

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | cgh |
| 学号 | 1190501614 |
| 班号 | 1903006 |
| 电子邮件 | stucgh@qq.com |
| 手机号码 | 15778541719 |

**目录**

[1 实验目标概述 1](#_Toc76242944)

[2 实验环境配置 1](#_Toc76242945)

[3 实验过程 1](#_Toc76242946)

[3.1 待开发的三个应用场景 1](#_Toc76242947)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 2](#_Toc76242948)

[3.2.1 IntervalSet<L>的共性操作 2](#_Toc76242949)

[3.2.2 局部共性特征的设计方案 3](#_Toc76242950)

[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） 3](#_Toc76242951)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 4](#_Toc76242952)

[3.3.1 MultiIntervalSet<L>的共性操作 4](#_Toc76242953)

[3.3.2 局部共性特征的设计方案 4](#_Toc76242954)

[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） 4](#_Toc76242955)

[3.4 面向复用的设计：L 6](#_Toc76242956)

[3.5 可复用API设计 7](#_Toc76242957)

[3.5.1 计算相似度 7](#_Toc76242958)

[3.5.2 计算时间冲突比例 8](#_Toc76242959)

[3.5.3 计算空闲时间比例 8](#_Toc76242960)

[3.6 应用设计与开发 9](#_Toc76242961)

[3.6.1 封装ADT 9](#_Toc76242962)

[3.6.2 排班管理系统 10](#_Toc76242963)

[3.6.3 操作系统的进程调度管理系统 12](#_Toc76242964)

[3.6.4 课表管理系统 14](#_Toc76242965)

[3.7 基于语法的数据读入 16](#_Toc76242966)

[3.8 应对面临的新变化 18](#_Toc76242967)

[3.8.1 变化1 18](#_Toc76242968)

[3.8.2 变化2 18](#_Toc76242969)

[3.9 Git仓库结构 19](#_Toc76242970)

[4 实验进度记录 19](#_Toc76242971)

[5 实验过程中遇到的困难与解决途径 20](#_Toc76242972)

[6 实验过程中收获的经验、教训、感想 20](#_Toc76242973)

[6.1 实验过程中收获的经验和教训 20](#_Toc76242974)

[6.2 针对以下方面的感受 20](#_Toc76242975)

# 实验目标概述

本次实验覆盖课程第前两次课的内容，目标是编写具有可复用性和可维护性的软件，主要使用以下软件构造技术：

* 子类型、泛型、多态、重写、重载
* 继承、代理、组合
* 常见的OO设计模式
* 语法驱动的编程、正则表达式
* 基于状态的编程
* API设计、API复用

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

# 实验环境配置

实验前面已经配置好的环境，无需配置额外的实验环境

GitHub Lab3仓库的URL地址（Lab3-学号）：

<https://github.com/ComputerScienceHIT/Lab3-HIT-1190501614>

# 实验过程

## 待开发的三个应用场景

待开发三个应用为大学课表、值班表、进程调度。

1. **共性**：

三个应用均是关于时间段的分配，调度等问题，关键在于时间片段的管理和按排

三个应用在基本操作上有相似之处，插入、删除、查找等具有一定的共性

1. **差异**
2. **时间重叠问题**：值班表问题不允许不同标签在同一时间段重复，每个标签只一个连续的时间段（后续变化可扩展为多个时间段）；课程表问题允许在同一时间段存在重复；进程调度不允许重复，当前时间段只有一个进程
3. **空白问题：**值班表最后一般要求不能出现空白，即在某一时间段不能没有人值班，而进程调度问题允许CPU处于休眠状态（空白），课表也允许存在空白时间
4. **周期时间段：**课表是周期性的，而值班表和进程调度不是周期性的。
5. **多时间段问题：**值班表只允许每个标签有一个时间段，而课表和进程允许有多个时间段

## 面向可复用性和可维护性的设计：IntervalSet<L>

IntervalSet<L>作为底层接口，是三个问题共性的抽象

### IntervalSet<L>的共性操作

* **checkRep()**

检查当前状态的数据是否符合规约（这里为了方便分离一部分检查，放在了接口里面定义）

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

向当前计划中插入一个时间段，标签为label，开始时间为start，结束时间为end，通过一个定义一个Map容器timeslot来存储信息。

* **Set<L> labels()**
* Set<L> copySet = new HashSet<>();
* for (L unit:timeslot.keySet())
* copySet.add(unit);
* return copySet;

返回当前标签的副本

* **long start（L label）**

从timeslot中取出label的开始时间

* **long end（L label）**

从timeslot中取出label的结束时间

* **Map<L,Map<Long,Long>> getTimeslot()**
* Map<L,Map<Long,Long>> dup\_timeslot = new HashMap<>();
* dup\_timeslot.putAll(timeslot);
* return dup\_timeslot;

返回当前容器的副本

* **boolean remove(L label)**
* if(!timeslot.keySet().contains(label))
* return false;
* timeslot.remove(label);
* return true;

**移除某个标签的所有时间段**

### 局部共性特征的设计方案

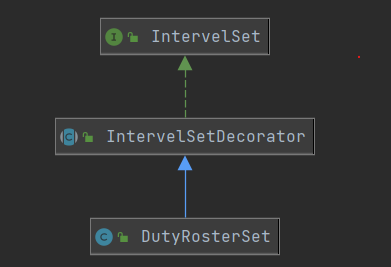
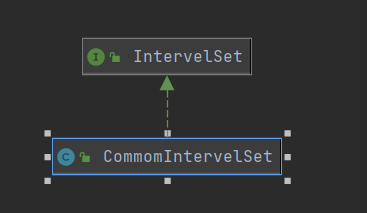
通过一个强大的Map容器（多重映射），可以为后续的扩展到multi类型时提供方便，直接复用IntervalSet的insert方法。此外，可以通过Map的内置函数快速实现相应的功能。最重要的是，这里将每个标签只能对应一个时间段的约束放到了子类的装饰器中，主要是为了尽可能实现multiIntervalSet的复用

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

采用装饰器模式进行设计，首先在前面已经实现了IntervalSet接口的具体类CommonIntervalSet。然后设计一个抽象类IntervelSetDecorator<L>实现接口IntervalSet，这个抽象类是基础装饰器，无需实现新的方法，可以直接复用已实现的方法（通过委派一个IntervalSet<L>对象）。然后根据值班表的个性化特征，通过另一个具体类DutyRosterSet<L>

（个性化装饰）继承之前的抽象类IntervelSetDecorator<L>（基础装饰器）来实现个性化功能。这里，我为了后续方便，在DutyRosterSet<L>中重写了insert方法和新增加了一个方法getendofstart获得开始时间对应的结束时间

结构图如下：



这里的装饰物只有DutyRosterSet<L>，为实现值班表个性化设计。而进程调度和课表主要通过MultiIntervelSet<L>来进行装饰，所以这里不设计关于课表和进程按排的子类型

## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

* 1. **Set<L> labels()**

**返回当前对象的标签集合，通过委派给IntervalSet对象去实现**

* 1. **boolean insert(long start, long end, L lab1)**

**插入标签的时间段，通过委派给IntervalSet对象去实现**

* 1. **boolean remove(L label)**

**移除有关标签的时间段，通过委派给IntervalSet对象去实现**

* 1. **IntervelSet<Integer> intervels(L label)**

**返回标签的所有时间段，是MultiIntervalSet的个性化方法，需独立实现**

* 1. **IntervelSet<L> getcommomIntervelSet()**

**返回委派的对象，读取时间信息**

### 局部共性特征的设计方案

通过委派，只有第４个和第５个函数为MultiIntervalSet独有的方法，需要独立实现，其它函数可复用IntervalSet的方法即可。

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

采用装饰器模式，定义一个基础装饰器MultiIntervelSetDecorator（抽象类）实现接口MultiIntervelSet，然后定义个性化装饰物ClassScheduleSet<L>（实现课表的个性化功能）和ProcessScheduleSet（实现进程调度的个性化功能），ProcessScheduleSet和ClassScheduleSet均继承MultiIntervelSetDecorator（抽象类）。

在**ProcessScheduleSet**中，新增加４个方法方便信息的读取，

* 1. **void insert\_process(long righttime,L label)**

**加入将进程信息加入到一个方便读取的列表中**

* 1. **ArrayList<L> getLabellist()**

**返回标签列表的副本（维护和时间列表的对应关系）**

* 1. **ArrayList<Long> getTimelist()**

**返回时间列表的副本（维护上述对应关系）**

* 1. **boolean removeall()**

**方便后续再次调用时清除已有的信息，可以实现多次模拟进程调度的需要**

在**ClassScheduleSet中，**新增３个方法

1. **boolean insert\_class(int day,long start,L course)**

**将课程加入到对应的星期几和时间段（比如8－10）**

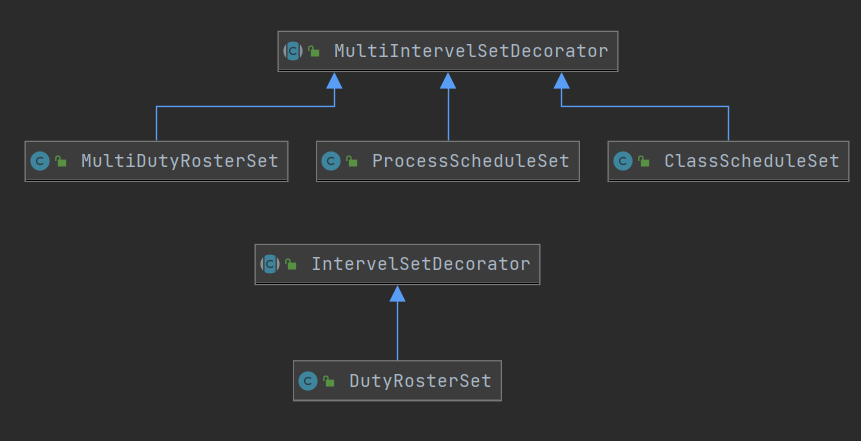
1. **void setClasstime()**

**初始化一张空课表**

1. **Map<Integer,Map<Long,Set<L>>> getClasstime()**

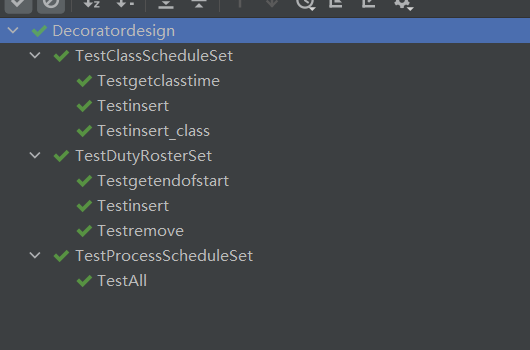
**返回当前的课表的副本，避免表示泄露**

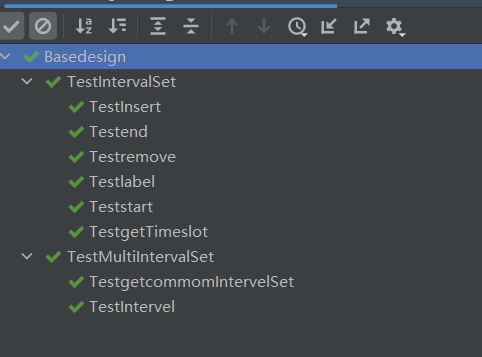
**最后结构图如下：（从IDEA导出，包含IntervalSet）**

****

****

**Junit测试结果：**

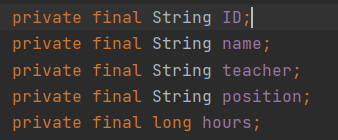
****

****

## 面向复用的设计：L

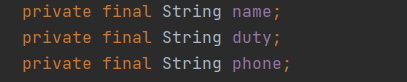
将员工、课表和进程封装成三个类，在进行ADT设计时，用泛型L来进行替换，可实现一些共性方法的复用。此外，在进行员工、课表或者进程的信息的扩展和变化时，减少对底层代码的改变，甚至无需改变底层代码，提供很好的可维护性。

* Class　Course



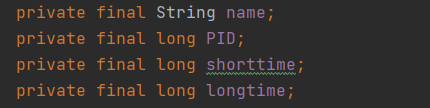
可通过构造相应方法返回相应字段的信息的副本

* **class Employee**



可通过构造相应方法返回员工信息副本，如getName函数获取姓名

* **class ProCess**

****

可通过构造相应方法返回员工信息副本，如getName函数获取姓名

## 可复用API设计

### 计算相似度

* **double Similarity(MultiIntervelSet<L> s1,MultiIntervelSet<L> s2)**

**计算两个MultiIntervelSet<L>对象的相似度，算法核心如下：**

for(L label : starttime1.keySet())

    {

        for(int i=0;i<starttime1.get(label).size();i++)

        {

            if(starttime2.get(label)==null)

            {

                break;

            }

            for(int j=0;j<starttime2.get(label).size();j++)

            {

                long s2f = starttime2.get(label).get(j);

                long s2t = fromto2.get(s2f);

                long s1f = starttime1.get(label).get(i);

                long s1t = fromto1.get(s1f);

                if(s2t>s1f && s2f<s1t)

                {

                    long simf,simt;

                    simt= Math.min(s2t, s1t);

                    simf= Math.max(s2f, s1f);

                    similarity+=simt-simf;

                }

            }

        }

    }

### 计算时间冲突比例

* **double calcConflictRatio(****IntervelSet<L> set)**
* **double calcConflictRatio(MultiIntervelSet<L> set)**

**计算时间冲突比，可用于大学课表中计算时间冲突比例，算法核心：**

for(L label: set.getTimeslot().keySet())

    {

        for(long start :set.getTimeslot().get(label).keySet())

        {

            if(start<mint)

                mint=start;

            long end = set.getTimeslot().get(label).get(start);

            if(end>maxt)

                maxt=end;

            for(int i=(int)start;i<(int)end;i++)

            {

                if(times[i]==1)

                {

                    concflict++;

                }

                else {

                    times[i]=1;

                }

            }

        }

}

### 计算空闲时间比例

* **double calcFreeTimeRatio(MultiIntervelSet<L> set)**
* **double calcFreeTimeRatio(IntervelSet<L> set)**

**计算空闲时间比例，可用于大学课表设计，算法核心：**

 for(L label: set.getcommomIntervelSet().getTimeslot().keySet())

    {

　　for(long start :set.getcommomIntervelSet().getTimeslot().get(label).keySet())

   {

        if(start<mint)

            mint=start;

       long end = set.getcommomIntervelSet().getTimeslot().get(label).get(start);

       if(end>maxt)

             maxt=end;

       for(int i=(int)start;i<(int)end;i++)

       {

            times[i]=1;

       }

   }

}

    if(mint==Long.MAX\_VALUE-1)

        return 0.0;

    for(int i=(int) mint;i<(int) maxt;i++)

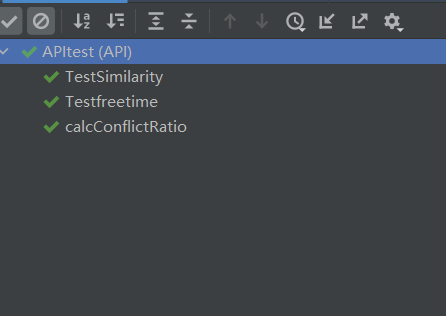
    {

        if(times[i]==0)

            freetimes++;

    }

**Junit测试结果：**

****

## 应用设计与开发

### 封装ADT

为了方便，进一步封装成三个具有特定功能的ADT，

* **CourseIntervalSet<L>**

**方法：**

* **CourseIntervalSet(LocalDate start, long weeks)**

**有参构造器**

* **boolean add\_course(L course,String name)**

**添加一组课程**

* **boolean select\_course(int day,long begin,long end,String name,long hours)**

**选择课程的安排时间**

* **void print\_Unscheduleclasss()**

**打印未安排课程信息**

* **void print\_perofleisure()**

**打印空闲时间比**

* **print\_perofrepeat()**

**打印冲突时间比**

* **void print\_curclassSchedule()**

**打印当前课表**

* **DutyIntervalSet<L>**

**方法：**

* **DutyIntervalSet(LocalDate start, LocalDate end)**

**构造器**

* **public DutyIntervalSet(String fileReader)**

**通过文件构造**

* **boolean addEmployee(LocalDate from,LocalDate to,L Label)**

**将员工加入值班表**

* **void deleteEmployee(L Label)**

**删除员工，附加功能**

* **void checkfull()**

**检查是否已满，打印空闲时间段及比例**

* **void showschedule()**

**展现当前课表**

* **ProcessIntervalSet<L>**

**方法：**

* **boolean addprocess(L PS,long longtime,long shorttime)**

**增加进程到等待队列中**

* **void randomexe()**

**随机执行调度**

* **void shortprocessexe()**

**最大进程最短优先**

* **void show()**

**展示当前进程调度情况**

### 排班管理系统

通过控制台交互，按照提示信息进行输入即可（注意输入时必须切换为英文状态，否则会报错），核心代码如下：

 while (true)

    {

        System.out.println("input your choice 1-4");

        Scanner input3 = new Scanner(System.in);

        int ch = input3.nextInt();

        switch (ch)

        {

            case 1: dutyRoster.checkfull();break;

            case 2: dutyRoster.showschedule();break;

            case 3:

                System.out.println("input the start date , ending date and the name of employee");

                System.out.println("input q to quit");

                System.out.println("yyyy-MM-dd,yyyy-MM-dd,name");

                while (true)

                {

                    Scanner input2 = new Scanner(System.in);

                    String addingmessage = input2.nextLine();

                    String[] messagess =addingmessage.split(",");

                    if(addingmessage.equals("q"))

                    {

                        break;

                    }

                    if(messagess.length!=3)

                    {

                        System.out.println("the format is wrong");

                        continue;

                    }

                    LocalDate from = LocalDate.parse(messagess[0],formatter);

                    LocalDate to = LocalDate.parse(messagess[1],formatter);

                    dutyRoster.addEmployee(from,to,employeegroup.get(messagess[2]));

                }

                break;

            case 4:break;

            default:break;

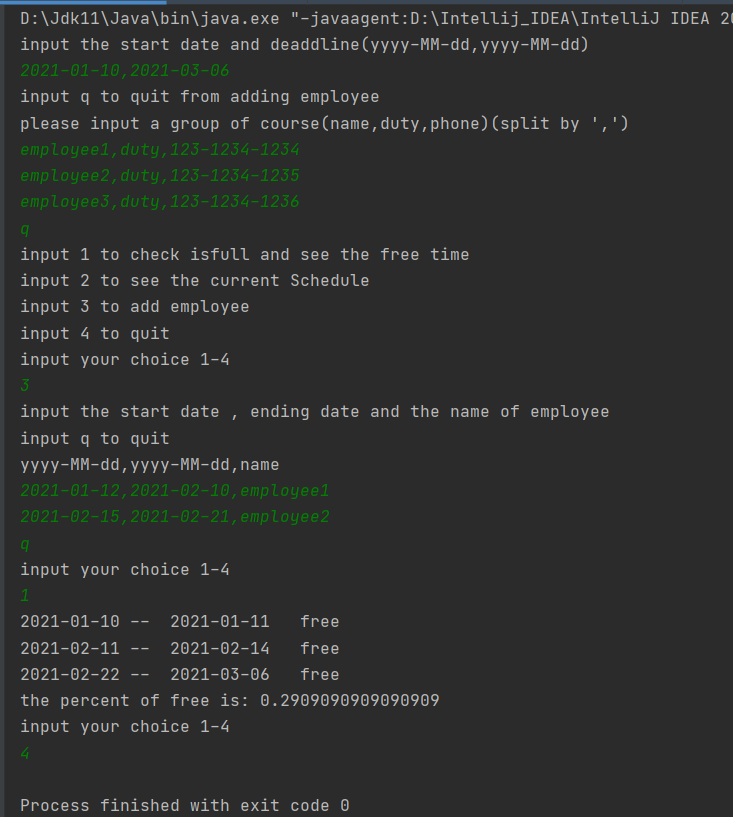
        }

        if(ch==4)

            break;

    }

测试截图：



### 操作系统的进程调度管理系统

通过控制台交互，注意输入时必须切换为英文状态，否则会报错，核心代码如下：

while (true)

    {

        System.out.println("input your choice 1-5");

        Scanner input = new Scanner(System.in);

        int ch = input.nextInt();

        switch (ch)

        {

            case 1:

                System.out.println("input q to quit");

                System.out.println("input(name,PID,Shorttime,longtime)");

                while (true)

                {

                    Scanner input1 = new Scanner(System.in);

                    String message = input1.nextLine();

                    String[] pmessage = message.split(",");

                    if(message.equals("q")) break;

                    if(pmessage.length!=4)

                    {

                        System.out.println("the wrong format");

                        continue;

                    }

                    long pid =Long.valueOf(pmessage[1]);

                    long st =Long.valueOf(pmessage[2]);

                    long lt =Long.valueOf(pmessage[3]);

                    ProCess proCess = new ProCess(pmessage[0],pid,st,lt);

                    processSchedule.addprocess(proCess,proCess.getLongtime(),proCess.getShorttime());

                }

                break;

            case 2:

                processSchedule.randomexe();

                break;

            case 3:

                processSchedule.shortprocessexe();

                break;

            case 4:

                processSchedule.show();

                break;

            case 5:break;

            default:break;

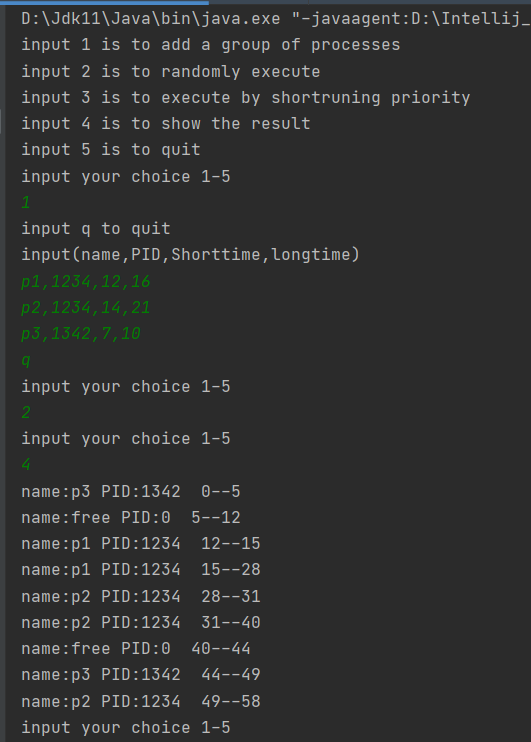
        }

        if(ch==5)

            break;

    }

测试截图：



### 课表管理系统

通过控制台交互，注意输入时必须切换为英文状态，否则会报错，核心代码如下：

 while (true) {

        System.out.println("input your choice  1-6");

        Scanner input3 = new Scanner(System.in);

        int ch = input3.nextInt();

        switch (ch) {

            case 1:

                courseSchedule.print\_perofleisure();

                break;

            case 2:

                courseSchedule.print\_perofrepeat();

                break;

            case 3:

                courseSchedule.print\_Unscheduleclasss();

                break;

            case 4:

                courseSchedule.print\_curclassSchedule();

                break;

            case 5:

                System.out.println("select the time of a course");

                System.out.println("Monday to Sunday is corresponding 1 to 7 and you can only chooose 8,10,13,15,19 at a day");

                System.out.println("you can only chooose the time of (8,10),(10,12),(13,15),(15,17),(19,21) at a day");

                System.out.println("input q to quit");

                System.out.println("input(weekday start end classname(split by ',')");

                while (true) {

                    Scanner input2 = new Scanner(System.in);

                    String classmessage = input2.nextLine();

                    String[] message1 = classmessage.split(",");

                    if (message1[0].equals("q")) break;

                    if(message1.length!=4)

                    {

                        System.out.println("the format is wrong");

                        continue;

                    }

                    int weekday = Integer.valueOf(message1[0]);

                    long from = Long.valueOf(message1[1]);

                    long to = Long.valueOf(message1[2]);

                    if(!nametohour.containsKey(message1[3]))

                    {

                        System.out.println("no such a course,input again");

                        continue;

                    }

                    long h = nametohour.get(message1[3]);

                    courseSchedule.select\_course(weekday, from, to, message1[3], h);

                }

                break;

            default:break;

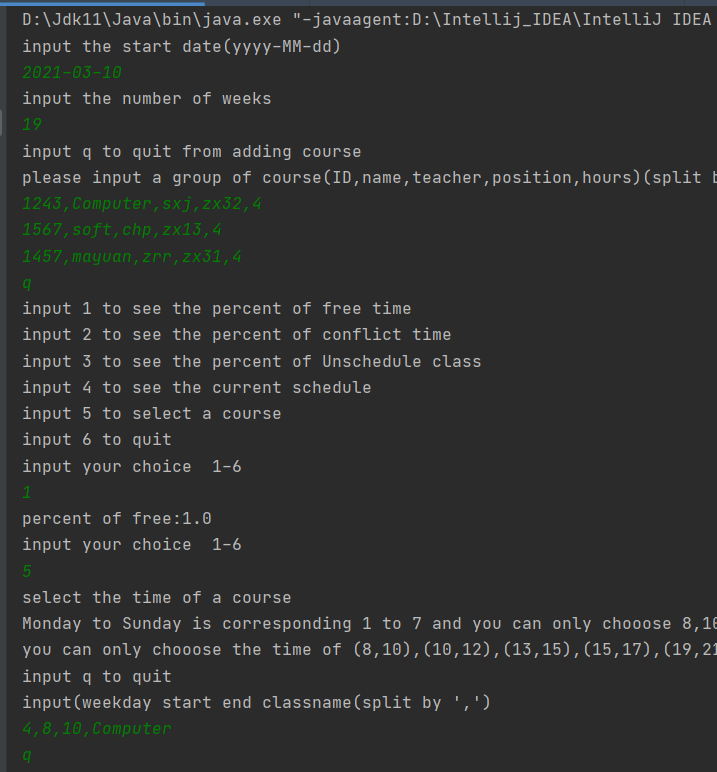
        }

        if(ch==6)

            break;

    }

测试结果：



## 基于语法的数据读入

新增一个方法来实现该功能

void readfromfile(String filename)

核心部分讲解：

* 正则表达式
* Pattern pattern = Pattern.compile("([a-zA-Z]+)\\s\*([a-zA-Z]+)");
* Pattern pattern1 = Pattern.compile("([0-9]{3})-([0-9]{4})-([0-9]{4})");
* Pattern pattern2 = Pattern.compile("([0-9]{4})-([0-9]{2})-([0-9]{2})");
* Matcher matcher = pattern.matcher(buffline);
* Matcher matcher1 = pattern1.matcher(buffline);
* Matcher matcher2 = pattern2.matcher(buffline);

Pattern匹配的是姓名和职位，pattern１匹配的是电话号码，需满足3-4-4关系；pattern２匹配的是日期，满足4-2-2结构。

* 不满足约束的非法检查：

1. 电话号码格式检查，3-4-4结构

if(phone.length()==0)

{

System.out.println("the file is illegal(the format of phone id wrong)");

    System.exit(0);

}

1. 名字中包含数字检查

if(matcher0.find())

{

    System.out.println("the file is illegal(name including number)");

    System.exit(0);

}

1. 职位中包含数字检查

if(matcher4.find())

{

    System.out.println("the file is illegal(duty including number)");

    System.exit(0);

}

1. 添加重复检查

if(employeeSet.containsKey(infor[0]))

{

    System.out.println("the file is illegal(including the same employee)");

    System.exit(0);

}

1. 日期格式检查

if(date[0].length()==0||date[1].length()==0)

{

    System.out.println("the file is illegal(the format of date is wrong)");

    System.exit(0);

}

1. 员工是否存在检查

if(employeeSet.get(infor[0])==null)

{

    System.out.println("the file is illegal(cannot found the employee)");

    System.exit(0);

}

## 应对面临的新变化

### 变化1

只需再增加一个装饰器即可，然后修改函数调用的返回值，不需要更改底层实现。

更改代码较小，相对容易

新增装饰类MultiDutyRosterSet<L>　继承MultiIntervelSetDecorator<L>

新增方法：

* ArrayList<Long> start(L label)

ArrayList<Long> starttime=new ArrayList<>();

for(Long time :multiIntervelSet.getcommomIntervelSet().getTimeslot().get(label).keySet())

{

    starttime.add(time);

}

return starttime;

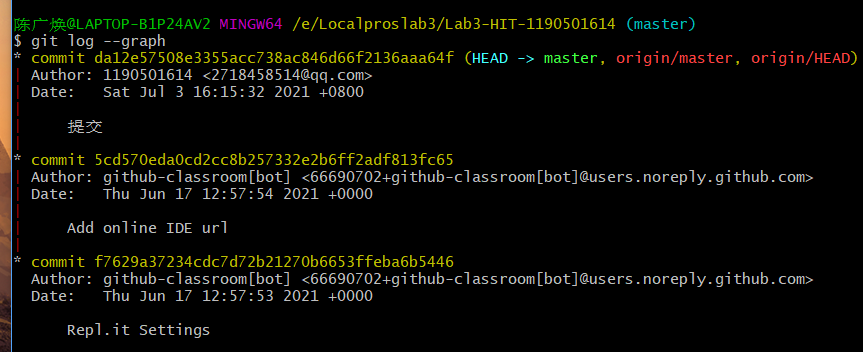
* Long getendofstart(Long time,L label)
* long ending = multiIntervelSet.getcommomIntervelSet().getTimeslot().get(label).get(time);
* return ending;

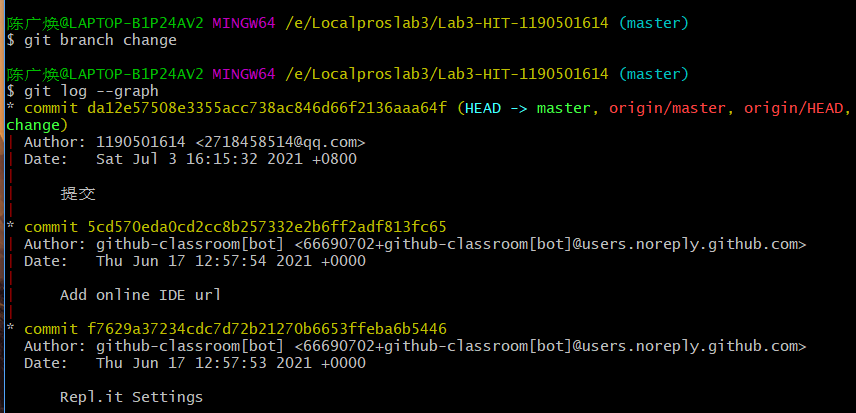
### 变化2

由于底层实现时将是否可重叠问题留给了装饰器去检查，所以只需要在相应装饰器里面更改检查条件即可，即在ClassScheduleSet<L>。或者再写一个装饰器继承已有的装饰器，只需重写checkRep()方法即可

## 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。





# 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 |
| 6-27 | 9：00-20：00 | 完成底层接口实现和测试 | 完成 |
| 6-28 | 10：00-23：00 | 设计方案选取 | 未完成 |
| 6-29 | 8：00－23：00 | 修改底层，选择装饰器模式 | 未完成 |
| 6-30 | 9；00-22：00 | 装饰器设计和测试 | 完成 |
| 7-1 | 8；30-23：00 | 完成ADT设计和测试 | 完成 |
| 7-2 | 8：30-23：00 | 完成APP设计，和语法读入 | 完成 |
| 7-3 | 8：00-23：00 | 完成变化修改和实验报告 | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 不会设计装饰器模式 | 通过博客，看别人的例子理解了该模式的原理 |
| 设计APP时，通过Scanner读入字符串错误 | 查阅资料得知每次需要去除缓冲区的回车符 |
| 不会正则语言语法  匹配错误 | 查阅网上博客  匹配前先分割字符串 |
| 每次运行进程调度程序时，  当再次运行时，会保留上次信息，无法多次模拟 | 通过在装饰器里面增加清除方法，每次模拟时清除上一次信息 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

* 要先考虑ADT的设计原则
* 认真学习设计模式，设计模式真的好用
* 不要过早优化，会影响程序可复用性

## 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？

ADT更加易于管理和维护

体会到了

1. 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？

避免错误蔓延，将错误限制在某一个类中

愿意

1. 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？

学习提供API能够在之后适应面向客户端编程

1. 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？

对ADT的要求更高了

1. 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？

设计模式和抽象，还有一些原则。

不断学习

1. “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？

抽象真的很重要，特别是面向OOP编程

1. 关于本实验的工作量、难度、deadline。

难度大，工作量大

1. 到目前为止你对《软件构造》课程的评价。

一言难尽，仍需不断学习