# 概述

# 分类

## 创建型

### 简单工厂模式

简单工厂模式，用户直接向Factory传递“命令”（也是需要创建对象的类名）来new对象，如果新增加类，需要重新修改工厂的方法（相当于重新录入命令）；

#### UML类图

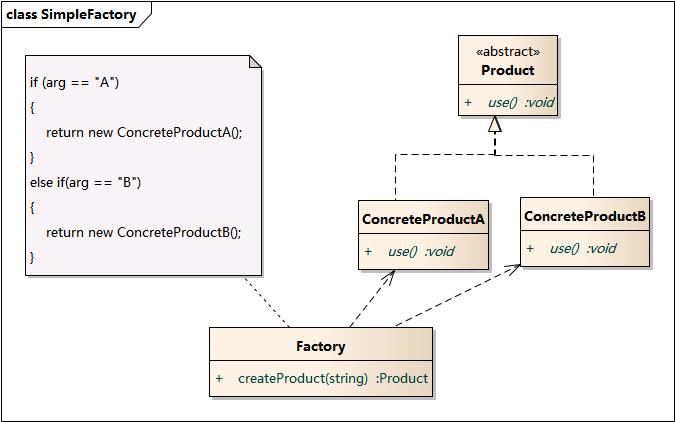


Figure 1 简单工厂模式UML类图

#### 模式角色

* Factory

工厂，负责完成各种不同产品的创建；

* Product

抽象产品类，所有的具体产品都需要继承该类；

* ConcreteProduct

具体的产品类

#### 代码分析

参考DesignPattern项目下的SimpleFactoryDesignPattern

#### 优缺点

##### 优点

* 代码简单
* 易于理解

##### 缺点

* 增加一个ConcreteProduct类时，需要修改factory中的生产方法

### 工厂方法模式

和简单工厂模式比较起来，工厂方法模式在原有工厂的基础上为每一个产品分配一个车间，每个车间只负责生产对应的产品。如果要新增产品，需要同时增加该产品对应的类和车间。

#### UML类图

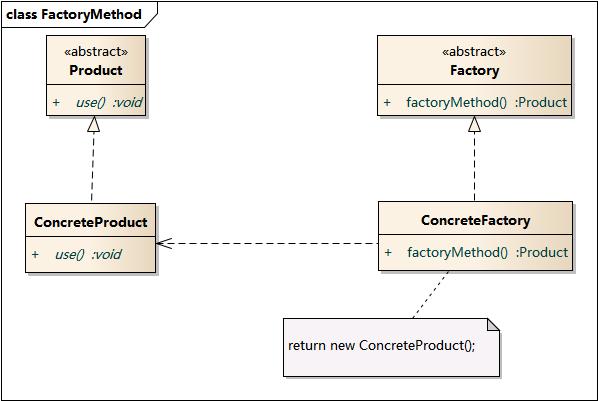


Figure 2 工厂方法模式UML图

#### 模式角色

* Factory

抽象的工厂，用于作为所有“车间”的父类或父接口

* ConcreteFactory

具体工厂，相当于一个“车间”，负责完成ConcreteProduct的构造

* Product

抽象产品，用于作为所有产品的父类

* ConcreteProduct

具体产品

#### 代码分析

参考DesignPattern项目下的FactoryMethodDesignPattern

#### 优缺点

##### 优点

* 在工厂方法模式中，工厂方法用来创建客户所需要的产品，同时还向客户隐藏了哪种具体产品类将被实例化这一细节，用户只需要关心所需产品对应的工厂，无须关心创建细节，甚至无须知道具体产品类的类名。
* 基于工厂角色和产品角色的多态性设计是工厂方法模式的关键。它能够使工厂可以自主确定创建何种产品对象，而如何创建这个对象的细节则完全封装在具体工厂内部。工厂方法模式之所以又被称为多态工厂模式，是因为所有的具体工厂类都具有同一抽象父类。
* 使用工厂方法模式的另一个优点是在系统中加入新产品时，无须修改抽象工厂和抽象产品提供的接口，无须修改客户端，也无须修改其他的具体工厂和具体产品，而只要添加一个具体工厂和具体产品就可以了。这样，系统的可扩展性也就变得非常好，完全符合“开闭原则”。

