



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

目录

1 实验目标概述	1
2 实验环境配置	1
3 实验过程	1
3.1 待开发的三个应用场景	2
3.2 面向可复用性和可维护性的设计： <code>PlanningEntry<R></code>	3
3.2.1 <code>PlanningEntry<R></code> 的共性操作	3
3.2.2 局部共性特征的设计方案	4
3.2.3 面向各应用的 <code>PlanningEntry</code> 子类型设计（个性化特征的设计方案）	4
3.3 面向复用的设计： <code>R</code>	7
3.4 面向复用的设计： <code>Location</code>	10
3.5 面向复用的设计： <code>Timeslot</code>	11
3.6 面向复用的设计： <code>EntryState</code> 及 <code>State</code> 设计模式	12
3.7 面向应用的设计： <code>Board</code>	15
3.8 <code>Board</code> 的可视化：外部 API 的复用	19
3.9 可复用 API 设计及 Façade 设计模式	21
3.9.1 检测一组计划项之间是否存在位置独占冲突	21
3.9.2 检测一组计划项之间是否存在资源独占冲突	21
3.9.3 提取面向特定资源的前序计划项	22
3.10 设计模式应用	23
3.10.1 Factory Method	23
3.10.2 Iterator	24
3.10.3 Strategy	27
3.11 应用设计与开发	29
3.11.1 航班应用	29
3.11.2 高铁应用	34
3.11.3 课程应用	38
3.12 基于语法的数据读入	43
3.13 应对面临的新变化	49
3.13.1 变化 1	49

3.13.2 变化 2	51
3.13.3 变化 3	52
3.14 Git 仓库结构.....	53
4 实验进度记录.....	54
5 实验过程中遇到的困难与解决途径.....	55
6 实验过程中收获的经验、教训、感想	55
6.1 实验过程中收获的经验和教训.....	55
6.2 针对以下方面的感受	55

1 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）。

2 实验环境配置

这次实验需要的环境前两个实验已经配置完成。

在这里给出你的 GitHub Lab3 仓库的 URL 地址（Lab3-学号）。

<https://github.com/ComputerScienceHIT/Lab3-1180800811.git>

3 实验过程

请仔细对照实验手册，针对每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

3.1 待开发的三个应用场景

列出你所选定的三个应用。

分析三个应用场景的异同，理解需求：它们在哪些方面有共性、哪些方面有差异。

- 选用的三个应用：
- 1、航班管理
 - 2、高铁车次管理
 - 3、大学课表管理

三个应用的共性：①均表示一个待执行的活动/任务，它需要配置特定类型/数的资源，并在特定的物理位置加以执行。

②活动的状态有交集：都有未分配资源、已分配资源但未启动、已启动、已完成、已取消五个状态。

三个应用的差异：①**位置差异：**航班：两个位置，位置可共享
高铁车次：多个位置，位置可共享
大学课程：一个位置，位置不可共享

②**资源差异：**航班/课程：单个可区分的资源(飞机)
高铁：多个可区分的资源

③**状态差异：**高铁相较于航班/课程，多一个可阻塞的状态

④**位置是否可更改差异：**航班/高铁：可提前设定，设定后不可更改
课程：可提前设定，设定后可更改

⑤**时间差异：**高铁：多个起止时间对
课程/航班：1 个时间起止对

3.2 面向可复用性和可维护性的设计：**PlanningEntry<R>**

该节是本实验的核心部分。

3.2.1 PlanningEntry<R>的共性操作

① 和状态有关的方法

- 启动一个计划项
- 结束一个计划项
- 取消一个计划项
- 获取当前的状态
- 判断当前状态是否可接受
- 设定计划项的状态

② getter 方法

- 获取计划项使用的全部资源
- 获取计划项的起止时间
- 获取计划项的名字
- 得到计划项所用的位置和时间对的对应关系(主要用于 API 设计，判断位置冲突)

③ setter 方法

- 设定计划项的名字

说明：①在 PlanningEntry 中提供一个统一的获取资源的方法 `getresource`，这个方法可以用来 APIs 中判断资源冲突，返回值类型是一个 List，因为考虑到可能使用多个资源，也可能使用单个资源，所以一起放在一个 List 列表存储所有的资源，

②得到计划项所用的位置和时间对的对应关系是为了 3.10 API 的设计需要判断位置冲突。通过判断相同位置对应的时间是否重叠来判断是否有位置冲突。这个方法既可以得到所有的时间对，也可以得到所有的位置。

以上两个方法抽象出来放在公共接口 PlanningEntry 中，完全是为了设计 API。经验证，这样设计之后，314change 我不需要改动任何的 APIs 类的代码。可复用性很高。

3.2.2 局部共性特征的设计方案

创建一个新的子类型：利用工厂方法能够创建一个指定类型的子类

```
static<R> PlanningEntry<R> NewFlightEntry(String type, String name){  
    >> return new Factory().getPlanningEntry(type, name);  
}
```

```
>> public PlanningEntry getPlanningEntry(String type, String name){  
>>     if(type.equals("CourseEntry")) {  
>>         >>         return new CourseEntry(name);  
>>     }else if(type.equals("FlightEntry")) {  
>>         >>         return new FlightEntry(name);  
>>     }else if(type.equals("TrainEntry")) {  
>>         >>         return new TrainEntry(name);  
>>     }else {  
>>         >>         throw new IllegalArgumentException("输入错误, 请输入CourseEntry/FlightEntry/TrainEntry");  
>>     }  
}
```

方法	说明
accept()	判断计划项状态是否是可接收的
BeginPlanningEntry()	启动一个计划项
CancelPlanningEntry()	取消一个已阻塞计划项
EndPlanningEntry()	结束一个计划项
getBeginEndTime()	获取计划项的起止时间对
getName()	获得计划项的名字
getResource()	获取计划项所使用的资源
getState()	获得计划项的状态
getTimeLocation()	得到计划项所用的位置和时间对的对应关系
NewFlightEntry(java.lang.String type, java.lang.String name)	创建一个新的计划项
setName(java.lang.String name)	设置计划项的名字
setState(State state)	设置计划项的状态

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）

1.设计方案： 方案 5：CRP，通过接口组合

思想： ①把三个应用公有的特征抽象成一个公共的接口 PlanningEntry<R>

- ② 对于三个应用的不同维度的差异，为每个维度上的不同特征取值分别定义不同的接口，在接口中定义特殊的操作，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。

- ③ 对于具体的应用，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。
- ④ 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
以 TrainEntry 为例：通过接口的组合来形成满足应用的接口。

```
public interface TrainPlanningEntry
    extends MultipleLocationEntry,
           BlockableEntry,
           MultipleSortedResourceEntry { }

public class TrainEntry extends CommonPlanningEntry
    implements TrainPlanningEntry { }
```

2. 个性化特征方案

三个子类的差异方法就是 Location 、Timeslot 、Resource 属性的获取和修改。以及状态的转化。

TrainEntry：

构造器方法： 无参数、带参数

构造器	说明
TrainEntry()	不带参数的构造器方法
TrainEntry(java.lang.String name)	构造器方法

方法

方法	说明
addResource(Railway resource)	添加资源
Block()	计划项阻塞
deleteResource(Railway resource)	删除资源
getBeginEndTime()	获取计划项的起止时间对
getLocation()	get the Locations of the PlanningEntry
getResource()	获取计划项所使用的的资源
getResource()	
getTimeLocation()	得到计划项所用的位置和时间对的对应关系
getTimeslots()	获得一系列时间对
RunBlockedPlanningEntry()	启动阻塞的计划项
setLocations(java.util.List<Location> locs)	设置计划项的多个位置
setResource(java.util.List<Railway> r)	设置多个资源
setTimeslots(java.util.List<Timeslot> time)	设置一系列时间对
toString()	

CourseEntry:

构造器方法

构造器	说明
CourseEntry(java.lang.String name)	
CourseEntry(java.lang.String name, SingleLocationEntryImpl<loc> loc, UnBlockableEntryImpl<be> be, SingleDistinguishResourceEntryImpl<Teacher> res)	

其他方法

equals(java.lang.Object obj)	
getBeginEndTime()	获取计划项的起止时间对
getLocation()	get the Location of the PlanningEntry
getResource()	获取计划项所使用的的资源
getResource()	获得资源
getTime1()	获得起始时间
getTime2()	得到终止时间
getTimeLocation()	得到计划项所用的位置和时间对的对应关系
getTimeslot()	
hashCode()	
main(java.lang.String[] args)	
setLocation(Location loc)	设定计划项的位置
setResource(Teacher r)	设置资源
setTime(Timeslot timeslot)	
toString()	

FlightEntry

构造器方法

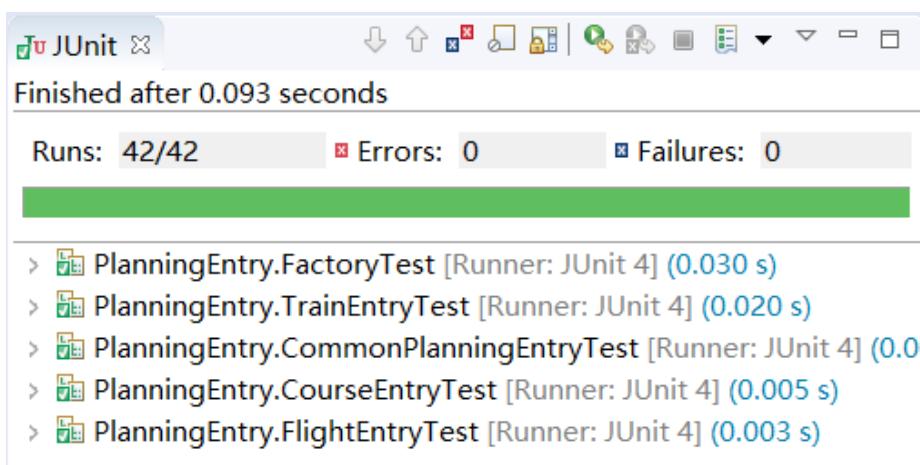
构造器	说明
FlightEntry(java.lang.String name)	
FlightEntry(java.lang.String Name, TwoLocationEntryImpl<loc> loc, SingleDistinguishResourceEntryImpl<Plane> res, UnBlockableEntryImpl<be> be)	

其他方法

方法	说明
getBeginEndTime()	获取计划项的起止时间对
getEndLocation()	get the end Location
getResource()	获取计划项所使用的的资源
getResource()	获得资源
getStartLocation()	get the start Location
getTime1()	获得起始时间
getTime2()	得到终止时间
getTimeLocation()	得到计划项所用的位置和时间对的对应关系
getTimeslot()	
setLocations(Location start, Location end)	设置计划项的起始和终止位置
setResource(Plane r)	设置资源
setTime(Timeslot timeslot)	

测试：

PlanningEntry	74.1 %	648	227	875
CourseEntry.java	54.8 %	182	150	332
CommonPlanningEntry.java	63.8 %	74	42	116
TrainEntry.java	88.7 %	220	28	248
PlanningEntry.java	0.0 %	0	7	7
Factory.java	100.0 %	35	0	35
FlightEntry.java	100.0 %	137	0	137



3.3 面向复用的设计：R

1. Plane

属性：

```
private String PlaneNumber ;//飞机编号
private String MachineNumber ;//机型号
private int Size ;//座位数
private double Age ;//机龄
```

checkRep:

```
>>> public void checkRep(){}
>>>   assert PlaneNumber != null;
>>>   assert MachineNumber != null ;
>>>   assert Size > 0 ;
>>>   assert Age > 0 ;
>>> }
```

方法：

方法	说明
checkRep()	初始化一架飞机
equals(java.lang.Object obj)	
getAge()	获得飞机机龄
getMachineNumber()	获得机型号
getPlaneNumber()	获得飞机编号
getSize()	获得座位个数
hashCode()	
setAge(double age)	设置飞机机龄
setMachineNumber(java.lang.String machineNumber)	设置机型号
setPlaneNumber(java.lang.String planeNumber)	设置飞机编号
setSize(int size)	设置座位个数
toString()	

2. Railway

属性：

```
private String Number ;//编号
private Type type ; //车厢类型
private int Size ; //定员数
private int Year ;//出厂年份
```

checkRep:

```
public void checkRep(){
    assert Number != null ;
    assert type != null ;
    assert Size > 0 && Year > 0 ;
}
```

方法：

方法	说明
checkRep()	
equals(java.lang.Object obj)	
getNumber()	获得编号
getSize()	获得定员数
getType()	获得车厢类型
getYear()	获得出厂年份
hashCode()	
setNumber(java.lang.String number)	设置编号
setSize(int size)	设置定员数
setType(Type type)	设置车厢类型
setYear(int year)	设置出厂年份
toString()	

3. Teacher

属性：

```
private String IdNumber;//教师的身份证号
private String Name;//教师的名字
private boolean Sex; //教师的性别,true表示男, false表示女
private String Title;//教师职称
```

checkRep:

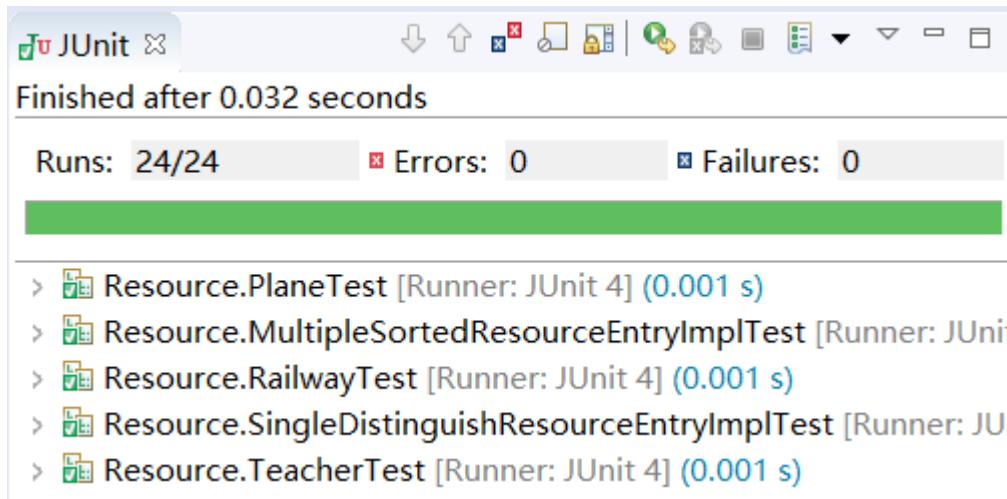
```
public void checkRep(){
    assert IdNumber != null;
    assert Name != null;
    assert Title != null;
}
```

方法

方法	说明
checkRep()	初始化
equals(java.lang.Object obj)	
getIdNumber()	获取教师的身份证号
getName()	获得教师的姓名
getTitle()	获得教师的职称
hashCode()	
isSex()	获得教师的性别,
main(java.lang.String[] args)	
setIdNumber(java.lang.String idNumber)	设置教师的身份证号
setName(java.lang.String name)	设置教师的姓名
setSex(boolean sex)	设置教师的性别
setTitle(java.lang.String title)	设置教师的职称
toString()	重写toString方法

测试：

Resource	75.0 %	779	260	1,039
SingleDistinguishResource	15.5 %	16	87	103
SingleDistinguishResource	15.5 %	16	87	103
MultipleSortedResourceEr	42.4 %	61	83	144
Teacher.java	83.7 %	211	41	252
Plane.java	87.2 %	211	31	242
Railway.java	91.6 %	196	18	214
Type.java	100.0 %	84	0	84



3.4 面向复用的设计：Location

属性：

```
private double longitude ; //经度
private double latitude ;//纬度
private String name ; //位置名称
private boolean share ;//位置是否可以共用, true 表示可以共用, false 表示不能
共用
```

checkRep:

```
> public void checkRep(){}
>   >> assert PlaneNumber != null;
>   >> assert MachineNumber != null ;
>   >> assert Size > 0 ;
>   >> assert Age > 0 ;
> }
```

Equals 方法：由于考虑到老师给的航班管理的文件对位置的读取没有经纬度的信息，而且老师说经纬度是为了以后复用可能被用来求解距离，所以我觉得在本次实验中，可以认为经纬度没有什么作用，只要位置的名字相等，就可以认为两个位置相同。

```

@>> @Override
>> public boolean equals(Object obj) {
>>     if (this == obj)
>>         return true;
>>     if (obj == null)
>>         return false;
>>     if (getClass() != obj.getClass())
>>         return false;
>>     Location other = (Location) obj;
>>     if (name == null) {
>>         if (other.name != null)
>>             return false;
>>     } else if (!name.equals(other.name))
>>         return false;
>>     return true;
>> }

```

其他方法

方法	说明
checkRep()	
equals(java.lang.Object obj)	
getLatitude()	get the latitude of the Location
getLongitude()	get the longitude of the Location
getName()	return the name of the Location
hashCode()	
isShare()	get the name of the Location
setLatitude(double latitude)	set the latitude of the Location
setLongitude(double longitude)	set the longitude of the Location
setName(java.lang.String name)	set the name of the Location
toString()	

