

**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 | 张钟升 |
| 学号 | 1170300431 |
| 班号 | 1703004 |
| 电子邮件 | [1170300431@stu.hit.edu.cn](mailto:1170300431@stu.hit.edu.cn) |
| 手机号码 | 13029801568 |

**目录**

[1 实验目标概述 1](#_Toc9189058)

[2 实验环境配置 1](#_Toc9189059)

[3 实验过程 2](#_Toc9189060)

[3.1 Error and Exception Handling 2](#_Toc9189061)

[3.2 Assertion and Defensive Programming 4](#_Toc9189062)

[3.2.1 checkRep()检查invariants 4](#_Toc9189063)

[3.2.2 Assertion保障pre-/post-condition 4](#_Toc9189064)

[3.3 Logging 4](#_Toc9189065)

[3.3.1 写日志 4](#_Toc9189066)

[3.3.2 日志查询 4](#_Toc9189067)

[3.4 Testing for Robustness and Correctness 5](#_Toc9189068)

[3.4.1 Testing strategy 5](#_Toc9189069)

[3.4.2 测试用例设计 5](#_Toc9189070)

[3.4.3 测试运行结果与EclEmma覆盖度报告 5](#_Toc9189071)

[3.5 SpotBugs tool 6](#_Toc9189072)

[3.6 Debugging 6](#_Toc9189073)

[3.6.1 理解待调试程序的代码思想 6](#_Toc9189074)

[3.6.2 发现并定位错误的过程 6](#_Toc9189075)

[3.6.3 如何修正错误 7](#_Toc9189076)

[3.6.4 结果 8](#_Toc9189077)

[4 实验进度记录 9](#_Toc9189078)

[5 实验过程中遇到的困难与解决途径 9](#_Toc9189079)

[6 实验过程中收获的经验、教训、感想 9](#_Toc9189080)

[6.1 实验过程中收获的经验和教训 9](#_Toc9189081)

[6.2 针对以下方面的感受 9](#_Toc9189082)

