

# Java Programming, 9e

## Chapter 1

### Creating Java Programs





# Objectives

---

- Upon completion of this chapter you will be able to:
  - 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



# Learning Programming Terminology (1 of 3)

---

- **Computer program**

- A set of written instructions that tells the computer what to do

- **Machine language**

- The most basic circuitry-level language
- A **low-level programming language**



# Learning Programming Terminology (2 of 3)

---

- **High-level programming language**
  - Allows you to use a vocabulary of reasonable terms
- **Syntax**
  - A specific set of rules for the language
- **Program statements**
  - Similar to English sentences
  - **Commands** to carry out program tasks



# Learning Programming Terminology (3 of 3)

---

- **Compiler or interpreter**

- Translates language statements into machine code

- **Syntax error**

- Misuse of language rules
- A misspelled programming language word

- **Debugging**

- Freeing program of all errors

- **Logic errors**

- Also called semantic errors
- Incorrect order or procedure
- The program may run but provide inaccurate output



# Comparing Procedural and Object-Oriented Programming Concepts (1 of 2)

---

- **Procedural programming**

- Sets of operations executed in sequence
- **Variables**
  - Named computer memory locations that hold values
- Procedures
  - Individual operations grouped into logical units

- **Object-oriented programs**

- Create classes
  - Blueprints for an object
- Create objects from classes
- Create applications



# Comparing Procedural and Object-Oriented Programming Concepts (2 of 2)

---

- Object-oriented programming was used most frequently for two major types of applications
  - Computer simulations
  - Graphical user interfaces (GUIs)
    - Not all object-oriented programs are written to use a GUI
- Object-oriented programming differs from traditional procedural programming
  - Polymorphism
  - Inheritance
  - Encapsulation



# Understanding Classes, Objects, and Encapsulation (1 of 3)

---

- **Class**

- Describes objects with common properties
- A definition
- An instance

- **Attributes**

- Characteristics that define an object
- Differentiate objects of the same class
- The value of attributes is an object's state

