


# 装饰者模式

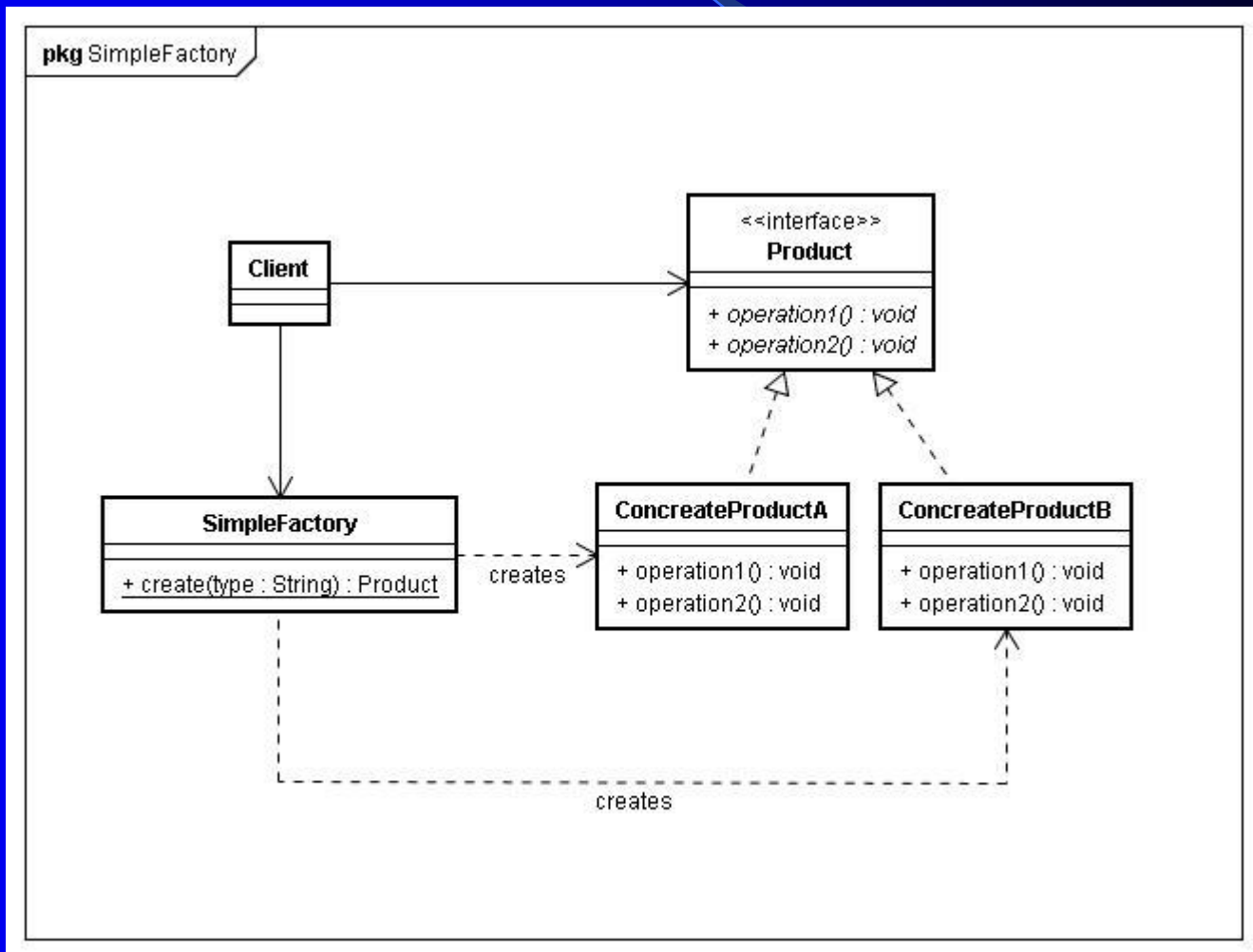
The background of the slide is a dark blue gradient. A light blue curved line starts from the left edge and curves downwards towards the bottom right. A semi-transparent, lighter blue triangular shape is positioned in the bottom right corner, pointing towards the center of the slide.

