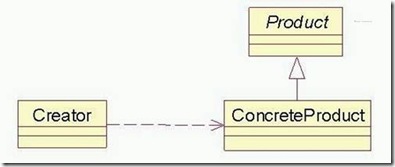
工厂模式主要是为创建对象提供了接口。工厂模式按照《Java与模式》中的提法分为三类：  
1. 简单工厂模式(Simple Factory)   
2. 工厂方法模式(Factory Method)   
3. 抽象工厂模式(Abstract Factory)   
这三种模式从上到下逐步抽象，并且更具一般性。还有一种分类法，就是将简单工厂模式看为工厂方法模式的一种特例，两个归为一类。下面是使用工厂模式的两种情况：  
1.在编码时不能预见需要创建哪种类的实例。  
2.系统不应依赖于产品类实例如何被创建、组合和表达的细节

三、简单工厂模式  
顾名思义，这个模式本身很简单，而且使用在业务较简单的情况下。  
它由三种角色组成（关系见下面的类图）：  
1、工厂类角色：这是本模式的核心，含有一定的商业逻辑和判断逻辑。在java中它往往由一个具体类实现。

2、抽象产品角色：它一般是具体产品继承的父类或者实现的接口。在java中由接口或者抽象类来实现。

3、具体产品角色：工厂类所创建的对象就是此角色的实例。在java中由一个具体类实现。  
[](http://images.cnblogs.com/cnblogs_com/poissonnotes/WindowsLiveWriter/3b3b76bcffad_13405/clip_image002_2.jpg)  
那么简单工厂模式怎么用呢？我来举个例子吧，我想这个比讲一大段理论上的文字描述要容易理解的多！下面就来给那个暴发户治病: P   
在使用了简单工厂模式后，现在暴发户只需要坐在车里对司机说句："开车"就可以了。来看看怎么实现的：

//抽象产品角色

public interface Car{

public void drive();

}

//具体产品角色

public class Benz implements Car{

public void drive() {

System.out.println("Driving Benz ");

}

}

public class Bmw implements Car{

public void drive() {

System.out.println("Driving Bmw ");

}

}

。。。（奥迪我就不写了:P）

//工厂类角色

public class Driver{

//工厂方法

//注意 返回类型为抽象产品角色

public static Car driverCar(String s)throws Exception {

//判断逻辑，返回具体的产品角色给Client

if(s.equalsIgnoreCase("Benz")) return new Benz();

else if(s.equalsIgnoreCase("Bmw"))

return new Bmw();

......

else throw new Exception();

。。。

//欢迎暴发户出场......

public class Magnate{

public static void main(String[] args){

try{

//告诉司机我今天坐奔驰

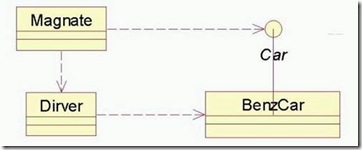
Car car = Driver.driverCar("benz");

//下命令：开车

car.drive();

。。。

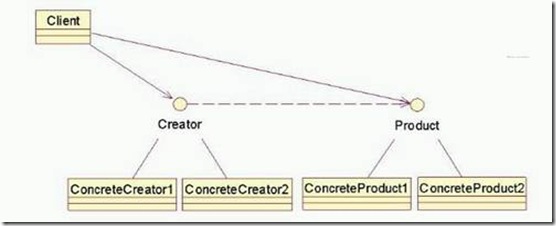
如果将所有的类放在一个文件中，请不要忘记只能有一个类被声明为public。 程序中类之间的关系如下：

[](http://images.cnblogs.com/cnblogs_com/poissonnotes/WindowsLiveWriter/3b3b76bcffad_13405/clip_image004_2.jpg)

这便是简单工厂模式了。下面是其好处：

首先，使用了简单工厂模式后，我们的程序不在"有病"，更加符合现实中的情况；而且客户端免除了直接创建产品对象的责任，而仅仅负责"消费"产品（正如暴发户所为）。  
下面我们从开闭原则上来分析下简单工厂模式。当暴发户增加了一辆车的时候，只要符合抽象产品制定的合同，那么只要通知工厂类知道就可以被客户使用了。那么对于产品部分来说，它是符合开闭原则的--对扩展开放、对修改关闭；但是工厂部分好像不太理想，因为每增加一辆车，都要在工厂类中增加相应的商业逻辑和判断逻辑，这显自然是违背开闭原则的。  
对于这样的工厂类（在我们的例子中是为司机师傅），我们称它为全能类或者上帝类。  
我们举的例子是最简单的情况，而在实际应用中，很可能产品是一个多层次的树状结构。由于简单工厂模式中只有一个工厂类来对应这些产品，所以这可能会把我们的上帝类坏了，进而累坏了我们可爱的程序员:(   
正如我前面提到的简单工厂模式适用于业务将简单的情况下。而对于复杂的业务环境可能不太适应阿。这就应该由工厂方法模式来出场了！！

四、工厂方法模式  
先来看下它的组成吧：  
1、抽象工厂角色：这是工厂方法模式的核心，它与应用程序无关。是具体工厂角色必须实现的接口或者必须继承的父类。在java中它由抽象类或者接口来实现。  
2、具体工厂角色：它含有和具体业务逻辑有关的代码。由应用程序调用以创建对应的具体产品的对象。在java中它由具体的类来实现。  
3、抽象产品角色：它是具体产品继承的父类或者是实现的接口。在java中一般有抽象类或者接口来实现。  
4、具体产品角色：具体工厂角色所创建的对象就是此角色的实例。在java中由具体的类来实现。  
来用类图来清晰的表示下的它们之间的关系：