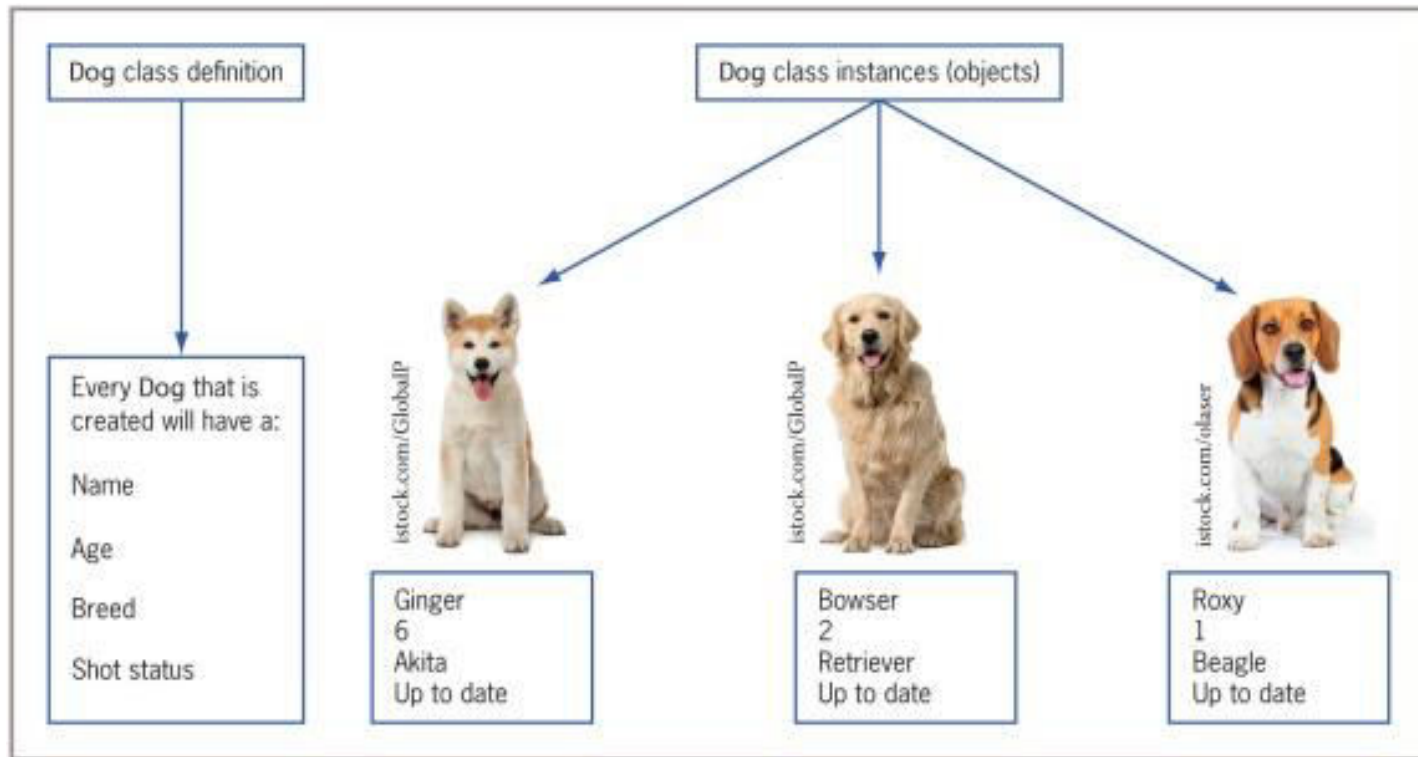
- **Objects**

- Specific, concrete instances of a class





# Understanding Classes, Objects, and Encapsulation (2 of 3)



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



# Understanding Classes, Objects, and Encapsulation (3 of 3)

---

- **Method**

- A self-contained block of program code that carries out an action
- Similar to a procedure

- **Encapsulation**

- Conceals internal values and methods from outside sources
- Provides security
- Keeps data and methods safe from inadvertent changes



# Understanding Inheritance and Polymorphism


---

- **Inheritance**

- An important feature of object-oriented programs
- Classes share attributes and methods of existing classes but with more specific features
- Helps you understand real-world objects

- **Polymorphism**

- Means “many forms”
- Allows the same word to be interpreted correctly in different situations based on context




# Features of the Java Programming Language (1 of 5)

---

- **Java**


- Developed by Sun Microsystems
- An object-oriented language
- General-purpose
- Advantages
  - Security features
  - **Architecturally neutral**



# Features of the Java Programming Language (2 of 5)

---

- **Java (cont'd.)**
  - Can be run on a wide variety of computers
  - Does not execute instructions on the computer directly
  - Runs on a hypothetical computer known as a **Java Virtual Machine (JVM)**
- **Source code**
  - Programming statements written in high-level programming language



# Features of the Java Programming Language (3 of 5)

---

- **Development environment**


- A set of tools used to write programs

- **Bytecode**

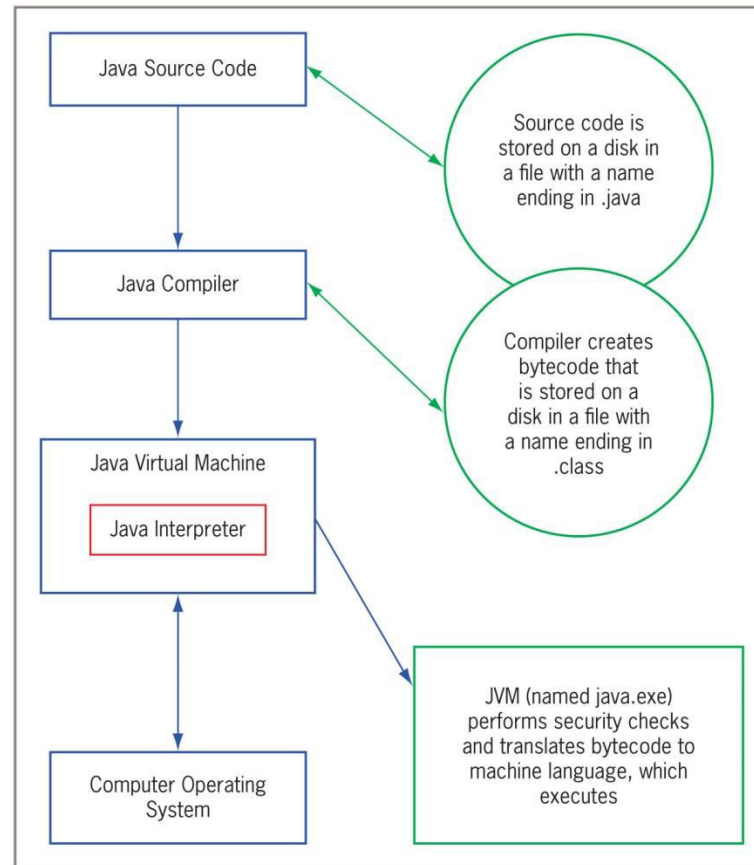
- Statements saved in a file
- A binary program into which the Java compiler converts source code

