

**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 | 汤添凝 |
| 学号 | 1170300728 |
| 班号 | 1703007 |
| 电子邮件 | ttn912@126.com |
| 手机号码 | 18016240627 |

**目录**

[1 实验目标概述 1](#_Toc26555)

[2 实验环境配置 1](#_Toc10399)

[3 实验过程 2](#_Toc30432)

[3.1 Error and Exception Handling 2](#_Toc32515)

[3.2 Assertion and Defensive Programming 3](#_Toc23764)

[3.2.1 checkRep()检查invariants 3](#_Toc13280)

[3.2.2 Assertion保障pre-/post-condition 3](#_Toc9136)

[3.3 Logging 4](#_Toc25111)

[3.4 Testing for Robustness and Correctness 4](#_Toc30545)

[3.4.1 测试用例设计 4](#_Toc6278)

[3.4.2 测试运行结果与EclEmma覆盖度报告 4](#_Toc23469)

[3.5 SpotBugs tool 5](#_Toc408)

[3.6 Debugging 7](#_Toc22180)

[3.6.1 理解待调试程序的代码思想 7](#_Toc1092)

[3.6.1.1 FindMedianSortedArrays 7](#_Toc12680)

[3.6.1.2 RemoveComments 7](#_Toc4610)

[3.6.1.3 TopVotedCandidate 7](#_Toc22448)

[3.6.2 发现并定位错误的过程 7](#_Toc17100)

[3.6.2.1 FindMedianSortedArrays 7](#_Toc5193)

[3.6.2.2 RemoveComments 7](#_Toc10981)

[3.6.2.3 TopVotedCandidate 8](#_Toc13578)

[3.6.3 如何修正错误 8](#_Toc7964)

[3.6.3.1 FindMedianSortedArrays 8](#_Toc12303)

[3.6.3.2 RemoveComments 9](#_Toc5548)

[3.6.3.3 TopVotedCandidate 9](#_Toc18853)

[3.6.4 结果 10](#_Toc13500)

[4 实验进度记录 10](#_Toc32283)

[5 实验过程中遇到的困难与解决途径 10](#_Toc25349)

[6 实验过程中收获的经验、教训、感想 11](#_Toc3687)

[6.1 实验过程中收获的经验和教训 11](#_Toc26196)

[6.2 针对以下方面的感受 11](#_Toc22279)

# 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

⚫ Assertion 和防御式编程

⚫ 日志

⚫ 调试技术

⚫ 黑盒测试及代码覆盖度

# 实验环境配置

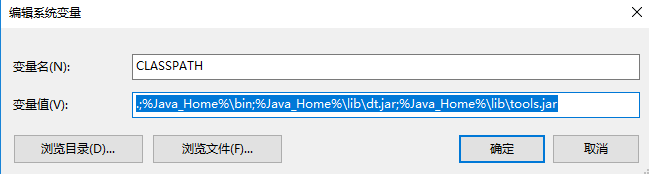
<http://web.mit.edu/6.031/www/fa18/getting-started/>中的安装Eclipse和Gitd的指南非常详细，按照文字和图片即可完成，但是JDK的官网里没有8、9、10版本的，所以我先下载了JDK11，后来在其他网站找到了JDK8版本。

java环境的配置：

安装好JDK后，计算机→属性→高级系统设置→高级→环境变量→系统变量→新建JAVA\_Home变量（填写JDK的安装路径如下图） 

接下来找到并编辑Path变量，在变量值最后输入“%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;”

最后新建CLASSPATH变量，变量值填写“ .;%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar ”，

系统变量就配置好了。

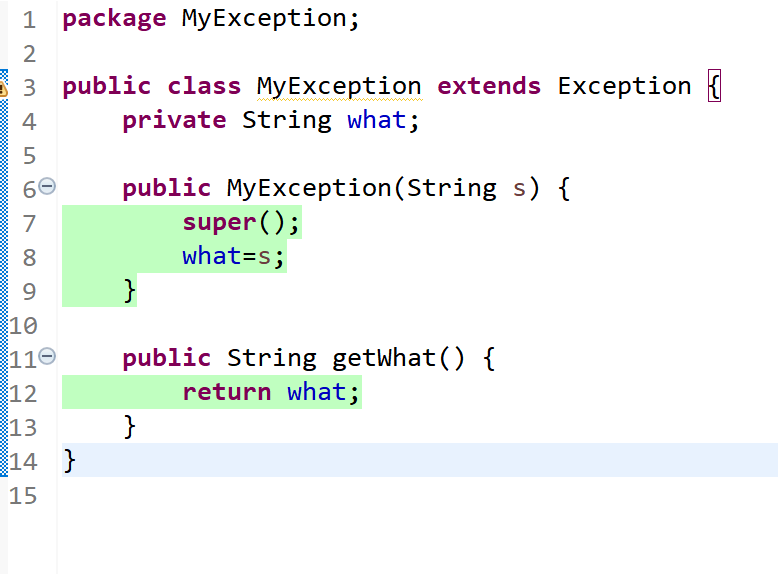
URL：[https://github.com/ComputerScienceHIT/Lab4-1170300728](https://github.com/ComputerScienceHIT/Lab2-1170300728)

