



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	魏志豪
学号	1190302020
班号	1903009
电子邮件	<a href="mailto:1207410841@qq.com">1207410841@qq.com</a>
手机号码	15209945766

## 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）	4
3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L>	6
3.3.1 MultiIntervalSet<L>的共性操作	6
3.3.2 局部共性特征的设计方案	8
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）	8
3.4 面向复用的设计：L	10
3.5 可复用 API 设计	11
3.5.1 计算相似度	12
3.5.2 计算时间冲突比例	13
3.5.3 计算空闲时间比例	13
3.6 应用设计与开发	13
3.6.1 排班管理系统	13
3.6.2 操作系统的进程调度管理系统	16
3.6.3 课表管理系统	18
3.7 基于语法的数据读入	20
3.8 应对面临的新变化	22
3.8.1 变化 1	22
3.8.2 变化 2	23
3.9 Git 仓库结构	23
4 实验进度记录	23
5 实验过程中遇到的困难与解决途径	23
6 实验过程中收获的经验、教训、感想	24

6.1 实验过程中收获的经验教训 .....	24
6.2 针对以下方面的感受 .....	24

## 1 实验目标概述

本次实验覆盖课程第 4-11 讲的内容，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：

- 1 子类型、泛型、多态、重写、重载
- 2 继承、代理、组合
- 3 语法驱动的编程、正则表达式
- 4 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

实验环境已在 lab1 和 lab2 中配置完成。

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

<https://github.com/ComputerScienceHIT/HIT-Lab3-1190302020>

## 3 实验过程

### 3.1 待开发的三个应用场景

#### 1 值班表管理

该应用创建一个可以维护的值班表，能够向其中添加、删除员工，并能手动或自动的进行排班，还可以实时的查看员工的排班情况。

#### 2 操作系统进程调度管理

该应用模拟单独的 CPU 运行时的进程管理和控制流的切换，在进程执行到最大执行时间之前，自主（这里有两种策略，一种是随机选择进程策略，另一种是最短进程优先策略）地选择新的进程来执行，在所有进程都结束后可以查看进

程运行的轨迹。

### 3 大学课表管理

模拟大学课表的设置与管理，可以在同一时段设置多节课，只需设置第一周的课程，后续几周的课程就会周期性的添加，在课表安排完成后还可以查看与课程安排的相关信息。

### 4 相同之处：

这三个应用场景的基本功能都是在时间轴上进行的，用户通过在时间轴上插入元素来达到使用应用的目的。

### 5 不同之处：

在值班表管理应用中，每个对象在时间轴上占用的时间必须是连续的且不能够有重叠和空隙；在操作系统进程调度管理应用中，每个对象在时间轴上占用的时间可以是不连续的且可以有空隙，但不能有重叠；在大学课表管理应用中，每个对象在时间轴上可以不连续，可以重叠，可以有空隙，但必须是周期性出现的。

## 3.2 面向可复用性和可维护性的设计：IntervalSet<L>

### 3.2.1 IntervalSet<L>的共性操作

```
/**
 * 给label分配一段时间，且label不能重复
 * @param start 起始时间
 * @param end 结束时间
 * @param label 标签
 */
public void insert(long start, long end, L label);

/**
 * 返回所有标签的集合
 * @return
 */
public Set<L> labels();

/**
 * 移除label及其对应的时间段
 * @param label 标签
 * @return 若label存在返回true；若label不存在返回false
```

```
*/  
public boolean remove(L label);  
  
/**  
 * 得到label对应的起始时间  
 * @param label 标签  
 * @return 若label存在返回label的起始时间；若label不存在返回-1  
 */  
public long start(L label);  
  
/**  
 * 得到label对应的结束时间  
 * @param label 标签  
 * @return 若label存在返回label的结束时间；若label不存在返回-1  
 */  
public long end(L label);  
  
/**  
 * 得到最小时间  
 * @return  
 */  
public long getmin();  
  
/**  
 * 得到最大时间  
 * @return  
 */  
public long getmax();  
  
/**  
 * 返回一个包含当前时间的label的集合  
 * @param currenttime  
 * @return  
 */  
public Set<L> containtime(long currenttime);  
  
/**  
 * 返回一个按起始时间从小到大顺序排序的label列表  
 * @return  
 */  
public List<MyMap<L>> arrangestart();  
  
/**  
 * 删除所有元素
```

```
*/  
public void removeall();
```

### 3.2.2 局部共性特征的设计方案

由于 `IntervalSet<L>` 是不容许同一元素多次出现的分隔集，所以每个标签都对应有唯一的时间段，所以就可以得到对应标签的开始时间和结束时间，又因为 `IntervalSet` 是最基础的插入的接口，所以它应该有前面所说的三个应用的共性特点，不能带有特殊性（即对元素的插入不做要求，除了前面所说的同一个元素只能出现一次）。

### 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）

我使用的是实验指导书中的第 6 种设计方式：decorator 设计模式，具体设计方案如下：

1 使用类 `CommonIntervalSet` 实现 `IntervalSet` 接口，作为 `IntervalSet` 的基础类，来实现最共性的操作。`CommIntervalSet` 里的所有方法如下图所示：

```
• G CommonIntervalSet<L>  
  ▫ labellist : List<L>  
  ▫ startlist : List<Long>  
  ▫ endlist : List<Long>  
  ▫ checkRep() : void  
  ●▲ insert(long, long, L) : void  
  ●▲ labels() : Set<L>  
  ●▲ remove(L) : boolean  
  ●▲ start(L) : long  
  ●▲ end(L) : long  
  ●▲ getmin() : long  
  ●▲ getmax() : long  
  ●▲ containtime(long) : Set<L>  
  ●▲ arrangestart() : List<MyMap<L>>  
  ●▲ toString() : String  
  ●▲ removeall() : void
```

`MyMap` 是我为了方便自己定义的一个 ADT，它有着与 `Map` 类似的功能，它是一个 `label` 对应两个时间 `start` 和 `end`，并且它里面所有的元素都是公有的可以直接访问和使用。

```
public class MyMap<L>{
    public L label;
    public Long start;
    public Long end;

    /**
     * 判断两个冲突时间段是否一样
     * @param m
     * @return
     */
    public boolean conflicttimeequal(MyMap<L> m) {
        if(start==m.start&&end==m.end) {
            return true;
        }
        else {
            return false;
        }
    }
}
```

2 使用抽象类 IntervalSetDecorator 作为 Decorator 的基础，利用委托机制使用一个 IntervalSet（其实就是 CommonIntervalSet）的方法来实现接口 IntervalSet 里的所有方法。

```
public abstract class IntervalSetDecorator<L> implements IntervalSet<L>{
    protected final IntervalSet<L> set;

    public IntervalSetDecorator(IntervalSet<L> set) {
        this.set=set;
    }

    public void insert(long start,long end,L label) {
        set.insert(start, end, label);
    }

    public Set<L> labels(){
        return set.labels();
    }








    public boolean remove(L label) {
        return set.remove(label);
    }

    public long start(L label) {
        return set.start(label);
    }

    public long end(L label) {
        return set.end(label);
    }
}
```

3 如下图所示，里面的除 IntervalSetDecorator 的其他类都是 IntervalSetDecorator 的子类，它们继承抽象类 IntervalSetDecorator 里的方法，并在其基础上实现某种特有的功能。



- ▼  Decorator
  - >  IntervalSetDecorator.java
  - >  MultiIntervalSetDecorator.java
  - >  NoBlankIntervalSet.java
  - >  NonOverlapIntervalSet.java
  - >  NonOverlapMultiIntervalSet.java
  - >  PeriodicMultiIntervalSet.java

例如类 `NonOverlapIntervalSet`，它在基础方法上又添加了插入时不能重叠的特性，若发生重叠会报错并插入失败，这就是一层装饰，在使用时只用通过构造函数一层层的包装即可实现想要的特性

```
public class NonOverlapIntervalSet<L> extends IntervalSetDecorator<L> {
    public NonOverlapIntervalSet(IntervalSet<L> set) {
        super(set);
    }

    /**
     * 给label分配一段时间，若时间段发生重叠则报错
     * @param start 起始时间
     * @param end 结束时间
     * @param label 标签
     */
    public void insert(long start, long end, L label) {
        boolean flag = true;
        for (L temp : set.labels()) {
            if (!(end <= set.start(temp) || start >= set.end(temp))) {
                flag = false;
                System.out.println("发生重叠");
            }
        }
        if (flag) {
            super.insert(start, end, label);
        }
    }
}
```

### 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

```
/**
 * 给label分配一段时间
 * @param start 起始时间
 * @param end 结束时间
 * @param label 标签
 */
public void insert(long start, long end, L label);

/**
 * 返回所有标签的集合
 * @return
```

```
*/  
public Set<L> labels();  
  
/**  
 * 移除label及其所有的时间段  
 * @param label 标签  
 * @return 若label存在返回true; 若label不存在返回false  
 */  
public boolean remove(L label);  
  
/**  
 * 得到label对应的所有时间段  
 * @param label 标签  
 * @return 时间段从小到大排列  
 */  
public IntervalSet<Integer> intervals(L label);  
  
/**  
 * 得到最小时间  
 * @return  
 */  
public long getmin();  
  
/**  
 * 得到最大时间  
 * @return  
 */  
public long getmax();  
  
/**  
 * 返回一个包含当前时间的label的集合  
 * @param currenttime  
 * @return  
 */  
public Set<L> containtime(long currenttime);  
  
/**  
 * 返回一个按起始时间从小到大顺序排序的label列表,同一个label可能重复出现  
 * @return  
 */  
public List<MyMap<L>> arrangestart();  
  
/**  
 * 删除所有元素
```

```
*/  
public void removeAll();
```

### 3.3.2 局部共性特征的设计方案

MultiIntervalSet 容许同一个元素多次出现，但是同一个元素的时间段不能发生重叠，若发生重叠会将两段时间和为一段。这个接口也代表该种类型的最共性的操作，所以在插入的其他方面并不做限制。

### 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）

MultiIntervalSet 的子类型的设计和 IntervalSet 的子类型的设计方式一模一样，只是在具体实现时有些许差别。具体设计方案如下：

1 使用 CommonMultiIntervalSet 类实现 MultiIntervalSet 的最基础的操作，其具体内容如下：

```
▼ CommonMultiIntervalSet<L>  
  ▢ IntervalSetlist : List<IntervalSet<L>>  
  ● ▲ insert(long, long, L) : void  
  ● ▲ labels() : Set<L>  
  ● ▲ remove(L) : boolean  
  ● ▲ intervals(L) : IntervalSet<Integer>  
  ● ▲ getmin() : long  
  ● ▲ getmax() : long  
  ● ▲ containtime(long) : Set<L>  
  ● ▲ arrangestart() : List<MyMap<L>>  
  ● ▲ toString() : String  
  ● ▲ removeAll() : void
```

在这里面使用了之前实现的 IntervalSet 做 rep，并在方法的实现过程中使用了 IntervalSet 中的方法，例如 insert 实现时，首先遍历已经存在的 IntervalSet，寻找是否有某个 IntervalSet 里没有该 label 标签对应的元素，若有这样的 IntervalSet 则，将 label 插入到找到的 IntervalSet；若找不到这样的 IntervalSet，则新建一个 IntervalSet 将 label 插入其中，并将新建的 IntervalSet 添加到 IntervalSetlist 当中。Insert 的具体实现如下：

```

@Override public void insert(long start,long end,L label) {
    int i=0;
    boolean isoverlap=false;
    boolean flag=true;
    for(int j=0;j<IntervalSetlist.size();j++) {
        IntervalSet<L>current=IntervalSetlist.get(i);
        if(current.labels().contains(label)) {
            if(!(end<=current.start(label)||start>=current.end(label))) {
                isoverlap=true;
                System.out.println("当前label已在该时间段存在, 请勿重复插入");
            }
        }
    }
    if(!isoverlap) {
        for(;i<IntervalSetlist.size();i++) {
            if(!IntervalSetlist.get(i).labels().contains(label)) {
                flag=false;
                break;
            }
        }
        if(flag==false) {
            IntervalSetlist.get(i).insert(start, end, label);
        }
        else {
            IntervalSet<L> temp=new CommonIntervalSet<>();
            temp.insert(start, end, label);
            IntervalSetlist.add(temp);
        }
    }
}
}

```








## 2 使用抽象类 MultiIntervalSetDecorator 作为所有装饰类的基础。

```

1 public abstract class MultiIntervalSetDecorator<L>implements MultiIntervalSet<L>{
2     protected final MultiIntervalSet<L> set;
3
4     public MultiIntervalSetDecorator(MultiIntervalSet<L> set) {
5         this.set=set;
6     }
7
8     public void insert(long start,long end,L label) {
9         set.insert(start, end, label);
10    }
11
12    public Set<L> labels(){
13        return set.labels();
14    }
15
16    public boolean remove(L label) {
17        return set.remove(label);
18    }
19
20    public IntervalSet<Integer> intervals(L label){
21        return set.intervals(label);
22    }
23
24 }

```










## 3 根据功能要求设计具体的装饰类。具体方法与 IntervalSet 的装饰类的设计方法类似。

- ▼  Decorator
  - >  IntervalSetDecorator.java
  - >  MultiIntervalSetDecorator.java
  - >  NoBlankIntervalSet.java
  - >  NonOverlapIntervalSet.java
  - >  NonOverlapMultiIntervalSet.java
  - >  PeriodicMultiIntervalSet.java













### 3.4 面向复用的设计：L

L 是泛型，它可以被替换为其他的 ADT，例如后面应用中用到的员工类 Employee、进程类 Process 和课程类 Course。当然在替换的时候要保证应用中设计的那几个类是不可变的数据类型。

Empolyee 的具体实现：

- ▼  Employee
  - ▢  name : String
  - ▢  position : String
  - ▢  phone : String
  -  Employee(String, String, String)
  -  getname() : String
  -  getposition() : String
  -  getphone() : String
  -  toString() : String

Process 的具体实现：

- ▼  Process
  - ▢  pid : long
  - ▢  mintime : long
  - ▢  maxtime : long
  - ▢  name : String
  -  Process(long, String, long, long)
  -  getpid() : long
  -  getmintime() : long
  -  getmaxtime() : long
  -  getname() : String
  -  equal(Process) : boolean
  -  toString() : String

Course 的具体实现：

```

Course
  F courseid : long
  F coursename : String
  F teachername : String
  F position : String
  F weeklyhours : long
  C Course(long, String, String, String, long)
  C getcourseid() : long
  C getcoursename() : String
  C getteachername() : String
  C getposition() : String
  C getweeklyhours() : long
  C equal(Course) : boolean
  C toString() : String

```

除了上述的几种 ADT, 我还设计了一些 CommonADT, 像之前所说的 MyMap, 还有 Date, Date 是一个表示时间的 ADT, 它有着计算两个日期之差, 增加天数, 转换成字符串等许多操作, 该 ADT 在应用 1 和应用 3 中都会频繁的用到, 它具体的构造如下图所示:

```

Date
  F calendar : Calendar
  C Date()
  C Date(Calendar)
  F getdayofmonth(int, int) : int
  C betweenday(Date) : long
  C betweenhours(Date) : long
  C setdate(int, int, int) : boolean
  C setdate(int, int, int, int) : boolean
  C getdata() : Calendar
  C addday(long) : void
  C addhours(long) : void
  C getdayofweek() : int
  C gethourString() : String
  C toString() : String

```

### 3.5 可复用 API 设计

API 中除了要求实现的那几个功能还有一些为实现要求功能而定义的私有方法, 它们都是为实现要求功能而服务的。API 的具体构造如下图所示:

### ▼ APIs<L>

- isconflict(MyMap<L>, MyMap<L>) : int
- computeconflicttime(MyMap<L>, MyMap<L>, List<MyMap<L>>) : double
- gettotaltime(MultiIntervalSet<L>, MultiIntervalSet<L>) : double
- getminindex(List<MyMap<L>>, Set<Integer>) : int
- mergetimearray(List<MyMap<L>>) : List<MyMap<L>>
- getfreetime(List<MyMap<L>>, long, long) : List<MyMap<L>>
- Similarity(MultiIntervalSet<L>, MultiIntervalSet<L>) : double
- calcConflictRatio(IntervalSet<L>, long) : double
- calcConflictRatio(MultiIntervalSet<L>, long) : double
- calcFreeTimeRatio(IntervalSet<L>, long) : double
- calcFreeTimeRatio(MultiIntervalSet<L>, long) : double

### 3.5.1 计算相似度

```
public double Similarity(MultiIntervalSet<L> s1, MultiIntervalSet<L> s2) {
    List<MyMap<L>> arrays1=s1.arrangestart();
    List<MyMap<L>> arrays2=s2.arrangestart();
    double same=0;
    double totaltime=gettotaltime(s1,s2);
    for(int i=0;i<arrays1.size();i++) {
        MyMap<L> m1=arrays1.get(i);
        for(int j=0;j<arrays2.size();j++) {
            MyMap<L> m2=arrays2.get(j);
            if(m1.label.equals(m2.label)) {
                same+=computeconflicttime(m1,m2,null);
            }
        }
    }
    return same/totaltime;
}
```

arrays1、arrays2 是通过 MultiIntervalSet 里 arrangestart 方法得到的开始时间从小到大的 list，并以 MyMap 为元素，totaltime 是两个时间段的总时间，之后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。

### 3.5.2 计算时间冲突比例

```

public double calcConflictRatio(MultiIntervalSet<L> set, long end) {
    List<MyMap<L>> array=set.arrangestart();
    List<MyMap<L>> conflictarray=new ArrayList<>();
    double same=0;
    double totaltime=end;
    for(int i=0;i<array.size();i++) {
        MyMap<L>m1=array.get(i);
        for(int j=i+1;j<array.size();j++) {
            MyMap<L>m2=array.get(j);
            computeconflicttime(m1,m2,conflictarray);
        }
    }
    List<MyMap<L>> mergeconflictarray=mergetimearray(conflictarray);
    for(int i=0;i<mergeconflictarray.size();i++) {
        same+=mergeconflictarray.get(i).end-mergeconflictarray.get(i).start;
    }
    System.out.println(same/totaltime);
    return same/totaltime;
}

```

end 代表截止时间，即在 0 到 end 这一时间段上计算时间冲突比例，mergetimearray 顾名思义，就是将时间轴上的重合的时间合并起来，防止多次重复计算重合的时间段，最后用 same/totaltime 即可得到冲突时间在给定时间段上的比例。

### 3.5.3 计算空闲时间比例

```

public double calcFreeTimeRatio(MultiIntervalSet<L> set, long end) {
    List<MyMap<L>> array=set.arrangestart();
    List<MyMap<L>> fullarray=mergetimearray(array);
    double totaltime=end;
    double fulltime=0;
    for(int i=0;i<fullarray.size();i++) {
        fulltime+=fullarray.get(i).end-fullarray.get(i).start;
    }
    return (totaltime-fulltime)/totaltime;
}

```

end 代表截止时间，即在 0 到 end 这一时间段上计算空闲时间的比例，使用 mergetimearray 将时间轴上使用的时间段合并起来，之后先计算使用的时间再用总时间减去使用的时间即可得到空闲的时间，最后除 totaltime 即可得到空闲时间在给定时间段上的比例。

## 3.6 应用设计与开发

### 3.6.1 排班管理系统

排班管理系统的具体实现如下图所示：



```

v DutyRosterApp
  startdate : Date
  enddate : Date
  employeelist : List<Employee>
  set : NoBlankIntervalSet<Employee>
  clear() : void
  getIntervalSet() : NoBlankIntervalSet<Employee>
  addfileemployee(String) : void
  setfiledate(String) : void
  insertfile(String) : void
  ReadFile(File) : boolean
  setstartdate(int, int, int) : boolean
  printstartdate() : void
  printenddate() : void
  setenddate(int, int, int) : boolean
  checkdate() : boolean
  addemployee(String, String, String) : void
  deleteemployee(String) : boolean
  getemployeeindex(String) : int
  insert(String, int, int, int, int, int, int) : void
  autoinsert() : void
  timetoDate(long) : Date
  getlength() : long
  printDutyRoster() : void
  printEmployeeelist() : void
  main(String[]) : void

```

其中 main 是客户端程序，用户运行客户端之后通过输入数字来选择自己要使用的功能，其中使用的分隔集的包装如下图所示，根据要求我们可以指导排班表里员工的值班时间必须是连续的，所以我们使用 `CommonIntervalSet` 作为基础类，之后又知道排班时间是不能够重合的，即一天只有一人值班，所以我又用 `NonOverlapIntervalSet` 来装饰 `CommonIntervalSet`，最后由于每天都得有人值班，是不能有空闲时间段的，所以我又使用 `NoBlankIntervalSet` 来包装 `NonOverlapIntervalSet`，这样通过两层装饰的添加，就实现了排班表所需要的功能特性。

```

private Date startdate=new Date();
private Date enddate=new Date();
private List<Employee> employeelist=new ArrayList<>();
private NoBlankIntervalSet<Employee> set= new NoBlankIntervalSet<>(new NonOverlapIntervalSet<>(new CommonIntervalSet<>()));

```

之后具体功能的实现就调用我们包装好的分隔集 `set` 里的方法就很容易实现了。

运行时的图片：

输入2开始设置结束日期

2

请输入年：

2000

请输入月：

1

请输入日：

3

开始日期:2000/1/1

结束日期(不包含该天):2000/1/3

3 添加员工

4 删除员工

5 指定员工排班

6 打印排班表

7 检查是否排满

8 自动生成排班表

9 打印员工表

0 退出

请输入3~9选择功能：

(在这里我手动添加了两个员工，并使用自动排班了)

开始日期:2000/1/1

结束日期(不包含该天):2000/1/3

3 添加员工

4 删除员工

5 指定员工排班

6 打印排班表

7 检查是否排满

8 自动生成排班表

9 打印员工表

0 退出

请输入3~9选择功能：

6

开始日期	结束日期	姓名	职位	电话
2000/1/1	2000/1/2	王五	厨师	10000
2000/1/2	2000/1/3	张三	服务员	20000

开始日期:2000/1/1

结束日期(不包含该天):2000/1/3

3 添加员工

4 删除员工

5 指定员工排班

6 打印排班表

7 检查是否排满

8 自动生成排班表

9 打印员工表

0 退出

请输入3~9选择功能：

9

姓名	职位	电话	是否排班
王五	厨师	10000	已排班
张三	服务员	20000	已排班

```

开始日期:2000/1/1
结束日期(不包含该天):2000/1/3
3  添加员工
4  删除员工
5  指定员工排班
6  打印排班表
7  检查是否排满
8  自动生成排班表
9  打印员工表
0  退出
请输入3~9选择功能:
7
已排满

```

### 3.6.2 操作系统的进程调度管理系统

操作系统的具体实现如下图所示：

```

v ProcessScheduleApp
  processlist : List<ProcessContext>
  set : NonOverlapMultiIntervalSet<Process>
  runtime : long
  addprocess(long, String, long, long) : boolean
  runprocessrandom() : void
  getminremainprocessindex(Set<Integer>) : int
  runminprocess() : void
  print() : void
  getprocess(long) : int
  printSelectpid(long) : void
  clearrun() : void
  main(String[]) : void

```

为了更方便的实现该应用我又定义了一个 `ProcessContext` 的 ADT 用来保存进程的上下文信息，它主要记录着进程运行的时间和进程的状态，它在操作系统选择下一时刻执行的进程中起着重要的作用，具体实现如下图所示：

```

ProcessContext
  p : Process
  status : boolean
  runtime : long
  ProcessContext(Process)
  run(long) : boolean
  isfinished() : boolean
  maxremaintime() : long
  getprocess() : Process
  clear() : void

```

根据要求我们知道进程是可以多次执行的，所以我选用 `CommonMultiIntervalSet` 作为最基础的实现类，又由于操作系统可以挂机，在一段时间内不进行进程，所以我们并不需要用 `NoBlankIntervalSet` 来进行装饰，但因为模拟的是单核 CPU 不能在同一时间运行多个进程，所以又需要用 `NonOverlapIntervalSet` 来装饰

CommonIntervalSet。具体装饰如下图所示：

```
private List<ProcessContext> processlist=new ArrayList<>();
private NonOverlapMultiIntervalSet<Process> set=new NonOverlapMultiIntervalSet<>(new CommonMultiIntervalSet<>());
private long runtime=0;
```

运行时的图片：

```
1 添加进程
2 随机选择进程模拟运行
3 最短进程优先模拟运行
4 查看选定进程的运行轨迹
0 退出

1 添加进程
2 随机选择进程模拟运行
3 最短进程优先模拟运行
4 查看选定进程的运行轨迹
0 退出
1
请输入进程的pid:
1
请输入进程的名称:
qq
请输入进程的最小执行时间:
20
请输入进程的最大执行时间:
30
添加成功
（在这里我还添加了个微信）

1 添加进程
2 随机选择进程模拟运行
3 最短进程优先模拟运行
4 查看选定进程的运行轨迹
0 退出
2
时间：3-15      进程ID:3 进程名称:微信
时间：20-39     进程ID:3 进程名称:微信
时间：51-53     进程ID:1 进程名称:qq
时间：66-90     进程ID:1 进程名称:qq
时间：139-147   进程ID:3 进程名称:微信
时间：165-167   进程ID:3 进程名称:微信
模拟完成

1 添加进程
2 随机选择进程模拟运行
3 最短进程优先模拟运行
4 查看选定进程的运行轨迹
0 退出
4
请输入进程的pid:
1
进程ID:1 进程名称:qq      进程最小执行时间：20      进程最小执行时间：30
当前进程运行时段：
51-53
66-90
```

### 3.6.3 课表管理系统

课表管理系统的具体实现如下图所示：

```
✓ CourseScheduleApp
  ▫ startdate : Date
  ▫ weeks : long
  ▫ courselist : List<Course>
  ▫ set : PeriodicMultiIntervalSet<Course>
  ● setdate(int, int, int) : boolean
  ● getstartdate() : Date
  ● getweeks() : long
  ● getMultiIntervalSet() : PeriodicMultiIntervalSet<Course>
  ● setweeks(long) : boolean
  ● addcourse(long, String, String, String, long) : void
  ● getcourse(long) : int
  ● arrangeCourse(long, int, int) : void
  ● longtoDate(long) : Date
  ● print() : void
  ● countweeklyhours(Course) : int
  ● printallcourse() : void
  ● getenddate() : Date
  ● printallcourseofday(int, int, int) : void
  ● s main(String[]) : void
```

由于课表管理系统里的课程是可以多次重复出现的，所以我选用 CommonMultiIntervalSet 作为基础类，又由于课程是周期性重复出现的，所以我使用 PeriodicMultiIntervalSet 来装饰 CommonMultiIntervalSet。

```
private Date startdate=new Date();
private long weeks;
private List<Course> courselist=new ArrayList<>();
private PeriodicMultiIntervalSet<Course> set=new PeriodicMultiIntervalSet<>(new CommonMultiIntervalSet<>());
```

运行时的图片：

```
1 设置学期开始日期:
2 设置总周数
0 退出
```

（我随便设置了学期开始日期和总周数）

学期开始日期:2021/2/3

总周数:4

```
3 添加课程
4 安排上课时间
5 打印一周的课表
6 查看所有课程
7 打印某一天所有课程
8 计算每周的空闲时间比例
9 计算每周的重复时间比例
0 退出
```

```
学期开始日期:2021/2/3
总周数:4
3 添加课程
4 安排上课时间
5 打印一周的课表
6 查看所有课程
7 打印某一天所有课程
8 计算每周的空闲时间比例
9 计算每周的重复时间比例
0 退出
3
请输入课程id:
200
请输入课程名称:
数学
请输入授课教师:
张三
请输入上课地点:
正心22
请输入周学时数(偶数):
4
```

```
学期开始日期:2021/2/3
总周数:4
3 添加课程
4 安排上课时间
5 打印一周的课表
6 查看所有课程
7 打印某一天所有课程
8 计算每周的空闲时间比例
9 计算每周的重复时间比例
0 退出
6
课程id:200      课程名称:数学      授课教师:张三      上课地点:正心22  周课时数未排满, 还需安排4节课
```

```
学期开始日期:2021/2/3
总周数:4
3 添加课程
4 安排上课时间
5 打印一周的课表
6 查看所有课程
7 打印某一天所有课程
8 计算每周的空闲时间比例
9 计算每周的重复时间比例
0 退出
4
请输入课程id:
200
请输入上课星期(输入1~7)
4
请选择上课时间(输入1~5):
1  8:00-10:00
2  10:00-12:00
3  13:00-15:00
4  15:00-17:00
5  19:00-21:00
1
```

```

学期开始日期:2021/2/3
总周数:4
3 添加课程
4 安排上课时间
5 打印一周的课表
6 查看所有课程
7 打印某一天所有课程
8 计算每周的空闲时间比例
9 计算每周的重复时间比例
0 退出
5
第一节上课日期: 2021/2/3      8:00-10:00      星期4      课程id:200      课程名称:数学      授课教师:张三      上课地点:正心22

学期开始日期:2021/2/3
总周数:4
3 添加课程
4 安排上课时间
5 打印一周的课表
6 查看所有课程
7 打印某一天所有课程
8 计算每周的空闲时间比例
9 计算每周的重复时间比例
0 退出
6
课程id:200      课程名称:数学      授课教师:张三      上课地点:正心22      周课时数未排满, 还需安排3节课

```

### 3.7 基于语法的数据读入

先使用如下的正则表达式从文件中得到 Employee,Roster,Period 的对应内容

```

public boolean ReadFile(File fp) throws IOException {
    FileReader reader=new FileReader(fp);
    BufferedReader in=new BufferedReader(reader);
    String str="";
    String temp=null;
    while((temp=in.readLine())!=null) {
        str+=temp;
    }
    in.close();
    str=str.replaceAll("\\s", "");
    Pattern context=Pattern.compile("\\{.*\\}");
    Pattern employee=Pattern.compile("Employee\\{.*?\\}\\}");
    Pattern period=Pattern.compile("Period\\{.*?\\}\\}");
    Pattern roster=Pattern.compile("Roster\\{.*?\\}\\}");
    String employeestr=null;
    String periodstr=null;
    String rosterstr=null;
    String []employees=null;
    String []rosters=null;
    Matcher m=employee.matcher(str);

```

之后再对获得的块进行分别读取，读取的关键就在于要对正则表达式进行分组，这样才可以用 `group()` 函数得到我们想要的内容，在正则表达式中用 `()` 括住想要的内容就行，`()` 出现的顺序就代表子字符串分组的顺序。

#### 1 读取 Period

```

public void setfiledate(String str) {
    Pattern dateP=Pattern.compile("(\\d+)-(\\d+)-(\\d+),(\\d+)-(\\d+)-(\\d+)");
    Matcher m=dateP.matcher(str);
    int startyear;
    int startmonth;
    int startday;
    int endyear;
    int endmonth;
    int endday;
    if(m.find()) {
        startyear=Integer.valueOf(m.group(1));
        startmonth=Integer.valueOf(m.group(2));
        startday=Integer.valueOf(m.group(3));
        endyear=Integer.valueOf(m.group(4));
        endmonth=Integer.valueOf(m.group(5));
        endday=Integer.valueOf(m.group(6));
        setstartdate(startyear,startmonth,startday);
        setenddate(endyear,endmonth,endday);
    }
    else {
        System.out.println("文件格式有误");
        System.exit(0);
    }
}
}

```

## 2 读取 Employee

```

/**
 * 根据格式添加员工
 * @param str
 */
public void addfileemployee(String str) {
    Pattern s=Pattern.compile("[a-zA-Z+]{([a-zA-Z+), (\\d+)-(\\d+)-(\\d+)})");
    String phone="";
    String name=null;
    String position=null;
    Matcher m=s.matcher(str);
    if(m.find()) {
        name=m.group(1);
        if(getemployeeindex(name)!=-1) {
            System.out.println("文件格式错误:有员工名字重复");
            System.exit(0);
        }
        position=m.group(2);
        if(m.group(3).length()==3&&m.group(4).length()==4&&m.group(5).length()==4) {
            phone+=m.group(3)+m.group(4)+m.group(5);
            addemployee(name,position,phone);
        }
        else {
            System.out.println("文件格式错误:手机共 11 位，分为三段（3-4-4）");
            System.exit(0);
        }
    }
}
}

```

## 3 读取 Roster



```

/**
 * 根据文件排班
 * @param str
 */
public void insertfile(String str) {
    Pattern s=Pattern.compile("[a-zA-Z+]{1}((\\d+)-(\\d+)-(\\d+), (\\d+)-(\\d+)-(\\d+))");
    Matcher m=s.matcher(str);
    String name=null;
    int startyear;
    int startmonth;
    int startday;
    int endyear;
    int endmonth;
    int endday;
    if(m.find()) {
        name=m.group(1);
        startyear=Integer.valueOf(m.group(2));
        startmonth=Integer.valueOf(m.group(3));
        startday=Integer.valueOf(m.group(4));
        endyear=Integer.valueOf(m.group(5));
        endmonth=Integer.valueOf(m.group(6));
        endday=Integer.valueOf(m.group(7));
        if(getemployeeindex(name)==-1) {
            System.out.println("文件格式有误:出现在 Roster 内的员工, 必须在 Employee 中已有定义");
            System.exit(1);
        }
        else
            insert(name,startyear,startmonth,startday,endyear,endmonth,endday);
    }
    else {
        System.out.println("文件格式有误");
        System.exit(0);
    }
}
}

```

## 3.8 应对面临的新变化

### 3.8.1 变化 1

可以，基本没有代价，需要修改的地方如下图所示：

#### 1 需要修改装饰

```

private Date startdate=new Date();
private Date enddate=new Date();
private List<Employee> employeelist=new ArrayList<>();
private NonOverlapMultiIntervalSet<Employee> set= new NonOverlapMultiIntervalSet<>(new CommonMultiIntervalSe

```

#### 2 需要修改得到 set 的拷贝对象的函数

注：其实前面的设计有点冗余，DutyRosterApp 类里本身的方法就可以检查时间段是否有空闲，（主要最开始做的时候受到了 lab3 指导书的影响就创建了一个 NoBlankIntervalSet 的类后面也一直没改。。。。）当然把检查是否有空闲时间段放到 NOBlankIntervalSet 里肯定是更好的，因为这也是一个通用方法。

```

public NonOverlapMultiIntervalSet<Employee> getIntervalSet(){
    NonOverlapMultiIntervalSet<Employee> temp= new NonOverlapMultiIntervalSet<>(new CommonMultiIntervalSet<>());
    for(MyMap<Employee> e:set.arrangestart()) {
        temp.insert(e.start, e.end, e.label);
    }
    return temp;
}

```

### 3.8.2 变化 2

可以，基本没有代价，需要修改的地方如下图所示：

#### 1 需要修改装饰

```
private List<Course> courseList=new ArrayList<>();
private PeriodicMultiIntervalSet<Course> set=new PeriodicMultiIntervalSet<>(new NonOverlapMultiIntervalSet<>(n
```

## 3.9 Git 仓库结构

```
魏志豪@LAPTOP-SNU08S3D MINGW64 ~/Desktop/HIT-Lab3-1190302020 (change)
$ git log
commit 69873cf70d649b03900ef7bf20ea70b812f2f3c6 (HEAD -> change)
Author: weizhiao <1207410841@qq.com>
Date: Sat Jul 3 18:21:13 2021 +0800

    change1

commit 5a82b81199560023505da0904b3918ae6c90953b (origin/master, master)
Author: weizhiao <1207410841@qq.com>
Date: Sat Jul 3 16:42:51 2021 +0800

    master
```

## 4 实验进度记录

日期	时间段	计划任务	实际完成情况
2021/6/28	11:00-23:00	实现共性方法	按时完成
2021/6/29	11:00-0:00	完成装饰器	按时完成
2021/6/30	12:00-0:30	完成 CommonADT	按时完成
2021/7/1	12:00-2:00	完成排班表	按时完成
2021/7/2	12:00-1:00	完成课程表管理和操作系统进程调度	按时完成
2021/7/3	11:00-15:00	完成后续所有内容	按时完成

## 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
最开始使用设计方案 5：CRP，通过接口组合实现局部共性特征的复用，后来在委托方面遇到了困难	当机立断，转换设计思路使用方案 6 的装饰器方法实现 lab3，只不过现在做完了，也明白当初方案 5 的问题出在哪了。

日期的计算，用 java 自带的 Calendar 类计算两个日期之间相差的天数时会出现将 1 变为 0 的问题	通过测试我发现这与 long 类型的除法问题有关，所以我决定用减法来代替除法实现天数的计算
不会使用正则表达式	上网找资料

## 6 实验过程中收获的经验、教训、感想

### 6.1 实验过程中收获的经验教训

- 1 好的抽象可以帮助应用程序更好的实现。
- 2 好的设计方案可以减少代码的书写，在后来应用发生改变的时候也能轻松适应应用的变化。
- 3 学到的东西只有会用才会真正变成自己的东西。
- 4 改变设计方案要趁早，要不然积重难返。

### 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 编程是面向应用场景编程的基础，只有好的 ADT 才能更好的帮助我们面向应用编程。体会到了，如果不复用的话估计一周都写不完 lab3。

- (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？

防止在面向应用场景编程的时候发生一些匪夷所思的错误，这种错误通常十分难发现。我会坚持这么做的。

- (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？

API 还是很好用的，但设计起来比较麻烦。

- (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？

正则表达式实在是太强大了，有了它读取规定格式的文件方便多了，健壮性也变得更好了。

- (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？

难度主要在通用性，通过将 ADT 映射到现实世界里的客观事物中，再根据客观事物所具有的规律来设计 ADT，这样设计起来会更方便，也会更具有逻辑性。

- (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？

一个好的设计方式能够帮助我们实现最大程度的抽象和复用。

- (7) 关于本实验的工作量、难度、deadline。

本实验的工作量实现了质的飞跃，大概有 5 个 lab2 那么多，一共花费了 40 个小时左右的时间；难度适中但是做起来很花时间；抛开考试来说 deadline 设置的时间其实挺合理的，但是由于最后一次实验在考试周期间，就导致时间很不够用，复习的时间被大大压缩。

- (8) 到目前为止你对《软件构造》课程的评价。

获益匪浅，感觉程序设计水平比之前有了很大的提高。