# 回顾：创建型模式

- 简单工厂
- 工厂方法
- 抽象工厂

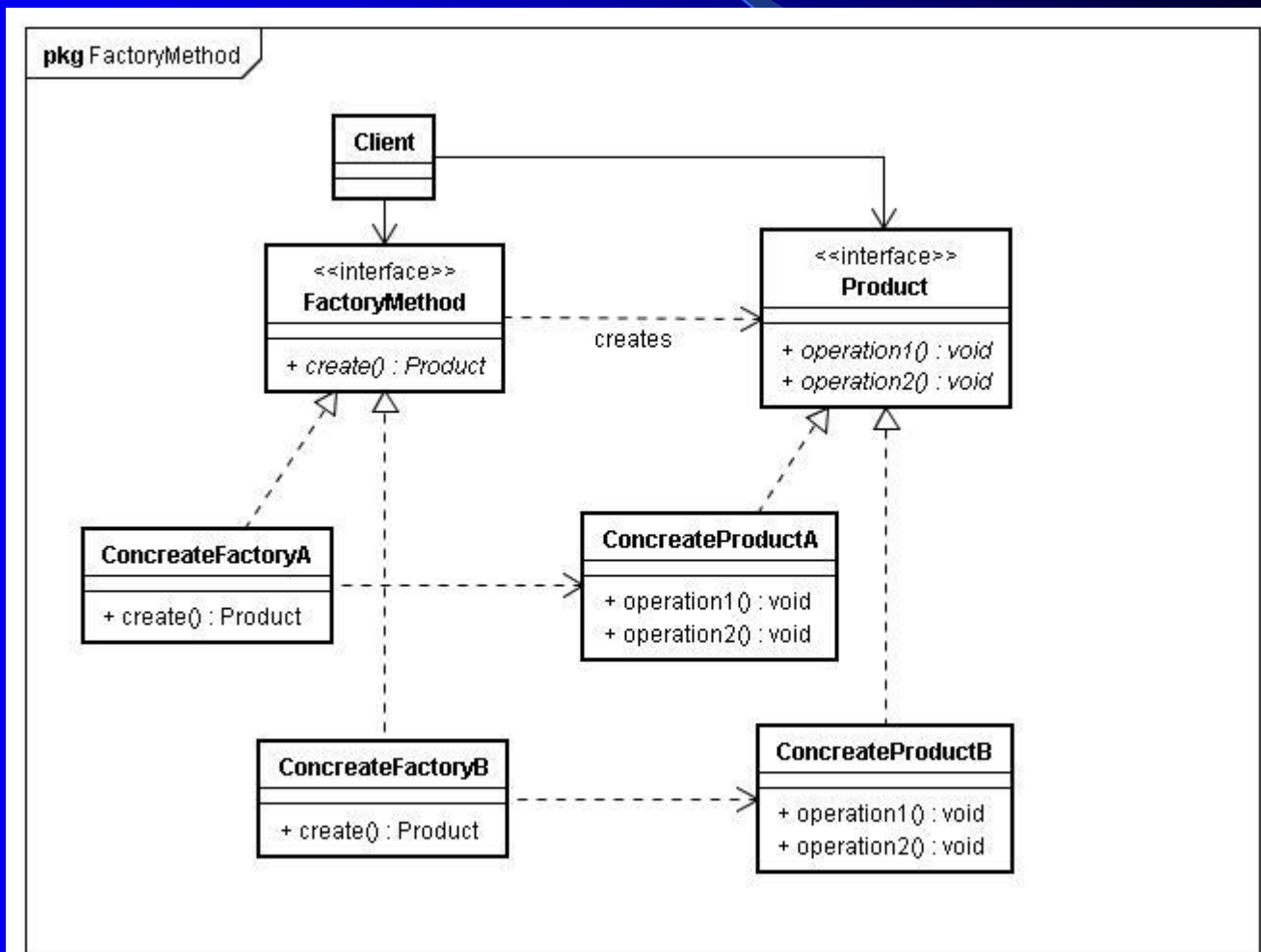
# 简单工厂

- 通过接收的参数不同来返回不同的对象实例。



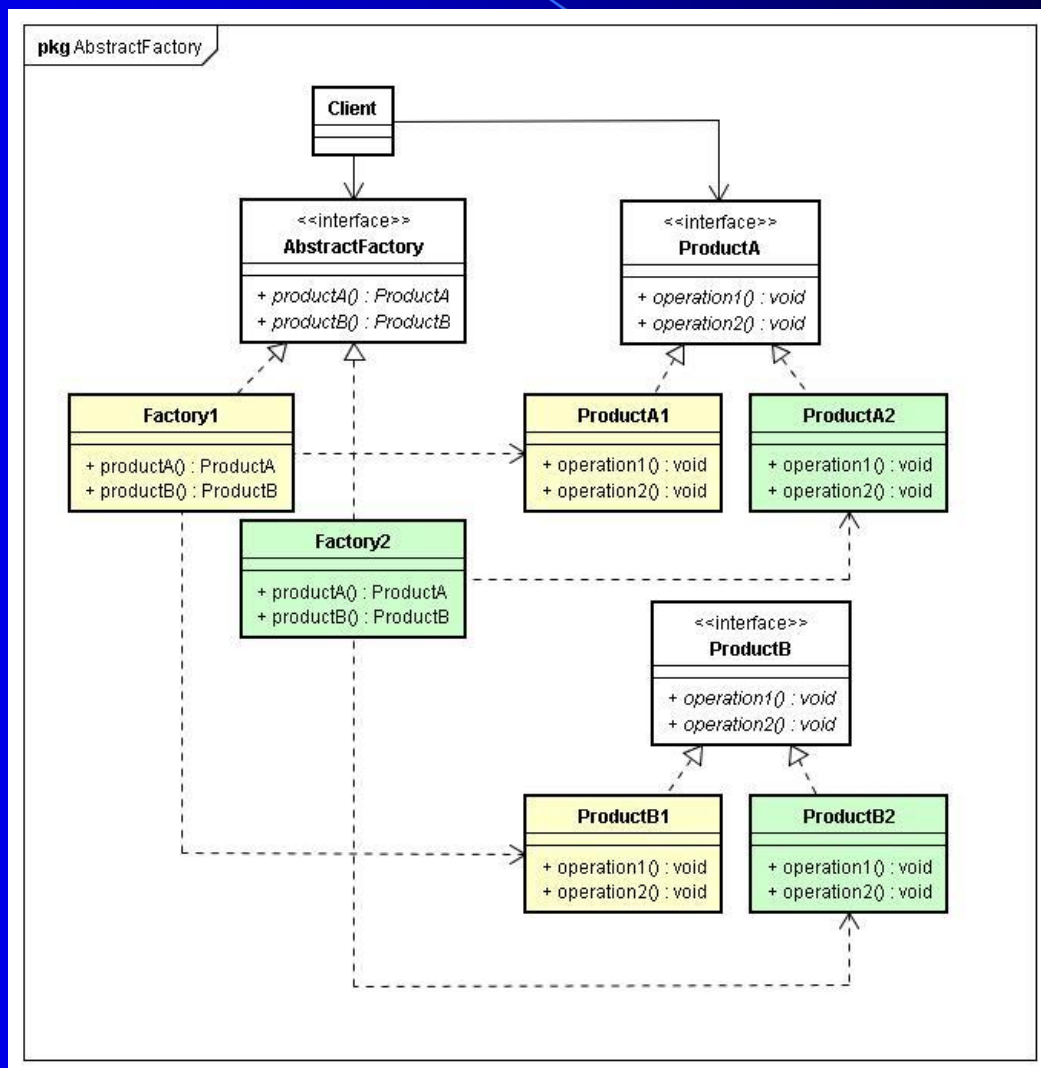
# 工厂方法

- 工厂方法是针对每一种产品提供一个工厂类。通过不同的工厂实例来创建不同的产品实例。



# 抽象工厂

- 抽象工厂是应对产品族概念的。



# 装饰者模式

- 本章讲解如何使用对象组合的方式，做到在运行时装饰类。

