**实验二 创建型设计模式**

学生姓名： 黄丹 学 号： 6130116093 专业班级：航天班

实验类型：□ 验证 □ 综合 □ 设计 □ 创新 实验日期： 2019/3/20 实验成绩：

一、实验项目名称

创建型设计模式

二、实验目的

1、熟悉UML统一建模语言；

2、熟悉XML文件的操作；

3、熟悉java的反射机制；

4、了解抽象类和接口的区别；

5、熟练使用多态；

6、回顾熟悉简单工厂模式、工厂方法模式、抽象工厂模式、建造者模式、原型模式、单例模式。

三、实验基本原理

四、主要仪器设备及耗材

PC微机

Windows 操作系统

IDEA/JDK1.8编程环境

github

五、实验内容

1、使用合理的模式模拟女娲(Nvwa)造人(Person)，如果传入参数M，则返回一个Man对象，如果传入参数W，则返回一个Woman对象，用Java语言模拟实现该场景。现需要增加一个新的Robot类，如果传入参数R，则返回一个Robot对象，对代码进行修改并注意“女娲”的变化。

2、假设我们要开发一个绘图程序，用来绘制简单的几何图形，这个软件应该能够处理下面的几种几何对象：圆形（Circle）、矩形（Rectangle）和正方形（Square）。除了各自特有的属性和方法之外，所有的几何图形几乎都可以抽象出绘制（draw）和擦除（erase）两个公共方法，利用简单工厂模型进行设计，画出结构图并指明类之间的关系，然后用工厂方法模式将其改进。

3、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，JPEG图片读取器（JpgReader）用于读取JPEG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。

4、某系统为了改进数据库操作的性能，自定义数据库连接对象Connection和语句对象Statement，可针对不同类型的数据库提供不同的连接对象和语句对象，如提供Oracle或MySQL专用连接类和语句类，而且用户可以通过配置文件等方式根据实际需要动态更换系统数据库。

5、有一快餐店经营良好，并逐渐发展壮大，为了适合不同地方人的饮食习惯，创建了两大系列（相当于产品族）快餐，北方系列和南方系列。1）每个系列分别由一个大厨掌勺，要求给出快餐店类的关系图及代码。2）在快餐店案例中添加一个美国系列快餐，给出类图并说明如何添加。

6、程序中经常有这样的要求，整个程序运行时只有一个实例被使用。比如：数据库连接池，系统参数配置，Java API 中的 Runtime, Calendar 等等。如何实现这种需求是一个值得讨论的问题。 以往的做法，是在程序的某个类里面（比如是GlobalObject）建立一个这个此种类的实例，然后规定所有需要用到此类的，都从GlobalObject那里获得，看下面的例子：

public class ConnectionPoolManager {

public ConnectionPoolManager { // do some initialize work }

public Connection getConnection(){ .... }

}

public class GlobalObject {

private ConnectionPoolManager connectionPoolManager;

publicGlobalObject () { connectionPoolManager = new ConnectionPoolManager(); ... }

public void getConnectionPoolManager() { return connectionPoolManager; }

}

public class QueryFunctions {

public static Collection findStudentByName(String name){

ConnectionPoolManager = globalObject.getConnectionPoolManager();

Connection connection = connectionPoolManager.getConnection();

// query database }

}

但这样做有如下缺点：其他人可能调用ConnectionPoolManager的构造函数自己建立一个数据库连接池，导致程序中存在多个ConnectionPoolManager，人为的规定只创建一个数据库连接池往往不会很好的执行。如何利用单件模式来改进设计，保证系统只建立唯一的连接池，并完善代码。

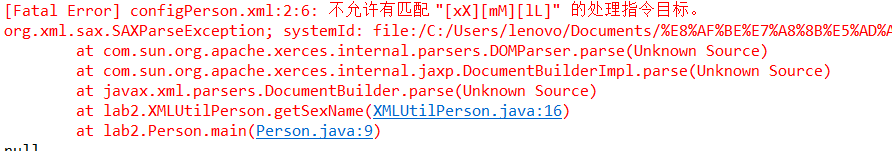
7、在游戏中，要求设计一个程序来画小人（person），要求小人有头（head）、身体（body）、两手（arm）、两脚（leg）就可以了，小人有高、矮之分，利用建造者模式进行设计，并画出其类关系图。

8、某软件公司欲开发一个音频和视频播放软件，为了给用户使用提供方便，该播放软件提供了多种界面显示模式，如完整模式、精简模式、记忆模式、网络模式等。在不同的显示模式下主界面的组成元素有所差异，如在完整模式下将显示菜单、播放列表、主窗口、控制条等，在精简模式下只显示主窗口和控制条，而在记忆模式下将显示主窗口、控制条、收藏列表等。现使用建造者模式设计该软件。

