



جامعة الجلالة  
GALALA UNIVERSITY

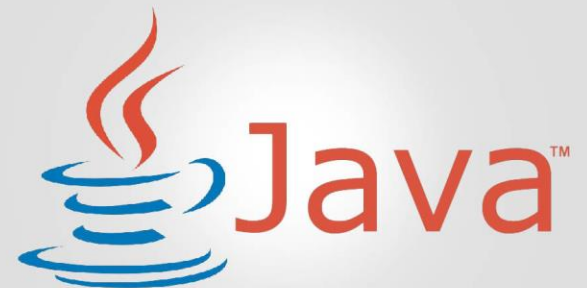
# CSE110 Principles of Programming

Lecture 2: Algorithms; Java Applications; Input/Output;  
and Operators

Professor Shaker El-Sappagh

[Shaker.elsappagh@gu.edu.eg](mailto:Shaker.elsappagh@gu.edu.eg)

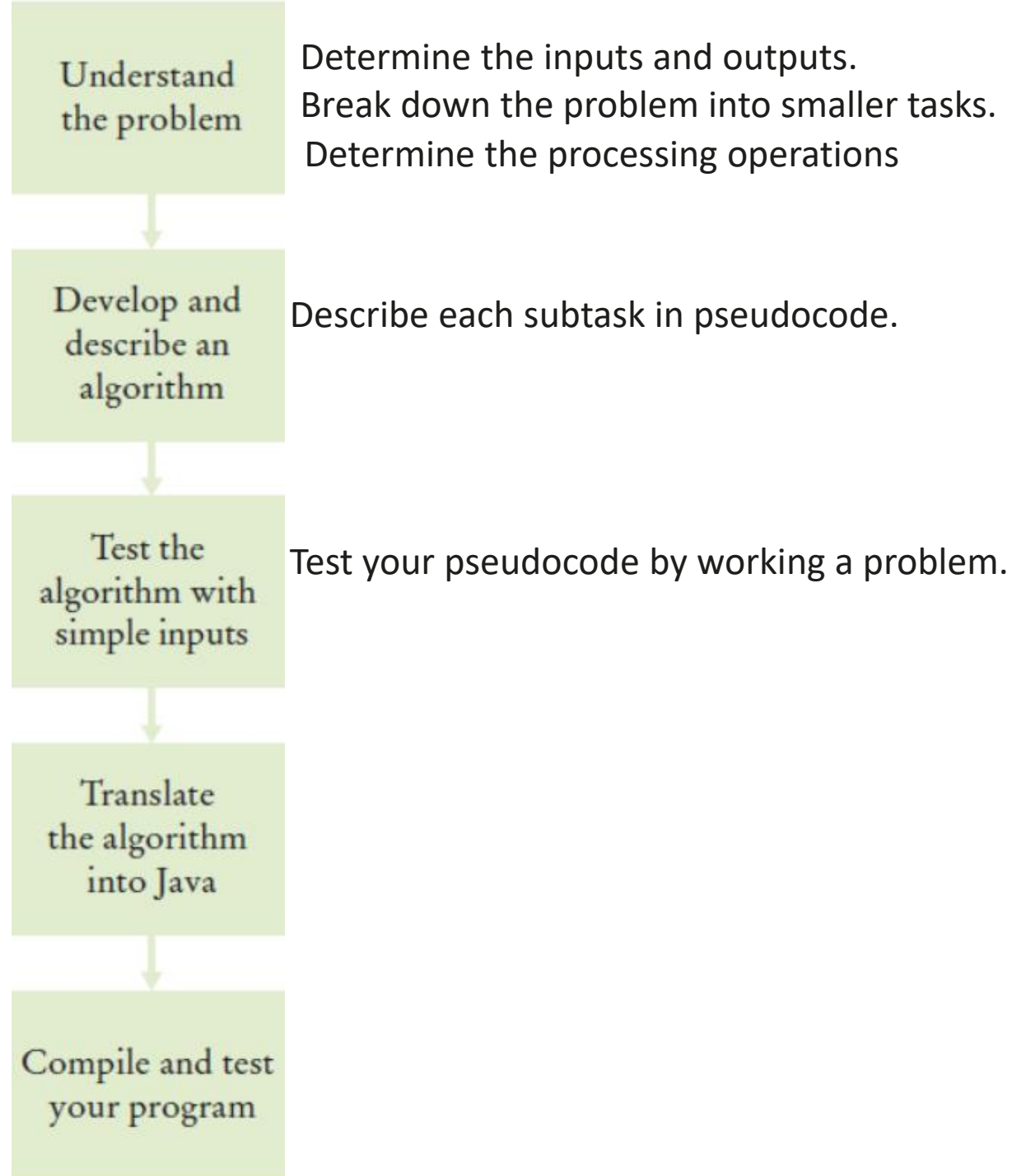
Fall 2023



# Outline

1. Problem Solving: Algorithm Design
2. Java Applications;
3. Input/Output;
4. Operators

# Problem Solving



# Problem Solving

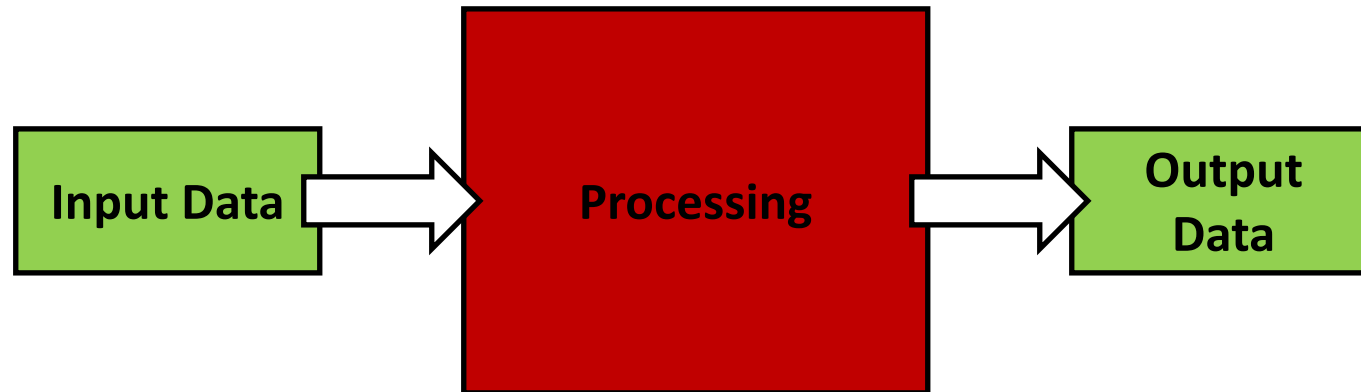
**So, any problem have the following components:**

- **Input:** Data to begin with to solve the program.
- **Output:** The target of the problem. This is the expected result.

Nouns in the problem statement suggest input and output data.

- **Processing:** This is the set of instructions that drive from the input to the output.

Verbs in the problem suggest the processing steps.



# Problem Solving

## Problem definition

**1** Write a program that prints the sum of two numbers.

Write a program that prints the average of three numbers.

Write a program that prints the volume of a cube and the sum of its surfaces' areas.

**2** Write a program that prints the sum of two numbers

➤ Input:

1<sup>st</sup> number (x1)

➔ Value=? ➔ entered by the user

2<sup>nd</sup> number (x2)

➔ Value=? ➔ entered by the user

➤ Output:

The summation (sum)

➤ Processing: (Input ➔ Output)

Summation = first + second

sum = x1 + x2

# Problem Solving

## Example 1 - Area and Perimeter of a Rectangle

Write a program that calculates the area and the perimeter of a rectangle of width and length 5 cm and 3 cm respectively.

### ➤ Input

Width = 5 cm

Length = 3 cm

### ➤ Output

Area = ?

Perimeter = ?

### ➤ Processing (Input → Output)

$\text{Area} = \text{length} * \text{width} = 5 * 3 = 15 \text{ cm}^2$

$\text{Perimeter} = 2 * (\text{length} + \text{width}) = 2 * (3 + 5) = 16 \text{ cm}$

**If the values of the input data are not given in the problem statement, they may be read from the user through the keyboard.**

# Problem Solving

## Example 2 - Sum and Average of 5 Numbers

Calculate the sum and the average of the numbers 10, 20, 30, 40 and 50.

➤ Input

Five numbers =  $x_1, x_2, x_3, x_4, x_5$

➤ Output

Sum = ?

Average = ?

➤ Processing (Input → Output)

Sum =  $x_1 + x_2 + x_3 + x_4 + x_5 = 10 + 20 + 30 + 40 + 50 = 150$

Average =  $\text{Sum} / 5 = 150 / 5 = 30$

**If the values of the input data are not given in the problem statement, they may be read from the user through the keyboard.**

# Problem Solving: Algorithm Design

## The Algorithm Concept

- A sequence of steps that is unambiguous, executable, and terminating is called an **algorithm**. An algorithm is a recipe for finding a solution
- **Example:** An Algorithm for Solving an Investment Problem

You put \$10,000 into a bank account that earns 5 percent interest per year. How many years does it take for the account balance to be double the original?

year	interest	balance
0		10000
1	$10000.00 \times 0.05 = 500.00$	$10000.00 + 500.00 = 10500.00$
2	$10500.00 \times 0.05 = 525.00$	$10500.00 + 525.00 = 11025.00$
3	$11025.00 \times 0.05 = 551.25$	$11025.00 + 551.25 = 11576.25$
4	$11576.25 \times 0.05 = 578.81$	$11576.25 + 578.81 = 12155.06$



# Problem Solving: Algorithm Design

**Example** (Cont'd): Computer can do this task faster and more accurately.

Set year to 0, balance to 10000.

year	interest	balance
0		10000

# Problem Solving: Algorithm Design

## Example (Cont'd):

While the balance is less than \$20,000

Add 1 to the year.

