

**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 | 强文杰 |
| 学号 | 1171000410 |
| 班号 | 1703005 |
| 电子邮件 | [672334335@qq.com](mailto:672334335@qq.com) |
| 手机号码 | 18800421389 |

**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 2](#_Toc3922820)

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

[3.2 Assertion and Defensive Programming 4](#_Toc3922822)

[3.2.1 checkRep()检查invariants 4](#_Toc3922823)

[3.2.2 Assertion保障pre-/post-condition 6](#_Toc3922824)

[3.3 Logging 8](#_Toc3922825)

[3.3.1 写日志 8](#_Toc3922826)

[3.3.2 日志查询 9](#_Toc3922827)

[3.4 Testing for Robustness and Correctness 10](#_Toc3922828)

[3.4.1 Testing strategy 10](#_Toc3922829)

[3.4.2 测试用例设计 11](#_Toc3922830)

[3.4.3 测试运行结果与EclEmma覆盖度报告 11](#_Toc3922831)

[3.5 SpotBugs tool 13](#_Toc3922832)

[3.6 Debugging 14](#_Toc3922833)

[3.6.1 理解待调试程序的代码思想 14](#_Toc3922834)

[3.6.2 发现并定位错误的过程 15](#_Toc3922835)

[3.6.3 如何修正错误 16](#_Toc3922836)

[3.6.4 结果 17](#_Toc3922837)

[4 实验进度记录 18](#_Toc3922838)

[5 实验过程中遇到的困难与解决途径 19](#_Toc3922839)

[6 实验过程中收获的经验、教训、感想 19](#_Toc3922840)

[6.1 实验过程中收获的经验和教训 19](#_Toc3922841)

[6.2 针对以下方面的感受 19](#_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： 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

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。

1. 根据助教在群里发的链接，完成Eclipse和JDK11的安装，然后去网上查询Java运行环境的配置，配置计算机的环境变量。

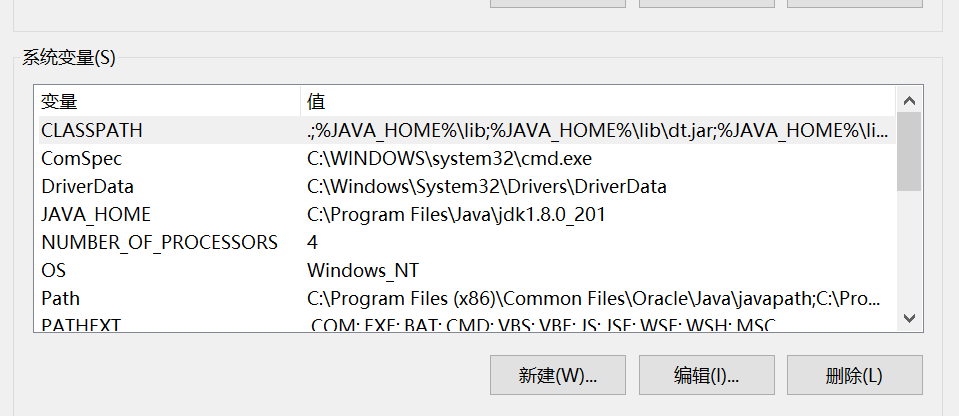


图2-1 环境变量配置

2.授权GitHub classroom，获取lab4的GitHub classroom的URL地址。

3. Git代码配置管理。首先安装最新版Git，获取个人GitHub的URL地址，通过在Git bash输入指令建立管理本地仓库，将文件push到远程仓库。

git push origin master

4. SpotBugs的安装：Eclipse MarketPlace

在这里给出你的GitHub Lab4仓库的URL地址（Lab4-学号）。

https://github.com/ComputerScienceHIT/Lab4-1171000410.git

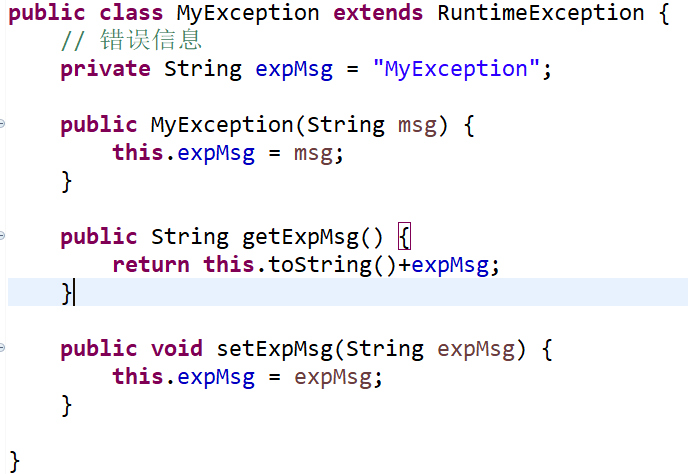
# 实验过程

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

**1.自定义Excetion类型**

首先自定义异常类MyException继承RuntimeException，如果继承自Exception则需要明确对抛出的Exception进行声明，这里选择RuntimeException，与直接继承Exception相比不需要显示抛出（函数声明throws...）。



对于异常来说，最重要的部分就是类名。因此我们在建立很多特定异常类，表达异常的信息，继承自MyException即可

依赖关系异常：**public** **class** DependencyException **extends** MyException

语法异常：**public** **class** GrammarException **extends** MyException

Label类型异常：**public** **class** LabelException **extends** MyException

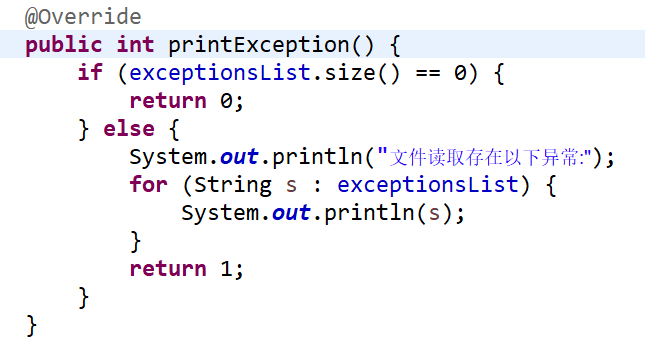
Number数字的异常：**public** **class** NumberException **extends** MyException

