抽象工厂模式

抽象工厂设计模式概念：

针对抽象工厂这个设计模式，我查找了不少资料，感觉只有涉及产品级别和产品族的才是理解了抽象工厂设计模式的精髓，工厂方法模式针对的是一个产品等级结构；而抽象工厂模式针对的是多个产品等级结构。有些观点认为抽象工厂模式是为了解决客户端代码与工厂类的耦合问题，我认为这种观点的解决方案只是简单工厂模式的一个应用，而这种观点认为的抽象工厂模式是：

工厂模式+简单工厂模式=抽象工厂模式，这是不正确。

针对的问题：

针对多个产品等级结构。相对工厂模式针对的是一个产品等级结构。

例子描述

我们通过一个简单的例子来解释抽象设计模式，同时这个例子贯穿全文：

——

组装电脑，我们在组装电脑的时候，通常需要选择一系列的配件，比如CPU、硬盘、内存、主板、电源、机箱等。为讨论使用简单点，只考虑选择CPU和主板的问题。

事实上，在选择CPU的时候，面临一系列的问题，比如品牌、型号、针脚数目、主频等问题，只有把这些问题都确定下来，才能确定具体的CPU。

同样，在选择主板的时候，也有一系列问题，比如品牌、芯片组、集成芯片、总线频率等问题，也只有这些都确定了，才能确定具体的主板。

选择不同的CPU和主板，是每个客户在组装电脑的时候，向装机公司提出的要求，也就是我们每个人自己拟定的装机方案。

在最终确定这个装机方案之前，还需要整体考虑各个配件之间的兼容性。比如：CPU和主板，如果使用Intel的CPU和AMD的主板是根本无法组装的。因为Intel的CPU针脚数与AMD主板提供的CPU插口不兼容，就是说如果使用Intel的CPU根本就插不到AMD的主板中，所以装机方案是整体性的，里面选择的各个配件之间是有关联的。

对于装机工程师而言，他只知道组装一台电脑，需要相应的配件，但是具体使用什么样的配件，还得由客户说了算。也就是说装机工程师只是负责组装，而客户负责选择装配所需要的具体的配件。因此，当装机工程师为不同的客户组装电脑时，只需要根据客户的装机方案，去获取相应的配件，然后组装即可。

