

**2024年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 |  |
| 学号 | 2022112831 |
| 班号 | 2237101 |
| 电子邮件 | Phantasia\_March@outlook.com |
| 手机号码 |  |

**目录**

[1 实验目标概述 1](#_Toc164601170)

[2 实验环境配置 1](#_Toc164601171)

[3 实验过程 1](#_Toc164601172)

[3.1 待开发的三个应用场景 1](#_Toc164601173)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 1](#_Toc164601174)

[3.2.1 IntervalSet<L>的共性操作 1](#_Toc164601175)

[3.2.2 局部共性特征的设计方案 2](#_Toc164601176)

[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） 2](#_Toc164601177)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 2](#_Toc164601178)

[3.3.1 MultiIntervalSet<L>的共性操作 2](#_Toc164601179)

[3.3.2 局部共性特征的设计方案 2](#_Toc164601180)

[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） 2](#_Toc164601181)

[3.4 面向复用的设计：L 2](#_Toc164601182)

[3.5 可复用API设计 2](#_Toc164601183)

[3.5.1 计算相似度 2](#_Toc164601184)

[3.5.2 计算时间冲突比例 2](#_Toc164601185)

[3.5.3 计算空闲时间比例 2](#_Toc164601186)

[3.6 应用设计与开发 2](#_Toc164601187)

[3.6.1 排班管理系统 2](#_Toc164601188)

[3.6.2 操作系统的进程调度管理系统/课表管理系统 2](#_Toc164601189)

[3.7 基于语法的数据读入 3](#_Toc164601190)

[4 实验进度记录 3](#_Toc164601191)

[5 实验过程中遇到的困难与解决途径 3](#_Toc164601192)

[6 实验过程中收获的经验、教训、感想 3](#_Toc164601193)

[6.1 实验过程中收获的经验和教训 3](#_Toc164601194)

[6.2 针对以下方面的感受 3](#_Toc164601195)

# 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

按照Lab 3中的步骤将仓库clone到本地，按提交格式命名与组织。

在这里给出你的GitHub Lab3仓库的URL地址（HIT-Lab3-学号）。

<https://github.com/ComputerScienceHIT/HIT-Lab3-2022112831>

# 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。

三个应用简述如下：

1. 值班表管理（DutyRoster）

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

1. 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。

1. 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”）且由同一位教师承担并在同样的教室进行；允许课表中有空白时间段（未安排任何课程）；考虑到不同学生的选课情况不同，同一个时间段内可以安排不同的课程（例如周一上午 3-4 节的“计算方法”和“软件构造”）；一位教师也可以承担课表中的多门课程。

共性：都涉及在一段时间上，截取时间片段与某个标签进行对应

差异：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 值班表 | 操作系统进程 | 大学课表 |
| 实体类信息 | 员工名字，职位，手机号码 | 进程 ID、进程名称、最短执行时间、最长执行时间 | 课程 ID、课程名称、教师名字、地点、周学时数 |
| 标签和时间段 | 一个标签只能对应多个时间段 | 一个标签可以对应多个时间段 | 一个标签可以对应多个时间段 |
| 分配的时间段之间可否有空闲 | 不可 | 可 | 可 |
| 是否包含周期性的时间段 | 否 | 否 | 是 |
| 是否允许不同的 interval 之间有重叠 | 否 | 否 | 是 |

## 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

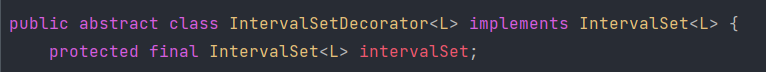
IntervalSet<L> 定义了一组在时间轴上分布的“时间间隔”（interval），每个时间间隔都与一个特定的标签相关联，且不允许有重复的标签。它将所有共同的操作封装在 IntervalSet<L> 接口中。这些共同的操作包括：

|  |  |
| --- | --- |
| 创建一个空对象 | empty() |
| 在当前对象中插入新的时间段和标签 | void insert(long start,long end, L label) |
| 获得当前对象中的标签集合 | Set<L> labels() |
| 从当前对象中移除某个标签所关联的时间段 | boolean remove(L label) |
| 返回某标签对应的时间段的开始时间 | long start (L label) |
| 返回某标签对应的时间段的结束时间 | long end (L label) |
| 判断IntervalSet是否为空 | boolean isEmpty() |
| 以字符串形式打印相关信息 | String toString() |

其中，empty()方法设置为静态工厂方法，其余方法设置为实例方法。

### 局部共性特征的设计方案

为了增强代码的可复用性，我们选择方案 6用于设计 IntervalSet 的通用特性。在该方案中，CommonIntervalSet 被视为原始的、未经修饰的对象，而是否重叠、是否存在空闲、是否有周期性这三个属性被视为三种不同的“装饰”（每个维度的不同特性可以产生不同的装饰效果）。利用设计模式中的装饰器模式，可以分别实现这些装饰，以便在具体应用中，通过为一个 CommonIntervalSet 对象逐步添加不同特性的装饰，来实现所需的组合特性。装饰器模式的优势在于，它允许装饰类和被装饰类独立发展，两者之间不需要有太多的相互依赖或影响。

具体来说，对于 IntervalSet，我采用了装饰器模式，实现了 IntervalSet<L> 接口的一个抽象装饰类 IntervalSetDecorator，并将 IntervalSet 对象作为它的实例变量。这样设计允许 IntervalSetDecorator 扩展 IntervalSet 的功能，同时保持原有类的封装和接口不变。

在这之后，创建扩展了 IntervalSetDecorator类的实体装饰类。对于IntervalSet及其可能的应用场景，设计了两个实体装饰类：NoBlankIntervalSet和NonOverlapIntervalSet，分别用于判断IntervalSet中是否存在空闲，是否存在重叠。对这两个实体装饰类的判断算法分别阐释如下：

**NoBlankIntervalSet**

NoBlankIntervalSet 装饰类通过以下步骤判断 IntervalSet 是否存在空闲：

* 使用迭代器遍历 intervalSet，记录时间间隔的最小值和最大值，并将每个标签对应的时间间隔存储在映射中。
* 从最小值到最大值逐一检查每个整数，若该整数不在任何时间间隔内，则表示存在空闲，返回 true。
* 若所有整数均被时间间隔覆盖，则 intervalSet 无空闲，返回 false。



