

**2020年春季学期  
计算机学院《软件构造》课程**

**项目报告**

|  |  |
| --- | --- |
| 姓名 | 余涛 |
| 电子邮件 | [1063695334@qq.com](mailto:1063695334@qq.com) |

**目录**

[1 实验目标概述 1](#_Toc37611518)

[2 实验环境配置 1](#_Toc37611519)

[3 实验过程 2](#_Toc37611520)

[3.1 待开发的三个应用场景 2](#_Toc37611521)

[3.2 面向可复用性和可维护性的设计：PlanningEntry<R> 3](#_Toc37611522)

[3.2.1 PlanningEntry<R>的共性操作 3](#_Toc37611523)

[3.2.2 局部共性特征的设计方案 4](#_Toc37611524)

[3.2.3 面向各应用的PlanningEntry子类型设计（个性化特征的设计方案） 7](#_Toc37611525)

[3.3 面向复用的设计：R 10](#_Toc37611526)

[3.4 面向复用的设计：Location 13](#_Toc37611527)

[3.5 面向复用的设计：Timeslot 15](#_Toc37611528)

[3.6 面向复用的设计：EntryState及State设计模式 15](#_Toc37611529)

[3.7 面向应用的设计：Board 19](#_Toc37611530)

[3.8 Board的可视化：外部API的复用 21](#_Toc37611531)

[3.9 可复用API设计及Façade设计模式 24](#_Toc37611532)

[3.9.1 检测一组计划项之间是否存在位置独占冲突 24](#_Toc37611533)

[3.9.2 检测一组计划项之间是否存在资源独占冲突 25](#_Toc37611534)

[3.9.3 提取面向特定资源的前序计划项 26](#_Toc37611535)

[3.10 设计模式应用 28](#_Toc37611536)

[3.10.1 Factory Method 28](#_Toc37611537)

[3.10.2 Iterator 30](#_Toc37611538)

[3.10.3 Strategy 32](#_Toc37611539)

[3.11 应用设计与开发 35](#_Toc37611540)

[3.11.1 航班应用 36](#_Toc37611541)

[3.11.2 高铁应用 40](#_Toc37611542)

[3.11.3 进程应用 43](#_Toc37611543)

[3.11.4 课表应用 49](#_Toc37611544)

[3.11.5 学习活动应用 53](#_Toc37611545)

[3.12 基于语法的数据读入 53](#_Toc37611546)

[3.13 应对面临的新变化 58](#_Toc37611547)

[3.13.1 变化1 59](#_Toc37611548)

[3.13.2 变化2 62](#_Toc37611549)

[3.13.3 变化3 63](#_Toc37611550)

[3.14 Git仓库结构 66](#_Toc37611551)

[4 实验进度记录 66](#_Toc37611552)

[5 实验过程中遇到的困难与解决途径 67](#_Toc37611553)

[6 实验过程中收获的经验、教训、感想 68](#_Toc37611554)

[6.1 实验过程中收获的经验和教训 68](#_Toc37611555)

[6.2 针对以下方面的感受 68](#_Toc37611556)

# 实验目标概述

本次实验覆盖课程第3、4、5章的内容，目标是编写具有可复用性和可维护性的软件，主要使用以下软件构造技术：

 子类型、泛型、多态、重写、重载

 继承、代理、组合

 常见的OO设计模式

 语法驱动的编程、正则表达式

 基于状态的编程

 API设计、API复用

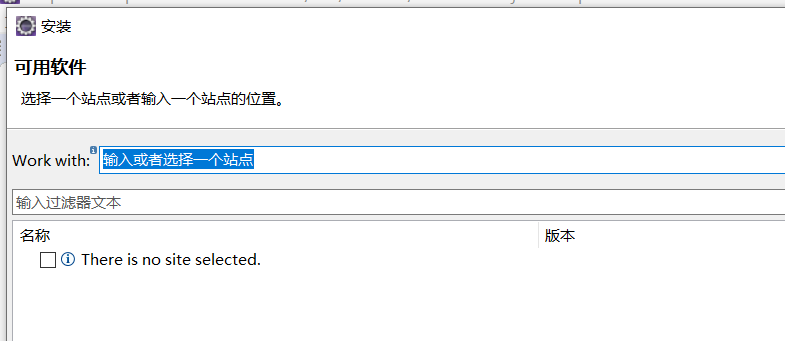
本次实验给定了五个具体应用（高铁车次管理、航班管理、操作系统进程管理、大学课表管理、学习活动日程管理），学生不是直接针对五个应用分别编程实现，而是通过ADT和泛型等抽象技术，开发一套可复用的ADT及其实现，充分考虑这些应用之间的相似性和差异性，使ADT有更大程度的复用（可复用性）和更容易面向各种变化（可维护性）。

# 实验环境配置

本次实验因为需要完成Board的可视化，所以我在eclipse中下载了WindowBuilder插件，方便GUI的实现。

具体安装过程为：

打开eclipse,选择help--->install new software



来源http://www.cnblogs.com/xiaobo-Linux/p/7954274.html

打开网址

http://www.eclipse.org/windowbuilder/

点击download

在相对应eclipse版本，link右键复制链接，然后复制到这里

然后下一步安装即可。

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

<https://github.com/ComputerScienceHIT/Lab3-1180300829>

# 实验过程

## 待开发的三个应用场景

三个应用

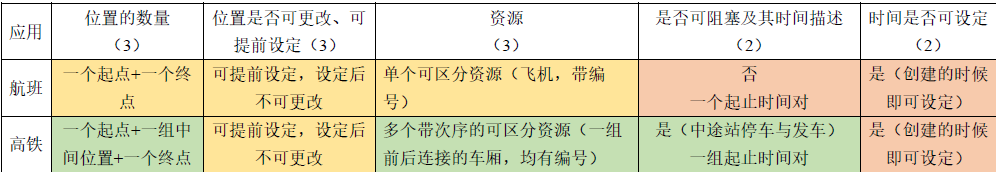
1.航班管理：一个航班从某地机场起飞，抵达另一个机场（不考虑中途经停的情况）。航空公司需要分配一架具体飞机（资源）来执飞特定日期的航班。航空公司可以发布新航班、给新航班分配飞机、起飞、抵达目的地。航班在出发前可以被取消。航班一旦确定，不可更改起始和目的地。

2.高铁车次管理：有很多高铁站和很多高铁车次，每个车次从起点高铁站出发，按次序经过一组高铁站，抵达终点高铁站，经过各站的时间都是预先计划好的。例如，G381次高铁7:55从北京南站出发，8:30抵达天津西站，然后8:32继续出发，历经天津站、唐山站等高铁站，当日14:31抵达终点哈尔滨西站，全程1331公里，历时6小时36分钟。仍以G381为例，该车次每天都有，但不同日期的车次可能由不同的高铁动车组执行。铁路局可以增加一个新车次、为新车次分配一组车厢、始发、在中间站停车、在中间站发车、抵达终点站。高铁在起点站未出发前或中间站停车时可以被取消。同样的，高铁车次一旦确定，也不可更改起始站、中间站、终点站。

3.大学课表管理：上课需要特定的物理位置（教室），需要特定的资源（教师）。学校教务处每学期末统一安排下学期的全校课表，为每门课分配教室、安排教师、确定时间。教师可以上课、下课、更换教室。课程一旦启动，不可被阻塞/挂起、取消，但在未启动之前，教师可以因为临时有事而取消该次课程。

共性：创建一个新的计划项对象启动，取消，完成，获取计划项的名字，获取当前状态

差异：





## 面向可复用性和可维护性的设计：PlanningEntry<R>

### PlanningEntry<R>的共性操作

**3.2.1.1：**PlanningEntry<R>的实现

R代表计划项中涉及的资源类型。

三个应用的共性操作包括：创造一个计划项，启动一个计划项，取消一个计划项，结束一个计划项，得到计划项的名字，得到当前计划项的状态，设置计划项的名字，设置计划项当前的状态：

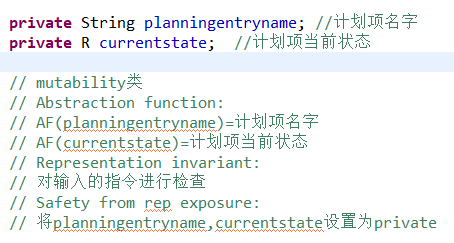
把这些操作都放入PlanningEntry<R>接口中封装起来。

其方法：

|  |  |
| --- | --- |
| createplanningentry | 创造一个计划项 |
| launch | 启动一个计划项 |
| cancel | 取消一个计划项 |
| finish | 结束一个计划项 |
| getplanningentryname | 得到计划项的名字 |
| getcurrentstate | 得到当前计划项的状态 |
| setplanningentryname | 设置计划项的名字 |
| setcurrentstate | 设置计划项当前的状态 |

**3.2.1.2：Common**PlanningEntry<R>的实现

CommonPlanningEntry<R>是PlanningEntry<R>接口的实现类，implements了PlanningEntry<R>接口，对其中的共性方法进行了实现，并且增加了两个字段：



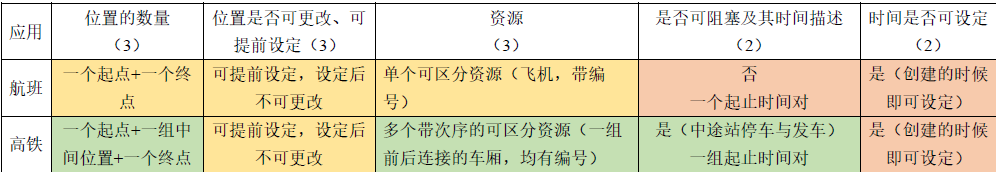
具体的指令导致计划项状态的变化我选择了在app中实现

### 局部共性特征的设计方案

**3.2.2.1：选择实现方案：**

为了提高代码的复用性，我选择了方案5来完成局部共性特征的设计，即CRP，通过接口组合实现局部共性特征的复用，通过delegation 机制进行改造。每个维度分别定义自己的接口，针对每个维度的不同特征取值，分别实现针对该维度接口的不同实现类，实现其特殊操作逻辑。进而，通过接口组合，将各种局部共性行为复合在一起，形成满足每个应用要求的特殊接口（包含了该应用内的全部特殊功能），从而该应用子类可直接实现该组合接口。在应用子类内，不是直接实现每个特殊操作，而是通过delegation 到外部每个维度上的各具体实现类的相应特殊操作逻辑。

**3.2.2.2：找到共性特征并实现：**





观察上表可以综合出三者的共同特征：

按照提取每个方面的共性的方法，可以分别在每个维度设置接口如下：

关于位置（第一列和第二列可以综合在一起）方面：根据上表我们可以设置以下三个接口：

**public** **interface** TwoLocationEntry 两个位置的，设定后不可更改

其方法：

|  |  |
| --- | --- |
| setlocations | 设置起点站和终点站的位置 |
| changelocations | 更改起点站和终点站的位置 |
| getfromlocation | 得到起点站的位置 |
| gettolocation | 得到终点站的位置 |

**public** **interface** MultipleLocationEntry 多个位置的，设定后不可更改

其方法：

|  |  |
| --- | --- |
| setlocations | 设置一组位置 |
| changelocations | 更改这一组位置 |
| getlocations | 得到这一组位置 |

**public** **interface** OneLocationEntry 一个位置的，设定后可更改

其方法：

|  |  |
| --- | --- |
| setlocations | 设置该位置 |
| changelocations | 更改该位置 |
| deletelocations | 删除该位置 |
| getlocations | 得到该位置 |

关于资源方面：根据上表我们可以设置以下两个接口：

**public** **interface** OneDistinguishResourceEntry<R> 单个可区分资源

其方法：

|  |  |
| --- | --- |
| setresource | 设置该资源 |
| getresource | 得到该资源 |
| changeresource | 更改该资源 |

**public** **interface** MultipleSortedResourceEntry<R> 多个带次序的可区分资源

其方法：

|  |  |
| --- | --- |
| setresource | 设置该高铁资源 |
| getresource | 得到该高铁资源 |
| changeresource | 更改某车厢资源 |
| addresource | 向高铁资源里面加入一节车厢 |
| deleteresource | 删除高铁资源中的某车厢 |

关于时间方面（第四列和第五列可以综合在一起）方面：根据上表我们可以设置以下两个接口：

**public** **interface** NoBlockableEntry 不可阻塞时间

其方法：

|  |  |
| --- | --- |
| whetherblockable | 判断该类是否可阻塞 |
| settimeslot | 设置该时间对 |
| gettimeslot | 得到该时间对 |

**public** **interface** BlockableEntry 可阻塞时间

其方法：

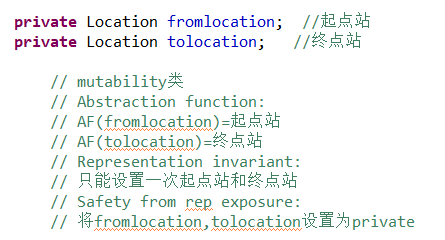
|  |  |
| --- | --- |
| whetherblockable | 判断该类是否可阻塞 |
| settimeslot | 设置这一组时间对 |
| gettimeslot | 得到这一组时间对 |

### 面向各应用的PlanningEntry子类型设计（个性化特征的设计方案）

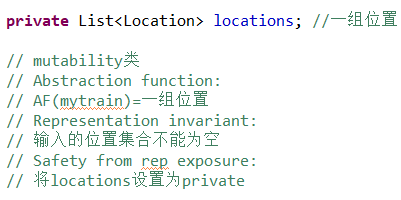
1.分别对每个接口进行实现：

位置：

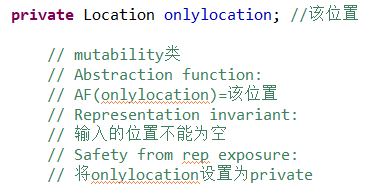
**public** **class** TwoLocationEntryImpl **implements** TwoLocationEntry



**public** **class** MultipleLacationEntryImpl **implements** MultipleLocationEntry

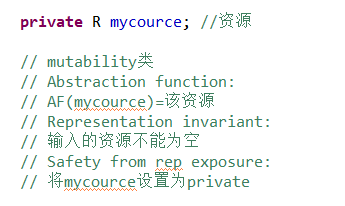


**public** **class** OneLocationEntryImpl **implements** OneLocationEntry



资源：

**public** **class** OneDistinguishResourceEntryImpl<R> **implements** OneDistinguishResourceEntry<R>



**public** **class** MultipleSortedResourceEntryImpl<R> **implements** MultipleSortedResourceEntry<R>



时间：

**public** **class** NoBlockableEntryImpl **implements** NoBlockableEntry

**public** **class** BlockableEntryImpl **implements** BlockableEntry

2.然后定义三个计划项接口：

**public** **interface** FlightPlanningEntry<Flight> **extends** TwoLocationEntry,OneDistinguishResourceEntry<Flight>,NoBlockableEntry

**public** **interface** TrainPlanningEntry<Train> **extends** MultipleLocationEntry,MultipleSortedResourceEntry<Train>,BlockableEntry

**public** **interface** CoursePlanningEntry<Teacher> **extends** OneLocationEntry,OneDistinguishResourceEntry<Teacher>,NoBlockableEntry

3.然后实现这三个计划项接口并继承CommonPlanningEntry中的方法：

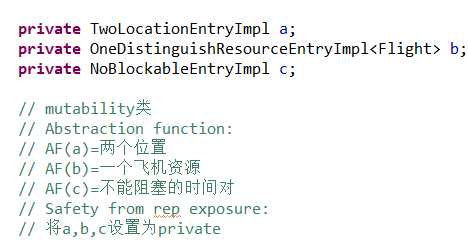
**public** **class** FlightEntry<Flight> **extends** CommonPlanningEntry **implements** FlightPlanningEntry<Flight>