**2. 异常处理的大体思想**

根据实验的要求，我们异常处理之后，需要将不法之处汇总起来提示给用户，这意味着我们需要在ConcreteCircularOrbit定义域：

**private** ArrayList<String> exceptionsList = **new** ArrayList<String>(); // 存储异常的集合

并且有如下打印异常的函数，这种处理也是提高复用性：



对于AtomStructure和SocialNetworkCircle，因为处理的特殊性（如SocialNetworkCircle需要先读SocialTie后读Friend），我一次将txt读入，因此在异常处理的时候，需要有多处的try catch语句，try中一旦判断异常情况，即throws new …异常，然后catch捕获异常，并且getExceptionsList().add(e.getExpMsg()); 添加到异常的集合中。

对于StellarSystem，因为我采用了按行读取文件的方式，因此匹配异常的时候一个try语句模块，多个catch，然后while循环读入多个行即可，try中一旦判断异常情况，即throws new …异常，然后catch捕获异常，并且getExceptionsList().add(e.getExpMsg()); 添加到异常的集合中。

**3.对于以下做出的非法情况的异常处理，截出提示给用户的不法情况汇总即可说明。**

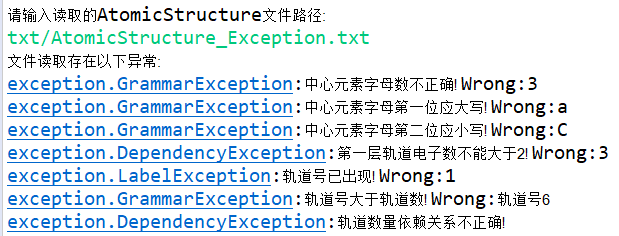
其中对于异常类名的解释：

输入文件中存在不符合语法规则的语句：GrammarException和NumberException

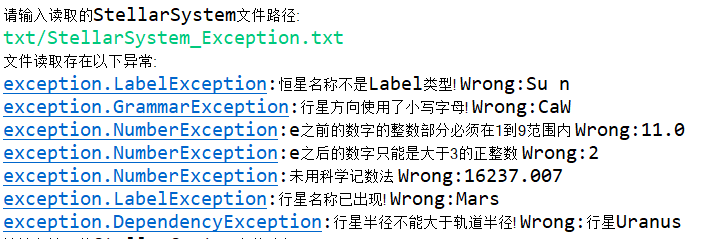
存在标签完全一样的元素：LabelException

文件中各元素之间的依赖关系不正确：DependencyException

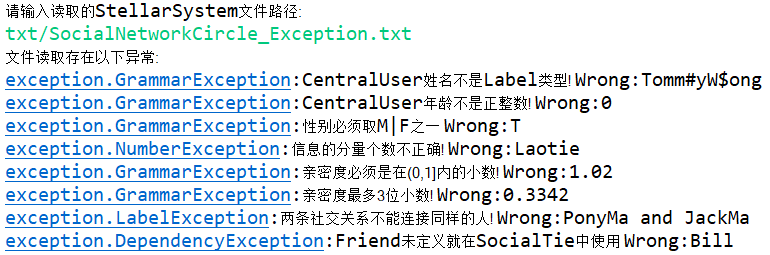
AtomStructure：



StellarSystem：



SocialNetworkCircle:



## Assertion and Defensive Programming

### checkRep()检查invariants

针对每个类，我们根据lab3中的rep invariants (RI)撰写checkRep()，但是有些类中没有fields，因此无法写RI，从而没有该类的checkRep()。

以下针对各类的checkRep()进行介绍：

**1.Difference类**

特定轨道物体数目差异和轨道物体性质差异的集合大小一定相等。

**private** **void** checkRep() {

**assert** specificTracksNumDifference.size() == objectsDifference.size() ;

}

**2.centralObject包中的类**，保证中心物体的特定属性符合文件中的，不为空，大于0

**assert** name != **null**;

**assert** centralAge > 0;

**assert** centralGender != **null**;

**assert** stellarRadius > 0;

**assert** stellarQuality > 0;

**3.SocialNetworkCircle**

不管社交关系如何增加或删除，第i层轨道上的人与中心点的人之间的最短路径等于i。

朋友集合大小n1，社交关系数目n2

assert n1\*n1/2 >= n2

**4.AtomStructure**

不管轨道电子如何增加或减少，内层轨道总是不为空

**5.StellarSystem**

不管行星如何运动，两颗行星之间的距离总是大于两个行星半径之和；

行星自身的行星半径总是小于轨道半径。

**6.circularOrbit包中的类**

读文件时，有时候需要一些集合来存储信息，集合的大小大于0.

对于轨道上所有物体，无论它们如何运动，角度总是在[0,360]区间内

**private** **void** checkRep() {

**for**(E e:objectSitha.keySet()) {

**assert** objectSitha.get(e) >=0 && objectSitha.get(e) <=360;

}

}

ConcreteCircularOrbit中轨道数目和轨道映射物体的数目一直相等。

中心体存在物体时，中心点物体不为空。

**private** **void** checkRep() {

**assert** tracks.size() == T2E.size();

**assert** centralObj != null;

}

**7.physicalObject包中的类**

保证轨道物体在变化的同时，它们的域始终满足文件中的要求，包括字符串不为空，年龄大于0，行星的行星半径小于轨道半径，行星的角度在[0,360]之间。

**8.relation包中的类**

对于表达轨道物体映射关系的图，由于是有向图，因此边的数目，和keys、values的数目相等，**private** **void** checkRep() {

**assert** keys.size() == values.size();

**assert** edges.size() == keys.size();

}

对于表达中心物体映射轨道物体的类，边的数目等于目标点的数目。

**private** **void** checkRep() {

**assert** edges.size() == values.size();

}

对于内部类LE\_Edge<K, V>，需要保证的是每条边的权值大于0

**9.Track**

track中只有radius一个field，因此保证radius大于0即可。

### Assertion保障pre-/post-condition

Pre-condition 前置条件（requires）是对客户端的约束，指的是在使用方法时必须满足的条件。

Post-condition 后置条件（effects）是对开发者的约束，指的是方法结束时必须满足的条件。

其实在写代码时，应该早注意以下几点，这样对前置条件和后置条件的处理会简单很多。契约：如果前置条件满足了，后置条件必须满足除非在后置条件中声明，否则方法内部不应该改变输入参数。尽量不设计mutating的spec，否则容易引发bugs。尽量避免使用mutable对象。

