哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Lab Manuals for

Software Construction

Lab-3

Reusability and Maintainability oriented Software Construction

Faculty of Computing

Harbin Institute of Technology

Spring 2021

目录

1 实验目标

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），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

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

请访问该 URL，按照提示建立自己的 Lab3 仓库并关联至自己的学号。

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。

3 实验要求

本实验需要设计并实现抽象数据类型 `IntervalSet`，对现实中各类“在特定时间段进行的特定任务”进行抽象，并在三个具体应用中使用它。

在设计该 ADT 时，需要对其进行泛型化，从而使其抽象能力更强、适应现实中不同情况需求的能力更强。

请为每个你设计和实现的 ADT 撰写 mutability/immutability 说明、AF、RI、safety from rep exposure。给出各 ADT 中每个方法的 spec。为每个 ADT 编写测试用例，并写明 testing strategy。

3.1 待开发的应用场景

要为以下场景开发 Java 程序，故在设计和实现 ADT 的时候需充分考虑这些应用之间的相似性和差异性，使 ADT 有更大程度的复用和更容易面向各种变化。

(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 日到 3 月 6 日期间），每天只能安排唯一一个员工在单位值班，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
…			

(2) 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。

同一颜色的矩形表示同一个进程

(3) 大学课表管理（`CourseSchedule`）：看一下你自己的课表，每一上午 10:00-12:00 和每周三上午 8:00-10:00 在正心楼 42 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 3-4 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。

2021春季学期1936601班级课表

		星期一	星期二	星期三	星期四	星期五	星期六	星期日
上午	第1,2节	数理方程◇王峰[1-10周]◇正心42	计算机系统◇刘宏伟[1-14周]◇正心21	计算方法◇张池平[1-8周]◇正心42 软件构造◇王思杰[9-16周]◇正心42	数理方程◇王峰[1-10周]◇正心42	形式语言与自动机◇孙大烈[1-8周]◇正心12		
上午	第3,4节	计算方法◇张池平[1-8周]◇正心42 软件构造◇王思杰[9-16周]◇正心42	马克思主义基本原理概论◇史焕翔[2-13周]◇正心33	体育◇[1-16周]	计算机系统◇刘宏伟[1-14周]◇正心21	计算方法(上机)◇张池平[5-8周]◇L520 信息安全概论◇瞿健宏[9-16周]◇正心13		
下午	第5,6节	信息安全概论◇瞿健宏[9-16周]◇正心13	软件构造(上机)◇王思杰[10-17周]◇格物207		计算机系统(上机)◇刘宏伟[2，4，7，9，11-14周]◇G709 计算机系统(上机)◇刘宏伟[2，4，7，9，11-14周]◇G712 马克思主义基本原理概论(辅导)◇史焕翔[8周]◇正心710	马克思主义基本原理概论◇史焕翔[2-13周]◇正心33		
下午	第7,8节			形式语言与自动机◇孙大烈[1-8周]◇正心12 形势与政策（2）◇姜昱子[9-12周]◇致知31		学术英语听说◇周之南[1-16周]◇正心504		
晚上	第9,10,11节							
晚上	第12节							

（为简化起见，无需考虑图中各课程后面标注的起至周次）

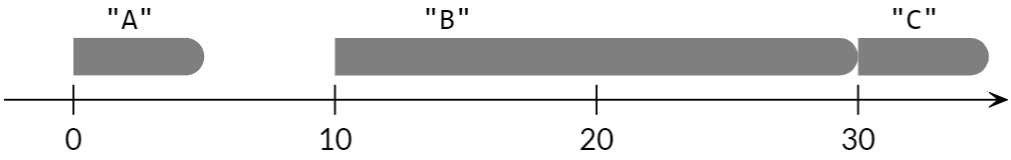
3.2 面向可复用性和可维护性的 ADT 设计：`IntervalSet<L>`

考虑上节给出的三个应用，其中都包含了具有不同特征的“时间段集合”对象，为了提高软件构造的可复用性和可维护性，可为其设计和构造一套统一的 ADT。

3.2.1 `IntervalSet<L>`

引入 `IntervalSet<L>`，这是一个 mutable 的 ADT，描述了一组在时间轴上分布的“时间段”（interval），每个时间段附着一个特定的标签，且标签不重复。

例如：下图的时间段集合可表示为`{ A=[0,5], B=[10,30], C=[30,35] }`。在该例子中，标签是简单的字符串（“`A`”、“`B`”、“`C`”），但实际应用中标签可以是任何 immutable 类型的 ADT，也就是 `IntervalSet<L>`中的 `L`。

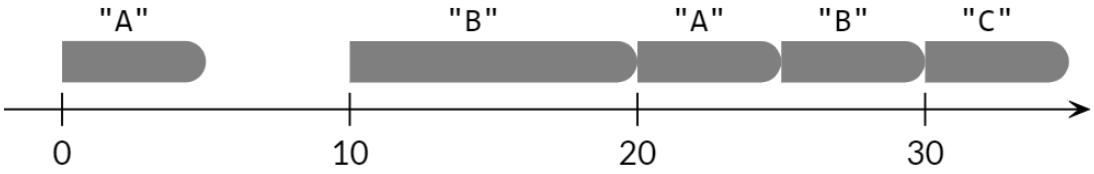


例如上节应用 1 中的标签应该是“员工”ADT（`Employee`），应用 2 中的标签是“进程”ADT（`Process`），应用 3 中的标签是“课程”ADT（`Course`）。注意，一定要保证 `L` 是 immutable 的。

3.2.2 `MultiIntervalSet<L>`

但是，`IntervalSet<L>`中，同一个标签对象 `L` 只被绑定到唯一一个时间段上，这可以满足诸如排班表管理这样的简单应用，却无法满足诸如操作系统进程调度和课程表管理这种复杂应用：同一个标签对象 `L` 可被绑定到多个时间段上。

如下图所示，标签对象“`A`”与两个时间段发生联系，“`B`”也是如此。该图可表示为`{ A=[[0,5],[20,25]], B=[[10,20],[25,30]], C=[[30,35]] }`。



3.3 ADT 的设计与实现

3.3.1 `IntervalSet<L>`的设计与实现

为接口 `IntervalSet<L>`设计应提供的共性接口方法，并确定它们是静态方法还是实例方法：

- 创建一个空对象：`empty()`
- 在当前对象中插入新的时间段和标签：`void insert(long start, long end, L label)`
- 获得当前对象中的标签集合：`Set<L> labels()`
- 从当前对象中移除某个标签所关联的时间段：`boolean remove(L label)`
- 返回某个标签对应的时间段的开始时间：`long start (L label)`
- 返回某个标签对应的时间段的结束时间：`long end (L label)`
- …不限于上述方法，你可以继续抽象各应用所需要的其他全局共性方法

注 1：为简化起见，时间轴上的时刻都统一用 long 数据类型表示。

注 2：上述方法若有需要抛出的异常，可自定义异常类并在上述方法定义中使用。

注 3：接口中定义的方法应当是该 ADT 所有实例对象都统一具有的“行为”（由各方法的 spec 所规定，但实现方式可能不同）。如果某方法对某些实例对象是有价值的、对其他实例对象是不存在的“行为”，那么就不应该放在接口里。当你读完 3.6.1 节描述的各类差异之后，可能需要返回到这里重新评估上述给出的各个方法是否都是“统一具有的行为”，从而对它们做出删减或增加。

请根据你对应用的理解和抽象，为上述各方法设计 Spec，并据此设计相应的测试用例。

进而，设计一个类 CommonIntervalSet<L>来实现接口 IntervalSet<L>。请设计它的 rep、RF、AI、Safety from Rep Exposure，实现各个接口方法、checkRep()、toString()，其中 toString()将对象内容表示为人容易理解的文本字符串形式。

3.3.2 MultiIntervalSet<L>的设计与实现

你可自行决定将 MultiIntervalSet<L>实现为接口、抽象类或是具体类。如果是接口或抽象类，那么必须还要实现一个具体类，其名字应为 CommonMultiIntervalSet<L>。

在其 rep 中，要求必须使用 IntervalSet<L>作为其 rep 的一部分，意即必须复用 IntervalSet<L>及其实现类中已经实现的功能。除此之外没有其他限制。

MultiIntervalSet<L>所应提供的方法包括：

- 创建一个空对象：empty()或不带任何参数的构造函数
- 创建一个非空对象：构造函数 MultiIntervalSet(IntervalSet<L> initial)，利用 initial 中包含的数据创建非空对象
- 在当前对象中插入新的时间段和标签：void insert(long start, long end, L label)
- 获得当前对象中的标签集合：Set<L> labels()
- 从当前对象中移除某个标签所关联的所有时间段：boolean remove(L label)
- 从当前对象中获取与某个标签所关联的所有时间段：IntervalSet<Integer> intervals(L label)，返回结果表达为 IntervalSet<Integer>的形式，其中的时间段按开始时间从小到大的次序排列。例如：当前对象为{ "A"=[[0,10],[20,30]], "B"=

[[10,20]] }，那么 intervals("A")返回的结果是{ 0=[0,10], 1=[20,30] }。

- …不限于上述方法，你可以继续抽象需要的其他全局共性方法

为上述各方法设计 Spec，并据此设计相应的测试用例。

进而，设计 rep、RF、AI、Safety from Rep Exposure，实现各个方法、checkRep()、toString()，其中 toString()将对象内容表示为人容易理解的文本字符串形式。

3.3.1 节里的三个“注”，对 MultiIntervalSet 来说仍然有效。

问题：考虑 MultiIntervalSet<L>和 IntervalSetSet<L>之间的共性和差异，它们二者可以进一步抽象形成一个更高层次的 ADT 吗？如果可以，你可以不受本节要求的限制，使用一个 ADT 来替代这两个 ADT。

3.4 实际应用中的变化

3.4.1 三个维度上的差异

针对 IntervalSet<L>和 MultiIntervalSet<L>，除了上述共性的设计，实际应用中还会有以下三个维度上的差异：

1. 是否允许时间轴上有空白。若“是”，则时间轴上某些时间段没有跟任何标签关联；若“否”，则时间轴上的任何时间点都隶属于某个标签的时间段。例如：在应用 1 中，不允许有空白；在应用 2 和应用 3 中，允许有空白。//一种直观的想法是在 ADT 的 RI 中将“有无空白”的约束定义出来并在 checkRep()中加以检查，但这是不合理的：该 ADT 是 mutable 的，其实例对象可不断调用 insert(...)向其中插入 interval，只有到用户彻底插入结束之后才能够判断“有无空白”的约束是否被遵循，而不是在每次 insert 之后均要判定。因此，对该约束的判定需要在具体子类中表达：当不再增加新的时间段时，检查时间轴上特定范围内是否存有空白，以判断其是否合法。
2. 是否允许不同的 interval 之间有重叠。若“允许”，则多个 interval 可能存在重叠，但具有相同 label 的多个 interval 之间仍不允许有重叠；若“不允许”，则任何两个 interval 之间不会有任何重叠（即某个时间点只能属于一个时间段）。例如：在应用 1 和应用 2 中，不允许有重叠；在应用 3 中，允许有重叠。//这个需要在 ADT 的 RI 和 checkRep()中检查，并根据“允许”或“不允许”，从 ADT 中派生出不同的子类型。

3. 是否包含周期性的时间段。周期性的时间段是指某个时间段按特定的周期长度重复出现，例如应用 3 中以“一周”为单位重复某个课程，但应用 1 和应用 2 中不存在这种情况。//可派生出子类型，针对每个时间段，标识其重复与否

3.4.2 六种可能的设计方案

如果各应用在上述某个维度上的特征值完全相同，则可以将针对处理该共性特征值的操作在上层的实现类中定义和实现，可实现完全的复用。如果在某个维度上的特征取值完全不同，就必须在各个应用的具体子类中分别实现其对不同特征取值的个性化操作。

但是，这三个应用在任何一个维度上的特征取值既不是完全相同，也不是完全不同。如何最大程度上实现在每个维度上具有相同特征的多个应用之间的功能复用？

方案 1：将所有特殊操作均放入顶层的抽象接口

将各应用中出现的所有特殊操作都放入顶层接口 IntervalSet<L>中定义并在相应的实现类 CommonIntervalSet<L>中实现它们。对各应用的具体子类，如果接口中的某个操作不适用于自己，则应用开发者不要使用它们。例如，接口中定义了一个 void insert(long start, long end, L label) throws IntervalConflictException 方法，用于针对“不允许不同的 interval 之间有重叠”的 ADT 对象的插入操作；对课表管理和操作系统进程调度来说，就不能使用该 insert 方法（即无需考虑插入新 interval 时与已有各 interval 的重叠问题），故可以在其子类中 override 该 insert 方法并将其内部功能禁用（例如，override 后的方法体设置为空实现，或者直接 assert false，或者 throws new Exception）。但是，这么做会导致各个应用子类中出现了较多 override 空实现的方法，代码会很不美。另一方面，这种做法会导致违反 LSP 原则，因为子类中空实现的 insert 方法不再符合接口中 insert 方法的 spec，子类型对象无法替代父类型对象。

方案 2：将各特殊操作分别放入底层的应用子类

将针对不同特征取值的具体操作分别放在三个应用的子类中加以实现。例如：在实现排班表的子类中实现上述 insert 方法，其他两个应用的子类不需实现该方法。该方案的缺点是：某些方法的代码可能是重复的、分散在多个类中，可维护性和可复用性差，将来一旦面临变化就需要修改多处代码。

方案 3：为不同特征取值分别定义接口并在子类中实现其特殊操作

为每个维度上的不同特征取值分别定义不同的接口，在接口中定义特殊的操作，各应用的具体子类根据自己的需求来实现不同特征的接口。以“是否允许重叠”维度为例：

特征取值	接口	接口中的方法
是	OverlapIntervalSet<L>	void insert(...) //无抛出异常
否	NonOverlapIntervalSet<L>	void insert(...) throws xxxException

在这种方案下，每个应用的具体子类可以按以下方式加以实现：

```
class DutyIntervalSet<L>
    extends CommonIntervalSet<L>        //继承共性行为
    implements NonOverlapIntervalSet<L>,//增加“无重叠”的行为
               NoBlankIntervalSet<L>,//增加“非空”的行为
               NonPeriodicIntervalSet<L>//增加“非周期”行为
```

其中，共性的功能通过继承 CommonIntervalSet 类实现复用，而特殊的功能则通过 override 三个接口中的方法来实现。

这种方案与方案 2 有同样的缺陷：某些局部共性的操作仍然需要在多个子类中分别 override、分别实现。

方案 4：定义接口并实现具体类，通过继承树实现各应用在多维度上的不同特征取值的组合

在方案 3 基础上，除了为每个维度上的每个特征取值定义相应的接口，另外为每个接口分别构造一个实现类，该类一方面继承 CommonIntervalSet 中的全局共性操作，另一方面在该类中完成对局部共性操作的代码。例如针对上表中的 NonOverlapIntervalSet<L>接口所构造的实现类如下所示，从而避免了在多个应用子类中分别实现 insert(...) throws xxException 方法。

```
public class NonOverlapIntervalSetImpl<L>
                  extends CommonIntervalSet<L>
                  implements NonOverlapIntervalSet<L>
   @Override
   public void insert(long start, long end, L label)
                        throws IntervalConflictException{
       //在这里实现对“不允许存在重叠”的检查并决定是否抛出异常
   }
}
```

但是，该方案存在的问题是“组合爆炸”。上面这个例子只是实现了“是否可重叠”这个维度上的“不可重叠”特征，如果再考虑另一个维度“是否可空白”的“不允许空白”特征，就需要继续定义一个新的类，继承上面这个类，并进一步实现“不允许空白”的特征：

```
public class NoBlankAndNonOverlapIntervalSetImpl<L>
                  extends NonOverlapIntervalSetImpl<L>
                  implements NoBlankIntervalSet<L> {
   private long start, end;
       //增加 rep，表示在[start, end]范围内不应存在空白时间段
   @Override //方法在 NoBlankIntervalSet 接口中定义
   public void checkNoBlank() {...}  //在此检查是否有空白
}
```

如果继续考虑第三个维度上的特征，就需要在上述 NoBlankAndNonOverlapIntervalSetImpl 基础上继续扩展。将各个维度的个性化逐步组合进去之后，即可得到符合各应用特征的具体子类。

最终，多个维度上的多种特征取值的不同组合，将形成庞大的继承树，这将给软件系统的维护代码极大困难。

方案 5：CRP，通过接口组合实现局部共性特征的复用

针对方案 3，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。

```
public interface NonOverlapIntervalSet {...}
public interface NoBlankIntervalSet{...}
public interface NonPeriodicIntervalSet{...}
...
public class NonOverlapIntervalSetImpl
                   implements NonOverlapIntervalSet
public class NoBlankIntervalSetImpl
                   implements NoBlankIntervalSet
```

```
public class NonPeriodicIntervalSetImpl
                  implements NonPeriodicIntervalSet

public interface IDutyIntervalSet
         extends   NonOverlapIntervalSet,
                     NoBlankIntervalSet,
                     NonPeriodicIntervalSet { }

class DutyIntervalSet extends CommonIntervalSet
                 implements IDutyIntervalSet {
   private NonOverlapIntervalSetImpl nois;
   private NoBlankIntervalSetImpl nbis;
   private NonPeriodicIntervalSetImpl npis;
   public DutyIntervalSet(NonOverlapIntervalSet nois,
               NoBlankIntervalSet nbis,
               NonPeriodicIntervalSet npis)  {
      ...//设置 delegation 关系
   }
   @Override
   public void checkNoBlank() {..  //在此检查是否有空白
      nbis.checkNoBlank();
   }
   @Override
   public void insert(long start, long end, L label)
                     throws IntervalConflictException{
      //在此检查“不允许存在重叠”检查并决定是否插入、是否抛出异常
      nois.insert(...);
   }
   ...
}
```

方案 4 和方案 5 体现了在复杂场景下使用 inheritance 和 delegation 实现复用的差异。

方案 6：使用 decorator 设计模式

将 CommonIntervalSet 看作是原始的、未被装饰的对象，将这三个维度看

作是三种“装饰”（每个维度的不同特征取值可以产生不同的“装饰”效果）。请参照讲义上关于该设计模式的说明，设计相应的子类型继承关系树，然后在具体应用中通过为一个 CommonIntervalSet 对象逐层装饰三个不同特征，即可实现应用所需的组合特征。

在你的实验中，请根据你的偏好选择上述某种方案加以设计和实现，或者采用你认为合理的设计方案。你也可以实现多种方案，并对它们的可维护性、可复用性等做些对比分析。

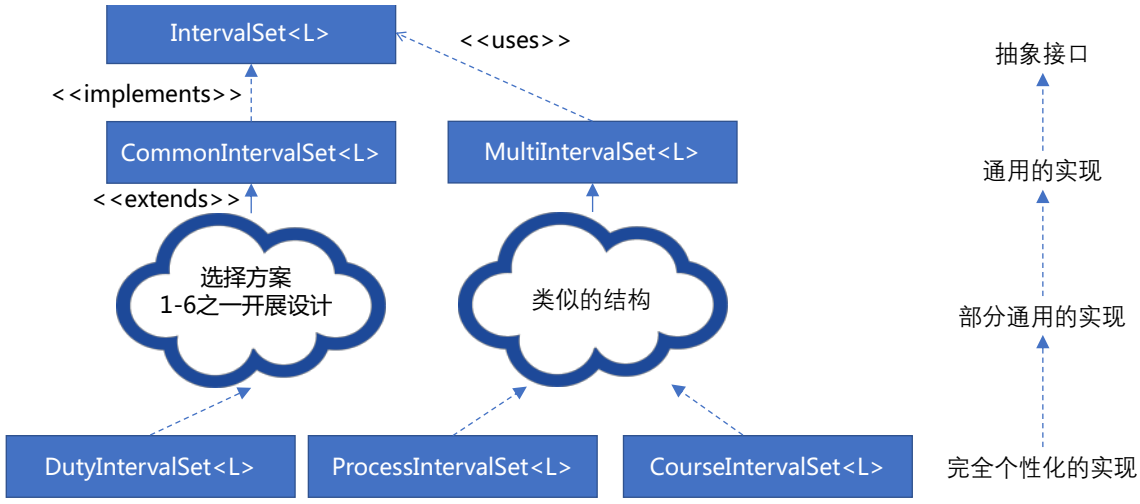
3.4.3 面向各应用的 ADT 子类型

为了支持三个应用，你需要基于上一小节的某个设计方案分别构造/派生出更具体的、面向应用的 ADT 子类型，每个子类型可根据应用需要来选择是基于 IntervalSet<L>派生还是基于 MultiIntervalSet<L>派生。

- DutyIntervalSet：代表一个排班表；
- ProcessIntervalSet：代表一个操作系统对进程的调度记录；
- CourseIntervalSet：代表某个班级的特定课表。

针对从 IntervalSet<L>和 MultiIntervalSet<L>出发所派生出的所有 ADT 子类型，撰写 AF、RI、Safety from rep exposure，以及每个方法的 spec。为它们设计和编写 JUnit 测试用例，并撰写 testing strategy。

归纳起来，最终设计得到的 ADT 结构类似于下图所示：



3.5 面向复用的设计：L

IntervalSet<L>和 MultiIntervalSet<L>中的泛型参数 L，可以是你所设

计的任何 immutable 的类。

对三个应用来说，其 L 分别应为“员工”（Employee）、“进程”（Process）、“课程”（Course），所需关注的属性分别为：

- Employee：姓名、职务、手机号码
- Process：进程 ID、进程名称、最短执行时间、最长执行时间
- Course：课程 ID、课程名称、教师名字、地点

分别实现这些 ADT。

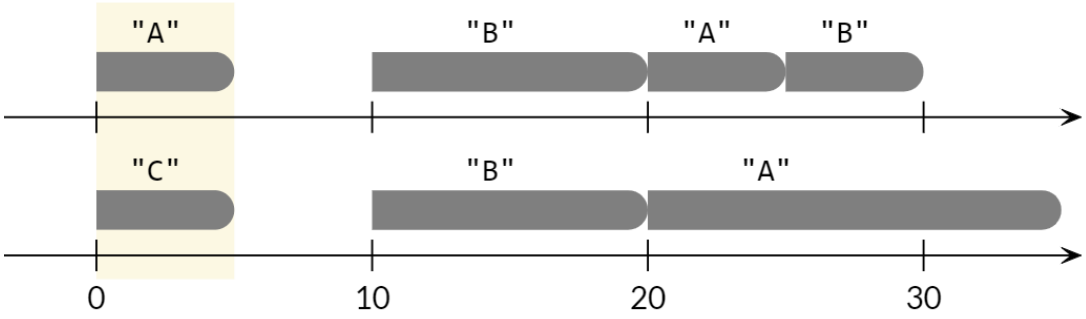
3.6 可复用 API 设计与开发

设计和实现以下各个 API，它们可以包含在 IntervalSet<L>和 MultiIntervalSet<L>的相关实现类或子类中，也可以放在单独的 APIs.java 类中。

1. 计算两个 MultiIntervalSet 对象的相似度：

```
double Similarity(MultiIntervalSet<L> s1, MultiInterval Set<L> s2)
```

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
```

2. 发现一个 IntervalSet<L>或 MultiIntervalSet<L>对象中的时间冲突比例（仅针对应用 3）

```
double calcConflictRatio(IntervalSet<L> set)
double calcConflictRatio(MultiIntervalSet<L> set)
```

所谓的“冲突”，是指同一个时间段内安排了两个不同的 interval 对象。用发生冲突的时间段总长度除于总长度，得到冲突比例，是一个[0,1]之间的值。

3. 计算一个 IntervalSet<L>或 MultiIntervalSet<L>对象中的空闲时间比例

```
double calcFreeTimeRatio(IntervalSet<L> set)
double calcFreeTimeRatio(MultiIntervalSet<L> set)
```

所谓的“空闲”，是指某时间段内没有安排任何 interval 对象。用空闲的时间段总长度除于总长度，得到空闲比例，是一个[0,1]之间的值。

3.7 应用设计与实现

利用前面所开发的 ADT，为三个应用场景分别开发程序，可以实现为 GUI 程序，也可以是命令行程序，三个应用的主程序分别实现为：

- DutyRosterApp.java
- ProcessScheduleApp.java
- CourseScheduleApp.java。

注 1：除了下面各节给出的功能，还可根据自己合理的设计，在应用中实现其他你认为有价值的功能（但不参与评分）。

注 2：有人会认为三个应用的开发工作量太大，且这些应用的功能非常相像，何必重复这么多次。这可以用来检查你的 ADT 设计是否有足够的可复用性和适应性，如果 ADT 设计得好，应用开发的工作量会大幅度减少。

3.7.1 排班管理系统

针对排班管理系统，所需完成的功能为：

Step 1　设定排班开始日期、结束日期，具体到年月日即可。

Step 2　增加一组员工，包括他们各自的名字、职务、手机号码，并可随时删除某些员工。如果某个员工已经被编排进排班表，那么他不能被删除，必须将其排班信息删掉之后才能删除该员工。员工信息一旦设定则无法修改。

Step 3　可手工选择某个员工、某个时间段（以“日”为单位，最小 1 天，可以是多天），向排班表增加一条排班记录，该步骤可重复执行多次。在该过程中，用户可随时检查当前排班是否已满（即所有时间段都已被安排了特定员工值班）、若未满，则展示给用户哪些时间段未安排、未安排的时间段占总时间段的比例。

Step 4　除了上一步骤中手工安排，也可采用自动编排的方法，随机生成排班表。

Step 5　可视化展示任意时刻的排班表。可视化要直观明了，可自行设计。

3.7.2　操作系统的进程调度管理系统

针对操作系统的进程调度管理系统，所需完成的功能为：

Step 1　增加一组进程，输入每个进程的 ID、名称、最短执行时间、最长执行时间；进程一旦设定无法再修改其信息。

Step 2　当前时刻（设定为 0）启动模拟调度，随机选择某个尚未执行结束的进程在 CPU 上执行（执行过程中其他进程不能被执行），并在该进程最大时间之前的任意时刻停止执行，如果本次及其之前的累积执行时间已落到[最短执行时间，最长执行时间]的区间内，则该进程被设定为“执行结束”。重复上述过程，直到所有进程都达到“执行结束”状态。在每次选择时，也可“不执行任何进程”，并在后续随机选定的时间点再次进行进程选择。

Step 3　上一步骤是“随机选择进程”的模拟策略，还可以实现“最短进程优先”的模拟策略：每次选择进程的时候，优先选择距离其最大执行时间差距最小的进程。

Step 4　可视化展示当前时刻之前的进程调度结果，以及当前时刻正在执行的进程。可视化的形式要直观明了，可自行设计。

3.7.3　课表管理系统

针对课表管理系统，所需完成的功能为：

Step 1　设定学期开始日期（年月日）和总周数（例如 18）；

Step 2　增加一组课程，每门课程的信息包括：课程 ID、课程名称、教师名字、地点、周学时数（偶数）；

Step 3　手工选择某个课程、上课时间（只能是 8-10 时、10-12 时、13-15 时、15-17 时、19-21 时），为其安排一次课，每次课的时间长度为 2 小时；可重复安排，直到达到周学时数目时该课程不能再安排；

Step 4　上步骤过程中，随时可查看哪些课程没安排、当前每周的空闲时间比例、重复时间比例；

Step 5　因为课程是周期性的，用户可查看本学期内任意一天的课表结果。

3.8　基于语法的数据读入

针对 3.10 节中开发的“值班表管理”应用，为其扩展一个功能：从一个外部文本文件读入数据并使用正则表达式 parser 对其进行解析，从中抽取信息，构造

值班表对象。输入文件的语法示例如下所示，你可从以下地址下载示例文件用于你的程序测试：

https://github.com/rainywang/Spring2021_HITCS_SC_Lab3

在以下示例中，不带有下划线的文字部分为固定说明部分，带有下划线的文字部分为可变化信息。“//”后续文字是解释说明，并非语法的构成部分。

文件里描述的 Employee、Period、Roster 的次序是不确定的，违反该次序不能被看作非法。

在读取文件过程中，若发现有违反本例中解释说明部分给出的规则，则该文件不符合语法，需结束读取并提示用户选择其他合法的文件。

另外，请忽略文件中的所有缩进，它并非语法的组成部分。

```
Employee{   //多个员工信息
  ZhangSan{Manger,139-0451-0000} //{}前是员工姓名，大小写字母构成
                                 //不同的行内的员工名字是唯一的，不能重复
  LiSi{Secretary,151-0101-0000} //{}内第一个分量是职务，大小写字母和空格
  WangWu{Associate Dean,177-2021-0301}
                                 //{}内第二个分量是中国境内合法的手机号码
                                 //共 11 位，分为三段（3-4-4），用“-”相连
}
Period{2021-01-10,2021-03-06}   //表示排班的开始日期和结束日期
Roster{                         //表示排班计划
  ZhangSan{2021-01-10,2021-01-31} //分别为姓名、开始日期、结束日期
  LiSi{2021-02-01,2021-02-28}       //上述所有日期的格式为 yyyy-MM-dd
  WangWu{2021-03-01,2021-03-10}
      //出现在 Roster 内的员工，必须在 Employee 中已有定义，否则不合法
}
```

3.9 新的变化

前面开发出的各应用，面临着以下的几个功能变化。请考查原有的设计，是否能以较小的代价适应这些变化。修改原有设计，使之能够应对这些变化。在修改之前，请确保你之前的开发已经 commit 到 Git 仓库，然后创建新分支“change”，在该分支上完成本节任务。

- 排班应用：可以出现一个员工被安排多段值班的情况，例如张三的值班日期为(2021-01-01, 2021-01-10), (2021-02-01, 2021-02-06)；
- 课表应用：不管学生选课状况如何，不能够出现两门课排在同一时间的情况（即“无重叠”）

在具体修改之前，请根据前面的 ADT 与应用设计，先大致估算一下你的程序需要变化多大才能适应这些变化。修改之后，再相对精确的度量一下你在该节之前所做的 ADT 设计以多大的代价适应了该表中的每一个变化？这里的“代价”可用“代码修改的量”和“修改所耗费的时间”加以估计。

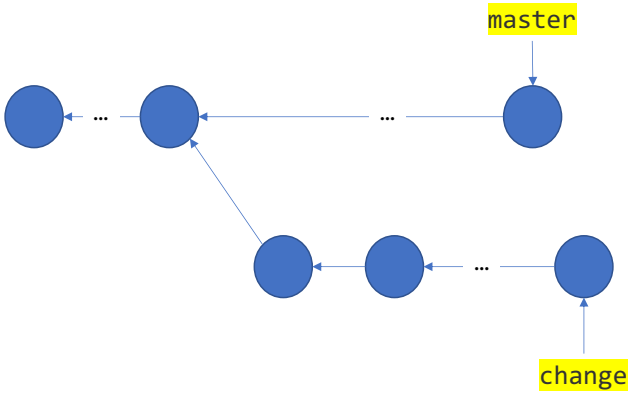
注意 1：除非这些变化修改了之前各节提及的某些功能，你针对这些变化所做的修改都不应破坏之前已经写好的功能。

注意 2：提交到仓库之后，`master` 分支需仍然指向本节任务之前的结果，TA 会到仓库的 `change` 分支上检查本节任务。

以下给出了 Git 的相关操作指南。

```
…最初在 master 上工作…
git checkout -b change 创建新分支
…按上面的要求进行代码修改…
git add *
git commit -m "change"在该分支上提交
git checkout master                    切换回 master 分支
…请不要使用 git merge change 进行合并修改
…请不要使用 git branch -d change 删除分支
```

你的 Git 仓库中的 Object Graph 应类似于下图所示。其中上方的 master 分支包含了本节之前的开发结果，下方的分支是针对本节的变化需求的开发结果。



3.10 项目结构

```
项目名：Lab3-学号
        src          java 文件自行组织包结构，做到有序、清晰
        test         JUnit 测试程序目录，与 src 结构保持一致
        lib          程序所使用的所有外部库文件
        doc          实验报告
        其他辅助目录
```

注：如果手册中给出的接口/类无法满足你的设计需要，请自行增加相应的包、

接口、类，但不要更改上述包结构。

4 实验报告

针对上述编程题目，请遵循雨课堂中 Lab3 页面给出的**报告模板**，撰写简明扼要的实验报告。

实验报告的目的是记录你的实验过程，尤其是遇到的困难与解决的途径。不需要长篇累牍，记录关键要点即可，但需确保报告覆盖了本次实验所有开发任务。

注意：

- 实验报告不需要包含所有源代码，请根据上述目的有选择的加入关键源代码，作为辅助说明。
- 请确保报告格式清晰、一致，故请遵循目前模板里设置的字体、字号、行间距、缩进；
- 实验报告提交前，请“目录”上右击，然后选择“更新域”，以确保你的目录标题/页码与正文相对应。
- 实验报告文件可采用 Word 或 PDF 格式，命名规则：Lab3-学号-Report。

5 提交方式

截止日期：第 17 周周日夜间 23:55。

源代码：从本地 Git 仓库推送至个人 GitHub 的 Lab3 仓库内。

实验报告：随代码仓库（doc）目录提交至 GitHub。

6 评分方式

Deadline 之后，教师和 TA 对学生在 GitHub 上的代码进行测试，阅读实验报告，做出相应评分。