

**2020年春季学期  
计算机学院《软件构造》课程**

**Lab 4实验报告**

|  |  |
| --- | --- |
| 姓名 | 张高玮 |
| 学号 | 1180300407 |
| 班号 | 1803004 |
| 电子邮件 | 1462034631@qq.com |
| 手机号码 | 15630559577 |

**目录**

[1 实验目标概述 1](#_Toc39518185)

[2 实验环境配置 1](#_Toc39518186)

[3 实验过程 1](#_Toc39518187)

[3.1 Error and Exception Handling 2](#_Toc39518188)

[3.1.1 处理输入文本中的三类错误 2](#_Toc39518189)

[3.1.2 处理客户端操作时产生的异常 2](#_Toc39518190)

[3.2 Assertion and Defensive Programming 2](#_Toc39518191)

[3.2.1 checkRep()检查rep invariants 2](#_Toc39518192)

[3.2.2 Assertion/异常机制来保障pre-/post-condition 2](#_Toc39518193)

[3.2.3 你的代码的防御式策略概述 2](#_Toc39518194)

[3.3 Logging 2](#_Toc39518195)

[3.3.1 异常处理的日志功能 2](#_Toc39518196)

[3.3.2 应用层操作的日志功能 2](#_Toc39518197)

[3.3.3 日志查询功能 2](#_Toc39518198)

[3.4 Testing for Robustness and Correctness 2](#_Toc39518199)

[3.4.1 Testing strategy 2](#_Toc39518200)

[3.4.2 测试用例设计 2](#_Toc39518201)

[3.4.3 测试运行结果与EclEmma覆盖度报告 2](#_Toc39518202)

[3.5 SpotBugs tool 2](#_Toc39518203)

[3.6 Debugging 3](#_Toc39518204)

[3.6.1 EventManager程序 3](#_Toc39518205)

[3.6.2 LowestPrice程序 3](#_Toc39518206)

[3.6.3 FlightClient/Flight/Plane程序 3](#_Toc39518207)

[4 实验进度记录 3](#_Toc39518208)

[5 实验过程中遇到的困难与解决途径 4](#_Toc39518209)

[6 实验过程中收获的经验、教训、感想 4](#_Toc39518210)

[6.1 实验过程中收获的经验和教训 4](#_Toc39518211)

[6.2 针对以下方面的感受 4](#_Toc39518212)

# 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

①安装JDK8，去官网下载，选择下载目录

②配置环境变量：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，值为JDK安装目录

Path变量→编辑：输入%JAVA\_HOME%\bin

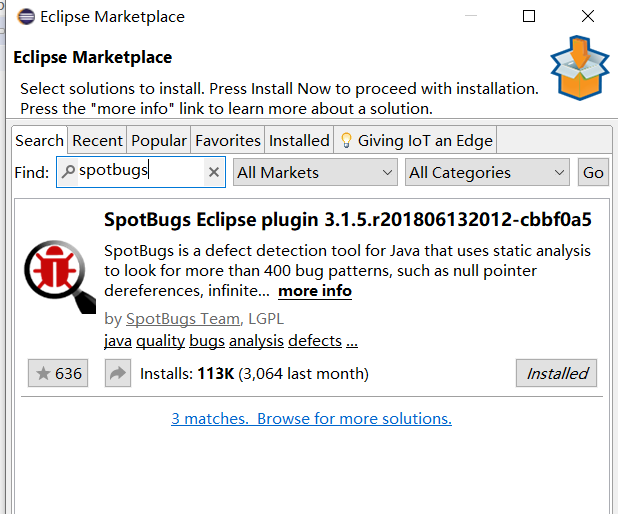
③官网下载、安装eclipse、Git

④根据老师实验课指导了解Junit创建、使用testcase的方法

⑤注册Github账号，根据给出的URL地址绑定学号，创建仓库

⑥在 Eclipse IDE 中安装配置 SpotBugs：

在eclipse中的eclipse marketplace搜索、安装SpotBugs插件



在这里给出你的GitHub Lab4仓库的URL地址（Lab4-学号）。

https://github.com/ComputerScienceHIT/Lab4-1180300407.git

# 实验过程

## Error and Exception Handling

为特殊情况定义异常，用异常名就可相对区分所发生的特殊情况，并在提示信息中进行标识,如语法错误：

异常名为



提示信息有标识:



共定义了24种异常



### 处理输入文本中的三类错误

为每一类错误都定义对应的异常类型以及各自的子异常:

①不符合语法规则:UnGrammaticalWordException

10种子异常:

机龄大小超出范围：AgeOutofBoundException

飞机座位数超出范围:SeatsSizeOutofBoundException

航班日期存在格式错误：DateFormatException

航班名称存在格式错误：FlightNameFormatException

机场名称出现非法字符：IllegalCharacterForAirportNameException

航班信息不完整(比如有Flight部分信息但缺失plane信息):

IncompleteFlightInformationException

机龄、座位数等出现非数字字符:NonNumberException

飞机的编号存在格式错误:PlaneIDFormatException

机型中出现非法字符:TypeContainsOtherSymbolException

②输入文件中存在标签完全一样的元素:SameLabelException

③输入文件中各元素之间的依赖关系不正确：

IncorrectElementDependencyException

4种子异常：

两个飞机ID相同，其他地方有不同：IllegalPlaneContentException

起飞日期和降落日期差距大于1天：DateDifferMuchException

航班号相同的航班出发或到达时间/机场不一致：

InconsistentStartOrEndException

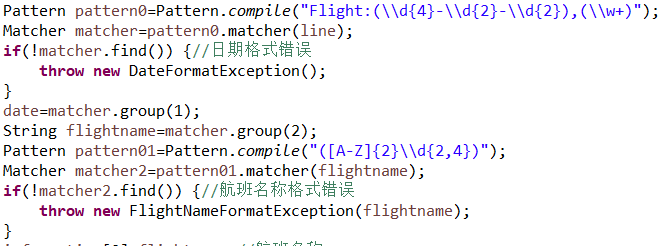
航班日期与内部起飞时间日期不一致：InconsistentDateException

正如3.1节总述部分所讲，每种异常以异常名和获得现场信息方法中的固定标识

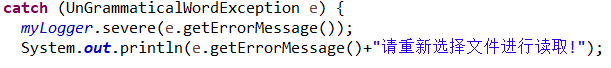
如 来加以区分并进行相应的提示。

这些异常情况大都出现在读入文本解析或创建航班时，在这些函数里，我们判断是否出现异常情况，如果出现异常情况，就throw，让client做更好地处理：

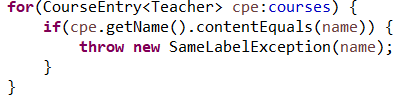
例如:



读入Flight行时可能遇到两种异常:日期格式错误和航班名称格式错误，判断发生后进行抛出，然后在app端捕获并提示相关信息,提示重新读取，这样完成异常处理



此外，这里的部分异常不仅是只适用于航班创建，同样适用于其他计划项，如创建时存在标签完全一样的元素:SameLabelException，因此在FlightSchedule和CourseSchedule中也加入这些异常处理，如课程创建不允许重名：



### 处理客户端操作时产生的异常

共考虑了8种异常情况:

位置占用冲突：LocationConflictException

资源分配冲突:ResourceConflictException

进行操作(如删除、分配)的位置还未创建:LocationNotFoundException

进行操作(如删除、分配)的资源还未创建:ResourceNotFoundException

进行操作(如分配、启动)的计划项还未创建:PlanEntryNotCreateException

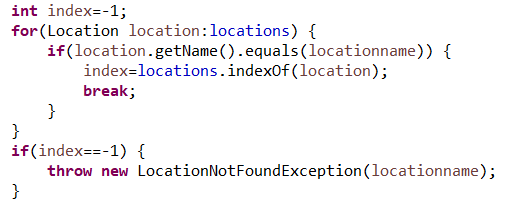
进行操作(如删除)的位置被占用:PlanEntryOccupyLocationException

进行操作(如删除、分配)的资源被占用:PlanEntryOccupyResourceException

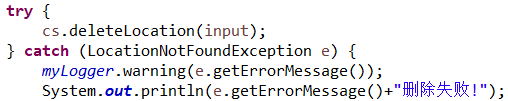
进行状态转换的计划项状态不匹配:PlanEntryStateNotMatchException

与3.1.1节异常处理手段类似，以异常名和获得现场信息方法中的固定标识来加以区分并进行相应的提示，判断出异常情况后进行抛出，上一级进行捕获处理、打印信息进行提示。

如:CourseSchedule中删除教室时判断出教室尚未创建:



在app中进行捕获处理、打印信息：



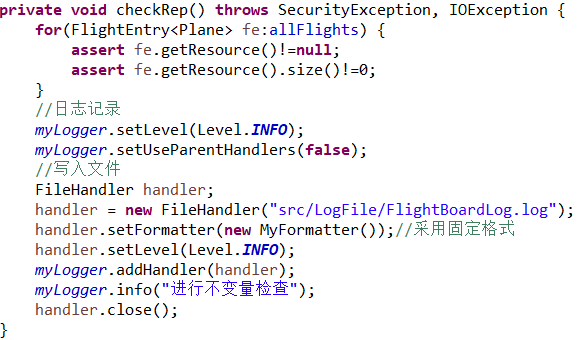
## Assertion and Defensive Programming

### checkRep()检查rep invariants

①Board类：待展示的所有计划项都应该已经分配资源

例如机场信息板:

checkRep：



循环部分即是使用断言来保证不变量，后续部分是3.3节的日志记录

②每个维度的计划项，如位置、时间：

这里以多位置维度为例，多位置就要求位置数>=2,并且各个位置互不相同:

**assert** locations.size()>=2;

Set<Location> locationset=**new** HashSet<Location>(locations);

**assert** locationset.size()==locations.size();//互不相同

因此用断言机制予以保证，其他维度与之类似,采用断言保证不变量

如多个可区分资源:资源不能重复:



③对计划项、位置、资源进行统一维护的schedule:

不变量要求管理的计划项、位置、资源都不应含有重复元素，因此同样用断言保证，如FlightSchedule中:

Set<Location> locationSet=**new** HashSet<Location>(locations);

Set<Plane> planeSet=**new** HashSet<Plane>(planes);

Set<FlightEntry<Plane>> flightEntriesSet=**new** HashSet<>(flights);

**assert** locationSet.size()==locations.size();

**assert** planeSet.size()==planes.size();

**assert** flightEntriesSet.size()==flights.size();

而TrainSchedule中因为为高铁站点额外维护了一个站点名称的String类型的list，

因此不变量应要求实际位置的list与其顺序一致，用断言加以保证:

**assert** locationnames.size()==locations.size();

**for**(**int** i=0;i<locationnames.size();i++) {

**assert** locationnames.get(i).equals(locations.get(i).getName());

}

④时间对Timeslot：

不变量：形式为 yyyy-MM-dd HH:mm，且结束时间应晚于开始时间

构造函数中用正则表达式保证形式，且没有提供变值器从而保证形式：

String pathString="\\d{4}-\\d{2}-\\d{2} \\d{2}:\\d{2}";

Pattern pattern=Pattern.*compile*(pathString);

Matcher matcher1=pattern.matcher(starttime);

Matcher matcher2=pattern.matcher(endtime);

在checkRep中转为Date形式进行比较来保证时间上先后顺序:

String pattern="yyyy-MM-dd HH:mm";

SimpleDateFormat dateFormat=**new** SimpleDateFormat(pattern);

**try** {

Date date1=dateFormat.parse(starttime);

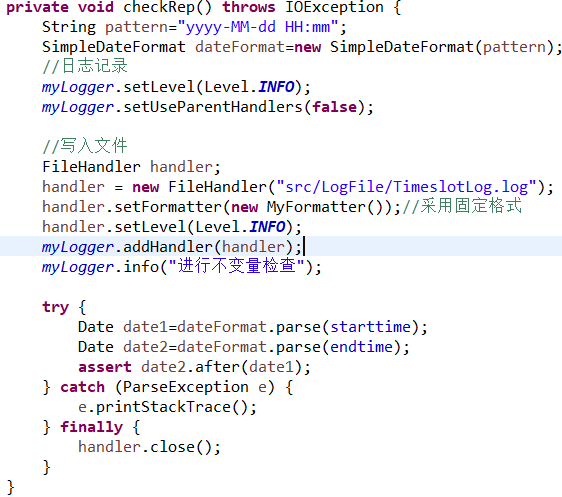
Date date2=dateFormat.parse(endtime);

**assert** date2.after(date1);

} **catch** (ParseException e) {

e.printStackTrace();

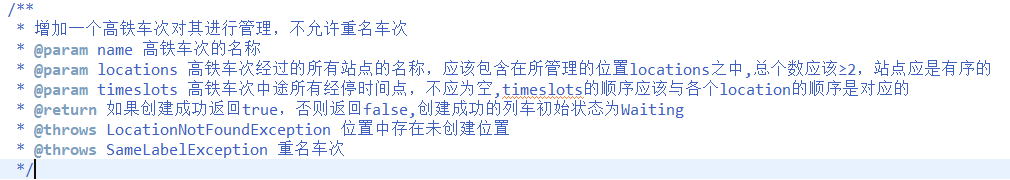
}



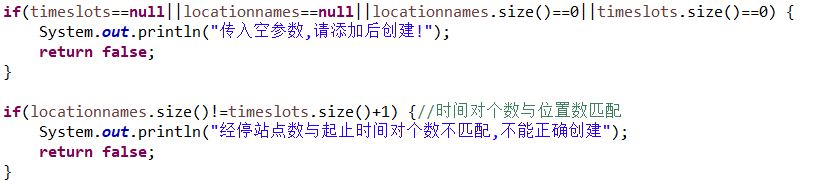
其余部分为3.3节日志记录

### Assertion/异常机制来保障pre-/post-condition

如创建高铁车次方法:

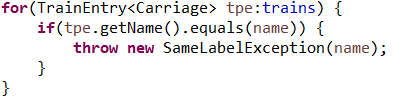


前置条件不允许重名，站点数与时间数匹配，首先对参数进行前置条件检查:



这里没有引入异常，直接提示调用者

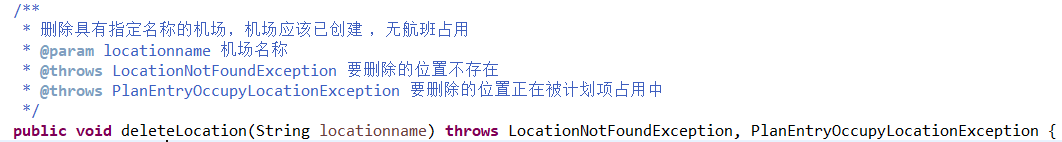
对于重名的前置条件检查:抛出异常



对于Waiting状态的后置条件检查:使用断言



再如删除机场的方法:

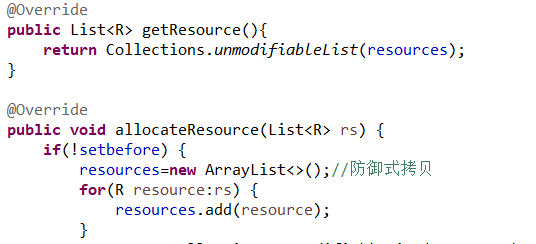


前置条件要求机场已创建、未被占用，在代码中进行检查:



### 你的代码的防御式策略概述

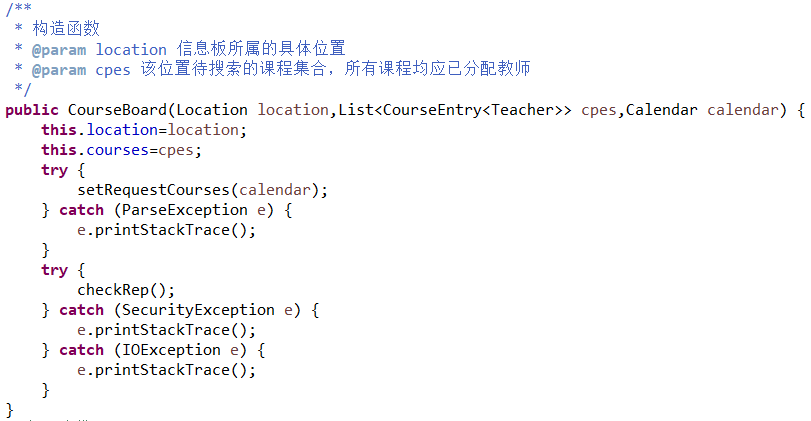
①防御式拷贝，对于可变类的对象属性，观察器和变值器都是生成一个副本，如:



②利用断言进行不变量检查，如：



构造器、变值器中对不变量检查,如:

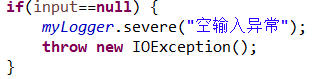


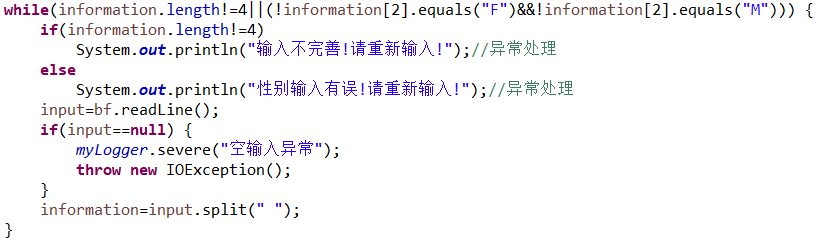
③用断言/异常保证前置/后置条件:

在3.2.2节叙述

④App中对各种非法输入进行防御

如:





## Logging

使用Java 的库 java.util.logging增加日志功能，将日志写入文件中。

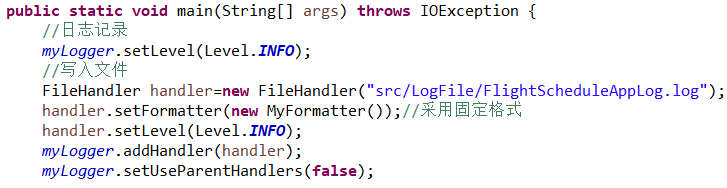
### 异常处理的日志功能

①异常：之前提到在每个Schedule中都将异常抛出，在app中捕获，因此对app进行日志记录：当捕获异常时进行日志记录

每个app中维护一个Logger变量，名称为类名+Log，如

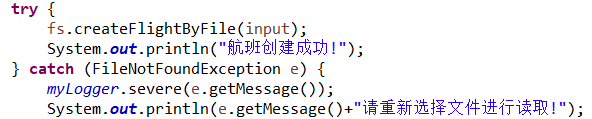
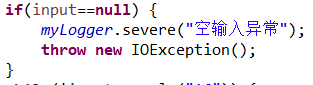
**private** **static** Logger *myLogger*=Logger.*getLogger*("FlightScheduleAppLog");

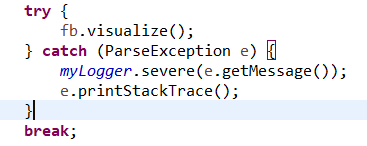
Main函数中为其设置级别以及handler，这里采用文件记录



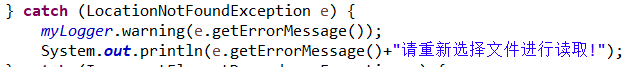
每当捕获一个异常时进行日志记录，这里记录的格式由MyFormatter指定，其格式会在3.3.3节日志查询中介绍。

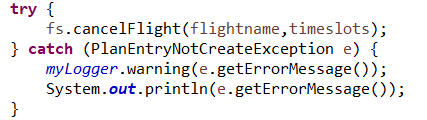
对于异常记录的级别：如果是危害很大，应该终止的异常，记录时设置为SEVERE级别，比如:空输入，文件不存在，时间转化时出错等等：

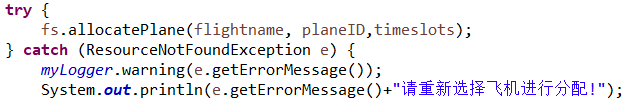




如果是用户对于计划项、位置等选择型错误，如删除的还未创建，分配的资源导致冲突等等，设置为WARNING级别，如：







②应用内部断言处理的日志记录

应用内部同样维护一个Logger变量，由于checkRep中调用断言进行处理，因此在checkRep中同时进行日志记录，如:

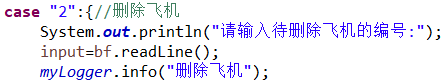


值得一提的是在测试过程中对于handler.close的使用，一开始没有handler.close时会导致各个类的对象在互相调用时都会执行checkRep从而使用日志文件，但只要有一个先行占用，就会导致其他类的对象在调用时出现锁定错误："java.io.IOException: Couldn't get lock for xxx”，经过查找资料，添加handler.close就可避免这一异常情况。(myLogger同理)

### 应用层操作的日志功能

App中使用3.3.1中的myLogger，用户每选择一个功能就进行日志记录，即记录用户操作，记录的格式可统一为3.3.1中提到的myFormatter，在3.3.3中介绍

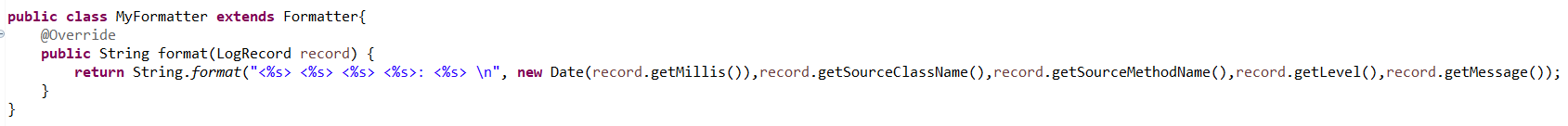
记录时设置日志级别为INFO，以与异常相区分，是正常的信息记录,如用户选择删除飞机:



信息级别为INFO，并记录操作信息:“删除飞机”

### 日志查询功能

为了便于查询，在日志记录时将异常和正常操作类型记录的格式统一起来,定义在MyFormatter类中，继承Formatter类:



其代表的信息为<时间> <类名称> <方法名称> <日志级别>: <具体信息> \n

前四项自然可以统一起来，至于最后一项，如果是异常，保存的就是异常的提示信息，如果是正常操作，就可以由我们自己定义传进去的信息，我这里就是保存正常操作的类型，比如

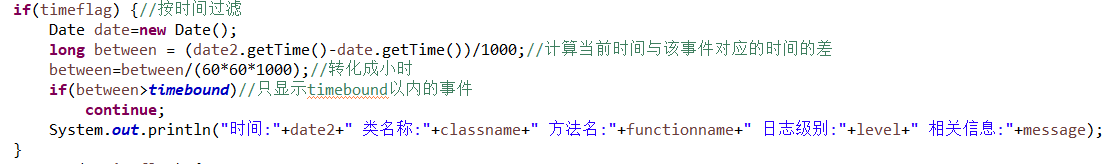
格式统一后查询就比较容易，我这里提供三种查询方式:

<1>按时间查询:



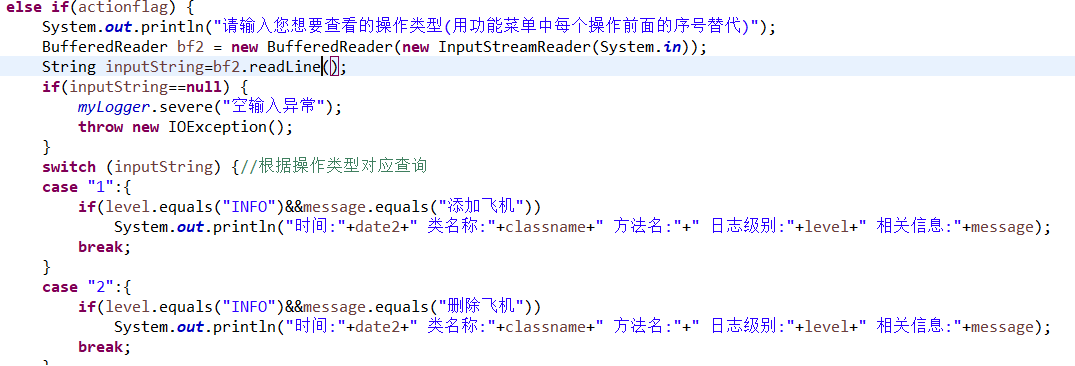
app中维护一个时间界限(小时)，设time表示当前时间，则用户按时间查询时就是查询time-timebound至当前时间以内的所有日志

这时我们只需要遍历日志记录，根据<时间>进行过滤，满足条件的进行显示即可



<2>按操作类型查询

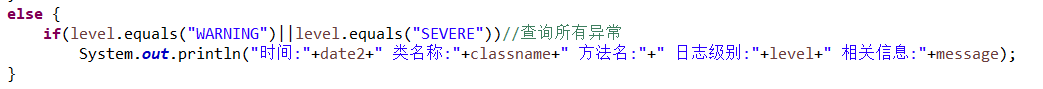
用户指定要查询的操作，即输入功能菜单上对应的编号，然后进行遍历查询:



用switch-case处理，根据日志级别为INFO以及记录的信息进行过滤

<3>查询所有异常:

根据日志级别过滤、显示即可



## Testing for Robustness and Correctness

需要注意的是，因为有些测试为了解开set函数与get函数之间的耦合，需要类的部分属性改为public进行测试，因此在测试完成后我已经将其注释掉，需要测试时修改属性即可

### Testing strategy

使用等价类和边界值的测试思想，为各 ADT 添加 testing strategy

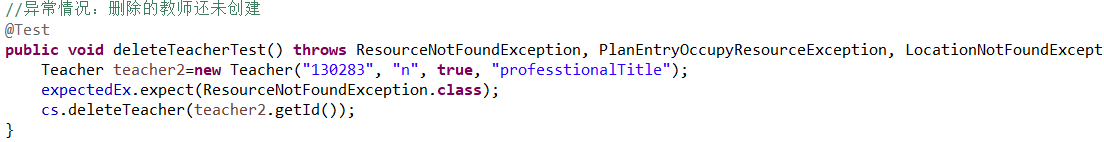
因为每个ADT的不同方法等价类和边界值可能有较大差别，因此我将测试策略细化到每个方法中，由于测试繁多，这里仅以一个例子说明：

