NCP2103: Object-Oriented Programming (Java Programming)

Introduction to Java Programming

Module 1

Errol John M. Antonio
Assistant Professor
Computer Engineering Department
University of the East – Manila Campus

Learning Objectives

At the end of this module, students shall 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

Module Outline

I. Programming Terminologies
II. Comparing Procedural and Object-Oriented Programming
III. Features of Java Programming Language
IV. Analyzing Java Application
V. Adding Comments to a Java Program
VI. Saving, Compiling, Running, and Modifying a Java Application
VII. Correcting Errors and Finding Help

I: Programming Terminologies

- **Program**
  - Set of written instructions that tells computer what to do
- **Machine Language**
  - Most basic circuitry-level language
  - Low-level programming language

Programming Terminologies

- **High-Level Programming Language**
  - Allows you to use vocabulary of reasonable terms
- **Syntax**
  - Rules of language
- **Program Statements**
  - Similar to English sentences
  - Carry out tasks of program

Programming Terminologies

- **Compiler or Interpreter**
  - Translates language statements into machine code
- **Syntax Error**
  - Misuse of language
  - Misspelled programming language word
- **Debugging**
  - Freeing program of all errors (bugs)
- **Logic Errors**
  - Also called semantic errors
  - Incorrect order or procedure

Programming Terminologies

- **Semantic Errors**
  - Using a syntax-correct command in a wrong context

Program Development Process

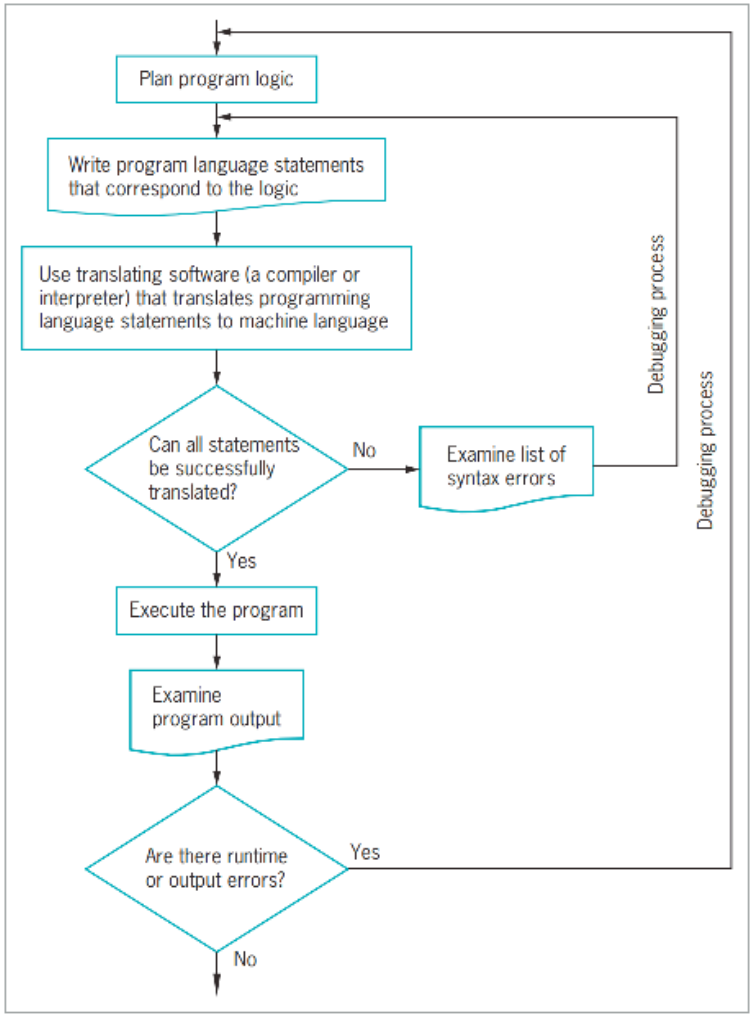


Figure 1-1 The program development process

TWO TRUTHS AND A LIE

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 misuse 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.

