



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2020 年春季学期 计算机学院《软件构造》课程

Lab 3 实验报告

姓名	崔同发
学号	1180300801
班号	1803008
电子邮件	1928511940@qq.com
手机号码	18593159223

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	1
3.2 面向可复用性和可维护性的设计：PlanningEntry<R>	2
3.2.1 PlanningEntry<R>的共性操作	2
3.2.2 局部共性特征的设计方案	2
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）	3
3.3 面向复用的设计：R	6
3.4 面向复用的设计：Location	9
3.5 面向复用的设计：Timeslot	11
3.6 面向复用的设计：EntryState 及 State 设计模式	12
3.7 面向复用的设计：PLanningEntryCollection	14
3.8 面向应用的设计：Board	16
3.9 Board 的可视化：外部 API 的复用	18
3.10 可复用 API 设计及 Façade 设计模式	18
3.10.1 检测一组计划项之间是否存在位置独占冲突	19
3.10.2 检测一组计划项之间是否存在资源独占冲突	19
3.10.3 提取面向特定资源的前序计划项	19
3.11 设计模式应用	20
3.11.1 Factory Method	20
3.11.2 Iterator	20
3.11.3 Strategy	22
3.12 应用设计与开发	22
3.12.1 航班应用	23
3.12.2 高铁应用	27
3.12.3 课表应用	28
3.13 基于语法的数据读入	29
3.14 应对面临的新变化	30

3.14.1 变化 1	30
3.14.2 变化 2	31
3.14.3 变化 3	32
3.15 Git 仓库结构	33
4 实验进度记录	33
5 实验过程中遇到的困难与解决途径	34
6 实验过程中收获的经验、教训、感想	34
6.1 实验过程中收获的经验教训	34
6.2 针对以下方面的感受	34

1 实验目标概述

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），需要通过 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

本次实验需要用到的 jdk 8, Git, eclipse-java-2019-12-R-win32-x86_64 和 EcLEmma 均已在在上面的实现中已经安装配置好。另外由于用到了 GUI 的实现，所以下载了 windowbuilder。

GitHub Lab3 仓库的 URL 地址（Lab3-学号）：

<https://github.com/ComputerScienceHIT/Lab3-1180300801.git>

3 实验过程

3.1 待开发的三个应用场景

列出你所选定的三个应用。

航班管理、高铁车次管理、大学课表管理。

三个应用场景的异同（下表中同一列中颜色相同的说明在对应的应用中处理方式是相同的）：

应用	位置的数 量	位 置 是 否可更改、可 提前设定	资源	是 否 可 阻塞及其时 间描述	时 间 是 否可设定
航班	一个起点 +一个终 点	可 提 前 设定,设定后 不可更改	单 个 可 区分资源(飞 机,带编 号)	否 一个 起止时间对	是(创建 的时候 即可 设定)
高铁	一个起点 +一组中 间位	可 提 前 设定,设定后	多 个 带 次序的可区	是(中途 站停车与发	是(创建 的时候 即可

	置+一个终点	不可更改	分资源(一组前后连接的车厢,均有编号)	车) 一组起止时间对	设定)
课程	一个具体位置	可更改(教室),可提前设定	单个可区分资源(飞机,带编号)	否一个起止时间对	是(创建的时候即可设定)

3.2 面向可复用性和可维护性的设计：PlanningEntry<R>

3.2.1 PlanningEntry<R>的共性操作

```

public boolean Start();//启动计划项

public boolean End();//结束计划项

public boolean Cancell();//取消计划项

public String getState();//获取计划项的状态

public String getLocation();//获取计划项的(第一个)地名

public List<Resource> getResource();//获取计划项的所有资源

public void setStartAndEndTime(Timeslot startAndEndTime);//设置计划项的起止时间

public Timeslot getStartAndEndTime();//获取计划项的起止时间

public String getEntryName();//获取计划项的名称

```

3.2.2 局部共性特征的设计方案

采用了提供的方案五。每个维度分别定义自己的接口，针对每个维度的不同特征取值，分别实现针对该维度接口的不同实现类，实现其特殊操作逻辑。进而，通过接口组合，将各种局部共性行为复合在一起，形成满足每个应用要求的特殊接口（包含了该应用内的全部特殊功能），从而该应用子类可直接实现该组合接口。在应用子类内不是直接实现每个特殊操作，而是通过 **delegation** 到外部每个维度上的各具体实现类的相应特殊操作逻辑。

1. 针对位置数量即位置是否可预先设定设置了三个接口：

```
public interface SingleLocationEntry {...} //单个位置及其设定
```

```
public interface TwoLocationEntry {...} // 一个起点+一个终点及其设定
```

```
public interface MultipleLocationEntry {...} // 一个起点+一组中间站+一个终点及其设定
```

2. 针对资源设置了2个接口：

```
public interface SingleSortedResourceEntry<L> {...} // 设置单个资源
```

```
public interface MultipleSortedResourceEntry<L> {...} // 设置多个资源
```

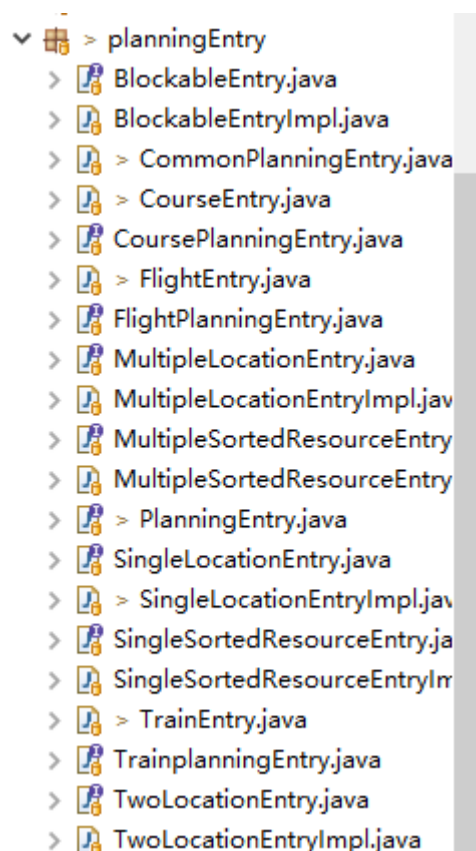
3. 针对是否可阻塞设置了一个接口，因为只有高铁需要阻塞：

```
public interface BlockableEntry {...} // 阻塞
```

4. 针对时间及其设置：

因为所选三个应用都是创建的时候设置时间，是三者所共有的，所以将其放到了 CommonPlanningEntry 中实现。

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. 设计一个类 CommonPlanningEntry 来实现 PlanningEntry:

```
public class CommonPlanningEntry implements PlanningEntry {
```

```
    private Context currentState = new Context(StateWaiting.instance);
```

```

    private Timeslot startAndEndTime; //开始时间
    //AF:代表一个计划项的起始时间
    //RI:起始时间和结束时间符合 yyyy-MM-dd HH:mm的语法规则
    //Safety from rep exposure:所有属性均为私有

    public CommonPlanningEntry(Timeslot startAndEndTime) {
        setStartAndEndTime(startAndEndTime);
    }
    ...
}

2. 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。
    1.) 针对航班应用：
    public interface FlightPlanningEntry extends TwoLocationEntry,
    SingleSortedResourceEntry<Flight> {} //组合接口

    public class FlightEntry extends CommonPlanningEntry implements
    FlightPlanningEntry {

        private String Flightnumber;
        private TwoLocationEntryImpl te;
        private SingleSortedResourceEntryImpl<Flight> se;
        //AF:一个拥有航班号，起止时间，起止地点，使用飞机的航班
        //RI:true
        //Safety from rep exposure:所有属性均为私有

        /**
         * 构造器
         * @param Flightnumber 航班号
         * @param startAndEndTime 起止时间对
         * @param te 两个位置设置器
         * @param se 单个资源设置器
         */
        public FlightEntry(String Flightnumber, Timeslot
startAndEndTime, TwoLocationEntryImpl
te, SingleSortedResourceEntryImpl<Flight> se) {
            super(startAndEndTime);
            this.Flightnumber = Flightnumber;
            this.te = te;
            this.se = se;
            if(this.se != null)
                setCurrentState("a");
        }
        ...
    }

