面向对象课程第二次总结分析

tags： 面向对象课程文档 Java

线程及其协同、同步设计结构分析

第五次作业

1. 线程对象

- 请求模拟器
- 调度器
- 电梯

2. 共享对象

- 请求队列
- 电梯 或 status board

3. 线程协作关系

- 请求模拟器产生请求并加入请求队列；调度器从请求队列取出请求
- 调度器向电梯请求队列加入请求；电梯从电梯请求队列读取请求
- 调度器通过 status board 或者电梯对象的引用获取电梯状态信息
- 调度器分配请求调度电梯运动

4. 同步关系

- 请求模拟器向请求队列增加请求与调度器取出请求需要同步
- 调度器读取电梯状态与电梯改变自身状态需要同步
- 调度器向电梯请求队列加入请求与电梯从电梯请求队列读取请求需要同步

第六次作业

1. 线程对象

- 获取快照线程（主动）
- 快照diff线程（被动）
- 触发控制线程（被动）
- summary 与 detail

2. 共享对象

- 快照
- 文件
- summary 与 detail

3. 线程协作关系

- 获取快照线程扫描文件建立快照
- 快照diff线程获取新建立的快照并比较差异变化
- 触发控制线程根据差异变化决定是否触发
- 触发控制线程通过 summary 与 detail 将信息写入文件

4. 同步关系

- 获取快照线程建立快照与快照diff线程读取快照需要同步
- 快照diff线程得出快照变化与触发控制线程获取变化需要同步
- 获取快照线程读取文件与测试线程更改文件需要同步
- 触发线程之间通过 summary 与 detail 将信息写入文件需要同步

