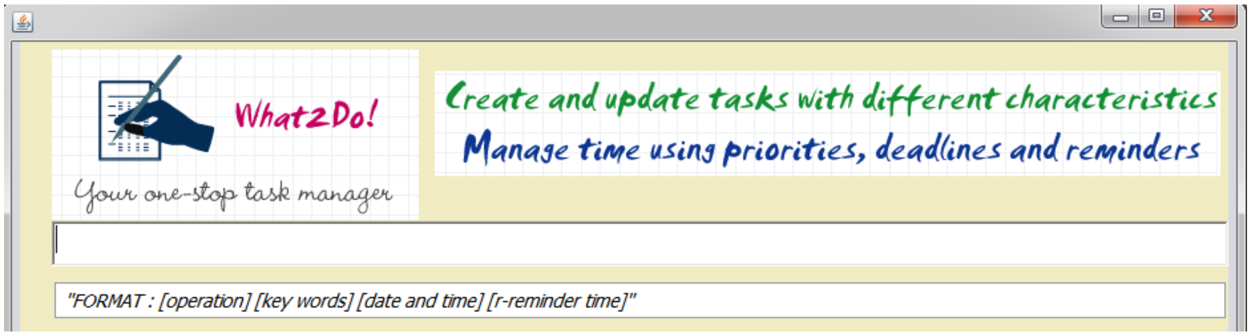
What2Do!

Your one-stop task manager


What2Do!
Your one-stop task manager
Create and update tasks with different characteristics
Manage time using priorities, deadlines and reminders

"FORMAT : [operation] [key words] [date and time] [r-reminder time]"

Members:

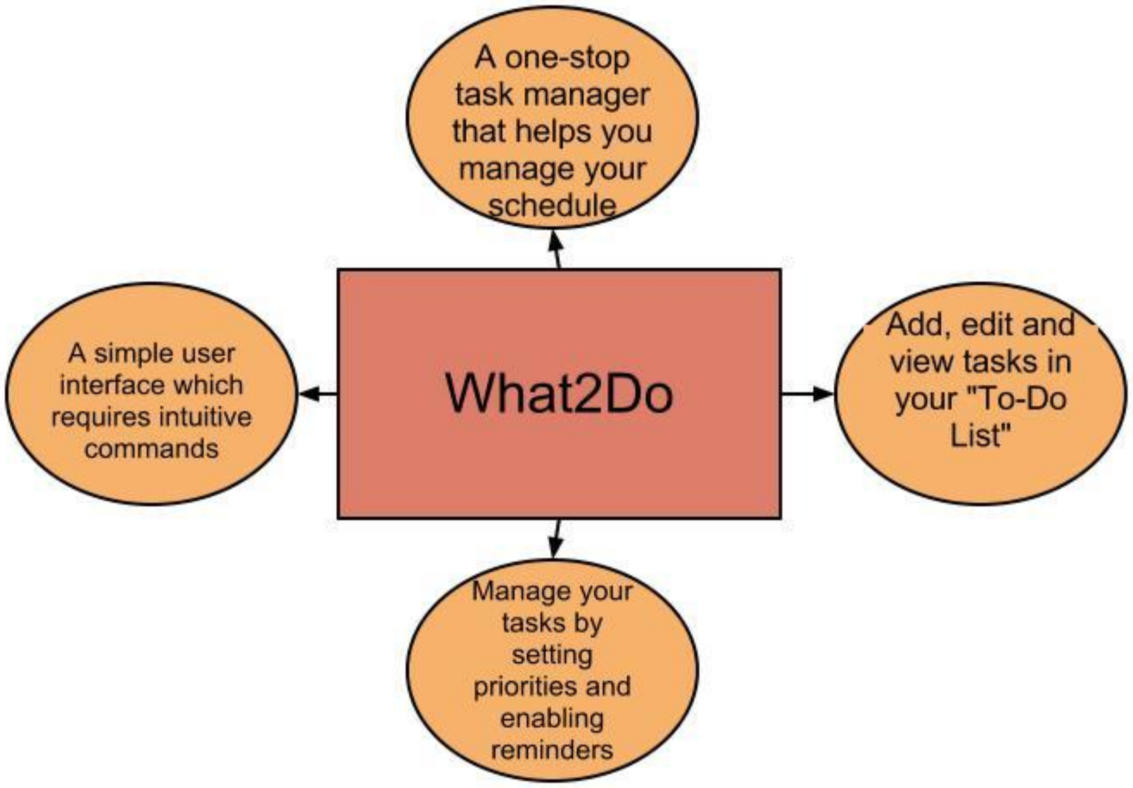
Akaash Gupta	Sandeep Paul	Le Khac Minh Tri	Anirup Sengupta
Team Lead, Logic Integrator	GUI	Database and Time Analyzing	Executor Class

USER GUIDE

What is What2Do!?

