我将带大家一起去看一看在面向对象编程中出现的23大设计模式和9大设计原则。设计原则是我们开发研究设计模式的目的，而设计模式则是达到设计原则的手段和方法。在记录笔记过程中，会在不同的设计模式中穿插讲解设计原则。
本笔记知识点主要来自于《Head First 设计模式》

行为型模式二：观察者模式（Observer Pattern）

观察者模式定义：
观察者模式定义了对象之间一对多的依赖模式，这样当对象的状态，包括数据更新发生改变时，所有依赖者都会根据这个状态的变化而实时更新自身状态
从上面的定义中我们可以看到，观察者模式有两个条件：

- 它定义了一系列对象之间的依赖关系
- 当一个对象的状态进行改变时，它的依赖者都会收到消息并且更新

设计原则：

- **尽量保持交互对象之间的松耦合(loose coupling)**

观察者模式的组成：

- 被观察者也即主题的接口，在这个接口中定义了注册观察者，撤销观察者，通知观察者的方法；
- 观察者接口，在这接口中定义了一些观察者所共同需要的方法；
- 具体的被观察者实现类，这些类实现了被观察者接口，并且在定义中含有一个观察者的List实例变量用来注册观察者；
- 具体的观察者实现类，这些类根据自己的功能实现观察者接口，然后给出自己的实现；

看下面一个代码，该代码来自于博客：

```
1. //被观察者
2. public interface Watched {
3.     public void addWatcher(Watcher watcher);
4.     public void deleteWatcher(Watcher watcher);
5.     public void notifyWatcher();
6. }
7. //观察者
8. public interface Watcher {
9.     public void display();
10. }
11. //具体被观察者实现类
12. public class concreteWatched implements Watched {
13.     ArrayList list;
14.     public concreteWatched() {
15.         list = new ArrayList();
16.     }
17.     public void addWatcher(Watcher watcher) {
18.         list.add(watcher);
19.     }
20.     public void deleteWatcher(Watcher watcher) {
21.         int index = list.indexOf(watcher);
22.         if (index >= 0) list.remove(index);
23.     }
24.     public void notifyWatcher() {
25.         for (Watcher watch : list) {
26.             watch.display();
27.         }
28.     }
29. }
30. //具体的观察者实现类
31. public class concreteWatcher implements Watcher {
32.     Watched watched;
33.     public concreteWatcher(Watched watched) {
34.         this.watched = watched;
35.         watched.add(this);
36.     }
37.     public void display() {
38.         System.out.println("This is watcher number 1.");
39.     }
40. }
41. 这个例子给出了一个基本的观察者模式的实现形式。可以看出来，每一个观察者具体实现类的构造函数都传入一个被观察者对象，这个步骤就是注册过程。当然我们也可以换种方式，不穿入任何参数，调用被观察者的add方法也能注册成功。
```

值得注意的是，在观察者模式中，当被观察者的数据或者状态发生改变时，被观察者与观察者之间的数据通信方式有两种：

- **推：**用这种方式的话，被观察者不仅仅会通知观察者我的数据或者状态已经改变，而且还会顺带将改变后的数据或者状态一并发送给观察者，至于观察者是否利用这些数据则不是被观察者所关心的；
- **拉：**用这种方式，被观察者只是通知观察者自身的数据或者状态已经改变，至于观察者需要哪些数据或者状态，则需要观察者自己本身来进行拉取。这也是为什么大多数的观察者实现类中往往有一个被观察者对象实例，通常都是为了拉取数据而用的

在Java自定义的库中，有自带的观察者模式类和接口。来简要看看：

- Observable类：这是一个类，并非接口；继承这个类的子类往往是作为被观察者的，在这个超类中，已经定义注册，撤销，通知等一系列方法；
- Observer接口：这是一个接口；实现这个接口的类往往是作为观察者的。
  因为Observable是一个超类，而且Java不支持多重继承，因此往往Observable不那么具有扩展性。