ASSIGNMENT: 1

Student Name: Amol Saurav
UID: 23BCS10067
Section/Group: KRG-2B
Semester: 6th
Subject Name: System Design
Subject Code: 23CSH-314

Question 1: Explain the role of interfaces and enums in software design with proper examples.

Solution:

Interface: An interface defines a contract that a class must follow, it specifies what operations a class must provide without defining how they are implemented.

Role of interface in software design:

- Interface enables loose coupling between components
- **Abstraction:** Exposes behavior without implementation details.
- **Polymorphism:** Different classes can be used interchangeably if they implement the same interface.
- **Scalability & Maintainability:** New implementations can be added easily without changing the existing code.

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

```
class Checkout {
   private Payment payment;
   Checkout(Payment payment) { this.payment = payment;}
   void completeOrder(double amount) {
      payment.pay(amount);
   }
}
```

Here we can see that whenever we need to add a method of payment then we don't need to change or add any thing in Payment interface rather we can implement it in some other class to add a payment method.

ENUM: An Enum (Enumeration) is a special data type that represents a fixed set of constant values.

Role of Enum in software design:

- Improves code readability.
- Avoids magic numbers and strings.
- Prevents invalid values.
- Used in workflows and state machines.
- **Behavior encapsulation:** Enums can contain fields and methods.

Example:

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

   void setStatus(OrderStatus status) {
      this.status = status;
   }
   void printStatus() {
```

```
            System.out.println("Order status: " + status);
        }
    }
```

The best thing of Enum is that it prevents magic characters like if we want to update status by using status=”Done” then it is not possible here, it restricts value to a predefined list of constants.

Question 2: Discuss how interfaces enable loose coupling with example.

Solution:

Loose coupling means that one part of a program depends as little as possible on another part. Changes in one module should not force changes in other modules.

Interfaces play a key role in achieving this by allowing code to depend on abstractions (interfaces) instead of concrete implementations (classes).

How Interfaces Enable Loose Coupling

- **Dependency on abstraction, not implementation**: Classes interact with an interface, not with a specific class.
- **Easy replacement of implementations**: One implementation can be replaced with another without changing dependent code.
- **Improved maintainability**: Changes in implementation do not affect the client code.
- **Supports polymorphism**: Different objects can be treated uniformly through the same interface.

Example:

```
interface NotificationService {
    void sendMessage(String message);
}
class EmailNotification implements NotificationService {
    public void sendMessage(String message) {
        System.out.println("Email sent: " + message);
    }
}
```

```
class SMSNotification implements NotificationService {
    public void sendMessage(String message) {
        System.out.println("SMS sent: " + message);
    }
}
class UserService {
    private NotificationService notificationService;
    UserService(NotificationService notificationService) {
        this.notificationService = notificationService;
    }
    void notifyUser() {
        notificationService.sendMessage("Welcome User");
    }
}
public class Main {
    public static void main(String[] args) {
        NotificationService email = new EmailNotification();
        UserService userService1 = new UserService(email);
        userService1.notifyUser();
        NotificationService sms = new SMSNotification();
        UserService userService2 = new UserService(sms);
        userService2.notifyUser();
    }
}
```

Here,

UserService does not know whether the message is sent via Email or SMS.

Switching from EmailNotification to SMSNotification requires no change in UserService.

New notification methods (e.g., WhatsApp) can be added without modifying existing code.