

**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 | 许健 |
| 学号 | 1183710113 |
| 班号 | 1837101 |
| 电子邮件 | 941197279@qq.com |
| 手机号码 | 18945062342 |

**目录**

[1 实验目标概述 1](#_Toc41744170)

[2 实验环境配置 1](#_Toc41744171)

[3 实验过程 1](#_Toc41744172)

[3.1 Error and Exception Handling 1](#_Toc41744173)

[3.1.1 处理输入文本中的三类错误 1](#_Toc41744174)

[3.1.1.1 存在不符合语法规则的语句 2](#_Toc41744175)

[3.1.1.2 存在标签完全一样的元素 2](#_Toc41744176)

[3.1.1.3 各元素之间的依赖关系不正确 3](#_Toc41744177)

[3.1.2 处理客户端操作时产生的异常 3](#_Toc41744178)

[3.2 Assertion and Defensive Programming 6](#_Toc41744179)

[3.2.1 checkRep()检查rep invariants 6](#_Toc41744180)

[3.2.1.1 Location 6](#_Toc41744181)

[3.2.1.2 MultipleLocation 6](#_Toc41744182)

[3.2.1.3 TwoLocation 7](#_Toc41744183)

[3.2.1.4 MultipleSortedResource 7](#_Toc41744184)

[3.2.1.5 Timeslot： 8](#_Toc41744185)

[3.2.1.6 PresetMultipleTimeslot 8](#_Toc41744186)

[3.2.1.7 CommonPlanningEntry 8](#_Toc41744187)

[3.2.1.8 PlanningEntryCollection 9](#_Toc41744188)

[3.2.1.9 Plane 9](#_Toc41744189)

[3.2.1.10 Carriage 9](#_Toc41744190)

[3.2.1.11 Material 10](#_Toc41744191)

[3.2.1.12 FlightEntry 10](#_Toc41744192)

[3.2.1.13 TrainEntry 11](#_Toc41744193)

[3.2.1.14 ActivityEntry 11](#_Toc41744194)

[3.2.2 Assertion/异常机制来保障pre-/post-condition 11](#_Toc41744195)

[3.2.3 你的代码的防御式策略概述 12](#_Toc41744196)

[3.3 Logging 12](#_Toc41744197)

[3.3.1 异常处理的日志功能 12](#_Toc41744198)

[3.3.2 应用层操作的日志功能 13](#_Toc41744199)

[3.3.3 日志查询功能 13](#_Toc41744200)

[3.4 Testing for Robustness and Correctness 14](#_Toc41744201)

[3.4.1 Testing strategy 14](#_Toc41744202)

[3.4.2 测试用例设计 15](#_Toc41744203)

[3.4.3 测试运行结果与EclEmma覆盖度报告 16](#_Toc41744204)

[3.5 SpotBugs tool 17](#_Toc41744205)

[3.6 Debugging 17](#_Toc41744206)

[3.6.1 EventManager程序 17](#_Toc41744207)

[3.6.2 LowestPrice程序 18](#_Toc41744208)

[3.6.3 FlightClient/Flight/Plane程序 19](#_Toc41744209)

[4 实验进度记录 21](#_Toc41744210)

[5 实验过程中遇到的困难与解决途径 21](#_Toc41744211)

[6 实验过程中收获的经验、教训、感想 21](#_Toc41744212)

[6.1 实验过程中收获的经验和教训 21](#_Toc41744213)

[6.2 针对以下方面的感受 21](#_Toc41744214)

# 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的软件市场中搜索安装spotbugs

Lab4仓库的URL: https://github.com/ComputerScienceHIT/Lab4-1183710113

# 实验过程

## Error and Exception Handling

### 处理输入文本中的三类错误

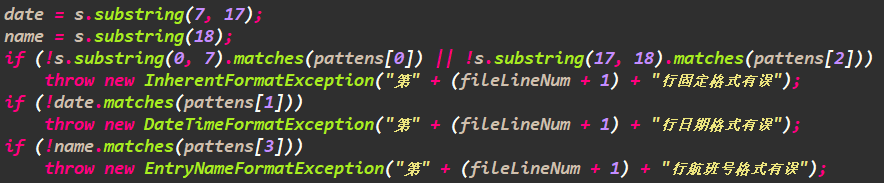
与Lab3中相比，readFileCreatFlightSchedule()只保留了添加读取的计划项和异常处理部分。readFileCreatFlightSchedule()中的解析文件内容并判断是否满足各种依赖关系的部分被抽取出成为了一个单独的方法entries()，当不满足条件时只需要抛出相应的异常信息即可。

#### 存在不符合语法规则的语句

|  |  |
| --- | --- |
| 异常类 | 意义 |
| InherentFormatException | 文件中一个完整的计划项的固有的格式如“Flight:”等字段和大括号的异常 |
| DateTimeFormatException | 日期时间格式异常 |
| EntryNameFormatException | 航班号格式异常 |
| AirportFormatException | 机场的名字含有不合法字符的异常 |
| PlaneCodeFormatException | 飞机编号格式异常 |
| PlaneAgeFormatException | 机龄格式异常 |
| PlaneSeatsFormatException | 飞机座位格式异常 |
| PlaneTypeFormatException | 飞机型号格式异常 |

