

**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 | 梁书育 |
| 学号 | 1180300208 |
| 班号 | 1803002 |
| 电子邮件 | 951009723@qq.com |
| 手机号码 | 18845773527 |

**目录**

[1 实验目标概述 1](#_Toc40604165)

[2 实验环境配置 1](#_Toc40604166)

[3 实验过程 1](#_Toc40604167)

[3.1 待开发的三个应用场景 1](#_Toc40604168)

[3.2 面向可复用性和可维护性的设计：PlanningEntry<R> 2](#_Toc40604169)

[3.2.1 PlanningEntry<R>的共性操作 2](#_Toc40604170)

[3.2.2 局部共性特征的设计方案 2](#_Toc40604171)

[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](#_Toc40604172)

[3.3 面向复用的设计：R 3](#_Toc40604173)

[3.4 面向复用的设计：Location 3](#_Toc40604174)

[3.5 面向复用的设计：Timeslot 4](#_Toc40604175)

[3.6 面向复用的设计：EntryState及State设计模式 4](#_Toc40604176)

[3.7 面向应用的设计：Board 4](#_Toc40604177)

[3.8 Board的可视化：外部API的复用 5](#_Toc40604178)

[3.9 可复用API设计及Façade设计模式 9](#_Toc40604179)

[3.9.1 检测一组计划项之间是否存在位置独占冲突 10](#_Toc40604180)

[3.9.2 检测一组计划项之间是否存在资源独占冲突 10](#_Toc40604181)

[3.9.3 提取面向特定资源的前序计划项 13](#_Toc40604182)

[3.10 设计模式应用 14](#_Toc40604183)

[3.10.1 Factory Method 14](#_Toc40604184)

[3.10.2 Iterator 16](#_Toc40604185)

[3.10.3 Strategy 17](#_Toc40604186)

[3.11 应用设计与开发 18](#_Toc40604187)

[3.11.1 航班应用 18](#_Toc40604188)

[3.11.2 高铁应用 19](#_Toc40604189)

[3.11.3 课表应用 19](#_Toc40604190)

[3.12 基于语法的数据读入 20](#_Toc40604191)

[3.13 应对面临的新变化 23](#_Toc40604192)

[3.13.1 变化1 ：FlightEntry的改变 23](#_Toc40604193)

[3.13.2 变化2：TrainEntry的改变 23](#_Toc40604194)

[3.13.3 变化3：CourseEntry的改变 25](#_Toc40604195)

[3.14 Git仓库结构 25](#_Toc40604196)

[4 实验进度记录 26](#_Toc40604197)

[5 实验过程中遇到的困难与解决途径 27](#_Toc40604198)

[6 实验过程中收获的经验、教训、感想 27](#_Toc40604199)

[6.1 实验过程中收获的经验和教训 27](#_Toc40604200)

[6.2 针对以下方面的感受 27](#_Toc40604201)

# 实验目标概述

本次实验覆盖课程第3、4、5章的内容，目标是编写具有可复用性和可维护 性的软件，主要使用以下软件构造技术：

·子类型、泛型、多态、重写、重载

·继承、代理、组合

·常见的OO设计模式

·语法驱动的编程、正则表达式

·基于状态的编程

·API设计、API复用

本次实验给定了五个具体应用（高铁车次管理、航班管理、操作系统进程管 理、大学课表管理、学习活动日程管理），学生不是直接针对五个应用分别编程 实现，而是通过 ADT 和泛型等抽象技术，开发一套可复用的ADT及其实现，充分考虑这些应用之间的相似性和差异性，使ADT有更大程度的复用（可复用性）和更容易面向各种变化（可维护性）。

# 实验环境配置

实验环境与之前相同，不再详细叙述。

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

<https://github.com/ComputerScienceHIT/Lab3-1180300208.git>

# 实验过程

## 待开发的三个应用场景

列出你所选定的三个应用。

1、航班管理

2、高铁车次管理

3、大学课程表管理

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。

经过分析，我们可以看到，这三个应用的状态改变有共性的方面：创建、启动、结束、分配资源、取消等，我们可以考虑将这些改变状态的方法放在同一个类中，共三种应用场景复用。在资源方面，航班管理和大学课程表管理都是单一资源，而高铁车次管理则是多个有次序的资源，可以考虑实现两个有关资源的接口分别对应这两种不同需求。物理位置上，这三种应用场景都不相同，航班管理有分别是起始地点和终点的两个地点，高铁车次管理除了始发地和终点之外还有一系列中间经停的地点，而大学课程表管理则只需一个地点，我们需要对这不同的三种情况做处理。时间属性上，航班管理和大学课程表管理均只需要一对起止时间对，而高铁车次管理则需要一系列的起止时间对用于中间的阻塞过程。

## 面向可复用性和可维护性的设计：PlanningEntry<R>

### PlanningEntry<R>的共性操作

PlanningEntry是一个接口，定义了对通用的状态进行改变、获取计划项的名称以及获取计划项的状态的方法。这些方法的具体实现在其实现类CommonPlanningEntry中，其中CommonPlanningEntry的rep有两个，分别为状态state（为之后要提到的EntryState类型的对象）以及String类型的name。三个应用场景公用的改变类型的allocate、start、cancel、end等方法和获取计划项名称的方法可以直接在这个实现类中实现，而其余各个计划项有所区别的获取状态信息以及作为排序依据的compareTo方法则作为抽象方法，由具体的应用场景子类来具体实现。

### 局部共性特征的设计方案

三个具体应用场景的局部共性特征有计划项名称——均为String类型、状态——均为EntryState的子类型，因此，我们需要将这些特征值相关的内容放在共性的CommonPlanningEntry中，对于除了有关于block状态转换的方法，都在共性的类中进行实现，从而降低开发的工程量，提高代码的复用率。

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

这一部分将针对三个应用场景分别进行描述，前面提到计划项有三个核心组成，分别为地点、时间和资源，所以总的设计思路就是为这三项内容分别实现能够满足不同需求的接口，实现具体Entry类时，除了继承CommonPlanningEntry这一共性方法外，再调用符合自己特征的接口，从而完成Entry子类的组装。

