

**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 | 刘璟烁 |
| 学号 | 1180500301 |
| 班号 | 1803001 |
| 电子邮件 | 907599581@qq.com |
| 手机号码 | 13102886813 |

**目录**

[1 实验目标概述 1](#_Toc37326126)

[2 实验环境配置 1](#_Toc37326127)

[3 实验过程 1](#_Toc37326128)

[3.1 待开发的三个应用场景 1](#_Toc37326129)

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

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

[3.2.2 局部共性特征的设计方案 2](#_Toc37326132)

[3.2.3 面向各应用的PlanningEntry子类型设计（个性化特征的设计方案） 2](#_Toc37326133)

[3.3 面向复用的设计：R 2](#_Toc37326134)

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

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

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

[3.7 面向应用的设计：Board 2](#_Toc37326138)

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

[3.9 PlanningEntryCollection的设计 2](#_Toc37326140)

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

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

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

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

[3.11 设计模式应用 2](#_Toc37326145)

[3.11.1 Factory Method 3](#_Toc37326146)

[3.11.2 Iterator 3](#_Toc37326147)

[3.11.3 Strategy 3](#_Toc37326148)

[3.12 应用设计与开发 3](#_Toc37326149)

[3.12.1 航班应用 3](#_Toc37326150)

[3.12.2 高铁应用 3](#_Toc37326151)

[3.12.3 进程应用 3](#_Toc37326152)

[3.12.4 课表应用 3](#_Toc37326153)

[3.12.5 学习活动应用 3](#_Toc37326154)

[3.13 基于语法的数据读入 3](#_Toc37326155)

[3.14 应对面临的新变化 3](#_Toc37326156)

[3.14.1 变化1 3](#_Toc37326157)

[3.14.2 变化2 4](#_Toc37326158)

[3.14.3 变化3 4](#_Toc37326159)

[3.15 Git仓库结构 4](#_Toc37326160)

[4 实验进度记录 4](#_Toc37326161)

[5 实验过程中遇到的困难与解决途径 4](#_Toc37326162)

[6 实验过程中收获的经验、教训、感想 5](#_Toc37326163)

[6.1 实验过程中收获的经验和教训 5](#_Toc37326164)

[6.2 针对以下方面的感受 5](#_Toc37326165)

# 实验目标概述

继Lab2实验中对ADT和OOP规范的训练，本次试验目标是编写具有更好的可复用性和可维护性的软件，并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、组合等，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复用等。

本次实验给通过 ADT 和泛型等抽象技术，为给定的五个应用开发一套可复用的 ADT 及其实现，并需要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)。 同时也会涉及对外部API的调用等技巧。

# 实验环境配置

过程：

本次试验的工具基本在前几次实验过程中已经准备的比较到位了，在实验过程中，我在设计测试用例时为了判断控制台输出是否正确，引入了外部jar包，（通过在project的properties中，加入额外的类库）。

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

https://github.com/ComputerScienceHIT/Lab3-1180500301

# 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！）。

## 待开发的三个应用场景

(1).选定的三个应用：

**航班管理、高铁车次管理、大学课表管理**。

(2).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。

**共性**：三个应用均有未分配资源、已分配资源、已启动、已完成、已取消等状态，且均有代表特定计划项的名称等；另外，三者均拥有结构属性相似的board。

**差异**：使用不同的资源R，地点数量、类型不同，时间对数量的类型不同等。

## 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。

### PlanningEntry<R>的共性操作

(1)获取计划项的名字:

/\*\*

\* get the name of the Entry

\* **@return**,the name of the entry

\*/

public String getName();

(2).启动一个任务项:

/\*\*

\* make the entry running

\*/

public  boolean  run();

(3).结束一个任务项:

/\*\*

\* end the entry

\*/

public boolean end();

(4)取消一个任务项:

/\*\*

\* cancel the entry

\*/

public boolean cancel();

(5):获取当前状态

/\*\*

\* get the state of the entry

\* **@return**,the state of the entry

\*/

public State getState();

### 局部共性特征的设计方案

1. **.抽象类CommonPlanningEntry<R>:**

**A.设计思想**：本抽象类主要目的是实现部分上层接口的函数（比如各种state的改变等），并新增一些共性的方法（比如强行显示设置一个状态等）。

**B.实现**：

表示变量：

private State state;//用于记录计划项当前状态

private final String name;//计划项的名字

方法：

首先实现planningEntry中的所有方法：

run()、end()、cancel()等方法的实现直接委托给变量state来实现，getState()、getName()直接返回相应的变量即可（因为成员变量都是不可变类型）。

其次新增强行设置状态的方法：

/\*\*

\* 设置该对象的状态

\* @param state ，要被设置成的状态类型

\*/

public void setState(State state)

1. **.BlockableEntry接口：**

**设计思想及实现方法**：由于阻塞计划项这一功能不是所有计划项的共性特征，所以将其相关的方法单独放到一个接口中，有需求的计划项至于要实现这个接口即可。

接口中只有两个方法：

A.根据时间要求阻塞计划项

/\*\*

\* block the entry within a timeSlot

\* @param atTime,the timeSlot that the entry should be blocked

\*/

public boolean blockTheEntry(Calendar atTime);

B.根据时间要求重启计划项

/\*\*

\* restart the entry

\* @param atThisTime,the time that the entry will be restarted

\*/

public boolean restart(Calendar atThisTime);

考虑到实验对其应用较少以及实现所需要的变量较多造成的复杂性问题，只需在TrainEntry中实现即可，不需单独构造一个实现类。

1. **.resource控制类型:**

A.接口SingleResourceEntry：用于控制只含有一个资源对象的计划项，应用CRP，通过对接口构造一个实现类SingleResourceEntryImple，用来将计划项设置、获取自身资源的功能委托给该类的对象。

a.接口方法：

/\*\*

\* 将对象管理的resource设为该resource

\* @param thisresource,被管理的对象

\*/

public boolean setResource(Resource thisresource);

/\*\*

\* 返回对象管理的resource

\* @return 被管理的对象

\*/

public Resource getResource();

b.实现类中的成员变量以及方法的实现：

成员变量：private Resource thisResource;（用于记录该对象所管理的资源对象）

方法的实现方案：由于Resource类的对象为不可变类型，setResource()和getResource()只需要设置、返回成员变量即可。

B.接口MutipleSortedResourceEntry及其实现类：用于控制含有多个可排序资源对象的计划项，应用CRP，通过对接口构造一个实现类MutipleSortedResourceEntryImple，用来将计划项设置、获取自身资源的功能委托给该类的对象。

a.接口方法：

/\*\*

\* 设置被该对象管理的资源序列

\* @param thisresource ,被管理的资源序列

\*/

public boolean setResource(List<? extends Resource> thisresource);

/\*\*

\* 获得被该对象管理的资源序列

\* @return 被管理的资源序列

\*/

public List<? extends Resource> getResource();

b.实现类中的成员变量以及方法的实现：

成员变量：private List<? extends Resource> thisResource;

（同用于记录该对象所管理的资源对象）

方法的实现方案：setResource()和getResource()设置或者返回的对象为一个list可能被改动，因此实现方法时设置或者返回一个相应list对象的复制品即可。

1. **.与location数量相关的设计方案**：
2. SingleLocationEntry接口及其实现类：用于控制只含有一个位置对象的计划项（比如课程计划项），应用CRP，通过对接口构造一个实现类SingleLocationEntryImple，用来将计划项设置、获取自身位置对象的功能委托给该类的对象。

a.接口方法：

/\*\*

\* 设置被管理的位置

\* @param thisLocation

\*/

public boolean setLocation(Location thisLocation);

/\*\*

\* 获取被管理的位置

\* @return 被管理的位置

\*/

public Location getLocation();

b.实现类中的成员变量以及方法的实现：

成员变量：private Location thisLocation;（用于记录该对象所管理的位置对象）

方法的实现方案：由于Location 类的对象为不可变类型，setLocation()和getLocation()只需要设置、返回其成员变量即可。

1. DoubleLocationEntry接口及其实现类：用于控制含有两个位置对象的计划项（比如航班计划项），应用CRP，通过对接口构造一个实现类DoubleLocationEntryImple，用来将计划项设置、获取自身位置对象的功能委托给该类的对象。

a.接口方法：

/\*\*

\* set the two orderly locations that will be controlled by this object

\* @param fromLoc,the first location

\* @param toLoc,the second location

\*/

public boolean setLocation(Location fromLoc,Location toLoc);

/\*\*

\* get the first location

\* @return,the first location

\*/

public Location getfromLocation();

/\*\*

\* get the second location

\* @return,the second location

\*/

public Location getToLocation();

b.实现类中的成员变量以及方法的实现：

成员变量：（用于记录该对象所管理的位置对象）

private Location fromLocation;

private Location toLocation;

