

**2020年春季学期  
计算机学院《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 张高玮 |
| 学号 | 1180300407 |
| 班号 | 1803004 |
| 电子邮件 | 1462034631@qq.com |
| 手机号码 | 15081521645 |

**目录**

[1 实验目标概述 1](#_Toc37326126)

[2 实验环境配置 1](#_Toc37326127)

[3 实验过程 1](#_Toc37326128)

[3.1 待开发的三个应用场景 1](#_Toc37326129)

[3.2 面向可复用性和可维护性的设计：PlanningEntry<R> 1](#_Toc37326130)

[3.2.1 PlanningEntry<R>的共性操作 1](#_Toc37326131)

[3.2.2 局部共性特征的设计方案 2](#_Toc37326132)

[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） 2](#_Toc37326133)

[3.3 面向复用的设计：R 2](#_Toc37326134)

[3.4 面向复用的设计：Location 2](#_Toc37326135)

[3.5 面向复用的设计：Timeslot 2](#_Toc37326136)

[3.6 面向复用的设计：EntryState及State设计模式 2](#_Toc37326137)

[3.7 面向应用的设计：Board 2](#_Toc37326138)

[3.8 Board的可视化：外部API的复用 2](#_Toc37326139)

[3.9 PlanningEntryCollection的设计 2](#_Toc37326140)

[3.10 可复用API设计及Façade设计模式 2](#_Toc37326141)

[3.10.1 检测一组计划项之间是否存在位置独占冲突 2](#_Toc37326142)

[3.10.2 检测一组计划项之间是否存在资源独占冲突 2](#_Toc37326143)

[3.10.3 提取面向特定资源的前序计划项 2](#_Toc37326144)

[3.11 设计模式应用 2](#_Toc37326145)

[3.11.1 Factory Method 3](#_Toc37326146)

[3.11.2 Iterator 3](#_Toc37326147)

[3.11.3 Strategy 3](#_Toc37326148)

[3.12 应用设计与开发 3](#_Toc37326149)

[3.12.1 航班应用 3](#_Toc37326150)

[3.12.2 高铁应用 3](#_Toc37326151)

[3.12.3 进程应用 3](#_Toc37326152)

[3.12.4 课表应用 3](#_Toc37326153)

[3.12.5 学习活动应用 3](#_Toc37326154)

[3.13 基于语法的数据读入 3](#_Toc37326155)

[3.14 应对面临的新变化 3](#_Toc37326156)

[3.14.1 变化1 3](#_Toc37326157)

[3.14.2 变化2 4](#_Toc37326158)

[3.14.3 变化3 4](#_Toc37326159)

[3.15 Git仓库结构 4](#_Toc37326160)

[4 实验进度记录 4](#_Toc37326161)

[5 实验过程中遇到的困难与解决途径 4](#_Toc37326162)

[6 实验过程中收获的经验、教训、感想 5](#_Toc37326163)

[6.1 实验过程中收获的经验和教训 5](#_Toc37326164)

[6.2 针对以下方面的感受 5](#_Toc37326165)

# 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

①安装JDK8，去官网下载，选择下载目录

②配置环境变量：计算机→属性→高级系统设置→高级→环境变量→系统变量→新建JAVA\_HOME变量，值为JDK安装目录

Path变量→编辑：输入%JAVA\_HOME%\bin

③官网下载、安装eclipse、Git

④根据老师实验课指导了解Junit创建、使用testcase的方法

⑤注册Github账号，根据给出的URL地址绑定学号，创建仓库

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

https://github.com/ComputerScienceHIT/Lab3-1180300407.git

# 实验过程

## 待开发的三个应用场景

我选择的三个应用：

1. 航班管理
2. 高铁车次管理
3. 大学课表管理

三个场景的相同点：

1. 三个计划项都需要配置资源，且资源是可区分的
2. 三个计划项都需要占用物理位置，均可提前设定
3. 三个计划项都需要时间来标识开始和结束，均可提前设定
4. 三个计划项的执行过程可划分为若干状态:：未分配资源、已 分配资源但未启动、已启动、已完成、已取消(、阻塞)等

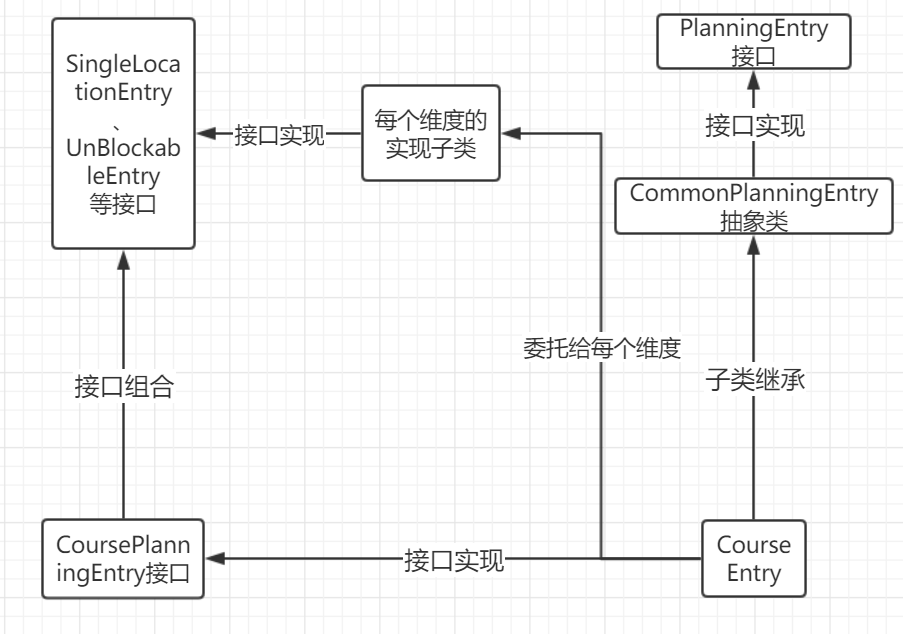
三个场景的不同点:

1. 资源的数量不同:航班和课程是单个，高铁是多个
2. 位置的数量不同:有1个，2个，多个；并且只有课程的位置是可更改的
3. 描述需要的时间对个数不同:只有高铁一组时间对，其他为1个
4. 高铁的执行过程会出现阻塞状态

## 面向可复用性和可维护性的设计：PlanningEntry<R>

我在实验中选择方案5通过接口组合+委托实现复用

以课程设计CourseEntry为例展现设计结构

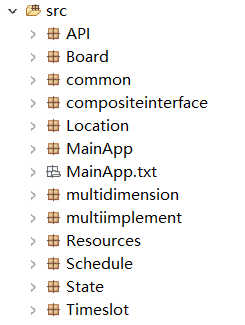


PlanningEntry是最顶层接口，存放共性操作。

CommonPlanningEntry是PlanningEntry的实现类,我设计为抽象类,将其中的抽象函数“特性实现的自由”完全交付子类。

每个计划项的特性维度在SingleLocation等接口中予以实现，进而通过这些接口的组合就可完成一个子类的功能，于是我设计CoursePlanningEntry等三个子类接口进行接口组合，继承那些接口的功能，它们的实现子类CourseEntry再利用委托来实现更大程度复用。

这里将我的包结构也粘贴出来予以说明：



API针对3.10中可复用API设计，3.11中策略模式的使用；

Board针对3.8中Board设计及3.9外部API、3.11中迭代器的使用

common包含PlanningEntry顶层接口、它的抽象类实现、以及专为了测试而写的模拟函数

compositinterface是具体应用的接口和具体子类

Location针对位置的实现，包含Location类

MainApp是3.12节的应用程序

multimension针对流程图中SingleLocationEntry等每一个维度定义出来的接口

multiimplement针对multimension中接口的实现

Resources包含各个资源类

Schedule针对具体应用场景，如航班管理：管理一组机场、一组飞机、一组航班等

State是State设计模式中的各个状态类

Timeslot包含Timeslot类，是时间对这一属性的实现

### PlanningEntry<R>的共性操作

在前面3.1分析相同点时，我们其实就将共性操作提炼出来了：

1.



分配资源以及获取计划项占用的资源，为了实现三者统一，全部用List存储。

2. 获取位置信息。同样为了实现三者统一，全部用List存储。因为对位置的设置三者还是各自具有鲜明特点：如课程的位置可修改，机场的起点-终点位置对等，为了避免其AF、RI发生冲突，这里不提供对应共性功能，而是交付给子类实现“特性”

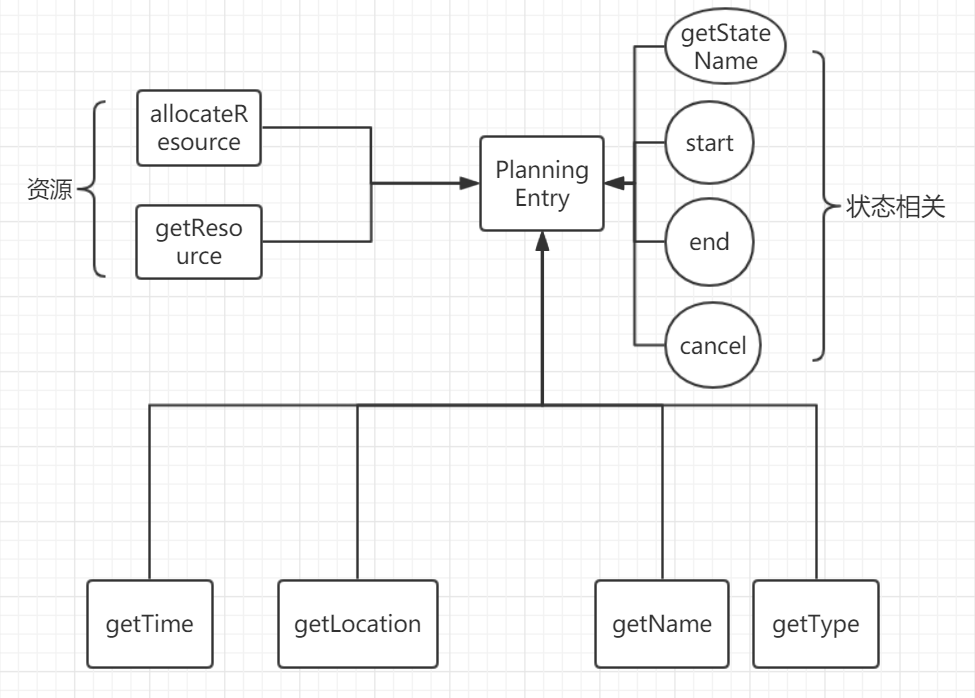
3. 时间信息的获取。同样为了实现三者统

一，全部用List存储，同2，将设置时间交付给具体子类

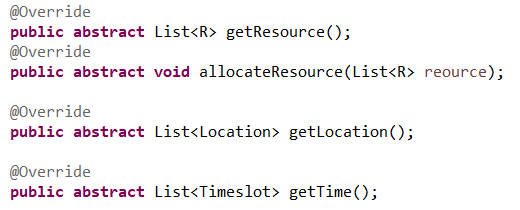
4.执行过程的一系列状态：start、cancel、end，采用state模式进行设计，在