FlightEntry：

根据之前的分析，Flight的位置特征为两个位置，时间特征为一个起止时间对，资源特征为单一资源。因此需要设计的三个接口分别为：双位置接口、不可阻塞时间接口、单一资源接口，在这三个接口中定义需要的对三个特征的方法，再定义这三个接口的实现类，实现这三个接口。为了使代码清晰整洁容易理解，将这三个接口封装在一个总的接口FlightPlanningEntry中。完成了以上准备工作之后实现FlightEntry，FlightEntry的数据域为三个接口的具体实现类，有关时间、地点、资源的操作均委托给接口实现类来完成，compareTo根据Flight的起始时间来完成，getPresentStateName则由state的再flight中的具体含义来完成。

TrainEntry：

TrainEntry的实现与FlightEntry的实现类似，需要完成的三个接口为：多位置接口、可阻塞时间接口、多个有序资源接口。唯一需要注意的是，由于TrainEntry可阻塞，因此block和restart方法需要在TrainEntry中实现，用以使TrainEntry的状态可以改变为BLOCKED。

CourseEntry：

CourseEntry需要的三个接口为：单一可变位置接口、单一资源接口、不可阻塞时间接口，其中方法的实现也与上述应用类的方法实现类似，不再赘述。

## 面向复用的设计：R

以上三个具体应用类用到的资源R分别为：飞机资源、车厢资源、教师资源。下面分别进行详述。

飞机资源（Plane）：

飞机资源有飞机编号、飞机机型、座位数、机龄等特征。在Plane类中实现getter方法获得这些信息，同时重写hashCode和equals方法。Plane的数据如下所示：

**private** String planeID;

**private** String planeType;

**private** **int** seatNumber;

**private** **float** planeAge;

车厢资源（Carriage）：

车厢资源有车厢编号、类型、定员数、出厂年份等属性。在Carriage类中实现getter方法来获得这些信息，同时重写hashCode和equals方法。Carriage的数据如下所示：

**private** String carriageID;

**private** String carriageType;

**private** **int** seatNumber;

**private** **int** productYear;

教师资源（Teacher）：

教师资源有身份证号、姓名、性别、职称等属性。在Teacher中实现getter方法来获得这些信息，同时重写hashCode和equals方法。Teacher的数据如下所示：

**private** String teacherID;

**private** String name;

**private** String gender;

**private** String position;

## 面向复用的设计：Location

一个“位置”对象的属性包括：经度、纬度、名称、是否可共享使用，同时由于一些地点没有经纬度信息，或者在一些应用场景下经纬度信息不重要，因此我们为Location提供两套构造方法，一种是需要提供全部的属性包括经纬度、名称和是否可共享，另一种则只需提供名称和是否可共享即可，在这种情况下，构造器会默认将经纬度全部设为0。

## 面向复用的设计：Timeslot

Timeslot是一个带有起始时间和结束时间的ADT，因此我们在Timeslot中会放两个LocalDateTime的数据来表示起始时间和终止时间。选取LocalDateTime是由于这个类是immutable的，使用时无需考虑数据被恶意篡改。为了后面使用及测试的方便，Timeslot也提供两种构造方法，一种是直接将两组年月日小时分钟信息传入，另一种是传入两个LocalDateTime类型的数据。

## 面向复用的设计：EntryState及State设计模式

EntryState是一个定义了状态类所有方法的接口，其他所有的状态均是EntryState的实现类。

其中共有6个状态，分别如下：

WAITING：已创建具体计划项，等待分配资源

ALLOCATED：已分配资源但未开始

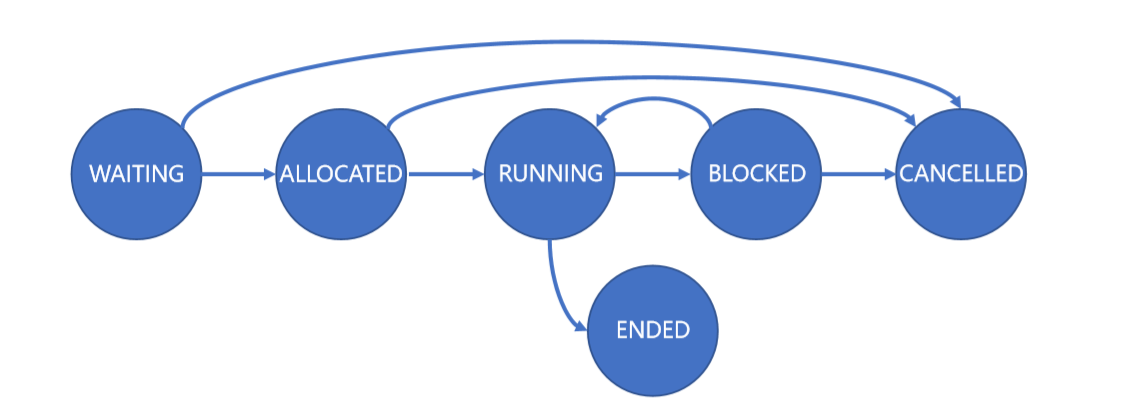
RUNNING：计划项正在执行中

BLOCKED：计划项已阻塞

CANCELLED：计划项已取消

ENDED：计划项已运行结束

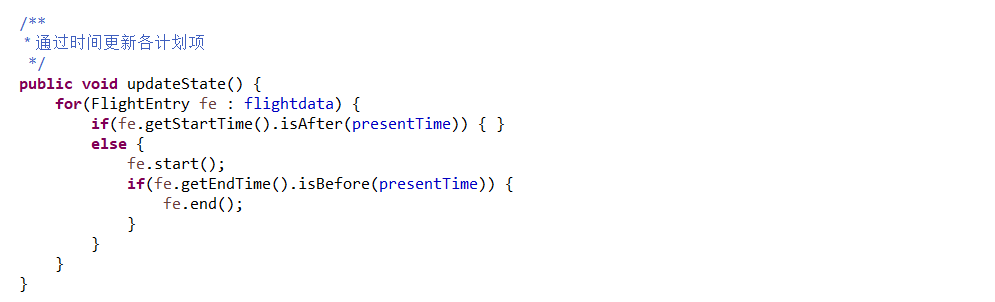
每一个状态类型中都有方法来改变状态，在满足条件的情况下对状态进行转换。具体的转换方式如下：



