CHANDIGARH UNIVERSITY

Subject: System Design

Subject code: 23CSH 314

Assignment No.: 1

Student Name: Somanyu Singh

UID: 23BCS 10801

Branch: CSE

Section Group: 23BCS KRG 3B

Submitted to: Er Alok kumar (E17546)

Date of Submission: 4 Feb 2026

Question 1: Explain the role of interface and enums in software design with proper example

Solution: Role of interface in software Design

- Definition of Interface
  * An interface is blueprint that define contract for classes
  * It specifies what method a class must implement but not how those method are implemented.
  * In system design terms, an Interface represents behavioral abstraction.

- why Interface Are important in system Design

Interface are used to achieve the following design goals:

1) Abstraction
   - Hide implementation details
   - Exposes only essential behavior
2) Loose coupling
   - High level modules depend on interface, not concrete class
   - changes in implementation do not affect the system
3) scalability
   - New implementation can be added without modifying existing code
4) Multiple inheritance
   - Language like Java allow multiple inheritance using interfaces
5) Testability
   - Mock or fake implementation can be used for testing.

Example

```
interface payments {
    void pay (double amount);
}
class credit card payment implements Payment {
    public void pay (double amount) {
        system.out.println ("paid using credit card : " +
            amount);
    }
}
class upi payment implements payment {
    public void pay (double amount) {
        system.out.println ("paid using UPI : " +
            amount);
    }
}
public class payment implementation {
    public static void main (String[] args) {
        payment payment = new upi payment();
        payment.pay (500);
    }
}
```

- Roles :
  1. used in API design
  2. used in microservice communication
  3. supports Dependency injection
  4. follows open-closed principle
  5. Interface enable contract first design which is critical in loose systems.

Role of Enums in software Design

An enum (enumeration) is data type that represent a fixed set of predefined constant values.

Enums ensure that only valid values are used in a system

Why Enums are Important

* Enums helps in:
  1. Data Integrity
     - Prevent Invalid or unexpected values
  2. Readability
     - code becomes self-explanatory
  3. Maintainability
     - centralized defination of constants
  4. Elimination of magic strings
     - Avoids hard-coded strings like "delivered", "DELIVERED", "deliverd".

Problem without Enums

```
string status = "Delivereed";
```

→ syntically it is correct But may lead to logical error and system failure.

Example (order processing system)

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

```
OrderStatus status = OrderStatus.SHIPPED;

switch (status) {
    case CREATED:
        System.out.println("Order placed");
        break;
    case DELIVERED:
        System.out.println("Ordered Delivered");
        break;
}
```

Enums are commonly used in

- order status tracking
- payment state management
- user roles (ADMIN, USER, GUEST)

Interface	Enums
✓ Define behavior	✓ Define fixed values
✓ Abstraction	✓ Data integrity
✓ Contracts	✓ Used for states / constants
✓ Architecture design	✓ Workflow control
System Role	System Role
✓ More flexibility	✓ less flexibility.

Question 2 Discuss how Interfaces enable loose coupling with suitable example.

solution: In software design, loose coupling is a key design principle that helps in building flexible, scalable, and maintainable systems. interfaces play a major role in achieving loose coupling by separating what a system does from how it is implemented.

Meaning of loose coupling

- components of a system have minimum dependancy on each other
- Changes in one components do not force changes in other components
- components can be replaced or modified independently.
- loose coupling is essential in system design, microservices, APIs and enterprise application

Role of Interfaces in Achieving loose coupling

An interface acts as middle layer (contract) between two components. Instead of one class depending directly on another class we design one class depending on an interface. This Remove tight binding Between components

Example

① Tightly coupled Design (without Interface)

```
class Payment service {
    CreditCardPayment payment = new CreditCardPayment();
void process payment (double amount) {
```

payment.pay(amount);

}

{

Problem with this design:

① Payment Service is tightly bound to CreditCard Payment

② Adding UPI or wallet require code modification

③ Violets open closed principle

④ Difficult to test and extend

Solution loose coupling using Interface

```
Interface Payment {
    void pay(double amount);
}
class CreditCardPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid using Credit
            Card" + amount);
    }
class UpiPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Paid using UPI:" +
            amount);
    }
}
class PaymentService {
    private Payment payment;
```

```
PaymentService(Payment payment){
    this.payment = payment;
}
void processPayment(double amount){
    payment.pay(amount);
}
}
Payment payment = new UPIPayment();
PaymentService service = new PaymentService(payment);
service.processPayment(500);
```

Hence we can see paymentservice depends on interface, not implementation

payment method can be changed without modifying service code