Hello BusyBee!

Ever been overwhelmed by your schedule? Unable to effectively manage time and prioritize tasks? Fret not, we have a software that shall solve all your woes! Ladies and gentleman, we proudly present to you What2Do!, your friendly "to-do" list manager, that shall enable you to take control over your schedule and make the best use of your time!


A one-stop task manager that helps you manage your schedule
A simple user interface which requires intuitive commands
What2Do
Add, edit and view tasks in your "To-Do List"
Manage your tasks by setting priorities and enabling reminders

COMMAND RUNTHROUGH

ADDING AN EVENT

[add/a/+] [Key words] [start-time and date] [End time and date] [Priority] [r-reminder time]

Additional Notes

- Adding hash tags using the '#' symbol classifies the task into the category which follows the hashtag
- Start and end time and date can be entered in highly flexible formats
- Priority can be set as high/h, low/l or normal(default)
- "r-reminder time" would indicate that a reminder is enabled for the task. This reminder goes of "reminder time" before the task deadline.

Example:

add meeting with cs2103 tutor #NUS 5pm today 6pm today r-1min22hr

- This would create a task "meeting with cs2103 tutor" under the "NUS Categry", starting time 5 pm of the current date and ending time 6 pm of the same date. Reminder would be set for 22 hours before.

SEARCHING FOR ENTRIES

[search/s] [Key words]

Additional Notes

- You can search for pre-entered tasks using the above command format
- The search results would be display in the "Search Result" tab, along with index number
- For subsequent operations on the searched for entries, you may reference the tasks using the index numbers displayed on the search pane.

Example:

search NUS

DELETING AN ENTRY

[delete/d/-] [index number]

Additional Notes

- You first need to search for a particular entry to be deleted. Do this using the search process as described above.
- To delete any of the display search results, type "delete [space] [index number as displayed in the search window]

UNDOING AN OPERATION

[undo]

Additional Notes

- The undo operation helps undo the last entered command.

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 -

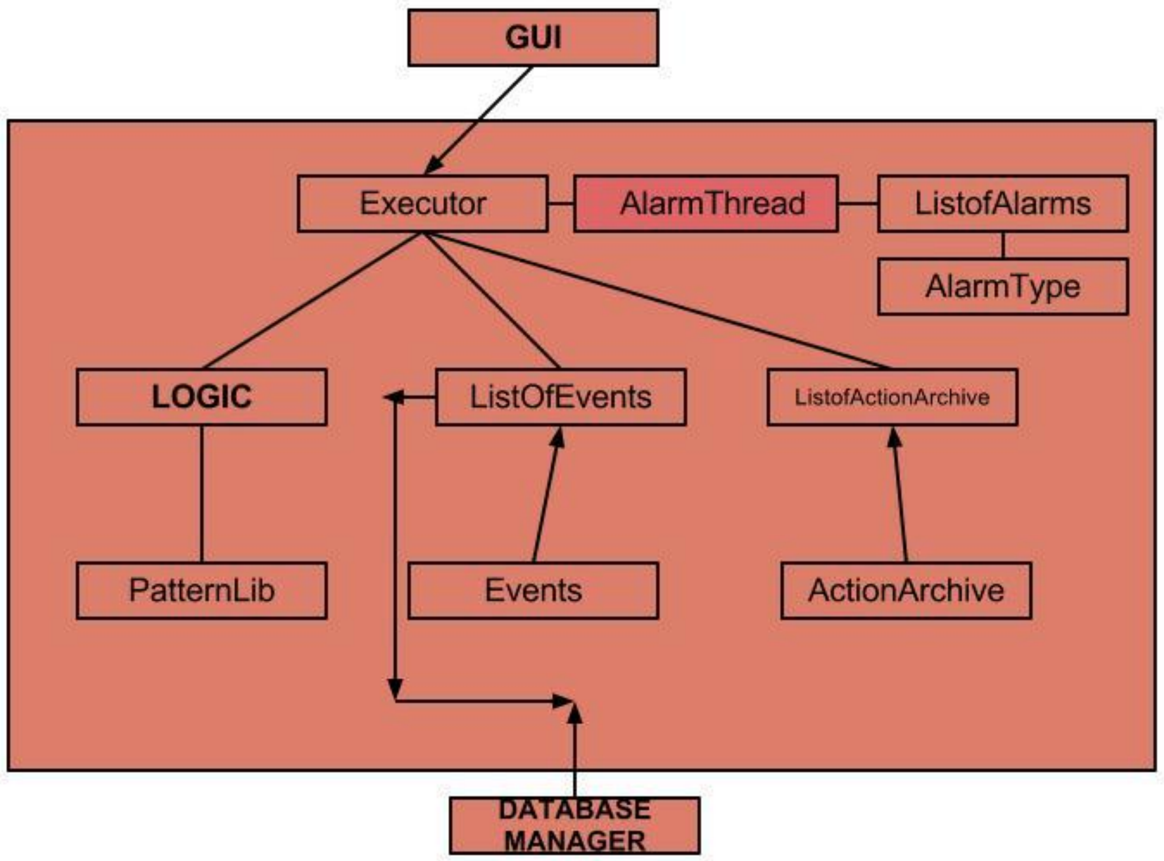
At the time of task creation, you shall have the option to set the priority to high (h/H), low(l/L) or normal(n/N)(default). High priority tasks shall be displayed in RED, and low priority ones in YELLOW.