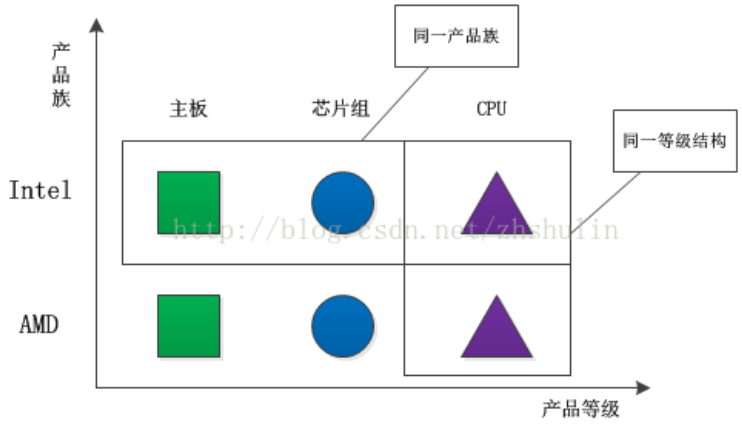
——

两个基本概念：

在学习抽象工厂具体实例之前需要了解2个基本概念：产品族和产品等级。

所谓产品族，是指位于不同产品等级结构中，功能相关联的产品组成的家族。比如AMD的主板、芯片组、CPU组成一个家族，Intel的主板、芯片组、CPU组成一个家族。

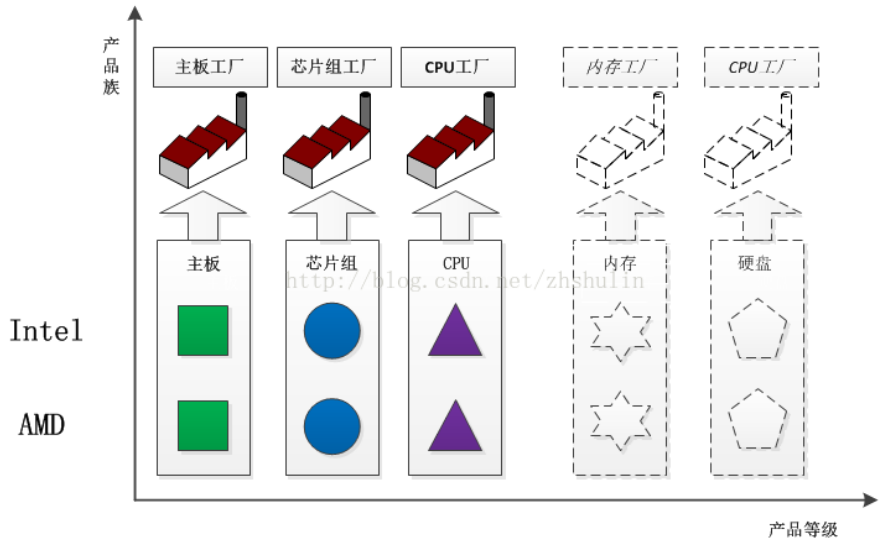
而这两个家族都来自于三个产品等级：主板、芯片组、CPU。一个等级结构是由相同的结构的产品组成，示意图如下：



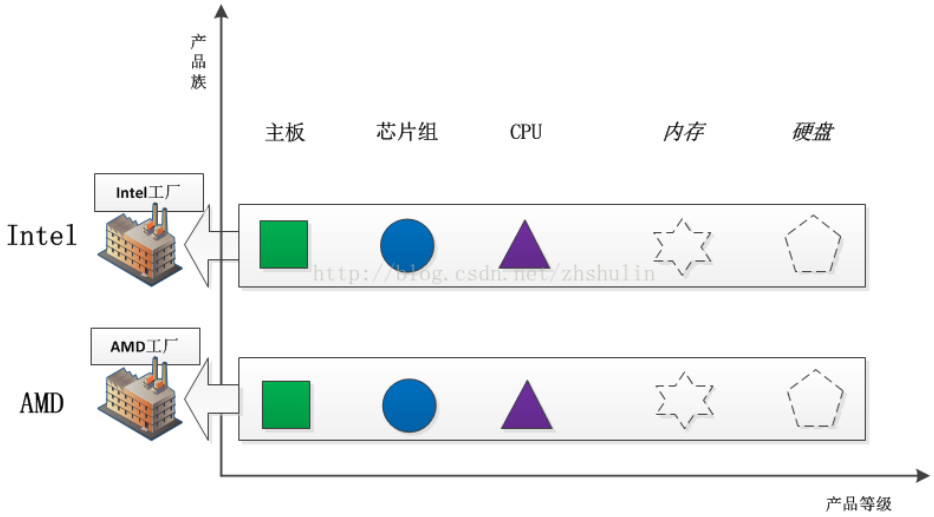
针对上述的例子进行程序设计，采用工厂模式和抽象工厂模式有什么区别？

工厂模式：

上面所给出的三个不同的等级结构具有平行的结构。因此，如果采用工厂方法模式，就势必要使用三个独立的工厂等级结构来对付这三个产品等级结构。由于这三个产品等级结构的相似性，会导致三个平行的工厂等级结构。随着产品等级结构的数目的增加，工厂方法模式所给出的工厂等级结构的数目也会随之增加。如下图：