# 实验过程

## Error and Exception Handling

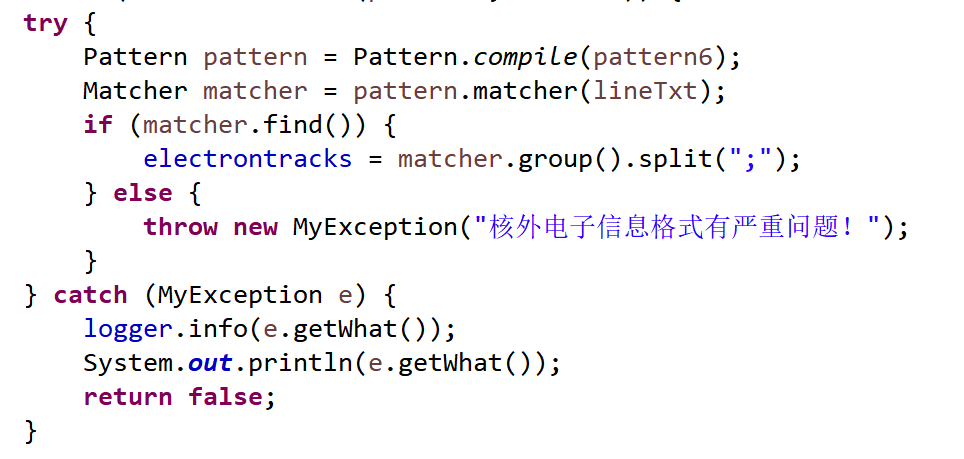
这一部分要求通过自定义 Exception 类型，表示程序的一些常见错误类型，为代码可能发生的问题提供新的含义，区分代码运行时可能出现的相似问题，或给出程序中一组共性错误的特殊含义。

于是我设计了MyException类，继承自Exception，如下图所示：



这里增设了一个what，用于储存需要抛出的问题描述语句。

随后便是应用MyException去完成实验内容。我们在可能发生错误的语句段外用try括起来，并加入判断语句，如果判断为异常状况则抛出异常，跳出语句块。以下图所示部分为例，可能出错在于信息格式不对，一旦发现，立即throw异常，打印错误信息，并返回false（正常返回为true）。可以看到有logging相关语句，是后面日志部分的内容，此处不详细说明了。



类似的，还有存在标签完全一样的元素、文件中各元素之间的依赖关系不正确等异常，这里不一一赘述。

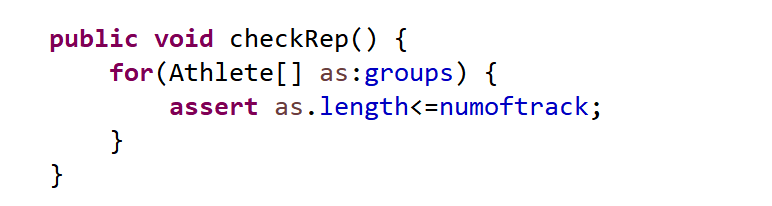
## Assertion and Defensive Programming

这一部分要求基于 Lab 3 里各个 ADT 所撰写的 rep invariants (RI)、Abstraction Function (AF)，以及它们的每个方法的 spec（pre-condition 和 post-condition），对代码进行防御式改造。

### checkRep()检查invariants

防御性改造的第一部分就是checkrep。

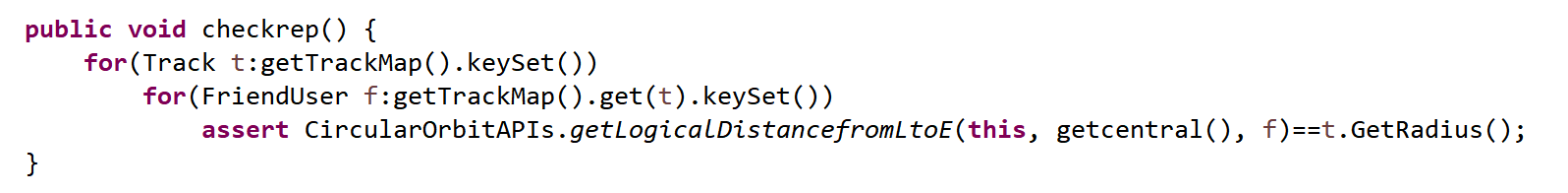
针对trackgame，我们要求了不管用户对比赛方案如何修改，每组比赛中的人数不超过跑道数这个要求不难实现。编写如下图所示的checkRep函数，比较轨道数与赛道数，并在所有可能的改变行为中加入即可。



