哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

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

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）。

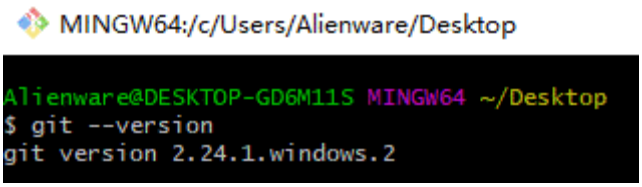
2 实验环境配置

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

3 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！）。

3.1 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

3.1.1 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。

3.1.2 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。

3.1.3 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。

3.1.4 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。

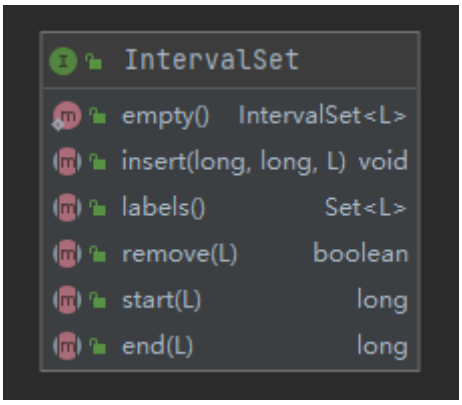
3.2 面向可复用性和可维护性的设计：IntervalSet<L>

该节是本实验的核心部分。

3.2.1 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()

```
public static<L> IntervalSet<L> empty()
{
    return new CommonIntervalSet<>();
}
```

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和标签
 * @param start the start
 * @param end the end
 * @param label a label
 */
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();

```
/**
 * 获得当前对象中的标签集合
 * @return 当前对象中标签的集合
 */
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
 * @param label a label
 * @return 是否成功移除当前对象label对应的标签关联的时间段
 */
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);

```
/**
 * 获得某个标签对应的时间段的结束时间
 * @param label a label
 * @return label标签对应的时间段的结束时间
 */
public long end(L label);
```

获得某个标签所对应的时间段的结束时间。

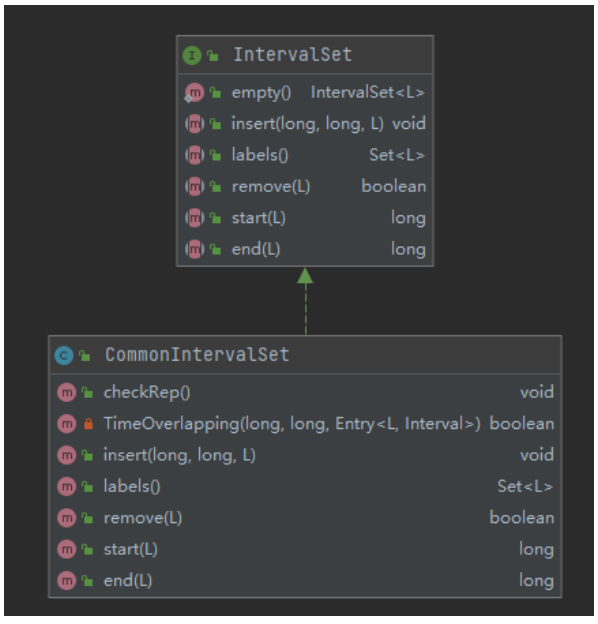
- public long start(L label);

```
/**
 * 获得某个标签对应的时间段的开始时间
 * @param label a label
 * @return label标签对应的时间段的开始时间
 */
public long start(L label);
```

获得某个标签所对应的时间段的开始时间。

3.2.2 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。

```
protected Map<L, Interval> IntervalMap;
```

Override 方法：

```
@Override
public void insert(long start, long end, L label) throws IntervalConflictException {...}

@Override
public Set<L> labels() {...}

@Override
public boolean remove(L label) { return this.IntervalMap.remove(label) != null; }

@Override
public long start(L label) { return this.IntervalMap.get(label).getStart(); }

@Override
public long end(L label) { return this.IntervalMap.get(label).getEnd(); }
```

私有方法：

- pulibc void checkRep();

```
public void checkRep()
{
    for(Map.Entry<L,Interval> entry:IntervalMap.entrySet())
    {
        assert entry.getValue().getStart()>=0;
        assert entry.getValue().getEnd()>entry.getValue().getStart();
        assert TimeOverlapping(entry.getValue().getStart(),entry.getValue().getEnd(),entry);
    }
}
```

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

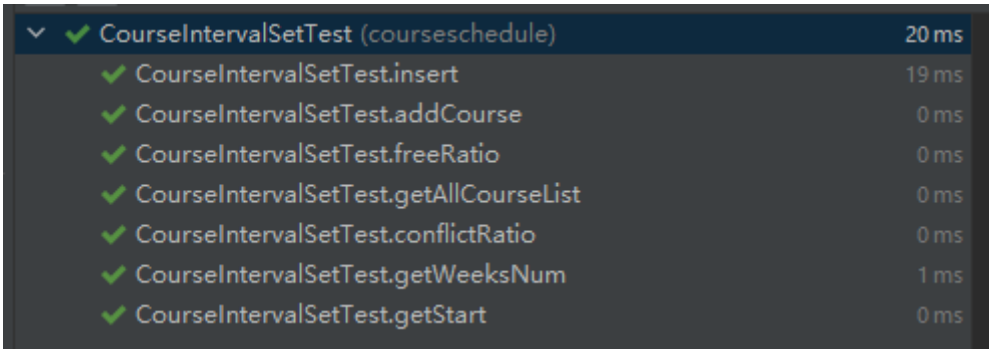
- private boolean TimeOverlapping(long start,long end,Map.Entry<L,Interval> entry)

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：

```java
public class CourseIntervalSetTest {

    @Test
    public void freeRatio() {...}

    @Test
    public void conflictRatio() {...}

    @Test
    public void getAllCourseList() {...}

    @Test
    public void getWeeksNum() {...}

    @Test
    public void getStart() {...}

    @Test
    public void addCourse() {...}

    @Test
    public void insert() {...}
}
```



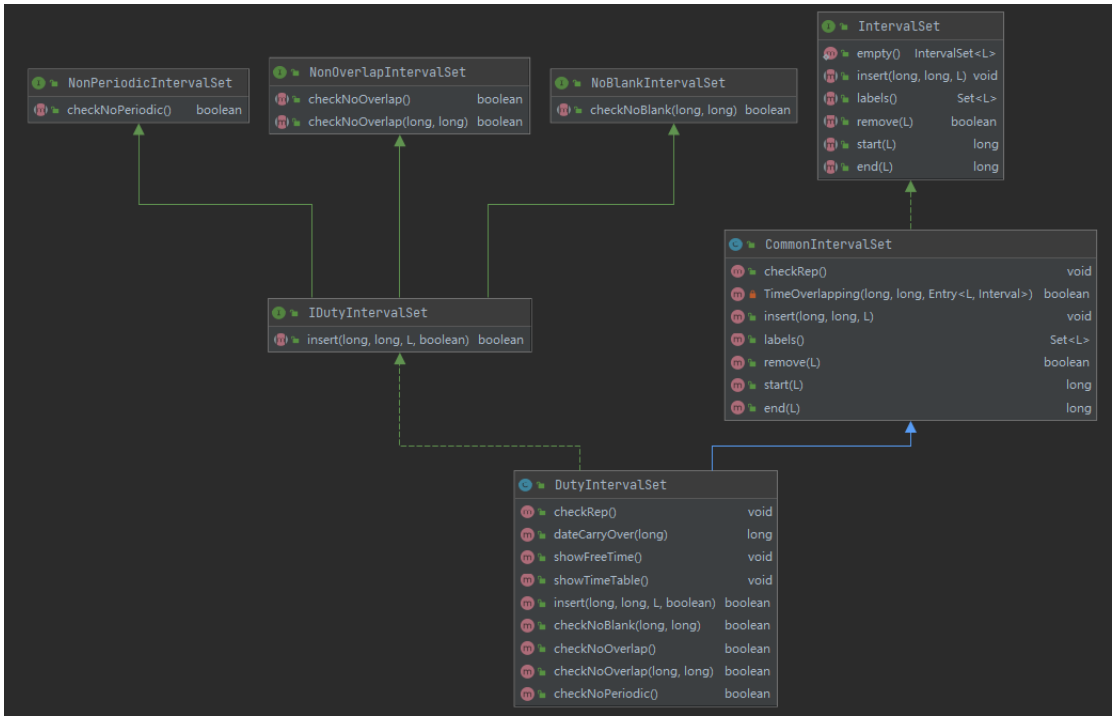
通过所有测试，并覆盖了 CourseIntervalSet 中的所有方法。

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）

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：

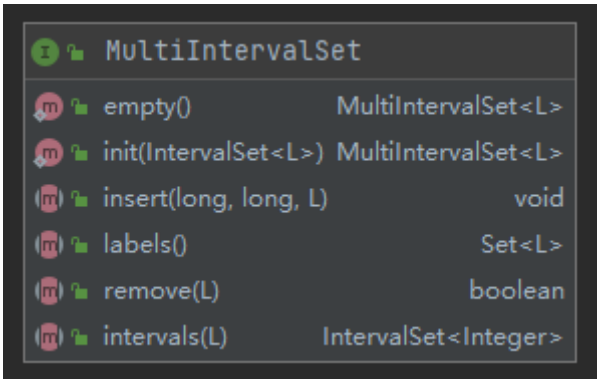


3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L>

3.3.1 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;

```
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();

```
public Set<L> labels();
```

获得当前对象中的标签的集合

- public boolean remove(L label);

```
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

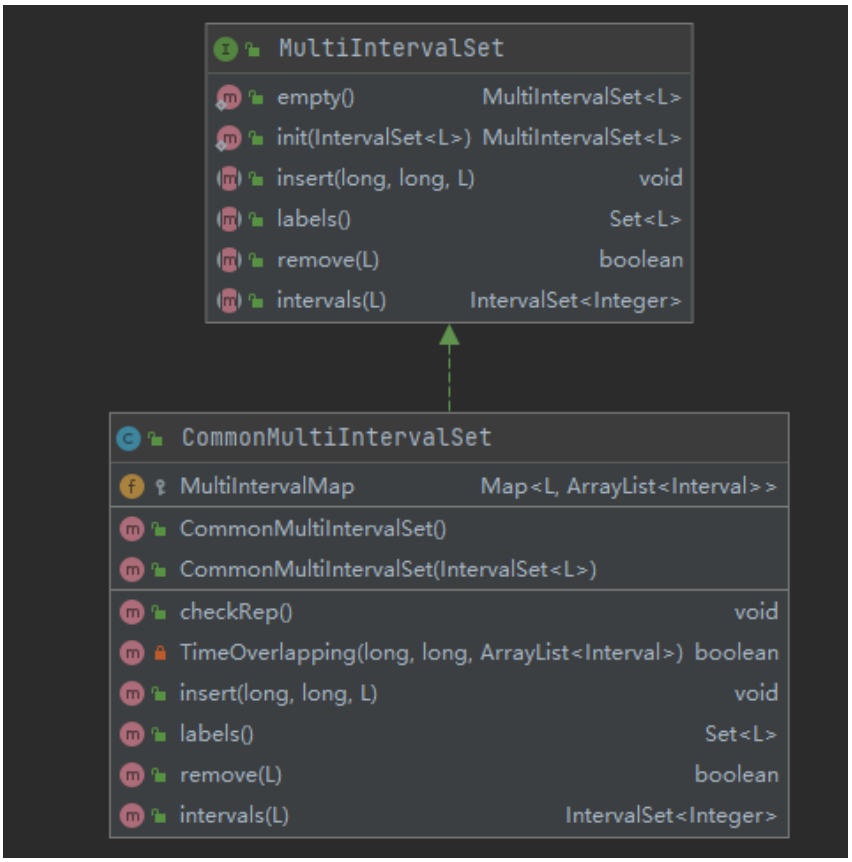
- public IntervalSet<Integer> intervals(L label);

```
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

3.3.2 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。

```
protected Map<L, ArrayList<Interval>> MultiIntervalMap;
```

Override 方法：

```
@Override
public void insert(long start, long end, L label) throws IntervalConflictException {...}


@Override
public Set<L> labels() {...}


@Override
public boolean remove(L label) { return MultiIntervalMap.remove(label) != null; }

@Override
public IntervalSet<Integer> intervals(L label) {...}
```

私有方法：

- public void checkRep()

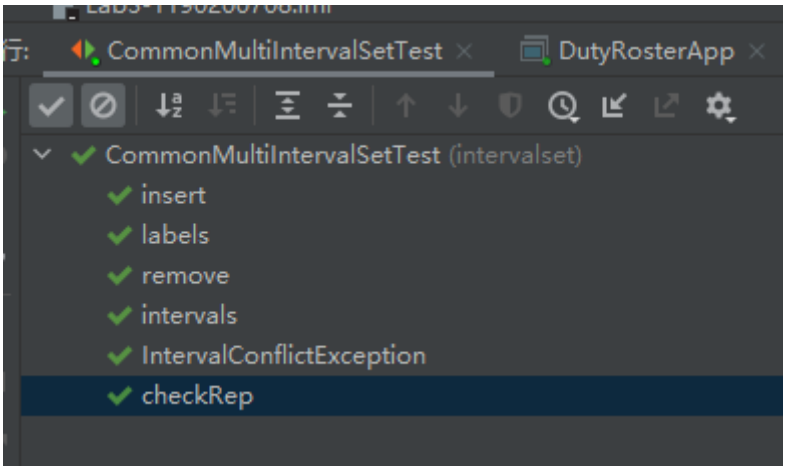
```
public void checkRep() {
    List<L> listL = new ArrayList<>();
    for (Map.Entry<L, ArrayList<Interval>> entry : MultiIntervalMap.entrySet()) {
        assert entry.getKey() != null;
        assert !(listL.contains(entry.getKey()));
        listL.add(entry.getKey());
        for (Interval interval : entry.getValue()) {
            assert interval.getStart() < interval.getEnd();

        }
    }
}
```

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。

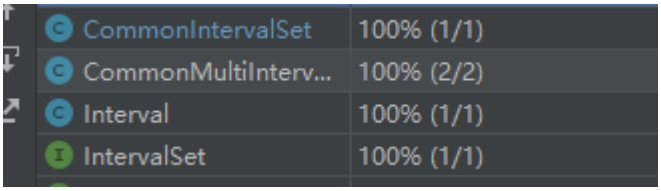
// Abstraction function:
// 由 Map 构成的映射关系
// 即从 L 到 ArrayList<Interval>的映射构成的 IntervalSet
// Representation invariant:
// ArrayList 中不存在重叠的时间
// label 不为空
// Map 中不存在相同的 label
// Safety from rep exposure:
// 使用 private、final 修饰变量
// 采用防御性复制

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

```
11  public class CommonMultiIntervalSetTest {
12
13      @Test
14      public void checkRep() {...}
34
35      @Test
36      public void insert() {...}
67
68      @Test
69      public void labels() {...}
87
88      @Test
89      public void remove() {...}
105
106     @Test
107     public void intervals() {...}
145     @Test
146     public void IntervalConflictException()
147     {...}
159
```



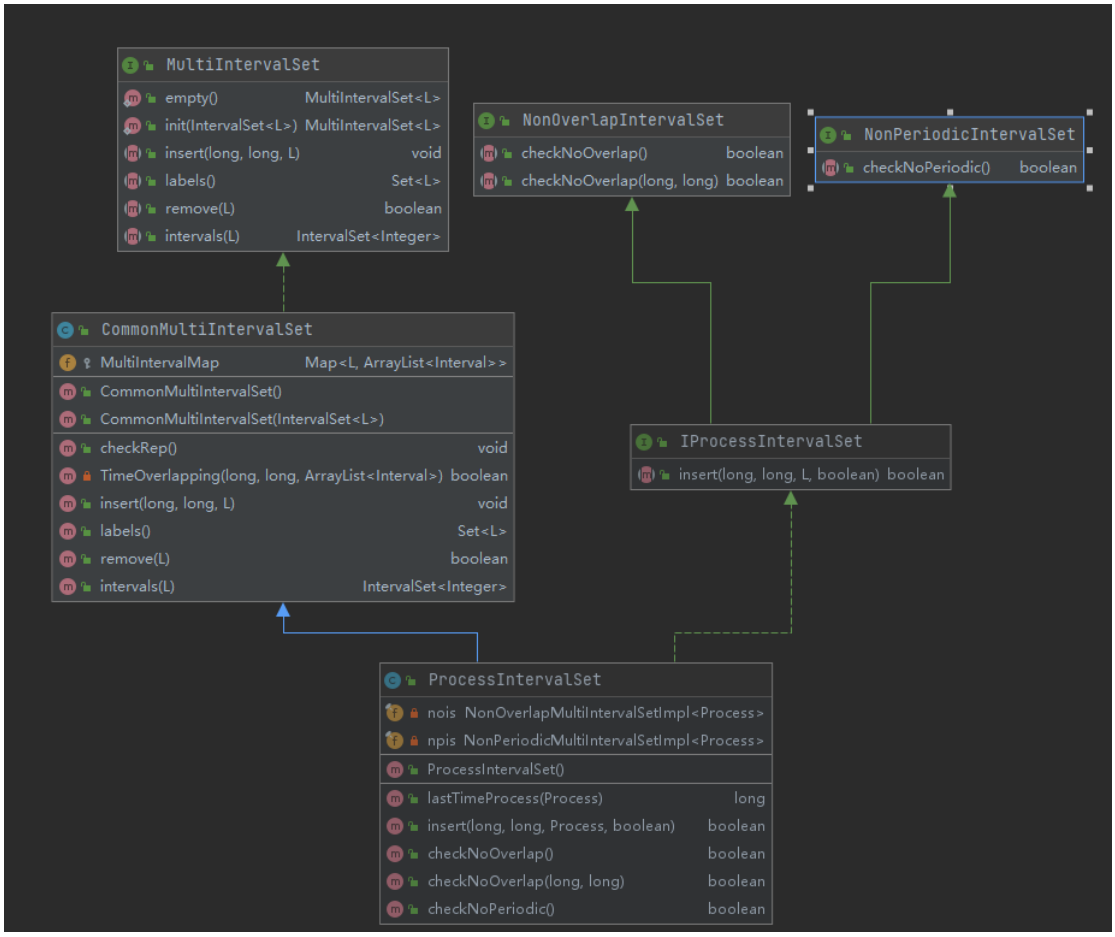
通过所有测试，并覆盖了 CourseMultiIntervalSet 中的所有方法。



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）

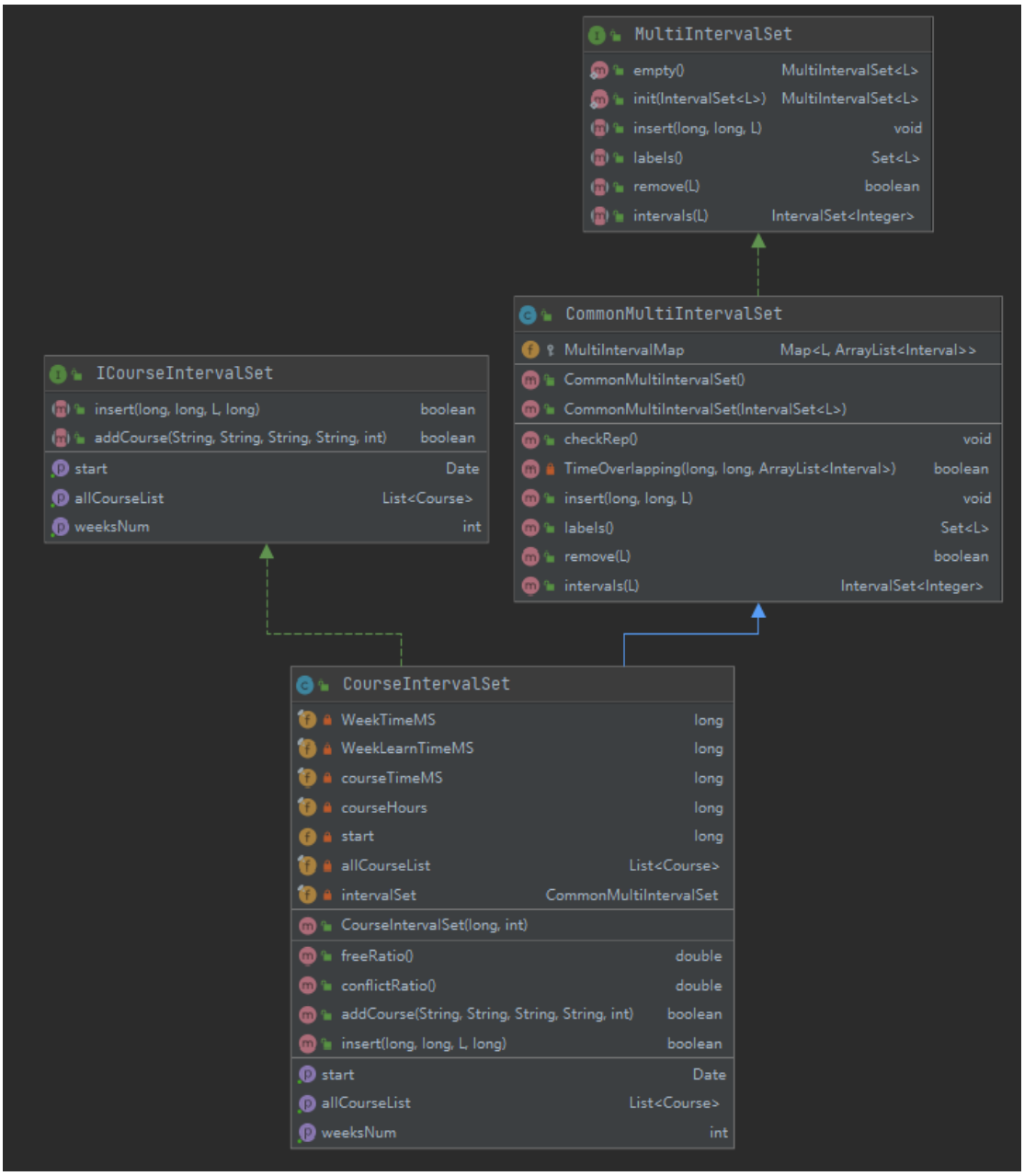
通过上述分析，详见 3.1.可以得出，只有进程管理系统、大学课表管理系统需要使用 MultiIntervalSet。因此需要分别对子类设置为无重叠时间委托、无周期时间委托和不设置委托。需要对每个维度上的不同特征值定义不同的接口，在接口中定义特殊的操作，各应用的具体子类根据自己需要实现不同特征的接口。分别实现该温度接口的不同实现类，实现该组合接口。在应用子类内，通过 delegation 到外部的每个维度上实现类的相应特殊操作。NonOverlapIntervalSet 是一个 mutable 的 ADT，用于在 IntervalSet 的基础上，添加“时间段”不可重叠的性质，每个时间段附着一个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

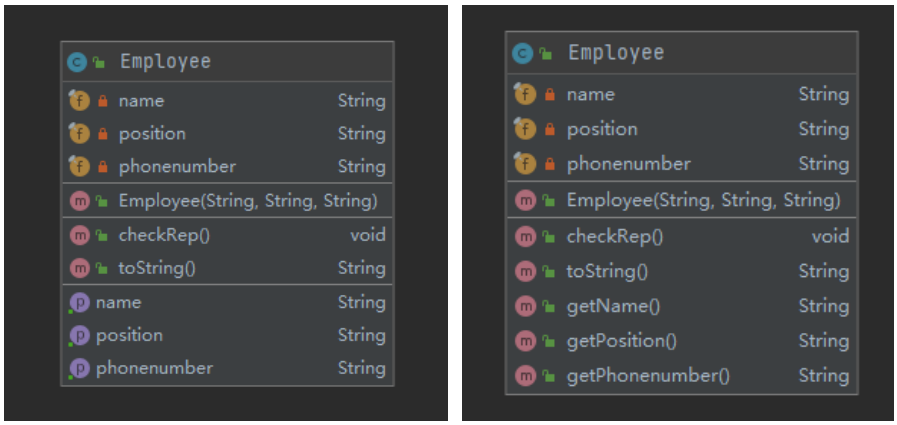


3.4 面向复用的设计：L

3.4.1 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。

3.4.2 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);

```
public Employee(String name,String position,String phonenumber)
{
    this.name = name;
    this.position = position;
    this.phonenumber = phonenumber;
    checkRep();
}
```

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()

```
public void checkRep()
{
    assert this.name!=null;
    assert this.position!=null;
    assert this.phonenumber!=null;
    String regExp = "^1(3[0-9]|4[01456879]|5[0-35-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\\d{8}$";
    Pattern pattern = Pattern.compile(regExp);
    Matcher matcher = pattern.matcher(this.phonenumber);
    assert  matcher.find();
}
```

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()

```
@Override
public String toString()
{
    String EmployeetoString = new String( original: this.name+"\t"+this.position+"\t"+this.phonenumber);
    return EmployeetoString;
}
```

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()

```
public String getName()
{
    return new String(this.name);
}
```

获得员工的姓名，返回员工的姓名

- public String getPosition()

```
public String getPosition() { return new String(this.position); }
```

获得员工的职位，员工的职位

- public String getPhonenumber()

```
public String getPhonenumber() { return new String(this.phonenumber); }
```

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

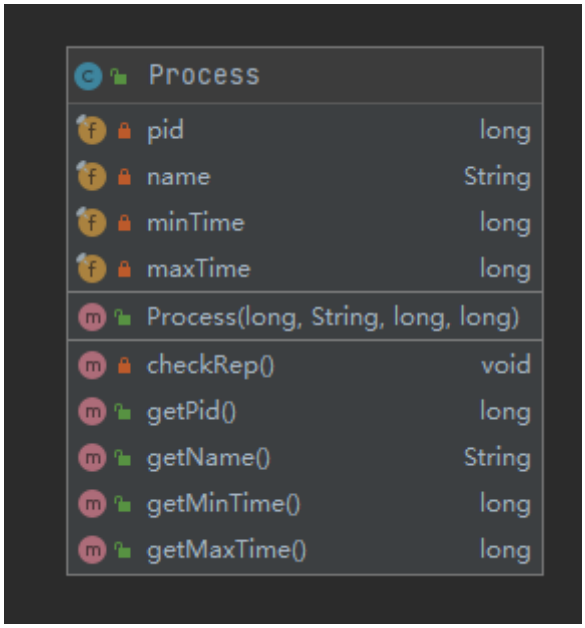
测试：通过分区测试，对每一个分支进行详尽的测试，由于测试逻辑较为简单，且只有返回字符串等基本操作，因此不详细展开测试策略。

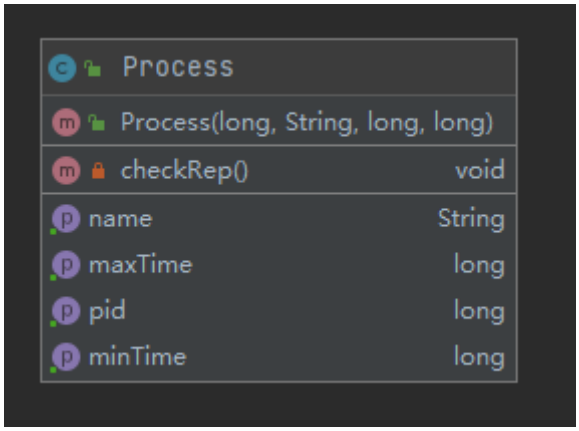
```
@Test
public void checkRep() {...}

@Test
public void testToString() {...}

@Test
public void getName() {...}

@Test
public void getPosition() {...}

@Test
public void getPhonenumber() {...}
```

可以看到全部通过测试，并且覆盖率为 100%.

元素	类(%)	方法(%)	行(%)
DutyIntervalSet	0% (0/1)	0% (0/11)	0% (0/89)
Employee	100% (1/1)	100% (7/7)	100% (20/...

3.4.3 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)

```
public Process(final long pid, final String name, final long minTime, final long maxTime) {
    this.pid = pid;
    this.name = name;
    this.minTime = minTime;
    this.maxTime = maxTime;
    checkRep();
}
```

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();

```
public long getPid() { return pid; }
```

获取进程的 Pid。

- public String getName();

```
public String getName() { return name; }
```

获取进程的名称。

- public long getMinTime();

```
public long getMinTime() { return minTime; }
```

获取进程的最小执行时间。

- public long getMaxTime();

```
public long getMaxTime() { return maxTime; }
```

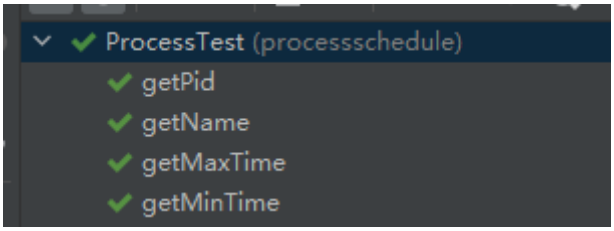
获取进程的最大执行时间。

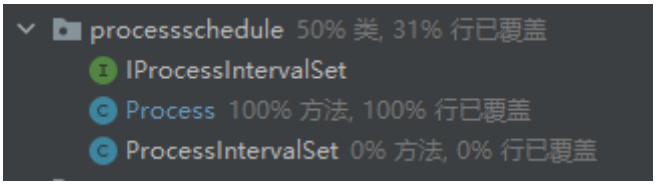
rep、AF、RI、Safety from Rep Exposure 说明如下：

```
// 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
```

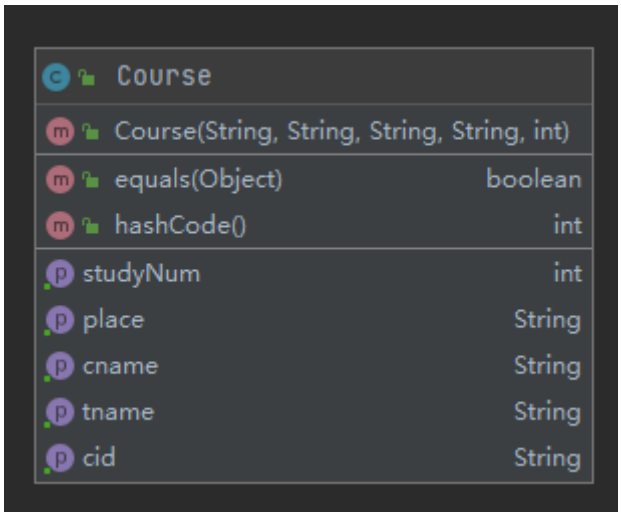
测试：通过分区测试，对每一个分支进行详尽的测试，由于测试逻辑较为简单，且只有返回字符串等基本操作，因此不详细展开测试策略。

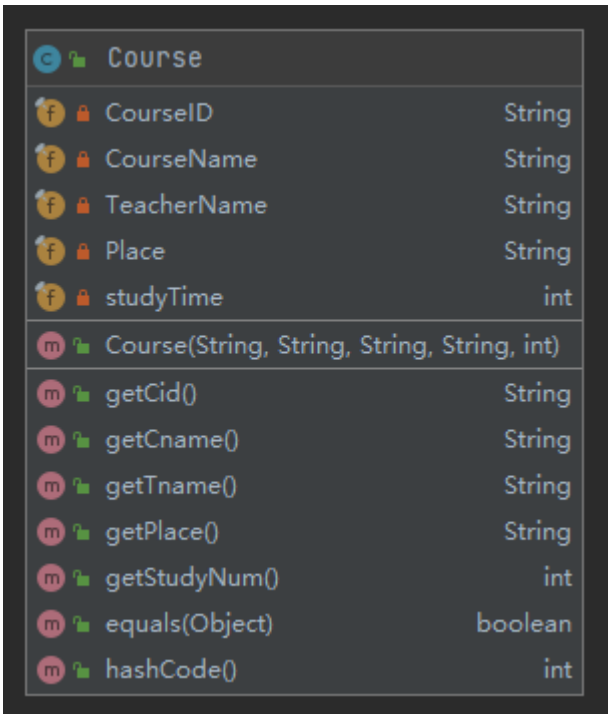
可以看到全部通过测试，并且覆盖率为 100%.





3.4.4 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);

```
public Process(final long pid, final String name, final long minTime, final long maxTime) {
    this.pid = pid;
    this.name = name;
    this.minTime = minTime;
    this.maxTime = maxTime;
    checkRep();
}
```

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()

```
public String getCid() { return CourseID; }
```

获得课程的 ID。

- public String getCname()

```
public String getCname() { return CourseName; }
```

获得课程的名称。

- public String getTname()

```
public String getTname() { return TeacherName; }
```

获得教师的名称。

- public String getPlace()

```
public String getPlace() { return Place; }
```

获得授课的地点。

- public int getStudyNum()

```
public int getStudyNum() { return studyTime; }
```

获得每周授课的学时。

- public boolean equals(Object object)

```java
@Override
public boolean equals(Object object) {
    if(this == object) return true;
    if(object == null) return false;
    Course CourseObject = (Course) object;
    return Objects.equals(this.CourseName, CourseObject.CourseName) &&
            Objects.equals(this.TeacherName, CourseObject.TeacherName) &&
            Objects.equals(this.Place, CourseObject.Place) &&
            Objects.equals(this.CourseID, CourseObject.CourseID);

}
```

判断两个 Course 是否相同。

- public int hashCode()

```java
@Override
public int hashCode() {
    return this.CourseName.hashCode()
            * this.CourseID.hashCode()
            * this.TeacherName.hashCode()
            * this.Place.hashCode()*this.studyTime;
}
```

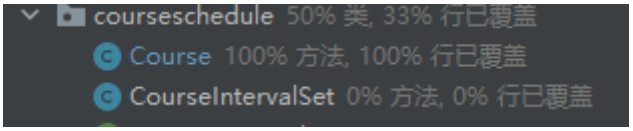
计算 Course 的 hashCode。

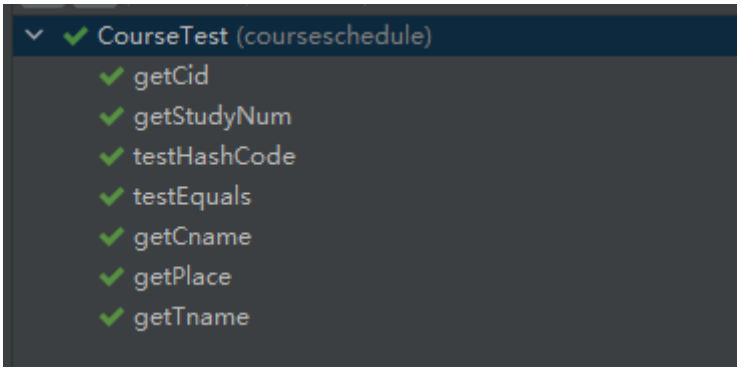
rep、AF、RI、Safety from Rep Exposure 说明如下：

// 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%.

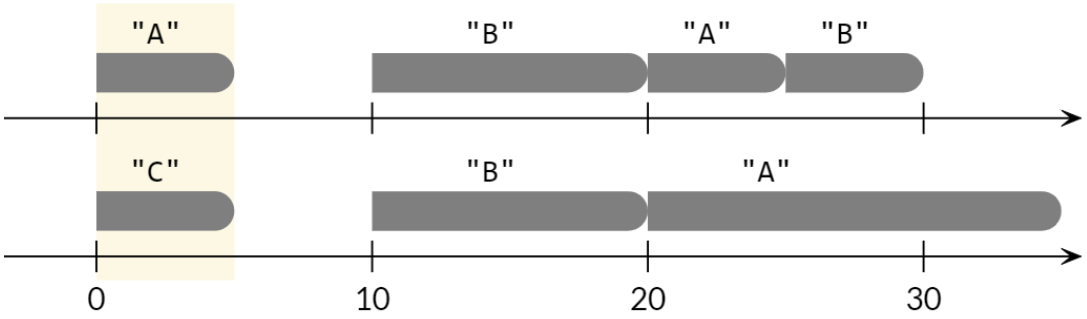




3.5 可复用 API 设计

3.5.1 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

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

获得 MultiIntervalSet 的最晚的时间

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

计算 s1 中的所有时间段的时间的长度

4. 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：

```
public static <L> double Similarity(MultiIntervalSet<L> s1, MultiIntervalSet<L> s2) {
    long startTimerShaft = Math.min(getStartTimerShaft(s1), getStartTimerShaft(s2));
    long endTimerShaft = Math.max(getEndTimeShaft(s1), getEndTimeShaft(s2));
    double lastTime = (double) endTimerShaft - (double) startTimerShaft;

    if (lastTime == 0)
        return 0;
    double timeSimilarity = 0;
    // 大的标签
    // 指ABC
    for (L label : s1.labels()) {
        if (s2.labels().contains(label)) {
            timeSimilarity += intervalLength(similarIntervals(s1.intervals(label), s2.intervals(label)));
        }
    }
    double ratio = timeSimilarity / lastTime;
    return ratio;
}
```

3.5.2 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。

3.5.3 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。

3.6 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。

3.6.1 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：

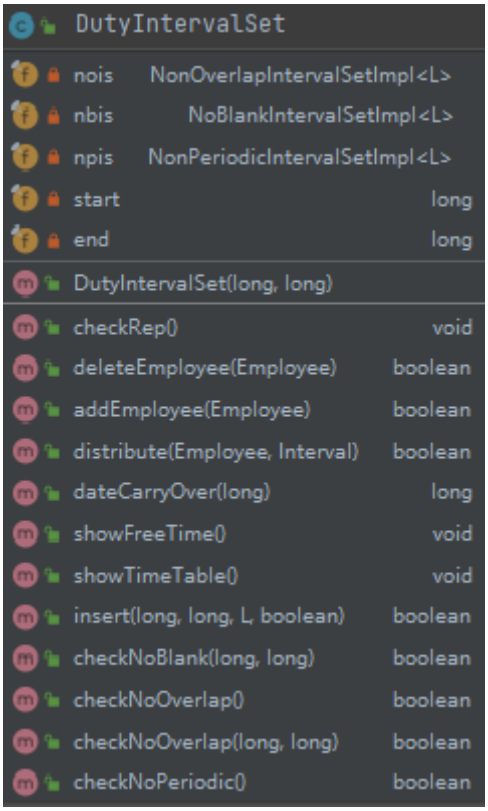
Step 1　设定排班开始日期、结束日期，具体到年月日即可。

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);

```
public DutyIntervalSet(long start,long end){
    //设置delegation关系
    this.IntervalMap = new HashMap<>();
    this.nbis = new NoBlankIntervalSetImpl<>(this.IntervalMap);
    this.npis = new NonPeriodicIntervalSetImpl<>(this.IntervalMap);
    this.nois = new NonOverlapIntervalSetImpl<>(this.IntervalMap);
    this.start = start;
    this.end = end;
    checkRep();
}
```

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

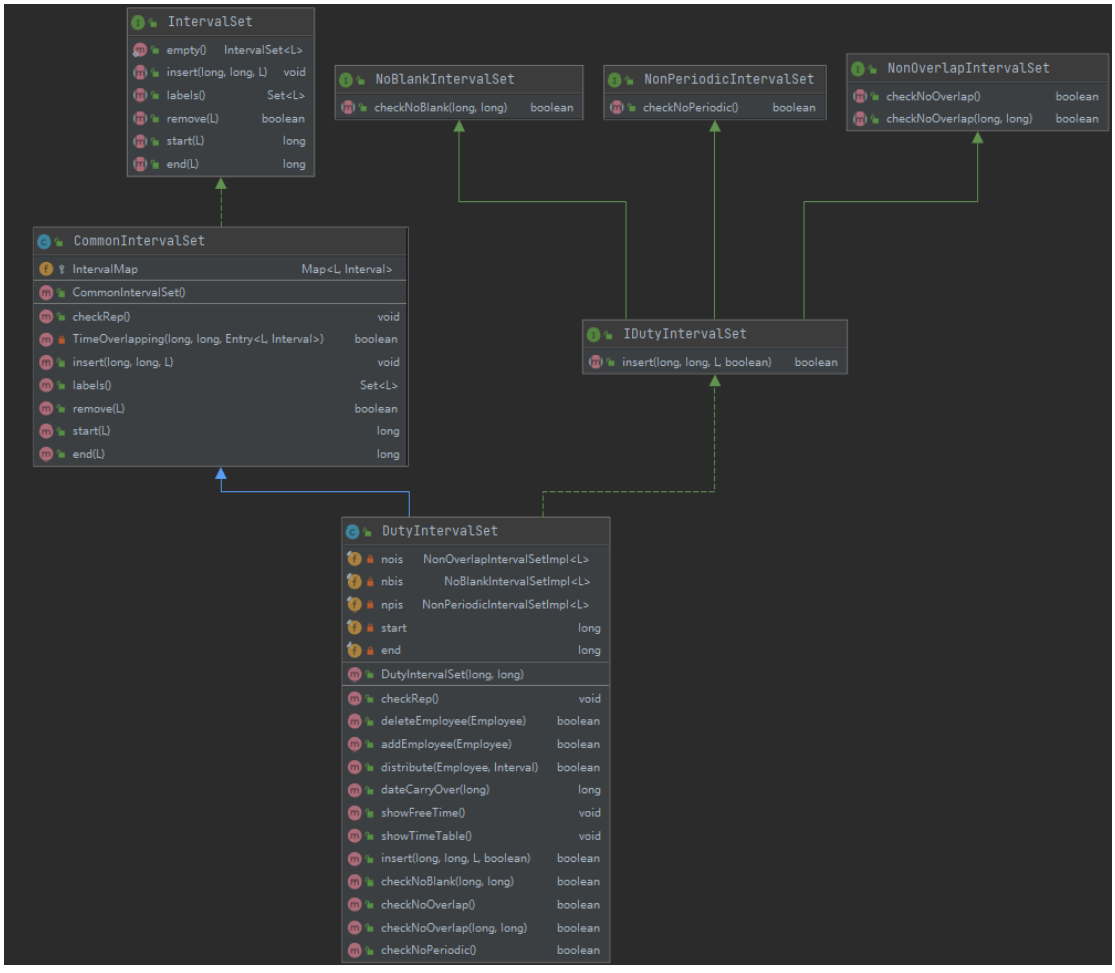
- public void showTimeTable()

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：

```
请输入您的选项:
2
请输入起始时间和截止时间:(eg:2021-03-06,2021-03-06)
2021-04-06,2021-06-01
```

添加员工信息：

```
3
请输入待添加的员工人数:
2
请输入员工姓名:
熊峰
请输入员工职位:
学生
请输入员工手机号:
13114991114
请输入员工姓名:
张三
请输入员工职位:
学生
请输入员工手机号:
13111111111
```

显示所有员工信息：

```
请输入您的选项:
5
Employee List:
Name    熊峰 Position    学生 PhoneNumber 13114991114
Name    张三 Position    学生 PhoneNumber 13111111111
```

为员工分配值班时间

```
请为员工分配值班时间
请输入待分配的员工的姓名:
张三
请输入待分配的员工的职位:
学生
请输入员工值班时间:(eg:2021-03-06,2021-03-06)
2021-04-06,2021-05-02
是否继续分配值班(y/Y):
n
```

检查当前分配是否合理：

```
请输入您的选项:
8
当前存在空闲时间!
```

检查当前空闲时间：

```
请输入您的选项:
7
当前未安排值班的时间为:
Sun May 02 00:00:00 CST 2021
Mon May 03 00:00:00 CST 2021
Tue May 04 00:00:00 CST 2021
Wed May 05 00:00:00 CST 2021
Thu May 06 00:00:00 CST 2021
Fri May 07 00:00:00 CST 2021
Sat May 08 00:00:00 CST 2021
Sun May 09 00:00:00 CST 2021
Mon May 10 00:00:00 CST 2021
Tue May 11 00:00:00 CST 2021
Wed May 12 00:00:00 CST 2021
Thu May 13 00:00:00 CST 2021
Fri May 14 00:00:00 CST 2021
Sat May 15 00:00:00 CST 2021
Sun May 16 00:00:00 CST 2021
Mon May 17 00:00:00 CST 2021
Tue May 18 00:00:00 CST 2021
Wed May 19 00:00:00 CST 2021
```

```
DutyRosterApp × DutyRosterApp ×
Sun May 23 00:00:00 CST 2021
Mon May 24 00:00:00 CST 2021
Tue May 25 00:00:00 CST 2021
Wed May 26 00:00:00 CST 2021
Thu May 27 00:00:00 CST 2021
Fri May 28 00:00:00 CST 2021
Sat May 29 00:00:00 CST 2021
Sun May 30 00:00:00 CST 2021
Mon May 31 00:00:00 CST 2021
当前未安排时间的比例为0.5357142857142857
```

当满足条件时：

```
请输入您的选项:
8
当前已满足分配条件!
```

3.6.2 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：

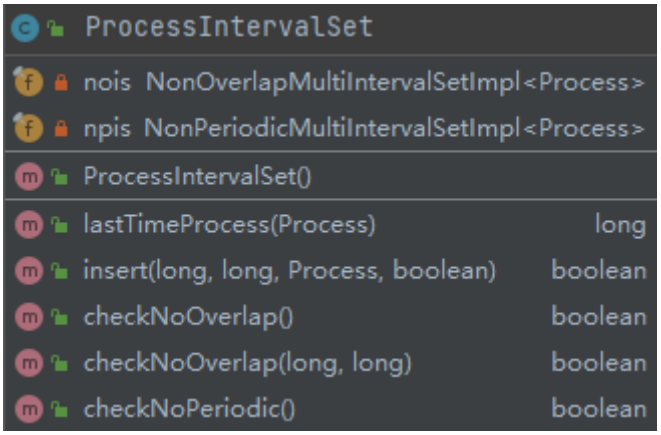
Step 1　增加一组进程，输入每个进程的 ID、名称、最短执行时间、最长执行时间；进程一旦设定无法再修改其信息。

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。

```
public ProcessIntervalSet()
{
    this.nois = new NonOverlapMultiIntervalSetImpl<>(this.MultiIntervalMap);
    this.npis = new NonPeriodicMultiIntervalSetImpl<>(this.MultiIntervalMap);

    checkRep();

}
```

ProcessIntervalSet 类有五个方法：

- public long lastTimeProcess(Process process)

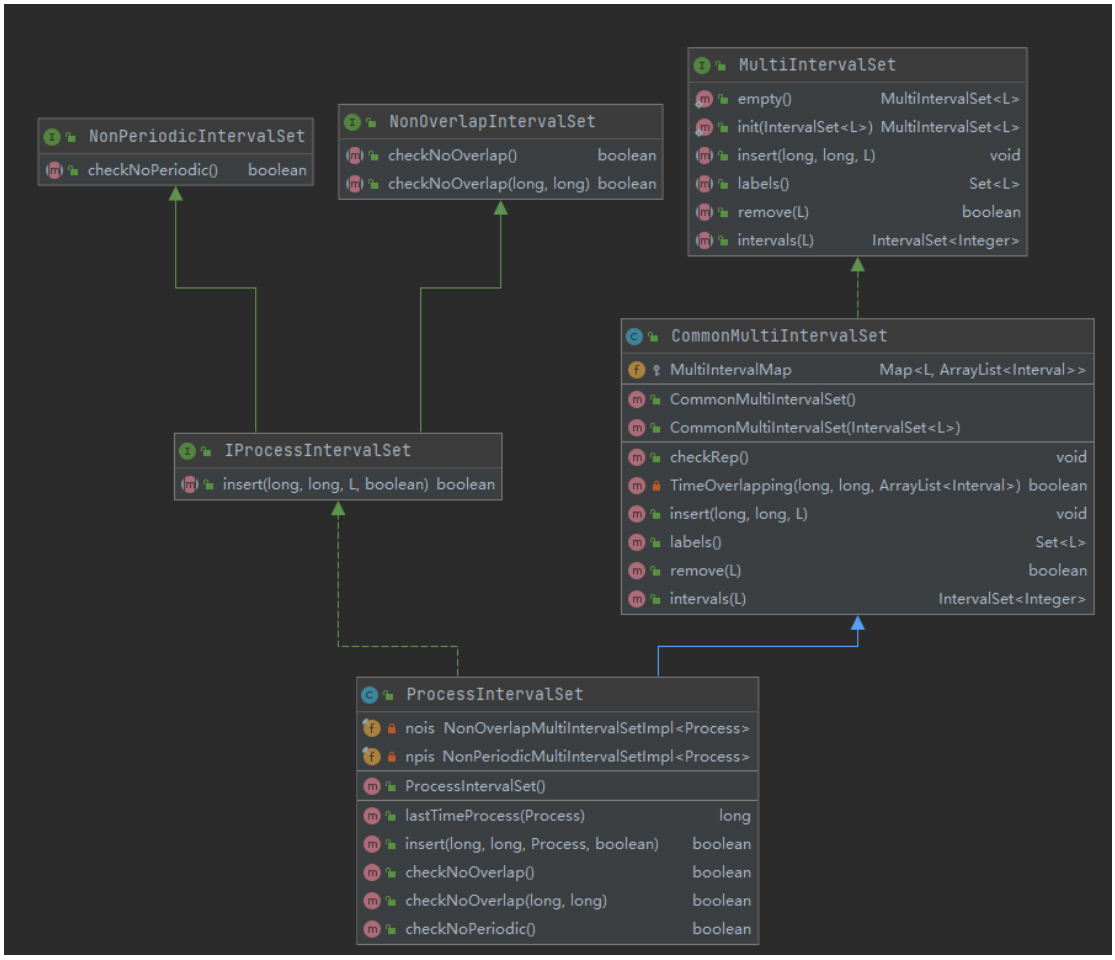
```
public long lastTimeProcess(Process process)
{
    long lastTime = 0;
    if(MultiIntervalMap.size()==0)
        return 0;
    if(!MultiIntervalMap.containsKey(process))
        return 0;
    if(MultiIntervalMap.get(process).size()==0)
        return 0;
    for(Interval interval:MultiIntervalMap.get(process))
    {
        lastTime += interval.getEnd()-interval.getStart();
    }
    return lastTime;
}
```

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：

```
请输入您的选项：
2
请输入进程名称：
a
请输入进程ID：
1
请输入进程最短执行时间：
12
请输入进程最长执行时间：
14
```

```
请输入您的选项：
2
请输入进程名称：
b
请输入进程ID：
2
请输入进程最短执行时间：
13
请输入进程最长执行时间：
14
```

模拟调度：

```
模拟调度(随机选择进程):
时间 进程名字  进程ID
0    空   空
1    a    1
2    a    1
3    a    1
4    a    1
5    a    1
6    a    1
7    a    1
8    a    1
9    a    1
10   a    1
11   a    1
12   a    1
13   b    2
14   b    2
15   b    2
16   b    2
17   b    2
18   b    2
19   b    2
20   b    2
21   b    2
22   b    2
23   b    2
24   空   空
25   b    2
26   b    2
```

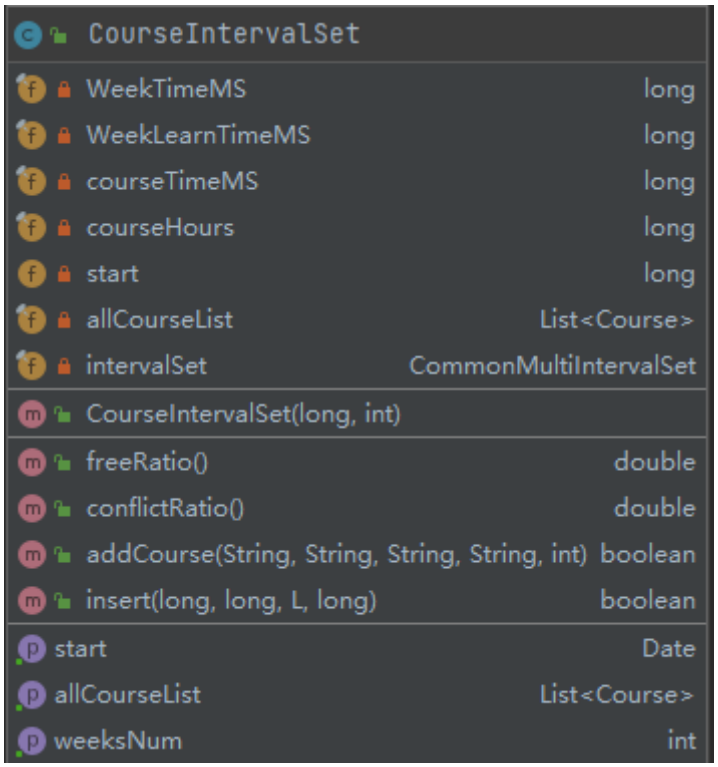
最短进程优先：

```
时间 进程名字  进程ID
0    空   空
1    b    2
2    b    2
3    b    2
4    b    2
5    b    2
6    b    2
7    b    2
8    b    2
9    a    1
10   a    1
11   空   空
12   空   空
13   a    1
14   a    1
15   a    1
16   a    1
17   a    1
18   a    1
19   a    1
20   a    1
21   a    1
22   a    1
23   a    1
```

3.6.3 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：

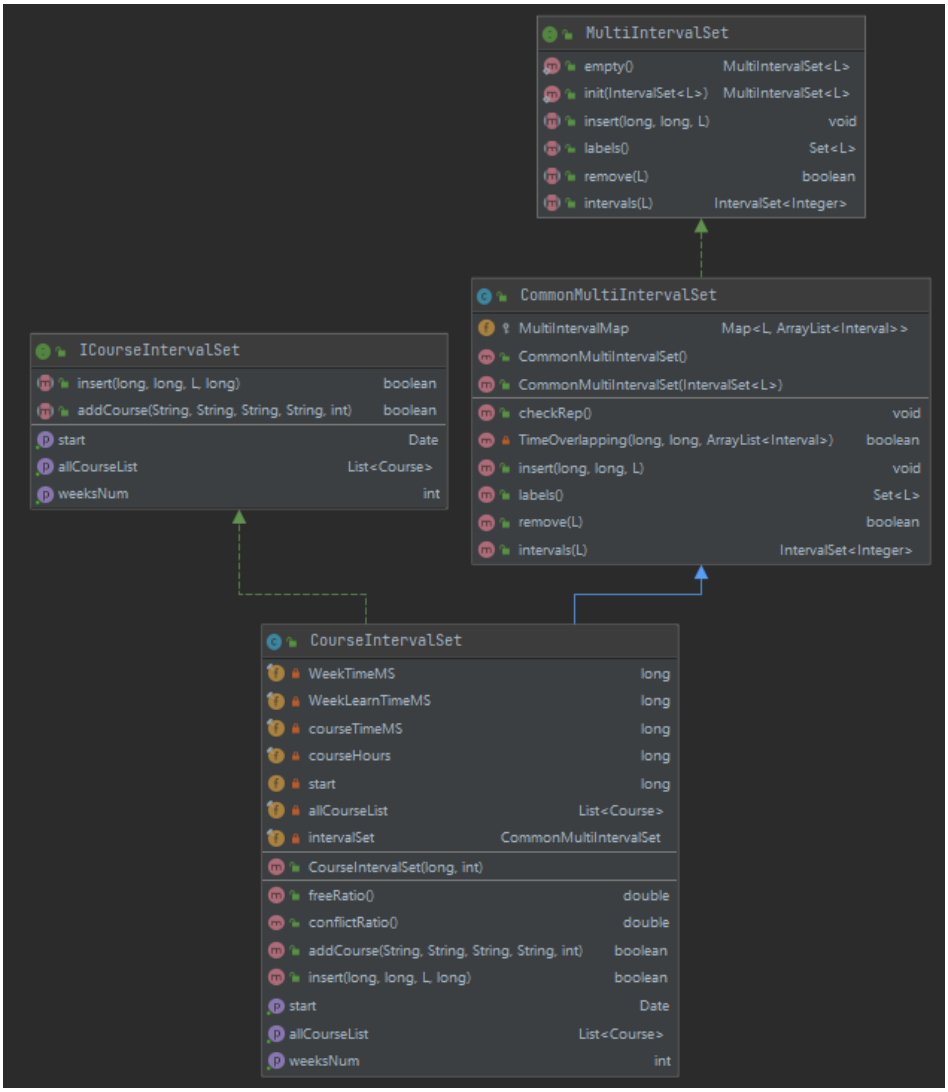
- public double freeRatio();

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：

```
请输入您的选项:
2
请输入学期开始时间(例:2021-01-02)
2021-01-01
请输入总周数(例:18)
18
设置学期信息成功!
```

设置一门课程：

```
请输入课程相关信息:
请输入课程ID:
1
请输入课程名称:
软件构造
请输入教师姓名:
陈惠鹏
请输入授课地点:
正心11
请输入周学时数(偶数)
4
```

```
请输入课程相关信息:
请输入课程ID:
2
请输入课程名称:
计算机系统
请输入教师姓名:
吴锐
请输入授课地点:
正心12
请输入周学时数(偶数)
4
```

设置一门课程的时间：

```
请输入您的选项:
4
请输入课程ID:
1
请输入课程名称:
软件构造
请输入教师姓名:
陈惠鹏
请输入授课地点:
正心11
请输入安排时间的开始:(yyyy-MM-dd hh:mm:ss)
2021-01-01 10:00:00
请输入安排时间的开始:(yyyy-MM-dd hh:mm:ss)
2021-01-01 11:45:00
请输入您的选项:
```

查看当前课程安排情况：

```
5
课程安排情况如下:
软件构造
Fri Jan 01 10:00:00 CST 2021->Fri Jan 01 11:45:00 CST 2021
计算机系统
请输入您的选项:
```

查看某天课表：

```
请输入您的选项:
8
请输入待查询的时间:(yyyy-MM-dd)
2021-01-01
软件构造  Fri Jan 01 10:00:00 CST 2021->Fri Jan 01 11:45:00 CST 2021
请输入您的选项:
8
请输入待查询的时间:(yyyy-MM-dd)
2021-01-08
软件构造  Fri Jan 08 10:00:00 CST 2021->Fri Jan 08 11:45:00 CST 2021
```

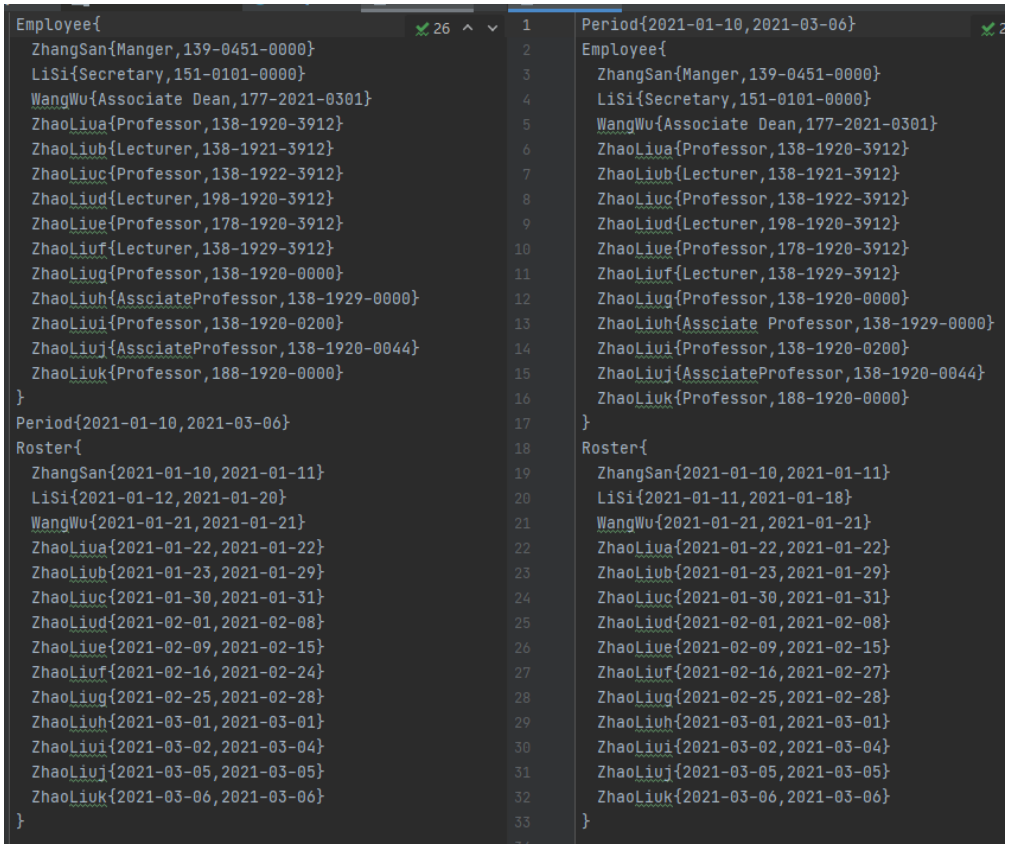
也可以查询每周冲突与空闲时间

```
6.每周空闲时间比例
7.每周重复时间比例
8.查看一天课表
请输入您的选项:
6
0.8640776699029126
请输入您的选项:
7
0.0
```

3.7 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。



```
Employee{
  ZhangSan{Manger,139-0451-0000}
  LiSi{Secretary,151-0101-0000}
  WangWu{Associate Dean,177-2021-0301}
  ZhaoLiua{Professor,138-1920-3912}
  ZhaoLiub{Lecturer,138-1921-3912}
  ZhaoLiuc{Professor,138-1922-3912}
  ZhaoLiud{Lecturer,198-1920-3912}
  ZhaoLiue{Professor,178-1920-3912}
  ZhaoLiuf{Lecturer,138-1929-3912}
  ZhaoLiug{Professor,138-1920-0000}
  ZhaoLiuh{AssciateProfessor,138-1929-0000}
  ZhaoLiui{Professor,138-1920-0200}
  ZhaoLiuj{AssciateProfessor,138-1920-0044}
  ZhaoLiuk{Professor,188-1920-0000}
}
Period{2021-01-10,2021-03-06}
Roster{
  ZhangSan{2021-01-10,2021-01-11}
  LiSi{2021-01-12,2021-01-20}
  WangWu{2021-01-21,2021-01-21}
  ZhaoLiua{2021-01-22,2021-01-22}
  ZhaoLiub{2021-01-23,2021-01-29}
  ZhaoLiuc{2021-01-30,2021-01-31}
  ZhaoLiud{2021-02-01,2021-02-08}
  ZhaoLiue{2021-02-09,2021-02-15}
  ZhaoLiuf{2021-02-16,2021-02-24}
  ZhaoLiug{2021-02-25,2021-02-28}
  ZhaoLiuh{2021-03-01,2021-03-01}
  ZhaoLiui{2021-03-02,2021-03-04}
  ZhaoLiuj{2021-03-05,2021-03-05}
  ZhaoLiuk{2021-03-06,2021-03-06}
}
```

```
Period{2021-01-10,2021-03-06}
Employee{
  ZhangSan{Manger,139-0451-0000}
  LiSi{Secretary,151-0101-0000}
  WangWu{Associate Dean,177-2021-0301}
  ZhaoLiua{Professor,138-1920-3912}
  ZhaoLiub{Lecturer,138-1921-3912}
  ZhaoLiuc{Professor,138-1922-3912}
  ZhaoLiud{Lecturer,198-1920-3912}
  ZhaoLiue{Professor,178-1920-3912}
  ZhaoLiuf{Lecturer,138-1929-3912}
  ZhaoLiug{Professor,138-1920-0000}
  ZhaoLiuh{Assciate Professor,138-1929-0000}
  ZhaoLiui{Professor,138-1920-0200}
  ZhaoLiuj{AssciateProfessor,138-1920-0044}
  ZhaoLiuk{Professor,188-1920-0000}
}
Roster{
  ZhangSan{2021-01-10,2021-01-11}
  LiSi{2021-01-11,2021-01-18}
  WangWu{2021-01-21,2021-01-21}
  ZhaoLiua{2021-01-22,2021-01-22}
  ZhaoLiub{2021-01-23,2021-01-29}
  ZhaoLiuc{2021-01-30,2021-01-31}
  ZhaoLiud{2021-02-01,2021-02-08}
  ZhaoLiue{2021-02-09,2021-02-15}
  ZhaoLiuf{2021-02-16,2021-02-27}
  ZhaoLiug{2021-02-25,2021-02-28}
  ZhaoLiuh{2021-03-01,2021-03-01}
  ZhaoLiui{2021-03-02,2021-03-04}
  ZhaoLiuj{2021-03-05,2021-03-05}
  ZhaoLiuk{2021-03-06,2021-03-06}
}
```

```java
public static boolean checkFileFormat(String string) {
    Pattern pattern1 = Pattern.compile(
            "Employee\\{\\s+[\n" +
                    "\u3000(A-Za-z+){a-zA-Z\\s,\\d3}\\-4]+\n" +
                    "}\n" +
                    "Period\\{(\\d{4}-\\d{2}-\\d{2})," +
                    "(\\d{4}-\\d{2}-\\d{2})}\n" +
                    "Roster\\{\\s+[\n" +
                    "\u3000(A-Za-z+){\\d4}\\-2,]+\n" +
                    "}\n"
    );
    Pattern pattern2 = Pattern.compile(
            "Period\\{\\d{4}-\\d{2}-\\d{2},\\d{4}-\\d{2}-\\d{2}}\n" +
                    "Employee\\{[\n" +
                    "\u3000A-Za-z+{A-Za-z\\s,\\d3}\\-4]+\n" +
                    "}+\n" +
                    "Roster\\{\\s+[\n" +
                    "\u3000A-Za-z+{\\d4}\\-2,]+\n" +
                    "}\n"
    );
    Matcher matcher1 = pattern1.matcher(string);
    Matcher matcher2 = pattern2.matcher(string);
    return matcher1.matches() || matcher2.matches();
}
```

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：

```
日期                          值班人姓名    职位 手机号
Sun Jan 10 00:00:00 CST 2021    ZhangSan    Manger  139-0451-0000
Mon Jan 11 00:00:00 CST 2021    ZhangSan    Manger  139-0451-0000
Tue Jan 12 00:00:00 CST 2021    LiSi    Secretary   151-0101-0000
Wed Jan 13 00:00:00 CST 2021    LiSi    Secretary   151-0101-0000
Thu Jan 14 00:00:00 CST 2021    LiSi    Secretary   151-0101-0000
Fri Jan 15 00:00:00 CST 2021    LiSi    Secretary   151-0101-0000
Sat Jan 16 00:00:00 CST 2021    LiSi    Secretary   151-0101-0000
Sun Jan 17 00:00:00 CST 2021    LiSi    Secretary   151-0101-0000
Mon Jan 18 00:00:00 CST 2021    LiSi    Secretary   151-0101-0000
Tue Jan 19 00:00:00 CST 2021    LiSi    Secretary   151-0101-0000
Wed Jan 20 00:00:00 CST 2021    LiSi    Secretary   151-0101-0000
Thu Jan 21 00:00:00 CST 2021    WangWu  AssociateDean   177-2021-0301
Fri Jan 22 00:00:00 CST 2021    ZhaoLiua    Professor   138-1920-3912
Sat Jan 23 00:00:00 CST 2021    ZhaoLiub    Lecturer    138-1921-3912
Sun Jan 24 00:00:00 CST 2021    ZhaoLiub    Lecturer    138-1921-3912
Mon Jan 25 00:00:00 CST 2021    ZhaoLiub    Lecturer    138-1921-3912
Tue Jan 26 00:00:00 CST 2021    ZhaoLiub    Lecturer    138-1921-3912
Wed Jan 27 00:00:00 CST 2021    ZhaoLiub    Lecturer    138-1921-3912
Thu Jan 28 00:00:00 CST 2021    ZhaoLiub    Lecturer    138-1921-3912
Fri Jan 29 00:00:00 CST 2021    ZhaoLiub    Lecturer    138-1921-3912
Sat Jan 30 00:00:00 CST 2021    ZhaoLiuc    Professor   138-1922-3912
Sun Jan 31 00:00:00 CST 2021    ZhaoLiuc    Professor   138-1922-3912
Mon Feb 01 00:00:00 CST 2021    ZhaoLiud    Lecturer    198-1920-3912
Tue Feb 02 00:00:00 CST 2021    ZhaoLiud    Lecturer    198-1920-3912
Wed Feb 03 00:00:00 CST 2021    ZhaoLiud    Lecturer    198-1920-3912
Thu Feb 04 00:00:00 CST 2021    ZhaoLiud    Lecturer    198-1920-3912
Fri Feb 05 00:00:00 CST 2021    ZhaoLiud    Lecturer    198-1920-3912
Sat Feb 06 00:00:00 CST 2021    ZhaoLiud    Lecturer    198-1920-3912
Sun Feb 07 00:00:00 CST 2021    ZhaoLiud    Lecturer    198-1920-3912
Mon Feb 08 00:00:00 CST 2021    ZhaoLiud    Lecturer    198-1920-3912
Tue Feb 09 00:00:00 CST 2021    ZhaoLiue    Professor   178-1920-3912
Wed Feb 10 00:00:00 CST 2021    ZhaoLiue    Professor   178-1920-3912
Thu Feb 11 00:00:00 CST 2021    ZhaoLiue    Professor   178-1920-3912
Fri Feb 12 00:00:00 CST 2021    ZhaoLiue    Professor   178-1920-3912
Sat Feb 13 00:00:00 CST 2021    ZhaoLiue    Professor   178-1920-3912
Sun Feb 14 00:00:00 CST 2021    ZhaoLiue    Professor   178-1920-3912
Mon Feb 15 00:00:00 CST 2021    ZhaoLiue    Professor   178-1920-3912
Tue Feb 16 00:00:00 CST 2021    ZhaoLiuf    Lecturer    138-1929-3912
Wed Feb 17 00:00:00 CST 2021    ZhaoLiuf    Lecturer    138-1929-3912
```

```
Thu Feb 18 00:00:00 CST 2021    ZhaoLiuf    Lecturer    138-1929-3912
Fri Feb 19 00:00:00 CST 2021    ZhaoLiuf    Lecturer    138-1929-3912
Sat Feb 20 00:00:00 CST 2021    ZhaoLiuf    Lecturer    138-1929-3912
Sun Feb 21 00:00:00 CST 2021    ZhaoLiuf    Lecturer    138-1929-3912
Mon Feb 22 00:00:00 CST 2021    ZhaoLiuf    Lecturer    138-1929-3912
Tue Feb 23 00:00:00 CST 2021    ZhaoLiuf    Lecturer    138-1929-3912
Wed Feb 24 00:00:00 CST 2021    ZhaoLiuf    Lecturer    138-1929-3912
Thu Feb 25 00:00:00 CST 2021    ZhaoLiug    Professor   138-1920-0000
Fri Feb 26 00:00:00 CST 2021    ZhaoLiug    Professor   138-1920-0000
Sat Feb 27 00:00:00 CST 2021    ZhaoLiug    Professor   138-1920-0000
Sun Feb 28 00:00:00 CST 2021    ZhaoLiug    Professor   138-1920-0000
Mon Mar 01 00:00:00 CST 2021    ZhaoLiuh    AssciateProfessor   138-1929-0000
Tue Mar 02 00:00:00 CST 2021    ZhaoLiui    Professor   138-1920-0200
Wed Mar 03 00:00:00 CST 2021    ZhaoLiui    Professor   138-1920-0200
Thu Mar 04 00:00:00 CST 2021    ZhaoLiui    Professor   138-1920-0200
Fri Mar 05 00:00:00 CST 2021    ZhaoLiuj    AssciateProfessor   138-1920-0044
Sat Mar 06 00:00:00 CST 2021    ZhaoLiuk    Professor   188-1920-0000
```

3.8 应对面临的新变化

3.8.1 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：

```
请输入起始时间和截止时间:(eg:2021-03-06,2021-03-06)
2021-03-01,2021-03-10
请输入您的选项:
3
请输入待添加的员工人数:
1
请输入员工姓名:
a
请输入员工职位:
a
请输入员工手机号:
13111111111
请输入您的选项:
6
请为员工分配值班时间
请输入待分配的员工的姓名:
a
请输入待分配的员工的职位:
a
请输入员工值班时间:(eg:2021-03-06,2021-03-06)
2021-03-01,2021-03-05
是否继续分配值班(y/Y):
y
请输入待分配的员工的姓名:
a
请输入待分配的员工的职位:
a
请输入员工值班时间:(eg:2021-03-06,2021-03-06)
2021-03-05,2021-03-10
是否继续分配值班(y/Y):
n
```

```
请输入您的选项:
9
日期                          值班人姓名    职位 手机号
Mon Mar 01 00:00:00 CST 2021    a    a    13111111111
Tue Mar 02 00:00:00 CST 2021    a    a    13111111111
Wed Mar 03 00:00:00 CST 2021    a    a    13111111111
Thu Mar 04 00:00:00 CST 2021    a    a    13111111111
Fri Mar 05 00:00:00 CST 2021    a    a    13111111111
Sat Mar 06 00:00:00 CST 2021    a    a    13111111111
Sun Mar 07 00:00:00 CST 2021    a    a    13111111111
Mon Mar 08 00:00:00 CST 2021    a    a    13111111111
Tue Mar 09 00:00:00 CST 2021    a    a    13111111111
请输入您的选项:
```

3.8.2 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：

```
请输入您的选项:
4
请输入课程ID:
1
请输入课程名称:
软件构造
请输入教师姓名:
陈恩鹏
请输入授课地点:
正心11
请输入安排时间的开始:(yyyy-MM-dd hh:mm:ss)
2021-01-02 10:00:00
请输入安排时间的开始:(yyyy-MM-dd hh:mm:ss)
2021-01-02 11:45:00
请输入您的选项:
4
请输入课程ID:
1
请输入课程名称:
软件构造
请输入教师姓名:
陈恩鹏
请输入授课地点:
正心11
请输入安排时间的开始:(yyyy-MM-dd hh:mm:ss)
2021-01-02 10:00:00
请输入安排时间的开始:(yyyy-MM-dd hh:mm:ss)
2021-01-02 11:45:00
time is overlapping!
```

3.9 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。

```
PS D:\Code\IdeaProjects\Lab3-1190200708> git log
commit 711e2c66c777bb2aab11613f9ccc19432510a853 (HEAD -> master, origin/master)
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 17:17:52 2021 +0800

    last

commit caaa7e149cb781f4da9ca0bd3e3e34ec608101e5
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 16:57:30 2021 +0800

    master before change

commit e1f9afa8c788068da2333c4c8ee63f4025f3f8d5
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 16:55:16 2021 +0800

    before change last

commit 16bd2c421d4c0fb6e5af62d55967f52aa7fd67ff
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 16:29:36 2021 +0800

    before change

commit 2cf59fc2b5f5e239e3357e0018e016da8f620606
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 10:43:50 2021 +0800

    app test
:...skipping...
commit 711e2c66c777bb2aab11613f9ccc19432510a853 (HEAD -> master, origin/master)
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 17:17:52 2021 +0800

    last

commit caaa7e149cb781f4da9ca0bd3e3e34ec608101e5
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 16:57:30 2021 +0800

    master before change

commit e1f9afa8c788068da2333c4c8ee63f4025f3f8d5
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 16:55:16 2021 +0800
```

```
commit e1f9afa8c788068da2333c4c8ee63f4025f3f8d5
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 16:55:16 2021 +0800

    before change last

commit 16bd2c421d4c0fb6e5af62d55967f52aa7fd67ff
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 16:29:36 2021 +0800

    before change

commit 2cf59fc2b5f5e239e3357e0018e016da8f620606
Author: Baileys <xiong257246@outlook.com>
Date:   Sun Jul 4 10:43:50 2021 +0800

    app test

commit 3d6be5be3615d0f7cd443bd457a222a0f044d786
Author: Baileys <xiong257246@outlook.com>
Date:   Fri Jul 2 19:03:49 2021 +0800

    DutyRoster

commit 46315e9f5216d99dd9f6e86048e8bc684d710b48
Author: Baileys <xiong257246@outlook.com>
Date:   Thu Jul 1 20:59:43 2021 +0800

    intervalset

commit a3a50ffe9b9f23a45203ed9af2a825a32061d0c5
Author: Baileys <xiong257246@outlook.com>
Date:   Thu Jul 1 15:53:52 2021 +0800

    intervalset
~
~
~
~
PS D:\Code\IdeaProjects\Lab3-1190200708>
```

4 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

5 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

6 实验过程中收获的经验、教训、感想

6.1 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。

6.2 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。

(2) 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。

(3) 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。

(4) 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。

(5) 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。

(6) “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。

(7) 关于本实验的工作量、难度、deadline。

工作量过大、难度过大、deadline 过紧。

(8) 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。