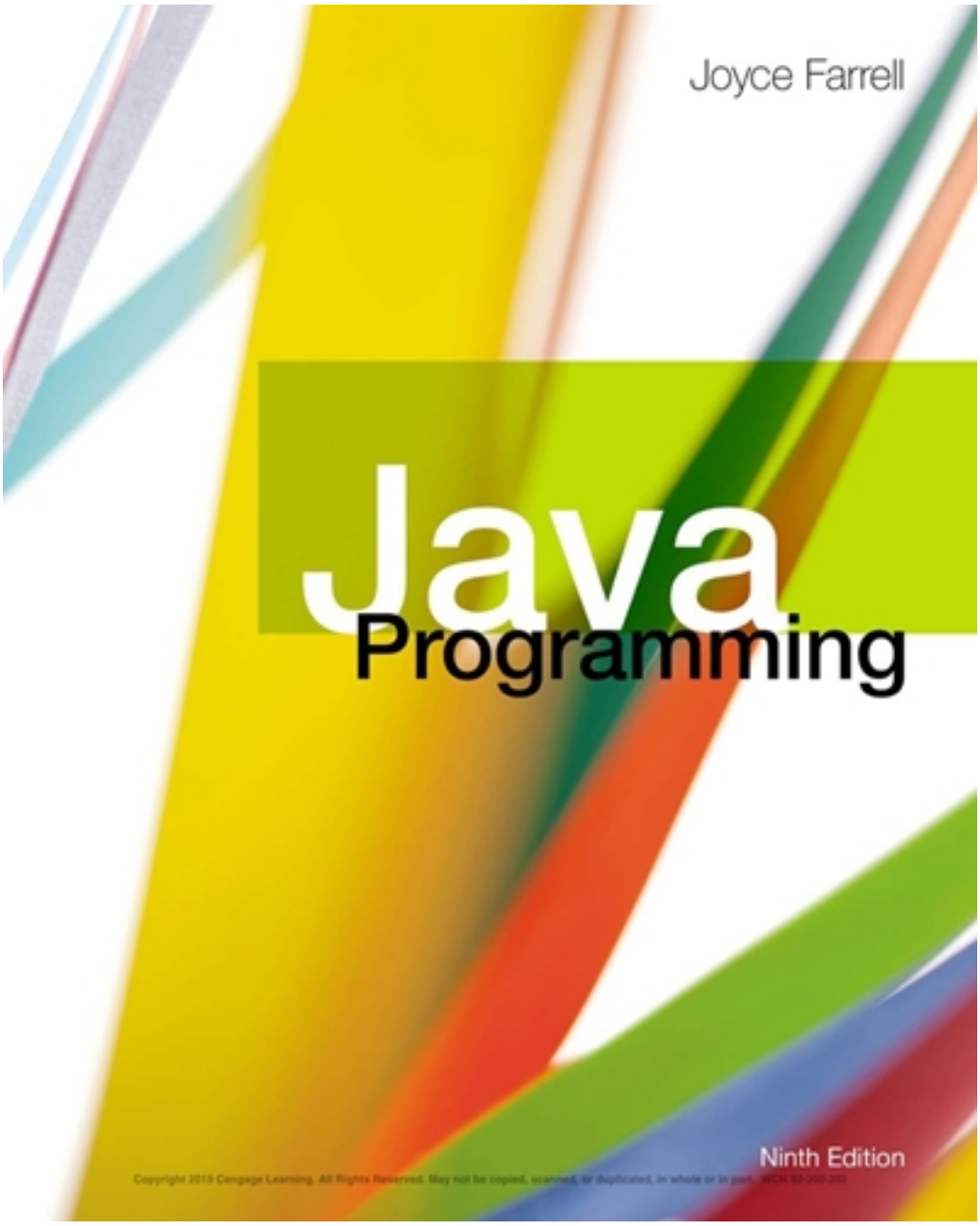
Solutions for Java Programming 9th Edition by Farrell

CLICK HERE TO ACCESS COMPLETE Solutions



Solutions

Chapter 1

Creating Java Programs

A Guide to this Instructor's Manual:

We have designed this Instructor's Manual to supplement and enhance your teaching experience through classroom activities and a cohesive chapter summary.