9、设计一个客户类Customer，其中客户地址存储在地址类Address中，用浅克隆和深克隆分别实现Customer对象的复制并比较这两种克隆方式的异同。

六、实验数据及处理结果

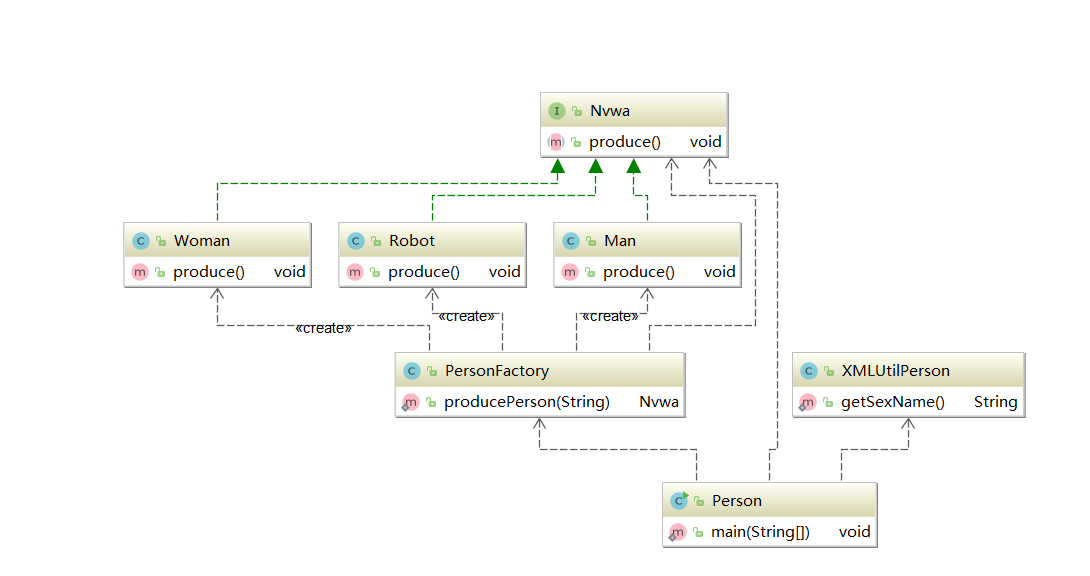
1. 女娲(Nvwa)造人(Person)



