

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 沈城有 |
| 学号 | 1190200526 |
| 班号 |  |
| 电子邮件 |  |
| 手机号码 |  |

**目录**

[1 实验目标概述 1](#_Toc76213464)

[2 实验环境配置 1](#_Toc76213465)

[3 实验过程 1](#_Toc76213466)

[3.1 待开发的三个应用场景 1](#_Toc76213467)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 2](#_Toc76213468)

[3.2.1 IntervalSet<L>的共性操作 2](#_Toc76213469)

[3.2.2 局部共性特征的设计方案 4](#_Toc76213470)

[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） 5](#_Toc76213471)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 7](#_Toc76213472)

[3.3.1 MultiIntervalSet<L>的共性操作 7](#_Toc76213473)

[3.3.2 局部共性特征的设计方案 10](#_Toc76213474)

[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） 10](#_Toc76213475)

[3.4 面向复用的设计：L 13](#_Toc76213476)

[3.5 可复用API设计 14](#_Toc76213477)

[3.5.1 计算相似度 14](#_Toc76213478)

[3.5.2 计算时间冲突比例 15](#_Toc76213479)

[3.5.3 计算空闲时间比例 15](#_Toc76213480)

[3.6 应用设计与开发 16](#_Toc76213481)

[3.6.1 排班管理系统 16](#_Toc76213482)

[3.6.2 操作系统的进程调度管理系统 19](#_Toc76213483)

[3.6.3 课表管理系统 20](#_Toc76213484)

[3.7 基于语法的数据读入 22](#_Toc76213485)

[3.8 应对面临的新变化 24](#_Toc76213486)

[3.8.1 变化1 24](#_Toc76213487)

[3.8.2 变化2 26](#_Toc76213488)

[3.9 Git仓库结构 28](#_Toc76213489)

[4 实验进度记录 29](#_Toc76213490)

[5 实验过程中遇到的困难与解决途径 30](#_Toc76213491)

[6 实验过程中收获的经验、教训、感想 30](#_Toc76213492)

[6.1 实验过程中收获的经验和教训 30](#_Toc76213493)

[6.2 针对以下方面的感受 31](#_Toc76213494)

# 实验目标概述

本次实验覆盖课程第2、3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
* 语法驱动的编程、正则表达式
* 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

实验环境与前两次实验相同，无需再次配置。

GitHub Lab3仓库的URL地址：

<https://github.com/ComputerScienceHIT/HIT-Lab3-1190200526>

# 实验过程

## 待开发的三个应用场景

1. 值班表管理(DutyRoster)：为n名员工安排某个时间段内的值班，要求每天均安排一人值班，且每个人的值班时间若为m天(m > 1)，则这m天一定连续。
2. 操作系统进程调度管理(ProcessSchedule)：单核CPU计算机，每个时间仅能随机调度一个进程运行，其他进程处于休眠状态，且特定时刻可不调度执行任何进程。
3. 大学课表管理(CourseSchedule)：简化版课表，可以存在空白时间段，允许同一时间段内安排不同的课程，允许一门课程一周可安排一次或多次。假设由同一位教师承担并在同样的教室进行，一位教师也可以承担课表中的多门课程。

**相同点：**三个应用场景都存在时间段及时间段集合的概念，都需要实现标签和时间段之间的对应。

**不同点：**

* 第一个场景中标签与时间段一一对应，而第二个、第三个场景较第一个场景略复杂，一个标签可能对应多个不连续的时间段；
* 前两个场景不会出现时间段重叠的情况，而第三个场景会出现时间段重叠；
* 第三个场景中时间段还会呈现为周期性分布。

## 面向可复用性和可维护性的设计：IntervalSet<L>

该节是本实验的核心部分。

IntervalSet<L>是一个 mutable 的 ADT，描述了一组在时间轴上分布的“时间段”（interval），每个时间段附着一个特定的标签，且标签不重复。标签可以是任意immutable的ADT。

### IntervalSet<L>的共性操作

共性操作定义在src/baseADTs目录下的接口文件IntervalSet.java中，此处简要介绍各方法。

* **public** **static** <L> IntervalSet<L> empty()

此静态方法创建一个空的IntervalSet（时间段集合）。规约如下：

/\*\*

\* 创建一个空的时间段集合

\*

\* **@param** <L> 时间段附着的标签的类型，可以是任何immutable的ADT

\* **@return** 一个新的空时间段集合

\*/

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

此方法在当前对象中插入新的时间段和标签，并附带检测。规约如下：

/\*\*

\* 在集合中插入新的带标签时间段

\*

\* **@param** start 开始时间

\* **@param** end 结束时间

\* **@param** label 标签

\* **@throws** Exception 标签重复或添加非法时间段时抛出异常

\* （非法指空标签或开始时间大于结束时间或时间存在负数）

\*/

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

此方法获得当前对象中的标签集合。规约如下：

/\*\*

\* 获得当前时间段集合的标签集合

\*

\* **@return** 当前对象的标签集合

\*/

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

此方法从当前对象中移除某个标签所关联的时间段。规约如下：

/\*\*

\* 从当前时间段集合中移除某个标签所关联的时间段

\*

\* 如果含有此标签关联的时间段，则进行移除，

\* 否则不进行任何修改。

\*

\* **@param** label 标签

\* **@return** 是否执行了移除

\*/

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

此方法返回某个标签对应的时间段的开始时间。规约如下：

/\*\*

\* 返回某个标签对应的时间段的开始时间

\*

\* **@param** label 标签

\* **@return** 此标签对应时间段的开始时间

\* 假如此标签未对应时间段返回-1

\*/

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

此方法返回某个标签对应的时间段的结束时间。规约如下：

/\*\*

\* 返回某个标签对应的时间段的结束时间

\*

\* **@param** label 标签

\* **@return** 此标签对应时间段的结束时间

\* 假如此标签未对应时间段返回-1

\*/

* **public** **boolean** isEmpty()

此方法是我在实验指导书基础上自行添加的，用于检测对象是否为空。规约如下：

/\*\*

\* 判断此时间段集合是否为空

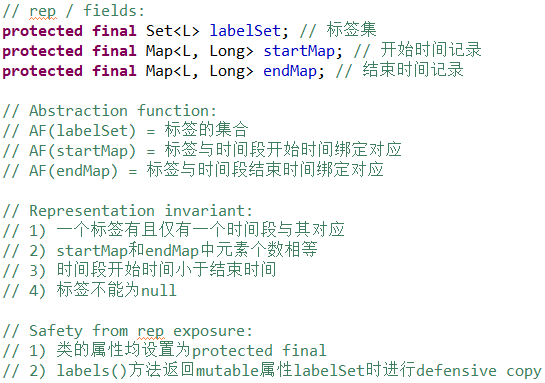
\*

\* **@return** 时间段集合是否为空的判断结果

\*/

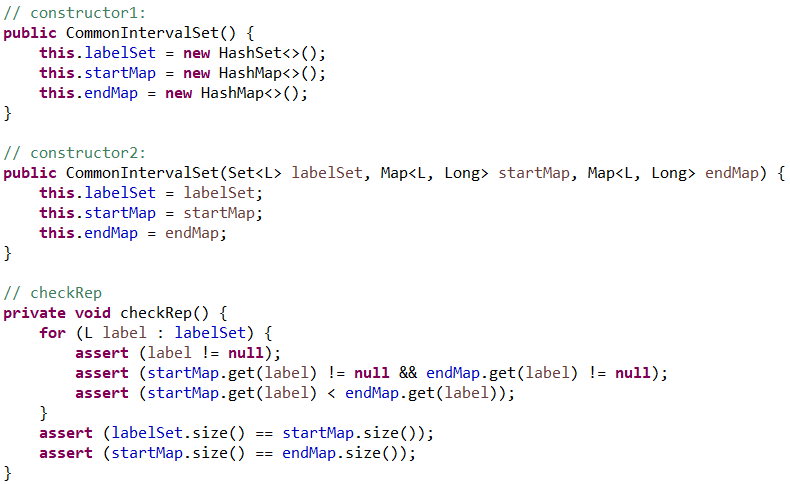
### 局部共性特征的设计方案

src/baseADTs/CommonIntervalSet.java中的CommonIntervalSet<L>类继承了上述的IntervalSet<L>接口，并进行了具体实现。其rep、AF、RI、Safety from rep exposure内容如下：

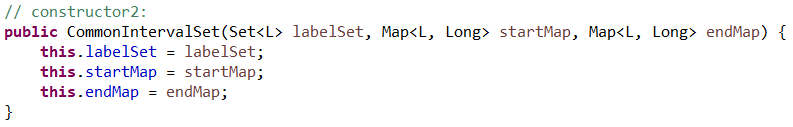


除实现共性操作方法（较简单，此处略去）之外，还应实现构造函数、checkRep()并重写toString()方法，如下图：

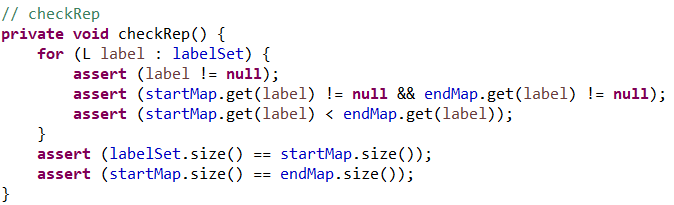
用于初始化空对象的构造函数：



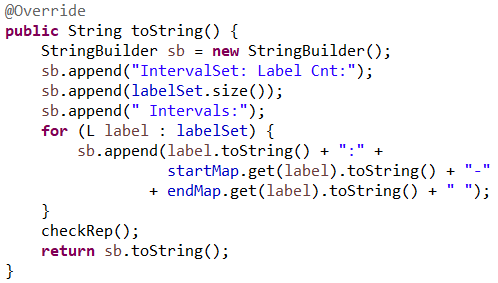
用于委托操作对象的构造函数：



检查不变性的私有方法checkRep()：

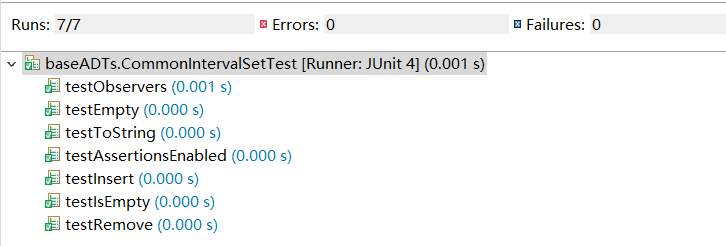


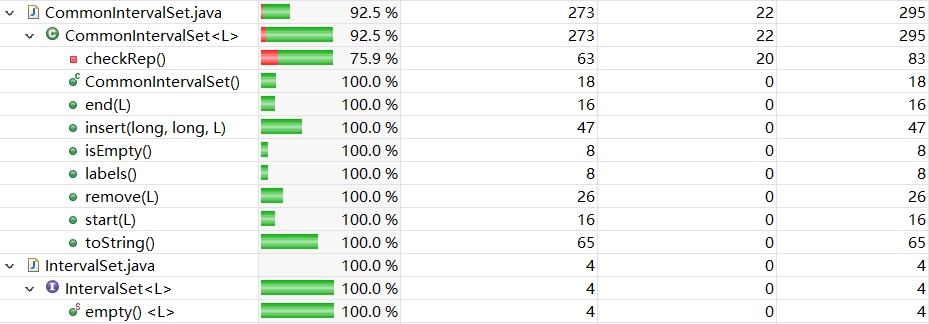
将对象转换为可读表示的toString()方法：



测试代码见test/baseADTs/CommonIntervalSetTest.java，此处略去。

测试结果如下：





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

我选择使用实验指导书中的方案五（CRP，通过接口组合实现局部共性特征的复用）来进行设计，首先需完成了三个维度的接口、具体实现，但此处仅能实现第一个维度——“无重叠”维度。

此维度操作的相关接口和具体实现分别在src/specialOps目录下的文件NonOverlapIntervalSet.java、NonOverlapIntervalSetImpl.java中。

我的思路是实现一个先检查是否会重叠再进行插入操作的insert()方法，故接口中仅定义了这一方法。具体规约如下：

（见下页）

/\*\*

\* 时间段无重叠的插入方法

\*

\* **@param** start 开始时间

\* **@param** end 结束时间

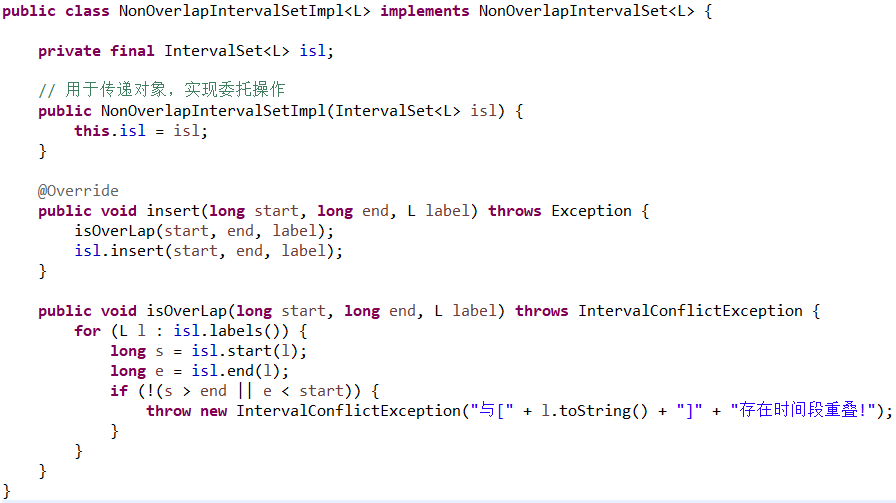
\* **@param** label 标签

\* **@throws** Exception 非法时间段、标签重复、时间段重叠抛出异常

\*/

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

实现思路是在实现类中执行对IntervalSet<L>的重叠检测和插入，具体实现如下：



在实现这个维度的基础上，现在可以具体实现排班表应用的IntervalSet子类型（具体代码可查看src/appADTs目录下的源文件IDutyIntervalSet.java、DutyIntervalSet.java），以下是对这此应用对应IntervalSet子类型设计、实现的简要说明：

rep、AF、RI、Safety from rep exposure内容如下：



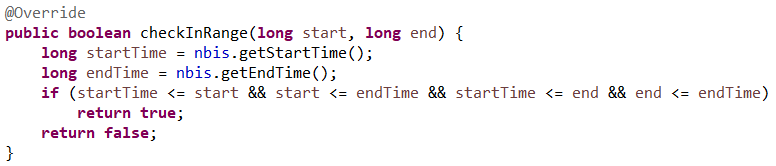
接口和具体实现类的继承、实现关系如下：

**public** **class** DutyIntervalSet<L> **extends** CommonIntervalSet<L> **implements** IDutyIntervalSet<L>

**public** **interface** IDutyIntervalSet<L> **extends** NonOverlapIntervalSet<L>,

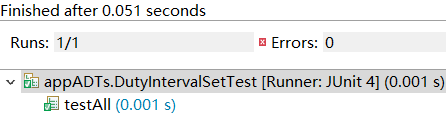
NoBlankIntervalSet<L>

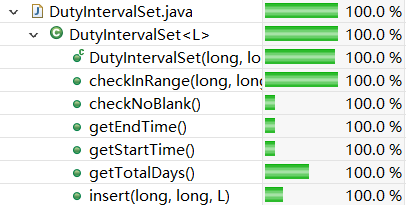
具体实现中方法较少，主要是进行委托绑定（传递对象引用）、获取一些信息用于应用设计时处理、输出，此处略去。后来在实际开发APP时添加了checkInRange()方法检查输入合法性，具体实现如下：



测试代码见test/appADTs/DutyIntervalSetTest.java，此处略去。

测试结果如下：

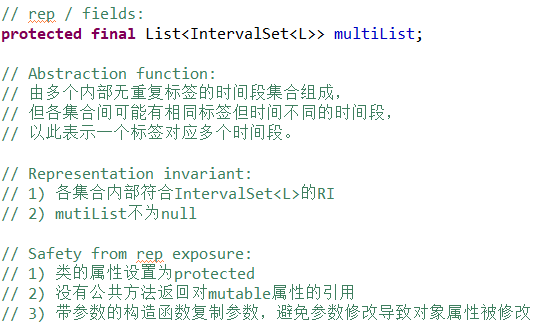


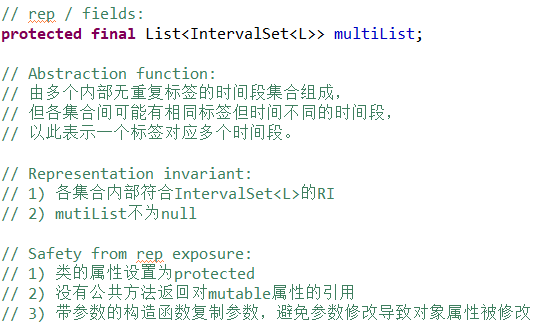


## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

我选择将MultiIntervalSet<L>实现为具体类，rep、AF、RI、Safety from rep exposure内容如下：





共性方法有以下几个：

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

此方法在当前对象中插入新的时间段和标签，具体规约如下：

/\*\*

\* 在当前对象中插入新的时间段和标签

\*

\* **@param** start 开始时间

\* **@param** end 结束时间

\* **@param** label 标签

\* **@throws** Exception 非法时间段抛出异常

\*/

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

此方法获取当前对象的标签集合。

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

规约如下：

/\*\*

\* 从当前对象中移除某个标签所关联的所有时间段

\*

\* 如果含有此标签对应的时间段，则进行移除，

\* 否则不进行任何修改。

\*

\* **@param** label 标签

\* **@return** 是否执行了移除

\*/

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

规约如下：

/\*\*

\* 从当前对象中获取与某个标签所关联的所有时间段

\* 其中的时间段按开始时间从小到大的次序排列

\*

\* **@param** label 标签

\* **@return** 按开始时间从小到大的次序排列的时间段集合

\* 如果没有此标签对应的时间段，返回空集

\*/

* **public** **boolean** isEmpty()

规约如下：

/\*\*

\* 判断对象是否为空

\*

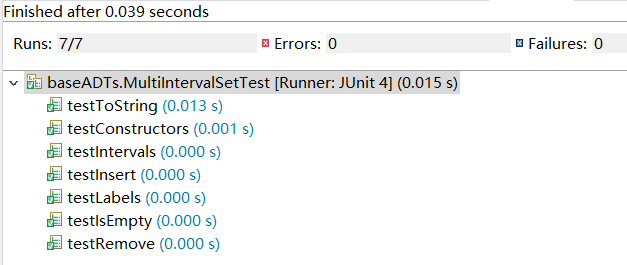
\* **@return** 对象是否为空的判断结果

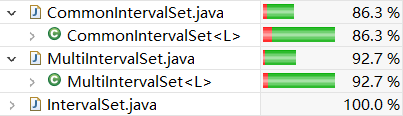
\*/

这些方法中比较难实现的是intervals()方法，通过此方法可以体现MultiIntervalSet<L>的大部分结构和内容。核心部分代码如下：



测试代码此处略，测试结果如下：





可以看到，IntervalSet的接口、具体实现类代码覆盖度也比较高，这是因为MultiIntervalSet进行了复用。

### 局部共性特征的设计方案

由于MultiIntervalSet<L>已实现为具体类，局部共性特征的设计和实现也体现在3.3.1节中，此处不再重复。

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

至此，后两个维度也可以进一步实现。以下是关于这两个维度设计实现的简要说明：

1. **“非空”维度：**

此维度操作的相关接口和具体实现分别在src/specialOps目录下的文件NoBlankIntervalSet.java、NoBlankIntervalSetImpl.java中。

我的思路是实现一个检查是否存在空白并输出相关信息的checkNoBlank()方法，此方法在接口中定义如下：

/\*\*

\* 检查是否存在空白时间段

\* 输出相关信息

\*/

**public** **void** checkNoBlank();

实现时，我在实现类中使用私有属性保存对象起止时间和传入的对象引用，检查时使用这一起止时间，统计空白并输出信息。私有属性定义如下：

**private** **final** **long** startTime;

**private** **final** **long** endTime;

**private** **final** CommonIntervalSet<L> cisl;

具体代码此处略去。

1. **“周期性”维度：**

此维度操作的相关接口和具体实现分别在src/specialOps目录下的文件PeriodicIntervalSet.java、PeriodicIntervalSetImpl.java中。

我的思路是实现一个进行周期性插入的insert()方法，此方法在接口中定义如下：

/\*\*

\* 插入周期性重复时间段的插入方法

\*

\* **@param** start 开始时间

\* **@param** end 结束时间

\* **@param** label 标签

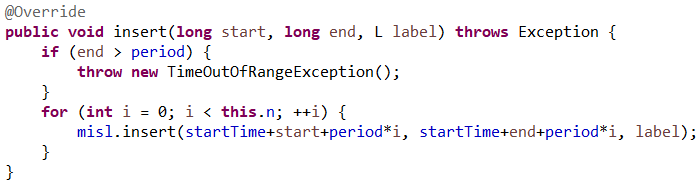
\* **@throws** Exception 非法时间段、超出时间范围抛出异常

\*/

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

实现时，我在实现类中使用私有属性保存开始时间、周期长度、周期数和传入的对象引用，插入时使用这些信息进行周期性插入，核心代码如下：

（见下页）

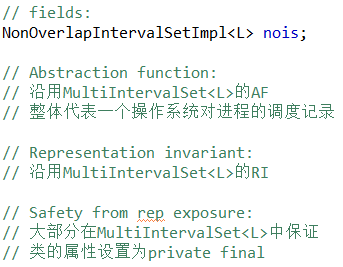


除此之外，要保证进行委托的对象能获取到周期相关信息，还实现了获取开始时间、周期数的公共方法，此处略。

在完成三个维度全部内容的设计实现的基础上，我随后完成了对后两个应用MultiIntervalSet子类型的设计和实现：

1. ProcessIntervalSet<L>：

rep、AF、RI、Safety from rep exposure内容如下：

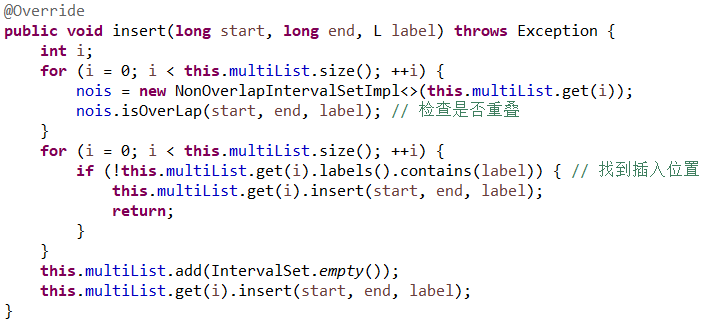


接口和具体实现类的继承、实现关系如下：

**public** **interface** IProcessIntervalSet<L> **extends** NonOverlapIntervalSet<L>

**public** **class** ProcessIntervalSet<L> **extends** MultiIntervalSet<L> **implements** IProcessIntervalSet<L>

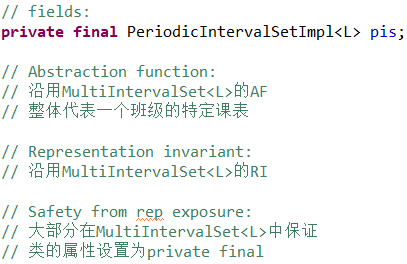
具体实现中仅有一个insert()方法，此方法重写MultiIntervalSet<L>中的insert()方法，实现了“无重叠”，代码如下：



1. CourseIntervalSet<L>：

rep、AF、RI、Safety from rep exposure内容如下：

（见下页）

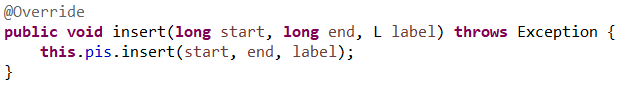


接口和具体实现类的继承、实现关系如下：

**public** **interface** ICourseIntervalSet<L> **extends** PeriodicIntervalSet<L>

**public** **class** CourseIntervalSet<L> **extends** MultiIntervalSet<L> **implements** ICourseIntervalSet<L>

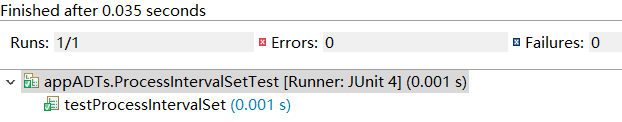
具体实现包括insert()和一些获取信息的方法，insert()方法重写MultiIntervalSet<L>中的insert()方法，通过委托实现了“周期性”，代码如下：

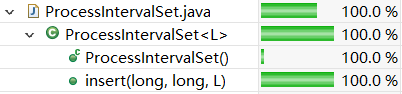


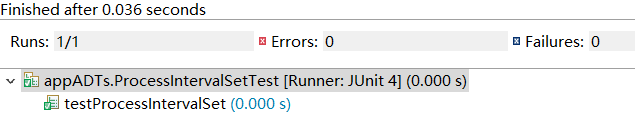
至此，三个应用所需的ADT均已实现。

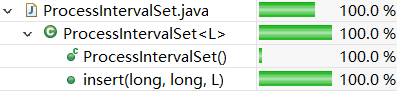
后实现的两个应用ADT的测试代码在test/appADTs目录下，此处略。

测试结果如下：







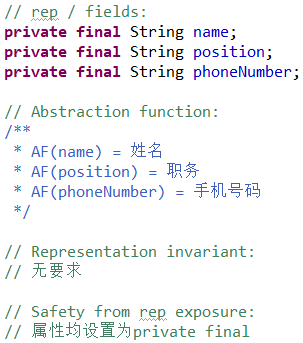


## 面向复用的设计：L

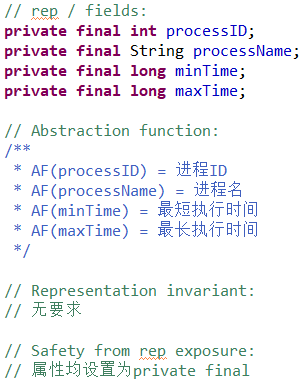
对三个应用来说，其 L 分别应为“员工”（Employee）、“进程”（Process）、

“课程”（Course），所需关注的属性及对应设计分别为：

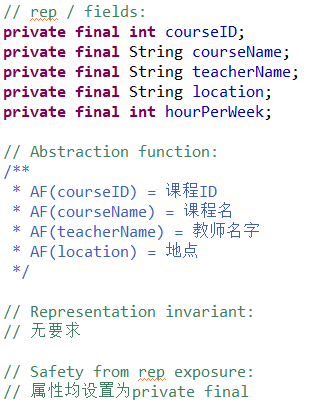
* Employee：姓名、职务、手机号码



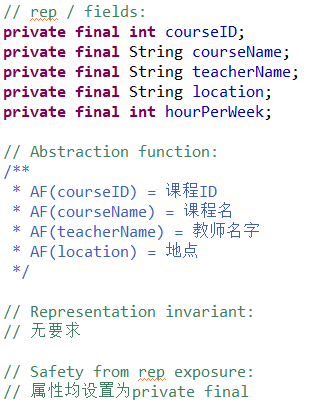
* Process：进程 ID、进程名称、最短执行时间、最长执行时间



* Course：课程 ID、课程名称、教师名字、地点、周学时（自行添加）



（接下页）



除此之外，各类中还有获取每个私有属性的公共方法，并重写了toString()、hashCode()、equals()方法，便于开发APP时使用。

具体实现可见src/labels目录下三个源文件中的代码。

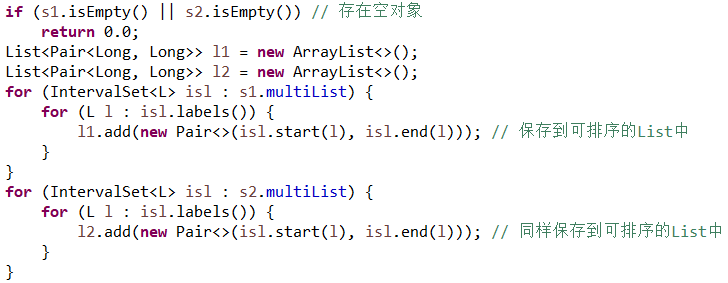
## 可复用API设计

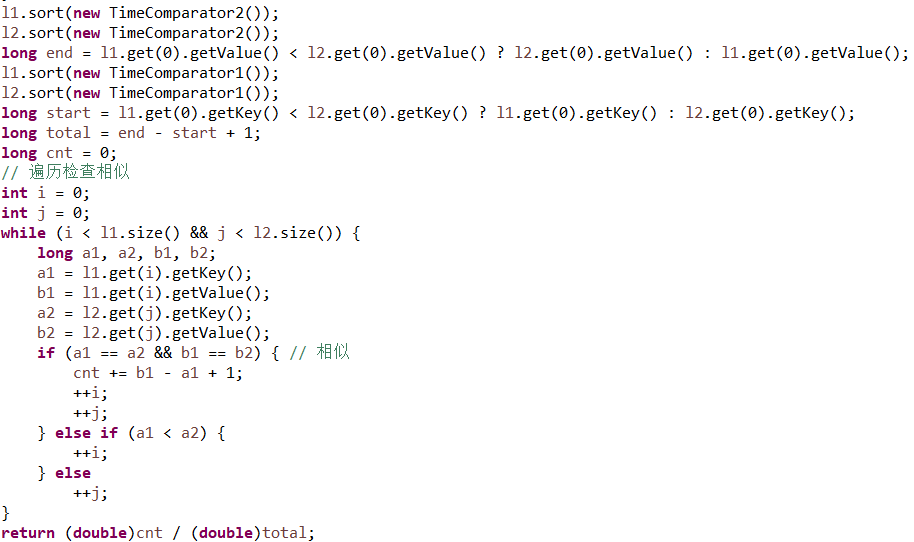
API全部代码可查看src/baseADTs目录下APIs.java文件。API各方法实现思路类似，MultiIntervalSet<L>和IntervalSet<L>的处理也类似，故在本节中仅进行简单展示。

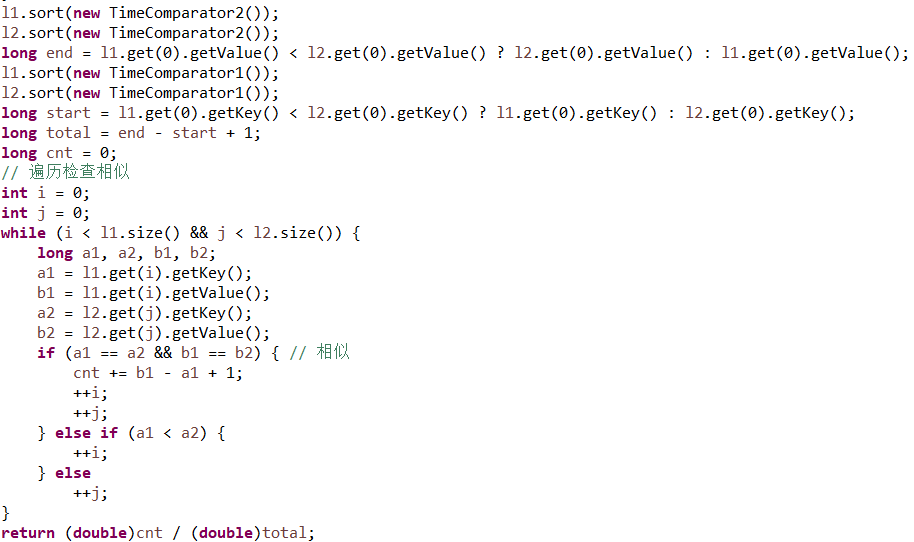
### 计算相似度

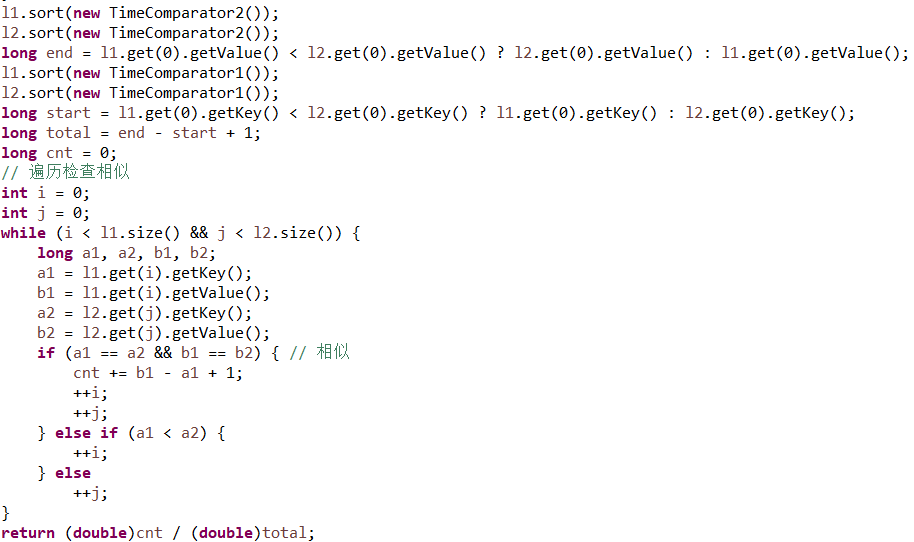
按照时间轴从早到晚的次序，针对同一个时间段内两个对象里的interval，若它们标注的label等价，则二者相似度为1，否则为0；若同一时间段内只有一个对象有interval或二者都没有，则相似度为0。将各interval的相似度与interval的长度相乘后求和，除以总长度，即得到二者的整体相似度。

我的实现思路如下：首先将两个对象里的时间段进行排序，获得开始、结束时间及总长度（结束时间 - 开始时间），随后遍历比较两个对象中已排序的时间段，判断是否相似并统计相似长度，最后计算相似度。主体代码如下：







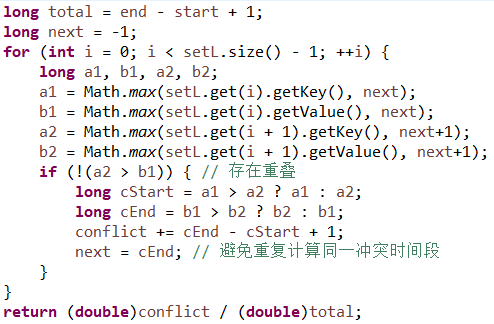


### 计算时间冲突比例

这里的“冲突”是指同一个时间段内安排了两个不同的interval对象。用

发生冲突的时间段总长度除于总长度，得到冲突比例，是一个[0,1]之间的值。

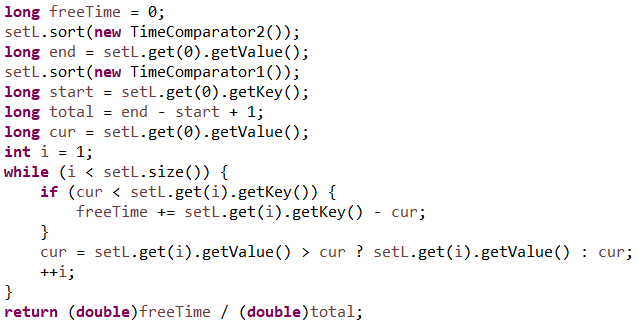
思路与3.5.1类似，首先同样使用List进行排序并获得开始、结束时间及总长度（结束时间 - 开始时间），但遍历的过程有所不同：为避免重复计算多次重叠的同一时间段，导致结果数值偏大甚至超过1，遍历中需要变量记录当前遍历时间段结束位置，并不断更新，避免重复发生。主体代码如下：



### 计算空闲时间比例

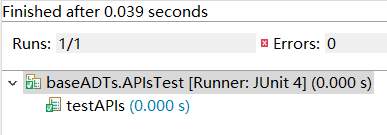
这里的“空闲”是指某时间段内没有安排任何 interval 对象。用空闲的时间段总长度除于总长度，得到空闲比例，是一个[0,1]之间的值。

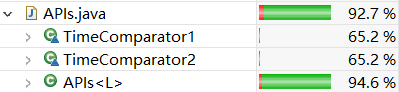
思路依然类似，略有不同的还是遍历，这次需要记录当前检查到的位置，并与排序好的下一个时间段开始位置进行比较，若下一个开始位置在当前位置之后，则将二者距离加入到空闲时间统计中。无论是否有空闲，每次遍历结束时都要更新当前位置为此时间段结束位置和当前位置的较大值。主体代码如下：



测试代码此处略，可查看test/baseADTs/APIsTest.java文件。

API部分整体测试结果如下：





## 应用设计与开发

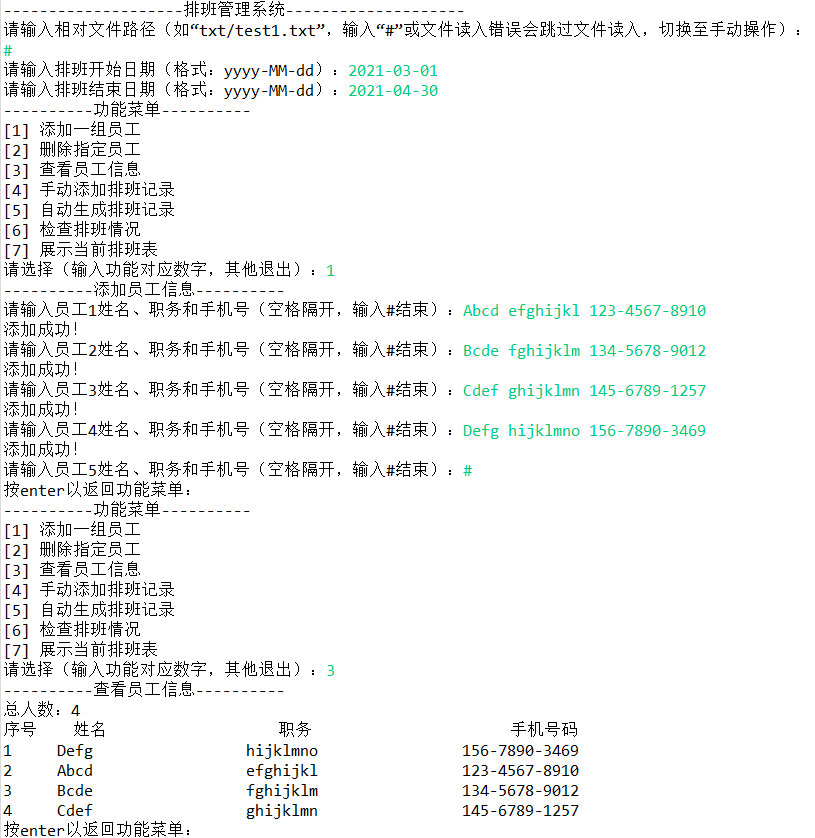
利用上述设计和实现的ADT，实现手册里要求的各项功能。主要的工作是利用这些已实现的方法完成实际操作。此部分源代码可见src/apps目录下的三个源代码文件。

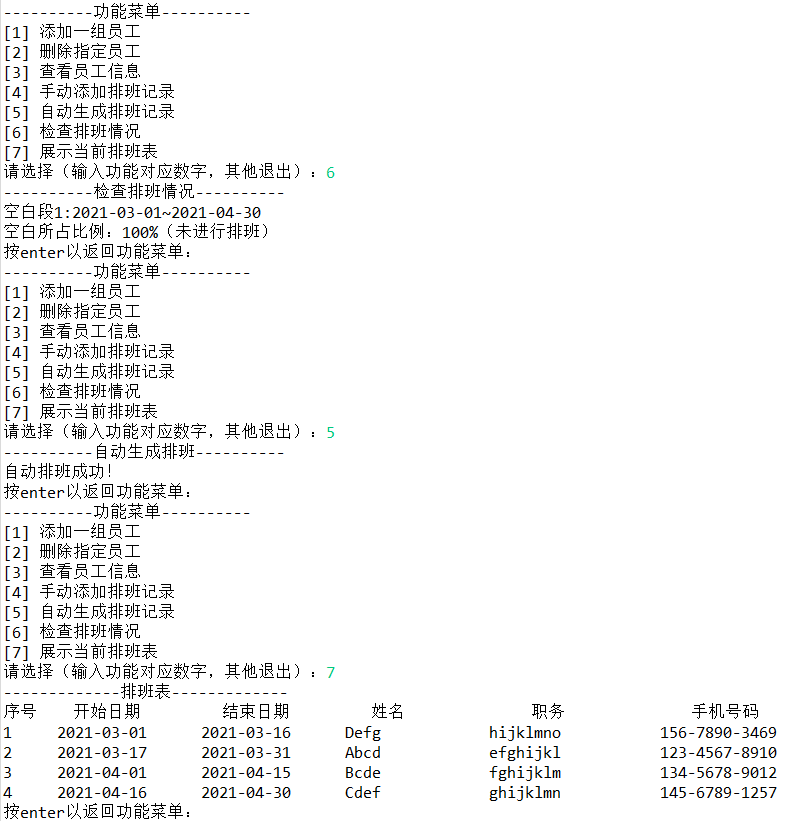
### 排班管理系统

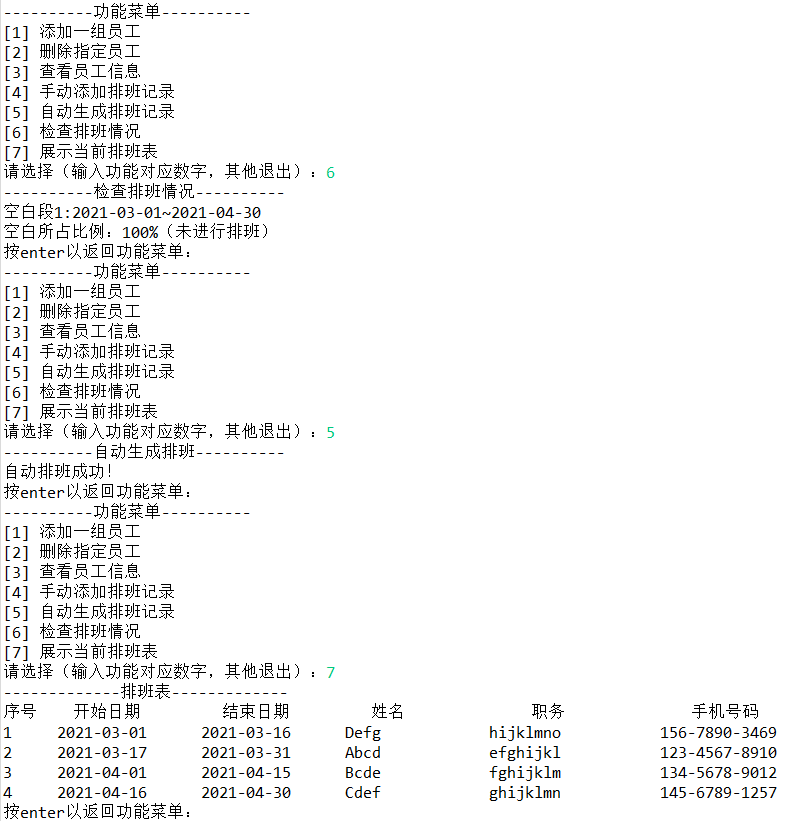
APP设计为命令行程序，完成了所要求功能的实现。具体代码此处略。这里总结几个要点：

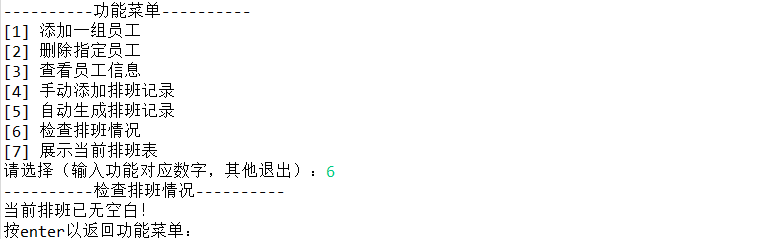
1. 使用一个HashSet暂存了员工信息，以便执行各种操作时使用；
2. 输入会涉及到合法性检测，需要注意（编写并使用了checkInRange()方法）；
3. 此应用涉及到大量的日期转换、处理等操作，故创建了帮助类dateHandler；
4. 应用捕获了之前自定义的异常类，并输出相关信息帮助用户了解问题所在并进行修改。

运行效果截图（部分功能展示）：











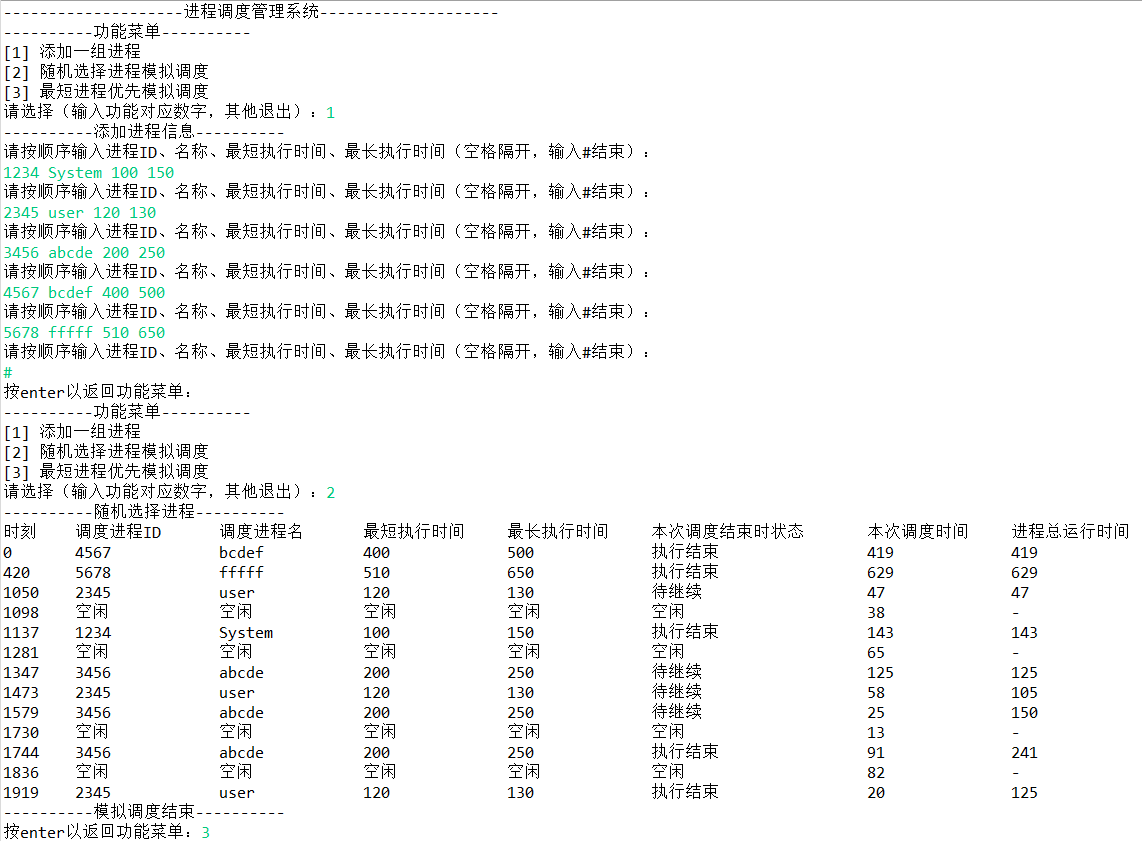


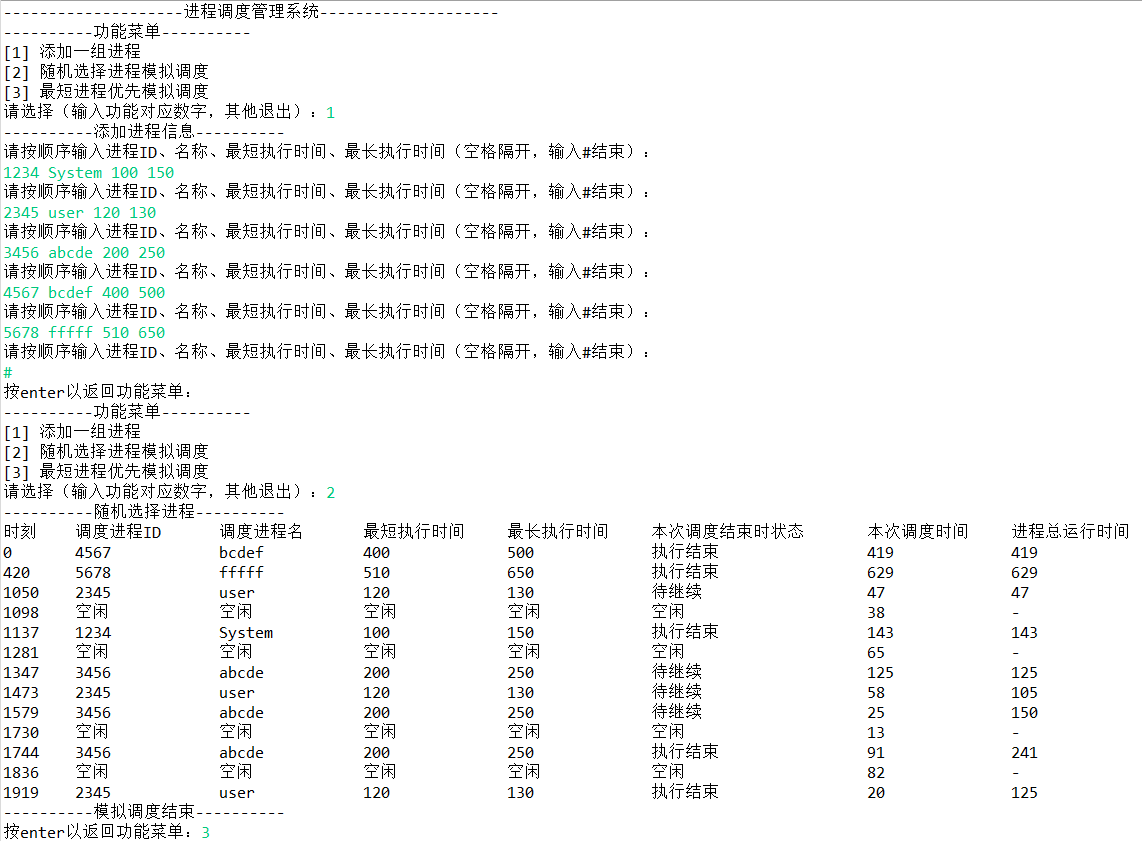
### 操作系统的进程调度管理系统

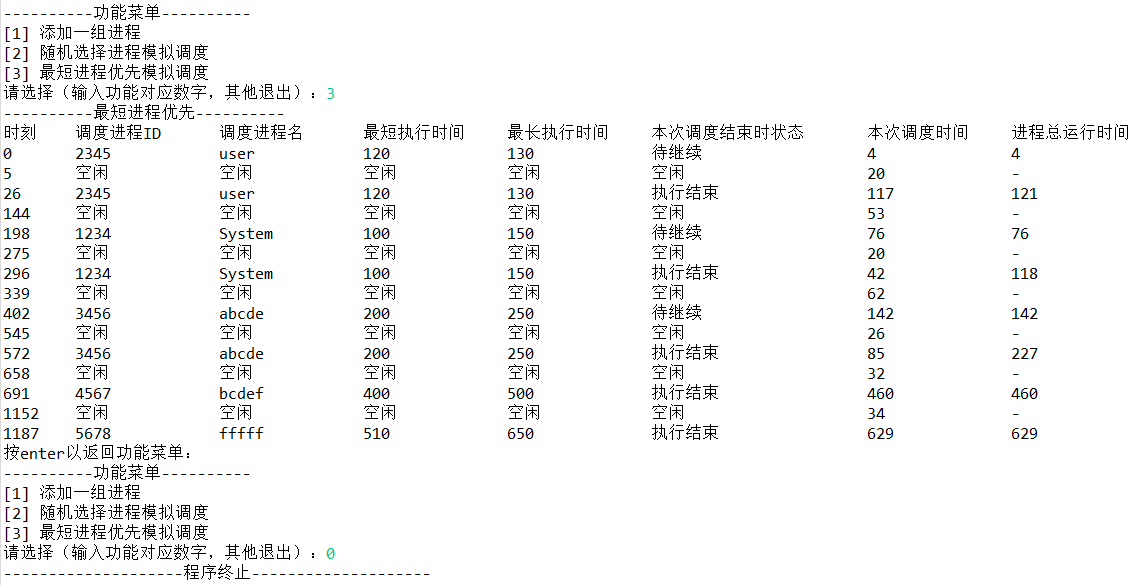
APP设计为命令行程序，完成了所要求功能的实现。具体代码此处略。这里总结几个要点：

1. 进程调度中的随机我选择使用Java的Math.random()实现，并开发了相关的帮助类randomGenerator（可查看src/helpers目录下的源代码文件randomGenerator.java）；
2. 两个策略均进行了实现，同时考虑到调度不应重复调度同一进程或空闲，设置了检测和处理；
3. 最短进程优先策略在执行前对进程按最大执行时间进行了排序，两种策略均使用List（或Map）实现动态修改进程可执行的剩余时间（或已运 行时间），以实现进程的调度时间限制与终止；
4. 程序中通过私有属性freeMax设置了进程调度过程中每次空闲的最大时间为100，可进行修改。

运行效果截图：





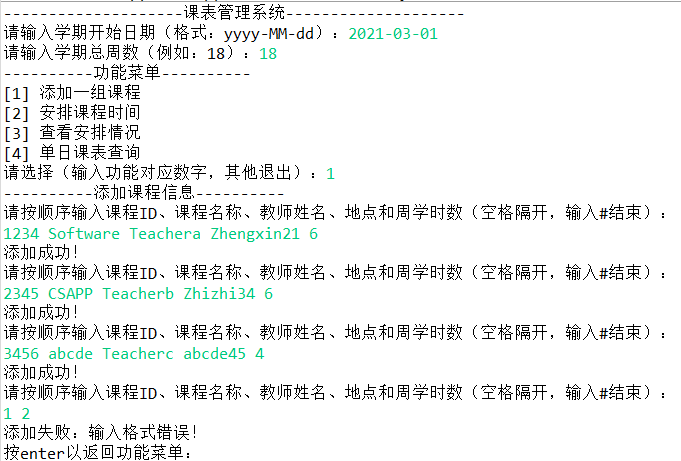


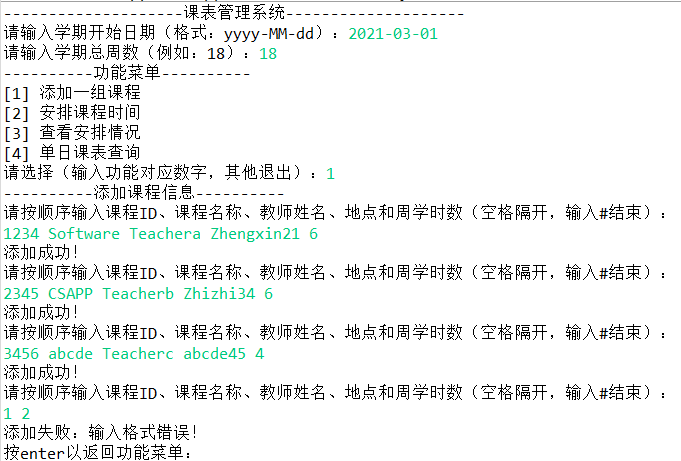
### 课表管理系统

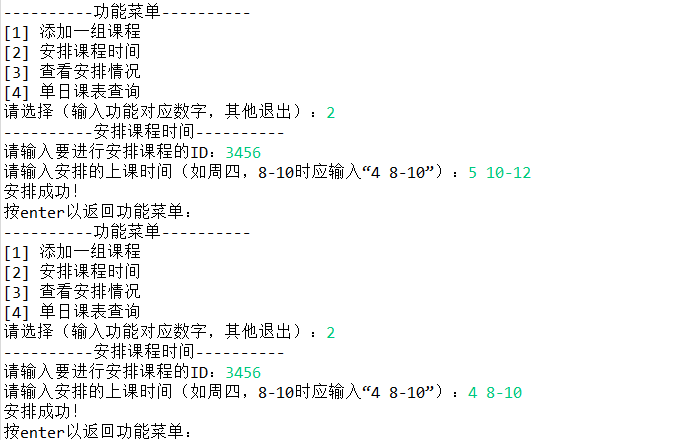
APP设计为命令行程序，完成了所要求功能的实现。具体代码此处略。这里总结几个要点：

1. 此应用再次涉及到日期相关处理，故完善并使用了dateHandler类，增加了计算日期对应一周第几天的方法，用于获取单日课程表；
2. 一周七天这一周期通过私有变量period确定；
3. 使用私有属性hourMap保存每门课程剩余的可安排周学时数，以便安排课程时进行检测及输出安排情况信息；
4. 添加课程、安排课程要考虑输入的较多情况，故代码较长；
5. 空闲时间、重复时间比例调用API中的方法进行计算。

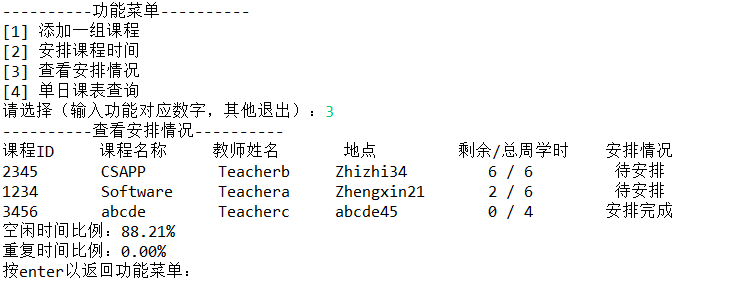
运行效果截图（部分）：

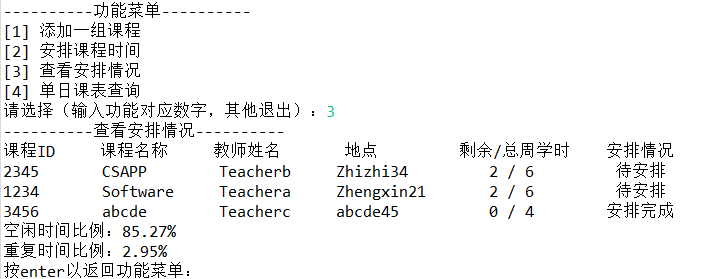


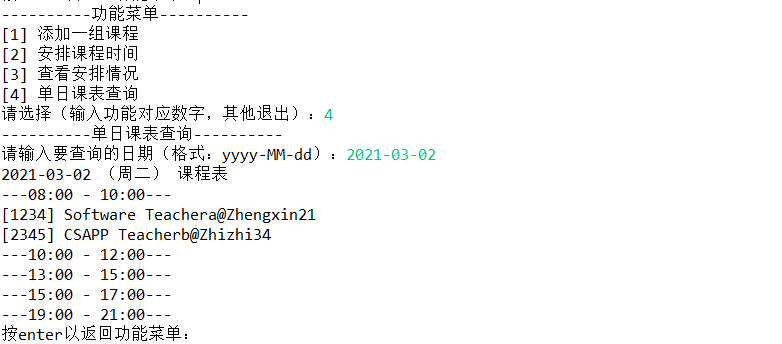




（省略了一部分安排过程）



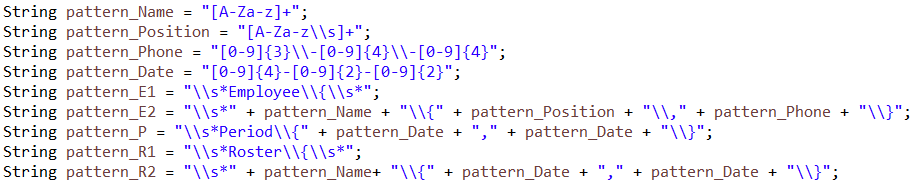




## 基于语法的数据读入

修改“排班管理”应用以扩展该功能。

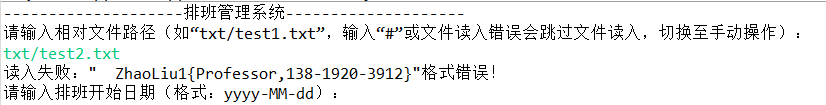
我根据实验指导书要求编写了以下正则表达式：



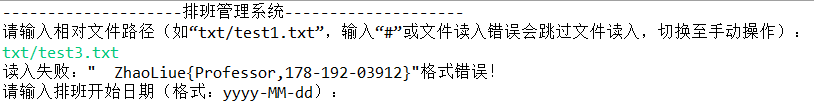
使用时，通过Pattern.matches()检查文件语法正确性并寻找各部分信息的起始位置，使用Pattern和Matcher的相关方法来进行局部截取获取信息，由于程序较长，此处截取一小部分展示：



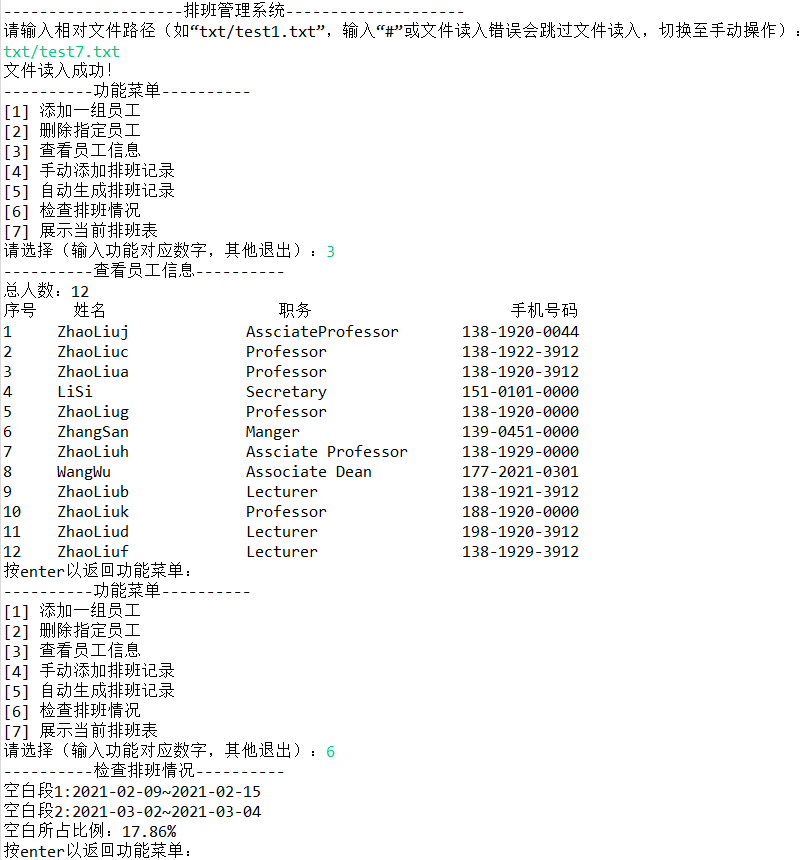
文件读取效果展示（部分测试用例）：



（姓名中不应含有数字）



（手机号格式不正确）







文件test7.txt中的数据为合法数据，且排班还存在空白，结果如上。

其他测试文件我也进行了测试，情况如下：

test1.txt、test4.txt合法，排班无空白；

test2.txt、test3.txt、test6.txt格式错误；

test5.txt存在时间段重叠；

test7.txt、test8.txt合法，排班分别有17.86%、14.29%空白。

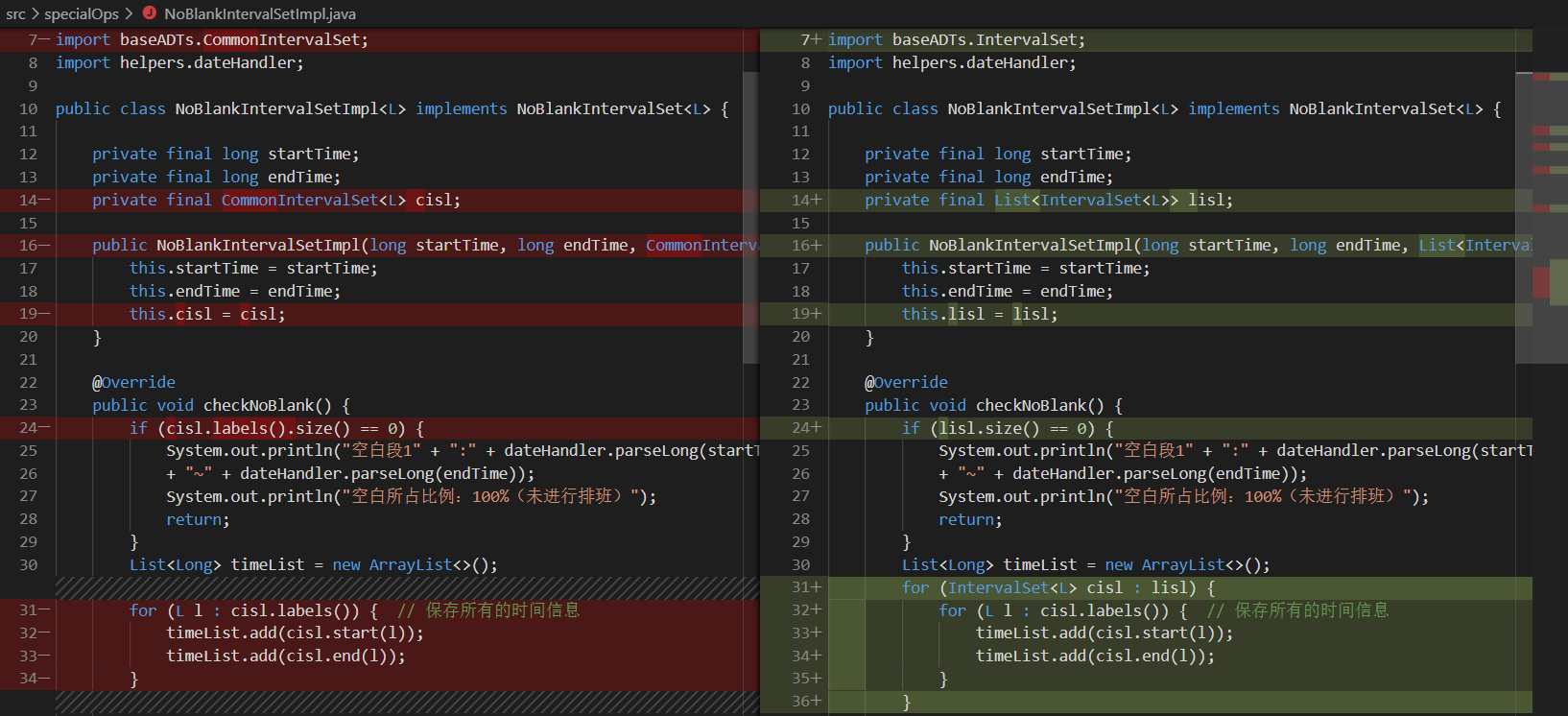
**注：对于员工信息重复或排班信息中对应员工不存在的情况，程序设计的处理策略是直接忽略这一项，故test7.txt、test8.txt能够正常读入。**

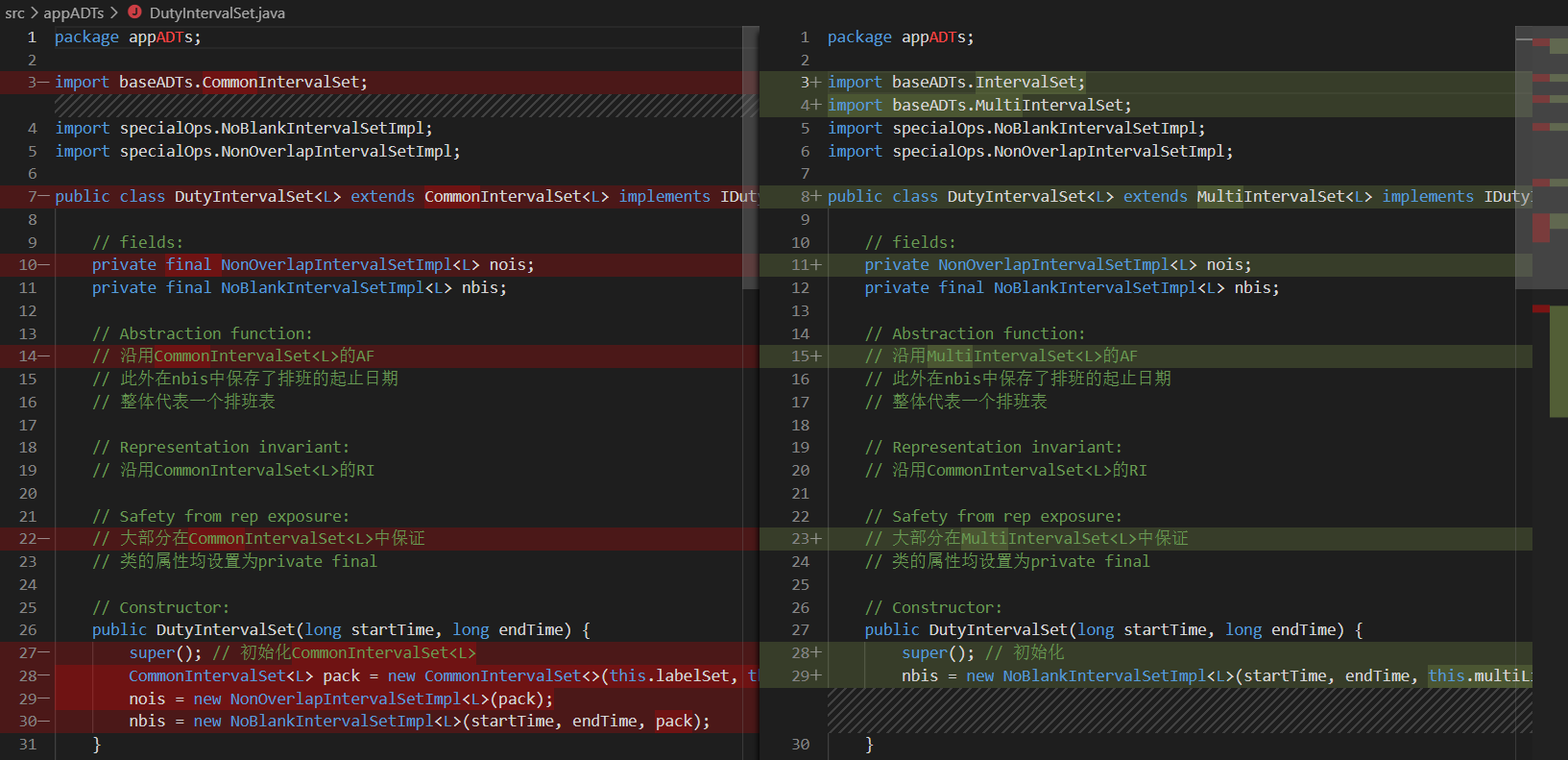
## 应对面临的新变化

### 变化1

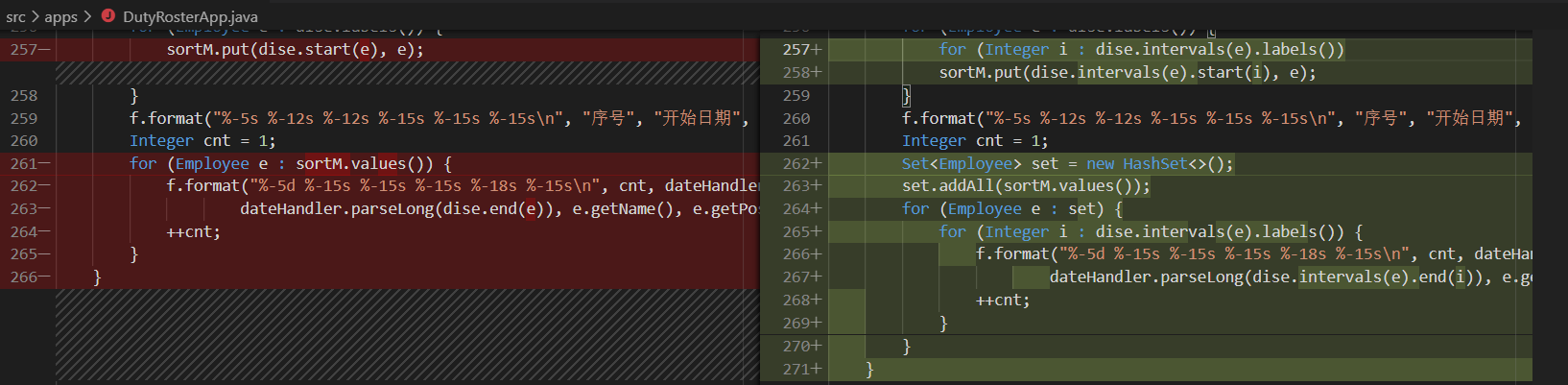
之前的设计可以应对变化，且修改难度不是很大。预计要修改三个文件，包括NoBlankIntervalSetImpl类的部分实现、DutyIntervalSet类部分方法实现、APP输出信息对应方法实现。

这些变化是围绕DutyIntervalSet<L>继承父类CommonIntervalSet<L>替换为MultiIntervalSet<L>而进行的，具体修改如下：

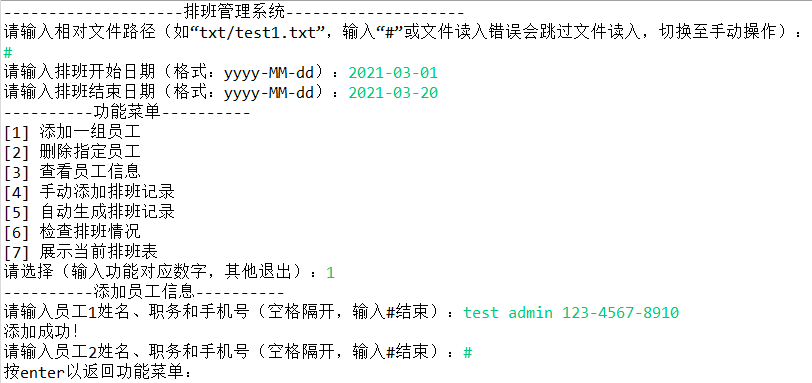


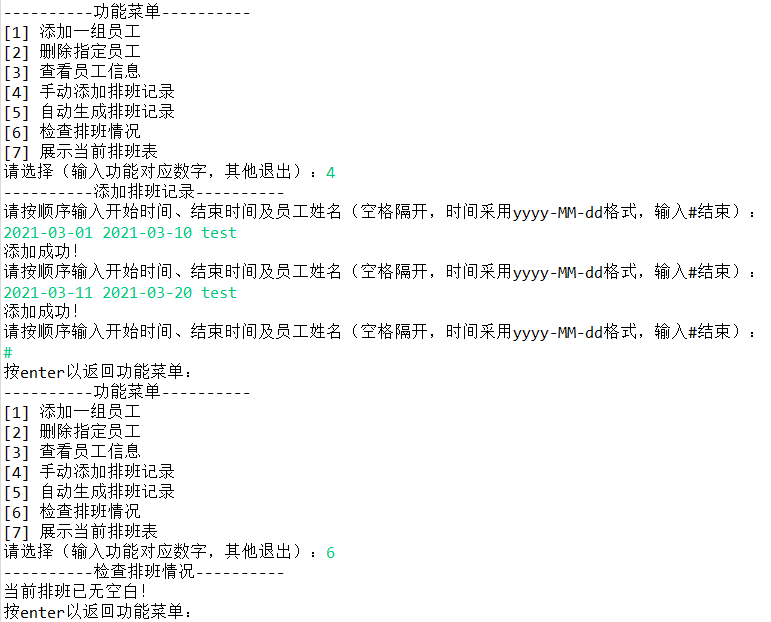


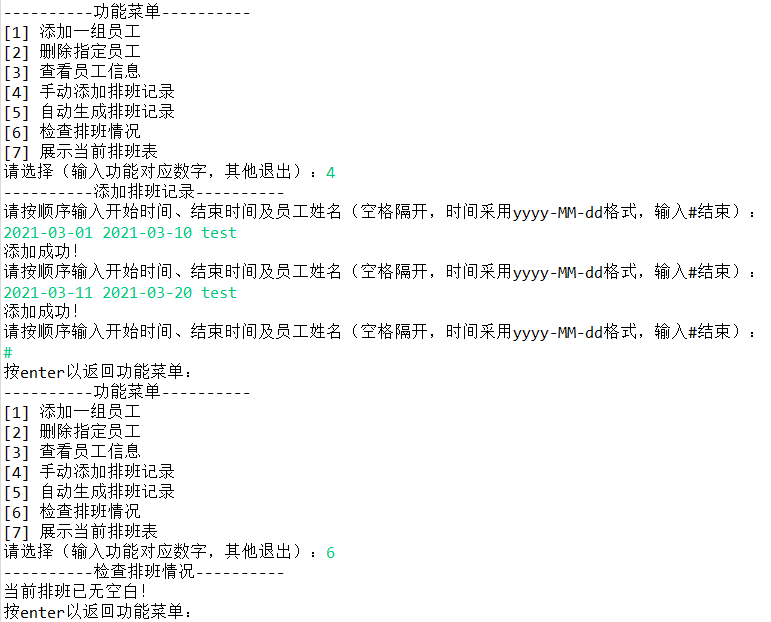


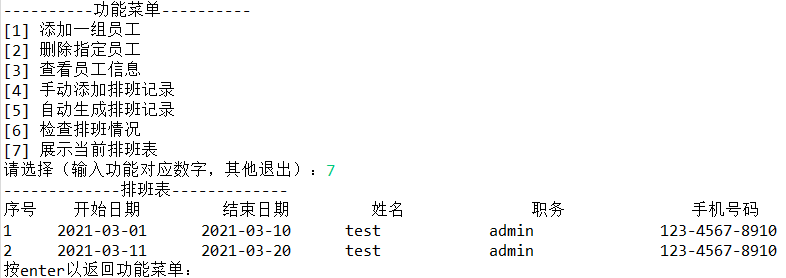


总计修改了大约50行，且所耗费时间不到半小时，不足的是部分破坏了原有的方法，但程序表现正常。如下图，修改后可以出现一个员工被安排多段值班的情况：





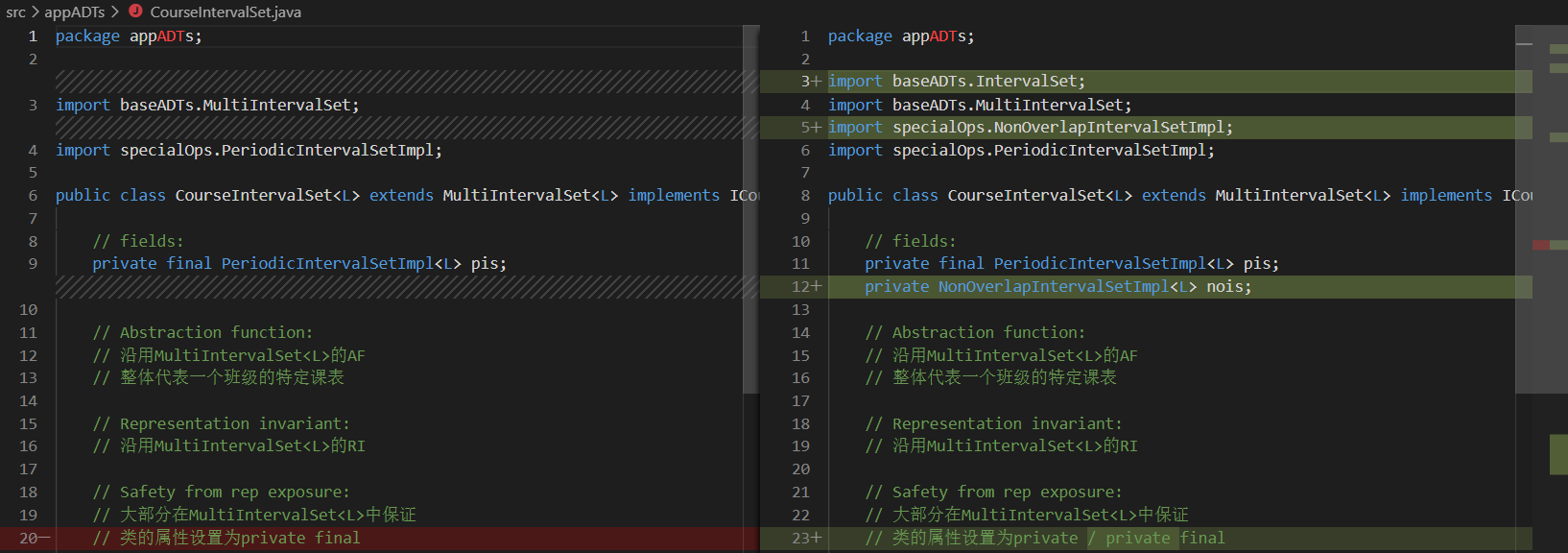


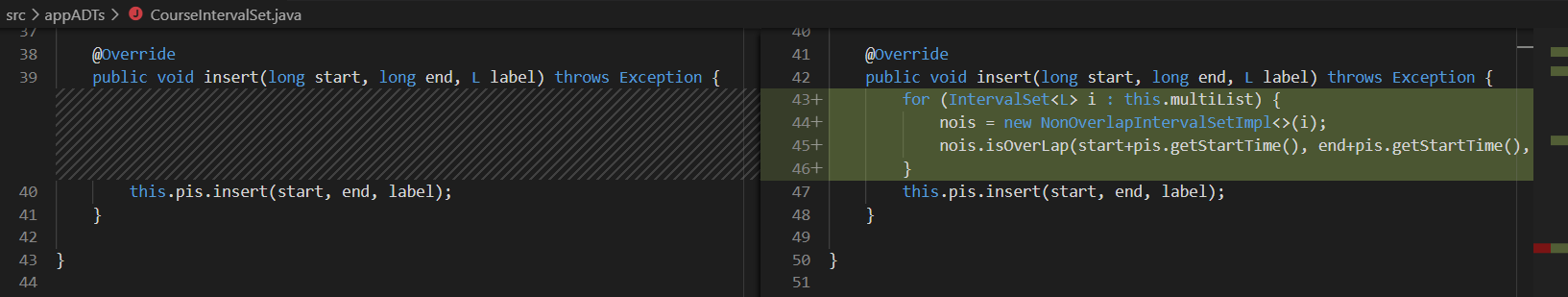


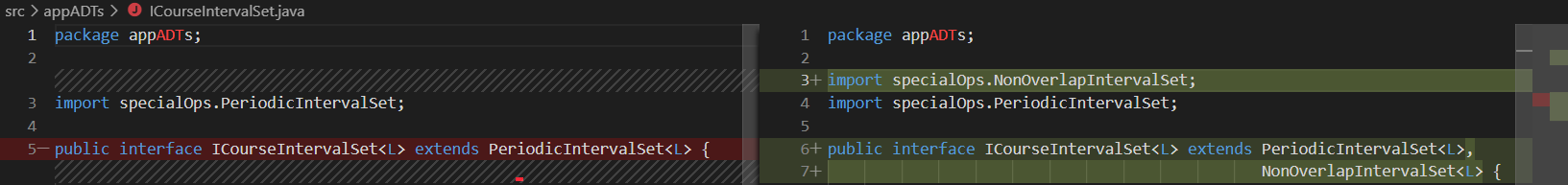
### 变化2

之前的设计可以应对变化，且修改难度不是很大。预计要修改两个文件，包括ICourseIntervalSet接口的继承关系、CourseIntervalSet类的私有属性（用于实现委托）和insert()方法实现，不需要修改APP。具体修改如下：

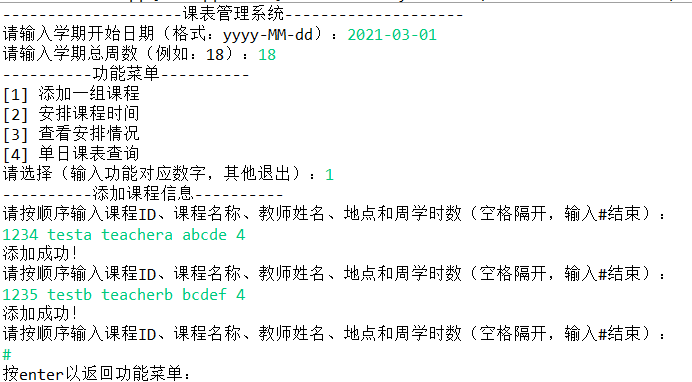
（图见下页）

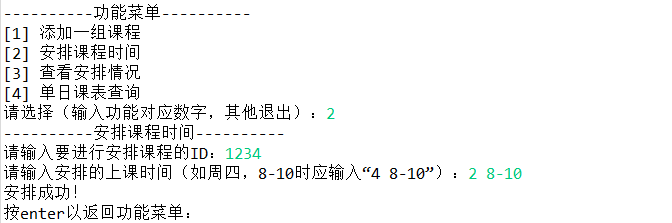


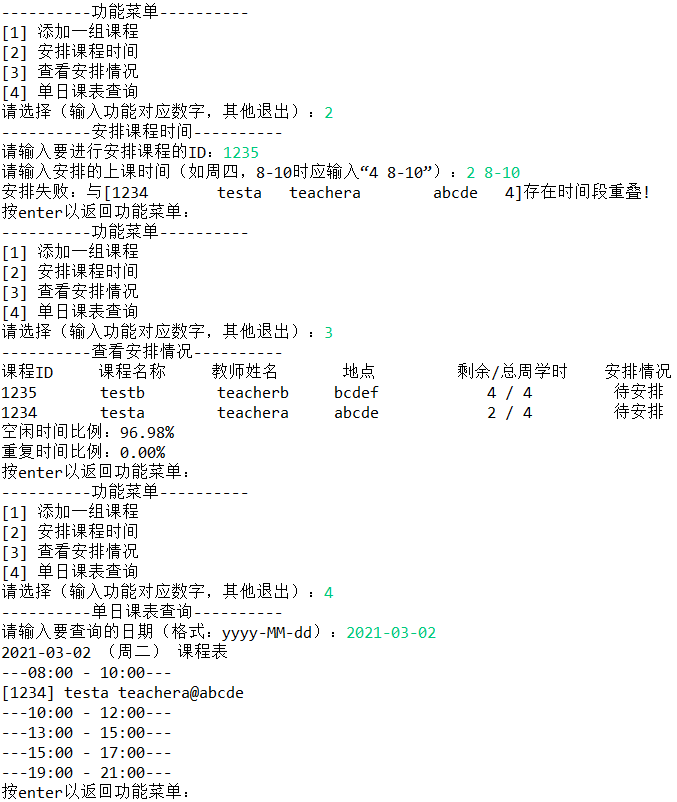


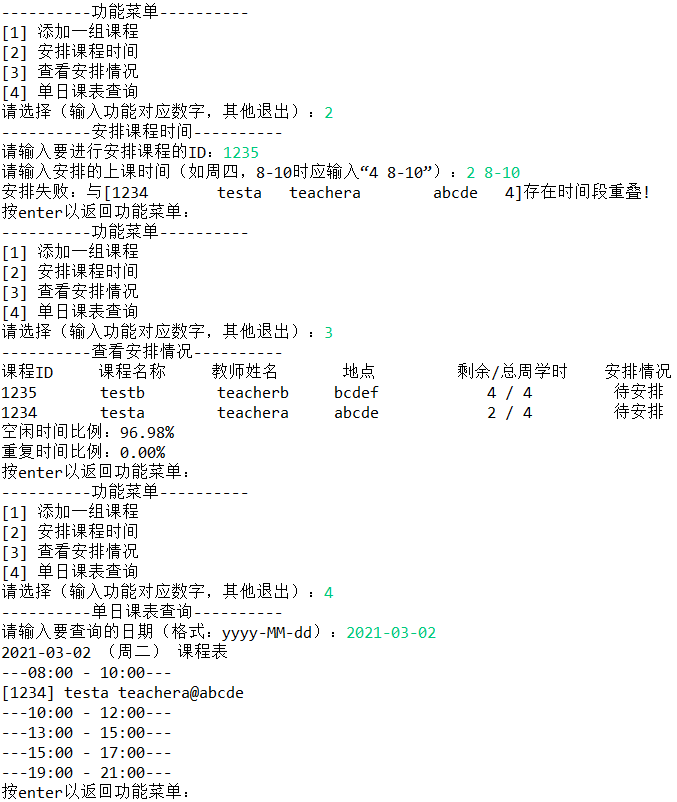


总计修改了大约10行，耗费时间几乎可以不计。如下图，程序现在可以实现课表的无重叠（插入时报错并返回）。







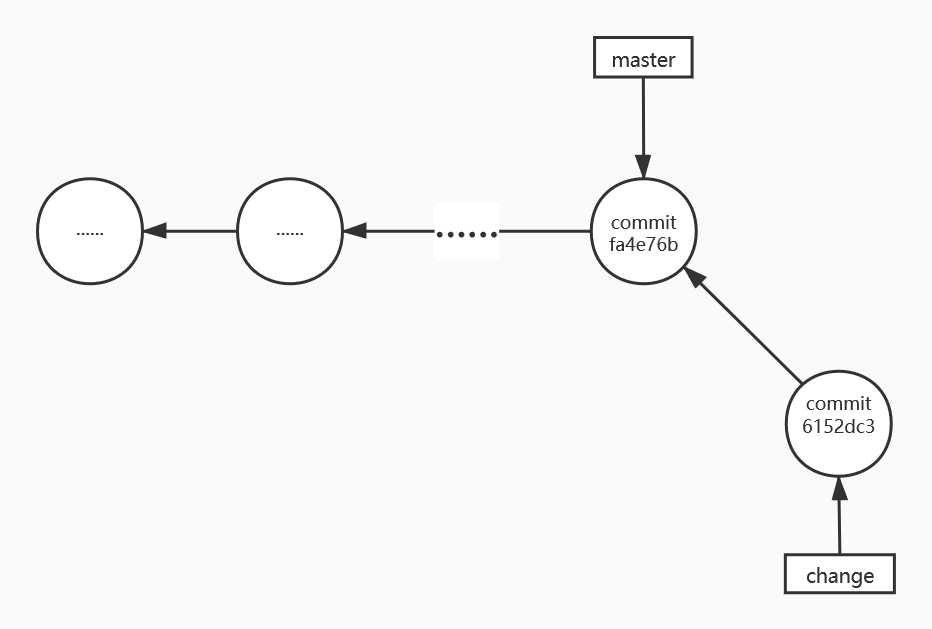


## Git仓库结构

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。

使用git log指令分别获取了两个分支的commit信息，依据信息画出了Git仓库结构示意图。

（图见下页）



说明：master分支当前最新commit为fa4e76b“Some adjustments.”

change分支当前最新commit为6152dc3“change”

由于需再次提交才能将本部分内容更新到库中，最终的Object Graph应该与此不同，master分支应该会指向更后面的commit。

# 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。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2021.6.14 | 13:20 - 14:00 | 完成接口IntervalSet<L>的设计 | 按计划完成 |
| 2021.6.16 | 19:40 - 21:00 | 编写测试并实现CommonIntervalSet<L> | 按计划完成 |
| 2021.6.17 | 15:00 - 15:30 | 初步设计MultiIntervalSet<L> | 按计划完成 |
| 2021.6.17 | 15:50 - 17:15 | 完成MultiIntervalSet<L>的设计与实现 | 按计划完成 |
| 2021.6.17 | 17:50 - 21:30 | 编写MultiIntervalSet<L>的测试，确定并初步构思设计方案五 | 按计划完成 |
| 2021.6.18 | 14:00 - 17:00 | 调整项目结构，编写标签ADT | 按计划完成 |
| 2021.6.21 | 14:00 - 17:00 | 编写方案五的前两个维度 | 基本完成 |
| 2021.6.23 | 14:30 - 17:30 | 实现全部三个维度 | 按计划完成 |
| 2021.6.26 | 10:00 - 11:30 | 实现DutyIntervalSet<L> | 遇到问题 |
| 2021.6.26 | 13:00 - 16:30 | 实现其余两个应用ADT | 按计划完成 |
| 2021.6.27 | 14:30 - 17:30 | 测试、完善两个应用ADT | 按计划完成 |
| 2021.6.28 | 9:00 - 11:00 | 设计可复用API的前半部分 | 按计划完成 |
| 2021.6.28 | 13:00 - 16:00 | 完成可复用API的设计并测试 | 按计划完成 |
| 2021.6.28 | 19:00 - 22:00 | 设计DutyRosterApp | 基本完成 |
| 2021.6.29 | 9:00 - 11:30 | 完善DutyRosterApp，初步设计ProcessScheduleApp | 按计划完成 |
| 2021.6.29 | 13:00 - 17:00 | 完善ProcessScheduleApp的设计，初步设计CourseScheduleApp | 按计划完成 |
| 2021.6.29 | 18:30 - 21:30 | 完成三个APP的设计 | 按计划完成 |
| 2021.6.30 | 9:00 - 11:00 | 初步实现基于语法的数据读入 | 遇到问题 |
| 2021.6.30 | 15:30 - 17:30 | 完成基于语法的数据读入 | 按计划完成 |
| 2021.6.30 | 18:30 - 21:00 | 测试、完善各部分内容，调整可视化格式 | 按计划完成 |
| 2021.7.1 | 16:00 - 17:15 | 初步设计变化 | 按计划完成 |
| 2021.7.1 | 18:00 - 22:40 | 完成实验全部内容，完善实验报告，进行在线测试，提交 | 按计划完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 对CRP原则的具体实现不熟悉，不了解如何进行组合、委托和复用。 | 复习课件上的相关内容，结合实验指导书进行尝试，与其他同学交流，最终理解了相关概念和实现方法，完成实验要求。 |
| 对Java中日期的相关的处理不太熟悉。 | 在网上查找相关教程并结合应用要求自行编写尝试，最终形成了可应用于本次实验的帮助类dateHandler（可查看src/helpers/dateHandler.java）。 |
| 对Java中正则表达式的书写规则、具体应用了解不够充分。 | 复习了课件上的相关内容，阅读了网上的相关教程，最终顺利完成了基于语法的读入。 |
| 基于语法的读入出现问题。 | 通过与同学交流得知，文件中空白字符并非UTF-8编码，正则表达式中的“\s\*”无法匹配，最后修改了文件中的空白符号，问题解决。 |

# 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

1. 认识到开发软件时复用ADT的巨大优势：减少重复工作量、更加适应变化等；
2. 更加深刻地理解了课上所学的继承、委托、重写等概念，复习了相关知识点；
3. 学会使用CRP原则设计简单的具有一定可复用性和可维护性的软件；
4. 学会使用正则表达式解析输入、处理数据；
5. 初次尝试结合使用接口、抽象类、类进行软件开发，深化了对Java项目结构的理解。

## 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？

答：面向ADT的编程更抽象、更通用，而直接面向应用场景编程更具体，同时也需要考虑应用场景的共同点，减少重复性不必要的工作；复用在这次实验中帮助我减少了许多工作量，提高了整体效率。

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？

答：意义是保证程序的正确性、健壮性、安全性和可维护性，提高程序质量，避免编程的实际工作脱离目标，且便于用户使用。我愿意在以后编程中坚持这样做。

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？

答：API的开发需同时考虑到通用性和具体的功能实现，对我来说有一定难度。但最终完成开发，通过自己编写的测试用例并应用到APP中也是很有成就感的。

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？

答：语法驱动编程对输入有着较为严格的要求，但也方便了程序的读入和处理。通过语法驱动编程进行开发能使用户与软件的交互更加直接。

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？

答：难度主要体现在ADT之间的继承、实现、复用等关系上，开发单个ADT并不是很难，但为了实现方案五，我耗费了大量的时间来调整ADT之间的关系，特别是面向APP的ADT与特殊维度操作类之间的委托关系。我克服这些困难的方法主要是不断尝试和吸取错误教训。

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。本实验的三个应用既不能完全抽象为同一个ADT，也不是完全个性化，如何利用“接口、抽象类、类”三层体系以及接口的组合、类的继承、设计模式等技术完成最大程度的抽象和复用，你有什么经验教训？

答：①注意寻找应用共同点，区分应用不同点；②从多个维度出发，从抽象到具体；③利用好三层体系之间的关系以及各种设计模式等才能最大程度上实现复用，减少工作量；④不能仅关注局部特性，过于依赖于特定实现。

1. 关于本实验的工作量、难度、deadline。

答：本次实验工作量很大，难度也比较大，如果考虑即将到来的期末考试并为复习预留时间，deadline也比较紧，需要兼顾速度和质量。

1. 到目前为止你对《软件构造》课程的评价。

答：通过软件构造课程学到了软件设计、开发和Java编程的很多相关知识、方法和技能。我认为这门课有很强的实际意义，对未来的职业发展也十分重要。此外也建议调整课程内容安排，让同学们有更充足的时间进行实验、预习和复习。