程序员青睐,因为它可以用来制造在视觉上有趣的图形用户界面(GUI)和互联网应用程序。本书也为学生在学习基本的结构化和面向对象程序设计技术的前提下，尽快开始程序编制提供了良好的指导。

本书可作为计算机专业的双语教材或教学参考书，也可供工程技术人员参考。

Joyce Farrell
Fundamentals of JAVA Programming , 5e
ISBN：978-1-133-69109-9

Cengage Learning Asia Pte. Ltd.
5 Shenton Way, # 01-01 UIC Building, Singapore 068808
北京市版权局著作权合同登记号 图字：01-2011-6335

本书封面贴有 Cengage Learning 防伪标签，无标签者不得销售。

图书在版编目(CIP)数据

JAVA 编程 = Fundamentals of JAVA Programming：5 版：英文/(美)法瑞尔(Farrell, J.)等著. —影印本. —北京：科学出版社，2012
(国外信息科学与技术经典图书系列)
ISBN 978-7-03-032910-3
Ⅰ.①J… Ⅱ.①法… Ⅲ. ①JAVA 语言-编程设计-英文 Ⅳ. ①TP312
中国版本图书馆 CIP 数据核字(2011)第 246556 号

责任编辑：王鑫光 匡 敏 / 责任印制：张克忠 / 封面设计：迷底书装

科 学 出 版 社 出版
北京东黄城根北街16号
邮政编码：100717
http://www.sciencep.com

保定市中画美凯印刷有限公司 印刷

科学出版社发行 各地新华书店经销
*
2012 年 1 月第 一 版 开本：787×960 1/16
2012 年 1 月第一次印刷 印张：45 1/4
字数：900 000
定价：99.00 元
（如有印装质量问题，我社负责调换）

BRIEF CONTENTS

CONTENTS

PREFACE

Java Programming, Fifth Edition provides the beginning programmer with a guide to developing applications using the Java programming language. Java is popular among professional programmers because it can be used to build visually interesting graphical user interface (GUI) and Web-based applications. Java also provides an excellent environment for the beginning programmer—a student quickly can build useful programs while learning the basics of structured and object-oriented programming techniques.

This textbook assumes that you have little or no programming experience. This book provides a solid background in good object-oriented programming techniques and introduces object-oriented terminology using clear, familiar language. The writing is nontechnical and emphasizes good programming practices. The 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. The explanations in this textbook are written clearly in straightforward sentences so that native and non-native English speakers alike can master the programming concepts. Complete, working code examples appear frequently in each chapter; these examples help the student make the transition from the theoretical to the practical. The code presented in each chapter is also provided on disk, so that students can easily run the programs and experiment with changes to them.

ORGANIZATION AND COVERAGE

Java Programming, Fifth 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 textbooks. You create your first Java program in Chapter 1. Chapters 2, 3, and 4 increase your understanding of 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, 10, and 11 thoroughly cover inheritance (the object-oriented concept that allows you to develop new objects quickly by adapting the features of existing ones) and exception handling (the object-oriented approach to handling errors). Both are important concepts in object-oriented design. Chapter 12 provides information on handling files so you can permanently store and retrieve program output. Chapter 13 introduces Swing components.

In every chapter, *Java Programming, Fifth Edition* follows the text explanation with a "You Do It" section that contains step-by-step exercises to illustrate the concepts just learned, reinforcing the student's understanding and allowing concepts to be better retained. Creating the programs in the step-by-step examples also provides students with a successful experience in the language; finishing the examples provides them with models for their own creations.

The student using *Java Programming, Fifth 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 book, they will know how to modify and create simple Java programs and 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.

FEATURES

Java Programming, Fifth Edition is a superior textbook because it also 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.
- **Notes:** These highlighted tips provide additional information—for example, an alternative method of performing a procedure, another term for a concept, background information on a technique, or a common error to avoid.
- **Figures:** Each chapter contains many figures. Code figures are most frequently 25 lines or less, illustrating one concept at a time. Frequently placed screen shots show exactly how program output appears.

NEW!
- **Callouts in more figures:** Callouts have been added to many figures to help students focus on the points emphasized in the text. Some icons contain the words "Don't Do It" to emphasize when an example illustrates a practice not to emulate.
- **Color:** The code figures in each chapter contain all Java keywords in brown. This helps students identify keywords more easily, distinguishing them from programmer-selected names.
- **Files:** The Student Disk holds more than 180 files that 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.

