# **简单工厂模式（Java）**

·创建型模式 对类的实例化进程进行了抽象，能够将软件模块中对象的创建和对象的使用分离。

·创建型模式在创建什么(What)，由谁创建(Who)，何时创建(When)等方面都为软件设计者提供了尽可能大的灵活性。创建型模式隐藏了类的实例的创建细节，通过隐藏对象如何被创建和组合在一起达到使整个系统独立的目的。

## **简单工厂模式**

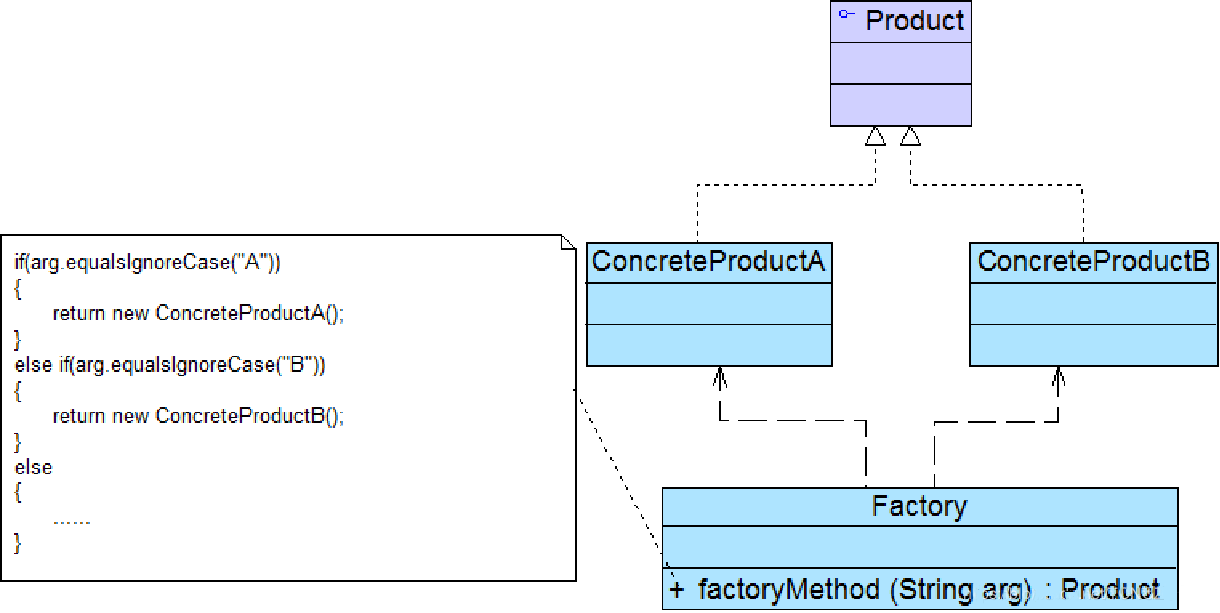
**·模式动机**

**考虑一个简单的软件应用场景，一个软件系统可以提供多个外观不同的按钮（如圆形按钮、矩形按钮、菱形按钮等），这些按钮都源自同一个基类，不过在继承基类后不同的子类修改了部分属性从而使得它们可以呈现不同的外观，如果我们希望在使用这些按钮时，\*不需要知道这些具体按钮类的名字，只需要知道表示该按钮类的一个参数，并提供一个调用方便的方法，把该参数传入方法即可返回一个相应的按钮对象，\*此时，就可以使用简单工厂模式。**

**1 、模式定义**

**简单工厂模式(Simple Factory Pattern)：又称为静态工厂方法(Static Factory Method)模式，它属于类创建型模式。在简单工厂模式中，可以根据参数的不同返回不同类的实例。简单工厂模式专门定义一个类来负责创建其他类的实例，被创建的实例通常都具有共同的父类。**