针对AtomStructure，由于电子跃迁的存在，核外电子排布并不一定按照元素标准核外电子排布而存在。所以并没有绝对的轨道系统要求，不做checkrep处理。

针对SocialNetworkCircle，根据描述我们可以知道，这个轨道模型的第 i 层

轨道上的人与中心点的人之间的最短路径等于 i，利用这条规则，编写如下图的checkRep函数，这里由于之前所讨论的都是轨道物体到轨道物体的逻辑距离，所以从中心物体到轨道物体的逻辑距离函数其实是实验过程中依照轨道物体间逻辑距离函数加以修改写成的，也耗费了不少意料之外的时间。



### Assertion保障pre-/post-condition

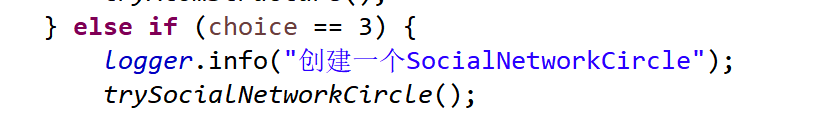
Assertion内容在checkRep、try-catch等过程中已经同时完成。

## Logging

这一部分要求使用 java logging（或其他第三方 java 日志库如 log4j、SLF4J），为 3.1和 3.2 节经过异常处理、错误处理、断言处理的程序增加日志功能。

