Unit4

一、第四单元作业的架构设计

第四单元个人认为主要是考察对于层次结构的理解，即如何理解并处理好UML图的树状结构组织，在理好层次之间以及层次内部的相互关系之后，就只剩下代码实现的问题了。但是不得不说，刚开始接触UML特别是对starUML不熟悉的时候，如果指导书说的不是太清楚，真的很难动手实现，比如说在第一次作业的时候，我就完全理解错了，导致难度飞升，一度怀疑老师所说的“简单”。~~虽然说相比JML是真的不太简单~~

由于主要架构课程组都已经做好了，所以我们基本不用构思什么架构，只需要做好层次化的组织就行了

第13次作业

这次作业只要求实现类图相关的存储与查询。

在数据的存储方面，考虑到不同层次的元素之间是有很强的关系的，常常需要进行特定的操作，比如对于`class`这个元素，查询指令中有查询与class相关联的需求以及查询他所实现的接口的需求，很自然的想到被关联的class实例和被实现的interface实例最好作为属性放进此class实例中，当然还需要提供一系列的查询操作。

因为查询操作众多，显然需要使用`hashmap`进行存储，刚好`hashmap`是一对一的查询使用起来非常舒服，这里如果出现有多个键相同但是值不同的话，说明存在异常（同名），再建一个`name2Valid`的`hashmap`，对每次查询时输入的键进行检查是否有效就好了，速度也非常快。

但是课程组提供的接口给出的是课程组写的类对应的实例，此时如果采用继承来做就不太好实例化自己写的类，因此我干脆把课程组给的实例封进了自己写的类中，虽然结构上有亿点丑陋，有一点不OO，但是因为到了烤漆，也就将就着用了，最终形成以下结构：（其他的元素也大都如此，把课程组给的实例封进自己写的类中，然后就能很方便地使用

```
public class MyClass {
    private UmlClass umlClass;

    private MyClass father = null;
    private MyClass topFather = null;
    private boolean updatedTopFather = false;
    ...
    private HashSet<MyInterface> interfaces = new HashSet<>();
    private String name;
    private ArrayList<MyClass> associations = new ArrayList<>();

    public MyClass(UmlClass umlClass) {
        this.umlClass = umlClass;
        name = umlClass.getName();
    }
    ...
    public void setTopFather(MyClass topFather) {
        this.topFather = topFather;
    }

    public MyClass getTopFather() {
        return topFather;
    }

    public void setUpdatedTopFather(boolean updatedTopFather) {
        this.updatedTopFather = updatedTopFather;
    }

    public boolean hasUpdatedTopFather() {
        return updatedTopFather;
    }
```

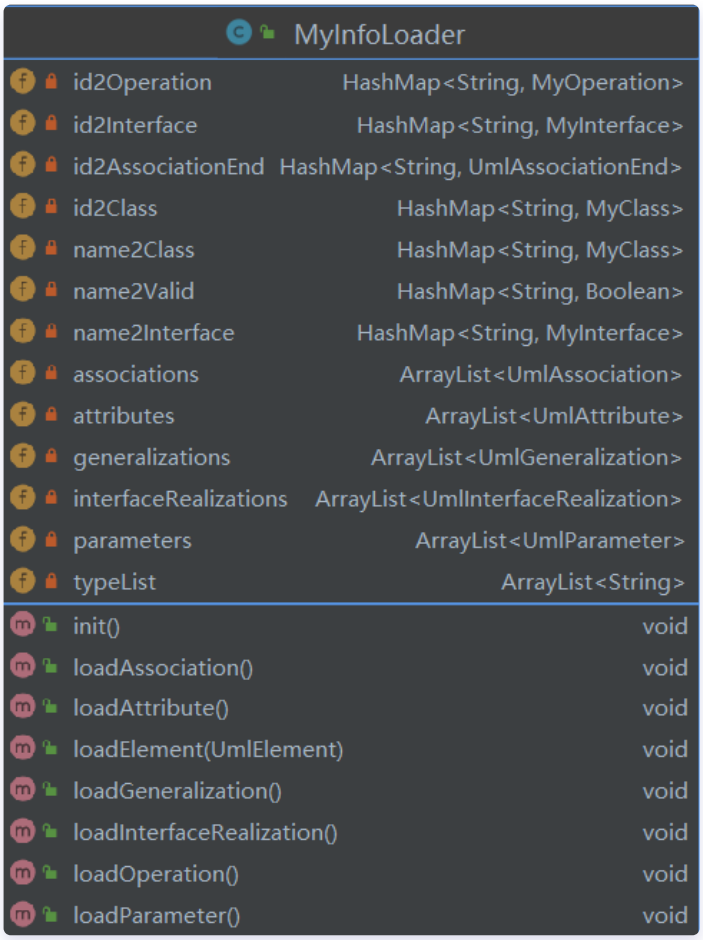
```
    ...

    public void addOperation(MyOperation operation) {
        operations.add(operation);
    }

    public HashSet<MyInterface> getInterfaces() {
        return interfaces;
    }
    ...
}
```