# 咖啡连锁店

- 销售各种饮料 (Beverage)
- 订单系统
- 咖啡类型：深焙咖啡 (DarkRoast), **DarkRoast, HouseBlend, Decaf, Espresso.....**
- 可以加入各种调料，例如：蒸奶 (Steamed Milk)、豆浆 (Soy)、摩卡 (Mocha, 也就是巧克力风味) 或覆盖奶泡

# 咖啡连锁店

- 因为扩张速度实在太快了，他们准备更新订单系统，以合乎他们的饮料供应要求。



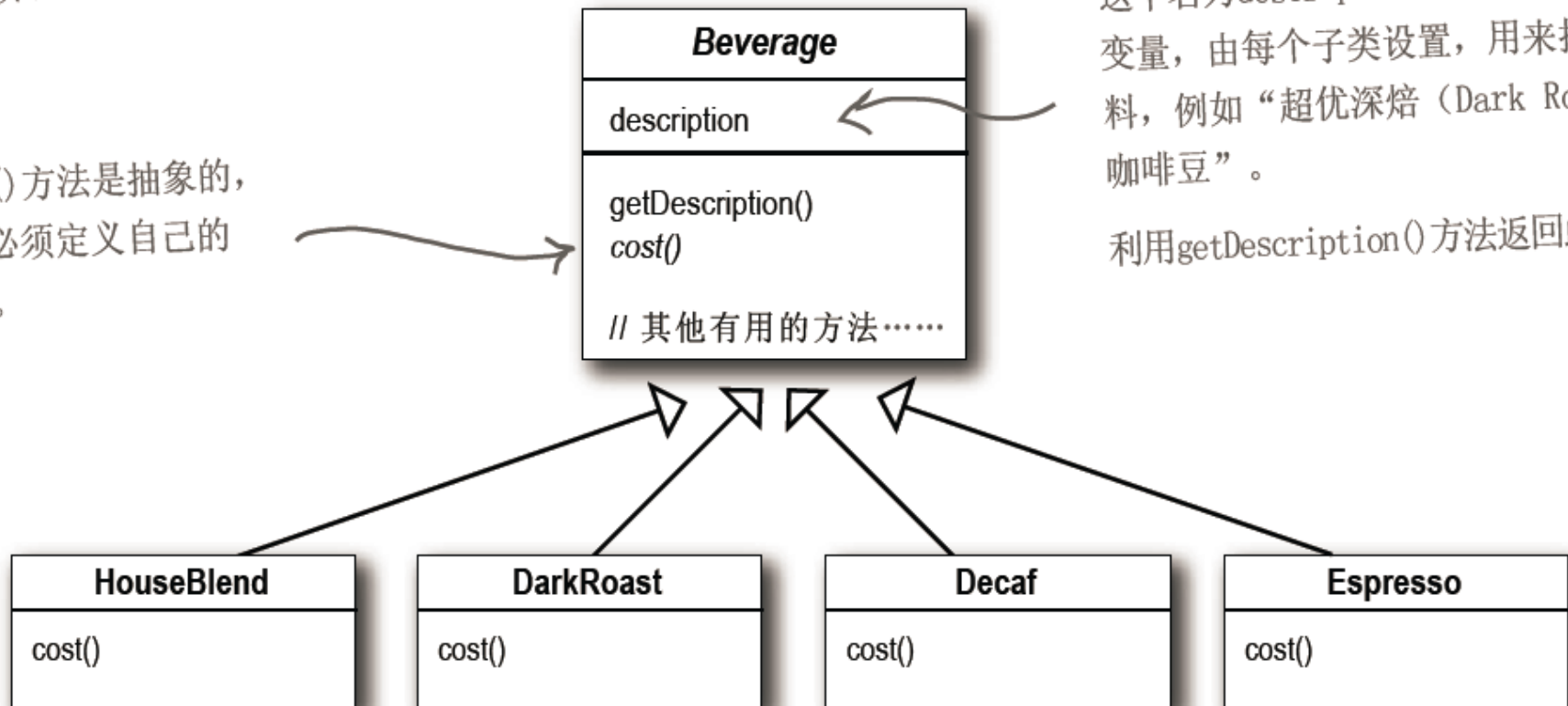
# 他们原先的类设计是这样的.....