NEW!
- **Two Truths and a Lie:** A new quiz reviews each chapter section, with answers provided. This quiz contains three statements from 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 mini-quizzes provide students with immediate feedback as they read, without "giving away" answers to the existing multiple-choice and programming problem questions.
- **You Do It:** In each chapter, step-by-step exercises help the student create multiple working programs that emphasize the logic a programmer uses in choosing statements to include. This section provides a means for students to achieve success on their own—even those in online or distance learning classes.

NEW!
- **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.

- **Key Terms:** Each chapter includes a list of newly introduced vocabulary, shown in the order of appearance in the text. The list of key terms provides a mini-review of the major concepts in the chapter.
- **Summaries:** Following each chapter is a summary that recaps the programming concepts and techniques covered in the chapter. This feature helps students check their understanding of the main points in each chapter.
- **Review Questions:** Each chapter includes 20 multiple-choice questions that serve as a review of chapter topics.
- **Exercises:** Each chapter concludes with meaningful programming exercises that provide additional practice of the skills and concepts learned in the chapter. These exercises vary in difficulty and are designed to allow exploration of logical programming concepts.
- **Game Zone:** Each chapter provides one or more exercises in which the student creates interactive games using the programming techniques learned up to that point; 70 game programs are suggested in the book. 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.
- **Tough Questions:** Each chapter includes two or more fairly difficult, and often open-ended, questions that are typical of what an applicant might encounter in a technical job interview. Some questions involve coding; others might involve research. NEW!
- **Up for Discussion:** Each chapter concludes with a few thought-provoking questions concerning programming in general or Java in particular. The questions can be used to start classroom or online discussions, or to develop and encourage research, writing, and language skills.
- **Glossary:** This edition includes a glossary that contains definitions for all key terms in the book, presented in alphabetical order. NEW!
- **Appendix on javadoc:** This edition includes a new appendix on creating javadoc comments. NEW!
- **Other pedagogical improvements:** This edition introduces the following pedagogical improvements: NEW!
  - The `Scanner` class is introduced in Chapter 2 to facilitate user keyboard entry in programs.
  - Programming examples provide earlier and more consistent use of named constants.
  - Clearer distinction between troublesome concepts is provided—for example, argument vs. parameter and static vs. nonstatic.
  - The `String` chapter focuses on `StringBuilder` instead of `StringBuffer` because `StringBuilder` is more efficient. However, it is emphasized that the two classes are used in exactly the same way.
  - The GUI chapters have been completely rewritten and moved later in the book, which makes it easier for instructors who want to cover the concepts of inheritance and polymorphism first. Similarly, applet coverage has been removed from the GUI chapters, which makes it easier for instructors who want to cover GUI topics first.
  - Applets have been moved to the last chapter in the book, reflecting their diminished popularity as a business tool.
- **Quality:** Every program example in the book, as well as every exercise and game solution, was tested by the author and then tested again by a Quality Assurance team using Java Standard Edition (SE) 6, the most recent version available. (The external version number used by Sun Microsystems is 6.0; the internal version number is 1.6.0. For more information on the features of the JDK, visit *http://java.sun.com*.)

- **CD-ROM included with book:** The CD that comes with this book includes the following items:
  - Sun Microsystems Java SE 6, the Java language, compiler, and runtime environment
  - The jGRASP integrated development environment for Java
  - Code files for all Java program examples contained in the text

TEACHING TOOLS

The following supplemental materials are available when this book is used in a classroom setting. All of the teaching tools available with this book are provided to the instructor on a single CD.

- **Electronic Instructor's Manual:** The Instructor's Manual that accompanies this textbook includes additional instructional material to assist in class preparation, including items such as Sample Syllabi, Chapter Outlines, Technical Notes, Lecture Notes, Quick Quizzes, Teaching Tips, Discussion Topics, and Key Terms.
- **ExamView®:** This textbook is accompanied by ExamView, a powerful testing software package that allows instructors to create and administer printed, computer (LAN-based), and Internet-based exams. ExamView includes hundreds of questions that correspond to the topics covered in this text, enabling students to generate detailed study guides that include page references for further review. The computer-based and Internet testing components allow students to take exams at their computers, and they save the instructor time by grading each exam automatically.
- **PowerPoint Presentations:** This book comes with Microsoft PowerPoint slides for each chapter. These are included as a teaching aid for classroom presentation, to make available to students on the network for chapter review, or to be printed for classroom distribution. Instructors can add their own slides for additional topics they introduce to the class.
- **Solution Files:** Solutions to "You Do It" exercises and all end-of-chapter exercises are provided on the Instructor Resources CD and on the Course Technology Web site at *www.course.com*. The solutions are password protected.