II: Comparing Procedural and Object-Oriented Programming Concepts

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

Comparing Procedural and Object-Oriented Programming Concepts

- **Object-oriented programs**
  - Create classes
  - Create objects from classes
  - Create applications

Comparing Procedural and Object-Oriented Programming Concepts (cont'd.)

- Object-oriented programming was used most frequently for two major types of applications
  - Computer simulations
  - Graphical user interfaces (GUIs)
    - Not all object-orientated programs written to use GUI

Comparing Procedural and Object-Oriented Programming Concepts (cont'd.)

- Object-oriented programming differs from traditional procedural programming
  - Basic concepts
    - Polymorphism
    - Inheritance
    - Encapsulation

Understanding Classes, Objects, and Encapsulation

- **Class**
  - Describes objects with common properties
  - Definition
  - Instance
- **Attributes**
  - Characteristics that define object
  - Differentiate objects of same class
  - Value of attributes is object's state
- **Objects**
  - Specific, concrete instance of a class

Understanding Classes, Objects, and Encapsulation (cont'd.)

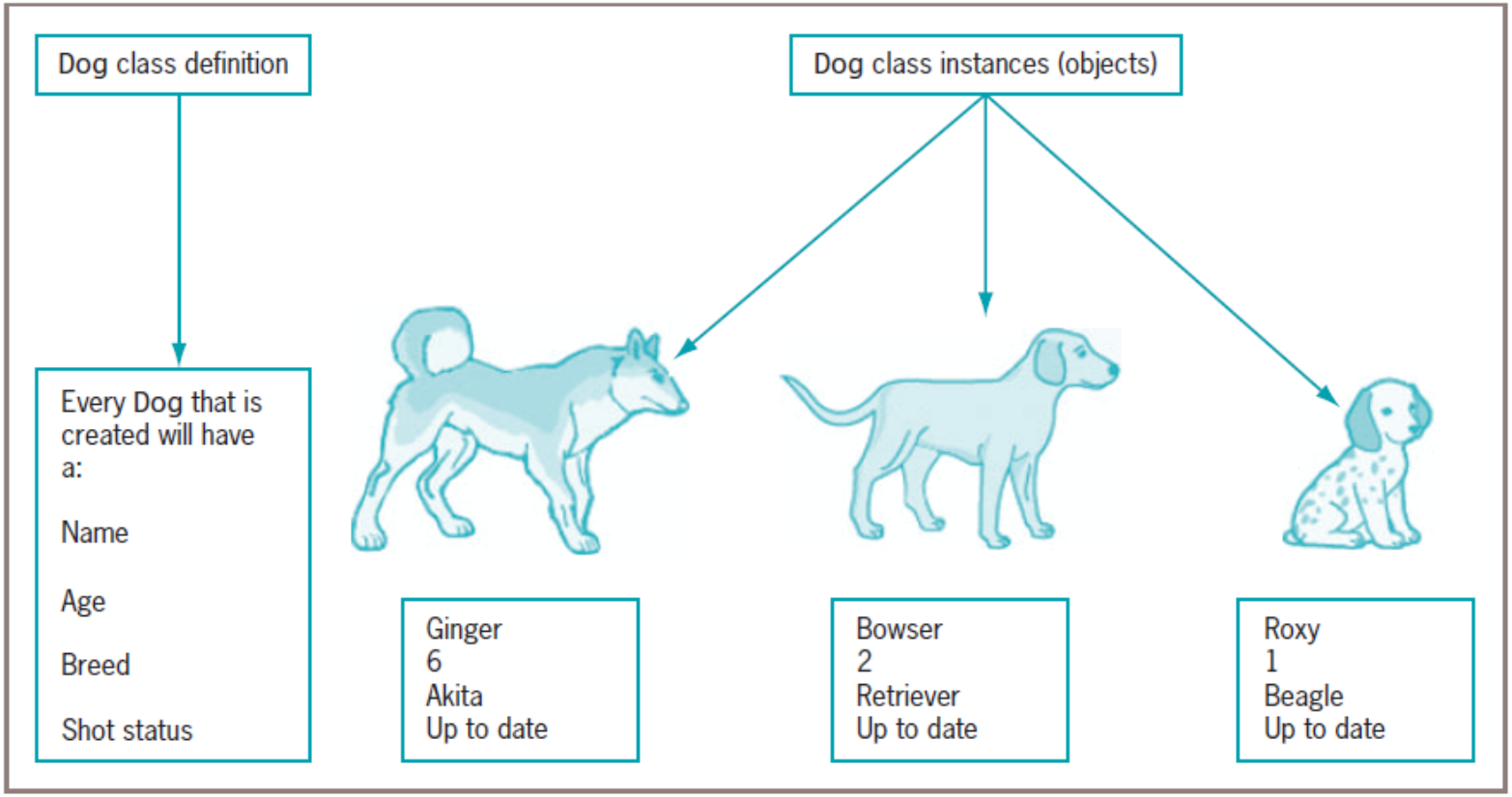


Figure 1-2 A class definition and some objects created from it

Understanding Classes, Objects, and Encapsulation

- **Method**
  - Self-contained block of program code
  - Similar to procedure
- **Encapsulation**
  - Refers to hiding of data and methods within object
