## 定义：

工厂模式属于创建型模式，它提供了一种创建对象的最佳方法。

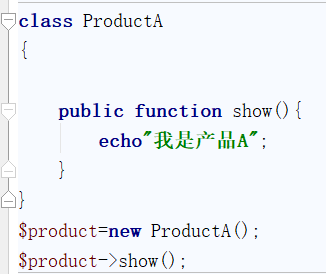
通常把用类创建对象的过程称为实例化

最后这边再说一下什么是实例化对象。小编突然像想到这样一个问题我的第一反应是不就是new嘛。可是真让我说我竟以实语塞。后来仔细想想。这几个字要分开来想 $student =new student();前面的$ student是我们的类对象而用new 关键字创建类对象的过程就是我们所说的实例化。

## 分类：

## 一．为什么要使用工厂模式

什么是工厂模式呢，很多地方有这样说道：工厂模式就是为了把对象的创建与使用的过程分开或者是将“类实例化的操作”与“使用对象的操作”分开。那我们怎么来理解这一句话呢。举个简单的例子。我们像调用ProductA类里面的show方法，我们一般的的做法是下面这样



这就是我们平时用new来实例化对象进而使用类中的方法。这样子的确没有什么问题。

但是如果一个类的构造函数很复杂，需要用到其他的类

**class** size  
{  
 **protected $size**;  
 **public function** \_\_construct($size)  
 {  
 $this->**size**=$size;  
 }  
 **public function** size(){  
 **echo** $this->**size**;  
 }  
}  
**class** name  
{  
 **protected $name**;  
 **public function** \_\_construct($name)  
 {  
 $this->**name**=$name;  
 }  
 **public function** name(){  
 **echo** $this->**name**;  
 }  
}  
**class** ProductA  
{  
 **protected $name**,**$size**;  
 **public function** \_\_construct(name $name,size $size)  
 {  
 $this->**name**=$name;  
 $this->**size**=$size;  
 }  
 **public function** show(){  
 **echo** $this->**name**->name();  
 **echo** $this->**size**->size();  
 }  
}  
$name=**new** name(**'产品A'**);  
$size=**new** size(**'50X50'**);  
$product=**new** ProductA($name,$size);  
$product->show();

这里面我们可以看到ProductA类的构造需要用到name和size类，也许这样这看起来有人说不是很复杂可以直接new。但是如ProductA类的构造函数需要很多个其他类的参数，$product=**new** ProductA($name,$size,$color……)

那我每new一个ProductA类就要new很多构造函数用到的其他类，一但代码中用的需要实例化很多ProductA类，那么此时就会有很多重复的代码，假设我们上面一样写了很多的代码，有一天发现ProductA类的构造函数的参数需要修改增或删除一个参数，或者修改ProductA类的名字为ProductB。那么我们就要一个个去修改我们new ProductA类的地方。这显然不现实，很费时间。那么造成这样的原因就是new 把对象的创建和使用的过程放在了一起，每次使用一个类时我们都要去创建一个类对象，耦合度高。此外我们还可以看到每一次new我们都要知道ProductA的构造函数有哪些参数，即对象的创建逻辑与过程。如果

对象的创建逻辑与过程很复杂很繁琐，那么我们每一次都要去理一遍显然不合理，也很麻烦。

一次我们使用工厂模式。

通过上面的的例子我们可以总结出来使用工厂模式的好处就是：

1. 解耦，将“类实例化的操作”与“使用对象的操作”分开。
2. 降低代码重复
3. 然使用者不需要知道类对象创建的逻辑，直接使用其中方法就好了。

## 二．分类

### 1.简单工厂

简单工厂又叫做静态方法模式。工厂类中定义了静态方法用来创建产品类实例。使用的时候直接调用静态方法。

#### 使用步骤

* 创建**抽象产品类** & 定义具体产品的公共接口；
* 创建**具体产品类**（继承抽象产品类） & 定义生产的具体产品；
* 创建**工厂类**，通过创建静态方法根据传入不同参数从而创建不同具体产品类的实例；
* 外界通过调用工厂类的静态方法，**传入不同参数**从而创建不同**具体产品类的实例**
* **<?php  
    
