面向对象设计与构造2022第四单元总结

一、架构设计

由于三次作业都是增量开发，所以这里仅仅截取最后一次作业进行分析。

可以看到基本可以分割为四个独立的部分，刨去输入输出等无用部分，大致可以分为类图，流程图，顺序图，检测四个类块。当然也可以把检测看成三个图的子部分。

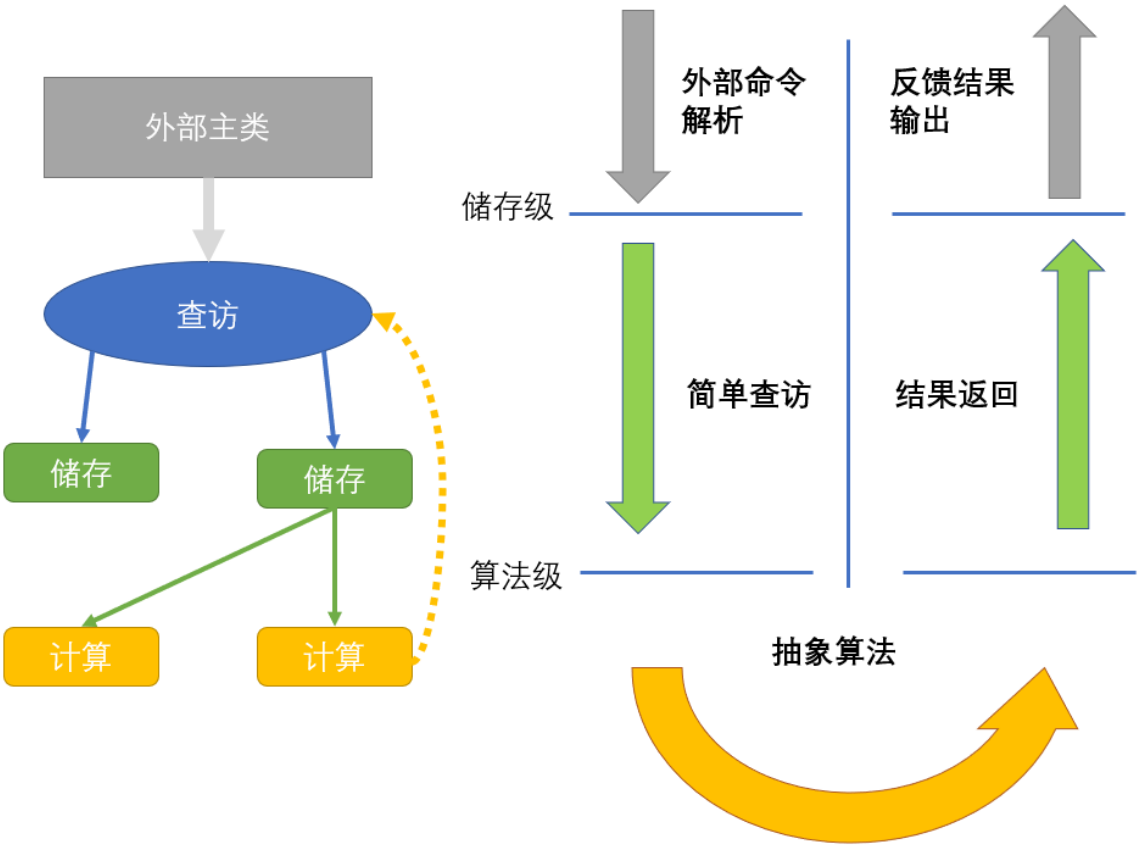
将需求分为三个部分：

- 查访：询问及询问的处理
- 存储：储存给定的模型
- 计算：对查访的补充，对无法直接查询的询问，进行计算

举例：

- 查访：询问人数，类的个数
- 存储：MyLifeLine
- 计算：关键节点

那么可以得到本单元作业的逻辑图如下：



这一次作业中，由于每一个元素都有唯一标识的id，因此我将id作为了区别元素唯一标准，规避自己构造的扩展类没有覆写equals方法和HashCode方法带来的问题，hashmap的O1查询真香。

```
private final HashMap<String, MyClass> idClassMap;
private final HashMap<String, MyClass> nameClassMap;
private final HashMap<String, MyOperation> idOperationMap;
private final HashMap<String, MyAttribute> idAttributeMap;
private final HashMap<String, MyParameter> idParameterMap;
private final HashMap<String, MyAssociation> idAssociationMap;
private final HashMap<String, MyAssociationEnd> idAssociationEndMap;
private final HashMap<String, MyInterface> idInterfaceMap;
private final HashMap<String, MyInterfaceRealization> idInterfaceRealizationMap;
private final HashMap<String, MyGeneralization> idGeneralizationMap;
```

为了规避万恶的500行限制，只好进行一些不那么优雅的操作，比如强行拆分，略有遗憾。

