Object Oriented Programming (JAVA)



## Semester: Fall 2024

**Software Engineering**

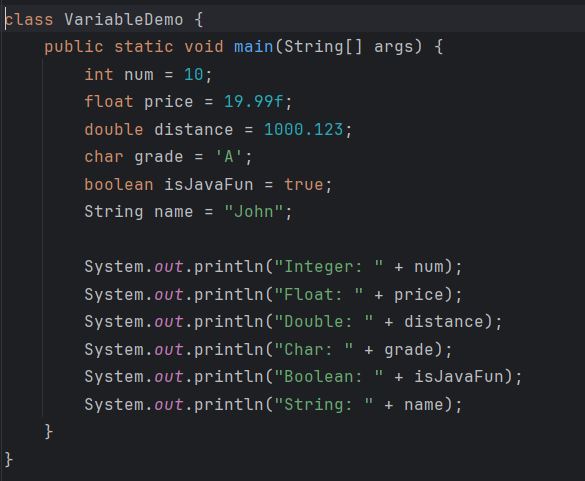
**Faculty of Information Technology UCP Lahore, Pakistan**

|  |  |
| --- | --- |
| **Week 2** | |
| **Topic** | **Basic Syntax and Control Structures in Java, Methods and Parameters in Java** |
| **Objective** | * Understanding and implementing basic data types in java * Writing and running basic java programs that involves the variables, conditionals, loops and operators, etc. * Implementation of methods in java, static vs. public methods |

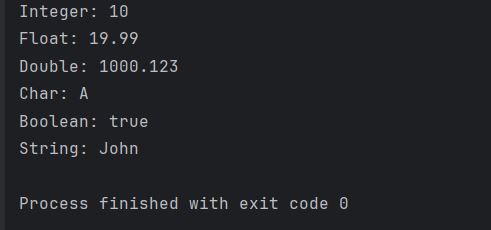
**Lab Task:** Declare different types of variables and print their values.

**Steps**:

1. Declare variables of types int, float, double, char, boolean, and String.
2. Initialize them with some values.
3. Print the values of these variables.



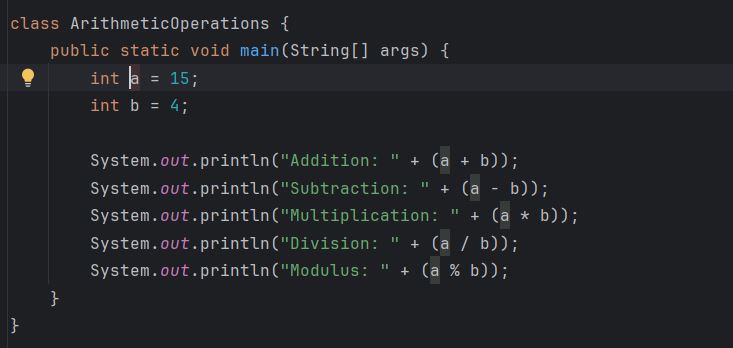
**Output:**



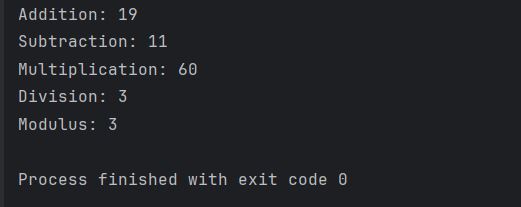
**Lab Task:** Perform arithmetic operations using variables.

**Steps:**

1. Declare two integer variables.
2. Perform addition, subtraction, multiplication, division, and modulus operations.
3. Print the results of each operation.

