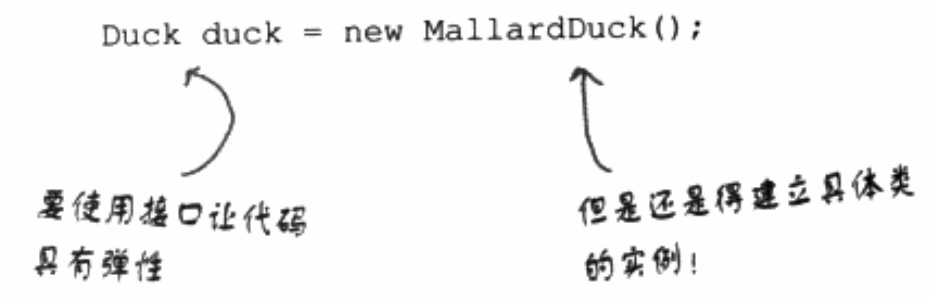
### 1、工厂模式

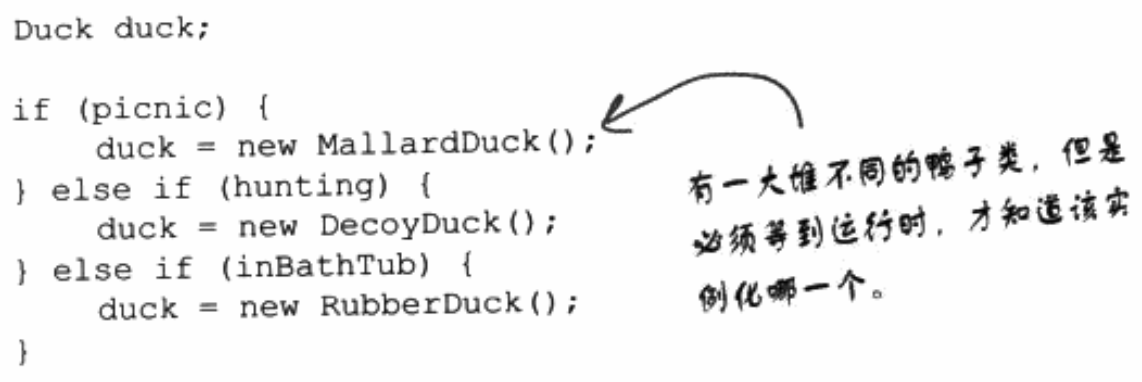
**准备好开始烘烤某些松耦合的OO设计**。除了使用new操作符之外，还有更多制造对象的方法。你将了解到实例化这个活动不应该总是公开地进行，也会认识到初始化经常造成“耦合”问题。了解工厂模式如何从复杂的依赖中帮你脱困。

