2.工厂模式

https://blog.csdn.net/jerry11112/article/details/80618420

1) 工厂模式的意义

将实例化对象的代码提取出来，放到一个类中统一管理和维护，达到和主项目的依赖

关系的解耦。从而提高项目的扩展和维护性。

2) 三种工厂模式（简单工厂模式、工厂方法模式、抽象工厂模式）

3) 设计模式的依赖抽象原则

小结:

1.创建对象实例时，不要直接 new 类，而是把这个 new 类的动作放在一个工厂的方法中，并返回。有的书上说， 变量不要直接持有具体类的引用。

2.不要让类继承具体类，而是继承抽象类或者是实现 interface(接口)

3.不要覆盖基类中已经实现的方法。

三个工厂都是各有利弊，简单工厂违反了最基本的原则，工厂方法与抽象工厂完美的解决了简单工厂的弊端！工厂方法的工厂个数过多，导致系统庞大，抽象工厂增加新的产品族很方便,但是增加新的产品会违反开闭原则！

2.1简单工厂模式

1) 简单工厂模式是属于创建型模式，是工厂模式的一种。简单工厂模式是由一个工

厂对象决定创建出哪一种产品类的实例。简单工厂模式是工厂模式家族中最简单实用

的模式

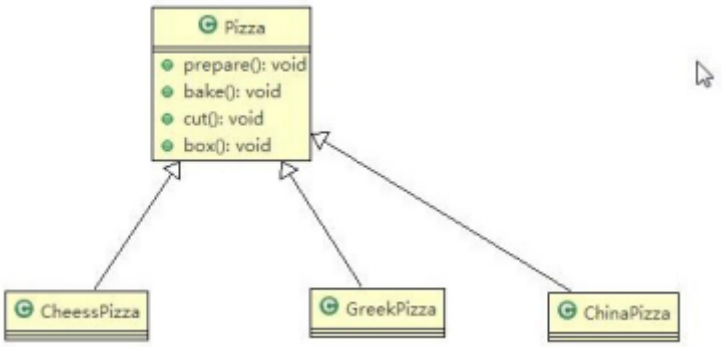
2) 简单工厂模式：定义了一个创建对象的类，由这个类来封装实例化对象的行为(代

码)

3） 在软件开发中，当我们会用到大量的创建某种、某类或者某批对象时，就会使用

到工厂模式.

不使用工厂模式:



1） 优点是比较好理解，简单易操作。

2） 缺点是违反了设计模式的 ocp 原则，即对扩展开放，对修改关闭。即当我们给类增加新功能的时候，尽量不修改代码，或者尽可能少修改代码.

3） 比如我们这时要新增加一个 Pizza 的种类(Pepper 披萨)，我们需要做如下修改. 如果我们增加一个 Pizza 类，只要是订购 Pizza 的代码都需要修改.

1） 改进的思路分析

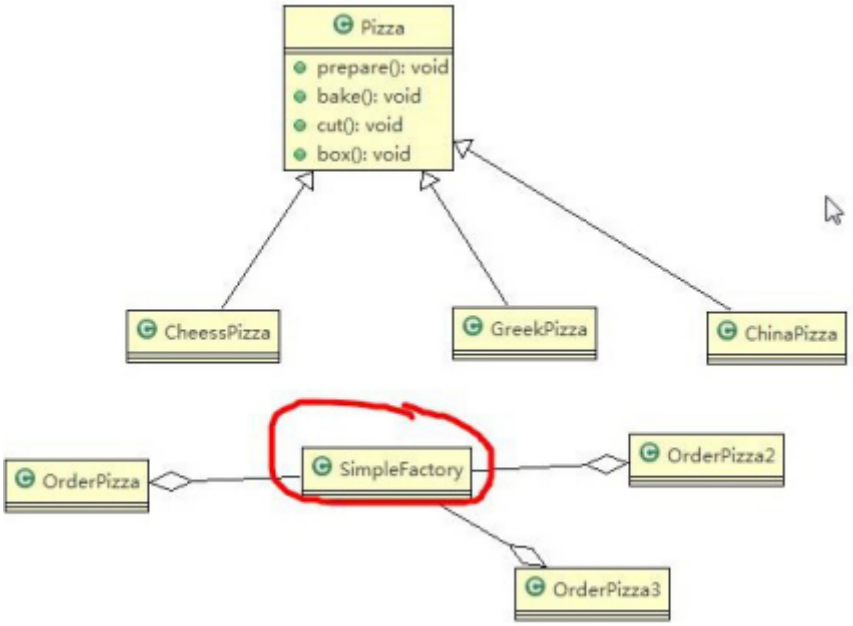
分析：修改代码可以接受，但是如果我们在其它的地方也有创建 Pizza 的代码，就意味着，也需要修改，而创建 Pizza的代码，往往有多处。