##### 缺点

* 在添加新产品时，需要编写新的具体产品类，而且还要提供与之对应的具体工厂类，系统中类的个数将成对增加，在一定程度上增加了系统的复杂度，有更多的类需要编译和运行，会给系统带来一些额外的开销。
* 由于考虑到系统的可扩展性，需要引入抽象层，在客户端代码中均使用抽象层进行定义，增加了系统的抽象性和理解难度，且在实现时可能需要用到DOM、反射等技术，增加了系统的实现难度。

### 抽象工厂方法模式

理解抽象工厂方法模式前，需要了解两个 概念：

* 产品等级结构：按照某一指标划分的产品树形结构图中，位于同一层级的产品被称为等级的（树形结构可以按照不同的指标进行划分，如Figure 3中按照车厢数划分，也可以按照汽车排量进行划分）
* 产品族：等级结构不相同，但是具有关联的产品

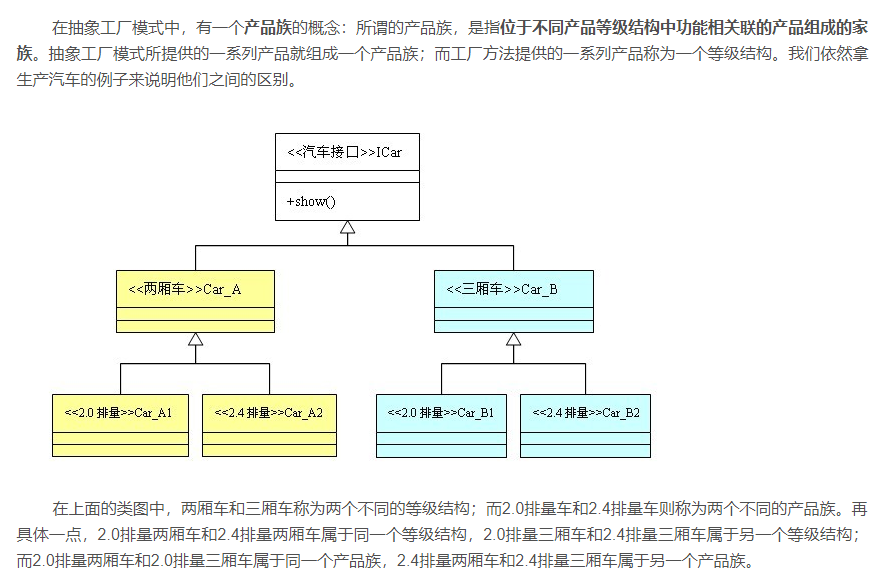


Figure 3 产品等级结构和产品族

简单的理解就是工厂中仍然有多个车间，而每个车间将不止生产一种产品，而是生产一族有关联的产品。

#### UML类图

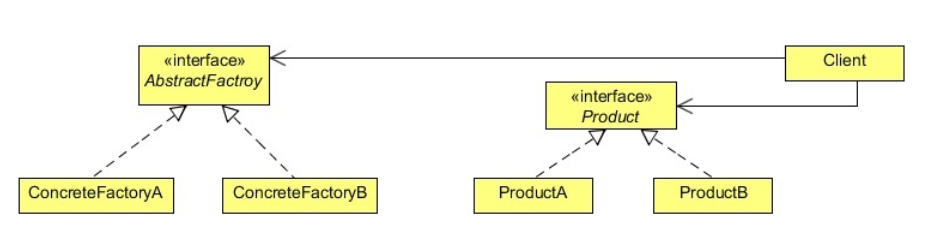


Figure 4 抽象工厂方法模式UML类图

#### 模式角色

* AbstractFactory：抽象工厂，作为其他所有具体工厂的父类
* AbstractProduct：抽象产品，作为其他所有具体产品的父类
* ConcreteFactory：具体工厂，负责创建具体产品对象，一个具体工厂类中可以创建多种具体产品
* ConcreteProduct：具体产品，描述一个具体的产品

#### 代码分析

#### 优缺点

## 结构型

## 行为型

# 参考资料

1. 图说设计模式<https://design-patterns.readthedocs.io/zh_CN/latest/>
2. 《Java Design Pattern》
3. 抽象工厂方法模式：<https://blog.csdn.net/zhengzhb/article/details/7359385>