





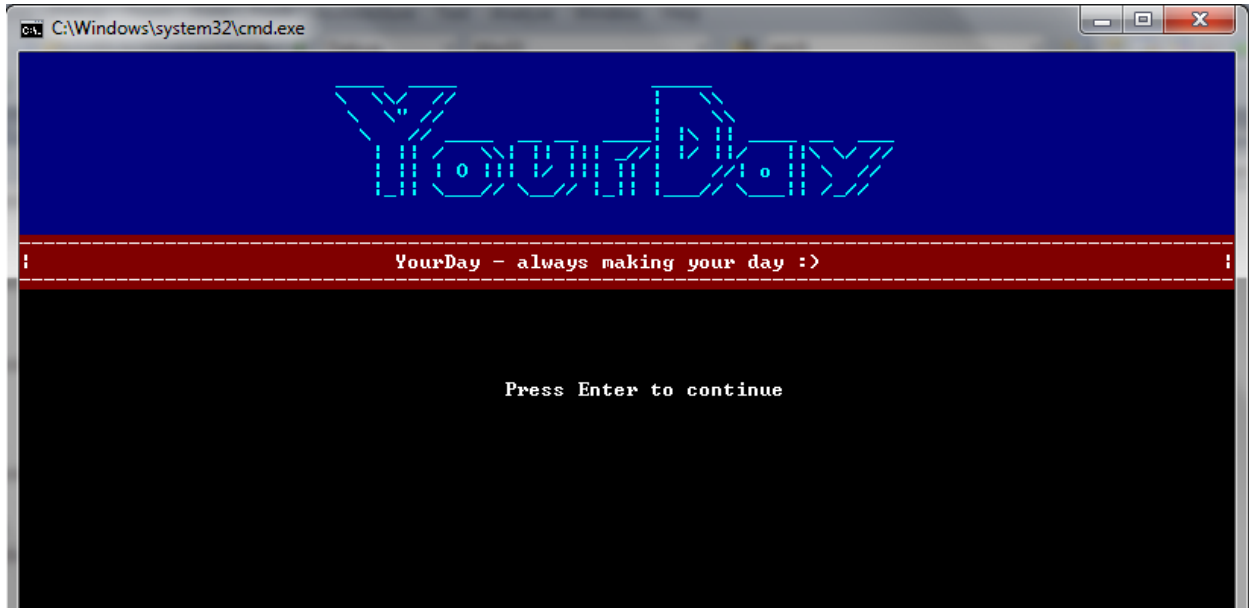
## YourDay



 <p>Ignatius Damai Team Leader Deadline Watcher</p>	 <p>Huang Da Lead Coder</p>	 <p>Wu Pei Lead Tester</p>	 <p>Nguyen Ngoc Nhu Thao Lead UI</p>
--	--	--	---

