Group Assignment Final Report

This report contains all paperwork, which includes the designs, introduction, evaluation, testing, requirements task allocation and the user manual, as required for this project.

MEETING MANAGER

GROUP 22

Afzal Miah

University of Dundee

170010026

[amiah@dundee.ac.uk](mailto:amiah@dundee.ac.uk)

Ryan Anderson

University of Dundee

170013613

[rdanderson@dundee.ac.uk](mailto:rdanderson@dundee.ac.uk)

Isaac Lowry

University of Dundee

170025555

[iawlowry@dundee.ac.uk](mailto:iawlowry@dundee.ac.uk)

Andrew Hart

University of Dundee

160013951

[ahart@dundee.ac.uk](mailto:ahart@dundee.ac.uk)

# Introduction

This project requires the development of a meeting manager. The meeting manager will need to support multiple employees who will each have electronic diaries which can record meetings they will have. This includes a range of topics such as, presentation, client and project meetings as well as any other relevant meetings. Each employee will be supplied a generated unique employee ID. The meeting manager will be able to search through the diaries of employees to isolate any information needed. The meeting manager will need to be able to output the information in a formatted easy to read manner. The manager should allow for some key options/operations to be carried out by the user; Search as mentioned above, a Display option to output as necessary, Edit options including add, delete, edit(change details of a meeting), and undo.

In terms of research, some things as it pertained to programming needed to be looked up. The group decided that rather than program the data structure classes ourselves, rather it would be more efficient to use the built in collection classes so research was done to see how the collection classes work and how to integrate them so that the implementation would work seamlessly. In this case the TreeSet collection class was researched as well as the LocalDateTime collection class.

# Requirements

R1. The program shall create a separate electronic diary for each employee in the company to record their meetings

R2. The meeting manager shall allow for the recording of the date of an employee’s meeting.

R3. The meeting manager shall allow for the recording of the start time of an employee’s meeting.

R4. The meeting manager shall allow for the recording of the end time of an employee’s meeting.

R5. The meeting manager shall allow for the recording of the description of an employee’s meeting.

R6. The meeting manager shall allow for the recording of the location of an employee’s meeting.

R7.The program shall provide the option to add entries from the electronic diaries for each employee.

R8.The program shall provide the option to delete entries from the electronic diaries for each employee.

R9.The program shall provide the option to edit entries from the electronic diaries for each employee.

R10. It shall also provide an ‘undo’ feature, which will allow for the user to undo the last ‘add’, ‘delete’ or ‘edit’ of a diary entry.

R11. An option to display a list of diary entries for each employee shall be implemented.

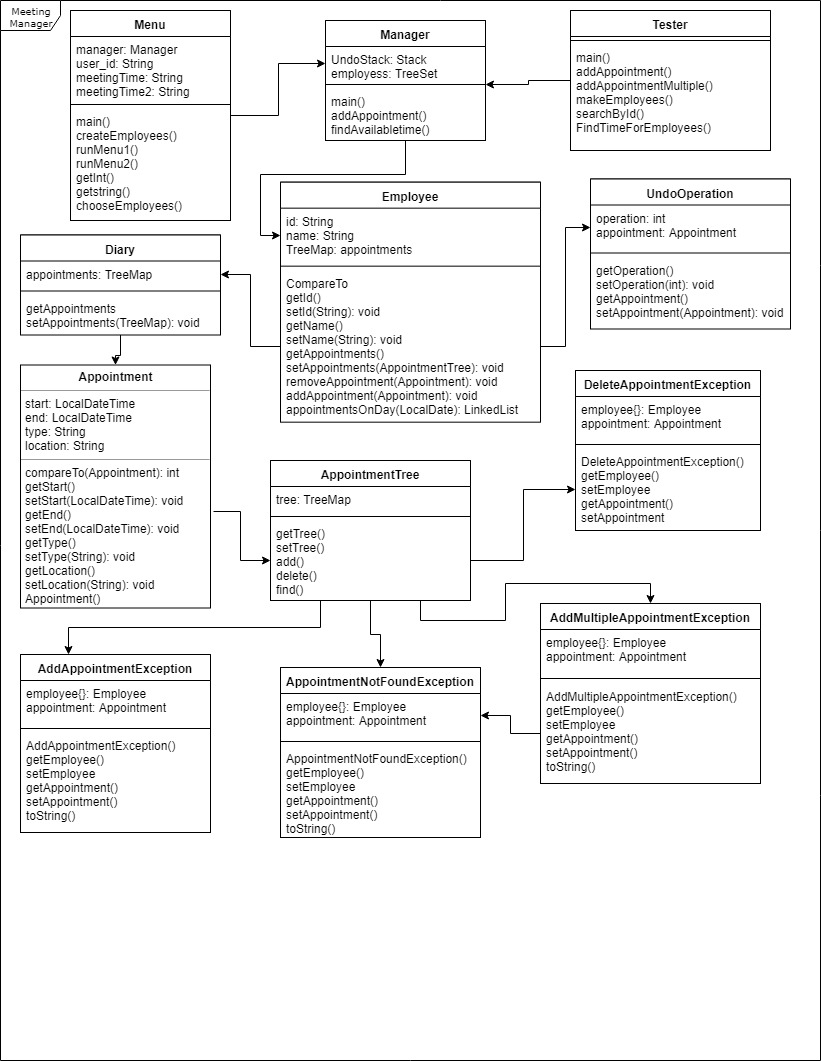
R12 Another feature the program shall have is to provide an option to save diary entries to file.

R13. Another feature the program shall have is to provide an option restore diary entries from a file.

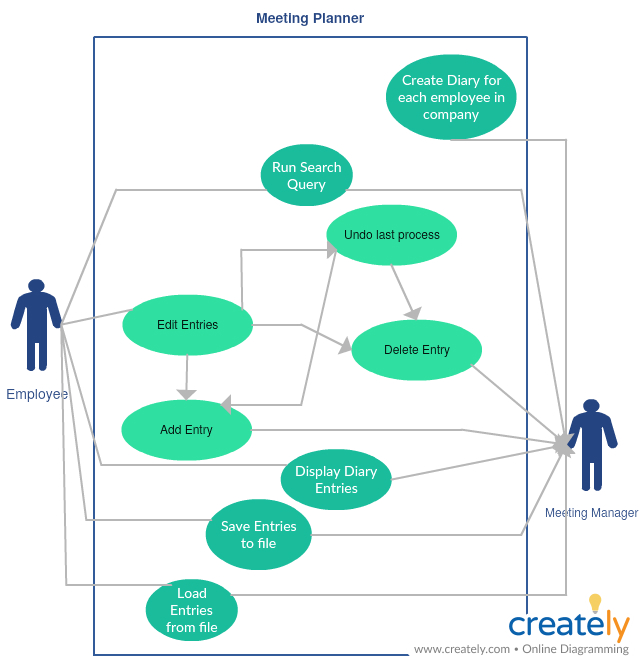
R14. There shall be a search feature implemented which will: accept a list of employee names, a search start date, a search end date, and identify potential meeting dates / times within this range for the employees specified.

# Design

## Class Diagram

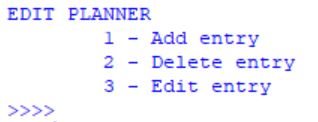
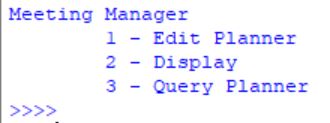


## Use Case Diagram

****

## Interface Design

The UI of the program is designed to work as a command line based UI



The UI will follow the two examples above for each separate menu needed in the program. The program will output the menu and its options with each option having its own corresponding input. The arrows indicate asking for an input and the user is expected to input the corresponding number of the choice they would like to run from the menu they are on

## Pseudocode

searchEmployees(meeting)

Loop all employees

Check if current employee has meeting in diary

Add current employee to list

Return list of employees at meeting

DEF add(appointment) THROWS EXCEPTION

SET aptDate TO appointment.getStart().toLocalDate

IF tree.get(aptDate IS null:

tree.put(aptdate, <Appointment>())

tree.get(aptDate).add(appointment)

END IF

ELSE:

SET treeNode TO tree,get(aptDate)

SET listIter TO treeNode.iterator()

SET found TO false

SET i TO 0

WHILE listIter HAS NEXT VALUE AND IS NOT found:

SET CURRENT TO NEXT VALUE

IF current.getStart() IS appointment.getStart():

EXECUTE AddAppointmentException

END IF

ELSE IF current.getStart() IS BEFORE appointment.getEnd():

SET FOUND TO true

END IF

ELSE

i++

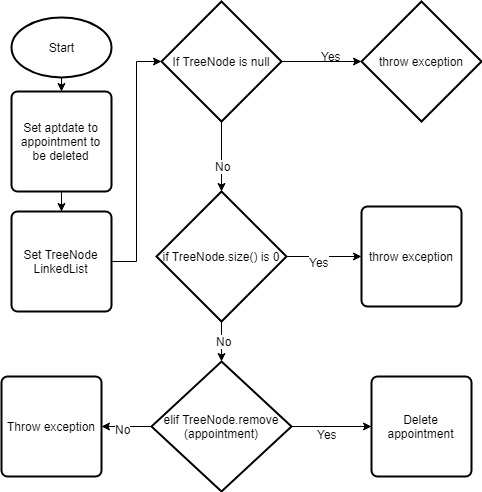
END ELSE

END WHILE

TreeNode.add(i,appointment)

END ELSE

# Flowchart



# Task Allocation

Ryan Anderson – Manager, Tester

Isaac Lowry – Appointment, Tester

Afzal Miah – Diary, Tester

Andrew Hart – Menu, Employee, Tester

# Testing

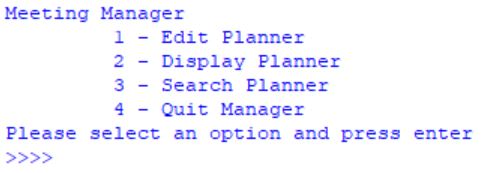
## Test Data Plan

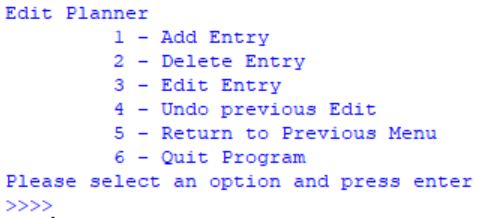
|  |  |  |  |
| --- | --- | --- | --- |
| **Test Data Set** | **Reason** | **Expected Output** | **Actual Output** |
| Menu1 = 1, menu 2 = 1, start = 2018-08-12T13:00:00, end = 2018-08-12T14:00:00, description = deposition, id1 = 123, id2 = 345, menu2 = 6, menu1 = 4 | Extreme | Menu 1 and ask for input, menu 2 and ask for input, ask user for start, end, type, employee 1 id, employee 2 id, menu 2 and ask for input, menu 1 and ask for input. | As expected |
| menu 1 = 5,1, menu 2 = 7,6 | exceptional | Menu 1 and ask for input. Error message and ask for re-input. Menu 2 and ask for input. Error message and ask for re-input. | As expected |
| menu 1 = 1, menu 2 = 2, 2018-08-12T13:00:00, menu 2 = 6, menu 2 = 4 | normal | Menu 1 and ask for input. Menu 2 and ask for input. Ask for meeting to be deleted. Menu 2 and ask for input. Menu 1 and ask for input. | As expected |
| Menu1 = 3, users = 123,345, start = 2018-08-10, end = 2018-08-15 | normal | Menu 1 and ask for input. Ask for users to be searched, start date and end date for search. Outputs all meetings which fit search criteria. | As expected |
| Menu1 = 1, menu 2 = 1, start = 2018-08-16T13:00:00, end = 2018-08-16T14:00:00, description = review, id1 = 123, id2 = 345, menu2 = 4, | normal | Menu 1 and ask for input, menu 2 and ask for input, ask user for start, end, type, employee 1 id, employee 2 id, menu 2 and ask for input, Program will undo previous edit which was an add entry. | As Expected |
| Menu 1 = 2, employee = 123, | normal | Menu 1 and ask for input, ask for employee to be displayed, displays the chosen employees diary. | As expected |

# User Manual

The program is a command line based program meaning it requires users to read the information from the screen and physically type in the required inputs.

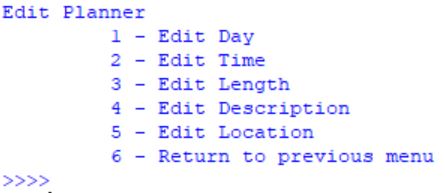
The program works by using different menus for different operations and the user navigates through these menus to use the program.

When the program is started the following menu is outputted to the user and is given the 4 options shown; Edit planner, display planner, search planner, quit planner. The user is expected to input their desired option by typing in the number corresponding to an option. E.g for Edit planner the user would type in “1” then press enter. Depending on the input another menu is outputted.

If the user decides the “Edit Planner”, another sub-menu is outputted with all the available editing options. This menu works with the same principles as the main menu where users enter the corresponding number for that option and the relevant information is outputted depending on the input.

If user chooses Add Entry Option, the user is expected to enter 5 different inputs. The user must enter when the meeting begins, ends, its description, user id for both the employees in the meeting.

If delete entry is chosen, all that user enters is the beginning time of the meeting to be deleted.

For the edit entry option, at first the user is asked to input the meeting which they would like to edit. Another menu is then outputted with all the available editing options, working the same as previous menus. Depending on the user input, the corresponding option is executed asking the user for what changes they would like to make to that piece of data.

Undo previous option, return, quit options don’t output or ask for inputs and simply execute the function.

From the main menu, if display planner is chosen, the user is asked to enter the user of whose diary they wish to be displayed.

Search planner option if chosen requires 3 inputs from the user to work. First it asks for what users are being searched, the start date of the search and the end date of the search after which the search will display the meetings which fit the inputted criteria from the user. Option 4 if chosen, terminates the program.

# Evaluation

At the start of this process we were all confident in our ability to complete the assignment to it's fullest. We began our plan and set about deciding on what data structures to use and how to go about implementing them. We initially had a simpler design, with fewer classes but with each class doing more work. However, as time passed we found ourselves struggling. This is in part due to a certain lack of foresight in planning. We hadn’t thought about how our individual classes would interact and as a result certain methods didn’t work very well with others. Luckily, we noticed this with enough time to regroup and focus more on a unified vision rather than working on our aspects separately. It also helped at this stage in the process that Ryan took charge more, and it was with his guidance that the rest of us could work our classes together more effectively. This new plan for the program involved merging certain classes together and creating new classes to handle different processes not existent in the original plan. We also moved to using java collection classes more for more robust code. Ultimately we decided near the end of the process to focus on the core requirements and leave out the additional ones from our original plan to ensure the base code was functioning at it's peak. Apart from the change halfway through the project, the implementation went smoothly, largely in part due to the team being able to work through problems together, ensuring that they only halted progress momentarily rather than for a larger period of time.