后面章节进行介绍。这里没有包含block是因为block是高铁的特性，应在子类中予以提供。

函数图如下展示：



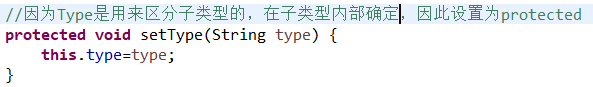
CommonPlanningEntry是其一个实现类



我将这些函数设置为抽象函数是为了简化，如果这里将这些函数实现，势必会引入数据结构进行存储，而因为set函数要由每个维度的ADT予以实现，诸如

SingleLocationEntryImpl，但这些类如果要实现set势必还会进行信息的存储，这样具体到应用子类时实际上会有重复的信息存储。因此这里设置为抽象函数。

此外还包括对计划项类型的设定、对状态的设定





### 局部共性特征的设计方案

局部共性—位置、资源、时间等每个维度的特征,我在multidimension中设计了这些接口，并在multiimplement中予以实现

①单位置—两个位置—多位置

由于我们在CommonPlanningEntry里面把getLocation设置为抽象函数,因此这里既要实现对位置的设定(set),也要有位置的observer—get函数

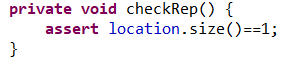
<1>单位置:两个函数



由于为了在子类实现共性的CommonPlanningEntry接口中的get函数,因此这里内部实现用一个list保存计划项的位置



并且get、set函数都是以list作参数，但为了保证单位置特性，在checkRep中判断location的大小，在每次setLocation时进行判断



<2>两个位置:与单位置相似,不同的是两个位置起点和终点是非常明确的,因此也提供了起点和终点分别的get函数,同时内部实现也以两个Location变量代替List进行存储

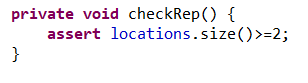


此外,位置是要求不能更改的,



用一个布尔变量标识是否已经对位置进行设定，从而保证位置只设定一次

<3>多位置:与两个位置类似,用布尔变量标识是否已经对位置进行设定。内部实现方式改变

用一个list存储，并在checkRep中检查locations的大小是>=2的,从而实现对多位置的限定

三者的实现中均使用防御式编程来避免表示泄露

②单资源-多资源(有序)

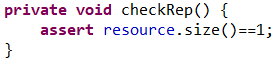
均只包含资源的set和get函数

为了实现CommonPlanningEntry中的共性,这里参数全部统一为list型

为了保证资源只设置一次,同样均用布尔变量标识

<1>单资源:

为了保证单资源特性,checkRep中进行判断



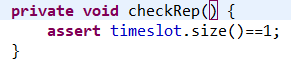
<2>多资源:

因为资源顺序的标准以及具体安排时的考量是未知的，因此“有序性”应该是由client来确定的,其在确定计划项时候给定的位置顺序就被认为是“已有序、可区分”了的。

③可阻塞-不可阻塞:

在这个维度均实现了对时间的处理:设置与获取,内部实现均用List来存储实现CommonPlanningEntry中的共性要求,用布尔变量保证只设置一次

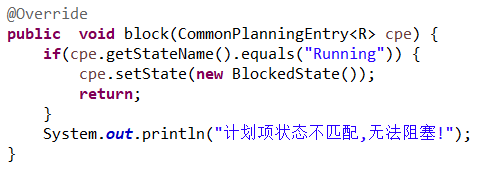
<1>不可阻塞



应该只有一个时间对来表示起始和终止时间

<2>可阻塞

添加block功能进行计划项阻塞



因为在三者执行过程的状态转移图中,block部分是高铁特有的，因此单独拿出来进行判断，仅当计划项已开始时才可以阻塞，阻塞后便可以将Blocked状态引入状态转移过程中，实现高铁的状态转换图

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

核心思想:委托

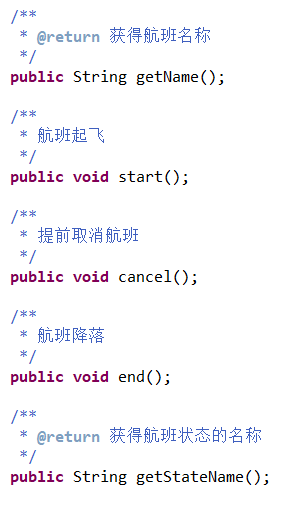
以FlightPlanningEntry和FlightEntry为例

①首先对于FlightPlanningEntry接口，进行三个接口的组合



因为该接口无需继承PlanningEntry接口，为了以后更好地扩展,将PlanningEntry中获取基本信息的函数和状态相关函数在这里再次声明，以便如果将来有接口的新完成方式时可以直接实现该FlightPlanningEntry接口而无需和PlanningEntry有直接关系。

如图所示:



②具体子类的实现:FlightEntry

<1>首先继承CommonPlanningEntry,利用父类已经实现的状态相关函数、获取计划项基本属性信息的函数

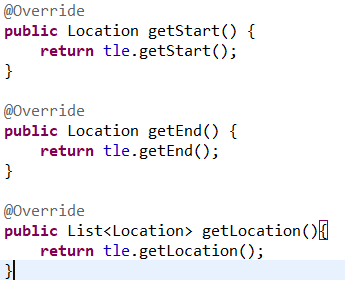


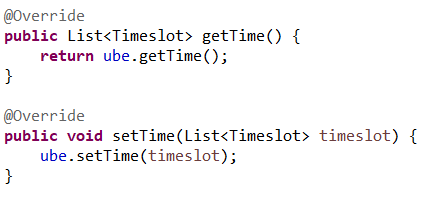
<2>再实现

完成每个维度提炼出来的功能，这些功能全部利用委托实现

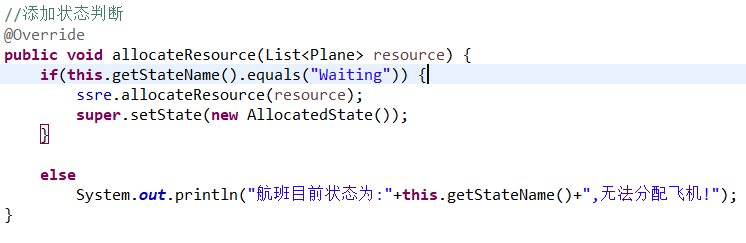


如时间、位置：





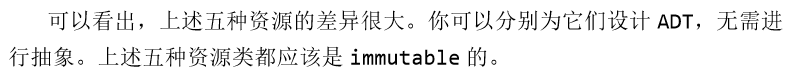
其中资源分配既要实现状态的变换,又要实现资源的设定,而在资源维度设计时为了减少与PlanningEntry的重复,将资源的具体设置与状态分离，因而在子类组合时需加之以状态判断。因此辅之以父类函数的状态获取函数与设定函数



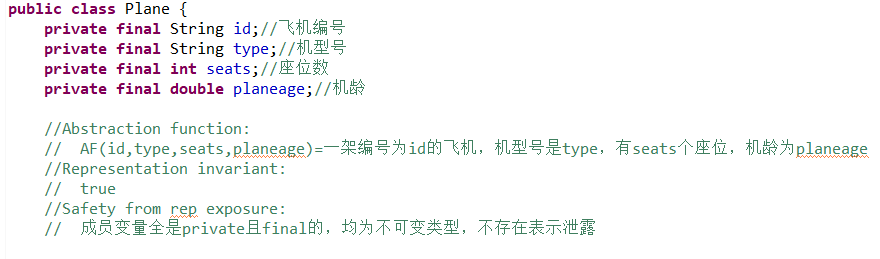
这样一来具体子类的实现完全是由共性的CommonPlanningEntry和局部共性维度的实现相组合就可实现，实现了复用

## 面向复用的设计：R

根据实验指导中的说明,三种资源分别设计，无需抽象

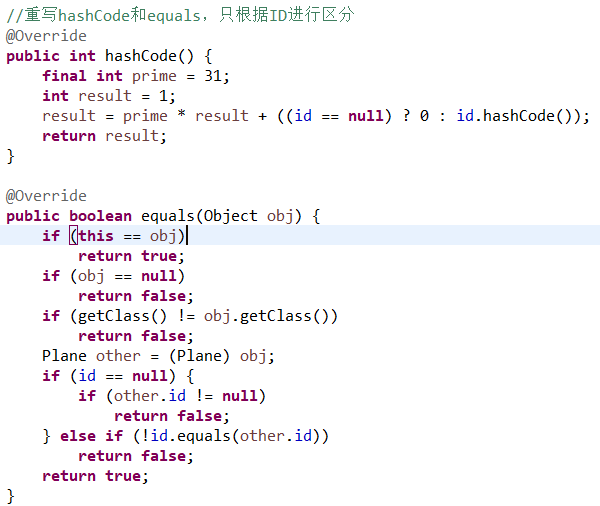


①Plane类



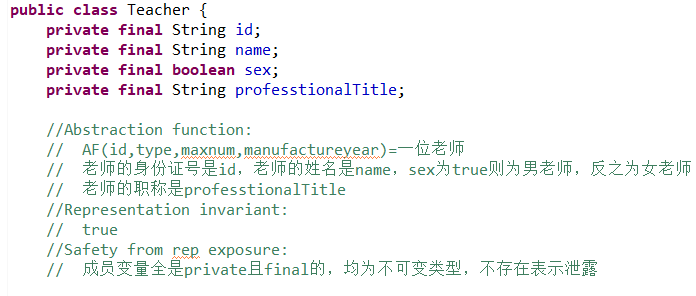
方法是常规的构造函数与get方法，由于是不可变类，因此对equals和hashCode进行重写，只将飞机的编号(id)作为相等的条件

注意，这里并没有对飞机的编号作任何限制,3.13节语法输入的飞机要求是在3.13读入文件时单独处理的，在app使用命令行创建飞机时飞机编号没有任何限制，降低前置条件



