

**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 | 余涛 |
| 学号 | 1180300829 |
| 班号 | 1803008 |
| 电子邮件 | 1063695334@qq.com |
| 手机号码 | 15586430583 |

**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 2](#_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 6](#_Toc39518190)

[3.2 Assertion and Defensive Programming 10](#_Toc39518191)

[3.2.1 checkRep()检查rep invariants 10](#_Toc39518192)

[3.2.2 Assertion/异常机制来保障pre-/post-condition 10](#_Toc39518193)

[3.2.3 你的代码的防御式策略概述 22](#_Toc39518194)

[3.3 Logging 22](#_Toc39518195)

[3.3.1 异常处理的日志功能 23](#_Toc39518196)

[3.3.2 应用层操作的日志功能 23](#_Toc39518197)

[3.3.3 日志查询功能 24](#_Toc39518198)

[3.4 Testing for Robustness and Correctness 29](#_Toc39518199)

[3.4.1 Testing strategy 29](#_Toc39518200)

[3.4.2 测试用例设计 29](#_Toc39518201)

[3.4.3 测试运行结果与EclEmma覆盖度报告 32](#_Toc39518202)

[3.5 SpotBugs tool 35](#_Toc39518203)

[3.6 Debugging 38](#_Toc39518204)

[3.6.1 EventManager程序 38](#_Toc39518205)

[3.6.2 LowestPrice程序 39](#_Toc39518206)

[3.6.3 FlightClient/Flight/Plane程序 40](#_Toc39518207)

[4 实验进度记录 42](#_Toc39518208)

[5 实验过程中遇到的困难与解决途径 43](#_Toc39518209)

[6 实验过程中收获的经验、教训、感想 43](#_Toc39518210)

[6.1 实验过程中收获的经验和教训 43](#_Toc39518211)

[6.2 针对以下方面的感受 43](#_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

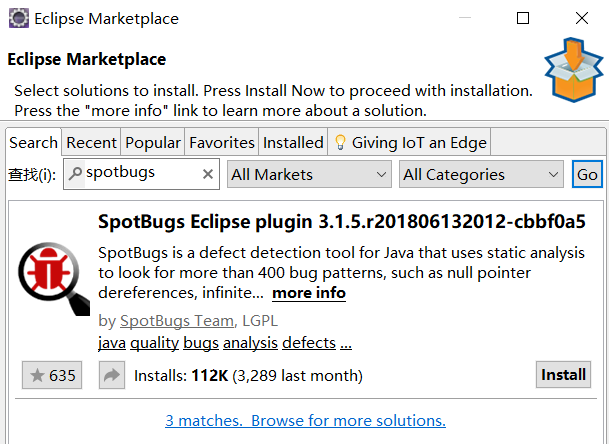
# 实验环境配置

本次实验需要安装SpotBugs来进行代码bug的静态检测。

具体安装过程为：

1.在eclipse找到help并选择eclipse marketplace

2、在find中输入spotbugs找到后点击install即可



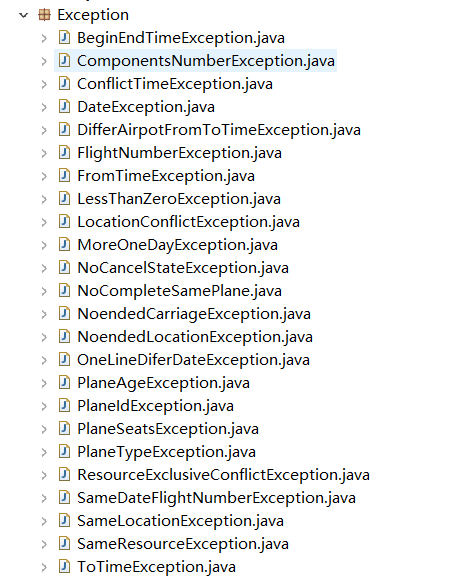
在这里给出你的GitHub Lab4仓库的URL地址（Lab4-学号）。

<https://github.com/ComputerScienceHIT/Lab4-1180300829>

# 实验过程

## Error and Exception Handling

创建了一个Exception包，里面创建了各种异常类：其中包括文本中的错误异常，客户端操作时的异常

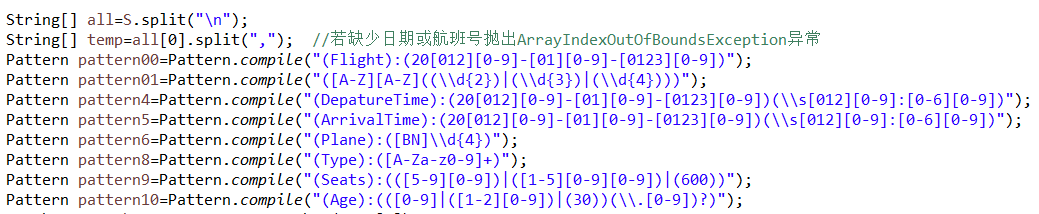


### 处理输入文本中的三类错误

**3.1.1.1 输入文件中存在不符合语法规则的语句**

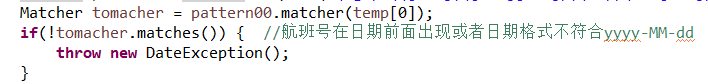
对原有的Parser包中的Parser类进行改造，将代表航班计划项的字符串进行拆分，对每一行内容进行语法规则的判断。

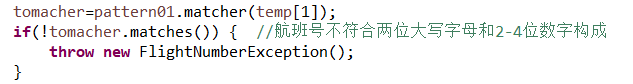
首先按照正则表达式定义一系列Pannter变量，然后需要语法判断的每一行进行匹配，若不匹配则抛出异常：

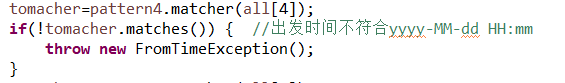


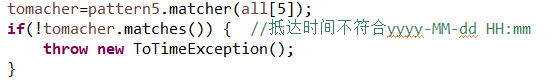
对于各种异常：

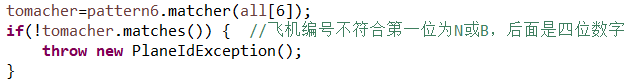




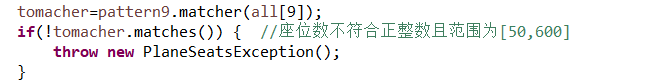


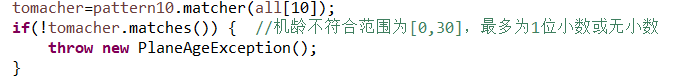












其中元素定义的分量不合适的异常在FlightScheduleApp中抛出



然后在FlightScheduleApp中进行异常的捕获，若捕获到这些异常则输出错误信息并结束文件读取：

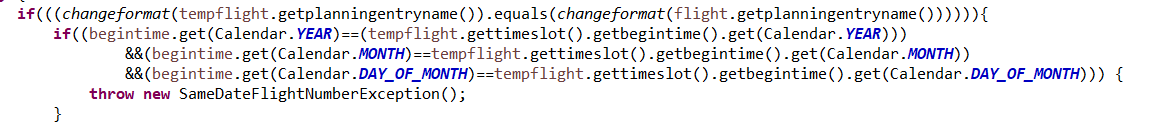




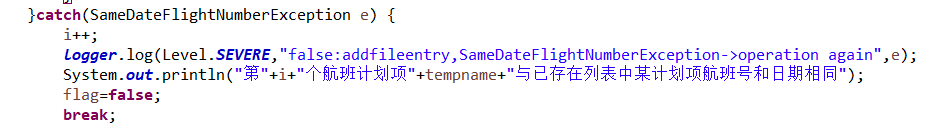
**3.1.1.2 输入文件中存在标签完全一样的元素**

在FlightScheduleApp中进行异常的捕获

对多个航班计划项的“日期，航班号”信息完全一样的情况，若出现则抛出SameDateFlightNumberException异常



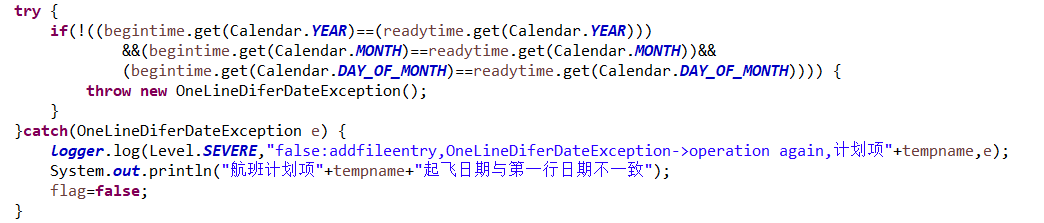
然后进行捕获，输出异常信息，并结束读入文件



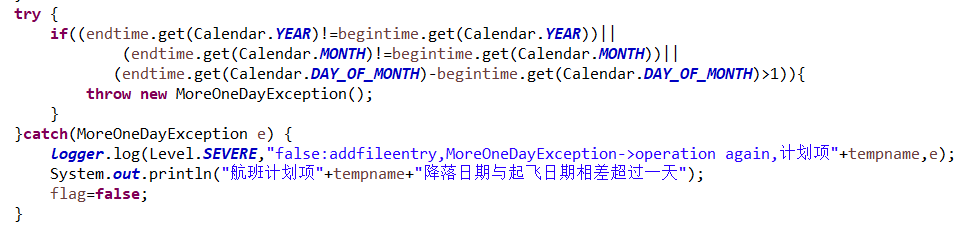
**3.1.1.3 输入文件中中各元素之间的依赖关系不正确**

在FlightScheduleApp中进行异常的捕获

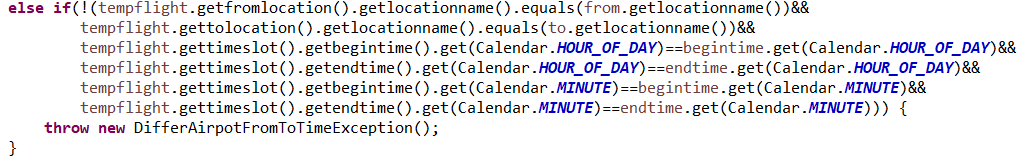
1.第一行出现的航班日期与内部出现的起飞时间的日期不一致，抛出OneLineDiferDateException异常并进行捕获

