

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

**目录**

[1 实验目标概述 1](#_Toc39928085)

[2 实验环境配置 1](#_Toc39928086)

[3 实验过程 1](#_Toc39928087)

[3.1 待开发的三个应用场景 1](#_Toc39928088)

[3.2 面向可复用性和可维护性的设计：PlanningEntry<R> 2](#_Toc39928089)

[3.2.1 PlanningEntry<R>的共性操作 2](#_Toc39928090)

[3.2.2 局部共性特征的设计方案 5](#_Toc39928091)

[3.2.2.1 实现可阻塞功能 5](#_Toc39928092)

[3.2.2.2 实现两个位置 5](#_Toc39928093)

[3.2.2.3 实现多个有序的位置 7](#_Toc39928094)

[3.2.2.4 实现可修改的单个位置 8](#_Toc39928095)

[3.2.2.5 实现一组有序资源 9](#_Toc39928096)

[3.2.2.6 实现一个起止时间对 10](#_Toc39928097)

[3.2.2.7 实现一组起止时间对 11](#_Toc39928098)

[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） 12](#_Toc39928099)

[3.2.3.1 实现FlightEntry<R> 12](#_Toc39928100)

[3.2.3.2 实现TrainEntry<R> 14](#_Toc39928101)

[3.2.3.3 实现ActivityEntry<R> 16](#_Toc39928102)

[3.3 面向复用的设计：R 18](#_Toc39928103)

[3.3.1 实现飞机资源Plane 18](#_Toc39928104)

[3.3.2 实现车厢资源Carriage 19](#_Toc39928105)

[3.3.3 实现活动资源Material 21](#_Toc39928106)

[3.4 面向复用的设计：Location 22](#_Toc39928107)

[3.5 面向复用的设计：Timeslot 24](#_Toc39928108)

[3.6 面向复用的设计：EntryState及State设计模式 25](#_Toc39928109)

[3.6.1 状态接口State 26](#_Toc39928110)

[3.6.2 未分配状态Waiting 26](#_Toc39928111)

[3.6.3 已分配状态Allocated 26](#_Toc39928112)

[3.6.4 已启动状态Running 26](#_Toc39928113)

[3.6.5 挂起中状态Blocked 27](#_Toc39928114)

[3.6.6 已完成状态Ended 27](#_Toc39928115)

[3.6.7 已取消状态Cancelled 27](#_Toc39928116)

[3.7 面向应用的设计：Board 28](#_Toc39928117)

[3.7.1 FlightEntry的Board 28](#_Toc39928118)

[3.7.2 TrainEntry的Board 29](#_Toc39928119)

[3.7.3 ActivityEntry的Board 30](#_Toc39928120)

[3.8 Board的可视化：外部API的复用 30](#_Toc39928121)

[3.9 PlanningEntryCollection的设计 31](#_Toc39928122)

[3.10 可复用API设计及Façade设计模式 32](#_Toc39928123)

[3.10.1 检测一组计划项之间是否存在位置独占冲突 32](#_Toc39928124)

[3.10.2 检测一组计划项之间是否存在资源独占冲突 32](#_Toc39928125)

[3.10.3 提取面向特定资源的前序计划项 33](#_Toc39928126)

[3.11 设计模式应用 34](#_Toc39928127)

[3.11.1 Factory Method 34](#_Toc39928128)

[3.11.2 Iterator 35](#_Toc39928129)

[3.11.3 Strategy 36](#_Toc39928130)

[3.12 应用设计与开发 36](#_Toc39928131)

[3.12.1 航班应用 36](#_Toc39928132)

[3.12.2 高铁应用 39](#_Toc39928133)

[3.12.3 学习活动应用 43](#_Toc39928134)

[3.13 基于语法的数据读入 46](#_Toc39928135)

[3.14 应对面临的新变化 47](#_Toc39928136)

[3.14.1 变化1 47](#_Toc39928137)

[3.14.2 变化2 48](#_Toc39928138)

[3.14.3 变化3 49](#_Toc39928139)

[3.15 Git仓库结构 50](#_Toc39928140)

[4 实验进度记录 51](#_Toc39928141)

[5 实验过程中遇到的困难与解决途径 51](#_Toc39928142)

[6 实验过程中收获的经验、教训、感想 52](#_Toc39928143)

[6.1 实验过程中收获的经验和教训 52](#_Toc39928144)

[6.2 针对以下方面的感受 52](#_Toc39928145)

# 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

# 实验环境配置

简要陈述你配置本次实验所需环境的过程，必要时可以给出屏幕截图。

特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

Lab3的URL：

<https://github.com/ComputerScienceHIT/Lab3-1183710113.git>

# 实验过程

## 待开发的三个应用场景

应用场景：航班管理、高铁车次管理、学习日程管理

共性：

都有创建一个新计划项的功能；

都可以为对应的计划项申请资源。

都可以启动、取消、完成指定的计划项；

每个计划项都有状态，应用可以获得这个状态；

获得每个计划项的名字。

差异：

航班计划项的位置数量只有两个，包括起止点，高铁计划项的有一个起点、一组中间位置和一个终点，而学习活动只有一个位置；

这三个计划项的位置都可以提前设定，但航班计划项和高铁计划项的位置设定后不可更改，而学习活动的位置在设定后可以更改；

航班计划项的飞机资源是单个可区分的，高铁计划项的资源是多个有序排列的一组可区分的车厢资源，学习活动的资源是多个不带次序的不可区分的材料资源；

航班计划项和学习活动计划项的时间表是一个起止时间对，并且是不可阻塞的，而高铁计划项的时间表是一组起止时间对，并且可以阻塞，这三个计划项的时间在创建的时候都可以设定。

## 面向可复用性和可维护性的设计：PlanningEntry<R>

### PlanningEntry<R>的共性操作

1. 实现PlanningEntry<R>

|  |  |
| --- | --- |
| 方法 | 方法作用 |
| public static <R> PlanningEntry<R> flightEntry (String name, Location departure, Location arrival, String start, String end) | 静态工厂方法，创建一个航班计划项 |
| public static <R> PlanningEntry<R> trainEntry (String name, List<Location> locationList, List<String> times) | 静态工厂方法，创建一个高铁计划项 |
| public static <R> PlanningEntry<R> activityEntry (String name, String start, String end) | 静态工厂方法，创建一个活动计划项 |
| public boolean allocate() | 将计划项状态设置为已分配状态 |
| public boolean running() | 将计划项状态设置为运行状态 |
| public boolean cancel() | 将计划项状态设置为已取消状态 |
| Public boolean End() | 将计划项状态设置为完成状态 |
| public String getName() | 获取计划项的名字 |
| public String getState() | 获得计划项状态 |

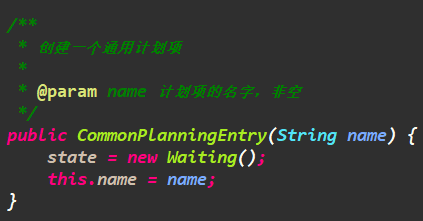
1. 实现CommonPlanningEntry<R>

（1）数据域：保存计划项状态的state和表示计划项名字的name。

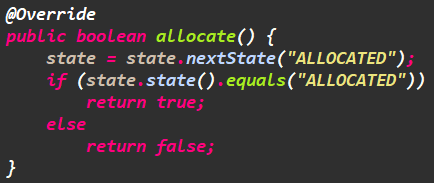


（2）方法：

① 构造方法：把计划项的状态设为未分配（WAITING），计划项命名为name。



② allocate()、running()、cancel()、end()方法的实现都是调用state.nextState()方法，然后判断返回的状态是否是目的状态，如果是，返回true，否则说明不能转移到该状态，返回false。实现样例如下：



③ getState()方法调用state.state()方法，获得状态。

1. 实现计划项集合类PlanningEntryCollection<R>

（1）数据域：计划项列表



（2）方法：

|  |  |
| --- | --- |
| 方法 | 实现 |
| public boolean addEntries(PlanningEntry<R> entry) | 判断计划项是否为空，若为空返回false，不为空则加入到列表返回true |
| public List<PlanningEntry<R>> getEntries() | 用Collections.unmodifiableList()转换成不可修改的列表 |
| public Iterator<PlanningEntry<R>> iterator() | 返回entries.iterator() |
| public void sort() | 调用entries.sort()对计划项排序 |

（3）AF、RI、Safety from rep exposure：

Abstraction function:

AF(entries)=现实中的计划项清单

Representation invariant:

entries!=null

Safety from rep exposure:

所有的数据域都是私有的用final限定

获得entries时使用Collections.unmodifiableList()转化为不可更改的list返回

1. 测试：

（1）测试静态方法：

Test strategy

测试flightEntry()

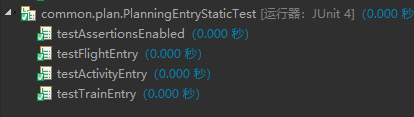
通过getName()、getState()、getDeparture()、getArrival()、getTimeslot()观察

测试trainEntry()

通过getName()、getState()、getLocations()、getTimeslot()观察

测试activityEntry()

通过getName()、getState()、getLocation()、getTimeslot()观察



（2）测试实例方法：

Test strategy

测试allocate()

此时的状态为WATIING或ALLOCATED，此时的状态不为WAITING和ALLOCATED

测试running()

此时状态为ALLOCATED或BLOCKED或RUNNING，此时状态不为ALLOCATED和BLOCKED和RUNNING

测试cancel()

此时状态为WATIING或ALLOCATED或BLOCKED或CANCELLED，此时状态不为WATIING和ALLOCATED和BLOCKED或CANCELLED

测试end()

此时状态为RUNNING或ENDED，此时状态不为RUNNING和ENDED

测试getName()

测试是否返回计划项的名称

测试getState()

测试是否正确返回计划项的状态

（3）测试PlanningEntryCollection

Test strategy

测试AddEntries()

添加的计划项未null，计划项不为null

测试getEntries()

当没有计划项时，当有计划项时

测试sort()

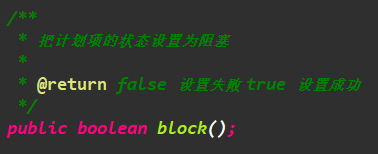
测试能够按照时间的先后顺序给计划项排序



### 局部共性特征的设计方案

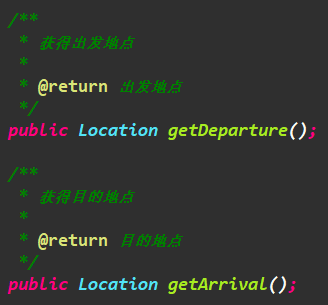
#### 实现可阻塞功能

定义具有阻塞功能的接口BlockableEntry，当计划项是可以被阻塞的时候实现此接口，接口声明方法如下：



#### 实现两个位置

1. 定义表示出发和抵达两个地点的接口TwoLocationEntry，对与选择的航班计划项可以实现此接口，方法声明如下：



1. 定义一个immutable的实现该接口的类TwoLocation。

（1）数据域：表示开始位置和结束位置。



（2）方法：

|  |  |
| --- | --- |
| 方法 | 作用 |
| public TwoLocation() | 构造方法 |
| public Location getDeparture() | 获得出发地点 |
| public Location getArrival() | 获得到达地点 |
| public String toString() | 把存储的位置按照“departure->arrival”的形式输出 |
| public boolean equals() | 判断两个对象是否相同 |

