CS212: Object-Oriented Programming

Introduction

Ayesha Kanwal

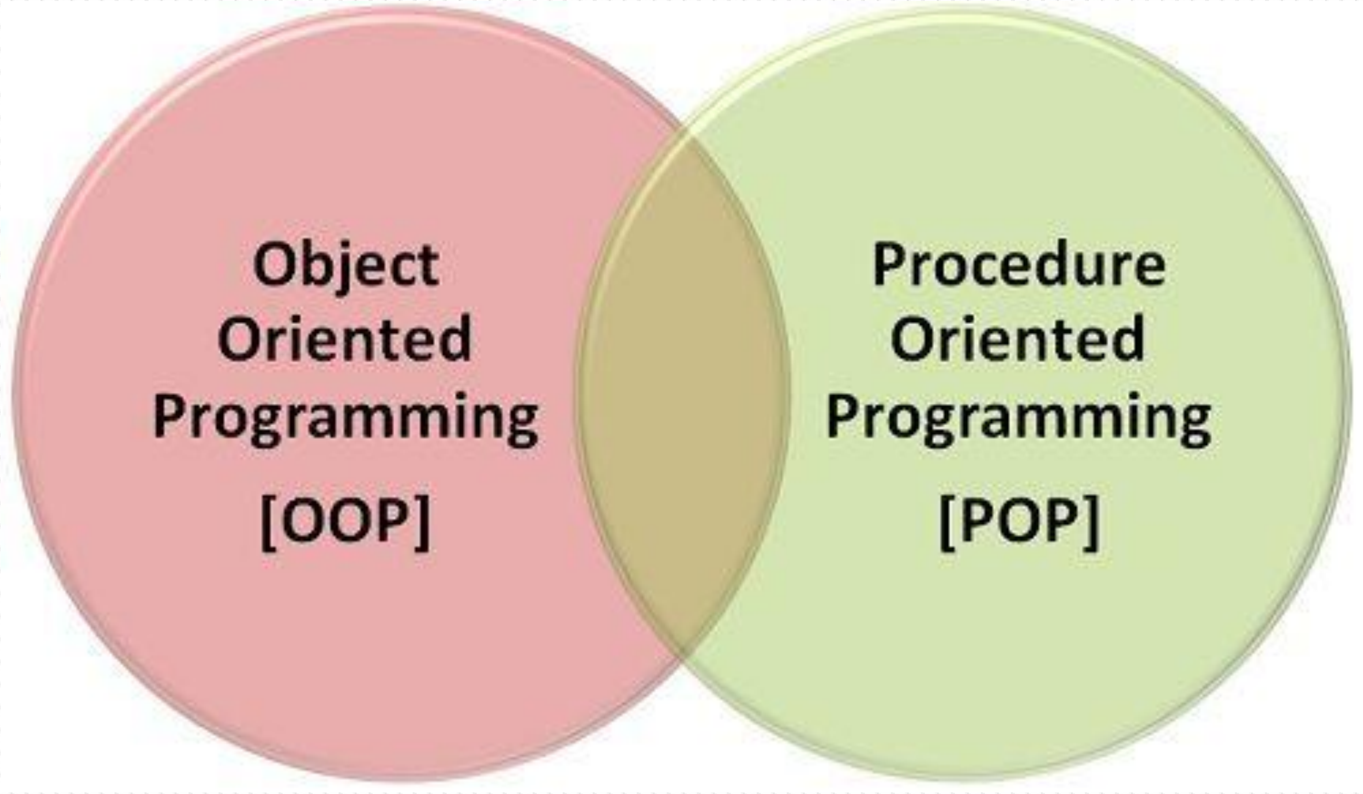
Ayesha.kanwal@seecs.edu.pk

Slides Courtesy (Dr Nazia Perwaiz, SEECS)

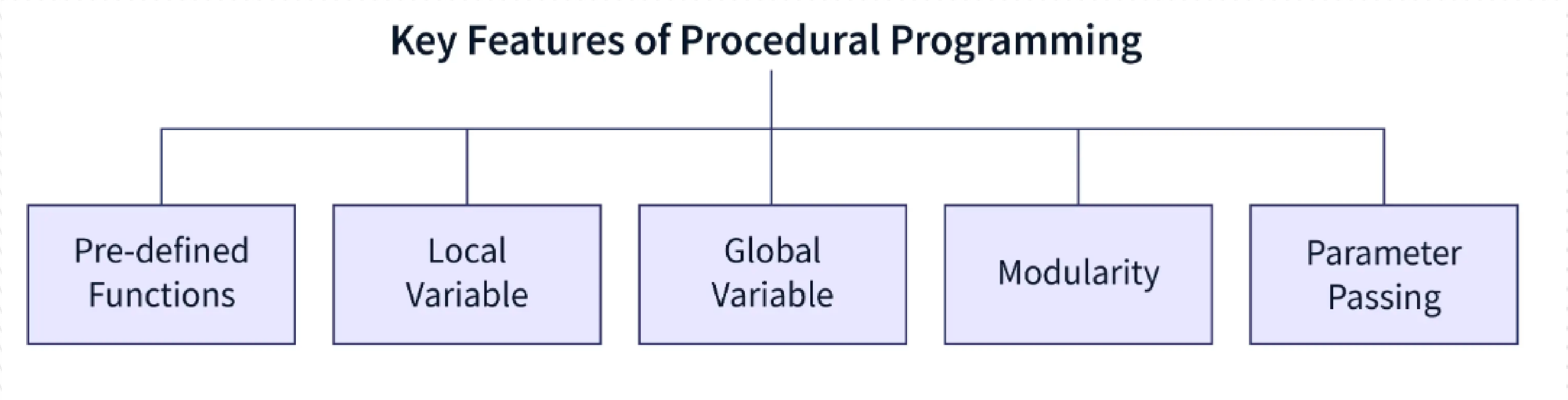
OBJECTIVES

- Understand object-oriented programming basics
- Introduce Object-oriented analysis-and-design (OOAD) process
- Learn a typical Java program development environment
- Write simple Java applications

PROGRAMMING PARADIGMS


Object Oriented Programming [OOP]
Procedure Oriented Programming [POP]

