**第三章 简单工厂模式**

**一、简单工厂模式概述**

简单工厂模式并不属于创建型模式的一种，但它的特点与创建型模式的特点类似，即将对象的创建和使用分离，用户在创建对象时不用考虑对象的使用，反之亦然，从而降低了系统的耦合度，让设计方案更易于修改和扩展。

简单工厂模式：定义一个工厂类，它可以根据参数的不同返回不同的类的实例，被创建的实例通常都有共同的父类（举个栗子：通俗点来讲就是水果农场（工厂类）会根据客户（客户端）提供的不同的水果名称（参数）给予客户不同的水果（类的实例））。

简单工厂模式的要点在于当用户需要什么时，只需要传入一个正确的参数就可以获取所需要的对象，而无需直到其创建细节。

**二、简单工厂模式结构与实现**

****

简单工厂模式结构图

**（有关UML类图中类之间关系的表示可参考课本P396-P402，这里有很详细的介绍。）**

（1）Factory（工厂角色）：即工厂类，负责创建所有产品实例（即产品对象），创建的方法通常使用静态方法，从而能直接被客户端或者其他外界调用，创建所需的产品对象。典型代码如下：

**public** **class** Factory{

•public修饰的static成员方法本质是全局方法，在客户端类中可以直接**用类名调用**该方法，即product=Factory.*getProduct*("A")

//静态工厂方法

**public** **static** Product getProduct(String arg){

Product product = **null**;

**if**(arg.equalsIgnoreCase("A")) {

product = **new** ConcreteProductA();

//初始化设置product

...

}

**else** **if**(arg.equalsIgnoreCase("B")) {

product = **new** ConcreteProductB();

//初始化设置product

...

}

//if-else结构，意味着如果想要添加新的具体产品，就要重新修改这部分的代码，违反了开闭原则，不利于系统的扩展和维护，这也是简单工厂方法的一个缺点

**return** product;

}

}

（2）Product（抽象产品角色）：它是工厂类创建的所有对象的父类，封装了各种产品对象的公有方法，这样工厂类中只需定义一个通用的工厂方法。

典型代码如下：

•抽象类

public abstract class 类名

抽象类一般将事物的共性的东西提取出来，由子类继承去实现，易扩展和维护。子类继承抽象类使用extends关键字。

抽象类不能被实例化，但有自己的构造方法，也可以有实例方法并实现业务逻辑。

**public** **abstract** **class** Product{

//所有产品类的公共业务方法

**public** **void** methodSame() {

//公共方法的实现

}

//声明抽象业务方法,子类中要对该方法进行实现

**public** **abstract** **void** methodDiff();

}

（3）ConcreteProduct（具体产品角色）：它是简单工厂模式的创建目标。每一个具体产品角色都继承了抽象产品角色，要实现在抽象产品中声明的抽象方法。典型代码如下：

**public** **class** ConcreteProduct **extends** Product{

//实现业务方法

**public** **void** methodDiff() {

//业务方法的实现

}

}

（4）客户端代码

**public** **class** Client{

**public** **static** **void** main(String args[]) {

Product product;

//getProduct()为工厂类中的静态方法，直接用类名引用

product=Factory.*getProduct*("A");//通过工厂类创建产品对象

product.methodSame();

product.methodDiff();

}

}

**三、简单工厂模式优缺点**

**优点：**

（1）实现了对象创建和使用的分离

（2）客户端无须知道所创建的具体产品类的类名，只需要知道具体产品类所对应的参数即可

（3）通过引入配置文件，可以在不修改任何客户端代码的情况下更换和增加新的具体产品类，在一定程度上提高了系统的灵活性

**缺点：**

（1）工厂类集中了所有产品的创建逻辑，职责过重，一旦不能正常工作，整个系统都要受到影响

（2）增加系统中类的个数（引入了新的工厂类），增加了系统的复杂度和理解难度

（3）系统扩展困难，一旦添加新产品不得不修改工厂逻辑

（4）由于使用了静态工厂方法，造成工厂角色无法形成基于继承的等级结构，工厂类不能得到很好地扩展

**四、简单工厂模式应用实例**

**1.实例说明**

某软件公司要基于Java语言开发一套图表库，该图表库可以为应用系统提供多种不同外观的图表，例如柱状图(HistogramChart)、饼状图(PieChart)、折线图(LineChart)等。该软件公司图表库设计人员希望为应用系统开发人员提供一套灵活易用的图表库，通过设置不同的参数即可得到不同类型的图表，而且可以较为方便地对图表库进行扩展，以便能够在将来增加一些新类型的图表。

