# 抽象工厂模式简介

## 概述

1. 抽象工厂模式比工厂方法模式的抽象程度更高。在工厂方法模式中，每一个具体工厂只需要生产一种具体产品，但是在抽象工厂模式中，每一个具体工厂可以生产一组相关的具体产品，这样的一组产品称为产品族，产品族中的每一个产品都分属于某一个产品继承等级结构（也就是每一个产品是某个抽象产品的子类）。
2. 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产品等级结构中的产品对象的创建。
3. 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。

## 相关术语

为了更好地理解抽象工厂模式，引入两个概念：产品等级结构和产品族

1. 产品等级结构：产品等级结构即产品的继承结构，例如一个抽象类或接口是电视机，其子类包括海尔电视机、海信电视机等，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子类。
2. 产品族：在抽象工厂模式中，产品族是指由一同一个工厂生产的位于不同产品等级结构中的一个组产品，例如海尔电器工厂生产的海尔电视机、海尔冰箱，海尔电视机位于电视机产品等级结构中，海尔冰箱位于冰箱等级结构中，海尔电视机，海尔冰箱则构成了一个产品族。

# 结构和实现

## 结构



抽象工厂模式包含以下4个角色。

1. AbstractFactory（抽象工厂）：通常是一个接口或抽象类，它声明了一组用于创建一族产品的方法，每一个方法对应着一种产品。
2. ConcreteFactory（具体工厂）：是抽象工厂的子类，实现抽象工厂后，覆写了在抽象工厂中声明的创建产品的方法，负责创建一个个具体产品，这些产品构成了一个产品族，每一个产品都位于具体产品等级结构中。
3. AbstractProduct（抽象产品）：通常是一个接口或抽象类，是具体产品的超类，也是产品等级结构中的顶部。
4. ConcreteProduct（具体产品）：是抽象产品的子类，也是产品等级结构中的产品。

在一个完整的抽象工厂模式系统中，存在一个抽象工厂，多个具体工厂，多个抽象产品，且每个抽象产品有多个具体产品。

## 实现

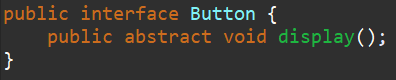
实现说明：某软件公司要开发一套界面皮肤库，可以对基于Java的桌面软件进行界面美化。用户在使用时可以通过菜单来选择皮肤，不同的皮肤将提供视觉效果不同的按钮、文本框、组合框等界面元素，例如春天（Spring）风格的皮肤将提供浅绿色的按钮、绿色边框的文本框和绿色边框的组合框，而夏天（Summer）风格的皮肤则提供浅蓝色的按钮、蓝色边框的文本框和蓝色边框的组合框。