# 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。

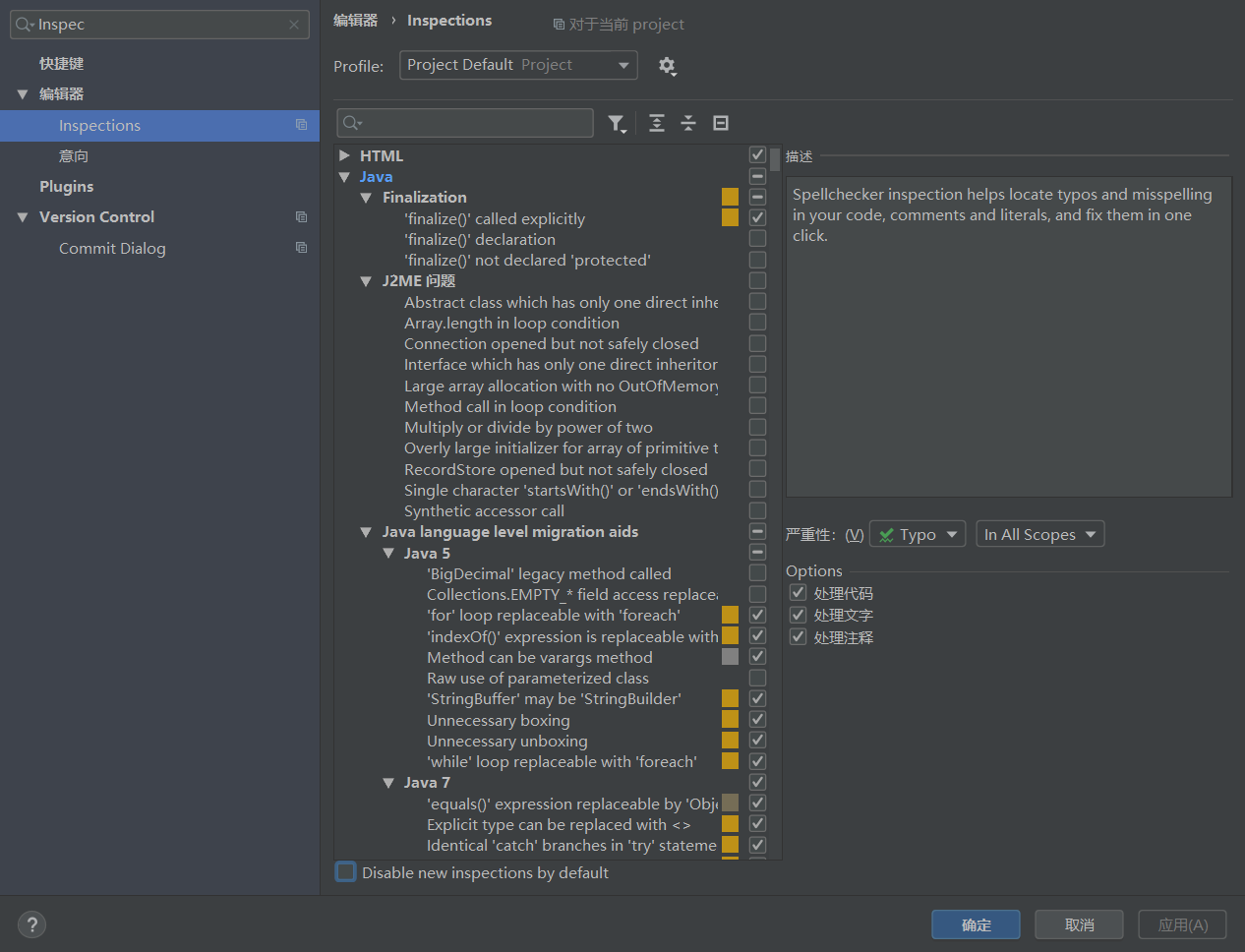
实验针对 Lab 3 中写好的 ADT 代码和基于该 ADT 的三个应用的代码，使用 以下技术进行改造，提高其健壮性和正确性：

* 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

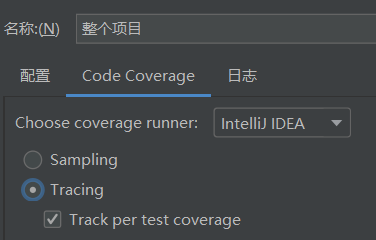
* 有关SpotBugs

由于开发环境是IDEA, 不能使用SpotsBugs; 尝试使用FindBugs失败; 故用ItelliJ IDEA静态检查替代. 如图, 当检查级别使用Inspections时, 能够检查大部分的NullPointer, unchecked, rawtypes等可能存在的bug.



* 有关EclEmma

IDEA不支持EclEmma, 有关Coverage工具, 可以使用ItelliJ IDEA本身的Coverage工具或是JaCoCo.



repo: <https://github.com/ComputerScienceHIT/Lab4-1170300431.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

重载了两个Exception, ExceptionGroup和LogicErrorException.

* ExceptionGroup

1. **public** **class** ExceptionGroup **extends** RuntimeException **implements** Iterable<Exception>{
2. **private** List<Exception> exs;
4. **public** **boolean** join(Exception ex);
6. @Override
7. **public** String getMessage();
9. @Override
10. **public** String getLocalizedMessage();
12. @Override
13. **public** **void** printStackTrace();
15. **public** **int** size();
17. **public** Exception get(**int** index);
19. **public** **boolean** merge(ExceptionGroup other);
21. **public** **void** clear();
23. **public** **boolean** isEmpty();
25. @Override
26. **public** Iterator<Exception> iterator();
27. }

接口和方法如上. 显然, ExceptionGroup是作为一个Exception的容器出现的, 它主要适用于从文件读取的时候, 汇集输入错误信息.

