装饰者设计模式

# 装饰者模式

装饰者模式：动态地将责任附加到对象上。若要扩展功能，装饰者提供了比继承更有弹性的替代方案。

需要扩展类的功能：

## 继承的方法：编译时扩展；

## 装饰者模式：运行时扩展。

# 装饰者模式具体介绍

## 装饰者与被装饰者

**被装饰者**一般是一个组件，即为**Component**。

**装饰者**：通过拥有一个组件（被装饰者）对象这个属性，扩展其他的功能，对组件进行扩展。

## 装饰者模式的要求

**装饰者与被装饰者都必须实现或继承同一个接口或基类**（抽象、具体都可以）。

装饰者与被装饰者实现相同接口或相同基类的目的不同：

### 被装饰者：实现接口或继承基类，目的是继承功能；

### 装饰者：实现接口或继承基类，目的是获取正确的类型，并不主要是继承它的行为。因为只有是相同类型时，才能被另外的其他装饰者装饰，即只有装饰者们是属于同一类型的，这样才能相互装饰。装饰者也可以作为被装饰者。

# Java IO流体现了装饰者设计模式

示例 ：

**InputStream**是一个抽象基类，FileInputStream、ByteArrayInputStream就是对其的具体实现类，作为**被装饰者**。

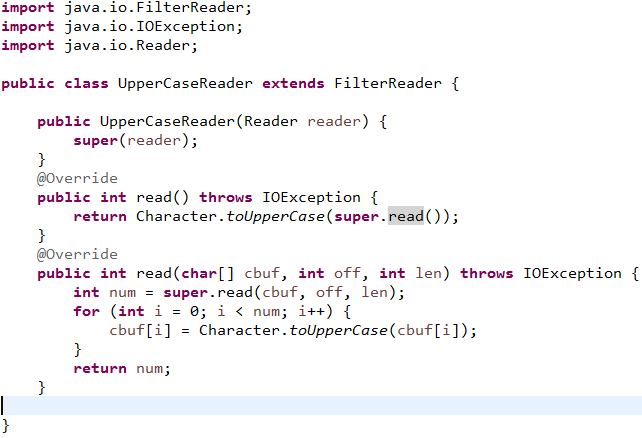
**FilterInputStream**是一个具体类，目的就是为那些扩展InputStream功能的流提供同一的基类。如BufferedInputStream、LineNumberInputStream等都继承自**FilterInputStream流**，两者都是装饰者，同时也都是被装饰者，因为它们可以相互装饰对方。

总的来说，这些流都是直接或间接地继承自**InputStream**。都可以利用InputStream基类的引用指代。虽然不是接口，但是利用了面向接口编程的思想，充分体现了多态的思想。

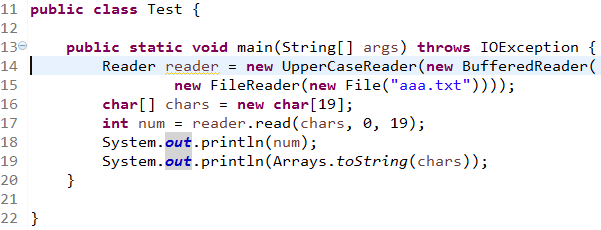
其他OutputStream、Reader、Writer都是类似的。

示例：构建一个装饰者UpperCaseReader类，继承FilterReader。

只需要覆盖read方法即可。



测试：







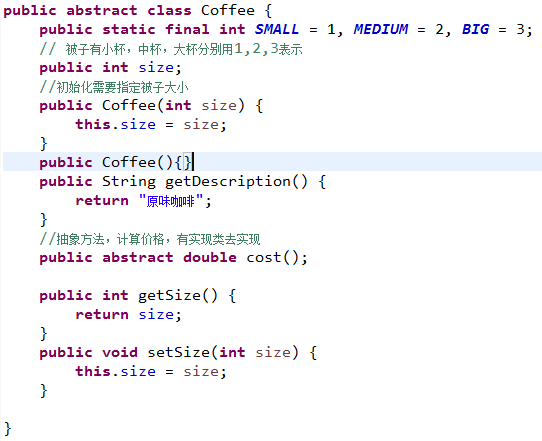
# 示例：

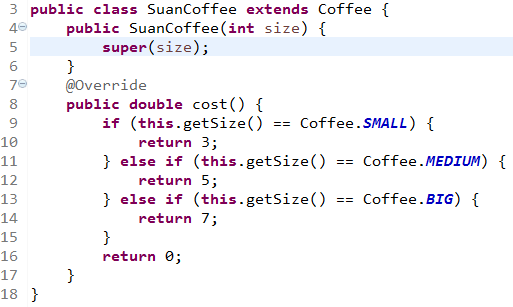
这里建立一个简单的模型，就是去咖啡店喝咖啡，咖啡本身分很多类型，这里分了四种酸甜苦辣，同时还有很多调料可以加，这里给出四种调料，醋、盐、苦、辣椒。例子举的不好，不过思想就是这样的。

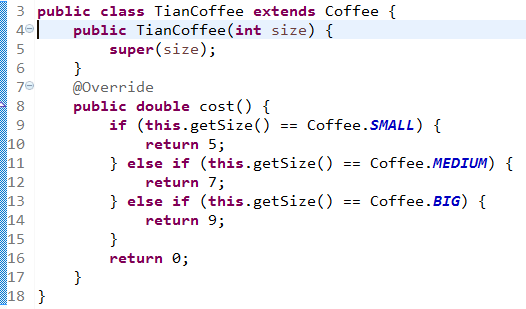
**抽象基类**：Coffee，具体的实现类有：SuanCoffee、TianCoffee、KuCoffee、LaCoffee。