第七次作业

1. 线程对象

- 出租车
- 出租车系统
- 抢单窗口
- 请求模拟器

2. 共享对象

- 请求队列
- 出租车

3. 线程协作关系

- 请求模拟器产生请求并加入请求队列；出租车系统取出请求并开启抢单窗口
- 抢单窗口获取出租车位置与状态并分配请求

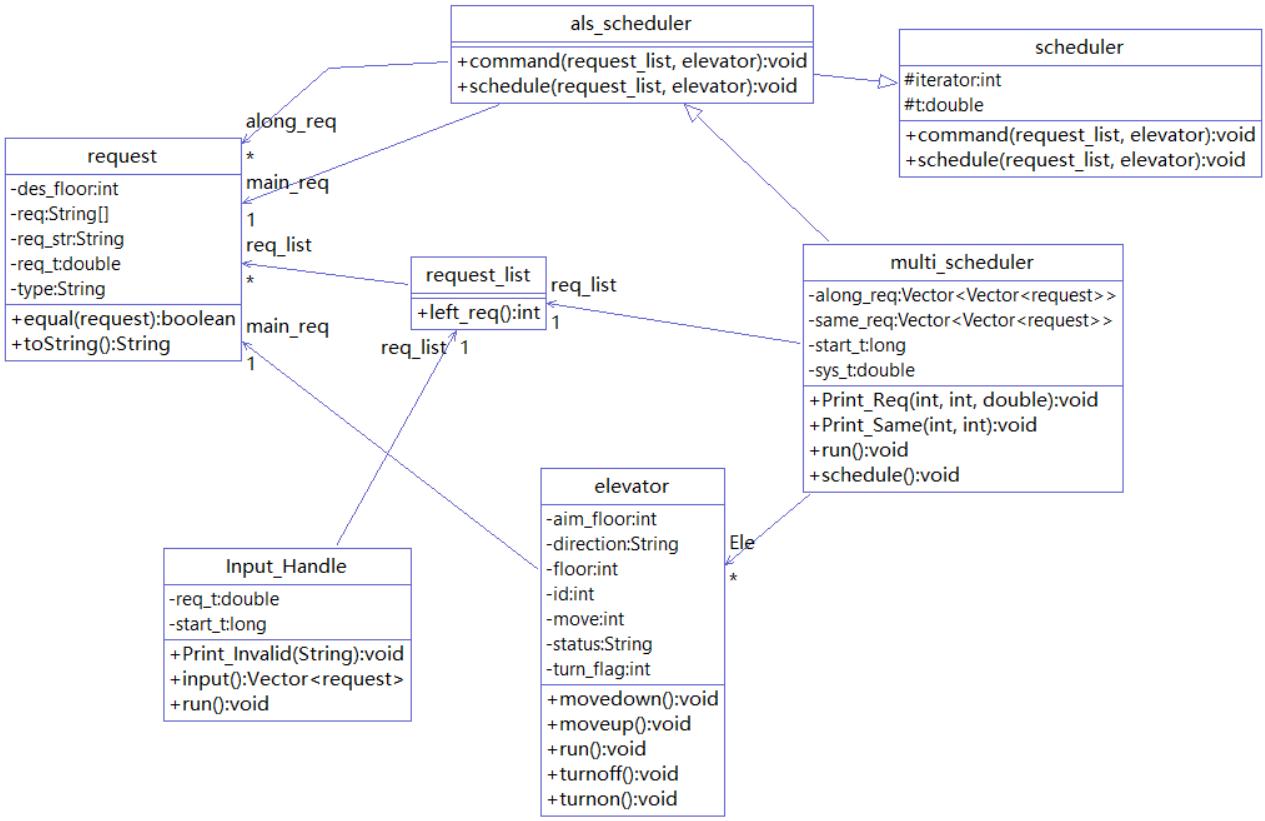
4. 同步关系

- 请求模拟器将请求加入请求队列与出租车系统取出请求需要同步
- 抢单窗口获取出租车状态信息与出租车改变自身状态信息需要同步

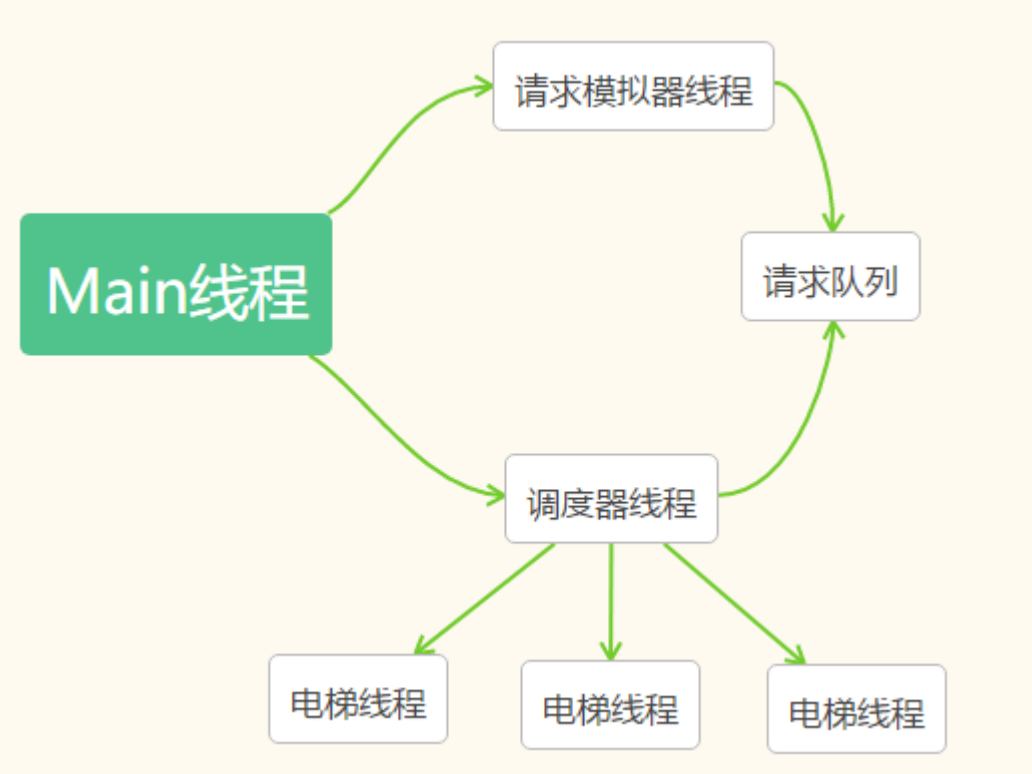
对自己程序的设计结构的分析

第五次作业

UML



线程协作图



度量分析

Input_Handle
属性：4 方法：5 规模：110

方法	规模	控制分支
format_check()	35	7
content_check()	25	8
input()	10	0
Print_Invalid()	8	0
get_time()	2	0

request
属性：5 方法：2 规模：40

方法	规模	控制分支
toString()	1	0
equal()	3	1

request_list