下面介绍对于大部分方法的前置条件和后置条件如何assert。

由于方法过多，以下分类列出通过 assertion或异常机制分别处理 pre-condition 和 post-condition的处理策略。

**一.对于relation包中的表示图的类，**

1.remove方法

pre-condition处理 ：**assert** edgeIndex != -1;

post-condition处理：**assert** initialSizeVertices - 1 == vertices.size();

**assert** initialSizeEdges != edges.size();

2.put方法

pre-condition处理：**assert** weight >= 0;

post-condition处理：**assert** sourcePrevWeight == targetPrevWeight;

assert key != value;

3.getAdjacentObjects方法

pre-condition处理：**assert** obj != **null**;

post-condition处理：**assert** adjcentObject.size() != 0;

4.getNumOfClosers

pre-condition处理：**assert** intimacy > 0 && intimacy <=1; //亲密度大小

post-condition处理：**assert** counter >= 0; //数目

**二. 对于APIS包中的类**

因为这里面的方法的参数大部分都是轨道和轨道物体，因此前置条件的assertion大体相同。几个方法返回的都是距离等数值型数据，因此后置条件判断返回值是否符合要求即可。

pre-condition处理：**assert** c != **null**;

**assert** e1 != **null**;

**assert** e2 != **null**;

post-condition处理：**assert** entropy >= 0;

**assert** distance >= 0;

**三. 对于ConcreteCircularOrbit**

**策略如下：**

1.涉及到半径作为参数，pre-condition要判断半径大小是否符合规范。

2.返回值是Track类型或list，要判断不为null，不为空。

3.为轨道添加、删除物体，添加删除轨道：需要判断传入的轨道和物体不为null，添加后轨道数或者轨道物体数会加一，删除后轨道数或轨道物体数会减一。

4.为中心物体和轨道物体添加关系，为轨道物体之间添加关系，需要判断亲密度的范围

具体示例（同类型的不在报告中赘述了）：

findTrack方法：

pre-condition处理：**assert** r >= 0;

post-condition处理：**assert** t != **null**;

removeTrack方法：

pre-condition处理：**assert** t != **null**;

post-condition处理：**assert** tracks.size() - 1 == oldSize;

addTrackObject方法：

pre-condition处理：**assert** track != **null**;

**assert** obj != **null**;

post-condition处理：**assert** oldSize +1 == list.size();

addCenterTrackObjectsRelation方法

pre-condition处理：**assert** intimacy>0 &&intimacy<=1;

post-condition处理：assert L2E.size() = oldSize +1;

对于其他的centralObject包、physicalObject包和Track的类型，由于里面没什么方法，因此pre-condition和post-condition无过多的处理，一些限制条件已在checkRep()中判断。

## Logging

### 写日志

1.首先获取调用日志记录类的名称

**public** **static** **final** Logger ***log*** = Logger.*getLogger*(MyOrbitScenes.**class**.getSimpleName());

2.在MyOrbitScences中初始化静态变量log，方便调用

**public** **static** **final** Logger ***log*** = Logger.*getLogger*(MyOrbitScenes.**class**.getSimpleName());

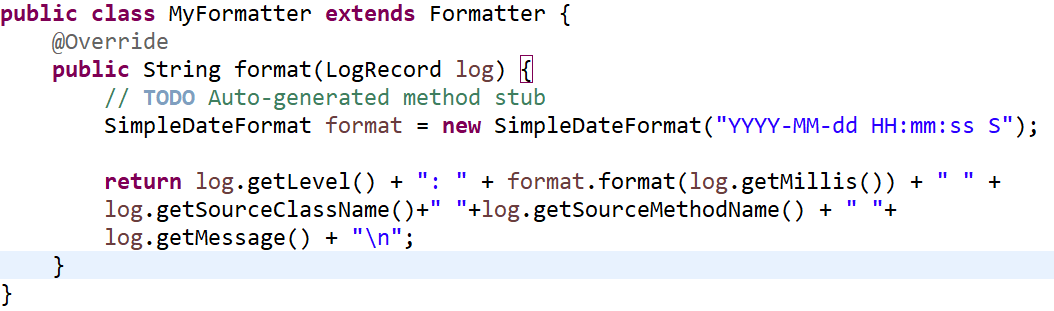
3.初始化一个FileHandler 指针，设置logger记录的信息不在控制台上输出

4.设置要写入的文件

FileHandler fileHandler = **new** FileHandler("log/info.log", **true**);

5.设置自己设置的写入文件的日志格式

fileHandler.setFormatter(**new** MyFormatter());



6.最后将此日志信息写入文件

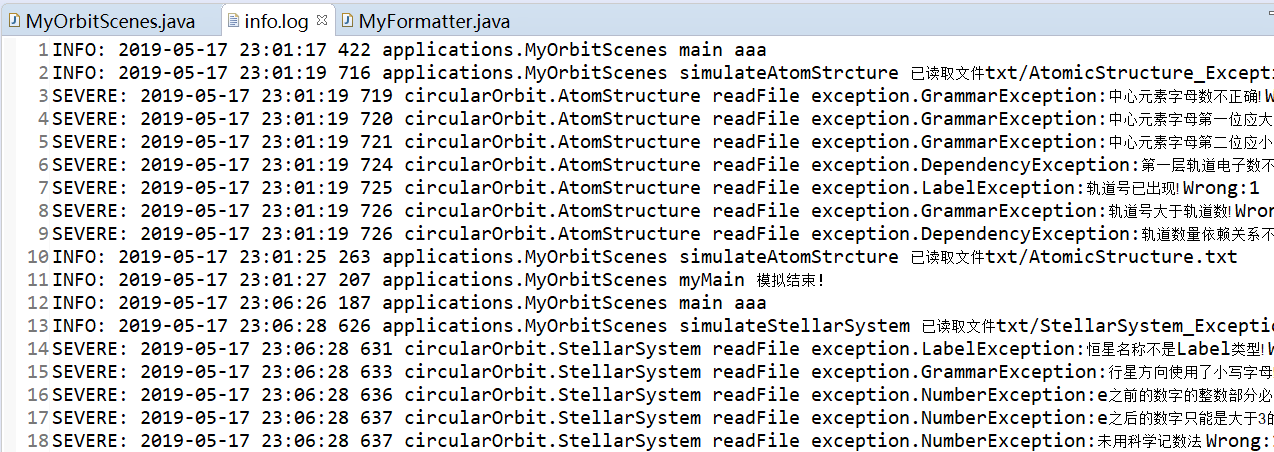
log.addHandler(fh);

log.info("需要用日志记录的信息");