This document is organized chronologically, using the same headings that you see in the textbook. Under the headings you will find: lecture notes that summarize the section, Teaching Tips, Class Discussion Topics, and Additional Projects and Resources. Pay special attention to teaching tips and activities geared towards quizzing your students and enhancing their critical thinking skills.

In addition to this Instructor's Manual, our Instructor's Resources also contain PowerPoint Presentations, Test Banks, and other supplements to aid in your teaching experience.

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

Chapter 1 provides an introduction to programming. Students will learn basic programming terminology and apply this to the Java programming environment. They will also learn how to create a simple Java application that produces output to the console. The chapter covers the basic components of a Java application and how to compile and run a Java program. Students will also create a simple GUI application.

Objectives

- Define basic programming terminology
- Compare procedural and object-oriented programming
- Describe the features of the Java programming language
- Analyze a Java application that produces console output
- Compile a Java class and correct syntax errors
- Run a Java application and correct logic errors
- Add comments to a Java class
- Create a Java application that produces GUI output
- Find help

Teaching Tips

Learning Programming Terminology

1. Introduce the concept of a **computer program**. Define the terms **machine language**, **low-level programming language**, **high-level programming language**, and **syntax**. Spend more time discussing high-level languages than machine and low-level languages.

2. Discuss Java's rules of syntax. Define a few of the Java **keywords**. Identify the semicolon, curly braces, double quotes, and single quotes. Students quite often do not know the punctuation marks used by programming languages. Compare the rules of syntax to the students' native language. In this comparison, highlight that Java's syntax rules are much more logical and easier to learn.

3. Explain that a program is composed of **program statements**. Explain how a program is translated by a compiler or an interpreter and executed. Provide a description and example of the difference between syntax and logic errors using simple math. A **syntax error** is adding 3, 5, and 7 but getting 9. A logic error is trying to add 3 and 5 but multiplying instead.

4. Define a **compiler** and an **interpreter**, and the role they play. This concept is usually quite foreign to new programming students. Discuss how the Java environment uses both a compiler and an interpreter. How this process works will be discussed later.

5. Work through the process of **debugging** shown in Figure 1-1. Mention that debugging a program is usually harder than writing the program.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 5.

Comparing Procedural and Object-Oriented Programming Concepts

1. Note that two popular programming approaches are **procedural programming** and object-oriented programming.

Procedural Programming

1. Explain how a procedural program is written. Introduce the concepts of **variables** and **procedures**.

2. Explain that individual operations are often combined into a logical unit called a procedure. Programs can contain many procedures, which are also called modules, methods, functions, or subroutines.

Object-Oriented Programming

1. Spend time discussing what **objects** mean in **object-oriented programs**. Select some topic that the students understand and describe possible objects in this topic.

2. Using the types of applications on page 6, discuss how object orientation makes creating these applications easier by reducing the amount of work involved to generate an application. If possible, demonstrate the process to create **GUIs** using an IDE.

Understanding Classes, Objects, and Encapsulation

1. The book introduces the **classes** of `Automobile` and `Dog` on pages 7-8. Define the **attributes** an `Automobile` is likely to have. Describe individual **instances** of class `Automobile`, such as Ford Mustang or Honda Civic. In the same vein, define the attributes a `Dog` class is likely to have. Create a few instances of a `Dog`.

2. Again using the classes of `Automobile` and `Dog`, have the class describe the behavior or methods of an `Automobile`. Examples could include `selfPark`, `drive`, and `reverse`. Similarly, the `Dog` class would have methods of `run`, `play`, and `eat`.

