OO 博客作业

——15061125 赵泽

一、第一次作业

1、程序主体

第一次作业是处理多项式加减运算的模拟计算器。由于数据是一行读入的，为了避免因为一行读入数据过多，使程序崩溃，因此我采用了逐字符读取的方法。通过和同学的交流，在对输入格式匹配的方法上，我选择了使用状态机，通过当前状态、当前读入字符，判断下一状态，并对每一个状态进行操作分析（具体的状态转换后有图表描述）。由于对 java 编程和类的相互调用不是很了解，所以虽然我采用了五个类，但实际上大部分类中重要的方法只有一个。

各类大致说明：

Read 主类完成了程序的输入工作，每次读取一个字符后，交由 word_check 类进行判断是否为合法字符，即以下字符：

各类别字符 check 函数的返回值如下：

check返回值	对应字符	check返回值	对应字符
0	空格	5	)
1	数字0-9	6	+
2	{	7	-
3	}	8	,
4	(	9	其他字符

其中“其他字符”即对应不合法字符，进行报错提醒。之后将合法字符交由 State 类判断次状态，通过返回值决定相应操作。

Data 类用数组存储当前已有多项式，若多项式超出单条项数要求或是总条数，则在该类提示报错。当 read 全部完成后，将提示指令传递至 Data 类中，进行合并计算并按顺序输出。

Warning 类为报错提醒类，为了便于管理报错信息，采用不同的方法存储不同的报错信息，通过出错时调用不同的方法，起到对不同的错误进行不同相应的操作。

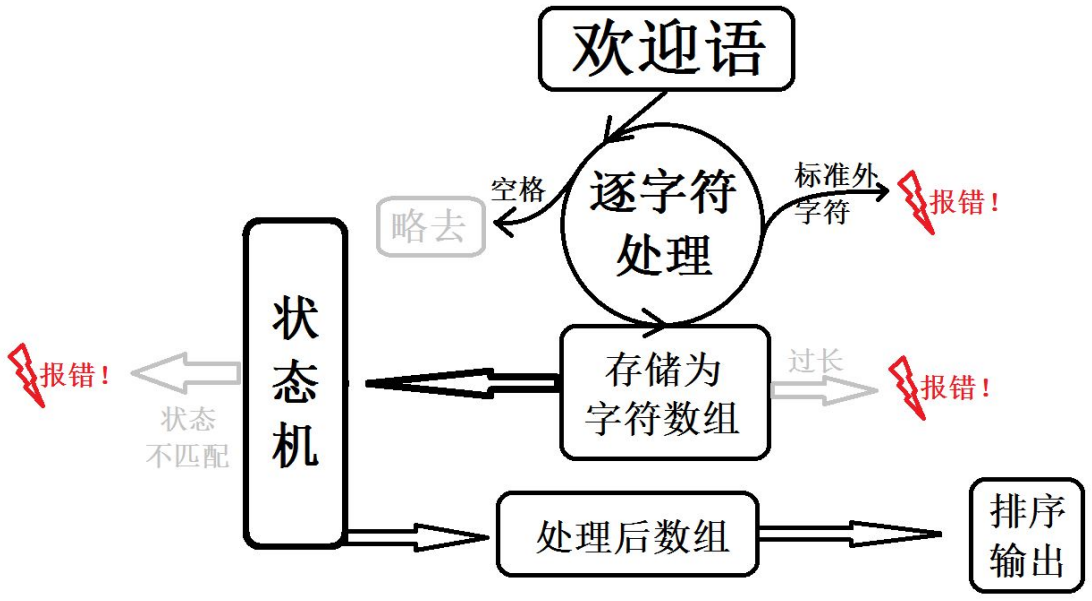
State 类即为状态机转换类，状态机通过 int 型变量 state 的不同取值，代表