  - Provides security
  - Keeps data and methods safe from inadvertent changes

Understanding Inheritance and Polymorphism

- **Inheritance**
  - 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 same word to be interpreted correctly in different situations based on context

TWO TRUTHS AND A LIE

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.

III: Features of the Java Programming Language

- Java programming language
  - Developed by **Sun Microsystems**
  - Initiated by **James Gosling**
  - **R**eleased in 1995
  - The latest release of the Java Standard Edition is Java SE 14.

Features of the Java Programming Language

- Advantages of Java Language
  - WORA (Write Once, Run Anywhere)
  - Object Oriented
  - Platform Independent
  - Simple
  - Secure
  - Architecture-Neutral
  - Portable
  - Robust

Features of the Java Programming Language

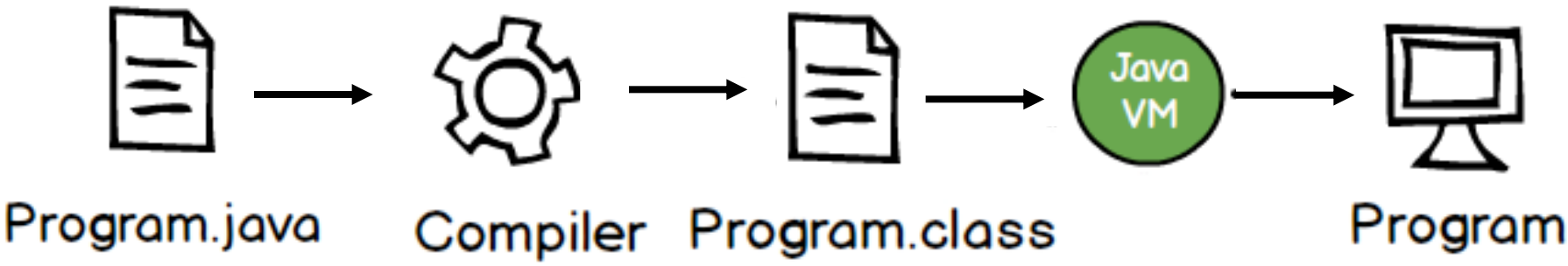
- Can be run on wide variety of computers
- Does not execute instructions on computer directly
- Runs on hypothetical computer known as Java virtual machine (JVM)

- **Source code**
  - Programming statements written in high-level programming language

Features of the Java Programming Language

- **Bytecode**
  - Statements saved in file
  - Java compiler converts source code into binary program
- **Java interpreter**
  - Checks bytecode and communicates with operating system
  - Executes bytecode instructions line by line within Java virtual machine

How Java Works?



How Java Works?

