

**2021年春季学期**

**计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 王家琪 |
| 学号 | 1190301610 |
| 班号 | 1903603 |
| 电子邮件 | 2509260383@qq.com |
| 手机号码 | 15663525758 |

**目录**

2 实验环境配置

3 实验过程

3.1 待开发的三个应用场景

3.2 面向可复用性和可维护性的设计：IntervalSet<L>

3.2.1 IntervalSet<L>的共性操作

3.2.2 局部共性特征的设计方案

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）

3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L>

3.3.1 MultiIntervalSet<L>的共性操作

3.3.2 局部共性特征的设计方案

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）

3.4 面向复用的设计：L

3.5 可复用API设计

3.5.1 计算相似度

3.5.2 计算时间冲突比例

3.5.3 计算空闲时间比例

3.6 应用设计与开发

3.6.1 排班管理系统

3.6.2 操作系统的进程调度管理系统

3.6.3 课表管理系统

3.7 基于语法的数据读入

3.8 应对面临的新变化

3.8.1 变化1

3.8.2 变化2

3.9 Git仓库结构

4 实验进度记录

5 实验过程中遇到的困难与解决途径

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验和教训

6.2 针对以下方面的感受

# **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**

GitHub Lab3仓库的URL地址（HIT-Lab3-学号）。

https://github.com/ComputerScienceHIT/lab3-reusability-and-maintainability-programming-2021-1190301610

# **实验过程**

## **待开发的三个应用场景**

三个应用为：排班表，进程，课表。需求如下：

(1) 值班表管理（DutyRoster）：一个单位有 n 个员工，在某个时间段内（例如寒假 1 月 10 日到 3 月 6 日期间），每天只能安排唯一一个员工在单位值班，且不能出现某天无人值班的情况；每个员工若被安排值班 m 天（m>1），那么需要安排在连续的 m 天内。值班表内需要记录员工的名字、职位、手机号码，以便于外界联系值班员

(2) 操作系统进程调度管理（ProcessSchedule）：考虑计算机上有一个单核 CPU，多个进程被操作系统创建出来，它们被调度在 CPU 上执行，由操

作系统决定在各个时段内执行哪个线程。操作系统可挂起某个正在执行的进程，在后续时刻可以恢复执行被挂起的进程。可知：每个时间只能 有一个进程在执行，其他进程处于休眠状态；一个进程的执行被分为多个时间段；在特定时刻，CPU 可以“闲置”，意即操作系统没有调度执行任何进程；操作系统对进程的调度无规律，可看作是随机调度。

(3) 大学课表管理（CourseSchedule）：看一下你自己的课表，每一上午

10:00-12:00 和每周三上午 8:00-10:00 在正心楼 42 教室上“软件构造”课程。课程需要特定的教室和特定的教师。在本应用中，我们对实际的课表进行简化：针对某个班级，假设其各周的课表都是完全一样的（意即同样的课程安排将以“周”为单位进行周期性的重复，直到学期结束）；一门课程每周可以出现 1 次，也可以安排多次（例如每周一和周三的“软件构造”）且由同一位教师承担并在同样的教室进行；允许课表中有空白时间段（未安排任何课程）；考虑到不同学生的选课情况不同，同一个时间段内可以安排不同的课程（例如周一上午 3-4 节的“计算方法”和“软件构造”）；一位教师也可以承担课表中的多门课程。

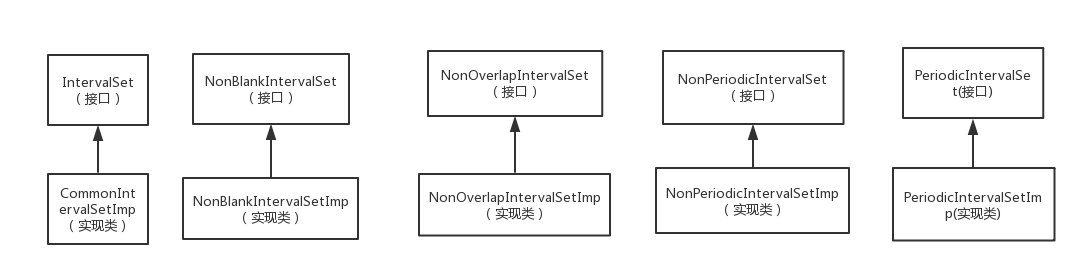
考虑上节三个应用，其中都包含了具有不同特征的“时间段集合”对象，为了提高软件构造的可复用性和可维护性，可为其设计和构造一套统一的ADT。

共同点：涉及到对于时间段的加入，删除，查找特定的时间段，以及对不同时间段进行标记。

差异性：一个标记是否可以加入多个时间段，加入的时间段是否可以重叠，以及整段时间段上是否有空白。

## **面向可复用性和可维护性的设计：IntervalSet<L>**

IntervalSet类是基础类，NonBlankIntervalSet,NonOverlapIntervalSet,NonPeriodicIntervalSet,PeriodicIntervalSet是四个不同的类，在基础类IntervalSet的基础上加入不同的新功能。



### **IntervalSet<L>的共性操作**

接口IntervalSet<L>的方法

创建一个空对象：empty()

在当前对象中插入新的时间段和标签：boolean insert(long start,

long end, L label)

获得当前对象中的标签集合：Set<L> labels()

从当前对象中移除某个标签所关联的时间段：boolean remove(L

label)

返回某个标签对应的时间段的开始时间：long start (L label)

返回某个标签对应的时间段的结束时间：long end (L label)

### **局部共性特征的设计方案**

利用CommonIntervalSet<L>实现IntervalSet<L>

CommonIntervalSet的实现方法如下：

|  |  |
| --- | --- |
| Rep | Set<L> Labels 储存标签 |
| Map<L,long[]> TimeSchedule 储存时间段 |
| boolean insert(long start, long end, L label) | 向TimeSchedule里面插入新元素，标志加入时间段 |
| Set<L> labels() | 对Labels进行防御式拷贝 |
| boolean remove(L label) | 同时移除Labels和 TimeSchedule里面的和label有关的元素 |
| long start (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[0] |
| long end (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[1] |

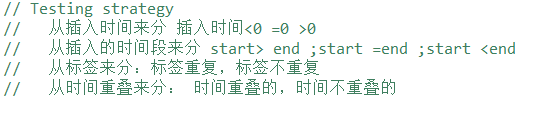
关键函数boolean insert(long start, long end, L label) 的函数代码如下：



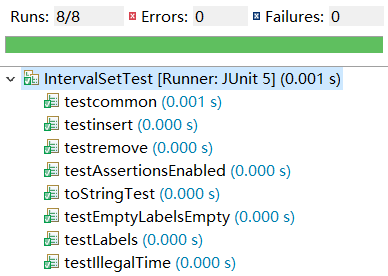
RI,AF,表示泄露如下：



测试结构如下：



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）**

设计了三个接口和相应的实现类，可以理解为是Intervalset的子类，只不过多加了一层接口，实现了自己独有的功能。

独有的功能简要概括如下：

**NonblankIntervalSet：**检查是否时间段有空白并返回空白时间段

**NonOverlapIntervalSet：**插入时间段要求没有重叠

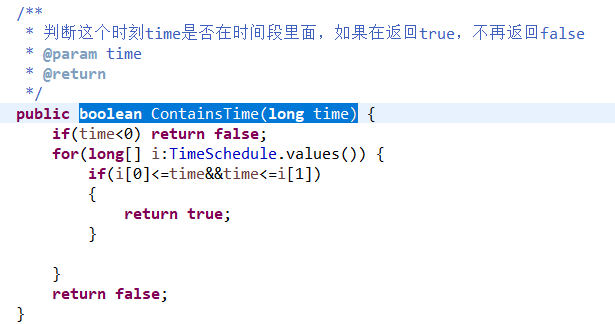
**NonPeriodicIntervalSet：**设置开始结束时间，时间无周期

**PeriodicIntervalSet：**设置开始结束时间，时间有周期

1. **实现类NonblankIntervalSet**

用NonblankIntervalSetImp的具体实现类实现NonblankIntervalSet这个接口，NonblankIntervalSetImp实现方法如下：