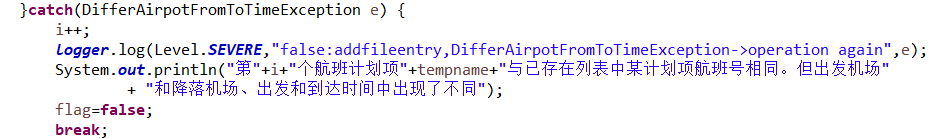


2.降落时间中的日期与航班日期差距大于1天，抛出MoreOneDayException异常并进行捕获

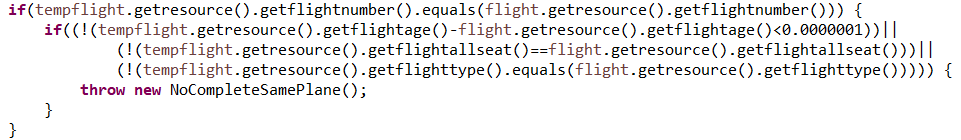


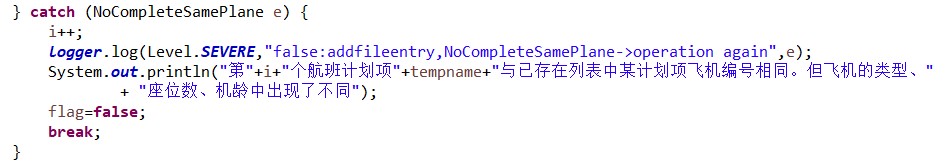
3. 同一个航班号，虽然日期不同，但其出发或到达机场、出发或到达时间有差异，抛出DifferAirpotFromToTimeException异常并进行捕获



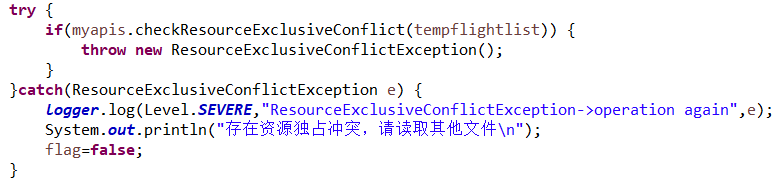


4. 在不同航班计划项中出现编号一样的飞机，但飞机的类型、座位数或机龄却不一致，抛出NoCompleteSamePlane异常并进行捕获





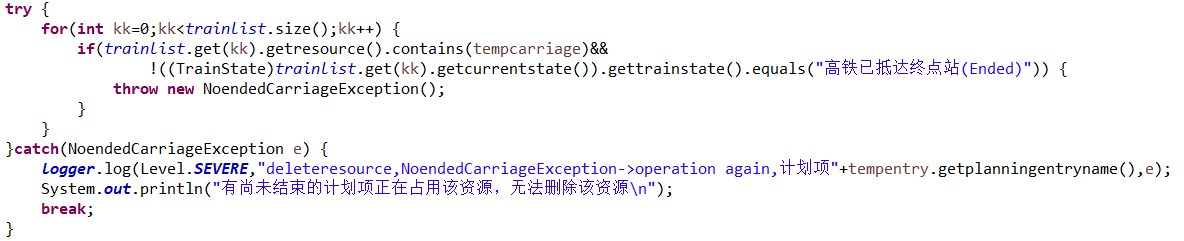
5.若航班计划项之间存在资源独占冲突，抛出ResourceExclusiveConflictException异常并进行捕获



### 处理客户端操作时产生的异常

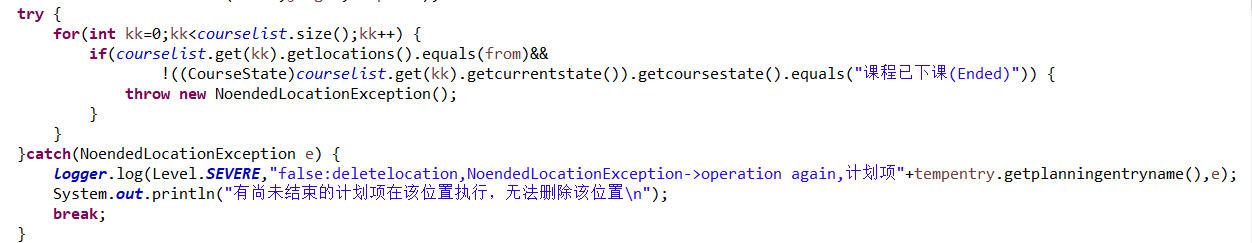
**3.1.2.1 在删除某资源的时候，如果有尚未结束的计划项正在占用该资源**

只有TrainSchuduleApp里面有删除资源的操作，遍历所有计划项，找到含有该资源的计划项，如果该计划项状态不为Ended，则抛出NoendedCarriageException异常并进行捕获



**3.1.2.2 在删除某位置的时候，如果有尚未结束的计划项会在该位置执行**

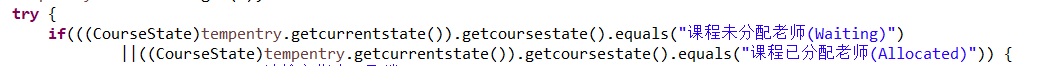
只有CourseCalendarApp里面有删除位置的操作，遍历所有计划项，找到含有该位置的计划项，如果该计划项状态不为Ended，则抛出NoendedLocationException异常并进行捕获



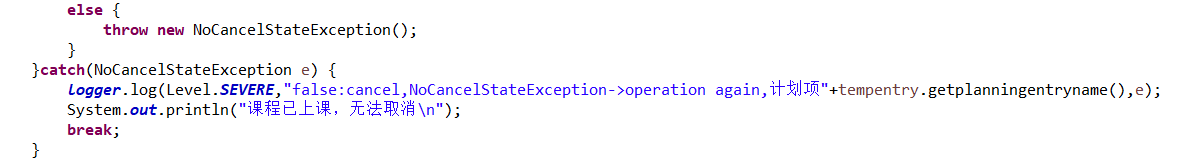
**3.1.2.3 在取消某计划项的时候，如果该计划项的当前状态不允许取消**

三个App均有取消计划项的操作，判断当前计划项的状态是否允许取消，不允许则抛出NoCancelStateException并捕获

对于CourseCalendarApp：



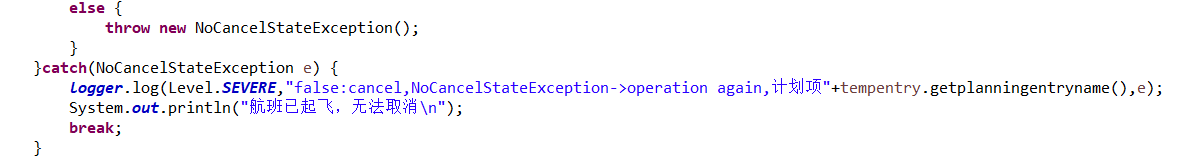
……

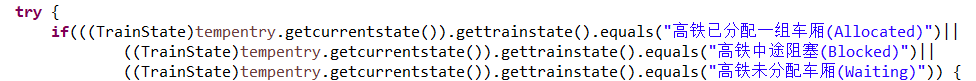


对于FlightScheduleApp：

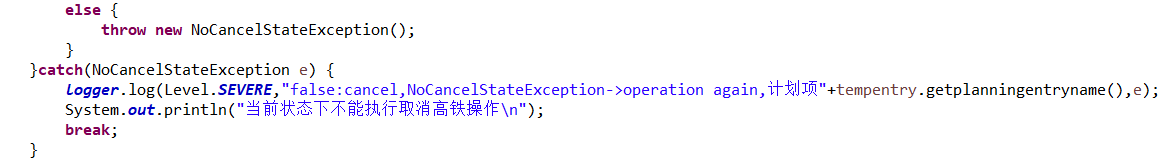


……



对于TrainScheduleApp：  


……



对于其他计划项的操作，在Lab3中已经考虑了非法状态的情况，只是未用异常作为处理方法，故不需要修改。已完成健壮性。

**3.1.2.4 在为某计划项分配某资源的时候，如果分配后会导致与已有的其他计划项产生“资源独占冲突”**

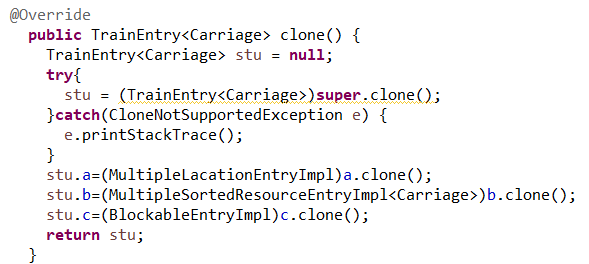
1.在每个计划项里面实现clone方法，完成对计划项的深拷贝，防止判断资源独占冲突时发生对原计划项发生了修改。

具体实现为：

继承接口：

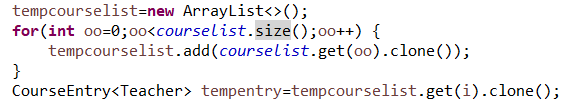


实现clone方法：



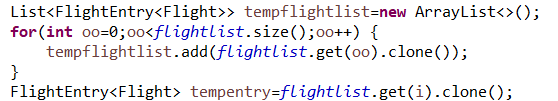
其他计划项同理。

2.先构建一个计划项集合的克隆，和待加入资源的计划项的克隆，然后将该克隆计划项放入克隆集合中，判断该集合中是否存在资源独占冲突，若存在资源独占冲突，则抛出ResourceExclusiveConflictException异常并进行捕获：

对于CourseCalendarApp：

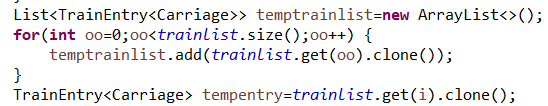
……



对于FlightScheduleApp：

……



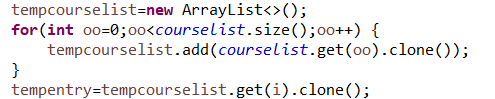
对于TrainScheduleApp：  


……

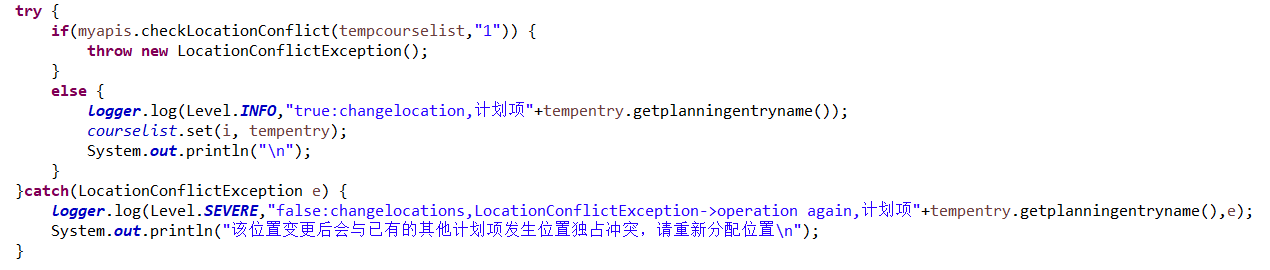


**3.1.2.5 在为某计划项变更位置的时候，如果变更后会导致与已有的其他计划项产生“位置独占冲突”**

只有CourseCalendarApp能够成功变更位置且位置不可共享，所以与3.1.2.4同理，先构建克隆集合和克隆计划项，然后判断是否存在位置独占冲突，存在冲突则抛出LocationConflictException异常并进行捕获：