```

}//接口实现

2.) 针对高铁应用:

```
public interface TrainplanningEntry extends MultipleLocationEntry,
MultipleSortedResourceEntry<Carriage>, BlockableEntry {}//统一接口
```

```
public class TrainEntry extends CommonPlanningEntry implements
TrainplanningEntry {

    private String trainNumber;//计划项名称
    private MultipleLocationEntryImpl mle;//设置多个位置
    private MultipleSortedResourceEntryImpl<Carriage> msre;//设置多个资源
    private BlockableEntryImpl be;//中停
    private int blockNum;//记录block次数，不能超过中间站次数
    //AF:一个拥有列车名称名称，起止时间，起止及中间站，行驶列车的高铁计划
    //RI:true
    //Safety from rep exposure:所有属性均为私有

    /**
     * 构造器
     * @param trainNumber 列车号
     * @param startAndEndTime 起止时间对
     * @param mle 多个位置设置器
     * @param msre 多个资源设置器
     * @param be 阻塞器
     */
    public TrainEntry(String trainNumber, Timeslot
startAndEndTime, MultipleLocationEntryImpl
mle, MultipleSortedResourceEntryImpl<Carriage> msre, BlockableEntryImpl be) {
        super(startAndEndTime);
        this.trainNumber = trainNumber;
        this.mle = mle;
        this.msre = msre;
        this.be = be;
        blockNum = 0;
        if(this.msre != null)
            setCurrentState("a");
    }
    ...
} //具体实现
```

3.) 针对课程应用:

```
public interface CoursePlanningEntry extends SingleLocationEntry,
SingleSortedResourceEntry<Teacher> {}//组合接口
```



```

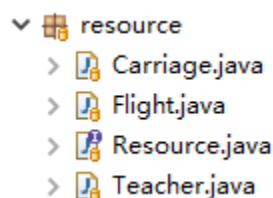
public class CourseEntry extends CommonPlanningEntry implements
CoursePlanningEntry{

    private String className;
    private SingleLocationEntryImpl se;
    private SingleSortedResourceEntryImpl<Teacher> ssre;
    //AF:一个拥有课程名称, 起止时间, 发生地点, 上课教室的课程
    //RI:true
    //Safety from rep exposure:所有属性均为私有

    /**
     * 构造器
     * @param className 课程名称
     * @param startAndEndTime 起止时间
     * @param se 单个位置设置器
     * @param ssre 单个资源设置器
     */
    public CourseEntry(String className, Timeslot
startAndEndTime, SingleLocationEntryImpl
se, SingleSortedResourceEntryImpl<Teacher> ssre) {
        super(startAndEndTime);
        this.className = className;
        this.se = se;
        this.ssre = ssre;
        if(this.ssre != null)
            setCurrentState("a");
    }
    ...
} //实现接口

```

3.3 面向复用的设计：R



针对三种应用里面的资源 R，统一设置了一个接口 Resource, 三种资源分别实现接口 Resource，实现了个性化。

```
public interface Resource {...} //资源接口
```

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

```
public class Flight implements Resource{

    private String numbering; //飞机编号
    private String modelNumber; //机型
    private int seatsNumber; //座位数
    private double age; //机龄

    //AF:代表一架具有特定编号，机型，座位数和机龄的客机
    //RI：numbering和modelNumber非空，seatsNumber和age大于0
    //Safety from rep exposure:所有属性均为私有

    /**
     * checkRep
     */
    public void checkRep() {
        assert !numbering.equals(null);
        assert !modelNumber.equals(null);
        assert seatsNumber>0;
        assert age>0;
    }

    /**
     * 构造器
     */
    public Flight(String numbering,String modelNumber,int
seatsNumber,double d) {
        this.numbering = numbering;
        this.modelNumber = modelNumber;
        this.seatsNumber = seatsNumber;
        this.age = d;
        checkRep();
    }
    ...
}
```

2. 列车车厢：高铁应用中的资源是“车厢”，属性包括车厢唯一编号、类型（商务、一等、二等、软卧、硬卧、硬座、行李车、餐车）、定员数（例如 100 人）、出厂年份。

```
public class Carriage implements Resource{

    private String numbering; //车厢编号
    private String Type; //类型
    private int personnelNumber; //定员数
```

```

    private String FactoryYear; //出厂年号

    //AF:代表一个具有特定编号, 类型, 定员数和出厂年号的车厢
    //RI: numbering, Type和FactoryYear非空, personnelNumber>0,且年号在2000-
    2020年之间
    //Safety from rep exposure:所有属性均为私有

    /**
     * checkRep
     */
    public void checkRep() {
        assert !numbering.equals(null);
        assert !Type.equals(null);
        assert !FactoryYear.equals(null);
        Pattern p = Pattern.compile("(20[01][0-9])|(2020)");
        Matcher mc = p.matcher(FactoryYear);
        assert mc.find();
        assert personnelNumber>0;
    }

    /**
     * 构造器
     */
    public Carriage(String numbering,String Type,int personnelNumber,String
    FactoryYear) {
        this.numbering = numbering;
        this.Type = Type;
        this.personnelNumber = personnelNumber;
        this.FactoryYear = FactoryYear;
        checkRep();
    }
    ...
}

```

3. 教师：课表应用中的资源为“教师”，属性包括身份证号、姓名、性别、职称。

```

public class Teacher implements Resource{

    private String IDNumber; //身份证号
    private String name; //姓名
    private String sex; //性别
    private String title; //职称

    //AF:代表一位具有特定身份证号, 姓名, 性别和职称的教师
    //RI: numbering,modelNumber,sex和title非空

```

```






//Safety from rep exposure:所有属性均为私有

/**
 * checkRep
 */
public void checkRep() {
    assert !IDNumber.equals(null);
    assert !name.equals(null);
    assert !sex.equals(null);
    assert !title.equals(null);
}

/*
 * 构造器
 */
public Teacher(String IDNumber,String name,String sex,String title) {
    this.IDNumber = IDNumber;
    this.name = name;
    this.sex = sex;
    this.title = title;
    checkRep();
}
...
}

```

3.4 面向复用的设计：Location

 location
 >  Airport.java
 >  ClassRoom.java
 >  Location.java
 >  RailwayStation.java

一个“位置”对象的属性包括：经度、纬度、名称、是否可共享使用。所谓 的“是否可共享”是指：该位置是否可同时被多个计划项所使用。例如：“北京 南”高铁站是可共享的，多个高铁车次可在同一时间停在或经过该站；“D01 教 室”是不可共享的，同一时间只能有一个课程在此上课。

此处设计了一个接口 Location，并根据三个应用需求分别实例化：Airport,RailwayStation, 和 ClassRoom。这样设计符合 LSP 原则，可复用性良好，便于维护。

public interface Location {...} //Location接口，统一三个类型的位置。

Airport的属性包括：经度、纬度、名称、是否可共享使用。经纬度暂时设置，未有进一步需求，有相应的需求可以很简单的设置。飞机场是可共用的。

public class Airport **implements** Location {

```

private String LocationName;//名称
private String Longitude;//经度,格式为X:mm.m,例如N:35.6表示北纬35.6度
private String Dimensions;//维度,格式为X:mm.m,例如E:33.2表示东经33.2度
private boolean Shareble;//是否可共享
//AF:一个特定经纬度的位置
//RI:Location非空, Longitude和Dimensions合理且格式为X:mm.m
//Safety from rep exposure:所有属性均为私有

/**
 * 构造器
 * @param LocationName
 */
public Airport(String LocationName) {
    this.LocationName = LocationName;
    this.Shareble = true;
}
...
}