**2、模式结构**



##### **模式结构**

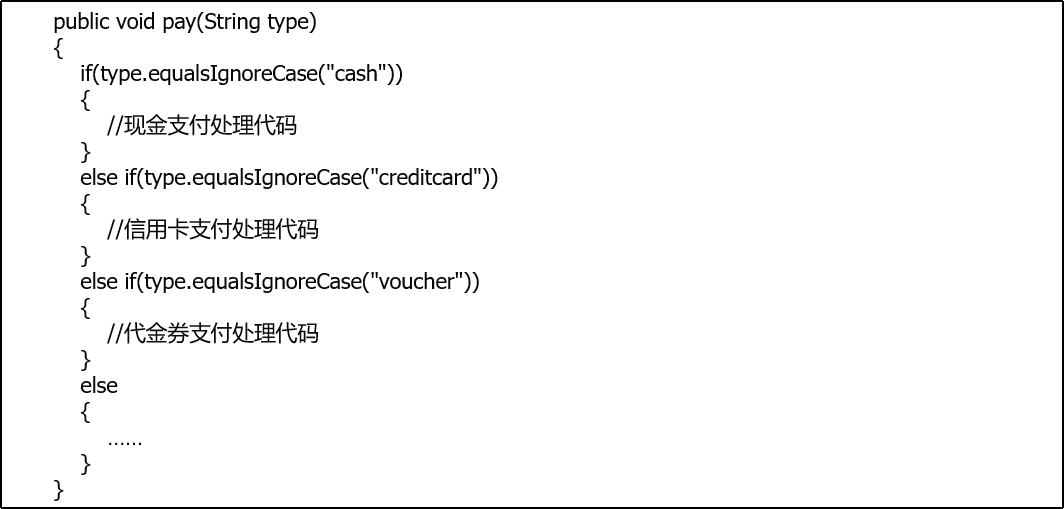
1 Factory：工厂角色

2 Product：抽象产品角色

3 ConcreteProduct：具体产品角色

##### **模式分析**

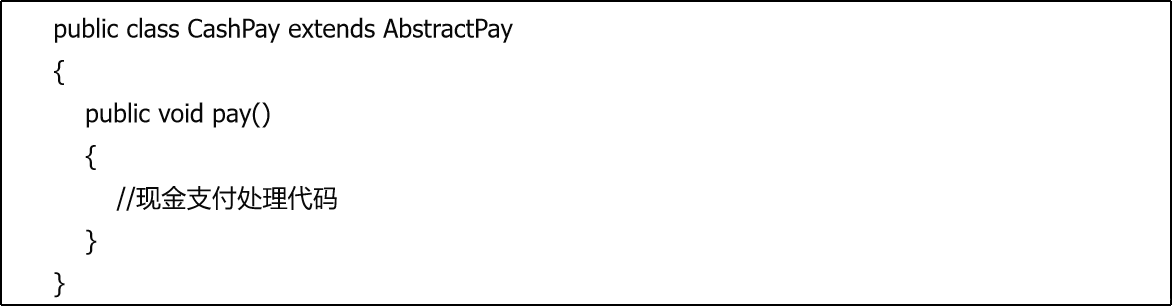
##### **·**代码复杂，难以维护



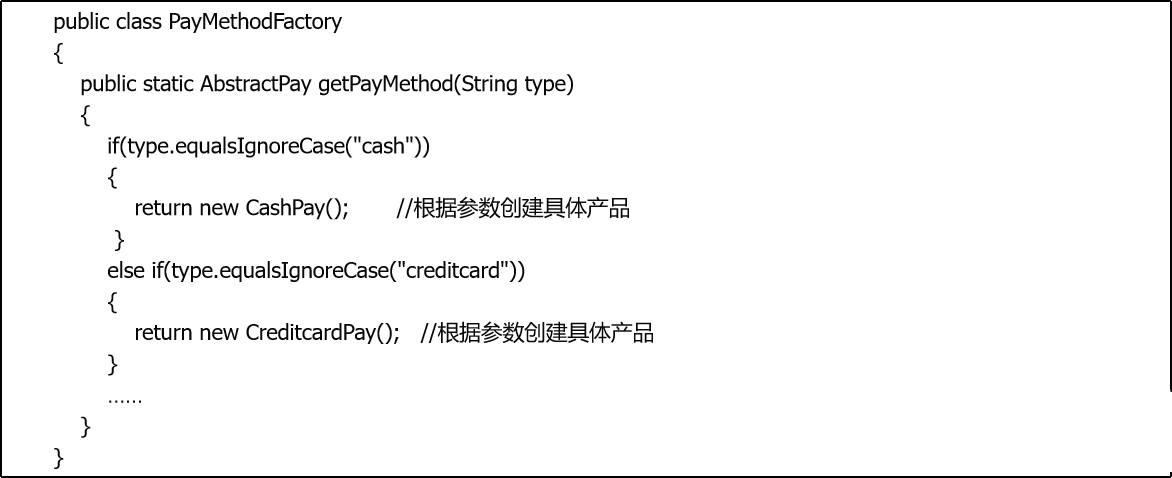
·重构后的代码

（ 抽象支付类）  


（具体支付类）



（支付工厂）



5、模式分析

·将对象的创建和对象本身业务处理分离可以降低系统的耦合度，使得两者修改起来都相对容易。

·在调用工厂类的工厂方法时，由于工厂方法是静态方法，使用起来很方便，可通过类名直接调用，而且只需要传入一个简单的参数即可，在实际开发中，还可以在调用时将所传入的参数保存在XML等格式的配置文件中，修改参数时无须修改任何Java源代码。

·简单工厂模式的要点在于：当你需要什么，只需要传入一个正确的参数，就可以获取你所需要的对象，而无须知道其创建细节。

·简单工厂模式最大的问题在于工厂类的职责相对过重，增加新的产品需要修改工厂类的判断逻辑，这一点与开闭原则是相违背的。

##### **6、模式实例与解析**

1、简单电视机工程

//抽象产品角色

package 简单工厂模式;

public interface TV {

//public String Brand(String brand);

}

// 具体产品角色

package 简单工厂模式;

public class HisenseTV implements TV {

@Override

public String Brand(String brand) {

return brand="hisense";

