


设计模式之装饰者模式(Decorator Pattern)



六尺帐篷 (/u/f8e9b1c246f1)

2016.07.27 20:27 字数 1733 阅读 186 评论 0 喜欢 6

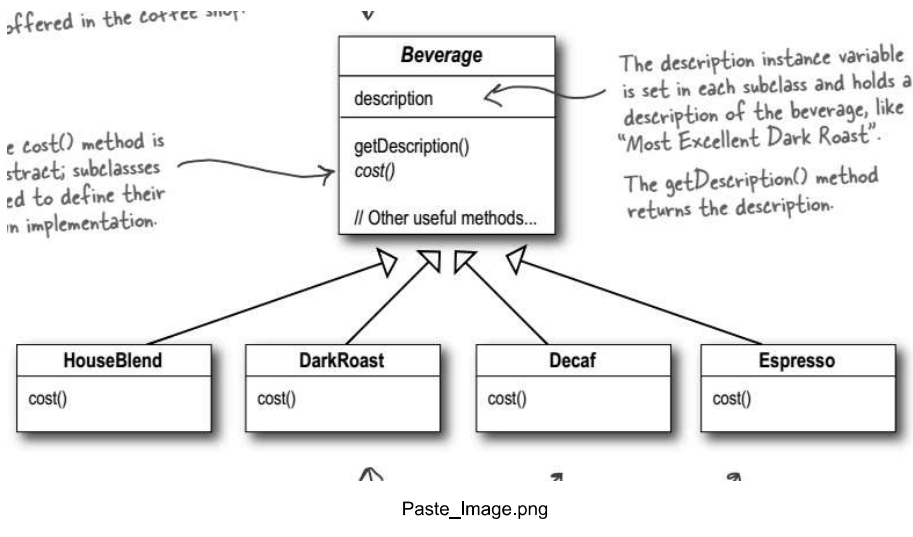
(/u/f8e9b1c246f1)

编辑文章 (/writer#/notebooks/5229761/notes/4989176)

装饰者模式可以做到在不修改任何底层代码的情况下，给对象增加的新的方法。
首先，我们通过对一个现实问题的模拟分析，了解什么是装饰者模式以及装饰者模式的作用。

问题提出

咖啡店在街头随处可见。我们以咖啡店的饮品订单系统为例。假设我们要设计一个饮品的订单系统。
设计了一个这样的类图：

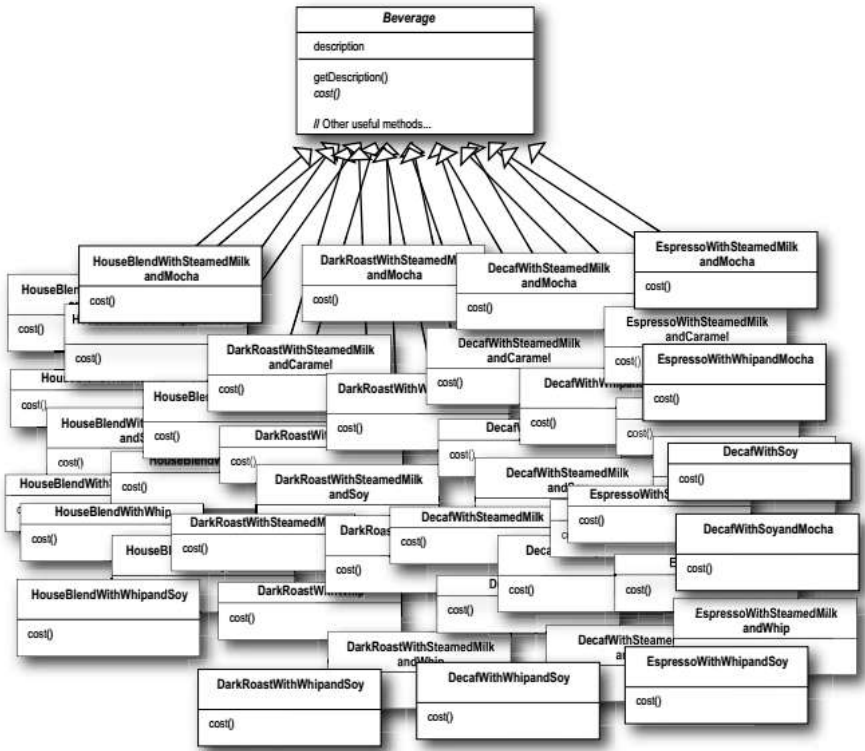


Beverage是一个 抽象类，所有咖啡店的饮品都必须继承这个类，description是饮品的描述信息，cost()是计算此种饮品的价格。

我们会遇到这样的问题，在购买饮品的时候，我们可以要求在其中加入不同的调料配料，比如，摩卡（Mocha），加奶泡等。除了原本饮料需要的价格的外，咖啡店会根据所加入调料的再收取不同的费用。

