OO 博客作业

——15061125 赵泽

一、第五次作业

1、程序主体

第五次作业是第三次电梯作业。与之前两次电梯相比，这次作业最大的改变点在于电梯的数量由一台变成了三台。由于引入了多线程，所以出现了抢单时的冲突问题。为了解决并模拟三台电梯同时的运动，我将三台电梯放进了三个不同线程中，同时增加了调度器类，调度器位于另一线程，Request 类不断地读取外部输入的请求，将请求发送给调度器，调度器不断地重复扫描摊派指令动作，若某电梯可完成捎带，或某电梯当前状态为空闲，则将指令发送给相应电梯线程。并从调度器线程的列表中删除相应的指令。

各类大致说明：

Controller 类：程序主类，用来开启线程，并统一实例化各类并把实例化对象传递给相应有需要的类。最后起到关闭所有线程的作用，通过传递变量“shut”的值给各线程，结束各线程任务。

Elevator 类：独立的电梯线程，不断刷新判断当前电梯线程有没有被摊派的任务，依照捎带原理，不断地执行已摊派任务，并不断更新当前状态，方便调度器将新的指令分给该电梯线程。

Floor 类：由于考虑不周未被使用，用于输出结束语

Jiegou 类：用于提供整个程序所需要使用的多种结构体

Queue 类：保存由 Request 类读取到的合法指令，等待调度器类调度

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。

Scheduler_old 类：引用的第三次作业的调度器，可完成单电梯的捎带判断。

Scheduler 类:作为独立线程出现，不断扫描，判断队列中任务是否可以摊派给某台电梯。摊派方法是按照 FR 指令：相同指令>某台电梯可以顺路捎带>某些电梯停了来摊派，ER 指令直接摊派给相应电梯。

