

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 魏志豪 |
| 学号 | 1190302020 |
| 班号 | 1903009 |
| 电子邮件 | [1207410841@qq.com](mailto:1207410841@qq.com) |
| 手机号码 | 15209945766 |

**目录**

[1 实验目标概述 1](#_Toc74666198)

[2 实验环境配置 1](#_Toc74666199)

[3 实验过程 1](#_Toc74666200)

[3.1 待开发的三个应用场景 1](#_Toc74666201)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 1](#_Toc74666202)

[3.2.1 IntervalSet<L>的共性操作 1](#_Toc74666203)

[3.2.2 局部共性特征的设计方案 2](#_Toc74666204)

[3.2.3 面向各应用的IntervalSet子类型设计（个性化特征的设计方案） 2](#_Toc74666205)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 2](#_Toc74666206)

[3.3.1 MultiIntervalSet<L>的共性操作 2](#_Toc74666207)

[3.3.2 局部共性特征的设计方案 2](#_Toc74666208)

[3.3.3 面向各应用的MultiIntervalSet子类型设计（个性化特征的设计方案） 2](#_Toc74666209)

[3.4 面向复用的设计：L 2](#_Toc74666210)

[3.5 可复用API设计 2](#_Toc74666211)

[3.5.1 计算相似度 2](#_Toc74666212)

[3.5.2 计算时间冲突比例 2](#_Toc74666213)

[3.5.3 计算空闲时间比例 2](#_Toc74666214)

[3.6 应用设计与开发 2](#_Toc74666215)

[3.6.1 排班管理系统 3](#_Toc74666216)

[3.6.2 操作系统的进程调度管理系统 3](#_Toc74666217)

[3.6.3 课表管理系统 3](#_Toc74666218)

[3.7 基于语法的数据读入 3](#_Toc74666219)

[3.8 应对面临的新变化 3](#_Toc74666220)

[3.8.1 变化1 3](#_Toc74666221)

[3.8.2 变化2 3](#_Toc74666222)

[3.9 Git仓库结构 3](#_Toc74666223)

[4 实验进度记录 3](#_Toc74666224)

[5 实验过程中遇到的困难与解决途径 4](#_Toc74666225)

[6 实验过程中收获的经验、教训、感想 4](#_Toc74666226)

[6.1 实验过程中收获的经验和教训 4](#_Toc74666227)

[6.2 针对以下方面的感受 4](#_Toc74666228)

# 实验目标概述

本次实验覆盖课程第 4-11 讲的内容，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：

1 子类型、泛型、多态、重写、重载

2 继承、代理、组合

3 语法驱动的编程、正则表达式

4 API 设计、API 复用

本次实验给定了三个具体应用（值班表管理、操作系统进程调度管理、大学课表管理），学生不是直接针对每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

实验环境已在lab1和lab2中配置完成。

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

https://github.com/ComputerScienceHIT/HIT-Lab3-1190302020

# 实验过程

## 待开发的三个应用场景

1 值班表管理

该应用创建一个可以维护的值班表，能够向其中添加、删除员工，并能手动或自动的进行排班，还可以实时的查看员工的排班情况。

2 操作系统进程调度管理

该应用模拟单独的CPU运行时的进程管理和控制流的切换，在进程执行到最大执行时间之前，自主（这里有两种策略，一种是随机选择进程策略，另一种是最短进程优先策略）地选择新的进程来执行，在所有进程都结束后可以查看进程运行的轨迹。

3 大学课表管理

模拟大学课表的设置与管理，可以在同一时段设置多节课，只需设置第一周的课程，后续几周的课程就会周期性的添加，在课表安排完成后还可以查看与课程安排的相关信息。

4 相同之处：

这三个应用场景的基本功能都是在时间轴上进行的，用户通过在时间轴上插入元素来达到使用应用的目的。

5 不同之处：

在值班表管理应用中，每个对象在时间轴上占用的时间必须是连续的且不能够有重叠和空隙；在操作系统进程调度管理应用中，每个对象在时间轴上占用的时间可以是不连续的且可以有空隙，但不能有重叠；在大学课表管理应用中，每个对象在时间轴上可以不连续，可以重叠，可以有空隙，但必须是周期性出现的。

