Introduction to Java

Objectives

- To write simple Java applications.
- To use input and output statements.
- Java’s primitive types.
- Basic memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write decision-making statements.
- To use relational and equality operators.

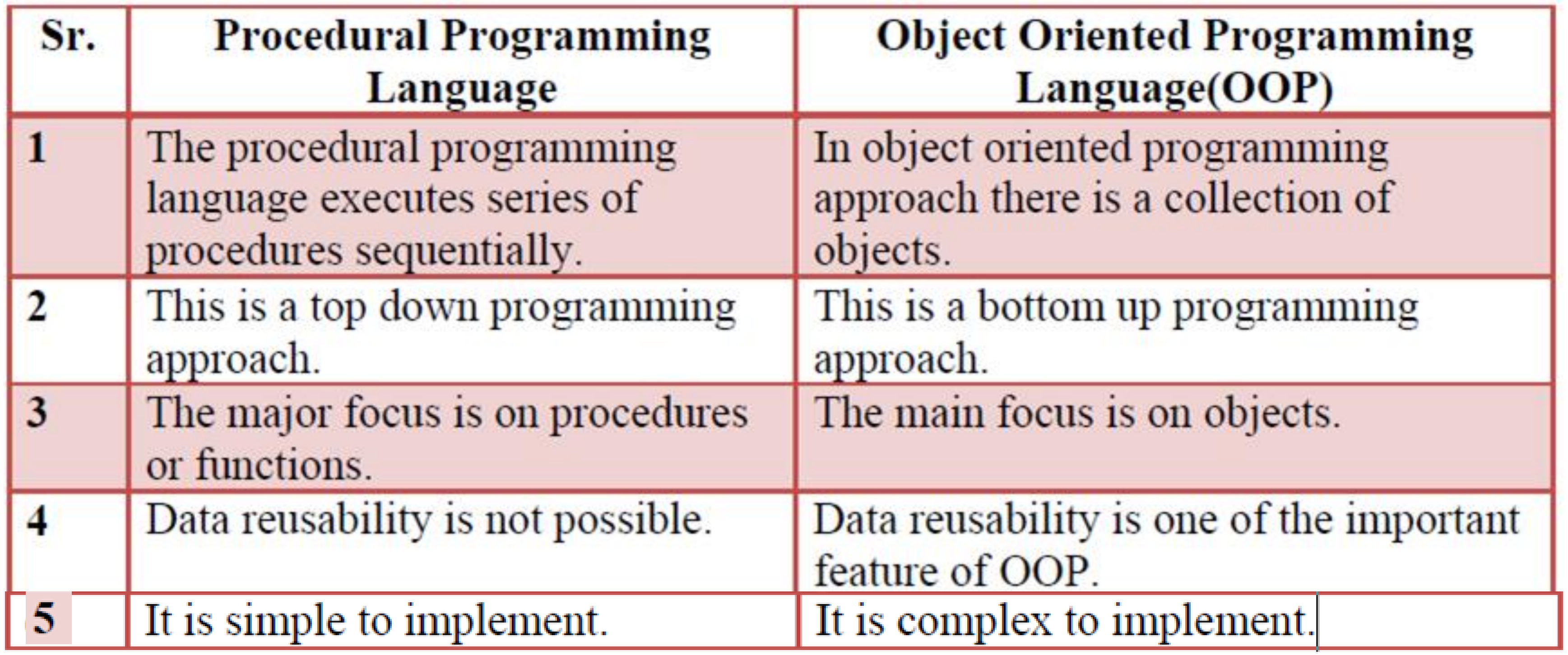
History of Java

- Java
  - Originally for intelligent consumer-electronic devices
  - Then used for creating web pages with dynamic content
  - Now also used to:
    - Develop large-scale enterprise applications
    - Enhance web server functionality
    - Provide applications for consumer devices (cell phones, etc.)

Java Class Libraries

- Java programs consist of classes
  - Include methods that perform tasks
    - Return information after task completion
- Java provides class libraries
  - Known as Java APIs (Application Programming Interfaces)
- To use Java effectively, you must know
  - Java programming language
  - Extensive class libraries

Procedural Programming vs Object Oriented Programming



Sr.	Procedural Programming Language	Object Oriented Programming Language(OOP)
1	The procedural programming language executes series of procedures sequentially.	In object oriented programming approach there is a collection of objects.
2	This is a top down programming approach.	This is a bottom up programming approach.
3	The major focus is on procedures or functions.	The main focus is on objects.
4	Data reusability is not possible.	Data reusability is one of the important feature of OOP.
5	It is simple to implement.	It is complex to implement.