Set the interest to  $\text{balance} \times 0.05$  (i.e., 5 percent interest).

Add the interest to the balance.

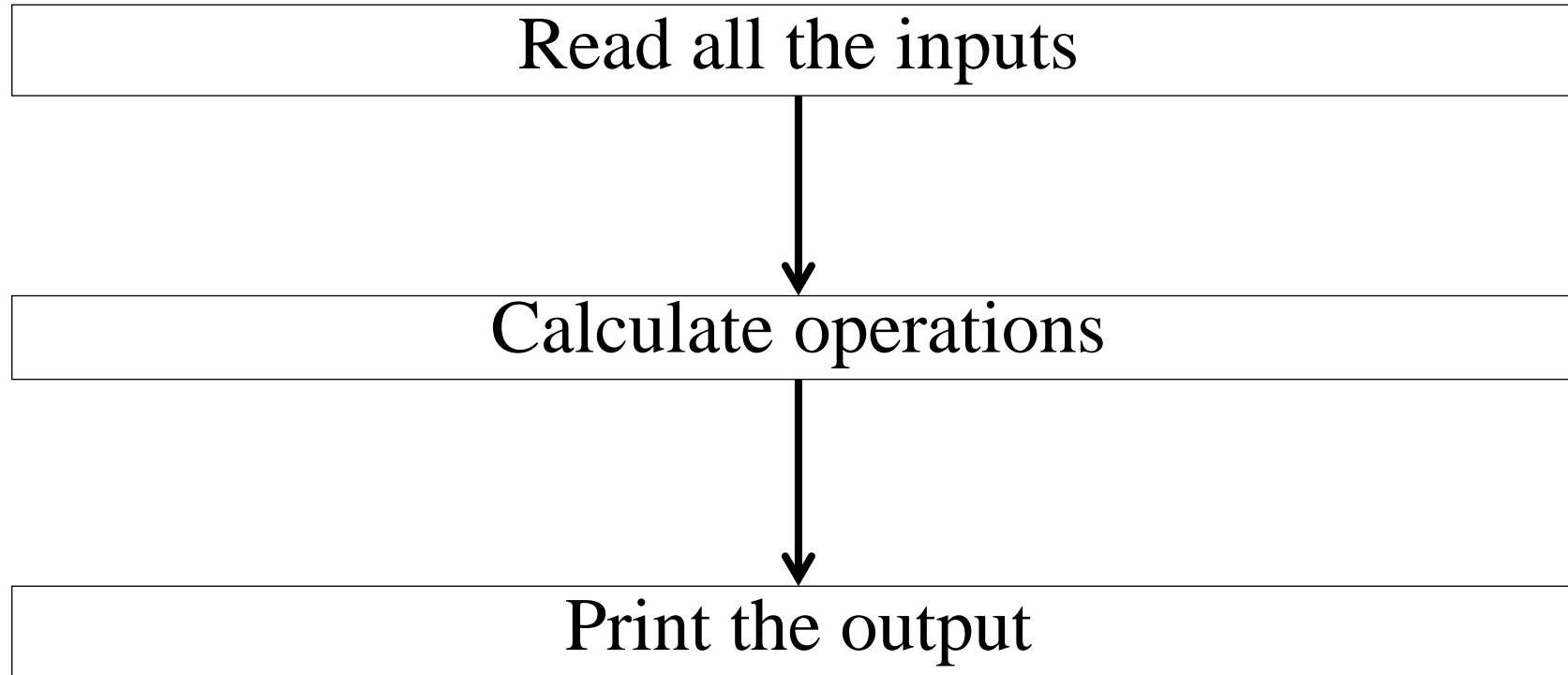
year	interest	balance
0		10000
1	500.00	10500.00
14	942.82	19799.32
15	989.96	20789.28

Report year as the answer.

