





YourDay



 <p>Ignatius Damai Team Leader Lead Documenter 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.

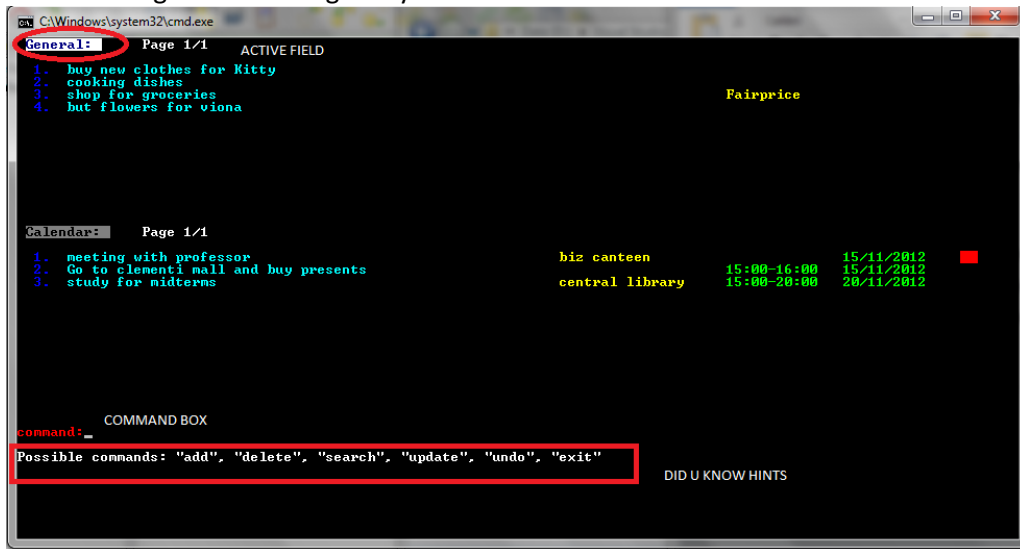


Figure 1

First of all, our program has special features called the Active Field. The available Active fields are:

- General Field
- Calendar Field
- SearchBox Field (activated only after search command is commenced)

The current chosen Active Field's title will be highlighted.

To simplify your life, we also provided a DidUKnow box below of the command bar. This box will give you hints for the correct inputs.

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

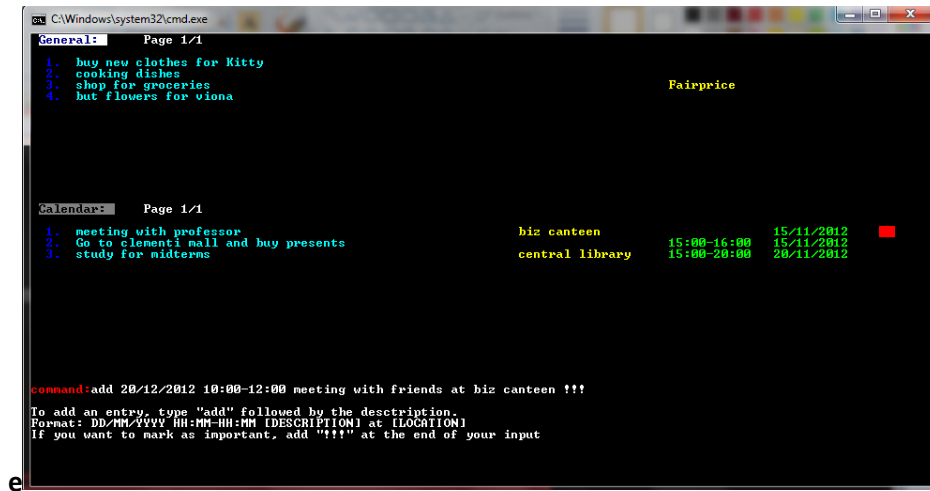


Figure 2

You can add more task 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 “Mark as Important”, simply add “!!!” after your event details . (Figure 3)

Date format: DD/MM/YYYY

Time format: HH:MM-HH:MM

The Program will automatically detect whether it is a calendar entry or a general entry.

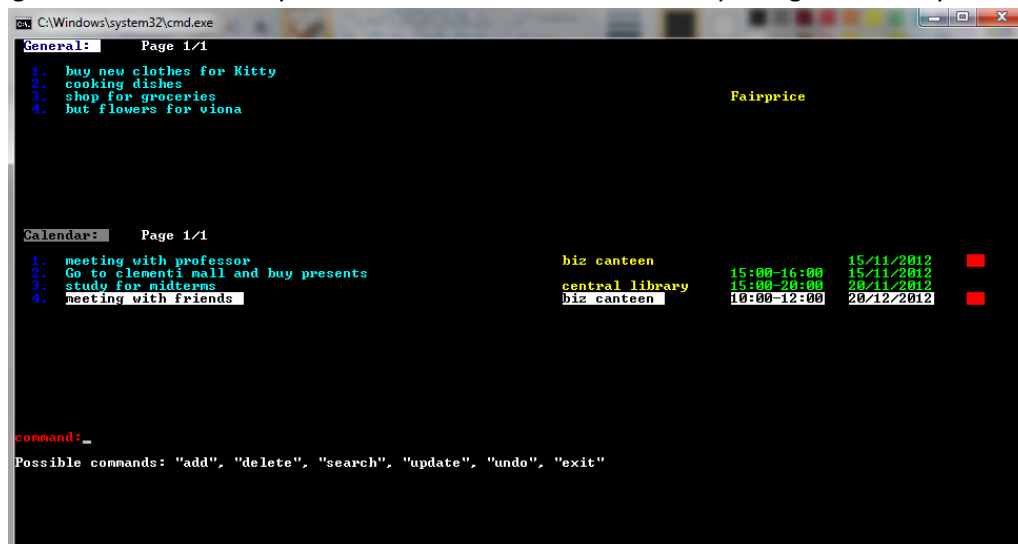


Figure 3

3. Updating an Entry

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

Command: [edit commands] [index] [updated entry]

Format: [DD/MM/YYYY] [Time] [Event Details]

Possible edit commands: update

The update can be a word or a part of the entry details, date, or time of the event (figure4). After you enter the command, YourDay will update your entry . To mark an entry as important simply add “mark” or “unmark”. (Figure 4 and 5)