- **Java interpreter**


- Checks bytecode and communicates with the operating system
- Executes bytecode instructions line by line within the Java Virtual Machine



# Features of the Java Programming Language (4 of 5)



**Figure 1-3** The Java environment



# Features of the Java Programming Language (5 of 5)

---

- **WORA**
  - Write once, run anywhere
- **Console applications**
  - Support character output
- **Windowed applications**
  - Menus
  - Toolbars
  - Dialog boxes





# Analyzing a Java Application that Produces Console Output (1 of 2)

---

- Even the simplest Java application involves a fair amount of confusing syntax
- Print “First Java application” on the screen



# Analyzing a Java Application that Produces Console Output (2 of 2)

---

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

**Figure 1-4** The First class



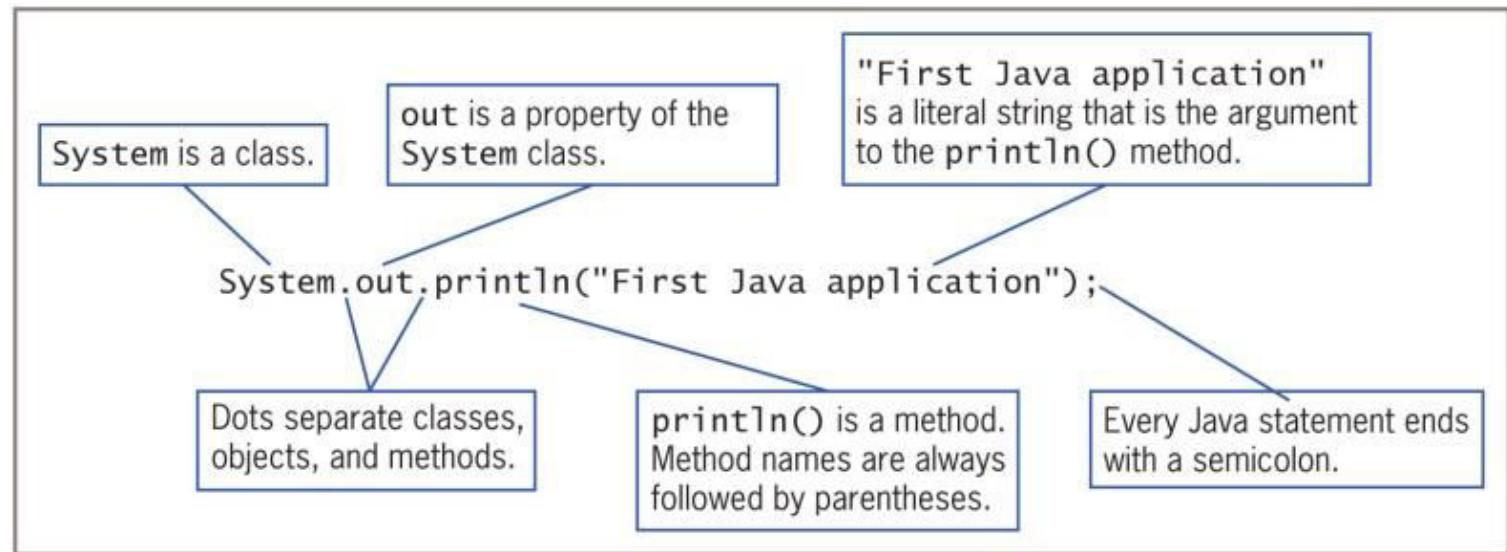
# Understanding the Statement that Produces the Output (1 of 2)