#### 当看到new，就会想到“具体”

当使用“new”时，的确是在实例化一个具体类，所以用的确实是实现，而不是接口。代码绑着具体类会导致代码更脆弱，缺乏弹性。



当有一群相关类时，通常会写出这样的代码：



这里有一些要实例化的具体类，究竟实例化哪个类，要在运行时由一些条件来决定。

当看到这样的代码，一旦有变化或扩展，就必须重新打开这段代码进行检查和修改。

#### “new”有什么不对劲？

真正的犯人是我们的老朋友“改变”，以及它是如何影响new的使用的。

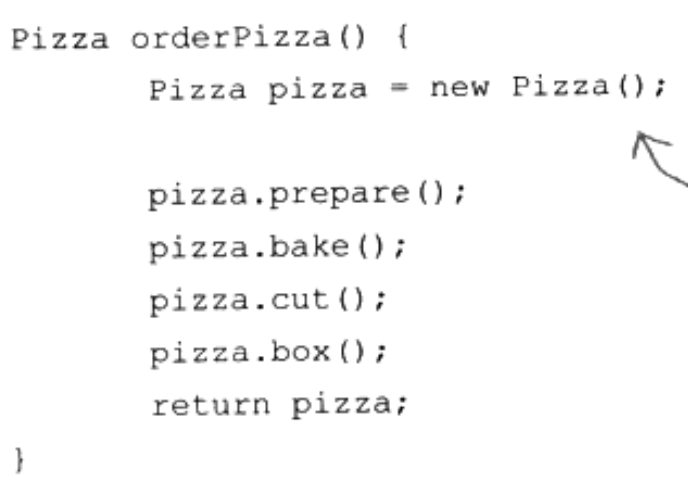
针对接口编程，可以隔离掉以后系统可能发生的一大堆改变。为什么呢？如果代码是针对接口写的，那么通过多态，它可以与任何新类实现该接口。但是，当代码使用大量的具体类时，等于是自找麻烦，因为一旦加入新的具体类，就必须改变代码。也就是说，你的代码并非“对修改关闭”（对扩展开放，对修改关闭）。想用新的具体类型来扩展代码。必须重新打开它。

