Assignment - 1

Student Name: Geetika
UID: 23BCS12885
Branch: BE-CSE
Section/Group: KRG_3A
Semester: 6th
Date of Submission: 04-02-2026
Subject Name: System Design
Subject Code: 23CSH-314

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

Ans.
In software design, interfaces and enums play an important role in building modular, maintainable, and scalable systems.

An interface defines a contract that specifies a set of method declarations without providing their implementations. Any class that implements an interface must implement all its methods. Interfaces support abstraction by separating what a system does from how it does it. They promote loose coupling, enable polymorphism, and allow multiple implementations to be used interchangeably, which improves extensibility and maintainability of the system.

Example:

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

class CreditCardPayment implements Payment {
   public void pay(double amount) {
      System.out.println("Payment made using Credit Card: " + amount);
   }
}

class UPIPayment implements Payment {
   public void pay(double amount) {
      System.out.println("Payment made using UPI: " + amount);
   }
}
```

In this example, both CreditCardPayment and UPIPayment implement the Payment interface. New payment methods can be added without modifying existing code.

An **enum (enumeration)** is a special data type that represents a fixed set of named constants. Enums improve readability, type safety, and reduce the use of invalid values in a program. They are commonly used to represent states, modes, or predefined options in an application.

Example:

```
enum OrderStatus {
```

```
    PLACED,

    SHIPPED,

    DELIVERED,

    CANCELLED

}
```

Here, the OrderStatus enum ensures that an order can only have valid and predefined states, preventing incorrect or invalid status values.

Q2. Discuss how interfaces enable loose coupling with example.

Ans:

Loose coupling refers to a design approach where components of a software system have minimal dependency on each other. Interfaces play a key role in achieving loose coupling by allowing interaction through contracts rather than concrete implementations.

Role of Interfaces in Loose Coupling

1. Abstraction
   Interfaces define what operations are required, not how they are performed. This hides implementation details from the client code.
2. Dependency on Interface, Not Implementation
   Classes depend on interfaces instead of concrete classes. This follows the Dependency Inversion Principle, making systems more flexible.
3. Easy Replacement and Extension
   New implementations can be added or existing ones replaced without changing client code, improving scalability.
4. Improved Maintainability and Testing
   Changes in one module do not directly affect others. Mock implementations can be used for unit testing.

Example

```
interface Notification
{    void send(String
message);
}

class EmailNotification implements Notification
{    public void send(String message) {
       System.out.println("Email sent: " + message);
    }
}

class SMSNotification implements Notification
{    public void send(String message) {
       System.out.println("SMS sent: " + message);
    }
}
```

```
class AlertService {     private
Notification notification;

   AlertService(Notification notification) {
      this.notification =
notification;     }

   void alert(String msg) {
      notification.send(msg);
   }  }
```

The AlertService depends on the Notification interface, not on EmailNotification or SMSNotification. Any new notification type can be added without modifying AlertService.