MisterMusik - User Guide

By: `Team CS2113T-F11-4` Since: `Aug 2019` Licence: `MIT`

1. Introduction

MisterMusik is a scheduler program created 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.

2. Quick Start

1. Ensure you have Java 11 or above installed in your computer
2. Download the latest `mistermusik.jar`
3. Copy the file to the folder you want to use as the home folder for your MisterMusik
4. Double-click the file to start the app.
5. Type the command in the command box and press `Enter` to execute it.
   E.g. typing `help` and pressing `Enter` will open the help window.
6. Please refer to Section 3, “Features” for details of each command.

3. Features

1. Words in <> are the parameters to be supplied by the user
2. Items in [] are optional
3. Items with | in between them indicate the user can choose to use either of them

3.1. Help list : `help`

Brings up a list of available commands that the user can input. Users can also use `help <command type>` to see the combined or detailed commands.+ The help list will be printed every time MisterMusik is opened.

Format: `help [<command type>]` Note: Command types are the first word of following commands or some key words listed in the basic help list.

3.2. List : `list`

List all the events from the current date. Additionally it tracks which events are already over and prints out only future events, starting from the index of the nearest future event e.g. if the first and second events of the list are already over, it will list from the third event onwards while keeping the event index the same.

It also gives a reminder of which of the past events contain goals that are not yet marked as achieved.

Format: `list`

3.3. Reminders : `reminder`

Displays the list of tasks over the next three days in default. The user is also able to see the events in days they wants by typing in `reminder <days>`

Format: `reminder [<reminder days>]`
Note: The number of days must be an integer.

3.4. Adding a to-do task : `todo`

Adds a task defined by the user to the to-do list with a deadline.

Format: `todo <name of task> /dd-MM-yyyy [HHmm]`

Reminder that the bracketed input is optional.

E.g. `todo complete theory homework /10-10-2019` adds a deadline to complete the user's theory homework by 10 October 2019.

3.5. Adding an event task :

Adds an event defined by the user to the list of tasks. The former and latter time inputs denote the start and end time respectively.

For recurring events, please see Recurring events.

1. Practice sessions : `practice`
   Format: `practice <practice description> /dd-MM-yyyy HHmm HHmm`
2. Lessons : `lesson`
   Format: `lesson <lesson description> /dd-MM-yyyy HHmm HHmm`
3. Examinations : `exam`
   Format: `exam <exam description> /dd-MM-yyyy HHmm HHmm`
4. Recitals: `recital`
   Format: `recital <recital description> /dd-MM-yyyy HHmm HHmm`
5. Concerts : `concert`
   Format: `concert <concert description> /dd-MM-yyyy HHmm HHmm /[<cost of concert>]`

Cost of concert must be an integer (for simplicity regarding budgeting).

3.6. 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.

E.g. “lesson Theory class /18-09-2019 0900 1030 /7” adds a weekly recurring event from 9:00am to 10:30am, starting on 18 September 2019.

3.7. Calendar Table : `calendar`

The calendar table is generated from the `EventList`. It prints on the screen a table of calendar of 7 days starting from a specified day, including the events within this time period.

3.7.1. Commands for CalendarView

- `calendar` This prints the calendar table of this 7 days.
- `calendar next` This prints the calendar table of the next 7 days.
- `calendar last` This prints the calendar table of the last 7 days.
- `calendar on` Allow the calendar to be printed after every command execution.
- `calendar off` Not allowing the calendar to be printed after every command execution.

3.8. Viewing schedules : `view`

Brings up a list of events for a particular date.

Note: This only displays the schedule for events. Viewing of todo tasks on a particular date will be implemented in v2.0.

Format: `view dd-MM-yyyy`
E.g. “view 18-09-2019” displays all events on the date 18 September 2019.

3.9. Checking for free days : `check`

The user can check for the nearest days that are free. This will list the next 3 free days on the users' schedule. A day is considered free if there are no events scheduled. ToDos are not counted as events.

Format: `check`

3.10. Marking a todo as done : `done`

Marks a ToDo in the list as completed, denoted by a tick when displaying the task. This functionality does not work for events that are not ToDos.

Format: `done <task index>`
E.g. “done 5” marks task number 5 as done.

3.11. Rescheduling events : `reschedule`

The user is able to reschedule an existing event.

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

3.12. Search : `find`

The user is able to search for a specific event/task using keywords.

Format: `find <keyword>` will display all events/tasks previously entered containing the desired keyword.

3.13. Edit an event/ToDo : `edit`

The user is able to edit the name of the event/ToDo he entered in case he entered it wrongly.

Format: `edit <index for edition>/<new description>`

3.14. Contact management : `contact`

The user is able to add, delete, view, and edit contacts information of an event or todo. A contact includes name, email address, and phone number.

3.14.1. Add contact item

Format: `contact add <event index> /<name>, [<email>], [<phone number>]` This adds a contact to a specific event.
Users can type `Space` instead of `<name>` or `<email>` or `<phone number>` if the user do not want to write in one or two type of information. (e.g. `contact add 1 /name, ,12345678` adds a contact set without email address to the first event)

3.14.2. Delete contact item

Format: `contact delete <event index> <contact index>` This deletes a specified contact.

3.14.3. View contact

Format: `contact view <event index>` This displays the contacts list of a specified event.

3.14.4. Edit contact item

Format: `contact edit <event index> <contact index> <edit type> /<new contact information>` This edits an existing contact.
The edit types are name, email, and phone.

3.14.5. Storing contact lists in a text file : [coming v2.0]

Contact lists now cannot be seen after restarting MisterMusik. In future, contact lists will be stored into a text file automatically. The user will be able to edit them in the text file.

3.15. Goals list : goal

Goals list of each event helps the user keep track of the outcome that they wishes to achieve by the end of the event. The user is able to add, edit, delete or set a goal as achieved.

3.15.1. Adding a goal

`goal add <event index>/<input goal>` This adds a goal to a specific event.

3.15.2. Editing a goal

`goal edit <event index> <goal index>/<new input goal>` This edits an existing goal.

3.15.3. Deleting a goal

`goal delete <event index> <goal index>` This deletes a specified goal.

3.15.4. Setting a goal as achieved

`goal achieved <event index> <goal index>` This sets the specified goal as achieved.

3.15.5. Viewing the goal list

`goal view <event index>` This displays the goals list of a specified event along with their status - whether a goal is achieved or not.

3.15.6. Storing goals lists in a text file : [coming v2.0]

Like the contact lists, the goal lists now cannot be seen after restarting MisterMusik. In future, goal lists will be stored into a text file automatically. The user will be able to edit them in the text file.

3.16. Checklists : checklist

Checklist of each event can be used to remind users of certain items (e.g. bring glasses to concert). This is implemented by storing an array list of strings in `Event` objects. Storing checklist data to files will be available in v2.0.

Checklist implementation contains 4 operations:

3.16.1. add checklist item

`checklist add <event index>/<checklist item>` This adds an item to a specific event's checklist.

3.16.2. view checklist

`checklist view <event index>` This displays the checklist of a specified event.

3.16.3. edit checklist item

`checklist edit <event index> <item index>/<new item>` This edits a specific item in the checklist of an event.

3.16.4. delete checklist item

`checklist delete <event index> <item index>` This deletes an item from the checklist of an event.

3.17. Budget handling : `budget`

3.17.1. Viewing monthly costs of concerts

The total cost of concerts each month can be viewed by the user using the `budget` command.

Format: `budget MM-yyyy` symbolising the month and year the user wishes to take a look at.
Note: MM must be a two digit value. For example, the month of may, 2019 must be entered as `05-2019` instead of `5-2019`.

3.17.2. Setting new monthly budget

The user is able to set the budget for concerts of each month by using the `budget` command.

Format: `budget set <new budget>`
FOR example, `budget set 87` will set the budget for every month to $87.

Storing the user-defined budget to a txt file upon shutdown will be added in [v2.0]

3.18. Instruments handling : `instruments`

This function allows the user to store maintenance information about the instruments that the user possesses.

3.18.1. Adding an instrument

`instrument add /<instrument name>` This adds an instrument named `<instrument name>` to the list of instruments stored in the system.

3.18.2. Viewing list of instruments stored in the system

`instrument view instruments` This lists out the instruments stored in the system in order of their indexes.

3.18.3. Servicing an instrument

`instrument service <instrument index> /<brief description of servicing> /<date>` This adds the information that the instrument with instrument index `<instrument index>` is serviced on `<date>`.

3.18.4. Viewing list of instruments stored in the system

`instrument view services <instrument index>` This lists out the servicing done to the instrument with instrument index `<instrument index>`.

3.19. Exiting the program : `bye`

Exits the program.

Format: `bye`

3.20. Rating a practice session : `[coming v2.0]`

The user will be able to rate the efficiency of a particular practice session after completing it. When the practice session is marked as done, the user will be prompted to add an efficiency rating and any extra comments or feedback.

Format: `rate <task index>`

3.21. Adding details to events : `[coming v2.0]`

The user will be able to add any extra details to an event when adding it to the task list. For practice sessions or lessons, it can be used to take feedback from instructors and lesson notes respectively.

Format: `details <task index>` or `notes <task index>`

3.22. Searching for a free slot : `[coming v2.0]`

The user will be able to search for any days within the next 2 weeks that contains a specified period of time that is free.
For example, searching for a 4h free slot will display all the dates within the 2 weeks with at least 4 hours of free time in the schedule.

Format: `searchfree 4h`

4. Additional Features

4.1. Detecting Clashes :

When the user inputs a new event, MisterMusik will check if it has any clashing date and time with existing events and warns the user of the clash. This also detects clashes when recurring events are

entered, so there is no need to manually check for schedule clashes.

4.2. Sorting By Date :

MisterMusik will automatically sort the list by the date and time of each task so the user will not have to manually prioritise each event and todo task. ToDos are listed at the front of each day as they do not have an included time.

4.3. Budget handling system :

The system allows for the user to handle their own monthly budget. There is a set budget for each month (fixed at $50 initially) that stops users from entering new concerts into the schedule if that new concert would cause the costs of concerts for the month to exceed the stipulated budget.

For example, if the list contains a concert on the 4th of July, 2019 that costs $49, entering a new concert that costs more than a dollar would causes the system to cancel the operation as it would lead to monthly costs of $51 that exceeds the monthly budget of $50.

The user also has the ability to change the stipulated budget for each month or universally if he/she wishes to.

4.4. Reminder for specific task types : `[coming v2.0]`

The user will be able to get reminders of specific task type using the reminder command and specifying which type they would like to be displayed.

E.g. “reminder exam” displays a reminder of all exam events within the next 3 days.

5. FAQ

Q: How do I transfer my data to another Computer?
A: Install the app in the other computer and overwrite the empty data file it creates with the file that contains the data of your previous MisterMusik folder.

6. Glossary

1. Event: an event refers to any activity with a start and end time, includes concerts, practices, exams, and recitals.
2. Todo: a todo refers to a task with a deadline specified by the user.