

**2024年春季学期  
计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 蓝鸿翔 |
| 学号 | 2022110504 |
| 班号 | 2223702 |
| 电子邮件 | [2242962302@qq.com](mailto:2242962302@qq.com) |
| 手机号码 | 15970911569 |

**目录**

[1. 实验目标概述 1](#_Toc167386755)

[2. 实验环境配置 1](#_Toc167386756)

[3. 实验过程 1](#_Toc167386757)

[1. 待开发的三个应用场景 1](#_Toc167386758)

[2. 面向可复用性和可维护性的设计：IntervalSet<L> 1](#_Toc167386759)

[1. IntervalSet<L>的共性操作 1](#_Toc167386760)

[2. 局部共性特征的设计方案 1](#_Toc167386761)

[3. 面向各应用的IntervalSet子类型设计（个性化特征的设计方案） 1](#_Toc167386762)

[3. 面向可复用性和可维护性的设计：MultiIntervalSet<L> 2](#_Toc167386763)

[1. MultiIntervalSet<L>的共性操作 2](#_Toc167386764)

[2. 局部共性特征的设计方案 2](#_Toc167386765)

[3. 面向各应用的MultiIntervalSet子类型设计（个性化特征的设计方案） 2](#_Toc167386766)

[4. 面向复用的设计：L 2](#_Toc167386767)

[5. 可复用API设计 3](#_Toc167386768)

[1. 计算相似度 3](#_Toc167386769)

[2. 计算时间冲突比例 3](#_Toc167386770)

[3. 计算空闲时间比例 3](#_Toc167386771)

[6. 应用设计与开发 4](#_Toc167386772)

[1. 排班管理系统 4](#_Toc167386773)

[2. 操作系统的进程调度管理系统/课表管理系统 4](#_Toc167386774)

[7. 基于语法的数据读入 4](#_Toc167386775)

[4. 实验进度记录 5](#_Toc167386776)

[5. 实验过程中遇到的困难与解决途径 5](#_Toc167386777)

[6. 实验过程中收获的经验、教训、感想 6](#_Toc167386778)

[1. 实验过程中收获的经验和教训 6](#_Toc167386779)

[2. 针对以下方面的感受 6](#_Toc167386780)

# 实验目标概述

本次实验覆盖课程第 4-12 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：

⚫ 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）

# 实验环境配置

在给定的classroom https://classroom.github.com/a/ZAJ8w2eC中创建git仓库，新建java项目与仓库建立连接。

<https://github.com/ComputerScienceHIT/HIT-Lab2-2022110504>（HIT-Lab2-202210504）

# 实验过程

## 待开发的三个应用场景

第一个应用：值班表管理，要求每天都有人值班、一人只能在一个时间段值班、值班时间无重叠

第二个应用：操作系统进程调度管理，进程不能有重叠、可以有闲置时间、一个进程可在多个时间段运行

第三个应用：大学课表管理，课程安排以周为一个周期、一周可以有多节课、一个时间段可以有多门课、一个老师可以有多门课

## 面向可复用性和可维护性的设计：IntervalSet<L>

### IntervalSet<L>的共性操作

1. Insert：插入一个label为其安排时间段，同时检查时间段是否有效，end>start?，除此之外，还判断label是否已经出现过，若出现过则抛出异常

for (L l : this.commonintervalset.keySet()){

if(l.equals(label)){

throw new IllegalArgumentException("该label已经存在!");

}

}

1. Labels：获取整个表中的所有label，
2. Remove：删除一个label及其相关的所有时间段，通过遍历整个数据的label找到后进行删除、否则返回false
3. Start：获取一个label的起点，若不存在则返回-1
4. End：获取一个label的终点，若不存在则返回-1
5. InputStartEndTime：输入整个活动的开始及结束时间，并判断输入是否合法，若end<start则返回false
6. GetStart：获取整个活动开始时间，若输入过则返回，否则返回-1
7. GetEnd：获取整个活动结束时间，若输入过则返回，否则返回-1
8. toString：时间表的打印，循环整个Map按照格式加入。

public String toString(){

StringBuilder s = new StringBuilder();

s.append("label"+" 时间段\n");

for (Map.Entry<L,ArrayList<Long>> entry :this.commonintervalset.entrySet()){

s.append(entry.getKey()).append(" ").append(entry.getValue()).append("\n");

}

return s.toString();

}

### 局部共性特征的设计方案

对不重叠性采用了装饰器的实现方案public class NonOverlapIntervalSet<L> extends IntervalSetDecorator<L>，可通过IntervalSet<L> intervalSet = new NonOverlapIntervalSet<L>(new CommonIntervalSet<L>());创建，将检查不重叠性放入checkRep()中每进行一次操作就进行检测，确保全程不会重叠，一旦发现重叠则抛出异常，在处理基本操作时均采用父类的操作。

对于无空闲性，我新建了一个类NoBlankIntervalSet，在其中我在IntervalSet基础上添加了IsInputOver方法用来标记是否完成输入，在确认完成输入后可以进行检测表中是否无空闲，

if(flag){

checkEmpty();

}

其他基本方法使用委托用IntervalSet帮助实现。

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

DutyIntervalSet（值班表）是通过

NoBlankIntervalSet<L> set = new NoBlankIntervalSet<L>(new NonOverlapIntervalSet<L>(new CommonIntervalSet<L>()));

来构建一个无空闲、不重叠的时间表，将其中的基础方法委托给NoblankInteralSet来实现，主要特征化完成toString及其他特征方法。

toString使用按天进行遍历，依次输出对应的值班人员。

public String toString() {  
 StringBuilder stringBuilder = new StringBuilder();  
 stringBuilder.append("日期\t\t").append("值班人名字\t").append("职位\t\t").append("手机号码\n");  
 long max = 0, min = Long.*MAX\_VALUE*;  
 // 找出时间区间  
 for (L l : set.labels()) {  
 long e = set.end(l);  
 long s = set.start(l);  
 if (max < e) {  
 max = e;  
 }  
 if (min > s) {  
 min = s;  
 }  
 }  
 // 将最小值和最大值设置为整天的时间戳  
 min = truncateToDay(min);  
 max = truncateToDay(max) + 86399999L; // 一天结束的时间戳（午夜前一秒）  
 SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  
 Date currentDate = new Date(min);  
 while (currentDate.getTime() <= max) {  
 stringBuilder.append(dateFormat.format(currentDate)).append("\t\t");  
 stringBuilder.append(findLabel(currentDate).toString()).append("\n");  
 currentDate = addDays(currentDate, 1);  
 }  
 return stringBuilder.toString();  
}

## 面向可复用性和可维护性的设计：MultiIntervalSet<L>

### MultiIntervalSet<L>的共性操作

数据结构的实现：ArrayList<IntervalSet<L>> multiintervalset =new ArrayList<>();通过嵌套一层list完成多个表的重叠

1. Insert：插入一个label并为其安排时间段，插入前先判断end是否大于start，再按照时间表依次往下插入。

public void insert(long start, long end, L label) throws IllegalArgumentException{

if(end<start){

throw new IllegalArgumentException("时间段输入错误");

}

for(IntervalSet<L> set : this.multiintervalset){

boolean flag = false;

//寻找每个表中对应标签

for(L l : set.labels()){

//找到则跳往下一个时间表

if(l.equals(label)){

flag = true;

break;

}

}

//没找到则在时间表中插入新标签

if (!flag){

set.insert(start,end,label);

return;

}

}

//如果全都有对应标签

IntervalSet<L> set = new CommonIntervalSet<L>();

set.insert(start,end,label);

this.multiintervalset.add(set);

}

1. Labels：获取整个表中的所有出现的label，在第一层就包括了所有的label，只需调用第一层的labels即可。
2. Remove：删除一个label及其相关的所有时间段，遍历每一层时间表，每层都调用remove，只要进行了删除操作则返回true。
3. Intervals：将一个label的所有时间段获取为一个label为Integer类型的Interval。遍历每一层时间表，将每一层中找到的时间段加入。

public IntervalSet<Integer> intervals(L label) {

IntervalSet<Integer> set = new CommonIntervalSet<Integer>();

int time=0;

for(IntervalSet<L> s : this.multiintervalset){

for (L l : s.labels()){

if(l.equals(label)){

set.insert(s.start(label),s.end(label), time);

time++;

}

}

}

return set;

}

1. InputStartEndTime：输入整个活动的开始及结束时间，并判断输入是否合法，若end<start则返回false
2. GetStart：获取整个活动开始时间，若输入过则返回，否则返回-1
3. GetEnd：获取整个活动结束时间，若输入过则返回，否则返回-1
4. toString：MultiIntervalSet打印，对每个label依次寻找其时间区间，然后按照格式将其加入字符串中。

public String toString(){

StringBuilder s = new StringBuilder();

s.append("label"+" 时间段\n");

for (L l : this.labels()) {

IntervalSet<Integer> set = this.intervals(l);

StringBuilder time = new StringBuilder();

for (int i = 0;i<set.labels().size();i++){

time.append("[").append(set.start(i)).append(", ").append(set.end(i)).append("] ");

}

s.append(l).append(" "+time+"\n");

}

return s.toString();

}

### 局部共性特征的设计方案

周期性：使用全新的数据结构来储存时间段，使其带有周期特性

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

对insert进行重载，可添加其周期属性，SetPeriodic可对已存在的时间段添加周期，对不存在的时间段则会先加入时间段再添加周期。使用check Periodicity可实现对某时间段周期性地检查。

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

ProcessSet(进程表)：MultiIntervalSet<L> multiIntervalSet = new NoOverlapMultiIntervalSet<L>(new CommomMultilntervalSet<L>());

构建一个无重叠的多区间的时间表，时间表的基本操作均是委托给NoOverMultiIntervalSet来实现，特殊地实现toString方法

public String toString() {

StringBuilder s = new StringBuilder();

s.append("进程"+" 时间段\n");

for (L l : this.labels()) {

IntervalSet<Integer> set = this.intervals(l);

StringBuilder time = new StringBuilder();

for (int i = 0;i<set.labels().size();i++){

time.append("[").append(set.start(i)).append(", ").append(set.end(i)).append("] ");

}

s.append(l).append(" "+time+"\n");

}

return s.toString();

}

## 面向复用的设计：L

L可代表任意immutable类型数据作为标签进行复用。这些元素可以在不同的程序或系统的多个部分中被重复使用，从而提高代码的效率、减少开发时间和维护成本。

## 可复用API设计

### 计算相似度

按照时间（按天为单位）进行遍历

对于multiIntervalSet1中的每个标签，检查其对应的时间区间是否包含当前时间。如果包含，则将该标签添加到similarSet1中。

对multiIntervalSet2执行相同的操作，将符合条件的标签添加到similarSet2中。

遍历similarSet1中的每个标签，检查similarSet2中是否包含相同的标签。

如果存在至少一个相同的标签，则将intervalSimilarity设置为oneDayInMillis（即当前时间段的总时长，表示两个集合在该时间段内完全相似）。

将intervalSimilarity累加到totalSimilarity中。

将totalSimilarity除以totalTime即可得到重叠时间比。

### 计算时间冲突比例

从IntervalSet集合的最早开始时间开始，以oneDayInMillis（表示一天的毫秒数）为步长，遍历到最晚结束时间。对于每个时间点，遍历IntervalSet集合中的所有标签。如果一个标签的时间区间覆盖了当前时间点（即该时间点在标签的起始时间和结束时间之间），则conflict计数加1。如果在某个时间点上的conflict计数大于1（即存在多于一个的时间区间覆盖了该时间点），则将conflictedLength增加oneDayInMillis（即一天的长度）。

遍历完所有时间点后，计算冲突时间段长度与总时间范围长度的比率，即conflictedLength / totalLength得到冲突时间比。

### 计算空闲时间比例

从IntervalSet集合的最早开始时间开始，以oneDayInMillis（表示一天的毫秒数）为步长，遍历到最晚结束时间。对于每个时间点，遍历IntervalSet集合中的所有标签。如果一个标签的时间区间覆盖了当前时间点（即该时间点在标签的起始时间和结束时间之间），则freetime计数加1。如果在某个时间点上的freetime计数大于0（即存在至少一个的时间区间覆盖了该时间点），则将totalFreeTime减少oneDayInMillis（即一天的长度）。遍历完所有时间点后，计算totalFreeTime与totalTime的比率，即位空闲比。

## 应用设计与开发

### 排班管理系统

将系统分为十个操作：

0：退出程序；

1：从文件读取员工安排信息；

2：插入员工信息；

3：删除员工信息；

4：删除排班表中的员工信息；

5：输入开始结束时间;

6：打印员工排班表;

7：打印目前输入的员工;

8：自动编排值班表

9：对员工安排是否结束(进行对空白的判断)");

读取文件时根据正则表达式对各个数据进行读取，在输入日期时按照”yyyy-mm-dd”的格式输入若格式不正确则打印"输入的日期格式不正确，请确保格式为yyyy-MM-dd"并将其转为微秒存于排班表中。

在删除员工信息前先检查是否排班表中有该员工，如果有则提示"员工正在排表中"，没有则直接调用remove将其删除。

自动编排值班表前先将排班表置空再进行随机编排。再随机编排的过程中，要将区间划分为n个小区间是通过将一个从start到end的区间随机划分为n个子区间，并确保这些子区间的总长度等于原始区间的长度。首先，它计算整个区间的长度totalLength，并预留每个子区间至少一个单位长度，然后根据剩余长度remainingLength随机分配长度给n-1个子区间。如果没有剩余长度，则每个子区间的大小相等。对于随机分配的部分，代码使用一个随机数生成器Random的nextLong方法来确保每个区间长度（包括至少一个单位长度和随机分配的部分）之和不超过剩余长度和当前位置到end的长度之间的较小值。最后，它添加了一个从上一个子区间结束位置到end的最后一个子区间，从而完成了整个区间的划分。

再确认输入完毕后每次操作都会对排班表的空闲进行检查  
if(flag){

checkEmpty();

}

### 操作系统的进程调度管理系统/课表管理系统

操作系统的进程调度管理具有以下操作

0：退出程序；

1:输入进程

2:随机安排进程并执行

3:最短进程优先安排进程并执行

4:删除进程

5:显示进程执行情况

6:显示输入的进程

在输入进程时，使用了一个无限循环，输入n退出。首先提示用户输入进程的ID、名称、最小运行时间和最大运行时间。然后，根据用户输入的信息，创建一个Process对象，并将该对象与一个布尔值false（表示进程当前未运行的状态）一起作为键值对存入processes这个映射。接着，代码询问用户是否继续输入下一个进程的信息。如果用户输入y（不区分大小写）表示继续，则循环继续；如果用户输入n表示不再继续，则循环终止。整个过程允许用户连续输入多个进程的信息，直到用户选择不再继续为止。

随机安排进程时，使用makeRandom方法首先，它获取了进程的最大运行时间nextTime。然后，通过查询processProcessIntervalSet中该进程已经占用的时间区间（使用intervals方法），并遍历这些区间，从nextTime中减去这些已占用时间段的长度。接下来，代码创建了一个Random对象用于生成随机数，并计算出一个起始时间start，这个时间是基于当前时间now加上一个0到100000之间的随机毫秒数（取绝对值确保时间不会倒退）。然后，代码又计算了一个结束时间end，这个时间是基于start时间加上一个0到剩余nextTime时间长度之间的随机毫秒数（同样取绝对值）。之后，代码检查生成的end时间是否超出了进程的最小运行时间限制。如果end时间减去start时间大于进程的最大运行时间减去其最小运行时间，那么它将processes映射中该进程的状态设置为true。最后，代码将生成的start和end时间区间以及对应的Process对象插入到processProcessIntervalSet中，并返回结束时间end。通过这个方法，可以实现对一个进程的随机划分，再对所有未执行的进程进行划分直到全部完成划分。

在进行最短进程优先划分时添加了一个方法寻找最大执行时间最短的进程findMinTime。

private static Process findMinTime(){

long min = 9223372036854775807L;

Process p = new Process();

for (Process process : processes.keySet()){

long time = process.getMaxRunningTime();

IntervalSet<Integer> intervalSet = processProcessIntervalSet.intervals(process);

for (Integer i : intervalSet.labels()){

time -= intervalSet.end(i) - intervalSet.start(i);

}

if (time<min&&!processes.get(process)){

min = time;

p = process;

}

}

return p;

}

在删除进程时，首先，程序输出提示用户输入要删除的进程的ID，并从用户输入中读取这个ID。然后，它遍历processProcessIntervalSet（中的标签（即进程），并检查每个进程的ID是否与用户输入的ID相匹配。如果找到了匹配的进程ID，程序会输出一条消息表明该进程已经被执行，打印"该进程已经被按执行."并将变量f设置为true以表示找到了匹配的进程。之后，由于找到了匹配的进程，它会立即跳出第一个循环。如果f为true（即进程已经执行），程序会跳出case。然后，程序会进入第二个循环，它遍历processes的键集（即进程对象）。在这个循环中，它再次检查每个进程的ID是否与用户输入的ID相匹配。如果找到了匹配的进程，程序会从processes中移除这个进程，并输出一条消息表明该进程已被删除。之后，它会立即跳出这个循环。如果processProcessIntervalSet和processes中都没有找到与用户输入的ID匹配的进程，那么程序将不会输出任何关于进程已被删除的消息。

toString：先将标题加入s中"进程"+" 时间段\n"，再对set进行遍历依次打印process和对应时间段。全部按照一定格式排列。

要实现可视化我使用了JavaFX来实现具体实现在ProcessVisualizer中

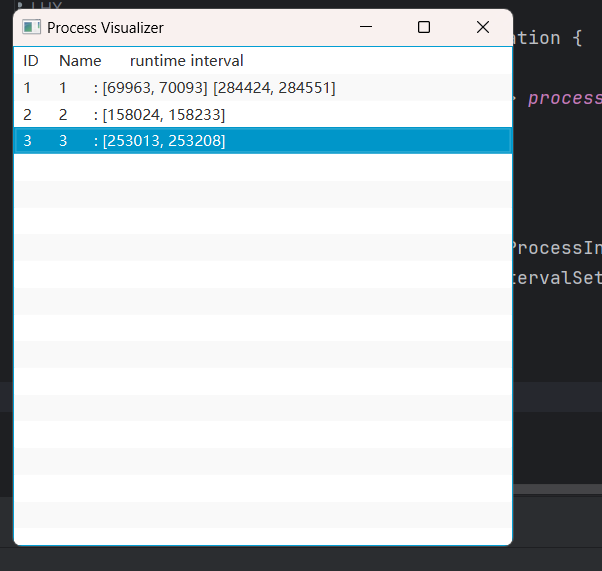
ProcessVisualizer 实现了以下功能：

1. 显示进程执行情况：在用户选择操作 5 时，程序会调用 ProcessVisualizer，该类会将当前所有进程的执行时间段可视化展示出来。

2. 可视化进程执行情况：ProcessVisualizer 类将进程的执行时间段以可视化的形式展示出来，使用户能够清晰地了解每个进程的执行情况。

3. 实时更新：在每次调用 ProcessVisualizer 时，它会检查是否已经启动过，如果已经启动，则会刷新当前的可视化界面，以便用户能够实时获取最新的进程执行情况。

4. 多线程启动：为了避免阻塞主程序，ProcessVisualizer 在启动时会创建一个新的线程来运行，确保主程序能够继续执行其他操作。



可视化界面

## 基于语法的数据读入

利用下面三个正则表达式来获取员工、活动开始结束、值班信息，并将这些信息存入表中。

EMPLOYEE\_PATTERN = Pattern.compile("([A-Za-z]+)\\{(\\w+(?:\\s+\\w+)\*),([0-9\\-]+)}");

([A-Za-z]+)：匹配一个或多个英文字母，并将其捕获为一个组（即员工的名字）。

\\{ 和 \\}：匹配大括号 {}。在Java字符串中，反斜杠 \ 是转义字符，因此需要使用两个反斜杠 \\ 来表示一个。

(\\w+(?:\\s+\\w+)\*)：基本上它是用来匹配由单词和空格组成的字符串（即员工的ID或标识）。这里 \\w+ 匹配一个或多个单词字符（字母、数字、下划线），(?:...) 是一个非捕获组，它里面的 \\s+ 匹配一个或多个空白字符，\\w+ 再次匹配单词字符。整个非捕获组后面跟着一个 \* 表示它可以出现零次或多次。但注意这里的 (?:...) 实际上没有捕获功能，可能您想要的是使用普通的捕获组 (\\w+(\\s+\\w+)\*)。

,：匹配逗号。

\\s\*：匹配零个或多个空白字符（如空格、制表符等），这允许在逗号和日期之间有空格。

([0-9-]+)：匹配一个或多个数字或短横线 -（假设日期格式允许短横线作为分隔符），并将其捕获为一个组（即员工的日期或编号）。

PERIOD\_PATTERN = Pattern.compile("Period\\{(\\d{4}-\\d{2}-\\d{2}),(\\d{4}-\\d{2}-\\d{2})}");

Period\\{ 和 \\}：匹配文本 "Period{" 和 "}"。

(\\d{4}-\\d{2}-\\d{2})：匹配形如 "YYYY-MM-DD" 的日期字符串，并将其捕获为一个组（即开始日期）。

,：匹配逗号。

(\\d{4}-\\d{2}-\\d{2})：再次匹配形如 "YYYY-MM-DD" 的日期字符串，并将其捕获为一个组（即结束日期）。

Pattern ROSTER\_ENTRY\_PATTERN = Pattern.compile("([\\w+)\\{(\\d{4}-\\d{2}-\\d{2}),(\\d{4}-\\d{2}-\\d{2})}](file:///\\w+)\\%7b(\\d%7b4%7d-\\d%7b2%7d-\\d%7b2%7d),(\\d%7b4%7d-\\d%7b2%7d-\\d%7b2%7d)%7d)");

(\\w+)：匹配一个或多个单词字符，并将其捕获为一个组（可能代表员工的ID或某种标识）。

\\{ 和 \\}：匹配大括号 {}。

接下来的部分与 PERIOD\_PATTERN 相同，分别匹配开始和结束日期，并将它们捕获为组。

# 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 |
| 2024/4/25 | 20：00-22：00 | 完成接口和框架的搭建、以及文件的结构构建 | 完成但有欠缺 |
| 2024/4/27 | 15：00-17：00 | 完成IntervalSet中CommonIntervalSet功能 | 完成 |
| 2024/5/2 | 19：00-21：00 | 完成MultiIntervalSet的构造，CommonIntervalSet代码完成 | 初步完成 |
| 2024/5/7 | 20：00-23：00 | 完成IntervalSet不重叠、无空白的装饰 | 完成 |
| 2024/5/11 | 15：00-16：00 | 完成应用一的set及app | 完成 |
| 2024/5/16 | 19：00-22：00 | 完成MultiIntervalSet周期性的扩展，并完成应用二app | 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 |
| 在构造无空白set时需要判断何时结束输入。 | 将NoBlankIntervalSet定义为新的类，其他基本操作通过委托给CommonIntervalSet实现，加入IsInputOver方法判断是否输入完毕 |
| 在对周期性进行构造时，如何完成周期性的标志。 | 在周期性表中定义新的数据类型，来储存周期性的特性。 |
| 在API计算相似度等属性时，需要对时间遍历，若对long进行遍历时间过长。 | 将时间按天进行遍历，每次按天为单位进行计算。 |

# 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

在项目的构建过程中，我深刻体会到考虑项目的可复用性和扩展性的重要性。在项目初期，对项目的需求和特点进行详尽的分析至关重要，这不仅能帮助我们理解项目的核心价值和预期目标，还能引导我们思考如何通过设计可复用的类和模块来减少未来的维护成本。同时，当面对需要增加新功能或修改现有功能时，扩展性好的设计能让我们更加灵活和高效地应对这些变化。

具体到类的扩展，我们需要仔细权衡不同的扩展策略，如继承、接口实现或组合等，以选择最适合当前项目场景的方式。每种扩展方式都有其优缺点，比如继承可能带来层次过深的问题，而接口则可能增加实现的复杂性。因此，在做出选择之前，我们需要对项目的具体需求和约束有深入的理解。

在功能测试方面，我认识到测试的全面性和细致性对于确保项目质量至关重要。我们需要根据每个功能的具体需求和预期行为，设计相应的测试用例，并对测试结果进行仔细分析。同时，为了避免遗漏，我们还需要制定详细的测试计划，确保对已实现的各种功能进行全面的测试。

最后，对于应用功能的解读，我意识到全面而准确的理解是完成应用的关键。我们需要仔细阅读相关文档，理解每个功能的输入、输出和预期行为，并在实际使用中不断验证和修正我们的理解。只有这样，我们才能确保应用的正确性和可靠性。

## 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（抽象数据类型）的编程与直接面向应用场景编程的主要差异在于抽象层次和复用性。面向ADT的编程强调将数据和操作抽象化，以降低与具体实现和编程语言的耦合度，从而提高代码的可读性、可维护性和复用性。这种编程方式要求开发者对数据结构及其操作有深入的理解，并能从更高的抽象层次思考问题。而直接面向应用场景编程则更侧重于解决具体问题，编程思路直接针对具体需求，虽然简单直观，但复用性较低，当需求变化时可能需要较大的修改量。因此，面向ADT的编程在应对复杂系统或需要高复用性的场景时更为优势。

复用能显著提高开发效率，减少错误，促进知识的共享和传承。它使开发者能够利用已有的成果，快速构建新功能，加速项目进程，降低维护成本，增强系统的稳定性和可靠性。

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？

为ADT撰写复杂的规约（specification）、不变量（invariants）、表示不变性（RI）、抽象函数（AF），并时刻注意ADT是否有表示泄漏（rep exposure），这些工作的意义在于确保程序的正确性、健壮性和可维护性。

我认为是非常有必要在以后的编程中坚持下去。虽然这些工作可能会增加一些开发成本，但它们能够显著提高代码的质量和可维护性。在长期的项目开发和维护过程中，这些好处将远远超过初期的投入成本。因此，我会坚持在编程中遵循这些原则，以确保我的代码是健壮、正确且易于维护的。

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？

从将别人提供的API用于自己的程序开发，转变到尝试开发给别人使用的API，这一过程中确实能够深刻体会到其中的难处和乐趣。

开发API需要创造新的解决方案，解决用户的问题，这是一个非常有趣和富有挑战性的过程。当看到其他人使用我开发的API，并因此提高了他们的工作效率或解决了他们的问题时，会感到非常有成就感。

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？

在开发解析器并使用语法和正则表达式去解析输入文件以构造对象的过程中，我深刻体会到了语法驱动编程的精确性和强大性。这种编程方式要求我们对输入数据的结构有深入的理解，并能够将其转化为明确的语法规则。虽然这增加了开发的复杂性和初始的学习成本，但一旦完成，解析器就能准确、高效地处理大量输入数据，并且易于维护和扩展。

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？

在设计ADT的过程中，难度主要体现在几个方面。首先，对问题领域的深入理解和抽象是关键，我需要确保我的ADT设计能够准确地反映实际需求，同时保持足够的通用性和灵活性。这需要对问题有全面的了解，并考虑各种可能的边界情况和异常情况。其次，设计合适的接口和类层次结构也是一项挑战，我需要确保接口简洁易用，同时类之间的继承、聚合和关联关系清晰合理。

为了克服这些困难，我积极与同学交谈，寻求帮助，上网查找资料，学习相关知识利用最终构造合适的继承树，并成功地完成了实验任务。

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？

在利用“接口、抽象类、类”三层体系以及接口的组合、类的继承、设计模式等技术来完成最大程度的抽象和复用时，我深刻体会到平衡抽象与具体实现的重要性。过度抽象可能导致接口过于庞大和复杂，难以理解和使用；而抽象不足则可能导致代码冗余和难以维护。我的经验教训是，在设计阶段，需要仔细分析需求，明确抽象的边界，尽量将共性的部分抽象成接口或抽象类，而将具体的实现细节留给子类。同时，要善用设计模式来指导代码结构的设计，如使用工厂模式来管理对象的创建，使用策略模式来灵活切换算法等。这样既能保证代码的复用性和扩展性，又能使代码结构清晰易懂。

1. 关于本实验的工作量、难度、deadline。

本实验工作量相对较大，难度相对较大，及时完成实验。

1. 到目前为止你对《软件构造》课程的评价。

《软件构造》课程不仅是一门至关重要的课程，它更是一次全面的软件开发思维的洗礼。通过这门课程，我深入理解了软件过程的三个核心维度：代码管理、抽象数据类型（ADT）设计和面向对象编程（OOP）。课程详细阐述了版本控制的重要性，教会了我如何运用Git等工具来有效管理代码变更和团队协作。在ADT设计方面，我学习到了如何从实际问题中抽象出关键概念和操作，进而构建出高效、可复用的数据结构。而在OOP方面，课程强调了封装、继承和多态等核心概念，指导我如何构建模块化、可扩展的软件系统。此外，课程还涵盖了设计模式、异常处理、测试等关键技能，为我未来的软件开发之路奠定了坚实的基础。

哈哈哈