Assignment-1

Student Name: Charu Singla **UID: 23BCS12553**
Branch: CSE **Section/Group: KRG 3-A**
Semester: 6th **Subject Code: 23CSH-314**
Subject Name: System Design

Q1: Explain the role of Interfaces and Enums in software design with proper examples?

Solution: Interfaces and Enums are powerful tools that help build clean, scalable, and maintainable systems.

1. **Interfaces** - Defining What a Class Must Do

- **What is an Interface?**
  An interface is a blueprint that defines method signatures only (no implementation).
  It tells what actions are required, not how they are performed.

- **Role of Interfaces:**
  a. **Loose Coupling**
     High-level modules depend on interfaces, not concrete classes.
  b. **Multiple Implementations**
     Different classes can implement the same interface in their own way.
  c. **Supports Polymorphism**
  d. **Improves Testability**
     Easy to mock interfaces during unit testing.
  e. **Follows SOLID Principles** (especially Dependency Inversion)

- **Example CODE:**

```
// INTERFACE
interface Vehicle {
   void start();
}

// IMPLEMENTATION
class Car implements Vehicle {

   public void start() {
      System.out.println("Car started");
   }
}
```

```
// main
public class InterfaceDemo {
    public static void main(String[] args) {

        Vehicle v = new Car();  // loose coupling
        v.start();
    }
     }
```

2. Enums-Representing Fixed Set of Values

An enum (enumeration) represents a fixed group of constants.

- **What is an Enum?:**
  An enum is a data type that represents a group of constant values, providing type safety, better readability, and centralized control over fixed options.

- **Role of Enums:**
  a. Prevents Invalid Values
  b. Improves Code Readability
  c. Centralized Constants
  d. Type Safety
  e. Better Maintainability

- **Example Code:**

```
enum OrderStatus {
    PLACED,
    SHIPPED,
    DELIVERED
}
public class EnumDemo {

    public static void main(String[] args) {

        OrderStatus status = OrderStatus.SHIPPED;
        System.out.println("Order Status: " + status);
    }
     }
```

Q2:Discuss how interfaces enable loose coupling with example.

Solution: Loose coupling means that one part of a system depends on abstractions (interfaces) rather than concrete implementations.

This allows components to change independently without affecting the whole system.

- **Role of Interfaces in Loose Coupling:**
  a. Separating what a class does from how it does it
  b. Allowing multiple implementations of the same behavior
  c. Making code flexible and extensible
  d. Improving testability (easy to mock interfaces)
  e. Reducing dependency between modules

- **Create Interface:**

```
interface MessageService {
    void sendMessage(String message);
}
```

- **Implement Interface:**

```
class EmailService implements MessageService {
    public void sendMessage(String message) {
        System.out.println("Email sent: " + message);
    }
}

class SmsService implements MessageService {
    public void sendMessage(String message) {
        System.out.println("SMS sent: " + message);
    }
}
```

- **Use Interface (Loose Coupling):**

```
public class Notification {

    private MessageService service;

    // Constructor Injection
    public Notification(MessageService service) {
        this.service = service;
    }

    public void notifyUser(String msg) {
        service.sendMessage(msg);
    }

    public static void main(String[] args) {

        MessageService service = new EmailService(); // can change to SmsService
        Notification notification = new Notification(service);

        notification.notifyUser("Hello  User!");
    }
     }
```