如果按照之前的设计方式，那么会出现如下的情况：





Paste_Image.png

显然这似乎已经是类爆炸了！

而且我们永远无法预测，顾客会选取怎样的调料的搭配，每当出现一个新的调料搭配时，我们就需要增加一个新的类。
更加糟糕的是，当原料配料的价格上涨后或者下降后，那么所有涉及到这种配料的类都得重新改过。这简直是个噩梦！很显然这很不符合我们设计模式的原则。作为一个程序员，我们是决不能容忍这种情况发生的！

那么我们该如何设计呢？

这里就需要用到我们的装饰者模式！

引出装饰者模式

让我们转换思路，我们以饮品beverage为主体，在运行时以顾客选择的调料来装饰beverage。比如，如果顾客想要摩卡和奶泡的拿铁咖啡，我们要做的应该是这样的：

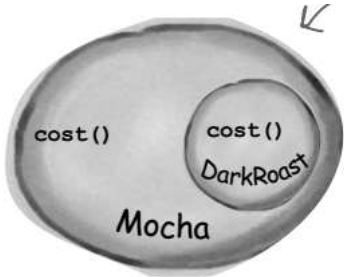
- 取一个拿铁咖啡的对象



Paste_Image.png

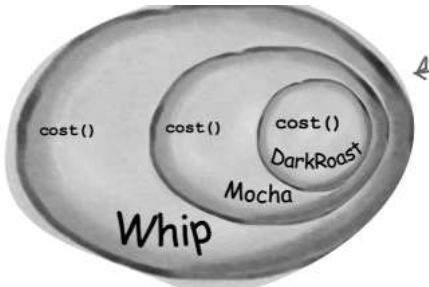
- 用摩卡对象装饰它





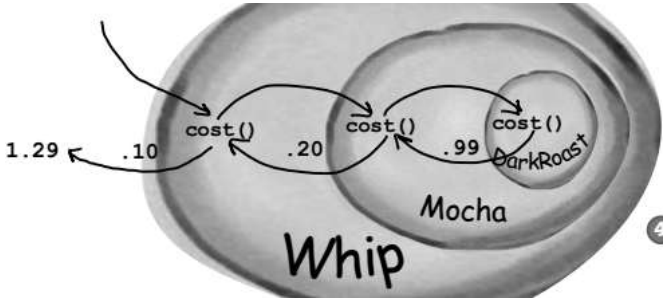
Paste_Image.png

- 用奶泡对象装饰它



Paste_Image.png

- 调用cost方法计算价钱，并依赖委托将配料摩卡和奶泡加上去。



Paste_Image.png

会先计算whip的cost然后调用mocha的cost，然后调用拿铁的cost，这样就计算出了总价。
这样就是实现的装饰者模式解决这个问题的思路。
下面我们看一下装饰者模式的定义，以及代码实现的基本思路