Java is an Object Oriented Programming Language

Software Engineering Observation 1

- 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.

Software Engineering Observation 2

- When programming in Java, you will typically use the following building blocks: Classes and methods from class libraries, classes and methods you create yourself and classes and methods that others create and make available to you.

Software Engineering Observation 3

- Extensive class libraries of reusable software components are available over the Internet and the web, many at no charge.

Performance Tip

- Using Java API classes and methods instead of writing your own versions can improve program performance, because they are carefully written to perform efficiently. This technique also shortens program development time.

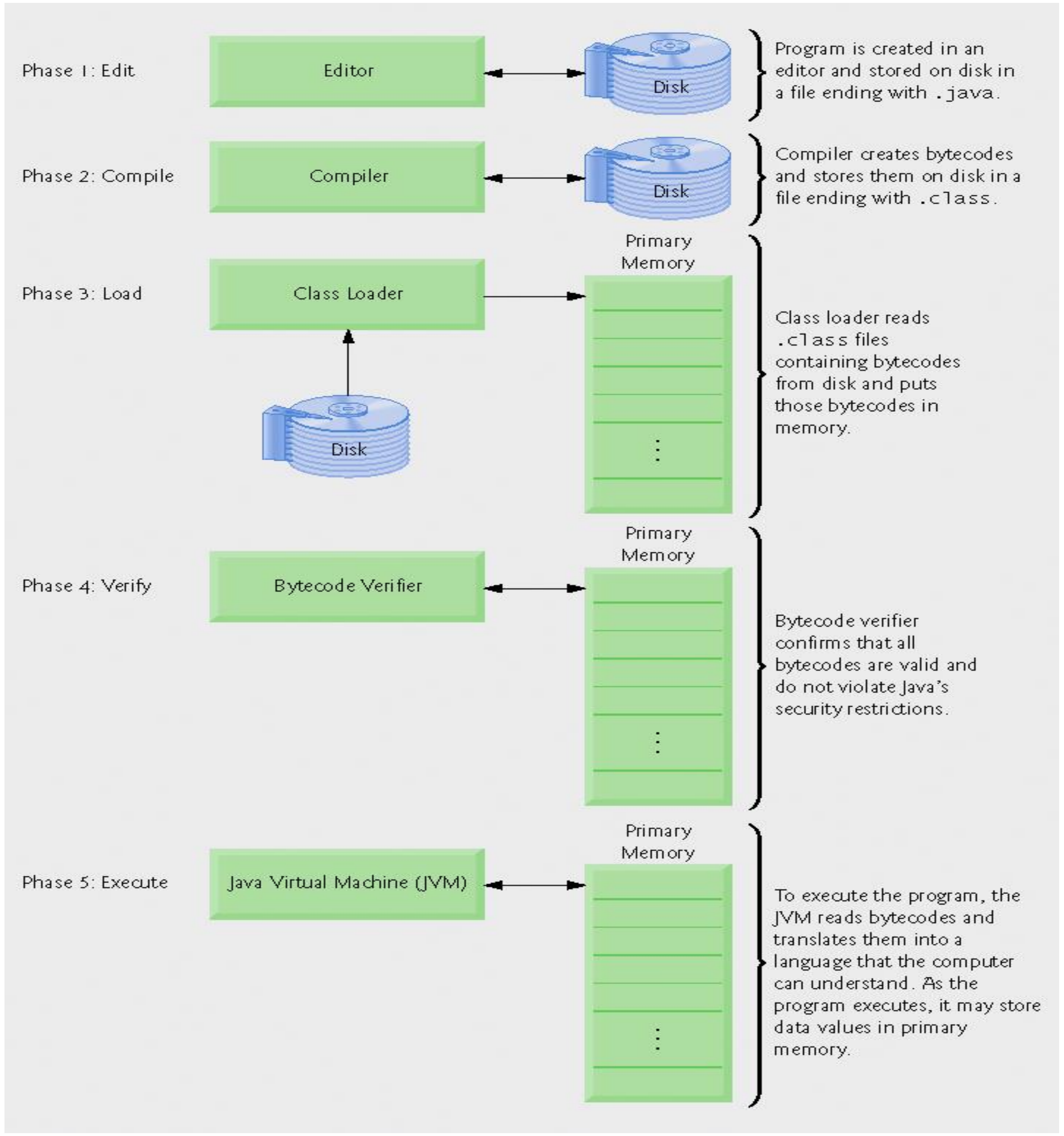
Portability Tip

- Using classes and methods from the Java API instead of writing your own improves program portability, because they are included in every Java implementation.

