Assignment - 1

Student Name: Yashwat Pratap Singh
UID: 23BCS12565
Branch: BE-CSE
Section/Group: KRG_2A
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 methods which a class must implement, without providing their implementation. Interfaces help achieve abstraction by separating what a system does from how it does it. They promote loose coupling, allowing different implementations to be used interchangeably without changing client code. Interfaces also support polymorphism and enable multiple inheritance of behavior in languages like Java. This improves code reusability, testability, and system extensibility.
Example: A Payment interface can be implemented by CreditCardPayment and UPIPayment, allowing new payment methods to be added without modifying existing code.

An enum (enumeration) is a special data type used to define a fixed set of named constants. Enums enhance code readability by replacing magic values with meaningful names. They provide type safety, ensuring that variables can only hold predefined valid values. Enums are widely used to represent states, modes, or configuration options, reducing runtime errors and improving maintainability.

Example: An OrderStatus enum may include values such as PLACED, SHIPPED, DELIVERED, and CANCELLED, ensuring valid order state management.
Interfaces enable flexible and extensible behavior, while enums ensure controlled and meaningful value representation. Together, they contribute to robust, clean, and well-designed software architectures.

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.