**Exercise 5: Implementing the Decorator Pattern**

public class Main {

interface Notifier {

void send(String message);

}

static class EmailNotifier implements Notifier {

@Override

public void send(String message) {

System.out.println("Sending 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);

}

@Override

public void send(String message) {

super.send(message);

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

}

}

static class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

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

}

}

public static void main(String[] args) {

Notifier baseNotifier = new EmailNotifier();

Notifier smsNotifier = new SMSNotifierDecorator(baseNotifier);

Notifier fullNotifier = new SlackNotifierDecorator(smsNotifier);

fullNotifier.send("Your order has been shipped!");

}

}

