DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

# Assignment-01

**Student Name:** Muskan Yadav **UID:** 23BCS10179
**Branch:** BE-CSE **Section:** Group:KRG-3B
**Semester:** 6 **Date of Submission**: 04-02-2026
**Subject name**: System Design **Subject code**: 23CSH-314

1. **Explain the role of interfaces and enums in software design with proper examples.**
   - **Interfaces:** An **interface** is a collection of method declarations without implementation. It defines a **contract** that a class must follow.
   - **Role in software design:**
     1. Promotes **abstraction** by separating *what* a class does from *how* it does it.
     2. Supports **loose coupling**, making systems easier to modify and extend.
     3. Enables **polymorphism**, allowing different classes to be treated uniformly.
     4. Improves **maintainability and testability**
   - Example:

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

class CreditCardPayment implements Payment {
   public void pay(double amount) {
      System.out.println("Paid using credit card");
   }
}
```

   - **Enums**: An **enum** (enumeration) is a data type that consists of a **fixed set of predefined constants**.
   - **Role in software design:**
     1. Restricts values to a **valid set**, preventing invalid states
     2. Improves type safety compared to strings or integers
     3. Makes code more readable and self-documenting
     4. Centralizes related constants

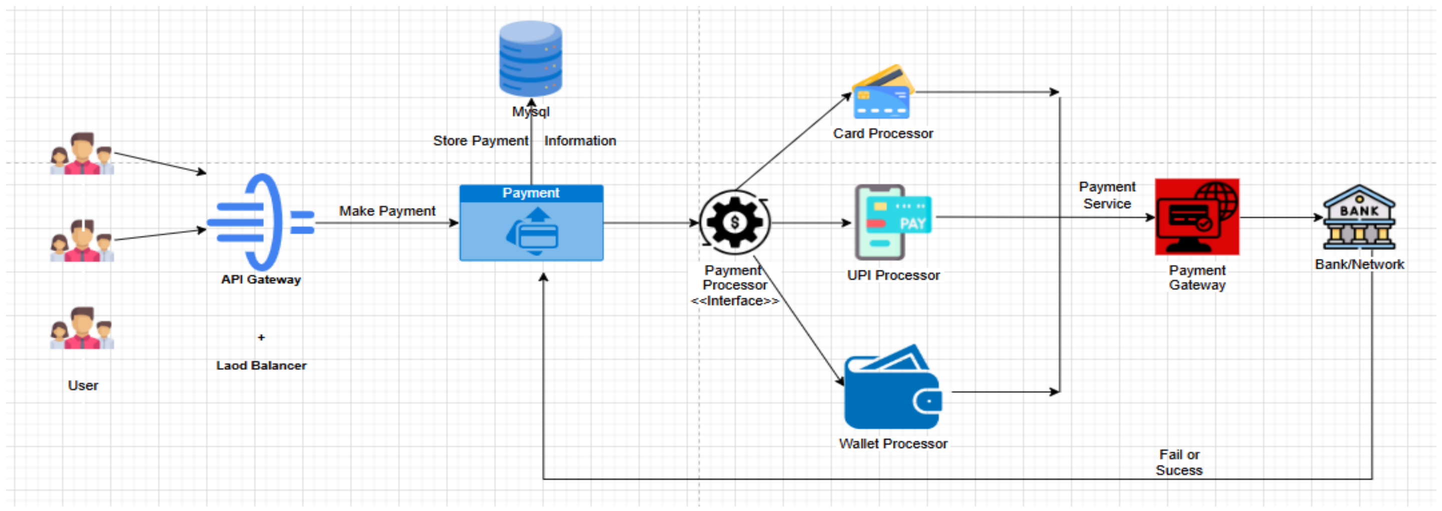
- **Example**:
```
enum OrderStatus {
    NEW, SHIPPED, DELIVERED, CANCELLED
}
```

2. **Discuss how interfaces enable loose coupling with examples?**
   - **Loose coupling** means that classes depend on **abstractions** rather than concrete implementations. Changes in one class have minimal or no impact on other classes.
   - **Role of Interfaces:**
     1. An interface defines a common set of methods without implementation. When a class depends on an interface instead of a concrete class, the dependency becomes flexible.
     2. Interfaces enable loose coupling by:
        A. Decoupling client code from implementation details
        B. Allowing implementations to be changed or extended without modifying client code
        C. Supporting polymorphism
        D. Improving maintainability and testability
   - **Example**:

```
interface MessageService {

    void sendMessage(String message);

}

class EmailService implements MessageService {

    public void sendMessage(String message) {

        System.out.println("Sending email: " + message);

    }

}
```

```
class SMSService implements MessageService {
    public void sendMessage(String message) {
        System.out.println("Sending SMS: " + message);
    }
}
class Notification {
    private MessageService service;
    Notification(MessageService service) {
        this.service = service;
    }
    void notifyUser(String message) {
        service.sendMessage(message);
    }
}
```

3. **Design an HLD for a Payment Processing System, showing where interfaces would be used.**


Mysql
Store Payment Information
User
API Gateway
+
Laod Balancer
Make Payment
Payment
Payment Processor <<Interface>>
Card Processor
PAY
UPI Processor
Wallet Processor
Payment Service
Payment Gateway
BANK
Bank/Network
Fail or Sucess