（3）AF、RI、Safety from rep exposure：

Abstraction function:

AF(departure,arrival)=现实中从departure位置到arrival位置

Representation invariant:

departure!=null

arrival!=null

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

1. 测试：

（1）接口方法：

Test strategy

测试getDeparture

测试返回的位置是否和期望相同

测试getArrival

测试返回的位置是否和期望相同

（2）子类型方法：

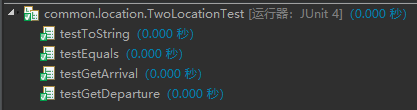
Test strategy

测试toString()

测试返回的字符串是否与期望的相同

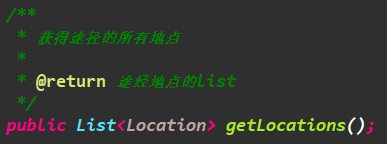
测试equals()

相同的两个位置，两个位置不相同



#### 实现多个有序的位置

1. 定义一个表示多个不可更改的位置的接口MultipleLocationEntry，对于选择的高铁计划项可以实现此接口，方法声明如下：

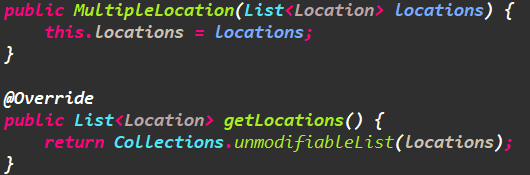


1. 定义一个immutable的实现该接口的类MultipleLocation。

（1）数据域：表示一组有序的位置



（2）方法：只有一个构造方法和一个实现的接口定义的方法。



（3）AF、RI、Safety from rep exposure：

Abstraction function:

AF(locations)=以locations中的对象为位置且按顺序排列的一组位置

Representation invariant:

locations!=null

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

返回位置列表时使用Collections.unmodifiableList()返回一个不可修改的列表

1. 测试：

Test strategy

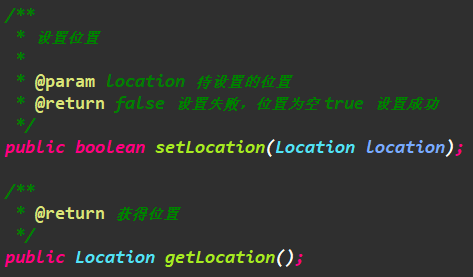
测试getLocations()

测试返回的位置列表与期望是否相等



#### 实现可修改的单个位置

1. 定义一个表示可变更的单个位置的接口ModifiableSingleLocationEntry，对于选择的高铁计划项可实现此接口，方法声明如下：



1. 定义一个mutable的实现该接口的类ModifiableSingleLocation。

（1）数据域：表示一个位置。



（2）方法：

实现接口中的方法，public boolean setLocation(Location location)：当location是null是返回false，否则把当前的位置改为location返回true。

（3）AF、RI、Safety from rep exposure：

Abstraction function:

AF(location)=一个现实中的位置

Representation invariant:

location!=null

Safety from rep exposure:

所有的数据域都是私有的

1. 测试：

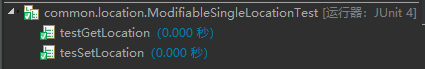
Test strategy

测试setLocation()

位置为空，位置不为空

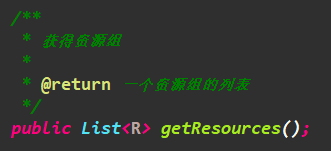
测试getLocation()

测试返回的位置是否和预期相同



#### 实现一组有序资源

1. 定义表示一个可复用的有序的资源组的接口MultipleSortedResourceEntry<R>，对于选择的高铁计划项可实现此接口。方法声明如下：

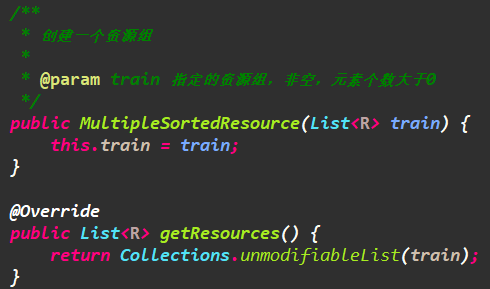


1. 定义一个immutable的实现该接口的类MultipleSortedResource<R>。

（1）数据域：表示一组有序的资源。



（2）方法：



（3）AF、RI、Safety from rep exposure：

Abstraction function:

以train中的顺序排好序的，有train.size()个个体的一组资源

Representation invariant:

train!=null

train.size()>0

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

获得资源时用Collections.unmodifiableList()转化为不可变的List输出

1. 测试：

Test strategy

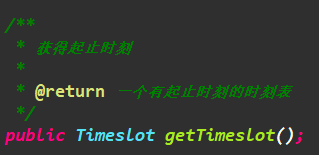
测试getResources()

测试返回的资源是否与期望的相等



#### 实现一个起止时间对

1. 定义一个代表可被预设的起止时刻的时刻表的接口PresetSingleTimeslotEntry，对于选择的航班计划项和学习活动计划项可实现此接口，方法声明如下：

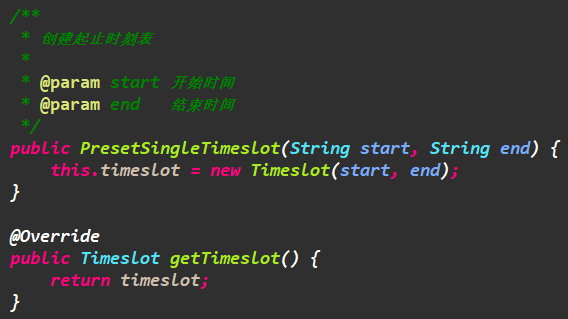


1. 定义一个immutable的实现该接口的类PresetSingleTimeslot。

（1）数据域：表示一个起止时间对。



（2）方法：



（3）AF、RI、Safety from rep exposure：

Abstraction function:

AF(timeslot)=计划项运行过程中的起止时刻

Representation invariant:

timeslot!=null

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

1. 测试：

Test strategy

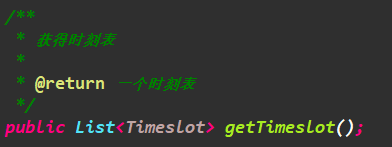
测试getTImeslot()

测试返回的时间表是否和预期的相等



#### 实现一组起止时间对

1. 定义一个表示被预设好的一组有序的时间表的接口PresetMultipleTimeslotEntry，对于高铁计划项可实现此接口，方法声明如下：

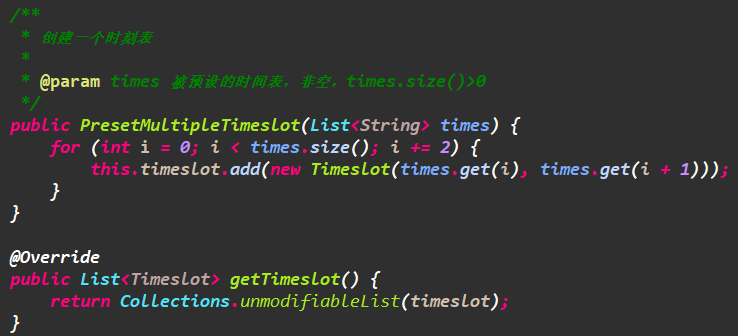


1. 定义一个immutable的实现该接口的类PresetMultipleTimeslot。

（1）数据域：表示一组有序时间对的列表。



（2）方法：



（3）AF、RI、Safety from rep exposure：

Abstraction function:

以timeslot中的顺序排好序的，有timeslot.size()个个体的一组资源

Representation invariant:

timeslot!=null

timeslot.size()>0

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

获得资源时用Collections.unmodifiableList()转化为不可变的List输出

1. 测试：

Test strategy

测试getTImeslot()

测试返回的时间表是否和预期的相等

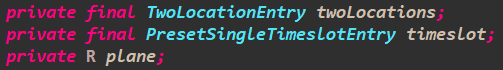


### 面向各应用的PlanningEntry子类型设计（个性化特征的设计方案）

#### 实现FlightEntry<R>

继承抽象类CommonPlanningEntry，实现接口TwoLocationEntry, PresetSingleTimeslotEntry。

1. 数据域：表示起止位置的twoLocations，表示起止时刻表的timeslot，表示所占用的资源的plane。



1. 方法：

（1）构造方法：调用父类CommonPlanningEntry的构造方法，为计划项命名，然后设定twoLocations和timeslot为给定起止位置和时刻表。

（2）public boolean allocatePlane(R plane)：调用父类的allocate()方法，判断返回值，如果返回false，说明当前的状态不可为计划项分配资源，直接返回false，否则说明可以分配资源，把参数plane赋给计划项资源this.plane，然后返回true表示分配成功。

（3）public R getResource()：获得资源，直接返回plane。

（4）public Location getDeparture()：获得出发位置。

public Location getArrival()：获得到达位置。

委托给twoLocations，分别调用twoLocations.getDeparture()和twoLocations.getArrival()获得起止位置。

（5）public Timeslot getTimeslot()：委托给timeslot，调用timeslot.getTimeslot()获得起止时间对。

（6）public int compareTo(PlanningEntry<R> entry)：比较两个计划项的开始时间，当this在entry之前开始，返回-1，当this在entry之后开始，返回1，两者同时开始，返回0。

1. AF、RI、Safety from rep exposure：

Abstraction function:

AF(twoLocations, timeslot, plane)=一个timeslot.getStartTime()从

twoLocations.getDeparture()出发timeslot.getEndTime()到达twoLocations.getArrival()的航班

Representation invariant:

twoLocations!=null

timeslot!=null

Safety from rep exposure:

twoLocations,timeslot数据域都是私有的用final限定

plane是私有的

1. 测试：

Test strategy

测试allocatePlane()

状态可分配资源，状态不可分配资源

测试getResource()

测试返回的资源与期望的是否相同

测试getDeparture()和getArrival()

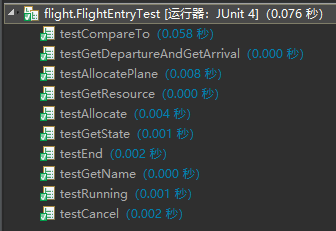
测试返回的位置是否与期望的相同

测试getTimeslot()

测试返回的时刻表是否和预期的相同

测试compareTo()

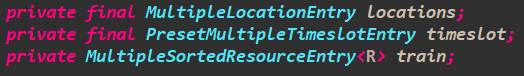
在pe之前开始，在pe之后开始，在pe同时开始



#### 实现TrainEntry<R>

继承抽象类CommonPlanningEntry，实现接口MultipleLocationEntry, PresetMultipleTimeslotEntry, MultipleSortedResourceEntry<R>, BlockableEntry。

1. 数据域：表示途径站点的locations，表示时间表的timeslot，表示所使用的资源组的train。



1. 方法：

（1）构造方法：调用父类CommonPlanningEntry的构造方法，为计划项命名，然后设定locations和timeslot为给定途径位置和时刻表。

（2）public boolean allocateTrain(List<R> train)：调用父类的allocate()方法，判断返回值，如果返回false，说明当前的状态不可为计划项分配资源，直接返回false，否则说明可以分配资源，把参数train赋给计划项资源this.train，然后返回true表示分配成功。