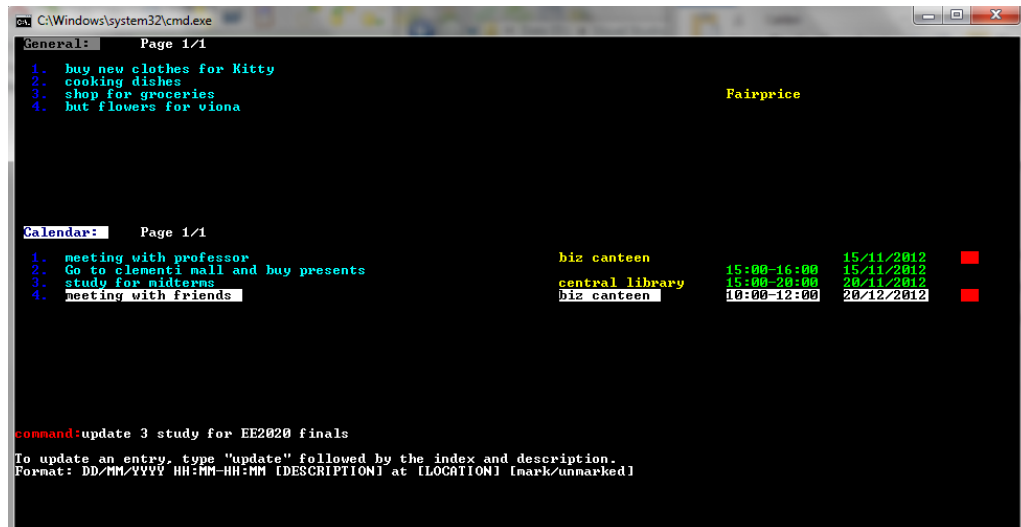


Figure 4

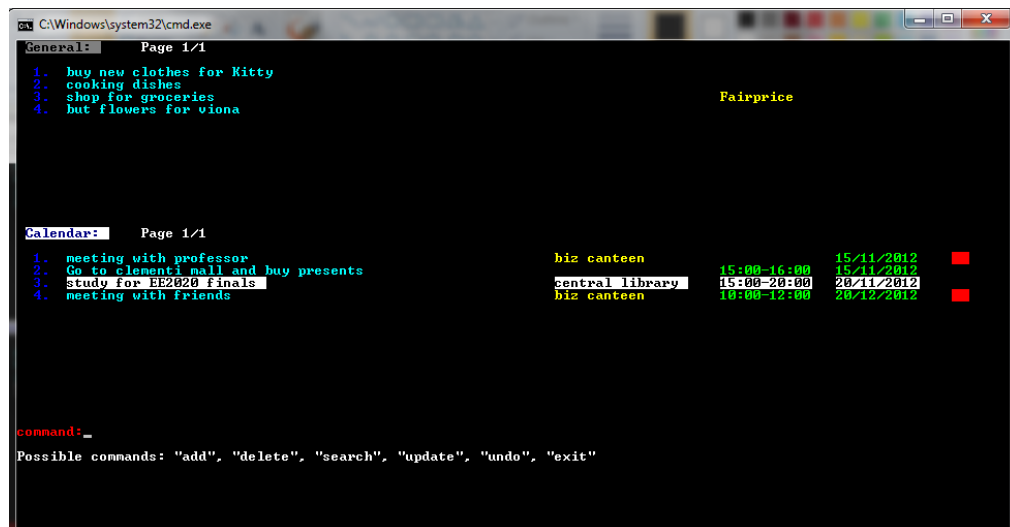


Figure 5

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 from the active focus field you want to delete by entering the entry ID shown on your search result. (Figure 6 and 7)

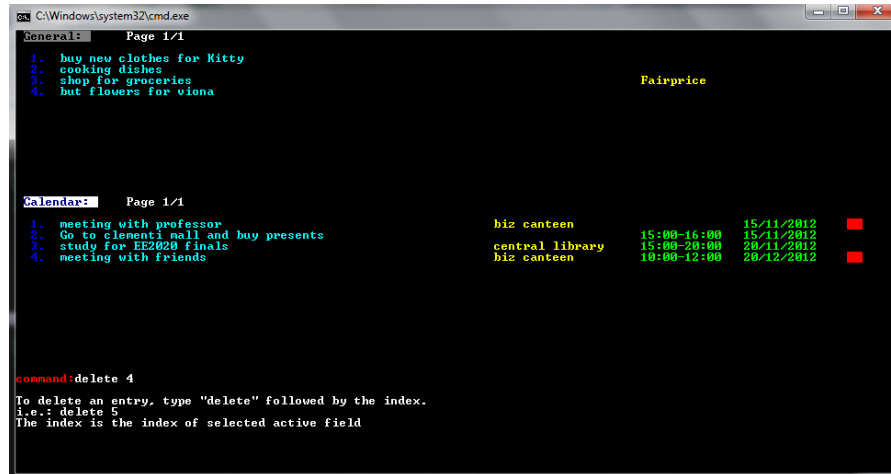


Figure 6

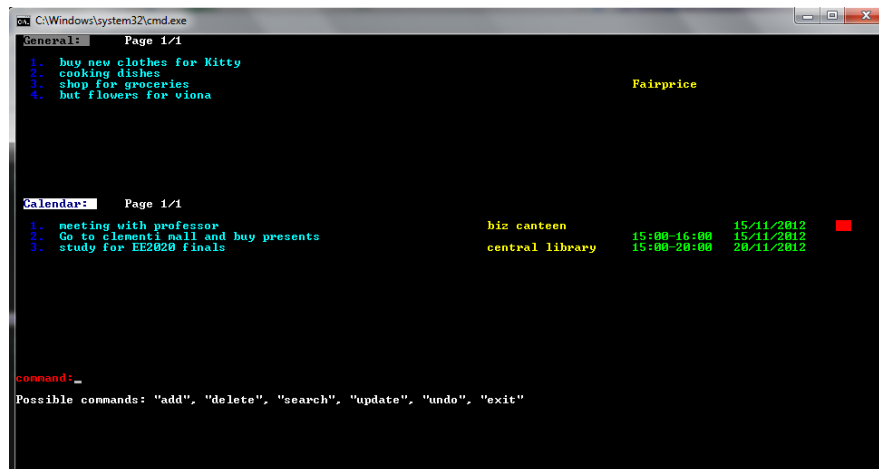


Figure 7

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

After you enter the command, YourDay will give you a list of entries that matches your keywords (Figure 9). 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 complete word. Our software will detect what you want to search and suggest a search word that you probably want to enter. If YourDay could not found your keyword from the database, it will give suggestions that are closely related to your keyword.

