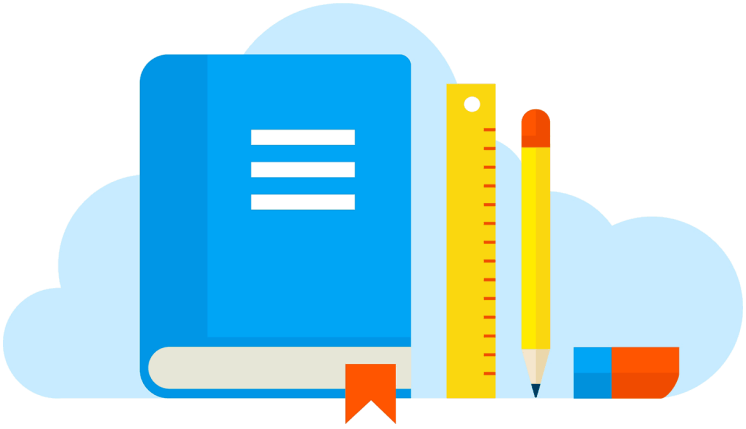


Java Programming

Date: 08/02/2025

What Is JAVA?

- Java is a high-level, object-oriented programming language.
- Designed for platform independence: *“Write Once, Run Anywhere”*
- Key features of Java programming are:
  - Simple and easy to learn
  - Secure and robust.
  - Portable and platform-independent.
  - Multi-threaded and dynamic

Applications of Java:

- Android app development.
- Web applications.
- Enterprise softwares.
- Embedded systems.

Java Development Environment:

- The Java development environment consists of the following components.
    - JDK (Java Development Kit): Contains tools for development.
    - JRE (Java Runtime Environment): Enables execution of Java programs.
    - JVM (Java Virtual Machine): Converts bytecode to machine code.
    - JIT (Just-In-Time Compiler): performance optimization of java based applications during run time

Objects:

- Object has two properties, which are:
    - Properties or State
    - Behaviour or Function

  Example,

    - Dog is an object, because it has:

      i. Properties like bread, color, age etc.
      ii. Behaviours like bark, sleep, run, eat etc.

    - Car is an object, because it has:

      i. Properties like brand, color, weight etc.
      ii. Behaviours like drive, speed, brake etc.

Classes:

- Class provides the Template or Blueprint, from which Objects can be created.
- To create an Object, a Class is required.
- From one Class, we can create multiple Objects.
- To create a Class, use the keyword class.

Basic Syntax of Java Program:

Example Program:

```
public class HelloWorld {              // Basic Syntax of Java program
        int a = 10;
        int b = 20;
        char c = 'd';
  public static void main( String[] args ) {
        System.out.println("Hello, World");
        }
}

HelloWorld hello = new HelloWorld();      // creating an Object using class HelloWorld
```

Explanation:

- public class HelloWorld- declares a Class named HelloWorld.
- public static void main (String[] args)- main class; which is entry point of the program
- System.out.println("Hello, World")- prints the output to the console.
- An object is created with keyword 'new', object name is 'hello'.

1st Pillar of OOPS - DATA ABSTRACTION:

Definition:

- It hides the internal implementation and shows only essential functionality to the user.
- It can be achieved through **Interface** and **Abstract** classes.

Example:

- A car only shows the brake pedal to stop the car, but the inner functionality is abstracted to us.
- A remote control can operate a device, but how it works is abstracted to us.

Advantages:

- Increases security and confidentiality.

Code Example for Interface and Abstract Class:

```
interface Vehicle {                    // Java program demonstrating Interface
        void start();
        }
class Car implements Vehicle {
        public void start() {
            System.out.println("Car is starting");
                }
        }
-----------------------------------------------------

abstract class Animal {                // Java program demonstrating Abstraction
        abstract void makeSound();
        }
class Cow extends Animal {
        void makeSound() {
            System.out.println("Cow moos.");
                }
        }
```

ABSTRACT Class in Java:

Definition:

- An abstract class is a class that cannot be instantiated directly. It serves as a blueprint for other classes to derive from.
- An abstract class can contain both abstract methods (methods without an implementation) and concrete methods (methods with an implementation).
- Abstract classes can have member variables, including final, non-final, static, and non-static variables.
- Abstract classes can have constructors, which can be used to initialize variables in the abstract class when it is instantiated by a subclass.

