DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

ASSIGNMENT-1

Student Name: Chirag Pandit **UID: 23ICS10001**
Branch: BE- CSE **Section/Group: 23BCS-KRG_3-A**
Semester: 6th **Date: 03/02/2026**
Subject Name: System Design **Subject Code: 23CSH-307**

Q1. Explain the role of Interfaces and Enums in software design with suitable examples.

Ans: Interfaces:

An interface acts as a blueprint or contract for classes. It defines what operations a class must perform, but not how those operations are implemented. Interfaces help in achieving abstraction, polymorphism, and loose coupling in software design.

Importance of Interfaces:

- **Improves flexibility:** Multiple classes can implement the same interface in their own way.
- **Supports multiple inheritance:** A class can implement more than one interface.
- **Enhances scalability and testing:** New implementations can be added without modifying existing code.

Example

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

class CreditCardPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using Credit Card");
    }
}

class PayPalPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using PayPal");
    }
}
```

Here, different payment methods follow the same contract but provide their own implementations.

Enums:

An enum (enumeration) is a special data type used to define a collection of fixed, constant values. Enums increase type safety, readability, and maintainability of code.

Importance of Enums

- **Restricts invalid values:** Only predefined constants are allowed.
- **Makes code self-explanatory:** Improves clarity and understanding.
- **Reduces bugs:** Avoids errors caused by using arbitrary strings or numbers.

Example

```
enum OrderStatus {
   PLACED, SHIPPED, DELIVERED, CANCELLED
}

class Order {
   OrderStatus status;
}
```

This ensures that the order status can only have valid predefined values.

Q2. Discuss how interfaces enable loose coupling with an example.

Ans: **Loose coupling** means that components of a system are minimally dependent on each other. Interfaces help achieve this by allowing classes to interact through a common contract instead of concrete implementations.

When a class depends on an interface, it does not need to know:

- which specific implementation is being used, or
- how the functionality is internally handled.

This makes the system easier to modify, extend, and test.

Example

Step 1: Define an Interface

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

Step 2: Create Implementations

```
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);
    }
}
```

Step 3: Use the Interface in a Client Class

```
class Notification {
    private MessageService service;

    Notification(MessageService service) {
        this.service = service;
    }

    void notifyUser(String message) {
        service.sendMessage(message);
    }
}
```

Usage:

```
MessageService service = new EmailService();
Notification notification = new Notification(service);
notification.notifyUser("Hello!");
```

Explanation

The **Notification** class depends only on the **MessageService** interface, not on any specific implementation. This allows switching from **EmailService** to **SMSService** without changing the Notification class, thus achieving **loose coupling**.