## 面向应用的设计：Board

为了将数据和表示相分离，并更大程度上对代码进行复用，设计了Data类来存储计划项，关于数据的操作均通过Data类来完成，而Board则只负责数据的可视化。由于在getPlan等操作上各计划项仍然有细微的差别，因此选择将个性化的方法放在具体的子类CourseData、FlightData、TrainData中，实际Board中使用的数据在这些子类中。在可视化表格前需要对数据进行处理，以FlightBoard为例。

更新计划项：



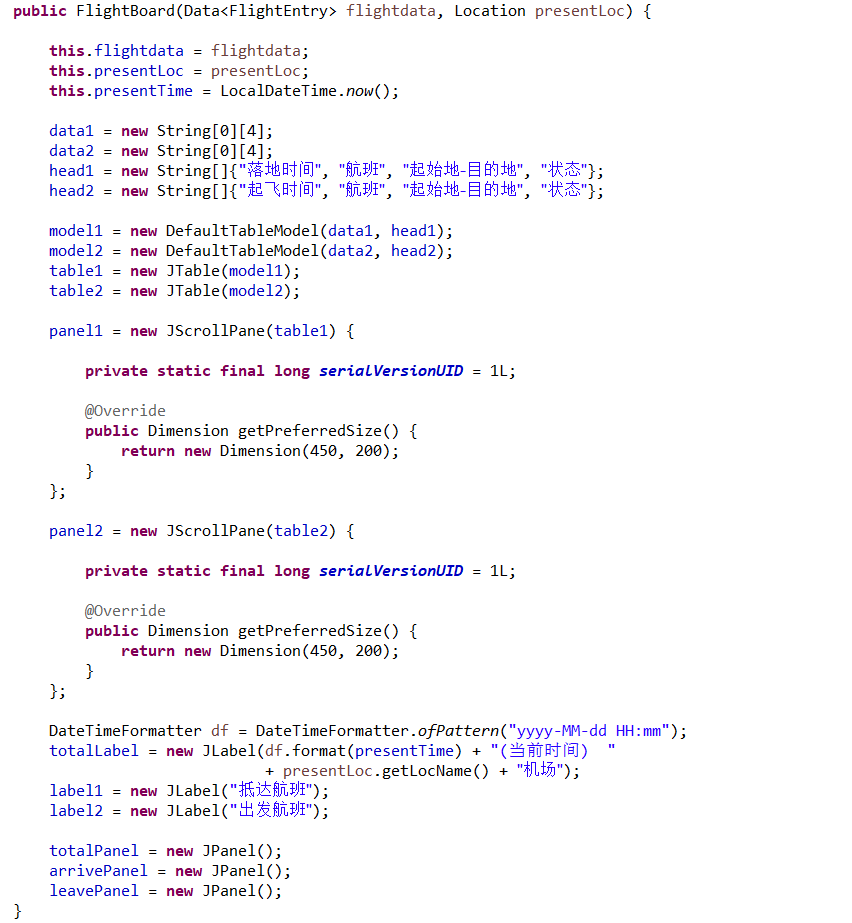
整合数据到表格：



## Board的可视化：外部API的复用

Board使用了JTable进行数据可视化。FlightBoard中由于有抵达航班和出发航班两个表，所以使用了两个JTable，TrainBoard同理。同时为了表格的美观和布局的协调，使用了Y轴箱式布局（BoxLayout）。两个表格都有滚轮结构，能够满足多组数据的显示需要，由于数据在展示前进行了排序，因此计划项的排列是按照出发时间升序排列的。下面以FlightBoard为例，介绍可视化方法。

