Yuan Jiayi - Project Portfolio for MisterMusik

1. About the project

My team of 5 computer engineering students were tasked with enhancing a program Duke for our CS2113T project. We chose to design a scheduler application for serious music students pursuing a professional music career as a western classical music performer. The program is designed to automate and streamline most of the process in scheduling and organisation of materials, allowing the students to focus more on the important aspects of their education.

This is what our project looks like:

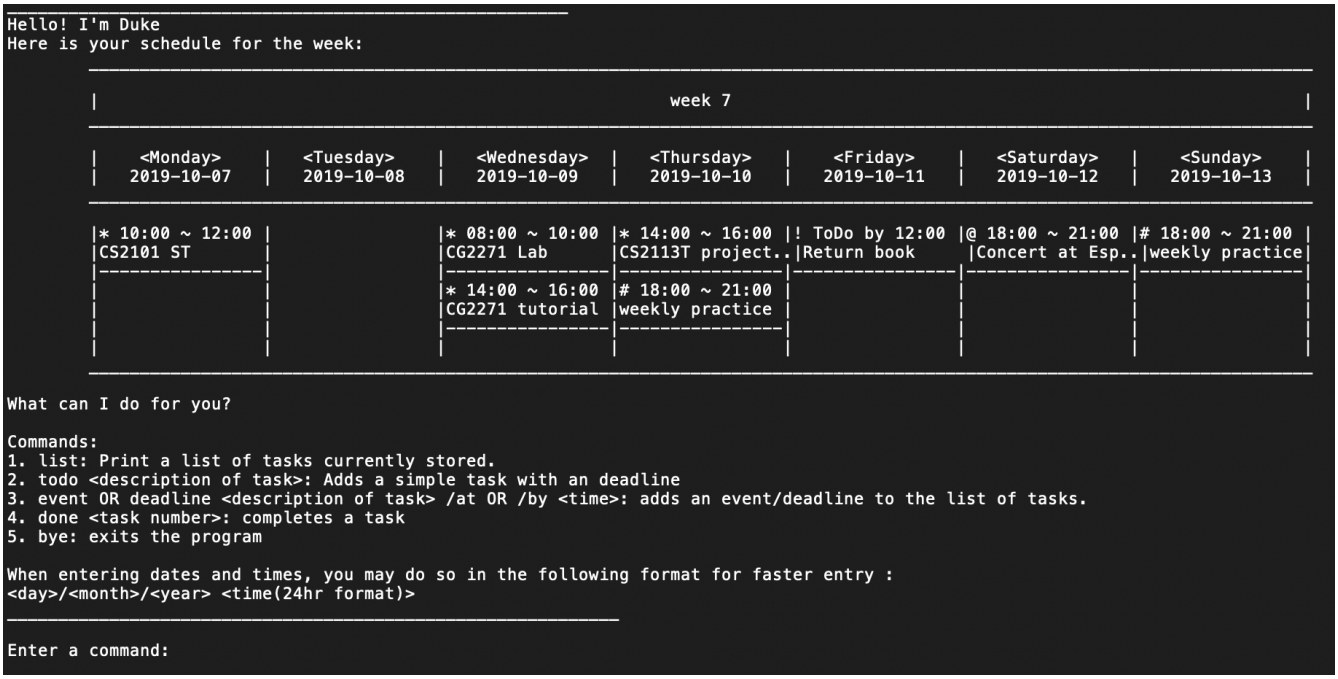

*Figure 1. The graphical user interface for **MisterMusik**.*

My role is to design and write the codes for the `recurring event` and `reschedule` features. The following sections illustrate these enhancements in more detail, as well as the relevant

documentation I have added to the user and developer guides in relation to these enhancements.

Note the following symbols and formatting used in this document:

This symbol indicates important information.

`reschedule` : A grey highlight (called a mark-up) indicates that this is a command that can be inputted into the command line and executed by the application.

addRecurringEvent : Italic text indicates a component, object, class, or method.

2. Summary of contributions

This section shows a summary of my coding, documentation, and other helpful contributions to the team project.

2.1. Enhancement added:

I added the ability to add recurring events, reschedule the events in the list, and add contacts.

2.1.1. Add recurring events

1. What it does: The `lesson` and `practice` commands allow the user to add recurring events happen in 16 weeks, using input format `lesson/practice <description> /dd-MM-yyyy HHmm HHmm /period(in days)`.
2. Justification: Some events are recurring, like some lessons happen once a week, or practices happen every two days. The command with period enables users to add recurring events easily, instead of adding them one by one.
3. Highlight: This implementation was challenge because the date in String format is needed when adding the new event to the list. However, the way to get the new date by calendar calculating was in Java Date type. Hence, a way to transfer Java Date type date to String type date was added to *EventDate* class.

2.1.2. Reschedule

1. What it does: The `reschedule` command allows the user to change the date and time of an event in the list.
2. Justification: In the event that users have made a mistake or changed their minds about the date and time, the `reschedule` command enables them to change the start and end date and time of the event in the list.

2.2. Code contributed:

Please click these links to see a sample of my code: #10, #15, #42, #81, #95, #96

2.3. Other contributions:

1. Project management:
    a. Create and Manage issues on GitHub.
    b. Create and Manage milestones (1.1 to 1.4) on GitHub.
2. Enhancements to existing features:
3. Documentation:
    a. Add the basic information to the User Guide: #66
    b. Restructure README file to make it fits the project style: #37, #38
    c. Add AboutUs page to introduce the team: #37
    d. Add information about recurring events and reschedule implementations to the Developer Guide: #91, #105
4. Community:
    a. Reviewed Pull Requests:

3. Contributions to the User Guide

We had to update the original User Guide with instructions for the enhancements that we had added. The following is an excerpt from our MisterMusik User Guide, showing additions that I have made for the `reschedule` and recurring events features.

This section also contains an excerpt for the feature that I have planned for the next version (v2.0) of MisterMusik.

3.1. Recurring events :

MisterMusik allows the user to add recurring events (e.g. weekly lessons). The input format is as per normal with an extra recurring period input. This only works with lesson and practice type events. Concerts, exams and recitals cannot be entered as recurring events.

Format: `<type of event> <description> /dd-MM-yyyy HHmm HHmm /period(days)`

Example:
Let's say that you have a CG lesson which is on every Monday morning in 19/20 semester 1.
Instead of typing the command to add these lessons one by one, you can easily add `/7` after `lesson CG /13-08-2019 1000 1200` which is the command of adding the first lesson to add these recurring events in one semester in one command.

To add recurring events:

1. Type `lesson CG /13-08-2019 1000 1200 /7` into the command line, and press `Enter` to execute it.

    image
2. The message "" will be displayed.

image

3. And you can type `list` to check whether the events have been added.

image

The recurring events feature applies to lesson and practice types of events only.
Format: `practice <description> /dd-MM-yyyy HHmm HHmm /period(in days)`, `lesson <description> /dd-MM-yyyy HHmm HHmm /period(in days)`

The maximum days between the first recurring event happens and the last one added to the list are fixed to 112 days(16 weeks).
It will waste a lot of memory if adding too much events which happen far from now. 112 days are quite similar to the length of one semester, so that you can manage recurring events easier.

3.2. Rescheduling events : `reschedule`

The user will be able to reschedule an existing event.

Format: `reschedule <task index> dd-MM-yyyy HHmm HHmm`

4. Contributions to the Developer Guide