

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 刘小川 |
| 学号 | 1190200817 |
| 班号 | 1936602 |
| 电子邮件 | shax2345@qq.com |
| 手机号码 | 15663748939 |

**目录**

[1 实验目标概述 1](#_Toc76066097)

[2 实验环境配置 1](#_Toc76066098)

[3 实验过程 1](#_Toc76066099)

[3.1 待开发的三个应用场景 1](#_Toc76066100)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 2](#_Toc76066101)

[3.2.1 IntervalSet<L>的共性操作 2](#_Toc76066102)

[3.2.2 局部共性特征的设计方案 3](#_Toc76066103)

[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） 4](#_Toc76066104)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 7](#_Toc76066105)

[3.3.1 MultiIntervalSet<L>的共性操作 7](#_Toc76066106)

[3.3.2 局部共性特征的设计方案 7](#_Toc76066107)

[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） 9](#_Toc76066108)

[3.4 面向复用的设计：L 11](#_Toc76066109)

[3.5 可复用API设计 13](#_Toc76066110)

[3.5.1 计算相似度 13](#_Toc76066111)

[3.5.2 计算时间冲突比例 14](#_Toc76066112)

[3.5.3 计算空闲时间比例 15](#_Toc76066113)

[3.6 应用设计与开发 15](#_Toc76066114)

[3.6.1 排班管理系统 15](#_Toc76066115)

[3.6.2 操作系统的进程调度管理系统 17](#_Toc76066116)

[3.6.3 课表管理系统 18](#_Toc76066117)

[3.7 基于语法的数据读入 18](#_Toc76066118)

[3.8 应对面临的新变化 20](#_Toc76066119)

[3.8.1 变化1 20](#_Toc76066120)

[3.8.2 变化2 21](#_Toc76066121)

[3.9 Git仓库结构 21](#_Toc76066122)

[4 实验进度记录 22](#_Toc76066123)

[5 实验过程中遇到的困难与解决途径 23](#_Toc76066124)

[6 实验过程中收获的经验、教训、感想 24](#_Toc76066125)

[6.1 实验过程中收获的经验和教训 24](#_Toc76066126)

[6.2 针对以下方面的感受 24](#_Toc76066127)

# 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

Eclipse和Git在上一次实验中已经配置好了。本次实验无需额外配置环境。

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

https://github.com/ComputerScienceHIT/HIT-Lab3-1190200817

# 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，一个单位有 n 个员工，在某个时间段内安排值班。每天只能安排一个员工且不能出现无人值班的情况；每个员工需要安排在连续的几天内。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，进程被调度在 CPU 上执行，操作系统决定在各个时段内执行哪个进程。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”；调度无规律，可看作是随机调度。

③大学课表管理：课程需要特定的教室和特定的教师。假设各周的课表都是完全一样的，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，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；同一个时间段内可以安排不同的课程；一位教师也可以承担课表中的多门课程。

相同之处：都包含了具有不同特征的“时间段集合”对象。每个时间段对应一个对象标签。

不同之处：有三个维度上的差异。①是否允许时间轴上有空白。在应用1中，不允许有空白；在应用2和应用3中，允许有空白。②是否允许不同的 interval 之间有重叠。在应用1和应用2中，不允许有重叠；在应用3中，允许有重叠。③是否包含周期性的时间段。应用 3 中以“一周”为单位重复某个课程，但应用1和应用2中不存在这种情况。

## 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

这个ADT描述了一组在时间轴上分布的“时间段”，每个时间段附着一个特定的标签，且标签不重复。因此共性的方法包括：

1. 静态工厂方法empty()：创建一个空对象。
2. void insert(long start, long end, L label)：在当前对象中插入新的时间段和标签。
3. Set<L> labels()：获得当前对象中的标签集合。
4. boolean remove(L label)：从当前对象中移除某个标签所关联的时间段。
5. long start (L label)：返回某个标签对应的时间段的开始时间。
6. long end (L label)：返回某个标签对应的时间段的结束时间。
7. IntervalSet<L> copy()：返回这个对象的副本。

### 局部共性特征的设计方案

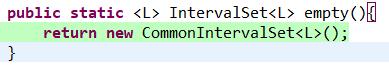
由于IntervalSet是一对一的结构，即一个标签对应一个时间段，因此可以使用Map作为内部的数据结构，定义为：

private final Map<L, time> intervalMap = new HashMap<>();

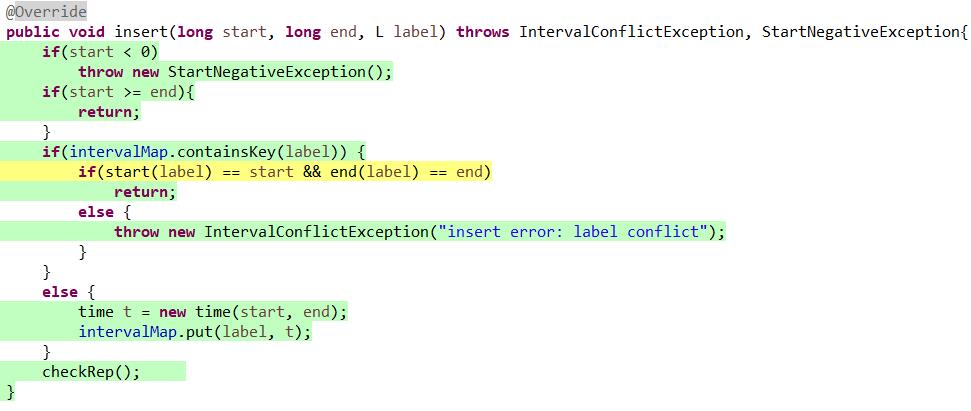
由于时间段是两个long型的整数构成的，因此可以定义一个辅助类time表示一个时间段。其中start是时间段的开始，end是时间段的结束，同时要求start一定大于0且start一定比end小。这里time实现了Comparable接口是为了满足下面MultiIntervalSet中的一些功能。



对于empty()方法，直接返回一个具体实现类。



对于insert()方法，首先判断非法条件（start<0和start>=end），然后判断Map中是否包含标签，若包含该标签，那只有在重复设置相同时间段时才合法，否则会抛出IntervalConflictException；若不包含该标签，就直接插入时间段。（注意，这里是允许不同标签之间存在Overlap的）



对于labels()方法，直接返回Map中所有的键构成的集合即可。

对于remove()方法，判断Map的键中是否存在该标签，若不存在，直接返回false；若存在，就在Map中删除该标签，并返回true。

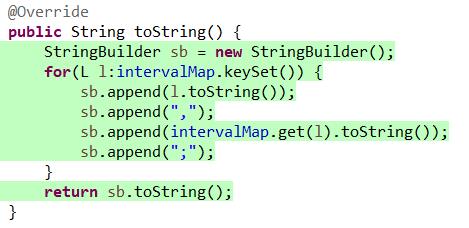
对于start()方法，判断Map的键中是否存在该标签，若不存在，直接返回-1，否则可以得到标签对应的时间段time，调用time的getStart()方法就可以得到时间段的开始时间并返回。

对于end()方法，同上，但要调用time的getEnd()方法得到时间段的结束时间并返回。

对于copy()方法，先新建一个空白的IntervalSet副本，然后遍历Map。将遍历得到的所有时间段和标签通过insert()方法插入到新的副本中，最后返回该副本。