## 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.

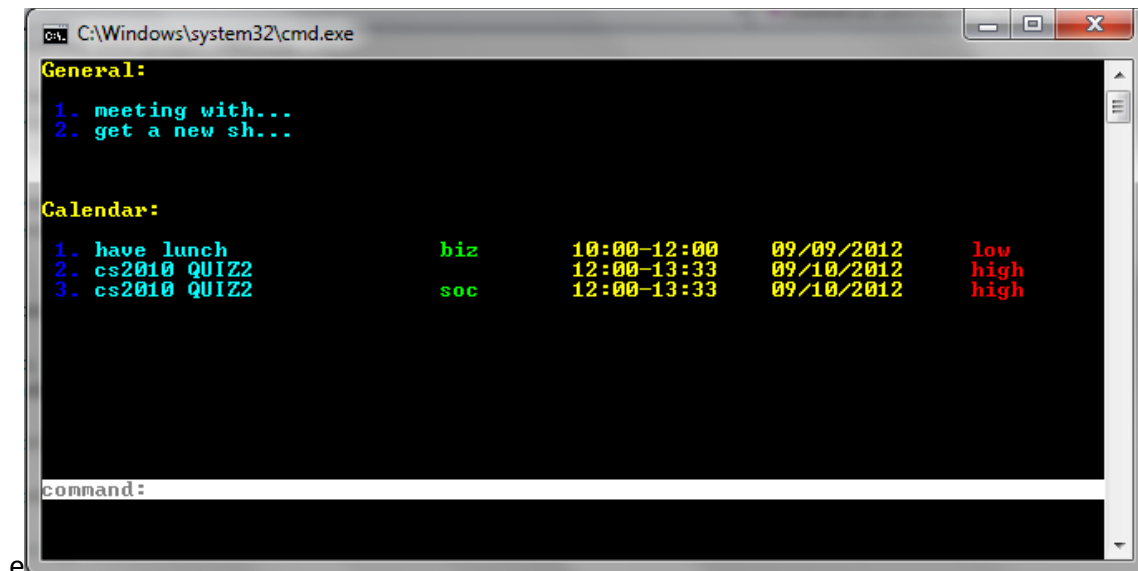


Figure 1

### 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

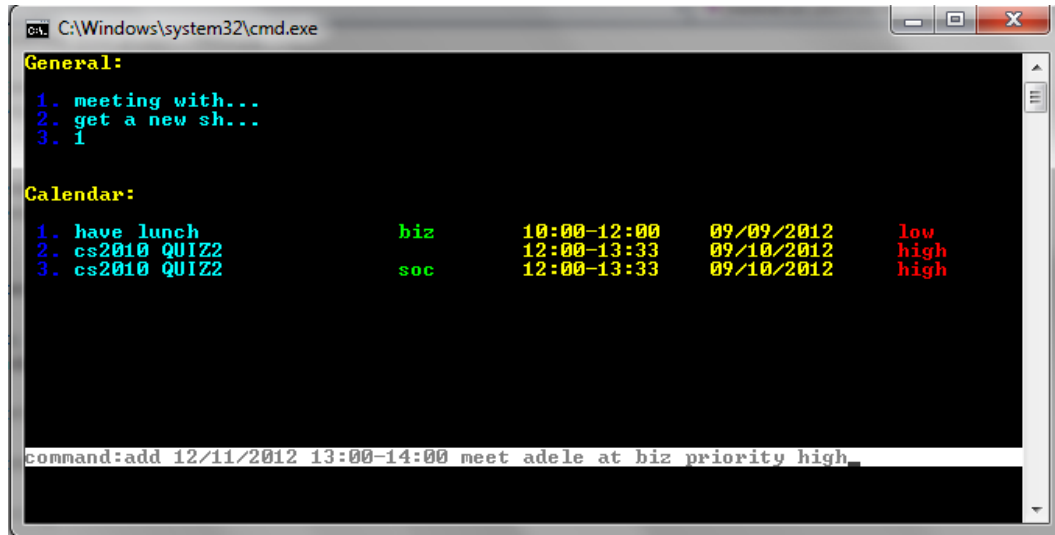


Figure 2

You can add more tasks details such as priority or location easily as shown on figure 2 (above). 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.

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, !

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 in your 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.

#### 4. Deleting an Entry

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

**Command:** [delete command] [index]

**Possible delete commands:** delete

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

```

C:\Windows\system32\cmd.exe
General:
1. meeting with...
2. 1

Calendar:
1. have lunch          biz      10:00-12:00    09/09/2012    low
2. cs2010 QUIZ2        12:00-13:33    09/10/2012    high
3. cs2010 QUIZ2        12:00-13:33    09/10/2012    high
4. meet adele          biz      13:00-14:00    12/11/2012    high

command:delete 2G
  
```

Figure 3

#### 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

```

C:\Windows\system32\cmd.exe
General:
6. Add to test
7. Add to test
8. Add to test
9. Add to test
10. Add to test
Calendar:
1. have lunch          biz      10:00-12:00    09/09/2012    low
2. cs2010 QUIZ2        12:00-13:33    09/10/2012    high
3. cs2010 QUIZ2        soc       12:00-13:33    09/10/2012    high
command:search lunch

```

Figure 4

After you enter the command, YourDay will give you a list of entries that matches your keywords (Figure 5).

```

C:\Windows\system32\cmd.exe
General:
6. Add to test
7. Add to test
8. Add to test
9. Add to test
10. Add to test
Calendar:
1. have lunch          biz      10:00-12:00    09/09/2012    low
2. cs2010 QUIZ2        12:00-13:33    09/10/2012    high
3. cs2010 QUIZ2        soc       12:00-13:33    09/10/2012    high
11 have lunch          biz      10:00-12:00    09/09/2012    low
command:

```

Figure 5

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.