抛出与处理：

利用正则表达式解析读入的字符串，如果匹配失败则抛出相应的异常。



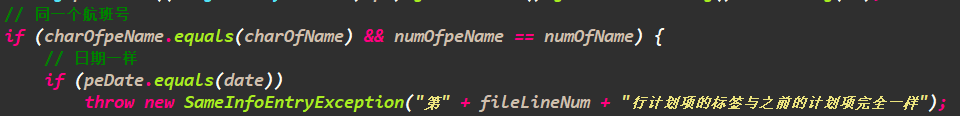
抛出后不在方法内部处理，抛给上级方法，在上级方法中统一处理。

#### 存在标签完全一样的元素

异常类：SameInfoEntryException

记录从文件中读取到的所有计划项，每读入一个新的计划项就遍历之前所有的计划项，找出相同航班号的计划项，然后判断航班日期，如果一样则抛出该异常。

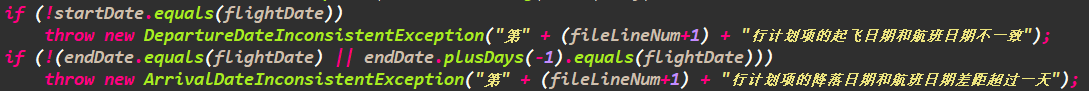
与3.1.1.1相同，不在方法内部处理，抛给上级方法，在上級方法中统一处理。



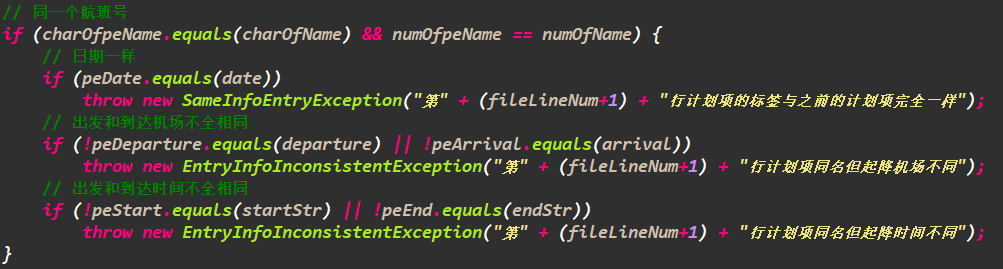
#### 各元素之间的依赖关系不正确

|  |  |
| --- | --- |
| 异常类 | 意义 |
| DepartureDateInconsistentException | 起飞日期和航班日期不一致异常 |
| ArrivalDateInconsistentException | 降落日期和航班日期的差距大于一天异常 |
| EntryInfoInconsistentException | 具有不同日期相同航班号的航班的起降机场和时间有差异的异常 |
| PlaneInfoInconsistentException | 相同编号的飞机信息不一致异常 |

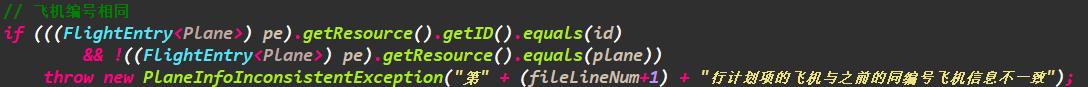
判断新计划项的日期是否满足依赖：



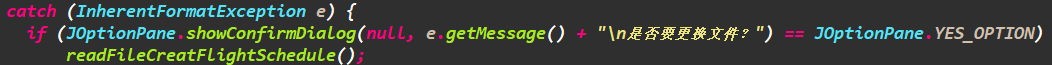
判断与新计划项相同航班号的起降具体时间和起降机场信息：



判断相同编号的飞机信息是否完全相同：



异常处理：在readFileCreatFlightSchedule()方法中调用entries()方法，如果捕获到异常则按如下方式处理，让用户选择是否更换文件，如果是，则再次调用readFileCreatFlightSchedule()方法



### 处理客户端操作时产生的异常

定义几种新的异常：

CannotCancelledException：当前状态不可取消异常

EntryUseResourceOrLocationException：删除在被使用的资源或位置异常

ResourceOrLocationConflictException：资源或位置冲突异常

(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

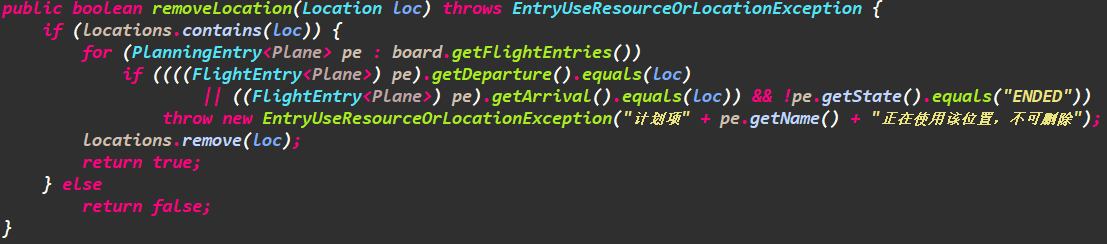
三个App的实现方式相同，以FlightScheduleApp为例:

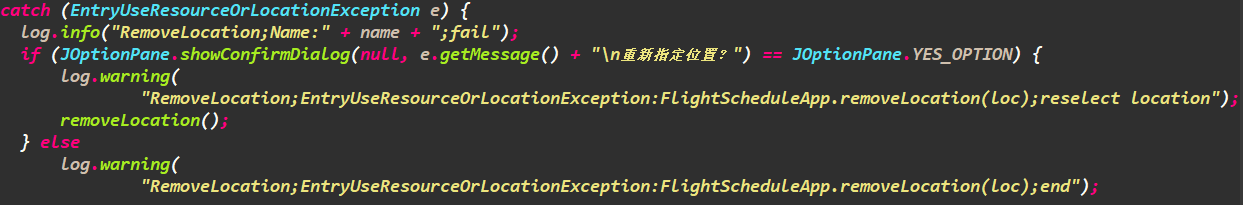
异常抛出：在执行planes.remove(plane)从资源列表中删除指定资源前，先遍历计划项寻找使用该资源且状态不为“ENDED”的计划项，如果有，抛出删除在被使用的资源或位置异常EntryUseResourceOrLocationException；

异常处理：提示错误信息并让用户选择是否重新指定资源，如果是，则再次调用removePlane()方法。

(2) 在删除某位置的时候，如果有尚未结束的计划项正在该位置执行

三个App的实现方式相同，且与删除资源的实现类似。



处理：  


(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的实现方式相同，以FlightScheduleApp为例:

异常抛出：PlanningEntry.cancel()方法的返回值为false时抛出计划项当前状态不可取消异常CannotCancelledException。

异常处理：提示用户当前状态不可取消，并让用户选择是否重新选择取消的计划项。

(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”

FlightScheduleApp和TrainScheduleApp有可能发生这个异常，以FlightScheduleApp为例。

抛出异常：先按照指定的资源进行资源分配，然后调用checkResourceExclusiveConflict()对分配后的计划项列表检测资源冲突，如果返回值为true ，则存在资源冲突，抛出资源或位置冲突异常ResourceOrLocationConflictException。

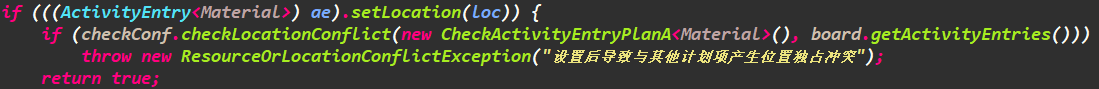
处理异常：弹出提示信息并让用户重新输入待分配的资源。

(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ActivityScheduleApp可能发生这个异常。

① 在setLocation()中：

抛出异常：与(4)中资源冲突异常类似，先按照指定位置进行设置，然后调用checkLocationConflict()方法检测冲突，如果返回true，说明有冲突，抛出资源或位置冲突异常ResourceOrLocationConflictException。

