## 装饰（Decorator）

### Intent

为对象动态添加功能。

### Class Diagram

装饰者（Decorator）和具体组件（ConcreteComponent）都继承自组件（Component），具体组件的方法实现不需要依赖于其它对象，而装饰者组合了一个组件，这样它可以装饰其它装饰者或者具体组件。所谓装饰，就是把这个装饰者套在被装饰者之上，从而动态扩展被装饰者的功能。装饰者的方法有一部分是自己的，这属于它的功能，然后调用被装饰者的方法实现，从而也保留了被装饰者的功能。可以看到，具体组件应当是装饰层次的最低层，因为只有具体组件的方法实现不需要依赖于其它对象。

### Implementation

设计不同种类的饮料，饮料可以添加配料，比如可以添加牛奶，并且支持动态添加新配料。每增加一种配料，该饮料的价格就会增加，要求计算一种饮料的价格。

下图表示在 DarkRoast 饮料上新增新添加 Mocha 配料，之后又添加了 Whip 配料。DarkRoast 被 Mocha 包裹，Mocha 又被 Whip 包裹。它们都继承自相同父类，都有 cost() 方法，外层类的 cost() 方法调用了内层类的 cost() 方法。

public interface Beverage {
 double cost();
}

public class DarkRoast implements Beverage {
 @Override
 public double cost() {
 return 1;
 }
}

public class HouseBlend implements Beverage {
 @Override
 public double cost() {
 return 1;
 }
}

public abstract class CondimentDecorator implements Beverage {
 protected Beverage beverage;
}

public class Milk extends CondimentDecorator {

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

 @Override
 public double cost() {
 return 1 + beverage.cost();
 }
}

public class Mocha extends CondimentDecorator {

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

 @Override
 public double cost() {
 return 1 + beverage.cost();
 }
}

public class Client {

 public static void main(String[] args) {
 Beverage beverage = new HouseBlend();
 beverage = new Mocha(beverage);
 beverage = new Milk(beverage);
 System.out.println(beverage.cost());
 }
}

3.0

### 设计原则

类应该对扩展开放，对修改关闭：也就是添加新功能时不需要修改代码。饮料可以动态添加新的配料，而不需要去修改饮料的代码。

不可能把所有的类设计成都满足这一原则，应当把该原则应用于最有可能发生改变的地方。

### JDK

* java.io.BufferedInputStream(InputStream)
* java.io.DataInputStream(InputStream)
* java.io.BufferedOutputStream(OutputStream)
* java.util.zip.ZipOutputStream(OutputStream)
* java.util.Collections#checkedList|Map|Set|SortedSet|SortedMap