Beverage (饮料) 是一个抽象类，店内所提供的饮料都必须继承自此类。

cost() 方法是抽象的，子类必须定义自己的实现。

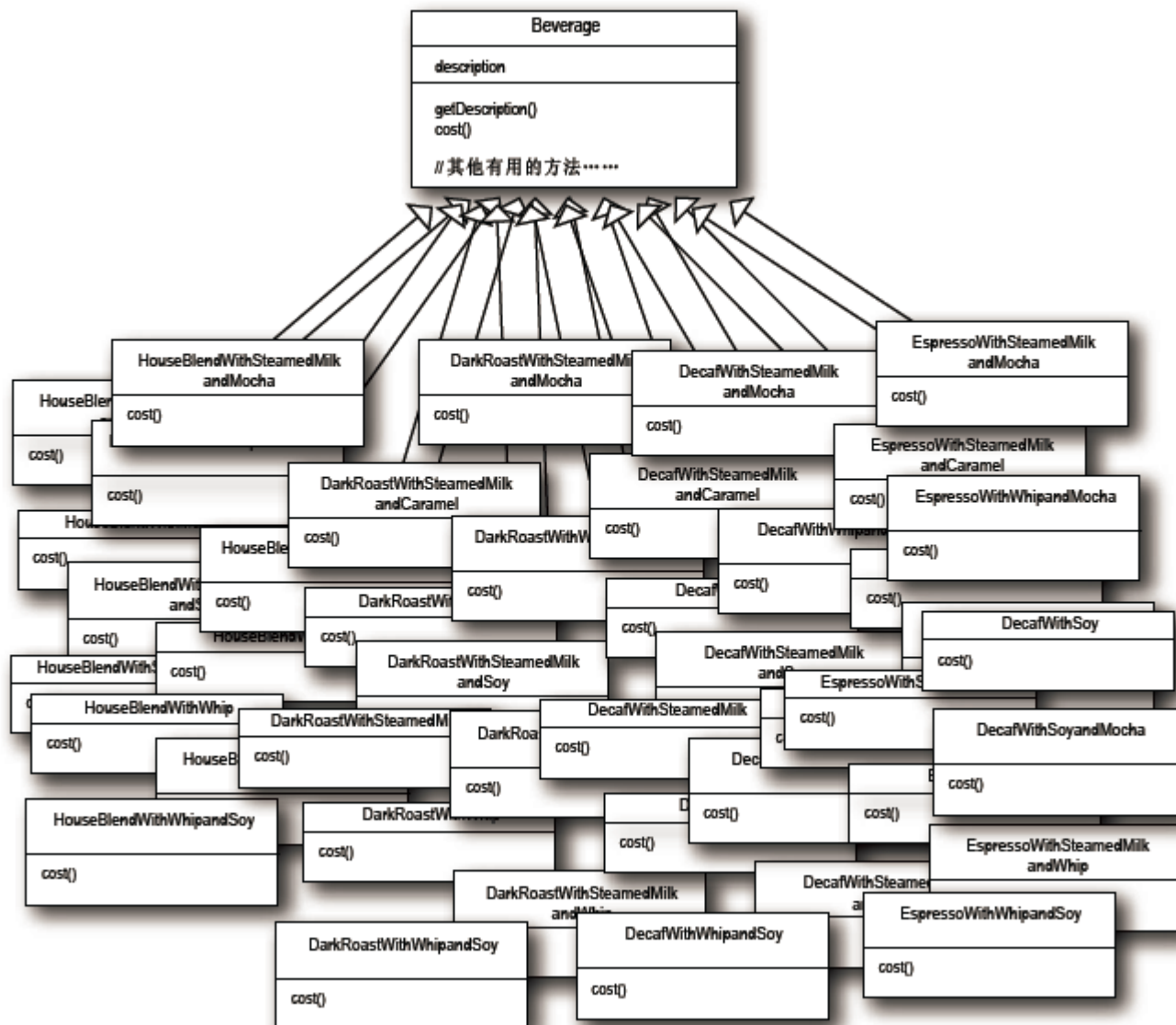
这个名为description (叙述) 的实例变量，由每个子类设置，用来描述饮料，例如“超优深焙 (Dark Roast) 咖啡豆”。

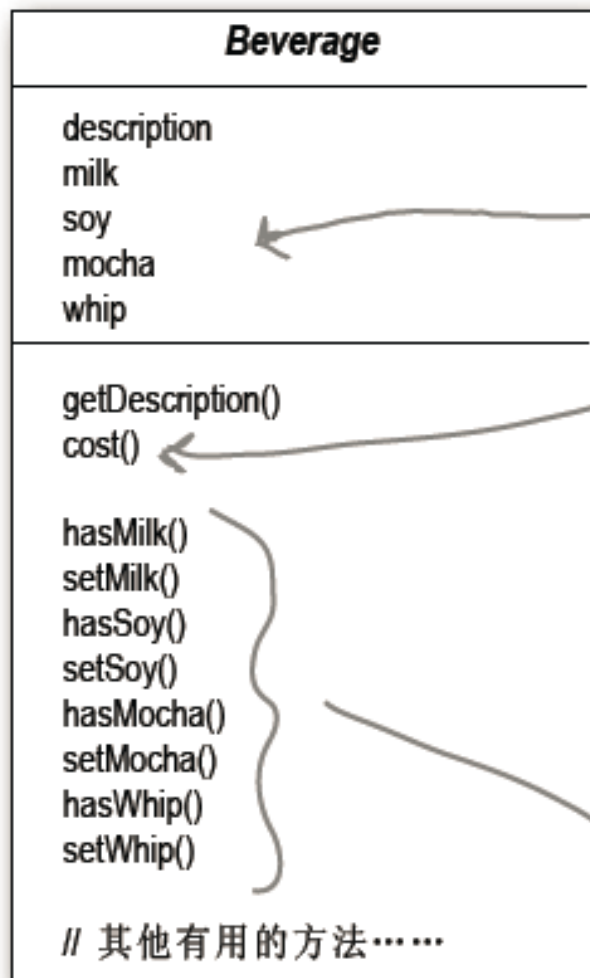
利用getDescription() 方法返回此叙述。



- 加入各种调料，例如：蒸奶（Steamed Milk）、豆浆（Soy）、摩卡（Mocha，也就是巧克力风味）或覆盖奶泡。咖啡店会根据所加入的调料收取不同的费用。所以订单系统必须考虑到这些调料部分。