  Annotated solutions are provided for the multiple-choice Review Questions. For example, if students are likely to debate answer choices, or not understand the choice deemed to be the correct one, a rationale is provided.
- **Distance Learning:** Course Technology is proud to present online test banks in WebCT and Blackboard to provide the most complete and dynamic learning experience possible. Instructors are encouraged to make the most of the course, both online and offline. For more information on how to access the online test bank, contact your local Course Technology sales representative.

ACKNOWLEDGEMENTS

I would like to thank all of the people who helped to make this book a reality, especially Dan Seiter, Development Editor. Dan's suggestions and attention to detail made this a superior book, and his sense of humor made writing it practically painless.

Thanks also to Tricia Coia, Managing Editor; and Heather Furrow, Content Project Manager. I am lucky to work with Tricia and Heather; they are dedicated to producing quality instructional materials.

Thanks to Serge Palladino, John Freitas, and Chris Scriver of the Quality Assurance Department.

Thank you to Dick Grant of Seminole Community College, Sanford, Florida. He provided important technical and pedagogical suggestions based on his classroom use of this book. He possesses the rare combination of excellent teacher and programmer, and he made this book more accurate and more useful to students.

I am also grateful to the many other reviewers who provided comments and encouragement during this book's development, including Karlyn Barilovits, Kaplan University; Kay Chen, Bucks County Community College; Roman Erenshteyn, Goldey-Beacom College; Jeff Hedrington, University of Phoenix-Online; and Aaron Jagers, Louisiana Technical College.

Thanks, too, to my husband, Geoff, who supports me every step of the way. Finally, this book is dedicated to our lifelong friends, George and Mary Profeta.

Joyce Farrell

READ THIS BEFORE YOU BEGIN

The following information will help you as you prepare to use this textbook.

TO THE USER OF THE DATA FILES

To complete the steps and projects in this book, you need data files that have been created specifically for this book. Your instructor will provide the data files to you. You also can obtain the files electronically from the Course Technology Web site by connecting to *www.course.com* and then searching for this book title. Note that you can use a computer in your school lab or your own computer to complete the exercises in this book.

USING YOUR OWN COMPUTER

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

- **Software:** Java SE 6, available from *http://java.sun.com*. (Although almost all of the examples in this book will work with earlier versions of Java, this book was created using Java 6.) The book clearly points out the few cases when an example does not work with earlier versions of Java. You also need a text editor, such as Notepad. A few exercises ask you to use a browser, such as Internet Explorer.
- **Hardware:** To install Java on your computer, the Java Web site suggests at least a Pentium III 500-MHz system with 512 MB of memory and at least 850 MB of disk space. A Pentium IV 1.4-GHz system with 1 GB of memory and 1 GB of disk space is recommended.
- **Data Files:** You cannot complete all the chapters and projects in this book using your own computer unless you have the data files. You can get the data files from your instructor, or you can obtain the data files electronically from the Course Technology Web site by connecting to *www.course.com* and then searching for this book title.

The following material is provided on the CD that comes with this book:

- Sun Microsystems Java SE 6, the Java language, compiler, and runtime environment
- Sun Microsystems Java Application Programming Interface (API) Specification, official documentation for the Java programming language
- The jGRASP integrated development environment for Java
- Code files for all Java program examples contained in the text

VISIT OUR WORLD WIDE WEB SITE

Additional materials designed especially for this book might be available for your course. Periodically search *www.course.com* for more details and updates.

CHAPTER ONE

1

CREATING YOUR FIRST JAVA CLASSES

In this chapter, you will:

- Learn about programming
- Be introduced to object-oriented programming concepts
- Learn about Java
- Analyze a Java application that uses console output
- Add comments to a Java class
- Save, compile, run, and modify a Java application
- Create a Java application using GUI output
- Correct errors and find help

JAVA ON THE JOB, SCENE 1

As you read your e-mail, your heart sinks. There's no denying the message: "Please see me in my office as soon as you are free—Lynn Greenbrier." Lynn is the head of programming for Event Handlers Incorporated, and you have worked for her as an intern for only two weeks. Event Handlers manages the details of private and corporate parties; every client has different needs, and the events are interesting and exciting.

"Did I do something wrong?" you ask as you enter her office. "Are you going to fire me?"