运行程序时出现配置文件错误，原因是配置文件第一行是空行，什么都没有，将其删除就可以运行成功。

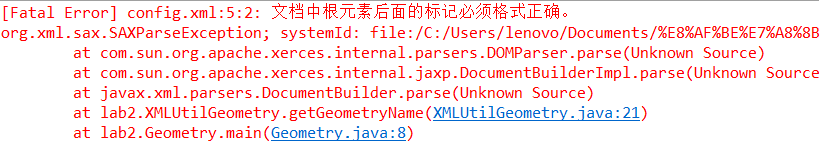






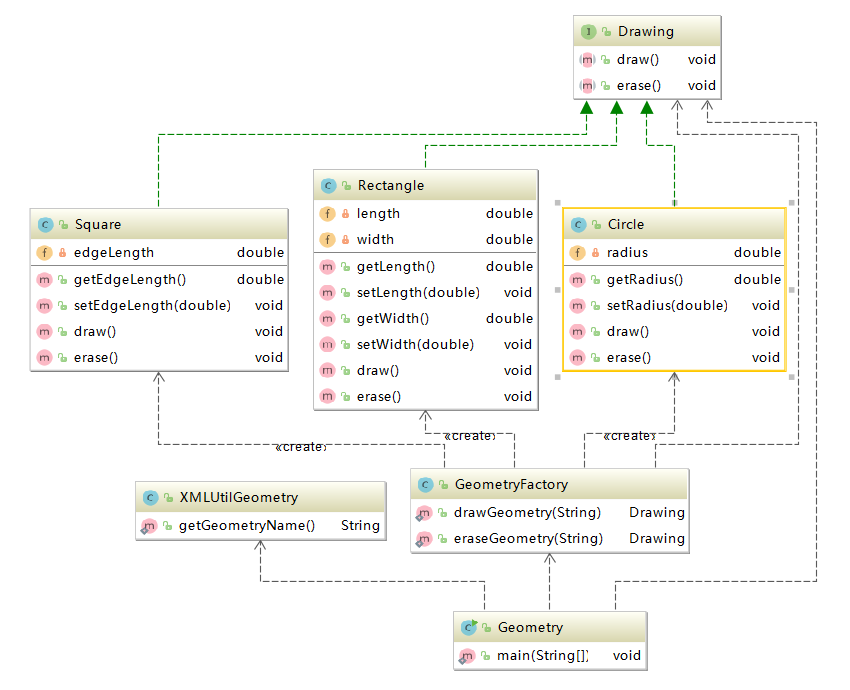
2. 绘图程序

**简单工厂模式**

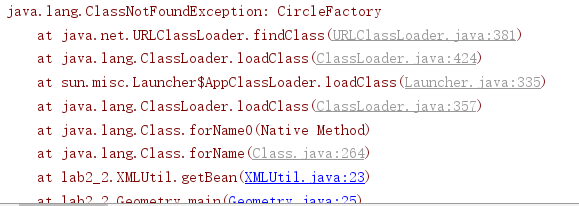


还是配置文件出错，因为我另外加了一对<config></config>，将<geometry>放入<sexName>后即可运行成功。