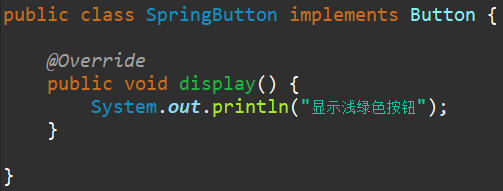


使用抽象工厂模式实现：

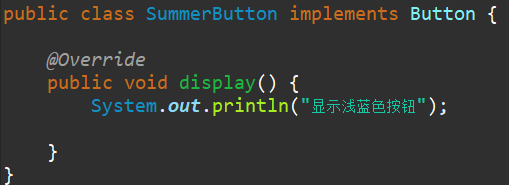
1. Button：按钮接口，充当抽象产品



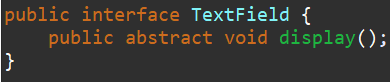
1. SpringButton：Spring风格按钮类，充当具体产品



1. SummerButton：Summer风格按钮类，充当具体产品



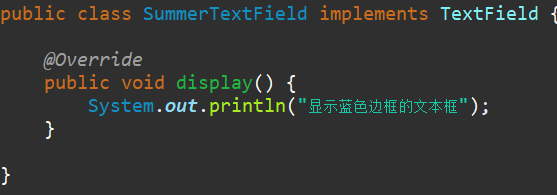
1. TextField：文本框接口，充当抽象产品



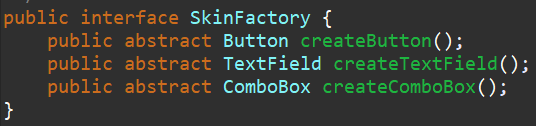
1. SpringTextField：Spring风格文本框，充当具体产品



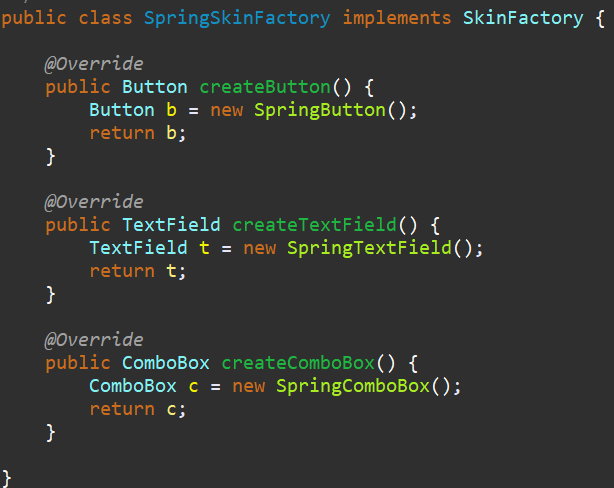
1. SummerTextField：Summer风格文本框，充当具体产品



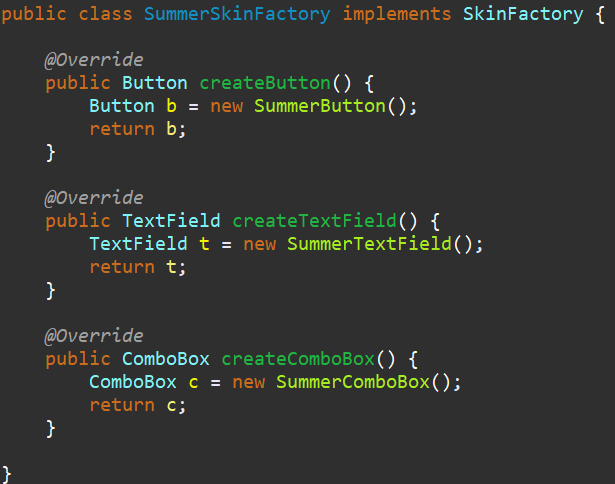
1. SkinFactory：界面皮肤工厂，充当抽象工厂



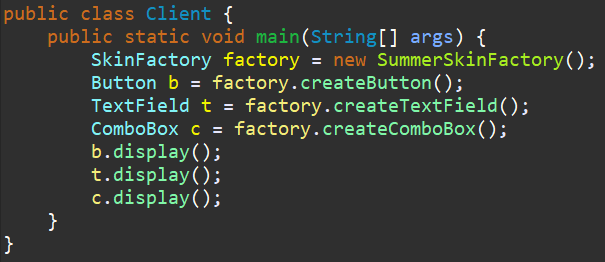
1. SpringSkinFactory：Spring风格皮肤工厂，充当具体工厂



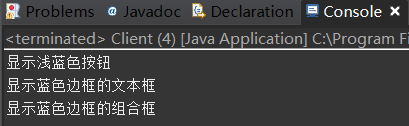
1. SummerSkinFactory：Summer风格皮肤工程，充当具体工厂



1. Client：测试类，用于测试抽象工厂模式



在这里SkinFactory皮肤工厂是通过SummerSkinFactory实例化的，所有皮肤工厂创建的按钮、文本框、组合框都是Summer风格的。运行结果如下：



# 优缺点以及适用环境

## 优点

1. 抽象工厂模式隔离了具体类的生成，使得客户端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定义的那些公共接口方法，因此只需要改变具体工厂的实例（修改抽象工厂实例的实例化方式，即通过哪个具体工厂来实例化抽象工厂）就可以在某种程度上改变整个系统的行为。
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

## 缺点

增加新的产品等级结构（由一个抽象产品以及其多个子类组成）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。

## 适用环境

1. 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。
2. 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，用户无须关系对象的创建过程，将对象的创建和使用解耦。
3. 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用户能够动态改变产品族，也可以很方便地增加新的产品族。
4. 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提现出来。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，即操作系统的类型。
5. 产品等级结构稳定，在设计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，至于为何这样，请看抽象工厂方法模式的缺点。