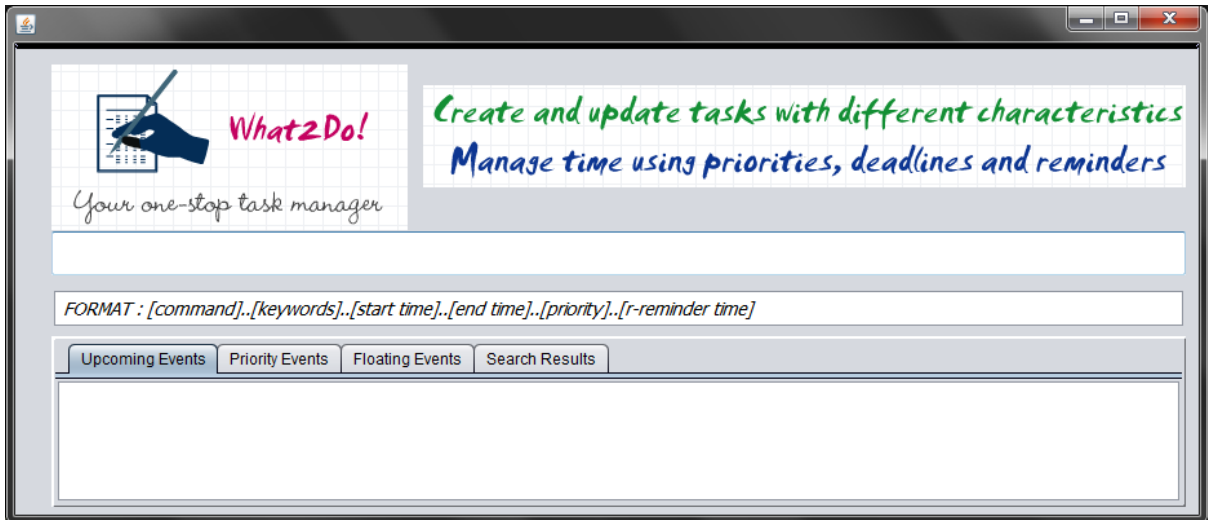






What2Do!



Members:

			
Akaash Gupta	Sandeep Paul	Le Khac Minh Tri	Anirup Sengupta
Team Lead, Logic	GUI	Database and Time	Executor Class

CREDITS

- We have used the free web-application, LogoMaker to help design our software logo.
- We have also used the Jodatetime library for handling Dates and Times.

USER GUIDE

What2Do! is a text based to-do manager which has several features to make it very easy for the user (i.e. you) to manage their schedule.

1. ADDING AN EVENT

You can add an event using the command line interface at the top (i.e. the bar at the top) by typing in the following format -

Format: [add]..[Key words]..[start-time and date]..[End time and date]..[Priority]..[r-reminder time]

Key Words:

You can chose to insert a hash-tag. For example, if the entry is:

add..meeting prof. damith #cs2103..monday 10/9

The task name would be meeting prof. damith and this task would be under the cs2103 category.

The hash tags are **optional**.

Optional Components:

*Time and Date: If left blank, then classify the task as floating task.

*Priority: In case this field is left blank, the system shall assume normal priority (h/high/l/low)

*Reminder Time: In case left blank, then the system shall not have a reminder for that task.

Examples:

add..meeting with cs2103 teammates..Sunday 9/9 11 am..r-2hours

add..study for cs2103 final exam

add..project submission deadline..26/10..high..r-10days

add..complete tutorial homework..Sunday 16/9..r-1day

2. SEARCHING

You can search for any event using keywords or tags, regardless of whether it is completed or incomplete by typing the following command in the command-line.

The command would be:

[search]..[searchwords]

You can also include hash tags in your search – for example –

search..abcd # abc

This would result in a point – wise display of all the entries matching the search words. The search results are sorted in order of priority and then by date.

Examples: search meeting Results:

1. Meeting with Prof. Damith on Monday 2012/02/01 11:00 #HIGH (**High priority so it would appear in red**)

2. Meeting with CS2103 Teammates 2012/09/09 13:00

3. Meeting with girlfriend 2012/09/11 20:00 #HIGH (**High priority**)

Thereafter, you can update or deleted these entries using the following commands: update and delete and referring to the search result option.