# 这是他们的第一个尝试.....





各种调料的新的布尔值

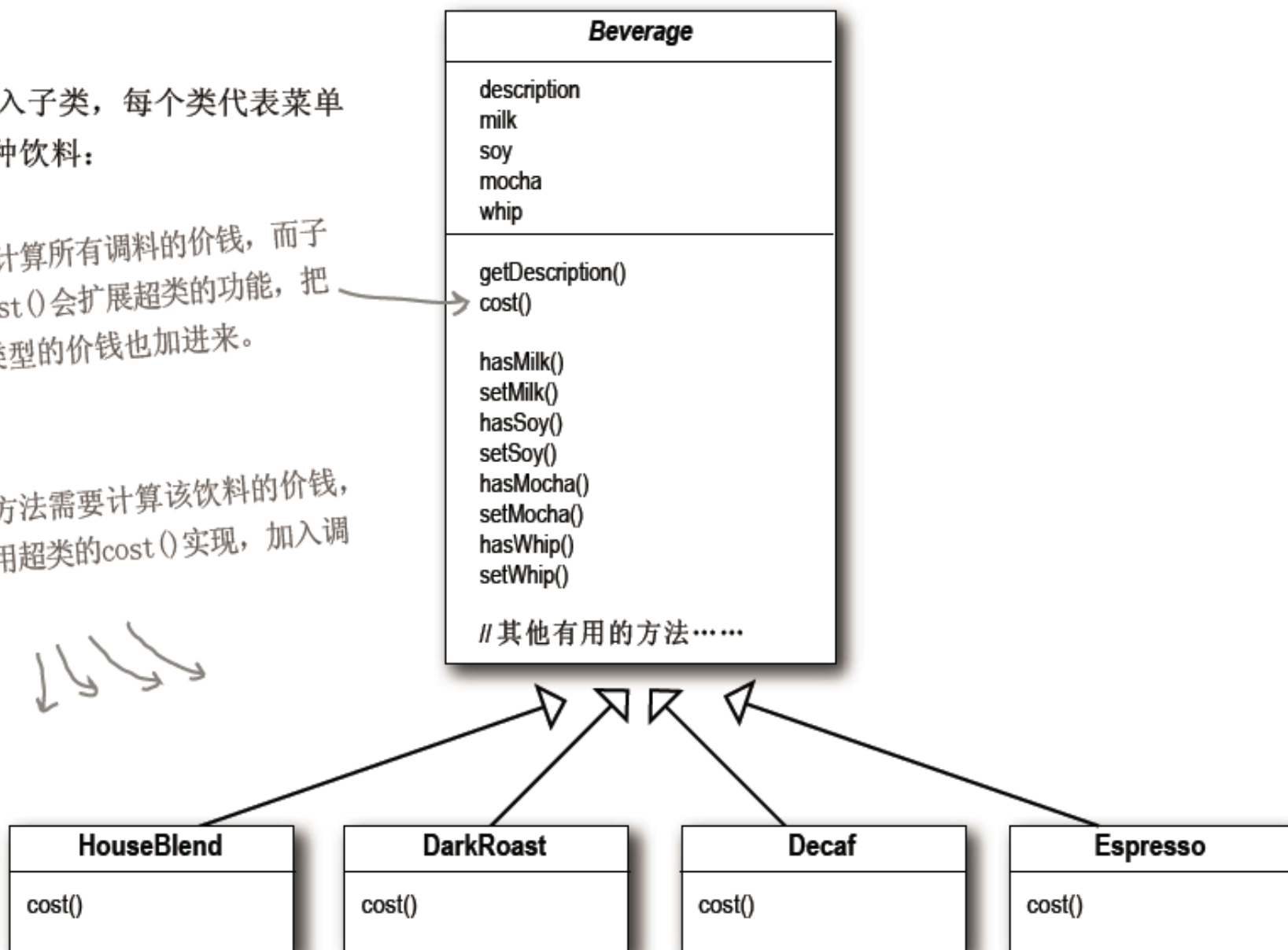
现在，Beverage类中的`cost()`不再是一个抽象方法，我们提供了`cost()`的实现，让它计算要加入各种饮料的调料价钱。子类仍将覆盖`cost()`，但是会调用超类的`cost()`，计算出基本饮料加上调料的价钱。

这些方法取得和设置调料的布尔值。

现在加入子类，每个类代表菜单上的一种饮料：

超类cost()将计算所有调料的价钱，而子类覆盖过的cost()会扩展超类的功能，把指定的饮料类型的价钱也加进来。

每个cost()方法需要计算该饮料的价钱，然后通过调用超类的cost()实现，加入调料的价钱。





## Sharpen your pencil

请为下面类的`cost()`方法书写代码（用伪Java代码即可）。

```
public class Beverage {  
    public double cost() {
```

```
    }  
}
```

```
public class DarkRoast extends Beverage {  
  
    public DarkRoast() {  
        description = "Most Excellent Dark Roast";  
    }  
  
    public double cost() {
```

```
    }  
}
```

```
public class Beverage {  
  
    //为milkCost、soyCost、mochaCost  
    //和whipCost声明实例变量。  
    //为milk、soy、mocha和whip  
    //声明getter与setter方法。  
  
    public double cost() {  
        float condimentCost = 0.0;  
        if (hasMilk()) {  
            condimentCost += milkCost;  
        }  
        if (hasSoy()) {  
            condimentCost += soyCost;  
        }  
        if (hasMocha()) {  
            condimentCost += mochaCost;  
        }  
        if (hasWhip()) {  
            condimentCost += whipCost;  
        }  
        return condimentCost;  
    }  
}
```

```
public class DarkRoast extends Beverage {  
  
    public DarkRoast() {  
        description = "Most Excellent Dark Roast";  
    }  
  
    public double cost() {  
        return 1.99 + super.cost();  
    }  
}
```