不同状态。State 共有-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8 共十二种状态。状态转换关系如表：

	state状态	check值	对应字符	末状态		
报错状态	-3			此时为报错状态！		
		2	{	1		
第一次读取状态	-2	6	+	0		
		7	-	0		
第n+1次读取状态	-1	6	+	0		
		7	-	0		
					对应字符	check返回值
准备读{状态	0	2	{	1	空格	0
					0-9	1
准备读(状态	1	4	(	2	{	2
					}	3
		1	数字0-9	4	(	4
准备读系数状态	2	6	+	4	)	5
		7	-	3	+	6
					-	7
准备读系数状态	3	1	数字0-9	4	,	8
已读取负号					else	9
	4	1	数字0-9	4		
读系数状态		8	,	5		
		1	数字0-9	6		
准备读次数状态	5	6	+	6		
		7	-	-2	此处使用不再使用的-2状态用于特殊报错	
读次数状态	6	1	数字0-9	6		
		5	)	7		
一项读完状态	7	3	}	8		
		8	,	1		
重新读下一项	8	6	+	0		
准备读(状态		7	-	0		

所有不在表上的 state 与 check 对应关系，均为触发末状态至-3 报错状态，提示输入不符合规范。

具体的程序控制流程图：



2、程序结构度量

Data 类：共 90 行，八个属性，五个方法，其中每个方法均不超过 20 行

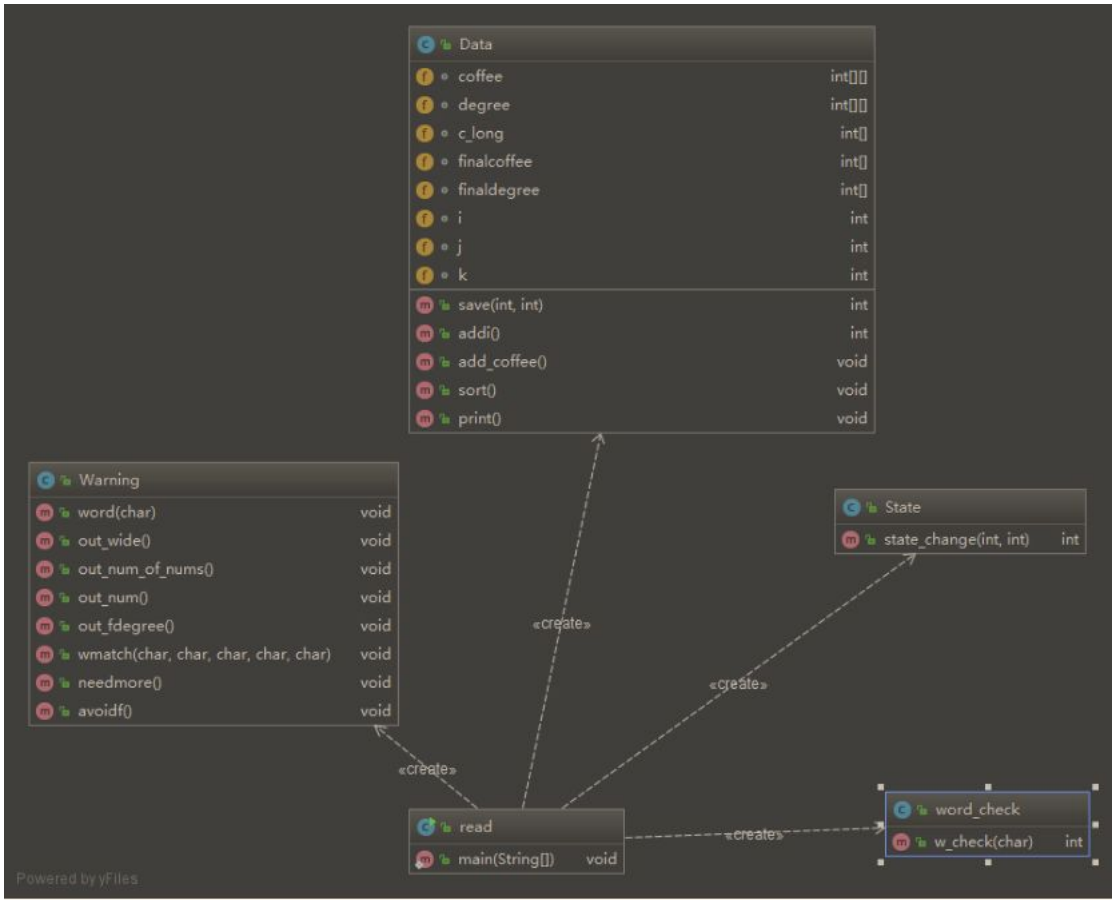
Read 主类：共 130 行，无属性与方法，其中读取部分 15 行，状态转换 80 行

