面向对象第二单元博客作业

1. 三次作业的设计策略

作业要求：从多线程的协同和同步控制方面，分析和总结自己三次作业的设计策略。

1. 第一次作业

第一次作业是单部电梯的调度，要求实现一个单部多线程傻瓜调度（FAFS）电梯的模拟。实际上在这个系统中，只有两个线程就足够了：

·调度器（Dispatcher）线程：负责从键盘中即时接受请求，放进一个请求队列中。

·单部电梯（Elevator）线程：在请求队列中调取相应的方法得到最先到达的请求，并根据自身的调度算法进行执行。

我们可以看出，实际上这两个线程是一个简单的消费者-生产者模式。而“盘子”就是请求队列。当时我考虑的是将这个请求队列单独封装成一个类，并把它做成线程安全的。这样在调度器和单部电梯启动后，其所有的功能就依赖请求队列提供的线程安全方法进行电梯请求的处理。这是线程的同步控制设计。

由于第一次并没有严格要求CPU运行时间，并且缺少多线程编程经验，没有使用notifyAll()和wait()机制，而是使用了在请求队列为空时，电梯线程轮询的方式。这样做虽然达到了题目要求，但是显然性能（尤其是CPU运行时间）方面是非常差的。

因为第一次作业没有性能上面的高要求，所以在电梯调度直接选用先来先服务的方式，没有进行捎带操作。

1. 第二次作业

第二次作业虽然也是单部电梯的调度，但是这次作业要求实现了多线程可捎带调度（ALS）电梯的模拟。可以看出，这次作业和上一次作业在多线程协同和同步控制上没有任何区别，所以并不需要耗费很多的精力进行多线程架构的重新构建。唯一在线程同步方面的改进是，为了减少CPU运行时间，我使用了notifyAll() - wait()机制。使得电梯队列在没有请求的时候可以不用轮询。

题目给我们提供了一个思路：设置一个主线程，以及用一个队列来实现捎带线程。我用了题目中给出的办法，其实效果并不好，并且还容易引入一些Bug。这也是我最后在强测中失败的主要原因——考虑主请求的话，实际上没有什么好处，反而会使得程序运行的逻辑变得有些混乱。比如在主请求对应的乘客还没有进入电梯的时候，一些可以捎带的请求捎带还是不捎带？如果捎带了，哪个请求才是当前的主请求，是第一个真正进入电梯的乘客对应的请求，还是电梯第一个检测到的请求？如果不把这个捎带请求当做主请求，例如，主请求是9-15层，而现在电梯在1层，电梯运行到3层时有一个3-5层的请求。也就是说如果考虑捎带，那么这个请求在主请求还没有上电梯的时候就能够到达了目的地。那实际上这个请求才是我们说的主请求的真正概念，如果不把他当做主请求，主请求又有什么意义？等等这样的问题，我当时在真正做的时候造成了不少的逻辑混乱。而最后强测中也测出了我的这样的问题，这种问题当然首先和我低估了这次作业的难度、测试不足有关系，也和一开始的电梯线程的架构有很大的关系。

事实上这次作业只需要改动一下电梯线程的运行策略。我在修改了自己的Bug之后明白了：题目提示我们所谓的主请求，实际上可以抽象为电梯运行的方向和目的地两点。修改之后我的想法基本是和普通的现在使用的电梯的捎带想法一致：电梯每到达一层，就调用请求队列中的查询功能，查询当前层有无现在正在等待的、与电梯运行方向一致的请求。电梯中建立一个集合用来保存当已经上过电梯的乘客——一个乘客是不是已经上了电梯。如果这样来看，一开始的请求（主请求）仅仅是规定电梯运行的方向。而如果新进入的电梯请求到达的位置更远，那么更新电梯的目的地。如果电梯已经运行到达了目的地，电梯就进入等待状态，等待下一次被请求唤醒。上面的诸多问题就可以解决：仅考虑方向和目的地的话，主请求引入的相关问题也就不复存在。

根据以上的分析我得到了一些很宝贵的经验：一个具体问题的抽象、关键信息的提取和利用在面向对象程序设计中是非常重要的。当时也有很多同学提醒我不要把“主请求”这样的东西考虑的那么死，要考虑更灵活的设计，但是因为当时我的架构已经基本完成，所以懒得去修改，最后导致了非常很差的结果。虽然以上内容不属于多线程设计范畴，但是是我出错的点，也属设计思路的问题，故提前阐述之。