方法的实现方案：由于Location 类的对象为不可变类型，setLocation()和getLocation()只需要设置、返回其成员变量即可。

1. MultipleLocationEntry接口及其实现类：用于控制含有多个有序位置对象的计划项（比如列车计划项），应用CRP，通过对接口构造一个实现类MultipleLocationEntryImple，用来将计划项设置、获取自身位置对象列表的功能委托给该类的对象。

a.接口方法：

/\*\*

\* set a list of locations to be controlled by this object

\* @param locaitonList, the locations

\*/

public boolean setLocationList(List<? extends Location> locaitonList);

/\*\*

\* 获取位置列表

\*/

public List<? extends Location> getLocationList();

b.实现类中的成员变量以及方法的实现：

成员变量：（用于记录该对象所管理的位置对象）

private List<? extends Location> locationList;

方法的实现方案：setLocationList()和getLocationList()设置或者返回的对象为一个list可能被改动，因此实现方法时设置或者返回一个相应list对象的复制品即可。

1. .与location能否改变相关的接口和类：
2. LocationCouldChange接口及其实现类：用于为只含一个位置对象的计划项（比如课程计划项）提供改变位置的功能，应用CRP，通过对接口构造一个实现类LocationCouldChangeImple，

**不过这里控制的是控制计划项位置对象的对象（即上面提到的SingleLocationEntry类型的对象），调用该对象带有的相应方法实现位置的更改。**

a.接口方法：

/\*\*

\* change the location

\* @param thisLocation , the location to be changed to

\*/

public void changeSingleLocation(Location thisLocation);

b.实现类中的成员变量以及方法的实现：

成员变量：private SingleLocationEntry singleLoc;（用于记录相应的**管理位置对象的对象**）

方法的实现方案：为实现changeSingleLocation（Location thisLocation）方法，只需要调用成员变量的setLocation（）方法即可。

1. .关于管理时间对数量的共性设计：

A.SingleSlot接口及其实现类：用于为只含一个时间对的计划项（比如课程、航班计划项）提供设置时间对的功能，应用CRP，通过对接口构造一个实现类SingleSlotimple，用来将计划项设置、获取自身时间对对象的功能委托给该类的对象。

a.接口方法：

/\*\*

\* set the management timeslot of this object

\* @param timeSlot

\*/

public void setSlot(TimeSlot timeSlot);

/\*\*

\* 返回该对象管理的时间对

\* @return the timeSlot which is being managed by this manager

\*/

public TimeSlot getSlot();

b.实现类中的成员变量以及方法的实现：

成员变量：private TimeSlot singleSlot;（用于记录相应的时间对对象）

方法的实现方案：由于TimeSlot类型的对象为不可变的，因此只需要进行相应的成员变量的设置即返回即可。

B.MutiSlot接口及其实现类：用于为含有多个有序时间对的计划项（比如列车计划项）提供设置时间对的功能，应用CRP，通过对接口构造一个实现类MutiSlotImple，用来将计划项设置、获取自身时间对列表对象的功能委托给该类的对象。

a.接口方法：

/\*\*

\* 设置该对象管理的时间对序列

\* @param timeSlot,被管理的时间对

\*/

public void setSlot(List<TimeSlot> timeSlot);

/\*\*

\* get the list of TimeSlot that this manager have

\* @return,the TimeSlot list

\*/

public List<TimeSlot> getSlotList();

b.实现类中的成员变量以及方法的实现：

成员变量：private List<TimeSlot> mutiSlot = new ArrayList<>();（用于记录相应的时间对列表对象）

方法的实现方案：setSlot() 、getSlotList()设置或者返回的对象为一个list可能被改动，因此实现方法时设置或者返回一个相应list对象的复制品即可。

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

3.2.3.1 FlightEntry及其接口FlightPlanningEntry:

**设计思想：**

首先考虑FlightPlanningEntry的设计，即航班计划项的接口。由于航班计划项具有单一资源，两个位置，单一时间对的特性，因此该接口需要继承SingleSlot,SingleResourceEntry,DoubleLocationEntry三个接口，从而使具体类具有相应的功能，另外，由于该计划项所需要的方法在各个接口中都有所实现，因此未在FlightPlanningEntry中添加额外的方法。

其次，考虑具体的计划项类FlightEntry的设计，由于上面提到的CommonPlannningEntry<R>中已经实现了作为计划项所需要的状态、名称相关的基本功能，因此可以让FlightEntry直接继承CommonPlannningEntry，并将R改为相应的Plane资源类型（下面会说到）。另外，关于成员变量，需要设置接口中提到的几个类的对象来管理该计划项的资源位置等属性；剩下的只需要实现接口相应的方法即可。

**实现方法：**

A.**成员**变量(负责管理资源位置时间对的对象)：

private final SingleResourceEntry singleRes;

private final DoubleLocationEntry doubleLoc;

private final SingleSlot singleSlot;

文档说明：

// Abstraction function:

//AF(singleRes,doubleLoc,singleSlot) = the manager of the resources of the object, the manager of the locations of the object,the manager of the timeSlots of the object,

//Representation invariant:

//    the field must be non null

//Safety from rep exposure:

//      all fields are private and final

// all the Observer return an immutable object

B.方法实现方案（以下方法均为对相应接口的实现）：

public boolean setResource(Resource thisresource)：

该方法为计划项分配资源。首先检查参数的类型是否为Plane类型，若否报错退出。另外，方法涉及到状态的改变，因此先尝试调用父类方法：

super.allocateResource() 返回boolean值判断当前状态能否进行资源分配：若可以，则将设置资源对象的操作委托给成员变量singleRes去做；否则，返回相应错误信息。

public Resource getResource()：

该方法用于获取计划项所使用的资源。直接委托给singleRes调用同名方法即可。

public boolean setLocation(Location fromLocation, Location toLocation)：

方法用于设置计划项位置。检查参数类型是否为Airport（后面会提到的机场位置类型），若否报错推出；否则调用doubleLoc的同名方法进行委托。

public Location getfromLocation() 和public Location getToLocation()：返回计划项起飞和降落机场位置，调用doubleLoc的同名方法进行委托即可。

public void setSlot(TimeSlot timeSlot)及public TimeSlot getSlot()：设置和返回航班计划项的起飞降落时间对。调用singleSlot的同名方法进行委托即可。

public boolean run()：航班起飞。先调用父类方法：super.run()，根据返回值判断当前状态是否可进行该操作，若是输出相应个性化语句并返回true；否则报错并返回false。

public boolean end()：航班降落。同样是调用父类方法：super.end()，根据返回值判断当前状态是否可进行该操作，若是输出相应个性化语句并返回true；否则报错并返回false。

public boolean cancel()实现与前两者相同。

3.2.3.2 CourseEntry及其接口CoursePlanningEntry:

设计思想：

首先考虑CoursePlanningEntry的设计，即课程计划项的接口。由于课程计划项具有单一资源，单一位置，位置可更改，单一时间对的特性，因此该接口需要继承SingleSlot,LocationCouldChange,SingleLocationEntry,

SingleResourceEntry接口，从而使具体类具有相应的功能，另外，由于该计划项所需要的方法在各个接口中都有所实现，因此未在CoursePlanningEntry中添加额外的方法。

其次，考虑具体的计划项类CourseEntry的设计，像FlightEntry一样直接继承CommonPlannningEntry，并将R改为相应的Teacher资源类型（下面会说到）。关于成员变量，同FlightEntry一样需要设置接口中提到的几个类的对象来管理该计划项的资源位置等属性；剩下的只需要实现接口相应的方法即可。

实现方法：

A.成员变量(负责管理资源位置时间对的对象)：

private final LocationCouldChange changeObject;

private final SingleLocationEntry singleLoc;

private final SingleResourceEntry singleRes;

private final SingleSlot  singleSlo;

文档说明：  
// Abstraction function:

//AF(changeObject,singleLoc,singleRes，singleSlo) = the object to change the location ,the manager of the locations of the object,the manager of the resources of the object, the manager of the timeSlots of the object,

//Representation invariant:

//   the field must be non null

//Safety from rep exposure:

//     all fields are private and final

// all the Observer return an immutable object

B.方法实现方案（以下方法均为对相应接口的实现）：

构造器：

public CourseEntry(LocationCouldChange changeObject, SingleLocationEntry singleLoc,SingleResourceEntry singleRes, SingleSlot singleSlo,String name)：调用父类构造器并将name传入，其余的用于设置成员

public boolean setResource(Resource thisresource)：

该方法为计划项分配资源。首先检查参数的类型是否为Plane类型，若否报错退出。另外，方法涉及到状态的改变，因此先尝试调用父类方法：

super.allocateResource() 返回boolean值判断当前状态能否进行资源分配：若可以，则将设置资源对象的操作委托给成员变量singleRes去做；否则，返回相应错误信息。

public Resource getResource()：

该方法用于获取计划项所使用的资源。直接委托给singleRes调用同名方法即可。

public boolean setLocation(Location fromLocation, Location toLocation)：

方法用于设置计划项位置。检查参数类型是否为Classroom（后面会提到的机场位置类型），若否报错推出；否则调用singleLoc的同名方法进行委托。

public Location getLocation() ：方法返回计划项位置。调用singleLoc的同名方法进行委托即可。

public void setSlot(TimeSlot timeSlot)及public TimeSlot getSlot()：设置和返回课程计划项的上课下课时间对。调用singleSlot的同名方法进行委托即可。

public boolean run()：上课。先调用父类方法：super.run()，根据返回值判断当前状态是否可进行该操作，若是输出相应个性化语句并返回true；否则报错并返回false。

public boolean end() 下课。同样是调用父类方法：super.end()，根据返回值判断当前状态是否可进行该操作，若是输出相应个性化语句并返回true；否则报错并返回false。

public boolean cancel()实现与前两者相同。

public void changeSingleLocation(Location thisLocation)：变更该计划项的位置，首先为成员变量changeObject设置**需要管理的管理计划项位置的对象，即**singleLoc，随后委托changeObject调用相应方法即可。

3.2.3.3 TrainEntry及其接口TrainPlanningEntry:

**设计思想：**

首先考虑TrainPlanningEntry的设计，即列车计划项的接口。由于列车计划项具有多个可排序资源，多个位置，多时间对和可以阻塞的特性，因此该接口需要继承MutiSlot，MultipleLocationEntry, MutipleSortedResourceEntry，BlockableEntry接口，从而使具体类具有相应的功能，另外，由于该计划项所需要的方法在各个接口中都有所实现，因此未在TrainPlanningEntry中添加额外的方法。

其次，考虑具体的计划项类TrainEntry的设计，像前两个计划项一样直接继承CommonPlannningEntry，并将R改为相应的Coach资源类型（下面会说到）。关于成员变量，同前两个计划项一样需要设置接口中提到的几个类的对象来管理该计划项的资源位置等属性；剩下的只需要实现接口相应的方法即可。

**实现方法：**

A.成员变量(负责管理资源位置时间对的对象)：

private final MutipleSortedResourceEntry mutiSorResource ;

private final MultipleLocationEntry multiLocation;

private final MutiSlot mutiSlo;

文档说明：

// Abstraction function:

// AF(mutiSorresource,multiLocation,mutiSlo) = the manager of the resources of the object, the manager of the locations of the object,the manager of the timeSlots of the object,

// Representation invariant:

// the field must be non null

// Safety from rep exposure:

// all fields are private

// all the Observer return an immutable object

B.方法实现方案（以下方法均为对相应接口的实现）：

构造器：

public TrainEntry(MutipleSortedResourceEntry mutiSorResource,MultipleLocationEntry multiLocation,MutiSlot mutiSlo,String name)：调用父类构造器并将name传入，其余的用于设置成员。

**public boolean setResource(List<? extends Resource> thisresource)：**

该方法为计划项分配资源。首先检查参数列表中对象的类型是否为Coach类型，若否报错退出。另外，方法涉及到状态的改变，因此先尝试调用父类方法：

super.allocateResource() 返回boolean值判断当前状态能否进行资源分配：若可以，则将设置资源对象的操作委托给成员变量mutiSorResource 去做；否则，返回相应错误信息。

**public List<? extends Resource> getResource()：**

该方法用于获取计划项所使用的资源。直接委托给mutiSorResource 调用同名方法即可。

**public boolean setLocationList(List<? extends Location> locaitonList)：**

方法用于设置计划项位置。检查参数列表中对象类型是否为TrainStation（后面会提到的机场位置类型），若否报错推出；否则调用multiLocation的同名方法进行委托。

**public List<? extends Location> getLocationList()**：方法返回计划项位置。调用multiLocation的同名方法进行委托即可。

**public void setSlot(TimeSlot timeSlot)及public TimeSlot getSlot()**：设置和返回**列车始发车以及到达终点站的时间对**，区分于下面提到的设置和返回时间对序列，下面提到的时间对序列用于描述在每一个途经站的停车和启动时间对，调用mutiSlo的同名方法进行委托即可。

public void setSlot(List<TimeSlot> slotList)和public List<TimeSlot> getSlotList() ：用于设置和返回在每一个途经站的停车和启动时间对。调用mutiSlo的同名方法进行委托即可。

**public boolean run()**：列车出发。先调用父类方法：super.run()，根据返回值判断当前状态是否可进行该操作，若是输出相应个性化语句并返回true；否则报错并返回false。

**public boolean blockTheEntry(Calendar atTime)：**

在途经站停留的期间挂起计划项。首先判断参数是否处于途径站时间对列表中的某一时间对的时间范围内，若是，调用getState（）方法获取当前状态，并通过其调用block（）尝试进行阻塞；否则报错返回。

public boolean restart(Calendar atThisTime)：

用于在途经站停留的期间挂起后重启计划项。首先判断参数是否等于途径站时间对列表中的某一时间对的截止时间，若是，调用父类的run（），否则报错返回。

public boolean end()到终点站。同样是调用父类方法：super.end()，根据返回值判断当前状态是否可进行该操作，若是输出相应个性化语句并返回true；否则报错并返回false。

public boolean cancel()实现与前两者相同。

## 面向复用的设计：R

**3.3.1资源类相关接口：Resource**

接口中仅含有一个共性方法：

/\*\*

\* get the ID of the resource

\* @return,the ID of it

\*/

**public String getID();**

**3.3.2航班计划项资源：Plane**

**A.设计思想**：根据指导书的要求，Plane类型为不可变类型，每个对象具有特定的几个属性（如ID、座位数等）。因此，只需要按要求为其配有特定的成员变量，并且实现基本的getter方法和equals()等方法即可。

B.实现方法：

成员变量：

private final String ID;

private final String type;

private final int seats;

private final double age;

相关文档：

// Abstraction function:

// AF(ID,type,seats,age) = the ID of the plane,what type is the plane,how many seats does the plane have,the age of the plane

// Representation invariant:

// the field must be non null,seats > 0 ,age >=0

// Safety from rep exposure:

//  all fields are private and final

// the Observer return an immutable

方法实现：

构造器：**public Plane(String ID, String type, int seats, double age)**：设置相关成员变量即可。

**public String getID()**：返回成员变量ID即可（因为其为String类型不可变）。

**public String toString()**：组合返回四个成员变量的String形式即可。

**equals()和hashCode()：**使用eclipse提供的实现。

**3.3.3课程计划项资源：Teacher**

**A.设计思想**：根据指导书的要求，Teacher类型仍为不可变类型，且每个对象具有特定的几个属性（如ID、姓名等）。因此，需要按要求为其配有特定的成员变量，并且实现基本的getter方法和equals()等方法，同时，由于教师的名称信息也较常使用，需要为其多添加一个getName（）的方法。

B.实现方法：

成员变量：

private final String ID;

private final String name;

private final String gender;

private final String title;

相关文档：

// Abstraction function:

// AF(ID,name,gender,title) = the ID of the teacher,the name of the teacher,the gender of the teacher,teacher's title

// Representation invariant:

// the field must be non null,gender equals male or female

// Safety from rep exposure:

//  all fields are private and final

// the Observer return an immutable

方法实现：

构造器：public Teacher(String ID, String name, String gender, String title)：设置相关成员变量即可。

/\*\*

\* 获取授课教师的姓名

\* @return 授课教师的姓名

\*/

**public String getname()**：返回成员变量name。

**public String getID()**：返回成员变量ID即可（因为其为String类型不可变）。

**public String toString()**：组合返回四个成员变量的String形式即可。

**equals()和hashCode()：**使用eclipse提供的实现。

**3.3.3列车资源：Coach**

1. **设计思想**：根据指导书的要求，Coach类型仍为不可变类型，且每个对象具有特定的几个属性（如ID、座位数等）。因此，需要按要求为其配有特定的成员变量，并且实现基本的getter方法和equals()等方法。
2. 实现方法：

成员变量：

private final String ID;

private final String type;

private final int seats;

private final int birthYear;

// Abstraction function:

// AF(ID,type,seats,birthYear) = the ID of the coach,what type is the coach,how many seats does the coach have,when did it born

// Representation invariant:

// the field must be non null,seats >0;birthYear > 1900

// Safety from rep exposure:

//  all fields are private and final

// the Observer return an immutable

方法实现：

构造器：public Coach(String iD, String type, int seats, int birthYear)：设置相关成员变量即可。

**public String getID()**：返回成员变量ID即可（因为其为String类型不可变）。

**public String toString()**：组合返回四个成员变量的String形式即可。

**equals()和hashCode()：**使用eclipse提供的实现。

## 面向复用的设计：Location

3.4.1 **接口Location：**

**设计思想：**根据指导书的要求，所有的位置类型的对象都具有名字、经纬度、可否共享等属性，因此，在接口中设置这几个属性的getter即可。

**实现：**

/\*\*

\* get the name of this location

\* @return the name of this location

\*/

public String getName();

/\*\*

\* get whether it could share

\* @return whether it could share

\*/

public boolean whetherCouldShare();

/\*\*

\* get the latitude of the location

\* @return the latitude of the location

\*/

public double getLatitude();

/\*\*

\* get the longitude of the location

\* @return the longi

\* tude of the location

\*/

public double getLongitude();

3.4.2 父类CommonLocation（实现Location接口）：

**设计思想**：

根据上面的说明可知，飞机、列车、课程计划性所需要的位置的属性基本都是相同的，仅仅有是否可共享的区别。因此可以设计一个父类，包含上述计划项的位置类型所涉及的属性，并实现相关getter，相关类通过继承该类，增强了可服用性。另外，如果以后做进程设置计划项的开发，只需要将该类对象作为委托而非父类即可，因此具有一定可扩展性。注意，该类为不可变类型。

**实现方法：**

成员变量：

private final String nameOfLocation;

private final boolean shareAavilable;

private final double latitude ;

private final double longitude;

相关文档：

// Abstraction function:

// AF(nameOfLocation,shareAavilable,latitude，longitude) = the name of the location,if the location is available to share,the latitude and the longitude of the location

//Representation invariant:

// the field must be non null,latitude and longitude must be in (-180,180]

//Safety from rep exposure:

// all fields are private and final

// all the Observers return an immutable object

方法实现**：**

**构造器：**

**public CommonLocation(String nameOfLocation, boolean shareAavilable, double latitude, double longitude)：设置相应成员变量即可。**

**public boolean whetherCouldShare()：判断位置是否可共享。返回相应变量即可。**

**public double getLatitude()：返回相应代表经度的成员变量即可。**

**public double getLongitude()：返回相应代表纬度的成员变量。**

**hashCode()和equals（）：使用eclipse自带实现即可。**

**3.4.3 航班计划项位置类型：Airport**

**设计：**继承CommonLocation。该类型的对象所需要的属性与其父类完全相同，唯一的特点是不可共享，我们可以在其构造器中解决这一问题：声明对象时仅需传入除了父类成员变量shareAavilable之外的参数，并且在构造器中调用父类构造器，并将shareAavilable对应参数显示设置为true。

**实现**：

**public Airport(String nameOfLocation, double latitude, double longitude) {**

**super(nameOfLocation, true, latitude, longitude);**

**}**

**3.4.4 课程计划项位置类型：ClassRoom**

**设计：**同样继承CommonLocation。该类型的对象所需要的属性与其父类完全相同，不可以共享，同样在其构造器中解决这一问题：声明对象时仅需传入除了父类成员变量shareAavilable之外的参数，并且在构造器中调用父类构造器，并将shareAavilable对应参数显示设置为false。

**实现**：

**public ClassRoom(String nameOfLocation, double latitude, double longitude) {**

**super(nameOfLocation, false, latitude, longitude);**

**}**

**3.4.5 列车计划项位置类型：TrainStation**

**设计：**同样继承CommonLocation。该类型的对象所需要的属性与其父类完全相同，特点是可以共享，同样在其构造器中解决这一问题：声明对象时仅需传入除了父类成员变量shareAavilable之外的参数，并且在构造器中调用父类构造器，并将shareAavilable对应参数显示设置为true。

**实现**：

**public TrainStation(String nameOfLocation, double latitude, double longitude) {**

**super(nameOfLocation, true, latitude, longitude);**

**}**

## 面向复用的设计：Timeslot

**设计思想：**

该类对象用于表示一个时间对，因此可以采用两个Calendar类型的成员变量表示分别表示其起止时间。其次，为了方便向外输出合法格式的时间对的字符串形式，可以在添加两个String类型的成员变量，用于保存说明书要求格式的时间字符串。另外，关于类的方法，除了实现相应成员变量的getter方法，还可以添加用于判断时间对是否重合的方法，以及判断某一时刻是否在该时间对之内的方法。该类属于不变型类。

**实现方法：**

成员变量：

private final String startTime;

private final String endTime;

private final Calendar from;

private final Calendar to;

相关文档：

// Abstraction function:

// AF(startTime,endTime,from,to) = the string format of the begin time, the string format of the end time,the real time of begin and end.

// Representation invariant:

// all fields must be non-null

// Safety from rep exposure:

//  the fields are private and final

// the Observers make defensive copy

各方法的实现方案：

构造器：

public TimeSlot(Calendar from,Calendar to)：防御式拷贝参数为成员变量赋值，另外还需在构造器内为另外的字符串成员变量赋值：

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

startTime = form.format(from.getTime());

endTime = form.format(to.getTime());

public String toString()：返回报告要求的格式的时间对字符串。

/\*\*

\* 获取约定字符串形式的开始时间

\* @return 字符串形式的开始时间

\*/

public String getStrBegin()：返回相应成员变量即可。

/\*\*

\* 获取约定字符串形式的结束时间

\* @return 字符串形式的结束时间

\*/

public String getStrEnd()；返回相应成员变量即可。

/\*\*

\* get the begin time of the slot

\* @return,the begin time of the slot

\*/

public Calendar getBeginTime()：对相应成员变量做防御使拷贝并返回。

/\*\*

\* get the end time of the slot

\* @return,the end time of the slot

\*/

public Calendar getEndTime()：对相应成员变量做防御使拷贝并返回。

/\*\*

\* 判断给定时间是否在该对象对应的时间对范围内,临界时不算在范围内

\* @param thisTime ,要用于判断的给定时间

\* @return true 若是，否则false

\*/

public boolean timeWithintheSLot(Calendar thisTime)：首先调用成员变量from的before（）和成员变量to的after（）方法判断参数是否完全在字符串之外。另外，我们还认为与起始或终止时间相同也要返回true，但是传入的参数往往是以分钟为设置单位的，Calendar对象的equals()是以毫秒为单位进行比较的，这一点我们可以将需要进行比较的Calendar对象均**转化为字符串的形式**从而判断是否相等。

/\*\*

\* 判断两个TimeSlot是否有重合部分 不包括临界相交

\* @param first 第一个时间对

\* @param second 第二个时间对

\* @return true 如果存在重合部分 否则false

\*/

public boolean checkCoinOrNot(TimeSlot first)：实现方法与上一函数基本相同，只不过这次比较的是时间对，需要分别调用其起止时间进行比较。

hashCode（）和equals（）：使用eclipse自带实现即可。

## 面向复用的设计：EntryState及State设计模式

3.6.1 State接口：

设计思想：实验中的计划项需要未分配资源、已启动、已结束等多个状态，采用state设计模式，为每个状态单独设计一个类。每个状态均需要实现相同的数个方法，从而帮助计划项进行相应的状态改变。

实现：

/\*\*

\* 改为分配状态

\* @param entry 状态所属的计划项

\* @return 状态成功改变则返回true 否则返回false

\*/

public boolean allocate(CommonPlanningEntry<?> entry);

/\*\*

\* 改为启动状态

\* @param entry 状态所属的计划项

\* @return 状态成功改变则返回true 否则返回false

\*/

public boolean run(CommonPlanningEntry<?> entry);

/\*\*

\* 改为结束状态

\* @param entry 状态所属的计划项

\* @return 状态成功改变则返回true 否则返回false

\*/

public boolean end(CommonPlanningEntry<?> entry);

/\*\*

\* 改为取消状态

\* @param entry 状态所属的计划项

\* @return 状态成功改变则返回true 否则返回false

\*/

public boolean cancel(CommonPlanningEntry<?> entry);

/\*\*

\* 改为挂起状态

\* @param entry 状态所属的计划项

\* @return 状态成功改变则返回true 否则返回false

\*/

public boolean block(CommonPlanningEntry<?> entry);

3.6.2 未分配资源状态；WAITING

设计思路：根据state设计模式的思想，可以将该类对象实例以类成员变量的形式存在，并且将构造器设为私有。从而，所有使用WAITING类状态的对象都是用的同一个实例。

成员变量如下：

public static final WAITING instance = new WAITING();

构造器：private WAITING() {};

方法实现：

public boolean allocate(CommonPlanningEntry<?> entry) ：直接调用entry的setState（）方法，并以一个ALLOCATED类型实例作为参数即可。

public boolean block(CommonPlanningEntry<?> entry) ：打印输出当前状态无法进行此操作。

public boolean cancel(CommonPlanningEntry<?> entry)：直接调用entry的setState（）方法，并以一个CANCELED类型实例作为参数即可。

public boolean end(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean run(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