****

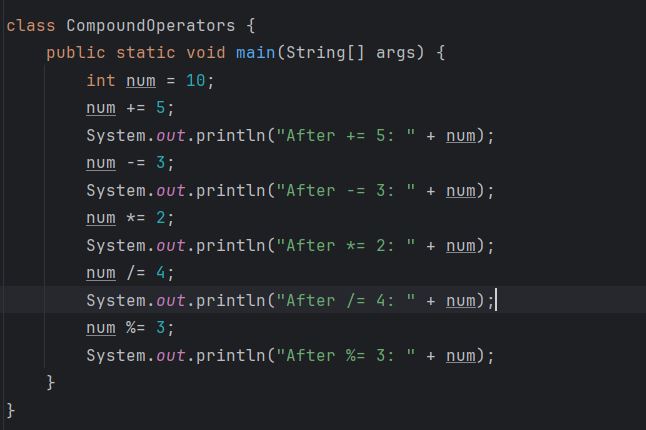
**Output:**

****

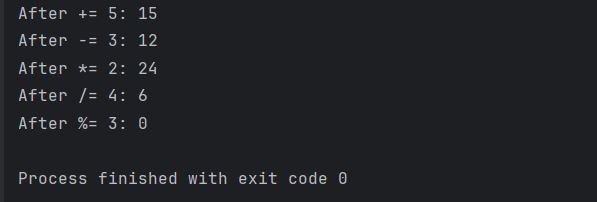
**Lab Task:** Use compound assignment operators to modify variable values.

**Steps:**

1. Declare an integer variable and initialize it.
2. Use the +=, -=, \*=, /=, and %= operators to modify the variable.
3. Print the updated value after each operation.

****

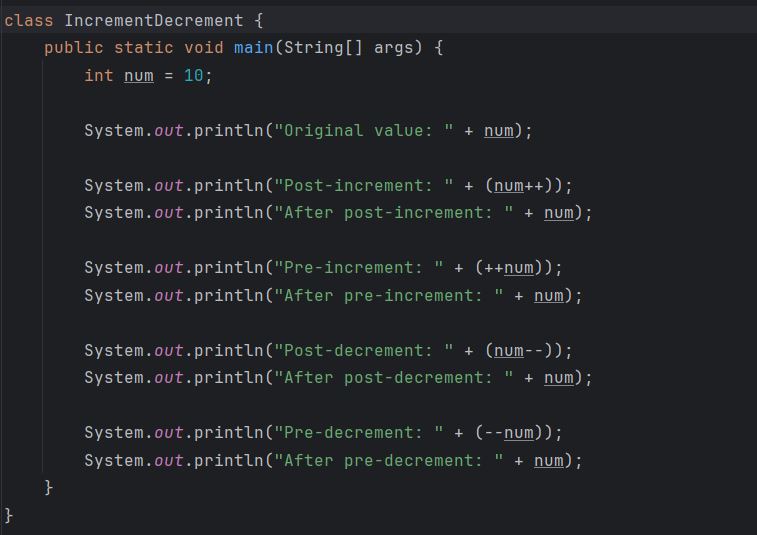
**Output:**

****

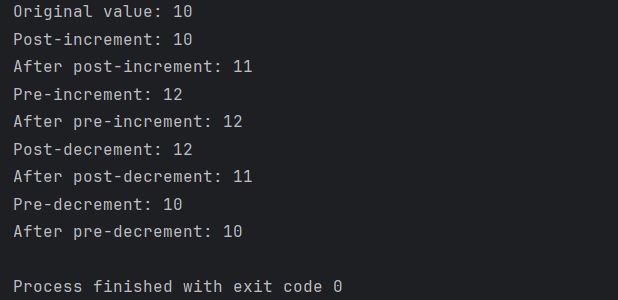
**Lab Task:** Demonstrate the use of pre-increment, post-increment, pre-decrement, and post-decrement operators.