Example:

- delete..2

This would result in the entry pertaining to meeting with cs2103 teammates being cancelled or deleted from the database

- update..3

The original entry would appear and you would be free to edit the information. In case of tasks that are completed, you can add a done at the end, and this shall result in the task being archived as finished.

3. UNDOING OPERATIONS

You can undo the last made action/operation by either typing the command undo or by clicking on the undo button in the top-right corner of the interface. The undo keyword, when entered in the CLI, cancels the previous action and the system reverts to the previous state.

4. ENTERING A COMMAND IN DIFFERENT FORMATS

You shall have the option to use shorthand notations, if you wish to do so. This will make your work easier and faster. **Examples:**

- add: + OR a
- delete: - OR d
- search: s
- update: u

ADDITIONAL USER GUIDES:

[These features are yet to be implemented but shall be done so in subsequent versions of the project]

1. How to access Help and Troubleshooting -

You can seek help using the help button. The Help button is available on the top right hand corner of the GUI. Clicking on this button shall link to a new side window opening. This window shall have detailed information pertaining to all the features of our system, as well as instructions on how to use them. Also, provided will be troubleshooting for the software.

2. Suggestions box listing all the possible commands that you can make

Right below the CLI, there shall be a “Suggestion Box” which displays the format of the user entry. (Example: [available operations]..[keywords]..[date and time]..[priority]..[r-reminder time]

3. How to use the reminder command - Alarm with snooze intervals -

While creating a task, you have the option of setting a reminder, which would be in the form of a audio alarm(if the system is running at that time) or a textual reminder. You have the option of setting the time for the reminder – 20 minutes before deadline, 1 day before the meeting etc. See “ADDING AN EVENT” for command syntax example.

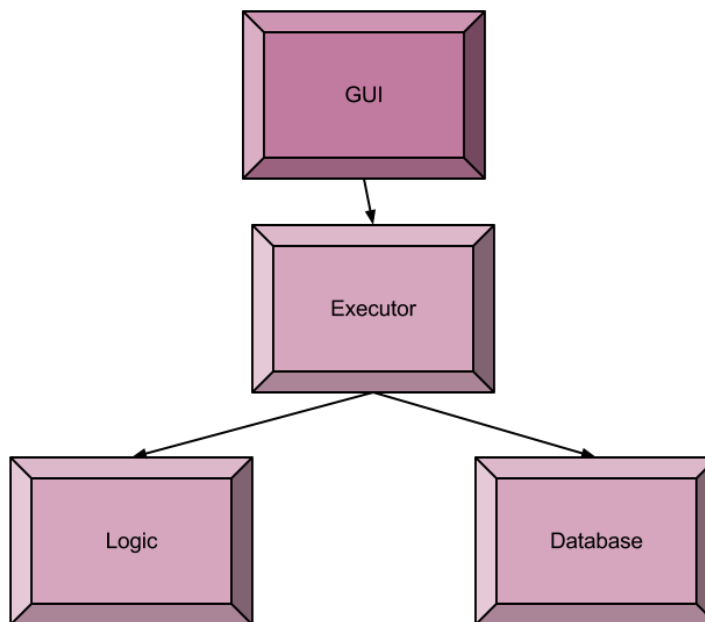
4. Priority based colouring – how to set priority for events -

At the time of task creation, you shall have the option to set the priority to high (h/H), low(l/L) or normal(n/N)(default). High priority tasks shall be displayed in RED, and low priority ones in YELLOW.

