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	
学号	
班号	
电子邮件	
手机号码	

目录

目录

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：

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

使用 IDEA+JDK8+Junit

https://github.com/ComputerScienceHIT/HIT-Lab3-1190300321.git

3 实验过程

请仔细对照实验手册，针对每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来!）。

3.1 待开发的三个应用场景

同：三个应用场景都是类似于安排表的实现形式，都是为一些元素安排一段时间的形式。

异：1.具体表现形式不同，有的可以多段插入，有的只能一个元素对应一段时间；有的可以存在空白，有的必须把安排表排满。前两个任务都没有周期性的要求，第三个任务有周期性的要求。

2.实现方法不同，针对表现形式不同，需要使用不同的 ADT 来实现这三个应用场景。

分析三个应用场景的异同，理解需求：它们在哪些方面有共性、哪些方面有差异。

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

为 IntevalSet 设计了插入，删除，返回元素，返回某个元素对应的开始时间和某个元素对应的结束时间这几个操作，主要是针对三个具体应用场景中的一些最基本的操作设计的。

以插入为例，主要功能是向 InevalSet 中插入一个元素，并设定其开始时间与结束时间。详细 Spec 如下图所示：

```
/**
 * 在当前对象中插入一个新的标签和时间段，start和end都应该是非负值且应该满足start<end
 * 如果label已经存在，更新其关联的时间片段
 * @param start 时间段的开始时间
 * @param end 时间段的结束时间
 * @param label 插入的标签
 */
public void insert(long start,long end,L label);
```

图 3.1 insert 的 spec

其余操作基本思想类似，主要是将 intervalSet 设计为一个最基本的 ADT，方便后续任务使用。

3.2.2 局部共性特征的设计方案

局部共性使用 CRP 方法设计，通过 delegation 机制进行改造。每个维度分别定义自己的接口， 针对每个维度的不同特征取值，分别实现针对该维度接口的不同实现类，实现其特殊操作逻辑。进而，通过接口组合，将各种局部共性行为复合在一起，形成满足每个应用要求的特殊接口（包含了该应用内的全部特殊功能）， 从而该应用子类可直接实现该组合接口。

主要涉及的不同维度为 NoBlankSet，NoOverLapSet，PeriodSet 三个维度，在这三个维度中为了方便委派，分别定义了各自的检测方法，分别用于检测是否存在空白，是否存在重叠，而 PeriodSet 较为不同，因为这一类型与前面的类型不同，需要额外管理一个周期信息，因此将 insert 等方法进行重写，方便后续调用。前面两个 ADT 中的检测方法分别设计了针对 IntervalSet 和 MultiIntervalSet 的检测方法，为了方便后续调用和修改，除了输入的 Set 的类型不一样，其余形参完全一致，这一设计主要是为了提高可维护性，在子类型改变继承关系的时候尽量减少代码的修改量。

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）

1. IntervalSet 的一个子类型 DutyIntervalSet 设计：

```
public class DutyIntervalSet<L> extends CommonIntervalSet<L>
        implements IDutyIntervalSet<L> {
```

图 3.2 dutyIntervalSet 设计

dutyIntervalSet 继承了 IntervalSet，是 IntervalSet 的一个子类，其设计基本与 IntervalSet 一致，复用了 IntervalSet 的 insert 和 remove 等方法，并在这一基础上主要修改了 checkRep 和 toString 方法，和 IntervalSet 最大的不同就是这个子类型需要检查排班表中是否存在空白，其余功能与 IntervalSet 基本一致。

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

和 InterValSet 类似，为 MultiIntervalSet 设计了 Insert 和 remove 等方法。其中 Insert 复用了 intervalSet 的方法，这是因为 MultiIntervalSet 在设计的时候与 InterValSet 类似，不同的就是一个的安排情况使用的一维数组管理，一个使用的是二维数组管理，其余基本类似。针对共性操作主要是插入，删除等最基本的操作。

```
public class CommonMultiIntervalSet<L> extends MultiIntervalSet<L> {
    final private Map<L, List<List<Long>>> multiIntervalMap =new HashMap<>();
    final private Map<L,IntervalSet<Integer>> multiIntervalMaptemp=new HashMap<>();
    final private Set<L>labelSet=new HashSet<>();
    /**
     * AF:
     * AF(multiIntervalMap)=标签与时间段的分配关系
     * AF(labelSet)=所有标签的集合
     * RI:
     * 1、labelSet中的Label不重复
     * 2、intervalMap中的key都在LabelSet中
     * Safety from Rep Exposure:rep使用final private，必要时候使用防御式拷贝
     */
```

图 3.3 MultiIntervalSet 的 rep

MultiIntervalSet 是一个 Mutable 的 ADT，为其设计了构造函数可以根据一个初始 IntervalSet 来初始化它的参数，详细情况如下：

```java
/**
 * 构造函数
 * @param initial 初始参数
 */
public MultiIntervalSet(IntervalSet<L> initial) {
    Set<L> keySet=initial.labels();
    for(L key:keySet)
    {
        try
        {
            this.insert(initial.start(key),initial.end(key),key);
        }catch (Exception e)
        {
            System.out.println("参数不正确");
            e.printStackTrace();
        }
    }
}
```

图 3.4 构造函数

为了在后续使用过程中在子类型中能与 InterValSet 的使用基本相似，MultiIntervalSet 的所有方法的名字与形参斗鱼 IntervalSet 的设计一致，这也是为了后续在 change 那一步两种不同的 ADT 之间在切换的时候子类型中的很多方法都不要修改参数。

3.3.2 局部共性特征的设计方案

局部共性使用 CRP 方法设计，通过 delegation 机制进行改造。每个维度分别定义自己的接口， 针对每个维度的不同特征取值，分别实现针对该维度接口的不同实现类，实现其特殊操作逻辑。进而，通过接口组合，将各种局部共性行为复合在一起，形成满足每个应用要求的特殊接口（包含了该应用内的全部特殊功能）， 从而该应用子类可直接实现该组合接口。在应用子类内，不是直接实现每个特殊操作，而是通过 delegation 到外部每个维度上的各具体实现类的相应特殊操作逻辑。

主要涉及的不同维度为 NoBlankSet，NoOverLapSet，PeriodSet 三个维度，在这三个维度中为了方便委派，分别定义了各自的检测方法，分别用于检测是否存在空白，是否存在重叠，而 PeriodSet 较为不同，因为这一类型与前面的类型不同，需要额外管理一个周期信息，因此将 insert 等方法进行重写，方便后续调用。

3.3.3 面向各应用的 MultiIntervalSet 子类型设计（个性化特征的设计方案）

在任务要求没有发生变化的时候，应用场景 2 和应用场景 3 都是使用 MultiIntervalSet 这个 ADT 来作为最基本的基本类型，以下详细说明。

1.针对应用场景 2，设计子类型 ProcessIntervalSet，详细 AF，RI，Rep 如下：

图 3.5 processIntervalSet 设计信息

在设计过程中基本的方法没有改变，主要加入了检测是否重叠的的功能，这是因为根据应用场景 2 的要求，两个进程执行时间之间不能存在重叠的片段，这一功能主要是使用 NonOverLapSet 这个 ADT 中的检测功能使用委派的方式来实现的，详细实现如下：

```
public void checkNoOverlap()throws Exception
{
    long start=Long.MAX_VALUE;
    long end=0;
    for(L key:this.multiIntervalSet.labels())
    {
        IntervalSet<Integer>tempSet=multiIntervalSet.intervals(key);
        start=Math.min(start,tempSet.getStart());
        end=Math.max(end,tempSet.getEnd());
    }
    nois.checkNoOverlap(this.multiIntervalSet,start,end);
}
```

图 3.6 processIntervalSet 检测无重叠

2.针对应用场景 3，设计子类型 CourseInterValSet，详细 AF，RI，Rep 如下：

```
public class CourseIntervalSet<L>  implements IcourseIntervalSet<L> {
    /**
     * AF(periodicIntervalSet)=安排表
     * AF(courseSet)=所有课程表
     * AF(courseMap)=所有课程的周期性记录
     * rep:安排表里的课程必须在所有课程集合里
     * RI：使用private final，必要时候使用防御式拷贝
     */
    private final PeriodicIntervalSet<L>periodicIntervalSet=new PeriodicIntervalSetImpl<>();
    private final Set<L> courseSet=new HashSet<>();
    private final Map<L, List> courseMap=new HashMap<>();
```

图 3.7 CourseInterValSet 设计信息

针对这一应用场景，在未修改之前主要的特征是可以存在重叠和空白，但是需要是具有周期的，因此在设计的时候使用的是 PeriodIntevalSet 作为其中的一个 rep，主要实现的方法和前述 ADT 基本一致，较为不同的就是维护了一个 Map 来记录所有课程的周期性信息，同时较为不同的是在安排课程之前必须先设置课程的周期性信息，这也是针对应用场景 3 设计的较为方便调用的形式。如下方法就是为了设置课程的周期性信息设计的：

```
public void setPeriod(L label,long start,long end)
{
    if(start>end)
    {
        System.out.println("周期时间设置错误");
        return;
    }
    List<Long>tempList=new ArrayList<>();
    tempList.add(start);
    tempList.add(end);
    courseMap.put(label,tempList);
    checkRep();
}
```

图 3.8 设置课程周期性信息

3.4 面向复用的设计：L

由于后续安排表中元素的类型可能都不同，所以在设计的时候尽量使用泛型 L 来设计，在设计的时候不使用一些具体类型的数据可能具有的方法来设计，只有在最后设计 APP 的时候才将 L 转化为具体的数据类型，例如在应用场景 1 中使用 Employee 作为基本数据类型，而应用场景 2 中使用 Process，应用场景 3 使用 Course 作为基本数据类型。使用泛型的设计使得设计的时候不需要对于每一种不同的数据类型设计不同的 ADT，底层 ADT 只需要设计一次，而顶层设计在调用的时候再决定使用的数据类型是什么。使用泛型的意义在于 1,适用于多种数据类型执行相同的代码（代码复用）2, 泛型中的类型在使用时指定，不需要强制类型转换（类型安全，编译器会检查类型）

3.5 可复用 API 设计

3.5.1 计算相似度

计算相似度以及以下的可复用的 API 设计都是在一个单独的 APIs 这一 ADT 中实现的，在计算相似度的时候主要实现了输入两个 MultiIntervalSet 比较二者的相似度。因此存在两个计算相似度的函数，方法的实现使用遍历的方法计算相似度。在调用的时候使用 API 这个实现类中的 Similarity 函数，这个函数输入两个 MultiIntervalSet，输出一个 double 表示二者的相似度，具体 Spec 如下图所示：

```
/**
 * 计算两个 MultiIntervalSet 对象的相似度
 * @param s1 需要计算相似度的一个MultiIntervalSet
 * @param s2 需要计算相似度的一个MultiIntervalSet
 * @return 相似度，小数形式
 */
public double Similarity(MultiIntervalSet<L> s1, MultiIntervalSet<L> s2);
```

图 3.9 Similarity 的 spec

3.5.2 计算时间冲突比例

计算时间冲突的方法同样是在 APIs 中实现的，主要存在两个形参不同的函数用于计算 Intevalset 和 MultiIntervalSet 两种数据类型的时间冲突比例。使用 overload 方法来构造两个方法。两个方法从根本上来说是一致的，都是使用遍历的方法来查看是否存在时间冲突，如果存在冲突的话记录存在冲突的时间长度，当遍历结束的时候将记录存在冲突的时间长度的变量除以整个时间的长度得到最后的结果。在调用的时候两个方法调用方式一致，都是直接输入一个 Set 即可。详细方法的 spec 如下：

```
/**
 * 计算一个intervalSet中的时间冲突比例
 * @param set 需要计算的intervalSet
 * @return 返回冲突比例
 */
public double calcConflictRatio(IntervalSet<L> set);

/**
 * 计算一个MultiIntervalSet中的时间冲突比例
 * @param set 需要计算的MultiIntervalSet
 * @return 返回冲突比例
 */
public double calcConflictRatio(MultiIntervalSet<L> set);
```

图 3.10 calcConflictRatio 函数 spec

3.5.3 计算空闲时间比例

计算空闲时间的比例的设计方法与上述两个方法基本一致，都是使用了 overload 的方式，构造了两个计算 MultiIntervalSet 和 IntervalSet 的不同方法。在调用的时候同样只需要输入 Set 即可，具体 spec 如下：

```java
/**
 * 计算一个IntervalSet的空闲时间占比
 * @param set intervalSet
 * @return 空闲时间占比
 */
public double calcFreeTimeRatio(IntervalSet<L> set);

/**
 * 计算一个MultiIntervalSet的空闲时间占比
 * @param set MultiIntervalSet
 * @return 空闲时间占比
 */
public double calcFreeTimeRatio(MultiIntervalSet<L> set);
```

图 3.11 calcFreeTimeRatio 函数的 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

排班管理系统主要是基于 DutyIntervalSet 进行的开发，实现的主要功能就是管理排班人员的信息，同时手动或者自动进行排班。首先实现的就是加入员工的功能，这一功能主要是向一个 Set 中加入每一个员工的信息，详细 rep 如下图所示：

```java
public class DutyRosterApp  implements DutyRoster {
    private final Set<Employee>EmployeeSet=new HashSet<>();
    private  String startTime;
    private  String endTime;
    private  IDutyIntervalSet<Employee>roster;
    private final List<Integer>arrangeList=new ArrayList<>();
    /**
     * RF:
     * RF(EmployeeSet)=存储每一个员工信息
     * RF(startTime)=值班表开始时间
     * RF(endTime)=值班表结束时间
     * RF(roster)=排班表
     * RF(arrangeList)=存储每一天是否已经安排的list
     * RI:使用private final，必要的时候使用防御式拷贝
     * Rep:
     * 1.一天不能安排多个人
     * 2.删除一个人之前要先把排班信息删掉
     * 3.所有人必须先添加才能排班
     */
```

图 3.12 排班管理系统 rep，RI，AF

主要实现的就是实验报告上要求的功能。主要工作流程就是首先设置排班表的起止时间，这一功能在管理系统中使用 setPeriod 函数实现，输入的是两个年

月日格式的时期，标志起止时间；接着就是向排班表中加入一些员工，需要的信息是员工姓名，职位和电话号码，其中电话号码的格式为 xxx-xxxx-xxxx；在加入员工之后就是选择手动或者自动排班。如果自动排班的话使用系统中的算法，尽量做到每一个加入排班表的员工的排班长度基本一致，在自动排班结束之后输出排班情况；如果采用手动排班则手动输入排班信息，在没有排满之前正常操作并不会退出系统，直到排满之后输出排班信息，退出系统。详细流程演示如下：

```
E:\JDK\JDK8\bin\java.exe ...
请输入排班表的开始时间，格式如2000-01-01
2021-01-01
请输入排班表的结束时间，格式如2000-01-01
2021-01-03
请输入值班人员信息，格式为姓名,职位,电话号码(xxx-xxxx-xxxx)
zhangsan,manager,111-1111-1111
请输入值班人员信息，格式为姓名,职位,电话号码(xxx-xxxx-xxxx)
e
排班表信息如下：

这一排班表空闲率为：1.0
需要安排的日期为：2021-01-01——2021-01-03
已经安排的日期及每天安排人为：

是否需要自动排班：y/n
n
请输入排班信息，格式为:姓名，开始日期，结束日期 日期格式为2000-01-01：
zhangsan,2021-01-01,2021-02
输入时间格式错误
java.lang.IllegalArgumentException: start应该比end小
    at APISet.DutyIntervalSet.DutyIntervalSet.insert(DutyIntervalSet.java:111)
    at API.DutyRoster.DutyRosterApp.setRoster(DutyRosterApp.java:103)
    at API.DutyRoster.DutyRosterApp.main(DutyRosterApp.java:350)
这一排班表空闲率为：1.0
需要安排的日期为：2021-01-01——2021-01-03
已经安排的日期及每天安排人为：
```

```
未安排时间：
2021-01-01
2021-01-02
2021-01-03

请输入排班信息，格式为:姓名，开始日期，结束日期 日期格式为2000-01-01：
zhangsan,2021-01-01,2021-01-02
这一排班表空闲率为：0.3333333333333333
需要安排的日期为：2021-01-01——2021-01-03
已经安排的日期及每天安排人为：
2021-01-01--zhangsan    manager    111-1111-1111
2021-01-02--zhangsan    manager    111-1111-1111
未安排时间：
2021-01-03

请输入排班信息，格式为:姓名，开始日期，结束日期 日期格式为2000-01-01：
zhangsan,2021-01-03,2021-01-03
这一排班表已经排满，详细信息如下：
需要安排的日期为：2021-01-01——2021-01-03
已经安排的日期及每天安排人为：
2021-01-03--zhangsan    manager    111-1111-1111
未安排时间：
```

图 3.13 排班管理系统执行流程演示

3.6.2 操作系统的进程调度管理系统

操作系统的进程调度管理系统中的基本数据结构为 Process，这个数据结构中定义了进程 ID，进程名字，进程最大执行时间和最小执行时间。这一管理系统是基于 ProcessIntervalSet 开发的，主要的功能就是实现对一组进程的调用。需要注意的是根据 ProcessIntervalSet 的定义，进程调用过程中两段调用之间是可以存在空白的但是不可以存在空白。为了方便调用，使用的随机调用或是优化调用两种方式二选一的方式进行调用，两种调用方式都是自动进行进程分配，随机分配采用随机选择进程的方式进行调度，而优化调用则选择最接近最小执行时间的进程调用的方式，主要调用形式如下：

```
请输入进程信息，格式为：ID,name,最短执行时间,最长执行时间
111,init,10,20
请输入进程信息，格式为：ID,name,最短执行时间,最长执行时间
222,process,15,20
请输入进程信息，格式为：ID,name,最短执行时间,最长执行时间
e
进程信息如下：

时间      ID   进程名称
详细安排如下：
安排表空闲率：1.0
是否需要优化进程安排：y/n
n
start应小于end
时间      ID   进程名称
详细安排如下：
0——16:111    init
16——27:222   process
27——28:222   process
28——32:222   process
安排表空闲率：0.0
```

图 3.14 操作系统的进程调度管理系统流程示意图

3.6.3 课表管理系统

课表管理系统是基于 CourseIntervalSet 开发的管理系统，主要特征是可以重叠，可以空白，存在周期。这一管理系统的最基本的数据结构就是 Course，这一数据结构包括一门课程的 ID，名字，教师姓名，上课地点，周学时数（偶数）。详细 AF 如下：

```
public class CourseScheduleAPP {
    /**
     * AF(courseSet)=所有课程集合
     * AF(courseIntervalSet)=课程安排集合
     * AF(arrangeMap)=表示某门课是否安排完的Map
     * AF(startTime)=学期开始时间
     * AF(length)=学期周数
     * Rep:
     * 1.所有课程应该都先加入courseSet
     * 2.课程安排不能超过周最大学时
     * RI:使用private final，必要时候防御式拷贝
     */
    private final Set<Course>courseSet=new HashSet<>();
    private final IcourseIntervalSet<Course>courseIntervalSet=new CourseIntervalSet<>();
    private final Map<Course,Boolean> arrangeMap=new HashMap<>();
    private final List<Course> courseList=new ArrayList<>();
    private String startTime;
    private int length;
```

图 3.15 课表管理系统详细信息

在执行的时候的流程是首先设置学期的开始日期和学期的周数，接着输入一组待分配的课程的信息；在课程信息输入完成之后对课程进行手动安排，如果一门课程达到最大周学时数则不再分配，为了方便管理，同时也是时间略显不足，每一门课的周期数都是设置为第一周到最后一周，如果后续有时间需要对周期数进行更详细的设置。详细运行流程如下：

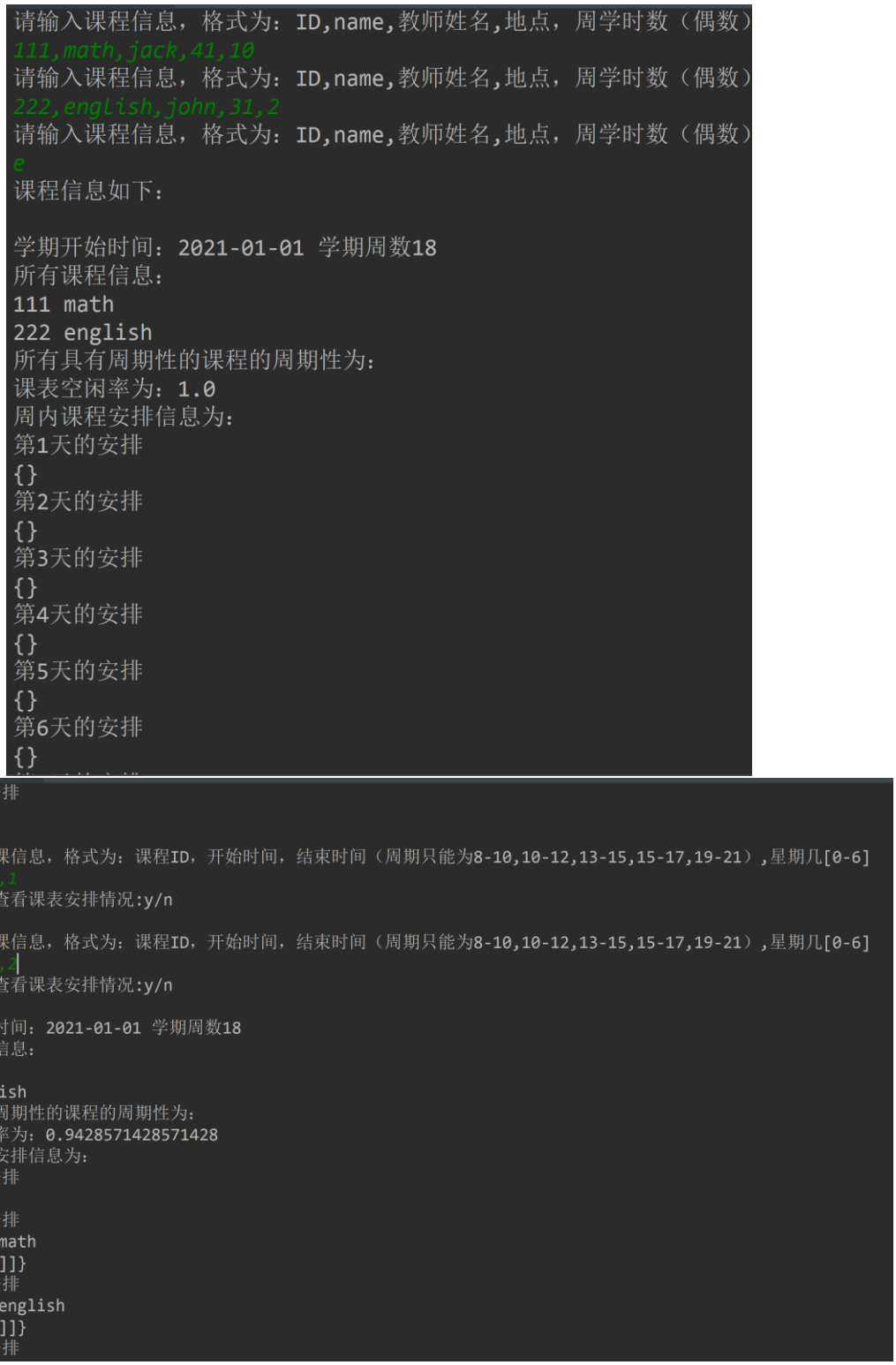

图 3.16 课表管理系统执行流程

3.7 基于语法的数据读入

基于语法的输入主要是使用正则表达式对读取文件中的信息进行匹配，主要执行方法是首先将文件中的所有信息读取，得到一个 String，之后根据不同部分将其分为两三个部分，对于三个部分分别进行解析，其中一部分解析方法见下图：

```
//添加安排
Pattern p2=Pattern.compile("\\s*(\\w+)\\{([^\\}]+)\\}");
Matcher m2=p2.matcher(partList.get(2));
while((m2).find())
{
    String name=m2.group(1);
    String start="";
    String end="";
    Pattern p3=Pattern.compile("(\\d{4})-(\\d{2})-(\\d{2}),(\\d{4})-(\\d{2})-(\\d{2})");
    Matcher matcher3=p3.matcher(m2.group(2));
    if(matcher3.find())
    {
        start=matcher3.group(1)+"-"+matcher3.group(2)+"-"+matcher3.group(3);
        end=matcher3.group(4)+"-"+matcher3.group(5)+"-"+matcher3.group(6);
    }
    this.setRoster(name,start,end);
}
```

图 3.17 读取时间安排信息

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，代价并不大。主要的修改是将 DutyIntervalSet 中的继承关系由 IntervalSet 改变为 MultiIntervalSet，接着修改在设计中使用的几处利用 IntervalSet 信息的方法即可，主要修改的方法有 toString 等方法。

3.8.2 变化 2

之前的设计可以应对变化，代价并不大。主要的修改是将在 CourseIntevalSet 的设计中增加一个 checkNoOverlap 的方法，这一方法主要是使用委派的方法，调用上述 NoOverlapIntervalSet 中的方法检测是否存在重叠。主要是在每一次插入安排的时候检测这次插入是否会和已经存在的安排产生重叠。

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。

```
11903@DESKTOP-UDFN7RA MINGW64 /e/Code/java/Lab3-1190300321 (master)
$ git log
commit 9a2fd3ac73626e06dcea27ae7f89609c58769a00 (HEAD -> master)
Author: zsh <531905990@qq.com>
Date:   Sun Jul 4 09:29:07 2021 +0800

    report

commit 5f58f78b56cf78683830a5e49be4fcc99a9c40e6 (origin/master)
Author: zsh <531905990@qq.com>
Date:   Sun Jul 4 01:02:29 2021 +0800

    master is almost OK
```

4 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

日期	时间段	计划任务	实际完成情况
2021.6.15	13：45-15:30	读实验要求，思考如何设计	完成
2021.6.22	13:45-15:30	写 IntervalSet 和 MultiIntervalSet 的 Spec 和测试用例	完成
2021. 6.27	19:00-23:00	写 IntervalSet 和 MultiIntervalSet，完成子类型的 spec 和测试样例	弯沉
2021.6.28	19:00-23:00	写完三个子类型设计，完成 dutyIntervalSet 的设计	完成
2021.6.29	13:45-18:00	写完 dutyIntevalSet 的测试样例和设计	未完成
2021.6.29	19:00-22:00	完成 dutyIntevalSet 的设计	完成

2021.6.30	19:00-23:45	完成 processIntevalSet 设计和 courseIntevalSet 的测试样例和 spec	完成
2021.7.1	19:00-24:00	完成 courseIntevalSet 设计	完成
2021.7.2	14:00-18:00 19:00-24:00	完成 dutyRosterApp 设计，ProcessscheduleApp 设计	完成
2021.7.3	8:00-11:00	完成 CourseScheduleApp 设计	完成
2021.7.3	14:00-17:30	修改上述三个 app 中的问题	完成
2021.7.3	18:45-2:00	完成基于语法的输入和修改	未完成
2021.7.4	8:00-9:30	完成修改	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
正则表达式书写不正确	查询有关资料后多次尝试解决
读取文件时格式有问题	读取文件时去掉所有的换行符再进行处理
刚开始设计的时候没有头绪	在设计好基本的继承关系之后开始写代码,在写完一部分之后对于设计有了初步认识

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验和教训

6.2 针对以下方面的感受

(1) 重新思考 Lab2 中的问题：面向 ADT 的编程和直接面向应用场景编程，你体会到二者有何差异？本实验设计的 ADT 在五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 的编程可能在设计的过程中会显得略为繁琐，但是在设计完成之后是可以用于多个应用场景的。如果直接面向应用场景编程容易导致每一个应用场景都要进行一次全新的设计。复用的好处就是一次设计之后可以用在多处。

(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？

这些工作的意义在于在复用的过程中更加方便，可以通过这些信息判断哪些部分需要修改。愿意。

(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？

自己尝试开发的时候发现开发一个 API 需要考虑的内容很多，首先就是正确性，其次是健壮性，同时还要考虑到可复用性。

(4) 你之前在使用其他软件时，应该体会过输入各种命令向系统发出指令。本次实验你开发了一个解析器，使用语法和正则表达式去解析输入文件并据此构造对象。你对语法驱动编程有何感受？

语法驱动编程可能在第一次设计的时候略微有些晦涩，但是是较为方便的设计方式，主要是为了针对复用，在文件格式发生变化的时候可以使用较小的代价进行修改。

(5) Lab1 和 Lab2 的工作都不是从 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，经过三周之后，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？

主要体现在每一个层次的设计都能较为完美的达到共性和个性的统一，同时在开始就考虑好如何用底层 ADT 实现上层功能也是比较困难的。主要是在设计的时候先考虑好应该如何适应上层调用和如何有更好的可复用性。

(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、委派、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？

在设计的使用应该多用委派而不是继承，将一些个性化的功能抽象出来之后形成一个独立的类，其他的类要使用的时候使用委派的方式进行调

用。

(7) 关于本实验的工作量、难度、deadline。

工作量和难度略大，但是主要是因为前几周准备考试没怎么写所以感觉工作量略大，考虑到三周的 deadline 工作量可以接受。

(8) 下周就要进行考试了，你对《软件构造》课程总体评价如何？

这门课是大学以来第一门真正到代码中的 ADT 和 OOP 以及一些设计模式的课程，上了半学期的课收获很大。