**Steps:**

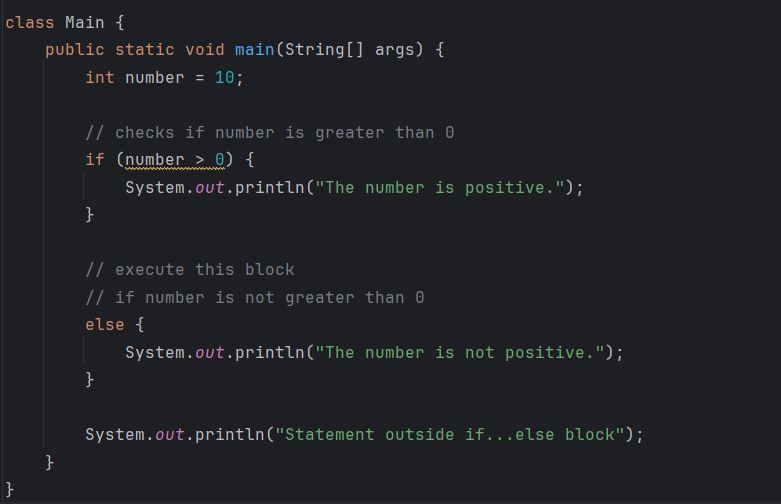
1. Declare an integer variable.
2. Use the increment (++) and decrement (--) operators in both pre and post forms.
3. Print the variable before and after each operation

****

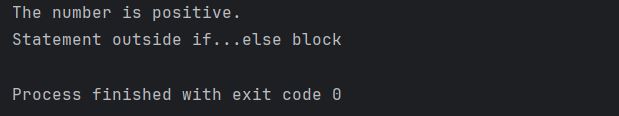
**Output:**

****

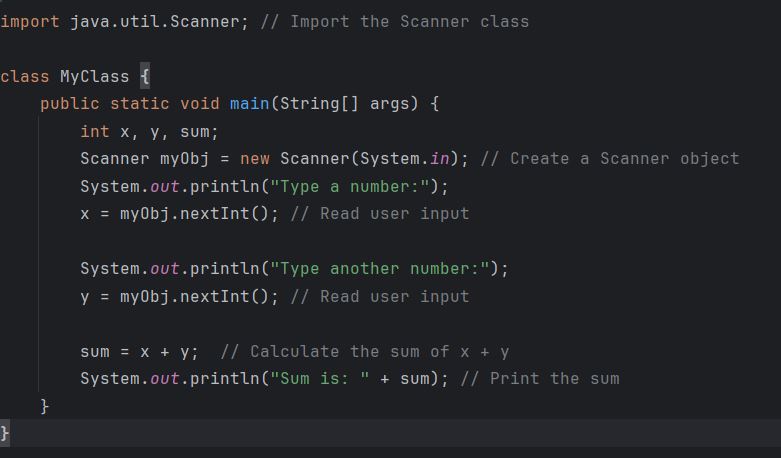
**Lab task:** Use of Control Structures in Java

****

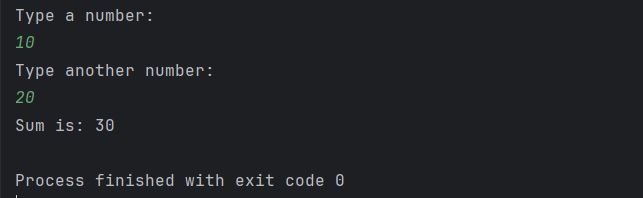
**Output:**

****

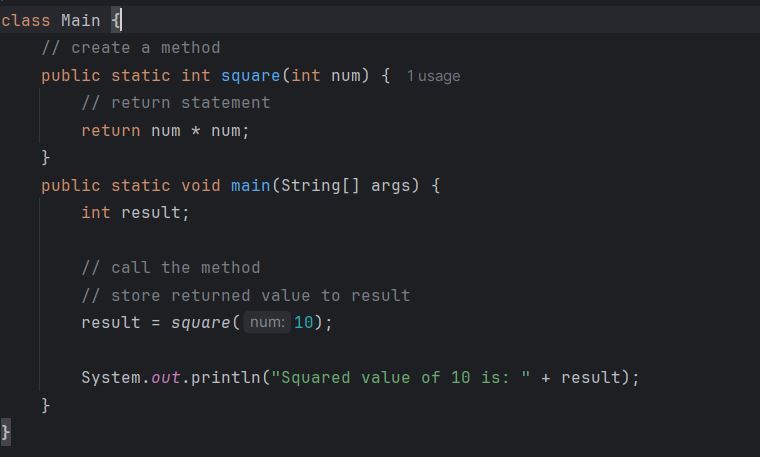
**Lab Task: A program that defines a method to calculate and return the sum of two numbers taken as an input from the user.**