**Pseudocode** is an informal description of a sequence of steps for solving a problem.

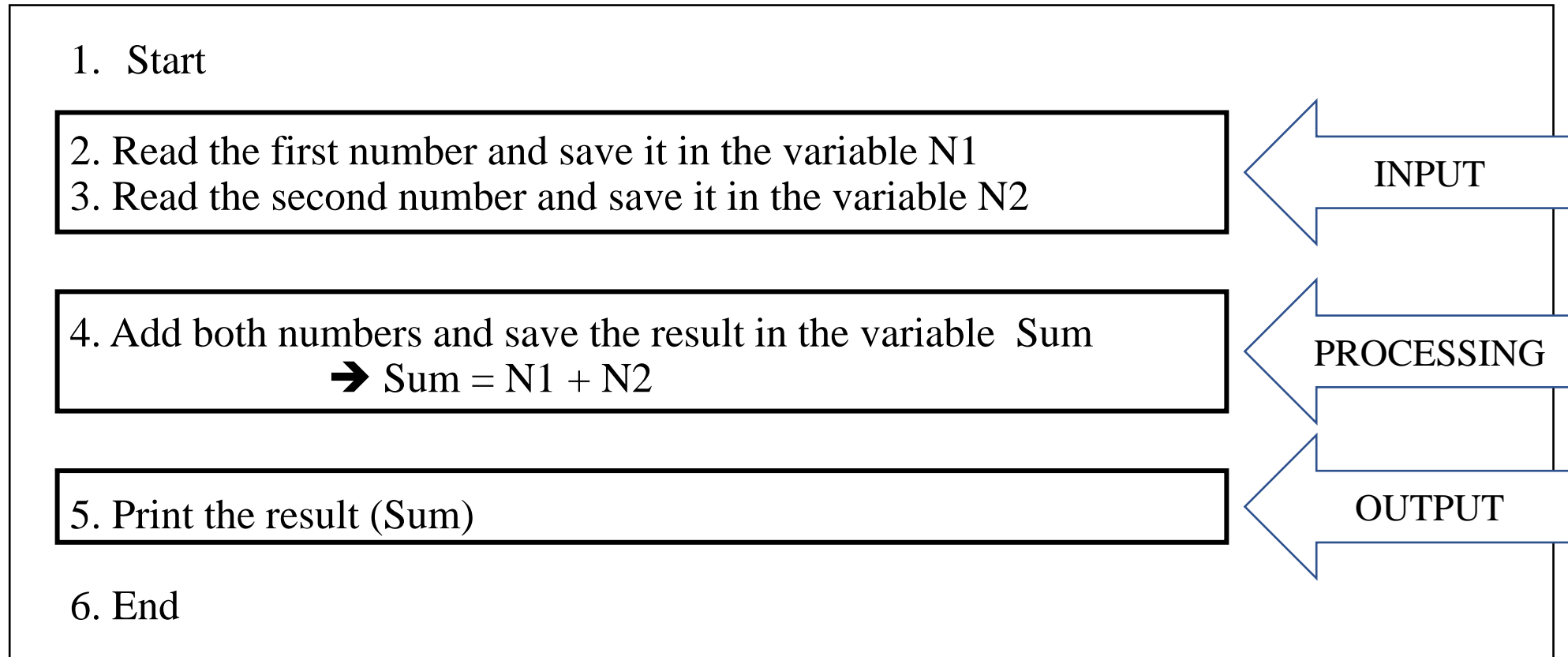
# Problem Solving: Algorithm Design

## Algorithm's Basic Steps



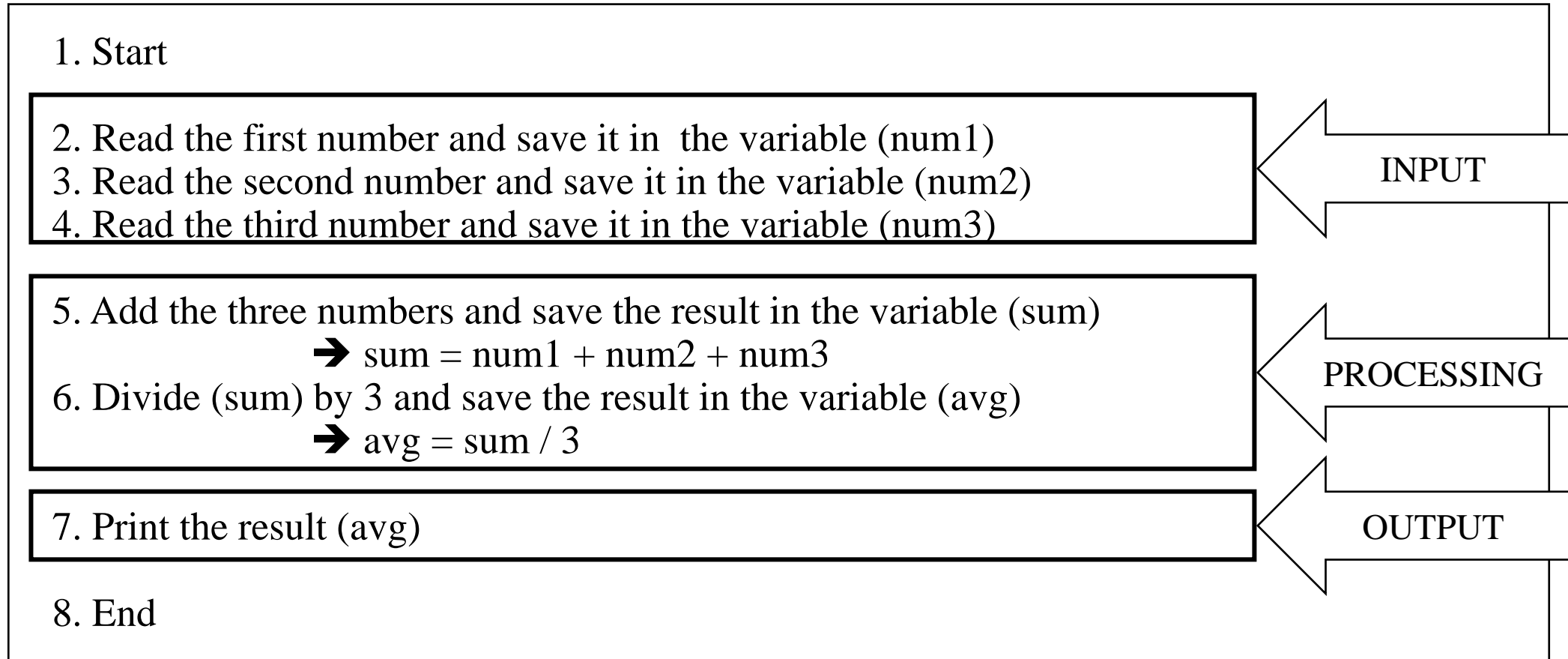
# Problem Solving: Algorithm Design

**EXAMPLE 1** Write a program that prints the sum of two numbers



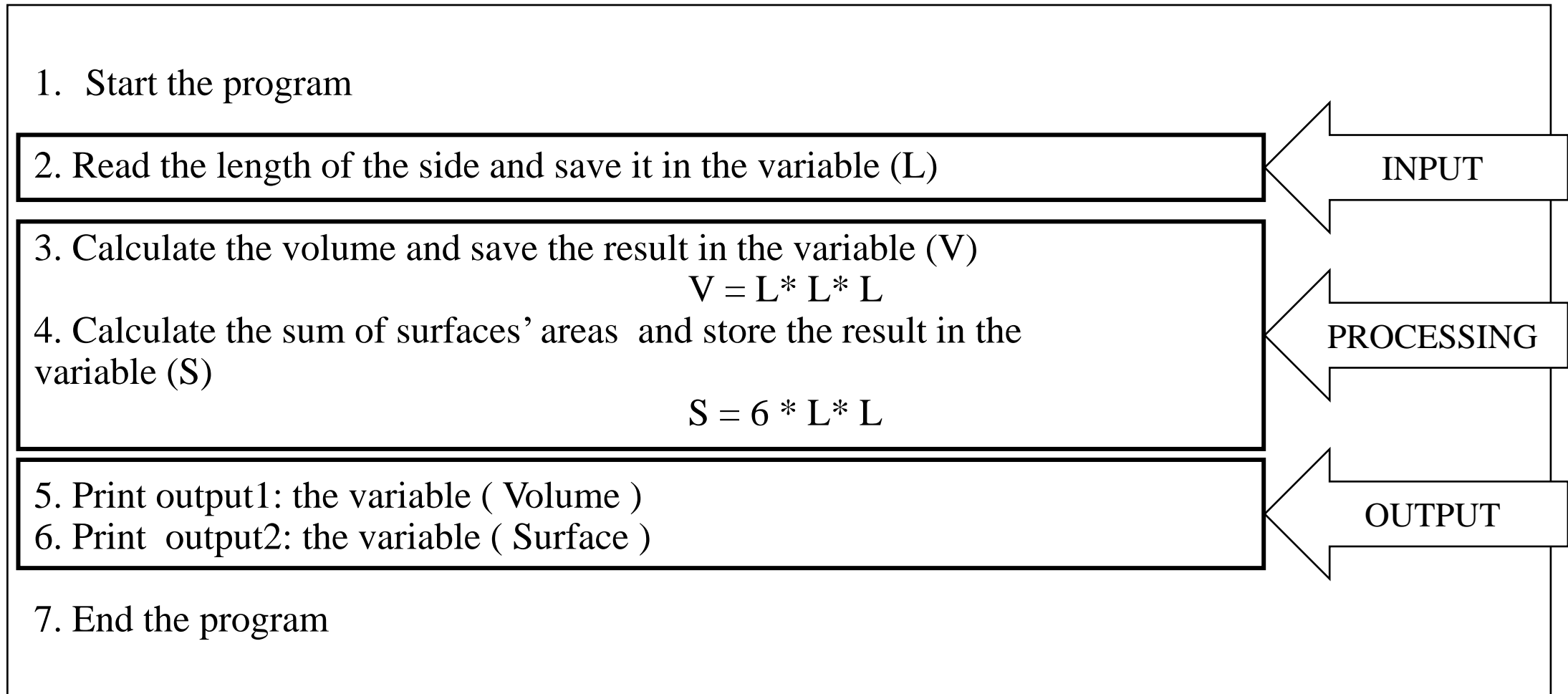
# Problem Solving: Algorithm Design

**EXAMPLE 2** Write a program that prints the average of three numbers



# Problem Solving: Algorithm Design

**EXAMPLE 3** Write a program that prints the volume of a cube and the sum of its surfaces' areas

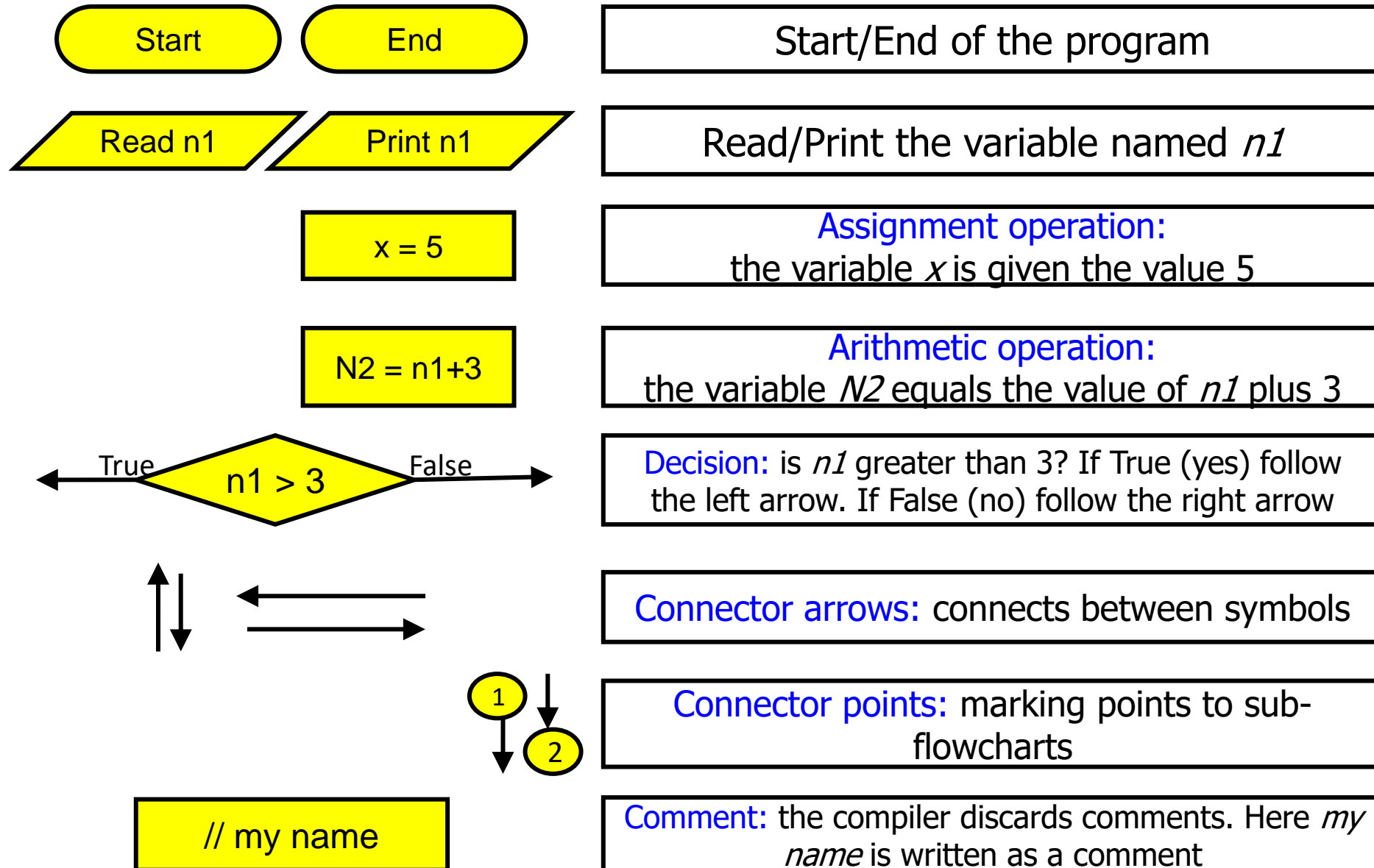


# Problem Solving: Algorithm Design

## Flowchart

- A flowchart is the **graphical representation** of the algorithm.
- The flowchart illustrates the previously developed algorithm in **more details**.
- These details make the algorithm **very close to the code implementation** in any high-level programming language.
- Both algorithm and flowchart are **independent of the used programming language**.
- The flowchart uses **standard symbols** to illustrate each action. These symbols are shown in the next slide.