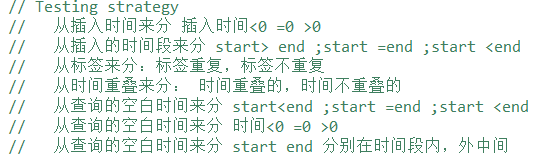
|  |  |
| --- | --- |
| Rep | Set<L> Labels 储存标签 |
| Map<L,long[]> TimeSchedule 储存时间段 |
| boolean insert(long start, long end, L label) | 向TimeSchedule里面插入新元素，标志加入时间段 |
| Set<L> labels() | 对Labels进行防御式拷贝 |
| boolean remove(L label) | 同时移除Labels和 TimeSchedule里面的和label有关的元素 |
| long start (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[0] |
| long end (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[1] |
| 独有的方法 | |
| List<Long> checkBlank(long start,long end) | 返回没有时间段覆盖的时间点的集合 |
| boolean checkIfBlank(long start,long end) | 查看给定时间段long~end之间是否有未被覆盖的时间段，检查checkBlank(long start,long end)是否为空 |
| boolean ContainsTime(long time) | 查看现在的ADT里面是否已经包含某个时间点time |

boolean ContainsTime(long time)的具体实现

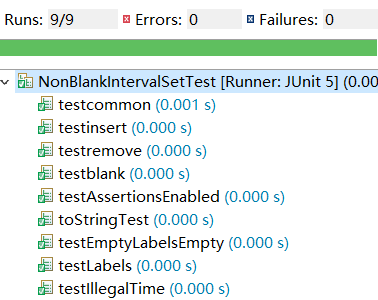
应用了抽象出来的boolean ContainsTime(long time)函数，List<Long> checkBlank(long start,long end) 的具体实现如下：



测试结构如下：



测试结果如下：



1. **实现类NonOverlapIntervalSet**

用NonOverlapIntervalSetImp的具体实现类实现NonOverlapIntervalSet这个接口，NonOverlapIntervalSetImp实现方法如下：

|  |  |
| --- | --- |
| Rep | Set<L> Labels 储存标签 |
| Map<L,long[]> TimeSchedule 储存时间段 |
| boolean insert(long start, long end, L label) | 重写方法，向TimeSchedule里面插入新元素，要求时间段不重叠 |
| Set<L> labels() | 对Labels进行防御式拷贝 |
| boolean remove(L label) | 同时移除Labels和 TimeSchedule里面的和label有关的元素 |
| long start (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[0] |
| long end (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[1] |

关键函数boolean insert(long start, long end, L label)实现方法如下：



1. **实现类NonPeriodicIntervalSet**

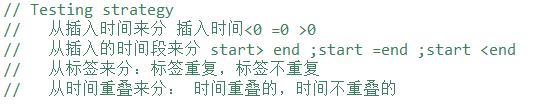
用NonPeriodicIntervalSetImp的具体实现类实现NonPeriodicIntervalSet这个接口，NonPeriodicIntervalSetImp实现方法如下：

|  |  |
| --- | --- |
| Rep | Set<L> Labels 储存标签 |
| Map<L,long[]> TimeSchedule 储存时间段 |
| boolean insert(long start, long end, L label) | 重写方法，向TimeSchedule里面插入新元素，要求时间段不重叠 |
| Set<L> labels() | 对Labels进行防御式拷贝 |
| boolean remove(L label) | 同时移除Labels和 TimeSchedule里面的和label有关的元素 |
| long start (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[0] |
| long end (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[1] |

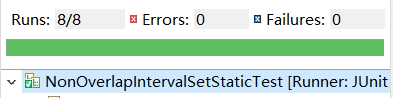
关键函数boolean insert(long start, long end, L label)实现方法如下，只是在CommonIntervalSet函数的基础上加了时间的判断，要求加入的时间段必须在Start~End（人为设置的时间段之内，否则加入无效）：



测试结构如下：



结果如下：



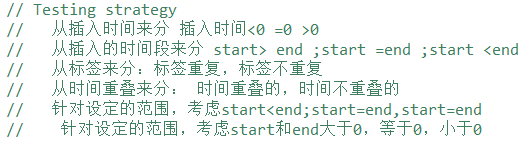
1. **实现类PeriodicIntervalSet**

用PeriodicIntervalSetImp的具体实现类实现PeriodicIntervalSet这个接口，PeriodicIntervalSetImp实现方法如下：

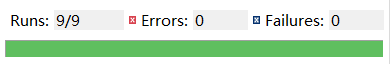
|  |  |
| --- | --- |
| Rep | Set<L> Labels 储存标签 |
| Map<L,long[]> TimeSchedule 储存时间段 |
| boolean insert(long start, long end, L label) | 重写方法，向TimeSchedule里面插入新元素，要求时间段不重叠 |
| Set<L> labels() | 对Labels进行防御式拷贝 |
| boolean remove(L label) | 同时移除Labels和 TimeSchedule里面的和label有关的元素 |
| long start (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[0] |
| long end (L label) | 返回TimeSchedule里面的key为label的value的第一个元素long[1] |
| checkTime(long start,long end) | 检查加入的时间是否在规定的时间之内，是返回true，否则返回false |

PeriodicIntervalSet是有周期性的时间段安排集合，所以在已有的insert，remove的基础上，还需要对加入的时间进行检测，对他的时间段进行取模分析。

测试的等价类划分如下：



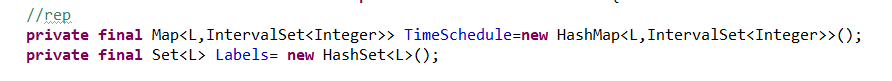
结果如下：



## **面向可复用性和可维护性的设计：MultiIntervalSet<L>**

### **MultiIntervalSet<L>的共性操作**

MultiIntervalSet<L>实现为接口。 CommonMultiIntervalSet<L>为这个接口的实现类。他的rep如下：



rep 中，TimeSchedule为标签——时间段的Map，一个标签可以加入多个时间段。Labels是所有Label的集合。

MultiIntervalSet<L>的方法包括：

1. 创建一个空对象：empty()或不带任何参数的构造函数
2. 创建一个非空对象：构造函数 MultiIntervalSet(IntervalSet<L>initial)，利用 initial 中包含的数据创建非空对象

3、在当前对象中插入新的时间段和标签：void insert(long start,

long end, L label)

1. 获得当前对象中的标签集合：Set<L> labels()

5、从当前对象中移除某个标签所关联的所有时间段：boolean remove(L

label)

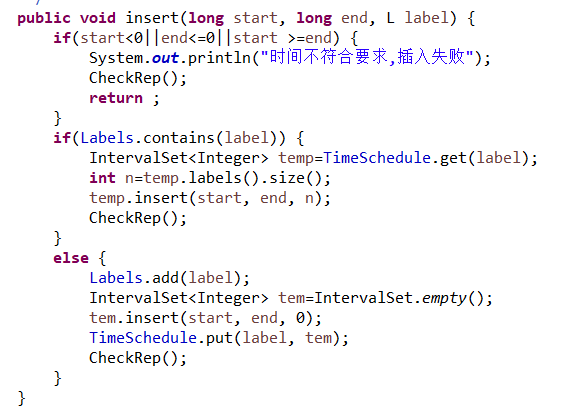
6、从当前对象中获取与某个标签所关联的所有时间段 ：

IntervalSet<Integer> intervals(L label)，返回结果表达为

IntervalSet<Integer>的形式，其中的时间段按开始时间从小到大的

次序排列。例如：当前对象为[[10,20]] }，那么 intervals("A")返回的结果是{ 0=[0,10], 1=[20,30] }。

**关键函数**insert函数实现逻辑如下：



### **局部共性特征的设计方案**

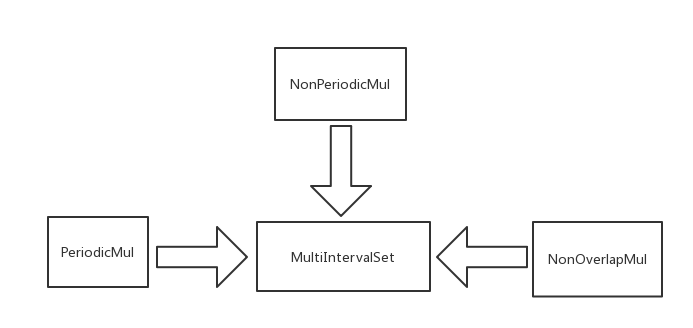
在已有的接口上，增加三个新接口来实现独有的功能，分别是

NonOverlap：插入时间段不允许重叠。

Periodic：总的时间是周期性的（针对课表）

NonPerodic：总的时间不是周期性的（针对进程资源管理）

这三个类在已有MultiIntervalSet的基础上实现对应的独有功能，自己没有insert，remove等操作方法，相当于这些接口只是增加了判断的方法，来帮助MultiIntervalSet来完善自己的功能。



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

1. **NonOverlapMul**

NonOverlapMul是一个接口，实现类为NonOverlapMulImp。

实现类的方法如下:

|  |  |
| --- | --- |
| 实现方法 | 说明 |
| empty（） | 初始化NonOverlapMul,创建一个空的对象 |
| Boolean insert（long start,long end,L label,MultiIntervalSet<L> mul） | 在in里面插入时间段start~end。只有时间段没有重叠的情况下才会加入，否则不做操作。  加入成功返回true，否则返回false |