3.6.3 已分配资源状态：ALLOCATED

设计思路：同WAITING，根据state设计模式的思想，可以将该类对象实例以类成员变量的形式存在，并且将构造器设为私有。

成员变量如下：

public static ALLOCATED instance = new ALLOCATED();

构造器：private ALLOCATED() {};

方法实现：

public boolean allocate(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean block(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean cancel(CommonPlanningEntry<?> entry)：直接调用entry的setState（）方法，并以一个CANCELED类型实例作为参数即可。

public boolean end(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean run(CommonPlanningEntry<?> entry) ：直接调用entry的setState（）方法，并以一个RUNNING类型实例作为参数即可。

3.6.4 已启动状态：RUNNING

设计思路：同上，根据state设计模式的思想，可以将该类对象实例以类成员变量的形式存在，并且将构造器设为私有。

成员变量如下：

public static RUNNING instance = new RUNNING();

构造器：private RUNNING() {};

方法实现：

public boolean allocate(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean block(CommonPlanningEntry<?> entry) ：直接调用entry的setState（）方法，并以一个BLOCKED类型实例作为参数即可。

public boolean cancel(CommonPlanningEntry<?> entry)：打印提示无法进行操作。

public boolean end(CommonPlanningEntry<?> entry) ：直接调用entry的setState（）方法，并以一个ENDED类型实例作为参数即可。

public boolean run(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

3.6.5 已取消状态：CANCELDED

设计思路：同上，根据state设计模式的思想，可以将该类对象实例以类成员变量的形式存在，并且将构造器设为私有。

成员变量如下：

public static CANCELED instance = new CANCELED();

构造器：private CANCELED() {};

方法实现：

public boolean allocate(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean block(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean cancel(CommonPlanningEntry<?> entry)：打印提示无法进行操作。

public boolean end(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean run(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

3.6.6 已挂起状态：BLOCKED

设计思路：同上，根据state设计模式的思想，可以将该类对象实例以类成员变量的形式存在，并且将构造器设为私有。

成员变量如下：

public static BLOCKED instance = new BLOCKED();

构造器：private BLOCKED() {};

方法实现：

public boolean allocate(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean block(CommonPlanningEntry<?> entry) ：直打印提示无法进行操作。

public boolean cancel(CommonPlanningEntry<?> entry)：直接调用entry的setState（）方法，并以一个CANCELED类型实例作为参数即可。

public boolean end(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean run(CommonPlanningEntry<?> entry) ：接调用entry的setState（）方法，并以一个RUNNING类型实例作为参数即可。

3.6.7 已结束状态：ENDED

设计思路：同上，根据state设计模式的思想，可以将该类对象实例以类成员变量的形式存在，并且将构造器设为私有。

成员变量如下：

public static ENDED instance = new ENDED();

构造器：private ENDED() {};

方法实现：

public boolean allocate(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean block(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean cancel(CommonPlanningEntry<?> entry)：打印提示无法进行操作。

public boolean end(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

public boolean run(CommonPlanningEntry<?> entry) ：打印提示无法进行操作。

## 面向应用的设计：Board

3.7.1 FlightBoard

**设计思想：**

机场信息板，可变类型，用于实时公布该位置上过去一小时出发或抵达和后续一小时即将出发或抵达的计划项，为此，在成员变量方面，信息板必须有时间和位置的属性，添加Calendar类和Location类的成员变量即可。另外，为了方便调用，可以将抵达的计划项和出发的计划项分别放在两个list中。其次，我们在后续任务中可以看到需要为board实现iterator的功能，至于是返回性计划项的list的迭代还是出发型的迭代，可有一个布尔变量作为属性进行控制。

另外，关于类中的方法，我们将实现两个set（）函数，用于向上面提到的两个list分别添加合适的计划项，其次为了判断时间差是否在一小时之内，我们也实现一个函数，方便set（）函数调用。另外，改变返回上述布尔量的方法、迭代器也需要实现。

**实现方法：**

成员变量：

private final Calendar timeNow; //当前时间

private final Location thisLocation;//该board所在地点

private List<FlightEntry> ToTherePlane = new ArrayList<>();//过去未来一小时抵达该地点的飞机

private List<FlightEntry> FromTherePlane = new ArrayList<>();//过去未来一小时从该地点出发的飞机

private boolean arrivIterator = true;

相关文档：

//Abstraction function:

// AF(timeNow,thisLocation,ToTherePlane,FromTherePlane,arrivIterator) = the time now,the location of the board,

// the list of the flight will start from this locaiton,the list of the flight will end in this locaiton,

// the controller of the type of the iterator

//Representation invariant:

// all fields must be non null

//Safety from rep exposure:

// all fields are private

// timeNow and thisLocation are final,ToTherePlane and FromTherePlane and arrivIterator are allowed to be changed

**方法的实现：**

**构造器public FlightBoard(Calendar timeNow, Location thisLocation)：实现相应两个基本的成员变量。**

**/\*\***

**\* 为信息板设置抵达航班列表（前后1小时抵达该机场的的计划项）**

**\* @param totalEntrys ，所有的计划项**

**\*/**

**public void setToList(List<? extends PlanningEntry<?>> totalEntrys)：**从计划项列表中取出所有目的地为该board所在机场并且抵达时间在当前时间前后一小时之内的计划项，加入成员变量ToTherePlane 中，

/\*\*

\* 为信息板设置起飞航班列表（前后1小时从该机场起飞的的计划项）

\* @param totalEntrys ，所有的计划项

\*/

public void setFromList(List<? extends PlanningEntry<?>> totalEntrys)：从计划项列表中取出所有出发地为该board所在机场并且出发时间在当前时间前后一小时之内的计划项，加入成员变量FromTherePlane 中，

/\*\*

\* 判断两个时间差距是否在withinGap分钟数之内

\* @param withinGap，可以接受的分钟数差距的范围

\* @param timeNow ，现在的时间

\* @param anoTime ，需要比较的时间

\* @return 若是返回true 否则false

\*/

public  boolean timeWithin(long withinGap,Calendar timeNow,Calendar anoTime)：调用Calendar自带的after()方法来比较两个时间参数的先后顺序，并均调用getTimeInMillis（），以此来进行相减操作，若结果在60\*1000内，则返回true，否则false。

visualize（）：下一部分说明。

/\*\*

\* 将Iterator修改为抵达计划项Iterator模式

\*/

public void setIterToArr()：改变成员变量arrivIterator 为true。

/\*\*

\* 将Iterator修改为出发计划项Iterator模式

\*/

public void setIterToFro()：改变成员变量arrivIterator 为false。

/\*\*

\*  返回Iterator的模式

\* @return 若为arrive的iterator返回true 否则若为departure的模式返回false

\*/

public boolean IterArrOrNot()：返回成员变量arrivIterator 值。

public Iterator<FlightEntry> iterator()：根据当前成员变量arrivIterator的值返回相应的list的迭代器，注意，返回前先将其按时间先后排序（排序交给按时间先后顺序实现了Compartor的类来进行）。

3.7.2 TrainBoard

**设计思想：**

火车站信息板，可变类型，用于实时公布该位置上过去一小时出发或抵达和后续一小时即将出发或抵达的计划项，成员变量方面，添加Calendar类和Location类的成员变量。将抵达的计划项和出发的计划项分别放在两个list中。返回性计划项的list的迭代是到达型还是出发型的迭代，可有一个布尔变量作为属性进行控制。

另外，关于类中的方法，实现两个set（）函数，用于向上面提到的两个list分别添加合适的计划项，其次为了判断时间差是否在一小时之内，我们也实现一个函数，方便set（）函数调用。另外，改变返回上述布尔量的方法、迭代器也需要实现。

**实现方法：**

成员变量：

private final Calendar timeNow; //当前时间

private final Location thisLocation;//该board所在地点

private List<TrainEntry> ToThereTrain  = new ArrayList<>();//过去未来一小时抵达该车站的火车

private List<TrainEntry> FromThereTrain  = new ArrayList<>();//过去未来一小时从该车站出发的火车

private boolean arrivIterator = true;

相关文档：

//Abstraction function:

// AF(timeNow,thisLocation,ToThereTrain,FromThereTrain,arrivIterator) = the time now,the location of the board,

// the list of the train will start from this locaiton,the list of the train will end in this locaiton,

// the controller of the type of the iterator

//Representation invariant:

// all fields must be non null

//Safety from rep exposure:

// all fields are private

// timeNow and thisLocation are final,ToThereTrain and ToThereTrain and arrivIterator are allowed to be changed