当遇到这样的问题时，就应该回到OO设计原则去寻找线索。别忘了，我们的第一个原则来处理改变，并帮助我们“找出会变化的方面，把它们从不变的部分分离出来”。

如何将实例化具体类的代码从应用中抽离，或者封装起来，使它们不会干扰应用的其他部分？

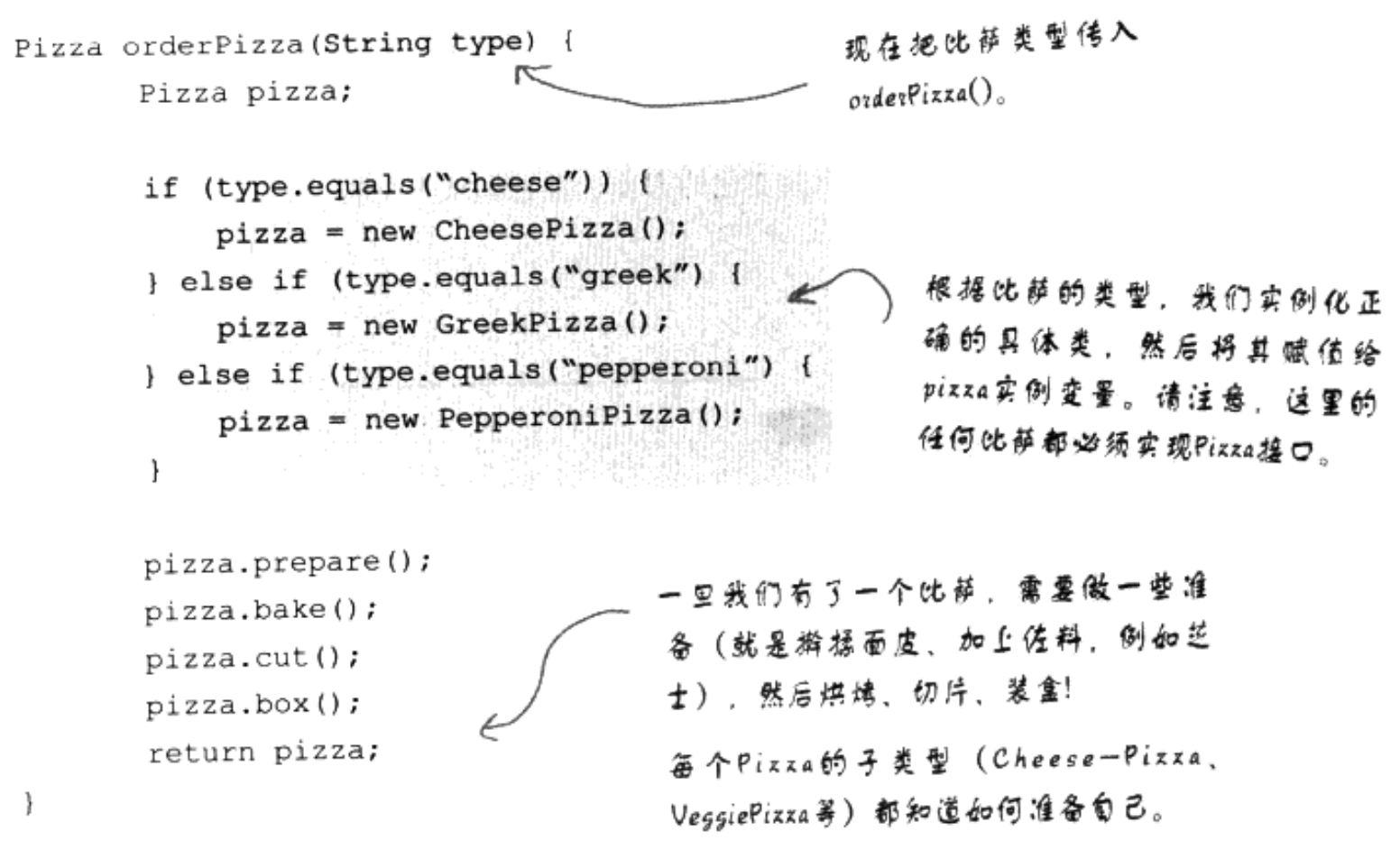