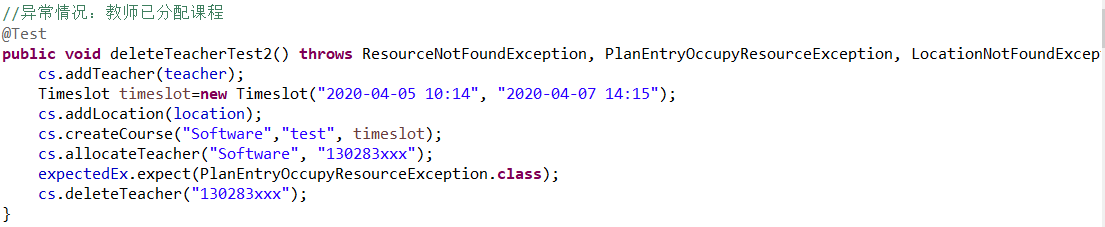
例如删除教师deleteTeacher方法：

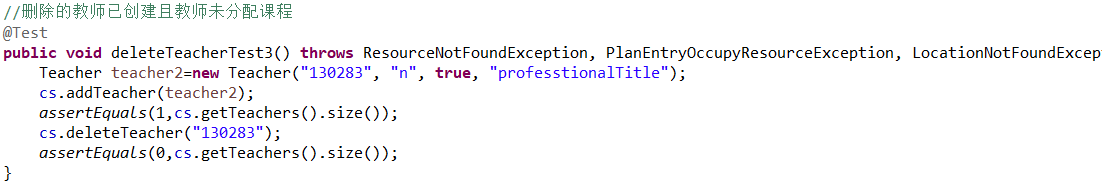
测试策略:



然后对每个维度设置测试用例:







在测试异常抛出时利用@Rule声明错误期望对象,

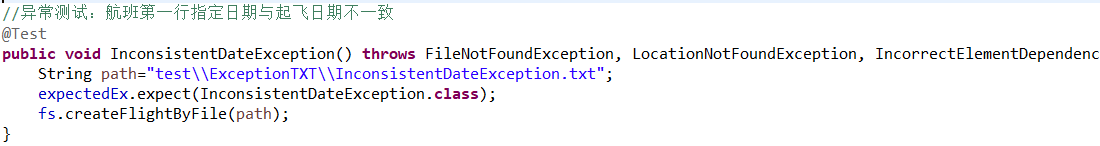


在执行预计产生异常的语句前进行期望(expect):

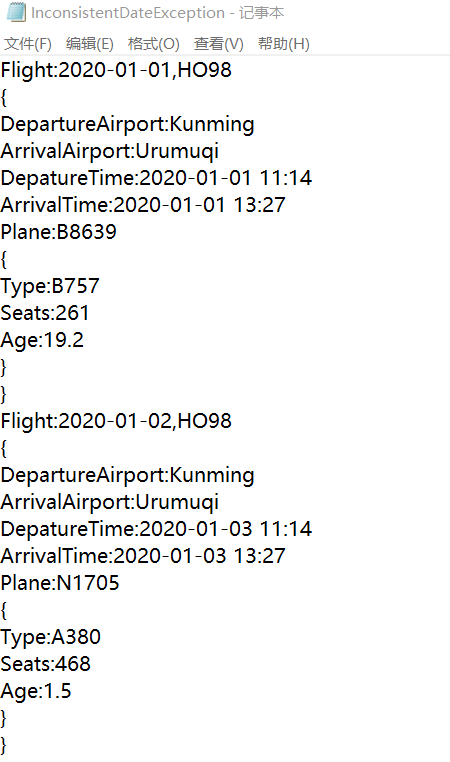
expectedEx.expect(ResourceNotFoundException.**class**);

### 测试用例设计

对于输入文本带来的异常，在FlightSchedule原有测试的基础上增加对于自定义异常情况的测试，自行构造文本文件,不失一般性，所有测试文件中至少包含两个航班，如航班第一行指定日期与起飞日期不一致:

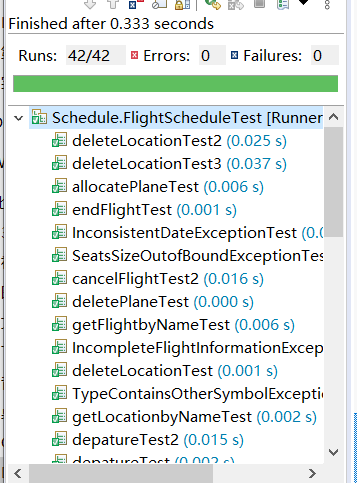


构造文本使得航班第一行日期与内部指定的起飞日期不一致:



其中第二个航班是异常的。

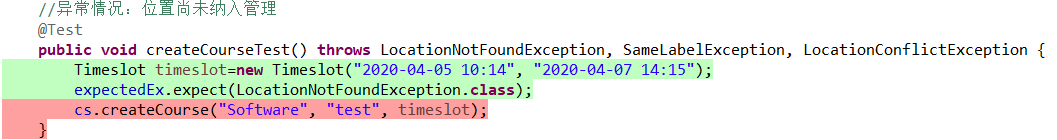
所有的测试结果：



其他与输入文件无关的异常已经在每个方法等价类划分时予以考虑并测量，如3.4.1中介绍的deleteTeacher中等价类划分时就考虑了异常.

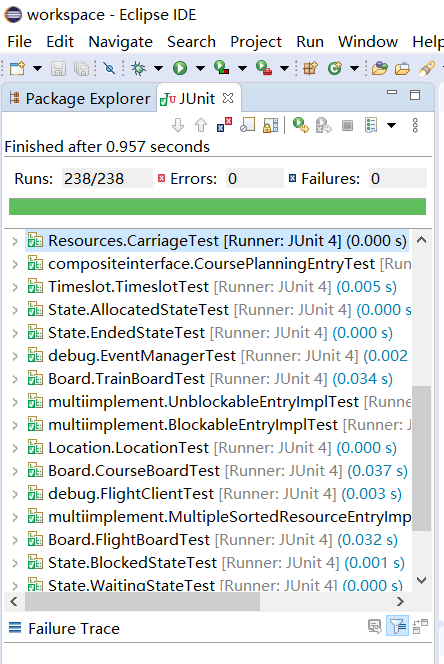
按照每个方法的测试策略进行测试用例设计，这里不再赘述。

而test部分没达到100%是因为用Rule期望的异常捕获以后，执行语句算作未执行,如:

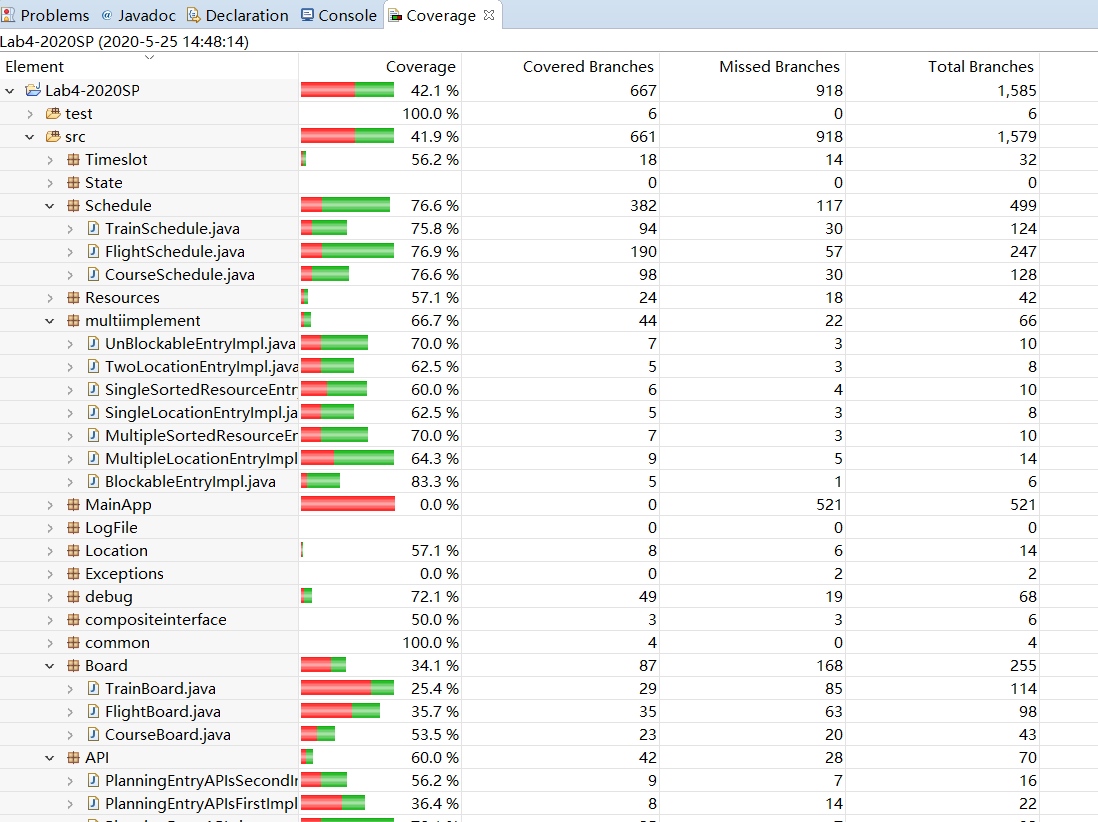


### 测试运行结果与EclEmma覆盖度报告

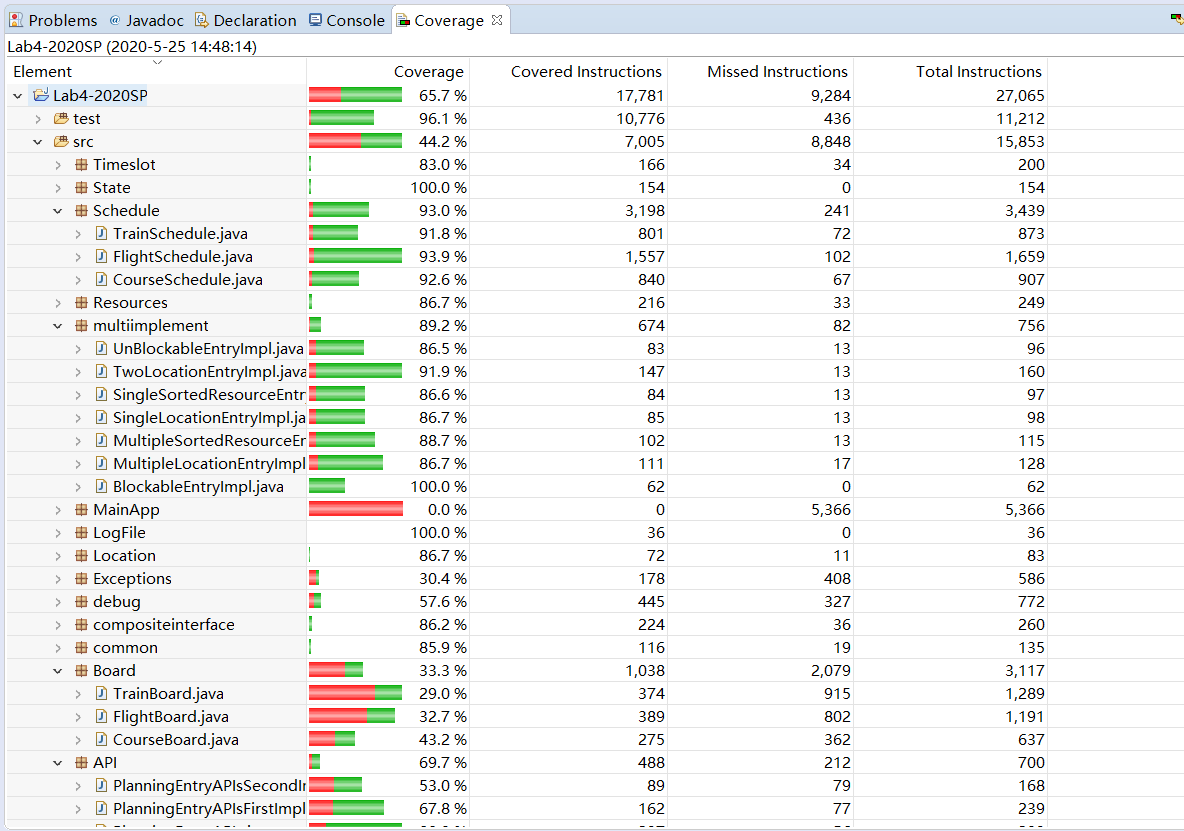
测试结果:



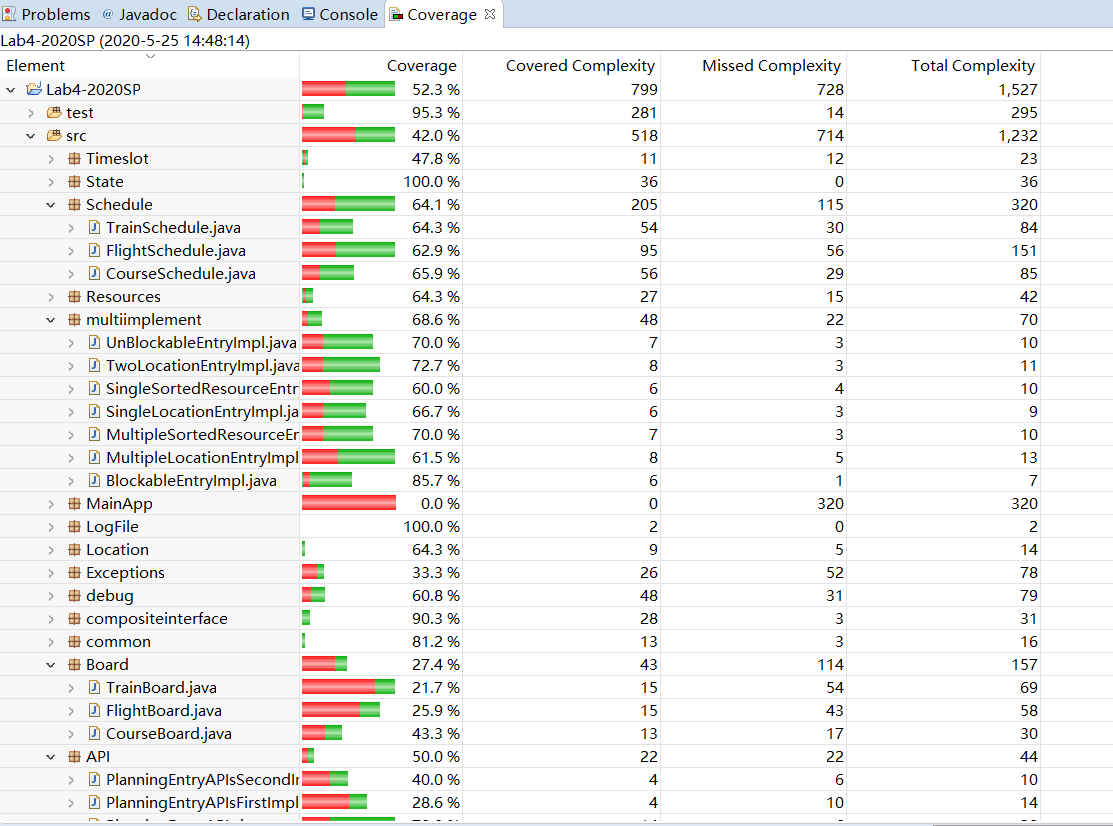
分支覆盖度:



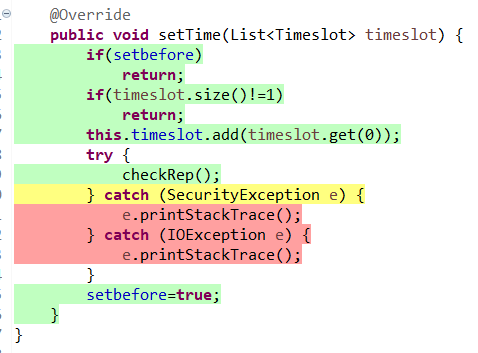
语句覆盖度:



路径覆盖度:

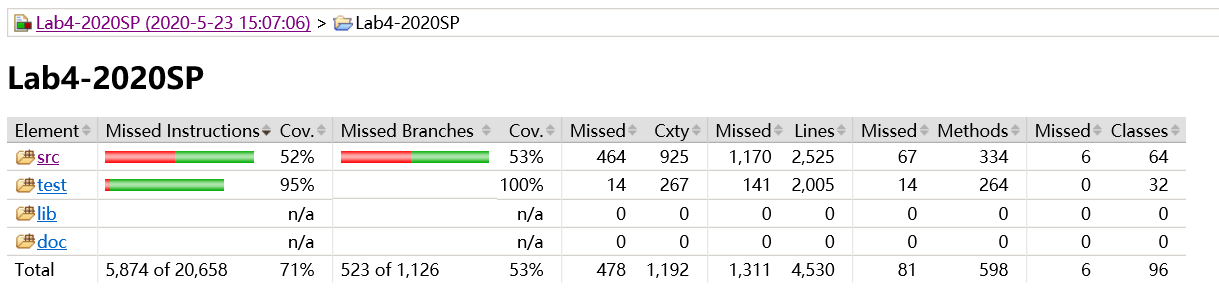


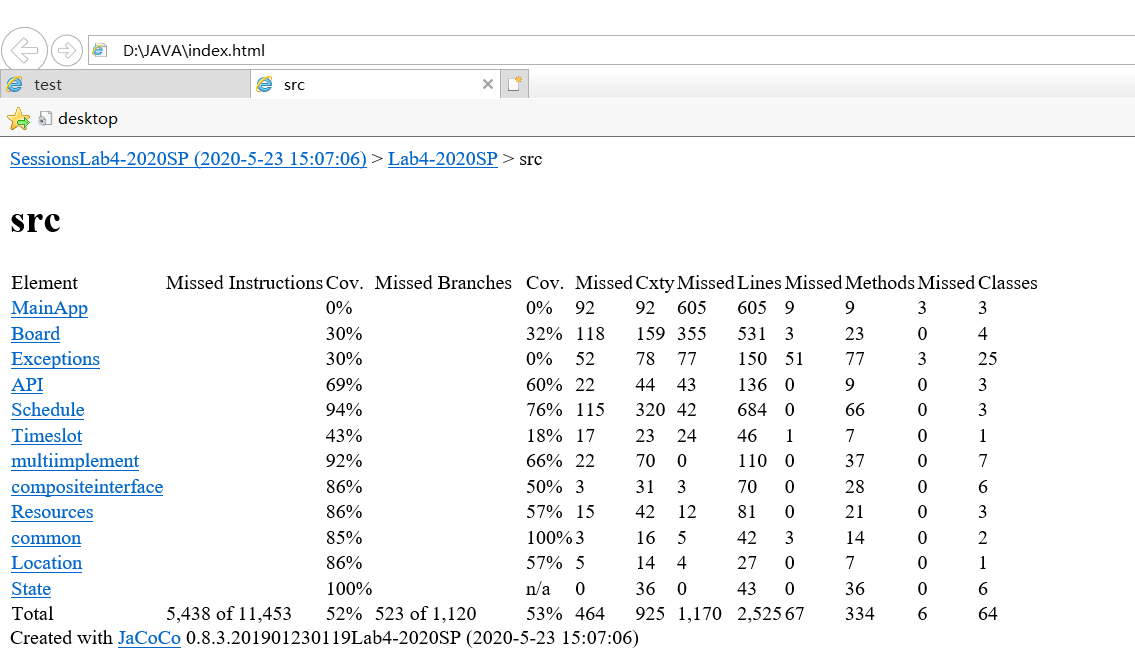
这里覆盖度较低是因为client程序MainApp占比很大，Board中可视化部分占比大，这些无法测试进行覆盖，而像multiimplement每个维度的计划项中miss的部分为checkRep时对断言失败的异常处理，如:

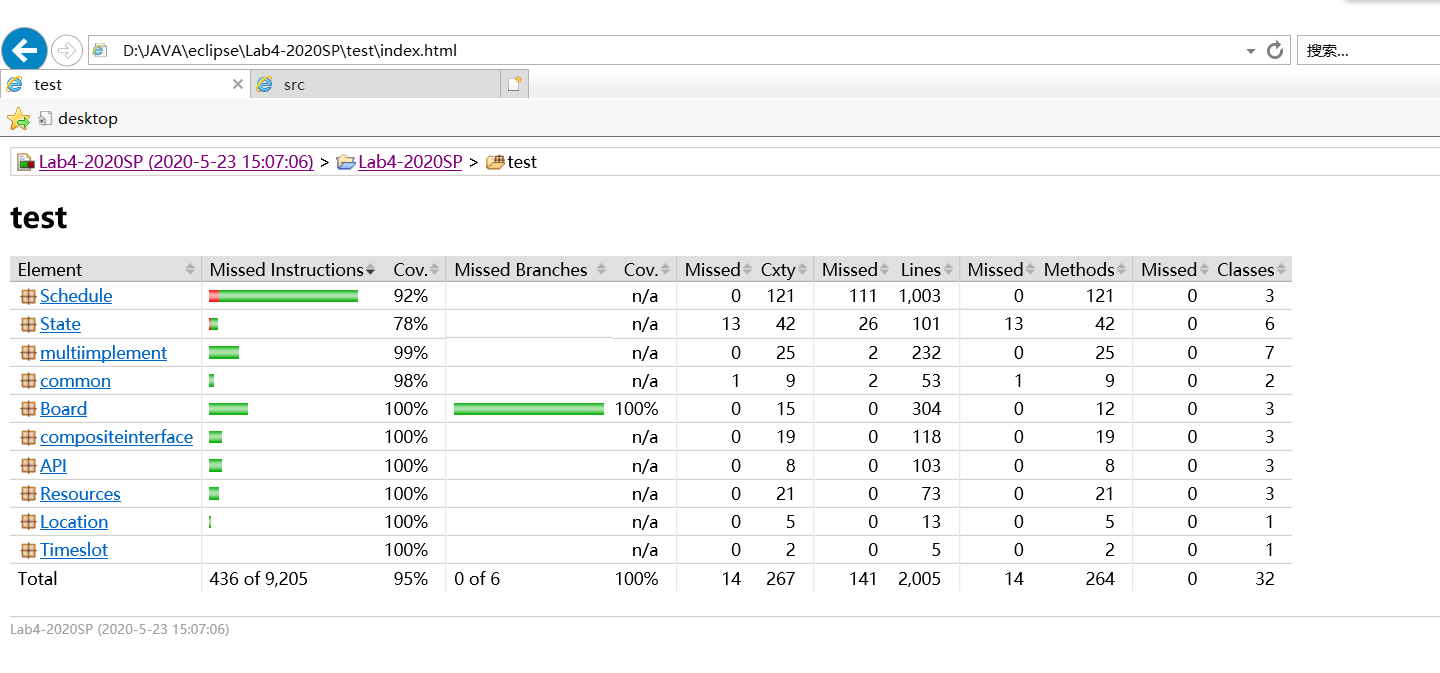


这部分确实不应执行，因为这是不变量不满足时执行的代码，而不变量应该时刻满足。

测试覆盖度报告:

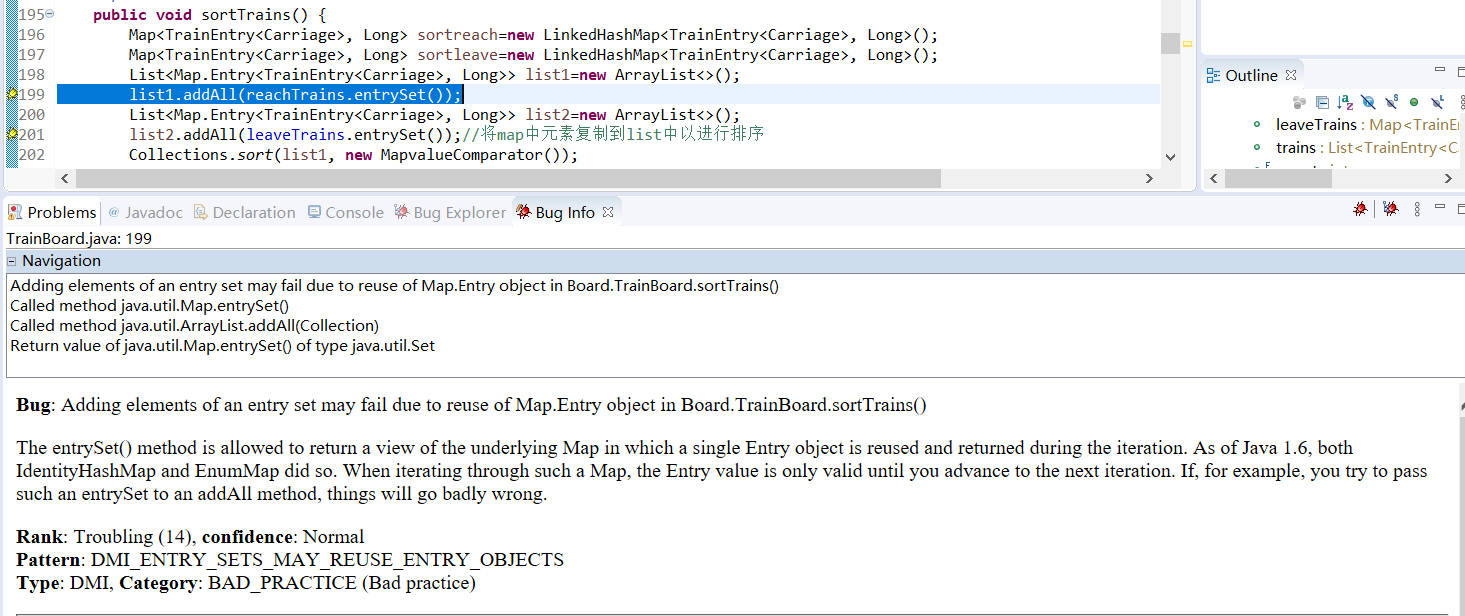






## SpotBugs tool

Bug 1：对于map排序实现了一个comparator，然后对于entryset使用list自带的addall方法加入list，使用Collections+实现的comparator进行排序