Developer's Guide

- **Architecture:** N-tier Architecture –

For our project, we chose a N-tier architecture with a top-down approach. So, we started building all the components individually and integrated them continuously as we developed the parts individually. Our Architecture can be demonstrated by this diagram –

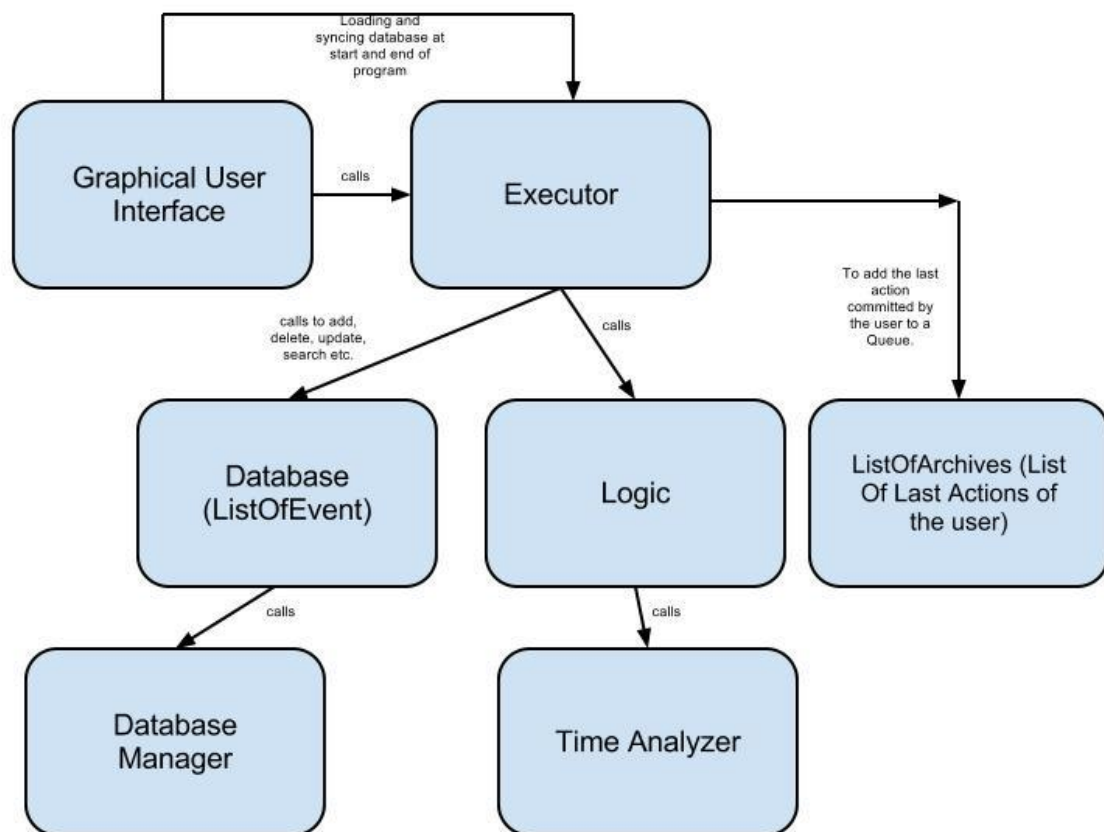


- **Design Description:**

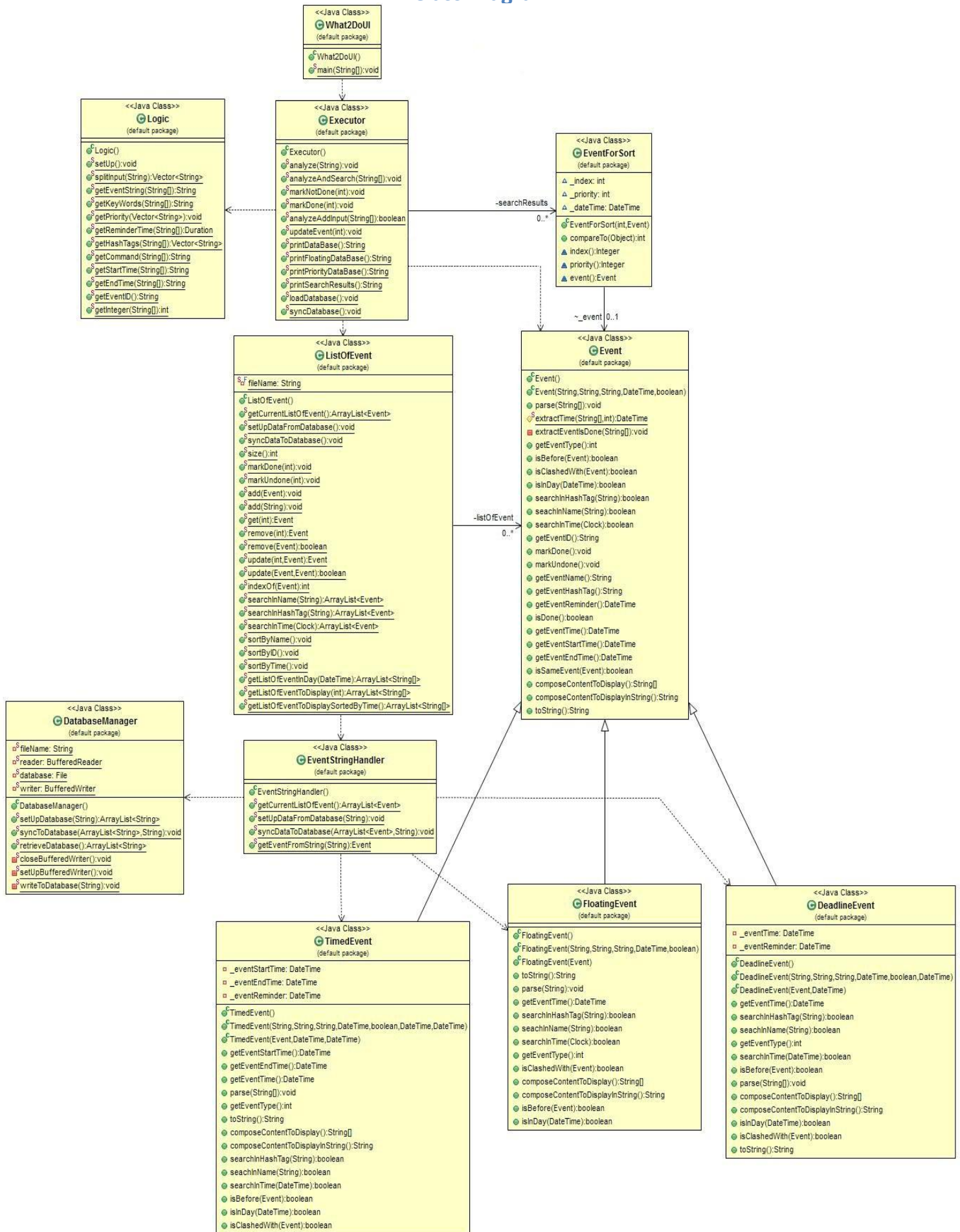
- So, the GUI class is the interface between the user and system. How the GUI works is explained in the next section.
- The GUI first loads the data from the local database at the start of the program by calling the relevant function in the executor class.
- The GUI then calls functions in Executor Class to analyse the user Input. The analyse function in the executor class is called by the GUI. This class then splits the user Input according to the Splitter used (“..”) and gets the command in the user input. It

then decides which function to call according to the parsed command. For example – analyzeAddInput for add command.