思路：把创建 Pizza 对象封装到一个类中，这样我们有新的 Pizza 种类时，只需要修改该类就可，其它有创建到 Pizza

对象的代码就不需要修改了.-> 简单工厂模式

使用简单工厂模式

1） 简单工厂模式的设计方案: 定义一个可以实例化 Pizaa 对象的类，封装创建对象的代码。



(代码看代码文件夹，此处过于简单就不放入代码)

2.2工厂方法模式

披萨项目新的需求：客户在点披萨时，可以点不同口味的披萨，比如 北京的奶酪 pizza、北京的胡椒 pizza 或者是伦敦的奶酪 pizza、伦敦的胡椒 pizza。

方案:

1） 工厂方法模式设计方案：将披萨项目的实例化功能抽象成抽象方法，在不同的口味

点餐子类中具体实现。

2） 工厂方法模式：定义了一个创建对象的抽象方法，由子类决定要实例化的类。工厂

方法模式将对象的实例化推迟到子类。

优点:

（1）工厂方法用来创建客户所需要的产品，同时隐藏了哪种具体产品类将被实

例化的细节，用户只需要要关注工厂，不需要关注创建的细节！从客户端代

码就可以看出！只知道对应的工厂就好！

(2)在增加修改新的运算类的时候不用修改代码，只需要增加对应的工厂就好，完

全符合开放——封闭性原则！

（3）创建对象的细节完全封装在具体的工厂内部，而且有了抽象的工厂类，所有

的具体工厂都继承了自己的父类！完美的体现了多态性！

缺点：

（1）在增加新的产品（对应UML图的算法）时，也必须增加新的工厂类，会带

来额外的开销

（2）抽象层的加入使得理解程度加大

```
//伦敦奶酪披萨
public class LDCheesePizza extends OrderPizza {
  @Override
  Pizza createPizza() {
    return new LDCheesePizza();
  }
}
//伦敦胡椒披萨
public class LDPepperPizza extends OrderPizza {
  @Override
  Pizza createPizza() {
    return new LDCheesePizza();
  }
}
//北京奶酪披萨
public class BJChessPizza extends OrderPizza {
  @Override
  Pizza createPizza() {.....}
}

//北京胡椒披萨
public class BJPepperPizza extends OrderPizza {
  @Override
  Pizza createPizza() {.....}
}
```