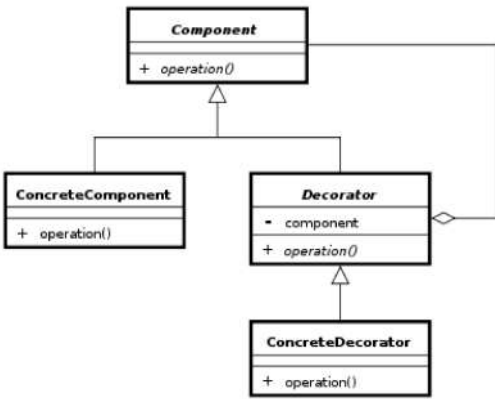
定义装饰者模式

装饰者模式动态的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。

📄

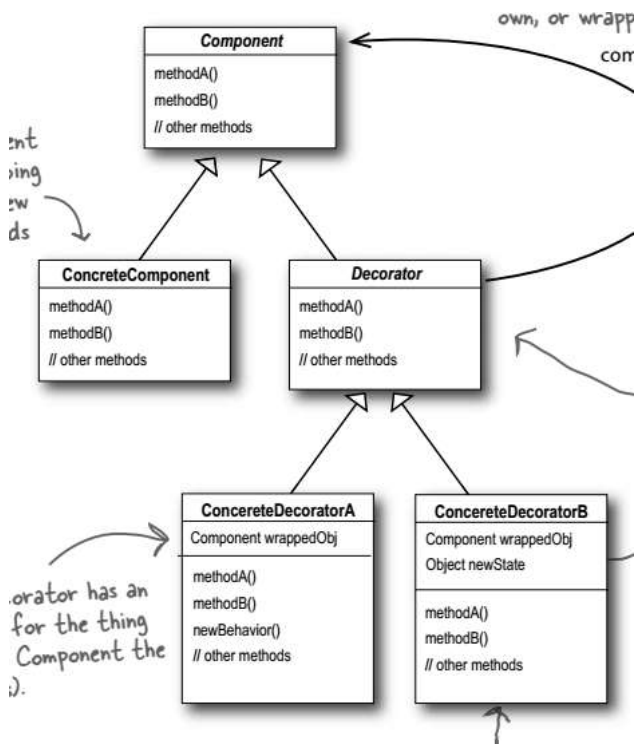
🔖

🔗



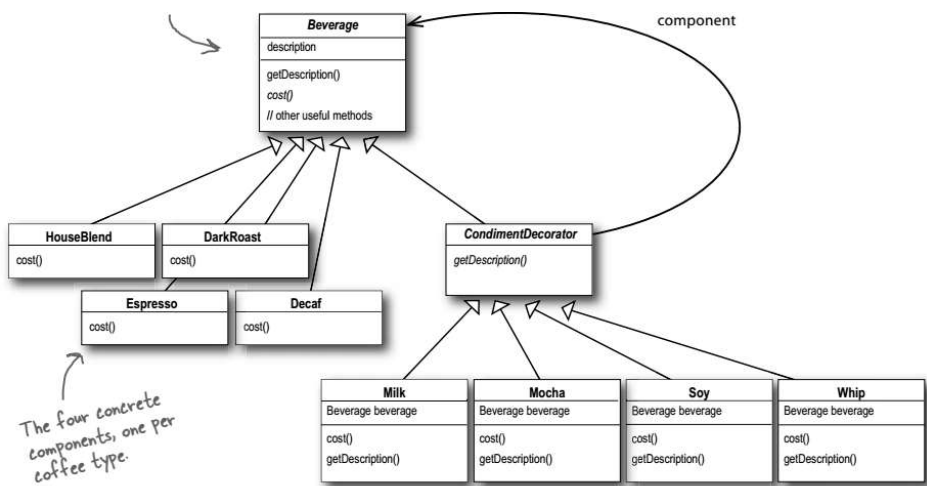
Paste_Image.png

这个类图就是装饰者模式的实现方式。更详细的是如下这个版本的类图。



Paste_Image.png

下面我们就根据这个类图来解决我们之前在实现咖啡店饮料系统上遇到的问题。



Paste_Image.png

分析设计类图：

- beverage相当于抽象的component类，具体的component和decorator都需要继承实现这个抽象类。
- 四个具体的饮料的类，相当于concrete component！每一个类代表了一个饮料类型。
- condimentDecorator是抽象的decorator类，它是所有调料类的抽象，它保存了beverage的一个引用。
- 调料装饰者类继承自condimentDecorator，是各种具体调料的实现，他们都实现了cost方法。

上面有一个非常关键的地方，就是我们注意到装饰者和被装饰者必须是一样的类型，也就是拥有共同的超类。这样做是因为我们要装饰者必须能取代被装饰者。这样我们就可以利用对象的组合，将调料和饮料的行为组合起来。这符合我们之前提到的设计原则**多用组合，少用继承**

实现装饰者模式

如果看到这里还是不太清楚，也没关系，接下来我们将具体实现代码，对装饰者模式有一个直观根本的了解。

- 首先实现beverage和condiment两个抽象类

```
package abstractComponent;

public abstract class Beverage {
    protected String description = "Unknow Beverage";

    public String getDescription() {
        return description;
    }

    public abstract double cost();
}
```

```
package abstractDecorator;
import abstractComponent.Beverage;

public abstract class CondimentDecorator extends Beverage {
    public abstract String getDescription();
}
```