Tray 类：托盘类，通过给其他线程加锁来达到指令冲突不互斥的效果。

2、程序结构度量

Controller 类：共 80 行，两个属性，一个方法，其中实例化对象并传递到相应类，开始线程占用约 40 行，关闭所有线程操作占用约 15 行。

Elevator 类：共 428 行，十六个属性，二十个方法，其中模拟电梯运动大致占用 140 行，相同指令判断占用约 25 行，顺路捎带判断占用约 50 行。

Floor 类：共 5 行，一个方法没有属性，用于输出结束语。

Jiegou 类：共 60 行，七种属性，三种构造方法，九种方法，每种方法都很简单，最多不超过五行。

Queue 类：共 50 行，两种属性，一种构造方法，六种方法，保存 Request 类读取到的新指令占用约 20 行。

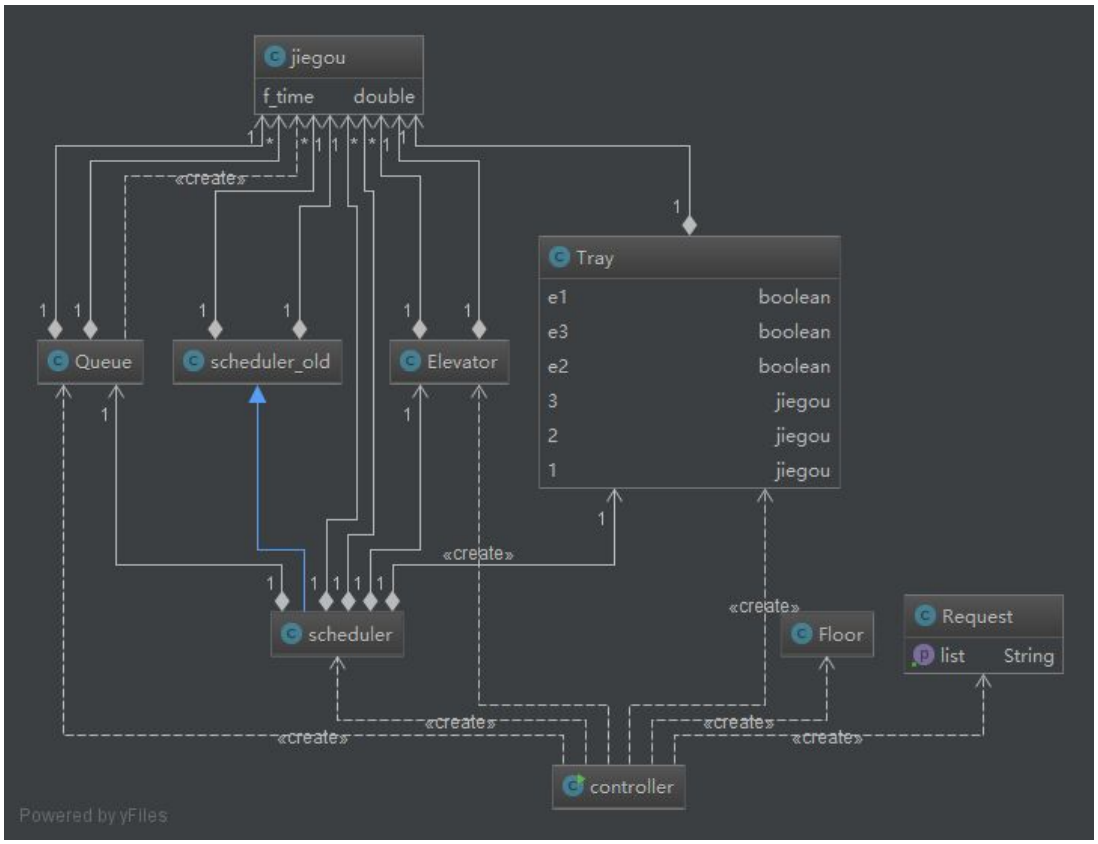
Request 类：共 195 行，六个方法六个属性，其中 read 方法用于外部输入并判断输入有效性，共占用 160 行。

Scheduler_old 类：共 195 行，七个属性，六个方法，其中 calculate 方法占用约 90 行，顺路捎带判断占用约 35 行。

Scheduler 类：共 210 行，七个属性，九个方法，其中摊派操作约占用 120 行。

Tray 类：共 105 行，六种属性，十种方法，大部分方法都是三个一组出现。

作业类图：



优点：程序运行伴随着控制台输出，可以很清晰的看出当前电梯状态，方便测试时获取当前时间和电梯状态，方便测试。类之间的交互很多，各司其职，不会发生严重的冲撞事故。

缺点：有的类经过几次的电梯修改，过于冗杂而难以重构，导致了某个方法占用的行数极其多但是又不好修改的窘境。

3、分析自己程序的 bug

在自行测试的时候，我发现如果调度器将某指令，比方说(FR,5,UP)发送给 1 号电梯后，由于电梯线程不是立刻进行处理，而是在经过一个短暂的 sleep 操作后才进行主请求分析和捎带分析。这造成了若同时发送另一个(FR,5,UP)请求，电梯 1 状态为有请求，但是主请求仍未更新。则最终另一个(FR,5,UP)请求将分配给 2 号电梯。为了解决这一问题，引入了变量

在互测时，对方通过卡时间，找出了我程序中的问题，即模拟延迟的 3s，6s，

是在程序执行判断等操作后再进行的。这样一来，在相同情况下，线程若执行的操作较少，则整体完成时间也较快。互测时对方就是通过这一点，构造了如下的指令输入：

(ER,#1,5) (ER,#1,3) (ER,#2,7) (ER,#3,7) (ER,#3,10) (FR,10,UP)

详细分析：(ER,#1,5) (ER,#1,3) 分配给 1 号电梯；(ER,#2,7) 分配给 2 号电梯；(ER,#3,7) (ER,#3,10)分配给 3 号电梯。(FR,10,UP)不断进行判断，等待分配。

到 24s 时，1 号电梯到达 5 层并开关门结束。2 号电梯到达 7 层并开关门结束。3 号电梯到达 7 层并开关门结束，此时 3 号电梯的主请求应该变更为(ER,#3,10)，可对(FR,10,UP)指令进行捎带。但是由于我的程序存在计算时间的误差，所以最终结果是由 1 号电梯完成(FR,10,UP)指令。

