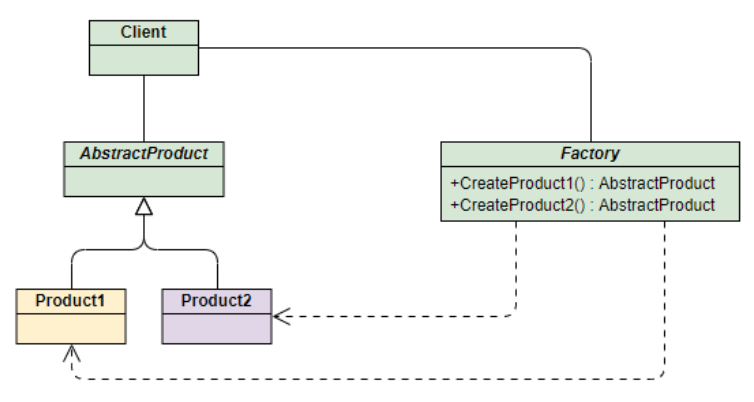
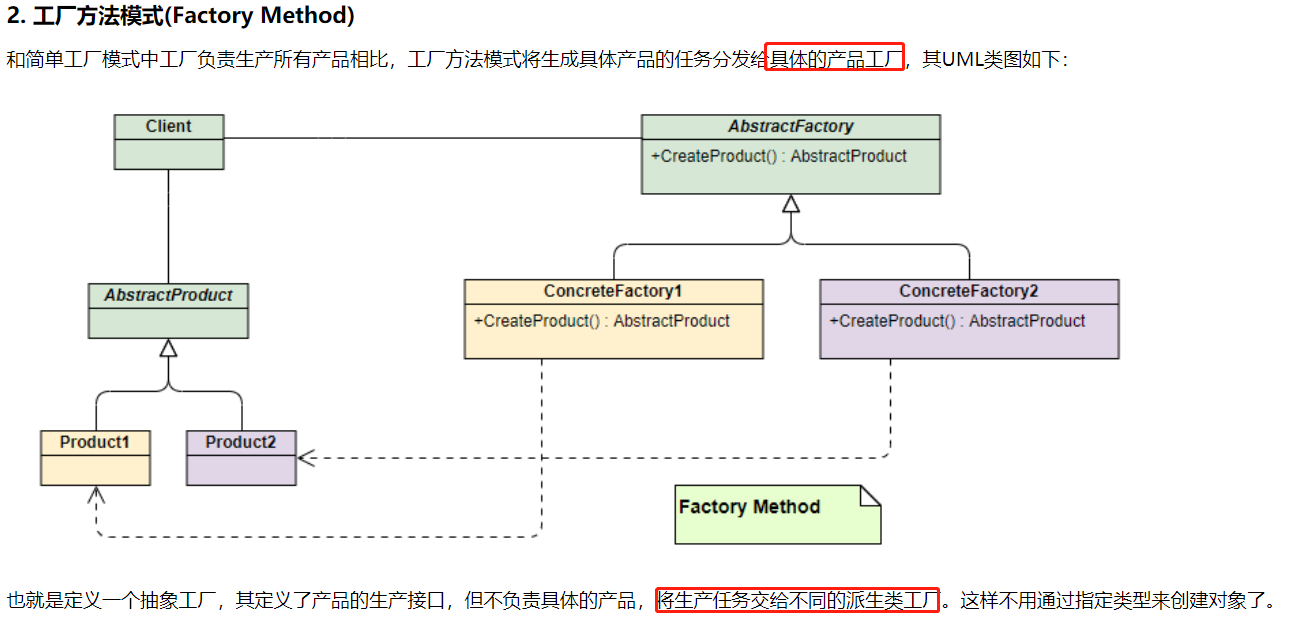


