# ASSIGNMENT

**Student Name: Saksham** **UID: 23BCS13613**
**Branch: BE- CSE** **Section/Group: 23BCS-KRG_2B**
**Semester: 6th** **Subject Name: System Design**

## Que 1. Explain the role of interfaces and enums in software design with proper examples.

Ans.

**INTERFACE:** An interface defines a contract that a class must follow. It specifies what a class should do, but not how it should do it. Interfaces help achieve abstraction, loose coupling, and polymorphism**.**

**IMPORTANCE OF INTERFACE:**

- Promote flexibility: different classes can implement the same interface in different ways.
- Support multiple inheritance of behavior (a class can implement multiple interfaces).
- Make systems easier to extend and test**.**

**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");
    }
}
```

**ENUMS:** An enum (enumeration) is a special data type that represents a fixed set of predefined constants. Enums improve type safety**,** readability**,** and maintainability.

**IMPORTANCE OF INTERFACE:**

- Prevent invalid values (only allowed constants can be used).
- Make code more self-documenting**.**
- Reduce errors caused by using strings or numbers for fixed values.

**EXAMPLE:**

```
enum OrderStatus {
    PLACED,
    SHIPPED,
    DELIVERED,
    CANCELLED
}
class Order {
    OrderStatus status;
}
```

---

## Que 2. Discuss how interfaces enable loose coupling with example.

## Ans.

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

- which concrete class is being used, or
- how the behavior is implemented?

This reduces dependency between components and makes the system easier to modify, extend, and test.

**EXAMPLE:**

Step 1: Define an Interface

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

Step 2: Provide 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!");
```