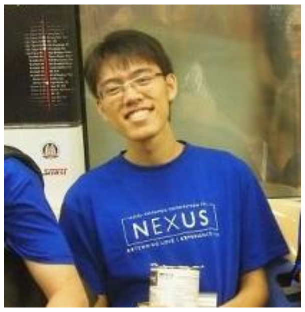
YourDay

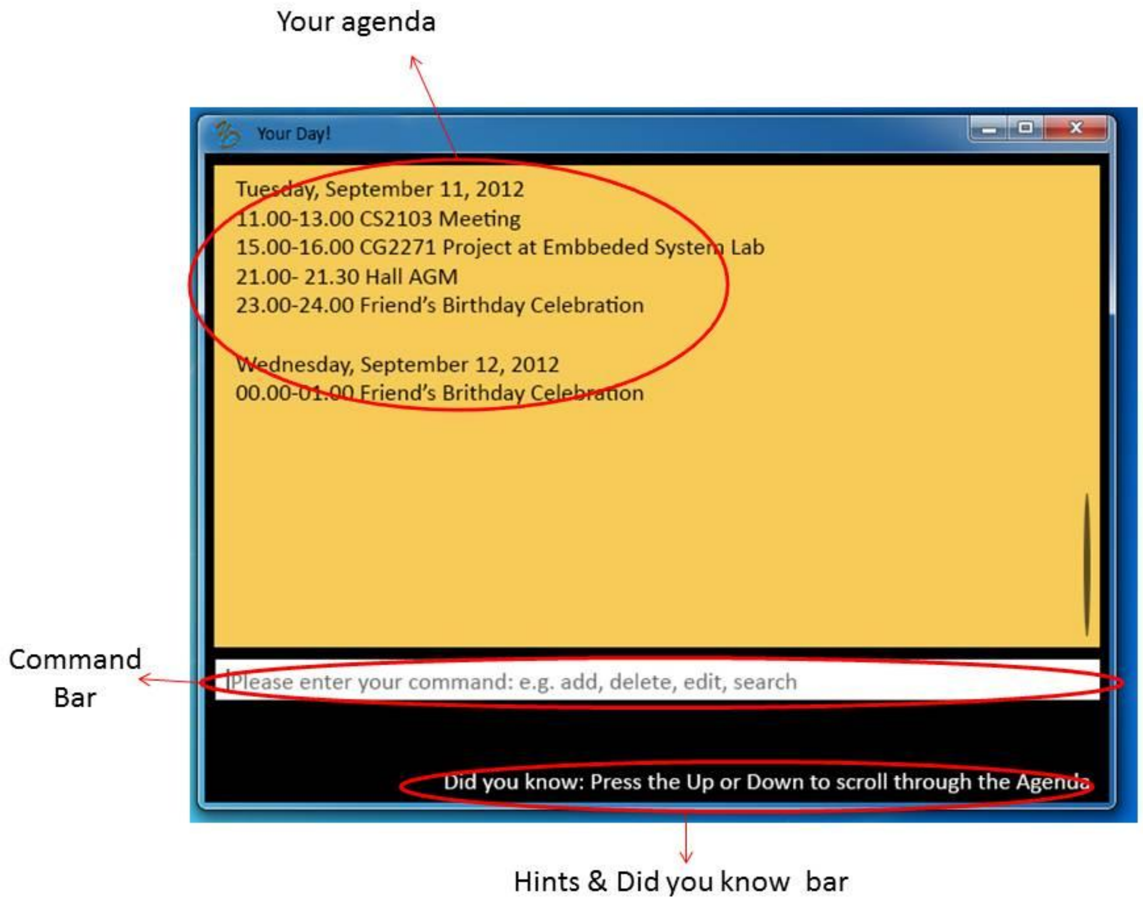
			
Ignatius Damai Team Leader Deadline Watcher	Huang Da Lead Coder	Wu Pei Lead Tester	Nguyen Ngoc Nhu Thao Lead UI

Quick Introduction

Tired of missing your appointments or having difficulties in managing your agenda? YourDay will help you to keep track of your appointments and tasks, as well as to keep them organized. YourDay is a powerful software which accepts semi-natural language commands via keyboard, provides easy-to-use functionalities and utilizes simple user interface.

User Guide

1. **The Main Screen**
   On the home screen, you can give commands such as add, edit, delete, or search agenda entries. The following sections will guide you how to use each of these functions.



Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

Wednesday, September 12, 2012
00.00-01.00 Friend's Brithday Celebration

Please enter your command: e.g. add, delete, edit, search

Did you know: Press the Up or Down to scroll through the Agenda

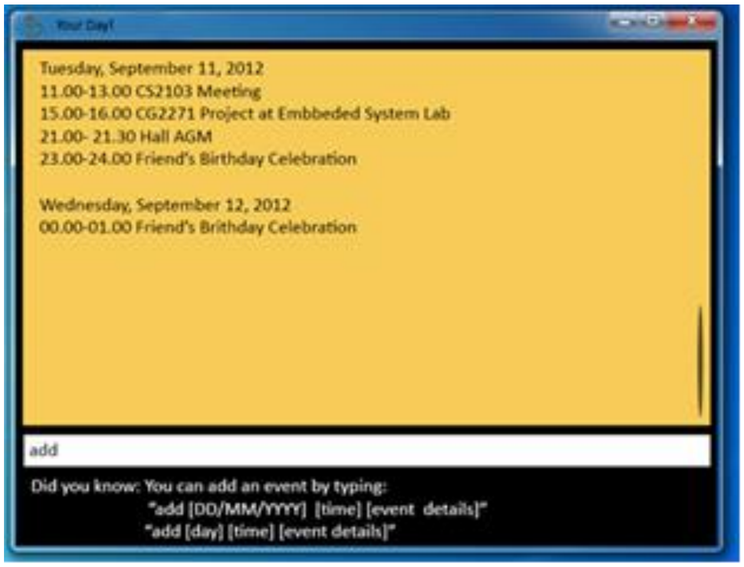
2. **Adding an Entry**
   Our product lets you add new agenda entries easily. There are several flexible formats for task addition:

Command:	**[add command] [DD/MM/YYYY] [Time] [Event Details]**
Possible add commands:	**add, create, +**

You can add more tasks details such as priority or location easily. To add location details of an entry, add "**at**" followed by the location after your event details. To add priority, simply add "**priority**" after your event details. The accepted priorities are:

- 0 : Highest Priority
- 1 : High priority

- 2 : Normal priority. This is the default priority if the user does not give a specific priority
- 3 : Low Priority.