MyOrbitScenes.***log***.log(Level.***SEVERE***, e.getExpMsg()); //在读文件的方法中记录异常信息

fileHandle.close();//关掉日志文本

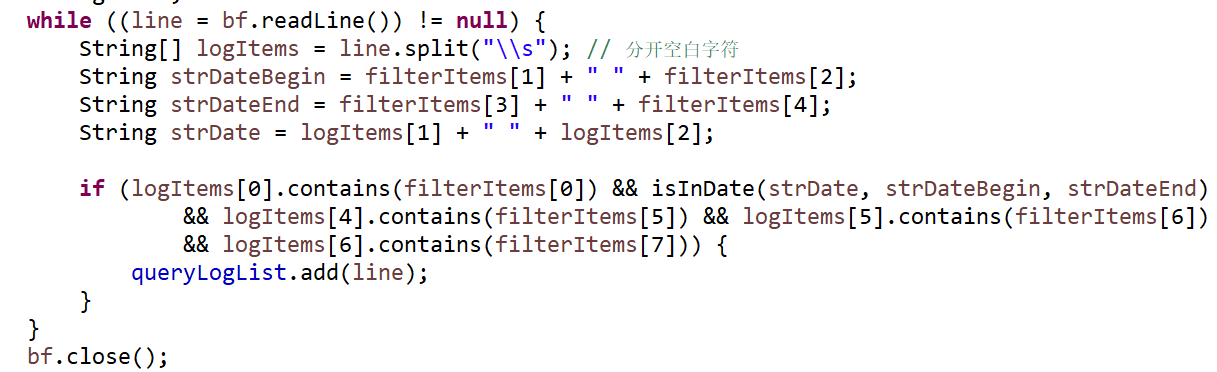
日志文件在log文件中提交，其中截图如下：



### 日志查询

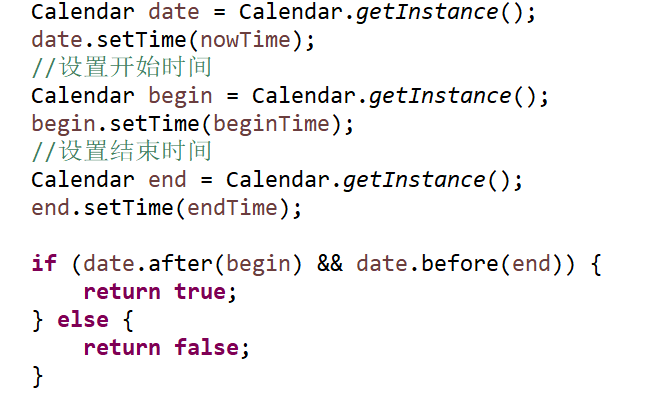
日志查询的过程就是根据用户输入的字符串，获取<类型，开始时间，结束时间，类，方法，操作类型>这些过滤条件，然后取匹配info.log文件，判断info.log中按行读取的文件，是否满足过滤条件，如果满足，保存到queryLogList，最终一次输出给用户。

匹配过程如下：



比较麻烦的是判断时间是否在用户输入的时间段中，我们先需要设置Date的Format SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