测试：

Runs: 15/15 Errors: 0 Failures: 0

Location.java	82.3 %	102	22	124
Location	82.3 %	102	22	124

- > Location.TwoLocationEntryImplTest [Runner: JUnit 4] (0.000 s)
- > Location.SingleLocationEntryImplTest [Runner: JUnit 4] (0.000 s)
- > Location.LocationTest [Runner: JUnit 4] (0.000 s)
- > Location.MultipleLocationEntryImplTest [Runner: JUnit 4] (0.000 s)

3.5 面向复用的设计：Timeslot

1. 属性：

```

private Calendar date1 = Calendar.getInstance(); //起始时间
private Calendar date2 = Calendar.getInstance(); //终止时间

```

2. checkRep

```

>   public void checkRep() {
>     >>   assert date1 != null;
>     >>   assert date2 != null;
>     >>   assert date1.compareTo(date2) < 0;
>   }

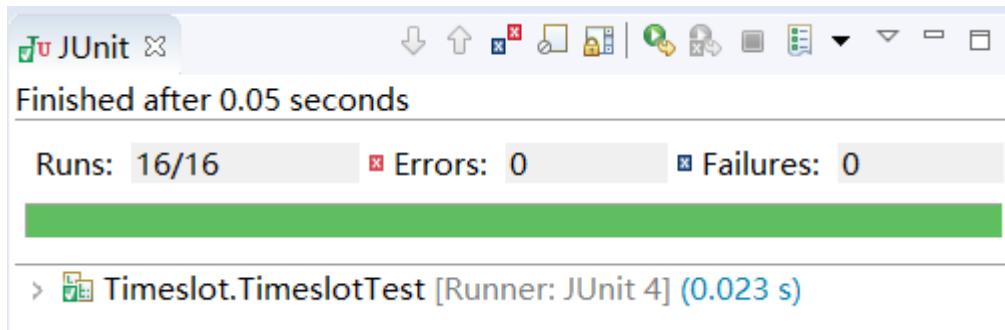
```

3. 方法

方法	说明
checkRep()	
equals(java.lang.Object obj)	
getdate1()	获得起始时间
getDate1()	get the string of the date1
getdate2()	获得终止时间
getDate2()	get the string of the date2
hashCode()	
setdate1(java.util.Calendar date1)	设置起始时间
setDate1(java.lang.String s)	set the date of the date1
setdate2(java.util.Calendar date2)	设置终止时间
setDate2(java.lang.String s)	set the date of the date2
toString()	

测试：

▼ Timeslot	73.0 %	170	63	233
▼ Timeslot.java	73.0 %	170	63	233
> Timeslot	73.0 %	170	63	233



3.6 面向复用的设计：EntryState 及 State 设计模式

1.State 接口

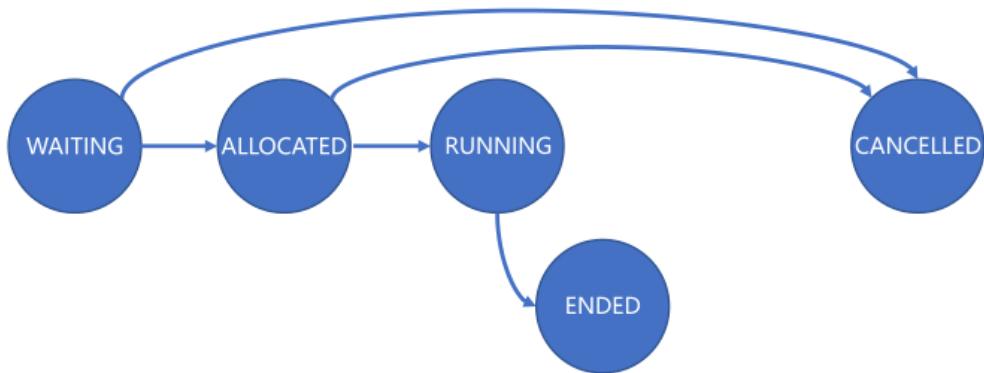
State 接口时状态类的公共接口，只有两个方法，改变状态和判断是否是可接受转态

方法	说明
accept()	状态是否是可接受
changeState(java.lang.String type)	改变状态

2.具体的状态类

WAITTING.java//未分配资源、

ALOCATED.java//已分配资源但未启动
 RUNNING.java//已启动
 CANCELLED.java//已取消
 ENDED.java//已完成
 BLOCKED.java//挂起



由状态关系转移图可知，对于每个状态类来说，只能转换为特定的状态类。

- ① 以 WAITTING 状态为例，只能转换为 ALOCATED 和 CANCELLED 两个状态，并且状态不可接受

```

```java
public static WAITTING instance = new WAITTING();
private WAITTING() {};
@Override
public State changeState(String type) {
 switch(type) {
 case "ALLOCATED": return ALOCATED.instance;
 case "CANCELLED": return CANCELLED.instance;
 default: throw new IllegalArgumentException("WAITTING状态不能转换为" + type + "状态");
 }
}
@Override
public boolean accept() {
 return false;
}
@Override
public String toString() {
 return "WAITTING";
}
```

```

- ② ALOCATED 状态

```

```java
public State changeState(String type) {
 switch(type) {
 case "RUNNING": return RUNNING.instance;
 case "CANCELLED": return CANCELLED.instance;
 default: throw new IllegalArgumentException();
 }
}
@Override
public boolean accept() {
 return false;
}
```

```

- ③ RUNNING 状态

```

'>    @Override
'>    public State changeState(String type) {
'>        switch(type) {
'>            case "ENDED": return ENDED.instance;
'>            case "BLOCKED": return BLOCKED.instance;
'>            default : throw new IllegalArgumentException("RUNNING状态不能转换为" + type + "状态");
'>        }
'>    }
'>
'>    @Override
'>    public boolean accept() {
'>        return false;
'>    }
'>

```

④ BLOCKED 状态

```

'>    @Override
'>    public State changeState(String type) {
'>        switch(type) {
'>            case "RUNNING" : return RUNNING.instance;
'>            case "CANCELLED" : return CANCELLED.instance;
'>            default : throw new IllegalArgumentException("BLOCKED状态不能转换为" + type + "状态");
'>        }
'>    }
'>
'>    @Override
'>    public boolean accept() {
'>        return false;
'>    }
'>

```

⑤ CANCELLED 状态

```

'>    @Override
'>    public State changeState(String type) {
'>        throw new IllegalArgumentException("CANCELLED状态不能转换为" + type + "状态");
'>    }
'>
'>    @Override
'>    public boolean accept() {
'>        return true;
'>    }
'>

```

⑥ ENDED 状态

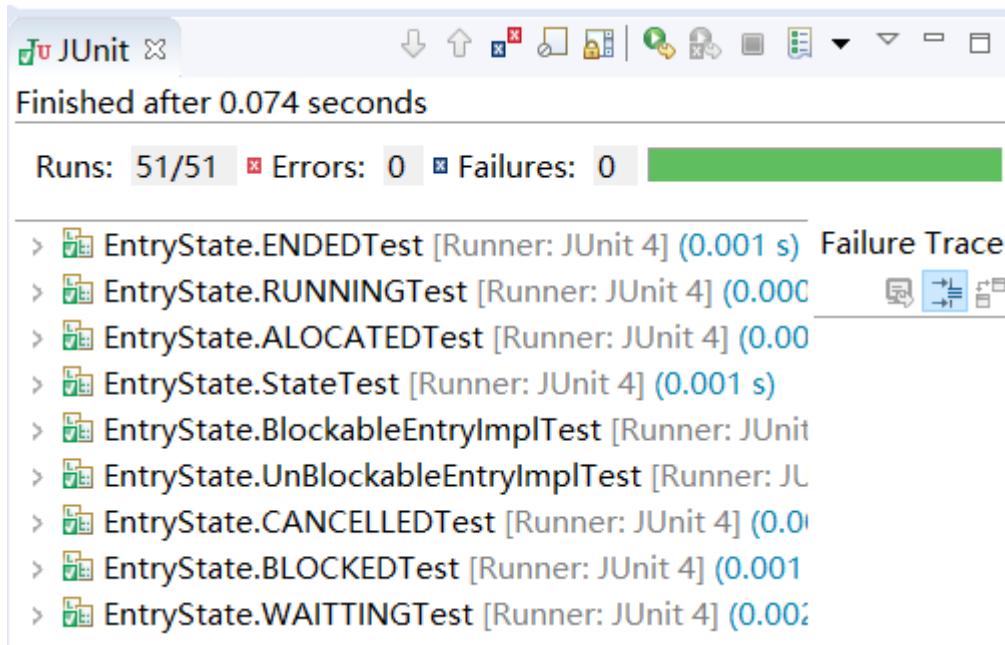
```

'>    @Override
'>    public State changeState(String type) {
'>        throw new IllegalArgumentException("ENDED状态不能转换为" + type + "状态");
'>    }
'>
'>    @Override
'>    public boolean accept() {
'>        return true;
'>    }
'>

```

测试：

| | | | | |
|-------------------|---------|----|---|----|
| > WAITING.java | 93.5 % | 29 | 2 | 31 |
| > ALOCATED.java | 100.0 % | 22 | 0 | 22 |
| > BLOCKED.java | 100.0 % | 31 | 0 | 31 |
| > CANCELLED.java | 100.0 % | 22 | 0 | 22 |
| > ENDED.java | 100.0 % | 22 | 0 | 22 |
| > RUNNING.java | 100.0 % | 31 | 0 | 31 |



3.7 面向应用的设计：Board

思路:对于每一个信息板，都有一个当前时间，当前位置与他相对应，并且包含了一组计划项。因此属性就是**当前时间、当前位置、一组计划项**

功能: ①对于每个信息板，首先实现按照实验指导书所说显示出发生在计划板所在位置和当前时间相差一个小时内的计划项信息，②其次展示所有的发生在该计划板的计划项信息(没有相差一个小时的限制)。

实现方法:首先在所有的计划项中找出和信息板位置相关的计划项，将他们按照发生在当前位置的时间排好序。然后利用 jtable 组件可视化信息板即可。

注意：而且当前时间并不是系统显示的时间，每次当前时间都是需要

由用户输入的。

1. CourseCalendarBoard

属性:

```
private Location location ;//信息板所在的位置
private List<CourseEntry> course = new ArrayList<CourseEntry>(); //一系列的计划项
private Calendar time = Calendar.getInstance() ; //当前时间
```

方法:

| 方法 | 说明 |
|---|----------------------|
| changestate(CourseEntry course) | 根据当前时间来设置高铁的状态 |
| getCourse() | 得到所有位置是location的计划项 |
| isSameDay(java.util.Calendar call, java.util.Calendar cal2) | 判断给定的时间是否是同一天 |
| iterator() | 实现迭代器 |
| show() | 展示和当前信息板所在的位置相关的所有课程 |
| visualize() | 可视化信息板 |

注释： **visualize** 方法显示的是按照实验指导书所说显示出发生在计划板所在位置和当前时间**同一天的课程信息**。

show 方法显示的是发生在当前位置的所有计划项的信息，**没有时间要求**。

changestate 方法：由于每次显示信息板都要输入当前时间，那么每次输入当前时间的不同，相较于用户输入的当前时间而言，所有的课程的状态也是不同的，因此 **changestate** 方法就是**根据用户输入的当前时间来改变计划项的状态**。

```
public void changestate(CourseEntry course) {
    if(course.getResource() == null){}
    course.setState(WAITTING.instance); //课程还未分配教师
    return;
}
if(time.compareTo(course.getTime1()) < 0){}
course.setState(ALLOCATED.instance); //课程还未发开始，设置状态为ALLOCATED
return;
}else if(time.compareTo(course.getTime2()) > 0){}
course.setState(ENDED.instance); //课程已经结束，设置状态为ENDED
return;
}else{
course.setState(RUNNING.instance); //课程正在进行，设置状态为RUNNING
return;
}
}
```

getCourse()方法： 得到的是所有发生在该位置的课程

```
public List<CourseEntry> getCourse() {
    List<CourseEntry> t = new ArrayList<>(); //所有位置是location的课程
    for(int i = 0; i < course.size(); i++){
        if(course.get(i).getLocation().equals(location)){
            t.add(course.get(i));
        }
    }
    Collections.sort(t, new CourseCompare()); //按时间排序
    for(CourseEntry e : t){
        changestate(e); //根据当前时间设定课程的状态
    }
    return t;
}
```

isSameDay()方法： 判断两个时间是否是同一天

2. FlightScheduleBoard

属性： `private Calendar time = Calendar.getInstance(); //屏幕版当前时间`
`private Location location; //屏幕板所在的位置`
`private List<FlightEntry> flight = new ArrayList<FlightEntry>(); //`
 一系列的计划项

方法：

| | |
|--|----------------------------------|
| <code>changeState(FlightEntry flight)</code> | 根据当前的时间来设置航班的状态 |
| <code>getFlight1()</code> | 获得所有出发位置在location并且按照出发时间递增排序的航班 |
| <code>getFlight2()</code> | 获得所有到达位置在location并且按照到达时间递增排序的航班 |
| <code>iterator()</code> | 迭代器 |
| <code>show()</code> | 展示出所有与该位置相关的航班信息 |
| <code>visualize(int x)</code> | 可视化展示出当前位置的信息板 |

注释： `visualize` 方法显示的是按照实验指导书所说显示出发生在计划板所在位置和当前时间相差一个小时的航班信息。

`show` 方法显示的是发生在当前位置的所有计划项的信息，没有时间要求。

Show 方法示例

| 2020-01-03 18:30(当前时间), Shanghai机场 | | | | | |
|------------------------------------|--------------------------------------|--------|-------------------|-----------|--|
| 序号 | 时间 | 航班 | 行程 | 状态 | |
| 1 | 2020-01-01 12:59 -> 2020-01-01 14:19 | MF85 | Weihai-Shanghai | ENDED | |
| 2 | 2020-01-02 12:59 -> 2020-01-02 14:19 | MF85 | Weihai-Shanghai | ENDED | |
| 3 | 2020-01-03 12:59 -> 2020-01-03 14:19 | MF85 | Weihai-Shanghai | ENDED | |
| 4 | 2020-01-04 12:59 -> 2020-01-04 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 5 | 2020-01-05 12:59 -> 2020-01-05 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 6 | 2020-01-06 12:59 -> 2020-01-06 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 7 | 2020-01-07 12:59 -> 2020-01-07 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 8 | 2020-01-08 12:59 -> 2020-01-08 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 9 | 2020-01-09 01:59 -> 2020-01-09 05:15 | AA754 | Urumqi-Shanghai | ALLOCATED | |
| 10 | 2020-01-09 12:59 -> 2020-01-09 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 11 | 2020-01-10 12:59 -> 2020-01-10 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 12 | 2020-01-11 12:59 -> 2020-01-11 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 13 | 2020-01-12 12:59 -> 2020-01-12 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 14 | 2020-01-13 12:59 -> 2020-01-13 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 15 | 2020-01-14 12:59 -> 2020-01-14 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 16 | 2020-01-15 12:59 -> 2020-01-15 14:19 | MF85 | Weihai-Shanghai | ALLOCATED | |
| 17 | 2020-01-01 01:00 -> 2020-01-01 06:31 | ZH85 | Shanghai-Dalian | ENDED | |
| 18 | 2020-01-01 08:11 -> 2020-01-01 11:16 | FM7079 | Shanghai-Nanning | ENDED | |
| 19 | 2020-01-01 08:51 -> 2020-01-01 12:41 | GS4209 | Shanghai-Shenyang | ENDED | |
| 20 | 2020-01-02 01:00 -> 2020-01-02 06:31 | ZH85 | Shanghai-Dalian | ENDED | |

Visualize 方法示例



3. TrainScheduleBoard

属性:

```
private Calendar time = Calendar.getInstance() ;//屏幕版当前时间
private Location location ; //屏幕板所在的位置
private List<TrainEntry> train = new ArrayList<TrainEntry>() ;//一系列的计划项
```

方法:

| 方法 | 说明 |
|--|--------------------------------|
| changeState(TrainEntry train) | 根据当前的时间来设置高铁的状态 |
| getTrain() | 得到所有和location相关的高铁车次 |
| gettrain1() | 得到所有的抵达location的高铁车次及其对应的抵达时间 |
| gettrain2() | 得到所有的从location出发的高铁车次及其对应的抵达时间 |
| iterator() | 迭代器 |
| show(Location loc, java.util.Calendar time1) | 列出和某个位置相关的所有计划项的信息 |
| visualize(int x) | 可视化信息板 |

注释： `visualize` 方法显示的是按照实验指导书所说显示出发生在计划板所在位置和当前时间相差一个小时的高铁车次信息。

`show` 方法显示的是发生在当前位置的所有计划项的信息，没有时间要求。

设计思路： 对于每一个经停 `location` 的高铁车次(由 `getTrain` 方法得到)，都对应一个抵达时间/出发时间，所以我是实现了 `gettrain1` 方法，得到所有抵达 `location` 的高铁车次和对应的抵达时间，这种对应关系用 `Treemap<Calendar, TrainEntry>` 存储，然后对键按照时间进行排序。然后 `visualize` 方法就可以按照时间增序显示所有的高铁车次信息。同理对于出发车次也是如此。