## 6. Undo Function

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

**Command:**                      **undo**

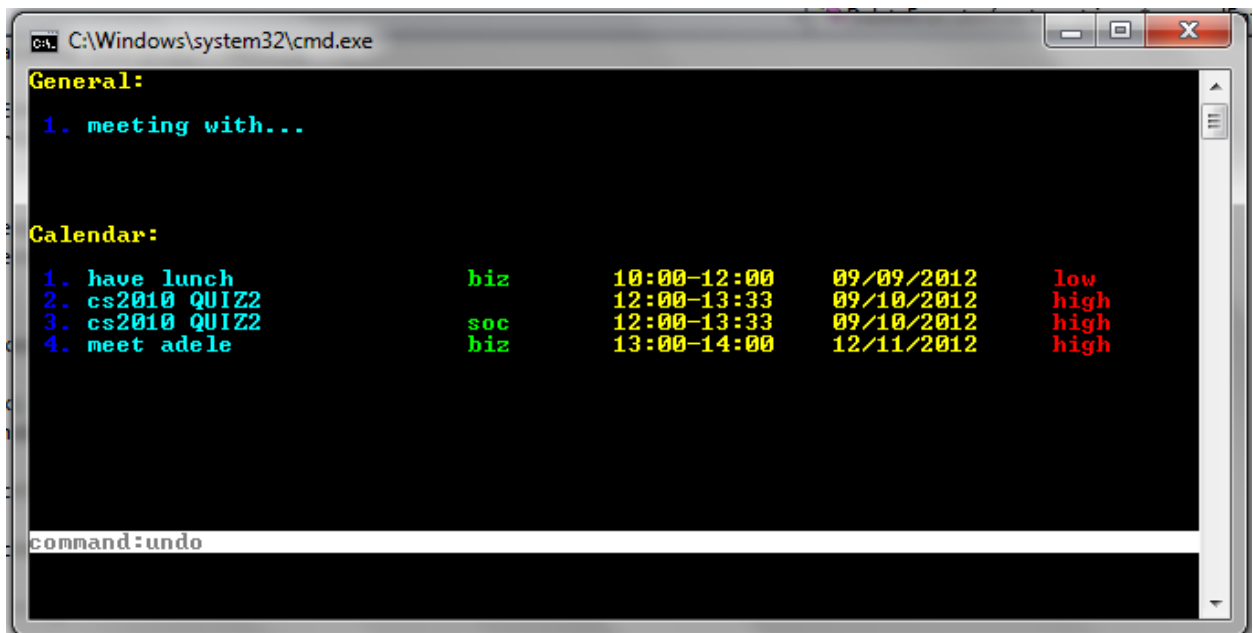


Figure 6