DEVELOPER'S GUIDE

Hello, new developer! The What2Do! team is delighted to have you on board. Do peruse through the following information, you need to familiarize yourself with the structural design of our program before you join us!

Architecture

- For our project, we have chosen an **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 –

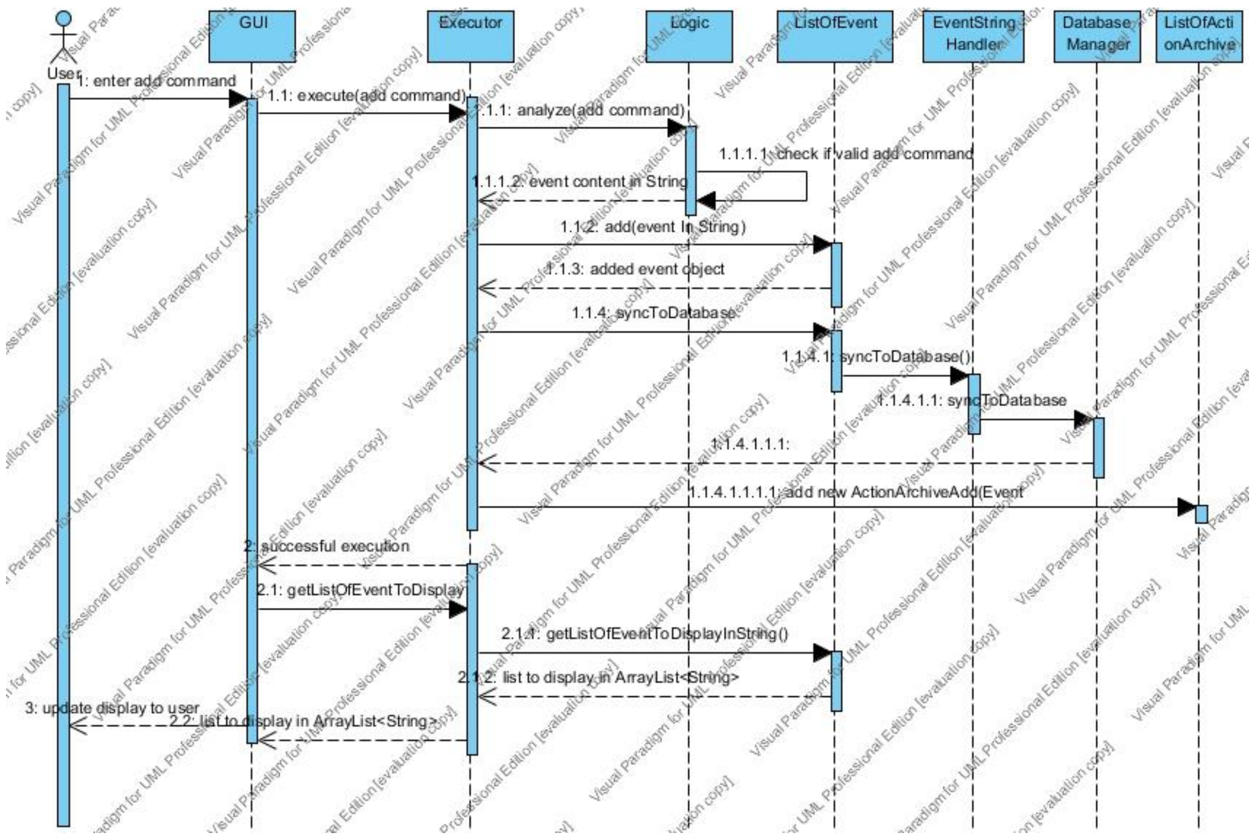


GUI Description

- The GUI contains **KeyListeners** at the JTextComponent where the user enters input. The KeyListener "listens" for the keyword "Enter" which indicates 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, 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)

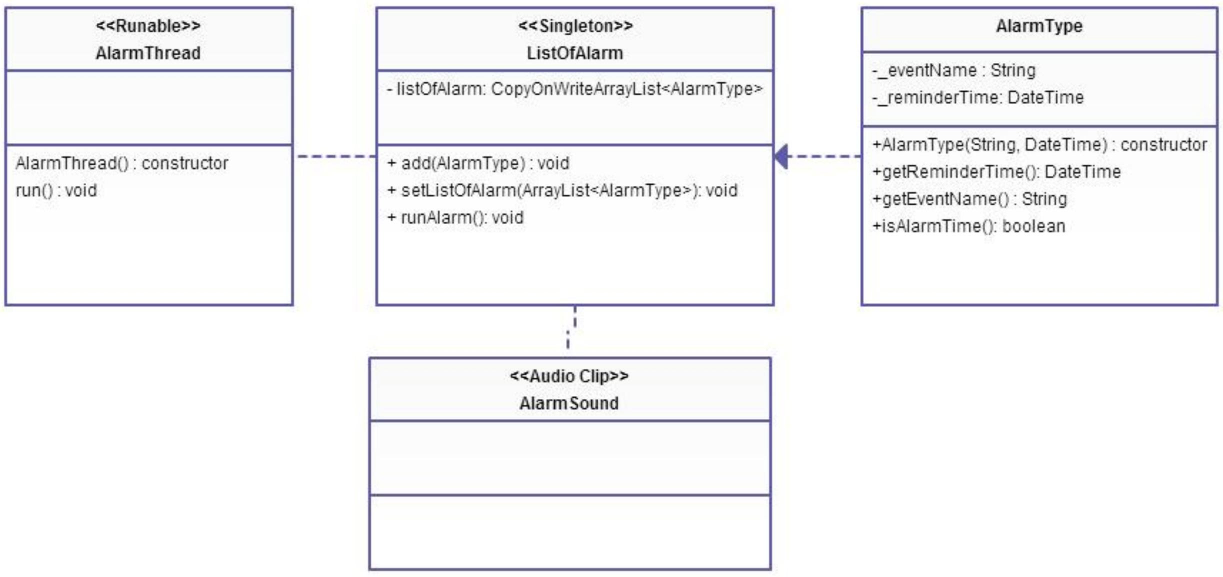
- **Design Description**:
    - The GUI class is the interface between the user and system. How the GUI works is explained in the previous section.
    - The GUI first loads the data from the local database at the start of the program by calling the relevant function in the executor class.
    - The 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.

SEQUENCE DIAGRAM FOR ADDING AN EVENT


User
GUI
Executor
Logic
ListOfEvent
EventString Handler
Database Manager
ListOfActi onArchive
1: enter add command
1.1: execute(add command)
1.1.1: analyze(add command)
1.1.1.1: check if valid add command
1.1.1.2: event content in String
1.1.2: add(event In String)
1.1.3: added event object
1.1.4: syncToDatabase
1.1.4.1: syncToDatabase()
1.1.4.1.1: syncToDatabase
1.1.4.1.1.1:
1.1.4.1.1.1.1: add new ActionArchiveAdd(Event
2: successful execution
2.1: getListOfEventToDisplay
2.1.1: getListOfEventToDisplayInString()
2.1.2: list to display in ArrayList<String>
2.2: list to display in ArrayList<String>
3: update display to user

