

**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 | 周牧云 |
| 学号 | 1180300315 |
| 班号 | 1836101 |
| 电子邮件 | 1036314134@qq.com |
| 手机号码 | 13912263240 |

**目录**

[1 实验目标概述 1](#_Toc40570995)

[2 实验环境配置 1](#_Toc40570996)

[3 实验过程 1](#_Toc40570997)

[3.1 待开发的三个应用场景 1](#_Toc40570998)

[3.2 面向可复用性和可维护性的设计：PlanningEntry<R> 2](#_Toc40570999)

[3.2.1 PlanningEntry<R>的共性操作 2](#_Toc40571000)

[3.2.2 局部共性特征的设计方案 2](#_Toc40571001)

[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](#_Toc40571002)

[3.3 面向复用的设计：R 4](#_Toc40571003)

[3.4 面向复用的设计：Location 4](#_Toc40571004)

[3.5 面向复用的设计：Timeslot 5](#_Toc40571005)

[3.6 面向复用的设计：EntryState及State设计模式 5](#_Toc40571006)

[3.7 面向应用的设计：Board 6](#_Toc40571007)

[3.8 Board的可视化：外部API的复用 6](#_Toc40571008)

[3.9 PlanningEntryCollection的设计 7](#_Toc40571009)

[3.10 可复用API设计及Façade设计模式 8](#_Toc40571010)

[3.10.1 检测一组计划项之间是否存在位置独占冲突 8](#_Toc40571011)

[3.10.2 检测一组计划项之间是否存在资源独占冲突 8](#_Toc40571012)

[3.10.3 提取面向特定资源的前序计划项 9](#_Toc40571013)

[3.11 设计模式应用 9](#_Toc40571014)

[3.11.1 Factory Method 9](#_Toc40571015)

[3.11.2 Iterator 9](#_Toc40571016)

[3.11.3 Strategy 9](#_Toc40571017)

[3.12 应用设计与开发 10](#_Toc40571018)

[3.12.1 航班应用 10](#_Toc40571019)

[3.12.2 高铁应用 10](#_Toc40571020)

[3.12.3 课表应用 11](#_Toc40571021)

[3.13 基于语法的数据读入 11](#_Toc40571022)

[3.14 应对面临的新变化 12](#_Toc40571023)

[3.14.1 变化1 12](#_Toc40571024)

[3.14.2 变化2 13](#_Toc40571025)

[3.14.3 变化3 13](#_Toc40571026)

[3.15 Git仓库结构 13](#_Toc40571027)

[4 实验进度记录 13](#_Toc40571028)

[5 实验过程中遇到的困难与解决途径 14](#_Toc40571029)

[6 实验过程中收获的经验、教训、感想 14](#_Toc40571030)

[6.1 实验过程中收获的经验和教训 14](#_Toc40571031)

[6.2 针对以下方面的感受 15](#_Toc40571032)

