Assignment 1

Student Name: Ankesh Amar **UID:** 23BCS12463
Branch: BE-CSE **Section/Group:** KRG-1A
Semester: 6th **Date of Performance:** 03/01/26
Subject Name: System Design **Subject Code:** 23CSH-314

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

Answer :

1. Role of Interface in Software Design

An **interface** is a blueprint of a class. It contains only method declarations (no implementation).

It is used to:

- Achieve **abstraction**
- Support **multiple inheritance**
- Promote **flexibility**
- Improve **maintainability**

An interface defines **what to do**, not **how to do**.

Example:

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

class CreditCard implements Payment {
   public void pay(double amount) {
      System.out.println("Paid by Credit Card");
   }
}

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

Here, `Payment` is an interface and different classes implement it.

2. Role of Enum in Software Design

An **enum (Enumeration)** is used to define a fixed set of constants.

It improves:

- **Code readability**
- **Type safety**
- **Error reduction**

Enums prevent invalid values.

Example:

```
enum PaymentStatus {
   SUCCESS,
   FAILED,
   PENDING
}
```

Usage:

```
PaymentStatus status = PaymentStatus.SUCCESS;
```

Conclusion:

- Interfaces help in designing flexible systems.
- Enums help in managing fixed values safely.
- Both improve software quality.

Q2. Discuss how interfaces enable loose coupling with example.

Answer :

Meaning of Loose Coupling

Loose coupling means that different parts of a software system are **independent of each other**.
In a loosely coupled system, changes in one class do **not affect other classes**, which makes the system easy to modify and maintain.

Role of Interface in Loose Coupling

An interface separates **what a class does** from **how it does it**.Classes depend on the **interface**, not on the actual implementation.

Because of this:

- Dependency between classes is reduced
- Code becomes flexible
- New features can be added easily

Example

Interface

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

Implementing Classes

```
class CreditCard implements Payment {
    public void pay(double amount) {
        System.out.println("Paid by Credit Card");
    }
}

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

Service Class Using Interface

```
class PaymentService {
    Payment payment;

    PaymentService(Payment payment) {
        this.payment = payment;
    }

    void makePayment(double amount) {
        payment.pay(amount);
    }
}
```

Usage

```
Payment p = new UPI();
PaymentService service = new PaymentService(p);
service.makePayment(500);
```

Explanation

Here, PaymentService does not depend on CreditCard or `UPI` directly.
It depends only on the `Payment` interface.

So, if a new payment method is added, no change is required in PaymentService.

This shows **loose coupling**.

Advantages of Loose Coupling Using Interface

1. Easy to change implementation
2. High flexibility
3. Better code reuse
4. Easy testing
5. Improved maintenance

Conclusion

Interfaces enable loose coupling by removing direct dependency between classes.
They make the system flexible, scalable, and easy to maintain.