属性：1 方法：4 规模：27

方法	规模	控制分支
add_req()	1	0
remove_req()	1	0
left_req()	1	0
get_req()	1	0

elevator
属性：8 方法：4 规模：115

方法	规模	控制分支
moveup()	8	0
movedown()	8	0
turnon()	5	0
turnoff()	2	0

multi_scheduler
属性：6 方法：4 规模：190

方法	规模	控制分支
shcedule()	110	27
Print_Req()	18	1
Print_Same()	15	1
get_time()	2	0

Ele_Sys
属性：0 方法：2 规模：36

方法	规模	控制分支
createFile()	20	2
main()	10	0

优缺点分析

1. 优点

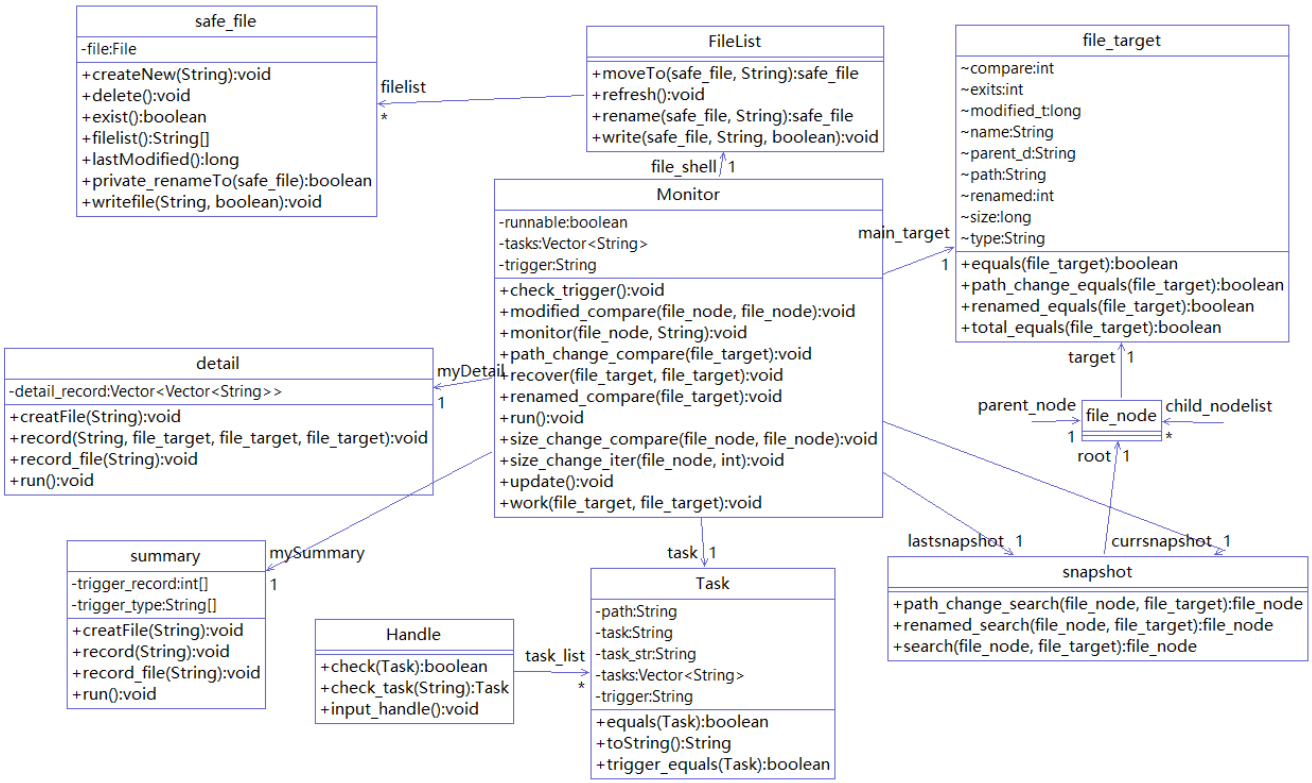
- 请求队列作为托盘，不需要构建单独托盘类
- 控制分支相对之前减少
- 除了 multi_scheduler 类中的 schedule() 方法，其他方法基本都做到了承担单个任务并减少代码量
- run() 方法类只是简单处理和调用函数，代码量很少

