

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 4实验报告**

|  |  |
| --- | --- |
| 姓名 | 林之浩 |
| 学号 | 1170300817 |
| 班号 | 1703008 |
| 电子邮件 | 630073498@qq.com |
| 手机号码 | 18065053516 |

**目录**

[1 实验目标概述 1](#_Toc9200740)

[2 实验环境配置 1](#_Toc9200741)

[3 实验过程 3](#_Toc9200742)

[3.1 Error and Exception Handling 3](#_Toc9200743)

[3.1.1 输入不合法 3](#_Toc9200744)

[3.1.1.1 TrackGame 4](#_Toc9200745)

[3.1.1.2 Atomstructure 5](#_Toc9200746)

[3.1.1.3 Socialnetwork 5](#_Toc9200747)

[3.1.2 标签一样错误 6](#_Toc9200748)

[3.1.3 依赖关系错误 6](#_Toc9200749)

[3.2 Assertion and Defensive Programming 7](#_Toc9200750)

[3.2.1 checkRep()检查invariants 7](#_Toc9200751)

[3.2.1.1 TrackGame： 7](#_Toc9200752)

[3.2.1.2 SocialNetCircle： 8](#_Toc9200753)

[3.2.1.3 针对各个physicalObject的子类： 8](#_Toc9200754)

[3.2.2 Assertion保障pre-/post-condition 9](#_Toc9200755)

[3.3 Logging 9](#_Toc9200756)

[3.3.1 写日志 9](#_Toc9200757)

[3.3.2 日志查询 10](#_Toc9200758)

[3.4 Testing for Robustness and Correctness 13](#_Toc9200759)

[3.4.1 Testing strategy 13](#_Toc9200760)

[3.4.2 测试用例设计 13](#_Toc9200761)

[3.4.2.1 文件读写过程： 13](#_Toc9200762)

[3.4.2.2 不在文件读入过程发生的参数错误 14](#_Toc9200763)

[3.4.2.3 健壮性测试 15](#_Toc9200764)

[3.4.3 测试运行结果与EclEmma覆盖度报告 15](#_Toc9200765)

[3.5 SpotBugs tool 16](#_Toc9200766)

[3.5.1 readLine 16](#_Toc9200767)

[3.5.2 类名大写开头 17](#_Toc9200768)

[3.5.3 Readline错误 17](#_Toc9200769)

[3.6 Debugging 17](#_Toc9200770)

[3.6.1 理解待调试程序的代码思想 17](#_Toc9200771)

[3.6.1.1 findMedianSortedArrays 17](#_Toc9200772)

[3.6.1.2 removeComments 18](#_Toc9200773)

[3.6.1.3 TopVotedCandidate 18](#_Toc9200774)

[3.6.2 发现并定位错误的过程 18](#_Toc9200775)

[3.6.2.1 findMedianSortedArrays 18](#_Toc9200776)

[3.6.2.2 removeComments 19](#_Toc9200777)

[3.6.2.3 TopVotedCandidate 19](#_Toc9200778)

[3.6.3 如何修正错误 20](#_Toc9200779)

[3.6.3.1 findMedianSortedArrays 20](#_Toc9200780)

[3.6.3.2 removeComments 20](#_Toc9200781)

[3.6.3.3 TopVotedCandidate 21](#_Toc9200782)

[3.6.4 结果 21](#_Toc9200783)

[3.6.4.1 findMedianSortedArrays 21](#_Toc9200784)

[3.6.4.2 removeComments 22](#_Toc9200785)

[3.6.4.3 TopVotedCandidate 23](#_Toc9200786)

[4 实验进度记录 23](#_Toc9200787)

[5 实验过程中遇到的困难与解决途径 23](#_Toc9200788)

[6 实验过程中收获的经验、教训、感想 24](#_Toc9200789)

[6.1 实验过程中收获的经验和教训 24](#_Toc9200790)

[6.2 针对以下方面的感受 24](#_Toc9200791)

# 实验目标概述

本次实验重点训练学生面向健壮性和正确性的编程技能，利用错误和异常处理、断言与防御式编程技术、日志/断点等调试技术、黑盒测试编程技术，使程序可在不同的健壮性/正确性需求下能恰当的处理各种例外与错误情况，在出错后可优雅的退出或继续执行，发现错误之后可有效的定位错误并做出修改。