Your Day!

Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

Wednesday, September 12, 2012
00.00-01.00 Friend's Brithday Celebration

add

Did you know: You can add an event by typing:
"add [DD/MM/YYYY] [time] [event details]"
"add [day] [time] [event details]"

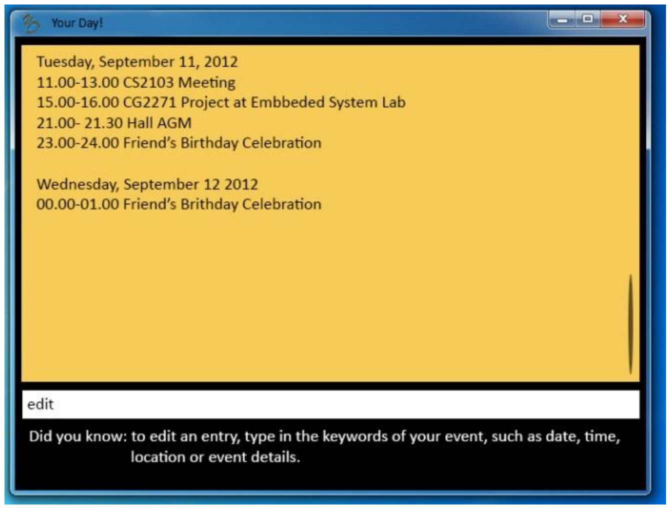
Date format: **DD/MM/YYY**

Time format: **HH:MM-HH:MM**

3. Editing an Entry

Editing an entry has never been easier. To update an entry simply type:

Command:	**[edit commands] [keyword(s)]**
Possible edit commands:	**edit, update, change, modify, !**



Your Day!

Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

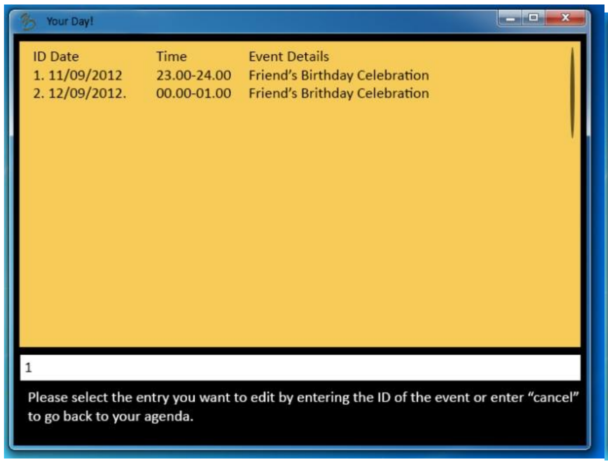
Wednesday, September 12 2012
00.00-01.00 Friend's Brithday Celebration

edit

Did you know: to edit an entry, type in the keywords of your event, such as date, time, location or event details.

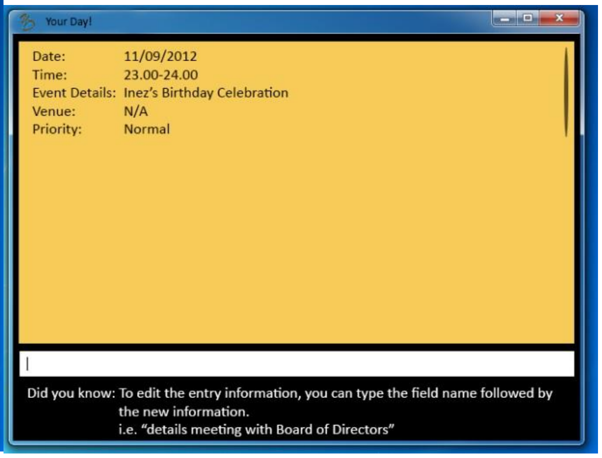
The keywords can be a word or a part of the entry details, date, or time of the entry. After you enter the command, YourDay will give you a list of entries that matches your keywords. After that, you can select which entry you want to edit by entering the event ID shown ur search result.

You will then be shown the entry's details. To edit the entry, type the field name you want to edit (details, time, date, venue, or pri) and type the new value for the respective field.



```
Your Day!
ID Date          Time          Event Details
1. 11/09/2012    23.00-24.00   Friend's Birthday Celebration
2. 12/09/2012.   00.00-01.00   Friend's Brithday Celebration

1
Please select the entry you want to edit by entering the ID of the event or enter "cancel"
to go back to your agenda.
```



```
Your Day!
Date:           11/09/2012
Time:           23.00-24.00
Event Details:  Inez's Birthday Celebration
Venue:          N/A
Priority:       Normal

|
Did you know: To edit the entry information, you can type the field name followed by
              the new information.
              i.e. "details meeting with Board of Directors"
```

4. **Deleting an Entry**

In order to delete an existing entry, you need to enter a command in the following format:

Command: **[delete command] [keyword(s)]**

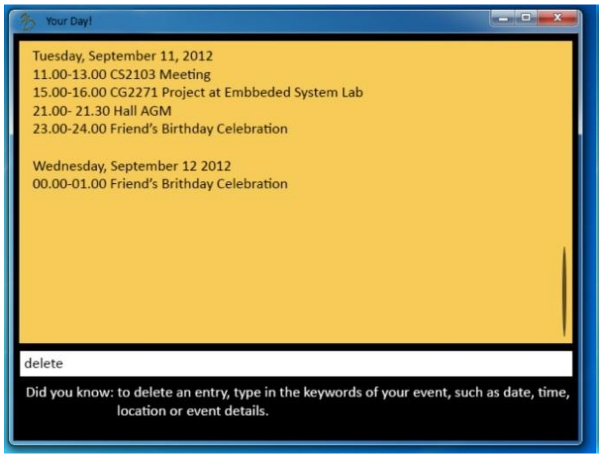
Possible delete commands: **delete, del, remove, cancel, clear, kill, -**

The delete function interface is similar to that of the edit function. The keywords can be a word or a part of the entry details, date, or time of the entry. After you enter the command, YourDay will give you a list of entries that matches your keywords. After that, you can select which entry you want to delete by entering the entry ID shown on your search result. You can delete multiple entries by entering:

Command: **[start ID]-[endID]**

[ID1],[ID2],[ID3],…

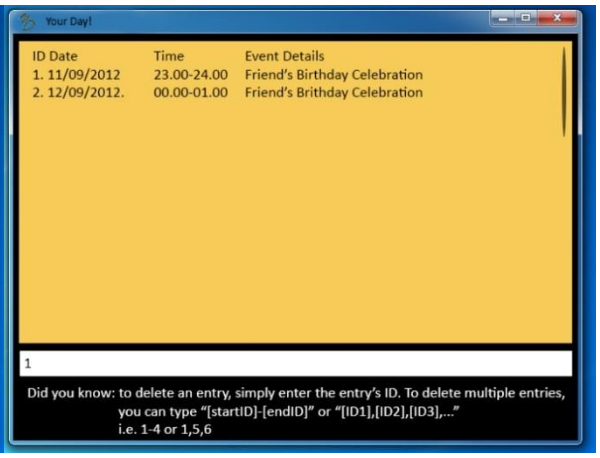
i.e. 1-6 or 1,2,4



```
Your Day!
Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

Wednesday, September 12 2012
00.00-01.00 Friend's Brithday Celebration

delete
Did you know: to delete an entry, type in the keywords of your event, such as date, time,
              location or event details.
```



```
Your Day!
ID Date          Time          Event Details
1. 11/09/2012    23.00-24.00   Friend's Birthday Celebration
2. 12/09/2012.   00.00-01.00   Friend's Brithday Celebration

1
Did you know: to delete an entry, simply enter the entry's ID. To delete multiple entries,
              you can type "[startID]-[endID]" or "[ID1],[ID2],[ID3],..."
              i.e. 1-4 or 1,5,6
```

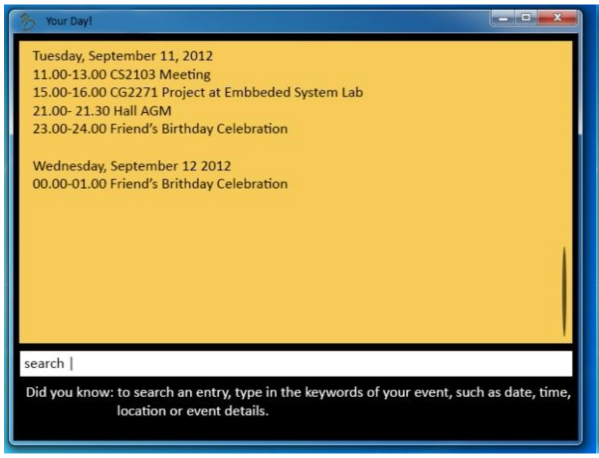
5. **Search Function (Power Search)**

In order to search a specific event, you need to enter a command in the following format:

Command: **[search command] [keyword(s)]**

Possible search commands: **search, find, lookup, ?**

After you enter the command, YourDay will give you a list of entries that matches your keywords. After that, you can select which entry you want to view by entering the entry ID shown on your search result. The keyword(s) can be anything from the event's details, location priority, time, and date. You would not even need to enter the exact word character by character. Our software will detect what you want to search.

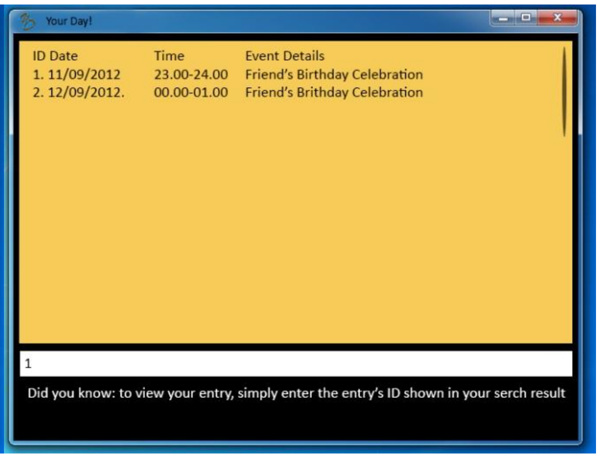


Tuesday, September 11, 2012
11.00-13.00 CS2103 Meeting
15.00-16.00 CG2271 Project at Embbeded System Lab
21.00- 21.30 Hall AGM
23.00-24.00 Friend's Birthday Celebration

Wednesday, September 12 2012
00.00-01.00 Friend's Brithday Celebration

```
search |
```

Did you know: to search an entry, type in the keywords of your event, such as date, time, location or event details.



```
ID Date          Time          Event Details
1. 11/09/2012    23.00-24.00   Friend's Birthday Celebration
2. 12/09/2012.   00.00-01.00   Friend's Brithday Celebration
```

```
1
```

Did you know: to view your entry, simply enter the entry's ID shown in your serch result

6. **Undo Function**

In order to undo an input, you need to enter a command in the following format:

Command: **undo**

Why YourDay?

YourDay offers so much more than your ordinary agenda. YourDay works very efficiently with our special feature, PowerSearch. Our PowerSearch features includes QuickSearch, Suggestions, MitsakeMeNot and

Glossary

PowerSearch: a powerful search function that provides suggestion and auto correction to user's search input.

QuickSearch : a powerful search feature that enables the program to output the entry found from the user's input without the need to push the return button or other special character to trigger the search engine

Suggestions : a powerful search feature that enables the user to input few characters and suggest a word that is included inside the entry database.

