[**Java 之工厂方法和抽象工厂模式**](http://www.cnblogs.com/devinzhang/archive/2011/12/19/2293160.html)

1. 概念

**工厂方法**：一抽象产品类派生出多个具体产品类；一抽象工厂类派生出多个具体工厂类；每个具体工厂类只能创建一个具体产品类的实例。

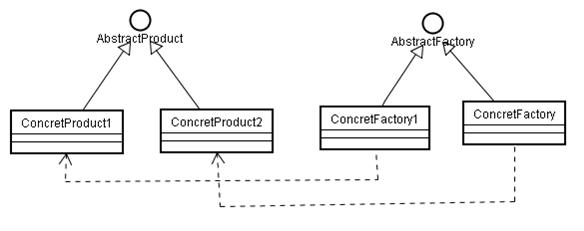
    即定义一个创建对象的接口（即抽象工厂类），让其子类（具体工厂类）决定实例化哪一个类（具体产品类）。“一对一”的关系。

**抽象工厂**：多个抽象产品类，派生出多个具体产品类；一个抽象工厂类，派生出多个具体工厂类；每个具体工厂类可创建多个具体产品类的实例。

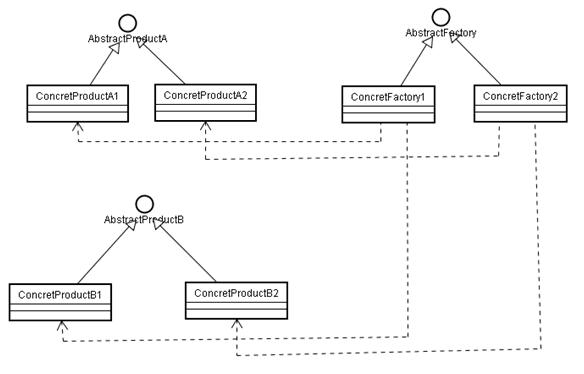
    即提供一个创建一系列相关或相互依赖对象的接口，而无需指定他们的具体的类。“一对多”的关系。

2. UML

**工厂方法**：



**抽象工厂**：



3. 代码

**工厂方法**：

[复制代码](javascript:void(0);)

public interface Product  
{   
}  
  
public interface Creator  
{  
 public Product factory();  
}  
  
public class ConcreteProduct1 implements Product  
{  
 public ConcreteProduct1()  
 {  
 System.out.println("ConcreteProduct1被创建");  
 }  
}  
  
public class ConcreteProduct2 implements Product  
{  
 public ConcreteProduct2()  
 {  
 System.out.println("ConcreteProduct2被创建");  
 }  
   
}  
  
public class ConcreteCreator1 implements Creator  
{  
 public Product factory()  
 {  
 return new ConcreteProduct1();  
 }  
}  
  
public class ConcreteCreator2 implements Creator  
{  
 public Product factory()  
 {  
 return new ConcreteProduct2();  
 }  
}  
   
public class Client  
{  
 private static Creator creator1, creator2;  
 private static Product prod1, prod2;  
  
 public static void main(String[] args)  
 {  
 creator1 = new ConcreteCreator1();  
 prod1 = creator1.factory();  
 System.out.println("----------------------------");  
 creator2 = new ConcreteCreator2();  
 prod2 = creator2.factory();  
 }  
}

[复制代码](javascript:void(0);)

**抽象工厂**：

[复制代码](javascript:void(0);)

//定义不同的产品之间的一定具备的标准，用interface实现   
//其中的method()方法可看作提取出不同产品的共性，如手机都有类似的功能   
interface IProductA{   
 public void method();   
}   
  
interface IProductB{   
 public void method();   
}   
  
//实现了产品标准实现的一系列具体产品   
//由于已经设计好A1由厂商1生产，故以下输出代码有“厂商x”   
class ProductA1 implements IProductA{   
 public void method() {   
 System.out.println("厂商1 生产ProductA1 ...");   
 }   
}   
  
class ProductA2 implements IProductA{   
 public void method() {   
 System.out.println("厂商2 生产ProductA2 ...");   
 }   
}   
  
class ProductB1 implements IProductB{   
 public void method() {   
 System.out.println("厂商1 生产ProductB1 ...");   
 }   
}   
  
class ProductB2 implements IProductB{   
 public void method() {   
 System.out.println("厂商2 生产ProductB2 ...");   
 }   
}   
  
//每一种牌子的产品生产工厂，即不同的厂商负责自己牌子产品的生产   
abstract class Factory1{   
 abstract IProductA getProductA1();   
 abstract IProductB getProductB1();   
}   
  
abstract class Factory2{   
 abstract IProductA getProductA2();   
 abstract IProductB getProductB2();   
}   
  
//具体的工厂用来生产相关的产品   
class ConcreteFactory1 extends Factory1{   
 public IProductA getProductA1() {   
 return new ProductA1();   
 }   
 public IProductB getProductB1() {   
 return new ProductB1();   
 }   
}   
  
class ConcreteFactoryB extends Factory2{   
 public IProductA getProductA2() {   
 return new ProductA2();   
 }   
 public IProductB getProductB2() {   
 return new ProductB2();   
 }   
}   
  
//测试类   
public class Client {   
 public static void main(String[] args) {   
 //厂商1负责生产产品A1、B1   
 Factory1 factory1 = new ConcreteFactory1();   
 IProductA productA1 = factory1.getProductA1();   
 IProductB productB1 = factory1.getProductB1();   
   
 productA1.method();   
 productB1.method();   
   
 //厂商2负责生产产品A2、B2   
 Factory2 factory2 = new ConcreteFactoryB();   
 IProductA productA2 = factory2.getProductA2();   
 IProductB productB2 = factory2.getProductB2();   
   
 productA2.method();   
 productB2.method();   
 }   
}

[复制代码](javascript:void(0);)

4. 应用场景

**工厂方法：**

在以下情况下，适用于工厂方法模式：

(1) 当一个类不知道它所必须创建的对象的类的时候。

(2) 当一个类希望由它的子类来指定它所创建的对象的时候。

(3) 当类将创建对象的职责委托给多个帮助子类中的某一个，并且你希望将哪一个帮助子类是代理者这一信息局部化的时候。

**抽象工厂：**

(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，这对于所有形态的工厂模式都是重要的。

(2) 这个系统有多于一个的产品族，而系统只消费其中某一产品族。

(3) 同属于同一个产品族的产品是在一起使用的，这一约束必须在系统的设计中体现出来。

(4) 系统提供一个产品类的库，所有的产品以同样的接口出现，从而使客户端不依赖于实现。