实验针对Lab 3中写好的ADT代码和基于该ADT的三个应用的代码，使用以下技术进行改造，提高其健壮性和正确性：

错误处理

异常处理

防御式编程

日志

调试技术

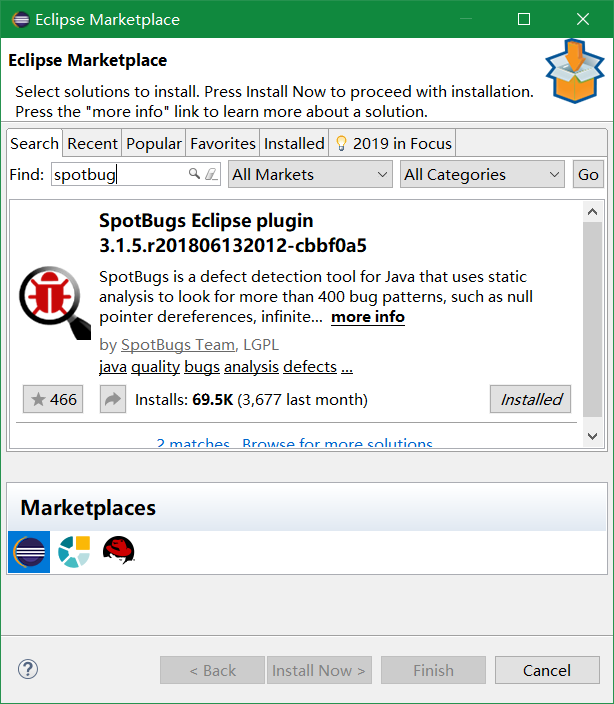
黑盒测试及代码覆盖度

# 实验环境配置

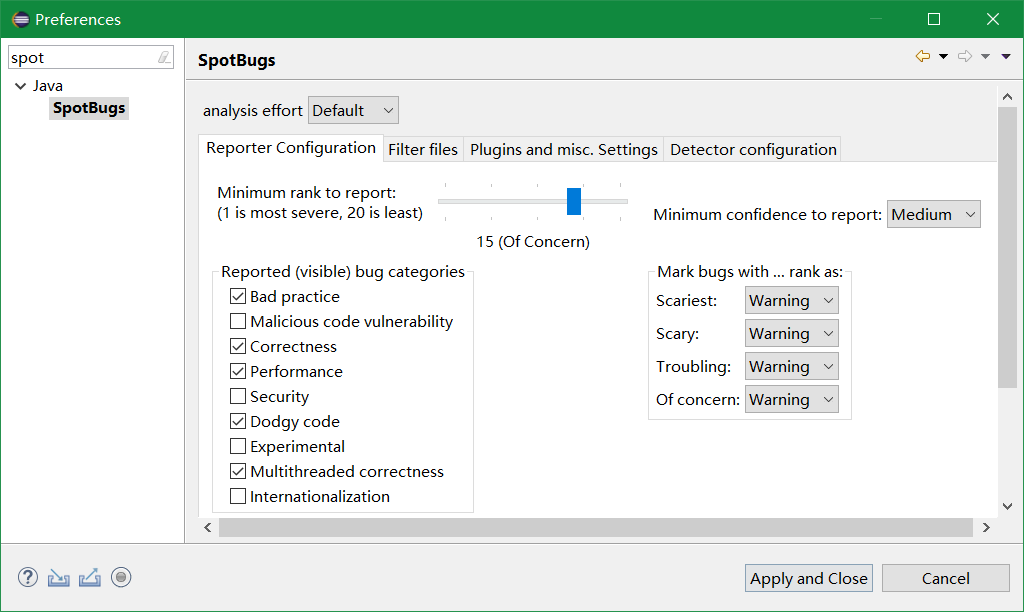
简要陈述你配置本次实验所需环境的过程，必要时可以给出屏幕截图。

特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

直接在eclipse marketplace中搜索spotbugs然后安装。



并且在windows-preference下的spotbugs菜单对其进行配置



在这里给出你的GitHub Lab4仓库的URL地址（Lab4-学号）。

<https://github.com/ComputerScienceHIT/Lab4-1170300817>

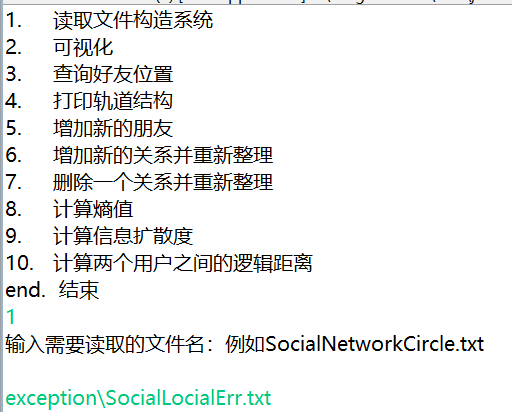
# 实验过程

请仔细对照实验手册，针对每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## Error and Exception Handling

