

**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 | 夏铭远 |
| 学号 | 1173710209 |
| 班号 | 1737102 |
| 电子邮件 | [462272701@qq.com](mailto:462272701@qq.com) |
| 手机号码 | 13029878656 |

**目录**

[1 实验目标概述 1](#_Toc3922818)

[2 实验环境配置 1](#_Toc3922819)

[3 实验过程 1](#_Toc3922820)

[3.1 Error and Exception Handling 2](#_Toc3922821)

[3.2 Assertion and Defensive Programming 2](#_Toc3922822)

[3.2.1 checkRep()检查invariants 2](#_Toc3922823)

[3.2.2 Assertion保障pre-/post-condition 2](#_Toc3922824)

[3.3 Logging 2](#_Toc3922825)

[3.3.1 写日志 2](#_Toc3922826)

[3.3.2 日志查询 2](#_Toc3922827)

[3.4 Testing for Robustness and Correctness 2](#_Toc3922828)

[3.4.1 Testing strategy 2](#_Toc3922829)

[3.4.2 测试用例设计 2](#_Toc3922830)

[3.4.3 测试运行结果与EclEmma覆盖度报告 2](#_Toc3922831)

[3.5 SpotBugs tool 2](#_Toc3922832)

[3.6 Debugging 3](#_Toc3922833)

[3.6.1 理解待调试程序的代码思想 3](#_Toc3922834)

[3.6.2 发现并定位错误的过程 3](#_Toc3922835)

[3.6.3 如何修正错误 3](#_Toc3922836)

[3.6.4 结果 3](#_Toc3922837)

[4 实验进度记录 3](#_Toc3922838)

[5 实验过程中遇到的困难与解决途径 3](#_Toc3922839)

[6 实验过程中收获的经验、教训、感想 4](#_Toc3922840)

[6.1 实验过程中收获的经验和教训 4](#_Toc3922841)

[6.2 针对以下方面的感受 4](#_Toc3922842)

# 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

在Eclipse中新建项目Lab4\_1173710209，首先把lab3的内容拷贝进来，再将项目与Github仓库连接并push到仓库里。

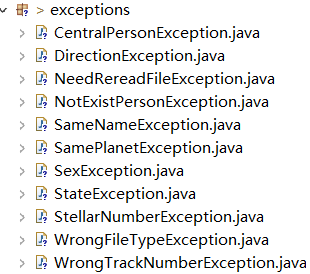
https://github.com/ComputerScienceHIT/Lab4-1173710209.git

# 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！）。

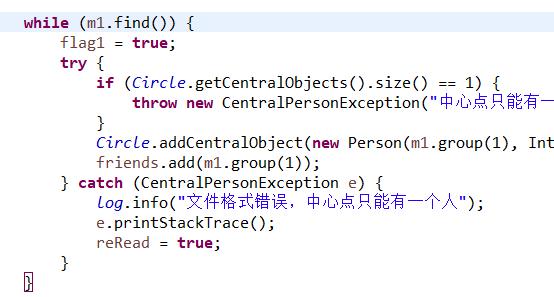
## Error and Exception Handling

在ReadFile方法中每一处文本解析都加了异常处理，创建了这些异常



设立标志变量reRead，只要捕捉到异常，就将其设置为true，以便判断是否需要重新读取文件，并添加到日志。

一处例子：



## Assertion and Defensive Programming

### checkRep()检查invariants

每个ADT都根据RI编写checkRep()

以Person为例



### Assertion保障pre-/post-condition

pre-condition如果有误则出异常

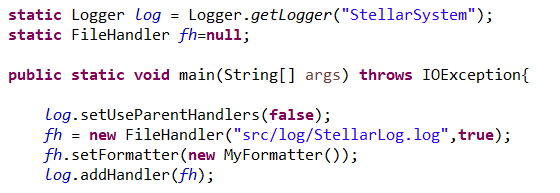
在每个mutator方法后面都调用checkRep进行post-condition检查