（3）public List<Location> getLocations()：委托给locations，直接调用方法locations.getLocations()获得途径的位置列表。

（4）public List<R> getResources()：调用train.getResources()得到一个不可更改的表示资源组的列表。

（5）public List<Timeslot> getTimeslot()：委托给timeslot，直接调用方法timeslot.getTimeslot()获得高铁计划项的时间表。

（6）public boolean block()：调用state.nextState()方法，判断返回的状态是否是目的状态，如果是，返回true，否则说明不能转移到该状态，返回false。

（7）public int compareTo(PlanningEntry<R> entry)：与FlighEntry.compareTo()相同，比较两个计划项的开始时间，当this在entry之前开始，返回-1，当this在entry之后开始，返回1，两者同时开始，返回0。

1. AF、RI、Safety from rep exposure：

Abstraction function:

AF(locations, timeslot, train)=一个以timeslot为时刻表，经过locations中的车站，使用train中的车厢资源的高铁车次

Representation invariant:

locations!=null

timeslot!=null

Safety from rep exposure:

locations,timeslot数据域都是私有的用final限定

train是私有的

1. 测试：

Test strategy

测试allocateTrain()

状态可分配资源，状态不可分配资源

测试getLocations()

测试返回的位置列表和预期的是否相同

测试getResources()

测试返回的资源列表和预期的是否相同

测试getTimeslot()

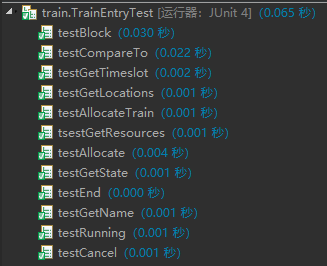
测试返回的时刻表和预期的是否相同

测试block()

此时状态为RUNNING或BLOCKED，此时状态不为RUNNING和BLOCKED

测试compareTo()

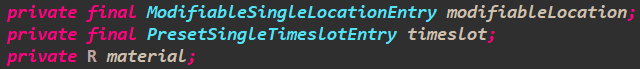
在pe之前开始，在pe之后开始，在pe同时开始



#### 实现ActivityEntry<R>

继承抽象类CommonPlanningEntry，实现两个接口ModifiableSingleLocationEntry, PresetSingleTimeslotEntry。

1. 数据域：表示可变位置的modifiableLocation，表示起止时刻的timeslot，表示所使用的资源的material。



1. 方法：

（1）构造方法：调用父类CommonPlanningEntry的构造方法，为计划项命名，然后设定modifiableLocation和timeslot为给定起止位置和时刻表。

（2）public boolean allocateMaterial(R material)：调用父类的allocate()方法，判断返回值，如果返回false，说明当前的状态不可为计划项分配资源，直接返回false，否则说明可以分配资源，把参数material赋给计划项资源this.material，然后返回true表示分配成功。

（3）public R getMaterial()：获得资源，直接返回material。

（4）public Timeslot getTimeslot()：委托给timeslot，调用timeslot.getTimeslot()方法。

（5）public boolean setLocation(Location location)：先判断计划项的状态，如果计划项的状态是WAITING（待分配）或者ALLOCATED（已分配），说明计划项未开始，可以进行位置设定，将任务委托给modifiableLocation，调用modifiableLocation.setLocation(location)设定位置，

（6）public Location getLocation()：委托给modifiableLocation，调用modifiableLocation.getLocation()方法获得位置。

（7）public int compareTo(PlanningEntry<R> entry)：与前两个相同，比较两个计划项的开始时间，当this在entry之前开始，返回-1，当this在entry之后开始，返回1，两者同时开始，返回0。

1. AF、RI、Safety from rep exposure：

Abstraction function:

AF(locations, timeslot, material) =一个以timeslot为起止时间，在modifiableLocation中的位置，使用material资源的活动计划项

Representation invariant:

modifiableLocation!=null

timeslot!=null

Safety from rep exposure:

locations,timeslot数据域都是私有的用final限定

material是私有的

1. 测试：

Test strategy

测试allocateMaterial()

状态可分配资源，状态不可分配资源

测试getMaterial()

测试得到的资源是否与期望的相同

测试getTimeslot()

测试得到的时间表是否与期望的相同

测试setLocation()

位置为空，位置不为空

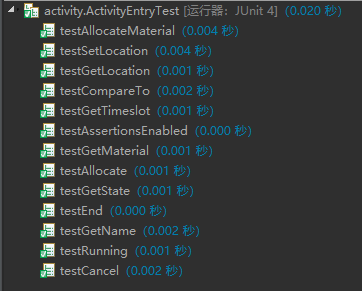
状态可更改位置，状态不可更改位置

测试getLocation()

测试得到的位置是否与期望的相同

测试compareTo()

在pe之前开始，在pe之后开始，在pe同时开始



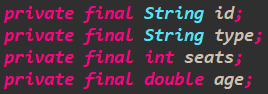
## 面向复用的设计：R

分别实现FlightEntry、TrainEntry、ActivityEntry所使用的资源类型：Plane，Carriage、Material。

### 实现飞机资源Plane

一个immutable的类，表示飞机资源

1. 数据域：表示飞机编号的id，表示飞机机型的type，表示座位数的seats，表示飞机机龄的age。



1. 方法：

|  |  |
| --- | --- |
| 方法 | 实现 |
| public Plane(String id, String type, int seats, double age) | 构造方法，创建一个飞机对象，为数据域赋值 |
| public String getID() | 获得飞机编号，直接返回id |
| public String getType() | 获得飞机机型，直接返回type |
| public int getSeats() | 获得飞机座位数，直接返回seats |
| public double getAge() | 获得飞机机龄，直接返回age |
| public int hashCode() | 分配哈希值，返回由id生成的哈希值 |
| public boolean equals (Object plane) | 判断两个对象的数据域是否都相等，如果都相等，返回true，否则返回false。 |

1. AF、RI、Safety from rep exposure：

Abstraction function:

AF(id, type, seats, age)=以id为编号，type类型，有seats个座位，机龄为age的一架飞机

Representation invariant:

id != null

type != null

seats > 0

age >= 0

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

1. 测试：

Test strategy

测试getID()

测试得到的ID是否和预期的相等

测试getType()

测试得到的类型是否和预期的相同

测试getSeats()

测试得到的座位数是否和预期的相等

测试getAge()

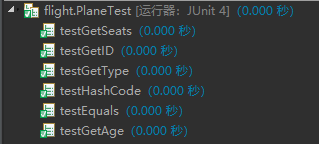
测试得到的机龄是否和预期的相等

测试hashCode()

测试得到的哈希值是否和预期大的相等

测试equals()

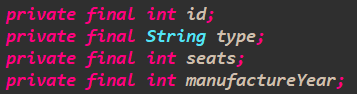
两架飞机是同一架飞机，两架飞机不是同一架飞机



### 实现车厢资源Carriage

一个immutable的类，表示车厢资源

1. 数据域：表示车厢编号的id，表示车厢类型的type，表示定员数的seats，表示生产年份的manufactureYear。



1. 方法：

|  |  |
| --- | --- |
| 方法 | 实现 |
| public Carriage(int id, String type, int seats, int manufactureYear) | 构造方法，创建一个车厢对象，为数据域赋值 |
| public int getID() | 获得车厢的编号，直接返回id |
| public String getType() | 获得车厢的类型，直接返回type |
| public int getSeats() | 获得定员数，直接返回seats |
| public int getManufactureYear() | 获得生产年份，直接返回manufactureYear |
| public int hashCode() | 分配哈希值，返回类型由type生成的哈希值 |
| public boolean equals (Object carriage) | 判断两个对象的数据域是否都相等，如果都相等，返回true，否则返回false。 |

1. AF、RI、Safety from rep exposure：

Abstraction function:

AF(id, type, seats,manufactureYear)=以id为编号，type类型，有seats个座位，出场年份为manufactureYear的一个火车车厢

Representation invariant:

id > 0

type != null

seats > 0

age >= 0

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

1. 测试：

Test strategy

测试getID()

测试得到的编号是否和预期的相等

测试getType()

测试得到的类型是否和预期的相同

测试getSeats()

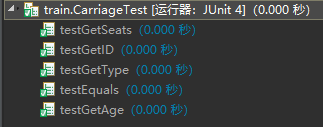
测试得到的座位数是否和预期的相等

测试getAge()

测试得到的生产年份是否和预期的相等

测试equals()

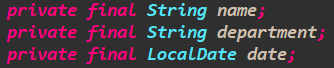
两车厢是同一车厢，两车厢不是同一个车厢



### 实现活动资源Material

一个immutable的类，表示学习材料资源

1. 数据域：表示材料名称的name，表示发布部门的department，表示发布日期的date。



1. 方法：

|  |  |
| --- | --- |
| 方法 | 实现 |
| public Material(String name, String department, String date) | 构造方法，创建一个学习材料对象，为数据域赋值 |
| public String getName() | 获得材料名称，直接返回name |
| public String getDepartment() | 获得发布部门，直接返回department |
| public String getDate() | 获得发布日期，把date转成字符串返回 |
| public int hashCode() | 分配哈希值，返回由name生成的哈希值 |
| public boolean equals(Object material) | 判断两个对象的数据域是否都相等，如果都相等，返回true，否则返回false。 |

1. AF、RI、Safety from rep exposure：

Abstraction function:

AF(name, department, date)=以name为名称，发布部门为department，发布日期为date的学习材料

Representation invariant:

name != null

department != null

date != null

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

获得发布日期时转化为字符串输出

1. 测试：

Test strategy

测试getName()

测试返回的名字与期望是否相同

测试getDepartment()

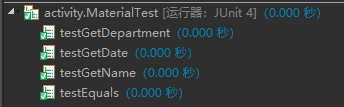
测试返回的名字与期望是否相同

测试getDate()

测试返回的时间与期望是否相同

测试equals()

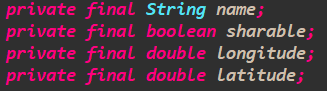
相同的资源，不同的资源



## 面向复用的设计：Location

一个immutable的类，表示位置

1. 数据域：表示地点名称的name，表示地点是否可共享的sharable，表示经度的longitude和表示纬度的latitude。



1. 方法：

|  |  |
| --- | --- |
| 方法 | 实现 |
| public Location(String place, boolean sharable) | 构造方法，创建一个位置对象，把名字和共享性赋给数据域，然后把经纬度都赋为0 |
| public Location(String place, boolean sharable, double longitude, double latitude) | 构造方法，创建一个位置对象，把数据域按照传入的参数设定。 |
| public String getName() | 获得地点的名字，直接返回name |
| public boolean isSharable() | 获得地点的共享性，直接返回sharable |
| public double getLongitude() | 获得经度，直接返回longitude |
| public double getLatitude() | 获得纬度，直接返回latitude |
| public int hashCode() | 分配哈希值，返回由name生成的哈希值 |
| public boolean equals(Object location) | 判断两个对象数据域书否均相等，如相等返回true，否则返回false |

1. AF、RI、Safety from rep exposure：

Abstraction function:

AF(name, sharable)=以name为名称，是否可共享为sharable的位置

AF(name, sharable, longitude,latitude)=以name为名称，是否可共享为sharable，东经longitude，北纬latitude的位置