State 类：共 87 行，有一个方法，无属性，全部为状态转换

Warning 类：共 35 行，八个方法无属性，每个方法均为两行

Word_check 类：共 33 行，一个方法无属性，全部为判断字符合法性

作业类图：



优点：类间联系较为紧密，方法调用较多

缺点：类内联系不足，很多类只有一个方法，可以合并统一或类内细化

3、分析自己程序的 bug

第一次 OO 作业可以说是我真正意义上的第一个 java 程序（之前的 helloworld 由于过于简单不算在内），由于对课程和语言的陌生，我尽可能的去贴合作业要

求，考虑各种可能的“极端输入”情况，但是由于疏忽，在状态机中未考虑当指数出现+的情况，使程序在某些时候会将正确输入视为错误格式。

4、自己发现别人 bug 采取的策略

第一次作业我分到的是一份无效作业，基本的功能和实现他都没有完成，因此我认为并没有太多代表性，所以我想讲一下我“自测”的方法，首先是和同学讨论极端情况，结合论坛中同学们的集思广益，构造一份较为完善的测试集。之后和同学进行对拍操作，通过文件对比进行输出的比对，找寻代码中的 bug 进行单步调试，直到修改成功。同时，在极端情况可实现，但总体测试样例篇幅较短时，通过其他编程语言生成随机数，构造合法的超长输入，测试大数据对程序的影响。

二、第二次作业

1、程序主体

第二次作业是模拟不会顺路捎带的电梯，通过读入用户指令，判断用户指令的合法性，按请求时间依次进行处理。这次作业中最重点的算法是对于相同请求的忽略，即当电梯还未到达时，按同一个按钮多次将被视为同一个操作，只响应一次。由于没有想到很精妙的算法，因此我采用的是将已处理指令的指令内容和结束时间存储到链表的方法。每次读取一个新指令，先和已有链表进行对比，如果发现是相同指令，则进行忽略。每处理完一个指令后，更新电梯所在楼层及当前时间。

各类大致说明：

Controller 主类：程序 main 函数所在类，此类无专属方法，主要起到调用其他各类，及在其他各类间传递信息的用处。

Elevator 类：用来计算当前读入指令受上一指令影响，判断是否为相同指令，根据当前楼层与目标楼层楼层对比，时间对比，输出相应的输出信息。

Floor 类：由于考虑不周而未被使用的类，由于强制要求使用楼层类，因此将结束语放了进去。

Jiegou 类：用于存储结构体，将请求的发出时间、目标楼层、内外指令、运动方向存储至一个结构体变量中，方便构成链表。

Queue 类：队列类，用来存储请求队列，由 QueueRequest 类每次调用 geton 方法，读取队列中队首元素。

QueueRequest 类：请求或取类，每次获取队列中队首元素，并经由 controller 类传递给电梯类进行处理。

Request 类：从控制台读取指令请求，判断请求的格式合法性，若合法，将合法请求传递至 Queue 类中存储

2、程序结构度量

Controller 主类：共 35 行，无属性与方法，全部为各类实例化及类间元素的

调度。

Elevator 类：共 66 行，五个属性，四个方法。其中 calculate 方法占 40 行，主要用于相同指令的判定与输出指令的格式操作。sub 方法占 7 行，用于返回两个数的差的绝对值。

