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:

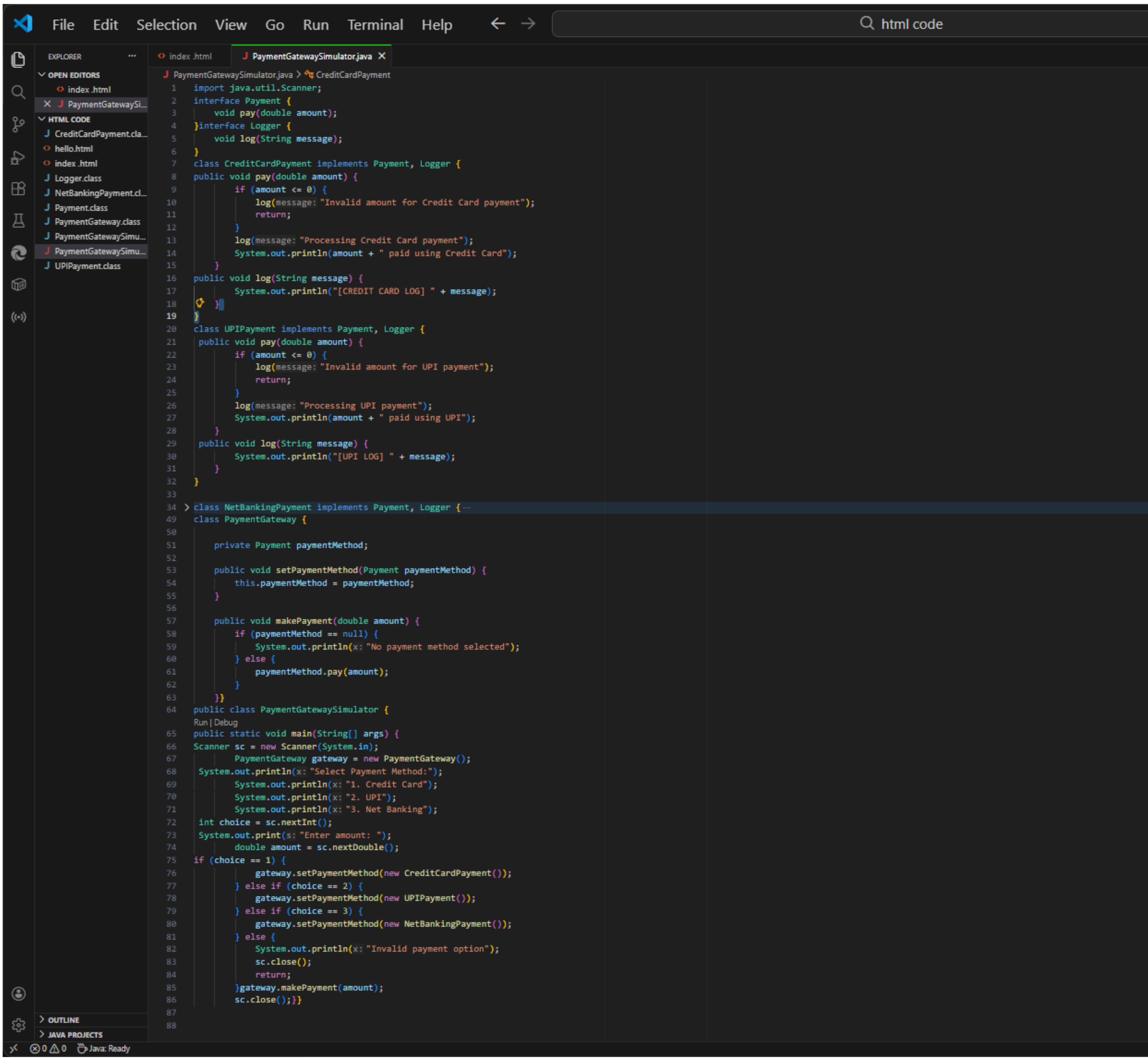
- Classes depend on interfaces
- Not on actual implementations

```
Payment payment;

payment = new CreditCardPayment();
// can change to
payment = new UPIPayment();
```

No code change is required in the main logic.