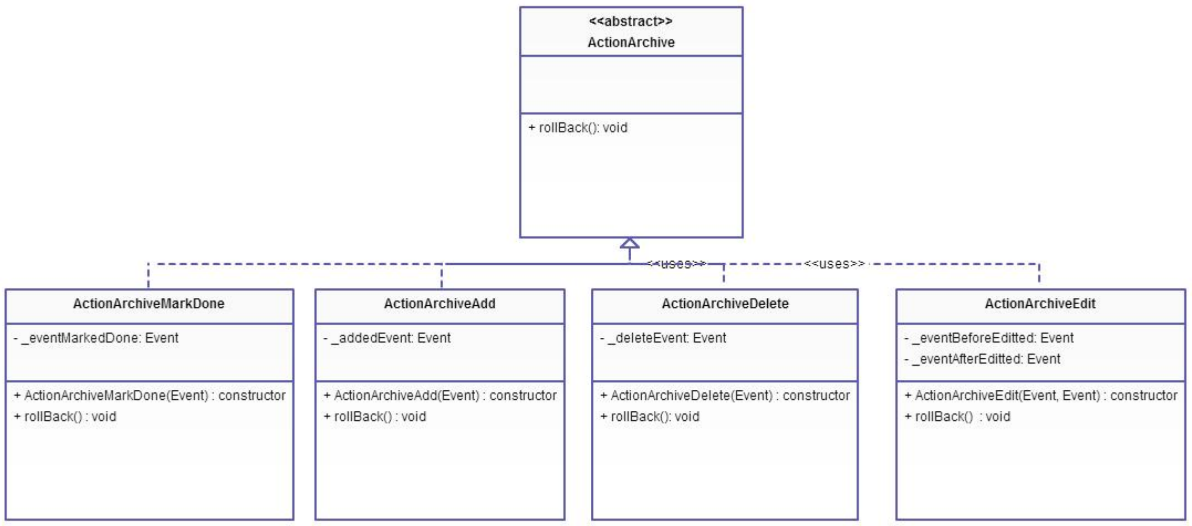
CLASS DIAGRAMS FOR SELECTED COMPONENTS OF THE PROGRAM

ALARMTYPE



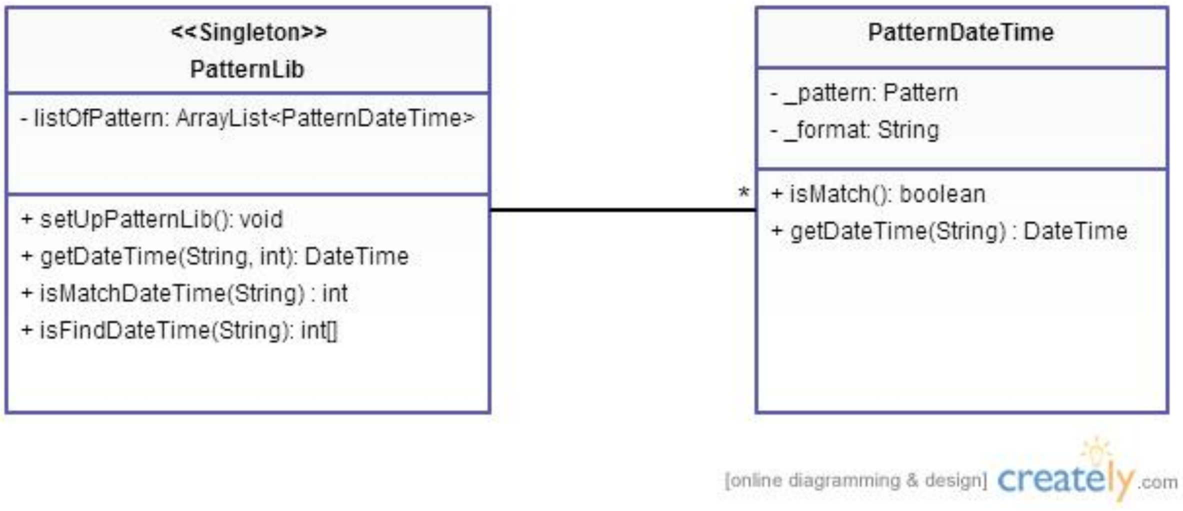
[online diagramming & design] creately.com