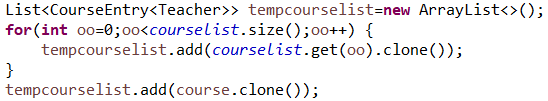


……

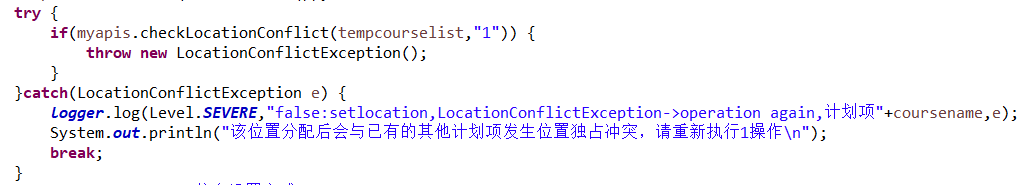


**3.1.2.6 在为某计划项分配位置的时候，如果分配后会导致与已有的其他计划项产生“位置独占冲突”**

只有CourseCalendarApp位置不可共享，所以与3.1.2.5同理，先构建克隆集合和克隆计划项，然后判断是否存在位置独占冲突，存在冲突则抛出LocationConflictException异常并进行捕获：



……

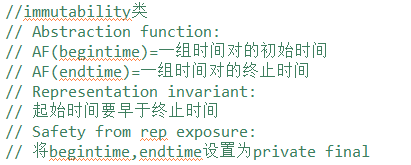


## Assertion and Defensive Programming

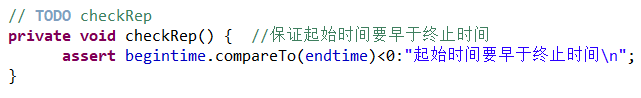
### checkRep()检查rep invariants

**3.2.1.1 Timeslot**

AF,RI如下：

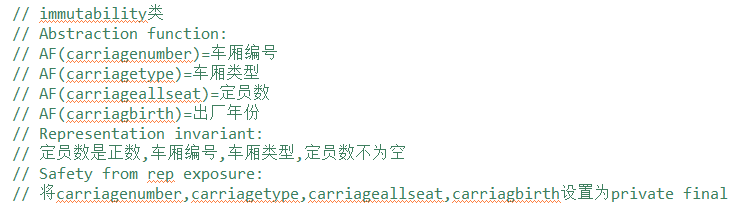


checkrep()：

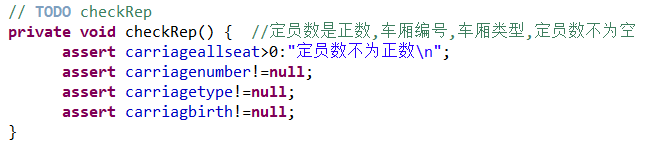


**3.2.1.2 Carriage**

AF,RI如下：

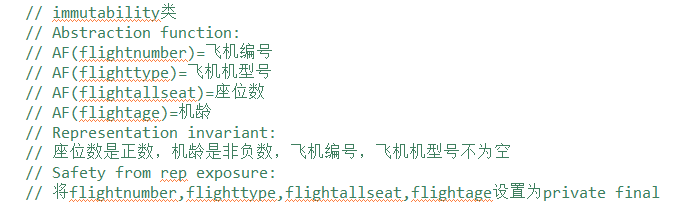


checkrep()：

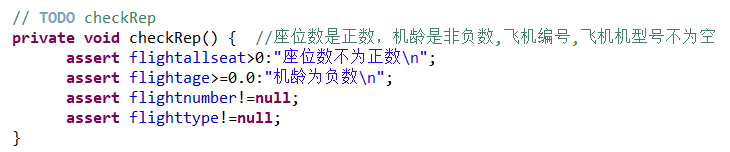


**3.2.1.3 Flight**

AF,RI如下：

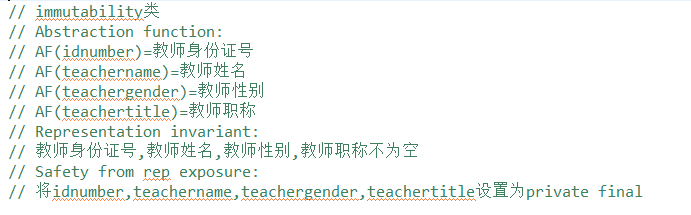


checkrep()：

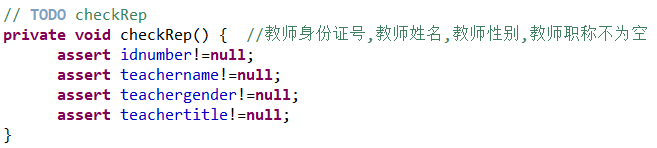


**3.2.1.4 Teacher**

AF,RI如下：

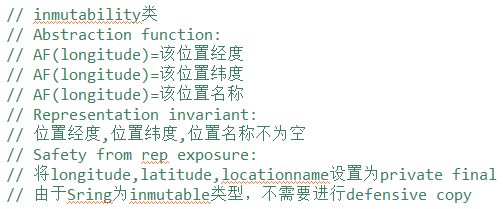


checkrep()：

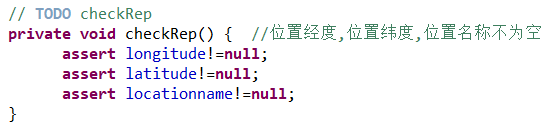


**3.2.1.5 CommonLocation**

AF,RI如下：

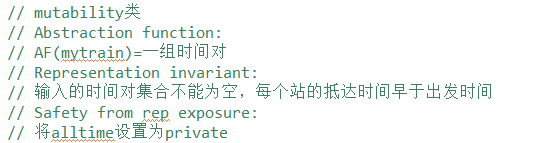


checkrep()：

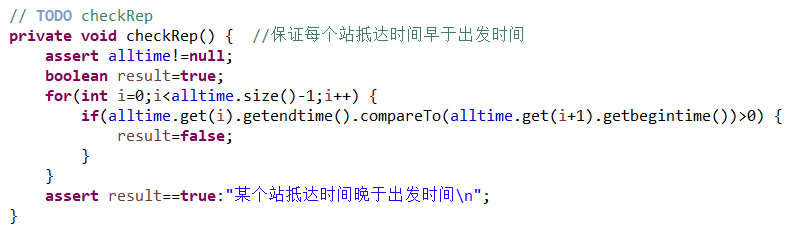


**3.2.1.6 BlockableEntryImpl**

AF,RI如下：

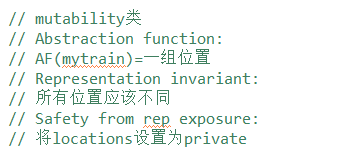


checkrep()：

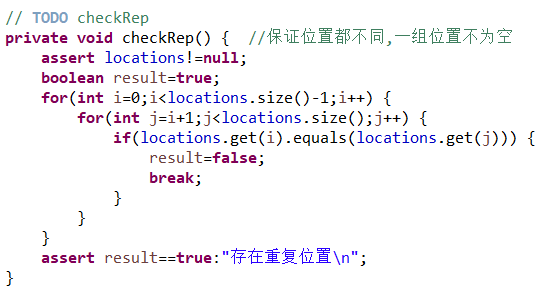


**3.2.1.7 MultipleLacationEntryImpl**

AF,RI如下：

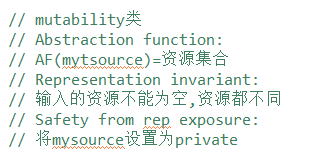


checkrep()：

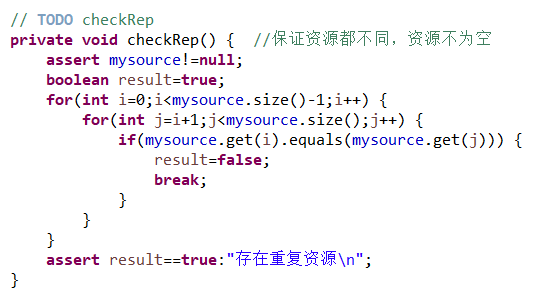


**3.2.1.8 MultipleSortedResourceEntryImpl**

AF,RI如下：

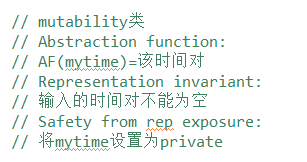


checkrep()：

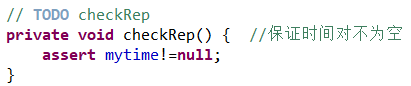


**3.2.1.9 NoBlockableEntryImpl**

AF,RI如下：

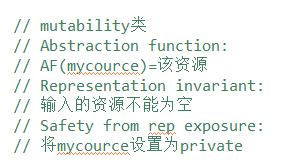


checkrep()：

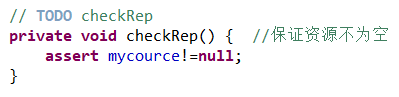


**3.2.1.10 OneDistinguishResourceEntryImpl**

AF,RI如下：

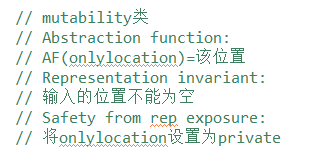


checkrep()：

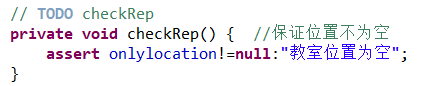


**3.2.1.11 OneLocationEntryImpl**

AF,RI如下：



checkrep()：

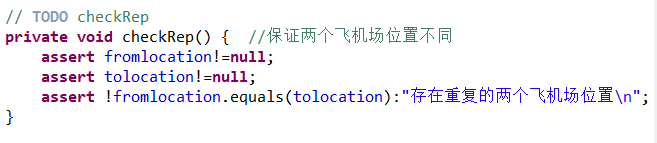


**3.2.1.12 TwoLocationEntryImpl**

AF,RI如下：



checkrep()：

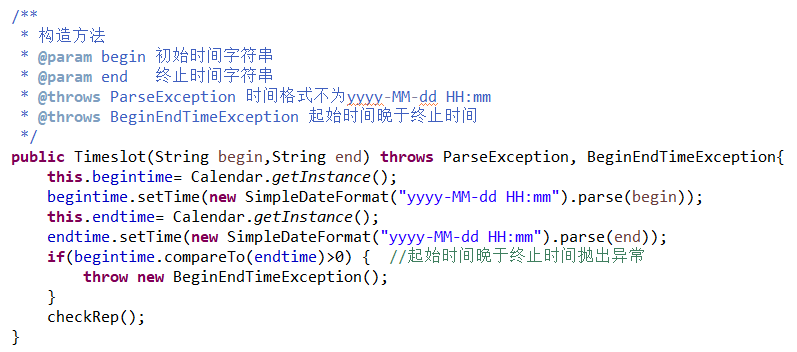


### Assertion/异常机制来保障pre-/post-condition

**3.2.2.1 Timeslot**

1.创建Timeslot对象时：

对precondition检测：时间格式必须是yyyy-MM-dd HH:mm，起始时间早于终止时间。若不满足则抛出异常



2.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.2 Carriage**

1.创建Carriage对象时：

对precondition检测：定员数是正数。若不满足则抛出异常



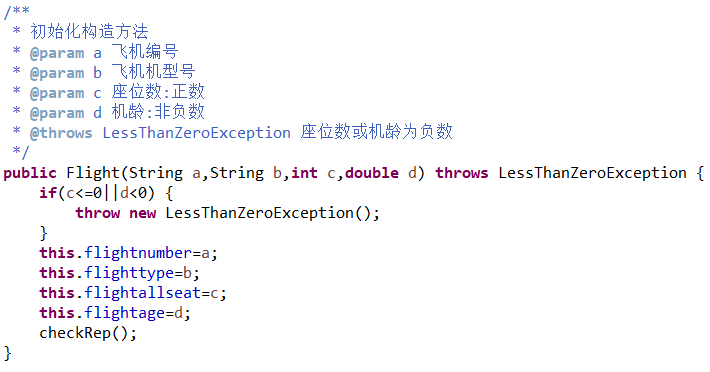
2.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.3 Flight**

1.创建Flight对象时：

对precondition检测：座位数是正数，机龄是非负数。若不满足则抛出异常



2.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.4 Teacher**

