

TASK 8: Interfaces & Abstraction – Payment Gateway Simulator.

1. What is Abstraction?

Abstraction means hiding internal implementation details and showing only what is necessary to the user.

Real-life example:

When you use an ATM:

- You insert a card
- Enter PIN
- Get money

You do not know how the bank server, database, or security works. This hiding of internal logic is called abstraction.

2. Why Abstraction is Important?

- Makes code simple
- Improves security
- Reduces complexity
- Makes system flexible and maintainable
- Allows loose coupling

3. How Abstraction is Achieved in Java?

Java supports abstraction using:

1. Abstract Class
2. Interface

4. What is an Interface?

An interface is a blueprint of a class that contains method declarations without implementation.

It defines what a class must do, not how it will do it.

5. Interface Syntax

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

Explanation:

- `interface` → keyword
- `Payment` → interface name
- `pay()` → abstract method
- No method body is present

6. Implementing an Interface:-

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

7. Interface and Abstraction Together:- Payment payment = new UPIPayment();
payment.pay(500);

8. Multiple Inheritance in Java

Java does not support multiple inheritance with classes
But Java supports multiple inheritance using interfaces

Example:

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

```
interface Logger {  
    void log();  
}
```

```
class CreditCardPayment implements Payment, Logger {
```

```
    public void pay() {  
        System.out.println("Payment successful");  
    }
```

```
    public void log() {  
        System.out.println("Payment logged");  
    }
```

```
}
```

```
}
```

9. Loose Coupling Using Interface

Loose coupling means:

- Classes depend on interfaces
- Not on actual implementations

Payment payment;

```
payment = new CreditCardPayment();
```

```
// can change to
```

```
payment = new UPIPayment();
```

No code change is required in the main logic.

Payment Gateway Simulator (Interfaces & Abstraction):-

- Use interface for abstraction
- Implement multiple payment modes
- Achieve loose coupling
- Demonstrate multiple inheritance using interfaces
- Handle invalid payment cases
- Switch payment methods at runtime
- Add meaningful logs.

The screenshot shows a Java IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** html code
- Explorer:** Shows various files including index.html, PaymentGatewaySimulator.java, CreditCardPayment.java, Payment.java, Logger.java, and several payment classes (UPIPayment, NetBankingPayment, etc.).
- Code Editor:** Displays the `PaymentGatewaySimulator.java` file with the following content:

```
1 import java.util.Scanner;
2 interface Payment {
3     void pay(double amount);
4 }interface Logger {
5     void log(String message);
6 }
7 class CreditCardPayment implements Payment, Logger {
8     public void pay(double amount) {
9         if (amount <= 0) {
10             log(message: "Invalid amount for Credit Card payment");
11             return;
12         }
13         log(message: "Processing Credit Card payment");
14         System.out.println(amount + " paid using Credit Card");
15     }
16     public void log(String message) {
17         System.out.println("[CREDIT CARD LOG] " + message);
18     }
19 }
20 class UPIPayment implements Payment, Logger {
21     public void pay(double amount) {
22         if (amount <= 0) {
23             log(message: "Invalid amount for UPI payment");
24             return;
25         }
26         log(message: "Processing UPI payment");
27         System.out.println(amount + " paid using UPI");
28     }
29     public void log(String message) {
30         System.out.println("[UPI LOG] " + message);
31     }
32 }
33 > class NetBankingPayment implements Payment, Logger {
34     public void pay(double amount) {
35         if (amount <= 0) {
36             log(message: "Invalid amount for Net Banking payment");
37             return;
38         }
39         log(message: "Processing Net Banking payment");
40         System.out.println(amount + " paid using Net Banking");
41     }
42     public void log(String message) {
43         System.out.println("[NET BANKING LOG] " + message);
44     }
45 }
46 > class PaymentGateway {
47     private Payment paymentMethod;
48     public void setPaymentMethod(Payment paymentMethod) {
49         this.paymentMethod = paymentMethod;
50     }
51     public void makePayment(double amount) {
52         if (paymentMethod == null) {
53             System.out.println("No payment method selected");
54         } else {
55             paymentMethod.pay(amount);
56         }
57     }
58 }
59 public class PaymentGatewaySimulator {
60     public static void main(String[] args) {
61         Scanner sc = new Scanner(System.in);
62         PaymentGateway gateway = new PaymentGateway();
63         System.out.println("Select Payment Method:");
64         System.out.println("1. Credit Card");
65         System.out.println("2. UPI");
66         System.out.println("3. Net Banking");
67         int choice = sc.nextInt();
68         System.out.print("Enter amount: ");
69         double amount = sc.nextDouble();
70         if (choice == 1) {
71             gateway.setPaymentMethod(new CreditCardPayment());
72         } else if (choice == 2) {
73             gateway.setPaymentMethod(new UPIPayment());
74         } else if (choice == 3) {
75             gateway.setPaymentMethod(new NetBankingPayment());
76         } else {
77             System.out.println("Invalid payment option");
78             sc.close();
79             return;
80         }
81         gateway.makePayment(amount);
82     }
83 }
84 
```

The code implements a payment gateway simulator that can handle three payment methods: Credit Card, UPI, and Net Banking. It uses a scanner to get user input for the payment method and amount, and prints the payment details to the console.

Output:-