`changestate` 方法：由于每次显示信息板都要输入当前时间，那么每次输入当前

时间的不同，相较于用户输入的当前时间而言，所有的高铁车次的状态也是不同的，因此 changeState 方法就是根据用户输入的当前时间来改变计划项的状态。

getTrain1 方法

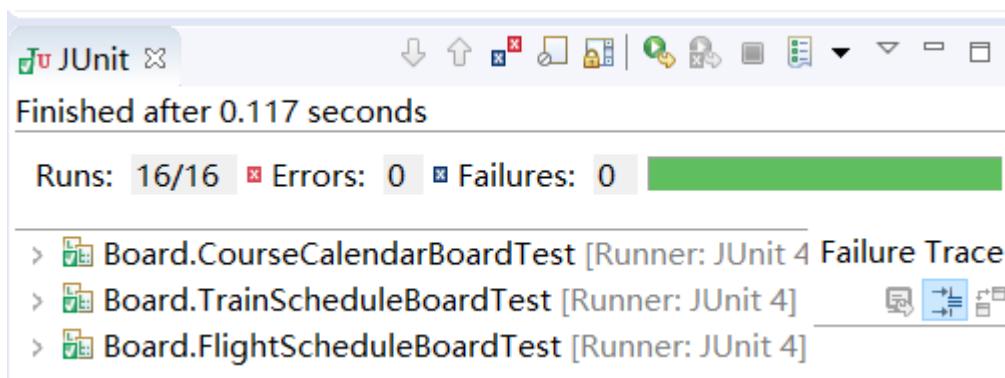
```

>> /**
>> * 得到所有的抵达location的高铁车次及其对应的抵达时间
>> */
>> public Map<Calendar,TrainEntry> getTrain1(){
>>     List<TrainEntry> trains = getTrain();
>>     Map<Calendar,TrainEntry> t = new TreeMap<>();//按照键的时间大小进行排序
>>     >>     new Comparator<Calendar>(){
>>         >>     @Override
>>         >>     public int compare(Calendar o1, Calendar o2) {
>>             >>             if(o1.getTime().before(o2.getTime())){
>>                 >>                 return -1;
>>             }else if(o1.getTime().before(o2.getTime())){
>>                 >>                 return 1;
>>             }
>>             >>             return 0;
>>         }
>>     });
>>     for(int i = 0 ; i < trains.size() ; i++ ) {
>>         for(int j = 1 ; j < trains.get(i).getLocation().size() ; j++ ){
>>             if(trains.get(i).getLocation().get(j).equals(location)){
>>                 t.put(trains.get(i).getTimeslots().get(j-1).getDate2(),trains.get(i));
>>             }
>>         }
>>     }
>>     return t;
>> }

```

测试：

| | Board | 38.9 % | 819 | 1,289 | 2,108 |
|---|--------------------------|--------|-----|-------|-------|
| > | FlightScheduleBoard.java | 31.6 % | 220 | 476 | 696 |
| > | TrainScheduleBoard.java | 47.5 % | 366 | 405 | 771 |
| > | CourseCalendarBoard.java | 37.3 % | 184 | 309 | 493 |
| > | Show.java | 0.0 % | 0 | 85 | 85 |
| > | FlightCompare2.java | 52.4 % | 11 | 10 | 21 |
| > | CourseCompare.java | 90.5 % | 19 | 2 | 21 |
| > | FlightCompare1.java | 90.5 % | 19 | 2 | 21 |



3.8 Board 的可视化：外部 API 的复用

只要把所有显示的东西放进一个二维数组，再利用 jTable 组件就能显示出当前的信息板。

Visualize 方法思路：首先用 getcourse 得到所有按时间排好序的和该位置相关的课程信息，然后遍历，如果课程和当前时间是同一天，就把相应的信息放进二维数组内。最后借助于外部 API 就可以显示出来。

```

>>> public void visualize(){  

>>>     SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");  

>>>     String s = sdf.format(time.getTime());  

>>>     SimpleDateFormat sdf1 = new SimpleDateFormat("HH:mm");  

>>>     List<CourseEntry> t_ = getCourse();  

>>>     String[] columnNames = {"序号", "时间", "课程", "教师", "状态"};  

>>>     List<CourseEntry> t1_ = new ArrayList<CourseEntry>(); //所有和当前时间同一天的课程  

>>>     for(int i = 0 ; i < t_.size() ; i++) {  

>>>         if(isSameDay(time, t_.get(i).getTime1())) {  

>>>             t1_.add(t_.get(i));  

>>>         }  

>>>     }  

>>>     Object[][] rowData = new String[t1_.size()][];  

>>>     for(int i = 0 ; i < t1_.size() ; i++) {  

>>>         rowData[i] = new String[] { String.valueOf(i+1) ,sdf1.format(t1_.get(i).getTime1().getTime()) + " - " + sdf1.format(t1_.get(i).getTime2().getTime()),  

>>>             t1_.get(i).getName(),t1_.get(i).getResource().getName(),t1_.get(i).getState().toString()};  

>>>     }  

>>>     JFrame jf = new JFrame(s + "(当前时间" + " , " + location.getName());  

>>>     Show.show(columnNames, rowData, jf);  

>>>

```

→ 将统一的外部API调用封装起来

说明： Show 类的 show 方法，参数是一个二维数组 rowData, 和一维数组(第一行信息) 和一个 JFrame 容器。该方法用来可视化。

```

>>> /**
>>> * 绘制表格
>>> * @param a 表格的第一项
>>> * @param t 表格的每一项信息
>>> * @param jf JFrame容器
>>> */
>>> public static void show(String[] a, Object[][] t, JFrame jf) {
>>>     jf.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
>>>     JPanel panel = new JPanel();
>>>     JTable table = new JTable(t, a);
>>>     // 设置表头
>>>     table.getTableHeader().setFont(new Font(null, Font.BOLD, 14)); //设置表头名称字体样式
>>>     table.getTableHeader().setForeground(Color.RED); //设置表头名称字体颜色
>>>     table.getTableHeader().setResizingAllowed(false); //设置不允许手动改变列宽
>>>     table.getTableHeader().setReorderingAllowed(false); //设置不允许拖动重新排序各列
>>>     table.setRowHeight(30);
>>>     table.getColumnModel().getColumn(0).setPreferredWidth(80);
>>>     table.setPreferredScrollableViewportSize(new Dimension(1200, 600));
>>>     JScrollPane scrollPane = new JScrollPane(table);
>>>     panel.add(scrollPane);
>>>     jf.setContentPane(panel);
>>>     jf.pack();
>>>     jf.setLocationRelativeTo(null);
>>>     jf.setVisible(true);
>>>     DefaultTableCellRenderer r = new DefaultTableCellRenderer();
>>>     r.setHorizontalAlignment(JLabel.CENTER);
>>>     table.setDefaultRenderer(Object.class, r);
>>>

```

Show 方法：和 visualize 方法类似，只是少了判断和当前时间是否是同一天的约束。

```

/*
 * 展示和当前信息板所在的位置相关的所有课程
 */
public void show() {
    SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy-MM-dd HH:mm");
    String s = sdf1.format(time.getTime());
    List<CourseEntry> t1_ = getCourse(); //和某个位置相关的所有课程
    String[] columnNames = {"序号", "时间", "课程", "教师", "状态"}; //表格第一行信息
    Object[][] rowData = new String[t1_.size()][];
    for(int i = 0 ; i < t1_.size() ; i++) {
        rowData[i] = new String[] { String.valueOf(i+1) ,sdf1.format(t1_.get(i).getTime1().getTime()) + " - " + sdf1.format(t1_.get(i).getTime2().getTime()),t1_.get(i).getName(),t1_.get(i).getResource().getName(),t1_.get(i).getState().toString()};
    }
    JFrame jf = new JFrame(s + "(当前时间" + " , " + location.getName());
    Show.show(columnNames, rowData, jf);
}

```

3.9 可复用 API 设计及 Façade 设计模式

Façade 设计模式：客户端需要通过一个简化的接口来访问复杂系统内的功能，我们只需要提供一个统一的接口来取代一系列小接口调用，相当于对复杂系统做了一个封装，简化客户端使用。

在 lab3 可复用 API 设计中，我们需要把检查资源冲突、检查位置冲突、获取前序计划项的方法封装在一个 APIs 类中，通过提供这个统一的类来提供功能，简化客户端的使用。

3.9.1 检测一组计划项之间是否存在位置独占冲突

位置冲突：即不同的计划项在同一个时间点上占用了相同的不可共享位置(在这个意义上，这个方法不适用与高铁车次应用和航班应用)。

设计思路：对于使用不可共享的位置计划项而言，遍历每个计划项，如果任意两个计划项的运行时间存在重叠，则比较他们的位置，如果位置相等，则说明存在位置冲突。否则不存在位置冲突。

```

/*
 * 检查计划项安排是否存在位置冲突
 * @param entries 计划项列表
 * @return 位置存在冲突，返回false，位置不存在冲突，返回true
 */
public static boolean checkLocationConflict(List<PlanningEntry> entries) {
    boolean flag = true;
    for(int i = 0 ; i < entries.size() ; i++) {
        Set<Timeslot> time1 = entries.get(i).getTimeLocation().keySet();
        for(Timeslot e : time1) {
            for( int j = i + 1 ; j < entries.size() ; j++) {
                Set<Timeslot> time2 = entries.get(j).getTimeLocation().keySet();
                for(Timeslot f : time2) {
                    Date time11 = e.getDate1().getTime(); //第1个计划项的起始时间
                    Date time22 = e.getDate2().getTime(); //第1个计划项的结束时间
                    Date time33 = f.getDate1().getTime(); //第2个计划项的起始时间
                    Date time44 = f.getDate2().getTime(); //第2个计划项的结束时间
                    if((time11.before(time33) && time22.after(time33)) || (time11.after(time33) && time11.before(time44))) { //时间存在重叠，则判断位置是否重叠
                        List<Location> lc1 = (List<Location>) entries.get(i).getTimeLocation().get(e); //计划项1的所有位置
                        List<Location> lc2 = (List<Location>) entries.get(j).getTimeLocation().get(f); //计划项2的所有位置
                        for(int k = 0 ; k < lc1.size() ; k++) {
                            for(int w = 0 ; w < lc2.size() ; w++) {
                                if(lc1.get(k).equals(lc2.get(w))) { //位置相同
                                    System.out.println(entries.get(i).getName() + " 和 " + entries.get(j).getName() + " 存在位置: (" + lc1.get(k).getName() + " ) 冲突"); //打印冲突信息
                                    flag= false ;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return flag;
}

```

3.9.2 检测一组计划项之间是否存在资源独占冲突

资源冲突：即不同的计划项在同一个时间点上占用了相同的资源

设计思路 1：对于使用不可共享的位置计划项而言，遍历每个计划项，如果任意两个计划项的运行时间存在重叠，则比较他们的位置，如果位置相等，则说明存在位置冲突。否则不存在位置冲突。

```

@Override
public boolean checkResourceExclusiveConflict(List<PlanningEntry> entries) {
    boolean flag = true;
    for(int i = 0 ; i < entries.size() ; i++) {
        for(int j = i + 1 ; j < entries.size() ; j++) {
            Calendar time1 = entries.get(i).getBeginEndTime().getdate1(); //第i个计划项的起始时间
            Calendar time2 = entries.get(i).getBeginEndTime().getdate2(); //第i个计划项的结束时间
            Calendar time3 = entries.get(j).getBeginEndTime().getdate1(); //第j个计划项的起始时间
            Calendar time4 = entries.get(j).getBeginEndTime().getdate2(); //第j个计划项的结束时间
            if((time1.compareTo(time3) < 0 && time2.compareTo(time3) > 0) || ((time1.compareTo(time3) > 0 && time1.compareTo(time4) < 0)) { //时间重叠
                for(int k = 0 ; k < entries.get(i).getResource().size() ; k++) {
                    for(int w = 0 ; w < entries.get(j).getResource().size() ; w++) {
                        if(entries.get(i).getResource().get(k).equals(entries.get(j).getResource().get(w))) //资源相同
                            System.out.println(entries.get(i).getName() + " 和 " + entries.get(j).getName() + " 存在资源冲突");
                        flag = false;
                    }
                }
            }
        }
    }
    return flag;
}

```

设计思路 2：对于使用不可共享的位置计划项而言，遍历每个计划项，如果任意两个计划项存在使用相同的资源，则比较他们的运行时间，如果运行时间重叠，则说明存在位置冲突。否则不存在位置冲突。

```

@Override
public boolean checkResourceExclusiveConflict(List<PlanningEntry> entries) {
    boolean flag = true;
    for(int i = 0 ; i < entries.size() ; i++) {
        for(int j = i + 1 ; j < entries.size() ; j++) {
            for(int k = 0 ; k < entries.get(i).getResource().size() ; k++) {
                for(int w = 0 ; w < entries.get(j).getResource().size() ; w++) {
                    if(entries.get(i).getResource().get(k).equals(entries.get(j).getResource().get(w))) //资源相同
                        System.out.println(entries.get(i).getName() + " 和 " + entries.get(j).getName() + " 存在资源冲突");
                    flag = false;
                }
            }
        }
    }
    return flag;
}

```



采用策略模式，由用户来选择调用的方法。其中公共接口是 `CheckResourceExclusiveConflict.java`，两种不同的实现分别是 `checkResourceExclusiveConflict1.java`

和 `checkResourceExclusiveConflict2.java`。根据传入的参数来选择调用哪个方法来实现。

```

/**
 * 检查计划项安排是否存在资源冲突
 * @param entries 计划项列表
 * @param s 输入的类型，根据类型来选择调用哪个方法
 * @return 资源存在冲突，返回false，位置不存在冲突，返回true
 */
public static boolean checkResourceExclusiveConflict(List<PlanningEntry> entries, CheckResourceExclusiveConflict s) {
    return s.checkResourceExclusiveConflict(entries);
}

```

3.10 节模式的引用中，策略模式会重新提及这个方法。

3.9.3 提取面向特定资源的前序计划项

特定资源的前序计划项：即在一组计划项中，针对某个资源 r 和使用 r 的某个计划项 e ，从一组计划项中找出 e 的前序 f ， f 也使用资源 r ， f 的执行时间在 e 之前，且在 e 和 f 之间不存在使用资源 r 的其他计划项。

设计思路：首先初始化一个前序计划项 x 和对应的最晚结束时间 $latestTime$ ，遍历所有的计划

项，如果某个计划项和计划项 e 共用资源 r，而且该计划项的结束时间晚于 latestTime，则重新设定前序计划项 e 和最晚结束时间 latestTime。循环结束，返回 x 即可。

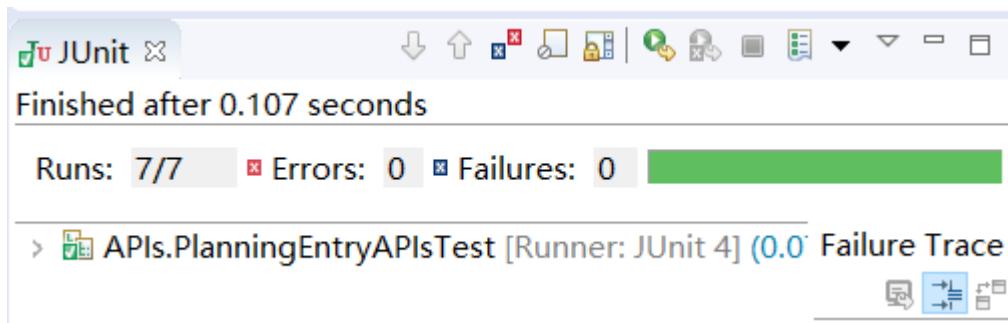
```

/*
 * 在一系列计划项中找出共用某个资源的某个计划项的前序计划项
 * @param <R> 泛型，代表资源的类型
 * @param r 共用的某个资源
 * @param e 计划项
 * @param entries 一系列计划项
 * @return
 */
public static<R> PlanningEntry<R> findPreEntryPerResource(R r, PlanningEntry<R> e, List< ? extends PlanningEntry<R>> entries) {
    PlanningEntry<R> x = null; //初始化计划项
    Calendar latestTime = null; //在选定的计划项e之前且最接近e的结束时间的计划项的结束时间
    for(int i = 0 ; i < entries.size() ; i++) {
        if(entries.get(i).getResource().contains(r)) {
            Calendar time = entries.get(i).getBeginEndTime().getdate2();
            if(time.compareTo(e.getBeginEndTime().getdate1()) < 0) { //e的前序计划项
                latestTime = time; //初始化
                x = entries.get(i); //改变最接近e的计划项
            } else {
                if(time.compareTo(latestTime) > 0) { //改变最接近e的计划项的时间
                    latestTime = time; //改变最接近e的计划项的时间
                    x = entries.get(i); //改变最接近e的计划项
                }
            }
        }
    }
    return x;
}

```

测试：

| APIs | 98.6 % | 509 | 7 | 516 |
|--------------------------|---------|-----|---|-----|
| PlanningEntryAPIs.java | 96.9 % | 221 | 7 | 228 |
| checkResourceExclusiveCo | 100.0 % | 144 | 0 | 144 |
| checkResourceExclusiveCo | 100.0 % | 144 | 0 | 144 |