1.返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.5 BlockableEntryImpl**

1.settimeslot：

对precondition检测：每个站抵达时间早于出发时间。若不满足则抛出异常



对posetcondition检测：加上checkrep()

**3.2.2.6 MultipleLacationEntryImpl**

1. setlocations：

对precondition检测：不能存在相同位置。若不满足则抛出异常



对posetcondition检测：加上checkrep()

2.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.7 MultipleSortedResourceEntryImpl**

1. setresource：

对precondition检测：资源不能相同。若不满足则抛出异常



对posetcondition检测：加上checkrep()

2. changeresource：

对precondition检测：直接沿用Lab3中的检查即可(没有使用异常处理机制)：



3. addresource：

对precondition检测：直接沿用Lab3中的检查即可(没有使用异常处理机制)：



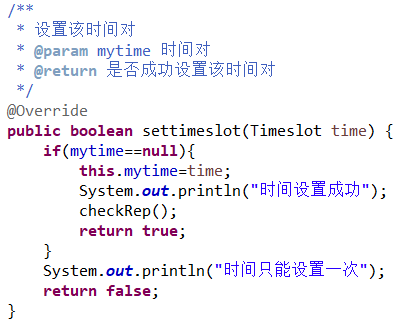
4.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.8 NoBlockableEntryImpl**

1. settimeslot：

对precondition检测：Timeslot类已经完成了检测。



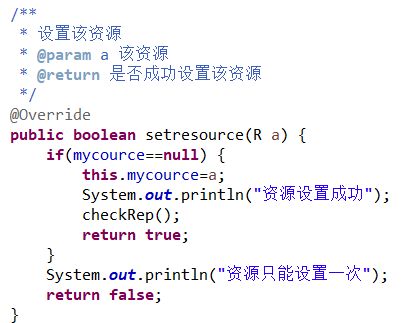
对posetcondition检测：加上checkrep()

2.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.9 OneDistinguishResourceEntryImpl**

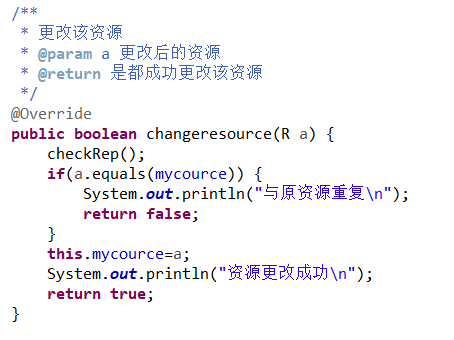
1. setresource：



对postcondition检测：加上checkrep()

2. changeresource：

对precondition检测：直接沿用Lab3中的检查即可(没有使用异常处理机制)：

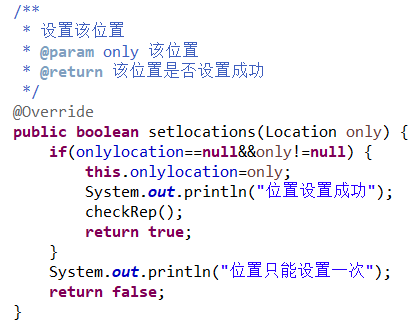


3.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.10 OneLocationEntryImpl**

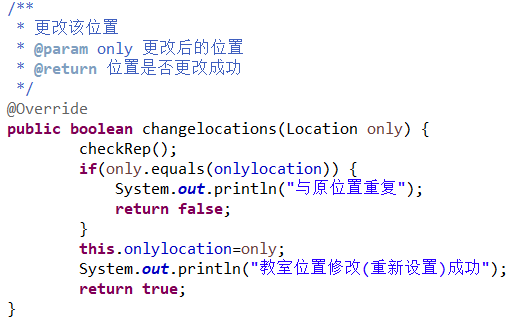
1. setlocations：



对postcondition检测：加上checkrep()

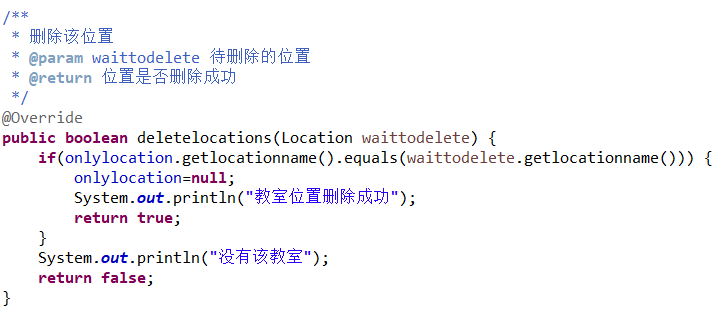
2. changelocations：

对precondition检测：直接沿用Lab3中的检查即可(没有使用异常处理机制)：



2. deletelocations：

对precondition检测：直接沿用Lab3中的检查即可(没有使用异常处理机制)：



4.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.11 TwoLocationEntryImpl**

1. setlocations：

对precondition检测：两个位置不能相同。若相同则抛出异常。



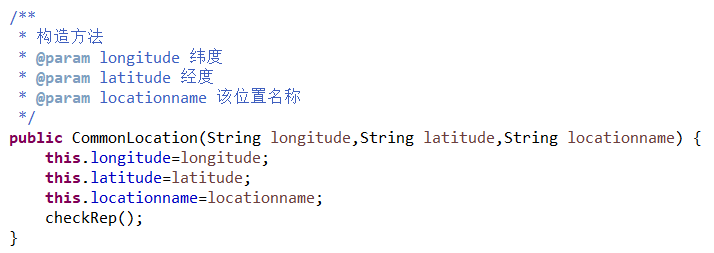
对postcondition检测：加上checkrep()

2.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

**3.2.2.12 CommonLocation**

1.创建CommonLocation时：



对postcondition检测：加上checkrep()

2.其他返回字段方法：

对posetcondition检测：在每一个返回字段的方法前面加上checkrep()

### 你的代码的防御式策略概述

1.对于前置条件的防御策略：

在所有需要检测前置条件的的方法中，遇到变量不满足前置条件的情况下，抛出异常。

2.对于后置条件的防御策略：  
 在所有需要获得返回字段的方法返回值前执行一个checkrep()检查表示不变量防止值被改变。

3.对于表示不变量的防御策略：

通过定义checkrep方法，在checkrep中定义一系列断言assert来保证不变量不被改变。

4.对于用户输入错误信息的防御策略：

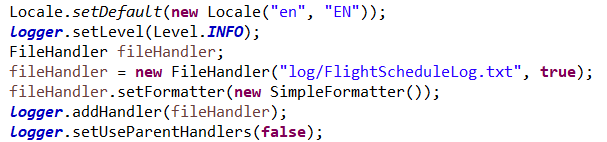
在Client端，即App类中，若用户输入错误信息，由于在API中完成了防御策略，所以会抛出异常，通过捕获异常输出错误信息并提示用户重新输入来保证程序防御性良好，不会崩溃。

## Logging

本人采用了jdk自带的java logging完成日志的记录，即调用java的库 java.util.logging。先进行一些日志初始化的设置：



定义一个全局静态变量，调用已经创建的相关logger，就可以实现日志记录了



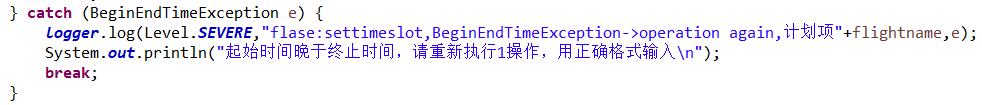
通过这些初始化操作，能够实现向FlightScheduleLog.txt中写入日志的功能，并且日志不会在控制台输出