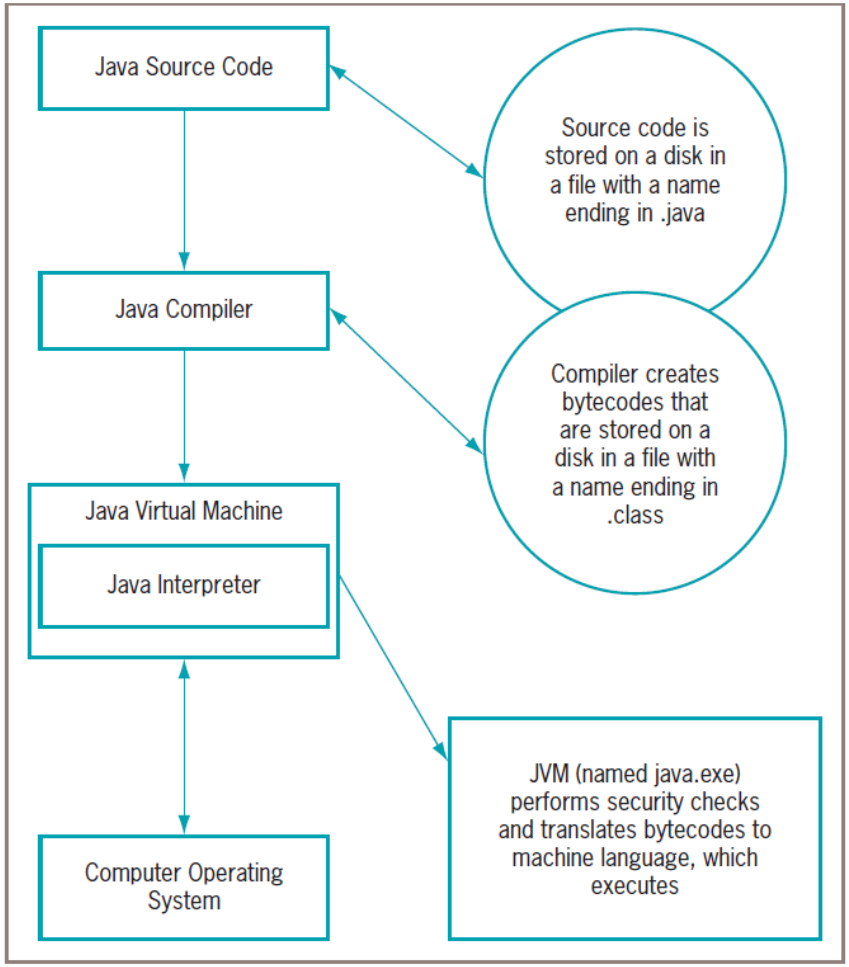