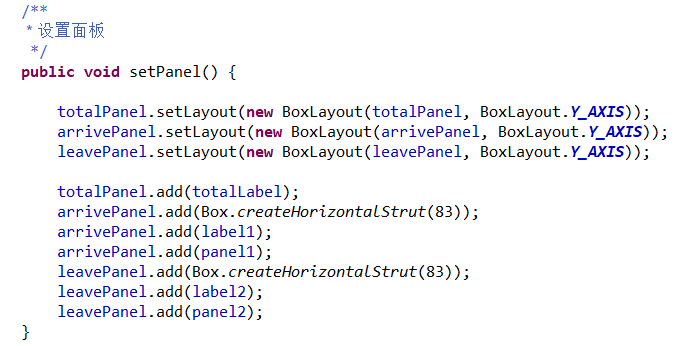
初始化FlightBoard：



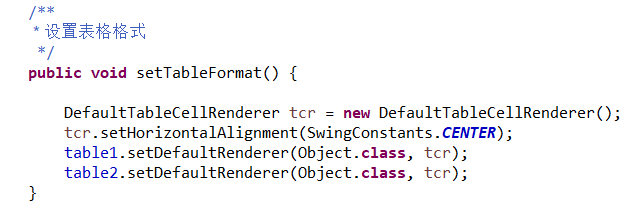
设置总体框架：



设置面板：



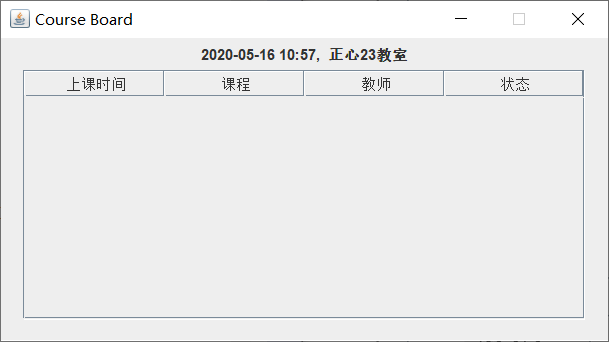
设置表格格式：



实现效果如下：



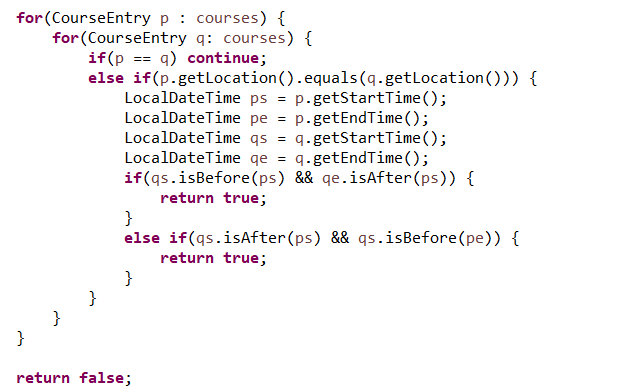




## 可复用API设计及Façade设计模式

### 检测一组计划项之间是否存在位置独占冲突

首先判断传入的计划项列表中的计划项类型，如果是航班或者车次则直接返回false，如果是课程，则遍历计划项集合，两两对比，遇到位置相同的则比对时间是否有冲突，如果有则返回true，表示存在位置独占冲突，否则返回false。一下为判断具体过程：



### 检测一组计划项之间是否存在资源独占冲突

首先判断传入的计划项列表中的计划项类型，Flight和Course可以直接判断是否有相同资源，而Train需要判断其两个计划项的资源列表中是否有重复出现的资源，之后将有相同资源或者重复资源的的计划项的时间进行比对，确定是否有资源独占冲突。