时间的然后把字符串转化为Date类型，最后比较时间即可。



最后附上查询的示例截图：

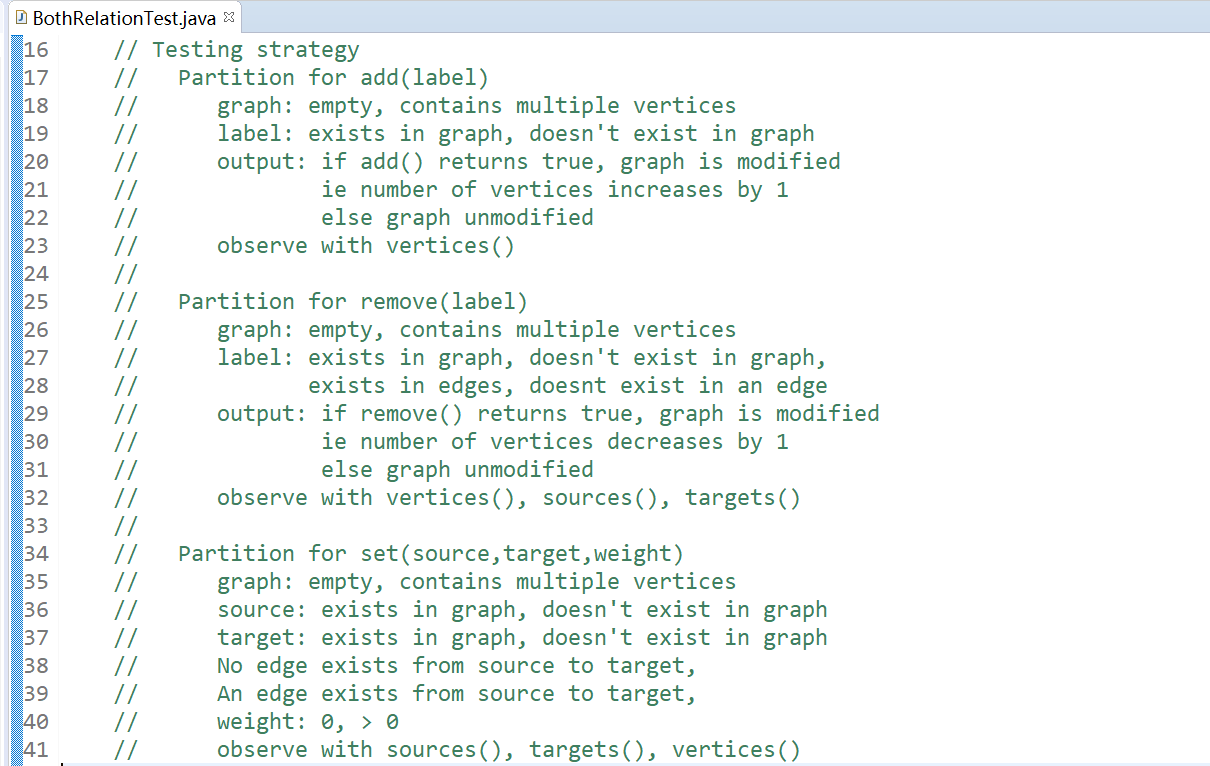


## Testing for Robustness and Correctness

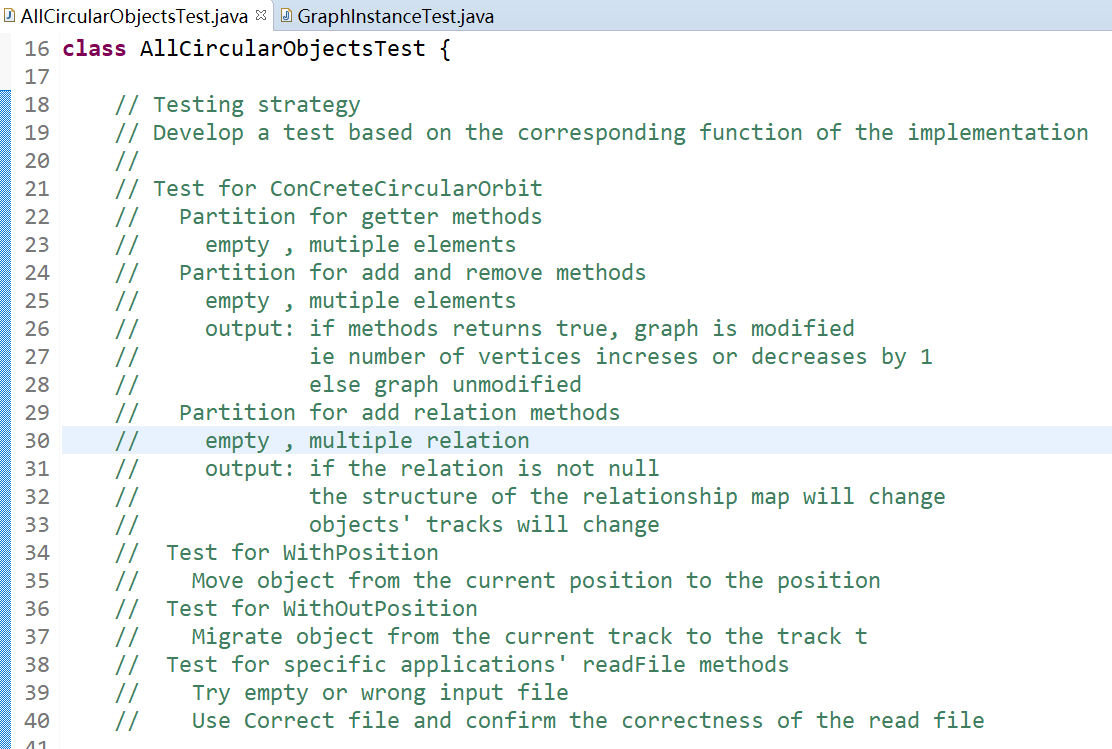
### Testing strategy

**具体的测试策略在每个测试文件的开头都已写明，以下给出几个Testing strategy的示例：**

**示例一. 关于relation的图的Testing strategy：**



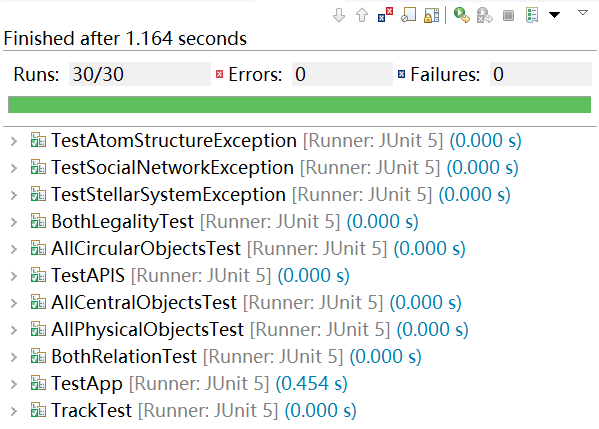
**示例二. 对于circularOrbit包中类的Testing strategy：**