```

RailwayStation的属性包括：经度、纬度、名称、是否可共享使用。经纬度暂时设置，未有进一步需求，有相应的需求可以很简单的设置。高铁站是可共用的。

```

public class RailwayStation implements Location {

    private String LocationName;//名称
    private String Longitude;//经度,格式为X:mm.m,例如N:35.6表示北纬35.6度
    private String Dimensions;//维度,格式为X:mm.m,例如E:33.2表示东经33.2度
    private boolean Shareble;//是否可共享
    //AF:一个特定经纬度的位置
    //RI:Location非空, Longitude和Dimensions合理且格式为X:mm.m
    //Safety from rep exposure:所有属性均为私有

    /**
     * 构造器
     * @param LocationName
     */
    public RailwayStation(String LocationName) {
        this.LocationName = LocationName;
        this.Shareble = true;
    }
    ...
}

```

ClassRoom的属性包括：经度、纬度、名称、是否可共享使用。经纬度暂时设置，未有进

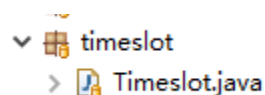
一步需求，有相应的需求可以很简单的设置。教室是不可共用的。

```
public class Classroom implements Location {

    private String LocationName; // 名称
    private String Longitude; // 经度, 格式为X:mm.m, 例如N:35.6表示北纬35.6度
    private String Dimensions; // 维度, 格式为X:mm.m, 例如E:33.2表示东经33.2度
    private boolean Shareble; // 是否可共享
    // AF: 一个特定经纬度的位置
    // RI: Location非空, Longitude和Dimensions合理且格式为X:mm.m
    // Safety from rep exposure: 所有属性均为私有

    /**
     * 构造器
     * @param LocationName
     */
    public Classroom(String LocationName) {
        this.LocationName = LocationName;
        this.Shareble = false;
    }
    ...
}
```

3.5 面向复用的设计: Timeslot



Timeslot: 为一个单独的“起止时间对”设计 Timeslot 类，它是一个带有起始时间和结束时间的 ADT，包含日期（年/月/日）和时间（时/分），符合 yyyy-MM-dd HH:mm 的语法规则。

```
public class Timeslot {

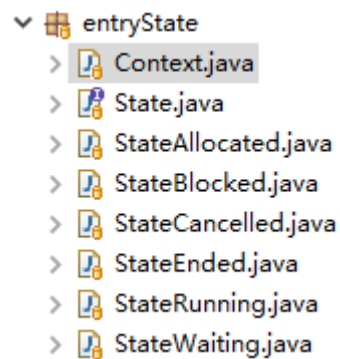
    private String start; // 起始时间
    private String end; // 结束时间

    // AF: 表示一个带有起始时间和结束时间的timeslot
    // RI: 起始时间和结束时间符合 yyyy-MM-dd HH:mm的语法规则
    // Safety from rep exposure: 所有属性均为私有

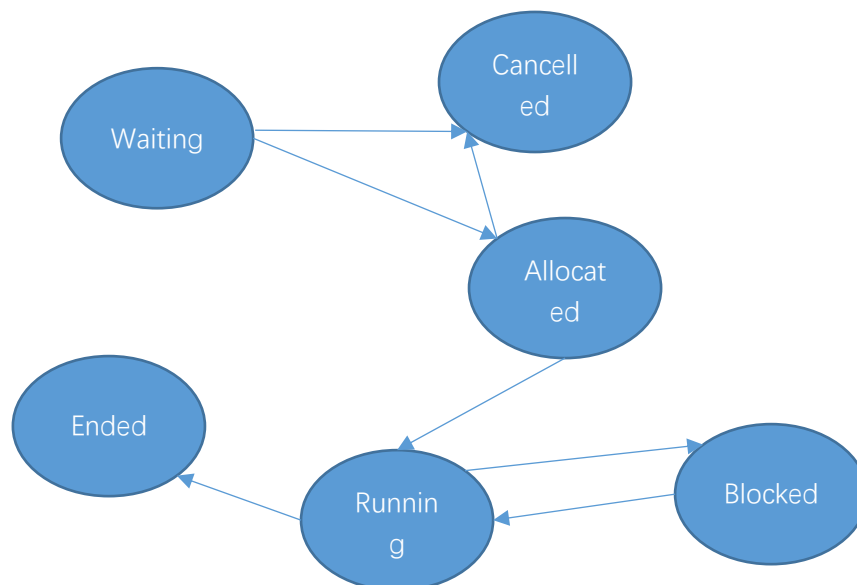
    public Timeslot(String start, String end) {
        if(!checkRep(start, end))
            System.out.println("起止时间输入不符合要求");
        else {
```

```
        this.start = start;  
        this.end = end;  
    }  
}  
...  
}
```

3.6 面向复用的设计：EntryState 及 State 设计模式



使用状态模式实现了 EntryState，总共设置了 6 个状态：Waiting（资源未分配），Allocated（已分配资源），Running（进行中），Blocked（中断，可再次启动），Ended（结束），Cancelled（取消）。这 6 个状态的转换模式如下：



其中 State 作为接口，设置了 move 函数：

```
public interface State {  
    State move(String c);  
}
```

StateWaiting（资源未分配），StateAllocated（已分配资源），StateRunning（进行中），StateBlocked（中断，可再次启动），StateEnded（结束），StateCancelled（取消）分别按照自己的转移模式实现了 move 函数。例如：

```
public class StateAllocated implements State {

    static StateAllocated instance = new StateAllocated();

    private StateAllocated() {}

    @Override
    public State move(String c) {
        // TODO Auto-generated method stub
        switch(c) {
            case "r":
                return StateRunning.instance;
            case "c":
                return StateCancelled.instance;
            default:
                throw new IllegalArgumentException();
        }
    }

    @Override
    public String toString() {
        return "Allocated";
    }
}
```

因为个状态的构造函数都是私有的，所以使用 Context 类用来实例化状态：

```
public class Context {

    State state; //保存对象的状态

    /**
     * 设置初始状态
     * @param s 初始状态
     */
    public Context(State s) {
        state = s;
    }

    /**
     * 接收外部输入，开始转换状态
     * @param c 外部输入参数，据此转换状态
     */
}
```



```

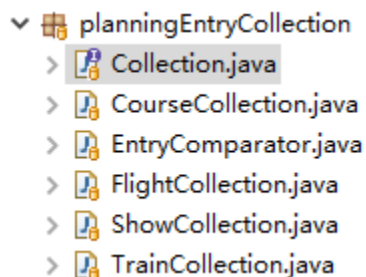
    */
    public void move(String c) {
        state = state.move(c);
    }

    /**
     *
     * @return 当前状态
     */
    public State getState() {
        return this.state;
    }
}

```

在三个应用中，航班和课程表应用都是拥有StateWaiting（资源未分配），StateAllocated（已分配资源），StateRunning（进行中），StateEnded（结束），StateCancelled（取消）这五个状态，而没有StateBlocked（中断，可再次启动）状态。只有高铁拥有全部的状态：StateWaiting（资源未分配），StateAllocated（已分配资源），StateRunning（进行中），StateBlocked（中断，可再次启动），StateEnded（结束），StateCancelled（取消）。

3.7 面向复用的设计：PLanningEntryCollection



为了方便使用，设计了PlanningEntryCollection，用于存放一个文件中所有的计划项。具体设计模式为：一个接口+三个实现。如上面截图所示，Collection作为接口，里面放置了三个具体实现的公有方法。

```
public interface Collection {...}
```

