哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2024 年春季学期

计算学部《软件构造》课程

Lab 3 实验报告

姓名	全俊康
学号	2022111553
班号	2237102
电子邮件	595630751@qq.com
手机号码	13841319239

目录

1 实验目标概述

本次实验的目标是编写一套可复用可维护的软件，主要使用了

① 子类型、泛型、多态、重写、重载

② 继承，委派，组合

③ 语法驱动的编程 正则表达式

④ API 的设计和 API 的复用

本次实验当中给定了三个具体的应用（值班表管理、操作系统进程调度管理、大学课表管理），但是我们并不直接针对这三个应用进行开发，而是先构建一套可以复用的 ADT，并且要考虑到这些 ADT 未来的可维护性可拓展性。需要利用这些 ADT 实现其中的两个应用，以及采用正则表达式读取文本

2 实验环境配置

本次实验并不提供初始的代码，也就是说所有的代码都是自己完成的，所以配制过程也比较简单。采用 IDEA 作为 IDE 环境，并且使用 GIT 作为版本控制工具

Github Lab3 地址：

https://github.com/ComputerScienceHIT/HIT-Lab3-2022111553

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

（1）排班管理系统(DutyRoster)

简要介绍：一个单位有 n 名员工，然后现在需要为他们排班，使 m 天，每天都有且仅有一个人值班并且每个人值班只能值一个连续时间段，也即要求一个人只能在排班表中拥有一个连续的时间段，值班表需要记录下员工的姓名，值班时间段，职位和电话号码，需要注意的是在我的实验中假定所有人都是不重名的，否则需要 ID 来进行区分。

（2）操作系统进程调度管理(ProcessSchedule)

简要介绍：假设我们有一台计算机，同时已知计算机的 CPU 为单核，一次

只能处理一个进程，那当我们已经获取到一组进程的 ID，名称，最短执行时间，最长执行时间。操作需要对进程进行调度，由于操作系统的分时性可以得到，每个进程都可以在执行一段时间后挂起等待继续执行，但同时只能有一个进程被执行。操作系统会随机选取进程进行执行，并且在特定时间 CPU 可以闲置，也即操作系统此刻没有调度任何进程。

（3）大学课表管理（CourseSchedule）

简要介绍：这里需要我们对实际的课表进行简化，对于某个班级，假定课表是以周为单位不断重复的，直到学期结束。每门课在一周内可以出现多次，并且允许课表中出现空白时间，也就是休息时间。同时一个时间段内可以安排多门课程（由不同老师承担），注意同一个老师不能在同一时间上多门课。

（4）具体分析

相同点：三个应用场景之中都蕴含着时间段这一抽象产物，因此我们可以将时间段集合作为他们的父类。同时每个时间段在三个应用场景之中都需要与别的时间段做出区分，那么就需要我们为每个时间段添加一个标签。

不同点：排班管理系统当中的排班表是不允许出现空白的，而操作系统进程调度管理和大学课表管理之中都是允许空白产生的，也即我们是否需要对空白进行检查和限定

排班管理系统之中一个连续时间段只能与一个人相对应，这就需要我们去检查每个时间段的标签是否重复，而另外两个系统之中是允许一个标签下绑定多个时间段的。

排班管理系统之中不允许一个时间段中有两个或两个以上的人同时值班，这需要我们检查某时刻下的排班表是否含有两个或更多的重叠时间段。同理，由于一次只能处理一个进程，因此操作系统进程调度管理也必须要检查时间段是否重叠。但是大学课表管理没有相关的要求，两节课可以在同一个时间上，但是需要注意的是一个老师不能同时上两节课，所以一个标签下的时间段还是不能重叠的。

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<L>共性操作包括：

静态方法：

- 创建一个空对象：empty()

普通方法：

- 在当前对象中插入新的时间段和标签：void insert(long start, long end, L label) throws IOExcption
- 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)
- 返回某个标签对应的时间段的开始时间：long start (L label)

- 返回某个标签对应的时间段的结束时间：long end (L label)

这些操作所有的 Interval<L>类型都要实现，主要是增删改查的操作，复用性很高所以可抽象到接口当中：

```
/**
 * 移除在时间段集合之中该标签对应的时间段
 *
 * @param label 时间段的标签，必须已经存在于时间段集合之中
 * @return 如果移除成功则返回true；如果没有找到标签对应的时间段或者移除失败则返回false
 */
2 implementations  alporis
public boolean remove(L label);
```

图（1） 接口当中 remove 方法的 spec

3.2.2 局部共性特征的设计方案

采用的是方案 5，也即 CRP，通过接口的组合来实现局部特征的复用，具体就是对于每个不同的维度都进行单独实现最后利用 delegation 将它们的特征进行复用，形成可以满足每个应用场景的特殊接口，大大提高了可复用性，由于是 delegation，也变相的提高了可维护性，我们不必直接修改程序，而是可以通过修改或者拓展其委派的 ADT 来实现程序的维护。

首先根据前面的分析和实验手册的提示，我们不难将三种程序的不同抽象到三个维度之上，首先是否允许重叠，也即 NonOverlapIntervalSet<L>接口，其次是否允许有空白时间段，也即 NoBlankIntervalSet<L>接口，最后是否允许有周期性，也即 NonPeriodicIntervalSet<L>接口。本文中将其放在了 inter 文件夹下，专门存储这三个不同维度的接口。

NonOverlapIntervalSet 接口中重写了方法 insert 以确保插入的时候不会发生重叠。NoBlankIntervalSet 接口中包含方法 checkNoBlank，方便我们在数据输入完毕之后检查是否还存在空白时间段。NonPeriodicIntervalSet 接口比较特殊，因为周期性并不包含于 IntervalSet 和 MultiIntervalSet 之中，所以接口中包含的方法 checkNoPeriodic 的参数 period 需要我们构建完子类型之后再传入，而不像其他两个接口一样可以直接根据传入的 IntervalSet 类型或者 MultiIntervalSet 类型进行判断，不需要额外参数。

之后对这三个接口进行实现，他们的实现放在 impl 文件夹下，值得注意的是，对于 MultiIntervalSet<L>而言，这三个维度也是必要的，因此我们需要在实现的时候区分 IntervalSet 和 MultiIntervalSet，并且都进行实现。本文采取的方法是利用 rep 中的 flag 区分不同两者，如果传入进来的是 IntervalSet 则 flag 为 0，对应着一种实现，MultiInterSet 同理。

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）

三种应用场景之中与 IntervalSet 相符的是排班管理系统之中，因此我们为其实现一种子类型，通过前文的叙述，排班管理系统对应的子类型 DutyIntervalSet

在三个维度上的体现是不允许重叠，不允许空白，不允许有周期。所以 DutyIntervalSet 的接口需要组合其他三个接口。

```
1 implementation  ▲ alporis
public interface IDutyIntervalSet<L> extends NonOverlapIntervalSet<L>, NoBlankIntervalSet<L>, NonPeriodicIntervalSet<L>{
    /**
```

图（2） DutyIntervalSet 接口组合的情况

```
public class DutyIntervalSet<L> implements IDutyIntervalSet<L>, IntervalSet<L> {
    private NonOverlapIntervalSetImpl<L> nois;
    private NoBlankIntervalSetImpl<L> nbis;
    private NonPeriodicIntervalSetImpl<L> npis;
    private IntervalSet<L> inter = IntervalSet.empty();
    private String startTime;
    private String endTime;

    // AF
    // AF (NonOverlapIntervalSetImpl, NoBlankIntervalSetImpl, NonPeriodicIntervalSetImpl,inter) = 排班表
    // AF(startTime) = 排班表的开始时间
    // AF(endTime) = 排班表的结束时间

    // RI
    // 与四个字段的RI相同
    // 也即四个字段中自带的rep已经实现了RI的检查
    //startTime和endTime必须符合“yyyy-MM-dd”的格式
    //endTime>=startTime

    //Safety from rep exposure
    // use private to modify field
    // use defensive copy
```

图（3） DutyIntervalSet 的 delegation 和字段情况

这里需要特殊说明的是，我个人认为这里不应该使用 extends CommIntervalSet<L>，原因是 DutyIntervalSet 之中也使用了 IntervalSet，如果使用继承的话，子类中包含父类会在逻辑上有些奇怪。并且其他三个维度需要传入 IntervalSet，如果是继承关系，那么就不方便传入到三个维度之中进行实现了，因为要求传入的是父类 CommIntervalSet。我觉得这里用委派反而更加顺畅，也即 DutyIntervalSet 实际上是将功能全部委派给了 CommIntervalSet 和其他三个维度。构造方法需要将字段 inter 分别传入三个维度，之后直接使用三个维度的实现类和 inter 就能实现所有功能，这样跟继承的效果是一样，只不过需要将所有方法@Override 一遍，但是每个方法之中基本只是写明委派关系。后文中的其他两种系统也使用这种策略

因此 DutyIntervalSet 之中比普通的 IntervalSet 增加了检查重叠，检查空白的方法，同时由于不存在周期性的信息，所以检查有无周期的方法其实实现起来相当简单，直接返回 false 即可。

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<L>的共性操作如下，和实验手册给的参考方法基本一致

- 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所有时间段：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)，返回结果表达为 IntervalSet<Integer>的形式，其中的时间段按开始时间从小到大的次序排列

主要也是增删改查的操作，不过我取消了通过 IntervalSet 生成 MultiIntervalSet 的方法，原因是我觉得这个方法的泛用性不强。此外原本我写了一个 MultiInterval 作为 MultiIntervalSet 的辅助类，但是由于实验要求必须要复用 IntervalSet，因此只好放弃。

```
/**
 * 向当前时间段集合中插入一组新的时间段和对应标签
 *
 * @param start 时间段的开始时间
 * @param end 时间段的结束时间，不能早于开始时间
 * @param label 时间段对应的标签，必须是不可变类型
 * @throws IOException 如果开始时间小于0;结束时间小于等于开始时间;输入的时间段与集合内已存的相同标签的时间段存在重叠
 */
3 implementations  alporis
public void insert(long start,long end,L label) throws IOException;
```

图（4）MultiIntervalSet 接口当中 insert 方法的 spec

```
alporis
public class CommonMultiIntervalSet<L> implements MultiIntervalSet<L> {
    private List<IntervalSet<L>> intervals = new ArrayList<~>();

    // AF
    // AF(intervals) = 允许标签多绑定的时间段集合

    // RI
    // 同一个标签段的时间不能重叠

    // Safety from rep exposure
    // use private to modify the field
    // use defensive copy
```

图（5）MultiIntervalSet 的相关情况（字段，AF，RI 等）

使用了一个 IntervalSet 的数组来实现 MultiIntervalSet，主要思想就是一个 IntervalSet 中放一组无重复标签的时间段，第一组中包含所有标签。如果新增则寻找到一个没有该标签的数组来存入。

3.3.2 局部共性特征的设计方案

这里的设计方案和前文中有关 IntervalSet 在局部共性特征的设计方案相似，主要也是采用 CRP 的方法，通过委派来在三个维度上进行取值，然后组合起来形成对于每一个应用的特殊组合 ADT。这种方法中不需要我们去在每一个子类型中进行实现（那样相当繁琐），而是通过委派来复用我们之前已经写好了的在不同维度上的特征。

三个维度还是前文所提及的 NonOverlapIntervalSet<L> 接口 NoBlankIntervalSet<L>接口和 NonPeriodicIntervalSet<L>接口，通过对三个接口的组合我们就可以实现我们想要的局部共性特征，省去了大量的重复代码。三个接口之中都有不同的检测方法，方便我们之后的委派，值得注意的是 NonPeriodicIntervalSet 接口之中的方法需要参数 period，这其实需要一个特定的子类型才能发挥作用，如果只是不包含周期信息的子类型，即使组合了这个接口也不能发挥功效，只是作为一个提示的作用，明确该子类型没有周期。

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）

与 MultiIntervalSet 相符的两个引用是操作系统进程调度系统和大学课表管理，分别对应两个子类型 ProcessIntervalSet<L>和 CourseIntervalSet<L>。

首先要说明，和前文 IntervalSet 中提到的一样，我个人因为不应该直接继承 CommonMultiIntervalSet<L>，至少是在我的框架不太适宜，而应当去作为子类实现 MultiIntervalSet<L>，同时 rep 中包含一个 CommonMultiIntervalSet<L>，也即将功能委派出去，这样虽然要将方法都重写一遍，但是基本上只需要明确委派关系即可，工作全部交给三个维度对应的实现类和 CommonMultiIntervalSet<L>,也比较方便维护。

接下来看 ProcessIntervalSet<L>，他应该不允许重叠，但是允许有空白时间，同时不允许有重复的周期。那么就需要组合 NonOverlapIntervalSet 和 NonPeriodicIntervalSet。

```
1 implementation  alporis
public interface IProcessIntervalSet<L> extends NonOverlapIntervalSet<L>, NonPeriodicIntervalSet<L> {
}
```

图（6）IProcessIntervalSet 接口的组合情况

```
public class ProcessIntervalSet<L> implements IProcessIntervalSet<L>, MultiIntervalSet<L> {
    private NonOverlapIntervalSetImpl<L> nois;
    private NonPeriodicIntervalSetImpl<L> npis;
    private MultiIntervalSet<L> intervals = MultiIntervalSet.empty();

    // AF
    // AF（NonOverlapIntervalSetImpl, NonPeriodicIntervalSetImpl,inter,intervals） = 操作系统进程调度记录

    // RI
    // 与三个字段的RI相同
    // 也即三个字段中自带的rep已经实现了RI的检查

    //Safety from rep exposure
    // use private to modify field
```

图（7）ProcessIntervalSet 的委派和 AF，RI 等

通过将功能委派出去，我们不需要重复编写太多代码，因为每个方法都已经

实现过了，这里只需要在每一种重写的方法中明确委派关系即可。

最后来看 CourseIntervalSet<L> ，大学课表管理中允许重叠，允许有空白，允许有周期，因此在接口上三个维度都不需要进行组合，只需要继承 MultiIntervalSet 即可。但是在实现的时候要注意添加周期字段，并且允许修改周期。

```
public interface ICourseIntervalSet<L> extends MultiIntervalSet<L> {
```

图（8） ICourseIntervalSet<L>接口的组合/继承情况

```
public class CourseIntervalSet<L> implements ICourseIntervalSet<L>, MultiIntervalSet<L> {
    private MultiIntervalSet<L> intervals = MultiIntervalSet.empty();
    private final Map<L,List<Long>> PeriodMap = new HashMap<>();

    // AF
    // AF(PeriodMap) = 所有时间段的重复周期
    // AF(intervals) = 课表

    // RI
    // PeriodMap中List的的数值要么为-1，要么>0

    // safety from rep exposure
    // use private and final to modify field
```

图（9）CourseIntervalSet<L> 的继承情况和 AF，RI 等

其中使用 PeriodMap 来记录每一个时间段对应的重复周期，并且提供方法去修改。

```
@Override
public boolean setPeriod(L label, int count, long period) {
    if(count<1) return false;
    if(period != -1 &&  period <0){
        return false;
    }
    if(PeriodMap.containsKey(label)){
        if(PeriodMap.get(label).size()>=count){
            PeriodMap.get(label).add( index: count-1,period);
            return true;
        }
    }
    return false;
}
```

图（10）CourseIntervalSet 之中用来修改重复周期的 setPeriod 方法

3.4 面向复用的设计：L

L 表示的泛型符号可以代表不同的数据类型，当我们在实现 ADT 的时候在

后面附上 L，并且在对应的位置将数据类型替换成通配符 L，就可以实现泛型，这大大提高了我们 ADT 的可复用性，例如在课表管理系统我们传入 Course，在操作系统进程调度系统中我们使用的数据类型就是 Process。只需要设计一个 ADT，然后在使用的时候选择采用的数据类型。对于本文中的 ADT，我们只能向构造不可变数据类型，传入可变数据类型会导致未知错误。泛型的好处就在于大大提高了 ADT 的复用性，允许不同的数据结构使用相同的 ADT。

前文中所有 ADT 都使用了泛型，以方便后续构建系统时使用。

3.5 可复用 API 设计

可复用的 API 我全部放入了 calcAPIs 之中方便管理和使用，其中主要有以下三种可供 ADT 调用的三种方法。

```
1 implementation  alporis
public interface IcalcAPIs<L> {
    /** 计算两个MultiIntervalSet集合的相似度 ...*/
    1 implementation  alporis
    public double Similarity(MultiIntervalSet<L> intervals1,MultiIntervalSet<L> intervals2);

    /** 发现一个 IntervalSet<L>对象中的时间冲突比例 ...*/
    1 implementation  alporis
    public double calcConflictRatio(IntervalSet<L> intervals);

    /** 发现 MultiIntervalSet<L>对象中的时间冲突比例 ...*/
    1 implementation  alporis
    public double calcConflictRatio(MultiIntervalSet<L> intervals);

    /** 计算一个IntervalSet集合的空闲时间比例 ...*/
    1 implementation  alporis
    public double calcFreeTimeRatio(IntervalSet<L> intervals,long length);

    /** 计算一个MultiIntervalSet集合的空闲时间比例 ...*/
    1 implementation  alporis
    public double calcFreeTimeRatio(MultiIntervalSet<L> intervals,long length);
}
```

图（11）calcAPIs 的具体情况

3.5.1 计算相似度

计算相似度这一方法主要实现的是在两个 MultiIntervalSet 之中计算相似程度，并最终以[0-1]的 double 类型小数来表示相似比例。具体实现的思想就是将两个集合都进行遍历，并且根据两个集合中取出的两个时间段的相对位置来确定是否对相似程度有贡献，最后除以总长度即可。这里的总长度为两个集合合在一起的最晚结束时间减去最早开始时间的长度。

```
/**
 * 计算两个MultiIntervalSet集合的相似度
 *
 * @param intervals1 第一个MultiIntervalSet集合
 * @param intervals2 第二个MultiIntervalSet集合
 * @return 计算成功的情况下返回两个时间段集合的整体相似度,介于0-1之间
 */
1 implementation  alporis
public double Similarity(MultiIntervalSet<L> intervals1,MultiIntervalSet<L> intervals2);
```

图（12）Similarity 方法的 spec

3.5.2 计算时间冲突比例

计算时间冲突比例要考虑到传入的参数可能是 IntervalSet 也有可能是 MultiInter，所以要针对不同的参数进行不同的实现，但是除了参数类型的不同，结果是相同。

我尝试了很多方法，但是都不能较好的实现计算时间冲突比例，比如我想通过两个时间段之间的位置来确定冲突时间，但是考虑到同一时间有可能有多个冲突的时间段，所以并不能实现。

所以最后只能采用穷举的办法，也即从开始时间依次加 1（注意我们的时间的数据类型是 long，所以不存在小数，可以通过穷举遍历所有的情况），当检测到该时间点上有两个或两个以上的时间段时，就视为冲突时间。

```
/**
 * 发现一个 IntervalSet<L>对象中的时间冲突比例
 *
 * @param intervals IntervalSet集合
 * @return 冲突指同一个时间段内安排了两个不同的 interval 对象。用
 *         发生冲突的时间段总长度除于总长度，得到冲突比例，是一个[0,1]之间的值
 *         计算成功时返回冲突比例。
 */
1 implementation  alporis
public double calcConflictRatio(IntervalSet<L> intervals);
```

图（13） 计算 IntervalSet 冲突时间比例的方法的 Spec，MultiIntervalSet 的同理

3.5.3 计算空闲时间比例

计算空闲时间原本已经实现过一次，得到末尾两个时间段，然后根据这两个时间段的位置关系来添加对应的空闲时间。但是不幸的由于总长度的定义是集合中最晚时间和最早时间的差，但是在后文排班管理系统之中我们需要的是人为规定的总长度，两者发生了冲突，且不太好平衡两个 length 之间的关系。所以直接采取简单粗暴的穷举法了，遍历每一个时间，如果该时间有对应的时间段，那么

就不是空闲时间，最后就可以得到空闲时间的比例了

```
/**
 * 计算一个IntervalSet集合的空闲时间比例
 *
 * @param intervals IntervalSet集合
 * @param length 人为规定的时间段长度(必须>=当前事件段集合的长度)，如果设为<=0的数则视为取当前集合中最早时间和最晚时间的差作为时间段的长度
 * @return 计算成功的情况下返回空闲时间比例；失败情况返回-1，如果集合为空返回0
 */
1 implementation  alporis
public double calcFreeTimeRatio(IntervalSet<L> intervals,long length);
```

图（14） 计算 IntervalSet 对应的空闲时间比例方法的 spec

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

针对排班管理系统，为了方便功能的实现，我又增加了一个 ADT——DayWithLong，它的功能主要是根据输入的字符串（yyyy-MM-dd 格式）来转化为日期，并且包含两个方法，
1.getInterval（）：用于计算两个以字符串形式表示的日期之间的天数
2.addDays（）将一个字符串转化为增加了对应天数的字符串

```
public interface DayWithLong {
    /**
     * 计算两个日期之间的日期差距
     *
     * @param day1 时间段的开始时间 格式为xxxx-xx-xx，年月日
     * @param day2 时间段的结束时间，必须要比day1更晚
     * @return 两个日期之间相差的日数，如果day2比day1更早或者其他情况则返回-1
     * @throws ParseException 如果字符串格式不正确则抛出异常
     */
    1 implementation  alporis
    public long getInterval(String day1,String day2) throws ParseException;

    /**
     * 计算一个日期加上几天以后的日期
     *
     * @param day 当前日期
     * @param length 需要向后延期多少天，必须要大于等于0
     * @return 当前日期过了length天以后的字符串形式的新日期，如果length小于0或其他异常情况则返回空字符串
     * @throws ParseException 如果字符串格式不正确则抛出异常
     */
    1 implementation  alporis
    public String addDays(String day,long length) throws ParseException;
}
```

图（15）DayWithLong 的接口详细情况

之后来实现排班管理系统 DutyRostAPP，它主要是基于之前我们实现了的

DutyIntervalsSet 实现的，但是也加入了很多其他的功能，例如计算空闲时间比例的方法 getFreeTimeRatio，主要是通过调用 calcAPIs 来进行实现的，还有检查当前排班表是否已满的 full 方法，通过调用 getFreeTime 方法来实现，如果调不出空闲的时间，则说明排班表已经装满。除此之外还有很多 showEmployee 等向用户展示的方法。通过对 DutyIntervalSet 的复用，实现起来也较为简单。

总而言之，由于有之前其他 ADT 的基础，实现排班管理系统并不是非常困难，相比于直接进行开发，减少了很多工作量。

```
public class DutyRostAPP implements IDutyRosterAPP {
    private DutyIntervalSet<Employee> duty;
    private final List<Employee> Employees = new ArrayList<>();
    private String startTime;
    private String endTime;
//    private final List<Integer> arrange= new ArrayList<>();

    // AF
    // AF(Employees) = 所有员工信息
    // AF(startTime) = 排班表的开始时间
    // AF(endTime) = 排班表的结束时间
    // AF(duty) = 排班表
    // AF(arrange) = 用于标示每一个日期是否已经安排完毕

    // RI
    // 构造duty之后要满足duty的相关RI，已被duty自身的checkRep满足
    // 员工被删除之后，排班信息也要对应删除
    // 排班信息中的所有员工必须已经在集合之中
    // 员工的姓名不能重复

    // safety from rep exposure
    // 使用private和final修饰字段
    // 必要时采用防御式拷贝
```

图（16）DutyRostAPP 的字段，AF，RI 等信息

最后编写一个 main 方法让用户可以在命令行中使用这一系统，不过做的也许并不够从分，但是客户端的程序员可以根据 DutyRoster 提供的各类方法丰富系统展示给用户的功能。

```
请先输入排班表的开始时间（yyyy-MM-dd）：
2024-05-01
请输入排班表的结束时间（格式同上）：
2024-05-05
菜单退出程序0
增加员工1
删除员工2
安排排班3
查看空闲时间比例和空闲时段4
自动排班5
展示当前排班表6
展示员工列表7

请选择您想执行的指令:1
请输入人员信息，注意不能重名，格式为姓名,职位,电话号码（xxx-xxxx-xxxx）：
zhangsan,jingli,231-2234-2342
lisi,baojie,213-4353-4655
wangwu,dongshizhang,323-3244-5465
stop
请选择您想执行的指令:4
还未填入排班记录，比例暂时设为0
空闲时间比例为：0.0
以下时间还是空闲的哦：
2024-05-01
2024-05-02
2024-05-03
2024-05-04
2024-05-05

请选择您想执行的指令:7
zhangsan  jingli  231-2234-2342
lisi  baojie  213-4353-4655
wangwu  dongshizhang  323-3244-5465

请选择您想执行的指令:5
排班表的起止时间为2024-05-01---2024-05-05
以下是已经安排的时间段
2024-05-01  zhangsan  jingli  231-2234-2342
2024-05-02  lisi  baojie  213-4353-4655
2024-05-03  wangwu  dongshizhang  323-3244-5465
2024-05-04  wangwu  dongshizhang  323-3244-5465
2024-05-05  wangwu  dongshizhang  323-3244-5465

请选择您想执行的指令:2
请输入要删除的员工名称(输入-1为停止删除)：
zhangsan
请输入要删除的员工名称(输入-1为停止删除)：
-1
想要删除的人不存在
请选择您想执行的指令:6
排班表的起止时间为2024-05-01---2024-05-05
以下是已经安排的时间段
2024-05-02  lisi  baojie  213-4353-4655
2024-05-03  wangwu  dongshizhang  323-3244-5465
2024-05-04  wangwu  dongshizhang  323-3244-5465
2024-05-05  wangwu  dongshizhang  323-3244-5465

请选择您想执行的指令:4
空闲时间比例为：0.2
以下时间还是空闲的哦：
2024-05-01

请选择您想执行的指令:
```

图（17）使用排班管理系统的一个例子

3.6.2 操作系统的进程调度管理系统

操作系统的调度管理系统和排班管理系统类似，也是基于 ProcessIntervalSet 实现的 ProcessScheduleAPP

```
public class ProcessScheduleAPP {
    private final ProcessIntervalSet<Process> intervals = new ProcessIntervalSet<~>();
    private final List<Process> allProcess = new ArrayList<>();
    private final Map<Process, Boolean> processMap = new HashMap<>();

    //AF
    //AF(intervals) = 进程的安排表
    //AF(allProcess) = 所有的进程组成的集合
    //AF(processMap) = 进程的状态（是否已经运行完毕）

    //RI
    //进程安排表中的所有进程必须存在于进程集合之中
    //进程的ID不能重复
    //进程安排表初始化后，满足进程安排表的RI

    //rep safety form exposure
    //使用private和final修饰字段
    //必要时使用防御式编程
```

图（18）ProcessScheduleAPP 的字段，AF，RI 等信息

在 ProcessScheduleAPP 之中我们额外实现了 optimize 方法和 random 方法，用来模拟优化进程调度和随机进程调度两种情况。

虽然在前文设计 ProcessIntervalSet 的时候是允许出现空白的，但是根据系统的功能描述，每一次调用进程都需要进行一次执行，为了符合现实逻辑，不应该存在调用一个进程瞬间就结束的情况（本次调用进程的开始时间和结束时间相同），所以实际上的进程调度系统之中是不包含空白的。

```
请输入进程信息（ID，名称，最短执行时间，最长执行时间）：
1,h1,4,5
请输入进程信息（ID，名称，最短执行时间，最长执行时间）：
2,h2,6,8
请输入进程信息（ID，名称，最短执行时间，最长执行时间）：
3,h3,4,9
请输入进程信息（ID，名称，最短执行时间，最长执行时间）：
4,h4,10,13
请输入进程信息（ID，名称，最短执行时间，最长执行时间）：
stop
需要执行的进程信息如下
ID   名称 最短执行时间   最长执行时间
1    h1  4   5
2    h2  6   8
3    h3  4   9
4    h4  10  13

是否需要优化进程安排表(Y/N):
Y
ID   进程名称  开始时间  结束时间
1    h1  0   4
2    h2  8   14
3    h3  4   8
4    h4  14  24
```

图（19）操作系统进程调度系统的一个实例

同样这个系统可能还不够智能和完善。但是客户端程序员可以根据 ProcessIntervalSet 的方法，optimize 方法以及 random 方法丰富更多功能

3.7 基于语法的数据读入

基于语法的读入主要需要使用正则表达式来对文本进行匹配，通过阅读实验手册，我们可以得知文本主要分为三个部分，分别是 Period,Roster 和 Employee，我们首先要先对这三个部分进行分割。

我采用的方法是新增一个 PaeserForDuty 类，这个类只有一个方法 threeKind（),作用就是将文本进行分割，分别匹配并取出其中的内容按照 Period，Employee，Roster 的顺序放在一个 List<String> kindList 中以便于我们取用。对应的正则表达式如下：

```
//根据三个部分的不同次序，一共有六种可能，全部写出来
String kind1 = "^Period\\{(.*)}Roster\\{(.*)}Employee\\{(.*)}";
String kind2 = "^Period\\{(.*)}Employee\\{(.*)}Roster\\{(.*)}";
String kind3 = "^Roster\\{(.*)}Employee\\{(.*)}Period\\{(.*)}";
String kind4 = "^Roster\\{(.*)}Period\\{(.*)}Employee\\{(.*)}";
String kind5 = "^Employee\\{(.*)}Roster\\{(.*)}Period\\{(.*)}";
String kind6 = "^Employee\\{(.*)}Period\\{(.*)}Roster\\{(.*)}";
```

图（20） 用于分割三个部分的正则表达式

之后我们需要在 DutyRosterAPP 中新增一个 readFile 方法，首先调用 PaeserForDuty 类拆分文本，然后对应的将三种文本都按照对应的正则表达式放入到 DutyIntervalSet 之中，完成排班表的设计。

用于读取 Period，Employee，Roster 的内容的正则表达式如下：

Period："(.*)[,](.*)"
Employee:"\\s*(\\w+)\\{([^,]+),(\\d{3})-(\\d{4})-(\\d{4})}"
Roster："\\s*(\\w+)\\{(\\d{4})-(\\d{2})-(\\d{2}),(\\d{4})-(\\d{2})-(\\d{2})\\}"

接下来给出一个实例，这个实例来自于实验手册，当我们把实验手册对应的文本放在文件之中让程序读入，最后的排班表如下所示：

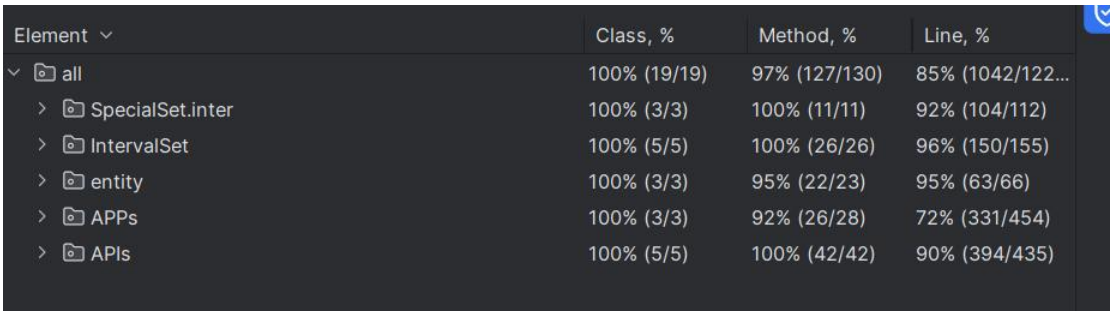
```
排班表的起止时间为2021-01-10---2021-03-06
以下是已经安排的时间段
2021-01-10  ZhangSan  Manger  139-0451-0000
2021-01-11  ZhangSan  Manger  139-0451-0000
2021-01-12  ZhangSan  Manger  139-0451-0000
2021-01-13  ZhangSan  Manger  139-0451-0000
2021-01-14  ZhangSan  Manger  139-0451-0000
2021-01-15  ZhangSan  Manger  139-0451-0000
2021-01-16  ZhangSan  Manger  139-0451-0000
2021-01-17  ZhangSan  Manger  139-0451-0000
2021-01-18  ZhangSan  Manger  139-0451-0000
2021-01-19  ZhangSan  Manger  139-0451-0000
2021-01-20  ZhangSan  Manger  139-0451-0000
2021-01-21  ZhangSan  Manger  139-0451-0000
2021-01-22  ZhangSan  Manger  139-0451-0000
2021-01-23  ZhangSan  Manger  139-0451-0000
2021-01-24  ZhangSan  Manger  139-0451-0000
2021-01-25  ZhangSan  Manger  139-0451-0000
2021-01-26  ZhangSan  Manger  139-0451-0000
2021-01-27  ZhangSan  Manger  139-0451-0000
2021-01-28  ZhangSan  Manger  139-0451-0000
2021-01-29  ZhangSan  Manger  139-0451-0000
2021-01-30  ZhangSan  Manger  139-0451-0000
2021-01-31  ZhangSan  Manger  139-0451-0000
2021-02-01  LiSi  Secretary  151-0101-0000
2021-02-02  LiSi  Secretary  151-0101-0000
2021-02-03  LiSi  Secretary  151-0101-0000
2021-02-04  LiSi  Secretary  151-0101-0000
2021-02-05  LiSi  Secretary  151-0101-0000
2021-02-06  LiSi  Secretary  151-0101-0000
2021-02-07  LiSi  Secretary  151-0101-0000
2021-02-08  LiSi  Secretary  151-0101-0000
2021-02-09  LiSi  Secretary  151-0101-0000
2021-02-10  LiSi  Secretary  151-0101-0000
2021-02-11  LiSi  Secretary  151-0101-0000
2021-02-12  LiSi  Secretary  151-0101-0000
2021-02-13  LiSi  Secretary  151-0101-0000
2021-02-14  LiSi  Secretary  151-0101-0000
2021-02-15  LiSi  Secretary  151-0101-0000
2021-02-16  LiSi  Secretary  151-0101-0000
2021-02-17  LiSi  Secretary  151-0101-0000
2021-02-18  LiSi  Secretary  151-0101-0000
2021-02-19  LiSi  Secretary  151-0101-0000
2021-02-20  LiSi  Secretary  151-0101-0000
2021-02-21  LiSi  Secretary  151-0101-0000
2021-02-22  LiSi  Secretary  151-0101-0000
2021-02-23  LiSi  Secretary  151-0101-0000
2021-02-24  LiSi  Secretary  151-0101-0000
2021-02-25  LiSi  Secretary  151-0101-0000
2021-02-26  LiSi  Secretary  151-0101-0000
2021-02-27  LiSi  Secretary  151-0101-0000
2021-02-28  LiSi  Secretary  151-0101-0000
2021-02-28  LiSi  Secretary  151-0101-0000
2021-03-01  WangWu  Associate Dean  177-2021-0301
2021-03-02  WangWu  Associate Dean  177-2021-0301
2021-03-03  WangWu  Associate Dean  177-2021-0301
2021-03-04  WangWu  Associate Dean  177-2021-0301
2021-03-05  WangWu  Associate Dean  177-2021-0301
2021-03-06  WangWu  Associate Dean  177-2021-0301


Process finished with exit code 0
```

图（21）实验手册文本对应的排班表

值得一提的是实验手册里面给出的实例上有一个小错误，也即结束时间是 03-06，但是给 WangWu 的时间安排截止到了 03-10，但是由于我设计的 DutyRostApp 允许这种时间跨度超过起止时间的排班（会直接忽略掉超过的部分），所以仍能输入正常的情况

最后全部程序的测试覆盖度结果如下：



图（22） 全部代码的测试覆盖度

可以看出，除了 main 方法不能进行测试之外，基本覆盖了全部的方法，代码覆盖行数由于有 main 函数被拉低了一些，但是仍在 85%，也算覆盖的比较全面

2024.05.23 补充
为了安全性和代码的严谨性，将原来代码中的 Date 类型全部替换成了 LocalDate 不可变数据类型，代码可能会和前面的报告有一定的出入，例如前文图中显示抛出的 ParseException 异常应该全部改为 DataTimeException 异常等，由于时间关系，未能完全重新编写实验报告，请见谅。

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
2024.04.22	14:00-17:00	写完 IntervalSet 和 MultiIntervalSet 对应的接口和测试	完成任务
2024.04.22	18:30-22:00	实现两个 ADT	未能准时完成任务
2024.04.27	18:00-22:00	实现两个 ADT 并且写出三个维度的接口和测试	顺利完成
2024.04.28	8:30-12:00	实现三个维度的接口，并且写出测试和实现	顺利完成

2024.04.28	14:00-21:00（有休息）	实现三种子类型	卡了很长一段时间，但最终完成了
2024.04.28	21:30-22:30	完成三个实体类	顺利完成
2024.04.29	11:00-12:00	完成 calcAPI 的接口和测试	未能顺利完成，测试没写完
2024.04.29	16:00-17:45	完成 calcAPIs 的接口和测试，并且尝试实现一部分	顺利完成
2024.04.29	9:30-12:00	实现 calcAPIs	顺利完成
2024.05.01	8:30-11:45	实现排班表系统需要的 DayWithLong 类并且写出排班表系统的接口	顺利完成
2024.05.01	14:00-17:45	写完排班表的测试和部分方法	未能顺利完成，计算空闲比例的方法产生了问题
2024.05.01	18:45-23:00	初步实现排班表系统和正则表达式的读取	完成，但是系统有相当多不完善的地方，正则表达式也有错误
2024.05.02	8:30-11：45	完成排班表系统和正则表达式	顺利完成
2024.05.02	14:00-18:00	完成操作系统进程调用系统	顺利完成！
2024.05.23	17:00-20:30	修改了 Date 类型并且修改了系统一些不太完善的部分	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
MultiIntervalSet 不知道怎么基于 IntervalSet 实现	想到了使用多个 IntervalSet 组成的数组来实现 MultiIntervalSet
三种子类型的委派关系没弄明白，由于子类中包含一个父类，所以 checkRep 会报莫名其妙的错	决定让三种子类型分别去委派 IntervalSet 和 MultiIntervalSet 来完成他们的功能，而不是直接继承下来
计算空闲比例的方法出现问题	由于我们需要认为的设定总长度，所以最后干脆使用穷举法
不知道字符串形式的日期怎么存入到 ADT 之中	增加一个新的类 DayWithLong 专门处理 yyyy-MM-dd 这样格式的字符串，将其以天为单位转化到 long 形式的数据之中，节省了排班表要存储时间段的长度和所占用的空间，也为遍历求得空闲时间比例奠定可能性，因为如果直接用字符串转化成毫秒单位的 long 型数据，遍历将会变得非常慢，而现在一年最多 366 天，哪怕来上 2000 年也不会是巨量的数据。
正则表达式出错，匹配不到	如果是用字符串先写出正则表达式的语法，那么需要用\\来进行转义，同时在网上查询资料学习了正则表达式的相关知识，还有一个特别有用的

	网站 https://regex101.com/可以帮助测试正则表达式的正确性。

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

在本次实验当中体验了一次从头构建 ADT 的过程，在这个过程里面有很多经验教训。总结而言是以下几点

1. 软件构造是一个非常复杂的过程，不要指望快速的构建出来，那样往往会留下很多 bug，像是在构建 DaywithLong 类的时候我就有些操之过急，使用了 Date 数据类型，为后面的错误埋下伏笔，这里属于是没有深思熟虑。
2. 一定要“胸有成竹”，意思是在还没有开始写代码之前就要大概的想明白这个系统的全貌，不然闷头走路只会撞墙....
3. 测试优先非常重要，如果不进行测试的话，正确性将无法保证，那么之后如果调用了这个类，那么等待你的是无穷无尽的 bug，而且你还不知道具体的位置在哪里。
4. 多用委托而不是继承，如果采用继承的话会遇到组合爆炸的问题，而委派既能实现功能的拓展，又能避免掉组合爆炸的问题，本次实验当中三个维度的特性如果使用继承，那么将会有 8 种情况，显著增加了工作量

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？

答：面向 ADT 编程往往需要考虑的更加全面和抽象一些，比如说泛型的使用，又比如一个基础的增删改查类的方法，而直接面向应用场景编程则更加具体，在 lab2 中直接面对应用场景编程，就是先使用的 String 作为标签，后改的泛型。

本文中的 ADT 只需要稍加改造就可以复用在三个不同的应用场景，的确大大减少了开发的重复性，而且这还只是三个应用场景，应用场景越多，节省的代码量就越多。让我有所体会的就是委派的快乐，不需要写太多代码，只需要把委派关系指明，数据传进去就可以使用。这样 ADT 封装好了以后可以留待取用，不必每次见到一个新应用场景就要想一套实现方法。

(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？

答：我个人认为这些工作的意义就是让 ADT 变的规范化，方便下次其他人或者自己去复用，如果一个 ADT 的 spec 等信息都没有写明，那么复用也是相当麻烦，想要看懂那些没有注释的代码也是需要很多精力的，可能最后都不如自己写一份。至于 RI 和 AF 等我认为就是对 ADT 的封装，ADT 要让外界不能干预它，越稳定越好，这样在复用的时候才不会出现莫名其妙的错误。防止泄漏就是典型的防止客户端修改 ADT。经过这些规范，ADT 就能变得稳定且易于复用，换句话，能够让别人放心的使用。

在以后的编程之中，如果时间不是很紧，我也很乐意坚持这些习惯，但是不得不说这样给 ADT 写明 AF 等信息确实比较麻烦，但是就像老师所说，如果没有这些信息和测试用例，那么说明这些代码就是垃圾代码，别人根本不会去用，出现问题了也很难纠错。

(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？

答：难处就是要考虑到很多的情况，例如设计一个 ADT 要不要采用泛型，具体的方法要有哪些，委派和继承关系是怎么样的，具体要用什么 Rep 等等，都是很恼人的问题，尤其是你要进行测试，这也是一个相当繁琐的过程。

乐趣在于构造具体系统的时候通过委派或者继承的时候写方法真的很舒服，只需要指明委派关系即可，省去了大量的力气，有一种畅快感。

(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？

答：语法驱动编程大大降低了输入指令的繁琐，提前在文件当中写好指令和数据，系统就能自动的进行工作，省去了大量的工作，并且语法驱动编程还有一个好处就是只要语法错了，系统就会停止或者报错，这时候只需要去修改错误的部分即可。不像单独指令操作，中间错一步就得重头来过，测试的时候会非常的痛苦。

(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？

答：我认为难度在于设计的时候由于没有经验，很难一次就想好所有要实现的方法。经常出现我想要完成某个功能，但是 ADT 并没有实现，甚至连相关的方法的返回值也不太符合我的预期，因此不得不去修改 ADT 或者在当前的方法之中从头开始写。

克服的办法就是提前构思好系统需要什么功能，通过这些功能反推出需要 ADT 提供什么样的方法，避免这种返工的情况发生。当然很难一次构思出来一个完全没见过的系统，所

以设计 ADT 的时候稍微多设计一些方法，甚至可以采用 visitor 模式。有的时候虽然有些返回值和方法看着没有用，但是可能在日后某些比较奇怪的功能里面就能派上用场。

(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？

答：接口要抽象出普遍的方法并且完成接口之间的组合，抽象类起到一个承上启下的作用。类的继承是相当有用的特性，但是要注意的就是如果是想实现个性化的功能，使用继承很容易陷入到组合爆炸的情况下，所以最好多多使用委派，委派一方面加强了可复用性，另一方面也加强了可维护性，只需要扩展被委派功能的 ADT 即可修改当前 ADT 的功能。

最一开始最好想好自己采取的设计模式，不然可能会陷入混乱，也即理不清继承和委派关系，面对一堆 Override 茫然失措。与其埋头苦哈哈的写代码，不如多花点时间在设计 ADT 和选择设计模式之上，这些时间绝对是值得的，例如我三个子类型的委派关系一开始没这么做，checkRep 总是报错，耽误了相当长的时候，而且还让我一头雾水，完全不知道怎么改。

(7) 关于本实验的工作量、难度、deadline。

答：个人感觉本实验的工作量比前两个实验要大一些，同时难度也有上升，不过好在 deadline 比较充裕，又正好赶上假期，加加班也就干完了...

(8) 到目前为止你对《软件构造》课程的评价。

答：一门很不错的课程，教会了我很多之前从未了解过的概念，我觉得我在向着软件工程师的角色不断靠近。其实相比于做题，我更喜欢动手实操的实验，当然是时间宽裕的情况下，不然就是折磨了 QAQ。总而言之，软件构造中有很多之前的我完全不会考虑的事，以前主要停留在写算法的层面上，但是软件构造课带着我向上迈进了一步。