### 异常处理的日志功能

对于三个App所有产生的异常，在每个catch之前，设置SEVERE级别的日志，为了最后输出日志比较美观，所以记录日志我采用了如下格式：

true/false:操作名称，异常名->异常的处理结果，计划项名字，堆栈信息（包含异常的具体信息）

示例：



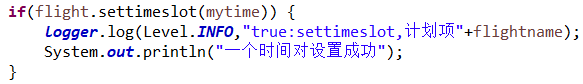
### 应用层操作的日志功能

对于不抛出异常的应用中的其他操作，设置INFO级别的日志

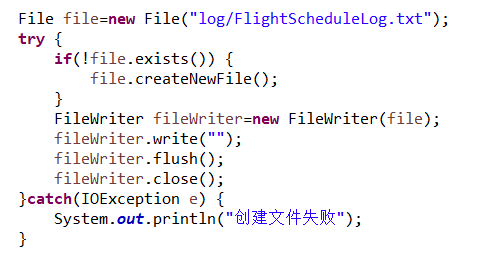
为了最后输出日志比较美观，所以记录日志我采用了如下格式：

true/false:操作名称，计划项名字

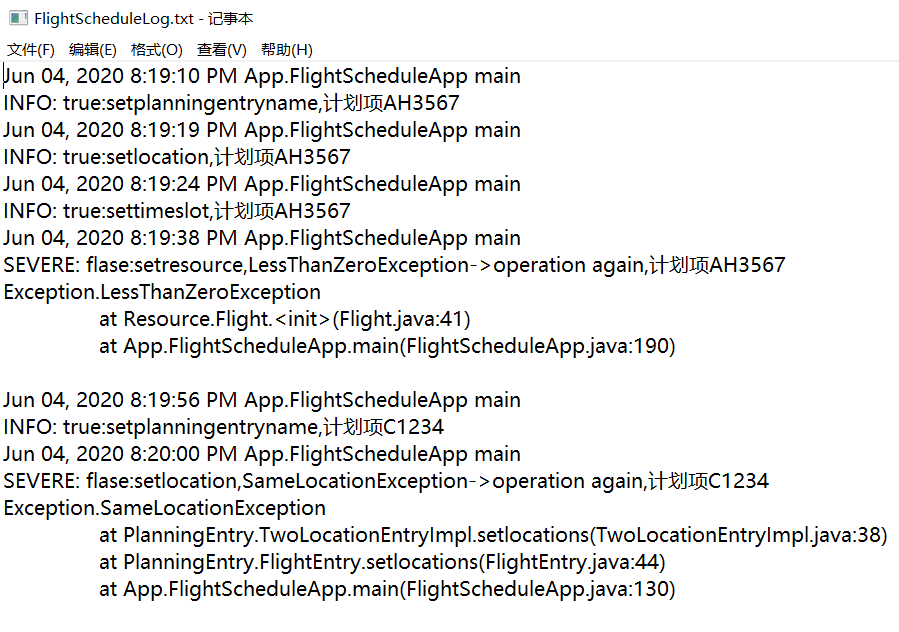
示例：



另外，在每次运行新的App程序时，需要清空原日志文件，如下：



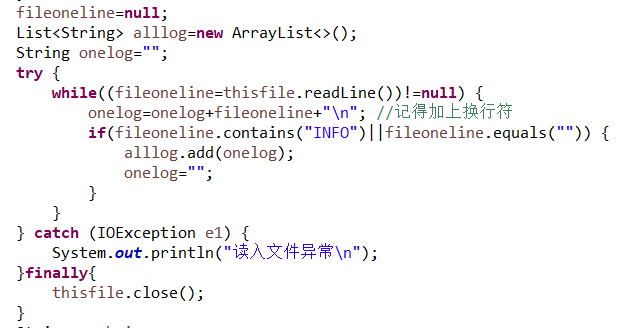
异常处理和应用层操作后的日志文本效果为：



### 日志查询功能

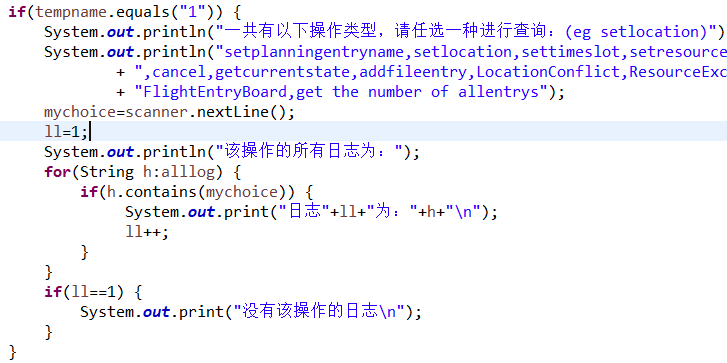
以FlightScheduleApp为例：首先需要将日志的文本文件中的一条条日志分开。

每次读一行文本内容，如果该行中包含INFO或者该行为空行，则将该行和之前行合为一个字符串，作为一条日志。具体实现为：

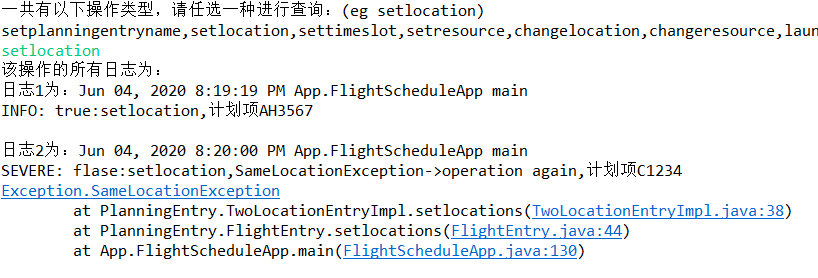


**3.3.3.1 按操作类型查询**

输入一个操作，如果某个日志含有该操作，则输出该日志

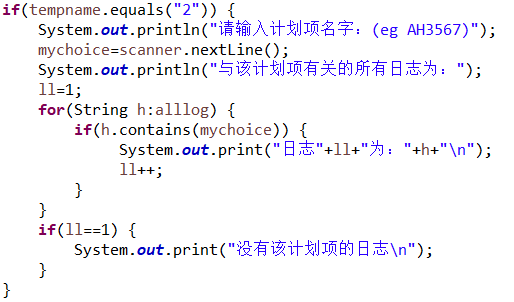


示例：

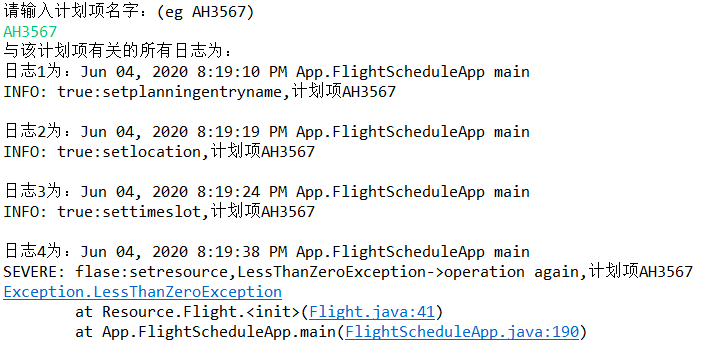


**3.3.3.2 按计划项名字查询**

输入一个计划项名字，如果某个日志含有该计划项，则输出该日志



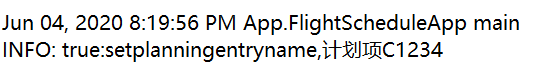
示例：



**3.3.3.3 按时间段查询**

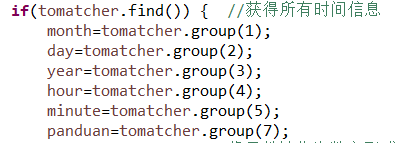
输入一个时间段，如果某个日志的时间在这个时间段之间，则输出该日志

由于文本文件中的日志计划项的格式为



需要用正则表达式对其进行匹配，取出时间：

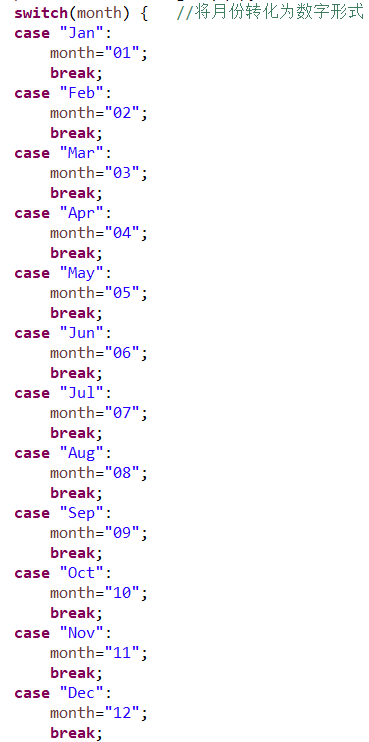




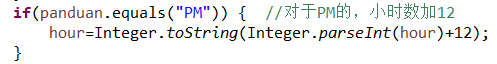
分别获得了时间中的年，月，日，小时，分钟，上午还是下午

然后将这些String转化为标准格式：

月：对于每种英文月份，对其转换。



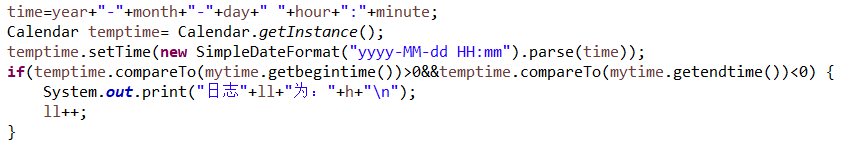
小时：对于PM的小时，需要将其加12



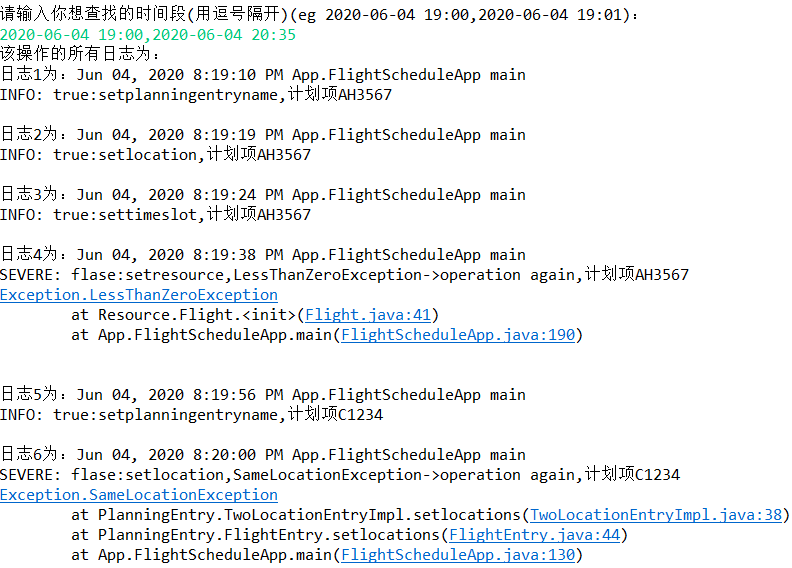
对于AM的小时，若其小于10，则需要在开头加个0



然后将这些分量合为一个时间变量，再进行判断，若满足条件的日志则输出：

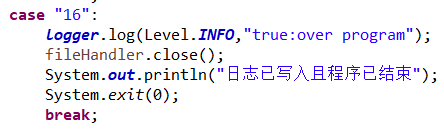


示例：



**3.3.3.4 保存日志到文件**

最后结束程序，保存日志

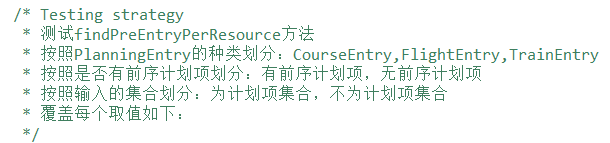




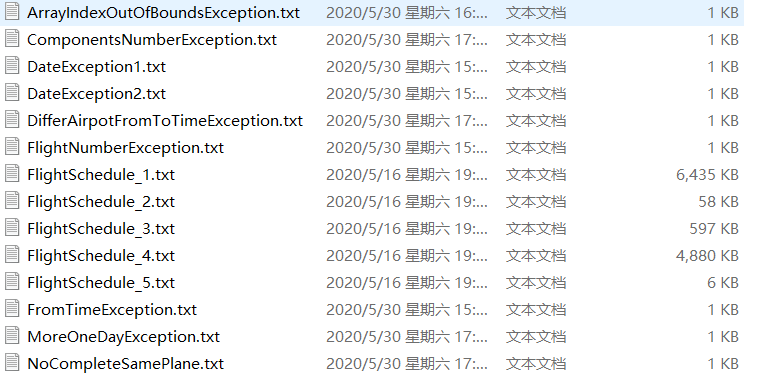
## Testing for Robustness and Correctness

