

**2021年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 冯开来 |
| 学号 | 1190201215 |
| 班号 | 1903007 |
| 电子邮件 | [3223200086@qq.com](mailto:3223200086@qq.com) |
| 手机号码 | 13771513877 |

**目录**

[1 实验目标概述 1](#_Toc74666198)

[2 实验环境配置 1](#_Toc74666199)

[3 实验过程 1](#_Toc74666200)

[3.1 待开发的三个应用场景 1](#_Toc74666201)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 1](#_Toc74666202)

[3.2.1 IntervalSet<L>的共性操作 1](#_Toc74666203)

[3.2.2 局部共性特征的设计方案 2](#_Toc74666204)

[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） 2](#_Toc74666205)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 2](#_Toc74666206)

[3.3.1 MultiIntervalSet<L>的共性操作 2](#_Toc74666207)

[3.3.2 局部共性特征的设计方案 2](#_Toc74666208)

[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） 2](#_Toc74666209)

[3.4 面向复用的设计：L 2](#_Toc74666210)

[3.5 可复用API设计 2](#_Toc74666211)

[3.5.1 计算相似度 2](#_Toc74666212)

[3.5.2 计算时间冲突比例 2](#_Toc74666213)

[3.5.3 计算空闲时间比例 2](#_Toc74666214)

[3.6 应用设计与开发 2](#_Toc74666215)

[3.6.1 排班管理系统 3](#_Toc74666216)

[3.6.2 操作系统的进程调度管理系统 3](#_Toc74666217)

[3.6.3 课表管理系统 3](#_Toc74666218)

[3.7 基于语法的数据读入 3](#_Toc74666219)

[3.8 应对面临的新变化 3](#_Toc74666220)

[3.8.1 变化1 3](#_Toc74666221)

[3.8.2 变化2 3](#_Toc74666222)

[3.9 Git仓库结构 3](#_Toc74666223)

[4 实验进度记录 3](#_Toc74666224)

[5 实验过程中遇到的困难与解决途径 4](#_Toc74666225)

[6 实验过程中收获的经验、教训、感想 4](#_Toc74666226)

[6.1 实验过程中收获的经验和教训 4](#_Toc74666227)

[6.2 针对以下方面的感受 4](#_Toc74666228)

# 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

实验环境设置请参见Lab-0 实验指南。

本次实验在GitHub Classroom中的URL地址为：

<https://classroom.github.com/a/ZAJ8w2eC>

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

https://github.com/ComputerScienceHIT/HIT-Lab3-1190201215.git

本地开发时，本次实验只需建立一个项目，统一向GitHub仓库提交。实验包含的多个任务分别在不同的包内开发，具体目录组织方式参见各任务最后一部分的说明。请务必遵循目录结构，以便于教师/TA进行测试。

# 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！）。

## 待开发的三个应用场景

简要介绍三个应用。

1. 值班表管理（DutyRoster）：一个单位有n个员工，在某个时间段内（例如寒假1月10日---3月6日期间），每天只能安排唯一1个员工在单位值班，且不能出现某天无人值班的情况；每个员工若被安排值班m天（m>1），那么需要安排在连续的m天内。值班表内需要记录员工的名字、职位、手机号码，以便于外界联系值班员。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 值班人名字 | 职位 | 手机号码 |
| 2021-01-10 | 孙XX | 书记 | 13900000000 |
| 2021-01-11 | 孙XX | 书记 | 13900000000 |
| 2021-01-12 | 孙XX | 书记 | 13900000000 |
| 2021-01-13 | 刘XX | 主任 | 13811111111 |
| 2021-01-14 | 张XX | 院长 | 18266666666 |
| 2021-01-15 | 张XX | 院长 | 18266666666 |
| 2021-01-16 | 王XX | 副院长 | 18677777777 |
| 2021-01-17 | 王XX | 副院长 | 18677777777 |
| 2021-01-18 | 邬XX | 副主任 | 15533333333 |
| … |  |  |  |

1. 操作系统进程调度管理（ProcessSchedule）：考虑计算机上有一个单核CPU，多个进程被操作系统创建出来，它们被调度在CPU上执行，由操作系统来调度决定在各个时段内执行哪个线程。操作系统可挂起某个正在执行的进程，在后续时刻可以恢复执行被挂起的进程。可知：每个时间只能有一个进程在执行，其他进程处于休眠状态；一个进程的执行被分为多个时间段；在特定时刻，CPU可以“闲置”，意即操作系统没有调度执行任何进程；操作系统对进程的调度无规律，可看作是随机调度。



1. 大学课表管理（CourseSchedule）：看一下你自己的课表，每一上午8:00-10:00和每周三上午8:00-10:00在正心楼13教室上“软件构造”课程。课程需要特定的教室和特定的教师。在本应用中，我们对实际的课表进行简化：针对某个班级，假设其各周的课表都是完全一样的（意即同样的课程安排将以“周”为单位进行周期性的重复，直到学期结束）；一门课程每周可以出现1次，也可以安排多次（例如每周一和周三的“软件构造课”）且由同一位教师承担并在同样的教室进行；允许课表中有空白时间段（未安排任何课程）；考虑到不同学生的选课情况不同，同一个时间段内可以安排不同的课程（例如周一上午1-2节的计算方法和软件构造）；一位教师也可以承担课表中的多门课程；



（为简化起见，无需考虑图中各课程后面标注的起至周次）

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。

共性：

1. 一个人可以值班多次，进程也可以被执行多次，一门课在一周也可以被安排多次。
2. 值班人名字和职位手机号存在某种关系，进程和它的PID存在关系，课程和老师教室存在关系。而且这些应该属于该类的属性（不可变）。
3. 都可以插入一个时间段，删除一个时间段，返回一个时间的开始时间和结束时间，返回所有时间段的标签。

差异：

值班表保证每个时间段都必须有人，但是操作系统进程和课表可以有闲置时间。

值班表和操作系统要求一个时间段只能有一个人值班，有一个进程在执行，但是课表可以有不同的课被同时安排

值班表要求同一个人必须连续值班，但是操作系统和课表中的进程和课程不要求连续。

一名值班人和他的职位和手机号成双射关系，一个进程和它的PID一一对应。虽然课程要求唯一对应一名老师和一间教室，但是老师不要求唯一对应一门课。

## 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>的共性操作

**3.2.1.1:IntervalSet<L> 实现**

返回一个空的时间段，其中泛型可以是employee，pid或者course

public static <L> IntervalSet<L> empty()

返回所有时间段的标签，返回的类型是一个集合

public Set<L> labels();

插入一个时间段，这个时间段要求有标签名以及对应的开始时间和结束时间

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

移除一个时间段，如果当前对象有这个时间段，返回ture，如果没有返回false

public boolean remove(L label);

返回某个时间段的开始时间

public long start(L label);

返回某个时间的结束时间

public long end(L label);

以上封装成接口IntervalSet<L>

**3.2.1.2:CommonIntervalSet<L> 实现**

public class CommonIntervalSet<L> implements IntervalSet<L>{

//TODO fields

private final Map<L, List<Long>> labelsMap = new HashMap<>();

// Abstract function:

// represent a intervalSet with start, end, name

// Representation invariant:

// the end is larger than start

// the start > 0

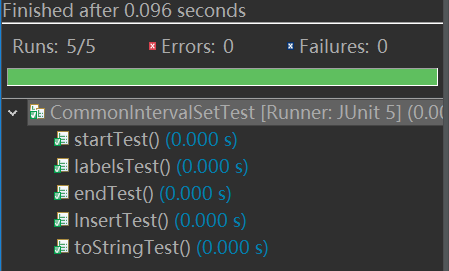
// Safety from rep exposure

// all fields are private final

}

通过设计Map类型的label和start与end的对应关系。具体实现有手就行，看代码就可以。

相比较接口，多了一个toString() 的方法，该方法是打印出时间表中所有时间标签和对应的开始结束时间。



### 局部共性特征的设计方案

**3.2.2.1:选择实验方案：**

为了提高代码的复用性，我选择了方案5来完成局部共性特征的设计，即CRP，通过接口组合实现局部共性特征的复用，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。

**3.2.2.2:实现设计：**

定义了多个接口：

检查多个时间段之间没有空隙：

public class NoBlankIntervalSetImpl<L> implements NoBlankIntervalSet<L>

|  |  |
| --- | --- |
| 方法 | 描述 |
| sort() | 对多个时间段的开始时间进行从小到大的排序 |
| checkNoBlank() | 检查是否有空隙  若开始时间和排序后的第一个开始时间不相等，返回false  若结束时间和排序后最后一个结束时间不相等，返回false  排序后的时间段为空，返回false  除了以上情况，每个时间段的结束时间等于下一个时间段开始时间，返回true |

检查一个标签中几个时间段是否有重叠：

Public class NonOverlapIntervalSetImpl<L> implements NonOverlapIntervalSet<L>

|  |  |
| --- | --- |
| 方法 | 描述 |
| Insert(long start, long end, L label) | 判断所插入时间段的开始时间是否在一个时间段内，若是，flag=false  判断所插入时间段的结束时间是否在一个时间段内，若是，flag=false  若flag==false，抛出异常 |

增加“非周期”行为：

public class NonPeriodicIntervalSetImpl<L> implements NonPeriodicIntervalSet<L>

|  |  |
| --- | --- |
| 方法 | 描述 |
| checkNonPeriodic(long start, long end) | 如果插入时间段的结束时间大于整体开始时间加持续时间或者插入时间段的开始时间小于整体开始时间，抛出异常，该时间段不得加入该interval |

定义IDutyIntervalSet接口继承上面三个接口

public interface IDutyIntervalSet<L> extends NoBlankIntervalSet<L>, NonOverlapIntervalSet<L>, NonPeriodicIntervalSet<L>

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

**3.2.3.1:DutyIntervalSet<L>**

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

//TODO fileds

private final NonOverlapIntervalSet<L> nois;

private final NoBlankIntervalSet<L> nbis;

private final IntervalSet<L> intervalSet = new CommonIntervalSet<>();

// Abstraction function:

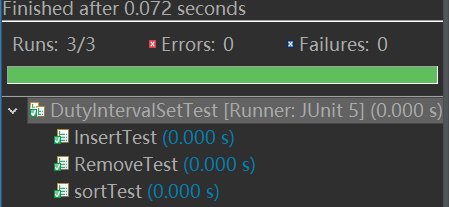
// Represent a dutyInterval with Employee, arrangement and time

// Representation invariant:

// there are no blanks, no overlap and no periodic in arrangement

// Safety from rep exposure:

// All fields are private final



|  |  |
| --- | --- |
| 方法 | 描述 |
| DutyIntervalSet(long startTime, long endTime) | 分别声明一个检验没有时间重叠和检验没有时间空隙的接口 |
| checkNoBlank() | 检验是否有时间空隙 |
| Insert(long start, long end, L label) | 通过没有时间重叠的方法插入该时间标签，即排班。  总的时间表中插入该时间段。 |
| Set<L> labels() | 返回总的时间表中所有的时间标签。 |
| remove(L label) | 移除一个时间标签  总的时间表若包含该标签，返回true  总的时间表不包含该标签，返回false |
| start(L label) | 返回该时间标签的开始时间 |
| end(L label) | 返回该时间标签的结束时间 |
| toString() | 返回总的时间中所有时间标签和对应的开始结束时间 |
| checkNonPeriodic(long start, long end) | 这个类没有用到，所以没有override |

## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

**3.3.1 MultiIntervalSet<L> 实现**

public class MultiIntervalSet<L>

//TODO fields

private final Map<L, List<Long>> labelsMap = new HashMap<>();

// Abstract function:

// represent a MultiIntervalSet with start, end and IntervalSet<Integer>

// Representation invariant:

// the end is bigger than start, the start > 0

// Safety from rep exposure:

// all fields are private final

检查rep exposure

private void checkRep()

返回一个空的时间段，其中泛型可以是employee，pid或者course

public static <L> MultiIntervalSet<L> empty()

构造函数MultiIntervalSet(IntervalSet<L> initial)，利用initial中包含的数据创建非空对象

public MultiIntervalSet(IntervalSet<L> initial)

返回所有时间段的标签，返回的类型是一个集合

public Set<L> labels();

插入一个时间段，如果这个时间段的标签已经存在，则按开始时间从小到大插入到这个标签中，若这个时间标签不存在，则新建一个时间标签并插入标签名以及对应的开始时间和结束时。

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

移除一个时间段，如果当前对象有这个时间段，返回ture，如果没有返回false

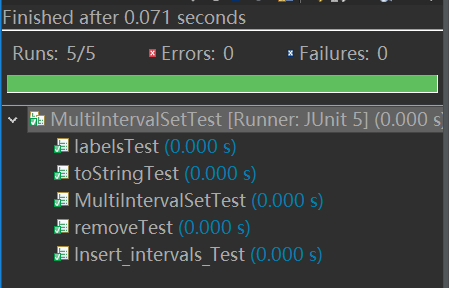
public boolean remove(L label);

返回某个时间标签的所有时间段

public IntervalSet<Integer> intervals(L label)

打印所有时间标签和对应的所有时间段

public String toString()



### 局部共性特征的设计方案

**3.3.2.1:选择实验方案：**

为了提高代码的复用性，我选择了方案5来完成局部共性特征的设计，即CRP，通过接口组合实现局部共性特征的复用，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。

**3.3.2.2:实现设计：**

定义了多个接口：

检查多个标签是否有重叠：

public class NonOverlapMultiIntervalSetImpl<L> implements NonOverlapMultiIntervalSet<L>

|  |  |
| --- | --- |
| 方法 | 描述 |
| Insert(long start, long end, L label) | 对每个标签进行上述检查 |

定义IProcessIntervalSet接口继承NonOverlapMultiIntervalSet

public interface IProcessIntervalSet<L> extends NonOverlapMultiIntervalSet<L>

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

**3.3.3.1:ProcessIntervalSet<L>**

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

private final NonOverlapMultiIntervalSet<L> noms;

private final MultiIntervalSet<L> multiIntervalSet = new MultiIntervalSet<>();

// Abstraction function:

// Represent a dutyInterval with Process and arrangement

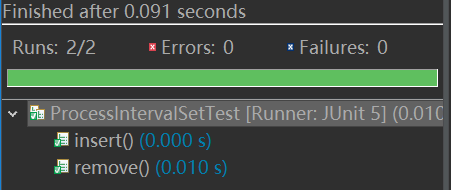
// Representation invariant:

// No overlap, no periodic

// Safety from rep exposure:

// All fields are private final, and safety from rep exposure.

|  |  |
| --- | --- |
| 方法 | 描述 |
| ProcessIntervalSet() | 声明一个检验没有时间标签重叠的接口 |
| Insert(long start, long end, L label) | 通过没有时间重叠的方法插入该时间标签，即排班。  并在总的时间表中插入该时间段。 |
| Set<L> labels() | 返回总的时间表中所有的时间标签。 |
| remove(L label) | 移除一个时间标签  总的时间表若包含该标签，返回true  总的时间表不包含该标签，返回false |
| intervals(L label) | 返回一个时间标签的所有时间段 |
| toString() | 返回总的时间中所有时间标签和对应的开始结束时间 |



**3.3.3.2:CourseIntervalSet<L>**

public class CourseIntervalSet<L> extends MultiIntervalSet<L>

private final NonPeriodicIntervalSet<L> npis;

private final MultiIntervalSet<L> multiIntervalSet = new MultiIntervalSet<>();

// Abstraction function:

// Represent a courseInterval with courses(id, name, teacher, classroom and timeOfWeek) and arrangement

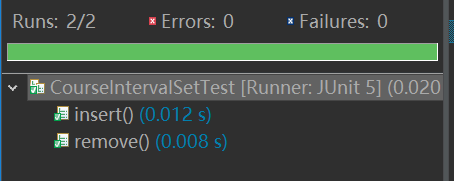
// Representation invariant:

// one course have only one id, only one name and only one classroom

// Safety from rep exposure:

// All fields are private final, and safety from rep exposure.

|  |  |
| --- | --- |
| 方法 | 描述 |
| CourseIntervalSet() | 声明一个检验增加”非周期“的接口 |
| Insert(long start, long end, L label) | 通过没有时间重叠的方法插入该时间标签，即排班。  并在总的时间表中插入该时间段。 |
| Set<L> labels() | 返回总的时间表中所有的时间标签。 |
| remove(L label) | 移除一个时间标签  总的时间表若包含该标签，返回true  总的时间表不包含该标签，返回false |
| intervals(L label) | 返回一个时间标签的所有时间段 |
| toString() | 返回总的时间中所有时间标签和对应的开始结束时间 |



## 面向复用的设计：L

**3.4.1员工的实现类Employee：**

public class Employee

private final String name;

private final String job;

private final String phoneNum;

|  |  |
| --- | --- |
| 方法 | 描述 |
| Employee(String name, String job, String phoneNum | 初始化构造 |
| getName() | 返回员工姓名 |
| getJob() | 返回员工职位 |
| getPhoneNum() | 返回员工电话号码 |

**3.4.2进程的实现类Process：**

public class Process

private final String pid;

private final String pName;

private final long minTime;

private final long maxTime;

|  |  |
| --- | --- |
| 方法 | 描述 |
| Process() | 初始化构造 |
| getPid() | 返回进程id |
| getPname() | 返回进程名字 |
| getMinTime() | 返回进程最小执行时间 |
| getMaxTime() | 返回进程最大执行时间 |

**3.4.3课程的实现类Course：**

public class Course

private final String cid;

private final String cName;

private final String tName;

private final String clsroom;

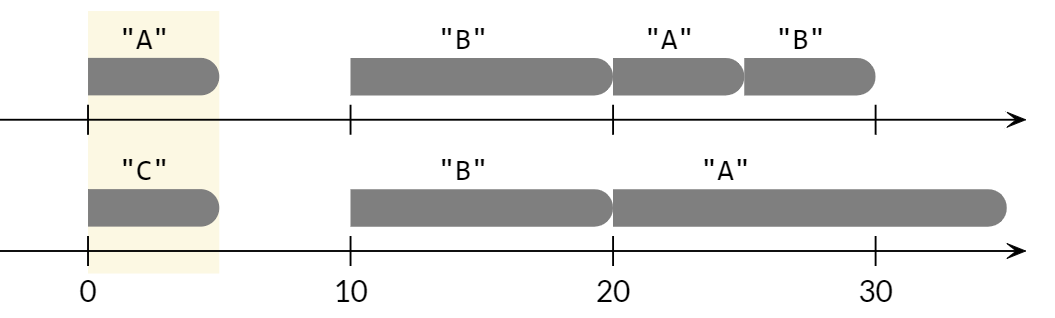
private final int timeOfWeek;

|  |  |
| --- | --- |
| 方法 | 描述 |
| Course() | 初始化构造 |
| getCourseID() | 返回课程id |
| getCourseName() | 返回课程名 |
| getTeacherName() | 返回课程教师名字 |
| getPlace() | 返回上课教室 |
| getTimeOfWeek() | 返回课时数 |

## 可复用API设计

### 计算相似度

具体计算方法：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。



针对上图中的两个对象：

{ A = [[0,5),[20,25)], B = [[10,20),[25,30)] }

{ A = [[20,35)], B = [[10,20)], C = [[0,5)] }

它们的相似度计算如下：

( 5×0 + 5×0 + 10×1 + 5×1 + 5×0 + 5×0 ) ÷ ( 35 - 0 ) = 15 / 35 ≈ 0.42857

### 计算时间冲突比例



声明一个exit数组，每次遇到没有安排的空闲时间，则空闲时间++，遇到大于1的时间段，重复时间段++，最后除以总时间段得到空闲率和重复率。

### 计算空闲时间比例



声明一个exit数组，每次遇到没有安排的空闲时间，则空闲时间++，遇到大于1的时间段，重复时间段++，最后除以总时间段得到空闲率和重复率。

## 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。

### 排班管理系统

public class DutyRosterApp

private final static long[][] dayOfMonth = { { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },

{ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 } };

private final static Scanner input = new Scanner(System.in);

private final static Map<Employee, Integer> People = new HashMap<>();

static long begin = 0;

static long end = 0;

static long totalLength = 0;

static DutyIntervalSet<Employee> dutyInterval = null;

|  |  |
| --- | --- |
| 方法 | 描述 |
| private static int menu() | 循环显示菜单 |
| private static int readNum() | 读入一个整型数 |
| private static long timeToLong(int year, int month, int day) | 计算整个时间段开始时间到结束时间经历了多少天 |
| private static long timeInput(String S) | 输入年月日，开始时间和结束时间，注意闰平年的转换 |
| private static int getYear(long sum) | 通过持续时间得到从开始时间到一个节点的年份 |
| private static int getMonth(long sum) | 通过持续时间得到从开始时间到一个节点的月份 |
| private static int getDay(long sum) | 通过持续时间得到从开始时间到一个节点的日期 |
| private static void showPeople(Map<Employee, Integer> people, int choice) | 返回可安排人员和已安排人员 |
| private static void readFile(int choose) | 读入编号为choose的文件 |
| private static void checkLine(String str) | 见3.7 |
| public static long checkFormat(String str) | 见3.7 |
| public static void main(String[] args) | 显示菜单，并根据菜单选项执行相应操作。 |

### 操作系统的进程调度管理系统

public class ProcessScheduleApp

static Scanner input = new Scanner(System.in);

static Map<Process, Long> processList = new HashMap<>();

static Process current = null;

static long start = 0;

|  |  |
| --- | --- |
| 方法 | 描述 |
| private static int readNum() | 读入一个整型数 |
| private static long timeToLong(int year, int month, int day) | 计算整个时间段开始时间到结束时间经历了多少天 |
| public static void main(String[] args) | 显示菜单，并根据菜单选项执行相应操作。 |
| private static void disSet(ProcessIntervalSet<Process> timeline) | 选择一个进程，执行时间最短优先 |
| private static void randomSet(ProcessIntervalSet<Process> timeline) | 随机选择一个进程 |
| public static void showString(ProcessIntervalSet<Process> processInterval) | 显示现在的时刻，现在正值执行的进程，以及现在进程之前的结果 |

### 课表管理系统

public class CourseScheduleApp

private static final long[][] dayOfMonth = { { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },

{ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 } };

private static final Scanner input = new Scanner(System.in);

private static final Map<String, Course> courseMap = new HashMap<>();

private static final Map<String, Integer> courseTime = new HashMap<>();

|  |  |
| --- | --- |
| 方法 | 描述 |
| private static int menu() | 循环显示菜单 |
| private static int readNum() | 读入一个整型数 |
| private static long timeToLong(int year, int month, int day) | 计算整个时间段开始时间到结束时间经历了多少天 |
| private static long timeInput(String S) | 输入年月日，开始时间和结束时间，注意闰平年的转换 |
| public static void main(String[] args) | 显示菜单，并根据菜单选项执行相应操作。 |
| private static void arrangeCourse | 安排课程，将课程安排在一周35个时间段内，并判断是否合法 |
| private static void insertCourse() | 插入一门新课程 |
| public static int timeChange(long i) | 时间表示转换 |

## 基于语法的数据读入

创建3个函数分别读取employee、period、roster。用ArrayList来存储每行读取到的字符串，然后创建文件流开始按行读取字符串，按照text中数据的特征进行分割。将“{”符号之前的字符串存在第一个数组中表示姓名，将“{”到“，”之间的字符串存在第二个数组中表示职业，将“,”到“}”的字符串存在第三个数组中表示电话号，这样得到了employee类的3个属性。同理对period和Roster进行解析。

## 应对面临的新变化

### 变化1

理论上可以通过DutyIntervalSet修改为继承MultiIntervalSet实现。

我通过直接修改源代码中让目录可以不停展示可以安排的人员，这样就能给一个员工安排多段时间。

### 变化2

这个设计已经实现了，在安排课程的函数里添加了检测是否重叠的条件判断，若已重叠会抛出异常，由于在主函数会调用安排课程的函数，所以并入了catch异常的处理，输出“输入错误”字样。

只要在源代码中加几行检查重复的代码即可。

## 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 |
| 6.29 | 8：00–23：00 | 看题，自顶而下设计代码 | 完成 |
| 6.30 | 8：00-23：00 | Package set | 完成 |
| 7.1 | 8：00-23：00 | Package check | 完成 |
| 7.2 | 8：00-23：00 | Package app | 完成 |
| 7.3 | 8：00-23：00 | 新的change | 完成 |
| 7.4 | 8：00-23：00 | 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 |
| 时间不够用 | 通宵 |
| 时间不够用 | 通宵 |
| 时间不够用 | 通宵 |
| 对于依据某个类的特性进行对类的排序不会 | 选择排序，每次选中最小的类的特性的值，将该类放入一个List中，最后返回List |
| 排班系统的自动编排怎么自动不是很会 | 随机生成一个范围内的数，然后这个数即为持续时间，得到该时间段结束的年月日 |
| 员工信息排入排班后，删掉员工之后一些操作有bug | 得先删除排班信息，再删除员工信息 |
| 操作系统中挂起一个程序，在后续选在的时候仍然处在挂起的程序那里 | 不断更新进程 |
| 排课表的时候不知道怎么表达每个时间段 | 将一周分为35个时间段，交给用户选择 |
| 排课表的时间转换 | 阅读相关资料 |

# 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

## 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编程复用性高，面向应用编程复用性低。好处很多。省略很多代码。

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？

提高代码的可读性，抽象出类的具体。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？

要是时间在充裕一点就好了

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？

一种隐秘的编程方式。提高私密性。

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？

开始的抽象化和调用关系。以及团队合作。

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？

最开始设计好思路和框架，后期一改则动全身。

1. 关于本实验的工作量、难度、deadline。

工作量太大，难度太高，deadline太紧凑

1. 到目前为止你对《软件构造》课程的评价。

一门很好的课，可惜压缩了学时，可惜跟计算机系统一个学期，很忙。