
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 | 管健男 |
| 学号 | 1190200703 |
| 班号 | 1903008 |
| 电子邮件 | guanjiannan@outlook.com |
| 手机号码 | 15636001206 |

# 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

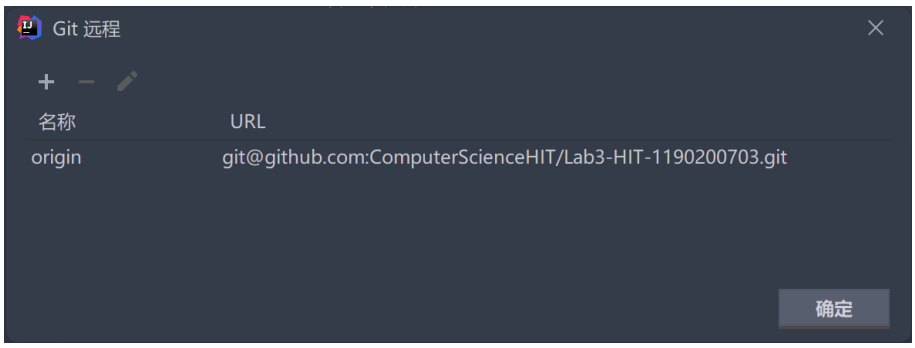
本次实验给定了三个具体应用（值班表管理、操作系统进程调度管理、大学课表管理），学生不是直接针对每个应用分别编程实现，而是通过 ADT 和泛型等抽象技术，开发一套可复用的 ADT 及其实现，充分考虑这些应用之间的相似性和差异性，使 ADT 有更大程度的复用（可复用性）和更容易面向各种变化（可维护性）。

# 2 实验环境配置

在 https://classroom.github.com/a/ZAJ8w2eC 获取此次实验的仓库

在这里给出你的 GitHub Lab3 仓库的 URL 地址（Lab3-学号）。
https://github.com/ComputerScienceHIT/Lab3-HIT-1190200703

设置本地 git 仓库的远程仓库为上述仓库即可：



# 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

### 3.1.1 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---3 月 6 日期间），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。

### 3.1.2 操作系统进程调度管理（ProcessSchedule）

考虑计算机上有一个单核 CPU，多个进程被操作系统创建出来，它们被调度在 CPU 上执行，由操作系统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。

### 3.1.3 大学课表管理（CourseSchedule）

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；

### 3.1.4 三个应用分析

三个应用都是针对“时间段”的应用，可以定义一个时间段 ADT，即下文的 IntervalSet<L>。

对三个应用来说，其 L 分别应为“员工”（Employee）、“进程”（Process）、“课程”（Course），所需关注的属性分别为：

- Employee：姓名、职务、手机号码
- Process：进程 ID、进程名称、最短执行时间、最长执行时间
- Course：课程 ID、课程名称、教师名字、地点、周学时数

三个应用的共性和差异如下表所示

| 应用 | 时间轴有空白 | 不同时间段重叠 | 周期性时间段 |
|---|---|---|---|
| 值班表 | 不可以 | 不可以 | 不可以 |
| 进程调度 | 可以 | 不可以 | 不可以 |
| 大学课表 | 可以 | 可以 | 可以 |

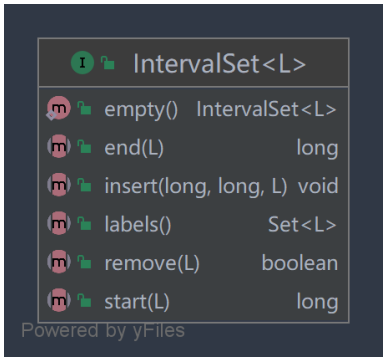
## 3.2 面向可复用性和可维护性的设计：IntervalSet<L>

该节是本实验的核心部分。

### 3.2.1 IntervalSet<L>的共性操作

IntervalSet 是一个 mutable 的 ADT，其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，标签类型为 L，是 immutable 的。

IntervalSet<L>的所有方法如下：




这个接口要求的各个方法说明如下：

static <L> IntervalSet<L> empty()

```
创建一个空对象
返回值: 返回创建的空对象
public static <L> IntervalSet<L> empty() {
    return new CommonIntervalSet<>();
}
```

这个 empty 方法类似于 lab-2 中，graph 接口的 empty 方法，使用 static 修饰，可以直接调用，并返回一个实现了本接口的类的实例。

Set<L> labels();

```
返回值: 获得当前对象中的标签集合
public Set<L> labels();
```

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

```
插入新时间段
参数: start – 开始时间
      end – 停止时间
      label – 时间段标签
抛出: IntervalConflictException – 已存在此标签，或时间段已被占用
public void insert(long start, long end, L label)
        throws IntervalConflictException;
```

Insert 方法再本次实验中时重点，在不同的约束下需要进行不同的调整。

boolean remove(L label);

```
从当前对象中移除某个标签所关联的时间段
参数:    label – 移除时间段的标签
返回值: 若集合中没有找到指定的标签则返回 false；成功删除则返回 true
public boolean remove(L label);
```

long start(L label) throws NoSuchElementException;

```
参数:    label – 标签
返回值: 返回某个标签对应的时间段的开始时间
抛出:    NoSuchElementException – 找不到标签
public long start(L label) throws NoSuchElementException;
```

long end(L label) throws NoSuchElementException;

```
参数:   label – 标签
返回值: 返回某个标签对应的时间段的结束时间
抛出:   NoSuchElementException – 找不到标签
public long end(L label) throws NoSuchElementException;
```

根据测试优先的原则，规划完成后进行测试用例的编写。对上述各个方法进行测试：

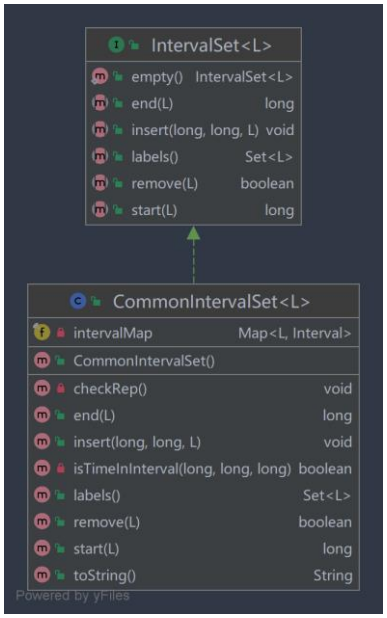
```
// Testing Strategy
// 插入的标签：重复，不重复
@Test
public void labels() {...}

// Testing Strategy
// 插入时间段：重复，不重复
// 插入的标签：重复，不重复
@Test
public void insert() {...}

// Testing Strategy
// 删除时间段：存在，不存在
@Test
public void remove() {...}
```

```
// Testing Strategy
// 查找时间段：存在，不存在
@Test
public void start() {...}

// Testing Strategy
// 查找时间段：存在，不存在
@Test
public void end() {...}
```

由于 IntervalSet 的逻辑比较简单，所以 Testing Strategy 也都比较简单。这一点和 lab-2 中的 graph 不同。各个测试方法的 Testing Strategy 见上图。

### 3.2.2 局部共性特征的设计方案

使用 CommonIntervalSet<L>类实现 IntervalSet<L>接口中的共性方法，CommonIntervalSet<L>的关系图如下：




CommonIntervalSet<L>的 rep、AF、RI、Safety from Rep Exposure 说明如下：

Abstraction function:

AF() = 一组在时间轴上分布的时间段，每个时间段附着一个不重复的标签

Representation invariant:

所有时间段的开始时间是非负整数，每个标签不重复且非空

Safety from rep exposure:

字段使用 private 和 final 修饰，

同时在赋值和返回时使用防御性复制

CommonIntervalSet<L>中定义了字段 intervalMap，如下：

```
标签到时间段的映射
private final Map<L, Interval> intervalMap = new HashMap<>();
```

该字段用于记录标签 L 到时间段的映射，在各个实现的方法中，用这个 Map 来寻找标签对应的时间段。

从这个 Map 也可以看出，IntervalSet 是标签和时间段一一对应的。不可以一个标签对应多个时间段。

CommonIntervalSet<L>中定义了独有的方法 isTimeInInterval

```
判断 time 是否处于 [start, end) 的时间段中
参数:  time – 时间
       start – 时间段起点
       end – 时间段终点
返回值: 在时间段中则返回 true，否则 false
private boolean isTimeInInterval(long time, long start, long end) {
    return time ≥ start && time < end;
}
```

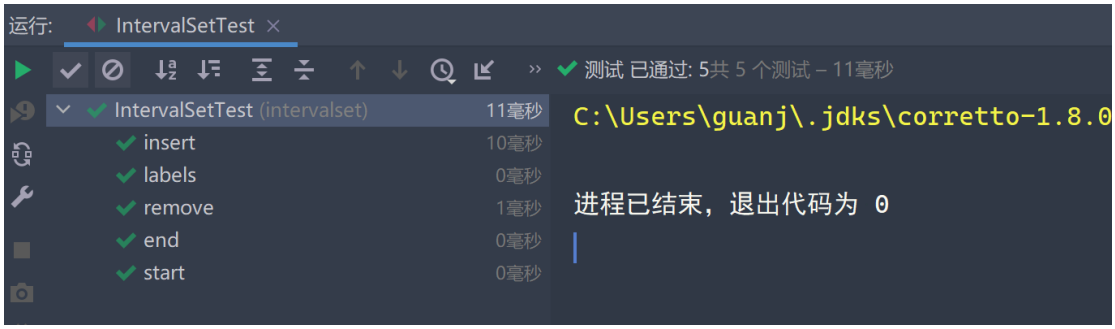
用于判断参数 time 是否处于 [start, end) 的时间段中。这个方法用于在 insert 时判断是否出现时间段重叠的区间：

```
if (isTimeInInterval(start, entry_start, entry_end)
        || isTimeInInterval(entry_start, start, end)) {
    // 出现重叠的区间
    throw new IntervalConflictException();
}
```

上图中判断了待插入的时间段起始时间 start 是否在一个现有时间段内部，或者一个现有时间段的起始时间在待插入时间段的内部，如果有任意条件为真，则说明有区间重叠，此时 insert 方法抛出 IntervalConflictException 异常。

CommonIntervalSet<L>中重写了 toString 方法，将对象内容表示为人容易理解的文本字符串形式。

实现接口之后，进行测试：




可见各个类的测试全部通过。

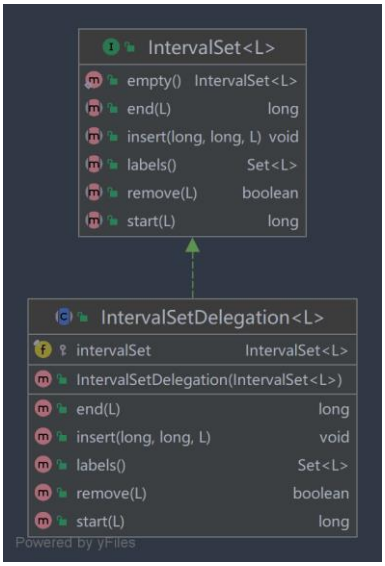
### 3.2.3 面向各应用的 IntervalSet 子类型设计（个性化特征的设计方案）

由于需要使用 IntervalSet 的应用只有值班表 APP，而这个 APP 需要保证无时间段重叠、无时间段空闲、无周期性，因此需要在 CommonIntervalSet 共性特征的基础上增加两个装饰，分别是无时间段重叠 NoOverlapIntervalSet 和无时间

段空闲 NoBlankIntervalSet。

由于无周期性是默认就存在的，除非需要周期性（如大学课表管理系统场景），否则不需要添加功能。

首先实现专门使用委托的类 IntervalSetDelegation，这个类可以看作是对 IntervalSet 的实现：




但是每一个方法都是使用委托，如 insert：

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

这种方式有利于对 IntervalSet 进行特定性质的装饰，如 NoBlankIntervalSet<L>。

NoBlankIntervalSet<L>是在 IntervalSet 共性特征的基础上添加了不允许时间段空闲的装饰。

```
不允许时间段空闲装饰器
public class NoBlankIntervalSet<L> extends IntervalSetDelegation<L> {
```

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

Abstraction function:
AF() = 一组在时间轴上分布的时间段，
每个时间段附着一个不重复的标签，
时间轴上不允许有空闲的时间段
Representation invariant:
时间段的开始时间 > 结束时间

| Safety from rep exposure:<br>字段使用 private 和 final 修饰 |
| --- |

由于不允许时间轴出现空白这个条件需要指明时间轴的开始和结束位置，所以 NoBlankIntervalSet 使用 startTime 和 endTime 进行初始化，这两个参数指明时间轴的起止位置。

```
初始化不允许时间段空闲装饰器
参数: startTime – 总的时间段开始时间
      endTime – 总的时间段结束时间
      intervalSet – 初始时间段
public NoBlankIntervalSet(
        long startTime,
        long endTime,
        IntervalSet<L> intervalSet
) {
```

除了继承父类 IntervalSetDelegation 的所有方法之外，NoBlankIntervalSet 中定义了独有的方法 Set<Interval> blankIntervalSet()，用于查询时间轴上空闲的时间段。这个功能在值班时间表中用于获取还没有被安排的时间段，以及计算时间表的空闲情况。

NoBlankIntervalSet 同样需要测试其独有的方法 blankIntervalSet。

```
// Testing Strategy
// 空闲时间段数量：0 个，1 个，多个
@Test
public void blankIntervalSet() { ... }
```

由于方法是返回空闲时间段的个数，所以测试时 Testing Strategy 选择对空闲时间段的数量：0 个，1 个，多个进行测试。

本测试通过，如下图：


对 IntervalSet 的第二个装饰是不允许时间段重叠，采用 NoOverlapIntervalSet 类进行实现。

```
不允许时间段重叠装饰器
public class NoOverlapIntervalSet<L> extends IntervalSetDelegation<L> {
```

这个类与 NoBlankIntervalSet 的构造方式几乎相同。但是需要注意的是，由于时间段不可重叠的特性，在 insert 的时候需要额外检查插入的时间段。所以 NoOverlapIntervalSet 中重写了 insert 方法，这个是 NoBlankIntervalSet 中没有的。

如下图所示，重写的 insert 方法首先使用父类原有的、不带任何限制的 insert 进行插入，在后期进行时间段的检查：

```
// 先尝试插入
super.insert(start, end, label);
```

当在循环中发现插入的时间段在别的时间段内部时，说明出现时间段重叠，此时应该拒绝插入。所以代码将调用父类 remove 将一开始插入的时间段删除，再抛出 IntervalConflictException 异常。

```
if (start < interval.getEnd() &&
        end > interval.getStart()) {
    // 插入的时间段在别的时间段内部
    super.remove(label);
    throw new IntervalConflictException();
}
```

下面进行对 NoOverlapIntervalSet 重写的 insert 方法进行测试，需要检查时间段重叠时是否抛出异常，以及不重叠时是否可以正常插入，策略如下：

```
// Testing Strategy
// 插入的时间段与现有时间段：不重叠，左侧重叠，右侧重叠，完全包含
@Test
public void insert() {...}
```

当需要检测是否抛出异常时，使用 assert false 在 insert 的下一行进行检测，如果走到了 insert 的下一行，则说明没有抛出异常。否则说明正常抛出了异常：

```
// 左侧重叠
try {
    noOverlapIntervalSet.insert( start: 7,  end: 9,  label: "c");
    assert false;
} catch (IntervalConflictException e) {
    assert true;
}
```

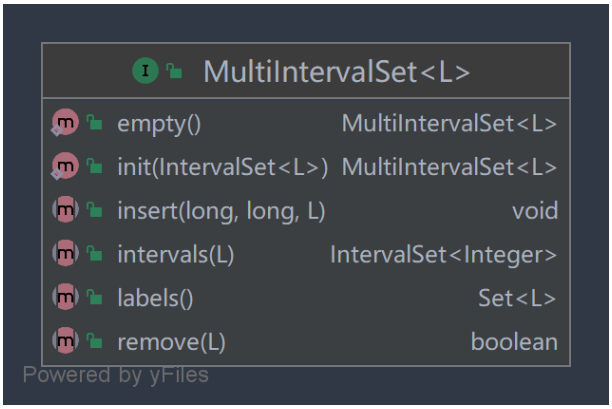
本测试正常通过，如下：


## 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

MultiIntervalSet 是一个 mutable 的 ADT，其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标签，一个标签可以对应多个时间段，标签类型为 L，是 immutable 的。

MultiIntervalSet 的所有方法如下：




本接口要求的各个方法说明如下：

static <L> MultiIntervalSet<L> empty()

```
创建一个空对象
返回值: 返回创建的空对象
public static <L> MultiIntervalSet<L> empty() {
    return new CommonMultiIntervalSet<>();
}
```

empty 方法类似于 IntervalSet 的 empty 方法，都是使用 static 修饰，可以直接调用。

static <L> MultiIntervalSet<L> init(IntervalSet<L> initial)

```
创建一个非空对象
参数:  initial – 数据来源
返回值: 返回创建的非空对象
public static <L> MultiIntervalSet<L> init(IntervalSet<L> initial) {
    return new CommonMultiIntervalSet<>(initial);
}
```

init 方法类似于 empty 方法，但是接收一个 initial 参数，用于根据这个 initial 参数初始化一个 MultiIntervalSet 对象。这个方法通常用于 IntervalSet 转换为 MultiIntervalSet。

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

```
插入新时间段
参数: start – 开始时间
      end – 停止时间
      label – 时间段标签
抛出: IntervalConflictException – 已存在此标签且时间段已被同一标签占用
public void insert(long start, long end, L label)
        throws IntervalConflictException;
```

和 IntervalSet 的 insert 方法类似，MultiIntervalSet 的 insert 方法也需要在不同的约束下进行不同的调整

Set<L> labels();

```
返回值: 获得当前对象中的标签集合
public Set<L> labels();
```

boolean remove(L label);

```
从当前对象中移除某个标签所关联的时间段
参数:  label – 移除时间段的标签
返回值: 若集合中没有找到指定的标签则返回 false；成功删除则返回 true
public boolean remove(L label);
```

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， 返回的时间段按开始时间从小到大的次序排列；
例如：当前对象为 {"A" = [[0, 10), [20, 30)], "B" = [[10, 20)]}， 那么 intervals("A") 返回的结果是{0 =
[0, 10), 1 = [20, 30) }
参数:  label – 搜索的标签
返回值: 返回与 label 所关联的所有时间段
抛出:  NoSuchElementException – 找不到标签
public IntervalSet<Integer> intervals(L label)
        throws NoSuchElementException;
```

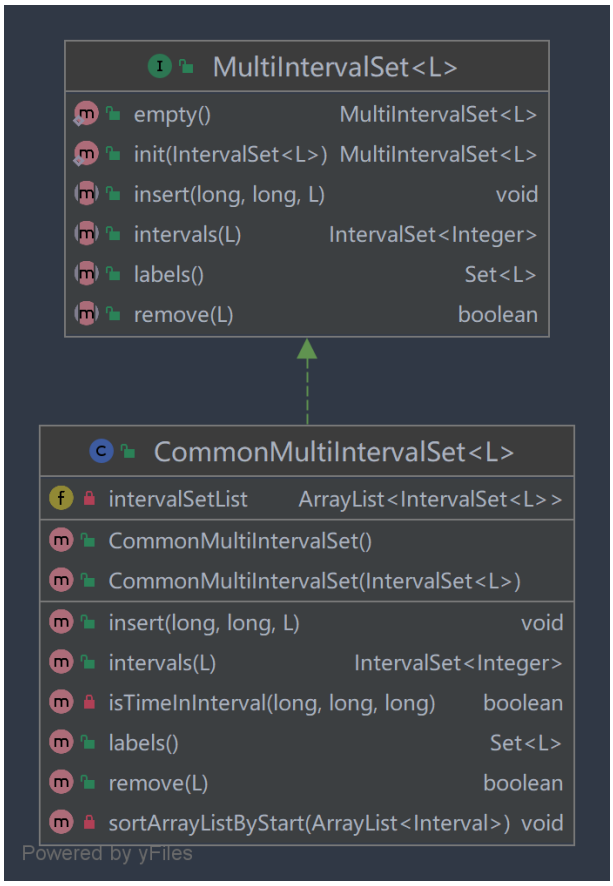
根据测试优先的原则，规划完成后，进行测试用例的编写。上述各个方法单元测试如下：

```java
// Testing Strategy
// 插入时间段：重复，不重复
// 插入的标签：重复，不重复
@Test
public void insert() {...}

// Testing Strategy
// 插入的标签：重复，不重复
@Test
public void labels() {...}

// Testing Strategy
// 删除时间段：存在，不存在
@Test
public void remove() {...}

// Testing Strategy
// 查找的标签：存在，不存在
@Test
public void intervals() {...}
```

其中各个方法的 Testing Strategy 与 IntervalSetTest 中的策略相似。

### 3.3.2 局部共性特征的设计方案

使用 CommonMultiIntervalSet 类来实现 MultiIntervalSet 接口中的共享方法。这个类的关系图如下：




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

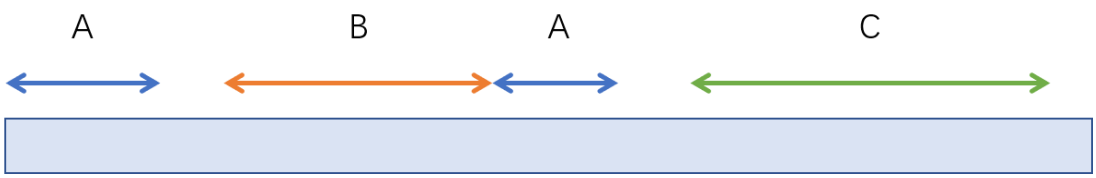
Abstraction function:
AF() = 一组在时间轴上分布的时间段，每个时间段附着一个标签
Representation invariant:
所有时间段的开始时间是非负整数，每个标签且非空
Safety from rep exposure:
字段使用 private 和 final 修饰，
同时在赋值和返回时使用防御性复制

CommonMultiIntervalSet 中定义了字段 intervalSetList：

```
可重叠集合视为若干个不可重叠集合
private final ArrayList<IntervalSet<L>> intervalSetList;
```

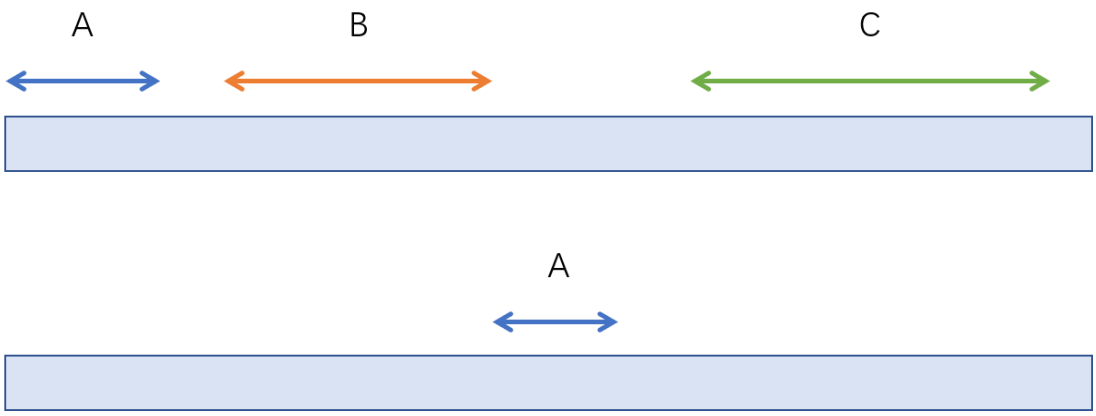
这个字段用于将一个可重复出现标签的时间轴切分成几个不可重复出现标签的时间轴，示意图如下：

MultiIntervalSet 的正常结构示意图：




由上图，标签 A 出现了两次，这在 IntervalSet 中同一个时间轴是不可以出

现这种情况的的，因此，MultiIntervalSet 使用多个时间轴，分别储存同意标签的不同时间段：




由上图，将重复的 A 标签时间段放入下一个时间轴，每个时间轴对应一个 IntervalSet。由于时间轴的数量不确定，所以用 ArrayList 存储各个时间轴。

CommonMultiIntervalSet 中同样定义了自己独有的方法 isTimeInInterval，与 CommonIntervalSet 相同，用于判断一个时间点是否在一个时间段范围内。

```
判断 time 是否处于 [start, end) 的时间段中
参数:   time – 时间
        start – 时间段起点
        end – 时间段终点
返回值: 在时间段中则返回 true，否则 false
private boolean isTimeInInterval(long time, long start, long end) {
```

用 CommonMultiIntervalSet 实现了 MultiIntervalSet 接口后，进行单元测试：


上图可见测试全部通过。

### 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）

在第三个场景大学课表管理系统中，由于时间段可以出现周期，所以需要给 MultiIntervalSet 添加周期功能，即 PeriodicMultiIntervalSet 周期时间段装饰：

```
开启时间段周期性装饰器
public class PeriodicMultiIntervalSet<L>
        extends MultiIntervalSetDelegation<L> {
```

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

Abstraction function:
AF() = 一组在时间轴上分布的时间段，
每个时间段附着一个标签
Representation invariant:
时间段的开始时间 > 结束时间，
时间段周期 > 0
Safety from rep exposure:
字段使用 private 和 final 修饰

PeriodicMultiIntervalSet 在初始化时，接受一个 period 参数，作为时间段的周期：

```
初始化时间段周期装饰器
参数: period – 周期
      multiIntervalSet – 初始时间段
public PeriodicMultiIntervalSet(
        long period,
        MultiIntervalSet<L> multiIntervalSet
) {
```

PeriodicMultiIntervalSet 同样需要重写 insert 方法，用于与大学课表管理系统中的星期几上课进行配合。对 PeriodicMultiIntervalSet 的 insert 测试如下：


可见测试全部通过。

## 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，时间段的标签 L 分别为员工 Employee，进程

Process 和课程 Course，三个标签各自为一个 ADT。

### 3.4.2 员工 Employee

员工 Employee 是一个 immutable 的 ADT，员工的信息包括的名字、职位、手机号码。

这个类的关系图如下：


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

Abstraction function:
AF() = 一名员工，其信息包括的名字、职位、手机号码
Representation invariant:
员工姓名非空；手机号为 11 位数字，满足运营商号段
Safety from rep exposure:
字段使用 private 和 final 修饰

员工的三个信息：名字、职位、手机号码分别使用 private 和 final 修饰：

```
员工名字
private final String name;

员工职位
private final String position;

员工手机号码
private final String phoneNumber;
```

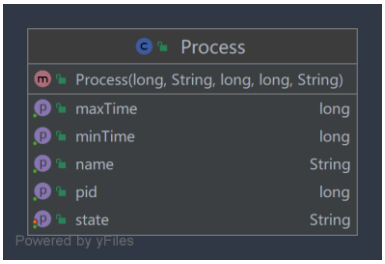
同时为三者提供 getter 进行安全访问：

```
返回值: 获取员工名字
public String getName() { return name; }

返回值: 获取员工职位
public String getPosition() { return position; }

返回值: 获取员工手机号码
public String getPhoneNumber() { return phoneNumber; }
```

### 3.4.3 进程 Process

员工 Process 是一个 immutable 的 ADT，进程的信息包括进程 ID、进程名称、最短执行时间、最长执行时间。

这个类的关系图如下：




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

Abstraction function:
AF() = 一个进程，其信息包括 ID、名称、最短执行时间、最长执行时间
Representation invariant:
ID 为正整数，名称非空，
最短执行时间和最长执行时间为正整数，
最长执行时间 > 最短执行时间
Safety from rep exposure:
字段使用 private 和 final 修饰

进程 ID、进程名称、最短执行时间、最长执行时间 private 和 final 修饰：

```
进程号
private final long pid;

进程名称
private final String name;

最短执行时间
private final long minTime;

最长执行时间
private final long maxTime;
```

同时提供 getter 进行安全访问：

```
返回值: 获取进程号
public long getPid() { return pid; }

返回值: 获取进程名称
public String getName() { return name; }

返回值: 获取进程最短执行时间
public long getMinTime() { return minTime; }

返回值: 获取进程最长执行时间
public long getMaxTime() { return maxTime; }
```

但是由于进程状态在 APP 中可以被修改，所以不使用 final 修饰，同时也提供 setter 进行修改：

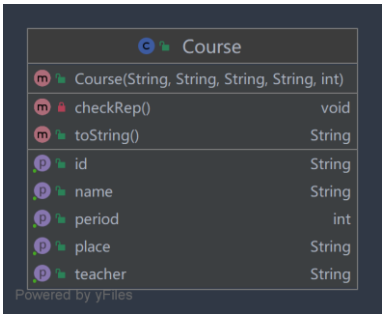
```
进程状态
private String state;
```

```
返回值: 获取进程状态
public String getState() { return state; }

参数: state – 修改进程状态
public void setState(String state) { this.state = state; }
```

### 3.4.4 课程 Course

课程 Course 是一个 immutable 的 ADT，课程的信息包括课程 ID、课程名称、教师名字、上课地点、周学时数

这个类的关系图如下：




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

| |
|---|
| Abstraction function:<br>AF() = 一门课程，信息包括：<br>课程 ID、课程名称、教师名字、上课地点、周学时数<br>Representation invariant:<br>课程 ID 非空，<br>课程名称非空， |

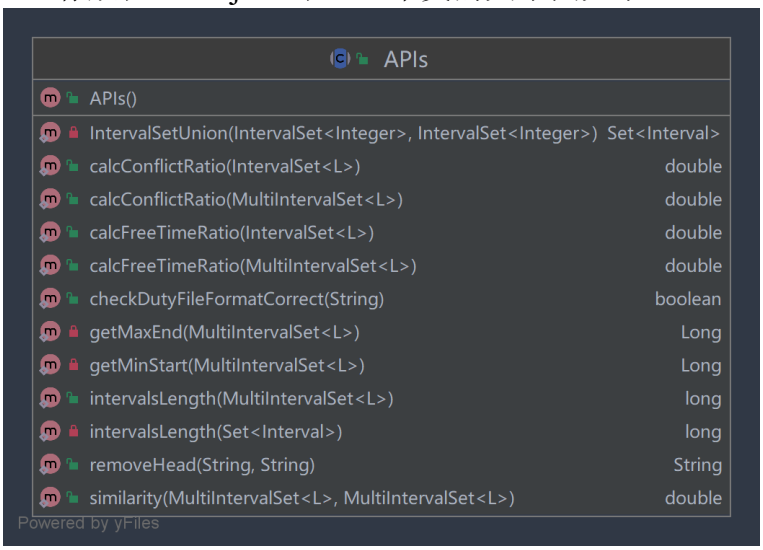
| 教师名字非空，<br>上课地点非空，<br>周学时数为正整数<br>Safety from rep exposure:<br>字段使用 private 和 final 修饰 |
|---|

课程的信息包括课程 ID、课程名称、教师名字、上课地点、周学时数分别使用 private 和 final 修饰，同时提供 getter 进行安全访问。

## 3.5 可复用 API 设计

### 3.5.1 计算相似度

可复用 API 存放在 APIs.java 中。这个类的关系图如下：




计算两个 MultiIntervalSet 的相似度，使用 similarity 方法：

```
计算两个 MultiIntervalSet 对象的相似度
计算方法： 按照时间轴从早到晚的次序，针对同一个时间段内两个对象里的 interval， 若它们标注的
label 等价，则二者相似度为 1，否则为 0； 若同一时间段内只有一个对象有 interval 或二者都没有，则
相似度为 0。 将各 interval 的相似度与 interval 的长度相乘后求和，除以总长度， 即得到二者的整体相
似度。
参数:   set1 – 第一个intervalset
        set2 – 第二个intervalSet
返回值: set1, set2的相似度
public static <L> double similarity(
        MultiIntervalSet<L> set1,
        MultiIntervalSet<L> set2
) {
```

计算原理为：按照时间轴从早到晚的次序，针对同一个时间段内两个对象里的 interval，若它们标注的 label 等价，则二者相似度为 1，否则为 0；若同一时间段内只有一个对象有 interval 或二者都没有，则相似度为 0。将各 interval 的相似度与 interval 的长度相乘后求和，除以总长度，即得到二者的整体相似度。

在获取相似度时，当同一个时间段内两个对象里的 interval 的 label 相同时，需要使用计算两个 Interval 的交集来进行判断。这里定义了 IntervalSetUnion 方法用于求交集。

```
求两个 IntervalSet 的交集
参数:  intervalSet1 – 第一个intervalset
       intervalSet2 – 第二个intervalSet
返回值: 返回 intervalSet1, intervalSet2 的交集
private static Set<Interval> IntervalSetUnion(
        IntervalSet<Integer> intervalSet1,
        IntervalSet<Integer> intervalSet2
) {
```

### 3.5.2 计算时间冲突比例

计算冲突时间比例时，使用 calcConflictRatio 方法，并针对 IntervalSet 和 MultiIntervalSet 两种时间段 ADT 进行重载。

```
求一个时间段集合中，重叠时间段的比例
参数:  set – 时间段集合
返回值: 返回重叠时间段的比例
public static <L> double calcConflictRatio(IntervalSet<L> set) { ... }

求一个时间段集合中，重叠时间段的比例
参数:  set – 时间段集合
返回值: 返回重叠时间段的比例
public static <L> double calcConflictRatio(MultiIntervalSet<L> set) { ... }
```

为提高代码复用率，可以使用 MultiIntervalSet.init 方法，将一个 IntervalSet 对象转换成一个 MultiIntervalSet 对象，再统一使用计算 MultiIntervalSet 时间冲突率的方法进行计算，示意如下：

```
求一个时间段集合中，重叠时间段的比例
参数:  set – 时间段集合
返回值: 返回重叠时间段的比例
public static <L> double calcConflictRatio(IntervalSet<L> set) {
    return calcConflictRatio(MultiIntervalSet.init(set));
}

求一个时间段集合中，重叠时间段的比例
参数:  set – 时间段集合
返回值: 返回重叠时间段的比例
public static <L> double calcConflictRatio(MultiIntervalSet<L> set) { ... }
```

### 3.5.3 计算空闲时间比例

使用于计算时间冲突比例相似的方法，针对对两种不同的 ADT 进行重载：

```
求一个时间段集合中，空闲时间段的比例
参数:  set – 时间段集合
返回值: 返回空闲时间段的比例
public static <L> double calcFreeTimeRatio(IntervalSet<L> set) {
    return calcFreeTimeRatio(MultiIntervalSet.init(set));
}

求一个时间段集合中，空闲时间段的比例
参数:  set – 时间段集合
返回值: 返回空闲时间段的比例
public static <L> double calcFreeTimeRatio(MultiIntervalSet<L> set) { ... }
```

## 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

排班管理系统要求实现的功能如下：

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。

针对这些功能，设计 ADT，即值班时间表 DutyIntervalSet。

```
值班时间表 DutyIntervalSet 是一个 mutable 的 ADT，
每天只能安排唯一的 1 个员工在单位值班，且不能出现某天无人值班的情况， 即值班时间表需要满足无时间段重
叠、无时间段空闲、无周期性； 值班表内需要记录员工的名字、职位、手机号码
public class DutyIntervalSet<L> extends NoBlankIntervalSet<L> {
```

值班时间表 DutyIntervalSet 是一个 mutable 的 ADT，每天只能安排唯一的 1 个员工在单位值班，且不能出现某天无人值班的情况，即值班时间表需要满足无时间段重叠、无时间段空闲、无周期性；值班表内需要记录员工的名字、职位、手机号码。3.6.2

由于 Java 中一个类不能继承自多个类，所以 DutyIntervalSet 只继承自 NoBlankIntervalSet，但在初始化时，接收 NoOverlapIntervalSet 的时间段 ADT，从而同时实现不可空白和不可时间段重叠：

```
super(
        start,
        end,
        new NoOverlapIntervalSet<>(IntervalSet.empty())
);
```

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

Abstraction function:

AF() = 值班时间表：每天只能安排唯一的 1 个员工在单位值班，不能出现某天无人值班的情况，没有周期性

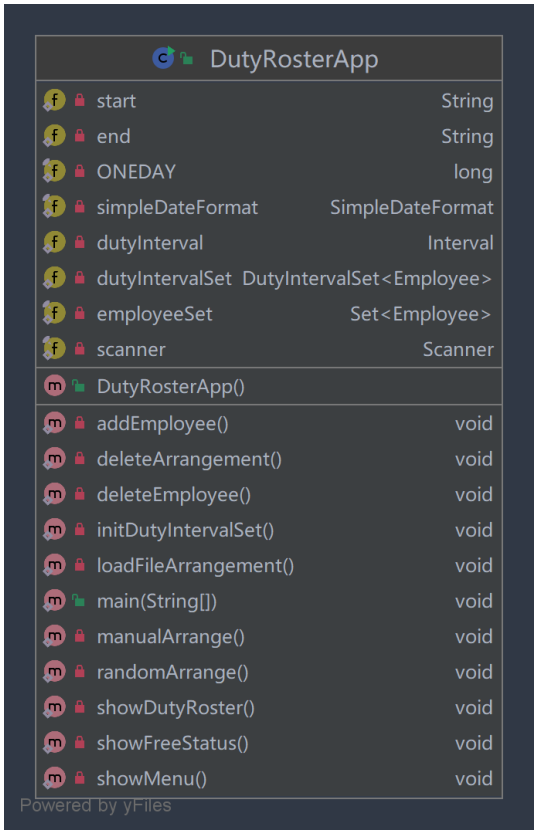
Representation invariant:

无时间段重叠、无时间段空闲、无周期性

Safety from rep exposure:

字段使用 private 和 final 修饰

根据 ADT，实现的排班 APP 关系图如下：



在程序执行的一开始，打印出本程序的功能菜单，如下：

```
输出用户选项菜单
private static void showMenu() {
    System.out.println("1. 设置值班表信息");
    System.out.println("2. 添加员工信息");
    System.out.println("3. 删除员工信息");
    System.out.println("4. 随机生成值班表");
    System.out.println("5. 手动安排值班表");
    System.out.println("6. 查看排班表是否已满或空闲比例");
    System.out.println("7. 查看排班表");
    System.out.println("8. 从文件导入员工信息");
    System.out.println("9. 删除某一员工值班信息");
    System.out.println("10. 退出");
}
```

接下来循环读取用户的输入，根据输入跳转到对应功能的函数中。Switch 分支设计如下：

```
while (loop) {
    System.out.print("请输入操作: ");
    choice = scanner.nextLine();
    switch (choice) {
        case "1":
            initDutyIntervalSet();
            break;
        case "2":
            addEmployee();
            break;
        case "3":
            deleteEmployee();
            break;
        case "4":
```

在每个要求用户输入的结果里，都检查用户输入格式的合法性，如 addEmployee 添加新员工时，若格式有误，则捕获异常优雅推出，并提示用户重新输入：

```
System.out.println("输入新员工信息（姓名,职务,电话号码）：");
try {
    String[] input = scanner.nextLine().split( regex: ",");
    myEmployee = new Employee(input[0], input[1], input[2]);
    employeeSet.add(myEmployee);
    System.out.println("添加成功");
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("输入有误");
}
```

排版系统运行结果如下：

首先初始化排班系统：

```
1. 设置值班表信息
2. 添加员工信息
3. 删除员工信息
4. 随机生成值班表
5. 手动安排值班表
6. 查看排班表是否已满或空闲比例
7. 查看排班表
8. 从文件导入员工信息
9. 删除某一员工值班信息
10. 退出
最开始请先设置值班表相关信息（输入1）
请输入你想进行的相关操作：
1
请输入值班阶段的开始时间：
2021-1-1
请输入值班阶段的结束时间：
2021-5-1
时间设置完成！
```

添加员工：

```
请输入待加入的员工信息（姓名,职务,电话号码）：
管健男,学生,1111
添加员工信息成功！
```

手动安排值班：

```
请输入你想进行的相关操作：
5
请输入选择员工的姓名：
管健男
找到了被选中的员工

请输入员工值班的起始时间和结束时间：
2021-1-3,2021-1-9
值班任务添加完成，请用功能6查看值班表是否存在空闲；
```

查看值班表空闲情况：

```
请输入你想进行的相关操作：
6
值班表有空闲，空闲时间段是：

[2021-01-09,2021-05-01]

[2021-01-01,2021-01-03]

空闲时间占比是：0.05
```

查看值班表：

```
请输入你想进行的相关操作：
7
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话

2021-01-03  2021-01-09  管健男    学生 1111
```

### 3.6.3 操作系统的进程调度管理系统

操作系统进程调度管理系统要求实现的功能如下：

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。

先针对上述功能设计 ADT，即 ProcessIntervalSet。

```
进程调度器 ProcessIntervalSet 是一个 mutable 的 ADT
每个时间只能有 1 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； 即进程调度需要满足无时间段重叠、可以出现时间段空
闲、无周期性
public class ProcessIntervalSet<L>
        extends CommonMultiIntervalSet<L> {
```

进程调度器 ProcessIntervalSet 是一个 mutable 的 ADT。每个时间只能有 1 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；即进程调度需要满足无时间段重叠、可以出现时间段空闲、无周期性。

由于进程调度系统允许 CPU “闲置”，即允许时间轴上出现空白，所以只需要有时间段无重叠的限制即可。在初始化时使用 NoOverlapIntervalSet 的时间段 ADT 作为参数即可实现时间段无重叠：

```
根据 initial 信息初始化进程调度器
参数: initial – 初始信息
public ProcessIntervalSet(IntervalSet<L> initial) {
    super(new NoOverlapIntervalSet<>(initial));
}
```

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

Abstraction function:

AF(nois, nbis, npis) =

进程调度器：每个时间只能有 1 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

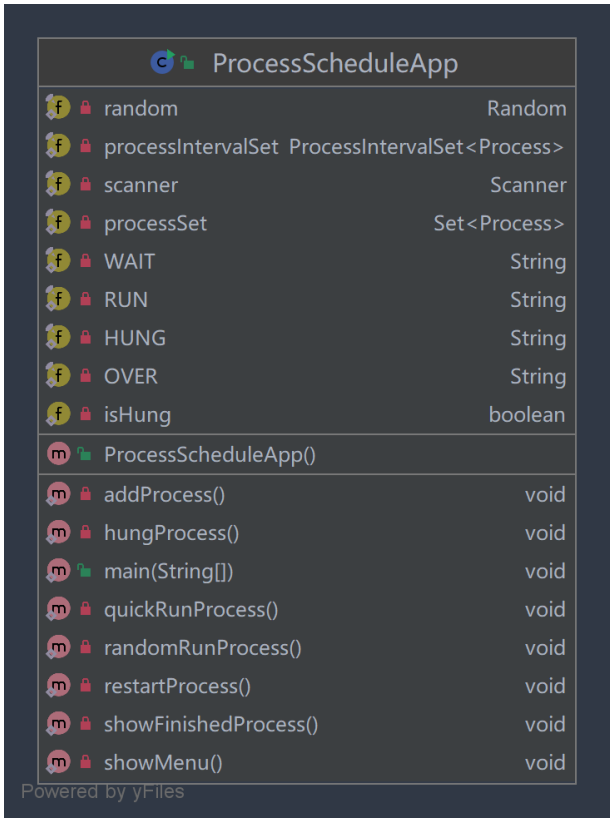
Representation invariant:

无时间段重叠、无周期性

Safety from rep exposure:

字段使用 private 和 final 修饰

根据上述 ADT，实现的进程调度 APP 关系图如下：




和值班时间表相似，程序的一开始会打印出用户选项功能菜单：

```
输出用户选项菜单
private static void showMenu() {
    System.out.println("1. 增加一个进程");
    System.out.println("2. 随机调度进程");
    System.out.println("3. 继续调度进程");
    System.out.println("4. 不执行进程");
    System.out.println("5. 显示进程调度结果");
    System.out.println("6. 执行最短时间进程调度");
    System.out.println("7. 退出");
}
```

接下来根据用户的输入，借助 switch 跳转到指定的方法即可。

进程调度运行如下：

初始化进程调度系统，添加进程：

```
最开始请先设置一组进程（输入1）
1．增加一个进程
2．随机调度进程
3．继续调度进程
4．不执行进程
5．显示进程调度结果
6．执行最短时间进程调度
7．退出
请输入您要进行的操作：
1
输入新进程名：proc
输入新进程 ID：1
输入新进程最短执行时间：10
输入新进程最长执行时间：20
添加成功
```

执行最短时间进程调度：

```
请输入您要进行的操作：
6
最短时间进程调度
当前执行的进程是：
进程 ID   进程名

1    proc
```

### 3.6.4 课表管理系统

课表管理系统要求实现的功能如下：

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。

设计 ADT，即大学课表管理 CourseIntervalSet，说明如下：

大学课表管理 CourseIntervalSet 是一个 mutable 的 ADT，针对某个班级，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，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；

由于第三个场景时间段可以有周期，所以需要继承自 PeriodicMultiIntervalSet，实现与周期有关的功能。在初始化时也要接受与周期有关的参数 period。由于没有时间段空闲和时间段重叠的限制，所以初始化时只需要用原始的 MultiIntervalSet 即可，这一点与前两个 APP 不同：

```
public CourseIntervalSet(long period, IntervalSet<L> initial) {
    super(period, MultiIntervalSet.init(initial));
}
```

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

Abstraction function:

AF() = 大学课表管理系统

Representation invariant:

每个课程的 start 时间 > 一个学期的开始日期

每个课程的 end 时间 < 一个学期的总周数

每个课程的周学时数 <= 学期总周数

Safety from rep exposure:

字段使用 private 和 final 修饰

进入 APP 后，打印出程序的功能菜单：

```java
输出用户选项菜单
private static void showMenu() {
    System.out.println("1. 设置课程表信息");
    System.out.println("2. 添加课程");
    System.out.println("3. 删除课程");
    System.out.println("4. 手动生成课程表");
    System.out.println("5. 查看未安排的课程");
    System.out.println("6. 查看课程表");
    System.out.println("7. 查看每周空闲时间占比");
    System.out.println("8. 查看每周课程重复时间比例");
    System.out.println("9. 查看特定日期的课表");
    System.out.println("10. 退出");
}
```

根据用户的输入，借助 switch 进行跳转。

APP 的运行如下：

初始化学期信息：

```
1. 设置课程表信息
2. 添加课程
3. 删除课程
4. 手动生成课程表
5. 查看未安排的课程
6. 查看课程表
7. 查看每周空闲时间占比
8. 查看每周课程重复时间比例
9. 查看特定日期的课表
10. 退出
最开始请先设置课程表相关的信息（输入1）
输入操作：1
学期开始时间（yyyy-MM-dd）：2021-9-1
学期周数：18
添加成功
```

添加两门课程：

```
输入操作：2
输入新课程信息（课程Id,课程名,教师姓名,上课地点）：0001,软件构造,chp,zx
输入新课程周学时数：
12
添加成功
```

```
输入操作：2
输入新课程信息（课程Id,课程名,教师姓名,上课地点）：0002,计算机系统,wr,zz
输入新课程周学时数：
15
周学时数必须为偶数，请重新输入
16
添加成功
```

查看未安排的课程：

```
输入操作：5
未安排的课程：
课程 ID  课程名    教师姓名  上课教室  周学时数

0001    软件构造  chp zx  12
0002    计算机系统    wr  zz  16
```

安排课程：

```
输入操作：4
输入你课程 ID：
0001
上课时间是星期几（1-7）:2
上课的节数(1, 3, 5, 7, 9)：1
添加成功
```

## 3.7 基于语法的数据读入

先根据实验中给的 test1.txt~test8.txt 找到正确的格式，使用正则表达式进行匹配，判断读取的文本是否是正确的。

可以看到，第一种格式是先给出员工，再给出起止时间：

```
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}
```

第二种格式是先给出起止时间，再给出员工：

```
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}
}
```

所以这里使用两种正则表达式进行匹配，分别匹配上述两种情况：

```
// "Employee" 在 "Period" 之前
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"
);

// "Employee" 在 "Period" 之后
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"
);
```

从而可以判断读取的文件内容是否满足要求。

在确定文件内容正确后，使用 loadFileArrangement 方法进一步读取文件内容，将 json 内容转换为值班表信息：

```
从文件中加载值班安排
private static void loadFileArrangement() {
```

改进后，程序的运行结果如下：

```
请输入操作：8
选择 test1.txt ~ test8.txt:
test1.txt
文件中格式符合要求
添加成功
```

```
请输入操作：7
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-01-23  2021-01-30  ZhaoLiub    Lecturer    138-1921-3912
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-01-10  2021-01-12  ZhangSan    Manger  139-0451-0000
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-01-30  2021-02-01  ZhaoLiuc    Professor   138-1922-3912
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-01-21  2021-01-22  WangWu  AssociateDean   177-2021-0301
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-03-02  2021-03-05  ZhaoLiui    Professor   138-1920-0200
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-02-16  2021-02-25  ZhaoLiuf    Lecturer    138-1929-3912
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-01-22  2021-01-23  ZhaoLiua    Professor   138-1920-3912
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-02-25  2021-03-01  ZhaoLiug    Professor   138-1920-0000
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-03-01  2021-03-02  ZhaoLiuh    AssciateProfessor   138-1929-0000
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-02-09  2021-02-16  ZhaoLiue    Professor   178-1920-3912
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-03-05  2021-03-06  ZhaoLiuj    AssciateProfessor   138-1920-0044
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-03-06  2021-03-07  ZhaoLiuk    Professor   188-1920-0000
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-01-12  2021-01-21  LiSi    Secretary   151-0101-0000
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话
2021-02-01  2021-02-09  ZhaoLiud    Lecturer    198-1920-3912
```

## 3.8 应对面临的新变化

### 3.8.1 变化 1

第一个变化要求排班应用：可以出现一个员工被安排多段值班的情况，例如张三的值班日期为(2021-01-01, 2021-01-10), (2021-02-01, 2021-02-06)；

分析可知，只需要在原来 NoBlankIntervalSet 的基础上，添加一个 NoBlankMultiIntervalSet 的 ADT，实现 MultiIntervalSet 的时间段无空白即可。

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

| Abstraction function:<br>AF() = 一组在时间轴上分布的时间段，<br>每个时间段附着一个的标签，<br>时间轴上不允许有空闲的时间段 |
|---|

| Representation invariant:<br>时间段的开始时间 > 结束时间<br>Safety from rep exposure:<br>字段使用 private 和 final 修饰 |
|---|

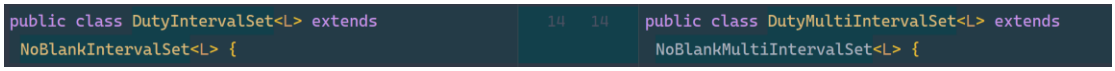
然后，修改值班时间表 ADT 的

class DutyIntervalSet<L> extends NoBlankIntervalSet<L>

改为

class DutyMultiIntervalSet<L> extends NoBlankMultiIntervalSet<L>

即可，git diff 可视化如下图：




修改后，程序正常运行，如下（这里以从文件导入为例）：

```
1. 设置值班表信息
2. 添加员工信息
3. 删除员工信息
4. 随机生成值班表
5. 手动安排值班表
6. 查看排班表是否已满或空闲比例
7. 查看排班表
8. 从文件导入员工信息
9. 删除某一员工值班信息
10. 退出
请先设置值班表信息（1）
请输入操作：8
选择 test1.txt ~ test8.txt：
test3.txt
文件中格式符合要求
添加成功
请输入操作：7
值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话

2021-01-23  2021-01-30  ZhaoLiub    Lecturer    138-1921-3912

值班开始日期  值班结束日期  值班人员姓名  值班人员职位  值班人员电话

2021-03-02  2021-03-05  ZhaoLiui    Professor   138-1920-0200
```

### 3.8.2 变化 2

第二个变化要求课表应用不管学生选课状况如何，不能够出现两门课排在同一时间的情况（即“无重叠”）

从而可知，只需删除课表 ADT 中的重叠性即可，在原来的

class CourseIntervalSet<L> extends PeriodicMultiIntervalSet<L>

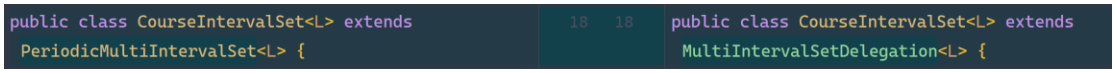
的基础上，删除 PeriodicMultiIntervalSet

注意由于 PeriodicMultiIntervalSet 继承自 MultiIntervalSetDelegation，所以删除之后要添加上 MultiIntervalSetDelegation 作为父类。

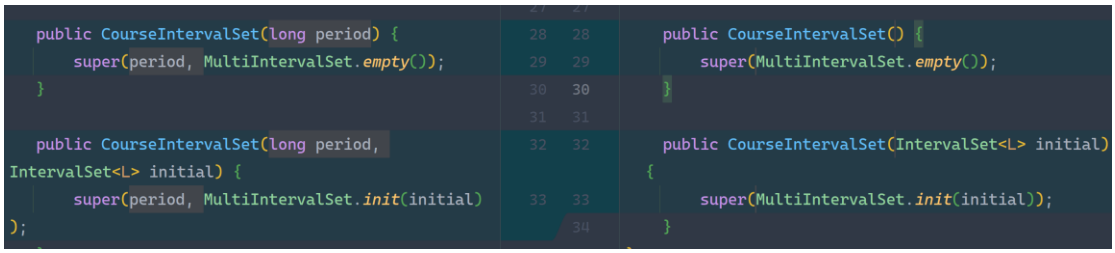
综上，修改后的类声明为：

public class CourseIntervalSet<L> extends MultiIntervalSetDelegation<L>

git diff 可视化如下图：




删除周期参数的 git diff 可视化如下图：




## 3.9 Git 仓库结构

`git log` 记录如下：


可见 master 分支比 change 分支落后两次 commit。change 分支在 master 基础上修改了 DutyRoster 和 CourseSchedule。

# 4 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 |
|---|---|---|---|
| 2021-6-29 | 9：00-10：00 | 理解实验意图，建立仓库 | 完成 |
| 2021-6-30 | 9：00-23：00 | 完成两个需要的 ADT | 完成 |
| 2021-7-1 | 15：00-24：00 | 完善 ADT | 完成 |
| 2021-7-2 | 7：00-24：00 | 完成三个应用 | 基本完成，有小 bug |
| 2021-7-3 | 7：00-10：00 | 完成实验 | 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、装饰器的认识和具体应用方法不清晰 | 多写代码，摸索规律 |
| 文件和类比较多，编写代码的时候有时会找不到需要的类 | 花时间梳理代码逻辑 |

# 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

## 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？

面向 ADT 编程可以提高代码的复用性，降低代码的维护难度，有利于程序功能上的调整。

而直接面向应用场景编程思路更加直接，针对性更强，但是可维护性和可扩展性不如面向 ADT 编程。

(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？

编写 rep、AF、RI、Safety from Rep Exposure 说明等有利于在多个 ADT 开发中迅速找到当前 ADT 的功能，加快开发效率。我愿意基于编写这类说明。

(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？

难处：需要时刻考虑哪些字段、方法等需要暴露给用户；以及怎样设计一个接口，使得后端代码调整后，也不影响前端使用。

乐趣：更有挑战性

(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？

有利于程序的扩展，但是时刻要注意语法的正确性和解析语法的方法

(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？

最大的难点是怎样设计一个接口，使得后端代码调整后，也不影响前端使用。

解决方式是不断重构代码，推翻重来。先提出接口，编写测试，在实际实现功能。

(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，再写测试，最后进行具体实现，不断用测试完善自己的实现。

实现的过程中，可以再通过接口-抽象类-具体类的方式对一个基础的、通用的功能类进行加强。从而提高复用性。

(7) 关于本实验的工作量、难度、deadline。

工作量和难度偏上，deadline 比较紧

(8) 到目前为止你对《软件构造》课程的评价。

很有帮助。设计 ADT 时，可以进行测试优先的编程，从而保证对用户提供良好的功能接口。