再拿3.1的例子来看，这里出现了异常处理的部分，需要输出日志，于是利用logging的info功能获取异常信息并记录日志。

类似的在操作选项中也有相应的操作日志。如下图例子：

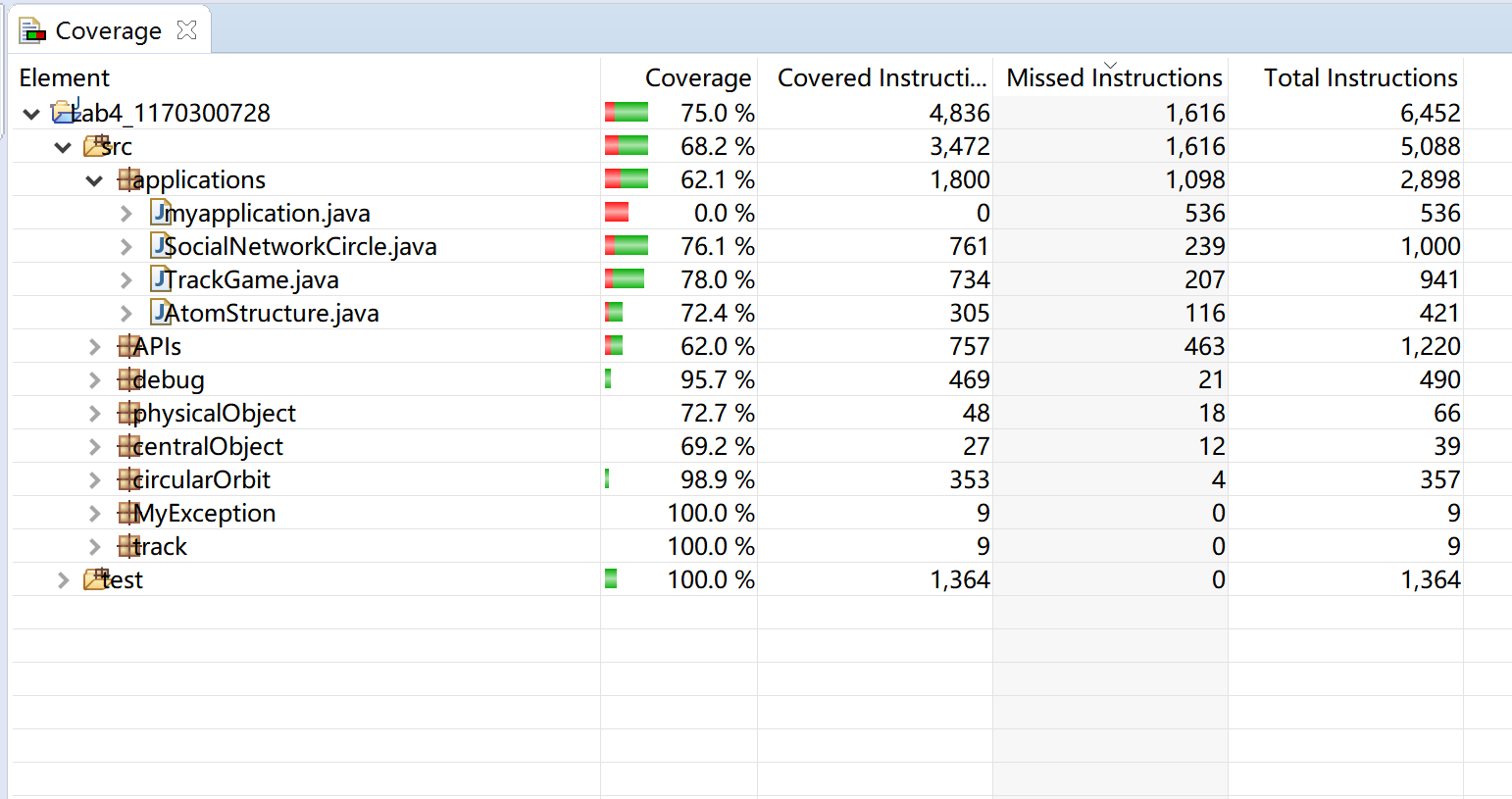


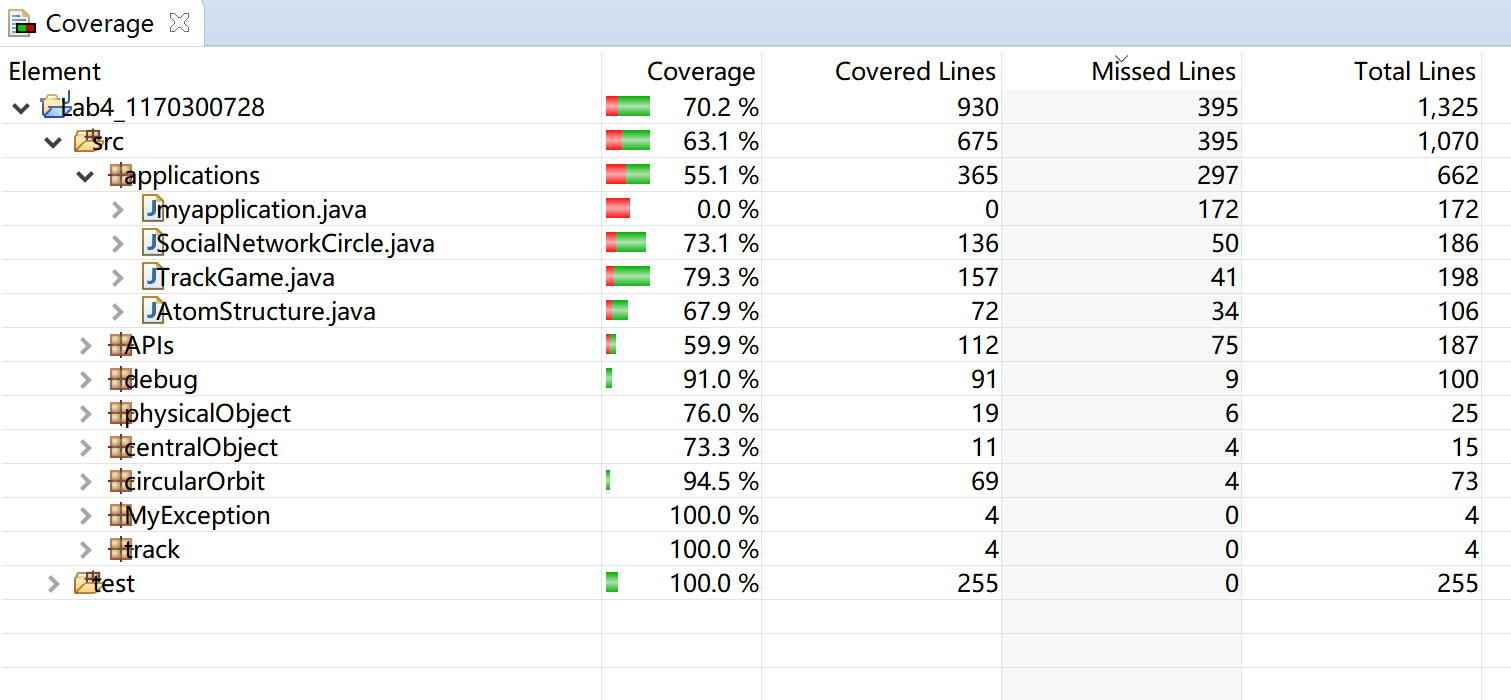
## Testing for Robustness and Correctness

### 测试用例设计

这一部分有健壮性的测试要求，即设计一组测试用例，人为制造非法输入的文件和非法输入的图操作指令验证程序是否有容错能力。于是修改了样例测试文件XXXtest.txt为XXXtesterrorsX.txt，其中分别做出了不同的非法输入操作等。在test函数中运行，如下图可见运行顺利。