### Testing strategy

对每种方法，按照等价类划分的思想添加Testing Strategy，以findPreEntryPerResourcetest为例



对于3.1中的各种非法情形，构建了一系列每种异常的文件，进行测试

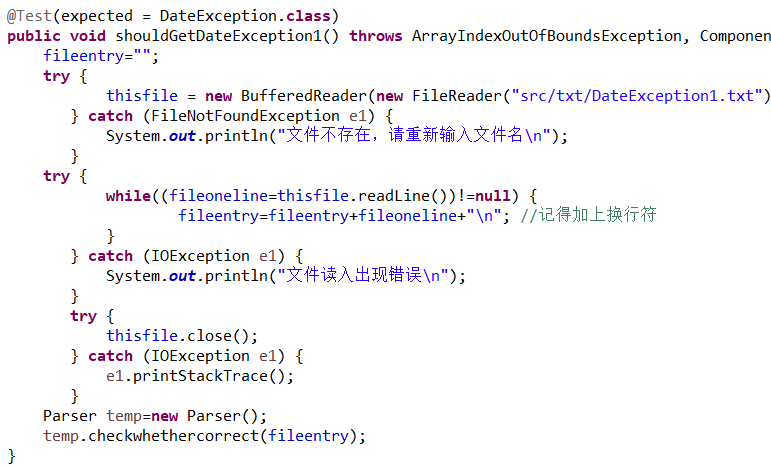




### 测试用例设计

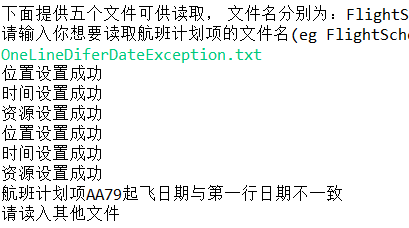
由于在Lab3中已经完成了各种方法的测试，只需要增加抛出异常的测试即可。

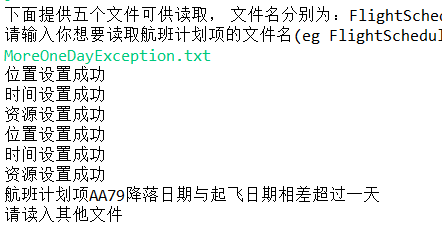
1.API中对于异常文本文件的测试，以DateException为例，在测试类中测试：

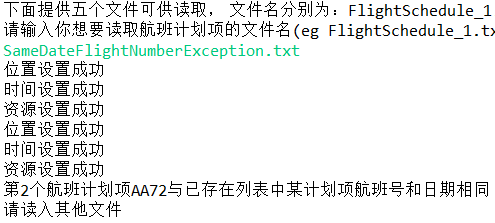


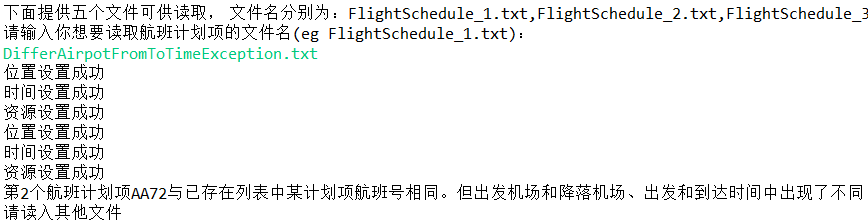
最后测试了所有API中抛出的异常

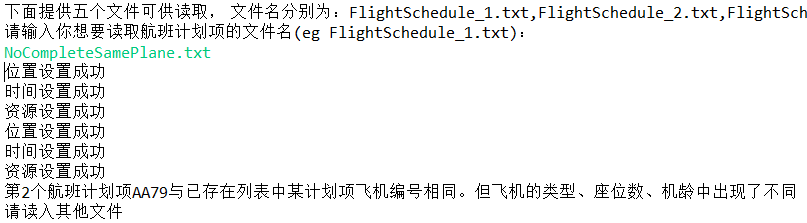
2.Client端对于异常文本文件的测试，直接读入异常文件即可：



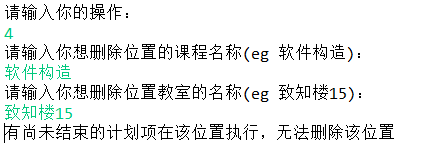


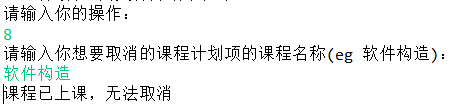


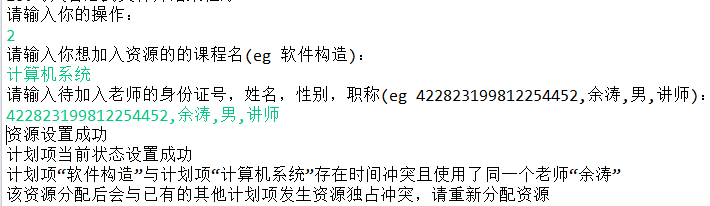


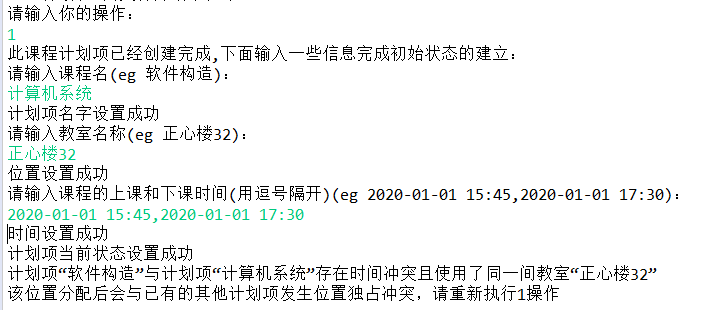


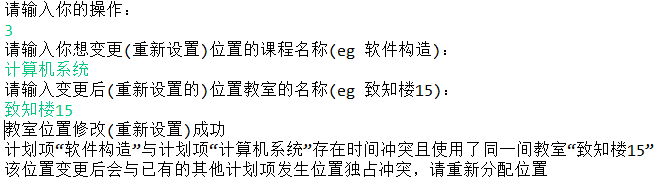
3.Client端对于操作异常的测试，执行异常操作即可：（以CourseCalendarApp和TrainScheduleApp为例）（由于非法情况太多不一一例举，只例举3.1节中提到的非法操作）