---

- **Literal string**
  - Will appear in output exactly as entered
  - Written between double quotation marks
- **Arguments**
  - Pieces of information passed to a method
- **Method**
  - Requires information to perform its task
- **System class**
  - Refers to the standard output device for a system



# Understanding the Statement that Produces the Output (2 of 2)



**Figure 1-5** Anatomy of a Java statement



# Understanding the First Class (1 of 6)

---

- Everything used within a Java program must be part of a class
- Define a Java class using any name or **identifier**
- Requirements for identifiers
  - Must begin with one of the following:
    - Letter of the English alphabet
    - Non-English letter (such as  $\alpha$  or  $\pi$ )
    - Underscore
    - Dollar sign
  - Cannot begin with a digit



# Understanding the First Class (2 of 6)

---

- Requirements for identifiers (cont'd.)
  - Can only contain:
    - Letters
    - Digits
    - Underscores
    - Dollar signs
  - Cannot be a Java reserved keyword
  - Cannot be `true`, `false`, or `null`
- **Upper Camel casing (Pascal casing)**
  - Each word of an identifier begins with uppercase letter
    - `UnderGradStudent`
    - `InventoryItem`
- **Access specifier**
  - Defines how a class can be accessed



# Understanding the First Class (3 of 6)

**Table 1-1: Java reserved keywords**

abstract	continue	for	new	switch
assert	default	got	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while



# Understanding the First Class (4 of 6)

**Table 1-3: Legal but unconventional and nonrecommended class names in Java**

Class Name	Description
Undergradstudent	New words are not indicated with initial uppercase letters, making this identifier difficult to read
Inventory_Item	Underscore is not commonly used to indicate new words
BUDGET2019	Using all uppercase letters for class identifiers is not conventional
budget2019	Conventionally, class names do not begin with a lowercase letter





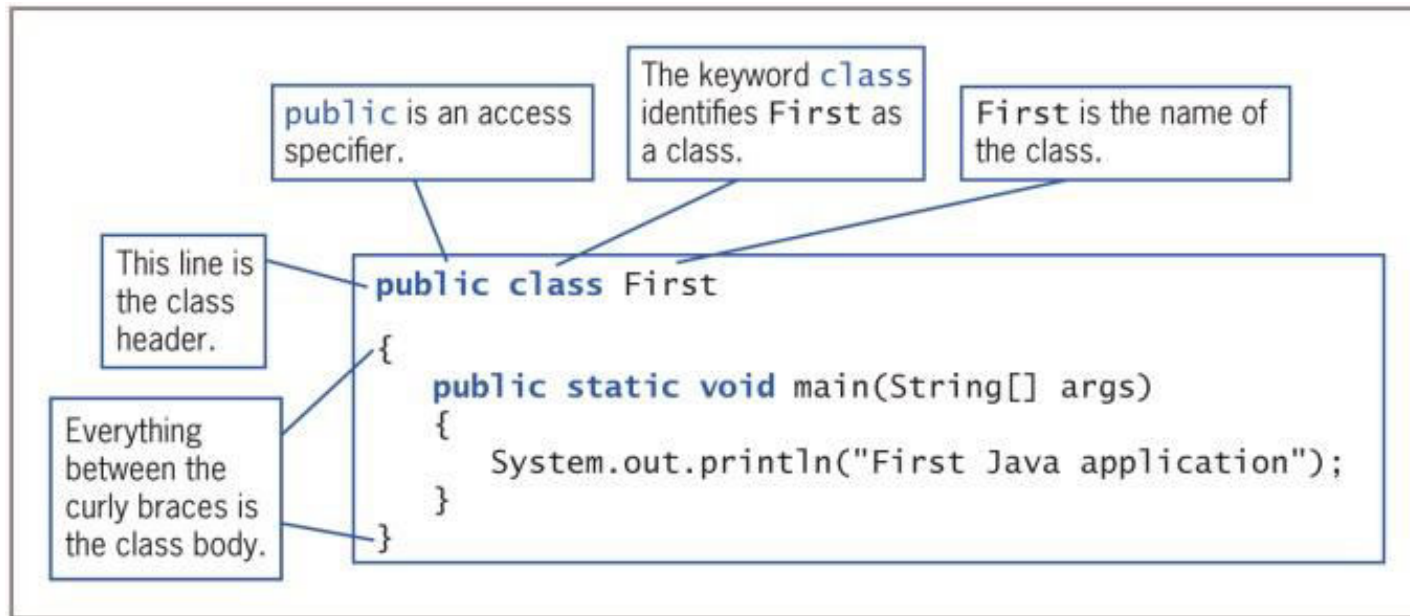
# Understanding the First Class (5 of 6)