1. 第三次作业

第三次作业的要求和前两次的要求有了很大的变动，要求完成多部多线程智能（SS）调度电梯的模拟。其主要变动大致可以分为以下两个方面：

①电梯数量发生了改变，由以前的一部变成三部；

②电梯运行的机制显然要发生改变，因为电梯能够到达的楼层有限、速度也不同；

③一个请求有时候必须要拆分成两次进行处理。

这个问题在多线程的设计上，难度有了很大的升级：从以前的一个生产者对应一个消费者变成了一个生产者对应多个消费者。在多线程同步上面，主要体现在：所有线程在请求队列变为空时都处于wait状态，等待插入请求。插入请求的是主线程，主线程插入请求后立即notifyAll，让所有阻塞进程去抢这些资源。在抢的时候也有一定的机制，保证能够直达的线程抢到对应的请求。除此之外没有其他的要求。这次我把控制和请求队列放在一起形成一个统一的调度器，并且是线程安全的。这样就保证了线程的同步。

怎么来保证多个线程协同作用一个请求呢？我一开始的想法是将每个不可直达线程在进入调度器的时候就把它分好，规定好谁来运送第一程、第二程。到后来感觉一开始拆分这样的方法太过暴力，也没有考虑三部电梯当前的状态。于是我就想到电梯在运行时把请求划分。在捎带不可到达请求时，先考虑谁能到达这个位置，然后把它放在离着换乘点比较近的可到达楼层，之后再把它扔回到调度器内部，换乘后电梯的专属队列中。等待换成后电梯来捎带他。

除此之外，在线程结束的时候也要考虑线程的同步结束。在获得结束信号后，主线程给调度器发送一个结束信号，调度器接受结束信号后将内部的结束标志设为1。当其他线程发现结束调度重新找调度器要请求的时候，如果调度器内其专属的请求队列为空、请求队列为空、自己内部运行的队列也为空，那么这个线程结束运行。

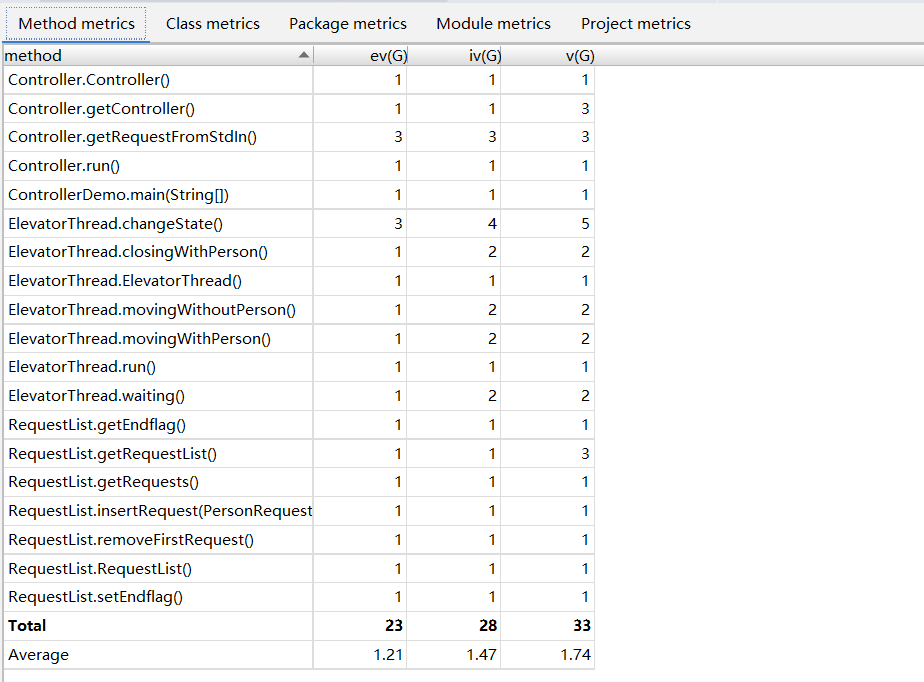
总而言之，这次作业较前两次作业有了很大的难度提升，不仅要考虑电梯的运行策略，还要考虑多线程的协同、合作、同步。是一次非常锻炼人的多线程编程水平的作业。