## Logging

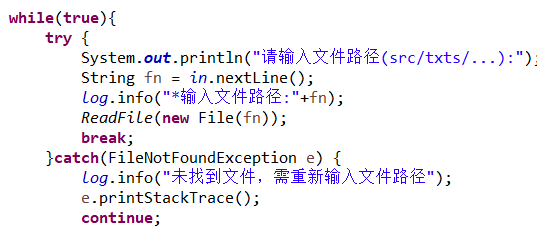
### 写日志

先在每个应用中建立logger对象，在每次捕捉异常和用户进行操作时都添加一条info输出到日志文件中，以StellarSystem为例

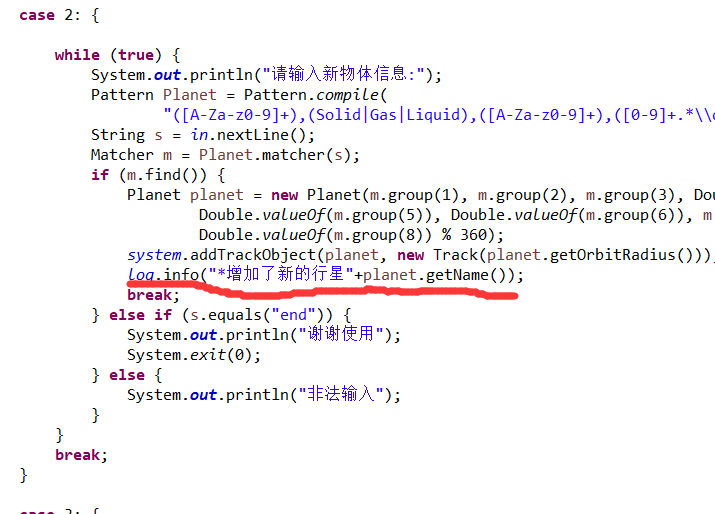
建立日志，文件为StellarLog.log



捕捉异常

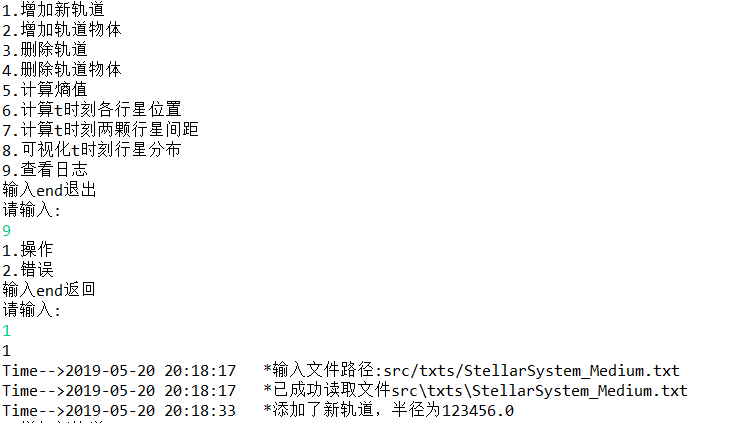


操作



### 日志查询

在添加日志时把操作类的日志前加了星号（\*），便于区分，查询时用户可进行筛选，然后将文件中的内容输出到控制台

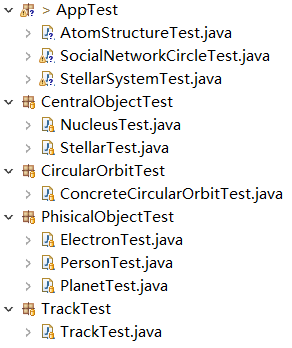


## Testing for Robustness and Correctness

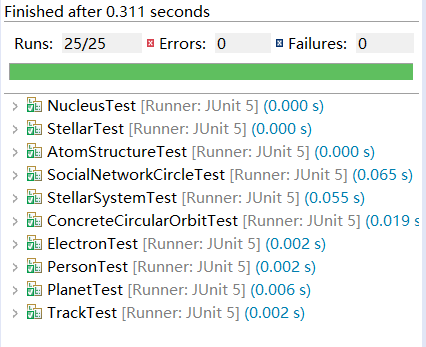
### Testing strategy

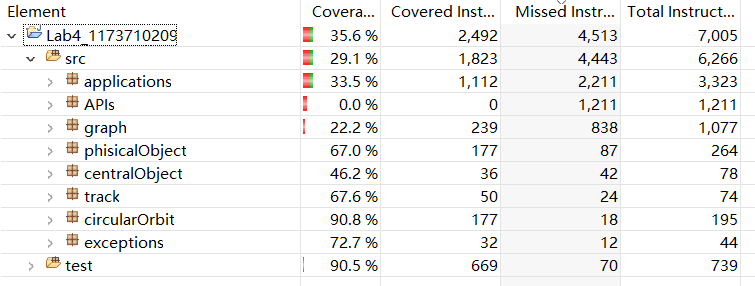
测试最底层的ADT以及三个APP

### 测试用例设计



### 测试运行结果与EclEmma覆盖度报告





由于大量代码都是客户端，而只测了方法，所以覆盖率较低



## 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。

## Debugging

### 理解待调试程序的代码思想

1. FindMedianSortedArrays()

思想：

采用分治的思想，取size小的作为数组A（size=m），另一个为数组B（size=n）。

当m+n为奇数时，存在Ai+BhalfLen-i,取max即中间值；

当m+n为偶数时，存在Ai/Ai+1/BhalfLen-i/BhalfLen-i-1，使得 (max{left}+min{right})/2即中间值。

1. RemoveComments()

思想：

按行读入每一行字符后，将其转换为字符数组。

分别对“/\*…\*/”和“//…”进行处理。

1. TopVotedCandidate()

思想：

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)。

需要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

1. FindMedianSortedArrays()

（1）halfLen = (m + n) / 2

设置断点，观察变量取值，发现对于halfLen的计算存在问题，取中间位时，存在小数需要进位。

（2）if ((m + n + 1) % 2 == 1)