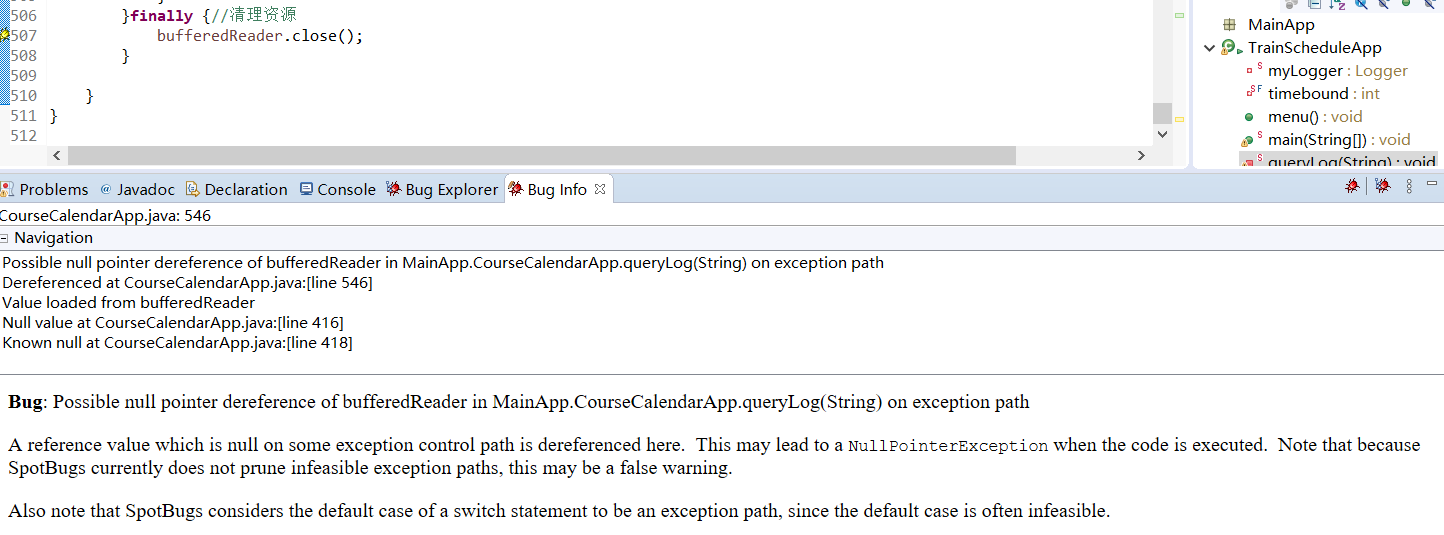


而对于entry以及entryset应该避免add以及addall方法的使用，其存在隐患。

修改:排序时使用treemap

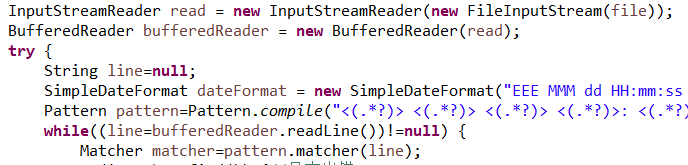


Bug 2：清理资源时，资源在某路径上可能为null



原因：在try之前声明buffReader=null，而如果try中语句不执行，那么此时buffReader=null未改变，导致错误。

修改:在try之前即赋值



## Debugging

### EventManager程序

①理解待调试程序的代码思想

按照原程序思想，我的理解是调用一次book函数，需要将一整年中的最大交集区间数返回，而不同天之间必然不会存在交集，因此这个最大交集区间数应该是某一天的最大交集区间数，而这一天并不一定是第day天。

而对于具体实现的思想，我的理解是将区间的每一个整数点进行标记（用Map），查询是累加Map.values()查找最大值，也就是说value增加，意味着事件增多，而value增加即代表对应的value为正值，因此将start对应的value设正，end对应的设负，这样就可以通过累加求最大值保证最大交集数。但在调试的过程中发现，仅仅这样还不够，因为我们无法区分不同天之间的事件，因此这时就需要一个额外的小处理，在下面会介绍到。

②发现并定位错误+修正错误的过程

<1>首先书写测试用例：

//测试策略:1.新增1个事件；新增n个事件

// 2.n个事件位于不同天；n个事件有位于同一天内的

// 3.n个事件互不相交；n个事件有交集

测试时报错:nullpointer

发现程序中未对事件的初始value赋值

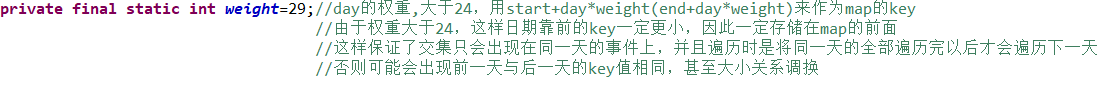
<2>然后先简单加上一个判断：

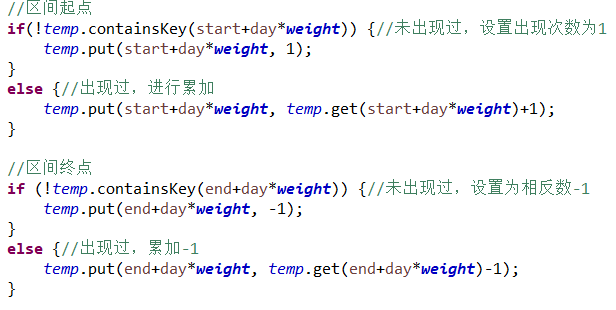
，为其简单赋初值1（end置-1）

再进行测试，在类中创建一个main函数，将测试不通过的移植到main中，进行debug调试

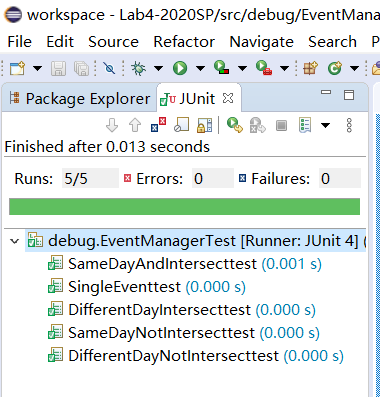
<3>经过debugger理解思想后，确定map的value大概表示这一时间点出现的次数，修改后再测试，发现不同天的事件无法区分，且参数day未使用

<4>为day设置权重，也加入到key中





③修复之后的测试结果



### LowestPrice程序

①理解待调试程序的代码思想

逐个offer进行试探，每次将一个special offer加入“购物车”，更新需求，递归求解。对于offer1,offer2,……对于每个offer，都作为选取顺序中的第一个，递归时又作为第二/三/……个，进行重复试探再次进行验证，则遍历了所有offer的组合，维护整个过程中的最小结果，保证了是这些试探结果中的最小的组合

②发现并定位错误+修正错误的过程

<1>首先书写测试用例:

//测试策略:1.单个商品；6个商品；n个商品(1<n<6)

// 2.没有special offer；单个special offer;n个special offer(1<n<=100)

// 3.special offer中商品数超过需求的数目；special offer中商品数不超过需求的数目

// 4.special offer中只选择一次；special offer选择多次；

<2>分析代码思想，可以得知dot方法是用来求不用offer时商品的总价

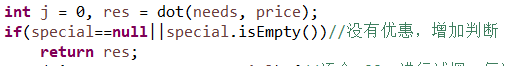
很容易发现

会导致循环无论如何都会完成，则后面的判断

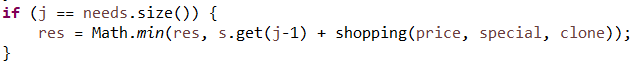
没有意义，因此continue改为break

<3>用测试用例测试，出现null-pointer错误，发现代码未对special参数进行检查

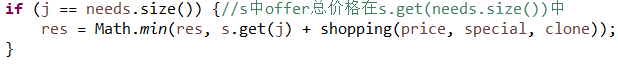
可能存在没有优惠的情况，因此添加判断条件



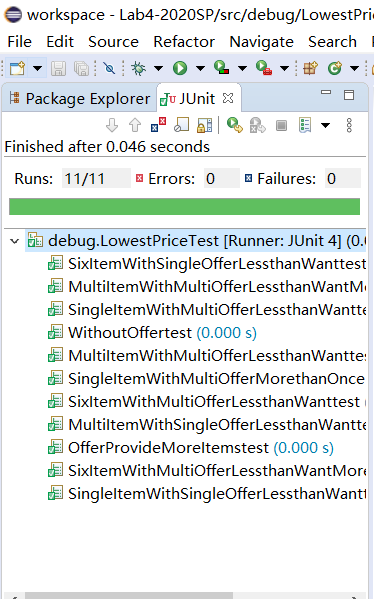
<4>再测试，发现结果不对，用debugger进行调试，发现计算优惠结果与期望不符，总是special中的最后一个商品的数量:因此定位在计算special价格部分