二、架构分析

第一单元

第一单元还在学习java语法，上学期没选上java，这学期开局就做了大牢。最终基于表达式、项、因子的三个层次的结构，对表达式解析程序进行扩展，完成了第一单元的作业。

在完成课程之后，再来看第一单元，确实难度并不高。只能说自己成长了。

第二单元

第二单元是面向多线程的编程，对于多线程这个从未get过的概念，非常艰辛。一直是不求甚解的完成任务。

后来慢慢想明白了逻辑，构建了一个双重的生产者消费者模型。分别是输入流，调度流，电梯。这样的方式还是挺好看的，但性能优化上略有不足。只能放弃部分优化。

分电梯算法，我自己想了一个及其复杂的加权动态最短路算法，这里不再进一步展开（详见我的U2博客），当然也理所应当的被强测hack麻了。事实证明，不应该过度coding。

第三单元

第三单元主要是契约式编程，直接给了架构让本懒癌狂喜。

当然，大部分的精力还是放在了评测机上，由于jml过于谜语人，而且缺乏资料（好像北航对其的重视有些。。。）大部分时候只能靠群众的智慧。也就是大家分头写，在一起对拍。

偶尔复习一下DS也不错嘛，就当准备社会计算期末了。

第四单元

第四单元是UML建模。不得不说，这是体验最好的一个单元。

这单元非常的面向对象，只要有一颗面向对象的心，只需要coding就行了（指一般不会有什么奇奇怪怪的坑）。hhh，更多内容详见上一段。

三、测试分析

笔者在互测中，使用了测评机和对拍机。

使用对拍的方式进行测试。未使用Junit（真的不好用，又不能造数据）

完全随机

采用完全随机的策略，通过扩大测试次数以及单次测试的指令规模发现Bug

单元随机

可动态调整各指令出现的概率和指令中id生成的概率等，集中测试相应的功能模块

评测机暴力Hack，大概率是Hack不下来的。假如这位同学有评测机，那么大概率在A房，地毯式轰炸并不一定能保证效果。对于高手互博，往往功能本身不会出错，只有想怪数据，反常识数据，才可克敌制胜。

既然如此，看代码就是必不可少的。和刀我的同学交流，发现他也是通过看代码才刀到。

据我不完全观察，防御力较低的代码遵循以下规律。

- 优化多
- 父子类之间很少调用方法
- 重复代码过多

四、课程收获

最大的收获，应当算是学会了如果写工程级的代码。

以往的coding，例如ds，难度可能更高，但往往不需要工程化的方法。故而也就随心而动，没有做好区分，究其原因，还是解构问题的时间，不如留给coding。当然对于小量级的代码是可用的，但是随着功能增多，面向对象的思维变得不可或缺。