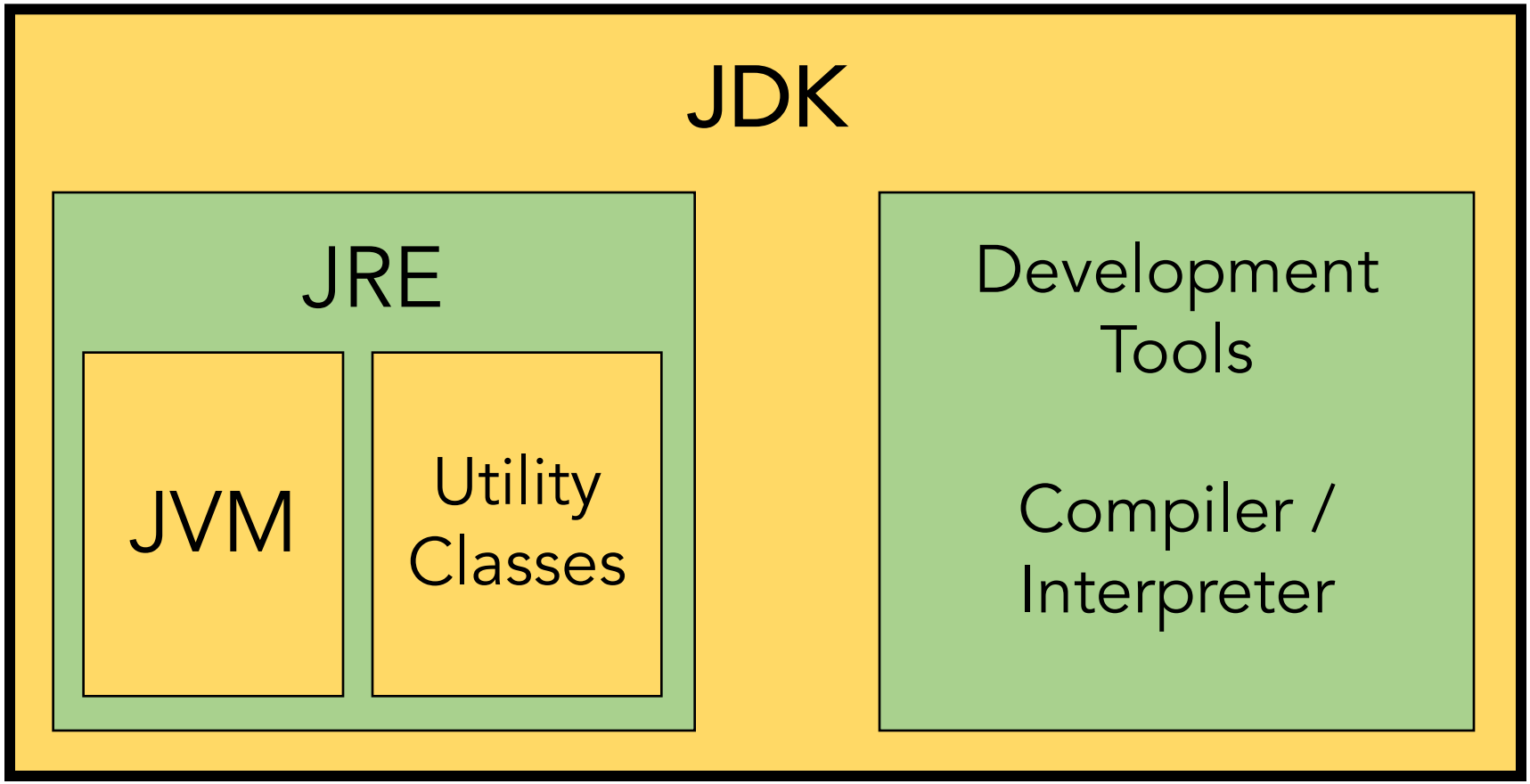


