Lab 07: Strategy

Áp dụng mẫu thiết kế Strategy trong phát triển phần mềm

1. Giới thiệu bài toán

Bạn đang phát triển một ứng dụng thương mại điện tử cho một cửa hàng trực tuyến. Ứng dụng này cần hỗ trợ nhiều phương thức thanh toán khác nhau, chẳng hạn như thanh toán qua thẻ tín dụng, thanh toán qua PayPal và thanh toán qua chuyển khoản ngân hàng. Mỗi phương thức thanh toán có một cách xử lý và tính toán phí giao dịch khác nhau.
Nhiệm vụ của bạn là áp dụng mẫu thiết kế Strategy để xây dựng một hệ thống linh hoạt cho phép thêm hoặc thay đổi các phương thức thanh toán mà không cần thay đổi mã nguồn chính của ứng dụng.

2. Yêu cầu thiết kế và cài đặt

- Định Nghĩa Interface Strategy: Tạo một interface `PaymentStrategy` với một phương thức `processPayment(double amount)` để xử lý thanh toán.
- Cài Đặt Các Chiến Lược Thanh Toán:
    - Thẻ Tín Dụng: Cài đặt một lớp `CreditCardPayment` triển khai `PaymentStrategy`. Tính toán phí thanh toán và xử lý thanh toán bằng thẻ tín dụng.
    - PayPal: Cài đặt một lớp `PayPalPayment` triển khai `PaymentStrategy`. Tính toán phí thanh toán và xử lý thanh toán qua PayPal.
    - Chuyển Khoản Ngân Hàng: Cài đặt một lớp `BankTransferPayment` triển khai `PaymentStrategy`. Tính toán phí thanh toán và xử lý thanh toán qua chuyển khoản ngân hàng.
- Tạo Context: Tạo một lớp `PaymentContext` để lưu trữ một đối tượng `PaymentStrategy` và cung cấp phương thức `executePayment(double amount)` để thực hiện thanh toán.
- Tạo Lớp Test: Tạo một lớp `PaymentTest` để kiểm tra các phương thức thanh toán khác nhau. Trong lớp này, bạn nên tạo các đối tượng `PaymentContext` với các chiến lược thanh toán khác nhau và thực hiện thanh toán với các phương thức này.

3. Gợi ý hướng dẫn giải quyết vấn đề

Bước 1: Tạo interface `PaymentStrategy` với phương thức `processPayment(double amount)`.

```
public interface PaymentStrategy {
	void processPayment(double amount);
}
```

Bước 2: Cài đặt các lớp cụ thể cho các chiến lược thanh toán.

```
public class CreditCardPayment implements PaymentStrategy {
	@Override
```

```
	public void processPayment(double amount) {
		System.out.println("Processing credit card payment of $" + amount);
	}
}

public class PayPalPayment implements PaymentStrategy {
	@Override
	public void processPayment(double amount) {
		System.out.println("Processing PayPal payment of $" + amount);
	}
}

public class BankTransferPayment implements PaymentStrategy {
	@Override
	public void processPayment(double amount) {
		System.out.println("Processing bank transfer payment of $" + amount);
	}
}
```

Bước 3: Tạo lớp `PaymentContext` để quản lý chiến lược thanh toán.

```
public class PaymentContext {
	private PaymentStrategy paymentStrategy;
	public PaymentContext(PaymentStrategy paymentStrategy) {
		this.paymentStrategy = paymentStrategy;
	}
	public void executePayment(double amount) {
		paymentStrategy.processPayment(amount);
	}
}
```

Bước 4: Tạo lớp `PaymentTest` để thử nghiệm.

```
public class PaymentTest {
	public static void main(String[] args) {
		PaymentContext context;
		context = new PaymentContext(new CreditCardPayment());
		context.executePayment(100.0);
		context = new PaymentContext(new PayPalPayment());
		context.executePayment(200.0);
		context = new PaymentContext(new BankTransferPayment());
		context.executePayment(300.0);
	}
}
```

4. Kết quả cần đạt

- Người học phải triển khai đầy đủ các lớp theo yêu cầu.
- Người học phải có thể thay đổi và mở rộng các chiến lược thanh toán một cách dễ dàng mà không làm ảnh hưởng đến mã nguồn chính của ứng dụng.
- Kết quả của bài kiểm tra phải hiển thị thông tin thanh toán chính xác cho từng chiến lược thanh toán.

5. Hướng phát triển mở rộng bài toán

- Thêm Các Chiến Lược Thanh Toán Mới: người học có thể mở rộng hệ thống bằng cách thêm các phương thức thanh toán mới như thanh toán qua ví điện tử, thanh toán qua tiền mã hóa, v.v.
- Tính Năng Xác Thực: Thêm tính năng xác thực và bảo mật cho các phương thức thanh toán.
- Tính Toán Phí Giao Dịch: Cập nhật các lớp thanh toán để tính toán phí giao dịch dựa trên các điều kiện khác nhau.

6. Hướng dẫn đánh giá

- Sự Chính Xác: Kiểm tra xem các chiến lược thanh toán có được triển khai chính xác và có hoạt động đúng như mong đợi không.
- Khả Năng Mở Rộng: Đánh giá khả năng mở rộng của hệ thống khi thêm các chiến lược thanh toán mới.
- Chất Lượng Mã Nguồn: Kiểm tra chất lượng mã nguồn, bao gồm cách tổ chức và cấu trúc mã, cũng như việc sử dụng các nguyên tắc của mẫu thiết kế Strategy.