### 测试运行结果与EclEmma覆盖度报告

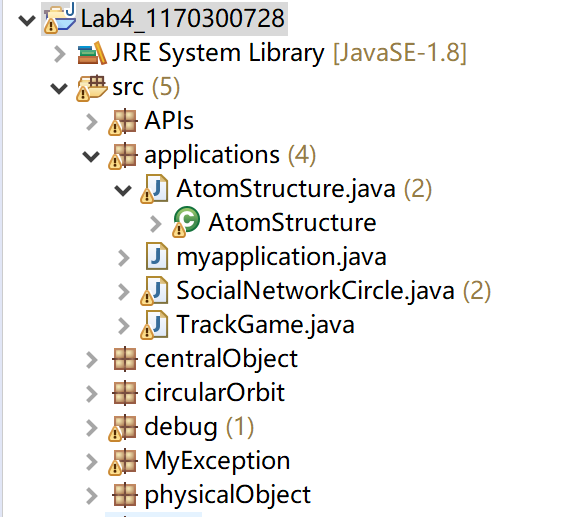




覆盖率在70-80左右。

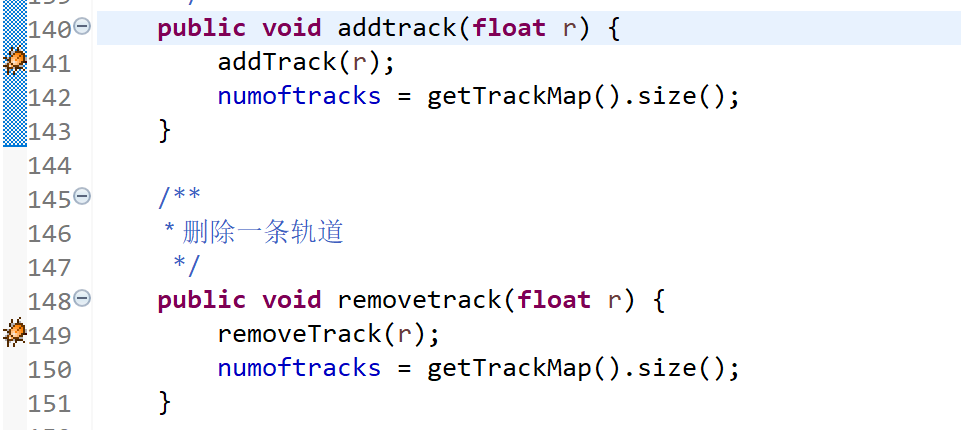
## SpotBugs tool

发现的错误不多，没有按下按钮前预测的一片bug的情景出现。



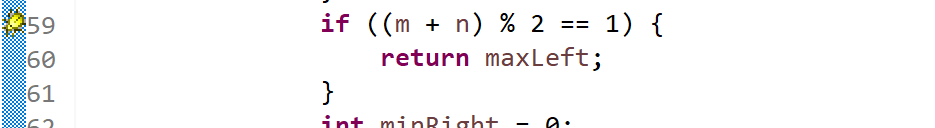
如上图所示，总共有5个bug。具体如下：

首先如下图所示，提示我在函数名称上有问题。

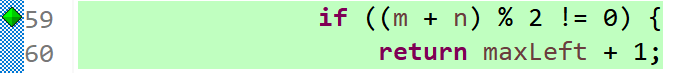


由于trackgame.addtrack与ConcreteCircularOrbit.addTrack相差不多，我在实现时调用了该函数，spotbugs提示我说函数名过于相似，于是我在名字上少加了改动。类似这样的bug总共出现了四次。

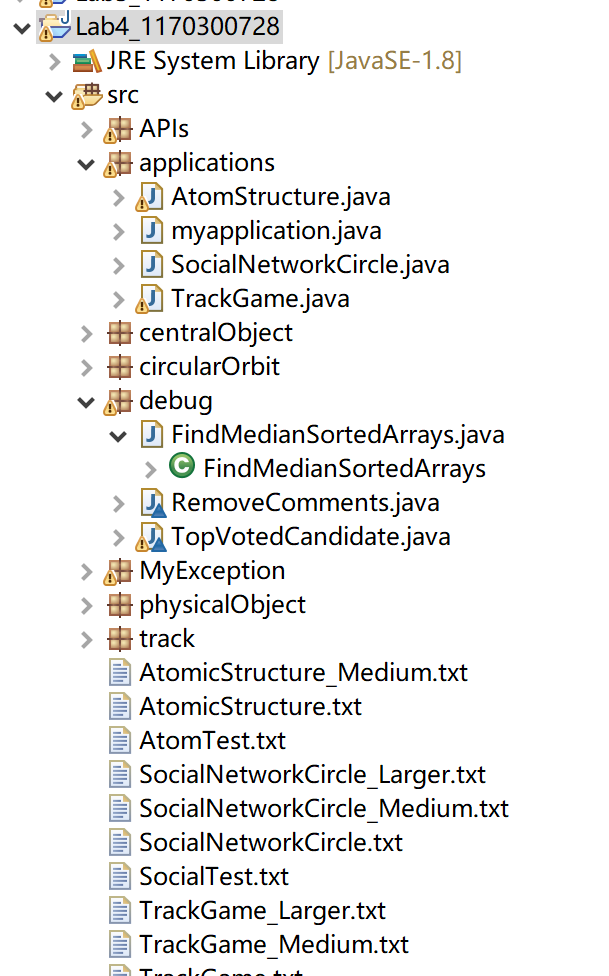
第五个bug出现在3.6的findMedianSortedArrays。



Spotbugs提示我java的整数求余会出问题，例如：-3%2=-1而不是1，于是改成下图形式。



最后spotbugs不再能给我更多的提示了。



## Debugging

### 理解待调试程序的代码思想

### 3.6.1.1 FindMedianSortedArrays

将两个数组按数组长度排序，大的为A，随后，利用类似二分法的方法，区分两组数字，大的在右侧，小的在左侧，但两侧并不绝对排序。然后再利用相似的方法找到左右两侧的边界值，即可获得中位数。

