

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 熊峰 |
| 学号 | 1190200708 |
| 班号 | 1903008 |
| 电子邮件 | xiong257246@outlook.com |
| 手机号码 | 13114991114 |

**目录**

[1 实验目标概述 1](#_Toc76310236)

[2 实验环境配置 1](#_Toc76310237)

[3 实验过程 2](#_Toc76310238)

[3.1 待开发的三个应用场景 2](#_Toc76310239)

[3.1.1 排班管理系统 2](#_Toc76310240)

[3.1.2 操作系统进程调度管理（ProcessSchedule） 3](#_Toc76310241)

[3.1.3 大学课表管理（CourseSchedule） 3](#_Toc76310242)

[3.1.4 三个应用场景分析 4](#_Toc76310243)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 4](#_Toc76310244)

[3.2.1 IntervalSet<L>的共性操作 4](#_Toc76310245)

[3.2.2 局部共性特征的设计方案 6](#_Toc76310246)

[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） 8](#_Toc76310247)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 9](#_Toc76310248)

[3.3.1 MultiIntervalSet<L>的共性操作 9](#_Toc76310249)

[3.3.2 局部共性特征的设计方案 10](#_Toc76310250)

[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） 13](#_Toc76310251)

[3.4 面向复用的设计：L 15](#_Toc76310252)

[3.4.1 概述 15](#_Toc76310253)

[3.4.2 员工Employee 16](#_Toc76310254)

[3.4.3 进程Process 19](#_Toc76310255)

[3.4.4 课程Course 21](#_Toc76310256)

[3.5 可复用API设计 24](#_Toc76310257)

[3.5.1 计算相似度 24](#_Toc76310258)

[3.5.2 计算时间冲突比例 25](#_Toc76310259)

[3.5.3 计算空闲时间比例 26](#_Toc76310260)

[3.6 应用设计与开发 26](#_Toc76310261)

[3.6.1 排班管理系统 26](#_Toc76310262)

[3.6.2 操作系统的进程调度管理系统 31](#_Toc76310263)

[3.6.3 课表管理系统 35](#_Toc76310264)

[3.7 基于语法的数据读入 38](#_Toc76310265)

[3.8 应对面临的新变化 41](#_Toc76310266)

[3.8.1 变化1 41](#_Toc76310267)

[3.8.2 变化2 43](#_Toc76310268)

[3.9 Git仓库结构 44](#_Toc76310269)

[4 实验进度记录 46](#_Toc76310270)

[5 实验过程中遇到的困难与解决途径 46](#_Toc76310271)

[6 实验过程中收获的经验、教训、感想 46](#_Toc76310272)

[6.1 实验过程中收获的经验和教训 46](#_Toc76310273)

[6.2 针对以下方面的感受 46](#_Toc76310274)

# 实验目标概述

本次实验覆盖课程第前两次课的内容，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：

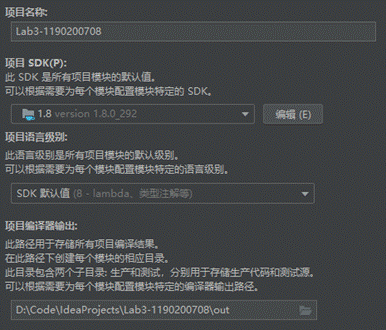
* 子类型、泛型、多态、重写、重载
* 继承、代理、组合
* 常见的OO设计模式
* 语法驱动的编程、正则表达式
* 基于状态的编程
* API设计、API复用

本次实验给定了三个具体应用（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

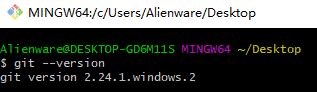
环境配置：

IntelliJ IDEA 2020.3.1  (JDK 1.8  Junit 4.12(下载到lib目录中))



       Git:

              Git version 2.24.1.windows.2



GtiHub Lab3 URL： <https://github.com/ComputerScienceHIT/HIT-Lab3-1190200708.git>

# 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。

场景1：排班管理系统

### 排班管理系统

实际场景应用：

一个单位有n个员工，在某个时间段内，每天只能安排唯一1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。

实际功能需求：

设置值班表信息、添加员工信息、删除员工信息、显示所有员工信息、分配员工值班、显示当前空闲时间、检验当前值班信息是否符合要求、保存值班信息并导出、自动随机编排值班表、从外部文件导入值班信息。

抽象数据类型：

首先设定排班开始日期、结束日期，需要具体到年月日，此时可以将排班管理系统的开始日期、结束日期，抽象为两个长整形数据，在添加员工时，可以用字符串类型的数据，保存员工的姓名、职位、手机号等，并用Interval类型，保存每个员工的值班时间。

实际值班管理系统由Employee到IntervalSet的映射，构成了实际值班管理系统的分配模式。

### 操作系统进程调度管理（ProcessSchedule）

实际场景应用：

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，由操作系统来调度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。

实际功能需求：

增加一组进程，并设置进程相关信息、启动模拟调度(随机选择进程)、启动模拟调度(最短进程优先)。

抽象数据类型：

在添加进程时，需要用字符串类型保存进程的名称、使用长整型保存pid、最短执行时间、最长执行时间。用MultiIntervalSet类型保存一个进程所对应的多段时间。

实际操作系统进程调度管理系统是由process到MultiIntervalSet的映射。

### 大学课表管理（CourseSchedule）

实际场景应用：

每一上午8:00-10:00和每周三上午8:00-10:00在正心楼13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“软件构造课”）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1-2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；

实际功能需求：

设定学期相关信息、增加一组课程，及其相关信息、安排课时、当前所有课程的安排情况、每周空闲时间比例、每周重复时间比例、查看一天课表。

抽象数据类型：

课程由字符串类型保存课程的ID、课程的名称、教师的名称、授课地点以及使用整型保存每周的学时。由于，每门课程可能在一周内多次授课，因此课程对应多个时间段。大学课表管理实际是由课程到MultiIntervalSet的映射。

### 三个应用场景分析

值班管理系统中的时间轴不能有空白，因为实际要求中，每天都要有人值班，因此需要检查是否有空白，并且由于每天只有一人值班，因此不存在重叠的时间段，同时也不具有周期性，值班管理系统，由于每名员工只需要值班一段时间，因此每名员工只对应一个时间段；操作系统进程管理系统由于存在某时可以不执行进程，因此时间轴可以存在空白，但CPU一次只能执行一个程序，因此不可以存在时间段的重叠，同时也不具有周期性，操作系统进程管理由于一个进程可能在多个时间段运行，因此每个进程对应多个时间段；大学课表管理系统时间可以存在空白区域，此时可以理解为除上课之外的时间，同时由于可能周次与课表存在关系，因此某课程的时间段可以重叠，并且由于课表需要周期性重复，因此课表存在周期性，由于大学课表中的课程可能在分布在每周的多个时间段，因此每个课程可能对应多个时间段。

总结：

值班管理系统：时间轴无空白，时间段无重叠，不存在周期性，对应一段时间。

进程管理系统：时间轴可能有空白，时间段无重叠，不存在周期性，对应一段或多段时间。

课表管理系统：时间轴可能有空表，时间段可能有重叠，存在周期性，对应一段或多段时间。

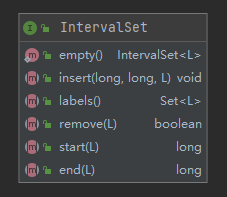
## 面向可复用性和可维护性的设计：IntervalSet<L>

该节是本实验的核心部分。

### IntervalSet<L>的共性操作

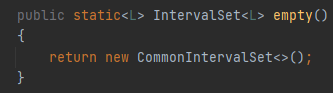
IntervalSet是mutable类型的ADT，它描述了一组在时间轴上分布的“时间段”（interval），每个时间段附着一个特定的标签，且标签不重复，且标签可以是任何immutable类型的ADT，也就是IntervalSet<L>中的L。

IntervalSet的方法如下：



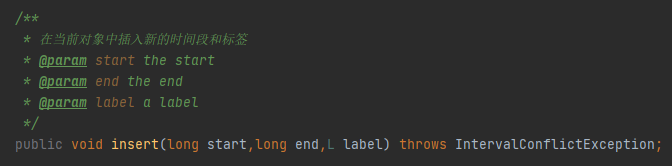
接口的方法说明如下：

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



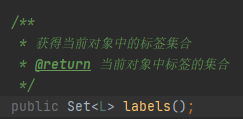
使用static修饰empty方法，可以直接调用，并返回接口类的实例。

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



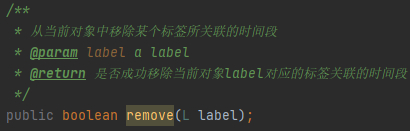
在当前对象中插入新的时间段及其所对应的标签。

* public Set<L> labels();



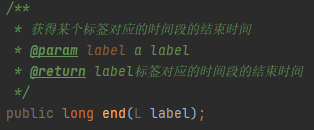
获得当前对象中所有的标签的集合。

* public boolean remove(L label);



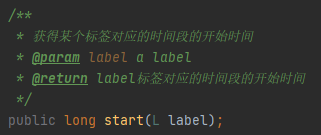
从当前对象中移除某个标签所关联的时间段

* public long end(L label);