# 实验目标概述

本次实验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

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

https://github.com/ComputerScienceHIT/Lab3-1180300315

# 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：

航班管理、高铁车次管理、大学课表管理

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：

都有时间段、地点、资源；

它们在哪些方面有差异：

地点的数量不同

资源的数量也不同

时间段的个数也不同

## 面向可复用性和可维护性的设计：PlanningEntry<R>

### PlanningEntry<R>的共性操作

创建一个新的计划项

获取计划的名字

启动

取消

挂起

结束

增加/删除一个资源

获取所有资源

增加/删除一个位置

获取所有位置

增加/删除一个时间段

获取所有时间段

获取开始时间

获取结束时间

### 局部共性特征的设计方案

**public** **static** <R> CommonPlanningEntry<R> getNewCommonPlanningEntry(String name)创建一个新的计划项

**public** String getName();获取计划的名字

**public** **boolean** start();启动计划项

将计划的状态转为running，不能转换会返回false

**public** **boolean** cancel();取消计划项

将计划的状态转为cancelled，不能转换会返回false

**public** **boolean** block();阻塞计划项

将计划的状态转为blocked，不能转换会返回false

**public** **boolean** complete();完成计划项

将计划的状态转为ended，不能转换会返回false

**public** State getState();获取计划项状态

**public** **boolean** addResource(R resource);增加一个资源resource

如果该资源已经加入了，会返回false

如果该计划不是waiting或者allocated状态，会返回false

加入资源后会将该计划状态改为allocated

**public** List<R> getResources();获取所有资源的集合

**public** **boolean** addLocation(Location location);增加一个位置

如果位置已经加入，会返回true

**public** Location getLocation(String locationname);通过名字找到位置信息

如果找不到会返回null

**public** **boolean** removeLocation(String locationname);删除一个位置

如果该位置没有找到会返回false

**public** List<Location> getLocations();获取所有位置的集合

**public** **boolean** addTimeslot(Timeslot timeslot);增加一个时间段

如果时间段已经存在会返回false

**public** List<Timeslot> getTimeslots();获取所有时间段的集合

**public** Time getBegintime();获取计划的开始时间

**public** Time getEndtime();获取计划的结束时间

**public** **int** compareTo(CommonPlanningEntry<R> o);比较两个计划项的早晚，用于后续迭代器使用

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

各种子类型的资源数量、地址数量、时间段数量略有差别

对于航班：

**public** **boolean** addPlane(String planename, String type, **int** seats, **double** age)

增加一架飞机需要飞机名、类型、座位数、机龄

**public** **boolean** addPlan(String planname, String from, String to, String begintime, String endtime)

增加一个航班需要航班名、出发机场、到达机场、起飞时间、到达时间

对于高铁管理:

**public** **boolean** addTrain(String name, String type, **int** seats, **int** year)

增加火车需要火车名、类型、座位数、出厂日期

**public** **boolean** addPlan(String planname, List<String> locations, List<String> times)

增加一个路线需要路线名、所有站点名、所有时间段

对于课程管理：

addTeacher(String name, String id, String gender, String work)

增加一个老师需要老师名、身份证号、性别、职称

addPlan(String planname, String locationname, String begintime, String endtime)

增加一节课需要课程名、教室名、上课时间、下课时间

**public** **boolean** changeclassroom(String planname, String locationname)

改变一节课的教室

先删除原教室，再加入新教室

注意如果该课不是计划中不能改变教室

## 面向复用的设计：R

**private** **final** String name;存储资源的名字

对于每种情况来说资源各不相同，只有名字是共性的。

**public** String getName() 返回资源的名字

**public** String toString() 将资源以字符串的形式打印

**public** **boolean** equals(Object resource) 通过名字来判断两个资源是否等价

## 面向复用的设计：Location

**private** **final** String name;储存位置的名字

位置信息只有名字一个属性即可

**public** String getName() 返回位置的名字

**public** **boolean** equals(Object o) 通过名字来判断两个位置是否等价

## 面向复用的设计：Timeslot

先设计Time类

该类型表示一个时间点

**private** **int** year;

**private** **int** month;

**private** **int** day;

**private** **int** hour;

**private** **int** minute;

一个时间点有年月日时分

**public** **int** getYear()

**public** **int** getMonth()

**public** **int** getDay()

**public** **int** getHour()

**public** **int** getMinute()

五个方法分别获得该时间的年月日时分

**public** **static** **boolean** isLegalTime(String t) 判断一个字符串是不是时间

标准字符表示为：年-月-日 时:分

**public** **void** addTime(Time time) 将当前时间加上一个时间

需要注意进位

**public** **boolean** sameday(Time time) 判断两个时间是否相同

**public** **int** compareto(Time time)比较两个时间的早晚

若当前时间更早，返回-1

若两个时间相同，返回0

若当前时间更晚，返回1

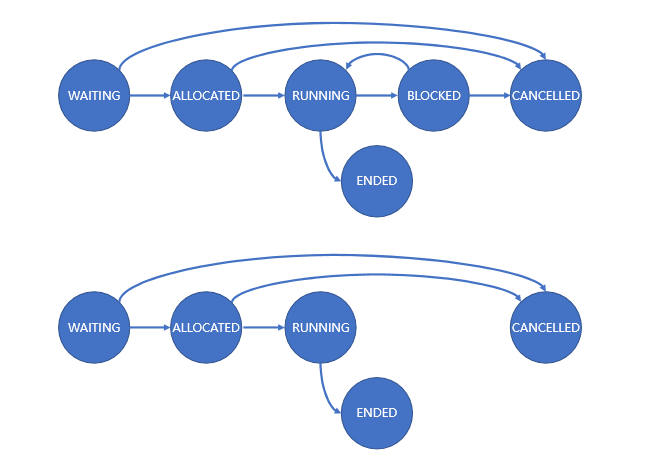
**public** String toString()将当前时间以字符串形式打印出来

## 面向复用的设计：EntryState及State设计模式

**private** String name;储存当前状态

**public** String getName()获取当前状态的名字

**public** **boolean** canchange(String name)判断当前状态能否转换到目标状态



## 面向应用的设计：Board

**public** **static** <R **extends** Resource> **void** Board(Time time, Location location, List<PlanningEntry<R>> plans)打印信息板

time 为当前时间

location为当前位置

plans为所有的计划项的集合

## Board的可视化：外部API的复用

CreateJTable(Time time, Location location, List<PlanningEntry<R>> lines)

