Design patterns and principles

SuperSet ID:6412063

Exercise 5: Implementing the Decorator Pattern

Code:

public interface Notifier {

void send(String message);

}

public class EmailNotifier implements Notifier {

public void send(String message) {

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

}

}

public abstract class NotifierDecorator implements Notifier {

protected Notifier wrappedNotifier;

public NotifierDecorator(Notifier notifier) {

this.wrappedNotifier = notifier;

}

public void send(String message) {

wrappedNotifier.send(message); // delegate to wrapped notifier

}

}

public class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message); // send via base notifier

sendSMS(message); // add SMS

}

private void sendSMS(String message) {

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

}

}

public class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message); // send via base notifier

sendSlack(message); // add Slack

}

private void sendSlack(String message) {

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

}

}

public class NotificationTest {

public static void main(String[] args) {

Notifier notifier = new EmailNotifier();

notifier = new SMSNotifierDecorator(notifier);

notifier = new SlackNotifierDecorator(notifier);

notifier.send("Server down alert!");

}

}

Output:

A close-up of a text

Description automatically generated