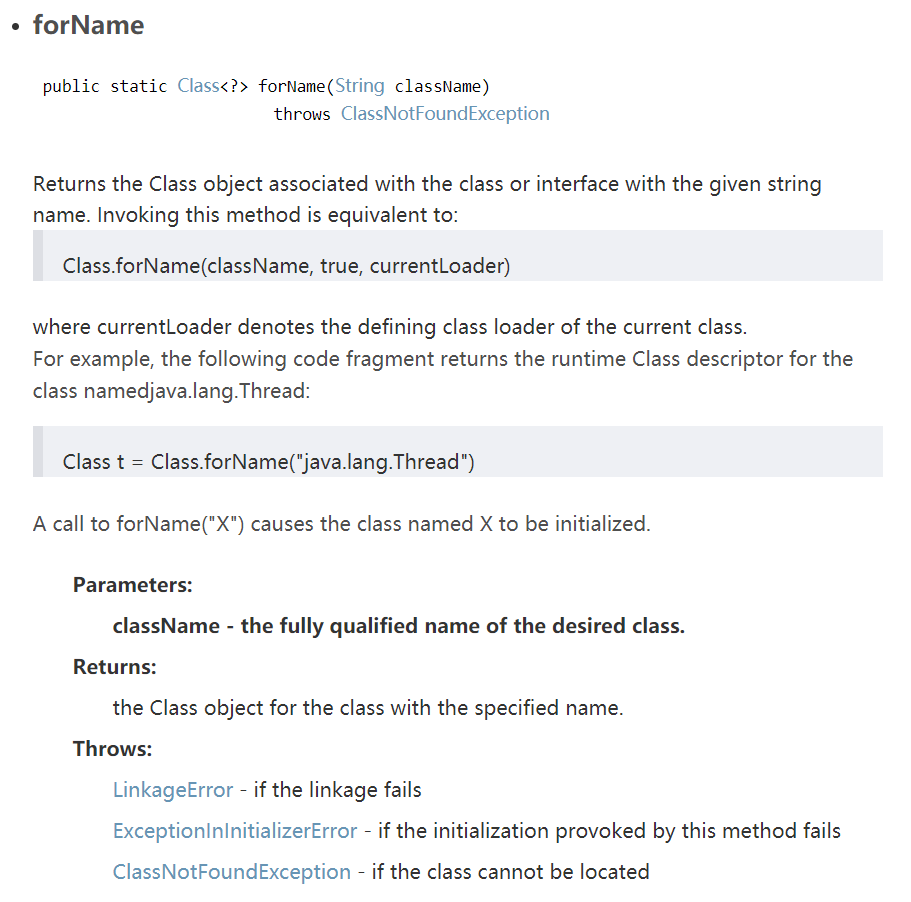
 



**工厂方法模式**



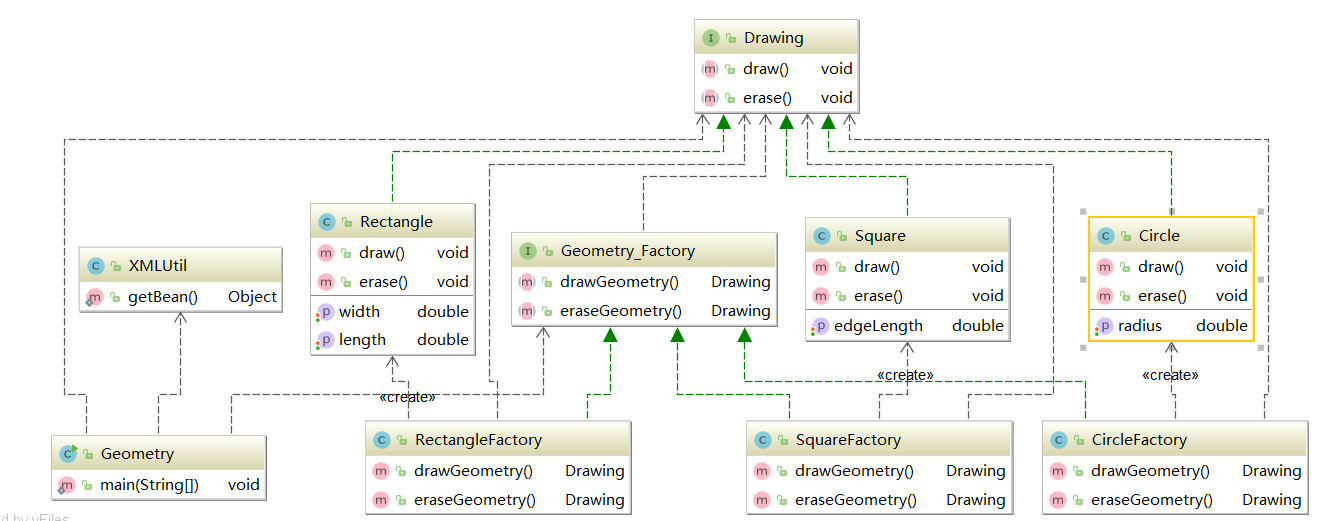
一直说找不到类，查找forName文档得知



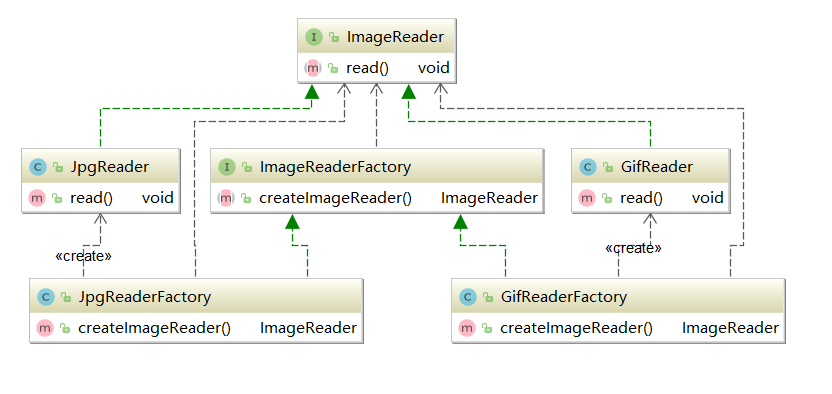
要在实例化类时加入路径，修改代码为：

*//通过类名生成实例对象并将其返回*Class c=Class.*forName*(**"lab2\_2."**+cName);  
Object obj=c.newInstance();  
**return** obj;