#### 识别变化的方面

假如你有一个比萨店，身为对象村内最先进的比萨店主人，你的代码可能这么写：

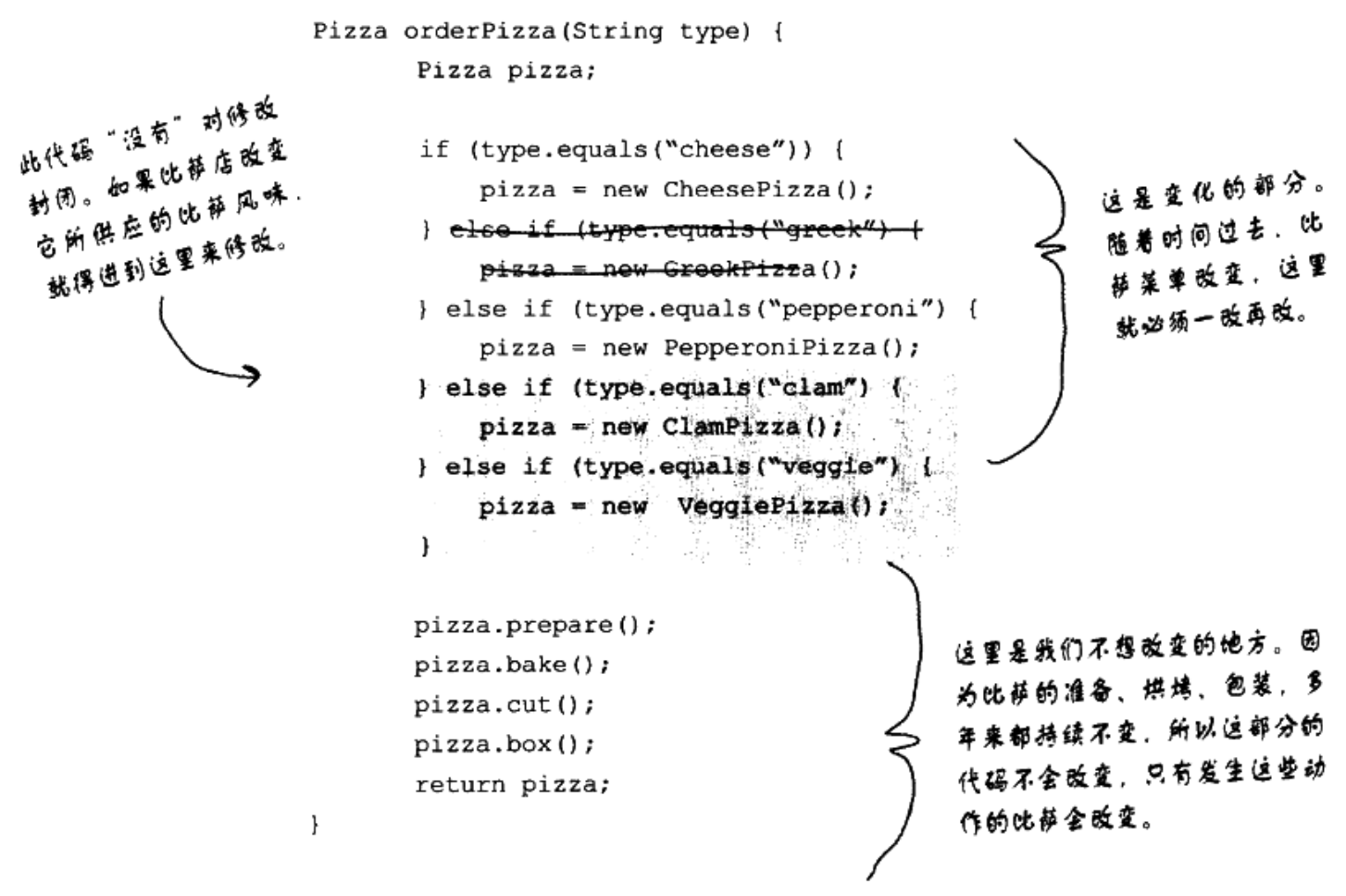
为了让系统有弹性，我们希望Pizza是一个抽象类或接口。但如果这样，这些类或接口就无法直接实例化。