## 面向可复用性和可维护性的设计：IntervalSet<L>

### IntervalSet<L>的共性操作

/\*\*

\* 给label分配一段时间，且label不能重复

\* **@param** start 起始时间

\* **@param** end 结束时间

\* **@param** label 标签

\*/

**public** **void** insert(**long** start,**long** end,L label);

/\*\*

\* 返回所有标签的集合

\* **@return**

\*/

**public** Set<L> labels();

/\*\*

\* 移除label及其对应的时间段

\* **@param** label 标签

\* **@return** 若label存在返回true；若label不存在返回false

\*/

**public** **boolean** remove(L label);

/\*\*

\* 得到label对应的起始时间

\* **@param** label 标签

\* **@return** 若label存在返回label的起始时间；若label不存在返回-1

\*/

**public** **long** start(L label);

/\*\*

\* 得到label对应的结束时间

\* **@param** label 标签

\* **@return** 若label存在返回label的结束时间；若label不存在返回-1

\*/

**public** **long** end(L label);

/\*\*

\* 得到最小时间

\* **@return**

\*/

**public** **long** getmin();

/\*\*

\* 得到最大时间

\* **@return**

\*/

**public** **long** getmax();

/\*\*

\* 返回一个包含当前时间的label的集合

\* **@param** currenttime

\* **@return**

\*/

**public** Set<L> containtime(**long** currenttime);

/\*\*

\* 返回一个按起始时间从小到大顺序排序的label列表

\* **@return**

\*/

**public** List<MyMap<L>> arrangestart();

/\*\*

\* 删除所有元素

\*/

**public** **void** removeall();

### 局部共性特征的设计方案

由于IntervalSet<L>是不容许同一元素多次出现的分隔集，所以每个标签都对应有唯一的时间段，所以就可以得到对应标签的开始时间和结束时间，又因为IntervalSet是最基础的插入的接口，所以它应该有前面所说的三个应用的共性特点，不能带有特殊性（即对元素的插入不做要求，除了前面所说的同一个元素只能出现一次）。

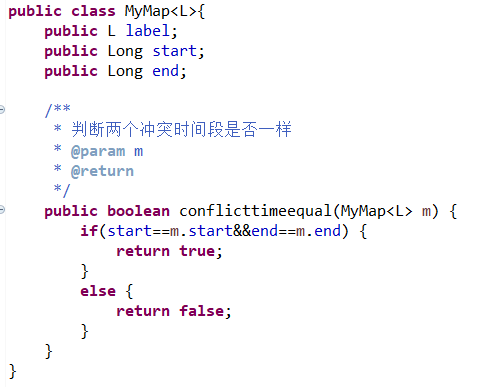
### 面向各应用的IntervalSet子类型设计（个性化特征的设计方案）

我使用的是实验指导书中的第6种设计方式：decorator 设计模式，具体设计方案如下：

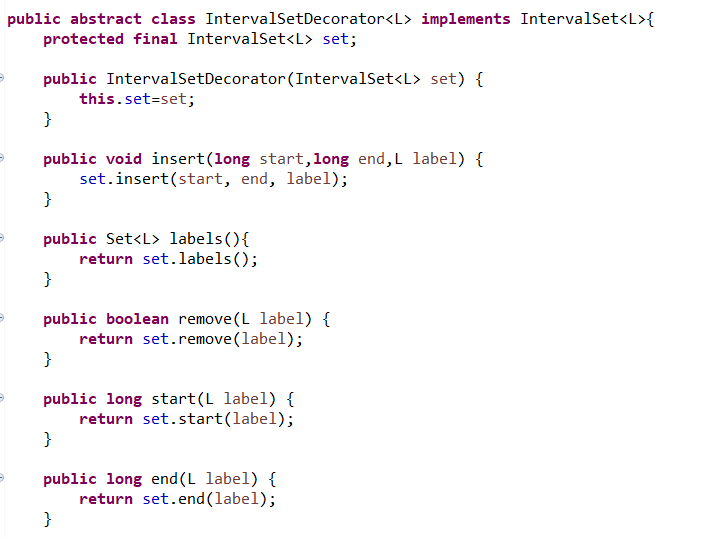
1 使用类CommonIntervalSet实现IntervalSet接口，作为IntervalSet的基础类，来实现最共性的操作。CommIntervalSet里的所有方法如下图所示：