****

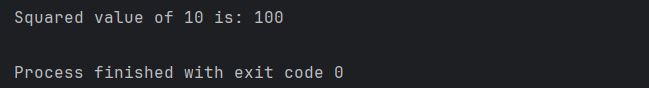
**Output:**

****

**Lab Task: Demonstrate the use of methods with return type**

****

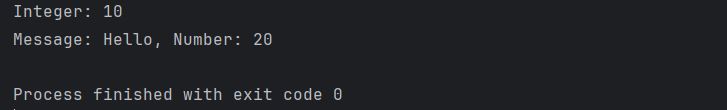
**Output:**

****

**Lab Task: A program that demonstrates method overloading by defining multiple methods with the same name but different parameters.**

****

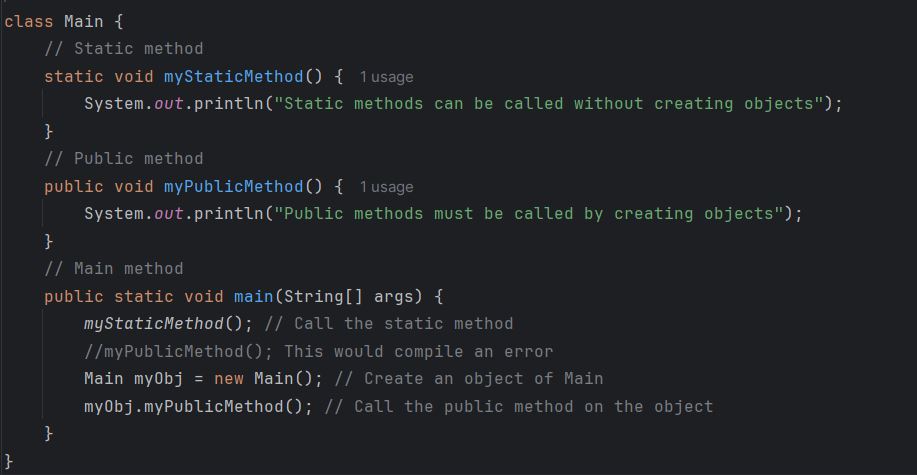
**Output:**

****

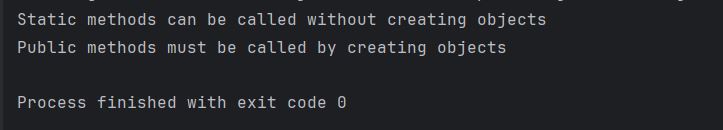
**Static Vs. Public Methods:**

Java programs can either have static or public attributes and methods.

In the example, a static method gets created, which means that it can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:



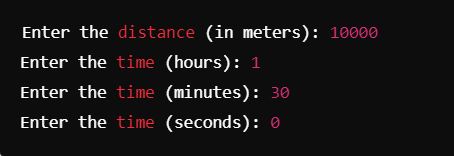
**Output:**



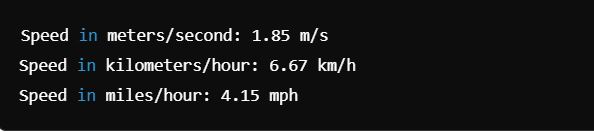
**Basic Practice tasks:**

1. Write a Java program that asks the user for a distance (in meters) and the time taken (as three numbers: hours, minutes, seconds), and display the speed, in meters per second, kilometers per hour and miles per hour (hint: 1 mile = 1609 meters).