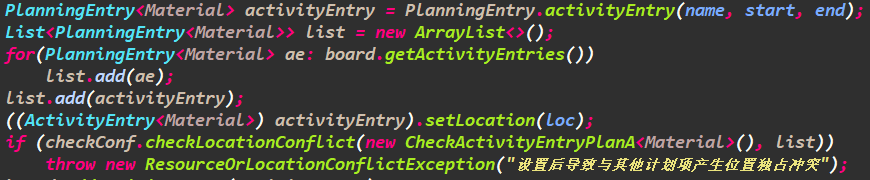


处理异常：把位置设为原来的位置，然后提示用户错误信息，并让用户选择是否重新设置位置，如果是则再次调用setLocation()方法。



② 在addActivity中：

抛出异常：和setLocation()中相同，将设置位置的部分作为死循环，当设置成功的时候break跳出循环即可结束。



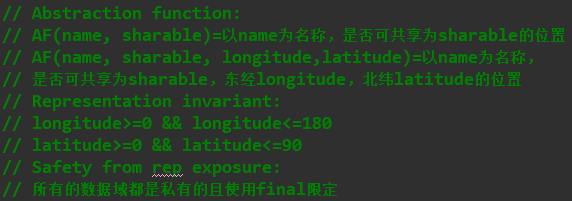
处理异常：捕获到异常时给用户提示错误并让用户重新输入地点，继续循环。

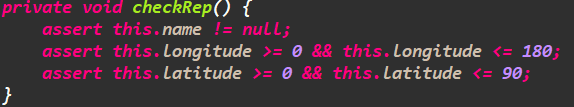


## Assertion and Defensive Programming

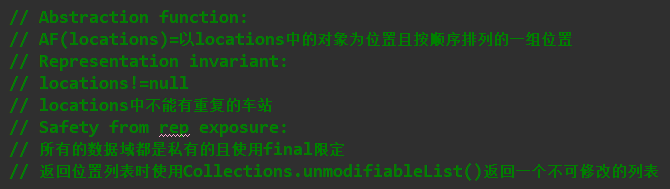
### checkRep()检查rep invariants

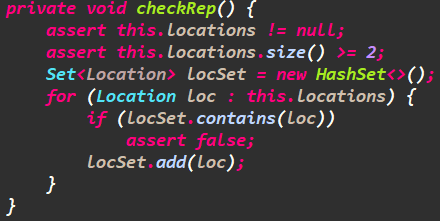
#### Location



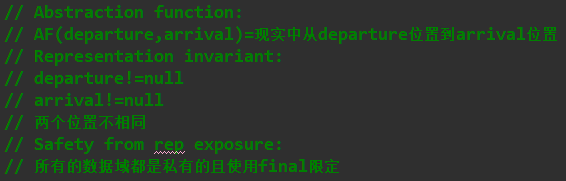


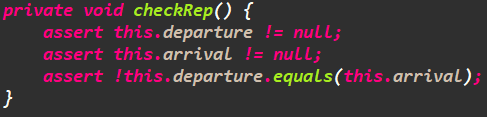
#### MultipleLocation



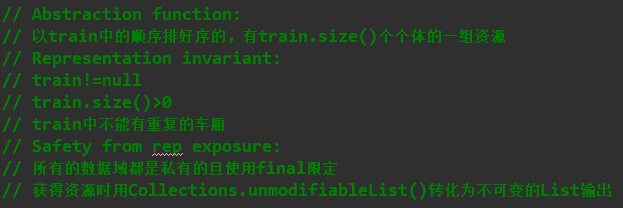


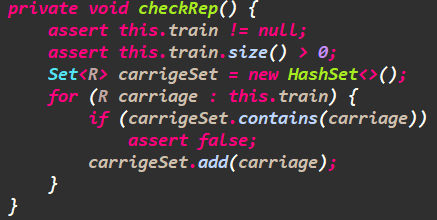
#### TwoLocation



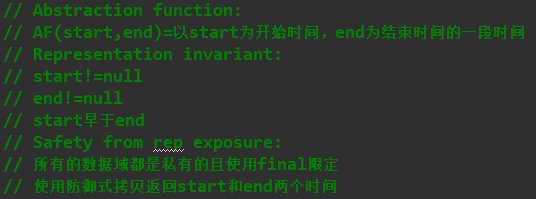


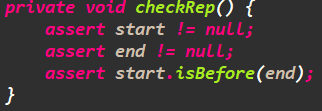
#### MultipleSortedResource



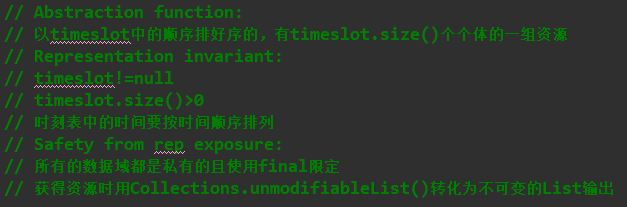


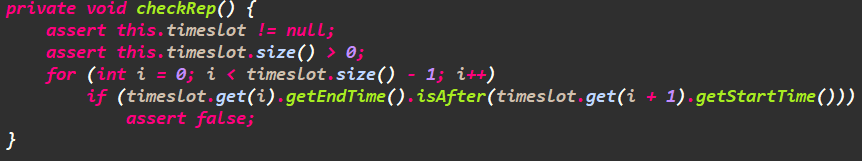
#### Timeslot：



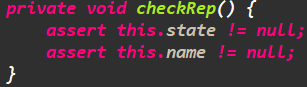


