

**面向对象程序设计上机实验报告**

实验题目7：类的动态创建和注入

**学院名称 智能与计算学部**

**专 业 软件工程**

**学生姓名 王杨森**

**学 号 3020244116**

**年 级 2020级**

**班 级 软件工程3班**

**时 间 2022年 4月30日**

1. **实验目的**
2. 理解配置文件的读写；
3. 理解反射、注入；
4. 练习使用DEBUG工具调试程序；
5. **实验内容**
6. 系统中有若该类等待构造（这些类的名字在程序运行的时候动态的从配置文件中读取），配置文件格式类似如下：

#系统的配置文件

#这个文件里面可能有注释，以#开头表示注释

#这个文件肯定以UTF-8编码

#每个类的配置部分由如若干行描述构成

#每个描述如下： [类的完整路径名].[属性]="一个字符串"/或者一个整数，其中属性可能是任意小写单词

#测试的时候，要考虑文件是否存在乱码的问题

#com.huawei.classroom.student.h18.Apple 是类名; color/weight 是属性名,"red"/500是属性

#如果属性以""开头结尾则表示一个字符串属性，否则肯定是一个整数的属性

com.huawei.classroom.student.h18.Apple.color="红色"

com.huawei.classroom.student.h18.Apple.weight=500

com.huawei.classroom.student.h18.Car.price=200000

com.huawei.classroom.student.h18.Car.color="黑色"

com.huawei.classroom.student.h18.Car.brand="大众"

......

1. 程序从配置文件中动态的读取每个类的名字、属性名、属性值，动态构造出对象，并通过测试用例。
2. **程序实现**

**public** **class** MyClassFactory {

**private** **final** Map<String, Map<String, Object>> params; //保存参数，键为类名，值为一个键为属性名，值为参数的Map

//构造函数

**public** MyClassFactory(String fileName) {

**this**.params = readLines(fileName);

}

//创建实例

**public** <T> T createInstance(Class<T> clazz) **throws** InvocationTargetException, NoSuchMethodException{

T t = **null**;

**try** {

String className = clazz.getName();

t = clazz.~~newInstance~~();

Map<String, Object> map = params.get(className);//获取该类的所有参数

**for**(String key: map.keySet()) {//获取属性名

Object value = map.get(key);//获取值

String methodName = "set" + key.substring(0, 1).toUpperCase() + key.substring(1);

dynamicInvokeMethod(t, methodName, value);//动态注入方法，调用setXxx

}

} **catch** (InstantiationException | IllegalAccessException e) {

e.printStackTrace();

}

**return** t;

}

//多参数动态注入方法

**public** **void** dynamicInvokeMethod(Object obj, String methodName, Object... values)

**throws** IllegalAccessException, IllegalArgumentException, InvocationTargetException, NoSuchMethodException,

SecurityException{

Class[] classes = **new** Class[values.length];

**for**(**int** i = 0; i < values.length; i++) {

classes[i] = values[i].getClass();

}

Method method = obj.getClass().getMethod(methodName, classes);

method.invoke(obj, values);

}

//读取配置文件

**public** Map<String, Map<String, Object>> readLines(String fileName) {

Map<String, Map<String, Object>> res = **new** HashMap<>();

**try** (InputStream in = **new** FileInputStream(fileName);

//不然会导致乱码，必须设置为UTF-8

InputStreamReader reader = **new** InputStreamReader(in, "UTF-8");

LineNumberReader lineReader = **new** LineNumberReader(reader)){

String line = "";

line = lineReader.readLine();

**while**(line != **null**) {

**if**(line.trim().length() != 0 && line.trim().charAt(0) != '#') {//不读取注释或空行

String[] param = line.split("=");//进行分割

String name = param[0];

String attribute = param[1];

**int** idx = name.lastIndexOf(".");

String className = name.substring(0, idx);

String paramName = name.substring(idx + 1);

Map<String, Object> map = **null**;

**if**(res.containsKey(className)) {

map = res.get(className);//如当前类已存在，把这个Map拿出来往里面赋值

} **else** {

map = **new** HashMap<>();//出现了一个新的类，给这个类存储参数

}

**if**(attribute.charAt(0) == '\"') {//是字符串

String value = attribute.substring(1, attribute.length() - 1);

map.put(paramName, value);