## 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

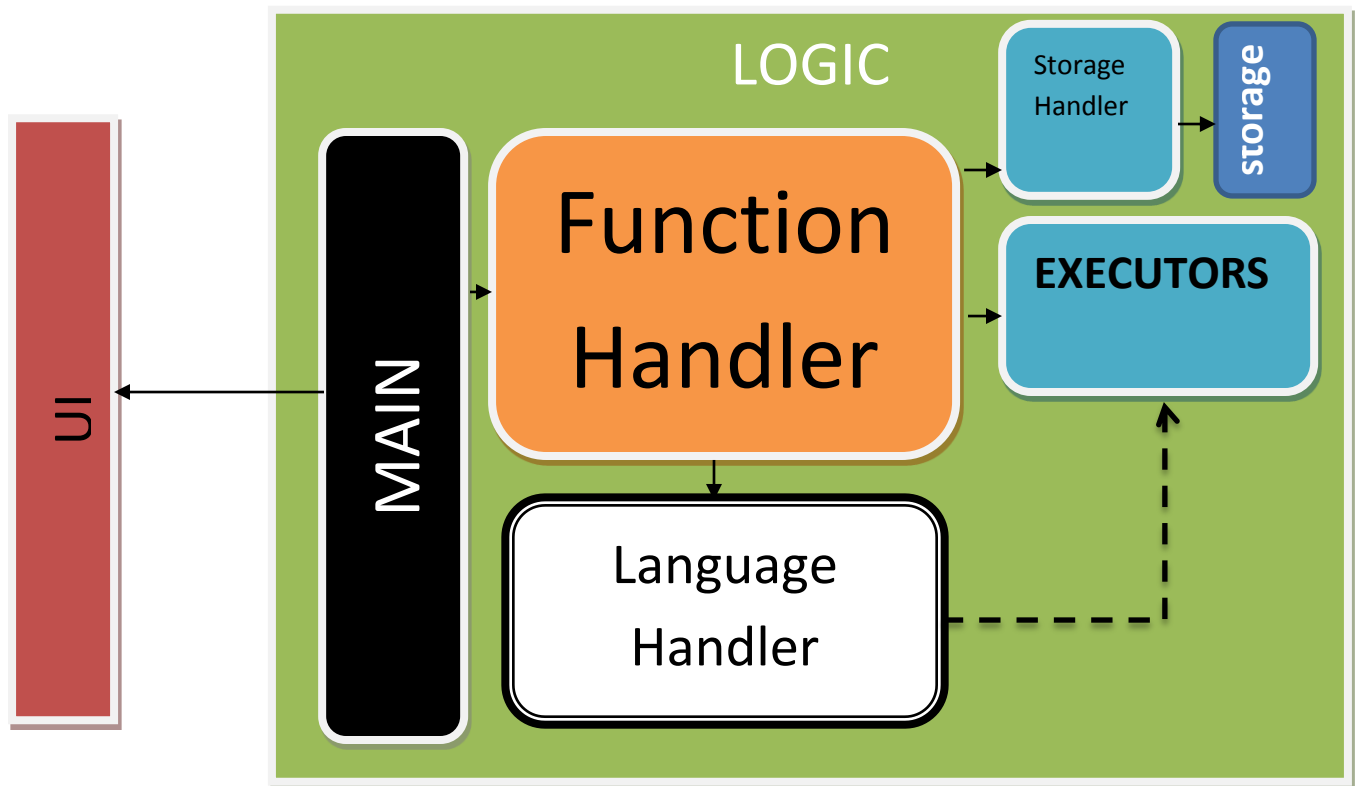
## Purpose

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).