2. 缺点

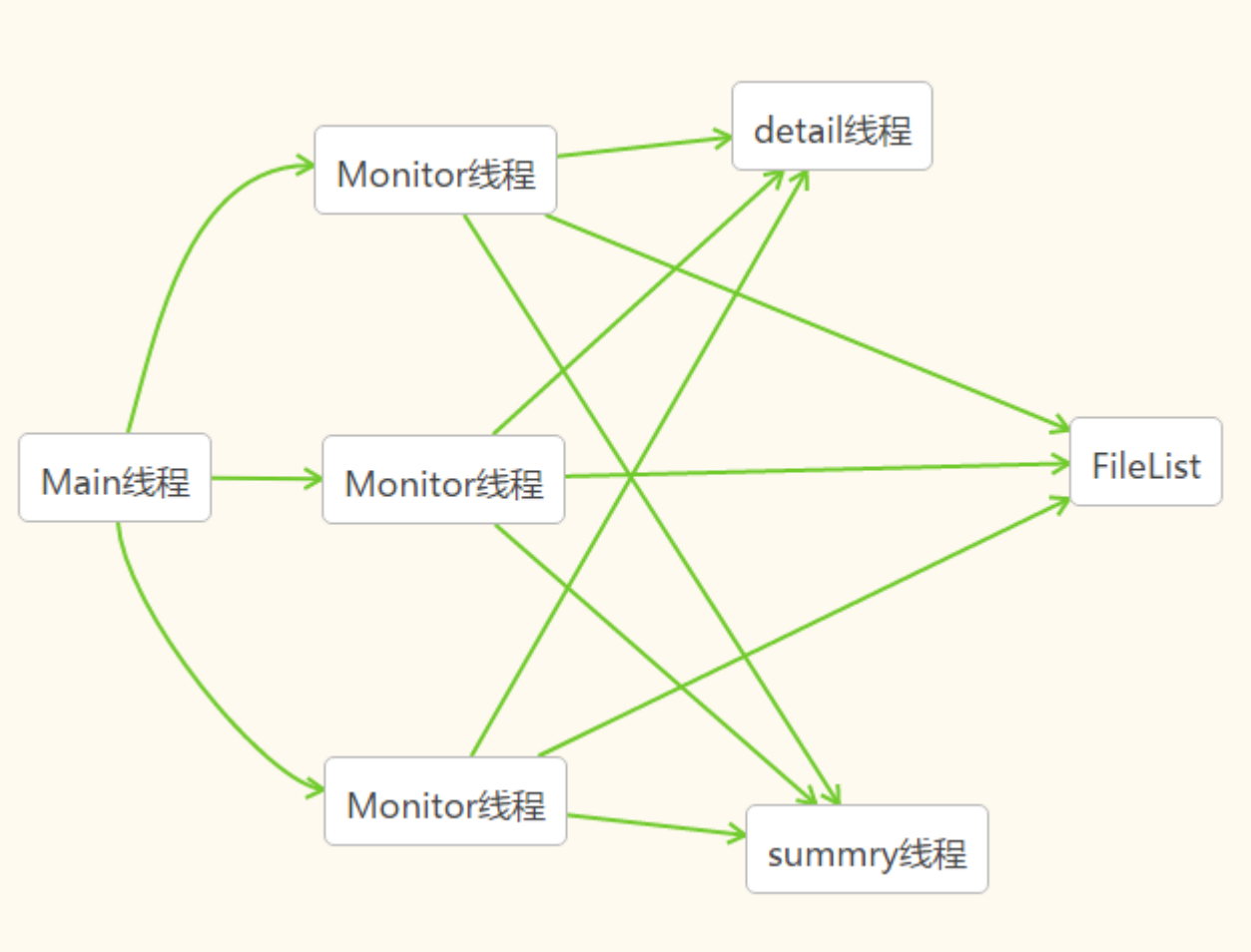
- multi_scheduler 类中的 schedule() 方法成了过长方法，代码较为冗长；承担的任务包括分配电梯任务，判断捎带，电梯到达目标楼层后的输出，承担的任务过多，应该简化
- 子类和父类之间没有交互
- multi_scheduler 中的方法大量使用电梯类中的属性数据

