抽象工厂：

常规的对象创建方法:

//创建一个Road对象

Road road = new Road();

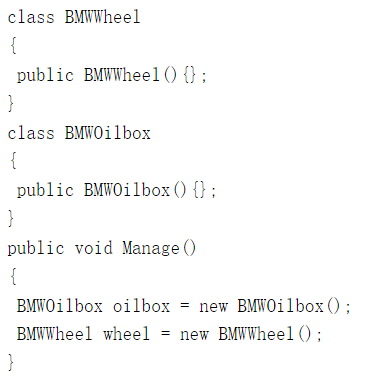
new 的问题：

实现依赖，不能应对“具体实例化类型”的变化。

解析：

抽象工厂是一种创建型模式，是为了解决实例化时所带来的问题。

我们先看是什么问题，有的时候我们会遇到这种情况，我们需要一系列的对象。举个例子，有一系列BMW汽车零部件的对象：轮子bmwwheel， 邮箱bmwoilbox，在一个管理函数中调用他们，代码如下：



如果现在需求变了，我们要用大众BORA的零件，不用BMW的，那么我们除了要加上相应的零件对象外还要将Manage函数中的对象更改为BORA的零件对象。

那这是发现new会带来了一些问题：实现依赖，不能应对具体实例化类型的变化。（不符合开闭原则。只扩展代码，而不是修改代码）

解决思路：

封装变化点——哪里变化，封装哪里

潜台词：如果没有变化。当然不需要额外的封装

工厂模式的缘起：

1. 变化点在“对象创建”，因此就封装“对象创建”
2. 面向接口编程——依赖接口，而非依赖实现

简单工厂问题：

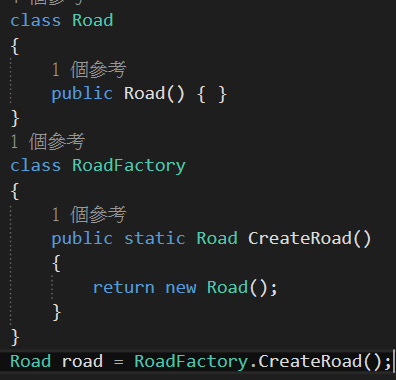
1. 不能应对“不同系列对象”的变化。如：我们要在上面的代码中加上其他的对象就不能很好的应对了。
2. 使用面向对象的技术来封装变化点

动机：在软件系统中，经常面临着“一系列相互依赖的对象”的创建工作；同时，由于需求的变化，往往存在更多系列对象的创建工作。对于这种问题，我们就必须绕过常规的对象创建方法，提供一种“封装机制”来避免客户程序和这种“多系列具体对象创建工作”的紧耦合。

为什么如今的程序要摆脱紧耦合，正是因为面对频繁变化的需求，紧耦合会使程序的编写变得很吃力。如果是一个不会改变的需求，松耦合还是紧耦合就不会那么重要了。

总的来说就是将这些不同的具体类进行抽象，实现一个抽象基类。我们再通过不同的需求来在继承这个基类的基础上进行变化。

最简单的解决方法：



如上可见简单工厂的问题：

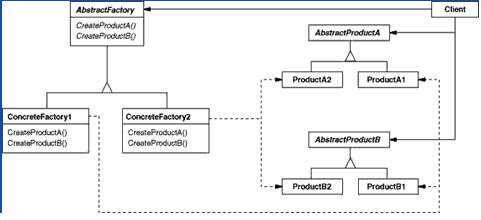
不能应对“不同系列对象”的变化。比如有不同风格的场景——对应不同风格的道路。

如何解决：

使用面向对象的技术来“封装”变化点。

提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

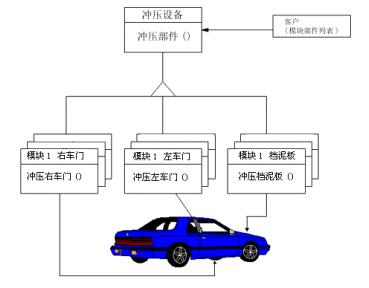
结构图：



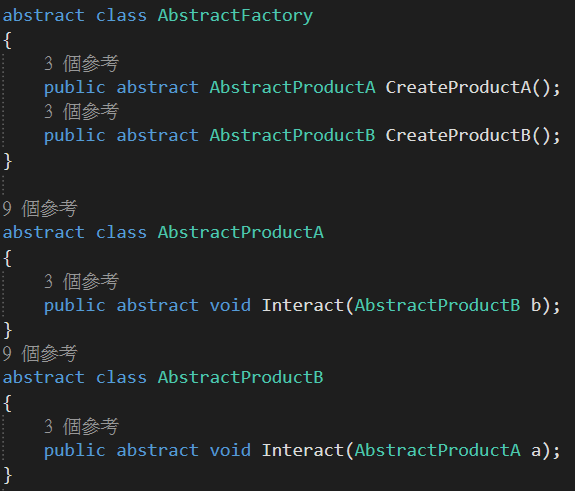
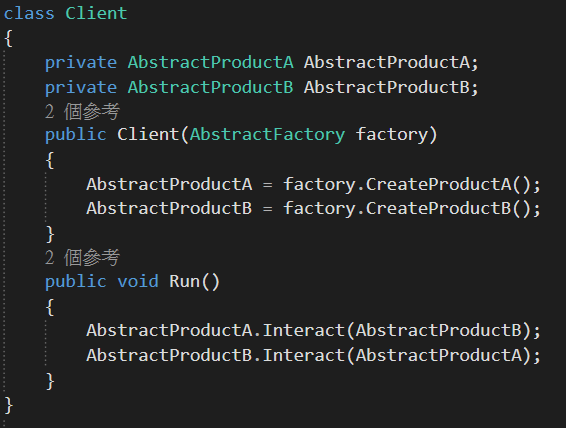
适用性：