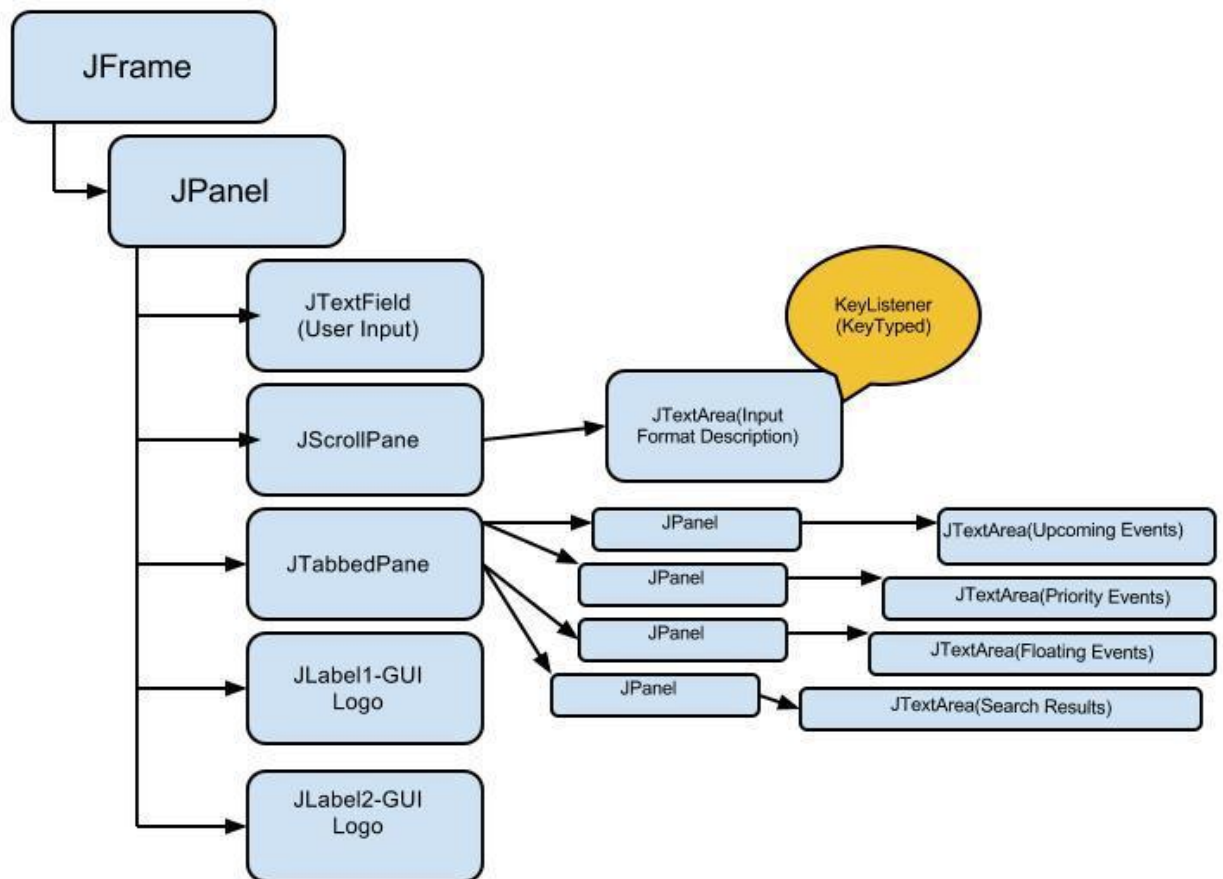
- After this, the functions then get the required details from the user input by calling the various parser functions in the Logic class and subsequently, perform the functions on the static ListOfEvent class, which is the temporary database while the program is executing.
- These functions, after performing the given command from the user input also add the last performed task to an Archive of Actions by the User which is used to undo operations. After the control returns to the GUI, it updates all the text fields with the new Databases (See next section).
- At the end of the execution of program – the GUI syncs the temporary database to the local database by calling a function in the Executor class.



Class Diagram



- **GUI Description:**



- The JTextField contains a KeyListener-it “listens” for the keyword “\n” or “Enter” which signifies that the user has completed his input.
- The GUI then sends the user input to the Executor class, which analyses, parses the input and updates the database.
- These databases values are then returned to the GUI and are displayed in the three JTextAreas under the JTabbedPane-one each for Upcoming, Priority and Floating Events. There is another such JTabbedPane for Search Results to be displayed(the search results are obtained from the Executor class)