4、自己发现别人 bug 采取的策略

这次作业我分到的是一个很出色的同学的作业。我通过构造极端情况；构造一般的篇幅较短的简单测试样例；通过其他编程语言生成随机数，构造合法的超长输入。种种方法，结果发现他的程序都没有出现严重问题。只是在随机指派电梯时存在些许随机性问题。

二、第六次作业

1、程序主体

第六次作业是文件及文件夹的监控与管理。用户可以选择监控 5-8 个路径，监控后若该路径或其子路径文件发生了大小变化、重命名、路径移动等变化，则按照监控时的输入，相应的触发触发器，按触发器结果完成相应触发。我将每一个新的监控变成一个新的线程，很多个线程不断的扫描，如果有满足监控条件的触发情况，则完成触发。

各类大致说明：

Main 主类：程序 main 函数所在类，主要起到调用其他各类，及在其他各类间传递信息的用处。

Out 类：用于给测试者提供文件操作的接口

Readin 类：用于读入监控的内容，正则匹配后开启相应线程

SafeFile 类：整合了所有涉及到的文件及文件夹操作的方法，将这些方法分别加锁，及不允许同时调用两个删除或两个添加，防止线程冲突。

Syntagm 类：用于提供程序中所有用到的结构体。

Tree 类：用文件树的形式存储文件，并按照相应的触发器，完成相应的触发操作。

2、程序结构度量

Main 主类：共 70 行，六个属性，四个方法，大部分均为给各类实例化及类间元素的调度。

Out 类：共 210 行，三个属性，十三个方法，controller 方法占用约 20 行，每种操作都有相应的方法支持，每种相应的方法均为 5-10 行。

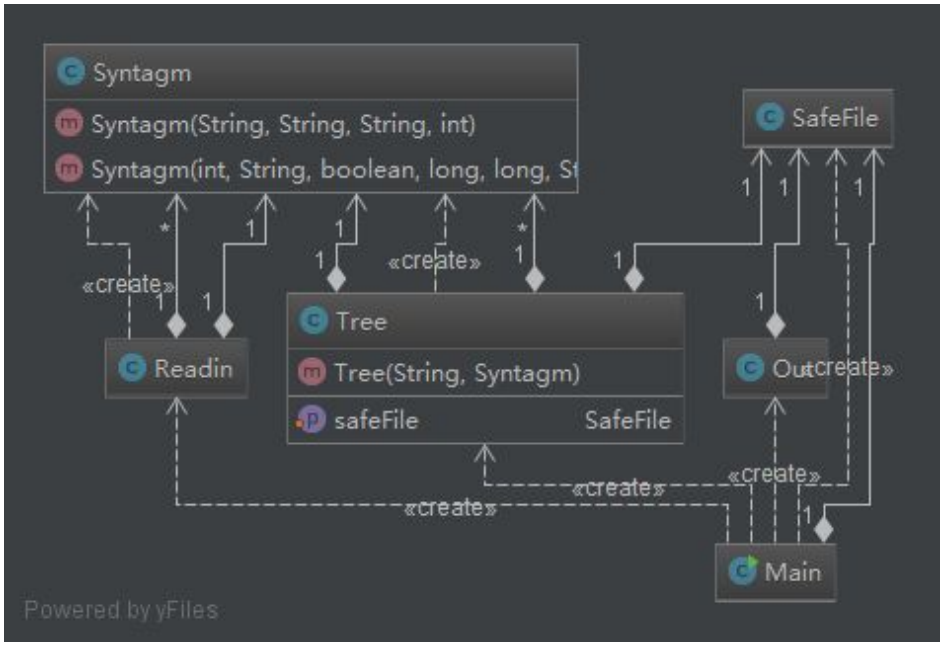
Readin 类：共 105 行，三个属性，六个方法，read 方法占用约 50 行。

SafeFile 类：共 280 行，一个属性，九个方法，模块化，每个方法都不超过 25 行，且彼此独立。

Syntagm 类：共 80 行，九个属性，两个构造方法，十五个方法，方法大多为一两行的 get 和 set 方法。

Tree 类：共 340 行，十五个属性，一个构造方法，十一个方法，四个触发器中 renamed 占用约 60 行，modified 占用约 30 行，path-changed 占用约 30 行，size-changed 占用约 80 行。

