**DESIGN PATTERNS AND PRINCIPLES**

**Exercise 5: Implementing the Decorator Pattern**

**Scenario:**

You are developing a notification system where notifications can be sent via multiple channels (e.g., Email, SMS). Use the Decorator Pattern to add functionalities dynamically.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named DecoratorPatternExample.
2. **Define Component Interface:**
   * Create an interface Notifier with a method send().
3. **Implement Concrete Component:**
   * Create a class EmailNotifier that implements Notifier.
4. **Implement Decorator Classes:**
   * Create abstract decorator class NotifierDecorator that implements Notifier and holds a reference to a Notifier object.
   * Create concrete decorator classes like SMSNotifierDecorator, SlackNotifierDecorator that extend NotifierDecorator.
5. **Test the Decorator Implementation:**
   * Create a test class to demonstrate sending notifications via multiple channels using decorators.

**DESCRIPTION:**

This program uses the Decorator Pattern to send notifications through multiple channels like Email, SMS, and Slack.

* Notifier is the main interface with a send() method.
* EmailNotifier is the basic notifier.
* NotifierDecorator is an abstract class that wraps a Notifier.
* SMSNotifierDecorator and SlackNotifierDecorator add extra notification methods.

The DecoratorPatternExample class shows how you can easily combine these decorators to send messages through multiple channels.

**PROGRAM:**

public class DecoratorPatternExample {

public static void main(String[] args) {

Notifier emailOnly = new EmailNotifier();

emailOnly.send("Welcome!");

Notifier emailAndSMS = new SMSNotifierDecorator(new EmailNotifier());

emailAndSMS.send("Your OTP is 1234");

Notifier allChannels = new SlackNotifierDecorator(new SMSNotifierDecorator(new EmailNotifier()));

allChannels.send("System Alert: High CPU Usage");

}

}

interface Notifier {

void send(String message);

}

class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

abstract class NotifierDecorator implements Notifier {

protected Notifier wrappee;

public NotifierDecorator(Notifier notifier) {

this.wrappee = notifier;

}

public void send(String message) {

wrappee.send(message);

}

}

class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

**OUTPUT:**

****