关键函数insert的实现代码如下：

主要是思路是针对mul里面所有的时间段进行一次遍历，看现在要加入的时间段是否与已经加入的时间段有重叠。重叠情况包括三种：

开始时间位于某个时间段之内：s<start<e

结束时间位于某个时间段之内：s<end<e

开始时间和结束时间包含这个时间段：start<s<e<end

如果遍历之后发现没有重叠，再向mul里面加入这个时间段。



1. **NonPeriodicMul**

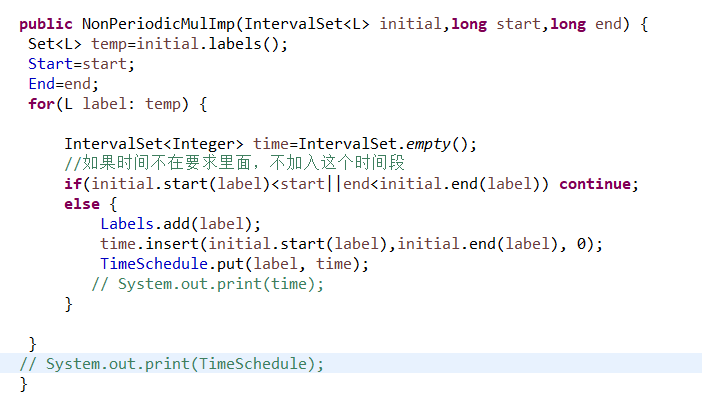
NonPeriodicMul是一个接口，实现类为NonPeriodicMulImp。

方法如下：

|  |  |
| --- | --- |
| 方法 | 说明 |
| empty(long start,long end) | 创建一个空对象，设置开始和结束时间 |
| creater (IntervalSet<L> initial,long start,long end) | 用一个已有的IntervalSet创建一个对象，将IntervalSet里面的内容复制，设置开始和结束时间 |
| boolean judgeTime(long start,long end) | 判断加入的时间段是否在规定的时间之内，是返回true，否则返回false |
| long BeginTime() | 返回整段时间的开始时间 |
| long EndTime() | 返回整段时间的结束时间 |

比较复杂的函数creater代码如下：

如果IntervalSet里面的时间不在规定的时间段之内，不加入这个时间



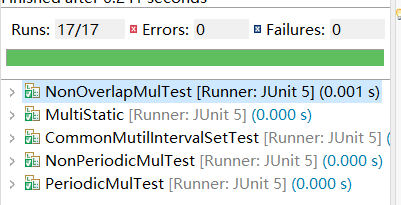
1. **PeriodicMul**

PeriodicMul是一个接口，实现类为PeriodicMulImp。

方法如下：

|  |  |
| --- | --- |
| 方法 | 说明 |
| empty（long end） | 创建空方法，设置周期的时间长度，  默认开始时间为0 |
| Boolean SetTime(long time) | 设置总时间 |
| Long transtime(long time) | 时间转化，将time转到0~end之间，取模运算 |

针对设计的如上四个类，分别进行测试，结果如下：



## **面向复用的设计：L**

所有的接口和方法都使了L，L在后面面向具体应用场景的时候，可以换成对象course，employee，process，也可以是一般的对象Integer，String等。

## **可复用API设计**

这部分的具体实现代码在三个应用类中进行了实现，没有专门封装一个java文件里面。具体实现代码见3.6的应用设计与开发。

### **计算相似度**

### **计算时间冲突比例**

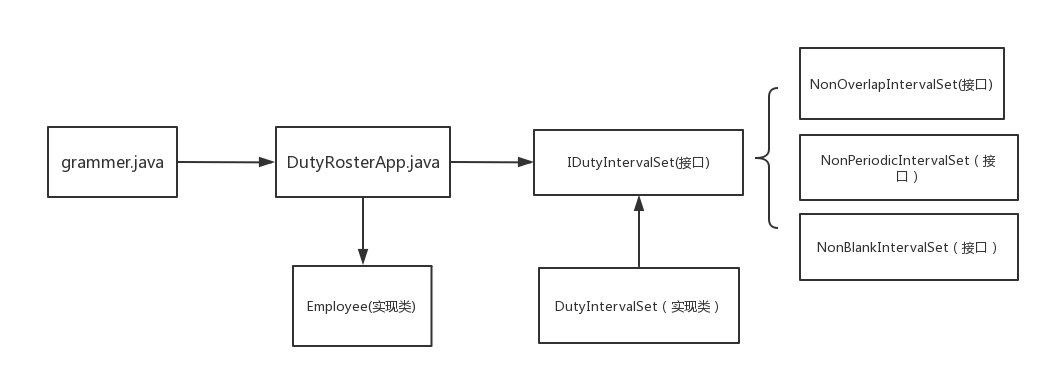
### **计算空闲时间比例**

## **应用设计与开发**

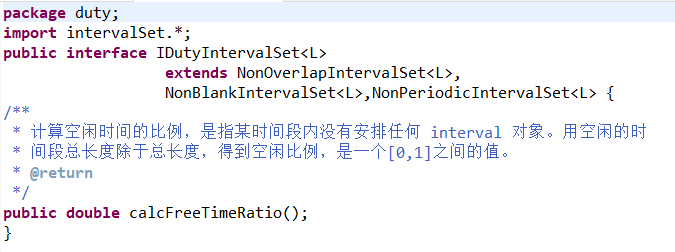
### **排班管理系统**

排班管理系统是IntervalSet性质的，一个标签只能加入一个时间段。考虑到现实要求中排班表的特性，他在三个维度分别属于：时间不可重叠，无空白，无周期。

结构如下：

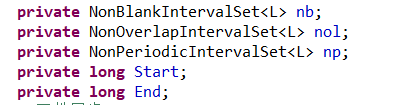


IDutyIntervalSet接口继承三个类，并在此基础上多加一个方法，计算当前空闲比例。



DutyIntervalSet实现IDutyIntervalSet接口

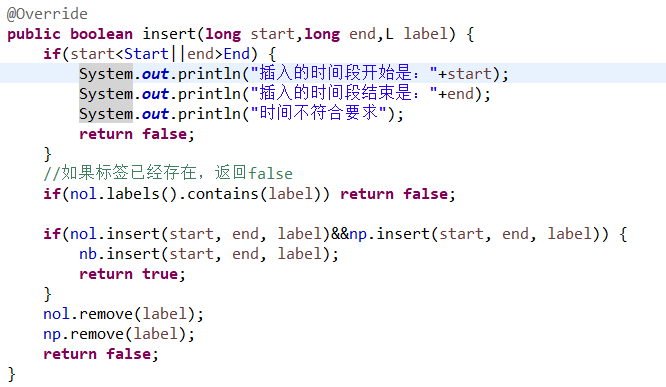
DutyIntervalSet的rep为：



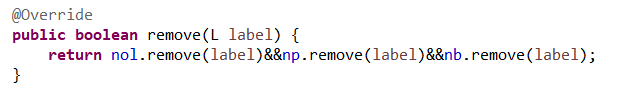
Insert函数的代码如下：

主要用了委派，insert和remove一起同步三次。

主要思路是三个IntervalSet一起加入某个时间段，只有三个IntervalSet都可以成功加入的时候，insert才合法，否则再将label从三个ADT里面delete。

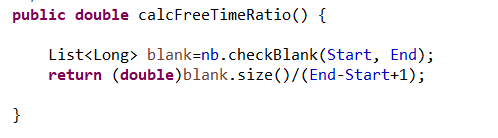


Remove的函数同理：



CalcFreeTimeRatio的函数代码如下：

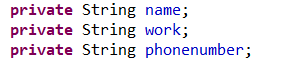
其中checkblank函数是调用了NonblankIntervalSet里面的方法，返回的是一个List<Long>数据结构的数组，其中的元素是未分配的时间。



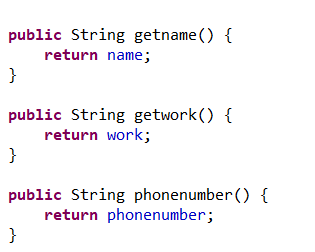
其余的方法只要利用其中一个ADT进行委派即可，我选用的是nol的方法。

Employee单独一个类，储存员工信息。

Rep如下：

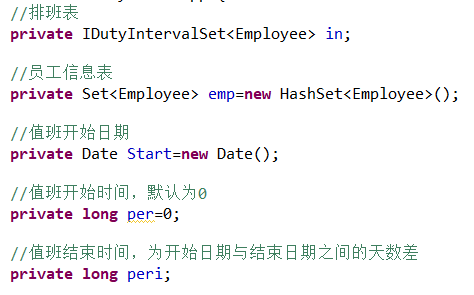


Employee里面的主要是观察方法。

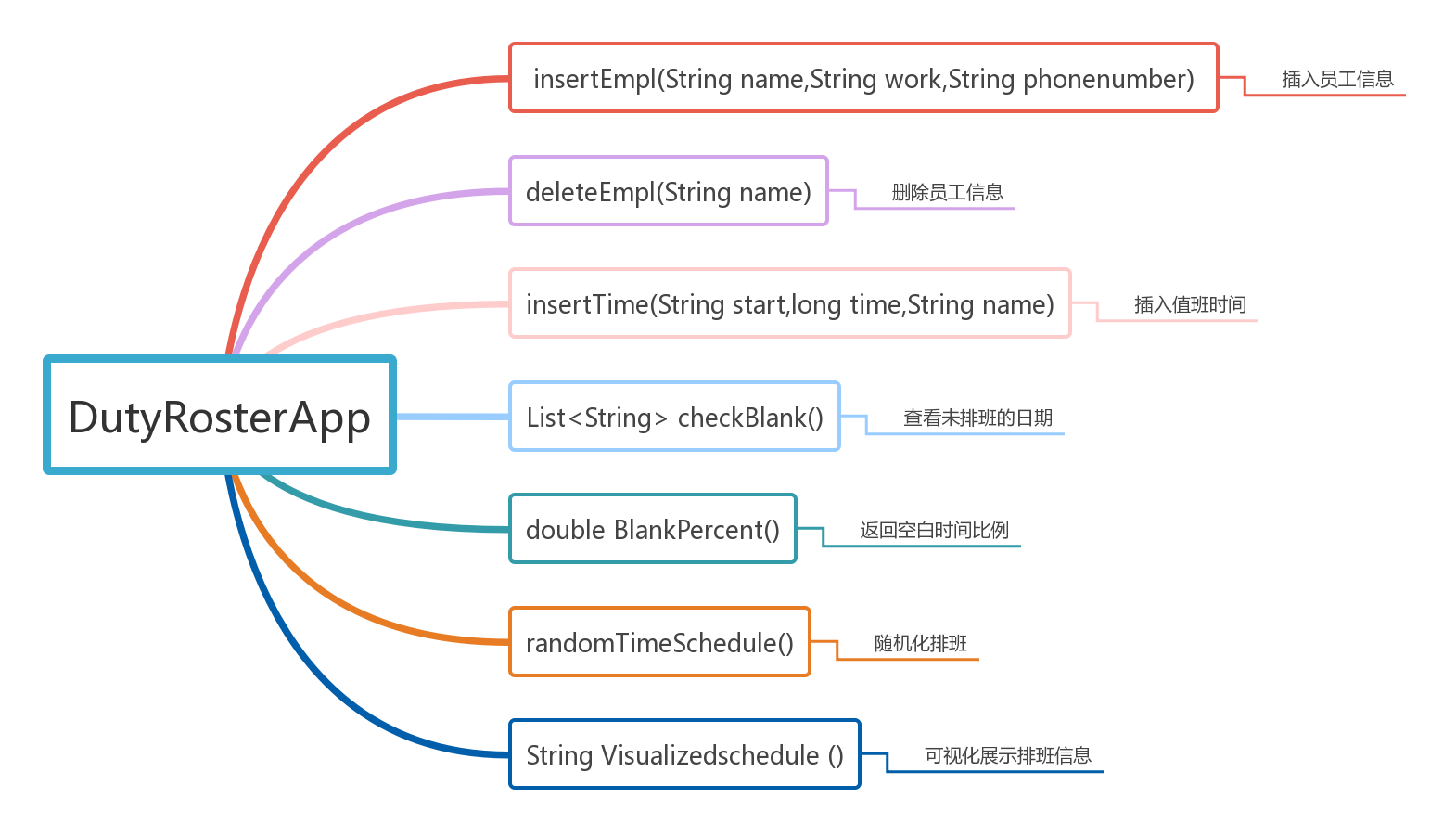


DutyRosterApp是真正的实现类。

Rep如下：除了运用我们之前设计的IDutyIntervalSet接口来记录排班信息，还用一个Set来记录员工的信息，值班开始的时间为Start，是输入的String类型的字符串转化为Date的结果，peri为值班的总天数，方便应用我们设计的ADT。



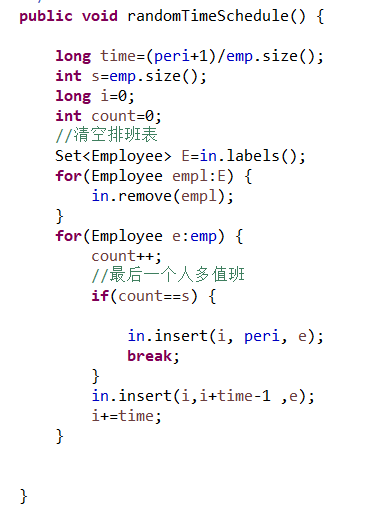
DutyRosterApp的方法如下：



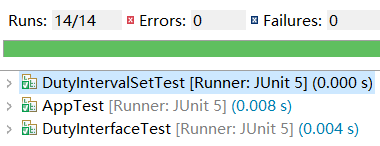
关键函数分析

RandomTimeSchedule()函数，实现功能随机排班：

实现的原则：将总的时间段平均分，给每个人尽量分相同时间的排班，每次调用randomTimeSchedule()函数，清除之前所有的排班信息，重新排班。



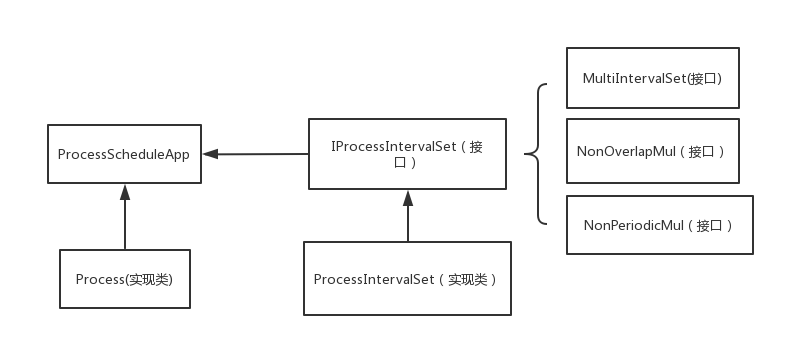
测试结果如下：



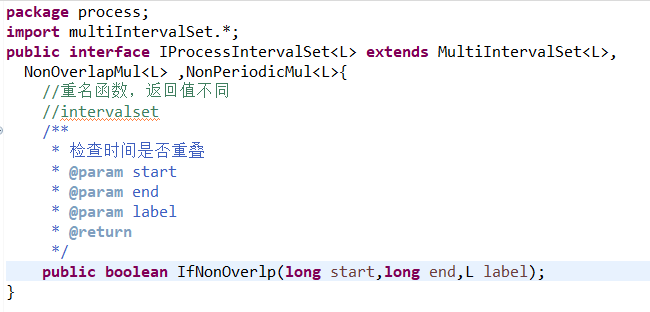
### **操作系统的进程调度管理系统**

ProcessScheduleApp是MultiIntervalSet性质的，一个标签可以加入多个时间段。考虑到现实要求中进程管理的特性，他在三个维度分别属于：时间不可重叠，无周期。对于空白时间没有什么要求。

ProcessScheduleApp的实现结构如下：



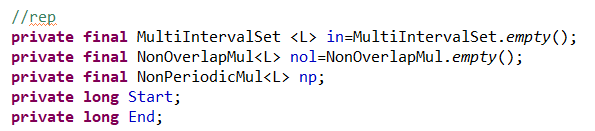
IProcessIntervalSet接口继承了之前定义的MultiIntervalSet, NonOverlapMul, NonPeriodicMul三个接口,在此基础上多定义了一个方法，IfNonOverlp函数，来判断加入的时间段是否与已经有的时间安排重叠。



ProcessIntervalSet是IProcessIntervalSet的实现类：

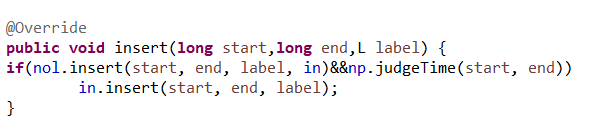
他的Rep如下：

调用了in，nol，np三个之前定义的接口，通过委派实现ProcessIntervalSet自己的功能。



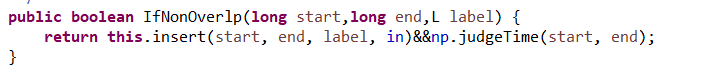
关键函数insert的实现如下：

关键点是在三个维度的ADT都可以加入，才可以在本ADT里面加入。需要用nol和np里面的独有方法，来判断是否可以在本ADT的MultiIntervalSet 里面加入该时间段。

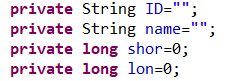


独有函数IfNonOverlp的实现如下：

通过调用本函数的insert和np里面的judgeTime函数,来判断是否符合本ADT的要求。为了之后在ProcessScheduleApp里面更方便的使用IProcessIntervalSet。



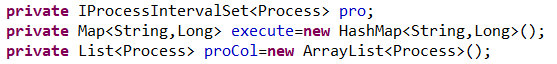
Process单独一个实现类，rep定义如下，包括process的ID，名字，最短执行时间和最长执行时间。



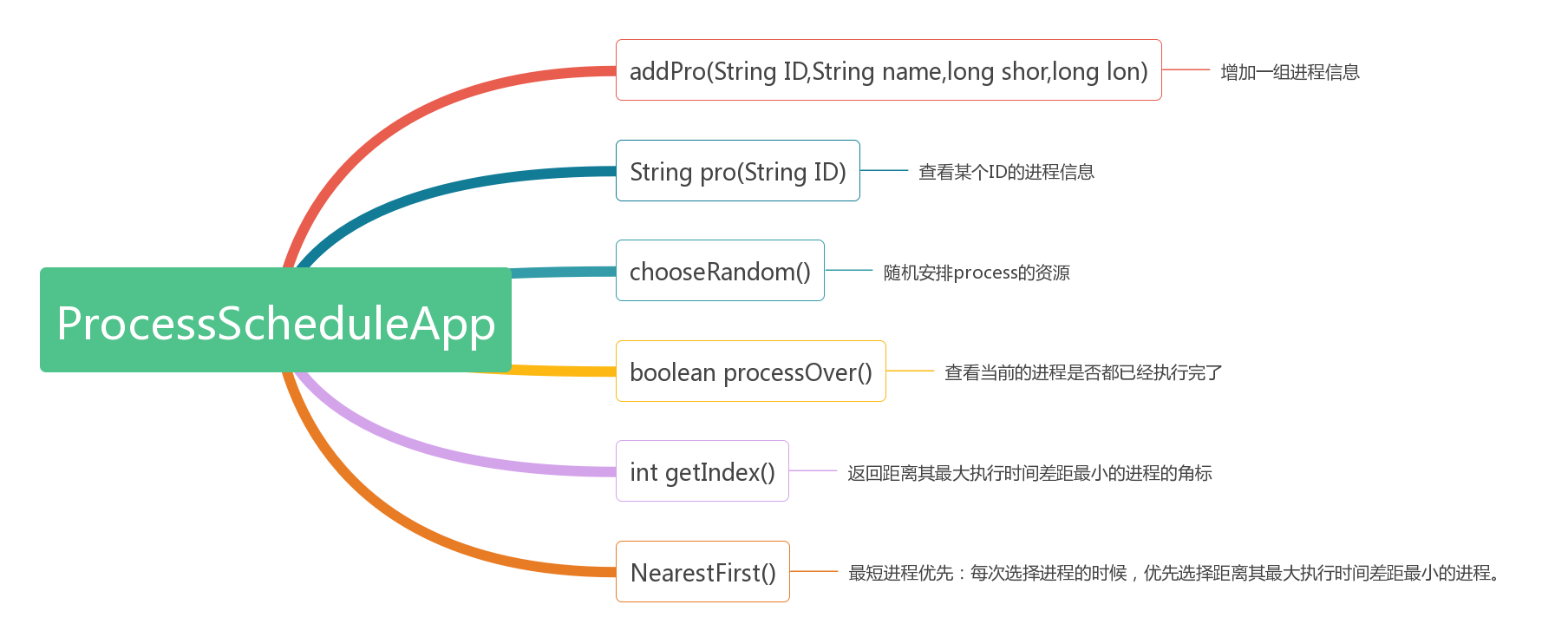
ProcessScheduleApp是真正的进程管理实现类。

rep如下：

除了应用我们已经定义的IProcessIntervalSet为进程管理专门设计的ADT之外，我们还专门定义了一个List<Process> proCol来记录Process的信息，一个Map来方便查找每个Process的执行时间，一开始初始化为0，之后每次执行，更新当前的执行时间。



方法如下：



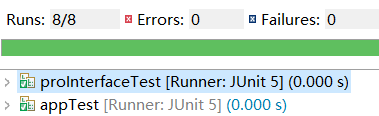
关键函数chooseRandom（）的代码如下：

利用java里面的rand函数库，每次随机分配选择进程和分配进程的开始时间、结束时间。利用processOver函数判断是否进程已经执行完成，进而判断是否可以停止随机进程资源管理函数。

具体的实现细节还有很多，都在下面的代码中进行了注释。

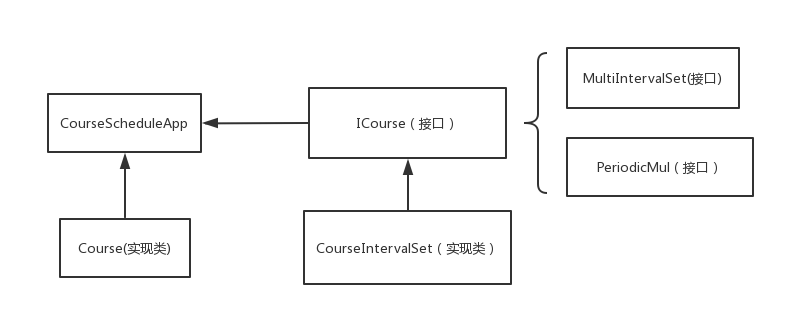


测试结果如下：



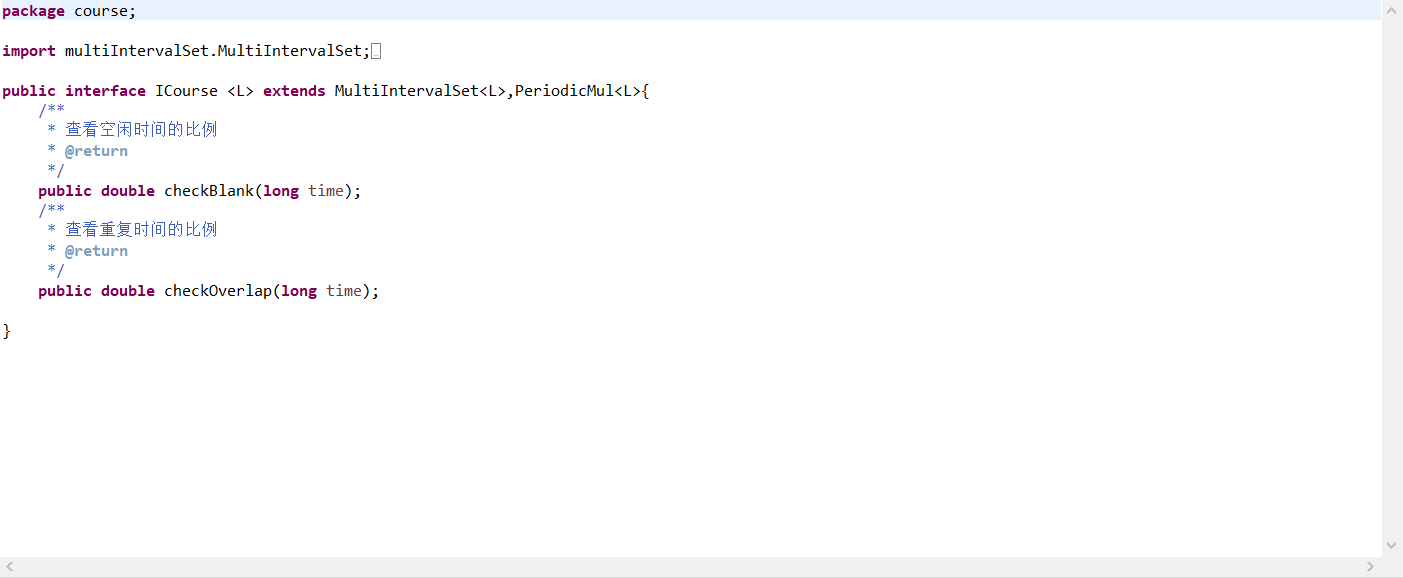
### **课表管理系统**

CourseScheduleApp是MultiIntervalSet性质的，一个标签可以加入多个时间段。考虑到现实要求中进程管理的特性，他在三个维度分别属于：时间可重叠，有周期。对于空白时间没有什么要求。

CourseScheduleApp项目的实现结构如下：

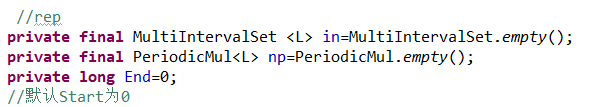
ICourse接口的定义如下:

除了继承基础MultiIntervalSet的两个类之外，为了检查空白时间，我们又重新定义了两个方法，也就是API.s里面要求实现的，checkBlank——检查空闲时间比例，checkOverlap——检查重叠时间的比例。



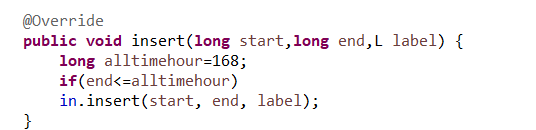
CourseIntervalSet是ICourse的具体实现类:

Rep如下：



关键函数的实现：

Insert函数：（设置的结束时间是一周，也就是168小时，为了之后的课表设计）



checkBlank函数：

思路：遍历0~168的每个long，看看是否这个时间点在已经加入的时间段里面（这需要遍历一下in里面的所有IntervalSet的时间段），利用count来计数，最后1-count/alltimehour，来返回空闲时间比例。



checkOverlap函数：

思路：遍历0~168的每个long，看看是否这个时间点在加入的时间段里面出现至少两次（这需要遍历一下in里面的所有IntervalSet的时间段），利用count来计数，最后count/alltimehour，来返回重叠时间比例。

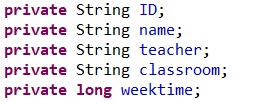
代码如下：



Course是一个课程的实现类，同process和employee一样，以观察方法为主。

Rep为:

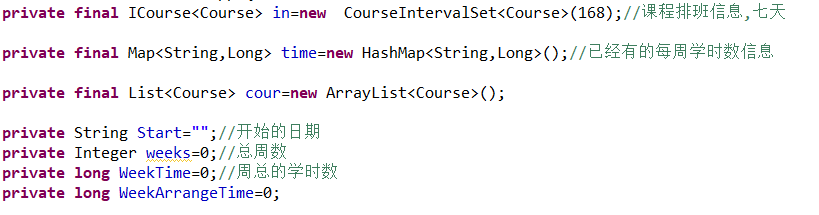
其中ID为课程的编号，每个课程不同，name为课程名字，teacher为课程的任课老师，classroom为课程的上课教室，weektime为该门课程每周最多安排的课时数（偶数）。



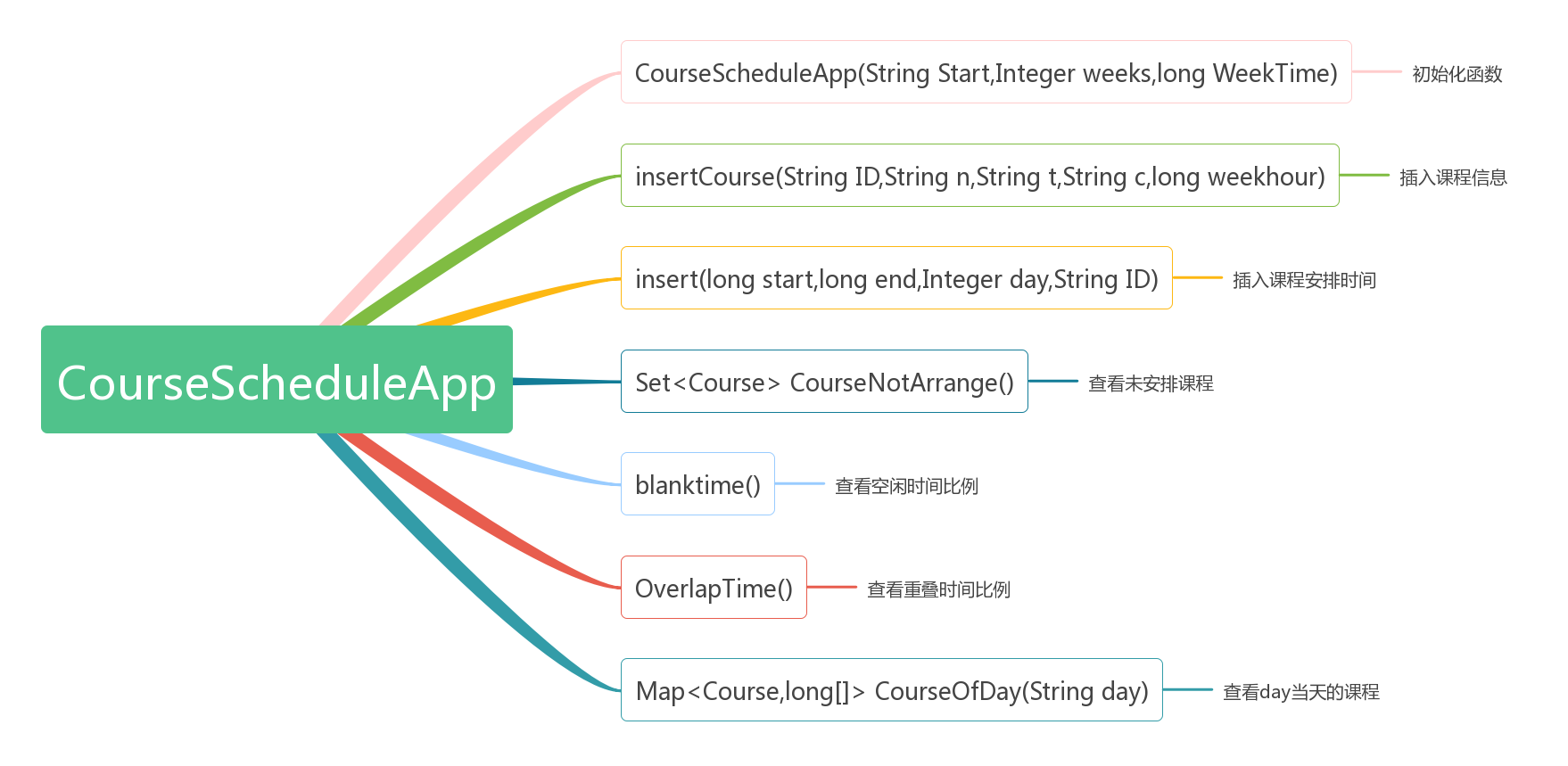
CourseScheduleApp是真正的课表管理实现类。

rep如下：

除了应用我们已经定义的ICourse为进程管理专门设计的ADT之外，我们还专门定义了一个List<Course> cour来记录Course的信息，一个Map来方便查找每个Course的排课时间，一开始初始化为该门课程的最大排课时间，之后每次执行，更新当前的执行时间（减去已经安排的课程时间）。

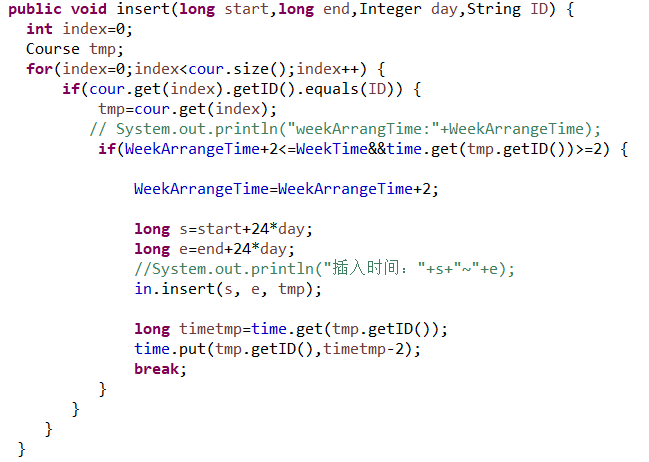


方法如下：

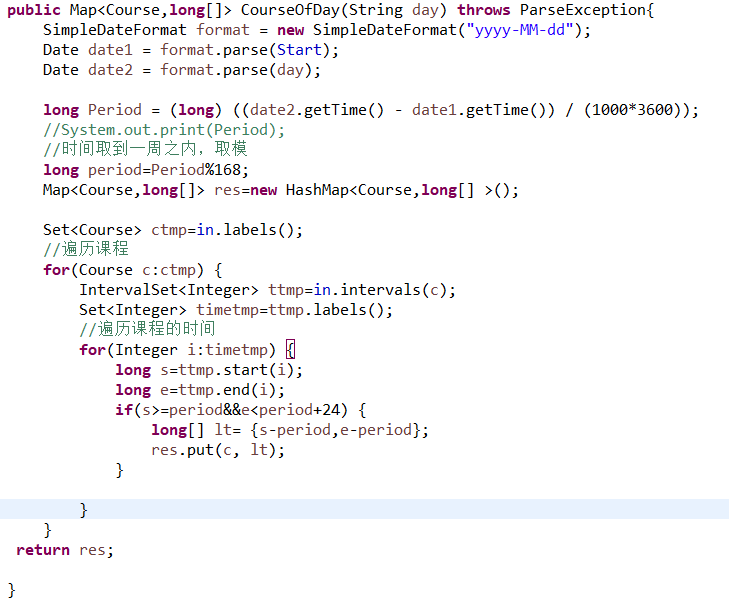


关键函数分析：

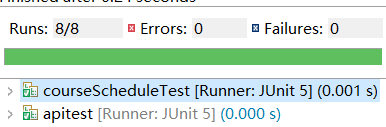
Insert——插入某天，某个时间段的课程安排，委派应用in的ADT方法，并且储存标签到time里面。



CourseOfDay——查看当天的课程安排，首先解析字符串为天数，然后对天数进行取模，变成一周内的莫一天，然后遍历查找数据结构in，将课程时间段在当天的加入到Map结构的res里面，最后返回res。



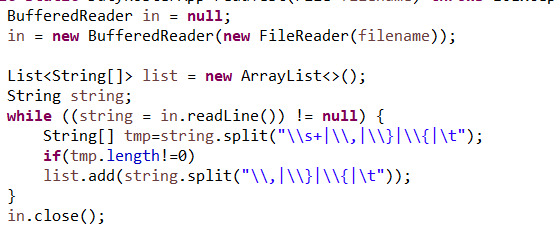
测试结果如下：



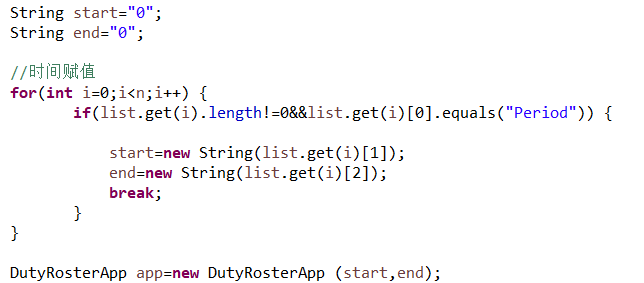
## **基于语法的数据读入**

要求：文件里描述的 Employee、Period、Roster 的次序是不确定的，违反该次序不能被看作非法。 在读取文件过程中，若发现有违反本例中解释说明部分给出的规则，则该文件不符合语法，需结束读取并ᨀ示用户选择其他合法的文件。另外，请忽略文件中的所有缩进，它并非语法的组成部分。

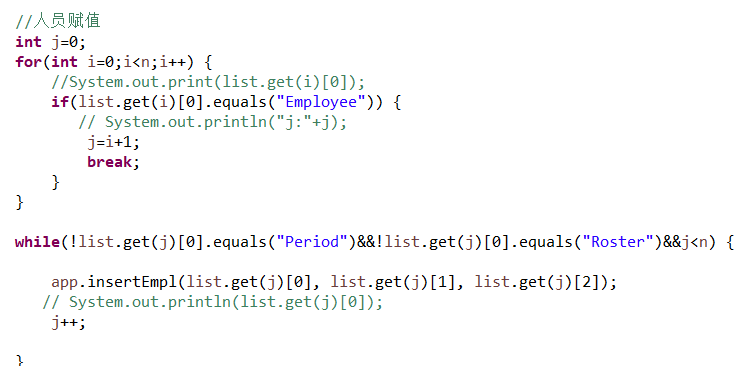
思路如下：首先按行读取文本文件，并将他储存，以，{，}为分隔符分割



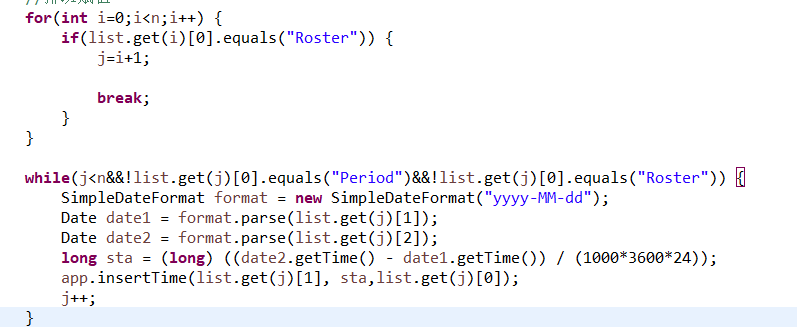
然后确定roster对应的开始时间和结束时间，初始化ADT。



然后插入employee的信息



插入排班时间信息：



最后对text文件进行测试，发现均满足要求。

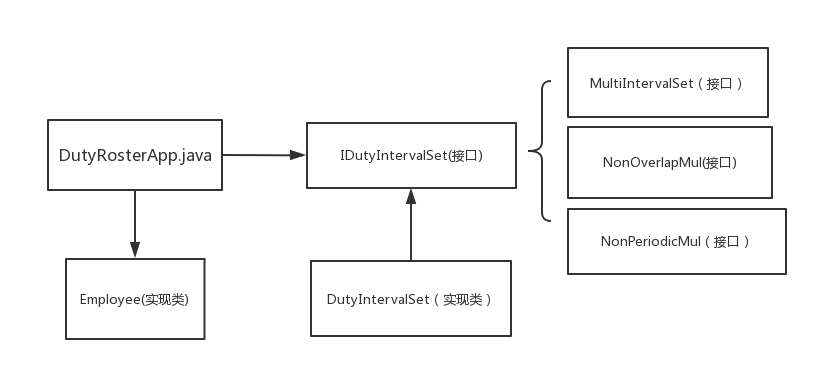
## **应对面临的新变化**

### **变化1**

排班应用：可以出现一个员工被安排多段值班的情况，例如张三的值班日期为(2021-01-01, 2021-01-10), (2021-02-01, 2021-02-06)；

修改：之前我们的dutyRosterApp是IntervalSet形式的，三个维度对应的是：时间段不可重叠，时间无周期，时间段无空白。现在只需要把IDutyIntervalSet的继承的接口改成MultiIntervalSet类型的即可。

duty.change的结构示意图如下：



IDutyIntervalSet的接口如图，除了继承的方法为，额外定义满足NonBlank性质的方法（类别IntervalSet里面的NonBlankIntervalSet的独有方法）。

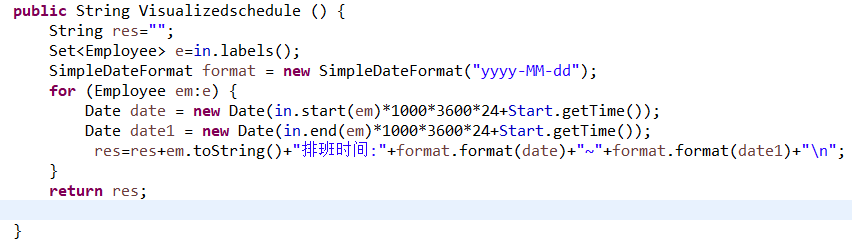


对于DutyIntervalSet和DutyRosterApp里面的实现，和之前未改变之前的实现思路类似，甚至代码也基本一致。

变化的根本在于：当遍历标签的时间段的时候，时间段有一个由long[],变成了多个时间段的集合IntervalSet<Integer>类型的，需要对此ADT里面的时间段再次进行遍历。

举个栗子：

Visualizedschedule ()，未变化前，他的代码是这样的。

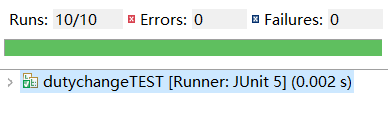


变化后，增加了对标签的时间段的循环遍历，也就是先返回label的intervals集合，在对他进行遍历，找到这个标签所有的排班时间段。



其他的功能变化也是同理。

测试用例直接对未修改前的测试用例进行增加就好：



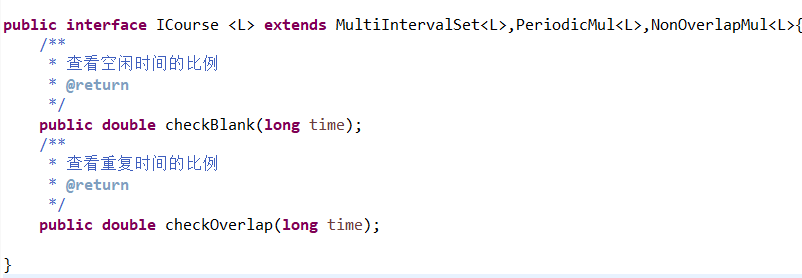
### **变化2**

课表应用：不管学生选课状况如何，不能够出现两门课排在同一时间的

情况（即“无重叠”）

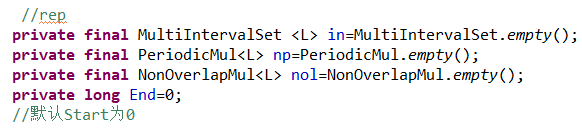
修改：修改只需要在原来的基础上，加上时间段不可重叠的特性即可。

首先在ICourse接口里面，多继承一个接口：

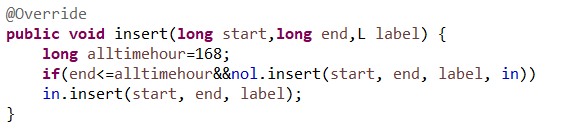


CourseIntervalSet里面的rep和实现方法也类似。

Rep里面加入一个新的ADT



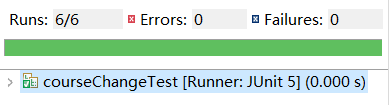
Insert函数进行修改，加入一个新的判断条件（时间段不可重叠）：



CourseScheduleApp里面，修改比较多的是insert函数，多加一个判断加入时间段是否成功的判断。具体的判断过程为，遍历ICourse类型的排课表in里面所有的时间段，看是否有符合加入的条件的时间段（start，end均相等），如果有，则证明时间段加入成功，满足NonOverlap的条件，然后就可以更新其他的数据结构，time的已经排课时间对应-2。



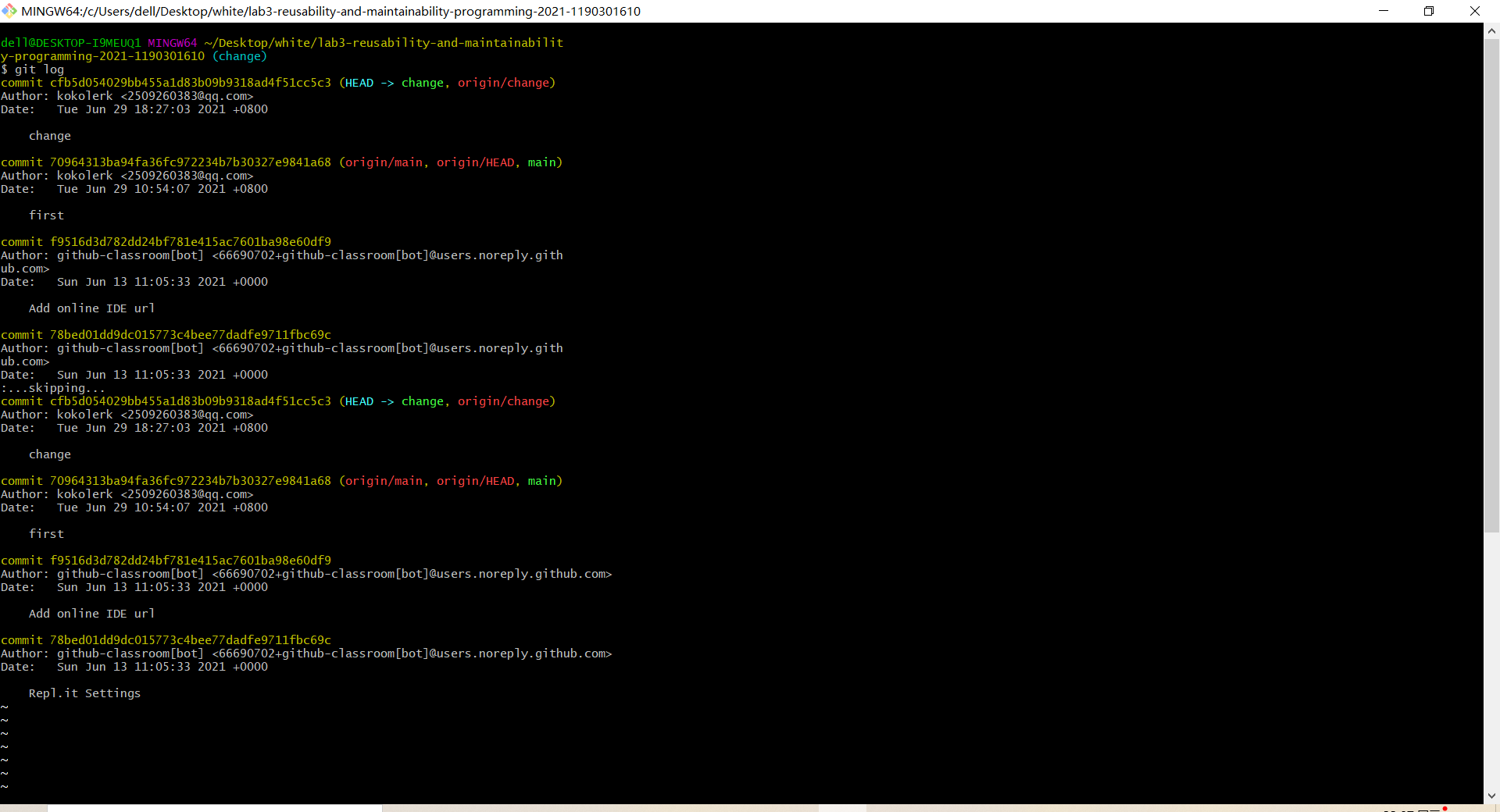
测试用例需要修改一下未修改前的测试用例关于Overlap的相关部分，剩下的不变。



## **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。

Change分支使用git log命令截图如下：



# **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 |
| 6.20 | 全天 | 完成multi和intervalset的接口 | 基本完成 |
| 6.22 | 全天 | 完成duty和process的应用代码 | 基本完成 |
| 6.25 | 下午 | 完成course的应用代码 | 基本完成 |
| 6.29 | 上午 | 完成正则语法的解析 | 基本完成 |
| 6.29 | 下午 | 变化 | 基本完成 |
| 6.30 | 全天 | 完成实验报告 | 基本完成 |

# **实验过程中遇到的困难与解决途径**

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 对于接口、抽象类、实现类之间的关系不清晰 | 通过写实验报告，理清思路。 |
| 接口到三个具体应用的委派关系不清晰 | 放弃高妙的委派策略，之间一视同仁，对所有的接口进行一样的操作。虽然进行了空间的浪费，但是简化了代码。 |
| 有的代码重复使用了，感觉可复用性比较差。 | 发现这个问题，对该应用的底层接口进行改造，除了继承已有的接口（IntervalSet和MultiIntervalSet的接口）的方法外，增加量身定做的一些辅助性的新的功能，方便我们在之后app的实现过程中进行判断。 |

# **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**

感觉在做的过程中，想法和观念在不断的变化。接口和实现类改了三四遍，最后才确定下来，在修改的过程中，对于应用的实现也是一种帮助，因为对于各个接口之间的关系更加清晰，后面进行针对具体的应用和接口进行继承和委派的时候才会游刃有余。

## **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？

体会到了设计这一过程。ADT的rep，方法需要自己设计，ADT之间的关系也

需要自己把握。体会到了一定好处，委派方法降低了代码的重复度。

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？

意义就是自己的脑子不要乱，在面对复杂的情况下，很有可能我们implement的时候就忘记了自己当初要这个ADT干什么了。这样有助于编程的清晰和准

确。

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？

可以体会。抽象这个过程还是挺有趣的，你会觉得自己发现了一些本质性的东西，但是API需要考虑一般性，对于代码要求比较多，要进行多种情况的考虑。

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？

其实很有趣。之前的形式语言和自动机的课程进行了学习，现在应用到了具体的实例中，感觉挺不错的。找到一个可以一般化所有文字的生成语法也是一个需要技巧的过程。

1. Lab1和Lab2的工作都不是从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，经过三周之后，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？

难道主要体现在设计，包括抽象接口，进行接口之间的关系选择，设计ADT的rep和方法。设计其实很难，而且可能需要你再实现的过程中进行不断的修改。有很多巧妙的设计，可能目前我还没有经验，想不到。但是可以先考虑一些虽然不是很巧妙，比较笨，但是比较稳妥简单的设计模式。

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、委派、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？

**接口是很有必要的**，他是面向可复用性的基础，定义很多接口，应用的时候也就越灵活。抽象类是接口之下的一个抽象，我这次实现lab3没有用到太多抽象类，但是感觉用了很多接口可以代替抽象类。具体的实现类就是我最熟悉的代码了，对于实现类里面，有继承和委派两种关系，**现在感觉到了委派的好处**，因为新的类和父类面向的应用场景可能不同，如果继承所有的方法，可能有很多冗余的方法。但是委派就很灵活。Lab3几乎所有的实现类的关系都是委派。

1. 关于本实验的工作量、难度、deadline。

确实有点大，大概两周的工作量吧，关键和考试周赶到了一起就有点尴尬。

1. 下周就要进行考试了，你对《软件构造》课程总体评价如何？

感觉学习了两门课，一门软件构造原理，一门软件构造实验。收获很多的，实践性很强，提高了自己的实践动手能力。就是感觉上课讲的内容有点太理论化了，可能理解的不是很深。而实验的内容又不能全面设计上课的理论，导致上课讲的可能有点枯燥（只是有一点点）。