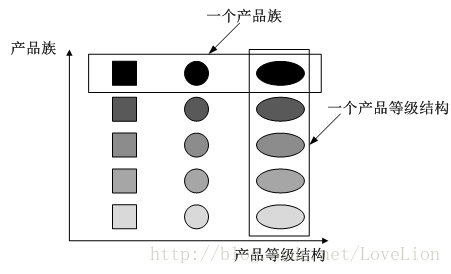
工厂方法模式通过引入工厂等级结构，解决了简单工厂模式中工厂类职责太重的问题，但由于工厂方法模式中的每个工厂只生产一类产品，可能会导致系统中存在大量的工厂类，势必会增加系统的开销。此时，我们可以考虑**将一些相关的产品组成一个“产品族”，由同一个工厂来统一生产**，这就是我们本文将要学习的抽象工厂模式的基本思想。

在工厂方法模式中具体工厂负责生产具体的产品，每一个具体工厂对应一种具体产品，工厂方法具有唯一性，一般情况下，一个具体工厂中只有一个或者一组重载的工厂方法。但是有时候我们**希望一个工厂可以提供多个产品对象，而不是单一的产品对象**，如一个电器工厂，它可以生产电视机、电冰箱、空调等多种电器，而不是只生产某一种电器。为了更好地理解抽象工厂模式，我们先引入两个概念：

(1) 产品等级结构：**产品等级结构即产品的继承结构**，如一个抽象类是电视机，其子类有海尔电视机、海信电视机、TCL电视机，则抽象电视机与具体品牌的电视机之间构成了一个产品等级结构，抽象电视机是父类，而具体品牌的电视机是其子类。

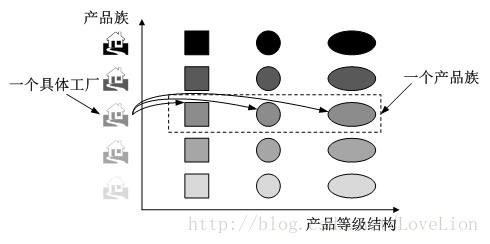
(2) 产品族：在抽象工厂模式中，**产品族是指由同一个工厂生产的，位于不同产品等级结构中的一组产品**，如海尔电器工厂生产的海尔电视机、海尔电冰箱，海尔电视机位于电视机产品等级结构中，海尔电冰箱位于电冰箱产品等级结构中，海尔电视机、海尔电冰箱构成了一个产品族。

产品等级结构与产品族示意图如图所示：



在图中，不同颜色的多个正方形、圆形和椭圆形分别构成了三个不同的产品等级结构，而相同颜色的正方形、圆形和椭圆形构成了一个产品族，每一个形状对象都位于某个产品族，并属于某个产品等级结构。图3中一共有五个产品族，分属于三个不同的产品等级结构。我们只要指明一个产品所处的产品族以及它所属的等级结构，就可以唯一确定这个产品。

当系统所提供的工厂生产的具体产品并不是一个简单的对象，而是多个位于不同产品等级结构、属于不同类型的具体产品时就可以使用抽象工厂模式。**抽象工厂模式是所有形式的工厂模式中最为抽象和最具一般性的一种形式**。抽象工厂模式与工厂方法模式最大的区别在于，工厂方法模式针对的是一个产品等级结构，而抽象工厂模式需要面对多个产品等级结构，一个工厂等级结构可以负责多个不同产品等级结构中的产品对象的创建。**当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时，抽象工厂模式比工厂方法模式更为简单、更有效率**。抽象工厂模式示意图如图所示：

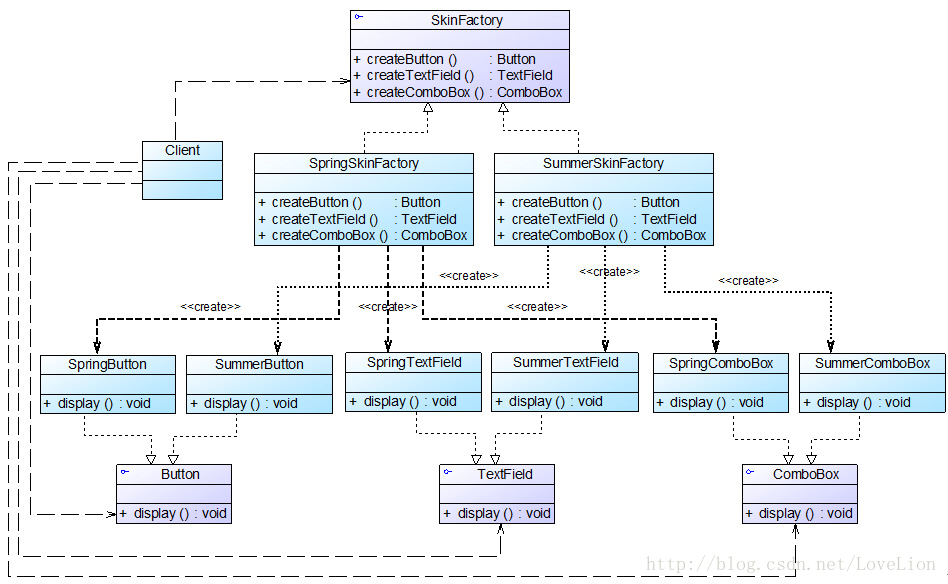


在图中，每一个具体工厂可以生产属于一个产品族的所有产品，例如生产颜色相同的正方形、圆形和椭圆形，所生产的产品又位于不同的产品等级结构中。如果使用工厂方法模式，图所示结构需要提供15个具体工厂，而使用抽象工厂模式只需要提供5个具体工厂，极大减少了系统中类的个数。