获得某个标签所对应的时间段的结束时间。

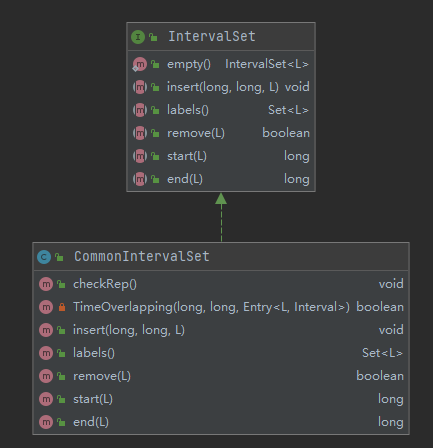
* public long start(L label);



获得某个标签所对应的时间段的开始时间。

### 局部共性特征的设计方案

在CommonIntervalSet<L>类中对Interval<L>实现。通过IDEA，生成如下继承关系图：

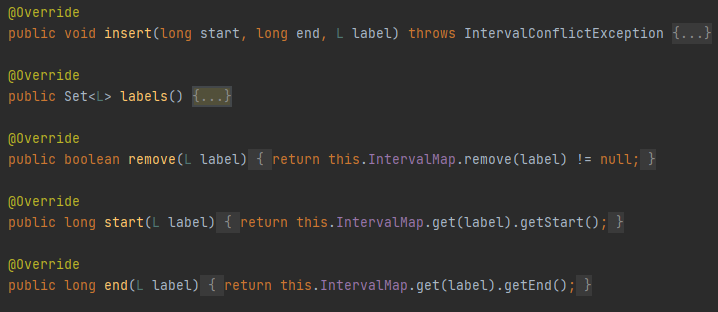


CommonIntervalSet<L>类的说明如下：

私有变量：IntervalMap是由L到Interval的映射关系。

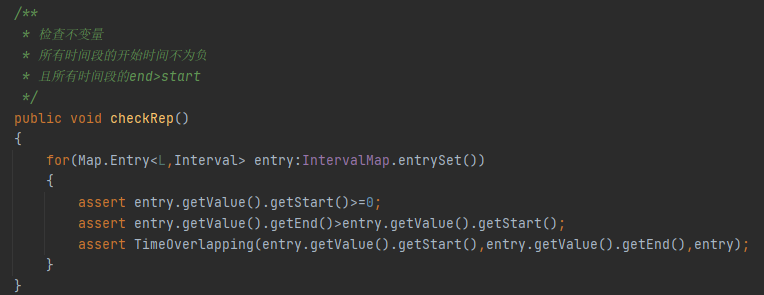


Override方法：



私有方法：

* pulibc void checkRep();



检查不变量、所有时间段的开始时间不为负、且所有时间段的end>start

* private boolean TimeOverlapping(long start,long end,Map.Entry<L,Interval> entry)

检查是否存在时间重叠

// Abstraction function:

// 由Map构成的映射关系

// 即从L映射到Interval的关系

// Representation invariant:

// 标签不重复且为不为空

// 所有时间段的开始时间不为负

// 且所有时间段的end>start

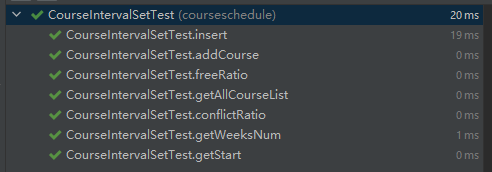
// Safety from rep exposure:

// 使用private、final修饰变量

// 采用防御性复制

根据测试优先，编写以下测试，测试CommonIntervalSet的方法：





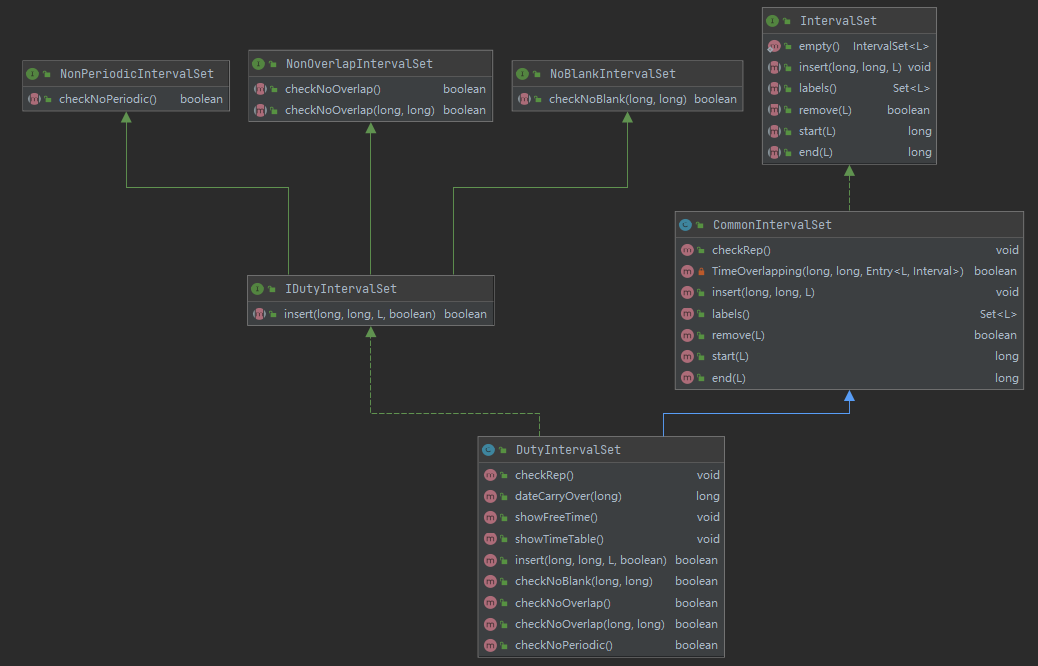
通过所有测试，并覆盖了CourseIntervalSet中的所有方法。



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

通过上述分析，详见3.1.可以得出，只有排班管理系统需要使用IntervalSet。通过设置无重叠时间委托、无空闲时间委托、无周期时间委托，使其满足排班管理系统的无空闲时间、无重叠时间、无周期时间的特性。因此需要对每个维度上的不同特征值定义不同的接口，在接口中定义特殊的操作，各应用的具体子类根据自己需要实现不同特征的接口。分别实现该温度接口的不同实现类，实现该组合接口。在应用子类内，通过delegation到外部的每个维度上实现类的相应特殊操作。NonOverlapIntervalSet 是一个 mutable 的 ADT，用于在 IntervalSet 的基础上，添加“时间段”不可重叠的性质，每个时间段附着一个特定的标签，标签类型为 L，类型为immutable。NoBlankIntervalSet 是一个 mutable 的 ADT，用于在 IntervalSet 的基础上，添加不可出现空闲“时间段”的性质，每个时间段附着一个特定的标签，标签类型为 L，类型为immutable。NonPeriodicIntervalSet 是一个 mutable 的 ADT，用于在IntervalSet的基础上，添加不可出现周期“时间段”的性质，每个时间段附着一个特定的标签，标签类型为 L，类型为immutable的。通过委托，排班管理系统保持不可出现空闲时间段、不可出现周期性时间段、不可出现重叠时间段的特性。

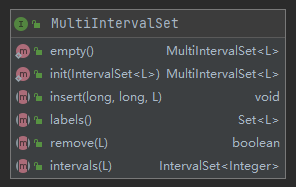
因此在实现DutyIntervalSet时关系如下：



## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

IntervalSet<L>中，同一个标签对象L只被绑定到唯一一个时间段上，这可以满足诸如排班表管理这样的简单应用，却无法满足诸如操作系统进程调度和课程表管理这种复杂应用：同一个标签对象L可被绑定到多个时间段上。 MultiIntervalSet的方法如下：



接口的方法说明如下：

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



在当前对象中插入与label关联的时间段。

* public Set<L> labels();



获得当前对象中的标签的集合

* public boolean remove(L label);



从当前对象中移除label标签所关联的所有时间段

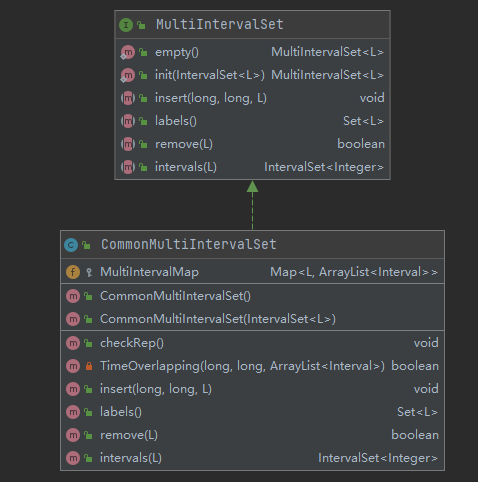
* public IntervalSet<Integer> intervals(L label);