**NonOverlapIntervalSet**

NonOverlapIntervalSet 装饰类通过以下步骤判断 IntervalSet 是否存在时间重叠：

* 同样使用迭代器遍历 intervalSet，确定时间间隔的边界，并将信息存储在映射中。
* 遍历从最小值到最大值的每个整数，使用标志位检查每个点是否被重复覆盖。
* 若发现任一整数被两个时间间隔共享，则存在重叠，返回 true。
* 若整个遍历过程中未发现重叠，则返回 false，表示没有时间重叠发生。



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

CommonIntervalSet<L> 是 IntervalSet<L> 接口的一个具体实现，实现了接口中的共性方法。它包含两个成员变量：timeschedule 和 labels。timeschedule 是一个 HashMap，以键值对形式存储每个时间段，其中键是标签 L，值是包含起始时间和结束时间的 List<Long>。labels 是一个 HashSet，存储已添加到 IntervalSet 中的标签 L。



方法实现：

1. void insert(long start, long end, L label)：首先检查异常情况，如果正常，则将标签 L 和时间区间 [start, end] 作为键值对存入 timeschedule，并更新 labels。



2. Set<L> labels()：返回 labels 集合的一个防御性拷贝，以保护原始数据不被外部修改。

3. boolean remove(L label)：使用迭代器遍历 timeschedule，找到并删除特定标签 L 的键值对，并从 labels 中移除该标签。如果找到并删除成功，返回 true；否则返回 false。



4. long start(L label)：遍历 timeschedule，返回与标签 L 相关联的时间区间的起始时间。如果未找到，则返回 -1。

5. long end(L label)：遍历 timeschedule，返回与标签 L 相关联的时间区间的结束时间。如果未找到，则返回 -1。

6. boolean isEmpty()：检查 schedule 和 labels 是否为空，从而确定 IntervalSet 是否为空。

7. boolean checkBlank() 和 boolean checkOverlap()：使用装饰器模式，委托给装饰类检查是否存在空闲或重叠，并返回结果。



8. String toString()：将 IntervalSet 的信息以字符串形式打印出来，格式为 “标签 + 对应的时间段”。



CommonIntervalSet<L> 类通过实现这些方法，提供了一套完整的接口来管理时间区间和相关联的标签。装饰器模式的应用允许进一步的功能扩展，而不影响现有代码结构。

## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

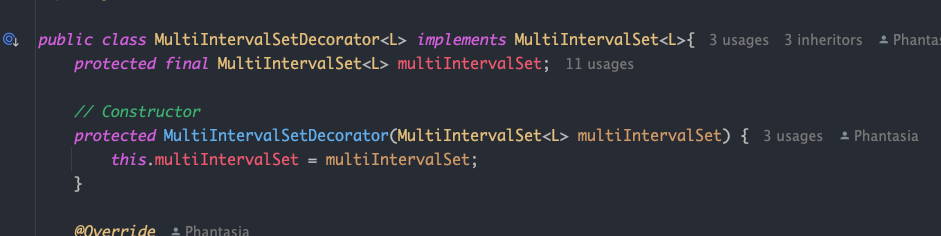
MultiIntervalSet<L> 是一个接口，它描述了一组在时间轴上分布的时间段，与 IntervalSet<L> 不同的是，MultiIntervalSet<L> 允许同一个标签对象 L 绑定到多个时间段上。这个接口封装了 MultiIntervalSet 的共性操作，具体的共性操作可能包括：

|  |  |
| --- | --- |
| 判断MultiIntervalSet是否为空 | boolean isEmpty() |
| 在当前对象中插入新的时间段和标签 | void insert(long start,long end, L label) |
| 获得当前对象中的标签集合 | Set<L> labels() |
| 从当前对象中移除某个标签所关联的时间段 | boolean remove(L label) |
| 从当前对象中获取与某个标签所关联的所有时间段 | IntervalSet<Integer> intervals(L label) |
| 判断时间轴是否允许空白 | boolean checkBlank() |
| 判断是否允许不同的 multiInterval 之间有重叠 | boolean checkOverlap() |
| 判断是否包含周期性的时间段 | boolean checkPeriodic() |
| 以字符串形式打印相关信息 | String toString() |

这些方法都设置为实例方法。

### 局部共性特征的设计方案

与IntervalSet类似，在设计MultiIntervalSet的局部共性特征时，也采用了Decorator设计模式。实现MultiIntervalSet<L>接口的抽象装饰类MultiIntervalSetDecorator，其中将MultiIntervalSet对象作为其实例变量。



接下来，创建了扩展MultiIntervalSetDecorator类的具体装饰类。针对MultiIntervalSet及其应用场景（如进程调度和课表安排），实现了三个具体的装饰类：NoBlankMultiIntervalSet、NonOverlapMultiIntervalSet和NonPeriodicMultiIntervalSet，分别用于判断MultiIntervalSet中是否存在空闲、是否存在重叠、是否具有周期性。在这部分中，具体装饰类的实现与之前的IntervalSet类似，仅阐述一些实现细节上的不同：

1. NoBlankMultiIntervalSet

在NoBlankMultiIntervalSet中，判断MultiIntervalSet中是否存在空闲的思路如下：

首先利用迭代器遍历intervalSet，找到时间段的最小值和最大值min和max；并在遍历的同时，将每组标签对应的start-end保存在Map<Long, Long>中，以便后续使用。与IntervalSet不同，这里采用MultiIntervalSet的intervals方法来获取某个标签对应的所有时间段。

从min到max，步长为1遍历每个键值对，这部分与IntervalSet基本相同。

2. NonOverlapMultiIntervalSet

在NonOverlapMultiIntervalSet中，判断MultiIntervalSet中是否存在重复时间段。这部分与NonOverlapIntervalSet的区别主要在遍历寻找min和max值，故不再赘述。