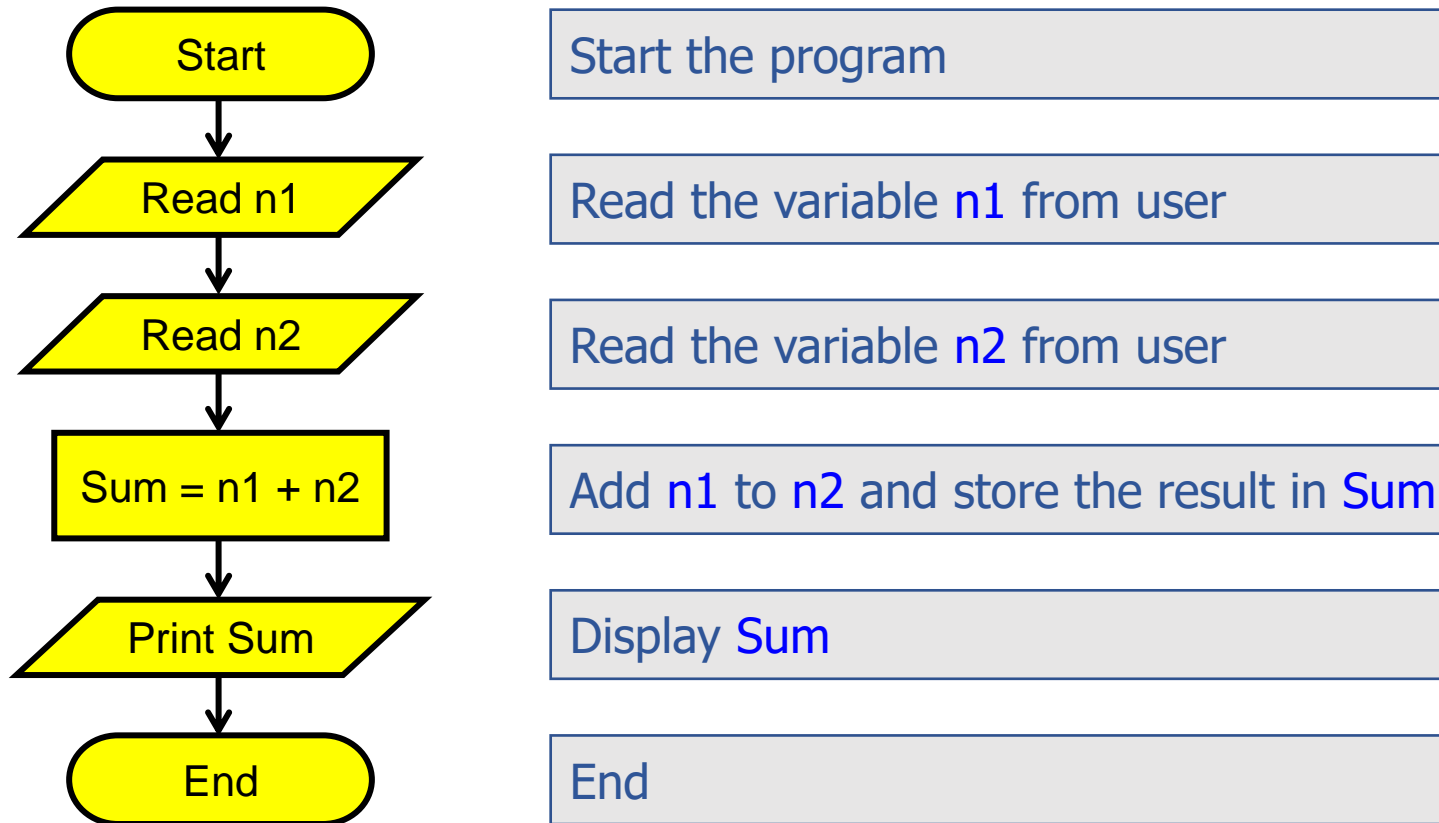
# Problem Solving: Algorithm Design: *Flowchart*





# Problem Solving: Algorithm Design: *Flowchart*

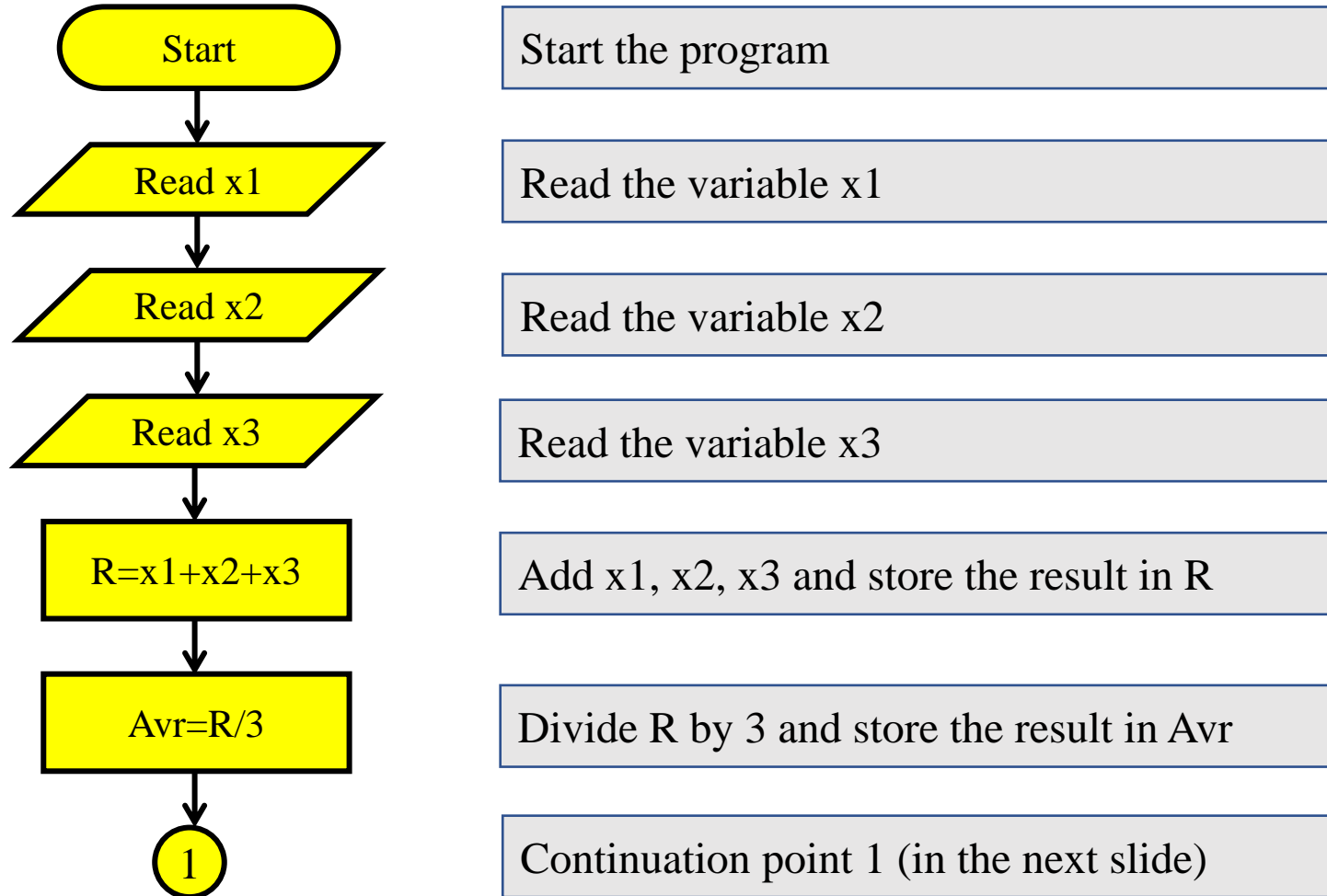
**EXAMPLE 1** Draw a flowchart that adds two numbers



Note that variables should have different names

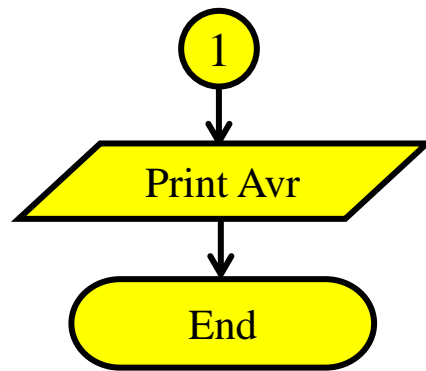
# Problem Solving: Algorithm Design: *Flowchart*

**EXAMPLE 2** Write a program that prints the average of three numbers



# Problem Solving: Algorithm Design: *Flowchart*

## EXAMPLE 2 (Cont'd)



Continuation point 1 (from previous slide)