观察代码，发现当m+n为奇数时((m+n+1)%2==0)，返回一个中间值即可;为偶数时((m+n+1)%2==1)，才会取中间两个数的平均数。

1. RemoveComments()

（1）List<String> ans = new List();

语法错误，静态检查报错。

（2）缺少对“//…”的处理

观察代码，发现缺少对该情况的处理。

（3）if (inBlock && newline.length() > 0)

!inBlock时（即不在“/\*…\*/”状态内，用于处理“//…”），才会转入此状态。

1. TopVotedCandidate()

（1）A = new ArrayList();

Warning，静态检查给出。

（2）int c = count.getOrDefault(p, 1);

设置断点，观察变量取值，c应该取当前p的得票数＋1；若不存在p，得票数取0，再加1。

（3） lo = mi;

二分法的两种情况，与下方代码同理，lo设置错误。

（4）int j = Math.max(lo, 0);

取下标值，与位置（第x位，下标为x-1）混淆。

### 如何修正错误

1. FindMedianSortedArrays()

**（1）int** iMin = 0, iMax = m, halfLen = (m + n + 1) / 2;

(m+n+1)保证进位，取到正确的下标值。

**（2）if** ((m + n + 1) % 2 == 0)

根据代码含义分析，当m+n为奇数时((m+n+1)%2==0)，返回一个中间值即可。

1. RemoveComments()
2. List<String> ans = **new** ArrayList<>();

语法错误改成即可，使用ArrayList<>的数据结构。

**（2）else** **if**(!inBlock && i + 1 < line.length() && chars[i] == '/' && chars[i + 1] == '/') {

**break**;

}

设置对“//…”情况的判断。

**（3）if** (!inBlock && newline.length() > 0)

根据代码含义，此处判断条件设置为“!inBlock”。

1. TopVotedCandidate()

（1）A = **new** ArrayList<>();

语法问题，设置<>泛型，传入List<Vote>。

(2)Map<Integer, Integer> count = **new** HashMap<>();

语法问题，设置<>泛型，传入<Integer,Integer>。

**（3）int** c = count.getOrDefault(p, 0)+1;

存在此候选人，取当前票数+1；不存在，则设当前票数为0，再+1.

