Assignment-01

Student Name: Parv Bansal
UID: 23BCS13701
Branch: BE-CSE
Section/Group: KRG_2B
Semester: 6th
Date of Performance: 1/2/26
Subject Name: System Design
Subject Code: 23CSH-314

Q1. Explain the role of interfaces and enums in software design with proper examples.

Ans) **Interface:** An interface in software design is used to define a common set of methods that different classes must implement, ensuring uniform behavior across the system. Interfaces help in achieving abstraction and flexibility by allowing the implementation to vary independently from the usage.

For example, in a **Payment Processing Application**, an interface PaymentMethod can define a method payAmount(), which is implemented by classes such as CreditCardPayment, DebitCardPayment, and UPIPayment. This allows the application to process payments through different modes without changing the core business logic, making the system easier to maintain and extend.

```
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: Enums are used in software design to represent a fixed set of predefined constants, ensuring consistency and type safety in applications. They help avoid errors caused by invalid values and improve code readability.

For example, in an **Order Management System**, an enum OrderStatus can contain values such as PENDING, CONFIRMED, SHIPPED, and DELIVERED. Using an enum ensures that the order status can only take one of these valid states, reducing bugs and making the application logic clearer and more reliable.

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

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

Ans) Interfaces enable loose coupling by allowing a program to depend on abstractions rather than concrete implementations.

- Interfaces define *what* an object can do, not *how* it does it.
- The calling class depends only on the interface, not the actual implementation.
- Changes in implementation do not affect the dependent classes.
- New implementations can be added easily without modifying existing code.

For example, In a Notification System, an interface **NotificationService** defines a method **sendNotification**(). This interface is implemented by classes such as **EmailNotification** and **SMSNotification**, where each class provides its own implementation for sending notifications through different channels."

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

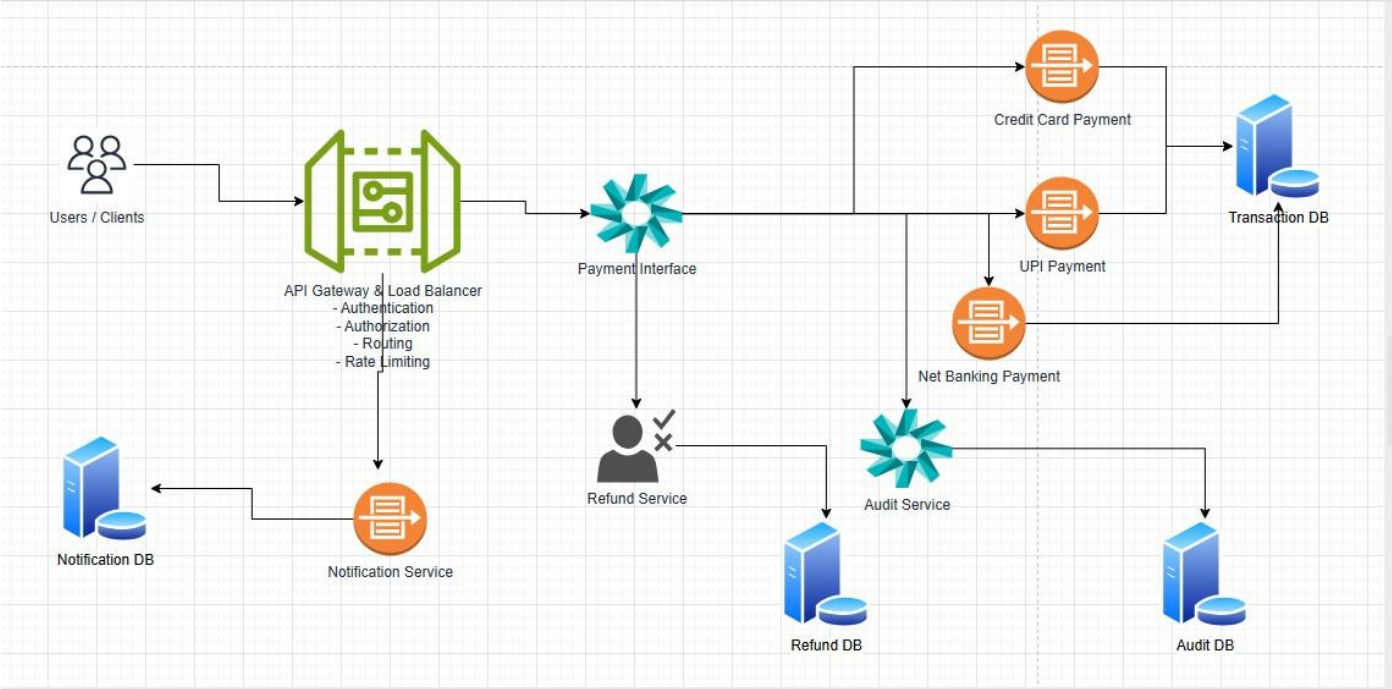
```
class Notification {
    private MessageService service;

    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!");
```

Q3. Design a High-Level Design (HLD) for a Payment Processing System, showing where interfaces would be used

***Ans*)**



This diagram shows the High-Level Design (HLD) of a Payment Processing System.

- Users/Clients initiate payment requests.
- Requests first go to the API Gateway & Load Balancer, which handles authentication, authorization, routing, and rate limiting.
- The request is forwarded to a Payment Interface (interface layer), which abstracts different payment methods.
- Based on the selected option, the interface routes the request to:
    - Credit Card Payment
    - UPI Payment
    - Net Banking Payment
- All successful or failed transactions are stored in the Transaction Database.
- If a refund is required, the Refund Service is triggered and details are saved in the Refund DB.
- The Audit Service records all payment and refund activities in the Audit DB for compliance and tracking.
- The Notification Service sends payment or refund updates to users and stores logs in the Notification DB.

Role of Interfaces:

The **Payment Interface** ensures loose coupling by providing a common contract for all payment methods.