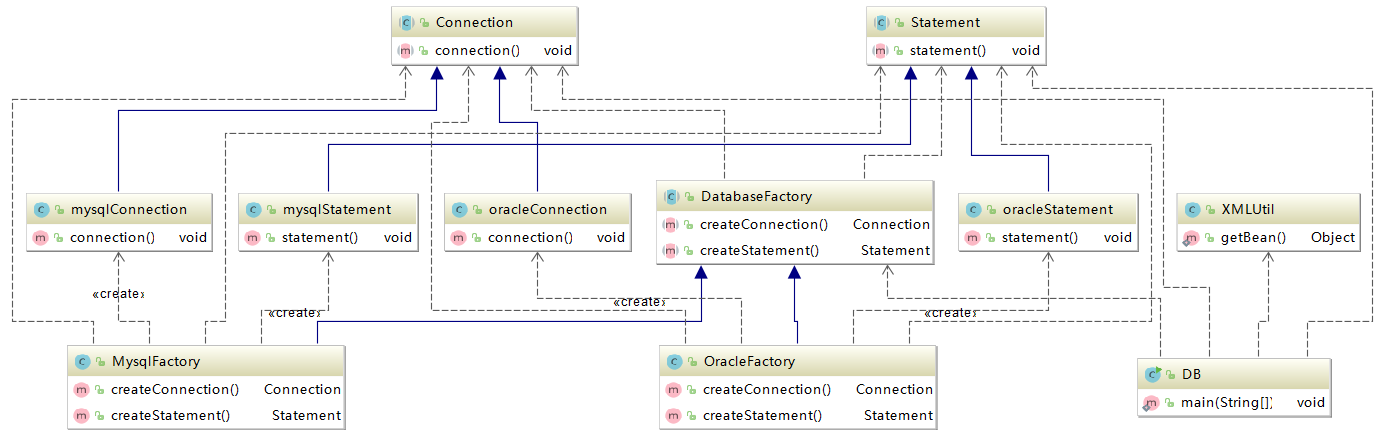
3. 读取多种不同类型的图片格式的程序



客户端测试同第2题一样

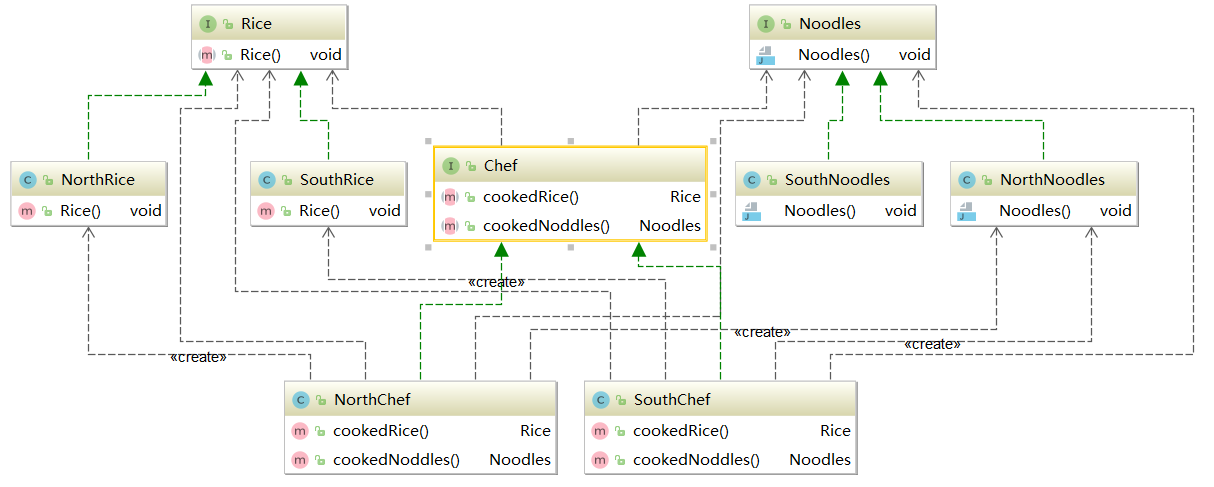
4.数据库





使用的是抽象工厂模式，提供了工厂接口和产品接口。用户可以通过配置文件等方式根据实际需要动态更换系统数据库。

5.快餐店

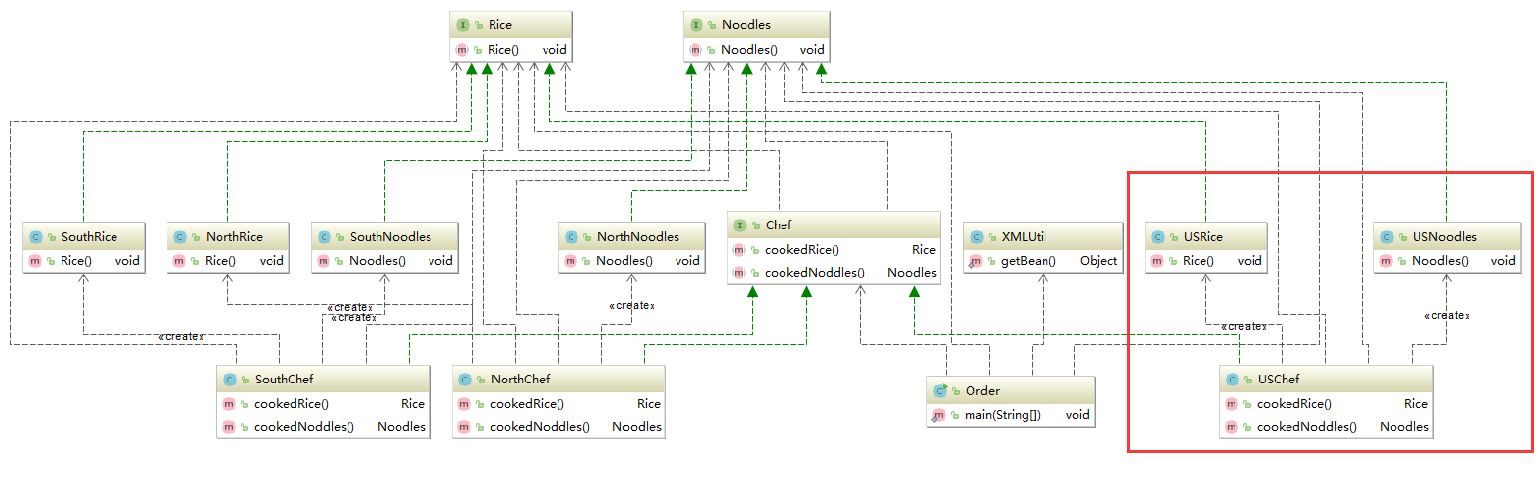




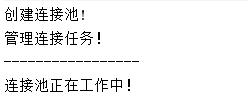
新增美国系列快餐：

不用修改之前的代码，只需增加一位厨师类，米饭类、面条类即可，然后修改配置文件。



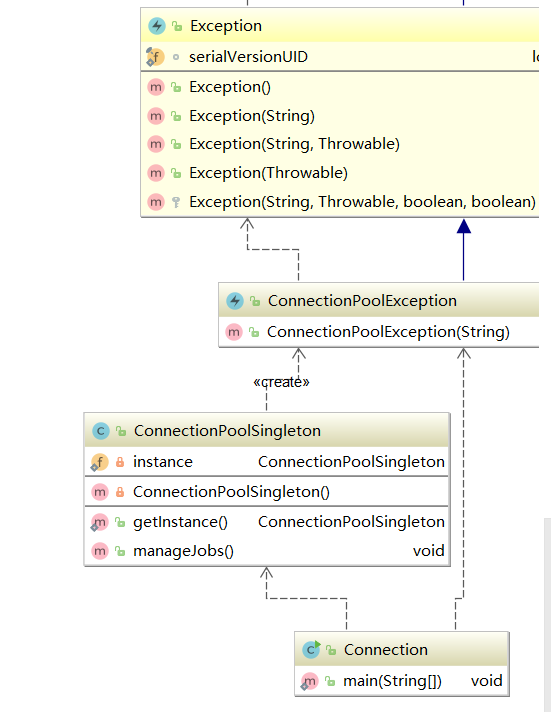


6. 保证系统只建立唯一的连接池



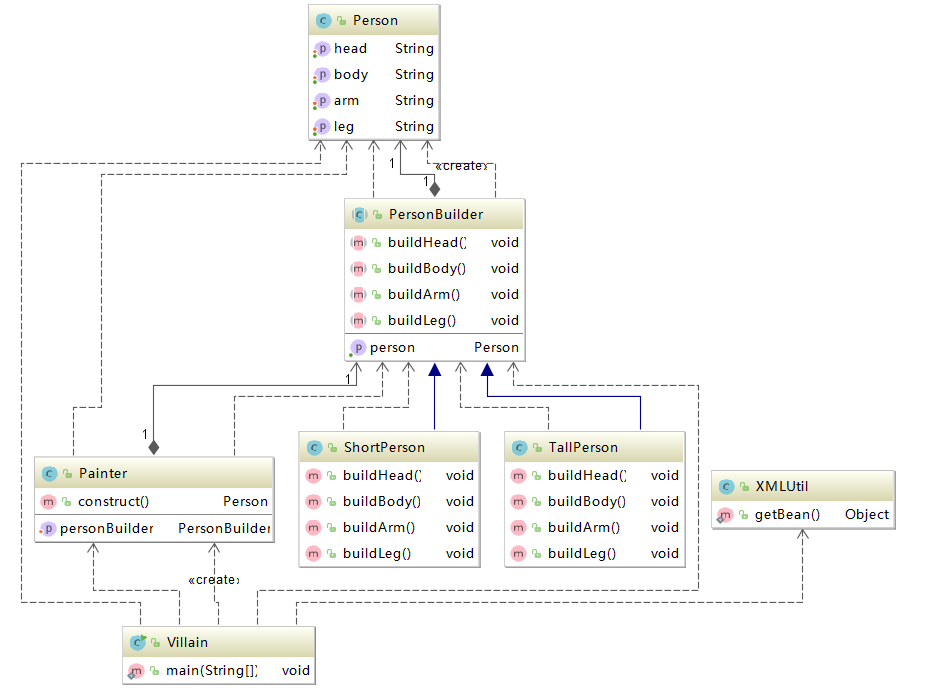
在类内部只生成一个实例，同时它提供一个静态的getInstance()工厂方法，让客户可以使用它的唯一实例；为了防止在外部对其实例化，将其构造函数设计为私有；在ConnectionPoolSingleton内部定义了一个ConnectionPoolSingleton类型的静态对象，作为外部共享的唯一实例。

**public class** ConnectionPoolSingleton {  
 **private static** ConnectionPoolSingleton *instance*=**null**;  
 **private** ConnectionPoolSingleton(){  
  
 }  
 **public static** ConnectionPoolSingleton getInstance() **throws** ConnectionPoolException{  
 **if**(*instance*==**null**){  
 System.***out***.println(**"创建连接池!"**);  
 *instance*=**new** ConnectionPoolSingleton();  
 }  
 **else**{  
 **throw new** ConnectionPoolException(**"连接池正在工作中！"**);  
 }  
 **return** *instance*;  
 }  
 **public void** manageJobs(){  
 System.***out***.println(**"管理连接任务！"**);  
 }  
}

