**DESIGN PATTERNS AND PRINCIPLES**

**Exercise 5: Decorator Pattern**

**Step 1: Interface – Notifier.java**

public interface Notifier {

void send(String message);

}

**Step 2: Concrete Component – EmailNotifier.java**

public class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

Step 3: Abstract Decorator – **NotifierDecorator**.java

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappee;

public NotifierDecorator(Notifier notifier) {

this.wrappee = notifier;

}

public void send(String message) {

wrappee.send(message);

}

}

**Step 4: Concrete Decorators:**

**// File: SMSNotifierDecorator.java**

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

**// File: SlackNotifierDecorator.java**

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

**Step 5: Test Class – TestDecoratorPattern.java**

public class TestDecoratorPattern {

public static void main(String[] args) {

// Basic Email Notification

Notifier email = new EmailNotifier();

// Add SMS Notification on top of Email

Notifier emailWithSMS = new SMSNotifierDecorator(email);

// Add Slack Notification on top of SMS + Email

Notifier fullNotifier = new SlackNotifierDecorator(emailWithSMS);

// Send the notification via all three channels

fullNotifier.send("System maintenance at 10 PM tonight.");

}

}

**Output:**

