# 设计模式6大原则：

开闭原则：

对扩展开发，对修改关闭(尽量在不修改源代码下进行扩展)

尽量修改调用者的代码，而不是修改提供者的代码

抽象层尽量保持稳定，减少修改

优点：可复用 + 可维护

里氏代换原则：

里氏代换原则是对“开-闭”原则的补充。实现“开-闭”原则的关键步骤就是抽象化。

就是多态，父类调用的子类的方法(重写的)而非父类方法

依赖倒置原则：开闭原则基础

程序要依赖于抽象接口，不要依赖于具体实现。

针对接口编程而不是针对实现编程

简单的说就是要求对抽象进行编程，不要对实现进行编程，这样就降低了客户与实现模块间的耦合。

底层类采用抽象类或接口

方法传入的参数最好是接口后抽象类，方便后期维护

接口隔离原则(角色隔离原则)：

使用多个专门的接口，而不使用单一的总接口，即客户端不应该依赖那些它不需要的接口。

迪米特法则：

一个软件实体应当尽可能少地与其他实体发生相互作用。

使类与类之间保持松散的耦合关系，其中一个类修改时，减少其他类的影响

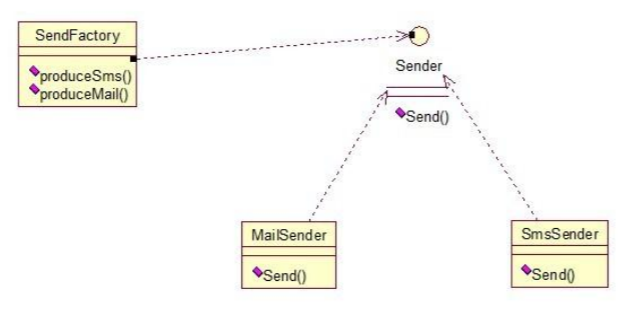
可以通过引入第三者来降低对象之间的耦合程度

合成复用原则：

一个对象聚合(调用)另外一个以存的对象[在类声明另外一个类]，而不是采用继承的方式实现

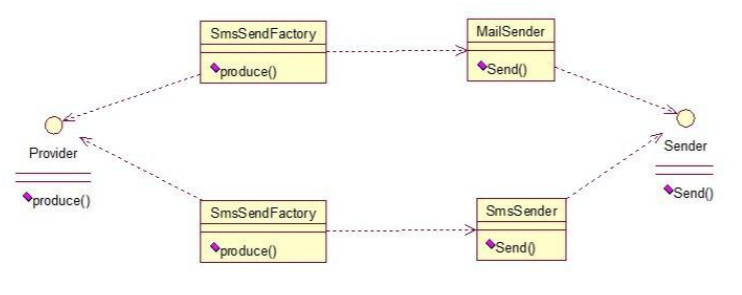
# 设计模式：

## 01工厂模式（Factory）：



普通工厂

工厂返回的Send，但是当引入新Send子类时，会修改SendFactory



抽象工厂

Provider为工厂，为具体的创建具体的工厂方法，有新类时，创建新的provider

## 02单例模式（Singleton）

懒汉/恶汉模式

双重检查 + volatile

## 03建造者模式（Builder）

和工厂模式类似，只是工厂模式是创建一个对象，建造者是批量创建多个同类型对象

## 04原型模式（Prototype）

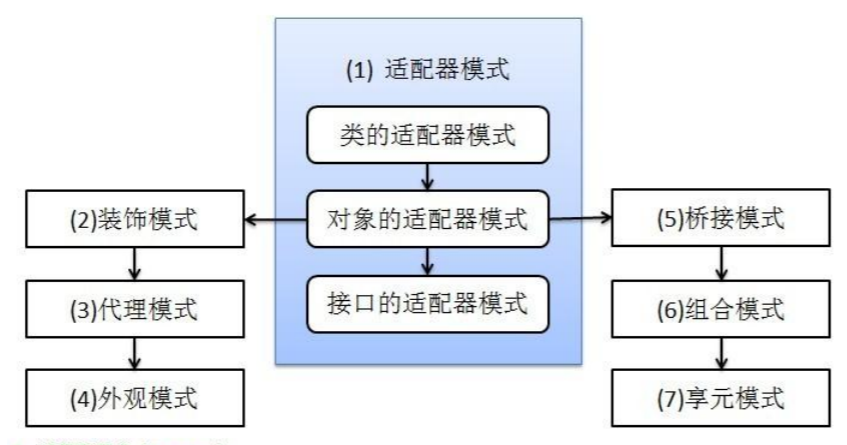