此外，还需要重写toString方法提供容易阅读的信息。



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

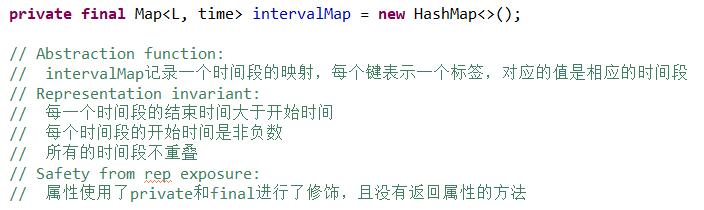
一、使用decorator装饰器的方法进行是否允许不同的 interval 之间有重叠以及周期性的维度的个性特征的设计。向装饰器中传入待装饰的IntervalSet，并在继承该装饰器的具体子类中实现相应的个性化功能。如下图，在装饰器IntervalSetDecorator中有属性intervalSet，是一个待装饰的IntervalSet类，而IntervalSetDecorator中的所有方法都调用intervalSet的相应方法。



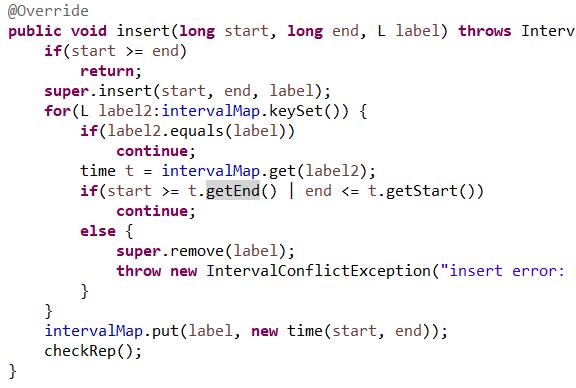
①不允许重叠的IntervalSet实现如下：



首先NoOverlapIntervalSet继承装饰器IntervalSetDecorator，并实现IntervalSet接口。在NoOverlapIntervalSet中添加一个属性intervalMap记录添加的所有时间段。



在重写的insert()方法中，首先判断新添加的时间段是否与已添加的时间段发生重叠，如果有，直接抛出异常；否则就调用未重写的父类的insert()方法进行插入。



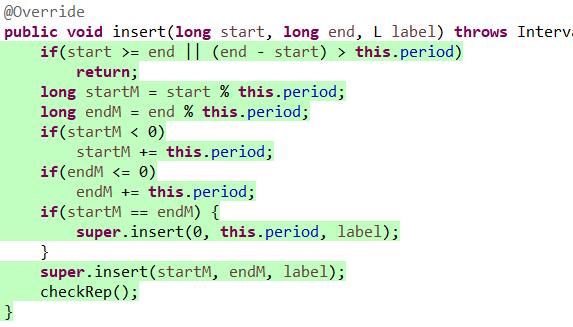
②周期性的IntervalSet实现如下：



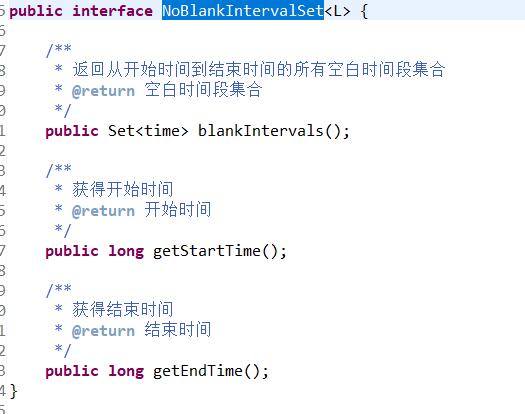
首先PeriodicIntervalSet继承装饰器IntervalSetDecorator，并实现IntervalSet接口。在PeriodicIntervalSet中添加一个属性period记录时间周期。



在重写的insert()方法中，将开始时间和结束时间对周期取模后再调用未重写的父类的insert()方法进行插入。

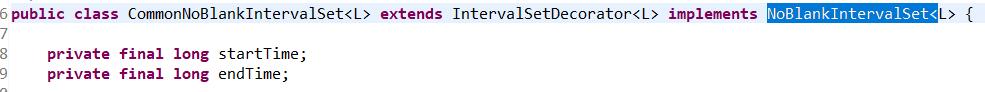


二、使用代理的方式进行是否允许时间轴上有空白这个维度的个性特征的设计。由于不允许空白相当于添加了新的方法，因此将这个特性抽象为一个接口NoBlankIntervalSet：

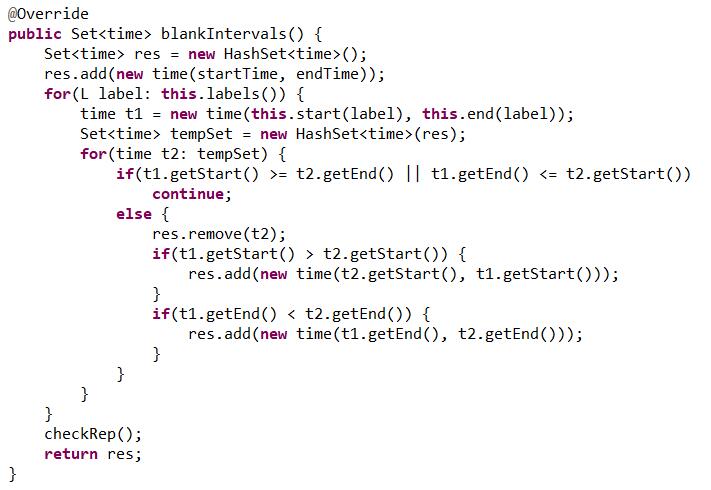


接口中包含三个方法：blankIntervals()返回所有的空白时间段集合；getStart-Time()返回总的开始时间；getEndTime()返回总的结束时间。

一个具体的不允许时间轴上有空白的IntervalSet只需要实现NoBlank-IntervalSet接口。具体实现类CommonNoBlankIntervalSet的实现如下：



属性startTime是总的开始时间，endTime是总的结束时间。blankIntervals()的实现如图，首先将空白时间段设置为整个时间段，然后遍历所有的时间段，将这些时间段从空白时间段中去除，最后得到的就是所有的空白时间段。



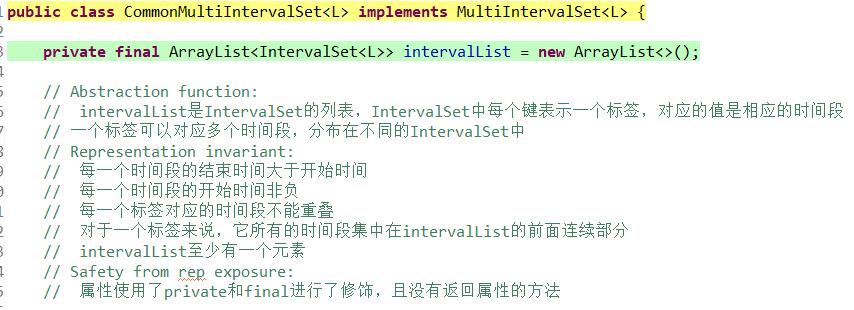
## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

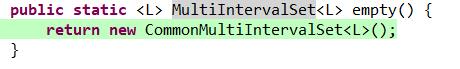
1. 静态工厂方法empty()：创建一个空对象。
2. MultiIntervalSet(IntervalSet<L> initial)：利用initial中包含的数据创建非空对象。
3. void insert(long start, long end, L label)：在当前对象中插入新的时间段和标签。
4. Set<L> labels()：获得当前对象中的标签集合。
5. boolean remove(L label)：从当前对象中移除某个标签所关联的所有时间段。
6. IntervalSet<Integer> intervals(L label)：从当前对象中获取与某个标签所关联的所有时间段。