第六次作业

UML



线程协作图



度量分析

Handle
属性：2 方法：4 规模：70

方法	规模	控制分支
input_handle()	15	2
check_task()	30	4
check()	10	2
get_tasklist()	2	0

Task
属性：5 方法：4 规模：45

方法	规模	控制分支
addTask()	1	0
equals()	3	1
trigger_equals()	3	1
toString()	1	0

safe_file
属性：1 方法：14 规模：93

方法	规模	控制分支
writefile()	12	2
delete()	10	2
createNew()	15	5
getfile()	1	0
private_rename_to()	1	0
isDirectory()	1	0
isFile()	1	0
getName()	1	0
getPath()	1	0
getParent()	1	0
lastModified()	1	0
getlength()	1	0
exits()	1	0
filelist()	6	1

FileList
属性：1 方法：8 规模：100

方法	规模	控制分支
get_shell()	12	3
write()	1	0
rename()	20	6
moveTo()	20	6
getfile()	1	0
addFile()	1	0
remove()	1	0
refresh()	10	1

file_target
属性：9 方法：4 规模：45

方法	规模	控制分支
total_equals()	3	1
equals()	3	1
path_change_equals()	3	1
renamed_equals()	3	1

file_node
属性：3 方法：0 规模：10

snapshot
属性：1 方法：4 规模：50

方法	规模	控制分支
add_node()	10	3
search()	10	2
path_change_search()	8	2
renamed_search()	12	3

Monitor
属性：10 方法：10 规模：270

方法	规模	控制分支
monitor()	20	1
check_trigger()	70	20
work()	15	3
recover()	15	2
update()	3	0
modified_compare()	10	2
size_change_compare()	12	2
size_change_iter()	10	2
path_change_compare()	20	5
renamed_compare()	16	5