作业类图：



优点：制作了一个伪 GUI 的用户输入提示，增加了程序的可使用性。伪 GUI 大致如下：

```
<已终止> Main [Java 应用程序] C:\Program Files\Java\jdk1.8.0_121\bin\javaw.exe ( 2017年
--------------------------Begin Work--------------------------
IF[E:\OO\test] size-changed THEN record-detail
Add Succeed [IF[E:\OO\test] size-changed THEN record-detail]
run
thread opened
--------------------可---执---行---操---作---------------------
1       新建文件                    2       删除文件
3       新建文件夹                  4       删除文件夹
5       查看文件/文件夹属性         6       文件重命名
7       文件移动                    8       写入内容到文件
9       退出命令输入，停止所有监控
------------------------------------------------------------
请输入您想执行的操作：
1
请输入新建文件的路径及文件名：
E:\OO\test\123.txt
创建成功
--------------------可---执---行---操---作---------------------
1       新建文件                    2       删除文件
3       新建文件夹                  4       删除文件夹
5       查看文件/文件夹属性         6       文件重命名
7       文件移动                    8       写入内容到文件
9       退出命令输入，停止所有监控
------------------------------------------------------------
请输入您想执行的操作：
9
外部输入已终止！
--------------------------End Work--------------------------
```

缺点：细节考虑缺失，文件树的构建存在弊端。

3、分析自己程序的 bug

本次作业由于时间分配有误，因此是匆匆完成，没有进行自我的细节测试。导致程序出现了很多致命 bug。比如当我监控一个文件的时候，若文件移动路径或重命名或被删除，我并没有判断文件是否存在，则监控线程会 crash。在之后的作业里，为了防止有类似疏忽，所有的线程的 run 方法我都用 try，catch 将其包括起来。

4、自己发现别人 bug 的策略

构造了简单的例子来测试对方的 bug，发现他和我一样有同样的文件跟踪的错误存在。后来在课上荣老师告诉我们，可以通过程序，构造一个深度为 20000 的目录，或者构造一个宽度为 20000 的目录。我觉得通过撰写程序的方法确实可以很快、很极端的来测试别人程序的性能。

三、第七次作业

1、程序主体

第七次作业是在一个 80*80 的地图上，模拟 100 台出租车的抢单接客情景。我采用的是每台出租车一个线程，每次通过用刷新周期减去当前时刻到上一刷新周期的时间的差值，来进行 sleep 等待，让每台出租车接近“同一个钟”的状态。若一个叫车指令发出尚未达 3 秒，则每 100ms 扫描一次周围 4*4 格子中有没有符合条件的出租车。若达到 3 秒，则从叫车的点开始向周围进行广度优先遍历，找到第一个 mark 最大的且可以接客的出租车，完成摊派。通过逆序，将出租车到达叫车点的路径存储到出租车任务队列中。出租车接到客人后，从出租车所在点进行广度优先遍历，仍然通过逆序得到前往终点的路径。

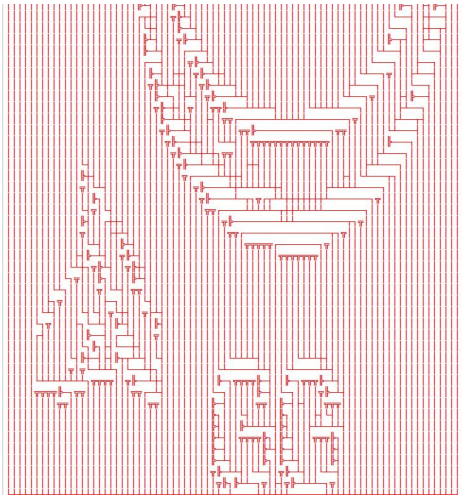
各类大致说明：

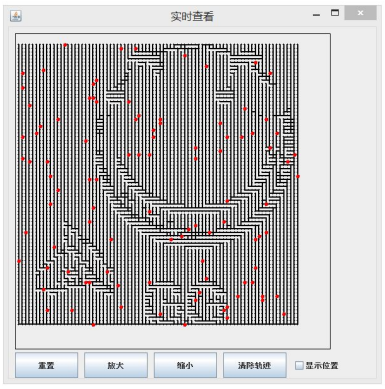
Main 主类：程序 main 函数所在类，此类无专属方法，主要起到调用其他各类，及在其他各类间传递信息的用处。