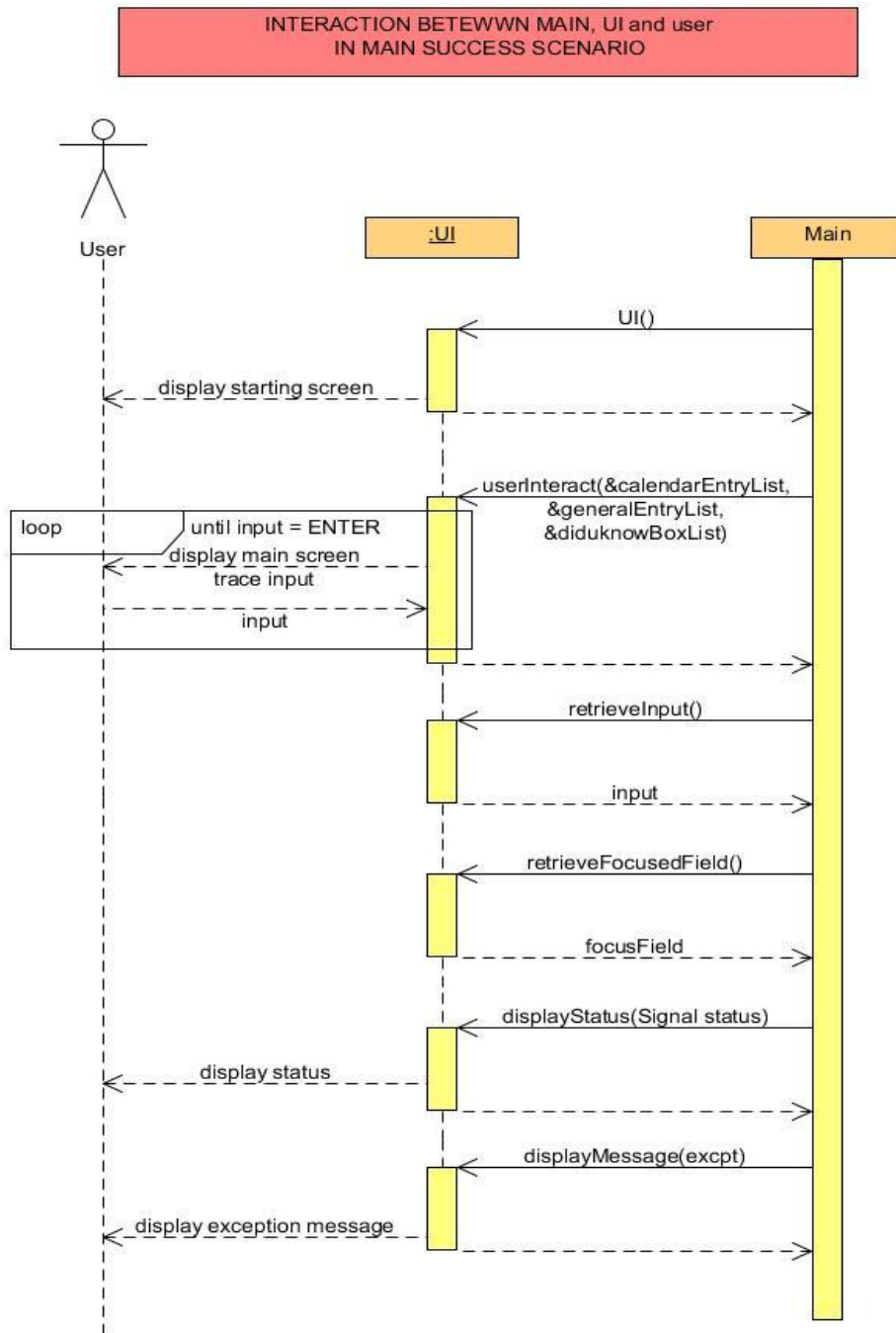
## Transaction Architecture



The next 3 pages containing the UML Sequence Diagram, classes' attributes and properties, and inheritance will be presented in landscape.