Lynn stands to greet you and says, "Please wipe that worried look off your face. I want to see if you are interested in a new challenge. Our Programming Department is going to create several new programs in the next few months. We've decided that Java is the way to go. It's object-oriented, platform independent, and perfect for applications on the Web, which is where we want to expand our marketing efforts."

"I'm not sure what 'object-oriented' and 'platform independent' mean," you say, "but I've always been interested in computers, and I'd love to learn more about programming."

"Based on your aptitude tests, you're perfect for programming," Lynn says. "Let's get started now. I'll describe the basics to you."

LEARNING ABOUT PROGRAMMING

A computer **program** is a set of instructions that you write to tell a computer what to do. Computers are constructed from circuitry that consists of small on/off switches, so you could create a computer program by writing something along the following lines:

```
first switch—on
second switch—off
third switch—off
fourth switch—on
```

NOTE
Programmers often say that machine language consists of 1s and 0s. What they mean is that you can use 1s and 0s to represent on and off switches.

Your program could go on and on, for several thousand switches. A program written in this style is written in **machine language**, which is the most basic circuitry-level language. For this reason, machine language is a **low-level programming language**, or one that corresponds closely to a computer processor's circuitry. The problems with this approach lie in keeping track of the many switches involved in programming any worthwhile task and in discovering the errant switch or switches if the program does not operate as expected. In addition, the number and location of switches vary from computer to computer, which means that you would need to customize a machine language program for every type of machine on which you want the program to run.

NOTE
In every high-level programming language, the names of memory locations cannot include spaces.

Fortunately, programming has evolved into an easier task because of the development of high-level programming languages. A **high-level programming language** allows you to use a vocabulary of reasonable terms, such as "read," "write," or "add," instead of the sequences of on and off switches that perform these tasks. High-level languages also allow you to assign

intuitive names to areas of computer memory, such as "hoursWorked" or "rateOfPay," rather than having to remember the memory locations (switch numbers) of those values. Java is a high-level programming language.

Each high-level language has its own **syntax**, or rules of the language. 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 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 really are learning the vocabulary and syntax rules for that language.

Using a programming language, programmers write a series of **program statements**, similar to English sentences, to carry out the tasks they want the program to perform. After the program statements are written, high-level language programmers use a computer program called a **compiler** or **interpreter** to translate their language statements into machine code. A compiler translates an entire program before carrying out the statement, or **executing** it, whereas an interpreter translates one program statement at a time, executing a statement as soon as it is translated. 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. Subsequently, the programmer can correct the error and attempt another translation by compiling or interpreting the program again. Locating and repairing all syntax errors is part of the process of **debugging** a program—freeing the program of all errors. 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 are 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 at run time.

NOTE You will learn more about debugging Java programs later in this chapter.

In addition to learning the correct syntax for a particular language, a programmer also must understand computer programming logic. The **logic** behind any program involves executing the various statements and procedures in the correct order to produce the desired results. Although you begin to debug a program by correcting all the syntax errors, it is not fully debugged until you have also fixed all logical errors. For example, you would not write statements to tell the computer program to process data until the data had been properly read into the program. Similarly, you might be able to use a computer language's syntax correctly, but fail to end up with a logically constructed, workable program. Examples of logical errors include multiplying two values when you meant to divide them, or producing output prior to obtaining the appropriate input. Tools that will help you visualize and understand logic are presented in Chapter 5.

NOTE Programmers call some logical 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.

T T F

»TWO TRUTHS AND A LIE: LEARNING ABOUT PROGRAMMING

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 compiler executes each program statement as soon as it is translated, whereas an interpreter translates all of a program's statements before executing any.
3. A syntax error occurs when you misuse a language; locating and repairing all syntax errors is part of the process of debugging a program.

The false statement is #2. A compiler translates an entire program before carrying out any statements, whereas an interpreter translates one program statement at a time, executing a statement as soon as it is translated.

INTRODUCING OBJECT-ORIENTED PROGRAMMING CONCEPTS

Two popular approaches to writing computer programs are procedural programming and object-oriented programming.

PROCEDURAL PROGRAMMING

»NOTE
Procedures are also called modules, methods, functions, and subroutines. Users of different programming languages tend to use different terms. Java programmers most frequently use the term "method."