**Table 1-4: Some illegal class names in Java**

Class Name	Description
Inventory Item	Space character is illegal in an identifier
class	class is a reserved word
2019Budget	Class names cannot begin with a digit
phone#	The number symbol (#) is illegal in an identifier



# Understanding the First Class (6 of 6)



**Figure 1-6** The parts of a typical class



# Understanding the `main()` Method (1 of 3)

---

- **static**

- A reserved keyword
- Means the method is accessible and usable even though no objects of the class exist

- **void**

- Use in the `main()` method header
- Does not indicate the `main()` method is empty
- Indicates the `main()` method does not return a value when called
- Does not mean that `main()` doesn't produce output



# Understanding the main() Method (2 of 3)

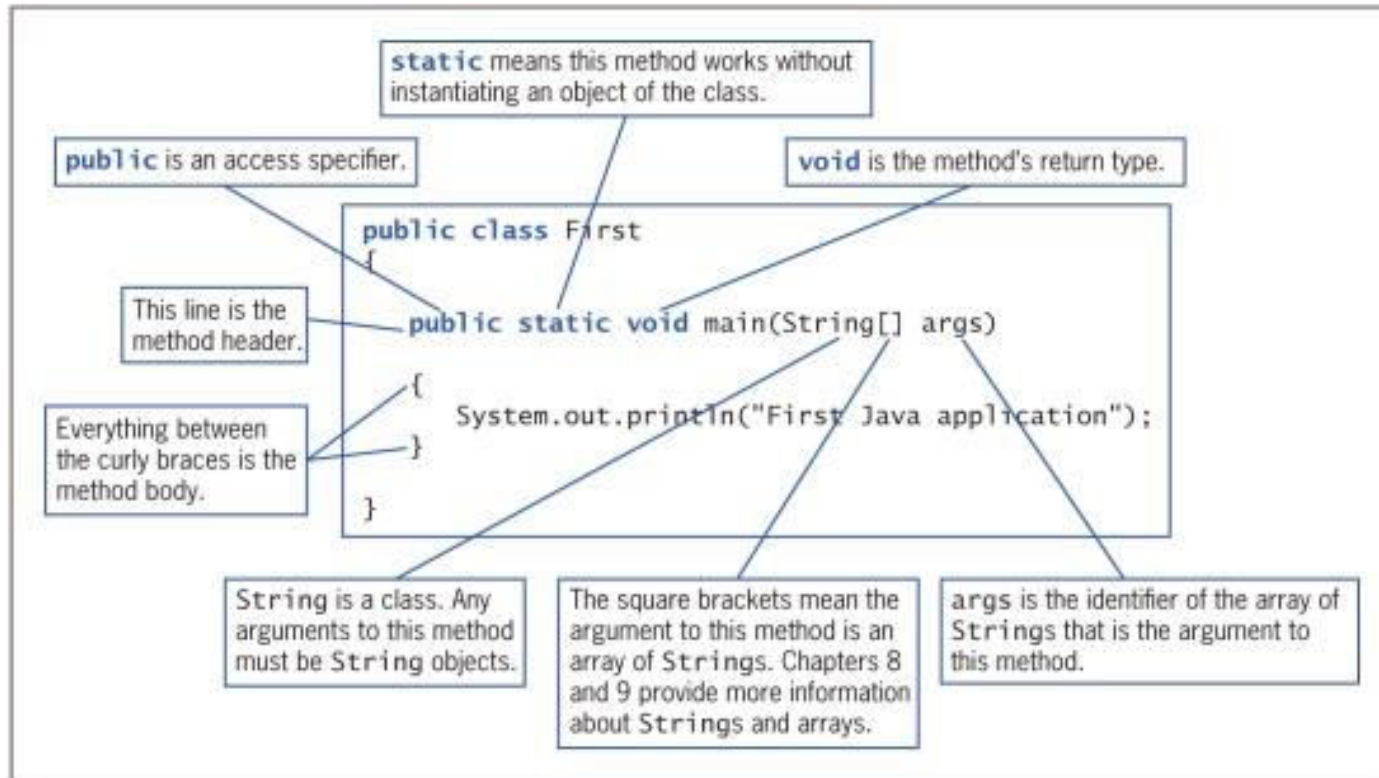


Figure 1-7 The parts of a typical `main()` method



# Understanding the main() Method (3 of 3)

---

```
public class AnyClassName
{
    public static void main(String[] args)
    {
        /***/
    }
}
```

**Figure 1-8** Shell code



# Indent Style

---

- Use **whitespace** to organize code and improve readability
- For every opening curly brace ( { ) in a Java program, there must be a corresponding closing curly brace  
( } )
- Placement of the opening and closing curly braces is not important to the compiler
- **Allman style** used in text



# Saving a Java Class

---

- Saving a Java class
  - Save the class in a file with exactly the same name and .java extension
    - For public classes, class name and filename must match exactly



# Compiling a Java Class and Correcting Syntax Errors (1 of 2)

---

- Compiling a Java class
  - Compile the source code into bytecode
  - Translate the bytecode into executable statements
    - Using a Java interpreter
  - Type `javac First.java`
- Compilation outcomes
  - `javac` is an unrecognized command
  - Program language error messages
  - No messages indicating successful completion





# Compiling a Java Class and Correcting Syntax Errors (2 of 2)

---

- Reasons for error messages
  - Misspelled the command `javac`
  - A misspelled filename
  - Not within the correct subfolder or subdirectory on the command line
  - Improper installation of Java



# Correcting Syntax Errors

---

- The first line of the error message displays:
  - The name of the file where the error was found
  - The line number
  - The nature of the error
- Next lines identify:
  - The symbol
  - The location
- **Compile-time error**
  - The compiler detects a violation of language rules
  - Refuses to translate the class to machine code
- Parsing
  - Compiler divides source code into meaningful portions