Figure 1-3 The Java environment

JRE versus JDK



JRE versus JDK

- **JRE (Java Runtime Environment)**
  - It is basically the Java Virtual Machine where your Java programs run on. It also includes browser plugins for Applet execution.

- **JDK (Software Development Kit)**
  - Java's full package including **JRE**, and the compilers and tools (like JavaDoc, and Java Debugger) to create and compile programs.

Java Program Types

- **Applets**
  - Programs embedded in Web page
- **Java applications**
  - Called Java stand-alone programs
  - Console applications
    - Support character output
  - Windowed applications (GUI-Based)
    - Menus
    - Toolbars
    - Dialog boxes

TWO TRUTHS AND A LIE

1. Java was developed to be architecturally neutral, which means that anyone can build an application without extensive study.
2. After you write a Java program, the compiler converts the source code into a binary program of bytecode.
3. Java programs that are embedded in a Web page are called applets, while standalone programs are called Java applications.

IV: Analyzing a Java Application that Produces Console Output

- Even simplest Java application involves fair amount of confusing syntax
- Print "First Java application" on screen

Analyzing a Java Application that Produces Console Output


```
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

- **Literal string**
  - Will appear in output exactly as entered
  - Written between double quotation marks
- **Arguments**
  - Pieces of information passed to 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

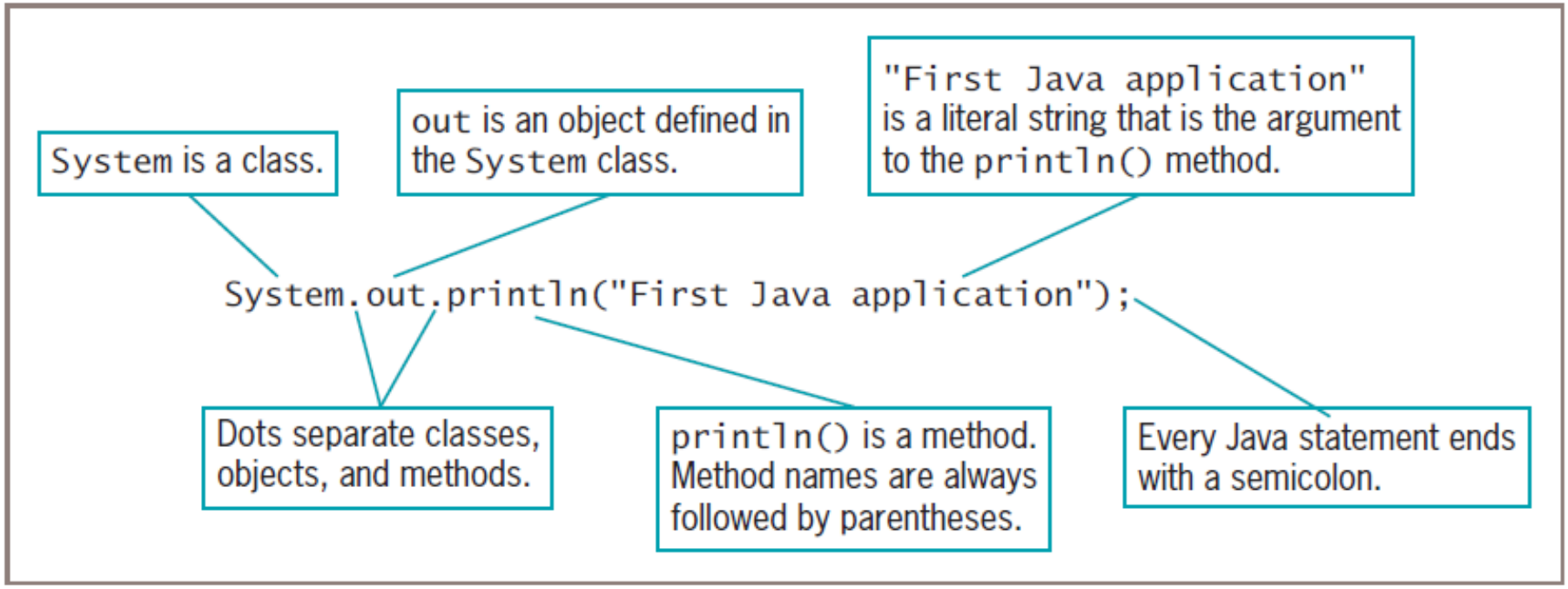


Figure 1-5 Anatomy of a Java statement

Understanding the `First` Class

- Everything used within Java program must be part of a class
- Define Java class using any name or identifier
- Identifiers must begin with:
  - Letter of English alphabet
  - Non-English letter (such as **α** or π)
  - Underscore
  - Dollar sign
  - *Cannot* begin with digit

Understanding the `First` Class (cont'd.)

- Identifiers **can only contain**:
  - Letters
  - Digits
  - Underscores
  - Dollar signs
  - *Cannot* be Java reserved keyword
  - *Cannot* be true, false, or null
- **Access specifier**
  - Defines how class can be accessed

Understanding the `First` Class

abstract	continue	for	new	switch
assert	default	goto	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

Table 1-1 Java reserved keywords

Understanding the `First` Class

