

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 童彦澎 |
| 学号 | 1190201701 |
| 班号 | 1903011 |
| 电子邮件 | [1308091501@qq.com](mailto:1308091501@qq.com) |
| 手机号码 | 18959076997 |

**目录**

[1 实验目标概述 1](#_Toc76301556)

[2 实验环境配置 1](#_Toc76301557)

[3 实验过程 2](#_Toc76301558)

[3.1 待开发的三个应用场景 2](#_Toc76301559)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 3](#_Toc76301560)

[3.2.1 IntervalSet<L>的共性操作 3](#_Toc76301561)

[3.2.2 局部共性特征的设计方案 5](#_Toc76301562)

[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](#_Toc76301563)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 8](#_Toc76301564)

[3.3.1 MultiIntervalSet<L>的共性操作 8](#_Toc76301565)

[3.3.2 局部共性特征的设计方案 10](#_Toc76301566)

[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](#_Toc76301567)

[3.4 面向复用的设计：L 12](#_Toc76301568)

[3.5 可复用API设计 13](#_Toc76301569)

[3.5.1 计算相似度 14](#_Toc76301570)

[3.5.2 计算时间冲突比例 15](#_Toc76301571)

[3.5.3 计算空闲时间比例 15](#_Toc76301572)

[3.6 应用设计与开发 16](#_Toc76301573)

[3.6.1 排班管理系统 16](#_Toc76301574)

[3.6.2 操作系统的进程调度管理系统 19](#_Toc76301575)

[3.6.3 课表管理系统 21](#_Toc76301576)

[3.7 基于语法的数据读入 24](#_Toc76301577)

[3.8 应对面临的新变化 30](#_Toc76301578)

[3.8.1 变化1 30](#_Toc76301579)

[3.8.2 变化2 30](#_Toc76301580)

[3.9 Git仓库结构 30](#_Toc76301581)

[4 实验进度记录 30](#_Toc76301582)

[5 实验过程中遇到的困难与解决途径 31](#_Toc76301583)

[6 实验过程中收获的经验、教训、感想 31](#_Toc76301584)

[6.1 实验过程中收获的经验和教训 31](#_Toc76301585)

[6.2 针对以下方面的感受 31](#_Toc76301586)

# 实验目标概述

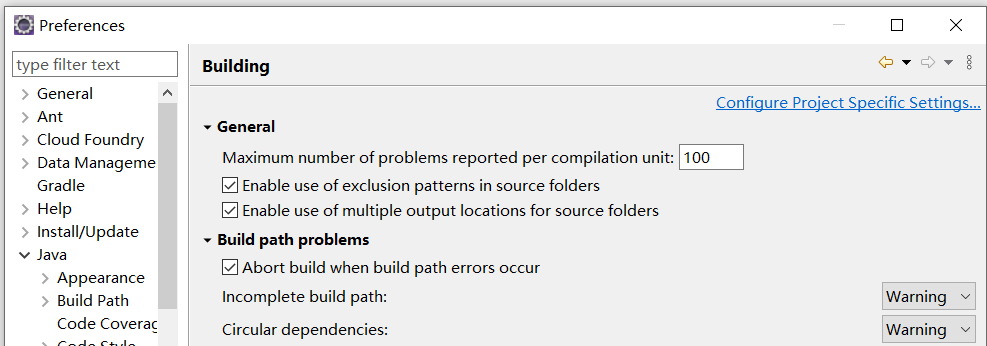
本次实验覆盖课程第 4-11 讲的内容，目标是编写具有可复用性和可维护性的软件，主要使用以下软件构造技术：  
⚫ 子类型、泛型、多态、重写、重载  
⚫ 继承、代理、组合  
⚫ 语法驱动的编程、 正则表达式  
⚫ API 设计、 API 复用  
本次实验给定了三个具体应用（值班表管理、 操作系统进程调度管理、大学  
课表管理），学生不是直接针对每个应用分别编程实现，而是通过 ADT 和泛型等抽象技术，开发一套可复用的 ADT 及其实现，充分考虑这些应用之间的相似性和差异性，使 ADT 有更大程度的复用（可复用性） 和更容易面向各种变化（可维护性）。

# 实验环境配置

简要陈述你配置本次实验所需环境的过程，必要时可以给出屏幕截图。

特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

其余配置与前两个实验类似，但本次实验自己组织子目录，在打包的过程中遇到ClassNotFound Error，通过搜索引擎获得解决方案：



在preference-java-compiler-building中，将前两个选项设为warning

注：若助教老师/老师发现同样的错误，烦请更改一下这两个设置，谢谢！

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

<https://github.com/ComputerScienceHIT/HIT-Lab3-1190201701>

# 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。

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）： 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。

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。

场景分析：

共性：三个应用场景均对时间段进行了抽象，都可以将发生的事件理解为一些时间段的集合。

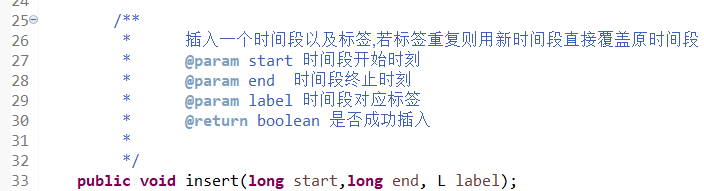
差异：

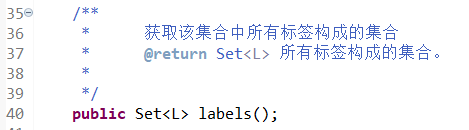
1. 标签不同：这是由应用场景的特殊性决定的，分别为员工、进程、课程。
2. 一个标签能否对应多个时段：在场景一中，排班为连续排班，且假设后续不继续安排，故一个标签不对应多个时段；在场景二中，操作系统的进程调度不是连续的，进程可以被挂起，故一个标签可以对应多个时段；在场景三中，更加复杂，一个标签可以对应多个时段（一周上多次课），并且一个时段还能对应多个标签（选课情况不同）。

## 面向可复用性和可维护性的设计：IntervalSet<L>

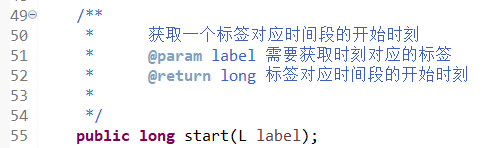
该节是本实验的核心部分。

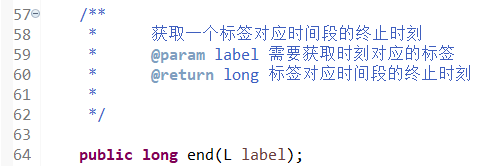
### IntervalSet<L>的共性操作









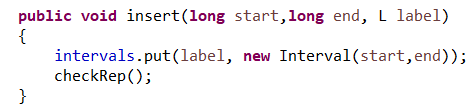


以上为IntervalSet接口的定义。

共性操作通过CommonIntervalSet实现：



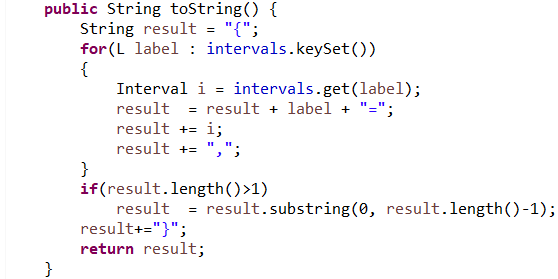
Rep为一个标签到时间段的映射，所有标签的映射即构成了intervalset。



Insert的实现也较为简单，label若重复也是直接put进map中，直接更新map，最后checkRep，检查RI。

注：intervalset的tostring方法采用指导书上的集合表示方式实现：

关键代码：

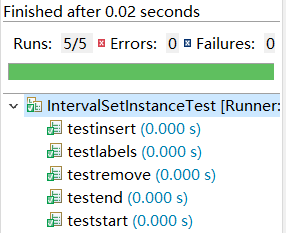


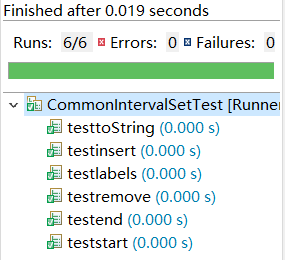
基于interval的tostring方法，结合标签输出即可，对最后一个时间段的“，“做裁剪。