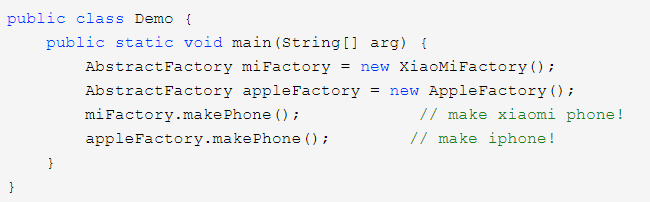
**工厂模式**

该模式对对象创建管理方式最为简单，因为其仅仅简单的对不同类对象的创建进行了一层薄薄的封装。该模式通过向工厂传递类型来指定要创建的对象

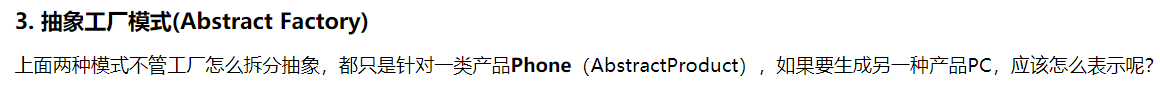


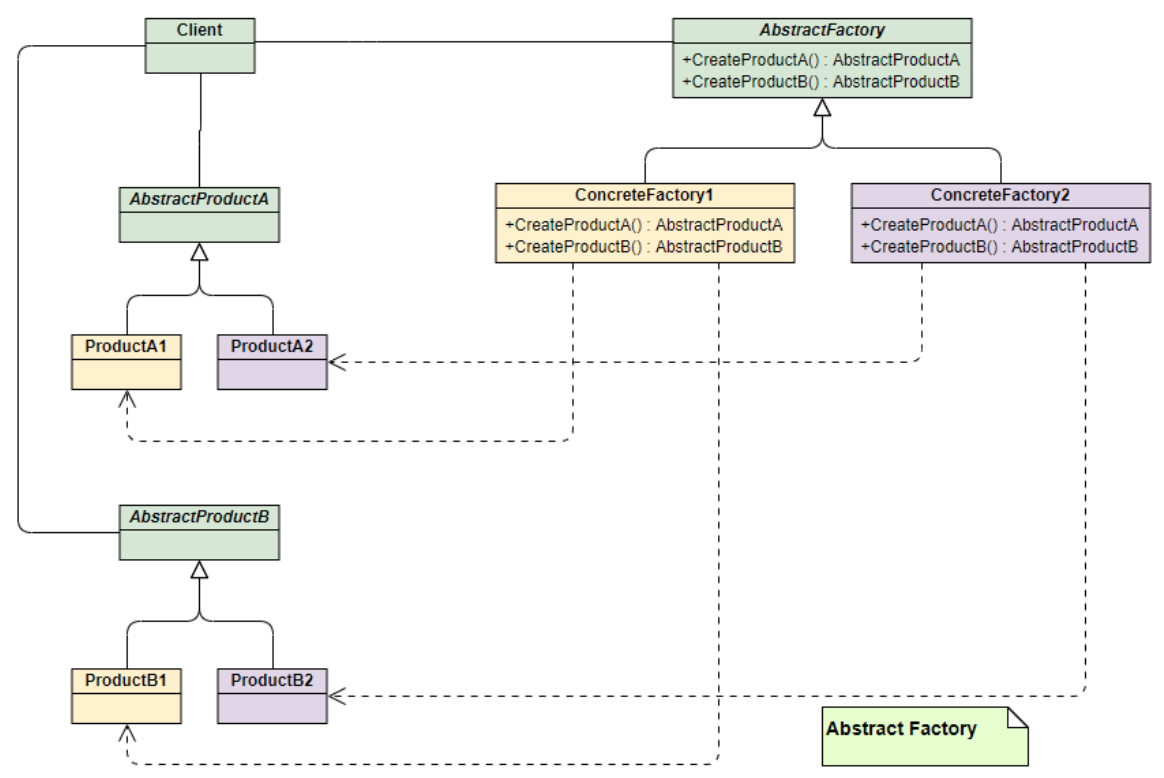
即为一个工厂既可以生成苹果手机也可以生成华为手机

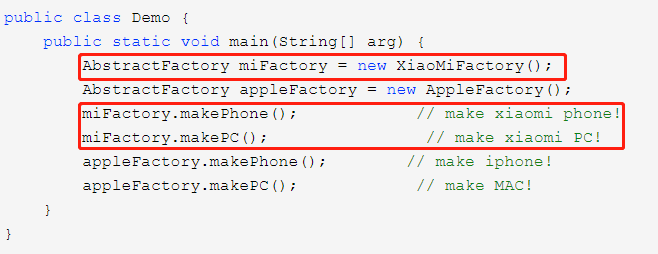




即为苹果手机再苹果工厂生成，华为手机在华为工厂生产

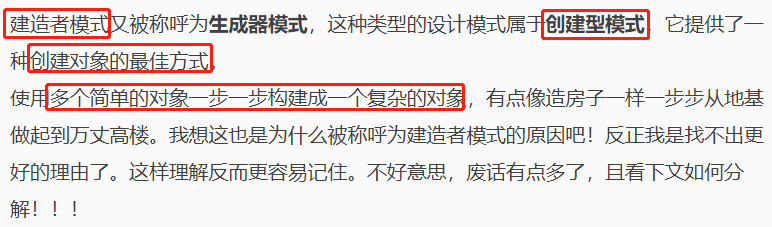




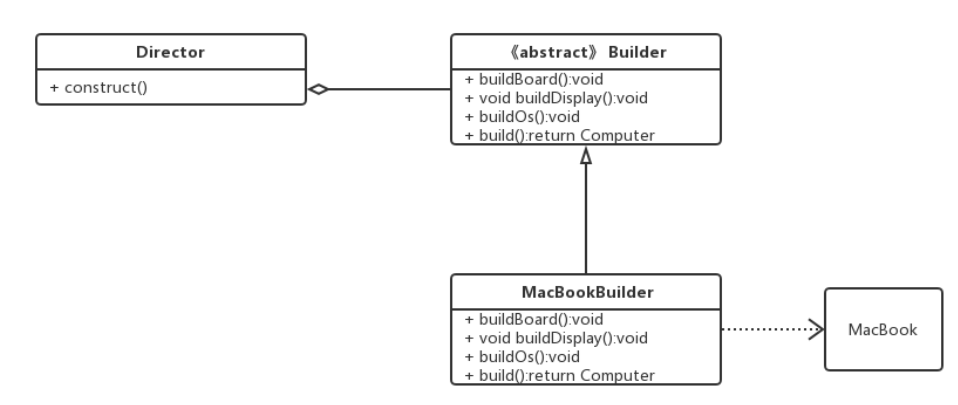


抽象工厂模式在工厂模式的基础上多实现一个接口方法

<https://www.cnblogs.com/yssjun/p/11102162.html>