### 局部共性特征的设计方案

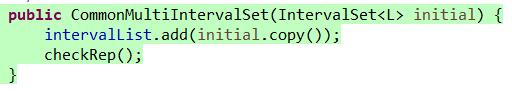
由于要求必须使用 IntervalSet<L>作为其 rep 的一部分，因此选择IntervalSet<L>组成的List作为rep。一个IntervalSet中只存储某个标签对应的一个时间段，如果一个标签对应多个时间段，需要分散在不同的IntervalSet中。



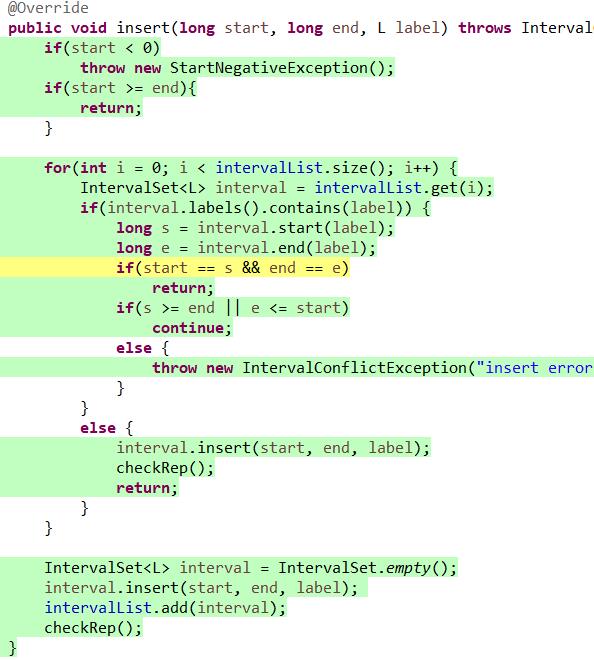
对于empty()方法，直接返回一个具体实现类。



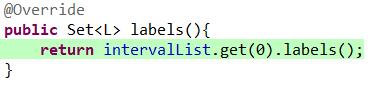
对于initial初始化方法，直接将传入的IntervalSet的副本作为rep的一个元素。



对于insert()方法，首先判断非法情况。之后遍历IntervalSet列表得到标签对应的所有时间段，判断已存在的时间段和要增加的时间段是否存在重叠，如果重叠则抛出异常；若不重叠，就寻找某个不存在该标签的IntervalSet，将这个时间段插入该IntervalSet，如果列表中所有的IntervalSet都包含该标签，就需要新建一个空白的IntervaSet加入列表，再进行插入。

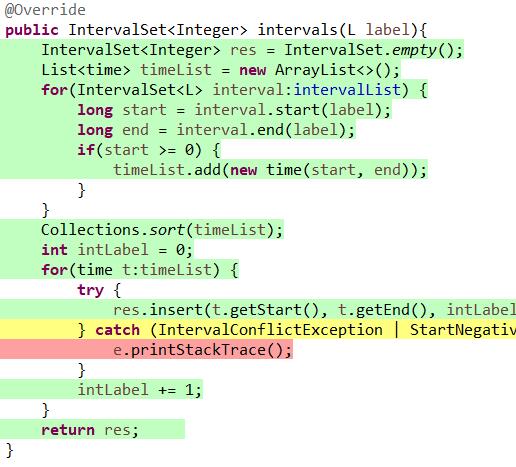


对于labels()方法，可以直接返回第一个IntervalSet中的所有标签组成的集合。（因为插入时都是从头开始遍历的，因此不会存在某个标签出现在后面的IntervalSet而不在第一个IntervalSet中的情况）

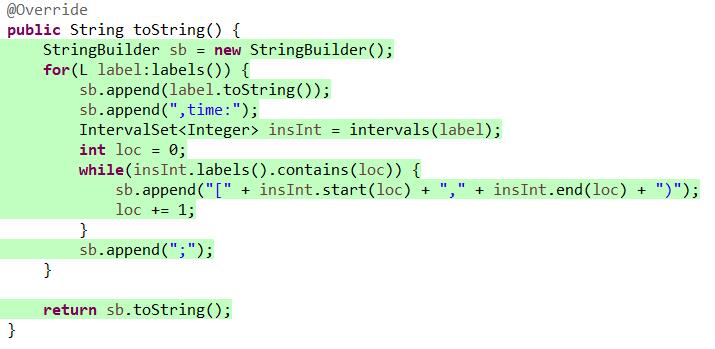


对于remove()方法，直接对每个IntervalSet调用remove()方法即可。

对于intervals()方法，遍历IntervalSet列表得到标签对应的所有时间段，将时间段从小到大进行排序（前面time的实现中进行了说明）。然后将每个时间段以它的顺序作为标签插入到一个IntervalSet中并返回。



此外，还需要重写toString方法提供容易阅读的信息。



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

与3.2.3同理，使用decorator装饰器的方法进行是否允许不同的interval之间有重叠以及周期性的维度的个性特征的设计。向装饰器中传入待装饰的MultiIntervalSet，并在继承该装饰器的具体子类中实现相应的个性化功能。如下图，在装饰器MultiIntervalSetDecorator中有属性multiIntervalSet，是一个待装饰的MultiIntervalSet类，而MultiIntervalSetDecorator中的所有方法都调用multiIntervalSet的相应方法。

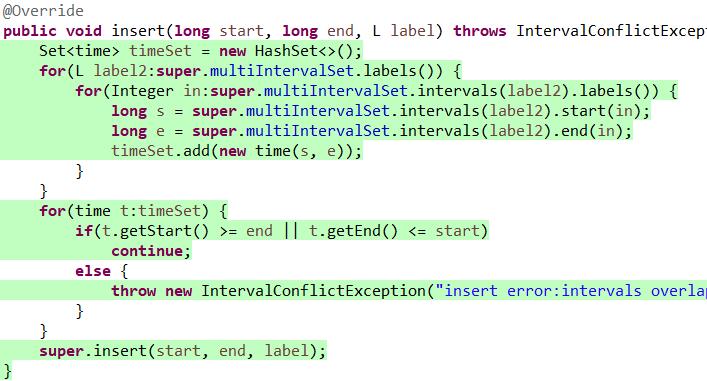


①不允许重叠的MultiIntervalSet实现如下：



首先NoOverlapMultiIntervalSet继承装饰器MultiIntervalSetDecorator，并实现MultiIntervalSet接口。

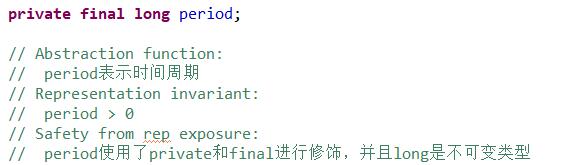
在重写的insert()方法中，首先判断新添加的时间段是否与已添加的时间段发生重叠，如果有，直接抛出异常；否则就调用未重写的父类的insert()方法进行插入。



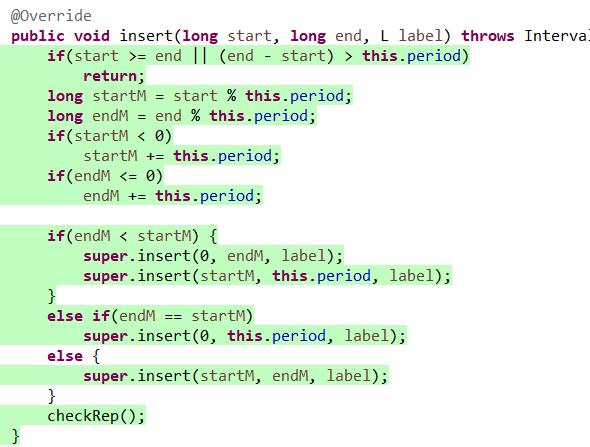
②周期性的MultiIntervalSet实现如下：



首先PeriodicMultiIntervalSet继承装饰器MultiIntervalSetDecorator，并实现MultiIntervalSet接口。在PeriodicMultiIntervalSet中添加一个属性period记录时间周期。



在重写的insert()方法中，将开始时间和结束时间对周期取模后再调用未重写的父类的insert()方法进行插入。

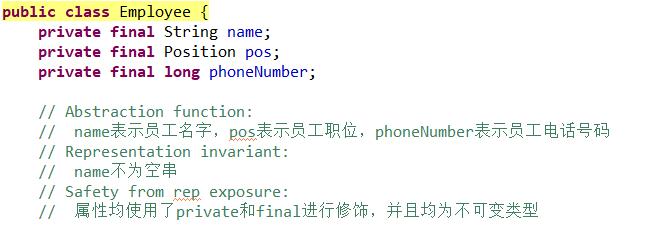


## 面向复用的设计：L

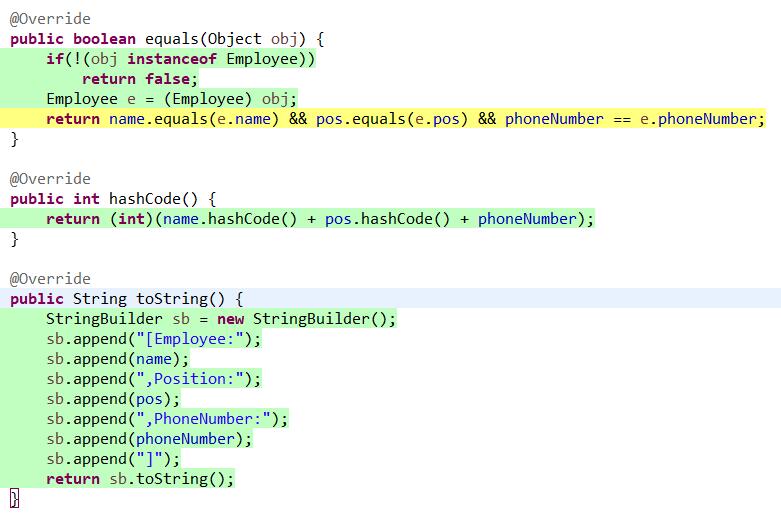
设计三个应用的不同标签，分别为“员工”（Employee）、“进程”（Process）、

“课程”（Course）。并且它们都是immutable类。

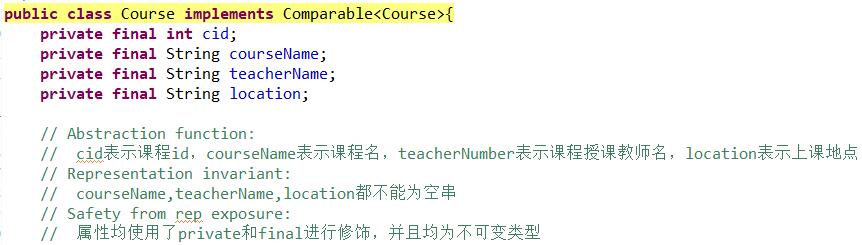
①对于Employee，具有的属性为：姓名、职务、手机号码。



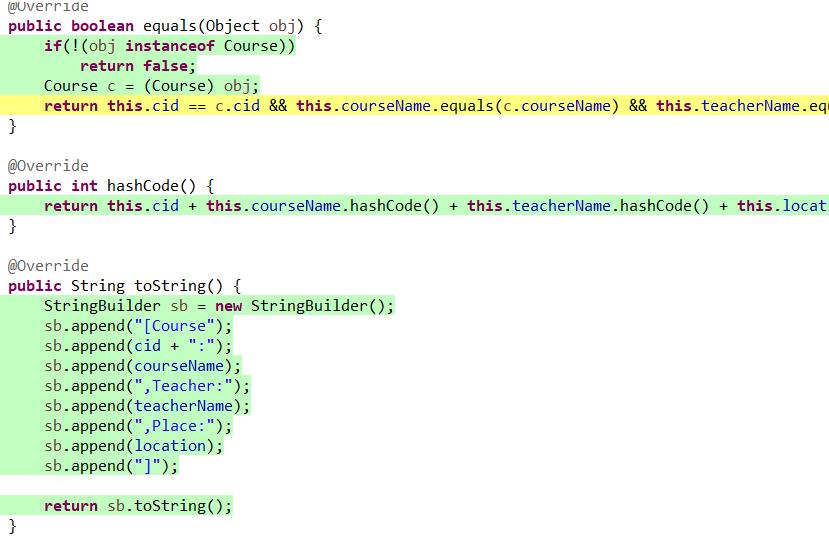
除了相应的get方法之外，还需要重写equals()、hashCode()和toString()方法。equals()的判断依据是：只有三个属性均相同时才认为是相同的。



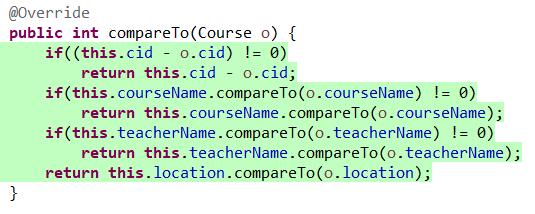
②对于Course，具有的属性为：课程 ID、课程名称、教师名字、地点。



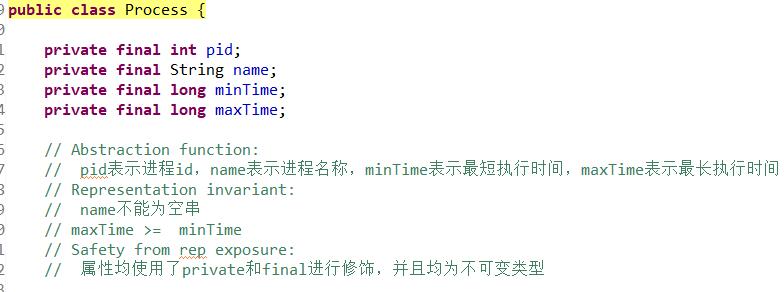
除了相应的get方法之外，还需要重写equals()、hashCode()和toString()方法。equals()的判断依据是：只有四个属性均相同时才认为是相同的。



此外，Course类实现了Comparable接口，这是为了APP中显示课程顺序的合理性。



③对于Process，具有的属性为：进程 ID、进程名称、最短执行时间、最长执行时间。



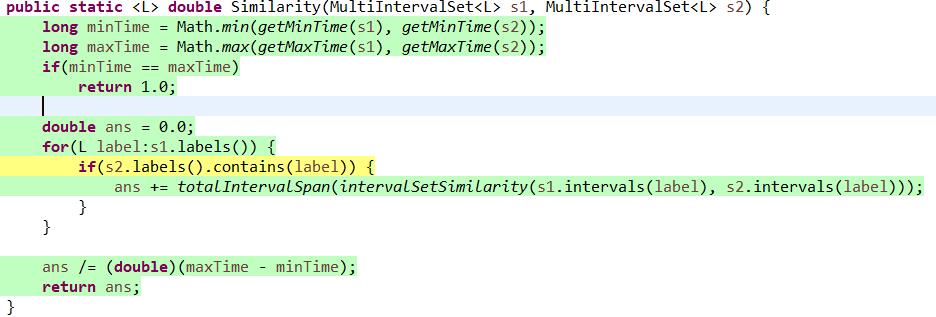
除了相应的get方法之外，还需要重写equals()、hashCode()和toString()方法。equals()的判断依据是：进程ID相同时认为是相同的。



## 可复用API设计

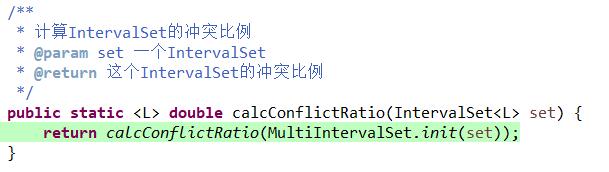
### 计算相似度

对于两个MultiIntervalSet：s1和s2，遍历s1中的标签，查看s2中是否存在相同的标签，如果不存在，则对相似度没有贡献；如果存在，那么这个标签在s1和s2中各有一个时间段的集合，计算这两个时间段集合的重合长度，将所有的重合长度加在一起除以MultiIntervalSet的时间跨度就是两个MultiIntervalSet的相似度。



### 计算时间冲突比例

对于IntervalSet来说，由于它是一个特殊的MultiIntervalSet，因此可以把它转换成MultiIntervalSet后再调用针对MultiIntervalSet的计算时间冲突比例的函数。

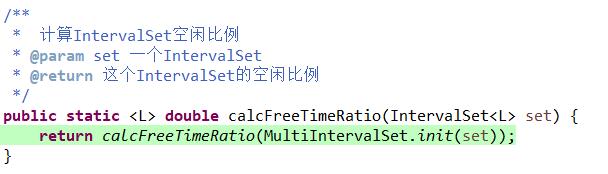


对于MultiIntervalSet，维护一个冲突时间段的集合conflictTime，初始时为空。遍历set中的标签，判断该标签和其他标签的时间段是否存在重合，如果存在，就将冲突的时间段加入conflictTime。向conflictTime加入时间段也需要考虑重合的问题，集合中不能有重合的时间段，因此向conflictTime加入时间段时需要进行适当的合并。最后conflictTime中的时间长度除以总的时间跨度就是时间冲突比例。



### 计算空闲时间比例

对于IntervalSet来说，由于它是一个特殊的MultiIntervalSet，因此可以把它转换成MultiIntervalSet后再调用针对MultiIntervalSet的计算空闲时间比例的函数。



对于MultiIntervalSet，原理类似于3.2.3中的blankIntervals()方法。维护一个空闲时间段的集合freeTime，初始时为整个时间段。遍历set中的所有时间段，并将其从freeTime中删去。最后freeTime中的时间长度除以总的时间跨度就是空闲比例。

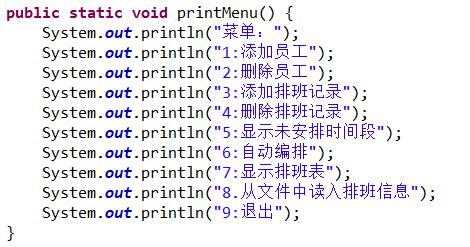


## 应用设计与开发

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

使用DutyIntervalSet作为数据结构，同时维护一个Employee的集合可以存储未被安排的员工。刚进入时APP，会提示初始化一些信息，包括：排班开始日期、结束日期以及一组员工信息。初始化结束后，打印一个菜单，告诉用户提供的一些功能。



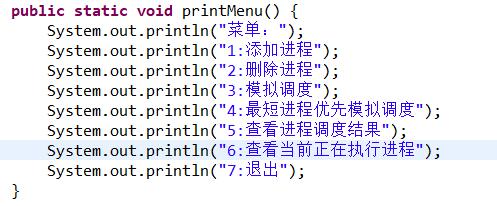
根据用户选择的功能采取相应的操作。其中选项1,2涉及对Employee集合的增删；3对应DutyIntervalSet的insert操作；4对应DutyIntervalSet的remove操作；5对应DutyIntervalSet的blankIntervals操作；7对应DutyIntervalSet的blankIntervals操作以及DutyIntervalSet的labels、start、end操作；8的操作见3.7节。由于这些操作都是简单地使用一些函数，因此不再详细叙述。这里只介绍“6：自动编排”的实现方法：首先调用DutyIntervalSet的blankIntervals操作得到未被安排的时间段，然后遍历Employee集合，找出未被安排的员工，将这些未被安排的时间段和员工一对一匹配起来，并调用insert方法插入到DutyIntervalSet中。

此外，APP还拥有很好的健壮性，能面对用户各种非法的、不符合格式的输入。举例来说，针对添加员工的操作，存在各种非法的输入情况，在APP中都得到了相应的解决，并提示给用户。



### 操作系统的进程调度管理系统

使用ProcessIntervalSet作为数据结构，同时维护一个的Map存储进程和已执行时间的映射关系。进入APP后，会打印一个菜单，告诉用户提供的一些功能。



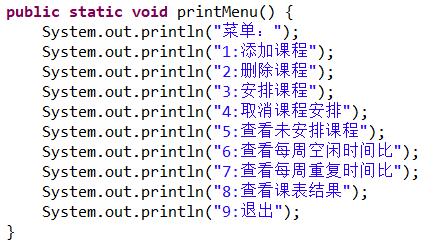
根据用户选择的功能采取相应的操作。其中选项1,2涉及对Map的增删；5，6对应ProcessIntervalSet的intervals操作。由于这些操作都是简单地使用一些函数，因此不再详细叙述。这里介绍3,4的实现方法，选项3：从时间点0开始进入一个循环，当所有的进程都被执行完成后退出循环。在循环中，首先使用随机数（random.nextBoolean）决定是否调度进程，如果决定不调度进程，则闲置一段随机的时间（random.nextInt）；如果决定调度进程，则随机选择一个未完成的进程（使用随机数选择进程的序号）并执行一段随机的时间（random.nextInt），执行结束后，如果该进程的总执行时间已经落到最短执行时间和最长执行时间的区间内，则该进程被执行完成，然后从这个时间点开始进行下一轮的循环。选项4：和选项3唯一的不同在于进程的选择不是随机的，而是选择距离其最大执行时间差距最小的进程。



同样地，APP拥有很好的健壮性，能面对用户各种非法的、不符合格式的输入。

### 课表管理系统

使用CourseIntervalSet作为数据结构，同时维护一个Course的Map存储未被安排的课程以及对应的剩余学时数。刚进入时APP，会提示初始化一些信息，包括：学期开始日期、总周数、以及一组课程信息。初始化结束后，打印一个菜单，告诉用户提供的一些功能。



根据用户选择的功能采取相应的操作。其中选项1,2涉及对Course映射的增删；3对应CourseIntervalSet的insert操作；4对应CourseIntervalSet的remove操作；5对应对Course映射的遍历以及显示；6,7对应API中操作的使用；8对应CourseIntervalSet的intervals操作。由于这些操作都是简单地使用一些函数，因此不再详细叙述。

同样地，APP拥有很好的健壮性，能面对用户各种非法的、不符合格式的输入。

## 基于语法的数据读入

首先对文件的格式进行分析，文件总共包括三个部分，分别是Employee、Period和Roster，每个部分有自己独特的格式。在假设没有空格、缩进和空行的情况下，可以分别设计识别每个部分的正则表达式。

Employee部分的正则表达式：



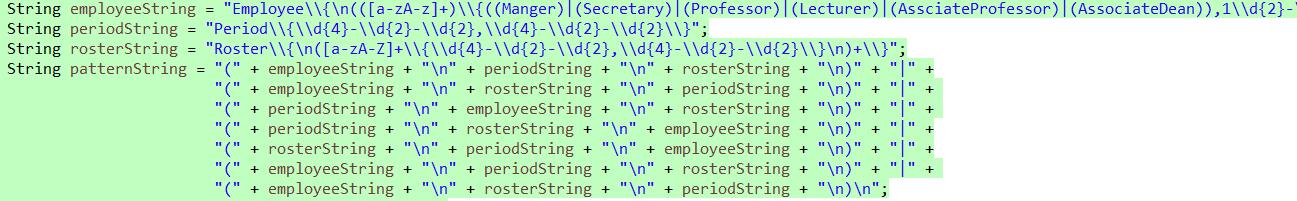
Period部分的正则表达式：



Roster部分的正则表达式：

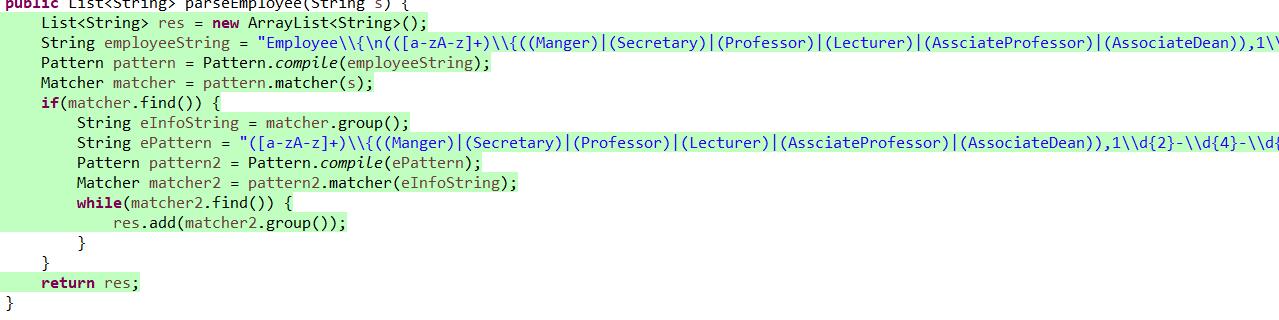


对于整个文件，由于三个部分出现的顺序是不定的，即共有六种情况，因此识别整个文件的正则表达式为：

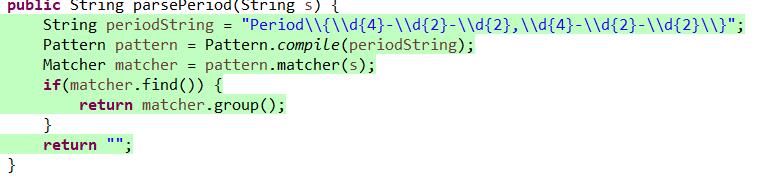


首先利用这个正则表达式判断文件的格式是否正确，如果正确就抽取出每个部分。

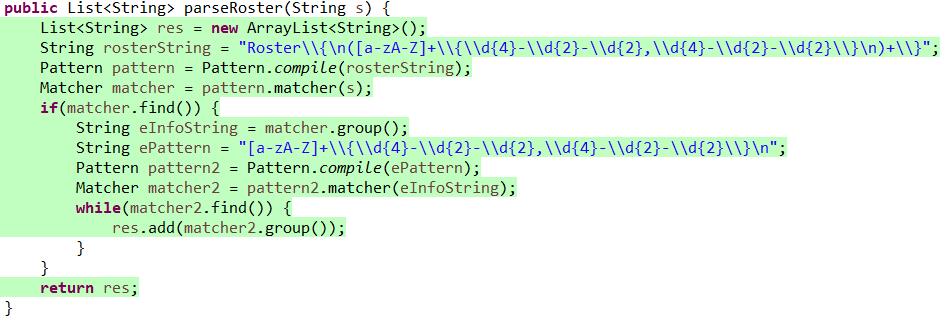
利用Employee内部的格式抽取出员工信息：



利用Period的格式抽取出时间段信息：



利用Roster内部的格式抽取出排班信息：



根据抽取出的信息，就可以构造出一个排班表了，在构造的同时判断一些错误，如：员工信息重复、员工未定义、时间重叠等。

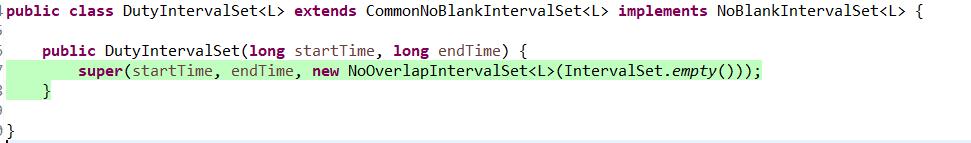
在APP中，读入用户指定的文件，并去除所有的空格、缩进、空行，之后调用Parser解析信息，在出现错误时提示给用户相应错误信息。

## 应对面临的新变化

### 变化1

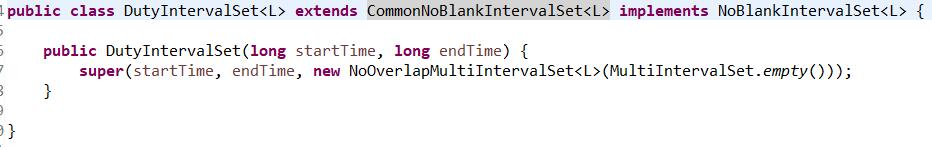
修改之前的DutyIntervalSet实现的是IntervalSet接口，但是本次变化要求每个标签可以对应多个时间段，因此要求DutyIntervalSet实现MulitIntervalSet接口，同时还要保持不允许重叠的特征。因此只需要修改DutyIntervalSet继承的装饰器类型以及实现的接口类型即可。具体修改如图：

修改前的代码：



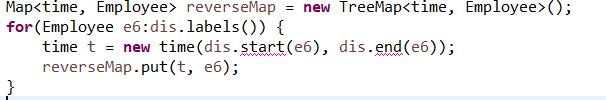


修改后的代码：

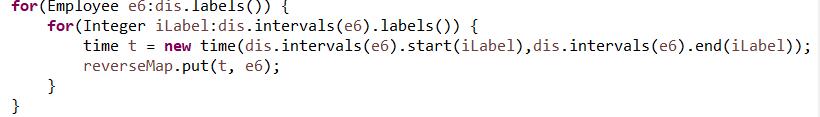




由于DutyIntervalSet从实现IntervalSet接口变为实现MulitIntervalSet接口，因此一些方法会发生变化，例如不再支持start()和end()方法，同时新增加了intervals()方法。因此，在DutyRosterApp需要修改相应的实现方法。举例来说，对于APP中的可视化排班信息的功能，需要遍历DutyIntervalSet中的信息，之前的遍历方式直接使用labels()方法：



但是，修改之后需要使用intervals()方法：



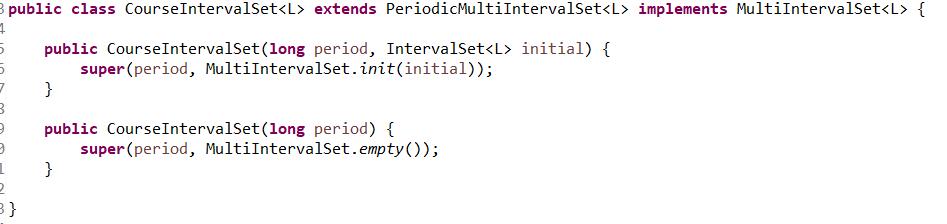
在其他地方也涉及这种变化，就不一一列举了。同时，需要修改测试代码保持测试的正确性。

除修改测试代码的变化，一共修改大约50行代码，花费时间较短，说明之前的设计较为合理，应对变化的能力比较强，可维护性很好。测试代码的修改大约也在50行左右。

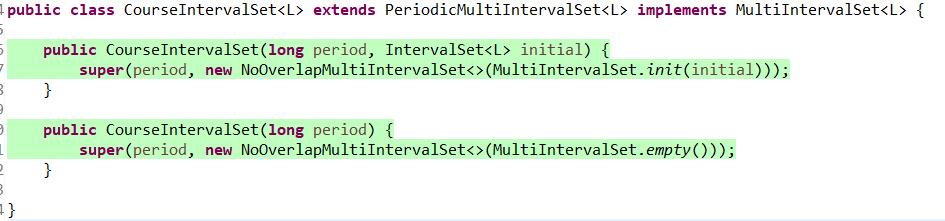
### 变化2

由于使用了装饰器，只需要将原来传入装饰器的MultiIntervalSet改为不能重叠的NonOverlapMultiIntervalSet即可。如果不考虑测试代码的话，真正修改的代码只有一行！说明之前的设计较为合理，应对变化的能力比较强，可维护性很好。具体修改如图：

修改前的代码（传入可重叠的MultiIntervalSet）：



修改后的代码（传入不可重叠的NoOverlapMultiIntervalSet）：

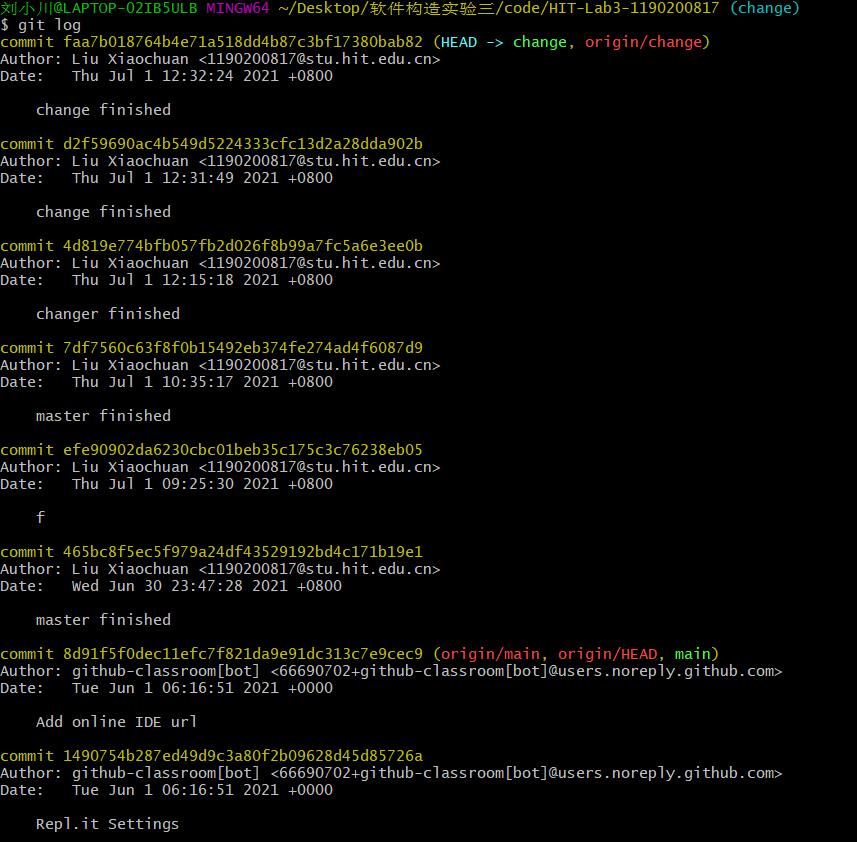
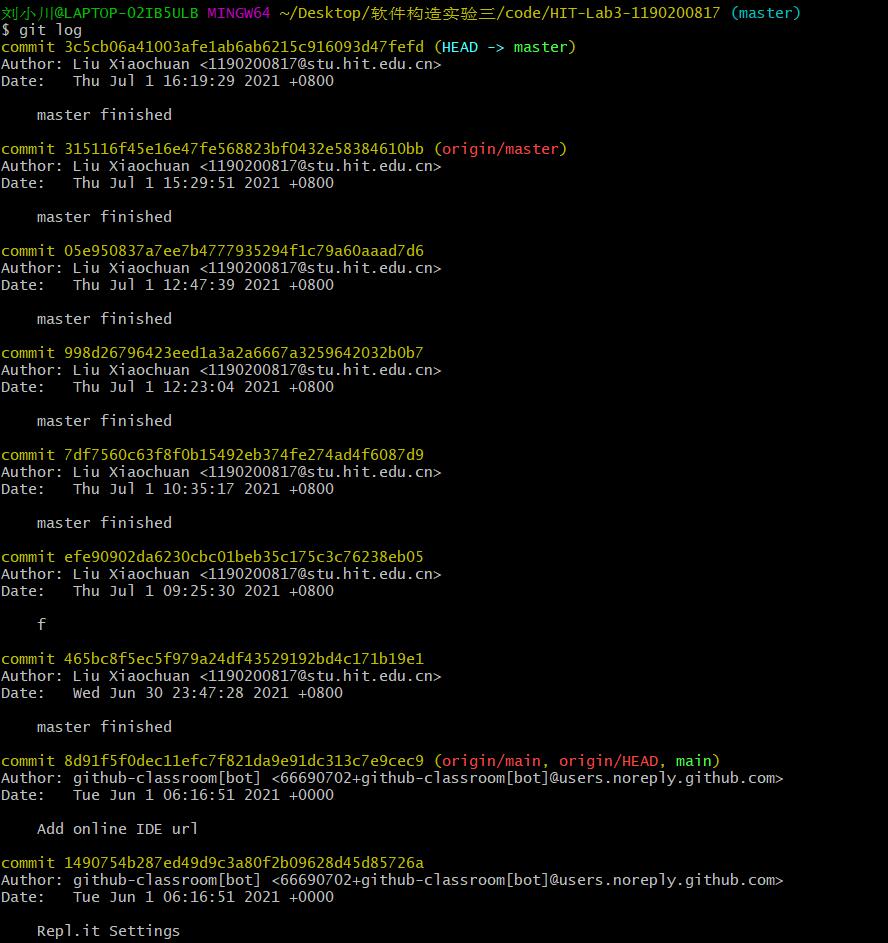