Typical Java Development Environment

- Java programs go through five phases
  - Edit
    - Programmer writes program using an editor; stores program on disk with the .java file name extension
  - Compile
    - Use javac (the Java compiler) to create bytecodes from source code program; bytecodes stored in .class files
  - Load
    - Class loader reads bytecodes from .class files into memory
  - Verify
    - Bytecode verifier examines bytecodes to ensure that they are valid and do not violate security restrictions
  - Execute
    - Java Virtual Machine (JVM) uses a combination of interpretation and just-in-time compilation to translate bytecodes into machine language

Fig. 1.1 | Typical Java development environment.



Phase 1: Edit
Editor
Disk
Program is created in an editor and stored on disk in a file ending with .java.
Phase 2: Compile
Compiler
Disk
Compiler creates bytecodes and stores them on disk in a file ending with .class.
Phase 3: Load
Class Loader
Disk
Primary Memory
Class loader reads .class files containing bytecodes from disk and puts those bytecodes in memory.
Phase 4: Verify
Bytecode Verifier
Primary Memory
Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.
Phase 5: Execute
Java Virtual Machine (JVM)
Primary Memory
To execute the program, the JVM reads bytecodes and translates them into a language that the computer can understand. As the program executes, it may store data values in primary memory.

Common Programming Error

- Errors like division by zero occur as a program runs, so they are called *runtime errors* or *execution-time errors*. *Fatal runtime errors* cause programs to terminate immediately without having successfully performed their jobs. *Nonfatal runtime errors* allow programs to run to completion, often producing incorrect results.

Good Programming Practice

- Write your Java programs in a simple and straightforward manner. This is sometimes referred to as KIS (“keep it simple”). Do not “stretch” the language by trying bizarre usages.
- Read the documentation for the version of Java you are using. Refer to it frequently to be sure you are aware of the rich collection of Java features and are using them correctly.
- Your computer and compiler are good teachers. If, after carefully reading your Java documentation manual, you are not sure how a feature of Java works, experiment and see what happens. Study each error when you compile your programs (called *compilation errors*), and correct the programs to eliminate these messages.

Java application programming

- ➢Display messages
- ➢Obtain information from the user
- ➢Arithmetic calculations
- ➢Decision-making fundamentals

First Program in Java: Printing a Line of Text

```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!
```

First Program in Java: Printing a Line of Text (Cont.)

- Comments start with: //
  - Indicate that the remainder of the line is a comment
  - Comments are ignored during program execution
  - Comments are used to document programs, describe code, and improve readability
- Traditional comments: /* ... */

```
/* This is a traditional
   comment. It can be
   split over many lines */
```

- Note: line numbers not part of program, added for reference

Common Programming Error 2.1

- Forgetting one of the delimiters of a traditional or Javadoc comment is a syntax error. The *syntax* of a programming language specifies the rules for creating a proper program in that language. A *syntax error* occurs when the compiler encounters code that violates Java’s language rules (i.e., its syntax). In this case, the compiler does not produce a `.class` file. Instead, the compiler issues an error message to help the programmer identify and fix the incorrect code. Syntax errors are also called *compiler errors*, *compile-time errors* or *compilation errors*, because the compiler detects them during the compilation phase. You will be unable to execute your program until you correct all the syntax errors in it.

First Program in Java: Printing a Line of Text (Cont.)

- Blank line
  - Makes program more readable
  - Blank lines, spaces, and tabs are white-space characters
    - Ignored by compiler

```
public class Welcome1
```

- Begins class declaration for class Welcome1
  - 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
    - SampleClassName

First Program in Java: Printing a Line of Text (Cont.)

```
public class Welcome1
```

- A Java class name is an **identifier**
  - Identifier-Series of characters consisting of letters, digits, underscores (_) and dollar signs ($) that does not begin with a digit and does not contain spaces
  - Examples: Welcome1, $value, _value, button7
    - 7button is invalid
  - Java is case sensitive (capitalization matters)
    - a1 and A1 are different