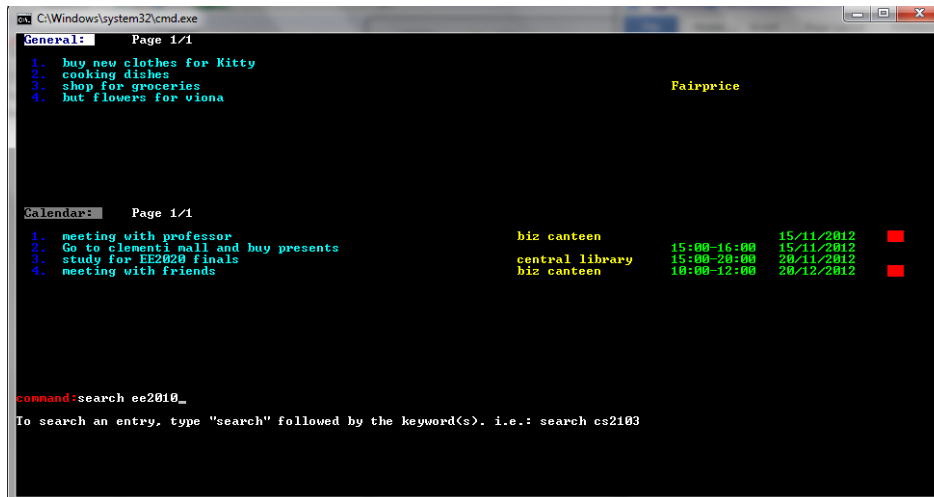


Figure 8

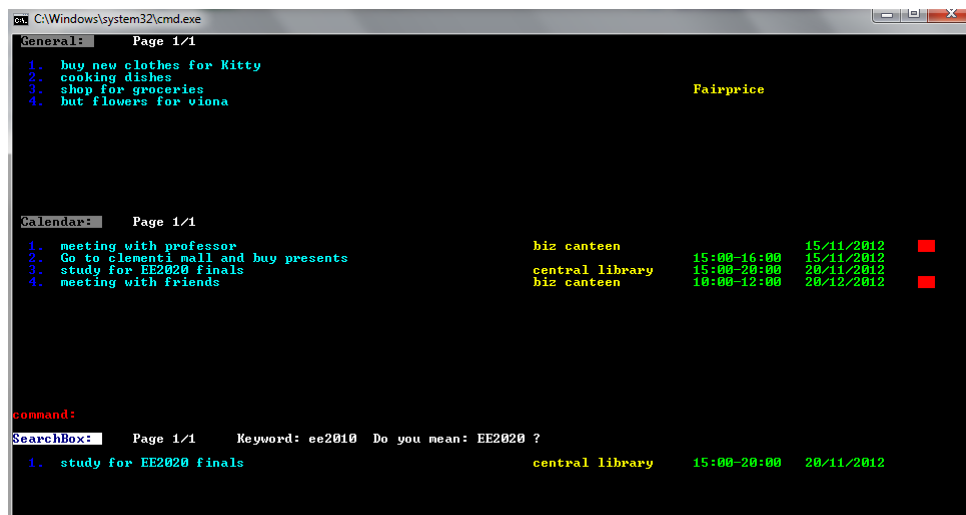


Figure 9

6. Undo Function

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

Command: undo

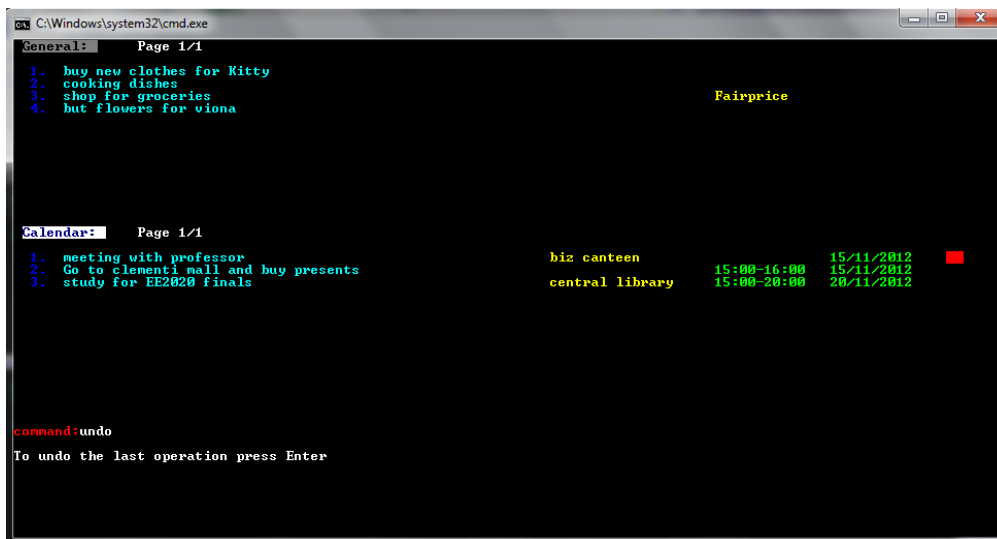


Figure 10

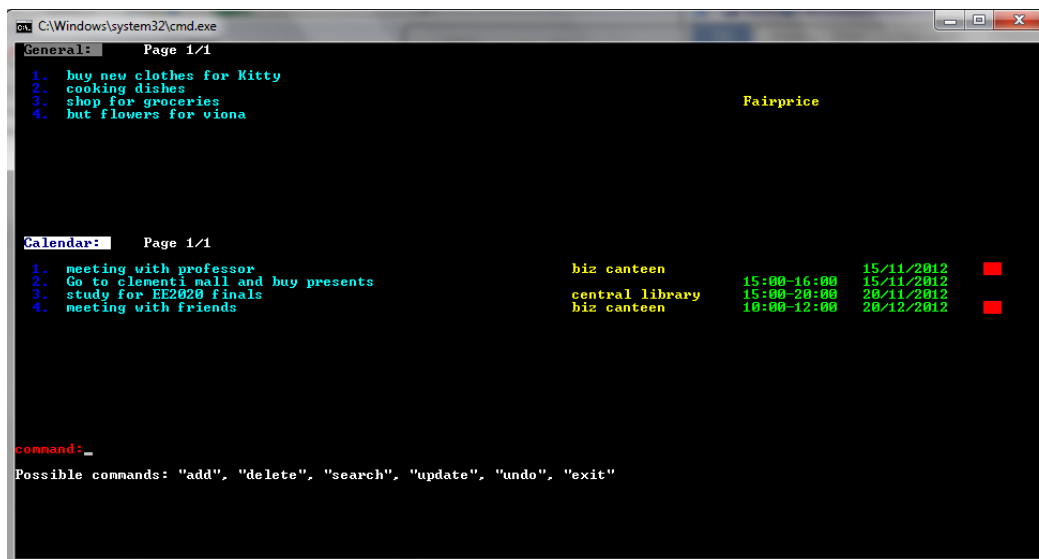


Figure 11

Why YourDay?