### 局部共性特征的设计方案

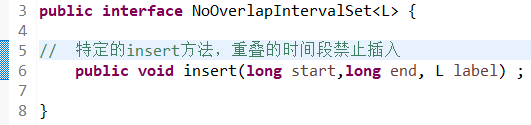
局部共性特征采用继承CommonIntervalSet实现：

Intervalset的实例测试与Common的测试均通过：

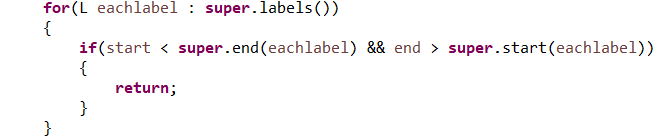




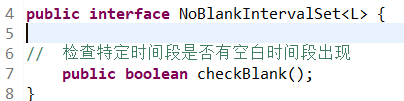
### 面向各应用的IntervalSet子类型设计（个性化特征的设计方案）



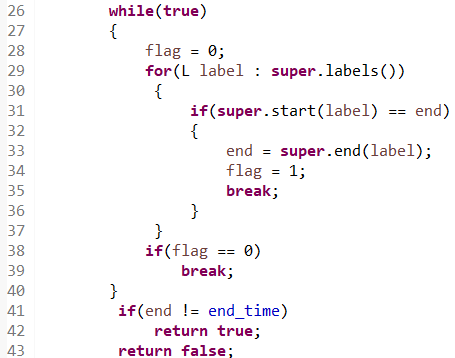
NoOverlap接口要求重写insert方法，有重叠的时间段的插入是不被允许的。

关键代码：

有冲突则不插入。



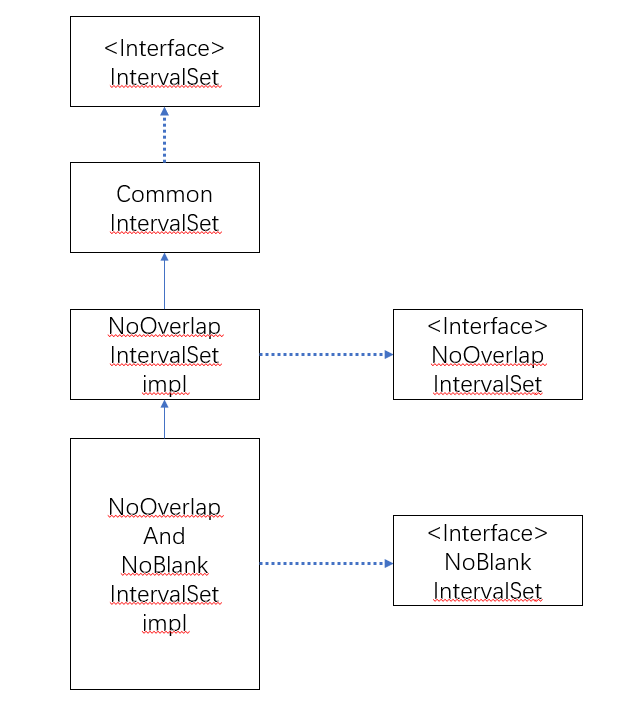
NoBlank接口实现检查某一时间段是否有空白的功能。

关键代码：

主要思想为从起始时间不断依据安排的interval移动，若最后没有移动到end\_time，就是有空白。

通过实现这两个接口，获得个性化的IntervalSet特征。

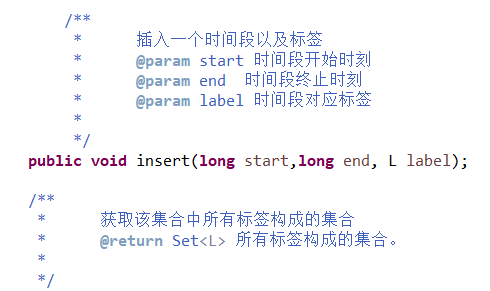
继承树如图所示：

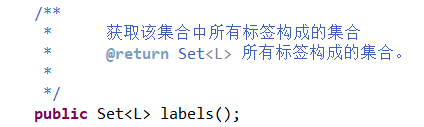


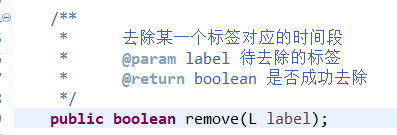
本设计用到两个维度特征，若维度再高则继承树会更加庞大。

## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

MultiIntervalSet接口中的方法：



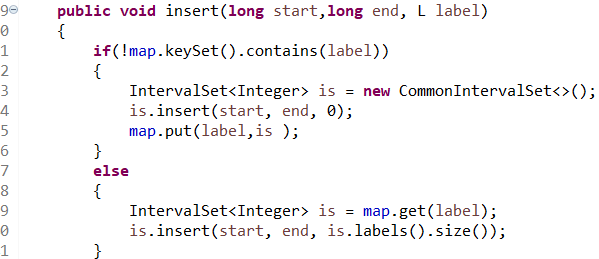




与intervalset类似，通过CommonMultiIntervalSet进行实现：



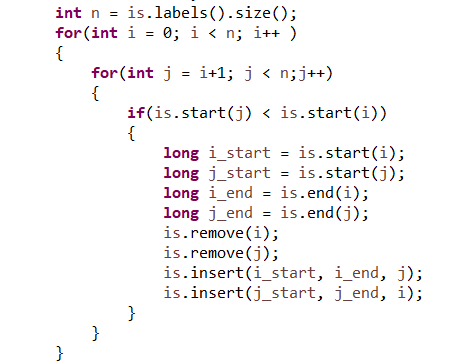
Rep复用了Intervalset，是一个标签到intervalset的映射。若标签重复，则该标签对应的时间段集合可视为是int标号集作为标签。