* LogicErrorException

参考C++的logic\_error....主要用于汇报用户输入中产生的逻辑异常, 如前后矛盾等等. 它继承RuntimeException.

* IllegalArgumentException

程序中主要的错误就是参数错误, 比如不能将字符串转为数字, 抛出NumberFormatException, 它是IllegalArgumentException的子类; 不能转为枚举, 直接抛出IllegalArgumentException; 除此之外, 任何输入的参数不合法均可视为IllegalArgumentException并加入ExceptionGroup实例化的对象, 并最后抛出它.

## Assertion and Defensive Programming

### checkRep()检查invariants

在实验3中,checkRep已上传到源代码.

首先CircularOrbit留出接口checkRep, 并由子类StellarSystem, AtomStructure, SocialNetworkCircle实现该接口. 对StellarSystem, 需要检查两颗相邻行星半径小于它们的两条轨道间的距离差. 这里距离差也就是高轨道短半轴减低轨道长半轴. 另外发现似乎有行星运行在恒星内部.... 对于AtomStructure, 给一个空实现... 对SocialNetworkCircle, 利用已经存在的getLogicalDistance, 计算每个User与`中心用户的逻辑距离, 断言它等于该User的轨道半径. (与中心用户不连通的点轨道半径为-1).

### Assertion保障pre-/post-condition

使用assert和annotation分别保证运行时和静态检查. 使用assert的情况, 比如setRelation函数, 可以为PhysicalObject a和b设置关系. 此时用assert !a.equals(b)来保证a和b不相等就是应当的. 再如addTrack函数, 应当断言传入的参数个数大于等于1. 除此之外assert更多的用于查询(CircularOrbit.query, Collection.get)后的NotNull保证.

## Logging

### 写日志

用Java Logger写日志. 比较值得注意的有:

* 添加FileHandler以输出到文件;
* 重载Formatter以控制输出格式;
* getLogger得到的Logger对象都有父对象LoggerManager.root(可能吧), 这个Logger有默认的ConsoleHandler. 获取这个Handler并置Level.OFF就可以实现静默输出日志;
* 如果希望用附加的方式打开文件, 可以用相关的构造函数;
* 要注意FileHandler是应当关闭的.

### 日志查询

1. 实现两个parser类来解析info和warning的文本.
2. 分别读取两个log文件, 用两个parser得到List<List>类型的参数矩阵.
3. 用户输入查询类型和查询条件, 查找符合条件的List, 将这些参数join后就还原回了日志记录, 将这些日志记录显示出来即可.

## Testing for Robustness and Correctness

### Testing strategy

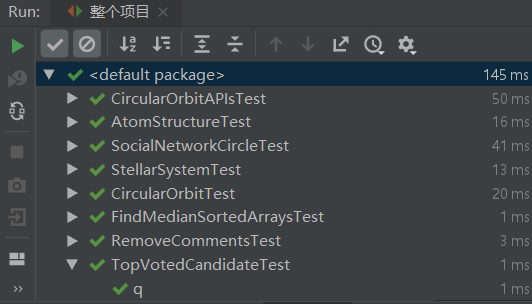
根据等价类和边界条件设计testing strategy. 要单独注意的是对抛出异常的条件测定, 有的异常并不容易出现, 尽可能地复现大多数的异常. 借助coverage工具可以发现未被覆盖的分支, 反复分析, 增加合适的测试用例.