MitsakeMeNot : Error correction for user's input inside the search function

Developer Guide

Purpose

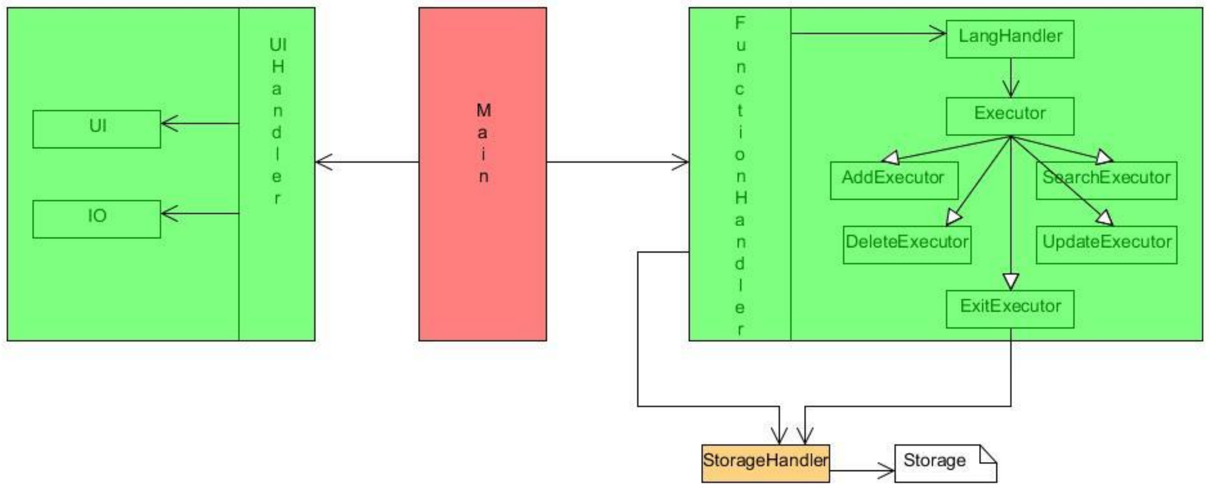
The developer's guide is intended for future developers that intend to further enhance the program. The guide will include how the program is structured, how the different classes work together and the main API's that runs the backbone of the program. This guide serves to introduce the program classes and methods, not the actual implementation of the program. With the information from this guide, the reader will be able to understand how each of our functions works and the architecture of our program.

Intended Audience

The information in this guide assumes that the reader has a decent level of C++ programming experience such as working with STL Vectors and Classes(Object Oriented Programming).