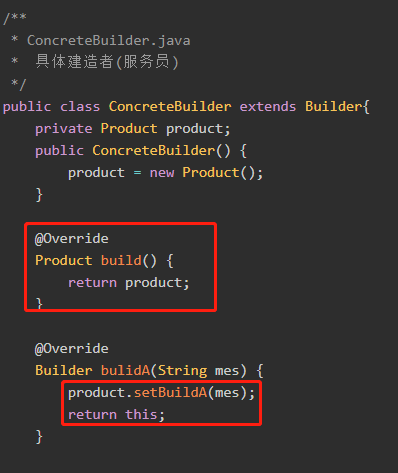


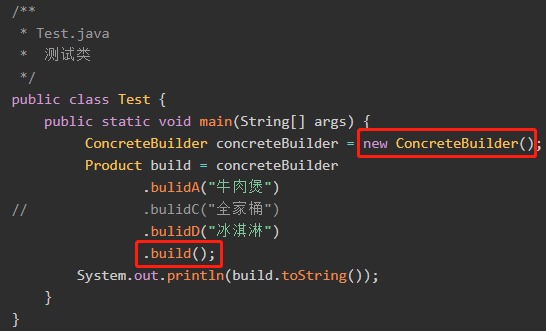
Builder定义产品中各个组件的抽象方法

Direct即为产品，里面get和set组件的值

MacBookBuilder工厂实现类，里面定义产品对象，与组件变量，通过产品的set保存数据，最后build这个产品。

MacBook具体实现类

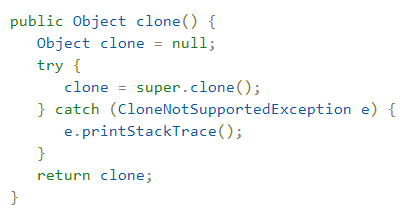


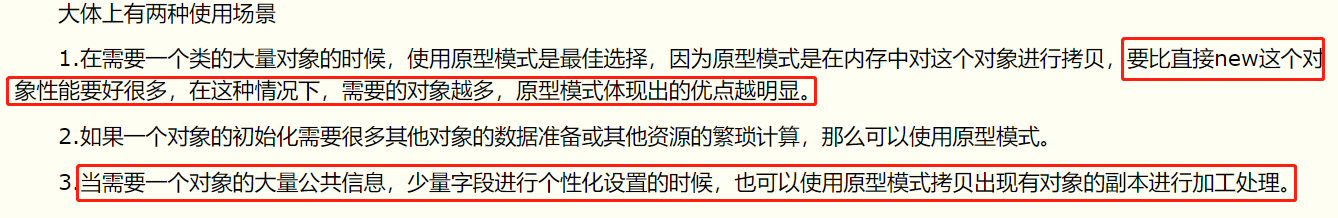


<https://www.jianshu.com/p/47329a94f5dc>

原型模式（Prototype Pattern）是用于创建重复的对象，同时又能保证性能。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。