下列所有错误出现后都会提示用户重新输入读取的文件的地址

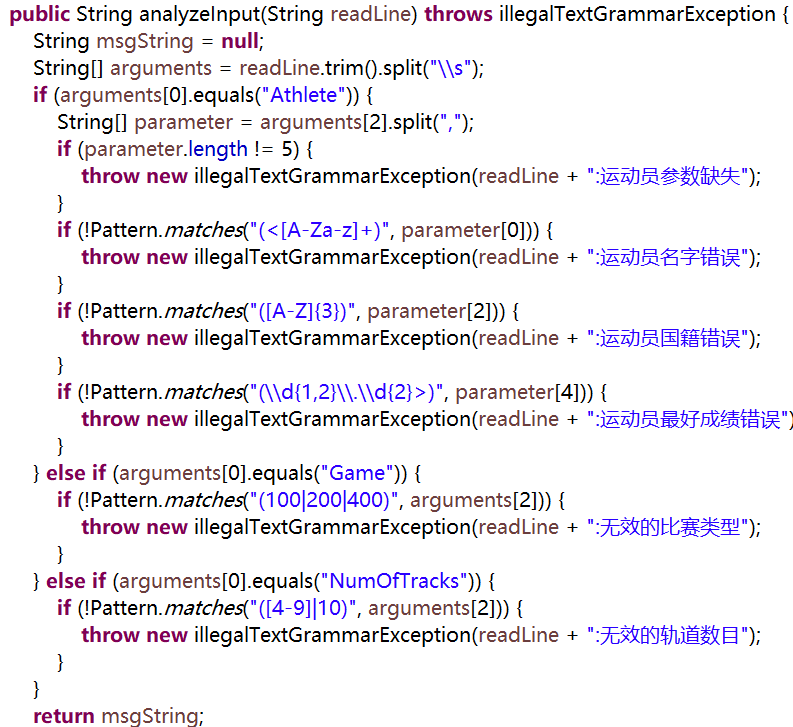
以下省略开始选择菜单项1的步骤：



### 输入不合法

截取到不合规定的文件输入之后，调用一个分析文本的方法analyzeInput，以trackgame为例：

将输入拆分成若干小块，然后将他们分别与原来的正则表达式的每个小部分匹配，就能找到是哪个部分出现了何种类型错误



测试文件位于

各异常触发情况如下：分别按顺序读取异常文件

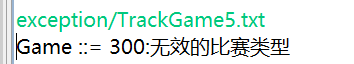
#### TrackGame

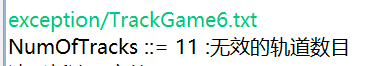








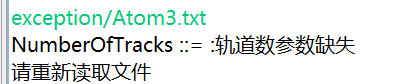


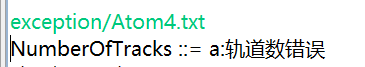


#### Atomstructure









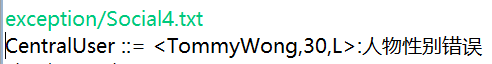


#### Socialnetwork









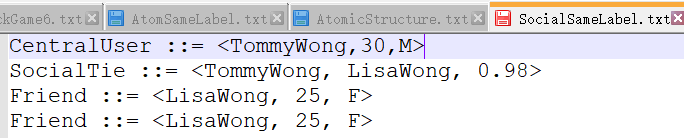




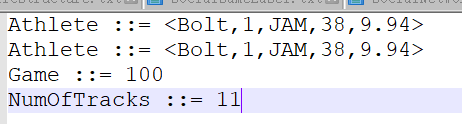


### 标签一样错误

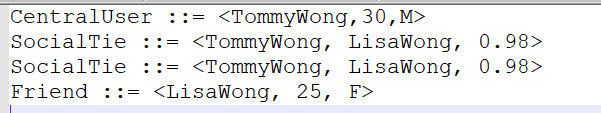
设计如下输入，获得以下输出

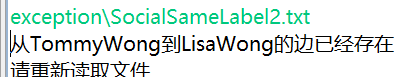








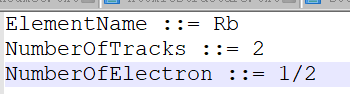


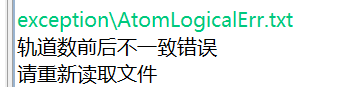


### 依赖关系错误

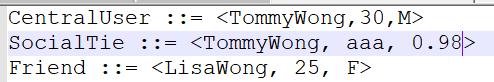
设计如下输入，获得输出：

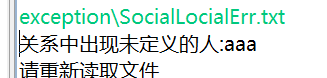
轨道数目不一致：





关系中出现未定义的人：



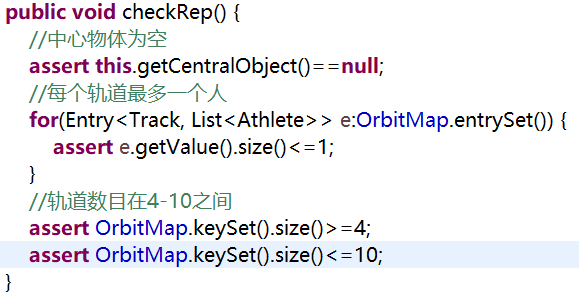


## Assertion and Defensive Programming

### checkRep()检查invariants

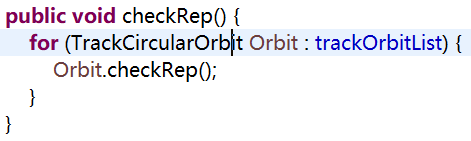
#### TrackGame：

在TrackCircularOrbit修改checkRep：



在TrackGame中则要对比赛每一轮对应的TrackCircularOrbit调用checkRep，所以再写一个

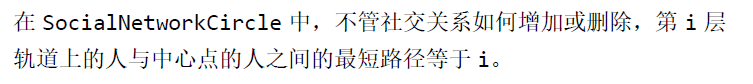
checkRep检查每一轮的轨道系统：



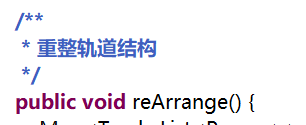
对于一些其他条件例如：“前n-1个轨道必须满编”通过安排竞赛的算法的正确性保证的，故不做检验。

#### SocialNetCircle：

对于：



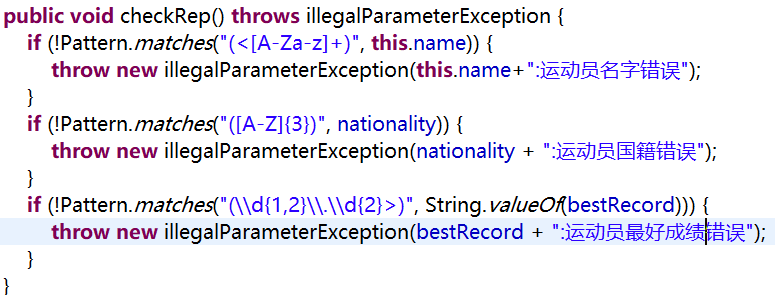
这个要求，因为我的实现在安排轨道的结构的时候是先用BFS搜索算出物体到中心点的逻辑距离，再把这个物体放在对应逻辑距离对应的轨道上，所以，这个验证无异于“自己验证自己”，故没有实现，这一个要求是通过每次增删关系都重新调用一次重整轨道结构的reArrange方法，以及在reArrange方法中使用的BFS算法的正确性来保证的。



#### 针对各个physicalObject的子类：

用正则匹配的方式检验参数是否合法，不合法抛出异常，方法和文件读取阶段的异常处理无异：

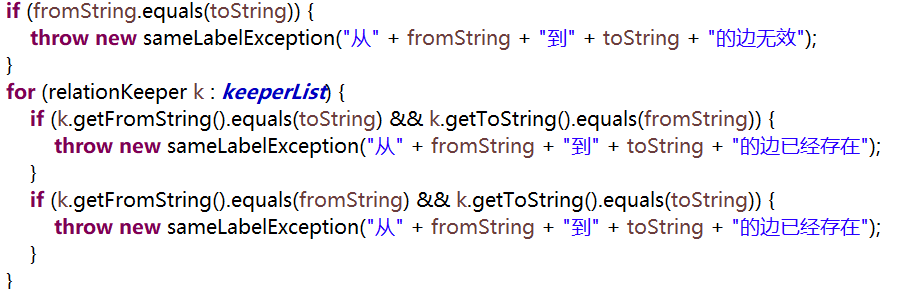
举例：



### Assertion保障pre-/post-condition

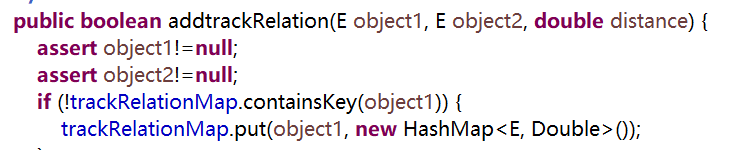
有的时候pre-/post-condition不是通过assertion来保证，而是通过抛出异常

