设计模式

开闭原则： 对于扩展是开放的，对于修改是封闭的

设计模式六大原则

1. **开闭原则**

   开闭原则的意思是：对**扩展开放**，对**修改关闭**，在程序需要进行拓展的时候，不能去修改原有的代码

2. **里氏代换原则**

   里氏代换原则中说，任何**基类可以出现的地方，子类一定可以出现**，里氏代换原则是对开闭原则的补充，实现开闭原则的关键步骤就是抽象化，而基类与子类的继承关系就是抽象化的具体实现，所以里氏代换原则是对实现抽象化的具体步骤的规范

   简单的理解为一个软件实体如果使用的是一个父类，那么一定适用于其子类，而且它察觉不出父类对象和子类对象的区别。也就是说，软件里面，把父类都替换成它的子类，程序的行为没有变化。

3. **依赖倒转原则**

   这个原则是开闭原则的基础，具体内容：针对接口编程，**依赖于抽象**而**不依赖于具体**

4. **接口隔离原则**

   这个原则的意思是：**使用多个隔离的接口**，比使用单个接口要好，它还有另外一个意思是：降低类之间的耦合度，此可见，其实设计模式就是从大型软件架构出发、便于升级和维护的软件设计思想，它强调降低依赖，降低耦合

5. **最少知道原则**

   最少知道原则是指：一个实体应当**尽量少**地与其他实体之间发生**相互作用**，使得系统**功能模块相对独立**

6. **合成复用原则**

   合成复用原则是指：尽量使用合成/聚合的方式，而不是使用继承

> 聚合（Cohesion）是一个模块内部各成分之间相关联程度的度量。 耦合（Couping）是模块之间相关联程度的度量。

单例

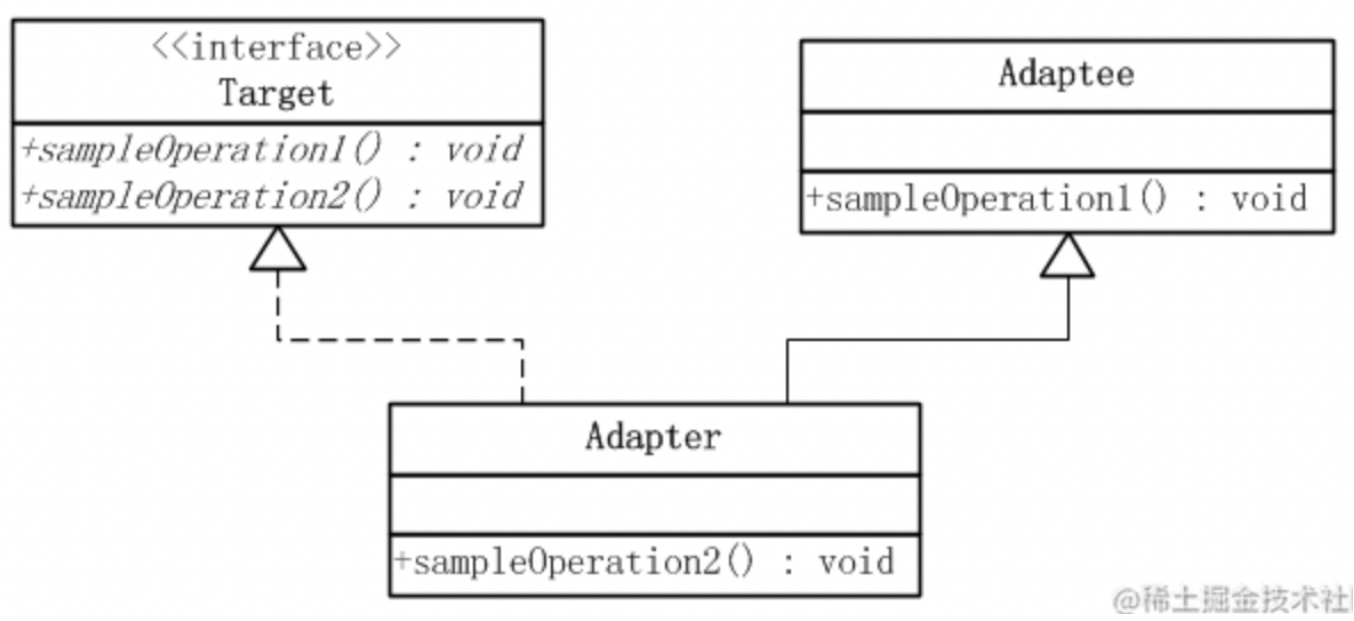
一个类只有一个实例

为实例提供一个全局访问节点

适配器

适配器模式把一个**类的接口**变换成客户端所期待的**另一种接口**，从而使原本因接口不匹配而无法在一起工作的两个类能够在一起工作

客户想用微改的target，所以adapter结合了target和adaptee。客户最终调用adapter



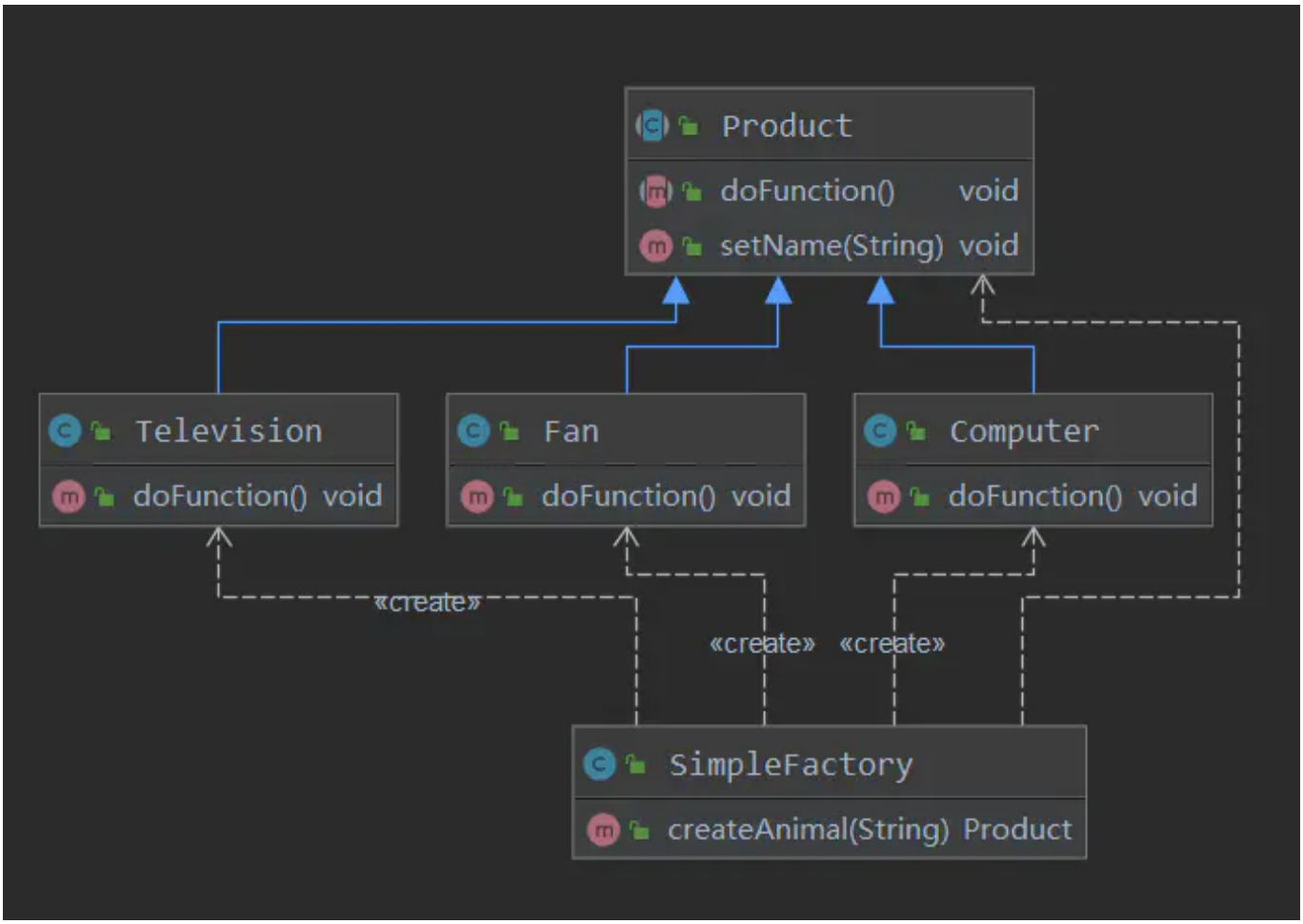
工厂

简单工厂

创建的对象 = “产品” 【】

创建“产品”的对象 = “工厂”【新建类实例的地方】

如果只有**一个**“工厂” = 简单工厂模式



对于产品种类相对较少的情况，考虑使用简单工厂模式。使用简单工厂模式的客户端只需要传入工厂类的参数，不需要关心如何创建对象的逻辑，可以很方便地创建所需产品。

优势：

1. 工厂类包含必要的逻辑判断，可以决定在什么时候创建哪一个产品的实例。客户端可以免除直接创建产品对象的职责，很方便的创建出相应的产品。工厂和产品的职责区分明确。
2. 客户端无需知道所创建具体产品的类名，只需知道参数即可。
3. 也可以引入配置文件，在不修改客户端代码的情况下更换和添加新的具体产品类。

缺点：

1. 简单工厂模式的工厂类单一，负责所有产品的创建，职责过重，一旦异常，整个系统将受影响。且工厂类代码会非常臃肿，违背高聚合原则。
2. 使用简单工厂模式会增加系统中类的个数（引入新的工厂类），增加系统的复杂度和理解难度
3. 系统扩展困难，一旦增加新产品不得不修改工厂逻辑，在产品类型较多时，可能造成逻辑过于复杂

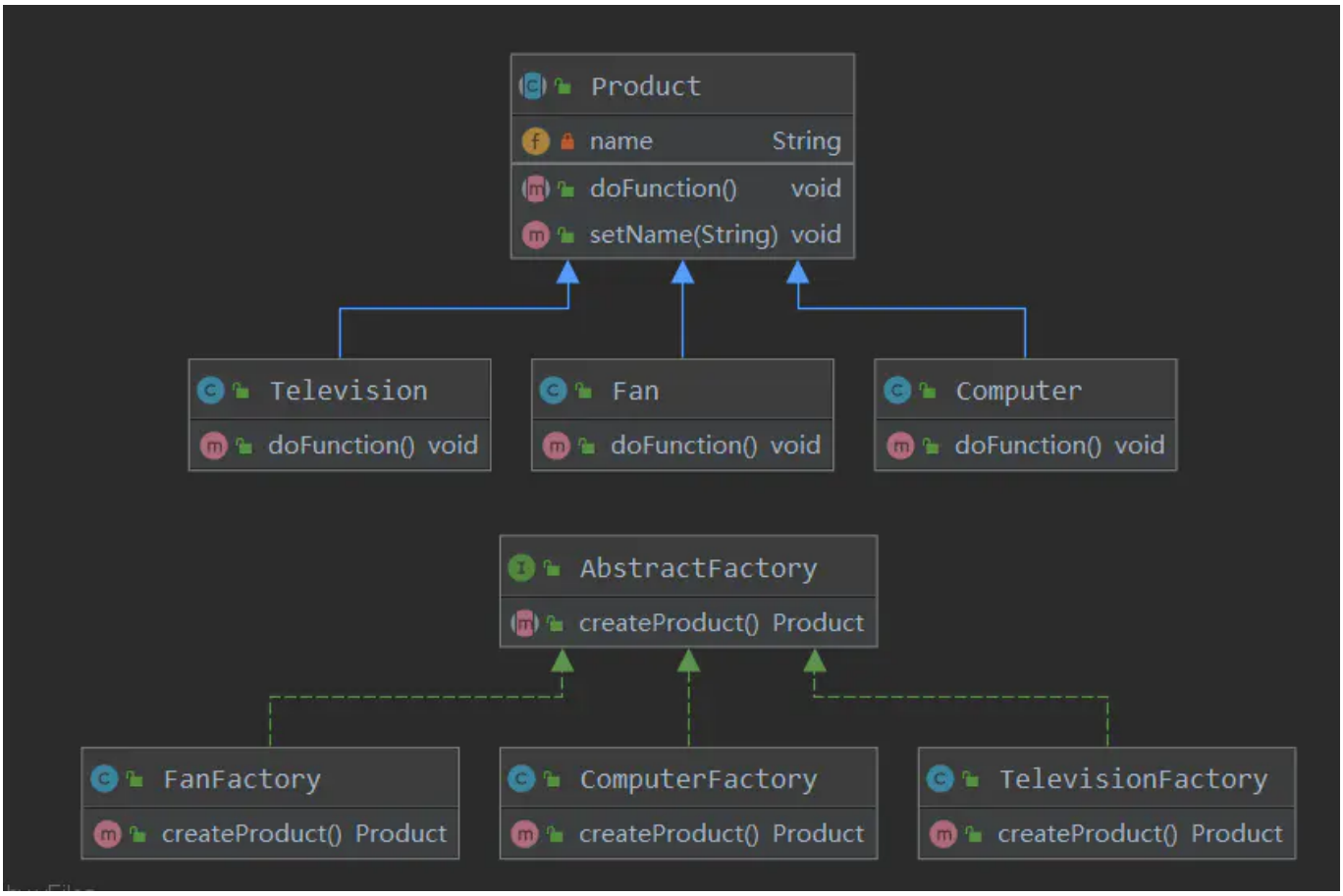
工厂方法（多态工厂模式）

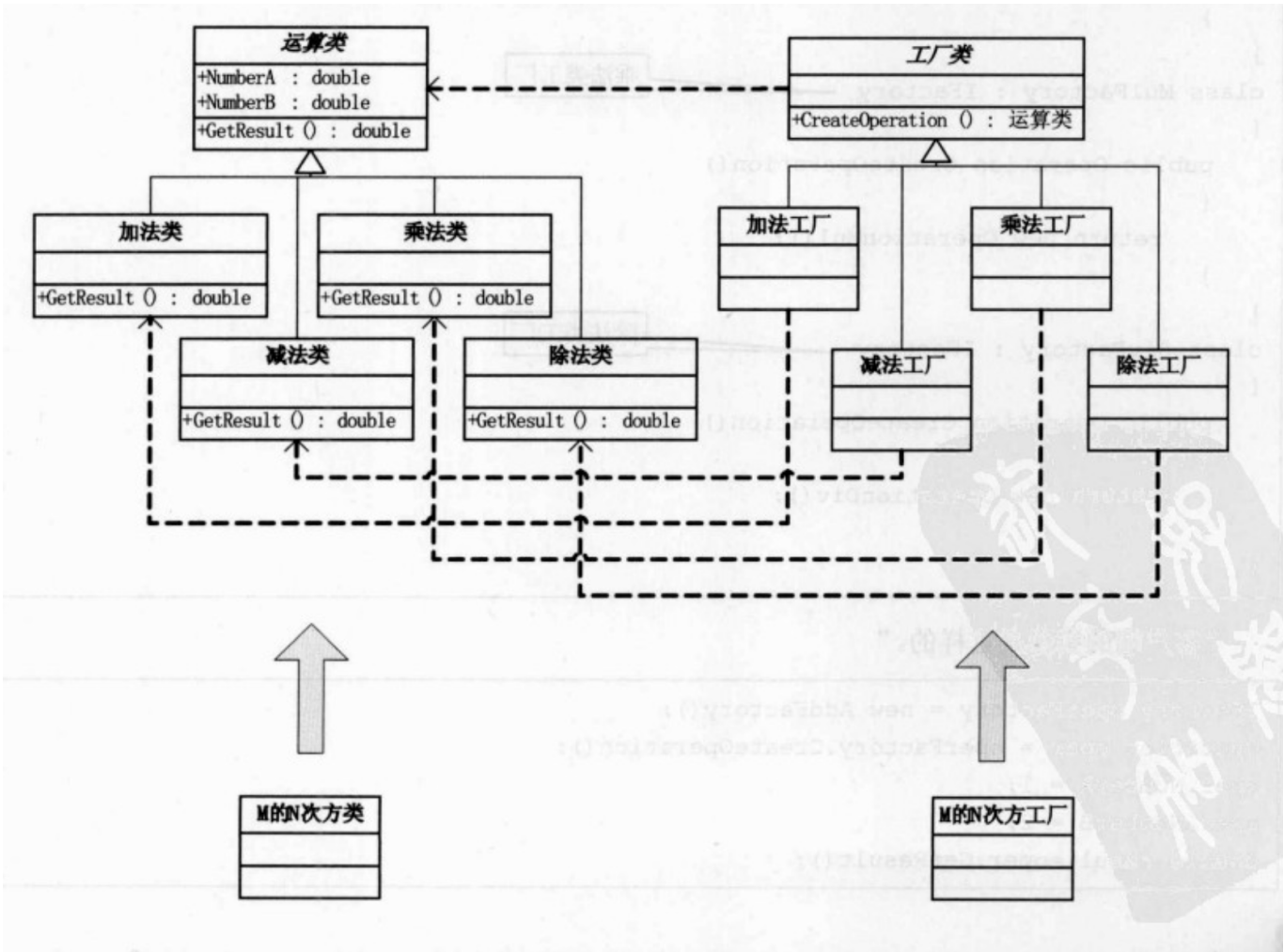
提供一个接口去创造object 但是让**子工厂类决定**什么产品类被实例

“工厂方法模式”是对简单工厂模式的进一步抽象化

在不修改原来代码的情况下引进新的产品，即满足开闭原则。

把产品的实例化推迟到子类

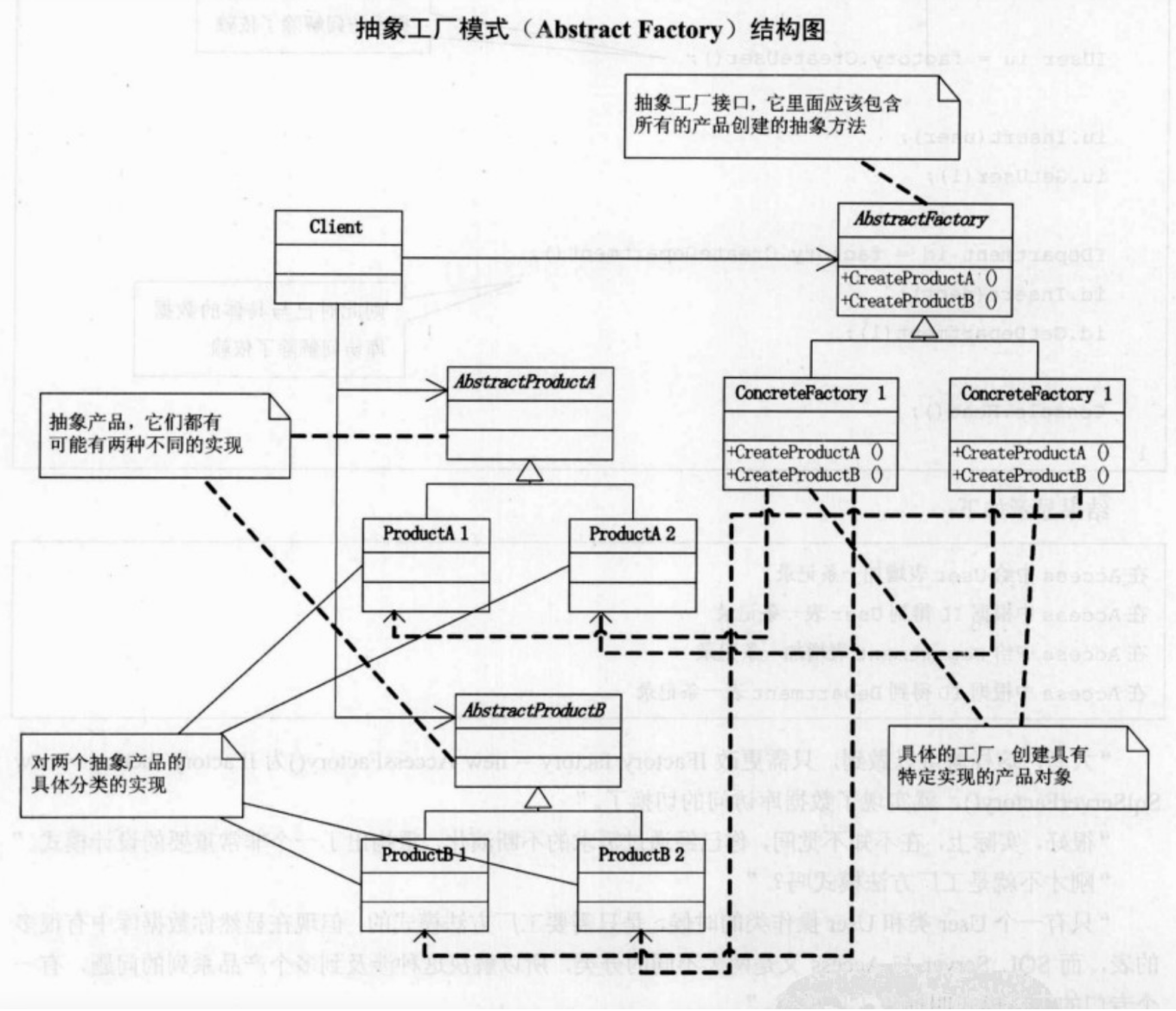

Product
name String
doFunction() void
setName(String) void
Television
doFunction() void
Fan
doFunction() void
Computer
doFunction() void
AbstractFactory
createProduct() Product
FanFactory
createProduct() Product
ComputerFactory
createProduct() Product
TelevisionFactory
createProduct() Product


运算类
+NumberA : double
+NumberB : double
+GetResult () : double
工厂类
+CreateOperation () : 运算类
加法类
+GetResult () : double
乘法类
+GetResult () : double
减法类
+GetResult () : double
除法类
+GetResult () : double
加法工厂
减法工厂
乘法工厂
除法工厂
M的N次方类
M的N次方工厂

主要角色：

- 抽象工厂（AbstractFactory）：提供了创建产品的接口，调用者通过它访问具体工厂的工厂方法 crateProduct() 来创建产品。
- 具体工厂（FanFactory、ComputerFactory、TelevisionFactory）：主要是实现抽象工厂中的抽象方法，完成具体产品的创建。
- 抽象产品（Product）：定义了产品的规范，描述了产品的主要特性和功能。
- 具体产品（Television、Fan、Computer）：实现了抽象产品角色所定义的接口，由具体工厂来创建，它同具体工厂之间一一对应。

抽象工厂



提供一个接口去创造families of related or dependent objects 不指定具体的类

和工厂方式的区别就是，工厂方式指定单独的类，但是抽象工厂比较multiple

策略模式

- 功能：具体算法从具体业务处理中独立
- 多个if-else出现考虑使用策略模式

- 策略算法是形同行为的不同实现（多态）
- 客户端选择，上下文来具体实现策略算法

策略模式：定义了算法家族，分别封装起来，让他们之间可以相互替换，此模式让算法的变化，不会影响到使用算法的客户。

前面写了简单工厂模式，其主旨是通过父类衍生出各种产品子类，再通过多态的方式生产各种产品。

但如果说出现的产品过多，就需要写大量的产品子类。这是无可避免的。

而策略模式，从类图上看与简单工厂类似，但其本质是封装算法逻辑，这是与简单工厂的主要区别。

责任链

建造者

观察者

装饰器