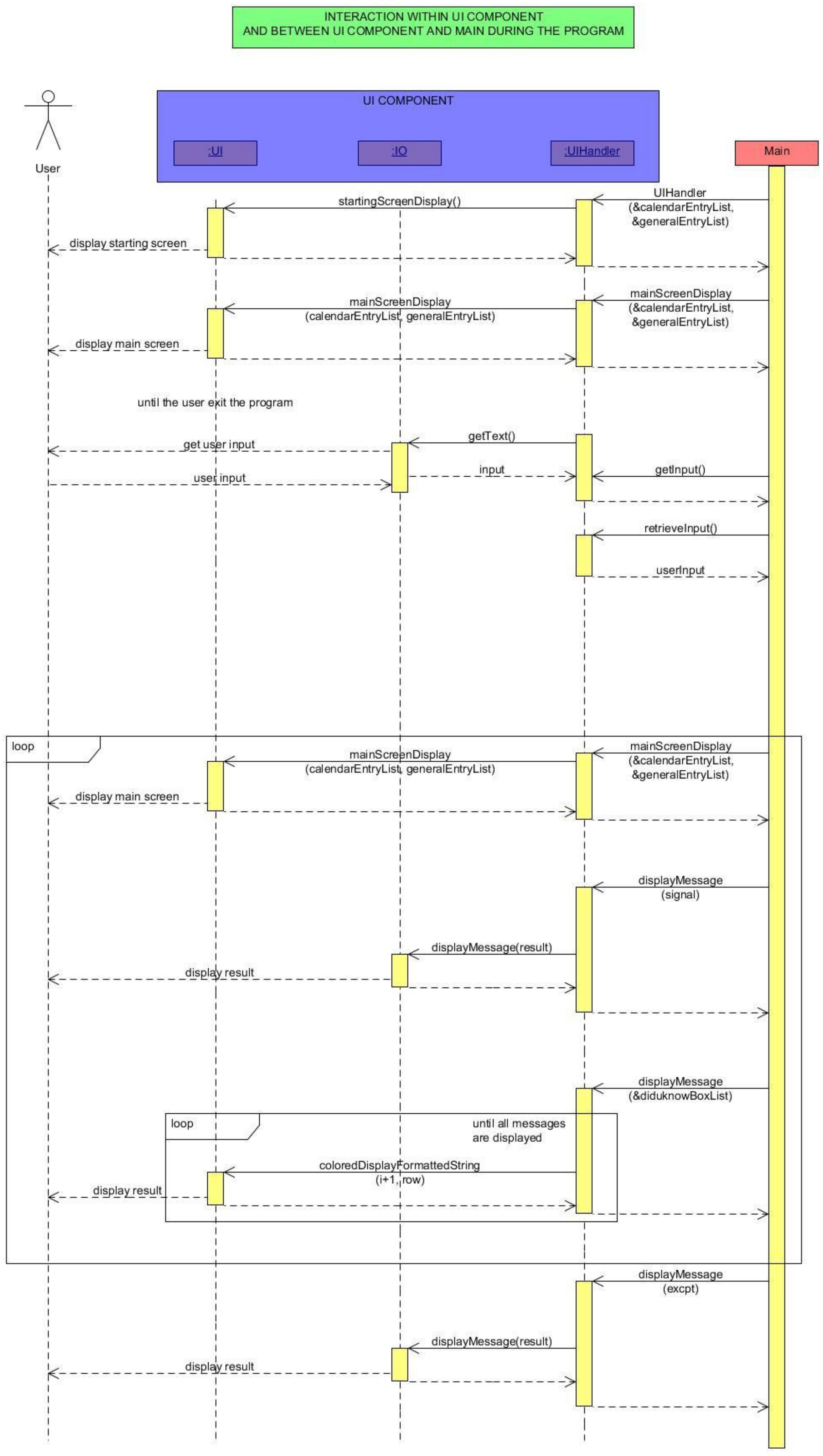
Architecture

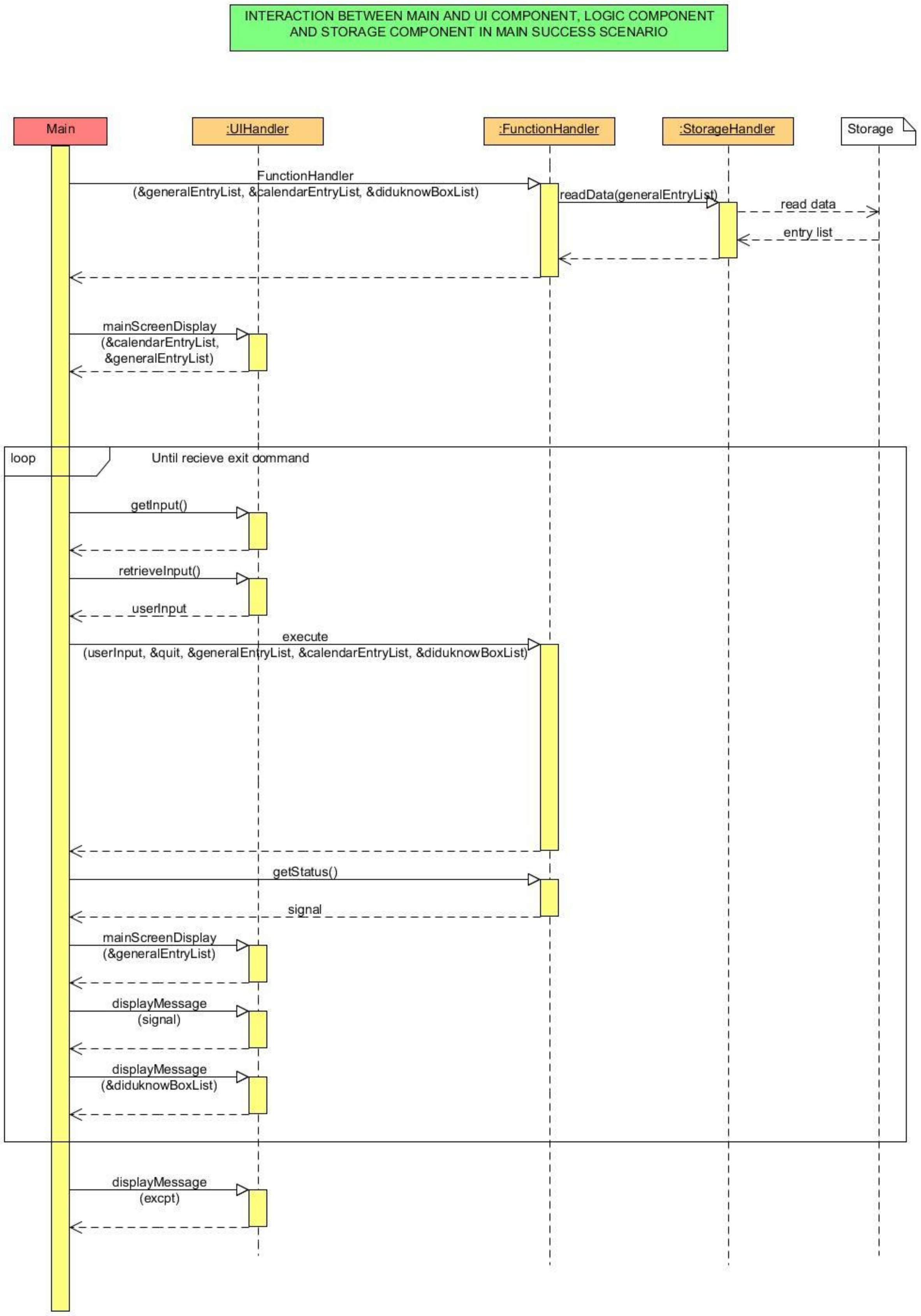
We use the transaction architecture for this program. The Main Handler does the transaction between UI and Logic part.

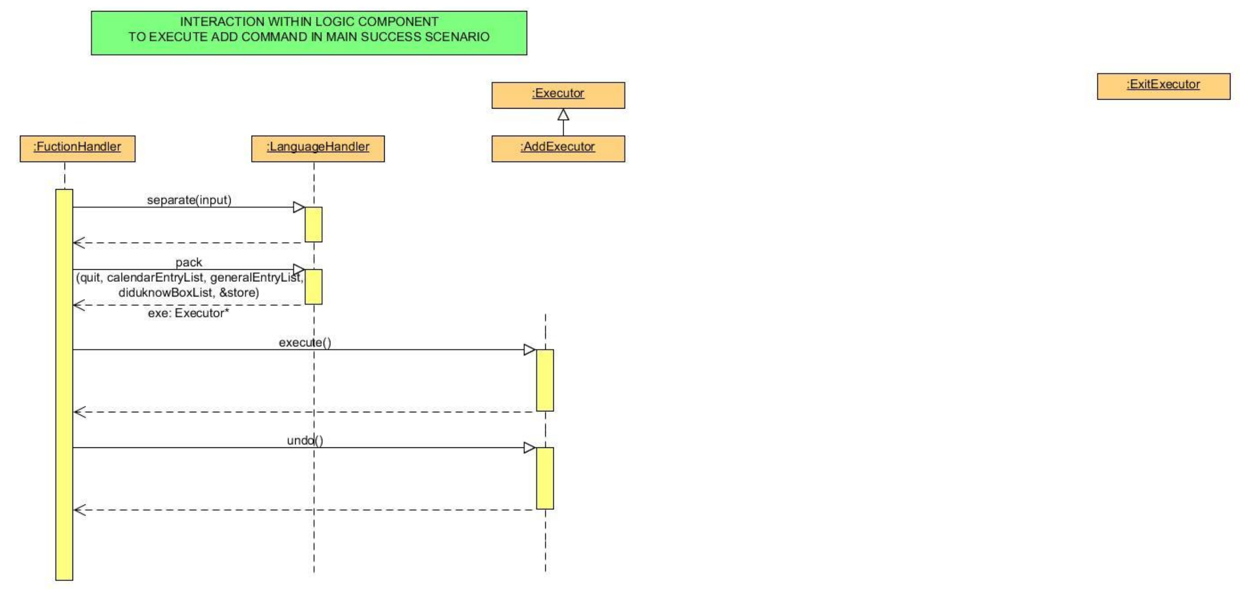


UML Sequence

The UML Sequences shown here is the main successful scenario for add command and it's executions