### 3.6.1.2 RemoveComments

将读入的字符串列表一个个取出并修正为字符数组，检索并读取数组中的字符，若发现连续的“/”和“\*”则跳过接下来的内容不做读取，直到出现连续的“\*”和“/”为止。

### 3.6.1.3 TopVotedCandidate

依次读入候选人的得票信息。使用List<List<Vote>>，此时，第一层list代表这个list中的每个list中的对象都曾获得过相当于其目录的票数、第二层list则是记录这其中每个候选人得到这个票数时是什么时候。

然后在q函数中，利用二分法，查找第一个获得各种票数的人中恰好时间慢于所求时间的那个目录，即得票数；随后再用二分法找到同样得票的人中，时间最接近的候选人并返回。

### 发现并定位错误的过程

### 3.6.2.1 FindMedianSortedArrays

直接根据逻辑发现一些算数表达式上的问题。而且在例子中要求对null也能使用，于是在开始的时候判断了一下是否为空。

### 3.6.2.2 RemoveComments

首先根据提示中的样例编写test函数，运行发现不通过。检查判断语句，发现逻辑问题，inblock的布尔值代表“是否在注释块中”而它读取了“在注释块中”的字符，显然不对。更正后发现输出中总会带着注释块的边缘即“/”字符，思考读取逻辑，发现当读取到“\*/”时，只删除了\*。而且他没有考虑“//”的情况，需要添加入代码中。

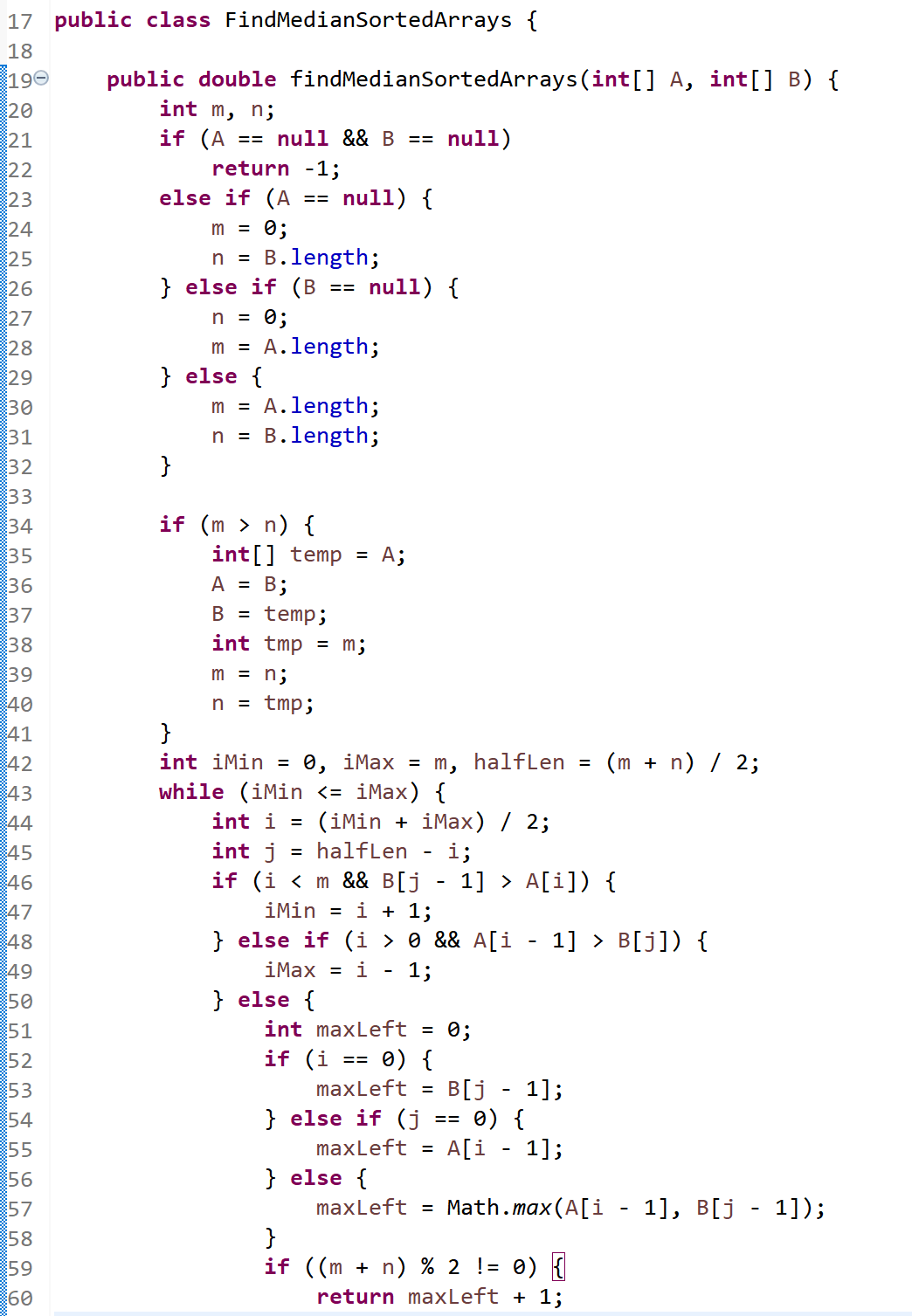
### 3.6.2.3 TopVotedCandidate

同样根据提示中的样例编写test函数，运行发现会死循环。查看循环语句，发现问题并解决。随后发现两个二分法是不同的，而且各有对错，进行修正。最后串联逻辑，找到正确的人。