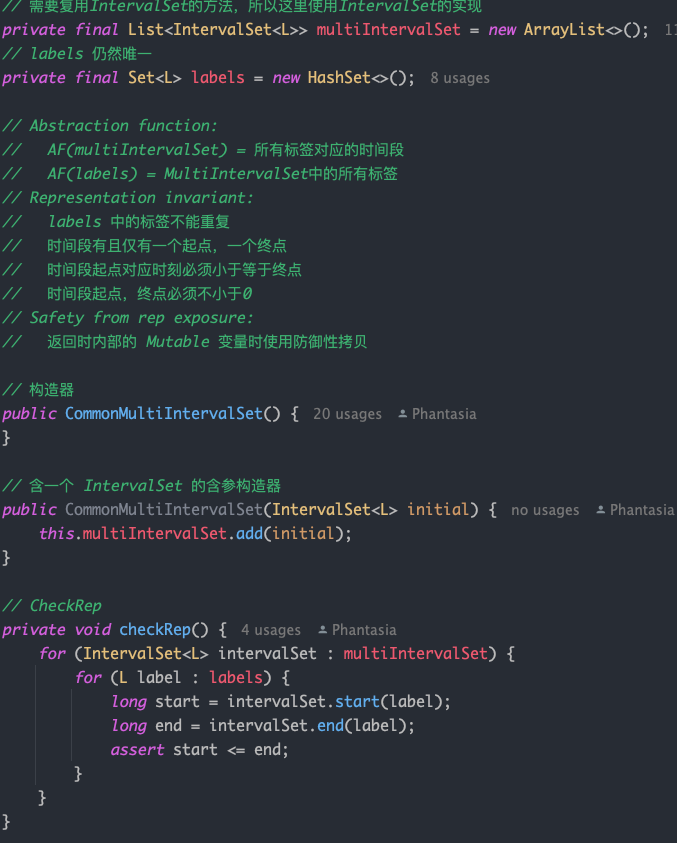
3. NonPeriodicMultiIntervalSet

暂未实现，后续未使用，设计思路有所区别。

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

为MultiIntervalSet<L>设计其具体的实现类CommonMultiIntervalSet<L>。在CommonMultiIntervalSet中，主要利用了委派的方法，基于IntervalSet实现其相关功能。

在CommonMultiIntervalSet<L>中，设置了如下Rep，AF，RI，Safety from exposure，checkRep，以及构造器：



接下来，分点阐述每个函数的具体实现。对于部分关键函数，附上图示说明。

1. boolean isEmpty()

根据multi和label调用isEmpty()方法的返回结果，来判断CommonMultiIntervalSet是否为空。

1. void insert(long start, long end, L label)

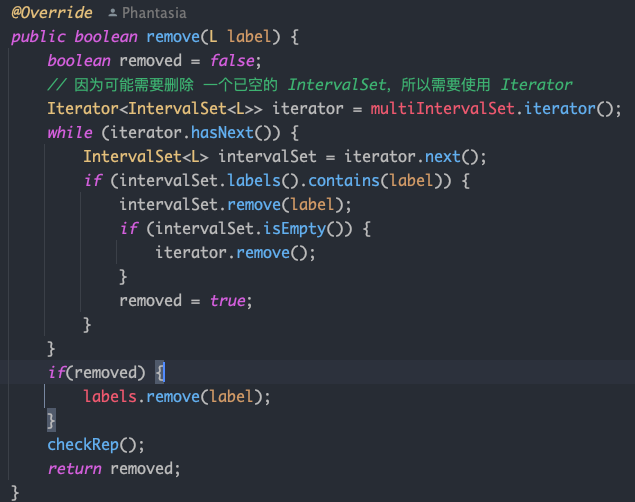
基本思路是：multi的时间段保存在List的IntervalSet中，因为IntervalSet不能保存重复的标签，所以需要对multi进行遍历判断。若在其中的IntervalSet中发现了重复的标签，则新建一个IntervalSet，将标签与对应时间段的键值对加入新的IntervalSet，再将新的IntervalSet加入到multi中。

1. Set<L> labels()

采用防御性拷贝的方式，返回一份新的Set<L>。

1. boolean remove(L label)

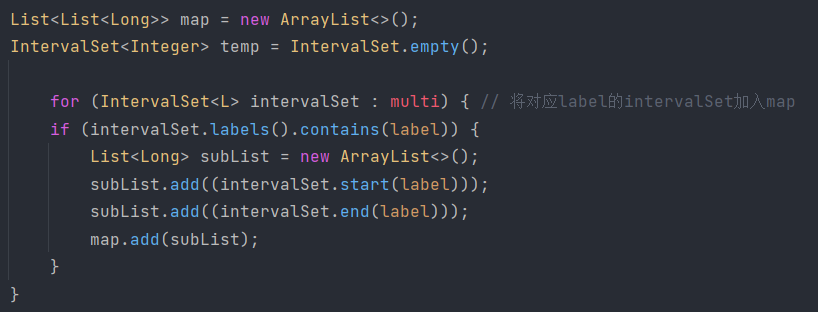
从当前MultiIntervalSet中移除某个标签所关联的所有时间段。在实现时，用增强for循环对multi中的IntervalSet<L>进行检索，若在当前IntervalSet<L>中发现了label对应的时间段，则通过IntervalSet的remove()方法删掉该时间段，若当前IntervalSet<L>中没有该时间段，则退出循环（因为根据之前设计的insert()方法，若当前IntervalSet<L>没有该标签，则之后的IntervalSet<L>必定也没有）。返回值为true说明成功进行了删除，返回值false说明没有找到label对应的时间段。



1. IntervalSet<Integer> intervals(L label)

该方法的作用是从当前对象中获取与某个标签所关联的所有时间段，并将其保存在IntervalSet<Integer>中。

首先调用自身的isEmpty()方法进行判断，若为空则直接返回null；然后进行下一步，将对应label的intervalSet加入List<List<Long>> map。这里叫map的原因是该数据结构模拟的是键值对集合，即map，但其与Map的区别为Map中键是唯一的，若重复添加同一个键对应的不同值，则后值会覆盖新值。所以为避免这个问题，采用自行设计的List<List<Long>>数据结构来表示map。这部分的具体实现如下：

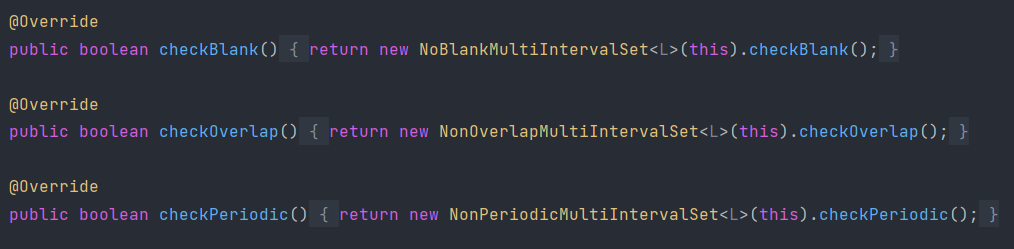


完成以后，将该label对应的intervalSet按从小到大的次序加入list，具体实现方式为遍历map,寻找到其中start的最小值，将该“键值对”加入待返回的intervalSet，然后将其从map中删去，直到map为空为止。