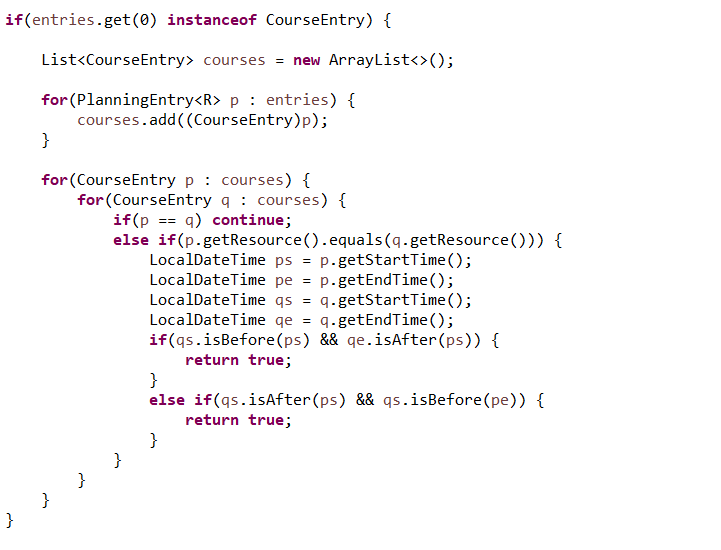
Flight的判断：



Train的判断：



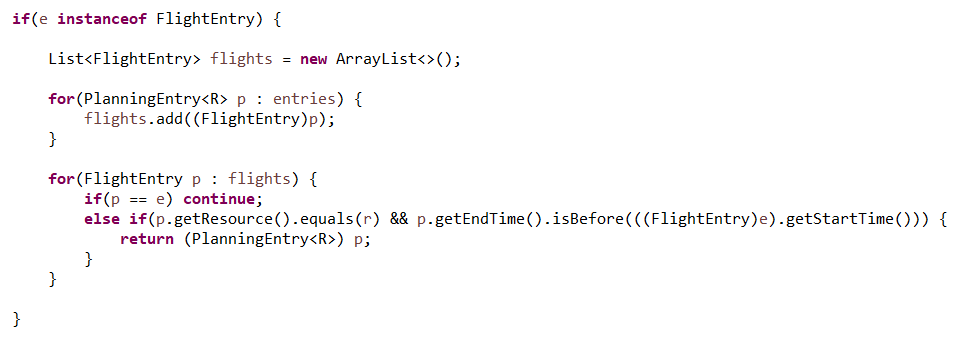
Course的判断：



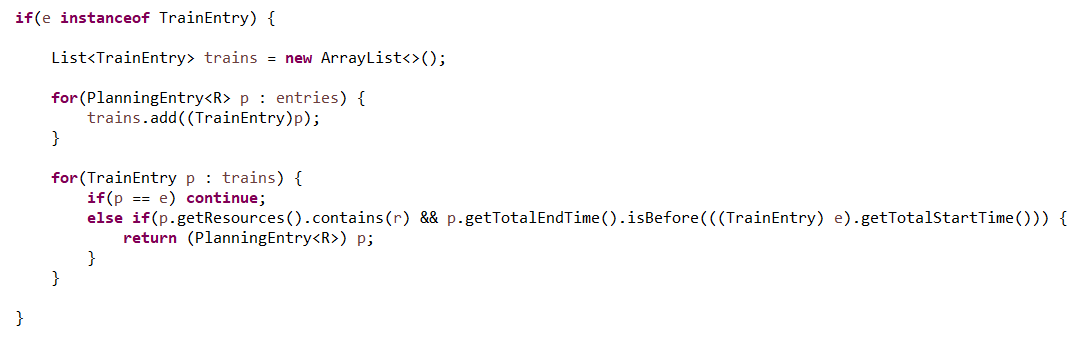
### 提取面向特定资源的前序计划项

先判断计划项的类型，之后在列表中寻找是否有相同资源的计划项，如果有则判断时间，如果有当前计划项的前置计划项则将其返回。

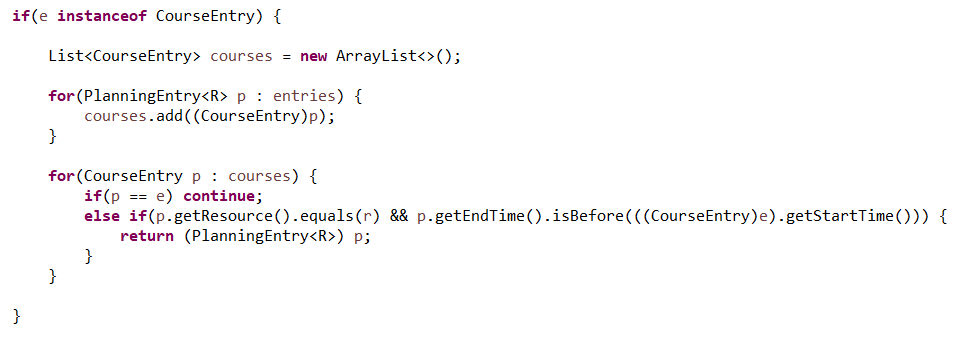
Flight的判断：



Train的判断：



Course的判断：



## 设计模式应用

### Factory Method

为三个具体Entry类实现工厂方法，三个工厂方法类均实现两个工厂方法，一个是直接传入Entry类的接口实现类，另一个是传入实现接口实现类需要的参数，在工厂方法中组装成接口实现类。具体实现如下：

FligthFactory：



TrainFactory：



CourseFactory：



### Iterator

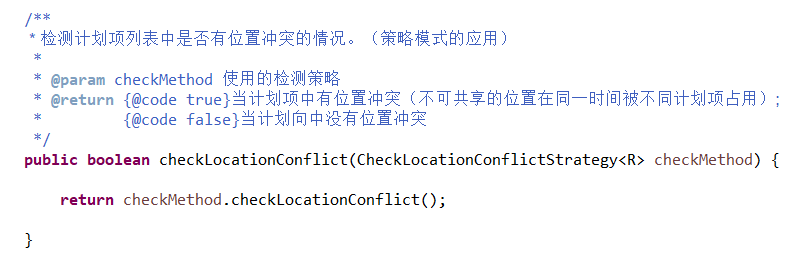
由于board的数据都放在了Data类中，因此原本要给board加的迭代器也就加到了Data类中。添加迭代器，首先要让Data类实现iterable接口，之后完成接口中的iterator方法，完成代码如下：



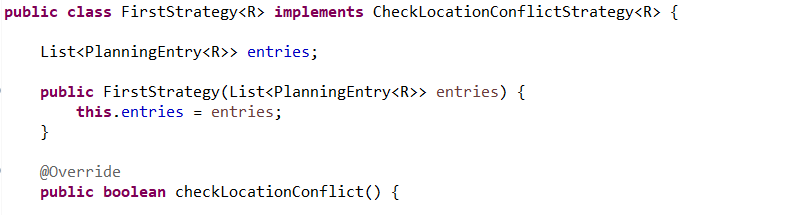
### Strategy

为此前的API设计实现不同的策略，我们这里选取的是为测试位置冲突实现不同的策略。

首先实现一个策略接口CheckLocationConflictStrategy，其中定义了策略中要用到的方法checkLocationConflict，这里的目的是统一不同的策略，使不同的策略能够被API中的方法调用。之后要重载API中的checkLocationConflict方法，使之能够接受参数列表为CheckLocationConflictStrategy类型的参数，重载的方法中调用策略类型中实现的checkLocationConflict方法。过程如下：



实现的具体策略类的大体实现如下所示：



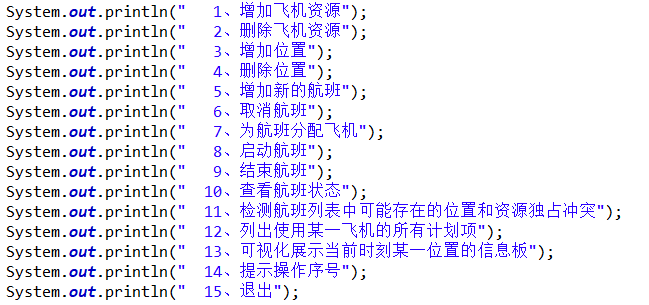
至此就可以调用策略方法了，客户端可以自由选择策略来进行判断，而达到的效果是相同的。

## 应用设计与开发

为了使app本身的操作整洁可观，因此将操作放到了app的外面，是为Operation。而三个operation均用到且可复用的操作进一步被提取出来，形成ScheduleOperation。ScheduleOperation中包含了对resource列表和location列表的操作，以及checkLocationAndResourseConflict()、getAllPlanWithResource(R resource)、viewBoardOfLoc(String locName)的抽象方法。ScheduleOperation的子类FlightSchedule、TrainOperation、CourseOperation主要调用了之前在Data类以及APIs还有Entry中的方法，在外面包装了条件判断，形成了各应用要用到的方法，供具体的app类来使用。

### 航班应用

通过命令行来进行操作，输入数字进行功能选择，若输入的数字满足条件，则进入相应的分支执行相应的操作，如有必要则提示要输入的信息以及格式。FlightScheduleApp中的操作如下所示：