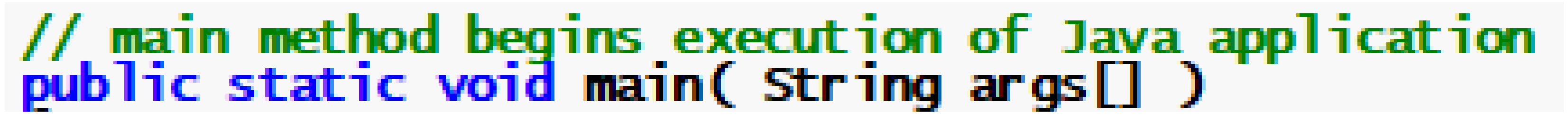
Good Programming Practice

- 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 Error

- Java is case sensitive. Not using the proper uppercase and lowercase letters for an identifier normally causes a compilation error.

First Program in Java: Printing a Line of Text (Cont.)



```
// main method begins execution of Java application
public static void main( String args[] )
```

- The above statement is the starting point of every Java application.
- The parentheses after the identifier indicate that it is a program building block called a method.
- Java class declarations normally contain one or more methods.
- For a Java application, exactly one of the methods must be called main.
- Keyword void indicates this method will not return any information when it completes its task.
- The String args[] in the parentheses is a required part of the method main's declaration.

Note: We will learn more about methods in the upcoming chapters, for now, simply mimic main's first line in your java application .

First Program in Java: Printing a Line of Text (Cont.)

```
public class Welcome1
```

- Saving files
  - File name must be class name with `.java` extension
  - `Welcome1.java`

```
{
```

- Left brace {
  - Begins body of method declaration
  - Right brace } ends declarations

Common Programming 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 an error not to end a file name with the `.java` extension for a file containing a class declaration. If that extension is missing, the Java compiler will not be able to compile the class declaration.
- Whenever you type an opening left brace, `{`, in your program, immediately type the closing right brace, `}`, then reposition the cursor between the braces and indent to begin typing the body. This practice helps prevent errors due to missing braces.

First Program in Java: Printing a Line of Text (Cont.)

```
System.out.println( "Welcome to Java Programming!" );
```

- Instructs computer to perform an action
    - Prints string of characters
        - String – series of characters inside double quotes
    - White-spaces in strings are not ignored by compiler
- `System.out`
    - Standard output object
    - Print to command window (i.e., MS-DOS prompt)
- Method `System.out.println`
    - Displays line of text
- This line is known as a statement
    - Statements must end with semicolon `;`

First Program in Java: Printing a Line of Text (Cont.)

```
    } // end method main
```

- Ends method declaration

```
} // end class Welcome1
```

- Ends class declaration
- Can add comments to keep track of ending braces

Error-Prevention Tip

- 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.

Modifying Our First Java Program

- Modify the previous example to print same contents using different code: **System.out.print**

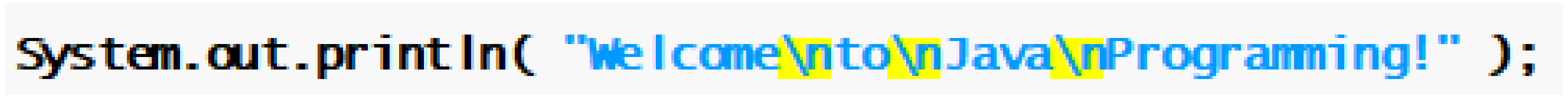
```java
// Fig. 2.3: Welcome2.java
// Printing a line of text with multiple statements.

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
```

```
Welcome to Java Programming!
```

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

Modifying Our First Java Program (Cont.)



```
System.out.println( "Welcome\nto\nJava\nProgramming!" );
```

- Escape characters
  - Backslash (\ )
  - Indicates special characters to be output
- Newline characters (`\n`)
  - Interpreted as “special characters” by methods `System.out.print` and `System.out.println`
  - Indicates cursor should be at the beginning of the next line

Modifying Our First Java Program (Cont.)

```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
```

```
Welcome
to
Java
Programming!
```

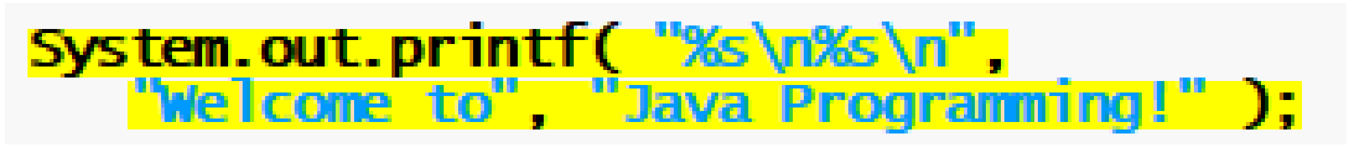
A new line begins after each `\n` escape sequence is output.

