Mark Joseph C. Nagrampa

BS IT - 2

**OOP**

**Project Documentation**

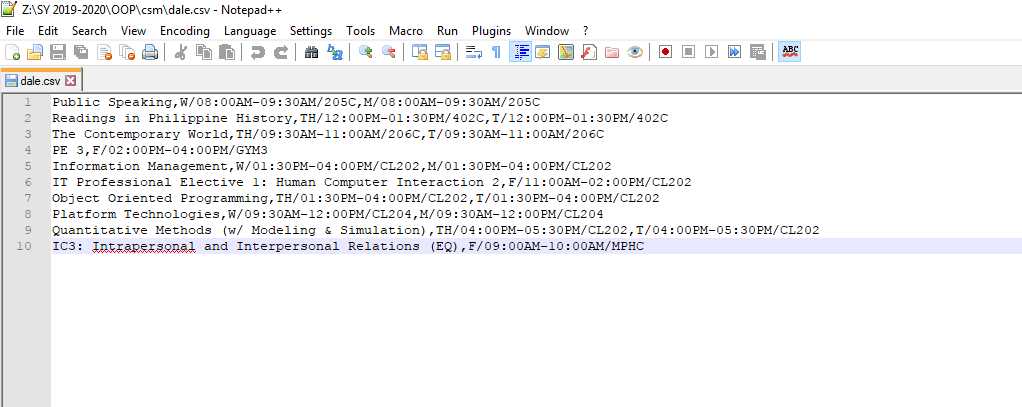
1. **Abstract**

This project is a class schedule maker. Given a CSV file listing a student's subject list and schedule, this program will make an Excel file which shows the student's schedule in an organized table. See the programmer’s printed Class Schedule for reference.

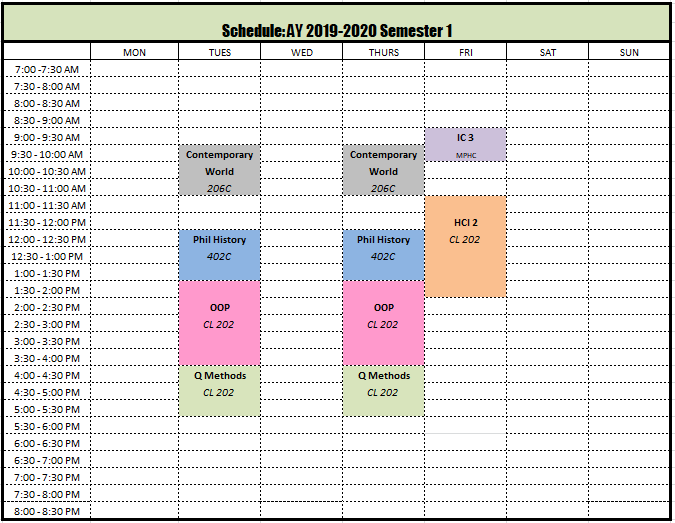
The rationale for choosing this sample input is that these inputs are auto-generated when a Southville student receives his registration form. The only thing that should be added is the comma delimiter.

The goal of this program is to integrate the process of generating a class schedule within the enrollment process; enhancing the experience therewith.

Sample Input:



Sample Output:



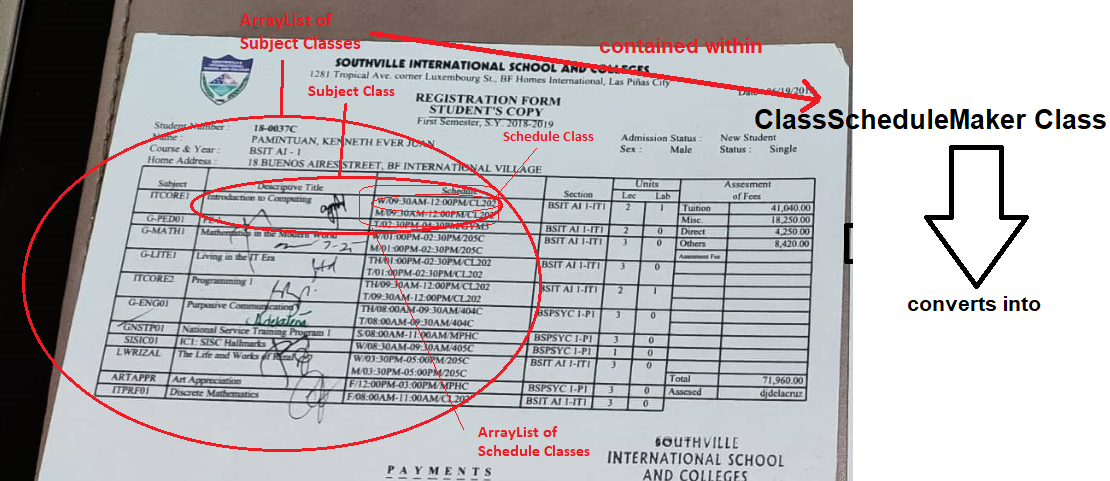
Changes from the Proposal:

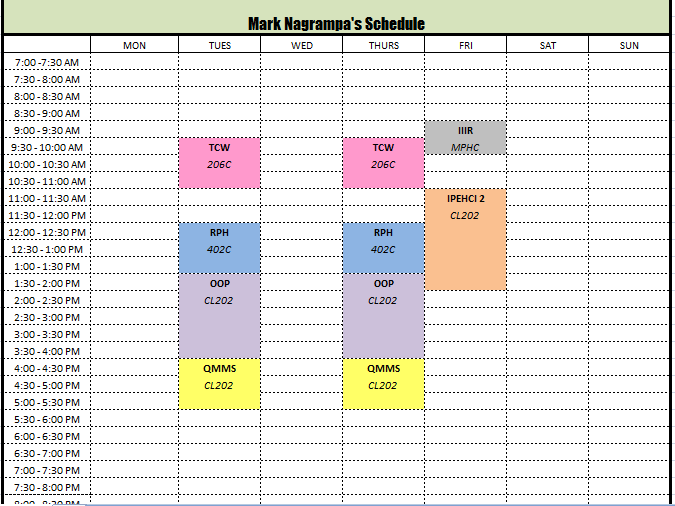
The following changes have been introduced:

1. The *getSubjectsFromConsole()* method has been removed from the *ClassSchedukeMaker* class. The programmer has evaluated that this function is not necessary because it doesn’t streamline the process of obtaining the subjects list and then creating the Schedule file. Asking the user to input all subject information from the console is tedious. And the function will most likely be unutilized.
2. The *abbreviate()* method has been renamed to the *getFormattedName()* method. The subject name is the only field which exceeds the 12 character cell width limit and requires special formatting asides from the abbreviation.
3. The *getDimensions()* method has been split into three separate methods - the *getStartY()*, *getEndY()*, and *getX()* methods. Each of these methods is too specialized to be included into a single method and it is more convenient to call each of these methods separately.
4. The *Subject* class was split into two classes: the *Subject* class and the *Schedule* class. This is because a single subject can take multiple schedules. These inputs should be stored in a separate object.
5. The maximum number of subjects has been increased to 14 to accommodate students with slightly overloaded schedules.
6. **Design**
7. **csm Package**

This is a standalone package which holds all the necessary codes for the project. The package contains 3 classes: *ClassScheduleMaker*, *Subject*, and *Schedule*.

Abstraction of Class Relationship:





This project uses the Apache POI library (a set of dependencies which specialize in Microsoft Office Object manipulation) – a key library for editing Excel files. Because of this, the project is required to be setup as a Java Maven Project (and not a vanilla Java Project).

Additional setup configurations are listed after this package description.

* 1. **ClassScheduleMaker Class**

This is the driver class which processes the subjects and their corresponding schedules into a formatted excel file.

Field list:

*ArrayList<Subject> subjects* – an array holding the user’s subject list

* + 1. **main**

the main method is responsible for obtaining the user inputs for the following information:

* The file name of the CSV file containing the student’s subject schedule
* Color Scheme of choice (“1” for black and white, any string for colored)
* Student’s name
* Excel Output File Name

It simply obtains these inputs and afterwards runs the *getSubjectsFromFile()* method and *createExcelFile()* method

* + 1. **void getSubjectsFromFile(String filename)**

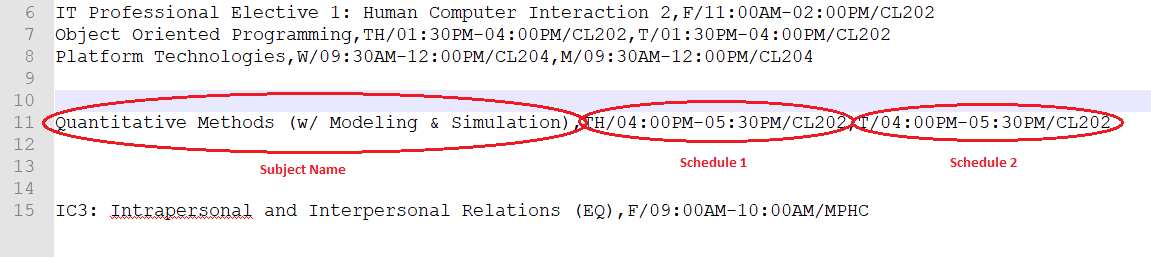
Parameter:

*String filename* – the subject list’s filename

This method gets this schedule's subject list from an existing file

Although not explicitly stated, this program takes a file which follows the csv format – that is rows which follow the structure of:

*subject name, schedule 1, schedule 2, …*



Methods iii.to vi. are all run within the *getSubjectsFromFile()* method. If any of them are true, this means that the file has an invalid schedule, thus the program must be terminated (We cannot make a formatted Excel schedule file for an erroneous input).

* + 1. **boolean hasConflictingSubjects()**

This method checks if any of the *Subjects* in the ArrayList have *Schedule* class pairs which conflict such as two subjects sharing the same start time, or two subjects wherein the start/end time of one is between the duration of the other.

It returns true for the following conditions:

* the start time of subject A = the start time of subject B
* end time of A = end time of B
* start time of B <= start time of A < end time of B
* start time of B < end time of A <= end time of B

Return value:

True – the read file has conflicting subjects and thus the program must be terminated

False – the read file has no conflicting subjects, we can continue creating the Excel file

* + 1. **boolean hasExceedingSubjectCount()**

This method checks if the *subjects ArrayList* has more than 14 subjects.

The color palette for this project only contains 14 colors, and thus the restriction for 14 subjects was made so that there can only be 14 subjects at a time with a distinct color.

Return value:

True – the read file has more than 14 subjects and thus the program must be terminated (the no. of subjects exceeds our color palette)

False – the read file has less than 14 subjects, we can continue creating the Excel file

* + 1. **boolean hasExceedingScheduleTime()**

This method checks if any of the *Subjects* in the ArrayList have *Schedule* classes with time exceeding 7:00 AM - 8:30 PM.

This method checks for the following cases:

* An hour less than 7 or greater than 20 (8pm)
* A starting time of 8:30pm
* An ending time of 7:00 am

If any of these cases are true, the method returns true (otherwise false).

This method is used in the class’ *getSubjectsFromFile()* method as it will check whether the selected file has invalid inputs.

Return value:

True – the read file has subjects which exceed the file’s encompassed 7 am – 8:30 pm schedule and thus the program must be terminated

False – the read file has subject times within the schedule. We can continue creating the Excel file

* + 1. **boolean hasInvalidScheduleDuration()**

This method checks if any of the *Subjects* in the ArrayList have *Schedule* class durations with end times less than or equal to their corresponding start times.

It returns true for the following cases:

* end[0] (ending hour) - start[0] (starting hour) < 0
* end[0] (ending hour) - start[0] (starting hour) == 0 and end[1] (ending minute) – start[1] (starting minute) <= 0

Return value:

True – the read file has subjects with end less than or equal to their corresponding start times and thus the program must be terminated

False – the read file has valid schedule durations, we can continue creating the Excel file

* + 1. **void setColorScheme(String choice)**

Parameter:

String choice – (“1” – black and white, otherwise - colored)

Description:

This method sets the color scheme of the schedule file. It takes a single string parameter choice which corresponds to the user’s selected color scheme. “1” Indicates a black and white theme, and any other string indicates a colored theme.

In the case of a black and white theme, the program iterates through each Subject class and sets its Color class to a new color object with RGB values of (255,255,255) (white).

In the case of a colored theme, the program iterates through each Subject class and sets its Color class to an indexed Color from a pre-defined palette (a static Color[] object with preset RGB values based on the programmer’s liking)

* + 1. **void createExcelFile(String headerName, String outputFileName, String colorScheme)**

Parameter List:

String headerName = The student’s name

String outputFileName = the output file name (excluding extension)

String colorScheme = (“1” – black and white, otherwise - colored)

Description:

This is the method which converts the pre-defined objects into the formatted excel file. String headerName corresponds to the student’s name which will be placed on the table’s top merged cell region. String outputFileName is exactly what is stated. Note that this argument doesn’t include the extension as the program is preset in making a specific file format – nothing else (.xlsx)

The entire project is hinged on this method. In order to create the Excel file and have control of its formatting, this project used the Java POI package (a set of libraries which specialize in manipulating Microsoft Office Objects). The dependencies for this package require the project to be set up as a Java Maven Project (and not the vanilla Java Project).

The logic of this method is outlined by the following steps:

1. Create the .xlsx file and create a single sheet

2. Create 30 rows (2 header rows, 1 day row, and 27 thirty minute block rows)

2a. Create each cell and apply default styling to each:

- Calibri 8 Font

- 11.86 points column width

- Dotted Border

2b. Set row height to 12.75 points



3. Create The Schedule Header

- Merge the entire top 2 rows of the excel file

- Set the title according to the user's input

- Apply Impact font

- Apply the selected color scheme

4. Create Day Headers

- MON TUES WED THURS FRI SAT SUN



5. Create Time Headers

- Hour count starts at 7

- Every even (0-based) index has a :00 - :30 minute schedule

- Every odd (0-based) index has a :30 - :00 minute schedule

- The first 9 blocks are AM schedules, the rest are PM

- Reset Hour Count to 1 after reaching the 12th hour

6. Fill Out Subject Blocks

- Apply custom fill color to each affected subject cell

- If the subject is only 30 minutes:

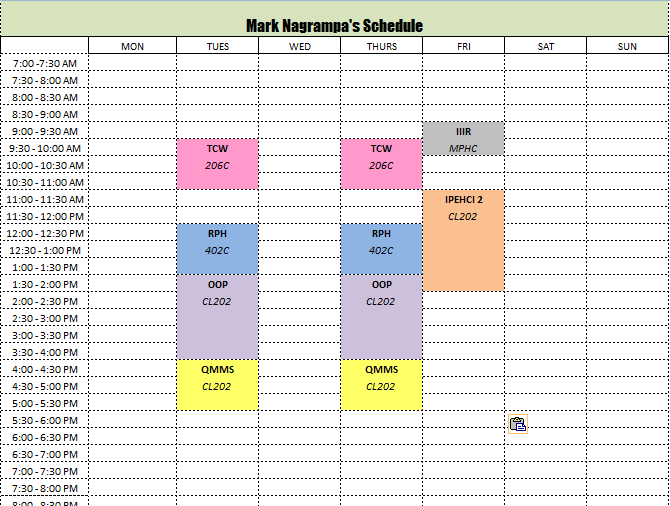
- Place the subject name on the designated cell

- Else:

- Place the subject name on the top cell

- Place the room name on the cell immediately below

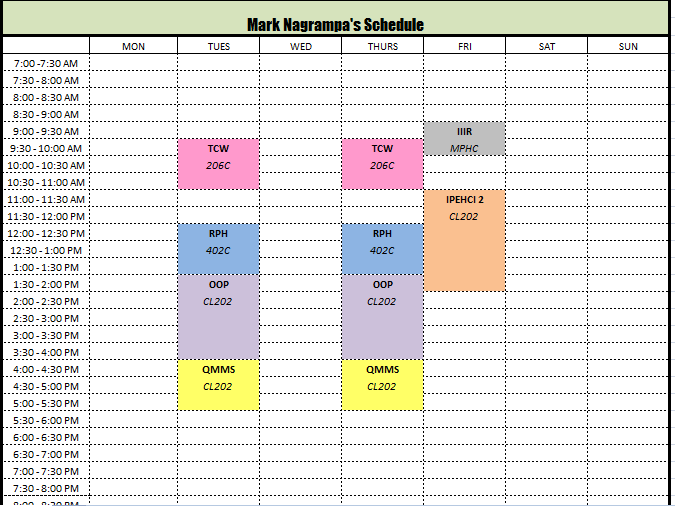
- Remove excess dotted borders for cells in between the subject's schedule range



7. Finalize Border Styles

- Apply a thin border around each Day Header

- Apply a thick black border around the table's edges

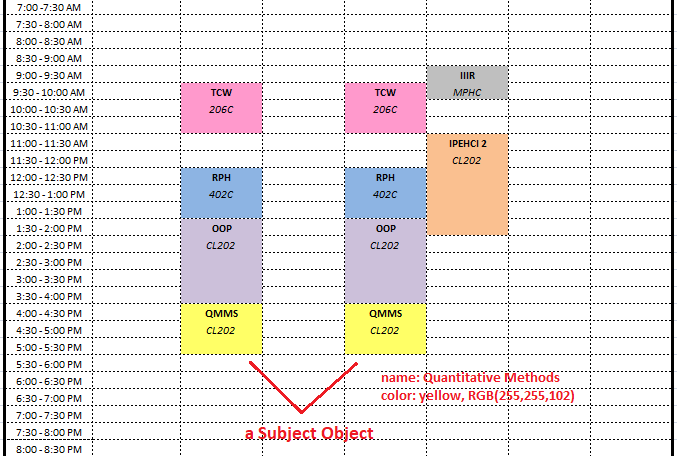


8. Write the WorkBook Object into the File

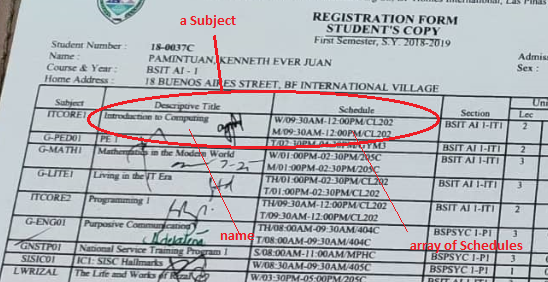
* 1. **Subject Class**

This class contains the subject’s name, assigned color code, and an array of Schedule classes for each meeting of the said Subject within the week.

Abstraction:



Actual:



Field list:

*String name* – the subject name

*Color color* – the subject’s foreground color

*ArrayList<Schedule> schedules* – an array holding each of this subject’s schedule info

*Static Color[] palette* – a fixed array holding all the possible colors which can be used in this schedule’s formatted output

* + 1. **String getFormattedName()**

This method returns this subject's formatted name (abbreviated if it exceeds the cell width).

The formatting works as follows:

If the subject’s title doesn’t exceed 12 characters, it simply returns the subject name as is.

If it exceeds the column width limit, the method splits the title by spaces and includes every first word uppercase character and stand-alone numbers in the title.

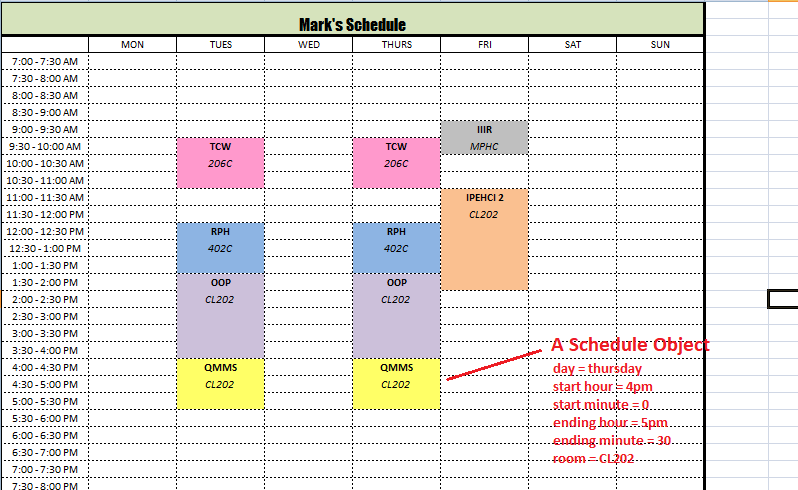
Return Value:

String – the properly abbreviated subject name

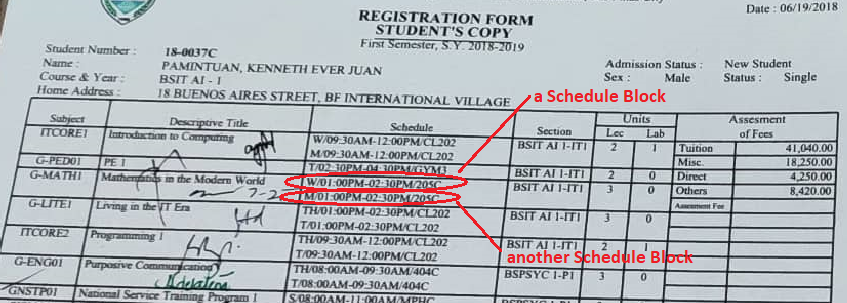
* 1. **Schedule Class**

This class contains information regarding a subject’s day, room, start hour, start minute, end hour, and end minute.

Abstraction:



Actual:



Field list:

*String room* – the room name

*Int day* – the day of the week

0 – Monday

1 – Tuesday

2 – Wednesday

3 – Thursday

4 – Friday

5 – Saturday

6 - Sunday

*Int[] start* – the start time

* index 0 = hour, index 1 = minute
* hours follow a 24 hour format, so 1pm is represented as 13

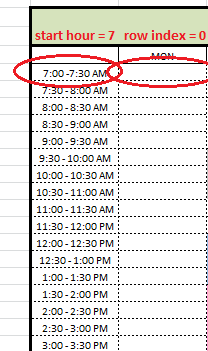
*Int[] end* – the end time

* index 0 = hour, index 1 = minute
* hours follow a 24 hour format, so 1pm is represented as 13
  + 1. **int getStartY()**

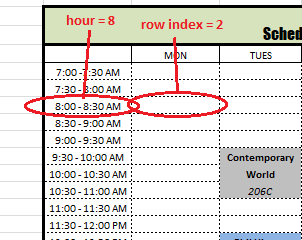
This method returns this schedule's first row number in the formatted Excel schedule file.

The return formula is broken down as follows:

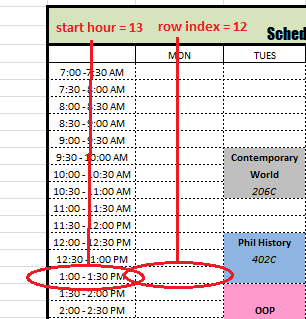
If the hour is 7am, its relative row index is 0 (and thus – 7 of the hour field).



If the hour is 8 am, its relative row index is 2 (-6 of the hour field).

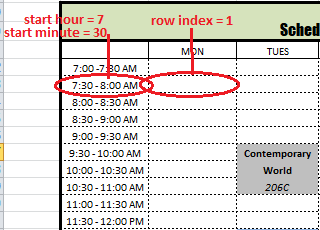


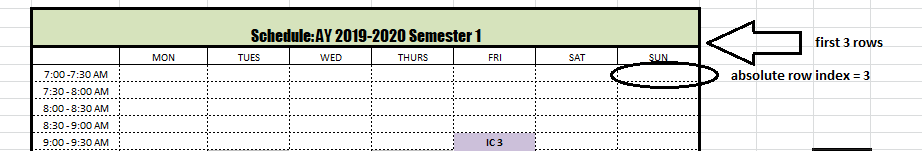
Continuing in this pattern shows that for any start hour (x), its relative starting row index is (2x - 14) [hour - 7 plus the hour’s distance from the table’s initial hour 7].



*(ie: start hour = 13, row index => 13(2) – 14 = 12)*

If it starts with a minute count of 30, the start row is 1 index lower than its usual (and thus +1).



The + 3 is included because the first, second, and third rows are for the table’s headers.

Return Value:

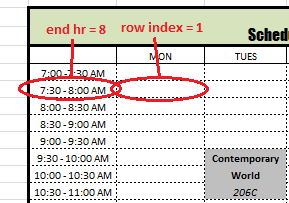
Int – the first row number of this Schedule within the Excel file

* + 1. **int getEndY()**

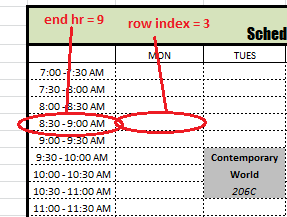
This method returns this schedule's last row number in the formatted Excel schedule file.

The return formula is broken down as follows:

If the ending time is 8am, its relative row index is 1 (and thus – 7 of the hour field).



If the ending time is 9am, its relative row index is 3 (and thus -6 of the hour field).



Continuing in this pattern concludes that the end row is basically 2 \* hr – 15.

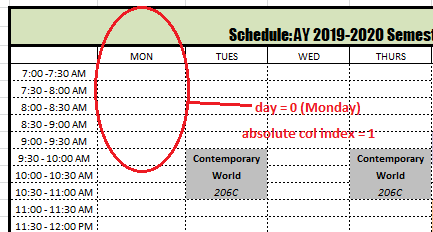
If it ends with a minute count of 30, the end row is 1 index below the usual (and thus +1).The + 3 is included because the first, second, and third rows are for the table’s headers.

Return Value:

Int – the last row number of this Schedule within the Excel file

* + 1. **int getX()**

This method returns this schedule's column number in the formatted Excel schedule file. Notice that Monday is represented as 0 and Sunday is 6. Because of this, the relative column index is basically the day. The +1 is included because the first column is reserved for the time display.



Return Value:

Int – the column number of this Schedule within the Excel file

* + 1. **boolean isSingleBlock()**

This method checks if this subject will only take a single cell in the formatted Excel schedule file (a 30 minute class). It was necessary to implement this code because a single block cannot display both the subject’s name and room, so the scheduler must settle with displaying only the subject’s name.

It returns true for the following case:

* start[0] (start hour) = end[0] (end hour) AND end[1] (end minute) – start[1] = 30
* end[0] (end hour) - start[0] (start hour) = 1 AND start[1] (start minute) – end[1] (end minute) = 30

Return Value:

True – this subject is a single block (30 mins only), only the subject name should be displayed

False – this subject is a multiple block subject (> 30 mins), display both the subject name and the room string

* + 1. **boolean isConflicting(Schedule other)**

Parameter:

Schedule Other – the Schedule object to be compared with this one

Description:

This method checks if this schedule is conflicting with the schedule of the other subject.

It checks for the following test-cases:

Two subjects sharing the same start/end time

Two subjects wherein the start/end time of one is between the duration of the other

It returns true for these cases and false otherwise.

Specifically, it returns true for the following conditions:

* the start time of subject A = the start time of subject B
* end time of A = end time of B
* start time of B <= start time of A < end time of B
* start time of B < end time of A <= end time of B

Return Value:

True – this Schedule conflicts with the schedule of another

False – this Schedule and the other do not conflict

1. **POM.xml**

This .xml file specifies the project to link the Apache POI dependency to its import list.

Please include this code in the pom.xml’s dependency list:

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi-ooxml</artifactId>

<version>3.15</version>

</dependency>