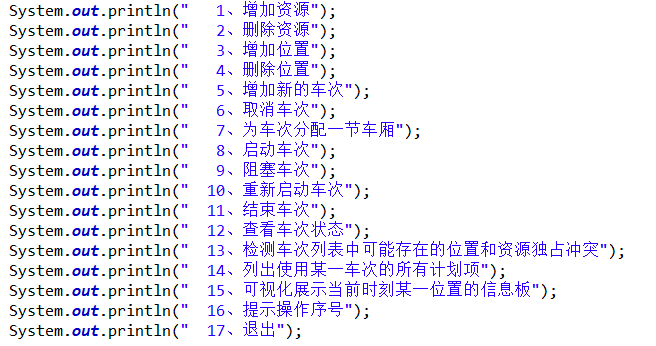
具体的操作流程如下所示（部分）：



### 高铁应用

与FlightScheduleApp的实现方式类似。

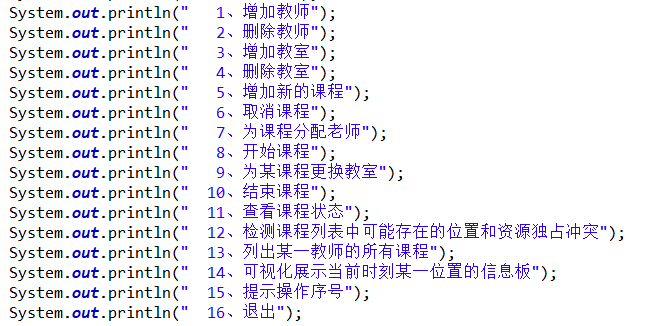
具体实现的功能为：



### 课表应用

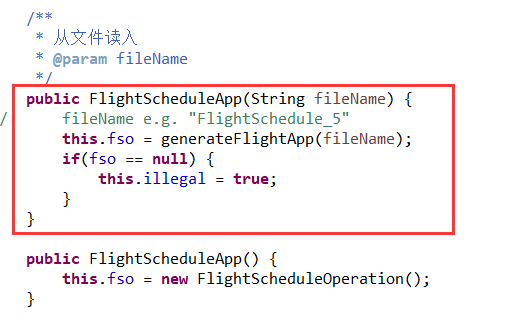
与FlightScheduleApp的实现方式类似。

具体实现的功能为：

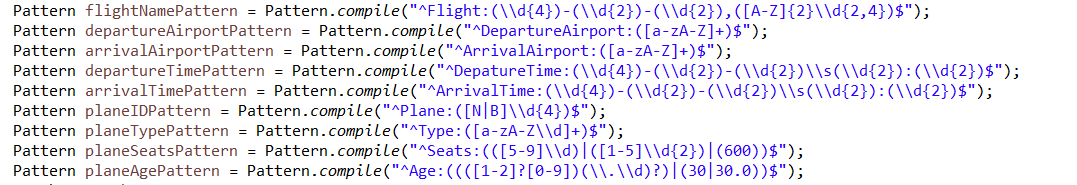


## 基于语法的数据读入

修改“航班”应用以扩展该功能。新增FlightScheduleApp的构造方法，可以从文件读入信息。



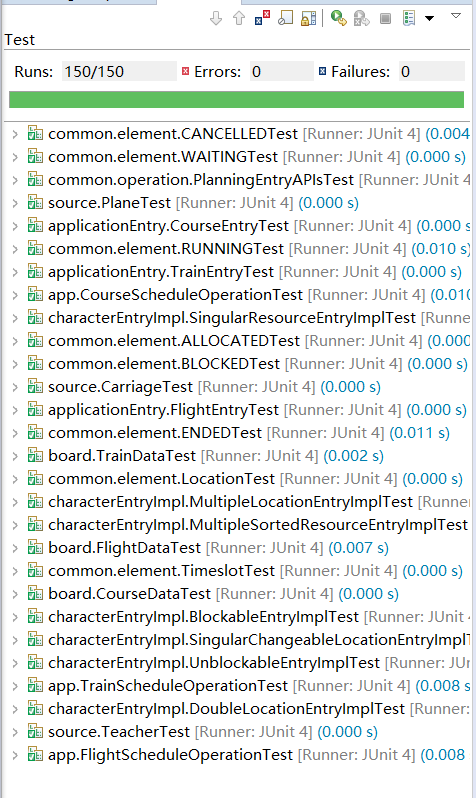
下面实现generateFlightApp方法，根据实验手册中的语法规则使用正则表达式来进行信息提取。



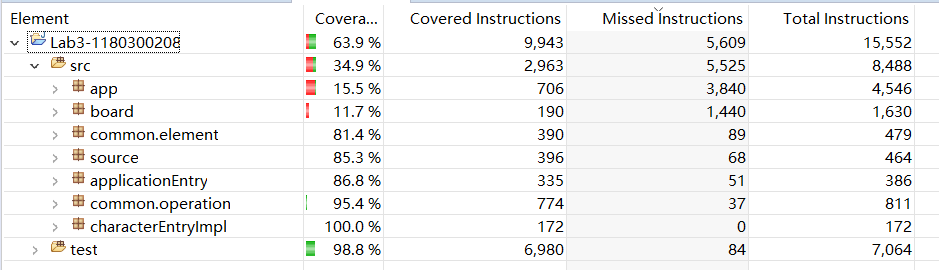
以上的正则表达式分别用来提取航班的日期、名称、起降机场、起降时间以及飞机的信息。由于给的文件13行构成一个Entry的完整信息，因此记录行数，以13为周期每次提取一行中的相应信息，同时对信息进行语法判断。最后与之前录入的信息进行比对，看是否有同样编号的航班的时间不相同、地点不同的情况，一旦发现上述过程有不符合语法的信息，则退出文件读取，同时提示输入正确的文件。判断文件中新读入的FlightEntry信息是否合法的方法如下所示，放在FlightScheduleOperation方法中。



此外，以上的类和方法进行了junit测试，测试均通过，测试结果如下：



测试覆盖度如下：



注意到对src的测试率达到了34.9%，其中app和board的测试率较低是由于app中有大量的客户端代码，而board中有大量的gui代码，其他类中覆盖度有一些在85%左右是因为有一些方法只是通过委托调用了已经测试的过的方法，已无测试必要，因此综上分析，测试基本全面。

## 应对面临的新变化

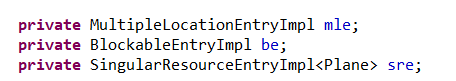
### 变化1 ：FlightEntry的改变

之前的设计可以应对变化，航班支持经停，那么在构造FlightEntry中是就应当用多地点和时间可阻塞的接口，由于需要支持经停，因此除了FlightEntry本身需要更改，其余的FlightScheduleApp、FlightScheduleOperation、FlightBoard都需要进行更改，工作量略大。

修改过程：

首先修改调用的接口，改变数据域中的接口实现类：





然后将随之发生改变的工厂方法以及FlightData以及FlightBoard类进行相应的修改，最后修改app中的提示以及执行过程。最后修改test中的相应测试即可。