**public** **class** TrainEntry<Train> **extends** CommonPlanningEntry **implements** TrainPlanningEntry<Train>

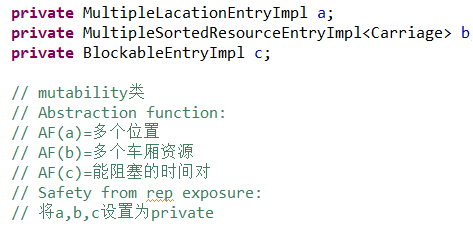
**public** **class** CourseEntry<Teacher> **extends** CommonPlanningEntry **implements** CoursePlanningEntry<Teacher>

4.然后在计划项中通过委托机制委托到外部每个维度的类实现功能：

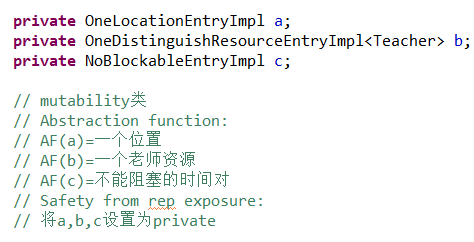
FlightEntry的字段：



TrainEntry的字段：

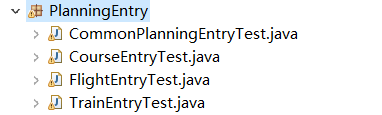


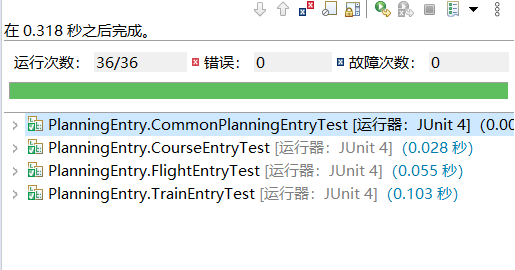
CourseEntry的字段：

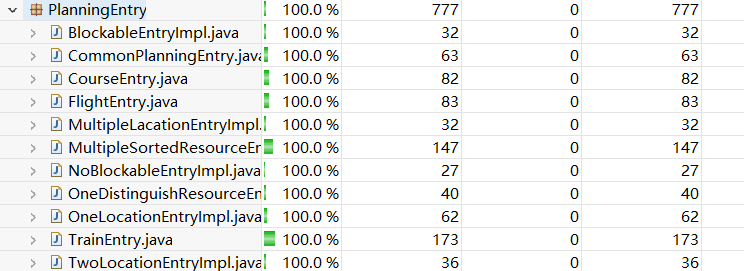


然后通过这些字段完成委托和功能的实现

2.进行测试：





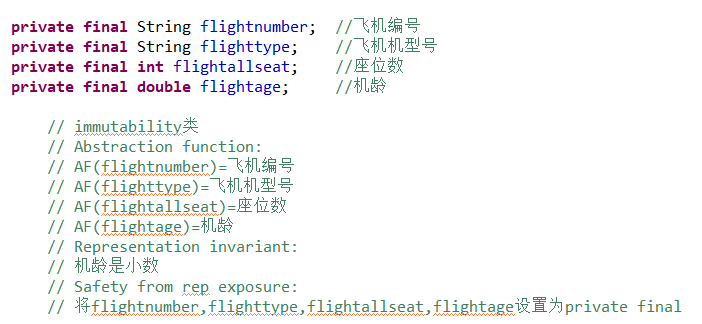


## 面向复用的设计：R

**3.3.1：飞机：**

航班应用中的资源是“飞机”，属性包括飞机编号、机型号（A350、B787、C919等）、座位数、机龄（例如2.5年）。

字段：

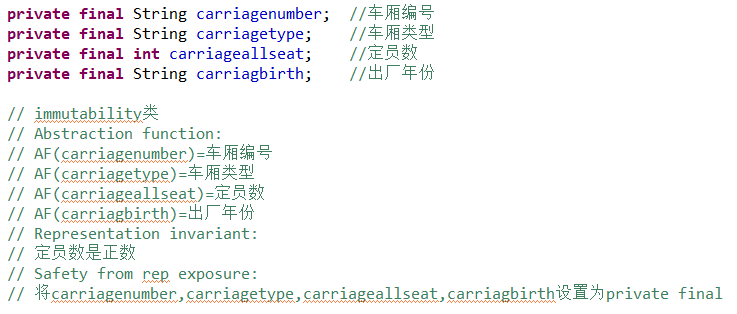


其方法：

|  |  |
| --- | --- |
| Flight | 初始化构造方法 |
| getflightnumber | 返回飞机编号 |
| getflighttype | 返回飞机机型号 |
| getflightallseat | 返回飞机座位数 |
| getflightage | 返回飞机机龄 |
| hashCode | 重写hashcode方法 |
| equals | 重写equals方法 |

**3.3.2：车厢：**

字段：

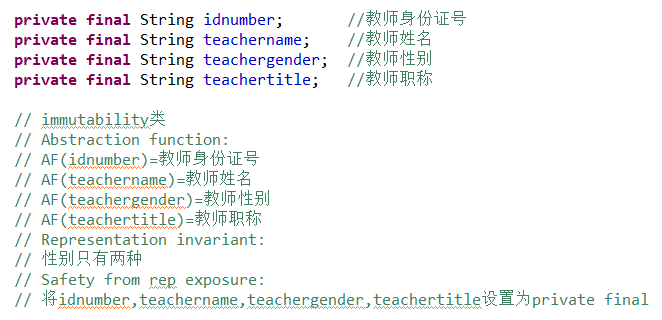


其方法：

|  |  |
| --- | --- |
| Carriage | 初始化构造方法 |
| getcarriagenumber | 返回车厢编号 |
| getcarriagetype | 返回车厢类型 |
| getcarriageallseat | 返回定员数 |
| getcarriagbirth | 返回车厢出厂年份 |
| hashCode | 重写hashcode方法 |
| equals | 重写equals方法 |

**3.3.3：教师：**

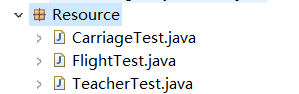
字段：

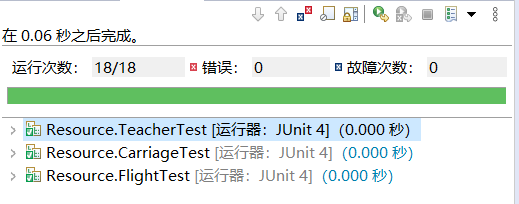


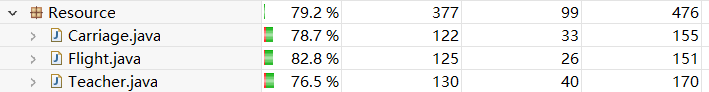
其方法：

|  |  |
| --- | --- |
| Teacher | 初始化构造方法 |
| getidnumber | 返回教师身份证号 |
| getteachername | 返回教师姓名 |
| getteachergender | 返回教师性别 |
| getteachertitle | 返回教师职称 |
| hashCode | 重写hashcode方法 |
| equals | 重写equals方法 |

**3.3.4：测试**







## 面向复用的设计：Location

**3.4.1：接口Location：**

定义一个接口Location，拥有几种位置的公有方法

**public** **interface** Location

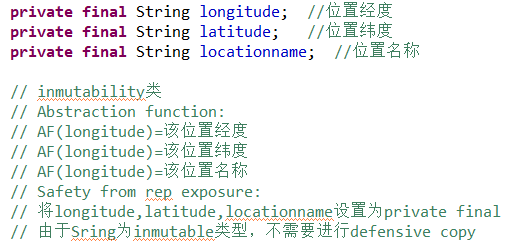
其方法：

|  |  |
| --- | --- |
| getlongitude | 得到该位置的的纬度 |
| getlatitude | 得到该位置的的经度 |
| getlocationname | 得到该位置的的名称 |

**3.4.2：接口的实现类CommonLocation：**

**public** **class** CommonLocation **implements** Location

字段：



相较接口，多了一个构造方法：CommonLocation

**3.4.3：教室位置的实现类CourseLocation：**

继承父类

**public** **class** CourseLocation **extends** CommonLocation

比父类多了一个whethershare()方法，判断该位置是否可共享

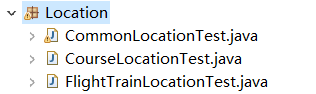
**3.4.4：火车站飞机场位置的实现类CourseLocation**

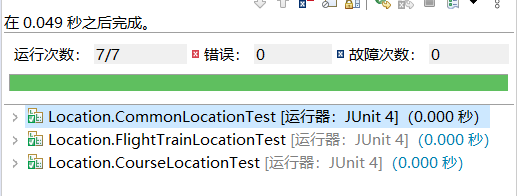
继承父类

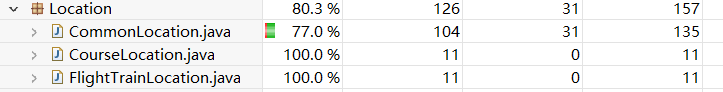
**public** **class** CourseLocation **extends** CommonLocation

比父类多了一个whethershare()方法，判断该位置是否可共享

**3.4.5：进行测试**



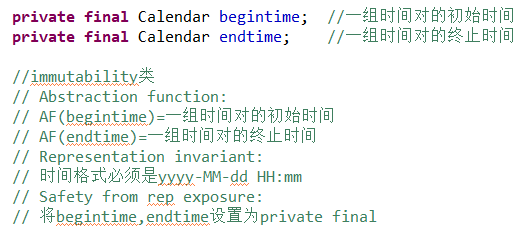




## 面向复用的设计：Timeslot

**public** **class** Timeslot

包含起始时间和终止时间两个字段：

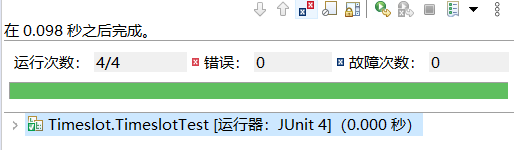


其方法：

|  |  |
| --- | --- |
| Timeslot | 构造方法 |
| getbegintime | 返回初始时间 |
| getendtime | 返回终止时间 |
| hashCode | 重写hashcode方法 |
| equals | 重写equals方法 |

进行测试如下：







## 面向复用的设计：EntryState及State设计模式

**3.6.1：设置三个State接口：**

航班状态接口：

**public** **interface** FlightState

其方法：

|  |  |
| --- | --- |
| move | 改变当前状态 |
| getflightstate | 得到当前状态 |

高铁状态接口：

**public** **interface** TrainState

其方法：

|  |  |
| --- | --- |
| move | 改变当前状态 |
| getflightstate | 得到当前状态 |

课程状态接口：

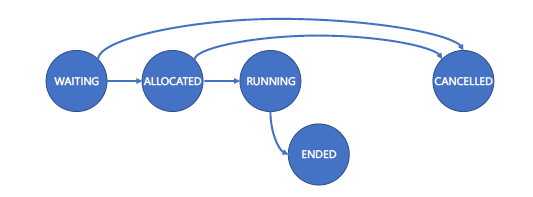
**public** **interface** CourseState

其方法：

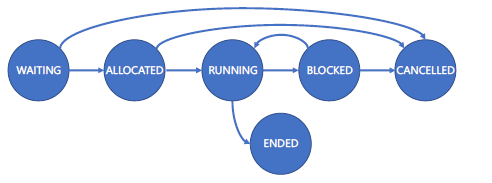
|  |  |
| --- | --- |
| move | 改变当前状态 |
| getflightstate | 得到当前状态 |

**3.6.2：针对每个接口进行实现：**

航班状态转换图和课程状态转换图如下：



高铁状态转换图如下：



因此可以根据这两个转换图分别实现不同子类的状态转换方法：

航班状态转换和课程状态转换具体转换关系为：

waiting：’a’->allocated; ‘b’->cancelled

allocated: ‘a’->running; ‘b’->cancelled

running: ‘a’->ended

高铁状态转换具体转换关系为：

waiting：’a’->allocated; ‘b’->cancelled

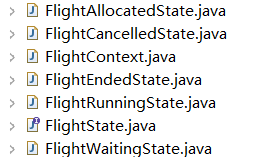
allocated: ‘a’->running; ‘b’->cancelled

running: ‘a’->ended; ‘b’->blocked

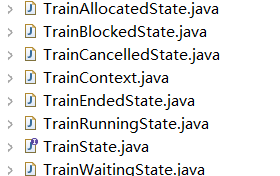
blocked: ‘a’->running; ‘b’->cancelled

根据这些转换关系可以实现各中状态类：

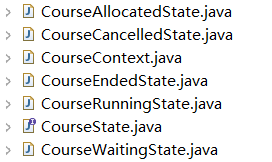
航班：



高铁：



课程：



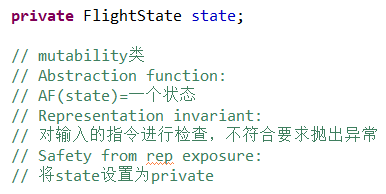
**3.6.3：设置三个Context类：**

Context类用来调用各种状态类的转换：

以FlightContext为例：

**public** **class** FlightContext

字段：

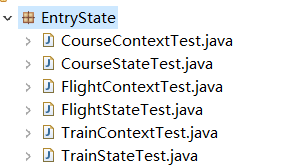


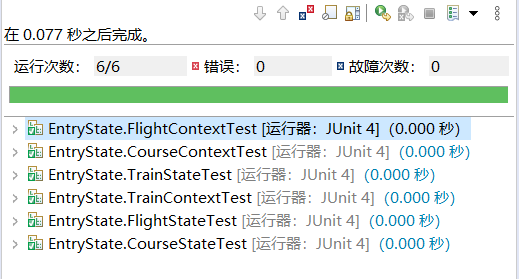
其方法：

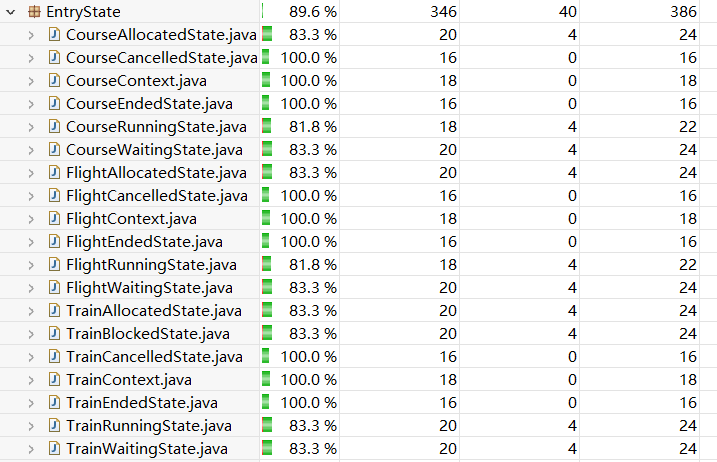
|  |  |
| --- | --- |
| FlightContext | 构造方法 |
| move | 改变当前状态 |
| getstate | 得到当前状态 |

