Object Oriented Programming (JAVA)



## Semester: Fall 2024

**Software Engineering**

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

***Lab Instructor: Danish Ali Khan***

|  |  |
| --- | --- |
| **Week 4** | |
| **Topic** | **Introduction to OOP: Classes and Objects, Constructors and Methods, Getters and Setters, Access Specifiers** |
| **Objective** | * Define a single class in Java with appropriate attributes and methods. * Create and manipulate instances (objects) of the class. * Use constructors, including constructor overloading, to initialize objects with default and specific values. * Implementing the concept of Encapsulation and using getters and setters * Understanding the access modifiers in Java * Implementing constructor overloading and method overloading * Creating UML class diagram |

**Understanding Classes and objects in Java:**

* Java is an object-oriented programming language.
* Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.
* A Class is like an object constructor, or a "blueprint" for creating objects.

**Creating a Class and creating an Object:**

To create a class, use the keyword Class:

For Eg:

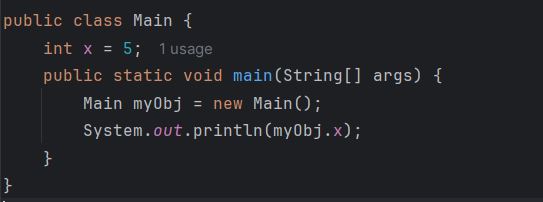
public class Main {

int x = 5;

}

In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects. To create an object of Main, specify the class name, followed by the object name, and use the keyword new:

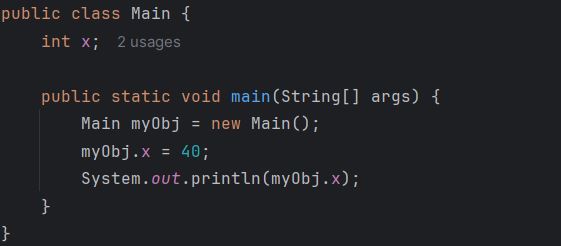
For Eg:



**Java Class Attributes**

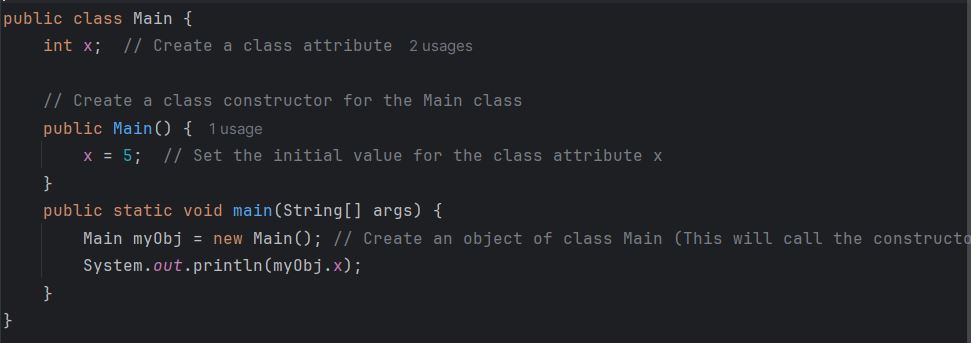
* The term "variable" for x used in the class, it is actually an attribute of the class. Or we can say that class attributes are variables within a class. Another term for class attributes is **fields**.
* We can access attributes by creating an object of the class, and by using the dot syntax (.).
* We can also modify the attribute values.

For Eg:



**Java Constructors:**

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:



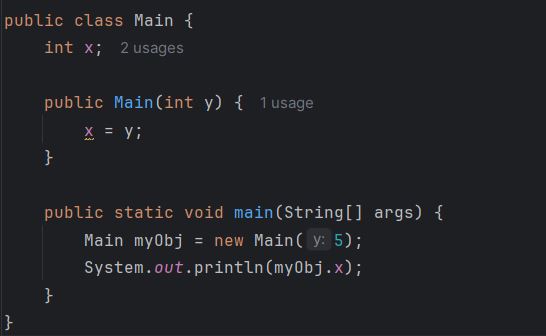
Note that the constructor’s name must **match the class name**, and it cannot have a **return type** (like void).

Also note that the constructor is called when the object is created.

All classes have constructors by default: if you do not create a class constructor yourself, Java creates one for you. However, then you are not able to set initial values for object attributes.

**Constructor Parameters:**

* Constructors can also take parameters, which is used to initialize attributes.
* The following example adds an int y parameter to the constructor. Inside the constructor we set x to y (x=y).
* When we call the constructor, we pass a parameter to the constructor (5), which will set the value of x to 5:



**This keyword:**

In Java, the **“this”** keyword is used in several situations to refer to the current object instance. It helps in distinguishing between instance variables and parameters, invoking constructors, and passing the current object as an argument.