FlightCollection，TrainCollection，CourseCollection分别作为Collection的对应三个应用的实现。在这三个Collection中都分别实现了迭代器功能。以FlightCollection为例：

```
public class FlightCollection implements Iterable<FlightEntry>,Collection{
```

```
    private List<FlightEntry> flightCollection = new
    ArrayList<FlightEntry>();//存放所有航班计划
```

```
    private Set<Location> airports = new HashSet<Location>();//存放所有航班
    计划中出现的机场
```

```

    private Set<Flight> flights = new HashSet<Flight>(); //存放所有航班计划中
    使用到的飞机 //AF:一个存储航班计划, 可用位置和资源的计划项集
    //RI : true
    //Safety from rep exposure:所有属性均为私有

```

```

    /**
     * 构造器
     * @param fileName 存储计划项的文件
     * @throws IOException
     */
    public FlightCollection(String fileName) throws IOException {...}
    ...
}

```

FlightCollection以从外部文件读取的数据形成一个航班计划项集合, 里面存放了文件中所有的航班计划项及其经过的机场和使用的飞机。并且对应个rep都设置了对应的一些方法。另外还实现了迭代器:

```

public Iterator<FlightEntry> iterator(){
    return new Itr();
}

/**
 * 迭代器
 * @author Administrator
 */
class Itr implements Iterator<FlightEntry>{

    Iterator<FlightEntry> it;

    public Itr() {
        it = flightCollection.iterator();
    }

    @Override
    public boolean hasNext() {
        // TODO Auto-generated method stub
        return it.hasNext();
    }

    @Override
    public FlightEntry next() {
        // TODO Auto-generated method stub
        return it.next();
    }
}

```

```
}
```

这里迭代器直接放在了FlightCollection内部作为一个辅助函数。

另外在上面截图中还出现了EntryComparator类，这是Comparator的一个实例化，用于比较两个计划项（PlanningEntry）的大小，方便实现排序。在各应用的Collection中，在从文件中读取数据形成计划项集合后，通过EntryComparator实现了按计划项时间从小到大的排序。

还可以看到一个ShowCollection类，这是用来展示各计划项集合的内容的，使用GUI实现，同Board。样式如下：



航班号	起止时间	位置	状态	
ZH9673	2020-05-01 15:06 to 2020-05-01 18:00	Guangzhou-Jinan	Allocated	▲
CZ276	2020-05-01 18:26 to 2020-05-01 22:03	Guiyang-Xian	Allocated	≡
SC1342	2020-05-01 18:28 to 2020-05-01 22:16	Guangzhou-Kunming	Allocated	
SC956	2020-05-01 22:11 to 2020-05-01 23:20	Changchun-Taipei	Allocated	
KL98	2020-05-01 23:39 to 2020-05-02 03:13	Changchun-Hongk...	Allocated	
AA0644	2020-05-02 00:04 to 2020-05-02 03:51	Jinan-Qingdao	Allocated	
ZH85	2020-05-02 01:00 to 2020-05-02 06:31	Shanghai-Dalian	Allocated	
CA04	2020-05-02 01:50 to 2020-05-02 04:12	Tianjin-Xian	Allocated	
HU947	2020-05-02 02:39 to 2020-05-02 08:18	Nanchang-Fuzhou	Allocated	
MF139	2020-05-02 02:41 to 2020-05-02 06:07	Kunming-Qingdao	Allocated	
CX1909	2020-05-02 03:25 to 2020-05-02 09:15	Shenyang-Tianjin	Allocated	
MU059	2020-05-02 04:08 to 2020-05-02 05:46	Wuhan-Xian	Allocated	▼

3.8 面向应用的设计：Board

每个特定位置都有一个信息板，用于实时公布该位置上过去发生过的和后续即将发生的计划项，以便于广大用户及时了解计划项的状态变化。针对三个应用，分别实现了它们各自的信息板，实现方式都是 GUI。而 Board 中的数据来自上面提到的 Collection，针对每一个地方和一个特定的时间段，Board 从对应的 Collection 中获取相应的数据，并图形化展示出来。

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

2020-05-16 01:04:23(当前时间),Tianjin				
抵达航班				
预计抵达时间	编号	起点-终点	状态	
2020-05-15 19:16	N4446	Tianjin-Dalian	Allocated	▲
2020-05-16 00:55	B9410	Tianjin-Xiamen	Allocated	≡
2020-05-16 10:36	B7226	Tianjin-Urumqi	Allocated	
2020-05-16 15:32	N0552	Tianjin-Nanchang	Allocated	
2020-05-16 16:59	B3494	Tianjin-Wuhan	Allocated	▼
出发航班				
预计起飞时间	编号	起点-终点	状态	
2020-05-16 00:50	B1979	Hefei-Tianjin	Allocated	▲
2020-05-16 11:01	B2402	Nanning-Tianjin	Allocated	≡
2020-05-16 11:01	B3955	Nanning-Tianjin	Allocated	
2020-05-16 13:23	B4404	Qingdao-Tianjin	Allocated	▼

2. 高铁站里的车次状态显示屏：与航班的类似，唯一的区别是要额外考虑经停高铁车次。因为在表格里位置不够容纳下一列高铁经过的所有高铁站，所以也只是展示了起点和终点，具体查看中间站点可以在相应的 APP 里面，有对应的功能来查找每一列高铁途经的所有站点。

2020-05-16 01:29:07(当前时间),Tianjin				
抵达航班				
预计抵达时间	编号	起点-终点	状态	
2020-05-16 03:39	[B5104]	Tianjin-Taiyuan	Allocated	▲
2020-05-16 06:36	[N6964]	Tianjin-Nanjing	Allocated	≡
2020-05-16 07:22	[N8547]	Tianjin-Guiyang	Allocated	
2020-05-16 07:56	[N4288]	Tianjin-Nanning	Allocated	
2020-05-16 20:57	[B5906]	Tianjin-linan	Allocated	▼
出发航班				
预计起飞时间	编号	起点-终点	状态	
2020-05-16 06:00	[B3048]	Yantai-Tianjin	Allocated	▲
2020-05-16 09:23	[N8295]	Taiyuan-Tianjin	Allocated	≡
2020-05-16 19:02	[N5482]	Haikou-Tianjin	Allocated	▼

3. 教室外悬挂的教室占用情况表：列出当日在本教室内上的所有课程及其状态。与航班的相同，第一列的时间都是计划时间。



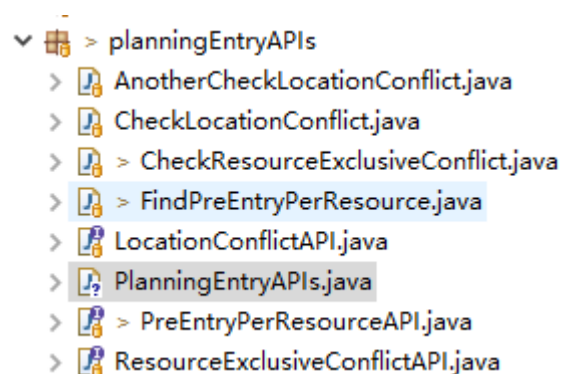
2020-05-16 01:32:40(当前时间),正心24			
教室			
上课计时间	课程	教师	状态