PROGRAMMING PARADIGMS



Key Features of Procedural Programming

Pre-defined Functions

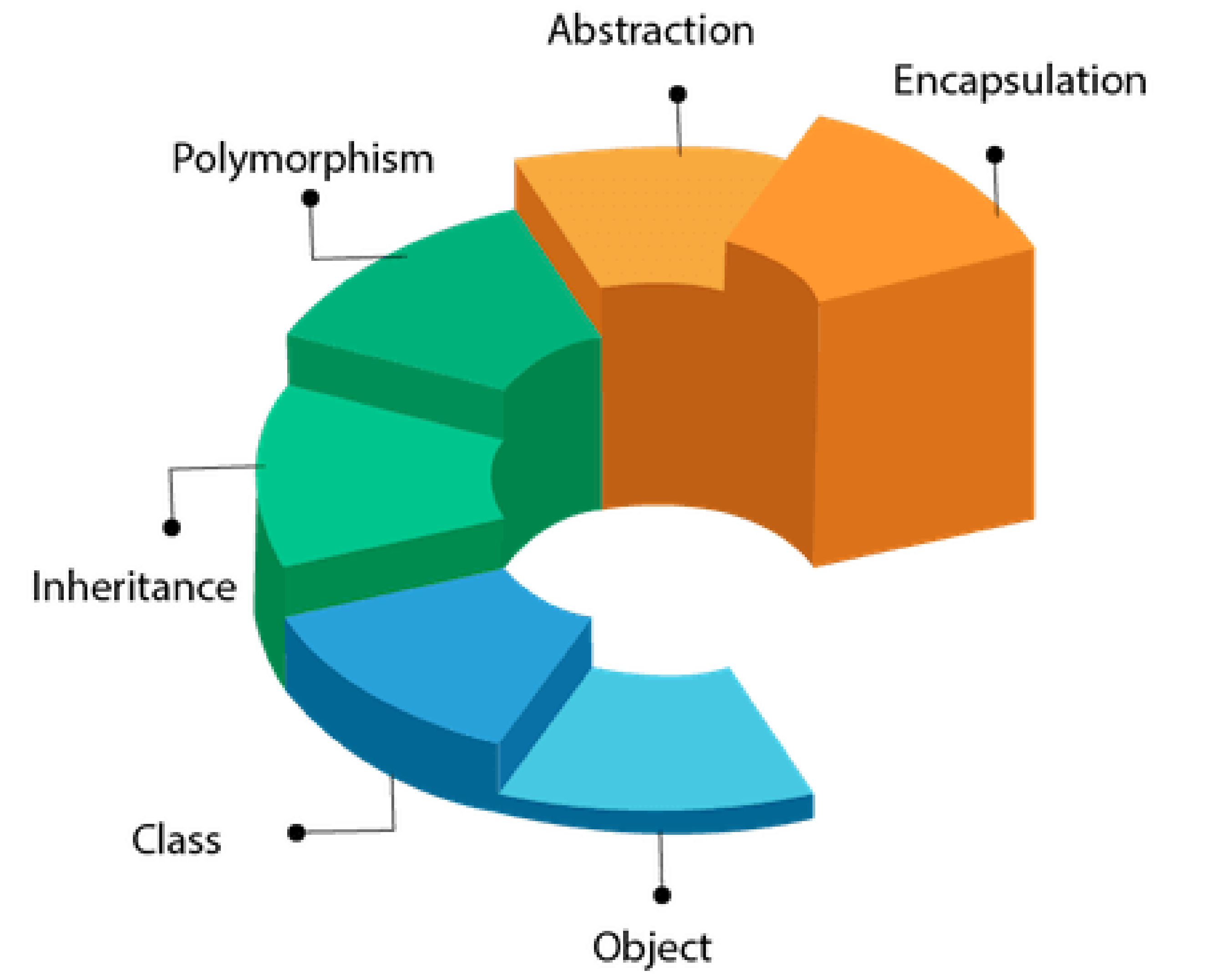
Local Variable

Global Variable

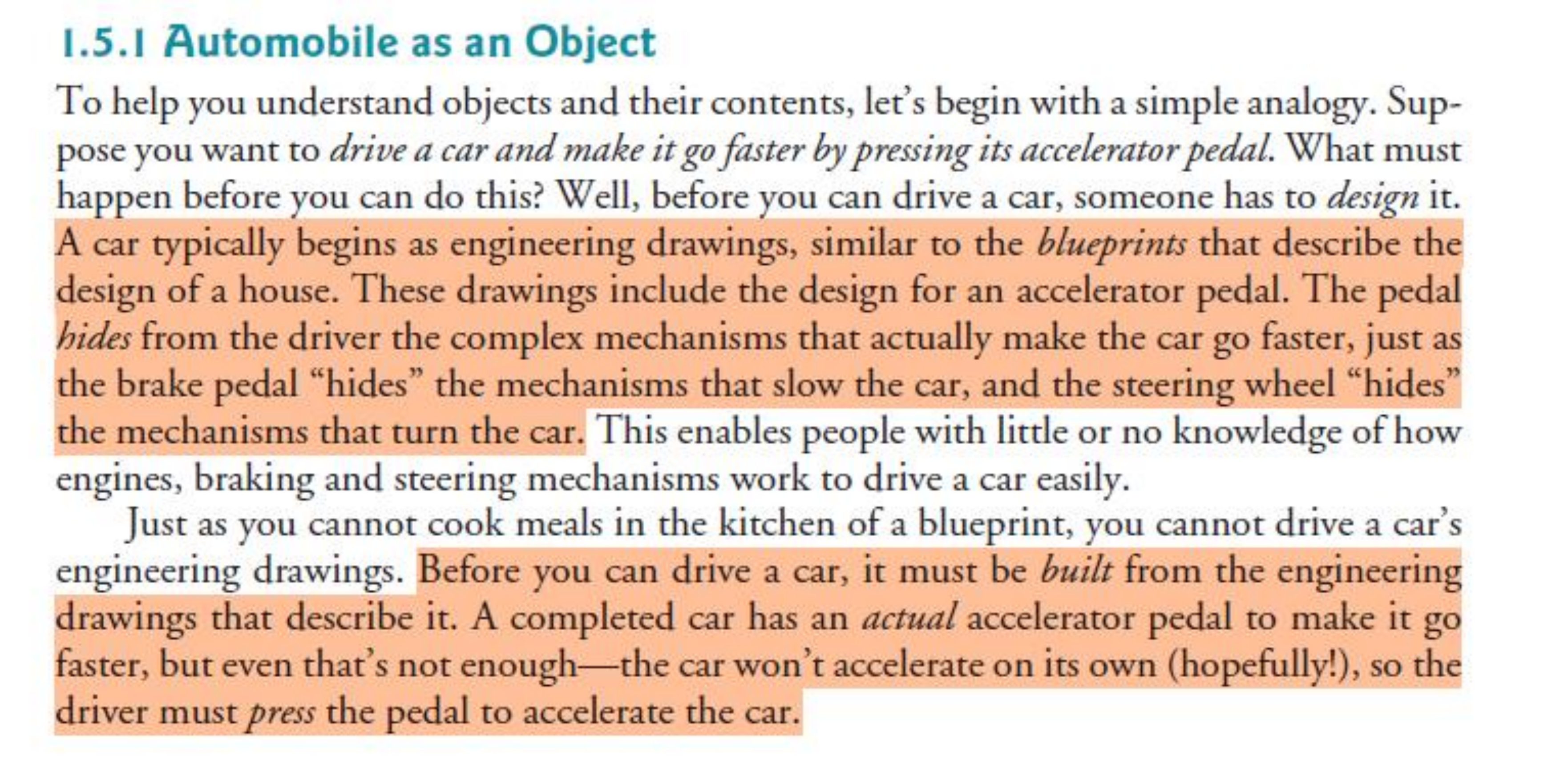
Modularity

Parameter Passing

CORE CONCEPTS OF OOP



CLASS/ OBJECT



1.5.1 Automobile as an Object

To help you understand objects and their contents, let's begin with a simple analogy. Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal*. What must happen before you can do this? Well, before you can drive a car, someone has to *design* it. A car typically begins as engineering drawings, similar to the *blueprints* that describe the design of a house. These drawings include the design for an accelerator pedal. The pedal *hides* from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal "hides" the mechanisms that slow the car, and the steering wheel "hides" the mechanisms that turn the car. This enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.

Just as you cannot cook meals in the kitchen of a blueprint, you cannot drive a car's engineering drawings. Before you can drive a car, it must be *built* from the engineering drawings that describe it. A completed car has an *actual* accelerator pedal to make it go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must *press* the pedal to accelerate the car.

Book reading task: Section 1.5.1 to 1.5.8

OBJECT-ORIENTED ANALYSIS-AND-DESIGN (OOAD) PROCESS

1. A detailed analysis process for determining your project's requirements (i.e., defining **what the system is supposed to do**)

1. Developing a design that satisfies them (i.e., specifying **how the system should do it**).

Standard Tool?

OBJECT-ORIENTED ANALYSIS-AND-DESIGN (OOAD) PROCESS

- The UML **(Unified Modeling Language)**
  - The most widely used graphical scheme for modeling object oriented systems.

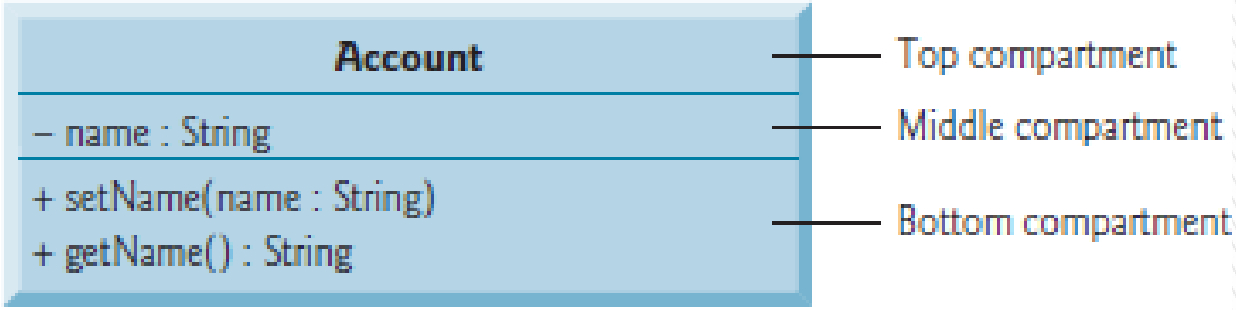


Fig. 3.3 | UML class diagram for class Account of Fig. 3.1.

Java Class Libraries

- Classes
  - Include methods that perform tasks
    - Return information after task completion
  - Used to build Java programs
- Java provides class libraries
  - Known as Java APIs (Application Programming Interfaces)
- Use a building-block approach to create programs. Avoid reinventing the wheel—use existing pieces wherever possible. Called *software reuse*, this practice is central to object-oriented programming.

Typical Java Development Environment

» Java programs normally undergo five phases

1. **Edit**
   - ❖ Programmer writes program/ source code (and stores program on disk)

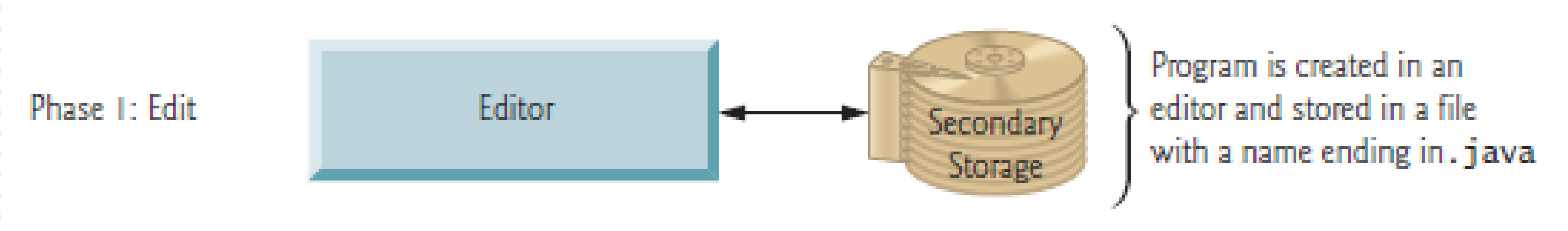

Phase 1: Edit
Editor
Secondary Storage
Program is created in an editor and stored in a file with a name ending in .java

Fig. 1.6 | Typical Java development environment—editing phase.

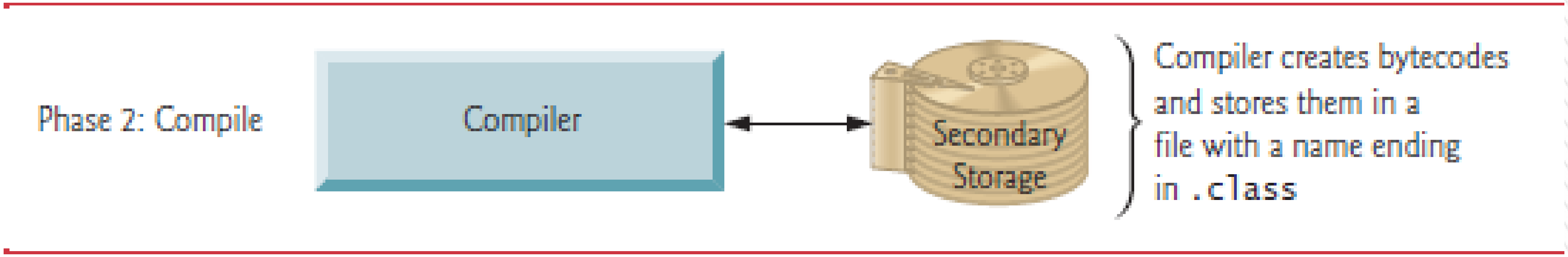
ook reading task: Section 1.9

TYPICAL JAVA DEVELOPMENT ENVIRONMENT

» Java programs normally undergo five phases

2. Compile

❖ Compiler creates bytecodes from source code



Compiler creates bytecodes and stores them in a file with a name ending in `.class`

Fig. 1.7 | Typical Java development environment—compilation phase.

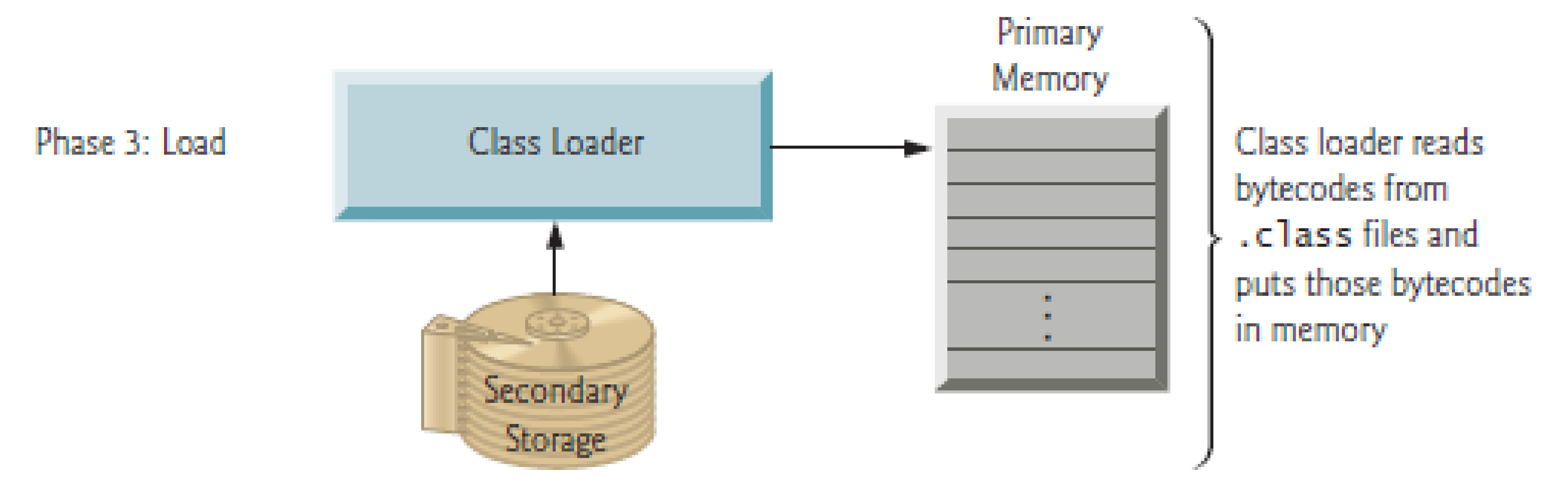
ook reading task: Section 1.9

Typical Java Development Environment

» Java programs normally undergo five phases

3. **Load**

   ❖ the JVM places the program in memory to execute it



Class loader reads bytecodes from `.class` files and puts those bytecodes in memory

Fig. 1.8 | Typical Java development environment—loading phase.