例如：社交网络中要求加入的边不能有自己到自己，或者是ab，ba同时出现：



还有一些简单的使用assert保障pre-/post-condition的：举个例子：

向ConcreteCircularOrbit中添加新的点、轨道、关系、中心物体时，都要防止对象是一个null。



另外一些防御式的编程的例子比如使用防御性拷贝



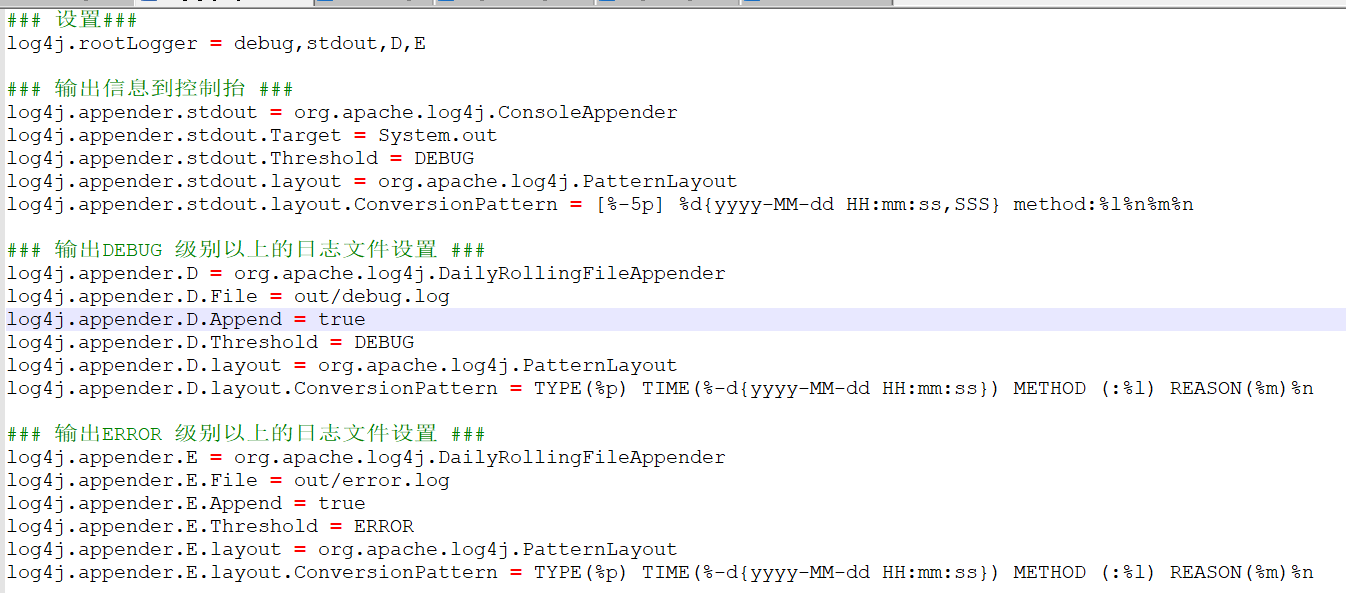
## Logging

### 写日志

导入log4j包，编写配置文件放置于src文件夹下（参考网上）

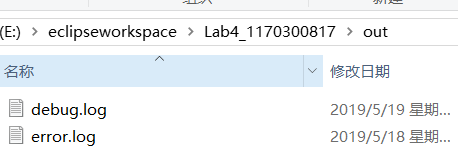
设置好控制台输出和文本输出，

配置文件如下：

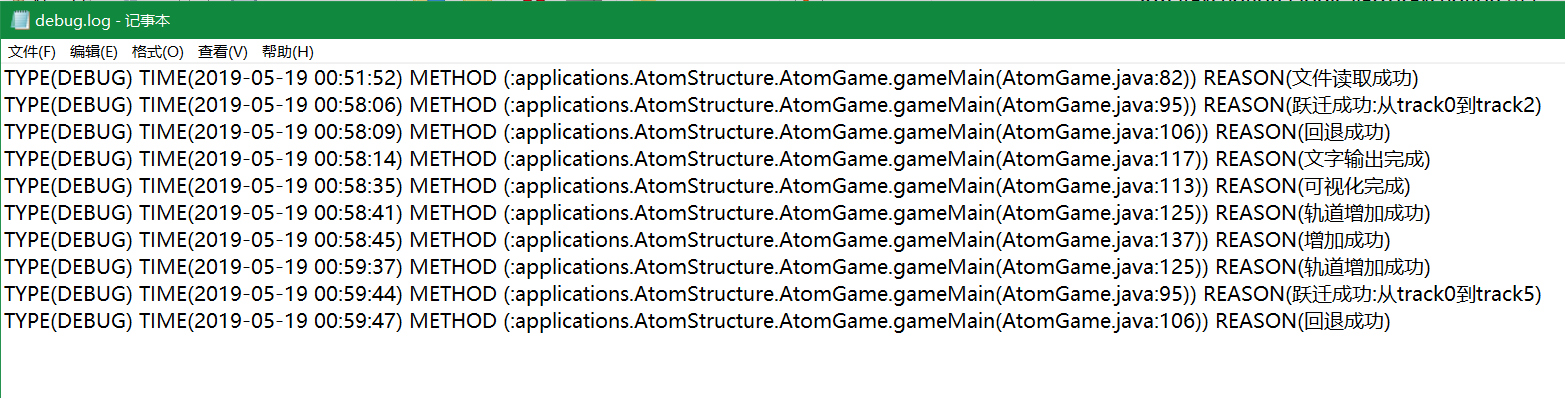


在每一个方法后面增加记录日志的操作：

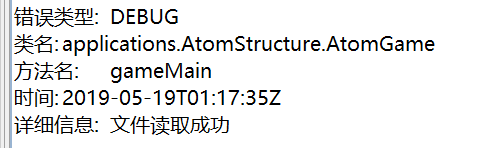
对于所有的异常，都以 ERROR级别记录，对于所有的操作提示和记录，都已DEBUG级别记录，分别输出到以下两个文件中



生成的日志文件如下（以trackgame为例）：

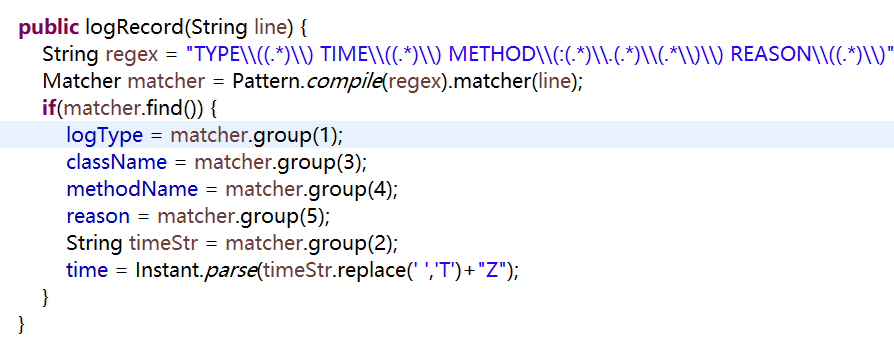


输出如下：

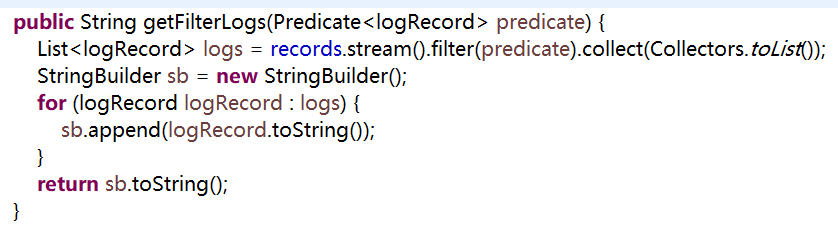


### 日志查询

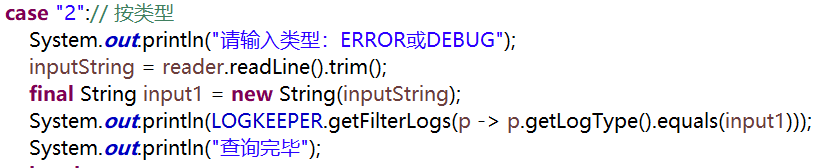
首先用正则表达式读取整个日志文件：



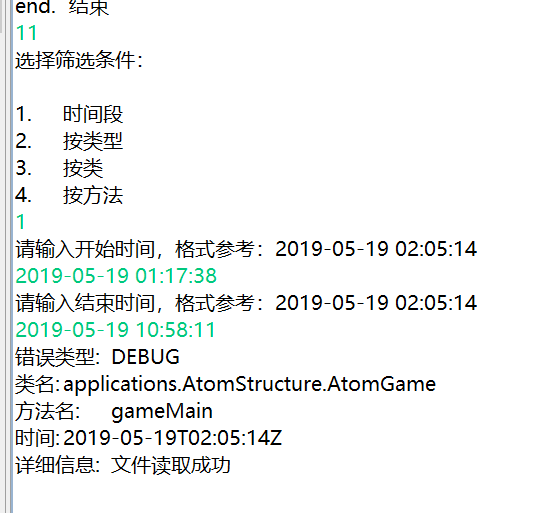
学习使用了JAVA 8的新特性用stream来筛选日志列表

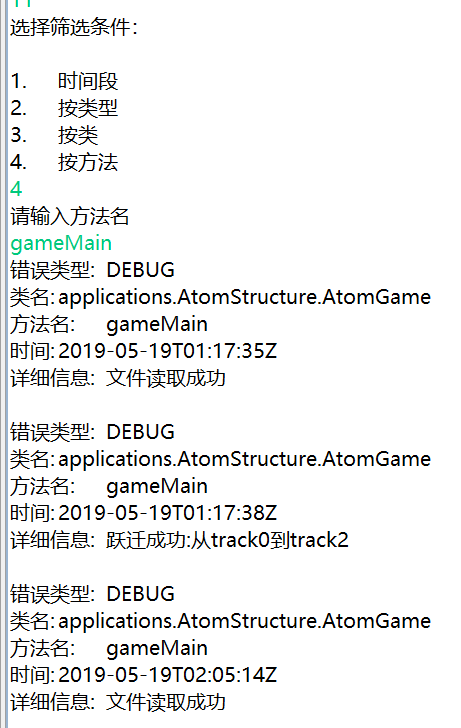


只需传入一个Predicate即可完成筛选例如:



查询演示：





## Testing for Robustness and Correctness

### Testing strategy

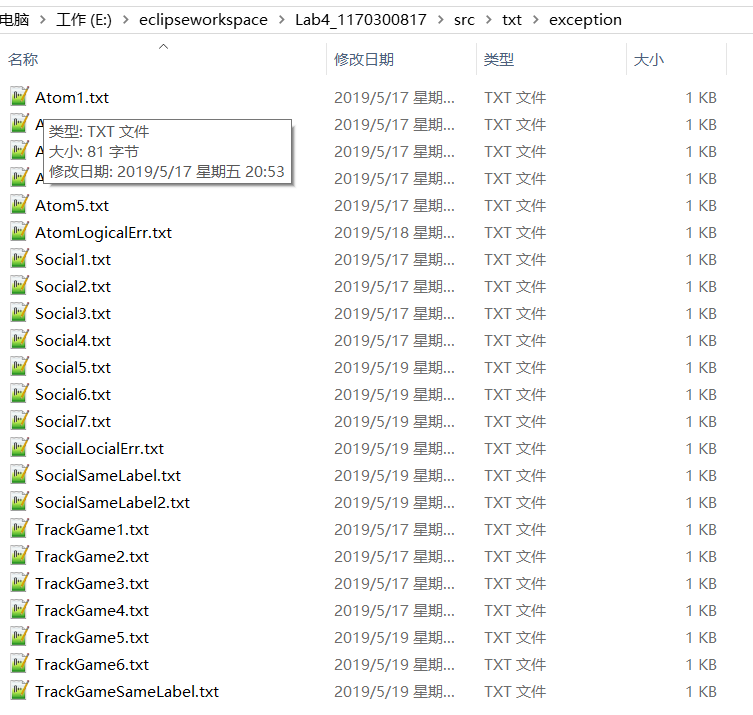
总体策略是尽可能提高代码的覆盖率，尽可能覆盖到每一个分支，所以需要等价类划分的思想。对于有exception的功能，必须测试到每一个exception和相应的处理是否正确执行，对3.1节编写的每一个exception都编写单独的测试文档来测试。