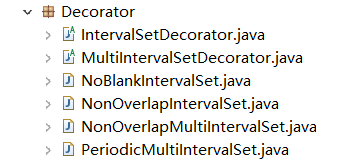
MyMap是我为了方便自己定义的一个ADT，它有着与Map类似的功能，它是一个label对应两个时间start和end，并且它里面所有的元素都是公有的可以直接访问和使用。



2 使用抽象类IntervalSetDecorator作为Decorator的基础，利用委托机制使用一个IntervalSet（其实就是CommonIntervalSet）的方法来实现接口IntervalSet里的所有方法。



3 如下图所示，里面的除IntervalSetDecorator的其他类都是IntervalSetDecorator的子类，它们继承抽象类IntervalSetDecorator里的方法，并在其基础上实现某种特有的功能。



例如类NonOverlapIntervalSet，它在基础方法上又添加了插入时不能重叠的特性，若发生重叠会报错并插入失败，这就是一层装饰，在使用时只用通过构造函数一层层的包装即可实现想要的特性



## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

/\*\*

\* 给label分配一段时间

\* **@param** start 起始时间

\* **@param** end 结束时间

\* **@param** label 标签

\*/

**public** **void** insert(**long** start,**long** end,L label);

/\*\*

\* 返回所有标签的集合

\* **@return**

\*/

**public** Set<L> labels();

/\*\*

\* 移除label及其所有的时间段

\* **@param** label 标签

\* **@return** 若label存在返回true；若label不存在返回false

\*/

**public** **boolean** remove(L label);

/\*\*

\* 得到label对应的所有时间段

\* **@param** label 标签

\* **@return** 时间段从小到大排列

\*/

**public** IntervalSet<Integer> intervals(L label);

/\*\*

\* 得到最小时间

\* **@return**

\*/

**public** **long** getmin();

/\*\*

\* 得到最大时间

\* **@return**

\*/

**public** **long** getmax();

/\*\*

\* 返回一个包含当前时间的label的集合

\* **@param** currenttime

\* **@return**

\*/

**public** Set<L> containtime(**long** currenttime);

/\*\*

\* 返回一个按起始时间从小到大顺序排序的label列表,同一个label可能重复出现

\* **@return**

\*/

**public** List<MyMap<L>> arrangestart();

/\*\*

\* 删除所有元素

\*/

**public** **void** removeall();

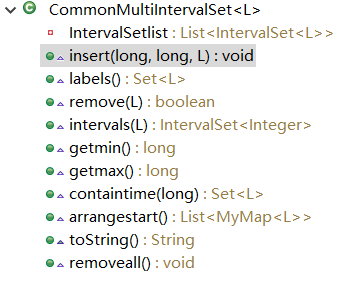
### 局部共性特征的设计方案

MultiIntervalSet容许同一个元素多次出现，但是同一个元素的时间段不能发生重叠，若发生重叠会将两段时间和为一段。这个接口也代表该种类型的最共性的操作，所以在插入的其他方面并不做限制。

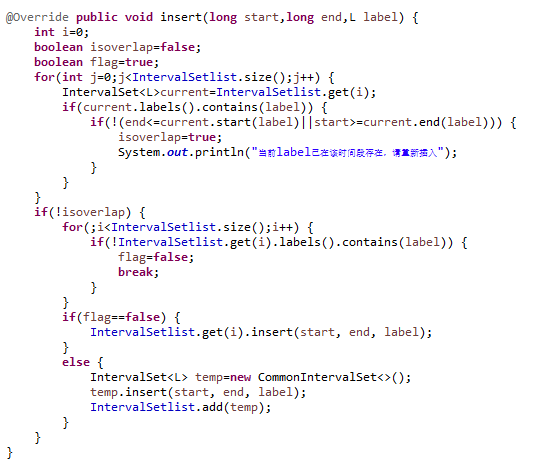
### 面向各应用的MultiIntervalSet子类型设计（个性化特征的设计方案）

MultiIntervalSet的子类型的设计和IntervalSet的子类型的设计方式一模一样，只是在具体实现时有些许差别。具体设计方案如下：

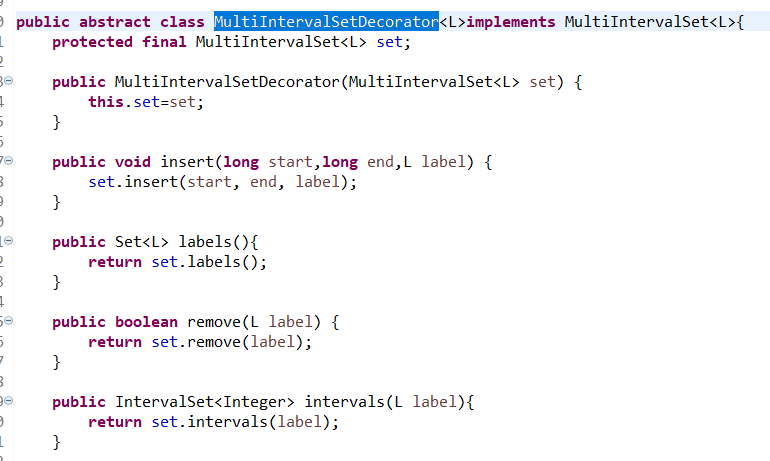
1 使用CommonMultiIntervalSet类实现MultiIntervalSet的最基础的操作，其具体内容如下：



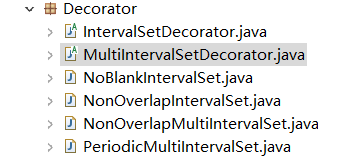
在这里面使用了之前实现的IntervalSet做rep，并在方法的实现过程中使用了IntervalSet中的方法，例如insert实现时，首先遍历已经存在的IntervalSet，寻找是否有某个IntervalSet里没有该label标签对应的元素，若有这样的IntervalSet则，将label插入到找到的IntervalSet；若找不到这样的IntervalSet，则新建一个IntervalSet将label插入其中，并将新建的IntervalSet添加到IntervalSetlist当中。Insert的具体实现如下：



2 使用抽象类MultiIntervalSetDecorator作为所有装饰类的基础。



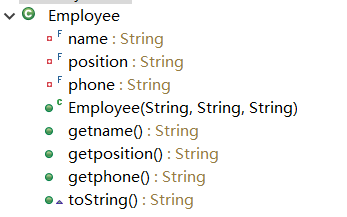
3 根据功能要求设计具体的装饰类。具体方法与IntervalSet的装饰类的设计方法类似。



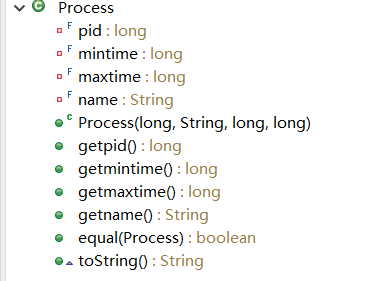
## 面向复用的设计：L

L是泛型，它可以被替换为其他的ADT，例如后面应用中用到的员工类Employee、进程类Process和课程类Course。当然在替换的时候要保证应用中设计的那几个类是不可变的数据类型。

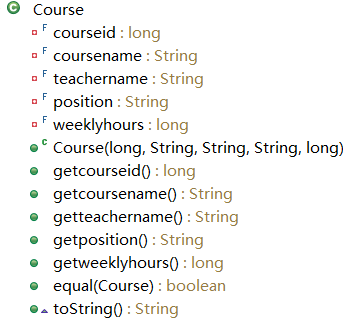
Empolyee的具体实现：



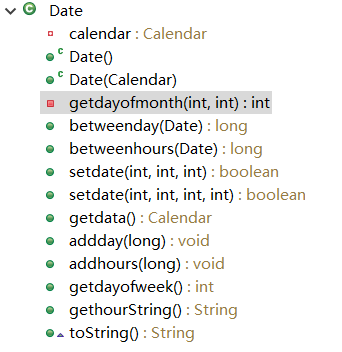
Process的具体实现：



Course的具体实现：

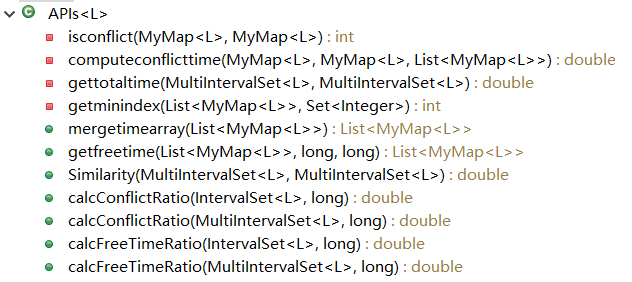


除了上述的几种ADT，我还设计了一些CommonADT，像之前所说的MyMap，还有Date，Date是一个表示时间的ADT，它有着计算两个日期之差，增加天数，转换成字符串等许多操作，该ADT在应用1和应用3中都会频繁的用到，它具体的构造如下图所示：

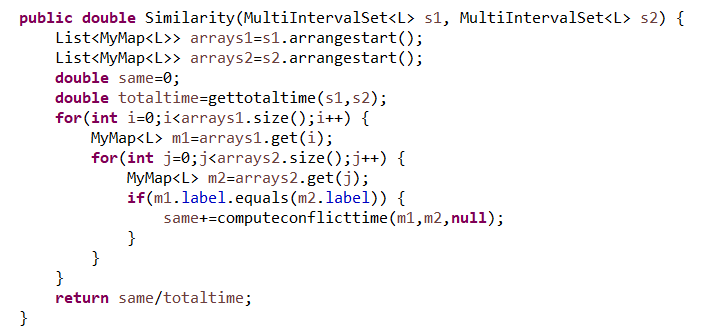


## 可复用API设计

API中除了要求实现的那几个功能还有一些为实现要求功能而定义的私有方法，它们都是为实现要求功能而服务的。API的具体构造如下图所示：

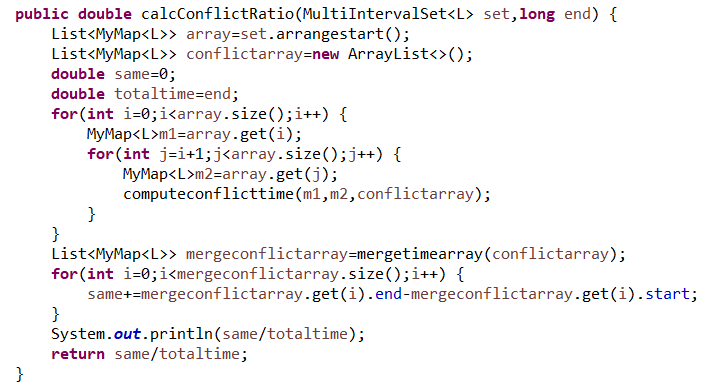


### 计算相似度



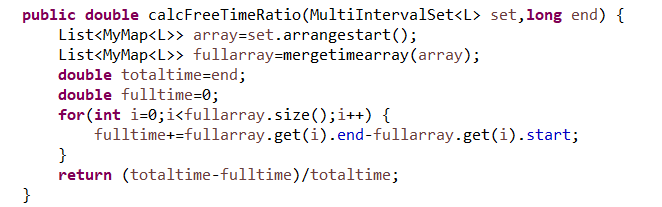
arrays1、arrays2是通过MultiIntervalSet里arrangestart方法得到的开始时间从小到大的list，并以MyMap为元素，totaltime是两个时间段的总时间，之后按照时间轴从早到晚的次序，针对同一个时间段内两个对象里的 interval，若它们标注的 label 等价，则二者相似度为 1，否则为 0；若同一时间段内只有一个对象有 interval 或二者都没有，则相似度为 0。将各interval 的相似度与 interval 的长度相乘后求和，除以总长度，即得到二者的整体相似度。

### 计算时间冲突比例



end代表截止时间，即在0到end这一时间段上计算时间冲突比例，mergetimaearray顾名思义，就是将时间轴上的重合的时间合并起来，防止多次重复计算重合的时间段，最后用same/totaltime即可得到冲突时间在给定时间段上的比例。

### 计算空闲时间比例

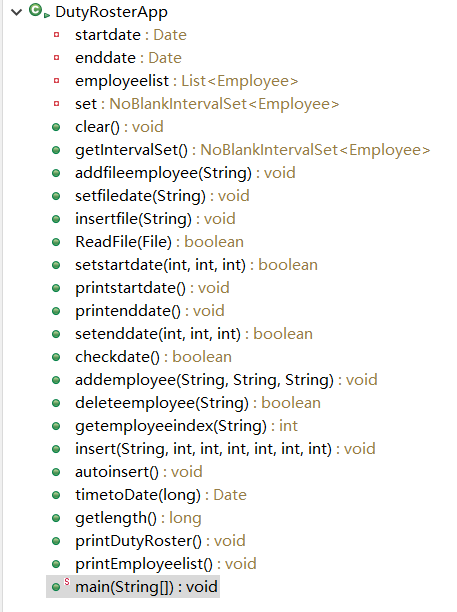


end代表截止时间，即在0到end这一时间段上计算空闲时间的比例，使用mergetimaearray将时间轴上使用的时间段合并起来，之后先计算使用的时间再用总时间减去使用的时间即可得到空闲的时间，最后除totaltime即可得到空闲时间在给定时间段上的比例。

## 应用设计与开发

### 排班管理系统

排班管理系统的具体实现如下图所示：

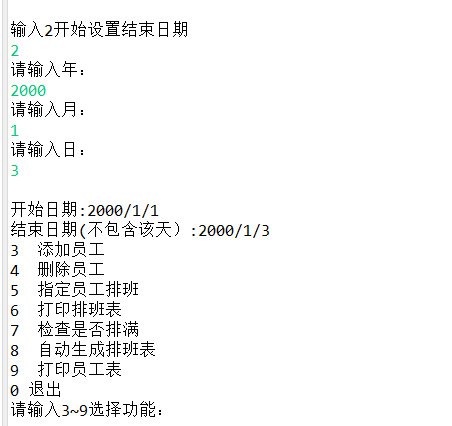


其中main是客户端程序，用户运行客户端之后通过输入数字来选择自己要使用的功能，其中使用的分隔集的包装如下图所示，根据要求我们可以指导排班表里员工的值班时间必须是连续的，所以我们使用CommonIntervalSet作为基础类，之后又知道排班时间是不能够重合的，即一天只有一人值班，所以我又用NonOverlapIntervalSet来装饰CommonIntervalSet，最后由于每天都得有人值班，是不能有空闲时间段的，所以我又使用NoBlankIntervalSet来包装NonOverlapIntervalSet，这样通过两层装饰的添加，就实现了排班表所需要的功能特性。

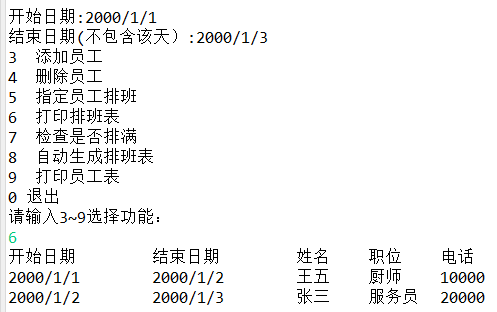


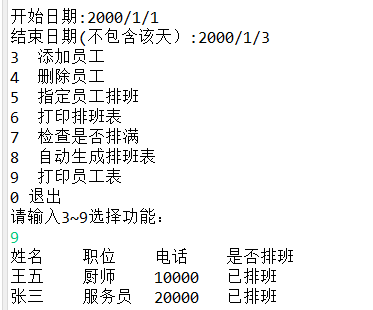
之后具体功能的实现就调用我们包装好的分隔集set里的方法就很容易实现了。

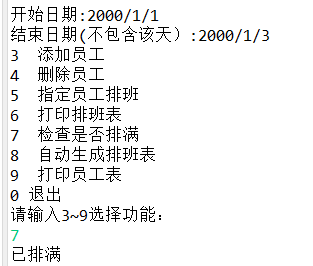
运行时的图片：



（在这里我手动添加了两个员工，并使用自动排班了）

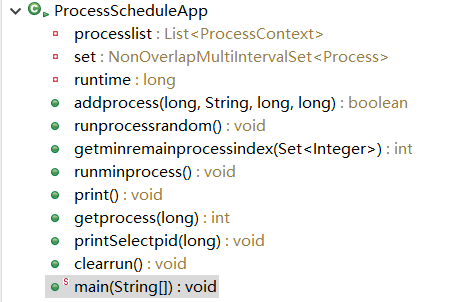




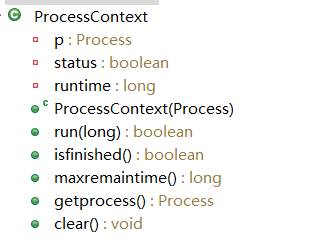


### 操作系统的进程调度管理系统

操作系统的具体实现如下图所示：



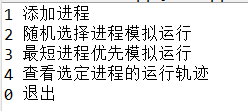
为了更方便的实现该应用我又定义了一个ProcessContext的ADT用来保存进程的上下文信息，它主要记录着进程运行的时间和进程的状态，它在操作系统选择下一时刻执行的进程中起着重要的作用，具体实现如下图所示：

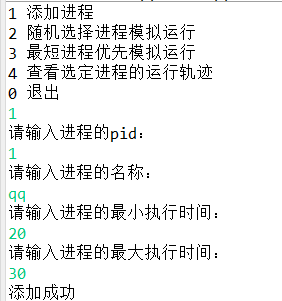


根据要求我们知道进程是可以多次执行的，所以我选用CommonMultiIntervalSet作为最基础的实现类，又由于操作系统可以挂机，在一段时间内不进行进程，所以我们并不需要用NoBlankIntervalSet来进行装饰，但因为模拟的是单核CPU不能在同一时间运行多个进程，所以又需要用NonOverlapIntervalSet来装饰CommonIntervalSet。具体装饰如下图所示：

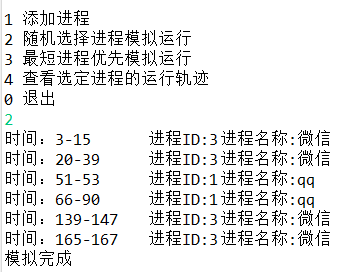


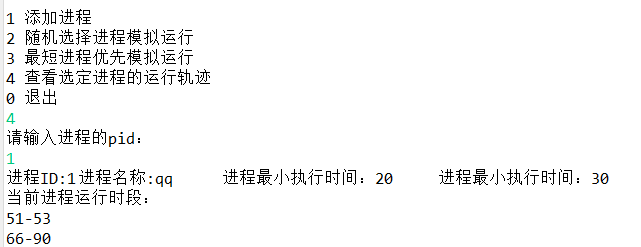
运行时的图片：





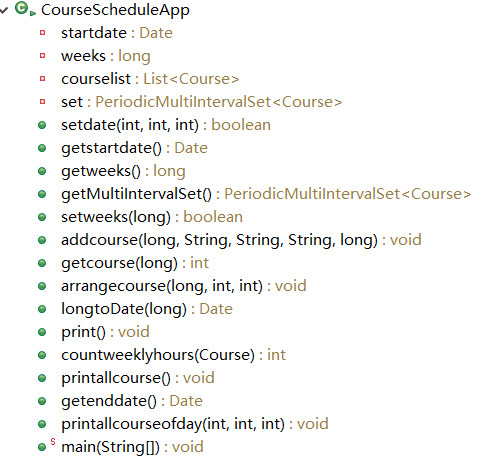
（在这里我还添加了个微信）



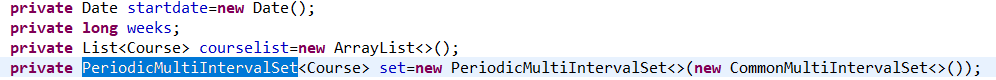


### 课表管理系统

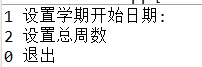
课表管理系统的具体实现如下图所示：



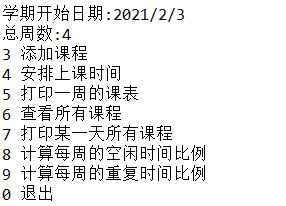
由于课表管理系统里的课程是可以多次重复出现的，所以我选用CommonMultiIntervalSet作为基础类，又由于课程是周期性重复出现的，所以我使用PeriodicMultiIntervalSet来装饰CommonMultiIntervalSet。

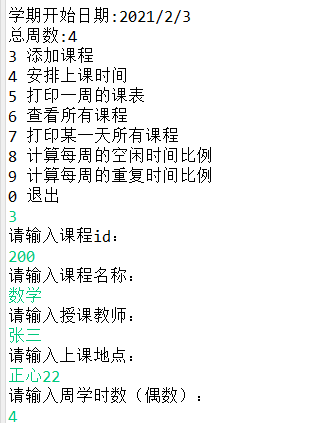


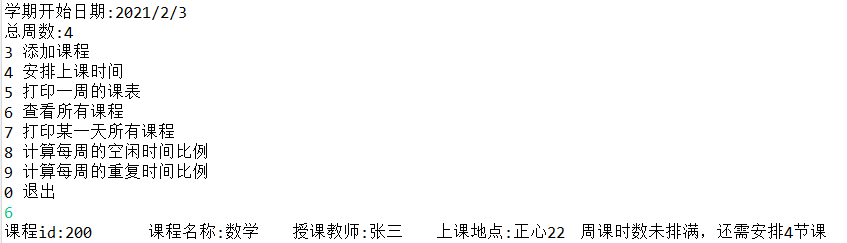
运行时的图片：

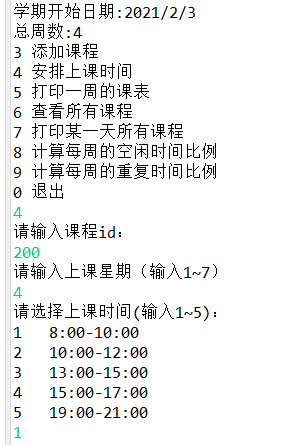


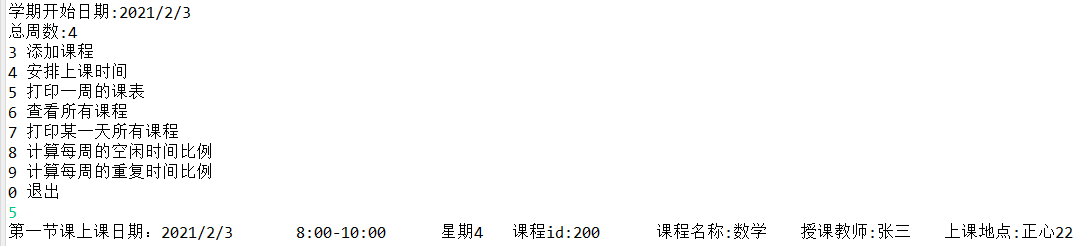
（我随便设置了学期开始日期和总周数）

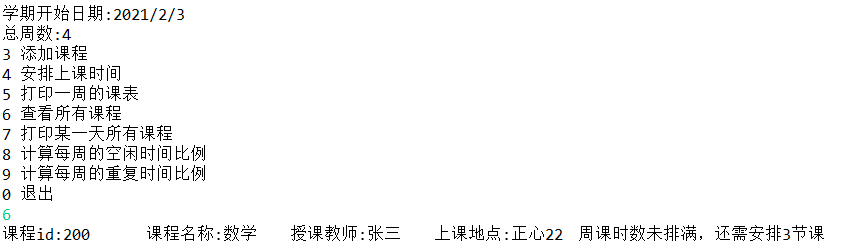












## 基于语法的数据读入

先使用如下的正则表达式从文件中得到Employee,Roster,Period的对应内容



之后再对获得的块进行分别读取，读取的关键就在于要对正则表达式进行分组，这样才可以用group（）函数得到我们想要的内容，在正则表达式中用（）括住想要的内容就行，（）出现的顺序就代表子字符串分组的顺序。

1 读取Period



2 读取Employee



3 读取Roster



## 应对面临的新变化

### 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

### 变化2

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

## 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change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 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
|  |  |
|  |  |
|  |  |

# 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

## 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在三个不同的应用场景下使用，你是否体会到复用的好处？
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？
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？
4. 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？
5. 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？
6. “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？
7. 关于本实验的工作量、难度、deadline。
8. 到目前为止你对《软件构造》课程的评价。