#### 但是你需要更多比萨类型

所以必须增加一些代码，来“决定”适合的比萨类型，然后再“制造”这个比萨。



#### 但是压力来自于增加更多的比萨类型



如果实例化“某些”具体类，将使oderPizza()出问题，而且也无法让orderPizza()对修改关闭；但是，现在我们已经知道哪些会改变，哪些不会改变，该是使用封装的时候了。

#### 封装创建对象的代码

现在最好将创建对象移到orderPizza()之外，但怎么做呢？要把创建比萨的代码移到另一个对象中，由这个新对象专职创建比萨。

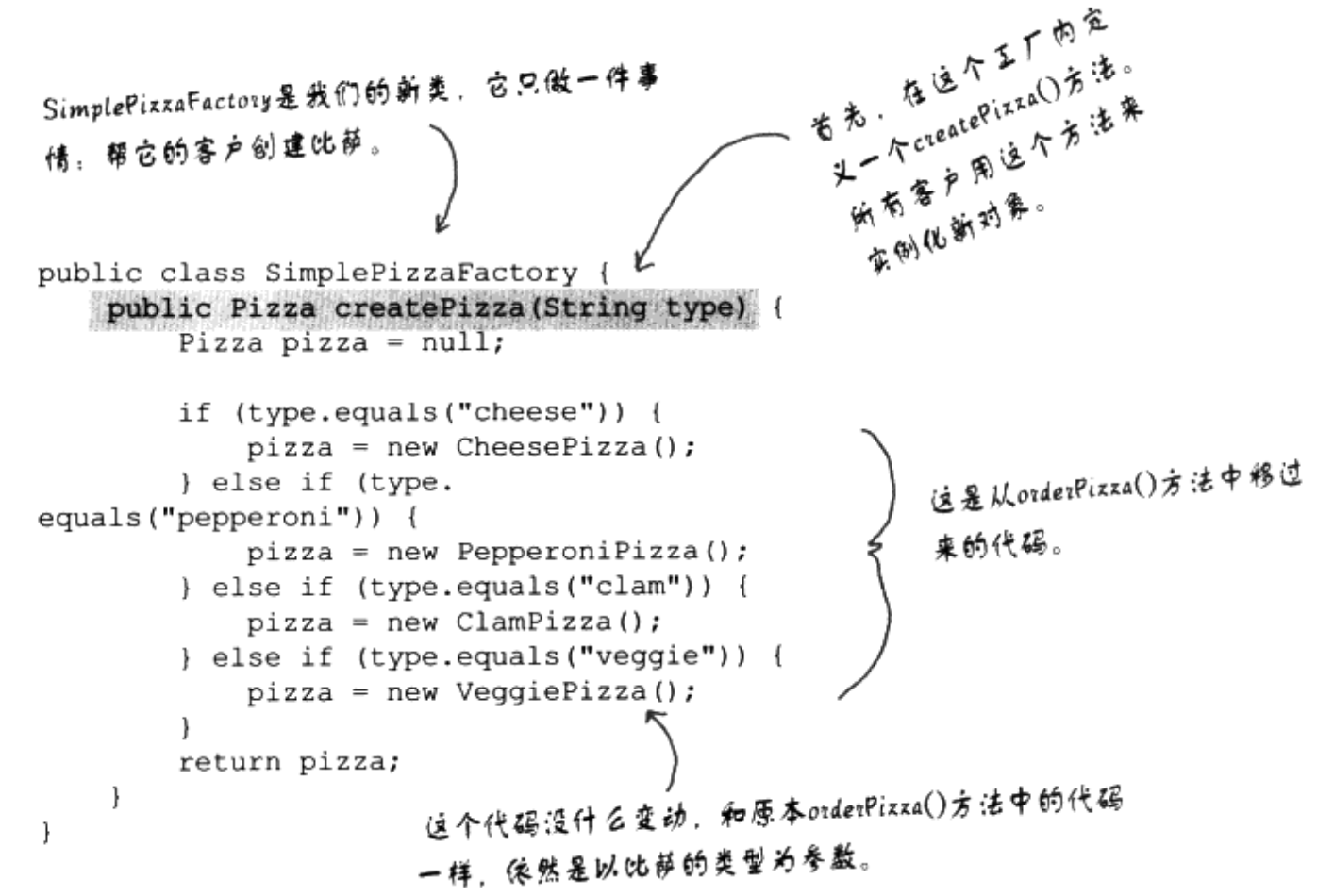
* 首先，把创建对象的代码从orderPizza()方法中抽离；
* 然后把这部分代码搬到另一个对象中，这个新对象只管如何创建比萨。如果任何对象想要创建比萨，找SimplePizzaFactory就对了。

