

|  |
| --- |
| 实验报告  成绩（满分100） |
|  |

**《软件设计》实验报告**

**2019-2020学年第一学期**

姓名：刘文博

学号：201716040224

专业：软件工程

班级：1702班

2019年9月

**实验项目2．创建型模式实验（6学时）**

**1．实验内容**

使用创建型模式实现软件系统。

**2．基本要求**

（1）理解6种创建型模式。

（2）在软件项目开发过程中，使用创建型模式来设计软件。

**3．支撑的课程目标**

本实验项目可以支撑“课程目标2：能够在软件项目开发过程中,培养学生的创新精神”和“课程目标3：能够运用设计模式设计软件系统，支持复杂工程问题的求解”。

本实验使学生能分析软件系统，在软件项目的设计过程中使用创建型模式来设计软件，达到课程目标的要求。

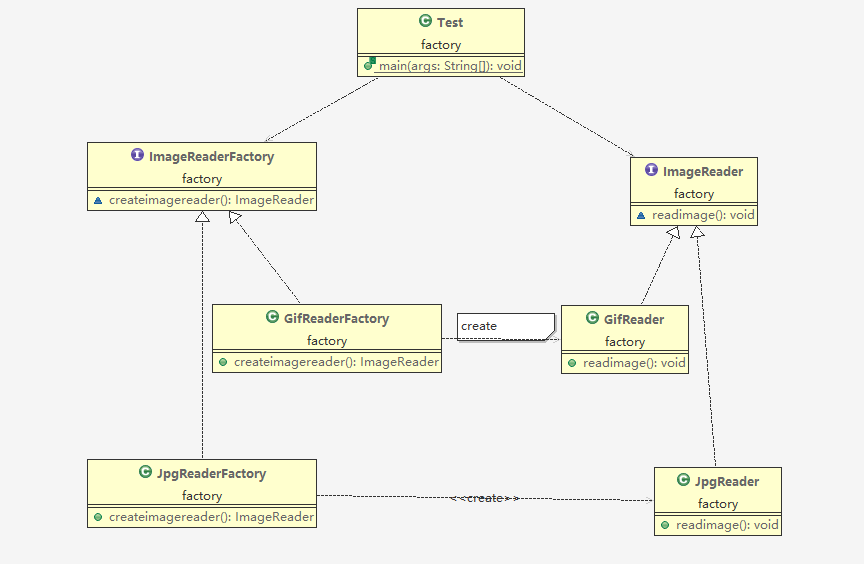
**4.实验题目**

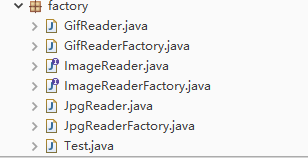
**（1）**现需要设计一个程序来读取多种不同类型的图片格式，针对每一种图片格式都设计一个图片读取器(ImageReader)，如GIF图片读取器(GifReader)用于读取GIF格式的图片、JPG图片读取器(JpgReader)用于读取JPG格式的图片。图片读取器对象通过图片读取器工厂ImageReaderFactory来创建，ImageReaderFactory是一个抽象类，用于定义创建图片读取器的工厂方法，其子类GifReaderFactory和JpgReaderFactory用于创建具体的图片读取器对象。试使用工厂方法模式设计该程序。

**（2）**使用单例模式的思想实现多例模式，确保系统中某个类的对象只能存在有限个，如两个或者三个，并编写代码实现一个多例类

**5.实验方法、实验过程及实验分析**

题目1：

做出类图如下



**抽象实体类:**

public interface ImageReader {

void readimage();

}

**实体类：**

public class GifReader implements ImageReader{

@Override

public void readimage() {

System.*out*.println("读取GIF图片！");

}

}

public class JpgReader implements ImageReader{

@Override

public void readimage() {

System.out.println("读取JPG图片！");

}

}

**抽象工厂类:**

public interface ImageReaderFactory {

ImageReader createimagereader();

}

**工厂类:**

public class JpgReaderFactory implements ImageReaderFactory{

@Override

public ImageReader createimagereader() {

return new JpgReader();

}

}

public class GifReaderFactory implements ImageReaderFactory{

@Override

public ImageReader createimagereader() {

return new GifReader();

