# 深入浅出设计模式

## 设计模式入门

1. 模拟鸭子

使用继承并不能很好的解决问题，当鸭子的行为在子类中不断地发生变化时，让所有的子类都有这些行为是不恰当的。因为Java接口不具备实现代码的能力，所以继承接口达不到代码的复用，这意味着：无论何时你需要修改某个行为，你必须得往下追踪并在每一个定义此行为的类中修改它，一不小心，可能会造成新的错误。因此，有一个设计原则——

* 找出应用中可能需要变化之处，把他们独立出来，不要和那些不需要变化的代码混合在一起（把会变化的部分取出并“封装”起来，好让其他部分不会受到影响）

从现在开始，鸭子的行为将被放在分开的类中，此类专门提供某行为接口的实现，因此，鸭子类就不再需要知道行为的实现细节，由此，有一个设计原则——

·针对接口编程，而不是针对实现编程

所以这次鸭子类不会负责实现Flying 与 Quack，反而由我们制造出的一组其他类专门实现FlyBehavior 与 QuackBehavior，这就称为“行为”类。由行为类而不是Duck类实现接口。

* 在Duck类里面声明FlyBehavior 和QuackBehavior接口，
* 利用public void setFlyBehavior（FlyBehavior）动态设置行为

设计原则——

* 多用组合，少用继承

以上设计模式即为策略模式。

* 策略模式定义了算法族，分别分装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户

1. 观察者模式
2. 气象站问题

* 观察者模式定义对象之间的一对多依赖，这样一来，当一个对象改变状态时，他的所有依赖者都会收到通知并自动更新

改变主题或观察者其中一方，并不会影响另一方。因为两者是松耦合的，只要他们之间的接口仍被遵守，我们就可以自由地改变他们。一个设计原则——

* 为了交互对象之间的松耦合设计而努力

松耦合的设计之所以让我们建立有弹性的OO系统，能够应对变化，是因为对象之间的相互依赖降到最低。

Java内置的观察者模式

要点：

1. 主题（可观察者）用一个共同的接口来更新观察者
2. 观察者和可观察者之间用松耦合方式结合，可观察者不知道观察者的细节，只知道观察者实现了观察者接口
3. 使用此模式时，你可从被观察者处推（push）或（pull）数据（然而，推的方式被认为更正确）
4. 有多个观察者时，不可以依赖特定的通知顺序
5. Java有多种观察者模式的实现，包括了通用的java.util.Observable
6. 要注意java.util.Observable实现上带来的一些问题
7. 如果有必要的化，可以实现自己的Observable，这并不难，不要害怕
8. 此模式也被应用在许多地方，如：JavaBean、RMI
9. 装饰者模式

设计原则——

* 类应该对扩展开发，对修改关闭
* 允许类容易扩展，在不修改现有代码的情况下，就可搭配新的行为。这样的设计具有弹性可以应对改变，可以接受新的功能来应对改变的需求。

装饰者:

* 装饰者和被装饰者对象有相同的超类型
* 你可以用一个或多个装饰者包装一个对象
* 既然装饰者和被装饰者对象有相同的超类型，所以在任何需要原始对象（被包装的）场合，可以用装饰过的对象代替它
* 装饰着可以在所委托被装饰者的行为之前与/或之后，加上自己的行为，以达到特定的目的
* 对象可以在任何时候被装饰，所以可以在运行时动态地、不限量地用你喜欢的装饰者来装饰对象
* 装饰者模式动态地将责任附加到对象，若要扩展功能，装饰者提供了比继承更有弹性的替代方向

1. 工厂模式

披萨问题

工厂方法：工厂方法是抽象的，所以依赖子类来处理对象的创建

工厂方法必须返回一个产品。超类中定义的方法，通常使用到工厂方法的返回值

工厂方法将客户（也就是超类中的代码，例如orderPizza（））和实际创建具体产品的代码分隔开来

工厂方法可能需要参数（也可能不需要）来指定所需要的产品

所有工厂方法都是用来封装对象的创建。工厂方法模式通过让子类决定该创建的对象是是什么，来达到将对象创建的过程封装的目的。

工厂方法模式定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个。工厂方法让类把实例化推迟到子类

177