就是使用clone对象

注意：深浅拷贝

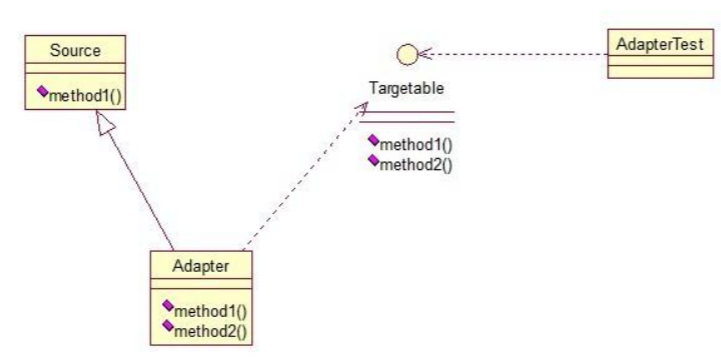
以上属于创建模式：工厂，单例，建造，原型

## 分割线-----------------------------------------------------------------------

构型模式：适配器模式、装饰模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

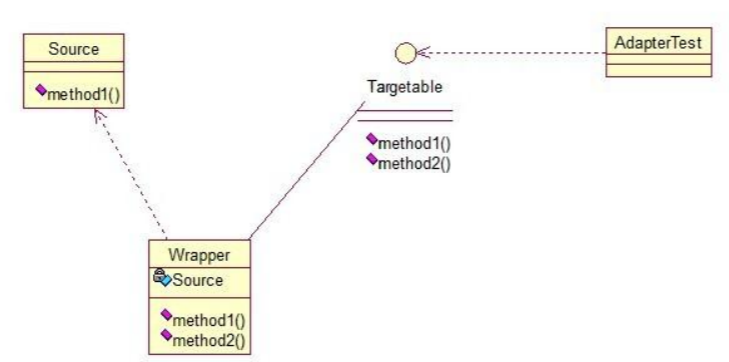


## 05适配器模式（Adapter）



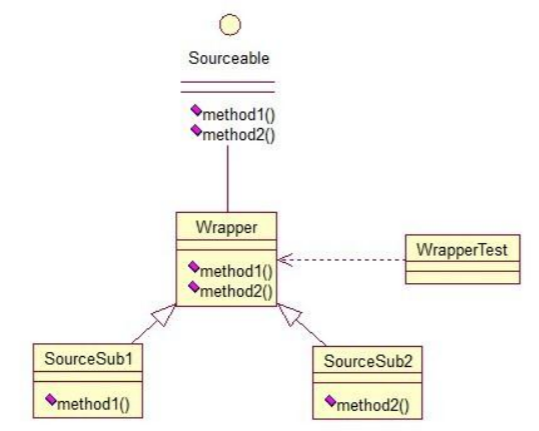
类适配器

Tagetable继承Source和实现Adapter，使用Source父类的方法method1重写Adapter接口的方法method1，这样Tagetable就可以调用Source的方法



对象适配器

Source作为Wrapper的属性，在Wrapper的method1中调用Source的method1



接口适配器

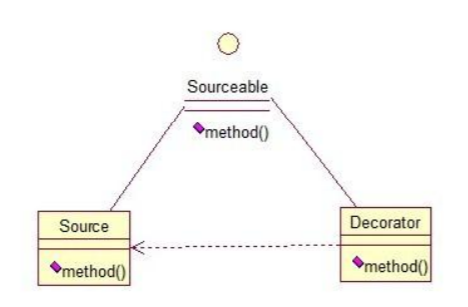
类适配器： 将一个类转换成满足另一个新接口的 类(被适配类的方法名必须和接口方法名相同)

对象适配器：将一个对象转换成满足另一个新接口的 对象

接口适配器：当需要实现一个接口时，又不想重写接口中的方法，可以先让一个空类实现接口，然后在继承这个空类，根据实际情况重写方法

## 06装饰模式（Decorator）

在对象上添加新功能，而不修改对象类源码



Source 类是被装饰类，Decorator 类是一个装饰类，共同实现Sourceable接口

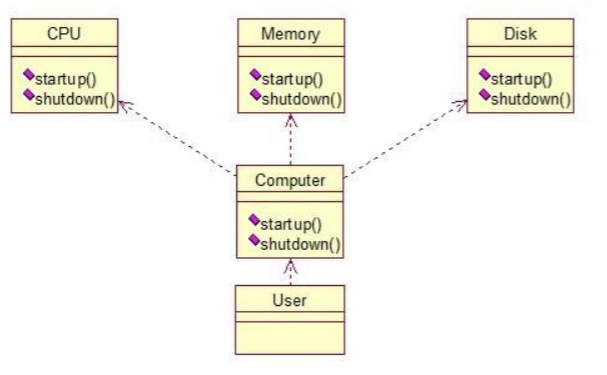
Decorator中声明一个Sourceable属性，实现时将Source传入Decorator中，Decorator对方法进行装饰，实现跟多功能

