**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.

**CODE:**

**Step 1:** Create a New Java Project

Create a new Java project named `DecoratorPattern`.

**Step 2:** Define Component Interface

//Notifier

interface Notifier {

void send();

}

**Step 3:** Implement Concrete Component

//EmailNotifier

class EmailNotifier implements Notifier {

private String email;

private String username;

public EmailNotifier(String email, String username) {x

this.email = email;

this.username = username;

}

@Override

public void send() {

System.out.println("The email was sent to the user " + username + " having email " + email + "!");

//System.out.println();

}

}

**Step 4:** Implement Decorator Classes

//Abstract Decorator Class

abstract class NotifierDecorator implements Notifier {

protected Notifier notifier;

public NotifierDecorator(Notifier notifier) {

this.notifier = notifier;

}

@Override

public void send() {

notifier.send();

}

}

//Concrete Decorator Classes

class SMSNotifierDecorator extends NotifierDecorator {

private String username;

private String phone;

public SMSNotifierDecorator(Notifier notifier, String username, String phone) {

super(notifier);

this.username = username;

this.phone = phone;

}

@Override

public void send() {

System.out.println();

super.send(); // Call the original notifier's send method

System.out.println("The SMS was sent to the user " + username + " whose phone number is " + phone + "!");

System.out.println();

}

}

class SlackNotifierDecorator extends NotifierDecorator {

private String username;

private String userId;

public SlackNotifierDecorator(Notifier notifier, String username, String userId) {

super(notifier);

this.username = username;

this.userId = userId;

}

@Override

public void send() {

super.send(); // Call the original notifier's send method

System.out.println("User Id " + userId + " received a Slack notification!");

System.out.println();

}

}

**Step 5:** Test the Decorator Implementation

public class DecoratorPattern {

public static void main(String[] args) {

Notifier emailNotifier = new EmailNotifier("DrakeRamaroy@gmail.com", "Drake");

Notifier smsNotifier = new SMSNotifierDecorator(emailNotifier, "Drake Ramaroy", "8976785674");

Notifier slackNotifier = new SlackNotifierDecorator(emailNotifier, "Drake Ramaroy", "12345");

emailNotifier.send();

smsNotifier.send();

slackNotifier.send();

}

}

**SAMPLE OUTPUT:**