抽象工厂模式为创建一组对象提供了一种解决方案。与工厂方法模式相比，抽象工厂模式中的具体工厂不只是创建一种产品，它负责创建一族产品。抽象工厂模式定义如下：

抽象工厂模式(Abstract Factory Pattern)：**提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。抽象工厂模式又称为Kit模式，它是一种对象创建型模式。**

在抽象工厂模式中，每一个具体工厂都提供了多个工厂方法用于产生多种不同类型的产品，这些产品构成了一个产品族，抽象工厂模式结构如图所示：



在抽象工厂模式结构图中包含如下几个角色：

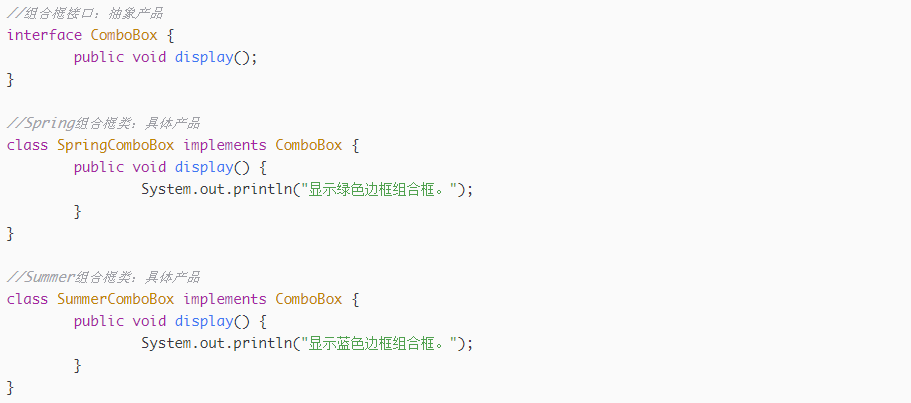
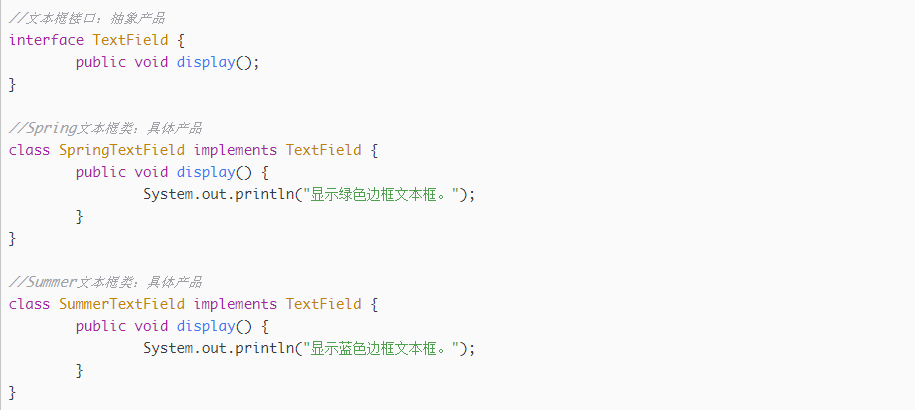
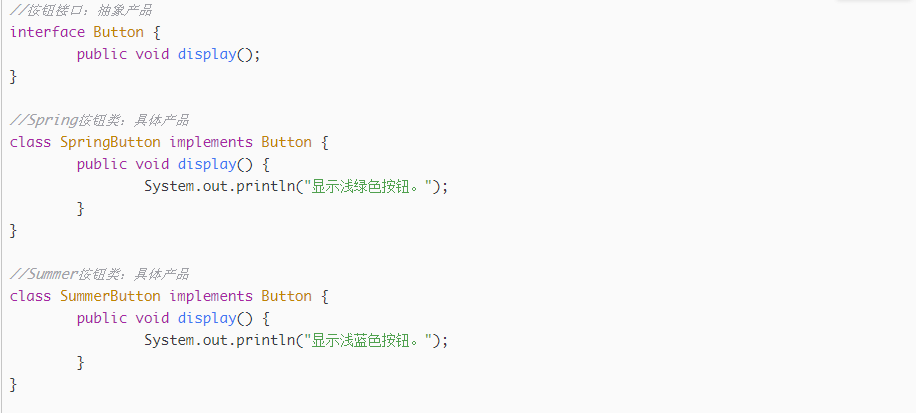
       ● SkinFactory（抽象工厂）：它声明了一组用于创建一族产品的方法，每一个方法对应一种产品。