} **else** {//是int

Integer value = Integer.*parseInt*(attribute);

map.put(paramName, value);

}

**if**(!res.containsKey(className)) {

res.put(className, map);//放进去一个新的类的参数

}

}

line = lineReader.readLine();

}

} **catch** (IOException e) {

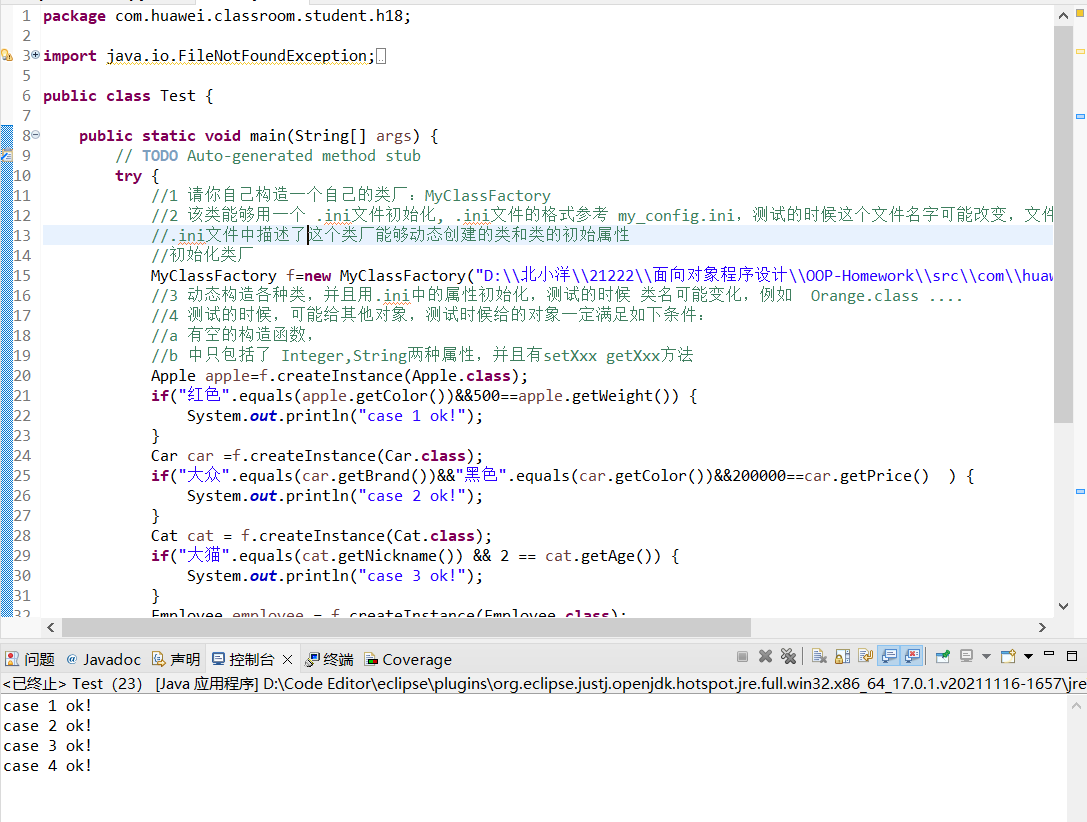
e.printStackTrace();

}

**return** res;

}

1. }**实验结果**



1. **实验中遇到的问题及解决方法**

遇到的问题是起初不知道该用什么数据结构存储配置文件的参数，后来想到可以构造一个Map，这个Map的键是类的名，值是这个类的所有参数，可以再通过一个Map实现，键是属性名，值是Object，然后读取配置文件，遇到一个新的类就新建一个Map<String, Object>，否则就取出该类名对应的Map做修改。

第二个问题是起初不知道怎么返回多个类，经过思考和查阅资料，可以使用template来解决这个问题。

另一个问题是最初在本地运行可以得到正确结果，但提交之后始终会有乱码问题，这是在读入配置文件时出现的乱码问题，应该在读文件时就设置好字符集为utf-8，否则会出现乱码问题。

通过这次实验，帮助理解了Java中的反射和注入机制，意识到了这种机制在开发框架类代码时的伟大之处。Java果然是一门非常优秀的语言。