从当前对象中获取与某个标签所关联的所有时间段，且它的时间段按照开始时间从小到大次序排列

### 局部共性特征的设计方案

在CommonMultiIntervalSet<L>类中对MultiInterval<L>实现。通过IDEA，生成如下继承关系图：

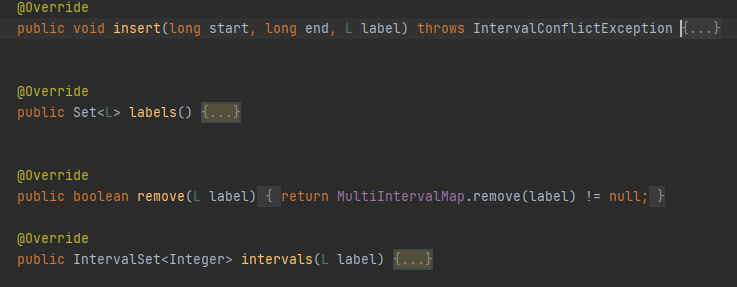


CommonMultiIntervalSet<L>类的说明如下：

私有变量：MultiIntervalMap是由L到ArrayList<Interval>的映射关系。

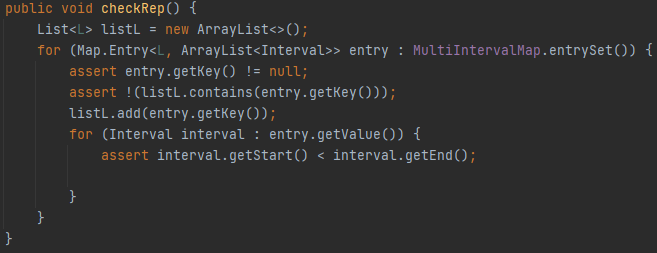


Override方法：



私有方法：

* public void checkRep()



检查不变量，检查label不重复，label对应的标签不存在重叠的时间。

// Abstraction function:

// 由Map构成的映射关系

// 即从L到ArrayList<Interval>的映射构成的IntervalSet

// Representation invariant:

// ArrayList中不存在重叠的时间

// label不为空

// Map中不存在相同的label

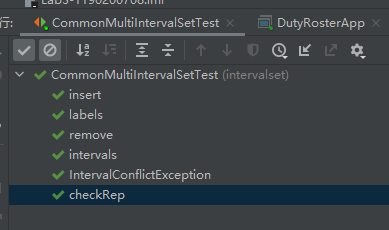
// Safety from rep exposure:

// 使用private、final修饰变量

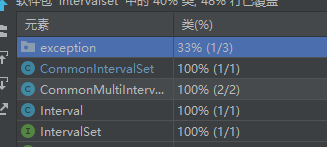
// 采用防御性复制

根据测试优先，编写以下测试，测试CommonMultiIntervalSet的方法：





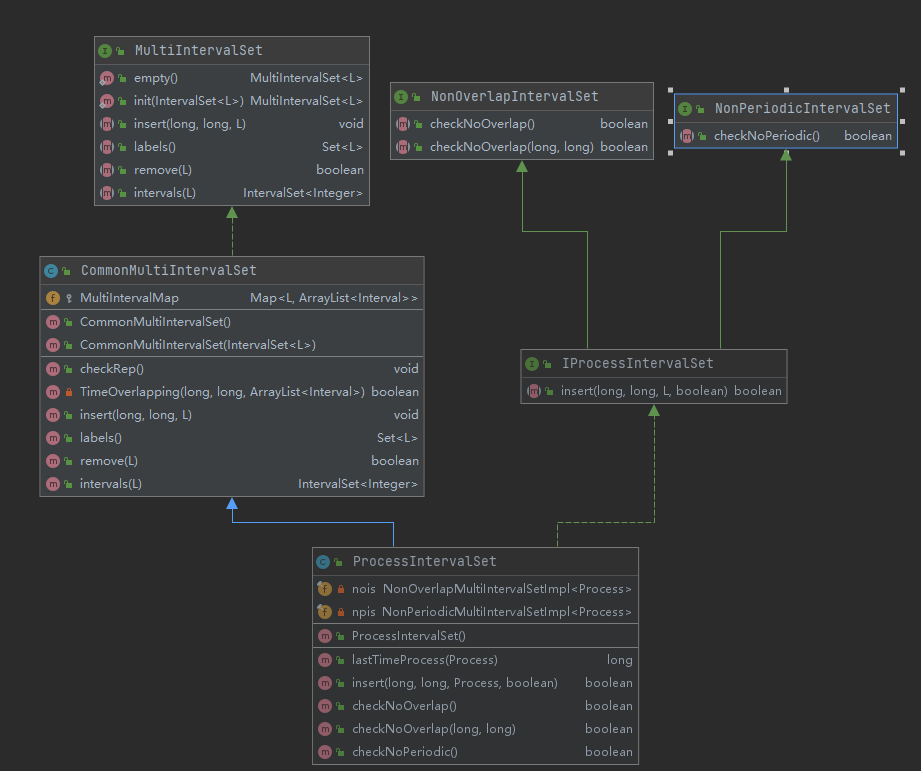
通过所有测试，并覆盖了CourseMultiIntervalSet中的所有方法。



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

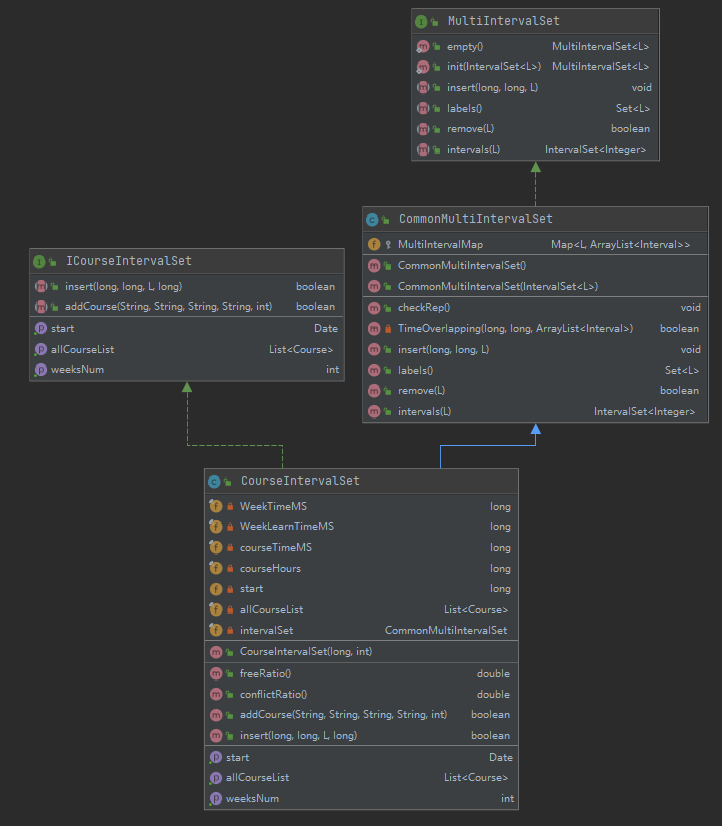
通过上述分析，详见3.1.可以得出，只有进程管理系统、大学课表管理系统需要使用MultiIntervalSet。因此需要分别对子类设置为无重叠时间委托、无周期时间委托和不设置委托。需要对每个维度上的不同特征值定义不同的接口，在接口中定义特殊的操作，各应用的具体子类根据自己需要实现不同特征的接口。分别实现该温度接口的不同实现类，实现该组合接口。在应用子类内，通过delegation到外部的每个维度上实现类的相应特殊操作。NonOverlapIntervalSet 是一个 mutable 的 ADT，用于在 IntervalSet 的基础上，添加“时间段”不可重叠的性质，每个时间段附着一个特定的标签，标签类型为 L，类型为immutable。NonPeriodicIntervalSet 是一个 mutable 的 ADT，用于在IntervalSet的基础上，添加不可出现周期“时间段”的性质，每个时间段附着一个特定的标签，标签类型为 L，类型为immutable的。通过委托，排班管理系统保持不可出现空闲时间段、不可出现周期性时间段、不可出现重叠时间段的特性。

在实现ProcessIntervalSet时关系如下：



可以看到，在ProcessIntervalSet中需要设置NonOverlapIntervalSet和NonPeriodicIntervalSet委托来保证ProcessIntervalSet中没有重叠及周期性，而在CourseIntervalSet中，则没有这样的要求。

因此在实现CourseIntervalSet时关系如下：

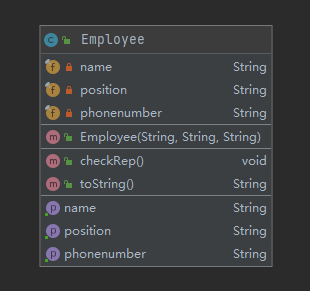
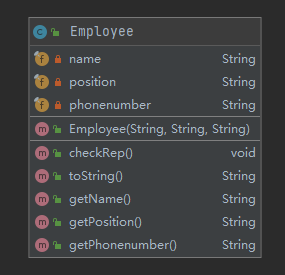


## 面向复用的设计：L

### 概述

在实验要求的三个场景中，我们依次需要实现以下三种标签：员工Employee、进程Process、课程Course。因此我们将员工、进程、课程的属性使用数据类型表示，再分别将实现的三种抽象ADT作为时间段的标签L。