ACTION ARCHIVE

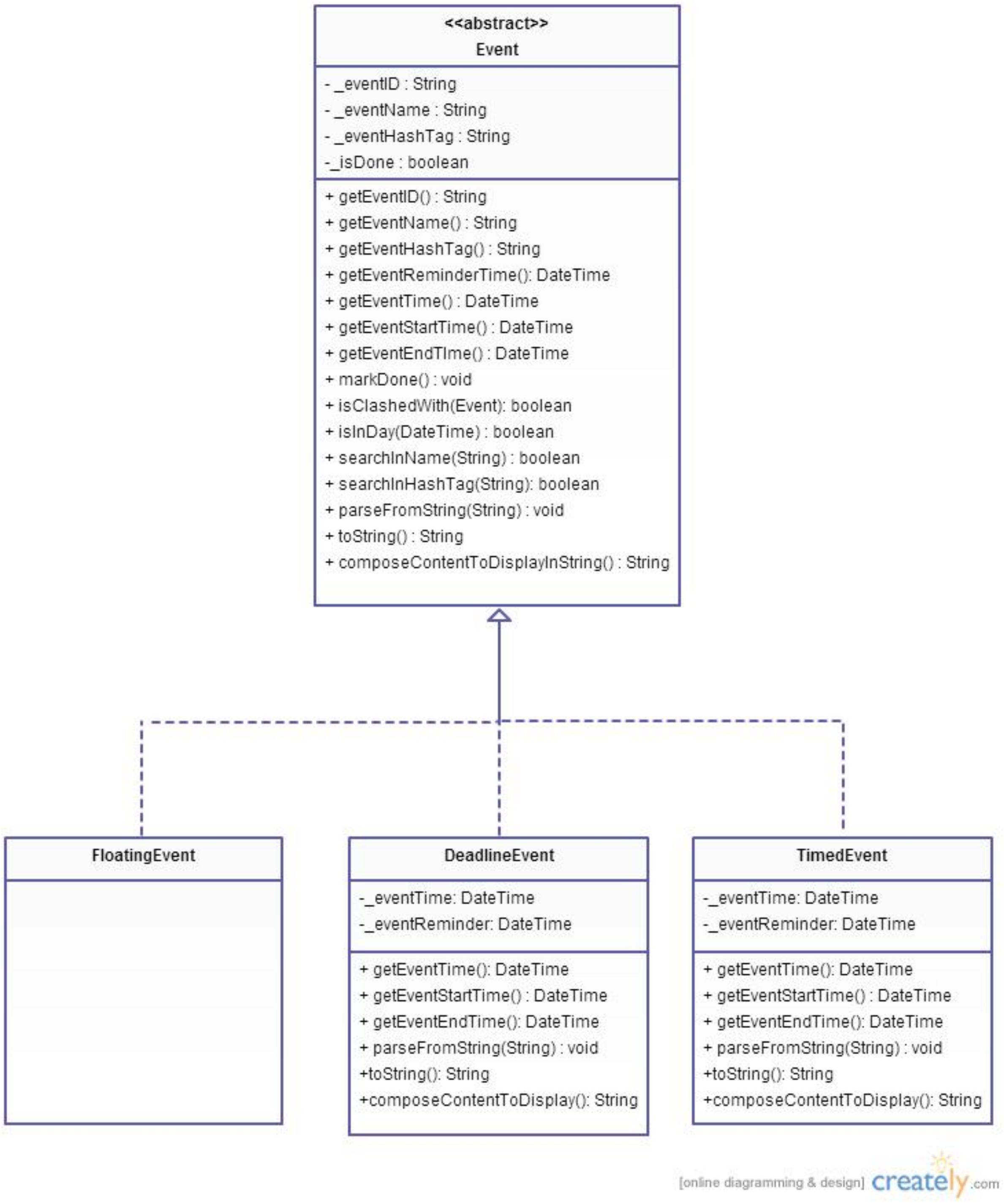

<<abstract>>
ActionArchive
+ rollBack(): void
<<uses>>
<<uses>>
ActionArchiveMarkDone
- _eventMarkedDone: Event
+ ActionArchiveMarkDone(Event) : constructor
+ rollBack() : void
ActionArchiveAdd
- _addedEvent: Event
+ ActionArchiveAdd(Event) : constructor
+ rollBack() : void
ActionArchiveDelete
- _deleteEvent: Event
+ ActionArchiveDelete(Event) : constructor
+ rollBack(): void
ActionArchiveEdit
- _eventBeforeEditted: Event
- _eventAfterEditted: Event
+ ActionArchiveEdit(Event, Event) : constructor
+ rollBack() : void