### 如何修正错误

### 3.6.3.1 FindMedianSortedArrays

由于是式子细节的问题，这里不做具体描述了，修改后的代码如下：

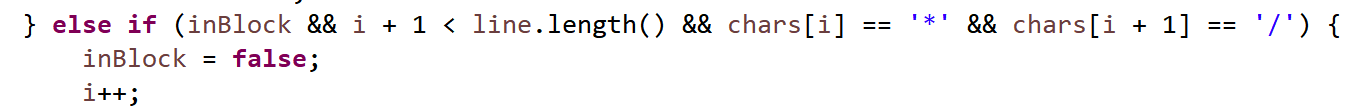


### 3.6.3.2 RemoveComments

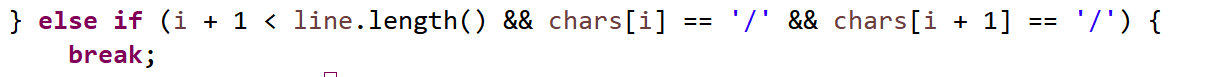
三个问题逐个解决：一、如下图所示将if(inblock)改为if(!inblock);

IMG_256

如下图添加“i++”语句使目录标记到达正确位置

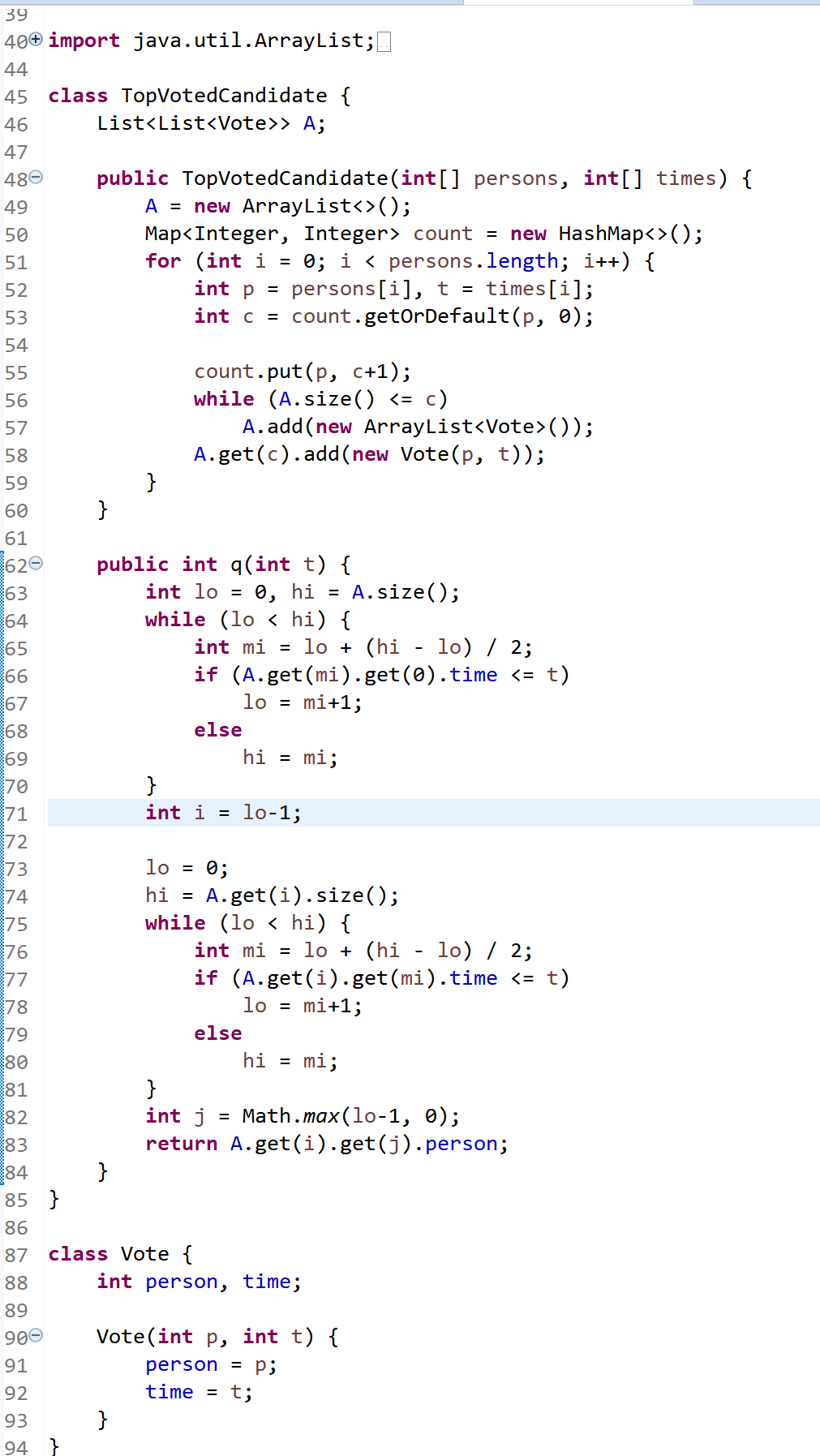


添加如下语句，当检测到“//”时直接跳过这一行，开始下一行的读取。

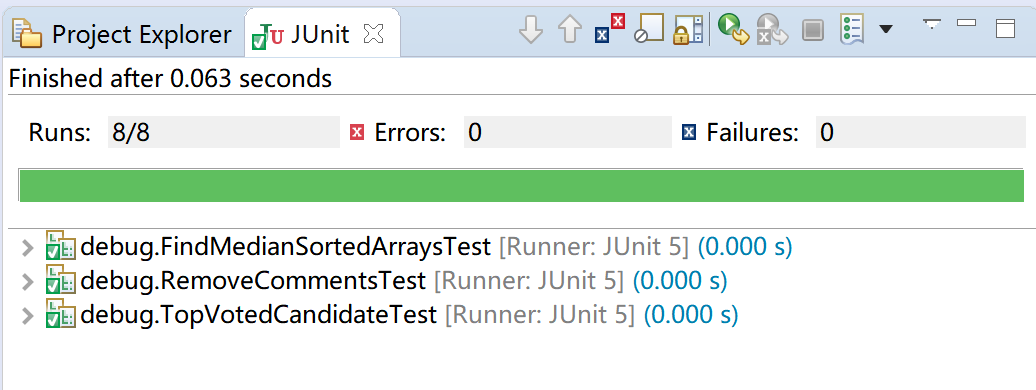


### 3.6.3.3 TopVotedCandidate

由于这个函数的数理逻辑与我的思路稍有不同，所以在目录索引的微调上下了较大功夫，此处直接贴出修改后的代码。需要注意的是，我的思路中很难加入“++i”代码，容易打乱我的思路，所以我直接改为了“i++”这也可能是我的修改幅度比前两问大的原因。



### 结果



# 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 |
| 2019-05-9 | 13:00-16:30 | 自学并编写异常处理单元 | 按计划完成 |
| 2019-05-11 | 14:00-16:00 | 测试异常处理单元 | 按计划完成 |
| 2019-05-12 | 13:45-15:30 | 自学并编写日志内容 | 遇到困难，延时完成 |
| 2019-05-13 | 13:00-17:00 | 设计错误用例，并测试代码健壮性 | 按计划完成 |
| 2019-05-14 | 16:00-19:00 | 学习spotbugs使用方法并使用 | 按计划完成 |
| 2019-05-15 | 10:45-20:30 | 3.6 Debug相关内容 | 按计划完成 |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 没用过日志 | 通过自学logging完成 |
| 较少使用异常处理，JAVA更少 | 通过自学编写MyException完成 |

