**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.

package Decorator;

interface Notifier {

void send(String msg);

}

class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

abstract class NotifierDecorator implements Notifier {

protected Notifier notifier;

public NotifierDecorator(Notifier notifier) {

this.notifier = notifier;

}

public void send(String msg) {

notifier.send(msg);

}

}

class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

private void sendSMS(String message) {

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

}

public void send(String message) {

super.send(message);

sendSMS(message);

}

}

class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

private void sendSlackMessage(String message) {

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

}

public void send(String message) {

super.send(message);

sendSlackMessage(message);

}

}

public class Decoratorpattern {

public static void main(String[] args) {

Notifier basicEmail = new EmailNotifier();

System.***out***.println("\nEmail only");

basicEmail.send("Server is down");

Notifier emailWithSMS = new SMSNotifierDecorator(basicEmail);

System.***out***.println("email and sms");

emailWithSMS.send("High memory usage detected.");

Notifier emailSMSAndSlack = new SlackNotifierDecorator(emailWithSMS);

System.***out***.println("email,sm,slc");

emailSMSAndSlack.send("space is almost full.");

}

}

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.