### 测试用例设计

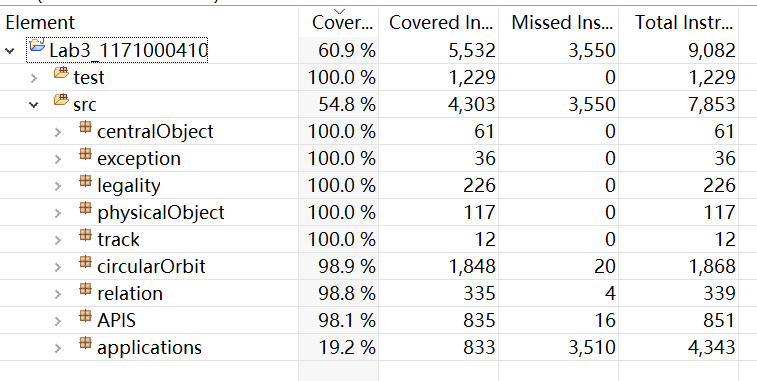
### 测试运行结果与EclEmma覆盖度报告

**测试运行结果：**

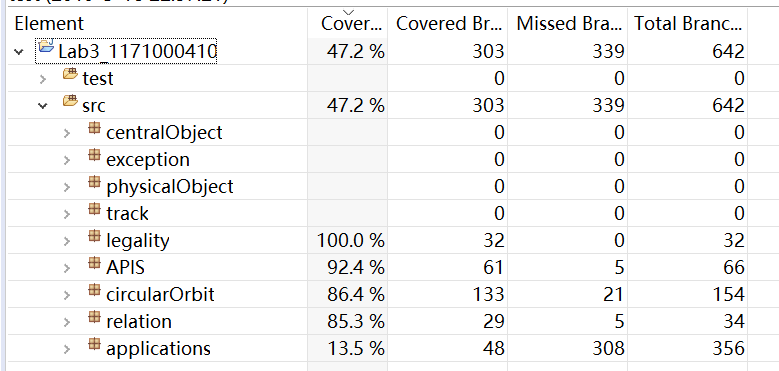


**注：对于applications中的应用类，画静态图的类和画同态图等没有进行测试，因此applications包中的测试覆盖率较低。**

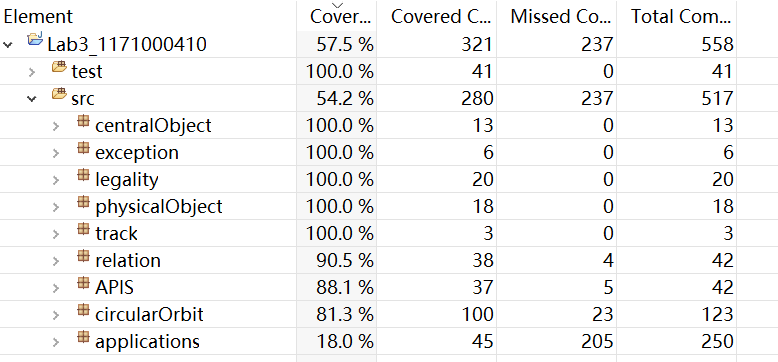
**Instruction Counters:**



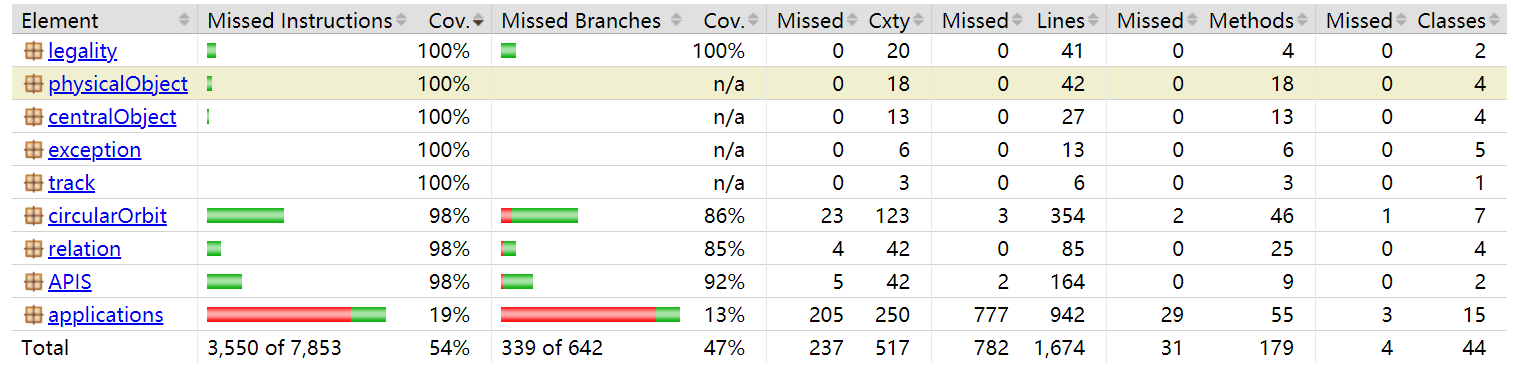
**Branch Counters:**



**Complexity:**



**EclEmma覆盖度报告（还是被applications影响了很多）：**

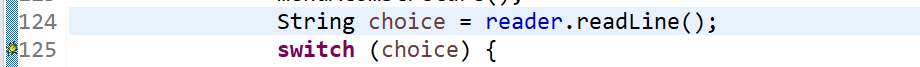


## SpotBugs tool

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。

1.调用BufferedReader.readline()时



错误：取消引用调用readLine（）的结果，而不检查结果是否为null。 如果没有更多的文本行要读取，readLine（）将返回null并且取消引用将生成空指针异常。

修改：增加判断语句**if**(choice == **null**) {

**break**;

}

2.对于Debugging的FindMedianSortedArrays，其中有个对于m+n是否为奇数的判断代码使用(m+n)％2 == 1来检查值是否为奇数，但这对负数不起作用（例如，（- 5）％2 == -1）。 如果代码打算检查奇数，我们需要使用(m+n)＆1 == 1或(m+n)％2！= 0。

## Debugging

### 理解待调试程序的代码思想

**1.FindMedianSortedArrays**

这个问题时找两个有序数组的中位数，并且要求算法的复杂度较低，使用二分查找的方法。

程序的基本思想是使用i将数组A划分成两部分，使用j将数组B划分成两部分，最后合并，leftPart是A左+B左，rightPart是A右+B右。其中有以下的关键条件：leftPart和rightPart长度相等，leftPart最大值小于等于rightPart最小值，中位数为leftPart最大值和rightPart最小值除以2。

重要的是确定i和j的关系，由程序后面m+n为奇数时返回leftPart的最大值，可确定关系式：i + j = m -i + n -j + 1 。由此式可根据i解出j。

为了保证leftPart最大值小于等于rightPart最小值，当i过大和i过小的时候需要进行调整。

最后当i大小刚好命中目标，我们需要对m+n的奇偶性进行讨论。m+n为奇数时返回leftPart最大值，m+n为偶数时，返回leftPart最大值和rightPart最小值除以2。

**2.RemovComments**

这题的程序思想不难理解。我们要做的工作是删除/\* \*/之间的block注释，和//之间的行注释。

基本的方法是按行进行处理，当读到/\*时，表示在block中；当读到\*/时，表示出block；当不在block但是读到//时，代表是行注释，需要单独处理；当字符不在block中且未读到注释符，连接到StringBuilder。最后确定已经读到\*/，不在block中，才将字符串添加到list中。

**3.TopWotedCandidateTest**

个人认为这道题理解题意比发现bug更难，尤其是嵌套列表A的使用，刚开始并没有清楚程序意图。

首先看Input部分，对于“候选人”们，persons的数组存的是对应次序投的人的编号，times数组存的是对应次序的时刻，而后面出现的六个时刻，则是穿插在times之间。Output输出的是六个时刻票数最多“候选人”的编号。

嵌套列表A存储投票，每个投票都有一个人和一个时间戳，A [count]是某人收到的第count+1次投票的列表。而A.get(i)列表则存的是所有人第i+1次被投的vote的列表。