# 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

Java语言是面向对象的典型编程语言。这次实验遇到了许多java语言日志撰写、异常处理上的困难，但是经过查阅书籍、上网搜索或者寻求助教以及同学的帮助之后都成功解决了。

这次实验复习并巩固了以前学习的编程知识，加强了面向ADT编程的计算思维。还学会了各种eclipse的操作方法，有助于java编程。 体验了错误处理、异常处理、Assertion 和防御式编程、日志、调试技术、黑盒测试及代码覆盖度的体验。相信之后的编程中也会更多的注重健壮性和正确性。

## 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？

健壮性保证了代码不容易崩溃。正确性保证了代码的逻辑通顺。

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？）

不划算，过于注重细节不利于完整考虑代码的正确性。

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？

前者会增加大量代码，后者会增加代码崩溃或被攻击的可能性。在大型完整的项目中更多地使用前者，而在小规模的自测试或开源测试中尽量用后者保证开发进度。

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？

我目前不清楚这之间的平衡关系。

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？

Print大法好。

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？

不是，可能在细节部分由于测试的复杂度不够导致为充分测试。

1. Debug一个错误的程序，有乐趣吗？

没有

1. 关于本实验的工作量、难度、deadline。

本次饰演的工作量不算特别大，但是难度极高。

1. 到目前为止你对《软件构造》课程的评价和建议。

一般

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。

老师上课节奏与实验节奏脱节严重，上课PPT全英文，也没有教材，自学难度极高，我很难保证最后的考试是按照何种节奏出卷，更不知道没有教材且PPT难以阅览的情况下如何预习、复习知识点。简而言之，我感觉自己上了“课堂”、“自学”、“实验”三个不同的课程，对于我这种资质较为普通的学生压力巨大。

建议：至少在考前提供中文或译文版的复习资料，使学生能够明确考试节奏。