TrainContext和CourseContext同理

**3.6.4：测试：**







## 面向应用的设计：Board

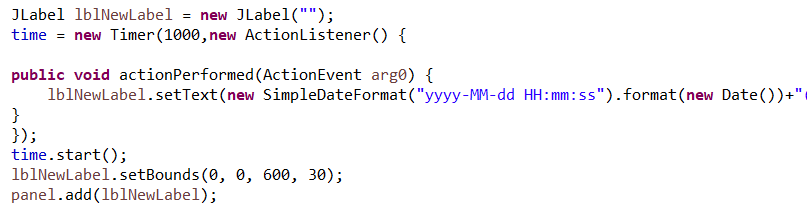
使用下载的插件WindowsBuilder来设计框架：

新建->其他->Swing Designer->Jframe创建一个Jframe即可：

然后根据报告要求设计框架即可：

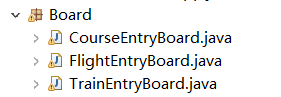
然后就能自动生成对应的代码：

因为要求实时显示时间，在查找相关资料后，通过：

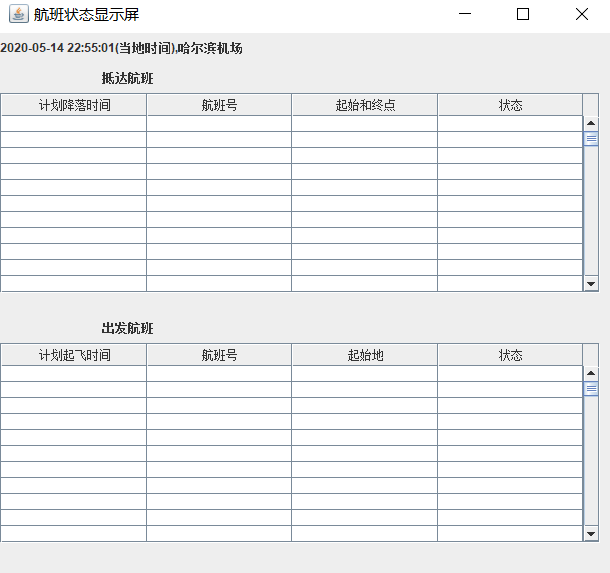


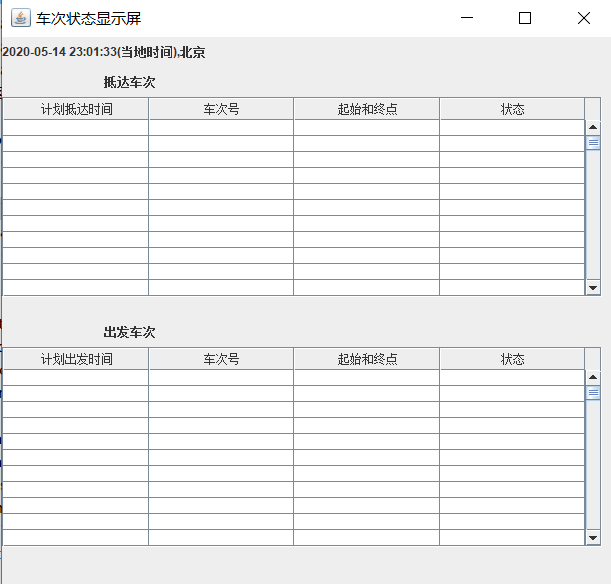
就能够显示电脑的时间，

分别设计三个Board：



显示出来效果如下：







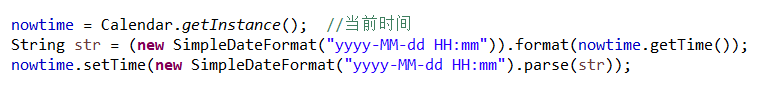
## Board的可视化：外部API的复用

1.对于FlightEntryBoard：

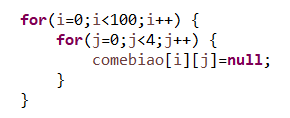
分为抵达屏和出发屏，前者展示过去1 小时内和未来1 小时内抵达该机场的航班信息及其状态，后者展示过去1 小时内和未来1 小时内从该机场出发的航班信息及其状态。注意：第一列里的所有时间都是计划降落时间（对抵达航班来说）、计划起飞时间（对出发航班来说）。

为了实现这种功能：

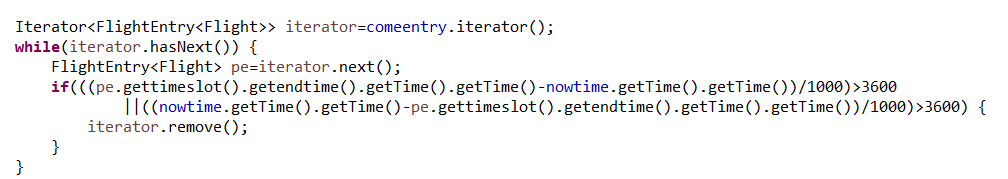
对时间进行从小到大排序在后面Iterator中再讲到，首先我们看如何限定一小时这个条件，具体实现如下：

对于抵达屏，首先用得到当前时间。

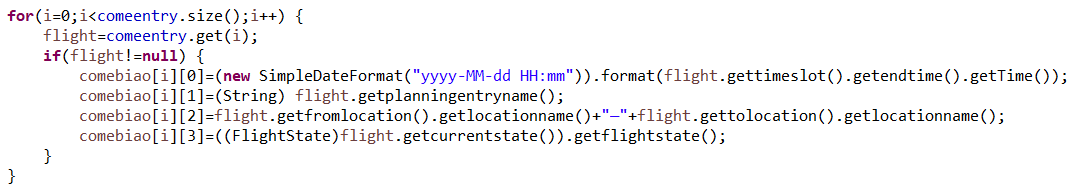
初始化要显示的所有信息：



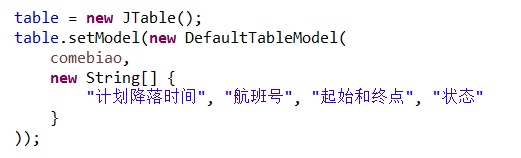
对于抵达航班来说，需要判断抵达时间与当前时间差的绝对值与1小时的关系，除去超过一小时的所有计划项



然后将计划项的信息填入初始化的数组中：



然后显示该数组中内容即可：



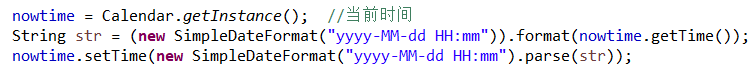
出发屏同理。

2.对于TrainEntryBoard：与FlightEntryBoard同理：

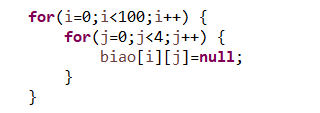
唯一不同的是最后一个站点没有出发车次，第一个站点没有抵达车次，需要除去

3.对于CourseEntryBoard：需要列出当日在本教室内上的所有课程及其状态：

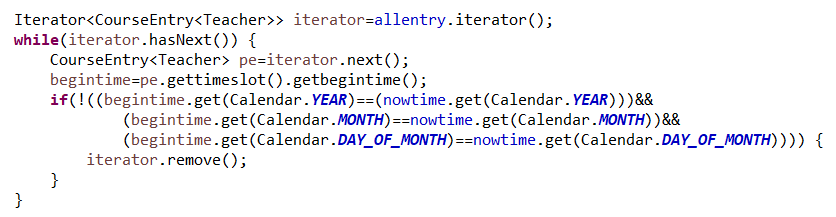
为了得到当前日期的所有课程，我们同样需要得到当前时间：



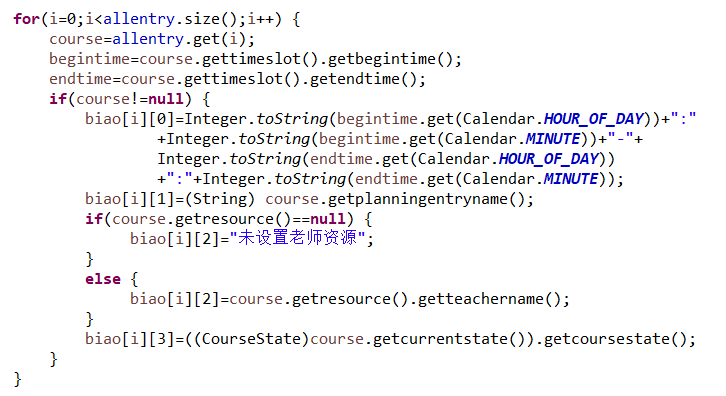
初始化需要显示的信息：



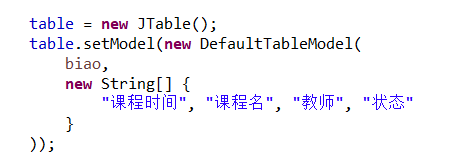
然后通过判断年月日是否相同进行排除：



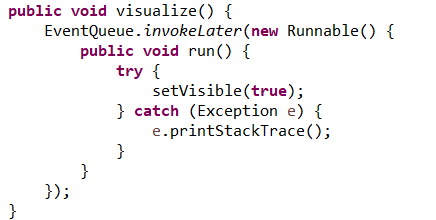
然后将计划项的信息填入初始化的数组中：



然后显示数组中的内容即可：



4.完成的visualize方法：



## 可复用API设计及Façade设计模式

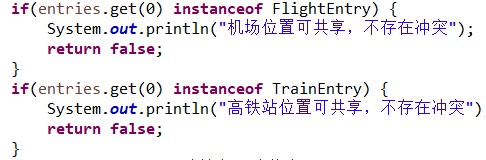
需要设计三个API放入一个PlanningEntryAPIs类中

### 检测一组计划项之间是否存在位置独占冲突

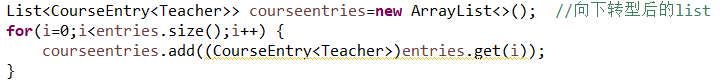
**public** **boolean** checkLocationConflict(List<PlanningEntry> entries)

1.由于PlanningEntry是一个接口，无法判断其类型，所以我利用instanceof进行判断其具体类型，其中只有CourseEntry能够存在位置独占冲突：





2.然后通过向下转型来完成功能：

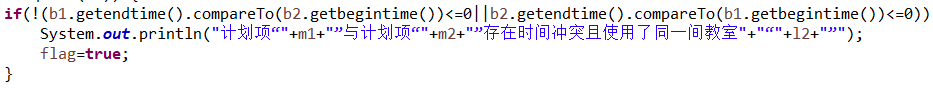


3.然后具体实现：

实现思路为：将所有计划项从开始进行遍历，没遍历一个计划项，将该计划项与其后的所有计划项进行比较，如果找到两个计划项占用了相同的地址，然后就对他们的Timeslot进行比较，若存在重叠，则证明两个计划项存在位置独占冲突。

4.检查两个计划项时间是否重叠的方法为：

第一个的结束时间小于第二个的起始时间或者第二个的结束时间小于第一个的结束时间，则不存在冲突。除去不存在时间冲突的情况后，则必然存在时间冲突。

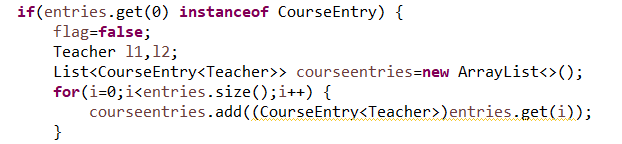


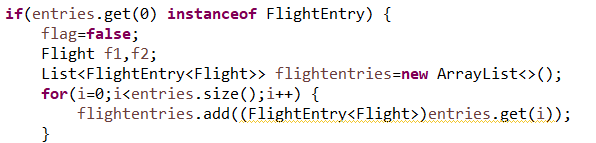
### 检测一组计划项之间是否存在资源独占冲突

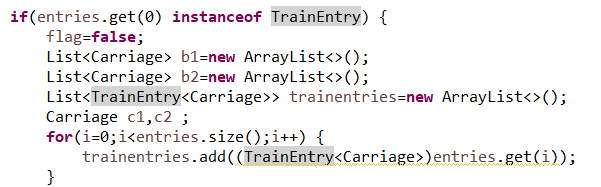
**public** **boolean** checkResourceExclusiveConflict(List<PlanningEntry> entries)

1.主要思路和时间独占冲突类似，只是把是否占用同样的位置改成了是否占用同样的资源。用同样的思路判断是否用相同的资源，然后对拥有相同资源的两个计划项进行是否存在时间冲突的判断。

2.由于航班，高铁，课程均存在资源独占冲突，所以需要通过三次向下转型完成判断：

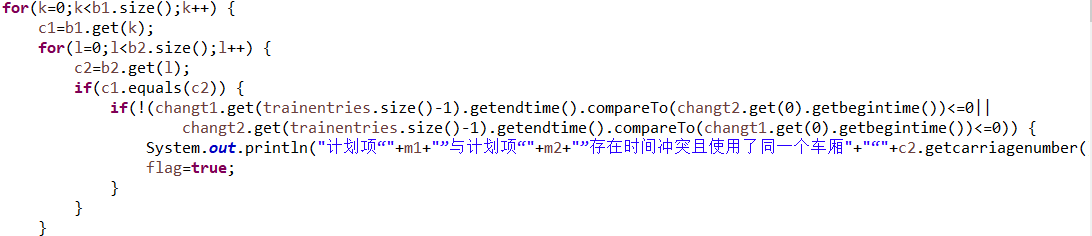






3.课程和航班的判断方式类似，但是高铁的资源是车厢，判断就更复杂，同样是遍历每个计划项，但在比较两个计划项的资源时，需要继续遍历资源，从而找到两个相同的车厢。

对于相同的车厢，需要判断两个计划项Timeslot的集合的初始时间与到达时间，而不是简单的一个Timeslot的时间判断。



### 提取面向特定资源的前序计划项

**public** PlanningEntry findPreEntryPerResource(R r,PlanningEntry e,List<PlanningEntry> entries)

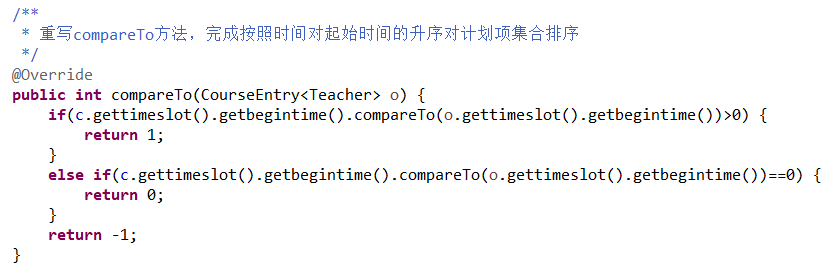
1.三种计划项都需要向下转型

2.要提取某计划项的前序计划项，则需要得到一个有序初始时间递增的计划项集合，为了实现这种功能，我选择了在每个计划项类重写Compareble里面的compareTo方法

