3.装饰者模式

装饰者模式用于解决需要新建大量的类

星巴克咖啡订单项目（咖啡馆）：

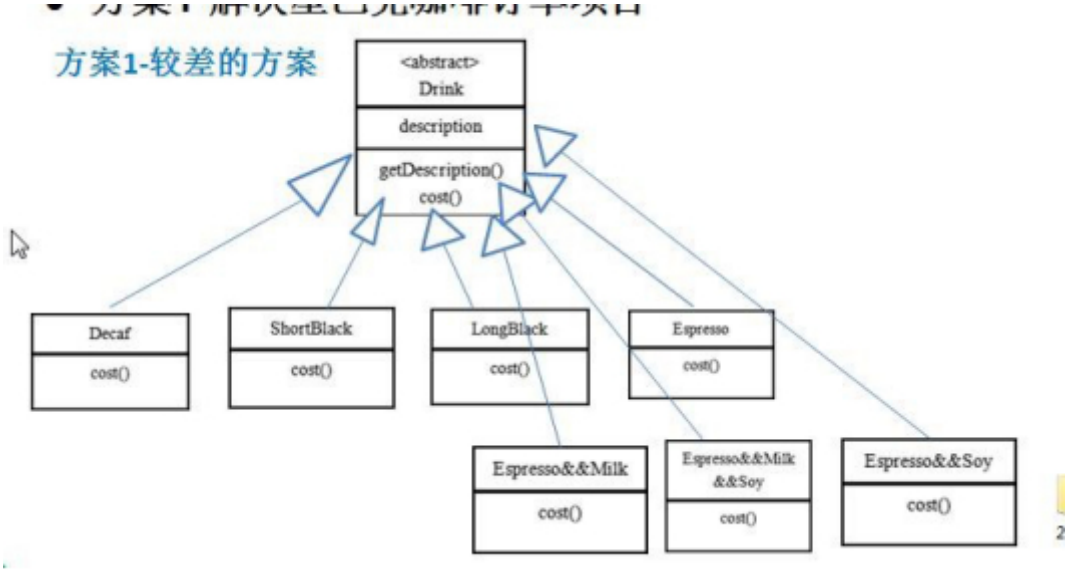
1） 咖啡种类/单品咖啡：Espresso(意大利浓咖啡)、ShortBlack、LongBlack(美式咖啡)、Decaf(无因咖啡)

2） 调料：Milk、Soy(豆浆)、Chocolate

3） 要求在扩展新的咖啡种类时，具有良好的扩展性、改动方便、维护方便

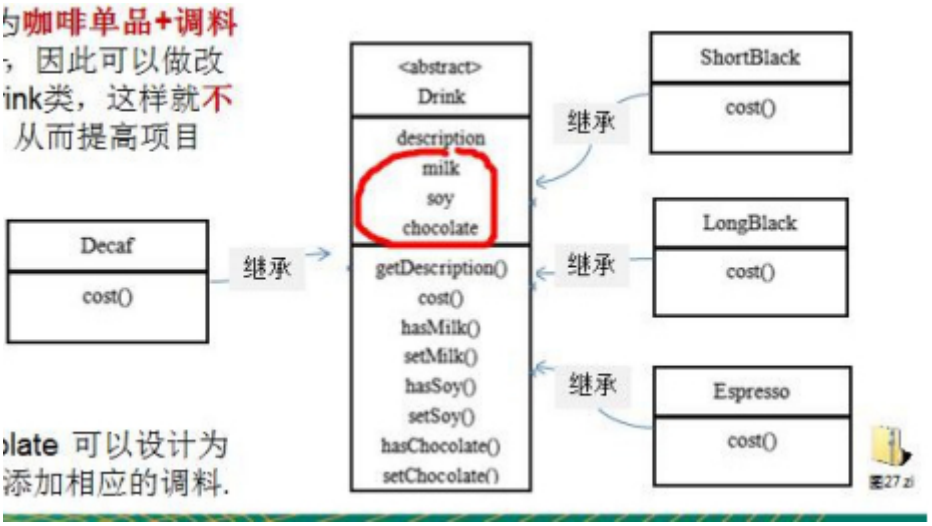
4） 使用 OO 的来计算不同种类咖啡的费用： 客户可以点单品咖啡，也可以单品咖啡+调料组合。

1） 问题：这样设计，会有很多类，当我们增加一个单品咖啡，或者一个新的调料，类的数量就会倍增，就会出现类爆炸



方案 2-解决星巴克咖啡订单(好点)

1） 前面分析到方案 1 因为咖啡单品+调料组合会造成类的倍增，因此可以做改进，将调料内置到 Drink 类，这样就不会造成类数量过多。从而提高项目的维护性(如图)



1） 说明：milk,soy,chocolate 可以设计为 Boolean,表示是否要添加相应的调料.

