ASSIGNMENT-1

Student Name: Nitish Kumar Singh
UID: 23BCS12331
Branch: BE-CSE
Section/Group: KRG_2B
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

An interface is a blueprint of a class that defines a set of methods without providing their implementation. It specifies what a class must do rather than how it should do it. Interfaces are widely used in software design to achieve abstraction, multiple inheritance, and flexibility.

Role of Interfaces in Software Design

1. Promotes Abstraction: Interfaces hide implementation details and expose only the essential functionalities.
2. Encourages Consistency: Multiple classes implementing the same interface must follow a common structure.
3. Supports Multiple Inheritance: A class can implement multiple interfaces, allowing better design compared to single class inheritance.
4. Improves Maintainability: Changes in one class do not affect others as long as the interface contract is maintained.

Example

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

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

class UPI implements Payment {
```

```
    public void pay(double amount) {
        System.out.println("Paid using UPI: " + amount);
    }
}
```

Explanation:

Here, Payment is an interface. Both CreditCard and UPI implement it but provide their own payment methods. This makes the system flexible — new payment methods can be added without modifying existing code.

Enums

An enum (enumeration) is a special data type used to define a collection of constants. It improves code readability and prevents invalid values from being assigned.

Role of Enums in Software Design

1. Improves Code Clarity: Replaces numeric or string constants with meaningful names.
2. Enhances Type Safety: Only predefined values are allowed.
3. Reduces Errors: Prevents assigning incorrect values.
4. Better Maintainability: Easy to update constants in one place.

Example

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

class Order {
    OrderStatus status;
}
```

Explanation:

Instead of using strings like "pending" or "shipped", the enum ensures only valid statuses are used, making the application safer and easier to understand.

Q2. Discuss how interfaces enable loose coupling with example.

SOLUTION-

Loose coupling refers to a design principle where components of a system depend minimally on each other. This makes the system more scalable, flexible, and easier to modify.

Interfaces separate the definition of functionality from its implementation. A class interacts with the interface rather than the actual implementation, so the implementation can be changed without affecting the dependent class.

Example

```
interface Notification {
    void send(String message);
}

class EmailNotification implements Notification {
    public void send(String message) {
        System.out.println("Sending Email: " + message);
    }
}

class SMSNotification implements Notification {
    public void send(String message) {
        System.out.println("Sending SMS: " + message);
    }
}

class AlertService {
    private Notification notification;

    AlertService(Notification notification) {
        this.notification = notification;
    }

    void alertUser(String msg) {
```

```
        notification.send(msg);
    }
}
```

Explanation:

AlertService does not depend on EmailNotification or SMSNotification directly. It only depends on the Notification interface. If a new notification type (e.g., Push Notification) is added, no changes are required in AlertService.