Yen Pei Chih – Project Portfolio

PROJECT: MisterMusik

Overview

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 is written in java. My role was to design and implement the features that allow our users to set goals for their educational tasks such as practice sessions or lessons. Below shows the details for my design and implementation, as well as my contributions to the user guide and developer guide.

Summary of contributions

Main enhancement added: A management system to set and manipulate goals for specific events.

- What it does: Allows the user to add, edit or delete a goal to a specific event. Additionally, a specified goal can be set as achieved. The user will also be able to view all the goals for an event in the form of a list.
- Justification: We would like our target users to be able to keep track of their desired outcome of each event, particularly for lessons and practices. This way the users can keep track of their progress much more efficiently and effectively as their progress and schedule are both stored on the same platform.
- Highlights: This enhancement works with the function to track past events. When tracking through past events, the system will also check for any goals that have not been achieved for those past events and remind the user.

Other enhancements: The ability to track past events.

- What it does: When the user inputs the command to view their event list, the system will check which events are already past the current date. Subsequently the list displayed will not show the past events.

Code contributions: Please go to this link to view the codes for my implementation.

Other contributions:

- Project management
    - Co-managed the release of v1.3 with teammates.
- Enhancement to non-functional features
    - Coded the logo for our product to be displayed every time it is launched. (Pull request #132)
- Code quality management
    - Refactored essential parts of the code to make it more streamlined and readable. (Pull requests #29, #34, #39, #70)

Contributions to the User Guide

The following is an excerpt from our user guide that shows my enhancement that has been added to our product, namely the goals list and past event management.

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.

Contributions to the Developer Guide

The follow is an excerpt from our developer guide showing my contributions to the goal list management features.

3.8. Goals List

3.8.1. How is it implemented

The goals list is an array list type that stores a list of goals to be achieved by the user for each individual event, particularly for Lesson and Practice type events. When the user first creates the event, the event is created with an empty goal list. Only when the user types in "goal add <event ID>" then the goal list will be updated with the particular goal. The user can then manipulate the goal list by using "goal edit", "goal delete" or "goal view" commands.

Adding a goal

Step 1: When the command "goal add <event ID>/<goal description>" is entered, the goalsManagement() method will be called.

Step 2: The command description will be split up into separate strings. The string for event ID will be parsed into an integer type.

Step 3: The method will check the string for the goal command description. In this case it will be "add" and execute the code for this case.

Step 4: A new Goal class instance will be created with the goal description string. Its achieved status set as false.

Step 5: The event corresponding to the ID entered along with its method addGoal() will be called to add the goal instance into the goal list.

Step 6: The goalAdded() method of the UI class will be called to reflect the change to the user.

Editing a goal

Step 1: The user will enter the command "goal edit <event ID> <goal ID>/<new goal description>". The goalsManagement() method is called.

Step 2: The command description will be split up into separate strings. The strings for event ID and goal ID will be parsed into an integer type.

Step 3: The method will check the string for the goal command description. In this case it will be "edit" and execute the code for this case.

Step 4: A new Goal class instance will be created for the new goal description.

Step 5: The method editGoalList() of the event corresponding to the input ID will be called. The method will set the goal indicated as the new Goal instance.

Step 6: The goalUpdated() method of the UI class will be called to reflect the change to the user.

Deleting a goal

Step 1: When the user enters the command "goal delete <event ID> <goal ID>", the goalsManagement() method will be called.

Step 2: The command description will be split up into separate strings. The strings for event ID and goal ID will be parsed into an integer type.

Step 3: The method will check the string for the goal command description. In this case it will be "delete" and execute the code for this case.

Step 4: The corresponding event will have its removeGoal() method called which removes the indicated goal from the list.

Step 5: The goalDeleted() method of the UI class will be called to reflect the change to the user.

Setting the goal as achieved

Step 1: When the command "goal achieved <event ID> <goal ID>" is given, the goalsManagement() method will be called.

Step 2: The command description will be split up into separate strings. The strings for event ID and goal ID will be parsed into an integer type.

Step 3: The method will check the string for the goal command description. In this case it will be "achieved" and execute the code for this case.

Step 4: The method updateGoalAchieved() for the corresponding event will be called.

Step 5: The goals list within the event is called with the indicated goal.

Step 6: The indicated goal will then call its setAchieved() method that assigns the boolean isAchieved attribute of that particular goal to "true".

Step 7: The goalSetAsAchieved() method of the UI class will be called to reflect the change to the user.

Viewing the goal list

Step 1: The user will enter the command "goal view <event ID>". The goalsManagement() method will be called.

Step 2: Command description will be split and the string for event ID will be parsed into an integer type.

Step 3: The printEventGoals() method will be called to check the contents of the goal list for the indicated event.

Step 4: If the goals list is not empty, it will print out the contents of the list using a for loop. Otherwise it will print a message to the user to reflect that the goal list is empty.

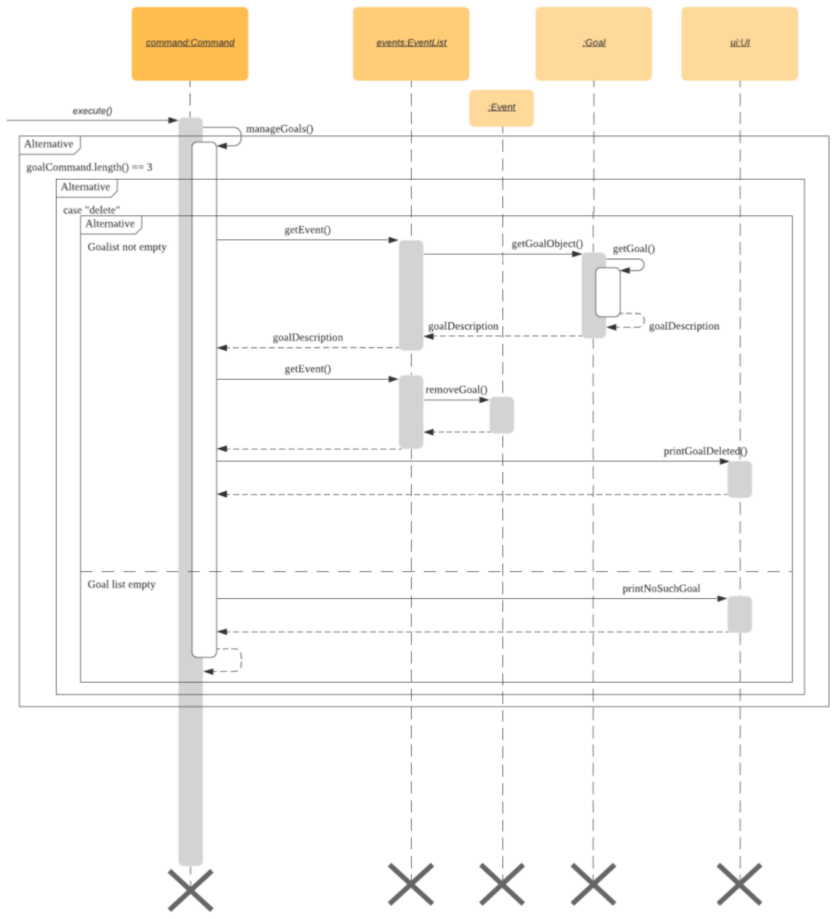
3.8.2. Why is it implemented this way

The goal managing function is implemented as a separate list within an event in order to utilise the indexing of the list elements. This way, a particular goal for a particular event can be easily accessed and manipulated via the input of an integer.

3.8.3. Alternatives considered

An alternate method considered was to implement the goals list as a separate class of its own. Each goal within this list will then be mapped to their corresponding events. This implementation method would cause difficulties on the users' part in identifying the ID of a particular goal and would generally makes the goal list less organised.

3.8.4. Sequence Diagram



3.9. Past event management

This functionality basically tracks which tasks in the list have already passed, and subsequently only displays future tasks when the user uses the "list" command to view the list. This function is linked with the goals management function as it also detects unachieved goals in events that are already over.

3.9.1. How is it implemented

The EventList class contains an integer attribute called currentDateIndex, which will be initialised to 0. The system will then compare the current date with the start dates of each of the events in the event list. Once an event that is in the future has been reached, the index of the event will be assigned to the currentDateIndex. After this has been done, the "list" command will only list out events starting from that particular index.

Step 1: The integer attribute currentDateIndex in the EventList class instance will be initialised to 0 and the Date class currentDate attribute will be initialised to the system date and time of that instance

Step 2: Whenever the user inputs the "list" command, the listOfEvents_String() method is called which will in turn call the findNextEventAndSetBoolean() method which takes in the Date class that has been initialised in step 1.

Step 3: In the findNextEventAndSetBoolean() method, the start dates of each event are compared to the current date using Date class’s compareTo() method using a for loop.

Step 4: The moment the loop reaches an event that has not happened yet i.e. its start date is later than the current date, the index of the event in the list will be assigned to the currentDateIndex and the loop terminates.

Step 5: Back in the listOfEvents_String() method, the list of events starting from the one with the same index integer as currentDateIndex will be returned to the UI class to be printed out.

3.9.2. Why is it implemented this way

This function is implemented this way to maximise the usefulness of the indexing nature of ArrayList. Even if an event has passed and the list no longer shows it, the indexes of all the events are kept constant. This is crucial as the commands for most of our functions require the user to indicate the index of an event they want to manipulate, and keeping them constant makes it very easy for the user to input commands without having to take note of the change in the event indexes after making any changes to the list.

3.9.3. Alternatives considered

We have considered the possibility of deleting the past events that are over. However, as mentioned above we require the indexing of the events to stay constant and by deleting the past events, the event indexes will change as well.

3.9.4. Integration with goal list management

Implementation

The management of goals list can be integrated with the ability to track past events. This is done through a boolean attribute gotPastUnachieved in the EventList class. If true, there are unachieved goals for past events and vice versa.

Step 1: User enters the "list" command. The listOfEvents_String() method is called which then calls the findNextEventAndSetBoolean() method.

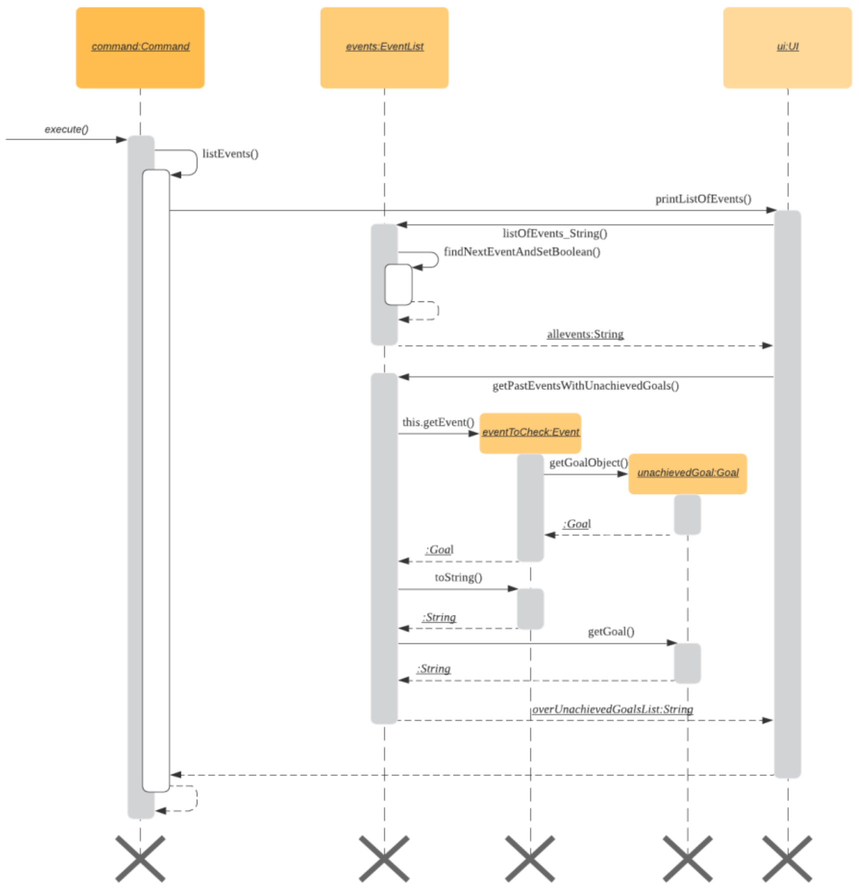
Step 1: Within the findNextEventAndSetBoolean() method, the attribute currentDateIndex is checked to see if it is more than 0. If it is, that means there are past events.

Step 2: If there are past events, the method will enter a nest of two for loops. The outer loop runs through all the past events, while the inner loop iterates through the goal list for each individual past event.

Step 3: Each time an inner loop is iterated through, the getBooleanStatus() method for the particular event will be called. If returns true, gotPastUnachieved will be assigned as true and the method will break out of the loop.

Step 4: After the event list has been printed out, the method getPastEventsWithUnachievedGoals() is called. This method utilises two nested for loops to identify the unachieved goals among all the past events and passes both the past events along with all their unachieved goals to the UI class to be printed out below the event list.

3.9.5. Sequence Diagram


command:Command
events:EventList
ui:UI
execute()
listEvents()
printListOfEvents()
listOfEvents_String()
findNextEventAndSetBoolean()
allevents:String
getPastEventsWithUnachievedGoals()
this.getEvent()
eventToCheck:Event
getGoalObject()
unachievedGoal:Goal
:Goal
:Goal
toString()
:String
getGoal()
:String
overUnachievedGoalsList:String