Fei Dong – Project Portfolio

Project: MisterMusik

Overview

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. The target audience for this project is NUS Music students. The program is written in java and the user interface is command-line based.

Summary of contributions

Code contributed: [functional code] (click on the link to see my contributions)

Major enhancement: Created a reminders function

- What it does: allows the user to view all the upcoming tasks for the next input number of days. If no input is given, the upcoming tasks for the next three days are displayed.
- Justification: This feature is important as the one of the purposes of the application is to help to improve the efficiency of which Music students complete their work. The reminders function serves this purpose as to helps to alert music students to any important upcoming tasks that they may have forgotten due to their busy lifestyles.
- Highlights: The main challenge of this feature was to filter out the relevant tasks that are to be displayed in the reminders function.

Major enhancement: Created an instruments handling function

- What it does: allows the user to keep track of their instruments and their last serviced dates.
- Justification: This feature is important as musical instruments are the serious music students' most valued possessions, and those who have studied music seriously before will understand that these instruments require regular servicing to be at their best. Thus, by keeping track of the past servicing dates of the Music students' musical instruments, this feature improves the efficiency of the music students lives as they can just use the app to find out and record down information on the servicing of their musical instruments.
- Highlights: The key challenge of this feature was that there are three key elements required in the implementation: a list of instruments, the instruments themselves and the servicing of the instruments. This requires careful handling to ensure that the code is efficient.

Minor enhancement:

Implemented a class named DateObj that helps to handle dates more efficiently. (This is later renamed to EventDate and enhanced by my team members)

Other contributions:

- Created and added basic information as well as some manual test cases to the Developers' Guide: #43

Contributions to the User Guide

The following is an excerpt from our user guide that shows the enhancements that I have added to the final product, i.e. reminders and instruments handling.

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.

3.18. 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>`.

Contributions to the Developer's Guide

The following is an excerpt from our developers' guide that shows some of enhancements that I have added to the final product, i.e. reminders and instruments handling.

3.10. Reminders

3.10.1. How it is implemented

The reminder function filters out the tasks that are due or are happening before 2359 an input number of days after the current date, and prints them out as a reminder for the users. If the number of days is not provided, the number of days is 3 by default.

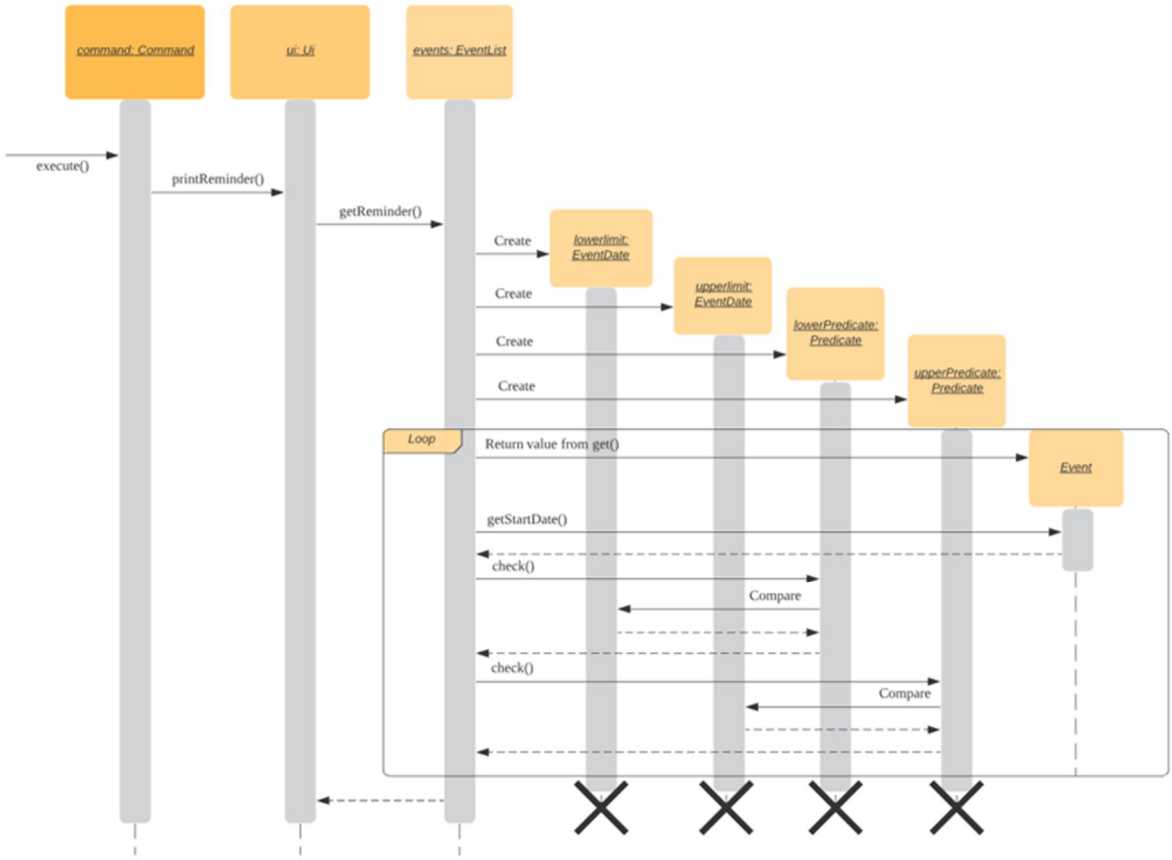
After the user enters 'reminder', the Command.execute method calls Command.remindEvents, which in turn calls the Ui.printReminders function. The Ui.printReminders function calls the EventList.getReminder method, which uses the EventList.filteredList method to filter out a list of events that are due or are happening before 2359 the input number of days after the current date. The EventList.filteredList method filters out events from the stored list of events according to a certain input predicate.

The constructor of the Predicate class takes in two arguments: the reference and the comparator. The reference is the item that is used for the comparison reference (comp) input, and the comparator is the operator that is used for the comparison. The comparator should be either one of the three global integer variables: EQUAL, GREATER_THAN or SMALLER_THAN. The Predicate.check method takes in an input and checks if reference (comp) input is true by calling the appropriate method in the Predicate class depending on the type of the reference and input.

In the EventList.getReminder method, the reference of the input Predicate object is set to an eventDate object set to 2359 the input number of days after the current date, and the comparator is GREATER_THAN. After that, the EventList.getReminder method calls the EventList.filteredList method. In the EventList.filteredList method, the system iterates through the list of Events in `eventArrayList`. The EventDate object stored in the Event object is passed into the Predicate.check method. If the EventDate object stored in the Event object is a date before the reference date, the Predicate.check method returns true and the Event object is added to the output. After all the elements in EventList.filteredList are parsed and the EventList.filteredList method terminates, EventList.getReminder method returns a string containing the current date and time, the date and time at 2359 the input number of days after the current date, and the filtered list of events. This string is printed to stdout in the Ui.printReminders function.

3.10.2. Sequence diagram

The following sequence diagram shows how the reminder functionality works.



3.10.3. Why it is implemented this way

The reminder function is split the various components into different methods for easier testing. In this case, Ui.printReminders prints the output to the user interface, EventList.getReminder is responsible for compiling the output whereas EventList.filteredList obtains the filtered list of events from `eventArrayList`.

This implementation also implements scalability as the Predicate class and the EventList.filteredList method can be reused for other functions.

3.11. Instruments

3.11.1. How it is implemented

The mistermusik.commons.Instruments.InstrumentList class stores an array list of instruments. Instruments are objects of class mistermusic.commons.Instruments.Instrument. Each instance of the mistermusic.commons.Instruments.Instrument class also has an array list of services. Services are objects of class mistermusic.commons.Instruments.Services.

3.11.2. Adding an instrument

Step 1: The user will type in the command `instrument add /<instrument name>`. The addInstrument() method from the InstrumentList class is called.

Step 2: The addInstrument() method creates a new instance of Instrument with the given input name.

Step 3: The getInstrumentIndexAndName() method, which returns a string containing the index of the instrument and the name, is called.

Step 4: The instrumentAdded() method from the Ui class is then called, which prints a message stating that the instrument has been added.

3.11.3. Viewing the list of instruments stored in the system

Step 1: The user will type in the command `instrument view instruments`. The getInstruments() method from the InstrumentList class is called.

Step 2: The getInstruments() method returns a string containing a list of instruments stored in the InstrumentList instance used for the program.

Step 3: The printInstruments() method from the Ui class is then called, which prints out the list of instruments stored in the InstrumentList instance used for the program.

3.11.4. Servicing an instrument

Step 1: The user will type in the command `instrument service <instrument index> /<brief description of servicing> /<date>`. The service() method from the InstrumentList instance is called.

Step 2: The service() method of the InstrumentList instance calls the addService() method of the referenced Instrument stored in the InstrumentList instance based on the input index.

Step 3: The addService() method adds the relevant servicing information to the Instrument instance by creating a new ServiceInfo instance and adding it to the array list of ServiceInfo stored in the Instrument instance, and returns the number of ServiceInfo instances stored in the array list.

Step 4: The getInstrumentIndexAndName() method of the InstrumentList instance, which returns a string containing the index of the instrument and the name, is called.

Step 5: The getIndexAndService() method of the InstrumentList instance is called, which in turn calls the getIndexAndService() method of the relevant Instrument instance stored in the InstrumentList instance used for the program. Both methods return the index of the required servicing and the brief description of the servicing. The getIndexAndService() method of the Instrument instance calls the getServiceInfo() method on the relevant service to obtain the brief description of the servicing.

Step 6: The serviceAdded() method of the Ui class, which prints a message notifying the user that the servicing information has been added, is then called.

3.11.5. Viewing the list of servicing done for a particular instrument

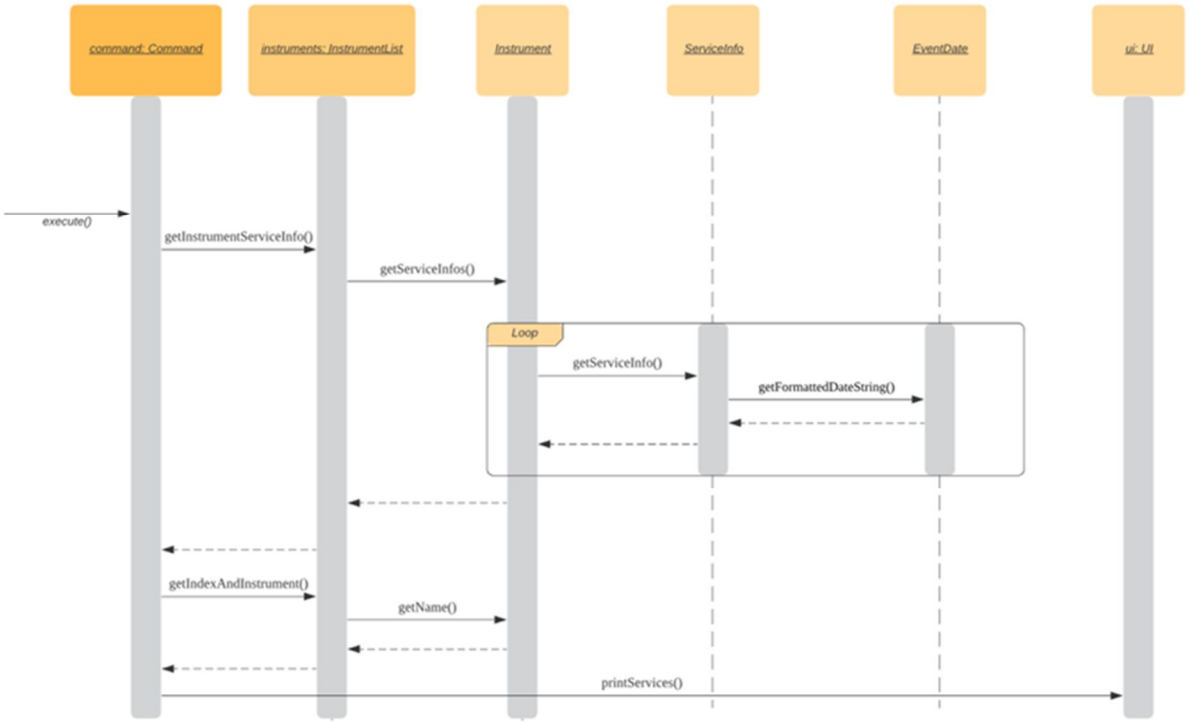
Step 1: The user will type in the command `instrument view services <instrument index>`. The getInstrumentServiceInfo() method from the InstrumentList class is called, which in turn calls the getServiceInfos() method of the relevant Instrument instance stored in the array list of Instruments in the InstrumentList instance. Both methods return the list of servicing done to the relevant instrument. The getServiceInfos() function iterates through the stored list of ServiceInfo instances and calls getServiceInfo() on all the ServieInfo instances to obtain the descriptions of the servicings.

Step 2: The getInstrumentIndexAndName() method of the InstrumentList instance, which returns a string containing the index of the instrument and the name, is called.

Step 3: The printServices() method from the Ui class is then called, which prints out the list of servicing stored in a particular Instrument instance.

3.11.6. Sequence diagram

The following sequence diagram shows how the view services functionality works.



3.11.7. Why it is implemented this way

The implementation makes use of three classes, which are InstrumentList, Instrument and ServiceInfo. This allows for easier testing as each of the classes represent a different component of the functionality. The InstrumentList instance stores an array list of Instrument instances, an Instrument instance stores an array list of ServiceInfo instances.