Display Avr

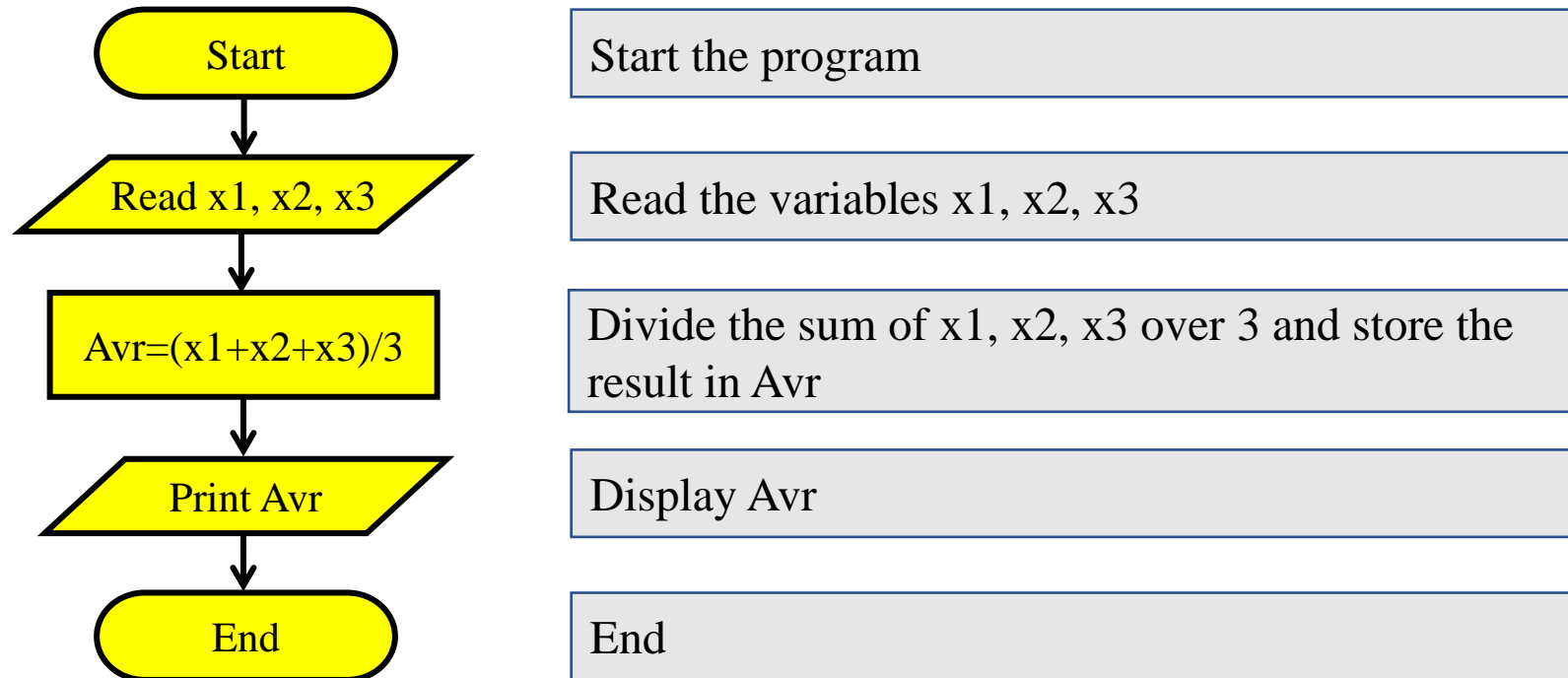
End

**Note that variables in the Right-Hand Side of any formula should have values. These are marked in blue in the flowchart.**

**Also, the variables to be printed should already have values.**

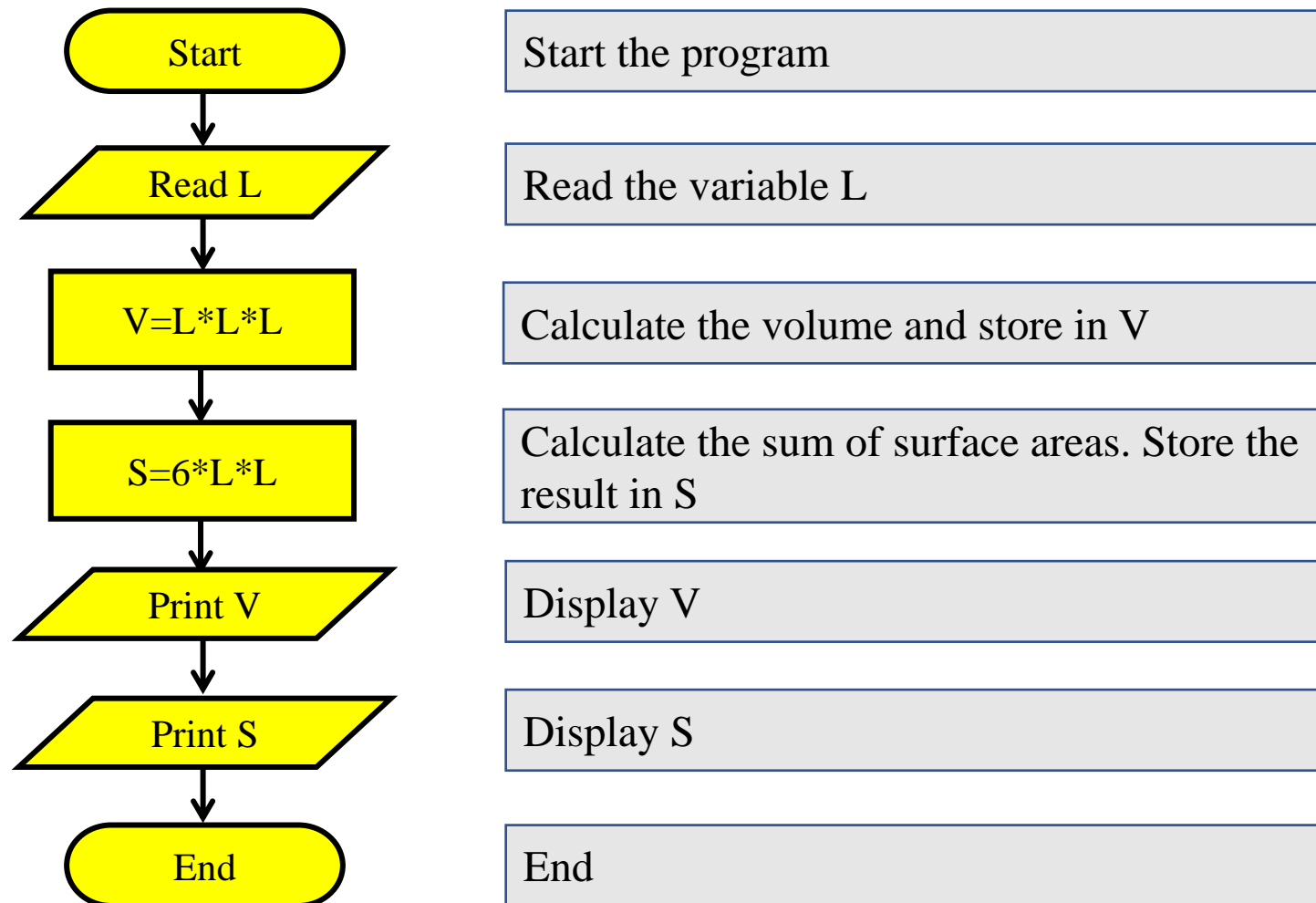
# Problem Solving: Algorithm Design: *Flowchart*

**EXAMPLE 2 (cont'd)** This is a more simplified way of the flowchart of Example 2



# Problem Solving: Algorithm Design: *Flowchart*

**EXAMPLE 3** Write a program that prints the volume of a cube and the sum of its surfaces' areas



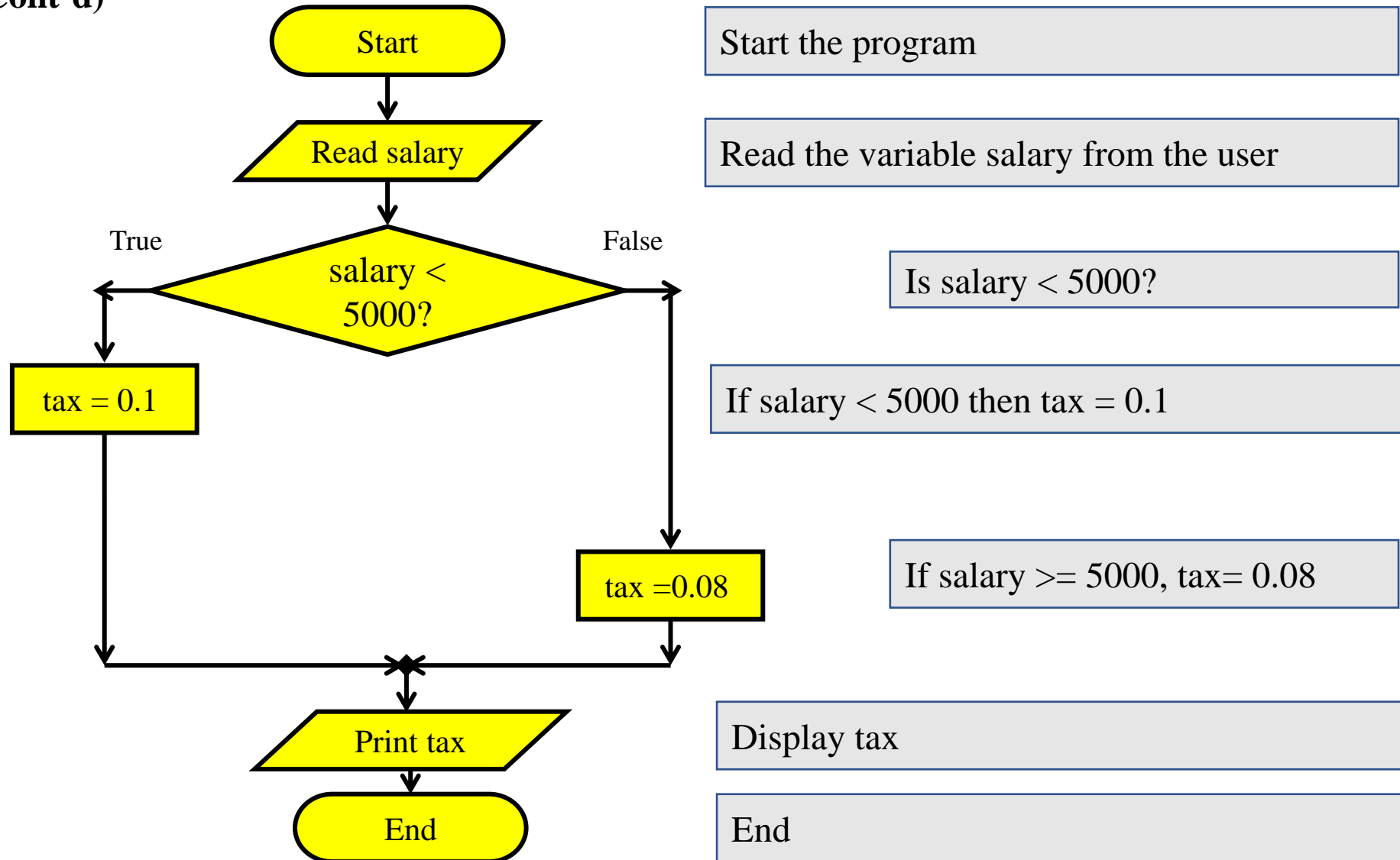
# Problem Solving: Algorithm Design: *Flowchart*