}

}

**测试类:**

public class Test {

public static void main(String[] args) {

ImageReader reader1 = new GifReaderFactory().createimagereader();

ImageReader reader2 = new JpgReaderFactory().createimagereader();

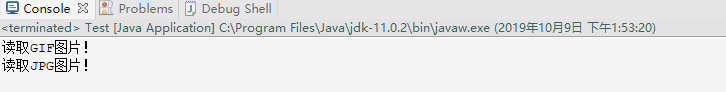
reader1.readimage();

reader2.readimage();

}

}

**运行结果：**



题目2：

**源代码：**

import java.util.ArrayList;

public class Money {

/\*\*

\* 多例类，确保系统中，某个对象的类只能存在有限个。

\*/

private static final int num = 3; //总定义实例存在的个数

private static int count = 0;

private static ArrayList<Money> m = null;

static {

m = new ArrayList<Money>(num);

}

private Money() {

}

/\*\* 普通模式，效率高\*/

public static Money getInstance() {

if (m.size() < num) {

while (m.size() < num) {

Money money = new Money();

m.add(money);

}

}

return m.get((count++) % 3);

}

/\*\* 线程安全，效率低\*/

public static Money getInstance\_safe() {

if (m.size() < num) {

synchronized (m.getClass()) {

Money money;

if (m.size() < num) {

synchronized (m.getClass()) {

while (m.size() < num) {

money = new Money();

m.add(money);

}

}

}

}

}

return m.get((count++) % 3);

}

}

**测试代码：**

public class Test {

public static void main(String[] args) {

for(int i = 0 ; i < 10 ; i++) {

Money m = Money.getInstance();

System.out.println(m);

}

for (int i = 0; i < 20; i++) {

new Thread(new Runnable() {

@Override

public void run() {

Money m = Money.getInstance\_safe();

System.out.println(m);

}

}).start();

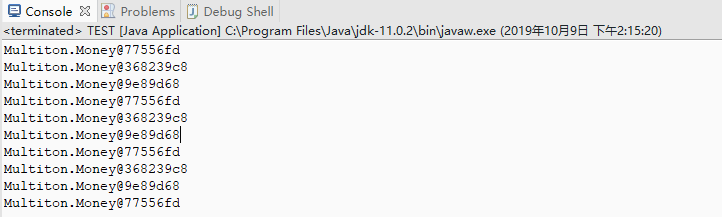
}

}

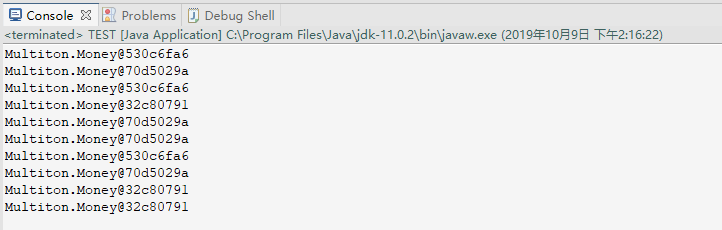
}

**测试结果：**

普通模式结果



多线程结果



**6．实验总结**

工厂模式优缺点

经过本次工厂模式和单例模式的实验，通过自己动手编代码，是自己理解工厂单例模式机制，并且知

道工厂模式有适用性在下列情况下使用工厂方法模式：1.一个类不能预期它必须创建的对象的类2.一个类希望由其子类来指定它创建的对象优点1.通过消除应用特定的类的实例化，代码变得更灵活、更可复用2.代码只处理Product类的接口，能与任何支持该接口的ConcreteProduct类一起工作缺陷1.客户端可能只因实例化一个特定的ConcreteProduct而生成Creator 类的子类。

单例模式的优点

1、实例控制

单例模式会阻止其他对象实例化其自己的单例对象的副本，从而确保所有对象都访问唯一实例。

2、灵活性

因为类控制了实例化过程，所以类可以灵活更改实例化过程。

单例模式的缺点

1、开销

虽然数量很少，但如果每次对象请求引用时都要检查是否存在类的实例，将仍然需要一些开销。可以通过使用静态初始化解决此问题。

2、可能的开发混淆

使用单例对象（尤其在类库中定义的对象）时，开发人员必须记住自己不能使用new关键字实例化对象。因为可能无法访问库源代码，因此应用程序开发人员可能会意外发现自己无法直接实例化此类。