       ● SpringSkintFactory|SummerSkinFactory（具体工厂）：它实现了在抽象工厂中声明的创建产品的方法，生成一组具体产品，这些产品构成了一个产品族，每一个产品都位于某个产品等级结构中。

       ● Button|TextField|ComboBox（抽象产品）：它为每种产品声明接口，在抽象产品中声明了产品所具有的业务方法。

       ● SpringButton|SummerButton|SpringTextField|SummerField|SpringComnboBox|SummerComboBox（具体产品）：它定义具体工厂生产的具体产品对象，实现抽象产品接口中声明的业务方法。

完整代码如下：







如果需要增加新的皮肤，只需增加一组新的具体组件并对应提供一个新的具体工厂（产品族），修改配置文件即可使用新的皮肤，原有代码无须修改，符合“开闭原则”。但是，如果需要增加一个单选按钮的组件（新的产品等级结构），发现抽象工厂方法无法做到。这也是抽象工厂方法最大的缺点。**在抽象工厂模式中，增加新的产品族很方便，但是增加新的产品等级结构很麻烦**，抽象工厂模式的这种性质称为**“开闭原则”的倾斜性**。“开闭原则”要求系统对扩展开放，对修改封闭，通过扩展达到增强其功能的目的，对于涉及到多个产品族与多个产品等级结构的系统，其功能增强包括两方面：

(1) **增加产品族**：对于增加新的产品族，抽象工厂模式很好地支持了“开闭原则”，只需要增加具体产品并对应增加一个新的具体工厂，对已有代码无须做任何修改。

(2) **增加新的产品等级结构**：对于增加新的产品等级结构，需要修改所有的工厂角色，包括抽象工厂类，在所有的工厂类中都需要增加生产新产品的方法，违背了“开闭原则”。

正因为抽象工厂模式存在“开闭原则”的倾斜性，它以一种倾斜的方式来满足“开闭原则”，为增加新产品族提供方便，但不能为增加新产品结构提供这样的方便，因此要求设计人员在设计之初就能够全面考虑，不会在设计完成之后向系统中增加新的产品等级结构，也不会删除已有的产品等级结构，否则将会导致系统出现较大的修改，为后续维护工作带来诸多麻烦。

抽象工厂模式总结

       抽象工厂模式是工厂方法模式的进一步延伸，由于它提供了功能更为强大的工厂类并且具备较好的可扩展性，在软件开发中得以广泛应用，尤其是在一些框架和API类库的设计中，例如在Java语言的AWT（抽象窗口工具包）中就使用了抽象工厂模式，它使用抽象工厂模式来实现在不同的操作系统中应用程序呈现与所在操作系统一致的外观界面。抽象工厂模式也是在软件开发中最常用的设计模式之一。

        1. **主要优点**

       (1) 抽象工厂模式隔离了具体类的生成，使得客户并不需要知道什么被创建。由于这种隔离，更换一个具体工厂就变得相对容易，所有的具体工厂都实现了抽象工厂中定义的那些公共接口，因此只需改变具体工厂的实例，就可以在某种程度上改变整个软件系统的行为。

       (2) 当一个产品族中的多个对象被设计成一起工作时，它能够保证客户端始终只使用同一个产品族中的对象。

       (3) 增加新的产品族很方便，无须修改已有系统，符合“开闭原则”。

        2. **主要缺点**

       增加新的产品等级结构麻烦，需要对原有系统进行较大的修改，甚至需要修改抽象层代码，这显然会带来较大的不便，违背了“开闭原则”。

        3. **适用场景**

       (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，这对于所有类型的工厂模式都是很重要的，用户无须关心对象的创建过程，将对象的创建和使用解耦。

       (2) 系统中有多于一个的产品族，而每次只使用其中某一产品族。可以通过配置文件等方式来使得用户可以动态改变产品族，也可以很方便地增加新的产品族。

       (3) 属于同一个产品族的产品将在一起使用，这一约束必须在系统的设计中体现出来。同一个产品族中的产品可以是没有任何关系的对象，但是它们都具有一些共同的约束，如同一操作系统下的按钮和文本框，按钮与文本框之间没有直接关系，但它们都是属于某一操作系统的，此时具有一个共同的约束条件：操作系统的类型。

       (4) 产品等级结构稳定，设计完成之后，不会向系统中增加新的产品等级结构或者删除已有的产品等级结构。

[抽象工厂模式](https://blog.csdn.net/LoveLion/article/details/17517213)

[工厂三兄弟之抽象工厂模式（一）：界面皮肤库的初始设计](https://blog.csdn.net/lovelion/article/details/9319181)

[工厂三兄弟之抽象工厂模式（二）：产品等级结构与产品族](https://blog.csdn.net/lovelion/article/details/9319323)

[工厂三兄弟之抽象工厂模式（三）：抽象工厂模式概述](https://blog.csdn.net/lovelion/article/details/9319423)

[工厂三兄弟之抽象工厂模式（四）：界面皮肤库的抽象工厂模式解决方案](https://blog.csdn.net/lovelion/article/details/9319481)

[工厂三兄弟之抽象工厂模式（五）：“开闭原则”的倾斜性，抽象工厂模式总结](https://blog.csdn.net/lovelion/article/details/9319571)