方案 2-解决星巴克咖啡订单问题分析

1） 方案 2 可以控制类的数量，不至于造成很多的类

2） 在增加或者删除调料种类时，代码的维护量很大

3） 考虑到用户可以添加多份 调料时，可以将 hasMilk 返回一个对应 int

4） 考虑使用 装饰者 模式

装饰者模式定义

1） 装饰者模式：动态的将新功能附加到对象上。在对象功能扩展方面，它比继承更有弹性，装饰者模式也体现了开闭原则(ocp)

2） 这里提到的动态的将新功能附加到对象和 ocp 原则，在后面的应用实例上会以代码的形式体现，请同学们注意体会。

装饰者模式原理

1） 装饰者模式就像打包一个快递

主体：比如：陶瓷、衣服 (Component) // 被装饰者

包装：比如：报纸填充、塑料泡沫、纸板、木板(Decorator)

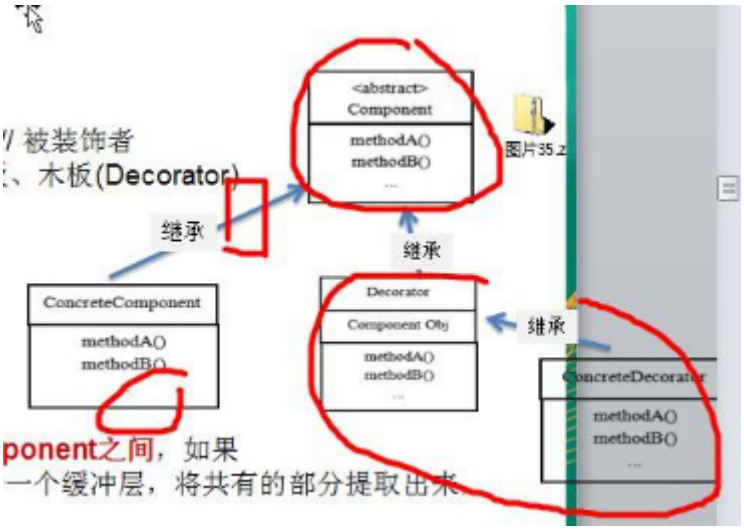
2） Component 主体：比如类似前面的 Drink

3） ConcreteComponent 和 Decorator

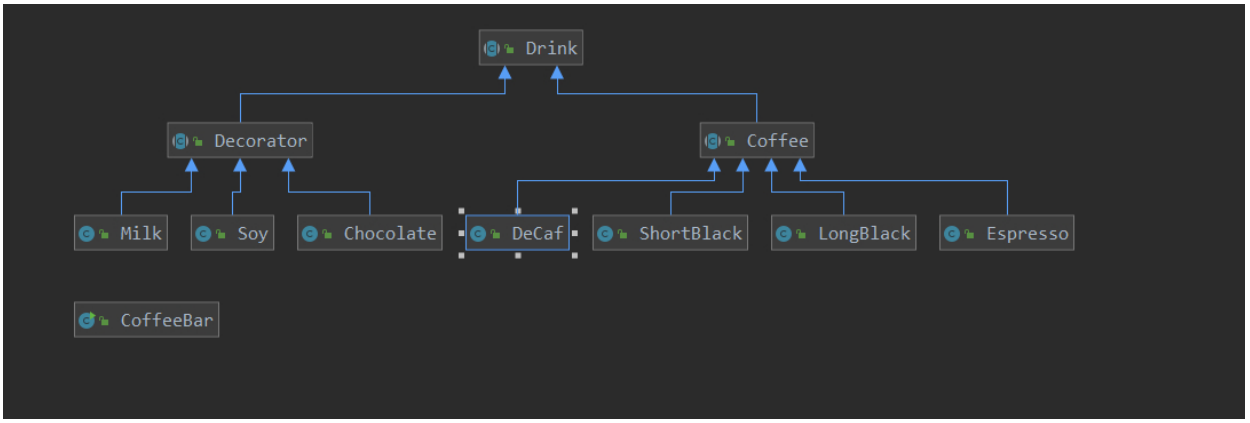
ConcreteComponent：具体的主体， 比如前面的各个单品咖啡

4) Decorator：装饰者，比如各调料.