**Example 4: Write a program that calculates the required tax based on the employer's salary. If the salary is less than 5,000 SAR then tax = 10% of the salary; otherwise, the tax = 8%.**

- **Input:**  
Employer's salary (salary)  
    ➔ Value=? ➔ entered by the user
- **Output:**  
The required tax (tax)
- **Processing (Input ➔ Output)**  
If salary < 5000 then tax = 0.1  
If salary >= 5000 then tax = 0.08

# Problem Solving: Algorithm Design: *Flowchart*

## Example 4 (Cont'd)



# Problem Solving: Algorithm Design: *Flowchart*

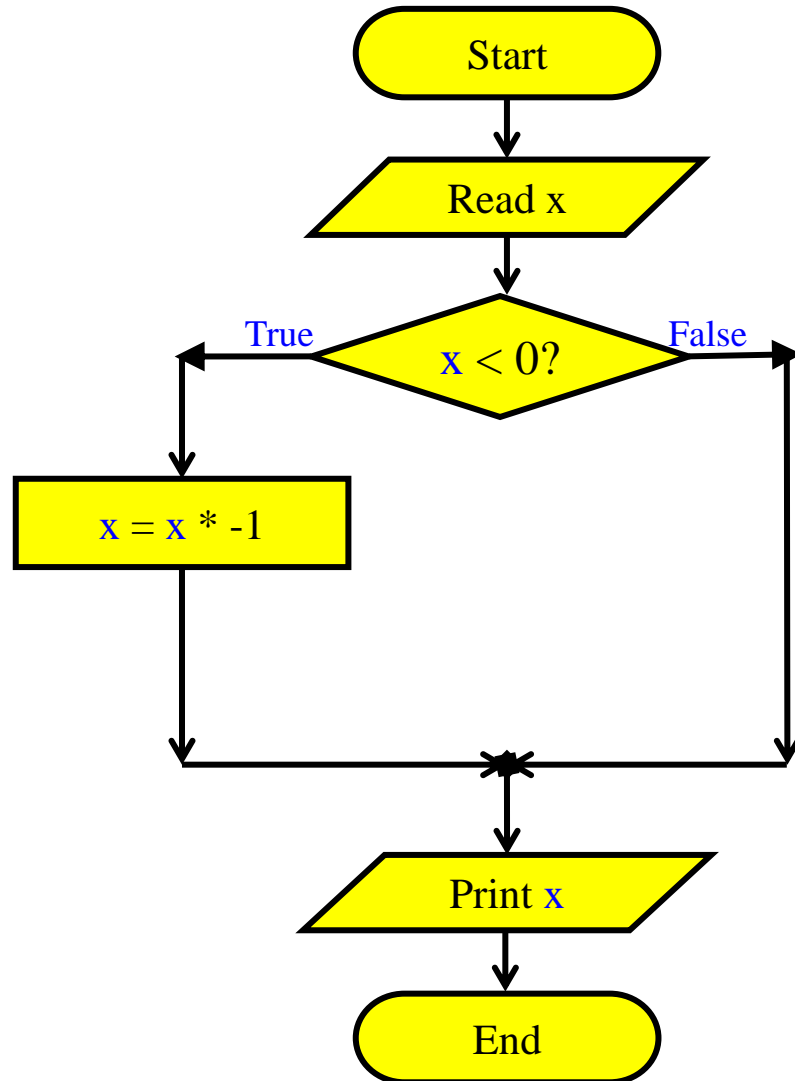
**Example 5: Write a program that prints the absolute value of a number  $x$**

- Input:  
A number ( $x$ )  
  
➔ Value=? ➔ entered by the user
- Output:  
The absolute value of  $x$  ( $|x|$ )
- Processing (Input ➔ Output)  
If  $x \geq 0$  then do nothing  
If  $x < 0$  then  $x = x * -1$



# Problem Solving: Algorithm Design: *Flowchart*

## Example 5 (Cont'd)



Start the program

Read the variable  $x$  from the user

Is  $x$  negative?

If  $x$  is negative then multiply  $x$  by  $-1$ .  
Store the new value in  $x$

If  $x$  is not negative then do nothing

Display  $x$

End

# Your First Program in Java: Printing a Line of Text

## Setting up JDK and Eclipse

```
public class ClassName
{
    public static void main(String[] args)
    {
        . . .
    }
}
```

# Your First Program in Java: Printing a Line of Text


Every Java program contains a main method with this header.

The statements inside the main method are executed when the program runs.

Be sure to match the opening and closing braces.

```
public class HelloPrinter
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Every program contains at least one class. Choose a class name that describes the program action.

Each statement ends in a semicolon.  
 See Common Error 1.1.

Replace this statement when you write your own programs.

# Your First Program in Java: Printing a Line of Text

---

```
1  // Fig. 2.1: Welcome1.java
2  // Text-printing program.
3
4  public class Welcome1 {
5      // main method begins execution of Java application
6      public static void main(String[] args) {
7          System.out.println("Welcome to Java Programming!");
8      } // end method main
9  } // end class Welcome1
```

---

Welcome to Java Programming!

- **Commenting Your Programs**: // Fig. 2.1: Welcome1.java  
/\* This is a traditional comment. It  
can be split over multiple lines \*/

# Declaring a Class

```
public class Welcome1 {
```

- Every Java program consists of **at least one class** that you define.
- The `class` keyword introduces a class declaration and is immediately followed by the class name (Welcome1).
- **Keywords** are reserved for use by Java and are spelled with all lowercase letters.
- Every class we define begins with the `public` keyword.
- A `public` class must be placed in a file that has a filename of the form `ClassName.java`, so class Welcome1 is stored in the file Welcome1.java.



## 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.*

# Declaring a Class

## Class Names and Identifiers

- By convention, class names **begin with a capital letter** and capitalize the first letter of each word they include (e.g., `SampleClassName`).
- A class name is an **identifier**—a series of characters consisting of letters, digits, underscores (`_`) and dollar signs (`$`) that **does not** begin with a digit and **does not** contain spaces.
- Some valid identifiers are `Welcome1`, `$value`, `_value`, `m_inputField1` and `button7`.
- The name *7button* is not a valid identifier because it begins with a digit, and the name *input field* is not a valid identifier because it contains a space.
- **Normally**, an identifier that does not begin with a capital letter is **not** a class name.
- Java is **case sensitive**—uppercase and lowercase letters are distinct—so `value` and `Value` are different (but both valid) identifiers.

# Declaring a Class

## Class Names and Identifiers

**Table 1-3** The Java key words

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



# Declaring a Class

## Class body

A **left brace** (at the end of line 4), `{`, begins the body of every class declaration. A **corresponding right brace** (at line 9), `}`, must end each class declaration.



### Common Programming Error 2.3

*It's a syntax error if braces do not occur in matching pairs.*



### Error-Prevention Tip 2.1

*When you type an opening left brace, `{`, 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. Many IDEs do this for you.*



# Declaring a Class

## Declaring a Method

```
public static void main(String[] args) {
```

- is the starting point of every Java application.
- The parentheses after the identifier `main` indicate that it's a program building block called a **method**.
- Java class declarations normally contain **one or more** methods.
- For a Java application, one of the methods **must be called *main*** and **must be defined as in line 6; otherwise, the program will not execute**.
- Methods perform tasks and can return information when they complete their tasks.
- Keyword **void** indicates that this method will not return any information.
- The **String[] args** in parentheses is a required part of **main**'s declaration.
- The **left brace** at the end of line 6 begins the body of the method declaration. A corresponding **right brace** ends it (line 8).

# Declaring a Class

## Performing Output with `System.out.println`

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

- **instructs** the computer to perform an action—namely, to display the characters between the double quotation marks.
- The quotation marks themselves **are not displayed**.
- Together, the quotation marks and the characters between them are a **string**—also known as a **character string** or a **string literal**.
- White-space characters in strings are **not ignored by the compiler**.
- Strings **cannot** span multiple lines of code.
- The `System.out` object is known as the **standard output object**.
- Method `System.out.println` displays (or prints) a line of text in the command window.

# Compiling the Application

1. open a command window and change to the directory where the program is stored.