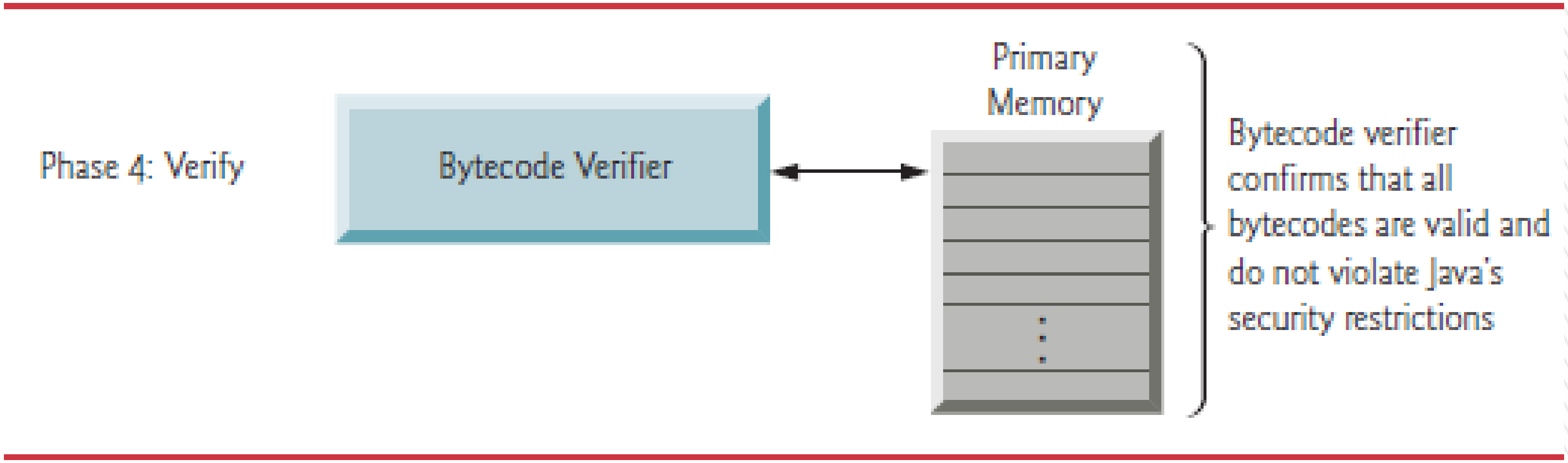
ook reading task: Section 1.9

Typical Java Development Environment

» Java programs normally undergo five phases

4. **Verify**

❖ Bytecode Verifier confirms bytecodes do not violate security restrictions



Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions

Fig. 1.9 | Typical Java development environment—verification phase.

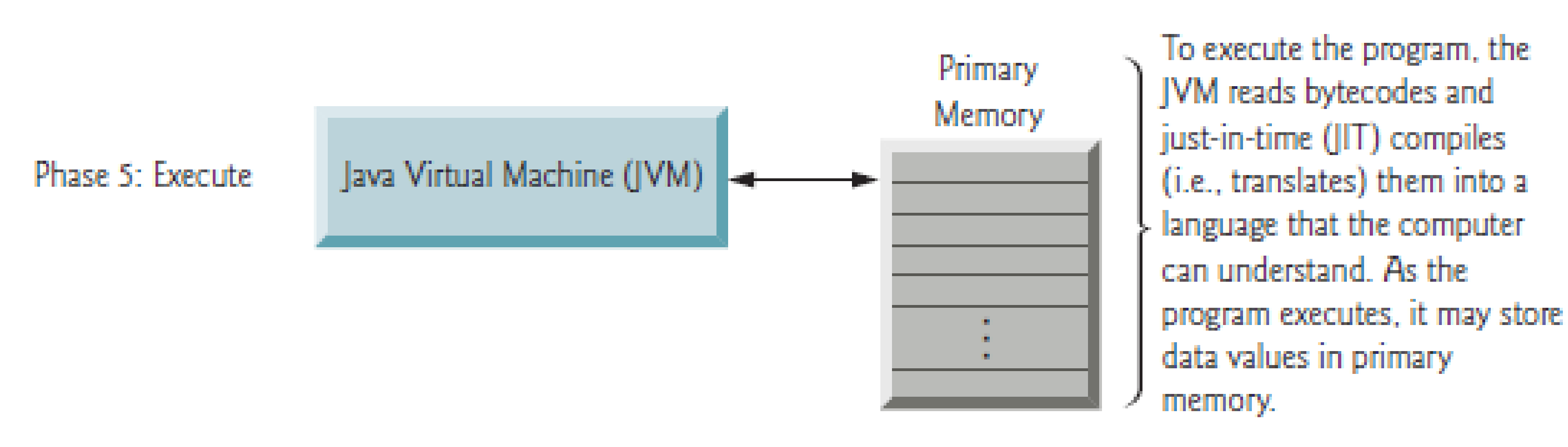
Book reading task: Section 1.9

Typical Java Development Environment

» Java programs normally undergo five phases

5. Execute

❖ JVM translates bytecodes into machine language

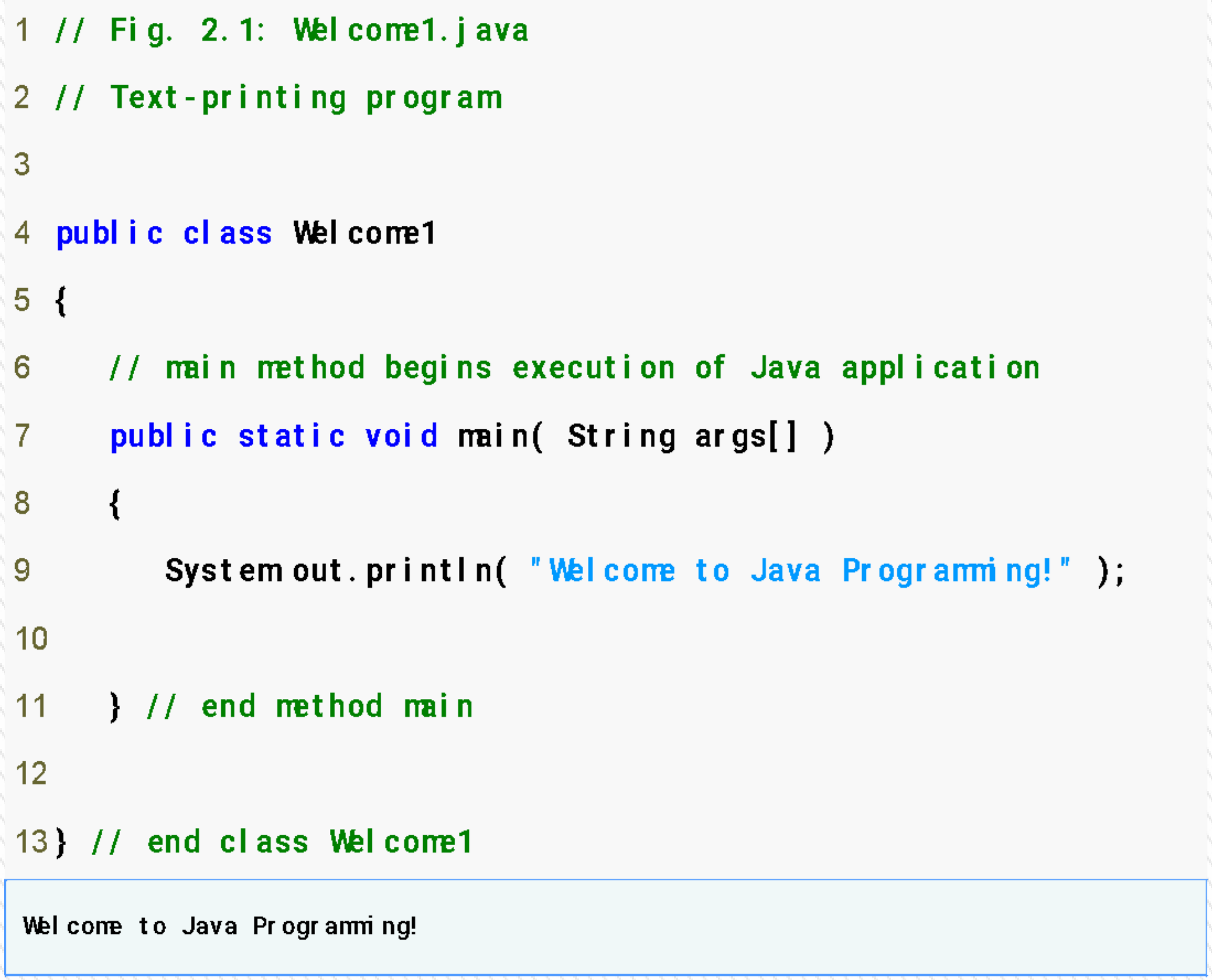


To execute the program, the JVM reads bytecodes and just-in-time (JIT) compiles (i.e., translates) them into a language that the computer can understand. As the program executes, it may store data values in primary memory.

Fig. 1.10 | Typical Java development environment—execution phase.

Book reading task: Section 1.9

First Program in Java



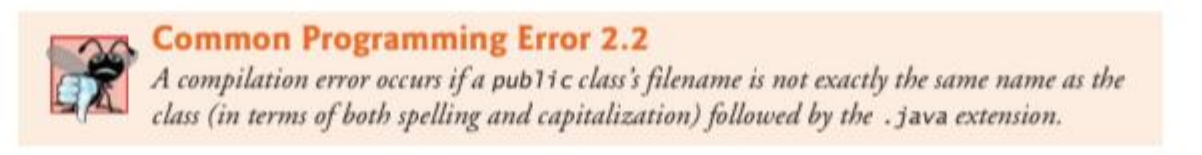
```java
// Fig. 2.1: Welcome1.java
// Text-printing program

public class Welcome1
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.println( "Welcome to Java Programming!" );

   } // end method main

} // end class Welcome1
```

```
Welcome to Java Programming!
```

Key Components:

- Comments
- Line Numbers!
  - For reference only
- Keywords
- White Space
  - Readability
- File Name
  - *.java
  - Same as public class name
- main method
- print statement

Book reading:Section 2.1 - 2.5



Common Programming Error 2.2

A compilation error occurs if a `public` class's filename is not exactly the same name as the class (in terms of both spelling and capitalization) followed by the `.java` extension.

FIRST PROGRAM IN JAVA – IMPORTANT POINTS

- Every Java program has at least one user-defined class

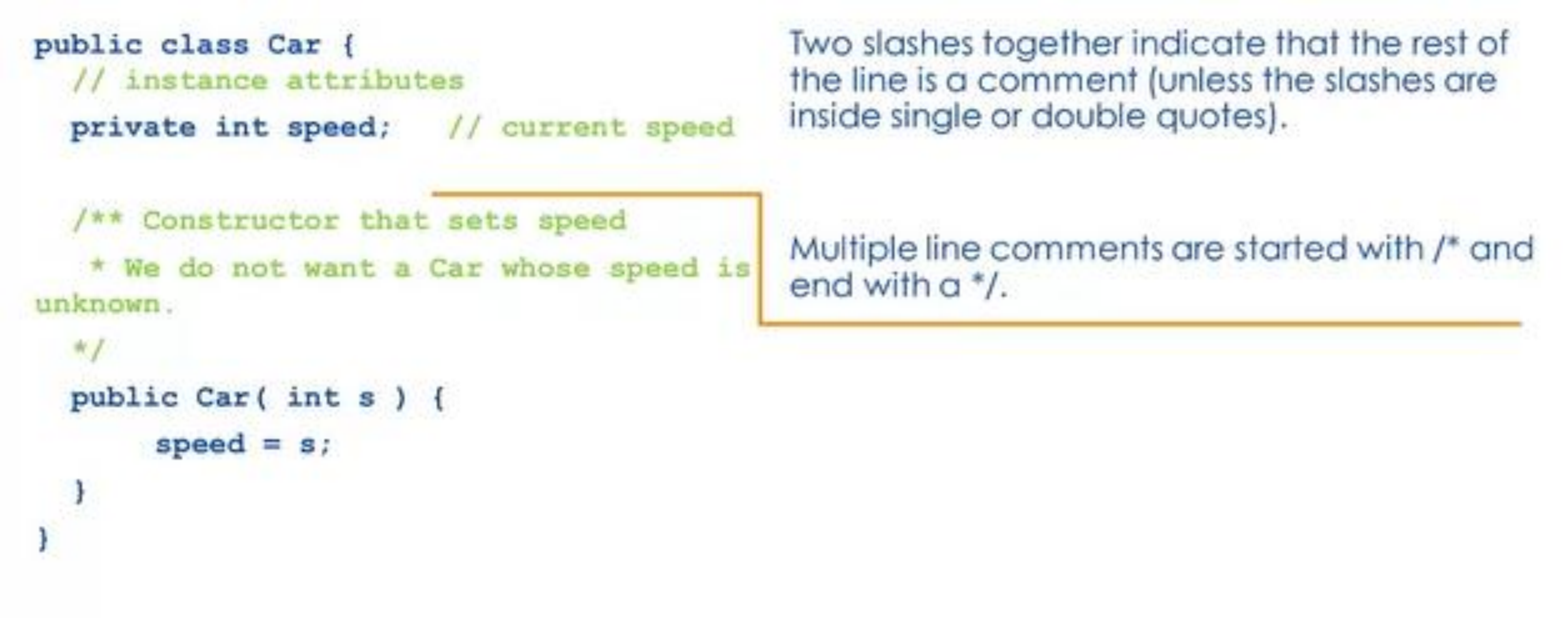
- Keyword: words reserved for use by Java
  - class keyword followed by class name

- Naming classes: capitalize every word (camel case)
  - SampleClassName

- Java identifiers
  - Series of characters consisting of letters, digits, underscores (_) and dollar signs ($)
  - Does not begin with a digit, has no spaces
  - Examples: Welcome1, $value, _value, button7
    - 7button is invalid
  - Java is case sensitive (capitalization matters)
    - a1 and A1 are different

Book reading:Section 2.1 - 2.5

FIRST PROGRAM IN JAVA – IMPORTANT POINTS



```
public class Car {
  // instance attributes
  private int speed;    // current speed

  /** Constructor that sets speed
   * We do not want a Car whose speed is
unknown.
  */
  public Car( int s ) {
       speed = s;
  }
}
```

Two slashes together indicate that the rest of the line is a comment (unless the slashes are inside single or double quotes).