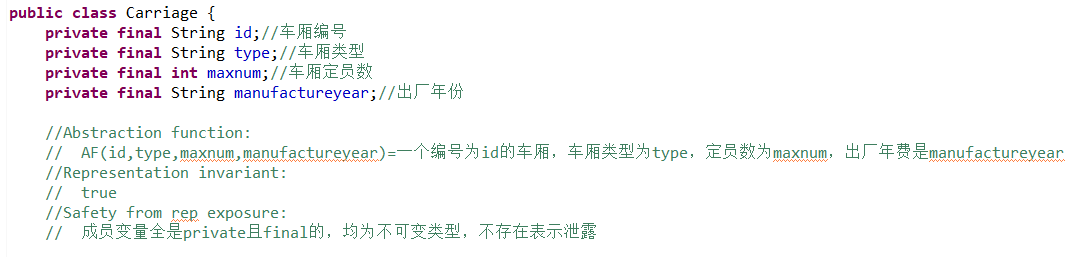
②Teacher类

与Plane类似



重写equals和hashCode时是根据老师的身份证号id

③Carriage类

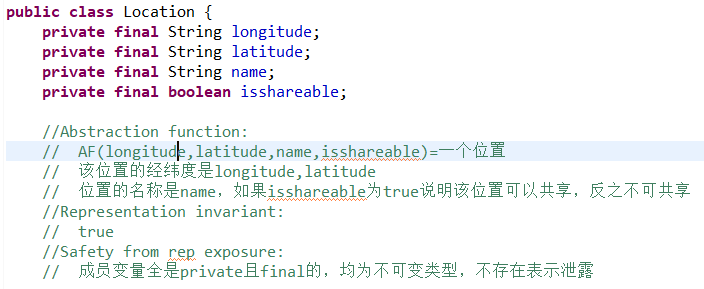


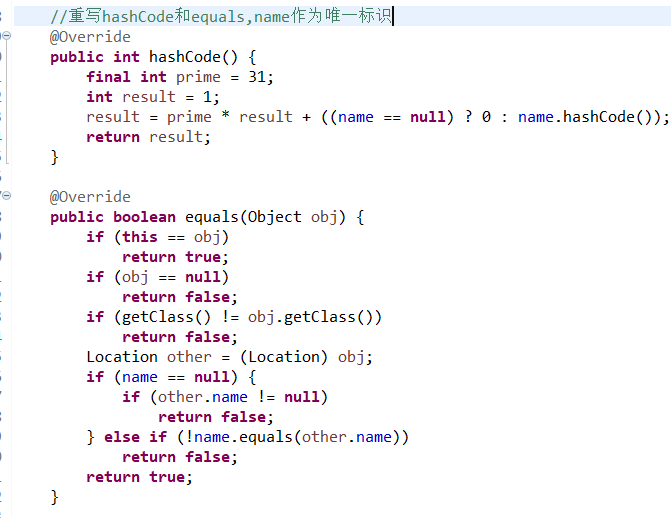
重写equals和hashCode时是根据车厢的编号id

## 面向复用的设计：Location

实现实验指导上要求：







重写equals和hashCode。由于Location设计的基础性,任何涉及“位置”的ADT都可以复用Location类，复用性高，但其只是一个基础单元,复用价值相对没有那么高

前面已经强调过,位置相对顺序的标准是未知的。如果要求一组有序的位置集合,则位置的相对顺序必须由client自己设置

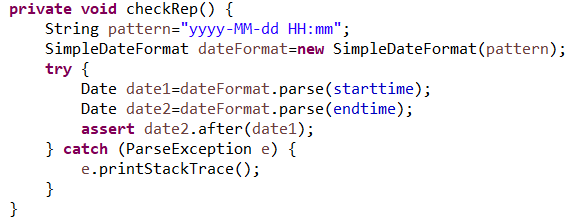
## 面向复用的设计：Timeslot



时间对是起始时间+终止时间的组合,每个时间点都符合 yyyy-MM-dd HH:mm 的语法规则，这一点在构造函数处用正则表达式加以保证



并且终止时间应严格晚于起始时间



在checkRep中加以限制

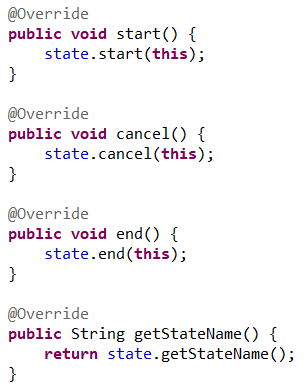
注意,在高铁这种多站点多时间对的计划项中,一组时间对的相对顺序与一组位置的顺序应该是一致的、同时由client确定的，如第一个Timeslot的开始时间意味离开第一个站点的时间，终止时间意味到达第二个站点的时间……依此类推。

## 面向复用的设计：EntryState及State设计模式

①CommonPlanningEntry中维护一个EntryState成员变量

 ,因为计划项创建时还未分配资源，故初始化为Waiting状态。

②CommonPlanningEntry中表示计划项开始、取消、结束等状态转移的方法全部委派给该成员变量执行,利用state在不同时刻处于的不同状态自动执行该状态下对应的方法，从而免去繁琐的状态判断



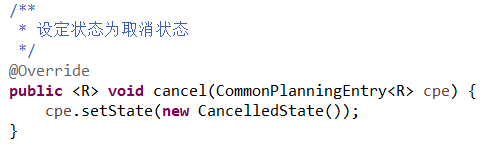
③EntryState作为对状态的抽象描述,应该提供共性的状态转移的方法的接口(即②中需要被委派执行的任务),然后分别创建各个状态子类来予以实现



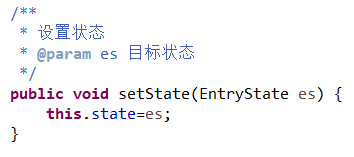
因为不同子类执行相应状态变换的结果不同,这里没有列出统一的方法规约。

列举其中一个状态子类如下:

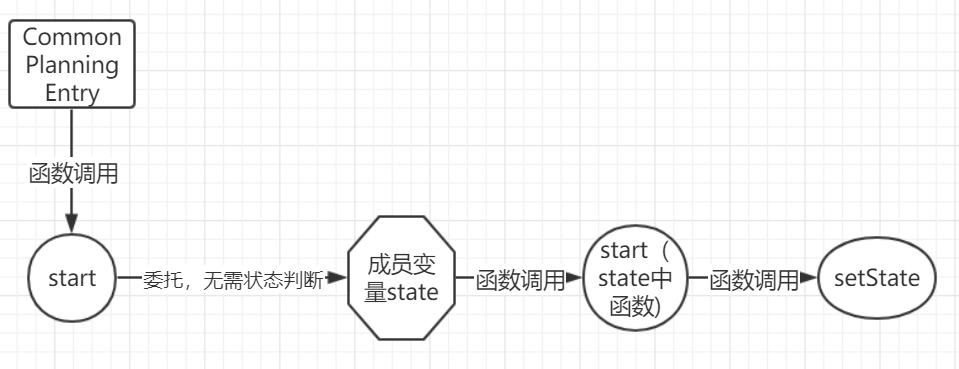




对应于实验指导上的状态转移图,状态子类中的每个状态转换都有合法和非法之分,这时只需根据该子类到底是何种状态作出确定的执行即可。每个子类中具体状态转换的实现是通过CommonPlanningEntry中留有的set函数对状态进行设定



因此一次状态转换如start的执行过程可以用下面的函数调用简单概括:



## 面向应用的设计：Board

注意，Board中的计划项都应该是已分配资源的、不处于Waiting状态的计划项

①FlightBoard



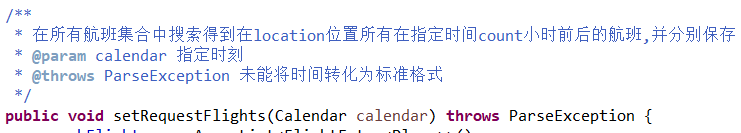
<1>count是个常量，用来表示时间要求：如这里要展示一小时前后的航班，则设置count=1，这样避免当时间变化时对一堆类里的常数进行修改，提高可维护性和复用性。location表明该Board所处的地点,reachFlights保存所有count小时前后抵达location的航班,departureFlights保存所有在count小时前后从该位置起飞的航班，allFlights是二者的整合，用以实现迭代器。这里所有航班都应该是已分配飞机的航班,否则会在board展示时出现空指针异常，因此用checkRep加以限制；

<2>构造函数

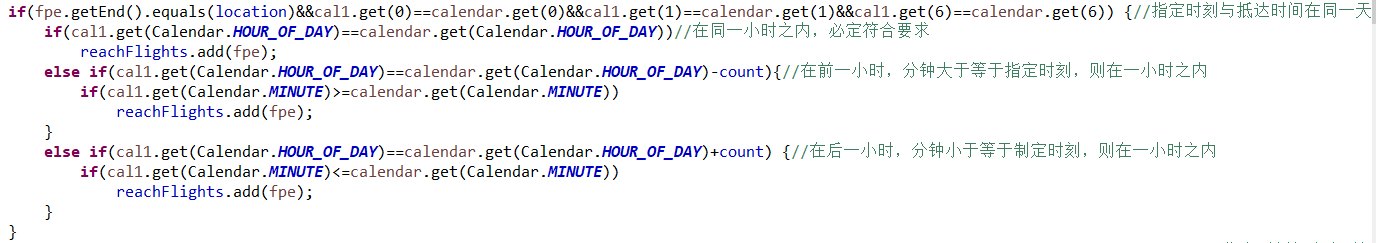


指定位置、计划项集合、要求时间。注意传进来的航班集合并没有要求一定经过该位置，也没有时间上的要求,而我的想法是在构造函数时就将所有满足要求的航班设置好,以避免出现在构造函数与搜索符合条件航班的空档期使得client错误调用。因此我将这个搜索功能分离出去,即setRequestFlights方法,并在构造函数中调用,使得每个Board被创建时就已经将航班搜索完毕

<3>setRequestFlights方法



该方法对在构造函数时传进来的“杂乱”计划项进行搜索只保留经过该location且在count小时前后离开或抵达的航班,并分别保存在departureFlights、reachFlights中。其中在搜索时就用到了count常量进行比较：



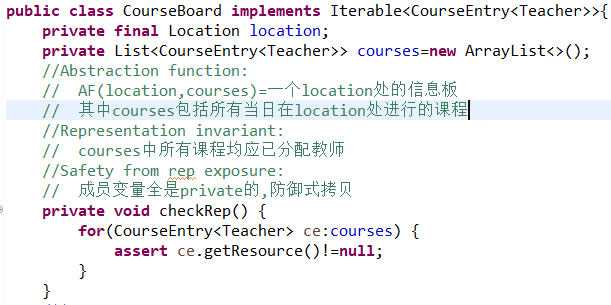
将Timeslot中的字符串型变量转为Date类型，再进而转为Calendar类型进行同一天、count小时的分别判断。

再对其进行从早到晚时间排序，为了保证方法的单一功能，将该排序功能分离出去,即调用sortFlights函数完成，这里设置为private的一个方法。

<4>sortFlights方法:该方法设置为private,只在setRequestFlights中调用，避免随意调用。在setRequestFlights中，调用该函数之前,departureFlights和reachFlights中已经保存了所有可以展示的计划项,只不过是无序的,而allFlights中保存的还是构造函数伊始传进来的“杂乱”计划项集合，因此该方法的功能就是将reachFlights和departureFlights调整为从早到晚，这里使用了选择排序。排序完成后对二者的元素执行一个整合，直到这时allFlights中的航班才是符合要求的

注意，测试时抵达航班状态应为Running/Cancelled/Ended,因为Waiting状态不符合抵达航班实际。

②CourseBoard：整体思路完全一样,而且更加简单：单位置，因此不用像Flight一样将抵达与离开分开保存,并且时间要求是当日所有课程，也无需用常量进行设定



③TrainBoard:比起Flight，思路很接近,但要更加复杂:要包括中间站点是该位置的列车。实现细节略有改变:

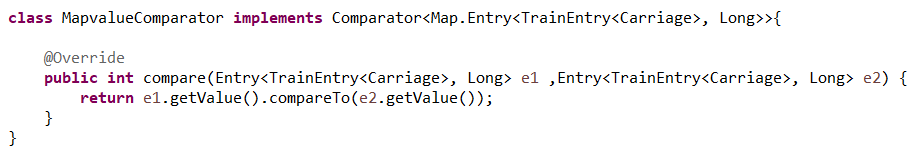


这里将list改为Map存储,将计划项搜索出来的同时将计划项对应的抵达/出发时间也予以保存，这样在排序时可以方便快速地进行。而FlightBoard中使用List就可以完成的原因是Flight只有一个时间对,保存了计划项就可以得到它经过该位置的时间。但Train不可以,因为只保存了计划项也无法确定该位置是第几个站点、以及经过该位置的时间。

此外增加的就是对于中间站点的搜索:

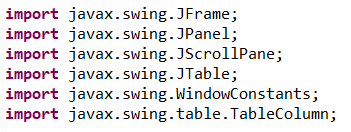


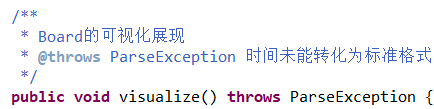
而为了对Map中的计划项排序,实现一个Comparator接口，根据Map的value进行排序即可



## Board的可视化：外部API的复用

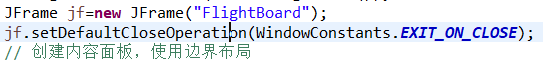
使用JSwing、JTable实现可视化





没有参数，展示时间即为调用该函数进行展示的当前时间。注意，这里在对可视化部分进行测试时,在构造函数传进的时间参数要尽量与当前时间一致，并且测试用的计划项时间也要尽量在该范围内。否则在构造函数中传一个比较久远的时间,会将当前时间附近的计划项剔除,从而失去展示效果

以FlightBoard为例。首先再次调用setRequestFlights函数，将计划项对准为当前时间，然后创建内容面板，设置标题为FlightBoard



第一行展示当前时间与位置



时间的展示可以先将Timeslot中的String成员变量转为Date类型,再转为Calendar类型获取小时和分钟,起点和终点可以通过计划项容易地获得，航班状态显示只需要根据计划项的状态信息进行相应的转换即可



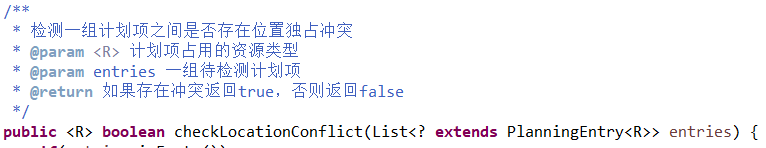
分别对抵达航班和出发航班作这样的处理即可完成。

其他两个计划项的可视化与该函数思路类似，不再赘述。其中每个类内部都用注释形式保留了我测试效果时写的main函数，可通过该例进行查看，不过需要注意的是，就像上面提示的一样，构造函数传进去的时间参数一定要与调用可视化函数进行展示的系统时间尽量接近或一致！！另外需要注意的就是计划项创建时的预计时间。效果展示如下:

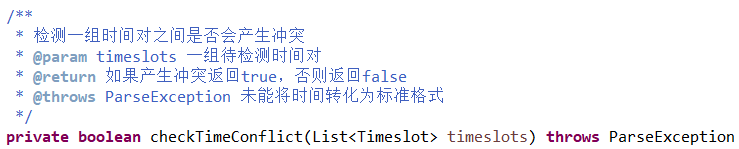


## 可复用API设计及Façade设计模式

### 检测一组计划项之间是否存在位置独占冲突



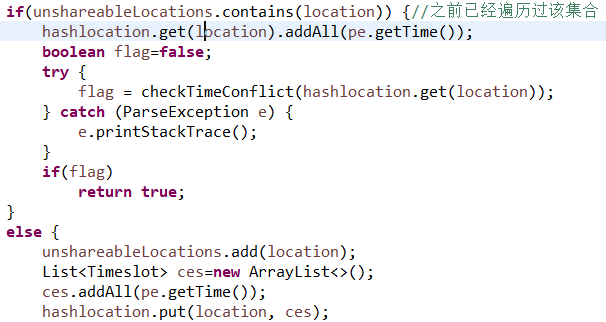
核心思想是维护一个Map,其中的key是一个非共享位置,value是一个list，保存entries中所有占用该位置的计划项的占用时间,遍历一次entries就可以建立哈希表，然后这样就可以对Map中的所有value中的list进行搜索，查看该list中的占用时间是否存在对于某一个位置的冲突，如果存在，就存在位置冲突。将这个遍历value的过程分离出一个函数保证方法的“单一功能”：



上述实现过程的简要代码展示如下:



具体实现时又引入一个set，记录所有已经遍历过的非共享位置，这样对于每一个非共享位置可进行快速判断:如果已经加入了set,则已经遍历过,通过key直接找到map中的value，对list进行扩充，反之就新建一个list，put进map之中，构成新的键值对



### 检测一组计划项之间是否存在资源独占冲突

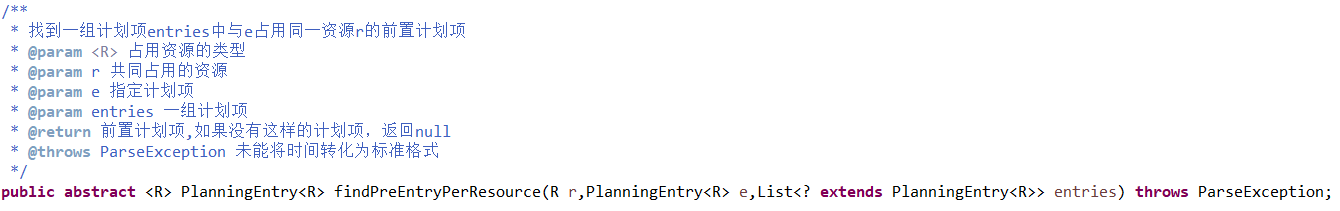
与3.9.1中位置冲突检测的思路非常类似，仍然用一个Map+一个Set实现检测，不同的是Map的key的类型和Set的类型都转为对应资源类型即可，并且仍然可以用checkTimeConflict函数进行对Map的value遍历检测的功能



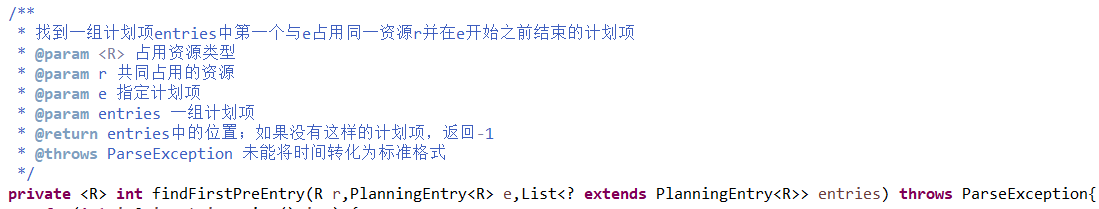


### 提取面向特定资源的前序计划项

针对该函数进行策略模式设计，在PlanningEntryAPIs类中设置为抽象函数，然后在两个具体类中给出两个不同的具体实现。



<1>第一种实现：首先找到该组计划项的list中第一个与e占用相同资源r并在e开始之前就结束的计划项，



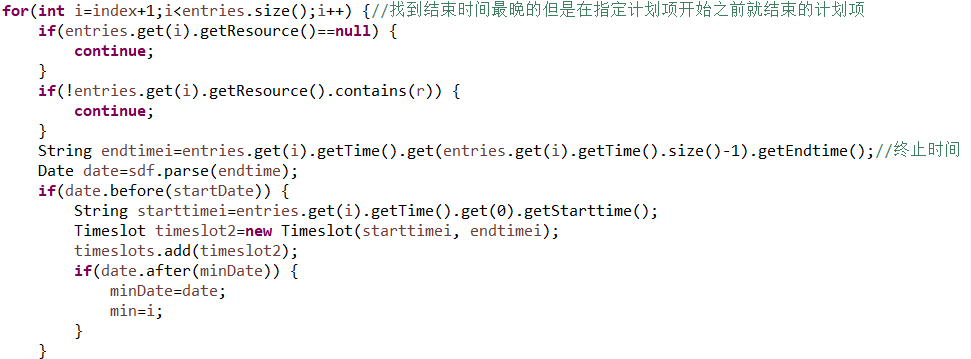
然后将其作为最晚结束时间计划项，遍历之后的计划项，与这个“最晚结束时间”的计划项进行比较，找到结束时间更晚的计划项并更新，就这样在维护一个“最晚结束时间”计划项的过程中完成搜索与判断。

在不存在结束时间早于e的开始时间的计划项时，该种方法可通过findFirstPreEntry函数无法找到计划项，快速的返回false判断

关键代码展示如下：

①快速寻找第一个搜索起点

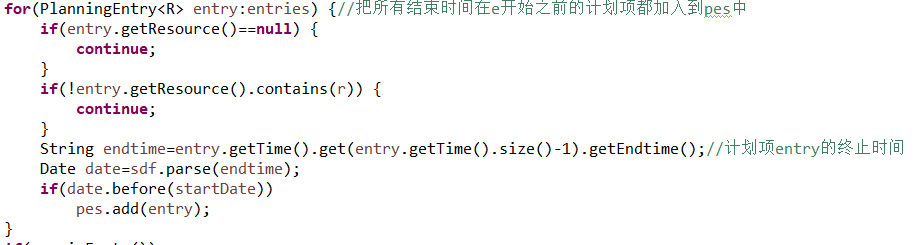
②之后遍历所有计划项，维护“最晚结束”计划项



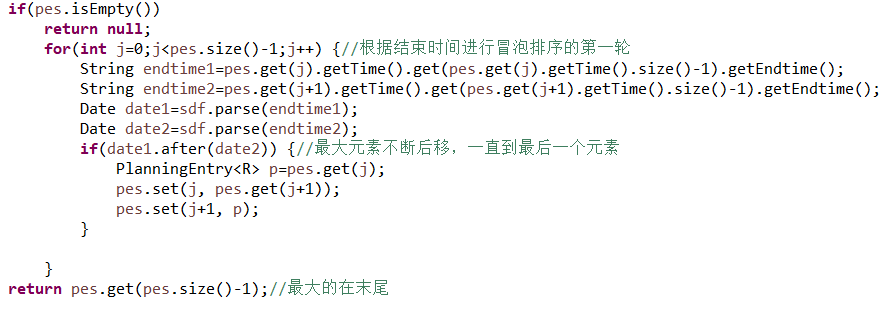
这是一定有返回的计划项的，因为如果findFirstPreEntry找到了一个起点，那说明“最晚结束”计划项是肯定存在的，而结束时间最晚就说明了在该计划项和e之间没有其他计划项存在，满足要求