```
cd c:\examples\ch02\fig02_01
```

2. To compile the program, type

```
javac Welcome1.java
```

3. If the program does not contain compilation errors, this command creates the file called *Welcome1.class* (known as Welcome1's class file) containing the **platform-independent Java bytecodes** that represent our application.
4. When we use the *java* command to execute the application on a given platform, the JVM will **translate these bytecodes into instructions** that are understood by the underlying **operating system and hardware**.



## Common Programming Error 2.4

*The compiler error message “class Welcome1 is public, should be declared in a file named Welcome1.java” indicates that the filename does not match the name of the public class in the file or that you typed the class name incorrectly when compiling the class.*



## Error-Prevention Tip 2.2

*When the compiler reports a syntax error, it may not be on the line that the error message indicates. First, check the line for which the error was reported. If you don't find an error on that line, check several preceding lines.*

# Executing the Application

- Now that you've compiled the program, type the following command and press Enter:

```
java Welcome1
```


- to launch the JVM and load the Welcome1.class file.
- The command *omits* the .class filename extension; otherwise, the JVM will not execute the program.
- The JVM calls Welcome1's **main** method. Next, line 7 of *main* displays "Welcome to Java Programming!".



## Error-Prevention Tip 2.3

*When attempting to run a Java program, if you receive a message such as “Exception in thread “main” java.lang.NoClassDefFoundError: Welcome1,” your CLASSPATH environment variable has not been set properly. Please carefully review the installation instructions in the Before You Begin section of this book. On some systems, you may need to reboot your computer or open a new command window after configuring the CLASSPATH.*

# Executing the Application



```
Command Prompt
c:\examples\ch02\fig02_01>javac Welcome1.java
c:\examples\ch02\fig02_01>java Welcome1
Welcome to Java Programming!
c:\examples\ch02\fig02_01>
```

The screenshot shows a Windows Command Prompt window with the title 'Command Prompt'. The command prompt shows the directory 'c:\examples\ch02\fig02\_01'. The first command entered is 'javac Welcome1.java', which compiles the Java file. The second command is 'java Welcome1', which runs the application. The output of the application is 'Welcome to Java Programming!'. The prompt then returns to 'c:\examples\ch02\fig02\_01>'.

You type this command to execute the application

The program outputs to the screen  
Welcome to Java Programming!



# Executing the Application

- Displaying a Single Line of Text with Multiple Statements

```
1 // Fig. 2.3: Welcome2.java
2 // Printing a line of text with multiple statements.
3
4 public class Welcome2 {
5     // main method begins execution of Java application
6     public static void main(String[] args) {
7         System.out.print("Welcome to ");
8         System.out.println("Java Programming!");
9     } // end method main
10 } // end class Welcome2
```

Welcome to Java Programming!

# Executing the Application

- Displaying Multiple Lines of Text with a Single Statement

---

```
1 // Fig. 2.4: Welcome3.java
2 // Printing multiple lines of text with a single statement.
3
4 public class Welcome3 {
5     // main method begins execution of Java application
6     public static void main(String[] args) {
7         System.out.println("Welcome\nto\nJava\nProgramming!");
8     } // end method main
9 } // end class Welcome3
```

```
Welcome
to
Java
Programming!
```

# Executing the Application

- The escape character

The backslash (\) is an escape character, which has special meaning to `System.out`'s `print` and `println` methods.

Escape sequence	Description
<code>\n</code>	Newline. Position the screen cursor at the beginning of the <i>next</i> line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor at the beginning of the <i>current</i> line—do <i>not</i> advance to the next line. Any characters output after the carriage return <i>overwrite</i> the characters previously output on that line.
<code>\\</code>	Backslash. Used to print a backslash character.
<code>\"</code>	Double quote. Used to print a double-quote character. For example, <pre>System.out.println("\"in quotes\");</pre> displays "in quotes".



# Reading from Keyboard: Adding Integers Application

```
import java.util.Scanner;
```

Gets the Scanner class from the package (library) java.util

```
public class FirstProgram  
{
```

Name of the class—your choice. "This program should be in a file named FirstProgram.java"

```
    public static void main(String[] args)  
    {
```

```
        System.out.println("Hello out there.");  
        System.out.println("I will add two numbers for you.");  
        System.out.println("Enter two whole numbers on a line:");
```

```
        int n1, n2;
```

Says that n1 and n2 are variables that hold integers (whole numbers)

```
        Scanner keyboard = new Scanner(System.in);
```

Readies the program for keyboard input

```
        n1 = keyboard.nextInt();  
        n2 = keyboard.nextInt();
```

Reads one whole number from the keyboard

```
        System.out.println("The sum of those two numbers is");  
        System.out.println(n1 + n2);
```

```
    }
```

```
}
```

## Sample Screen Output

```
Hello out there.  
I will add two numbers for you.  
Enter two whole numbers on a line:  
12 30  
The sum of those two numbers is  
42
```

# Reading from Keyboard: Adding Integers Application

- Our next application reads (or inputs) two integers typed by a user at the keyboard, computes their sum and displays it.

---

```
1 // Fig. 2.7: Addition.java
2 // Addition program that inputs two numbers then displays their sum.
3 import java.util.Scanner; // program uses class Scanner
4
5 public class Addition {
6     // main method begins execution of Java application
7     public static void main(String[] args) {
8         // create a Scanner to obtain input from the command window
9         Scanner input = new Scanner(System.in);
10
11         System.out.print("Enter first integer: "); // prompt
12         int number1 = input.nextInt(); // read first number from user
13
14         System.out.print("Enter second integer: "); // prompt
15         int number2 = input.nextInt(); // read second number from user
16
17         int sum = number1 + number2; // add numbers, then store total in sum
18
19         System.out.printf("Sum is %d\n", sum); // display sum
20     } // end method main
21 } // end class Addition
```

---

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



# import Declarations

- A great strength of Java is its rich set of **predefined classes** that you can reuse rather than “reinventing the wheel.”
- These classes are grouped into **packages**—named groups of related classes—and are collectively referred to as the **Java class library**, or the **Java Application Programming Interface** (Java API).

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

- is an **import** declaration that helps the compiler locate a class that’s used in this program. It indicates that the program uses the predefined **Scanner** class from the package named **java.util**.



## Common Programming Error 2.5

*All import declarations must appear before the first class declaration in the file. Placing an import declaration inside or after a class declaration is a syntax error.*



## Common Programming Error 2.6

*Forgetting to include an import declaration for a class that must be imported results in a compilation error containing a message such as “cannot find symbol.” When this occurs, check that you provided the proper import declarations and that the names in them are correct, including proper capitalization.*

# Declaring and Creating a Scanner to Obtain User Input from the Keyboard

- A **variable** is a location in the computer's memory where a value can be stored for use later in a program.
- All Java variables must be declared with a *name* and a *type* **before** they can be used.
- A variable's name enables the program to access the variable's value in memory.
- A **variable name** can be any **valid identifier**—again, a series of characters consisting of letters, digits, underscores (\_) and dollar signs (\$) that does not begin with a digit and does not contain spaces.
- A **variable's type** specifies what kind of information is stored at that location in memory. Declaration statements end with a **semicolon** (;).
- A **Scanner** `Scanner input = new Scanner(System.in);` (e.g., numbers and strings) for use in a program. The data can come from many sources, such as the user at the **keyboard** or a **file** on disk.
- Before using a Scanner, you must **create** it and specify the **source of the data**.
- The **standard input object**, `System.in`, enables applications to **read bytes of data** typed by the user.
- **The Scanner translates these bytes into types (like ints) that can be used in a program.**

# Prompting the User for Input

Line 11

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

- uses `System.out.print` to display the message "Enter first integer: ". This message is called a prompt because it directs the user to take a specific action.
- Class `System` is part of package `java.lang`.



## Software Engineering Observation 2.1

*By default, package `java.lang` is imported in every Java program; thus, classes in `java.lang` are the only ones in the Java API that do not require an import declaration.*

## Declaring a Variable to Store an Integer and Obtaining an Integer from the Keyboard

- The variable declaration statement in line 12.

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

- declares that variable *number1* holds data of type **int**—that is, integer values, which are whole numbers such as 72, −1127 and 0.
- The range of values for an **int** is −2,147,483,648 to +2,147,483,647.

- **Obtaining a Second Integer**

- **Using Variables in a Calculation**

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

- The addition operator is a **binary operator**, because it has two operands—*number1* and *number2*.
- Portions of statements that contain calculations are called **expressions**.

# Displaying the Calculation Result

- After the calculation has been performed. line 19

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

- uses method `System.out.printf` to display the sum. The format specifier `%d` is a placeholder for an int value.
- Calculations also can be performed inside `printf` statements.

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



# Declaring and Initializing Variables in Separate Statements

- Each variable **must have a value before** you can use the variable in a calculation.
- The variable **declaration** and **initialization** could be done in the same command.
- Sometimes you declare a variable in one statement, then initialize in another.