由于是在周六弄的，然后课表上这天没课，所以显示的是一个空表。

3.9 Board 的可视化：外部 API 的复用

使用 GUI 实现了 Board 的可视化。具体在 3.8 节中已经讲过，这里不再赘述。

3.10 可复用 API 设计及 Façade 设计模式



三个 API，每个 API 对应一个接口和一个实现，然后 PlanningEntryAPIs 作为门面，将三种实现都放在了里面：

```
public class PlanningEntryAPIs {

    private LocationConflictAPI lf; // 位置矛盾
    private ResourceExclusiveConflictAPI pec; // 资源矛盾
    private PreEntryPerResourceAPI ppr; // 前序计划

    public PlanningEntryAPIs() {
        lf = new CheckLocationConflict();
        pec = new CheckResourceExclusiveConflict();
    }
}
```

```

        ppr = new FindPreEntryPerResource();
    }

    public boolean checkLocationConflict(List<PlanningEntry> entries) {
        return lf.checkLocationConflict(entries);
    }

    public boolean checkResourceExclusiveConflict(List<PlanningEntry>
entries) {
        return pec.checkResourceExclusiveConflict(entries);
    }

    public PlanningEntry findPreEntryPerResource(Resource r, PlanningEntry
e, List<PlanningEntry> entries) {
        return ppr.findPreEntryPerResource(r,e,entries);
    }
}

```

3.10.1 检测一组计划项之间是否存在位置独占冲突

检测一组计划项之间是否存在位置独占冲突：如果两个计划项在同一时间点上占用了不可共享的位置，那么就存在了位置冲突。例如：某次《软件构造》课的时间是本日 8:00-10:00，而某次《计算机系统》课是在本日 9:00-11:00，二者均在 D01 教室，而“教室”这种位置是不可共享的，那么在 9:00-10:00 这个时间段里就存在着位置冲突。但是对机场、车站等可共享位置来说，则不会存在位置冲突

实现方式：对于位置不可共享的计划项，遍历计划项集，对于位置相同的两个计划项，对比其时间是否矛盾。若矛盾，返回 false，否则返回 true。

3.10.2 检测一组计划项之间是否存在资源独占冲突

检测一组计划项之间是否存在资源独占冲突：如果两个计划项在同一时间点上占用了同样的资源，那么就存在了资源冲突，例如同一个教师、同一个车厢、同一架飞机。对 ROM、学习资料等不可区分个体的资源，无需考虑资源独占冲突。

实现方式很简单：遍历计划项集，对比每两个计划项，看它们是否使用了相同的资源，若是，看它们的时间是否冲突，是的话返回 true。

3.10.3 提取面向特定资源的前序计划项

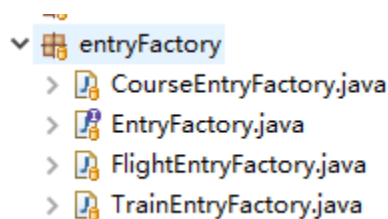
提取面向特定资源的前序计划项：针对某个资源 r 和使用 r 的某个计划项 e，从一组计划项中找出 e 的前序 f，f 也使用资源 r，f 的执行时间在 e 之前，且在 e 和 f 之间不存在使用资源 r 的其他计划项。若不存在这样的计划项 f，则返回 null。如果存在多个这样的 f，返回其中任意一个即可。对 ROM、学习资料等不可区分个体的资源，该 API 不适

用。

实现方式也很简单：遍历计划项集中时间在给定计划项之前的计划项，若二者使用的资源相同，记录下这个计划项，接着比对，直到时间与给定计划项相同或大于为止，返回最后一个被记录的计划项。

3.11 设计模式应用

3.11.1 Factory Method



如上图所示，包括一个接口，三个对应不同应用的实现类。每一个工厂返回的是一个对应的计划项。

例如 FlightEntryFactory:

```
public class FlightEntryFactory implements EntryFactory<FlightEntry> {  
    public FlightEntry getEntry(String S) {  
        /**  
         * @param front  
         * @param S  
         * @return S中front所在行中在front后面的字符串  
         */  
        private String getMessage(String front,String S) {  
    }  
}
```

通过从文件中读取的一个计划项的信息字符串构造一个 FlightEntry 并返回之。避免了通过 new 的方式实例化具体的 PlanningEntry 子类型

3.11.2 Iterator

迭代器在上面的 Collection 部分已经讲过，这里再简单说一下：

对于每一个计划项集合（PlanningEntryCollection），都实现了迭代功能。在 Iterator 里面重写了 next()和 hasNext()方法，并在 PlanningEntryCollection 中提供了一个返回迭代器的方法，这样就实现了对计划项集的遍历功能。

重写 next()和 hasNext()的方法很简单，因为容器实质上是一个 List，所以直接利用 List 的迭代器来重写这两个方法就行了。

以下展示了 FlightEntryCollection 中的迭代器：

```
public Iterator<FlightEntry> iterator(){
```

```
        return new Itr();
    }

    /**
     * 迭代器
     * @author Administrator
     *
     */
    class Itr implements Iterator<FlightEntry>{

        Iterator<FlightEntry> it;

        public Itr() {
            it = flightCollection.iterator();
        }

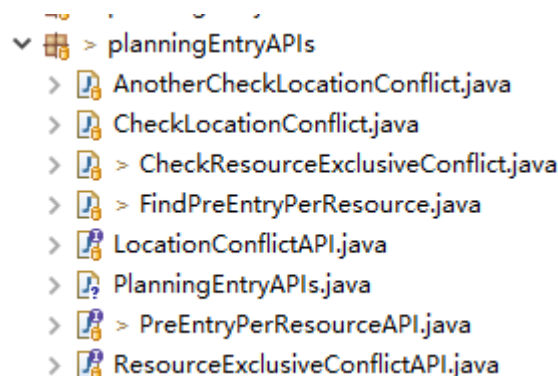
        @Override
        public boolean hasNext() {
            // TODO Auto-generated method stub
            return it.hasNext();
        }

        @Override
        public FlightEntry next() {
            // TODO Auto-generated method stub
            return it.next();
        }

    }
```

这里与要求稍有不同：要求是在 Board 中实现迭代器及排序功能，不过我多增加了一个 PlanningEntryCollection，二者实质上是一致的，在 PlanningEntryCollection 中已经实现了迭代器，所以就没有再在 Board 中重复这一动作了。另外的 Comparator 在上面描述 PlanningEntryCollection 时已经讲过。

3.11.3 Strategy



策略模式要求在 3.10 节的 API 设计中，选定其中一个 API，使用两种不同的算法实现之，进而在客户端应用程序中使用 strategy 模式灵活替换不同的算法。

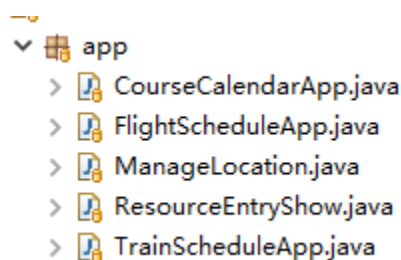
这里我选择的 API 是检查位置冲突：public boolean checkLocationConflict(List entries)

在上面的截图中可以看到一个名为 LocationConflictAPI 的接口类，然后还有另外两个它的实现类：CheckLocationConflict 和 AnotherCheckLocationConflict，在这两个类中，分别以不同的算法实现了检查位置冲突。前者是先遍历找出位置相同的计划项，再查看它们时间是否矛盾，以此查看是否存在位置冲突；后者刚好反过来，先遍历找出时间矛盾的计划项，再不济位置是否相同，以此查看是否存在位置冲突。

在应用中，可以根据实际情况灵活的选择使用哪一种策略。

3.12 应用设计与开发

对选定的三个应用均以 GUI 的方式实现。



上图中，FlightScheduleApp、TrainScheduleApp、CourseCalendarApp 分别为所选三个应用的实现，ManageLocation 为三个应用中用来管理位置的公共部分，ResourceEntryShow 用来展示使用某个资源的计划项。