**方法的实现：**

构造器public TrainBoard(Calendar timeNow, Location thisLocation)：实现相应两个基本的成员变量。

/\*\*

\* 将最终车站为该车站或途径车站列表中含有该车站的抵达时间与当前时间差小于一小时的对象加入抵达列表

\* @param totalEntrys 所有计划项构成的列表

\*/

**public void setToList(List<? extends PlanningEntry<?>> totalEntrys)：**从计划项列表中取出所有途径或者最终目的地为该board所在车站并且抵达时间在当前时间前后一小时之内的计划项，加入成员变量ToThereTrain 中，

/\*\*

\* 将起始车站为该车站或途径车站列表中含有该车站的抵达时间与当前时间差小于一小时的对象加入抵达列表

\* @param totalEntrys 所有计划项构成的列表

\*/

**public void setFromList(List<? extends PlanningEntry<?>> totalEntrys)：**从计划项列表中取出所有出发地或途径出发地为该board所在车站并且出发时间在当前时间前后一小时之内的计划项，加入成员变量FromThereTrain

中，

/\*\*

\* 判断两个时间差距是否在withinGap分钟数之内

\* @param withinGap，可以接受的分钟数差距的范围

\* @param timeNow ，现在的时间

\* @param anoTime ，需要比较的时间

\* @return 若是返回true 否则false

\*/

**public  boolean timeWithin(long withinGap,Calendar timeNow,Calendar anoTime)：**调用Calendar自带的after()方法来比较两个时间参数的先后顺序，并均调用getTimeInMillis（），以此来进行相减操作，若结果在60\*1000内，则返回true，否则false。

/\*\*

\* 已知该列车计划项会抵达该车站 获得抵达时间

\* @param thisEntry 对应的计划项

\* @return 抵达时间

\*/

**public Calendar getToTime(TrainEntry thisEntry)：**只需找出当前位置在该计划项中的index，并去其时间对列表中找相应的时间对的起始时间即可。当然先要做鲁棒性处理，防止参数不满足前置条件。

/\*\*

\* 已知该列车计划项会从该车站发车 获得出发时间

\* @param thisEntry

\* @return

\*/

**public Calendar getFromTime(TrainEntry thisEntry)**：只需找出当前位置在该计划项中的index，并去其时间对列表中找相应的时间对的截止时间即可。当然先要做鲁棒性处理，防止参数不满足前置条件。

/\*\*

\* 已知一个任务项会从这个地点出发 获取它的当前状态

\* @param thisEntry 上述的计划项

\* @return 文字描述的状态

\*/

**public String getStateOfFrom(TrainEntry thisEntry)**：为了方便后面信息板可视化中状态的描述，我们用字符串的形式表述出该计划项对于该车站来说当前的状态（比如即将出发、已出发等等）。实现方法就是根据当前时间与出发时间进行比较即可，当然需要考虑被取消的特殊情况。

/\*\*

\* 已知一个计划项会抵达这个地点 获取它的当前状态

\* @param thisEntry 上述的计划项

\* @return 文字描述的状态

\*/

**public String getStateOfTo(TrainEntry thisEntry)**：如上，为了方便后面信息板可视化中状态的描述，我们用字符串的形式表述出该计划项对于该车站来说当前的状态（比如即将到站、已抵达等等）。实现方法就是根据当前时间与出发时间进行比较即可，当然需要考虑被取消的特殊情况。

**visualize（）**：下一部分说明。

/\*\*

\* 将Iterator修改为抵达计划项Iterator模式

\*/

**public void setIterToArr()**：改变成员变量arrivIterator 为true。

/\*\*

\* 将Iterator修改为出发计划项Iterator模式

\*/

**public void setIterToFro()**：改变成员变量arrivIterator 为false。

/\*\*

\*  返回Iterator的模式

\* @return 若为arrive的iterator返回true 否则若为departure的模式返回false

\*/

public boolean IterArrOrNot()：返回成员变量arrivIterator 值。

public Iterator<FlightEntry> iterator()：根据当前成员变量arrivIterator的值返回相应的list的迭代器，注意，返回前先将其按时间先后排序（排序交给按时间先后顺序实现了Compartor的类来进行）。

3.7.3 ClassBoard

**设计思想：**

教室信息板，可变类型，用于实时公布在当天该教室上课的所有计划项，为此，在成员变量方面，信息板必须有时间和位置的属性，添加Calendar类和Location类的成员变量即可。另外，将相关计划项放入list中。

另外，关于类中的方法，我们将实现一个set（）函数，用于向上面提到的list添加合适的计划项。

**实现方法：**

成员变量：

private final Calendar timeNow; // 当前时间

private final Location thisLocation;// 该board所在地点

private List<CourseEntry> thisList = new ArrayList<>();

相关文档：

//Abstraction function:

// AF(timeNow,thisLocation,thisList) = the time now,the location of the board,the list of the entrys that will be in this location

//Representation invariant:

// all fields must be non null

//Safety from rep exposure:

//  all fields are private

// timeNow and thisLocation are final,thisList is allowed to be changed

方法的实现：

构造器public ClassBoard(Calendar timeNow, Location thisLocation)：实现相应两个基本的成员变量。

/\*\*

\* 将当天在该教室上课的课程加入列表

\* @param totalEntrys 所有的计划项

\*/

public void setClasses(List<? extends PlanningEntry<?>> totalEntrys )：从计划项列表中取出所有位置为该board所在教室并且上课时间在当前时间当天的计划项，加入成员变量thisList 中，

visualize（）：下一部分说明。

public Iterator<FlightEntry> iterator()：thisList的迭代器，注意，返回前先将其按时间先后排序（排序交给按时间先后顺序实现了Compartor的类来进行）。

## Board的可视化：外部API的复用

**设计思想：**为了实现board的GUI的可视化，（即指导书中提到的visualize()方法），我引入javax.swing库对方法进行实现。以上面提到的ClassBoard为例，通过JFrame classBoard = new JFrame();声明一个框架，在该框架上声明一个JTable，随后再声明两个数组，分别填入与计划项相关的信息即可：

for(CourseEntry thisCourse :thisList)

{

SimpleDateFormat form = new SimpleDateFormat("HH:mm");

String calOneStr = form.format(thisCourse.getSlot().getBeginTime().getTime());

String calTwoStr = form.format(thisCourse.getSlot().getEndTime().getTime());

thisInfo[i][0] = calOneStr +"-"+ calTwoStr;//获取抵达时间

thisInfo[i][1] = thisCourse.getName();//课程的名字

thisInfo[i][2] = ((Teacher)(thisCourse.getResource())).getname();//授课教师名

thisInfo[i++][3] = thisCourse.getState();//课程状态

}

    // 创建表格中的横标题

    String[] Names = { "课程时间" , "课程名称", "授课教师", "课程状态"};

    // 以Names和playerInfo为参数，创建一个表格

JTable table = new JTable(thisInfo, Names);

最后，对表格的题目，框架的大小、位置、可视等进行设置，还可以添加一个上下滚动轮，并设置窗口关闭的结果：dispose()，就完成了可视化的实现。

另外，关于FlightBoard和TrainBoard，二者均有抵达和出发的两个部分，对此，我将其分别放到两个表格中，调节位置，使其并列出现在屏幕上。

## 可复用API设计及Façade设计模式

### 检测一组计划项之间是否存在位置独占冲突

方法spec：

/\*\*

\* 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。

\* @param thisCheckStrategy,选择检查的算法

\* @param entries，要检查的计划项列表

\* @return 是否存在conflict

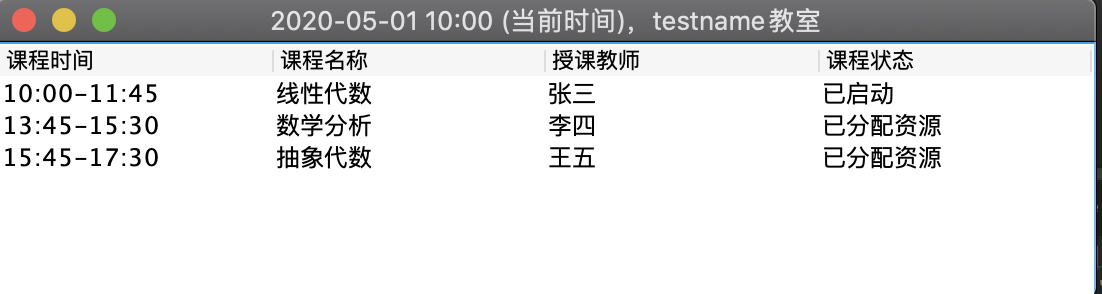
\*/

Public boolean checkLocationConflict (CheckLocationInterface thisCheckStrategy,List<? extends PlanningEntry<?>>   entries)

