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:

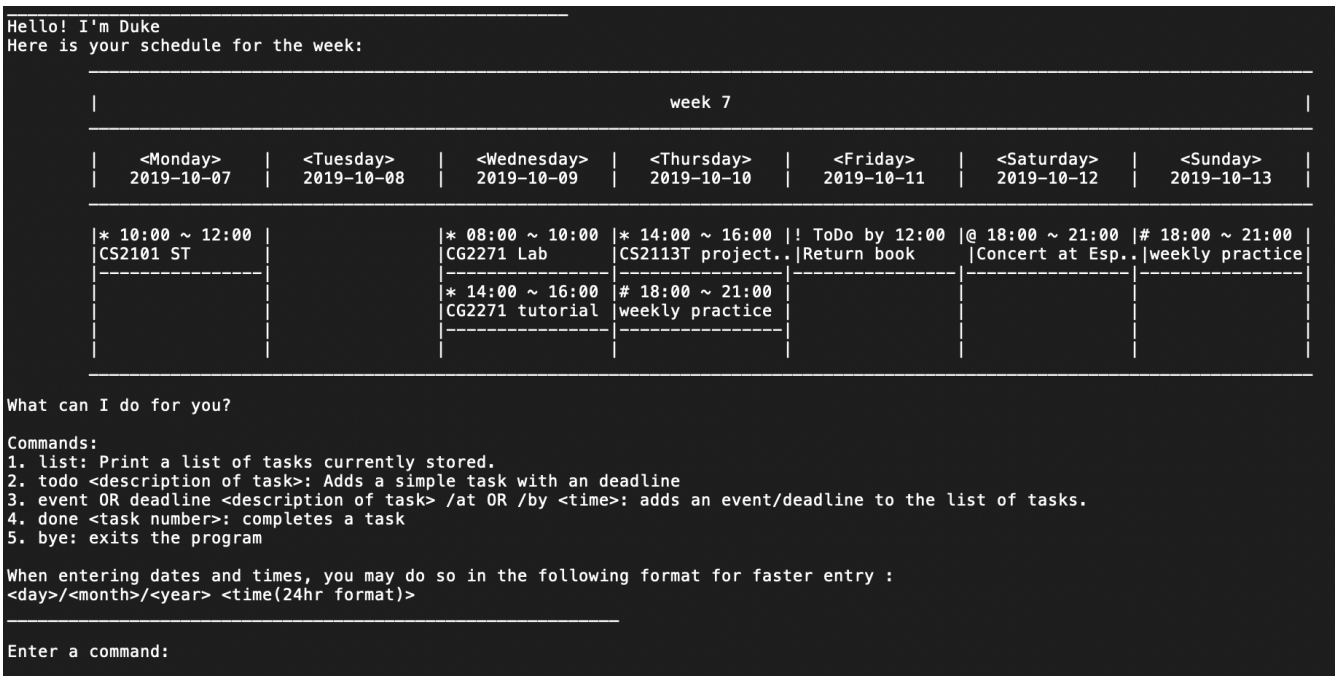


*Figure 1. The graphical user interface for **MisterMusik**.*

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.

2.1.3. Contact management

1. What it does: Each event in the list has a contact list to store the information of contacts which are relative to this event. The commands `contact add <event index> /<name>, <email> <phone number>`, `contact delete <event index> <contact index> /`, `contact edit <event index> <contact index> <edit type> /<new contact information>`, `contact view <event index> /` allow users to add, delete, edit, and view contacts in the list. Each *Contact* item includes name, email address, and phone number.
2. Justification: For every type of event, there are some people need to be contacted. Contact management enables users to add, delete, edit, and view sets of information (includes name, email, and phone number) for the people to contact. Also, users can edit only the name, email, or phone number of a set of contact information.

2.2. Code contributed:

Please click these links to see a sample of my code: sample code

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. 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
    e. Add information about contact management: #119, #126

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:

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

```
Enter a command:
lesson CG /13-08-2019 1000 1200 /7
```

- The message that shows events have been added will be displayed.

```
____________________________________________________________
Got it. I've added these recurring events:
[X][L] cg  START: Tue, 13 Aug 2019, 10:00 END: Tue, 13 Aug 2019, 12:00 (every 7 days)
Now you have 17 events in the list.
____________________________________________________________
```

- And you can type `list` and press `Enter` to check whether the events have been added. The following figure shows the message outputs after `list` operation.

```
____________________________________________________________
1. [X][L] cg  START: Tue, 13 Aug 2019, 10:00 END: Tue, 13 Aug 2019, 12:00
2. [X][L] cg  START: Tue, 20 Aug 2019, 10:00 END: Tue, 20 Aug 2019, 12:00
3. [X][L] cg  START: Tue, 27 Aug 2019, 10:00 END: Tue, 27 Aug 2019, 12:00
4. [X][L] cg  START: Tue, 03 Sep 2019, 10:00 END: Tue, 03 Sep 2019, 12:00
5. [X][L] cg  START: Tue, 10 Sep 2019, 10:00 END: Tue, 10 Sep 2019, 12:00
6. [X][L] cg  START: Tue, 17 Sep 2019, 10:00 END: Tue, 17 Sep 2019, 12:00
7. [X][L] cg  START: Tue, 24 Sep 2019, 10:00 END: Tue, 24 Sep 2019, 12:00
8. [X][L] cg  START: Tue, 01 Oct 2019, 10:00 END: Tue, 01 Oct 2019, 12:00
9. [X][L] cg  START: Tue, 08 Oct 2019, 10:00 END: Tue, 08 Oct 2019, 12:00
10. [X][L] cg  START: Tue, 15 Oct 2019, 10:00 END: Tue, 15 Oct 2019, 12:00
11. [X][L] cg  START: Tue, 22 Oct 2019, 10:00 END: Tue, 22 Oct 2019, 12:00
12. [X][L] cg  START: Tue, 29 Oct 2019, 10:00 END: Tue, 29 Oct 2019, 12:00
13. [X][L] cg  START: Tue, 05 Nov 2019, 10:00 END: Tue, 05 Nov 2019, 12:00
14. [X][L] cg  START: Tue, 12 Nov 2019, 10:00 END: Tue, 12 Nov 2019, 12:00
15. [X][L] cg  START: Tue, 19 Nov 2019, 10:00 END: Tue, 19 Nov 2019, 12:00
16. [X][L] cg  START: Tue, 26 Nov 2019, 10:00 END: Tue, 26 Nov 2019, 12:00
17. [X][L] cg  START: Tue, 03 Dec 2019, 10:00 END: Tue, 03 Dec 2019, 12:00
____________________________________________________________
```

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 the date and time of an existing event in the list.

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

Example:
Let's say that there is the event with date and time: 30-10-2019 1000 1200 in the list, and its index in the event list is 2. You want to change the date and time of this event to 11-11-2019 1200 1300.
Instead of deleting the existing event and adding a new one with updated date and time, you can easily type `reschedule 2 11-11-2019 1200 1300` to reschedule it.

To reschedule the existing event:

- Type `reschedule 2 11-11-2019 1200 1300` into the command line, and press `Enter` to execute it.

```
reschedule 2 11-11-2019 1200 1300
```

- The message "" will be displayed.

```
Rescheduled event to [X][P] b  START: Mon, 11 Nov 2019, 12:00 END: Mon, 11 Nov 2019, 13:00 successfully!
```

- And you can type `list` and press `Enter` to check whether the event has been rescheduled. The following figure shows the list after rescheduling.

```
1. [X][L] a  START: Thu, 10 Oct 2019, 10:00 END: Thu, 10 Oct 2019, 12:00
2. [X][P] b  START: Mon, 11 Nov 2019, 12:00 END: Mon, 11 Nov 2019, 13:00
```

3.3. Contact management : `contact`

The user is able to add, delete, view, and edit contacts information of an existing event in the list. A contact set includes name, email address, and phone number.

1. Add a contact set into an existing event
   Format: `contact add <event index> /<name>, <email>, <phone number>`
2. Delete a contact set in an existing event
   Format: `contact delete <event index> <contact index> /`
3. View contact list of an existing event
   Format: `contact view <event index> /`
4. Edit one type of information in the contact set of an existing event
   Format: `contact edit <event index> <contact index> <edit type> /<new contact information>`
   The edit type could be one of name, email, and phone.

Example:
Let's say that you want to manage your TA's contact information to a lesson event whose index is 1.
- Type `contact add 1 /TA: Jason, jason@u.nus.edu, 87311432` and press `Enter` to add it.

```
contact add 1 /TA: Jason, jason@u.nus.edu, 87311432
Ok, the contact has been added to the event.
```

- Type `contact view 1 /` and press `Enter` to view the contact list of the event.

```
contact view 1 /
Here is the list of contacts for the following event [X][L] a  START: Sat, 12 Oct 2019, 12:00 END: Sat, 12 Oct 2019, 13:00
1. Name: TA: Jason Email:  jason@u.nus.edu Phone Number:  87311432
```

- Type `contact edit 1 1 phone /87311433` and press `Enter` to edit the phone number of the first contact set in the event.

```
contact edit 1 1 phone /87311433
The contact has been edited to: Name: TA: Jason Email:  jason@u.nus.edu Phone Number: 87311433
```

- Type `contact delete 1 1 /` and press `Enter` to delete the first contact set of the event.

```
contact delete 1 1 /
Ok, the contact has been deleted from the event.
```

If the event does not have any contact set in its contact list, the message "Do not have any contact in this event." will be displayed.

4. Contributions to the Developer Guide

The following section shows my additions to the MisterMusik Developer Guide for recurring events feature.

4.1. Recurring events feature

4.1.1. How it is implemented

The program is able to detect recurring events and their periods when creating new events. When the user enters the command to add a new *Lesson* or *Practice* event with a period (in days) followed, *createNewEvent* method will call *entryForEvent* to get the period.
If the new event is not a recurring event, the period value will be assigned to NON-PERIOD and then call the *addEvent* method in the *EventList* class.

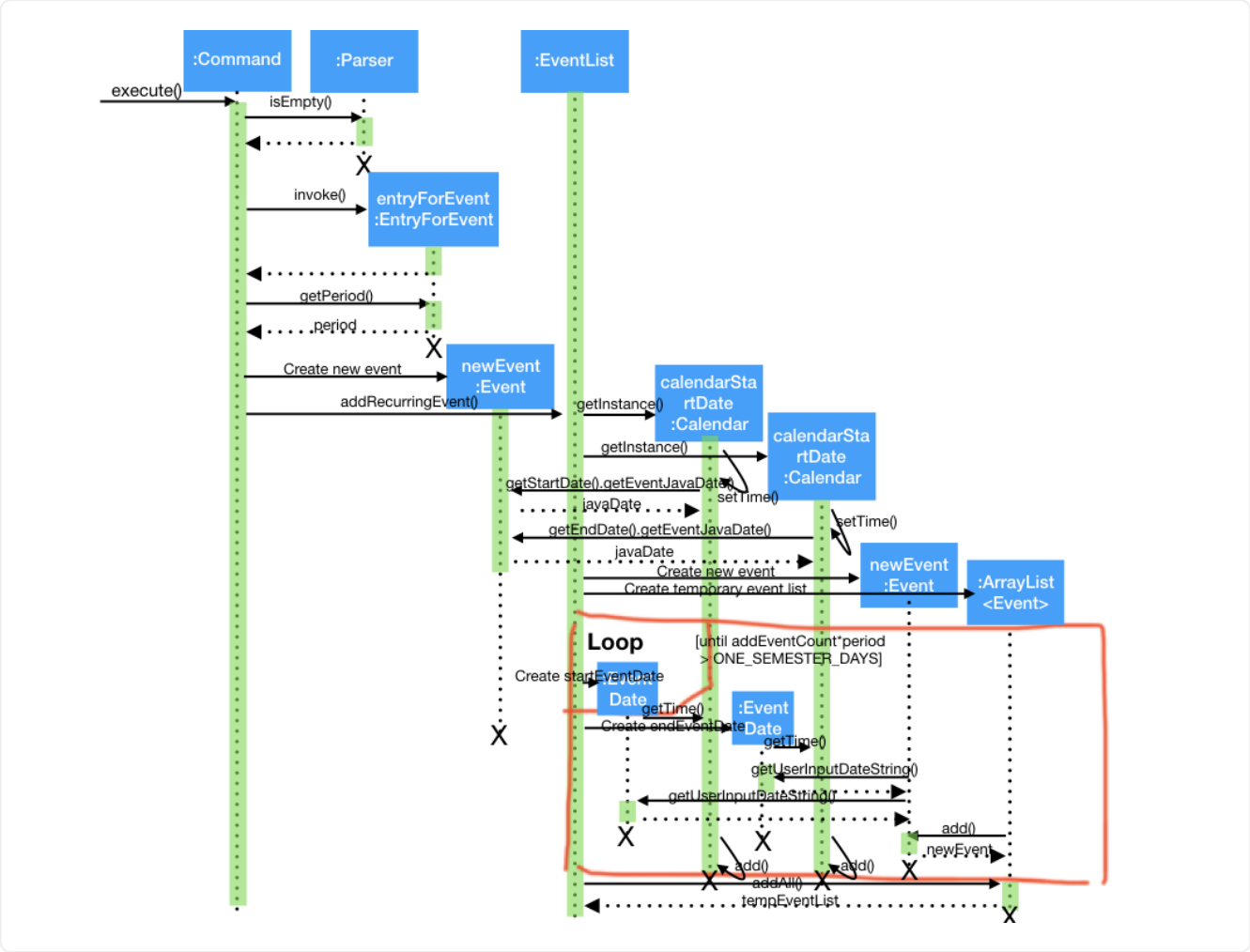
After getting the period, the *createNewEvent* method will call the *addRecurringEvent* method in the *EventList* class to create and store new events in the eventList.
The calculation of dates are done by Java Calendar, *Calendar.add* function is called to calculate the startDate and endDate of new events in Java Date type. The number of recurring events is depended on the period, since the maximum date between the first recurring event and the last one is up to ONE_SEMESTER_DAYS which is assigned to 16 weeks (112 days) now.
When creating the *startEventDate* and *endEventDate* objects of the new event, *calendar.getTime* is called and the *identifier* in EventDate will be assigned to DATE_TO_STRING, so that the *startDateAndTime* and *endDateAndTime* are in String type, which fits the requirement of the *Event* class.

All the events created in the *addRecurringEvent* method will be checked whether having clash with the events in the current eventList and then added in a temporary event list one by one. If no clash happens, the temporary event list will be added to the current event list.

Given below is an example usage scenario compared to adding non-recurring event.
Recurring event: `lesson` or `practice <event description> /dd-MM-yyyy HHmm HHmm /period(in days)`
Non-recurring event: `<event type> <event description> /dd-MM-yyyy HHmm HHmm`

4.1.2. Sequence diagram

The following sequence diagram shows how the adding recurring event operation works.



4.1.3. Why it is implemented this way

1. Whether the input command has a period is considered at the first, so that the dependency between adding recurrent events and adding normal events could be reduced.
2. The *add(int field, int amount)* method of *Calendar* class is used to add or subtract from the given calendar field and a specific amount of time, based on the calendar's rules. The calendar add function has the format: `public abstract void add(int field, int amount)`
3. Since the number of recurrent events with a short period could be large, it is more likely to have clashes with the current eventList. Hence, before added in the temporary event list, the new event need to be ensured that no clash happens.
4. To keep the format of creating new events, the format process of changing Java Date to String is done in the `EventDate` class instead of messing the `Event` class to accept both Date and String types as input date and time.