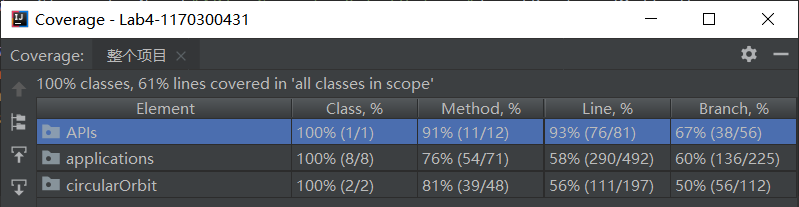
### 测试用例设计

对大多数返回布尔值的修改操作, 比如addObject, addTrack, 可以调用多次, 一次使它返回真, 多次返回假(操作成功往往只有一种情况, 而操作失败可能有多种情况), 往往可以完全覆盖. 一些observer方法, 如findTrack, isEmpty一类的方法, 根据它们所获取的对象类型设计测试用例. 对CircularOrbitAPI中的方法, 事先确保其所调用的方法的测试通过, 再设计一个或多个情形作为测试用例.

值得注意的有两点, 一是一些GUI操作(包括main函数)无法测试; 二是对于一些protected和private方法, 一般情况下测试模块无法直接测试, 可能的解决方案有两种. 一是测试他们的公有的调用者; 当公有的调用者不便测试, 或者方法本身的功能比较独立, 对外部条件依赖不大的时候, 可以用反射的方式直接调用, 完成测试.

### 测试运行结果与EclEmma覆盖度报告



[[1]](#footnote-1)

## SpotBugs tool

主要的, 有影响的bug是NullPointerWarning. 由于各种原因将Nullable表达式直接传给要求NotNull的参数, 或者是调用Null对象的方法. 对此大多数可以加入if检查, 或者是assert not null. 其次是modifier warning, 因为个人习惯, 大多数接口被声明为public... 实际上有些可以是private或package-private. 当然有些为了反射使用, 静态检查不能识别, 可以SupressWarning... 其次是rawtype和unchecked warning, 主要出现于对范型变量, 直接使用它的裸类型. 有些是有意这样做的, 故同样抑制异常...

## Debugging

### 理解待调试程序的代码思想

* FindMedianSortedArrays

看介绍可以很容易地得知是求两个数组的并的中位数. 算法并不显然, 但仔细观察可以发现是的思想.

* RemoveComments

看介绍显然是去掉代码中的注释. 扫描整个输入串1次即可.

* TopVotedCandidate

看介绍并不显然. 仔细观察发现是给一段时间内的投票情况, 查询某一时间点的领先者. 代码本身也很令人困惑, 但发现是用二分查找后变得相对容易理解.

### 发现并定位错误的过程

* FindMedianSortedArrays

首先显然是个二分查找, 那么这句就变得很明显.而为了保证, j应该是, 调试, 测试通过.

* RemoveComments

首先是开关量置之后, 如果不把读取指针后移一位的话, 就会多读入一个. 那么, 为了统一其实inBlcok后也可以加一个i++(没有影响). 其次是单行注释, 给的代码没考虑这一点, 补全. 开调试器, 发现什么也没加进去... 检查是if条件中inBlock反了, 取反即可.

* TopVotedCandidate

首先猜测A这个数据结构代表了什么. 反复观察构造函数和q(), 发现唯一的线索是Map count这个变量. 反复考虑, 从count中取出的c应当每次自增. 此时测试不通过. 继续观察, 发现q中的while其实是二分法. 第一个while寻找对应的time所在的行数(次数), 第二个在对应行中找小于等于该时间最大的vote, 输出它的投票对象. 据此修改, 测试通过.

### 如何修正错误

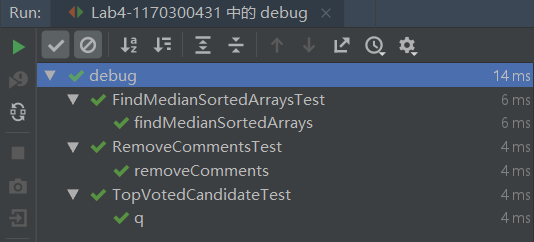
* FindMedianSortedArrays
* RemoveComments