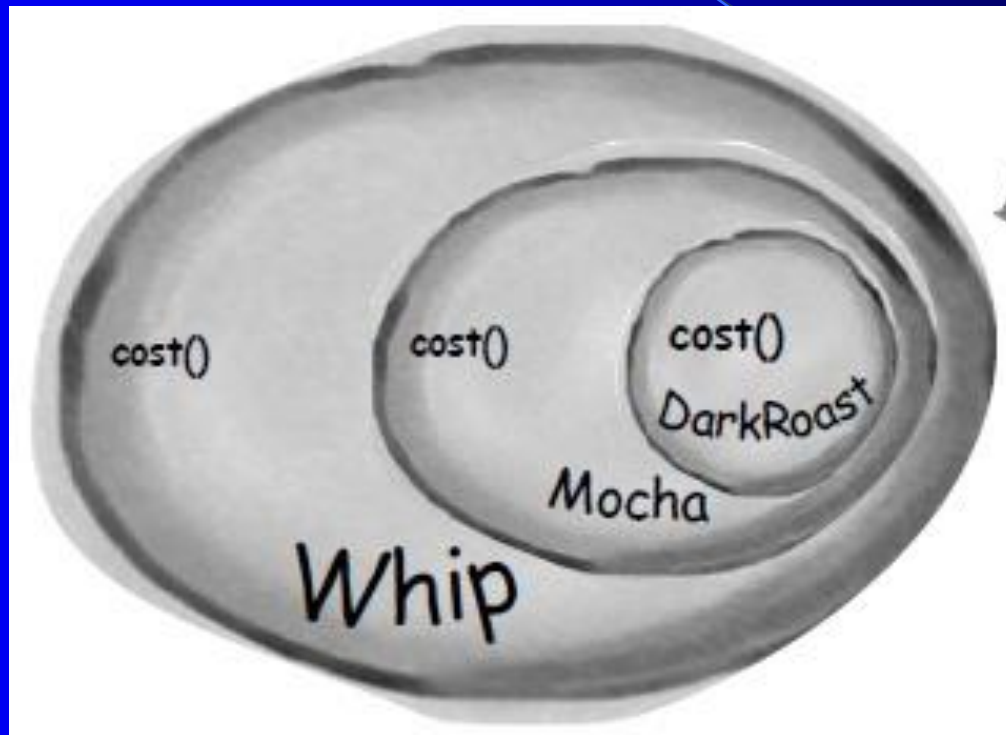
# 当哪些需求或因素改变时会影响这个设计？

- 调料价钱的变化会使我们更改现有代码
- 一旦出现新的调料，我们就需要加上新的方法，并改变超类中的cost()方法
- 以后可能会开发出新饮料。对这些饮料而言（例如：冰茶），某些调料可能并不适合，但是在这个设计方式中，Tea（茶）子类仍将继承那些不适合的方法，例如：hasWhip()（加奶泡）。
- 万一顾客想要双倍摩卡咖啡，怎么办？