为了使得测试仍然保持正确，需要修改原来的测试代码，总共修改的代码量大约为50行，花费时间较短。

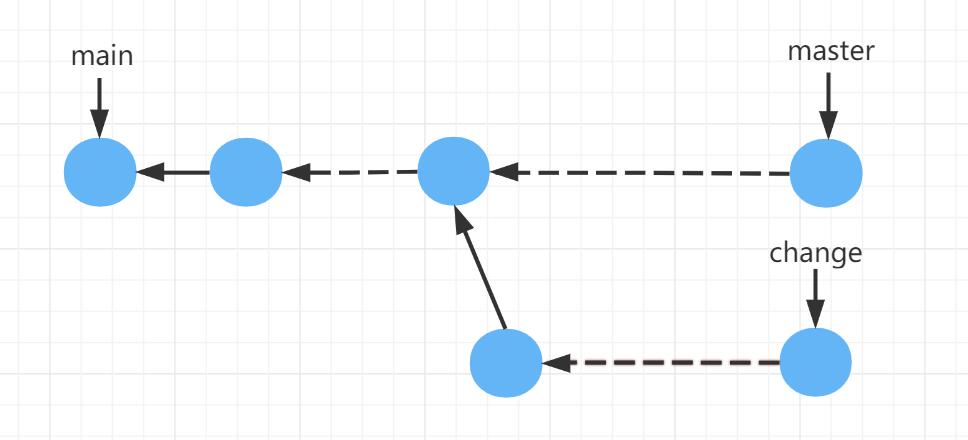
## Git仓库结构

请在完成全部实验要求之后，利用Git log指令或Git图形化客户端或GitHub上项目仓库的Insight页面，给出你的仓库到目前为止的Object Graph，尤其是区分清楚change分支和master分支所指向的位置。

使用git log指令，得到如下结果：



可以看出，Git仓库到目前为止的Object Graph有如下形式：



# 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 |
| 2021.6.28 | 12:30-14:30 | 设计IntervalSet接口并完成一个具体的实现类CommonIntervalSet，编写测试 | 完成 |
| 2021.6.28 | 14:30-17:00 | 设计MultiIntervalSet接口并完成具体的实现类CommonMultiIntervalSet，编写测试 | 完成 |
| 2021.6.28 | 17:30-18:30 | 设计IntervalSet和Common-MultiIntervalSet的装饰器 | 完成 |
| 2021.6.28 | 18:30-19:30 | 在有无空白的维度上，设计NoBlankIntervalSet接口并完成具体实现类CommonNoBlankIntervalSet，编写测试 | 完成 |
| 2021.6.28 | 19:30-21:00 | 在是否允许重叠的维度上，设计实现不允许重叠的装饰类NoOverlapIntervalSet和NoOverlap-MultiIntervalSet，编写测试 | 完成 |
| 2021.6.28 | 21:30-22:30 | 在周期性的维度上，设计实现周期性的装饰类PeriodicIntervalSet和PeriodicMultiIntervalSet，编写测试 | 完成 |
| 2021.6.29 | 9:00-10:00 | 实现Employee、Process、Course类，编写测试 | 完成 |
| 2021.6.29 | 10:00-12:30 | 设计实现API，编写测试 | 完成 |
| 2021.6.29 | 13:30-14:00 | 实现DutyIntervalSet、Process-IntervalSet和CourseIntervalSet，编写测试 | 完成 |
| 2021.6.29 | 14:00-17:30 | 实现DutyRosterApp | 测试健壮性时发现很多不足，延期一小时完成 |
| 2021.6.29 | 19:00-22:00 | 实现CourseScheduleApp | 测试健壮性时发现很多不足，延期半小时完成 |
| 2021.6.30 | 9:00-12:00 | 实现ProcessScheduleApp | 完成 |
| 2021.6.30 | 16:30-17:30 | 学习正则表达式 | 完成 |
| 2021.6.30 | 18:30-20:30 | 设计实现解析器Parser，编写测试 | 由于对正则表达式不太熟悉，延期半小时完成 |
| 2021.6.30 | 21:00-22:30 | 向DutyRosterApp中加入解析文件功能 | 完成 |
| 2021.7.1 | 9:00-10:00 | 完善整个项目的注释（spec、AF、RI、safe from exposure、test strategy） | 完成 |
| 2021.7.1 | 10:30-11:30 | 修改代码以面对新的变化 | 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 |
| 对装饰器不够熟悉，不知如何编写正确的装饰器以实现功能。 | 在网上查看其它应用使用装饰器的方法并加以总结，逐渐熟悉装饰器的使用。 |
| 编写APP时遇到很多的健壮性问题。 | 仔细分析每个步骤中用户可能的所有输入，针对任何非法情况都作出提示。虽然很耗费时间，但效果很好。 |
| 不知如何设计正则表达式来抽取大量的文件信息。 | 在学习正则表达式语法的同时进行一些小的测试，从小的正则表达式开始，逐渐累积成复杂的正则表达式，最终实现对文件的解析。 |

# 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

设计ADT时一定要考虑得全面且清楚，在最终决定实现方案后再编写代码，否则在后面的应用实现中发现问题时只能再重新设计。同时，在编写APP时，一定要事先考虑健壮性的问题，不然，在编写完代码之后进行测试时会遇到很多的健壮性问题，这时再去修改就会使得代码很臃肿，可读性变差，出错的可能性更高。

## 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？

面向ADT的编程需要从实际中进行抽象，并进行合理的设计，有很好的可扩展性和可复用性；而直接面向应用场景编程只针对特定应用，每次更换应用场景时都要重新编程，扩展性很差，只适合简单的应用场景。本实验中设计一个ADT就可以应用到三个不同的场景，大大缩短了开发时间。

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？

这些工作使得客户端了解各方法的功能但无法得知内部具体实现，可以防止内部变量被客户端恶意修改，时刻检查表示不变量，保证安全性。虽然这些工作有些麻烦，但却是好的软件必须具备的，因此我愿意在以后编程中坚持这么做。同时，在复杂的软件开发过程中，好的注释可以节省大量阅读代码的时间，使得开发时间大大降低。

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？

由于API面向的场景是广泛的，因此开发难度很大，但是一旦开发一个好的API，就可以在大量场合中应用，大大提高代码的复用性。

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？

语法驱动编程为实际应用提供了很大的便利，用户无需繁琐地一行一行地输入信息，而只需提供一个文件以及一定的语法规则即可。而对于应用的开发者，只需根据语法规则编写代码，即使改变语法规则也可以很快地修改实现代码，有很好的可维护性。

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？

ADT的难度主要体现在抽象上。一个好的ADT既不能过于具体，也不能过于抽象。需要从大量应用场景中寻找共性，抽象的程度也很难把握。对于这种情况，只能反复的推敲，比较不同设计方案的差异并选择最好的ADT设计方案。

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？

接口、抽象类、类的抽象程度一定是逐渐降低的，将所有应用的共性抽象为接口，然后在抽象类以及类中添加新的特性。通过接口的组合可以形成新的接口，并可以具备不同接口中的抽象。类的继承也可以增加更具体的新的特性。同时，使用正确的设计模式可以使得代码的可复用性和可维护性最大化。

1. 关于本实验的工作量、难度、deadline。

难度适中，可以接受，但是工作量很大，尤其是APP的编写要花费很长时间。虽然给了三周时间，但是和其他课的实验、大作业、考试有重叠，总体上时间还是很紧。希望各课程的老师可以相互协调一下实验安排。

1. 到目前为止你对《软件构造》课程的评价。

逐渐理解了软件构造过程中独有的思路和方法，也逐渐适应了与之前完全不同的编程过程。通过Lab3大量的代码训练，自己的编程水平也有了极大的提高。