# **Employee Diary**

**Student Name:** Ryan Crampton

## **Statement of requirements**

# Brief

The project is to develop a Meeting Manager. So, imagine there are employees working for a company. Each employee will have their own electronic diary which allows them to record details of any meetings that they have, e.g. project meetings, presentations, client meetings, etc. Various features will be required including the capability to search across the diaries of several employees to identify a set of possible dates

And times that the different employees could get together with each other for a meeting and then display those dates and times..

Assumptions

The user assumptions include:

The user has used a computer before and can navigate buttons

The users will be of varied experience with databases and will be able to work the database in basic, but may need help/prompts

There is more than one employee in the company

Inputs

The inputs to the system will be though a keyboard and mouse. The mouse will be used to select the items while the keyboard is used to input the game data

**Use 1: Menu**

The Menu will have options that do the following:

* To add a meeting
* To check an employee and their current schedule
* Delete a meeting
* Undo a meeting/redo a meeting
* Search the database for Employees and mass add a meeting, checking if they have the space
* Exit the software

**Use 2: Save**

There will be the option to export the database to a save file, and call upon it when needed

**Use 3: Exit**

The third use is to exit the software

Outputs

The outputs will provide the results as follows:

**Use 1: Menu**

When an option is chosen by the user, the system should load the appropriate screen.

**Use 2: Save**

During the save function, this should either send this to an external document, specified by the user, or find the specified document and load that data, showing it to the user

**Use 3: Exit System**

The system should display a message letting them know the system is exiting

# Requirements

Functional Requirements

* **R1** the system shall have a menu allowing the user to select the appropriate option.
* **R2** The system shall have an option to add meetings to each employee
* **R3** The system shall have an option to delete a meeting for each employee
* **R4** The system shall have an option to check how long a meeting lasts
* **R5** The system shall have an option to search the employees and mass add a meeting
* **R6** If there is a meeting already present, the system won’t allow the user to add it
* **R7** The menu must load between options
* **R8** a record of employees must be shown upon request, in order of ID
* **R9** the system shall have an option which allows the user to save the file
* **R10** the system shall have a ‘Load File’ option to load a previously saved diary, allowing the users to find information from outside of the system
* **R11** the system may have an option that allows the search to add to those who can make the meeting
* **R12** The system shall display the meeting that the current meeting will clash with, and the employee
* **R13** The system will display the current meetings, as well as save the previously set one.

Non-functional Requirements

* **R1** the system should have a clear user interface, allowing users to access what they need to
* **R2** the system should run on Java
* **R3** the system should look user-friendly
* **R4** the system shall be playable on a keyboard
* **R5** the system may allow the users custom inputs
* **R6** the system shall produce an output when the user asks for it

# User Interface

The interface will be text based and start off with the user entering the menu, this will display the options that exist within the menu and explain what each option does, this will then prompt the user to input a value based on what they would like to do.

If the user has selected add, delete or undo, they will be asked what they would like to add, delete or undo, and to whom, identified by the ID number (This will be specified to the user).They will be able to input information when prompted, and this will feedback information as they do this. If it finds issue with what they input it will let them know.

If the user selects search, it will act similar, asking them to input what they would like hunted for, giving feedback as it goes, this is the same with the time length, as this will be determined by the user too.

If the user selects to view the diary or employees, the system will display these in order of ID, then offer to display the diaries.

If the user selects exit, the programme will exit.

# Use Cases

|  |  |  |
| --- | --- | --- |
| **Diary Entry** | | **Alternative** |
| 1. User: | Opens program |  |
| 1. System: | Loads main menu |  |
| 1. System: | Prompts user to enter Employee ID |  |
| 1. User: | Inputs Employee ID | E1 |
| 1. System | Checks if ID is valid | A1 |
| 1. System: | Loads employee diary |  |
| 1. System: | Displays employee diary and menu options |  |
| 1. User: | Selects “Add to diary” | A2/A3/A4/A5/ A6/A7/A8/E1 |
| 1. System: | Asks user if they want to add this meeting to other people’s diaries | A9 |
| 1. System: | Asks user to fill in information (Time, Location etc.) |  |
| 1. User: | Inputs Details | E1 |
| 1. System: | Checks if the time doesn’t clash with other meetings | A10 |
| 1. System: | Adds information |  |
| 1. System: | Message to tell the user information has been added |  |
| 1. System: | Returns to step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A1. Employee ID does not exist or was entered incorrectly** | | **Alternative** |
| 1. System: | Displays error message and asks user to try again |  |
| 1. System: | Returns to ‘Diary Entry’ step 3 |  |

