



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	沈城有
学号	1190200526
班号	
电子邮件	
手机号码	

## 目录

1 实验目标概述	1
2 实验环境配置	1
3 实验过程	1
3.1 待开发的三个应用场景	1
3.2 面向可复用性和可维护性的设计：IntervalSet<L>	2
3.2.1 IntervalSet<L>的共性操作	2
3.2.2 局部共性特征的设计方案	4
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）	5
3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L>	7
3.3.1 MultiIntervalSet<L>的共性操作	7
3.3.2 局部共性特征的设计方案	10
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）	10
3.4 面向复用的设计：L	13
3.5 可复用 API 设计	14
3.5.1 计算相似度	14
3.5.2 计算时间冲突比例	15
3.5.3 计算空闲时间比例	15
3.6 应用设计与开发	16
3.6.1 排班管理系统	16
3.6.2 操作系统的进程调度管理系统	19
3.6.3 课表管理系统	20
3.7 基于语法的数据读入	22
3.8 应对面临的新变化	24
3.8.1 变化 1	24
3.8.2 变化 2	26
3.9 Git 仓库结构	28
4 实验进度记录	29
5 实验过程中遇到的困难与解决途径	30

---

6 实验过程中收获的经验、教训、感想 .....	30
6.1 实验过程中收获的经验教训 .....	30
6.2 针对以下方面的感受 .....	31

## 1 实验目标概述

本次实验覆盖课程第 2、3 章的内容，目标是编写具有可复用性和可维护性的软件，主要使用以下软件构造技术：

- 子类型、泛型、多态、重写、重载
- 继承、代理、组合
- 语法驱动的编程、正则表达式
- API 设计、API 复用

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

## 2 实验环境配置

实验环境与前两次实验相同，无需再次配置。

GitHub Lab3 仓库的 URL 地址：

<https://github.com/ComputerScienceHIT/HIT-Lab3-1190200526>

## 3 实验过程

### 3.1 待开发的三个应用场景

- (1) 值班表管理(DutyRoster): 为  $n$  名员工安排某个时间段内的值班，要求每天均安排一人值班，且每个人的值班时间若为  $m$  天( $m > 1$ )，则这  $m$  天一定连续。
- (2) 操作系统进程调度管理(ProcessSchedule): 单核 CPU 计算机，每个时间仅能随机调度一个进程运行，其他进程处于休眠状态，且特定时刻可不调度执行任何进程。
- (3) 大学课表管理(CourseSchedule): 简化版课表，可以存在空白时间段，允许同一时间段内安排不同的课程，允许一门课程一周可安排一次或多次。假设由同一位教师承担并在同样的教室进行，一位教师也可以承担课表

中的多门课程。

**相同点：**三个应用场景都存在时间段及时间段集合的概念，都需要实现标签和时间段之间的对应。

**不同点：**

- 第一个场景中标签与时间段一一对应，而第二个、第三个场景较第一个场景略复杂，一个标签可能对应多个不连续的时间段；
- 前两个场景不会出现时间段重叠的情况，而第三个场景会出现时间段重叠；
- 第三个场景中时间段还会呈现为周期性分布。

### 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。

IntervalSet<L>是一个 mutable 的 ADT，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。标签可以是任意 immutable 的 ADT。

#### 3.2.1 IntervalSet<L>的共性操作

共性操作定义在 src/baseADTs 目录下的接口文件 IntervalSet.java 中，此处简要介绍各方法。

- **public static <L> IntervalSet<L> empty()**  
此静态方法创建一个空的 IntervalSet（时间段集合）。规约如下：

```
/**
 * 创建一个空的时间段集合
 *
 * @param <L> 时间段附着的标签的类型，可以是任何immutable的ADT
 * @return 一个新的空时间段集合
 */
```

- **public void insert(long start, long end, L label) throws Exception**

此方法在当前对象中插入新的时间段和标签，并附带检测。规约如下：

```
/**
 * 在集合中插入新的带标签时间段
 *
 * @param start 开始时间
 * @param end 结束时间
 * @param label 标签
 * @throws Exception 标签重复或添加非法时间段时抛出异常
 *                  （非法指空标签或开始时间大于结束时间或时间存在负数）
 */
```

- **public Set<L> labels()**

此方法获得当前对象中的标签集合。规约如下：

```
/**
 * 获得当前时间段集合的标签集合
 *
 * @return 当前对象的标签集合
 */
```

- **public boolean remove(L label)**

此方法从当前对象中移除某个标签所关联的时间段。规约如下：

```
/**
 * 从当前时间段集合中移除某个标签所关联的时间段
 *
 * 如果含有此标签关联的时间段，则进行移除，
 * 否则不进行任何修改。
 *
 * @param label 标签
 * @return 是否执行了移除
 */
```

- **public long start(L label)**

此方法返回某个标签对应的时间段的开始时间。规约如下：

```
/**
 * 返回某个标签对应的时间段的开始时间
 *
 * @param label 标签
 * @return 此标签对应时间段的开始时间
 *         假如此标签未对应时间段返回-1
 */
```

- **public long end(L label)**

此方法返回某个标签对应的时间段的结束时间。规约如下：

```
/**
 * 返回某个标签对应的时间段的结束时间
 *
 * @param label 标签
 * @return 此标签对应时间段的结束时间
 *         假如此标签未对应时间段返回-1
 */
```

- **public boolean isEmpty()**

此方法是我在实验指导书基础上自行添加的，用于检测对象是否为空。规约如下：

```
/**
 * 判断此时间段集合是否为空
 *
 * @return 时间段集合是否为空的判断结果
 */
```

### 3.2.2 局部共性特征的设计方案

src/baseADTs/CommonIntervalSet.java 中的 `CommonIntervalSet<L>` 类继承了上述的 `IntervalSet<L>` 接口，并进行了具体实现。其 rep、AF、RI、Safety from rep exposure 内容如下：

```
// rep / fields:
protected final Set<L> labelSet; // 标签集
protected final Map<L, Long> startMap; // 开始时间记录
protected final Map<L, Long> endMap; // 结束时间记录

// Abstraction function:
// AF(labelSet) = 标签的集合
// AF(startMap) = 标签与时间段开始时间绑定对应
// AF(endMap) = 标签与时间段结束时间绑定对应

// Representation invariant:
// 1) 一个标签有且仅有一个时间段与其对应
// 2) startMap和endMap中元素个数相等
// 3) 时间段开始时间小于结束时间
// 4) 标签不能为null

// Safety from rep exposure:
// 1) 类的属性均设置为protected final
// 2) labels()方法返回mutable属性labelSet时进行defensive copy
```

除实现共性操作方法（较简单，此处略去）之外，还应实现构造函数、`checkRep()`并重写 `toString()`方法，如下图：

用于初始化空对象的构造函数：

```
// constructor1:
public CommonIntervalSet() {
    this.labelSet = new HashSet<>();
    this.startMap = new HashMap<>();
    this.endMap = new HashMap<>();
}
```

用于委托操作对象的构造函数：

```
// constructor2:
public CommonIntervalSet(Set<L> labelSet, Map<L, Long> startMap, Map<L, Long> endMap) {
    this.labelSet = labelSet;
    this.startMap = startMap;
    this.endMap = endMap;
}
```

检查不变性的私有方法 `checkRep()`：

```
// checkRep
private void checkRep() {
    for (L label : labelSet) {
        assert (label != null);
        assert (startMap.get(label) != null && endMap.get(label) != null);
        assert (startMap.get(label) < endMap.get(label));
    }
    assert (labelSet.size() == startMap.size());
    assert (startMap.size() == endMap.size());
}
```

将对象转换为可读表示的 `toString()` 方法：

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("IntervalSet: Label Cnt:");
    sb.append(labelSet.size());
    sb.append(" Intervals:");
    for (L label : labelSet) {
        sb.append(label.toString() + ":" +
            startMap.get(label).toString() + "-"
            + endMap.get(label).toString() + " ");
    }
    checkRep();
    return sb.toString();
}
```

测试代码见 `test/baseADTs/CommonIntervalSetTest.java`，此处略去。

测试结果如下：

Runs: 7/7

Errors: 0

Failures: 0

baseADTs.CommonIntervalSetTest [Runner: JUnit 4] (0.001 s)

testObservers (0.001 s)

testEmpty (0.000 s)

testToString (0.000 s)

testAssertionsEnabled (0.000 s)

testInsert (0.000 s)

testIsEmpty (0.000 s)

testRemove (0.000 s)

CommonIntervalSet.java

CommonIntervalSet<L>

checkRep()

CommonIntervalSet()

end(L)

insert(long, long, L)

isEmpty()

labels()

remove(L)

start(L)

toString()

IntervalSet.java

IntervalSet<L>

empty() <L>

	92.5 %	273	22	295
	92.5 %	273	22	295
	75.9 %	63	20	83
	100.0 %	18	0	18
	100.0 %	16	0	16
	100.0 %	47	0	47
	100.0 %	8	0	8
	100.0 %	8	0	8
	100.0 %	26	0	26
	100.0 %	16	0	16
	100.0 %	65	0	65
	100.0 %	4	0	4
	100.0 %	4	0	4
	100.0 %	4	0	4

### 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）

我选择使用实验指导书中的方案五（CRP，通过接口组合实现局部共性特征的复用）来进行设计，首先需完成了三个维度的接口、具体实现，但此处仅能实现第一个维度——“无重叠”维度。

此维度操作的相关接口和具体实现分别在 `src/specialOps` 目录下的文件 `NonOverlapIntervalSet.java`、`NonOverlapIntervalSetImpl.java` 中。

我的思路是实现一个先检查是否会重叠再进行插入操作的 `insert()` 方法，故接口中仅定义了这一方法。具体规约如下：

（见下页）



```

/**
 * 时间段无重叠的插入方法
 *
 * @param start 开始时间
 * @param end 结束时间
 * @param label 标签
 * @throws Exception 非法时间段、标签重复、时间段重叠抛出异常
 */

```

```

public void insert(long start, long end, L label) throws Exception;

```

实现思路是在实现类中执行对 `IntervalSet<L>` 的重叠检测和插入，具体实现如下：

```

public class NonOverlapIntervalSetImpl<L> implements NonOverlapIntervalSet<L> {

    private final IntervalSet<L> isl;

    // 用于传递对象，实现委托操作
    public NonOverlapIntervalSetImpl(IntervalSet<L> isl) {
        this.isl = isl;
    }

    @Override
    public void insert(long start, long end, L label) throws Exception {
        isOverLap(start, end, label);
        isl.insert(start, end, label);
    }

    public void isOverLap(long start, long end, L label) throws IntervalConflictException {
        for (L l : isl.labels()) {
            long s = isl.start(l);
            long e = isl.end(l);
            if (!(s > end || e < start)) {
                throw new IntervalConflictException("与[" + l.toString() + "]" + "存在时间段重叠!");
            }
        }
    }
}

```

在实现这个维度的基础上，现在可以具体实现排班表应用的 `IntervalSet` 子类型(具体代码可查看 `src/appADTs` 目录下的源文件 `IDutyIntervalSet.java`、`DutyIntervalSet.java`)，以下是对这此应用对应 `IntervalSet` 子类型设计、实现的简要说明：

rep、AF、RI、Safety from rep exposure 内容如下：

```

// fields:
private final NonOverlapIntervalSetImpl<L> nois;
private final NoBlankIntervalSetImpl<L> nbis;

// Abstraction function:
// 沿用CommonIntervalSet<L>的AF
// 此外在nbis中保存了排班的起止日期
// 整体代表一个排班表

// Representation invariant:
// 沿用CommonIntervalSet<L>的RI

// Safety from rep exposure:
// 大部分在CommonIntervalSet<L>中保证
// 类的属性均设置为private final

```

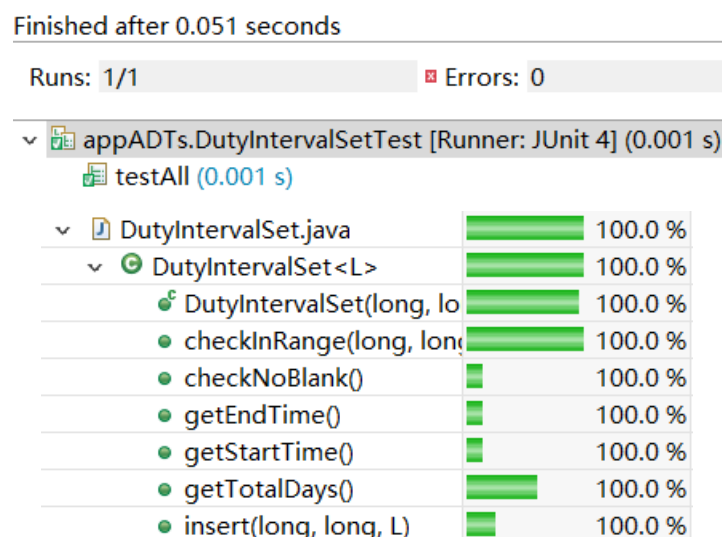
接口和具体实现类的继承、实现关系如下：

```
public class DutyIntervalSet<L> extends CommonIntervalSet<L> implements
IDutyIntervalSet<L>
public interface IDutyIntervalSet<L> extends NonOverlapIntervalSet<L>,
NoBlankIntervalSet<L>
```

具体实现中方法较少，主要是进行委托绑定（传递对象引用）、获取一些信息用于应用设计时处理、输出，此处略去。后来在实际开发 APP 时添加了 `checkInRange()` 方法检查输入合法性，具体实现如下：

```
@Override
public boolean checkInRange(long start, long end) {
    long startTime = nbis.getStartTime();
    long endTime = nbis.getEndTime();
    if (startTime <= start && start <= endTime && startTime <= end && end <= endTime)
        return true;
    return false;
}
```

测试代码见 `test/appADTs/DutyIntervalSetTest.java`，此处略去。  
测试结果如下：



### 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

我选择将 `MultiIntervalSet<L>` 实现为具体类，rep、AF、RI、Safety from rep exposure 内容如下：

```
// rep / fields:
protected final List<IntervalSet<L>> multiList;

// Abstraction function:
// 由多个内部无重复标签的时间段集合组成，
// 但各集合间可能有相同标签但时间不同的时间段，
// 以此表示一个标签对应多个时间段。
```

```
// Representation invariant:
// 1) 各集合内部符合IntervalSet<L>的RI
// 2) mutiList不为null

// Safety from rep exposure:
// 1) 类的属性设置为protected
// 2) 没有公共方法返回对mutable属性的引用
// 3) 带参数的构造函数复制参数，避免参数修改导致对象属性被修改
```

共性方法有以下几个：

- **public void insert(long start, long end, L label) throws Exception**

此方法在当前对象中插入新的时间段和标签，具体规约如下：

```
/**
 * 在当前对象中插入新的时间段和标签
 *
 * @param start 开始时间
 * @param end 结束时间
 * @param label 标签
 * @throws Exception 非法时间段抛出异常
 */
```

- **public Set<L> labels()**  
此方法获取当前对象的标签集合。

- **public boolean remove(L label)**

规约如下：

```
/**
 * 从当前对象中移除某个标签所关联的所有时间段
 *
 * 如果含有此标签对应的时间段，则进行移除，
 * 否则不进行任何修改。
 *
 * @param label 标签
 * @return 是否执行了移除
 */
```

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

规约如下：

```
/**
 * 从当前对象中获取与某个标签所关联的所有时间段
 * 其中的时间段按开始时间从小到大的次序排列
 *
 * @param label 标签
 * @return 按开始时间从小到大的次序排列的时间段集合
 * 如果没有此标签对应的时间段，返回空集
 */
```

- **public boolean isEmpty()**

规约如下：

```
/**
```

```

* 判断对象是否为空
*
* @return 对象是否为空的判断结果
*/

```

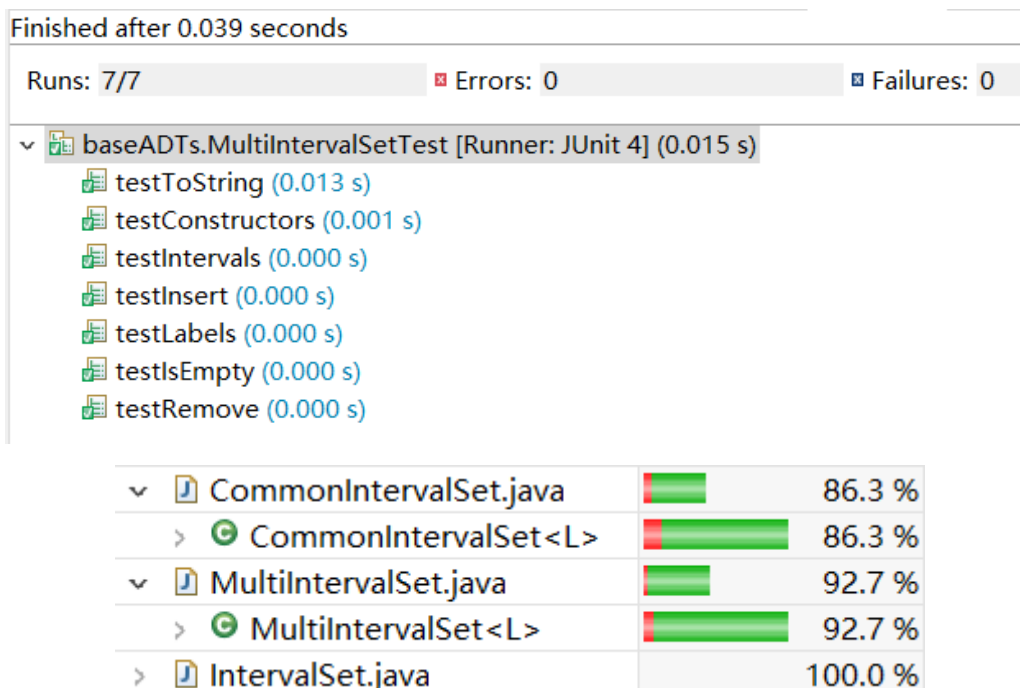
这些方法中比较难实现的是 `intervals()` 方法，通过此方法可以体现 `MultiIntervalSet<L>` 的大部分结构和内容。核心部分代码如下：

```

// 如果没有此标签对应的时间段，返回空集
if (!this.labels().contains(label))
    return IntervalSet.empty();
// 用于排序的Map, key记录起始时间, value为时间段集合在对象中的索引值
Map<Long, Integer> sort = new TreeMap<>();
for (int i = 0; i < multilist.size(); ++i) {
    if (multilist.get(i).start(label) != -1) {
        sort.put(multilist.get(i).start(label), i);
    }
}
Collection<Integer> c = sort.values();
Iterator<Integer> it = c.iterator();
IntervalSet<Integer> result = IntervalSet.empty();
int cnt = 0;
try {
    while (it.hasNext()) {
        // 按开始时间次序向结果集添加项
        int v = (int)it.next();
        long s = multilist.get(v).start(label);
        long e = multilist.get(v).end(label);
        result.insert(s, e, cnt);
        ++cnt;
    }
} catch (Exception e) { // 正常情况下不会抛出异常

```

测试代码此处略，测试结果如下：



可以看到，`IntervalSet` 的接口、具体实现类代码覆盖度也比较高，这是因为 `MultiIntervalSet` 进行了复用。

### 3.3.2 局部共性特征的设计方案

由于 `MultiIntervalSet<L>` 已实现为具体类，局部共性特征的设计和实现也体现在 3.3.1 节中，此处不再重复。

### 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）

至此，后两个维度也可以进一步实现。以下是关于这两个维度设计实现的简要说明：

#### (1) “非空” 维度：

此维度操作的相关接口和具体实现分别在 `src/specialOps` 目录下的文件 `NoBlankIntervalSet.java`、`NoBlankIntervalSetImpl.java` 中。

我的思路是实现一个检查是否存在空白并输出相关信息的 `checkNoBlank()` 方法，此方法在接口中定义如下：

```
/**
 * 检查是否存在空白时间段
 * 输出相关信息
 */
public void checkNoBlank();
```

实现时，我在实现类中使用私有属性保存对象起止时间和传入的对象引用，检查时使用这一起止时间，统计空白并输出信息。私有属性定义如下：

```
private final long startTime;
private final long endTime;
private final CommonIntervalSet<L> cisl;
```

具体代码此处略去。

#### (2) “周期性” 维度：

此维度操作的相关接口和具体实现分别在 `src/specialOps` 目录下的文件 `PeriodicIntervalSet.java`、`PeriodicIntervalSetImpl.java` 中。

我的思路是实现一个进行周期性插入的 `insert()` 方法，此方法在接口中定义如下：

```
/**
 * 插入周期性重复时间段的插入方法
 *
 * @param start 开始时间
 * @param end 结束时间
 * @param label 标签
 * @throws Exception 非法时间段、超出时间范围抛出异常
 */
public void insert(long start, long end, L label) throws Exception;
```

实现时，我在实现类中使用私有属性保存开始时间、周期长度、周期数和传入的对象引用，插入时使用这些信息进行周期性插入，核心代码如下：

（见下页）

```

@Override
public void insert(long start, long end, L label) throws Exception {
    if (end > period) {
        throw new TimeOutOfRangeException();
    }
    for (int i = 0; i < this.n; ++i) {
        misl.insert(startTime+start+period*i, startTime+end+period*i, label);
    }
}

```

除此之外，要保证进行委托的对象能获取到周期相关信息，还实现了获取开始时间、周期数的公共方法，此处略。

在完成三个维度全部内容的设计实现的基础上，我随后完成了对后两个应用 `MultiIntervalSet` 子类型的设计和实现：

#### (1) `ProcessIntervalSet<L>`:

rep、AF、RI、Safety from rep exposure 内容如下：

```

// fields:
NonOverlapIntervalSetImpl<L> nois;

// Abstraction function:
// 沿用MultiIntervalSet<L>的AF
// 整体代表一个操作系统对进程的调度记录

// Representation invariant:
// 沿用MultiIntervalSet<L>的RI

// Safety from rep exposure:
// 大部分在MultiIntervalSet<L>中保证
// 类的属性设置为private final

```

接口和具体实现类的继承、实现关系如下：

```

public interface IProcessIntervalSet<L> extends NonOverlapIntervalSet<L>
public class ProcessIntervalSet<L> extends MultiIntervalSet<L> implements

```

`IProcessIntervalSet<L>`

具体实现中仅有一个 `insert()` 方法，此方法重写 `MultiIntervalSet<L>` 中的 `insert()` 方法，实现了“无重叠”，代码如下：

```

@Override
public void insert(long start, long end, L label) throws Exception {
    int i;
    for (i = 0; i < this.multilist.size(); ++i) {
        nois = new NonOverlapIntervalSetImpl<>(this.multilist.get(i));
        nois.isOverLap(start, end, label); // 检查是否重叠
    }
    for (i = 0; i < this.multilist.size(); ++i) {
        if (!this.multilist.get(i).labels().contains(label)) { // 找到插入位置
            this.multilist.get(i).insert(start, end, label);
            return;
        }
    }
    this.multilist.add(IntervalSet.empty());
    this.multilist.get(i).insert(start, end, label);
}

```

#### (2) `CourseIntervalSet<L>`:

rep、AF、RI、Safety from rep exposure 内容如下：

(见下页)

```
// fields:
private final PeriodicIntervalSetImpl<L> pis;

// Abstraction function:
// 沿用MultiIntervalSet<L>的AF
// 整体代表一个班级的特定课表

// Representation invariant:
// 沿用MultiIntervalSet<L>的RI

// Safety from rep exposure:
// 大部分在MultiIntervalSet<L>中保证
// 类的属性设置为private final
```

接口和具体实现类的继承、实现关系如下：

```
public interface ICourseIntervalSet<L> extends PeriodicIntervalSet<L>
public class CourseIntervalSet<L> extends MultiIntervalSet<L> implements
ICourseIntervalSet<L>
```

具体实现包括 `insert()` 和一些获取信息的方法，`insert()` 方法重写 `MultiIntervalSet<L>` 中的 `insert()` 方法，通过委托实现了“周期性”，代码如下：





```
@Override
public void insert(long start, long end, L label) throws Exception {
    this.pis.insert(start, end, label);
}
```

至此，三个应用所需的 ADT 均已实现。





后实现的两个应用 ADT 的测试代码在 `test/appADTs` 目录下，此处略。

测试结果如下：

Finished after 0.035 seconds

Runs: 1/1	Errors: 0	Failures: 0
appADTs.ProcessIntervalSetTest [Runner: JUnit 4] (0.001 s)		
testProcessIntervalSet (0.001 s)		
ProcessIntervalSet.java		100.0 %
ProcessIntervalSet<L>		100.0 %
ProcessIntervalSet()		100.0 %
insert(long, long, L)		100.0 %

Finished after 0.036 seconds

Runs: 1/1	Errors: 0	Failures: 0
appADTs.ProcessIntervalSetTest [Runner: JUnit 4] (0.000 s)		
testProcessIntervalSet (0.000 s)		
ProcessIntervalSet.java		100.0 %
ProcessIntervalSet<L>		100.0 %
ProcessIntervalSet()		100.0 %
insert(long, long, L)		100.0 %

### 3.4 面向复用的设计：L

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

- Employee: 姓名、职务、手机号码

```
// rep / fields:
private final String name;
private final String position;
private final String phoneNumber;

// Abstraction function:
/**
 * AF(name) = 姓名
 * AF(position) = 职务
 * AF(phoneNumber) = 手机号码
 */

// Representation invariant:
// 无要求

// Safety from rep exposure:
// 属性均设置为private final
```

- Process: 进程 ID、进程名称、最短执行时间、最长执行时间

```
// rep / fields:
private final int processID;
private final String processName;
private final long minTime;
private final long maxTime;

// Abstraction function:
/**
 * AF(processID) = 进程ID
 * AF(processName) = 进程名
 * AF(minTime) = 最短执行时间
 * AF(maxTime) = 最长执行时间
 */

// Representation invariant:
// 无要求

// Safety from rep exposure:
// 属性均设置为private final
```

- Course: 课程 ID、课程名称、教师名字、地点、周学时（自行添加）

```
// rep / fields:
private final int courseID;
private final String courseName;
private final String teacherName;
private final String location;
private final int hourPerWeek;
```

（接下页）



```
// Abstraction function:
/**
 * AF(courseID) = 课程ID
 * AF(courseName) = 课程名
 * AF(teacherName) = 教师名字
 * AF(location) = 地点
 */

// Representation invariant:
// 无要求

// Safety from rep exposure:
// 属性均设置为private final
```

除此之外，各类中还有获取每个私有属性的公共方法，并重写了 `toString()`、`hashCode()`、`equals()` 方法，便于开发 APP 时使用。

具体实现可见 `src/labels` 目录下三个源文件中的代码。

### 3.5 可复用 API 设计

API 全部代码可查看 `src/baseADTs` 目录下 `APIs.java` 文件。API 各方法实现思路类似，`MultiIntervalSet<L>` 和 `IntervalSet<L>` 的处理也类似，故在本节中仅进行简单展示。

#### 3.5.1 计算相似度

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。

我的实现思路如下：首先将两个对象里的时间段进行排序，获得开始、结束时间及总长度（结束时间 - 开始时间），随后遍历比较两个对象中已排序的时间段，判断是否相似并统计相似长度，最后计算相似度。主体代码如下：

```
if (s1.isEmpty() || s2.isEmpty()) // 存在空对象
    return 0.0;
List<Pair<Long, Long>> l1 = new ArrayList<>();
List<Pair<Long, Long>> l2 = new ArrayList<>();
for (IntervalSet<L> isl : s1.multiList) {
    for (L l : isl.labels()) {
        l1.add(new Pair<>(isl.start(l), isl.end(l))); // 保存到可排序的List中
    }
}
for (IntervalSet<L> isl : s2.multiList) {
    for (L l : isl.labels()) {
        l2.add(new Pair<>(isl.start(l), isl.end(l))); // 同样保存到可排序的List中
    }
}

l1.sort(new TimeComparator2());
l2.sort(new TimeComparator2());
long end = l1.get(0).getValue() < l2.get(0).getValue() ? l2.get(0).getValue() : l1.get(0).getValue();
l1.sort(new TimeComparator1());
l2.sort(new TimeComparator1());
```

```

long start = l1.get(0).getKey() < l2.get(0).getKey() ? l1.get(0).getKey() : l2.get(0).getKey();
long total = end - start + 1;
long cnt = 0;

// 遍历检查相似
int i = 0;
int j = 0;
while (i < l1.size() && j < l2.size()) {
    long a1, a2, b1, b2;
    a1 = l1.get(i).getKey();
    b1 = l1.get(i).getValue();
    a2 = l2.get(j).getKey();
    b2 = l2.get(j).getValue();
    if (a1 == a2 && b1 == b2) { // 相似
        cnt += b1 - a1 + 1;
        ++i;
        ++j;
    } else if (a1 < a2) {
        ++i;
    } else {
        ++j;
    }
}
return (double)cnt / (double)total;

```

### 3.5.2 计算时间冲突比例

这里的“冲突”是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以总长度，得到冲突比例，是一个[0,1]之间的值。

思路与 3.5.1 类似，首先同样使用 List 进行排序并获得开始、结束时间及总长度（结束时间 - 开始时间），但遍历的过程有所不同：为避免重复计算多次重叠的同一时间段，导致结果数值偏大甚至超过 1，遍历中需要变量记录当前遍历时间段结束位置，并不断更新，避免重复发生。主体代码如下：

```

long total = end - start + 1;
long next = -1;
for (int i = 0; i < setL.size() - 1; ++i) {
    long a1, b1, a2, b2;
    a1 = Math.max(setL.get(i).getKey(), next);
    b1 = Math.max(setL.get(i).getValue(), next);
    a2 = Math.max(setL.get(i + 1).getKey(), next+1);
    b2 = Math.max(setL.get(i + 1).getValue(), next+1);
    if (!(a2 > b1)) { // 存在重叠
        long cStart = a1 > a2 ? a1 : a2;
        long cEnd = b1 > b2 ? b2 : b1;
        conflict += cEnd - cStart + 1;
        next = cEnd; // 避免重复计算同一冲突时间段
    }
}
return (double)conflict / (double)total;

```

### 3.5.3 计算空闲时间比例

这里的“空闲”是指某时间段内没有安排任何 interval 对象。用空闲的时间段总长度除以总长度，得到空闲比例，是一个[0,1]之间的值。

思路依然类似，略有不同的还是遍历，这次需要记录当前检查到的位置，并与排序好的下一个时间段开始位置进行比较，若下一个开始位置在当前位置之后，则将二者距离加入到空闲时间统计中。无论是否有空闲，每次遍历结束时都要更新当前位置为此时间段结束位置和当前位置的较大值。主体代码如下：

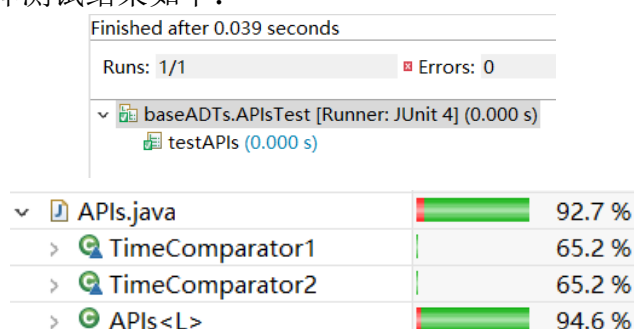
```

long freeTime = 0;
setL.sort(new TimeComparator2());
long end = setL.get(0).getValue();
setL.sort(new TimeComparator1());
long start = setL.get(0).getKey();
long total = end - start + 1;
long cur = setL.get(0).getValue();
int i = 1;
while (i < setL.size()) {
    if (cur < setL.get(i).getKey()) {
        freeTime += setL.get(i).getKey() - cur;
    }
    cur = setL.get(i).getValue() > cur ? setL.get(i).getValue() : cur;
    ++i;
}
return (double)freeTime / (double)total;

```

测试代码此处略，可查看 test/baseADTs/APIsTest.java 文件。

API 部分整体测试结果如下：



## 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。主要的工作是利用这些已实现的方法完成实际操作。此部分源代码可见 src/apps 目录下的三个源代码文件。

### 3.6.1 排班管理系统

APP 设计为命令程序，完成了所要求功能的实现。具体代码此处略。这里总结几个要点：

- (1) 使用一个 **HashSet** 暂存了员工信息，以便执行各种操作时使用；
- (2) 输入会涉及到合法性检测，需要注意（编写并使用了 **checkInRange()** 方法）；
- (3) 此应用涉及到大量的日期转换、处理等操作，故创建了帮助类 **dateHandler**；
- (4) 应用捕获了之前自定义的异常类，并输出相关信息帮助用户了解问题所在并进行修改。

运行效果截图（部分功能展示）：

```
-----排班管理系统-----
请输入相对文件路径（如“txt/test1.txt”，输入“#”或文件读入错误会跳过文件读入，切换至手动操作）：
#
请输入排班开始日期（格式：yyyy-MM-dd）： 2021-03-01
请输入排班结束日期（格式：yyyy-MM-dd）： 2021-04-30
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 1
-----添加员工信息-----
请输入员工1姓名、职务和手机号（空格隔开，输入#结束）： Abcd efghijkl 123-4567-8910
添加成功！
请输入员工2姓名、职务和手机号（空格隔开，输入#结束）： Bcde fghijklm 134-5678-9012
添加成功！
请输入员工3姓名、职务和手机号（空格隔开，输入#结束）： Cdef ghijklmn 145-6789-1257
添加成功！
请输入员工4姓名、职务和手机号（空格隔开，输入#结束）： Defg hijklmno 156-7890-3469
添加成功！
请输入员工5姓名、职务和手机号（空格隔开，输入#结束）： #
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 3
-----查看员工信息-----
总人数： 4
序号   姓名           职务           手机号码
1      Defg           hijklmno       156-7890-3469
2      Abcd           efghijkl       123-4567-8910
3      Bcde           fghijklm       134-5678-9012
4      Cdef           ghijklmn       145-6789-1257
按enter以返回功能菜单：

-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 6
-----检查排班情况-----
空白段1: 2021-03-01~2021-04-30
空白所占比例：100%（未进行排班）
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 5
-----自动生成排班-----
自动排班成功！
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
```

```

请选择（输入功能对应数字，其他退出）： 6
-----检查排班情况-----
空白段1: 2021-03-01~2021-04-30
空白所占比例：100%（未进行排班）
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 5
-----自动生成排班-----
自动排班成功！
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 7
-----排班表-----

```

序号	开始日期	结束日期	姓名	职务	手机号码
1	2021-03-01	2021-03-16	Defg	hijklmno	156-7890-3469
2	2021-03-17	2021-03-31	Abcd	efghijkl	123-4567-8910
3	2021-04-01	2021-04-15	Bcde	fghijklm	134-5678-9012
4	2021-04-16	2021-04-30	Cdef	ghijklmn	145-6789-1257

```

按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 6
-----检查排班情况-----
当前排班已无空白！
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 2
-----删除员工信息-----
请输入要删除的员工姓名： Bcde
员工[Bcde]存在排班信息，将被一同删除！
是否确认删除（输入Y/y确认，其他取消）： y
删除成功！
按enter以返回功能菜单：

```

```

-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 6
-----检查排班情况-----
空白段1: 2021-04-01~2021-04-15
空白所占比例: 24.59%
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 7
-----排班表-----

```

序号	开始日期	结束日期	姓名	职务	手机号码
1	2021-03-01	2021-03-16	Defg	hijklmno	156-7890-3469
2	2021-03-17	2021-03-31	Abcd	efghijkl	123-4567-8910
3	2021-04-16	2021-04-30	Cdef	ghijklmn	145-6789-1257

```

按enter以返回功能菜单：

```

### 3.6.2 操作系统的进程调度管理系统

APP 设计为命令行程序，完成了所要求功能的实现。具体代码此处略。这里总结几个要点：

- (1) 进程调度中的随机我选择使用 Java 的 `Math.random()` 实现，并开发了相关的帮助类 `randomGenerator`（可查看 `src/helpers` 目录下的源代码文件 `randomGenerator.java`）；
- (2) 两个策略均进行了实现，同时考虑到调度不应重复调度同一进程或空闲，设置了检测和处理；
- (3) 最短进程优先策略在执行前对进程按最大执行时间进行了排序，两种策略均使用 `List`（或 `Map`）实现动态修改进程可执行的剩余时间（或已运行时间），以实现进程的调度时间限制与终止；
- (4) 程序中通过私有属性 `freeMax` 设置了进程调度过程中每次空闲的最大时间为 **100**，可进行修改。

运行效果截图：

```

-----进程调度管理系统-----
-----功能菜单-----
[1] 添加一组进程
[2] 随机选择进程模拟调度
[3] 最短进程优先模拟调度
请选择（输入功能对应数字，其他退出）： 1
-----添加进程信息-----
请按顺序输入进程ID、名称、最短执行时间、最长执行时间（空格隔开，输入#结束）：
1234 System 100 150
请按顺序输入进程ID、名称、最短执行时间、最长执行时间（空格隔开，输入#结束）：
2345 user 120 130
请按顺序输入进程ID、名称、最短执行时间、最长执行时间（空格隔开，输入#结束）：
3456 abcde 200 250
请按顺序输入进程ID、名称、最短执行时间、最长执行时间（空格隔开，输入#结束）：
4567 bcdef 400 500
请按顺序输入进程ID、名称、最短执行时间、最长执行时间（空格隔开，输入#结束）：
5678 fffff 510 650
请按顺序输入进程ID、名称、最短执行时间、最长执行时间（空格隔开，输入#结束）：
#
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组进程
[2] 随机选择进程模拟调度
[3] 最短进程优先模拟调度
请选择（输入功能对应数字，其他退出）： 2

```

```

-----随机选择进程-----
时刻  调度进程ID  调度进程名  最短执行时间  最长执行时间  本次调度结束时状态  本次调度时间  进程总运行时间
0      4567      bcdef      400          500          执行结束        419          419
420    5678      fffff      510          650          执行结束        629          629
1050   2345      user       120          130          待继续          47           47
1098   空闲      空闲       空闲         空闲         空闲            38           -
1137   1234      System     100          150          执行结束        143          143
1281   空闲      空闲       空闲         空闲         空闲            65           -
1347   3456      abcde      200          250          待继续          125          125
1473   2345      user       120          130          待继续          58           105
1579   3456      abcde      200          250          待继续          25           150
1730   空闲      空闲       空闲         空闲         空闲            13           -
1744   3456      abcde      200          250          执行结束        91           241
1836   空闲      空闲       空闲         空闲         空闲            82           -
1919   2345      user       120          130          执行结束        20           125
-----模拟调度结束-----
按enter以返回功能菜单: 3

-----功能菜单-----
[1] 添加一组进程
[2] 随机选择进程模拟调度
[3] 最短进程优先模拟调度
请选择(输入功能对应数字, 其他退出): 3
-----最短进程优先-----
时刻  调度进程ID  调度进程名  最短执行时间  最长执行时间  本次调度结束时状态  本次调度时间  进程总运行时间
0      2345      user       120          130          待继续          4             4
5      空闲      空闲       空闲         空闲         空闲            20            -
26     2345      user       120          130          执行结束        117           121
144    空闲      空闲       空闲         空闲         空闲            53            -
198    1234      System     100          150          待继续          76            76
275    空闲      空闲       空闲         空闲         空闲            20            -
296    1234      System     100          150          执行结束        42            118
339    空闲      空闲       空闲         空闲         空闲            62            -
402    3456      abcde      200          250          待继续          142           142
545    空闲      空闲       空闲         空闲         空闲            26            -
572    3456      abcde      200          250          执行结束        85            227
658    空闲      空闲       空闲         空闲         空闲            32            -
691    4567      bcdef      400          500          执行结束        460           460
1152   空闲      空闲       空闲         空闲         空闲            34            -
1187   5678      fffff      510          650          执行结束        629           629
按enter以返回功能菜单:
-----功能菜单-----
[1] 添加一组进程
[2] 随机选择进程模拟调度
[3] 最短进程优先模拟调度
请选择(输入功能对应数字, 其他退出): 0
-----程序终止-----

```

### 3.6.3 课表管理系统

APP 设计为命令行程序，完成了所要求功能的实现。具体代码此处略。这里总结几个要点：

- (1) 此应用再次涉及到日期相关处理，故完善并使用了 `dateHandler` 类，增加了计算日期对应一周第几天的方法，用于获取单日课程表；
- (2) 一周七天这一周期通过私有变量 `period` 确定；
- (3) 使用私有属性 `hourMap` 保存每门课程剩余的可安排周学时数，以便安排课程时进行检测及输出安排情况信息；
- (4) 添加课程、安排课程要考虑输入的较多情况，故代码较长；
- (5) 空闲时间、重复时间比例调用 API 中的方法进行计算。

运行效果截图（部分）：

```

-----课表管理系统-----
请输入学期开始日期(格式: yyyy-MM-dd): 2021-03-01
请输入学期总周数(例如: 18): 18
-----功能菜单-----
[1] 添加一组课程
[2] 安排课程时间
[3] 查看安排情况
[4] 单日课表查询
请选择(输入功能对应数字, 其他退出): 1
-----添加课程信息-----
请按顺序输入课程ID、课程名称、教师姓名、地点和周学时数(空格隔开, 输入#结束):
1234 Software Teachera Zhengxin21 6
添加成功!

```



请按顺序输入课程ID、课程名称、教师姓名、地点和周学时数（空格隔开，输入#结束）：

2345 CSAPP Teacherb Zhizhi34 6

添加成功！

请按顺序输入课程ID、课程名称、教师姓名、地点和周学时数（空格隔开，输入#结束）：

3456 abcde Teacherc abcde45 4

添加成功！

请按顺序输入课程ID、课程名称、教师姓名、地点和周学时数（空格隔开，输入#结束）：

1 2

添加失败：输入格式错误！

按enter以返回功能菜单：

-----功能菜单-----

[1] 添加一组课程

[2] 安排课程时间

[3] 查看安排情况

[4] 单日课表查询

请选择（输入功能对应数字，其他退出）： 2

-----安排课程时间-----

请输入要进行安排课程的ID： 3456

请输入安排的上课时间（如周四，8-10时应输入“4 8-10”）： 5 10-12

安排成功！

按enter以返回功能菜单：

-----功能菜单-----

[1] 添加一组课程

[2] 安排课程时间

[3] 查看安排情况

[4] 单日课表查询

请选择（输入功能对应数字，其他退出）： 2

-----安排课程时间-----

请输入要进行安排课程的ID： 3456

请输入安排的上课时间（如周四，8-10时应输入“4 8-10”）： 4 8-10

安排成功！

按enter以返回功能菜单：

（省略了一部分安排过程）

-----功能菜单-----

[1] 添加一组课程

[2] 安排课程时间

[3] 查看安排情况

[4] 单日课表查询

请选择（输入功能对应数字，其他退出）： 3

-----查看安排情况-----

课程ID	课程名称	教师姓名	地点	剩余/总周学时	安排情况
2345	CSAPP	Teacherb	Zhizhi34	6 / 6	待安排
1234	Software	Teacher a	Zhengxin21	2 / 6	待安排
3456	abcde	Teacherc	abcde45	0 / 4	安排完成

空闲时间比例：88.21%

重复时间比例：0.00%

按enter以返回功能菜单：

-----功能菜单-----

[1] 添加一组课程

[2] 安排课程时间

[3] 查看安排情况

[4] 单日课表查询

请选择（输入功能对应数字，其他退出）： 3

-----查看安排情况-----

课程ID	课程名称	教师姓名	地点	剩余/总周学时	安排情况
2345	CSAPP	Teacherb	Zhizhi34	2 / 6	待安排
1234	Software	Teacher a	Zhengxin21	2 / 6	待安排
3456	abcde	Teacherc	abcde45	0 / 4	安排完成

空闲时间比例：85.27%

重复时间比例：2.95%

按enter以返回功能菜单：



```

-----功能菜单-----
[1] 添加一组课程
[2] 安排课程时间
[3] 查看安排情况
[4] 单日课表查询
请选择（输入功能对应数字，其他退出）： 4
-----单日课表查询-----
请输入要查询的日期（格式：yyyy-MM-dd）： 2021-03-02
2021-03-02 （周二） 课程表
---08:00 - 10:00---
[1234] Software Teachera@Zhengxin21
[2345] CSAPP Teacherb@Zhizhi34
---10:00 - 12:00---
---13:00 - 15:00---
---15:00 - 17:00---
---19:00 - 21:00---
按enter以返回功能菜单：

```

### 3.7 基于语法的数据读入

修改“排班管理”应用以扩展该功能。

我根据实验指导书要求编写了以下正则表达式：

```

String pattern_Name = "[A-Za-z]+";
String pattern_Position = "[A-Za-z\\s]+";
String pattern_Phone = "[0-9]{3}\\-[0-9]{4}\\-[0-9]{4}";
String pattern_Date = "[0-9]{4}-[0-9]{2}-[0-9]{2}";
String pattern_E1 = "\\s*Employee\\s*";
String pattern_E2 = "\\s*" + pattern_Name + "\\{" + pattern_Position + "\\," + pattern_Phone + "\\}";
String pattern_P = "\\s*Period\\{" + pattern_Date + "," + pattern_Date + "\\}";
String pattern_R1 = "\\s*Roster\\s*";
String pattern_R2 = "\\s*" + pattern_Name + "\\{" + pattern_Date + "," + pattern_Date + "\\}";

```

使用时，通过 `Pattern.matches()` 检查文件语法正确性并寻找各部分信息的起始位置，使用 `Pattern` 和 `Matcher` 的相关方法来进行局部截取获取信息，由于程序较长，此处截取一小部分展示：

```

// 找到“Employee{”
do {
    line = br.readLine();
} while (!Pattern.matches(pattern_E1, line) && line != null);
if (line == null) {
    System.out.println("读入失败：文件中不存在员工信息！");
    br.close();
    return false;
}
line = br.readLine();
// 读入员工信息
do {
    if (!Pattern.matches(pattern_E2, line)) {
        System.out.println("读入失败：\"\" + line + "\"\" + \"格式错误！");
        infoClear(br);
        return false;
    } else {
        Pattern p = Pattern.compile(pattern_Name);
        Matcher m = p.matcher(line);
        m.find();
        String name = line.substring(m.start(), m.end()); // 截取姓名
        int tmp = m.end() + 1;
        p = Pattern.compile(pattern_Phone);
        m = p.matcher(line);
        m.find();
        String position = line.substring(tmp, m.start() - 1); // 截取职务
        String phone = line.substring(m.start(), m.end()); // 截取手机号
        eSet.add(new Employee(name, position, phone));
    }
    line = br.readLine();
} while (!Pattern.matches("\\s*\\}", line) && line != null);

```

## 文件读取效果展示（部分测试用例）：

```
-----排班管理系统-----
请输入相对文件路径（如“txt/test1.txt”，输入“#”或文件读入错误会跳过文件读入，切换至手动操作）：
txt/test2.txt
读入失败：“ ZhaoLiu1{Professor,138-1920-3912}”格式错误！
请输入排班开始日期（格式：yyyy-MM-dd）：
```

（姓名中不应含有数字）

```
-----排班管理系统-----
请输入相对文件路径（如“txt/test1.txt”，输入“#”或文件读入错误会跳过文件读入，切换至手动操作）：
txt/test3.txt
读入失败：“ ZhaoLiue{Professor,178-192-03912}”格式错误！
请输入排班开始日期（格式：yyyy-MM-dd）：
```

（手机号格式不正确）

```
-----排班管理系统-----
请输入相对文件路径（如“txt/test1.txt”，输入“#”或文件读入错误会跳过文件读入，切换至手动操作）：
txt/test7.txt
文件读入成功！
```

-----功能菜单-----

```
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 3
```

-----查看员工信息-----

```
总人数：12
序号    姓名                职务                手机号码
1      ZhaoLiuJ              AssociateProfessor   138-1920-0044
2      ZhaoLiuc               Professor            138-1922-3912
3      ZhaoLiua               Professor            138-1920-3912
4      LiSi                   Secretary            151-0101-0000
5      ZhaoLiug               Professor            138-1920-0000
6      ZhangSan              Manger              139-0451-0000
7      ZhaoLiuH              Associate Professor  138-1929-0000
8      WangWu                Associate Dean       177-2021-0301
9      ZhaoLiub              Lecturer            138-1921-3912
10     ZhaoLiuk               Professor            188-1920-0000
11     ZhaoLiud              Lecturer            198-1920-3912
12     ZhaoLiuf              Lecturer            138-1929-3912
```

按enter以返回功能菜单：

-----功能菜单-----

```
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 6
```

-----检查排班情况-----

空白段1:2021-02-09~2021-02-15

空白段2:2021-03-02~2021-03-04

空白所占比例：17.86%

按enter以返回功能菜单：

```
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 7
```

-----排班表-----					
序号	开始日期	结束日期	姓名	职务	手机号码
1	2021-01-10	2021-01-11	ZhangSan	Manger	139-0451-0000
2	2021-01-12	2021-01-20	LiSi	Secretary	151-0101-0000
3	2021-01-21	2021-01-21	WangWu	Associate Dean	177-2021-0301
4	2021-01-22	2021-01-22	ZhaoLiua	Professor	138-1920-3912
5	2021-01-23	2021-01-29	ZhaoLiub	Lecturer	138-1921-3912
6	2021-01-30	2021-01-31	ZhaoLiuc	Professor	138-1922-3912
7	2021-02-01	2021-02-08	ZhaoLiud	Lecturer	198-1920-3912
8	2021-02-16	2021-02-24	ZhaoLiuf	Lecturer	138-1929-3912
9	2021-02-25	2021-02-28	ZhaoLiug	Professor	138-1920-0000
10	2021-03-01	2021-03-01	ZhaoLiuH	Associate Professor	138-1929-0000
11	2021-03-05	2021-03-05	ZhaoLiuJ	AssociateProfessor	138-1920-0044
12	2021-03-06	2021-03-06	ZhaoLiuk	Professor	188-1920-0000

按enter以返回功能菜单：

文件 test7.txt 中的数据为合法数据，且排班还存在空白，结果如上。

其他测试文件我也进行了测试，情况如下：

test1.txt、test4.txt 合法，排班无空白；

test2.txt、test3.txt、test6.txt 格式错误；

test5.txt 存在时间段重叠；

test7.txt、test8.txt 合法，排班分别有 17.86%、14.29%空白。

注：对于员工信息重复或排班信息中对应员工不存在的情况，程序设计的处理策略是直接忽略这一项，故 test7.txt、test8.txt 能够正常读入。

## 3.8 应对面临的新变化

### 3.8.1 变化 1

之前的设计可以应对变化，且修改难度不是很大。预计要修改三个文件，包括 NoBlankIntervalSetImpl 类的部分实现、DutyIntervalSet 类部分方法实现、APP 输出信息对应方法实现。

这些变化是围绕 DutyIntervalSet<L>继承父类 CommonIntervalSet<L>替换为 MultiIntervalSet<L>而进行的，具体修改如下：

```

src > specialOps > NoBlankIntervalSetImpl.java
7- import baseADTs.CommonIntervalSet;
8 import helpers.dateHandler;
9
10 public class NoBlankIntervalSetImpl<L> implements NoBlankIntervalSet<L> {
11
12     private final long startTime;
13     private final long endTime;
14     private final CommonIntervalSet<L> cisl;
15
16     public NoBlankIntervalSetImpl(long startTime, long endTime, CommonIntervalSet<L> cisl) {
17         this.startTime = startTime;
18         this.endTime = endTime;
19         this.cisl = cisl;
20     }
21
22     @Override
23     public void checkNoBlank() {
24         if (cisl.labels().size() == 0) {
25             System.out.println("空白段1" + ":" + dateHandler.parseLong(startTime)
26                 + "-" + dateHandler.parseLong(endTime));
27             System.out.println("空白所占比例: 100% (未进行排班)");
28             return;
29         }
30         List<Long> timelist = new ArrayList<>();
31         for (L l : cisl.labels()) { // 保存所有的时间信息
32             timelist.add(cisl.start(l));
33             timelist.add(cisl.end(l));
34         }
35     }
36 }
7+ import baseADTs.IntervalSet;
8 import helpers.dateHandler;
9
10 public class NoBlankIntervalSetImpl<L> implements NoBlankIntervalSet<L> {
11
12     private final long startTime;
13     private final long endTime;
14     private final List<IntervalSet<L>> lisl;
15
16     public NoBlankIntervalSetImpl(long startTime, long endTime, List<IntervalSet<L>> lisl) {
17         this.startTime = startTime;
18         this.endTime = endTime;
19         this.lisl = lisl;
20     }
21
22     @Override
23     public void checkNoBlank() {
24         if (lisl.size() == 0) {
25             System.out.println("空白段1" + ":" + dateHandler.parseLong(startTime)
26                 + "-" + dateHandler.parseLong(endTime));
27             System.out.println("空白所占比例: 100% (未进行排班)");
28             return;
29         }
30         List<Long> timelist = new ArrayList<>();
31         for (IntervalSet<L> cisl : lisl) {
32             for (L l : cisl.labels()) { // 保存所有的时间信息
33                 timelist.add(cisl.start(l));
34                 timelist.add(cisl.end(l));
35             }
36         }
37     }
38 }

```

```

src > appADTs > DutyIntervalSet.java
1 package appADTs;
2
3- import baseADTs.CommonIntervalSet;
4 import specialOps.NoBlankIntervalSetImpl;
5 import specialOps.NonOverlapIntervalSetImpl;
6
7- public class DutyIntervalSet<> extends CommonIntervalSet<> implements IDuty
8
9 // fields:
10 private final NonOverlapIntervalSetImpl<> nois;
11 private final NoBlankIntervalSetImpl<> nbis;
12
13 // Abstraction function:
14 // 沿用CommonIntervalSet<>的AF
15 // 此外在nbis中保存了排班的起止日期
16 // 整体代表一个排班表
17
18 // Representation invariant:
19 // 沿用CommonIntervalSet<>的RI
20
21 // Safety from rep exposure:
22 // 大部分在CommonIntervalSet<>中保证
23 // 类的属性均设置为private final
24
25 // Constructor:
26 public DutyIntervalSet(long startTime, long endTime) {
27- super(); // 初始化CommonIntervalSet<>
28- commonIntervalSet<> pack = new CommonIntervalSet<>(this.labelSet, t
29- nois = new NonOverlapIntervalSetImpl<>(pack);
30- nbis = new NoBlankIntervalSetImpl<>(startTime, endTime, pack);
31 }

1 package appADTs;
2
3+ import baseADTs.IntervalSet;
4+ import baseADTs.MultiIntervalSet;
5 import specialOps.NoBlankIntervalSetImpl;
6 import specialOps.NonOverlapIntervalSetImpl;
7
8+ public class DutyIntervalSet<> extends MultiIntervalSet<> implements IDuty
9
10 // fields:
11 private NonOverlapIntervalSetImpl<> nois;
12 private final NoBlankIntervalSetImpl<> nbis;
13
14 // Abstraction function:
15 // 沿用MultiIntervalSet<>的AF
16 // 此外在nbis中保存了排班的起止日期
17 // 整体代表一个排班表
18
19 // Representation invariant:
20 // 沿用CommonIntervalSet<>的RI
21
22 // Safety from rep exposure:
23+ // 大部分在MultiIntervalSet<>中保证
24 // 类的属性均设置为private final
25
26 // Constructor:
27 public DutyIntervalSet(long startTime, long endTime) {
28- super(); // 初始化
29+ nois = new NonOverlapIntervalSetImpl<>(startTime, endTime, this.multitask
30 }

40- nois.insert(start, end, label);
41
39+ for (IntervalSet<> i : this.multitask) {
40+ nois = new NonOverlapIntervalSetImpl<>(i);
41+ nois.isOverlap(start, end, label);
42+ }
43+ for (IntervalSet<> i : multitask) {
44+ if (!i.labels().contains(label)) {
45+ // 当前集合中无此标签的时间段则插入
46+ i.insert(start, end, label);
47+ return;
48+ }
49+ }
50+ // 所有时间段集合中均有此标签的时间段
51+ // 需要创建一个新的时间段集合来加入
52+ IntervalSet<> toAdd = IntervalSet.empty();
53+ toAdd.insert(start, end, label);
54+ this.multitask.add(toAdd);

src > apps > DutyRosterApp.java
257- sortM.put(dise.start(e), e);
258- }
259- f.format("%-5s %-12s %-12s %-15s %-15s %-15s\n", "序号", "开始日期",
260- Integer cnt = 1;
261- for (Employee e : sortM.values()) {
262- f.format("%-5d %-15s %-15s %-18s %-15s\n", cnt, dateHandler
263- dateHandler.parseLong(dise.end(e)), e.getName(), e.getPos
264- ++cnt;
265- }
266- }

257+ for (Integer i : dise.intervals(e).labels())
258+ sortM.put(dise.intervals(e).start(i), e);
259+ }
260+ f.format("%-5s %-12s %-12s %-15s %-15s %-15s\n", "序号", "开始日期",
261+ Integer cnt = 1;
262+ Set<Employee> set = new HashSet<>();
263+ set.addAll(sortM.values());
264+ for (Employee e : set) {
265+ for (Integer i : dise.intervals(e).labels()) {
266+ f.format("%-5d %-15s %-15s %-18s %-15s\n", cnt, dateHan
267+ dateHandler.parseLong(dise.intervals(e).end(i)), e.g
268+ ++cnt;
269+ }
270+ }
271+ }

```

总计修改了大约 50 行，且所耗费时间不到半小时，不足的是部分破坏了原有的方法，但程序表现正常。如下图，修改后可以出现一个员工被安排多段值班的情况：

```

-----排班管理系统-----
请输入相对文件路径（如“txt/test1.txt”，输入“#”或文件读入错误会跳过文件读入，切换至手动操作）：
#
请输入排班开始日期（格式：yyyy-MM-dd）： 2021-03-01
请输入排班结束日期（格式：yyyy-MM-dd）： 2021-03-20
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 1
-----添加员工信息-----
请输入员工1姓名、职务和手机号（空格隔开，输入#结束）： test admin 123-4567-8910
添加成功！
请输入员工2姓名、职务和手机号（空格隔开，输入#结束）： #
按enter以返回功能菜单：

```

```

-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 4
-----添加排班记录-----
请按顺序输入开始时间、结束时间及员工姓名（空格隔开，时间采用yyyy-MM-dd格式，输入#结束）：
2021-03-01 2021-03-10 test
添加成功！
请按顺序输入开始时间、结束时间及员工姓名（空格隔开，时间采用yyyy-MM-dd格式，输入#结束）：
2021-03-11 2021-03-20 test
添加成功！
请按顺序输入开始时间、结束时间及员工姓名（空格隔开，时间采用yyyy-MM-dd格式，输入#结束）：
#
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 6

-----检查排班情况-----
当前排班已无空白！
按enter以返回功能菜单：

-----功能菜单-----
[1] 添加一组员工
[2] 删除指定员工
[3] 查看员工信息
[4] 手动添加排班记录
[5] 自动生成排班记录
[6] 检查排班情况
[7] 展示当前排班表
请选择（输入功能对应数字，其他退出）： 7
-----排班表-----
序号    开始日期    结束日期    姓名    职务    手机号码
1      2021-03-01    2021-03-10    test    admin    123-4567-8910
2      2021-03-11    2021-03-20    test    admin    123-4567-8910
按enter以返回功能菜单：

```

### 3.8.2 变化 2

之前的设计可以应对变化，且修改难度不是很大。预计要修改两个文件，包括 `ICourseIntervalSet` 接口的继承关系、`CourseIntervalSet` 类的私有属性（用于实现委托）和 `insert()` 方法实现，不需要修改 APP。具体修改如下：

（图见下页）



The first screenshot shows the `CourseIntervalSet` class in `appADTs.java`. It imports `baseADTs.IntervalSet`, `baseADTs.MultiIntervalSet`, and `specialOps.PeriodicIntervalSetImpl`. The class extends `MultiIntervalSet` and implements `ICourseIntervalSet`. It has a private final field `pis` of type `PeriodicIntervalSetImpl`. The second screenshot shows the `insert` method in `CourseIntervalSet`, which calls `this.pis.insert(start, end, label)`. The third screenshot shows the `ICourseIntervalSet` interface in `ICourseIntervalSet.java`, which extends `PeriodicIntervalSet` and has a method `insert`.

总计修改了大约 10 行，耗费时间几乎可以不计。如下图，程序现在可以实现课表的无重叠（插入时报错并返回）。

```

-----课表管理系统-----
请输入学期开始日期（格式：yyyy-MM-dd）： 2021-03-01
请输入学期总周数（例如：18）： 18
-----功能菜单-----
[1] 添加一组课程
[2] 安排课程时间
[3] 查看安排情况
[4] 单日课表查询
请选择（输入功能对应数字，其他退出）： 1
-----添加课程信息-----
请按顺序输入课程ID、课程名称、教师姓名、地点和周学时数（空格隔开，输入#结束）：
1234 testa teachera abcde 4
添加成功！
请按顺序输入课程ID、课程名称、教师姓名、地点和周学时数（空格隔开，输入#结束）：
1235 testb teacherb bcdef 4
添加成功！
请按顺序输入课程ID、课程名称、教师姓名、地点和周学时数（空格隔开，输入#结束）：
#
按enter以返回功能菜单：

-----功能菜单-----
[1] 添加一组课程
[2] 安排课程时间
[3] 查看安排情况
[4] 单日课表查询
请选择（输入功能对应数字，其他退出）： 2
-----安排课程时间-----
请输入要进行安排课程的ID： 1234
请输入安排的上课时间（如周四，8-10时应输入“4 8-10”）： 2 8-10
安排成功！
按enter以返回功能菜单：

```

```

-----功能菜单-----
[1] 添加一组课程
[2] 安排课程时间
[3] 查看安排情况
[4] 单日课表查询
请选择（输入功能对应数字，其他退出）： 2
-----安排课程时间-----
请输入要进行安排课程的ID： 1235
请输入安排的上课时间（如周四，8-10时应输入“4 8-10”）： 2 8-10
安排失败：与[1234          testa          teachera          abcde  4]存在时间段重叠！
按enter以返回功能菜单：
-----功能菜单-----
[1] 添加一组课程
[2] 安排课程时间
[3] 查看安排情况
[4] 单日课表查询
请选择（输入功能对应数字，其他退出）： 3
-----查看安排情况-----
课程ID      课程名称      教师姓名      地点      剩余/总周学时      安排情况
1235        testb         teacherb      bcdef      4 / 4              待安排
1234        testa         teachera      abcde      2 / 4              待安排
空闲时间比例：96.98%
重复时间比例：0.00%
按enter以返回功能菜单：

-----功能菜单-----
[1] 添加一组课程
[2] 安排课程时间
[3] 查看安排情况
[4] 单日课表查询
请选择（输入功能对应数字，其他退出）： 4
-----单日课表查询-----
请输入要查询的日期（格式：yyyy-MM-dd）： 2021-03-02
2021-03-02 （周二） 课程表
---08:00 - 10:00---
[1234] testa teachera@abcde
---10:00 - 12:00---
---13:00 - 15:00---
---15:00 - 17:00---
---19:00 - 21:00---
按enter以返回功能菜单：

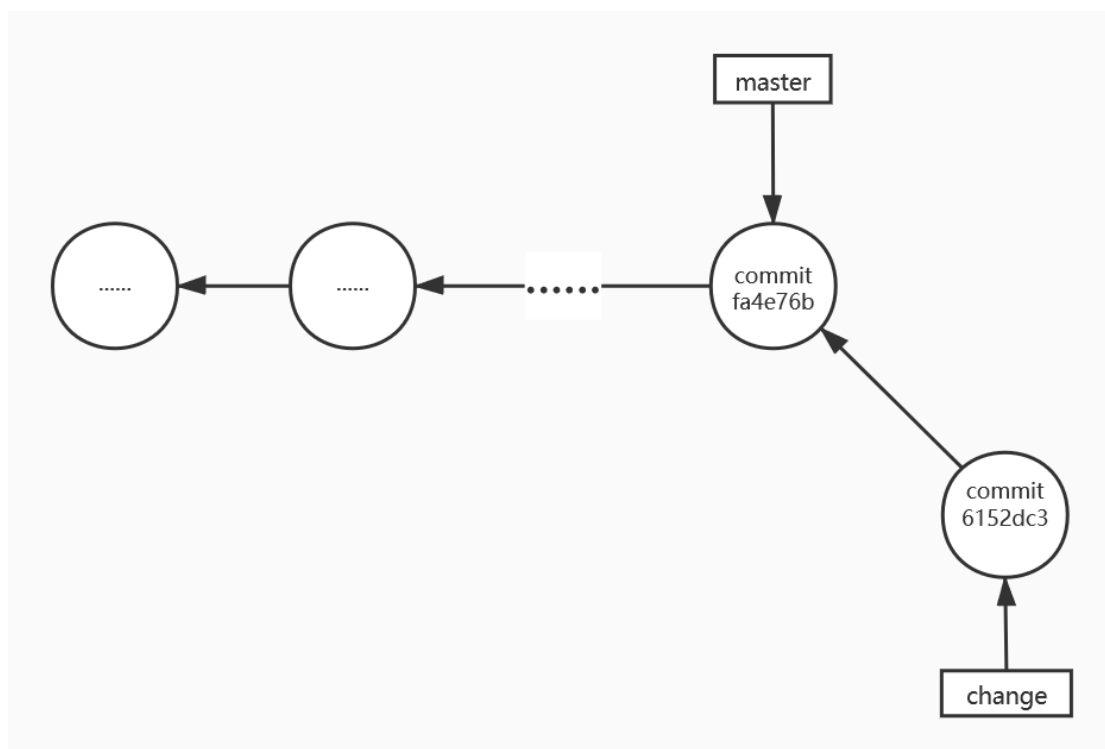
```

### 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。

使用 `git log` 指令分别获取了两个分支的 `commit` 信息，依据信息画出了 `Git` 仓库结构示意图。

（图见下页）



说明：**master** 分支当前最新 commit 为 **fa4e76b** “Some adjustments.”

**change** 分支当前最新 commit 为 **6152dc3** “change”

由于需再次提交才能将本部分内容更新到库中，最终的 Object Graph 应该与此不同，**master** 分支应该会指向更后面的 commit。

## 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。

日期	时间段	计划任务	实际完成情况
2021.6.14	13:20 - 14:00	完成接口 <code>IntervalSet&lt;L&gt;</code> 的设计	按计划完成
2021.6.16	19:40 - 21:00	编写测试并实现 <code>CommonIntervalSet&lt;L&gt;</code>	按计划完成
2021.6.17	15:00 - 15:30	初步设计 <code>MultiIntervalSet&lt;L&gt;</code>	按计划完成
2021.6.17	15:50 - 17:15	完成 <code>MultiIntervalSet&lt;L&gt;</code> 的设计与实现	按计划完成
2021.6.17	17:50 - 21:30	编写 <code>MultiIntervalSet&lt;L&gt;</code> 的测试，确定并初步构思设计方案五	按计划完成
2021.6.18	14:00 - 17:00	调整项目结构，编写标签 ADT	按计划完成
2021.6.21	14:00 - 17:00	编写方案五的前两个维度	基本完成
2021.6.23	14:30 - 17:30	实现全部三个维度	按计划完成
2021.6.26	10:00 - 11:30	实现 <code>DutyIntervalSet&lt;L&gt;</code>	遇到问题
2021.6.26	13:00 - 16:30	实现其余两个应用 ADT	按计划完成
2021.6.27	14:30 - 17:30	测试、完善两个应用 ADT	按计划完成
2021.6.28	9:00 - 11:00	设计可复用 API 的前半部分	按计划完成
2021.6.28	13:00 - 16:00	完成可复用 API 的设计并测试	按计划完成



2021.6.28	19:00 - 22:00	设计 DutyRosterApp	基本完成
2021.6.29	9:00 - 11:30	完善 DutyRosterApp, 初步设计 ProcessScheduleApp	按计划完成
2021.6.29	13:00 - 17:00	完善 ProcessScheduleApp 的设计, 初步设计 CourseScheduleApp	按计划完成
2021.6.29	18:30 - 21:30	完成三个 APP 的设计	按计划完成
2021.6.30	9:00 - 11:00	初步实现基于语法的数据读入	遇到问题
2021.6.30	15:30 - 17:30	完成基于语法的数据读入	按计划完成
2021.6.30	18:30 - 21:00	测试、完善各部分内容, 调整可视化格式	按计划完成
2021.7.1	16:00 - 17:15	初步设计变化	按计划完成
2021.7.1	18:00 - 22:40	完成实验全部内容, 完善实验报告, 进行在线测试, 提交	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 CRP 原则的具体实现不熟悉, 不了解如何进行组合、委托和复用。	复习课件上的相关内容, 结合实验指导书进行尝试, 与其他同学交流, 最终理解了相关概念和实现方法, 完成实验要求。
对 Java 中日期的相关的处理不太熟悉。	在网上查找相关教程并结合应用要求自行编写尝试, 最终形成了可应用于本次实验的帮助类 dateHandler (可查看 src/helpers/dateHandler.java)。
对 Java 中正则表达式的书写规则、具体应用了解不够充分。	复习了课件上的相关内容, 阅读了网上的相关教程, 最终顺利完成了基于语法的读入。
基于语法的读入出现问题。	通过与同学交流得知, 文件中空白字符并非 UTF-8 编码, 正则表达式中的 “\s*” 无法匹配, 最后修改了文件中的空白符号, 问题解决。

## 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教训

1. 认识到开发软件时复用 ADT 的巨大优势: 减少重复工作量、更加适应变化等;
2. 更加深刻地理解了课上所学的继承、委托、重写等概念, 复习了相关知识点;
3. 学会使用 CRP 原则设计简单的具有一定可复用性和可维护性的软件;
4. 学会使用正则表达式解析输入、处理数据;
5. 初次尝试结合使用接口、抽象类、类进行软件开发, 深化了对 Java 项目结构的理解。

## 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 的编程更抽象、更通用，而直接面向应用场景编程更具体，同时也需要考虑应用场景的共同点，减少重复性不必要的工作；复用在这次实验中帮助我减少了许多工作量，提高了整体效率。

- (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：意义是保证程序的正确性、健壮性、安全性和可维护性，提高程序质量，避免编程的实际工作脱离目标，且便于用户使用。我愿意在以后编程中坚持这样做。

- (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：API 的开发需同时考虑到通用性和具体的功能实现，对我来说有一定难度。但最终完成开发，通过自己编写的测试用例并应用到 APP 中也是很有成就感的。

- (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：语法驱动编程对输入有着较为严格的要求，但也方便了程序的读入和处理。通过语法驱动编程进行开发能使用户与软件的交互更加直接。

- (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：难度主要体现在 ADT 之间的继承、实现、复用等关系上，开发单个 ADT 并不是很难，但为了实现方案五，我耗费了大量的时间来调整 ADT 之间的关系，特别是面向 APP 的 ADT 与特殊维度操作类之间的委托关系。我克服这些困难的方法主要是不断尝试和吸取错误教训。

- (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。本实验的三个应用既不能完全抽象为同一个 ADT，也不是完全个性化，如何利用“接口、抽象类、类”三层体系以及接口的组合、类的继承、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：①注意寻找应用共同点，区分应用不同点；②从多个维度出发，从抽象到具体；③利用好三层体系之间的关系以及各种设计模式等才能最大程度上实现复用，减少工作量；④不能仅关注局部特性，过于依赖于特定实现。

(7) 关于本实验的工作量、难度、deadline。

答：本次实验工作量很大，难度也比较大，如果考虑即将到来的期末考试并为复习预留时间，deadline 也比较紧，需要兼顾速度和质量。

(8) 到目前为止你对《软件构造》课程的评价。

答：通过软件构造课程学到了软件设计、开发和 Java 编程的很多相关知识、方法和技能。我认为这门课有很强的实际意义，对未来的职业发展也十分重要。此外也建议调整课程内容安排，让同学们有更充足的时间进行实验、预习和复习。