动态的为一个对象增加功能，而且还能动态撤销。（继承不能做到这一点，继承的功能 是静态的，不能动态增删。）

## 07代理模式（Proxy）

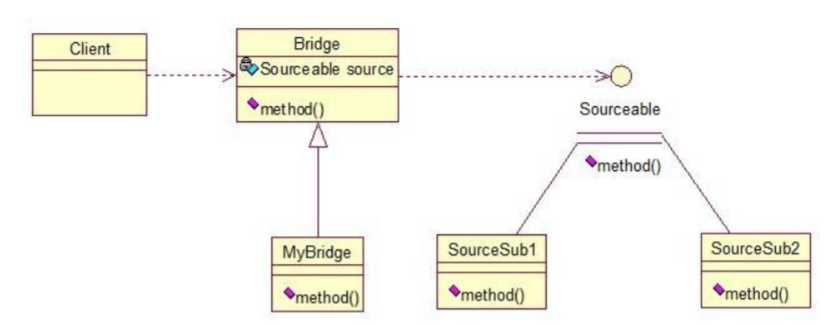
JDK动态代理，Cglib代理

## 08外观模式（Facade）/ 门面模式



将类进行整合，返回一个门店类给用户，此模式没有接口

## 09桥接模式（Bridge）



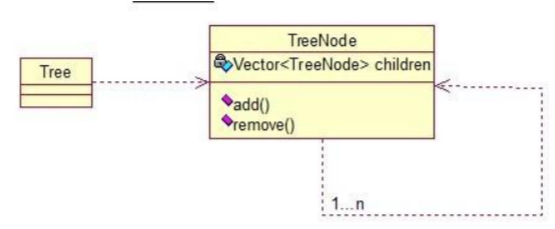
桥接模式就是把事物和其具体实现分开，将具体的实现类加到桥中，通过桥调用实现类的方法，如JDBC中的Driver就是Sourceable，DriverManager就是桥