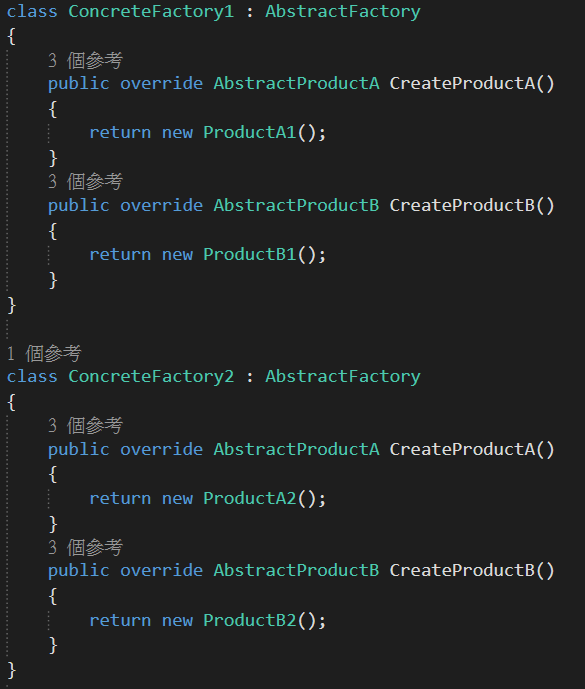
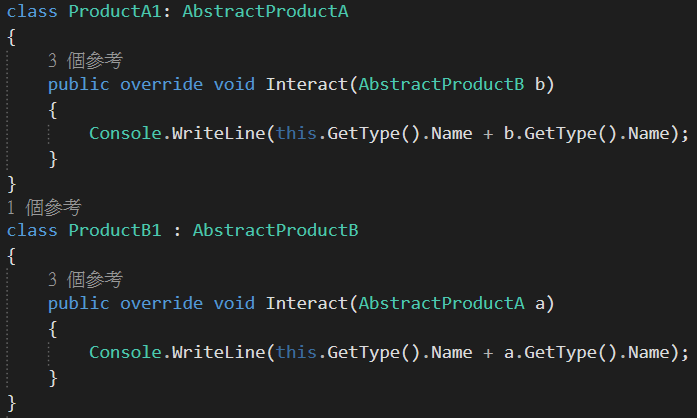
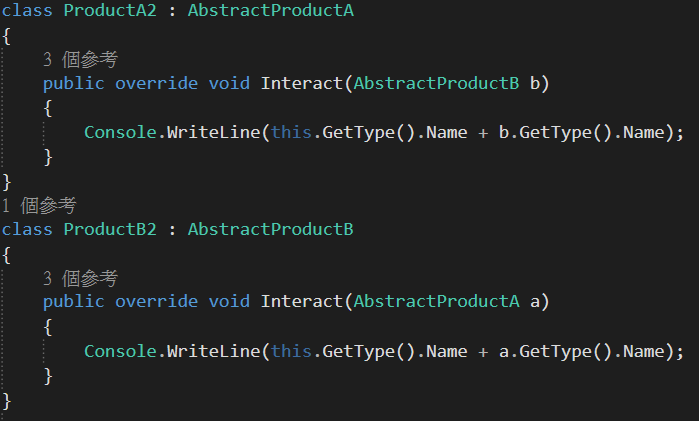
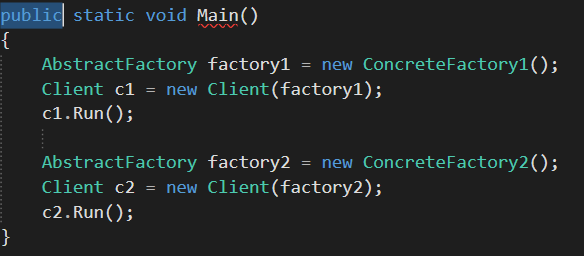
1. 一个系统要独立于它的产品的创建、组合和表示时
2. 一个系统要由多个产品系统中的一个来配置时
3. 当你要强调一系列相关的产品对象的设计以便进行联合使用时
4. 当你提供一个产品类库，而只想显示它们的接口不是实现时

生活实例：



结构图代码实现：

如果不存在“多系列对象创建”的需求变化则没有必要应用Abstract Factory模式，静态工厂方法足以。

**"系列对象"**指的是这些对象之间有相互依赖、或作用的关系。例如游戏开发场景中的"道路"与"房屋"依赖，“道路”与“地道”的依赖。（例如某一个房子对应某一个道路。正如抽象类中的AbstractProductA和AbstractProductB）  
**Abstract Factory模式主要在于应对**"新系列"的需求变动。其**缺点**在于难以应对”新对象“的需求变动。  
**Abstract Factory**模式经常和**Factory Method**模式共同组合来应对“**对象创建**”的需求变化。