1. **while** (i < line.length()) {
2. **if** (!inBlock && i+1 <= line.length() && chars[i] == '/' && chars[i+1] == '\*') {
3. inBlock = **true**;
4. i++;
5. } **else** **if** (inBlock && i+1 <= line.length() && chars[i] == '\*' && chars[i+1] == '/') {
6. inBlock = **false**;
7. i++;
8. } **else** **if** (!inBlock && i+1 <= line.length() && chars[i] == '/' && chars[i+1] == '/') {
9. **break**;
10. }
11. **else** **if** (!inBlock) {
12. newline.append(chars[i]);
13. }
14. i++;
15. }
16. **if** (!inBlock && newline.length() > 0) {
17. ans.add(**new** String(newline));
18. }

* TopVotedCandidate

1. **int** c = count.getOrDefault(p, 0) + 1;
3. count.put(p, c);
4. **while** (A.size() < c)
5. A.add(**new** ArrayList<>());
6. A.get(c - 1).add(**new** Vote(p, t));
7. **public** **int** q(**int** t) {
8. **int** lo = 0, hi = A.size();
9. **while** (lo < hi) {
10. **int** mi = lo + (hi - lo) / 2;
11. var lmi = A.get(mi);
12. **if** (lmi.get(0).time <= t && t <= lmi.get(lmi.size() - 1).time) {
13. lo = mi;
14. **break**;
15. }
16. **else** **if**(lmi.get(0).time > t)
17. hi = mi - 1;
18. **else** **if**(lmi.get(lmi.size() - 1).time < t)
19. lo = mi + 1;
20. }
21. **int** i = lo;
23. lo = 0;
24. hi = A.get(i).size();
25. **while** (lo < hi) {
26. **int** mi = lo + (hi - lo) / 2;
27. var vmi = A.get(i).get(mi);
28. **if** (vmi.time < t)
29. lo = mi + 1;
30. **else** **if**(vmi.time == t) {
31. lo = mi;
32. **break**;
33. }
34. **else** hi = mi - 1;
35. }
36. **int** j = Math.max(lo, 0);
37. **return** A.get(i).get(j).person;
38. }

### 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 |
| 5-13 |  | 改变了Lab3中实现方式(反射) | 完成 |
| 5-14 |  | 美化GUI | 完成 |
| 5-14 |  | 静态检查, 解决或抑制某些警告 | 一部分 |
| 5-15 |  | 重写SocialNetworkCircle的方法, 使它能保证不变性 | 完成 |
| 5-16 |  | 增加和重写test | 一部分 |
| 5-16 | ? - 23:50 | 3.6 debugging的部分 | 完成 |
| 5-17 |  | log查询GUI | 大致完成 |
| 5-18 |  | 简化代码. 增加test. 重新实现exception. | 大致完成 |
| 5-19 |  | 最后的某些处理, 处理了某些随机出现的bug. | 可能完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| IDEA不能安装SpotBugs和FindBugs | 用IDEA inspections大致代替... |
| 测试覆盖率始终不够 | private和protected方法可以用反射调用 |
| log一直输出到控制台 | Logger::getParent().getHandlers()[0].setLevel(Level.OFF). 详见3.3.1. |

# 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？

正确性是程序的基础, 任何程序都要考虑正确性. 健壮性是要额外考虑的内容, 大多数程序都要考虑健壮性, 而且适当考虑健壮性对开发过程有益处.

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.

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？

用户承担意味着使用困难和开发过程的相对容易. 客户端承担意味着使用方便但开发复杂. 当用户是开发人员的时候往往可以由用户承担. 当用户面向大众的时候一定要由客户端承担.

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？

没有. 对于消耗巨大的defensive措施, 最好的办法就是换用其它低消耗的措施, 即便它是”更弱的defensive”.

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？

条件断点配合assert往往能发现隐藏的bug. assertion在复杂工程中查找错误非常有用. log在排除问题中的作用是公认且显而易见的.

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？

充分的测试可能是涵盖了每一个等价类和边界条件的测试. 不一定.

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

1. JaCoCo Coverage. [↑](#footnote-ref-1)