发现我们期待的价格是在s.get(needs.size())中而s.get(j-1)取出的是前一个数，即最后一个商品的数量，因此修改为j



③修复之后的测试结果



### FlightClient/Flight/Plane程序

①理解待调试程序的代码思想

用随机数随机取一个飞机，和其他的航班进行比较，如果该飞机没有被分配到其他任何航班，可以分配。如果被分配过，看二者是否会产生冲突，如果有冲突，再取另外的飞机进行重复试探，当所有飞机都试探过也没有分配成功时则分配失败

②发现并定位错误+修正错误的过程

<1>书写测试用例:

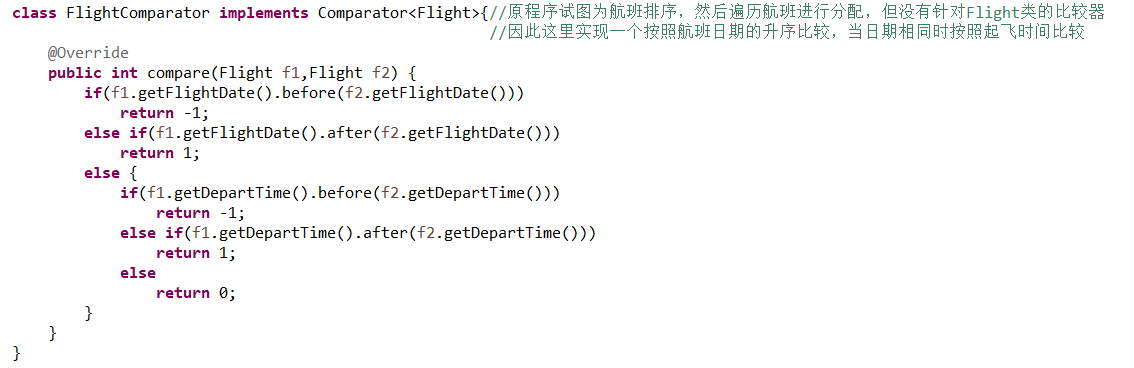
//测试策略:1.飞机数大于航班数；飞机数小于航班数

// 2.航班间存在日期/时间冲突；航班间不存在日期/航班冲突

<2>代码中有ide自己的报错提示:

1)

我理解这段代码是想要先将航班由早到晚排序然后从早到晚进行分配，但默认的比较器Comparator无法比较flight类，因此我们先实现一个Comparator，按照起飞日期(日期相同时按照起飞时间)由早到晚的顺序排序:



然后我们在排序时指定使用这个Comparator：

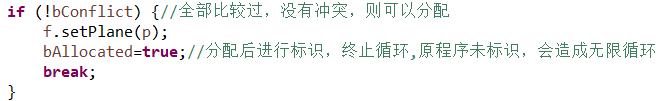


1. Calendar类对象的比较直接使用了>,<号，日期比较应使用Calendar类中的after，before方法:



<3>进行测试发现程序进入死循环,进行debug，发现控制循环条件的变量没有代码会对其进行改变:

从而导致死循环，因此分配成功时应将bAllocated设置为true以终止循环



<4>再次测试，发现分配失败的测试仍然出现死循环

再次用debugger调试，如果有冲突导致没有方案可以分配时也没有终止循环的出口

分析代码，用随机数进行分配，判断失败的条件应该是所有飞机都尝试分配，但分配失败，这时应该退出循环了。

因此我们加上一个判断，用nextInt产生的随机数是在0到planes.size()-1之间的，全部分配过时应该是指这些随机数全都生成过，因此我们用set保存已经生成的随机数，

当0到planes.size()-1之间的所有数都尝试过时，set的大小就为planes.size()，此时分配失败，退出循环:

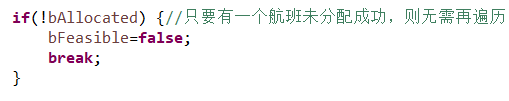


<5>进行测试，发现所有的结果都是true。

再次debugger调试，发现程序的返回变量是

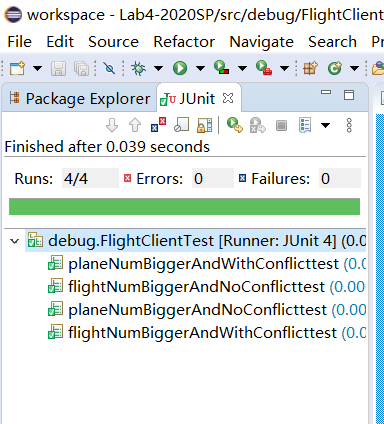
而这个变量只在开始赋值为true

后面没有任何变值操作，因此需要在分配失败时进行变值并直接返回



添加判断，只要有一个航班分配失败就是分配失败

③修复之后的测试结果



# 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。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2020/5/20 | 19:00-20:48 | 完成3.1节异常 | 完成了输入文本部分的异常定义与处理 |
| 2020/5/21 | 13:00-15:05 | 完成3.1节异常+3.2断言 | 3.1节异常完成 |
| 2020/5/22 | 18:40-21:00 | 测试异常是否正确处理 | 异常测试完成 |
| 2020/5/22 | 21:30-22:30 | 完成3.2节断言 | 完成 |
| 2020/5/23 | 13:20-15:45 | 完善测试用例，根据结果查找错误  即3.4节  Test for Robustness and Correctness | 完成 |
| 2020/5/24 | 7:40-11:00 | 完成3.3节日志功能 | 完成 |
| 2020/5/24 | 13:30-14:50 | 用spotbug发现bug并修改 | 完成 |
| 2020/5/24 | 15:40-18:00 | 完成3.6节debugging | 完成eventmanager与flightclient |
| 2020/5/25 | 13:00-15:20 | 完成3.6节debugging | 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 |
| 不熟悉java日志的使用 | 对照课件与网上的博客进行实践尝试 |
| Debugging一节不理解原程序的思想 | 根据piazza上同学们的提问以及老师的回答帮助理解 |
| 测试时日志文件中出现锁定错误：  "java.io.IOException: Couldn't get lock for xxx” | 网上查找资料，在stackoverflow中得到解答 |

# 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

在程序开发过程中，代码开发只是一部分，对于潜在bug以及各种异常情况的考虑还需要花费很多精力，这样才能使我们的程序有更好的体验，这样看来，之前写的程序似乎一直在关注了正确性而不是很重视健壮性，现在感觉都成了半成品的样子。这次实验让我体会到了健壮性的优点，也学会了各种工具，如spotbugs、EclEmma

## 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？

正确性只关注规约规定的事情，对于规约之外的事情没有限制，程序员可以自由发挥，使编程简单

健壮性要求容错，需要谨慎考虑规约之外的异常情况，并让程序尽量少的退出，程序员需要额外的考量与处理，要提供给用户最好的体验

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？）

我的理解是

①对于体验要求高的用户，需要追求更完善的处理

对于性能要求高的用户，则对于程序中次要部分可以放松考量

②对于与用户交互的外部部分，尽量考虑各种异常情况，将数据过滤输入给内部

内部部分则使用直接的报错，因为过滤后的数据应是满足条件的，错误说明是程序中的问题，从而让我们更早发现问题

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？

前者需要程序员进行额外的考量与处理，有更大的工作量，要提供给用户最好的体验

后者将责任全部推给client，工作量小，对程序员要求低，但用户体验差

我的理解是对于体验要求高的用户，需要追求更完善的处理

对于性能要求高的用户，则对于程序中次要部分可以放松考量

1. 过分谨慎的“防御”（excessively defensive）真的有必要吗？你如何看待过分防御所带来的性能损耗？如何在二者之间取得平衡？

我的理解是

①对于体验要求高的用户，需要追求更完善的处理

对于性能要求高的用户，则对于程序中次要部分可以放松考量

②对于与用户交互的外部部分，尽量考虑各种异常情况，将数据过滤输入给内部

内部部分则使用直接的报错，因为过滤后的数据应是满足条件的，错误说明是程序中的问题，从而让我们更早发现问题

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？

测试用例一定要全面，对于出错的用例可以移至main函数中进行debugger调试。每个对结果有影响的地方都可增加print语句/Assertion推测执行过程。

Assertion和log技术能够帮助我更有效的定位错误

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？

使用等价类和边界值的测试思想，测试用例囊括各个等价类部分

覆盖度%100也可能存在没有覆盖的等价类分支

1. Debug一个错误的程序，有乐趣吗？体验一下无注释、无文档的程序修改。

没有乐趣，无注释、无文档实在难以令别人理解

1. 关于本实验的工作量、难度、deadline。

适中

1. 到目前为止你对《软件构造》课程的评价和建议。

可以适当减少每一章开头部分的概念讲解，在后面相关技术上更细致一些讲解

1. 期末考试临近，你对占成绩60%的闭卷考试有什么预期？

希望能够线下考试!