INTERFACE Class in Java:

Definition:

- An interface is a reference type in Java, it is similar to a class, and it is a collection of abstract methods and static constants.
- All methods in an interface are by default abstract and must be implemented by any class that implements the interface.From Java 8, interfaces can have default and static methods with concrete implementations.From Java 9, interfaces can also have private methods.
- Variables declared in an interface are by default public, static, and final (constants).

2nd Pillar of OOPS - DATA ENCAPSULATION:

Definition:

- Encapsulation binds the data and methods together, while restricting access to some components.
- Encapsulation bundles the data and the code working on that data as a single unit, also known as Data Hiding.

Steps to achieve encapsulation:

- Declare a Variable of a Class as Private.
- Provide a Public Getters and Setters to modify and view the value of the variable.

Example code for data encapsulation:

```
class Person {                          // Java program demonstrating Encapsulation
        private String name;            // Getter and Setter for name
        public String getName() { return name; }
        public void setName(String name) { this.name = name; }
        }
        public class human {
                public static void main(String[] args) {
                        Person person = new Person();
                        person.setName("John Doe");
                        System.out.println("Name is: " + person.getName());
                        }
                }
```

Key Concepts:

- Access Modifiers: private, public, protected;
- Methods: Getters and Setters.

Advantages:

- Loosely coupled code;
- Better access control and security.

3rd Pillar of OOPS - DATA INHERITANCE:

Definition:

- Inheritance allows a class (**child**) to acquire the properties and behaviours of another class (**parent**).
- It can inherit both functions and variables, so we do not have to write them again in child classes.

Key Concepts:

- Keyword **extends**: Used to inherit a class.
- Superclass (Parent class) and Subclass (Child class)

Advantages:

- Code Reusability and Method overriding.
- We can achieve Polymorphism using Inheritance.

Example code for Inheritance:

```
class Employee {                    // Java Program to illustrate Inheritance
   int salary = 60000;              // Base or Super Class
        }

class Engineer extends Employee {   // Inherited or Subclass
   int benefits = 10000;
        }

class Payable {                                 // Driver Class
   public static void main(String args[])
        {
      Engineer E1 = new Engineer();
      System.out.println("Salary : " + E1.salary);
      System.out.println("Benefits : " + E1.benefits);
        }
}
```

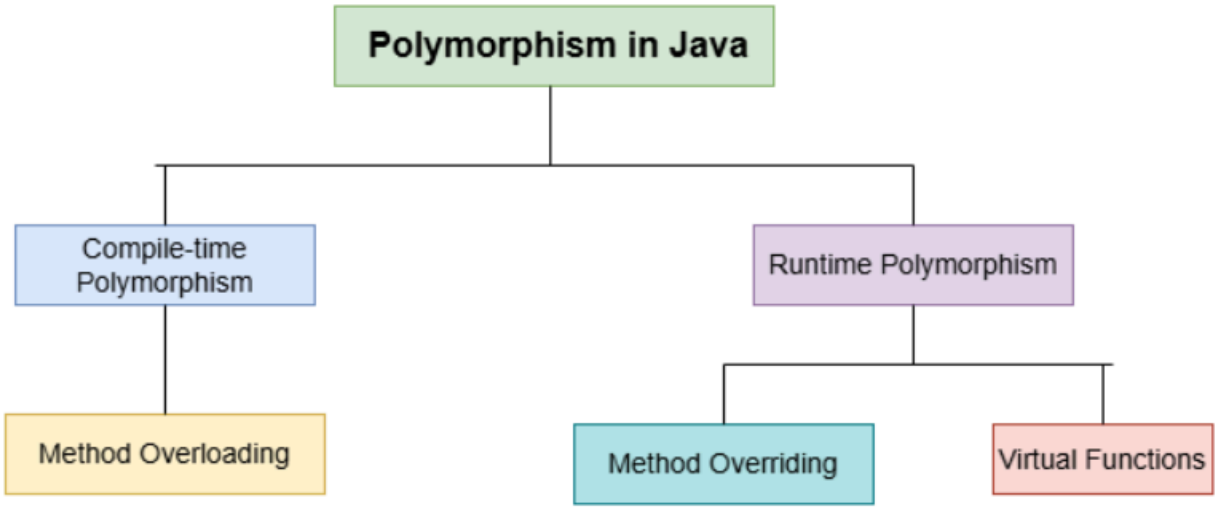
4th Pillar of OOPS - POLYMORPHISM:

Definition:

- Polymorphism refers to the ability of a message to be displayed in more than one form.
- it allows objects to behave differently based on their specific class type.

Types of Polymorphism:

- Compile-Time Polymorphism
- Runtime Polymorphism



Key Concepts:

Compile-Time Polymorphism in Java is also known as static polymorphism. This type of polymorphism is achieved by function overloading or operator overloading.

Method overloading in Java means when there are multiple functions with the same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by changes in the number of arguments or/and a change in the type of arguments.

Runtime Polymorphism in Java known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.

Method overriding, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden

Virtual Functions allows an object of a derived class to behave as if it were an object of the base class. The derived class can override the virtual function of the base class to provide its own implementation. The function call is resolved at runtime, depending on the actual type of the object.

Example code for Compile-Time Polymorphism:

```
// Method overloading by using different types of arguments
class Calculator {
        int add(int a, int b) { // Method 1 with 2 integer parameters
        return a + b;
                }
        double add(double a, double b) {  // Method 2 with 2 double parameters
        return a + b;
                }
        }
class Output {                          // Main driver method
public static void main(String[] args) {
    // Calling method by passing inputs as in arguments
    System.out.println(Calculator.add(2, 4));
    System.out.println(Calculator.add(5.5, 6.3));
                }
      }
```

Example code for Runtime Polymorphism:

```
// Java Program for Method Overriding
class Animal {
        void sound() {
        System.out.println("This animal makes a sound.");
                }
        }
class Cat extends Animal {
        void sound() {
        System.out.println("The cat meows.");
                }
        }
public class Main {
        public static void main(String[] args) {
        Animal myAnimal = new Cat(); // Upcasting
        myAnimal.sound();
                }
        }
```

Relationship in Java:

A relationship in Java means different relations between two or more classes. For example, if a class Bulb inherits another class Device, then we can say that Bulb is having is-a relationship with Device, which implies Bulb is a device.

In Java, we have two types of relationship:

- **Is-A relationship:** Whenever one class inherits another class, it is called an IS-A relationship.
- **Has-A relationship:** Whenever an instance of one class is used in another class, it is called a HAS-A relationship.

IS-A Relationship is wholly related to Inheritance. For example – a kiwi is a fruit; a bulb is a device.

- IS-A relationship can simply be achieved by using extends Keyword.
- IS-A relationship is additionally used for code reusability in Java and to avoid code redundancy.
- IS-A relationship is unidirectional, which means we can say that a bulb is a device, but vice versa; a device is a bulb is not possible since all the devices are not bulbs.
- IS-A relationship is tightly coupled, which means changing one entity will affect another entity.

Has-A Relationship In Java, a Has-A relationship essentially implies that an example of one class has a reference to an occasion of another class or another occurrence of a similar class. For instance, a vehicle has a motor, a canine has a tail, etc.

- **Association** is the relation between two separate classes which establishes through their Objects. Composition and Aggregation are the two forms of association.
- In **Aggregation**, both the entries can survive individually which means ending one entity will not affect the other entity.
- **Composition** is a stronger form of aggregation that means ownership and lifecycle dependent; if the whole gets destroyed, then the parts no longer exist.