根据后面的strategy设计模式，我为这个api方法实现了两种策略，将检查冲突的任务委托给传入的参数thisCheckStrategy。

具体的实现思想为，先判断entries中计划项所属的类别，若是对应类别的位置是可共享的，则直接返回false，否则两层循环便利entries，判断有无使用相同位置而且时间对重合（判断是否有时间重合只需要调用TimeSlot类中实现的checkCoinOrNot（）即可）。

结果：







### 检测一组计划项之间是否存在资源独占冲突

方法spec：

 /\*\*

    \* 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

    \* @param entries 计划项列表

    \* @return 是否有冲突

    \*/

  public boolean checkResourceExclusiveConflict(List<? extends PlanningEntry<?>> entries)

实现思想：三种计划项的实现有些许不同，因此先根据参数中计划项的所属类switch进行判断，若为课程或飞机计划项，则双重循环遍历比较是否有使用相同资源并且时间对重合（方法和上面说的基本相同），从而返回结果；若为列车计划项，则需要多一重循环，因为每个计划项有一个资源列表。

### 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

    \* @param r 相应的资源

    \* @param e 相应的计划项

    \* @param entries 所有计划项的列表

    \* @return entries中e的关于r的前序计划项

    \*/

  public  PlanningEntry<R> findPreEntryPerResource(R r,

        PlanningEntry<R> e, List<? extends PlanningEntry<R>> entries)：和前两者类似，首先需要判断计划项所属类，并且做一些鲁棒性处理比如参数r可能并不是参数e的资源。判断所属类别后，需要对entries循环遍历，查找同样使用资源r并且时间在e的前面的计划项，对所有满足条件者，在找一个最晚的，返回即可。

## 设计模式应用

请分小节介绍每种设计模式在你的ADT和应用设计中的具体应用。

### Factory Method

工厂模式的应用思想：利用工厂模式，为我实现的三个具体Entry类：FlightEntry，TrainEntry和ClassEntry实现了工厂方法，从而方便客户端的使用，也隐藏了一些信息。

我实现了三个工厂类，每一个类中都提供了返回一个entry的工厂方法。

以其中一个返回课程计划项的工厂方法为例：

public class CourseEntryFactory {

public CourseEntry getEntry(String name) {

LocationCouldChange changeObject  = new LocationCouldChangeImple();

SingleLocationEntry singleLoc = new SingleLocationEntryImplement();

SingleResourceEntry singleRes = new SingleSourceEntryImplement();

SingleSlot singleSlo = new SingleSlotimple();

return new CourseEntry(changeObject, singleLoc, singleRes, singleSlo, name);

}

### Iterator

设计：迭代器模式主要是用在了board包中，为每个board类的对象提供了迭代显示列表中计划项的方法，我通过实现Iterator<entry>接口中next（）等方法，为每个类构造特别的迭代器。board对象可以通过调用iterator（）返回一个迭代器，对相关计划项进行遍历。

### Strategy

在上面复用API的设计中，我为checkLocationConflict（CheckLocationInterface thisCheckStrategy,List<? extends PlanningEntry<?>>   entries）方法提供了两种实现策略，通过向该方法中传入不同的参数进行委托来实现。

第一种：CheckLocationConForceImp

先判断entries中计划项所属的类别，若是对应类别的位置是可共享的，则直接返回false，否则两层循环便利entries，判断有无使用相同位置而且时间对重合（判断是否有时间重合只需要调用TimeSlot类中实现的checkCoinOrNot（）即可）。

第二种：CheckLocationConFast

这一实现方案相对于前者进行了一部分优化，首先是双重迭代部分。内层迭代总是从外迭代对应遍数下一个开始，减少时间开销，同时内层迭代进行比较时，先看时间是否重合，若重合在比较位置，相对于上一种实现两种同时比较减少了调用次数。

## 应用设计与开发

### 航班应用

**设计与实现思想**：

航班应用的设计类似于lab2中棋类游戏的应用设计，主体框架可以通过while（true）循环读取client输入，并用swtich结构语句来根据输入的不同判断需要执行的组块。至于主体框架之外，我还实现了几个辅助函数，包括：打印帮助信息的printHelp()方法，通过输入资源ID和当前所有资源的列表查找特定资源的方法public static Plane findResource(String ID,Set<Plane>resourceList)，以及通过输入位置名称和当前所有的位置构成的列表来查找特定位置的方法public static Airport findAirport(String name,Set<Airport>locationList)，还有查找根据计划项的名称在现有所有计划项中查找特定计划项的方法public static FlightEntry findEntry(String name,List<FlightEntry>entryList)。

**使用说明**：程序开始运行后，控制台会输出帮助信息，即用户可以根据需求向控制台输入相应的字母从而进行相应的操作，比如，用户输入字符‘a’，可以进入增加一架飞机的功能中，随后控制台会继续输出关于用户想设定出的飞机的具体信息的提示，如果用户的输入符合格式，成功增加一架飞机后，总的while循环再次执行，进入下一个需求选择环节，你可以继续按照想要执行的指令选择相应的字母（还有单词，比如输入‘help’会再次给出帮助信息）。另外，app实现了通过正则表达式读取文件信息并建立计划项的功能，根据提示输入’parser‘，并根据随后给出的提示输入文件位置，即可根据文件读取信息生成计划项，当然如果文件格式有问题，会报错推出。

### 高铁应用

高铁应用的设计与前者几乎相同，即主体框架可以通过while（true）循环读取client输入，并用swtich结构语句来根据输入的不同判断需要执行的组块，增加了挂起重启等操作。只是其中几个部分如新建计划项时需要读入的信息更多一些。同样的，我也在该app中实现了上一个app提到的相关辅助方法。

**使用说明**：程序开始运行后，控制台会输出帮助信息，即用户可以根据需求向控制台输入相应的字母从而进行相应的操作，比如，用户输入字符‘a’，可以进入增加一个车厢的功能中，随后控制台会继续输出关于用户想设定出的车厢的具体信息的提示，如果用户的输入符合格式，成功增加一个车厢后，总的while循环再次执行，进入下一个需求选择环节，你可以继续按照想要执行的指令选择相应的字母（还有单词，比如输入‘help’会再次给出帮助信息）。相对前一个航班app，这里的变化是在分配资源或者新建计划项时，由于列车计划项的特性，往往都需要循环读入一列的时间对或者位置、资源等。

### 课表应用

课表应用的设计与前者几乎相同，即主体框架可以通过while（true）循环读取client输入，并用swtich结构语句来根据输入的不同判断需要执行的组块，只是增加了更改位置等操作。同样的，我也在该app中实现了前面的app提到的相关辅助方法。

**使用说明**：程序开始运行后，控制台会输出帮助信息，即用户可以根据需求向控制台输入相应的字母从而进行相应的操作，比如，用户输入字符‘a’，可以进入增加一个教师的功能中，随后控制台会继续输出关于用户想设定出的教师的具体信息的提示，如果用户的输入符合格式，成功增加一个教师后，总的while循环再次执行，进入下一个需求选择环节，你可以继续按照想要执行的指令选择相应的字母（还有单词，比如输入‘help’会再次给出帮助信息）。

## 基于语法的数据读入

设计方法：

按照指导书的要求，app实现了通过正则表达式读取文件信息并建立计划项的功能，根据提示输入’parser‘，并根据随后给出的提示输入文件位置，即可根据文件读取信息生成计划项，当然如果文件格式有问题，会报错推出。

其中，我运用正则表达式进行字符串匹配的主要方法是：根据文件特定的格式和语法，我将读入一个计划项单元拆分成了13个行的读取，每个行应匹配特定的正则表达式，其中正则表达式的实现调用的Pattern类和mathc类的方法，读取后进行匹配判断，若按照我所需要的语法规则匹配成功，则继续进行，否则退出；

下面是部分代码展示：

String pattern3 = "\\s\*DepartureAirport:(\\w+)";

newPar = Pattern.compile(pattern3);

thisLine = fileScan.nextLine();

m = newPar.matcher(thisLine);

if (!m.matches()) {

System.out.println(thisLine + "：该行不符合规范，停止读入");

System.exit(0);

}

String departureAir = m.group(1);

该段代码尝试匹配第4行关于起飞机场的信息，如匹配成功，将信息保存到一个变量中，方便后续进行新计划项的建立。

## 应对面临的新变化

只考虑你选定的三个应用的变化即可。

### 变化1:航班

评估之前的设计是否可应对变化、代价如何：我认为可以应对变化，并且代价较小，为了实现中途经停，只需要将位置数量和时间对数量的接口转变为多个的接口，并在app、board中改动一部分使用方法即可。但是test也需要相应的修改。