  class** size  
  {  
   **protected $size**;  
   **public function** \_\_construct($size)  
   {  
   $this->**size**=$size;  
   }  
   **public function** size(){  
   **echo** $this->**size**;  
   }  
  }  
  **class** name  
  {  
   **protected $name**;  
   **public function** \_\_construct($name)  
   {  
   $this->**name**=$name;  
   }  
   **public function** name(){  
   **echo** $this->**name**;  
   }  
  }  
    
  *//具体产品的公共接口***interface** Shoes{  
   **public function** show();  
  }  
  *//具体产品类***class** Shoes**A implements** Shoes  
  {  
   **protected $name**,**$size**;  
   **public function** \_\_construct(name $name,size $size)  
   {  
   $this->**name**=$name;  
   $this->**size**=$size;  
   }  
   **public function** show()  
   {  
   *//* ***TODO: Implement show() method.* echo** $this->**name**->name();  
   **echo** $this->**size**->size();  
   }  
  }  
  **class** ShoesB **implements** Shoes  
  {  
   **protected $name**,**$size**;  
   **public function** \_\_construct(name $name,size $size)  
   {  
   $this->**name**=$name;  
   $this->**size**=$size;  
   }  
   **public function** show(){  
   **echo** $this->**name**->name();  
   **echo** $this->**size**->size();  
   }  
  }  
  *//工厂类*

**class** Factory {  
 **static function** createshoes($name,$size,$class){  
  
 **if**($class==**"A"**){  
 $name1=**new** name($name);  
 $size1=**new** size($size);  
 **return new** ShoesA($name1,$size1);  
 }  
 **else if**($class==**'B'**){  
 $name1=**new** name($name);  
 $size1=**new** size($size);  
 **return new** ShoesB($name1,$size1);  
 }  
 }  
}  
 Factory:: createshoes(**'鞋子A'**,**'50x50'**,**"A"**)->show();  
  
简单工厂的缺点：

将所有的产品即实例放到一个工厂中，一但这个工厂出错那么所有的产品都无法生产

因此为了克服这个确定我们产生了工厂方法模式。

### 二．工厂方法模式

工厂方法模式是将类的实例化延迟到工厂类的子类中完成，即由子类来决定应该实例化哪一个类，哪个产品由哪个工厂生产。

**<?php  
  
class** size  
{  
 **protected $size**;  
 **public function** \_\_construct($size)  
 {  
 $this->**size**=$size;  
 }  
 **public function** size(){  
 **echo** $this->**size**;  
 }  
}  
**class** name  
{  
 **protected $name**;  
 **public function** \_\_construct($name)  
 {  
 $this->**name**=$name;  
 }  
 **public function** name(){  
 **echo** $this->**name**;  
 }  
}  
  
*//具体产品的公共接口***interface** Shoes{  
 **public function** show();  
}  
*//具体产品类***class** ShoesA **implements** Shoes  
{  
 **protected $name**,**$size**;  
 **public function** \_\_construct(name $name,size $size)  
 {  
 $this->**name**=$name;  
 $this->**size**=$size;  
 }  
 **public function** show()  
 {  
 *//* ***TODO: Implement show() method.* echo** $this->**name**->name();  
 **echo** $this->**size**->size();  
 }  
}  
**class** ShoesB **implements** Shoes  
{  
 **protected $name**,**$size**;  
 **public function** \_\_construct(name $name,size $size)  
 {  
 $this->**name**=$name;  
 $this->**size**=$size;  
 }  
 **public function** show(){  
 **echo** $this->**name**->name();  
 **echo** $this->**size**->size();  
 }  
}  
*//工厂类***interface** Factory {  
 **public function** createshoes($name,$size);  
}  
**class** FactoryA **implements** Factory {  
  
