

**2024年春季学期**

**计算学部《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 刘云宏 |
| 学号 | 2022113562 |
| 班号 | 2237101 |
| 电子邮件 | 2909741643@qq.com |
| 手机号码 | 15906023016 |

**目录**

[1.实验目标概述 3](#_Toc5940)

[2.实验环境配置 3](#_Toc7276)

[3.实验过程 3](#_Toc15912)

[3.1 待开发的三个应用场景 3](#_Toc17661)

[3.2 面向可复用性和可维护性的设计：IntervalSet<L> 4](#_Toc11217)

[3.2.1 IntervalSet<L>的共性操作 4](#_Toc1799)

[3.2.2 局部共性特征的设计方案 4](#_Toc319)

[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） 5](#_Toc32315)

[3.3 面向可复用性和可维护性的设计：MultiIntervalSet<L> 5](#_Toc22226)

[3.3.1 MultiIntervalSet<L>的共性操作 5](#_Toc26396)

[3.3.2 局部共性特征的设计方案 6](#_Toc3579)

[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） 7](#_Toc24662)

[3.4 面向复用的设计：L 7](#_Toc27090)

[3.5 可复用API设计 7](#_Toc2920)

[3.5.1 计算相似度 7](#_Toc28021)

[3.5.2 计算时间冲突比例 8](#_Toc12308)

[3.5.3 计算空闲时间比例 8](#_Toc13905)

[3.6 应用设计与开发 8](#_Toc24573)

[3.6.1 排班管理系统 8](#_Toc31171)

[3.6.2 课表管理系统 12](#_Toc17476)

[3.7 基于语法的数据读入 14](#_Toc18612)

[4.实验进度记录 15](#_Toc19769)

[5.实验过程中遇到的困难与解决途径 15](#_Toc4380)

[6.实验过程中收获的经验、教训、感想 16](#_Toc31824)

[6.1 实验过程中收获的经验和教训 16](#_Toc21645)

[6.2 针对以下方面的感受 17](#_Toc760)

# 1.实验目标概述

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：

（1）子类型、泛型、多态、重写、重载；（2）继承、代理、组合；（3）语法驱动的编程、正则表达式；（4）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），我们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

在idea编译器中利用Java环境完成项目；在GitHub上根据要求生成仓库，并使用git进行项目的提交与版本控制，在配置环境中没有遇到什么问题。

我的GitHub Lab3仓库的URL地址（Lab3-学号）如下：

https://github.com/ComputerScienceHIT/HIT-Lab3-2022113562.git

# 3.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！）。

## 3.1 待开发的三个应用场景

值班表管理（DutyRoster）：

每天只能安排唯一一个员工值班，且员工需要连续值班m天。需要记录员工的姓名、职位和手机号码，以便外界联系。考虑到员工可能在不同时间段内值班，需要维护一个值班表，记录每个时间段内值班的员工信息。

操作系统进程调度管理（ProcessSchedule）：

操作系统需要决定在特定时间执行哪个进程，可以随机调度。每个时间段只能有一个进程在执行，其他进程处于休眠状态。CPU在某些时刻可能处于闲置状态，没有进程在执行。需要考虑进程的挂起和恢复执行，以及进程的执行时间段。

大学课表管理（CourseSchedule）：

课表可能会周期性地重复，例如每周同一时间上相同的课程。同一时间段可以安排不同的课程。教室和教师的安排也需要考虑，可能一位教师会承担多门课程。可能存在空白时间段，即某些时间段没有安排课程。

在共性的实现上，考虑到他们三个都是需要使用一个包含多个时间段的表来实现并展示所需要的计划安排，所以我们可以设计一个通用的管理系统，它能够灵活应对不同场景的需求。可以通过抽象出共性的概念和功能来实现这个系统。然后针对不同的应用场景，实现具体的逻辑和功能，同时利用通用的资源调度和管理系统来支持这些功能的实现。

## 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

接口IntervalSet<L>定义了一组操作，用于管理时间段集合。这些操作包括：

empty(): 静态方法，用于创建一个空的时间段集合对象。

SetStartStop(long startTime, long stopTime): 对时间段集合的起止时间进行初始化。

GetStartTime(): 返回时间段集合的起始时间。

GetStopTime(): 返回时间段集合的终止时间。

insert(long start, long end, L label): 在当前对象中插入新的时间段和标签。

labels(): 返回时间段集合中所有的标签集合。

remove(L label): 移除时间段集合中某个标签所关联的时间段。

start(L label): 返回某个标签对应的时间段的开始时间。

end(L label): 返回某个标签对应的时间段的结束时间。

accept(MultiIntervalSet<L> multiIntervalSet): 与 MultiIntervalSet 类实现双向链接，用于支持一个标签有多个时间段的功能。（但该方法实际上不需要用户去使用）

intervals(L label): 返回对应输入的标签名的所有时间段集合。

这些操作定义了时间段集合的基本功能，允许用户管理和操作时间段及其关联的标签信息。

### 3.2.2 局部共性特征的设计方案

这个设计方案使用了实验指导中推荐的第六个，使用装饰器模式来增强 IntervalSet接口的功能，以满足不同场景下的需求。以下是局部共性特征的设计方案：

（1）基本装饰器类 (BasicIntervalSet)：

基本装饰器类实现了IntervalSet接口，并包含一个成员变量intervalSet，用于持有被装饰的对象。

实现了IntervalSet接口中的所有方法，并将它们委托给intervalSet对象。

在构造方法中对传入的intervalSet进行非空检查，确保装饰器对象的合法性。

（2）不允许空白时间段的装饰器类 (NoBlankIntervalSet)：

继承了基本装饰器类，进一步增强了时间段集合的功能。

添加了checkRep()方法用于检查时间段集合中是否存在空白时间段，如果存在则抛出异常。

在toString()方法中调用checkRep()方法，确保在输出对象字符串表示之前进行了合法性检查。

（3）不允许时间段重叠的装饰器类 (NonOverlapIntervalSet)：

同样继承了基本装饰器类，增加了时间段集合中不允许有重叠时间段的功能。

重写了insert()方法，在插入新的时间段后进行检查，如果有时间段重叠则抛出异常。

添加了checkRep()方法用于检查时间段集合中是否存在重叠时间段，如果存在则抛出异常。

在toString()方法中调用checkRep()方法，确保在输出对象字符串表示之前进行了合法性检查。

通过使用装饰器模式，这些装饰器类可以在运行时动态地添加功能，而不需要改变被装饰对象的结构，从而实现了代码的可扩展性和灵活性。

### 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）

我在排班表管理系统中使用IntervalSet，如下：

IntervalSet<L> dutyRosterIntervalSet = new NoBlankIntervalSet<L>(new NonOverlapIntervalSet<L>(new BasicIntervalSet<L>(IntervalSet.empty())));

就是使用的是底层的IntervalSet时间段集合，并使用装饰器，为其装饰上了不允许重叠和不允许空白的新功能。

## 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

创建空对象 (empty())：静态方法，用于创建一个空的 MultiIntervalSet 对象。返回一个新创建的空对象，其泛型类型为 L，要求 L 是不可变类型。

创建非空对象 (ConcretetMultiIntervalSet(IntervalSet<L> initial))：方法用于利用初始数据创建一个非空对象。接受一个初始的时间段集合 initial，并返回利用这个初始数据创建的非空对象。

对时间段集合的起止时间进行初始化 (SetStartStop(long startTime, long stopTime))：用于设置时间段集合的起始时间和终止时间。参数 startTime 表示时间段集合的起始时间，参数stopTime表示时间段集合的终止时间。

返回时间段集合的起始时间 (GetStartTime())：返回时间段集合的起始时间。

返回时间段集合的终止时间(GetStopTime())：返回时间段集合的终止时间。

插入新的时间段和标签 (insert(long start, long end, L label))：在当前对象中插入新的时间段和标签。参数 start 表示时间段的开始时间，参数 end 表示时间段的结束时间，参数 label 表示标签名。如果开始时间大于等于结束时间，抛出异常提醒客户端；如果输入的时间段已被使用，也抛出异常提醒客户端。

获取当前对象中的标签集合(labels())：返回一个集合，其中包含了当前对象中的所有标签。

移除某个标签所关联的时间段(remove(L label))：从当前对象中移除某个标签所关联的时间段。参数label表示标签名。首先检测当前对象中是否存在这个标签，有则删除；否则抛出异常提醒客户端。

获取与某个标签所关联的所有时间段 (intervals(L label))：返回当前对象中与指定标签关联的所有时间段。参数label表示标签名。返回的所有时间段按开始时间从小到大进行排序。

### 3.3.2 局部共性特征的设计方案

这个设计方案使用了实验指导中推荐的第六个，使用装饰器模式来增强 IntervalSet接口的功能，以满足不同场景下的需求。以下是局部共性特征的设计方案：

（1）BasicMultiIntervalSet:

这是所有装饰器的基类，实现了 MultiIntervalSet 接口，并包含了基本的时间段集合操作。

构造函数接受一个 MultiIntervalSet 对象作为参数，并将其保存在私有字段中。

对于所有方法，它们都是简单地调用基类对象的相应方法。

（2）NoBlankMultiIntervalSet:

这个装饰器确保时间段集合中不存在空白区域。

它在构造函数中调用了基类的构造函数，并重写了 toString() 方法以添加检查空白的逻辑。

checkRep() 方法用于检查时间段集合中是否存在空白，如果存在，则抛出异常。

（3）NonOverlapMultiIntervalSet:

这个装饰器确保不同标签对应的时间段不会重叠。

它在构造函数中调用了基类的构造函数，并在插入新时间段后重写了 insert() 方法以检查时间段是否重叠。

checkRep() 方法用于检查时间段集合中是否存在重叠，如果存在，则抛出异常。

（4）PeriodicMultiIntervalSet:

这个装饰器允许时间段集合中的部分或全部标签对应的时间段具有周期性。

它在构造函数中调用了基类的构造函数，并在插入新时间段后重写了 insert() 方法以添加周期性属性。

它还提供了一些额外的方法，如 SetPeriodic() 用于设置时间段的周期性，Periodicity() 用于检查时间段是否具有周期性，labelIntervalHours() 用于计算标签在给定时间段内的时间占用情况。checkPeriodic()方法用于将周期性时间段添加到整个时间轴上。

这些装饰器类各自解决了不同的需求，并且通过继承和重写基类的方法来实现特定的功能。同时，它们都实现了相应的不变性检查，确保了时间段集合的合法性和一致性。

### 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）

我在排班表管理系统中使用MultiIntervalSet，如下：

PeriodicMultiIntervalSet<L> CourseScheduleSet = new PeriodicMultiIntervalSet<L>(new BasicMultiIntervalSet<L>(MultiIntervalSet.empty()));

就是使用的是底层的MultiIntervalSet时间段集合为基础，并使用装饰器，为其装饰上了允许周期的新功能。

## 3.4 面向复用的设计：L

将上述所描述到的各种时间段集合的类型都泛化为L，以便应对各种应用场景的使用，例如我在后续使用到的排班系统中的员工，课表管理系统中的课程等等，就能是设计出的各种可变的对象。

## 3.5 可复用API设计

### 3.5.1 计算相似度

答：我选择把整个完整的时间段按单位一进行分割，紧接着分别遍历两个时间段集合对象，把在这个单位时间段内对应的标签分别存储到一个Set集合中，再比较这两个集合中有没有相同的标签，有的话这个单位时间段就算是相似的了，在相似长度上加一，最后用相似的长度除以总长度即可得到最终的答案。

### 3.5.2 计算时间冲突比例

答：（1）对于IntervalSet对象：首先还是选择针对单位的时间段，遍历时间段集合对象中的所有标签，如果它对应有的时间段包含了该单位时间段（set.start(label)<=time && set.end(label)>=time+1），那么使冲突的标识conflict++，

循环完这一轮所有标签集后就判断一下conflict的值，如果大于1了，则说明这个单位时间段是冲突的，在冲突长度上加一，最后用冲突的长度除以总长度即可得到最终的答案。

（2）对于MultiIntervalSet对象：其实该方法跟IntervalSet对象的大差不差，唯一的不同就是MultiIntervalSet对象允许同一个标签可以对应多个时间段，所以在遍历时间段集合的所有标签时，要调用intervals方法获取其所对应的所有时间段，再依次判断是否包含了该目标单位时间段，而后的步骤就跟IntervalSet对象的处理方法是一样的了。

### 3.5.3 计算空闲时间比例

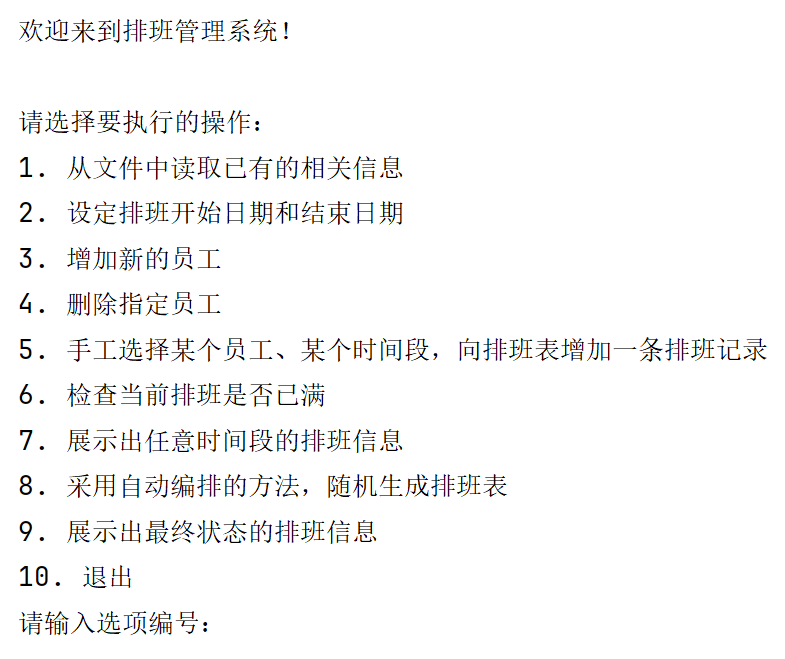
答：（1）对于IntervalSet对象：首先初始化一个空闲时间长度为整个时间段集合的长度，也还是选择针对单位的时间段，遍历时间段集合对象中的所有标签，如果它对应有的时间段包含了该单位时间段（set.start(label)<=time && set.end(label)>=time+1），那么使冲突的标识freetime++，循环完这一轮所有标签集后就判断一下freetime的值，如果大于0了，则说明这个单位时间段不是空闲的，在空闲时间长度上减一，最后用空闲时间的长度除以总长度即可得到最终的答案。

（2）对于MultiIntervalSet对象：该方法还是跟IntervalSet对象的类似，唯一的不同就是MultiIntervalSet对象允许同一个标签可以对应多个时间段，所以在遍历时间段集合的所有标签时，要调用intervals方法获取其所对应的所有时间段，再依次判断是否包含了该目标单位时间段，而后的步骤就跟IntervalSet对象的处理方法是一样的了。

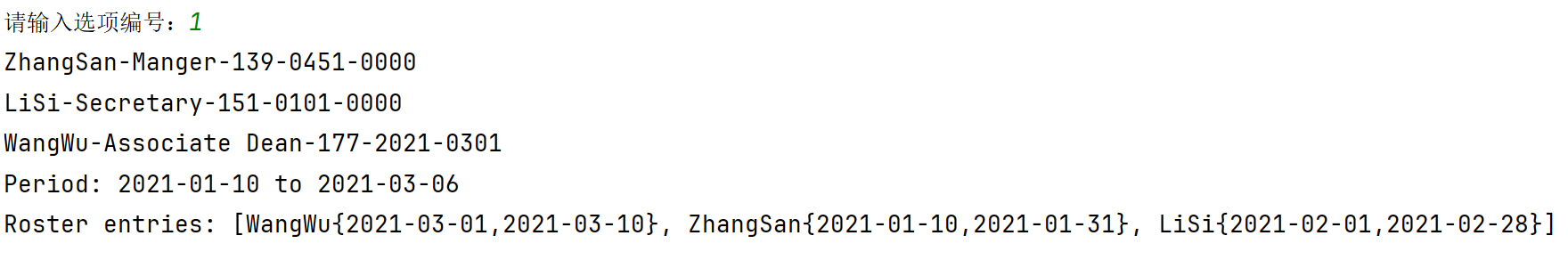
## 3.6 应用设计与开发

### 3.6.1 排班管理系统

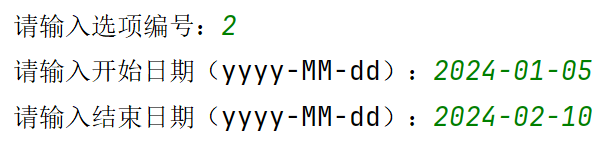
先展示一下最终实现的功能：



1. 从文件中读取已有的相关信息： 用户选择此选项后，系统会从指定的文件中读取已有的员工信息、排班信息以及排班开始日期和结束日期。

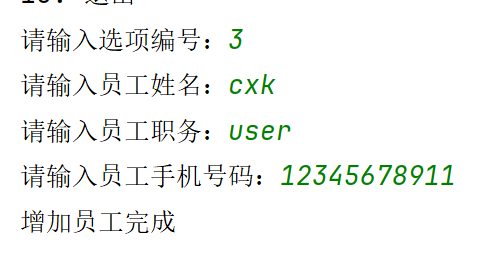


1. 设定排班开始日期和结束日期： 用户选择此选项后，系统会要求用户输入排班开始日期和结束日期，以便确定排班的时间范围。



若原来的排班表已经设定好时间，它会进行更新。

1. 增加新的员工： 用户选择此选项后，系统会要求用户输入新员工的姓名、职务和手机号码，并将其添加到员工列表中。



1. 删除指定员工： 用户选择此选项后，系统会列出当前存在的员工，并要求用户输入要删除的员工的编号。如果该员工已经被排班安排，则需要先删除其排班记录才能删除员工信息。

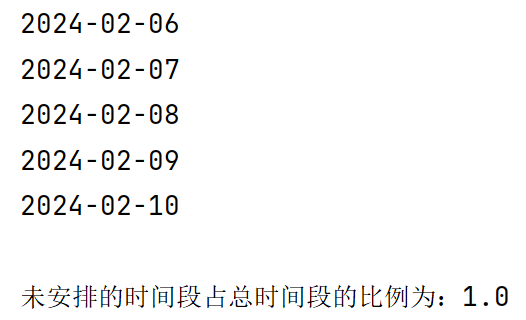
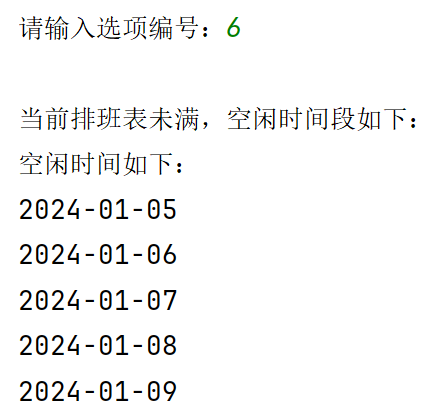


1. 手工选择某个员工、某个时间段，向排班表增加一条排班记录： 用户选择此选项后，系统会列出当前存在的员工，并要求用户选择要安排排班的员工和时间段，并将排班记录添加到排班表中。



因为在之前文件读取信息时已经为其安排好了工作时间段，所以不能再次安排。

1. 检查当前排班是否已满： 用户选择此选项后，系统会检查当前排班表是否已满员。如果未满，系统会展示未安排的时间段，并显示未安排时间段占总时间段的比例。



1. 展示出任意时间段的排班信息： 用户选择此选项后，系统会要求用户输入要查看的时间段范围，并展示该时间段内的排班信息。



1. 采用自动编排的方法，随机生成排班表： 用户选择此选项后，系统会根据当前的排班状态和员工信息，自动生成排班表。



仅展示出一部分结果，实际上它会排满整个时间段集合。

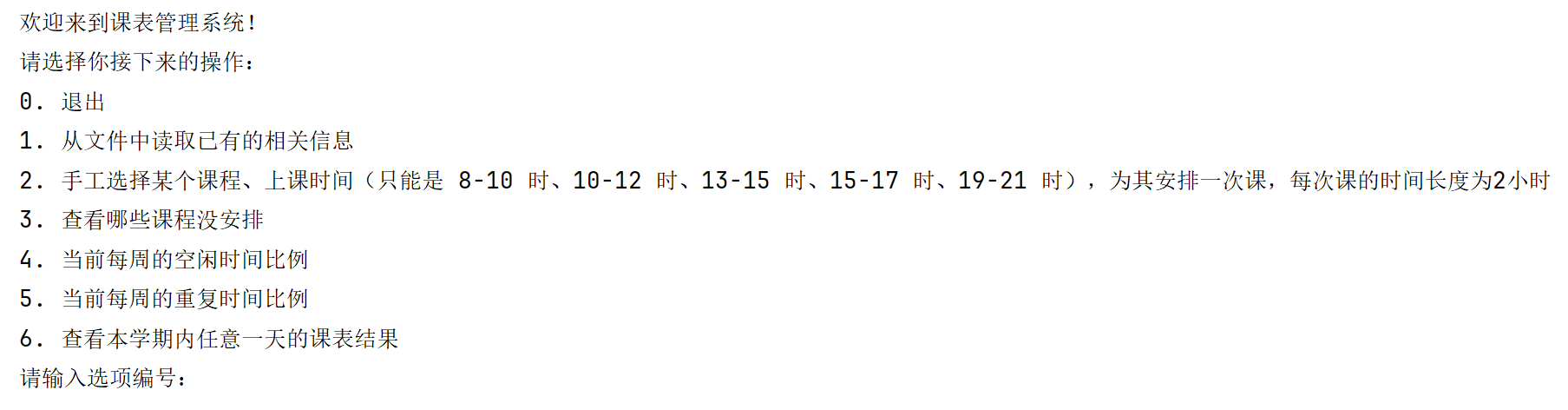
1. 展示出最终状态的排班信息： 用户选择此选项后，系统会展示排班管理系统的最终状态，包括员工信息、排班表状态等。



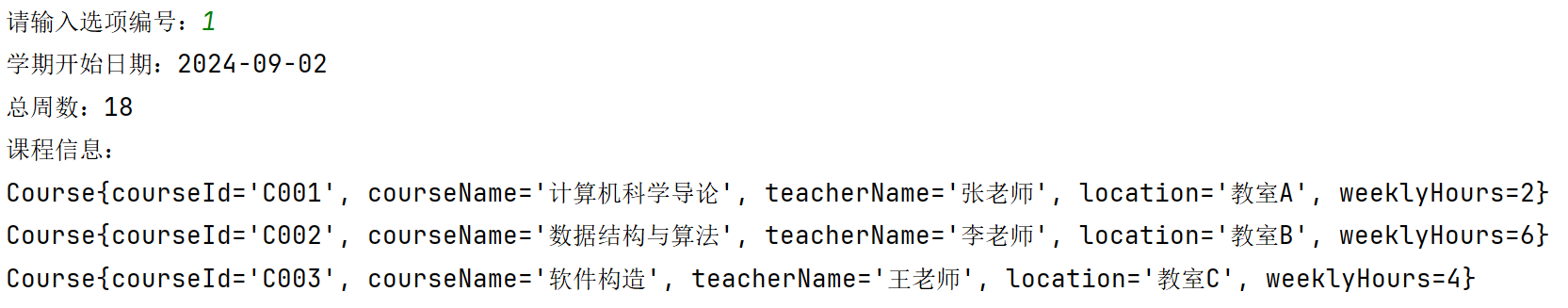
实际上也是会展示全的，若还含有空白，他会提醒你并不进行任何展示。

10、退出： 用户选择此选项后，系统会退出排班管理系统。

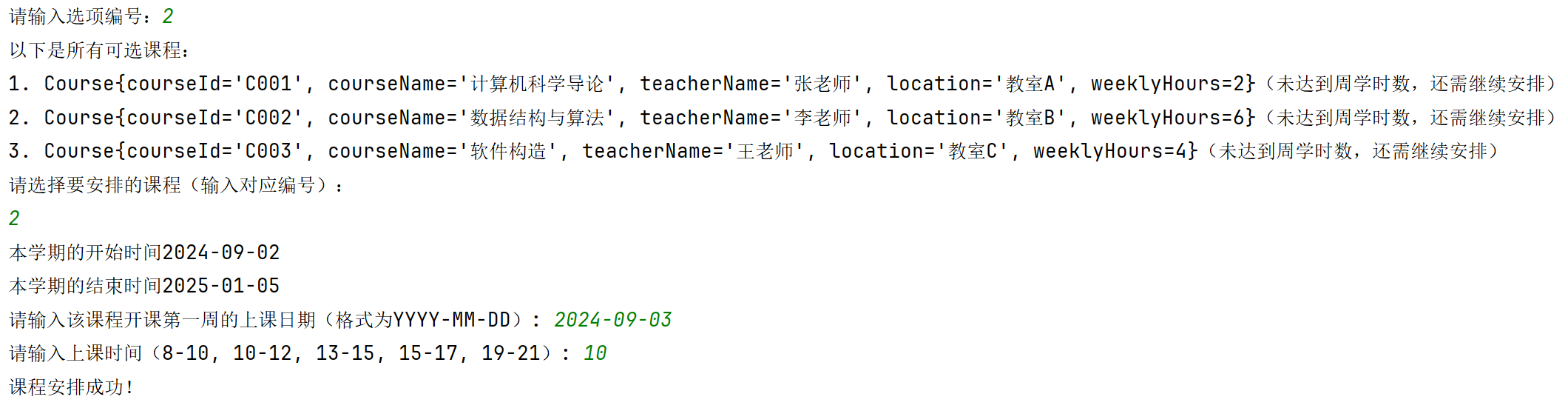
### 3.6.2 课表管理系统



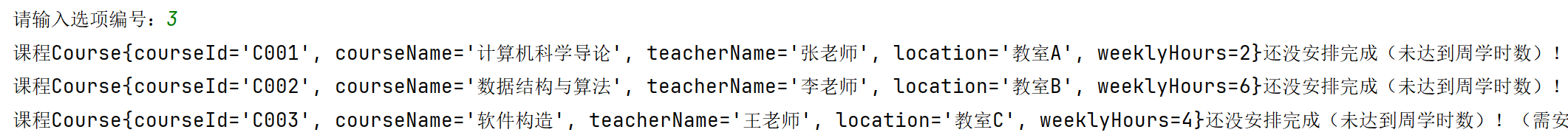
1. 从文件中读取已有的相关信息 用户选择此选项后，程序将读取指定文件中的课程信息，包括学期开始日期、总周数以及各门课程的详细信息。读取的信息将用于后续的课程安排和管理。

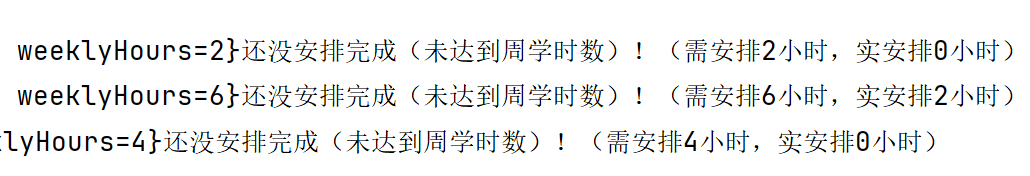


1. 手工选择某个课程并安排一次课 用户可以手动选择要安排的课程和上课时间段。系统会检查课程的周学时数，如果未达到要求，则可以安排新的课程。每次安排的课程时间长度为2小时，直到达到周学时数目时，该课程将不能再安排。



1. 查看哪些课程没安排 用户选择此选项后，系统会显示尚未安排课程的列表。系统会检查每门课程的安排情况，如果周学时数未达到要求，则该课程会被列出。





1. 当前每周的空闲时间比例 用户选择此选项后，系统会计算并显示当前每周的空闲时间比例。系统会统计每周未安排课程的时间段，并计算空闲时间所占比例。



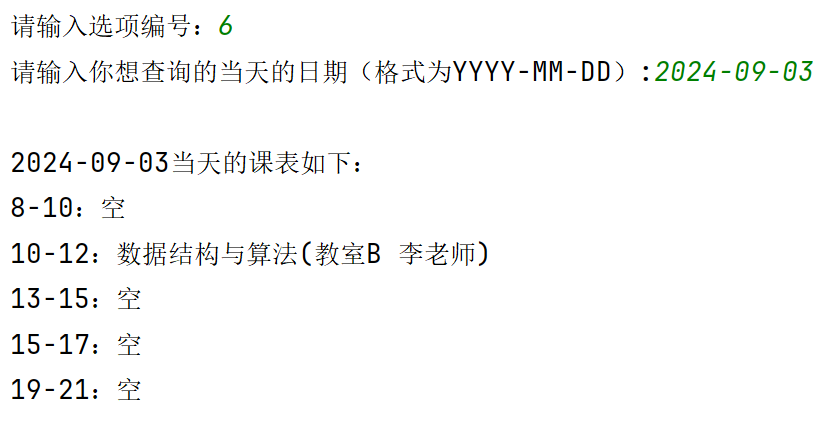
因为一开始只在第一周添加了一个周期为18的课程，所以十八周的空闲时间比例都是一样的。

1. 当前每周的重复时间比例 用户选择此选项后，系统会计算并显示当前每周的重复时间比例。系统会统计每周重复安排课程的时间段，并计算重复时间所占比例。



因为一开始也只是在第一周添加了一个周期为18的课程，所以十八周的重叠时间比例都是一样的。

1. 查看本学期内任意一天的课表结果 用户可以输入日期，系统会显示该日期的课程安排情况。系统会列出当天每个时间段内安排的课程，包括课程名称、教师姓名和地点等信息。



## 3.7 基于语法的数据读入

private static final Pattern EMPLOYEE\_PATTERN = Pattern.compile("([A-Za-z]+)\\{(\\w+(?:\\s+\\w+)\*),([0-9]{3}-[0-9]{4}-[0-9]{4})}");

private static final Pattern PERIOD\_PATTERN = Pattern.compile("Period\\{([0-9]{4}-[0-9]{2}-[0-9]{2}),(\\d{4}-\\d{2}-\\d{2})}");

private static final Pattern ROSTER\_ENTRY\_PATTERN = Pattern.compile("(\\w+)\\{([0-9]{4}-[0-9]{2}-[0-9]{2}),(\\d{4}-\\d{2}-\\d{2})}");

利用上述三个正则表达式分别对应读取文件中的多个员工信息、排班的开始日期和结束日期以及排班计划。

# 4.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 |
| 2024/4/30 | 18：30-22；00 | 完成3.3部分ADT的设计； | 基本完成 |
| 2024/5/3  2024/5/4 | 20：00-23：00  14：30-18：00 | 完成3.4部分的decorator 设计模式  和面向各应用的 ADT 子类型；  完成3.5部分的面向复用的设计；  完成3.6部分的APIs实现； | 基本完成 |
| 2024/5/5 | 一天中挑出的多段空闲时间 | 完成3.7.1部分的排班管理系统； | 基本完成 |
| 2024/5/12 | 16：00-23：00 | 完成了3.7.2部分的课表管理系统； | 基本完成 |

# 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 |
| 在针对MultiIntervalSet<L>  的设计时，MultiIntervalSet<L>类要求实现同一个标签对象 L可被绑定到多个时间段上；但实验要求必须使用 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。就针对这个简洁明了的问题，我该怎么把一个只允许同一个标签只能有一个时间段的对象应用到MultiIntervalSet<L>类中呢？ | 我想了多种解决方案，比如（1）在Interval<L>中添加基础属性Map<L, Interval>，这样的话在MultiIntervalSet<L>的实现中使用多重Map，这样也可以满足Interval<L>中单个标签只对应一个时间段的需求，但这样子总感觉有点不喜欢；（2）在Interval<L>中添加基础属性为Map<L,List<Interval>>，这样子如果同一个标签要对应多个时间段的话，只需该标签在map中对应的list里再添加一个时间段即可；但这样子会不会影响了原来的Interval<L>呢，而且MultiIntervalSet<L>里要怎么实现对Interval<L>类中的属性进行操作呢？再经过了一个时间的琢磨后，我还是选择了（2），为了解决那两个问题，我选择让Interval<L>类与MultiIntervalSet<L>实现双向绑定，也就是在Interval<L>类中也添加一个MultiIntervalSet<L>类的属性类，但MultiIntervalSet<L>不为空时，就意味着此时的Interval<L>类已经是为了服务MultiIntervalSet<L>类而生的了，此时它的insert方法会有所改变，允许同一个标签能够插入多个不重叠的时间段了，这样也不需要在MultiIntervalSet<L>类中操作Interval<L>类中的属性了！ |
| 在具体应用场景开发的CourseScheduleApp中的随时查看哪些课程还未安排满周学时，我发现我好像无法获取单个标签被安排的时长，这该怎么办呢？ | 既然子类无法得到更加具体的信息，我只好往父类去增加方法。首先因为我存储每个课程信息时我为其添加了一个额外的属性（该课程每个新的时间段开始的当天的日期）我得先在CourseScheduleSet<L>  类中增加计算给定课程在给定当天的学时数，再到父类PeriodicMultiIntervalSet<L>中添加public int labelIntervalHours(long start, long end, L label)  方法计算出结果即可。 |
| 在完成个性化类CourseScheduleSet<L>中计算每一周的空闲时间比例和计算每一周的重叠时间比例这两个方法时，我觉得需要调用之前写的API方法，来完成代码的复用，但我在课程管理这个时间段集合上该如何调用API方法才能计算每一周的比例呢？ | 经过琢磨后，我发现可以在计算时间比例的方法里动态的改变课程安排表时间段集合的起始时间和学期周数，即每次循环重新设定学期开始时间，每次只持续一周，这样就能做到每次计算一周的时间比例了。算完后再改回原来的即可。  但其实课表安排上每天的上课时间只有10个小时，所以也就是计算一周的时间比例时分母是70即可；但这个好解决，用数学的逻辑即可（k是API得到的结果）：  每周空闲时间比例：answer = 1 - (1-k)\*24/10;  每周重叠时间比例：answer = k\*24/10; |

# 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

答：在本次的实验经历中，在本次实验中，我使用子类型、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设计和API复用等软件构造技术，开发了一套可复用的ADT及其实现，并应用于值班表管理、操作系统进程调度管理和大学课表管理三个具体应用。通过这个过程，我收获了许多宝贵的经验和教训。

模块化思维：通过设计ADT，我学会了如何将复杂问题分解为更小、更可管理的模块。这种思维方式使代码更具结构性和可维护性。接口与实现分离：定义接口和抽象类，使得具体实现可以独立于接口演化，从而提高了代码的灵活性和复用性。泛型的使用：利用Java的泛型机制，编写了类型安全且通用的ADT，实现了不同应用间代码的复用。继承和多态：通过继承和多态，设计了灵活的类层次结构，允许不同的实现方式共存，并在运行时选择具体的实现。重写与重载：重写（override）和重载（overload）方法，使得同一个方法可以有不同的行为，增强了代码的灵活性。组合与代理：在设计复杂系统时，使用组合（composition）和代理（proxy）模式，有助于代码的解耦和职责的单一化。设计一致性：在API设计过程中，保持一致的命名规范和接口风格，提升了代码的可读性和易用性。文档与注释：撰写详尽的文档和注释，使API更易于理解和使用，并有助于维护和扩展。API复用：通过定义通用接口，实现了多个应用场景下的代码复用，降低了开发成本。语法分析：学习了如何使用语法驱动的编程技术，编写解析器来处理复杂的输入格式。正则表达式：通过正则表达式进行模式匹配和文本处理，解决了许多文本解析和验证的问题。

过度抽象：在设计过程中，容易陷入过度抽象的陷阱，导致设计过于复杂，难以实现和维护。应保持适度抽象，确保设计简洁且符合实际需求。类层次结构的深度：继承层次结构过深，会导致代码难以理解和维护。在设计时，应尽量避免过多的继承层次，选择合适的设计模式。泛型的性能影响：泛型虽然增加了代码的复用性，但在某些场景下可能带来性能开销。在性能敏感的部分，应进行性能分析和优化。边界条件测试：在测试过程中，未充分考虑所有可能的边界条件，导致一些隐藏的bug未能及时发现。应加强边界条件的测试，确保代码的健壮性。异常处理测试：异常处理路径未充分测试，导致在实际运行中出现未预见的错误。应设计全面的异常处理测试用例。

通过本次实验，我不仅掌握了多种软件构造技术，还在实际应用中积累了宝贵的经验和教训。这些收获将帮助我在未来的开发工作中设计出更高质量、更具复用性和可维护性的代码。

## 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在三个不同的应用场景下使用，你是否体会到复用的好处？

答：针对与Lab2中的问题，我现在的体会就是当你在一次只面对一项任务（也就是只针对一个应用场景时），直接面向应用场景编程确实会更快，直觉上给你一种没有面向ADT编程的必要。但实际上，在本次实验中，老师一次性给出了三个应用场景，那显而易见如果依然选择直接面向应用场景编程的话，那会把工作线拉的很长，事实上还会出现大量重复代码的书写，此时面向ADT编程的优势就体现出来了；我们首先审视这三个应用场景，找出其中具有的共同特点，设计出一套能够复用的共同的ADT，此后再针对每个特殊的应用场景具体实现特殊的功能即可。完成了本次实验后，我更加感受到了复用的优势，不仅能够为项目的开发提高效率，还能使整个项目的框架看起来更加的有逻辑、有条理。

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？

答：对ADT进行复杂的规范化工作，包括specification、invariants、RI（Representation Invariant）、AF（Abstraction Function），以及避免rep exposure的设计，其意义在于确保ADT的正确性、稳定性和可维护性。通过明确定义ADT的行为、状态和约束，可以帮助我们更好地理解和使用ADT，减少出错的可能性，提高代码质量。在以后的编程中坚持这么做是非常有益的，因为良好的规范化工作可以提高代码的可读性、可维护性和可扩展性，有助于团队协作和代码重用。我会尽量在以后的编程中坚持这样做。

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？

答：通过开发给别人使用的API，可以更深刻地体会到API设计的难处和乐趣。在设计API时，需要考虑到用户的需求和使用习惯，保证API的易用性和灵活性，同时要注意接口的清晰和稳定，避免对用户造成困扰。在开发过程中，不仅要考虑到功能的完整性，还要考虑到接口的设计是否合理、易于理解和使用。在开发给别人使用的API时，需要更加关注用户的体验，这对于提高自己的编程水平和设计能力都是很有帮助的。

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？

答：对于语法驱动编程的感受，我认为它能够提高代码的可读性和可维护性。通过使用语法和正则表达式去解析输入文件并据此构造对象，可以使代码更加清晰和易于理解，而不是直接处理底层的字符串操作。语法驱动编程可以将复杂的输入转化为简洁的数据结构，提高了代码的抽象层次，使得代码更具表达力和可扩展性。

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？

答：对需求的理解和分析：需要深入理解问题领域和用户需求，抽象出合适的数据结构和操作，这需要对问题有足够的认识和分析能力。设计规范化：需要定义清晰的specification、invariants、RI、AF等，确保ADT的正确性和稳定性。处理复杂性：有些问题可能涉及到复杂的逻辑和算法，需要设计合适的数据结构和算法来解决，这需要良好的抽象能力和设计能力。兼顾性能和灵活性：需要在性能和灵活性之间进行权衡，设计出既能满足需求又具有良好性能的ADT。我是通过深入理解问题领域、充分分析需求、参考经典的数据结构和算法设计来克服这些困难的，同时不断进行代码重构和优化，保证ADT的质量和性能。

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？

答：合理利用接口和抽象类：接口和抽象类是代码抽象的重要工具，可以定义一组相关的方法和属性，并提供标准的实现。在设计ADT时，可以通过接口和抽象类来定义ADT的行为和约束，然后通过具体类来实现具体功能。设计模式的应用：设计模式是解决特定问题的通用解决方案，可以提高代码的灵活性和可扩展性。在设计ADT时，可以考虑使用设计模式来解决一些常见的问题，如工厂模式、策略模式、观察者模式等，从而提高代码的复用性和可维护性。保持代码清晰和简洁：在设计ADT时，要尽量保持代码的清晰和简洁，避免过度设计和冗余代码，保持代码的可读性和可维护性。要注重接口的设计和命名，使得接口和类的功能和用途清晰明了。

1. 关于本实验的工作量、难度、deadline。

答：这次实验的工作量明显提高，难度也有明显的上升，主要是需要考虑的细节太多了，而且代码量也很大，特别是当在后面具体开发应用时，一旦遇到了一个逻辑上的错误，就需要追溯到时间段集合的好几代祖先那，debug的过程及其痛苦。

1. 到目前为止你对《软件构造》课程的评价。

《软件构造》课程为我带来了深刻的启发和丰富的知识体验。在这门课程中，我不仅学习了软件构造所涉及的广大层面的基本理论和方法，还深入了解了软件开发过程中的各种技术和工具（如git的使用等）。课程内容全面而深入，通过课程的学习，我系统地掌握了软件构造的基本概念、原则和方法，包括需求分析、设计模式、代码重构、测试技术等。课程内容涵盖了软件构造的方方面面，让我对软件开发过程有了更加清晰和全面的认识。当然记忆最深刻的就是测试优先的编程了，这是我第一次见识到这样的编程方式，属性让我倍感震撼。

课程还注重理论与实践相结合。在课堂教学中，老师不仅讲解了理论知识，还结合部分实际的代码示例以及习题课的及时介入来进行了详细的讲解和演示。同时，课程还针对学习的知识布置了三次实验，让我们有机会将理论知识应用到实际项目中，加深对知识的理解和掌握。

当然最主要的还是课程教师，教学资源丰富，讲解的难点非常形象透彻（尤其是每每举出鱼香肉丝的例子），能够将复杂的理论知识讲解得通俗易懂，深入浅出。同时，课程还提供了丰富的教学资源和学习资料，包括课件、教学视频、实验指导等，方便学生自主学习和查阅。

综上所述，我认为《软件构造》课程是一门内容丰富、实践性强、教师非常优秀的课程。通过学习这门课程，我不仅提高了软件构造的理论水平和实践能力，还培养了解决问题的能力。这门课程为我未来的学习和职业发展奠定了坚实的基础，我会继续努力学习，不断提升自己的技能和能力。