Multiple line comments are started with /* and end with a */.

Now we have three different kinds of comments:

1. One is a double slash, that's the example here. It's a **single line comment**, // instance variables or // current speed.
2. **Multiple line comments** start with a slash asterisk and eventually end with an asterisk slash.
3. That is a special form of multiline comment called **javadoc comments, delimited by /** and */.** These provide the documentation for that source code. (run this command: javadoc Car.java)

Book reading:Section 2.1 - 2.5

Good Programming Practice

- » By convention, always begin a **class name’s identifier** with a **capital letter** and start each subsequent word in the identifier with a capital letter.
- » Java programmers know that such identifiers normally represent Java classes, so naming your classes in this manner makes your programs **more readable.**

Common Programming Errors

- Java is case sensitive. Not using the proper uppercase and lowercase letters for an identifier normally causes a compilation error. Java is case sensitive. Not using the proper uppercase and lowercase letters for an identifier normally causes a **compilation error**.
- It is an error for a public class to have a file name that is not identical to the class name (plus the .java extension) in terms of both spelling and capitalization.
- It is a syntax error if braces do not occur in matching pairs.

Book reading:Section 2.1 - 2.5

Common Error Prevention Tips

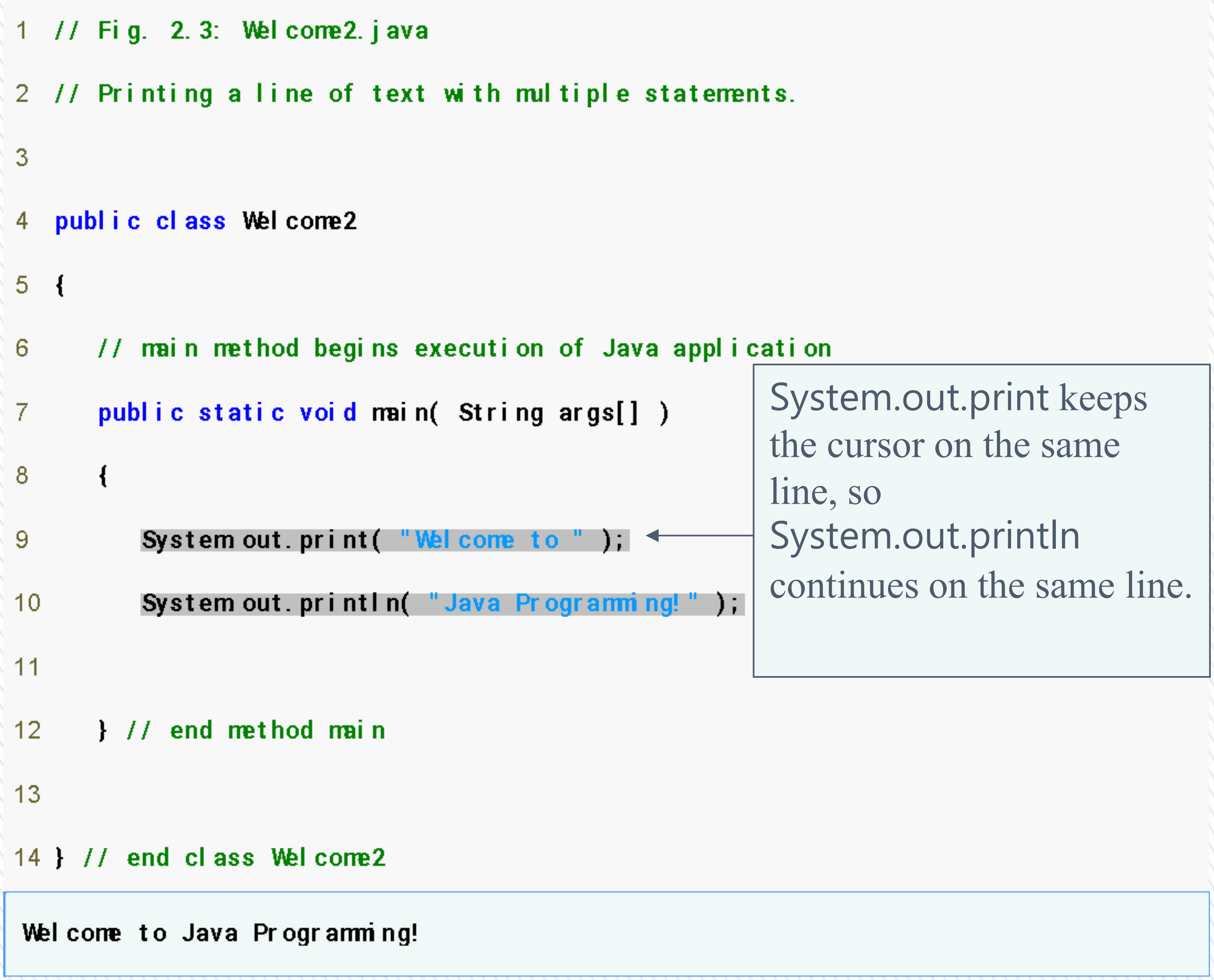
- When learning how to program, sometimes it is helpful to “break” a working program so you can familiarize yourself with the compiler's syntax-error messages.
  - These messages do not always state the exact problem in the code. When you encounter such syntax-error messages in the future, you will have an idea of what caused the error.
- When the compiler reports a syntax error, the error may not be on the line number indicated by the error message. First, check the line for which the error was reported. If that line does not contain syntax errors, check several preceding lines.

Compiling & Executing a Java Program

- Open a command prompt window, go to directory where program is stored
- Type **`javac Welcome1.java`**
- If no syntax errors, **`Welcome1.class`** is created
  - Has bytecodes that represent application
  - Bytecodes passed to JVM
- Type **`java Welcome1`**
  - Launches JVM
  - JVM loads **.class** file for class Welcome1
  - .class extension omitted from command
  - JVM calls method main

Book reading:Section 2.1 - 2.5

Modifying the Java Program



```java
// Fig. 2.3: Welcome2.java
// Printing a line of text with multiple statements.

public class Welcome2
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.print( "Welcome to " );
      System.out.println( "Java Programming!" );

   } // end method main

} // end class Welcome2
```

System.out.print keeps the cursor on the same line, so System.out.println continues on the same line.

```
Welcome to Java Programming!
```

» Welcome2.java

1. Comments

2. Blank line

3. Begin class Welcome2

3.1 Method main

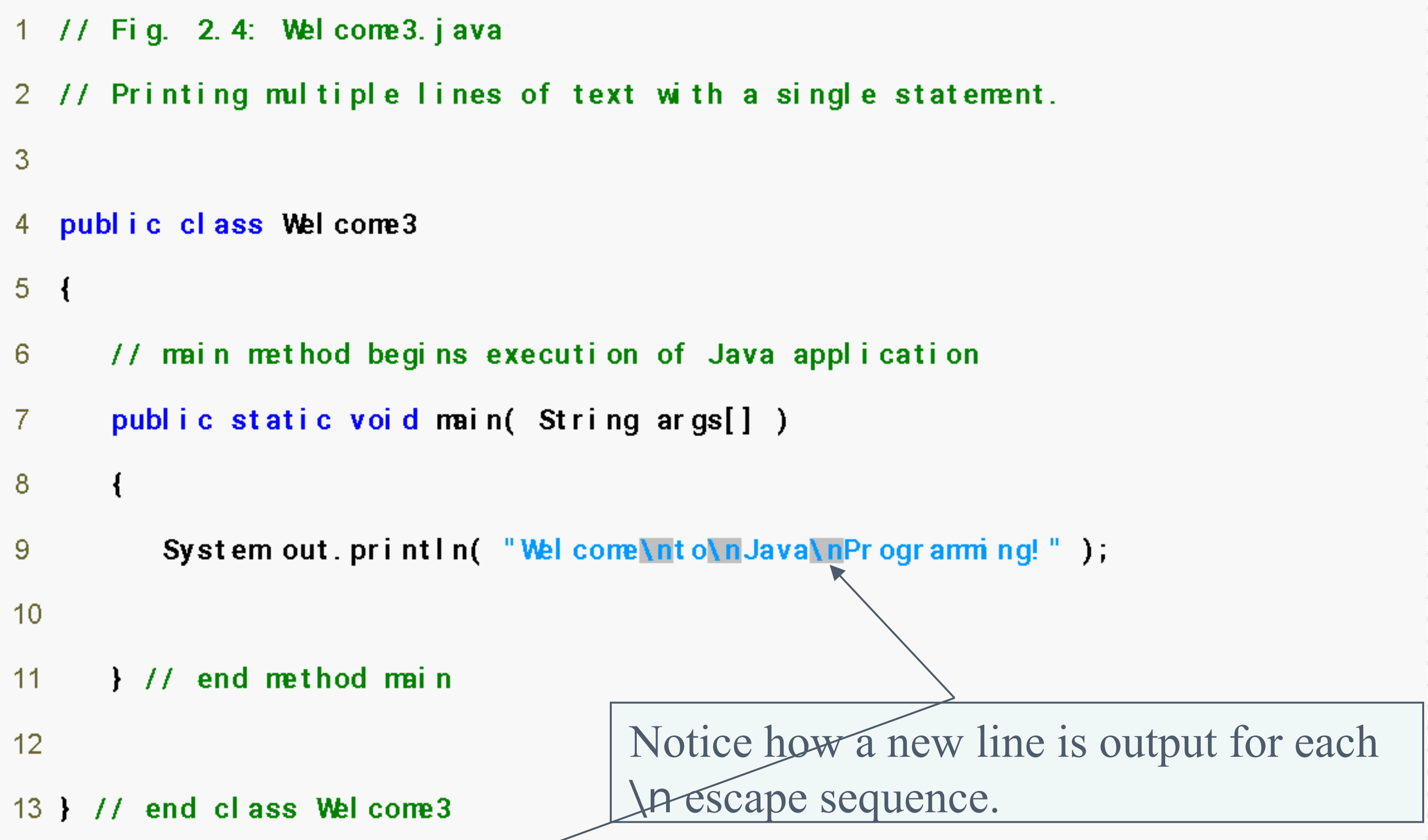
4. Method System.out.print

4.1 Method System.out.println

5. end main, Welcome2

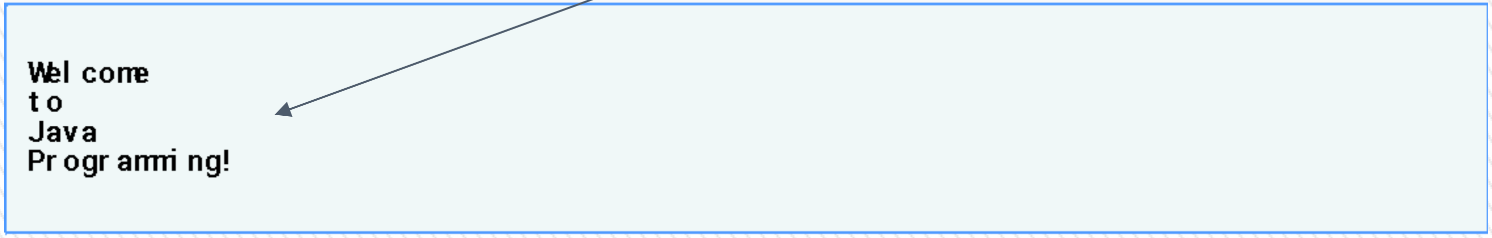
Program Output

A Slightly Different Version



```java
// Fig. 2.4: Welcome3.java
// Printing multiple lines of text with a single statement.

public class Welcome3
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.println( "Welcome\nto\nJava\nProgramming!" );

   } // end method main

} // end class Welcome3
```

Notice how a new line is output for each \n escape sequence.