3. Define **encapsulation**. Point out that when a user discusses the concept of an `Automobile`, all of the attributes of this class are included. The speaker does not have to explicitly list the color, year, mileage, and other attributes; they are all included with the concept. Additionally, the functionality of the car is hidden from the driver. One knows how to make the car go forward, but the actual process the drivetrain goes through to make the car move is a mystery. These provide excellent examples of encapsulation in action.

Understanding Inheritance and Polymorphism

1. Continue to use the automobile example to illustrate the concepts of **inheritance** and **polymorphism**. To discuss inheritance, discuss the `Vehicle` class defined in the book on page 9. Cars, boats, and airplanes are all vehicles. Discuss common attributes such as engines and passenger capacity. Then discuss how the inherited classes of cars and airplanes are created by taking the attributes of `Vehicle` and adding attributes specific to that vehicle. For further discussion of inheritance, examine the `SportsUtilityVehicle` class. This is a new class that is created by inheriting attributes of an `Automobile` and a `Truck`.

2. If the students are not grasping the concept, introduce the `Animal` class. Discuss common attributes of `Animals`. Then discuss how the `Dog` and `Cat` classes inherit from `Animal`. For inheritance, have fun by merging the `Dog` and `Cat` classes together, or other odd animal pairs. Discuss the attributes from each animal that will be passed to the inherited class.

3. Define polymorphism. Discuss the methods that all `Vehicles` have in common. Then describe how differently they act. To turn an `Automobile`, you simply turn a wheel. Turning a sailboat requires handling the wind in a particular manner. Both methods result in a turn, but the implementation is different.

Teaching Tip	Note that the concepts of inheritance, encapsulation, and polymorphism will be covered in greater detail as students create more complex programs.

Quick Quiz 1

1. Using a programming language, programmers write a series of ____, similar to English sentences, to carry out the tasks they want the program to perform.
   Answer: program statements

2. True or False: Object-oriented programming is a style of programming in which sets of operations are executed one after another in sequence.
   Answer: False

3. Writing ____ programs involves creating classes, creating objects from those classes, and creating applications.
   Answer: object-oriented

4. A(n) ____ of a class is an existing object of a class.
   Answer: instance

5. True or False: Polymorphism refers to the hiding of data and methods within an object.
   Answer: False

Two Truths and a Lie

1. Discuss the two truths and a lie on page 10.

Features of the Java Programming Language

1. Using Figure 1-3, explain the **Java** environment. Clearly point out the functions of the compiler, **Java Virtual Machine (JVM)**, and **Java interpreter**.

2. Strongly emphasize **"Write once, run anywhere" (WORA)** as one of the benefits of writing a program using Java, which means it can be run on any hardware. This hardware includes all computers, Android phones and tablets, implanted medical devices, and small kitchen appliances. Define **architecturally neutral**, and explain why students can benefit from this feature. If possible, write a short Java program on a Windows computer. Copy and paste the code to a Linux or Mac computer, and run the program with no changes.

3. Discuss the available options for students to write and compile Java. Distinguish **Console applications** from **Windowed applications**.

Teaching Tip	In the early history of Java, most Java programs were applets. As the language has matured, it is now more often used to code complex business applications. Now, Java is used to develop Android applications.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 12.

Analyzing a Java Application that Produces Console Output

1. Using Figure 1-4, introduce the first Java program. Note that this program will print the phrase "First Java application" to the console. You should actually run this in class. Let

students see you build and execute code. Point out each component of the program including classes, methods, string literals, and punctuation.

2. If possible, have students write this program in class.

Understanding the Statement that Produces the Output

1. Explain the statement used to print output to the console:
   `System.out.println("First Java Application");`

Teaching Tip	The `println()` method adds a newline character to the end of the line. The related method `print` will not add the newline character to the end of the line.