然后调用Collections的sort方法即可完成排序：

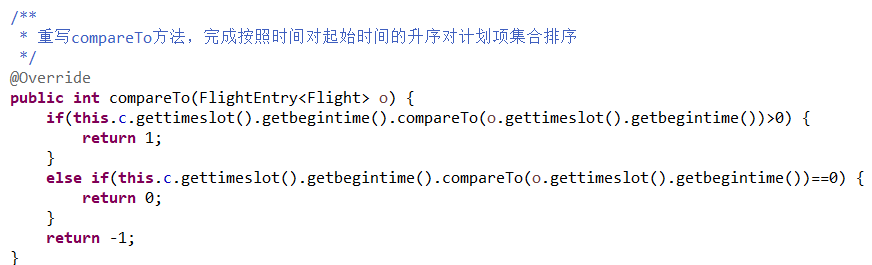
课程：





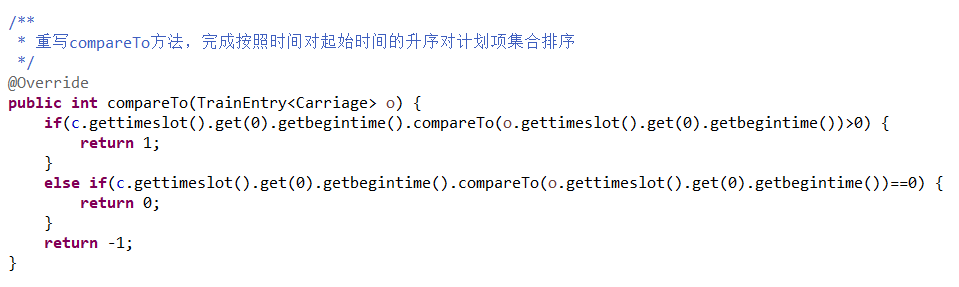
航班：





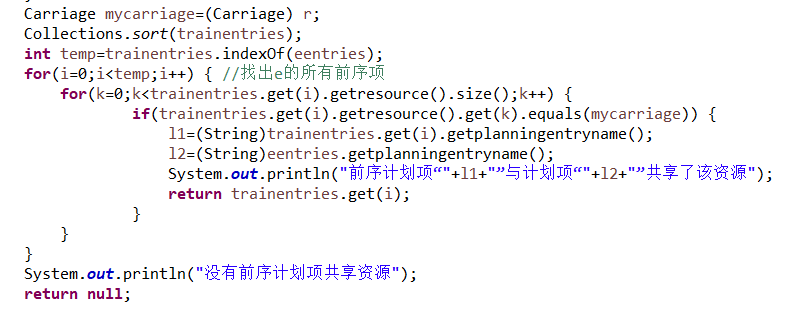
高铁：





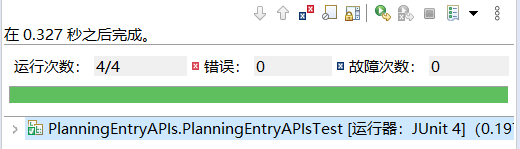
3.对于给定计划项的资源，先排序计划项集合，然后只需要遍历计划项集合到该计划项位置，找到和其资源相同的计划项并返回即可

4.对于高铁的同上，需要多一次内部车厢的遍历：



**测试：**







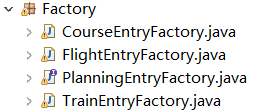
## 设计模式应用

### Factory Method

1.设计

要求我们使用factory method模式，使得五个应用中创建具体PlanningEntry子类型对象时，尽可能避免通过new的方式实例化具体的PlanningEntry。

所以我设计了以下类：



分别为

接口**public** **interface** PlanningEntryFactory，里面含有创建三种计划项的方法

创建航班计划项的类**public** **class** FlightEntryFactory **implements** PlanningEntryFactory，对其中创建航班计划项的方法进行实现，另外两种方法返回null



创建高铁计划项的类**public** **class** TrainEntryFactory **implements** PlanningEntryFactory，对其中创建高铁计划项的方法进行实现，另外两种方法返回null



创建课程计划项的类**public** **class** CourseEntryFactory **implements** PlanningEntryFactory，对其中创建课程计划项的方法进行实现，另外两种方法返回null



2.具体应用：

创建航班计划项时：



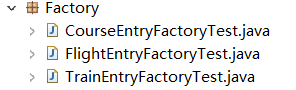
创建高铁计划项时：

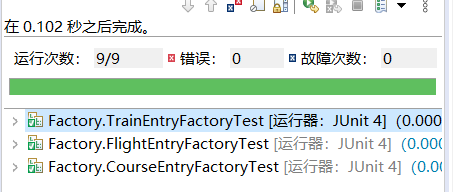


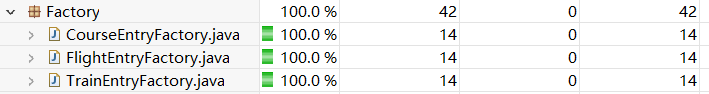
创建课程计划项时：



3.测试：







### Iterator

1.要求：为Board类增加迭代器功能，可通过迭代器按照时间从早到晚的次序遍历其中包含的所有计划项。

2.实现：

用迭代器依次遍历所有的计划项，除去与该位置不同的计划项，然后在每一个迭代器方法里面重写compare方法，调用Collections的sort方法，即可完成时间从早到晚排序所有计划项

2.1 FlightEntryBoard:

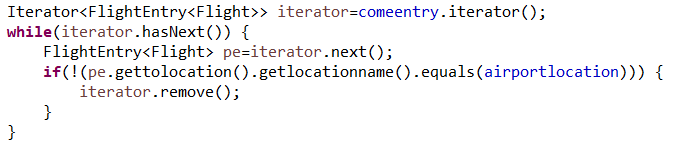
该board中有两个完成排序的计划项集合，一个是航班按照计划项的抵达时间从早到晚排序，一个是按照航班计划项的出发时间从早到晚排序：

具体实现为：

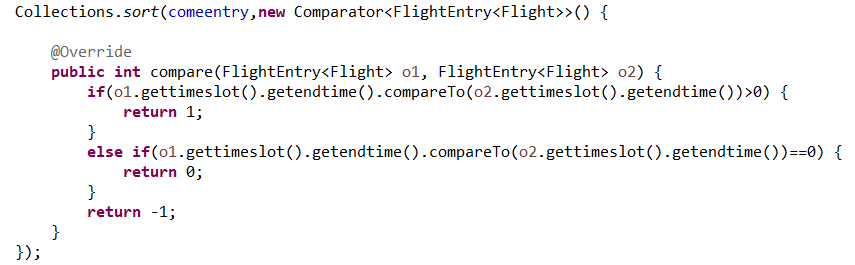
首先将原计划项集合赋值给一个新的计划项集合，目的是为了防止改变了原计划项集合：



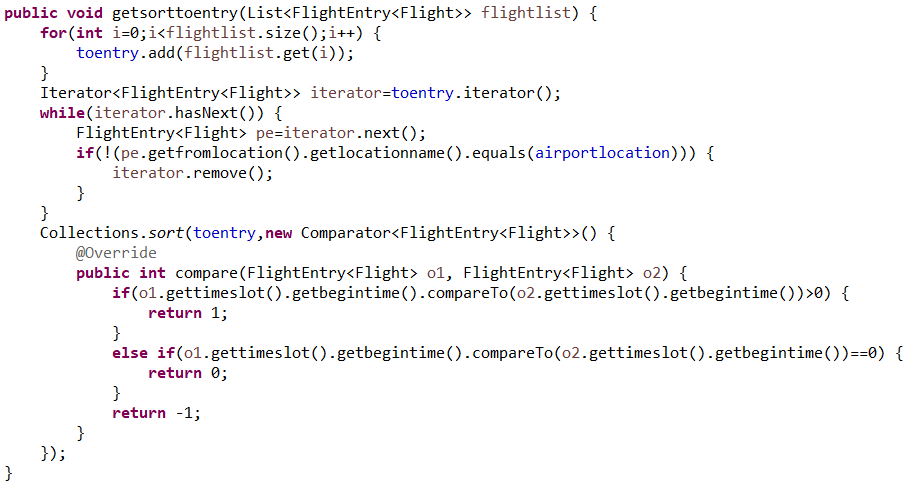
然后对计划项用iterator进行遍历，除去与该位置不是一个位置的计划项



然后重写compare方法，调用Collections的sort方法完成对计划项按照抵达时间从早到晚排序：



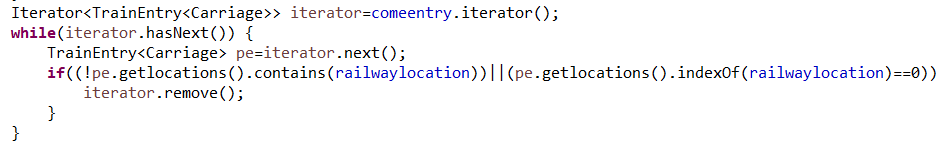
出发时间的排序同理，只需要把比较的时间改为计划项的出发时间即可：



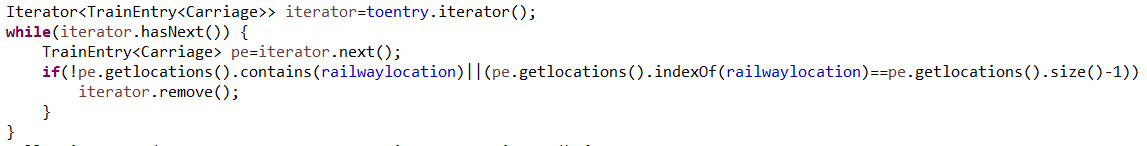
2.2 TrainEntryBoard

与FlightEntryBoard类似，都需要实现两个迭代器的排序功能，但是有不同。

对于抵达时间的iterator，在除去的计划项中除了除去与该位置不同的计划项外，还需要除去第一个位置的计划项，因为第一个位置的计划项没有抵达时间：



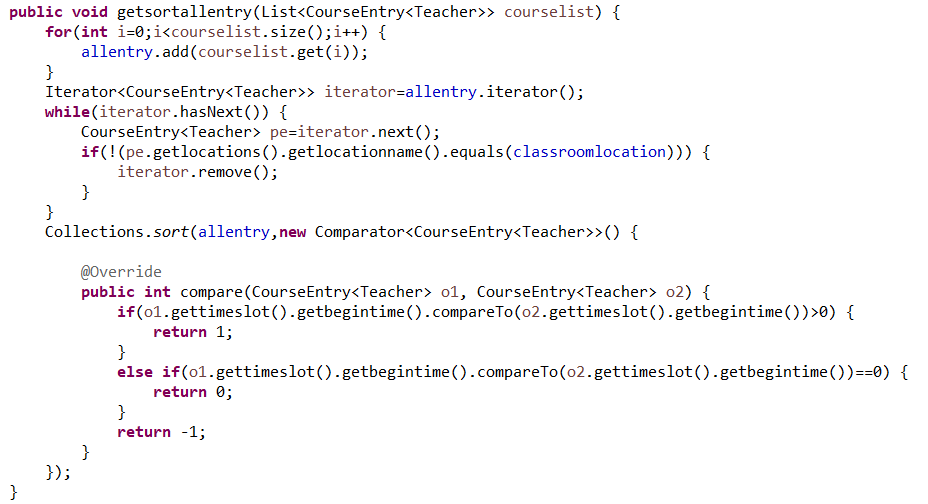
对于出发时间的iterator，在除去的计划项中除了除去与该位置不同的计划项外，还需要除去最后一个位置的计划项，因为最后一个位置的计划项没有抵达时间：



对于重写compare方法，只需要找到对应位置的计划项对应的时间进行比较即可。

2.3 CourseEntryBoard

与FlightEntryBoard几乎一样，只需要将计划项改成CourseEntry即可，但只需要按照课程的起始时间完成iterator的排序即可：



### Strategy

1. 要求：

在设计中，请选定其中一个API，使用两种不同的算法实现之，进而在客户端应用程序中使用strategy模式灵活替换不同的算法。

2. 接口及实现类：

我选择了用两种算法实现检查位置独占冲突的方法，应用策略模式进行设计：

首先创建一个方法接口：

**public** **interface** StrategycheckLocationConflict

在里面定义该方法：

**public** **boolean** checkLocationConflict(List<PlanningEntry> entries)

然后创建两个该方法的实现类：

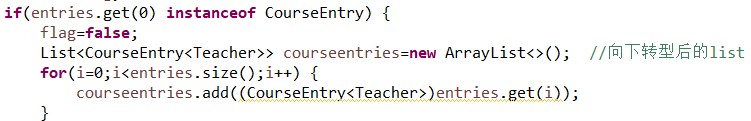
**public** **class** Strategy1checkLocationConflict **implements** StrategycheckLocationConflict

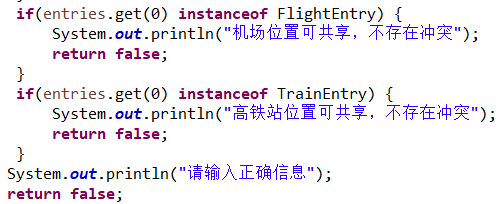
**public** **class** Strategy2checkLocationConflict **implements** StrategycheckLocationConflict

在两个方法实现类里面分别用不同的算法实现查找位置独占冲突的方法：

2.1方法1：

向下转型在上面已经提到过，且对于航班和高铁计划项不存在位置独占冲突：





对于具体算法实现：

将所有计划项从开始进行遍历，没遍历一个计划项，将该计划项与其后的所有计划项进行比较，如果找到两个计划项占用了相同的地址，然后就对他们的Timeslot进行比较，若存在重叠，则证明两个计划项存在位置独占冲突。

其中检查两个计划项时间是否重叠的方法为：

第一个的结束时间小于第二个的起始时间或者第二个的结束时间小于第一个的结束时间，则不存在冲突。除去不存在时间冲突的情况后，则必然存在时间冲突。

2.2方法2

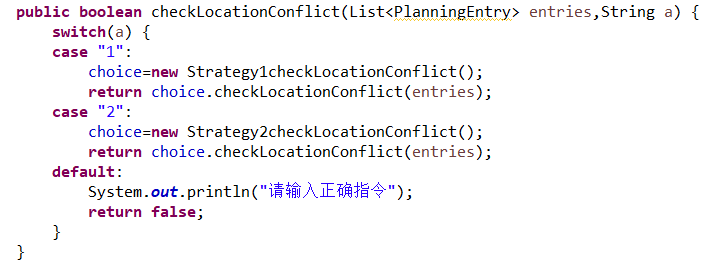
对于具体算法实现：

创建一个存储位置的集合，通过遍历得到所有的位置并储存，然后每次创建一个临时的计划项集合，将所有位置相同的计划项存储在该集合中（类似等价类划分，拥有同样位置的计划项看做一个整体进行划分）。然后创建一个集合，用来存储这些整体。然后在每一组整体里面进行遍历，如果整体集合的长度超过一，就用类似于方法1的方式判断两个计划项是否存在时间重叠，若存在，则输出并返回true。

3.原PlanningEntryAPIs中的改动：

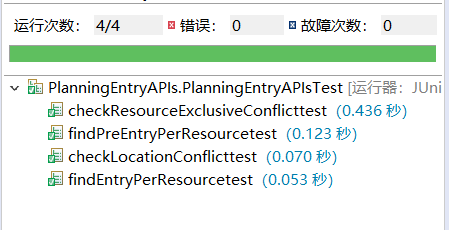
在原PlanningEntryAPIs对检查位置独占冲突的方法进行修改，增加选择的方法。首先定义一个StrategycheckLocationConflict接口的字段，然后根据输入选择不同的类来实现这种方法，完成Strategy策略模式。