此外，由于第一次作业CPU使用时间限制为2s，其中有几个非常耗时的查询操作，查实现的接口，（这里还需要查接口的继承情况），查关联的类等。

考虑到实现的难度，我选择用递归来解决问题，同时采用记忆化加速。即将相关的操作提供一个`flag`，如果`flag`有效则直接返回对应的查询元素，若无效，则进行对应的递归查找，并更新相应的信息的存储。

异常处理方面，要特别注意异常抛出的顺序，大体按照从顶层向底层的顺序抛异常就好了。

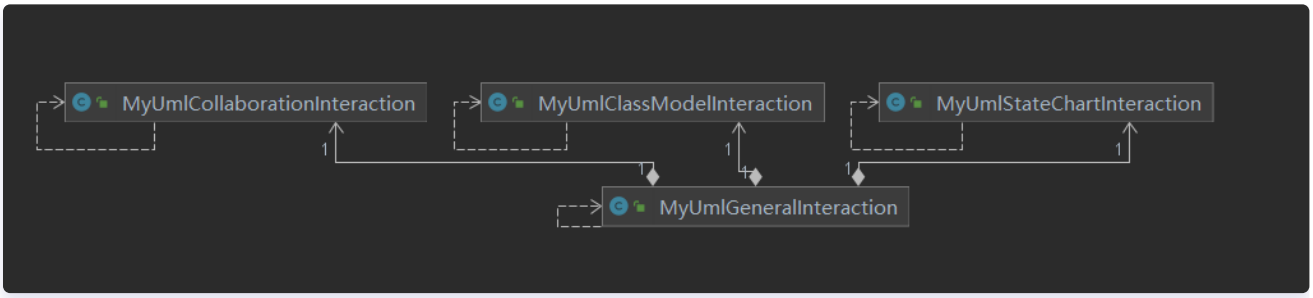
最后，考虑到输入元素的顺序可能不确定，因此我选择把所有输入的元素先分类存储，然后按照合理的顺序一个一个地存进相应的类中，为了方便管理，我写了一个`MyinfoLoader`的类来专门处理读入信息的问题。



第14次作业

第二次作业相比第一次多了时序图和状态图的处理，处理过程完全一样，有了第一次的经验，这次作业做起来显得比较简单。同时CPU使用时间放宽到**10s**，再加上多的两个UML图的查询基本上没有什么耗时的查询操作，完全不需要什么算法，直接用暴力做法硬莽就是了。

由于新增了两种UML图，我新增加了两个模块进行管理：



第15次作业

这次作业需要增加Rule的check操作，一共八条，可能是因为在烤漆，整体实现难度不太高

`Rule3` 和 `Rule4` 要求接口不能重复继承接口，类不能重复继承类，类不能重复实现接口。

由于在我的架构中，`MyinfoLoader` 在读取信息的时候需要将实现的类或者继承的类（接口）存进属性，只需要对于第一次作业信息读取存储的部分加一个判断，并提供一个 `flag`，如果存储信息的过程中出现了不合法的行为，那就直接把对应的 `flag` 置为有效，检查的时候查对应的 `flag` 就行了

比如在Rule4的检查中：

```
} else {
    updateClassInterface(fatherClass);
    HashSet<MyInterface> fatherSet = fatherClass.getInterfaces();
    if (fatherClass.getDup() == true) {
        myClass.setDup(true);
    }
    ArrayList<MyInterface> allInterface = new ArrayList<>(fatherSet);
    HashSet<MyInterface> interfaces = myClass.getInterfaces();
    for (MyInterface myInterface : interfaces) {
        updateFatherInterface(myInterface);
        ArrayList<MyInterface> fatherInterfaces = myInterface.getFatherInterfaces();
        allInterface.addAll(fatherInterfaces);
        allInterface.add(myInterface);
        if (myInterface.getDup() == true) {
            myClass.setDup(true);
        }
    }
    HashSet<MyInterface> ans = new HashSet<>(allInterface);
    if (ans.size() != allInterface.size()) {
        myClass.setDup(true);
    }
    myClass.setUpdatedInterface(true);
    myClass.updateInterface(ans);
    return;
}
```