2. Define the term **literal string**. Explain the concept of **passing arguments** to methods.

3. Using Figure 1-5, explain that in the program statement `System.out.println`, `println()` is a method, `out` is an object, and `System` is a class. Dots are used to separate the names of the class, object, and method. Dots also show the hierarchy. The `System` class is the parent to the `out` class, which is the parent to the `println` method. Compare the hierarchy to the building and room in which your class meets. OldEngineeringHall.Floor3.Room312 is a good example of how the dots show hierarchy.

4. Point out to students that methods always have `()` after their name. Not all methods will have **arguments** (parameters).

5. The `println` method encapsulates the method of printing onto the standard output device (monitor).

6. Using the code you typed, demonstrate Java's case sensitivity.

Understanding the `First` Class

1. Explain that a Java class is named using an **identifier**. Review the list of rules for identifier names on pages 15 and 16. Table 1-1 lists reserved words in the Java language. Table 1-2 lists valid class names that might be used in a Java program. Point out the **Pascal casing** (**upper camel casing**) used in `UnderGradStudent` and `InventoryItem`. Table 1-3 contains class names that are legal but not recommended. Finally, Table 1-4 lists illegal names.

Teaching Tip	Stress the importance of choosing meaningful identifiers for class, method, and variable names. Each programmer develops his or her own naming style.

2. Introduce the concept of an **access specifier**. The access specifier `public` is used in the `First` class.

3. Refer to Figure 1-6 as you define the parts of a class.

Understanding the `main()` Method

1. Provide an explanation of the `main()` method in the `First` class. Define the keywords **`static`** and **`void`**. Revisit the concept of an access specifier.

2. Refer to Figure 1-7 to explain the parts of the `main()` method.

3. Explain that not all Java classes have a `main()` method, but a `main()` method is required for a Java application.

Teaching Tip	The Java interpreter looks for a method with the name `main()` in order to run a Java class.

4. Review the shell code for any Java application in Figure 1-8. Explain how this can be used as a template to quickly generate new classes.

Indent Style

1. Define **whitespace**. Explain that whitespace in the code is ignored by the compiler. Use the whitespace to make the program readable. Explain that nearly any whitespace style is acceptable.

2. Explain that the curly braces, `{}`, are used in pairs to define a class and a method body. Note that in a properly formatted Java program, the curly brace that opens a method should be placed either at the end of the method header line or on the next line.

3. Contrast the **K & R style** and **Allman style** of brace placement. Explain that both styles are good to use, and students should pick one that they understand.

Saving a Java Class

1. Java classes should be saved in a file with exactly the same name, and a .java extension. For example, the `Baseball` class should be saved in a file called Baseball.java. Refer to Appendix A for more information on saving Java files.

2. Be sure to make multiple copies of your file on separate media.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 20.

You Do It

1. Students should follow the steps in the book on pages 20–22 to create a Java application.

Compiling a Java Class and Correcting Syntax Errors

1. Explain the two-step process that must occur before a program can execute.

2. Define **bytecode**. Explain that the compiler creates the bytecode.

3. Define the Java interpreter contained within the JVM. Explain that the JVM executes the lines of code one at a time.

Compiling a Java Class

1. You must very clearly demonstrate how students are to compile a program using `javac`. Review with students the three possible outcomes on page 22 of the book. Be sure to discuss the exact steps students must follow to compile in your lab.

2. Discuss how to resolve the errors listed on page 23. They are either user, path, or OS problems, and are outside the scope of the Java text.

3. If possible, have students compile the class during your lecture.

Teaching Tip	Note that most programmers use a tool called an Integrated Development Environment (IDE) to create source code, and compile and execute Java applications.

Correcting Syntax Errors

1. Explain to students that syntax errors and **compile-time errors** happen to all programmers, at all levels of expertise. They are not a sign of the student's ineptitude, only part of the nature of programming.

2. Spend a lot of time showing common error messages and how to resolve each error. Be sure to set up scenarios where the error message is incorrect.