Insert与intervalset有所不同：标签若不存在，插入还需要新创建一个intervalset；存在加入原有的intervalset即可，标号递增。

toString也是基于intervalset的tostring实现，类似于1的实现，不赘述。

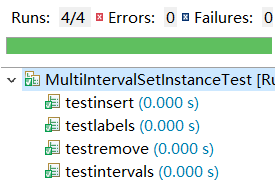
Intervals是较为重要的一个通用方法，关键代码：

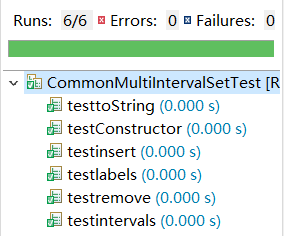


针对以标号集为标签的intervalset作交换排序，实现按照开始时间递增的顺序返回。

### 局部共性特征的设计方案

与1的设计类似，继承CommonMultiIntervalSet实现，实例测试与实现类测试均通过：

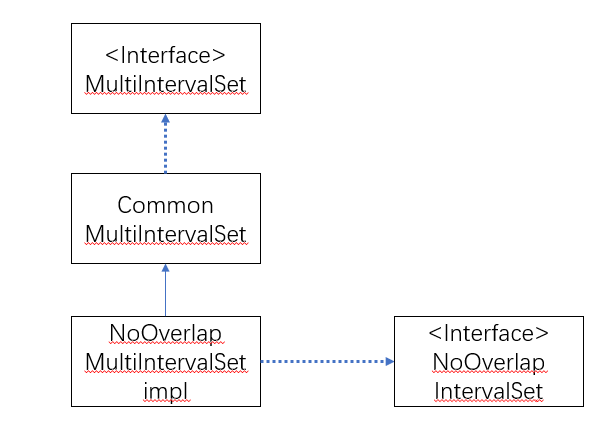




### 面向各应用的MultiIntervalSet子类型设计（个性化特征的设计方案）

与设计1相同的接口，采用继承树，实现个性化特征的获得。

继承树如下图所示：

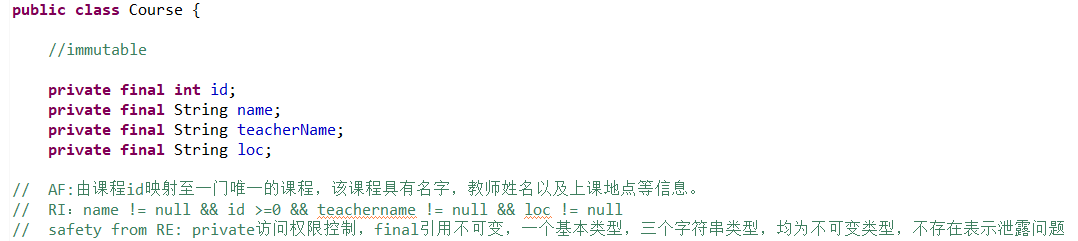


## 面向复用的设计：L

分别设计Course、Employee、Process三个ADT，通过泛型实现复用。

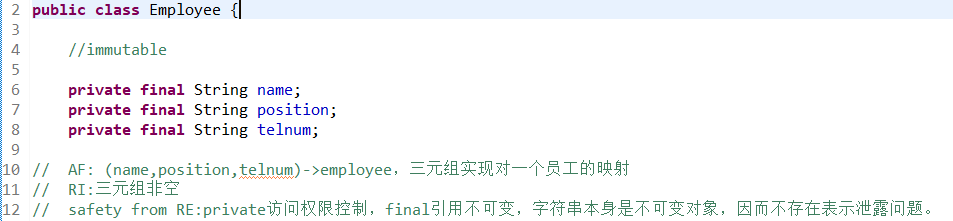
三个ADT的方法主要包含：setter、tostring、equals等

Course：



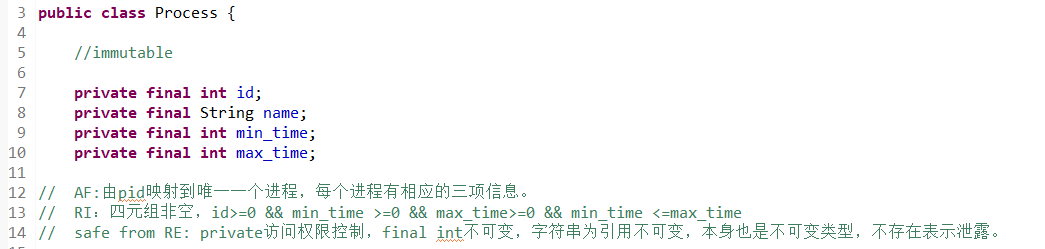
Course的equals标准为课程id相等则认为两个对象相等，Override。

Employee：



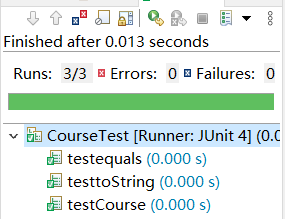
Equals标准为三元组相同，考虑重名且重职位的情况。

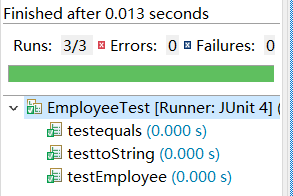
Process：

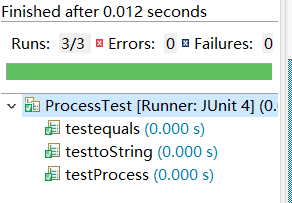


Process的equals标准为进程id相等则认为两个对象相等。

三者测试结果如图所示：







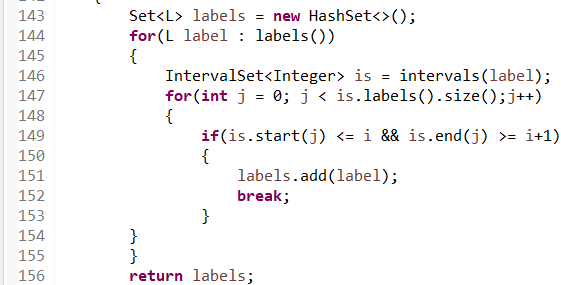
## 可复用API设计

为完成三个API的设计，为intervalset与multiintervalset接口扩展了一些通用方法并加以实现：



Getstart与getend的实现较为简单，遍历所有interval，找寻最大最小值即可。

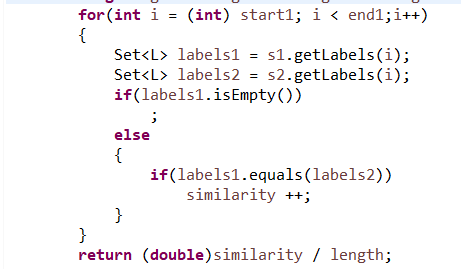
Getlabels实现关键代码：



遍历所有interval，start <= i && end >= i+1即覆盖了i,i+1时段，将标签加入labels集合中返回。基于上述通用方法，实现三个API较为简洁，如下：

### 计算相似度

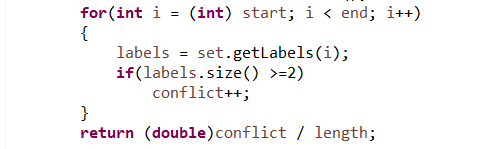
关键代码：



主要思想：对s1每一个时间单位进行遍历，若标签为空必定不相似，否则对比是否与2的标签相同，相同则similarity++，最后做除法即可。

### 计算时间冲突比例

关键代码：

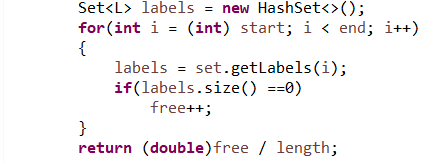


主要思想：

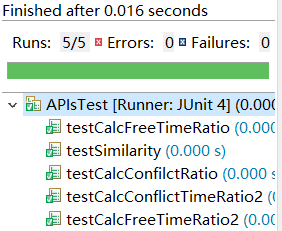
同样是遍历set的每一个时间单位，只要时间单位的标签集合大于等于2，则冲突了，增加冲突事件，最后做除法即可。

### 计算空闲时间比例

关键代码：



遍历set的每一个时间单位：，获取标签集合，集合个数为0才是空闲时间，free++，最后做除法即可。

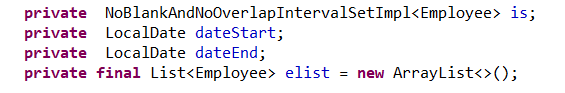
测试结果如图所示：

## 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

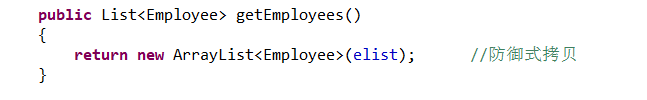
排班管理系统通过个性化API dutyRoster实现：



dutyRoster使用非空白非重叠的intervalset实现，由于要检查空白，应记录排班起始时间与终止时间，均通过localdate类存储。

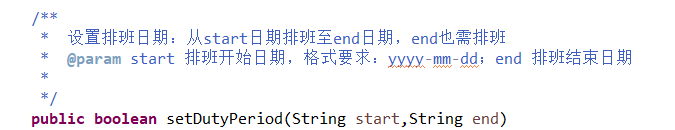
映射关系：将date类型映射为long，例如：2021-07-04~2021-07-21排班，起始日期映射为排班第一天，07-21映射为排班第18天，则intervalset的时间轴空白检查范围为1-19（18-19安排第18day）。

Elist存储员工信息，提供一个getter，实行防御式拷贝。

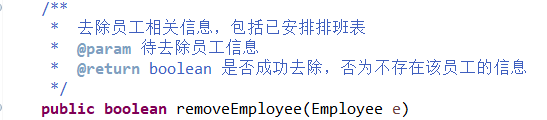


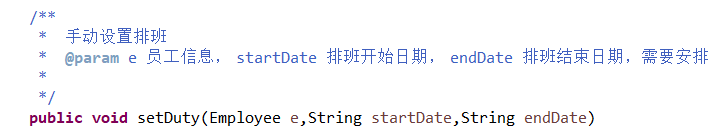
映射好的基础上，调用is的相关方法进行实现即可。

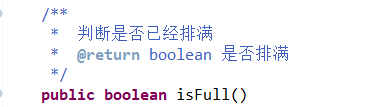
功能实现：



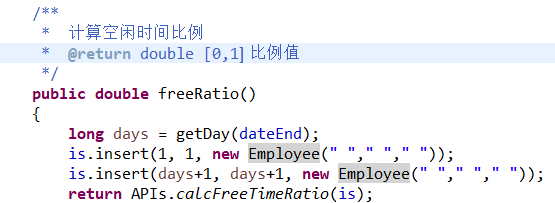








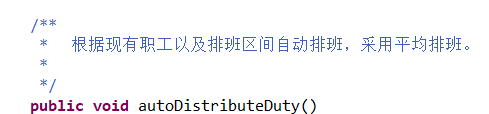
是否排满是有无空白的反逻辑。



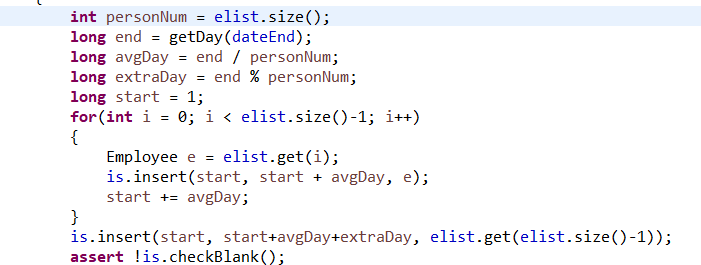
这里在测试用例暴露了一个问题，如下：



这样的测试用例可以看出是空闲了一天的，但空闲ratio结果是0，这是因为原逻辑是简单调用了calc接口，而calc接口的计算范围并不是安排的范围，而是实际插入的interval的最大范围，这里做了一个处理，无论如何都在末尾和起始插入一个无效标签的interval，固定范围，测试用例顺利通过。

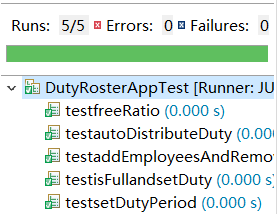


自动排班采取的业务逻辑是平均排班逻辑。

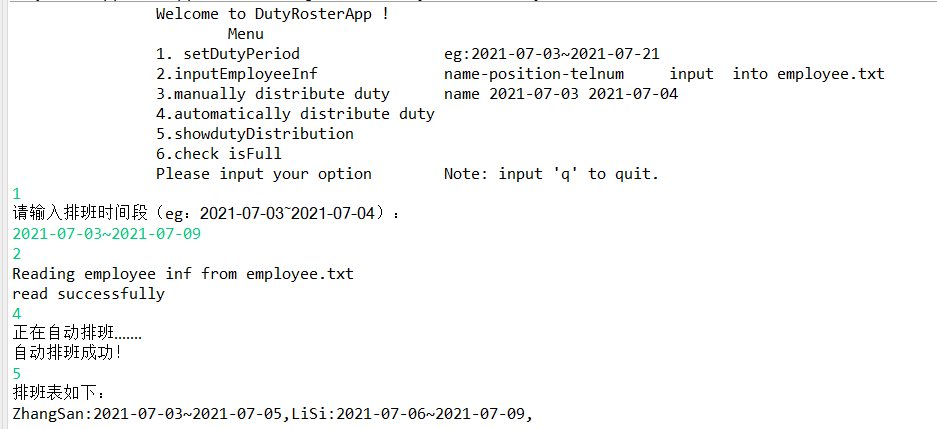


关键代码如图所示：最后要保证没有空白。

对该个性化类的测试结果如图所示：



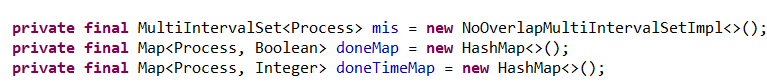
APP采用命令行实现，调用上述DutyRoster类的封装功能进行实现。

运行实例：

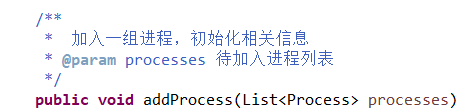
可以看到实现了基本功能

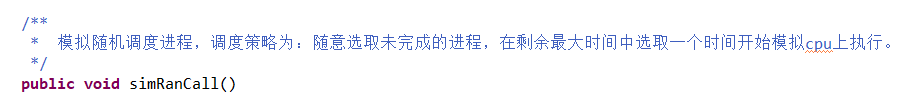
### 操作系统的进程调度管理系统

个性类通过ProcessSchedule实现：

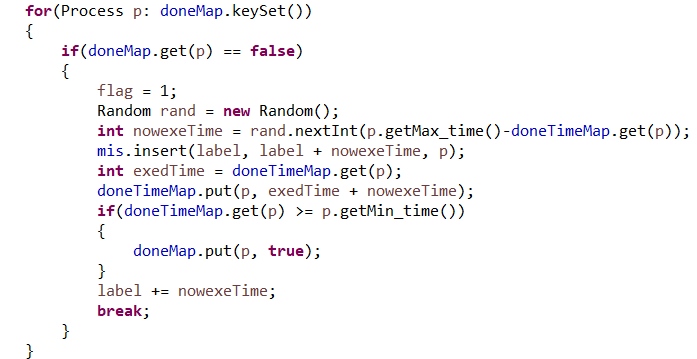


一组进程放置在两个map中，doneMap存放进程的是否完成信息，doneTimeMap存放进程已经执行时间信息，是NoOverlap的multiintervalset，映射到现实空间的一个进程安排

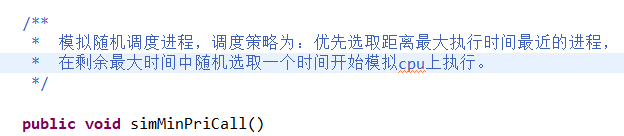




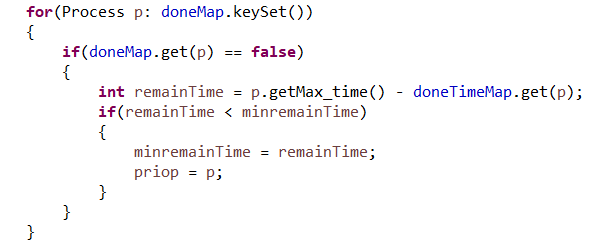
关键代码：



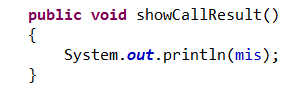
随机选取未执行进程，选取恰当的时间模拟执行，并作相应标记的更改，反复执行。



关键代码：



选取剩余时间最少的进程作为优先进程执行，执行过程与上述类似。



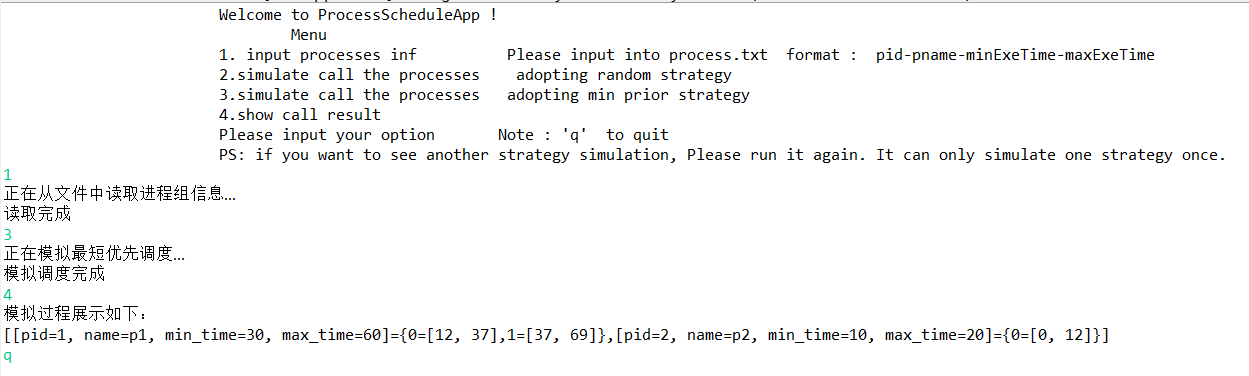
简单地输出mis即可看到模拟过程，由于是过程模拟，不做相应测试，在main函数中模拟通过。

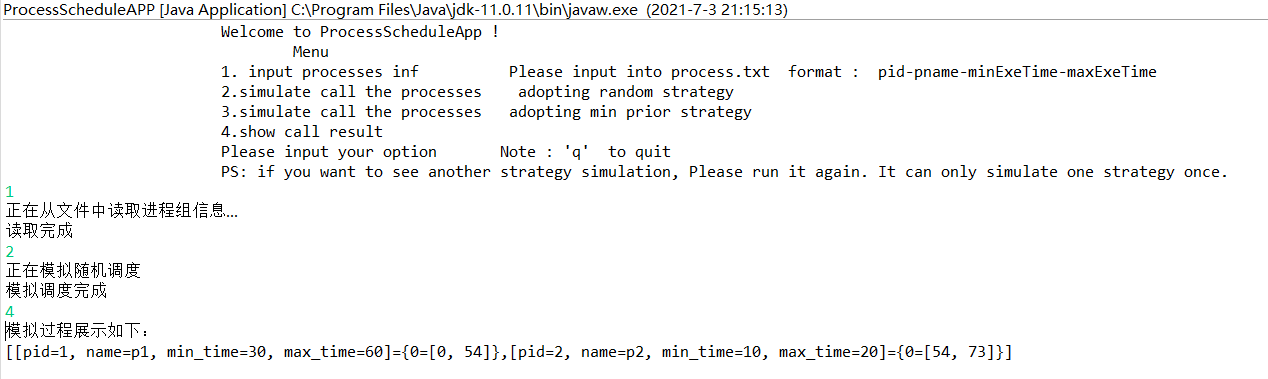


最短优先策略示意

下面基于该ADT实现APP命令行界面：

调用ProcessSchedule的相关功能，结合命令行实现，结果如图：



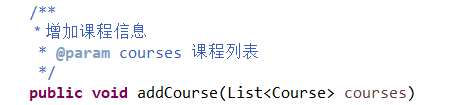


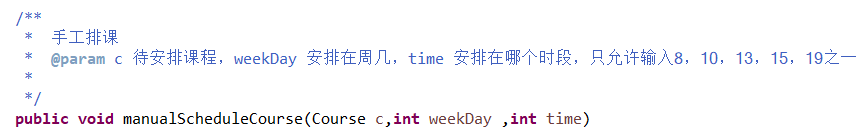
成功模拟了两种策略调度。

### 课表管理系统

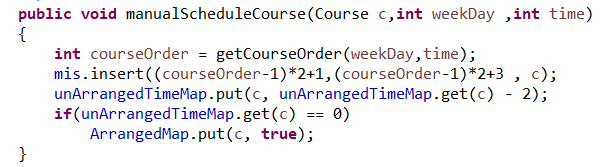
课表管理系统基于CourseSchedule个性化ADT:



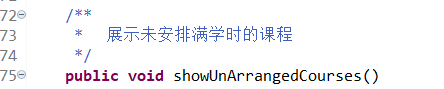


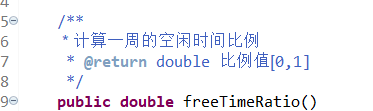


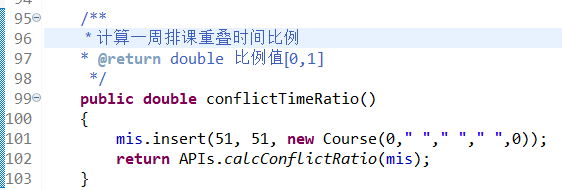
关键代码：



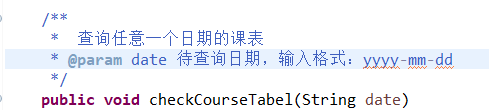
这里给出课程时间映射到时间轴的映射关系：一周有二十五个时段可以排课，先映射到安排在第几个时段，最后映射到安排在时间轴上的哪一个小时，例如：周一的8-10即安排为时间轴上的1-3小时（3只是起点），若安排满学时应作对应标记。



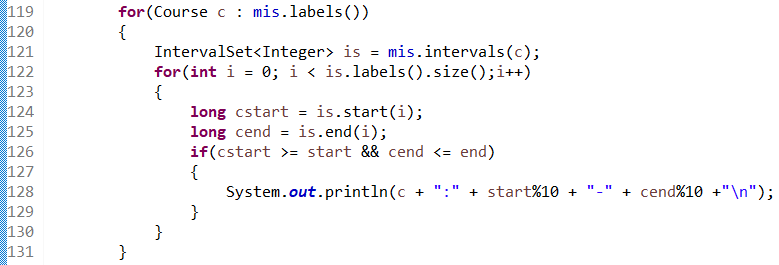




这里采用了前面阐述过的trick：API中只是单纯根据multiintervalset的长度来计算free与conflict的，这与实际应用不符，故在每周的起终点插入一个无效的课程，给出计算区间。

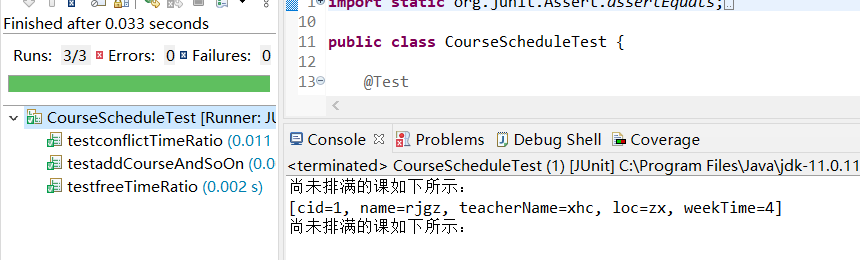


关键代码：



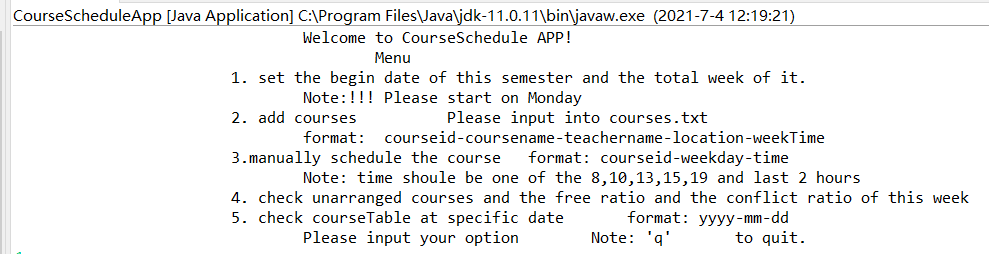
对所有排课时间段遍历即可，若在特定时间段内则输出到控制台

ADT测试结果如图所示：

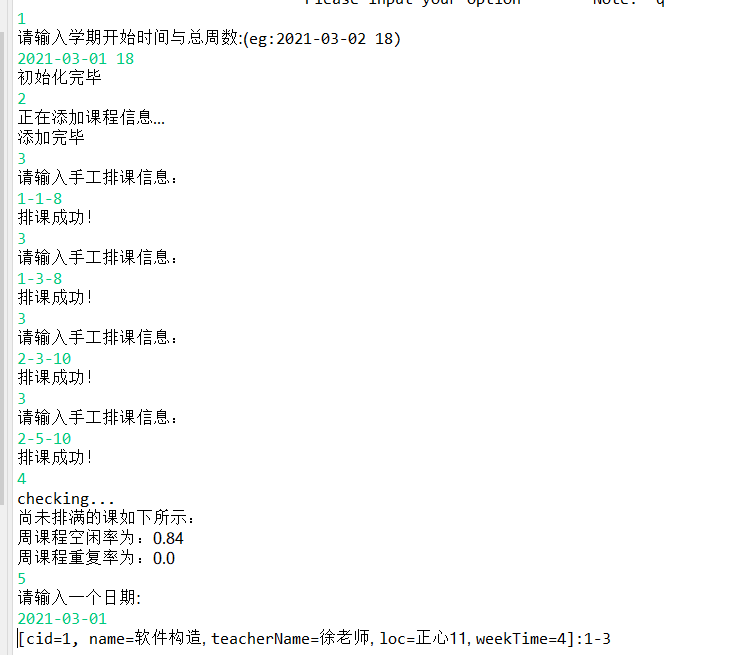


下面基于此ADT完成APP设计：  
APP调用CourseSchedule类进行简单的命令行实现，实现结果如下：

菜单：



排课APP：

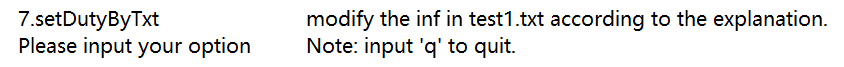


至此，完成了三个APP的全部设计与实现。

## 基于语法的数据读入

修改“排班管理”应用以扩展该功能。

在菜单中增加一个选项，如图所示：

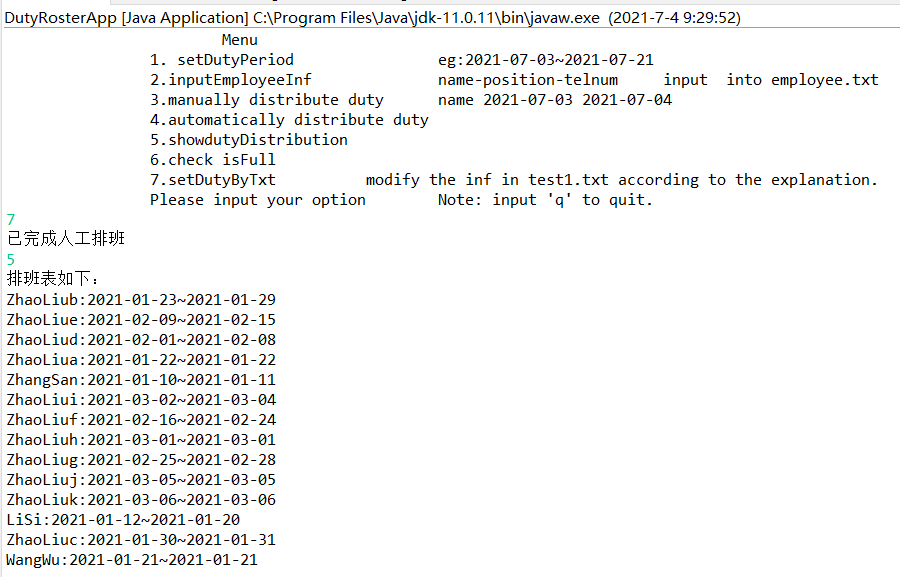


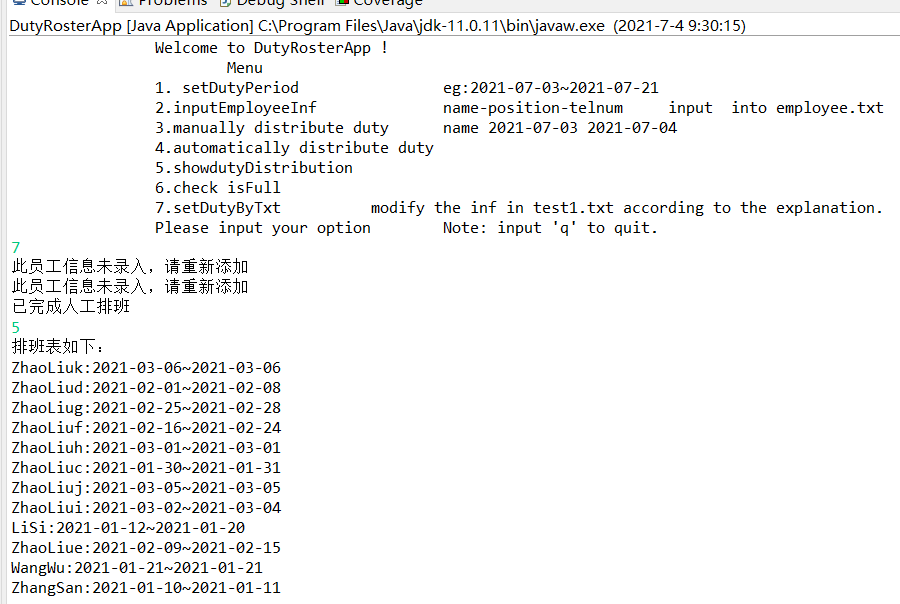
关键代码：



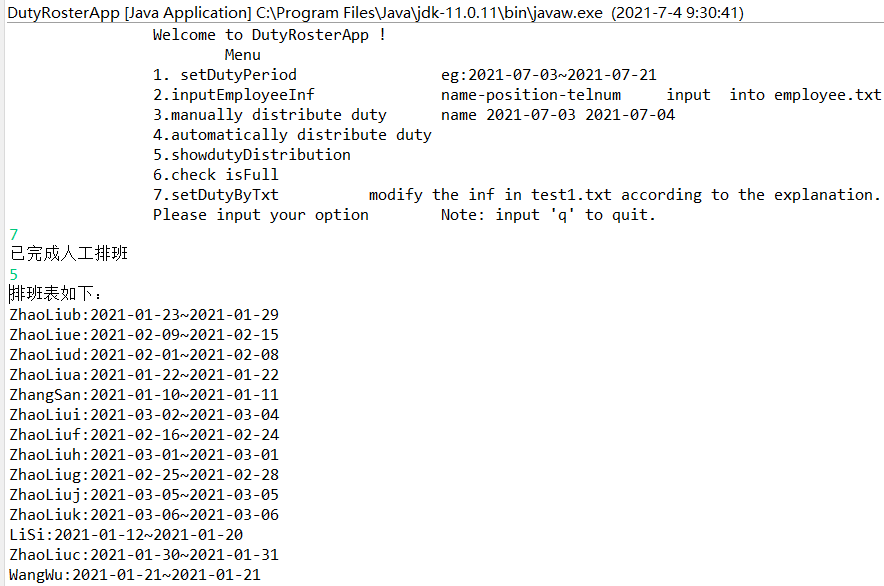
分类讨论，用正则表达式进行解析，依次提取员工列表，period，排班信息进行手动排班。

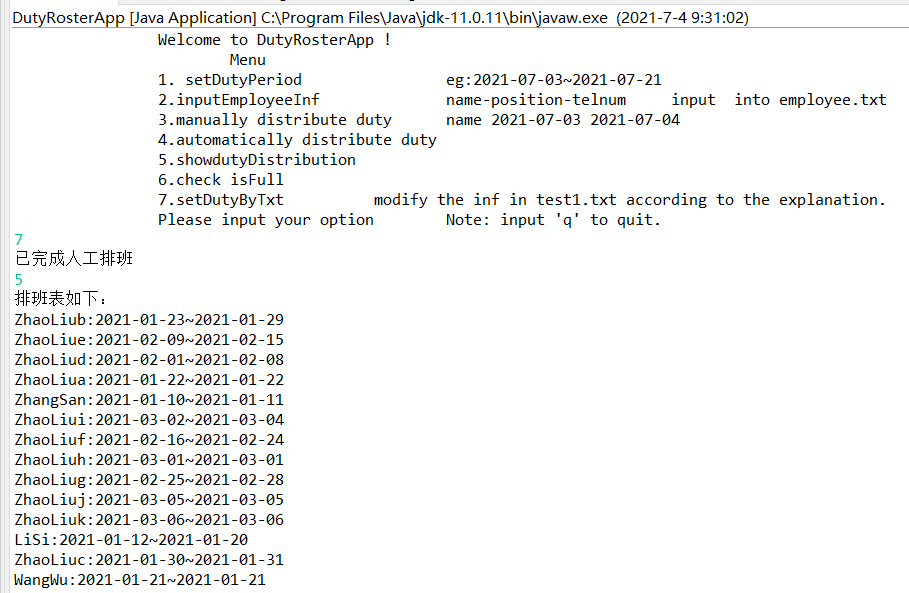
测试如下：

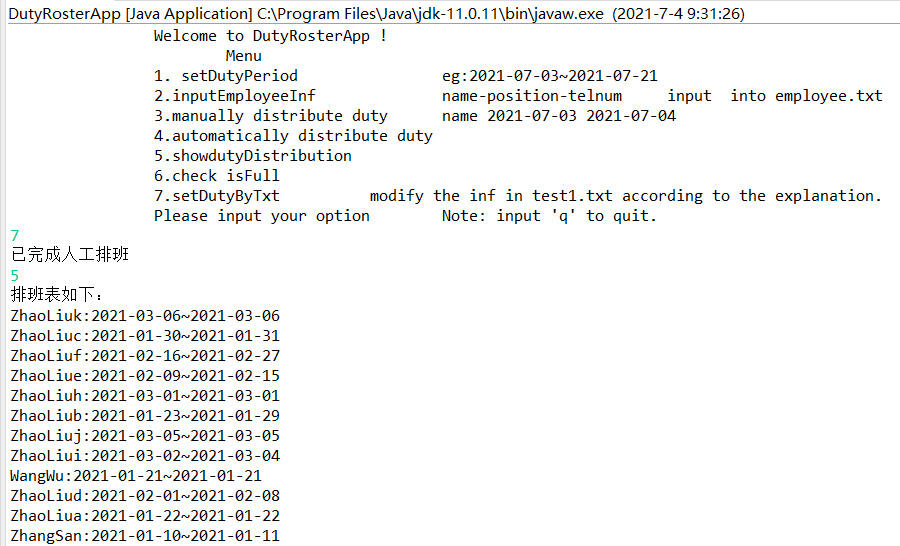


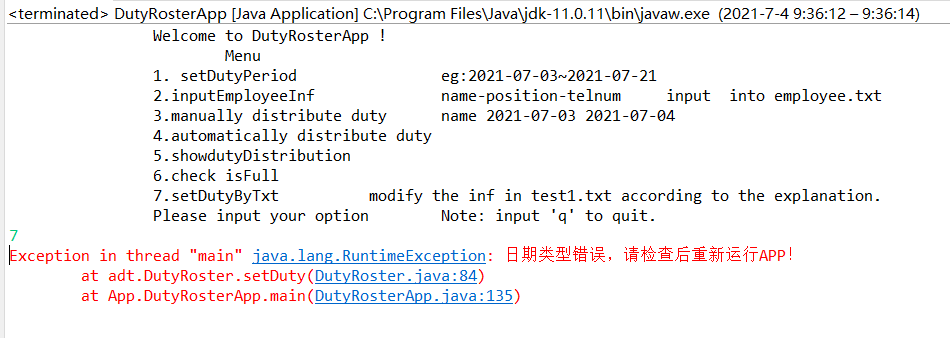


有两个员工信息录入错误。

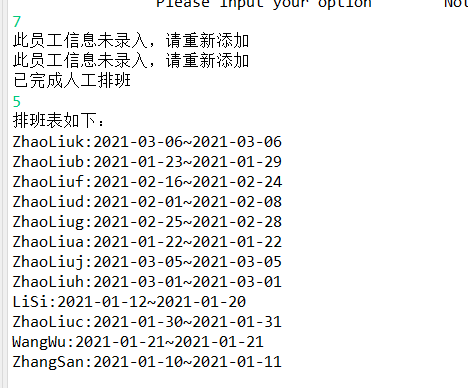


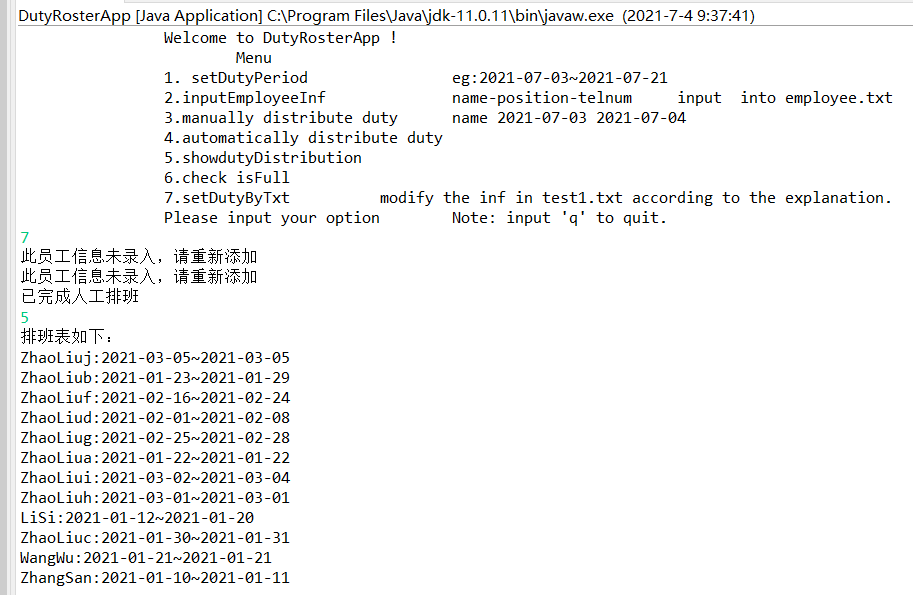






抛出运行时异常，提示日期格式输入错误。





至此，八个用例全部通过，或进行相应异常处理。

## 应对面临的新变化

### 变化1

评估之前的设计是否可应对变化、代价如何

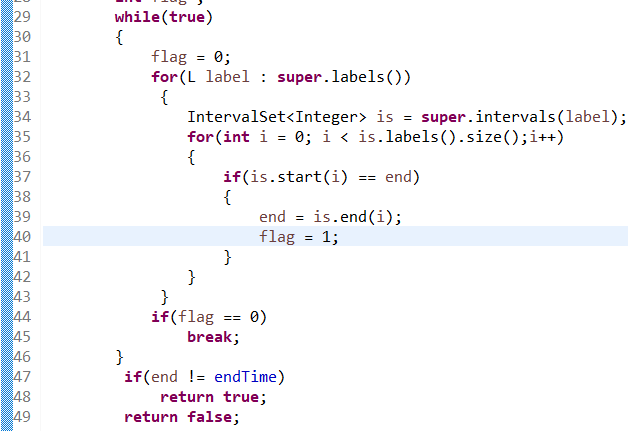
如何修改设计以应对变化

一个员工可被安排多段值班，但每一个时间段只安排一个员工值班，这意味着只是多标签问题，仍不留白、不可重叠，只需将原先底层的IntervalSet实现更改为MultiIntervalSet实现，去实现那两个特性接口即可，这是较为容易、代价小的。

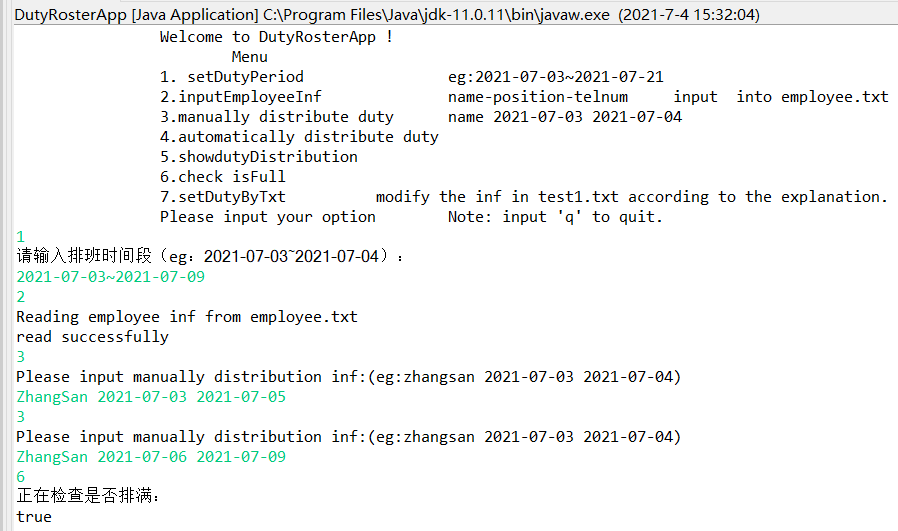
新设计一个实现类实现接口：



小幅度改动checkBlank代码：



排班APP成功实现一人被安排多段排班：



说明在两个维度下，继承树的维护开发代价还是可以接受的。

### 变化2

评估之前的设计是否可应对变化、代价如何

如何修改设计以应对变化

不允许同一个时间段内被排两门课：这意味着NoOverlap应被实现，采用NoOverlapMultiIntervalSet而不是简单的MultiIntervalSet实现，直接替换即可，这是在APP2内开发完成的类。

修改结果：



可以看到同时将软件构造与计算机系统排入周一第一节，但由于是NoOverlap的，故计算机系统不被排课，可以看到重复率永远为0



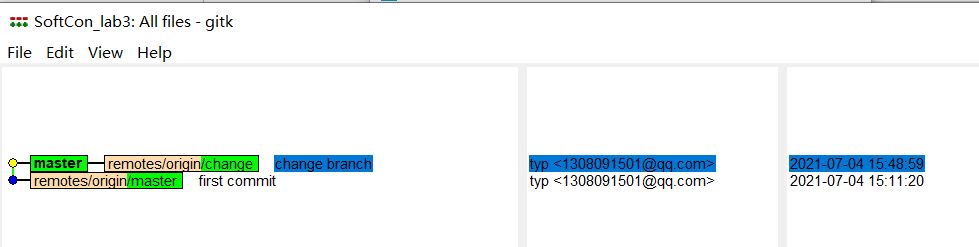
通过将个性化类的实现方式替换实现，修改代价一行代码

综上所述，采用继承树设计模式，对于维度较低的个性化特点的重新开发代价是比较小的，可以接受的，效果良好。

## 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。

如图所示：



# 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 |
| 2021.6.30 | 19：30-22：30 | 完成CommonIntervalSet的设计与实现 | 按时完成 |
| 2021.7.1 | 19：30-22：30 | 完成MultiIntervalSet的设计与实现；完成面向复用的三个ADT设计与实现 | 按时完成 |
| 2021.7.2 | 15：30-18：30 | 完成面向可复用性和可维护性的两个设计与实现 | 按时完成 |
| 2021.7.2 | 19：30-22：30 | 完成APIs的函数实现 | 按时完成 |
| 2021.7.3 | 9：30-12：30 | 排班管理系统App实现 | 按时完成 |
| 2021.7.3 | 14：30-17:30 | 进程调度APP实现 | 按时完成 |
| 2021.7.3 | 19：30-22：30 | 正则表达式功能拓展 | 按时完成 |
| 2021.7.4 | 9：30-12：30 | 课表管理APP实现 | 按时完成 |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 在导入txt文件进行正则表达式解析的时候，发现zhaoliue开始就无法排班 | 但跟进debug发现排班是有被执行的，因此推断可能是底层出了问题，但底层已经过test测试，且反复检查test并无问题，再观察zhaoliue开始进入了二月份，推测是日期映射关系出了问题，测试都是基于较近日期安排的。发现Period.getday()方法是以年月日为单位进行返回的，只是返回日这一部分，故重新更改映射函数，这次采用toEpochDay方法实现，解决bug！ |
| 初始intervalSet的实现是用集合做的，在后面设计到multiintervalset时，intervals方法需要排序，而且对multiintervalset时间段的遍历特别麻烦 | 与同学交流实现方式，换成了map实现，为设计ADT提供了较大的便利，进而能完成后续APP的开发 |
| 在设计API时基于原有的接口中的方法，设计代码量大，且思路复杂，遍历难度大 | 重新在接口中增加共性方法，实际上是为了便利API开发的，但或许在某些应用场景中也能使用。基于新增的三个接口，对于free以及conflict、similar的计算就显得格外简易且逻辑清晰。 |

# 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

经验：

1. 对于方法的抽象十分重要，要提高抽象能力，不断进行抽象。在进行三个API的设计时，发现只要在原有接口上增加三个通用方法，三个API的设计难度就大大降低。此外，在APP1中将排班日期映射到时间轴上的时候，这个映射在所有以String为日期参数的函数中都要使用，故将其抽象为getDay方法，恰好这个方法在本次实验中出现了bug，但花了两行代码的代价就更换了实现方法，感受到了抽象带来的代码的可维护性。

教训：

1. 在进行测试用例设计时候覆盖面一定要广，在本次实验中，有一个方法的测试用例出现了疏漏，导致在应用层暴露了bug，在调试的过程中非常非常麻烦。

## 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？

最大的差异就是代码的复用性：面向ADT的编程容易适应多个应用场景，代码的复用率高，只需花费极小的代价即可将ADT应用于新的应用场景，实现代码逻辑复用。此外，面向ADT的可维护性好，并且逻辑清晰：ADT每个方法都经过test检测，错误率较低，将底层的ADT的方法应用于上层应用，在实现应用的时候逻辑简单、结构清晰。

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？

Spec的撰写是必要的：这实现了调用者与开发人员之间的沟通，最大程度地降低了调用者的错误率，也为开发人员的实现提供了部分便利。而RI、rep exposure更多地能提醒开发人员检查正在开发的ADT是否注意到了这些可能会潜在导致bug的问题，在设计阶段就将这些做好，提高ADT的健壮性。AF也是为调用者提供便利，是对开发ADT从抽象空间到现实空间的一个映射。

在以后的编程中必须坚持这么做。

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？

难处是API作为提供给用户调用的函数接口，会面临大量的被调用

因此，在设计的过程中应考虑方方面面，在测试的过程中也应更加严谨。 但正因如此，带来的乐趣也是极大的，设计出一套功能完备的API较有

成就感。

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？

结构清晰：在编程时由于输入遵循一定的语法，对遵循语法的文本信息提取是较为容易的，且程序可以对不遵循语法的文本直接不处理并抛出相应异常，这极大地提高了程序的健壮性。语法就是用户与程序员之间的spec。同时作为用户，遵循相应的语法输入，在结构上也较为清晰明了。

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？

主要体现在最初架构上：Rep也是自己定义，在考虑Rep的实现开始就应该慎重考虑，不同的Rep会对后续不同的操作的实现难易程度有非常大的影响。其次，各类方法的参数也是自己定义，在设计时也应谨慎考虑参数的类型是什么更有利于调用。

在本次实验中遇到了Rep设计不当的困难：初始intervalSet的实现是用集合做的，在后面设计到multiintervalset时，intervals方法需要排序，而且对multiintervalset时间段的便利特别麻烦，由此，在与同学交流后，换成了map实现，为设计ADT提供了较大的便利，进而能完成后续APP的开发。

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？

经验：在本次lab中采用继承树的方式，通过继承与实现接口的方式继承公共特点，用接口实现个性化特征，框架较为清晰。但由于本次三个APP只用了两个维度的个性化特征实现，继承树能接受，若扩展到三维以上，开发的工作量以及继承树结构有些难以接受。

教训：为了实现最好地面向各种不同应用场景开发，应在设计模式上就花费精力去架构。尽量采用委托机制或者装饰器模式实现，好的架构对于后续不同应用场景的开发带来的好处是不可小觑的。

1. 关于本实验的工作量、难度、deadline

本实验工作量较大、难度较大、ddl适中。

1. 到目前为止你对《软件构造》课程的评价。

到目前为止，软件构造课程已经全部结束。从ADT的学习设计开始到最后的设计模式，无不体现着抽象和良好的设计模式对于开发大型工程以及针对各种相似应用场景开发所带来的极大便利。也不断加深着PPT上的那句话的理解：“切割石头的人，心中要想着大教堂”，通过本课程，收获颇丰。