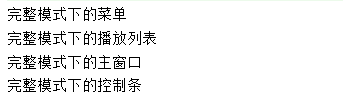


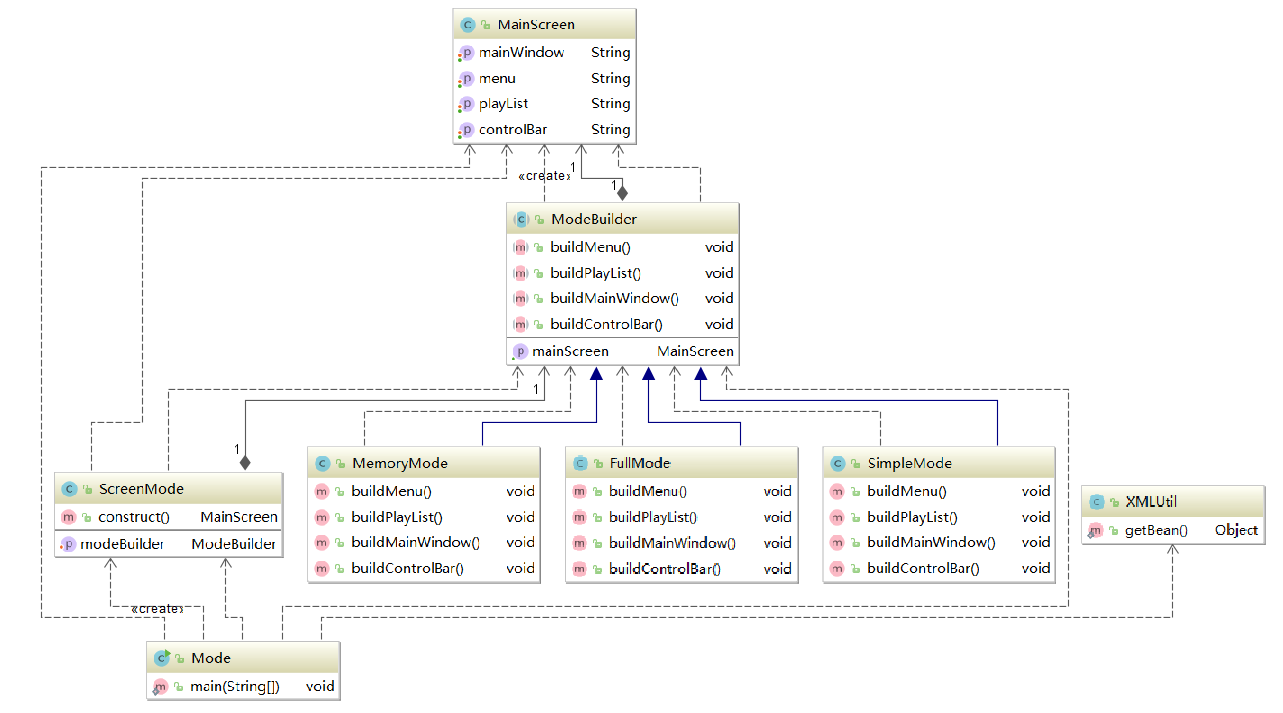
7.画小人





8.界面显示模式

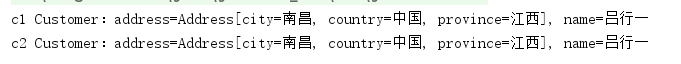
 

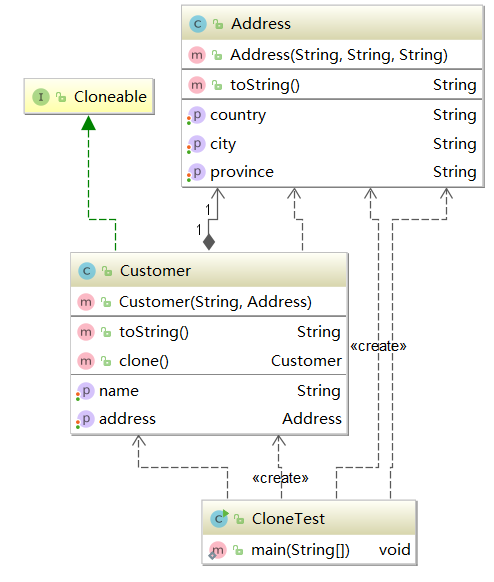


9.客户类

浅克隆





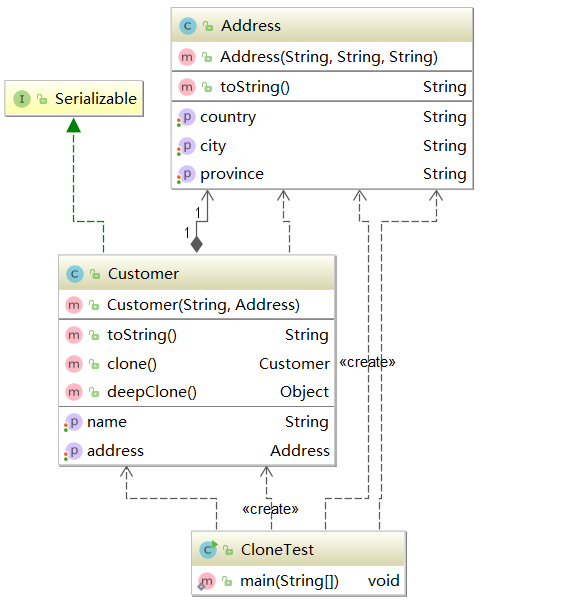


深克隆



可以看到克隆后报错了，说明通过复制得到的对象与原型对象的引用不一致了。

浅克隆仅仅复制所考虑的对象，不会复制它所引用的成员对象；在深克隆中，除了对象本身被复制外，对象包含的引用也被复制，也就是其中的成员对象也被复制。



七、思考讨论题

1、简单工厂模式解决的问题，该模式的优缺点？

答：

简单工厂模式解决的问题是：用户无须知道苹果、香蕉、橙如何创建，只需要知道水果的名字则可得到相应的水果，即根据传入的参数即可返回所需的对象，而不需要知道具体类的类名。

优点：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；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；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。

缺点：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；使用简单工厂模式将会增加系统中类的个数，在一定程度上增加了系统的复杂度和理解难度；系统扩展困难；简单工厂模式由于使用了静态工厂方法，造成了工厂角色无法形成基于继承的等级结构。

2、工厂方法模式解决的问题，该模式的优缺点？

答：

工厂方法模式解决的问题：定义一个用于创建对象的接口，让子类决定将哪一个类实例化。

优点：用户只需要关心所需产品对应的工厂，无须关心创建细节，甚至不需要知道具体产品类的类名；基于工厂角色和产品角色的多态性设计是工厂方法模式的关键；使用工厂方法模式的另一个优点是在系统中加入新产品时，无须修改抽象工厂和抽象产品的接口，无须修改客户端，也无须修改其它的具体工厂和具体产品，而只要添加一个具体工厂和具体产品即可。