2.3抽象工厂模式

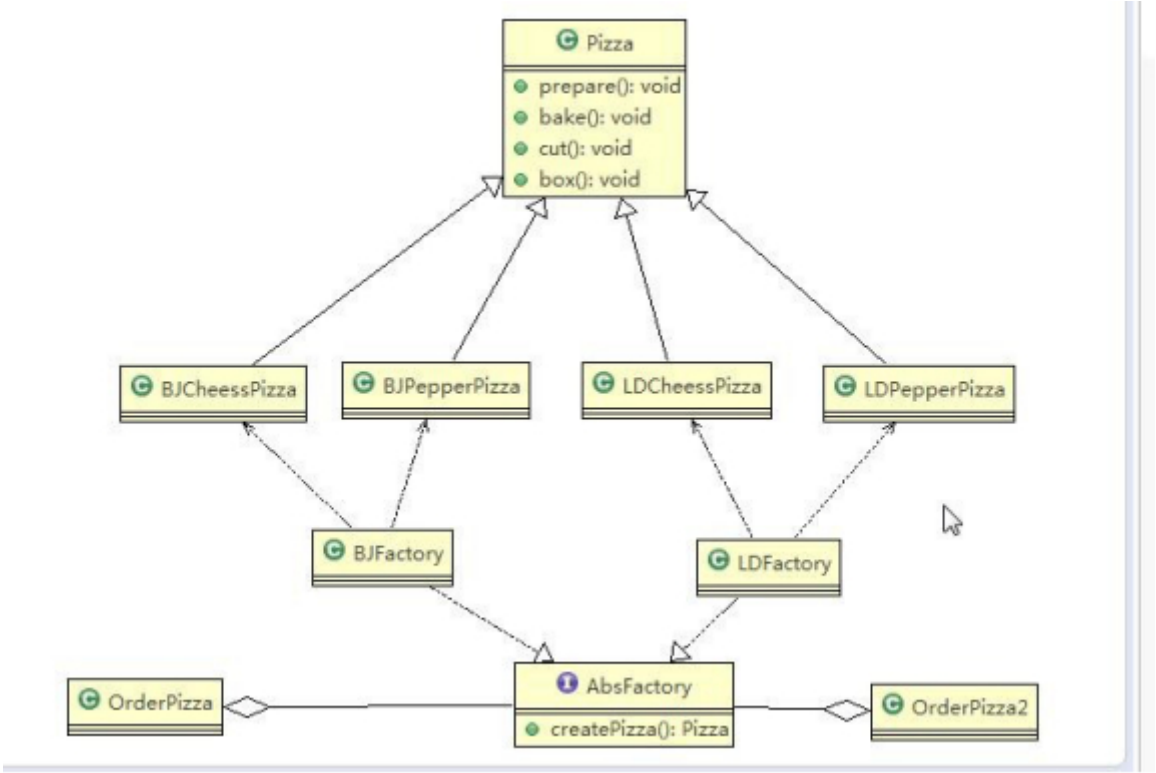
1） 抽象工厂模式：定义了一个 interface 用于创建相关或有依赖关系的对象簇，而无需指明具体的类

2） 抽象工厂模式可以将简单工厂模式和工厂方法模式进行整合。

3） 从设计层面看，抽象工厂模式就是对简单工厂模式的改进(或者称为进一步的抽象)。

4） 将工厂抽象成两层，AbsFactory(抽象工厂) 和 具体实现的工厂子类。程序员可以根据创建对象类型使用对应的工厂子类。这样将单个的简单工厂类变成了工厂簇，更利于代码的维护和扩展。

增加一个披萨族（产品族）很简单，而增加一个新的披萨（产品）就会非常复杂！例如，我要是增加纽约胡椒披萨，纽约奶酪披萨，那么我就可以直接增加了，工厂顺便增加一个纽约工厂，这样完美了的利用好了开放封闭的原则！棒极了！但是我要是增加一个新的披萨，比如是牛肉披萨，那么它就要同时增加招加北京，伦敦，纽约披萨，同时还需要修改factory，违反了开放封闭的原则！




```
interface AbsFactory {
    Pizza createCherryPizza();
    Pizza createPepperPizza();
}
public class LDFactory implements AbsFactory {
   @Override
   public Pizza createCherryPizza() {
      return  new LDCheesePizza();
   }

   @Override
   public Pizza createPepperPizza(){
       return new LDPepperPizza();
   }

}

public class BJFactory implements AbsFactory {
........
}
```