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

Glossary

Active Field: the current active/selected list.

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

Suggestions : a powerful search feature that suggests possible search words

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

Developer Guide

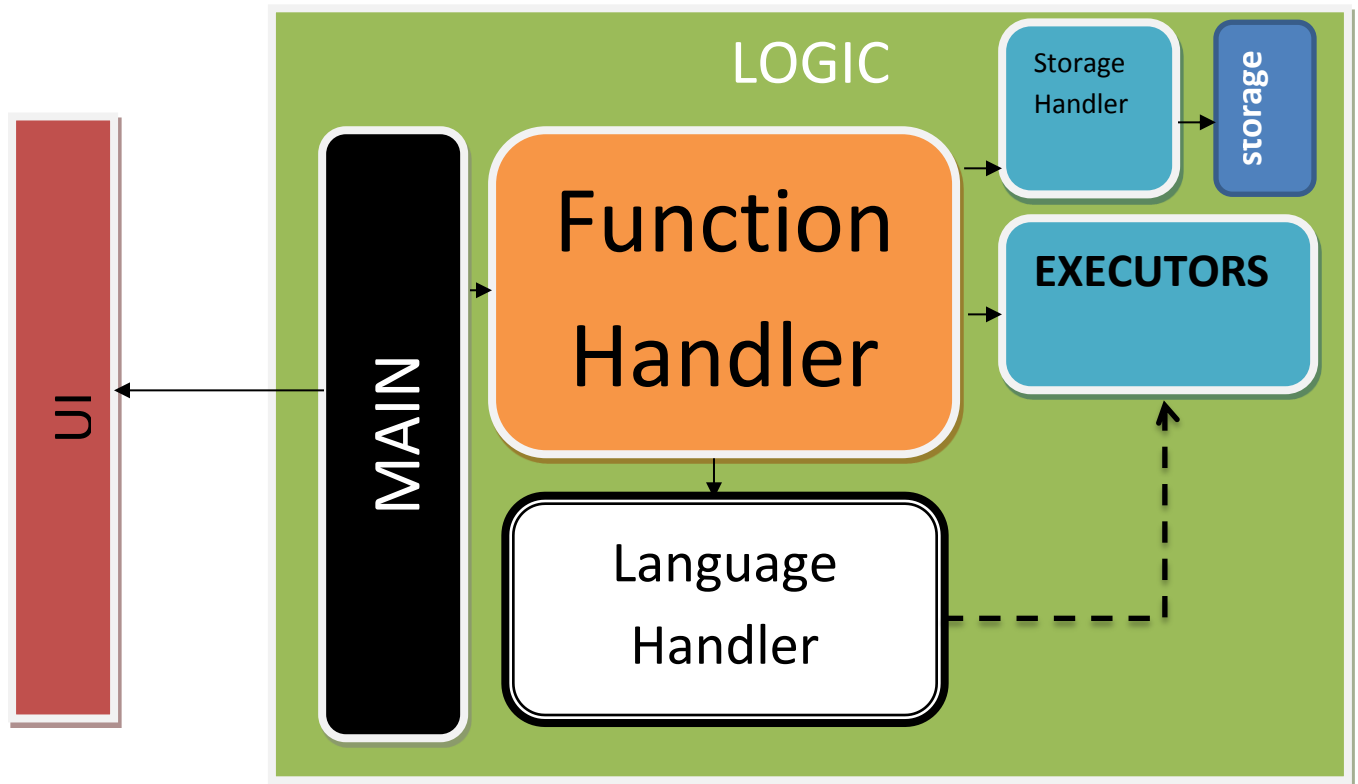
Purpose

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).