 **public function** createshoes($name, $size)  
 {  
 *//* ***TODO: Implement create() method.*** $name1=**new** name($name);  
 $size1=**new** size($size);  
 **return new** ShoesA($name1,$size1);  
  
 }  
}  
**class** FactoryB **implements** Factory {  
  
 **public function** createshoes($name, $size)  
 {  
 *//* ***TODO: Implement create() method.*** $name1=**new** name($name);  
 $size1=**new** size($size);  
 **return new** ShoesB($name1,$size1);  
  
 }  
}  
 $factorya=**new** FactoryA();  
$factorya->createshoes(**'鞋子A'**,**'50x50'**)->show();  
$factoryb=**new** FactoryA();  
$factoryb->createshoes(**'鞋子B'**,**'100x100'**)->show();  
  
缺点：每个工厂只能创建一类产品

## 三．抽象工厂模式

从工厂方法模式中我们可以看到FactoryA，FactoryB只能生产不同品牌的鞋子要是我们想让它们生产汽车它们就没有办法了。因此我们使用工厂模式用于每个工厂可以创建多种类的产品

**<?php  
  
class** size  
{  
 **protected $size**;  
 **public function** \_\_construct($size)  
 {  
 $this->**size**=$size;  
 }  
 **public function** size(){  
 **echo** $this->**size**;  
 }  
}  
**class** name  
{  
 **protected $name**;  
 **public function** \_\_construct($name)  
 {  
 $this->**name**=$name;  
 }  
 **public function** name(){  
 **echo** $this->**name**;  
 }  
}  
  
*//具体产品的公共接口***interface** Shoes{  
 **public function** show();  
}  
*//具体产品类***class** ShoesA **implements** Shoes  
{  
 **protected $name**,**$size**;  
 **public function** \_\_construct(name $name,size $size)  
 {  
 $this->**name**=$name;  
 $this->**size**=$size;  
 }  
 **public function** show()  
 {  
 *//* ***TODO: Implement show() method.* echo** $this->**name**->name();  
 **echo** $this->**size**->size();  
 }  
}  
**class** ShoesB **implements** Shoes  
{  
 **protected $name**,**$size**;  
 **public function** \_\_construct(name $name,size $size)  
 {  
 $this->**name**=$name;  
 $this->**size**=$size;  
 }  
 **public function** show(){  
 **echo** $this->**name**->name();  
 **echo** $this->**size**->size();  
 }  
}  
**interface** Car{  
 **public function** show();  
}  
**class** CarA **implements** Car  
{  
  
 **public function** show()  
 {  
 **echo"生产汽车A"**;  
 }  
}  
**class** CarB **implements** Car  
{  
 **public function** show(){  
 **echo"生产汽车B"**;  
 }  
}  
*//工厂类***interface** Factory {  
 **public function** createshoes($name,$size);  
 **public function** createcars();  
}  
**class** FactoryA **implements** Factory {  
  
 **public function** createshoes($name, $size)  
 {  
 *//* ***TODO: Implement create() method.*** $name1=**new** name($name);  
 $size1=**new** size($size);  
 **return new** ShoesA($name1,$size1);  
  
 }  
  
 **public function** createcars()  
 {  
 *//* ***TODO: Implement createcars() method.* return new** CarA();  
 }  
}  
**class** FactoryB **implements** Factory {  
  
 **public function** createshoes($name, $size)  
 {  
 *//* ***TODO: Implement create() method.*** $name1=**new** name($name);  
 $size1=**new** size($size);  
 **return new** ShoesB($name1,$size1);  
  
 }  
  
 **public function** createcars()  
 {  
 *//* ***TODO: Implement createcars() method.* return new** CarB();  
 }  
}  
 $factorya=**new** FactoryA();  
$factorya->createshoes(**'鞋子A'**,**'50x50'**)->show();  
$factorya->createcars()->show();  
$factoryb=**new** FactoryA();  
$factoryb->createshoes(**'鞋子B'**,**'100x100'**)->show();  
$factoryb->createcars()->show();