|  |  |  |
| --- | --- | --- |
| **A2. User selects Delete Meeting** | | **Alternative** |
| 1. User: | Selects “Delete meeting” |  |
| 1. System: | Displays diary |  |
| 1. System | Prompts user to input start date and time of meeting they want to delete |  |
| 1. System: | Checks if there is a meeting at that time and day | A2.1/E1 |
| 1. System: | Deletes selected meeting |  |
| 1. System | Returns to ‘Diary Entry’ step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A2.1. No meeting is scheduled for that time and date** | | **Alternative** |
| 1. System: | Displays error message and asks user to try again |  |
| 1. System: | Returns to A2 step 2 |  |

|  |  |  |
| --- | --- | --- |
| **A3. User selects Edit Meeting** | | **Alternative** |
| 1. User: | Selects “Edit meeting” |  |
| 1. System: | Displays diary |  |
| 1. System | Prompts user to input start date and time of meeting they want to edit |  |
| 1. User: | Inputs start date and time of meeting they want to edit | E1 |
| 1. System: | Checks if there is a meeting at that time and day | A3.1 |
| 1. System: | Prompts user to input what part of the meeting they want to edit |  |
| 1. User: | Inputs what they want to edit |  |
| 1. System: | Checks if what the user input was valid | E1 |
| 1. System: | Prompts user to input what they want changed |  |
| 1. User: | Inputs changes |  |
| 1. System: | Checks if what the user input was valid | E1 |
| 1. System: | Makes changes |  |
| 1. System: | Prompts user to edit another part of the meeting or return to diary | A3.2 |
| 1. User: | Selects Return to diary |  |
| 1. System: | Returns to ‘Diary Entry’ step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A3.1. No meeting is scheduled for that time and date** | | **Alternative** |
| 1. System: | Displays error message and asks user to try again |  |
| 1. System: | Returns to A3 step 2 |  |

|  |  |  |
| --- | --- | --- |
| **A3.2. User chooses to edit another part of the meeting** | | **Alternative** |
| 1. System: | Returns to A3 step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A4. User undoes a diary entry** | | **Alternative** |
| 1. User: | Selects “Undo” |  |
| 1. System: | Checks if undo is possible | A4.1 |
| 1. System | Undoes most recent thing the user has done to the diary |  |
| 1. System: | Returns to ‘Diary Entry’ step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A4.1. It is not possible to undo an action** | | **Alternative** |
| 1. System: | Displays message saying the undo was not done |  |
| 1. System: | Returns to ‘Diary Entry’ step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A5. User saves diary to a file** | | **Alternative** |
| 1. User: | Selects “Save Diary” |  |
| 1. System: | Checks if diary isn’t empty | A5.1 |
| 1. System | Saves diary to an external file | A5.2 |
| 1. System: | Returns to ‘Diary Entry’ step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A5.1. There are no meetings in the diary** | | **Alternative** |
| 1. System: | Displays message saying there are no meetings in the diary |  |
| 1. System: | Returns to ‘Diary Entry’ step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A5.2. Could not save to file** | | **Alternative** |
| 1. System: | Displays message saying that the diary could not be saved |  |
| 1. System: | Returns to ‘Diary Entry’ step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A6. User restores diary from a file** | | **Alternative** |
| 1. User: | Selects “Load Diary” |  |
| 1. System | Loads diary from an external file | A6.1 |
| 1. System: | Returns to ‘Diary Entry’ step 6 |  |