3.10 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。

3.10.1 Factory Method

思路：设计一个工厂类统一的接口 factoryInterface，工厂类 Factory 继承这个接口并实现里面的工厂方法。工厂类的工厂方法可以根据指定的类型来创建相应的计划项。

```

public interface FactoryInterface {
    /**
     * 工厂方法
     * @param type 类型
     * @param name 计划项名字
     * @return 指定类型的计划项
     */
    public PlanningEntry getPlanningEntry(String type, String name);
}

public class Factory implements FactoryInterface {
    @Override
    public PlanningEntry getPlanningEntry(String type, String name) {
        if(type.equals("CourseEntry")) {
            return new CourseEntry(name);
        } else if(type.equals("FlightEntry")) {
            return new FlightEntry(name);
        } else if(type.equals("TrainEntry")) {
            return new TrainEntry(name);
        } else {
            throw new IllegalArgumentException("输入错误，请输入CourseEntry/FlightEntry/TrainEntry");
        }
    }
}

```

在 PlanningEntry 类，就可以调用工厂方法来创建一个新的计划项。

```

static<R> PlanningEntry<R> NewFlightEntry(String type, String name) {
    return new Factory().getPlanningEntry(type, name);
}

```

3.10.2 Iterator

我的思路：让 board 类实现 Iterable 接口，并且实现 Iterator 方法即可。对于每一个信息板而言，都对应一个当前位置，信息板的迭代器只需要遍历和这个位置相关的计划项即可。首先，先得到所有和当前位置相关的计划项，然后按照时间进行排序。最后实现一个自己的迭代器，board 类的迭代器返回自己实现的迭代器即可。

1.CourseCalendarBoard 的迭代器方法

自己实现的迭代器：

属性：`private List<CourseEntry> course1 = getCourse(); // 获取和当前计划板所在的位置相关的计划项`

```

private int curror = -1; //
private int size = course1.size(); // 计划项的数量

```

方法： `hasNext` : 判断是否还有下个元素
`next`: 下一个元素
`remove` 方法: 本次实验用不上这个方法，所以不需要实现这个方法

实现 Comparator 方法:

```

public class CourseCompare implements Comparator<CourseEntry> {
    @Override
    public int compare(CourseEntry o1, CourseEntry o2) {
        if(o1.getTime1().compareTo(o2.getTime1()) < 0) {
            return -1;
        } else if(o1.getTime1().compareTo(o2.getTime1()) > 0) {
            return 1;
        }
        return 0;
    }

    public class PlanningEntryInteator implements Iterator<PlanningEntry> {
        private List<CourseEntry> course1 = getCourse(); // 获取和当前计划板所在的位置相关的计划项
        private int cursor = -1; // 
        private int size = course1.size(); // 计划项的数量
        @Override
        public boolean hasNext() {
            return cursor + 1 < size;
        }

        @Override
        public PlanningEntry next() {
            cursor++;
            return course1.get(cursor);
        }
    }
}

```

因此，在 `CourseCalendarBoard` 类中，只要返回我们实现的迭代器即可。

```

/**
 * 实现迭代器
 */
@Override
public Iterator<PlanningEntry> iterator() {
    return new PlanningEntryInteator();
}

```

2. FlightScheduleBoard 的迭代器方法

自己实现的迭代器:

属性: `private List<FlightEntry> flight1 = getFlight2(); // 抵达位置是`

```
location的所有航班  
private List<FlightEntry> flight2 = getFlight1() ;//出发位置是  
location的所有航班  
  
private int curror = -1;  
private int size1 = flight1.size() ; //抵达位置是location的航班个数  
private int size2 = flight2.size(); //出发位置是 location 的航班的个数
```

属性说明:由于机场对应抵达航班和出发航班，所以迭代器要遍历所有的抵达航班和出发航班。

方法: `hasNext` , `next` 方法同上

实现 Comparator 方法：

```
    /**
     * 将高铁车次按照抵达location的时间进行升序
     */
    @Override
    public int compare(FlightEntry o1, FlightEntry o2) {
        if(o1.getTime1().compareTo(o2.getTime1()) < 0) {
            return -1;
        } else if(o1.getTime1().compareTo(o2.getTime1()) > 0) {
            return 1;
        }
        return 0;
    }
}

public class FlightCompare2 implements Comparator<FlightEntry> {
    /**
     * 将高铁车次按照从location出发的时间进行升序
     */
    @Override
    public int compare(FlightEntry o1, FlightEntry o2) {
        if(o1.getTime2().compareTo(o2.getTime2()) < 0) {
            return -1;
        } else if(o1.getTime2().compareTo(o2.getTime2()) > 0) {
            return 1;
        }
        return 0;
    }
}

public class PlanningEntryIterator implements Iterator<PlanningEntry> {
    private List<FlightEntry> flight1 = ...getFlight2(); //抵达位置是location的所有航班
    private List<FlightEntry> flight2 = ...getFlight1(); //出发位置是location的所有航班
    private int cursor = -1;
    private int size1 = flight1.size(); //抵达位置是location的航班个数
    private int size2 = flight2.size(); //出发位置是location的航班的个数

    @Override
    public boolean hasNext() {
        return cursor + 1 < (size1 + size2);
    }

    @Override
    public PlanningEntry next() {
        cursor++;
        if(cursor < size1) {
            return flight1.get(cursor); //先遍历抵达航班
        } else {
            return flight2.get(cursor-size1); //再遍历出发航班
        }
    }
}
```

最后 FlightScheduleBoard 类中，只要返回我们实现的迭代器即可。

```
/* * 迭代器 */
@Override
public Iterator<PlanningEntry> iterator() {
    return new PlanningEntryInteator();
}
```

3. TrainScheduleBoard 方法

属性: `private List<TrainEntry> t = getTrain(); //所以经停location的高铁车次`

```
private int curror = -1;
private int size = t.size(); //高铁车次个数
```

属性说明：所有经停location的高铁车次已经按照时间排好序

实现 Comparator 方法：

```
Collections.sort(Train, new Comparator<TrainEntry>() {//按照出发时间排序}
    @Override
    public int compare(TrainEntry o1, TrainEntry o2) {
        return o1.getBeginEndTime().getdate1().compareTo(o2.getBeginEndTime().getdate1());
    }
});
```

```
public class PlanningEntryInteator implements Iterator<PlanningEntry> {
    private List<TrainEntry> t = getTrain(); //所以经停location的高铁车次
    private int curror = -1;
    private int size = t.size(); //高铁车次个数
    @Override
    public boolean hasNext() {
        return curror + 1 < size;
    }
    @Override
    public PlanningEntry next() {
        curror++;
        return t.get(curror);
    }
}
```

最后 TrainScheduleBoard 类中，只要返回我们实现的迭代器即可。

```
/* * 迭代器 */
@Override
public Iterator<PlanningEntry> iterator() {
    return new PlanningEntryInteator();
}
```

3.10.3 Strategy

策略模式:同一个功能用不同的方法去实现，具体客户端选择哪种实现由用户决定。

思路：我把策略模式用在了 3.9 节中判断资源冲突的方法中。首先创建一个公共接口

CheckResourceExclusiveConflict.java，两种不同的实现分别是checkResourceExclusiveConflict1.java

和 checkResourceExclusiveConflict2.java。根据传入的参数来选择调用哪个方法来实现。

```
public class PlanningEntryList {
    ...
    /**
     * 检查计划项安排是否存在资源冲突
     * @param entries 计划项列表
     * @param os 输入的类型, 根据类型来选择调用哪个方法
     * @return 资源存在冲突, 返回false, 位置不存在冲突, 返回true
     */
    public static boolean checkResourceExclusiveConflict(List<PlanningEntry> entries, CheckResourceExclusiveConflict s) {
        ...
        return s.checkResourceExclusiveConflict(entries);
    }
}
```

客户端具体调用哪种方法由传入的参数决定



```
public interface CheckResourceExclusiveConflict {
    /**
     * 检查计划项安排是否存在资源冲突
     * @param entries 计划项列表
     * @return 资源存在冲突, 返回false, 位置不存在冲突, 返回true
     */
    public boolean checkResourceExclusiveConflict(List<PlanningEntry> entries);
}
```

```
public class checkResourceExclusiveConflict1 implements CheckResourceExclusiveConflict {
    ...
    @Override
    public boolean checkResourceExclusiveConflict(List<PlanningEntry> entries) {
        ...
        boolean flag = true;
        for(int i = 0 ; i < entries.size() ; i++) {
            ...
            for(int j = i + 1 ; j < entries.size() ; j++) {
                ...
                Calendar time1 = entries.get(i).getBeginEndTime().getdate1(); //第i个计划项的起始时间
                ...
                Calendar time2 = entries.get(i).getBeginEndTime().getdate2(); //第i个计划项的结束时间
                ...
                Calendar time3 = entries.get(j).getBeginEndTime().getdate1(); //第j个计划项的起始时间
                ...
                Calendar time4 = entries.get(j).getBeginEndTime().getdate2(); //第j个计划项的结束时间
                ...
                if((time1.compareTo(time3) < 0 && time2.compareTo(time3) > 0) || ((time1.compareTo(time3) > 0 && time1.compareTo(time4) < 0))) { //时间重叠
                    ...
                    for(int w = 0 ; w < entries.get(i).getResource().size() ; w++) {
                        ...
                        if(entries.get(i).getResource().get(k).equals(entries.get(j).getResource().get(w))) { //资源相同
                            ...
                            System.out.println(entries.get(i).getName() + "和" + entries.get(j).getName() + "存在资源(" + w + ")");
                            ...
                            entries.get(j).getResource().get(w).toString() + "冲突");
                            ...
                            flag = false;
                        }
                    }
                }
            }
        }
        return flag;
    }
}
```



```
public class checkResourceExclusiveConflict2 implements CheckResourceExclusiveConflict {
    ...
    @Override
    public boolean checkResourceExclusiveConflict(List<PlanningEntry> entries) {
        ...
        boolean flag = true;
        for(int i = 0 ; i < entries.size() ; i++) {
            ...
            for(int j = i + 1 ; j < entries.size() ; j++) {
                ...
                for(int k = 0 ; k < entries.get(i).getResource().size() ; k++) {
                    ...
                    for(int w = 0 ; w < entries.get(j).getResource().size() ; w++) {
                        ...
                        if(entries.get(i).getResource().get(k).equals(entries.get(j).getResource().get(w))) { //资源相同
                            ...
                            Calendar time1 = entries.get(i).getBeginEndTime().getdate1(); //第i个计划项的起始时间
                            ...
                            Calendar time2 = entries.get(i).getBeginEndTime().getdate2(); //第i个计划项的结束时间
                            ...
                            Calendar time3 = entries.get(j).getBeginEndTime().getdate1(); //第j个计划项的起始时间
                            ...
                            Calendar time4 = entries.get(j).getBeginEndTime().getdate2(); //第j个计划项的结束时间
                            ...
                            if((time1.compareTo(time3) < 0 && time2.compareTo(time3) > 0) || ((time1.compareTo(time3) > 0 && time1.compareTo(time4) < 0))) {
                                ...
                                System.out.println(entries.get(i).getName() + "和" + entries.get(j).getName() + "存在资源(" + w + ")");
                                ...
                                entries.get(j).getResource().get(w).toString() + "冲突");
                                ...
                                flag = false;
                            }
                        }
                    }
                }
            }
        }
        return flag;
    }
}
```



3.11 应用设计与开发

3.11.1 航班应用

1. FlightSchedule 类

设计思路：所有的方法均是针对 APP 来设计，方法的参数也是根据实际需求来设计。创建一个航班管理，可以读取文件创建，也可以自己手动输入信息进行创建。

航班管理：航班管理应用由一系列的航班组成，其中包含了一系列的可用飞机和可用机场，并且对应一个当前时间。

属性：

```
private List<Plane> resource = new ArrayList<Plane>(); //可用的资源  
private List<Location> location = new ArrayList<Location>(); //可用的位置  
private List<FlightEntry> flightEntry = new ArrayList<FlightEntry>(); //一系列计划项  
private Calendar time = Calendar.getInstance(); //当前的时间
```

主要实现的功能：

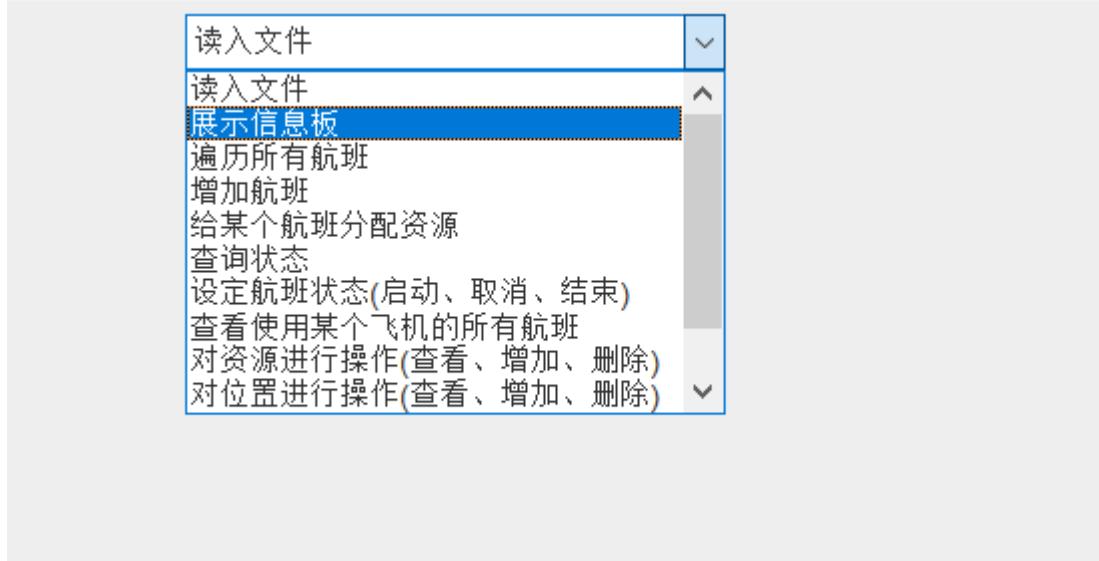
- 用户提供必要信息，管理（查看、增加、删除）可用的资源；
- 用户提供必要信息，管理（查看、增加、删除）可用的位置；
- 用户提供必要信息，增加一条新的计划项；
- 用户提供必要信息，取消某个计划项；
- 用户提供必要信息，为某个计划项分配资源；
- 用户从文件读取数据创建一个航班应用
- 用户提供必要信息，启动某个计划项；
- 用户提供必要信息，以重启动某个已挂起的计划项；
- 用户提供必要信息，结束某个计划项；
- 用户选定一个计划项，查看它的当前状态；
- 检测当前的计划项集合中可能存在的位置和资源独占冲突
- 针对用户选定的某个资源，列出使用该资源的所有计划项（包含尚未开的，始执行的、执行中的、已经结束的）。用户选中其中某个计划项之后，可以找出它的前序计划项
- 选定特定位置，可视化展示当前时刻该位置的信息板
- 展示和某个位置相关的所有计划项信息

方法：

| 修饰符和类型 | 方法 | 说明 |
|------------------|---|---------------------|
| boolean | addLocation(Location loc) | 增加可用的位置 |
| boolean | addPlanningEntry(Location loc1, Location loc2, java.util.Calendar time1, java.util.Calendar time2, java.lang.String name) | 增加可用的飞机 |
| boolean | addResource(java.lang.String planeNumber, java.lang.String machineNumber, int size, double age) | 启动某个高铁 |
| java.lang.String | BeginPlanningEntry(java.lang.String Name, java.util.Calendar time1, java.util.Calendar time2) | 展示某个位置的信息板 |
| void | board(Location loc, java.util.Calendar time1, int s) | 取消某个航班 |
| java.lang.String | cancelPlanningEntry(java.lang.String planeNumber, java.util.Calendar time1, java.util.Calendar time2) | 根据当前的时间来设置航班的状态 |
| void | changeState(FlightEntry flight) | 检测计划项是否存在资源冲突 |
| boolean | check() | 删除可用的位置 |
| boolean | deleteLocation(Location loc) | 删除可用的资源 |
| boolean | deleteResource(java.lang.String planeNumber, java.lang.String machineNumber, int size, double age) | 结束某个高铁 |
| java.lang.String | EndPlanningEntry(java.lang.String planeNumber, java.util.Calendar time1, java.util.Calendar time2) | 为某个计划项分配资源 |
| int | FeiFeiResource(java.lang.String planeNumber, java.util.Calendar time1, java.util.Calendar time2, java.lang.String name) | 从正则表达式解析计划项的信息 |
| FlightEntry | getFlightEntry(java.lang.String s) | 找出使用相同飞机的某个航班的前序计划项 |
| boolean | getPrePlanningEntry(java.util.Calendar time1, java.util.Calendar time2, java.lang.String Name, java.lang.String planNumber) | 查看使用某个飞机的所有计划项 |
| void | getResourcePlanningEntry(java.lang.String planeNumber, java.util.Calendar time1) | 查看某个航班的所有信息 |
| void | ReadFileCreatePlanningEntry(java.lang.String file) | 从文件读取数据建立一系列计划项 |
| void | setTime(java.util.Calendar time) | 设置当前时间 |
| void | show(Location loc, java.util.Calendar time1) | 列出和某个位置相关的所有航班 |
| void | showFlightEntry(Location loc, java.util.Calendar time) | 列出和某个位置相关的所有航班并可视化 |
| void | showLocation() | 列出可用的位置的信息 |
| void | showResource() | 列出可用的飞机的信息 |
| State | WatchState(java.lang.String name, java.util.Calendar time1, java.util.Calendar time2) | 查看某个航班的当前状态 |

