简单工厂模式实验

**【实验目的及要求】**

1. 熟悉简单工厂模式的定义；
2. 掌握简单工厂模式角色及作用，熟悉简单工厂模式结构图；
3. 掌握简单工厂模式编程。

**【实验内容】**

**假定一个淡水鱼养殖场可以生产数种类型的淡水鱼：草鱼、鲤鱼、鲶鱼等，只要告诉他淡水鱼的类型，它就会给你生产出来。使用简单工厂模式实现淡水鱼的生产。**

**【实验步骤】**

1. **淡水鱼生产的类图**
2. **实现代码**
3. **抽象产品----FreshWaterfish**

**package** fish;

//抽象产品

**public** **interface** FreshwaterFish {

**public** **void** grow(); //鱼在生长

**public** **void** harvest(); //收获鱼

}

1. **具体产品类**

**草鱼类（GrassCarp）**

**package** fish;

//草鱼类

**public** **class** GrassCarp **implements** FreshwaterFish {

@Override

**public** **void** grow() {

System.***out***.println("草鱼在生长！");

}

@Override

**public** **void** harvest() {

System.***out***.println("收获草鱼！");

}

}

**鲤鱼类（Carp）**

**package** fish;

//鲤鱼类

**public** **class** Carp **implements** FreshwaterFish {

**@Override  
public void grow() {  
 System.*out*.println("鲤鱼在生长");  
}  
  
@Override  
public void harvest() {  
 System.*out*.println("收获鲤鱼");  
  
}**

**}**

**鲶鱼类（Catfish）**

**package** fish;

//鲶鱼类

**public** **class** Catfish **implements** FreshwaterFish {

**@Override  
public void grow() {  
 System.*out*.println("鲶鱼在生长");  
}  
  
@Override  
public void harvest() {  
 System.*out*.println("收获鲶鱼");  
  
}**

**}**

1. **简单工厂类（FishFarm）**

**package** fish;

//简单工厂类

**public** **class** FishFarm {

**public static** FreshwaterFish productFish(String type) {

**if**("草鱼".equals(type)) {

**return** **new** GrassCarp(); //实例化草鱼对象

}

**else** **if**("鲤鱼".equals(type)) {

**return new Carp();**

//实例化鲤鱼对象

}

**else** **if**("鲶鱼".equals(type)) {

**return new Catfish();**

//实例化鲶鱼对象

}

**else** {

System.***out***.println("抱歉，不生产这种鱼");

**return** **null**;

}

}

}

1. **测试类Waiter**

**package** fish;

**public** **class** SimpleFactoryTest {

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

//生产鲤鱼

FreshwaterFish carp = FishFarm.*productFish* ("鲤鱼");

**if**(carp != **null**) {

carp.grow();

carp.harvest();

}

//生产草鱼

FreshWaterFish grashCarp =FishFarm.*productFish*(**"草鱼"**);  
**if**(grashCarp!=**null**){  
 grashCarp.grow();  
 grashCarp.harvest();  
}  
grashCarp.grow();

//生产鲶鱼

FreshWaterFish catFish=FishFarm.*productFish*(**"鲶鱼"**);  
**if**(catFish!=**null**){  
 catFish.grow();  
 catFish.harvest();  
}  
catFish.grow();

}

}

**【思考问题】**

如果增加一个鲫鱼类，程序该如何修改

1. 新增一个鲫鱼类继承FreshWaterFish

//鲫鱼  
**public class** Crucian **implements** FreshWaterFish{  
  
 @Override  
 **public void** grow() {  
 System.***out***.println(**"鲫鱼在生长"**);  
 }  
  
 @Override  
 **public void** harvest() {  
 System.***out***.println(**"收获鲫鱼"**);  
 }  
}

2.然后在工厂类中实例化这个对象

**public class** FishFarm {  
 **public static** FreshWaterFish productFish(String type){  
 **if**(**"草鱼"**.equals(type)){  
 **return new** GrassCarp();  
 } **else if** (**"鲤鱼"**.equals(type)) {  
 **return new** Carp();  
 } **else if** (**"鲶鱼"**.equals(type)) {  
 **return new** Catfish();  
 } **else if** (**"鲫鱼"**.equals(type)) {  
 **return new** Crucian();  
 }**else** {  
 System.***out***.println(**"不生产这种鱼"**);  
 **return null**;  
 }  
 }  
}

3.最后在工厂类中多态实例化鲫鱼，实现鲫鱼的方法

**public class** SimpleFactoryTest {  
 **public static void** main(String[] args) {  
 FreshWaterFish carp=FishFarm.*productFish*(**"鲤鱼"**);  
 **if**(carp!=**null**){  
 carp.grow();  
 carp.harvest();  
 }  
 carp.grow();  
  
 FreshWaterFish grashCarp =FishFarm.*productFish*(**"草鱼"**);  
 **if**(grashCarp!=**null**){  
 grashCarp.grow();  
 grashCarp.harvest();  
 }  
 grashCarp.grow();  
  
 FreshWaterFish catFish=FishFarm.*productFish*(**"鲶鱼"**);  
 **if**(catFish!=**null**){  
 catFish.grow();  
 catFish.harvest();  
 }  
 catFish.grow();  
  
// 鲫鱼  
 FreshWaterFish crucian = FishFarm.*productFish*(**"鲫鱼"**);  
 **if**(crucian!=**null**){  
 crucian.grow();  
 crucian.harvest();  
 }  
 crucian.grow();  
  
 }  
}

**【总结】**

|  |  |
| --- | --- |
| 实验总结  （不少于100字） | ### 实验总结：  在本次实验中，我们熟悉并掌握了简单工厂模式的定义及其实现。简单工厂模式是一种创建型设计模式，它通过一个工厂类来根据参数生产不同类型的对象。在淡水鱼养殖场的模拟中，工厂类根据传入的鱼类名称创建对应的鱼对象，如草鱼、鲤鱼和鲶鱼等。  实验中，我们通过编写工厂类来封装对象的创建逻辑，简化了客户端代码的复杂度，使得客户端只需通过传递参数来获取不同类型的淡水鱼对象，而无需直接了解各类鱼的具体创建过程。该模式增强了代码的可扩展性，当有新的鱼类加入时，只需要修改工厂类即可，无需更改客户端代码。  通过此次实验，我们不仅掌握了简单工厂模式的编程实践，还了解了它在解耦对象创建与使用方面的重要性。同时，简单工厂模式的结构图和角色定义让我们对其工作原理有了更深入的理解。 |
| 学号 | 22154A206 |
| 姓名 | 冯艳芳 |
| 成绩 |  |