缺点：添加产品时，需要添加一个具体工厂和具体产品，系统中类的个数会增加，这增加了系统的复杂度；考虑系统的可扩展性，需要引入抽象层，在客户端代码中均使用抽象层进行定义，增加了系统的抽象性和理解难度。

3、抽象工厂方法模式解决的问题，该模式的优缺点？

答：

抽象工厂方法解决的问题：提供一个创建一系列相关或相互依赖对象的接口，而无须指定它们具体的类。

优点：隔离了具体类的生成，使得客户并不需要知道什么被创建；当一个产品族中的多个对象被设计成一起工作时，它能够保证客户端始终只使用同一个产品族中的对象；增加新的具体工厂和产品族很方便。

缺点：在添加新的产品对象时，难以扩展抽象工厂来生产新种类的产品。

4、建造者模式解决的问题，该模式的优缺点？

答：

建造者模式解决的问题：将一个复杂的对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

优点：客户端不必知道产品内部组成的细节，将产品本身与产品的创建过程解耦，使得相同的创建过程可以创建不同的产品对象；每一个具体建造者都相对独立；可以更加精细地控制产品的创建过程；增加新的具体建造者无须修改原有类库的代码。

缺点：建造者模式所创建的产品一般具有较多的共同点，其组成部分相似，不适合建造组成部分相差很大的产品；如果产品的内部构造复杂，会导致系统变得庞大。

5、原型模式解决的问题，该模式的优缺点？

答：

原型模式解决的问题；用原型实例指定创建的对象的种类，并且通过复制这个原型来创建新的对象。

优点：可以简化对象的创建过程；可以动态增加或减少产品类；提供了简化的创建结构；可以使用深克隆的方式保存对象的状态。

缺点：需要为每一个类配备一个克隆方法，这样会修改其源代码，违背了开闭原则；实现深克隆时代码较为复杂。

6、单例模式解决的问题，该模式的优缺点？

答：

单例模式解决的问题：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

优点：提供了对唯一实例的受控访问；可以节约系统资源；允许可变数目的实例。

缺点；单例模式的扩展困难；单例类的职责过重，在一定程度上违背了单一职责原则；滥用单例将带来一些负面问题。

7、了解符合面向对象的设计原则。

答：单一职责原则，开闭原则，里氏代换原则，依赖倒转原则，接口隔离原则，合成复用原则，迪米特原则。

八、参考资料

九、附加信息

代码：https://github.com/inocentnine/JAVA-CreationalPatterns