INTERACTION WITHIN UI COMPONENT
AND BETWEEN UI COMPONENT AND MAIN DURING THE PROGRAM
UI COMPONENT
User
:UI
:IO
:UIHandler
Main
UIHandler (&calendarEntryList, &generalEntryList)
startingScreenDisplay()
display starting screen
mainScreenDisplay (&calendarEntryList, &generalEntryList)
mainScreenDisplay (calendarEntryList, generalEntryList)
display main screen
until the user exit the program
getText()
get user input
input
getInput()
user input
retrieveInput()
userInput
loop
mainScreenDisplay (&calendarEntryList, &generalEntryList)
mainScreenDisplay (calendarEntryList, generalEntryList)
display main screen
displayMessage (signal)
displayMessage(result)
display result
displayMessage (&diduknowBoxList)
loop
until all messages are displayed
coloredDisplayFormattedString (i+1, row)
display result
displayMessage (excpt)
displayMessage(result)
display result


INTERACTION BETWEEN MAIN AND UI COMPONENT, LOGIC COMPONENT AND STORAGE COMPONENT IN MAIN SUCCESS SCENARIO
Main
:UIHandler
:FunctionHandler
:StorageHandler
Storage
FunctionHandler (&generalEntryList, &calendarEntryList, &diduknowBoxList)
readData(generalEntryList)
read data
entry list
mainScreenDisplay (&calendarEntryList, &generalEntryList)
loop
Until recieve exit command
getInput()
retrieveInput()
userInput
execute (userInput, &quit, &generalEntryList, &calendarEntryList, &diduknowBoxList)
getStatus()
signal
mainScreenDisplay (&generalEntryList)
displayMessage (signal)
displayMessage (&diduknowBoxList)
displayMessage (excpt)


INTERACTION WITHIN LOGIC COMPONENT
TO EXECUTE ADD COMMAND IN MAIN SUCCESS SCENARIO
:Executor
:ExitExecutor
:FuctionHandler
:LanguageHandler
:AddExecutor
separate(input)
pack
(quit, calendarEntryList, generalEntryList, diduknowBoxList, &store)
exe: Executor*
execute()
undo()

APIs list

UI CLASSES

1. UI Class

UI
-Signal status - HANDLE hConsole

- void displayBoard(Signal statusSignal)
- void setStatus(Signal statusSignal)
- void setInput(string textInput);
- void drawBanner()
- void writeWords(string words, int startH, int startW)
- void displayEntryList(vector<string>* calendarEntryList, vector<string> *generalEntryList)
+ void coloredDisplayFormattedString(int,string)
+ void setNormal()
+ void drawBox()
+ void didUKnowBox()
+ Signal getStatus()
+ void setScreenSize()
+ void clearStatus()
+ void gotoxy(int x,int y)
+ void startingScreenDisplay()
+ void mainScreenDisplay(vector<string>* calendarEntryList, vector<string>* generalEntryList)

a. Signal getStatus()
   - get the status signal of UI displaying process

b. void clearStatus()
   - clear the status signal of UI process to default CLEAR signal

c. void startingScreenDisplay()
   - show the starting screen to user at the beginning of the program

d. void mainScreenDisplay(vector<string>* calendarEntryList, vector<string>* generalEntryList)
   - show the main screen to interact with the user

e. void gotoxy(int x, int y)
   - moves the cursor to specific coordinate of the screen

f. void setNormal()
   - Sets control attribute

g. Void drawBox()
   - Draws a box for text background

h. void didUKnowBox()
   - Draws the "Did you know box"

2. IO Class

IO
- Signal status - string input
- void setStatus(Signal statusSignal) - void setInput(string textInput); + Signal getStatus() + void clearStatus() + void getRawInput() + string getText() + void displayMessage(string output)

a. void getRawInput()
    - get user input string through command line
b. void displayMessage(string output)
    - display feedback message string of the system to user
c. Signal getStatus()
    - get the status signal of IO displaying process
d. void clearStatus()
    - clear the status signal of IO process to default CLEAR signal

EXECUTOR CLASSES

1. Executor Class (Super Class)

Executor
Signal status # int findBlockIndex(string details, int blockLocation) # string extractField(string details, int startLocation) # int extractIndex(string details) # string extractDescription(string details) # string extractLocation(string details) # string extractTime(string details) # string extractDate(string details)

int extractPriority(string details)
+ void execute() (virtual) + void undo() (virtual) + Signal getStatus()

a. void execute()
   - executes the predefined function of the executor

b. void undo()
   - Undo the last changes made by the executor

c. void getStatus
   - Returns the status of the executor

2. AddExecutor Class

AddExecutor
Signal status # int findBlockIndex(string details, int blockLocation) # string extractField(string details, int startLocation) # int extractIndex(string details) # string extractDescription(string details) # string extractLocation(string details) # string extractTime(string details) # string extractDate(string details) # int extractPriority(string details) - vector<string>* _calendarEntryList - vector<string>* _generalEntryList; - vector<string> _undoGeneralEntryList - vector<string> _undoCalendarEntryList - string _details
+ void execute() + void undo() + Signal getStatus()

a. void execute()
   - executes the add entry functionality to storage

- the entry can be saved either to the Calendar Entry List or the General Entry List depending on the format of the entry

b. void undo()

- Undo the last changes made by the executor

c. void getStatus

- Returns the status of the executor

3. DeleteExecutor Class

DeleteExecutor
Signal status
int findBlockIndex(string details, int blockLocation)
string extractField(string details, int startLocation)
int extractIndex(string details)
string extractDescription(string details)
string extractLocation(string details)
string extractTime(string details)
string extractDate(string details)
int extractPriority(string details)
- vector<string>* _calendarEntryList
- vector<string>* _generalEntryList;
- vector<string> _undoGeneralEntryList
- vector<string> _undoCalendarEntryList
- string _details
+ void execute()
+ void undo()
+ Signal getStatus()

a. void execute()

- executes the delete entry functionality from storage
- the entry can be deleted either from the Calendar Entry List or the General Entry List depending on the index chosen

b. void undo()