#### 这个新对象称为“工厂”

工厂“factory”处理创建对象的细节。一旦有了SimplePizzaFactory，orderPizza()就变成此对象的客户当需要比萨时，就叫比萨工厂做一个。那些orderPizza()方法就不需要关系做个什么比萨。现在orderPizza()方法只关心从工厂得到一个比萨，而这个比萨实现了Pizza接口，所以它可以调用prepare()、bake()、cut()、box()来分别进行准备、烘烤、切片、装盒。

#### 建立一个简单比萨工厂

先从工厂本身开始。我们要定义一个类，为所有比萨封装创建对象的代码。代码像这样。



这么做的好处：似乎只是把问题搬到另一个对象，问题依然存在。SimplePizzaFactory可以有许多的客户。虽然目前只看到orderPizza()方法，是它的客户，然而，可能还有PizzaShopMenu（比萨菜单）类，会利用这个工厂来取得比萨的价钱和描述。可能还有一个HomeDelivery（宅急送）类，会以与PizzaShop类不同的方式来处理比萨。总而言之，SimplePizzaFactory可以有许多不同的客户。

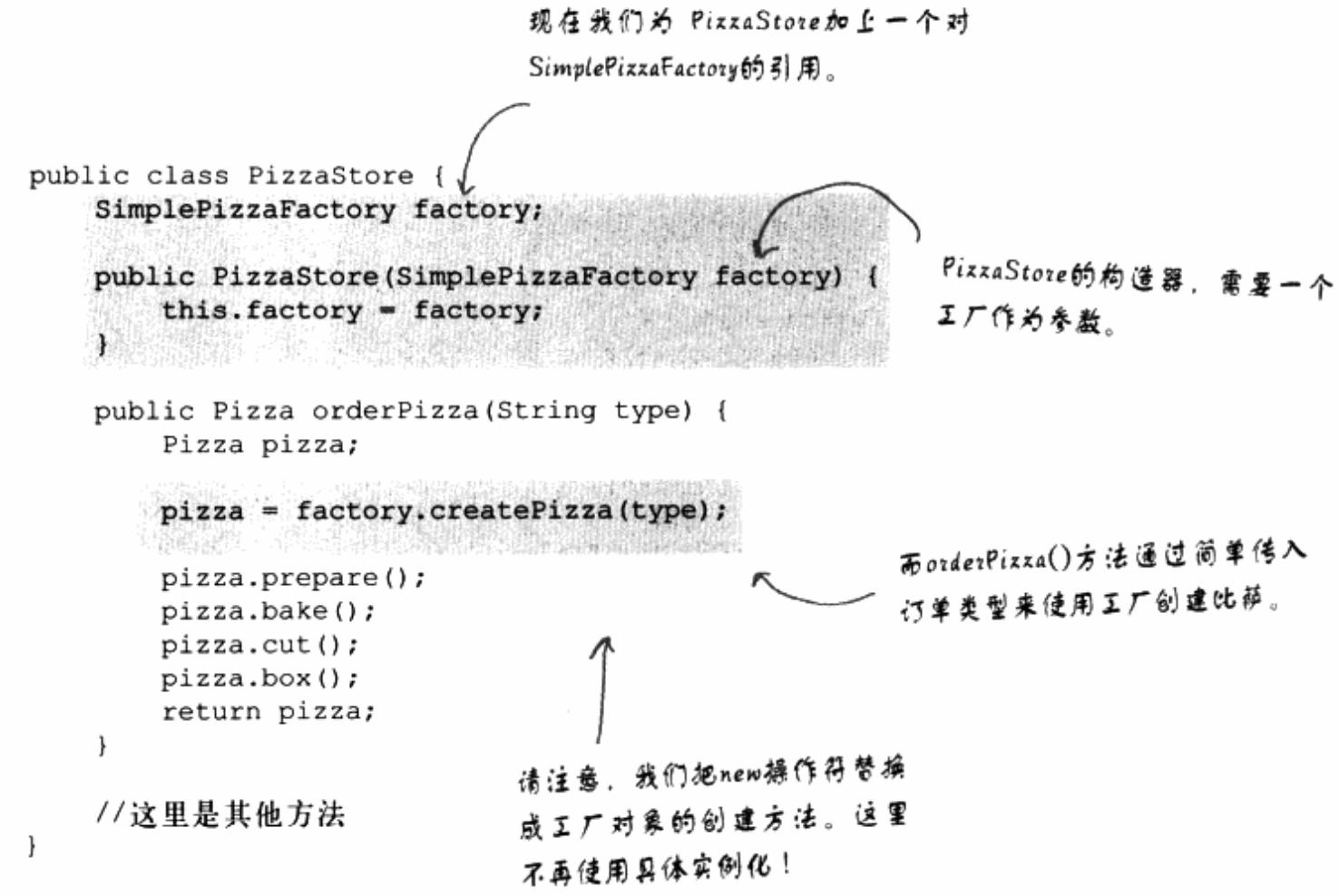
所以，把创建比萨的代码包装进一个类，当以后实现改变时，只需修改这个类即可。别忘了，也要把具体实例化的过程，从客户代码中删除。

