Object-Oriented Programming (OOP) Report

Author: Ammar Hasan

---

1. Introduction

This report is just a simple, straightforward explanation of the OOP concepts I went through what they mean, why they're useful, and how they actually help in organizing a messy codebase. My goal was to understand these ideas in a practical way and apply them directly to the project.

2. The Four Main Pillars of OOP The Four Main Pillars of OOP

2.1 Encapsulation

Encapsulation is about **hiding internal details** and exposing only what is necessary. It helps protect data and prevents other parts of the program from directly messing with an object's state.

Example:

```
class BankAccount {
    private double balance; // only this class can access it

    public BankAccount(double balance) {
        this.balance = balance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public double getBalance() {
        return balance;
    }
}
```

This ensures no one can randomly do `balance = -100`. All updates must go through proper methods.

2.2 Inheritance

Inheritance lets one class reuse another class's properties. It helps avoid repeated code and provides a cleaner relationship structure.

```java
class Vehicle {
    void start() {
        System.out.println("Vehicle starting...");
    }
}

class Car extends Vehicle {
    void playMusic() {
        System.out.println("Playing music...");
    }
}
```

Now `Car` automatically gets `start()` without rewriting it.

2.3 Polymorphism

Polymorphism simply means "one interface, many behaviors." The method name stays the same, but the actual behavior changes depending on the object.

```java
class Animal {
    void sound() { System.out.println("Animal sound"); }
}

class Dog extends Animal {
    void sound() { System.out.println("Dog barks"); }
}
```

If I call `sound()` on a Dog, it barks. On a Cat, it meows. Same method, different outcome.

2.4 Abstraction

Abstraction focuses on showing **only the essential features**. It's like using a phone—we don't need to know how the circuit works; we just press buttons.

```java
abstract class Payment {
    abstract void makePayment();
}

class UPIPayment extends Payment {
    void makePayment() {
        System.out.println("Paid using UPI");
    }
}
```

The complex logic stays hidden inside the class.

---

3. Other Useful OOP Concepts

3.1 Constructor Overloading

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

    Student(String name) { this.name = name; }
    Student(String name, int age) { this.name = name; this.age = age; }
}
```

Gives flexibility in how objects are created.

3.2 Method Overloading

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

Same method name, different input types.

3.3 Interfaces

Interfaces allow multiple behavior patterns without worrying about class hierarchy.

```
interface Engine {
    void start();
}

class Car implements Engine {
    public void start() {
        System.out.println("Engine starts");
    }
}
```

Useful when your design needs standard behavior that many classes will follow.

---

4. How OOP Helps in Refactoring a Messy Codebase

In the project I joined, the code had:

- Repeated logic everywhere
- One giant class doing everything
- Payment logic written inside random methods
- No separation of responsibilities

Here's an example of how the old code looked:

Before Refactoring (Hard to Maintain)

```
class OrderProcessor {
    public void process(String type) {
        if ("COD".equals(type)) {
            System.out.println("Processing COD");
        } else if ("UPI".equals(type)) {
            System.out.println("Processing UPI");
        }
    }
}
```

Adding a new payment method = editing this file again (not scalable).

After Refactoring (Clean and Extendable)

```
interface Payment {
    void pay();
}

class CODPayment implements Payment {
    public void pay() {
        System.out.println("Processed COD");
    }
}

class UPIPayment implements Payment {
    public void pay() {
        System.out.println("Processed UPI");
    }
}
```

```
class OrderProcessor {
    public void process(Payment payment) {
        payment.pay();
    }
}
```

Now, adding a new payment type means just creating a new class. No touching old code—less risk, fewer bugs.

---

5. Why OOP Makes Projects Easier to Manage

- Code becomes easier to read
- Components become reusable
- Bugs reduce because each class has a clear responsibility
- Testing becomes smoother
- Adding new features becomes safe

In short, OOP makes the entire project stable and organized.

---

6. Conclusion

Studying these OOP concepts helped me understand how to approach a messy project. By applying encapsulation, abstraction, inheritance, and polymorphism, I was able to break down large classes, structure the logic properly, and make the system easier to modify and extend.

This report reflects the understanding I gained and how I plan to use OOP principles to improve our existing codebase.