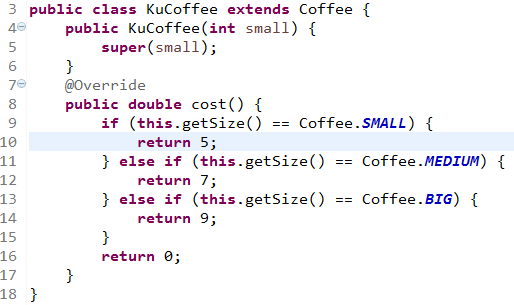
需要提供一个单独的调料基类，去继承Coffee，调料基类就是TiaoLiao，该类仍然是抽象的。具体的调料（装饰者）有CuTiaoLiao、SaltTiaoLiao、KuTiaoLiao、LajiaoTiaoLiao。

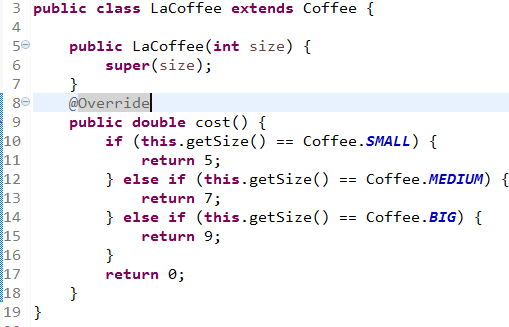
目标：咖啡店提供四种基本的咖啡，分别都具有大中小杯子，且提供四种调料，调料任意添加。杯子大中小，咖啡及调料都是不同的价格。





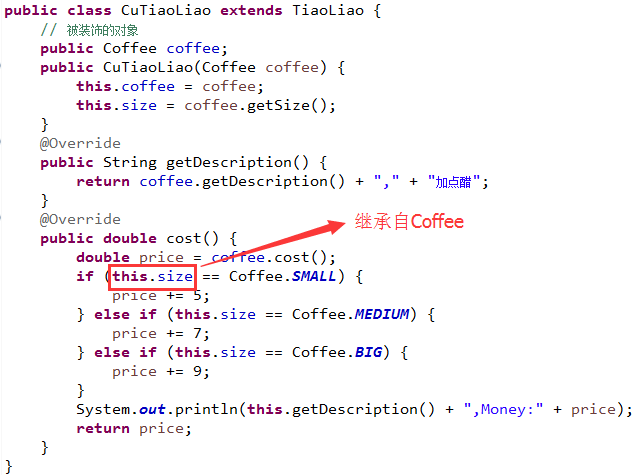


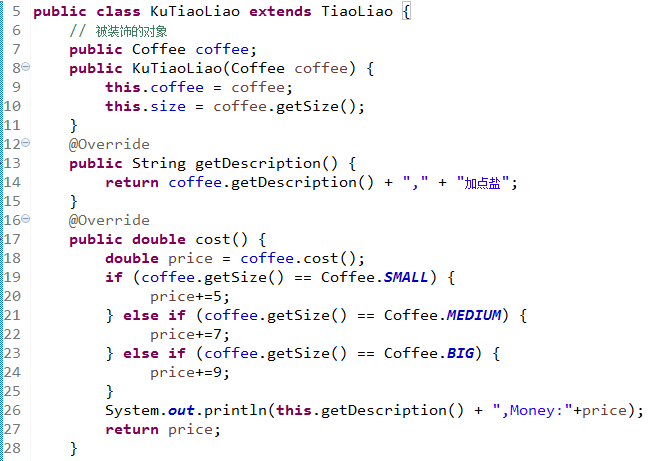


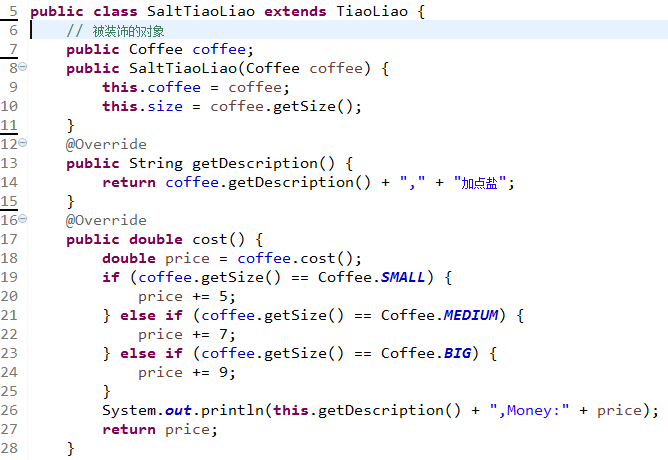


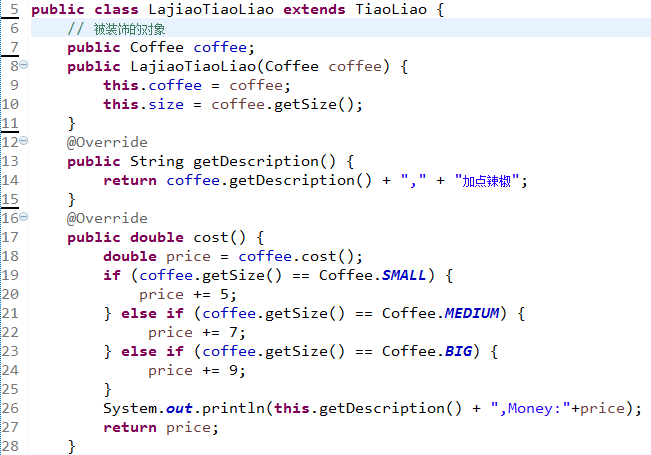
调料基类：











Test测试：