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
`BUDGET2012`	Using all uppercase letters for class identifiers is not conventional
`budget2012`	Conventionally, class names do not begin with a lowercase letter

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

Understanding the `First` Class

Class Name	Description
`Inventory Item`	Space character is illegal in an identifier
`class`	`class` is a reserved word
`2012Budget`	Class names cannot begin with a digit
`phone#`	The number symbol (#) is illegal in an identifier

Table 1-4 Some illegal class names in Java

Understanding the `First` Class

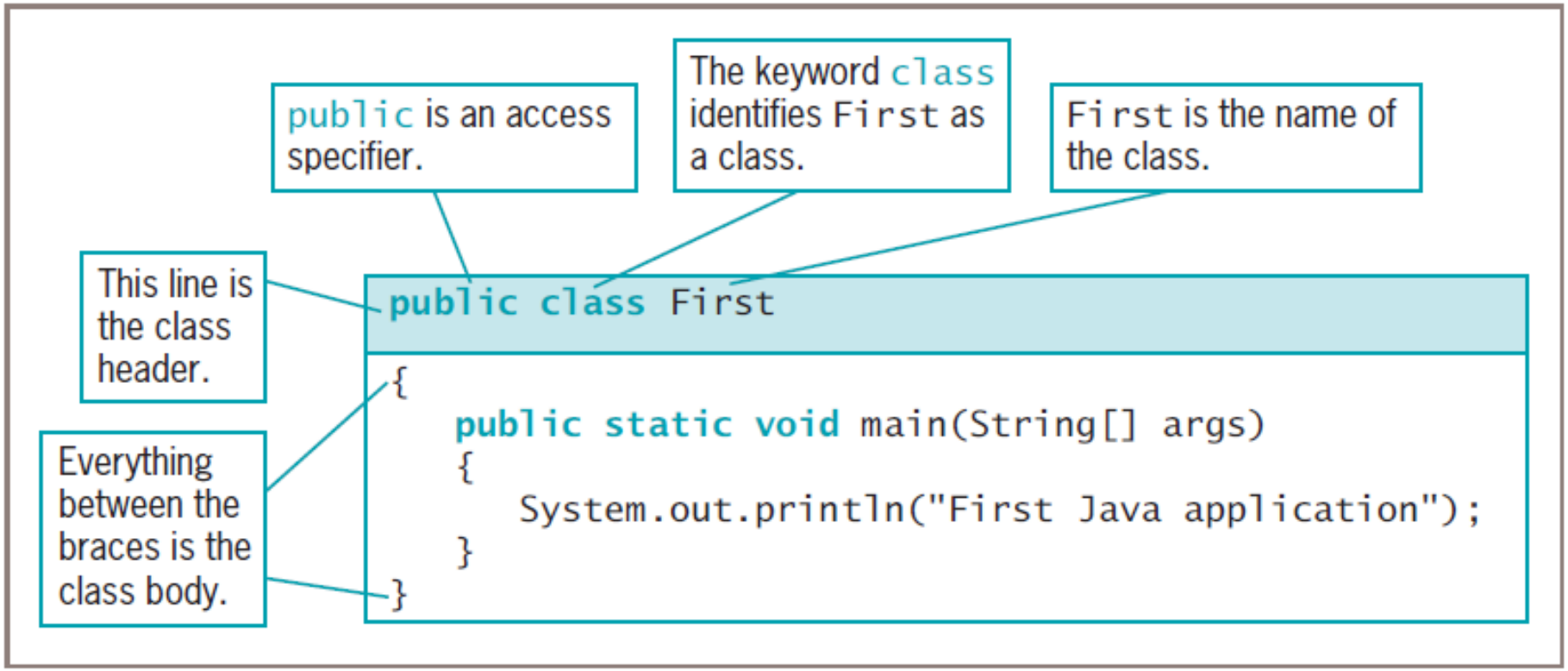


Figure 1-6 The parts of a typical class

Indent Style

- 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

Indent Style

- The **K & R style** is the indent style in which the opening brace follows the header on the **same line**; it is named for Kernighan and Ritchie, who wrote the first book on the C programming language.

- The **Allman style** is 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.

Understanding the `main()` Method

- **`static`**
  - Reserved keyword
  - Means method accessible and usable even though no objects of class exist
- **`void`**
  - Use in `main()` method header
  - Does not indicate `main()` method empty
  - Indicates `main()` method does not return value when called
  - Doesn't mean `main()` doesn't produce output

Understanding the `main()` Method

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

Figure 1-8 Shell code

TWO TRUTHS AND A LIE

1. In the method header `public static void main(String[] args)`, the word public is an access specifier.
2. In the method header `public static void main(String[] args)`, the word `static` means that a method is accessible and usable, even though no objects of the class exist.
3. In the method header `public static void main(String[] args)`, the word `void` means that the main() method is an empty method.

V: Adding Comments to a Java Class

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

Adding Comments to a Java Class

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

Adding Comments to a Java Class

- Types of Java comments
  - **Javadoc comments**
    - Special case of block comments
    - Begin with slash and two asterisks (/**)
    - End with asterisk and forward slash (*/)
    - Use to generate documentation

Adding Comments to a Java Class