**For e.g: this.length =>** 'this.length' refers to the instance variablewhile 'length' refers to the parameter.

**Encapsulation**

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

declare class variables/attributes as private

provide public get and set methods to access and update the value of a private variable

**Getter and Setter**

The private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public **get** and **set** methods.

The get method returns the variable value, and the set method sets the value.

Syntax for both is that they start with either get or set, followed by the name of the variable, with the first letter in upper case:

public class Person {

private String name; // private = restricted access

// Getter

public String getName() {

return name;

}

// Setter

public void setName(String newName) {

this.name = newName;

}

}

**Understanding the Modifiers in Java:**

The public keyword is an **access modifier**, meaning that it is used to set the access level for classes, attributes, methods and constructors.

We divide modifiers into two groups:

* **Access Modifiers** - controls the access level
* **Non-Access Modifiers** - do not control access level, but provides other functionality

**Types of access specifier:**

|  |  |
| --- | --- |
| Public | Specifies that the member can be accessed from any other class anywhere, whether within or in a different package. |
| Private | When a member (be it a field, method, or constructor) is declared private, it can only be accessed within the same class. |
| Protected | Members declared as protected can be accessed within the same package or in subclasses in different packages. |
| Default | When no access modifier is specified, Java uses a default access level, often called package-private. This means the member is accessible only within classes in the same package. |

myMethod() prints a text (the action), when it is **called**. To call a method, write the method's name followed by two parentheses **()** and a semicolon**;**

public class Main {

static void myMethod() {

System.out.println("Hello World!");

}

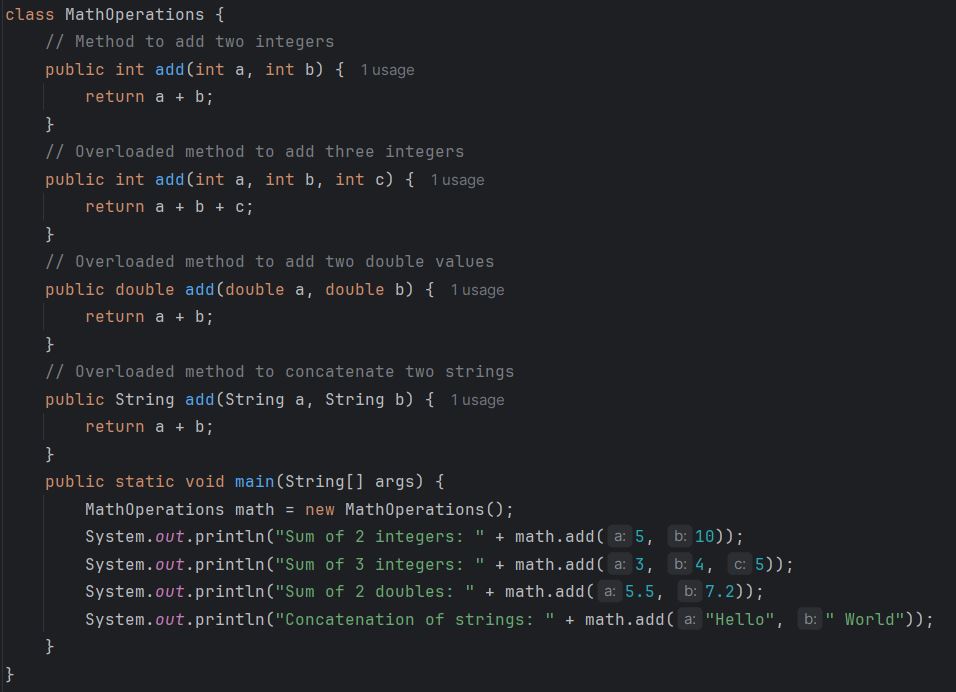
public static void main(String[] args) {

myMethod();

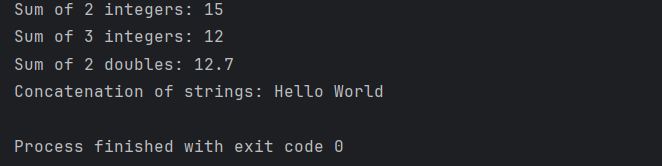
}

}

**Method Overloading:** Method overloading in Java allows a class to have more than one method with the same name, but different parameter lists (type, number, or both). Java determines which method to call based on the method signature, which includes the method name and the parameter list.