2. TrainScheduleAPP

用 GUI 图形界面结合展示



我的思路： 1. 用户可以选择从文件中创建计划项集合，也可以手动输入创建计划项集合。

2. 增加一个计划项的时候，只需要分配位置和时间即可，不需要分配资源。而且分配了的位置会自动添加为可用位置。

3. 展示资源冲突和位置冲突的时候，提示信息显示在终端，并没有用 GUI 组件来提示信息，GUI 只是提示了有无冲突。

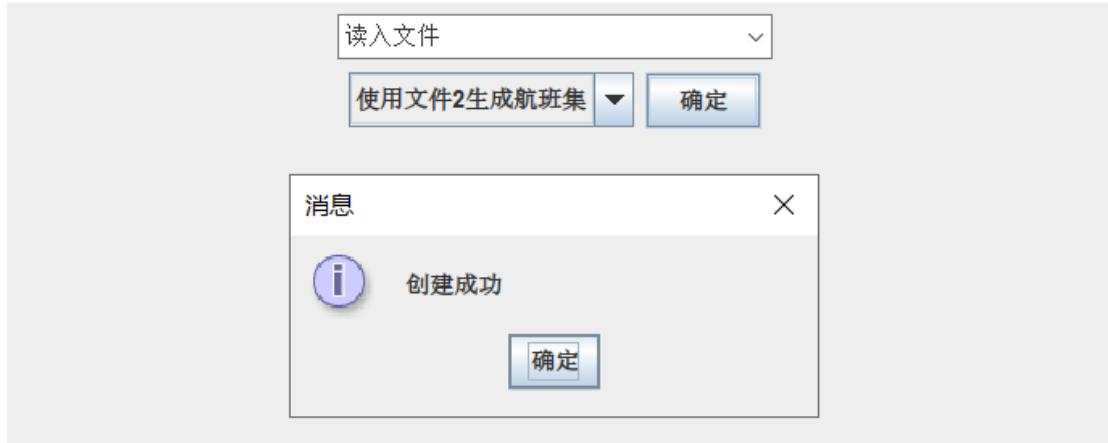
4. 对于和状态有关的功能，一般我都指明需要用户输入一个当前时间。这样能够更灵活的展示状态变化。

执行流程：

1. 增加可用的资源

2. 增加一个新的计划项(可以通过读取文件创建一系列计划项)
(也可以先 2 后 1, **但必须保证计划项分配资源之前得增加可用资源**)

3. 给新的计划项分配资源
其他的功能可以随便选择顺序。

1. 读取文件创建

| 所有可用的飞机的信息 | |
|------------|---|
| 序号 | 可用的飞机的信息如下 |
| 1 | Plane [PlaneNumber=B0741, MachineNumber=A380, Size=468, Age=1.1] |
| 2 | Plane [PlaneNumber=N5375, MachineNumber=B757, Size=261, Age=3.1] |
| 3 | Plane [PlaneNumber=B7408, MachineNumber=C829, Size=310, Age=27.5] |
| 4 | Plane [PlaneNumber=N6493, MachineNumber=C929, Size=310, Age=5.0] |
| 5 | Plane [PlaneNumber=B4007, MachineNumber=A330, Size=318, Age=2.1] |
| 6 | Plane [PlaneNumber=B9273, MachineNumber=A380, Size=468, Age=6.8] |
| 7 | Plane [PlaneNumber=N1992, MachineNumber=B747, Size=450, Age=15.8] |
| 8 | Plane [PlaneNumber=N6776, MachineNumber=A330, Size=318, Age=9.0] |
| 9 | Plane [PlaneNumber=N7932, MachineNumber=A330, Size=318, Age=3.2] |
| 10 | Plane [PlaneNumber=N8102, MachineNumber=A350, Size=286, Age=19.9] |
| 11 | Plane [PlaneNumber=N8003, MachineNumber=C929, Size=310, Age=14.2] |
| 12 | Plane [PlaneNumber=B4892, MachineNumber=B767, Size=289, Age=21.9] |

| 2020-01-03 11:30(当前时间), Shanghai机场 | | | | |
|------------------------------------|--------------------------------------|--------|-------------------|-----------|
| 序号 | 时间 | 航班 | 行程 | 状态 |
| 1 | 2020-01-01 12:59 -> 2020-01-01 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 2 | 2020-01-02 12:59 -> 2020-01-02 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 3 | 2020-01-03 12:59 -> 2020-01-03 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 4 | 2020-01-04 12:59 -> 2020-01-04 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 5 | 2020-01-05 12:59 -> 2020-01-05 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 6 | 2020-01-06 12:59 -> 2020-01-06 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 7 | 2020-01-07 12:59 -> 2020-01-07 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 8 | 2020-01-08 12:59 -> 2020-01-08 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 9 | 2020-01-09 01:59 -> 2020-01-09 05:15 | AA754 | Urumqi-Shanghai | ALLOCATED |
| 10 | 2020-01-09 12:59 -> 2020-01-09 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 11 | 2020-01-10 12:59 -> 2020-01-10 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 12 | 2020-01-11 12:59 -> 2020-01-11 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 13 | 2020-01-12 12:59 -> 2020-01-12 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 14 | 2020-01-13 12:59 -> 2020-01-13 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 15 | 2020-01-14 12:59 -> 2020-01-14 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 16 | 2020-01-15 12:59 -> 2020-01-15 14:19 | MF85 | Weihai-Shanghai | ALLOCATED |
| 17 | 2020-01-01 01:00 -> 2020-01-01 06:31 | ZH85 | Shanghai-Dalian | ENDED |
| 18 | 2020-01-01 08:11 -> 2020-01-01 11:16 | FM7079 | Shanghai-Nanning | ENDED |
| 19 | 2020-01-01 08:51 -> 2020-01-01 12:41 | GS4209 | Shanghai-Shenyang | ENDED |
| 20 | 2020-01-02 01:00 -> 2020-01-02 06:31 | ZH85 | Shanghai-Dalian | ENDED |

2. 手动增加一个计划项

The screenshot shows a flight scheduling application. At the top, there are input fields for flight start time (2020-05-06 11:30), end time (2020-05-06 13:30), departure location (Shanghai), arrival location (Nanchang), and flight name (aaa). A confirmation dialog box says "操作成功" (Operation successful) with a "确定" (Confirm) button. Below this, a message board displays the current time (2020-05-06 11:30) and location (Shanghai). A flight list table shows one entry:序号 (Index) 1, 时间 (Time) 11:30, 航班 (Flight) aaa, 行程 (Route) Shanghai-Nanchang, and 状态 (Status) WAITING.

3. 增加资源

The screenshot shows a resource addition interface. It includes fields for plane number (AJ62), machine number (SGFG), passenger capacity (300), and age (2.5). A confirmation dialog box says "操作成功" (Operation successful) with a "确定" (Confirm) button.

4. 分配资源

The screenshot shows a resource allocation interface. It includes fields for flight departure time (2020-05-06 11:30), arrival time (2020-05-06 13:30), flight number (aaa), and assigned plane (AJ62). A confirmation dialog box says "操作成功" (Operation successful) with a "确定" (Confirm) button. Below this, a message board displays the current time (2020-05-06 11:40) and location (Shanghai). A flight list table shows one entry:序号 (Index) 1, 时间 (Time) 11:30, 航班 (Flight) aaa, 行程 (Route) Shanghai-Nanchang, and 状态 (Status) RUNNING.

5. 查询状态

The screenshot shows a flight status query interface. It includes fields for departure time (2020-05-06 11:30), arrival time (2020-05-06 13:30), and flight number (aaa). A confirmation dialog box says "航班当前的状态为: RUNNING" (The current status of the flight is: RUNNING) with a "确定" (Confirm) button.

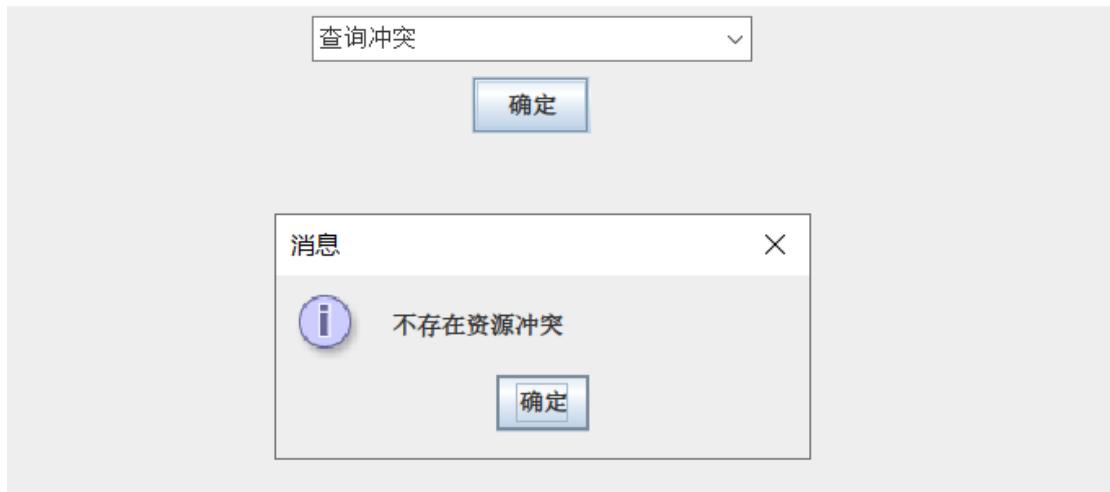
6. 查询使用某个飞机的航班

The screenshot shows a flight search by aircraft interface. It includes fields for the aircraft number (AJ62) and current time (2020-05-13 11:30). A message board displays the current time (2020-05-13 11:30) and location (使用飞机AJ62的所有航班). A flight list table shows one entry:序号 (Index) 1, 时间 (Time) 2020-05-06 11:30 -> 2020-05-06 13:30, 航班 (Flight) aaa, 行程 (Route) Shanghai-Nanchang, and 状态 (Status) ENDED.

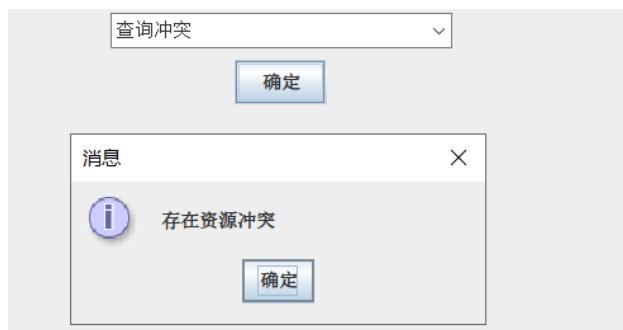
7. 查询所有可用的资源

The screenshot shows a query for all available resources interface. It includes fields for plane number (AJ62), machine number (SGFG), passenger capacity (300), and age (2.5). A message board displays the message "所有可用的飞机的信息如下" (Information about all available planes is as follows). A table shows one entry:序号 (Index) 1, 可用的飞机的信息如下 (Information about the available plane) Plane [PlaneNumber=AJ62, MachineNumber=SGFG, Size=300, Age=2.5].

8. 查询冲突



当读入文件2进行创建航班集合时：



```
Coverage Console
FlightScheduleApp (1) [Java Application] D:\jdk\bin\javaw.exe (2020年5月13日 下午4:53:27)
AA0644和NX749 存在资源:(Plane [PlaneNumber=B7408, MachineNumber=C929, Size=310, Age=27.5])冲突
AA0644和MF139 存在资源:(Plane [PlaneNumber=B9273, MachineNumber=A380, Size=468, Age=6.8])冲突
AA0644和MF139 存在资源:(Plane [PlaneNumber=N6776, MachineNumber=A330, Size=318, Age=9.0])冲突
CA04和HU947 存在资源:(Plane [PlaneNumber=N6776, MachineNumber=A330, Size=318, Age=9.0])冲突
CA04和MU059 存在资源:(Plane [PlaneNumber=N6776, MachineNumber=A330, Size=318, Age=9.0])冲突
CX1909和UA1082 存在资源:(Plane [PlaneNumber=N3522, MachineNumber=A340, Size=332, Age=15.4])冲突
CX1909和CZ29 存在资源:(Plane [PlaneNumber=B3380, MachineNumber=B737, Size=145, Age=20.1])冲突
CX1909和ZH85 存在资源:(Plane [PlaneNumber=B3380, MachineNumber=B737, Size=145, Age=20.1])冲突
CX1909和FM7079 存在资源:(Plane [PlaneNumber=B7408, MachineNumber=C929, Size=310, Age=27.5])冲突
CX1909和MU6171 存在资源:(Plane [PlaneNumber=B2230, MachineNumber=A380, Size=468, Age=3.9])冲突
CX1909和MF139 存在资源:(Plane [PlaneNumber=N7932, MachineNumber=A330, Size=318, Age=3.2])冲突
CZ29和HU947 存在资源:(Plane [PlaneNumber=B6802, MachineNumber=B767, Size=289, Age=14.0])冲突
CZ29和ZH85 存在资源:(Plane [PlaneNumber=B3380, MachineNumber=B737, Size=145, Age=20.1])冲突
CZ29和UA1082 存在资源:(Plane [PlaneNumber=N5375, MachineNumber=B757, Size=261, Age=3.1])冲突
CZ6145和NX49 存在资源:(Plane [PlaneNumber=N1543, MachineNumber=C929, Size=310, Age=22.6])冲突
GS28和GS3664 存在资源:(Plane [PlaneNumber=N3522, MachineNumber=A340, Size=332, Age=15.4])冲突
HU947和MU059 存在资源:(Plane [PlaneNumber=N6776, MachineNumber=A330, Size=318, Age=9.0])冲突
LH0263和SC01 存在资源:(Plane [PlaneNumber=N1992, MachineNumber=B747, Size=450, Age=15.8])冲突
MU059和UA1082 存在资源:(Plane [PlaneNumber=N3522, MachineNumber=A340, Size=332, Age=15.4])冲突
NX49和UA985 存在资源:(Plane [PlaneNumber=N3616, MachineNumber=A340, Size=332, Age=1.3])冲突
```

冲突信息显示在终端上

其他的功能不再一一阐述

3.11.2 高铁应用

1. TrainSchedule 类

设计思路：所有的方法均是针对 APP 来设计，方法的参数也是根据实际需求来设计。和其他的两个类的功能及方法特别类似。

高铁管理：高铁管理应用由一系列的高铁车次组成，其中包含了一系列的可用车厢和可用站，并且对应一个当前时间。

属性：

```
private List<Teacher> resource = new ArrayList<Teacher>(); //可用的资源  
private List<Location> location = new ArrayList<Location>(); //可用的位置  
private List<CourseEntry> courseEntry = new ArrayList<CourseEntry>(); //一系列计划项  
private Calendar time = Calendar.getInstance(); //当前时间
```

主要实现的功能：

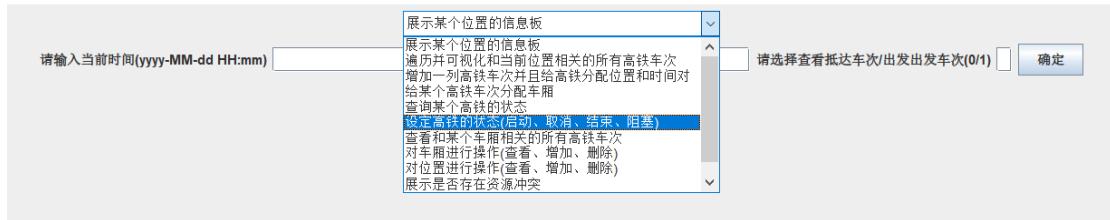
- 用户提供必要信息，管理（查看、增加、删除）可用的资源；
- 用户提供必要信息，管理（查看、增加、删除）可用的位置；
- 用户提供必要信息，增加一条新的计划项；
- 用户提供必要信息，取消某个计划项；
- 用户提供必要信息，为某个计划项分配资源；
- 用户提供必要信息，阻塞某个计划项；
- 用户提供必要信息，启动某个计划项；
- 用户提供必要信息，以重启动某个已挂起的计划项；
- 用户提供必要信息，结束某个计划项；
- 用户选定一个计划项，查看它的当前状态；
- 检测当前的计划项集合中可能存在的位置和资源独占冲突
- 针对用户选定的某个资源，列出使用该资源的所有计划项（包含尚未开的，始执行的、执行中的、已经结束的）。用户选中其中某个计划项之后，可以找出它的前序计划项
- 选定特定位置，可视化展示当前时刻该位置的信息板
- 展示和某个位置相关的所有计划项信息