1. 程序度量与评价

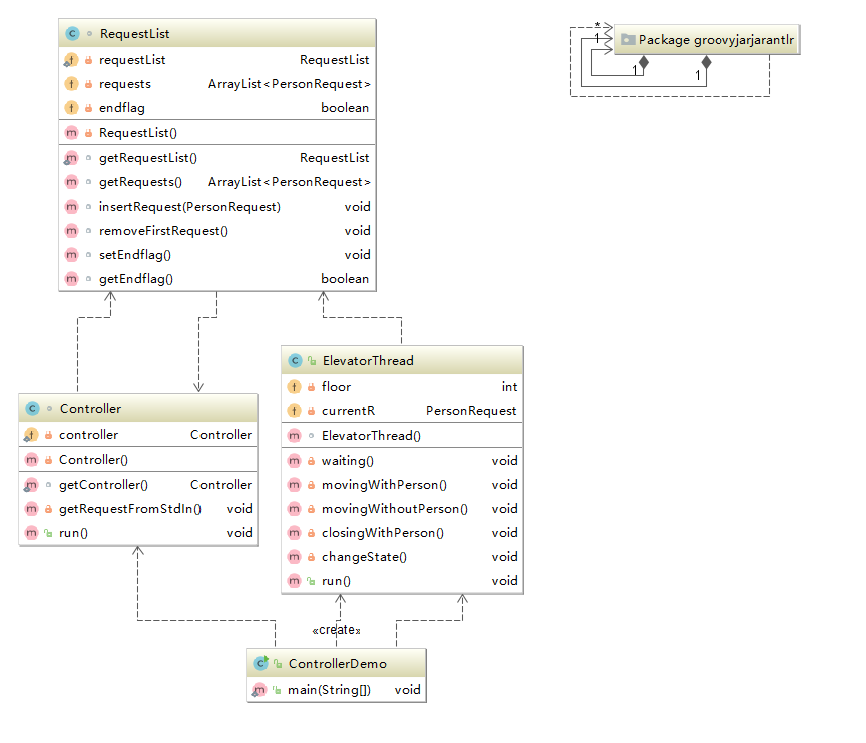
作业要求：基于度量来分析自己的程序结构度量类的属性个数、方法个数、每个方法规模、每个方法的控制分支数目、类总代码规模计算经典的OO度量，画出自己作业的类图，并自我点评优点和缺点，要结合类图做分析通过UML的协作图(sequence diagram)来展示线程之间的协作关系（别忘记主线程）从设计原则检查角度，检查自己的设计，并按照SOLID列出所存在的问题。

