哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

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	唐顺江
学号	1180300502
班号	1803005
电子邮件	2657871974@qq.com
手机号码	13640532286

目录

1 实验目标概述 1
2 实验环境配置 1
3 实验过程 1
3.1 Error and Exception Handling 1
3.1.1 处理输入文本中的三类错误 1
3.1.2 处理客户端操作时产生的异常 4
3.2 Assertion and Defensive Programming 5
3.2.1 `checkRep()`检查 rep invariants 5
3.2.2 Assertion/异常机制来保障 pre-/post-condition 6
3.2.3 你的代码的防御式策略概述 8
3.3 Logging 8
3.3.1 异常处理的日志功能 8
3.3.2 应用层操作的日志功能 9
3.3.3 日志查询功能 9
3.4 Testing for Robustness and Correctness 9
3.4.1 Testing strategy 9
3.4.2 测试用例设计 9
3.4.3 测试运行结果与 EclEmma 覆盖度报告 10
3.5 SpotBugs tool 11
3.6 Debugging 12
3.6.1 `EventManager` 程序 12
3.6.2 `LowestPrice` 程序 13
3.6.3 `FlightClient`/`Flight`/`Plane` 程序 14
4 实验进度记录 15
5 实验过程中遇到的困难与解决途径 15
6 实验过程中收获的经验、教训、感想 15
6.1 实验过程中收获的经验和教训 15
6.2 针对以下方面的感受 15

1 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

2 实验环境配置

在这里给出你的 GitHub Lab4 仓库的 URL 地址（Lab4-学号）。

https://github.com/ComputerScienceHIT/Lab4-1180300502

3 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！）。

3.1 Error and Exception Handling

3.1.1 处理输入文本中的三类错误

以 filght 为例，
1.在 plane 类里，

```
String planename;//飞机名称
String planemodel;//飞机型号
int planeseats;//飞机座位数
double planeage;//飞机机龄
```

由于飞机名称和型号是字符串类型，故不需要检查输入，而飞机座位数和机龄分别是 int 和 double，需要在输入时进行检测。

```
while (!flag) {
    System.out.println("请输入飞机座位数：");
    if (a.hasNextInt()) // 检测输入的是不是整型数字
        flag = true;
    else {
        flag =false;
        System.out.println("输入格式错误，请输入整型数字");
    }
    planeseats = a.nextInt();
}
flag =false;
    flag =false;
    while (!flag) {
        System.out.println("请输入飞机机龄：");
        if (a.hasNextDouble()) // 检测输入的是不是整型数字
            flag = true;
        else {
            flag =false;
            System.out.println("输入格式错误，请输入浮点型数字");
        }
        planeage = a.nextDouble();
    }
```

2.Filghtlocation 类同理

```
private double x;//经纬度
private double y;
```

需要检测经纬度是否维 double
采用 while 循环

```
while (!flag) {
    if (a.hasNextDouble()) // 检测输入的是不是浮点型数字
        flag = true;
    else {
        System.out.println("输入格式错误，只能输入浮点型");
        flag = false;
    }
    this.x = a.nextDouble();
}
```

3. CommonPlanningEntry 类

在 public L addresource(List<L> list) {// 让用户输入数字选择资源加入
检测用户输入的是否为 int 型

```
while(!flag) {
    if (a.hasNextInt()) // 检测输入的是不是整型数字
        flag = false;
    else {
        System.out.println("输入格式错误，请输入整形数字");
        flag = false;
    }
    i = a.nextInt();
```

public Location addlocation(List<Location> list) {// 让用户选择地址加入， 同理

函数 `public void addtimelot() {// 让用户输入一个时间，前一个时间的开始不能大于后一个时间的结束，boolean 代表是否有结束时间`
检测用户输入的时间是否符合规则

```
System.out.println("时间格式为yyyy-mm-dd hh:mm，例如：“2019-12-05 13:23”");
System.out.println("请输入开始时间：");
while (!flag) {// 判断是否符合规则
    starttime = a.nextLine();
    if (TimesLot.istime(starttime))
        flag = true;
    else{
        System.out.println("输入时间不符合规则，请重新输入");
        flag = false;
    }
```

用户如果输入的时间不符合规则，自动检测并判断

```
    if (timeslot.size() > 0 && flag) {//如果结束时间早于开始时间
        if (TimesLot.comparetimestring(starttime, overtime))
        {
            flag = false;
            System.out.println("结束时间早于开始时间，请重新输入");
            System.out.println("开始时间为："+starttime+"，请输入晚于该时间的数据");
        }
    }
}
```

4. TimesLot 类

`static public boolean istime(String s) {//判断是否为规范的字符串`
利用 java 中的正则表达式检测输入数据是否符合规则

```
static public boolean istime(String s) {//判断是否为规范的字符串
    Pattern a=Pattern.compile("(\\d{4})-(\\d{2})-(\\d{2}) (\\d{2}):(\\d{2})");
    Matcher b=a.matcher(s);
    if(!b.matches())
    {
        return false;
    }
    else
    return true;
}
```

5. PlanningEntryAPI 类

`static public boolean timeconflict(List<TimesLot> lot) {//按照时间的早晚给 TimesLot 排序，一旦发现有前一个的结束时间在后一个开始时间之后，就判定该位置有位置冲突返回 false`
检测时间冲突

```
public boolean checkResourceExclusiveConflict(List<PlanningEntry<L>>
```

```
entries){//检查一组计划项中的资源冲突
```

3.1.2 处理客户端操作时产生的异常

主函数异常检测

```
public FlightScheduleApp() throws ParseException, IOException {
```

```
static public int num(int a, int b) {// 要求输入数字，检测是否为范围内的整数
```

设置数字检测函数，检测数据输入是否符合规则

```
static public int num(int a, int b) {// 要求输入数字，检测是否为范围内的整数
    Scanner s = new Scanner(System.in);
    boolean flag=false;
    int i = 0;
    while (!flag){
        if (s.hasNextInt()) // 检测输入的是不是整型数字
            flag = true;
        else {
            System.out.println("输入格式错误，只能输入整型");
            flag = false;
        }
        i = s.nextInt();
        if (flag && (i < a || i > b)) {// 检测输入的是不是范围内的整数
            System.out.println("输入数据超出范围，请重新输入");
            flag = false;
        }
    }
    return i;
}
```

后续调用该函数检测输入

```
private boolean readfile()throws IOException, ParseException{
    System.out.println("请输入你想读的文件编号：(编号从1到5，分别对应老师给出的5个文件)");
    int i=num(1,5);
```

利用 pattern 类和 matcher 类判断输入数据是否符合规则

```
SimpleDateFormat s2 = new SimpleDateFormat("yyyy-MM-dd");

Pattern b = Pattern.compile("Flight:(\\d{4}-\\d{2}-\\d{2}),([A-Z]{2})(\\d{2,4})\\n"
        + "\\{\\nDepartureAirport:([A-Za-z]+)\\nArrivalAirport:([A-Za-z]+)\\nDepatureTime:(\\d{4}-\\d{2}-\\d{2})"
        + " (\\d{2}:\\d{2})\\nArrivalTime:(\\d{4}-\\d{2}-\\d{2}) (\\d{2}:\\d{2})\\nPlane:(B|N)"
        + "(\\d{4})+\\n\\{\\nType:([A-Z0-9]+)\\nSeats:(\\d{1,3})\\nAge:(\\d+)(.\\d{1}|)\\n\\}\\n\\}\\n");
Matcher matcher1 = b.matcher(s1);
```

输入数据与要求不符合，如时间间隔大于 1 天

```
if (betweenDate < 0 || betweenDate > 1)// 相差不为一天，返回false
{
    System.out.println("日期错误，时间间隔太长");
    return false;

}
if (matcher1.group(6).equals(matcher1.group(1)))// 与第一行不一致，返回false
    m++;
else{
    System.out.println("日期错误！");
    return false;
```

输入数据非要求类型

```
if (Integer.parseInt(matcher1.group(13)) > 600 || Integer.parseInt(matcher1.group(13)) < 50)// 与第一行不一致，返回false
{
    System.out.println("座位数非法");
    return false;

}
planeseat.add(matcher1.group(13));
if (matcher1.group(15).length()==2&&matcher1.group(14).indexOf("0") == 0 && matcher1.group(14).length() > 1) {
    System.out.println("机龄输入非法！");
    return false;
```

判断当前的航班数据是否和其他航班有冲突

```
for (int j = 0; j < n - 1; j++) {
    if (flightname.get(j).equals(name)) {// 同名航班
        if (!start.get(j).equals(thisstart) || !over.get(j).equals(thisover)) {
            System.out.println("同名航班地址不匹配！");
            return false;
        }
        if (starttime1.get(j).equals(thisstarttime1) || overtime1.get(j).equals(thisovertime1)) {
            System.out.println("同名航班每天有两个日期！");
            return false;
        }
        if (!starttime2.get(j).equals(thisstarttime2) || !overtime2.get(j).equals(thisovertime2)) {
            System.out.println("同名航班到达时间不同！");
            return false;
        }
```

输入文本类数据不符合要求

```
if (i / 13 > flightname.size() )// 每个信息项都有13行，如果行数除以13大于航班的数量，说明有的航班没有录上，输入非法
{
    System.out.println("文件输入不符合规定，该文件无法读入！");
    System.out.println(i);
    System.out.println(flightname.size());
    return false;
}
```

3.2 Assertion and Defensive Programming

3.2.1 checkRep()检查 rep invariants

1. TimesLot 类

```
private void checkRep() {
        assert (startString!="未设置"&&  overString!="未设置");
        for (int i = 0; i < 5; i++) {
            assert(starttime[i]!=0);
             assert (overtime[i] != 0);
```

```
        }
    }
```

2. Location 类

```
private void checkRep() {
    for (String strLocation1 : locations) {
        assert (strLocation1.length() > 0);
        for (String strLocation2 : locations) {
            if (strLocation1 != strLocation2)
                assert (!strLocation1.equals(strLocation2));
        }
    }
}
```

3. EntryState

```
    public void checkRep(){//判断函数保证只有同时存在一个状态
        int i=0;
        if(run)
            i++;
        if(hold)
            i++;
        if(complete)
            i++;
        if(cancel)
            i++;
        assert (i>=0&&i<=1);
        //assert (i==0&&!assign)||(i==1&&assign);
    }
```

4. PlanningEntry 类

抽象类，没有 checkrep

5. CommonPlanningEntry 类

```
    private void checkRep() {//状态检查
        entrystate.checkRep();
    }
```

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

TrainEntry 类

输入的火车站必须小于已有的火车站数量

```
    public void setEntry(List<Location> list) {// 要求用户依次键入始发站，终点站，时间段
```

```
        System.out.println("请输入一个数字确定要火车要经过几站：");
        int num =num(2, list.size());// 要求用户键入一个数字来确认该火车
经过几站;
        System.out.println("选择始发站：");
        addlocation(list);// 选择始发站
        for (int i = 0; i < num - 1; i++) {
            addtimelot();// 键入时间段
            System.out.println("选择第"+(i+2)+"站：");
            addlocation(list);
        }
    }


    static public int num(int a, int b) {// 要求输入数字，检测是否为范
围内的整数
        Scanner s = new Scanner(System.in);
        boolean flag=false;
        int i = 0;
        while (!flag){
            if (s.hasNextInt()) // 检测输入的是不是整型数字
                flag = true;
            else {
                System.out.println("输入格式错误，只能输入整型");
                flag = false;
            }
            i = s.nextInt();
            if (flag && (i < a || i > b)) {// 检测输入的是不是范围内的整
数
                System.out.println("输入数据超出范围，请重新输入");
                flag = false;
            }
        }
        return i;
    }
```

EntryState 类

函数条件是：函数保证只有同时存在一个状态

```
        int i=0;
        if(run)
            i++;
        if(hold)
            i++;
        if(complete)
            i++;
```

```
        if(cancel)
            i++;
        assert (i>=0&&i<=1);
```

3.2.3 你的代码的防御式策略概述

在 PlanningEntryAPI 类中

```
        for(Location location : locations) {//依次检查输入的地址表中的位置是否有冲突
            for(PlanningEntry<L> entry: entries) {//将所有的计划项中的location对应的TimesLot提取出来
                if(entry.getlocations().contains(location))
                timelots.add(entry.gettimeslot().get(entry.getlocations().indexOf(location)));
            }
            if(timeconflict(timelots)) {
                timelots.clear();//清空时间表
                return true;
            }
            else
                return false;
        }
        return true;
    }
```

在 TimesLot 类中

```
  static public boolean comparetimestring(String a,String b) {//比较两个时间是否为先后
                                                               //a大于b返回true
      if(comparetime(timestringtoint(a),timestringtoint(b)))
          return true;
      else
          return false;

    static public boolean istime(String s) {//判断是否为规范的字符串
        Pattern a=Pattern.compile("(\\d{4})-(\\d{2})-(\\d{2}) (\\d{2}):(\\d{2})");
        Matcher b=a.matcher(s);
```

在 CommonPlanningEntry 类中

```
        if (timeslot.size() > 0 && flag) {// 若前一个结束时间大于开始时间，即当前列表前方有其他计划项还未结束，将flag设置为false
            TimesLot time = timeslot.get(timeslot.size() - 1);// 获取该计划项的末尾时间段
            if (TimesLot.comparetimestring(time.getovertime(), starttime))// 前一个结束时间大于当前开始时间
            {
                flag = false;
                System.out.println("当前输入时间早于上一个结束时间，请重新输入");
                System.out.println("上一个结束时间为："+time.getovertime()+"，请输入晚于该时间的数据");
            }
        }
```

3.3 Logging

3.3.1 异常处理的日志功能

利用如下代码建议实现打印日志的功能

```
    public static void print() {
        Logger logger = Logger.getLogger(FlightScheduleApp.class.getName());
        try {
            "".getBytes("invalidCharsetName");
        } catch (UnsupportedEncodingException e) {
            logger.severe(e.toString());
            e.printStackTrace();
        }
    }
```

3.3.2 应用层操作的日志功能

```
            System.out.println("13:打印日志");
                break;
            case 13:
                print();
                break;
```

利用如上的简易代码，在使用程序时，输入指定的数字，打印当前的日志信息

3.3.3 日志查询功能

3.4 Testing for Robustness and Correctness

3.4.1 Testing strategy

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

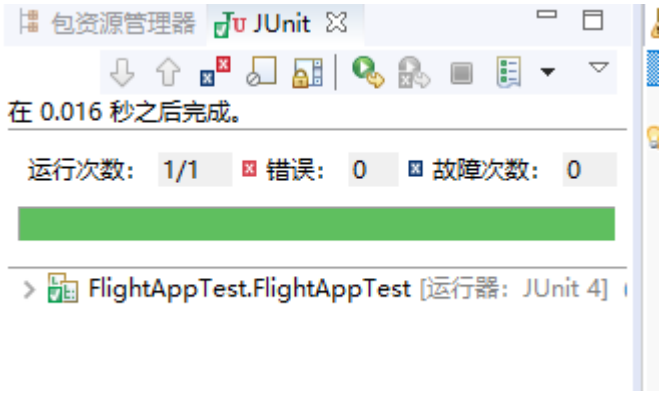
3.4.2 测试用例设计

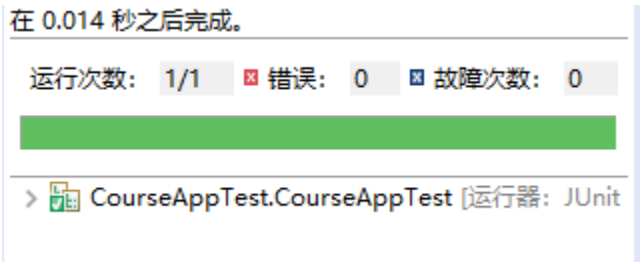
如 filghtapp 的测试
代码如下

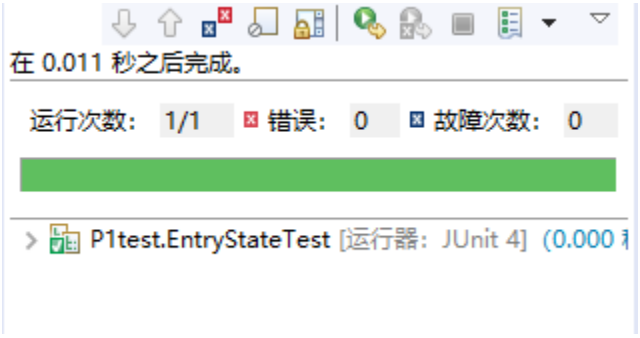
```java
package FlightAppTest;

import static org.junit.Assert.*;

public class FlightAppTest {

    @Test
    public void test() {
        FlightEntry f=new FlightEntry("plane");
        Plane p=new Plane("asr","a55",50,3.5);
        TimesLot t=new TimesLot("2019-10-01 10:10","2019-10-01 10:30");
        FlightLocation l1=new FlightLocation("beijing");
        FlightLocation l2=new FlightLocation("tianjin");
        f.setFlightEntry(p, t, l1, l2);
        assertEquals(f.getstartlocations(),l1);
        assertEquals(f.getoverlocations(),l2);
        assertEquals(f.gettimelots(),t);
        assertEquals(p.getname(),"asr");
        assertEquals(p.toString(),"飞机名：asr机型：a55座位数：50年龄：3.5");
    }

}
```

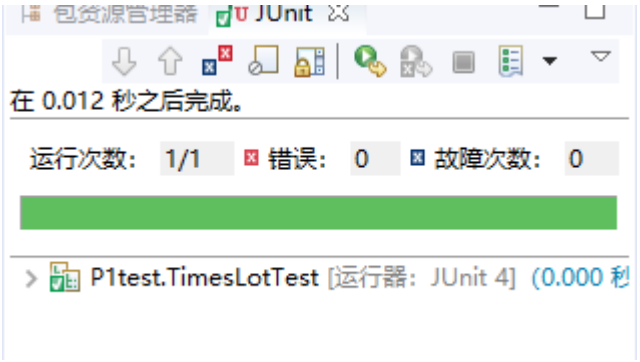
3.4.3 测试运行结果与 EclEmma 覆盖度报告

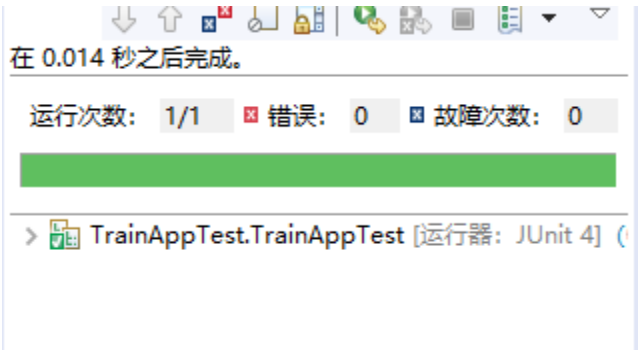
测试运行结果











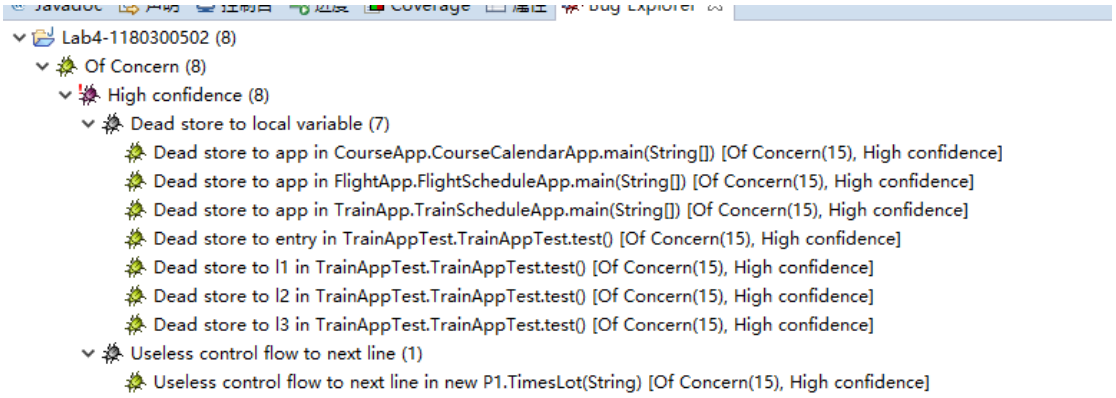
EclEmma 覆盖度报告

TimesLotTest (2) (2020-6-7 11:16:42)

Element	Coverage	Covered Instructi...	Missed Instructi...	Total Instructions
src	3.9 %	302	7,542	7,844
test	20.0 %	38	152	190
FlightAppTest	0.0 %	0	59	59
P1test	46.3 %	38	44	82
TrainAppTest	0.0 %	0	34	34
CourseAppTest	0.0 %	0	15	15

3.5 SpotBugs tool

发现以下 8 个 bug，但都是一些无关紧要的小问题



3.6 Debugging

3.6.1 EventManager 程序

理解待调试程序的代码思想

发现并定位错误的过程

原程序中 `static TreeMap<Integer, Integer> temp = new TreeMap();`

你如何修正错误

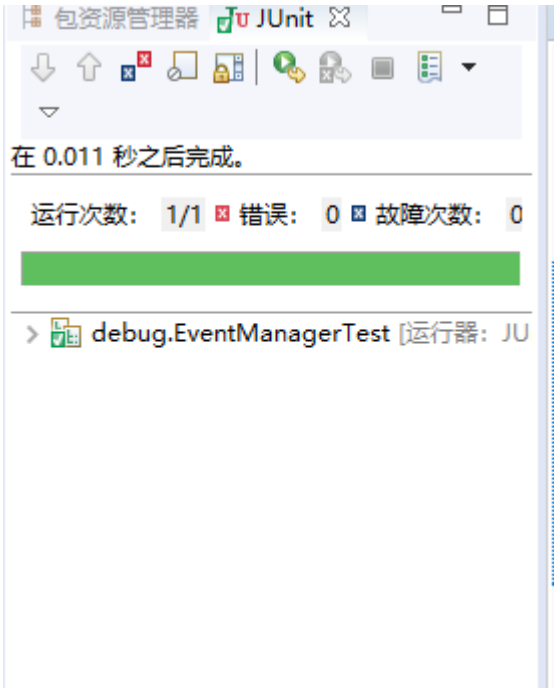
应该改为`static TreeMap<Integer,TreeMap<Integer, Integer>> dayMatch=new TreeMap<Integer, TreeMap<Integer, Integer>>();//每一天对应一个map`

`static TreeMap<Integer,Integer> dayOfk=new TreeMap<Integer, Integer>();//每一天都有一个 k 值`

这样才能实现功能

余下代码根据作用进行对应的修改

修复之后的测试结果



3.6.2 LowestPrice 程序

理解待调试程序的代码思想

发现并定位错误的过程

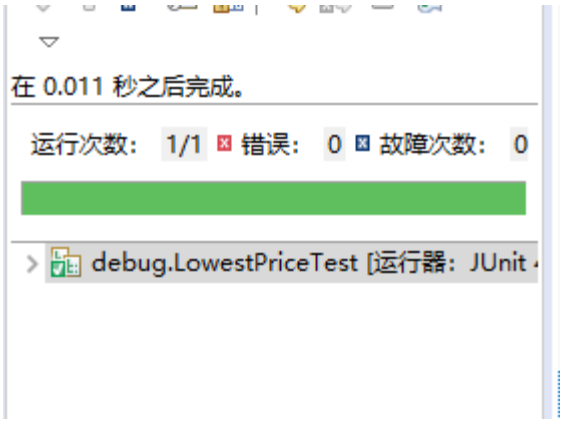
```
for (j = 0; j <= needs.size(); j++)
```

你如何修正错误

```
for (j = 0; j <needs.size(); j++)
```

循环不应超出指定范围

修复之后的测试结果



3.6.3 FlightClient/Flight/Plane 程序

理解待调试程序的代码思想

该程序通过枚举每个航班，尝试安排飞机，确保没有与其他已经分配的航班冲突，最后确认是否能所有同时分配成功

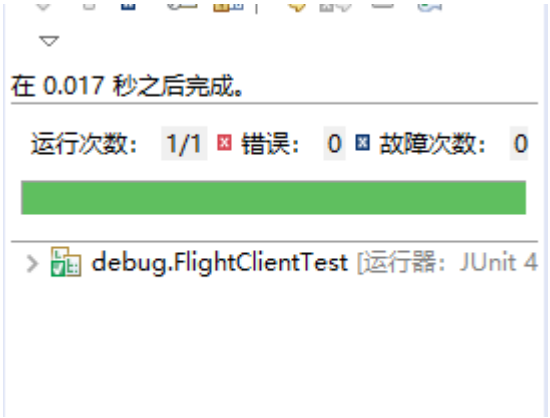
发现并定位错误的过程

你如何修正错误

添加部分代码，设置退出循环的条件

```
int num=planes.size();
int count=0;
if(count>num)//设置退出循环的条件
break;
```

修复之后的测试结果



4 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
2020-5-20	15h	完成 lab3 代码的防御性功能健全	完成
2020-5-28	6h	完成日志功能的实现	完成
2020-6-1	10h	完成 debug 实验	完成

5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
对于日志功能完全没有头绪	百度，搜索教程，参考实例熟悉
对 testing strategy 的完善难以进行	询问同学，对实验重新思考
对于 SpotBugs 工具的使用	搜索相关资料，并实践学习

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验和教训

6.2 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：在任何时刻都必须考虑如果用户输入错误数据时程序需要怎么运行才能得到正确结果或者给用户提示，在完成程序健壮性时，往往会因为疏忽忽略掉某些错误，导致健壮性不够号，所以这要求程序员要能够全

面考虑
正确性：在任何时候都必须输出正确的答案，这是程序完成的基本要求

(2) 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，因为程序员无法保证为了修复 bug 而添加的代码永远正确，如果为了百分之百的正确，程序完成难度大大提高并且付出的代价极高

(3) “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？

一个是健壮性，一个是正确性。
在做程序时选择健壮性，因为实际情况会很复杂，不同得到用户会有不同的操作，保证健壮性才能保证程序的基本功能。
但是在如考试的情况下，我选择正确性，这个时候正确性才是完成实验得到标准

(4) 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
没有必要
对于无关紧要的 bug 选择性忽略，对于重要的部分优化防御

(5) 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？

利用 print 方法打印信息，查看与自己的需求是否一致
有帮助

(6) 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？

一般来说，覆盖度高代码测试充分，但是 100%并不代表充分测试，一个代码对应的不会只有一个输入，而一个输入就能覆盖这个代码

(7) 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
很难受

(8) 关于本实验的工作量、难度、deadline。

合适，尤其是在经历了 lab3 的摧残后

(9) 到目前为止你对《软件构造》课程的评价和建议。

完美的课程，实验真的是对我帮助极大

(10)期末考试临近，你对占成绩 60%的闭卷考试有什么预期？

希望不要太难，或者不要超出预期的难