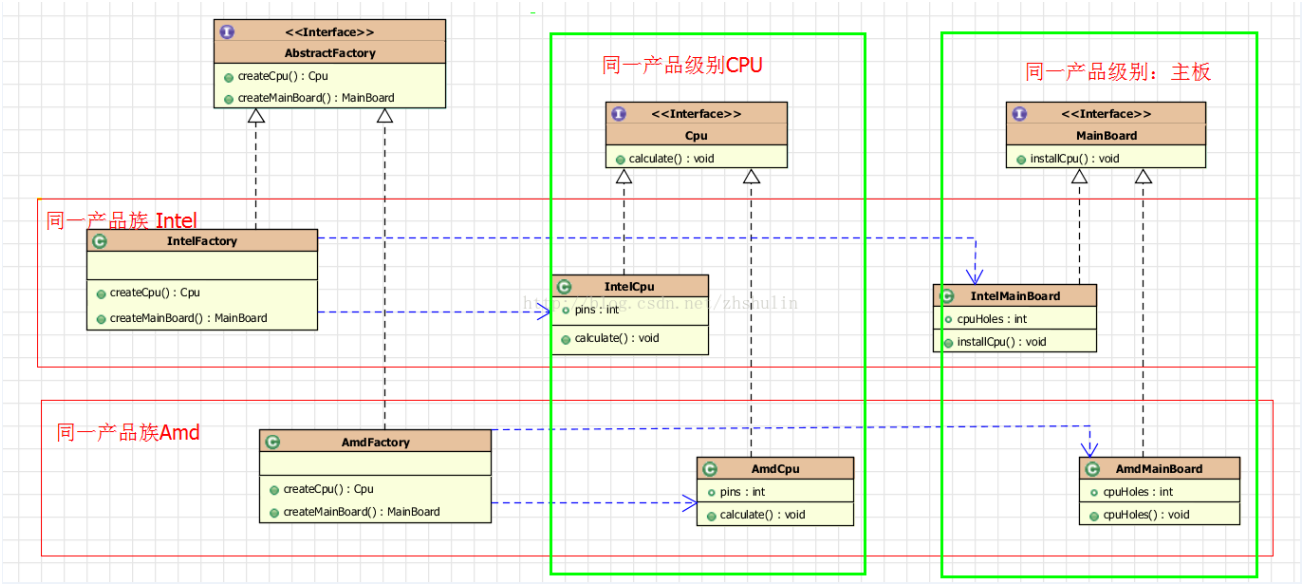


抽象工厂模式：



通过上面两幅图，可以看出，一个工厂等级结构可以创建出分属于不同产品等级结构的一个产品族中的所有对象。显然，这时候抽象工厂模式比简单工厂模式、工厂方法模式更有效率。对应于每一个产品族都有一个具体工厂。而每一个具体工厂负责创建属于同一个产品族，但是分属于不同等级结构的产品。

用抽象工厂模式设计类图如下：



使用情形：

1. 一个系统不应当依赖于产品类实例如何被创建、组合和表达的细节，这对于所有形态的工厂模式都是重要的。

2. 这个系统的产品有多于一个的产品族，而系统只消费其中某一族的产品。

3. 同属于同一个产品族的产品是在一起使用的，这一约束必须在系统的设计中体现出来。（比如：Intel主板必须使用Intel CPU、Intel　芯片组）

4.系统提供一个产品类的库，所有的产品以同样的接口出现，从而使客户端不依赖于实现。

优点：

1、 分离接口和实现

客户端使用抽象工厂来创建需要的对象，而客户端根本就不知道具体的实现是谁，客户端只是面向产品的接口编程而已。也就是说，客户端从具体的产品实现中解耦。

2、 使切换和添加产品族变得容易

因为一个具体的工厂实现代表的是一个产品族，比如上面例子的从Intel系列到AMD系列只需要切换一下具体工厂。我也可以再添加一个新的装机方案，一个新的产品族，都是很方便的。

不足：

不容易扩展新的产品等级，比如我要加一个硬盘、内存什么的。那么就需要修改抽象工厂，这样就会导致修改所有的工厂实现类。