#### PresetMultipleTimeslot

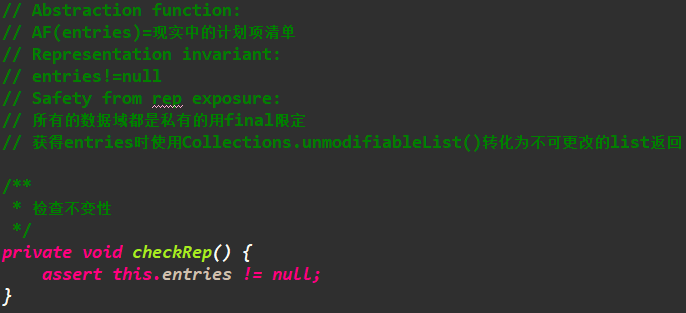


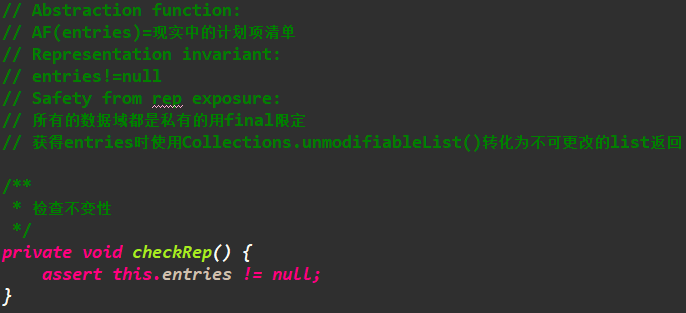


#### CommonPlanningEntry

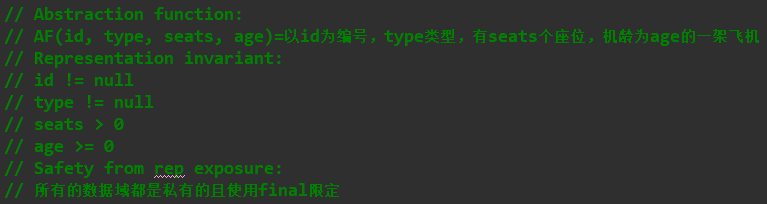


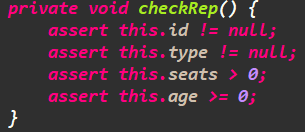
#### PlanningEntryCollection



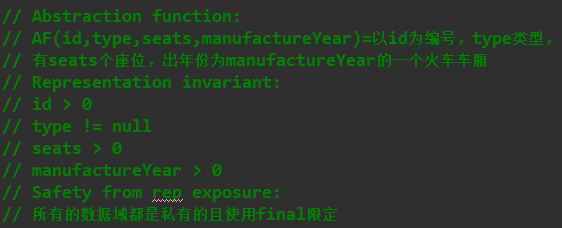


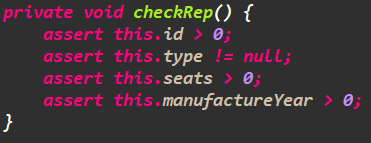
#### Plane



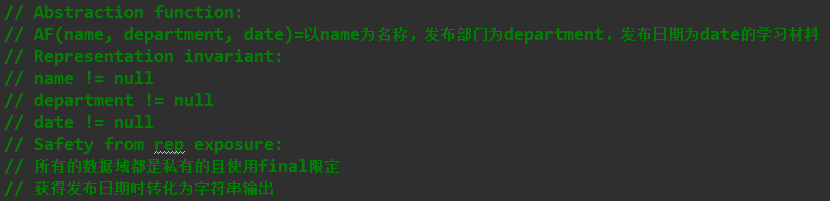


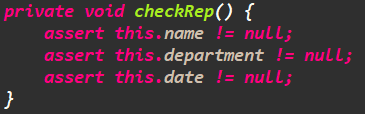
#### Carriage



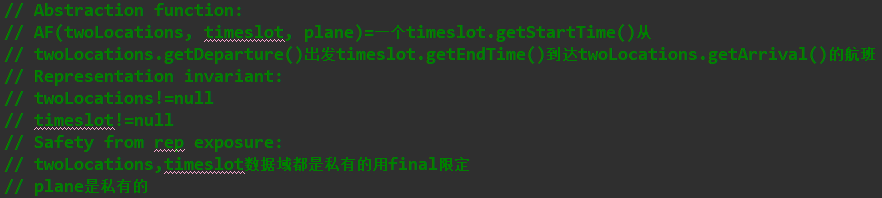


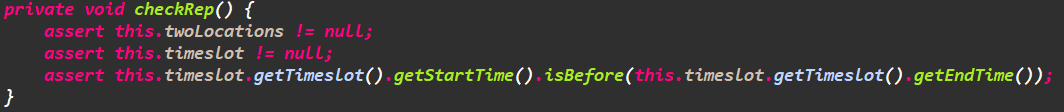
#### Material



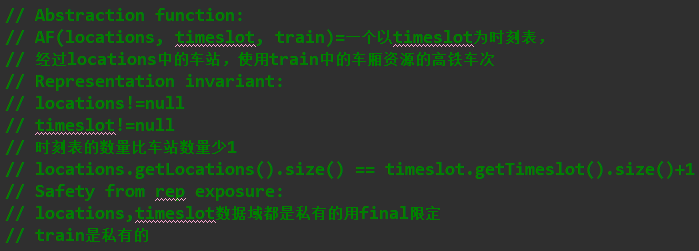


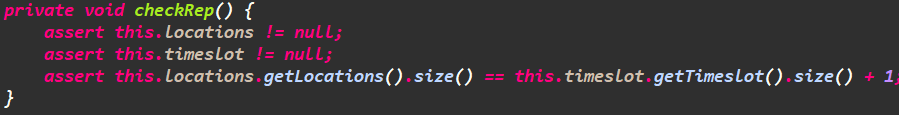
#### FlightEntry



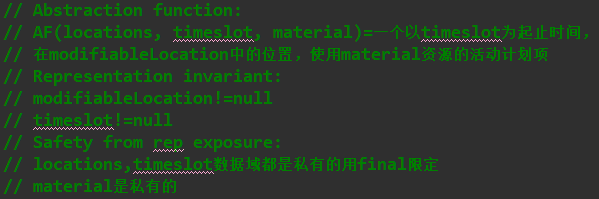


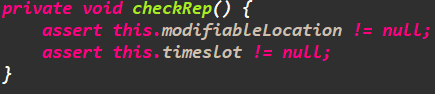
#### TrainEntry





#### ActivityEntry





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

在每个方法的开始添加检查参数的语句，保障pre-condition，当不满足前置条件时抛出参数非法异常（IllegalArgumentException），并给出相应的错误提示信息。

在每个方法返回前通过assert语句保证返回值满足post-condition。

### 你的代码的防御式策略概述

首先是在客户端中，尽量做到让传给ADT的参数都是合法的，即在客户端中先对传入的参数做合法性检测，当满足ADT方法的前置条件时再传给相应的ADT的方法，不满足时则抛出异常并处理异常或直接结束本次操作。

为了尽可能的让程序做到fail fast，在各ADT的方法中利用异常机制保证传入的参数符合前置条件，否则直接抛出异常，使得传入的参数为垃圾时直接fail，在方法返回前利用assertion机制检查后置条件，以保证代码的正确性。

## Logging

使用JDK自带库java.util.logging.\*在客户端实现日志功能。

在App中增加一个Logger，FlightScheduleApp中如下：



在构造函数中添加以下语句，以对Logger进行配置。

先创建log日志文件，由于每次启动客户端应该创建新的日志文件，于是覆盖原来的文件。



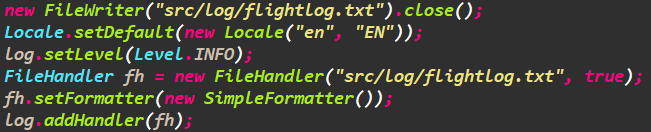
设置日志的输出语言为英语。



设置日志的最低级别为INFO。



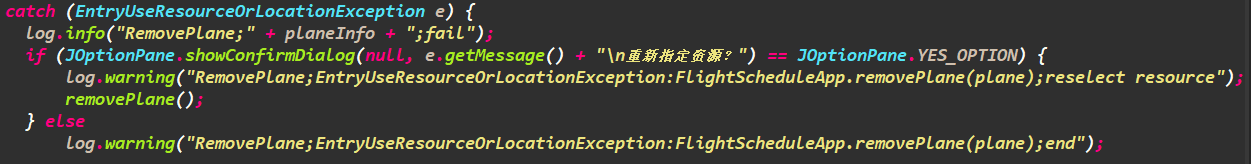
设置日志的输出文件以及日志格式。



到此，日志已经设置完成，可以记录日志了。

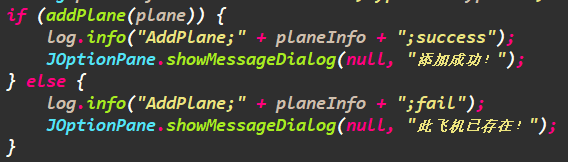
### 异常处理的日志功能

在处理异常过程中，设置异常级别为warning的异常信息。根据异常的不同处理方式及结果来设置不同的异常信息。以下为FlightScheduleApp在removePlane()中的异常处理日志记录。



### 应用层操作的日志功能

在应用层的操作中，根据不同的操作和操作对象以及操作结果在日志中添加不同的操作信息。以下为FlightScheduleApp在addPlane()中的操作日志记录。

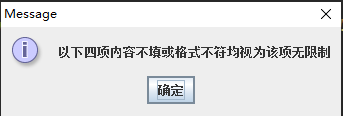


### 日志查询功能

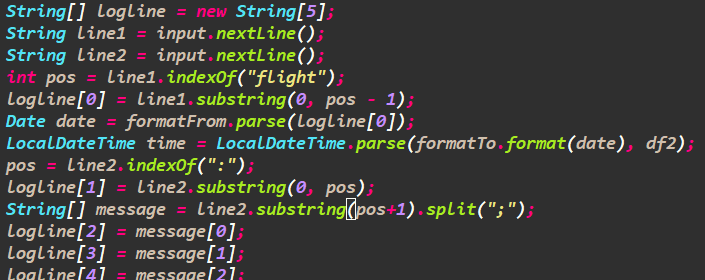
日志文件内容：



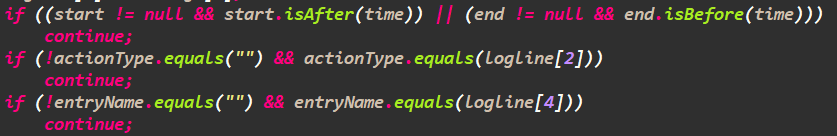
在应用中添加日志查询按钮，会弹出一个消息提示框，提示操作，然后可以选择筛选日志的时间段，可选择按计划项进行筛选，也可以根据操作类型筛选日志。



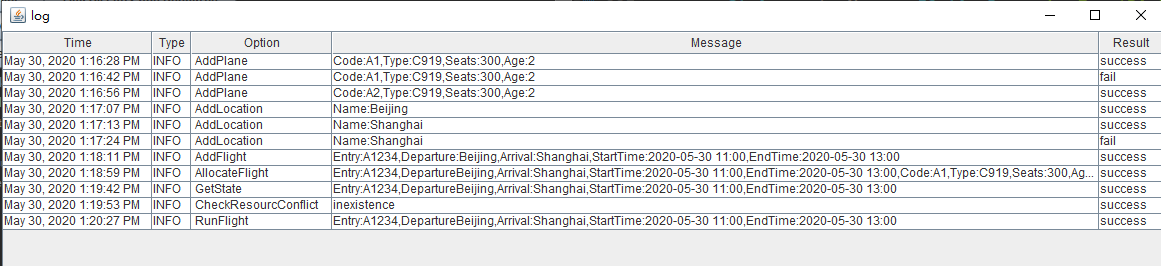
利用日志的格式读入日志文件，然后解析日志信息。



通过比对用户输入的过滤信息，确定该条日志是否被过滤。



最后将符合条件的内容通过委派给Board的visualize方法以JTable的方式显示。效果如下：

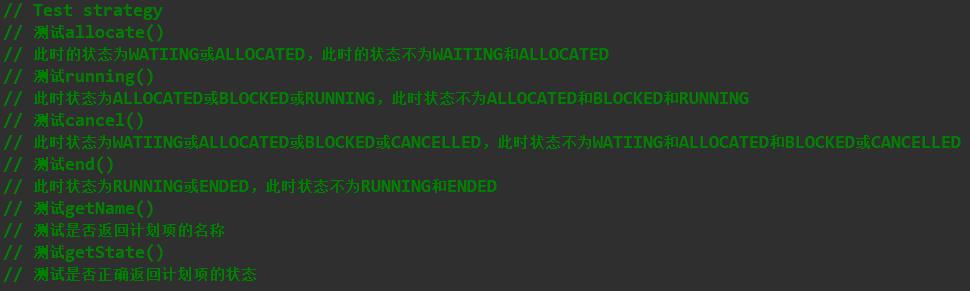


## Testing for Robustness and Correctness

### Testing strategy

使用等价类和边界值的测试思想，根据各ADT各方法的spec为其构造测试用例并编写测试。

下图中为PlanningEntry中实例化方法的测试策略。



### 测试用例设计

根据3.1中所处理的各种异常，为各种Exception构造不同的测试用例，放在test/flight/FlightTest文件夹下

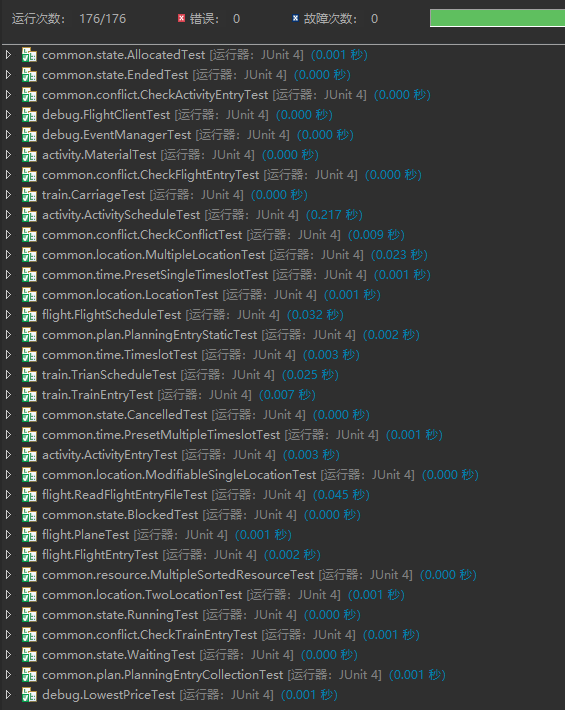


图中为飞机机场格式异常对应的测试文件，由于设定的格式是word，所以在名称里加入非法字符“\_”构造出非法文件。



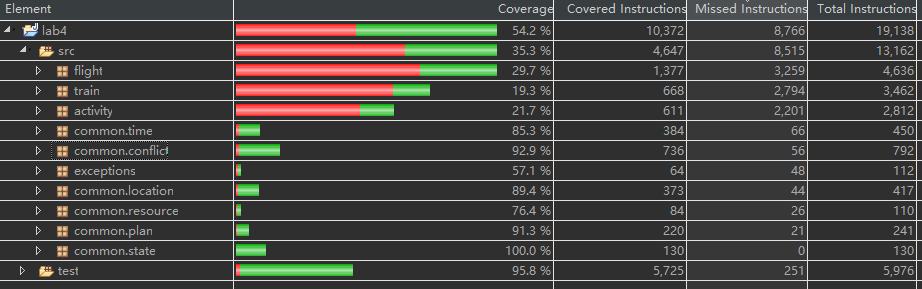
### 测试运行结果与EclEmma覆盖度报告

测试结果：

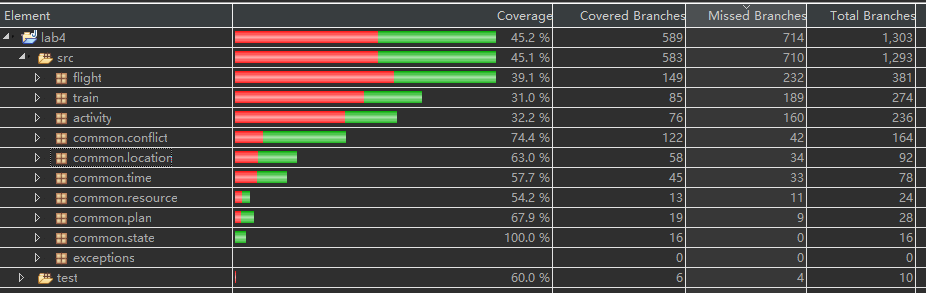


EclEmma覆盖度报告：（导出的html格式报告位于doc文件夹下）

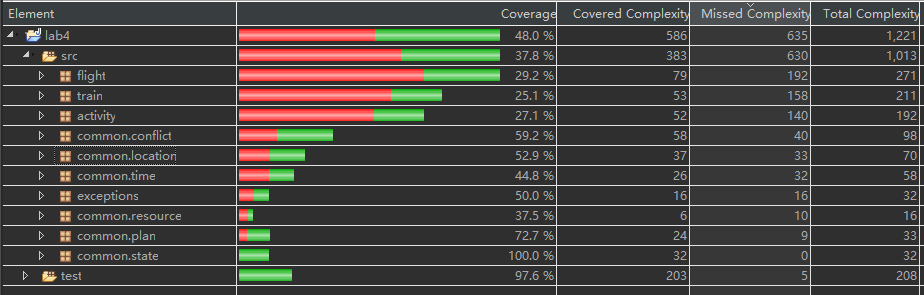
语句覆盖度：



分支覆盖度：



路径覆盖度：



## SpotBugs tool

使用SpotBugs工具发现了一个ADT重写了equals()方法但没有重写hashCode()方法。

## Debugging

### EventManager程序

理解待调试程序的代码思想：

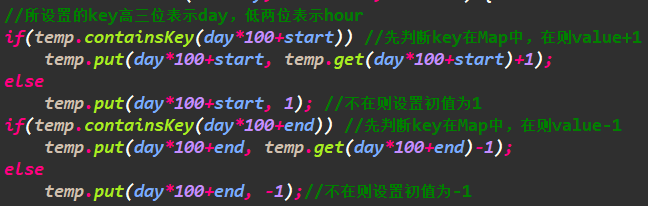
这个ADT试图维护一个保存着后项与前项同时进行事件数的差的TreeMap，然后求最大并发事件数就是对Map中的value求和，找所有value的最大值。

发现并定位错误的过程：

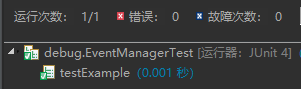
阅读代码，发现第一个错误在最开始，要先判断Map中contains对应的key才能get(key)。第二个错误在于没有使用day参数，这样就导致所有被添加的事件都是在同一天，并没有按照日期做区分。

你如何修正错误：

更正见注释



修复之后的测试结果：



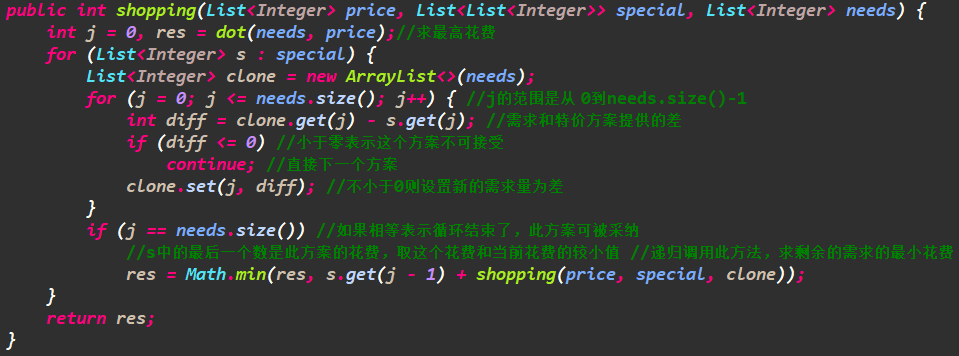
### LowestPrice程序

理解待调试程序的代码思想：

先计算在不使用优惠方案时的花费，也就是最大的最小代价，然后遍历所有的优惠方案，判断方案的物品数是否不超过所需要的数量，然后在满足条件的方案下递归地求剩余所需的最小代价。

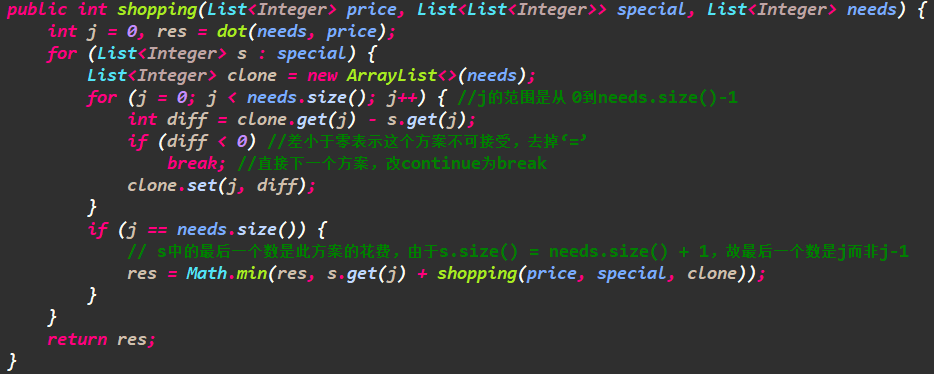
发现并定位错误的过程：

阅读代码，根据代码的思想理解实现。



你如何修正错误：

更正见注释



修复之后的测试结果：



### FlightClient/Flight/Plane程序

理解待调试程序的代码思想：

该程序试图枚举所有的航班，然后为每个航班分配资源，判断是否所有的飞机都不能满足不冲突的条件，如果是，则返回false。

发现并定位错误的过程：

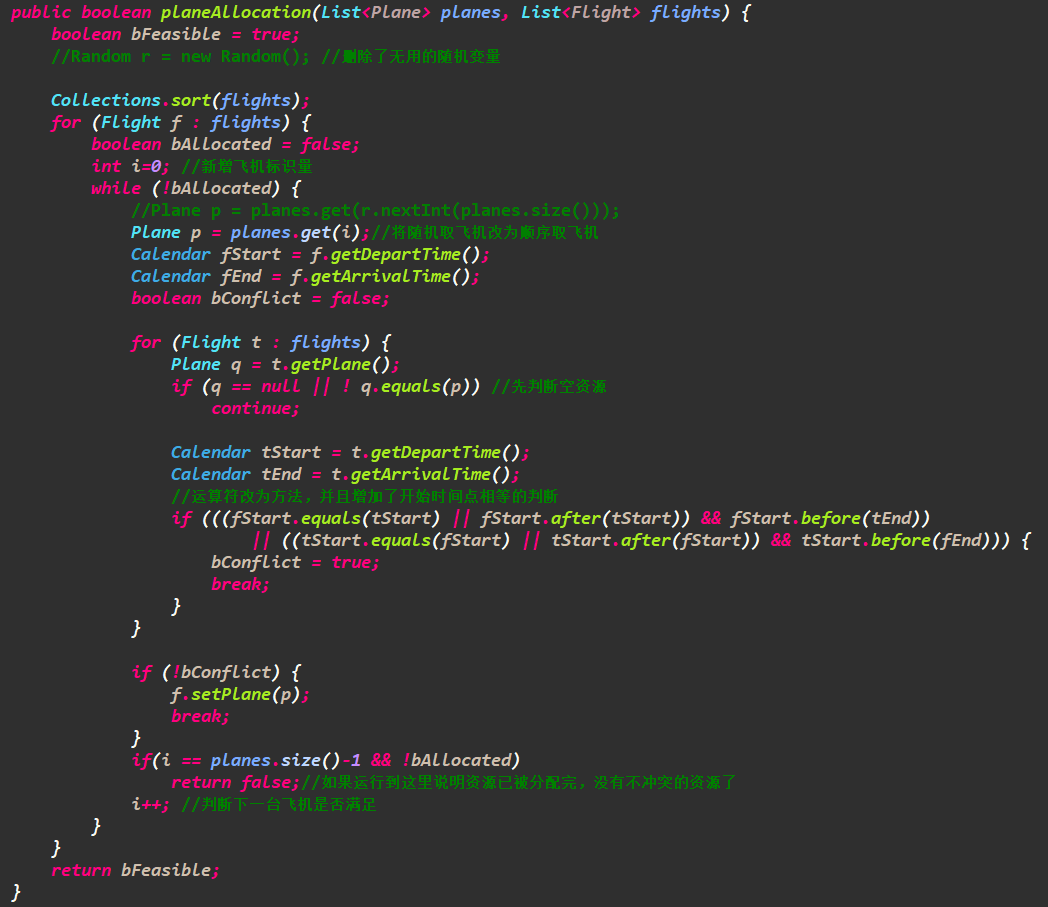
程序试图对航班排序，但Flight并没有实现Comparable或Comprator接口。程序试图比较Calendar时间对象，但是使用的是运算符比较，并且在判断时间是否有交集的时候没有判断开始时间相等的情况。程序试图遍历所有的飞机确保分配，但是用的是random这种无法判断是否所有的飞机都被遍历过的方式去获得飞机。

Plane的equals()方法判断飞机编号相等时使用等号。

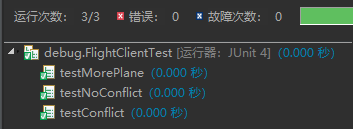
你如何正错误：

Plane.equals()中的等号判断改为equals()方法判断。

FlightClient更正见注释



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 |
| 2020.5.21 | 15:30-17:30 | 完成3.1文件读取异常的改造 | 按时完成 |
| 2020.5.22 | 18:00-21:00 | 3.2防御式改造 | 未完成 |
| 2020.5.23 | 13:00-18:00 | 3.2防御式改造 | 按时完成 |
| 2020.5.24 | 19:00-23:00 | 3.3日志功能 | 按时完成 |
| 2020.5.26 | 19:00-22:00 | 3.4测试 | 按时完成 |
| 2020.5.26 | 22:00-22:30 | 3.5 | 按时完成 |
| 2020.5.28 | 19:00-22:00 | 完场3.6对给定文件的debug | 按时完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 在读取log文件时，时间无法正常解析。 | 在网上查阅资料，最终该问题仍然没有得到解答，于是更换了解析时间的实现方式。 |
|  |  |
|  |  |

# 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？

正确性更关注程序在不同的正确输入下是否能得到正确的结果，这是对程序的基本要求，而健壮性则需要考虑不同的各种各样的输入，甚至可能是非法的输入，在这种情况下要让程序表现良好，不至于崩溃结束。

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？）

对于一个规模庞大或者安全性要求很高的程序来说，再大的代价要需要实现，这个例子像载人航天器。

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？

前者对用户负责，后者不对用户负责。

如果程序是给自己用的，并不会给别人用，就会更多的遵循后者，而如果ADT要发布出去，给别人使用，就要更多的考虑前者。

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？

给自己用的大可不必，给别人用的话我觉得是必须的。

安全和性能不可兼得，要保证一方就损失另一方。

根据程序的目的不同选择不同的倾向吧。

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？

根据spec编写测试用例，然后根据测试结果的反馈判断出错的类型，然后通过debug确定错误的位置。

是。

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？

代码能够覆盖所有的语句并不表明能够覆盖所有的输入情况，有些边界条件可能并没有被覆盖到。

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。

还算有趣吧，能够看到不同的实现方式。

没有注释，没有文档不行，尤其是rep表示什么需要知道，比如EventManage这个不知道Map表示啥就不太容易被想到原本的实现逻辑。

1. 关于本实验的工作量、难度、deadline。

工作量和Lab3的实现都很大关系，Lab3做的好的话这个实验的工作量并不大。难度一般，deadline时间充足。

1. 到目前为止你对《软件构造》课程的评价和建议。

还好。

1. 期末考试临近，你对占成绩60%的闭卷考试有什么预期？

希望题量不是太大，希望线下考试。