# Running a Java Application and Correcting Logical Errors (1 of 2)

---

- Run the application from the command line
  - Type `java First`
- Shows the application's output in the command window
- The class is stored in a folder named Java on the C drive



# Running a Java Application and Correcting Logical Errors (2 of 2)

---

```
C:\Java>java First
First Java application

C:\Java>
```

**Figure 1-17** Output of the First application



# Modifying a Compiled Java Class (1 of 2)

---

- Modify the text file that contains the existing class
- Save the file with changes using the same filename
- Compile the class with the `javac` command
- Interpret the class bytecode and execute the class using the `java` command



# Modifying a Compiled Java Class (2 of 2)

---

```
public class First
{
    public static void main(String[] args)
    {
        System.out.println("My new and improved");
        System.out.println("Java application");
    }
}
```

**Figure 1-18** First class containing output modified from the original version



# Correcting Logic Errors

---

- **Logic error**
  - The syntax is correct but incorrect results were produced when executed
- **Run-time error**
  - Not detected until execution
  - Often difficult to find and resolve



# Adding Comments to a Java Class (1 of 4)

---

- **Program comments**
  - Nonexecuting statements added to a program for documentation
  - Use to leave notes for yourself or others
  - Include the author, date, and class's name or function
- **Comment out** a statement
  - Turn it into a comment
  - The compiler does not translate, and the JVM does not execute its command





# Adding Comments to a Java Class (2 of 4)

---

- Types of Java comments
  - **Line comments**
    - Start with two forward slashes (//)
    - Continue to the end of the current line
    - Do not require an ending symbol
  - **Block comments**
    - Start with a forward slash and an asterisk (/\*)
    - End with an asterisk and a forward slash (\* /)



# Adding Comments to a Java Class (3 of 4)

---

- Types of Java comments (cont'd.)
  - **Javadoc** comments
    - A special case of block comments
    - Begin with a slash and two asterisks (`/ **`)
    - End with an asterisk and a forward slash (`* /`)
    - Use to generate documentation



# Adding Comments to a Java Class (4 of 4)

```
// Demonstrating comments
/* This shows
   that these comments
   don't matter */
System.out.println("Hello"); // This line executes
// up to where the comment started
/* Everything but the println()
   is a comment */
```

The only executable code in this segment is the part of this line up to the semicolon.