Floor 类：共 5 行，一个方法没有属性，用于输出结束语。

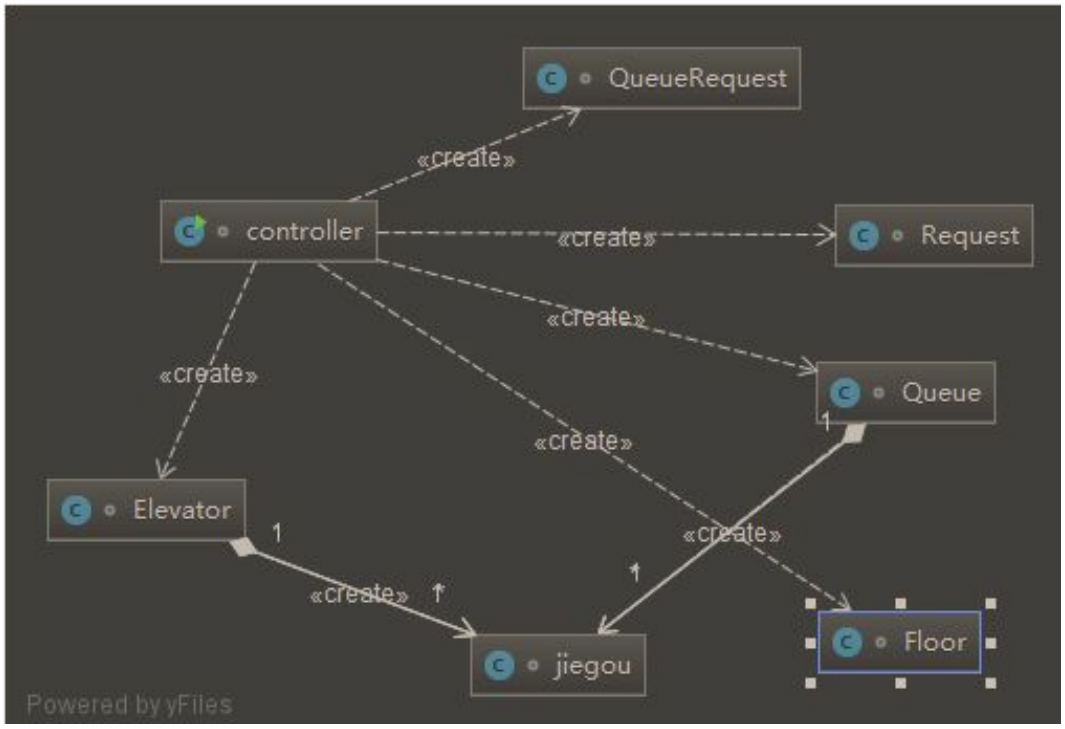
Jiegou 类：共 30 行，两个构造方法，四个方法，四个属性，方法均为 get 型提取私有属性方法，构造方法用以改变不再改变的私有属性值。

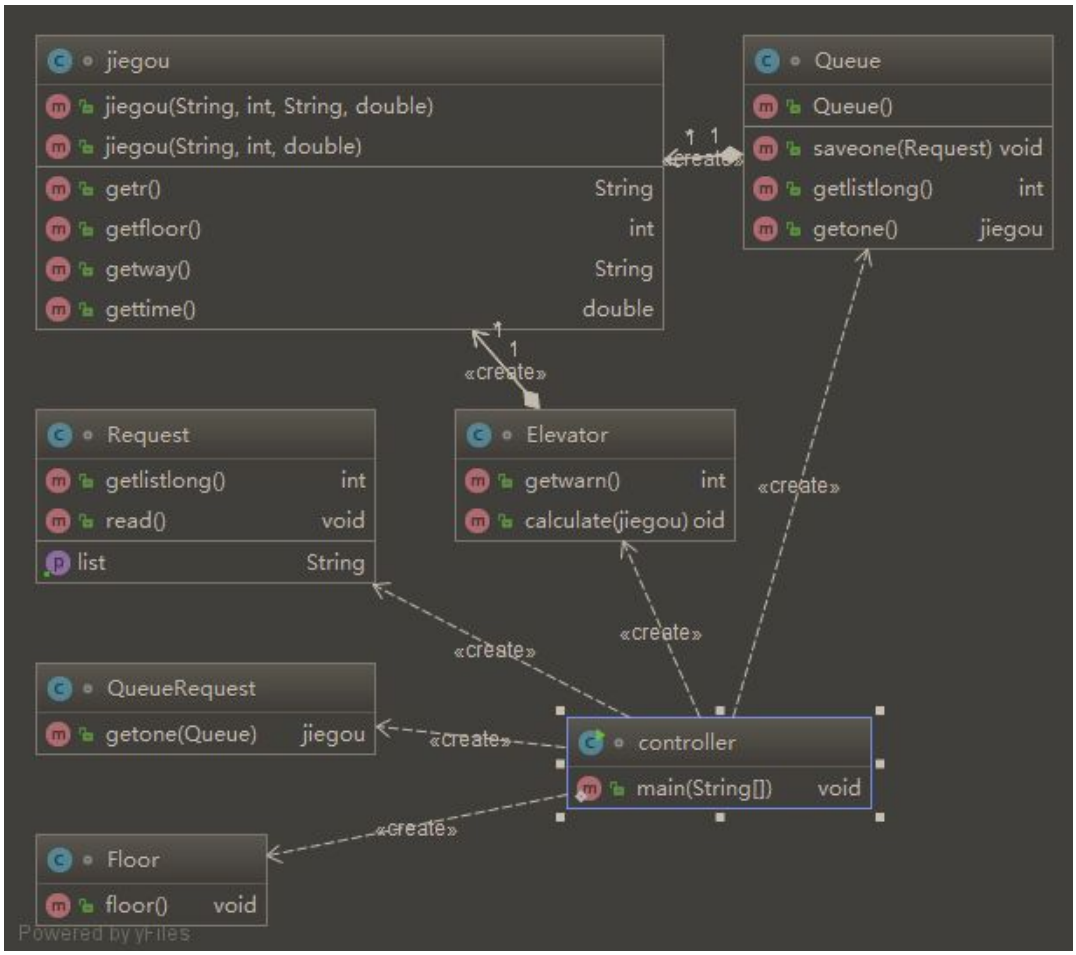
Queue 类：共 40 行，一个构造方法，三个方法，三个属性，其中 saveone 方法用以存储一个 jiegou 类型的节点，占去 20 行。

