## Exercise 5: Implementing the Decorator Pattern

### Step 1: Create a New Java Project

Create a new Java project named DecoratorPatternExample using your favorite IDE (e.g., IntelliJ IDEA, Eclipse).

### Step 2: Define Component Interface

Create an interface Notifier with a method send().

**Code:**

public interface Notifier {

void send(String message);

}

### Step 3: Implement Concrete Component

Create a class EmailNotifier that implements Notifier.

// EmailNotifier.java

public class EmailNotifier implements Notifier {

@Override

public void send(String message) {

System.out.println("Sending Email: " + message);

}

}

### Step 4: Implement Decorator Classes

Create an abstract decorator class NotifierDecorator that implements Notifier and holds a reference to a Notifier object.

// NotifierDecorator.java

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

@Override

public void send(String message) {

wrappedNotifier.send(message);

}

}

Create concrete decorator classes like SMSNotifierDecorator and SlackNotifierDecorator that extend NotifierDecorator.

// SMSNotifierDecorator.java

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

sendSMS(message);

}

private void sendSMS(String message) {

System.out.println("Sending SMS: " + message);

}

}

// SlackNotifierDecorator.java

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

sendSlackMessage(message);

}

private void sendSlackMessage(String message) {

System.out.println("Sending Slack message: " + message);

}

}

### Step 5: Test the Decorator Implementation

Create a test class to demonstrate sending notifications via multiple channels using decorators.

// DecoratorPatternTest.java

public class DecoratorPatternTest {

public static void main(String[] args) {

Notifier emailNotifier = new EmailNotifier();

Notifier smsNotifier = new SMSNotifierDecorator(emailNotifier);

Notifier slackNotifier = new SlackNotifierDecorator(smsNotifier);

// Sending notifications via Email, SMS, and Slack

slackNotifier.send("Hello, this is a test message!");

}

}

### Explanation:

1. **EmailNotifier** is the concrete implementation of the Notifier interface.
2. **NotifierDecorator** is the abstract decorator class that implements the Notifier interface and holds a reference to a Notifier object.
3. **SMSNotifierDecorator** and **SlackNotifierDecorator** extend NotifierDecorator and add their own behavior (sending SMS and Slack messages, respectively) while still calling the send method of the wrapped notifier.
4. **DecoratorPatternTest** demonstrates the use of decorators to send notifications via multiple channels.

When you run the DecoratorPatternTest class, you should see the following output:

Sending Email: Hello, this is a test message!

Sending SMS: Hello, this is a test message!

Sending Slack message: Hello, this is a test message!