Representation invariant:

longitude>=0 && longitude<=180

latitude>=0 && latitude<=90

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

1. 测试：

Test strategy

测试getName()

测试返回的名字与期望是否相同

测试isSharable()

测试返回的可共享性与期望是否相同

测试getLongitude()

测试返回的经度与期望是否相同

测试getLatitude()

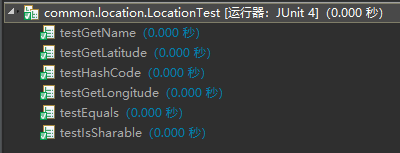
测试返回的纬度与期望是否相同

测试hashCode()

测试返回的哈希值与期望是否相同

测试equals()

两位置相同，两位置不同



## 面向复用的设计：Timeslot

一个immutable的类型，表示一个时间段

1. 数据域：分别表示开始结束时间的start和end



1. 方法：

|  |  |
| --- | --- |
| public Timeslot(String start, String end) | 构造方法，使用两个格式为“yyyy-MM-dd HH:mm”的字符串转化为LocalDateTime再创建时间段 |
| public Timeslot (LocalDateTime start, LocalDateTime end) | 构造方法，使用两个LocalDateTime类型的参数创建时间段 |
| public LocalDateTime getStartTime() | 获得开始时间，用start生成一个副本然后把副本返回 |
| public LocalDateTime getEndTime() | 获得结束时间，用end生成一个副本然后把副本返回 |
| public String getStartTimeString() | 获得表示开始时间的字符串，格式为yyyy-MM-ddHH:mm |
| public String getEndTimeString() | 获得表示结束时间的字符串，格式为yyyy-MM-ddHH:mm |
| public boolean isBefore(Timeslot time) | 判断this.end不晚于time.start，则返回true，否则返回false |
| public boolean isAfter(Timeslot time) | 判断this.start不早于time.end，则返回true，否则返回false |
| public boolean equals(Object time) | 判断两个对象保存的是否为同一个时间段，是则返回true，否则为false |

1. A F、RI、Safety from rep exposure：

Abstraction function:

AF(start,end)=以start为开始时间，end为结束时间的一段时间

Representation invariant:

start!=null

end!=null

Safety from rep exposure:

所有的数据域都是私有的且使用final限定

使用防御式拷贝返回start和end两个时间

1. 测试：

Test strategy

测试getStartTime()和getEndTime()

测试得到的时间是否与期望的时间相同

测试getStartTimeString()和getEndTimeString()

测试得到的时间字符串是否与期望的时间相同

测试isBefore()

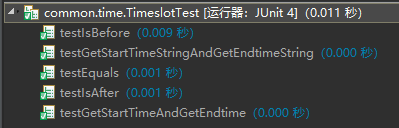
两个时间段相交，两个时间段不相交

测试isAfter()

两个时间段相交，两个时间段不相交

测试equals()

两个时间段的表示的时间相等，表示的时间不等



## 面向复用的设计：EntryState及State设计模式

根据状态转移图设计状态转移函数，在接口的子类型中各自实现向下一个状态的转换。

### 状态接口State

表示状态的接口。

方法：

（1）public String state()：获得一个表示状态的字符串。

（2）public State nextState(String nextState)：状态转移函数，根据参数nextState转移到下一个状态，如果参数符合状态转移图，则返回新的状态，如果不符合，则返回当前的状态。

（3）public boolean isFinalState()：判断当前状态是否为终态，如果是终态返回true，不是则返回false。

### 未分配状态Waiting

|  |  |
| --- | --- |
| 方法 | 实现 |
| public String state() | 直接返回WAITING |
| public State nextState(String nextState) | 判断参数是CANCELLED，返回已取消状态，判断参数是ALLOCATED，返回已分配状态，否则说明参数不在状态转移图中，返回未分配状态 |
| public boolean isFinalState() | 不是终态，返回false |

### 已分配状态Allocated

|  |  |
| --- | --- |
| 方法 | 实现 |
| public String state() | 直接返回ALLOCATED |
| public State nextState(String nextState) | 判断参数是RUNNING，返回已启动状态，判断参数是CANCELLED，返回已取消状态，否则说明参数不在状态转移图中，返回已分配状态 |
| public boolean isFinalState() | 不是终态，返回false |

### 已启动状态Running

|  |  |
| --- | --- |
| 方法 | 实现 |
| public String state() | 直接返回RUNNING |
| public State nextState(String nextState) | 判断参数是ENDED，返回已完成状态，判断参数是BLOCKED，返回已挂起状态，否则说明参数不在状态转移图中，返回未分配状态 |
| public boolean isFinalState() | 不是终态，返回false |

### 挂起中状态Blocked

|  |  |
| --- | --- |
| 方法 | 实现 |
| public String state() | 直接返回BLOCKED |
| public State nextState(String nextState) | 判断参数是RUNNING，返回已启动状态，判断参数是CANCELLED，返回已取消状态，否则说明参数不在状态转移图中，返回未分配状态 |
| public boolean isFinalState() | 不是终态，返回false |

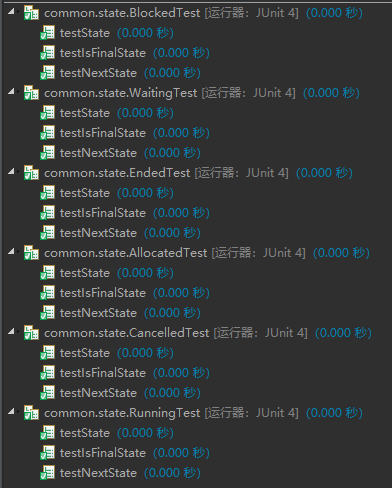
### 已完成状态Ended

|  |  |
| --- | --- |
| 方法 | 实现 |
| public String state() | 直接返回ENDED |
| public State nextState(String nextState) | 已经是最终状态，所以不论传入什么参数，都返回已完成状态 |
| public boolean isFinalState() | 是终态，返回true |

### 已取消状态Cancelled

|  |  |
| --- | --- |
| 方法 | 实现 |
| public String state() | 直接返回CANCELLED |
| public State nextState(String nextState) | 已经是最终状态，所以不论传入什么参数，都返回已取消状态 |
| public boolean isFinalState() | 是终态，返回true |

测试：



## 面向应用的设计：Board

Immutable的类，用于显示某时刻在某位置的所有计划项

### FlightEntry的Board

1. 数据域：



1. 方法：

（1）public Iterator<PlanningEntry<R>> iterator()：迭代器，委托给flightEntries，调用flightEntries.iterator()。

（2）public boolean addFlightEntry(PlanningEntry<R> flightEntry)：在计划清单中增加计划项，委托给flightEntries，调用flightEntries.addEntries(flightEntry)。

（3）public List<PlanningEntry<R>> getFlightEntries()：获得所有的计划项，委托给flightEntries，调用flightEntries.getEntries()。

（4）public void allEntryAtLocation(Location loc, String time)：显示loc位置，time时刻前后一小时内的所有计划项，先确定时间范围为time±1小时，再把所有的计划项按开始时间从早到晚排序，然后遍历所有的计划项，找到所有位置在loc的计划项，然后判断这些计划项的开始结束时间是否在范围内，把满足条件的计划项的信息加入到待显示的信息的集合中。最后调用visualize()把信息显示出来。

（5）public void visualize(Object[][] content, String[] title, String windowName)：可视化功能，见3.8外部API复用。

1. A F、RI、Safety from rep exposure：

Abstraction function:

AF(flightEntries)=现实中的航班计划项清单

Representation invariant:

flightEntries!=null

Safety from rep exposure:

所有的数据域都是私有的用final限定

### TrainEntry的Board

1. 数据域：



1. 方法：

（1）public Iterator<PlanningEntry<R>> iterator()：迭代器，委托给trainEntries，调用trainEntries.iterator()。

（2）public boolean addTrainEntry (PlanningEntry<R> trainEntry)：在计划清单中增加计划项，委托给trainEntries，调用trainEntries.addEntries(trainEntry)。

（3）public List<PlanningEntry<R>> getTrainEntries()：获得所有的计划项，委托给trainEntries，调用trainEntries.getEntries()。

（4）public void allEntryAtLocation(Location loc, String time)：显示loc位置，time时刻前后一小时内的所有计划项，先确定时间范围为time±1小时，再把所有的计划项按开始时间从早到晚排序，然后遍历所有的计划项，找到所有位置在loc的计划项，然后判断这些计划项的开始结束时间是否在范围内，把满足条件的计划项的信息加入到待显示的信息的集合中。最后调用visualize()把信息显示出来。

（5）public void visualize(Object[][] content, String[] title, String windowName)：可视化功能，见3.8外部API复用。

1. A F、RI、Safety from rep exposure：

Abstraction function:

AF(trainEntries)=现实中的高铁车次计划项清单

Representation invariant:

trainEntries!=null

Safety from rep exposure:

所有的数据域都是私有的用final限定

### ActivityEntry的Board

1. 数据域：



1. 方法：

（1）public Iterator<PlanningEntry<R>> iterator()：迭代器，委托给activityEntries，调用activityEntries.iterator()。

（2）public boolean addFlightEntry(PlanningEntry<R> activityEntry)：在计划清单中增加计划项，委托给activityEntries，调用activityEntries.addEntries(activityEntry)。

（3）public List<PlanningEntry<R>> get activityEntries()：获得所有的计划项，委托给activityEntries，调用activityEntries.getEntries()。

（4）public void allEntryAtLocation(Location loc, String day)：显示loc位置，day这一天所有的计划项，先把所有的计划项按开始时间从早到晚排序，然后遍历所有的计划项，找到所有位置在loc的计划项，然后判断这些计划项的开始结束时间是否在这一天，把满足条件的计划项的信息加入到待显示的信息的集合中。最后调用visualize()把信息显示出来。

（5）public void visualize(Object[][] content, String[] title, String windowName)：可视化功能，见3.8外部API复用。

1. A F、RI、Safety from rep exposure：

Abstraction function:

AF(activityEntries)=现实中的活动计划项清单

Representation invariant:

activityEntries!=null

Safety from rep exposure:

所有的数据域都是私有的用final限定

## Board的可视化：外部API的复用

三个计划项的board可视化功能都是相同的实现，利用JTable实现了表格的可视化功能。

public void visualize(Object[][] content, String[] title, String windowName)：以title中的内容为表格的标题，将content中的内容作为表格内容构造JTable，然后设置窗口大小为500\*500，添加一个动态显示的滚动条，设置窗口的标题为windowName，然后把窗口显示，最后添加监听器，当点击关闭时将窗口隐藏。

## PlanningEntryCollection的设计

1. 数据域：保存所有计划项的列表



1. 方法：

|  |  |
| --- | --- |
| 方法 | 实现 |
| public boolean addEntries (PlanningEntry<R> entry) | 判断entry如为null，则返回false，否则将entry加入到entries中并返回true |
| public List<PlanningEntry<R>> getEntries() | 把entries使用Collections.unmodifiableList() 封装后再返回 |
| public Iterator<PlanningEntry<R>> iterator() | 直接返回entries.iterator() |
| public void sort() | 直接调用entries.sort()为计划项排序 |

1. A F、RI、Safety from rep exposure：

Abstraction function:

AF(entries)=现实中的计划项清单

Representation invariant:

entries!=null

Safety from rep exposure:

所有的数据域都是私有的用final限定

获得entries时使用Collections.unmodifiableList()转化为不可更改的list返回

1. 测试：

Test strategy

测试AddEntries()

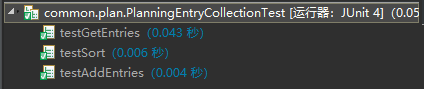
添加的计划项未null，计划项不为null

测试getEntries()

当没有计划项时，当有计划项时

测试sort()

测试能够按照时间的先后顺序给计划项排序



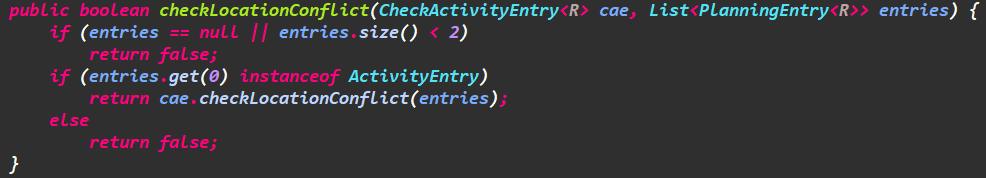
## 可复用API设计及Façade设计模式

按照facade设计模式，创建一个外观类PlanningEntryAPIs，通过委托给其他类实现相应的功能。

### 检测一组计划项之间是否存在位置独占冲突

所选定的计划项只有ActivityEntry可能存在位置独占冲突，创建检查其中的位置冲突的接口CheckActivityEntry，在其中定义方法public boolean checkLocationConflict(List<PlanningEntry<R>> entries)，在其实现类CheckActivityEntryPlanA中实现该方法，通过遍历entries中的所有计划项的位置和使用时间判断是否有冲突。

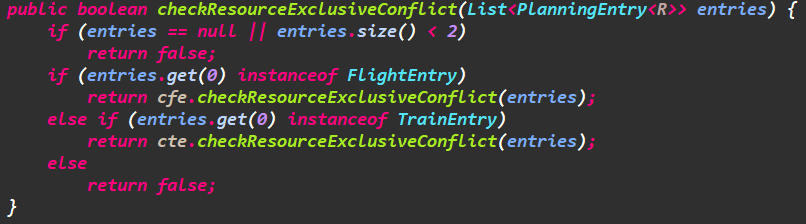
在PlanningEntryAPIs中定义用于检查计划项列表中的位置冲突的方法public boolean checkLocationConflict(CheckActivityEntry<R> cae, List<PlanningEntry<R>> entries)，先判断传入的entries中的元素的实际类型为ActivityEntry，则调用cae.checkLocationConflict(entries)委托给cae去做，否则，直接返回false表示不存在冲突。



### 检测一组计划项之间是否存在资源独占冲突

所选定的计划项中FlightEntry和TrainEntry可能存在资源独占冲突，创建检查其中的资源冲突的类CheckFlightEntry 和CheckTrainEntry，在这两个类中分别实现检查资源独占冲突的方法public boolean checkResourceExclusiveConflict(List<PlanningEntry<R>> entries)，通过遍历entries中的所有计划项的资源和使用时间判断是否有冲突。

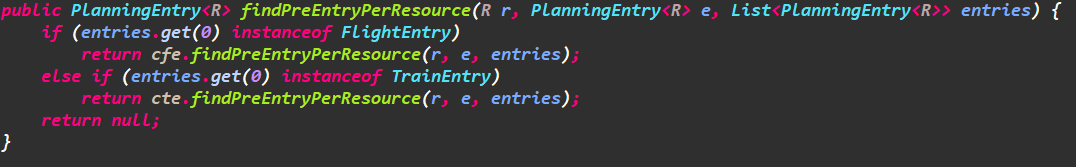
在PlanningEntryAPIs中定义检查计划项列表中资源冲突的方法public boolean checkResourceExclusiveConflict(List<PlanningEntry<R>> entries)，先判断entries中元素的实际类型是否为FlightEntry，如果是则调用cfe.checkResourceExclusiveConflict(entries)委托给cfe去做，否则再判断元素的实际类型是否为TrainEntry，如果是则调用cte.checkResourceExclusiveConflict(entries)委托给cte，如果不是这两种类型，则直接返回false表示不存在冲突。



### 提取面向特定资源的前序计划项

所选定的计划项中FlightEntry和TrainEntry可能存在使用特定资源的前序计划项，在类CheckFlightEntry 和CheckTrainEntry中分别实现查找使用特定资源的前序计划项的方法public PlanningEntry<R> findPreEntryPerResource(R r, PlanningEntry<R> e, List<PlanningEntry<R>> entries)，通过遍历entries中的所有计划项使用的资源和使用时间确定它是否为e的前序计划项，如果没有则返回null。

在PlanningEntryAPIs中实现查找使用特定资源的前序计划项方法public PlanningEntry<R> findPreEntryPerResource(R r, PlanningEntry<R> e, List<PlanningEntry<R>> entries)，先判断entries中元素的实际类型是否为FlightEntry，如果是则调用cfe.findPreEntryPerResource(r, e, entries)委托给cfe去做，否则再判断元素的实际类型是否为TrainEntry，如果是则调用cte.findPreEntryPerResource(r, e, entries)委托给cte，如果不是这两种类型，则直接返回null表示不存在。



测试：

Test strategy

测试checkLocationConflict()

计划项列表中没有计划项，计划项列表只有一个计划项，计划项列表中至少有两个计划项

计划项中不存在冲突，计划项中存在冲突

测试checkResourceExclusiveConflict()

按计划项类型分：计划项为航班计划项，计划项为高铁车次计划项

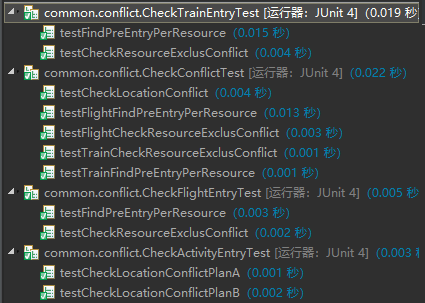
按计划项数量分：计划项列表中没有计划项，计划项列表只有一个计划项，计划项列表中至少有两个计划项

按计划项是否有冲突分：计划项中不存在冲突，计划项中存在冲突

测试findPreEntryPerResource()

按计划项类型分：计划项为航班计划项，计划项为高铁车次计划项

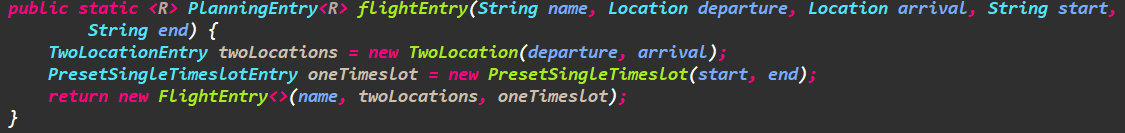
按是否存在满足条件的计划项分：存在满足条件的计划项，不存在满足条件的计划项

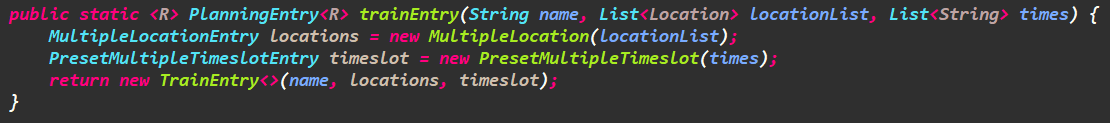


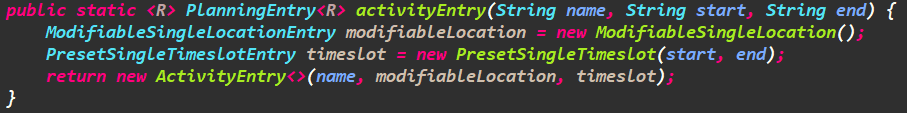
## 设计模式应用

### Factory Method

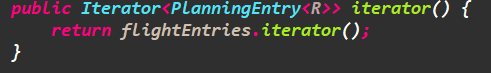
在PlanningEntry中，定义了三个静态工厂方法，用来创建具体的计划项实例。实现如下：

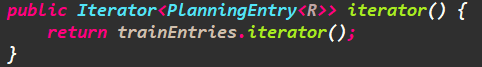


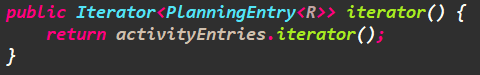




### Iterator

在Board中实现public Iterator<PlanningEntry<R>> iterator()，将实现委托给PlanningEntryCollection去做，在各PlanningEntry子类型对应的board中分别进行委托，如下：  


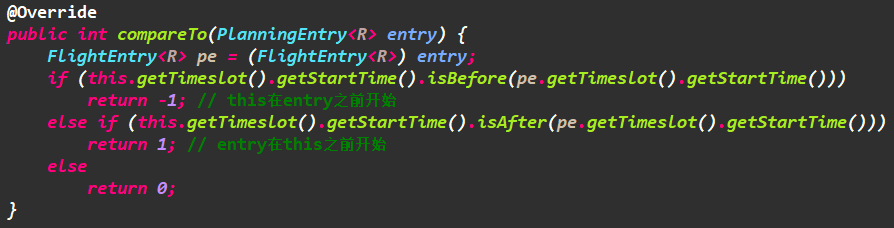


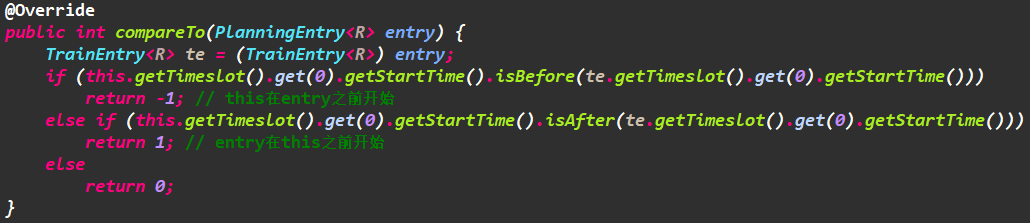


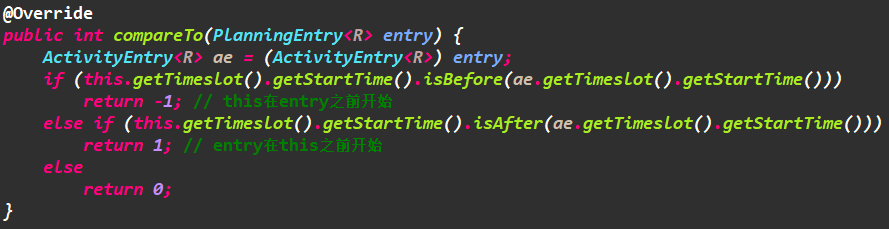
为了方便支持按时间对计划项排序，在PlanningEntry 及其子类型中增加比较功能，在PlanningEnry接口中扩展Comparable接口：