使用javax.swing来进行可视化

**private** **static** JFrame *container*; 它是屏幕上window的对象，能够最大化、最小化、关闭

**private** **static** JTable *jTable*;它是一个表格

*jTable* = **new** JTable(data, columnName);，建立一个显示二维数组数据的表格，且可以显示列的名称。

具体的操作是找的网络上的模板

为了显示表头，我们需要将Jtable先放入一个容器中，之后让容器自动添加

*jTable* = **new** JTable(data, columnName);

JScrollPane jScrollPane = **new** JScrollPane();

jScrollPane.setViewportView(*jTable*);

*container* = **new** JFrame(time.toString() + " " + location.getName());

*container*.setSize(800, 600);

*container*.setLayout(**new** BorderLayout());

*container*.add(jScrollPane, BorderLayout.***CENTER***);

*container*.setVisible(**true**);

## PlanningEntryCollection的设计

**protected** **final** List<R> resources;存储所有资源

**protected** **final** List<Location> locations;存储所有位置

**protected** **final** List<PlanningEntry<R>> plans;存储所有计划

**public** **boolean** start(String planname) 启动指定计划

**public** **boolean** cancel(String planname) 取消指定计划

**public** **boolean** block(String planname)阻塞指定计划

**public** **boolean** complete(String planname)完成指定计划

**public** State getState(String planname)获取指定计划的状态

**public** R getResource(String name)获取指定名字的资源

如果找不到返回null

**public** **boolean** removeResource(String name)删除指定资源

如果该资源未加入则返回false

**public** **boolean** allocateResource(String resourcename, String planname)分配资源

如果资源或计划项不存在则返回false

具体做法就是将该资源加入到该计划项中即可

**public** **boolean** addLocation(String name)增加一个位置

如果该位置已加入则返回false

**public** Location getLocation(String name)获得位置信息

如果位置不存在返回null

**public** **boolean** removeLocation(String name)删除一个位置

如果位置不存在则返回false

**public** PlanningEntry<R> getPlan(String planname)获取计划项的信息

如果计划不存在返回null

**public** **boolean** removePlan(String planname)删除一个计划项

如果计划项不存在，或者该计划不处于waiting或allocated的状态，则返回false

**abstract** **public** **void** showBoard(String time, String location);打印信息板

**public** **boolean** checkLocationConflict(**boolean** f)检查位置冲突

**public** **boolean** checkResourceConflict()检查资源冲突

**public** List<PlanningEntry<R>> getResourceEntries(String resourceName)获取使用某个资源的所有计划项

**public** PlanningEntry<R> getPreEntry(String resourceName, String entryName)获取使用某资源的前序计划项

**public** **boolean** checkResourceUsing(String resourcename)检查资源是否已被指派

**public** **boolean** checkLocationUsing(String locationname)检查位置是否已被指派

## 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

checkLocationConflict(List<PlanningEntry<R>> entries, **boolean** f)

由于只有课表会存在位置冲突，而高铁与航班都不会有冲突，所以先通过排除存在一个以上位置的计划项来排除高铁与航班的情况。

然后，对所有计划项进行o(n2)的遍历，如果有两个计划项它们占用了同一个位置且时间段有重叠，则判断有冲突，否则为无冲突

### 检测一组计划项之间是否存在资源独占冲突

checkResourceExclusiveConflict(List<PlanningEntry<R>> entries)

遍历计划项每两个元素，如果两个计划项有共同资源，判断它们的时间段是否有重叠，如果有则存在冲突，否则为无冲突

### 提取面向特定资源的前序计划项

indPreEntryPerResource(R r, PlanningEntry<R> e, List<PlanningEntry<R>> entries)

首先对计划项集合中的每个元素进行遍历，将所有早于计划e且存在资源r的计划项提取出来存放到一个新列表中，然后返回该列表中的一个元素即可

## 设计模式应用

### Factory Method

以PlanningEntry为例：

**private** CommonPlanningEntry(String name) {

**this**.name = name;

**this**.state= State.*getNewState*("WAITING");

**this**.timeslots = **new** ArrayList<>();

**this**.resources = **new** ArrayList<>();

**this**.locations = **new** ArrayList<>();

checkRep();