- 然后我们实现具体的饮料类

```
package concreteComponent;
import abstractComponent.Beverage;

public class Coco extends Beverage {
    public Coco(){
        description = "Coco";
    }

    public double cost(){
        return 0.89;
    }
}
```



```
package concreteComponent;  
import abstractComponent.Beverage;  
  
public class Espresso extends Beverage {  
    public Espresso() {  
        description = "Espresso";  
    }  
  
    public double cost() {  
        return 1.99;  
    }  
}
```

- 我们再实现具体的装饰者类，也就是调料类

```
package concreteDecorator;  
  
import abstractComponent.Beverage;  
import abstractDecorator.CondimentDecorator;  
  
public class Mocha extends CondimentDecorator {  
  
    Beverage beverage;  
  
    public Mocha(Beverage beverage){  
        this.beverage = beverage;  
    }  
  
    @Override  
    public double cost() {  
        // TODO Auto-generated method stub  
        return .20 + beverage.cost();  
    }  
  
    @Override  
    public String getDescription() {  
        // TODO Auto-generated method stub  
        return beverage.getDescription() + ", Mocha";  
    }  
}
```

```
package concreteDecorator;  
  
import abstractComponent.Beverage;  
import abstractDecorator.CondimentDecorator;  
  
public class Soy extends CondimentDecorator {  
  
    Beverage beverage;  
  
    public Soy(Beverage beverage){  
        this.beverage = beverage;  
    }  
  
    @Override  
    public String getDescription() {  
        // TODO Auto-generated method stub  
        return beverage.getDescription() + ", Soy";  
    }  
  
    @Override  
    public double cost() {  
        // TODO Auto-generated method stub  
        return .15 + beverage.cost();  
    }  
}
```



```
package concreteDecorator;

import abstractComponent.Beverage;
import abstractDecrator.CondimentDecorator;

public class Whip extends CondimentDecorator {

    Beverage beverage;

    public Whip(Beverage beverage){
        this.beverage = beverage;
    }

    @Override
    public String getDescription() {
        // TODO Auto-generated method stub
        return beverage.getDescription() + " , whip";
    }

    @Override
    public double cost() {
        // TODO Auto-generated method stub
        return .10 + beverage.cost();
    }

}
```

- 最后编写一个测试类，来测试我们装饰者模式的效果如何

```
import concreteComponent.Coco;
import concreteComponent.Espresso;
import concreteDecorator.Mocha;
import concreteDecorator.Soy;
import concreteDecorator.Whip;
import abstractComponent.Beverage;

public class Test {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Beverage beverage = new Espresso();
        System.out.println( beverage.getDescription() + "$" + beverage.cost());

        Beverage beverage2 = new Coco();
        beverage2 = new Mocha(beverage2);
        beverage2 = new Mocha(beverage2);
        beverage2 = new Whip(beverage2);
        System.out.println( beverage2.getDescription() + "$" + beverage2.cost());

        Beverage beverage3 = new Espresso();
        beverage3 = new Whip(beverage3);
        beverage3 = new Mocha(beverage3);
        beverage3 = new Soy(beverage3);
        System.out.println( beverage3.getDescription() + "$" + beverage3.cost());
    }

}
```

总结与分析

通过装饰者模式我们可以很好的解决咖啡店的问题，用装饰者去包装组件，可以达到很好的可扩展性。

- 装饰者模式用到的技术主要有两种就是组合和委托，这帮助我们动态的在运行时加上新的行为。
- 装饰者模式意味着一群装饰者类，这些类用来包装装饰者。
- 装饰者和被装饰者类实际上具有相同类型的。
- 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，甚至完全覆盖。
- 但装饰者模式的使用会导致出现很多小对象，就是装饰者对象，过度使用也会使程序变得复杂。





六尺帐篷 (/u/f8e9b1c246f1)

写了 245418 字，被 15240 人关注，获得了 1429 个喜欢

(/u/f8e9b1c246f1)

喜欢 6




更多分享

(http://cwb.assets.jianshu.io/notes/images/4989176


被以下专题收入，发现更多相似内容

投稿管理


+ 收入我的专题




程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)



Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)



Android... (/c/8404941e82d2?utm_source=desktop&utm_medium=notes-included-collection)



首页投稿 (/c/bDHhpK?utm_source=desktop&utm_medium=notes-included-collection)