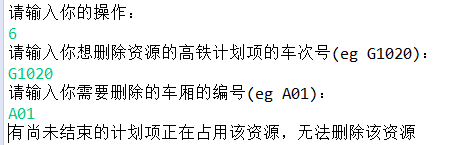








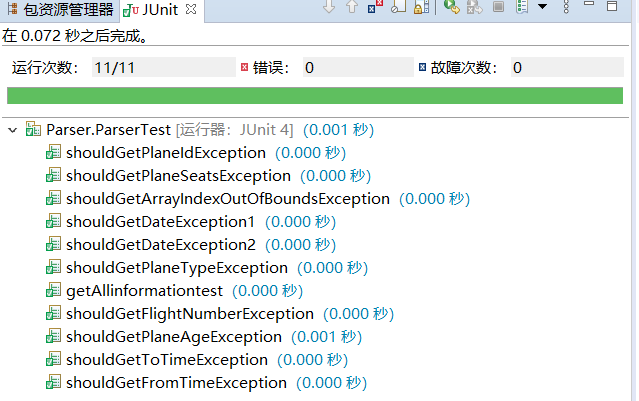




### 测试运行结果与EclEmma覆盖度报告

由于App和Board无法测试，所以只测试了其他类：

异常测试：

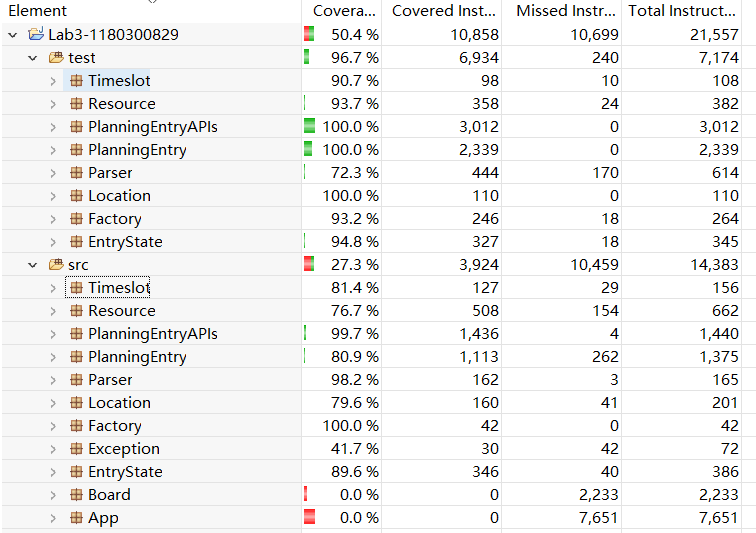


所有测试：

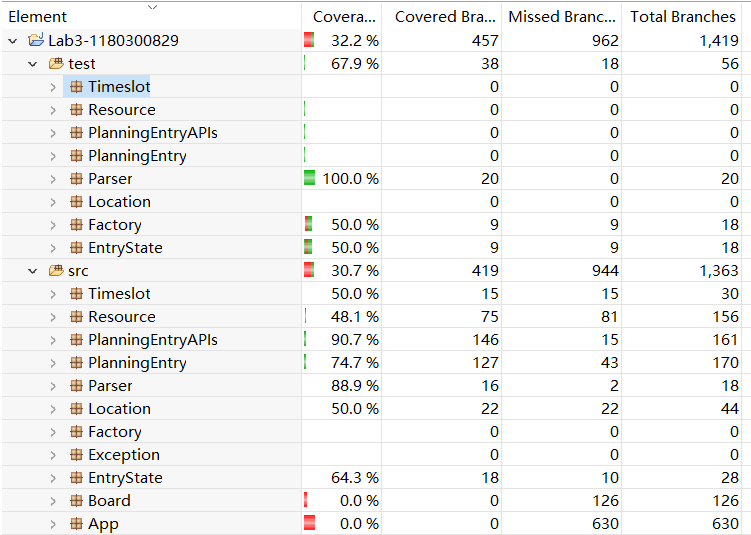


由于equals方法很难测全，所以可能部分覆盖度不高

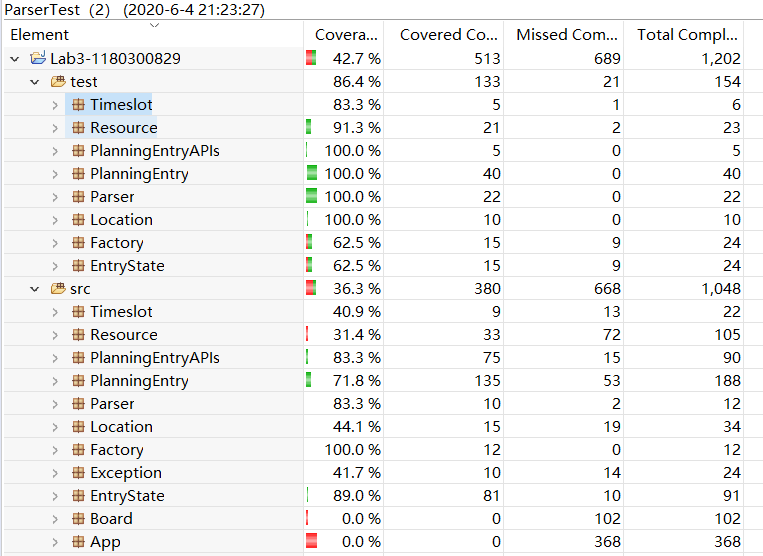
语句覆盖度：



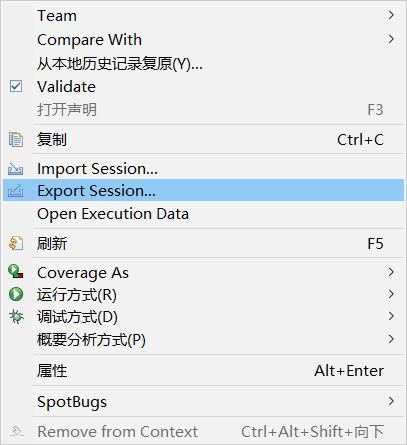
分支覆盖度：

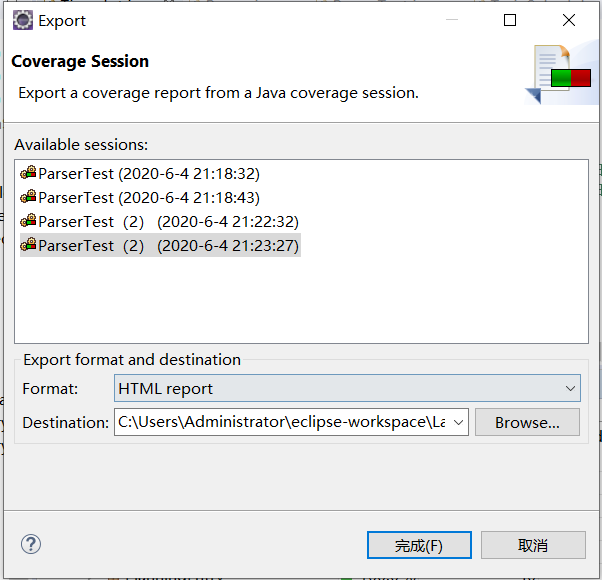


路径覆盖度：



生成测试报告：

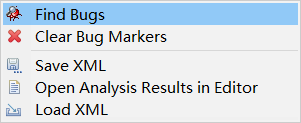




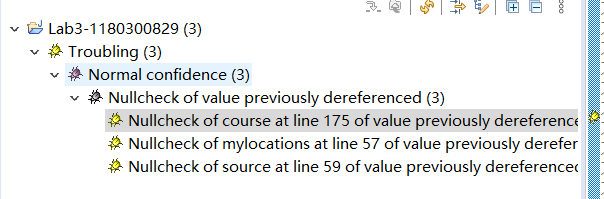


## SpotBugs tool

右键项目，SpotBugs，然后FindBugs



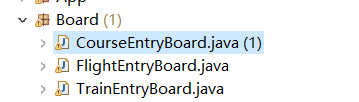
发现了三个bug：

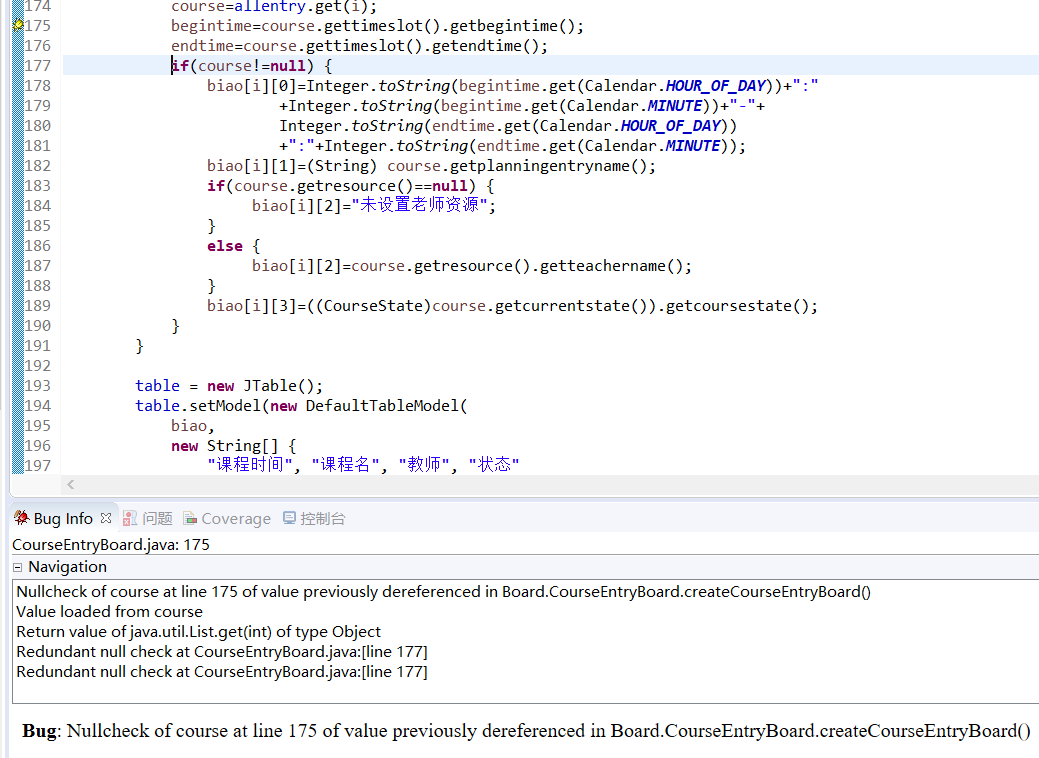


针对三个bug：

bug1：无效的null检测，删除null检测即可

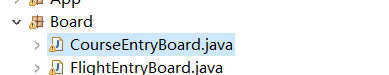
删除前：





删除后：





bug2：无效的null检测，删除null检测即可

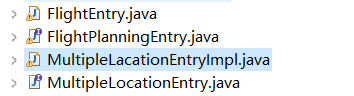
删除前：





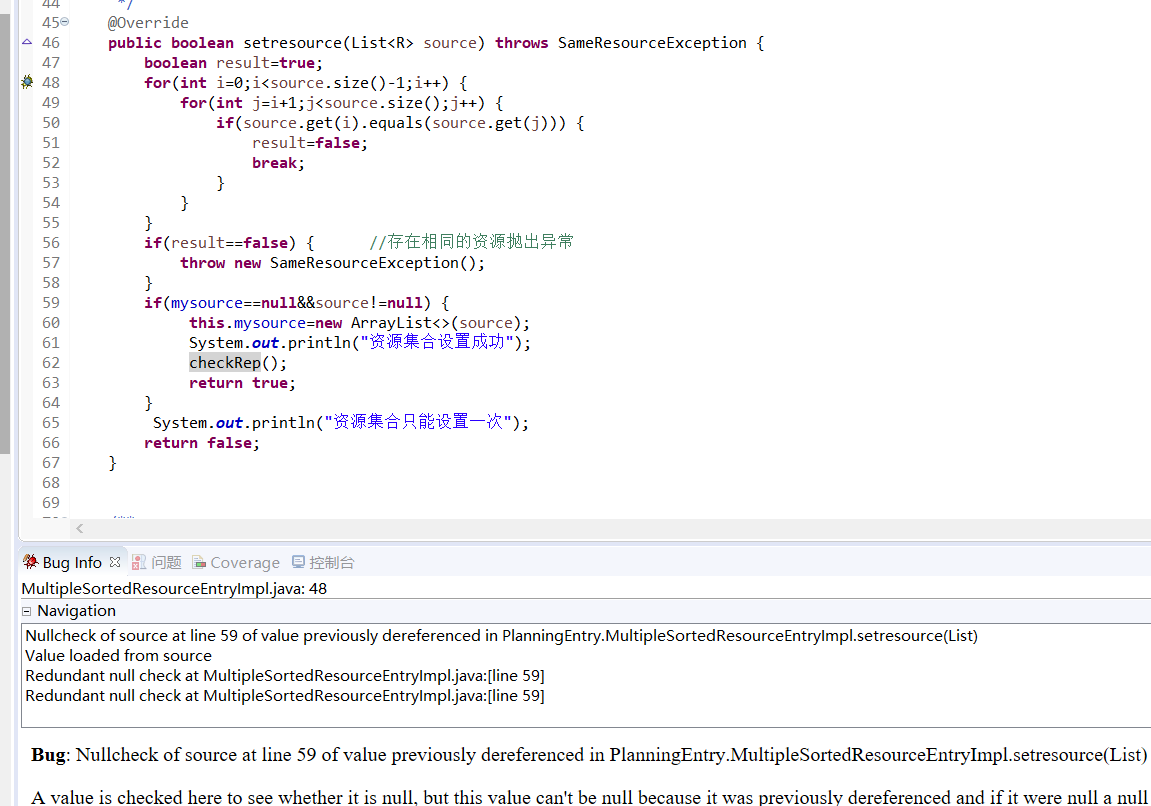
删除后：

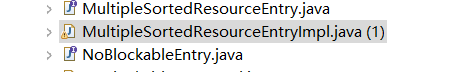




bug3：无效的null检测，删除null检测即可

删除前：





删除后：





然后就没有bug了

## Debugging

### EventManager程序

1.理解待调试程序的代码思想：

用了前缀和的思想，每读入一个时间段，对开始时间标记+1，结束时间标记-1，从而通过遍历所有时间节点，进行标记相加，最大的那个标记便是重叠时间段数的最大值

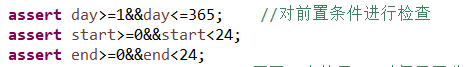
2.发现并定位错误的过程

先用spotbugs看有没有静态错误，然后一行一行分析，理解代码思想进行修改

3.你如何修正错误

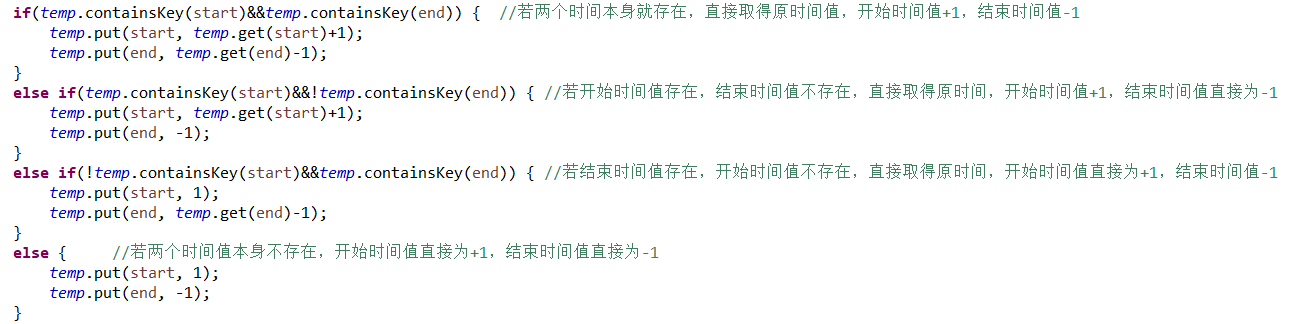
已在程序中注释，截图如下：

增加了前置条件检查：

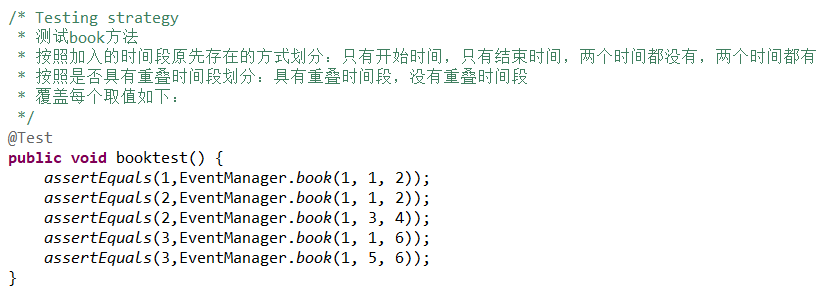


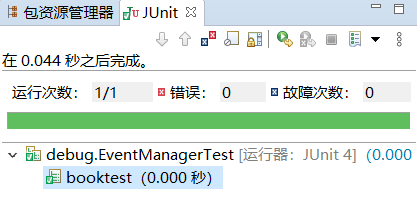
代码内容的修改：





4.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

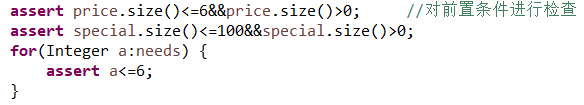
思想主要为贪心，先假设以零食价格购入所有商品，然后将每个special offer加进去，不断用新需求迭代比较产生最优解

2.发现并定位错误的过程

先用spotbugs看有没有静态错误，然后一行一行分析，理解代码思想进行修改

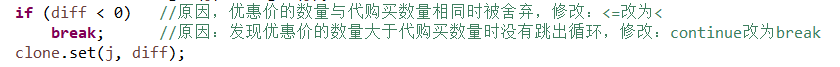
3.你如何修正错误

增加了前置条件检查：



代码内容的修改：







4.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

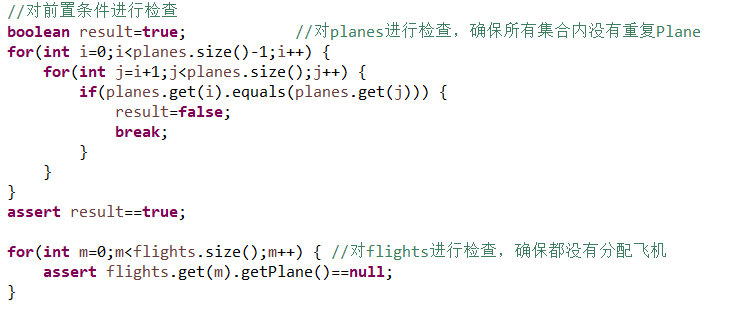
通过依次例举每个航班，安排飞机，如果没有与其他已经分配的航班冲突，则分配，最后确认是否能够将所有航班分配完成且没有时间冲突

2.发现并定位错误的过程

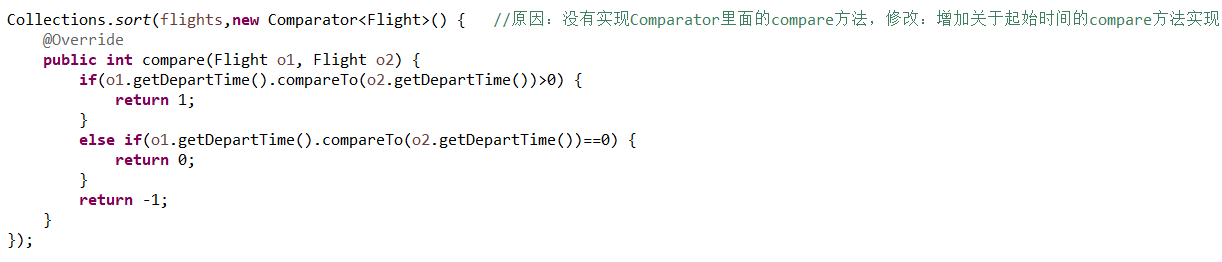