}

**public** **static** <R> CommonPlanningEntry<R> getNewCommonPlanningEntry(String name) {

**return** **new** CommonPlanningEntry<R>(name);

}

对每一个类，均是通过一个creater方法获取实例

Creater方法接受创建新子类所需的参数，然后在内部调用new生成实例，再返回，从而实现不在外部调用new。

### Iterator

该迭代器对于一个PlanningEntry的List，按照开始时间的顺序遍历其中的任务。

**public** PlanningEntryIterator(Map<PlanningEntry<R>, List<E>> map)

首先获取遍历的对象的大小，之后构建所用的迭代器。

### Strategy

checkLocationConflict(List<PlanningEntry<R>> entries, **boolean** f)

对于位置冲突检查，我使用两种策略

当f为true时调用第一种策略

checkLocationConflict(List<PlanningEntry<R>> entries)

遍历每两个计划项，先判断两个计划是否占用同一个位置

如果是，再判断两个计划时间是否有冲突

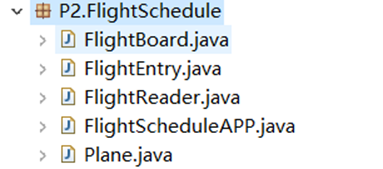
当f为false时调用第二种策略

遍历每两个计划项，先判断两个计划的时间段是否有冲突

若有，再判断两个计划项是否占用了同一个位置

## 应用设计与开发

### 航班应用



FlightEntry是PlanningEntryCollection的子类，增加的内容有

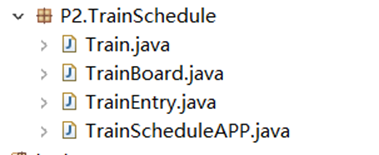
**public** **boolean** addPlane(String planename, String type, **int** seats, **double** age)

增加一架飞机需要飞机名、类型、座位数、机龄

**public** **boolean** addPlan(String planname, String from, String to, String begintime, String endtime)

增加一个航班需要航班名、出发机场、到达机场、起飞时间、到达时间

### 高铁应用



TrainEntry是PlanningEntryCollection的子类，增加的内容有

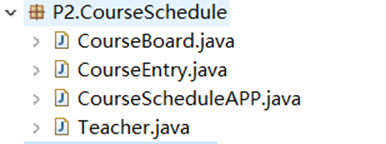
**public** **boolean** addTrain(String name, String type, **int** seats, **int** year)

增加火车需要火车名、类型、座位数、出厂日期

**public** **boolean** addPlan(String planname, List<String> locations, List<String> times)

增加一个路线需要路线名、所有站点名、所有时间段

### 课表应用



CoursetEntry是PlanningEntryCollection的子类，增加的内容有

addTeacher(String name, String id, String gender, String work)

增加一个老师需要老师名、身份证号、性别、职称

addPlan(String planname, String locationname, String begintime, String endtime)

增加一节课需要课程名、教室名、上课时间、下课时间

**public** **boolean** changeclassroom(String planname, String locationname)

改变一节课的教室

先删除原教室，再加入新教室

注意如果该课不是计划中不能改变教室

## 基于语法的数据读入

先将文件一行一行全部读入，跳过空白行

从文件中可知每一个航班均占13行，于是将文件以十三行为一个循环进行处理。

对每一个航班信息

先判断第一行是否合法

**if** (Pattern.*matches*("^Flight:(.\*?)", l) == **false**)

如果合法就记录航班名字信息

判断第三行是否合法

**if** (Pattern.*matches*("DepartureAirport:(.\*?)", l) == **false**)

如果合法记录出发机场信息

判断第四行是否合法

**if** (Pattern.*matches*("ArrivalAirport:(.\*?)", l) == **false**)

如果合法记录抵达机场信息

判断第五行是否合法

**if** (Pattern.*matches*(

"DepatureTime:((\\d{4})-(([0][1-9])|(1[012]))-((0[1-9])|([12]\\d)|30) (([01]\\d)|(2[0-4])):(([0-5]\\d)|60))",

l) == **false**)

如果合法则记录出发时间

判断第六行是否合法

**if** (Pattern.*matches*(

"ArrivalTime:((\\d{4})-(([0][1-9])|(1[012]))-((0[1-9])|([12]\\d)|30) (([01]\\d)|(2[0-4])):(([0-5]\\d)|60))",

l) == **false**)

