**🔶 What is OOP?**

**Object-Oriented Programming** (OOP) is a programming paradigm based on the concept of **objects**—entities that encapsulate **state (data)** and **behavior (methods)**.

**📘 Four Core OOP Principles in Java**

**1. Encapsulation**

**Definition**: Bundling data (fields) and behavior (methods) inside a class, and restricting direct access to some of the object's components.

**✅ Benefits:**

* Improves **data security**
* Reduces **code complexity**
* Supports **maintenance and testing**

**💡 Example:**

java

CopyEdit

public class Customer {

private String customerId;

private String name;

public String getCustomerId() {

return customerId;

}

public void setCustomerId(String id) {

this.customerId = id;

}

}

👉 customerId is private — accessible only through getters/setters.

**2. Abstraction**

**Definition**: Hiding the internal implementation and showing only relevant details to the user.

**✅ Benefits:**

* Reduces **complexity**
* Focuses on **what** instead of **how**

**💡 Example:**

java

CopyEdit

public abstract class Billing {

abstract double calculateBill();

}

java

CopyEdit

public class PrepaidBilling extends Billing {

@Override

double calculateBill() {

return 150.0;

}

}

👉 Users only need to call calculateBill()—not worry about implementation.

**3. Inheritance**

**Definition**: One class inherits fields and methods from another class.

**✅ Benefits:**

* Promotes **code reuse**
* Supports **hierarchical classifications**

**💡 Example:**

java

CopyEdit

public class User {

String name;

String mobile;

}

public class PostpaidUser extends User {

double monthlyCharge;

}

👉 PostpaidUser inherits name and mobile from User.

**4. Polymorphism**

**Definition**: One interface, many implementations. Polymorphism allows methods to behave differently based on the object that invokes them.

**✅ Types:**

* **Compile-time** (method overloading)
* **Runtime** (method overriding)

**🔸 Method Overloading (Same method name, different parameters)**

java

CopyEdit

public class Calculator {

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

double add(double a, double b) { return a + b; }

}

**🔸 Method Overriding (Subclass redefines parent method)**

java

CopyEdit

public class Billing {

double calculate() { return 0.0; }

}

public class PrepaidBilling extends Billing {

@Override

double calculate() { return 199.99; }

}

**🔧 Additional OOP Concepts**

**🔹 Class & Object**

* **Class**: Blueprint
* **Object**: Instance of class

java

CopyEdit

Customer c1 = new Customer();

**🔹 Constructor**

* Initializes an object during creation

java

CopyEdit

public Customer(String id, String name) {

this.customerId = id;

this.name = name;

}

**🔹 ‘this’ Keyword**

* Refers to the current object

java

CopyEdit

this.name = name;

**🔹 ‘super’ Keyword**

* Refers to the parent class

java

CopyEdit

super.calculate();

**🔹 Interfaces and Abstract Classes**

Used for defining **contracts** (APIs or Services).

**Example: VASService Interface**

java

CopyEdit

public interface VASService {

void activate(String customerId);

void deactivate(String customerId);

}

java

CopyEdit

public class CallerTune implements VASService {

public void activate(String id) {

System.out.println("Caller tune activated for " + id);

}

public void deactivate(String id) {

System.out.println("Caller tune deactivated for " + id);

}

}

**🔹 Composition (HAS-A)**

Using objects of one class inside another class.

java

CopyEdit

public class SIMCard {

private Customer customer; // HAS-A relationship

}

**🔹 Aggregation vs. Composition**

* **Aggregation**: Has-a (weak relationship)
* **Composition**: Strong containment (dependent lifecycle)

**✅ Why OOP in Real Projects (like Telecom Billing)?**

* Easier **code maintenance**
* Scalable architecture with **inheritance and interfaces**
* Secure billing data through **encapsulation**
* Extend VAS features using **polymorphism**

**🔁 Summary Table**

| **OOP Principle** | **Key Idea** | **Java Keyword/Feature** |
| --- | --- | --- |
| Encapsulation | Data hiding | private, get/set |
| Abstraction | Hide details | abstract, interface |
| Inheritance | Reuse behavior | extends, super |
| Polymorphism | Many forms of method | overloading, @Override |