ASSIGNMENT

Student Name: Sujal
Branch: BE- CSE
Semester: 6th
UID: 23BCS10271
Section/Group: KRG2_A
Subject Name: System Design

Q1. Explain the role of Interfaces and Enums in software design with proper examples?

Ans:

INTERFACE: An interface is a blueprint that defines what a class should do, not how it does it.
It contains method declarations (and constants), and any class that implements the interface must provide implementations for all its methods.

Role of Interfaces

1. Achieves Abstraction

- ☐ Hides implementation details
- ☐ Exposes only essential behavior to the user

2. Supports Multiple Inheritance

- ☐ A class can implement multiple interfaces
- ☐ Avoids ambiguity problems of multiple class inheritance

3. Promotes Loose Coupling

- ☐Code depends on interfaces rather than concrete classes
- ☐ Makes the system flexible and easier to modify

4. Improves Maintainability and Scalability

- ☐ New implementations can be added with minimal code changes
- ☐ Suitable for large-scale applications and APIs

Example:

```
interfacePayment {
    void pay(double amount);
}

class CreditCardPayment implements Payment {

    public void pay(double amount) {
        System.out.println("Paid using Credit Card");
    }
}
```

ENUMS: An enum (enumeration) is a special data type used to define a fixed set of constant values. Itimproves code readability and ensures type safety.

Role of Enums

1. Provides Type Safety: Prevents invalid or incorrect values
2. Improves Code Readability: Uses meaningful names instead of numbers or strings
3. Groups Related Constants: Keeps related values together in one place

Example of Enum

```
enum OrderStatus {

    PLACED,

    SHIPPED,

    DELIVERED,

    CANCELLED

}
```

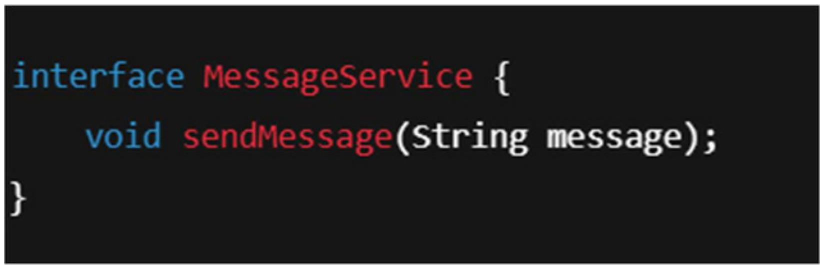
Que 2. Discuss how interfaces enable loose coupling with example.

Ans.

Loose Coupling: Loose coupling isasoftwaredesignprinciple in which differentcomponents of a system dependas little aspossible on each other.
In a loosely coupled system, changes made in one component do not force changes in other components, making the system flexible and easy to maintain.

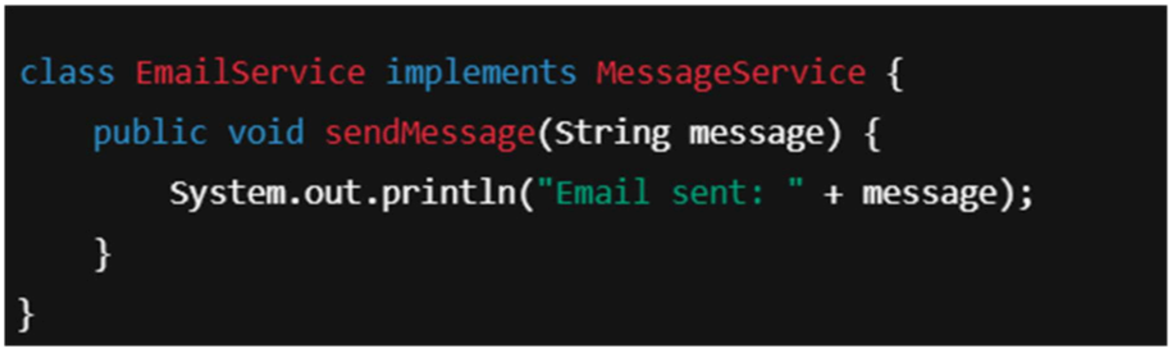
Role of Interfaces in Enabling Loose Coupling:
• Interfacesdefine what a classshoulddo, not how it should do it
• Client classes depend on interfaces instead of concrete implementations
• Implementation details are hidden from the client
• Multiple implementations of the same interface can be used interchangeably
• Enhances flexibility, reusability, scalability, and testability

Example Using Interface:



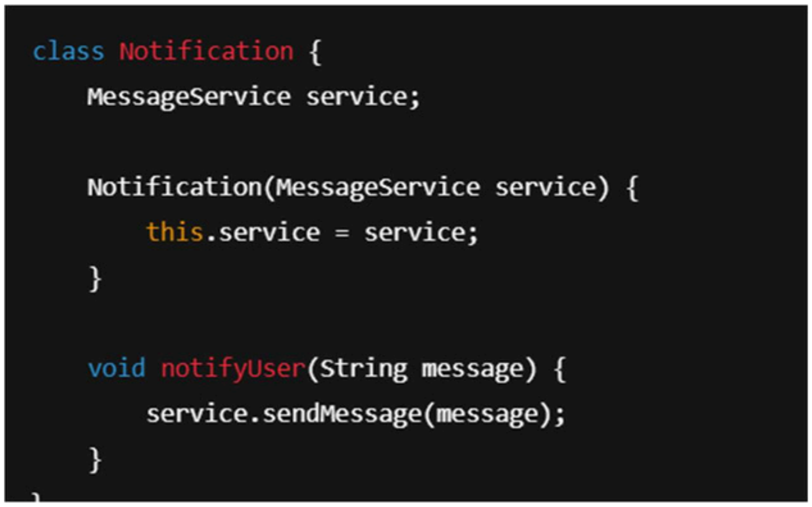
```
interface MessageService {
    void sendMessage(String message);
}
```

• The interface declares the required behavior



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

• EmailService provides one implementation of the interface



```
class Notification {
    MessageService service;

    Notification(MessageService service) {
        this.service = service;
    }

    void notifyUser(String message) {
        service.sendMessage(message);
    }
}
```

•The client class depends only on the interface
•The implementation can be changed without modifying the client class

Conclusion
•Interfaces help in achieving loose coupling by separating abstraction from implementation
•They allow easy modification and extension of the system
•This results in a flexible, maintainable, and scalable software design