```
int number1; // declare the int variable number1  
number1 = input.nextInt(); // assign the user's input to number1
```

- The **first statement** declares number1, but does not initialize it. The **second statement** uses the assignment operator, =, to assign (that is, give) number1 the value entered by the user.



# Memory Concepts

- **Variable** names such as *number1*, *number2* and *sum* correspond to **locations** in the computer's memory.
- Every variable has a **name**, a **type**, a **size** (in bytes) and a **value**.

# Arithmetic

Java operation	Operator	Algebraic expression	Java expression
Addition	+	$f + 7$	<code>f + 7</code>
Subtraction	-	$p - c$	<code>p - c</code>
Multiplication	*	$bm$	<code>b * m</code>
Division	/	$x / y$ or $\frac{x}{y}$ or $x \div y$	<code>x / y</code>
Remainder	%	$r \bmod s$	<code>r % s</code>

**Integer division** yields an **integer quotient**. For example, the expression `7 / 4` evaluates to 1, and the expression `17 / 5` evaluates to 3.

Any fractional part in integer division is simply truncated (i.e., discarded)—no rounding occurs.

Java provides the remainder operator, `%`, which yields the remainder after division.

# Arithmetic Expressions in Straight-Line Form

- Parentheses for Grouping Subexpressions.
- Rules of Operator Precedence:
  - Multiplication, division and remainder operations are applied first.
  - Addition and subtraction operations are applied next.

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

# Sample Algebraic and Java Expressions

*Algebra:*  $m = \frac{a + b + c + d + e}{5}$

*Java:* `m = (a + b + c + d + e) / 5;`

*Algebra:*  $z = pr \% q + w / x - y$

*Java:* `z = p * r % q + w / x - y;`

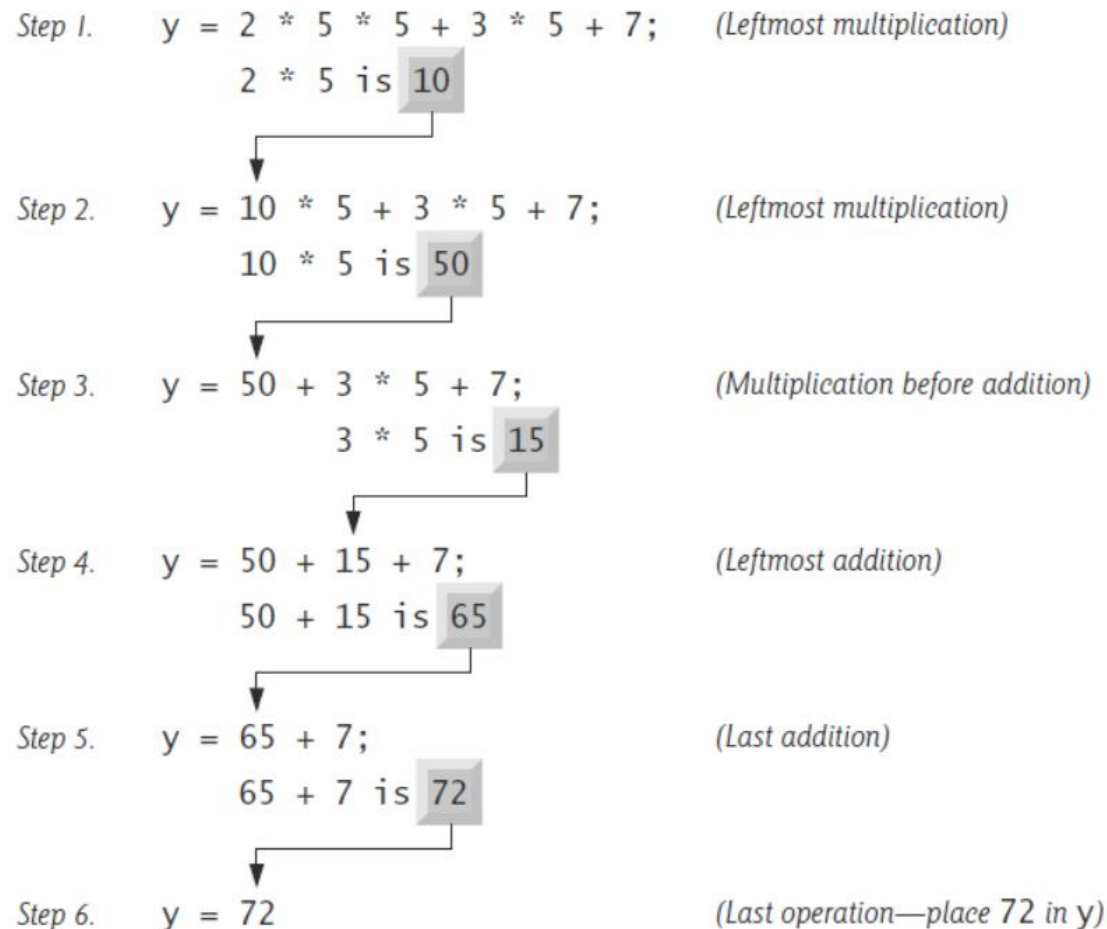


# Evaluation of a Second-Degree Polynomial

•  $y = a * x * x + b * x + c;$

6      1      2      4      3      5

Suppose that  $a$ ,  $b$ ,  $c$  and  $x$  are initialized (given values) as follows:  
 $a = 2$ ,  $b = 3$ ,  $c = 7$  and  $x = 5$ .



# Decision Making: Equality and Relational Operators

- A **condition** is an expression that can be **true** or **false**.
- Java's **if selection statement**, which allows a program to make a decision based on a condition's value.
- For example, the condition "grade is greater than or equal to 60" determines whether a student passed a test.
- If an **if statement's condition is true**, its body executes. If the **condition is false**, its body does not execute.
- Conditions in if statements can be formed by using the equality operators (== and !=) and relational operators (>, <, >= and <=)
- Both equality operators have the **same level of precedence**, which is **lower** than that of the relational operators.
- The equality operators associate from **left to right**.
- The relational operators all have the **same level of precedence** and also associate from **left to right**.

# Decision Making: Equality and Relational Operators

Algebraic operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	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



# Decision Making: Equality and Relational Operators

---

```
1 // Fig. 2.15: Comparison.java
2 // Compare integers using if statements, relational operators
3 // and equality operators.
4 import java.util.Scanner; // program uses class Scanner
5
6 public class Comparison {
7     // main method begins execution of Java application
8     public static void main(String[] args) {
9         // create Scanner to obtain input from command line
10        Scanner input = new Scanner(System.in);
11
12        System.out.print("Enter first integer: "); // prompt
13        int number1 = input.nextInt(); // read first number from user
14
15        System.out.print("Enter second integer: "); // prompt
16        int number2 = input.nextInt(); // read second number from user
```



# Decision Making: Equality and Relational Operators

```
18     if (number1 == number2)
19         System.out.printf("%d == %d\n", number1, number2);
20     }
21
22     if (number1 != number2) {
23         System.out.printf("%d != %d\n", number1, number2);
24     }
25
26     if (number1 < number2) {
27         System.out.printf("%d < %d\n", number1, number2);
28     }
29
30     if (number1 > number2) {
31         System.out.printf("%d > %d\n", number1, number2);
32     }
33
34     if (number1 <= number2) {
35         System.out.printf("%d <= %d\n", number1, number2);
36     }
37
38     if (number1 >= number2) {
39         System.out.printf("%d >= %d\n", number1, number2);
40     }
41 } // end method main
42 } // end class Comparison
```

# Decision Making: Equality and Relational Operators

```
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
```

# Decision Making: Equality and Relational Operators



## Good Programming Practice 2.11

*Indent the statement(s) in the body of an `if` statement to enhance readability. IDEs typically do this for you, allowing you to specify the indent size.*



## Error-Prevention Tip 2.4

*You don't need to use braces, `{ }`, around single-statement bodies, but you must include the braces around multiple-statement bodies. You'll see later that forgetting to enclose multiple-statement bodies in braces leads to errors. To avoid errors, as a rule, always enclose an `if` statement's body statement(s) in braces.*



## Common Programming Error 2.7

*Placing a semicolon immediately after the right parenthesis after the condition in an `if` statement is often a logic error (although not a syntax error). The semicolon causes the body of the `if` statement to be empty, so the `if` statement performs no action, regardless of whether or not its condition is true. Worse yet, the original body statement of the `if` statement always executes, often causing the program to produce incorrect results.*

# Precedence and associativity of operators

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

# Errors

```
System.ou.println("Hello, World!");  
System.out.println("Hello, Word!");
```



## In the first case:

- the compiler will complain. It will say that it has no clue what you mean by **ou**.
- This is a **compile-time error**. Something is wrong according to the **rules of the language** and the compiler finds it. For this reason, compile-time errors are often called **syntax errors**.
- When the compiler finds one or more errors, it refuses to translate the program into Java virtual machine instructions, and as a consequence you have no program that you can run.
- **You must fix the error and compile again.**
- The error in the **second line** is of a different kind. The program will compile and run, but its output will be wrong. (**Validation error**).
- What about the **run-time error**? `System.out.println(1 / 0);`



# Exercises

What does this program print?

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("39 + 3");
        System.out.println(39 + 3);
    }
}
```

What is the compile-time error in this program?

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Hello", "World!");
    }
}
```

What does this program print? Pay close attention to spaces.

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.print("Hello");
        System.out.println("World");
    }
}
```

Thank you