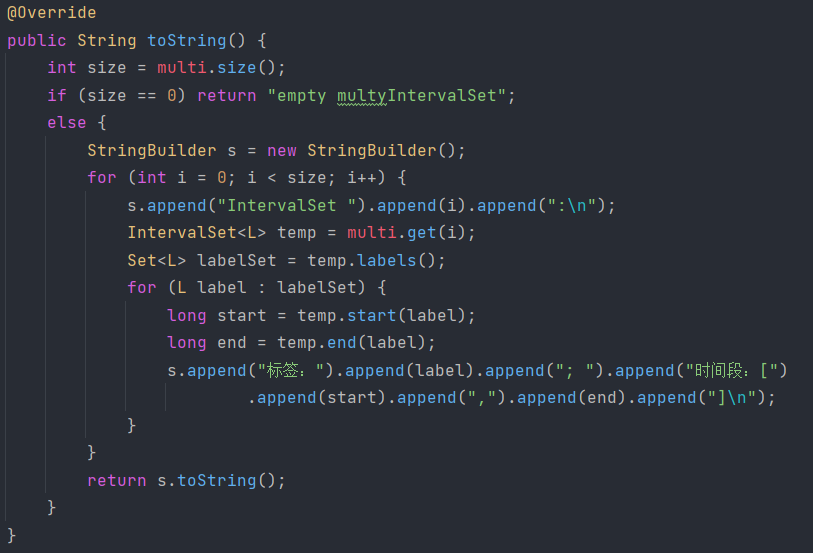
1. boolean checkBlank()、checkOverlap()、checkPeriodic()

这几个部分与IntervalSet中对应部分类似，直接返回一个新装饰对象该方法的返回值。



1. String toString()

在这个方法中，首先对multi是否为空进行检查，然后遍历multi的每一个IntervalSet，然后将toString的工作委派给IntervalSet来完成。具体实现如下：



## 面向复用的设计：L

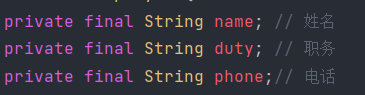
IntervalSet<L>和 MultiIntervalSet<L>中的泛型参数 L，可以是任何 immutable的类。对于要开发的三个具体应用来说，L 分别应为“员工”（Employee）、“进程”（Process）、“课程”（Course）。于是分别实现这三个ADT。

* + 1. **ADT的共有属性**

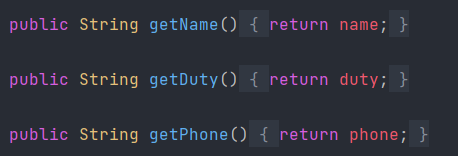
由于同类ADT之间可能出现相互比较的情况，故对于每个ADT，均需要重写equals()与hashCode()方法。另外，为了表示统一，三个ADT一律只设计带参构造方法，不设置无参构造。

* + 1. **员工(Emplpyee)**

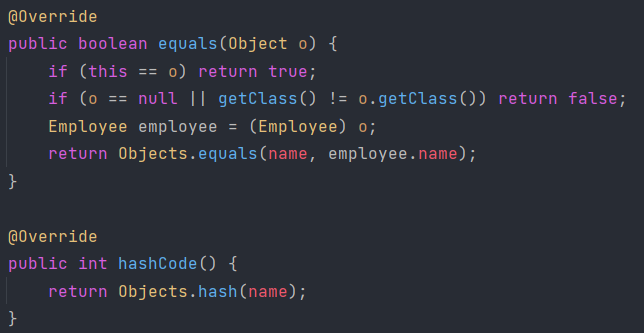
员工类包含三个私有成员变量，其字段与意义如下所示：



Employee中，为每个成员变量设置Getter方法。

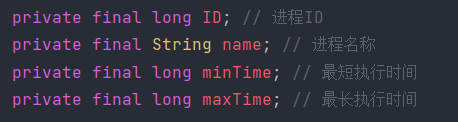


员工类重写了equals()与hashCode()方法，依据是成员变量中name字符串：

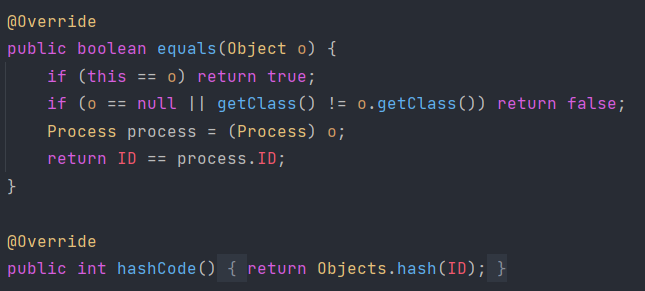


* + 1. **进程（Process）**

进程类包含三个私有成员变量，其字段与意义如下所示：

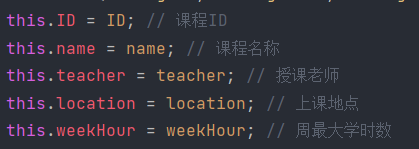


为每个成员变量设置Getter方法，并重写了equals()与hashCode()方法，依据是成员变量中的ID：

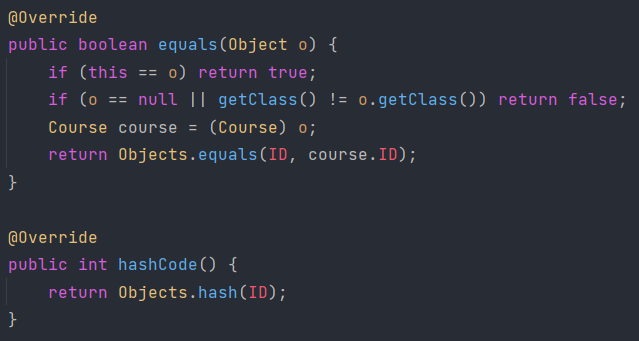


* + 1. **课程（Course）**

课程类包含五个私有成员变量，其字段与意义如下所示：



为每个成员变量设置Getter方法，并重写了equals()与hashCode()方法，依据是成员变量中的ID：



## 可复用API设计

在本工程中，设置单独的APIs.java文件，用来保存以下可复用的方法。

### 计算相似度

计算相似度的函数为：

public double Similarity(MultiIntervalSet<L> s1, MultiIntervalSet<L> s2)

具体实现思路如下：

首先，对s1和s2分别调用labels()方法，找到s1和s2的共有标签，保存在Set<L> labels中。按照与之前类似的方法，对时间段进行遍历，找到时间轴的起始值min和结束值max。