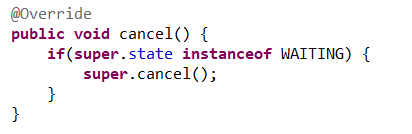
如图，实现了经停的功能。

### 变化2：TrainEntry的改变

之前的设计可以应对变化，当分配了车厢则车次不可取消，只需在取消的时候判断一下当前的状态，只有状态为WAITING时才可以取消计划，较为简单。

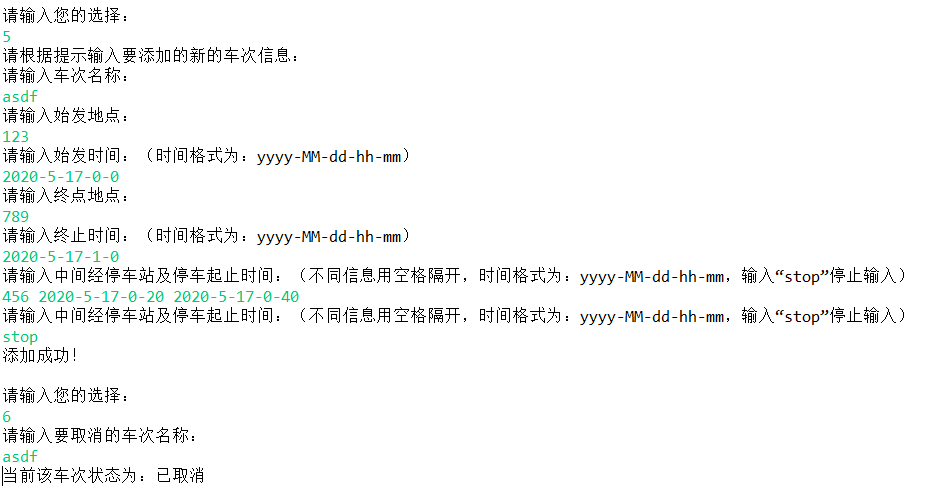
修改过程：

重写TrainEntry的cancel方法：



实现结果：

当创建了新的车次而没有分配车厢时，可以取消车次。



而当分配了车厢后，无法再将改车次取消。

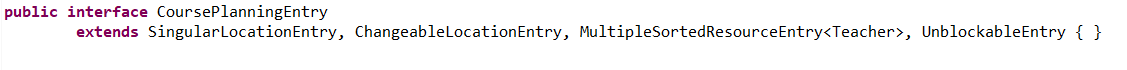


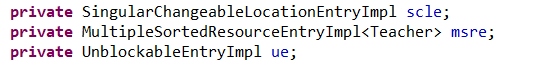
### 变化3：CourseEntry的改变

之前的设计可以应对变化，可以有多个教师一起上课且需要区分次序，那么将课程的资源接口改成与TrainEntry相同的MultipleSortedResourceEntry即可，相对工作量不大。