Procedural programming is a style of programming in which sets of operations are executed one after another in sequence. It involves using your knowledge of a programming language to create names for computer memory locations that can hold values—for example, numbers and text—in electronic form. The named computer memory locations are called **variables** because they hold values that might vary. For example, a payroll program written for a company might contain a variable named `rateOfPay`. The memory location referenced by the name `rateOfPay` might contain different values (a different value for every employee of the company) at different times. During the execution of the payroll program, each value stored under the name `rateOfPay` might have many operations performed on it—the value might be read from an input device, the value might be multiplied by another variable representing hours worked, and the value might be printed on paper. 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`. A procedural program defines the variable memory locations and then **calls** a series of procedures to input, manipulate, and output the values stored in those locations. A single procedural program often contains hundreds of variables and thousands of procedure calls.

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 creating classes, creating objects from those classes, and creating **applications**, which are stand-alone executable programs that use those objects. After being created, classes can

be reused over and over again to develop new programs. Thinking in an object-oriented manner involves envisioning program components as objects that belong to classes and are similar to concrete objects in the real world; then, you can manipulate the objects and have them interrelate with each other to achieve a desired result.

If you've ever used a computer that uses a command-line operating system (such as DOS), and if you've also used a graphical user interface (GUI), such as Windows, then you are familiar with one of the differences between procedural and object-oriented programs. If you want to move several files from a floppy disk to a hard disk, you can type a command at a prompt or command line, or you can use a mouse in a graphical environment to accomplish the task. The difference lies in whether you issue a series of commands, in sequence, to move the three files, or you drag icons representing the files from one screen location to another, much as you would physically move paper files from one file cabinet to another in your office. You can move the same three files using either operating system, but the GUI system allows you to manipulate the files like their real-world paper counterparts. In other words, the GUI system allows you to treat files as objects.

NOTE Do not assume that all object-oriented programs are written to use GUI objects—they are not. However, the difference between command-line and GUI operating systems provides an analogy that helps you envision object-oriented concepts.

Understanding how object-oriented programming differs from traditional procedural programming requires understanding three basic concepts:

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

You can remember these three concepts by remembering the acronym *PIE*, as shown in Figure 1-1.

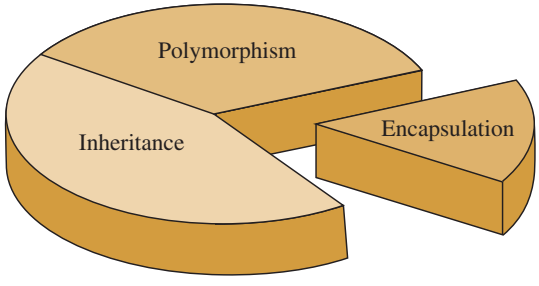

Figure 1-1 The three major features of object-oriented programming

UNDERSTANDING OBJECTS, CLASSES, AND ENCAPSULATION

Objects, both in the real world and in object-oriented programming, are made up of attributes and methods. **Attributes** are the characteristics that define an object; the values contained in attributes differentiate objects of the same class from one another. For example, some of your automobile's attributes are its make, model, year, and purchase price. Other attributes include whether the automobile is currently running, its gear, its speed, and whether it is dirty. All automobiles possess the same attributes, but not, of course, the same values for those attributes. Similarly, your dog has the attributes of its breed, name, age, and

NOTE In object-oriented programming grammar, an object is equivalent to a noun and an attribute is an adjective. Methods are similar to verbs.

whether his shots are current. The values of the attributes of an object are also referred to as the object's **state**.

In object-oriented terminology, a **class** is a term that describes a group or collection of objects with common properties. A **class definition** describes what attributes its objects will have and what those objects will be able to do. An **instance** of a class is an existing object of a class. Therefore, your red Chevrolet `Automobile` with the dent is an instance of the class that is made up of all automobiles, and your Akita `Dog` named Ginger is an instance of the class that is made up of all dogs. Thinking of items as instances of a class allows you to apply your general knowledge of the class to individual members of the class. A particular instance of a class takes its attributes from the general category. 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. You might not know the current state of your friend's `Automobile`—for example, its current speed, or the status of her `Dog`'s shots—but you do know what attributes exist for the `Automobile` and `Dog` classes. Similarly, in a GUI operating environment, you expect each component to have specific, consistent attributes, such as a button being clickable or a window being closable, because each component gains these attributes 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 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.

NOTE 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, so does a class exist before any objects are instantiated from it.

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 would probably be named `Automobile`, and the class for dogs would probably be named `Dog`. However, following this convention is not required to produce a workable program.