}

}

package 简单工厂模式;

public class HaierTV implements TV {

@Override

public String Brand(String brand) {

return brand="haier";

}

}

// 工厂角色

package 简单工厂模式;

public class 简单工厂模式 {

//简单电视机工厂

public static class TVFactory{

public static TV produceTV(String brand) throws Exception{

if(brand.equalsIgnoreCase("haier")){

System.out.println("电视机工厂生产海尔电视机！");

return new HaierTV();

}else if(brand.equalsIgnoreCase("hisense")){

System.out.println("电视机工厂生产海信电视机！");

return new HisenseTV();

}else{

throw new Exception("对不起，暂时不能生产该品牌电视机！");

}

}

public static void main(String[] args) throws Exception {

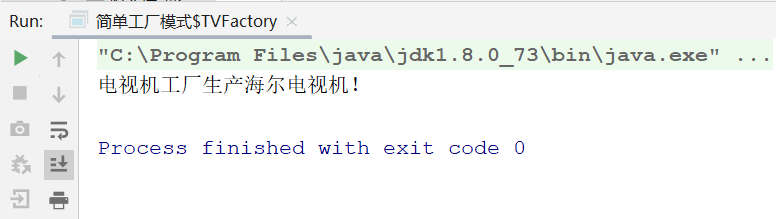
produceTV("haier");

}

}

}

·运行结果



1. 权限管理模式

package 简单工厂模式2;

//抽象产品

public interface User {

public default void sameOperaton(){

System.out.println("修改个人资料。");

}

public void diffOperation();

}

package 简单工厂模式2;

//具体产品

public class Manager implements User {

@Override

public void diffOperation() {

System.out.println("经理拥有创建和审批假条的权限");

}

public Manager(){

System.out.println("创建经理对象！");

diffOperation();

}

}

package 简单工厂模式2;

//具体cp

public class Employee implements User {

public Employee(){

System.out.println("创建员工对象！");

diffOperation();

}

@Override

public void diffOperation() {

System.out.println("员工拥有创建和审批假条的权限");

}

}

package 简单工厂模式2;

//具体产品

public class Administroter implements User {

public Administroter(){

System.out.println("创建总管对象");

diffOperation();

}

@Override

public void diffOperation() {

System.out.println("总管拥有创建和审批假条的权限");

}

}

package 简单工厂模式2;

// 工厂角色

public class 权限管理模式 {

public static User getUser(int permission){

if(0==permission){

return new Employee();

}else if(1==permission){

return new Manager();

}else if(2==permission){

return new Administroter();

}else{

return null;

}

}

public static void main(String[] args) {

getUser(1);

getUser(2);

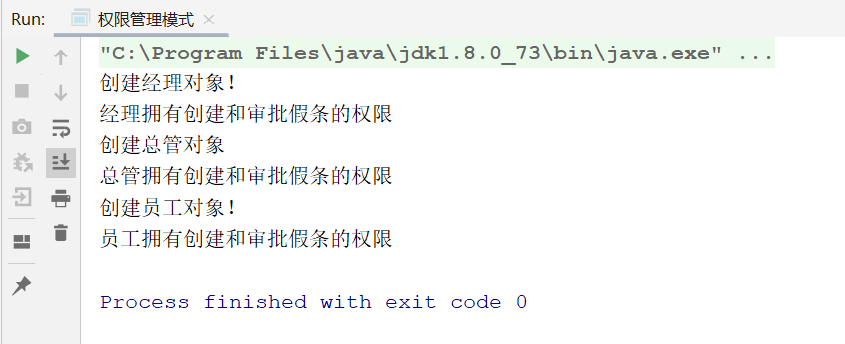
getUser(0);

//getUser(3);

}

}

·结果展示



7、简单工厂模式的优缺点

·优点

1、工厂类含有必要的判断逻辑，可以决定在什么时候创建哪一个产品类的实例，客 户端可以免除直接创建产品对象的责任，而仅仅“消费”产品；简单工厂模式通过这种做法实现了对责任的分割，它提供了专门的工厂类用于创建对象。

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，对于一些复杂的类名，通过简单工厂模式可以减少使用者的记忆量。

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、 由于工厂类集中了所有产品创建逻辑，一旦不能正常工作，整个系统都要受到影响。

2、使用简单工厂模式将会增加系统中类的个数，在一定程序上增加了系统的复杂度和理解难度。

3、系统扩展困难，一旦添加新产品就不得不修改工厂逻辑，在产品类型较多时，有可能造成工厂逻辑过于复杂，不利于系统的扩展和维护。

4、简单工厂模式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。

8、使用环境

在以下情况下可以使用简单工厂模式：

·工厂类负责创建的对象比较少：由于创建的对象较少，不会造成工厂方法中的业务逻辑太过复杂。

·客户端只知道传入工厂类的参数，对于如何创建对象不关心：客户端既不需要关心创建细节，甚至连类名都不需要记住，只需要知道类型所对应的参数。

##### **9、模式应用**

* ·Java加密技术