3. Suggest to students that they need to solve one error at a time, in order of listing. If possible, set up scenarios that result in multiple errors where the errors are in fact caused by the first error. Forgetting to close a { or " can lead to this type of scenario.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 25.

You Do It

1. Students should follow the steps in the book on pages 25–28 to create a Java application.

Running a Java Application and Correcting Logical Errors

1. Only after a Java program is successfully compiled into bytecode can a program be executed. The file containing the bytecode will be named Classname.class.

Running a Java Application

1. Explain the process of running a program from the command line. Demonstrate the `java Classname` command. Review the same user, path, or OS errors from page 24. Explain why `java Classname.class` is incorrect.

2. Execute the programs you have written in class.

3. If possible, have students execute their Java programs before leaving the room.

Modifying a Compiled Java Class

1. Explain to students that the class file is not changed until the Java source is recompiled. Thus, if a student makes a change to his or her Java source, the old version will still execute until the new version is successfully recompiled.

2. Compiling a modified class requires a recompilation. Repeat the steps of compiling after the source file is saved.

3. Demonstrate in class the process of modifying a file and recompiling it.

Teaching Tip	Periodically, the class file is not overwritten. In this case, perform a **clean build** by deleting the .class file and recompiling. Demonstrate this process in class.

Correcting Logic Errors

1. Define a **logic error**. Explain why this type of error is typically only found at run time. Discuss why logic errors are much more difficult to find and fix than syntax errors.

2. Spend time in class discussing different logic errors. Discuss some that you have written in your career. Emphasize that all programmers will deal with logic errors.

3. Visit the Web site *www.jamesshuggins.com/h/tek1/first_computer_bug.htm* mentioned on page 31 to see the first computer bug from Admiral Grace Hopper.

Teaching Tip	Admiral Grace Hopper developed the programming language COBOL. The world's first programmer was Ada Lovelace. She wrote a small program for Charles Babbage's Analytical Engine in 1842.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 32.

Adding Comments to a Java Class

1. Explain the need for **program comments**. Note that there are three different types of comment styles in the Java language, which are defined on page 33: **line comments**, **block comments**, and **javadoc** comments. Line and block comments are shown in Figure 1-21.

Quick Quiz 2

1. Programming statements written in a high-level programming language are called ____.
   Answer: source code

2. Stand-alone programs are called Java ____.
   Answer: applications

3. ____ are pieces of information that are sent into, or passed to, a method, usually because the method requires the information to perform its task or carry out its purpose.
   Answer: Arguments

4. True or False: Not all classes have a `main()` method.
   Answer: True

5. Line comments start with ____.
   Answer: two forward slashes (//)

Two Truths and a Lie

1. Discuss the two truths and a lie on page 34.

You Do It

1. Students should follow the steps in the book on pages 34–35 to create a Java application with comments.

Creating a Java Application that Produces GUI Output

1. Review the Java program in Figure 1-22. This program uses the `JOptionPane` class to produce a dialog box. The output of the program is shown in Figure 1-23. Explain the components of this program, including the **`import` statement**. Point out that the `showMessageDialog` method takes two arguments, the second being the message displayed to the user.

2. Mention that the `import` statement will be used in future chapters to provide additional functionality to classes.

Two Truths and a Lie

1. Discuss the two truths and a lie on page 37.

You Do It

1. Students should follow the steps in the book on pages 37–38 to create a Java application.

Finding Help

1. Review the places where students can find more information on Java programming.

2. Spend time actually showing students how to read the syntax pages and how to find sample code.

Teaching Tip	Students often misinterpret the term “help,” thinking that the help page will have the answer to their problem, not general instructions on the product.