- The GUI has been developed as a pure keyboard version of Siri. Therefore, we have deliberately omitted buttons for mouse clicks(ex. “Done”, “Exit”). Instead these actions are performed by keyboard commands (“\n” and “exit” respectively)
- The four tabs(Upcoming, Priority and Floating Events and Search Results) can be accessed by the keyboard as well. The shortcut for each of these tabs is Alt+1, Alt+2, Alt+3 and Alt+4 respectively.
- The GUI also has a JTable which has not been utilized (yet).
- Code Example for displaying data obtained from Executor Class:

```
private void textField1KeyTyped(java.awt.event.KeyEvent evt) throws
IOException { // GEN-FIRST:event_textField1KeyTyped
    jTabbedPane3.setMnemonicAt(0, evt.VK_1);
    jTabbedPane3.setMnemonicAt(1, evt.VK_2);
    jTabbedPane3.setMnemonicAt(2, evt.VK_3);
    if (evt.getKeyChar() == '\n') {
        String data = textField1.getText();
        if (data.equals("exit") == true) {
            System.exit(0);
        }
        Executor.analyze(data);
        String upcomingEvents=Executor.printDataBase();
        String priorityEvents=Executor.printPriorityDataBase();
        String floatingEvents=Executor.printFloatingDataBase();
        //String searchResults=Executor.printSearchResults();
        upcomingEvents=format(upcomingEvents);
        priorityEvents=format(priorityEvents);
        floatingEvents=format(floatingEvents);

        jTextArea1.setText(upcomingEvents);
        jTextArea2.setText(priorityEvents);
        jTextArea3.setText(floatingEvents);
        //jTextArea4.setText(searchResults);
        textField1.setText("");
    }
}
```

APPENDIX

Important APIs: Listed below are the important APIs of all the classes in a tabular format and what classes they are called by –

1) Logic –

<u>Function Name -</u>	<u>What It Does -</u>	<u>Called By -</u>
<u>getPriority()</u>	Gets the priority specified in the user input	Executor
<u>getKeyWords()</u>	Returns the keywords	Executor
<u>getReminderTime()</u>	Parses the reminder time and returns the number of milliseconds.	Executor
<u>getStartTime()</u>	Returns the start Time as a String	Executor
<u>getEndTime()</u>	Returns the End Time as a String	Executor
<u>getInteger()</u>	Gets the Integer from the delete Command	Executor
<u>getCommand()</u>	Returns the type of command (add, delete...)	Executor
<u>getHashTags()</u>	Returns the List of Hash Tags as a Vector	Executor
<u>getEventID()</u>	This function returns a unique ID for an event by converting the current time to a string of numbers.	Executor

2) Executor -

<u>Function Name -</u>	<u>What It Does -</u>	<u>Called By -</u>
<u>analyze()</u>	Analyzes the Input String to get the Command and calls other functions accordingly.	GUI
undoLast()	Undoes the last Action by the User.	Executor.analyze()
analyzeAndSearch()	Analyzes the search Input and performs the search.	Executor.analyze()
analyzeAddInput()	Analyzes the Add parameters and adds to database.	Executor.analyze()
markDone()	Marks a task as Done.	Executor.analyze()
markNotDone()	Marks a task as Undone.	Executor.analyze()
updateEvent()	Updates an event.	Executor.analyze()
printDatabase()	Returns a Formatted String with all the non-floating events in the Database.	GUI
printFloatingDatabase()	Returns a Formatted String with all the floating events in the Database.	GUI
printPriorityDatabase()	Returns a Formatted String with all the non-floating events in the Database sorted in a high to low priority order.	GUI
printSearchResults()	Returns a Formatted String with the search results.	GUI

3) **ListOfEvent** – This is the Database Class.

<u>Function Name -</u>	<u>What It Does -</u>	<u>Called By -</u>
syncDataToDatabase()	Writes all data to local database.	GUI, Executor
getCurrentListOfEvent()	Returns the entire Database as a Linked List	Executor
size()	Returns the Size	Executor
markDone()	Marks an event as Done	Executor
markUndone()	Marks an event as Undone	Executor
remove()	Removes an event from the database.	Executor
add()	Adds an event to the database.	Executor
setUpDataFromDatabase()	Loads the events into the program from the local database.	GUI
get()	Returns an event at a particular index.	Executor

4) **ListOfArchive** – This is the List of last actions performed by the User.

<u>Function Name -</u>	<u>What It Does -</u>	<u>Called By -</u>
add()	Adds the last action performed by the user to the	Executor
undo()	Undo's the last action performed by the User.	Executor