Controller 类：调度器，可完成查询出租车状态、查询某状态所有出租车、某单的扫描、某单的摊派等任务。

Driver 类：司机类，每台出租车一个线程。用于模拟每一台出租车的运动，写法类似于状态机，按照要求相应的进行状态转换。

Map 类：地图类，用于存储输入的地图，并判断是否合法（是否出界及是否为连通图）。并且在控制台输出我自行设计的按要求绘制的地图，效果如图：左侧为我画的静态图，右侧为 GUI 实际图。





People 类：模拟用户下单。不断读入外部输入，将查询或叫车请求发送给调度器类。

Random 类：辅助类，可提供两种操作，①传入一个 n，返回一个范围在[1,n]的随机数。此处使用的是 Math 中自带的随机数方法。②补全时间，可补全时间有 100ms，200ms 和 1s，在操作之前设置一次系统时间，操作完毕要睡眠时进行差值计算，计算结果即为睡眠时间。

Readin 类：读取外部地图文件，判断合法性后将地图发给地图类。

Syntagm 类：用于存储不同的结构体，可存储的大概有如下几类。

①存储叫车请求（出发点，目的地，时间）

②存储可抢单的出租车（位置，出租车号）

③存储广度优先遍历中已遍历过结点（位置，权值）

④存储司机在广度优先遍历过程中的中间位置（位置）

⑤存储查询指令的查询内容（查询内容及编号，查询结果存储路径）

2、程序结构度量

Main 主类：共 110 行，无属性与方法，全部为各类实例化及类间元素的调度。

Controller 类：共 460 行，八个属性，是一个方法，其中广度优先算法占用约 110 行，指令摊派处理方法占用 200 行。

Driver 类：共 470 行，十四个属性，一个构造方法，十六个方法。其中广度优先算法占用约 130 行。出租车状态变化及运动方法占用约 250 行。

Map 类：共 190 行，三个属性，一个构造方法，十个方法。其中绘制地图占用约 40 行，初始化设置地图占用约 100 行。