构造器中count为某人映射的选票数，每当一个人被投一票，count图就会加的value就会加一，并且A对应的会添加那张选票。

q方法。由于A[i][0]和A[i]单调递增，因此我们可以对它们进行二元搜索，以便按时间查找最近的投票。就是两次二元搜索，找到索引i和j，确定vote，获得vote对应的person。

### 发现并定位错误的过程

**1. FindMedianSortedArrays**

对于找两个有序数组的中位数程序，第二处错误还是比较好发现的，根据程序的意思，else语句处理的是i正好命中目标且需要对m+n分奇偶进行讨论的部分，而当m+n为偶数时，

**return** (maxLeft + minRight) / 2.0; 因此返回leftPart时m+n肯定是奇数，这样我们把

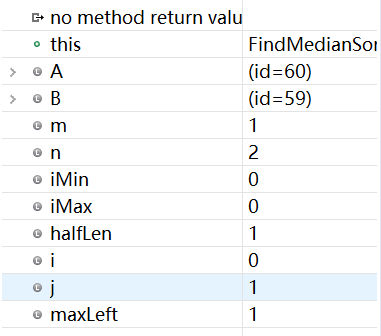
**if** ((m + n+1) % 2 == 1) {

**return** maxLeft;

}

修改为(m + n) % 2 != 0对m+n为奇数进行判断。

第一处错误刚开始难以发现，后来我通过debug调试，发现当m+n为奇数时，leftPart总是会较rightPart少一个数，并且我们还需要返回leftPart的最小值，很明显不符合题意，应当将leftPart改为比rightPart多一个数。



**2.RemovComments**

事实上，对于这题，并没有采用很多debug的工具，因为修改之处较多，因此需要理解清程序的设计思想。

对程序进行测试的时候，从test的情况，很容易就发现异常ArrayIndexOutOfBoundsException数组越界的问题。



变量inblock表示的就是读到的字符是否在/\* \*/划分的block之中，因此我们读到/\*或\*/时，需要对inblock的状态进行修改，并且通过i++跳过/\*或\*/字符。

行注释和块注释需要单独考虑，源程序没有对//的处理，显然会出错。

因为只有当我们读到\*/才表示出了block，此时才能将字符串添加到list中。

**3.TopWotedCandidateTest**

因为刚开始不知道嵌套列表A意义，因此尝试对测试文件debug，才知道A的作用。

知道A的使用，后面就比较简单了，构造器中的错误是某人映射的票数增加错误，q方法中是获取i和j索引的时候，两次二分搜索的问题。

### 如何修正错误

**1. FindMedianSortedArrays**

//修改1：halfLen = (m + n + 1) / 2

//修改原因：保证m+n为奇数的时候，中位数在左边的part

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

//修改2：(m + n) % 2 != 0

//修改原因：此时为m+n为奇数的情况，返回左边最大的

**2.RemovComments**

//修改1:i+1 < line.length()

//修改原因:防止数组越界

//修改2:i++

//修改原因:跳过/\*

//修改3: i++;

//修改原因:跳过\*/

//修改4:以下条件语句

//修改原因:增加对不在block但以//开头的注释

//修改5:!inBlock

//修改原因:确定已读到\*/ ,不在block才添加

**3.TopWotedCandidateTest**

//修改1：int c = count.getOrDefault(p, 1);