You Do It

1. Students should follow the steps in the book on page 39 to explore the Java website.

Don't Do It

1. Review this section, discussing each point with the class.

Quick Quiz 3

1. True or False: In Java, if a class is public (that is, if you use the `public` access modifier before the class name), you must save the class in a file with exactly the same name and a `.class` extension.
   Answer: False

2. To compile a file named `First.java`, you type ____ `First.java` and then press Enter.
   Answer: `javac`

3. To run the `First` application from the command line, you type ____ `First`.
   Answer: `java`

4. A(n) ____ is a GUI object resembling a window in which you can place messages you want to display.
   Answer: dialog box

Class Discussion Topics

1. What are the benefits of learning an object-oriented programming language versus a procedural language?

2. What is the best way to ensure that a program is free of logical errors?

Additional Projects

1. Research the Java programming language using the Internet. For what types of programs is Java normally used?

2. Create a Java program that prints out your name to the console and also displays it in a dialog box. Save, compile, and execute your program.

3. Research Android development. What languages are typically used to create Android applications? What resources are needed to develop Android using Java?

Additional Resources

1. The Java Tutorials:
   http://download.oracle.com/javase/tutorial/

2. Java Tutorial:
   www.java2s.com/Tutorial/Java/CatalogJava.htm

3. Object-Oriented Programming:
   http://docs.oracle.com/javase/tutorial/java/concepts/index.html

Key Terms

- **Access specifier:** defines the circumstances under which a class can be accessed and the other classes that have the right to use a class.
- **Allman style:** the indent style in which curly braces are aligned and each occupies its own line; it is named for Eric Allman, a programmer who popularized the style.
- **Application software:** performs tasks for users.
- **Architecturally neutral:** used to write a program that runs on any platform (operating system).
- **Arguments:** information passed to a method so it can perform its task.
- **At run time:** is a phrase that describes the period of time during which a program executes.
- **Attributes:** the characteristics that define an object as part of a class.
- **Block comments:** start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/). Block comments can appear on a line by themselves, on a line before executable code, or on a line after executable code. Block comments can also extend across as many lines as needed.
- **Bug:** Logic or syntax error in a program.
- **Bytecode:** consists of programming statements that have been compiled into binary format.
- **Calls a procedure:** when a procedural program calls a series of procedures to input, manipulate, and output values.
- **Class:** describes a group or collection of objects with common properties.
- **Class body:** set of data and methods that are enclosed within the {} following the class definition.
- **Class definition:** describes what attributes its objects will have and what those objects will be able to do.
- **Clean build:** created when you delete all previously compiled versions of a class before compiling again.
- **Commands:** program statements that are orders to a computer.
- **Comment out:** turn a statement into a comment so the compiler will not execute its command.
- **Compiler**, or **interpreter:** a program that translates language statements into machine code.

- **Compile-time error:** one in which the compiler detects a violation of language syntax rules and is unable to translate the source code to machine code.
- **Computer program:** a set of instructions that you write to tell a computer what to do.
- **Computer simulations:** attempt to mimic real-world activities.
- **Console applications:** support character output to a computer screen in a DOS window.
- **Debugging:** a process to remove errors from a program.
- **Development environment:** set of tools that help you write programs.
- **Dialog box:** a GUI object resembling a window in which you can place messages you want to display.
- **Documentation comments:** used by Javadoc to generate nicely formatted program documentation.
- **Encapsulation:** refers to the hiding of data and methods within an object.
- **Executing:** carrying out a statement.
- **FAQs:** frequently asked questions.
- **Graphical User Interfaces**, or **GUIs:** a graphical environment in which users interact with a program; pronounced "gooeys."
- **Hardware:** the general term for computer equipment.
- **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.
- **Identifier:** a name of a program component such as a class, an object, or a variable.
- **`import` statement:** used to access a built-in Java class that is contained in a package.
- **Inheritance:** the ability to create classes that share the attributes and methods of existing classes but with more specific features.
- **Instance:** an existing object of a class.
- **Instantiation:** to create a class variable.
- **Java:** a programming language developed by Sun Microsystems as an object-oriented language used both for general-purpose business applications and for interactive, Web-based Internet applications.
- **Java API:** the application programming interface; a collection of information about how to use every prewritten Java class.
- **Java interpreter:** a program that checks the bytecode and communicates with the operating system, executing the bytecode instructions line by line within the Java Virtual Machine.
- **Java Virtual Machine (JVM):** a hypothetical (software-based) computer on which Java runs.
- **Javadoc:** a special case of block comments. They begin with a forward slash and two asterisks (/**) and end with an asterisk and a forward slash (*/). You can use Javadoc comments to generate documentation with a program named Javadoc.
- **JDK:** the Java Development Kit.
- **K & R style:** the indent style in which the opening brace follows the header line; it is named for Kernighan and Ritchie, who wrote the first book on the C programming language.
- **Keywords:** the vocabulary of a programming language.
- **Line comments:** start with two forward slashes (//) and continue to the end of the current line. Line comments can appear on a line by themselves or at the end of a line following executable code.