（4）lo = mi+1;

lo取中间位+1。

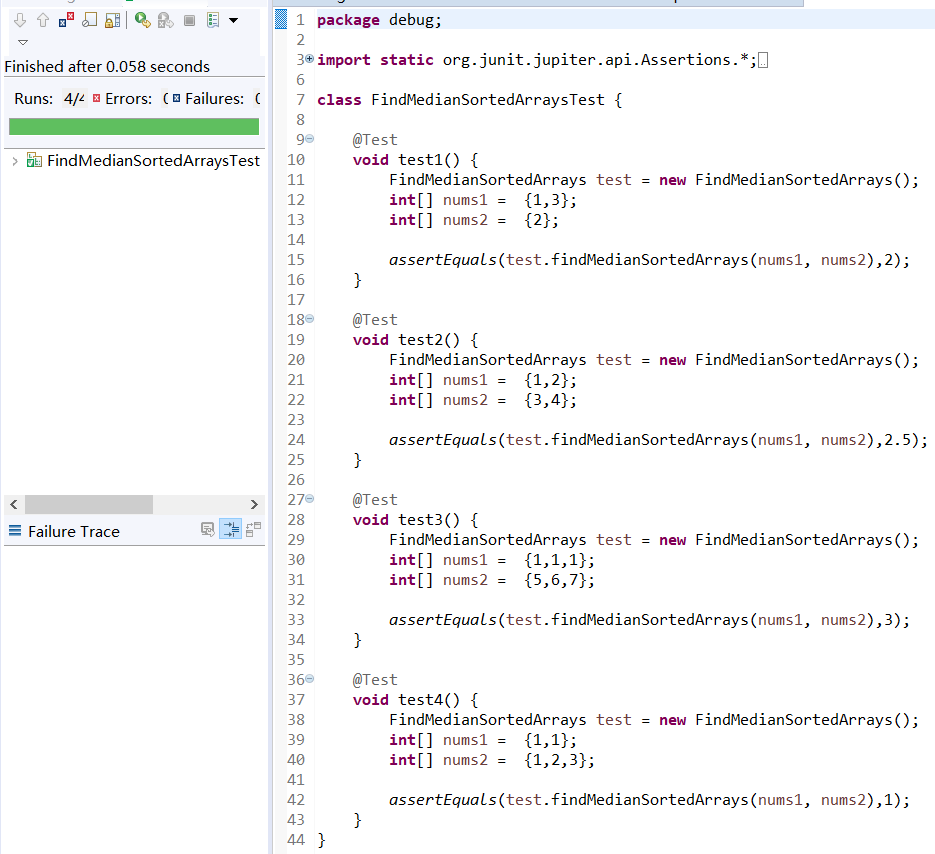
**（5）int** i = lo-1;