QueueRequest 类：共 5 行，一个方法没有属性，用以读取 Queue 队首元素。

Request 类：共 140 行，五个方法六个属性，其中 read 方法用于外部输入并判断输入有效性，共占用 105 行。

作业类图：





优点：无 public 属性，按照要求细化成了六个主体类，将取指令、建队列等工作与读取指令、输出结果分开。构建了专门的结构体类，用于方便链节存储。

缺点：类内仍存在“主次关系”，即有的方法篇幅长，调用多，完成工作过多。

3、分析自己程序的 bug

本次作业完成较好，不存在致命硬伤，但是对于相同指令的忽略，只做到了忽略，而没有进行提醒，想当然的将用户能得到的信息由程序内部处理掉。对于输入中存在+1 的情况未考虑。尝试学习使用了正则表达式，可以一行输入很多指令，但是无法对括号间的逗号进行有效识别，对公测样例中出现逗号的情况提示输入有误。

4、自己发现别人 bug 的策略

依然是构造极限测试集和长测试集的方法，测试后没有明显错误，因此通过

短指令，逐步调试对方输入部分正则表达式的判定。发现对方 split 方法使用误区，通过添加 advice 的方式给予对方提示。同时观察到对方因为过于依赖状态机，而使程序一次只能添加一行指令，将自己运用正则表达式一次读取一行的策略通过总评分享给对方，以使对方可以免于囿于单行输入而使公共测试难以测试的问题。

三、第三次作业

1、程序主体

第三次作业是模拟会顺路捎带的电梯，通过读入用户指令，判断用户指令的合法性，按请求时间依次进行处理。这次作业中最重点的算法是对于捎带请求的合理捎带，即当电梯还未到达目标楼层时，若中途存在可捎带指令，在不违反电梯运动状态的前提下，可以允以捎带。我采用的是先将所有指令存储至一个链表中，再重头每次读取一个指令，判断所有主指令可捎带指令，按楼层刷新当前电梯状态和指令状态，若可捎带，则进行捎带，同时将不合法捎带指令从捎带队列中剔除，之后更新主指令。