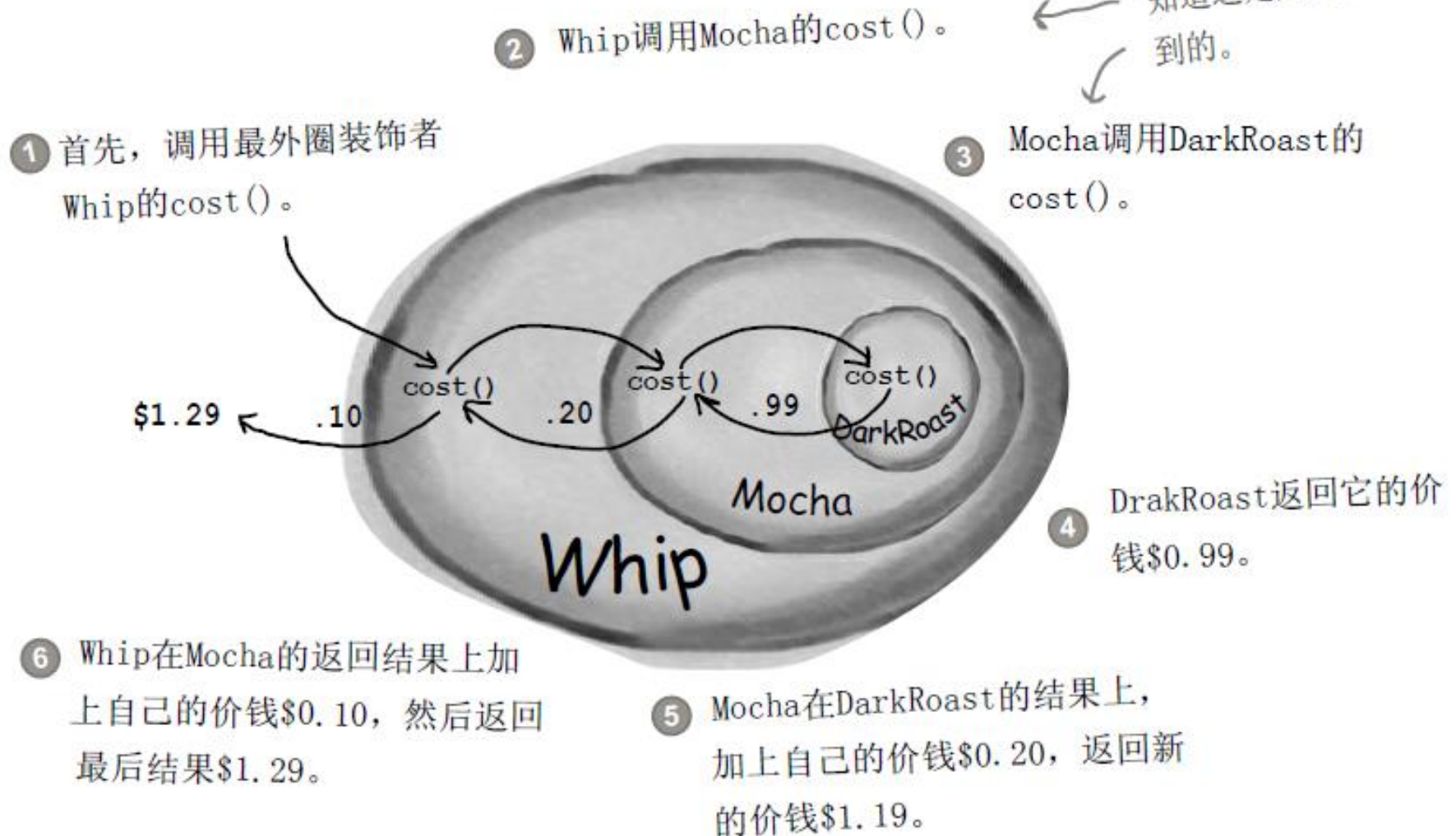


# 认识装饰者模式

- 我们要以**饮料**为主体，然后在运行时以**调料**来“装饰”（decorate）饮料。比方说，如果顾客想要摩卡和奶泡深焙咖啡，那么，要做的是：
- 1 拿一个深焙咖啡（**DarkRoast**）对象
- 2 以摩卡（**Mocha**）对象装饰它
- 3 以奶泡（**Whip**）对象装饰它
- 4 调用cost()方法，并依赖委托（**delegate**）将调料的价钱加上。



再过几页，你就会知道这是如何办到的。



每个组件都可以单独使用，或者被装饰者包起来使用。

ConcreteComponent是我们将要动态地加上新行为的对象，它扩展自Component。

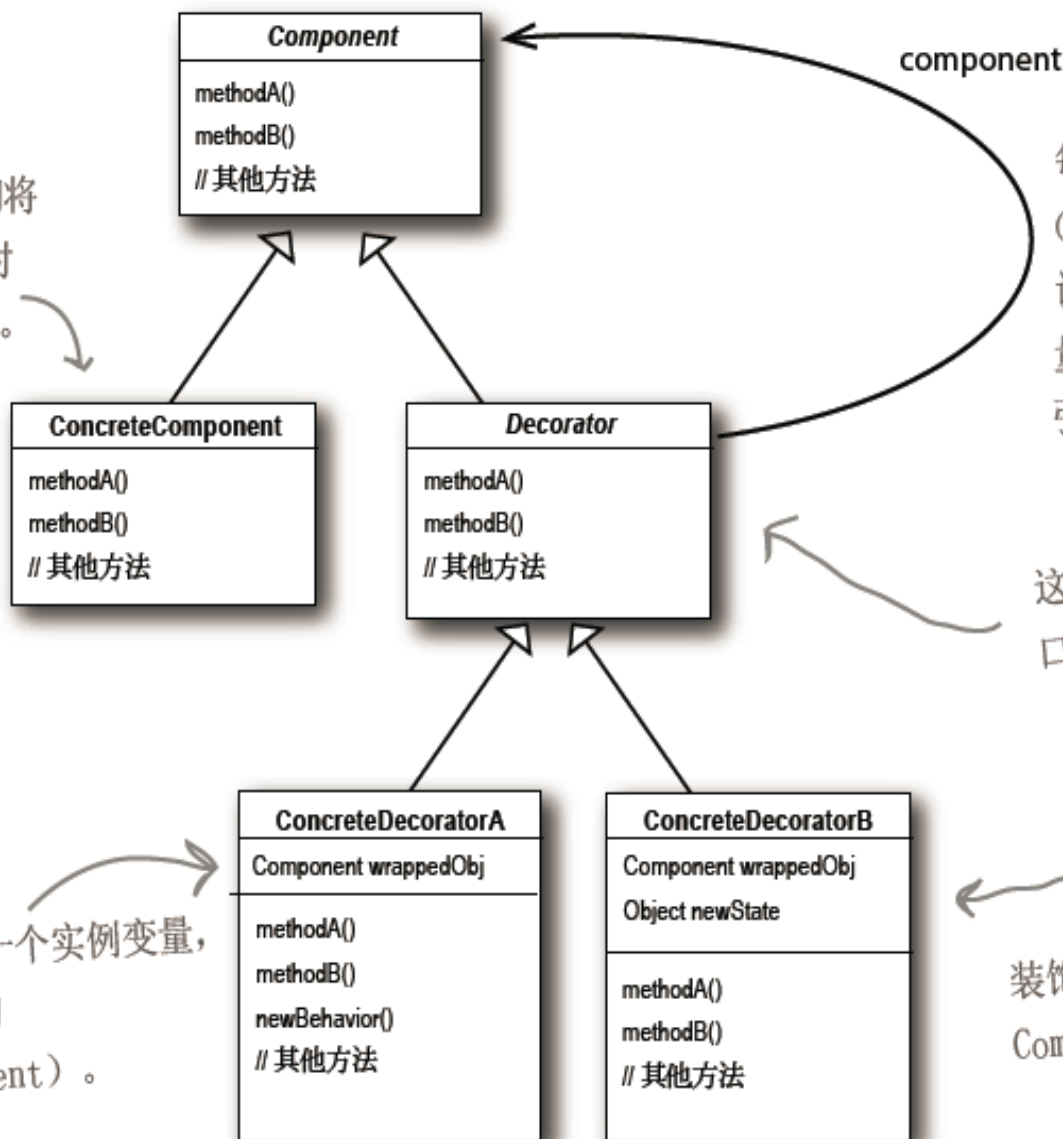
每个装饰者都“有一个”（包装一个）组件，也就是说，装饰者有一个实例变量以保存某个Component的引用。

