Payment System with Strategy Pattern

public class StrategyPatternExample {

public static void main(String[] args) {

// Create payment context

PaymentContext paymentContext = new PaymentContext();

// Process different payment methods

System.out.println("=== Processing Payments ===");

// Credit Card payment

paymentContext.setPaymentStrategy(new CreditCardPayment("1234-5678-9012-3456", "John Doe", "12/25", "123"));

paymentContext.processPayment(100.50);

// PayPal payment

paymentContext.setPaymentStrategy(new PayPalPayment("user@example.com", "password123"));

paymentContext.processPayment(55.75);

// Crypto payment

paymentContext.setPaymentStrategy(new CryptoPayment("1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa"));

paymentContext.processPayment(250.00);

// Bank Transfer payment

paymentContext.setPaymentStrategy(new BankTransferPayment("US123456789", "987654321"));

paymentContext.processPayment(500.25);

}

// Strategy Interface

interface PaymentStrategy {

void pay(double amount);

}

// Concrete Strategies

static class CreditCardPayment implements PaymentStrategy {

private String cardNumber;

private String name;

private String expiryDate;

private String cvv;

public CreditCardPayment(String cardNumber, String name, String expiryDate, String cvv) {

this.cardNumber = cardNumber;

this.name = name;

this.expiryDate = expiryDate;

this.cvv = cvv;

}

@Override

public void pay(double amount) {

System.out.printf("Processing credit card payment of $%.2f\n", amount);

System.out.printf("Card: %s, Name: %s, Exp: %s, CVV: %s\n",

maskCardNumber(cardNumber), name, expiryDate, cvv);

}

private String maskCardNumber(String cardNumber) {

return "\*\*\*\*-\*\*\*\*-\*\*\*\*-" + cardNumber.substring(cardNumber.length() - 4);

}

}

static class PayPalPayment implements PaymentStrategy {

private String email;

private String password;

public PayPalPayment(String email, String password) {

this.email = email;

this.password = password;

}

@Override

public void pay(double amount) {

System.out.printf("Processing PayPal payment of $%.2f\n", amount);

System.out.printf("Email: %s\n", email);

}

}

static class CryptoPayment implements PaymentStrategy {

private String walletAddress;

public CryptoPayment(String walletAddress) {

this.walletAddress = walletAddress;

}

@Override

public void pay(double amount) {

System.out.printf("Processing cryptocurrency payment of $%.2f\n", amount);

System.out.printf("Wallet: %s\n", walletAddress);

}

}

static class BankTransferPayment implements PaymentStrategy {

private String fromAccount;

private String toAccount;

public BankTransferPayment(String fromAccount, String toAccount) {

this.fromAccount = fromAccount;

this.toAccount = toAccount;

}

@Override

public void pay(double amount) {

System.out.printf("Processing bank transfer of $%.2f\n", amount);

System.out.printf("From: %s, To: %s\n", fromAccount, toAccount);

}

}

// Context Class

static class PaymentContext {

private PaymentStrategy paymentStrategy;

public void setPaymentStrategy(PaymentStrategy paymentStrategy) {

this.paymentStrategy = paymentStrategy;

}

public void processPayment(double amount) {

if (paymentStrategy == null) {

throw new IllegalStateException("Payment strategy not set");

}

System.out.println("\nStarting payment processing...");

paymentStrategy.pay(amount);

System.out.println("Payment processed successfully!");

}

}

}

Key Features:

1. **Interchangeable Algorithms**: Payment methods can be swapped at runtime
2. **Single Responsibility**: Each strategy handles one payment method
3. **Open/Closed Principle**: New payment methods can be added without changing existing code
4. **Clean Client Code**: Payment processing is consistent regardless of method
5. **Encapsulation**: Payment details are hidden within each strategy

Strategy Pattern Benefits:

1. **Flexibility**: Payment methods can be changed at runtime
2. **Eliminates Conditionals**: No need for large if-else or switch statements
3. **Testability**: Each strategy can be tested in isolation
4. **Separation of Concerns**: Payment logic is separate from context
5. **Extensibility**: Easy to add new payment methods

Output:

