

**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 | 张瑞豪 |
| 学号 | 1180800811 |
| 班号 | 1803002 |
| 电子邮件 | m17779349381@163.com |
| 手机号码 | 17779349381 |

**目录**

[1 实验目标概述 1](#_Toc42329498)

[2 实验环境配置 1](#_Toc42329499)

[3 实验过程 1](#_Toc42329500)

[3.1 Error and Exception Handling 2](#_Toc42329501)

[3.1.1 处理输入文本中的三类错误 2](#_Toc42329502)

[3.1.1.1 文件中的格式错误 2](#_Toc42329503)

[3.1.1.2 输入文件中存在标签完全一样的元素 5](#_Toc42329504)

[3.1.1.3 输入文件中各元素之间的依赖关系不正确 6](#_Toc42329505)

[3.1.2 处理客户端操作时产生的异常 7](#_Toc42329506)

[3.1.2.1 自定义的异常类 7](#_Toc42329507)

[3.2 Assertion and Defensive Programming 7](#_Toc42329508)

[3.2.1 checkRep()检查rep invariants 8](#_Toc42329509)

[3.2.2 Assertion/异常机制来保障pre-/post-condition 10](#_Toc42329510)

[3.2.3 你的代码的防御式策略概述 10](#_Toc42329511)

[3.3 Logging 11](#_Toc42329512)

[3.3.1 思路描述 11](#_Toc42329513)

[3.3.2 logRecord类 11](#_Toc42329514)

[3.3.3 LogKeeper类 11](#_Toc42329515)

[3.3.4 日志格式 12](#_Toc42329516)

[3.3.5 异常处理的日志功能 13](#_Toc42329517)

[3.3.6 应用层操作的日志功能 13](#_Toc42329518)

[3.3.7 日志查询功能 14](#_Toc42329519)

[3.4 Testing for Robustness and Correctness 14](#_Toc42329520)

[3.4.1 Testing strategy 14](#_Toc42329521)

[3.4.2 测试用例设计 14](#_Toc42329522)

[3.4.3 测试运行结果与EclEmma覆盖度报告 16](#_Toc42329523)

[3.5 SpotBugs tool 16](#_Toc42329524)

[3.6 Debugging 17](#_Toc42329525)

[3.6.1 EventManager程序 17](#_Toc42329526)

[3.6.1.1 理解待调试程序的代码思想 17](#_Toc42329527)

[3.6.1.2 发现并定位错误的过程 18](#_Toc42329528)

[3.6.1.3 你如何修正错误 18](#_Toc42329529)

[3.6.1.4 修复之后的测试结果 18](#_Toc42329530)

[3.6.2 LowestPrice程序 19](#_Toc42329531)

[3.6.2.1 理解待调试程序的代码思想 19](#_Toc42329532)

[3.6.2.2 发现并定位错误的过程 19](#_Toc42329533)

[3.6.2.3 你如何修正错误 19](#_Toc42329534)

[3.6.2.4 修复之后的测试结果 20](#_Toc42329535)

[3.6.3 FlightClient/Flight/Plane程序 20](#_Toc42329536)

[3.6.3.1 理解待调试程序的代码思想 21](#_Toc42329537)

[3.6.3.2 发现并定位错误的过程 21](#_Toc42329538)

[3.6.3.3 你如何修正错误 21](#_Toc42329539)

[3.6.3.4 修复之后的测试结果 22](#_Toc42329540)

[4 实验进度记录 22](#_Toc42329541)

[5 实验过程中遇到的困难与解决途径 22](#_Toc42329542)

[6 实验过程中收获的经验、教训、感想 23](#_Toc42329543)

[6.1 实验过程中收获的经验和教训 23](#_Toc42329544)

[6.2 针对以下方面的感受 23](#_Toc42329545)

# 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

URL: https://github.com/ComputerScienceHIT/Lab4-1180800811.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

### 处理输入文本中的三类错误

处理文本中三类错误的代码结构:

**public** **void** ReadFileCreatePlanningEntry(String file) **throws** Exception

负责读取文件，每次读取十三行信息并且调用getFileFlightEntry(String s )方法来完成航班信息的提取并且创建航班。

**public** FlightEntry getFileFlightEntry(String s )方法，

这个方法负责解析读取的十三行信息，同时也判断是否有各种异常，比如文件信息不符合格式要求，有相同的航班号等等。具体流程是:先调用IsillegalRegular方法判断是否有格式错误，如果没有则读取其中的信息构造一个航班，在调用PanDuan方法来判断构造的航班和原来的航班是否存在依赖关系错误，并且判断是否有完全相同的航班。

**public** **void** IsillegalRegular(String ff )

这个方法判断十三行组成的字符串是否存在格式错误，

**public** **void** PanDuan(FlightEntry flight)

判断新读取的航班和已经读取的航班集是否存在依赖关系错误或者是否具有完全相同的航班信息。

#### 文件中的格式错误

我检查了所有可能出现的错误，检查的思路是：一行一行的读取，判断每一行的格式是否均符合lab3给的格式要求。

思路:由lab3可知，去除空行后，文件中每13行表示一个航班计划项，每一行都要符合lab3给出的格式要求。我使用13个正则表达式进行匹配，第一个正则表达式匹配第一行，如果第一行格式错误，直接抛出异常和提示信息，提醒用户换另一个文件进行读取。如果第一行格式正确，则使用第二个正则表达式匹配第一行和第二行，如果不匹配，那么肯定是第二行格式错误，因为我们已经知道第一行格式正确。依次类推，那么就需要13个正则表达式进行匹配。

为啥匹配的时候不是一次匹配一行呢：因为一次匹配一行，可能会出现这样的情况:13行的每一行的信息都是符合格式要求的，但是每一行的顺序不是正确的，这种情况下每次匹配一行也能匹配到相应的信息，但是这显然不符合格式要求。

所有的错误：

* 第一行:①格式不符合Flight:xxx,xxx

②航班日期格式不符合xxxx-xx-xx

③ 航班号不符合格式要求，不是由两位大写字母和 2-4 位数字构成

* 第二行：①第二行不是‘{’
* 第三行：①格式不符合DepartureAirport:xxx,比如DepartureAirport拼 写错误
* 第四行: ①格式不符合ArrivalAirport:xxx,比如DepartureAirport拼 写错误
* 第五行: ①格式不符合DepatureTime:xxxx,比如DepatureTime 拼写错误

②出发时间格式不符合xxxx-xx-xx yy:yy

* 第六行: ①格式不符合ArrivalTime:xxx,比如ArrivalTime拼写错误

②到达时间格式不符合xxxx-xx-xx yy:yy

* 第七行:①不符合Plane:xxx格式，比如Plane拼写错误

②飞机编号格式不符合要求，不符合第一位为 N 或 B，后面是四位数字

* 第八行:①第八行不是‘{’
* 第九行:①第九行格式不符合Type:xxx，比如Type拼写错误

②机型格式错误，不符合大小写字母或数字构成，不含有空格和其他符号

* 第十行: ①第十行不符合Seats:xxx，比如Seats拼写错误

②座位数不是整数

③座位数不在[50,600]

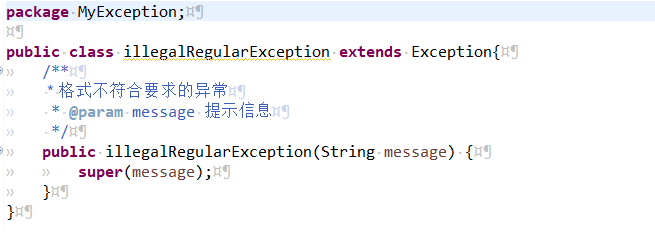
* 第十一行:①第十一行格式不符合Age:xxx，比如Age拼写错误

②机龄不符合要求，比如范围不在[0,30]，小数位数超过1位

* 第十二行：①第十二行不是‘{’
* 第十三行：①第十三行不是‘{’

注释：我判断文件是否合法均是按照以上的标号顺序进行判断。(详细见代码)

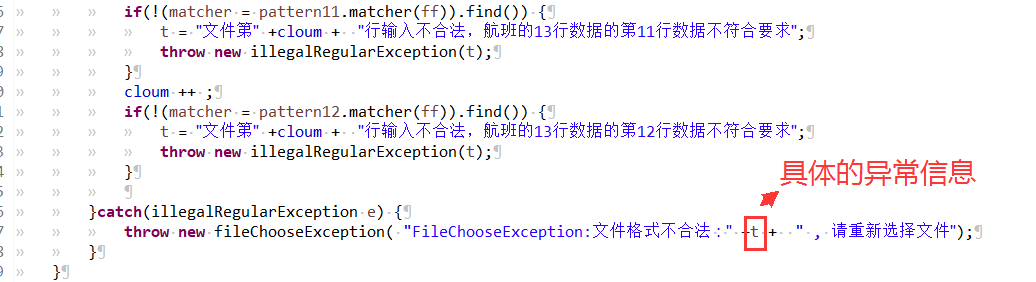
自定义的异常类：illegalRegularException ，FileChooseException

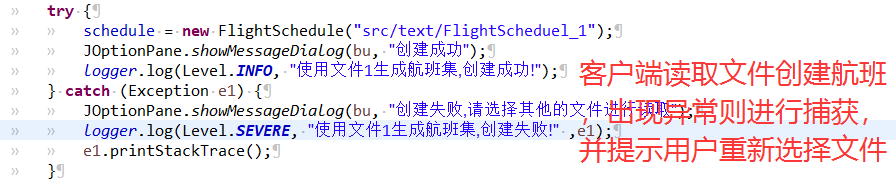


注释：我把所有的和格式相关的异常都统一当做illegalRegularException异常，即格式不合法，因为我检测的格式不合法的异常实在太多了，感觉一一列举的话重复的工作很多。我会在抛出illegalRegularException异常的提示信息里面会区分到底是哪一行的格式不符合。客户端可以直接捕获异常获取异常信息。

具体的异常信息的捕获：

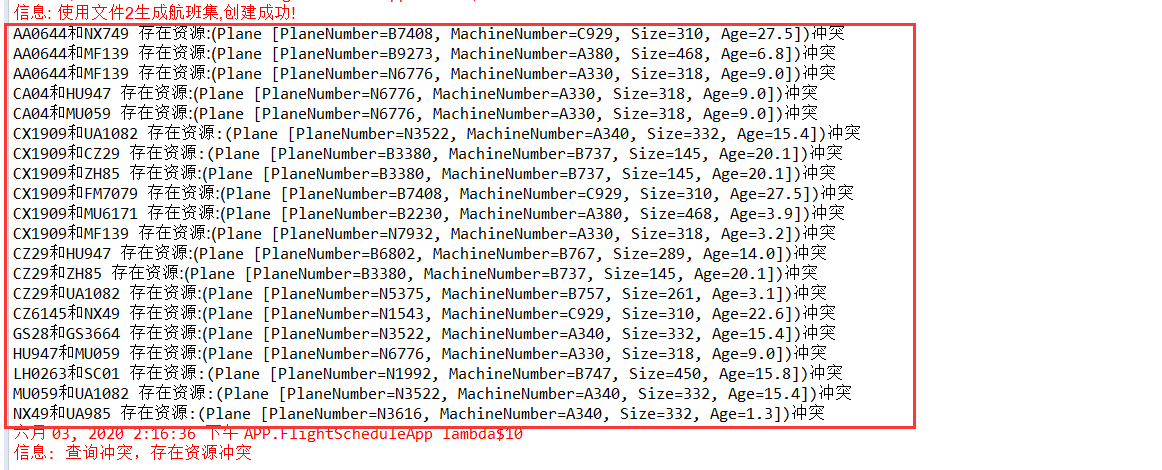
当出现格式错误，直接抛出illegalRegularException异常，捕获异常之后，再抛出FileChooseException异常给客户端代码，客户端代码获取异常之后，根据异常的提示信息来提示给用户，并进行日志记录。



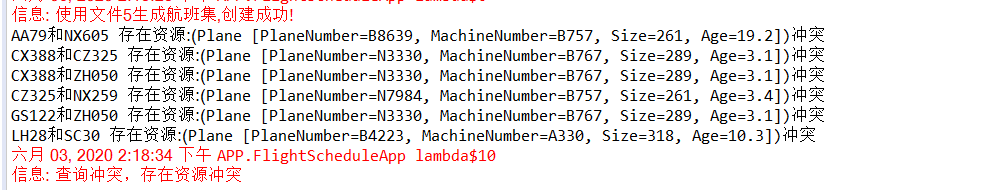


五个测试文件的结果:文件1、3、4的资源依赖关系错误，文件2、5存在资源冲突。









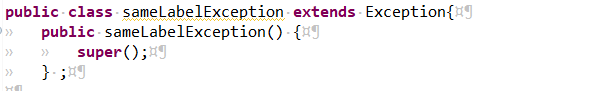
#### 输入文件中存在标签完全一样的元素

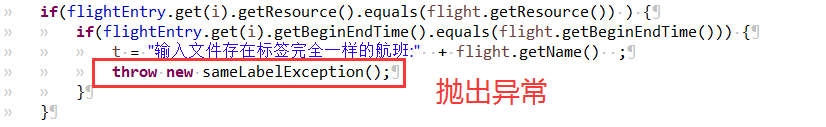
比如:存在多个航班计划项的“日期,航班号”信息完全一样。

思路：读取一个航班项之后，遍历之前已经读取到的航班集，判断是否存在和这个航班项完全相同的航班即可。

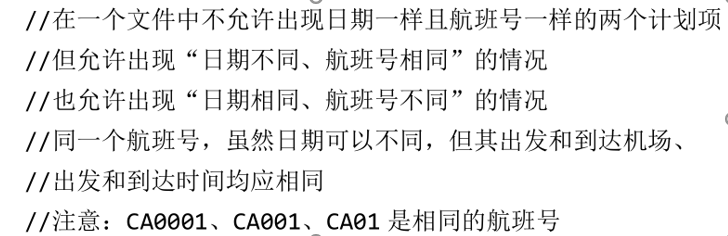
详细见代码

自定义的异常类：sameLabelException



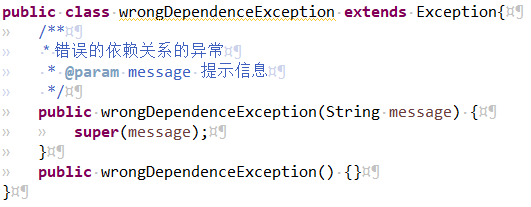


#### 输入文件中各元素之间的依赖关系不正确



思路:每次读取了一个航班项之后，就遍历已经读取的航班项，判断是否存在上述依赖关系不正确。详细见代码

自定义的异常类:wrongDependenceException类

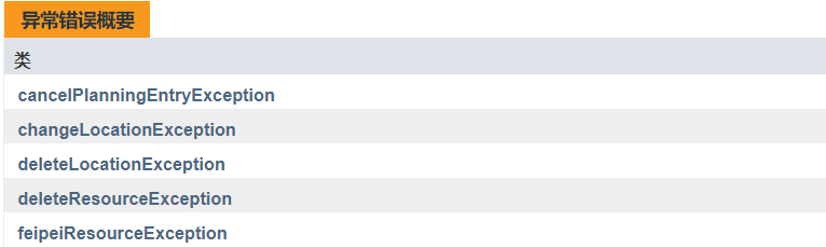


**部分代码示例**

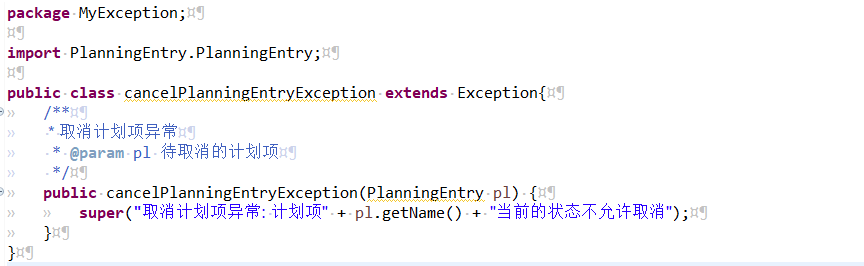


### 处理客户端操作时产生的异常

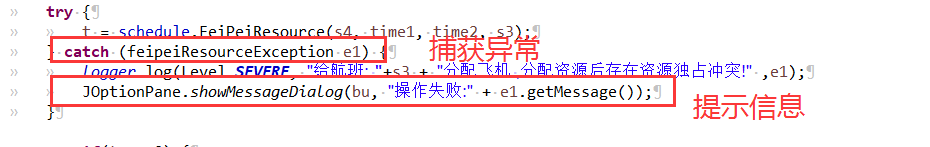
#### 自定义的异常类



以cancelPlanningEntryException异常为例，主要就是继承Exception类就行



具体思路:客户端收到用户指令之后，然后调用相应的方法，如果出现异常，则相应的方法直接抛出相应的异常，客户端直接捕获异常之后就可以返回给用户提示信息。



## Assertion and Defensive Programming

各项Rep , RI , AF在lab3都已经写的很详细了，这里不作过多的阐述，lab4新增加的就是断言和checkRep方法，这里介绍一下。

### checkRep()检查rep invariants

不同的类的checkRep介绍

FlightEntry：

**public** **void** checkRep() {

**assert** loc != **null** ;//位置不为空

**assert** res != **null** ;//资源不为空

**assert** be != **null** ;//时间对不为空

}

TrainEntry:

**public** **void** checkRep() {

**assert** mle != **null** ;//位置不为空

**assert** be != **null** ;//时间对不为空

**assert** msre != **null** ;//资源不为空

Set<String> s = **new** HashSet<>();

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

s.add(msre.getResource().get(i).getNumber());

}

**assert** s.size() == msre.getResource().size() ;//不存在重复的车厢编号

}

CourseEntry:

**public** **void** checkRep() {

**assert** loc != **null** ;//位置不为空

**assert** be != **null** ;//时间对不为空

**assert** res != **null** ;//资源不为空

}

TimeSlot:

**public** **void** checkRep() {

**assert** date1 != **null** ;//起始时间不为空

**assert** date2 != **null** ;//终止时间不为空

**assert** date1.compareTo(date2) < 0 ;//起始时间早于终止时间

}

MultipleLocationEntryImpl**：**

**public** **void** checkRep() {

**assert** locations != **null** ;//位置列表不为空

**assert** locations.size() != 0 ;//位置个数大于0

Set<Location> l = **new** HashSet<>() ;

**for**(Location loc : locations) {

l.add(loc);

}

**assert** l.size() == locations.size() : "多个位置中不能有重复的位置" ;

}

TwoLocationEntryImpl:

**public** **void** checkRep() {

**assert** start != **null** ;//起始位置不为空

**assert** end != **null** ;//终止位置不为空

**assert** !start.equals(end) : "起始位置和终止位置不能相同" ;

}

MultipleSortedResourceEntryImpl：

**public** **void** checkRep() {

**assert** **this**.r != **null** : "资源不能为空";

**assert** **this**.r.size() > 0 ;//资源数量大于0

}

BlockableImpl:

**public** **void** checkRep() {

**assert** timeslots != **null** ; //时间对的列表不为空

**if**(timeslots.size() > 1 ) {

**for**(**int** i = 0 ; i < timeslots.size() - 1 ; i ++ ) {

**assert** timeslots.get(i).getdate2().compareTo(timeslots.get(i+1).getdate1()) < 0 ;//时间对递增

}

}

}

UnBlockableImpl:

**public** **void** checkRep() {

**assert** timeslot != **null** ; //时间对不为空

**assert** timeslot.getdate1().compareTo(timeslot.getdate2()) < 0 ;//时间增序

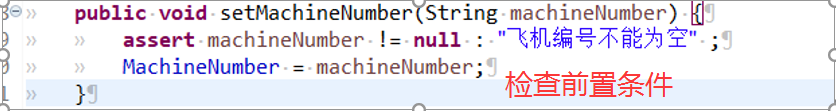
}

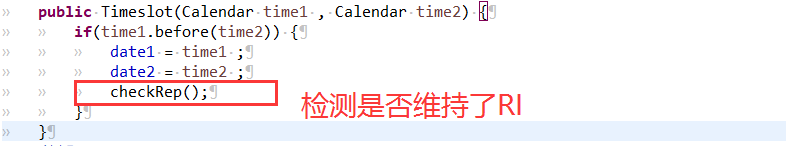
具体的Board类、application类、APP类的checkRep都是依靠基本的上述的checkRep来保证RI，这里不一一阐述

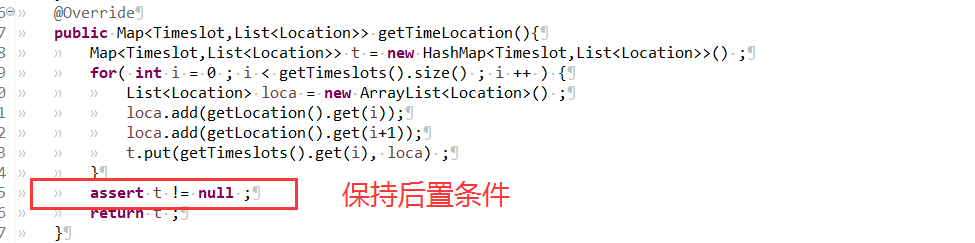
### Assertion/异常机制来保障pre-/post-condition

思路:对于一些前置条件的判断，为了fail fast ，我有的直接用断言检查前置条件是否符合，有的直接抛出异常。对于后置条件则是使用断言。由于很多的getter方法都是委托给其他的类来实现，所以大多数的后置条件的检查都是放在了一些基础的类里面，而基础的类的getter方法都是直接返回类的不变量，而类的不变量基本是由前置条件来保证，因此大多数的断言都是保证前置条件，对于一些不是直接返回类的属性的方法，需要使用断言保证后置条件。同时，对于自己实现的方法，为了保持RI，同时用checkRep方法检测。

具体的详细见代码







### 你的代码的防御式策略概述

1. 首先,类的属性都是使用private关键字来描述。这样保证了外部无法访问类的属性。
2. 对于可变数据类型，当使用getter方法返回时，要进行防御式拷贝。典型的就是Date变量，还有就是返回List列表时候需要进行防御式拷贝
3. 同时，对于setter方法来说，如果参数也是可变的，同样不能直接用“=”来给属性赋值，需要创建一个新的变量，用这个新的变量来给类的属性赋值。
4. 对于客户端传进来的参数，通过断言和抛出异常来保证不变量，在API内，同时使用checkRep来检查不变量的正确性。断言也保证后置条件的正确性。

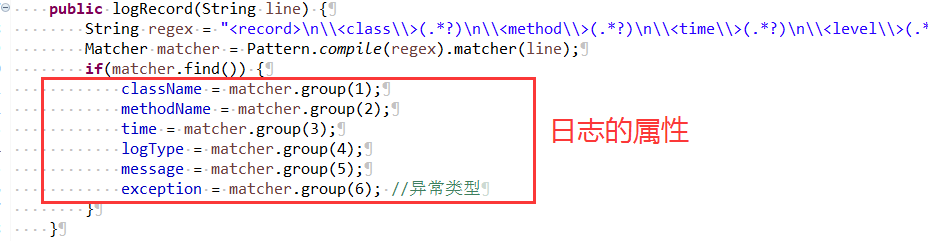
## Logging

### 思路描述

1. 自己声明一个日志类对象logRecord，每一个日志类对象表示一条日志信息，日志类的属性有:异常发生的类名、方法名、时间、具体信息、日志等级、异常类型。从文件中读取的每一条日志用这个日志对象表示。
2. 然后声明一个日志管理类logKeeper，管理所有的日志信息，比如可以按照时间或者异常类型等等进行日志的查询。我们把App的日志信息输出到一个文件中，日志管理类logKeeper就可以读取文件信息创建多个日志类对象当客户端需要查询日志时，可以按照自己的格式来反馈给用户。

### logRecord类

功能:构造器是读取一个字符串，通过对字符串解析构造一个日志类对象。



### LogKeeper类

功能:读取日志文件来构造多条日志的List，并且按照时间对日志进行排序。



LogKeeper类实现了对日志的查询，可以根据时间范围，或者日志类型进行日志筛选。



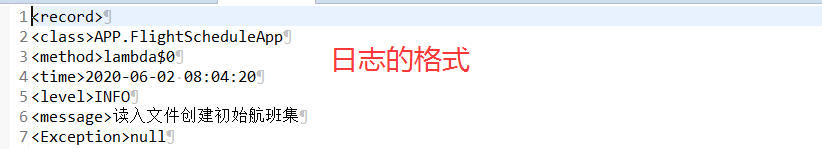


### 日志格式

记录的日志有:异常和用户操作的信息。对于异常来说，当抛出异常时，不仅提示用户相应的异常信息，同时通过日志记录相应的信息。每一条用户选择的指令都需要通过日志记录。

在APP类，定义一个日志对象，对日志记录的信息和格式进行设置。





注释：日志等级我分为三种：INFO、WARNING、SEVERE

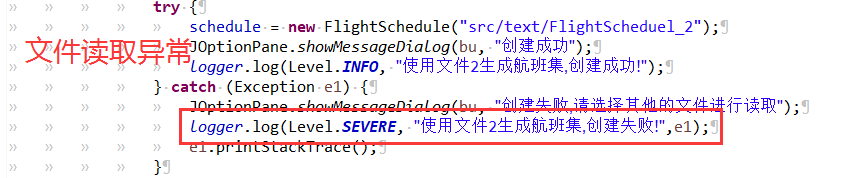
INFO:用户的操作成功步骤

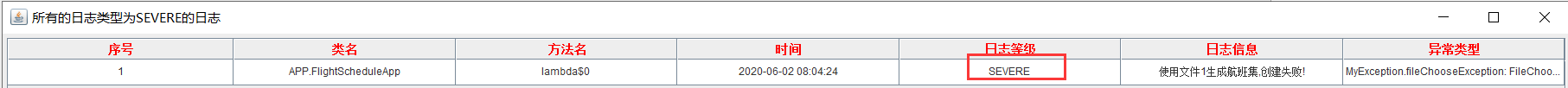
WARNING：用户操作失败的步骤，比如用户输入的时间格式错误等等。这种步骤不涉及ADT抛出的异常。

SEVERE：记录ADT内部抛出的异常，不如删除位置异常、删除资源异常、文件读取格式错误等等。

### 异常处理的日志功能

当出现异常时，需要记录异常的出现的类名、方法名、时间、提示信息、异常类、日志等级。





### 应用层操作的日志功能

对于用户的所有操作，均记录在日志内。



### 日志查询功能

我设定了两种日志查询的功能，一个是根据时间范围进行日志查询，另一个是根据日志类型来进行日志查询。







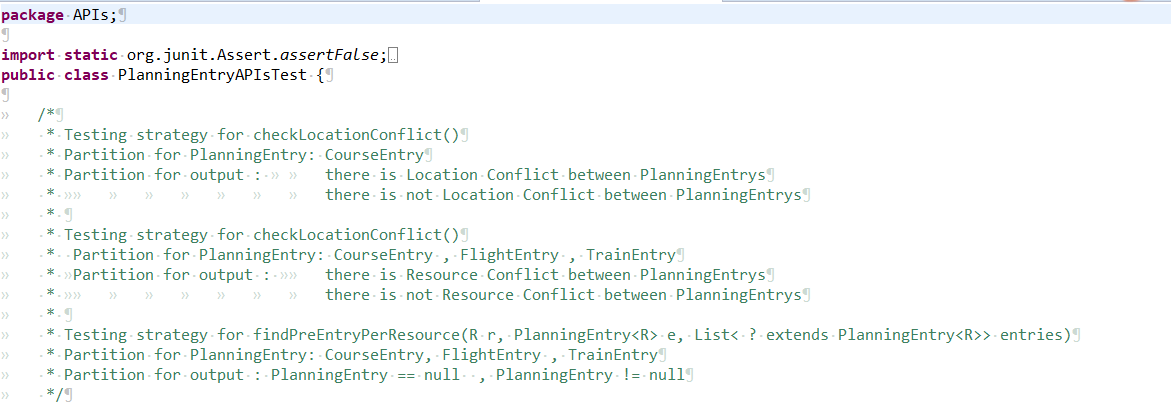
具体的实现，在上面已经阐述。

## Testing for Robustness and Correctness

### Testing strategy

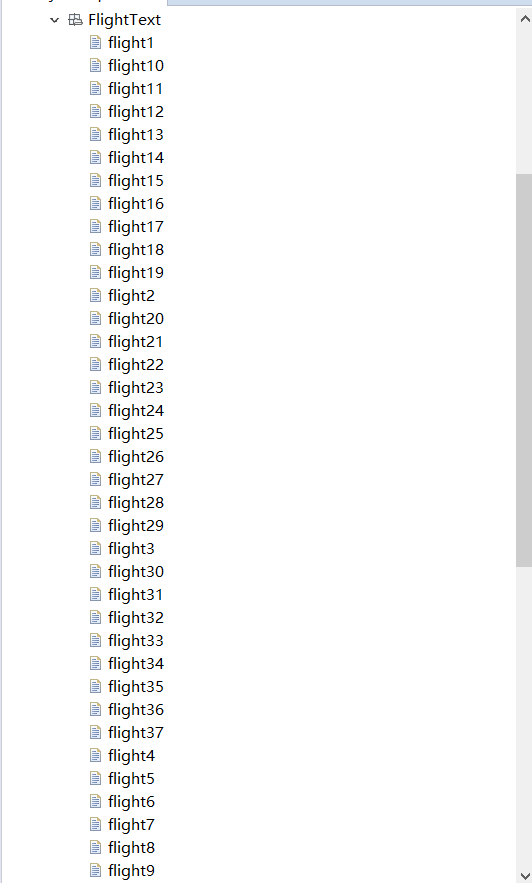
使用等价类和边界值的测试思想，为各 ADT 添加 testing strategy。

以PlanningEntryAPIS类为例：



### 测试用例设计

对于所有的涉及文件读取的异常，我都专门写了测试文件来进行测试。

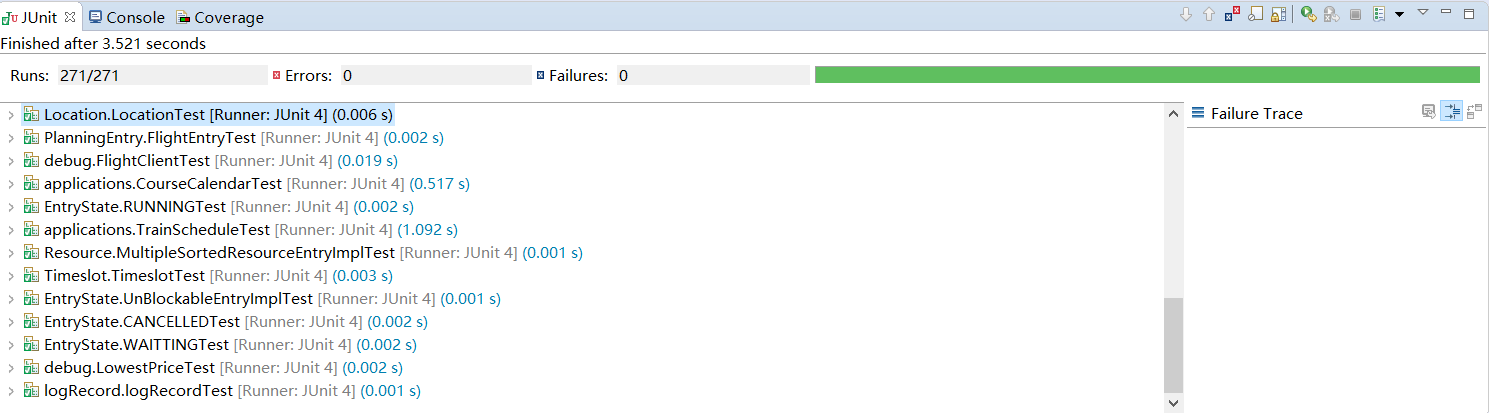
具体的每个文件的测试异常种类详细见代码

对于判断读取文件是否出现异常的四个方法，我的覆盖率为:

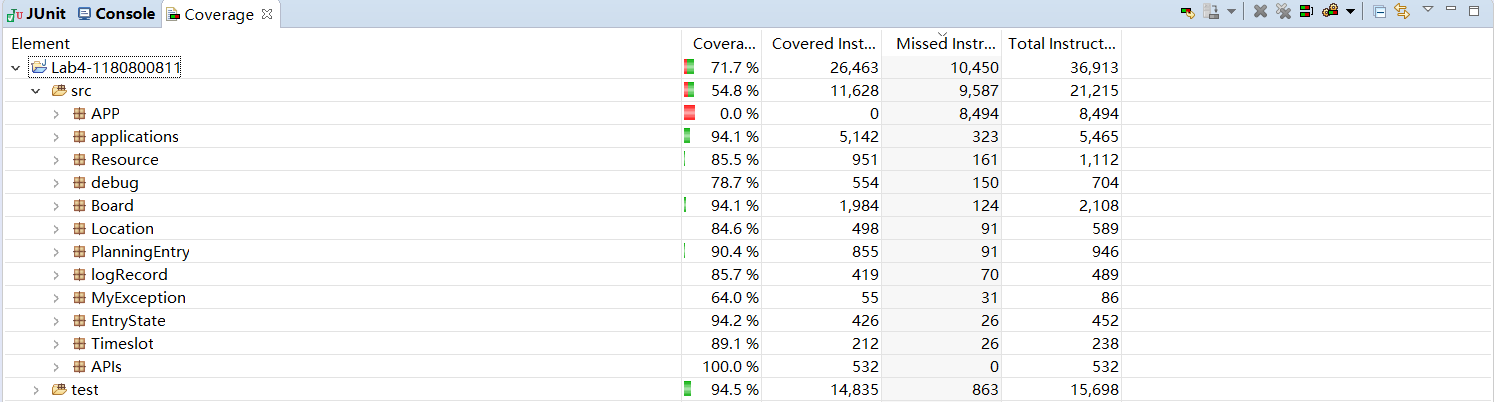


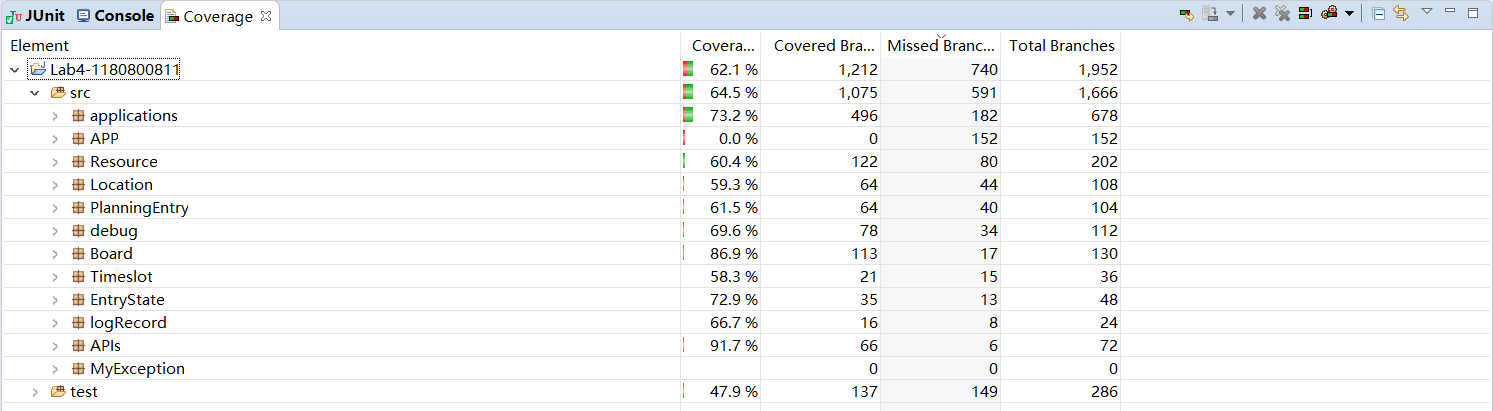
### 测试运行结果与EclEmma覆盖度报告



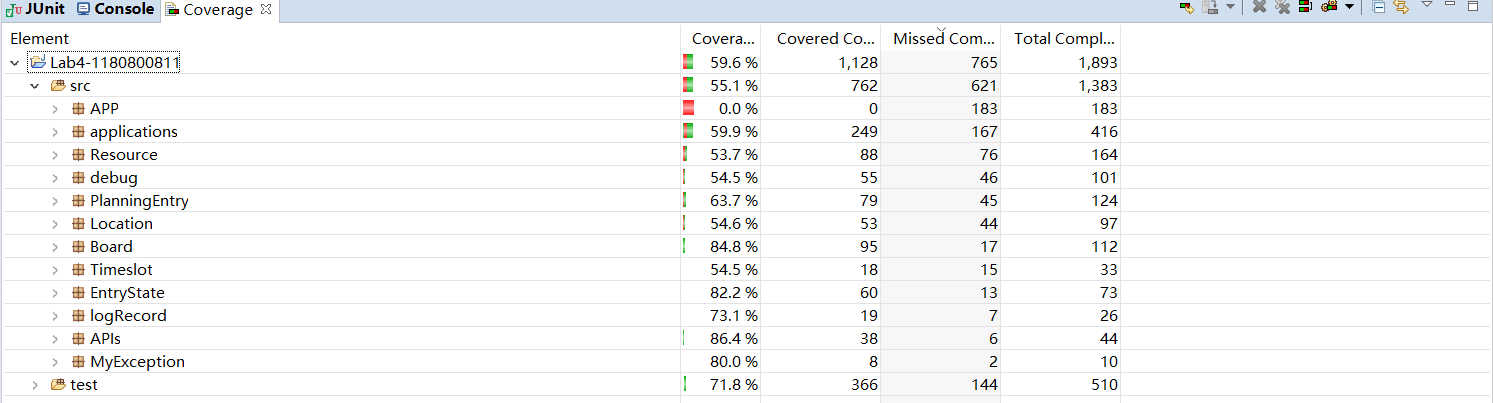
语句覆盖度:



分支覆盖度



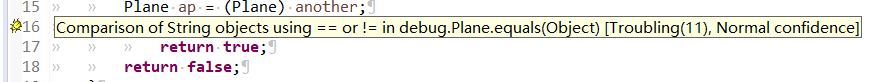
路径覆盖度



## SpotBugs tool

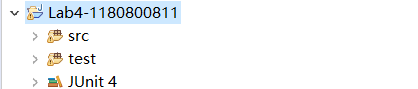
* 1. 一开始粘贴3.6节老师给的debug代码的时候， 出现的bug是判断相等用的是==，而不是equals方法





改为equals之后bug自然消失。

除了这个bug之外我没有其他的bug



## Debugging

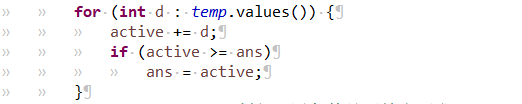
### EventManager程序

功能:给定许多个事件的起止时间，需要找出某一时刻同时进行的事件的最大数量。

#### 理解待调试程序的代码思想

属性是类型为TreeMap的变量temp，temp的键存储的是事件在时间轴上的时间，值对应的是在该时间点上面同时发生的时间数量的最大值。所以只需要循环temp的所有的值，找出最大的值即可。

使用TreeMap的好处:TreeMap已经是按照键的大小排序，所以可以直接遍历temp，表示从时间轴向后遍历所有的事件，而不需要对键重新进行排序。



初始化:每次加入一个事件，就把这个事件对应的时间点的原来发生的时间数量加一。

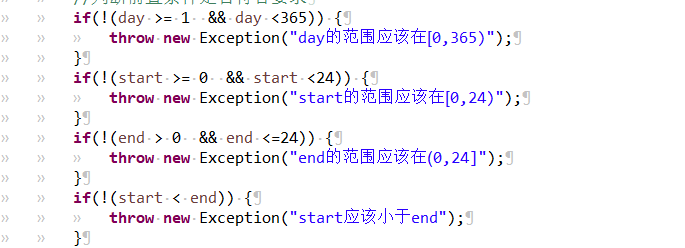
#### 发现并定位错误的过程

需要修改的程序有以下几点错误:

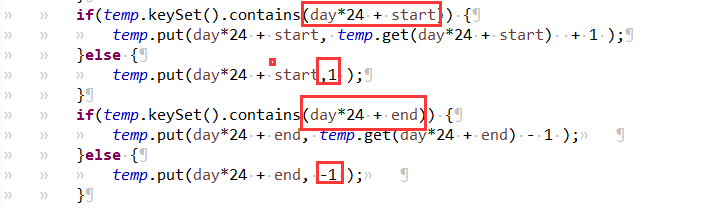
* 1. 没有检查前置条件是否符合。
  2. 初始化错误，一是需要存储的键没有考虑day这个属性，二是需要判断待加入的事件开始和结束的时间点在temp的键中是否存在，如果不存在，则开始时间对应的值应该置1，结束时间置-1。

#### **你如何修正错误**

1. 增加前置条件



1. 初始化:键要考虑day这个属性。而且需要判断待加入的事件开始和结束的时间点在temp的键中是否存在

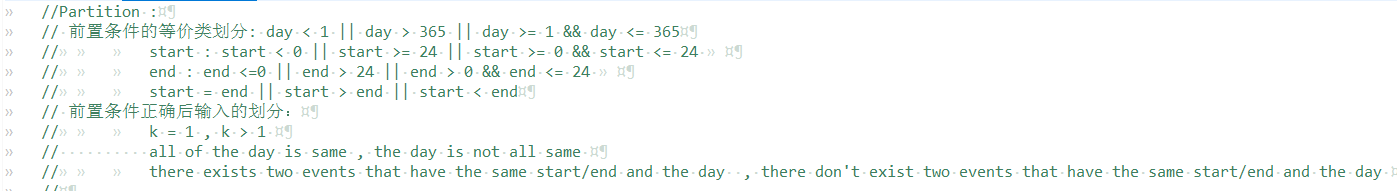


1. 后置条件



#### 修复之后的测试结果

测试策略:





### LowestPrice程序

程序功能:给定一组优惠，并且给出我们所需要的各个物品的数量，找出需要购买这些物品的最低价格。

#### 理解待调试程序的代码思想

程序思想:首先求出按照商品的单价直接购买商品所需要的价格res。然后，遍历所有的优惠，如果这个优惠的需要的商品的数量不超过我们需要的数量，则递归的求出使用优惠所需要的价格。最低价格就是res和使用优惠的最小值。

res = Math.*min*(res, s.get(j) + shopping(price, special, clone));

#### 发现并定位错误的过程

错误1:循环终止条件错误，如果优惠的需要的某种物品数量大于我们需要的这种物品的数量，则直接跳过这个优惠，选取下一个优惠，而不是继续判断另一种物品的数量是否满足要求。

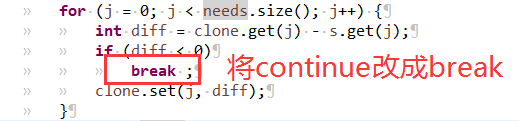
错误2:循环次数错误，List最大下标是size-1，而不是size。

错误3:判断优惠是否符合错误，diff= clone.get(j) - s.get(j)，diff = 0 也符合要求。就是优惠需要的数量和我们需要的数量恰好相等。

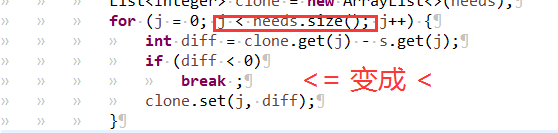
错误4:没有判断前置条件是否符合

#### 你如何修正错误

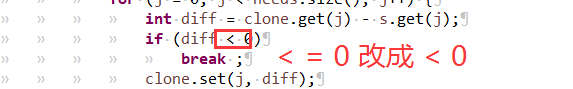
错误1：循环终止条件错误，直接将continue改成break。



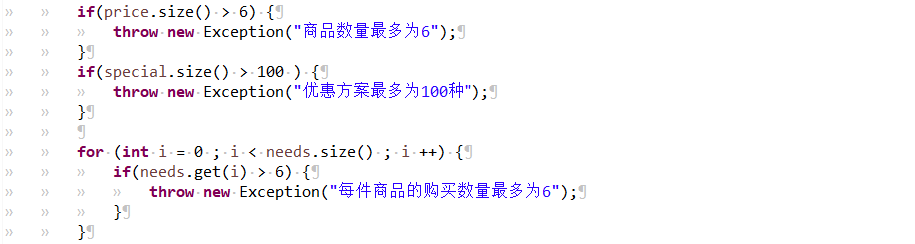
错误2：循环次数错误，将终止条件size 变成size-1



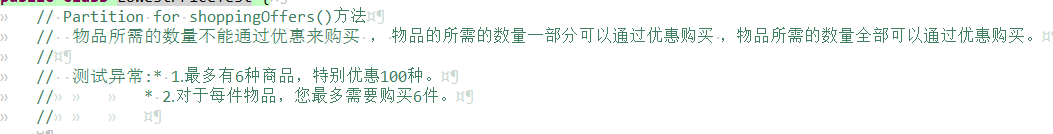
错误3: 判断优惠是否符合错误,将diff <= 0 改成 diff < 0 ;

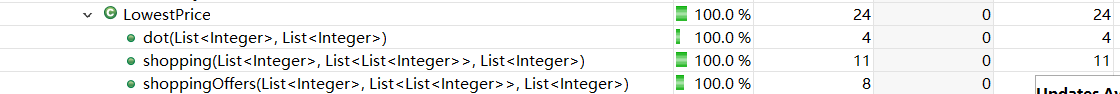


错误4: 判断前置条件是否符合



#### 修复之后的测试结果





### FlightClient/Flight/Plane程序

功能:给定一组航班和一组飞机，问能否在飞机不冲突的情况下给每个航班都安排一架飞机。

#### 理解待调试程序的代码思想

首先，对航班按照出发时间进行排序，然后随机选取一架飞机，判断如果给这个航班安排飞机是否存在冲突，如果存在冲突选取别的飞机，遍历所有的飞机如果仍然不能安排飞机，则返回false。

#### 发现并定位错误的过程

错误1:对航班的出发时间排序错误。

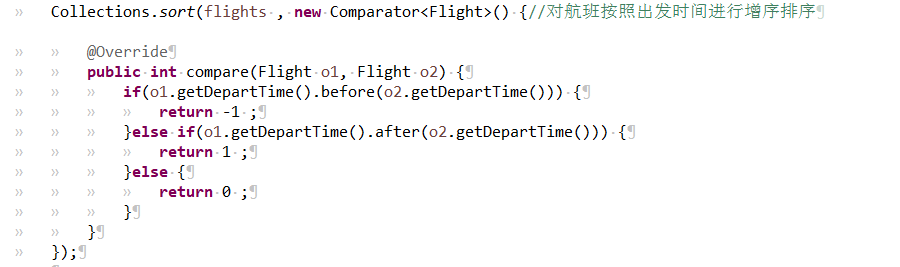
错误2:终止错误，没有给出当不能给某个航班分配飞机时候返回false

错误3:时间比较错误

错误4:随机分配飞机错误，无法保证随机的分配飞机会覆盖到所有的飞机。也没有判断随机选择的飞机之前是否已经选择。

#### 你如何修正错误

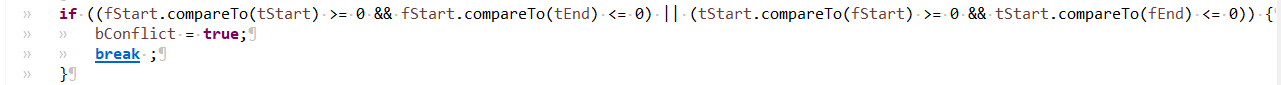
错误1：对航班的出发时间排序错误。



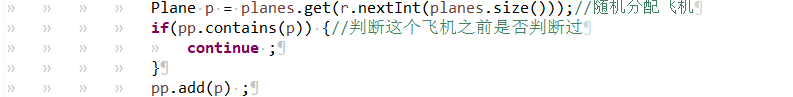
错误2：终止错误，没有给出当不能给某个航班分配飞机时候返回false



错误3：时间比较错误

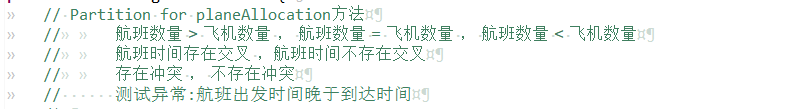


错误4：随机分配飞机错误





#### 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。

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 |
| 2020-05-24 | 20:30-11:00 | 完成3.1 | 未完成 |
| 2020-05-24 | 8:00-10:00 | 完成3.1 | 提前完成 |
| 2020-05-24 | 14:00-18:00 | 完成3.2 | 未完成 |
| 2020-05-24 | 20:00-22:00 | 完成3.2测试类bug | 未完成 |
| 2020-05-25 | 全天 | 完成3.4logging | 完成 |
| 2020-05-26 | 全天 | 完成所有类的测试和规约撰写等等 | 完成 |
| 2020-05-31 | 21:00-23:00 | 完成debug | 未完成 |
| 2020-06-01 | 07:30-09:30 | 完成debug | 未完成 |
| 2020-06-01 | 14:00-16:30 | 完成debug | 完成 |
| 2020-06-02 | 07:40-09:40 | 完成实验报告 | 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 |
| 日志不会 | 上b站学 |

# 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？

正确性只需要考虑保持不变量的不变，即保证ADT内部的正确性。

健壮性则是要考虑外部的错误对内部正确性的影响，考虑更多的是如果出现错误了怎么处理等等。

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？）

划算，我们应该考虑更多的异常，把困难留给开发人员，保证用户端的使用正确性。如果什么错误都考虑类似于在民航飞机上安装降落伞，我觉得这样没人敢坐飞机了。美国十几年前也是因为一个小小的错误导致火箭发射失败，这个代价可想而知。

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？

前者对客户的要求不高，客户的操作空间很大，考虑了客户的需求。后者对客户的限制比较多，把规范留给了客户。

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？

在我们构建代码时候，过分的防御是好的，这样可以帮助我们更快的发现错误，当代码构建完成时，将要发布给用户之后，过多的防御不必要，这样会影响性能。

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？

很明显，Assertion和log会帮助我们更快的定位错误。Lab3我遗留了一些错误，lab4使用断言才发现。

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？
2. 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。

一点也不快乐，没有注释没有文档，我得先猜出这个错误的程序的实现思路，然后再去改正。

1. 关于本实验的工作量、难度、deadline。

工作量小，难度较低，deadline合适。

1. 到目前为止你对《软件构造》课程的评价和建议。

完美的课程，让我有了想要学软件工程的冲动。

1. 期末考试临近，你对占成绩60%的闭卷考试有什么预期？

希望老师最后几个题目给的ADT不要太难，太难的话大家都没时间做完，根本没机会检验水平。要检验水平是检验不出来的，太简单的话，涉及的知识太浅，也检验不出来。中等偏上的题目就很容易检验出我们学到了多少东西，哪里没掌握等等。嘿嘿