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"`

| Some common escape sequences.

Displaying Text with printf

- `System.out.printf`
  - Displays formatted data



```
System.out.printf( "%s\n%s\n",
    "Welcome to", "Java Programming!" );
```

  - Format string
    - Fixed text
    - Format specifier – placeholder for a value
  - Format specifier `%s` – placeholder for a string

Displaying Text with printf

```java
// Fig. 2.6: Welcome4.java
// Displaying multiple lines with method System.out.printf.

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
```

```
Welcome to
Java Programming!
```

Another Java Application: Adding Integers

- Upcoming program
  - Use `Scanner` to read two integers from user
  - Use `printf` to display sum of the two values

```java
// Fig. 2.7: Addition.java
// Addition program that inputs two numbers then displays their sum.
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 a Scanner to obtain input from the command window
      Scanner input = new Scanner(System.in);

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

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

      int sum = number1 + number2; // add numbers, then store total in sum

      System.out.printf("Sum is %d%n", sum); // display sum
   } // end method main
} // end class Addition
```

`import` declaration imports class `Scanner` from package `java.util`.

Declare and initialize variable `input`, which is a `Scanner`.

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

Fig. 2.7 | Addition program that inputs two numbers then, displays their sum. (Part 1 of 2.)

Output:

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

Fig. 2.7 | Addition program that inputs two numbers then, displays their sum. (Part 2 of 2.)

Another Java Application: Adding Integers (Cont.)

```
3  import java.util.Scanner;  // program uses class Scanner
```

- `import` declarations
  - Used by compiler to identify and locate classes used in Java programs
  - Tells compiler to load class `Scanner` from `java.util` package

```
5   public class Addition
6    {
```

- Begins `public` class `Addition`
  - Recall that file name must be `Addition.java`
- Lines 8-9: begin `main`

Common Programming Error

- 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.

Another Java Application: Adding Integers (Cont.)

```
// create Scanner to obtain input from command window
Scanner input = new Scanner( System.in );
```

- Variable Declaration Statement
- Variables
  - Location in memory that stores a value
    - Declare with name and type before use
  - `Input` is of type `Scanner`
    - Enables a program to read data for use
  - Variable name: any valid identifier
- Declarations end with semicolons `;`
- Initialize variable in its declaration
  - Equal sign
  - Standard input object
    - `System.in`

Another Java Application: Adding Integers (Cont.)

```java
int number1; // first number to add
 int number2; // second number to add
int sum; // sum of number 1 and number 2
```

- Declare variable `number1`, `number2` and `sum` of type `int`
  - `int` holds integer values (whole numbers): i.e., 0, -4, 97
  - Types `float` and `double` can hold decimal numbers
  - Type `char` can hold a single character: i.e., x, $, \n, 7
  - `int`, `float`, `double` and `char` are primitive types
- Can add comments to describe purpose of variables
- Can declare multiple variables of the same type in one declaration using comma-separated list, e.g. int number1, number2, sum;

Another Java Application: Adding Integers (Cont.)

```
      System.out.print( "Enter first integer: " ); // prompt
```

- Message called a prompt - directs user to perform an action

```
      number1 = input.nextInt(); // read first number from user
```

- Result of call to `nextInt` given to `number1` using assignment operator =
  - Assignment statement
  - = binary operator - takes two operands
    - Expression on right evaluated and assigned to variable on left
  - Read as: `number1` gets the value of `input.nextInt()`

Another Java Application: Adding Integers (Cont.)