并在具体的子类中实现public int compareTo(PlanningEntry<R> entry)。如下：





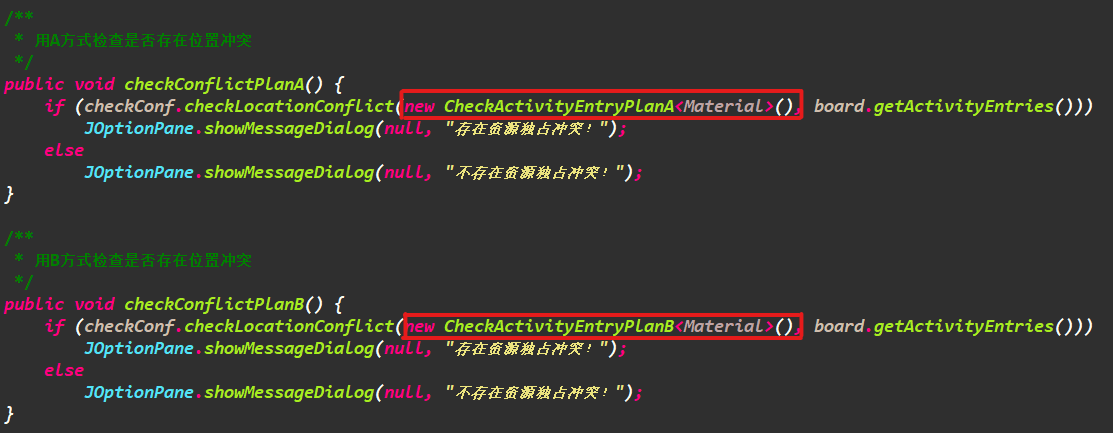


### Strategy

public boolean checkLocationConflict(List<PlanningEntry<R>> entries)在策略模式下灵活替换不同实现。

实现一个新的类CheckActivityEntryPlanB，在这个类中用另一种算法实现接口CheckActivityEntry中的上述方法。然后在PlanningEntryAPIs中的checkLocationConflict()的参数CheckActivityEntry<R> cae中传入不同的实例来实现灵活的替换不同的算法。

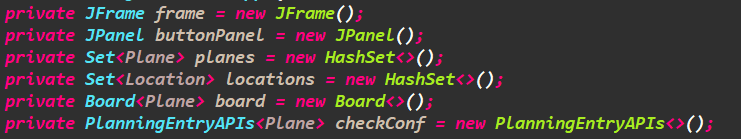
客户端实现：



## 应用设计与开发

### 航班应用

1. 数据域：实现GUI面板用到的frame和buttonPanel，保存所有可用资源的planes，保存所有可用位置的locations，计划项信息板borad，检查冲突和使用资源的计划项的checkConf。



1. 功能设计：

（1）用户提供必要信息，管理（增加、删除）可用的资源

public void addPlane()：增加，通过对话框让用户输入可用飞机资源的各个属性，然后判断有这些属性的飞机是否在资源集合planes中，如果在，输出提示信息“此飞机已存在！”，否则将此飞机资源添加到planes中并输出提示信息“添加成功！”。

public void removePlane()：删除，通过对话框让用户输入可用飞机资源的各个属性，然后判断有这些属性的飞机是否在资源集合planes中，如果在，将此飞机资源从planes中删除并输出提示信息“删除成功！”，否则输出提示信息“此飞机不存在！”。

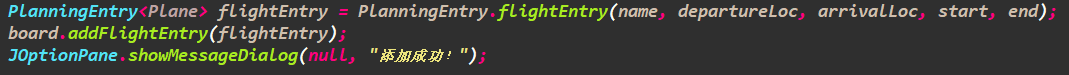
（2）用户提供必要信息，管理（增加、删除）可用的位置

public void addLocation()：增加，通过对话框让用户输入位置的名称，然后判断这个位置是否已经在位置集合locations中，如果是，输出提示信息“此机场已存在！”，否则把这个位置加入到locations中输出提示信息“添加成功！”。

public void removeLocation()：删除，通过对话框让用户输入位置的名称，然后判断这个位置是否已经在位置集合locations中，如果是，将此位置从locations中删除并输出提示信息“删除成功！”，否则输出提示信息“此机场不存在！”。

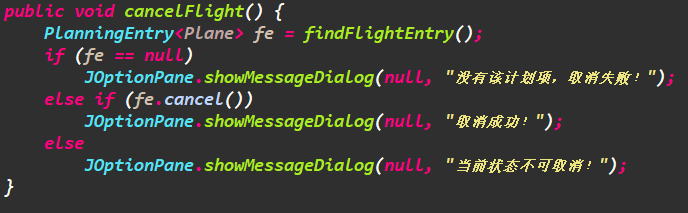
（3）用户提供必要信息，增加一条新的计划项

public void addFlight()：通过对话框让用户输入计划项的名称、位置、时间信息，然后根据输入的这些信息判断是否已有计划项包含这些信息，如果是，直接输出提示“已有该计划项，添加失败！”结束，否则判断输入的位置是否在可用位置集合中，如果不在，输出提示“无位置，添加失败！”结束，否则说明条件满足，新建一条计划项将其加入到计划项清单中，输出提示“添加成功！”结束。



（4）用户提供必要信息，取消某个计划项

public void cancelFlight()：调用findFlightEntry()方法，根据用户输入的数据找到指定的计划项，然后判断返回值是否为null，如果是，说明计划项不存在，提示“没有该计划项，取消失败！”并结束；如果不是就调用PlanningEntry.cancel()方法，判断返回值，如果为true，说明设置成功，提示“取消成功！”结束，如果为false说明设置失败，提示“当前状态不可取消！”结束。



（5）用户提供必要信息，为某个计划项分配资源

public void allocateFlight()：调用findFlightEntry()方法，根据用户输入的数据找到指定的计划项，再让用户输入资源的属性，然后判断指定的计划项是否存在，判断指定的资源是否存在，如不存在，输出对应的提示并结束，如存在，调用FlightEntry.allocatePlane(plane)分配资源，然后根据返回值确定是否分配成功。

（6）用户提供必要信息，启动某个计划项

public void runFlight()：实现逻辑与cancelFlight()相同，调用的计划项的方法变更为PlanningEntry.running()。

（7）用户提供必要信息，结束某个计划项

public void endFlight()：实现逻辑与cancelFlight()相同，调用的计划项的方法变更为PlanningEntry.end()。

（8）用户选定一个计划项，查看它的当前状态

public void getState()：调用findFlightEntry()方法，根据用户输入的数据找到指定的计划项，如果没找到输出提示，如果找到输出该计划项的状态。

（9）检测当前的计划项集合中可能存在的资源独占冲突

public void checkConflict()：通过委托给checkConf，调用方法checkConf.checkResourceExclusiveConflict()把所有计划项board.getFlightEntries()作为参数传入，判断返回值，如果为true，则说明存在资源独占冲突，如果为false，说明不存在资源独占冲突。

（10）针对用户选定的某个资源，列出使用该资源的所有计划项

public void findAllEntryUseResource()：根据用户输入的资源属性，先判断可用资源集合中包含该资源，然后遍历所有的计划项，找出使用该资源的所有计划项，然后把这些计划项的信息通过调用board.visualize()显示。

（11）用户选中其中某个计划项之后，可以找出它的前序计划项

public void findPreEntryUseResource()：根据用户输入的信息找到指定的资源和计划项，如没找到输出提示信息然后返回，如找到委托给checkConf，调用方法checkConf.findPreEntryPerResource()找到前序计划项然后输出信息。

（12）选定特定位置，可视化展示当前时刻该位置的信息板

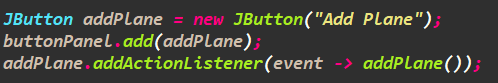
public void findLocation()：通过用户输入的信息判断是否存在指定的位置，如果不存在直接返回，如果存在则委托给board，通过调用方法board.allEntryAtLocation()找到并显示指定位置在指定时间前后一小时内的所有计划项。

（13）辅助方法：通过用户输入的信息找到指定的计划项

private PlanningEntry<Plane> findFlightEntry()：根据用户输入的信息，遍历所有的计划项，找到信息相同的计划项然后返回它，如果没有找到则返回null。

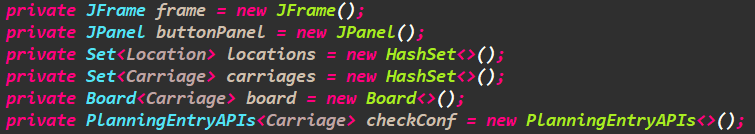
（14）构造方法

在面板上用如下的方式添加相应功能的按钮。在最后添加监听器，当点击关闭时结束程序。



### 高铁应用

1. 数据域：实现GUI面板用到的frame和buttonPanel，保存所有可用资源的carriages，保存所有可用位置的locations，计划项信息板borad，检查冲突和使用资源的计划项的checkConf。



1. 功能设计：

（1）用户提供必要信息，管理（增加、删除）可用的资源

public void addCarriage ()：增加，通过对话框让用户输入可用车厢资源的各个属性，然后判断有这些属性的车厢是否在资源集合carriages中，如果在，输出提示信息“此车厢已存在！”，否则将此车厢资源添加到carriages中并输出提示信息“添加成功！”。

public void removeCarriage ()：删除，通过对话框让用户输入可用车厢资源的各个属性，然后判断有这些属性的车厢是否在资源集合carriages中，如果在，将此车厢资源从carriages中删除并输出提示信息“删除成功！”，否则输出提示信息“此车厢不存在！”。

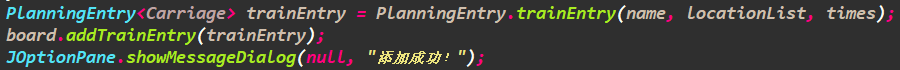
（2）用户提供必要信息，管理（增加、删除）可用的位置

public void addLocation()：增加，通过对话框让用户输入位置的名称，然后判断这个位置是否已经在位置集合locations中，如果是，输出提示信息结束，否则把这个位置加入到locations中输出提示信息结束。

public void removeLocation()：删除，通过对话框让用户输入位置的名称，然后判断这个位置是否已经在位置集合locations中，如果是，将此位置从locations中删除并输出提示信息结束，否则输出提示信息结束。

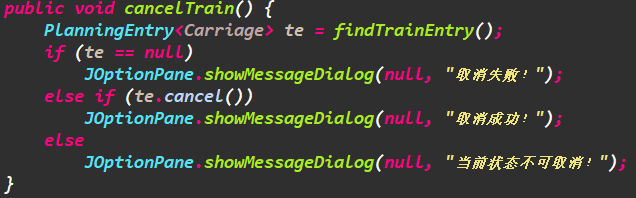
（3）用户提供必要信息，增加一条新的计划项

public void addTrain ()：通过对话框让用户输入计划项的名称、位置、时间信息，然后根据输入的这些信息判断是否已有计划项包含这些信息，如果是，直接输出提示“已有该计划项，添加失败！”结束，否则判断输入的位置是否在可用位置集合中，如果不在，输出提示“无位置，添加失败！”结束，否则说明条件满足，新建一条计划项将其加入到计划项清单中，输出提示“添加成功！”结束。



（4）用户提供必要信息，取消某个计划项

public void cancelTrain ()：调用findTrainEntry ()方法，根据用户输入的数据找到指定的计划项，然后判断返回值是否为null，如果是，说明计划项不存在，提示“没有该计划项，取消失败！”并结束；如果不是就调用PlanningEntry.cancel()方法，判断返回值，如果为true，说明设置成功，提示“取消成功！”结束，如果为false说明设置失败，提示“当前状态不可取消！”结束。