**Figure 1-21** A program segment containing several comments



# Creating a Java Application that Produces GUI Output (1 of 3)

---

- `JOptionPane`
  - Produces dialog boxes
- **Dialog box**
  - A GUI object resembling a window
  - Messages placed for display
- **import statement**
  - Use to access a built-in Java class
- **Package**
  - A group of classes



# Creating a Java Application that Produces GUI Output (2 of 3)

```
import javax.swing.JOptionPane;
public class FirstDialog
{
    public static void main(String[] args)
    {
        JOptionPane.showMessageDialog(null, "First Java dialog");
    }
}
```

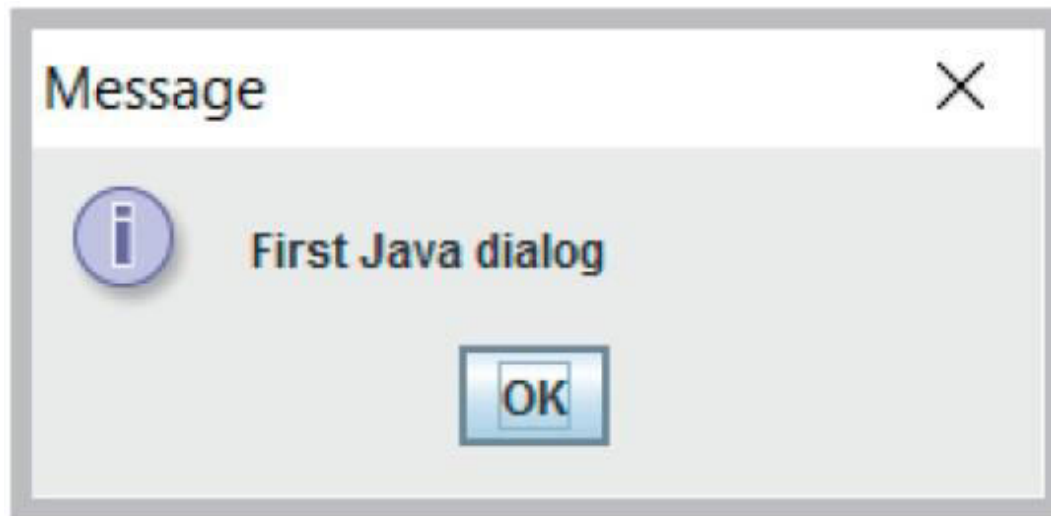
Only these two lines are new to you.

**Figure 1-22** The FirstDialog class



# Creating a Java Application that Produces GUI Output (3 of 3)

---



**Figure 1-23** Output of the FirstDialog application



# Finding Help

---

- **Java API**
  - Also called the Java class library
  - Provides prewritten information about Java classes
- **FAQs** on the Java Web site
- Java Development Kit (**JDK**)
  - A software development kit (**SDK**) of programming tools
  - Free to download



# Don't Do It

---

- Don't forget the file's name must match the class name
- Don't confuse these terms:
  - Parentheses, braces, brackets, curly braces, square brackets, and angle brackets
- Don't forget to end a block comment
- Don't forget that Java is case sensitive
- Don't forget to end every statement with a semicolon
  - Do not end class or method headers with a semicolon
- Don't forgot to recompile when making changes
- Don't panic when you see a lot of compiler error messages
- Don't assume your program is perfect when all compiler errors are eliminated.





# Summary (1 of 3)

---

- Computer program
  - A set of instructions that tells a computer what to do
- Object-oriented programs
  - Classes
  - Objects
  - Applications
- Java Virtual Machine (JVM)
  - A standardized hypothetical computer
- Everything in a Java program must be part of a class



# Summary (2 of 3)

---

- Access specifier
  - A word that defines circumstances under which a class can be accessed
- All Java applications must have a method named `main()`
- Program comments
  - Nonexecuting statements
  - Add to a file for documentation
- `javac`
  - A compile command



# Summary (3 of 3)

---

- java
  - An execute command
- JOptionPane
  - A GUI
  - Provides methods for creating dialogs