**Sample Input:**

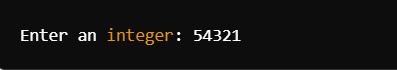


**Sample Output:**

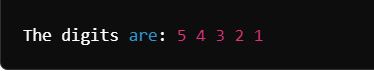


1. Write a Java program to break an integer into a sequence of digits.

**Sample Input:**

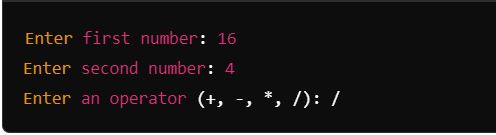


**Sample Output:**

****

1. Write a basic calculator that performs addition, subtraction, multiplication, and division.

**Sample Input:**

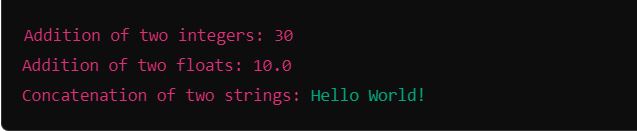
****

**Sample Output:**

****

1. Create a Java program with overloaded methods (e.g., to add two integers, two floats, and two strings)

**Sample Output:**



1. Write a program that defines a method with multiple return statements to check if a number is positive, negative, or zero. Take the number as an input from the user.

**Sample Input:**



**Sample Output:**



**Scenario Based Tasks**

**Task 1.1: Data Types and Variables**

Write a program that simulates turning a smart light on and off using variables to store its state.

**Define the following:**

* boolean to store whether the light is on or off.
* int to store the brightness level (0 to 100).

**Instructions:**

* Define a boolean variable isLightOn to represent whether the light is on (true) or off (false).
* Define an int variable brightnessLevel to store the brightness (0 when off, 100 at maximum brightness).
* Use System.out.println to output the current state of the light.

|  |
| --- |
| Sample output:  Light status: Off  Brightness: 0% |

**Task 1.2: Operators and Adjusting Brightness**

Use arithmetic operators to increase or decrease the brightness level. Let’s create a method to handle brightness control.

**Instructions:**

* Use the += and -= operators to modify the brightnessLevel.
* Write a method adjustBrightness that increases or decreases brightness based on a user-provided value.

|  |
| --- |
| Sample output:  Increased Brightness: 60%  Decreased Brightness: 40% |

**Part 2: Control Structures**

**Task 2.1: Using if-else for Light Control**

Implement logic that turns the light on or off using an if-else statement.

**Instructions:**

* Define a boolean variable isLightOn and initialize it to false.
* Use an if-else structure to simulate turning the light on or off based on user input.

|  |
| --- |
| Sample output:  Light is now ON |

**Task 2.2: Using switch-case for Light Modes**

Simulate different lighting modes (e.g., Normal, Reading, Night) using the switch-case statement.

**Instructions:**

* Define an int variable mode to represent the lighting mode (1 for Normal, 2 for Reading, 3 for Night).
* Use a switch-case to print the current lighting mode.

|  |
| --- |
| Sample output:  Normal mode |

**Part 3: Methods and Method Overloading**

**Task 3.1: Defining and Calling Methods**

Define a method to turn the light on or off. The method should take a boolean argument to control the light.

**Instructions:**

* Create a method toggleLight that takes a boolean parameter isLightOn.
* Print whether the light is on or off based on the input.

|  |
| --- |
| Sample output:  Light is ON  Light is OFF |

**Task 3.2: Method Overloading for Brightness Control**

Overload the adjustBrightness method to accept either an int for fixed adjustment or a double for a percentage-based adjustment.

Instructions:

* Create two overloaded methods adjustBrightness: one taking int, another taking double.
* Adjust brightness level based on the input type.

|  |
| --- |
| Sample output:  Adjusting brightness by 20 units.  Adjusting brightness by 15.5%. |

**Part 4: Static Members and Command-Line Arguments**

**Task 4.1: Using static Keyword**

Simulate a system where all lights share a global setting for energy-saving mode using a static variable.

**Instructions:**

* Define a static boolean energySavingMode in the Main class.
* Write a static method setEnergySavingMode to enable or disable energy-saving mode.

|  |
| --- |
| Sample output:  Energy-saving mode: Enabled  Energy-saving mode: Disabled |

**Task 4.2: Command-Line Arguments for Light Settings**

Use command-line arguments to set the initial state of the light and brightness level.

**Instructions:**

* Modify the main method to accept arguments: the first argument for the light state and the second for brightness.
* Parse the command-line arguments to set isLightOn and brightnessLevel.

|  |
| --- |
| Sample output:  java Main true 75 |