如果合法记录到达时间

判断第7、9、10、11行是否合法

**if** (Pattern.*matches*("Plane:(B|N)(\\d{4})", l) == **false**)

**if** (Pattern.*matches*("Type:([a-zA-Z0-9]+)", l) == **false**)

**if** (Pattern.*matches*("Seats:(([5-9][0-9])|([1-4]\\d{2})|500)", l)

**if** (Pattern.*matches*("Age:(30|30.0|([1-2]?[0-9]([.][0-9])?))", l)

如果合法，记录飞机信息

再判断出发日期与航班名中的日期是否一致

最后，将所有信息加入计划项中，再将该计划项加入计划项集合中

读完文件后返回计划项集合

如果有任何一步不合法，均会返回null，提示文件内容有误

## 应对面临的新变化

为了将变化分离，提高代码的可扩展性与可维护性，分别声明了FlightEntry、TrainEntry、CourseEntry的子类，命名为FlightChange、TrainChange、CourseChange，放在applications.change包下。

之后，在客户端将声明的entry类型换为相应的change类型。

### 变化1

在FlightEntry中增加一个可添加中转航班的addPlan

addPlan(String planname, List<String> locations, List<String> times)

更改了Board中判断抵达的方法，将中转站也加入判断中

更改了Board中判断航班状态的部分，将经停状态加入其中

### 变化2

在取消车次部分加入一个判断就行了：

System.***out***.println("请给出要取消的路线名称");

String line = in.next();

**if**(lines.getPlan(line).getResources().isEmpty() == **false**) {

System.***out***.println("已被分配车厢资源，不能取消");

**break**;

}

### 变化3

在Board中更改显示就行了

data[i][2] = "";

**int** max = courses.get(i).getResources().size();

**for**(**int** j = 0; j < max; j++) {

data[i][2] = data[i][2] + " " + courses.get(i).getResources().get(j).getName();

}

## Git仓库结构

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 4.14 | 18:30– 20:15 | 完成PlanningEntry接口和CommonPlanningEntry | 完成 |
| 4.18 | 14:00-18:00 | 完成State, Timeslot, Resource, Location类的设计 | 完成 |
| 4.21 | 18:30– 22:00 | 完成PlanningEntryCollection | 未完成 |
| 5.4 | 11:00-22:00 | 完成所有子类的设计 | 完成 |
| 5.5 | 18:30-20:15 | 完成PlanningEntryAPIs | 完成 |
| 5.12 | 18:30-20:15 | 完善所有Board类,完成Iterator，strategy | 完成 |
| 5:19 | 18:30-20:15 | 完成除3.14的所有内容 | 完成 |
| 5:23 | 18:30-23:50 | 完成3.14以及实验报告的撰写 | 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 |
| 不知道如何实现可视化 | 上网查找资料学了一些 |
| 无法比较两个时间点 | 将时间单独拉出来作为了一个类，建立了比较时间的方法 |
| 位置的添加出现了重复 | 需要重写equals来完成判断或者用getName()方法来进行判断 |
| 每种子类的资源添加需要的参数不一致，无法用接口实现 | 用抽象接口，添加资源全部由子类实现 |
| 文件读取的内容不正确 | 检查发现正则表达式书写错误 |
| 改变位置不成功 | 最后换了种实现方式，将旧位置计划删除，重新添加一个存有新位置的计划 |

# 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：

如果共性操作写得足够好的话，在后续写更多的应用会变得很轻松，反之则会非常类。

教训：

静态方法是不能被重写的

要修改某个类的域不能想当然修改get出来赋给临时变量的值。

## 针对以下方面的感受

1. 重新思考Lab2中的问题：面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？本实验设计的ADT在五个不同的应用场景下使用，你是否体会到复用的好处？

面先ADT编程，代码的模块化程度高，可复用性好，尤其是当要开发多个场景时，事半功倍。而场景编程在面对单一场景时灵活性更高，可以不用顾虑太多而去添加新功能。

能体会到。

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？

可以让程序员更快地理解代码，提高编程效率。

在多人协作时想必也会极大地遍历程序员之间的代码交流

愿意

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？

难处是自己的想法未必是正确的，或者自己设计出来的ADT质量非常差。

乐趣是自己设计的ADT可以完全按照自己的意愿进行修改

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？

我认为为了代码的可复用性和可维护性，使用设计模式是有必要的，即使增加了很多类，但相比带来的好处，这些代价是值得的。

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？

对于程序员来说工作量很大，但对客户来说提高了软件的便利性

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？

难点就是自己的想法未必可行，或者想不出比较好的解题思路。最终代码当中还是有不少重复代码，但是并没有能力进一步将其抽象出来。

网上找了很多资料，包括也自学了swing的内容。

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？

要尽可能地抽象出更多的内容才能为后续的工作减少压力。

1. 关于本实验的工作量、难度、deadline。

工作量对我而言略大，难度也比较高，好在时间够长，否则很可能完不成。

1. 到目前为止你对《软件构造》课程的评价。

感觉是比较难学的一门课，但是收获很大。