**Exercise 8: Implementing the Strategy Pattern**

**Scenario:**

You are developing a payment system where different payment methods (e.g., Credit Card, PayPal) can be selected at runtime. Use the Strategy Pattern to achieve this.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **StrategyPatternExample**.
2. **Define Strategy Interface:**
   * Create an interface PaymentStrategy with a method **pay()**.
3. **Implement Concrete Strategies:**
   * Create classes **CreditCardPayment**, **PayPalPayment** that implement **PaymentStrategy**.
4. **Implement Context Class:**
   * Create a class **PaymentContext** that holds a reference to **PaymentStrategy** and a method to execute the strategy.
5. **Test the Strategy Implementation:**
   * Create a test class to demonstrate selecting and using different payment strategies.

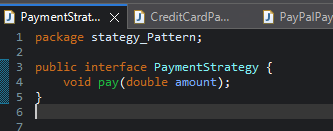
**Solution:**

| **Component** | **Role** | **Purpose** |
| --- | --- | --- |
| **PaymentStrategy** | Strategy Interface | Declares the method pay() |
| **CreditCardPayment, PayPalPayment** | Concrete Strategies | Implements different payment algorithms |
| **PaymentContext** | Context Class | Holds a PaymentStrategy reference and executes it |
| **StrategyTest** | Client Code | Selects and executes the strategy dynamically |

**Strategy Interface:**

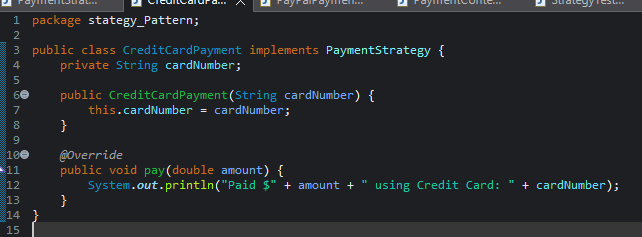
**PaymentStrategy.java**

* This interface defines the strategy behavior: a pay() method that takes an amount.

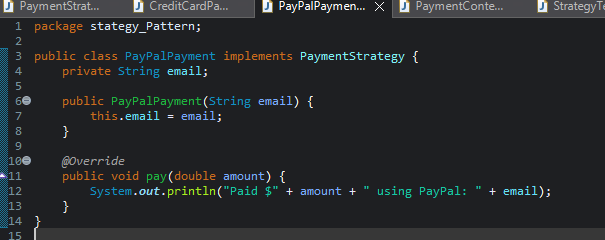


**Concrete Strategies**

**CreditCardPayment.java**

****

**PayPalPayment.java**

****

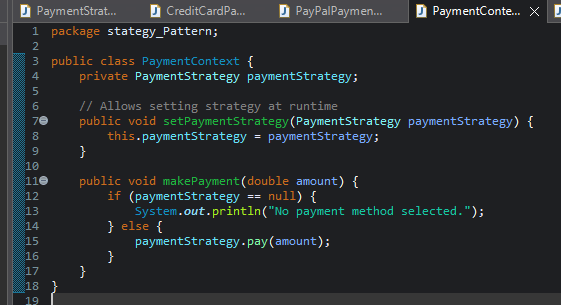
**Explanation:**

* Each strategy implements PaymentStrategy.
* Defines its own way of handling payment (card number, email, etc.).

**PaymentContext.java**

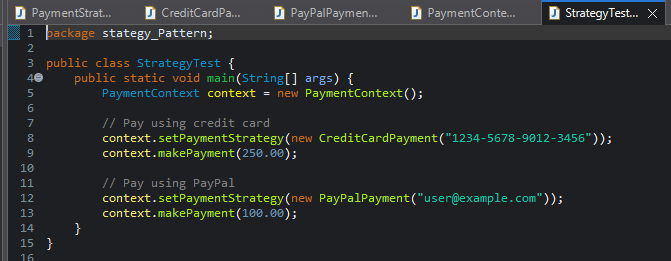
**Explanation:**

* The context class uses the strategy but does not know the actual implementation.
* You can change the strategy at runtime with setPaymentStrategy()

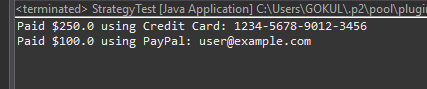


**StrategyTest.java**

* The client can switch payment methods dynamically.
* Both payment methods are used through the same interface (PaymentStrategy), promoting flexibility.



**Output:**

****

**Benefits of Strategy Pattern**

* ✔ Runtime flexibility: Choose algorithms/behaviors on the fly.
* ✔ Open/Closed Principle: Add new strategies without changing existing code.
* ✔ Decoupling: Business logic is separated from algorithm implementation.
* ✔ Used in payment systems, sorting, compression algorithms, etc.