### 测试用例设计

#### 文件读写过程：

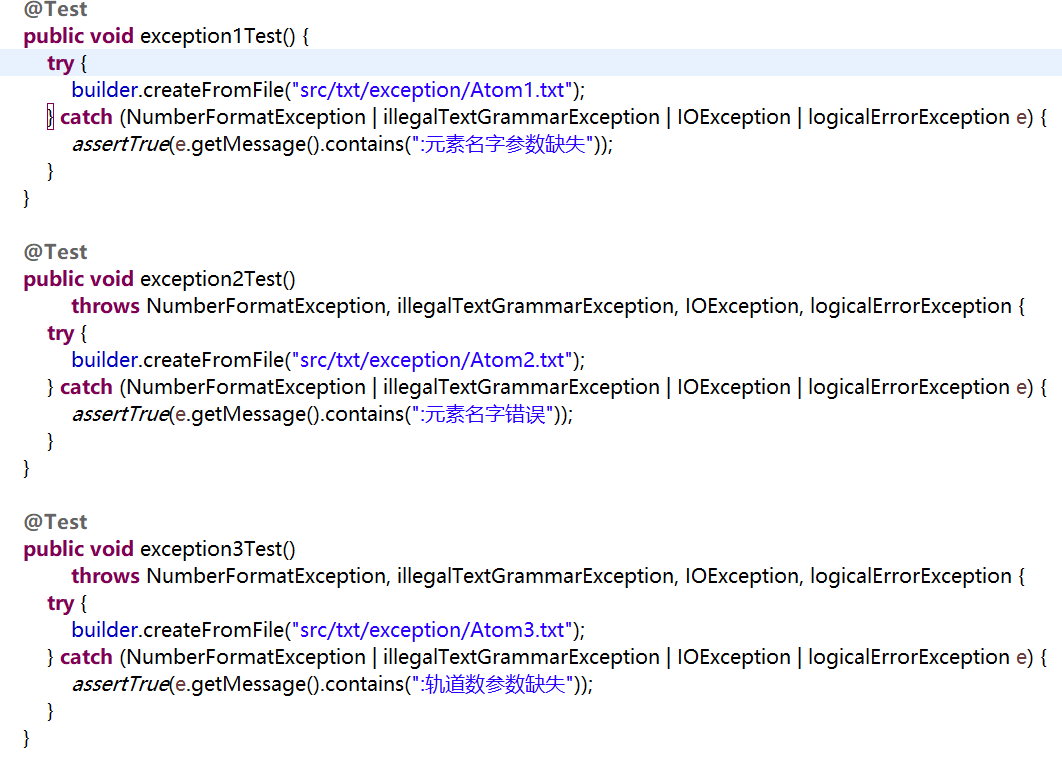
针对每一个exception的每一种触发情况都编写对应的测试用例和测试文件：

目录：src/txt/exception



编写对应的test读取这些文件，检测是否出现了设计好应该出现的错误：

举例如下：



#### 不在文件读入过程发生的参数错误

不在文件读入过程发生的参数错误，举例：



#### 健壮性测试

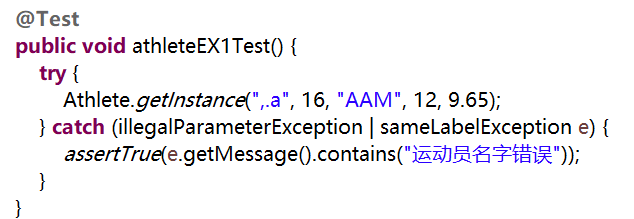
比如：移除目标的轨道上没有物体，移除物体不在轨道系统里等情况能否正常返回

举例：执行两次transit，轨道上只有一个物体，第二次返回的应该是个false

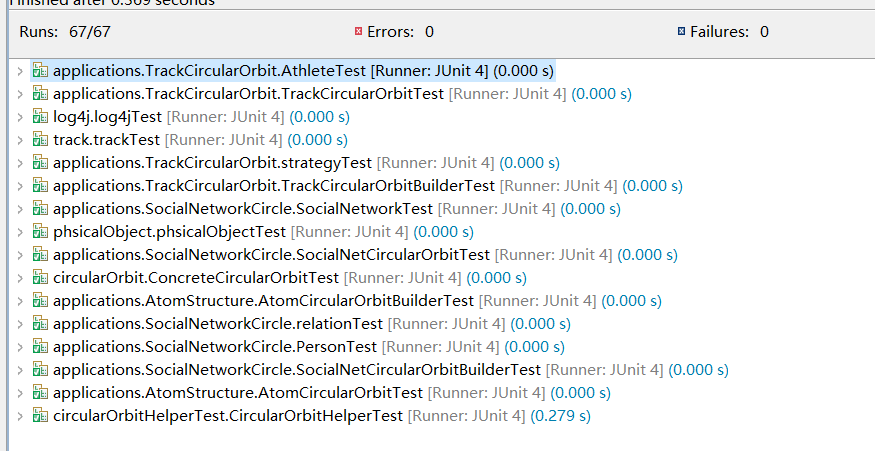


试图声明一些非法参数的物体：

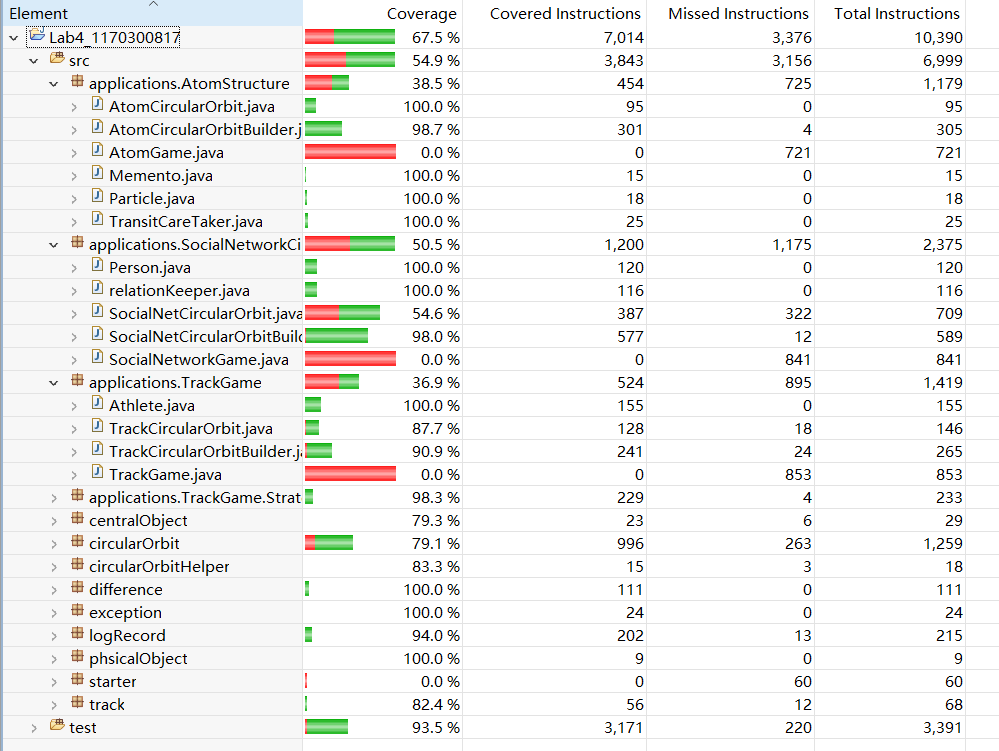
检测catch到的异常的信息是否和错误原因一致



### 测试运行结果与EclEmma覆盖度报告



注：客户端代码和图形化的部分无法测试，所以以下的AtomGame，SocialNetworkGame ，TrackGame无法覆盖，覆盖度为0。



## SpotBugs tool

发现了哪些错误，每种错误代表什么不良的编程习惯

对代码修改，消除这些错误。

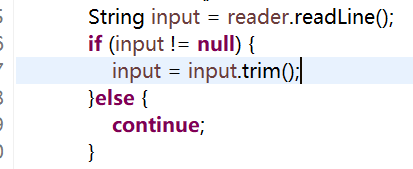
### readLine

BufferedReader类的readLine()在没有文本读取的情况下将 返回null，在操作之前需要判断是否为null。

旧：



新：



### 类名大写开头

旧：



新：

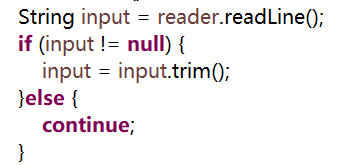
### Readline错误

Readline要判断非null

旧：



新：



## Debugging

### 理解待调试程序的代码思想

#### findMedianSortedArrays

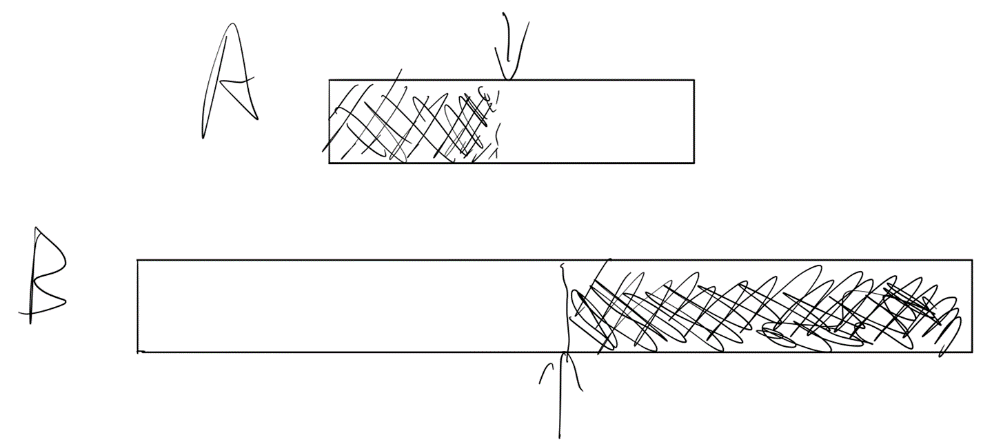
目的：寻找两个有序数组合起来的中位数

不需要真正的合并而是找到两个特殊值minRight和maxLeft分别是合并以后的右半边最小值和左半边最大值，然后对这两个数求平均值。

算法本质上是寻找两个指针i，j，使得A[0-i]和B[0-j]合起来正好是AB合并之后的左半部分

所以i+j一定等于总长度的一半。

先把A数组换成短的数组，因为A，B数组有序，所以A的前半部分和B的后半部分是不用考虑的，因为A的前半肯定在合并后的左侧，B的后半肯定在合并后的右侧。



随后一步步寻找这样的i，j使得B[j - 1] <=A[i]，A[i - 1] <= B[j]，这样就找到了要求的i，j。

随后得到

maxLeft=max(B[ j - 1], A[ i – 1 ])

minRight=min(B[ j ] , A[ i ])