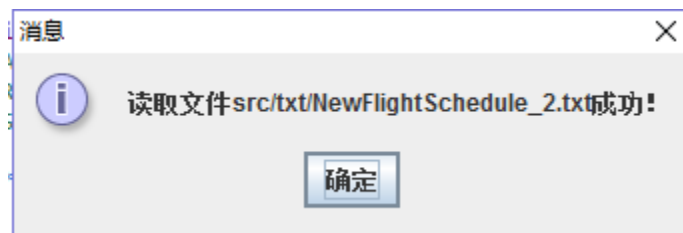
3.12.1 航班应用

如上图所示，以 GUI 的方式实现了航班应用。具体实现的功能一眼看见。在界面的下方位置放置了一个使用说明，介绍了使用这个 app 的方法，具体如下：

- 1.“所有航班计划”：查看当前计划集中的所有航班计划。
- 2.“当前位置航班”：只需在起飞机场处输入位置，就可查看该位置的航班表。
- 3.“当前资源航班”：输入对应的飞机编号，即可查看该飞机的使用情况。
- 4.“当前航班状态”：输入航班号和出发时间即可查看该航班当前状态。
- 5.“新增航班”：除计划项集不需要输入，其它都得输入。
- 6.“分配资源”：输入航班号，飞机编号，出发时间即可为该趟航班分配对应编号的飞机。
- 7.“启动当前航班”：输入航班号和起飞时间，点击即可启动。
- 8.“结束当前航班”：类比 7。
- 9.“查看冲突”：
- 10.“更改计划项集”：在左侧选择一个计划项集（文件），点击即可。
- 11.“检查矛盾”：查看是否存在资源抢占矛盾。
- 12.“前序计划项”：输入航班号，飞机编号，出发时间，以此搜索计划项集中与输入航班使用同一资源且时间在前面的计划项。
- 13.“管理资源”：可增加（删除、修改）当前计划项集中的资源。
- 14.“管理位置”：同 12。
- 15.“读取新文件”：在新文件处输入要读取的文件路径，点击即可以这个文件作为新的计划项集合。

以下简单展示一下部分功能：

1. 运行此 APP，首先会提示一个读取 XX 文件成功，点击确定或“X”进入主界面。



2. 主界面:

12. "前序计划项": 输入航班号, 飞机编号, 出发时间, 以此搜索计划项集中与输入

13. "管理资源": 可增加(删除、修改)当前计划项集中的资源。

14. "管理位置": 同12.

3. 单击"所有航班计划", 查看当前计划项集合中的所有计划项:

航班号	起止时间	位置	状态
MF139	2020-05-02 02:41 to 2020-05-02 06:07	Kunming-Qingdao	Allocated
CX1909	2020-05-02 03:25 to 2020-05-02 09:15	Shenyang-Tianjin	Allocated
MU059	2020-05-02 04:08 to 2020-05-02 05:46	Wuhan-Xian	Allocated
MU6171	2020-05-02 04:35 to 2020-05-02 06:25	Ningbo-Qingdao	Allocated
CZ29	2020-05-02 05:55 to 2020-05-02 07:18	Guangzhou-Hefei	Allocated
CZ6145	2020-05-02 06:56 to 2020-05-02 12:31	Guangzhou-Kunming	Allocated
NX49	2020-05-02 08:11 to 2020-05-02 11:42	Taiyuan-Zhengzhou	Allocated
GS4209	2020-05-02 08:51 to 2020-05-02 12:41	Shanghai-Shenyang	Allocated
SC01	2020-05-02 10:43 to 2020-05-02 12:03	Yantai-Hongkong	Allocated
MF85	2020-05-02 12:59 to 2020-05-02 14:19	Weihai-Shanghai	Allocated
SC1342	2020-05-02 18:28 to 2020-05-02 22:16	Guangzhou-Kunming	Allocated
SC956	2020-05-02 22:11 to 2020-05-02 23:20	Changchun-Taipei	Allocated

12. 前序计划项: 输入航班号, 飞机编号, 出发时间, 以此搜索计划项集中与输入

4. 在起飞机场处输入一个位置, 点击"单签位置航班"即可查看此地在一段时间内的信息板:

2020-05-16 03:04:45(当前时间),Tianjin

抵达航班

预计抵达时间	编号	起点-终点	状态

出发航班

预计起飞时间	编号	起点-终点	状态
2020-05-16 03:25	N5888	Shenyang-Tianjin	Allocated

5. “启动当前航班”：输入航班号和起飞时间，点击：

所有航班计划 当前位置航班 当前资源航班 当前航班状态

航班号 AA0644 飞机编号

出发时间 2020 05 01 00 04

抵达时间 2020 01 01 00 00

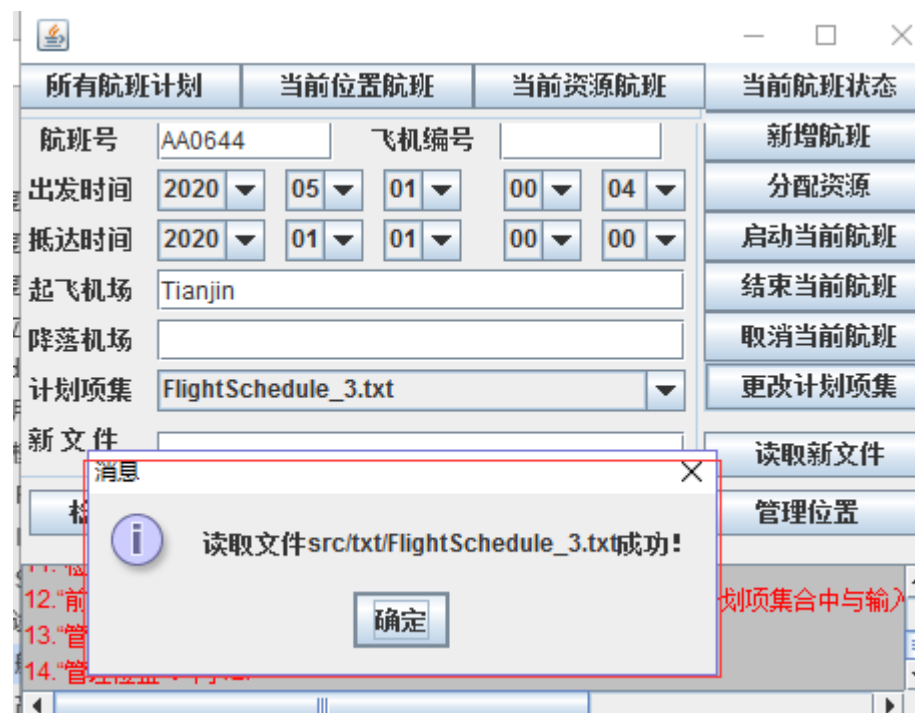
起飞机场 降落机场 计划项集 新文件

消息 启动成功，当前状态：Running 确定

检查矛盾 前序计... 管理资源 管理位置

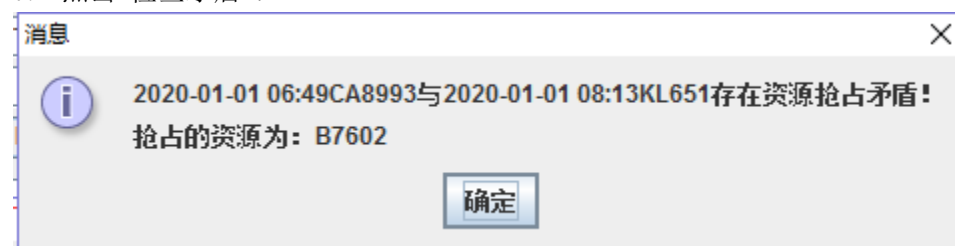
12.“前序计划项”：输入航班号，飞机编号，出发时间，以此搜索计划项集合中与输入
13.“管理资源”：可增加（删除、修改）当前计划项集中的资源。
14.“管理位置”：同12.