### 员工Employee

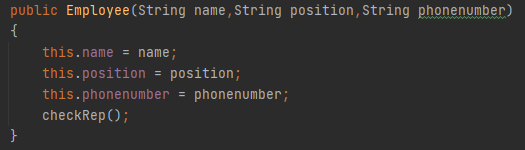
 

员工类的总体属性如上图所示。

员工类是一个immutable的抽象数据类型，员工需要分别包括姓名、职位、手机号等信息。分别通过字符串类型来表示这三个数据。并且通过private、final等字段，将它的设置为私有的，并且不可修改(Safety from Rep Exposure)。

员工类有一个构造函数：

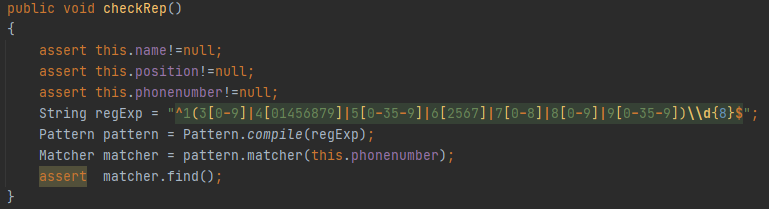
public Employee(String name,String position,String phonenumber);



创建员工初始信息，通过传入的name、position、phonenumber对员工初始化设置。

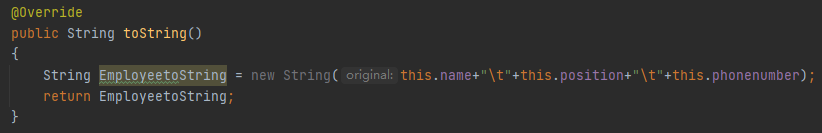
员工类有五个方法：

* public void checkRep()



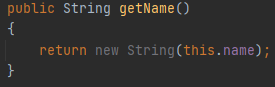
检查不变量，员工名字是否为空，员工职位是否为空，员工手机号是否满足运营商号段，运营商号段如下，移动号段：158、159、172、178、182、183、184、187、188、198、139、138、137、136、135、134、147、150、151、152、157，联通号段：130、131、132、140、145、186、145、175、176、146、155、156、166、167、185，电信号段：133、149、153、177、173、180、181、189、191、199。员工手机号需要满足以上号段。

* public String toString()



将Employee类的数据转化为字符串。方便打印值班信息，同时保证了数据的安全。

* public String getName()



获得员工的姓名，返回员工的姓名

* public String getPosition()



获得员工的职位，员工的职位

* public String getPhonenumber()



获得员工的手机号，返回员工的手机号

rep、AF、RI、Safety from Rep Exposure 说明如下：

// Abstraction function:

// 由name、Position、PhoneNumber映射的员工

// 包含员工的姓名、职位、手机号

// Representation invariant:

// 姓名非空、手机号非空、职位非空

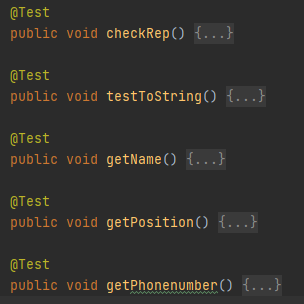
// 手机号满足运营商号段

// Safety from rep exposure:

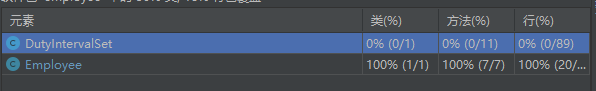
// 使用private、final修饰变量

// 采用防御性复制

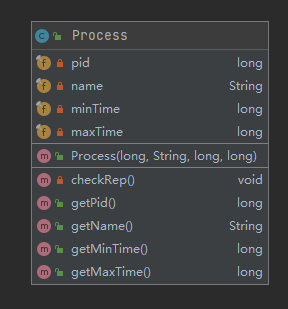
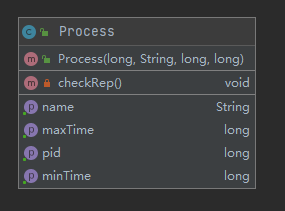
测试：通过分区测试，对每一个分支进行详尽的测试，由于测试逻辑较为简单，且只有返回字符串等基本操作，因此不详细展开测试策略。



可以看到全部通过测试，并且覆盖率为100%.



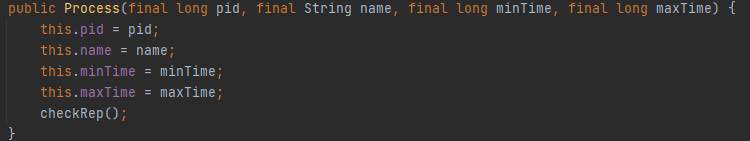
### 进程Process

进程类是一个immutable的抽象数据类型，进程需要分别包括pid、进程名称、进程最短运行时间、进程最长运行时间等信息。分别通过字符串和长整型类型来表示这四个数据。并且通过private、final等字段，将它的设置为私有的，并且不可修改(Safety from Rep Exposure)。

进程类有一个构造函数：

public Process(final long pid, final String name, final long minTime, final long maxTime)



创建进程的初始信息，通过参数pid、name、minTime、maxTime初始化Process。

进程类有五个方法：

* public long getPid();



获取进程的Pid。

* public String getName();



获取进程的名称。

* public long getMinTime();



获取进程的最小执行时间。

* public long getMaxTime();



获取进程的最大执行时间。

rep、AF、RI、Safety from Rep Exposure 说明如下：

// Abstraction function:

// 由pid、name、minTime、maxTime映射的进程

// 包含进程的pid、name、minTime、maxTime

// Representation invariant:

// 0<minTime<maxTime

// pid >= 0

// name != null

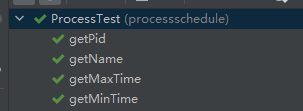
// Safety from rep exposure:

// 使用private、final修饰变量

// 采用防御性复制

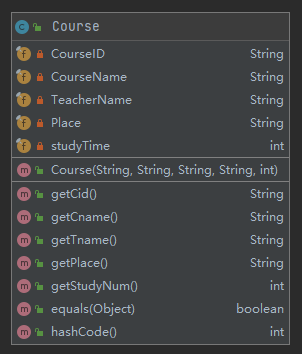
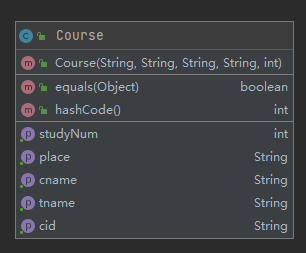
测试：通过分区测试，对每一个分支进行详尽的测试，由于测试逻辑较为简单，且只有返回字符串等基本操作，因此不详细展开测试策略。

可以看到全部通过测试，并且覆盖率为100%.





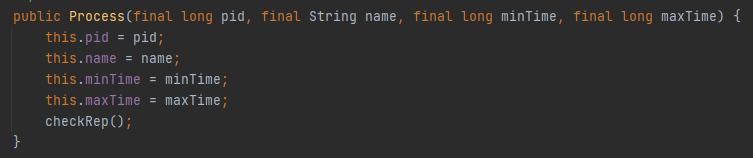
### 课程Course



课程类是一个immutable的抽象数据类型，课程需要分别包括CourseID、CourseName、TeacherName、Place、studyTime等信息，分别通过字符串类型、整型类型来表示。

课程类有一个构造函数：

public Process(final long pid, final String name, final long minTime, final long maxTime);



通过传入的参数pid、name、minTime、maxTime对course初始化Course。

Course类有七个方法：

* public String getCid()



获得课程的ID。

* public String getCname()



获得课程的名称。

* public String getTname()



获得教师的名称。

* public String getPlace()



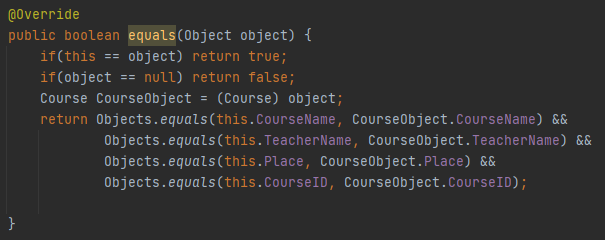
获得授课的地点。

* public int getStudyNum()



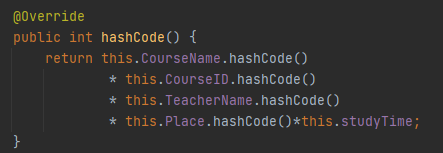
获得每周授课的学时。

* public boolean equals(Object object)



判断两个Course是否相同。

* public int hashCode()



计算Course的hashCode。

rep、AF、RI、Safety from Rep Exposure 说明如下：

// Abstraction function:

// 由CourseID、CourseName、TeacherName、Place、studyTime映射的课程

// 包含课程的ID、课程的名字、教师的名字、授课地点、每周学时

// Representation invariant:

// CourseID、CourseName、TeacherName、Place不为空

// studyTime 为偶数且大于0

// Safety from rep exposure:

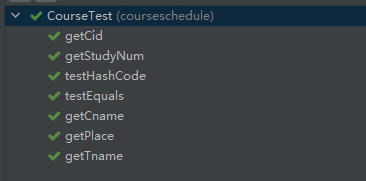
// 使用private、final修饰变量

// 采用防御性复制

测试：通过分区测试，对每一个分支进行详尽的测试，由于测试逻辑较为简单，且只有返回字符串等基本操作，因此不详细展开测试策略。

可以看到全部通过测试，并且覆盖率为100%.



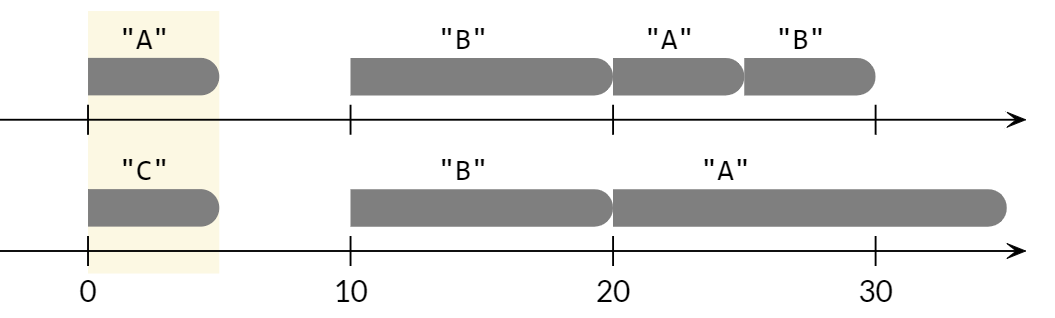


## 可复用API设计

### 计算相似度

首先分析相似度的计算：

我们以实验指导的例子为例：



首先我们可以得出对于第一个MultiIntervalSet中标签A，其对应的区域有两个地方[0,5)和[20，25)，而在第二个MultiIntervalSet中标签A，其对应的区域有一个为[20,35)，而由于不同标签的相似度，只存在于相同的标签对应的时间段，因此我们可以计算出相似度。Similar ratio = 15/35=0.42857。

在介绍方法前，首先简述求相似度的辅助函数：

1. public static <L> long getStartTimerShaft(MultiIntervalSet<L> s)

获得MultiIntervalSet的最早的时间

1. public static <L> long getEndTimeShaft(MultiIntervalSet<L> s)

获得MultiIntervalSet的最晚的时间

1. public static double intervalLength(Set<Interval> s1)

计算s1中的所有时间段的时间的长度

1. public static Set<Interval> similarIntervals(IntervalSet<Integer> s1, IntervalSet<Integer> s2)

求s1与s2的重合时间段，其中参数为由MultiIntervalSet<L>的label对应的多个Interval构成的Interval<Integer>， Integer为从小到大的顺序。

下方是对求相似度方法的概述

1. 首先利用getStartTimerShaft()和getEndTimerShaft()，获得两个MultiIntervalSet中最早的时间和最晚的时间，并相减得到总长度。
2. 遍历s1中的所有标签，例当标签为label时，通过intervals函数即可获得s1中所有标签为label的时间段，并以Integer为label标记排好序。再通过intervals函数获得s2中所有标签为label的时间段，并以Integer为label标记排好序。
3. 此时调用similarIntervals函数，获得两个IntervalSet中相同的部分。
4. 调用intervalLength函数，获得相同的部分的长度。
5. 重复上述循环
6. 通过计算相同的部分的长度/总长度，即为相似度。

具体函数如下：



### 计算时间冲突比例

时间冲突比例应为计算一个MultiIntervalSet类型中所有labels对应的时间段存在重叠的部分。

首先将所有的label映射到Boolean类型，即可认为当前是否已经遍历过该标签。初始化时候，将label对应的Boolean都设置为false。

在循环MultiIntervalSet中所有标签时，依次将每一个设置为true，表示遍历完成。通过双层循环实现对label的遍历，在遍历一个标签时，若发现另一个标签与其相同或已访问过，则跳过，否则通过调用simlarIntervals函数和intervals函数，检查label标签对应的时间段与待检查标签中是否有相同的部分，若有，则累加，即为冲突时间。

总时间通过对每一个label对应时间段求和即可得。

冲突事件/总时间即为时间冲突比例。

### 计算空闲时间比例

首先通过3.5.1中叙述的getStartTimerShaft(MultiIntervalSET<L>)函数，获得MultiIntervalSet的最早的时间点，再通过getEndTimerShaft(MultiIntervalSET<L>)函数，即可获得MultiIntervalSet的最晚的时间点。

此时即可计算总的时间。

首先将开头时间和结束时间放入Set<Inteval>中，即可得到此时对应的Interval的集合，开始时候，只有一个Interval，代表从头到结尾。

通过对MultiIntervalSet中的labels遍历，依次得到每个label对应的时间段。

当得到每个时间段后，将Set中的包含当前label对应的时间段的时间段拆分成，不包括不包括label的时间段的部分，并将原Interval删除，将新拆分的Intervals放入Set中，最后调用intervalLength函数，即可计算Set中的时间间隔的总长度。

通过空闲时间/总时间即可计算出空闲时间比例。

## 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

应用分析：设计出的排班管理系统应可以实现入下功能

针对排班管理系统，所需完成的功能为：

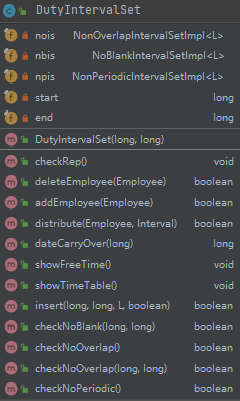
Step 1 设定排班开始日期、结束日期，具体到年月日即可。

Step 2 增加一组员工，包括他们各自的名字、职务、手机号码，并可随时删除某些员工。如果某个员工已经被编排进排班表，那么他不能被删除，必须将其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。

Step 3 可手工选择某个员工、某个时间段（以“日”为单位，最小1天，可以是多天），向排班表增加一条排班记录，该步骤可重复执行多次。在该过程中，用户可随时检查当前排班是否已满（即所有时间段都已被安排了特定员工值班）、若未满，则展示给用户哪些时间段未安排、未安排的时间段占总时间段的比例。

Step 4 除了上一步骤中手工安排，也可采用自动编排的方法，随机生成排班表。

Step 5 可视化展示任意时刻的排班表。可视化要直观明了，可自行设计。



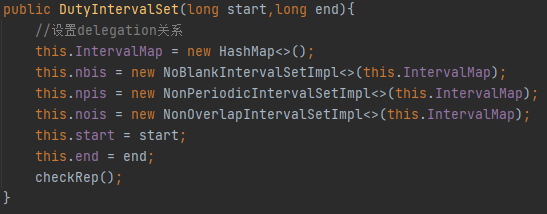
简要分析：我们只需要使用一个标签对应一个时间段的IntervalSet类，并且需要设置NoBlankIntervalSet、NonPeriodicIntervalSet、NonOverlapIntervalSet的委托，实现其没有空闲时间、没有周期性、没有重叠时间的特性。

我们所设计的ADT如下：

DutyIntervalSet如图所示，具有5个私有变量，nois用于设置无重叠时间委托，nbis用于设置无空闲时间段委托，npis用于设置无周期时间段委托，start表示排班的起始时间，end 表示排班的截止时间，employee用于表示员工的集合。

DutyIntervalSet具有一个构造函数：

public DutyIntervalSet(long start,long end);



通过传入的start和end设置排班的起始和截止时间，并设置委托。

DutyIntervalSet具有十二个方法，如下所示：

* public void checkRep();

检查不变量

* public boolean deleteEmployee(Employee employee);

删除某个员工的信息

* public boolean addEmployee(Employee employee) ;

添加某个员工的信息

* public boolean distribute(Employee employee, Interval interval)

给某个员工分配值班时间

* public long dateCarryOver(long date)

时间进位(后采用Date)

* public void showFreeTime()

展示剩余未分配的时间段

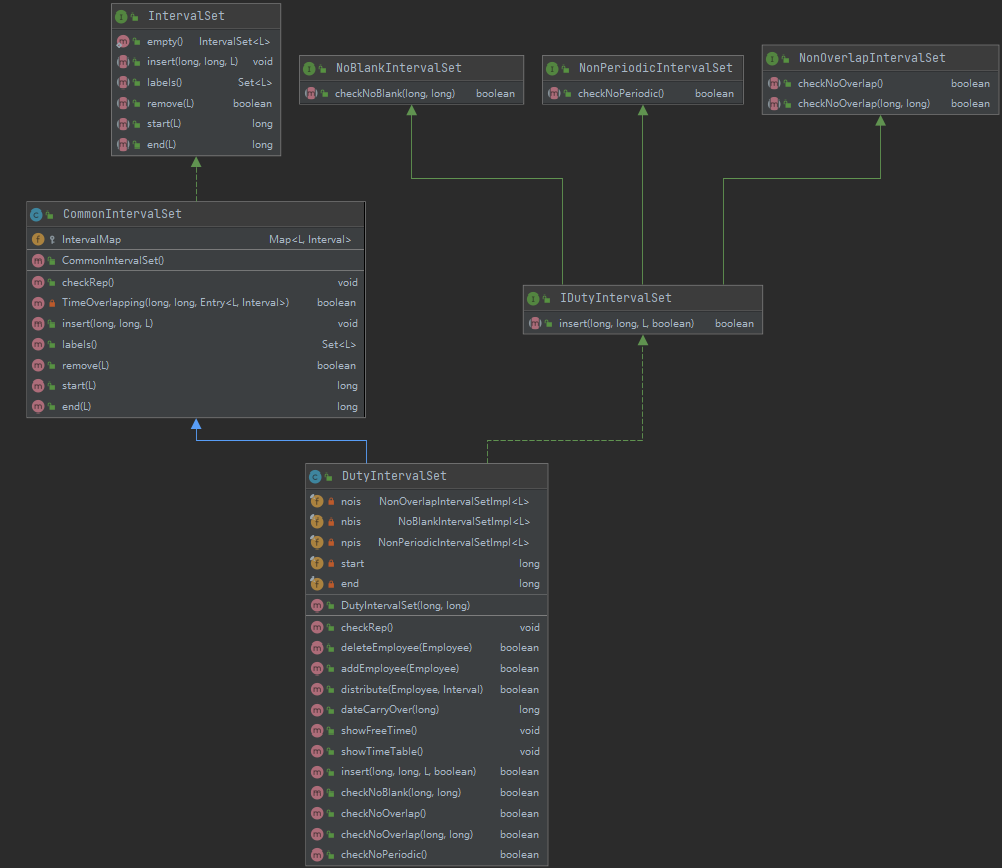
* public void showTimeTable()

展示所有已安排的时间段

* public boolean insert (long start, long end, L label, boolean finish) throws IntervalConflictException,IntervalBlankException,IntervalPeriodicException

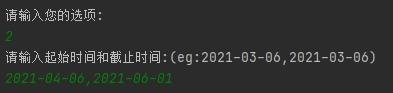
插入新的标签及其对应的时间段

DutyIntervalSet的关系图如下所示：

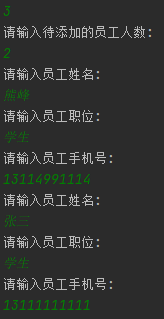


运行结果如下：

初始化值班表：



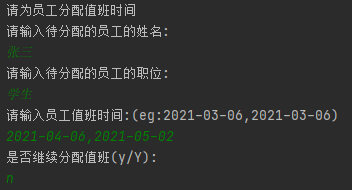
添加员工信息：



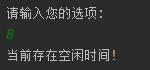
显示所有员工信息：



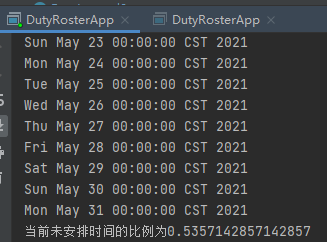
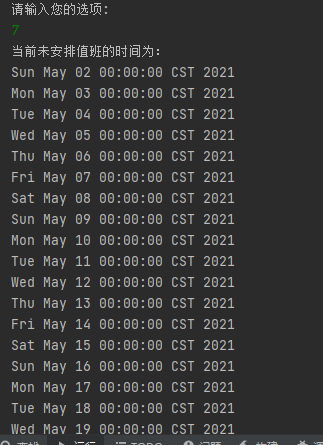
为员工分配值班时间



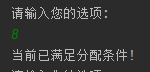
检查当前分配是否合理：



检查当前空闲时间：



当满足条件时：



### 操作系统的进程调度管理系统

针对操作系统的进程调度管理系统，所需完成的功能为：

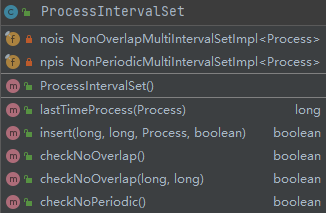
Step 1 增加一组进程，输入每个进程的ID、名称、最短执行时间、最长执行时间；进程一旦设定无法再修改其信息。

Step 2 当前时刻（设定为0）启动模拟调度，随机选择某个尚未执行结束的进程在CPU上执行（执行过程中其他进程不能被执行），并在该进程最大时间之前的任意时刻停止执行，如果本次及其之间的累积执行时间已落到[最小, 最大]的区间内，则该进程被设定为“执行结束”。重复上述过程，直到所有进程都达到“执行结束”状态。在每次选择时，也可“不执行任何进程”，并在后续随机选定的时间点再次进行进程选择。

Step 3 上一步骤是“随机选择进程”的模拟策略，还可以实现“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。

Step 4 可视化展示当前时刻之前的进程调度结果，以及当前时刻正在执行的进程。可视化的形式要直观明了，可自行设计。

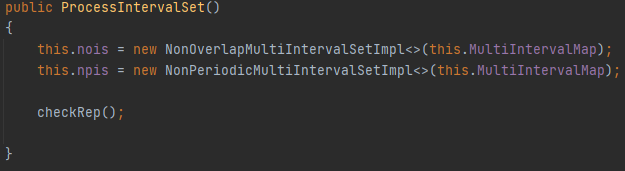
简要分析：我们需要一个一个标签对应多个时间段的数据类型，因此选择MultiIntervalSet类型。



ProcessIntervalSet中有两个私有变量即NonOverLapMultiIntervalSet和NonPeriodicMultiIntervalSet的委托nois、npis。通过这两个委托可以实现无周期性和无重叠部分的特性。

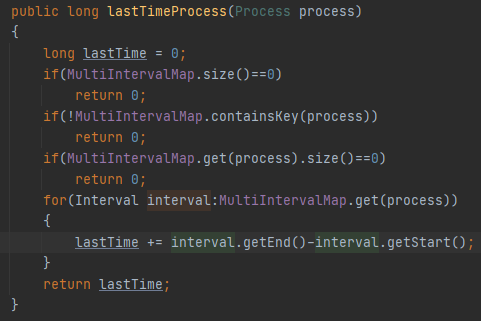
通过ProcessIntervalSet作为构造函数：

设置委托关系，并检查不变量。



ProcessIntervalSet类有五个方法：

* public long lastTimeProcess(Process process)

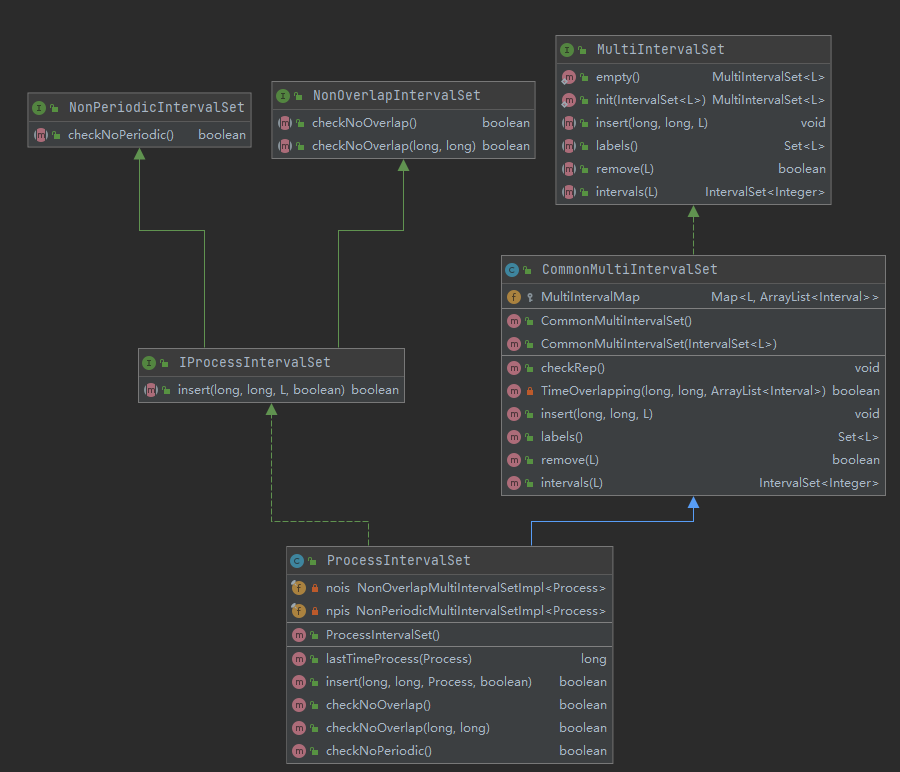


获得当前程序执行的时间

* public boolean insert(long start, long end, Process label, boolean finish) throws IntervalConflictException, IntervalBlankException, IntervalPeriodicException

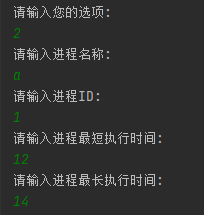
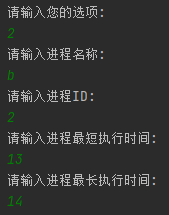
插入进程label，并且其运行起始时间为start，截止时间为end。

剩下方法为调用父类方法。具体关系如下所示：

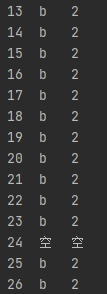
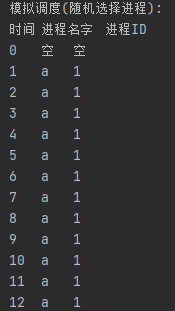


运行结果：

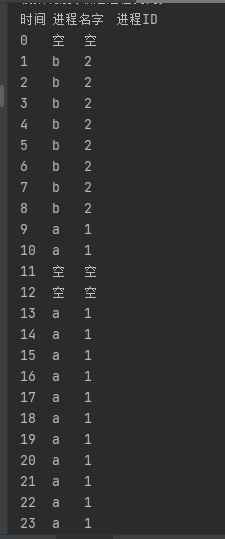
添加一组进程：

模拟调度：



最短进程优先：



### 课表管理系统

针对课表管理系统，所需完成的功能为：

Step 1 设定学期开始日期（年月日）和总周数（例如18）；

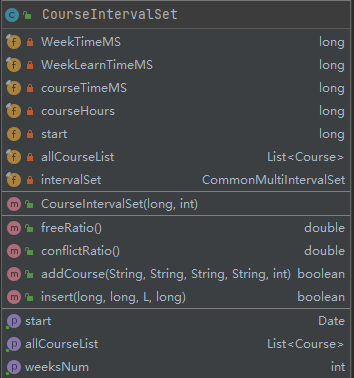
Step 2 增加一组课程，每门课程的信息包括：课程ID、课程名称、教师名字、地点、周学时数（偶数）；

Step 3 手工选择某个课程、上课时间（只能是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；

Step 4 上步骤过程中，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；

Step 5 因为课程是周期性的，用户可查看本学期内任意一天的课表结果。

简要分析，我们需要一个一个标签对应多个时间段的数据类型，因此选择MultiIntervalSet类型。



CourseIntervalSet有七个私有变量：

* private final long WeekTimeMS = 7 \* 24 \* 60 \* 60 \* 1000;

MS即毫秒数，一周的总毫秒数

* private final long WeekLearnTimeMS = 7 \* 5 \* 2 \* 60 \* 60 \* 1000;

每周的学时总毫秒数，每天五节课，每节课两小时

* private final long courseTimeMS = 2 \* 60 \* 60 \* 1000;

每节课的课时，两小时的毫秒数

* private final long courseHours = 2;

每节课的小时数

* private long start;

学期开始时间

* private int weeksNum;

学期周数

* private final List<Course> allCourseList = new ArrayList<>();

所有课程的列表

* private final CommonMultiIntervalSet intervalSet;

用于委派的数据类型，可以单个标签绑定多个时间段的时间段集合

通过CourseIntervalSet(long start,int weeksNum)初始化。

本模块采用到1970的毫秒数，作为时间线，较好的契合APIs中的计算时间冲突、空闲时间等功能。因此有一些固定值例如，每节课课时的毫秒数等。

CourseIntervalSet的私有方法：

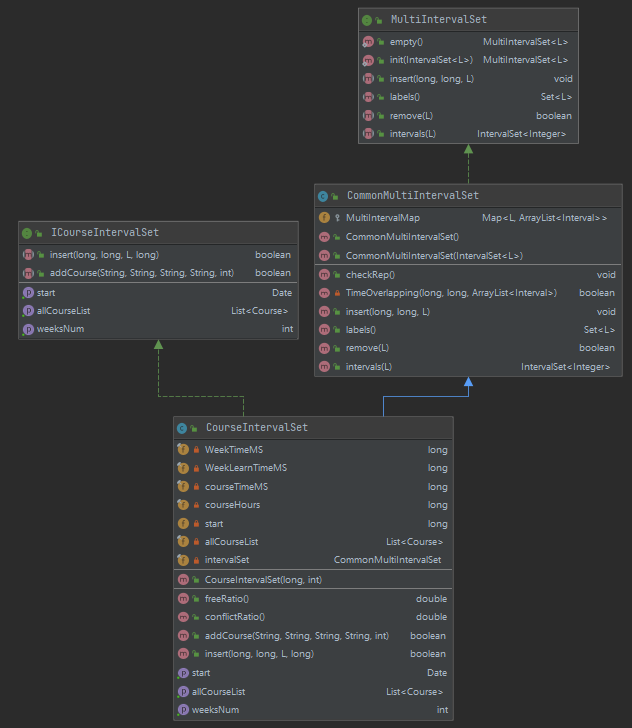
* public double freeRatio();

计算空闲比率

* public double conflictRatio();

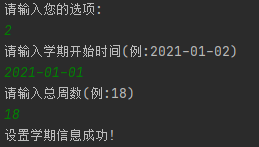
计算冲突比率

其余为override方法，关系如下图所示：

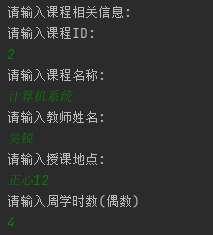
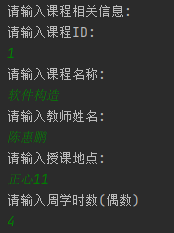


运行结果如下：

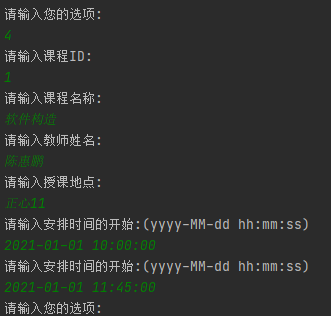
初始化学期：



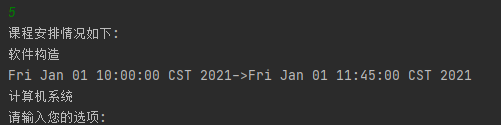
设置一门课程：



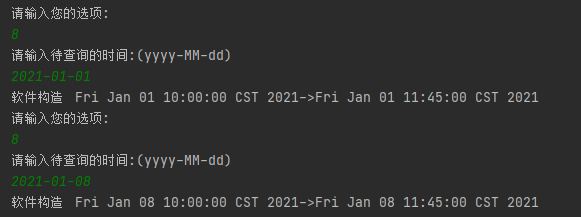
设置一门课程的时间：



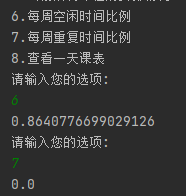
查看当前课程安排情况：



查看某天课表：



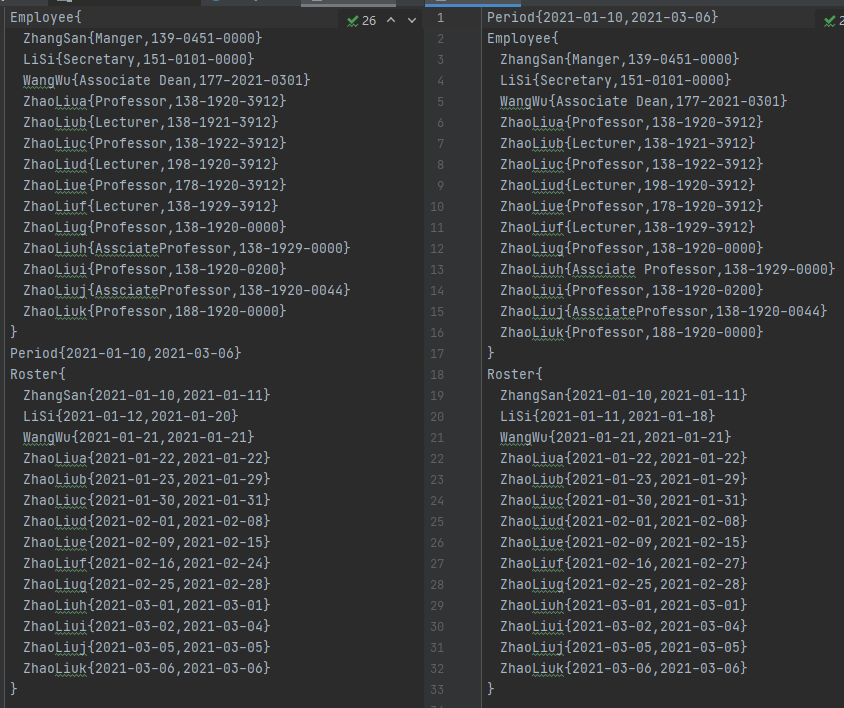
也可以查询每周冲突与空闲时间

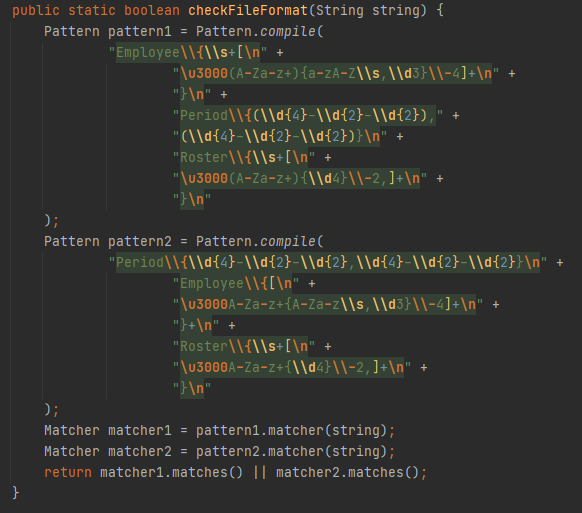


## 基于语法的数据读入

在APIs中提供静态方法，使用正则表达式分析读入的内容是否为所要求的格式串。

根据所提供的文件一共存在两种格式：通过写出这两种的正则表达式进行分析与匹配。

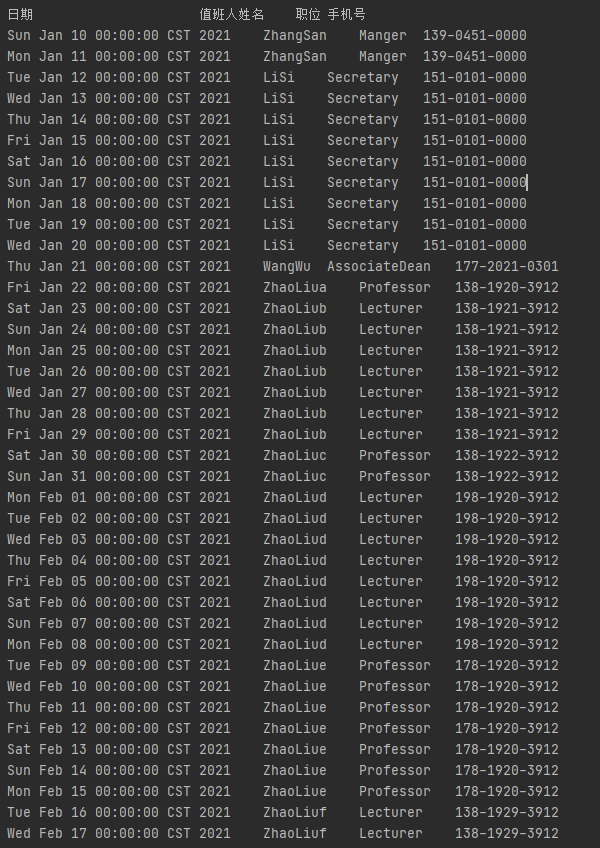


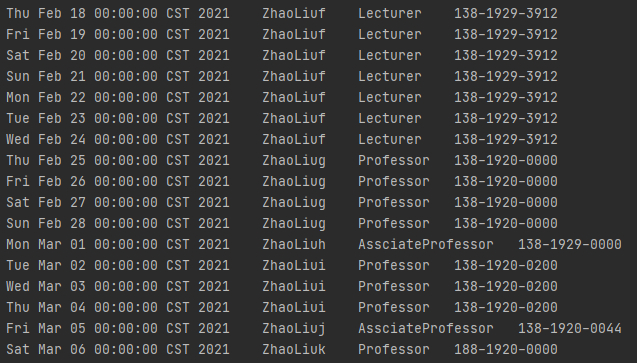


在匹配确认格式匹配后，将所读入的内容转化为值班表。

具体运行结果如下：

当读入test1.txt时，具体运行结果如下所示：



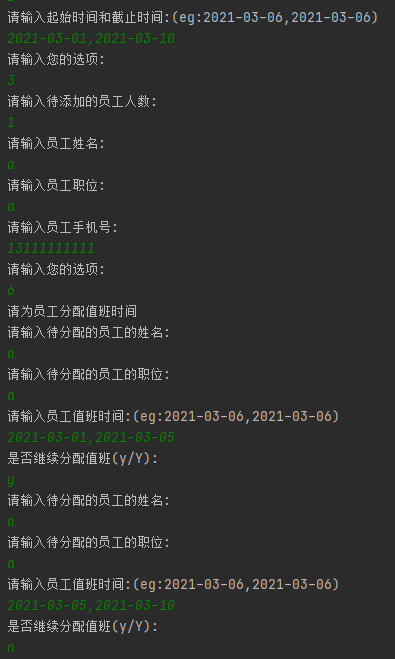


## 应对面临的新变化

### 变化1

由于原函数种含有所有待添加员工的集合，在添加后，即将其从list中删除，现在不将其删除，所有员工都处于待添加状态，即可完成修改。

运行结果如下所示：

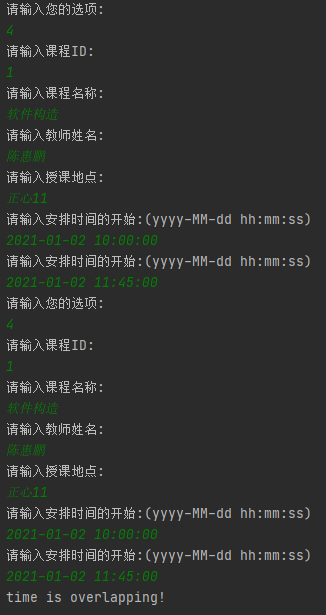




### 变化2

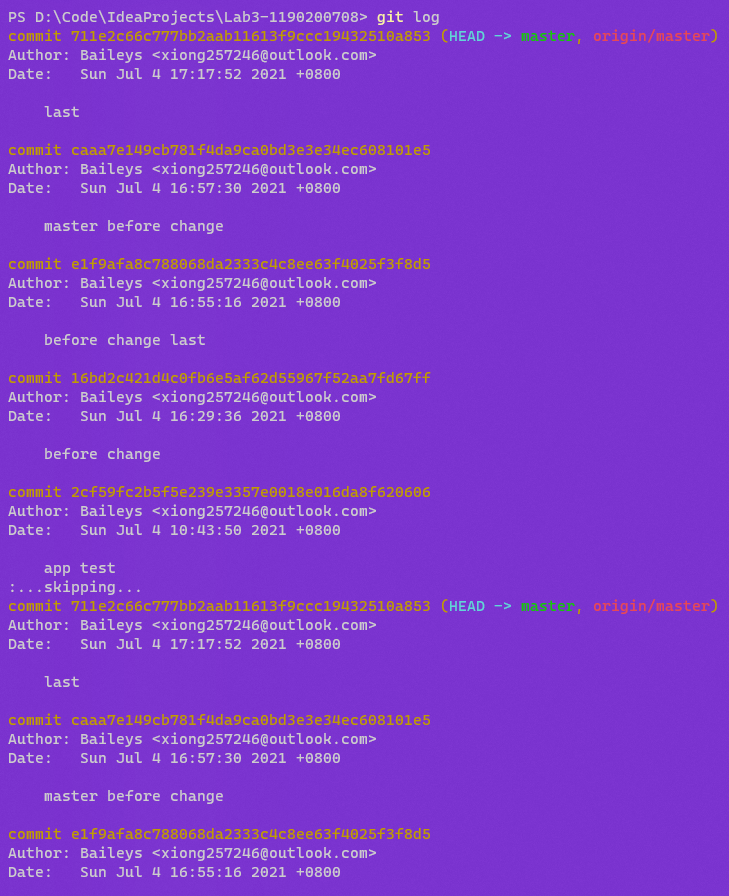
由于实现的时候，添加过辅助数组保存时间间隔，因此可以直接选择查找List中是否存在重叠时间。代价较小。

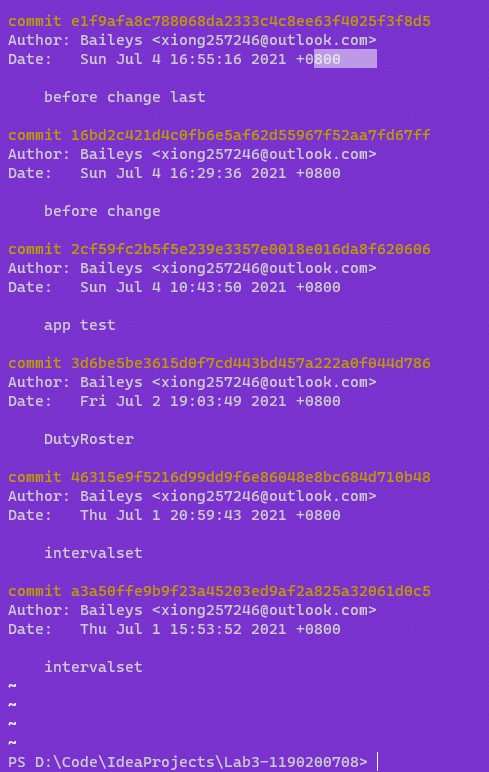
运行结果如下所示：



## 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

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2021.06.28 | 12:00-20:00 | 阅读实验指导，并复习相关知识。 | 按计划完成 |
| 2021.06.29 | 12:00-22:00 | 编写接口，及其实现类 | 按计划完成 |
| 2021.06.30 | 12:00-22:00 | 编写Employee及其实现与测试 | 按计划完成 |
| 2021.07.01 | 10:00-22:00 | 编写Process及其实现与测试 | 按计划完成 |
| 2021.07.02 | 10:00-23:00 | 编写Course及其实现与测试 | 按计划完成 |
| 2021.07.03 | 10:00-23:00 | 完成实验其余部分 | 按计划完成 |
| 2021.07.04 | 10:00-23:00 | 撰写报告 | 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 |
| 对委托的理解不到位 | 阅读资料 |
| 正则表达式的匹配问题 | 阅读资料 |

# 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

对代码复用的体会更加深刻了。

## 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？

复用在一定程度上增加了代码的难度，但另一方面复用大量的减少了代码量。复用更考察了抽象能力，将不同的场景的共同点抽象成抽象ADT。

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？

保证了正确性与安全性，并将一直坚持。

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？

开发API难处体会比较深刻，乐趣没有难处深刻。

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？

匹配过程更加实用。

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？

抽象出相同的地方，勤于思考。

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？

先写接口考虑好应用场景，并写测试，进而不断完善代码。

1. 关于本实验的工作量、难度、deadline。

工作量过大、难度过大、deadline过紧。

1. 到目前为止你对《软件构造》课程的评价。

对我编程起到了很大的作用，让我了解到如何编出更好的代码。