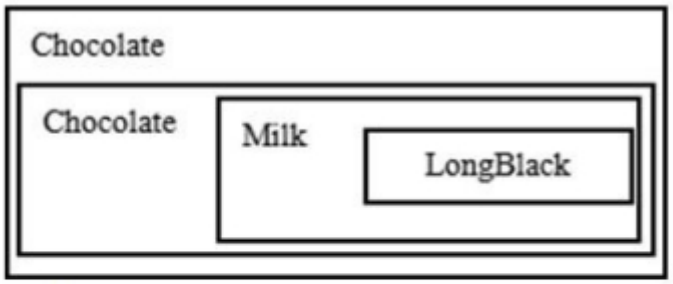
在如图的 Component 与 ConcreteComponent 之间，如果 ConcreteComponent 类很多,还可以设计一个缓冲层，将共有的部分提取出来，抽象层一个类。



装饰者模式解决星巴克咖啡订单



装饰者模式下的订单：2 份巧克力+一份牛奶的 LongBlack



说明

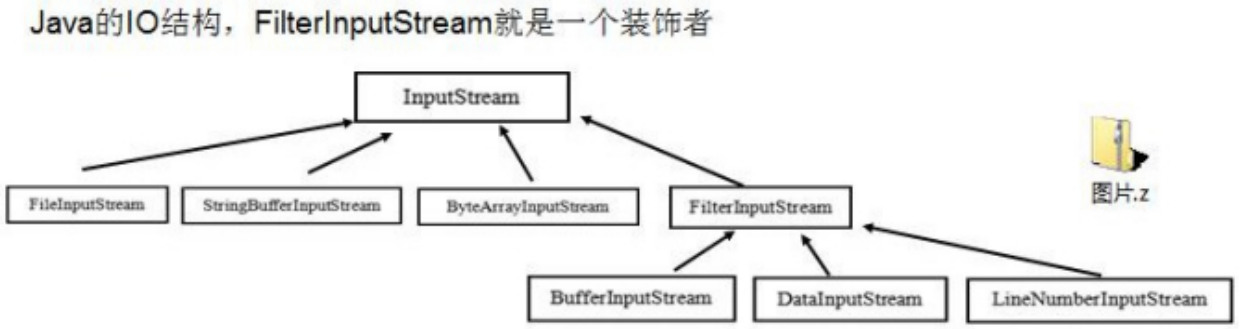
1) Milk包含了LongBlack
2) 一份Chocolate包含了(Milk+LongBlack)
3) 一份Chocolate包含了(Chocolate+Milk+LongBlack)
4) 这样不管是什么形式的单品咖啡+调料组合，通过递归方式可以方便的组合和维护。

```
public abstract class Drink {
  public String des; // 描述
  private float price = 0.0f;
  public String getDes() {
    return des;
  }
  public void setDes(String des) {
    this.des = des;
  }
  public float getPrice() {
    return price;
  }
  public void setPrice(float price) {
    this.price = price;
  }

  //计算费用的抽象方法
  //子类来实现
  public abstract float cost();

}
//抽象主体
public  abstract class Coffee  extends Drink {
  @Override
  public float cost() {
    // TODO Auto-generated method stub
    return super.getPrice();
  }
}
//具体主体
public class LongBlack extends Coffee {
  public LongBlack() {
    setDes(" longblack ");
    setPrice(5.0f);
  }
}
//抽象装饰
public abstract  class Decorator extends Drink {
  private Drink obj;

  public Decorator(Drink obj) { //聚合
    this.obj = obj;
  }

  @Override
  public float cost() {
    // getPrice 自己价格
```

```
        return super.getPrice() + obj.cost();
    }

    @Override
    public String getDes() {
        // obj.getDes() 输出被装饰者的信息
        return des + " " + getPrice() + " && " + obj.getDes();
    }
}
//具体装饰
public class Milk extends Decorator {
    public Milk(Drink obj) {
        super(obj);
        setDes(" 牛奶 ");
        setPrice(2.0f);
    }
}
```

装饰者模式在 JDK 应用的源码分析

Java 的 IO 结构，FilterInputStream 就是一个装饰者



1.InputStream 是抽象类， 类似我们前面讲的 Drink

2. FileInputStream 是 InputStream 子类，类似我们前面的 DeCaf， LongBlack

3. FilterInputStream 是 InputStream 子类：类似我们前面 的 Decorator 修饰者

4. DataInputStream 是 FilterInputStream 子类，具体的修饰者，类似前面的 Milk, Soy 等

5. FilterInputStream 类 有 protected volatile InputStream in; 即含被装饰者

6. 分析得出在 jdk 的 io 体系中，就是使用装饰者模式

装饰者模式和桥接模式的不同

1。桥接模式中所说的分离，其实是指将结构与实现分离（当结构和实现有可能发生变化时）或属性与基于属性的行为进行分离；而装饰者只是对基于属性的行为进行封闭成独立的类。

2。桥接中的行为是横向的行为，行为彼此之间无关联；而装饰者模式中的行为具有可叠加性，其表现出来的结果是一个整体，一个各个行为组合后的一个结果。

装饰者模式和享元模式的不同

装饰者模式用于解决需要新建大量的类

而享元模式是用于解决需要创建大量类似的对象