```
// 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 */
```

Figure 1-9 A program segment containing several comments

TWO TRUTHS AND A LIE

1. Line comments start with two forward slashes (//) and end with two backslashes (\\ ); they can extend across as many lines as needed.
2. Block comments start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/); they can extend across as many lines as needed.
3. Javadoc comments begin with a forward slash and two asterisks (/**) and end with an asterisk and a forward slash (*/); they are used to generate documentation with a program named javadoc.

VI: Saving, Compiling, Running, and Modifying a Java Application

- **Saving a Java class**
  - Save class in file with exactly same name and **.java** extension
    - For public classes
    - Class name and filename must match exactly
- **Compiling a Java class**
  - Compile source code into bytecode
  - Translate bytecode into executable statements
    - Using Java interpreter
  - Type **`javac First.java`**

Saving, Compiling, Running, and Modifying a Java Application

- **Compilation outcomes**
  - javac unrecognized command
  - Program language error messages
  - No messages indicating successful completion
- **Reasons for error messages**
  - Misspelled command javac
  - Misspelled filename
  - Not within correct subfolder or subdirectory on command line
  - Java not installed properly

VII: Running a Java Application

- Run application from command line
  - Type **`java First`**
- Shows application's output in command window
- Class stored in folder named Java on C drive

Running a Java Application

Figure 1-10 Output of the `First` application

Modifying a Java Class

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

TWO TRUTHS AND A LIE

1. After you write and save an application, you can compile the bytecode to create source code.
2. When you compile a class, you create a new file with the same name as the original file but with a .class extension.
3. Syntax errors are compile-time errors.

VII: Correcting Errors and Finding Help

- Most errors are easily fixed by carefully examining the program as we create it, in just the same way as we fix spelling and grammatical errors when we type an e-mail message.
  - **Compile-time errors**
    - These errors are caught by the system when we compile the program, because they prevent the compiler from doing the translation (so it issues an error message that tries to explain why).

Correcting Errors and Finding Help

- **Run-time errors**
  - These errors are caught by the system when we execute the program, because the program tries to perform an invalid operation (e.g., division by zero).
- **Logical errors.**
  - These errors are (hopefully) caught by the programmer when we execute the program and it produces the wrong answer. Bugs are the bane of a programmer's existence. They can be subtle and very hard to find.

Correcting Errors and Finding Help

- First line of error message displays:
  - Name of file where error found
  - Line number
  - Nature of error
- Next lines identify:
  - Symbol
  - Location
- Compile-time error
  - Compiler detects violation of language rules
  - Refuses to translate class to machine code

You Do It

- Your first application
- Adding comments to a class
- Modifying a class
- Creating a dialog box

Don't Do It

- File's name must match name of class
- 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
- End every statement with semicolon
  - Do not end class or method headers with semicolon
- Recompile when making changes

Summary

- Computer program
  - Set of instructions that tells a computer what to do
- Object-oriented programs
  - Classes
  - Objects
  - Applications
- Java virtual machine (JVM)
  - Standardized hypothetical computer
- Everything in a Java program must be part of a class

Summary (cont'd.)

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

Summary (cont'd.)

- `java`
  - Execute command
- `JOptionPane`
  - GUI
  - Provides methods for creating dialogs

End of Module.

REFERENCE:

Farrell, J. (2016). *Java Programming*. 8th Edition. Course Technology, Cengage Learning.