Classes & Objects in Java

Introduction

- **Class** is a blueprint or template for creating objects.
- **Object** is a real instance created from a class.

Example from real life:

- **Class:** Car (defines features like color, model, speed)
- **Object:** Your Honda City or Maruti Swift — an actual car you own.

Object’s State and Behavior in Java

- Every object has:
  - **State** → what it “has” //fields
  - **Behavior** → what it “does” //methods
- Example

  "A student has a name, roll number, and marks — these are the ***state****. The student can study and give exams — these are the* ***behaviors****."*

Key Points

- **Class:** Defines attributes (data) & methods (behavior)
- **Object:** Instance of class with its own data
- Objects access members using **dot (.) operator**
- Multiple objects can be created from one class

Real-Life Analogy

"Think of a class like a recipe for making cake. The recipe (class) tells you what ingredients and steps are required. But the actual cake (object) is made when you follow that recipe!"

Syntax to Create a Class in Java

```
class ClassName {
    // Fields (State)
    dataType variableName;

    // Methods (Behavior)
    returnType methodName() {
        // method body
    }
}
```

Syntax to Create an Object of a Class

```
ClassName objectName = new ClassName();
```

```
class Student {
    String name;
    int rollNo;

    void study() {
        System.out.println(name + " is studying");
    }
}
```

Complete Example for Class + Object

```
class Student {
    String name;     // Field (State)
    int rollNo;      // Field (State)

    void study() {   // Method (Behavior)
        System.out.println(name + " is
studying");
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();  //
Object creation
        s1.name = "Rahul";           // Accessing
state
        s1.rollNo = 101;
        s1.study();                  // Calling
behavior
    }
}
```

Access Control for **Inner/Nested Classes** (Classes inside another class)

Modifier	**Access Scope**
public	Accessible everywhere
protected	Accessible within the same package and subclasses
default	Accessible within the same package
private	Accessible only within the enclosing class

Fundamentals of Object-Oriented Programming (OOP)

1. Inheritance

Meaning: Inheritance allows a class to acquire (inherit) properties and behaviors of another class.

Why Use It: To reuse existing code and extend features.

Example: Child class inherits from Parent class.

2. Polymorphism

- **Meaning:**
  Polymorphism means **one name, many forms**.
  The same method behaves differently based on object or data.
- **Types:**
  - **Compile-Time (Method Overloading)**
  - **Run-Time (Method Overriding)**

3. Abstraction

- **Meaning:**
  Showing only essential details and hiding the complex implementation.
- **Why Use It:**
  To reduce complexity and increase security.
- **Achieved by:**
  - **Abstract Classes**
  - **Interfaces**

Real-Life Example

“Abstraction is like using a mobile phone.

You just tap on the screen to make a call or send a message.

You don’t know — and don’t need to know — how the circuit board, signal processing, or software code works inside.

You use the exposed interface (buttons/features),but the actual working is hidden inside the phone’s hardware and software system.”

4. Encapsulation

- **Meaning:**
  Wrapping data (variables) and methods into a single unit (class) and restricting direct access.
- **Why Use It:**
  To protect data from unauthorized access.
- **Achieved by:**
  - Using **private variables**
  - Providing **public getters & setters**

Real life example

“Encapsulation is like a bank account.
The money (data) is inside, you can’t directly take it.
You have to go through the cashier (methods) with proper verification.”

Dot Operator (.) in Java

What is Method Overloading in Java?

- **Method Overloading** means defining **multiple methods with the same name** in the same class,
  but with **different parameters (number, type, or order).**

Key Points:

- Same method name.
- Different parameter **types**, **order**, or **count**.
- Decided at **compile time** — known as **compile-time polymorphism**.
- Allows the same action to be performed in different ways.

Syntax Example:

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {  // Overloaded with different type
        return a + b;
    }

    int add(int a, int b, int c) {  // Overloaded with different number of parameters
        return a + b + c;
    }
}
```

Constructors

What is a Constructor in Java?

A **Constructor** is a special method in a class that **initializes objects** when they are created.

Key Points:

- Has the same name as the class.
- Does NOT have a return type — not even void.
- Called automatically when an object is created.
- Used to set initial values to object’s fields.

Syntax Example:

```
class Student {
    String name;

    // Constructor
    Student() {
        name = "Default Student";
    }
}

Student s = new Student();  // Constructor called automatically
```

Types of Constructors:

Type	Meaning	Example
Default Constructor	Provided by Java if no constructor is written	Student() {}
No-Argument Constructor	Written by you, takes no parameters	Student() { }
Parameterized Constructor	Takes parameters to set initial values	Student(String name) { this.name = name; }
Copy Constructor *(Not built-in like C++)*	You can manually create one to copy objects	Student(Student s) { this.name = s.name; }

Example of Parameterized Constructor:

```
class Student {
    String name;
    int age;

    Student(String n, int a) {  // Parameterized Constructor
        name = n;
        age = a;
    }
}
```