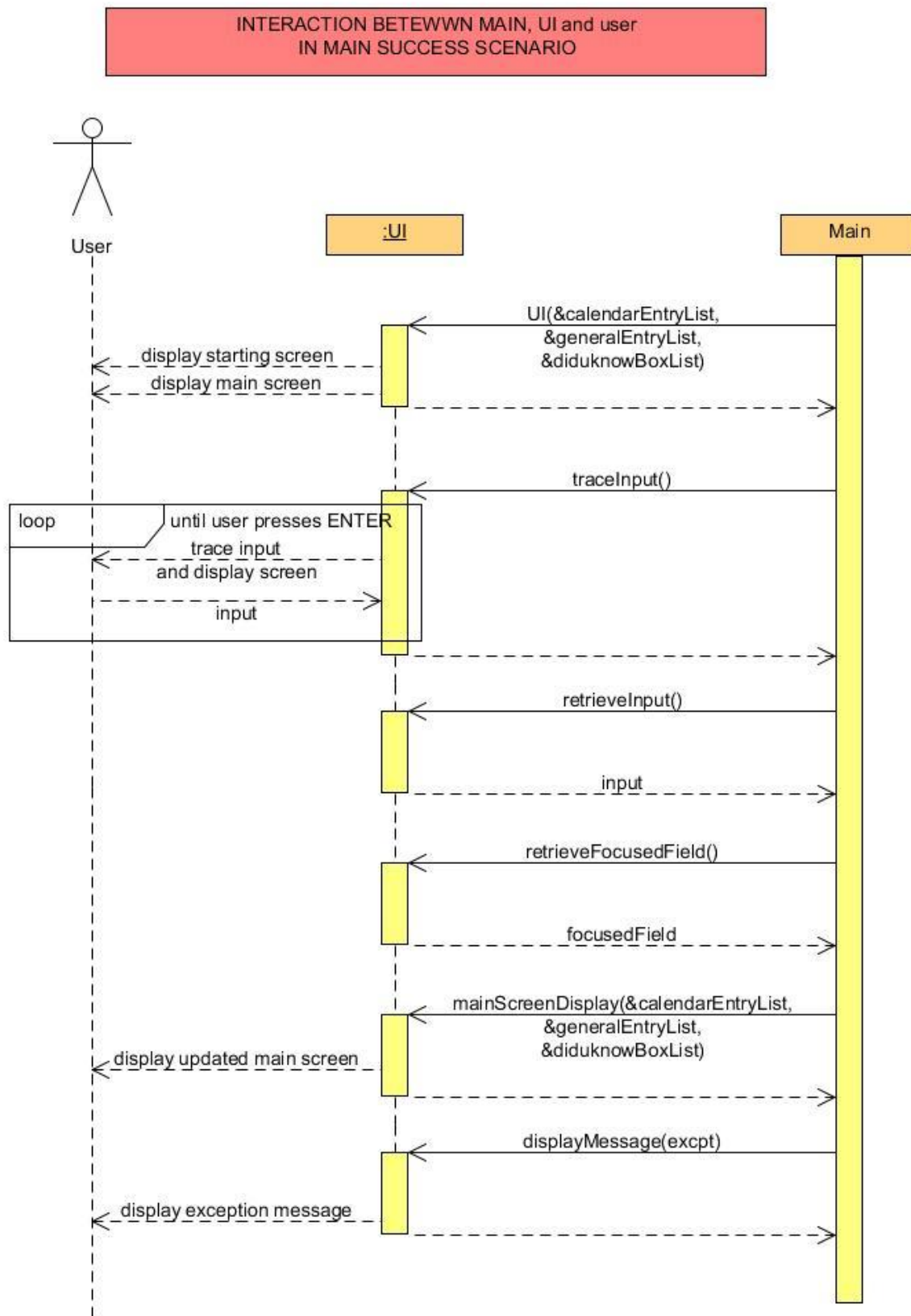
Transaction Architecture



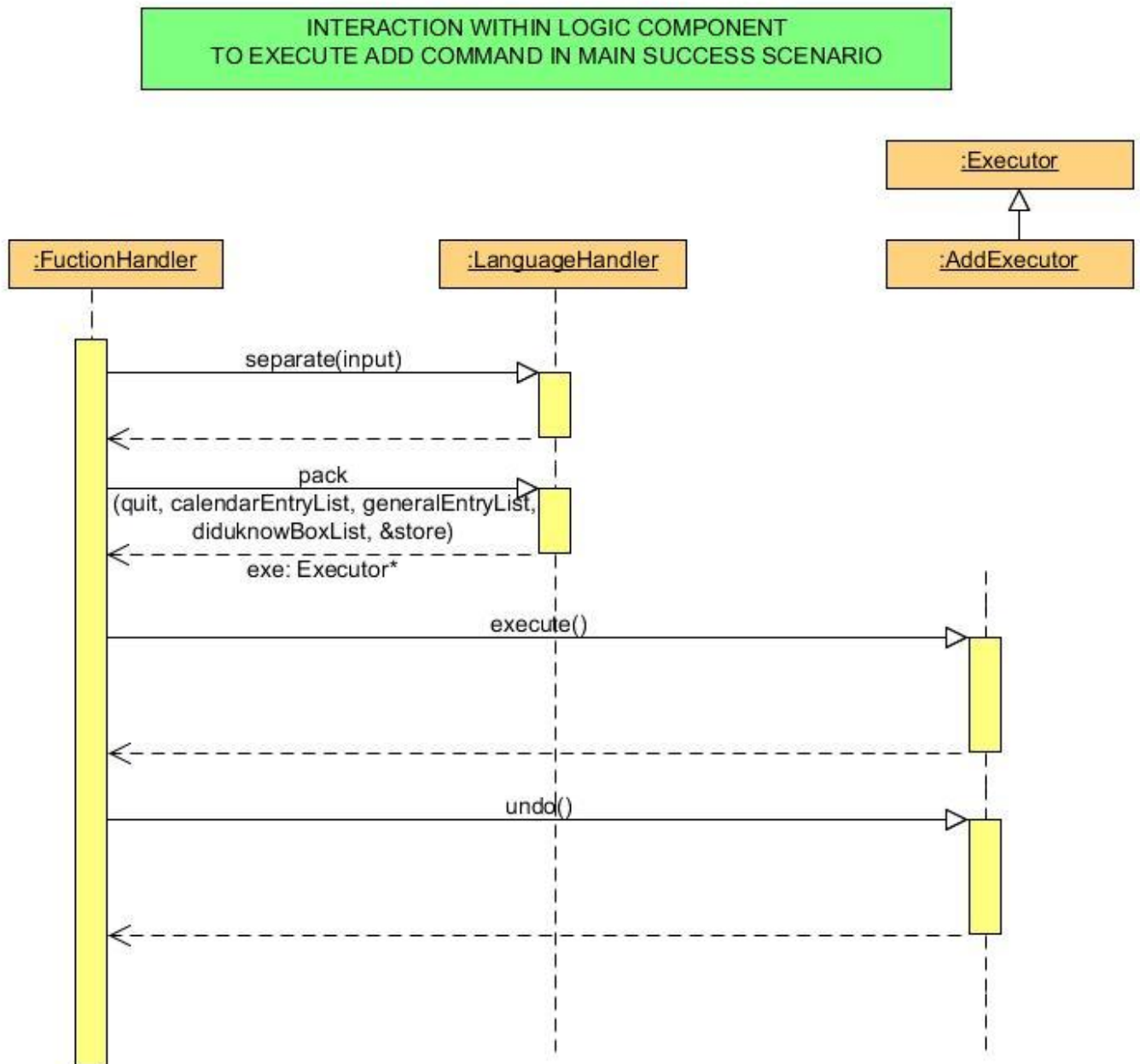
The next 3 pages contain UML Sequence Diagrams.

Sequence Diagrams:

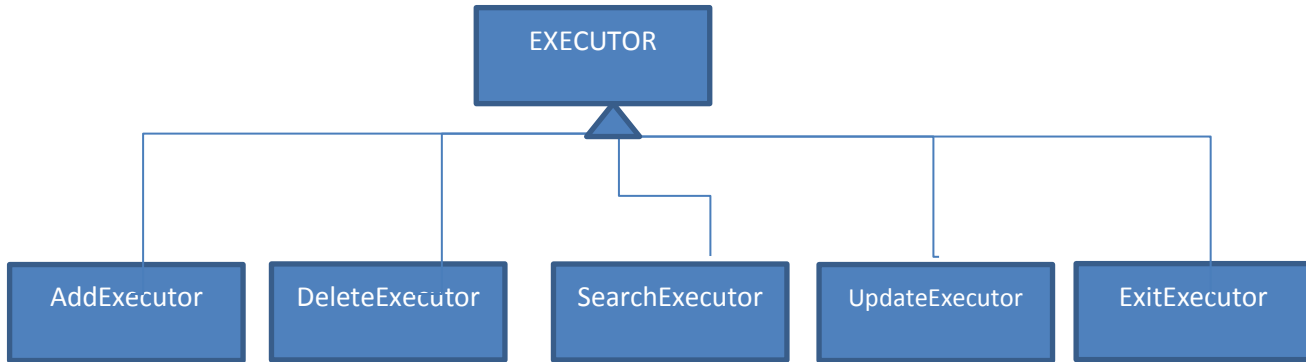
This is the sequence diagram to show interaction between main, UI and the user in main successful scenario