此外，还学会了一些些的java。虽然听起来有点水，但多少是些许聊以自慰的成绩罢。

非常感谢一路OO结识的几位友人，不仅在OO课上，也在OS等课上，给了我莫大的帮助。虽然在此有些不妥，但私认为，有幸与他们结识，比OO课程的收获更大。

五、课程建议

第四单元作为最面向对象的一个单元，写起来虽然很爽，但是总有些遗憾。可能是身处考期，但也许课程组可以对问题进行下修订，让最后的一个单元体验更好一点。

对于我这样的普通同学，一是老生常谈的第一次作业谜语人问题。和其他单元作业第一次一样，一脸懵的完成任务要求可能是比较常间的现象。

二是数据覆盖性问题。虽然虽然虽然（重要的事情说三次）测试题和中测数据已经覆盖很多了。但是功能点零散的第四单元，本就高概率覆盖不到错误。更别提许多人的测试题都是随便选的（急于看到中测结果）

对于提高同学，一是评测机难造，虽然课程组给了一些评测机造法提示，但是大家实际上都是生成标准输入而非uml图的形式进行测试，让本就漏洞百出（非贬意，这是自然的）的评测机更加覆盖不全。

二是迭代问题，这单元的评测机相比之前，迭代难度亿点点大（尤其是task3）

当然这些问题很多是第四单元的性质决定的，想改也改不了。但总有一些方面，可以让同学们体验好一些。毕竟这单元不像U2，需要大量新知识学习，而是单纯的面向对象（正因此写起来也很有快感）
可能的解决方法如下：

一，类似第一单元，设置不能得到满分的官方包，调用官方包后，只需要实现某些方法，数据输入等等都被抹去。相应会损失少许分数。这样不至于影响提高同学。

二，统一下格式，首先是task1，很多的说明在gitlab的包的readme里面，非常不直观。二是许多专业用语，并没有给出定义，也非常不形式化（重点），例如元素，TRIGGER等等。虽说这些是应该学习的内容，但是但是但是显示给出会更好。

三，针对Task3，一次不单一检测一个错误，中测只有十个点，再加上仓库也很少（更别说很多普通同学完全不看仓库）。一种可能的更改输出要求如下：类图检查无误，状态图检查无误，顺序图r009错误，错误信息为XXX。

虽然我了解部分错误可能在实现时出现重叠，例如r003和r004 某些实现方法下可能无法同时判断，故而要求输入仅有一错。但是更改输出要求，可以检测更多的理解错误，同时保留输入中仅有一错的限制。当然会有同学强制只输出一错来混过中测，但是在有不能得到满分的官方包下，应该不会产生过大影响。

四，服务强迫症，小建议，指导书顺序有时是类图状态图顺序图，有时是类图顺序图状态图。。。

对于讨论，关注到的问题主要有以下：

一，部分组员摸鱼。尤其是线上之后，小组讨论往往变成三两个人完成任务。
二，组员不认识。尤其是考虑到中间有一次换组。
三，时长。对于展开讨论的组，讨论时间不太够用，对于在上演《等待戈多》的组，讨论仿佛在坐牢。
解决方法很多（当然不一定有效）

首先对于point2，在oo研讨课开始之前，首先组建小组，并且通知要求进行一次小组作业。这次作业的内容很简单，对pre进行一些分析，小研讨等等（正好促进同学们写pre）

小组的分配按照pre的表现。一般来说，交的早的同学，更活泼，或者至少能成为group里的技术nerd。混合这些同学，可以至少保证组内不那么沉闷。

我推荐不再重组，如果课程组非常想的话，可以在重组（仿照前次，按照之前单元的表现重组）之后，依然安排一次简单的小组作业，插在两次研讨课的gap周。

由于6系特殊性，同学往往卷摆两极分化。虽然课程组可以通过分数的形式一定程度避免摸鱼，但是push的政策很容易招人反感，组内多个"卷王"的话，也可能导致不愉快。可以多安排助教老师，在讨论期间走访，记录活跃/不活跃的同学（反正座位固定）也可以帮助内向的同学参与讨论，或者对于尬住的组，主动开启话题。

或许可以要求每位同学进行一次上台汇报，并且老师和助教会进行即时提问（毕竟同学们提问热情不高）也可以督促那些group能力不强的同学加入进来（至少一次也是收获）

最后，如果不幸明年疫情常态化，可能需要做好预案。