1. 第一次作业

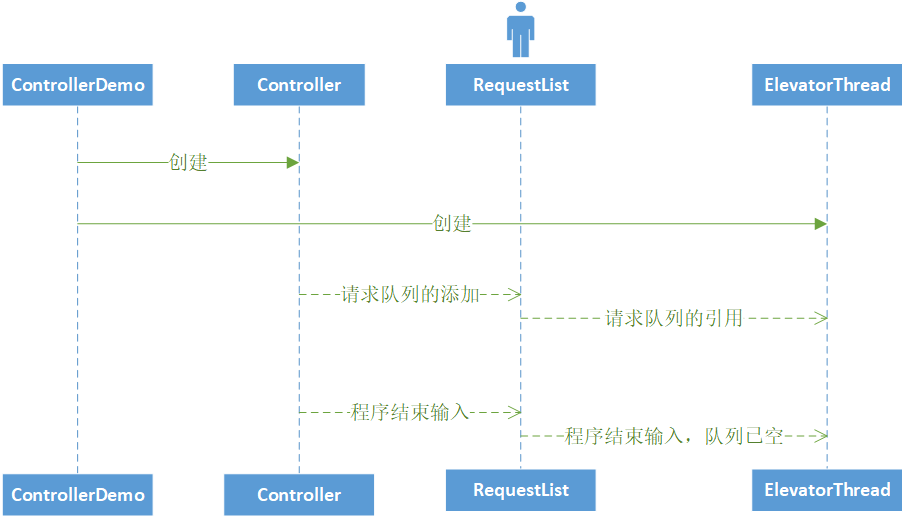
①类的复杂度分析（OO度量）



②类图



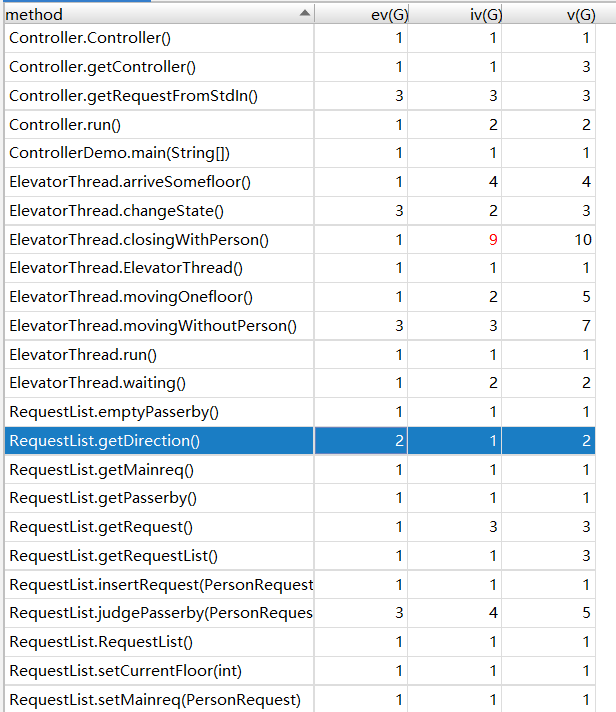
③UML协作图

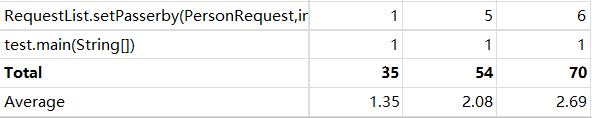


第一次作业比较简单，在程序性能上不做过多的分析。但是值得一提的是，在控制器的使用和创建上面，我使用了单例模式。可以这样去做的原因是控制器有且仅有一个，但是在第一次作业并没有体现控制器单例模式的好处。