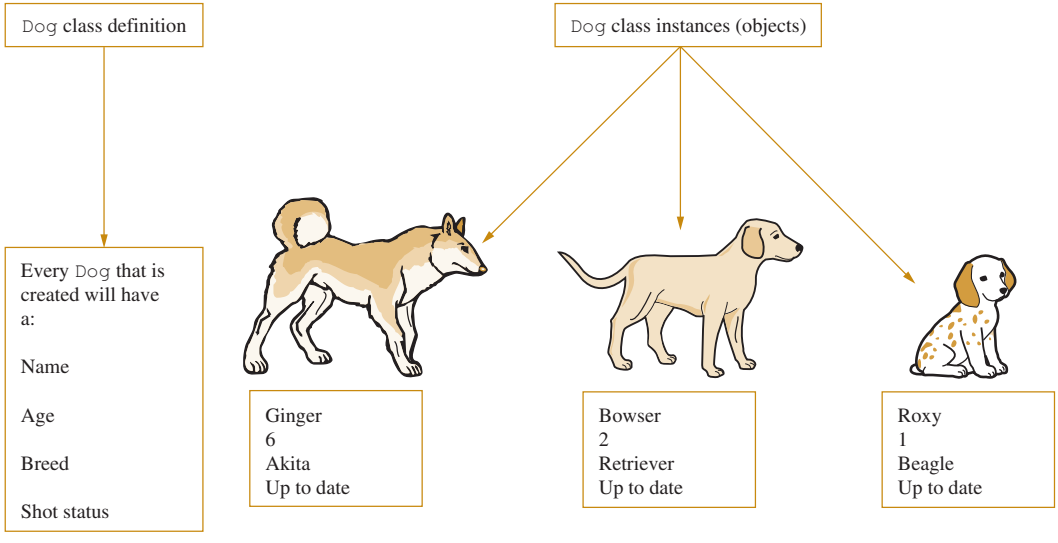

Figure 1-2 A class definition and some objects created from it

Besides attributes, objects can use methods to accomplish tasks. A **method** is a self-contained block of program code, similar to a procedure. An `Automobile`, for example, can move forward and backward. It also can be filled with gasoline or be washed. Some methods can ascertain certain attributes, such as the current speed of an `Automobile` and the status of its gas tank. Similarly, a `Dog` can walk or run, eat food, and get a bath, and there are methods to determine how hungry the `Dog` is or what its name is. GUI operating system components can be maximized, minimized, and dragged.

Like procedural programs, object-oriented programs have variables (attributes) and procedures (methods), but the attributes and methods are encapsulated into objects that are then used much like real-world objects. **Encapsulation** refers to the hiding of data and methods within an object. Encapsulation provides the security that keeps data and methods safe from inadvertent changes. Programmers sometimes refer to encapsulation as using a "black box," or a device that you can use without regard to the internal mechanisms. A programmer can access and use the methods and data contained in the black box but cannot change them.

If an object's methods are well written, the user is 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 figure 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 objects used in object-oriented programs.

UNDERSTANDING INHERITANCE AND POLYMORPHISM

An important feature of object-oriented programs 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 mechanism for and 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`.

»NOTE
Chapters 9 and 10 provide more information about inheritance and polymorphism, and how they are implemented in Java.

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 `Automobile` systems work. 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 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, in English the verb "run" means different things if you use it with "a footrace," a "business," or "a computer." You understand the meaning of "run" based on the other words used with it. Object-oriented programs are written so that the most useful verbs, such as "print" or "save," work differently based on their context. 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.

T T F

TWO TRUTHS AND A LIE: INTRODUCING 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.

LEARNING ABOUT JAVA

NOTE
When programmers call the JVM "hypothetical," they don't mean it doesn't exist. Instead, they mean it is not a physical entity created from hardware, but is composed only of software.

NOTE
Interactive applications are those in which a user communicates with a program by using an input device such as the keyboard or a mouse.

Java was developed by Sun Microsystems as an object-oriented language for general-purpose business applications and for interactive, Web-based Internet applications. Some of the advantages that have made Java so popular in recent years are its security features and the fact that it is **architecturally neutral**, which means that you can use Java to write a program that will run on any platform (operating system).

Java can be run on a wide variety of computers because it does not execute instructions on a computer directly. Instead, Java runs on a hypothetical computer known as the **Java Virtual Machine (JVM)**.

Figure 1-3 shows the Java environment. Programming statements written in a high-level programming language are called **source code**. When you write a Java program, you first construct the source code using a text editor such as Notepad. The statements are saved in a file; then, 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 Java Virtual Machine. Because the Java program is isolated from the operating system, the Java program 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