取平均值即可。

#### removeComments

判断一段代码是否是在注释中，只需要判断代码之前是否有”/\*”或者”//”即可，如果不是注释代码，加入数组返回。

#### TopVotedCandidate

参考了网上的解答（leetcode）

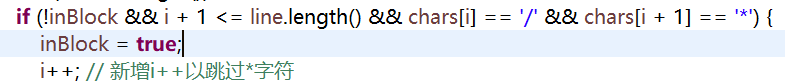
解法：设置一个目前得票最多的变量count，先循环得票人和时间，用hashmap或数组数组统计每个人的得票数，如果次数大于count，则更新count，用数组记录到目前时间得票最多的人(按时间顺序排列的目前得票多的人)。查找某一时间点得票最多的人时，用二分法查找按时间排序的数组。

### 发现并定位错误的过程

#### findMedianSortedArrays

根据算法要求的i，j的和的关系（A[i]左边和B[j]左边占总元素的一半）可以知道，这里长度和i指针的计算有错误。

#### removeComments



这里如果没有i++就会使指针越界。

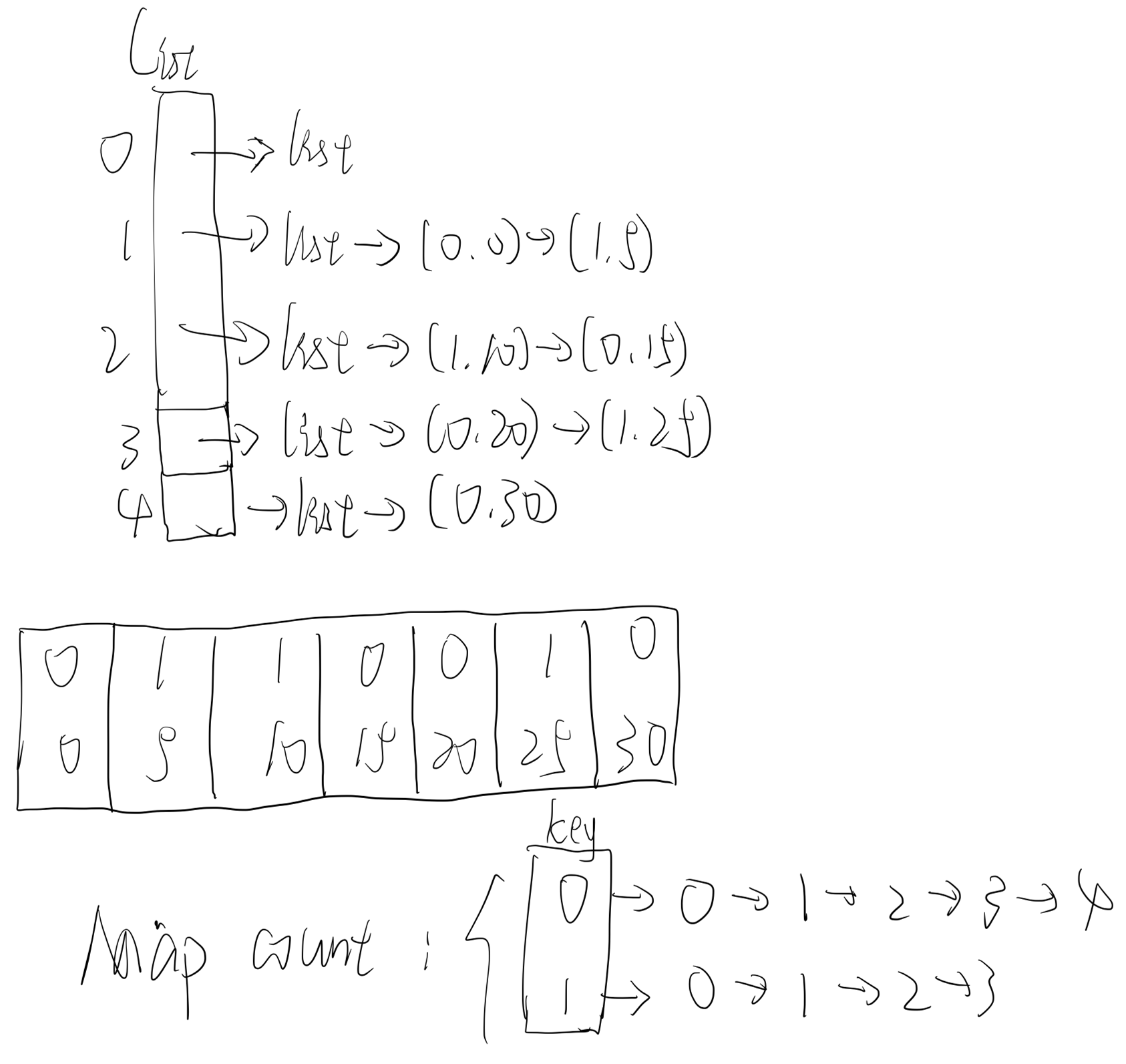
缺少处理单行注释的代码：

逻辑有误：

inblock时不应该写入

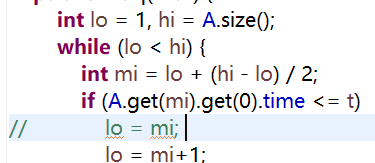
#### TopVotedCandidate

分析流程和数据结构如下：





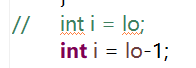
Count记录投票次数，显然初始值为0，每次加1.



lo类似搜索下界，当前的mi已经搜索过，可以改成mi+1



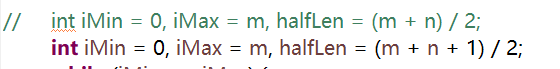
截止到某个时间点，所以是<=

和

### 如何修正错误

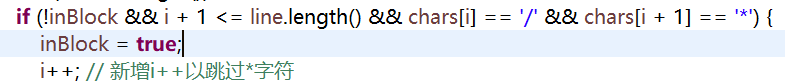
#### findMedianSortedArrays

修正i，j和halflen

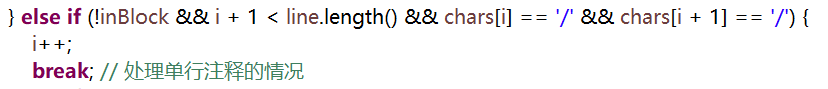


#### removeComments



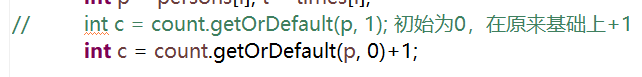
新增：

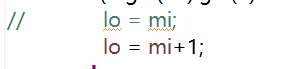
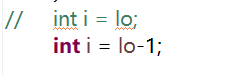


