MisterMusik - Developer Guide

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

1. Setting up

2. Design

2.1. Overall Architecture

There are three main components in the overall architecture of the application.

`Duke` class: contains the `main()` class of the application. * Upon startup: a Welcome Message is printed (see section under Ui) data is loaded from the memory storage to the system. * Upon exit (when the user enters 'bye'): a Goodbye Message is printed (see section under Ui).

`Events` Package: contains classes pertaining to the events.

`UserElements` Package: contains classes pertaining to I/O and the interaction between user and system.

2.2. UI component

2.3. Model component

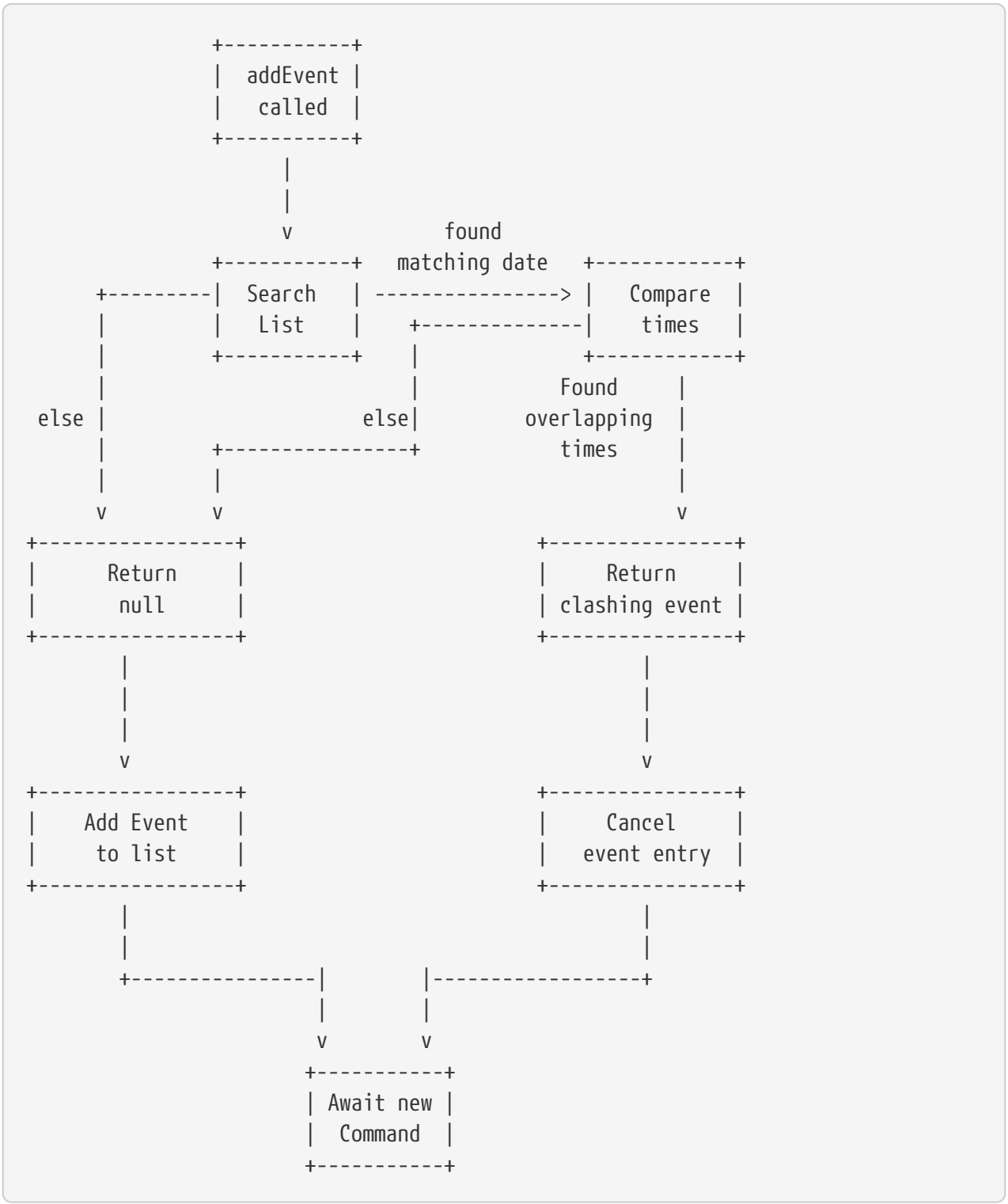
2.4. Storage component

2.5. Common classes

3. Implementation

3.1. Clash handling

3.1.1. Activity diagram



3.1.2. How is it implemented

The program is able to detect clashes when creating new events. When the user enters the command to add a new Event entry to the list, the method EventList.addEvent is called from the Command class upon execution.

The addEvent method will then call the clashEvent method to check the existing entries for any clash in schedule. This is done by first searching the list for an event that has a matching date with the new event.

If no such event is found, the method returns a null value, indicating that there is no schedule clash. If an event is found with a matching date, the method then calls the EventList.timeClash method to check whether the two events have overlapping time periods. This is done by simple mathematical comparison using > and < operators, comparing the times in 24h format.

If there is any overlap, the timeClash method will return a true boolean value, indicating there is a schedule clash. The clashEvent method then returns the specific task that caused the clash so that the details of the clash can be printed for the user, in order for him/her to fix the schedule conflict.

3.1.3. Why it is implemented this way

The process of checking for a clash was implemented as small, separate abstracted methods so as to ensure scalability, easier testing, and to reduce dependencies.

Because of the way the clash detection was split into multiple small methods, it was easy to implement this clash detection as a part of adding recurrent events (to check for clashes when recurrent events were automatically entered) as well as the rescheduling function (to check for clashes when the user attempts to reschedule an existing event, so that he/she does not inadvertently create a new schedule conflict).

3.2. Viewing specific dates

3.2.1. How is it implemented

The implementation is a simple for loop that runs through the existing task list. If a matching date is detected, it will return the corresponding task and add it to a temporary list of found tasks. After running through the whole list, the temporary list will be printed out to display all the tasks of a specific date.

Step 1: When the command "view <date>" is given, the viewEvents() method will be called.

Step 2: A temporary ArrayList is created by the method to be populated.

Step 3: The date string from the input command is passed into the EventDate class to be formatted into the same format as that of each event and returned as a string.

Step 4: The returned string is compared with each task in the event list to check for any event with a matching date.

Step 5: When an event with a matching date is found, the event is added to the temporary list.

Step 6: After the entire list has been checked, the temporary list is passed into a UI instance

Step 7: The printFoundEvents() method will be called. The said method then prints out the temporary list, displaying the list of events taking place on a specific date.

In the situation when an empty temporary list is passed into the UI for it to print, an exception will occur and the printFoundEvents() method will catch the exception before printing out a string to inform the user that there are no tasks taking place on that specific date.

3.2.2. Activity Diagram

[DGViewEventsActivityDiagram] | *images/DGViewEventsActivityDiagram.png*

3.2.3. Why is it implemented this way

The matching events are being stored individually into a separate temporary list before being printed out. This is to allow an easier handling of individual tasks as separate instances in case the user wishes to edit a particular task from the temporary list.

3.2.4. Alternative implementations considered

Storing all the matching events as a single string, passing the string into the printFoundEvents() method to print out. This implementation is undesirable as it will be difficult to access individual matching events in the case the user wishes to edit them as mentioned above.

4. Documentation

5. Testing

6. Dev Ops

Appendix A: Product Scope

A.1. Target User Profile

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.

Appendix B: User Stories

1. As a busy music student with multiple classes, I want to be able to track my practice sessions so that I won't miss any trainings.
2. As a music student with a heavy workload, I want to be able to track my upcoming recitals and their details so I can prioritize which pieces/what techniques to practice and focus on.
3. As a music student with a tendency to procrastinate in things I need to do, I want to be constantly reminded of my examination dates so i do not wait till the last minute to prepare for them.
4. As a student who wants to maximize my efficiency in practice sessions (performer), I want to be able to rate and comment on the efficiency of my practice sessions and be able to review them

to make sure that I learn and improve faster.

5. As a student who wants to categorize what I learn in classes, I want to be able to take notes and organize them into different categories, so that I can easily review it anytime.
6. As a student who wishes to be aware of his upcoming events, I wish to be able to view my schedule within a selected number of weeks so that I can plan for it.

Appendix C: Use cases

C.1. Entering a practice schedule into the system

1. User enters a command to add a practice followed by a date, and the details of the intended practice.
2. System prompts user about whether or not he wishes to make this a recurring practice (e.g every Tuesday).
3. User responds to the prompt accordingly.
4. System adds practice session to a stored list and saves it to a file on the user's hard drive.

C.2. Entering the details of a recital into the system

1. User choose to enter a recital.
2. System requires details of the recital.
3. User enters date, time, venue, the pieces to be performed, and a description if needed.
4. System adds recital to a stored list and saves it to a file on the user's hard drive.

Extension 3.1: System detects there is a clash with a concert/practice session at step 3. 3.1.1. System generates warnings and ask user to delete the corresponding entry and forgo that event. 3.1.2. User responds to the warning, deleting or rescheduling one of the events in a clash if necessary. 3.1.3. System updates the stored list and saves it to the file on the user's hard disk.

Extension 3.2: System detects a clash with another recital or an examination at step 3. 3.2.1. System generates warnings and ask user to reschedule and re-enter one of the events that clash. 3.2.2. User responds to the warning, rescheduling and re-entering one of the events in a clash. 3.2.3. System updates the stored list and saves it to the file on the user's hard disk. Use case ends

C.3. Making notes

1. User chooses to enter a command to start a note-taking/viewing session
2. System shows the user a list of categories (directories) of notes that have been made previously and prompts the user to enter one or create a new file.
3. User makes a choice to enter a category directory or create a new one.
4. System shows the user a list of files containing notes, each file is named with the corresponding date, and a description of the file decided by the user. System also prompts the user if he wants

to create a new file or enter an existing one.

5. User selects a file to enter or chooses to create a new file.
6. System opens the file for viewing and editing. The user can choose to make changes to the notes using commands: delete, add and move to delete, add or move notes around the file accordingly.

Extension 3.1: System detects there is no category file that user commands to edit on 3.1.1. System generates warnings and ask the user whether or not he wishes to add a new category and take notes in 3.1.2. User responds to the warning, and choose to add a new category 3.1.3. System adds and opens a new category file with the name given by user

C.4. Viewing upcoming events

1. User enters command `list` followed by an integer N representing the number of weeks in advance he would like to view his schedule.
2. System responds by showing the user a list of events in the next N weeks.
3. User may enter a command to remove specific tasks from the list, to have a cleaner viewing experience.

Extension 3.1: System can remove types of events from list at user's command 3.1.1. User can enter commands `remove examinations` for example, to remove the examinations from the display. 3.1.2. System responds accordingly, removing the type of event from the list and altering the list of events displayed.

Extension: 3.2. System can allow the user to only view a specific type of event. 3.2.1. User can also enter commands `show examinations` for example, to only show examinations in the list of events. 3.2.2. System will respond accordingly, displaying only type of event that the user specified.

Extension: 3.3. System can allow the user to only view events on a specific date. 3.3.1. User can enter the command "view <dd-MM-yyyy>"" to view all events on that specific date. 3.3.2. System will display the events, and the user can edit the events accordingly.

C.5. Entering the examination details

1. User enters command `examination`, together with a description of the examination, along with the date, time, venue and any other notes in a given format.
2. The system adds the examination to the list of events.

Extension 1.1: System detects a clash with the examination date entered and another event. 1.1.1. System will prompt the user to reschedule the event of lower priority. (examinations and recitals are of highest priority, followed by concerts and then practices). If the clash is with an event of the same priority, the user is prompted to choose which one to reschedule(simple y/n response). 1.1.2. User reschedules the specified event by entering a date and time. The user is also able to delete an event with the delete command to free up the schedule if he wishes to do so. 1.1.3. System once again checks for clashes and repeats the process of rescheduling if necessary.

C.6. Rate and comment on the efficiency of practice sessions

1. User enters a command to rate a practice session
2. System brings up a list of practice sessions that the user has already completed
3. User selects a practice session
4. System displays the details of the selected practice session and prompts the user to select an efficiency rating along with any additional notes
5. User rates the efficiency of the practice session and takes down any notes or feedback from their instructor
6. System saves the entry onto the hard disk

C.7. Edit descriptions of any existing event

1. User enters a command to edit the description of an event
2. System edits corresponding description and prompts user of success

Extension 1: System detects a clash of incorrect format entered by the user. 1.1. System will prompt the user that the entered format was incorrect.

Extensions 4.1: Selected practice session has already been rated. 4.1.1. If the selected practice session has already been rated, the system notifies the user and allows them to edit. 4.1.2. The user edits the rating and notes of the practice session accordingly 4.1.3. System saves the changes onto the hard disk

Extensions 5.1: User inputs an invalid rating. 5.1.1. System displays an error message to inform the user of the correct rating format until a valid input is detected.

C.8. Viewing all events in the coming 7 days.

1. User enters a command to view all events in the next 7 days.
2. System shows all events in the next 7 days in a calendar table.

Appendix D: Non-functional requirements

1. System should work on windows and linux.
2. System response within 1 second.
3. Usable by non tech-savvy individuals.
4. Clear user prompts
5. Visually pleasing display

Appendix E: Instructions for Manual Testing

E.1. Launch and Shutdown

1. Initial Launch
   a. Open Duke.java in src.
   b. Run the file
   Expected: The Welcome Message is printed in stdout and the system requests for an input.
2. Shutdown
   a. Key in `bye` as input.
   Expected: The Goodbye Message ('Bye. Hope to see you again soon!') is printed in stdout and the process exits.

E.2. Obtain the list of items currently in stored in the system

1. Key in `list` as input
   Expected: A list of items is printed in stdout.

E.3. Add a Task

1. Add Todo
   Key in `todo tdtask /01-01-2011 0100`
   Expected: Output should be
   Got it. I've added this task:
   [x][T] tdtask BY: Sat, 01 Jan 2011, 01:00
   Now you have <previous number of items + 1> items in the list.
2. Add Exam
   Key in `exam Music Rudiments /08-08-2018 0800 0900`
   Expected: Output should be
   Got it. I've added this task:
   [x][E] Music Rudiments START: Wed, 08 Aug 2018, 08:00 END: Wed, 08 Aug 2018, 09:00
   Now you have <previous number of items + 1> tasks in the list.
3. Add Practice session
   Key in `practice morningprac /07-08-2018 0800 0900`
   Expected: Output should be
   Got it. I've added this event:
   [x][P] morningprac START: Tue, 07 Aug 2018, 08:00 END: Tue, 07 Aug 2018, 09:00
   Now you have <previous number of items + 1> tasks in the list.
4. Add Concert
   Key in `concert Noon Concert /06-06-2016 1200 1400`
   Expected: Output should be
   Got it. I've added this task:

[x][C] Noon Concert START: Mon, 06 Jun 2016, 12:00 END: Mon, 06 Jun 2016, 14:00
Now you have <previous number of items + 1> tasks in the list.

5. Add Recital
   Key in `recital Evening Recital /07-07-2017 1900 2100`
   Expected: Output should be
   Got it. I've added this task:
   [x][R] Evening Recital START: Fri, 07 Jul 2017, 19:00 END: Fri, 07 Jul 2017, 21:00
   Now you have <previous number of items + 1> tasks in the list.
6. Add Lesson
   Key in `lesson Class /09-09-2019 0900 0100`
   Expected: Output should be
   Got it. I've added this task:
   [x][L] Class START: Mon, 09 Sep 2019, 09:00 END: Mon, 09 Sep 2019, 01:00
   Now you have <previous number of items + 1> tasks in the list.

E.4. Set a Task as done

Key in `done 1`
Expected: Prints a message that the first task on the list has been marked as done.

E.5. Find tasks that contains a certain keyword or key phrase

Key in `find <key>`, where `<key>` is the keyword or key phrase
Expected: Prints a list of tasks that contains the `<key>`.

E.6. Get reminder

Key in `reminder`
Expected: Prints a list of tasks to be completed in the next three days.

E.7. View all tasks in a certain day

Key in view `<date>`, where `<date>` has the format dd/MM/yyyy.
Expected: Prints a list of tasks that occurs on the given `<date>`.

E.8. Edit description of an event

Key in `<edit> <event index>/<new description>`.+ Expected: Prints the success of editing event of index `<event index>`.

E.9. Reschedule an event

Key in `<reschedule> <event index> <new date> <new start time> <new end time>`, where `<new date>`

`<new start time> <new end time>` has the format dd-MM-yyyy HH:mm HH:mm.
Expected: Prints the success of rescheduling event of index `<event index>`.

E.10. Check the next 3 free days

Key in `check`.
Expected: Prints the next 3 days without any events.

E.11. View the calendar table of the next 7 days.

Key in `calendar`.+ Expected: Prints the calendar table containing all events of the next 7 days, where the first column of the table is the current day.

Appendix F: Instructions for Manual Testing

F.1. Launch and Shutdown