<2>第二种实现:

①先遍历一次所有计划项,将所有结束时间早于e开始时间的全部保存



②然后利用冒泡排序整个排序



最晚结束时间的计划项位置就已经确定了，直接返回即可。

这种方法相对于第一种而言，需要进行多次遍历搜索，并且进行了排序，耗时可能较多，不过思路清晰，代码简洁，易于维护

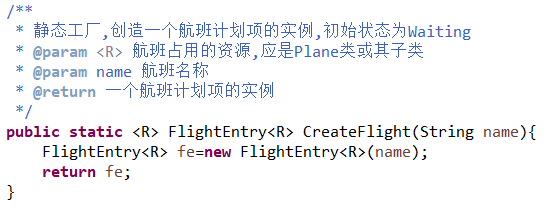
因为这些API函数都是对所有PlanningEntry及其子类适用的，避免了用户为了实现这些功能而进行复杂的函数调用，只需要传入参数即可，因此体现了Facade模式的优点

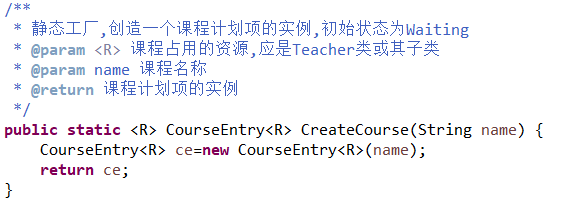
## 设计模式应用

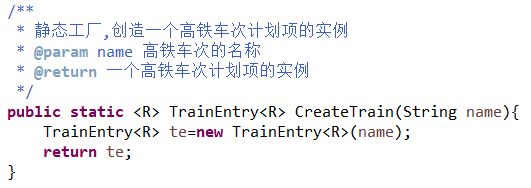
请分小节介绍每种设计模式在你的ADT和应用设计中的具体应用。

### Factory Method

这里为了实现和使用方便而使用静态工厂方法进行子类应用的创建





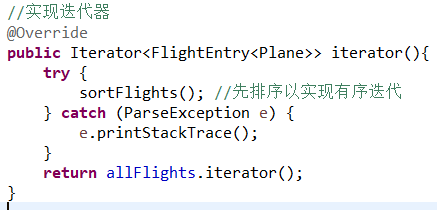


选择返回的实现方式都是已经实现的应用子类，如果将来有更多的实现,可以更换返回的实现类型。

### Iterator

在Board的设计分析时就已经提到过对于Iterator的考虑。航班和高铁中对于抵达和离开二者整合的一个list就是为了Iterator而考量的。之前已经提到过每个Board中都设计了一个sort函数,已经实现了对所有计划项从早到晚的排序，有序存放在List中,于是我们可以直接返回该list的迭代器即可。

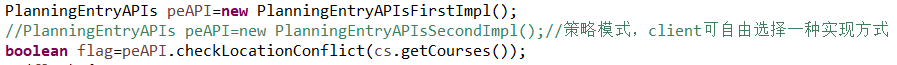
核心就在于每个Board中的sort函数，前面都已经详细介绍过。TrainBoard中为了实现sort还实现了Comparator接口。



### Strategy

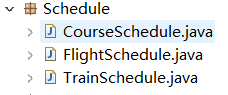
策略模式是针对3.9中的API中的两种实现，用户在使用时可以统一声明变量为PlanningEntryAPIs类，而在new一个具体实例时可自由选择二者之一，甚至在以后扩展多个实现时，灵活选择多者之一。然后调用其中的方法即可。

使用样例如下:



## 应用设计与开发

在每个App设计之前,我都设计了面向具体应用场景的一个schedule，作为对计划项集合的管理。



每个schedule的功能会在下面具体介绍到

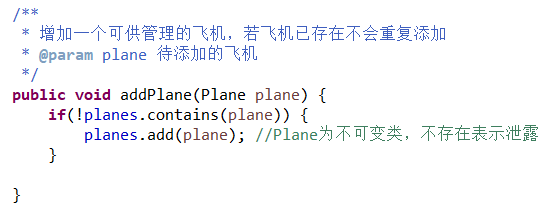
### 航班应用

维护一组飞机、一组机场、一组航班计划项集合，对外提供对其进行管理的功能接口，例如增加位置、删除位置、为航班分配飞机、启动航班等等一系列操作。这样方便简化client端App的使用，只需要调用Schedule中的函数，无需面向特定计划项进行操作。



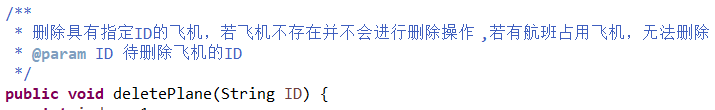
①对于飞机资源进行管理:

如增加一个飞机进行管理:



因为创建飞机需要多个参数,因此令client在外面整合好以后直接传进plane类型的参数

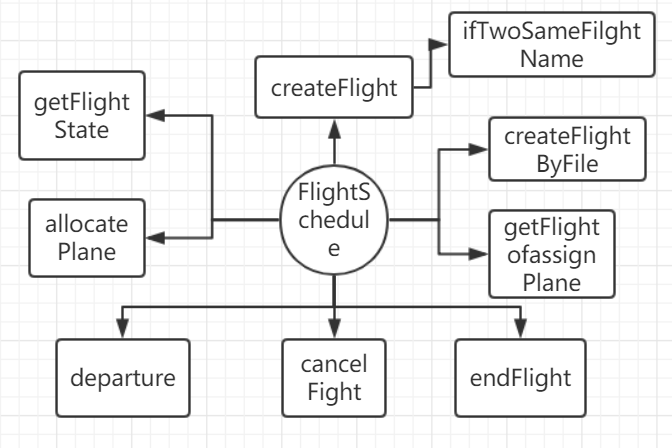
删除管理的飞机:



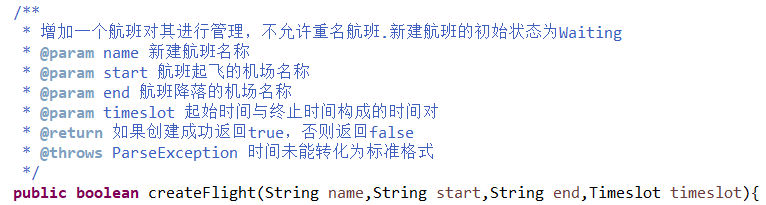
为了简便client的使用,删除飞机时直接指定名称即可,搜索得到飞机的过程在schedule方法内部封装。同时还有相应的get方法

②对location的管理:同样也是add、delete、get等方法，与资源类似，不再赘述

③核心部分是对于航班集合的管理:涉及到的函数展示如下:



<1>createFlight通过传入建立航班需要的航班名称、起点、终点的名称、起始和终止时间来建立航班



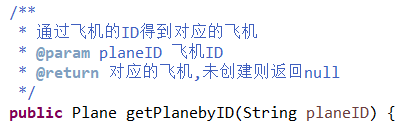
注意，这里的起点和终点要求必须位于FlightSchdule中维护的位置集合之中，如果随便两个位置都可以创建航班，那么对于位置的管理的就失去意义，因此建立航班之前一定要将位置先通过addLocation函数加入FlightSchedule之中,因此这样也可以简化client调用,只需要传入位置名称。其中调用了ifTwoSameFlightName方法,用来在创建航班时比较是否为重名航班，这里之所以分离出一个功能函数,是因为3.12节对于CA0001与CA01类是相同航班名称的要求。

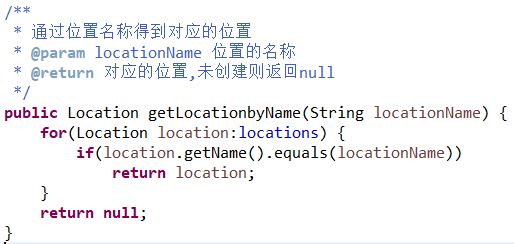
<2>createFlightByFile是3.12基于语法创建航班的实现,只需要传入文件路径即可

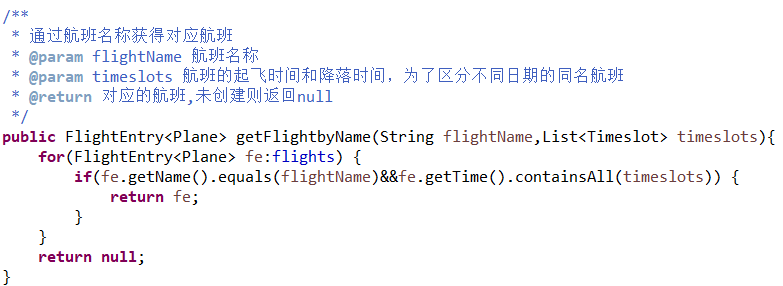
<3>其余基本都是改变航班状态的方法,基本都需要指定航班名称以及航班的出发、降落时间。这里之所以要求将时间传入，是因为根据3.12节要求,航班是允许重名的,而他们之间的区别就在于时间不同，因此要将时间传入，否则可能会出现混乱.

<4>getFlightofassignPlane的功能是找到占用指定飞机的所有航班

④对于飞机、位置、航班都提供一个功能:通过指定名称/ID获得对应的对象,便于client的操作

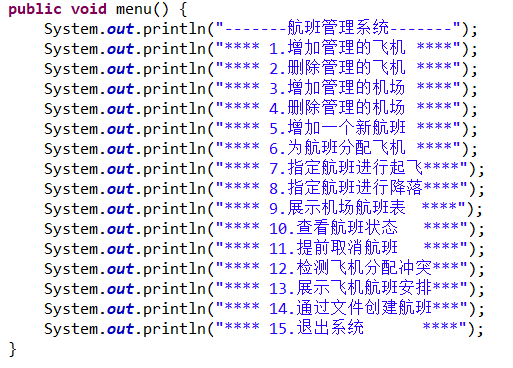






在Schedule部分完成后,client其实就可以很简单地通过创建一个FlightSchedule对象实现各种管理功能。因此我们在App里要做的就是提供一个功能菜单,根据用户的输入，调用FlightSchedule的对应功能即可

菜单展示如下:



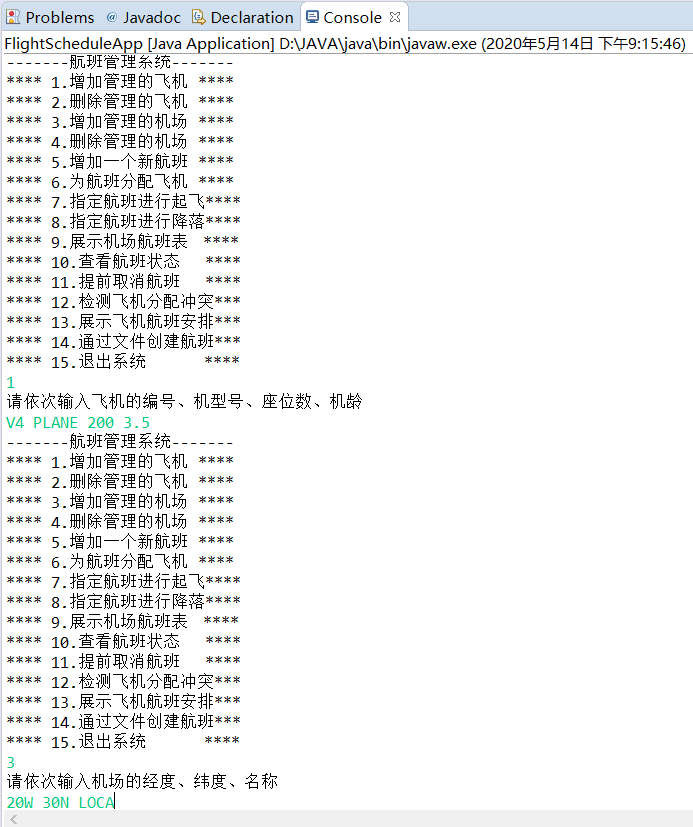
对输入进行switch-case判断即可，使用时需要注意的点其实在前面都涉及到了：

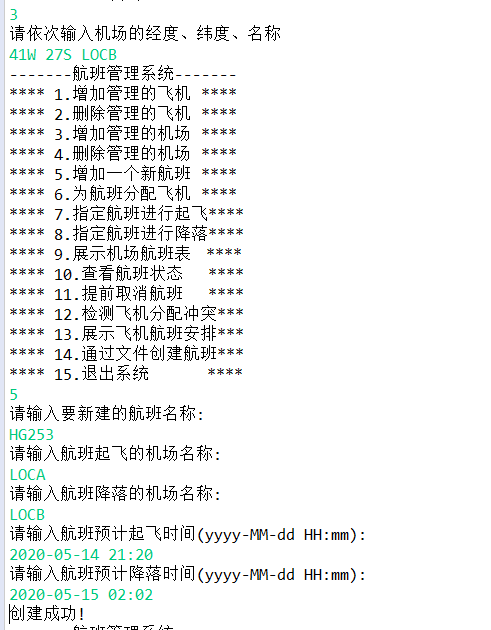
<1>创建航班时的位置应该提前加入管理中,否则创建失败

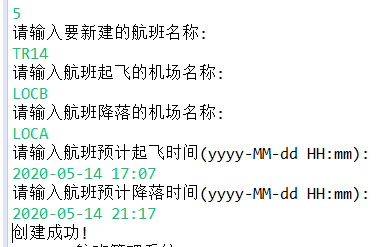
<2>改变航班状态的操作都需要输入相关时间，为了区分同名航班

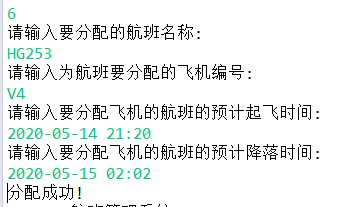
<3>Board展示时是自动获取系统时间作为当前时间,因此为了查看Board效果请尽量在为航班分配时间时接近当时系统时间,否则没有航班满足展示要求

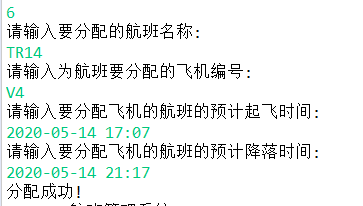
效果展示如下:

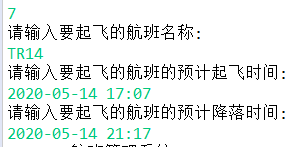


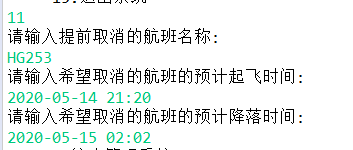


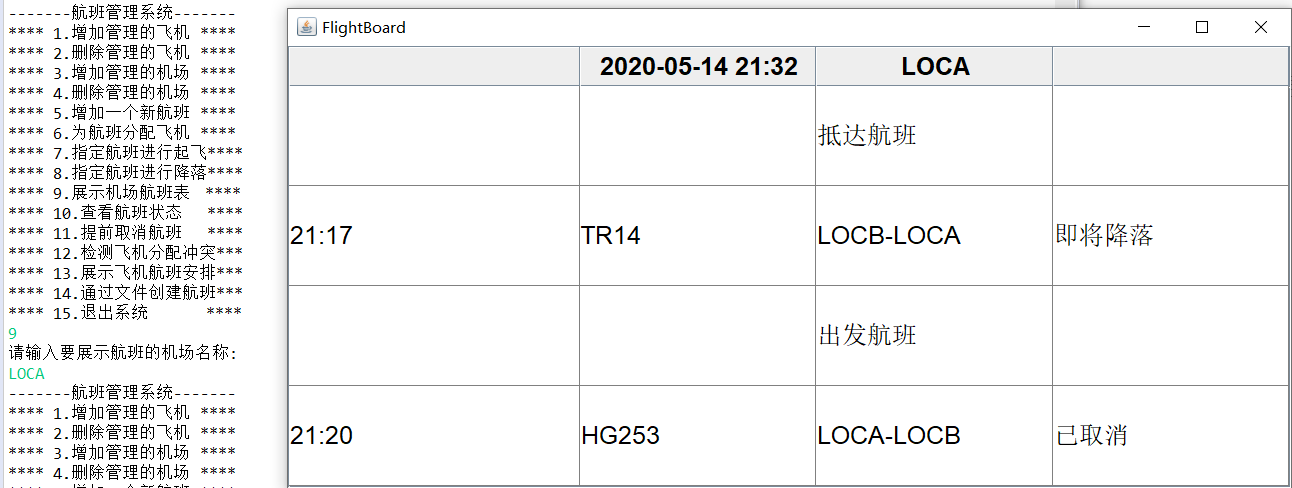






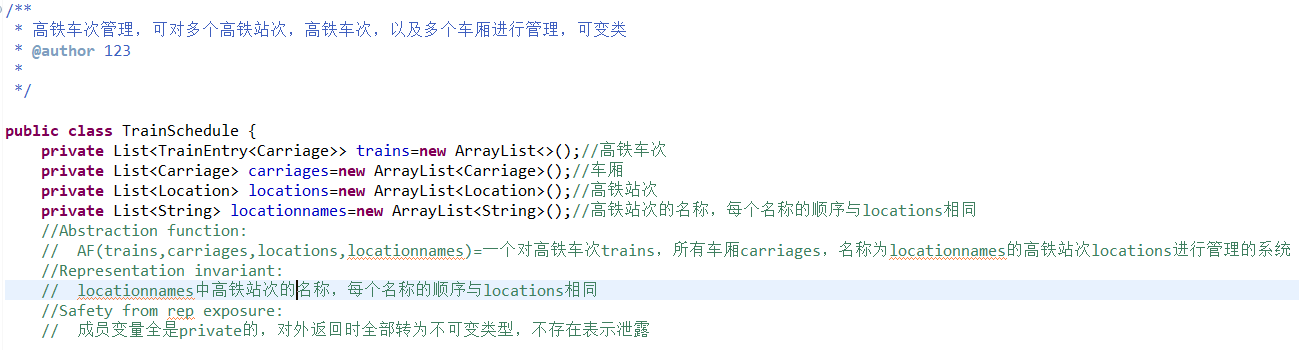






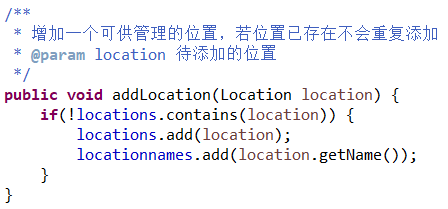
### 高铁应用

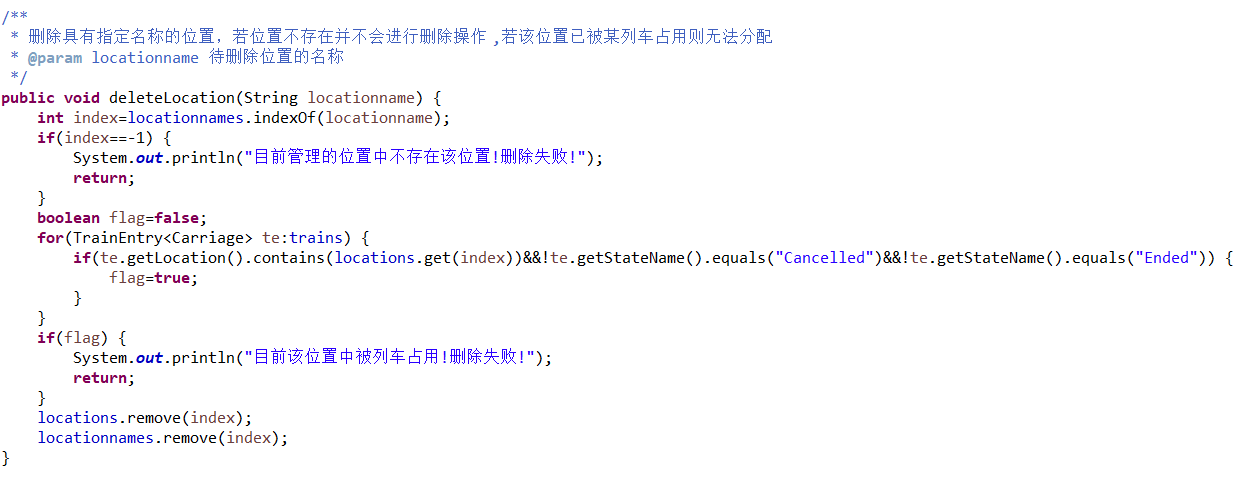
和航班应用非常类似，同样先实现一个Schedule实现管理,便于client简洁调用。而且无需提供读入文件创建的方法，所以略微简单一些。而且这里没有对重名车次的额外要求，因此这里限定:不允许重名车次。



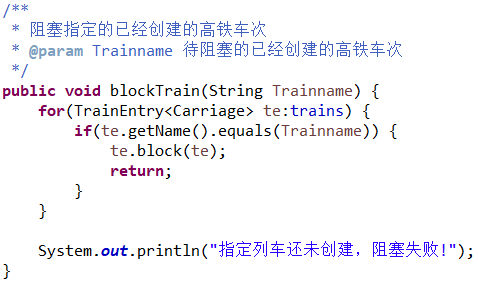
因为为了便于client调用，Location、车厢、高铁都尽量用名称交互，但是高铁的位置太多，逐一比较每个车次的所有位置效率太低，因此这里增加了一个locationnames成员变量,保存所有车厢的名称，查询时直接利用locationnames查询来提高效率。