6. “更改计划项集”：在左侧选择一个计划项集（文件），点击：



此时再查看所有航班计划项就会发现内容已经改变为对应文件内的计划项集合。

7. 点击“检查矛盾”:



8. 输入航班号，飞机编号和起飞时间，点击前序计划项，即可查看前序计划项:



9. 点击管理资源，输入正确的信息即可管理资源：

另外一些功能就不一一展开了。

3.12.2 高铁应用

与航班应用差不多：

红色字部分是使用说明，具体如下：

特别注意：输入中间站时，两个中间站之间以逗号隔开；输入多个车厢编号时，也以逗号隔开。

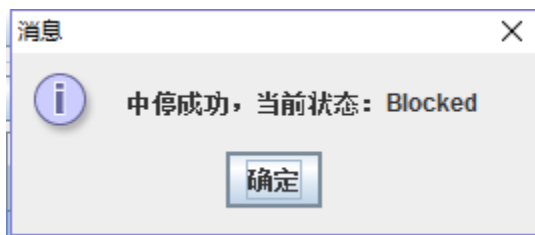
注：1.“所有列车计划”：查看当前计划集中的所有列车计划。

2.“当前位置列车”：只需在起始站处输入位置，就可查看该位置的列车表。

- 3.“当前资源列车”：输入对应的车厢编号，即可查看该车厢的使用情况。
- 4.“当前列车状态”：输入列车号和出发时间即可查看该列车当前状态。
- 5.“新增列车”：除计划项集不需要输入，其它都得输入。
- 6.“分配/添加资源”：输入列车号，车厢编号，出发时间即可为该趟列车分配/添加对应编号的车厢。
- 7.“启动当前列车”：输入航班号和起飞时间，点击即可启动。
- 8.“中途停车”：输入列车编号和出发时间，点击按钮，若该列车还有中间站且已启动，即可停车。
- 9.“结束当前列车”：类比 7。
- 10.“查看冲突”：
- 11.“更改计划项集”：在左侧选择一个计划项集（文件），点击即可。
- 12.“检查矛盾”：查看是否存在资源抢占矛盾。
- 13.“前序计划项”：输入列车号，车厢编号，出发时间，以此搜索计划项集合中与输入列车使用同一资源且时间在前面的计划项。
- 14.“管理资源”：可增加（删除、修改）当前计划项集中的资源。
- 15.“管理位置”：同 12。
- 16.“全部间站”：输入列车号和出发时间，点击按钮即可查看该趟列车途经的中间站。

可以看到新增了一个中途停车的功能和一个查看全部站点的功能。下面就介绍一下这两个功能：

1. 中途停车：输入列车编号和出发时间，点击按钮，若该列车还有中间站且已启动，即可停车。



2. 输入列车号和出发时间，点击“全部间站”即可查看当前列车的全部站点（这一功能是为了弥补 Board 无法显示所有站点的缺陷）：



3.12.3 课表应用

与前面两个应用大同小异：



同样，红字部分是说明，具体为：

- 1.“所有课程”：查看当前计划集中的所有课程。
- 2.“当教师课程”：只需在上课教室处输入位置，就可查看该教室的课程表。
- 3.“当前教师课程”：输入对应的教师编号，即可查看该教师当前是否在上课。已有的 9 个教师的编号为 001-009。
- 4.“当前课程状态”：输入课程名称和开始时间即可查看该课程当前状态。
- 5.“新增课程”：除计划项集不需要输入，其它都得输入。
- 6.“分配资源”：输入课程名称，教师编号，开始时间即可为该课程分配对应编号到教师。
- 7.“启动当前课程”：输入课程名称和开始时间，点击即可启动。
- 8.“结束当前航班”：类比 7。
- 9.“查看冲突”：
- 10.“更改计划项集”：在左侧选择一个计划项集（文件），点击即可。
- 11.“检查矛盾”：查看是否存在资源抢占矛盾。
- 12.“前序计划项”：输入课程名称，教师编号，开始时间，以此搜索计划项集中与输入课程使用同一资源且时间在后面的计划项。
- 13.“管理资源”：可增加（删除、修改）当前计划项集中的资源。
- 14.“管理位置”：同 12.

这个应用的功能基本与航班一致，就不再赘述了。

3.13 基于语法的数据读入

这一功能上面已经介绍过了，即下图中的红框圈起来的部分：

3.14 应对面临的新变化

只考虑你选定的三个应用的变化即可。

3.14.1 变化 1

评估之前的设计是否可应对变化、代价如何

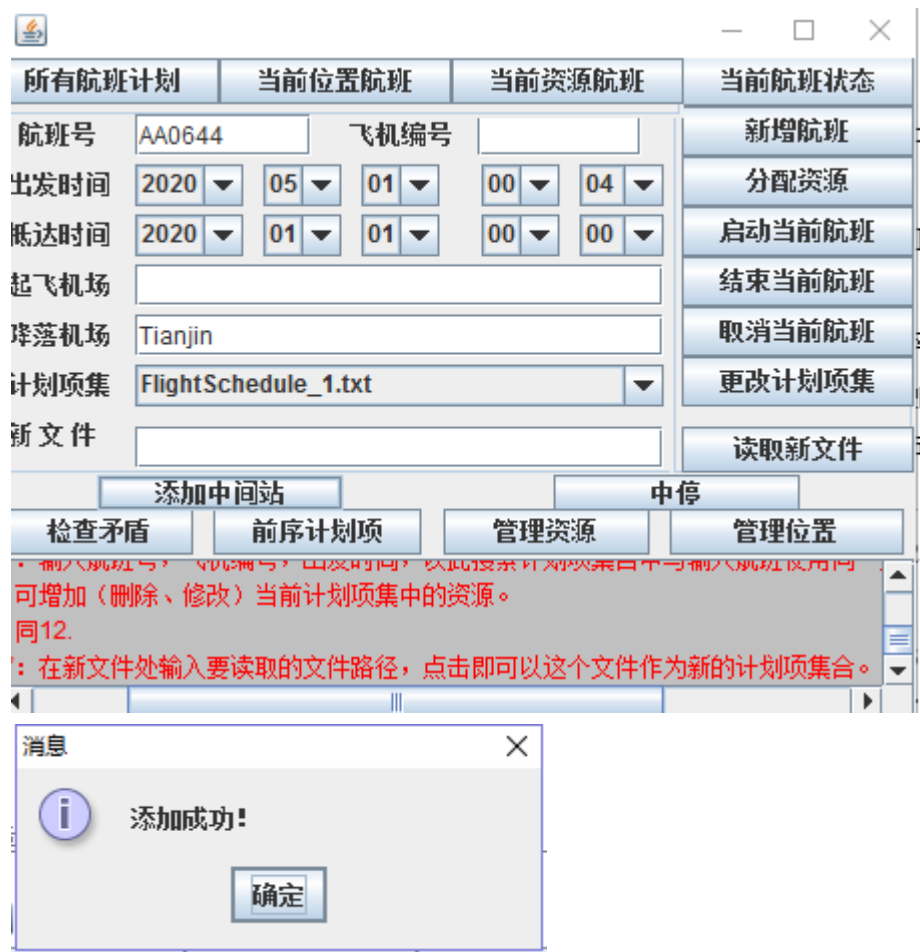
如何修改设计以应对变化

可以应变，不过代价不小：

1. 在 `BlockableEntryImpl` 中增加了一个属性 `blockNum` 来记录停止的次数。
2. 在 `FlightEntry` 中添加了 `BlockableEntry` 接口，增加了一个存放中间站的 `rep` 及其 `getter` 和 `setter`，增加了一个 `block` 方法
3. 在 `FlightAPP` 中增加了两个功能：给某个航班添加中间站，在中间站降落。

以下是结果：

- 1) 给某个航班添加中间站：输入航班号、起飞时间，并在降落机场输入要添加的中间站，点击添加中间站（输入为空，或输入的位置与飞机起止点相同均不能通过，需重新输入）：



2) 在中间站降落：在给航班添加中间站后，启动航班，然后才能中停，且只能停止一次，若无中间站，则不能停止：



3.14.2 变化 2

在 TrainEntry 中重写了 Cancel()方法，具体为：

```

@Override
public boolean Cancel() {
    if(msre.getResources() != null) {
        JOptionPane.showMessageDialog(null, "中已分配资源，不可取消！" + (msre.getResources().size()-2));
        return false;
    }
    else {
        super.Cancel();
        return true;
    }
}

```

这样就保证了在已经分配车厢后就把你取消了。

显然，之前的设计可适应此变化，且代价很小，不过几行代码。

3.14.3 变化 3

1. 简单修改了 `SingleSortedResourceEntryImpl`，使其能多次赋值。

1. 在 `CourseEntry` 中增加了一个 `rep` 来存储多位教师。增加了一个 `addTeacher` 方法和一个 `changeTeacher` 方法。

2. 在 APP 中增加了两个功能：给某个计划项添加教师，切换教师。

之前的设计可应对变化，代价可以接受。

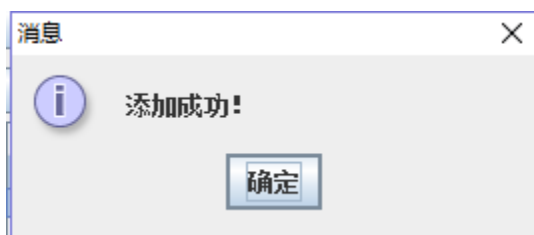
以下是结果：

- 1) 添加教师：为 2020-02-24 日 15:45 开始的近世代数课程添加一个教师，如下图所示，填写课程名称，开始时间，与及要添加的教师编号（若无该编号教师，请在管理资源中添加教师后再试），点击“课程添加教师”：

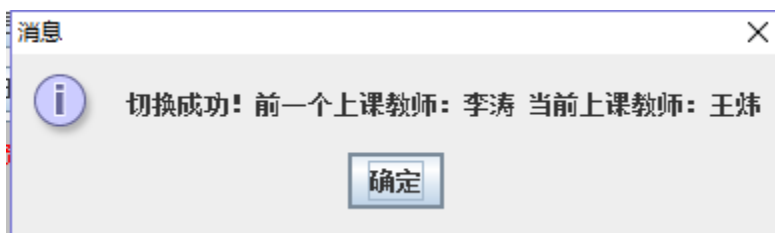
12. “前序计划项”：输入课程名称，教师编号，开始时间，以此搜索计划项集合中与输入项相同的计划项。

13. “管理资源”：可增加（删除、修改）当前计划项集中的资源。

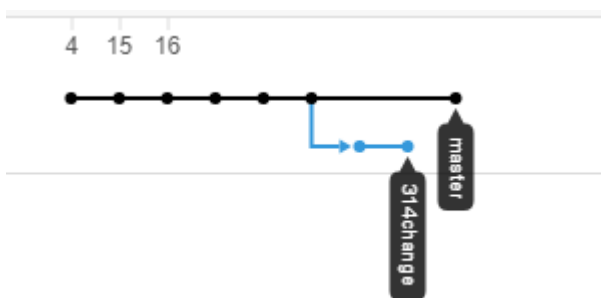
14. “管理位置”：同12.



- 2) 切换教师（是按顺序切换的，顺序即优先级）：在给课程添加教师后且课程当前状态为 **Running** 时，点击切换教师即可：



3.15 Git 仓库结构



4 实验进度记录

日期	时间段	计划任务	实际完成情况
2020/4/22	20:00-22:00	根据方案五整体构思 PlanningEntry	按时完成
2020/4/23	13:30-15:30	Timeslot+Resource	按时完成
2020/4/24	17:30-18:30	实现接口 :位置的数量+是否可阻塞及其时间描述	按时完成
2020/4/28	19:30-22:00	实现总体构架中的其它接口	按时完成
2020/4/30	20:30-21:30	FlightBoard	按时完成
2020/5/1	14:30-16:30	comparator,工厂方法, board	超时半小时
2020/5/1	18:30-20:40	航班信息正则表达式编写	按时完成
2020/5/2	13:30-16:30	FlightCollection,FlightFactory,迭代器,航班信息正则表达式修改	按时完成
2020/5/3	19:00-22:00	FlightBoard 改进+获取当前时间	延时 1 小时
2020/5/4	19:30-21:00	一些修改	按时完成

2020/5/5	16:00-18:00	Flightapp 框架	按时完成
2020/5/6	19:30-21:15	状态模式+Flightapp 部分功能	按时完成
2020/5/7	12:00-15:30	Flightapp 基本完成	延时 1 小时
2020/5/7	21:00-24:00	Flightapp 细节完善 +planningEntryAPIs+ 正则表达式完善	延时 1 小时
2020/5/9	12:00-13:50	debug Flightapp	按时完成
2020/5/10	12:30-17:00 19:00-21:30	TrainApp	延时 2 小时
2020/5/12	17:30-18:30 20:50-23:20	CourseApp	按时完成
2020/5/14	13:30-15:00 19:00-21:30	debug+状态模式	延时 1 小时
2020/5/14	14:30-15:30 21:00-22:40	debug+test/2	按时完成

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
正则表达式不会写	上网查阅学习
JSwing/Jtable 不会	上网查阅学习

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验教训

- 1.接口里面共性的方法处理的不够全面，会导致复用的时候变得很容易出问题。
- 2.处理好复用性写应用时会很方便。
- 3.各种模式的应用，使得程序变得更好。

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？

答：面向 ADT 的编程复用性很高，便于后序修改，不止要当下做好，还要着眼于“未来”，良好的 ADT 为编程带来便利和优化。

- (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？

答：为 ADT 撰写复杂的 specification, invariants, RI, AF，时刻注意 ADT 是否有 rep exposure，让我们编写 ADT 时思路更加清晰，且编写出来的 ADT 使用起来更安全，更方便。愿意在以后的编程中坚持做下去。

- (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？

答：难处就在于要需要有很好的策划才能使得自己编写出来的 ADT 复用性高，使用方便、安全。乐趣也在于这个策划，策划的不好，后期处理起来手忙脚乱，策划好，编写起来很舒服，用起来也很舒服。

- (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？

答：设计模式虽然增加了很多类，但带来的好处也是很多的，例如 State 模式使得复用性更高，结构也清晰，再如装饰模式，使得 ADT 能具有各种不同的性质，还有策略模式，针对一个问题，提供不同的策略，这样在使用时可以根据实际情况选择不同的策略来处理问题，效益是很高的...

- (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？

答：便于大规模处理，使用起来很方便，语法也不难，很容易掌握。

- (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？

答：最初时在提取公共方法到接口的时候没太重视，使得后期在有的地方复用起来不方便。这时候要改的话就得改很多，还容易错，所以有的地方选择了向下转型。

- (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？
- 答：抽象程度越高，复用程度也越高。为了最大程度的抽象和复用，设计的时候尽量维持单一职责原则，这样便于复用，还有 LSP 原则，这也是维持复用性的一个很重要的地方。
- (8) 关于本实验的工作量、难度、deadline。
- 答：说实话，这实验对我来说还是很有挑战性的，由于前期考虑的不周，给我后期使用带来了很多麻烦，使得工作量变大，不过五周的时间还是差不多够用的。
- (9) 到目前为止你对《软件构造》课程的评价。
- 很有用的一门课。