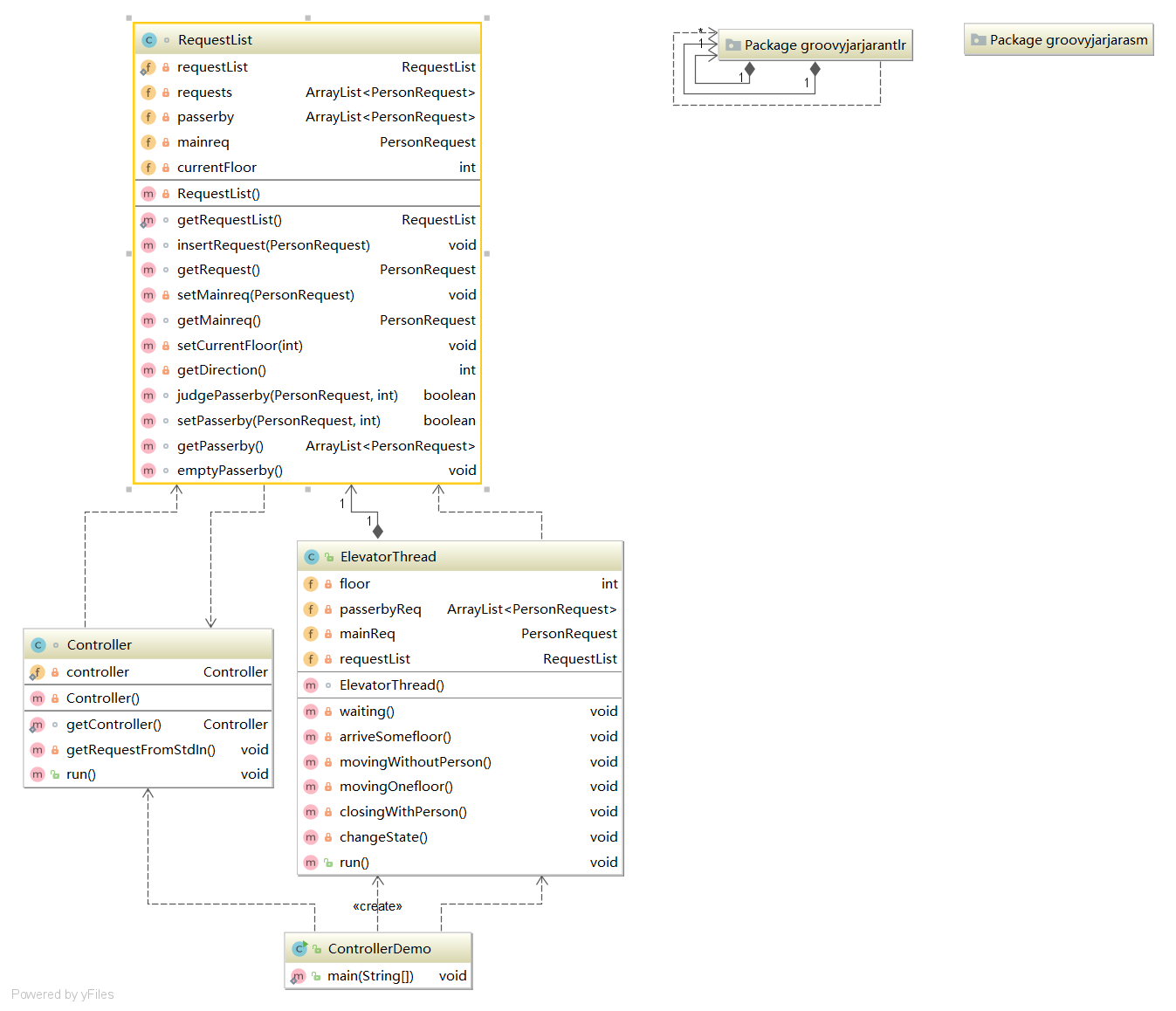
1. 第二次作业

①类的复杂度分析（OO度量）



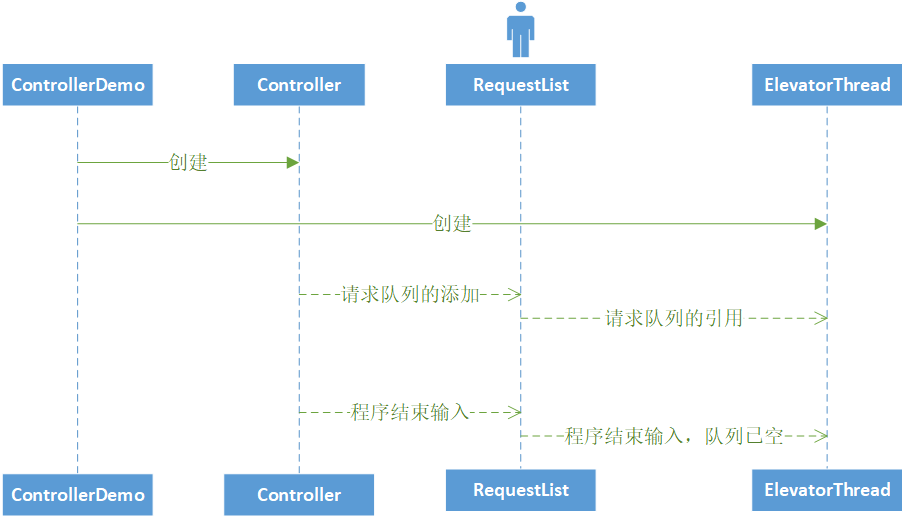


②类图



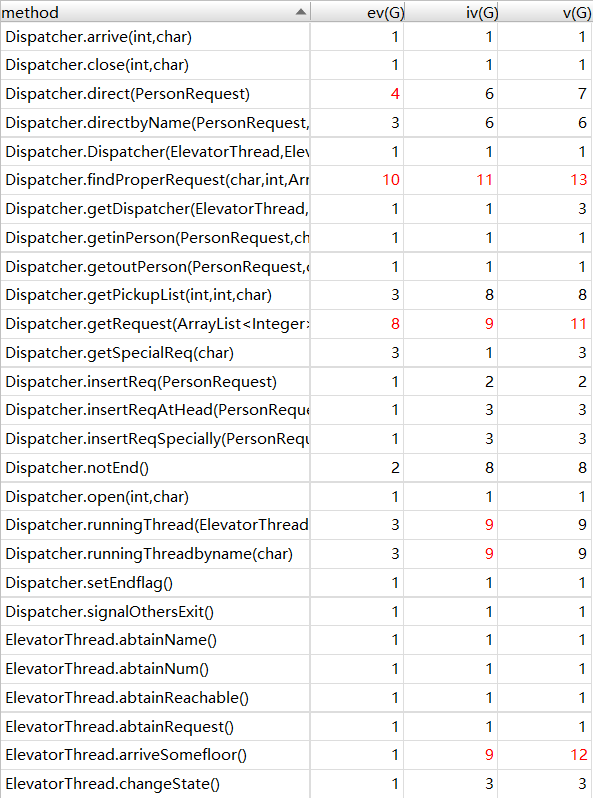
③UML协作图