实现克隆操作，在 JAVA 实现 Cloneable，重写 clone()



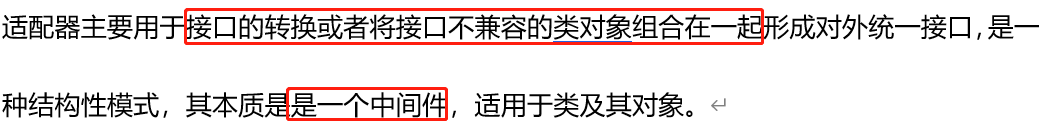


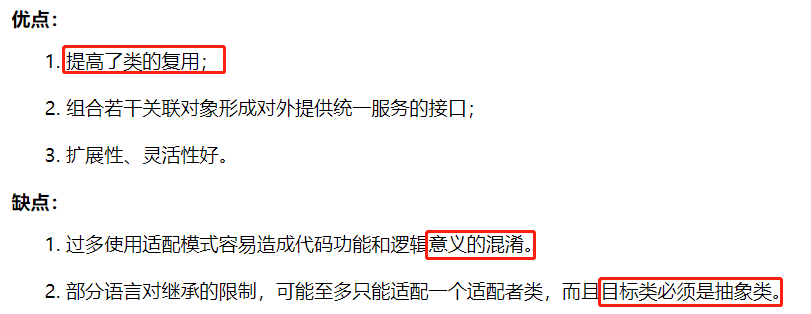
<https://www.cnblogs.com/fengyumeng/p/10646487.html>