对每个共有标签，对s1进行遍历，寻找其在s1中对应的时间段，这样的寻找是必定命中的，由此获取当前共有标签在s1中的某一个时间段，记为[baseStart, baseEnd]；之后遍历s2，在s2中，选取同一标签下的时间段进行比较，记某次选择的时间段为[compareStart, compareEnd]，若compareStart > baseEnd或compareEnd < baseStart，即两个时间段完全没有重合，则继续下一次循环；否则则计算accStart为两个起始值的最大值，accEnd为两个结束值的最小值，[accStart, accEnd]即为重合的长度，将该长度累加到返回变量中。

循环完所有的共有标签，得到最终的相似度，并将其返回。

步骤3对应的核心函数实现如下：

### 计算时间冲突比例



### 计算空闲时间比例

计算时间冲突比例的函数有两个，运用了方法重载，以便能够对IntervalSet<L>和MultiIntervalSet<L>都进行处理：

public double calcConflictRatio(IntervalSet<L> set)

public double calcConflictRatio(MultiIntervalSet<L> set)

具体实现思路如下：

1. 首先找到时间轴的起点与终点min和max，并将每组标签对应的start-end保存在键值对中。考虑到起点可能重复，键值对通过List<List<Long>>来实现。

2. 设置标志变量flag，设置冲突时间初始为0。从min到max进行循环，步长为1遍历所有键值对，若某个时间点同时在两个键值对中，则判断当前时间点冲突，冲突时间变量增加1。

3. 循环结束，返回冲突时间/总时间的值即为冲突比例。注意，总时间为(max - min + 1)。

关键函数实现如下：



## 应用设计与开发

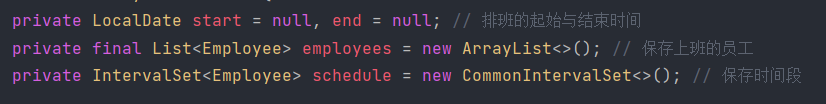
利用上述设计和实现的ADT，实现手册里要求的各项功能。

### 排班管理系统

针对排班管理系统的功能要求，设计了具体的ADT子类型DutyIntervalSet与客户端DutyRosterApp。以下分别阐述。

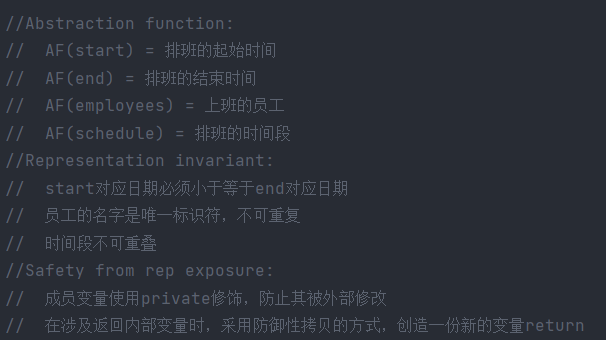
* DutyIntervalSet

DutyIntervalSet中设置了如下成员变量：



start与end是LocalDate类型对象，保存排班的起始与结束时间，List employees保存着上班的员工，schedule是IntervalSet类型，保存着排班的时间段。

DutyIntervalSet的AF，RI，Safety from exposure如下：



DutyIntervalSet中设计了如下方法：

|  |  |
| --- | --- |
| public boolean setDate(String s1, String s2) | 设定排班开始日期、结束日期，具体到年月日 |
| public boolean addEmployee(String info) | 增加一组员工，包括他们各自的名字、职务、手机号码 |
| public boolean deleteEmployee(String name) | 删除员工。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。 |
| public boolean manualRosters(String name, String sStart, String sEnd) | 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 |
| public boolean autoRosters() | 自动根据现有员工进行排班 |
| public boolean deleteRoster(String name, String start) | 删除特定员工的排班 |
| public void checkFullRoster() | 检查排班是否排满，并返回相应的信息 |
| public void rosterVisualization() | 可视化当前排班表 |
| public void clear() | 清空数据 |
| private boolean checkDate(int year, int month, int day) | 检查日期是否合法 |
| private boolean checkInRoster(Employee e) | 检查某员工是否在排班表中 |
| private long calculateDateDifference(LocalDate start, LocalDate end) | 计算两个时间点之间间隔的天数 |
| private List<Integer> dateConversion(long startPoint, long endPoint) | 根据时间轴上的起始与终止时间，返回对应的实际日期 |
| private Map<Long, Long> saveMap() | 将用户时间段保存在map中 |
| private double calcFreeTimeRatio(IntervalSet<Employee> set) | 计算一个 IntervalSet<Employee> 对象中的空闲时间比例 |

下面对各个方法进行阐述：

1. public boolean setDate(String s1, String s2)

该方法读入两个字符串s1与s2，分别代表输入的起始时间和结束时间，输入格式为YYYY-MM-DD。在方法中，首先检查已经有排班、输入格式错误、起始时间在终止时间后等情况，无误后对输入字符串进行分割，并保存至代表起始时间和终止时间的LocalDate类型成员中。完成后输出提示信息。

1. public boolean addEmployee(String info)

按照name{duty, phone}的格式读入字符串info，对员工是否已存在（通过employee类重写的equals()方法实现）、输入格式是否正确进行检查，若无误则新增该员工，并打印提示信息。

1. public boolean deleteEmployee(String name)

检查非法情况，然后根据输入的name在employees中进行匹配，若匹配到则返回true，并将其移除，否则返回false。

1. public boolean manualRosters(String name, String sStart, String sEnd)

输入员工的名字与排班时间，对员工进行排班。首先检查异常情况：排班起始结束未设定、输入格式错误、员工不存在、员工已在排班表中、日期有误等，如果无误，则向schedule中加入，在加入完成后调用intervalSet的checkOverlap方法，检测加入后是否出现日期重叠的现象，若是，则删去刚刚加入的时间段，并打印错误信息，返回false，若不是，则说明时间段没有重叠，可以正常加入。

1. public boolean autoRosters()

在此方法中，采用如下算法对员工进行排班：

如果排班总时长days可以被人数n整除，则每个人排days/n天班；如果days不能被n整除，则最后一个人排 days%n 天班，其他人排 days/n 天班。具体实现如下：



1. public boolean deleteRoster(String name, String start)

此方法的功能为删除特定员工的排班。在完成对异常情况的检查后将schedule中的特定时间段删除，并打印提示信息返回。

1. public void checkFullRoster()

首先对排班是否排满进行检查。具体实现为调用intervalSet的checkBlank方法。同时，由于checkBlank方法无法得知具体排班的最小时间和最大时间之外是否还有空闲时间，所以需要再通过检查具体排班的最小时间和最大时间是否等于start与end来进行准确的判断。

