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》

结构型模式之二、三：适配器模式(Adapter Pattern)、外观模式(Facade Pattern)

1:适配器模式

定义：
将一个类的接口转换为客户希望的另一个接口，使得不同接口之间可以合作无间而且使得客户察觉不到其中的区别

组成：

- 目标接口(Target)：这是客户所希望看见的目标对象
- 需要适配的类：(Adpatee)这是被适配器适配为目标类的类
- 适配器(Adapter): 这是适配器，它完成从被适配类到客户需要对象的过程

实现方式：
现在存在三个类：目标类，被适配类以及适配类。我们的任务是完成从被适配类到目标类的转变。因此适配器类必须实现目标类，里面含有一个被适配类的成员变量(**对象适配器**)，然后在目标类需要的方法里面实际上调用的是被适配类的方法，这样就完成了转变，而客户并不知道这其中的变化，她只是需要一个目标类的实例对象而已。

根据上面的解释，在实际操作中，适配器模式有两种实现方式：

- 类适配器模式
  类适配器模式采用的是继承方式来进行适配，适配器类继承Adpatee类并且实现Target类，然后在Target类的方法中其实调用Adaptee的方法；
- 对象适配器模式
  这是另外一个实现方式，适配器类只是实现了Target类。在适配器类中含有一个Adaptee的成员变量，然后初始化适配器时初始化这个成员变量。然后和上面一样，在Target类的方法中其实调用的是Adaptee的方法。

看具体的代码实现：

```
1. //先是目标类，这里我们选择Duck
2. public interface Duck {
3.     public void quack();
4.     public void fly();
5. }
6. 
7. //然后是被适配器类，这里选择Turkey
8. public class Turkey {
9.     public void goodle() {
10.         System.out.println("Goodle...");
11.     }
12.     public void fly() {
13.         System.out.println("I can't fly long");
14.     }
15. }
16. 
17. //最后是适配器,对象适配器方式
18. public class Adapter implements Duck {
19.     Turkey turkey;
20.     public Adapter(Turkey turkey) {
21.         this.turkey = turkey;
22.     }
23.     public void quack() {
24.         turkey.goodle();
25.     }
26.     public void fly() {
27.         turkey.fly();
28.     }
29. }
30. //最后是适配器,类适配器方式
31. public class Adapter extends Turkey implements Duck {
32.     public void quack() {
33.         super.goodle();
34.     }
35.     public void fly() {
36.         super.fly();
37.     }
38. }
```

2: 外观模式(Facade Pattern)

定义：
外观模式定义了一个统一的高层接口，使得这些接口方法中整合利用那些小接口的方法，使得暴露给客户的是一个简单明朗的操作。

新的设计原则：

- 最少知识原则(Least Knowledge)：又被称做Law of Demeter，只与直接的朋友通信。
  这个原则表明了在设计过程中，尽量不要使得太多的类耦合在一起

组成：

- 外观角色(Facade)：暴露给客户的简洁的接口或者类
- 子系统角色：外观角色所调用的一系列的子类，由这些子类的方法组合成一个大操作；

目前为止学到的三种结构型模式：***装饰者模式，适配器模式，外观模式***,
装饰者模式目的在于继承类型而非方法，给予原来对象以新的责任；适配器模式将一个接口转化为另一个客户需求的接口；外观模式简化众多子接口，使得暴露给客户的是一个简洁的接口。

看代码：

```
1. public class Operation1 {
2.     public void operation() {
3.         System.out.println("This is operation1 is operating.");
4.     }
5. }
6. public class Operation2 {
7.     public void operation() {
8.         System.out.println("This is operation2 is operating.");
9.     }
10. }
11. public class Operation3 {
12.     public void operation() {
13.         System.out.println("This is operation3 is operating.");
14.     }
15. }
16. //Facade
17. public class Facade {
18.     Operation1 operation1;
19.     Operation2 operation2;
20.     Operation3 operation3;
21. 
22.     public Facade(Operation1 operation1,Operation2 operation2,Oper
    ation3 operation3) {
23.         this.operation1 = operation1;
24.         this.operation2 = operation2;
25.         this.operation3 = operation3;
26.     }
27.     public void operation() {
28.         operation1.operation();
29.         operation2.operation();
30.         operation3.operation();
31.     }
32. }
33. 外观模式很简单明了，在平时的编程过程中也经常用到，只是我们并没有注意到这是一种模
    式而已。
```