（5）用户提供必要信息，为某个计划项分配资源

public void allocateTrain ()：调用findTrainEntry ()方法，根据用户输入的数据找到指定的计划项，再让用户输入资源的属性，然后判断指定的计划项是否存在，判断指定的资源是否存在，如不存在，输出对应的提示并结束，如存在，调用TrainEntry.allocateTrain (train)分配资源，然后根据返回值确定是否分配成功。

（6）用户提供必要信息，启动（重启动）某个计划项

public void runTrain ()：实现逻辑与cancelTrain()相同，调用的计划项的方法变更为PlanningEntry.running()。

（7）用户提供必要信息，以阻塞/挂起某个计划项

public void blockTrain()：实现逻辑cancelTrain()相同，调用的计划项的方法变更为TrainEntry.block ()。

（8）用户提供必要信息，结束某个计划项

public void endTrain ()：实现逻辑与cancelTrain ()相同，调用的计划项的方法变更为PlanningEntry.end()。

（9）用户选定一个计划项，查看它的当前状态

public void getState()：调用findTrainEntry ()方法，根据用户输入的数据找到指定的计划项，如果没找到输出提示，如果找到输出该计划项的状态。

（10）检测当前的计划项集合中可能存在的资源独占冲突

public void checkConflict()：通过委托给checkConf，调用方法checkConf.checkResourceExclusiveConflict()把所有计划项board. getTrainEntries ()作为参数传入，判断返回值，如果为true，则说明存在资源独占冲突，如果为false，说明不存在资源独占冲突。

（11）针对用户选定的某个资源，列出使用该资源的所有计划项

public void findAllEntryUseResource()：根据用户输入的资源属性，先判断可用资源集合中包含该资源，然后遍历所有的计划项，找出使用该资源的所有计划项，然后把这些计划项的信息通过调用board.visualize()显示。

（12）用户选中其中某个计划项之后，可以找出它的前序计划项

public void findPreEntryUseResource()：根据用户输入的信息找到指定的资源和计划项，如没找到输出提示信息然后返回，如找到委托给checkConf，调用方法checkConf.findPreEntryPerResource()找到前序计划项然后输出信息。

（13）选定特定位置，可视化展示当前时刻该位置的信息板

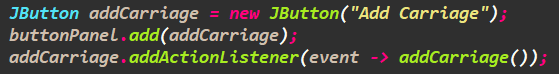
public void findLocation()：通过用户输入的信息判断是否存在指定的位置，如果不存在直接返回，如果存在则委托给board，通过调用方法board.allEntryAtLocation()找到并显示指定位置在指定时间前后一小时内的所有计划项。

（14）辅助方法：通过用户输入的信息找到指定的计划项

private PlanningEntry<Carriage> findTrainEntry()：根据用户输入的信息，遍历所有的计划项，找到信息相同的计划项然后返回它，如果没有找到则返回null。

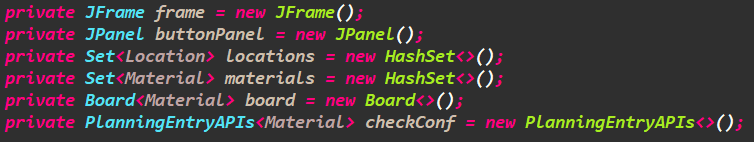
（15）构造方法

在面板上用如下的方式添加相应功能的按钮。在最后添加监听器，当点击关闭时结束程序。



### 学习活动应用

1. 数据域：实现GUI面板用到的frame和buttonPanel，保存所有可用资源的materials，保存所有可用位置的locations，计划项信息板borad，检查冲突和使用资源的计划项的checkConf。



1. 功能设计：

（1）用户提供必要信息，管理（增加、删除）可用的资源

public void addMaterrial ()：增加，通过对话框让用户输入可用材料资源的各个属性，然后判断有这些属性的材料是否在资源集合materials中，如果在，输出提示信息“此材料已存在！”，否则将此材料资源添加到materials中并输出提示信息“添加成功！”。

public void removeMaterial ()：删除，通过对话框让用户输入可用材料资源的各个属性，然后判断有这些属性的材料是否在资源集合materials中，如果在，将此材料资源从materials中删除并输出提示信息“删除成功！”，否则输出提示信息“此材料不存在！”。

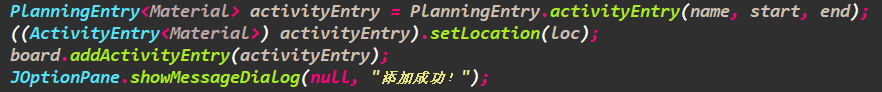
（2）用户提供必要信息，管理（增加、删除）可用的位置

public void addLocation()：增加，通过对话框让用户输入位置的名称，然后判断这个位置是否已经在位置集合locations中，如果是，输出提示信息“此地点已存在！”，否则把这个位置加入到locations中输出提示信息“添加成功！”。

public void removeLocation()：删除，通过对话框让用户输入位置的名称，然后判断这个位置是否已经在位置集合locations中，如果是，将此位置从locations中删除并输出提示信息“删除成功！”，否则输出提示信息“此地点不存在！”。

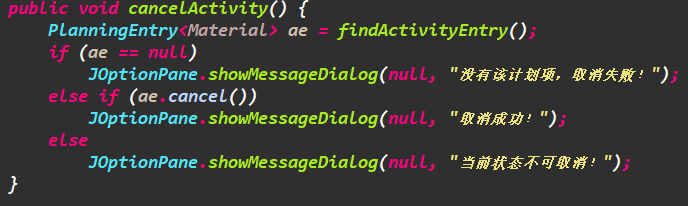
（3）用户提供必要信息，增加一条新的计划项

public void addActivity ()：通过对话框让用户输入计划项的名称、位置、时间信息，然后根据输入的这些信息判断是否已有计划项包含这些信息，如果是，直接输出提示“已有该计划项，添加失败！”结束，否则判断输入的位置是否在可用位置集合中，如果不在，输出提示“无位置，添加失败！”结束，否则说明条件满足，新建一条计划项将其加入到计划项清单中，输出提示“添加成功！”结束。



（4）用户提供必要信息，取消某个计划项

public void cancelActivity ()：调用findActivityEntry ()方法，根据用户输入的数据找到指定的计划项，然后判断返回值是否为null，如果是，说明计划项不存在，提示“没有该计划项，取消失败！”并结束；如果不是就调用PlanningEntry.cancel()方法，判断返回值，如果为true，说明设置成功，提示“取消成功！”结束，如果为false说明设置失败，提示“当前状态不可取消！”结束。



（5）用户提供必要信息，为某个计划项分配资源

public void allocateFlight()：调用findActivityEntry ()方法，根据用户输入的信息找到指定的计划项，再让用户输入资源的属性，然后判断指定的计划项是否存在，判断指定的资源是否存在，如不存在，输出对应的提示并结束，如存在，调用计划项的ActivityEntry.allocateMaterial(material)分配资源，然后根据返回值确定是否分配成功。

（6）用户提供必要信息，启动某个计划项

public void runActivity ()：实现逻辑与cancelActivity()相同，调用的计划项的方法变更为PlanningEntry.running()。

（7）用户提供必要信息，以变更某个已存在的计划项的位置

public void setLocation()：调用findActivityEntry()方法，根据用户输入的信息找到指定的计划项找到指定的计划项，再让用户输入新的地点，然后判断新的地点和指定的计划项是否存在，如不存在，输出提示信息并结束，如果存在，则调用ActivityEntry.setLocation()设定位置，根据返回值判断是否设置成功，然后输出相应的提示信息并结束。

（8）用户提供必要信息，结束某个计划项

public void endActivity ()：实现逻辑与cancelActivity ()相同，调用的计划项的方法变更为PlanningEntry.end()。

（9）用户选定一个计划项，查看它的当前状态

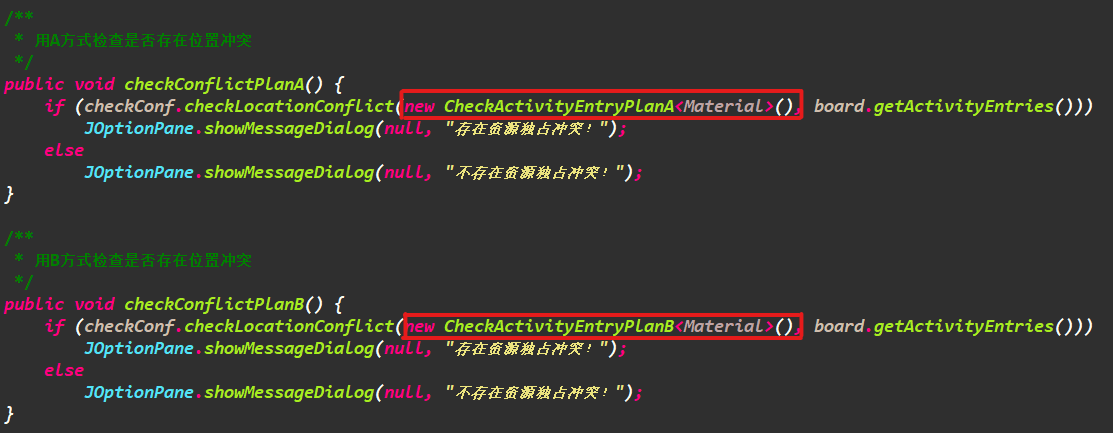
public void getState()：调用findActivityEntry ()方法，根据用户输入的数据找到指定的计划项，如果没找到输出提示，如果找到输出该计划项的状态。

（10）检测当前的计划项集合中可能存在的位置独占冲突

public void checkConflictPlanA():

public void checkConflictPlanB():

策略模式下两个不同的判断方式，但这里的实现除了传入的策略不同外全部相同，通过委托给checkConf，调用方法checkConf.checkLocationConflict()，判断返回值，如果为true，则说明存在位置独占冲突，如果为false，说明不存在位置独占冲突。



（11）针对用户选定的某个资源，列出使用该资源的所有计划项

public void findAllEntryUseResource()：根据用户输入的资源属性，先判断可用资源集合中包含该资源，然后遍历所有的计划项，找出使用该资源的所有计划项，然后把这些计划项的信息通过调用board.visualize()显示。

（12）选定特定位置，可视化展示当前时刻该位置的信息板

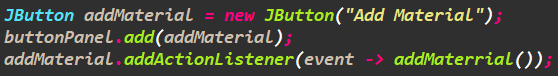
public void findLocation()：通过用户输入的信息判断是否存在指定的位置，如果不存在直接返回，如果存在则委托给board，通过调用方法board.allEntryAtLocation()找到并显示指定位置在指定时间前后一小时内的所有计划项。

（13）辅助方法：通过用户输入的信息找到指定的计划项

private PlanningEntry<Material> findActivityEntry()：根据用户输入的信息，遍历所有的计划项，找到信息相同的计划项然后返回它，如果没有找到则返回null。