//修改原因：c为p的选票数加上1

//修改2：mi+1

//修改原因：二分法

/修改3：对lo减一

//修改原因：使得i对应list中vote的time都<=t

//修改4：<=

//修改原因：二分法

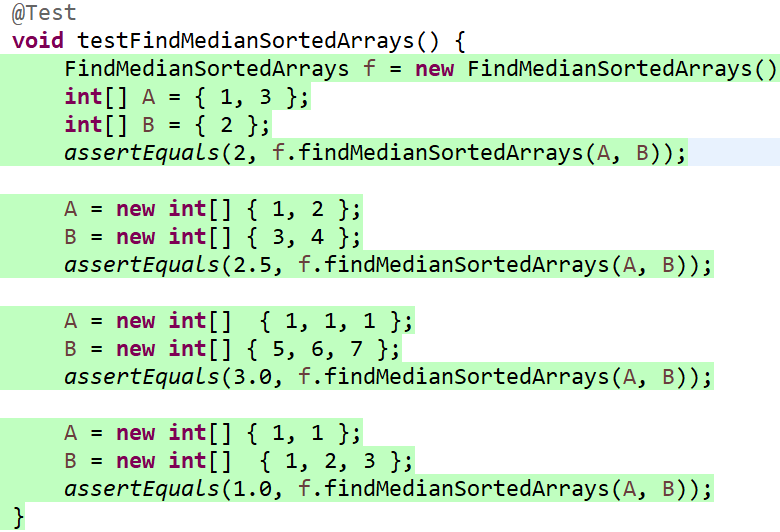
//修改5：lo-1

//修改原因：获得时间最接近且小于t的票

### 结果

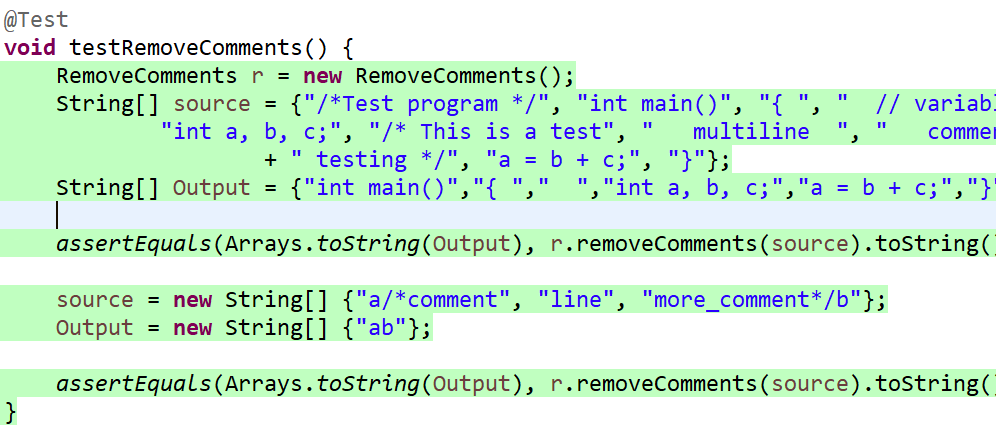
**1. FindMedianSortedArrays**

以下绿色部分代表测试通过：



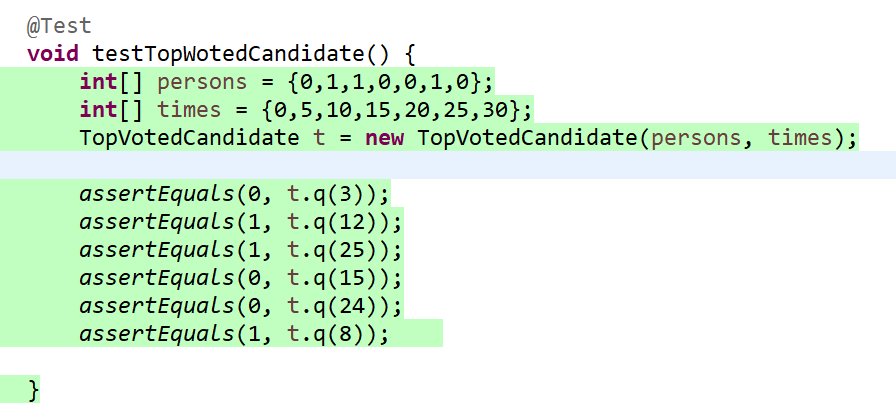
**2.RemovComments**

以下绿色部分代表测试通过：



**3.TopVotedCandidate**

以下绿色部分代表测试通过：



# 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 |
| 2019.5.4 | 14:00-18:00 | 阅读实验 | 未完按计划完成 |
| 2019.5.5 | 18:30-23:00 | 通读实验要求，配置环境 | 按计划完成 |
| 2019.5.6 | 15:30-23:00 | 自定义异常类 | 比预计晚2小时 |
| 2019.5.7 | 13:00-18:00 | 完成第一部分异常处理 | 比预计晚1小时 |
| 2019.5.8 | 19:00-23:00 | checkRep() assert | 未完成 |
| 2019.5.9 | 14:00-17:30 | 继续checkRep() assert | 按计划完成 |
| 2019.5.10 | 18:30-22:30 | 记录日志 | 按计划完成 |
| 2019.5.11 | 14:00-17:00 | 从日志中过滤信息 | 提前半小时完成 |
| 2019.5.12 | 17:30-23:00 | 写测试提高代码覆盖率 | 按计划完成 |
| 2019.5.13 | 13:30-15:30 | SpotBugs | 未完成 |
| 2019.5.14 | 18:30-23:00 | 完成前一日内容，开始debugging | 按计划完成 |
| 2019.5.15 | 10:00-17:30 | debugging找中间数 | 比预计晚2小时 |
| 2019.5.16 | 18:30-23:00 | 写完debugging | 比预计晚1小时 |
| 2019.5.18 | 18:30-22:30 | 写报告的前半部分 | 按计划完成 |
| 2019.5.19 | 14:00-17:00 | 写报告的后半部分 | 提前半小时完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 自定义异常类实现不佳，并且输出达不到预计结果 | 翻阅书籍并查询资料，反复阅读实验指导书和文件注释中的提示，冷静下来考虑解题思路 |
| assert对前置条件和后置条件判断 | 在老师课堂讲解的基础上，去MIT网站上找到相关的文献资料进行阅读。 |
| 写日志碰到很多问题，如不将日志打印到控制台，不覆盖以前的日志 | 翻阅Java编程思想，自习阅读log部分 |
| test代码覆盖率不佳 | 根据EclEmma生成的报告，设计测试用例，制造非法输入的文件和非法输入的图操作指令，对程序进行健壮性和正确性测试，不断增加测试用例 |
| debugging部分较难 | 在解题的同时翻阅各种算法类的书籍，从时间复杂度底的算法的角度取考虑问题。 |

# 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？

答：健壮性：总是假定用户为恶意用户，假定自己的代码会失败，对自己的代码要保守，对用户的行为要开放。

正确性：程序按照spec加以执行的能力，是最重要的质量指标！

有这样的原则：对外接口，倾向于健壮，对内实现，倾向于正确。

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？）

答：如果是满足程序的正确性，这是划算的。如果满足为了健壮性，此时需要权衡考虑，代价过大则不适宜。

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？

答：本质上来讲正确性倾向于直接报错，健壮性倾向于容错，避免给用户太大的压力，帮助用户承担一些麻烦（二者并不冲突）。

因此二者考虑的立场不同，对程序良好运行的出发点不同。如果对某个正确性的处理需要过多的代价，我会选择让用户承担部分确保正确性的职责，但是多数情况我会通过代码的完善，让自己的程序能应对更多的异常情况。

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？

答：excessively defensive当然是没必要的，因为我们会因为少量的健壮性而损失程序大量的性能，因此我们需要做出权衡，对必要的防御，我们不能懈怠，而对于极少可能出现的异常情况，我们需要估计它造成的影响，从而针对性防御。

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？

答:debug设置断点调试，写test文件定位错误。

会更有效，assert断言显示错误的位置和信息，log记录异常日志，都是良好的定位错误的方式。

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？

答：对于充分的测试用例，我们需要人为制造非法输入的文件和非法输入的图操作指令，对程序进行健壮性和正确性测试，想方设法让程序崩溃。

代码覆盖率100%只是能体现出程序测试对于代码覆盖的完整性，未必能让我们发现程序的缺陷和不足。

1. Debug一个错误的程序，有乐趣吗？

答：debug过程艰辛，结果有趣。

1. 关于本实验的工作量、难度、deadline。

答：工作量看似不大，做起来也不轻松，难度不大，deadline较合适。

1. 到目前为止你对《软件构造》课程的评价和建议。

答：这是我非常喜欢的一门课，课程教会了我很多的知识，带我走进了软件构造的新世界。但是由于课程进度快，且ppt都是英文，因此课程难度较大，整个课程的设计应该还有可以完善的地方。

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。

答：因为课程ppt内容过多，建议多考核重点部分，不过多在意细枝末节。嗯…还是希望能尽快上完课，给我们划一下重点。