# 设计模式

## 设计原则:

1.找出应用中可能变化之处,把它们独立出来,不要和那些不需要变化的代码混在一起.

2.针对接口编程,而不是针对实现编程

3.多用组合,少用继承.

4.为了交互对象之间的松耦合而努力.

## `````策略模式:

定义了算法族,分别封装起来,让它们之间可以相互替换,词模式让算法的变化独立于使用算法的客户.

--- 记住,知道抽象,继承,多态这些概念并不能马上让你变成好的面向对象设计者.设计大师关心的是建立弹性的设计,可以维护,可以应对变化.

OO基础:

抽象

封装

多态

继承

OO原则:

封装变化

多用组合,少用继承

针对接口编程,而不是针对实现编程

为交互对象之间的松耦合设计而努力 (松耦合设计更有弹性,更能应付变化)

开放-关闭原则,类应该对拓展开放,对修改关闭.

要依赖抽象,不要依赖具体类.

-----------------------------------------------------------------------------我是分割线-----------------------------------------------------------------------------

## `````观察者模式:

定义了对象之间的一对多依赖,这样一来,当一个对象状态改变时,它的所有依赖者都会收到通知并自动更新.

松耦合的威力:

当两个对象之间松耦合,它们依然可以交互,但是不太清楚彼此的细节.

观察者模式提供了一种对象设计,让主题和观察者之间松耦合.

(改变主题或观察者其中一方,并不影响另一方.因为两者是松耦合的,所以只要它们之间的接口仍被遵守,我们就可以自由地改变它们.)

要点:

1.观察者模式定义了对象之间一对多的关系.

2.主题(也可以是 可观察者),用一个共同的接口来更新观察者.

3.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.

4.使用此模式时,你可以从被观察者处 推(push)或 拉(pull)数据(然后 推(push)的方式被认为更"正确").

5.有多个观察者时不可以依赖特定的通知顺序.

6.java有多种观察者模式实现,包括了通用的java.util.Observable.

7.此模式被应用在许多地方,例如:JavaBeans,RMI.

-----------------------------------------------------------------------------我是分割线-----------------------------------------------------------------------------

## `````装饰模式:

动态地将职责附加到对象上,若要拓展功能,装饰者提供了比继承更有弹性的替代方案.

我们的目标是允许类容易扩展,在不修改现有代码的基础上,就可以搭配新的行为. 如果能实现这样的目标能有什么好处呢?

这样的设计更具有弹性,可以应对改变,可以接受新的功能 来应对改变的需求.

许多模式时长期经验的实证,可以提供扩展的方法免于被修改.

我们如何让设计的每个部分都遵循开放-关闭原则?

通常,办不到.要让OO设计同时具有开放性和关闭性,又不修改现有的代码,需要花费许多的时间和努力.一般来说我们是在没有闲工夫把设计的每个部分都这么设计(

而且就算做到了,也是一种浪费.

)遵循开放-关闭原则,通常会引入新的抽象层次,增加代码的复杂度.你需要把经历集中在设计中最有可能改变的地方,然后应用开放-关闭原则.

这里用到继承,重点在于装饰者和被装饰者必须是一样的类型,也就是有相同的超类. 这里利用继承达到了"类型匹配",是不是利用继承获得"行为".

如果依赖继承,那么类的行为只能在编译时静态决定,换句话说,行为如果不是来自超类,就是子类覆盖后的版本. 反之利用组合,可以把装饰者混合着用

....而且是在"运行时".

装饰者应该做的事,就是增加行为到被包装的对象上.

要点:

1.继承属于拓展形式之一,但不见得是达到弹性设计的最佳方式.

2.在我们的设计中,应该允许行为可以被拓展,无需修改现有代码.

3.组合和委托可用于在运行时动态的加上新的行为.

4.除了继承,装饰者模式也可以让我们拓展行为.

5.装饰者模式以为这一群装饰类,这些类用来包装具体组件.

6.装饰者类反应出被装饰的组件类型(事实上他们具有相同的类型,都是通过接口或继承实现).

7.装饰者可以在被装饰者的行为前面/或后面加上自己的行为,甚至将被装饰者的行为完全取代,而达到特定的目的.

8.你可以用无数个装饰者包装一个组件.

9.装饰者一般对组件的客户是透明的,除非客户程序依赖于组件的具体类型.

10.装饰者会导致设计中出现很多小对象,如果使用过度,会让程序变得复杂.

-----------------------------------------------------------------------------我是分割线-----------------------------------------------------------------------------

## 定义简单工厂:

简单工厂其实不是一个设计模式,反而比较像是一种编程习惯.

工厂方法用来处理对象的创建,并将这样的行为封装在子类中.这样,客户程序中关于超类的代码就和子类对象创建的代码解耦了.

1.工厂方法是抽象的,所以依赖子类来处理对象的创建.

2.工厂方法必须返回一个产品. 超类中定义的方法,通常使用工厂方法的返回值.

3.工厂方法将客户(也就是超类中的代码)和实际创建具体产品的代码分隔开来.

所有的工厂模式都是用来封装对象的创建. 工厂方法模式(Factory Method Pattern) 通过让子类决定应该创建的对象是什么,来达到将对象创建的过程封装的目的.

将 产品的 "实现" 从 "使用" 中解耦.

## `````工厂方法模式:

定义了创建对象的接口,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.

简单工厂 与 工厂方法模式的 差别

简单工厂把全部事情,在一个地方处理完, 然后工厂方法模式确实创建了一个框架,让子类决定如何实现.

依赖倒置原则:

很清楚地,代码里减少对具体类的依赖是件"好事",事实上有一个 OO设计原则就正式阐明了这一点 "依赖倒置原则"

\*\*\*\*\*\*\*要依赖抽象,不要依赖具体类.

下面的几个指导方针,能帮你避免在OO设计中违反 依赖倒置原则

1.变量不可以持有具体类的引用. (如果使用new ,就会持有具体类的引用. 你可以改用工厂来避免这种做法.)

2.不要让类派生子具体类. (如果派生自具体类,就需要依赖具体的类. 请派生自一个抽象(接口或者抽象类))

3.不要覆盖基类中已实现的方法. (如果覆盖基类中已实现的方法,那么你的基类就不是一个真正适合被继承的抽象类.基类中已实现的方法应该被所有子类共享.)

-----------------------------------------------------------------------------我是分割线-----------------------------------------------------------------------------

`````单件模式:

确保一个类只有一个实例,并提供一个全局的访问点

-----------------------------------------------------------------------------我是分割线-----------------------------------------------------------------------------

## `````命令模式:

将"请求"封装成对象,以便使用不同的请求,队列或者日志来参数化其他对象.命令模式也支持可撤销的操作.

这个模式 允许我们将动作封装成命令对象,这样一来我们就可以随心所欲的存储,传递,和调用它们.

==========要点:

1.命令模式将发出请求的对象和执行请求的对象解耦.

2.在被解耦的两者之间是通过命令对象进行沟通的,命令对象封装了接收者和一个或一组动作.

3.调用者通过调用命令对象的excute()方法发出请求,这会使得接收者的动作被调用.

4.调用者可以接受命令当参数,甚至在运行时动态的进行.

-----------------------------------------------------------------------------我是分割线-----------------------------------------------------------------------------

## `````适配器模式与外观模式:

将一个类的接口,转换成客户期望的另一个接口.适配器让原本接口不兼容的类可以合作无间.