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



Inheritance Diagrams

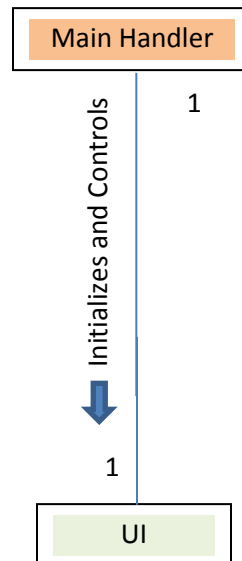


In this software, we use 1 super class which is the executor class. The main purpose of this class is to improve the simplicity of executing user's command and manage the tracking of the commands that has been executed. We have 5 inherited class from the main executor class, which are:

- AddExecutor
- DeleteExecutor
- SearchExecutor
- UpdateExecutor
- ExitExecutor

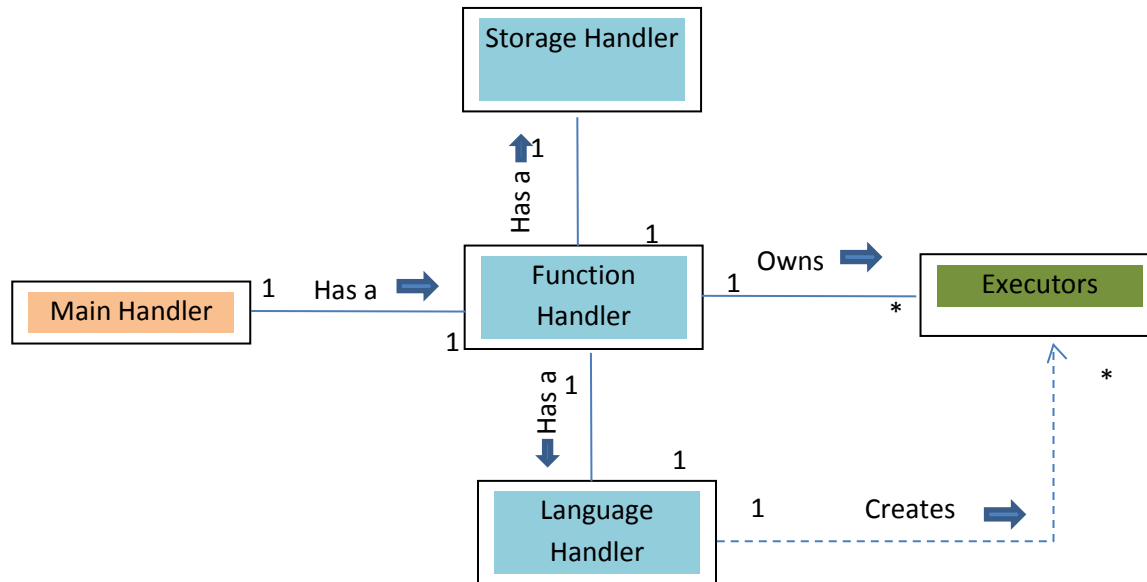
UI Association Diagrams

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.



<<enumeration>>

SIGNAL

DISPLAY, CLEAR, GENERAL, CALENDAR, SEARCH_RESULT, FULL_MODE, PART_MODE,
DIDUKNOW_INIT, DIDUKNOW_CLEAR, ADD_COMMAND, DELETE_COMMAND, UPDATE_COMMAND,
SEARCH_COMMAND, UNDO_COMMAND, EXIT_COMMAND, NULL_COMMAND

<<enumeration>>

OPEN_TYPE

APP, IN_TYPE, OUT_TYPE

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

UI CLASS

1. UI Class

UI

- int calendarPositionArray[NUMBER_OF_ENTRY_PARTS]

- int generalPositionArray[NUMBER_OF_ENTRY_PARTS]

- int colorArray[NUMBER_OF_ENTRY_PARTS]

HANDLE hConsole

- string input

- Signal focusedField

- Signal generalDisplayMode

- Signal calendarDisplayMode

- Signal resultDisplayMode

- Signal diduknowStatus

- Signal diduknowPrevStatus

- Signal prevCommand

- vector<string> calendarList;

- vector<string> generalList;

- vector<string> resultList

- vector<int> generalInitArrayPart

- vector<int> generalInitArrayFull

- vector<int> calendarInitArrayPart

- vector<int> calendarInitArrayFull

- vector<int> resultInitArrayPart

- vector<int> resultInitArrayFull

- int indexCurGeneralInitArray

- int indexCurCalendarInitArray

- int indexCurResultInitArray

- int highlightGeneralRowIndex;

- int highlightCalendarRowIndex;

- string searchKey;

- vector<string> searchSuggest

- int currentChar;

- bool isResultDisplay;