[online diagramming & design] creately.com

PATTERNLIB


<<Singleton>>
PatternLib
- listOfPattern: ArrayList<PatternDateTime>
+ setUpPatternLib(): void
+ getDateTime(String, int): DateTime
+ isMatchDateTime(String) : int
+ isFindDateTime(String): int[]
*
PatternDateTime
- _pattern: Pattern
- _format: String
+ isMatch(): boolean
+ getDateTime(String) : DateTime
[online diagramming & design] creately.com

EVENTS


<<abstract>>
Event
- _eventID : String
- _eventName : String
- _eventHashTag : String
-_isDone : boolean
+ getEventID() : String
+ getEventName() : String
+ getEventHashTag() : String
+ getEventReminderTime(): DateTime
+ getEventTime() : DateTime
+ getEventStartTime() : DateTime
+ getEventEndTIme() : DateTime
+ markDone() : void
+ isClashedWith(Event): boolean
+ isInDay(DateTime) : boolean
+ searchInName(String) : boolean
+ searchInHashTag(String): boolean
+ parseFromString(String) : void
+ toString() : String
+ composeContentToDisplayInString() : String
FloatingEvent
DeadlineEvent
-_eventTime: DateTime
-_eventReminder: DateTime
+ getEventTime(): DateTime
+ getEventStartTime() : DateTime
+ getEventEndTime(): DateTime
+ parseFromString(String) : void
+toString(): String
+composeContentToDisplay(): String
TimedEvent
-_eventTime: DateTime
-_eventReminder: DateTime
+ getEventTime(): DateTime
+ getEventStartTime() : DateTime
+ getEventEndTime(): DateTime
+ parseFromString(String) : void
+toString(): String
+composeContentToDisplay(): String
[online diagramming & design] creately.com

APPENDIX-I

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 –**

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

2) **Executor -**

Function Name -	***What It Does -***	***Called By -***
analyze()	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.

Function Name -	***What It Does -***	***Called By -***
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.

Function Name -	***What It Does -***	***Called By -***
add()	Adds the last action performed by the user to the Action Archive.	Executor
undo()	Undo's the last action performed by the User.	Executor

GUI Components

GUI Component	**Name In Program**	**Function**
JFrame	JFrame	The frame which contains the entire GUI
JPanel	Jpanel1	A panel which contains the core of the visible components of the GUI
TextField	textfield1	Contains the text area where the user shall enter his commands
JScrollPane-JTextPanel	JTextPane2	Contains the suggestiong box for user commands
JScrollPane-JTextArea	JTextArea5	The display box for Upcoming Events
JScrollPane-JTextArea	JTextArea6	The display box for Priority Events
JScrollPane-JTextArea	JTextArea7	The display box for search results
JScrollPane-JTextArea	JTextArea8	The display box for floating events

APPENDIX - 2

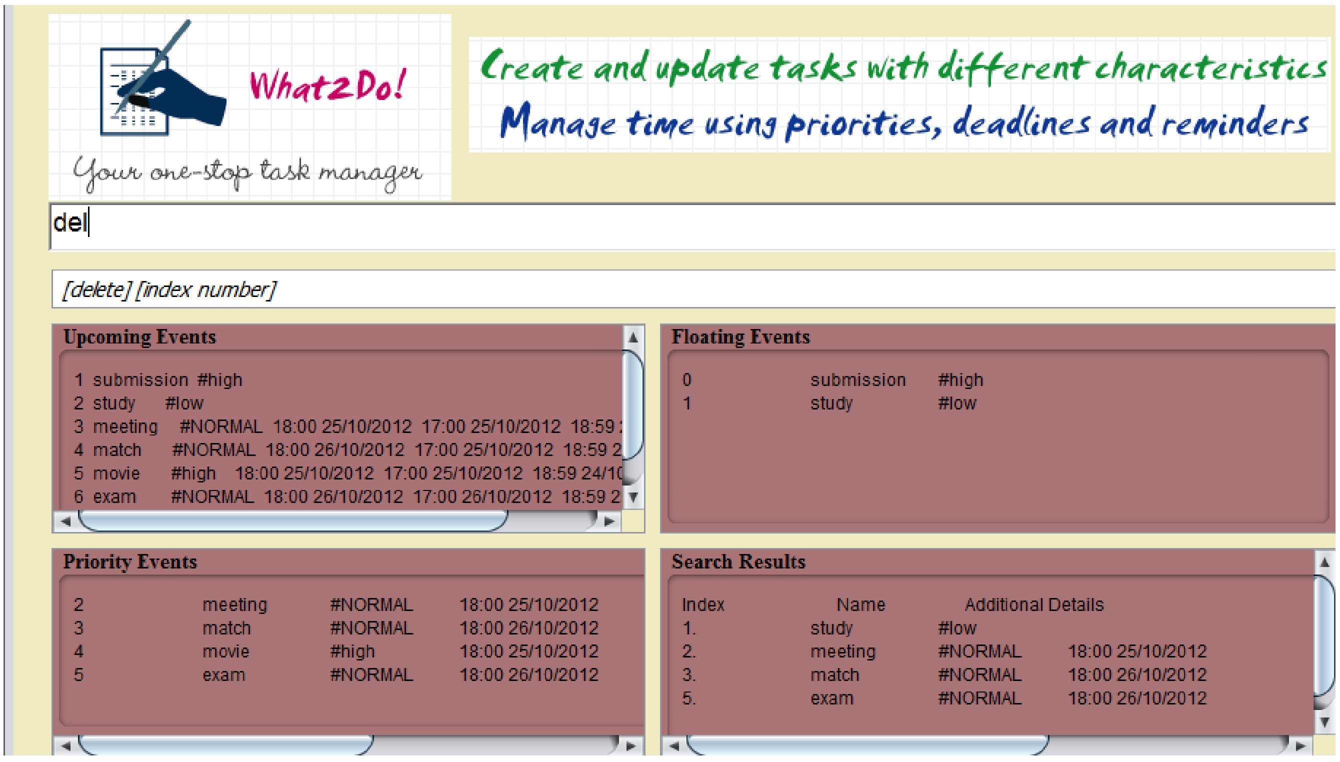
SALIENT FEATURES OF THE GUI

The GUI is the core of our software system, as we want users to have a hassle-free and enjoyable experience when they use What2Do!

Salient features of What2Do!'s GUI are as follows:

- <u>All-in-one display:</u>All the information that has been stored previously by the user is visible in one screen. The user need to navigate through multiple buttons/clicks/commands for this purpose.
- <u>Command format guide:</u> Since new users will take time to acquaint themselves with our system, we have a command format suggestion box right below the user input box. These suggestions change dynamically according to what the user is typing. For example, if he starts typing "de", the command suggestion box automatically shows the format for delete operation.
- <u>Keyboard Siri:</u> There is no need for any mouse click whatsoever. All reasonable tasks can be performed through keyboard commands.
- <u>Previous Command:</u> By pressing the UP arrow key, the user can access the command entered immediately before the current one.

A screenshot of the GUI is as follows:


What2Do!
Your one-stop task manager
Create and update tasks with different characteristics
Manage time using priorities, deadlines and reminders

```
del

[delete] [index number]
```

Upcoming Events

```
1 submission #high
2 study     #low
3 meeting   #NORMAL 18:00 25/10/2012 17:00 25/10/2012 18:59
4 match     #NORMAL 18:00 26/10/2012 17:00 25/10/2012 18:59 2
5 movie     #high   18:00 25/10/2012 17:00 25/10/2012 18:59 24/1
6 exam      #NORMAL 18:00 26/10/2012 17:00 26/10/2012 18:59 2
```

Floating Events

```
0        submission    #high
1        study         #low
```

Priority Events

```
2        meeting    #NORMAL    18:00 25/10/2012
3        match      #NORMAL    18:00 26/10/2012
4        movie      #high      18:00 25/10/2012
5        exam       #NORMAL    18:00 26/10/2012
```

Search Results

```
Index        Name        Additional Details
1.           study       #low
2.           meeting     #NORMAL    18:00 25/10/2012
3.           match       #NORMAL    18:00 26/10/2012
5.           exam        #NORMAL    18:00 26/10/2012
```

APPENDIX -3
TIME AND DATE FORMATS

- Time formats supported: 9h/hour(s), 9.00, 9:00, 9pm/am
- Date formats supported: today, this (Monday, etc.), next (Mon/Monday/etc.), 9Sep, 9-sep, 9/sep, 9Sep2000, 9/sep/2000
- Date Format : time followed by date. E.g. 7am this Mon, 7:00 9sep, 7pm today

CREDITS

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