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	张瑞
学号	1190201421
班号	1936603
电子邮件	1190201421@stu.hit.edu.cn
手机号码	15736059288

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：

- 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），并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

本次实验的环境之前已经搭建好，此处不再赘述。

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

https://github.com/ComputerScienceHIT/HIT-Lab3-1190201421

3 实验过程

3.1 待开发的三个应用场景

本次实验需要为三个应用场景开发相关程序，三个场景有较大的相似性，均为“在特定时间段进行特定任务”，分别为值班表管理、操作系统进程调度管理和课表管理。

三个应用都涉及到对一个个时间段的抽象，为了可复用性和可维护性，可以构造一套统一的 ADT。但是三个应用在个别细节上又有差别，例如值班表管理要求不能有重复时间段、不同时间段标签不能一样、整体时间段无空白；操作系统进程调度管理要求不能有重复时间段、不同时间段标签可以一样；课表管理要求有不同标签的时间段可以重叠、各个时间段周期性重复……这些异同要求合理设计继承树，将共性的东西写到上层、个性的东西写入下层。

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

将 IntervalSet 写成接口类型，定义 6 个共性的方法：empty、insert、remove、labels、start 和 end。完成各个方法的 spec 之后，撰写测试策略并设计对应测试用例。接下来完成它的一个实现类 CommonIntervalSet, 设计它的 rep、RF、AI、Safety from Rep Exposure，实现各个接口方法、checkRep()和 toString()。注意 IntervalSet 中各个时间段的标签不可重复。

3.2.2 局部共性特征的设计方案

IntervalSet 主要是为了实现值班表管理的应用，最终我采用的是方案 6 的 decorator 设计模式。首先设计一个 IntervalSetDecorator 类来实现 IntervalSet 接口，实际这个类实现了一个 delegation 的功能，对 IntervalSet 的功能未作改变。然后设计 NoBlankIntervalSet 和 NonOverlapIntervalSet，均继承 IntervalSetDecorator，分别使得 IntervalSet 具有了检查空白和不重叠的特性。

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）

主要是值班表管理用到了 IntervalSet，通过上一节提到的 decorator 方案，将一个 IntervalSet 装饰上两层“外衣”，使得其增加检查空白和不重叠的特性，这就得到了 DutyIntervalSet。

注意 NoBlankIntervalSet 增加了 IntervalSet 没有的新特性 checknoblank，所以在装饰的时候被放在了最外层，且在使用时需声明类型为 NoBlankIntervalSet。

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

我直接把 MultiIntervalSet 实现为了具体的类，而没有写接口，并且在 rep 中用到了 IntervalSet。MultiIntervalSet 主要的方法有 insert、labels、remove、intervals，注意其有两种构造方法，一种不传入参数，一种传入一个 IntervalSet 进行初始化。同样的，完成各个方法的 spec 之后，撰写测试策略并设计对应测试用例。接下

来设计它的 rep、RF、AI、Safety from Rep Exposure，实现各个方法、checkRep() 和 toString()。注意 MultiIntervalSet 不同于 IntervalSet，各个时间段的标签可重复。

值得一提的是，我在测试用例中遇到如下报错，无法通过：

```
//Cover: the interval of the given label already exists, the number of intervals > 1, the given label is not null
@Test
public void testIntervalsMorethanOne() {
    MultiIntervalSet<String> intervals = new MultiIntervalSet<>();
    intervals.insert(1, 2, "A");
    intervals.insert(2, 3, "A");
    intervals.insert(3, 4, "A");
    IntervalSet<Integer> inter = IntervalSet.empty();
    inter.insert(1, 2, 0);
    inter.insert(2, 3, 1);
    inter.insert(3, 4, 2);
    assertEquals(inter, intervals.intervals("A"));
}
```

查看结果明明是一样的，让我困惑了好久：

```
java.lang.AssertionError: expected: IntervalSet.CommonIntervalSet<Intervals :
1 -> 2 : 0
2 -> 3 : 1
3 -> 4 : 2
> but was: IntervalSet.CommonIntervalSet<Intervals :
1 -> 2 : 0
2 -> 3 : 1
3 -> 4 : 2
>
at MultiIntervalSet.MultiIntervalSetTest.testIntervalsMorethanOne(MultiIntervalSetTest
at java.util.stream.ForEachOps$ForEachOp$OfRef.accept(ForEachOps.java:184)
at java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:193)
```

后来突然想到应该是 IntervalSet 没有重写 equals 方法导致在 assertequals 时虽然内容相同，但由于指向的地址不同，结果不同。于是采用了迂回战术，用 IntervalSet 的 observer 方法验证其相等：

```
//Cover: the interval of the given label already exists, the number of
@Test
public void testIntervalsMorethanOne() {
    MultiIntervalSet<String> intervals = new MultiIntervalSet<>();
    intervals.insert(1, 2, "A");
    intervals.insert(2, 3, "A");
    intervals.insert(3, 4, "A");
    IntervalSet<Integer> interval = intervals.intervals("A");
    IntervalSet<Integer> inter = IntervalSet.empty();
    inter.insert(1, 2, 0);
    inter.insert(2, 3, 1);
    inter.insert(3, 4, 2);
    for (Integer i : inter.labels()) {
        assertEquals(inter.start(i), interval.start(i));
        assertEquals(inter.end(i), interval.end(i));
    }
}
```

3.3.2 局部共性特征的设计方案

MultiIntervalSet 主要是为了实现操作系统进程管理和课表管理的应用，最终我采用的也是类似方案 6 的 decorator 设计模式。首先设计一个 MultiIntervalSetDecorator 类来继承 MultiIntervalSet 类（虽然不同于一般 decorator 设计模式的“实现一个接口”的模式，但是思想上仍然是相同的），实际上这个类也是实现了一个 delegation 的功能，对 MultiIntervalSet 的功能未作改变。然后设计 PeriodicMultiIntervalSet 、OverlapMultiIntervalSet 和

NonOverlapMultiIntervalSet，均继承 MultiIntervalSetDecorator，分别使得 MultiIntervalSet 具有了周期性重复、允许不同标签时间段重叠和不允许重叠的特性。

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）

将一个 MultiIntervalSet 装饰上 NonOverlapMultiIntervalSet，使其增加不重叠的特性，这就得到了 ProcessIntervalSet，用于操作系统进程管理。

将一个 MultiIntervalSet 装饰上 OverlapMultiIntervalSet 和 PeriodicMultiIntervalSet，使其增加允许不同标签时间段重叠和周期性重复的特性，这就得到了 CourseIntervalSet，用于课表管理。

注意 PeriodicMultiIntervalSet 增加了 MultiIntervalSet 没有的新特性 getperiod，所以在装饰的时候被放在了最外层，且在使用时需声明类型为 PeriodicMultiIntervalSet。

3.4 面向复用的设计：L

三个应用场景中对时间段进行标识的标签分别为 Employee、Process 和 Course，均为 immutable 的类。分析清楚各自需要的属性，很快便能实现。

3.5 可复用 API 设计

3.5.1 计算相似度

计算两个 MultiIntervalSet 的相似度时，采用循环嵌套的遍历方法，计算重叠时间段的长度。将两个 MultiIntervalSet 中的时间段一一拿出来对比，对重叠且标签相同的四种情况逐一分析即可。我的实现中，对于同一个 MultiIntervalSet 的同一个时间段有多个标签的情况，会将两个标签都各自与另一个 MultiIntervalSet 进行对比，所以最终结果可能有大于 1 的情况（因为实验手册也没详说这里的处理，而且没有限制结果为 0-1 之间的值）。

```
if (label1.equals(label2) && start1 <= start2 && start2 <= end1 && end1 <= end2)
    similar += end1 - start2 + 1;
else if (label1.equals(label2) && start2 <= start1 && end1 <= end2)
    similar += end1 - start1 +1;
else if (label1.equals(label2) && start2 <= start1 && start1 <= end2 && end2 <= end1)
    similar += end2 - start1 +1;
else if (label1.equals(label2) && start1 <= start2 && end2 <= end1)
    similar += end2 - start2 +1;
```

3.5.2 计算时间冲突比例

计算时间冲突比例分别要对 IntervalSet 和 MultiIntervalSet 实现，两者采用相同的思路。先遍历确定总时间段的开始和结束，将冲突时间初始化为 0，然后在总时间段上，以长度为 1 的时间间隔，逐个检查是否有两个及以上的时间段将其覆盖，若有，则冲突时间加 1，一直到检查完整个时间段。

3.5.3 计算空闲时间比例

计算空闲时间比例同样也要对 IntervalSet 和 MultiIntervalSet 分别实现，思路同计算冲突比例的思路。同样是先遍历确定总时间段的开始和结束，将空闲时间初始化为总时间段长度，然后在总时间段上，以长度为 1 的时间间隔，逐个检查是否有时间段将其覆盖，若有，则空闲时间减 1，一直到检查完整个时间段。

3.6 应用设计与开发

3.6.1 排班管理系统

排班管理系统为 DutyRosterApp，里面主要利用前面已设计好的 DutyIntervalSet，实现设定排班开始和结束时间、增删员工、手动和随机排班以及展示排班表的功能。

其中涉及到时间计算的部分，在同学提醒下上网学习了关于 LocalDate 类的基本知识，特别方便的实现了。随机排班这一功能，我是将总的排班天数平均给了在名单上的所有员工，除完员工数得到的余数排给最后一个被安排的人。

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为 ProcessScheduleApp，里面主要利用前面已设计好的 ProcessIntervalSet，实现设定进程信息、随机选择进程和最短进程优先的处理策略以及展示进程调度结果的功能。

值得一提的是，我在一段代码的处理上出现了问题，最后发现是情况讨论不周到导致的。因为我的实现中，认为标签相同的进程不会直接相邻，要么被闲置时间段打断，要么被另外一个进程打断。又因为每次的执行时间是随机的，可能会出现一个进程尚未执行结束，而此次执行时间已到，也没能出现闲置时间段的情况，这时候，程序会陷入死循环无法继续。欠考虑代码如下：

```
if (rtime > p1.getlongesttime()) {
    rtime = p1.getlongesttime();
}
if (rtime + pi.get(p1) > p1.getlongesttime()) {
    rtime = p1.getlongesttime() - pi.get(p1);
}
pprocess.insert(start, start + rtime - 1, p1);
```

最终发现问题，将代码修改，覆盖了更全面的情况，解决死循环问题：

```
if (rtime > p1.getlongesttime()) {
    rtime = p1.getlongesttime();
}
if (rtime + pi.get(p1) > p1.getlongesttime()) {
    rtime = p1.getlongesttime() - pi.get(p1);
}
if (rtime + pi.get(p1) < p1.getshortesttime() && pp.size() == 1) {
    rtime = p1.getshortesttime() - pi.get(p1);
}
```

此外，在本应用中的随机——执行进程或闲置、执行第几个进程、执行多长时间、闲置多少时间等全用 Random 来实现，再次上网查阅了相关资料。

3.6.3 课表管理系统

课表管理系统为 CourseScheduleApp，里面主要利用 CourseIntervalSet 实现设定学期开始日期、设定总周数、增加课程、排课、查看未安排完成的课程情况、查看空白和冲突时间比例以及查看学期内任意一天课表的功能。

同样的，关于时间的部分依靠 LocalDate 能很好地解决。但是需要注意查看空白和冲突时间比例不能直接使用前面实现的两个函数，因为在前面的实现中，时间段的总长度是由 MultiIntervalSet 中已有时间段的最早和最晚时间确定的，而此处计算时候的总长度为整个周的时间段长度。此外，为了能区分周一到周日每天相同时间（如周一 8-10 时和周三 8-10 时）的时间段，我对各个时间段的起止时间做了一定调整，周一的起止时间均加上 100，周二加上 200，以此类推……这样虽然能很直观地分辨出来，但是在计算相关时间长度时要注意及时转换回来。

3.7 基于语法的数据读入

以文件形式读入数据并进行初始化，主要应用到正则表达式。上网查阅资料后，参照示例代码写出了构造排班起止时间、初始化员工名单和排班所需的正则表达式如下：

起止日期：

```
("Period\\{(\\d{4})-(\\d{2})-(\\d{2}),(\\d{4})-(\\d{2})-(\\d{2})\\}");
```

员工信息：

```
("([A-Z][a-z]+[A-Z][a-z]+)\\{([A-Z][A-Za-z\\s]+),(1(?:3\\d|4[4-9]|5[0-35-9]|6[67]|7[013-8]|8\\d|9\\d)-\\d{4}-\\d{4})\\}");
```

排班信息：

```
("([A-Z][a-z]+[A-Z][a-z]+)\\{(\\d{4})-(\\d{2})-(\\d{2}),(\\d{4})-(\\d{2})-(\\d{2})\\}");
```

在读入数据的时候，主要就是用到 Pattern 和 Matcher 以正则表达式为模板，在文件输入里面进行匹配，获取需要的信息。关于正则表达式、Pattern 和 Matcher，又是上网自学的。

获取信息的问题能解决，初始化也就简单了，和之前的方案是相似的。

3.8 应对面临的新变化

3.8.1 变化 1

初步估计要新增一个 NoBlankMultiIntervalSet 类，部分修改 DutyIntervalSet 和 DutyRosterApp，不会耗时太久。

之前的 DutyRosterApp 是用 DutyIntervalSet 实现的，而 DutyIntervalSet 实际是基于 IntervalSet 来实现的，不能允许相同标签的时间段出现。若要改变，可以基于 MultiIntervalSet 来实现。

再考虑排班管理系统的两个特性，检查空白和不重叠。后者在操作系统进程调度管理的设计中已经实现，只需考虑前者。我新建了一个 NoBlankMultiIntervalSet 类来继承 MultiIntervalSetDecorator，并设计测试用例调试其结果。再修改 DutyIntervalSet，将其变成用 NoBlankMultiIntervalSet 和 NonOverlapMultiIntervalSet 装饰的 MultiIntervalSet，并且修改对应测试用例；最后修改 DutyRosterApp 中利用 IntervalSet 特性实现的少量代码。

3.8.2 变化 2

这个变化更加简单，甚至不用新增类，只需修改 CourseIntervalSet 即可，将其装饰由 OverlapMultiIntervalSet 改变为 NonOverlapMultiIntervalSet 就完成了，几乎不用花时间。

但是为了测试用例的通过，修改完 CourseIntervalSet 之后，还要注意修改 CourseIntervalSetTest，将含有插入重叠时间段的测试用例进行微调。

3.9 Git 仓库结构

```
ZR@HUAWEIMateBookX MINGW64 /d/programming/java-latest-released/eclipse-workspace/lab3 (change)
$ git log
commit b5cf0cb2891c4b478cd12a66e96f12cb059876fa (HEAD -> change, lab3/change)
Author: ZR <1190201421@stu.hit.edu.cn>
Date:   Sun Jul 4 15:08:25 2021 +0800

    change

commit a354ae4da0317076b612b41fb11a47d707635101 (lab3/master, master)
Author: ZR <1190201421@stu.hit.edu.cn>
Date:   Sun Jul 4 13:20:27 2021 +0800

    fix some bugs

commit 932ca0946d3e96eb53fa75f7dd31be03d14004c7
Author: ZR <1190201421@stu.hit.edu.cn>
Date:   Sat Jul 3 22:35:25 2021 +0800

    Regular Expression

commit 8e907cbd33db9f6bde9ab23e37adb4bacabbb7c6
Author: ZR <1190201421@stu.hit.edu.cn>
Date:   Sat Jul 3 18:18:06 2021 +0800

    modify App

commit 0e0447584ba1548503ed43557296292a8df96260
Author: ZR <1190201421@stu.hit.edu.cn>
Date:   Sat Jul 3 15:58:54 2021 +0800

    App

commit 45daa0bb8d68b97a4a0a350316702114 38eded73
Author: ZR <1190201421@stu.hit.edu.cn>
Date:   Thu Jul 1 23:36:55 2021 +0800

    ADT & API
```

4 实验进度记录

日期	时间段	计划任务	实际完成情况
2021.6.15	13:45-15:30	阅读实验手册，完成 IntervalSet 接口设计与对应测试用例设计	完成
2021.6.22	13:45-15:30	完成 CommonIntervalSet 及对应测试用例	完成
2021.6.29	13:45-17:30	完成 MultiIntervalSet 的测试用例及实现	完成
2021.6.30	14:00-18:00	完成 IntervalSet 的 Decorator 相关实现及测试用例	未完成，在检查是否有空白问题上遇到难关，测试用例无法通过

2021.6.30	19:00-23:59	继续完成 IntervalSet，并完成部分 MultiIntervalSet 的 Decorator 相关实现及测试用例	解决 IntervalSet 的问题，完成两个 MultiIntervalSet 的 Decorator
2021.7.1	00:00-2:00	完成 MultiIntervalSet 的所有 Decorator 相关实现及测试用例	部分测试用例未通过，顶不住先睡了
2021.7.1	9：00-12:00	解决遗留问题，完成三个 L 的设计	完成
2021.7.1	13:00-23:59	完成面向应用的 ADT 设计及对应测试用例，API 设计及对应测试用例	提前完成
2021.7.2	9:00-23:59	完成值班表管理系统及进程管理系统	基本完成，手动调试仍有 bug
2021.7.3	00:00-2:00	继续解决 bug	完成
2021.7.3	9:00-15:00	完成课程管理系统	延误半小时完成
2021.7.3	16:00-18:00	调试三个应用，尽量找漏洞并修补	真的找到漏洞了
2021.7.3	20:00-22:00	查阅资料，完成正则表达式对数据的处理	完成
2021.7.4	10:00-12:00	完成变化 1	找到新的漏洞，修补，未完成变化 1
2021.7.4	13:00-15:00	完成变化	全部完成

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
不太懂 Decorator 原理	上网查资料，对照 PPT 理解
API 的设计不知道有什么好的算法	暴力计算，疯狂遍历

不知道怎么处理日期	上网查教程，学习 LocalDate
不会随机	上网查资料学 Random
不会正则表达式	上网学，参照多份代码反复看正则表达式的示例

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

通过实验对 Decorator 设计模式有了更深的理解，并亲自实现了一次，相信会在以后记忆深刻。同时也在实验中对 Java 类更加的熟悉了，比如这次新接触到的 LocalDate 和 Random，确实是很好很实用的类。而且也自学了正则表达式的写法，对于匹配操作有了一定了解。但是在实验中也发现自己有必要了解一些算法知识了，总是过度依赖遍历去实现各种功能，导致性能不理想，代码量也大。到最后一次实验还是没逃脱堆到最后赶进度的命运，确实很痛苦，希望以后能养成提前安排的习惯吧。

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？

前者是一种具有更长远眼光的做法，能在以后更加方便的实现更多具有相似性的应用；后者虽然在短期看来很方便，但是可扩展性差，在遇到相似情景的时候，只能做大量重复工作。本次实验的三个应用多次使用到自己实现设计的 ADT，带来了一定便利，能减少大量重复工作。

(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？

既能使他人更容易读懂自己的代码，也能使自己以后能快速回想起当初设计与实现的思路。时刻注意程序的正确性，同时保证安全性，不会被客户端恶意破坏。如果我以后还编程的话，应该会坚持，确实对于提高代码的可读性和程序的安全性都有所帮助。

(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？

不了解算法，只能暴力解决，性能肯定不理想，还是很难的。但是写出来

之后直接调用时带来的方便确实令人开心。

(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？

比起手动输入更方便，可以直接将要输入的信息存入文本，一次性读取，而且可以在以后多次使用而免于大量的手动重复输入。但是问题就是要能正确进行匹配，对于非法输入要能恰当处理，更具有挑战性。

(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？

后续的操作建立在前面的构造基础上，一旦基础没打好，后果将是灾难性的——“重头再来”，所以每一个设计都要慎重，提前考虑未来可能会遇到的问题。时间紧，任务多，心态崩了无数次，每次崩了就告诉自己学会坚强，又不是第一次崩了……

(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？

一定要提前考虑清楚各个方法的常见程度，最最普遍的往上走，实现为接口、抽象类方法；最最独特的往下走，实现为具体子类的方法；对于局部的共性，则需要合理选择设计模式、继承、组合等。总之就是要先设计好再具体实现。

(7) 关于本实验的工作量、难度、deadline。

工作量再创新高，考虑三周时间其实真的不多，但是在期末就显得有点超出承受范围了。难度也有所上升，毕竟除了手册上的思路提示，其他的全靠自己，但是也能进一步提升个人能力，还是不错的。Deadline 还好吧，也算是赶上了。

(8) 下周就要进行考试了，你对《软件构造》课程总体评价如何？

总体来说是一门优秀的课程，很锻炼编程水平，而且不是让我们仅仅以程序员的视角去考虑问题，而是启发我们以架构师的视角去思考程序的设计与实现，使得最终呈现出来的程序具有更高的质量。