**Exercise 5: Implementing the Decorator Pattern**

CODE:

public class DecoratorPatternExample {

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 wrappee;

public NotifierDecorator(Notifier notifier) {

this.wrappee = notifier;

}

@Override

public void send(String message) {

wrappee.send(message);

}

}

static class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

@Override

public void send(String message) {

super.send(message);

sendSMS(message);

}

private void sendSMS(String 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);

sendSlack(message);

}

private void sendSlack(String message) {

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

}

}

public static void main(String[] args) {

Notifier emailOnly = new EmailNotifier();

emailOnly.send("Server is up!");

Notifier emailSMS = new SMSNotifierDecorator(new EmailNotifier());

emailSMS.send("Low disk space alert!");

Notifier fullNotify = new SlackNotifierDecorator(

new SMSNotifierDecorator(

new EmailNotifier()));

fullNotify.send("Application crashed!");

}

}

OUTPUT:

