

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 傅浩东 |
| 学号 | 1190202105 |
| 班号 | 1903002 |
| 电子邮件 | [1091288450@qq.com](mailto:1091288450@qq.com) |
| 手机号码 | 13881165621 |

**目录**

[1 实验目标概述 1](#_Toc76234213)

[2 实验环境配置 1](#_Toc76234214)

[3 实验过程 1](#_Toc76234215)

[3.1 待开发的三个应用场景 1](#_Toc76234216)

[3.1.1 值班表管理（DutyRoster） 2](#_Toc76234217)

[3.1.2 操作系统进程调度管理（ProcessSchedule） 2](#_Toc76234218)

[3.1.3 大学课表管理（CourseSchedule） 2](#_Toc76234219)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 3](#_Toc76234220)

[3.2.1 IntervalSet<L>的共性操作 3](#_Toc76234221)

[3.2.2 局部共性特征的设计方案 4](#_Toc76234222)

[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） 6](#_Toc76234223)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 8](#_Toc76234224)

[3.3.1 MultiIntervalSet<L>的共性操作 8](#_Toc76234225)

[3.3.2 局部共性特征的设计方案 9](#_Toc76234226)

[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） 11](#_Toc76234227)

[3.3.3.1 面向Process进程安排的应用设计 11](#_Toc76234228)

[3.3.3.2 面向Course课表安排的应用设计 13](#_Toc76234229)

[3.4 面向复用的设计：L 15](#_Toc76234230)

[3.4.1 员工（Employee） 15](#_Toc76234231)

[3.4.2 进程（Process） 16](#_Toc76234232)

[3.4.3 课程（Course） 17](#_Toc76234233)

[3.5 可复用API设计 18](#_Toc76234234)

[3.5.1 计算相似度 18](#_Toc76234235)

[3.5.2 计算时间冲突比例 19](#_Toc76234236)

[3.5.3 计算空闲时间比例 20](#_Toc76234237)

[3.6 应用设计与开发 21](#_Toc76234238)

[3.6.1 排班管理系统 21](#_Toc76234239)

[3.6.2 操作系统的进程调度管理系统 26](#_Toc76234240)

[3.6.3 课表管理系统 27](#_Toc76234241)

[3.7 基于语法的数据读入 30](#_Toc76234242)

[3.8 应对面临的新变化 34](#_Toc76234243)

[3.8.1 变化1 34](#_Toc76234244)

[3.8.2 变化2 35](#_Toc76234245)

[3.9 Git仓库结构 36](#_Toc76234246)

[4 实验进度记录 37](#_Toc76234247)

[5 实验过程中遇到的困难与解决途径 38](#_Toc76234248)

[6 实验过程中收获的经验、教训、感想 38](#_Toc76234249)

[6.1 实验过程中收获的经验和教训 38](#_Toc76234250)

[6.2 针对以下方面的感受 38](#_Toc76234251)

# 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

Java环境实验一以及配置好，各种工具如Git也在实验一安装好。本实验要求安装配置的EclEmma（用于统计Junit测试代码覆盖度的plugin）在最新版的Eclipse中自带。

我的GitHub Lab3仓库的URL地址：

https://github.com/ComputerScienceHIT/HIT-Lab3-1190202105

# 实验过程

请仔细对照实验手册，针对每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## 待开发的三个应用场景

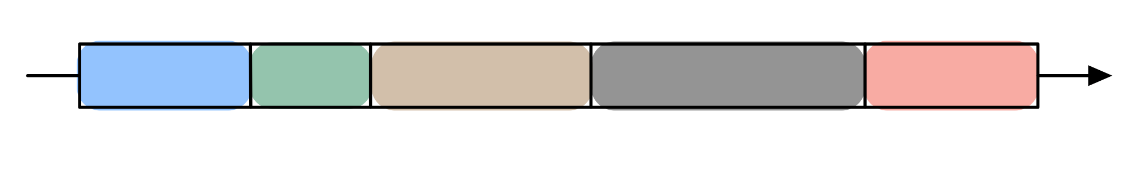
简要介绍三个应用。

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。

### 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。

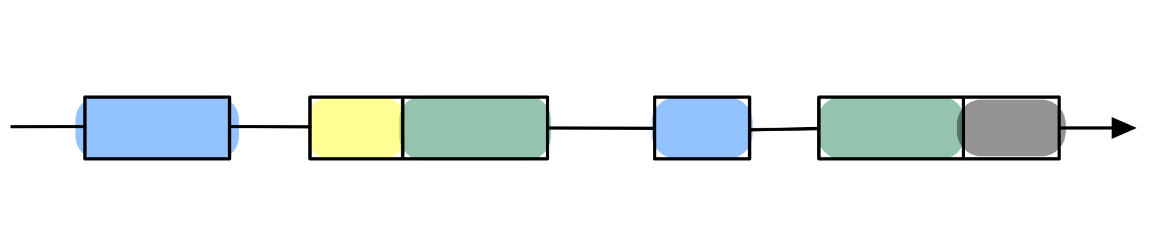
即每个员工值班在一个连续的时间段，不能让一个人出现在多个时间段，并且时间段之间不能有重合，最终的安排要使在某个期间内每个时间段都有人值班。具体图解如下：



### 操作系统进程调度管理（ProcessSchedule）

单核CPU，操作系统创建多个进程被调度在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。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。

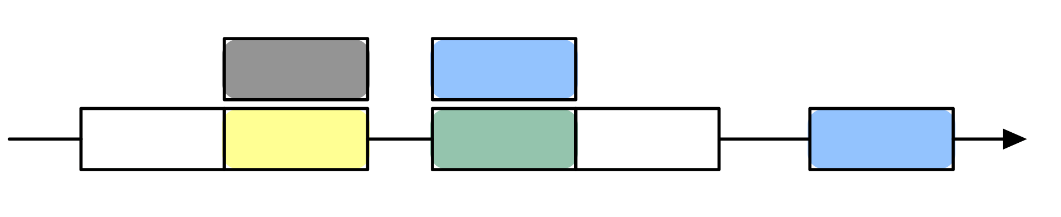
即在一个期间内，插入时间段，每个时间段的标志可以重复，但是不允许有时间段的重合部分，允许有空闲时间。图解如下：



### 大学课表管理（CourseSchedule）

课程安排将以“周”为单位进行周期性的重复，直到学期结束，每周课表完全相同，运行同一个时间段有重复课程。

允许有空闲时间、允许不同标签的时间段重复，并且是周期性存储的，每个周期内完全是相等的内容。如图一个周期内容：



## 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。

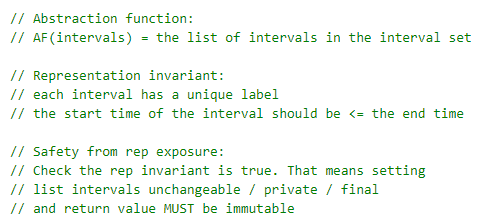
### IntervalSet<L>的共性操作

为接口IntervalSet设计应提供的共性接口方法：创建一个空对象、在当前对象中插入新的时间段和标签、获得当前对象中的标签集合、从当前对象中移除某个标签所关联的时间段、返回某个标签对应的时间段的开始时间、返回某个标签对应的时间段的结束时间、返回全部的时间段安排、改变某个标签、判断是否是空表。这些都放入IntervalSet<L>接口封装起来。

|  |  |
| --- | --- |
| **方法** | **含义** |
| empty() | 创建一个空对象 |
| boolean insert(long start, long end, L label) | 插入新的时间段和标签 |
| Set<L> labels() | 标签集合 |
| boolean remove(L label) | 移除某个标签所关联的时间段 |
| long start (L label) | 标签对应的时间段的开始时间 |
| long end (L label) | 标签对应的时间段的结束时间 |
| list<Interval<L>> getIntervals() | 全部的时间段及标签 |
| boolean changeLabel(L old, L new) | 改变某个标签为新标签 |
| boolean isEmpty() | 判断是否是空的安排表 |

CommonIntervalSet<L>是IntervalSet<L>的具体实现类，实现上述方法。

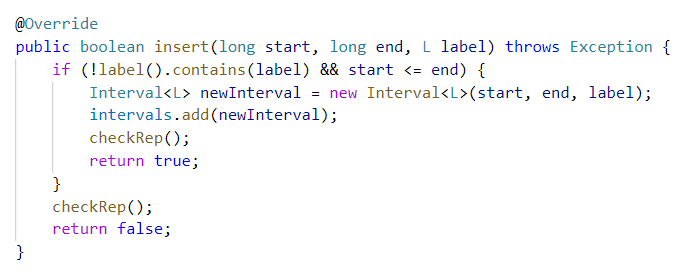
首先有具体的规约、AF、RI等：



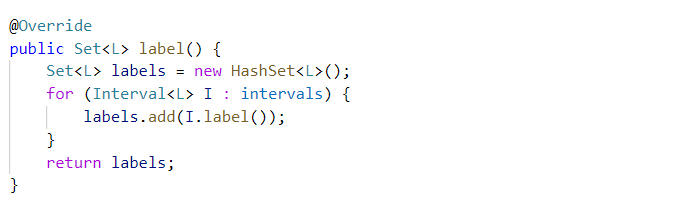
存储结构：采用简单的List<Interval<L>>存储，每次在插入前要检查是否有重复的标签。

### 局部共性特征的设计方案

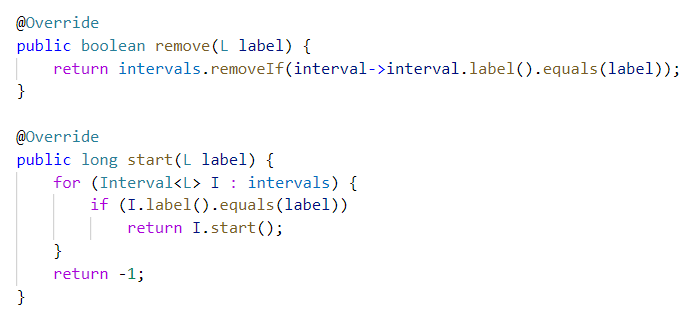
首先，对于方法insert检测不符合的要求的条件，例如开始时间大于结束时间、存储结构中存在时间段与待插入的标签是相等的，若有上述情况就就直接退出返回false；否则就往IntervalSet中插入该标签以及其对映的时间段。



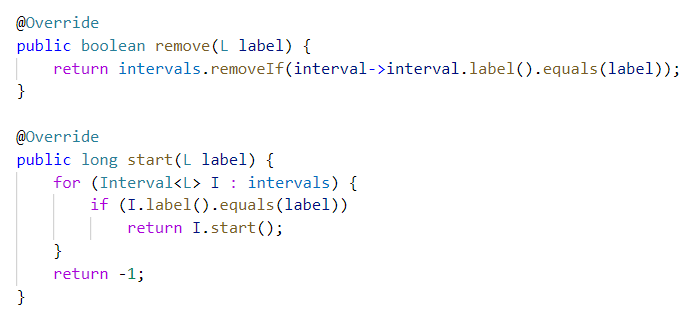
对于获取全部标签的方法，遍历全部时间段，将它们的标签全部插入Set最终返回。

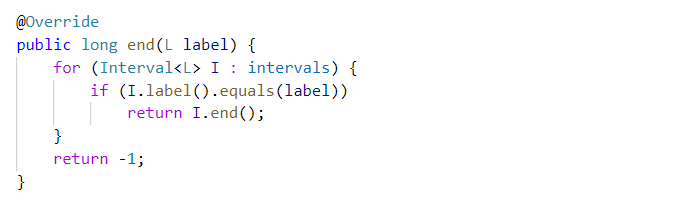


根据标签删除时间段集合中的某个时间段，因为标签唯一存在，所以检测interval List中的全部时间段，若有标签相等则删除并根据删除结果返回true或者false，具体来说就是调用方法removeIf。

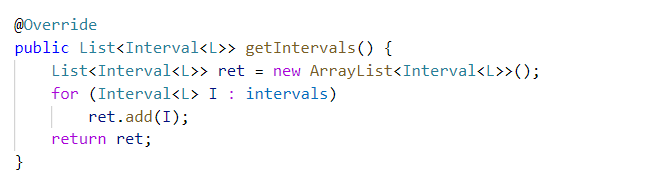


根据标签获取某个时间段的开始时间，若是不存在与该标签对映时间段则返回时间为-1（因为在本实验中，所有时间都是正数，-1为非法时间）。对获取结束时间也是同理。

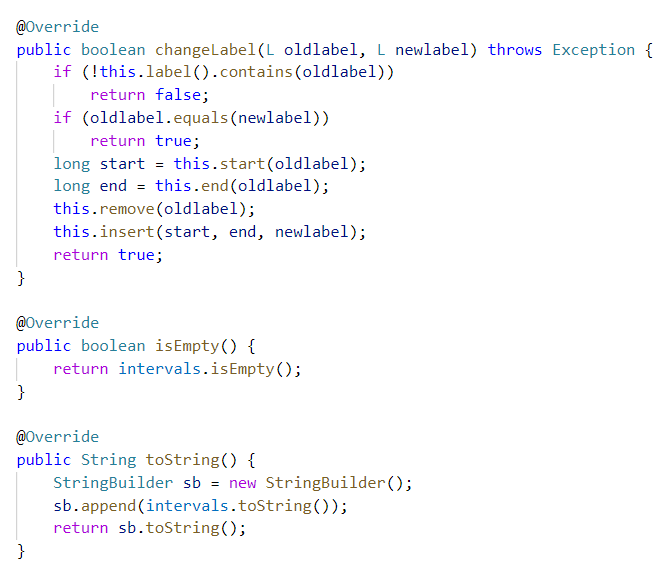




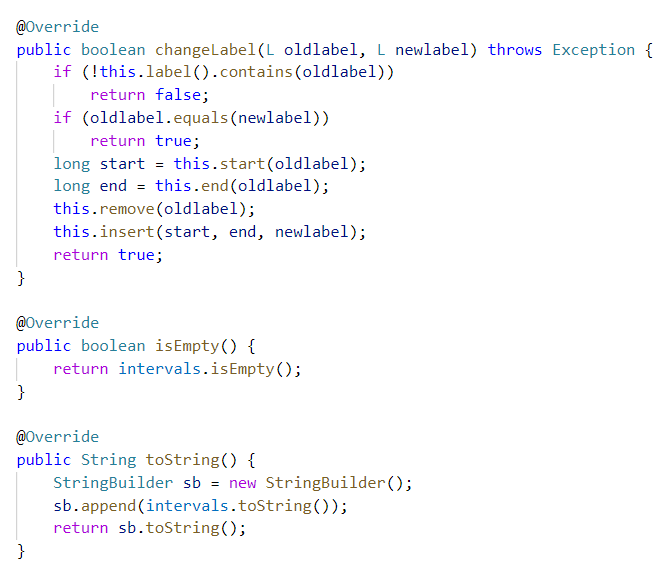
获取全部的时间段及其标签，创建新的关于Interval的列表，将全部时间段都加载进去，然后将该时间列表返回。

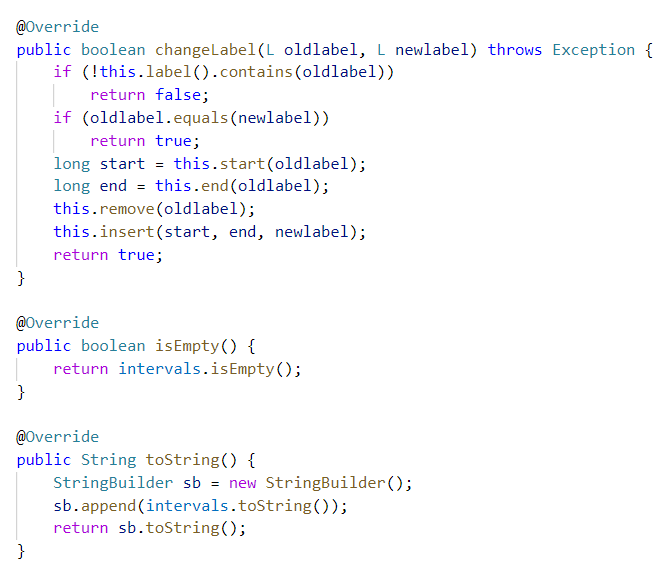


改变某个标签为新标签，首先检测是否存在旧标签，若不存在则返回fasle；否则根据插入情况来返回（若是新标签已经存在还是会返回false）。

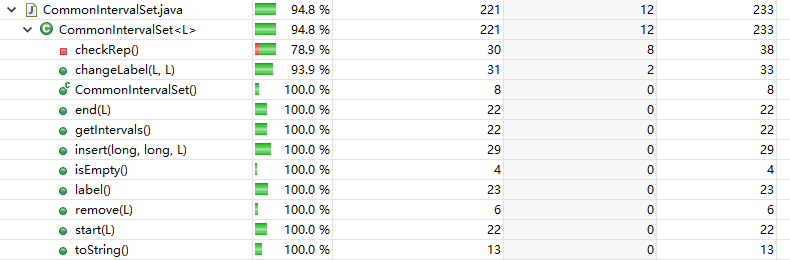


以下方法简单就不再赘述。





最终得到测试结果及其覆盖度如下所示：

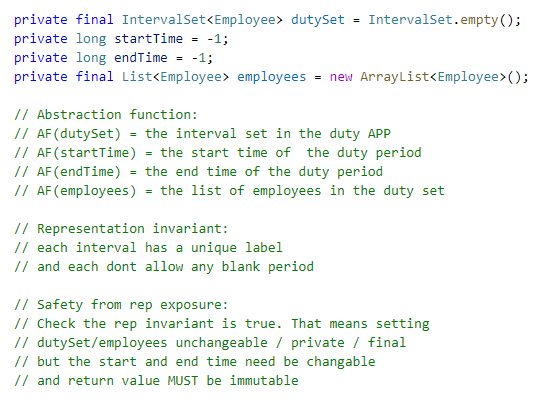


### 面向各应用的IntervalSet子类型设计（个性化特征的设计方案）

易知，例如IntervalSet这种不能插入相同标签的类只适用于排班APP，即在排班系统中使用该类CommonItervalSet。

实现方案：个人采用简单的方案是将各特殊操作分别放入底层的应用子类。将针对不同特征取值的具体操作分别放在三个应用的子类中加以实现，针对本实验来说代码量不大，可以轻松地实现，就算是重复的方法代码，也能轻松修改。

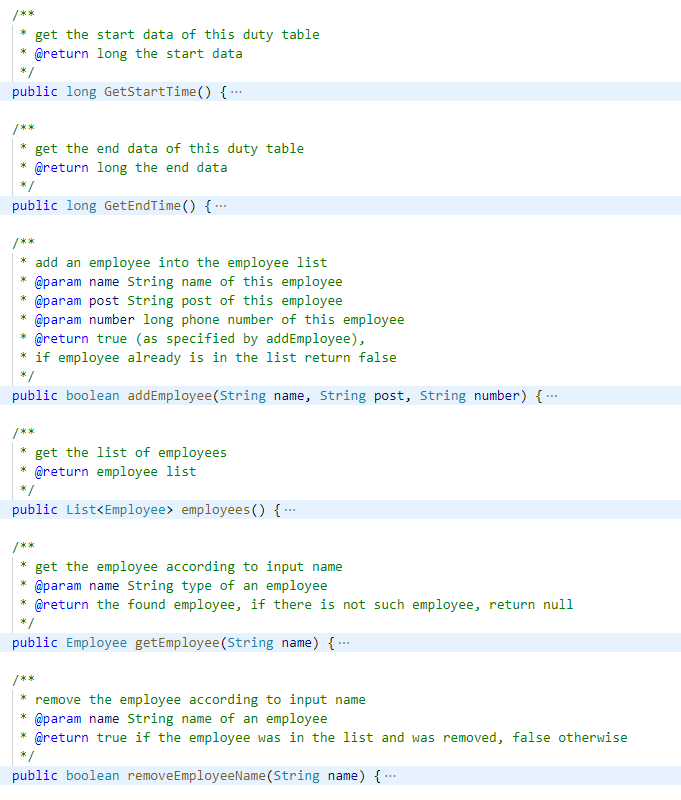
在排班系统中还定义了一系列的个性化方法和存储结构，如下所示。首先是规约等：定义了存储排班表的整体开始和结束时间，以及存储职员的单独列表。

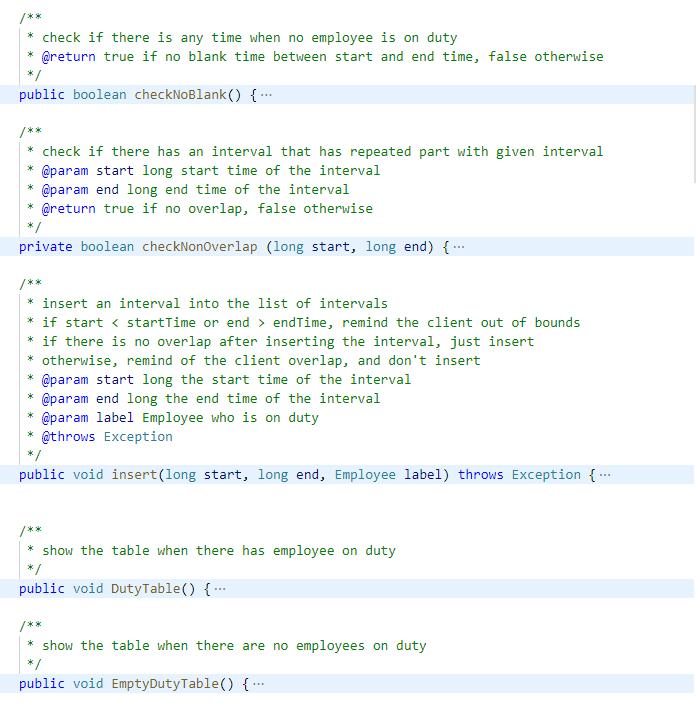


然后是一些自定义方法：

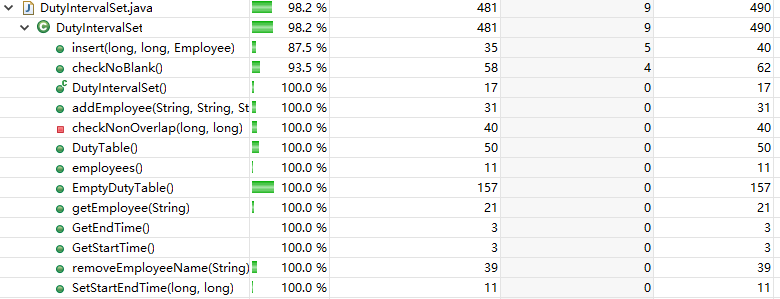
|  |  |
| --- | --- |
| **方法** | **含义** |
| long GetStartTime() | 获取时间表开始时间 |
| long GetEndTime() | 获取时间表结束时间 |
| boolean addEmployee(String name, String post, String number) | 加入新职员 |
| List<Employee> employees() | 获得职员列表 |
| Employee getEmployee(String name) | 根据职员名字得到该职员 |
| boolean removeEmployeeName(String name) | 根据名字将职员移除职员列表（注意此时要要求职员不能在安排表中） |
| boolean checkNoBlank() | 检测不能有空闲时间 |
| boolean checkNonOverlap() | 检查不能有重复时间段 |
| void DutyTable() | 输出值班表 |
| void EmptyDatyTable() | 输出还没有安排的时间表 |

它们的规约、声明等如下所示：





测试结果以及覆盖度：



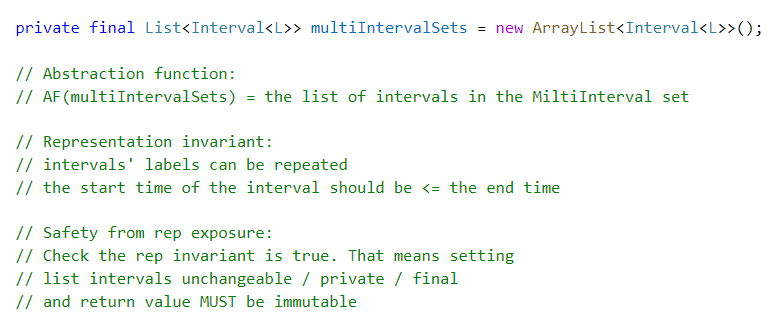
## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

为接口MultiIntervalSet设计应提供的共性接口方法：创建一个空对象、创建一个非空对象、在当前对象中插入新的时间段和标签、获得当前对象中的标签集合、从当前对象中移除某个标签所关联的全部时间段、从当前对象中获取与某个标签所关联的所有时间段、返回全部的时间段安排、判断是否是空表。这些都放入MultiIntervalSet<L>接口封装起来。

|  |  |
| --- | --- |
| **方法** | **含义** |
| empty() | 创建一个空对象 |
| Create(IntervalSet<L>) | 创建一个非空对象 |
| boolean insert(long start, long end, L label) | 插入新的时间段和标签 |
| Set<L> labels() | 标签集合 |
| boolean remove(L label) | 移除某个标签所关联的全部时间段 |
| IntervalSet<Integer> | 获取某个标签所关联的所有时间段 |
| list<Interval<L>> allIntervals() | 获取全部的时间段及标签 |

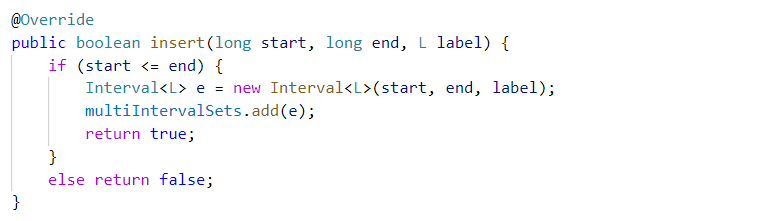
CommonMultiIntervalSet<L>是MultiIntervalSet<L>的具体实现类，实现上述方法。首先有具体的规约、AF、RI等：



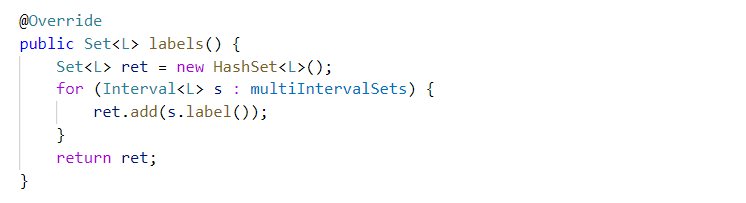
存储结构：采用简单的List<Interval<L>>存储，每次在插入前不用检查是否有重复的标签。

### 局部共性特征的设计方案

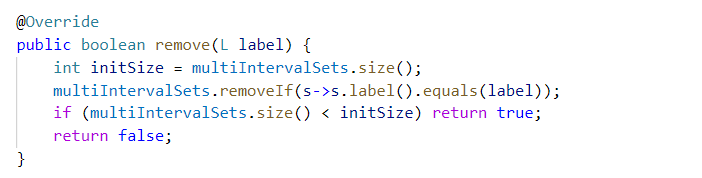
首先，对于方法insert检测不符合的要求的条件，例如开始时间大于结束时间，若有上述情况就就直接退出返回false；否则就往MultiIntervalSet中插入该标签以及其对映的时间段。（在这里不检测标签重复）



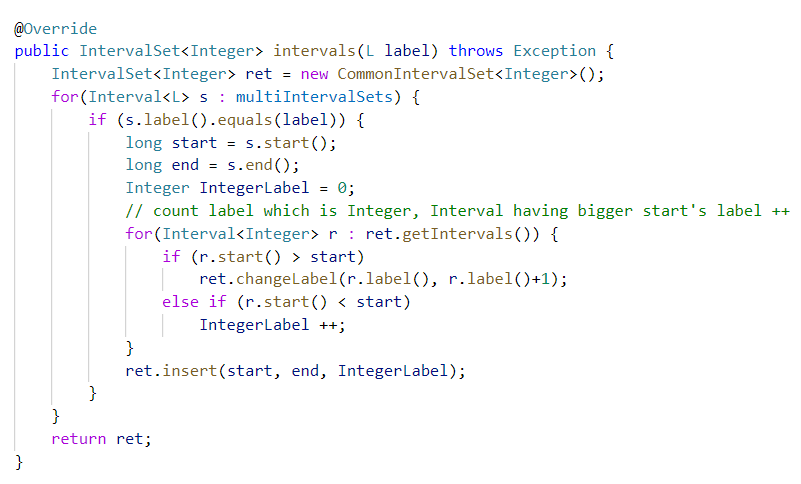
对于获取全部标签的方法，遍历全部时间段，将它们的标签全部插入Set最终返回，在这里利用了Set的插入特性，即重复插入不会出现多个标签。



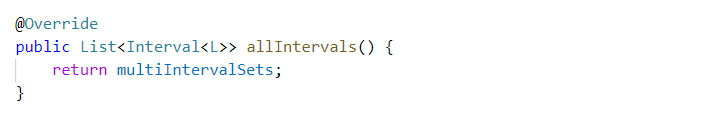
根据标签删除时间段集合中的某个时间段，因为标签唯一存在，所以检测interval List中的全部时间段，若有标签相等则删除并根据删除结果返回true或者false，具体来说就是调用方法removeIf。



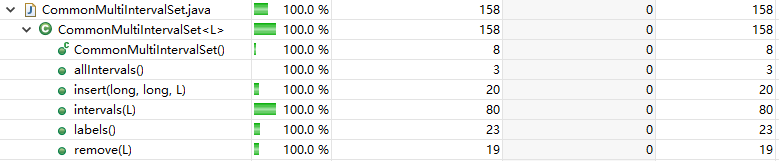
根据输入标签获取全部相关的数据段，并按开始时间进行排序，存储在一个IntervalSet<L>中返回，并且标签L为排序序列，所以这里具体使用的是Integer。具体来说，首先遍历时间段序列查询与标签相关的时间段，将其插入构建的IntervalSet中，在插入的过程中进行比较开始时间，若第一个插入为0，之后的和IntervalSet中每一段时间的开始时间比较，谁的开始时间大谁的序列就加一，比较完成之后就可以得到相应的序列，则完成排序。



将全部的时间段及其序列添加到创建的List中，然后返回。



最后是测试结果以及覆盖度：

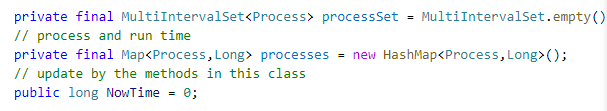


### 面向各应用的MultiIntervalSet子类型设计（个性化特征的设计方案）

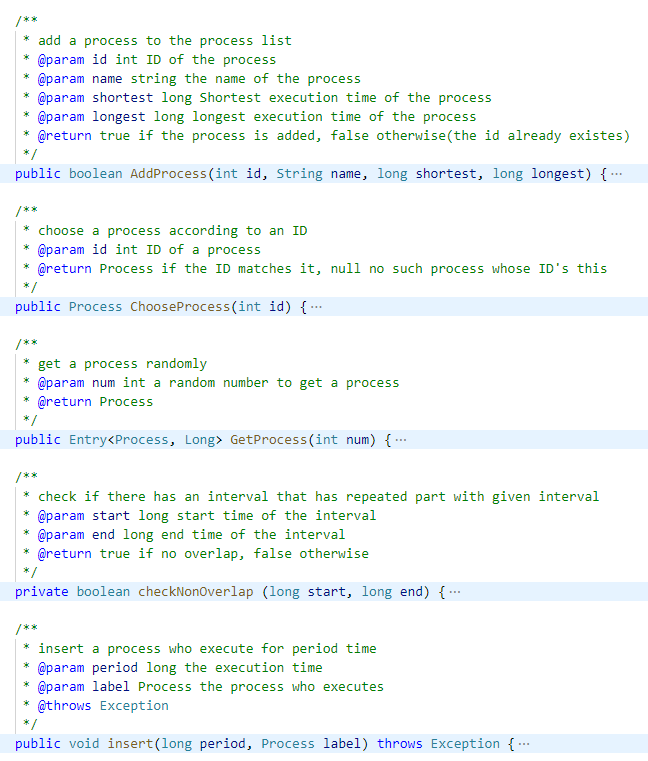
#### 面向Process进程安排的应用设计

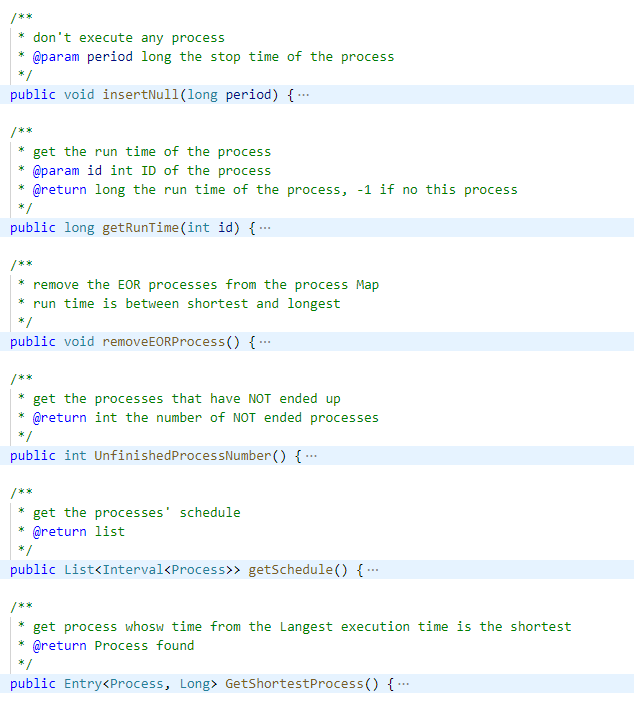
实现方案：个人采用简单的方案是将各特殊操作分别放入底层的应用子类。将针对不同特征取值的具体操作分别放在三个应用的子类中加以实现，针对本实验来说代码量不大，可以轻松地实现，就算是重复的方法代码，也能轻松修改。

在进程系统中还定义了一系列的个性化方法和存储结构，如下所示。首先是规约等：定义了当前时间，以及存储每个进程已运行时间（若运行结束，移除出该HashMap）。

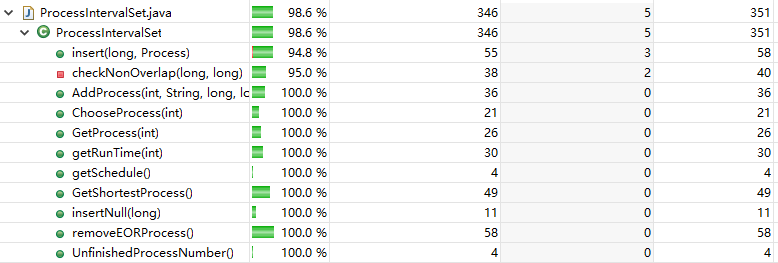


它们的规约以及声明等：



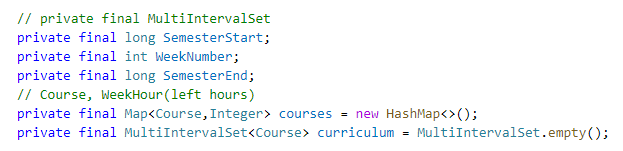


测试结果以及覆盖度：

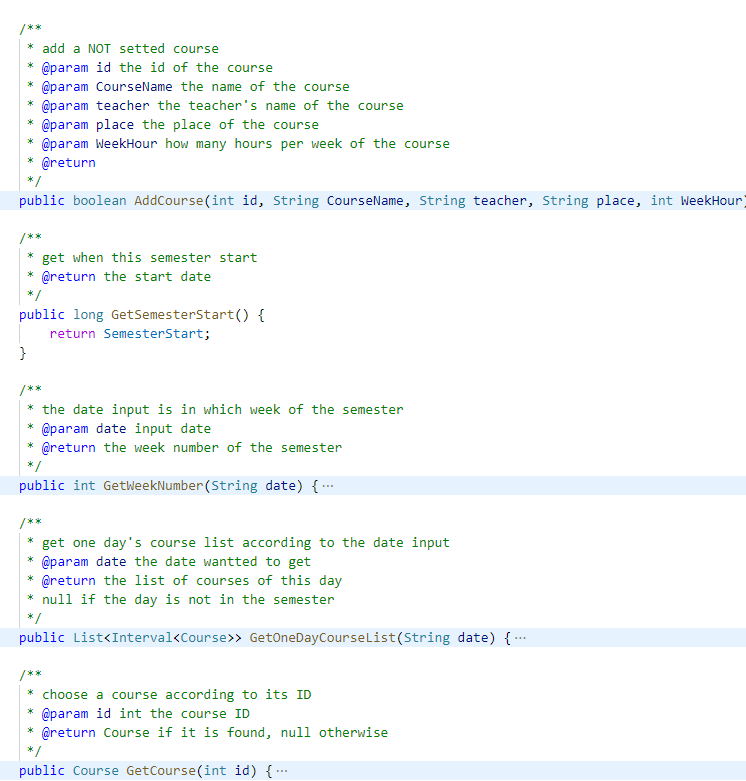


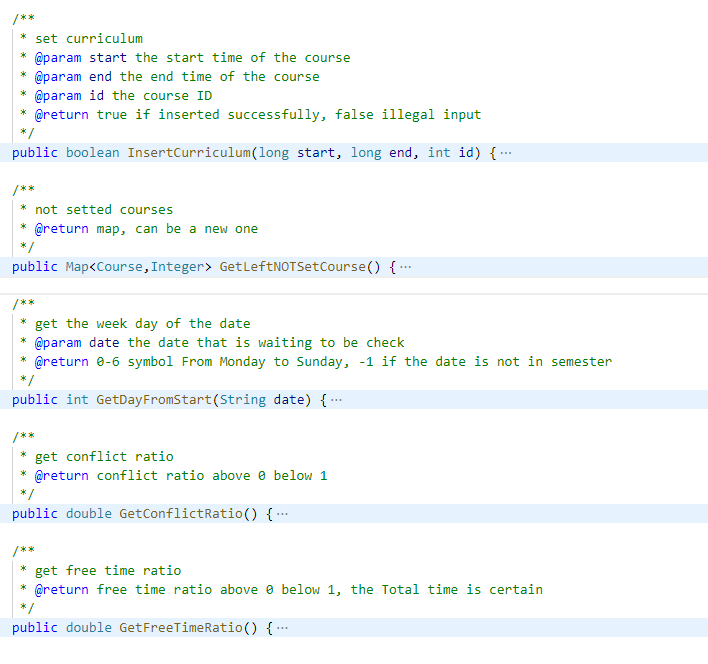
#### 面向Course课表安排的应用设计

在课表系统中还定义了一系列的个性化方法和存储结构，如下所示。首先是规约等：定义了学期开始时间、学期结束时间、学期周数，以及存储每门课程还没有安排的时间（若安排结束value为0，移除出该HashMap）。

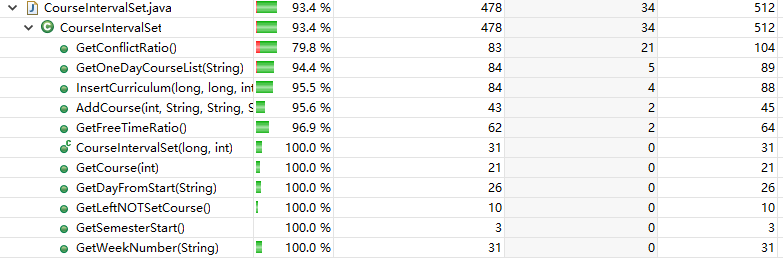


自定义方法如下规约和声明（注意这里的两个比例计算方法与下述APIs设计是不同的，所以重新书写）：





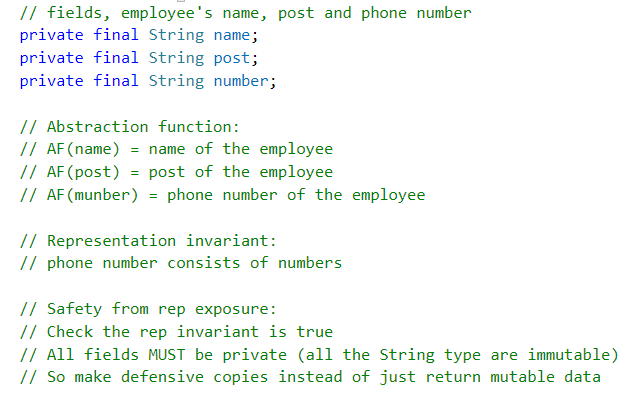
测试结果及覆盖度：



## 面向复用的设计：L

### 员工（Employee）

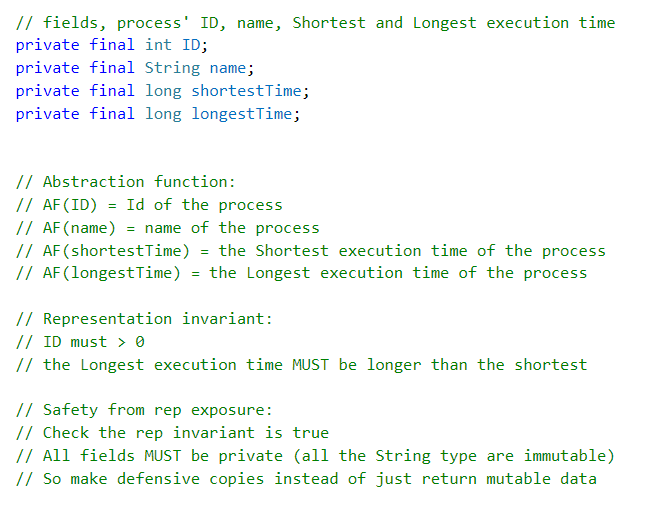
在排班系统中关心的属性应该为“员工“，包括员工的姓名、职务、手机号码，用一个Employee类来将这些都包括在里面。如下是这个类的规约、存储和方法及测试：

、

|  |  |
| --- | --- |
| **方法/Constructor** | **含义** |
| Employee(String name, String post, String number) | 构建一个职员，增加其信息 |
| String GetName() | 返回职员姓名 |
| String GetPost() | 返回职员职位 |
| String GetNumber() | 返回职员电话号码 |
| boolean equals(Object o) | （重写）比较是否是同一人，以名字作判断条件 |
| String toString() | （重写）将职员信息转为String型返回 |

### 进程（Process）

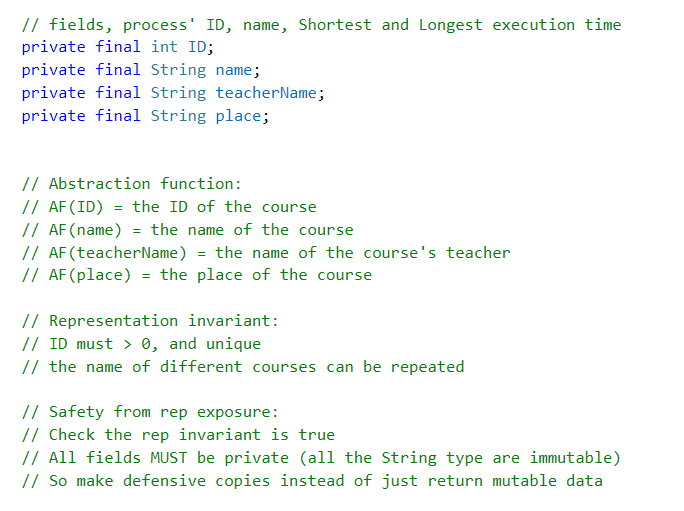
在进程安排系统中关心的属性应该为“进程“，包括进程 ID、进程名称、最短执行时间、最长执行时间，用一个Process类来将这些都包括在里面。如下是这个类的规约、存储和方法及测试：



|  |  |
| --- | --- |
| **方法/Constructor** | **含义** |
| Process(int ID, String name, long shortest, long longest) | 构建一个进程，增加其信息 |
| Int getID() | 返回进程唯一ID |
| String GetName() | 返回进程姓名 |
| long GetShortestTime() | 返回进程最短运行时间 |
| long GetLongestTime() | 返回进程最长运行时间 |
| boolean equals(Object o) | （重写）比较是否是同一进程，以ID作判断条件 |
| String toString() | （重写）将进程信息转为String型返回 |

### 课程（Course）

在课程安排系统中关心的属性应该为“课程“，包括课程 ID、课程名称、教师名字、上课地点，用一个Course类来将这些都包括在里面。如下是这个类的规约、存储和方法及测试：



|  |  |
| --- | --- |
| **方法/Constructor** | **含义** |
| Process(int ID, String name, String teacher, String place) | 构建一门课程，增加其信息 |
| Int getID() | 返回课程唯一ID |
| String GetName() | 返回课程姓名 |
| String GetTeacherName() | 返回课程教师名字 |
| long GetPlace() | 返回课程上课地点 |
| boolean equals(Object o) | （重写）比较是否是同一课程，以ID作判断条件 |
| String toString() | （重写）将课程信息转为String型返回 |

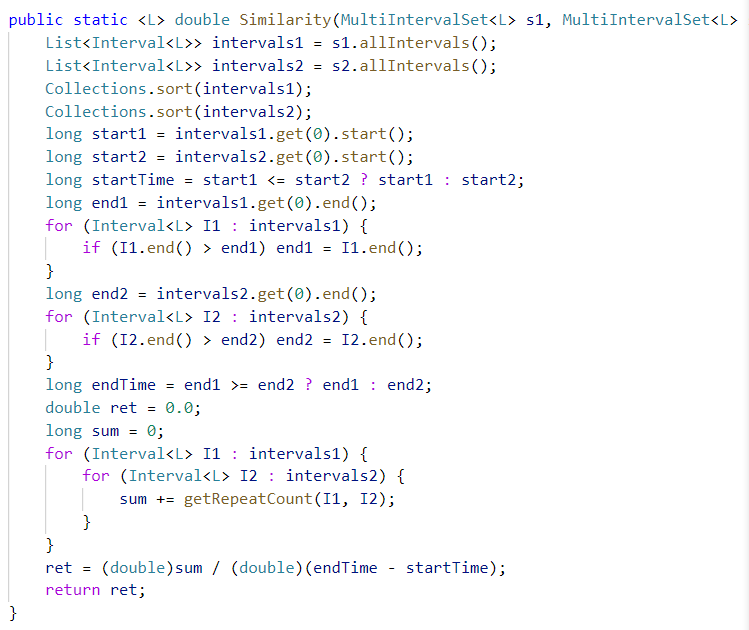
## 可复用API设计

### 计算相似度

要计算相似度，首先定义实现了两个方法，主方法为double Similarity(MultiIntervalSet<L> s1, MultiIntervalSet<L> s2)，辅助方法有long getRepeatedCount(Interval<L> i1, Interval<L> i2)。前者是计算相似度的主要过程，后者则是就算某两个时间段的重复区间长度。

首先来看getRepeatedCount方法，两个时间段若是标签不相等则认为它们没有重复区间，否则比较它们的开始时间和结束时间，若是某一方的开始时间大于另一方的结束时间，则没有重复段，否则用公式(Max(end1, end2) – Min(start1, start2)) – abs(start1 – start2) – abs(end1 – end2)来计算重复时间段。

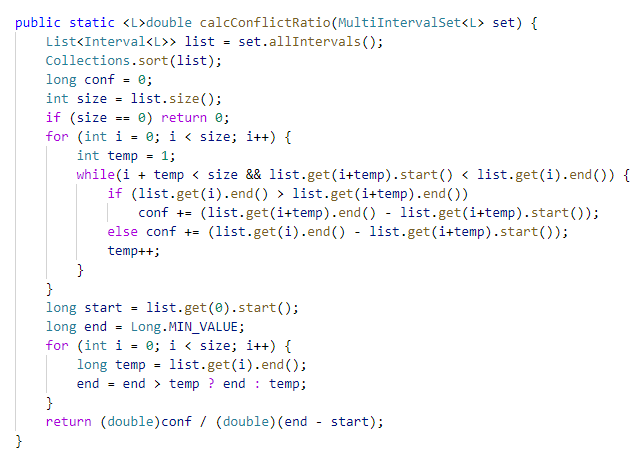
接着来分析主方法Similarity，对于两个MultiIntervalSet<L>类，可以获得它们的全部时间段，接着，分别获得它们整体的起始时间和结束时间，用每一个s1中的Interval去与每一个s2中的Interval作比较获得重复时间段。累加除以整体的时间段就可以获得相似度。



### 计算时间冲突比例

计算时间冲突比例，这里定义了两个方法：double calcConflictRatio(IntervalSet<L> set)和double calcConflictRatio(MultiIntervalSet<L> set)，但是它们的思想与实现几乎都是一样的，所以这里就只解释一个方法即可。

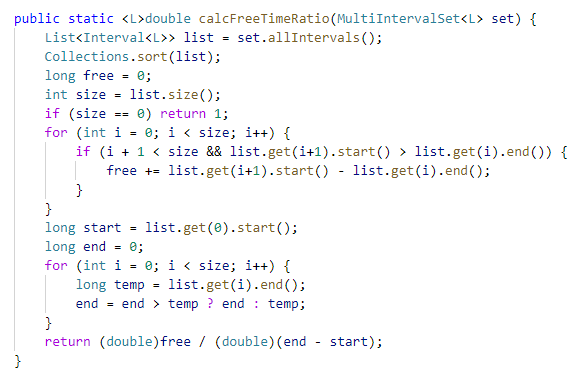
对于double calcConflictRatio(MultiIntervalSet<L> set)首先获取MultiIntervalSet<L>的整体开始时间与结束时间，按照时间段开始时间从小到大排序之后用前面的时间段与后面的时间段进行重复计算，累加之后除以整体的结束时间减去开始时间的差，就可以得到时间冲突比例。



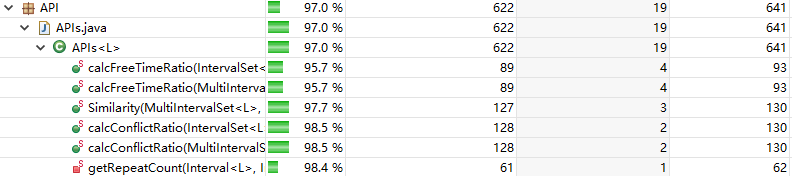
### 计算空闲时间比例

与冲突时间比例一样，这里定义了两种计算空闲时间比例的方法：double calcFreeTimeRatio(IntervalSet<L> set) double calcFreeTimeRatio(MultiIntervalSet<L> set)。

对于double calcFreeTimeRatio(MultiIntervalSet<L> set)解析：只需要在开始时间从小到大排序之后计算连续时间段之间的空闲部分，累加除以整体的结束时间减去开始时间的差，就可以得到时间冲突比例。



测试结果及覆盖度如下所示：



## 应用设计与开发

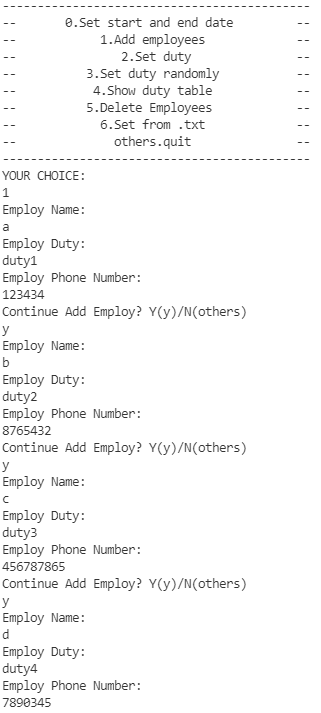
利用上述设计和实现的ADT，实现手册里要求的各项功能。

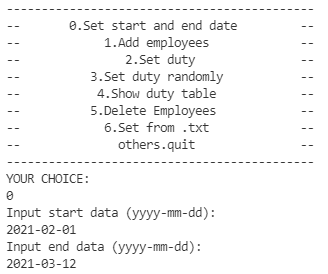
### 排班管理系统

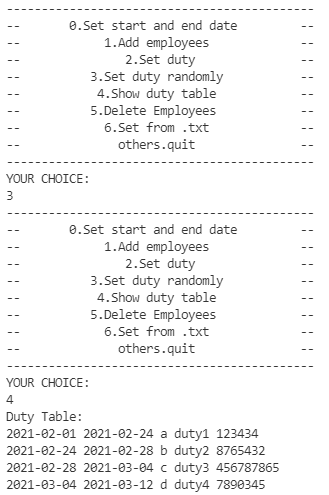
按照步骤，每个功能设计一个方法，其实就是每个步骤调用的DutyIntervaSet方法的组合。注意这里没有进行健壮性测试，如果实现是手动那么需要先进行一二步即选择0和1，否则就是从文件中全自动读入。

1. 设定排班开始日期、结束日期 (yyyy-MM-dd)。
2. 增加一组员工 (名字、职务、手机号码)，可删除未被编排进排班表的员工，从排班信息删掉之后才能删除该员工。员工信息一旦设定则无法修改。
3. 可手工选择某个员工、某个时间段，向排班表增加一条排班记录，该步骤可重复执行多次。随时展示给用户排班表中哪些时间段未安排、未安排的时间段占总时间段的比例。
4. 可自动编排的方法，随机生成排班表。
5. 可视化展示任意时刻的排班表。

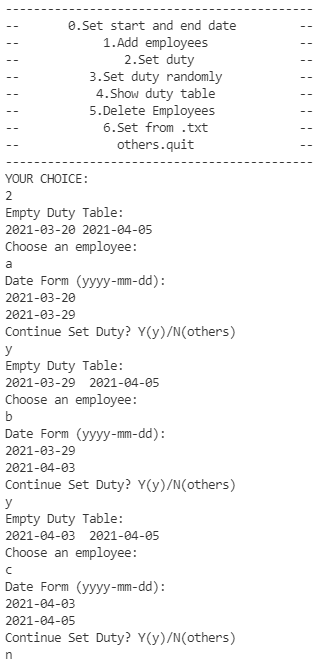
**试运行结果如下：**

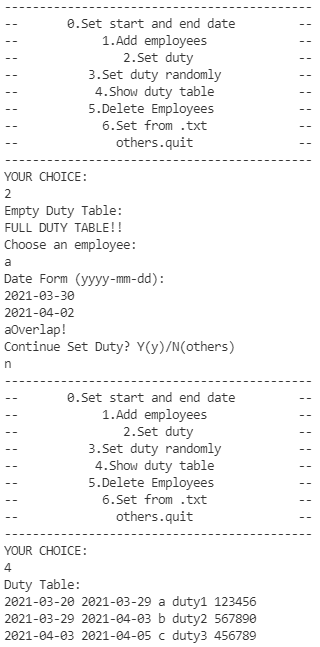


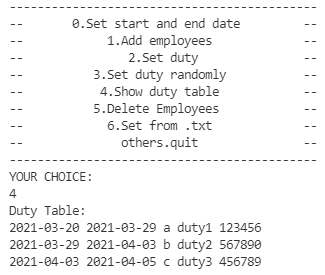




**如上所示，是自动排班功能，只需要设置起始时间和结束时间以及设置员工，再选择自动排班就可以得到如图所示结果，该结果是符合要求地一个排班。下面有一个手动排班过程及其结果，还有一些异常处理：**





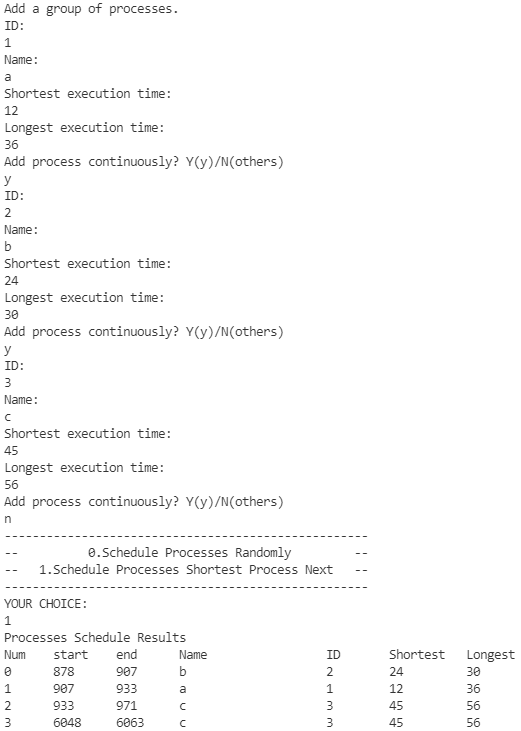


### 操作系统的进程调度管理系统

1. 增加一组进程（ID、名称、最短执行时间、最长执行时间）。
2. “随机选择进程”的模拟策略：从0开始启动模拟调度，随机地选择某个（小于等于一个）尚未执行结束进程执行，并在该进程最大时间之前的任意时刻停止执行，如果本次及其之前的累积执行时间已落到最短执行时间，最长执行时间的区间内，则该进程被设定为“执行结束”。重复直到所有进程结束。
3. “最短进程优先”的模拟策略：每次选择进程的时候，优先选择距离其最大执行时间差距最小的进程。
4. 可视化展示当前时刻之前的进程调度结果。

**试运行结果如下，有两种自动实现方法（完全随机和最短优先）：**

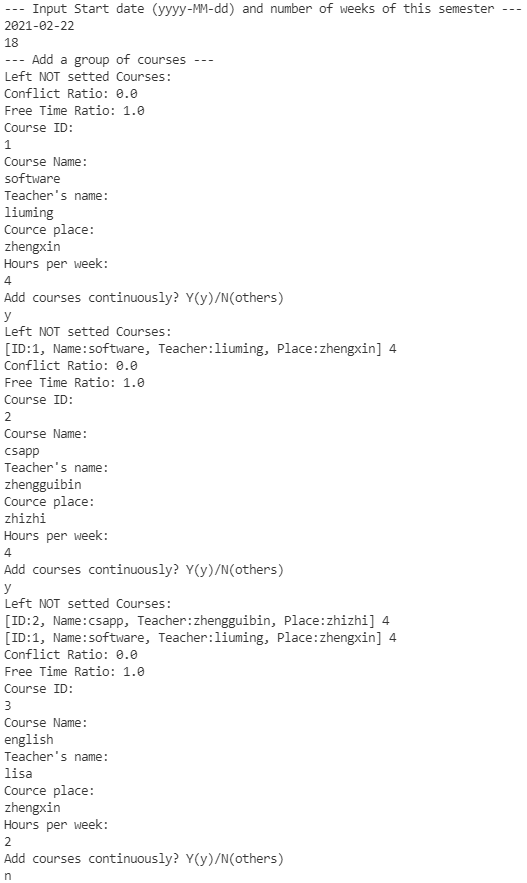


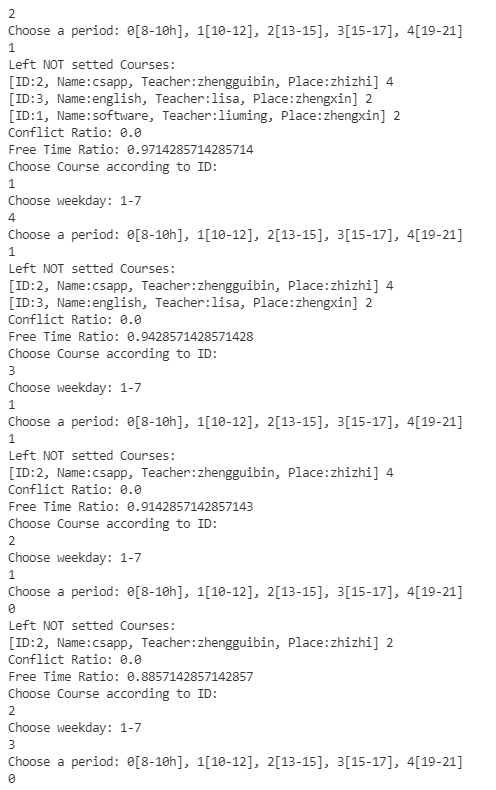


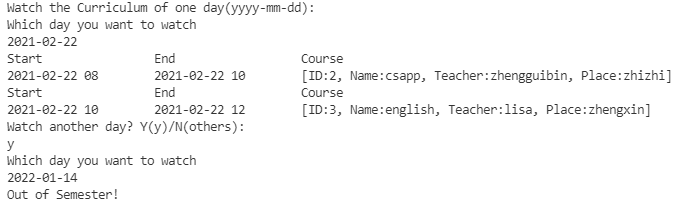
### 课表管理系统

1. 设定学期开始日期（年月日）和总周数。
2. 增加一组课程（课程 ID、课程名称、教师名字、地点、周学时数（偶数））。
3. 手工选择某个课程、上课时间（8-10 时、10-12 时、13-15 时、 15-17 时、19-21 时），为其安排一次课，每次课的时间长度为2小时；可重复安排，直到达到周学时数目时该课程不能再安排。
4. 随时可查看哪些课程没安排、当前每周的空闲时间比例、重复时间比例。
5. 可查看本学期内任意一天的课表结果。

**试运行结果如下：**



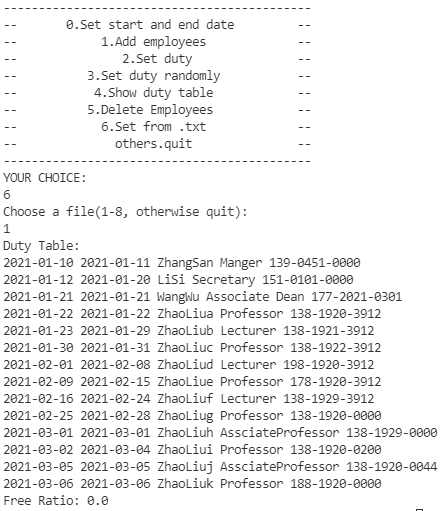


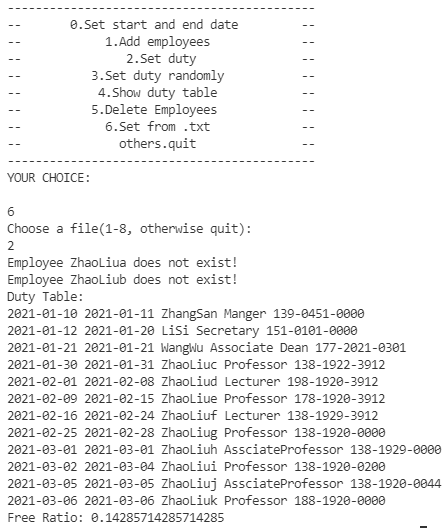


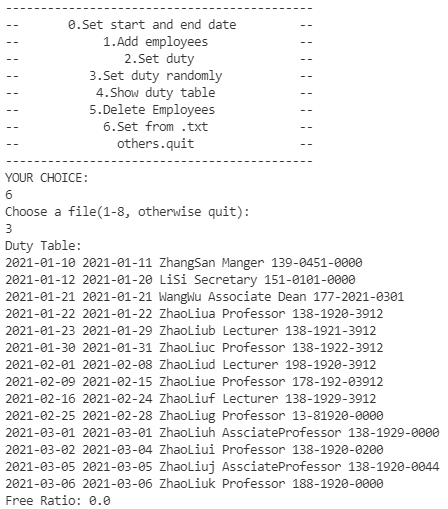
## 基于语法的数据读入

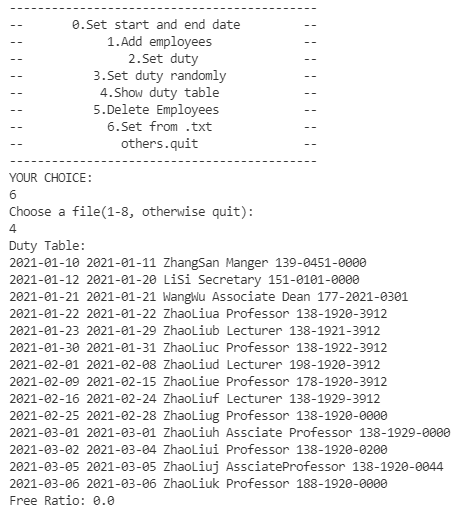
修改“排班管理”应用以扩展该功能。

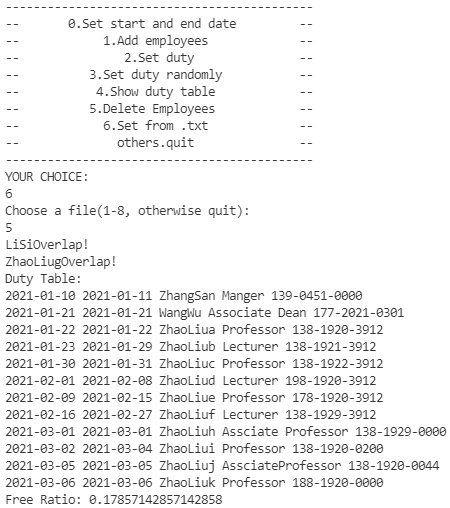
1. 首先，每个文件中前面的空格不能识别，所以先将其删除；
2. 定义多个正则表达式，用来逐行识别内容，若是与Employee匹配上则开始添加职员，此时需要注意职员名字只能由字母组成，电话号码改成3-4-4、2-5-4等多个形式都成立；若是与Period匹配上则设置起始结束时间；在这两个之后与Roster匹配上则开始设置值班表，每个职员都要到上述的职员名单中找，若是找不到则直接跳过。

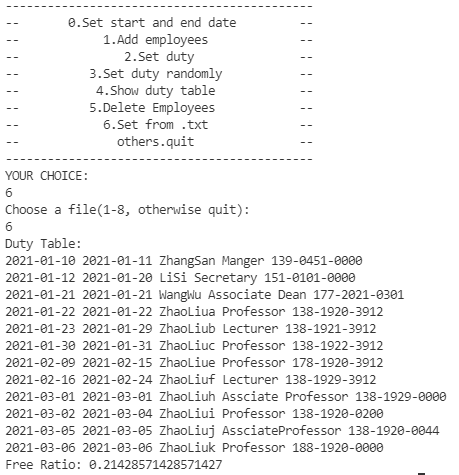


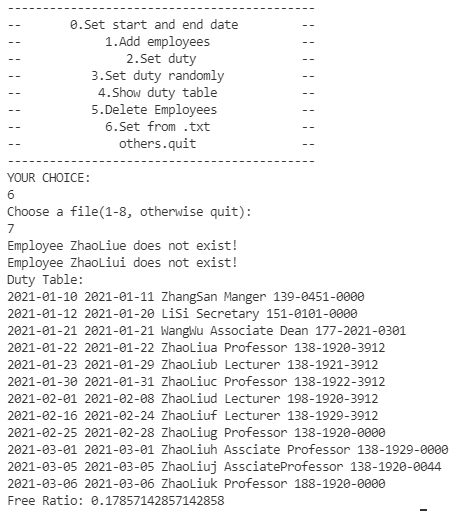


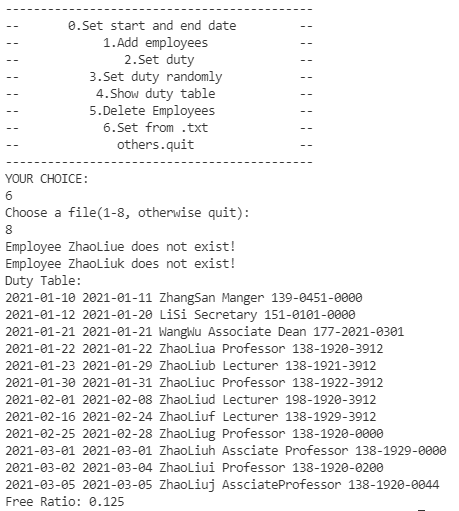












## 应对面临的新变化

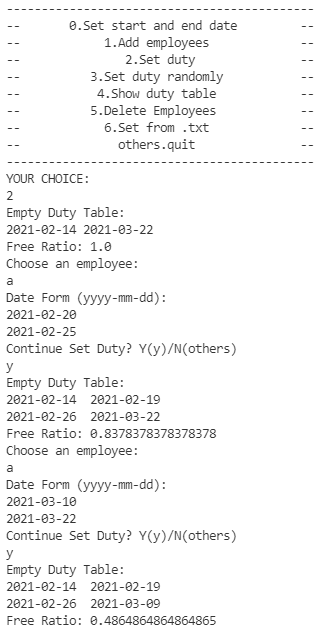
### 变化1

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)。但是此时任然不允许有任何重复段的出现，也不允许出现空白。

如何修改设计以应对变化：首先是，将IntervalSet改为MultiIntervalSet，这样就可以应对插入重复的label情况。接下来就是将功能相同但是命名不同方法改正确，一个是label改为labels，另一个是getIntervals改为allIntervals。

评估之前的设计是否可应对变化、代价如何：修改的代码量就只有几行，很容易应对该改变，如果在设计IntervalSet和MultiIntervalSet的时候将相同功能的方法命名为一样，那么就几乎只用修改一行代码。

如下是在插入相同段的显示：



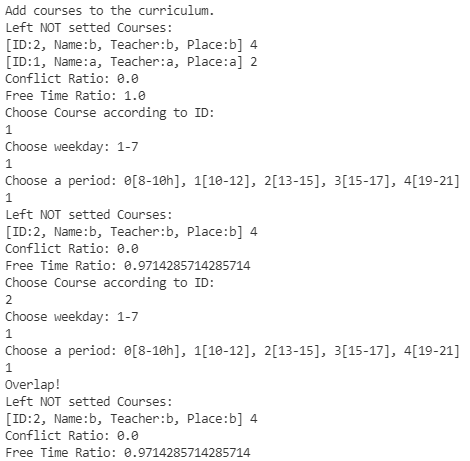
### 变化2

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”）。那么在原过程中需要调用checkNonOverlap来检查重复段，若有重复则做出相应应答，可以报错，并且不再插入该段课表。

评估之前的设计是否可应对变化、代价如何：代价一般高，只插入了二十行左右的代码，并且修改时间不长。

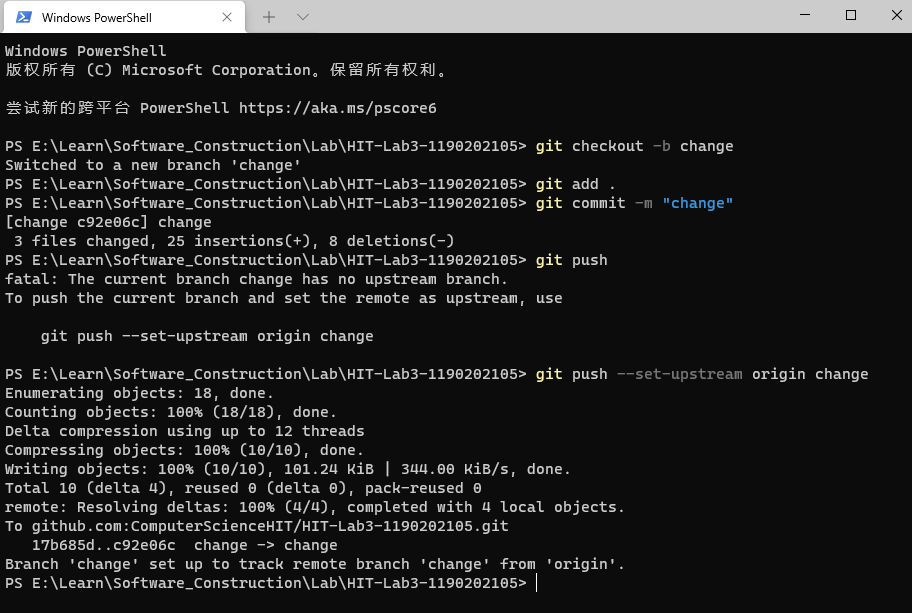
如何修改设计以应对变化：加入checkNonOverlap方法，并检测重复段。

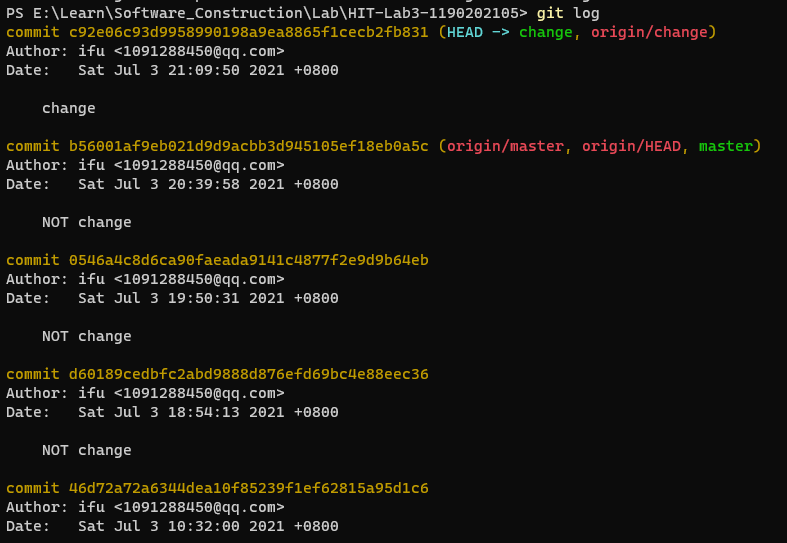
如下例所示：在插入重复段的时候会提示错误并且不会将重复段插入。

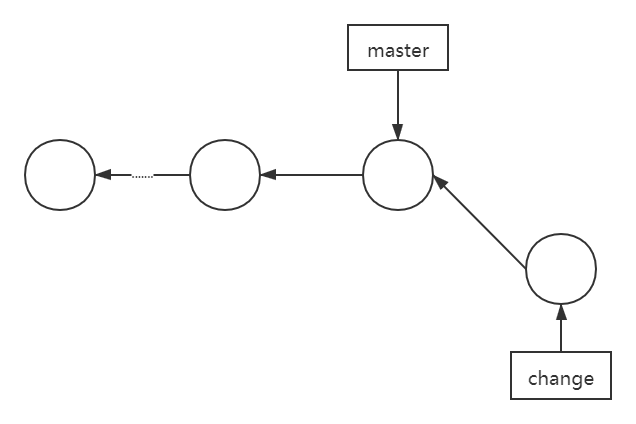


## Git仓库结构

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 |
| 2021.06.17 | 14:00-16:00 | 阅读实验要求，完成环境搭建等 | 完成 |
| 2021.06.24 | 14:00-16:00 | 根据实验要求，提取各类要点，编写IntervalSet的接口和实现代码 | 未完成 |
| 2021.06.27 | 19:00-23:00 | 提取IntervalSet共性方法、编写测试以及实现 | 未完成 |
| 2021.06.28 | 全天 | IntervalSet方法及实现，提取共性方法MutilIntervalSet，设计测试和编写方法 | 未完成 |
| 2021.06.29 | 全天 | 完成以上没有完成的内容，设计关于排班系统的APP，面向复用的设计Employee | 未完成，在实际中对方案五的应用出现疑惑，改变方案 |
| 2021.06.30 | 全天 | 面向复用的设计Process设计关于Process进程安排的APP | 完成 |
| 2021.07.01 | 全天 | 面向复用的设计Course，设计关于课表系统的APP | 未完成 |
| 2021.07.02 | 全天 | 设计关于课表系统的APP，基于语法的数据读入 | 未完成 |
| 2021.07.03 | 全天 | 课程系统设计，改变排班和课表应用并提交到change分支 | 完成 |
| 2021.07.04 | 8:00-12:00 | 撰写剩余报告，修改部分实现内容，完善细节等 | 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、继承和委托的使用不熟悉。 | 在考虑之后改变方法。 |
| 对于三个APP设计的方法不知到从何下手 | 面向具体应用编程，要使用某个功能的时候再根据这来设计具体方法。 |
| 不清楚正则表达式的使用 | 询问同学，根据PPT中有限的内容从网络上查找到相关方法和使用格式等。 |
| 各类之间调用关系混乱 | 仔细分辨内容 |

# 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在五个不同的应用场景下使用，你是否体会到复用的好处？

在面向ADT编程的时候需要考虑代码的可复用性，需要使整个编程更加具有普适性；面向应用场景可以直接根据要求来编写程序，更易于操作，但是程序复用性很低，难以在其他场景进行复用，或者是需要修改的内容增多。在不同的场景下的确能带来简化代码的功能，一定程度上体会了面向ADY编程带来的好处，但是由于三个应用场景中Course的应用还存在更为简单的操作，例如数组等，所以在应用过程中还增加了编程的复杂度。

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？

根据规约、不变量、RI、AF来编写程序，能有效避免在代码中插入错误，再检查错误的过程，节约时间；同时能够尽早找到程序中的错误，避免错误累积到后续难以修改。在以后的编程中可以尽量这样做，但是对于简单逻辑的程序其实意义不是很大。

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？

还好，在本题中所开发的API很难被复用，因为按题意开发的API在实际应用中很难处理自定义的边界。例如，开发的calcFreeTimeRatio和calcConflictRatio在实际情况Course的应用中，是不应该自己去计算整体时间，而是应该定义一个全部的具体时间，例如7\*5\*2=70个小时。

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？

语法驱动编程一定程度上固定了输入格式，例如本次实验中根据语法和正则表达式解析输入文件，能根据高效提取得到我们所期待的内容，还较为清晰和易解。同时这还是一种隐秘的编程方式，能够提高隐私性。

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？

设计问题主要两点，一是提取共同点，要将不同开发内容的共同点提取出来；二就是方法设计和调用很茫然，在实现过程中，很多方法的实现都是根据APP设计所要使用的内容来添加的，以及调用关系会随着代码量的增加而变得非常混乱。

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？

在本实验中，利用接口、抽象类、类的组合继承等，有效地减少了代码量，避免了重复编程，但是在复用时，需要理清各个关系。

1. 关于本实验的工作量、难度、deadline。

本实验工作量巨大，相对于实验二来说难度提高了很多，deadline与期末考试时间冲突，导致实际上能花在实验中的时间很短。

1. 到目前为止你对《软件构造》课程的评价。

能提高能力学到一些编程思想等，但是时间长度和内容多少匹配不上，太赶了。