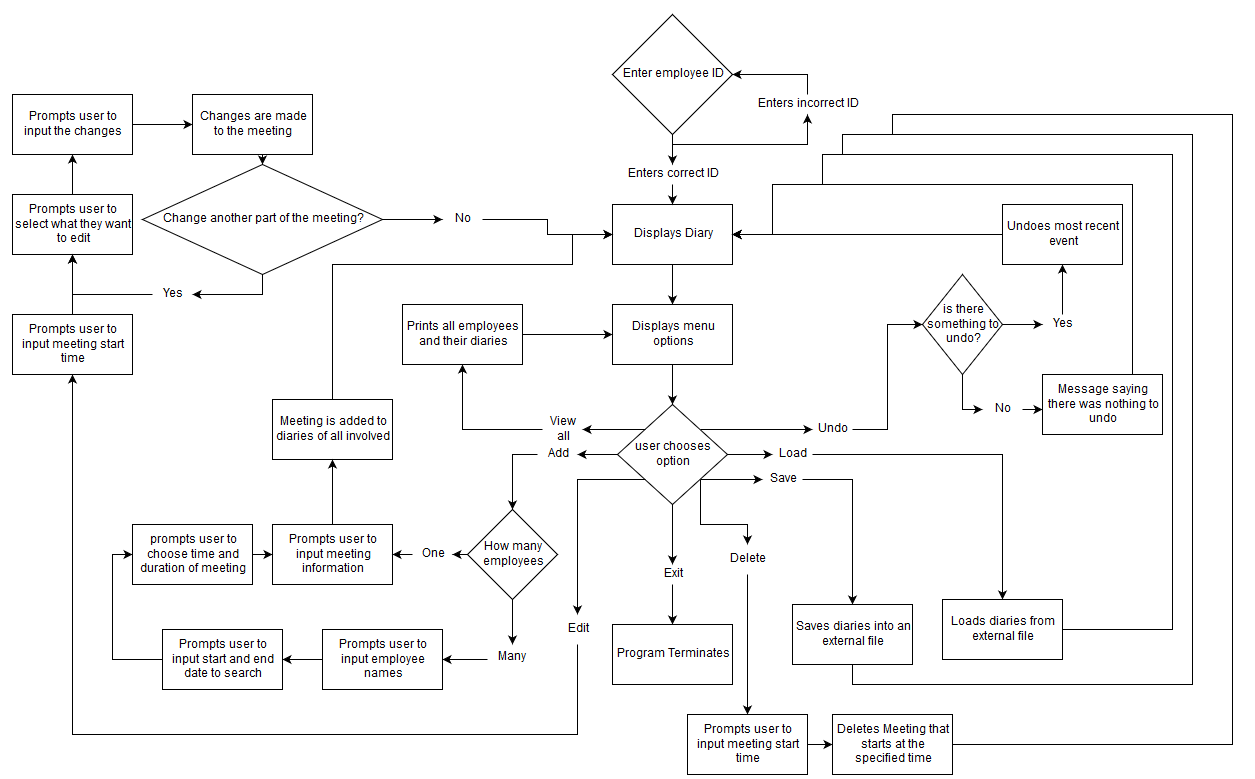
|  |  |  |
| --- | --- | --- |
| **A6.1. Could not load to file** | | **Alternative** |
| 1. System: | Displays message saying that the diary could not be loaded |  |
| 1. System: | Returns to ‘Diary Entry’ step 6 |  |

|  |  |  |
| --- | --- | --- |
| **E1. User incorrectly inputs data i.e. Inputs letters into an int value** | | **Alternative** |
| 1. System: | Displays error message and asks user to try again |  |
| 1. System: | Returns to previous step |  |

|  |  |  |
| --- | --- | --- |
| **A7. User selects “View all diaries”** | | **Alternative** |
| 1. User: | Selects “View all diaries” |  |
| 1. System: | Prints employee names and their diary in a list |  |
| 1. System: | Returns to ‘Diary Entry’ step 6 |  |

|  |  |  |
| --- | --- | --- |
| **A8. User selects “Exit”** | | **Alternative** |
| 1. User: | Selects “Exit” |  |
| 1. System: | Terminates |  |

# Use of Case Diagram:



**Classes:**

##### **Candidate Classes**

|  |  |  |
| --- | --- | --- |
| **Candidate Classes** | **Accept / Reject** | **Reason** |
| **Diary** | **Accept** | **List of Meetings** |
| **Menu** | **Accept** | **Starting class** |
| **Employee** | **Accept** | **Tree nodes class** |
| **Meeting** | **Accept** | **Stack Class** |
| **Manager** | **Accept** | **Tree Class/Top Level Coordinator** |
| date | Reject | Part of meeting |
| Time start | Reject | Part of Meeting |
| Location | Reject | Part of meeting |
| Time end | Reject | Part of meeting |
| description | Reject | Part of Meeting |
| Name | Reject | Part of employee |
| ID | Reject | Part of employee |
| Department/Role | Reject | Part of Employee |
| add | Reject | Method |
| delete | Reject | Method |
| search | Reject | Method |
| undo | Reject | Method |
| Display | Reject | Method |
| Edit | Reject | Method |

# 

# Class Descriptions

Including Responsibilities, Fields and Methods

**Class: Menu** - Appears on startup, responsible for allowing the user to select different options

**Objects:** Manager tree = new Manager();. Diary test = new Diary();

**Methods:** Void changeEmployeeDiary(), printAllDiaries(), saveEmployeeDiary(), loadEmployeeDiary(), searchFreeMeetingTimes(), exit();

**Class: Manager** – Part of the tree of employees, holds the info for employees

**Objects:** Employee root;

**Method:** getRoot() ,setRoot(), add(), print(), traverse(), find(), delete(), isTreeEmpty();

**Class: Employee**  - Treenode based class, holds the info form the employees

**Objects:** String Name, int ID, String department, Diary diary

**Method:** accessors & mutators

**Class: Meeting** – The stack of meetings, holds the info for the meetings

**Objects:** LocalDate dateOfMeeting, LocalTime startTime, LocalTime endTime, String description, String location, String description.

**Method:** accessors & mutators

**Class: Diary** – The held info for all employee meetings

**Objects:** Meeting meeting = new Meeting();

**Method:** addMeeting(); checkMeeting(); deleteMeeting(); , save(); ,load();, undo(); edit();

# Class Diagram

Manager

TreeSet<Employee> employeeTree

Void setRootNode(Employee)

Employee getRootNode()

Boolean isEmpty()

Employee findEmployee(Employee, int)

Void inorderTraversal(Employee)

Boolean removeEmployee(int)

Void searchForJointMeetingTimes(LocalDate startDate, LocalDate endDate)

Employee

Diary employeeDiary

Int employeeID

Void setDiary(Diary)

Diary getDiary()

Int getID()

Diary

LinkedList<Meeting> meetings

Stack<Meeting> undoStack

Boolean addEntry(Meeting)

Boolean removeEntry(LocalTime)

Boolean editEntry(LocalTime)

Void undo()

Void saveDiary()

Void loadDiary()

Boolean doMeetingsClash(Meeting)

Meeting

LocalDate dateOfMeeting

LocalTime startTime

LocalTime endTime

String description

Void setDateOfMeeting(LocalDate)

LocalDate getDateOfMeeting()

Void setStartTime(LocalTime)

LocalTime getStartTime()

Void setEndTime(LocalTime)

LocalTime getEndTime()

Void setDescription(String)

String getDescription

Menu

Manager managerTree

Void main(String[])

Display startMenu()

Void changeEmployeeDiary()

Void printAllDiaries()

Void saveEmployeeDiary()

Void loadEmployeeDiary()

Void searchFreeMeetingTimes()

# 

# Pseudocode

**Determining if a meeting clashes with an existing meeting**

1. For each diary entry in employee diary
   1. If existing start or end time is between new start and end time
      1. New meeting is invalid
2. End for

**Adding a meeting**

1. Create new meeting addMeeting
2. Enter user data into addMeeting
3. If addMeeting data is invalid or clashes with another meeting
   1. Return false
4. Create new meeting undoMeeting
5. undoMeeting = addMeeting
6. Set undoMeeting.undoReference to addMeeting
7. Add addMeeting to diary
8. Add undoMeeting to undoStack
9. Return true

**Removing a meeting**

1. Create new meeting removeMeeting
2. Enter user data into removeMeeting
3. If removeMeeting doesn’t exist in diary
   1. Return false
4. Else
   1. removeMeeting = found meeting
5. Create new meeting undoMeeting
6. undoMeeting = removeMeeting
7. Set undoMeeting.undoReference to null
8. Add undoMeeting to undoStack
9. Delete removeMeeting from diary

**Editing a meeting**

1. Create new meeting editMeeting
2. Enter user data into editMeeting
3. oldMeeting.undoReference = editMeeting
4. Add oldMeeting to undoStack
5. Delete oldMeeting from diary
6. Add editMeeting to diary

**Undo diary entry**

Whenever adding, removing, or editing a meeting, store a reference to it in a stack.

When undo function is called –

1. Pop undoMeeting off stack
2. If undoMeeting.undoReference = undoMeeting (Node was added)
   1. Delete undoMeeting
3. Else if undoMeeting.undoReference = null (Node was deleted)
   1. Add undoMeeting
4. Else (Node was edited)
   1. Delete undoMeeting.undoReference
   2. Add undoMeeting

**Save diary to file**

1. Create new Path diaryPath
2. Ask user where to save diary
3. Set diaryPath to user input
4. Create new BufferedWriter bw
5. For all meetings in diary
   1. Write meeting field values to file
   2. Write new line to file
6. Close resources

**Load diary from file**

1. Create new Path diaryPath
2. Ask user where to load diary from
3. Set diaryPath to user input
4. Create new BufferedReader br
5. While br is not empty
   1. Load diary data from file
   2. Create new meeting with data
   3. Add meeting to diary
6. Close resources

**Search valid meeting times for multiple employees**

1. Create new diary validTimes
2. For each employee
   1. For each day between start and end date
      1. Add employee meetings to validTimes
   2. End for
3. End for
4. Determine valid times from any gaps between meetings in validTimes diary
5. Output valid times to user
6. Ask user for meeting time
7. If user input is within valid bounds given
   1. Add meeting to each employees diary
8. Else
   1. Ask user to input meeting time again

**Add employee meetings to validTimes**

1. For each meeting with current date
   1. If meeting clashes with an existing meeting
      1. Set meeting start time to the earlier of the two start times
      2. Set meeting end time to the later of the two end times
      3. Replace existing meeting with new meeting
   2. Else add meeting to validTimes
2. End for

**Tasks Assigned**

Ryan Crampton – Search Feature

Hugh Hegarty – Add, Delete, Edit etc.

Tom Smith – Save/Load