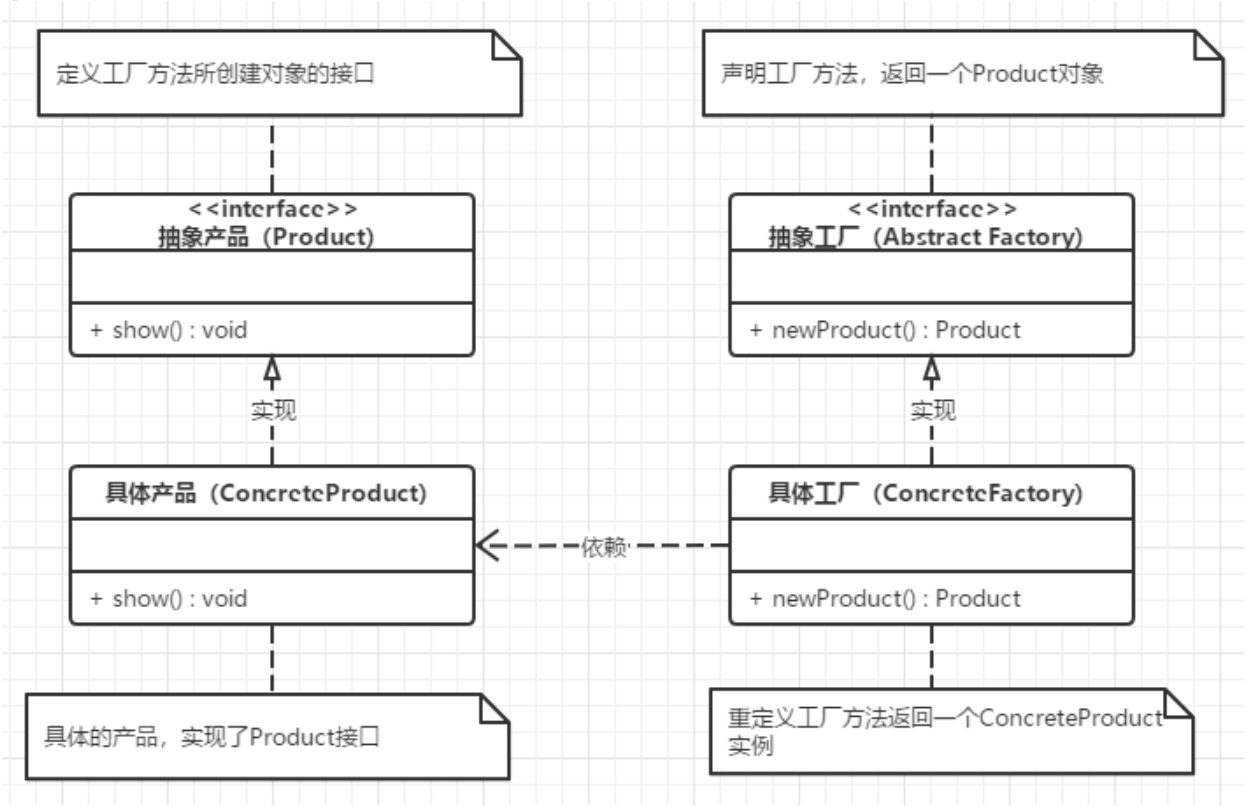
✿ 一、定义

定义一个用于创建对象的接口，让子类决定实例化哪一个类。工厂方法使一个类的实例化延迟到子类。

✿ 二、结构图

工厂方法模式的主要角色如下。
抽象工厂（Abstract Factory）：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 newProduct() 来创建产品。
具体工厂（ConcreteFactory）：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（ConcreteProduct）：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。



工厂方法模式结构图

✿ 三、优缺点

优点：
1.封装对象创建过程。只需要知道具体工厂的名称就可得到所要的产品，无须知道产品的具体创建过程
2.满足开闭原则。系统增加新产品时只需要添加具体产品类和对应的具体工厂类，无须对原工厂进行任何修改
缺点：
1.增加额外的开发量和系统的复杂度。每增加一个产品就要增加一个具体产品类和一个对应的具体工厂类

✿ 四、使用场景

1.客户只知道创建产品的工厂名，而不知道具体的产品名。如 TCL 电视工厂、海信电视工厂等。
2.创建对象的任务由多个具体子工厂中的某一个完成，而抽象工厂只提供创建产品的接口。
3.客户不关心创建产品的细节，只关心产品的品牌。

✿ 五、核心code

```
1  //抽象产品：提供了产品的接口
2  interface Product
3  {
```

```
 public void show();
}
//具体产品1：实现抽象产品中的抽象方法
class ConcreteProduct1 implements Product
{
 public void show()
  {
  System.out.println("我是具体产品1");
  }
 }
 //具体产品2：实现抽象产品中的抽象方法
 class ConcreteProduct2 implements Product
 {
  public void show()
  {
  System.out.println("我是具体产品2");
  }
 }
```

```
//抽象工厂：提供了厂品的生成方法
interface AbstractFactory
{
 public Product newProduct();
}
//具体工厂1：实现了厂品的生成方法
class ConcreteFactory1 implements AbstractFactory
{
 public Product newProduct()
  {
  System.out.println("我是具体工厂1");
  return new ConcreteProduct1();
  }
 }
 //具体工厂2：实现了厂品的生成方法
 class ConcreteFactory2 implements AbstractFactory
 {
  public Product newProduct()
  {
  System.out.println("我是具体工厂2");
  return new ConcreteProduct2();
```

```
 }
}
```

```
public static void main(String[] args) {
 //具体工厂1
 AbstractFactory factory = new ConcreteFactory1();
 factory.newProduct().show();
 //具体工厂2
 AbstractFactory factory2 = new ConcreteFactory2();
 factory2.newProduct().show();
}
 //输出
//我是具体工厂1
//我是具体产品1
//我是具体工厂2
//我是具体产品2
```