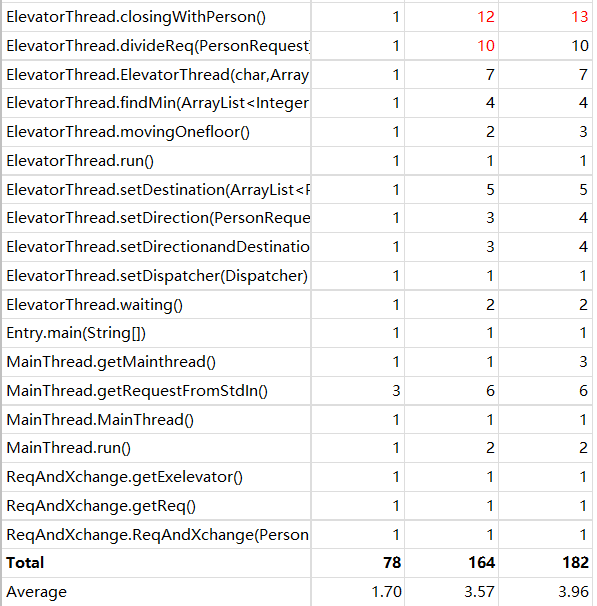
因为这次作业并没有线程之间调用关系的问题，一切多线程的访问、控制、协作和同步都沿用了第一次的方式，在这里我们再次把第一次作业的UML图放在这里。（事实上，如果完成了第一次作业之后再来写第二次作业，我认为写第二次作业要比第一次简单的多。因为多线程设计方面基本不涉及任何的改动，仅仅涉及了电梯调度算法的微小改进。）