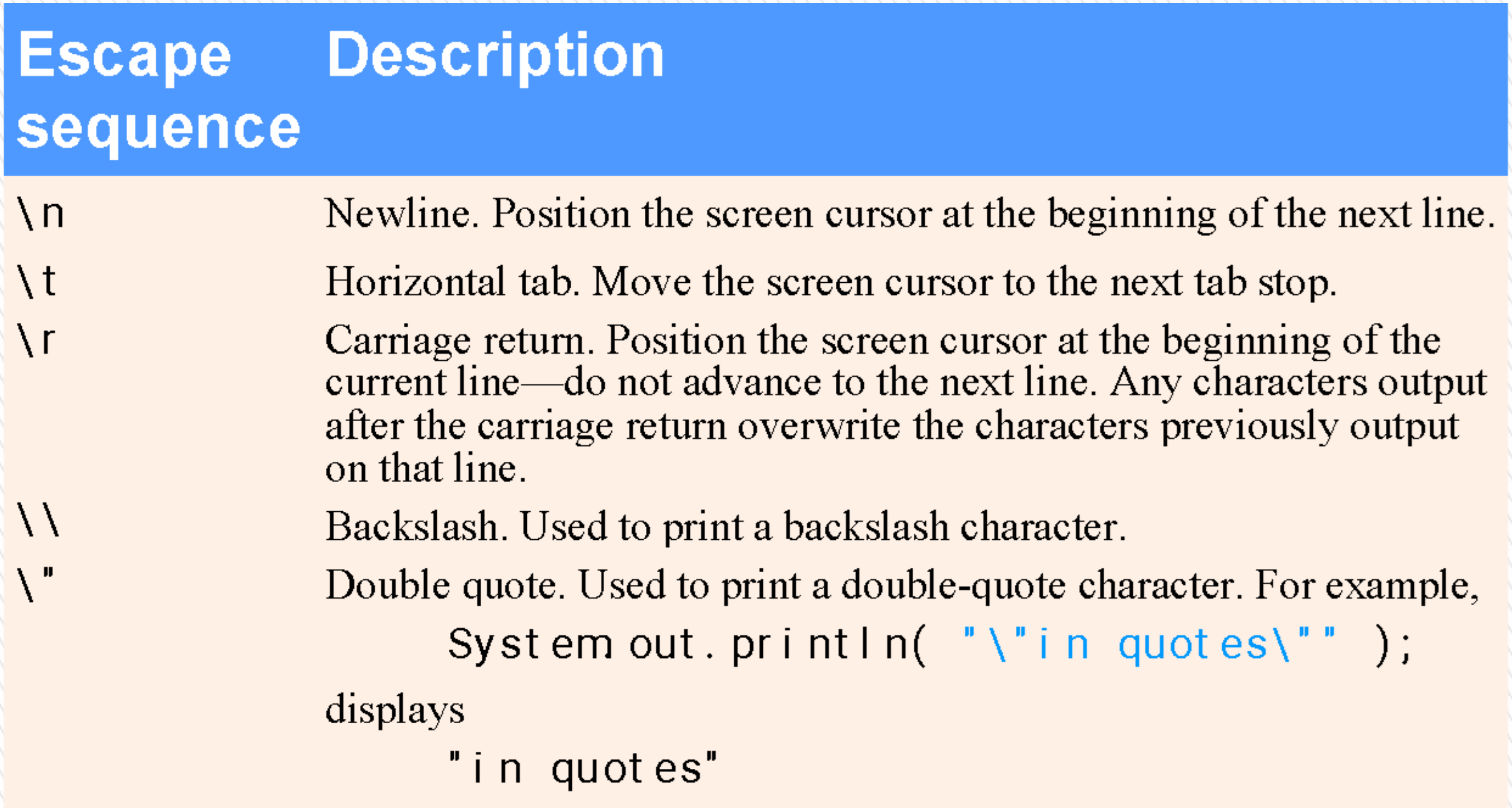
```
Welcome
to
Java
Programming!
```

- » Welcome3.java
- » 1. main

  2. System.out.println (uses \n for new line)
- » Program Output

Book reading:Section 2.1 - 2.5

Some Common Escape Sequences

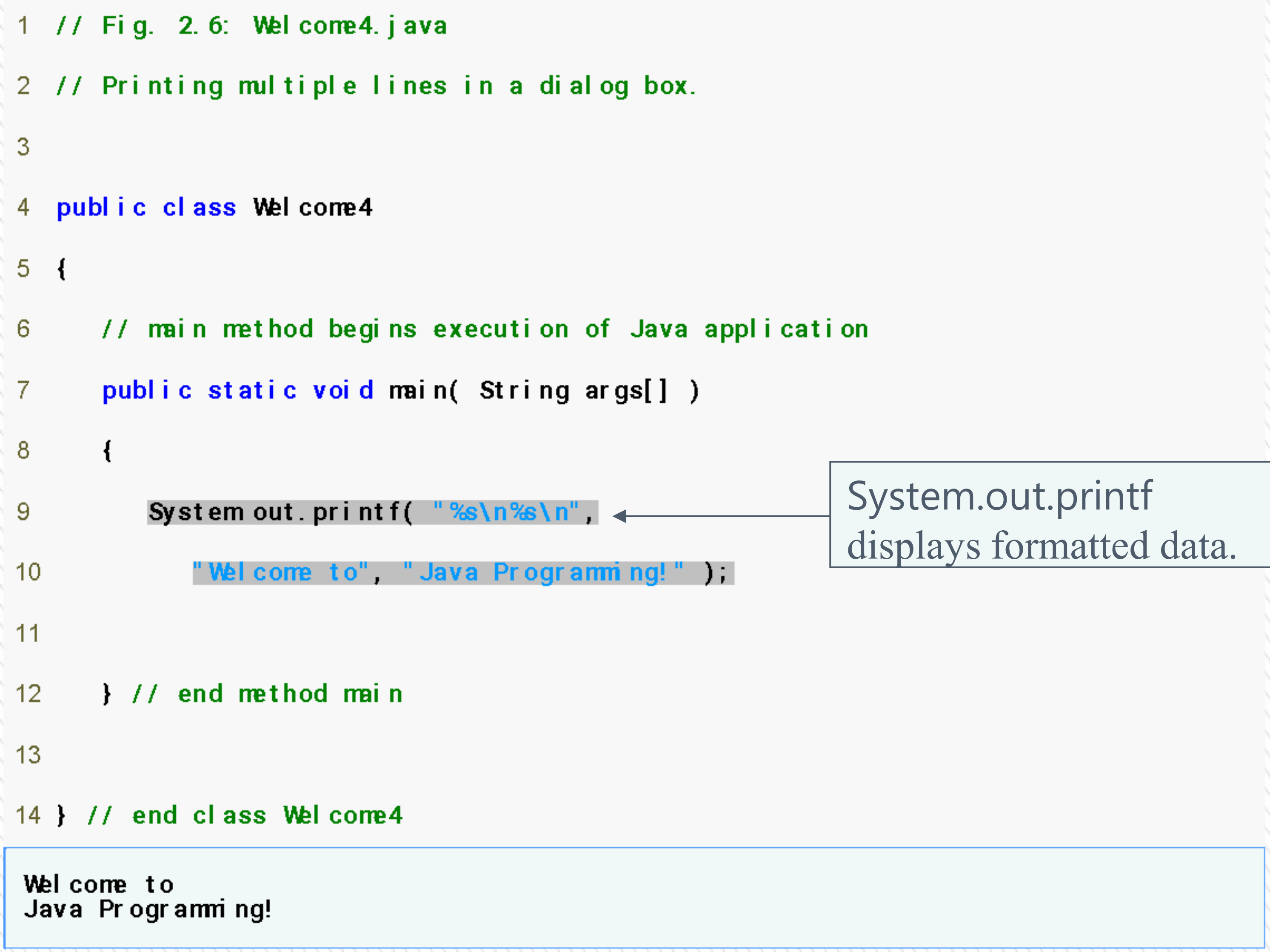


Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.

`\"`	Double quote. Used to print a double-quote character. For example,
	`System out.println( "\"in quotes\"" );`
	displays
	`"in quotes"`

Book reading:Section 2.1 - 2.5

Displaying Text with printf



```
// Fig. 2.6: Welcome4.java
// Printing multiple lines in a dialog box.

public class Welcome4
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.printf( "%s\n%s\n",
         "Welcome to", "Java Programming!" );

   } // end method main

} // end class Welcome4
```

System.out.printf displays formatted data.

```
Welcome to
Java Programming!
```

- » Welcome4.java
- » main
- » printf
- » Program output

Book reading:Section 2.1 - 2.5

Another Java Application: Adding Integers

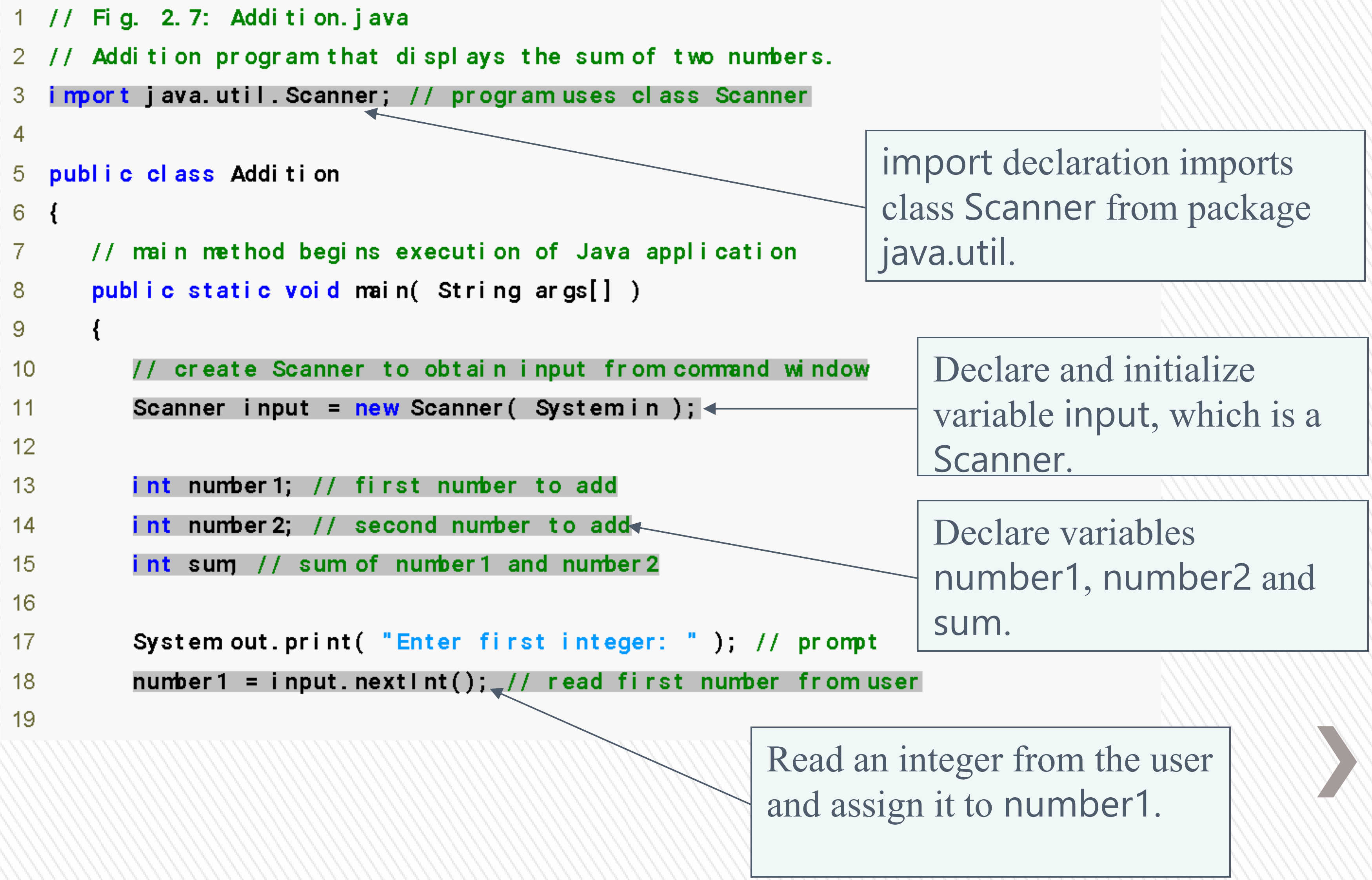
- » Use Scanner to read two integers from user
- » Some other methods to get input from user are:

http://stackoverflow.com/questions/5287538/how-can-i-get-the-user-input-in-java

- » Use printf to display sum of the two values
- » Use packages
  - > By default, package java.lang is imported in every Java program; thus, java.lang is the only package in the Java API that does not require an import declaration.

Book reading:Section 2.1 - 2.5

Another Java Application: Adding Integers



```java
// Fig. 2.7: Addition.java
// Addition program that displays the sum of two numbers.
import java.util.Scanner; // program uses class Scanner

public class Addition
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number1 and number2

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

```

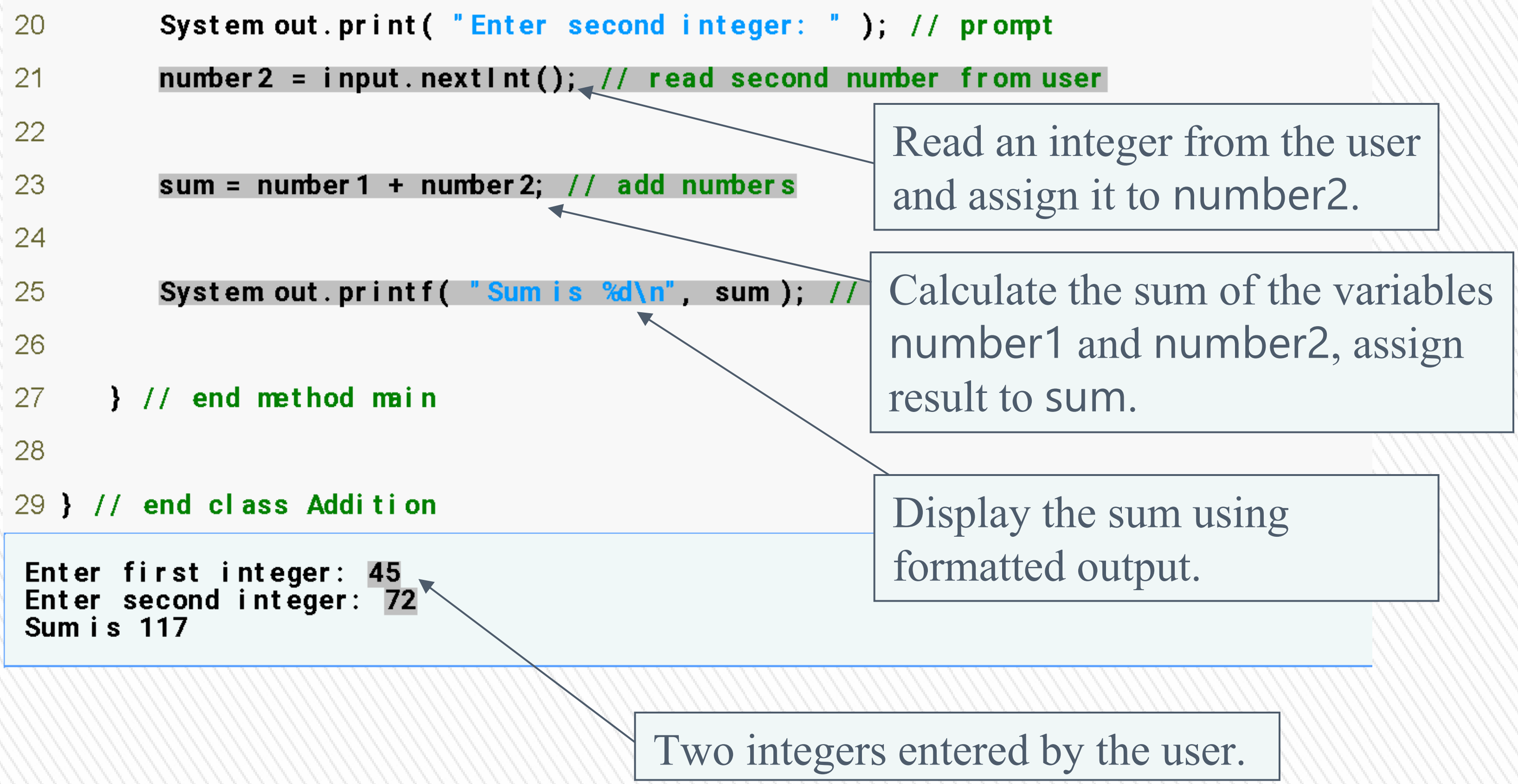
import declaration imports class Scanner from package java.util.

Declare and initialize variable input, which is a Scanner.

Declare variables number1, number2 and sum.

Read an integer from the user and assign it to number1.