如果没有update并且父类不为空，这时候更新他的继承的接口首先需要递归地更新他父类的接口，然后检查一下父类是否合法（如果父类标志位有效，子类置标志位）

再对父类实现的接口做一遍更新，同时如果接口标志位有效（不合法）那子类肯定不合法，置标志位。

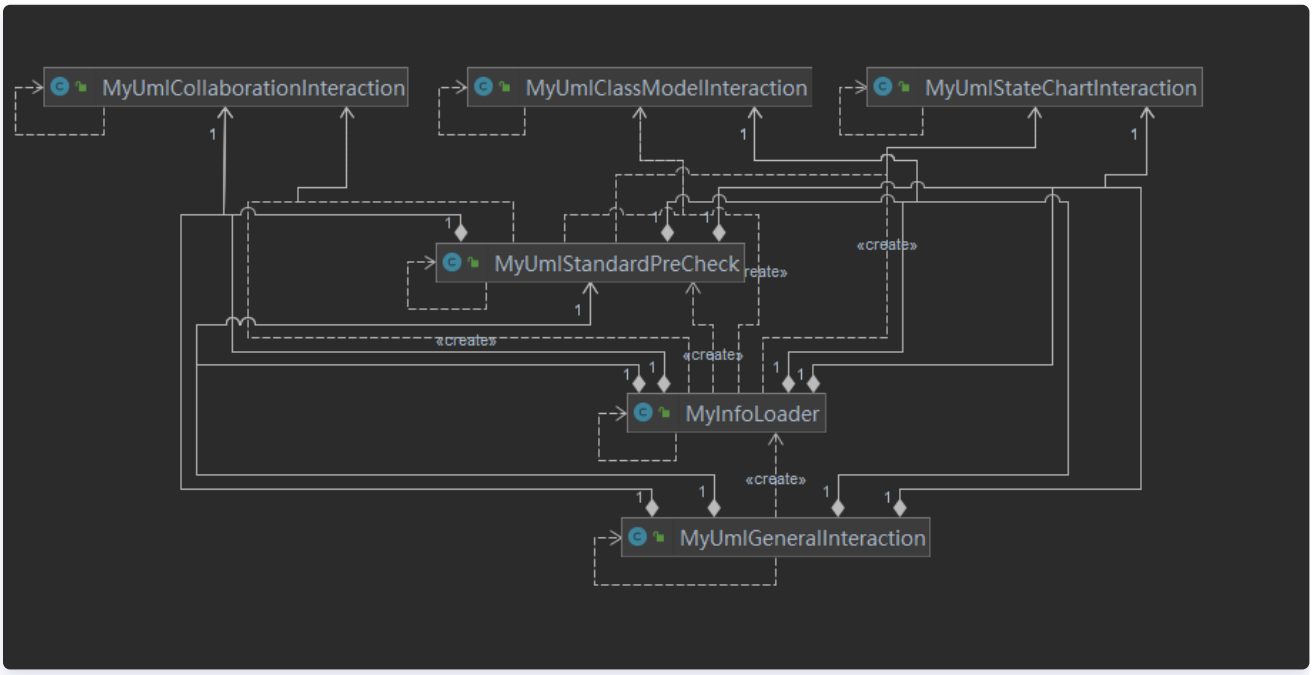
最后把所有的接口合在一起去重，看是否有重复的，如果有表明需要置标志位，无效。

在 `check` 模块中只需要对每个类更新一下，然后查一下标志位就好了：

```
public void checkForUml003() throws UmlRule003Exception {
        HashMap<String,MyInterface> id2Interface = myUmlClassModelInteraction.getId2Interface();
        Iterator iterator = id2Interface.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String,MyInterface> entry = (Map.Entry)iterator.next();
            MyInterface myInterface = entry.getValue();
```

```
            myUmlClassModelInteraction.updateFatherInterface(myInterface);
        }
        iterator = id2Interface.entrySet().iterator();
        HashSet<UmlInterface> ans = new HashSet<>();
        while (iterator.hasNext()) {
            Map.Entry<String,MyInterface> entry = (Map.Entry)iterator.next();
            MyInterface myInterface = entry.getValue();
            if (myInterface.getDup() == true) {
                ans.add(myInterface.getUmlInterface());
            }
        }
        if (!ans.isEmpty()) {
            throw new UmlRule003Exception(ans);
        }
    }
```