1. 第三次作业

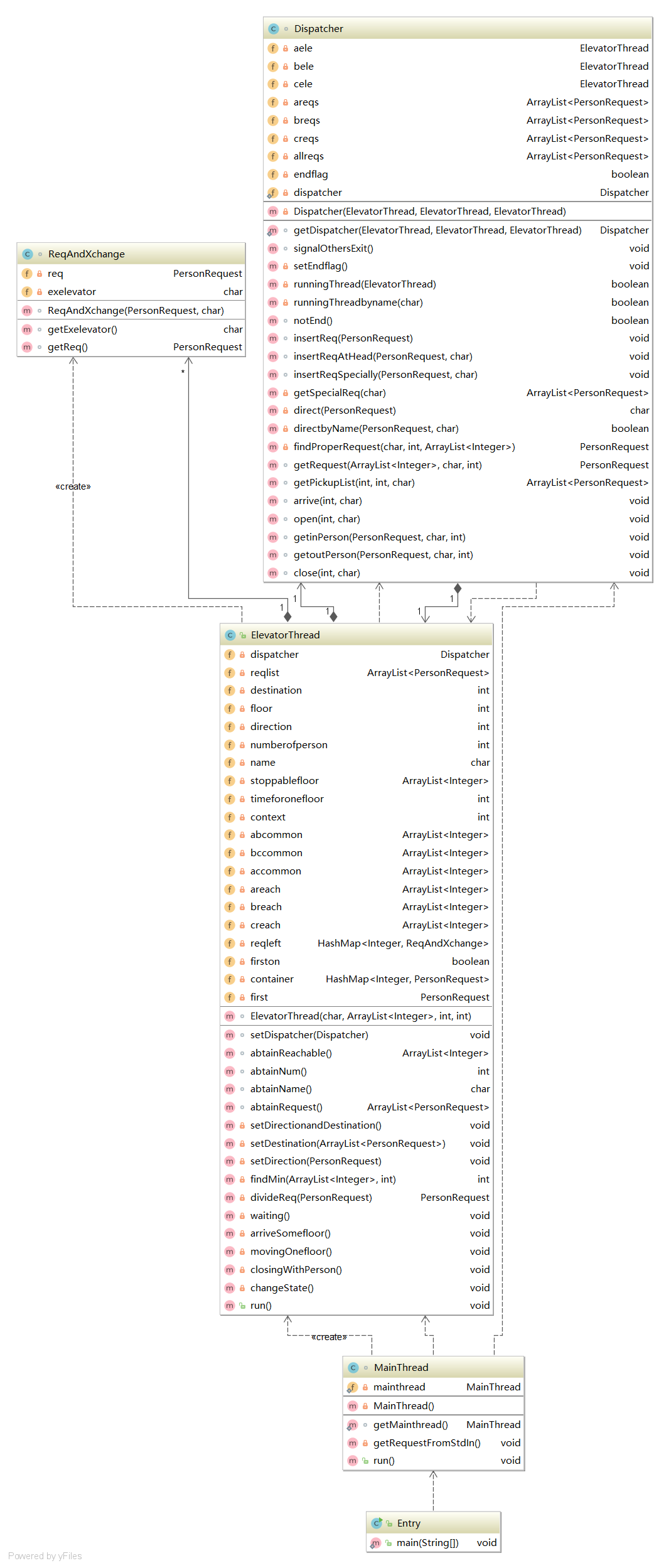
①类的复杂度分析



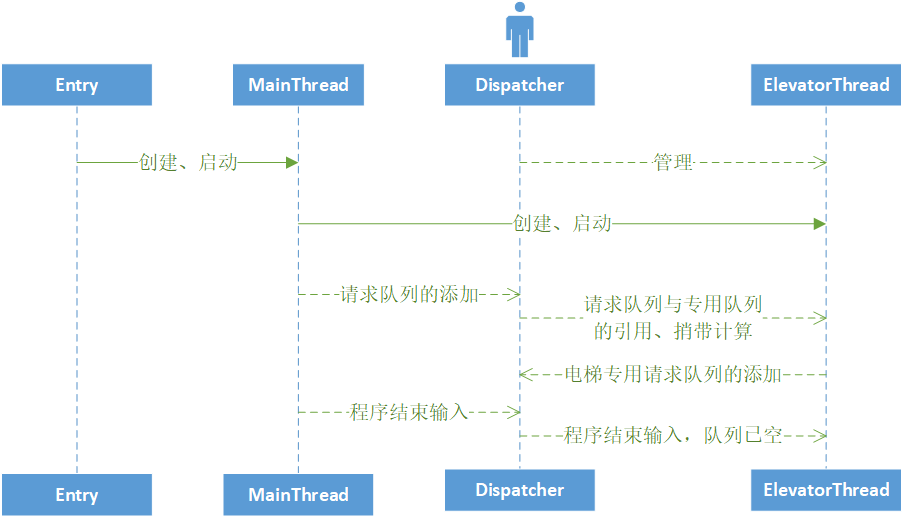


可以看出这次作业的OO度量就没有那么好了。实际上我比较过自己的代码和其他一些同学的代码，在这种比较复杂的多线程程序中就能够看出水平上的明显差异。今后还是要尽量做到高内聚低耦合，能够进一步提升程序性能。但是我想这是一种习惯，需要长时间的锻炼才能做到。

②类图



③UML协作图



1. SOLID原则以及从SOLID原则看这三次作业
2. SOLID原则

（摘选自http://www.cnblogs.com/shanyou/archive/2009/09/21/1570716.html）

①单一责任原则：

当需要修改某个类的时候原因有且只有一个（THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A CLASS TO CHANGE）。换句话说就是让一个类只做一种类型责任，当这个类需要承当其他类型的责任的时候，就需要分解这个类。

②开放封闭原则

软件实体应该是可扩展，而不可修改的。也就是说，对扩展是开放的，而对修改是封闭的。这个原则是诸多面向对象编程原则中最抽象、最难理解的一个。

③里氏替换原则

当一个子类的实例应该能够替换任何其超类的实例时，它们之间才具有is-A关系。

④依赖倒置原则

高层模块不应该依赖于低层模块，二者都应该依赖于抽象。

抽象不应该依赖于细节，细节应该依赖于抽象。

1. 作业反思

①单一责任原则：

并没有做到，尤其是第三次作业。在添加了一些简单的限制之后，自己虽然想办法把他们都写进单独的对象，尽量避免参数的传递，但是最后还是不免会这样。我想到一些解决措施：比如正如单一责任原则所说，我们可以开创一个新的数据类，专门用来管理各种参数、电梯的运行状态，这是一种很好的方式。事实上，当时在设计的时候我是这么想的，但是在真正实现的时候因为感觉这样做会有一些多多少少的困难，所以最后还是放弃了这样的想法。最后就显得程序的类与类之间重叠部分过多，参数调用严重。这些都值得我自己去好好反思。

②开放封闭原则：

第三次作业据我了解有很多同学运用了线程池，即工人-工厂模式。这样写大大有利于程序的扩展性，多开几部电梯也完全不是问题。但是反观我的程序，如果重新开一部电梯显然要耗费大量的时间和精力。就这一点来看，想办法找出不同对象的相同点是一个方面，而怎样把多个对象化成一个对象来考虑、来统一调度又是一个很重要的课题。

③里氏替换原则

因为这次作业没有用到子类，所以不再分析。

④依赖倒置原则：

这种原则我感觉涉及到了面向对象思想上面的很多内容。如何做到“高层不依赖底层，而都依赖抽象”呢？这就好比我们在写多线程电梯的时候，是不是能够做到不写电梯的实现就能够把整体的控制器框架搭建好、设计好，能够保证电梯内部的功能不影响控制器的功能？为什么能够不影响？这个法则的后半句就是答案：一切都依赖于抽象。也就是说一切在题目建模的时候，我们就应该可以分析出来，高层应该怎样建模才能不依赖底层，底层怎样建模才不借助高层构架。只有这样才能把一个个有联系的对象独立设计出来，才是面向对象本质的精神。