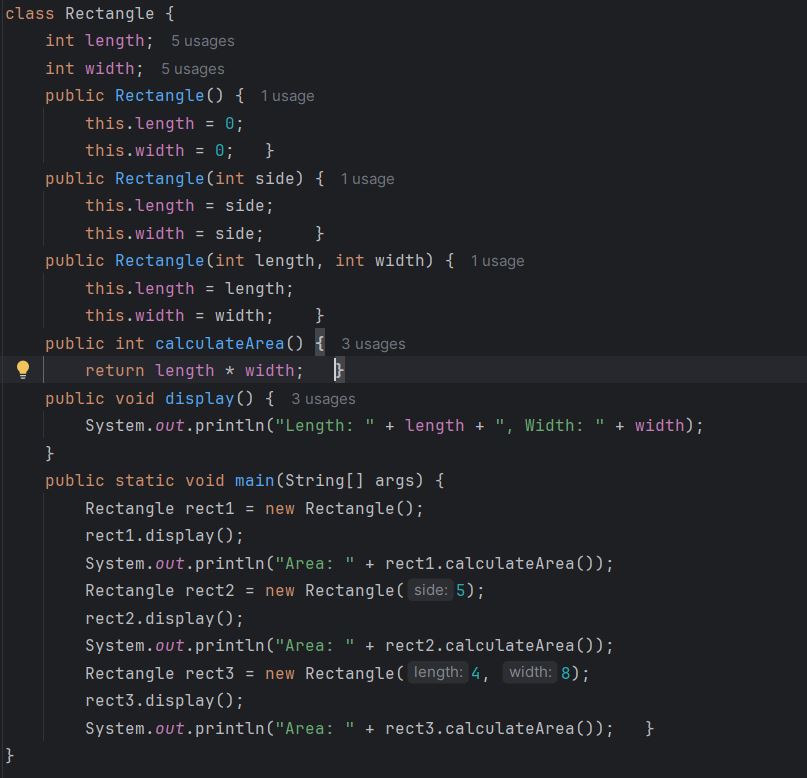
****

**Output:**

****

**Constructor Overloading:**

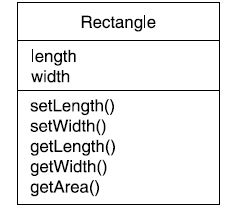
Constructor [overloading in Java](https://www.javatpoint.com/method-overloading-in-java) is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.



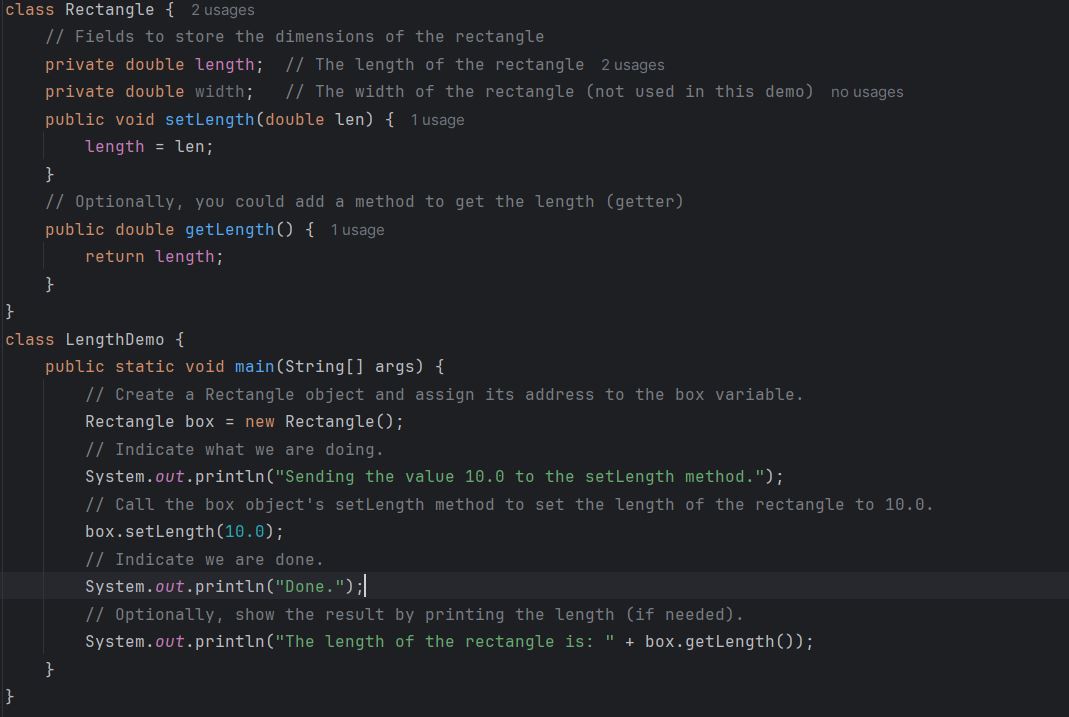
**Output:**

****

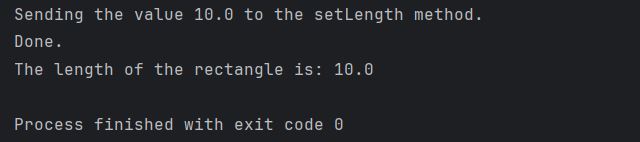
**Understanding UML diagram:**



**The code for the above diagram is shown below:**



**Output:**



**Basic Tasks:**

Create an employee class and perform the following tasks

* Create attributes called name, id, and salary.
* Implement multiple constructors to initialize the object.
* Overload methods to calculate the employee's yearly salary with or without bonuses.
* Initialize an array of string representing skills.