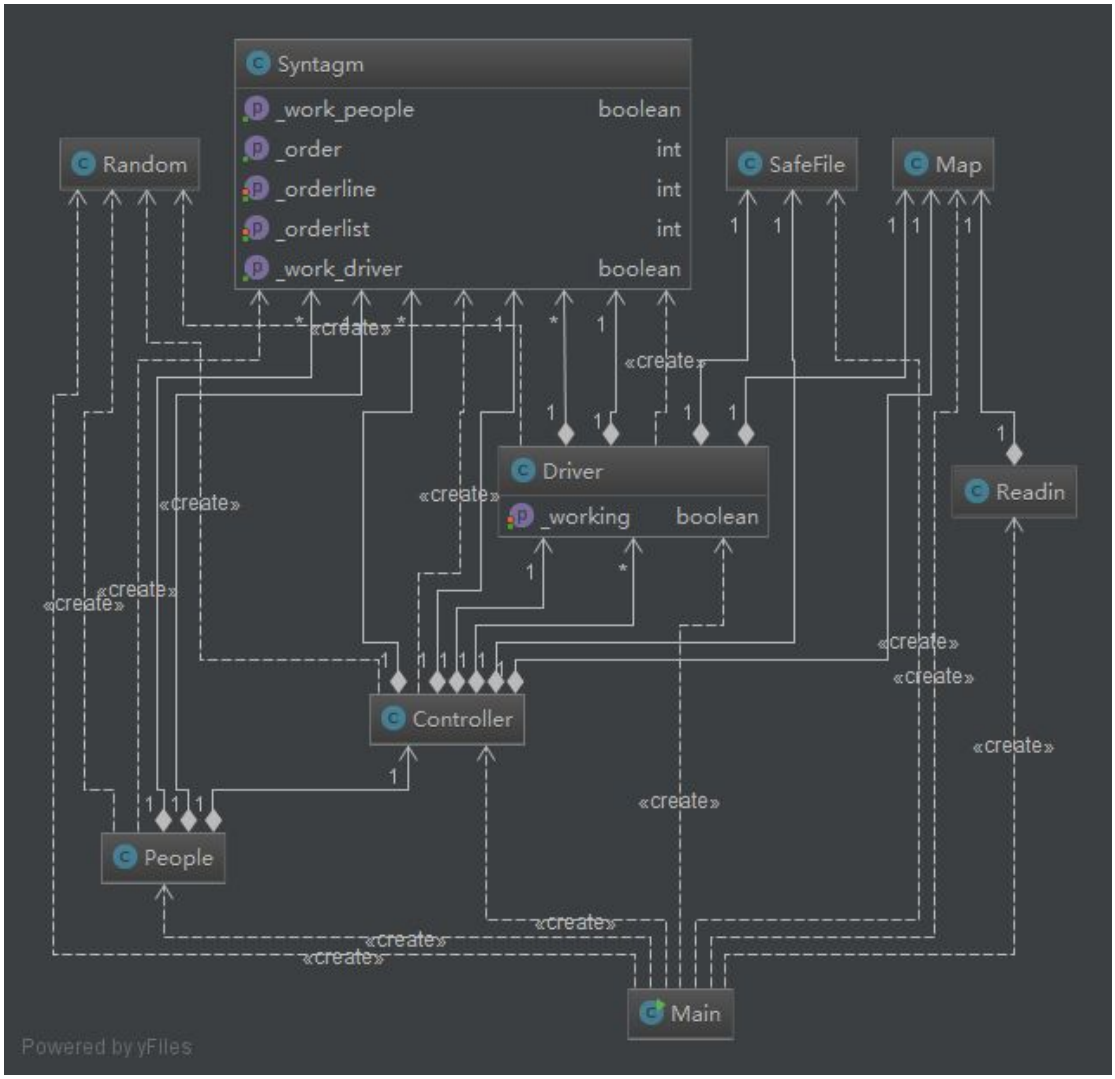
People 类：共 160 行，七个属性，四个方法。其中 run 方法占用约 130 行。

Random 类：共 35 行，两个属性，五个方法。其中非方法均不超过 5 行。

Readin 类：共 115 行，两个属性，三个方法。其中从文件读入占用约 95 行。

Syntagm 类：共 120 行，十六个属性，五个构造方法，二十三个方法。方法大多是一行的 get 或 set。

作业类图：



优点：方法细化，结构清晰

缺点：线程延迟较为严重

3、分析自己程序的 bug

本次作业完成较好，但是由于疏忽，在和课程下发的 GUI 进行衔接的时候，出现了一些衔接性的 bug，比如 GUI 支持的坐标是[0，79],而我的程序支持的则是[1，80],在 GUI 上打点的时候，由于 GUI 不存在 80，因此导致程序出现异常。

4、自己发现别人 bug 的策略

我通过观察与操作，让一些可以接单的出租车去接一些需要跑的时间较长的

单，来观察他最短路径的算法是否正确。和别人讨论我才知道，还可以将所有出租车一开始初始化到 0,0 点，然后在附近添加指令，观察对方的算法延迟导致的错误。

四、心得体会

通过这几次的作业，我对 java 的使用更为了解了。也尝试着将一些比较冗杂的方法，拆分成比较细碎的方法。这让我在修改的时候，十分的便捷，只需要找到相应的子方法然后修改就行，而不需要牵一发而动全身。同时我还尝试使用了一些静态的全局变量，这样在类之间就可以共用某些无需变化且需要同步的量了。

我觉得在撰写一个程序之前，必要的思考是少不了的，这可以让我们在之后的撰写及完善过程中，少走很多弯路。当然必要的实践也不可少，很多问题是隐藏在实际操作之中的，纸上谈兵并不就是最好的，还要搭配尝试和与他人的交换讨论，这样才能真正的完善自己的程序。