若排班未排满，则检测空闲时间块并返回具体的空闲时间段。具体实现如下：

首先获取到键值对形式保存的时间段，然后从排班起点start到结尾end依次遍历，步长为1，获得每个空缺的时间点，将空缺的时间点保存到List<Long> interval中。

其次，对interval中的时间点进行合并操作。通过算法找到每个空闲时间段的起点和终点，将他们以键值对的形式加入Map<Long, Long> intervalMap，从而得到空闲时间段的集合

最后，打印空闲时间段的信息并返回。

核心部分函数如下：



1. public void rosterVisualization()

对当前排班表进行可视化。打印输出相关信息。

1. public void clear()

清空该DutyRosterIntervalSet的所有数据

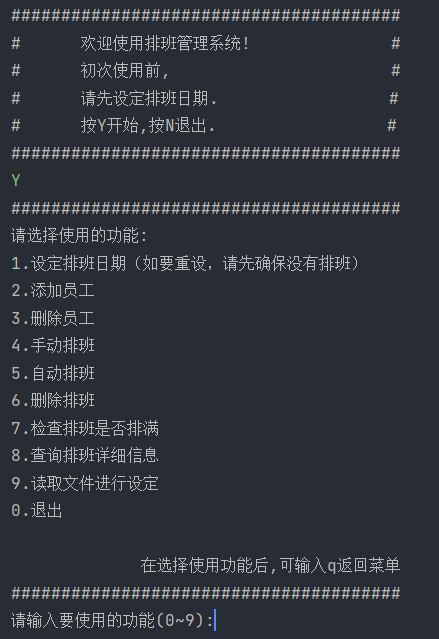
本类中的其他私有方法功能描述见前文表中，其具体实现较为简单，功能较为单一故不再赘述。

为DutyIntervalSet编写了测试用例，测试策略见DutyIntervalSetTest类的Testing Strategy部分。经过测试，DutyIntervalSet的代码覆盖度较高。



* DutyRosterApp

DutyRosterApp应用了DutyIntervalSet类，为其构造了命令行前端页面。



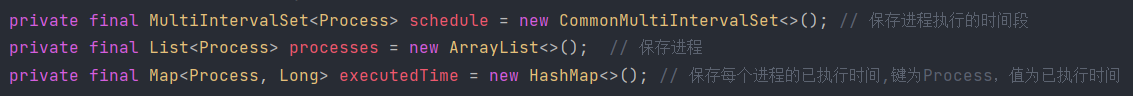
具体的图形化功能可以在实际运行中展示

### 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的功能要求，设计了具体的ADT子类型ProcessIntervalSet与客户端ProcessScheduleApp。以下分别阐述。

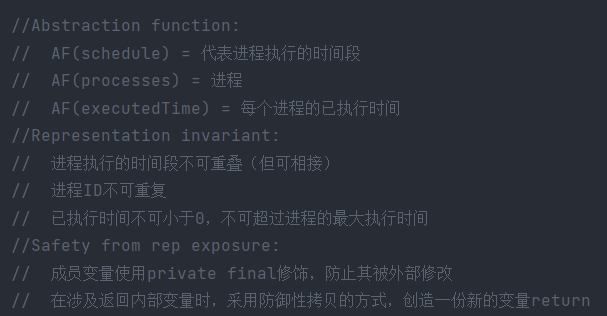
* ProcessIntervalSet

ProcessIntervalSet中设置了如下成员变量：



schedule是MultiIntervalSet类型，保存着排班的时间段; List processes保存着上班的员工，excutedtime中保存着每个进程的已执行时间,键为Process，值为已执行时间。

ProcessIntervalSet的AF，RI，Safety from exposure如下：



ProcessIntervalSet中设计了如下方法：

|  |  |
| --- | --- |
| public boolean addProcess(String info) | 添加进程 |
| public boolean RAschedule() | 随机选择进程进行调度 |
| public boolean SPschedule() | 按“最短进程优先”的原则进行调度 |
| public void visualization() | 可视化进程调度系统情况 |
| private List<List<long[]>> toList() | 将所有的时间段保存在list中，list.get(0):进程的ID list.get(1):[起始时间，终止时间] |

下面对各个方法进行阐述：

1. public boolean addProcess(String info)

该方法读入字符串info，以“ID-名称-最短执行时间-最大执行时间”的表示进程信息。检查完异常情况后更新processes与executedTime，将新进程添加至processes，并在executedTime置其对应值为0。

1. public boolean RAschedule()

随机选择进程进行调度。首先检查异常情况，无误后设置起始时间点timepoint为0，进行如下循环直到每个进程都执行到最大时间：

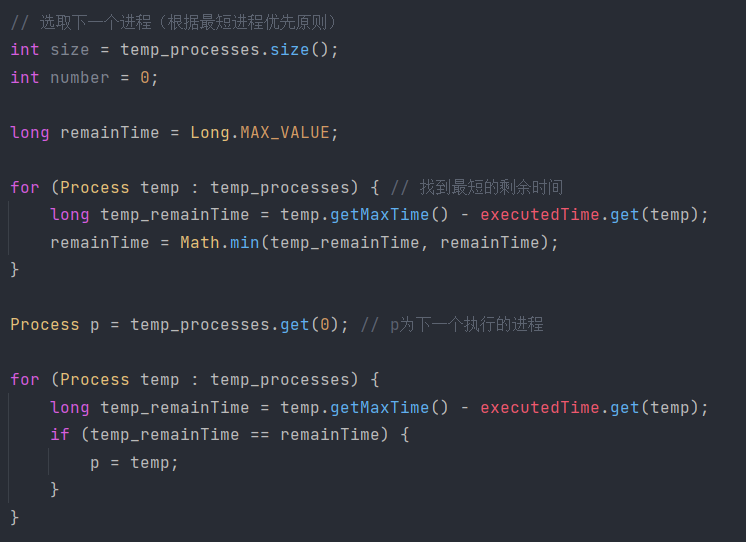
* 随机设置休眠时间。设置随机数种子long sleepTime = rand.nextInt(10)，每次执行前都在timePoint的原值基础上累加一个当次的sleepTime，代表内核的随机休眠时间。
* 设置一个随机数种子，从[0,temp\_process.size()]中随机选取一个数作为此次选取进程的序号，得到此次选取的进程。
* 利用随机数long thisTime = (long) (rand.nextDouble() \* maxTime)得到此次的进程执行时间。rand.nextDouble()返回一个[0,1]的浮点数值，在此代表时间执行比例。
* 因为累加上此次执行时间后最大执行时间可能超出，所以需要进行条件判断，从而得到真实的此次执行时间，具体算法如下：