如果把工厂定义成一个静态的方法。这有何差别？

为何使用静态方法，因为不需要使用创建对象的方法来实例化对象。但是这也有缺点，不能通过继承来改变创建方法的行为。

#### 重做PizzaStore类

是时候修改我们的客户代码了，我们所要做的是仰仗工厂来为我们创建比萨，要做这样的改变：

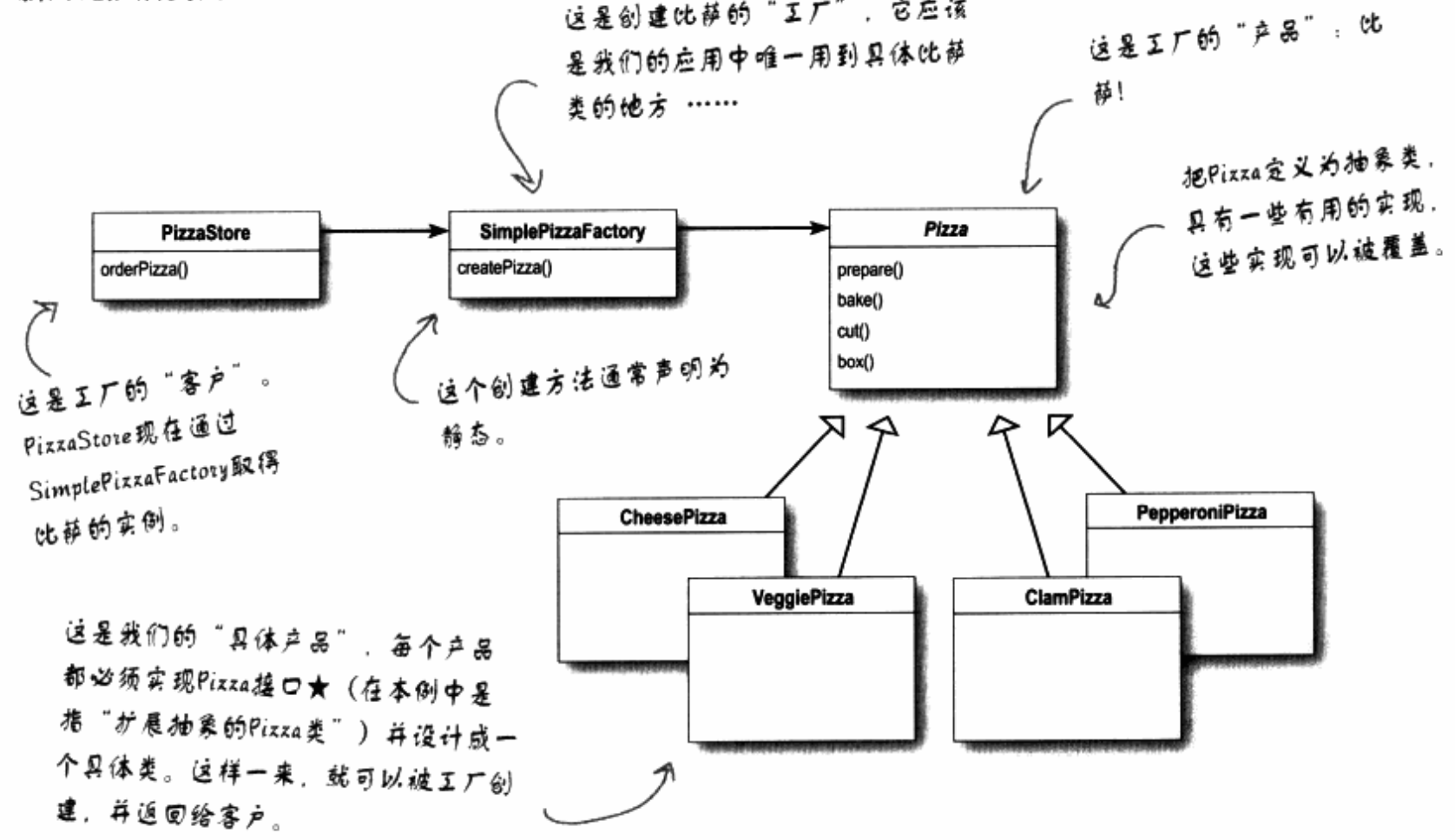


对象组合可以在运行时动态改变行为，因为我们可以更换不同的实现。在PizzaStore例子中要如何做到这点呢？有哪些工厂的实现能够被我们自由地更换？

### 2、定义简单工厂

简单工厂其实不是一个设计模式，反而比较像是一种编程习惯。有些开发人员的确是把这个编程习惯误认为是“工厂模式”（Factory Pattern）。当你下次和另一个开发人员之间无话可说的时候，这应当是打破沉默的一个不错的话题。

不要因为简单工厂不是一个“真正的”模式，就忽略了它的用法。让我们来看看新的比萨店类图。



在设计模式中，所谓的“实现一个接口”并不一定表示“写一个类，并利用implement关键字来实现某个Java接口”。“实现一个接口”泛指“实现某个超类型（可以是类或接口）的某个方法”。

### Spring如何创建Bean