**Exercise 5: Implementing the Decorator Pattern**

public class DecoratorPatternExample {

interface Notifier {

void send(String message);

}

static class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

static abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

public void send(String message) {

wrappedNotifier.send(message);

}

}

static class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

static class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

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

}

}

public static void main(String[] args) {

Notifier basicEmail = new EmailNotifier();

Notifier emailWithSMS = new SMSNotifierDecorator(basicEmail);

Notifier emailWithSMSAndSlack = new SlackNotifierDecorator(emailWithSMS);

System.out.println("Sending Email only:");

basicEmail.send("System update scheduled at 10 PM.");

System.out.println("\nSending Email + SMS:");

emailWithSMS.send("System update scheduled at 10 PM.");

System.out.println("\nSending Email + SMS + Slack:");

emailWithSMSAndSlack.send("System update scheduled at 10 PM.");

}

}

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.