现使用简单工厂模式来设计该图表库。

**2.实例类图**



**3.实例代码**

（1）Chart:抽象图表接口，充当抽象产品类

//接口

● 接口 interface

在写程序时，有时必须让一个子类继承多个类中所有的属性和方法，但Java并不支持多继承（即一个子类只能继承一个父类），此时接口就能实现多继承的效果。

接口是一种特殊的抽象类，其中**只包含常量和方法的定义，没有变量和方法的实现。**接口中的方法全部都是抽象方法（这也是接口和抽象类的一个不同点）。

实现一个接口用**implements**关键字，用来实现接口中定义的抽象方法。**实现一个接口，必须实现接口中的所有方法。**

**接口可以被多重实现，抽象类只能被单一继承。**

**public** **interface** Chart{

**public** **void** display();

}

（2）HistogramChart：柱状图类，充当具体产品类。

//子类继承接口要用implements关键字

**public** **class** HistogramChart **implements** Chart{

**public** HistogramChart() {

System.***out***.println("创建柱状图！");

}

**public** **void** display() {

System.***out***.println("显示柱状图！");

}

}

（3）PieChart：饼状图类，充当具体产品类。

**public** **class** PieChart **implements** Chart{

**public** PieChart() {

System.***out***.println("创建饼状图！");

}

**public** **void** display() {

System.***out***.println("显示饼状图！");

}

}

（4）LineChart：折线图类，充当具体产品类。

**public** **class** LineChart **implements** Chart{

**public** LineChart() {

System.***out***.println("创建折线图！");

}

**public** **void** display() {

System.***out***.println("显示折线图！");

}

}

（5）ChartFactory：图表工厂类，充当工厂类。

**public** **class** ChartFactory{

//静态工厂方法

**public** **static** Chart getChart(String type) {

•Java中创建对象的方法：

类名 对象名 = new 类名()

例：

Chart chart = **new** PieChart();

Chart chart = **null**;

**if**(type.equalsIgnoreCase("histogram")) {

chart = **new** HistogramChart();

System.***out***.println("初始化设置柱状图！");

}

**else** **if**(type.equalsIgnoreCase("pie")) {

chart = **new** PieChart();

System.***out***.println("初始化设置饼状图！");

}

**else** **if**(type.equalsIgnoreCase("Line")) {

chart = **new** LineChart();

System.***out***.println("初始化设置折线图！");

}

**return** chart;

}

}

（6）Client：客户端测试类。

**public** **class** Client{

**public** **static** **void** main(String arg[]) {

Chart chart;

chart = ChartFactory.getChart("histogram");//静态工厂方法直接类名调用，返回HistogramChart类的对象实例

chart.display();

}

}

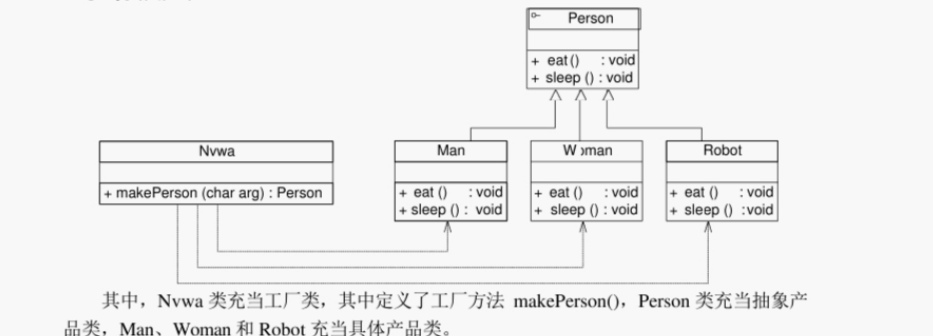
**五、课后习题答案**

1. C

2. C

3. A

4. 参考类图如下



5. 参考类图如下