在add和delete时将locationnames和locations的顺序同步





此外，由于高铁可阻塞，因此提供阻塞车次的功能，其实也就是搜索到车次以后调用高铁计划项的阻塞功能



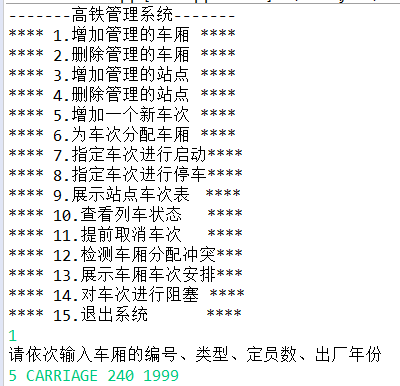
其余的实现和FlightSchedule非常类似，不再赘述

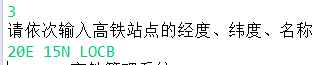
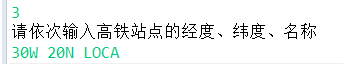
TrainScheduleApp中的结构也与FlightScheduleApp类似，打印功能菜单，利用switch-case处理输入。由于不允许重名,因此对于计划项实行改变状态的操作无需再输入时间。但另外两条要求仍然存在:

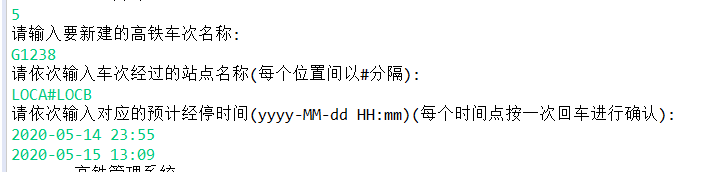
<1>创建高铁时的位置应该提前加入管理中,否则创建失败

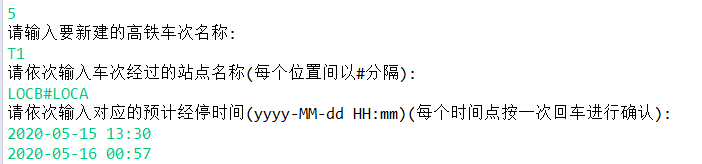
<2>Board展示时是自动获取系统时间作为当前时间,因此为了查看Board效果请尽量在为高铁分配时间时接近当时系统时间,否则没有高铁满足展示要求

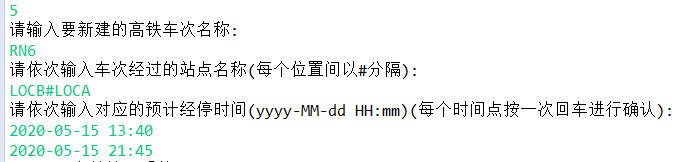
效果展示如下:

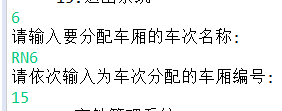
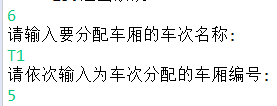
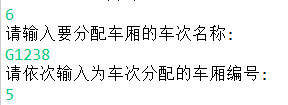


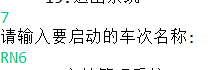
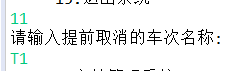








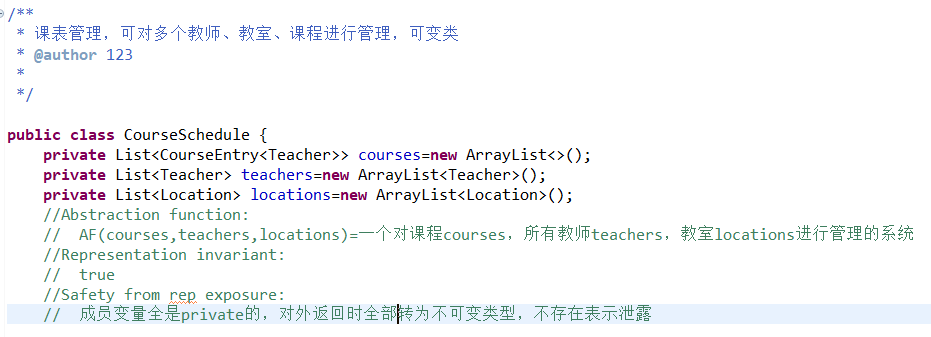






### 3.11.4 课表应用

同前面二者一样，也为App设计一个CourseSchedule，实现对课程、教师、教室的统一管理，简化client调用。



比起前面二者都要简单一些,分别维护三个list来实现管理：如增加、删除、改变课程状态等。同高铁车次的要求一样，不允许重名课程出现

与其他二者相比，增加更改计划项位置的函数，这是由课程计划项本身的特点决定的:设定位置后可以更改



其他函数的功能和前面介绍过的都是很类似的，比如增加位置、开始课程、查看课程状态等等。

这样一来CourseCalendar的实现比较简洁,client只需对一个CourseSchedule的变量进行各种调用即可：打印功能菜单，switch-case处理输入，根据不同case调用CourseCalendar的不同方法。

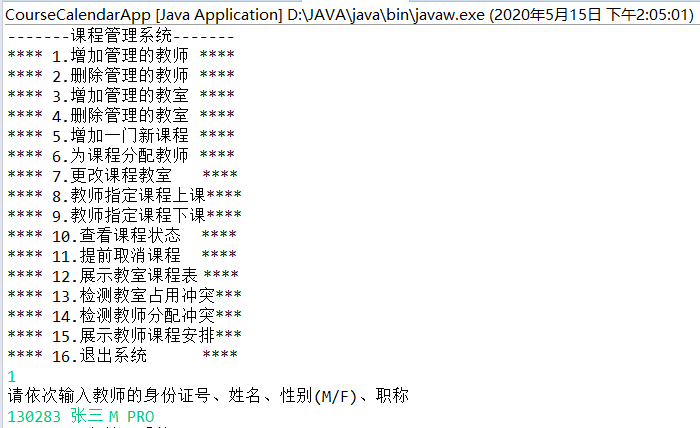
需要注意的点也是一样的：

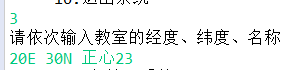
<1>创建课程时的教室应该提前加入管理中,否则创建失败

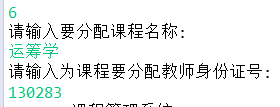
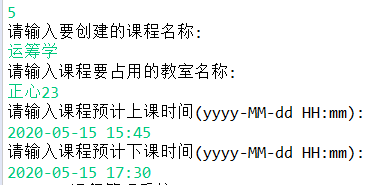
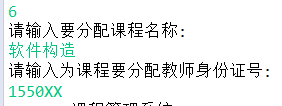
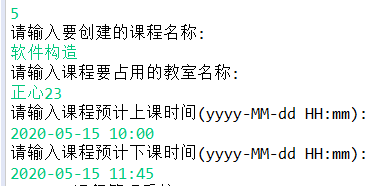
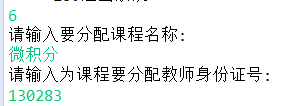
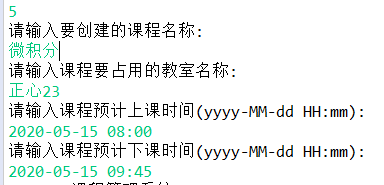
<2>Board展示时是自动获取系统时间作为当前时间,而课程的Board是展示当日所有课程，因此为了查看Board效果请尽量在为课程分配时间时符合测试当时系统时间当日的日期,否则没有课程满足展示要求

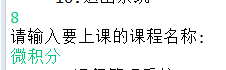
<3>在Resource设计时，Teacher的唯一标识不是名字而是身份证号,因此对老师的操作大多是由身份证号来触发的,这是出于对现实中重名老师的考量

测试效果展示如下:





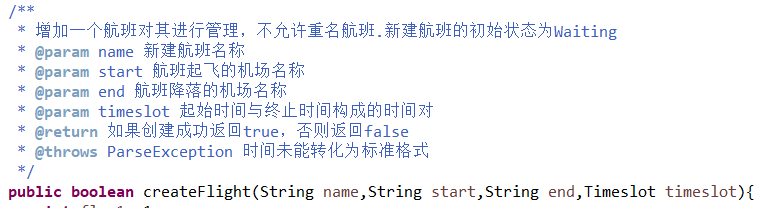






## 基于语法的数据读入

在FlightSchedule中增加函数:createFileByFile,通过读入文件创建航班,返回一个boolean值。当文件里所有航班创建成功时返回true，否则返回false



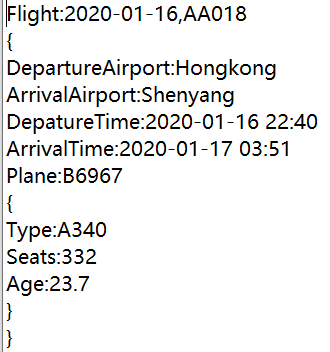
在App中目录展示时增加菜单项:



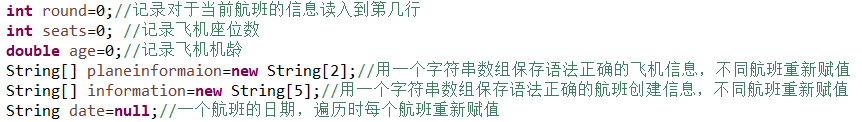
对应case的操作只需要调用该函数即可。

这里再具体讲一下对于文件中具体内容进行语法判断的正则表达式设计。

①首先我们观察文本文件中一个航班构成的单元



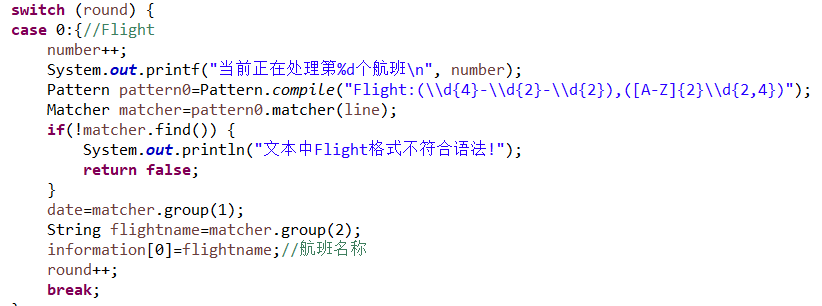
我们可以看到每个航班的信息都占据13行,每一行代表不同信息，其中的{}是无用信息。因此我的设计思路是逐行进行文件读取,记录一个round变量，表示目前读到第几行,round从0开始，每读一行便+1，直到round=12时再重新赋零，循环往复。然后根据round的值设计不同行的正则表达式处理:



而每个航班涉及到的信息个数都是固定的,因此我们在循环开始前都先初始化一下,每个循环对这些量进行赋值。如seats是座位数,information保存航班的名称、起飞机场、降落机场、起飞时间、降落时间等。

②每一行的正则表达式解析

首先，剔除不同Flight之间的空行



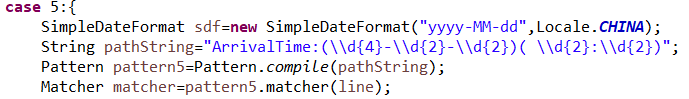
然后根据round当前值确定分支，如第0行,则代表Flight的名称、日期信息，匹配的正则表达式如上所示，如果匹配失败，打印错误信息后直接返回false

