**Exercise 8: Implementing the Strategy Pattern**

**PaymentStrategy.java**

**package** mypackage;

**public** **interface** PaymentStrategy {

**void** pay(**double** amount);

}

**Implement Concrete Strategies**

**CreditCardPayment.java**

**package** mypackage;

**public** **class** CreditCardPayment **implements** PaymentStrategy {

**private** String cardNumber;

**private** String cardHolderName;

**public** CreditCardPayment(String cardNumber, String cardHolderName) {

**this**.cardNumber = cardNumber;

**this**.cardHolderName = cardHolderName;

}

**public** **void** pay(**double** amount) {

System.***out***.println("Paid ₹" + amount + " using Credit Card (" + cardHolderName + ")");

}

}

**PayPalPayment.java**

**package** mypackage;

**public** **class** PayPalPayment **implements** PaymentStrategy {

**private** String email;

**public** PayPalPayment(String email) {

**this**.email = email;

}

**public** **void** pay(**double** amount) {

System.***out***.println("Paid ₹" + amount + " using PayPal account (" + email + ")");

}

}

**Implement Context Class**

**PaymentContext.java**

**package** mypackage;

**public** **class** PaymentContext {

**private** PaymentStrategy paymentStrategy;

**public** **void** setPaymentStrategy(PaymentStrategy paymentStrategy) {

**this**.paymentStrategy = paymentStrategy;

}

**public** **void** payAmount(**double** amount) {

**if** (paymentStrategy == **null**) {

System.***out***.println("No payment strategy selected.");

**return**;

}

paymentStrategy.pay(amount);

}

}

**Test the Strategy Implementation with User Input**

**TestStrategyPattern.java**

**package** mypackage;

**import** java.util.Scanner;

**public** **class** TestStrategyPattern {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

PaymentContext context = **new** PaymentContext();

System.***out***.println("Enter payment amount:");

**double** amount = sc.nextDouble();

sc.nextLine(); // Consume newline

System.***out***.println("Select payment method:");

System.***out***.println("1. Credit Card");

System.***out***.println("2. PayPal");

**int** choice = sc.nextInt();

sc.nextLine(); // Consume newline

**switch** (choice) {

**case** 1:

System.***out***.println("Enter Card Number:");

String cardNumber = sc.nextLine();

System.***out***.println("Enter Card Holder Name:");

String cardHolder = sc.nextLine();

context.setPaymentStrategy(**new** CreditCardPayment(cardNumber, cardHolder));

**break**;

**case** 2:

System.***out***.println("Enter PayPal Email:");

String email = sc.nextLine();

context.setPaymentStrategy(**new** PayPalPayment(email));

**break**;

**default**:

System.***out***.println("Invalid choice. Exiting...");

sc.close();

**return**;

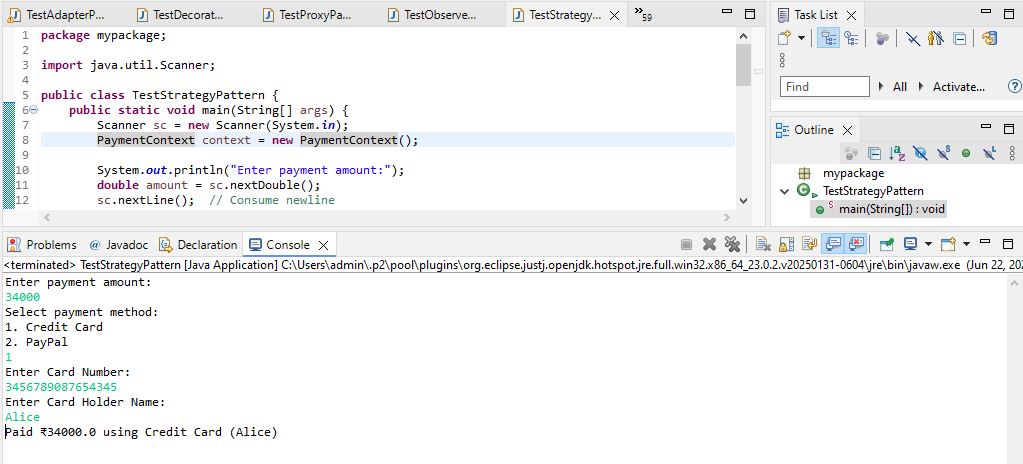
}

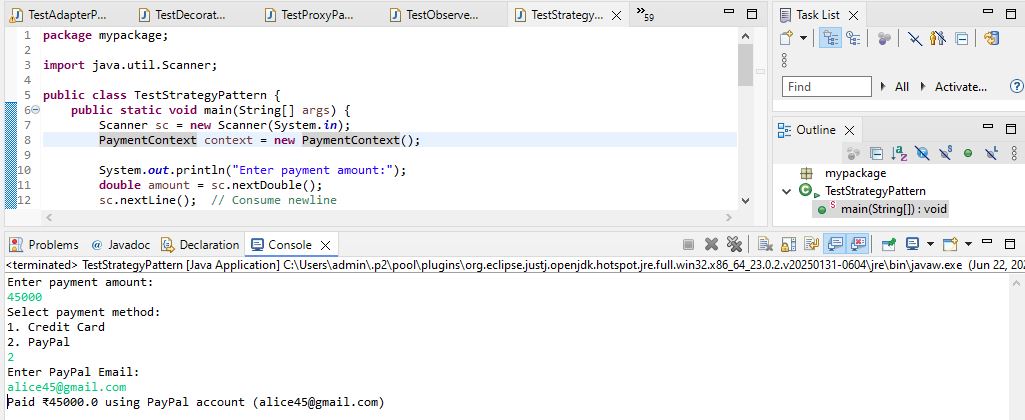
context.payAmount(amount);

sc.close();

}

}

****

****

We are developing a payment system that supports multiple payment methods like Credit Card and PayPal. The payment method is selected at runtime. We apply the Strategy Pattern to solve this.

**1. Strategy Interface**

* PaymentStrategy defines a common method pay().

**2. Concrete Strategies**

* CreditCardPayment implements PaymentStrategy and handles credit card payments.
* PayPalPayment implements PaymentStrategy and handles PayPal payments.

**3. Context Class**

* PaymentContext holds a reference to PaymentStrategy and delegates the payAmount() call to the selected strategy.

**4. Advantages of Strategy Pattern**

* Allows dynamic selection of algorithms at runtime.
* Promotes flexibility and easy extension of new payment methods.
* Removes conditional logic from the client code.
* Promotes open-closed principle (easy to add new strategies without modifying existing code).

**5. Time Complexity**

* O(1) for selecting and executing any strategy.

**6. Real-life Applications**

* Payment systems.
* Compression algorithms.
* Sorting algorithms selection.
* File format converters.