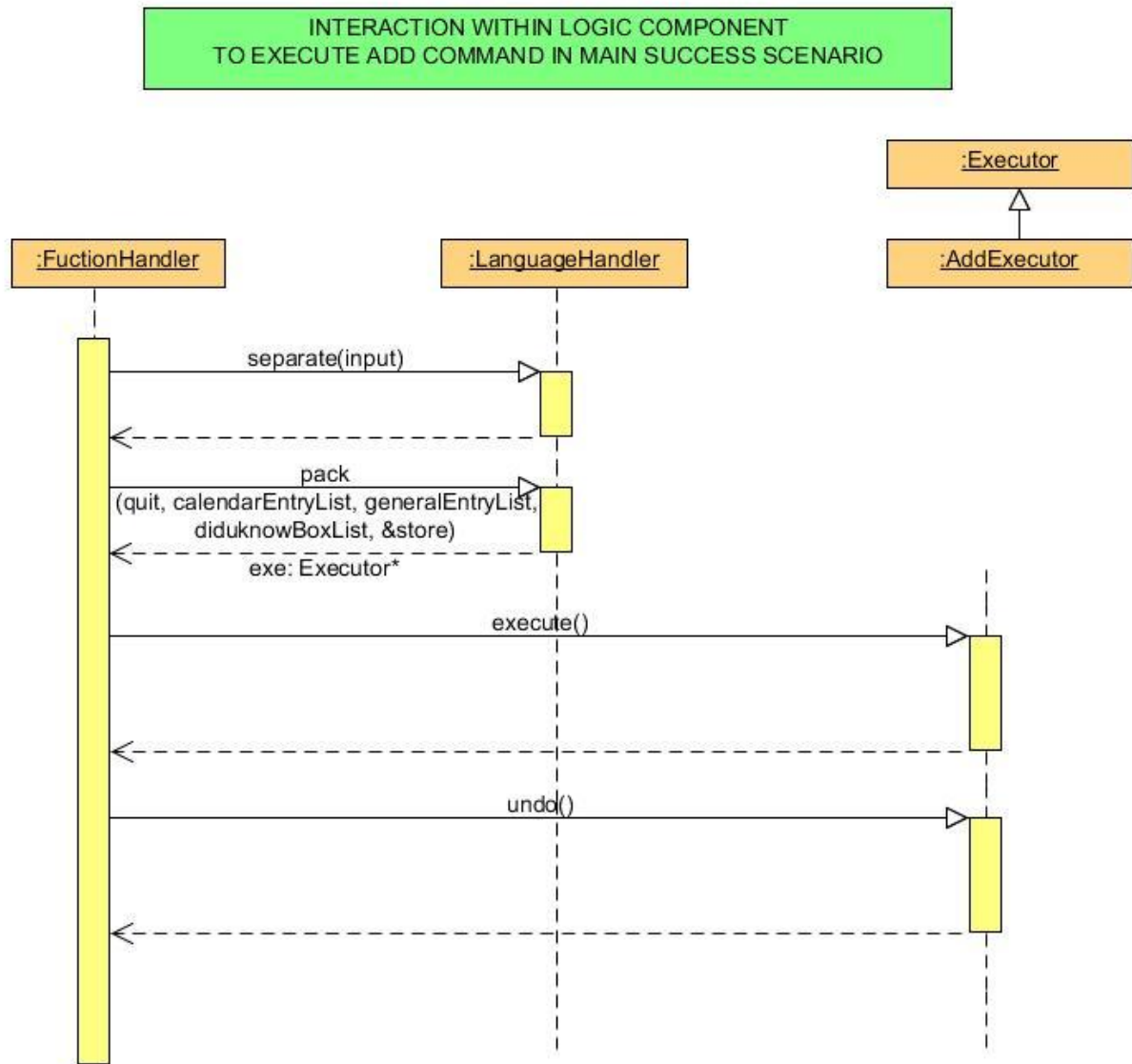
**Sequence Diagrams:**

This is the sequence diagram to show interaction between main, UI and the user in main success senario

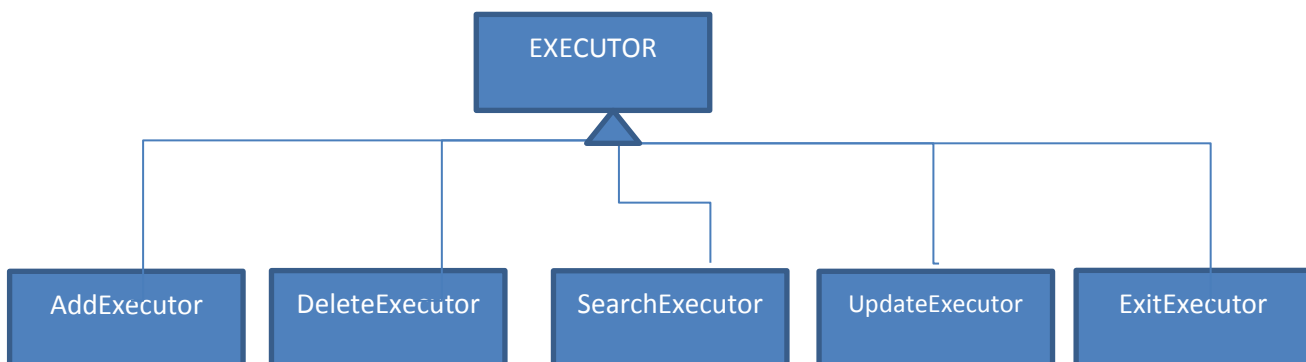




This is the sequence diagram to show interaction within logic component to execute add command in main success scenario.