- bool isDiduknowDisplay

```
- void setScreenSize()

- void setBackground()

- void drawBanner();

- void gotoCommandBox();

- void drawCommandBox();

- void clearBox(int startH, int height);lay(vector<string>* calendarEntryList, vector<string>*
generalEntryList)

- void gotoxy(int x, int y);

- void writeTitle(string words, int startH, int startW);

- void writeHighlightedTitle(string words,int startH, int startW);

- void highlightTitle()

- string intToString(int number)

- void initializeDidUKnowStatus();

- void initializeInitArrayIndices();

- void initializeDisplayModes();

- void initializeFocusedField();

- void initializeHighlightIndices();

- void extractParts(string entry, string* partArray);

- int countPartLine(string part, int maxLength);

- bool isGeneral(string row);

- int findMaxInt(int* intArray, int size);

- int countLineOccupied(string entry, int* maxLengthArray);

- void setInitArrayPart(vector<string> entryList, vector<int>* ansArray, int boxHeight);

- void setInitArrayFull(vector<string> entryList, vector<int>* ansArray, int boxHeight,

- int* generalMaxLengthArray, int* calendarMaxLengthArray);

- void setMaxLengthArray(int* locationArray, int size, int* ansArray);

- void setInitialIndexArrays();void setDidUKnowStatus()

- void printLimitedLengthPart(string part, int maxLength, int initX, int initY, int& endPosition);

- void printEntryPartMode(int* positionArray, int* colorArray, string* partArray, int index, int
rowPosition);

- void printEntryFullMode(int* positionArray, int* colorArray, string* partArray, int index, int&
rowPosition);

- void printMark(string mark);

- void printCalendarEntry(int index, string row, int &rowPosition);

- void printGeneralEntry(int index, string row, int &rowPosition);
```

```

- void printResultEntry(int index, string row, int &rowPosition)
- void printDiduknowHints();
- void printGeneralFooter();
- void printCalendarFooter();
- void printResultFooter();
- void generalEntryListDisplay();
- void calendarEntryListDisplay();
- void resultListDisplay();
void diduknowHintDisplay();
- void lockResultDisplay();
- void processResultList(string& info);
- void handleResultInfo(string info);
- void processAddUpdateInfo(string info);
- void processSearchInfo(string info);
- void printSearchInfo();
- void handleInitialGeneralIndexOverflow();
- void handleInitialCalendarIndexOverflow();
- void handleInitialResultIndexOverflow();
- void handleInitialIndicesOverflow();
- void copyList(vector<string>* calendarEntryList, vector<string>* generalEntryList,
vector<string>* operationResultList);
- void startingScreenDisplay();
+ void mainScreenDisplay(vector<string>* calendarEntryList, vector<string>* generalEntryList,
vector<string>* operationResultList);
+ void traceInput();
+ string retrieveInput();
+ Signal retrieveFocusedField();
+void displayMessage(string message);

```

- a. void mainScreenDisplay(vector<string>* calendarEntryList, vector<string>* generalEntryList)
 - show the main screen to interact with the user
- b. void traceInput()
 - Traces the user's input
- c. Signal retrieveFocusedField()
 - Changes the focus field
- d. displayMessage(string message);

- Displays message

EXECUTOR CLASSES

1. Executor Class (Super Class)

Executor

```
# bool undoEnable
# Log log

# 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)
# int extractIndexFromDescription(string description)

+ void execute() (virtual)
+ void undo() (virtual)
```

- 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. Bool isUndoAble()
 - Returns whether the executor can be undone
 - Parameter= void
 - Return = boolean condition of undoEnable
- ### 2. AddExecutor Class

AddExecutor


```

# bool undoEnable
# Log log
- vector<string>* _calendarEntryList
- vector<string>* _generalEntryList;
- vector<string>* _resultList;
- vector<string> _undoGeneralEntryList
- vector<string> _undoCalendarEntryList
- string _details

- string manipulateEntryDate(string entry)
- bool isEarlier(string &entry1, string &entry2)
- int searchIndex(vector<string>* _entryList, string input)
- int binarySearch(vector<string>* _entryList, string key, int imin, int imax)
- void shiftList(vector<string>* _entryList,int index)
- void addToPosition(vector<string>* _entryList, int position, string input)
# 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()
+ bool isUndoAble

```

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. Bool isUndoAble()

- Returns whether the executor can be undone
 - Parameter= void
 - Return = boolean condition of undoEnable

3. DeleteExecutor Class

DeleteExecutor

```
# bool undoEnable
# Log log
- vector<string>* _focusingEntryList;
- vector<string> _undoFocusingEntryList;
- 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()
+ bool isUndoAble()
```

- void execute()
 - executes the delete entry functionality from storage
 - the entry can be deleted from the active field Entry List depending on the index chosen
 - Parameter = void
 - Return =void
- void undo()
 - Undo the last changes made by the executor
 - Parameter = void
 - Return =void
- Bool isUndoAble()
 - Returns whether the executor can be undone

- Parameter= void
- Return = boolean condition of undoEnable

4. ExitExecutor Class

ExitExecutor

```
# bool undoEnable
# Log log
- vector<string>* _generalEntryList
- vector<string>* _calendarEntryList
- StorageHandler* _store;
- bool* _quit

# 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()
+ bool isUndoAble
```

- a. void execute()
 - signals the storage handler to save the data and exits the program
 - Parameter = void
 - Return =void
- b. Bool isUndoAble()
 - Returns whether the executor can be undone
 - Parameter= void
 - Return = boolean condition of undoEnable

5. SearchExector Class

SearchExecutor

```

# bool undoEnable
# Log log
- vector<string>* _calendarEntryList
- vector<string>* _generalEntryList
- vector<string>* _matchedEntryList
- vector<string> _undoGeneralEntryList
- vector<string> _undoCalendarEntryList
- vector<string> _undoMatchedEntryList
- vector<int> rank
- vector<int> score
- vector<string> _combinedEntryList
- int highestRank
- int totalEntries
- string _details
- int f[MAX_ENTRY][MAX_ENTRY]
- int g[MAX_ENTRY][MAX_ENTRY]
- bool noMatch;
- int treshold;

# 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)
- bool isDate(string date)
- bool isTime(string time)
- bool isLeap(int year)
- bool isLogicDate(string date)
- bool isLogicTime(string time)
- bool isInTimeRange(string time)
- int extractDay(string date)
- int extractMonth(string date)
- int extractYear(string date)
- int extractHour(string time)

```

```

- int extractMinute(string time)
- void splitStartEndTime(string* start, string* end, string timeRange)
- void initializeScore()
- void initializeRank()
- void initializeCombinedEntry()
- void splitWords(string encodedInput, vector<string>* list)
- void calInfo(int i, int j, string a, string b, string & x, string & y, matchInfo & t)
- int notsame(char a, char b)
- void edit(string a, string b, matchInfo & ans)
- static bool cmp(matchInfo a, matchInfo b)
- matchInfo compare(matchInfo a, matchInfo b)
- void updateSuggestWords(string* suggestWords, string updWord)
- bool unrelavent(matchInfo info, string key)
- void examRelavence(vector<matchInfo>* list, vector<int>* rank, string key);
- void encodeIndex(string* encodedEntry, int index)
- void searchTime(string keyword)
- void searchText(string key, vector<int>* rank, vector<string>* suggestWords);
- void initializeRank(int totalSize, vector<int>* rank)
- void initializeCombinedEntry()
- void setRank(int index, int level, vector<int>* rank, int* currentHighest)
- void adjustRank(vector<int>* rank, int currentHighest)
-void calInfo(int i, int j, string a, string b, string & x, string & y, matchInfo & t)
- int notsame(char a, char b)
- void edit(string a, string b, matchInfo & ans)
- static bool cmp(matchInfo a, matchInfo b)
- matchInfo compare(matchInfo a, matchInfo b)
- void searchData(string keyword, vector<int>* rank)
- void searchTime(string keyword, vector<int>* rank)
- void searchText(string keyword, vector<int>* rank)
+ 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. Bool isUndoAble()
 - Returns whether the executor can be undone
 - Parameter= void
 - Return = boolean condition of undoEnable

6. UpdateExecutor Class

UpdateExecutor

