28 Behavioral: Strategy Pattern — Pluggable Payment Gateway Assignment

Refactor a checkout flow so that payment processing is chosen **at runtime** via interchangeable strategy objects.  
 Every class must contain succinct JavaDoc, and a **Reflection** at the end captures strengths / trade-offs.

.

├── analysis

│ └── strategy\_need.md ← why strategies beat if-else ladders

├── src/main/java

│ └── payment

│ ├── strategy

│ │ ├── PaymentStrategy.java

│ │ ├── CreditCardPayment.java

│ │ ├── PayPalPayment.java

│ │ └── BitcoinPayment.java

│ ├── context

│ │ └── PaymentContext.java

│ └── StrategyDemo.java

├── src/test/java/payment

│ ├── CreditCardTest.java

│ ├── PayPalTest.java

│ └── NullStrategyTest.java

├── reflection.md

└── README.md

#### **1 PaymentStrategy.java**

package payment.strategy;

/\*\*

\* <p>Interchangeable algorithm for settling a monetary charge.</p>

\* Implementations handle the low-level gateway call.

\*/

public interface PaymentStrategy {

/\*\*

\* Executes the payment algorithm.

\*

\* @param amount amount to pay in USD

\*/

void pay(double amount);

}

#### **2 Concrete strategies**

package payment.strategy;

/\*\* Pays via credit-card network. \*/

public class CreditCardPayment implements PaymentStrategy {

private final String cardNumber;

public CreditCardPayment(String cardNumber){ this.cardNumber = cardNumber; }

@Override public void pay(double amount){

System.out.printf("Charged $%.2f to card %s%n", amount, cardNumber);

}

}

package payment.strategy;

/\*\* Pays via PayPal wallet. \*/

public class PayPalPayment implements PaymentStrategy {

private final String email;

public PayPalPayment(String email){ this.email = email; }

@Override public void pay(double amount){

System.out.printf("Paid $%.2f using PayPal account %s%n", amount, email);

}

}

package payment.strategy;

/\*\* Pays via Bitcoin address. \*/

public class BitcoinPayment implements PaymentStrategy {

private final String walletAddress;

public BitcoinPayment(String walletAddress){ this.walletAddress = walletAddress; }

@Override public void pay(double amount){

System.out.printf("Transferred $%.2f (equiv) from BTC wallet %s%n", amount, walletAddress);

}

}

#### **3 PaymentContext.java**

package payment.context;

import payment.strategy.PaymentStrategy;

/\*\*

\* Context that delegates payment processing to a pluggable {@link PaymentStrategy}.

\*/

public class PaymentContext {

private PaymentStrategy strategy;

public void setPaymentStrategy(PaymentStrategy s){ this.strategy = s; }

/\*\*

\* Processes a charge by delegating to the configured strategy.

\* @throws IllegalStateException if no strategy has been selected

\*/

public void checkout(double amount){

if(strategy == null) throw new IllegalStateException("No payment method selected");

strategy.pay(amount);

}

}

#### **4 StrategyDemo.java**

package payment;

import payment.context.PaymentContext;

import payment.strategy.\*;

public class StrategyDemo {

public static void main(String[] args){

PaymentContext ctx = new PaymentContext();

ctx.setPaymentStrategy(new CreditCardPayment("1234-5678-9876-5432"));

ctx.checkout(99.99);

ctx.setPaymentStrategy(new PayPalPayment("user@example.com"));

ctx.checkout(149.50);

ctx.setPaymentStrategy(new BitcoinPayment("1BitcoinAddrXYZ"));

ctx.checkout(200.00);

}

}

Console

Charged $99.99 to card 1234-5678-9876-5432

Paid $149.50 using PayPal account user@example.com

Transferred $200.00 (equiv) from BTC wallet 1BitcoinAddrXYZ

#### **5 JUnit-style tests (sketch)**

/\* CreditCardTest.java \*/

PaymentContext pc = new PaymentContext();

pc.setPaymentStrategy(new CreditCardPayment("0000"));

pc.checkout(10.0); // should print card charge

/\* NullStrategyTest.java \*/

assertThrows(IllegalStateException.class, () -> new PaymentContext().checkout(5.0));

## **reflection.md**

The Strategy pattern moves algorithmic variety out of PaymentContext, abolishing switch noise and granting:

* **Runtime flexibility** — front-end selects a gateway on the fly (Open/Closed Principle).
* **Isolated tests** — each payment class can be mocked or unit-tested alone.
* **Easy extension** — add ApplePayPayment without touching existing code.

Drawbacks:

* **Class proliferation** — one extra file per algorithm.
* **Client awareness** — caller must know which strategy fits a scenario.
* **Minor indirection cost** — trivial for I/O-bound payments.

For systems with multiple interchangeable algorithms — payments, compression, routing, AI heuristics — Strategy offers clean, modular dynamism.