- Undo the last changes made by the executor

c. void getStatus

- Returns the status of the executor

4. ExitExecutor Class

ExitExecutor
Signal status # int findBlockIndex(string details, int blockLocation) # string extractField(string details, int startLocation) # int extractIndex(string details) # string extractDescription(string details) # string extractLocation(string details) # string extractTime(string details) # string extractDate(string details) # int extractPriority(string details) -vector<string>* _generalEntryList; StorageHandler* _store; bool* _quit; vector<string>* _calendarEntryList
+ void execute() + Signal getStatus()

a. void execute()
    - signals the storage handler to save the data and exits the program

b. void getStatus
    - Returns the status of the executor

5. SearchExector Class

SearchExecutor
Signal status # int findBlockIndex(string details, int blockLocation) # string extractField(string details, int startLocation) # int extractIndex(string details) # string extractDescription(string details) # string extractLocation(string details) # string extractTime(string details)

string extractDate(string details)
int extractPriority(string details)
- vector<string>* _entryList;
- vector<string>* _matchedEntryList;
- vector<string> _undoEntryList;
- vector<string> _undoMatchedEntryList;
- string _details;
+ void execute()
+ void undo()
+ Signal getStatus()

a. void execute()
   - executes the search entry functionality from storage by using the user's input keyword
   - the entry can be either from the Calendar Entry List or the General Entry List and it will be stored inside the matchedEntryList;

b. void undo()
   - Undo the last changes made by the executor

c. void getStatus
   - Returns the status of the executor

6. UpdateExecutor Class

UpdateExecutor
Signal status
int findBlockIndex(string details, int blockLocation)
string extractField(string details, int startLocation)
int extractIndex(string details)
string extractDescription(string details)
string extractLocation(string details)
string extractTime(string details)
string extractDate(string details)
int extractPriority(string details)
- vector<string>* _calendarEntryList
- vector<string>* _generalEntryList;

- vector<string> _undoGeneralEntryList
- vector<string> _undoCalendarEntryList
- string _details
+ void execute()
+ void undo()
+ Signal getStatus()

a. void execute()
   - executes the update entry functionality from storage
   - the updated entry can be either from the Calendar Entry List or the General Entry List depending on the index chosen

b. void undo()
   - Undo the last changes made by the executor

c. void getStatus
   - Returns the status of the executor

HANDLER CLASSES

1. UI Handler

UIHandler
`- StatusHandler sh`
`- void setUIStatus(Signal statusSignal)`
`- string interpreteSignal(Signal outSignal)`
`- IO io;`
`- UI ui;`
`- Signal UIstatus`

```
+ Signal getStatus()

+ void setStatus()

+ void getInput()

+ void displayMessage(Signal outSignal)

+ void displayMessage(vector<string>* result)

+ void displayMessage(string result)

+ bool void startingScreenDisplay()

+ void mainScreenDisplay(vector<string>* calendarEntryList, vector<string>* generalEntryList)

+ string retrieveInput()

+ Signal getStatus()

+ void clearStatus()
```

a. void setStatus()
   - Sets the status of the UI handler after printing UI

b. void getInput()
   - This operation is used to get input message from the user. It calls IO to get user input through command line and store it in the private string named input

c. void displayMessage(Signal outSignal)
   - This operation is used to display output message to the user. It takes in the feedback signal passed by main(), call private interpretSignal(Signal) to intepret to a string and calls IO to output to the user

d. void displayMessage(vector<string>* result)
   - Displays messages or strings that have been stored in the string. We use this to output the search result.

e. void displayMessage(string result)
   - Displays a string containing the operation result.

f. void startingScreenDisplay()
   - This operation is used to display the startup screen to the user at the beginning of the program

g. void mainScreenDisplay(vector<string>* calendarEntryList, vector<string>* generalEntryList)
   - This operation is used to display the main screen through UI class which interfaces the interaction with user. It will call the UI to display the main screen which splits the screen into 3 part, the Calendar List, the General List, and the input screen.

h. string retrieveInput()

- This method retrieves user's input after processed in the IO class

i. Signal getStatus()
   - This method returns the status of UI handler

j. void clearStatus();
   - This method clears the UI handler's status

2. Function Handler

FunctionHandler
- StatusHandler sh - vector<string> ram - Signal fxStatus - StorageHandler store - stack<Executor*> undoStk
+ Signal getStatus() + void setStatus() + void execute(string input, bool* quit,vector<string>* generalEntryList, vector<string>* calendarEntryList, vector<string>* diduknowBoxList)

a. void setStatus()
   - sets the status attribute of the Function Handler class

b. Signal getStatus()
   - Retrieves the current status of the class

c. void execute(string input, bool* quit,vector<string>* generalEntryList, vector<string>* calendarEntryList, vector<string>* diduknowBoxList)
   - The operation is periodically called in main(). It will handle the flow of the logic component.

3. Language Handler

LanguageHandler

```
- StatusHandler sh;
- Signal command;
- Signal langStatus;
- string details;
```

```
- bool leap(int year)
- bool isDate(string date)
- bool isTime(string time)
- bool isInt(string inx)
- bool isLogicDate(string date)
- bool isLogicTime(string time)
- bool isLogicPriority(string priority)
- void encoder(string input, Signal command)
- void setCommand(string userCommand)
+ Signal getStatus()
+ void separate(string userInput) throw (string)
+ Executor* pack(bool* quit, vector<string>* calendarEntryList,
vector<string>* generalEntryList,vector<string>* diduknowBoxList,
StorageHandler* store)
```

a. void getStatus()
   - Retrieves the status of Language Handler
b. void separate(string userInput)
   - Separates user input's string into 2 parts, the input and the string to be processed
c. Executor* pack(bool* quit, vector<string>* calendarEntryList, vector<string>* generalEntryList,vector<string>* diduknowBoxList, StorageHandler* store)
   - Creates an appropriate executor based on the commands

4. Storage Handler

StorageHandler

```
- char buffer[MAXMIUM_WORDS]

- static string DataBaseFile

- static string DataBaseTempFile
```

```
+ Signal getStatus()

+ void setStatus()

+ void readData(vector<string>  *ram)

+ void writeData(vector<string>  *ram)

+ bool checkFileExistence(string filePath, string fileName)

+ void disassociateFile(fstream & file)

+ void associateFile(string filePath, string fileName, fstream & file,
OPEN_TYPE mode)

+ void deleteFile(string filePath, string fileName)

+ void renameFile(string filePath, string oriName, string newName)

+ void replaceFile(string oriPath, string oriName, string repName)
```

a. void setStatus()
   - Sets the status of the language handler after input processing

b. void readData(vector<string> *ram)
   - Read the data stored in the file, and put it in a vector of string.

c. void writeData(vector<string> *ram)
   - Write the data stored to the file, the data is stored in a vector of string.