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

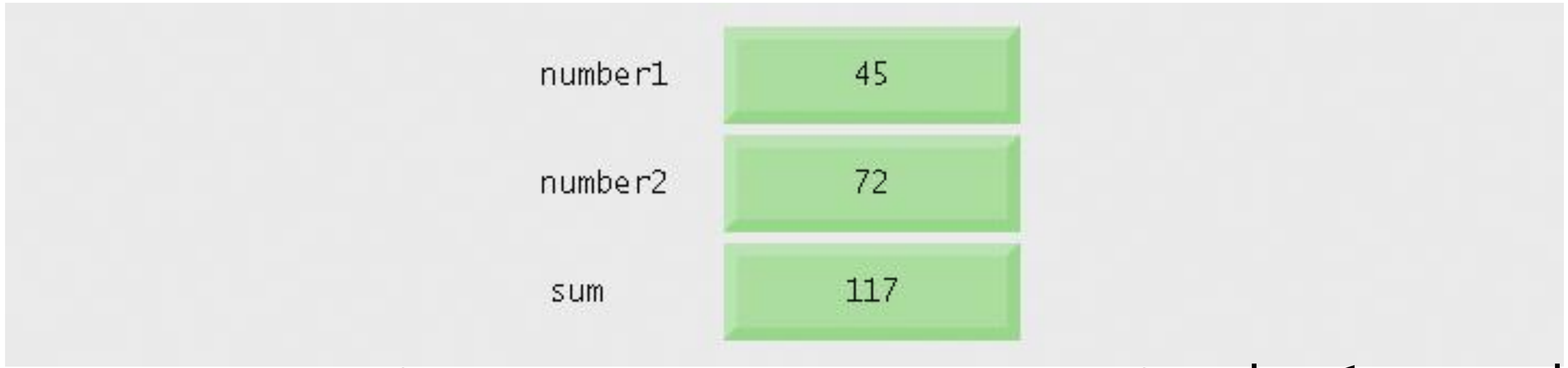
- Use `System.out.printf` to display results
- Format specifier %d
  - Placeholder for an `int` value

```
        System.out.printf( "Sum is %d\n: " , ( number1 + number2 ) );
```

- Calculations can also be performed inside `printf`
- Parentheses around the expression `number1` + `number2` are not required

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



Memory locations after calculating and storing the sum of `number1` and `number2`.

Arithmetic

- Arithmetic calculations used in most programs
  - Usage
    - `*` for multiplication
    - `/` for division
    - `%` for remainder
    - `+`, `-`
  - Integer division truncates remainder

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

    `7 % 5` evaluates to `2`

Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`

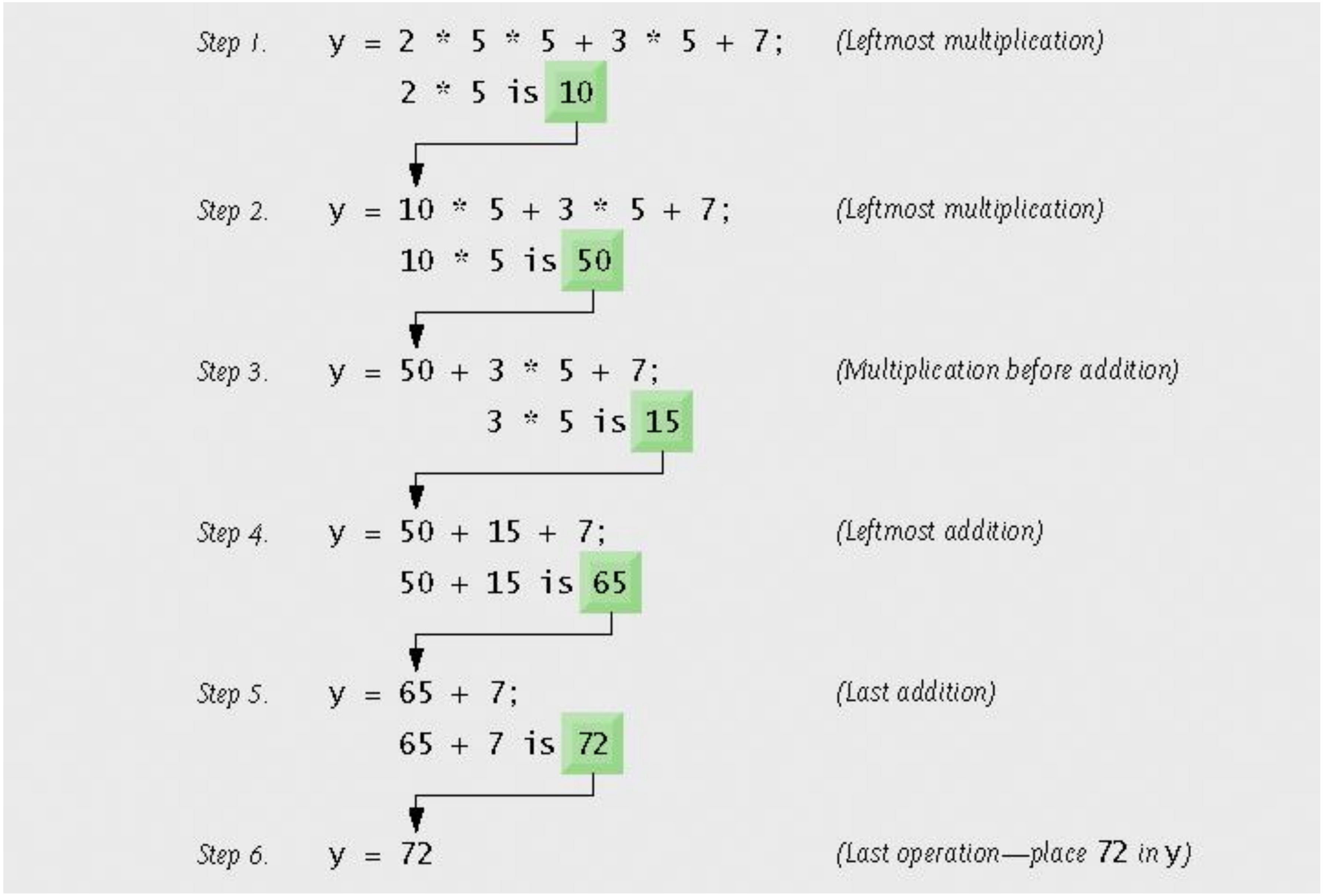
Arithmetic operators.