各类大致说明：

Controller 主类：程序 main 函数所在类，此类无专属方法，主要起到调用其他各类，及在其他各类间传递信息的用处。

Scheduler 类：继承了 elevator 类，添加了捎带指令方法。

Elevator 类：用来计算当前读入指令受上一指令影响，判断是否为相同指令，根据当前楼层与目标楼层楼层对比，时间对比，输出相应的输出信息。

Floor 类：由于考虑不周而未被使用的类，由于强制要求使用楼层类，因此将结束语放了进去。

Jiegou 类：用于存储结构体，将请求的发出时间、目标楼层、内外指令、运动方向存储至一个结构体变量中，方便构成链表。

Queue 类：队列类，用来存储请求队列，由 QueueRequest 类每次调用 geton 方法，读取队列中队首元素。

QueueRequest 类：请求或取类，每次获取队列中队首元素，并经由 controller 类传递给电梯类进行处理。

Request 类：从控制台读取指令请求，判断请求的格式合法性，若合法，将合法请求传递至 Queue 类中存储。

2、程序结构度量

Controller 主类：共 35 行，无属性与方法，全部为各类实例化及类间元素的

调度。

Scheduler 类：共 215 行，九个属性，八个方法（不包括继承的属性及方法），其中 calculate 方法被重写，加入了关于捎带指令的遍历，新的已处理指令存储。

Elevator 类：共 125 行，五个属性，十一个方法。作业二中的 calculate 方法被拆分，细化成为五个新方法，其中最长的方法为 samejudge 方法，共 25 行。

Floor 类：共 5 行，一个方法没有属性，用于输出结束语。

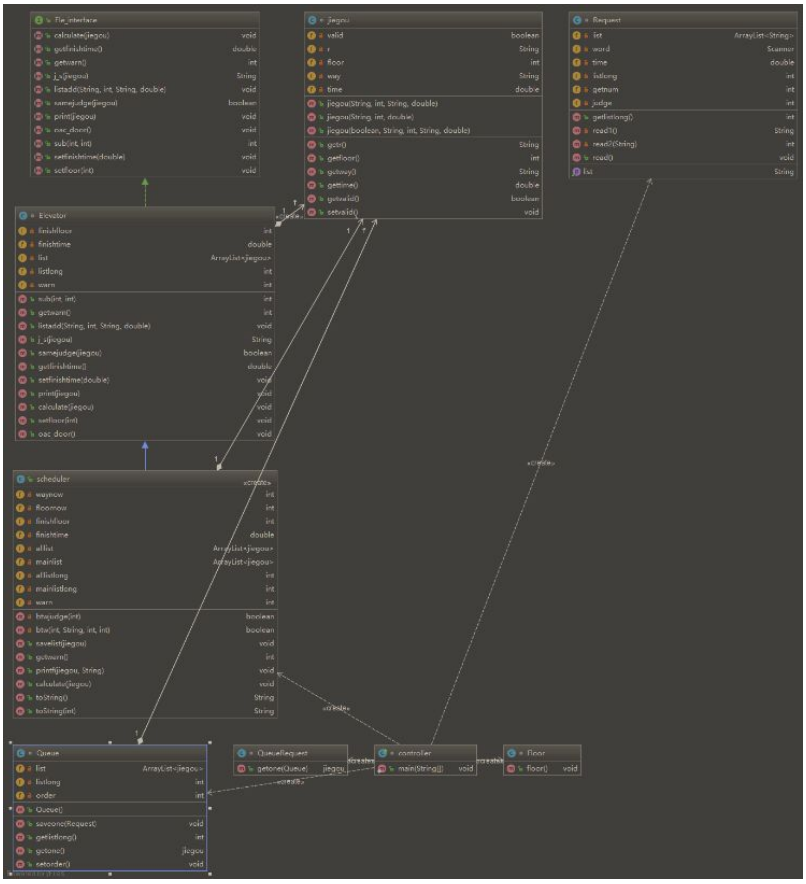
Jiegou 类：共 30 行，两个构造方法，四个方法，五个属性，方法均为 get 型提取私有属性方法，构造方法用以改变不再改变的私有属性值。添加了新的属性 valid，用来存储指令是否已经被处理。