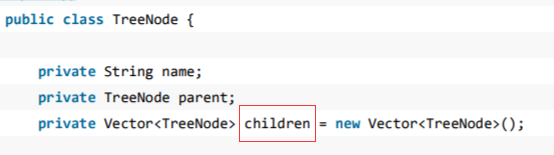
我不直接调用你，通过第三方桥来调用

桥接的用意是：将抽象化与实现化解耦，使得二者可以独立变化

实现类变化，不影响桥的使用

## 10组合模式（Composite）





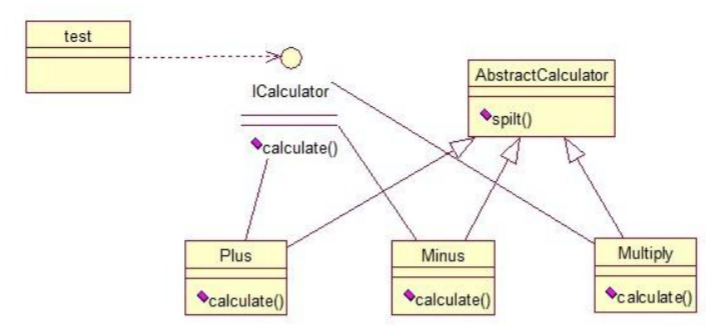
就是在当前类中引用当前类

## 11享元模式（Flyweight）

实现对象的复用，减少对象的创建，比如，在包装类中通过valueOf获取就是享元模式，数据库连接池等

这种要求只能只能使用对象方法，不能对属性进行修改

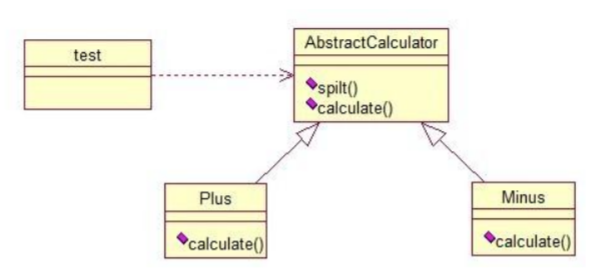
## 12策略模式（strategy）



策略模式就是根据不同的情况设计多种策略，但是客户端必须知道所有策略，策略的选取决定权在客户端

这里的AbstractCalculator只是将策略的实现抽象出来

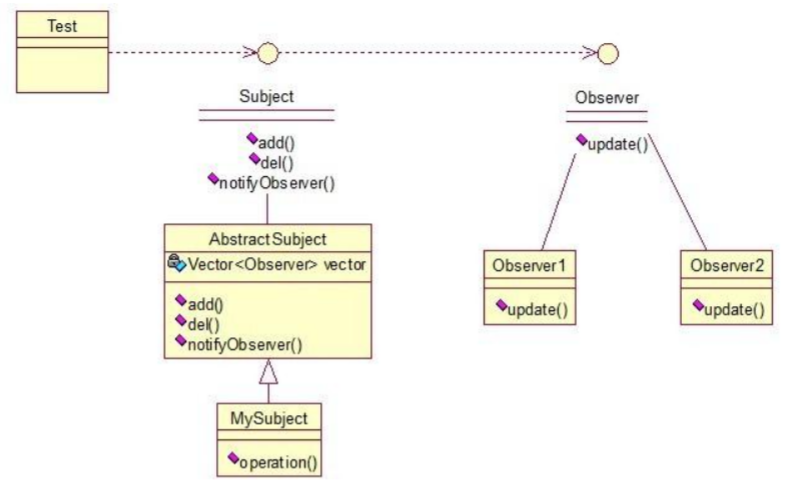
## 13模板方法模式（Template Method）



模板方法模式：一个抽象类中，有一个主方法，再定义 1...n 个方法，可以是抽象的，也可以是实际的方法，定义一个类，继承该抽象类，重写抽象方法，通过调用抽象类，实现对子类的调用

非抽象方法一般辅助抽象方法，实现具体的功能

## 14观察者模式（Observer）



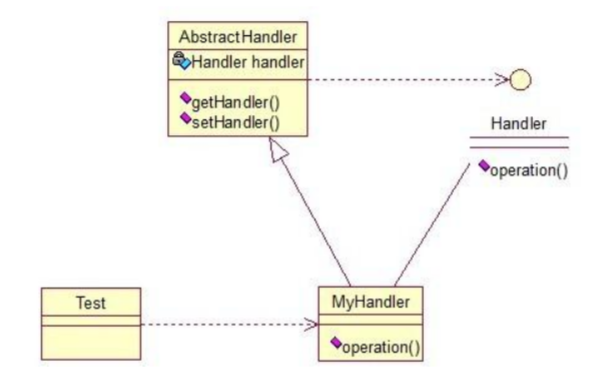
被观察者添加，删除，通知观察者等方法。被观察者状态改变，通知对应的观察者(可能会有多个)

Java提供了观察者Observer，被观察者Subject接口

## 15迭代子模式（Iterator）

对集合进行迭代操作，根据数据结构实现对应的功能(previous next hasNext)，fail-fast机制

## 16责任链模式（Chain of Responsibility）



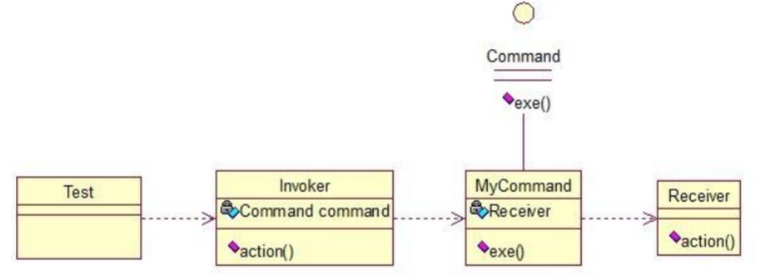
有多个对象，每个对象持有对下一个对象的引用，这样就会形成一条链，请求在这条链上传递，直到某一对象决定处理该请求。