如何修改设计以应对变化：首先，如上所述，将位置数量和时间对数量的接口转变为多个的接口，实现接口的方法可以先照搬高铁的实现，同时要限制经停站只有一个，这方面改动较小，只有20-30行的改动。主要的改动在board和app中，由于方法的接口改变，需要在所有调用该方法的地方进行改动，此处的改动较大，但仍可借用高铁的实现，只需要改一些变量所属的类即可。另外，test中也因为接口的变动而需要改动。

### 变化2：高铁

评估之前的设计是否可应对变化、代价如何：可以应对变化，代价很小。分配了车厢后不能取消，只需要在TrainEntry的方法cancel（）中添加一个判断条件即可。

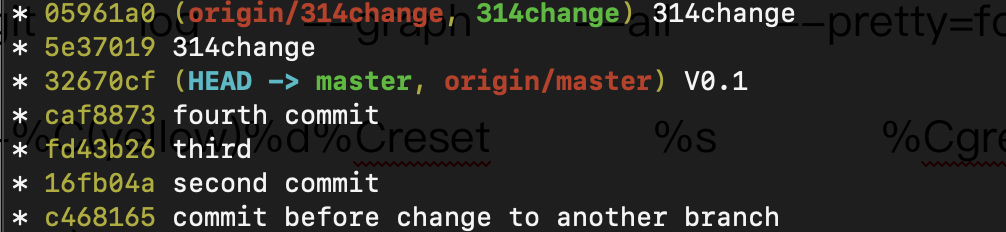
如何修改设计以应对变化：如果修改state包中类的方法，可能在未来会对其他的app产生影响。因此，我们只在TrainEntry中修改，在cancel（）中添加一个判断条件，即当状态为BLOCKED时显示禁止取消功能。这样，就完成了指导书的要求。另外还需要修改一下test中的断言。

### 变化3：课程

评估之前的设计是否可应对变化、代价如何：可应对变化，并且代价较小。类似于航班的修改，这里只需要更改一下CourseEntry实现的资源数量的接口，并且在board、app中调用处修改一下调用参数即可。

如何修改设计以应对变化：如上所述，更改CourseEntry实现的资源数量的接口，并且在board、app中调用处修改一下调用参数,同时需要修改test中的测试方法。

## 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。  


# 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 |
| 4-15 | 1:45-3:30 | 听课、阅读指导文档 | 完成 |
| 4-15 | 4:30-6:30 | 阅读文档、进行设计 | 完成 |
| 4-17 | 2:30-6:30 | 设计、实现ADT框架 | 未完成 |
| 4-18 | 2:30-6:30 | 继续设计、实现ADT框架 | 未完成 |
| 4-19 | 9:00-12:00 | 实现局部共性ADT | 未完成 |
| 4-20 | 6:00-9:00 | 实现局部共性ADT | 完成 |
| 4-22 | 13:45-3:30 | 实现CommonPlanningEntry | 未完成 |
| 4-24 | 8:00-11:00 | 实现CommonPlanningEntry | 完成 |
| 4-25 | 9:00-11:00 | 实现FlightEntry的组合设计及设计test | 部分完成（后续增加） |
| 4-26 | 2:00-6:00 | 实现CourseEntry的组合及test | 部分完成（后增） |
| 4-27 | 8:00-11:00 | 实现trainEntryEntry的组合及test | 未完成 |
| 4-28 | 7:30-11:00 | trainEntryEntry组合及test | 完成 |
| 4:30 | 8:30-11:00 | Flightentry的实现 | 完成 |
| 5-3 | 2:00-5:00 | CourseEntry实现 | 完成 |
| 5-3 | 7:00-10:00 | TrainEntry实现 | 未完成 |
| 5-4 | 8:00-11:00 | Resource的设计与test | 完成 |
| 5-5 | 2:00-5:30 | TrainEntry实现 | 完成 |
| 5-5 | 7:30\_10:30 | Location的设计与test | 完成 |
| 5-6 | 8:00-11:00 | TimeSlot的设计与test | 未完成 |
| 5-7 | 2:00-5:30 | TimeSlot总体实现 | 完成 |
| 5-7 | 8:00-11:00 | Resource补充实现 | 完成 |
| 5-8 | 2:00-5:30 | 航班board的部分实现 | 未完成 |
| 5-9 | 8:00-11:00 | 航班board的部分实现 | 完成 |
| 5-10 | 8:00-11:00 | 课程board的部分实现 | 完成 |
| 5-11 | 2:00-5:30 | 列车board的部分实现 | 完成 |
| 5-11 | 8:00-11:00 | 三个board可视化 | 未完成 |
| 5-12 | 2:00-5:30 | 三个board可视化 | 完成 |
| 5-12 | 8:00-11:00 | 可复用API | 完成 |
| 5-13 | 2:00-5:30 | 工厂方法及迭代器 | 未完成 |
| 5-13 | 7:30\_10:30 | 工厂方法及迭代器 | 完成 |
| 5-14 | 2:00-5:30 | 应用实现 | 未完成 |
| 5-14 | 8:00-11:00 | 应用实现 | 完成 |
| 5-15 | 8:00-11:00 | parser | 完成 |
| 5-16 | 2:00-6:00 | 314cahnge分支 | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 正则表达式的使用 | 通过查阅网上的资料，以及学习后续的课程，写出合适的正则表达式来匹配文件。 |
| 总体设计 | 经过多日思考，并画出各计划项设计的结构图 |
| 信息板可视化的实现 | 查阅JSwing中相关知识。学习了如何向Jtable中填充信息。 |
| 对信息板中计划项的排序 | 复习曾经学过的Iterator和Comparator的实现思路，解决问题。 |
| state设计模式 | 通过网上的相关资料，以及后续课程的学习得以解决。 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

经验：

面向ADT的编程需要更多的考虑程序结构的设计与组合，为此在最初设计主体框架时，应当选择较好的设计模式，比如应用CRP能较大的帮助我们减少多应用的开发成本，较大实现可复用功能。

另外，经过长时间的持续开发，不断的翻看各种以前实现的类的方法等，我深刻意识到，如果没有spec、RI等文档说明，在后面的开发过程中将会因为开发者的遗忘而使开发变得极度困难，这些工作帮助我们使长期的开发更为轻松。

同时，本次实验应用的parser解析器对于极大规模文件的读取匹配来说，是具有非常意义的工具，这为我以后的工程开发带来极大便利。

教训：

对于接口、抽象类、具体类的实现与设计而言，为了实现更大程度地可复用功能，我们尽量将方法、属性往上、往抽象了放，也就是完全共性的我们争取放到接口中，从而在client中使用时、复用时等减少开发成本。但是，也要考虑实际情况，不能一味的为了往上放而不计开发成本。因此，这方面的设计需要综合考量。

## 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？

相对于后者而言，面向ADT的编程需要更多的考虑程序结构的设计与组合，如何更好的复用，这是它的难点所在。而面向应用场景更多看单次的实现，不太注重复用。

本次试验，通过应用CRP等技术设计出的ADT具有较好的复用性，大大减少了我的工作量。

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

经过长达五个星期的持续开发，不断的翻看各种以前实现的类的方法等，我深刻意识到，如果没有spec、RI等文档说明，在后面的开发过程中将会因为开发者的遗忘而使开发变得极度困难，这些工作帮助我们使长期的开发更为轻松。

因此，我觉得很有必要将这种工作习惯坚持下去。

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

难处是我需要学习他人API的方法的调用是怎样的，同时我还需要揣摩他人的设计意图。乐趣就是减少了自己开发的成本与难度。

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

正如题目所述，设计模式是我们开发的程序具有更好的扩展性、复用性，同时也更加结构化。设计模式是对我们开发程序而言宝贵的思想财富。

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

在开发解析器的过程中，可能会遇到一些困难，但是一旦开发出来解析器，对于极大规模文件的读取匹配，将是一项非常轻松的任务，这令人惊叹。

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

正如题目，完全从0开始进行ADT的设计就是难度所在。首先，我们需要掌握一定的设计模式，从而使设计出的ADT更加具有可复用性；同时，我们需要明智的进行各种接口的组合、具体类的组合，生成所期望的应用等等。

我通过不断的学习，思考并记录设计的主体框架，从而使得程序设计的思想更加清晰，今儿克服这些困难。

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

为了实现更大程度地可复用功能，我们尽量将方法、属性往上、往抽象了放，也就是完全共性的我们争取放到接口中，从而在client中使用时、复用时等减少开发成本。但是，也要考虑实际情况，不能一味的为了往上放而不计开发成本。因此，这方面的设计需要综合考量。

1. 关于本实验的工作量、难度、deadline。

实验任务量较大；难度方面。在设计思想上具有一定的挑战性，实现难度则较为适当。deadline刚刚好。

1. 到目前为止你对《软件构造》课程的评价。

课程内容充实而有趣味，并且实验虽然任务量较大，具有一定的挑战性，但对于知识掌握和应用有很大帮助，我也比较喜欢这门课程。