```
# bool undoEnable
# Log log
- vector<string>* _focusingEntryList;
- vector<string>* _generalEntryList;
- vector<string>* _calendarEntryList;
- vector<string>* _resultList;
- vector<string> _undoGeneralEntryList
- vector<string> _undoCalendarEntryList
- Signal _focusingField;
- string _encodedUserInput;

# 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)
- bool isIndexValid(int index)
- bool verifyTheFiledChange(string newEntry)
```

```

- int extractNewIndex(string newEntry)
- void addNewEntryToRightPosition(string newEntry,int index, int newIndex, bool
changeToCalendar)
- string constructNewEntry(string oldEntry)
+ void execute()
+ void undo()
+ bool isUndoAble();

```

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. Bool isUndoAble()

- Returns whether the executor can be undone
 - Parameter= void
 - Return = boolean condition of undoEnable

HANDLER CLASSES

1. Function Handler

FunctionHandler

```

- Log log
- StorageHandler store
- stack<Executor*> undoStk

```

```
+ void execute(string input, bool* quit, Signal focusingField,
               vector<string>* generalEntryList,
               vector<string>* calendarEntryList,
               vector<string>* resultList)
```

- a. void execute(string input, bool* quit, Signal focusingField, vector<string>* generalEntryList, vector<string>* calendarEntryList, vector<string>* resultList)
 - 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>* resultList is the vector that contains hints and general feedbacks in using the programs.

2. Return =void

2. Language Handler

LanguageHandler

```
- Log log
- Signal command;
- string details;

- bool isLeap(int year)
- bool isDate(string date)
- bool isTime(string time)
- bool isInt(string inx)
- bool isLogicDate(string date)
- bool isLogicTime(string time)
- string getSuffix(string str, int pos)
- string getPrefix(string str, int pos)
- void splitPriority(string* str, string* priority, string indicator,
```



```

char token)
- void splitLocation(string* str, string* location- void
splitLocation(string* str, string* location)
- void setCommand(string userCommand)
- void splitDate(string* str, string* date)
- void splitTime(string* str, string* time, string* date, bool autoFill)
- void splitDescription(string* str, string* description, bool empty)
- void fillUpDate(string* date)
- void regulateDate(string* date)
- void regulateTime(string* time)
- void regulateDescription(string* description)
- void eliminateSpaces(string* str)
- void encoder(string input, Signal command)
+ void separate(string userInput)
+ Executor* pack(bool* quit, Signal focusingField,
                vector<string>* generalEntryList,
                vector<string>* calendarEntryList,
                vector<string>* resultList,
                StorageHandler* store);

```

- a. 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
- b. void encoder(string input, Signal command) throw (string)
 - 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>* resultList is the vector that contains hints and general feedbacks in using the programs.
 - f. StorageHandler* store is Storage Handler to save the files.

2. Return = void

3. Storage Handler

StorageHandler

```
- char buffer[MAXMIUM_WORDS]
```

```
- 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);
```

```
+ void readData(vector<string> *ram)
```

```
+ void writeData(vector<string> *ram)
```

a. 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

b. 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

Future Developments

- Natural language processing
- Graphical
- Further implementation in switching DidUKnow Hints box and search result box.

APPENDIX I

TESTING

1. Automated Unit Test

Our Unit test is focusing on the functionality verification which is implemented by gtest, the scope of our Unit test's targets include add, delete, update, search, and undo operations, all of them are tested by examining the Calendar entry list and general entry list. In the Unit test, all the system requirements are given to ensure those functions work in a "safe environment", so only the functionality are tested in the Unit Test.

2. Systematic Test

Being contrast to Unit Test, the systematic test is against the system specification and we take a whole system to test instead of a part of the system.

a) Performance testing

- 1) When the user input a 50-characters-long command, how long user will wait for the software responds, and is there any bug?
- 2) When the user input a 255-characters-long command, how long user will wait for the software responds, and is there any bug?
- 3) When the user input a 500-characters-long command, how long user will wait for the software responds, and is there any bug?
- 4) When there are 10 entries in the storage file, how long we need wait for the software responds?
- 5) When there are 100 entries in the storage file, how long we need wait for the software responds?
- 6) When there are 1000 entries in the storage file, how long we need wait for the software responds?
- 7) When there are 10000 entries in the storage file, how long we need wait for the software responds?

b) Load Testing

- 1) When there are 10 entries in the storage file, is it loadable?
- 2) When there are 100 entries in the storage file, is it loadable?
- 3) When there are 1000 entries in the storage file, is it loadable?
- 4) When there are 10000 entries in the storage file, is it loadable?

c) Security Testing

1) Wrong-format command Testing

1. Random strings eg. "xxxxx"
2. The command keyword correct but with random string following.
Eg.1) Add xxxx 2) delete xxxx 3) search xxxx 4) undo xxxx 7) update xxxx
3. The command correct, but details following is incomplete.(only

applicable for calendar entry as the general list would take all things as entry description)

Eg. add 10:00-01:00 (data missing) have lunch at soc priority high

4. Both of command and details are correct.

2) Sensitive number/words Testing

As our codes get some sensitive words lists:

For instance, '#', 'at' and 'priority'. When these three words are included in the command, it will be errors.

3) Modify the storage files.

a) Remove the storage files.

b) Modify the content inside the storage files.

d) Portability Testing

1) Run in Windows 7

2) Run in Windows XP, NT, Vista.

e) Usability Testing

1) Get the feedback for our software from different knowledge-background people

f) Compatibility testing

As our YourDay is independent from other software, we do not need to test this.

Note: All the test cases can be found under /testing/System Test Case/, but when you are using them to test, please put them under /source/YourDay/ directory and change their name "YourDayCEntry-XXX.txt" to "YourDayCEntry.txt" and "YourDayGEntry-XXX.txt" to "YourDayGEntry.txt".

3. User Acceptance Testing

: The testing will be against the requirements specification, not system specification.

For the Acceptance test, we launch it after the System testing, here are requirements for our final project, and we need to test whether the system does what it is intended to do, which means meets the requirements.