Another Java Application: Adding Integers



```
      System out.print( "Enter second integer: " ); // prompt
      number2 = input.nextInt(); // read second number from user

      sum = number1 + number2; // add numbers

      System out.printf( "Sum is %d\n", sum ); //

   } // end method main

} // end class Addition
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Read an integer from the user and assign it to number2.

Calculate the sum of the variables number1 and number2, assign result to sum.

Display the sum using formatted output.

Two integers entered by the user.

Import Declarations

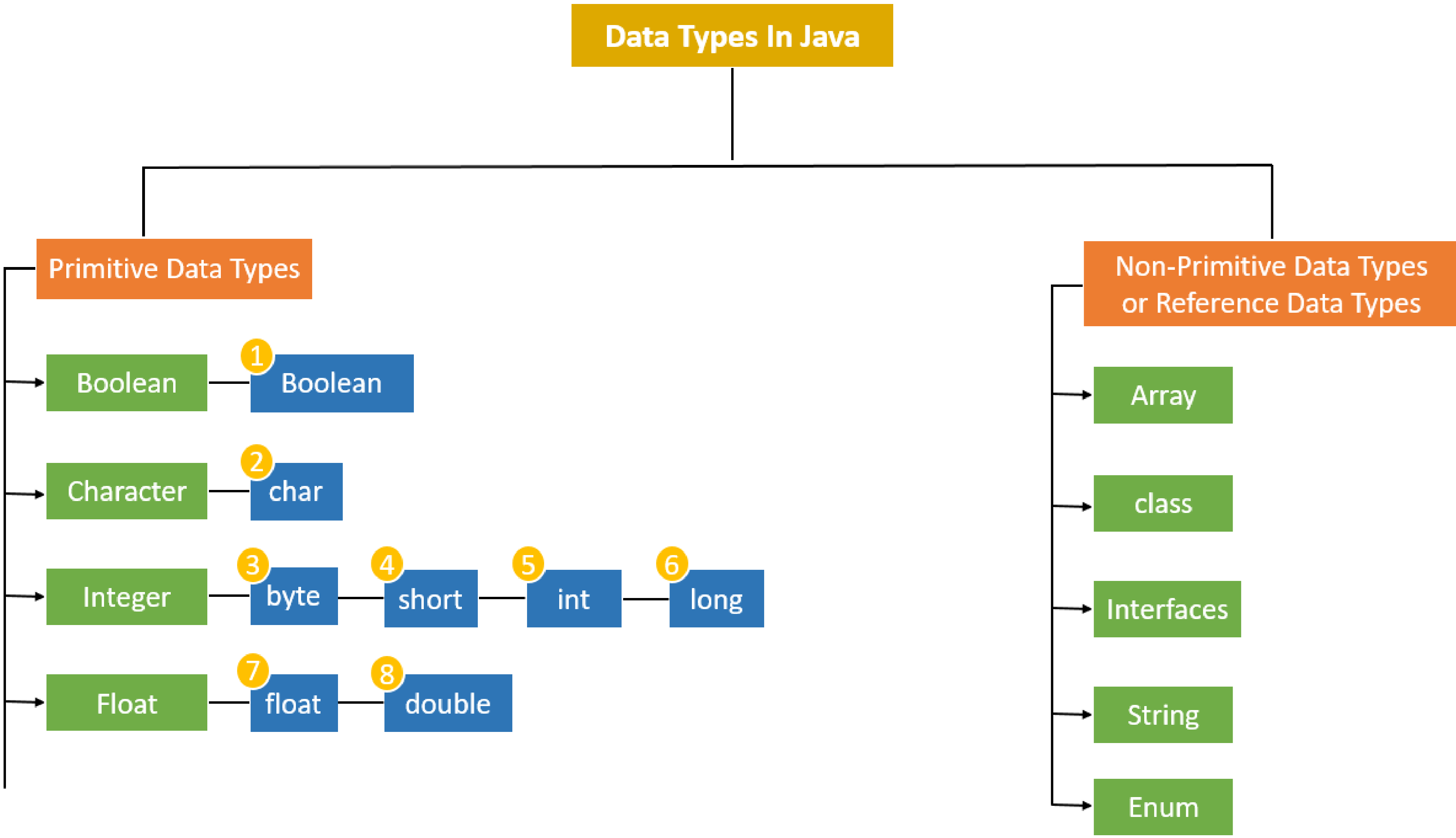
- » import declarations are used by compiler to identify and locate classes used in Java programs
- » All import declarations must appear before the first class declaration in the file. Placing an import declaration inside a class declaration's body or after a class declaration is a syntax error.
- » Forgetting to include an import declaration for a class used in your program typically results in a compilation error containing a message such as "cannot resolve symbol."
  - > When this occurs, check that you provided the proper import declarations and that the names in the import declarations are spelled correctly, including proper use of uppercase and lowercase letters.

Try this: https://www.carbonfootprint.com/calculator.aspx

Good Programming Practice – Tips

- Declare each variable on a separate line. This format allows a descriptive comment to be easily inserted next to each declaration.
- Choosing meaningful variable names helps a program to be *self-documenting* (i.e., one can understand the program simply by reading it rather than by reading manuals or viewing an excessive number of comments).
- By convention, variable-name identifiers begin with a lowercase letter, and every word in the name after the first word begins with a capital letter. This convention is usually referred as camel case.

JAVA DATATYPES



Data Types In Java

Primitive Data Types

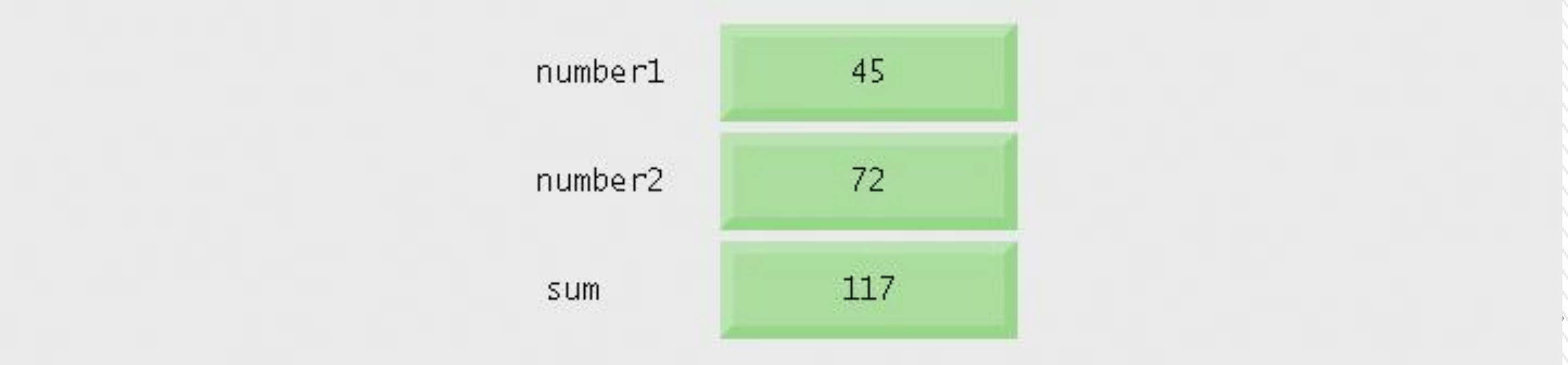
- Boolean — 1 Boolean
- Character — 2 char
- Integer — 3 byte — 4 short — 5 int — 6 long
- Float — 7 float — 8 double

Non-Primitive Data Types or Reference Data Types

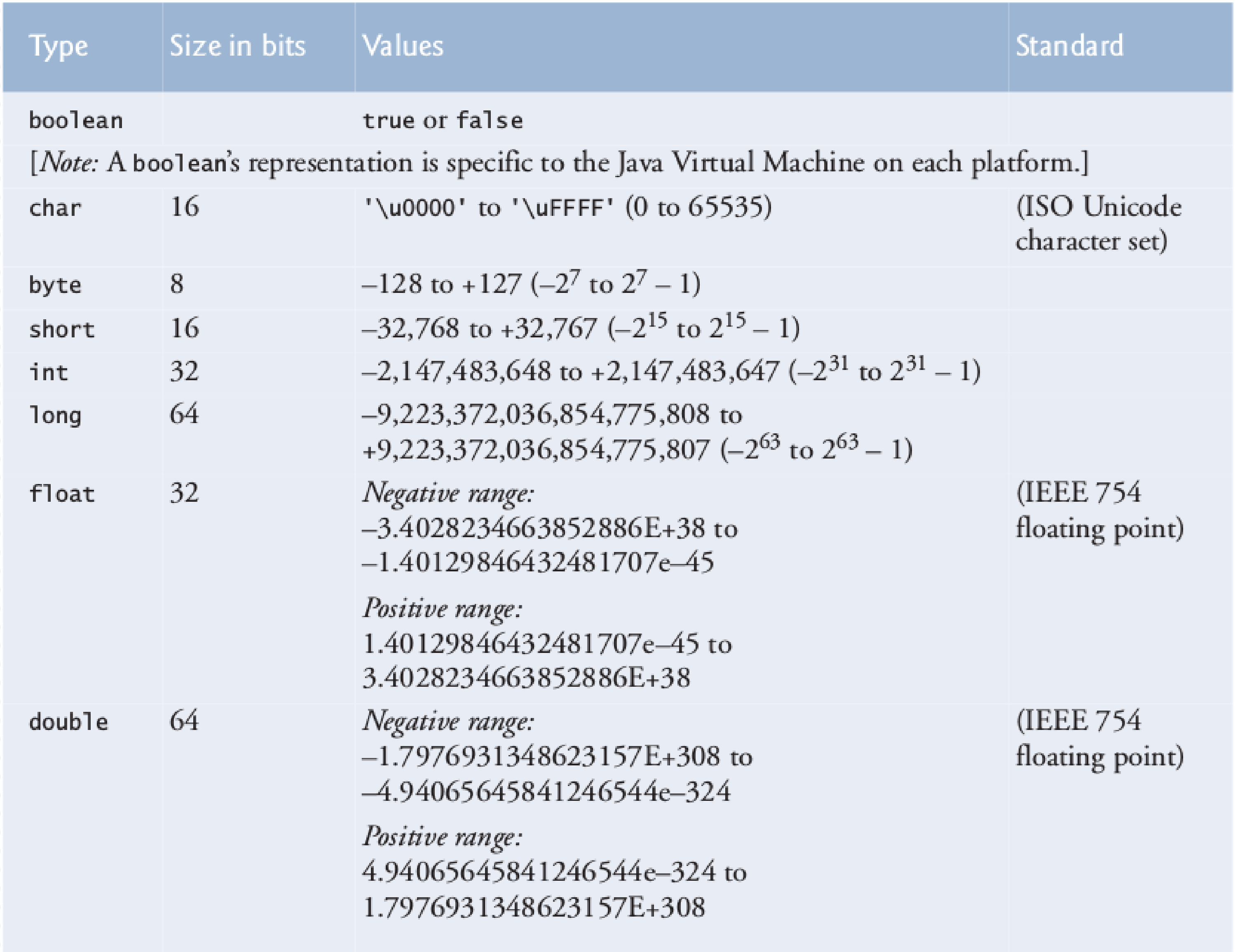
- Array
- class
- Interfaces
- String
- Enum

Memory Concepts

- » Variables
  - > Every variable has a **name**, a **type**, a **size** and a **value**
    - ❖ Name corresponds to location in memory
  - > When new value is placed into a variable, replaces (and destroys) previous value
  - > Reading variables from memory does not change them



PRIMITIVE DATA TYPES IN JAVA



Type	Size in bits	Values	Standard
`boolean`		`true` or `false`	
[*Note:* A `boolean`'s representation is specific to the Java Virtual Machine on each platform.]			
`char`	16	`'\u0000'` to `'\uFFFF'` (0 to 65535)	(ISO Unicode character set)
`byte`	8	–128 to +127 (-2^7 to $2^7 - 1$)	
`short`	16	–32,768 to +32,767 (-2^{15} to $2^{15} - 1$)	
`int`	32	–2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31} - 1$)	
`long`	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (-2^{63} to $2^{63} - 1$)	
`float`	32	*Negative range:* –3.4028234663852886E+38 to –1.40129846432481707e–45 *Positive range:* 1.40129846432481707e–45 to 3.4028234663852886E+38	(IEEE 754 floating point)
`double`	64	*Negative range:* –1.7976931348623157E+308 to –4.94065645841246544e–324 *Positive range:* 4.94065645841246544e–324 to 1.7976931348623157E+308	(IEEE 754 floating point)

ARITHMETIC OPERATIONS

- Arithmetic calculations used in most programs
  - Usage
    - * for multiplication
    - / for division
    - % for remainder
    - +, -
  - Integer division truncates remainder (no rounding off)

    7 / 5 evaluates to 1
  - Remainder operator % returns the remainder

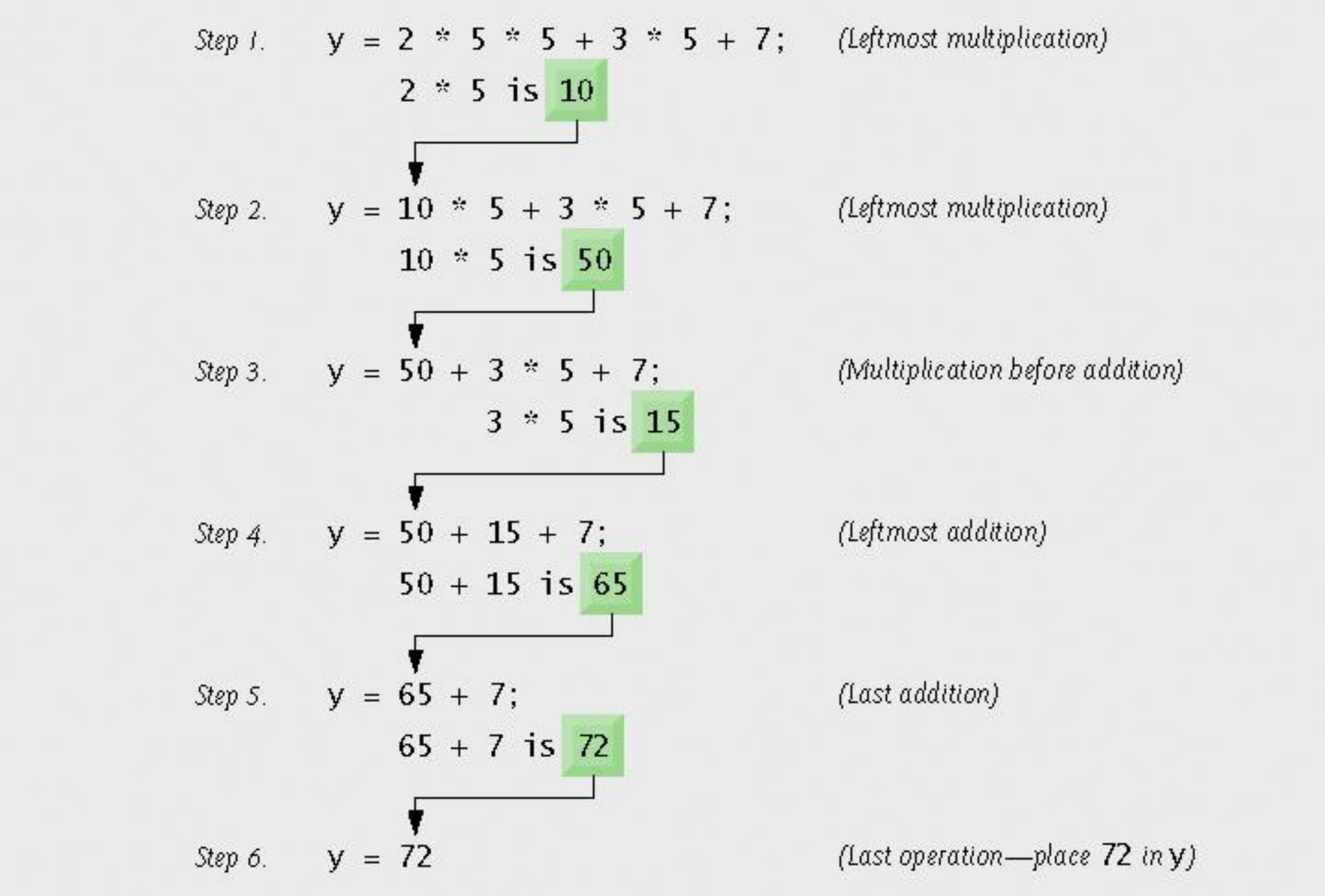
    7 % 5 evaluates to 2

Operator Precedence

- » Some arithmetic operators act before others (i.e., multiplication before addition)
  - > Use parenthesis when needed
- » **Example**: Find the average of three variables a, b and c
  - > Do not use: a + b + c / 3
  - > Use: (a + b + c) / 3
- » Question: How the following expression will be evaluated?

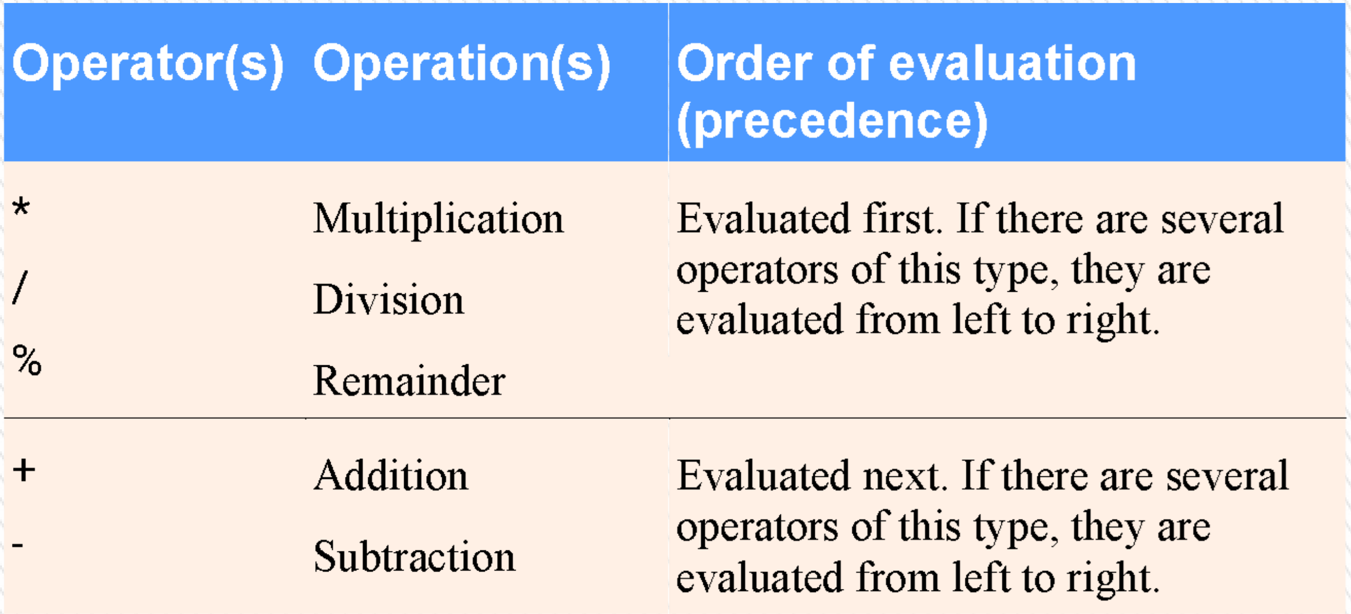
Y = 2 * 5 * 5 + 3 * 5 + 7;

Operator Precedence