1. bug分析

作业要求：分析自己程序的bug分析未通过的公测用例：特征、问题所在的类和方法。特别注意分析哪些问题与线程安全相关关联，分析bug位置与设计结构之间的相关性。

1. 第一次作业

因为第一次作业比较简单，也并没有花费很多时间去debug。在强测中也没有很多的问题。并且这次作业不涉及过度的调度算法，所以在此不再进行分析。

1. 第二次作业

第二次没有通过的强测样例点：2,7,8,9。

2,7,9问题都是一致的：





这三个问题出现的原因是：电梯捎带策略有问题。因为每次电梯的运行都是按照“主请求”来进行运行，并且输出“IN”、“OUT”信息都是通过轮询电梯中的请求数组各个请求的FROMFLOOR和TOFLOOR来进行操作的。但是当时在考虑的时候没有想过当主请求到达的时候，有可能反向进行运行。比如第一个点的错因就是：主请求为从1到10层，而person17是从5-16层。当主请求到达之后，选取当前等待请求队列的第一个为主请求，这个主请求为10到-2楼。这样的话，电梯没有把person17送到16层，又重回到了第五层。在第五层由于没有建立“电梯内是否真正进人”的机制，所以又把17号重新拉上了电梯。这三个问题就都是这样产生的。

至于第8个点，显示的内容是：



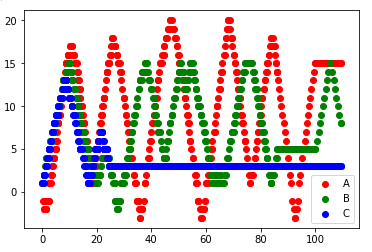
这个我仔细和其他同学进行了对比。发现我的原因主要是：在主请求还没有真正进入电梯的时候，我的电梯不捎带但是他们的捎带。我不想捎带的主要原因事实上还是和“主请求”这个概念有一定的关系。详情可以见第一部分的分析。

综上所述，事实上，这次作业我并没有因为多线程的设计而犯下多大的错误，反而是因为调度策略没有理解到位，加上测试做的不够导致的。我这次作业仅仅提交了一次，实际上我第一次通过之后就一直觉得自己的程序没有问题的心态麻痹自己。这样看来确实是做问题的一大忌讳，丢失分数也是自己应得。以后一定要做充分的测试再进行提交。

1. 第三次作业

第三次作业我在强测中有5个点REAL\_TIME\_LIMIT\_EXCEED，分别是4、9、10、15、20。这几个点错误的原因我到现在还没有找到，我使用python+管道的方式对每个测试样例测试了5遍以上，出来的结果都没有超出200s。由于超时状态不能够看到自己的输出是什么样子，我对这个问题也非常为难。我尝试寻求助教帮助，助教确认了评测机评测无误，但是基于课程规则不肯提供测出来的数据，希望我和同学探讨解决。因为确实我们自己测不出来bug，所以我只能对一个测试点进行分析。分析的结果对我们还是有一定的启发的：

这五个点，我自己测试的结果都比同学通过的结果要慢30-40s。有一些他的性能分基本上为0，也就是说我基本上比通过的最慢的速度还要慢30-40s。为什么会出现这样的情况呢？我要来了一些同学电梯的运行轨迹图，与我自己的轨迹图进行对比。



我的轨迹图

可以发现，电梯C在运行了25s就不动了。而我查看自己同学的电梯运行图时，发现C电梯一直是动的。这是为什么呢？因为我在请求分段的时候，请求回扔的顺序是固定的，必定是先往速度快的电梯扔。这样的话C实际上在拉完自己能够直达的请求之后，就没有相应的请求再给C了。这是我的电梯C停运的原因。我真的很希望能够找出我超时的原因，本地测试了很多遍了，每个点测试5遍已经是达到了25-30遍还是没能重现出评测机的结果。所以这也是我本次作业的很大的一个遗憾点。

1. 心得体会

作业要求：心得体会从线程安全和设计原则两个方面来梳理自己在本单元三次作业中获得的心得体会

这次作业，通过实现三种电梯的调度和协同，由浅入深地认识了多线程编程。也学会了各种各样的机制，这对于我之后的学习和工作都大有裨益。与此同时我非常感谢助教同学对我的帮助，在我第三次作业没有找到bug向他提问时他非常耐心的解答了我的问题。遗憾就是这次作业完成的并不好，虽然自己好好写了但是还有各种问题，说明自己的能力不足，以后还要多加强。希望在下一次作业能够认认真真展现自己的知识和能力。