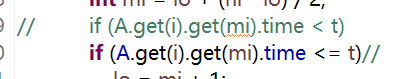
修改逻辑：

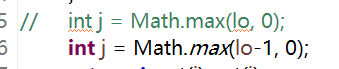


#### TopVotedCandidate



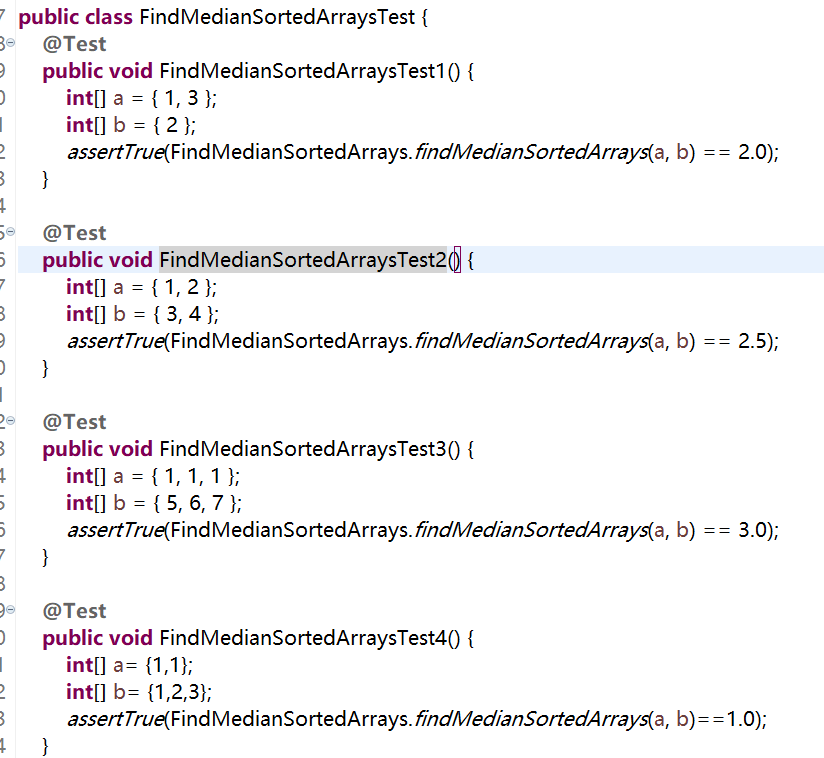
 

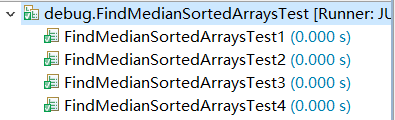




### 结果

#### findMedianSortedArrays

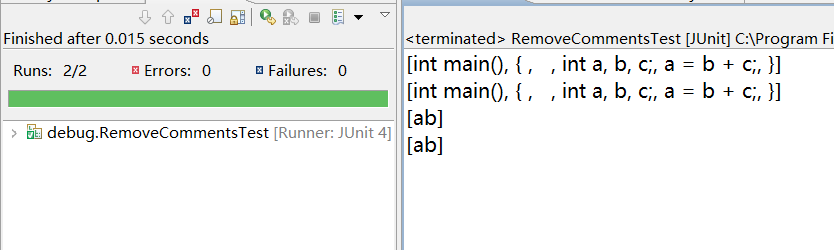




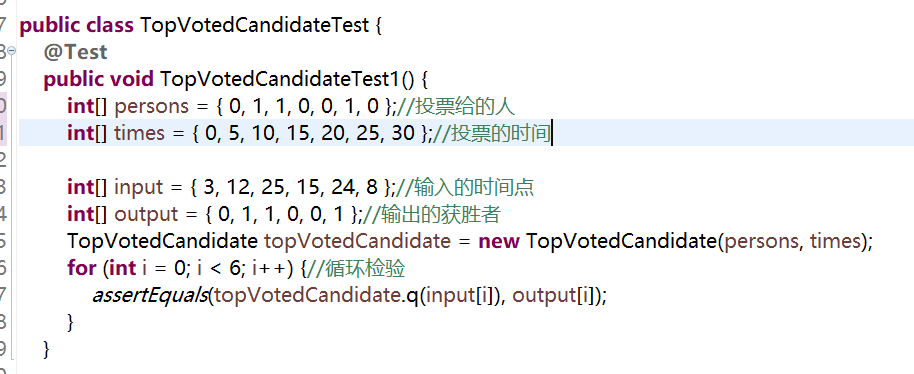
#### removeComments

直接放上测试代码：





#### TopVotedCandidate





# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 5.12 | 19:00-22:00 | 3.1 | 完成 |
| 5.13 | 19:00-22:00 | 3.2 | 完成 |
| 5.15 | 19:00-22:00 | 3.3 | 完成 |
| 5.17 | 19:00-22:00 | 3.3 | 完成 |
| 5.17 | 19:00-22:00 | 3.4 | 完成 |
| 5.18 | 19:00-22:00 | 3.5 | 完成 |
| 5.19 | 8:00-22:00 | 3.5+3.6 | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| Log4j不会用 | 上网找教程 |
| Stream新特性 | 翻看核心技术 |
|  |  |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

## 针对以下方面的感受

1. 健壮性和正确性，二者对编程中程序员的思路有什么不同的影响？

健壮性使程序员考虑更多异常情况。

1. 为了应对1%可能出现的错误或异常，需要增加很多行的代码，这是否划算？（考虑这个反例：民航飞机上为何不安装降落伞？）

具体看是什么样的系统，如果是很重要的系统，成本再大也要保证

1. “让自己的程序能应对更多的异常情况”和“让客户端/程序的用户承担确保正确性的职责”，二者有什么差异？你在哪些编程场景下会考虑遵循前者、在哪些场景下考虑遵循后者？

在外部接口处倾向于第一种，如果是内部接口的话倾向于第一种。

1. 过分谨慎的“防御”（excessively defensive）真的有必要吗？如果你在完成Lab5的时候发现Lab5追求的是I/O大文件时的性能（时间/空间），你是否会回过头来修改你在Lab3和本实验里所做的各类defensive措施？如何在二者之间取得平衡？

不太了解，希望继续学习能找到答案。

1. 通过调试发现并定位错误，你自己的编程经历中有总结出一些有效的方法吗？请分享之。Assertion和log技术是否会帮助你更有效的定位错误？

合理使用Assertion和log十分有益

1. 怎么才是“充分的测试”？代码覆盖度100%是否就意味着100%充分的测试？

不，有些边界条件测试覆盖度无法体现。

1. Debug一个错误的程序，有乐趣吗？

被迫阅读别人代码说实话没有什么乐趣。

1. 关于本实验的工作量、难度、deadline。

工作量太大。

1. 到目前为止你对《软件构造》课程的评价和建议。

有的要求太严苛。

1. 期末考试临近，你对占成绩60%的闭卷考试有什么期望或建议？//请严肃的提出，杜绝开玩笑，教师会认真考虑你们的建议。

希望不是考概念背诵。