Arithmetic (Cont.)

- 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`

Precedence of arithmetic operators.

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.



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`*)*

Decision Making: Equality and Relational Operators

- Condition
  - Expression can be either `true` or `false`
- `if` statement
  - Simple version in this section, more detail later
  - If a condition is `true`, then the body of the `if` statement executed
  - Control always resumes after the `if` statement
  - Conditions in `if` statements can be formed using equality or relational operators (next slide)

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	`==`	`x == y`	`x` is equal to `y`
≠	`!=`	`x != y`	`x` is not equal to `y`
Relational operators			
>	`>`	`x > y`	`x` is greater than `y`
<	`<`	`x < y`	`x` is less than `y`
≥	`>=`	`x >= y`	`x` is greater than or equal to `y`
≤	`<=`	`x <= y`	`x` is less than or equal to `y`

Equality and relational operators.

```java
// Fig. 2.15: Comparison.java
// Compare integers using if statements, relational operators
// and equality operators.
import java.util.Scanner; // program uses class Scanner

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

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

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

```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 1 of 4.)

```java
      if (number1 == number2)
         System.out.printf("%d == %d%n", number1, number2);
      }

      if (number1 != number2) {
         System.out.printf("%d != %d%n", number1, number2);
      }

      if (number1 < number2) {
         System.out.printf("%d < %d%n", number1, number2);
      }

      if (number1 > number2) {
         System.out.printf("%d > %d%n", number1, number2);
      }
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 2 of 4.)

```java

      if (number1 <= number2) {
         System.out.printf("%d <= %d%n", number1, number2);
      }

      if (number1 >= number2) {
         System.out.printf("%d >= %d%n", number1, number2);
      }
   } // end method main
} // end class Comparison
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 3 of 4.)

Output:

```
Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777
```

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 4 of 4.)

Decision Making: Equality and Relational Operators (Cont.)

- Line 6: begins class `Comparison` declaration
- Line 12: declares Scanner variable input and assigns it a Scanner that inputs data from the standard input
- Lines 14-15: declare `int` variables
- Lines 17-18: prompt the user to enter the first integer and input the value
- Lines 20-21: prompt the user to enter the second integer and input the value

Common Programming Error

- Confusing the equality operator, ==, with the assignment operator, =, can cause a logic error or a syntax error. The equality operator should be read as “is equal to,” and the assignment operator should be read as “gets” or “gets the value of.” To avoid confusion, some people read the equality operator as “double equals” or “equals equals.”

- It is a syntax error if the operators ==, !=, >= and <= contain spaces between their symbols, as in = =, ! =, > = and < =, respectively.

- Reversing the operators !=, >= and <=, as in =!, => and =<, is a syntax error.

Precedence and associativity of operations discussed.

Operators				Associativity	Type
*	/	%		left to right	multiplicative
+	-			left to right	additive
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment