哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Lab Manuals for

Software Construction

Lab-3

Reusability and Maintainability oriented Software Construction

School of Computer Science and Technology

Harbin Institute of Technology

Spring 2020

目录

1 实验目标

本次实验覆盖课程第 3、4、5 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
- 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

本次实验给定了五个具体应用（高铁车次管理、航班管理、操作系统进程管理、大学课表管理、学习活动日程管理），学生不是直接针对五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/aMg3ti15

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一个抽象数据类型 `PlanningEntry`，对现实中各类“利用特定资源、在特定地点/位置开展的一项任务”进行抽象，并在五个具体

应用中使用它。

在设计该 ADT 时，需要对其进行泛型化，考虑所用资源的不同类型，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。

-----请务必注意-----

本实验后续提及了以下五个应用场景，你需要完成的应用为：1 必做、2 和 3 任选 1 个、4 和 5 任选 1 个。也就是说，你至少要完成 3 个应用：1 and (2 or 3) and (4 or 5)。

1. 航班管理
2. 高铁车次管理
3. 操作系统进程管理
4. 大学课表管理
5. 学习日程管理

3.1 基本概念

计划项 `PlanningEntry`	表示一个待执行的活动/任务，它需要配置特定类型/数量的资源，并在特定的物理位置加以执行。 一个计划项的执行过程可分为若干个状态：未分配资源、已分配资源但未启动、已启动、已完成、已取消、挂起。
资源 `Resource`	计划项的执行过程中所需要占用的物理设施、数字设施或者人，例如一架飞机、一个动车组、一个投影机、计算机 I/O 设备或内存、飞行员、乘务员、教师等。在本实验中，每个计划项所占用的资源都是单一类型的。
物理位置 `Location`	计划项执行的物理地点，例如高铁站、机场、教室、CPU 核等。计划项可以是在一个位置上启动并执行完（例如在正心 11 教室上“软件构造”课），也可以从一个位置启动，抵达另一个位置才执行完（例如 CA1611 航班从北京起飞抵达哈尔滨结束），也可以经过一系列位置（例如高铁 G381 次从北京出发要经过 10 余个高铁站才抵达终点哈尔滨西），甚至可以在执行过程中切换位置（例如某进程最初在某个核上执行，挂起恢复后切换到另一个核上）。

时间属性 `Timeslot`	分为三种情况： ● 具有确定的开始时间和结束时间（例如某次“软件构造课”是 2020 年 3 月 1 日 8:00-10:00）。 ● 需要分段执行，因此其时间属性也需要分段标注（例如 G381 中间经停 10 余个车站，除了有出发和抵达时间，也需要有经停某车站的停车时间与发车时间）。 ● 事先无法确定具体的执行时间（例如 CPU 执行某个进程，其执行起始和结束时间不能提前计划，而是由 OS 动态调度，只有在执行结束后才能得知其时间属性）。
信息板 `Board`	每个具体地点都提供一个信息板，展示在该地点的过去发生和未来即将执行的计划项清单及各自的状态。例如：“哈尔滨西站”的大屏幕显示牌、“正心 11”教室外张贴的本教室课表等。
计划清单 `PlanningEntry Collection`	由一组计划项构成的整体。例如，1803101 班 2020 年春季的课表、哈尔滨机场 2020 年春夏季航班时刻表等。

3.2 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。

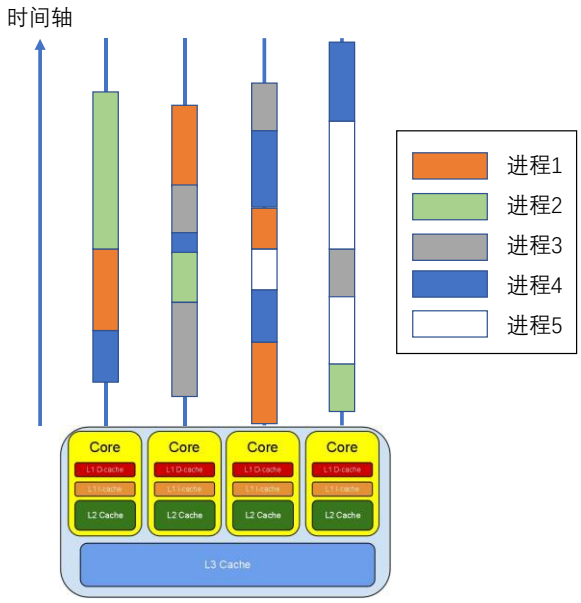
(1) 航班管理（`FlightSchedule`）：一个航班从某地机场起飞，抵达另一个机场（不考虑中途经停的情况）。航空公司需要分配一架具体飞机（资源）来执飞特定日期的航班。航空公司可以发布新航班、给新航班分配飞机、起飞、抵达目的地。航班在出发前可以被取消。航班一旦确定，不可更改起始和目的地。

(2) 高铁车次管理（`TrainSchedule`）：有很多高铁站和很多高铁车次，每个车次从起点高铁站出发，按次序经过一组高铁站，抵达终点高铁站，经

过各站的时间都是预先计划好的。例如，G381 次高铁 7:55 从北京南站出发，8:30 抵达天津西站，然后 8:32 继续出发，历经天津站、唐山站等高铁站，当日 14:31 抵达终点哈尔滨西站，全程 1331 公里，历时 6 小时 36 分钟。仍以 G381 为例，该车次每天都有，但不同日期的车次可能由不同的高铁动车组执行。铁路局可以增加一个新车次、为新车次分配一组车厢、始发、在中间站停车、在中间站发车、抵达终点站。高铁在起点站未出发前或中间站停车时可以被取消。同样的，高铁车次一旦确定，也不可更改起始站、中间站、终点站。

站次	站名	到达时间	开车时间	停留	天数	运行时间	里程
1	北京南站	始发站	07:55	0分钟	1	0分	0
2	天津西站	08:30	08:32	2分钟	1	35分	125
3	天津站	08:40	08:42	2分钟	1	45分	130
4	唐山站	09:14	09:16	2分钟	1	1小时19分	244
5	秦皇岛站	09:53	09:55	2分钟	1	1小时58分	391
6	葫芦岛北站	10:48	10:49	1分钟	1	2小时53分	529
7	锦州南站	11:06	11:08	2分钟	1	3小时11分	572
8	沈阳北站	12:23	12:25	2分钟	1	4小时28分	795
9	长春西站	13:31	13:33	2分钟	1	5小时36分	1093
10	哈尔滨西站	14:31	终点站	0分钟	1	6小时36分	1331

(3) 操作系统进程管理（`ProcessSchedule`）：考虑计算机上有一个多核 CPU，多个进程被操作系统创建出来，每个进程所需的内存等计算资源不同，它们在多核 CPU 上并行执行。由操作系统来调度决定在各个时刻由哪个核执行哪个线程。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，在进程被挂起前后，进程被分配的 CPU 核未必总是一样的。操作系统可以在进程被挂起的时候强行终止（取消）它。与前两个应用不同的是，在创建进程时，并不知道进程具体何时在哪个核上执行、何时被挂起、何时被重启、何时执行结束，只有进程执行结束后才可以得知。



带有4个核心的CPU

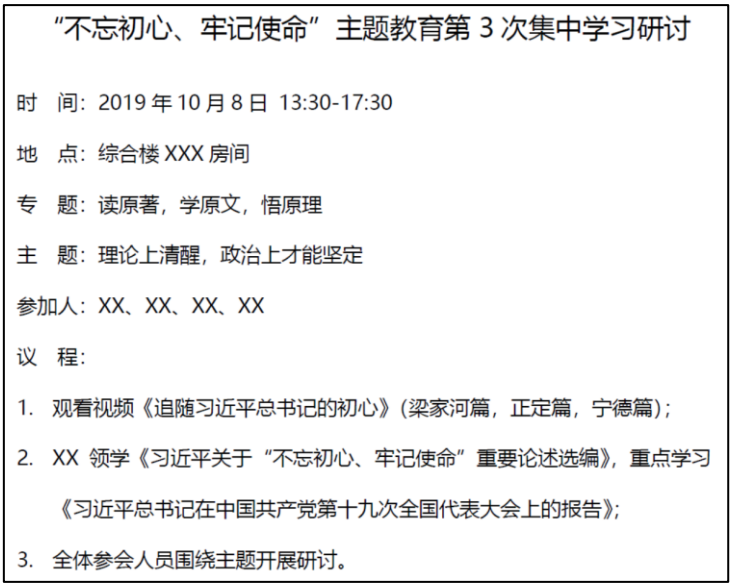
(4) 大学课表管理（`CourseSchedule`）：看一下你自己的课表，每周三上午10:00-11:45 在正心楼 23 教室上“软件构造”课程。上课需要特定的物理位置（教室），需要特定的资源（教师）。学校教务处每学期末统一安排下学期的全校课表，为每门课分配教室、安排教师、确定时间。教师可以上课、下课、更换教室。课程一旦启动，不可被阻塞/挂起、取消，但在未启动之前，教师可以因为临时有事而取消该次课程。

2020春季学期1803001班级课表

		星期一	星期二	星期三	星期四	星期五
上午	第1,2节	算法设计与分析◇张炜[2-13周]◇格物201		算法设计与分析◇张炜[2-13周]◇格物201		软件构造◇徐汉川[1-6，8-10，12-14周]◇正心23 软件构造◇徐汉川[7，11]单周◇正心23
上午	第3,4节	大学外语◇[1-16周]◇第3，4节	计算方法◇李道华[5-8周]◇正心44 计算方法◇吴勃英[1-4周]◇正心44 形式语言与自动机◇王春宇[10-17周]◇正心24	软件构造◇徐汉川[1-6，8-10，12-14周]◇正心23	计算方法◇李道华[5-8周]◇正心44 计算方法◇吴勃英[1-4周]◇正心44 马克思主义基本原理概论(辅导)◇张荣荣[9周]◇正心904 形式语言与自动机◇王春宇[10-17周]◇正心24	信息安全概论◇韩琦[9-16周]◇正心23
下午	第5,6节		近世代数◇李涛[1-8周]◇正心23 信息安全概论◇韩琦[9-16周]◇正心23	软件构造(上机)◇徐汉川[1-6，8-15周]◇格物213	马克思主义基本原理概论◇张荣荣[5-16周]◇正心43	近世代数◇李涛[1-8周]◇正心23
下午	第7,8节	马克思主义基本原理概论◇张荣荣[5-16周]◇正心43	体育◇[1-16周]	计算方法(上机)◇李道华[6-8周]◇L520		
晚上	第9,10,11节					
晚上	第12节					

(5) 学习日程管理（`ActivitySchedule`）：一个组织（政府部门、公司、大学、学院等）开展的各种学习活动，需要提前确定时间、地点，设定所需的学习资料及数量。学习活动可以启动执行、结束，也可以更改地点

等。学习活动一旦启动后，不可被阻塞/挂起。学习活动在启动之前可被取消。



“不忘初心、牢记使命”主题教育第 3 次集中学习研讨

时　间：2019 年 10 月 8 日 13:30-17:30

地　点：综合楼 XXX 房间

专　题：读原著，学原文，悟原理

主　题：理论上清醒，政治上才能坚定

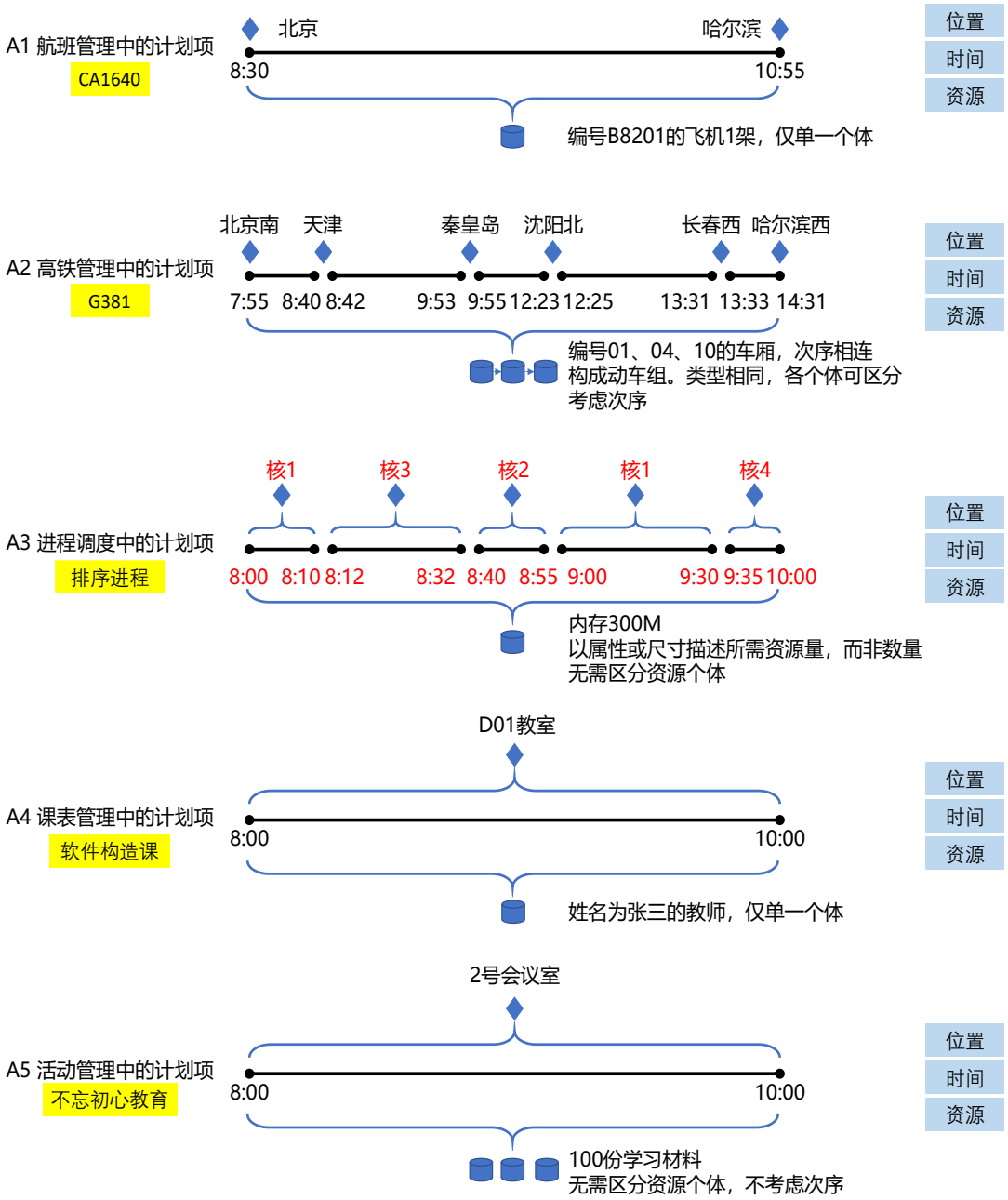
参加人：XX、XX、XX、XX

议　程：

1. 观看视频《追随习近平总书记的初心》（梁家河篇，正定篇，宁德篇）；
2. XX 领学《习近平关于“不忘初心、牢记使命”重要论述选编》，重点学习《习近平总书记在中国共产党第十九次全国代表大会上的报告》；
3. 全体参会人员围绕主题开展研讨。

3.3 面向可复用性和可维护性的设计：`PlanningEntry<R>`

考虑之前给出的五个应用，其中都包含了具有不同特征的“计划项”对象，为了提高软件构造的可复用性和可维护性，需为其设计和构造一套统一的 ADT。为了便于理解，下图给出了五个应用中包含的“计划项”的具体实例。



其中：

- 水平轴描述的是“时间属性”，分为：单一起始/结束时间（航班、课程、活动）、多段起始/结束时间（高铁、进程）。为简便起见，图中只给出了几时几分，但实际应用中需带有日期。需要说明的是：进程调度中的计划项上所标注的时间用红色区分，表明这些时间信息并非提前设定，而是在运行后才被事后记录的。
- 水平轴上方标注的信息为“位置”，分为：起始/结束位置（航班）、多段分别有起始/结束位置（高铁）、单一位置（课程、活动）、运行时确定的多段分别有不同位置（进程）。同样的，对进程调度中的位置（哪个核），无法事先确定，也是事后记录的。

- 水平轴下方标注的信息为“资源”，分为：单一资源（航班、课程）、多个体资源且考虑次序（高铁）、仅计数量/无需区分个体/无需考虑次序（进程、活动）。
- 左侧黄色标签为计划项的“名字”。

设计一个接口 PlanningEntry<R>用于刻画经过抽象的、可复用的“计划项”，其中 R 代表“计划项”中涉及的资源类型。考虑五个应用对 PlanningEntry<R>的功能需求，将其共性的、可复用的操作抽象出来放入接口。但从上图可以看出，除了“名字”是通用的，其他三方面的属性“位置”、“资源”和“时间”都存在较大的差异。设计 ADT 的时候需要考虑其接口的统一性，并对其差异进行最大程度的抽象以提高复用性。

3.3.1 PlanningEntry<R>的共性操作

从你所选定的应用中进行 ADT 抽象，设计 PlanningEntry 应提供的接口方法，这些方法应当是五个应用的全局共性方法，例如：

- 创建一个新的计划项对象
- 启动
- 取消
- 完成
- 获取计划项的名字
- 获取当前状态
- …但不仅限于上述方法，你可以继续抽象需要的其他全局共性方法

请根据你对应用的理解和抽象，为上述各方法设计名字、参数列表和返回值，给出其 Spec。

进而，设计一个类 CommonPlanningEntry<R>来实现 PlanningEntry<R>。它的 rep 请自行设计，并实现 PlanningEntry 中的所有方法。

3.3.2 各应用的个性化特征

从以上分析可以发现，各应用的个性化特征分为以下五个方面：位置的数量、位置是否可更改、资源特征（数量、个体是否可区分、多个体是否排序）、计划项是否可阻塞及其时间描述、时间是否可预先设定。下表给出了五个应用在这五方面特征上的具体差异，第一行表头中标注的数字表示该列维度上出现的不同情况的数目。

应用	位置的数量（3）	位置是否可更改、可提前设定（3）	资源（3）	是否可阻塞及其时间描述（2）	时间是否可设定（2）
航班	一个起点+一个终点	可提前设定，设定后不可更改	单个可区分资源（飞机，带编号）	否 一个起止时间对	是（创建的时候即可设定）
高铁	一个起点+一组中间位置+一个终点	可提前设定，设定后不可更改	多个带次序的可区分资源（一组前后连接的车厢，均有编号）	是（中途站停车与发车） 一组起止时间对	是（创建的时候即可设定）
进程	一个具体位置	可更改（CPU 核），但执行前无法预先设定	多个不带次序且不需区分 ID 的资源（例如 300M 内存）	是（被挂起和重新执行） 一组起止时间对	否（根据实际执行情况事后记录）
课程	一个具体位置	可更改（教室），可提前设定	单个可区分资源（教师，有 ID）	否 一个起止时间对	是（创建的时候即可设定）
活动	一个具体位置	可更改（会议室），可提前设定	多个不带次序且不需区分 ID 的资源（例如 100 份学习材料）	否 一个起止时间对	是（创建的时候即可设定）

如果各应用在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 PlanningEntry<R>和 CommonPlanningEntry<R>中定义和实现，即实现完全的复用。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。

但是从上表可以看出，在任何一个维度上，五个应用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 PlanningEntry 接口中定义并在 CommonPlanningEntry 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 block(Calendar time)操作，对航班、课程、学习活动这三个不具备“阻塞”功能的计划项来说，就不能使用该 block 操作，故可以在其子类中 override 该 block 操作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大量 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 block 方法不再符合接口中 block 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五个应用的子类中加以实现。例如：针对高铁和进程应用的 PlanningEntry 子类中实现 block 方法，其他三个应用的 PlanningEntry 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“位置数量”维度为例：

特征取值	接口	接口中的方法
一个位置	SingleLocationEntry	setLocation(Location loc)
两个位置	TwoLocationEntry	setLocations(Location start, Location end)

多个位置	MultipleLocationEntry	setLocations(List<Location> locs)

在这种方案下，每个应用的具体计划项子类可以按以下方式加以实现：

```
public class TrainEntry<R> extends CommonPlanningEntry<R>
    implements   MultipleLocationEntry<R>,
                 ChangeableLocationEntry<R>,
                 MultipleSortedResourceEntry<R>,
                 BlockableEntry<R>,
                 TimePresetEntry<R>
```

其中，共性的功能通过继承 CommonPlanningEntry 类实现复用，而特殊的功能则通过 override 五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。

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 CommonPlanningEntry 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 MultipleLocationEntry<R>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 setLocations(...)方法。

```
public class MultipleLocationEntryImpl<R>
                  extends CommonPlanningEntry<R>
                  implements MultipleLocationEntry<R> {
   private List<Location> locations;
    //个性化的 rep，与 CommonPlanningEntry<R>的共性 rep 共同
    //构成整体的 rep

   @Override
   public void setLocations(List<Location> locations) {
      this.locations  = locations; //暂不考虑表示泄露
    }
}
```

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“位置数目”这个维度上的“多位置”特征，再考虑另一个维度“是否可阻塞”的“可阻塞”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“可阻塞”的特征：

```
public class MultipleLocationAndBlockableEntryImpl<R>
                  extends MultipleLocationEntryImpl<R>
                  implements BlockableEntry<R>   {
   private List<Timeslot> timeslots;
   @Override
   public void block() {...}
}
```

如果同时考虑五个维度上的特征，将可能组合出非常多的子类数量，形成了庞大的继承树，给软件系统的维护代码极大困难。

最终，将五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。

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 4，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 MultipleLocationEntry {...}
public interface BlockableEntry {...}
public interface MultipleSortedResourceEntry {...}
...
public class MultipleLocationEntryImpl
                  implements MultipleLocationEntry
public class BlockableEntryImpl
                  implements BlockableEntry
public class MultipleSortedResourceEntryImpl
                  implements MultipleSortedResourceEntry

public interface TrainPlanningEntry
         implements   MultipleLocationEntry,
```

```
                        BlockableEntry,
                        MultipleSortedResourceEntry { }

public class TrainEntry extends CommonPlanningEntry
                    implements TrainPlanningEntry {
   private MultipleLocationEntryImpl mle;
   private BlockableEntryImpl be;
   private MultipleSortedResourceEntryImpl msre;
   public TrainEntry(MultipleLocationEntryImpl mle,
                BlockableEntryImpl be,
                MultipleSortedResourceEntryImpl msre)  {
      ...//设置 delegation 关系
   }
   @Override
   public void setLocations(List<Location> locs){
      mle.setLocations(locs);
   }
   @Override
   public void block(Calendar time) {
      be.block(time);
   }
   ...
}
```

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 CommonPlanningEntry 看作是原始的、未被装饰的计划项实体，将这五个维度看作是五种“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 CommonPlanningEntry 对象逐层装饰五个不同特征，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。

3.3.3 面向各应用的 PlanningEntry 子类型

为了支持五个应用，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 PlanningEntry 子类型：

- FlightEntry：代表一个具有时间表、有起飞和降落机场、由一架飞机执飞的航班；
- TrainEntry：代表一个具有时间表、经过一组站点、由一组车厢构成的高铁车次；
- ProcessEntry：代表一个被操作系统创建、执行、挂起、恢复、结束、取消的进程；
- CourseEntry：代表一次课，具有明确的日期/时间、地点、教师。
- ActivityEntry：代表一次学习活动，具有明确的日期/时间、地点，使用一组学习资料。

3.3.4 PlanningEntry 及其所有子类型的测试与注释

针对以上完成的 PlanningEntry、CommonPlanningEntry 和所有子类型，按实验 2 的要求撰写 AF、RI、Safety from rep exposure，以及每个方法的 specification（precondition、post-condition 等）。

为它们设计和编写 JUnit 测试用例，并撰写 testing strategy。

后续各节同样要求，不再重复。

3.4 面向复用的设计：R

PlanningEntry<R>中的泛型参数 R，可以是你所设计的任何表达资源的类。

对五个应用来说，其资源所需关注的属性可从 3.2 和 3.3 节的描述中抽取。请据此完成 R 的具体类的设计和实现。

- 航班应用中的资源是“飞机”，属性包括飞机编号、机型号（A350、B787、C919 等）、座位数、机龄（例如 2.5 年）。
- 高铁应用中的资源是“车厢”，属性包括车厢唯一编号、类型（商务、一等、二等、软卧、硬卧、硬座、行李车、餐车）、定员数（例如 100 人）、出厂年份。
- CPU 进程应用中的资源是“内存”，属性仅“计量单位”（例如 M、G 等）。
- 课表应用中的资源为“教师”，属性包括身份证号、姓名、性别、职称。

- 学习活动应用中的资源为“学习材料”，属性包括材料名称、发布部门、发布日期。

可以看出，上述五种资源的差异很大。你可以分别为它们设计 ADT，无需进行抽象。上述五种资源类都应该是 immutable 的。

3.5 面向复用的设计：Location

PlanningEntry 的某些子类型的 rep 中不可避免的需要表达“位置”信息。需设计 Location 类，它可以是接口、抽象类或具体类，是 immutable 的。

一个“位置”对象的属性包括：经度、纬度、名称、是否可共享使用。所谓的“是否可共享”是指：该位置是否可同时被多个计划项所使用。例如：“北京南”高铁站是可共享的，多个高铁车次可在同一时间停在或经过该站；“D01 教室”是不可共享的，同一时间只能有一个课程在此上课。

但是，应用 3 中的“位置”是“某个 CPU 核”，它无需使用经度和纬度加以描述，只需使用名称区分即可。

请自己设计关于 Location 的 ADT 及其子类型，并仍然从 LSP、可复用和可维护的角度评估你的设计。

3.6 面向复用的设计：Timeslot

PlanningEntry 的某些子类型的 rep 中需要表达“时间属性”，不妨为一个单独的“起止时间对”设计 Timeslot 类，它是一个带有起始时间和结束时间的 ADT，应包含日期（年/月/日）和时间（时/分），符合 yyyy-MM-dd HH:mm 的语法规则。例如：

```
(2020-03-01 12:00, 2020-03-01 14:00)
```

请设计其 rep 和方法，并实现之。该类也是 immutable 的。

3.7 面向复用的设计：EntryState

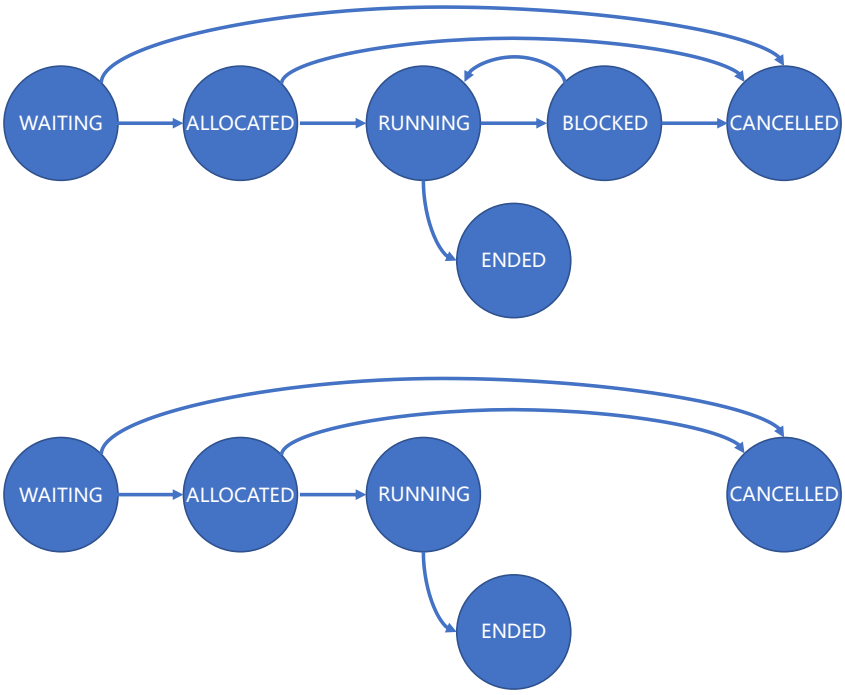
PlanningEntry 及其子类型需要记录计划项实体的当前“状态”。请使用 state 设计模式来完成该任务。请自行思考设计方案。

下表给出了各个应用中的计划项实体需要管理的状态：

应用	状态	说明
航班	未分配资源（WAITING）	航班其他信息均已确定，但未分配飞机
	已分配资源（ALLOCATED）	已分配具体飞机，但未从出发机场起飞
	已启动（RUNNING）	航班已起飞
	已完成（ENDED）	航班已降落目的地机场
	已取消（CANCELLED）	航班未起飞之前可被取消，不再执飞

高铁	未分配资源（WAITING）	车次信息已确定，但未分配车厢
	已分配资源（ALLOCATED）	分配了一组车厢
	已启动（RUNNING）	已从起始站发车
	挂起中（BLOCKED）	已抵达某个中间站，并停车
	已完成（ENDED）	抵达终点站
	已取消（CANCELLED）	高铁在起始站未发车之前可被取消，或者在某个中间站停车期间被取消
进程	未分配资源（WAITING）	OS 创建了进程
	已分配资源（ALLOCATED）	为进程分配了内存资源
	已启动（RUNNING）	为进程分配了特定核，在该核上启动执行
	挂起中（BLOCKED）	进程被挂起，暂停执行
	已完成（ENDED）	结束执行结束
	已取消（CANCELLED）	进程在启动之前可被取消，或挂起期间可被取消
课程	未分配资源（WAITING）	新建立了课程安排，并确定了时间地点
	已分配资源（ALLOCATED）	为某课程安排了教师
	已启动（RUNNING）	开始上课
	已完成（ENDED）	下课
	已取消（CANCELLED）	该次课程启动之前可被取消
学习活动	未分配资源（WAITING）	新建立了学习活动，并确定了时间地点
	已分配资源（ALLOCATED）	为活动安排了所需的学习材料
	已启动（RUNNING）	开始学习活动
	已完成（ENDED）	结束学习活动
	已取消（CANCELLED）	学习活动启动之前可被取消

下图给出了这些状态之间的合法转换关系。不存在箭头的两个状态之间不能转换。上半部分针对的是高铁和进程两个应用（有 BLOCKED 状态），下半部分针对的是其他三个应用（无 BLOCKED 状态）。



3.8 面向应用的设计：Board

每个特定位置都有一个信息板，用于实时公布该位置上过去发生过的和后续即将发生的计划项，以便于广大用户及时了解计划项的状态变化。针对本实验的五个应用，相应的信息板如下所述：

(1) 某机场里的航班状态显示屏：分为抵达屏和出发屏，前者展示过去 1 小时内和未来 1 小时内抵达该机场的航班信息及其状态，后者展示过去 1 小时内和未来 1 小时内从该机场出发的航班信息及其状态。注意：第一列里的所有时间都是计划降落时间（对抵达航班来说）、计划起飞时间（对出发航班来说）。

2020-02-24 8:00 (当前时间)，哈尔滨机场			
抵达航班			
7:01	CA1001	北京-哈尔滨	已降落
7:20	CZ6001	上海-哈尔滨	已取消
8:05	ZH9001	深圳-哈尔滨	即将降落
…			
出发航班			
7:00	CA1002	哈尔滨-北京	已起飞
7:50	CZ6002	哈尔滨-上海	已取消
8:10	ZH9002	哈尔滨-深圳	即将起飞

…			

(2) 高铁站里的车次状态显示屏：与(1)类似，唯一的区别是要额外考虑经停高铁车次。

2020-02-24 8:00 (当前时间)，哈尔滨西站			
抵达车次			
7:01	G1020	牡丹江-北京	已到达
8:20	G2020	北京-哈尔滨	即将抵达
…			

(3) CPU 某个核上的进程清单：列出过去 5 分钟内的进程清单，包括进程名和执行起止时间。如果正在执行，则无需给出结束时间。该表里的所有时间都是实际执行时间。

2020-02-24 8:00 (当前时间)，核 1	
7:55-7:56	微信
7:56-7:57	QQ
7:57-7:58	微信
7:58- //正在执行，无结束时间	微博

(4) 教室外悬挂的教室占用情况表：列出当日在本教室内上的所有课程及其状态。与(1)相同，第一列的时间都是计划时间。

2020-02-24 9:00，正心楼 23 教室			
8:00-9:45	软件构造	徐汉川	正在上课
10:00-11:45	电路设计	张三	已取消
13:45-15:30	英语	李四	计划
…			

(5) 会议室外悬挂的会议室预订表：与(4)同。

注：为了降低工作量，`Board` 相关功能的设计和实现无需面向复用，你只需要针对 5 个应用各自的 `PlanningEntry` 子类型（见 3.3.3 节）分别开发各自具体的 `Board` 类即可，无需进行抽象设计。

3.9 外部 API 复用

使用 `JSwing/JTable` 或其他第三方 Java 类库所提供的绘制表格 API，为你的三个 `Board` 类型添加数据可视化功能。`JSwing/JTable` 是基于 `GUI` 的，如果

你不想做 GUI，也可以在命令行界面中使用第三方 API 完成基于字符的可视化。实在不想用或学不会第三方 API，也可以自己从 0 开始实现表格形式的可视化。

在各 Board 类中实现以下方法：

```
public void visualize()
```

请自学你所选定的表格 API，并将所需的 jar 包添加至你的 lib 目录。

3.10 可复用 API 设计

基于 PlanningEntry<R>接口及其子类型的具体定义和代码实现，设计以下 API 并完成代码。基于 façade 设计模式，将它们放在一个新的 PlanningEntry APIs 类中。你可以实现

(1) 检测一组计划项之间是否存在位置独占冲突：如果两个计划项在同一时间点上占用了不可共享的位置，那么就存在了位置冲突。例如：某次《软件构造》课的时间是本日 8:00-10:00，而某次《计算机系统》课是在本日 9:00-11:00，二者均在 D01 教室，而“教室”这种位置是不可共享的，那么在 9:00-10:00 这个时间段里就存在着位置冲突。但是对机场、车站等可共享位置来说，则不会存在位置冲突（假设不考虑位置的容量问题）。

```
public boolean checkLocationConflict(List<PlanningEntry>
                                     entries)
```

(2) 检测一组计划项之间是否存在资源独占冲突：如果两个计划项在同一时间点上占用了同样的资源，那么就存在了资源冲突，例如同一个教师、同一个车厢、同一架飞机。对 ROM、学习资料等不可区分个体的资源，无需考虑资源独占冲突。

```
public boolean
checkResourceExclusiveConflict(List<PlanningEntry> entries)
```

(3) 提取面向特定资源的前序计划项：针对某个资源 r 和使用 r 的某个计划项 e，从一组计划项中找出 e 的前序 f，f 也使用资源 r，f 的执行时间在 e 之前，且在 e 和 f 之间不存在使用资源 r 的其他计划项。若不存在这样的计划项 f，则返回 null。如果存在多个这样的 f，返回其中任意一个即可。对 ROM、学习资料等不可区分个体的资源，该 API 不适用。

```
public PlanningEntry findPreEntryPerResource(R r,
    PlanningEntry e, List<PlanningEntry> entries)
```

3.11 设计模式应用

在以上的各节的要求中，你已经练习了 decorator、façade、state 三种设计模式。请在你的 ADT 设计和五个应用开发中使用更多的设计模式，对之前各节的

开发结果进行扩展。

(1) 工厂方法 factory method

请使用 factory method 模式，使得五个应用中创建具体 `PlanningEntry` 子类型对象时，尽可能避免通过 `new` 的方式实例化具体的 `PlanningEntry` 子类型。

(2) 迭代器 iterator

请为你的 `Board` 类增加迭代器功能，可通过迭代器按照时间从早到晚的次序遍历其中包含的所有计划项。为了方便支持“按时间对计划项排序”，请在你的 `PlanningEntry` 及其子类型中增加“比较”功能（使用 `Java` 提供的 `Comparator` 或 `Comparable` 接口，第一次习题课已经介绍过）。

```
Iterator<PlanningEntry> iterator = board.iterator();
          // board 是一个类型为 Board 的对象
while (iterator.hasNext()) {
   PlanningEntry pe = iterator.next();
   System.out.println(...);
}
```

(3) 策略模式 strategy

在 3.10 节的 API 设计中，请选定其中一个 API，使用两种不同的算法实现之，进而在客户端应用程序中使用 strategy 模式灵活替换不同的算法。

3.12 应用设计与实现

为五个应用场景分别开发程序，实现以下功能。可以实现为 GUI 程序，也可以是命令行程序。

- 用户提供必要信息，管理（增加、删除）可用的资源；
- 用户提供必要信息，管理（增加、删除）可用的位置；
- 用户提供必要信息，增加一条新的计划项；
- 用户提供必要信息，取消某个计划项；
- 用户提供必要信息，为某个计划项分配资源；
- 用户提供必要信息，启动某个计划项；
- 用户提供必要信息，以变更某个已存在的计划项的位置；
- 用户提供必要信息，以阻塞/挂起某个计划项；
- 用户提供必要信息，以重启动某个已挂起的计划项；
- 用户提供必要信息，结束某个计划项；
- 用户选定一个计划项，查看它的当前状态；
- 检测当前的计划项集合中可能存在的位置和资源独占冲突（参见 3.10 节）；
- 针对用户选定的某个资源，列出使用该资源的所有计划项（包含尚未开

始执行的、执行中的、已经结束的）。用户选中其中某个计划项之后，可以找出它的前序计划项（参见 3.10 节）；

- 选定特定位置，可视化展示当前时刻该位置的信息板（参见 3.9 节）。

五个应用的主程序类，请分别命名如下：

- `FlightScheduleApp`
- `TrainScheduleApp`
- `ProcessManagerApp`
- `CourseCalendarApp`
- `ActivityCalendarApp`

说明 1：在不同应用中，以下这些功能的具体含义是不同的，某些功能在某些应用中可能不适用（例如无法变更某个航班/某个车次的位置信息），请根据前面对各个应用的具体描述做出仔细甄别。

说明 2：CPU 进程调度应用因为无法提前设定时间，需要记录开始的时间、历次挂起和重启动的时间、结束的时间。一个进程执行结束后，用户可查看它的执行历史（何时到何时在哪个核上执行）。对其他应用来说，除了“取消”的时间需要记录，其他动作的时间都不需要记录和管理。

说明 3：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）。

说明 4：有同学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。这其实就是在检查你的 ADT 设计是否有足够的可复用性和适应性，如果 ADT 设计地好，应用开发的工作量会大幅度减少。

3.13 基于语法的数据读入

针对 3.12 节中开发的“航班”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，从中抽取一组航班信息，构造一组航班计划项实体集合。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：

https://github.com/rainywang/Spring2020_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不同航班的变化信息。“//”后续文字是解释说明，并非语法的构成部分。

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，须结束读取并提示用户选择其他合法的文件。各行的次序是确定的，违反该次序也是非法文件。后续 Lab4 里会着重考虑对非法语法的处理。

另外，请忽略文件中的所有缩进，它并非语法的组成部分。

```
Flight:2020-01-01,CA1001  //航班计划项 1
```

```
                //逗号前为航班日期，遵循 yyyy-MM-dd 格式
                //逗号后为航班号，由两位大写字母和 2-4 位数字构成
{
    DepartureAirport:Harbin    //出发机场，word
    ArrivalAirport:Beijing     //抵达机场，word
    DepatureTime:2020-01-01 12:00     //起飞时间
    ArrivalTime:2020-01-01 14:00  //降落时间
              //这两个时间均为 yyyy-MM-dd HH:mm 的格式
              //起飞时间中的日期必须与第一行的日期一致
              //抵达时间中的日期可以与起飞日期一致，也可以晚 1 天
              //例如 2020-01-01 23:00 起飞，2020-01-02 01:00 降落
    Plane:B9802 //飞机编号，第一位为 N 或 B，后面是四位数字
    {
      Type:A350 //机型，大小写字母或数字构成，不含有空格和其他符号
      Seats:300 //座位数，正整数，范围为[50,600]
      Age:2.5    //机龄，数字，范围是[0,30]，可最多 1 位小数或无小数
                 //诸如 0、0.0、2、2.0、20.0 这样的表述都是对的
                 //但 02.0、2.12 这样的表述则不符合规则
    }
}
Flight:2020-01-01,CA1002  //航班计划项 2
          //在一个文件中不允许出现日期一样且航班号一样的两个计划项
          //但允许出现“日期不同、航班号相同”的情况
          //也允许出现“日期相同、航班号不同”的情况
          //同一个航班号，虽然日期可以不同，但其出发和到达机场、
          //出发和到达时间均应相同
          //注意：CA0001、CA001、CA01 是相同的航班号
{
  ...
}
```

3.14 新的变化

3.12 节中开发出的应用，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

“314change”，在该分支上完成本节任。

- 航班支持经停，即包含最多 1 个中间经停机场，例如哈尔滨-威海-深圳。
- 如果高铁车次已经分配了车厢资源，则不能被取消。
- 如果某任务已经在某 CPU 核上执行超过 n 秒，则应自动将其挂起。
- 课程可以有多个教师一起上课，且需要区分次序（即多名教师的优先级）。
- 活动开始执行之后，可以被阻塞（临时暂停），后续继续进行（位置、资源均不变）。

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。

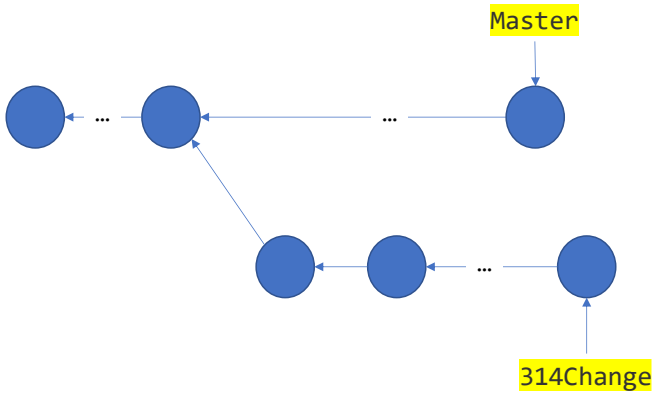
注意 1：除非这些变化修改了之前各节提及的某些功能，你针对这些变化所做的修改都不应破坏之前已经写好的功能。

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 `314change` 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 314change 创建新分支
…按上面的要求进行代码修改…
git add *
git commit -m "314change"在该分支上提交
git checkout master                切换回 master 分支
…请不要使用 git merge 314change 进行合并修改
…请不要使用 git branch -d 314change 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。



3.15 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 12 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

TA 在第 8-12 周实验课上现场验收：学生做完实验之后，向 TA 提出验收申请，TA 根据实验要求考核学生的程序运行结果并打分。现场验收并非必需，由

学生主动向 TA 提出申请。

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。