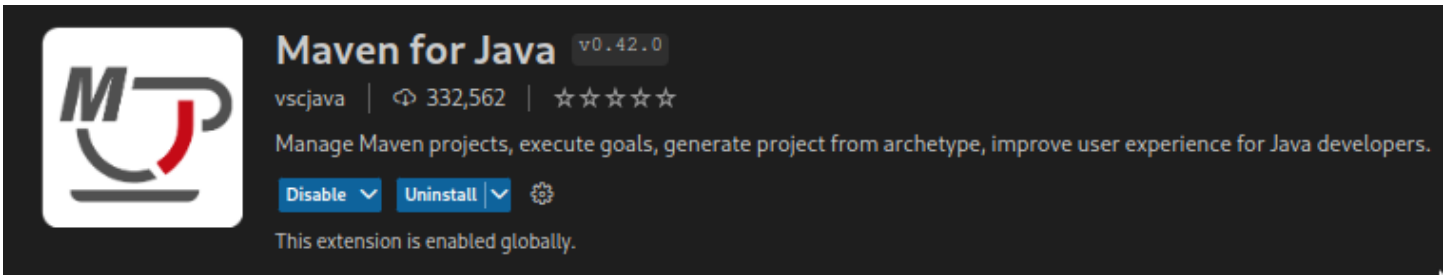
Technical Details of the Scheduling Program

By:Ahmad Fahes, Sajed Hamdan, Fadi Atwi

Technical details

Requirements:

The program is written entirely in java.

- **Java Development Kit (JDK)**
- **IDE**: with a capability to build and run a java maven project
  E.G.: VScode with "maven for java" extension (+ extensions to run java)
  

- **Apache POI library**: present in the pom.xml dependencies section; Maven should import it automatically when building.

The program has been compiled into a JAR file located in the target/ directory. However no main manifest has yet been specified. to run the program you should run the FileSelectorWindow class from the JAR file. for example using this command:

```
java -cp target/scheduler-4.7.2.jar scheduler.FileSelectorWindow
```

Alternatively, you can run the FileSelectorWindow.java file.

Docs:

```
This program started simple, but over time, it grew in complexity as new requirements
were added. The initial straightforward structure had to be adapted and extended to
accommodate these changes. As a result, you will encounter code that is patched
together or "unconventional".
```

The courses are scheduled over a 2D array of 5 columns (5 days a week) and 6 rows (each day split into 6 timeslots)

The 6 TimeSlots each start at a specific hour

- 08:00
- 09:30
- 11:00
  (break from 12:15 to 13:00)

- 13:00
- 14:30
- 16:00
  by default each slot is 1 hour 15 minutes in length, with a 15 minute break between each slot. However, courses can cross multiple TimeSlots if their duration is long enough.

Objects:

course:

Stores all attributes from excel file as String or int values.
In addition to:

- IsScheduled 2D array **(bool)**:
  represents the weekly array. true for TimeSlots this specific course is scheduled on.
- sessionsScheduled **(int)**:
  used to store the number of sessions scheduled during scheduling; in order to determine when course is fully scheduled by comparing
  sessionsScheduled == numberOfSessions(to be scheduled)
- durationMinutes **(int)**:
  total duration of each single session in minutes
- isPerfectlyScheduled **(bool)**:
  to determine if the course has been scheduled perfectly
  Perfect schedule means that a course was scheduled on the same TimeSlots on each day. this is expanded on further in CourseScheduling
- OtherInfo **(String)**:
  Stores extra info on the course such as error messages.

instructor:

- Hours as LocalTime
  Map<String, LocalTime> availabilityStart;
  Map<String, LocalTime> availabilityEnd;
  maps each day to its start time and end time.
- Hours as Integer List
  Map<String, List< Integer >> availability;
  Transforms the LocalTime hours into a list of integers representing the available hours.
  E.G.:
  start time: 08:00
  End time: 10:50
  resulting Integer list would be [0,1]
  representing the first 2 slots

```
1. 08:00 to 09:15
2. 09:30 to 10:45
```

When creating the course objects during file Reading,
the list of integers changes depending on the **Total Minutes** / **session** for each course.

each course creates its own copy of an instructor object to store this differing information.
E.G.:

start time: 08:00
end time: 08:50

- Case 1:
  course is of length 1 hour 15 mins
  Available Slots integer list would be empty []
- Case 2:
  course is if length 50 minutes
  Available Slots integer list would include slot 0 [0] (the index of the first slot)e

Course Scheduling:

The course objects are stored in a queue <|UUID|> and a hashmap of types <UUID, course>.

UUID from java.util library, is a unique 128 bit ID assigned to each course, this is passed into the methods and stored in the 2D Array instead of the entire course object.

Running the ScheduleCourses(boolean isSummer) method runs through the courses in the courseQueue one by one checking a few things:

- if isSummer == true:
  double the number of sessions of the course that were read from the excel file.
  E.G.: if input excel file states the course has 4 sessions/week
  double it to 8 sessions/week
- if Instructor object of course is null:
  if true -> add the course to the **errorCourses** Set and continue to the next course.
- if sessions/week of course == 0
  if true -> add the course to the **ZeroSessionCourses** set and continue to the next course.

if course passes all of these conditions it passes to the next stage where the program Handles different sections

- if course has only 1 Section -> move to scheduling the course
- if course has Sections > 1 ->
  create new course objects; one for each section, incrementing the lectureNb attribute for each section

E.G.:
MATH211 with 2 sections would become:

- MATH211-1
- MATH211-2
  each of these courses is added separately.

Adding courses:

courses are added into the schedule in specific patterns.
these patterns are (in order of priority):

1. **Day pair scheduling**:
   Attempt to schedule the course sessions equally on 2 days a week.
   These day pairs are:
   - M/W
   - T/TH
   - W/Fr
     With in day pair scheduling:
     first attempt:
   1. Schedule sessions on common slots on both days
      if unsuccessful unschedule course from all days.
      if successful -> set course isPerfectlyScheduled = true
   2. Attempt schedule on both days disregarding common slots entirely
      if unsuccessful unschedule course from all days.
      if successful -> return and continue to next course.

NOTE: Successful scheduling is defined by

```
sessionsScheduled == numberOfSessions(to be scheduled)
```

2. **Equal spread scheduling**:
   Attempt to schedule the course sessions equally on all available days.
   Similar to **Day pair scheduling**
   first attempt:
   1. Schedule sessions on common slots on all days
      if successful -> set course isPerfectlyScheduled = true
   2. Attempt schedule on both days disregarding common slots entirely
3. **Any spread scheduling**:
   Attempt to schedule the course on any available TimeSlot disregarding any pattern.

- A TimeSlot must satisfy 2 conditions in order to be available:
  1. is present in the course's instructors TimeSlots list.

2. No conflicts present:
   conflicts mean:
   - a scheduled course is present in the **conflictingCourses** list of the course to be scheduled.
   - course to be scheduled is present in the **conflictingCourses** of any of the scheduled courses.
   - any of the present courses share the same instructor as the course to be scheduled.

If none of these scheduled attempts are successful. the course is Added to **UnscheduledCourseHeap** set.

Rescheduling unscheduled courses:

The program attempts to reschedule all the courses that were not successfully scheduled during the first initial run of scheduling courses.

NOTE:
this part of the code does not do its job properly; as in testing with large numbers of courses there always remains some unscheduled courses.
a better solution might be to deal with unscheduled courses during the first scheduling run itself not after.

The program goes through each course in the **UnscheduledCourseHeap**
for every **conflicting course** for the **unscheduled course**:

1. Unscheduled the **conflicting course**
2. attempt to schedule the original **unscheduled course**
3. attempt to schedule the **conflicting course**
   if:

   - conflicting course was not successful added
     OR
   - conflicting course was **perfectly scheduled** and now isn't

   revert all changes and move on to the next conflicting course

if all courses are scheduled and rescheduled successfully move onto the next **unscheduled course**.