修改过程：

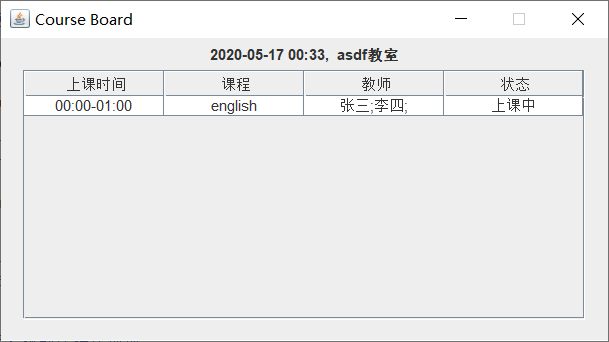
首先修改调用的接口，改变数据域中的接口实现类：





然后将随之发生改变的工厂方法以及CourseData以及CourseBoard类进行相应的修改。最后修改test中的相应测试即可。

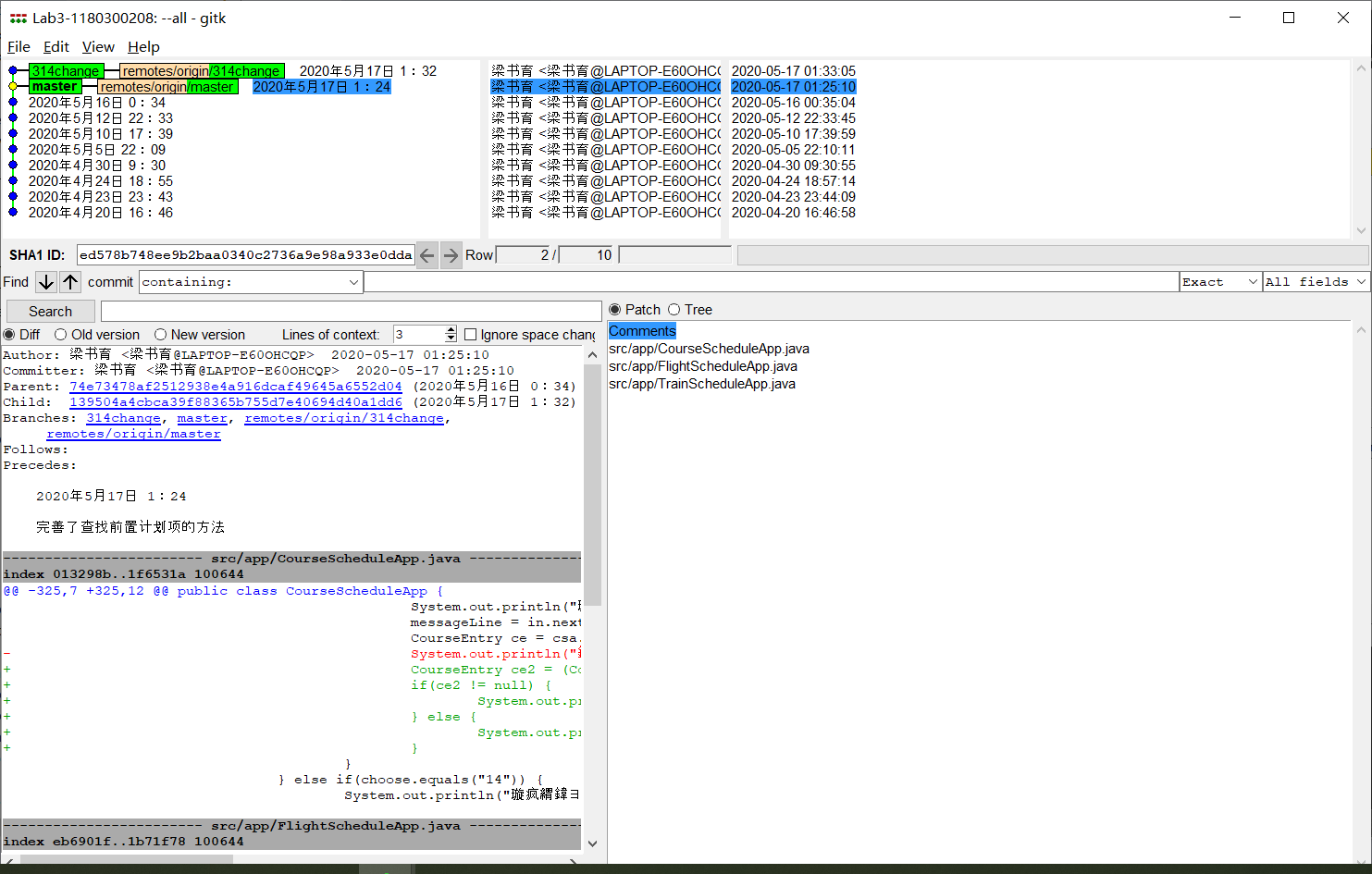
实现结果：



实现了多个教师按照次序进行上课。

## 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。

**

由Git图形化客户端可以查看到如上图所示的branch关系。

总结一下上图给出的信息，可以得到Object Graph（实验报告撰写时，doc目录还没有push）如下：

master

# 实验进度记录

314change

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。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2020/4/20 | 14:00-16:45 | 实现部分要用到的接口以及资源类 | 完成 |
| 2020/4/23 | 15:00-23:45 | 基本实现FlightBoard的代码 | 完成 |
| 2020/4/24 | 15:00-18:55 | 完成三个board的展示 | 完成 |
| 2020/4/30 | 07:00-09:00 | 实现各app的准备工作 | 完成 |
| 2020/5/5 | 13:00-22:00 | 完成基于语法的编程部分 | 完成 |
| 2020/5/10 | 15:00-17:35 | 基本完成app的编程 | 完成 |
| 2020/5/12 | 16:00-22:30 | 基本完成master线路 | 完成 |
| 2020/5/16 | 15:00-00:00 | 完成master的test以及纠正一些bug | 完成 |
| 2020/5/17 | 00:00-01:30 | 完成314change | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 正则表达式 | 到菜鸟教程上进行学习，在B站上看了一个讲解很透彻的视频<https://www.bilibili.com/video/BV1w4411C7Qp> 30分钟掌握正则表达式 |
| 时间表示 | 查找资料最终使用了LocalDateTime作为时间表示，因为这个类是immutable的，用起来更安全 |
| board表示时布局的问题 | 使用了很方便的BoxLayout进行布局 |
| 完成PlanningEntryAPIs时无法从PlanningEntry的层面上进行操作 | 用instanceof判断传入的数据为何种子类型，将PlanningEntry类型的数据强制转换成对应的子类型进行操作 |
| 在进行文件读取时，文件1原本由三万个entry数据但是只读入了两千多个 | 经过检查文件发现文件中有空行，导致文件读取提前终止，随即将文件读取过程改为了遇到空行就跳过，继续判断，完美解决问题 |

# 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

经验和教训：

1、在开始写代码前一定要做好计划，提前规划好方法的参数列表，防止在之后不断返工。

2、充分的测试很有必要。

3、针对像这次实验中的不同分支的编程应当提前做好备份防止误操作导致代码丢失。

## 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？

二者的差异在这侧实验中得到了更加充分的体现，对于多个相似应用场景的开发，面向ADT的编程的可复用性更强，在编程的过程中更加方便，效率更高，但是直接面向应用场景编程则可以省去抽象、归纳的过程，同时由于无需考虑可复用性，程序的运行效率可能会更高。在这次实验中，虽然设计ADT的过程很难，但实现app的过程非常高效，充分体现了复用的强大。

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？

这些工作的意义在于在最大程度上在编程的过程中及时发现问题，能够节省很多后期debug的时间。我愿意在以后的编程中这样做。

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：由于要考虑API的复用，因此API的设计一定要合理，能够适应编程的需要，而这也是API复用的难度所在。

乐趣：看到自己设计的API能够在很多看似关联不大的场景进行使用就十分开心。

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？

设计模式就相当于前人在编程过程中总结出的十分高效的编程模板，是符合科学的，使用设计模式毫无疑问可以增强我们的程序设计能力。

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？

正则表达式是一个十分强大的工具，可能在今后的编程中遇到识别提取文本信息时会优先想到正则表达式。

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？

难度主要体现在如何使ADT的设计充分的解耦但又不会出现组合爆炸的情况，而且设计ADT需要涉及到的类的数目也远远不是之前编程能够想象到的。克服方法：充分了解实验的要求，同时合理利用设计模式进行开发。

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？

接口可以说是设计中抽象度最高的层次，只定义了方法和参数列表，而抽象类中可以实现一些共性的方法，同时将接口中个性化的方法的实现继续交给下一级的子类来完成。而设计模式就是对于多层抽象如何能够方便快捷、易于理解。

1. 关于本实验的工作量、难度、deadline。

不愧是传说中的软件构造Lab3，工作量上应该已经超过了前几个实验的总和，代码量大约在7600行左右，难度上还是有一定的难度，ddl还算可以，可以不慌不忙完成。

1. 到目前为止你对《软件构造》课程的评价。

软件构造课程作为一个以实验为核心的课程，我本人还是非常喜欢的，可以学到不少东西，但是这次实验的安排上对于设计模式的使用上更多的是浮光掠影，个人感觉有些地方还可以更加完善。