方法：

| | |
|---|--------------------------|
| addLocation(Location loc) | 增加可用的位置 |
| addPlanningEntry(java.util.List<Location> loc, java.util.List<Timeslot> timeslot, java.lang.String name) | 增加一个高铁车次 |
| addResource(java.lang.String railNumber, java.lang.String type, int size, int year) | 增加可使用的车厢 |
| BeginPlanningEntry(java.lang.String trainNumber, java.util.Calendar time) | 启动某个高铁车次 |
| BlockPlanningEntry(java.lang.String trainNumber, java.util.Calendar time) | 阻塞某个高铁车次 |
| boardLocation(loc, java.util.Calendar time, int x) | 展示某个位置的信息板 |
| cancelPlanningEntry(java.lang.String trainNumber, java.util.Calendar time) | 取消某个高铁车次 |
| changeState(TrainEntry train) | 根据当前的时间来设置高铁的状态 |
| check() | 检测计划项是否存在资源冲突 |
| deleteLocation(Location loc) | 删除可用的位置 |
| deleteResource(java.lang.String railNumber, java.lang.String type, int size, int year) | 删除可使用的车厢 |
| EndPlanningEntry(java.lang.String trainNumber, java.util.Calendar time) | 结束某个高铁车次 |
| FeiFeiResource(java.lang.String name, java.util.Calendar time, java.util.List<Railway> railway) | 为某个计划项分配资源 |
| getPrePlanningEntry(java.lang.String railNumber, java.lang.String type, int size, int year, java.lang.String name, java.util.Calendar time) | 可视化某个和某个高铁使用相同车厢的前序高铁的信息 |
| getResourcePlanningEntry(java.lang.String railNumber, java.lang.String type, int size, int year, java.util.Calendar TIME) | 展示使用某个资源的所有高铁车次 |
| main(java.lang.String[] main) | 设置当前的时间 |
| setTime(java.util.Calendar time) | 列出和某个位置相关的所有高铁的信息板 |
| show(Location loc, java.util.Calendar time) | 列出可用的位置的信息 |
| showLocation() | 列出可用的车厢的信息 |
| showResource() | 查询航班的状态 |
| WatchState(java.lang.String name, java.util.Calendar time1, java.util.Calendar TIME) | |

2. TrainScheduleAPP

用 GUI 图形界面结合展示



我的思路： 1. 增加一个计划项的时候，需要分配若干个时间对和位置对，我的设计是每次只能增加一对时间和一对位置。如果有多个位置和时间，需要添加多次。
2. 分配资源也是一样，每次只能分配一个车厢，如果需要分配多个车厢，则需要分配多次。

3. 展示资源冲突和位置冲突的时候，提示信息显示在终端，并没有用 GUI 提示信息，GUI 只是提示了有无冲突。

4. 对于和状态有关的功能，一般我都指明需要用户输入一个当前时间。这样能更灵活的展示状态变化。

提示： 由于对 GUI 设计不是很熟练，界面做的比较难看。每次查看信息后如果点击关闭表格窗口，会导致 GUI 退出。所以建议老师每次如果输入命令的时候不要把表格的窗口关闭。

执行流程： 1. 增加可用的资源

2. 增加一个新的计划项

(也可以先 2 后 1，但必须保证计划项分配资源之前得增加可用资源)

3. 给新的计划项分配资源

其他的功能可以随便选择顺序。

1. 增加可用的资源

The screenshot shows a software interface for managing train carriages. At the top, there are input fields for carriage number (aaa), type (商务/Business), capacity (10), and year (1986). Below the input fields are two message boxes: "操作成功" (Operation successful) with a "确定" (Confirm) button. At the bottom is a table titled "所有可用的车厢的信息" (Information of all available carriages) with two rows:

| 序号 | 可用的车厢的信息如下 |
|----|---|
| 1 | Railway [Number=aaa, type=BUSINESS, Size=10, Year=1986] |
| 2 | Railway [Number=bbb, type=BUSINESS, Size=10, Year=1986] |

2. 增加一个新的计划项

增加一个新的计划项的时候，需要分配若干个时间对和位置对，我的设计是每次只能增加一对时间和一对位置。如果有多个位置和时间，需要添加多次。

The screenshot shows a software interface for adding a high-speed rail route. At the top, there are input fields for train number (A32H), departure location (Nanchang), arrival location (Shanghai), departure time (2020-05-06 11:30), and arrival time (2020-05-06 15:30). Below the input fields is a message box: "操作成功" (Operation successful) with a "确定" (Confirm) button.

必须保证第一次输入的终止位置和第二次输入的起始位置是一样的

The screenshot shows a software interface for adding a high-speed rail route. It consists of three parts: 1. Input fields for train number (A32H), departure location (Shanghai), arrival location (Beijing), departure time (2020-05-06 16:30), and arrival time (2020-05-07 15:30). 2. A message box: "操作成功" (Operation successful) with a "确定" (Confirm) button. 3. Another message box: "操作成功" (Operation successful) with a "确定" (Confirm) button.

3. 给计划项分配资源

分配资源也是一样，每次只能分配一个车厢，如果需要分配多个车厢，则需要分

配多次。每次分配资源的时候都会展示出当前的可用资源

The first screenshot shows a dialog box with the message "分配成功" (Allocation successful) over a form for allocating carriages to train A32H. The second screenshot shows a similar dialog box for train A32H from Shanghai to Harbin.

4. 展示信息板

The first screenshot displays a table for Nanchang with one row:序号 (Index) 1, 时间 (Time) 11:30, 高铁车次 (Train Number) A32H, 行程 (Journey) Nanchang-Harbin, and 状态 (Status) RUNNING. The second screenshot displays a table for Shanghai with one row:序号 (Index) 1, 时间 (Time) 15:30, 高铁车次 (Train Number) A32H, 行程 (Journey) Nanchang-Harbin, and 状态 (Status) RUNNING.

5. 遍历和当前位置相关的所有计划项

The screenshot shows a table for Shanghai with one row:序号 (Index) 1, 时间 (Time) 2020-05-06 11:30->2020-05-06 11:30, 高铁车次 (Train Number) A32H, 行程 (Journey) Nanchang - Haerbin, and 状态 (Status) BLOCKED.

6. 查询所有的车厢

The screenshot shows a table with two rows:序号 (Index) 1 and 2. The table includes a column titled "可用的车厢的信息如下" (Information about available carriages) which lists two carriages: Railway [Number=aaa, type=BUSINESS, Size=10, Year=1111] and Railway [Number=bbb, type=BUSINESS, Size=10, Year=1986].

7. 查询是否存在资源冲突



8. 查看使用某个车厢的全部计划项

| 序号 | 时间 | 高铁车次 | 行程 | 状态 |
|----|------------------------------------|------|--------------------|---------|
| 1 | 2020-05-06 11:30->2020-05-06 11:30 | A32H | Nanchang - Haerbin | RUNNING |

还有很多功能不再一一阐述，

3.11.3 课程应用

1. CourseCalendar 类

设计思路：CourseCalendar 类中所有的方法均是针对 APP 来设计的，参数是根据实际需要来设定的。比如增加一个计划项：需要提供计划项基本的信息：计划项的名字，计划项的位置，计划项的起止时间。启动一个计划项：也需要计划项的基本信息：计划项的名字，计划项的位置，计划项的起止时间。具体的方法实现就是基本的调用前面的已经实现的方法。（详细见代码）

课表管理：课表应用由一系列的课程组成，其中包含了一系列的可用教师和可用教室，并且对应一个当前时间。

属性：

```
private List<Teacher> resource = new ArrayList<Teacher>(); //可用的资源
private List<Location> location = new ArrayList<Location>(); //可用的位置
private List<CourseEntry> courseEntry = new ArrayList<CourseEntry>(); //一系列计划项
private Calendar time = Calendar.getInstance(); //当前时间
```

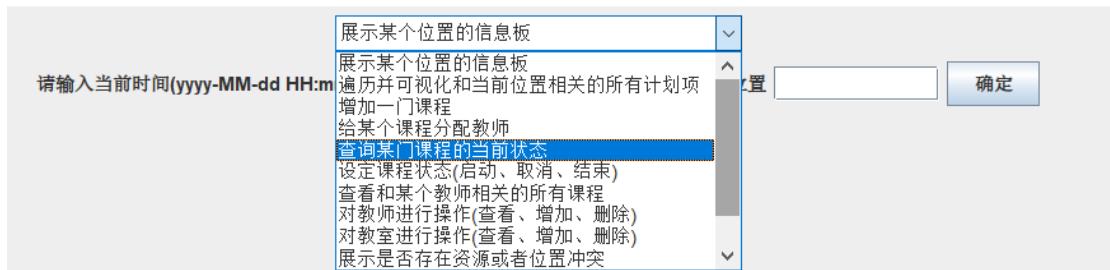
主要实现的功能：

- 用户提供必要信息，管理（查看、增加、删除）可用的资源；
- 用户提供必要信息，管理（查看、增加、删除）可用的位置；
- 用户提供必要信息，增加一条新的计划项；
- 用户提供必要信息，取消某个计划项；
- 用户提供必要信息，为某个计划项分配资源；
- 用户提供必要信息，启动某个计划项；
- 用户提供必要信息，以变更某个已存在的计划项的位置；
- 用户提供必要信息，以重启某个已挂起的计划项；
- 用户提供必要信息，结束某个计划项；
- 用户选定一个计划项，查看它的当前状态；
- 检测当前的计划项集合中可能存在的位置和资源独占冲突
- 针对用户选定的某个资源，列出使用该资源的所有计划项（包含尚未开的，始执行的、执行中的、已经结束的）。用户选中其中某个计划项之后，可以找出它的前序计划项
- 选定特定位置，可视化展示当前时刻该位置的信息板
- 展示和某个位置相关的所有计划项信息

| 修饰符和类型 | 方法 | 说明 |
|---------------------------------|---|---------------------------------|
| boolean | addLocation(Location loc) | 增加可用的位置 |
| boolean | addPlanningEntry(java.lang.String name, Location loc, java.util.Calendar time1, java.util.Calendar time2) | 增加一条新的计划项 |
| boolean | addResource(java.lang.String name, java.lang.String Number, boolean s, java.lang.String title) | 添加某个可用的教师 |
| java.lang.String | BeginPlanningEntry(java.lang.String Name, java.util.Calendar time1, java.util.Calendar time2, Location loc) | 启动某门课程 |
| void | board(Location loc, java.util.Calendar time1) | 展示当前位置当前时间的信息板(某个教师的当天的课程信息) |
| java.lang.String | cancelPlanningEntry(java.lang.String Name, java.util.Calendar time1, java.util.Calendar time2, Location loc) | 取消某门课程 |
| boolean | changeLocationEntry(java.lang.String Name, Location loc1, Location loc2) | 改变某门课程的位置 |
| void | changeState(CourseEntry course) | 改变某门课程的状态 |
| boolean | check() | 检测计划项是否存在资源冲突和位置冲突 |
| boolean | deleteLocation(Location loc) | 删除可用的位置 |
| boolean | deleteResource(java.lang.String name, java.lang.String Number, boolean s, java.lang.String title) | 删除某个可用的教师 |
| java.lang.String | EndPlanningEntry(java.lang.String Name, java.util.Calendar time1, java.util.Calendar time2, Location loc) | 结束某门课程 |
| int | FeePeiResource(java.lang.String Name, java.lang.String idNumber, java.lang.String name, boolean sex, java.lang.String title) | 给某门课程分配教师 |
| boolean | getPrePlanningEntry(java.lang.String Name, Location loc, java.lang.String name, java.lang.String idNumber) | 得到某个和某个课程使用相同教师的前序课程 |
| void | getResourcePlanningEntry(java.lang.String name, java.lang.String Number, boolean s, java.lang.String title, java.util.Calendar time1) | 得到使用和某个计划项使用相同的某个资源的时间最接近的前序计划项 |
| void | setTime(java.util.Calendar time) | 设置当前时间 |
| void | show(Location loc, java.util.Calendar time1) | 可视化展示和某个教室的所有课程信息 |
| void | showFlightEntry(Location loc, java.util.Calendar time1) | 利用迭代器遍历所有和当前位置当前这一天相关的课程 |
| void | showLocation() | 可视化列出可用的位置的信息 |
| void | showResource() | 可视化列出可用的资源的信息 |
| void | WatchState(java.lang.String name, java.util.Calendar time1) | 查看某个计划项的状态 |
| 从类继承的方法 java.lang.Object | | |

2.CourseCalendarAPP

用 GUI 图形界面展示出来



我的思路： 1.增加一个计划项的时候，只需要分配位置和时间即可，不需要分配资源。

而且分配了的位置会自动添加为可用位置。

2.展示资源冲突和位置冲突的时候，提示信息显示在终端，并没有用 GUI 提示信息，GUI 只是提示了有无冲突。

3.对于和状态有关的功能，一般我都指明需要用户输入一个当前时间。这样能够更灵活的展示状态变化。

运行流程： 1.先增加可用的资源

2.增加一个新的计划项

(也可以先 2 后 1，但必须保证计划项分配资源之前得增加可用资源)

3.给新的计划项分配资源

其他的功能可以随便选择顺序。

提示： 由于对 GUI 设计不是很熟练，界面做的比较难看。每次查看信息后如果点击关闭表格窗口，会导致 GUI 退出。所以建议老师每次如果输入命令的时候不要把表格的窗口关闭。

演示：

1. 增加可用的资源



然后查看可用的教师之后：

| 所有可用的教师的信息 | |
|------------|--|
| 序号 | 所有的资源 |
| 1 | Teacher [IdNumber=84651311561, Name=徐汉川, Sex=true, Title=教授] |

2.增加一个新的计划项



然后查看计划项

A screenshot of a software application window titled "查询某门课程的当前状态". It has input fields for "请输入课程名称" (Software Construction), "请输入当前时间(yyyy-MM-dd HH:mm)" (2020-05-13 14:00), and a "确定" (Confirm) button. A modal dialog box titled "消息" displays the message "操作成功" (Operation successful) with a "确定" (Confirm) button. Below the input fields is a table:

| 序号 | 时间 | 课程 | 状态 |
|----|-----------------------------------|------|----------|
| 1 | 2020-05-13 10:00-2020-05-13 11:45 | 软件构造 | WAITTING |

A red arrow points to the "WAITTING" status column. Below the table, a red message says "当前课程还未分配资源，所以为WAITTING".

3.给计划项分配资源

每次给计划项分配资源的时候，都会显示所有的可用资源的信息

A screenshot of a software application window titled "所有可用的教师的信息". It shows a table with one row:

| 序号 | 所有的资源 |
|----|--|
| 1 | Teacher [IdNumber=84651311561, Name=徐汉川, Sex=true, Title=教授] |

Below this is another window titled "给某个课程分配教师". It contains input fields for "请输入待分配教师的课程名称" (Software Construction), "请输入教师身份证号" (84651311561), "请输入教师姓名" (徐汉川), "输入教师性别(true男/false女)" (true), "输入教师职称" (教授), and a "确定" (Confirm) button. A modal dialog box titled "消息" displays the message "操作成功" (Operation successful) with a "确定" (Confirm) button.

