# Yuan Jiayi - Project Portfolio for MisterMusik



# 1. About the project

MisterMusik is a Command line interface (CLI) based application that allows for music students studying at the National University of Singapore(NUS) to have a centralized scheduling platform for all their music based activities. The user interface is entirely command line based and the application is written in java.
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 highlight 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, contact management, and help list.

### 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> name|email|phone /<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.

### 2.1.4. Help list

1. What it does: A help list including basic commands will be printed on the welcome page. The user is able to enter the first word of each command to see the detailed format of that command. Some commands are combined, for example, the user needs to enter `help event` to see the combined list of formats for adding todo and different types of events.
2. Justification: It is inconvenient for users to remember all the correct input formats of too many features and commands, so that a help list with basic command formats on welcome page could give users a clear map to find out what can MisterMusik do. Also, when users type a command with incorrect format, they would like to know the correct format of that specified command. Consider that printing out all the command formats together could lead to a long list which is hard to find and messy up the welcome page, the help list only contains basic command formats and commands to see the combined detailed formats.

## 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, and update UI image: #37, #38, #266
    c. Add AboutUs page to introduce the team: #37, #266
    d. Add Setting up and Documentation sections on Developer Guide: #266
    e. Restructure the User Guide to fit v1.4 and reader friendly: #242, #260, #266

# 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:** `lesson|practice <event description> /dd-MM-yyyy HHmm HHmm /<period(in days)>`
Note: The number of days must be an integer.

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

**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.
- The message that shows the event is rescheduled successfully and the new date and time will be displayed.
- And you can type `list` and press `Enter` to check whether the event has been rescheduled. The following figure shows the list after rescheduling.

## 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.

- Add a contact set into an existing event
  **Format:** `contact add <event index> /<name>, [<email>], [<phone number>]`

The email address and phone number are optional information for a contact set. If the user does not want to add email and phone number, just leave the information in [] blank, but remember to input `,`.

- Delete a contact set in an existing event
  **Format:** `contact delete <event index> <contact index>`
- View contact list of an existing event
  **Format:** `contact view <event index>`
- Edit one type of information in the contact set of an existing event
  **Format:** `contact edit <event index> <contact index> name|email|phone /<new contact information>`

**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-xxx, xxx@u.nus.edu, 87****32` and press `Enter` to add it.
- Type `contact view 1` and press `Enter` to view the contact list of the event.
- Type `contact edit 1 1 phone /87311433` and press `Enter` to edit the phone number of the first contact set in the event.
- Type `contact delete 1 1` and press `Enter` to delete the first contact set of 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.

## 3.4. Help list : help

The user is able to see a list of correct command formats on the welcome page of MisterMusik. The user is also able to seek help whenever he or she wants the correct format of a specified command.

**Format:** help [<command type>]

command type indicates the first word of the command. Or event to see the combined formats of adding todo and different types of events. Or change to see the combined formats of changing basic information including description, date, and status of a task in the list.

**Example:** Let's say that you forget the correct input format of rescheduling an event in the list.

- Type help reschedule|change and press Enter to see the correct format.

# 4. Contributions to the Developer Guide

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

## 4.1. Add recurring events

### 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.

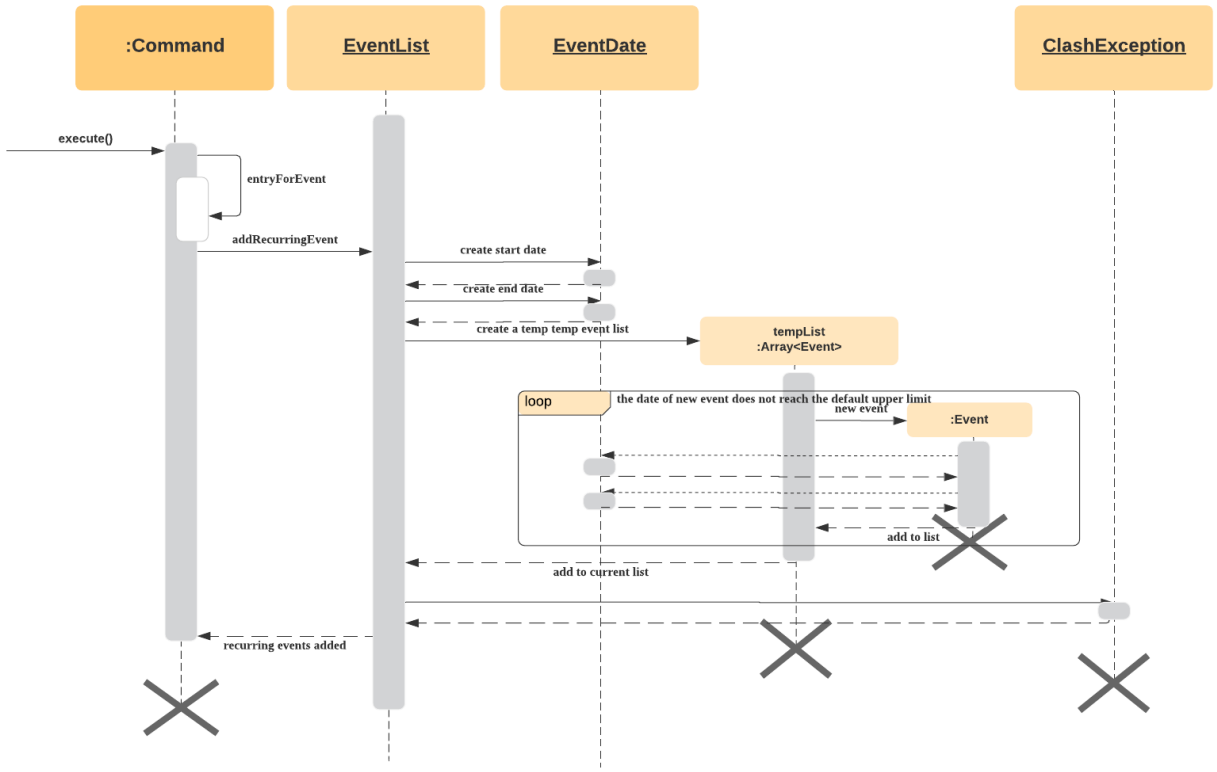
When creating the startEventDate and endEventDate of the new event, calendar.getTime is called and the identifier in EventDate will be assigned to DATE_TO_STRING, so that the startDateAndTime and endDateAndTime are in String type, which fits the requirement of the Event class.
All the events created 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 tempEventList will be added to the current eventList.

### 4.1.2. 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.
   `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.

### 4.1.3. Sequence diagram

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




### 4.1.4. Design considerations

Aspect: How to avoid adding infinite events

- **Alternative 1 (current choice):** Set a upper limit to ensure the interval between the first and last events added to list is not too long.

- Pros: Easy to implement. Easy to add lesson and practice in one semester.
- Cons: Not agility. May add too many extra events if users want short interval between the first and last recurring events.

- **Alternative 2:** Let users to set the interval between the first and last recurring events.
  - Pros: Will be more agility and user friendly.
  - Cons: Hard to unify the command format.

# 4.2. Contacts list

## 4.2.1. How is it implemented

The contacts list stores a list of contacts contain <name>, <email address>, and <phone number> information added by users. The list is stored in each individual event. When the user first creates the event, the event is created with an empty contact list.
**Users can add, edit, delete a specified contact item, and view the contacts list of a specified event.**

#### Adding a contact

Given below is an example to adding a contact:
`contact add <event number> /<name>, [<email address>], [<phone number>]`

1. When the command is entered, the manageContacts() method will be called.
2. The method will check whether the event ID is valid. Add a new Contact will be created with contact information in it.
3. The event corresponding to the ID entered along with its method addContact() will be called to add the contact into the contact list in the event.
4. The printContactAdded() method of the UI class will be called to reflect the change to the user.

## 4.2.2. Why is it implemented this way

1. For different events, users may have different people to contact with. So a empty contact list is created under each event. For some events, users may have more than one person to contact with. Hence, users are allowed add more contact items under one event.
2. We want to keep the information highly relative to contact, so users just need to add <name>, <email address>, and <phone number> for a contact item. Sometimes may not have the email or phone number of a person, so it also allows users to add one of <email address> and <phone number>.
3. The main propose of MisterMusik is schedule events, so the contact list is hidden on the basic list viewing. Users can use command `contact view <event number>` to see the contact list of the specified event.