#### Inheritance Diagrams

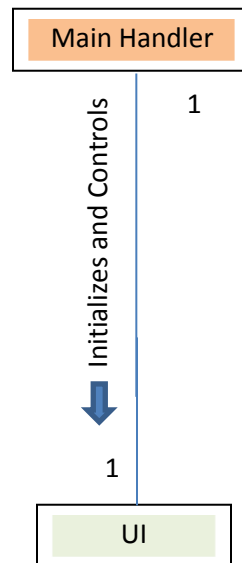


In this software, we use 1 super class which is the executor class. The main purpose of this class is to improve the simplicity of executing user's command and manage the tracking of the commands that has been executed. We have 5 inherited class from the main executor class, which are:

- AddExecutor
- DeleteExecutor
- SearchExecutor
- UpdateExecutor
- ExitExector

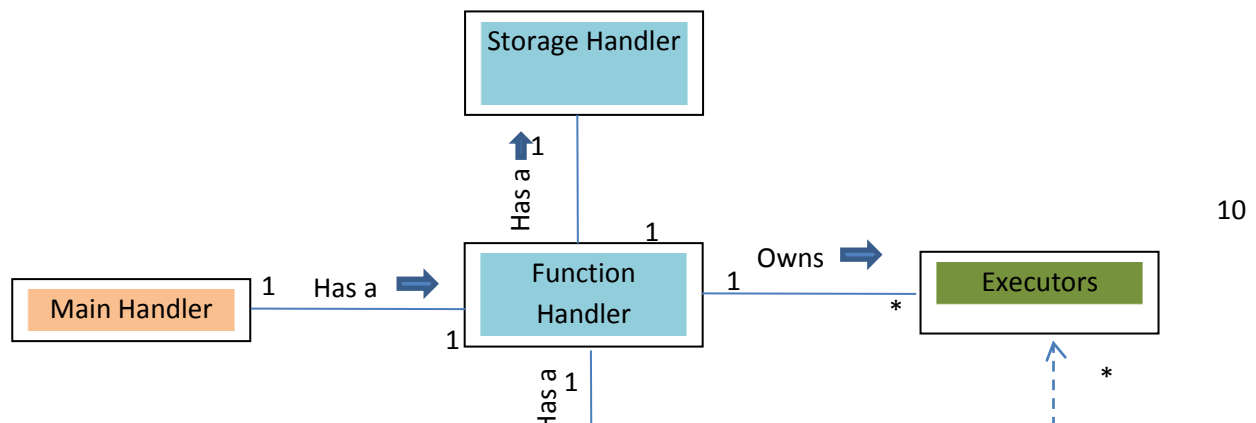
#### UI Association Diagrams

This diagram shows the association between the Main, 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.



The executors itself can be an add executor, delete executor, or any other executor inherited from the Executor super class

## 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”

**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
  - Parameter = void
  - Return =void

## b. void undo()

- Undo the last changes made by the executor
  - Parameter = void
  - Return = void

## c. Signal getStatus()

- Returns the status of the executor
  - Parameter= void
  - Return = Signal status of the executor

## 2. AddExecutor Class

**AddExecutor**

# Signal status

- vector&lt;string&gt;\* \_calendarEntryList

- vector&lt;string&gt;\* \_generalEntryList;

```

- vector<string> _undoGeneralEntryList
- vector<string> _undoCalendarEntryList
- string _details

# 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()
+ 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
  - Parameter = void
  - Return =void

b. void undo()

- Undo the last changes made by the executor
  - Parameter = void
  - Return =void

c. Signal getStatus

- Returns the status of the executor
  - Parameter = void
  - Return = Signal status of the executor

### 3. DeleteExecutor Class

## DeleteExecutor

```

# Signal status
- vector<string>* _calendarEntryList
- vector<string>* _generalEntryList;

```

```

- vector<string> _undoGeneralEntryList
- vector<string> _undoCalendarEntryList
- string _details

# 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()
+ 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
  - Parameter = void
  - Return =void

b. void undo()

- Undo the last changes made by the executor
  - Parameter = void
  - Return =void

c. void getStatus

- Returns the status of the executor
  - Parameter = void
  - Return = Current signal status of the executor