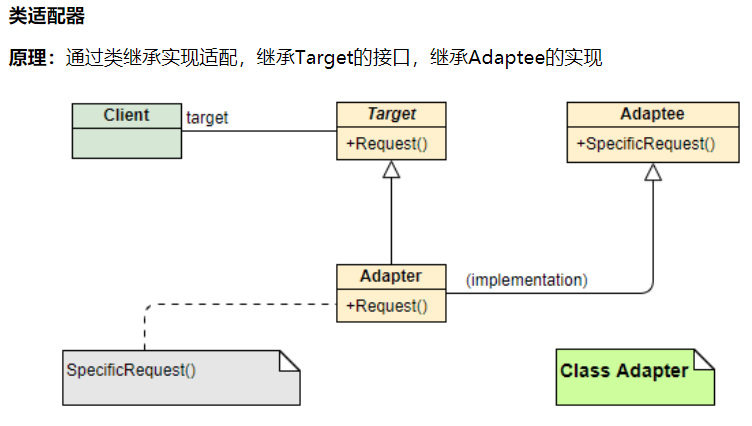


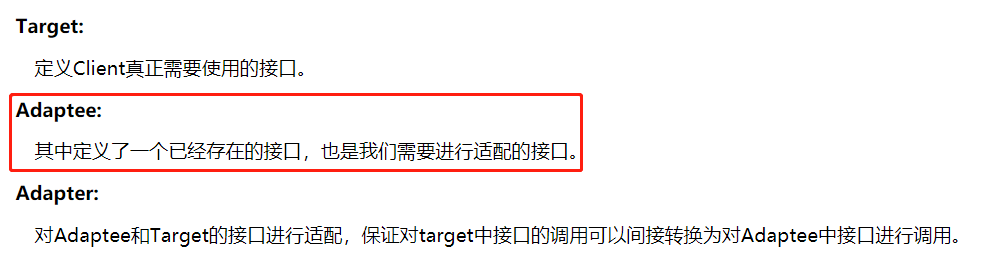
原型模式是浅拷贝

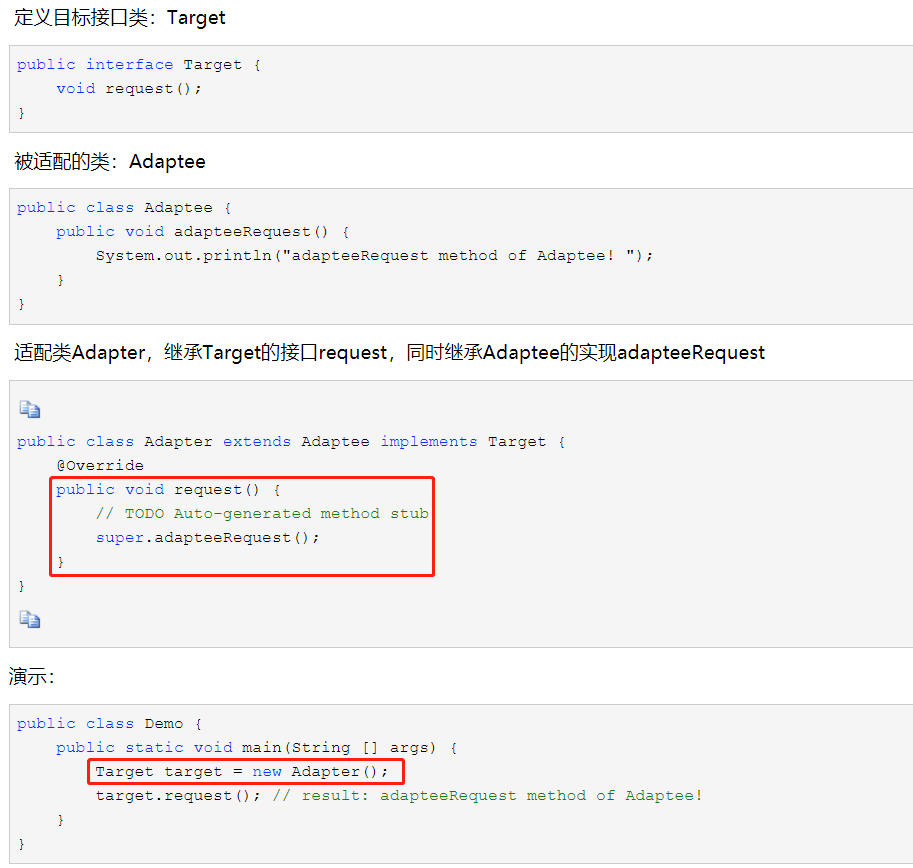




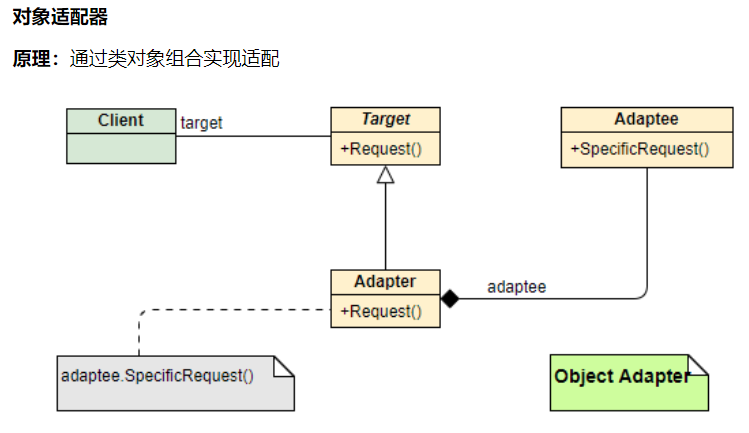




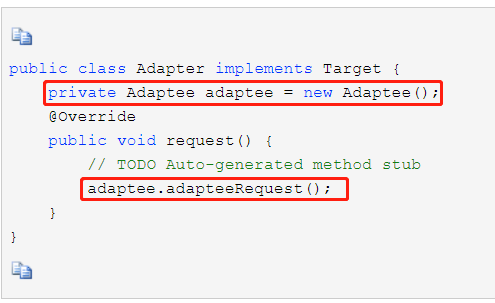




一个方法只能处理某个对象类型为A的对象，此时拥有的对象类型的B对象，通过适配器把B对象包装完转化为A对象实现方法的正常运行

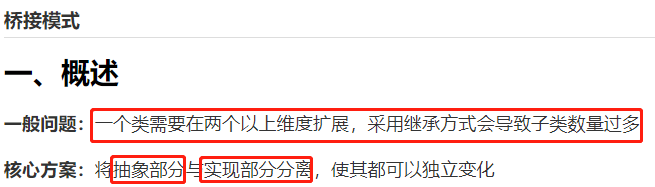


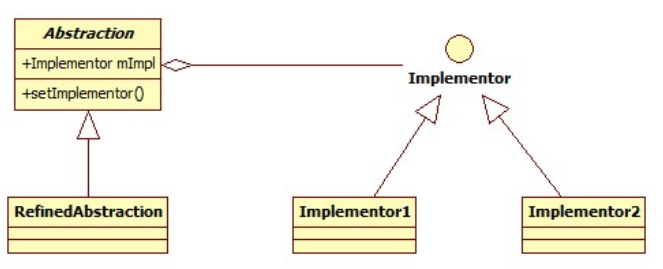
两张UML图中可以清楚的看出两者的区别，对象中Adapter不在继承Adaptee，而是将Adaptee作为一个数据成员组合到类定义中，从而实现对其接口的访问。