先用spotbugs看有没有静态错误，发现在Plane中用了==来判断两个String是否相等。然后编译器发现没有重写sort方法，并且用><来比较两个Calendar对象的大小。然后一行一行分析，理解代码思想进行修改

3.你如何修正错误

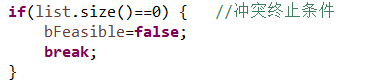
增加了前置条件检查：



代码内容的修改：

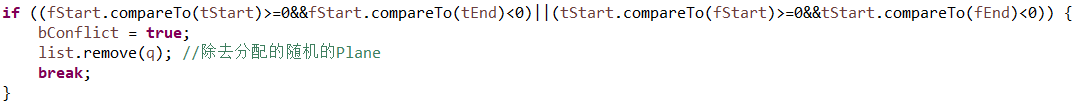




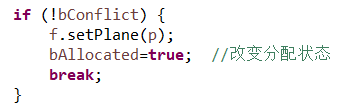




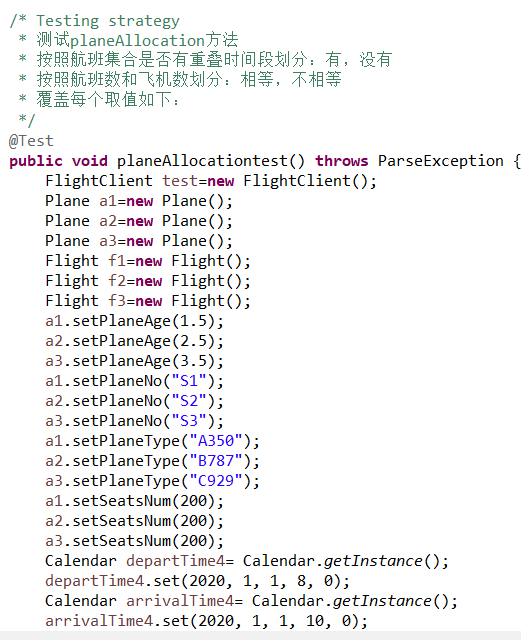




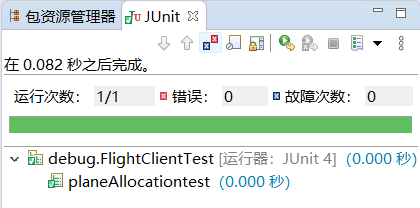




4.修复之后的测试结果



……



# 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 |
| 5.26 | 18:00-22:00 | 所有异常的抛出和捕获 | 未完成 |
| 5.27 | 16:00-22:30 | 所有异常的抛出和捕获 | 完成 |
| 5.28 | 19:00-23:00 | 所有类的防御式改造 | 完成 |
| 5.29 | 18:00-22:30 | 学习logging并尝试写入文件 | 未完成 |
| 5.30 | 18:00-22:00 | 完成日志的建立 | 完成 |
| 5.31 | 17:50-22:20 | 补全test用例 | 完成 |
| 6.1 | 19:00-20:30 | 使用soptbugs找bug | 完成 |
| 6.2 | 20:00-21:40 | EventManager的debugging和测试 | 完成 |
| 6.3 | 19:00-20:30 | LowestPrice和FlightClient的debugging和测试 | 完成 |
| 6.4 | 19:00-21:30 | 撰写报告 | 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 |
| 如何使用logging | 网上查找相关资料学习 |
| 如何读取文本中每一个日志的时间 | 经过不断分析，最终用正则表达式读出时间中的各个分量，然后转化为Calendar |
| 使用spotbugs改变错误后仍然报错 | 改错后没有保存代码 |

# 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

通过对Lab3进行改造，提升了Lab3程序的健壮性，使程序不容易崩溃。由于Lab3之前写的不太好，导致代码量太多，很多过程很繁琐。以后需要更好的布局代码框架。

## 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？

健壮性主要针对Client，是程序员为了防止用户输入错误信息而做的防御，

正确性主要针对程序员

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？

前者是程序健壮性的考虑，要求程序员对异常输入进行处理。后者是Client的考虑，需要用户输入正确的数据。在有很多输入情况下考虑前者，

在输入很少并且固定的情况下考虑后者。

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？

没必要。很浪费性能。对Client提更高的要求。

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？

先用SpotBugs找出代码的静态错误，然后进行代码逐行的分析，先找语法粗我不，再找内容错误。会。

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？

完成对所有等价类和边界条件的测试，以及非法输入的测试。

不是。

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。

有乐趣。

1. 关于本实验的工作量、难度、deadline。

工作量适中，就是报告内容太多，太繁琐了。难度适中。deadline合适。

1. 到目前为止你对《软件构造》课程的评价和建议。

学到了很多真本领。希望能将实验报告内容减少，侧重实验本身的考察。

1. 期末考试临近，你对占成绩60%的闭卷考试有什么预期？

加油吧好好复习。