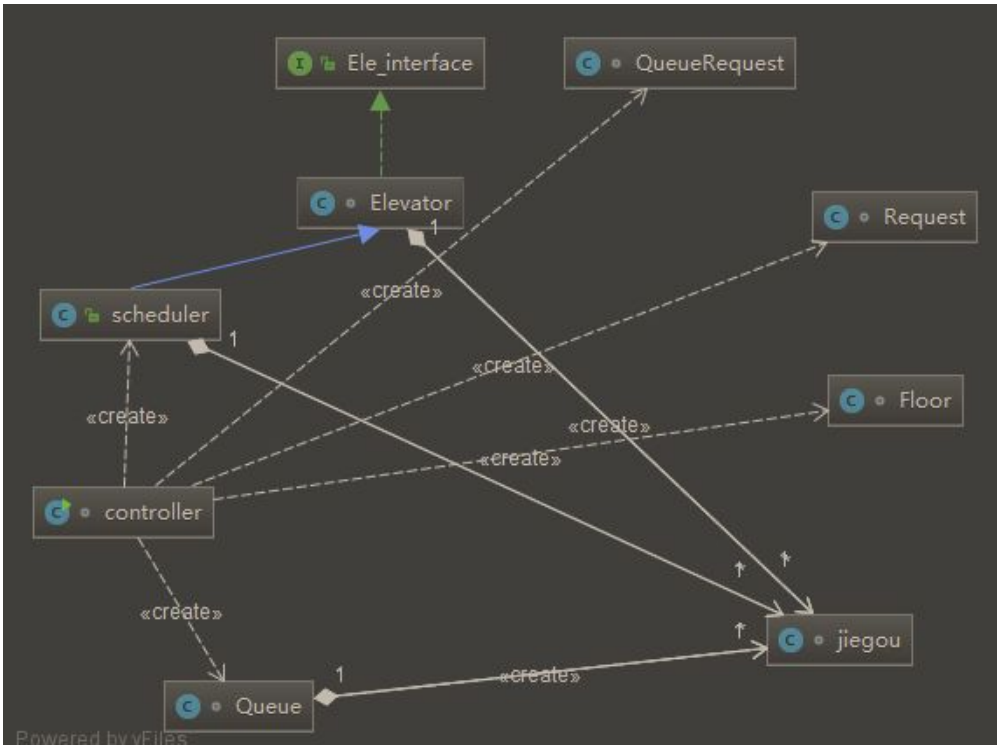
Queue 类：共 40 行，一个构造方法，三个方法，三个属性，其中 saveone 方法用以存储一个 jiegou 类型的节点，占去 20 行。

QueueRequest 类：共 5 行，一个方法没有属性，用以读取 Queue 队首元素。

Request 类：共 140 行，五个方法六个属性，其中 read 方法用于外部输入并判断输入有效性，共占用 105 行。

作业类图：





优点：类内方法更加细化，很多方法具有单一性，简单性。

缺点：类之间调用的仍较少，还需要很多变量在类之间传递。算法复杂度较高，需要优化。按楼层刷新不适合后期多线程处理。

3、分析自己程序的 bug

本次作业完成较好，捎带指令都能完成，但是对于无效指令的输出提示仍然不足，进提示输入无效而不提示哪些指令无效。考虑使用 split 指令进行替换和分割，之后抹去所有正确指令后便可以输出错误的指令。

4、自己发现别人 bug 的策略

通过公共测试集进行测试，发现了他关于楼层捎带时候的一些漏洞，由此及彼，观察它的代码，发现了捎带时多次捎带导致时间不对，上下方向指令导致不进行捎带等问题。

四、心得体会

通过前几次作业，我发现我对于 java 的理解和使用仍存在一些漏洞，但是我认为我可以利用博客、论坛等地方，积极学习具体的函数和特殊方法。我认为经常在课程中心讨论区、博客区进行学习和交流很有必要，因为很多时候很多极限例子需要大家的集思广益，来弥补自己思考的误区，完善自己的程序。我认为 OO 这门课，除了在课堂上从老师那里学习，更重要的是课前的预习和课下的自学，而自学，远不是自己一个人闷头学，更应该和别人交流，从别人那里学习好的算法，发现别人的问题，然后再自我提醒。我想，这才是互测的最终目的，即实现共同的进步，而非为了一点点的分数，让同学们互相争吵与指责。