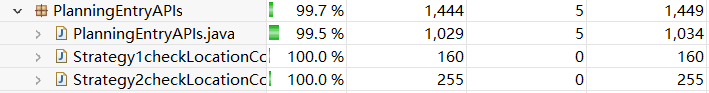




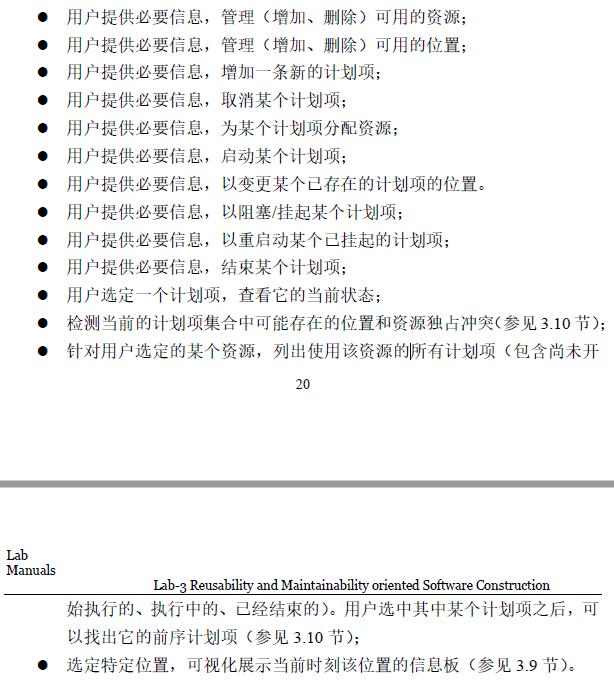
4.测试：







## 应用设计与开发



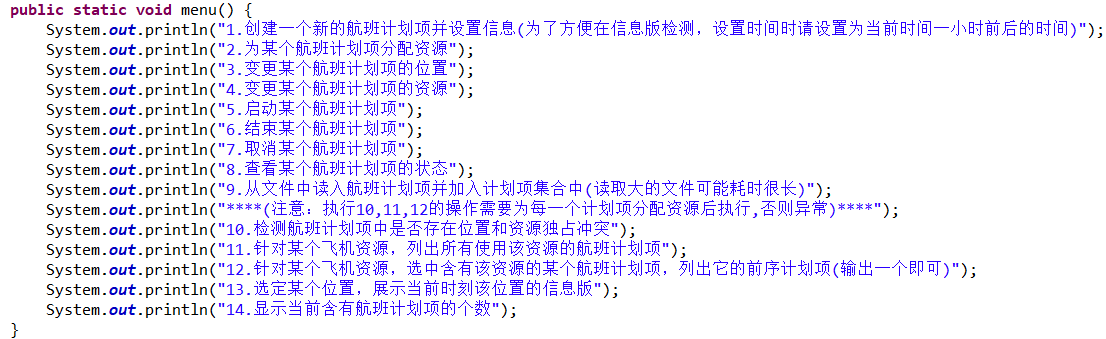
以上是要求实现的所有功能，在不同的应用中，功能可能不同，针对不同应用实现其能完成的功能（注意取消计划项需要显示取消的时间）

### 航班应用

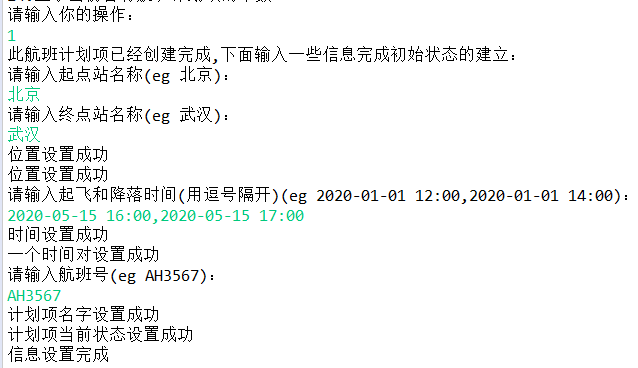
1. **public** **class** FlightScheduleApp

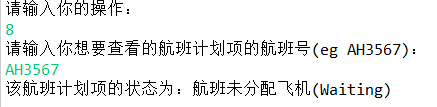
其字段：**private** **static** List<FlightEntry<Flight>> *flightlist*=**new** ArrayList<>(); //储存所有FlightEntry的集合

菜单：

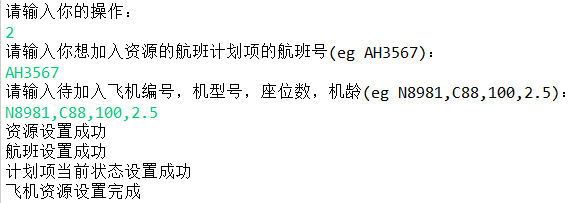


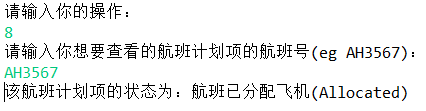
2.具体实现：  
2.1：创建一个新的航班计划项并设置信息：



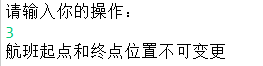


2.2为某个航班计划项分配资源(为了方便在信息版检测，设置时间时请设置为当前时间一小时前后的时间)：

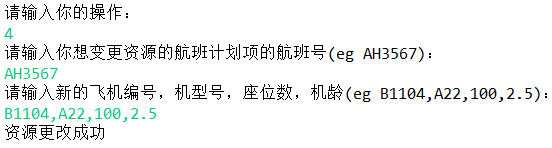




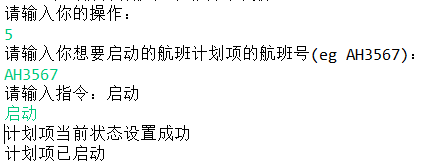
2.3变更某个航班计划项的位置：

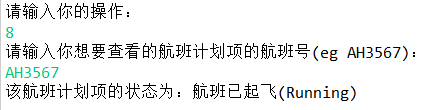


2.4变更某个航班计划项的资源：

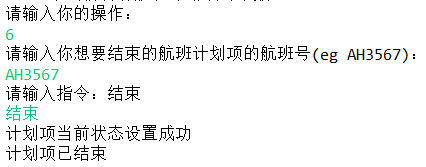


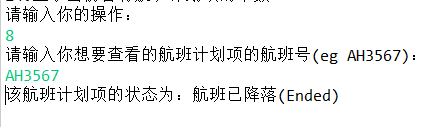
2.5启动某个航班计划项：





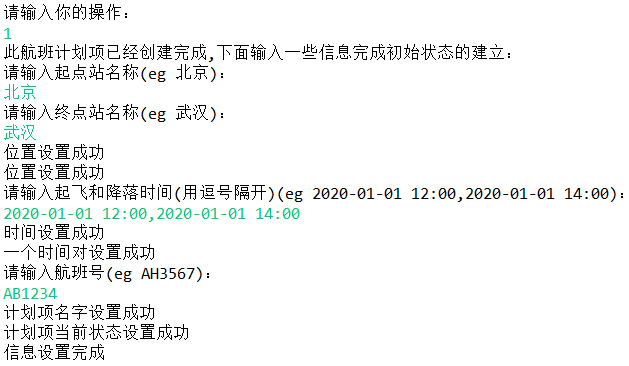
2.6结束某个航班计划项：

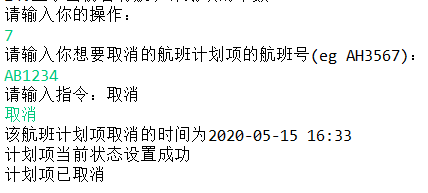




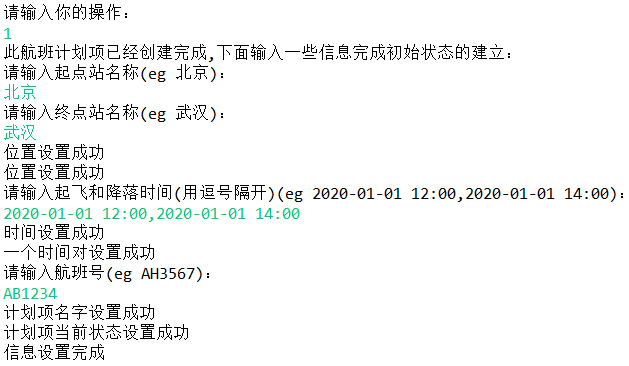
2.7 取消某个计划项

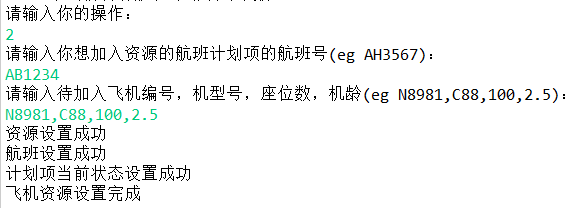
重新运行程序，创建一个新的计划项：(在waiting时取消)

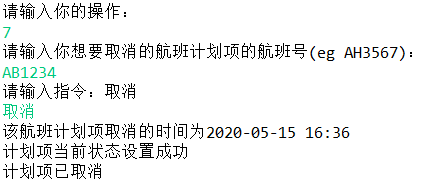




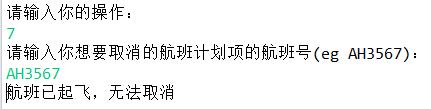
重新运行程序，创建一个新的计划项：(在Allocated时取消)







重新运行程序，创建一个新的计划项：(在其他状态时取消)



2.8 查看某个航班计划项的状态：

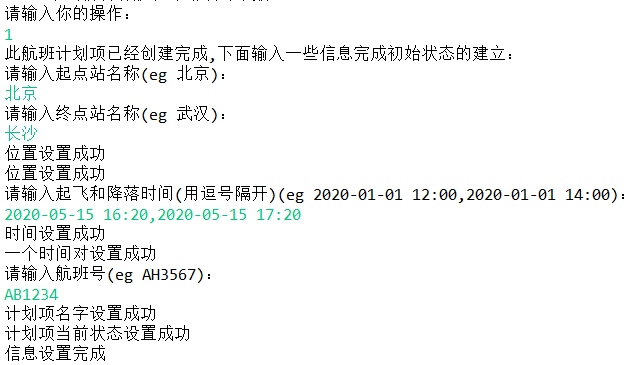
上面已显示

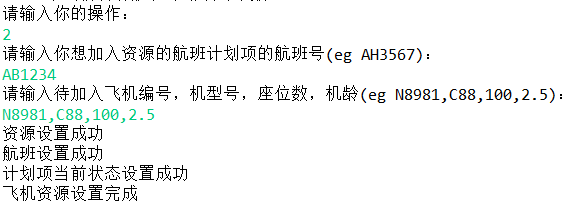
2.9 从文件中读入航班计划项并加入计划项集合中(读取大的文件可能耗时很长):

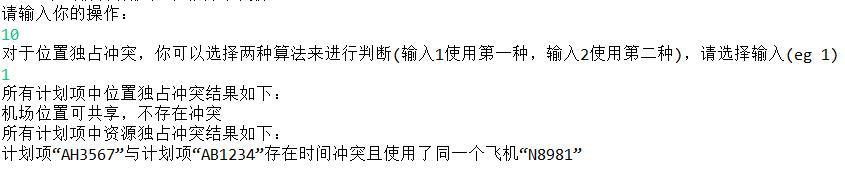
在下面3.12节详细讲述

2.10检测航班计划项中是否存在位置和资源独占冲突高铁应用：

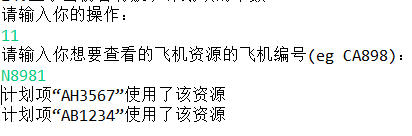
再创建一个具有冲突的新的计划项：



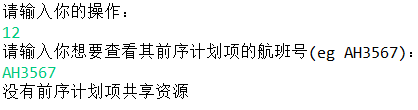


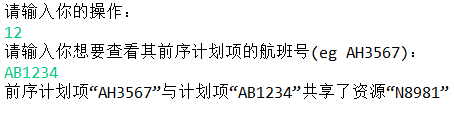


2.11针对某个飞机资源，列出所有使用该资源的航班计划项：



2.12针对某个飞机资源，选中含有该资源的某个航班计划项，列出它的前序计划项(输出一个即可)：





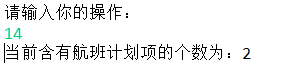
2.13选定某个位置，展示当前时刻该位置的信息版：







2.14显示当前含有航班计划项的个数：

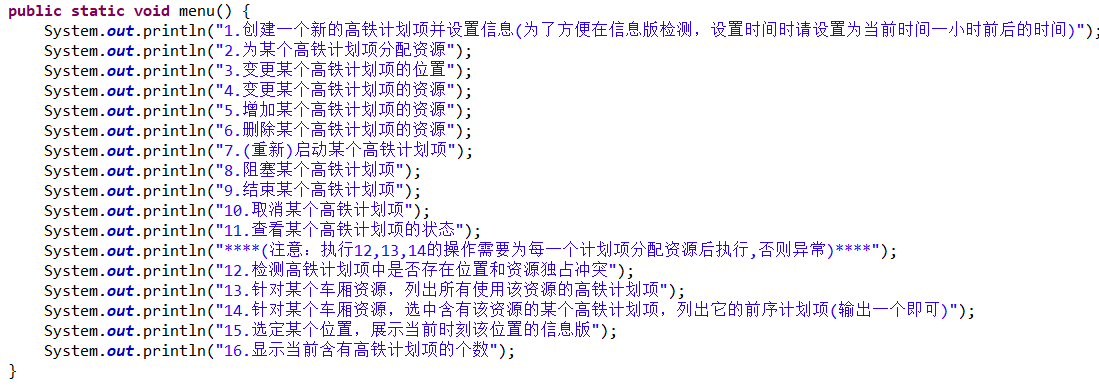


### 高铁应用

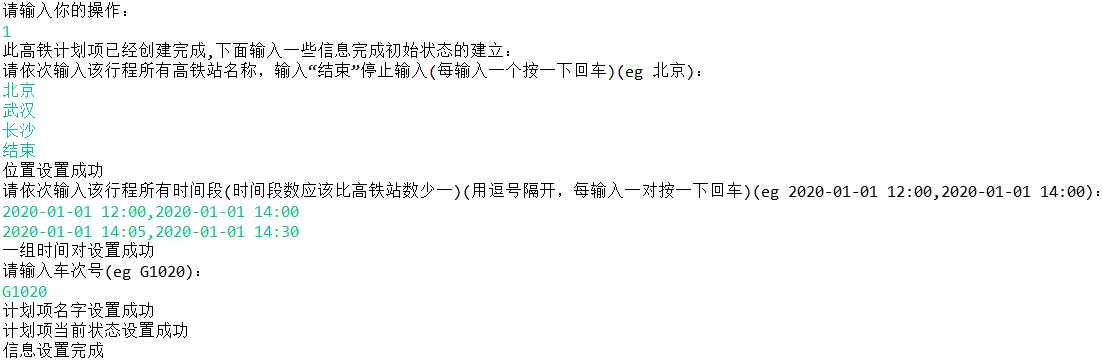
1. **public** **class** TrainScheduleApp

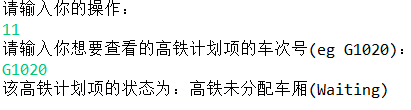
其字段：**private** **static** List<TrainEntry<Carriage>> *trainlist*=**new** ArrayList<>(); //储存所有TrainEntry的集合

菜单：

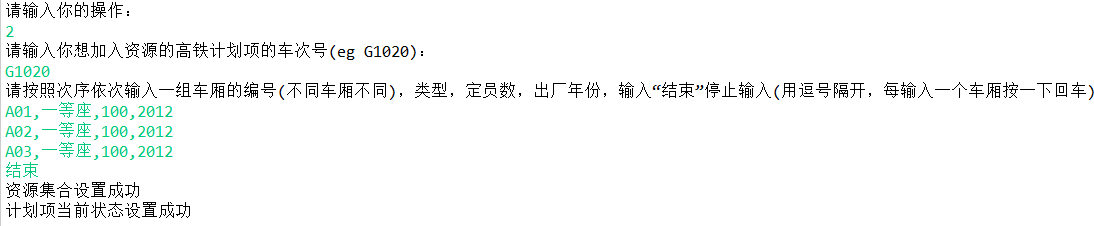


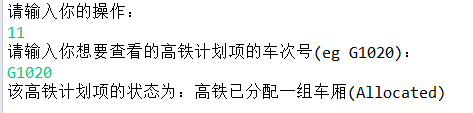
2.具体实现：  
2.1：创建一个新的高铁计划项并设置信息：



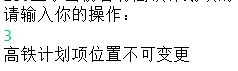


2.2为某个高铁计划项分配资源：

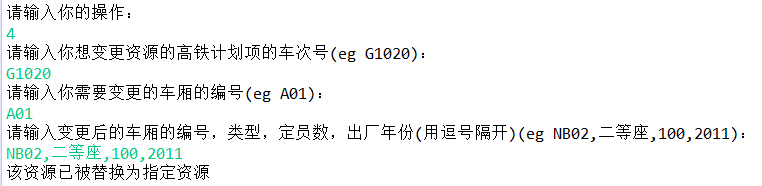


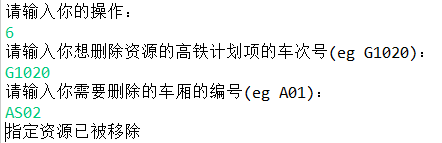


2.3变更某个高铁计划项的位置：

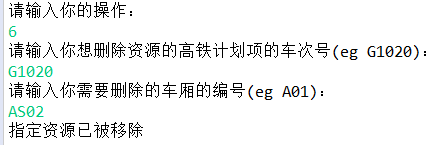


2.4变更某个高铁计划项的资源：

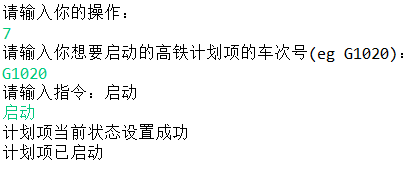


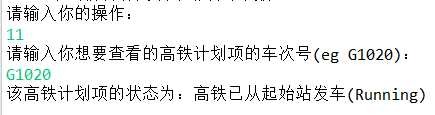
2.5增加某个高铁计划项的资源:  


2.6删除某个高铁计划项的资源

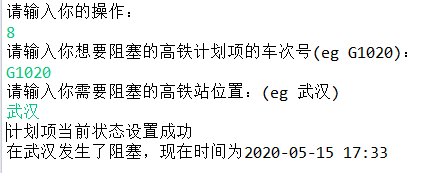


2.7(重新)启动某个高铁计划项



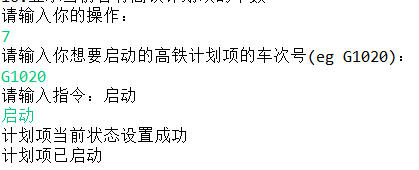


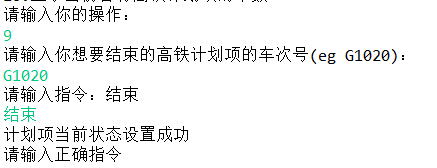
2.8阻塞某个高铁计划项：

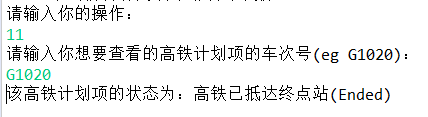


2.9结束某个高铁计划项：

先回到Running状态，再结束：

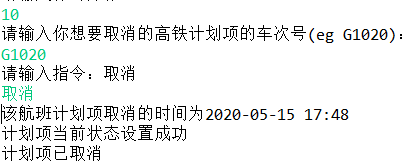




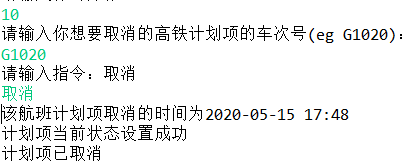


2.10取消某个高铁计划项：

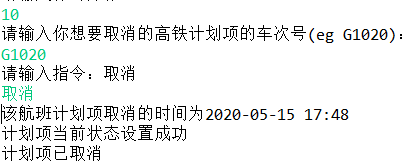
重新运行程序，创建一个新的计划项：(在waiting时取消)：



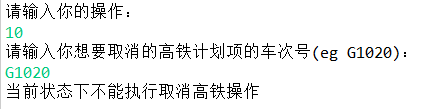
重新运行程序，创建一个新的计划项：(在Allocated时取消)：



重新运行程序，创建一个新的计划项：(在waiting时取消)：



重新运行程序，创建一个新的计划项：(在其他状态时取消)：

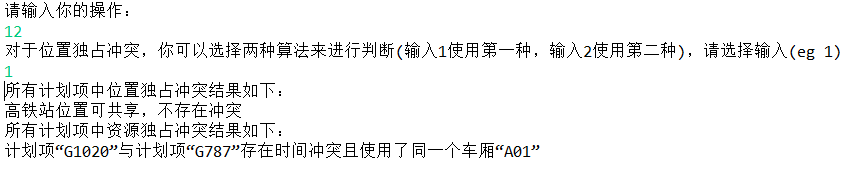


2.11查看某个高铁计划项的状态：

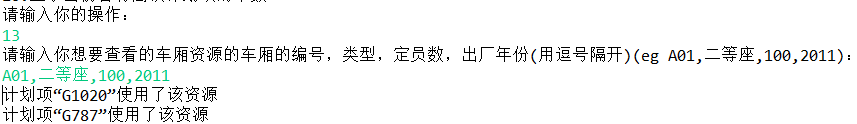
在上面已经查看

2.12检测高铁计划项中是否存在位置和资源独占冲突：

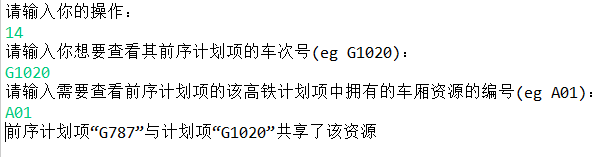
重新运行程序，创建两个存在独占冲突新的计划项：

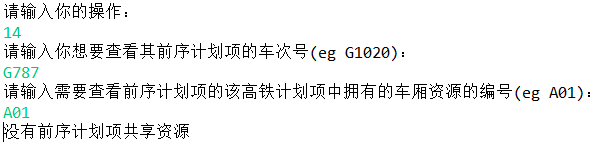


2.13针对某个车厢资源，列出所有使用该资源的高铁计划项：



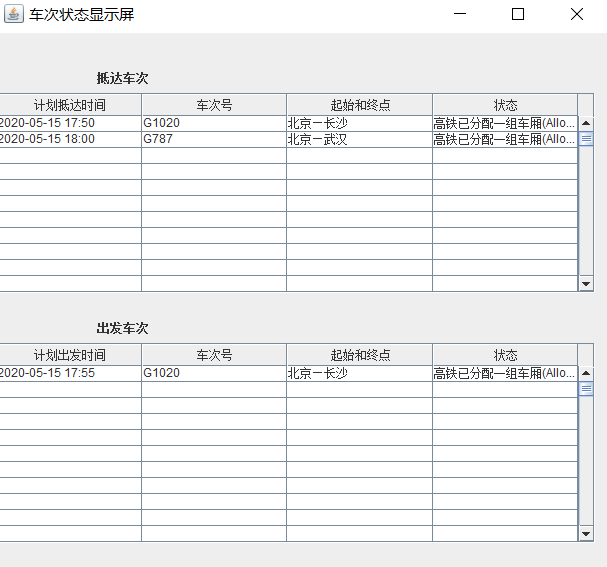
2.14针对某个车厢资源，选中含有该资源的某个高铁计划项，列出它的前序计划项(输出一个即可)：

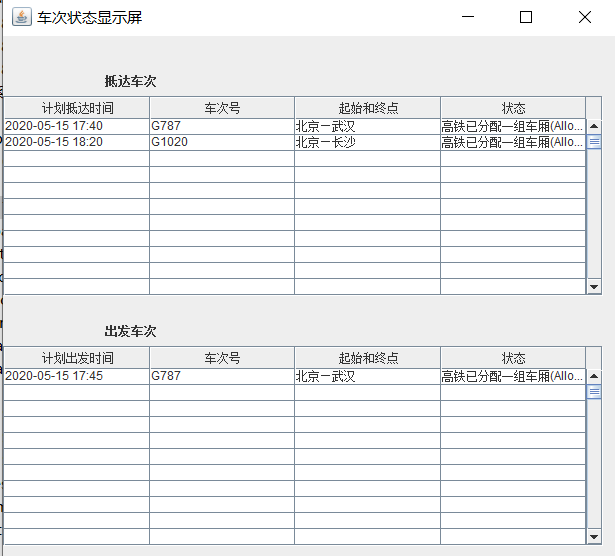




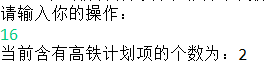
2.15选定某个位置，展示当前时刻该位置的信息版：







2.16显示当前含有高铁计划项的个数：

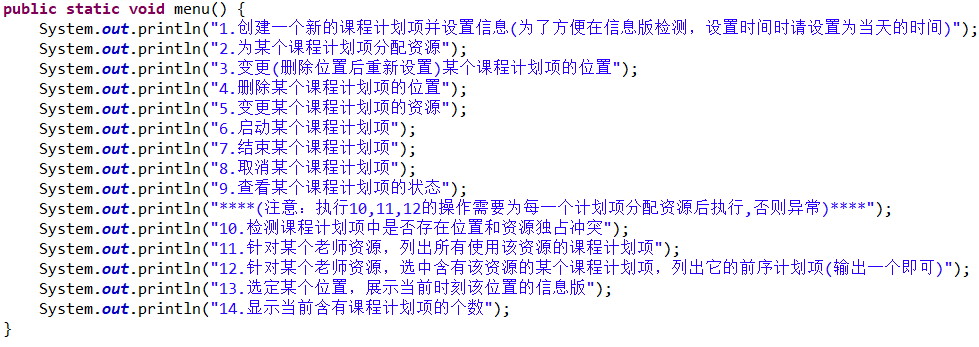


### 课表应用

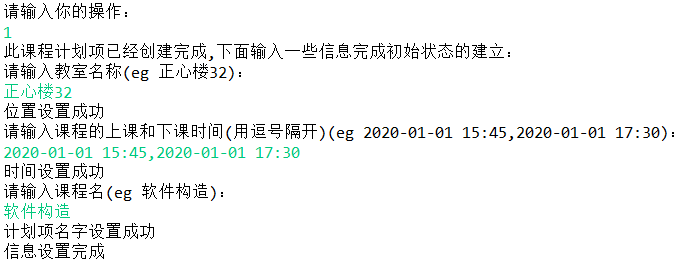
1. **public** **class** CourseCalendarApp

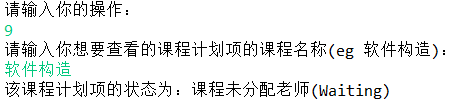
其字段：**private** **static** List<CourseEntry<Teacher>> *courselist*=**new** ArrayList<>(); //储存所有CourseEntry的集合

菜单：

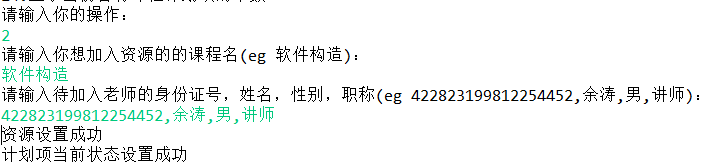


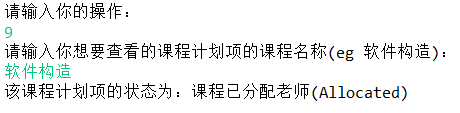
2.具体实现：  
2.1创建一个新的课程计划项并设置信息：



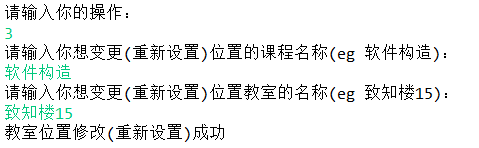


2.2为某个课程计划项分配资源：

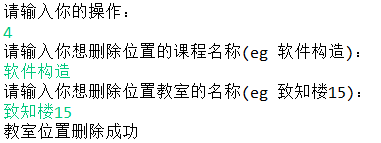




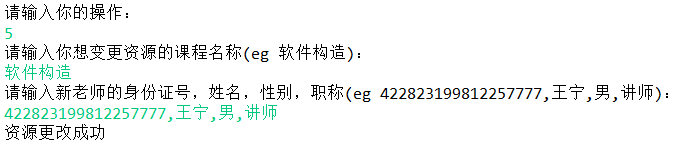
2.3变更(删除位置后重新设置)某个课程计划项的位置：



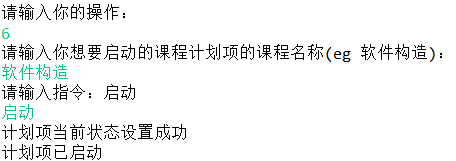
2.4删除某个课程计划项的位置：

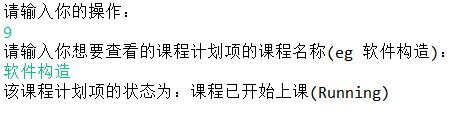


2.5变更某个课程计划项的资源

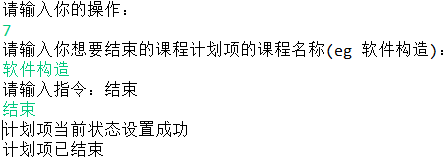


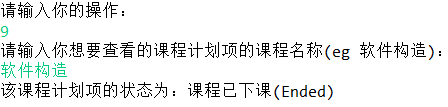
2.6启动某个课程计划项：





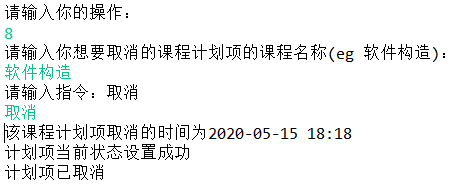
2.7结束某个计划项：



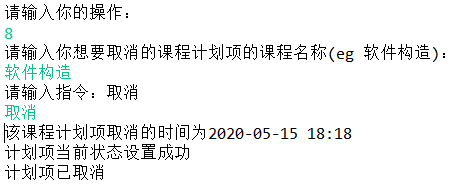


2.8取消某个计划项：

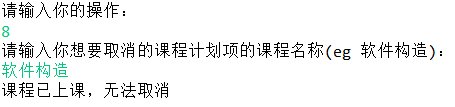
重新运行程序，创建一个新的计划项：(在waiting时取消)



重新运行程序，创建一个新的计划项：(在Allocated时取消)



重新运行程序，创建一个新的计划项：(在其他状态取消)

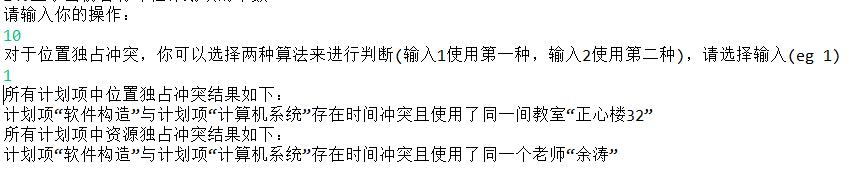


2.9 查看某个课程计划项的状态：

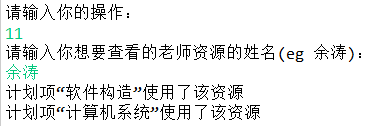
在上面已经展示

2.10 检测课程计划项中是否存在位置和资源独占冲突

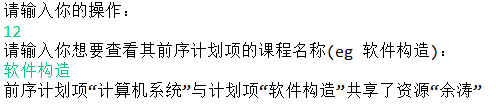
重新运行程序，创建两个存在独占冲突新的计划项：

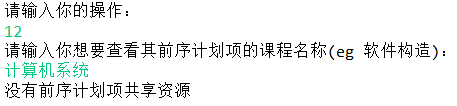


2.11针对某个老师资源，列出所有使用该资源的课程计划项：

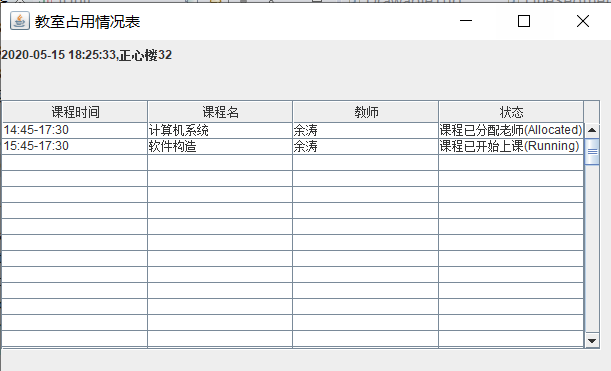


2.12针对某个老师资源，选中含有该资源的某个课程计划项，列出它的前序计划项(输出一个即可)：

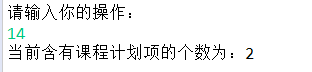




2.13选定某个位置，展示当前时刻该位置的信息版：



2.14显示当前含有课程计划项的个数：

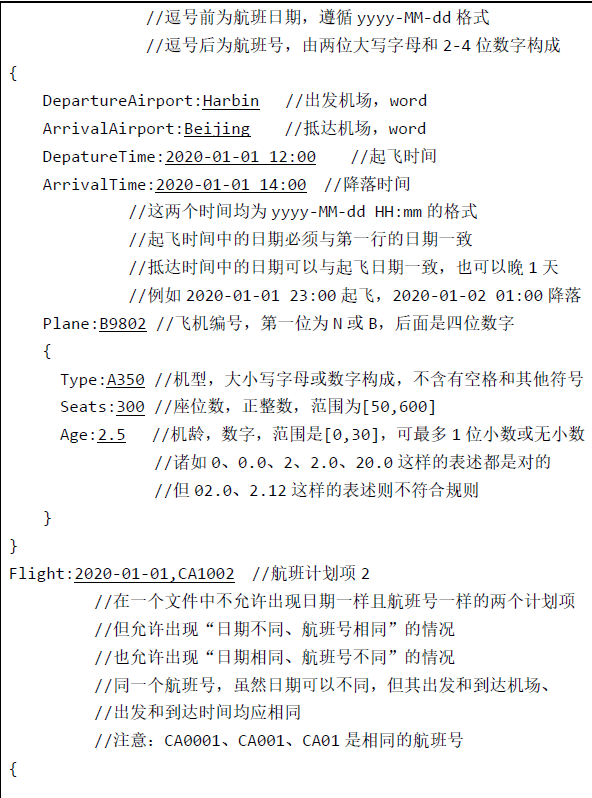


### 学习活动应用

## 基于语法的数据读入

根据实验手册，我们需要文件中的计划项数据满足以下特征：

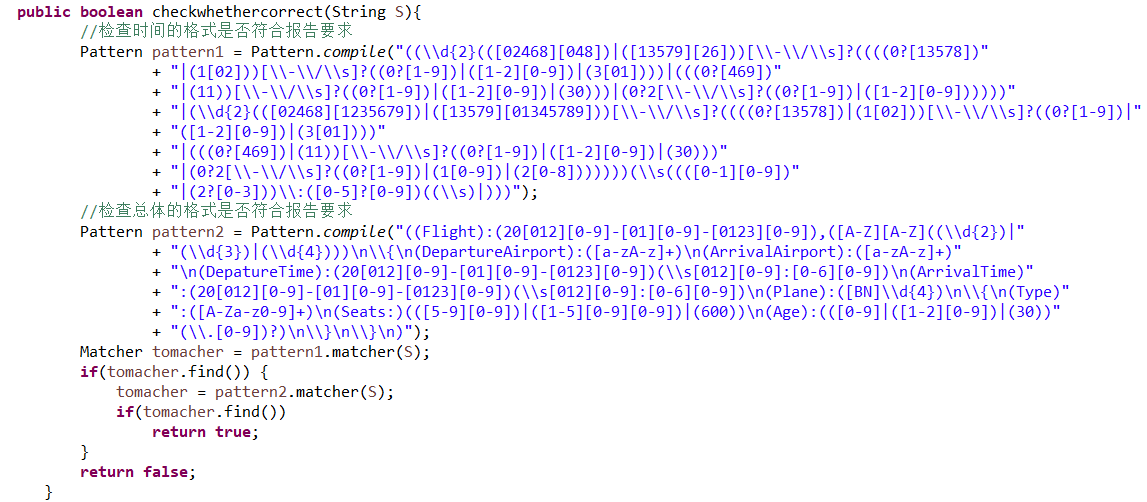




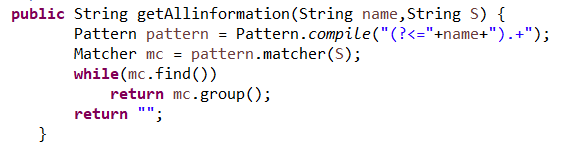
1.所以建立一个Parser类，里面有两个方法。

一个方法用来判断读入的一个航班计划项字符串格式是否符合报告要求：

根据以上要求可以设计方法如下：



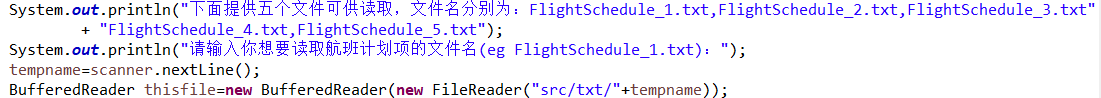
一个方法用来得到一行字符串中某个字符串后面的所有字符串。可以设计方法如下：



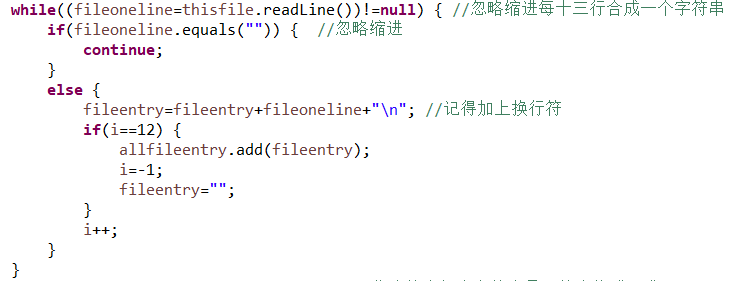
2.然后在FlightScheduleApp中添加读入文件中的计划项操作：



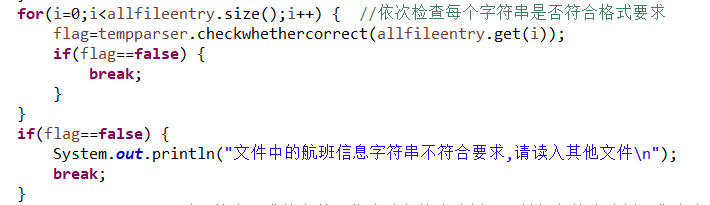
3.具体实现：  
3.1读取文件：



3.2将文件的每十三行合成一个字符串，忽略空行，每读取一行加一个换行符：

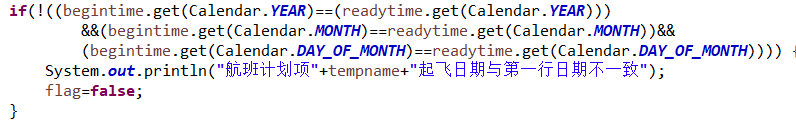


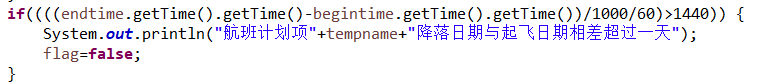
3.3 用Parser中的checkwhethercorrect方法来检查每个字符串是否符合手册的格式要求，不符合要求则退出：



3.4对于符合要求的字符串集合，每次读入一个字符串，创建一个航班计划项，用Parser中的getAllinformation来得到计划项的所有信息，为计划项赋值

3.5除去所有的非法情况：



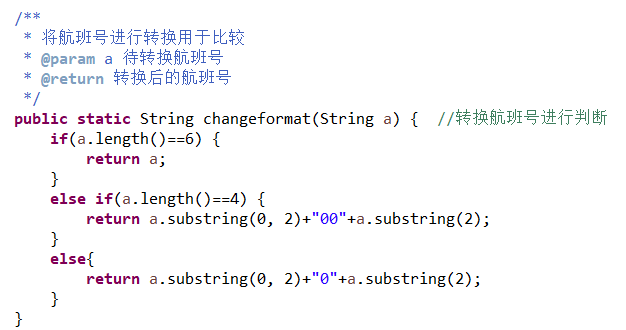




其中在判断两个计划项的名字是否相同时，因为手册规定了：



所以我使用了一个航班号转换方法：



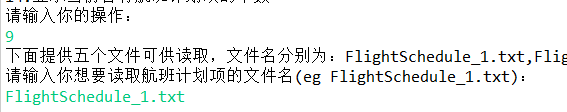
这样便能实现比较：

3.6只有最后flag为true，才把读取的所有计划项集合加入总的计划项集合中，

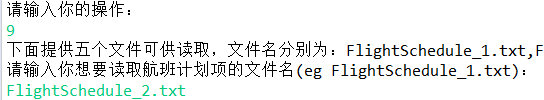
否则将读取的计划项集合初始化，这样遇到不符合规定的文件就不读入原计划项集合中：



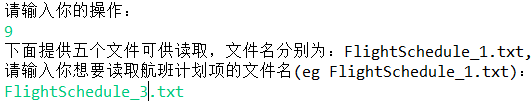
4.最后便能实现读取文件并且判断是否符合要求：



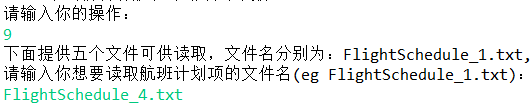




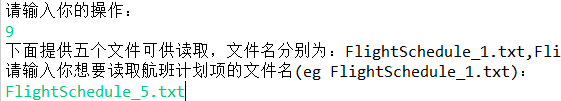








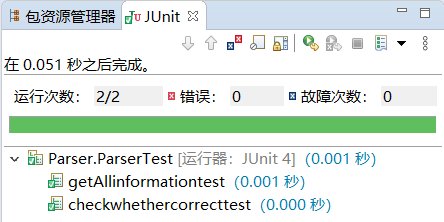






5.测试（覆盖所有非法情况）：







## 应对面临的新变化

只考虑你选定的三个应用的变化即可。

### 变化1

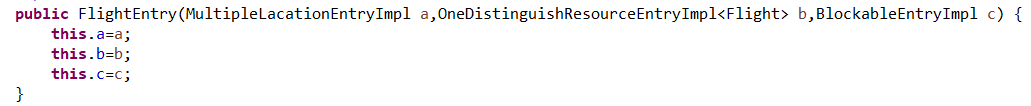
可应对变化，代价较小，简单复制粘贴稍加修改即可：修改时间：45分钟

1.改为MultipleLocationEntry和BlockableEntry





2.改构造方法：



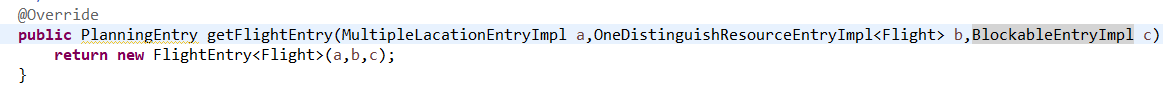
3.复制粘贴TrainEntry的方法







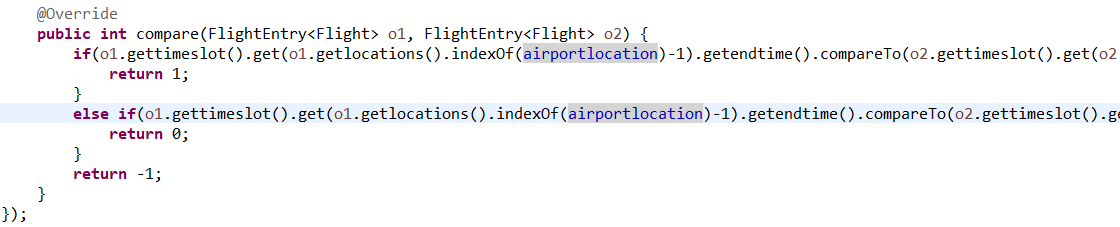
4.改factory里面的委托的变量内容



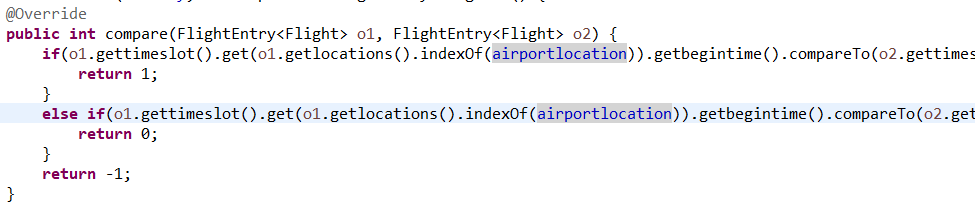
5.改FlightEntryboard，复制粘贴TrainEntryBoard

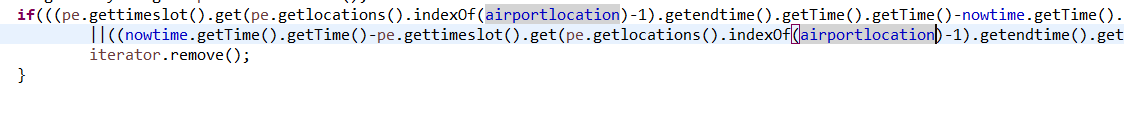


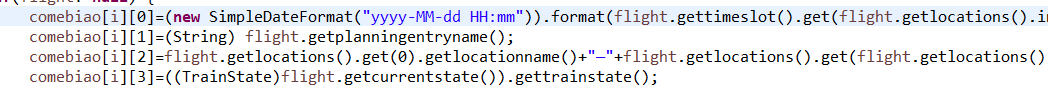


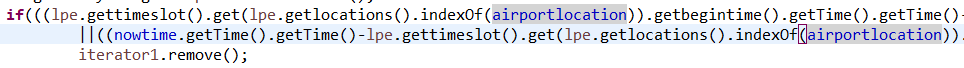


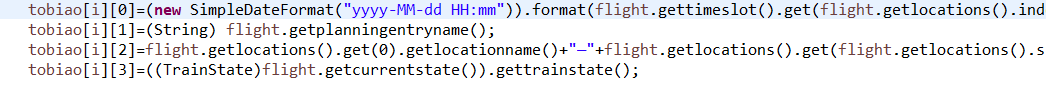












6.改state：创建新的阻塞状态

并复制高铁所有状态转换过程

，

参考高铁的状态转换

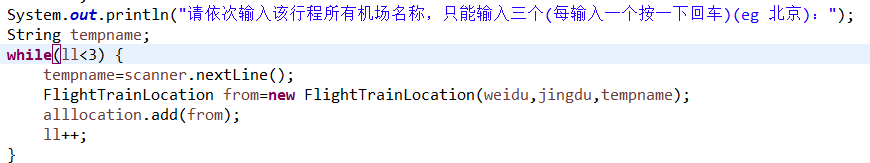
7. 改FlightScheduleApp：除去原来的读文件功能，将其改为阻塞计划项功能，然后改一些变量简单复制粘贴TrainScheduleApp即可