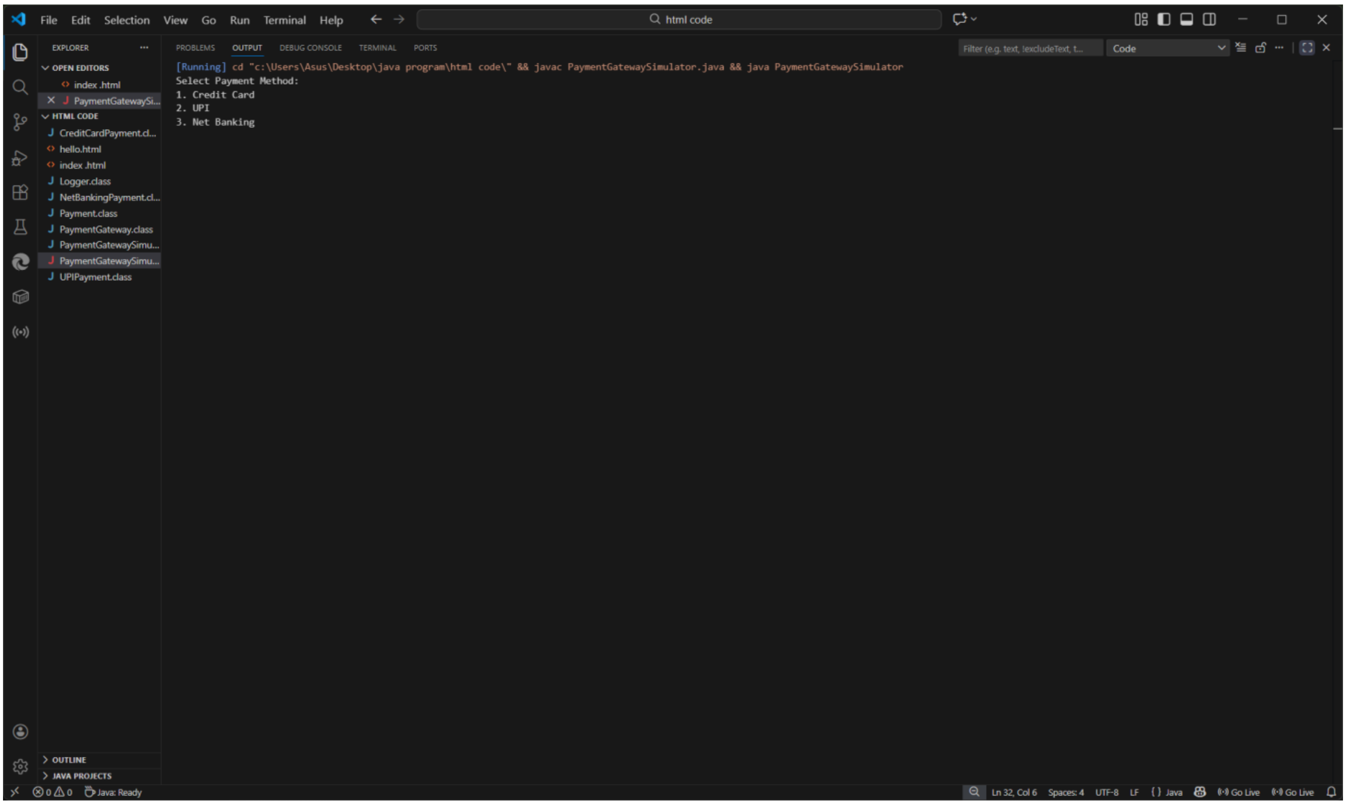
Payment Gateway Simulator (Interfaces & Abstraction):-

- 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.



```java
import java.util.Scanner;
interface Payment {
    void pay(double amount);
}interface Logger {
    void log(String message);
}
class CreditCardPayment implements Payment, Logger {
public void pay(double amount) {
        if (amount <= 0) {
            log(message: "Invalid amount for Credit Card payment");
            return;
        }
        log(message: "Processing Credit Card payment");
        System.out.println(amount + " paid using Credit Card");
    }
public void log(String message) {
        System.out.println("[CREDIT CARD LOG] " + message);
    }
}
class UPIPayment implements Payment, Logger {
 public void pay(double amount) {
        if (amount <= 0) {
            log(message: "Invalid amount for UPI payment");
            return;
        }
        log(message: "Processing UPI payment");
        System.out.println(amount + " paid using UPI");
    }
 public void log(String message) {
        System.out.println("[UPI LOG] " + message);
    }
}

class NetBankingPayment implements Payment, Logger {···
class PaymentGateway {

    private Payment paymentMethod;

    public void setPaymentMethod(Payment paymentMethod) {
        this.paymentMethod = paymentMethod;
    }

    public void makePayment(double amount) {
        if (paymentMethod == null) {
            System.out.println(x: "No payment method selected");
        } else {
            paymentMethod.pay(amount);
        }
    }}
public class PaymentGatewaySimulator {
Run | Debug
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
        PaymentGateway gateway = new PaymentGateway();
 System.out.println(x: "Select Payment Method:");
        System.out.println(x: "1. Credit Card");
        System.out.println(x: "2. UPI");
        System.out.println(x: "3. Net Banking");
 int choice = sc.nextInt();
 System.out.print(s: "Enter amount: ");
        double amount = sc.nextDouble();
if (choice == 1) {
            gateway.setPaymentMethod(new CreditCardPayment());
        } else if (choice == 2) {
            gateway.setPaymentMethod(new UPIPayment());
        } else if (choice == 3) {
            gateway.setPaymentMethod(new NetBankingPayment());
        } else {
            System.out.println(x: "Invalid payment option");
            sc.close();
            return;
        }gateway.makePayment(amount);
        sc.close();}}
```

Output:-



```
[Running] cd "c:\Users\Asus\Desktop\java program\html code\" && javac PaymentGatewaySimulator.java && java PaymentGatewaySimulator
Select Payment Method:
1. Credit Card
2. UPI
3. Net Banking
```