（14）构造方法

在面板上用如下的方式添加相应功能的按钮。在最后添加监听器，当点击关闭时结束程序。



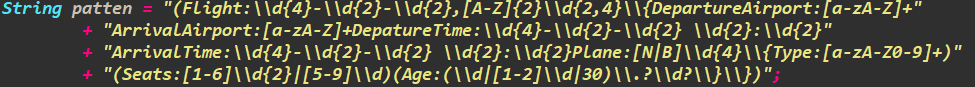
## 基于语法的数据读入

在航班应用中增加方法public void readFileCreatFlightSchedule()，通过用户输入的路径打开对应的文件，从文件中读取内容。根据语法，当读入两个“}”时确定一个计划项的内容读入完成。

使用正则表达式匹配读入的信息。

|  |  |  |
| --- | --- | --- |
| 内容 | 说明 | 对应的正则表达式 |
| Flight: | 固定的字符串 | Flight: |
| 2020-01-01 | 格式为yyyy-MM-dd的日期 | \d{4}-\d{2}-\d{2} |
| , | 固定的逗号字符 | , |
| CA1001 | 航班号，由两位大写字母和 2-4 位数字构成 | [A-Z]{2}\d{2,4} |
| { DepartureAirport: | 固定的字符串 | \{ DepartureAirport: |
| Harbin | 一个单词 | [a-zA-Z]+ |
| ArrivalAirport: | 固定的字符串 | ArrivalAirport: |
| Beijing | 一个单词 | [a-zA-Z]+ |
| DepatureTime: | 固定的字符串 | DepatureTime: |
| 2020-01-01 12:00 | 格式为yyyy-MM-dd HH:mm 的时间 | \d{4}-\d{2}-\d{2} \d{2}:\d{2} |
| ArrivalTime: | 固定的字符串 | ArrivalTime: |
| 2020-01-01 14:00 | 格式为yyyy-MM-dd HH:mm 的时间 | \d{4}-\d{2}-\d{2} \d{2}:\d{2} |
| Plane: | 固定的字符串 | Plane: |
| B9802 | 飞机编号，第一位为 N 或 B，后面是四位数字 | [N|B]\d{4} |
| { Type: | 固定的字符串 | \{ Type: |
| A350 | 机型，大小写字母或数字构成，不含有空格和其他符号 | [a-zA-Z0-9]+ |
| Seats: | 固定的字符串 | Seats: |
| 300 | 正整数，范围为[50,600] | [1-6]\d{2}|[5-9]\d |
| Age: | 固定的字符串 | Age: |
| 2.5 | 数字，范围是[0,30]，可最多 1 位小数或无小数 | \d|[1-2]\d|30)\.?\d? |
| }} | 固定的字符串 | \}\} |

所以一个计划项的完整的正则表达式如下：



读入的内容与正则表达式匹配之后根据固定字符串的位置分离出信息。

判断是否满足开始结束时间的要求：起飞时间中的日期必须与第一行的日期一致，抵达时间中的日期可以与起飞日期一致。

判断是否有日期一样且航班号一样的两个计划项。

判断同一个航班号，虽然日期不同，但其出发和到达机场、出发和到达时间是否都相同。

如果上述条件都满足，则先把不在集合中的资源和位置加入到集合中，然后根据读入的信息创建一个新的计划项，将其加入到计划项清单中，否则输出对应的提示信息结束读入。

## 应对面临的新变化

### 变化1

变化：航班支持经停，即包含最多 1 个中间经停机场，例如哈尔滨-威海-深圳。

评估：修改代价较其他两个大。

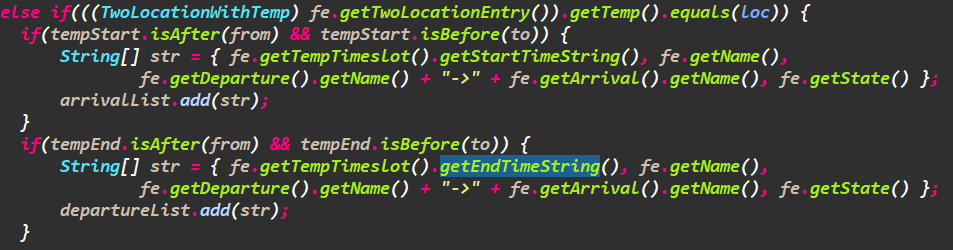
在FlightEntry的数据域中添加PresetSingleTimeslotEntry类的tempTimeslot表示中转的开始和结束时间，在构造方法中添加设定它。增加一个获得所有位置的方法public TwoLocationEntry getTwoLocationEntry()。

增加一个TwoLocationEntry的实现类TwoLocationWithTemp，相较TwoLocation类增加了表示经停位置的temp数据域，并增加相应的getter方法。

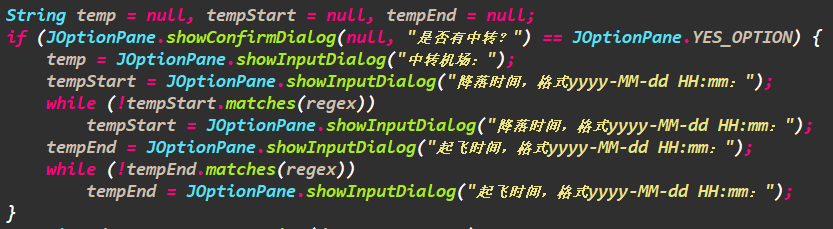
修改PlanningEntry中的静态工厂方法，增加经停机场位置时间参数以满足要求。把创建的TwoLocationEntry的实际类型由TwoLocation更改为TwoLocationWithTemp。



在Board. allEntryAtLocation()，显示某位置某时刻前后一小时内的所有计划项的方法中增加如下判断中转位置的部分。



在客户端findFlightEntry()方法和addFlight()方法中增加选择是否有中转的相关操作。

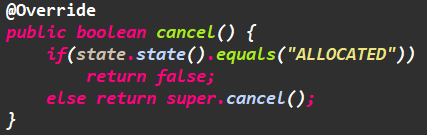


### 变化2

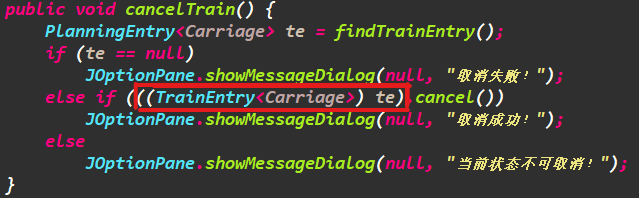
变化：如果高铁车次已经分配了车厢资源，则不能被取消。

评估：修改代价较小。

重写父类CommonPlanningEntry中的cancel()方法，当状态为已分配资源时直接返回false表示设置失败，否则调用父类的cacel()方法。



在客户端中把类型强转为子类型以调用子类型重写后的cancel()方法。

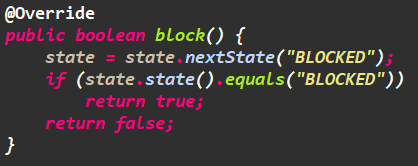


### 变化3

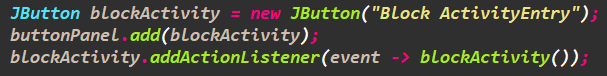
变化：活动开始执行之后，可以被阻塞（临时暂停），后续继续进行（位置、资源均不变）。

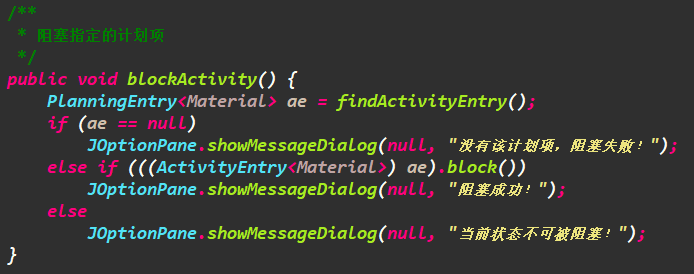
评估：修改代价较小。

在ActivityEntry添加BlockableEntry接口实现，实现接口中的block方法。



在应用中添加一个用于暂停的按钮，并增加对应的方法。





## Git仓库结构



# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2020.04.15 | 16:00-21:00 | 规划本次实验实现的大致结构 | 按时完成 |
| 2020.04.19 | 13:30-17:30 | 编写无界面应用FlightSchedule | 按时完成 |
| 2020.04.20 | 18:30-21:30 | 编写无界面应用FlightSchedule | 按时完成 |
| 2020.04.21 | 19:00-22:30 | 编写无界面应用FlightSchedule | 按时完成 |
| 2020.04.24 | 19:00-22:00 | 编写3.10的API | 未完成 |
| 2020.04.25 | 19:30-22:00 | 编写3.10 | 延时完成 |
| 2020.04.26 | 19:00-23:00 | 为activity包、conflict包、location包编写测试 | 按时完成 |
| 2020.04.27 | 19:00-23:00 | 为剩余的类编写测试 | 延时完成 |
| 2020.04.28 | 19:00-21:00 | 使用state模式实现状态转移 | 按时完成 |
| 2020.04.29 | 18:00-23:00 | 编写FlightScheduleApp的GUI界面 | 按时完成 |
| 2020.04.30 | 19:00-22:00 | 编写其余两个App的GUI界面 | 延时完成 |
| 2020.05.02 | 16:00-18:00 | 测试完善 | 按时完成 |
| 2020.05.03 | 14:00-18:00 | 编写3.14 | 按时完成 |
| 2020.05.05 | 19:00-23:00 | 撰写实验报告3.5之前并完善实现 | 按时完成 |
| 2020.05.06 | 15:30-18:30 | 撰写实验报告3.10之前并完善实现 | 按时完成 |
| 2020.05.07 | 19:00-23:00 | 撰写实验报告3.13之前并完善实现 | 延时完成 |
| 2020.05.09 | 9:00-11:00 | 撰写实验报告3.14并完善实现 | 按时完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 语法读入，正则表达式 | 在网上查阅资料学习正则表达式 |
| JTable的调用 | 查阅javadoc，了解JTable的用法 |
| GUI的编写 | 查阅资料，根据java核心技术的swing的相关章节学习 |

# 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？
2. 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？
3. 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？
4. 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？
5. 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？
6. 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？
7. “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？
8. 关于本实验的工作量、难度、deadline。
9. 到目前为止你对《软件构造》课程的评价。
10. 面向ADT编程可以复用。复用极大的减少了代码量。
11. 在撰写这些东西的时候可以提醒自己防止表示泄露，其他人使用自己写的ADT时可以放心使用。为了团队协作，我愿意坚持。
12. 使用别人的API感觉很方便，而写给别人使用的API要处处小心，要站在使用者的角度考虑开发问题。
13. 使用设计模式增加了程序的可维护性，方便未来对代码功能做修改和扩展，在做这些事情的时候不需要大规模的改动已有的代码，只需要更改相应的调用关系即可。
14. 方便数据的读入，可以适应不同的格式相同的输入。
15. 难度在于设计ADT，如何从不同的需求中抽取共性，如何从个性中抽取共性，如何满足对ADT的防止表示泄露的问题。这些问题只能是先做出一个初步的成果，然后根据要求去发现可以抽取成父类方法继承给子类的部分，然后做相应的修改。不可能一次性设计完美的。
16. 尽量使用接口，把所有能使用委托的功能都为委托到其他更具体的类实现，提高类内的聚合度。而使用接口也可以方便未来的功能修改或扩展。
17. 工作量很大，与Lab2相比大了不止一倍，难度偏难，deadline还好。
18. 很好，感觉能够充分锻炼自己的工程开发能力。