`Rule2` 有里面关于类和接口是否成环的问题，因为CPU时间十分充裕，直接用暴力做法就好，需要注意的是，对于已经遍历过的点需要打上标记，否则将导致TLE



关于类的成环问题，由于只存在单继承，最简单的做法就是一直查父类，看是否能回到开始的类，如果能，说明类的继承会成环

需要注意的是，当所查的类在一个环之外的时候，会出现死循环，所以还得记录查找路径，当再次出现记录的路径里的类的时候应该结束查找。

接口成环我采用了bfs如果会回到最初的接口表示成环，注意为查过的点打上 `flag`

二、四个单元中架构设计以及OO方法的演进

- 第一单元：主要是多项式求导的相关问题，重点在于理解类的继承以及对于面向对象思维的初步理解，由于刚开始基本没有面向对象的思维，做出来的架构很差，重构一次之后用二叉树做出了自认为扩展性很好很棒的架构，但是一到化简就**头皮发麻**了，之后听到老师说不一定是二叉树，可以是多叉树（其实我感觉二叉树相比多叉树更具有面向对象味），然后又做了一次重构，达到了比较好的效果。我的基本架构就是不同的函数虽然求导的具体做法不同，但是都可以抽象为项，然后项和多项式之间可以相互套娃，都有求导接口，对顶层求导，会将求导这个方法一层层分发下去，而底层有不同的具体实现。
- 第二单元： 主要是关于多线程的理解，关于多线程代码的编写与调试，关于生产者消费者以及其他部分设计模式的理解。在第一次写的时候由于没有考虑清楚细节，对于死锁的认识也不清晰，遇到了各种各样的bug，一团乱麻，再加上对于多线程debug也不熟悉，干脆推了重写，由于第一次作业架构还可以，后两次作业就直接把前面的架构**包一下**就能直接用了。总的来说就是需要分层次，该电梯管的就电梯管，该调度器管的就调度器管，不要有“长臂现象”，否则容易造成死锁，并且效率损失严重！
- 第三单元：主要是关于JML语言的理解与实现，把**规格**这个概念建立在我们脑海中。虽然写起来一长串一长串的（课程组写ಠ_ಠ）但是对于理解确实很友好，不会产生二义性。本单元是真的可以说没有任何架构，把指导书看好然后对着课程组的代码实现相关的函数就完事了，唯一需要注意的就是方法的复杂的，所以会涉及到一些算法的问题。

- 第四单元：主要是关于UML图的理解，训练的是层次化模块化的能力。首先需要通过starUML理解好作业中的UML图（类图，时序图，状态图）的相关概念与树状结构，然后自己构造相应的类提供相应的存储与查找方法进行查询。相比第三单元多了些架构设计，但没有多很多，做好层次化管理就好。

三、四个单元中测试理解

- 第一单元：第一次自己动手写评测机，对于个方面都不太熟悉，加上第一单元我并不是很会递归下降，不仅作业中读取很费力，评测机的数据生成也很费力，即使勉勉强强生成了数据强度也还是不高，导致第一单元作业中有很多bug~~日日~~，具体来说我的做法就是通过正则表达式解析逆向生成数据，然后跑出结果，然后与正确结果对拍，正确结果来自 `Python` 里的 `sympy` ……掌握了评测机的主要搭建思路：

  （构造数据->用命令行跑代码并记录结果->与其他结果对拍并记录对拍结果）为后面的评测机书写打好了基础。
- 第二单元：在 `Python` 中构造电梯类，生成数据之后，跑代码，每输出一个提示信息，就读取并解析，更新 `Python` 中电梯类的相关状态并检查，一旦不合规范就相应的错误
- 第三单元：随机生成各种数据（强度可能不高，用数量保证质量），然后找几个同学一起跑并对拍；专门构造一些容易超时的针对性数据，用 `time` 库来测运行时间，精度很高
- 第四单元：先自己构造各种边界数据，考虑到一些基本情况，进行手动人工测试，确保程序大致正确。然后手工生成针对性数据，多个同学一起跑并对拍。

四、课程收获

一学期的OO课终于结束啦！总结了一下，大概有一下收获：

- 面向对象思想的初步建立，无论什么好思想都是深邃的，思想的建立不可能一蹴而就，面向对象编程思想的初步建立将自己领进了这个领域的大门，为之后自己的探索指了一条明路
- 码量的飞跃，一学期算下来作业（含重构）加评测机大概也有6000~7000行了吧
- Java语言的熟悉，想要熟练掌握一门语言最好的方式就是多用它编写代码。此外踩了许多奇怪的坑，积累了经验
- 评测机编写能力大大提升
- 学会了许多工具的使用，特别是git，对于码量较大的项目是必不可少的东西，以及IDEA里各种大幅提升体验的插件
- 了解并掌握了相关的设计模式
- 建立了相关概念，例如：多线程，JML，UML
- **磨砺了心智**

五、具体改进建议

- 实验课最好能公布答案，不然自己根本不知道做对没，上机几乎等于无效上机，实在想不到不公布答案的理由
- 指导书中某些写的不清楚的地方希望能写的更清楚一点，特别是第四单元，看完指导书真的是一头雾水
- 希望在每个单元结束后能展示一部分优秀代码，因为真的觉得自己有的地方~~写得很丑陋~~
- 希望在互测中刀人之后能刀中了谁的反馈，这样就不会有漏网之鱼了，同时也能减少恶意Hack的情况
- 希望研讨课的质量能更高一些，最好定几个ddl，做完PPT后助教要进行审核，对于质量不合格的打回重做，仍旧不合格将取消本次研讨分享资格
- 讨论开通搜索功能，并开通标记功能（对于有的帖子以及看过了，可以标记为看过，避免浪费时间）