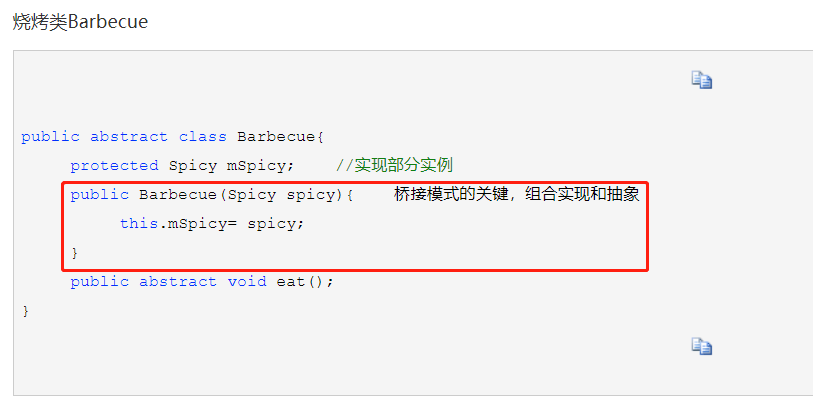
一般适配器模式用于代码已经开发七七八八了，突然更改需求，此时修改接口影响范围太大，使用适配器模式进行兼容

<https://www.cnblogs.com/yssjun/p/11094401.html>

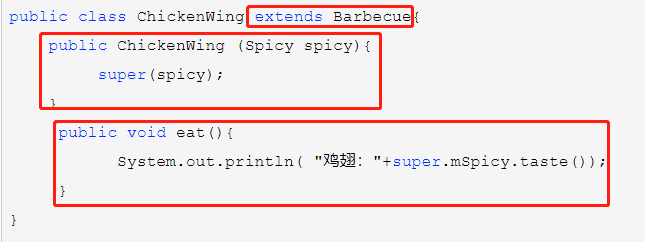


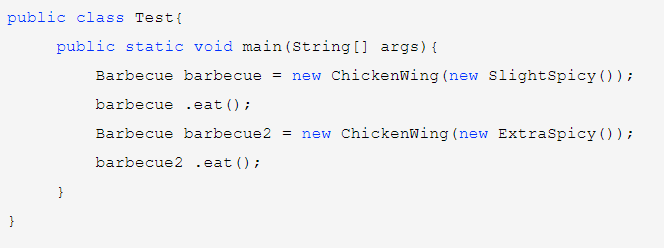


现实部分是已经确定的（实现接口），抽象部分不确定，交给子类去实现（用抽象类定义）



桥接的类，具体的部分作为参数传入，，eat是不确定的抽象的交给子类实现





就比如开一家咖啡店，店里的饮料大小是确定的（大杯小杯中杯），可以分配实现3个杯类。而咖啡种类是不确定的，以后还可以无限扩展种类，所以定为抽象咖啡类，而子类实现抽象类的咖啡类型得到具体的咖啡对象比如卡布奇诺对象，拿铁对象，美式对象等。