然后再次查询课程信息

A screenshot of a software application window titled "2020-05-13 14:00(当前时间)". It shows a table with one row:

| 序号 | 时间 | 课程 | 状态 |
|----|-----------------------------------|------|-------|
| 1 | 2020-05-13 10:00-2020-05-13 11:45 | 软件构造 | ENDED |

4.展示信息板

A screenshot of a software application window titled "展示某个位置的信息板". It contains input fields for "请输入当前时间(yyyy-MM-dd HH:mm)" (2020-05-13 15:08), "请输入当前位置" (线上), and a "确定" (Confirm) button. Below these is a table:

| 序号 | 时间 | 课程 | 教师 | 状态 |
|----|-------------|------|-----|-------|
| 1 | 08:00-10:00 | 算法设计 | 张伟 | ENDED |
| 2 | 10:00-11:45 | 软件构造 | 徐汉川 | ENDED |

5.查询冲突



6.改变某门课程的教室

改变某门课程的教室

请输入课程名称 软件构造 请输入课程的初始教室 线上 请输入课程更改后的教室 线下 确定

展示某个位置的信息板

请输入当前时间(yyyy-MM-dd HH:mm) 2020-05-13 15:08 请输入当前位置 线下 确定

| 序号 | 时间 | 课程 | 教师 | 状态 |
|----|-------------|------|-----|-------|
| 1 | 10:00-11:45 | 软件构造 | 徐汉川 | ENDED |

7.查看所有可用的教师

对教师进行操作(查看、增加、删除)

查看所有可用的教师 请输入教师名字 张伟 请输入教师身份证号 111111111111 输入教师性别(true男)false女 true 请输入教师职称 教授 确定

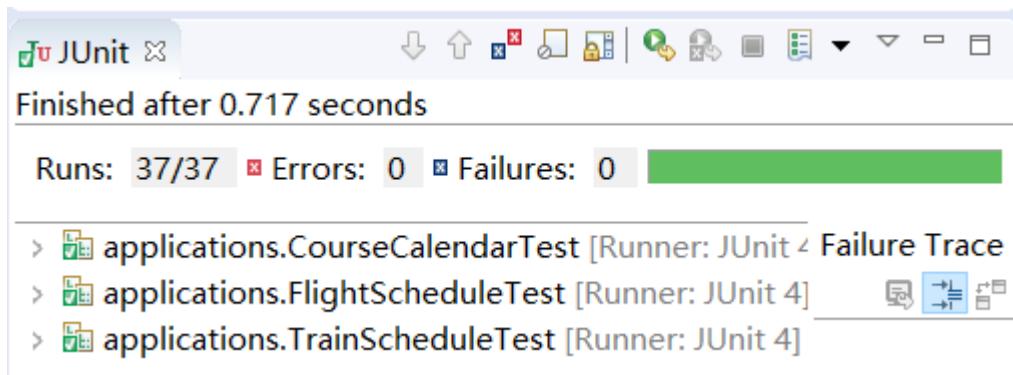
所有可用的教师的信息

| 序号 | 所有的资源 |
|----|--|
| 1 | Teacher [IdNumber=84651311561, Name=徐汉川, Sex=true, Title=教授] |
| 2 | Teacher [IdNumber=1111111111, Name=张伟, Sex=true, Title=教授] |

其他的功能就不一一阐述。

测试：

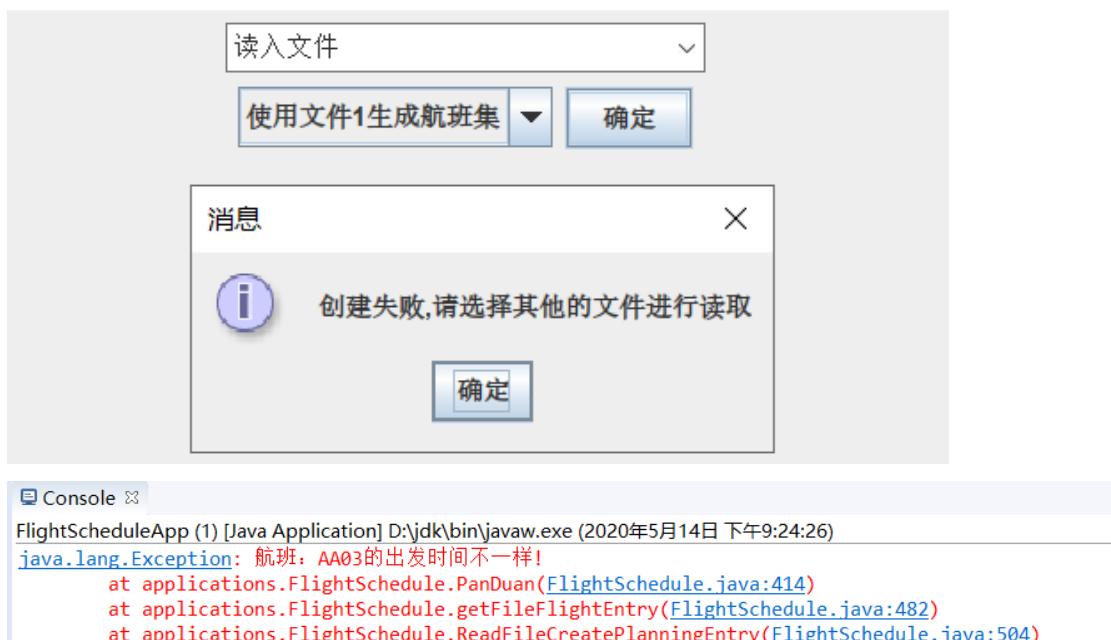
| | | | | |
|-----------------------|--------|-------|-------|-------|
| applications | 52.2 % | 2,011 | 1,845 | 3,856 |
| > FlightSchedule.java | 48.7 % | 634 | 669 | 1,303 |
| > TrainSchedule.java | 55.6 % | 765 | 611 | 1,376 |
| > CourseCalendar.java | 52.0 % | 612 | 565 | 1,177 |

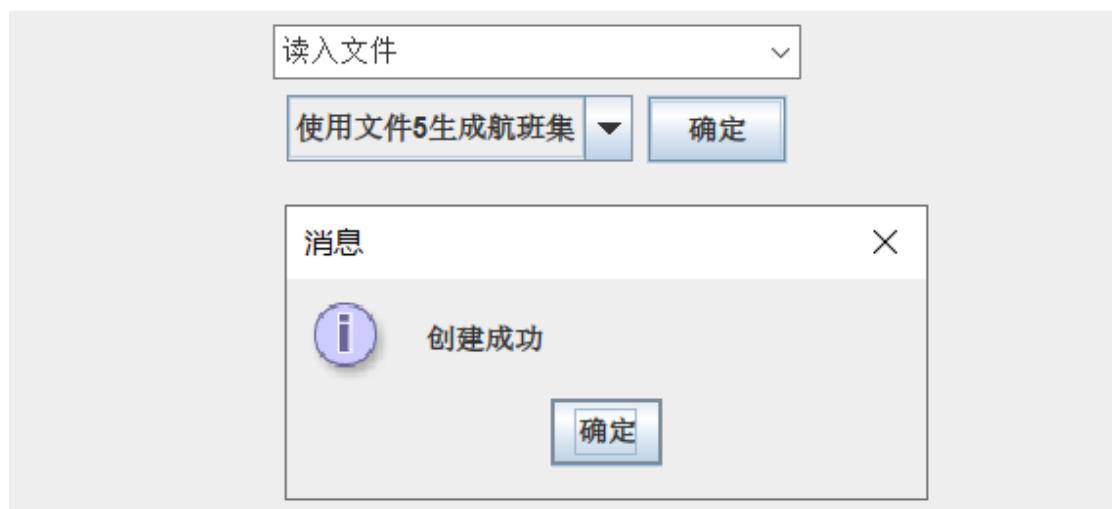
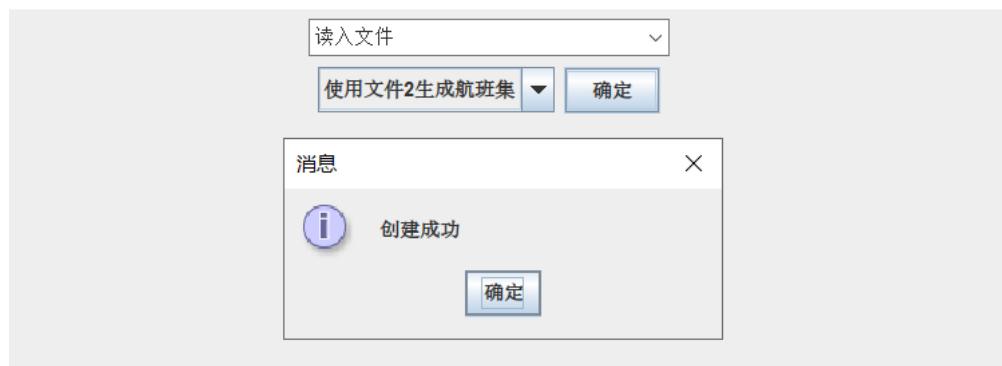


3.12 基于语法的数据读入

从文件中读取数据创建航班计划项分为两个过程，由于航班数据每 13 行代表一个计划项，因此每次读取 13 行就创建一个计划项。所以第一个过程是将 13 行数据读取到一个字符串中，第二个过程将这个字符串进行解析，提取创建航班所需要的数据创建一个新的航班。

提示：一旦我设计的正则表达式和文件中的数据匹配不到，立即抛出异常，提示错误信息。(经运行，文件 1 存在和实验指导书指明的语法错误)





1.从文件中提取 13 行数据到一个字符串 s 中，然后调用 getFileFlightEntry 函数创建一个新的计划项。(跳过空行)

```


    /**
     * 从文件读取数据建立一系列计划项
     * @param file 读取的文件
     * @throws Exception 文件读取错误
     */
    public void ReadFileCreatePlanningEntry(String file) throws Exception {
        BufferedReader reader = new BufferedReader(new FileReader(new File(file)));
        String line1 = "";
        int l = 0;
        StringBuffer s = new StringBuffer("");
        while((line1 = reader.readLine()) != null) {
            if(line1.equals("")) {
                continue; //跳过空行
            }
            s.append(line1 + "\n");
            l++;
            if(l % 13 == 0) { //已经读入完整的一个航班的信息
                this.flightEntry.add(getFileFlightEntry(s.toString()));
                s = new StringBuffer(""); //重新读入13行的信息
            }
        }
        reader.close(); //关闭文件
    }
}


```

2.利用字符串来提取构建航班需要的信息，匹配的正则表达式为：

```


Pattern pattern1 = Pattern.compile("Flight:([\d]{4}-[\d]{2}-"
    "[\d]{2}),([A-Z]{2}[\d]{2,4})\n\\{\nDepartureAirport:(.*?)\nArrivalAir-
    port:(.*?)\nDepartureTime:([0-9]{4}-[0-9]{2}-[0-9]{2}\s[0-9]{2}:[0-
    9]{2})\nArrivalTime:([0-9]{4}-[0-9]{2}-[0-9]{2}\s[0-9]{2}:[0-
    9]{2})\nPlane:([B || N][0-9]{4})\n\\{\nType:([0-9a-zA-
    Z]{4})\nSeats:([1-9][\d]{1,2})\nAge:(.*?)\n\\}\n\\}\n");


```

`Flight:2020-01-01,CA1001 //航班计划项 1`

//逗号前为航班日期，遵循 yyyy-MM-dd 格式

//逗号后为航班号，由两位大写字母和 2-4 位数字构成

1.使用的匹配航班计划项的时间和航班号的正则表达式为：

`"Flight:([\d]{4}-[\d]{2}-[\d]{2}),([A-Z]{2}[\d]{2,4})\n"`

| | |
|--------------------------------------|--------------|
| <code>DepartureAirport:Harbin</code> | //出发机场, word |
| <code>ArrivalAirport:Beijing</code> | //抵达机场, word |

2.匹配机场位置：机场位置是一个字符串，没有指明元素类型，

因此正则表达式为：

```
"DepartureAirport:(.*?)\nArrivalAir port:(.*?)\n"
```

```
DepartureTime:2020-01-01 12:00      //起飞时间
ArrivalTime:2020-01-01 14:00    //降落时间
                                         //这两个时间均为 yyyy-MM-dd HH:mm 的格式
                                         //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，也可以晚 1 天
                                         //例如 2020-01-01 23:00 起飞， 2020-01-02 01:00 降落
```

3. 匹配起止时间格式的正则表达式为：

```
"DepartureTime:([0-9]{4}-[0-9]{2}-[0-9]{2})\\s[0-9]{2}:[0-
9]{2})\nArrivalTime:([0-9]{4}-[0-9]{2}-[0-9]{2})\\s[0-9]{2}:[0-
9]{2})\n"
```

注意：判断起飞时间的日期和第一行的日期是否一致：

```
» »   SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy-MM-dd"); //判断是否是同一个日期
» »   if(!(sdf1.format(time1.getTime()).equals(planningEntryTime))){  

» »     throw new Exception("航班：" + planningEntryNumber + "的时间和出发时间不是同一个日期");  

» » }
```

```
Plane:B9802 //飞机编号，第一位为 N 或 B，后面是四位数字
{
  Type:A350 //机型，大小写字母或数字构成，不含有空格和其他符号
  Seats:300 //座位数，正整数，范围为 [50,600]
  Age:2.5 //机龄，数字，范围是 [0,30]，可最多 1 位小数或无小数
             //诸如 0、0.0、2、2.0、20.0 这样的表述都是对的
             //但 02.0、2.12 这样的表述则不符合规则
}
```

4. 匹配飞机编号的正则表达式为：

```
"Plane:([B || N][0-9]{4})\n"
```

5. 匹配机型的正则表达式为

```
"Type:([0-9a-zA-Z]{4})\n"
```

6. 匹配座位数的正则表达式为：

```
Seats:([1-9][\\d]{1,2})\nAge:(.*?)\n
```

注意：判断座位数是否符合要求：

```

    >>> if(Integer.valueOf(matcher.group(9)) < 50) {  

    >>>   throw new Exception("文件非法：飞机承载人数少于50");  

    >>> }  

    >>> if(Integer.valueOf(matcher.group(9)) > 600) {  

    >>>   throw new Exception("文件非法：飞机承载人数大于600");  

    >>> }
  
```

匹配机龄的正则表达式为

```
"Age:(.*?)\n
```

注意：判断机龄是否符合要求：

```

    >>> if(Double.valueOf(matcher.group(10)) > 30) {  

    >>>   throw new Exception("文件非法：飞机机龄大于30");  

    >>> }  

    >>> if(Double.valueOf(matcher.group(10)) < 0) {  

    >>>   throw new Exception("文件非法：飞机机龄小于0");  

    >>> }
  
```

利用正则表达式去和提取出来的字符串进行匹配，若匹配失败，直接抛出异常并提示信息

```

    >>> if(!matcher.find()) {  

    >>>   throw new Exception("文件非法！");  

    >>> }
  
```

否则，则提取字符串中的信息来创建新的计划项。

```

    >>> String planningEntryTime = matcher.group(1); //航班时间  

    >>> String planningEntryNumber = matcher.group(2); //航班号  

    >>> String departureAirport = matcher.group(3); //出发机场  

    >>> String arrivalAirport = matcher.group(4); //抵达机场  

    >>> String departureTime = matcher.group(5); //出发时间  

    >>> String arrivalTime = matcher.group(6); //抵达时间  

    >>> String number = matcher.group(7);  

    >>> String strType = matcher.group(8);  

    >>> FlightEntry t = new FlightEntry(planningEntryNumber); //航班  

    >>> Location loc1 = new Location(departureAirport);  

    >>> Location loc2 = new Location(arrivalAirport);  

    >>> if(!location.contains(loc1)) {  

    >>>   this.location.add(loc1); //增加可用的位置  

    >>> if(!location.contains(loc2)) {  

    >>>   this.location.add(loc2); //增加的可用位置  

    >>> t.setLocations(loc1, loc2); //航班的初始位置和终止位置  

    >>> SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm");  

    >>> Calendar time1 = Calendar.getInstance();  

    >>> time1.setTime(sdf.parse(departureTime)); //航班起飞时间  

    >>> Calendar time2 = Calendar.getInstance();  

    >>> time2.setTime(sdf.parse(arrivalTime)); //航班到达时间  

    >>> SimpleDateFormat sdf1 = new SimpleDateFormat("yyyy-MM-dd"); //判断是否是同一个日期  

    >>> if(!(sdf1.format(time1.getTime()).equals(planningEntryTime))) {  

    >>>   throw new Exception("航班：" + planningEntryNumber + "的时间和出发时间不是同一个日期");  

    >>> }
    >>> t.setTime(new Timeslot(time1, time2)); //设置航班的时间对
  
```

一直循环这两个步骤就可以将文件中的所有信息提取出来

判断航班的时间是否符合指导书要求

Flight: 2020-01-01, CA1002 // 航班计划项 2

// 在一个文件中不允许出现日期一样且航班号一样的两个计划项
 // 但允许出现“日期不同、航班号相同”的情况
 // 也允许出现“日期相同、航班号不同”的情况
 // 同一个航班号，虽然日期可以不同，但其出发和到达机场、
 // 出发和到达时间均应相同
 // 注意：CA0001、CA001、CA01 是相同的航班号

```

@>>> /**
@>>> * 判断相同的航班，起止时间是否相同以及航班日期和起飞时间日期是否相同
@>>> * @param flight ->
@>>> * @throws Exception ->
@>>> */
@>>> public void PanDuan(FlightEntry flight) throws Exception {
@>>>     String s = flight.getName().substring(0, 2); // 航班号前两个字符
@>>>     int x = Integer.valueOf(flight.getName().substring(2)); // 后面四个数字
@>>>     SimpleDateFormat sdf = new SimpleDateFormat("HH:mm"); // 时间形式的字符串
@>>>     String s1 = sdf.format(flight.getTime1().getTime()); // 出发时间
@>>>     String s2 = sdf.format(flight.getTime2().getTime()); // 终止时间
@>>>     for(int i = 0 ; i < flightEntry.size() ; i++) {
@>>>         String ss = flightEntry.get(i).getName().substring(0, 2); // 航班号前两个字符
@>>>         int xx = Integer.valueOf(flightEntry.get(i).getName().substring(2)); // 后面四个数字
@>>>         if(ss.equals(s) && xx == x) { // 航班号相同
@>>>             String s11 = sdf.format(flightEntry.get(i).getTime1().getTime()); // 出发时间
@>>>             String s22 = sdf.format(flightEntry.get(i).getTime2().getTime()); // 终止时间
@>>>             if(!s11.equals(s1)) {
@>>>                 throw new Exception("航班：" + flight.getName() + "的出发时间不一样!");
@>>>             }
@>>>             if(!s22.equals(s2)) {
@>>>                 throw new Exception("航班：" + flight.getName() + "的到达时间不一样!");
@>>>             }
@>>>             if(!flight.getStartLocation().equals(flightEntry.get(i).getStartLocation())) {
@>>>                 throw new Exception("航班：" + flight.getName() + "的出发机场时间不一样!");
@>>>             }
@>>>             if(!flight.getEndLocation().equals(flightEntry.get(i).getEndLocation())) {
@>>>                 throw new Exception("航班：" + flight.getName() + "的到达机场时间不一样!");
@>>>             }
@>>>         }
@>>>     }
@>>> }