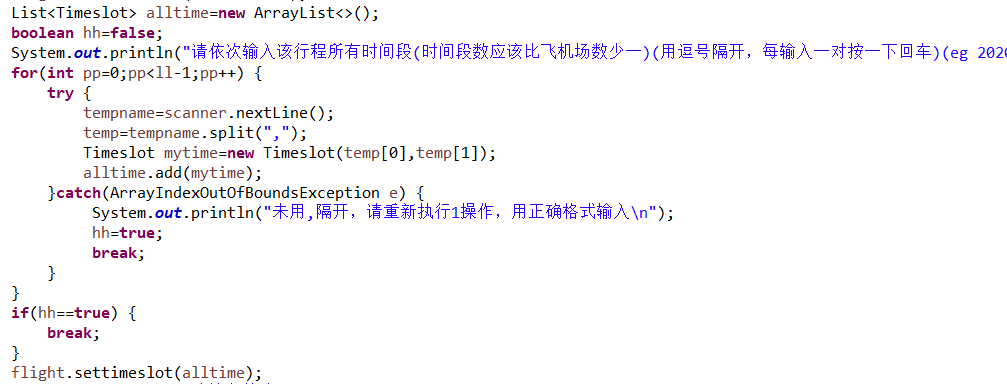


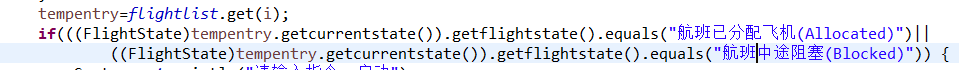




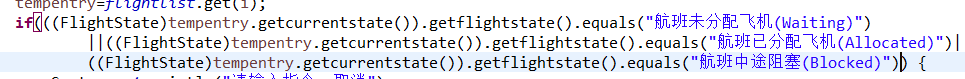
复制粘贴Train并改为只有三个站稍加修改：











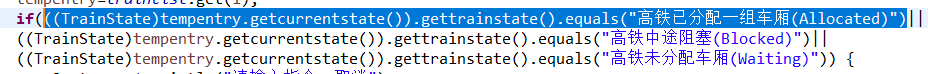
8.改PlanningEntryAPIs：稍加修改即可



### 变化2

只要分配资源，高铁计划项就不能被取消，可以应对变化，很简单，删掉两个判断条件即可。时间：一分钟

把TrainScheduleApp里面取消计划项的第一个判断条件Allocated和第二个判断条件Blocked去掉即可：



### 变化3

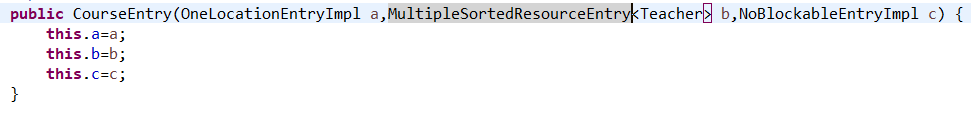
可应对变化，代价较小，简单复制粘贴稍加修改即可：修改时间：20分钟

1.改为MultipleSortedResourceEntry<Teacher>

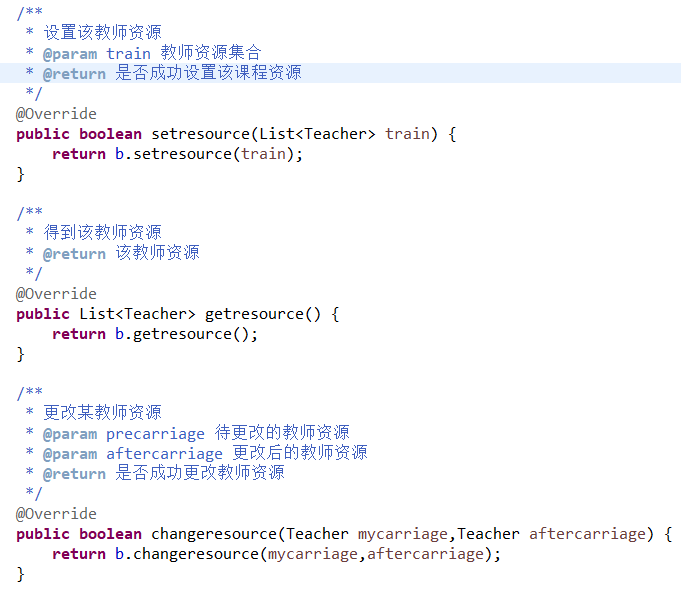


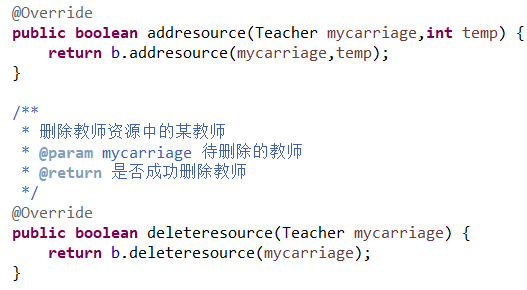


2.改构造方法：

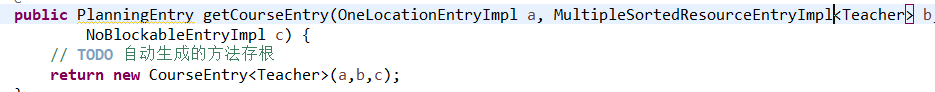


3.复制粘贴TrainEntry的方法稍加修改:

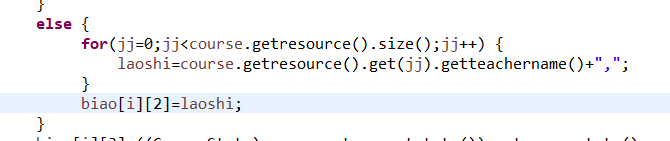




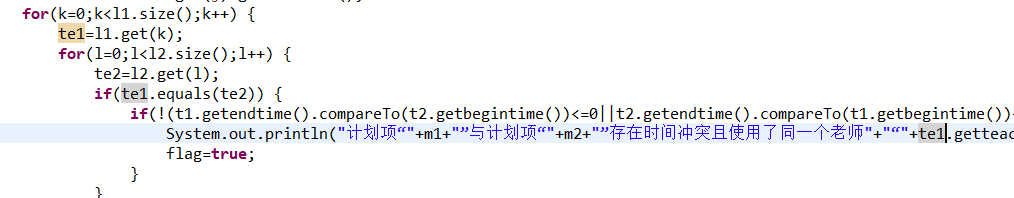
4.改Factory的变量：

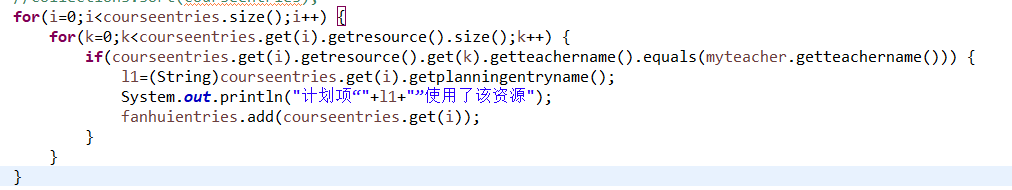


5.改CourseEntryBoard里面的资源显示：按次序显示所有老师的名字



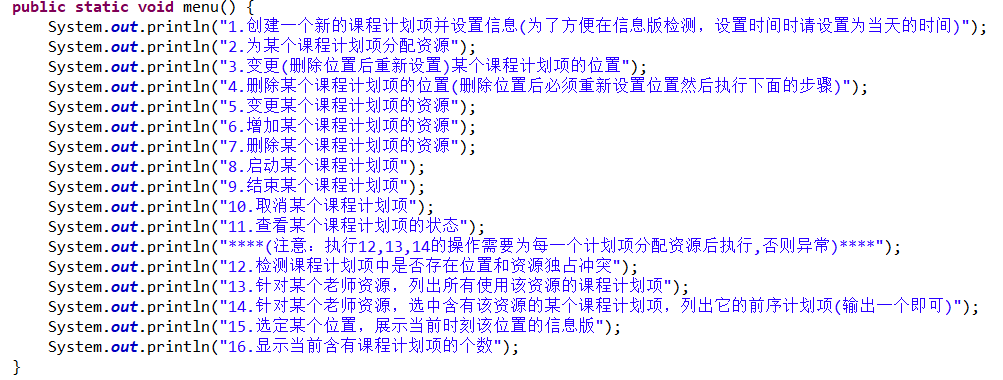
6. 改PlanningEntryAPIs：仿照TrainEntry的方法稍加修改即可







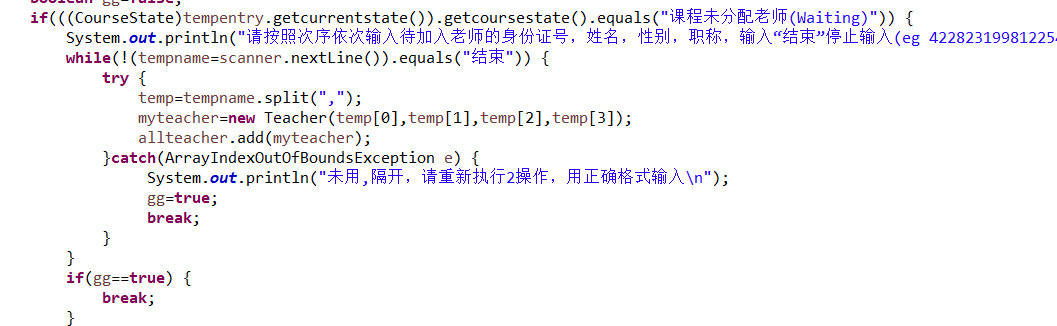
7.在CourseCalendarApp里面：

菜单变为：

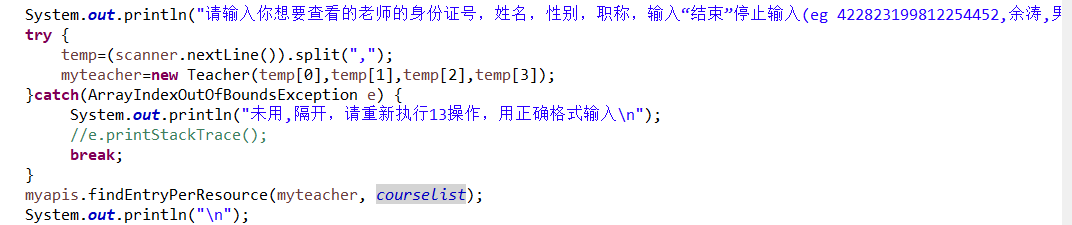
其他内容复制高铁的app然后稍加修改即可





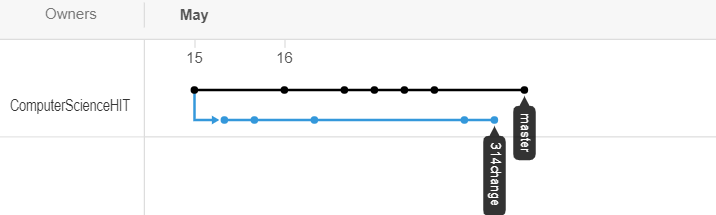








## Git仓库结构



# 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 |
| 4.20 | 18:00-22:00 | 完成环境的搭建 | 未完成 |
| 4.21 | 16:00-22:30 | 阅读报告及完成综合设计 | 完成 |
| 4.22 | 19:00-23:00 | 提取共性方法及实现 | 未完成 |
| 4.23 | 18:00-22:30 | 方案五的试用 | 完成 |
| 4.24 | 18:00-22:00 | 方案五的具体设计 | 未完成 |
| 4.25 | 17:50-22:20 | 完成所有维度的综合 | 完成 |
| 4.26 | 19:00-22:30 | R的设计 | 完成 |
| 4.27 | 20:00-22:40 | Location的设计 | 完成 |
| 4.28 | 14:00-20:30 | Timeslot的设计 | 完成 |
| 4.29 | 13:00-21:00 | 自学State设计模式并设计Flight的State | 完成 |
| 5.1 | 18:00-21:30 | 完成另外两个State的设计 | 完成 |
| 5.3 | 18:00-22:30 | 自学windowsbuilder完成board设计 | 未完成 |
| 5.4 | 19:00-21:30 | API设计功能一 | 完成 |
| 5.4 | 19:00-22:10 | API设计功能二 | 完成 |
| 5.6 | 19:00-21:30 | API设计功能三 | 完成 |
| 5.7 | 19:00-22:30 | 工厂方法和迭代器 | 未完成 |
| 5.8 | 14:00-22:30 | 策略模式，设计算法二 | 完成 |
| 5.9 | 15:00-20:30 | 航班APP | 完成 |
| 5.10 | 18:00-21:30 | 航班APP | 未完成 |
| 5.11 | 18:00-22:30 | 课程APP | 完成 |
| 5.12 | 19:00-21:30 | 高铁APP | 完成 |
| 5.13 | 19:00-22:10 | 所有测试 | 完成 |
| 5.14 | 19:00-21:30 | 分支 | 完成 |
| 5.15 | 15:00-20:30 | 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 |
| Calendar类的使用 | csdn查找：https://www.cnblogs.com/jdw12306/p/7699276.html |
| 如何抽象共性方法提高复用性 | 仔细阅读ppt和网上查找 |
| 正则表达式的应用 | 网上查找：  https://www.cnblogs.com/ggjucheng/p/3423731.html |

# 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

这次实验将课程上讲到的各种设计方法应用于实际编程中，极大的提高了程序的可复用性，对自身的代码能力有了很大的提高。

但是关于泛型的使用还是不太会，需要继续联系。

## 针对以下方面的感受

1. 重新思考Lab2中的问题：面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？本实验设计的ADT在五个不同的应用场景下使用，你是否体会到复用的好处？

面向ADT编程复用性高，面向应用编程复用性低。好处很多。省略很多代码。

1. 重新思考Lab2中的问题：为ADT撰写复杂的specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后的编程中坚持这么做？

提高代码的可读性，抽象出类的具体。愿意。

1. 之前你将别人提供的API用于自己的程序开发中，本次实验你尝试着开发给别人使用的API，是否能够体会到其中的难处和乐趣？

能。

1. 在编程中使用设计模式，增加了很多类，但在复用和可维护性方面带来了收益。你如何看待设计模式？

是一种提高代码复用性的好方法。设计起来难，但功能强大。

1. 你之前在使用其他软件时，应该体会过输入各种命令向系统发出指令。本次实验你开发了一个解析器，使用语法和正则表达式去解析输入文件并据此构造对象。你对语法驱动编程有何感受？

是一种隐秘的编程方式。能够提高隐私性。

1. Lab1和Lab2的大部分工作都不是从0开始，而是基于他人给出的设计方案和初始代码。本次实验是你完全从0开始进行ADT的设计并用OOP实现，经过五周之后，你感觉“设计ADT”的难度主要体现在哪些地方？你是如何克服的？

开始的抽象化和调用关系。

1. “抽象”是计算机科学的核心概念之一，也是ADT和OOP的精髓所在。本实验的五个应用既不能完全抽象为同一个ADT，也不是完全个性化，如何利用“接口、抽象类、类”三层体系以及接口的组合、类的继承、设计模式等技术完成最大程度的抽象和复用，你有什么经验教训？

最好开始设计好思路和框架，后期重新修改太麻烦。

1. 关于本实验的工作量、难度、deadline。

工作量太大，测试很繁琐，量太多，占用太多时间。难度大。截止日期合适。

1. 到目前为止你对《软件构造》课程的评价。

提高了我的编程能力。是一门能学到真本事的学科。