**工厂设计模式**

问题的提出

提起工厂设计模式，就有两个关键的角色：产品和工厂。因此工厂模式的关键就是把握好这两者之间的关系。可分以下四种情况：

1.单一产品系

也就是说这个工厂只生产一中类型的产品。比如小汽车厂，只生产奥迪A6，可以生产高配，中配，低配的3中类型的小汽车。

2.多产品系，特征相同

工厂生产多种类型的产品，各种类型的产品特征都相同。比如，工厂里还生产奥迪A8,也有高配，中配，低配三种类型。

3.多产品系，部分特征相同

工厂生产多种类型的产品，各种类型的产品部分特征相同。比如，工厂还生产奥迪S系列，只有高配，中配，两种类型。

4.多产品系，无特征相同

比如：工厂生产奥迪A1，只有低配，而S系列只有高配和中配。

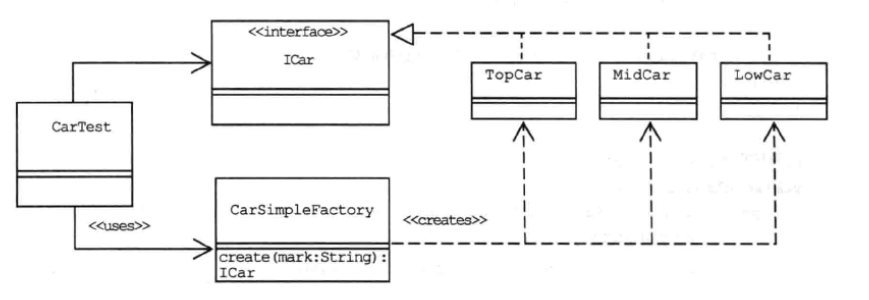
**工厂模式一般分为简单工厂，工厂，和抽象工厂，属于创建型设计模式。**

上面的工厂生产汽车的几种情况可以分别映射到这几种工厂模式中，下面加以描述。

一、简单工厂

|  |
| --- |
| public class ICar { }///定制抽象的产品 |
| public class TopCar extends ICar {  public TopCar() {  System.*out*.println("高配");  } }////定制具体的子类产品 |
| public class MiddleCar extends ICar {  public MiddleCar() {  System.*out*.println("中配");  } }////定制具体的子类产品 |
| public class LowCar extends ICar {  public LowCar() {  System.*out*.println("低配");  }  }////定制具体的子类产品 |
| public class CarFactory {  private static final String *TYPE\_TOP* = "top"; //高配  private static final String *TYPE\_MIDDLE* = "mid"; //中配  private static final String *TYPE\_LOW* = "low";    public static ICar create(String type){  ICar car = null;  if (*TYPE\_TOP*.equals(type)){  car = new TopCar();  }else if(*TYPE\_MIDDLE*.equals(type)){  car = new MiddleCar();  }else if(*TYPE\_LOW*.equals(type)){  car = new LowCar();  }  return car;  } } ///// 定制共厂类。 |
| public class DemoTest {  public static void main(String[] args) {  CarFactory.*create*("top");  CarFactory.*create*("mid");  CarFactory.*create*("low");  } } |

图示：



二、工厂

|  |
| --- |
| public interface ICar { } |
| public class LowCar implements ICar {  public LowCar() {  System.*out*.println("低配");  } } |
| public class MiddleCar implements ICar {  public MiddleCar() {  System.*out*.println("中配");  } } |
| public class TopCar implements ICar {  public TopCar() {  System.*out*.println("高配");  } } |
| public interface IFactory {  ICar create(); } |
| public class LowCarFactory implements IFactory {  @Override  public ICar create() {  return new LowCar();  } } |
| public class MiddleCarFactory implements IFactory {  @Override  public ICar create() {  return new MiddleCar();  } } |
| public class TopCarFactory implements IFactory {  @Override  public ICar create() {  return new TopCar();  } } |
| public class DemoTest {  public static void main(String[] args) {  TopCarFactory topCarFactory = new TopCarFactory();  topCarFactory.create();    MiddleCarFactory middleCarFactory = new MiddleCarFactory();  middleCarFactory.create();    LowCarFactory lowCarFactory = new LowCarFactory();  lowCarFactory.create();  } } |

工厂模式编写步骤：

1. 定制抽象接口，ICar
2. 定制具体的产品子类 TopCar,MiddleCar,LowCar
3. 定制抽象的工厂类或者接口，
4. 定制具体的工厂子类，负责生产专门的产品。

简单工厂模式和工厂模式的区别：

工厂模式把简单工厂中具体的工厂类划分为两层，抽象工厂层和具体工厂子类层，

抽象工厂层丰富了程序框架的内涵，符合从一般到特殊的语义特点。

工厂方法更易于软件的二次开发。当需求分析发生变化时，只需要增加，删除相应的类型，而不是修改已有的类。例如要生产一种超高档的小汽车，只需要添加SuperCar和SuperFactory就好了。

三、抽象工厂

简单工厂、工厂模式是单产品系的，抽象工厂是多产品系的。从本质上说，抽象工厂，工厂模式是统一的。

代码太多，现附上github：

编写抽象工厂模式的步骤：

1. 定制抽象产品接口
2. 定制具体产品子类
3. 定制抽象工厂类
4. 定制具体工厂子类。

其他应用：

编写文件读写功能：

读取文本文件，包括GBK，UTF8，UNICODE编码下的文本文件，要去获取全文内容；

读取图像文件BMP,GIF,JPG文件，要求获取图像的宽度，长度，每一点的RGB的三基色信息。