summary
属性：2 方法：3 规模：50

方法	规模	控制分支
record()	8	1
createFile()	10	2
record_file()	10	0

detail
属性：1 方法：3 规模：90

方法	规模	控制分支
record()	32	6
createFile()	10	2
record_file()	10	0

优缺点分析

1. 优点

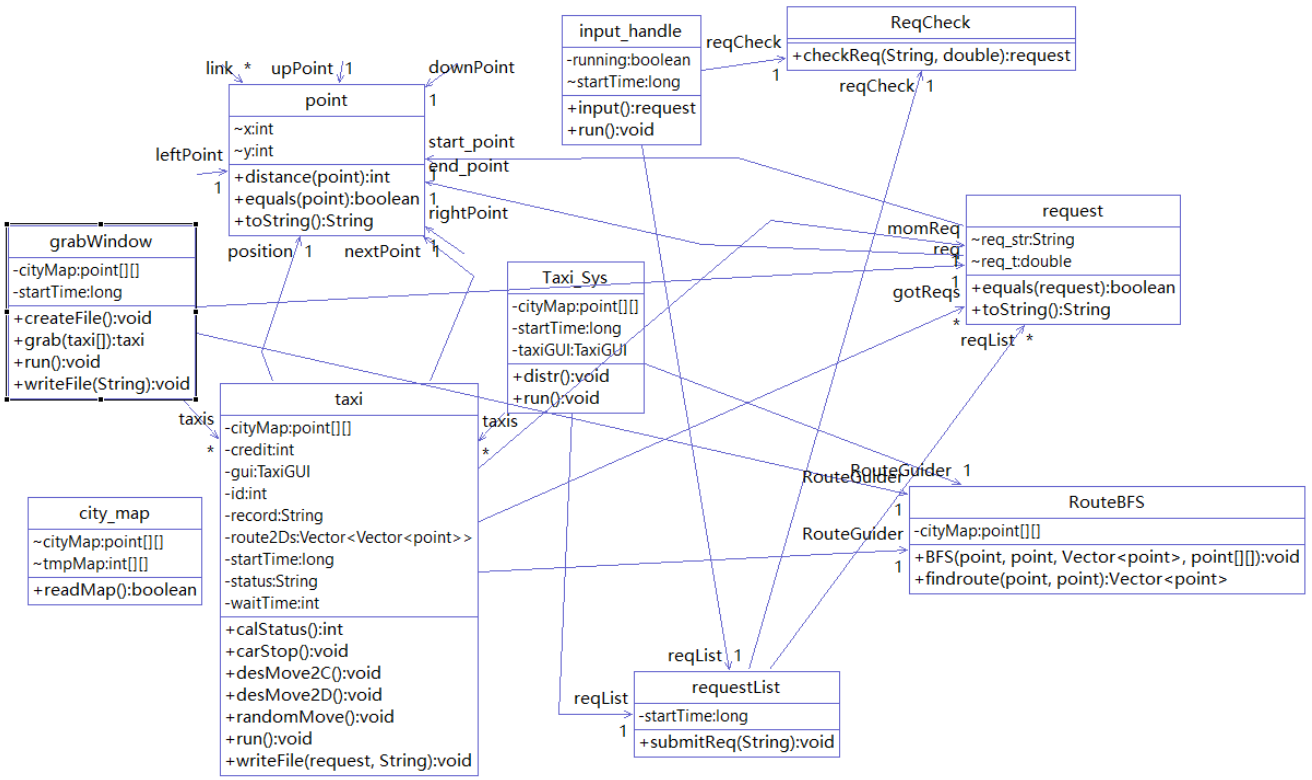
- 除了 Monitor 类的 check_trigger() 方法外，其他方法基本做到了承担单个或两个任务，减少代码量
- run() 方法内基本只是简单处理和调用函数，代码量少
- 通过类似于单例模式的方法管理文件，防止线程间读写文件产生冲突
- 通过构建文件节点树管理快照，并提供各种触发器需要的搜索树的方法