* 若当前进程执行完毕，则将其从temp\_processes中删除，循环执行这样的操作直到temp\_processes为空。

1. public boolean SPschedule()

按“最短进程优先”的原则进行调度。该方法大体上与RAschedule()类似，区别在于找到下一个该执行的进程的思路。在这里，通过遍历进程，用每个进程的最大执行时间减去已执行时间得到每个进程的剩余执行时间，选取剩余执行时间最短的进程作为下一个要执行的进程。进程选取部分的代码如下：



1. public void visualization()

进行进程调度情况的可视化，打印相关信息。

1. private List<List<long[]>> toList()

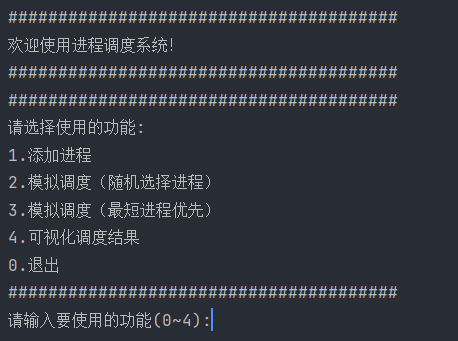
由于每个进程可能执行多次，所以在这里不能使用Map，只能自行构造List<List<long[]>>来模拟键值对。

为ProcessIntervalSet编写了测试用例，测试策略见ProcessIntervalSetTest类的Testing Strategy部分。经过测试，ProcessIntervalSetTest的代码覆盖度较高。



* ProcessScheduleApp

ProcessScheduleApp应用了ProcessIntervalSet类，为其构造了命令行前端页面。菜单页面如下：



### 课表管理系统

针对排班管理系统的功能要求，设计了具体的ADT子类型CourseIntervalSet与客户端CourseScheduleApp。以下分别阐述。

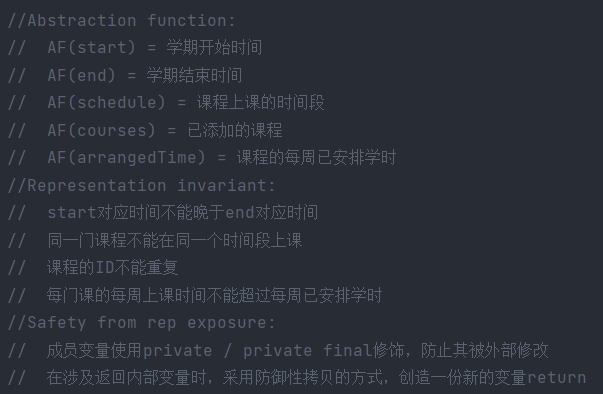
* CourseIntervalSet

CourseIntervalSet中设置了如下成员变量：



start与end是LocalDate类型对象，保存学期的起始与结束时间；schedule是MultiIntervalSet类型，保存着不同课程在一周中的时间段；List courses保存着不同的课程；arrangedTime保存着每门课的每周已安排学时。

CourseIntervalSet的AF，RI，Safety from exposure如下：



CourseIntervalSet中设计了如下方法：

|  |  |
| --- | --- |
| public boolean setTerm(String date, String rawWeek) | 设定学期开始日期（年月日）和总周数（例如 18） |
| public boolean addCourse(String info) | 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） |
| public boolean arrangeCourse(String courseID, String info) | 手工选择某课程、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，每次课时间长度为 2 小时，可重复安排，直到达到周学时数时该课程不能再安排； |
| public void checkSchedule() | 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 |
| public boolean showSchedule(String date) | 查看本学期内任意一天的课表结果 |
| private String numberToWeek(int num) | 根据数字返回对应的星期几字符串 |
| private String dayOffsetToString(int dayOffset) | 根据日偏移量返回对应的上课时间 |
| private double calcFreeTimeRatio() | 计算当前schedule中的空闲时间比例 |
| private double calcConflictRatio() | 计算当前schedule中的重复时间比例 |

下面对各个方法进行阐述：

1. public boolean setTerm(String date, String rawWeek)

该方法读入两个字符串date与rawWeek，分别代表输入的学期开始时间和总周数。首先对异常情况进行检查，无误后对输入字符串进行分割，保存开始时间至LocalDate类型对象start中，并通过计算得到学期终止时间end，完成后输出提示信息。

1. public boolean addCourse(String info)

按照ID-课程名称-教师名称-上课地点-周学时数的格式读入字符串info，对异常情况进行检查，若无误则更新courses与arrangedTime，将新课程添加至courses，并在arrangedTime置其对应值为0。

1. public boolean arrangeCourse(String courseID, String info)

检查非法情况，然后根据输入的courseID在course中进行匹配，得到要安排的课程。在这里，我们需要对schedule进行处理，具体的处理思路如下：

设定星期一的第一次课（8点-10点）为起始位置0，星期日的最后一次课（19点-21点）为终结位置34。每天的5个时段对应5个偏移量，这样最后便可以得到每次课相较于星期一的第一次课的偏移量，若记为n，则向其中插入[n,n]的课程，这样便能方便地抽象化表示每周的课程安排。

1. public void checkSchedule()

首先调用schedule的labels方法，检查是否有未安排的课程，然后调用私有函数calcFreeTimeRatio()与calcConflictRatio()计算空闲与冲突的时间比例。

1. public boolean showSchedule(String date)

在此方法中，用户输入一个日期，处理后得到该日期对应的LocalDate对象，检查该对象是否在学期内，并得到该日期对应的星期数。应用 arrangeCourse()方法中计算偏移量的同样思路，得到该日的检索时段。遍历courses，寻找安排在当天的课程，并按时间先后顺序打印。

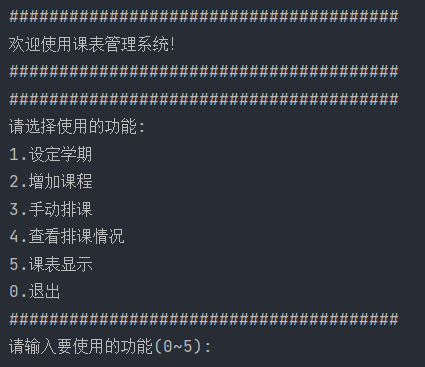
本类中的其他私有方法功能描述见前文表中，其具体实现较为简单，功能较为单一故不再赘述。其中，calcFreeTimeRate()与calcConflictRate()两方法与计算空闲与重叠的方法思路相似。

为CourseIntervalSet编写了测试用例，测试策略见CourseIntervalSetTest类的Testing Strategy部分。经过测试，CourseIntervalSet的代码覆盖度较高。



* CourseScheduleApp

CourseScheduleApp应用了CourseIntervalSet类，为其构造了命令行前端页面。



# 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 |
| 2024-5.12 | 19:00-02:00 | 构思整体思路，完成IntervalSet接口与测试代码的编写,完成CommonIntervalSet的编写完成MultiIntervalSet接口、具体实现、测试用例的编写考虑设计方案，实现Decorator相关部分 | 按计划完成 |
| 2024-5.14 | 14:00-03:00 | 实现API，同时编写部分DutyIntervalSet 和 Duty 部分的全部设计，同时修改之前完成的3个ADT，使其适合与客户端交互，编写部分测试 | 按计划完成 |
| 2024-5.15 | 15:30-2:30 | ProcessIntervalSet、以及部分CourseIntervalSet的功能完成CourseIntervalSet与完成剩下的 APP | 按计划完成 |
| 2024-5.16 | 10:00-1:30 | 修改部分代码，完成 readme，完成实验报告 | 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 |
| 在考虑设计方案时，在方案5与方案6中选择犹豫不定，设计思路不确定，对实验规划不清楚。 | 初步尝试了方案5，但发现始终无法解决构造方法的问题，故切换至方案6。参考了往届学长学姐的解决方案，并最终确定项目结构和解决方案 |
| 通过键值对的形式保存时间段的起点和终点，但因为java 中hashmap的固有特性，导致起点相同的情况下后一次加入的键值对会覆盖前一次加入的键值对 | 查阅资料后，发现似乎可以考虑使用Java中自带的另一种Map: IdentityHashMap，因其可以做到键可重复，但在实际使用过程中发现不行，深入研究IdentityHashMap的源码发现，其只是对键是否重复的定义和HashMap有所区别：a==b为true时，其仍然会使后添加的键值对覆盖之前添加的键值对。故只能回到最原始的方法，自己构造一个List<Long>，并随时检查其长度为2。 |
| 正则表达式的使用不熟练 | 参考网上的示例，了解了正则表达式的基本语法，通过正则表达式在线测试程序保证了代码的正确性。并且总结了正则表达式的内容，上传了博客。 |
| DutyIntervalSet 设计时遇到的算法问题 | 通过手动推到执行过程，并且适当使用 gpt 最终修改得到了正确的算法 |

# 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

前所未有的复杂，极致的煎熬，十分的折磨，加上复习算法的压力，总算是赶完了实验，现在是5.16 的12:33 昨晚睡了四个小时，现在头疼厉害。设计思路极其复杂，即便是从设计方案 1～6 当中选择了想对容易的方案 6，code 过程的困难也让人十分糟心。切身体会到了开发的痛苦。当然的确十分有意义，虽然过程是痛苦的，但是结束的时候十分开心，过程也对设计模式等内容有了很多新的认识和掌握，也算是一种变相的复习，并且 前面的 ADT 等内容也重新回顾了很多。

## 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编程复用性更高，而面向应用编程针对每个具体应用，虽然思路清晰，但复用性低，的确在实验中设计的 ADT 能在大多数情况下得到复用，只是部分方法仍需要重写，比如 API 的复用程度就过低，和后面具体实现的 ADT 的格式的差异导致的复用困难，需要重写。

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？

撰写Spec，RI，AF等可以帮助理清思路，极大提高代码的可读性，特别是在类中包含众多方法的情况下，可以极大地提高查找的效率以及编程的体验，也是为了代码阅读的方便，在其他人阅读自己的代码的时候也可以迅速的掌握代码的主要内容，这点是十分重要的，以后的类似的编程仍然会这么多，并且更多的是在团队项目。

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需要考虑接受数据的局限性，但单独编写API可以使项目整体的架构严谨，难度当然是十分的高，需要处理各种各样的异常输入和请求，带来了极大的安全性上的挑战，只能说幸好这是实验而不是具体的在公司里的代码开发......乐趣的话，苦中作乐吧(┬＿┬)。

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？

语法驱动编程是一种提高代码复用性的好方法。在具体设计时较为繁琐，需要对正则表达式有一定的掌握，当然语法驱动编程就在于虽然设计繁琐，但是编码容易，可以更好的对文件的结构进行分析。

1. Lab1和Lab2的工作都不是从0开始，而是基于他人给出的设计方案和初始代码。本次实验是你完全从0开始进行ADT的设计并用OOP实现，经过三周之后，你感觉“设计ADT”的难度主要体现在哪些地方？你是如何克服的？

设计ADT的主要难度在于构思。需要考虑怎样抽象化，怎样设计函数的参数与返回值，并理清调用与委派等的关系。通过有初步想法后动手实现，发现差别再重新修改设计来逐步克服设计ADT的困难。十分十分困难，容易在后期的代码实现中认识到过去设计的不足，反而要去修改过去的代码，导致编码的复杂程度大大提升。

1. “抽象”是计算机科学的核心概念之一，也是ADT和OOP的精髓所在。本实验的三个应用既不能完全抽象为同一个ADT，也不是完全个性化，如何利用“接口、抽象类、类”三层体系以及接口的组合、类的继承、委派、设计模式等技术完成最大程度的抽象和复用，你有什么经验教训？

最好在一开始就构思好整体的流程、思路和框架，当然想要完完全全的设计好是几乎不可能的，人脑毕竟不能很好的预支这些东西，但是通过更多的代码设计，在更多的训练下，可以对类似的设计有更好的第六感，也会对 ADT 的设计有更好的掌握

1. 关于本实验的工作量、难度、deadline。

工作量极大，代码量超过 4k，实验二 3.5k，略有提升，难度也相当程度的难，很难很好的增加代码的复用程度，项目结构一复杂后容易对过去的 code 造成遗忘。deadline 很紧很紧，即便是延迟了一周，这两周也是正好一周一门考试，还有总共 3 个 deadline，4个实验，时间十分超级紧迫.

1. 下周就要进行考试了，你对《软件构造》课程总体评价如何？

对编程思维有较大提升，实验的编程训练也是非常必要的。但课程的ddl确实很紧，如果在不降低标准的情况下任务安排能够再宽松一些就更好了。

请允许我在实验的最后写下我自己想说的话。感谢！