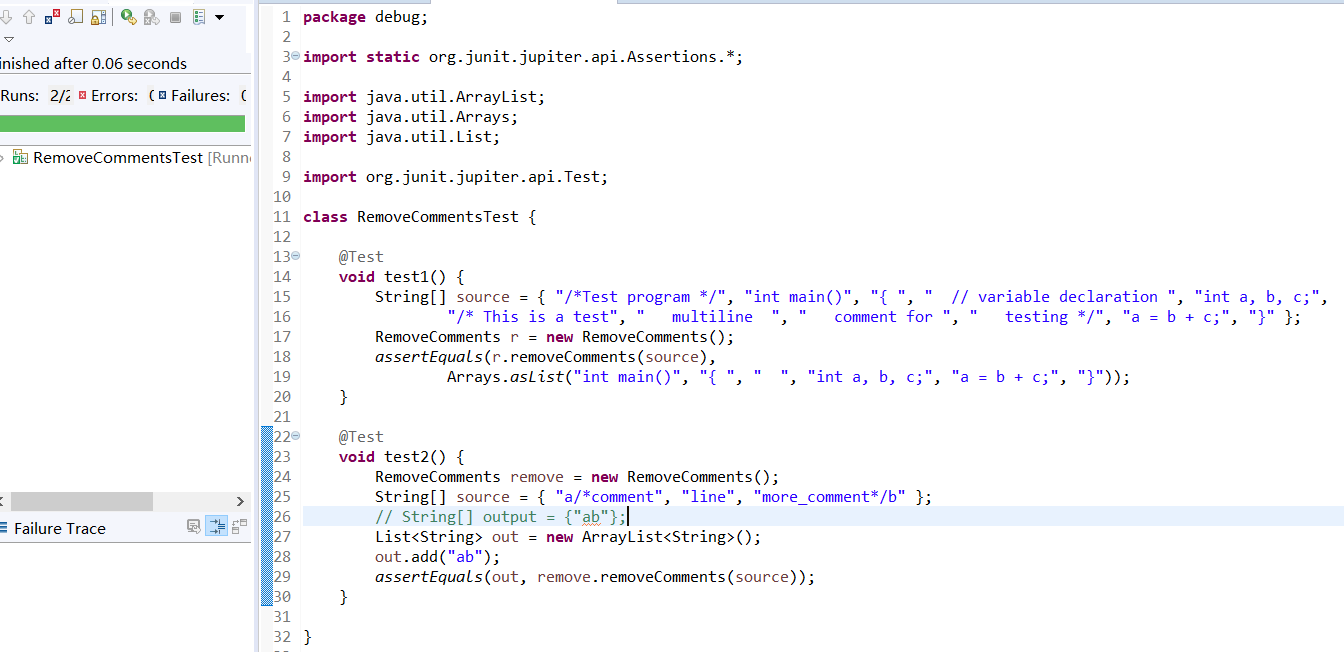
i作为下标，取lo-1。

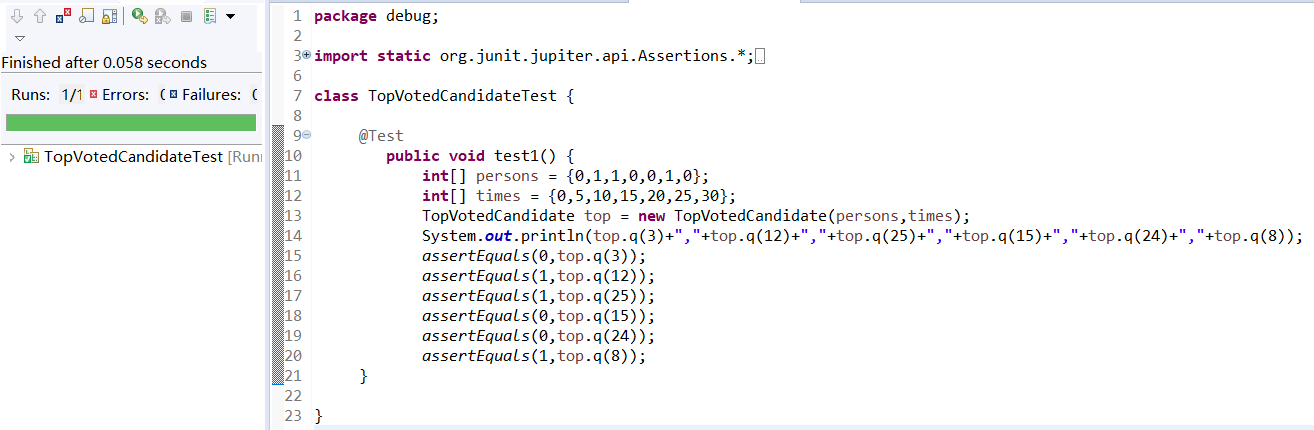
**（6）int** j = Math.*max*(lo-1, 0);

j作为下标，取lo-1。

### 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 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
|  |  |
|  |  |
|  |  |

# 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？
2. 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？）
3. “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？
4. 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？
5. 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？
6. 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？
7. Debug一个错误的程序，有乐趣吗？
8. 关于本实验的工作量、难度、deadline。
9. 到目前为止你对《软件构造》课程的评价和建议。
10. 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。