**Sample Output:**



**Scenario Based Tasks**

**Task 1: Define the SmartLight Class**

**Objective**: Create a class that represents a smart light.

**Instructions**:

In a single Java file (SmartHomeSystem.java), define a class named SmartLight with the following attributes:

* String lightName (e.g., "Living Room Light")
* boolean isOn (initially false)
* int brightness (initially 0, range from 0 to 100)

Add methods to the SmartLight class:

* void turnOn(): Turns the light on and sets isOn to true.
* void turnOff(): Turns the light off and sets isOn to false.
* void setBrightness(int level): Sets the brightness level, ensuring it is within the range of 0 to 100.
* void displayStatus(): Prints the current status of the light (on/off) and its brightness.

**Expected Output**: The methods should control the light’s status and brightness, and displayStatus() should reflect these changes.

**Task 2: Create and Use Instances of SmartLight**

**Objective**: Practice creating instances of the class and using its methods.

**Instructions**:

In the main method of the SmartHomeSystem class:

* Create an instance of the SmartLight class named livingRoomLight.
* Call the turnOn() method to turn the light on.
* Set the brightness to 75 using setBrightness().
* Display the status using displayStatus().
* Turn the light off using turnOff() and display the status again.

**Expected Output**: The console should display the status of the livingRoomLight before and after each operation.

**Task 3: Implement Constructors and Constructor Overloading**

**Objective**: Use constructors and constructor overloading to initialize objects with default and specific values.

**Instructions**: Define three constructors in the SmartLight class:

* A default constructor that initializes lightName to "Default Light", isOn to false, and brightness to 0.
* A parameterized constructor that initializes lightName and sets default values for isOn and brightness.
* An overloaded constructor that initializes all three attributes: lightName, isOn, and brightness.

In the main method, create three objects:

* One using the default constructor.
* One using the second constructor to initialize lightName only.
* One using the overloaded constructor to initialize all attributes.

Display their statuses using displayStatus().

**Expected Output**: The output should show three different lights with different initial values, demonstrating the use of constructor overloading.

**Task 4: Encapsulation**   
**Objective:** Create a class to represent a smart light with encapsulation.

**Instructions:**

1. In a Java file named AdvancedSmartLightSystem.java, define a package named smartlightcontroller.
2. Create a class named SmartLight within the package with the following private attributes:

* String lightName (e.g., "Living Room Light")
* boolean isOn (initially false)
* int brightness (initially 0, range from 0 to 100)
* String color (e.g., "White", initially "White")

1. Add public getter and setter methods for each attribute.
2. Implement the following methods in the SmartLight class:
   * void turnOn(): Turns the light on and sets isOn to true.
   * void turnOff(): Turns the light off and sets isOn to false.
   * void setBrightness(int level): Sets the brightness level, ensuring it is within the range of 0 to 100.
   * void setColor(String color): Sets the color of the light.
   * void displayStatus(): Prints the current status of the light (on/off), its brightness, and color.

**Expected Output:** The methods should manage the light’s status, brightness, and color.

**Task 5: Method Overloading**

**Objective**: Implement method overloading within a single class.

**Instructions**: Overload the setBrightness() method to allow users to set the brightness level using either:

* An int (as before), or
* A String (e.g., "low", "medium", "high").

Implement the overloaded setBrightness(String level) method to interpret the string values as specific brightness levels:

* "low" = 25
* "medium" = 50
* "high" = 75

In the main method, test both versions of the setBrightness() method.

**Expected Output**: The program should correctly interpret both numeric and string inputs for brightness levels.

**Task 6: Create a Simple Class Diagram**

**Objective**: Reinforce understanding by creating a visual representation.

**Instructions**: Draw a simple UML class diagram for the SmartLight class. Include all attributes, methods, and constructors in the diagram.

**Expected Output (Sample):**

Living Room Light is now ON.

Brightness set to 75

Living Room Light is ON with brightness level 75

Living Room Light is now OFF.

Living Room Light is OFF with brightness level 75

Default Light is OFF with brightness level 0

Kitchen Light is OFF with brightness level 0

Bedroom Light is ON with brightness level 50

Brightness set to 50 based on level: medium

Living Room Light is ON with brightness level