```

2020-02-03 11:40(当前时间), Shanghai机场

| 序号 | 时间 | 航班 | 行程 | 状态 |
|----|--------------------------------------|--------|-------------------|-------|
| 1 | 2020-01-01 12:59 -> 2020-01-01 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 2 | 2020-01-02 12:59 -> 2020-01-02 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 3 | 2020-01-03 12:59 -> 2020-01-03 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 4 | 2020-01-04 12:59 -> 2020-01-04 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 5 | 2020-01-05 12:59 -> 2020-01-05 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 6 | 2020-01-06 12:59 -> 2020-01-06 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 7 | 2020-01-07 12:59 -> 2020-01-07 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 8 | 2020-01-08 12:59 -> 2020-01-08 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 9 | 2020-01-09 01:59 -> 2020-01-09 05:15 | AA754 | Urumqi-Shanghai | ENDED |
| 10 | 2020-01-09 12:59 -> 2020-01-09 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 11 | 2020-01-10 12:59 -> 2020-01-10 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 12 | 2020-01-11 12:59 -> 2020-01-11 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 13 | 2020-01-12 12:59 -> 2020-01-12 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 14 | 2020-01-13 12:59 -> 2020-01-13 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 15 | 2020-01-14 12:59 -> 2020-01-14 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 16 | 2020-01-15 12:59 -> 2020-01-15 14:19 | MF85 | Weihai-Shanghai | ENDED |
| 17 | 2020-01-01 01:00 -> 2020-01-01 06:31 | ZH85 | Shanghai-Dalian | ENDED |
| 18 | 2020-01-01 08:11 -> 2020-01-01 11:16 | FM7079 | Shanghai-Nanning | ENDED |
| 19 | 2020-01-01 08:51 -> 2020-01-01 12:41 | GS4209 | Shanghai-Shenyang | ENDED |
| 20 | 2020-01-02 01:00 -> 2020-01-02 06:31 | ZH85 | Shanghai-Dalian | ENDED |

3.13 应对面临的新变化

只考虑你选定的三个应用的变化即可。

3.13.1 变化 1

- 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。

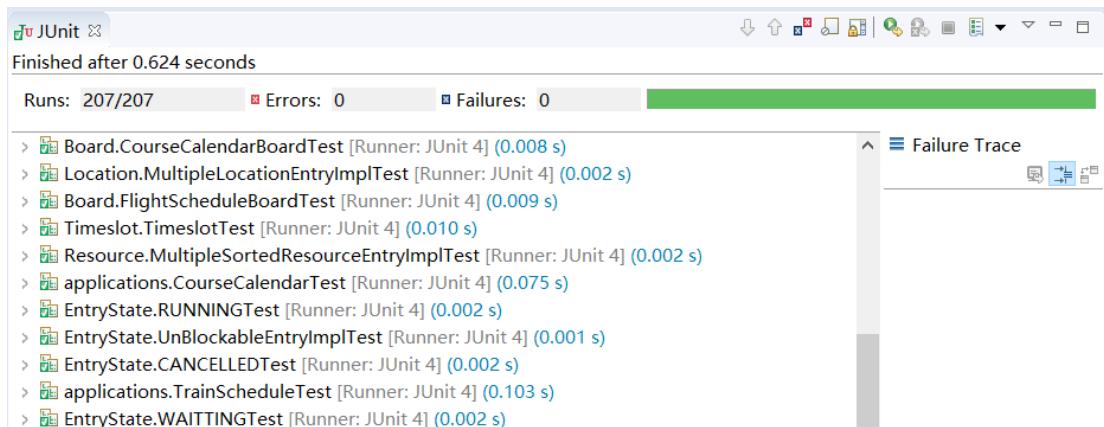
变化 1：单位置变成多个位置，因此接口组合中的单个位置的接口变为多个位置的接口

变化 2：不可阻塞变为可阻塞，因此接口组合中的不可阻塞的接口变为可阻塞的接口。

改变：首先根据两个变化改变一下接口组合。

```
package PlanningEntry;
import EntryState.BlockableEntry;
public interface FlightPlanningEntry<R> extends MultipleLocationEntry<BlockableEntry>, SingleDistinguishResourceEntry<R> {
}
```

其次，重写一下新增的接口的方法。然后重写一下相关的类的测试方法。



航班 change 最主要需要改的地方是 **board**，因为时间和位置都变成了多个，所以对位置的和时间的一一对应的信息的获取工作量更大。

主要 change：1.重写获取和某个位置相关的出发航班和抵达航班的方法

2.稍微改动 visualize 方法和 show 方法

3.对于改变状态的方法需要增加 blocked 状态

的改变。(详细见代码)

另外一个 change 是:
1.控制一个航班的最大的位置数目：因为最多只有一个经停站，所以对于一个航班来说，分配的位置最多数目是 3.

```

    public boolean addPlanningEntry(Location loc1, Location loc2, Calendar time1, Calendar time2, String name) {
        boolean flag = false;
        FlightEntry tra = new FlightEntry(name);
        for(int i = 0; i < flightEntry.size(); i++) {
            if(flightEntry.get(i).getName().equals(name)) {
                tra = flightEntry.get(i);
                flag = true;
            }
        }
        if(flag) {
            if(tra.getLocation().size() > 3) {
                return false; //位置数量超过3
            } else {
                if(!tra.getLocation().get(tra.getLocation().size() - 1).equals(loc1) || !tra.addLocation(loc2)) {
                    return false;
                }
                if(!tra.addTimeslots(new Timeslot(time1, time2))) {
                    return false;
                }
            }
        } else {
    
```

不能再增加位置

2.和 TrainEntryApp 一样，如果航班有经停站的话，需要一次一次输入，即每次输入只能输入两个位置和两个时间。



输入异常的条件有:时间格式错误，第一个时间第二个时间大，第一次输入的终止位置和第二次输入的起始位置不相同，为同一个航班输入的位置超过 3(经停站超过 1)

3.航班增加了阻塞的功能。

```

    /**
     * 阻塞某个高铁车次
     * @param trainNumber 待阻塞的航班号
     * @param time1 待阻塞的航班出发时间
     * @return 提示相关信息
     */
    public String BlockPlanningEntry(String flightNumber, Calendar time1) {
        for(FlightEntry e : flightEntry) {
            if(e.getName().equals(flightNumber)) {
                if(e.getBeginEndTime().getDate1().equals(time1)) {
                    if(e.getState() instanceof RUNNING) {
                        e.Block(); //阻塞高铁车次
                    }
                }
            }
        }
        return "指定的高铁不存在";
    }
}

```



修改时间:3个小时

3.13.2 变化 2

- 如果高铁车次已经分配了车厢资源，则不能被取消。

变化之前，高铁车次分配了资源仍可以取消，可以直接调用状态类的方法实现。

变化之后，高铁车次分配了资源不可以取消，则不能直接调用状态类的方法，为了保证代价更小，无需重新为 ALOCATED 状态类重新设计 changestate 方法，只需要 trainEntry 类只要重写取消计划项的方法就行。

```
    ...
    @Override
    public void CancelPlanningEntry(){
        if(getState() instanceof WAITTING){
            super.setState(CANCELLED);
        }else{
            System.out.println("当前状态为" + super.getState().toString() + "不能转换为 CANCELLED");
        }
    }
}
```

测试: CancelPlanningEntryTest (0.000 s)

```
/*
 * 测试取消计划项的方法
 */
@Test
public void CancelPlanningEntryTest() {
    TrainEntry train = new TrainEntry("复兴号");
    assertTrue(train.getState() instanceof WAITTING); //未分配资源
    Railway r1 = new Railway("aa", Type.BUSINESS, 30, 2019);
    Railway r2 = new Railway("bb", Type.BUSINESS, 30, 2019);
    List<Railway> msrel1 = new ArrayList<Railway>();
    msrel1.add(r1);
    msrel1.add(r2);
    train.setResource(msrel1); //分配资源
    assertTrue(train.getState() instanceof ALOCATED); //未分配资源
    train.CancelPlanningEntry(); //取消已经分配资源的高铁
    assertTrue(train.getState() instanceof ALOCATED); //高铁状态仍然未分配资源
    assertFalse(train.getState() instanceof CANCELLED); //分配资源的高铁不可取消
}
```

修改耗费的时间:10分钟

3.13.3 变化 3

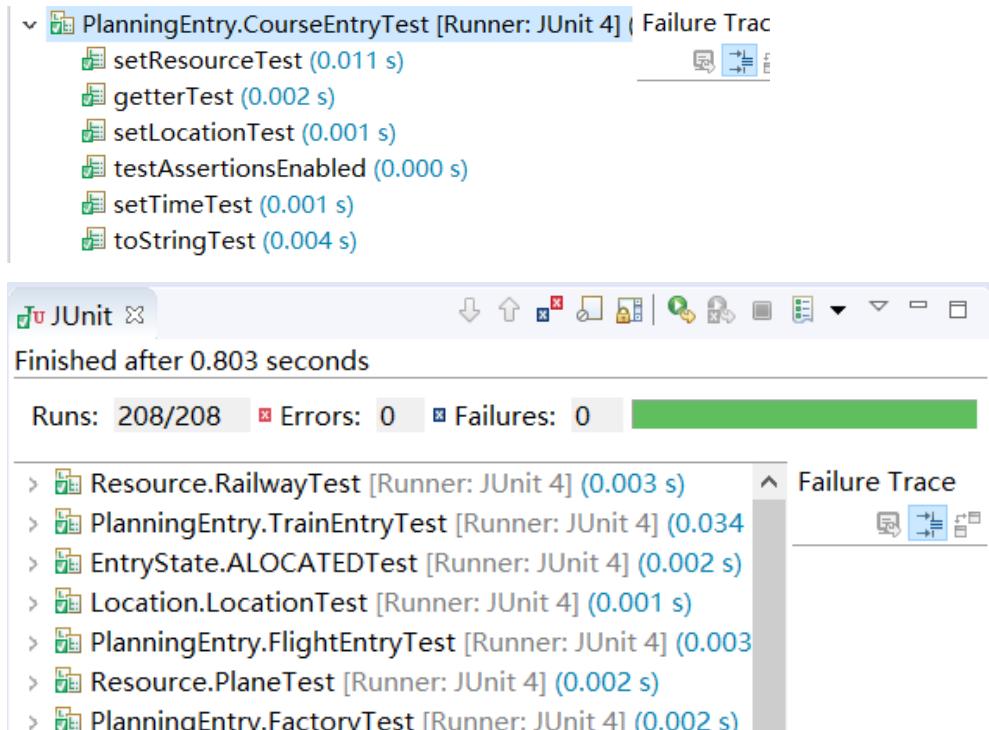
- 课程可以有多个教师一起上课，且需要区分次序(即多名教师的优先级)。

变化 1：由于我采用的是方案 5，接口组合，所以 courseEntry 的接口组合中单一资源的接口得改变为多资源的接口，同时，我默认认为区分老师的优先级是从给某门课程分配教师的顺序来区分。

```
package PlanningEntry;
import Location.SingleLocationEntry;
public interface CoursePlanningEntry<R> extends SingleLocationEntry, UnBlockableEntry<MultipleSortedResourceEntry<R>>
```

同时得重写多资源接口中的方法、以及测试类、所有用到 courseEntry 中和资源操作有关的代码。

重写 CourseEntryTest 方法



APP 不需要修改代码，只是给某门课程分配教师的功能改变了，原来只能分配一个教师，现在可以分配多个教师。

如图所示



| 序号 | 所有的资源 |
|----|--|
| 1 | Teacher [IdNumber=AA, Name=AA, Sex=true, Title=AA] |
| 2 | Teacher [IdNumber=BB, Name=BB, Sex=true, Title=BB] |

| 序号 | 时间 | 课程 | 状态 |
|----|-----------------------------------|----|---------|
| 1 | 2020-03-04 11:30-2020-03-04 11:35 | AA | RUNNING |

| 序号 | 时间 | 课程 | 教师 | 状态 |
|----|-------------|----|-------|---------|
| 1 | 11:30-11:35 | AA | AABB, | RUNNING |

修改耗费的时间:不到 1 个小时

所有的 change:

Showing 24 changed files with 617 additions and 389 deletions.

3.14 Git 仓库结构

请在完成全部实验要求之后，利用 `Git log` 指令或 `Git` 图形化客户端或

GitHub 上项目仓库的 **Insight** 页面，给出你的仓库到目前为止的 **Object Graph**，尤其是区分清楚 **314change** 分支和 **master** 分支所指向的位置。



4 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。
 每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。
 不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

| 日期 | 时间段 | 计划任务 | 实际完成情况 |
|------------|-------|------------------------------|--------|
| 2020-04-20 | 晚上 | 理解题意 | 完成 |
| 2020-04-21 | 下午 | 完成所有的类的组织 | 完成 |
| 2020-04-23 | 晚上 | 设计基本的 Timeslot 类 | 完成 |
| 2020-04-25 | 晚上 | 设计基本的 Location 类和 Resource 类 | 完成 |
| 2020-04-26 | 晚上 | 完成基本类的测试 | 完成 |
| 2020-04-28 | 下午 | 完成 planningEntry 类的设计 | 提前完成 |
| 2020-05-03 | 晚上 | 完成 TrainEntry 等子类的设计 | 未完成 |
| 2020-05-04 | 上午 | 完成具体的子类的设计 | 完成 |
| 2020-05-05 | 全天 | 完成 state 设计模式 | 完成 |
| 2020-05-06 | 全天 | 完成 board 类设计 | 未完成 |
| 2020-05-07 | 全天 | 完成 board 类设计 | 完成 |
| 2020-05-08 | 全天 | 完成 API 及其迭代器 | 完成 |
| 2020-05-09 | 全天 | 完成应用类的设计并且自学 GUI | 未完成 |
| 2020-05-10 | 全天 | 自学 GUI 并设计第一个 APP | 完成 |
| 2020-05-10 | 晚上 | 设计第二个 APP | 完成 |
| 2020-05-11 | 下午+晚上 | 设计第三个 APP | 完成 |
| 2020-05-12 | 全天 | 完善注释、测试类、规约的编写 | 完成 |
| 2020-05-13 | 全天 | 完成实验报告 | 完成 |
| 2020-05-15 | 晚上 | 完成 314change | 完成 |

5 实验过程中遇到的困难与解决途径

| 遇到的难点 | 解决途径 |
|---|-----------------------------|
| 对于 Calendar 类不了解 | 网上搜，学习 |
| 题意不理解，如何将位置和时间联系起来 | 自己思考并且和同学讨论 |
| 可阻塞的 block 方法的参数
不理解，卡了很久 | 问同学，问老师 |
| Jtable 不会 | 在网上自学 |
| GUI 不会 | 在网上自学 |
| API 设计如何通过
PlanningEntry 接口获取位置
资源和时间相关信息 | 在接口中增加公共方法，返回资源，位置等 |
| Trainboard 中如何获取和某
个位置相关的时间及其计划
项 | Map(Calendar,TrainEntry)来解决 |
| 迭代器不会 | 网上自学 |

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验和教训

6.2 针对以下方面的感受

- (1) 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？

面向应用的编程不仅要考虑面向 ADT 编程的问题，比如规约、不变量。更多的考虑是程序的复用性和健壮性。复用性很重要，314change 中，如果复用性很好，修改的代码会很少很少。

- (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？

一是为意义就是保持不变量，防止泄露，这是保证后置条件正确的基本要求，同时防止外部因素改变内部的不变量。 规约是为了用户能够清楚理解我的

程序的作用，这个自然很重要，以后的编程也必须得写。

- (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？

难处：自己从 0 开发任务量真的特别特别大，需要考虑的问题很多很多，乐趣就是完成之后有一种自豪感。

- (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？

设计模式提高了程序的复用性和可维护性，追求高内聚、低耦合，健壮性更强。虽然需要写很多类，但是减少了很多重复的工作，类与类之间的影响也变得更小，便于发现和定位错误，提高正确性。

- (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？

正则表达式太强大了，解析文件并且提取信息，更加的便捷。

- (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？

设计 ADT 的难度在于构建好类之间的框架，结构和厘清类与类之间关系。同时构思好每个功能应该怎么去求解。同时需要掌握扎实的 Java 编程知识，比如这次 GUI 就花费了我很多时间求学习，只学了一点皮毛。同时要选择合理的设计模式，这样不仅可以降低工作量和难度，也便于程序的复用。

- (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？

抽象出公共的方法放在顶层的接口中很重要，他很大程度的决定了程序

的复用性，接口的组合很神奇，体现了高内聚、低耦合的思想，类之间的影响很小，类的继承也可以看作是复用已有的类来进行编程，缩短了开放时间。设计模式的运用最为强大。不管是状态模式、迭代器模式，感觉将更多的模式运用到程序找那个，程序的复用性会大大提高，314change 给了我很深的感受。

(8) 关于本实验的工作量、难度、deadline。

工作量很大，难度中等，deadline 适合。(不得不说重复性的东西太多了，比如我的 GUI 和 applications，很多方法都很类似)

(9) 到目前为止你对《软件构造》课程的评价。

软件构造太强了，感受到设计模式的魅力。