2. 缺点

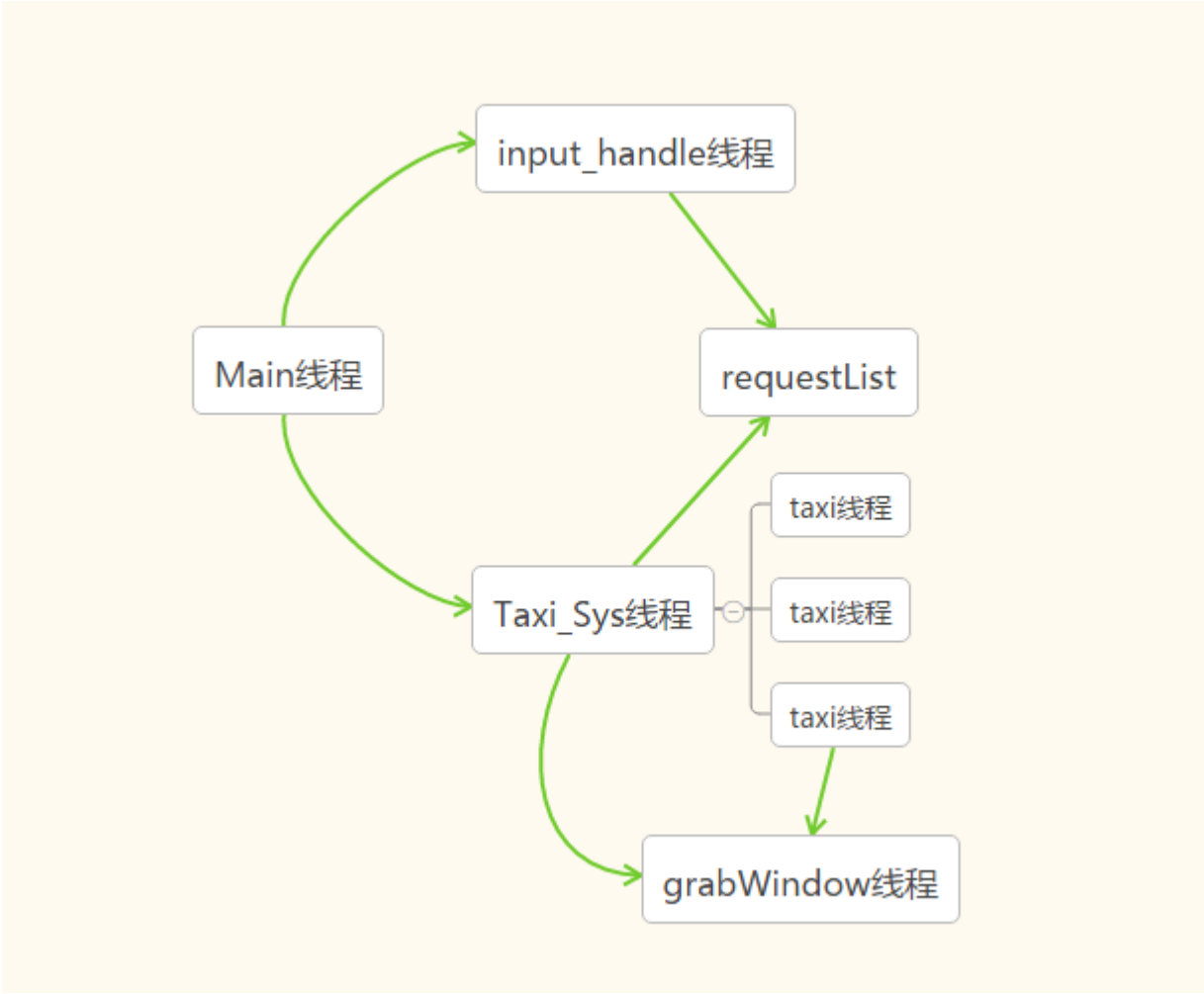
- Monitor 类的 check_trigger() 承担了过多责任，负责了所有触发器的触发检查以及调用处理方法
- Monitor 类负责了所有的触发器，代码量略偏大，没有进行适当抽象形成继承层次
- Monitor 类既要负责触发，又需要负责生成快照和比较快照差异，承担了太多责任
- 提供测试线程的线程安全文件类的方法被分割在两个类中，本可以统一在一个类中