这是装饰者共同实现的接口（也可以是抽象类）。

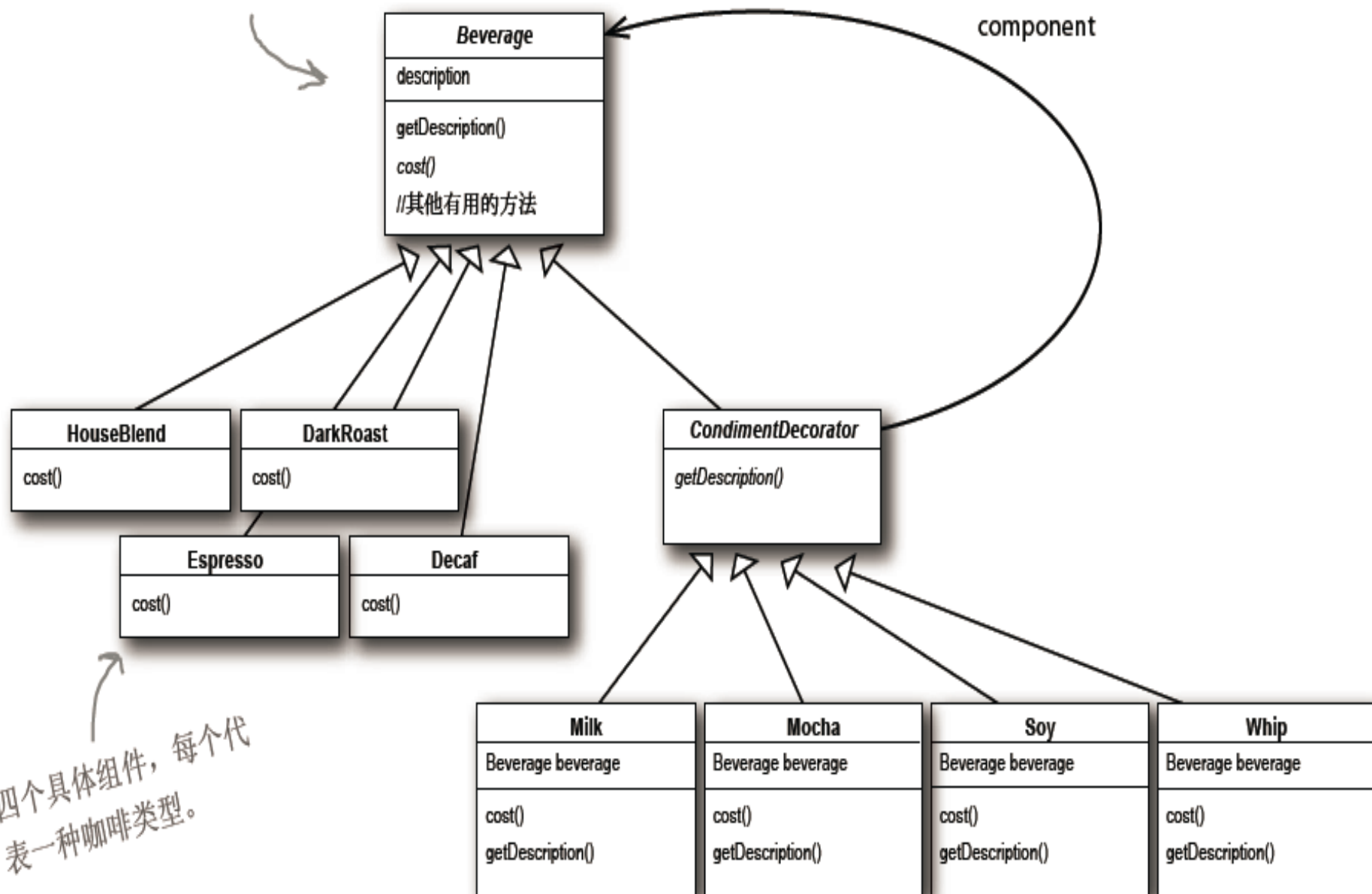
ConcreteDecorator有一个实例变量，可以记录所装饰的事物（装饰者包着的Component）。

装饰者可以扩展Component的状态。

装饰者可以加上新的方法。新行为是通过在旧行为前面或后面做一些计算来添加的。



Beverage相当于抽象的  
Component类。



```
public class Mocha extends CondimentDecorator {  
    Beverage beverage;  
  
    public Mocha(Beverage beverage) {  
        this.beverage = beverage;  
    }  
  
    public String getDescription() {  
        return beverage.getDescription() + ", Mocha";  
    }  
  
    public double cost() {  
        return .20 + beverage.cost();  
    }  
}
```

这是用来下订单的一些测试代码★:

```
public class StarbuzzCoffee {
```

```
    public static void main(String args[]) {  
        Beverage beverage = new Espresso();  
        System.out.println(beverage.getDescription()  
            + " $" + beverage.cost());
```

订一杯Espresso, 不需要调料, 打印  
出它的描述与价钱。

```
        Beverage beverage2 = new DarkRoast();
```

制造出一个DarkRoast对象。

```
        beverage2 = new Mocha(beverage2);
```

用Mocha装饰它。

```
        beverage2 = new Mocha(beverage2);
```

用第二个Mocha装饰它。

```
        beverage2 = new Whip(beverage2);
```

用Whip装饰它。

```
        System.out.println(beverage2.getDescription()  
            + " $" + beverage2.cost());
```

```
        Beverage beverage3 = new HouseBlend();
```

```
        beverage3 = new Soy(beverage3);
```

```
        beverage3 = new Mocha(beverage3);
```

```
        beverage3 = new Whip(beverage3);
```

最后, 再来一杯调料为豆浆、摩  
卡、奶泡的HouseBlend咖啡。

```
        System.out.println(beverage3.getDescription()  
            + " $" + beverage3.cost());
```

```
    }
```