#### 4. ExitExecutor Class

### ExitExecutor

```

# Signal status
-vector<string>* _generalEntryList;

```

```

- StorageHandler* _store;

- bool* _quit; vector<string>* _calendarEntryList

# 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()

+ Signal getStatus()

```

- a. void execute()
  - signals the storage handler to save the data and exits the program
    - Parameter = void
    - Return =void
- b. void getStatus
  - Returns the status of the executor
    - Parameter = void
    - Return = current signal status

## 5. SearchExecutor Class

### SearchExecutor

```

# Signal status

- vector<string>* _calendarEntryList;
- vector<string>* _generalEntryList;
- vector<string>* _matchedEntryList;
- vector<string> _undoEntryList;
- vector<string> _undoMatchedEntryList;
- string _details;

```



```

# 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()
+ 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;
  - Parameter = void
  - Return =void

b. void undo()

- Undo the last changes made by the executor
  - Parameter = void
  - Return =void

c. void getStatus

- Returns the status of the executor
  - Parameter = void
  - Return = current signal status of the executor

## 6. UpdateExecutor Class

### UpdateExecutor

```

# Signal status
- vector<string>* _calendarEntryList
- vector<string>* _generalEntryList;
- vector<string> _undoGeneralEntryList
- vector<string> _undoCalendarEntryList
- string _details

```

```
# 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()
+ 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
    - Parameter = void
    - Return =void
- b. void undo()
  - Undo the last changes made by the executor
    - Parameter = void
    - Return =void
- c. void getStatus
  - Returns the status of the executor
    - Parameter = void
    - Return = current signal status of the executor

## HANDLER CLASSES

### 1. Function Handler

#### FunctionHandler

```

- StatusHandler sh
- 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
      1. Parameter = void
      2. Return =void
  - b. Signal getStatus()
    - Retrieves the current status of the class
      1. Parameter = void
      2. Return = current signal status of the function handler
  - 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.
      1. Parameter
        - a. string input = passing the input string to the function
        - b. bool\* quit = pass the pointer for Boolean function that contains
 whether the user wants to quit the program
        - c. vector<string>\* generalEntryList is the vector contains general
 entries
        - d. vector<string>\* calendarEntryList is the vector that contains
 calendar entries
        - e. vector<string>\* diduknowBoxList is the vector that contains
 hints and general feedbacks in using the programs.
      2. Return =void
2. 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
    1. Parameter = void
    2. Return =void
- b. void separate(string userInput)
  - Separates user input's string into 2 parts, the input and the string to be processed
    1. Parameter = string that contains user's input
    2. Return =void
- c. Executor\* pack(bool\* quit, vector<string>\* calendarEntryList, vector<string>\* generalEntryList,vector<string>\* diduknowBoxList, StorageHandler\* store)
  - Creates an appropriate executor based on the commands Parameter
    1. Parameter
      - a. string input = passing the input string to the function

- b. `bool* quit` = pass the pointer for Boolean function that contains whether the user wants to quit the program
- c. `vector<string>* generalEntryList` is the vector contains general entries
- d. `vector<string>* calendarEntryList` is the vector that contains calendar entries
- e. `vector<string>* diduknowBoxList` is the vector that contains hints and general feedbacks in using the programs.

2. Return = void

### 3. 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 storage after input processing
    - Parameter = void
    - Return =void
- b. `void readData(vector<string> *ram)`
  - Read the data stored in the file, and put it in a vector of string.

- Parameter = vector that contains all of the string being used on the fly by the program
- Return =void

c. void writeData(vector<string> \*ram)

- Write the data stored to the file, the data is stored in a vector of string.
  - Parameter = vector that contains all of the string being used on the fly by the program
  - Return =void

#### 4. 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
  - Parameter = signal that is going to be checked
  - Return = the validity of the signal to the specification

b. bool error (Signal signal)

- Check if the signal is an error.
  - Parameter = signal that is going to be checked
  - Return = the validity of the signal to the specification

## Future Developments

- Extensive coverage of PowerSearch using combination of many proven algorithm such as Levensthein's Algorithm
- Improvements in refactoring
- Repolished UI
- Implementation of DidU KnowBox