d. bool checkFileExistence(string filePath, string fileName)
   - Checks whether the file located in filePath and fileName do exist

e. void disassociateFile(fstream & file)
   - Removes the file association within the program

f. void associateFile(string filePath, string filename, fstream & file, OPEN_TYPE mode)
   - Associate files to the program

g. void deleteFile(string filePath, string fileName)
   - Deletes text file with the same path and name as filePath and fileName

h. void renameFile(string filePath, string oriName, string newName)
   - Renames the file

i. void replaceFile(string oriPath, string oriName, string repName)
   - Replace an old file with the newly updated files

5. StatusHandler

StatusHandler

```
+ bool success(Signal signal)

+ bool error(Signal signal)
```

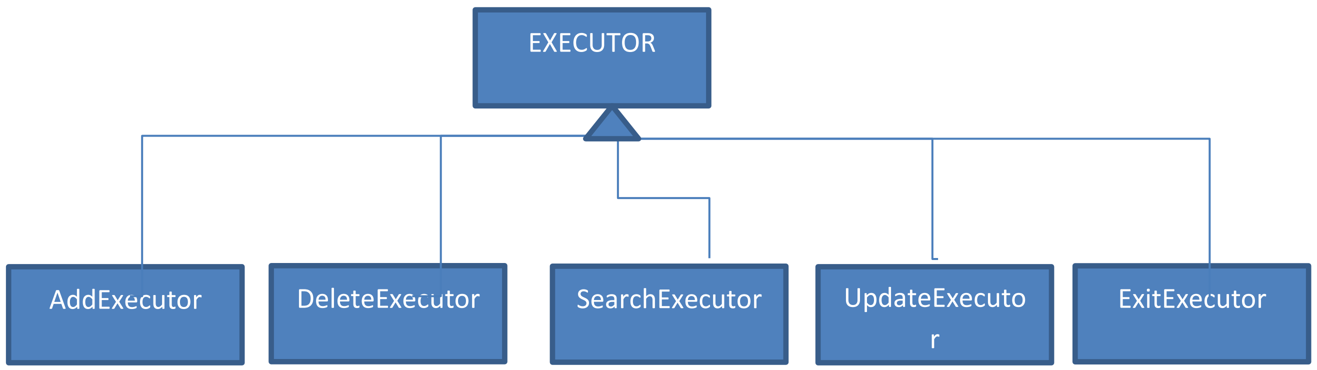
a. bool success(Signal signal)

- Check the signal whether it is a success or fail

b. bool error (Signal signal)

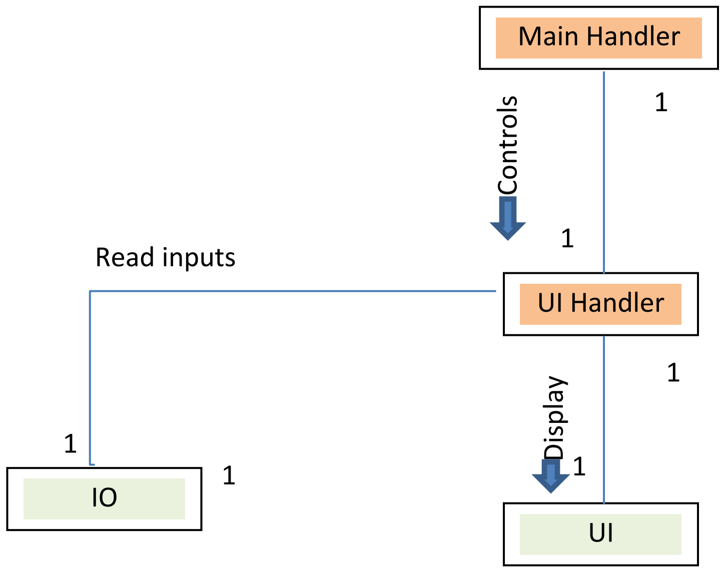
- Check if the signal is an error.

Inheritance Diagrams



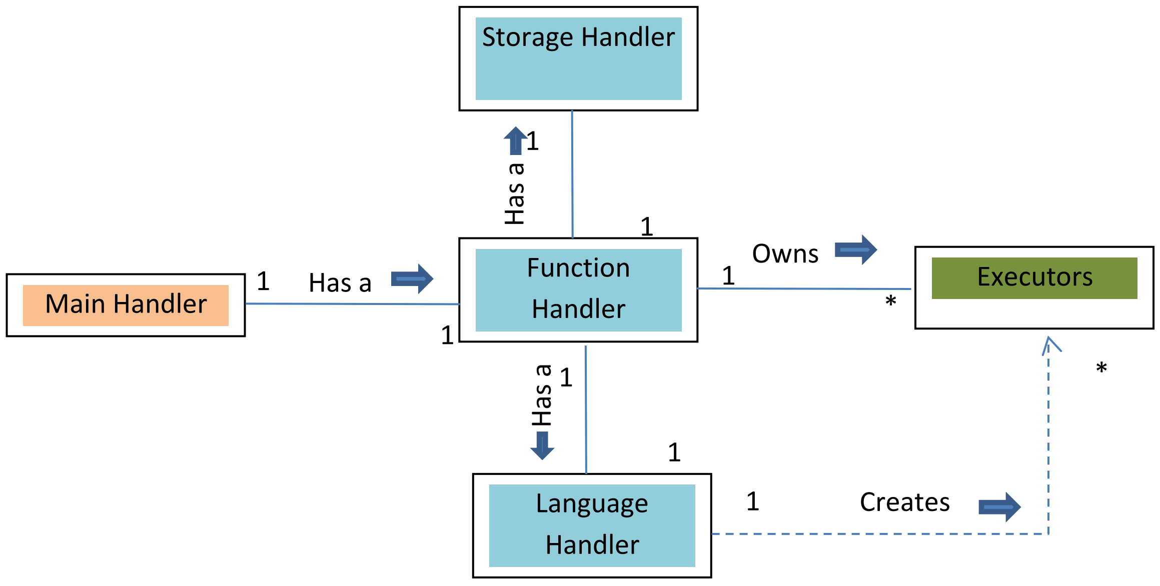
UI Association Diagrams

This diagram shows the association between the Main, UIHandler, and UI class.



Logic Associations Diagram

Our Logic part comprises of quite a handful of classes. This class diagram shows the association between the Main Handler and the Logic Parts.



FUTURE ONGOING DEVELOPMENTS

- Solving issues in coupling of functions
- Implements the UI layer thoroughly
- Implements all of the functionalities