工厂方法模式实验

**【实验目的及要求】**

1. 熟悉工厂方法模式的定义；
2. 掌握工厂方法模式角色及作用，熟悉工厂方法模式结构图；
3. 掌握工厂方法模式编程。

**【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。使用工厂方法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. **抽象工厂类（FishFarm）**

**package** fish;

//简单工厂类

**public** **interface** FishFarm {

**public** FreshwaterFish productFish();

}

1. **具体工程类**

**GrassCarpFarm类**

**package** fish;

@Override  
**public** FreshWaterFish productFish() {  
 **return new** GrassCarp();  
}

**CarpFarm类**

**package** fish;

**public class** CarpFarm **implements** FishFarm{  
 @Override  
 **public** FreshWaterfish productFish() {  
 **return new** Carp();  
 }  
}

**CatfishFarm类**

**package** fish;

**public class** CatfishFarm **implements** FishFarm{  
  
 @Override  
 **public** FreshWaterfish productFish() {  
 **return** (FreshWaterfish) **new** Catfish();  
 }  
}

1. **工具类**

**XML工具类（XMLUtil）**

**public class XMLUtil {  
 public static FishFarm getBean() {  
 DocumentBuilderFactory factory=DocumentBuilderFactory.*newInstance*();  
 try {  
 DocumentBuilder builder = factory.newDocumentBuilder();  
 Document document = builder.parse(new File("src/com/fancy/experiment3/config.xml"));  
 NodeList nodeList = document.getElementsByTagName("farm");  
// 获取第一个标签：  
 String farmName = nodeList.item(0).getTextContent();  
  
 if (farmName.equals("CarpFarm")) {  
 return new CarpFarm();  
 } else if (farmName.equals("CatfishFarm")) {  
 return new CatfishFarm();  
 } else if (farmName.equals("GrassCarpFarm")) {  
 return new GrassCarpFarm();  
 }else {  
 return null;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
}**

1. **配置文件**

**XML配置文件（config.xml）**

***<?*xml version="1.0" encoding="utf-8" *?>*<fishFramFatory>  
 <farm>CarpFarm</farm>  
 <farm>CatfishFarm</farm>  
 <farm>GrassCarpFarm</farm>  
</fishFramFatory>**

1. **测试类**

**package** fish;

**public** **class** Test {

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

FishFarm fishFarm = (FishFarm)XMLUtil.*getBean*();

FreshwaterFish fish = fishFarm.productFish();

fish.grow();

fish.harvest();

}

}

**【思考问题】**

如果增加一个鲫鱼类，程序该如何修改

**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"**);  
  
 }  
}

**public class** CrucianFarm **implements** FishFarm{  
 @Override  
 **public** FreshWaterfish productFish() {  
 **return new** Crucian();  
 }  
}

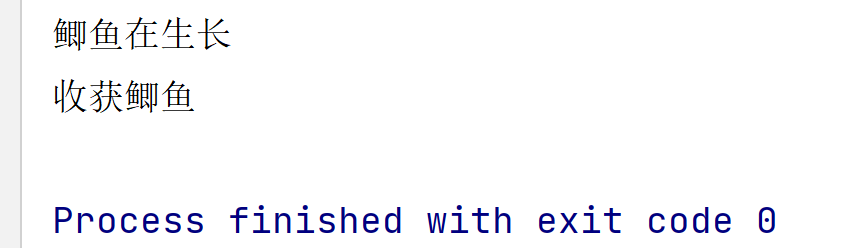
config.xml

<**fishFramFatory**>  
 <**farm**>CarpFarm</**farm**>  
 <**farm**>CatfishFarm</**farm**>  
 <**farm**>GrassCarpFarm</**farm**>  
 <**farm**>CrucianFarm</**farm**>  
</**fishFramFatory**>

在XMLUtil修改：

// 获取第四个标签：  
 String farmName = nodeList.item(3).getTextContent();

输出结果：



**【总结】**

|  |  |
| --- | --- |
| 实验总结  （不少于100字） | 在本次实验中，我们深入学习了工厂方法模式，这是一种常用的创建型设计模式。首先，我们熟悉了工厂方法模式的定义：它定义了一个创建对象的接口，但由子类决定要实例化的具体类。通过学习，我们掌握了工厂方法模式的角色及其作用，包括抽象工厂、具体工厂、抽象产品和具体产品等角色。此外，我们还熟悉了工厂方法模式的结构图，这有助于理解各个角色之间的关系。  在编程实践中，我们通过Java代码实现了工厂方法模式，这不仅加深了对模式的理解，也提高了我们的编程能力。通过实验，我们发现工厂方法模式能够提高系统的扩展性和灵活性，因为新增产品时只需添加新的具体工厂类，而无需修改现有代码。然而，这种模式也增加了类的数量，使得理解与维护变得更加复杂。  总体而言，本次实验使我们全面掌握了工厂方法模式的理论基础和实际应用，为我们在实际开发中灵活运用该模式打下了坚实的基础。 |
| 学号 | 22154A206 |
| 姓名 | 冯艳芳 |
| 成绩 |  |