一 建造者模式实现kfc

 KFC里面一般都有好几种可供客户选择的套餐，它可以根据客户所点的套餐，然后在后面做这些套餐，返回给客户的事一个完整的、美好的套餐。下面我们将会模拟这个过程，我们约定套餐主要包含汉堡、薯条、可乐、鸡腿等等组成部分，使用不同的组成部分就可以构建出不同的套餐。

       首先是套餐类：Meal.**[Java](http://lib.csdn.net/base/17" \o "Java EE知识库" \t "http://blog.csdn.net/chenssy/article/details/_blank)**

**package** kfc;

**public** **class** Meal {

**private** String food;

**private** String drink;

**public** String getFood() {

**return** food;

}

**public** **void** setFood(String food) {

**this**.food = food;

}

**public** String getDrink() {

**return** drink;

}

**public** **void** setDrink(String drink) {

**this**.drink = drink;

}

}

       然后是套餐构造器：MealBuilder.java

**package** kfc;

**public** **abstract** **class** MealBuilder {

Meal meal = **new** Meal();

**public** **abstract** **void** buildFood();

**public** **abstract** **void** buildDrink();

**public** Meal getMeal(){

**return** meal;

}

}  
      然后是套餐A、套餐B。这个两个套餐都是实现抽象套餐类。

**package** kfc;

**public** **class** MealA **extends** MealBuilder{

**public** **void** buildDrink() {

meal.setDrink("一杯西瓜果汁");

}

**public** **void** buildFood() {

meal.setFood("三个鸡腿");

}

}

**package** kfc;

**public** **class** MealB **extends** MealBuilder{

**public** **void** buildDrink() {

meal.setDrink("一杯柠檬果汁");

}

**public** **void** buildFood() {

meal.setFood("三个鸡翅");

}

}

      最后是KFC的服务员，它相当于一个指挥者，它决定了套餐是的实现过程，然后给你一个完美的套餐。

**public** **class** KFCWaiter {

**private** MealBuilder mealBuilder;

**public** **void** setMealBuilder(MealBuilder mealBuilder) {

**this**.mealBuilder = mealBuilder;

}

**public** Meal construct(){

//准备食物

mealBuilder.buildFood();

//准备饮料

mealBuilder.buildDrink();

//准备完毕，返回一个完整的套餐给客户

**return** mealBuilder.getMeal();

}

}  
       测试类

**package** kfc;

**public** **class** Client {

**public** **static** **void** main(String[] args) {

//服务员

KFCWaiter waiter = **new** KFCWaiter();

//套餐A

MealA a = **new** MealA();

//服务员准备套餐A

waiter.setMealBuilder(a);

//获得套餐

Meal mealA = waiter.construct();

System.***out***.print("套餐A的组成部分:");

System.***out***.println(mealA.getFood()+"---"+mealA.getDrink());

}

}

      运行结果：

       套餐A的组成部分:三个鸡腿---一杯西瓜果汁

# 模式优缺点

## 优点

       1、将复杂产品的创建步骤分解在不同的方法中，使得创建过程更加清晰，使得我们能够更加精确的控制复杂对象的产生过程。

       2、将产品的创建过程与产品本身分离开来，可以使用相同的创建过程来得到不同的产品。也就说细节依赖抽象。

       3、每一个具体建造者都相对独立，而与其他的具体建造者无关，因此可以很方便地替换具体建造者或增加新的具体建造者，用户使用不同的具体建造者即可得到不同的产品对象。

## 缺点

       1、建造者模式所创建的产品一般具有较多的共同点，其组成部分相似，如果产品之间的差异性很大，则不适合使用建造者模式，因此其使用范围受到一定的限制。  
       2、如果产品的内部变化复杂，可能会导致需要定义很多具体建造者类来实现这种变化，导致系统变得很庞大。

# 模式适用场景

       1、需要生成的产品对象有复杂的内部结构，这些产品对象通常包含多个成员属性。  
       2、隔离复杂对象的创建和使用，并使得相同的创建过程可以创建不同的产品。

# 模式总结

       1、建造者模式是将一个复杂对象的创建过程给封装起来，客户只需要知道可以利用对象名或者类型就能够得到一个完整的对象实例，而不需要关心对象的具体创建过程。

       2、建造者模式将对象的创建过程与对象本身隔离开了，使得细节依赖于抽象，符合依赖倒置原则。可以使用相同的创建过程来创建不同的产品对象。

二 能否用工厂模式与建造者模式结合实现kfc

首先我觉得是可以的。 工厂方法模式：把一个类的实例化延迟到它的子类中，核心的工厂类不负责所有产品的创建，而是将具体创建的工作交给子类去做。让子类决定实例化哪一个类；建造者模式将一个产品的内部表象和产品的生成过程分开来，从而可以使一个建造过程生成具有不同的内部表象的产品对象。需要构造的对象非常复杂，而调用者不需要知道这些细节。例如模仿一个特殊的编译器，该编译器编译所有源代码，将其链接，然后压缩，最后加密。有多个类：Editor（编辑器）、Builder（构建器）、Compiler（编译器）、Linker（链接器）、Compressor（压缩器）、Encryptor（加密机）；工厂模式一般都是创建一个产品，注重的是把这个产品创建出来就行，只要创建出来，不关心这个产品的组成部分。从代码上看，工厂模式就是一个方法，用这个方法就能生产出产品。建造者模式也是创建一个产品，但是不仅要把这个产品创建出来，还要关系这个产品的组成细节组成过程。所以如果要让两者结合，只需在建造者的代码上在加一层抽象工厂就好了。