第七次作业

UML



线程协作图



度量分析

city_map
属性：2 方法：4 规模：80

方法	规模	控制分支
readMap()	24	4
drawMap()	12	2
checkLine()	10	3
get_cityMap()	1	0

point
属性：7 方法：5 规模：70

方法	规模	控制分支
setPoint()	35	12
equals()	2	1
distance()	3	1
getNext()	2	1
toString()	1	0

ReqCheck
属性：0 方法：1 规模：25

方法	规模	控制分支
checkReq()	20	3

input_handle
属性：5 方法：2 规模：40

方法	规模	控制分支
input()	4	0
getTime()	3	0

request
属性：4 方法：2 规模：25

方法	规模	控制分支
equals()	3	1
toString()	2	0

requestList
属性：3 方法：4 规模：45

方法	规模	控制分支
getReqList()	5	0
addReq()	5	1
submitReq()	6	2
getTime()	3	0

RouteBFS
属性：1 方法：2 规模：40

方法	规模	控制分支
BFS()	15	2
findroute	12	0

taxi
属性：14 方法：7 规模：200

方法	规模	控制分支
desMove2C()	20	2
desMove2D()	22	2
randomMove()	12	0
carStop()	8	0
calStatus()	5	4
getTime()	3	0
writeFile()	7	0

grabWindow
属性：5 方法：6 规模：120

方法	规模	控制分支
grab()	30	9
getRange()	10	0
getHopeOne()	25	9
getTime()	3	0
createFile()	7	1
writeFile()	7	0

Taxi_Sys
属性：6 方法：4 规模：45

方法	规模	控制分支
distr()	4	0
getTaxiInfor()	2	0
getTaxiAtStatus()	6	2
getTime()	3	0

Taxi_Main
属性：0 方法：1 规模：25

方法	规模	控制分支
main()	20	0

优缺点分析

1. 优点

- 每个方法承担的责任明确，也减少了代码量

2. 缺点

- taxi 类存在大量 get 与 set 方法，代码量较大
- 代码中类主要使用组合，没有抽象出合适的继承层次

对自己程序的**bug**的分析

第五次作业

1. 公共测试集

- 被发现 2 个 bug
- bug 0：电梯到达目的地后都在开门时刻输出；错误原因：对于同层请求，处于 `STILL` 状态的电梯，输出应该在开关门完毕的时刻；错误所在类与方法：`multi_scheduler` 类的 `Print_Req()` 方法
- bug 1：有时存在 0.1s 误差；错误原因：未知

2. 互测

- 没有被发现 bug

第六次作业

1. 公共测试集

- 没有被发现 bug

2. 互测

- 没有被发现 bug
- 自己测试中发现且最终未解决的 bug：当一个文件夹下存在两个除了名字之外其他完全一样的文件，重命名其中一个文件，如果重命名后的文件排列在原有的另一个文件后面，则不会触发重命名触发器；错误所在类与方法：`Monitor` 类的 `check_trigger()` 方法

第七次作业

1. 公共测试集

- 被发现 2 个 bug
- bug 0：对于出发地和目的地相同的请求依然响应；错误所在类与方法：`ReqCheck` 类的 `checkReq` 方法
- bug 1：对于数字前有 `+` 的请求无法响应；错误所在类与方法：`ReqCheck` 类的 `checkReq` 方法

2. 互测

- 没有被发现 bug

对发现他人程序**bug**的策略分析

- 尝试造成资源争夺，查看程序运行情况
- 考察线程之间的同步工作是否完备
- 考察多个请求发出时程序运行情况
- 考察在某些较极端情况下的程序运行情况，如出租车同一地点的多个请求

心得体会

- 设计多线程程序一定要注意线程安全；对于多个线程访问修改的部分一定要以某种形式进行同步
- 生产者消费者模式能够解决生产者和消费者的强耦合问题，在很多情况下十分有效
- 在写代码前要先进行需求分析并进行设计，能够减少不必要的想法，明确目标，提高效率