责任链模式可以实现，在隐瞒客户端的情况下，对系统进行动态的调整。

责任链模式和装饰者模式在功能上类似，都是在以有基础上进行新功能添加，但是装饰者模型是否添加新功能有客户端决定，而责任链是隐瞒客户端

在operation中执行具体操作，并调用下一Handler

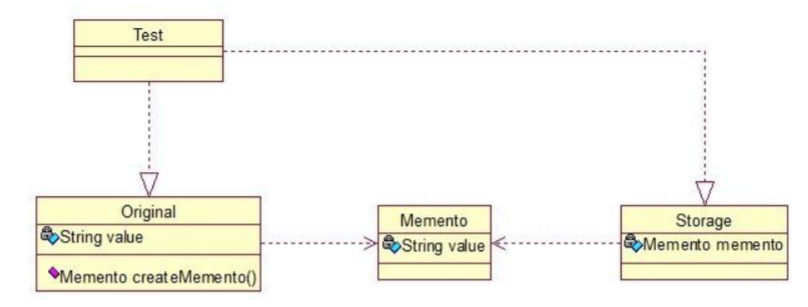
## 17命令模式（Command）



Invoker发起命令Command，Receiver接收命令

目的就是达到命令的发出者和执行者之间解耦，实现请求和执行分开

## 18备忘录模式（Memento）



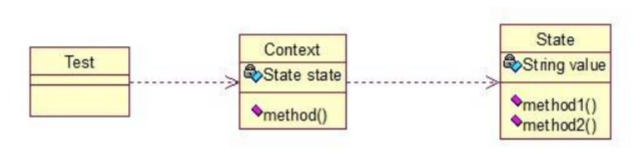
元素数据被修改后，可以通过备忘录获取被修改之前的值

Memento(备忘录)保存Original的value字段，storage存储备忘录

可以不使用Storage存储备忘录，直接将备忘录Memento作为Original的属性，但是这样不满足迪米特法则，使得类之间耦合

一个类是否需要备份，是由类本身决定的，而是外部因素，所以，类的备份和还原调用的是类的方法(createMemento)

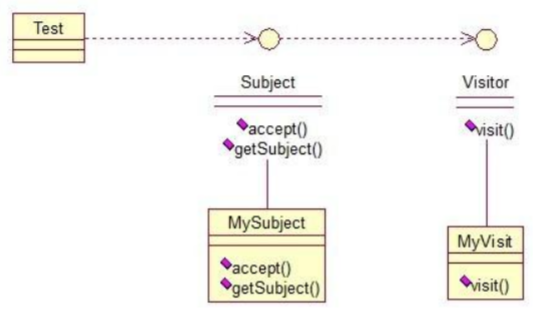
## 19状态模式（State）



当对象的状态改变时，同时改变其行为

State改变后，调用context的method改变行为(行为是State的，也可以是其他类的，而不是Context类的，context只负责通知)

## 20访问者模式（Visitor）



访问者模式就是一种分离对象数据结构与行为的方法，通过这种分离，可达到为一个被访问者动态添加新的操作而无需做其它的修改的效果。

被访问者的数据结构不改变，而访问者就是获取这些数据，然后进行操作

操作类型不同，就有多少和访问者

Visit访问被访问者，accept允许哪类访问者(这个还是看看代码)

虽然不使用访问者，获取数据后，在方法中操作，但是新操作就的创建新方法，修改调用者类，并且还有修改调用方法为新方法，改了2处

为一个现有的类增加新功能，不得不考虑几个事情：

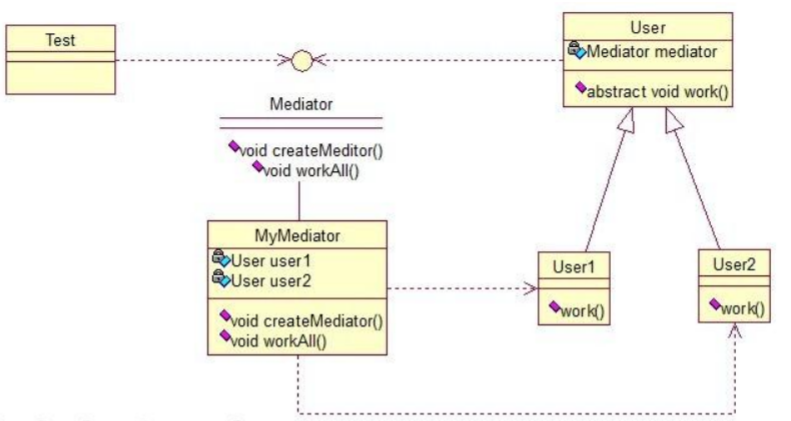
1、新功能会不会与现有功能出现兼容性问题？

2、以后会不会再需要添加？

3、如果类不允许修改 代码怎么办？

最好的解决方法就是使用访问者模式，访问者模式适用于数据结构相对稳定的系统，把数据结构和算法解耦

## 21中介者模式（Mediator）



Test调用的Mediator，而Mediator对User进行操作，这样Test就是间接操作User，就是引入一个第三方类

类之间有依赖关系的话，不利于功能的拓展和维护，因为只要修改一个对象，其它关联的对象都得进行修改。

如果使用中介者模式，只需关心和 Mediator 类的关系，具体类类之间的关系及调度交给 Mediator 就行