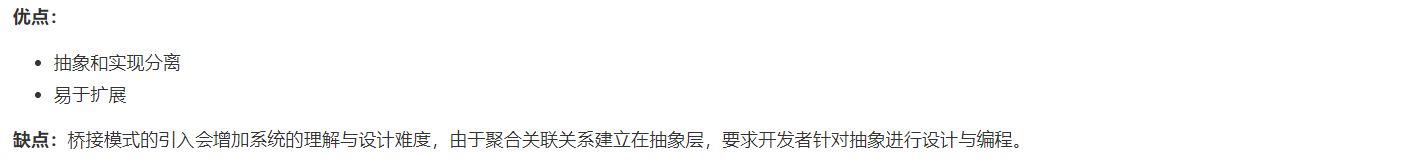
这里面的桥就是咖啡类，承接了具体的杯容量和不确定的咖啡类型。

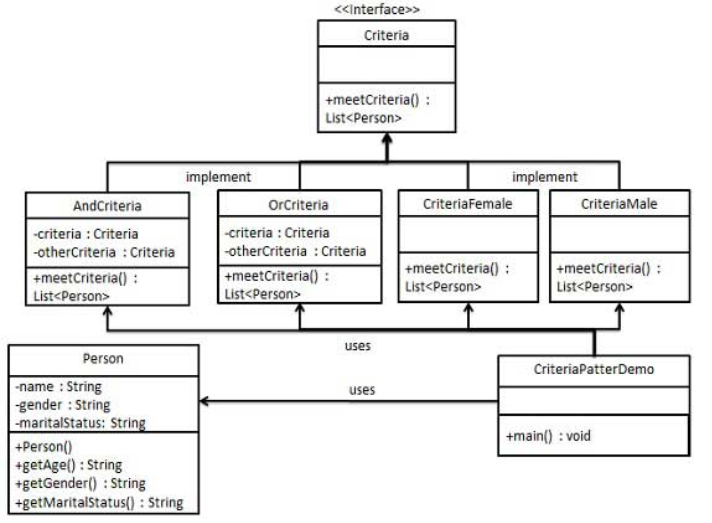
当然具体开发时候场景没那么简单设计模式经常是结合使用的，

比如加榛果和牛奶的卡布奇诺，加奥利奥和奶盖的拿铁。这些就可以用到建造者模式了，

食品固体添加物和液态添加物和咖啡种类等是一个一个的小对象，一起组合成一个具体的咖啡对象（榛果牛奶卡布奇诺）

<https://www.cnblogs.com/not2/p/11017132.html>

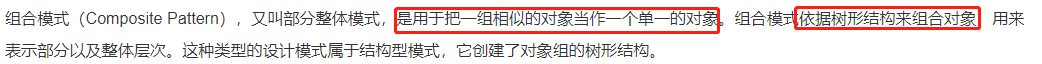






一个比较简单的设计模式，对一个对象进行过滤，有一个过滤类接口，不同的过滤功能类实现这个接口传入对象从而对对象进行过滤

<https://www.runoob.com/design-pattern/filter-pattern.html>



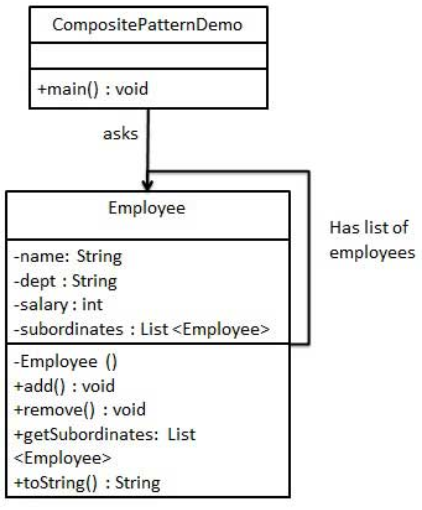
在以往项目中大量使用组合设计模式变种，即为父类包含子类集合对象，子类属性中又包含父类对象（正常不变种是一个类包含该类的集合对象，简单来说就是一棵树）

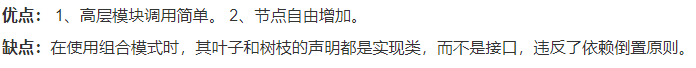


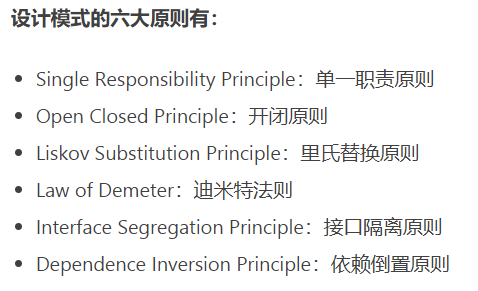
CEO页节点是部门经理，部门经理叶结点是各个社畜

实现起来也简单，类里包含该类的集合就行了，然后类中有对集合的增删改查等操作

<https://www.runoob.com/design-pattern/composite-pattern.html>







**里氏替换原则**：子类可以扩展父类的功能，但不能改变原有父类的功能