**EXERCISE 6:**

**DECORATOR PATTERN EXAMPLE**

interface Notifier {  
 void send(String message);  
}  
  
class EmailNotifier implements Notifier {  
 @Override  
 public void send(String message) {  
 System.*out*.println("Sending Email: " + message);  
 }  
}  
abstract class NotifierDecorator implements Notifier {  
 protected Notifier wrappedNotifier;  
  
 public NotifierDecorator(Notifier notifier) {  
 this.wrappedNotifier = notifier;  
 }  
  
 @Override  
 public void send(String message) {  
 wrappedNotifier.send(message);  
 }  
}  
  
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);  
 }  
}  
  
class SlackNotifierDecorator extends NotifierDecorator {  
 public SlackNotifierDecorator(Notifier notifier) {  
 super(notifier);  
 }  
  
 @Override  
 public void send(String message) {  
 super.send(message);  
 sendSlackMessage(message);  
 }  
  
 private void sendSlackMessage(String message) {  
 System.*out*.println("Sending Slack message: " + message);  
 }  
}  
  
public class DecoratorPatternExample {  
 public static void main(String[] args) {  
 Notifier emailNotifier = new EmailNotifier();  
 Notifier smsNotifier = new SMSNotifierDecorator(emailNotifier);  
 Notifier slackNotifier = new SlackNotifierDecorator(smsNotifier);  
 slackNotifier.send("This is a test message.");  
 Notifier emailAndSmsNotifier = new SMSNotifierDecorator(new EmailNotifier());  
 emailAndSmsNotifier.send("This is another test message.");  
 }  
}