- **Literal string:** a series of characters that appear exactly as entered. Any literal string in Java appears between double quotation marks.
- **Logic:** executing the various statements and procedures in the correct order to produce the desired results.
- **Logic error:** occurs when a program compiles successfully but produces an error during execution.
- **Low-level programming language:** written to correspond closely to a computer processor's circuitry.
- **Machine code:** a program written in circuitry-level language as a series of on and off switches.
- **Machine language:** a program written in circuitry-level language as a series of on and off switches.
- **Method:** a self-contained block of program code, similar to a procedure.
- **Object:** an instance of a class; made up of attributes and methods.
- **Object-oriented programs:** involve creating classes, creating objects from those classes, and creating applications that use those objects. Thinking in an object-oriented manner involves envisioning program components as objects that are similar to concrete objects in the real world; then, you can manipulate the objects to achieve a desired result.
- **Package:** contains a group of built-in Java classes.
- **Pascal casing:** using an uppercase letter to begin an identifier and starting each new word in an identifier.
- **Passing arguments:** sending arguments to a method.
- **Polymorphism:** describes the feature of languages that allows the same word to be interpreted correctly in different situations based on the context.
- **Procedural programming:** a style of programming in which sets of operations are executed one after another in sequence.
- **Procedures:** sets of operations performed by a computer program.
- **Program:** a set of instructions that you write to tell a computer what to do.
- **Program comments:** nonexecuting statements that you add to a Java file for the purpose of documentation.
- **Program statements:** similar to English sentences; they carry out the tasks that programs perform.
- **Properties:** another term for the attributes of a class.
- **`public`:** a reserved keyword; means that the fields and methods are accessible to any other class.
- **Run-time error:** occurs when a program compiles successfully but does not execute.
- **SDK:** a software development kit, or a set of tools useful to programmers.
- **Semantic errors:** occur when you use a correct word in the wrong context in program code.
- **Software:** a general term for computer programs.
- **Source code:** consists of programming statements written in a high-level programming language.
- **Standard output device:** normally the monitor.
- **State:** the values of the attributes of an object.
- **`static`:** a reserved keyword; means that a method is accessible and usable even though no objects of the class exist.

- **Syntax:** the rules of the language.
- **Syntax error:** a programming error that occurs when you introduce typing errors into your program or use the programming language incorrectly. A program containing syntax errors will not compile.
- **System software:** manages the computer.
- **Unicode:** an international system of character representation.
- **Upper camel casing:** another name for Pascal casing
- **Variables:** named computer memory locations that hold values that might vary.
- **`void`:** the keyword used in a method header that indicates that the method does not return any value when it is called.
- **Whitespace:** any combination of nonprinting characters; for example, spaces, tabs, and carriage returns (blank lines).
- **Windowed applications:** create a graphical user interface (GUI) with elements such as menus, toolbars, and dialog boxes.
- **"Write once, run anywhere" (WORA):** a slogan developed by Sun Microsystems to describe the ability of one Java program version to work correctly on multiple platforms.