其他分支与其类似，这里只将后面的正则表达式列出，其他请详见代码







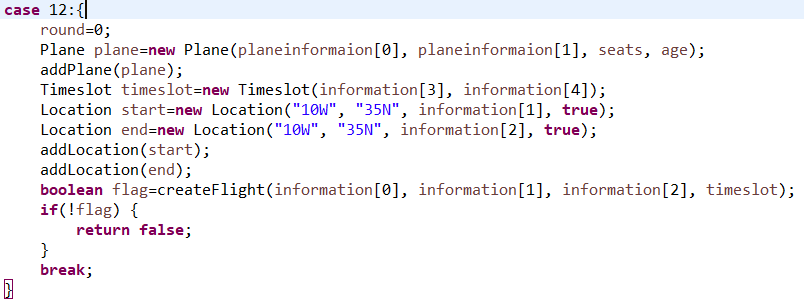












## 应对面临的新变化

只考虑你选定的三个应用的变化即可。

### 变化1

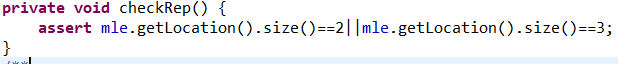
首先是航班变为可经停站点,顶层设计无需变化,对于计划项的修改也是非常少的,只需要改变接口的组合方式就行了。FLightPlanningEntry组合的具体接口发生变化即可,从继承TwoLocation到继承MultipleLocation



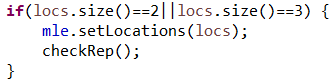
FlightEntry的应用子类只需要将内部维护的用来实现委托的TwoLocationImpl改为MultipleLocationImpl



此外,由于航班最多有一个经停点,有严格限制,因此在checkRep中对位置数进行判断以保证满足要求



在设定位置时也要进行个数的判断

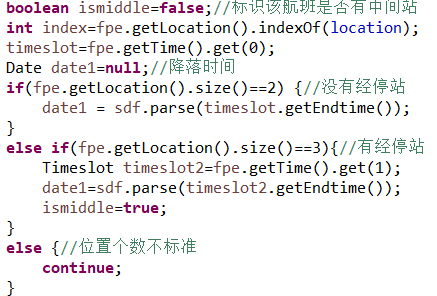


此外，还需要对FlightSchedule中创建航班的部分做简单修改，因为参数从简单地两个位置改为传入位置list



FlightBoard部分需要增添对中间站点的判断,与TrainBoard类似。

为了不改变原有FlightBoard设计中对于起点和终点的判断,这里在遍历时首先确定是否有中间站点,无论是有中间站点还是没有，都利用各自的方法将起点和终点以及对应的时间确定,从而无需修改原有Board中的部分



然后不是修改，而是增加对于中间站点的判断，体现OCP原则



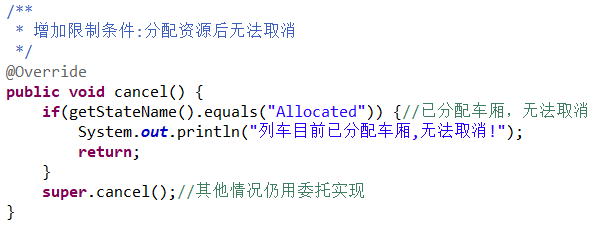
这里的判断逻辑就与高铁中的类似。

因此总结看来,对于计划项的改变时非常少的，只需要变换委托和继承逻辑即可，体现到其他应用上则需要修改一些参数、或是在判断逻辑上增加一条。变化不大。当然，相应的测试也是需要修改一下的，比如FlightSchedule中创建航班的参数需要改变

### 变化2

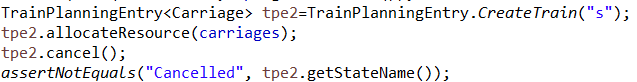
列车分配车厢后无法取消，这个带来的变化更加微小

只需要在应用子类的取消函数上增加一个额外的状态判断条件即可，检查是否已经分配车厢



其余没有什么变化,如果要对该增加限制条件后的函数进行测试,可在对应测试中增加一条策略:





可见变化的代价是非常小的

### 变化3

课程多名教师上课,并有次序。根据piazza里老师的回答，这里的次序由client自己确定。就像高铁车厢分配的顺序由client指定一样。同航班类似，这里对计划项的修改只需要改变委托和继承的逻辑即可



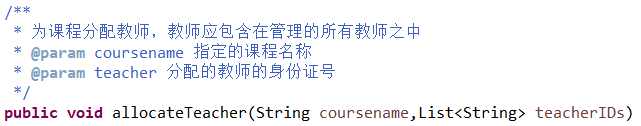
接口组合由继承单资源转为继承多资源

应用子类里的委托由单资源实现转为多资源实现



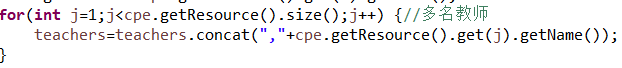
其余没有什么修改

对于CourseSchedule,需要对allocateTeacher进行变化，因为分配的教师参数需要由单个教师的ID转为一个list



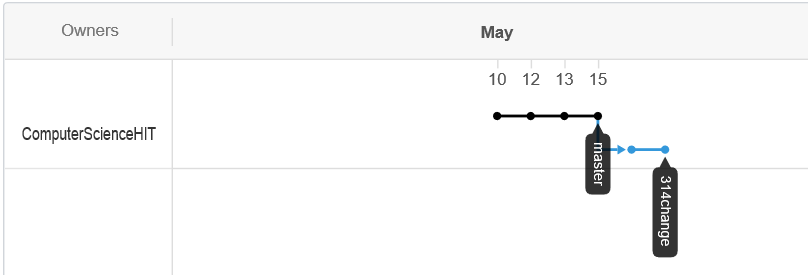
内部实现也无需多做修改，因为计划项应用子类里面本身的功能已经改变为适应多教师分配了。

此外，CourseBoard中可视化部分需要略加修改，因为在展示时需要将所有教师都展示出来，实际也就是增加一个遍历教师集合的过程



由此可见，变化也是很小的，当然，对应的测试用例也要做一定变化，因为某些函数的参数发生了变化

## Git仓库结构



如图，黑色点表示master分支，后面蓝色的为314change分支,由于之前写实验时全在eclipse里面改来改去的，忘记提交到git以及github上了，没能很好地显示进度情况及变化。

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2020-04-24 | 14:00-17:00 | 确定继承结构，屡清思路，实现时间、资源、位置等基础类 | 完成 |
| 2020-04-25 | 15:30-18:00 | 完成PlanningEntry及其各个应用子类 | 未完成，对于高铁阻塞始终有疑惑，state模式尚未掌握 |
| 2020-04-26 | 09:20-11:30 | 完成PlanningEntry及其各个应用子类 | 未完成,本来想把资源、位置等get、set函数全部放在应用子类，但察觉到了重复,state模式未掌握 |
| 2020-04-26 | 14:20-18:00 | 完成PlanningEntry及其各个应用子类 | 完成，将资源、位置等get函数放在Common实现中，并将其实现为抽象类，实现state模式 |
| 2020-04-29 | 14:00-17:35 | 对前面完成部分进行测试、调试纠错、确保正确 | 完成 |
| 2020-04-30 | 19:00-21:40 | 完成各个schedule | 未完成,中途又对block与系统时间的关系产生纠结，最终询问老师，确定实现方式为全手动 |
| 2020-05-02 | 09:00-11:40 | 完成各个schedule | 基本代码完成，尚未通过测试，需要调试 |
| 2020-05-02 | 15:00-17:00 | 测试、调试schedule | 完成 |
| 2020-05-02 | 19:00-23:00 | 完成API | 完成 |
| 2020-05-04 | 08:00-12:00 | 完成Board除可视化外的基础功能 | 排序部分未完成 |
| 2020-05-04 | 15:00-16:20 | 完成Board除可视化外的基础功能 | 完成 |
| 2020-05-05 | 08:00-10:50 | 完成Board可视化 | 仍存在bug |
| 2020-05-05 | 13:00-15:20 | 完成Board可视化 | 完成 |
| 2020-05-06 | 14:00-15:30 | 修改排序代码，完成Board迭代器 | 完成 |
| 2020-05-08 | 14:00-17:00 | 完成App | 完成 |
| 2020-05-08 | 19:30-23:00 | 学习正则表达式，完成语法输入部分 | 五个文件都认为正确，存在bug |
| 2020-05-09 | 10:00-11:20 | 修改语法输入部分 | 发现对于重名航班的要求，修改schedule中创建航班函数，修改正则表达式设计，完成目标 |
| 2020-05-10 | 09:15-11:00 | 完成314change | 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 |
| 实验中进行到State模式以及正则表达式时课程还没有讲到这里，难以进行实验 | 上网根据博客自学 |
| 设计的逻辑，如何更好实现复用性 | 基于实验指导上给出的方案，在编写代码时加深体会，不断思考是不是有可以修改的地方，边写边改 |
| 对于某些设计细节比较纠结和疑惑，如高铁的阻塞到底是根据时间还是认为操作，计划项的启动也是一样 | Piazza中提问、向老师提问 |
| 未接触过可视化 | 自学 |

# 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. 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？

Lab2单独设计ADT时并未仔细考虑复用的问题,有可能造成只能在一种特定条件下才能使用的尴尬。Lab3提炼各个应用场景的共性与差异让我体会到了复用的重要性，在一开始的过程还感觉很繁琐，但其实到应用子类时真正体会到了复用，应用子类都不用写什么额外的代码了。

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？

提醒我们始终注意表示泄露与表示不变量，愿意坚持

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？

API需要针对共性作考量,感觉也需要对ADT有比较高的要求，我就是在思考API时才把资源、位置等加入到顶层接口，体会到了艰辛。

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？

对于state模式非常喜欢,省去了一大堆的if-else判断，代码看起来非常简洁明了，而且修改也非常方便！！

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？

体会到了正则表达式的强大，如此多的语句竟能很快、准确地解析，以前从未这样考虑过，很震撼

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？

觉得尤其是在设计多个ADT时，如何设计这些ADT之间的逻辑关系是非常重要的，好的设计逻辑可以简化很多繁琐代码！！

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？

对client暴露的东西能多抽象就多抽象，但注意在设计时也不要盲目追求抽象，我在设计App时就一直想用FlightPlanningEntry代替FlightEntry实现更高抽象，但这样在设计API时就会发现必须要令FlightPlanningEntry继承PlanningEntry接口才可以。在这里纠结了许多时间，教训就是抽象也是要在“设计能力”范围内的。如果设计无法实现，也要退而求其次才行

1. 关于本实验的工作量、难度、deadline。

感觉量比lab2多了好几个档次,花了太多时间，以前还没写过这么多代码

觉得老师把lab3中删除一些部分会好一些，好多啊！

1. 到目前为止你对《软件构造》课程的评价。

越来越难，越来越抽象，实验的压力很大，非常大!