```
Step 1.   y = 2 * 5 * 5 + 3 * 5 + 7;      (Leftmost multiplication)
              2 * 5 is 10

Step 2.   y = 10 * 5 + 3 * 5 + 7;         (Leftmost multiplication)
              10 * 5 is 50

Step 3.   y = 50 + 3 * 5 + 7;             (Multiplication before addition)
                   3 * 5 is 15

Step 4.   y = 50 + 15 + 7;                (Leftmost addition)
              50 + 15 is 65

Step 5.   y = 65 + 7;                     (Last addition)
              65 + 7 is 72

Step 6.   y = 72                          (Last operation—place 72 in y)
```

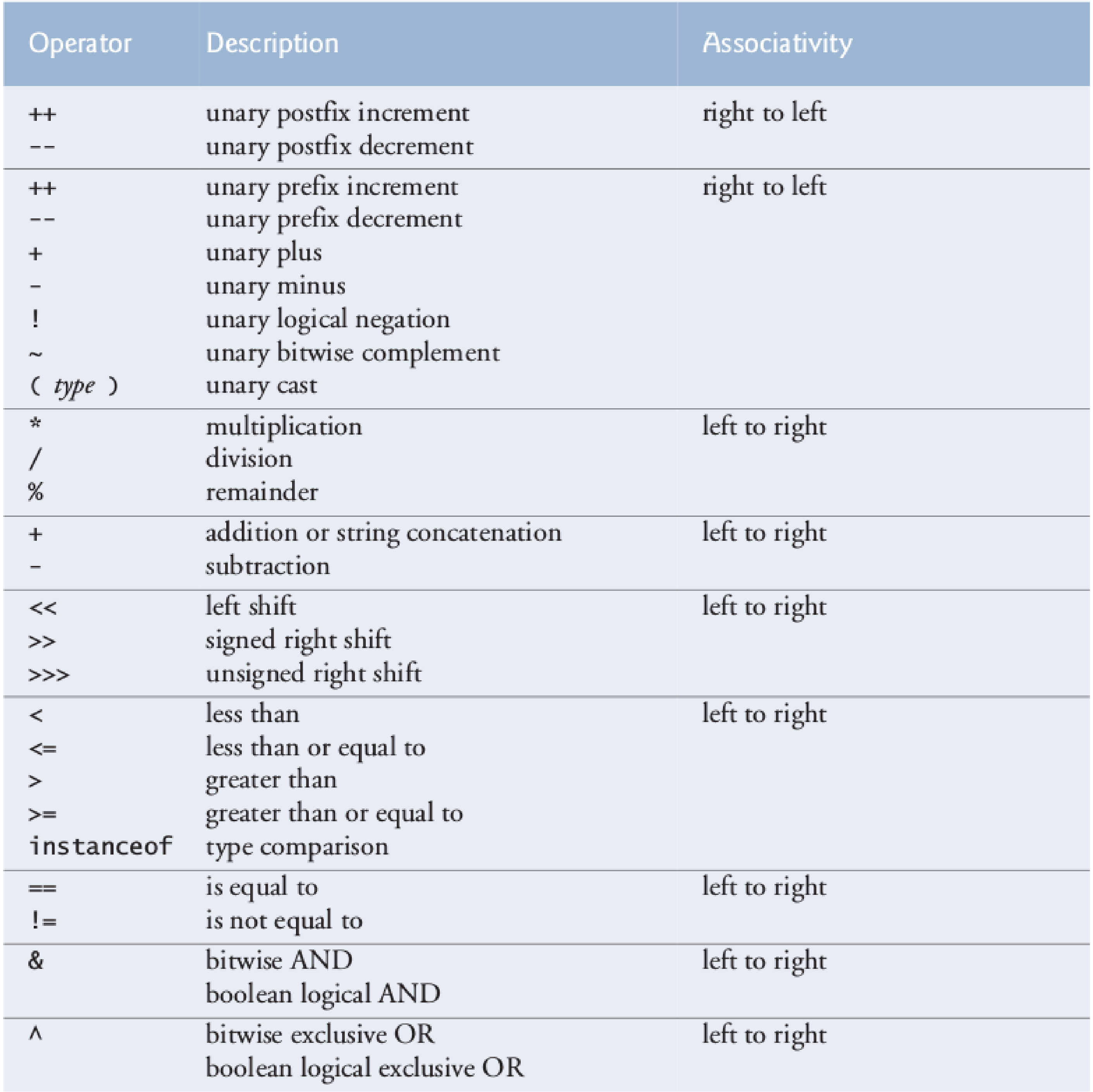
Operator Precedence



Operator(s)	Operation(s)	Order of evaluation (precedence)
* / %	Multiplication Division Remainder	Evaluated first. If there are several operators of this type, they are evaluated from left to right.
+ -	Addition Subtraction	Evaluated next. If there are several operators of this type, they are evaluated from left to right.

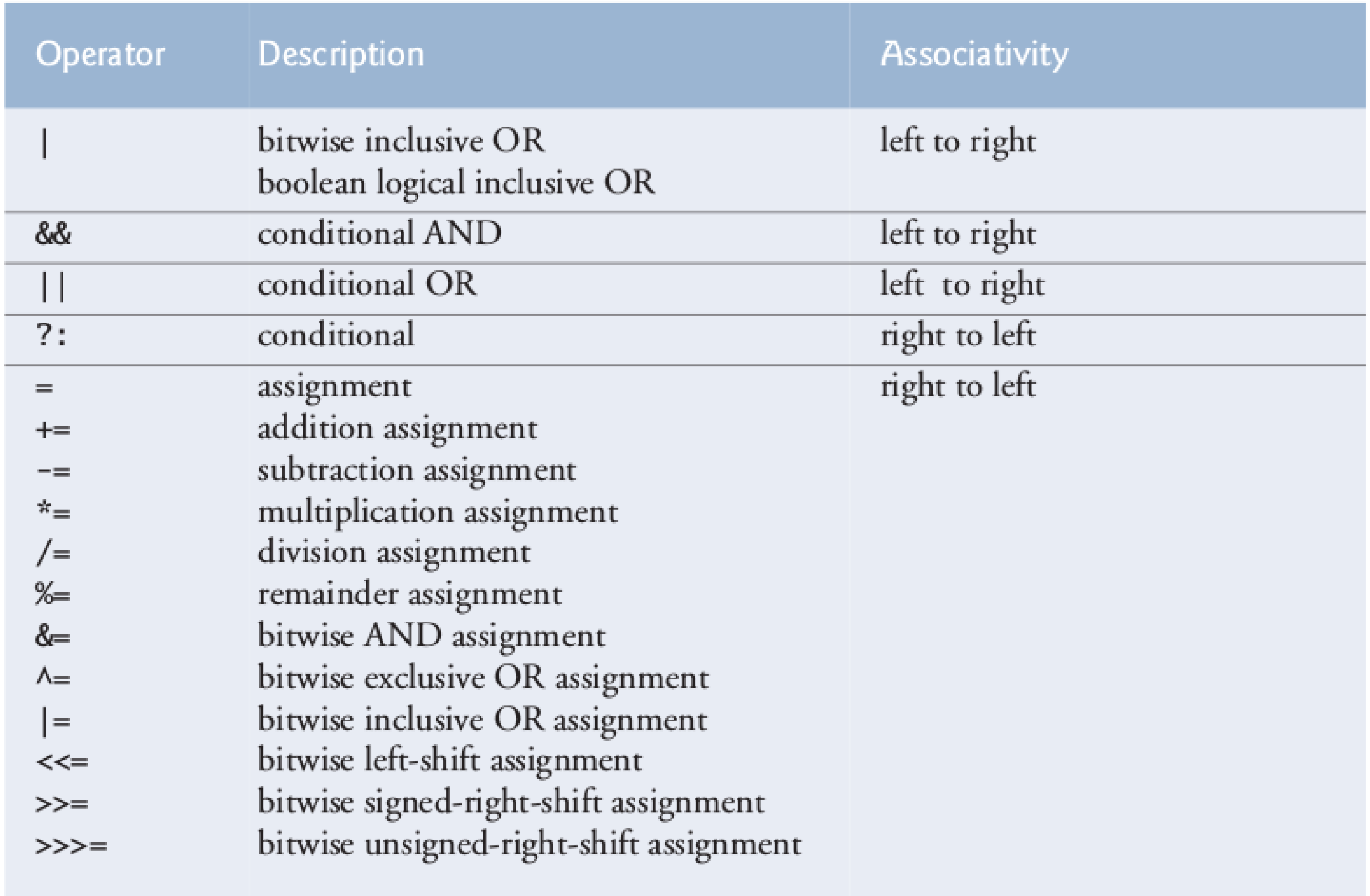
- **Tip:** Using parentheses for complex arithmetic expressions, even when the parentheses are not necessary, can make the arithmetic expressions easier to read.

Operator Precedence



Operator	Description	Associativity
++ --	unary postfix increment unary postfix decrement	right to left
++ -- + - ! ~ (*type*)	unary prefix increment unary prefix decrement unary plus unary minus unary logical negation unary bitwise complement unary cast	right to left
* / %	multiplication division remainder	left to right
+ -	addition or string concatenation subtraction	left to right
<< >> >>>	left shift signed right shift unsigned right shift	left to right
< <= > >= instanceof	less than less than or equal to greater than greater than or equal to type comparison	left to right
== !=	is equal to is not equal to	left to right
&	bitwise AND boolean logical AND	left to right
^	bitwise exclusive OR boolean logical exclusive OR	left to right

Operator Precedence



Operator	Description	Associativity
|	bitwise inclusive OR boolean logical inclusive OR	left to right
&&	conditional AND	left to right
||	conditional OR	left to right
?:	conditional	right to left
= += -= *= /= %= &= ^= |= <<= >>= >>>=	assignment addition assignment subtraction assignment multiplication assignment division assignment remainder assignment bitwise AND assignment bitwise exclusive OR assignment bitwise inclusive OR assignment bitwise left-shift assignment bitwise signed-right-shift assignment bitwise unsigned-right-shift assignment	right to left

Questions/Answers & Discussion