[](http://images.cnblogs.com/cnblogs_com/poissonnotes/WindowsLiveWriter/3b3b76bcffad_13405/clip_image006_2.jpg)

我们还是老规矩使用一个完整的例子来看看工厂模式各个角色之间是如何来协调的。话说暴发户生意越做越大，自己的爱车也越来越多。这可苦了那位司机师傅了，什么车它都要记得，维护，都要经过他来使用！于是暴发户同情他说：看你跟我这么多年的份上，以后你不用这么辛苦了，我给你分配几个人手，你只管管好他们就行了！于是，工厂方法模式的管理出现了。代码如下：  
//抽象产品角色，具体产品角色与简单工厂模式类似，只是变得复杂了些，这里略。

//抽象工厂角色

public interface Driver{

public Car driverCar();

}

public class BenzDriver implements Driver{

public Car driverCar(){

return new Benz();

}

}

public class BmwDriver implements Driver{

public Car driverCar() {

return new Bmw();

}

}

......//应该和具体产品形成对应关系，这里略...

//有请暴发户先生

public class Magnate

{

public static void main(String[] args)

{

try{

Driver driver = new BenzDriver();

Car car = driver.driverCar();

car.drive();

}catch(Exception e)

{ }

}

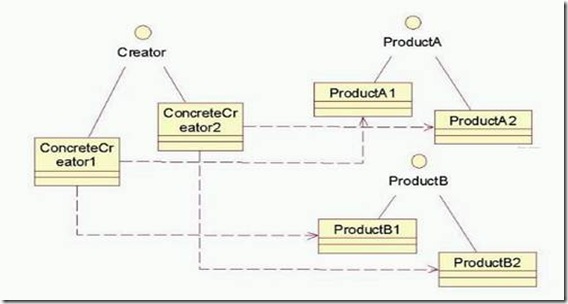
}

工厂方法使用一个抽象工厂角色作为核心来代替在简单工厂模式中使用具体类作为核心。让我们来看看工厂方法模式给我们带来了什么？使用开闭原则来分析下工厂方法模式。当有新的产品（即暴发户的汽车）产生时，只要按照抽象产品角色、抽象工厂角色提供的合同来生成，那么就可以被客户使用，而不必去修改任何已有的代码。看来，工厂方法模式是完全符合开闭原则的！  
使用工厂方法模式足以应付我们可能遇到的大部分业务需求。但是当产品种类非常多时，就会出现大量的与之对应的工厂类，这不应该是我们所希望的。所以我建议在这种情况下使用简单工厂模式与工厂方法模式相结合的方式来减少工厂类：即对于产品树上类似的种类（一般是树的叶子中互为兄弟的）使用简单工厂模式来实现。  
当然特殊的情况，就要特殊对待了：对于系统中存在不同的产品树，而且产品树上存在产品族，那么这种情况下就可能可以使用抽象工厂模式了。

五、小结  
让我们来看看简单工厂模式、工厂方法模式给我们的启迪：  
如果不使用工厂模式来实现我们的例子，也许代码会减少很多--只需要实现已有的车，不使用多态。但是在可维护性上，可扩展性上是非常差的（你可以想象一下，添加一辆车后要牵动的类）。因此为了提高扩展性和维护性，多写些代码是值得的。

六、抽象工厂模式  
先来认识下什么是产品族：位于不同产品等级结构中，功能相关联的产品组成的家族。如果光看这句话就能清楚的理解这个概念，我不得不佩服你啊。还是让我们用一个例子来形象地说明一下吧。

图中的BmwCar和BenzCar就是两个产品树（产品层次结构）；而如图所示的BenzSportsCar和BmwSportsCar就是一个产品族。他们都可以放到跑车家族中，因此功能有所关联。同理BmwBussinessCar和BenzSportsCar也是一个产品族。  
回到抽象产品模式的话题上，可以这么说，它和工厂方法模式的区别就在于需要创建对象的复杂程度上。而且抽象工厂模式是三个里面最为抽象、最具一般性的。抽象工厂模式的用意为：给客户端提供一个接口，可以创建多个产品族中的产品对象。而且使用抽象工厂模式还要满足一下条件：  
1.系统中有多个产品族，而系统一次只可能消费其中一族产品  
2.同属于同一个产品族的产品一起使用时。  
来看看抽象工厂模式的各个角色（和工厂方法的如出一辙）：  
抽象工厂角色：这是工厂方法模式的核心，它与应用程序无关。是具体工厂角色必须实现的接口或者必须继承的父类。在java中它由抽象类或者接口来实现。  
具体工厂角色：它含有和具体业务逻辑有关的代码。由应用程序调用以创建对应的具体产品的对象。在java中它由具体的类来实现。  
抽象产品角色：它是具体产品继承的父类或者是实现的接口。在java中一般有抽象类或者接口来实现。  
具体产品角色：具体工厂角色所创建的对象就是此角色的实例。在java中由具体的类来实现。

[](http://images.cnblogs.com/cnblogs_com/poissonnotes/WindowsLiveWriter/3b3b76bcffad_13405/clip_image008_2.jpg)

看过了前两个模式，对这个模式各个角色之间的协调情况应该心里有个数了，我就不举具体的例子了。只是一定要注意满足使用抽象工厂模式的条件哦，不然即使存在了多个产品树，也存在产品族，但是不能使用的。