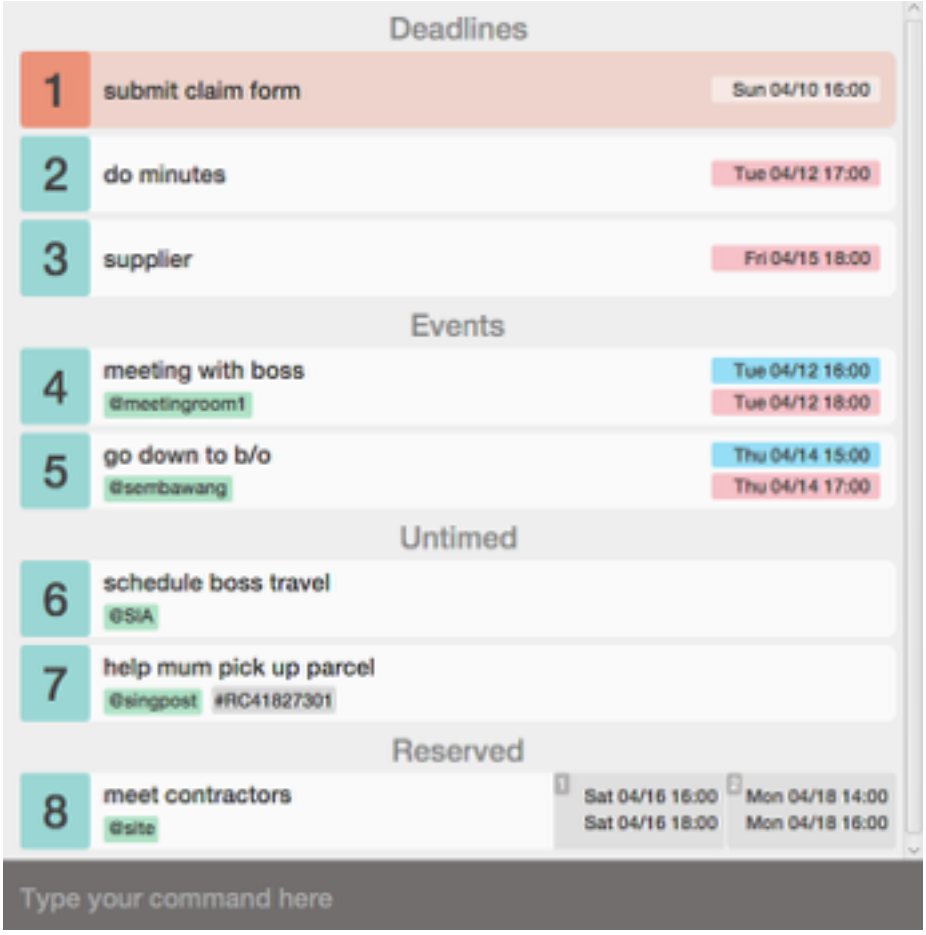
J.LISTEE



Supervisor: Karan Kamath **Extra feature**: GoodGui

			
Her Kai Lin Team lead, and Scheduling	Chloe Odquier Fortuna Git expert, Integration	Boh Tuang Hwee, Jehiel Code quality	Zhu Bingjing Documentation and UI

User Guide

Developer guide below

J.Listee V0.5
User guide

Quick Start

1

Install necessary files

Install and configure Java environment on your computer. You can download JDK here or copy paste http://www.oracle.com/technetwork/java/javase/downloads/index.html onto your web browser.

Download J.Listee Jar from GitHub.

Click here or copy paste https://github.com/cs2103jan2016-t15-2j/main/ onto your web browser and download jar file.

Run The Application

Double-click on the jar file you downloaded to run the application.

Try it Yourself

Try to type the following commands in the command box and see what's happening

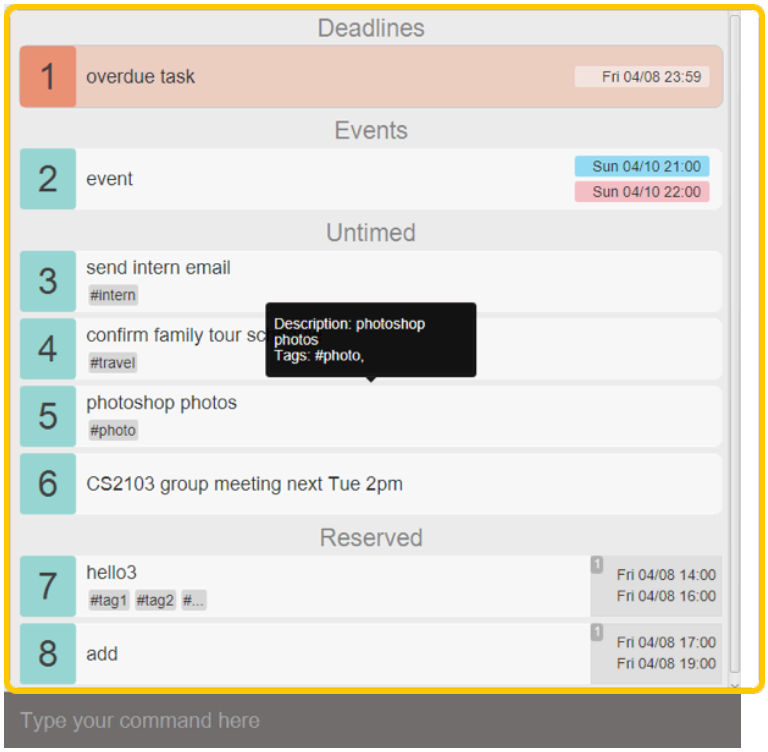
```
add my first task @icube from today 4pm to 6pm (press enter)
update 1 updated first task (press enter)
done 1 (press enter)
```

Need help?

Type "help" on the command box if you should require any assistance!

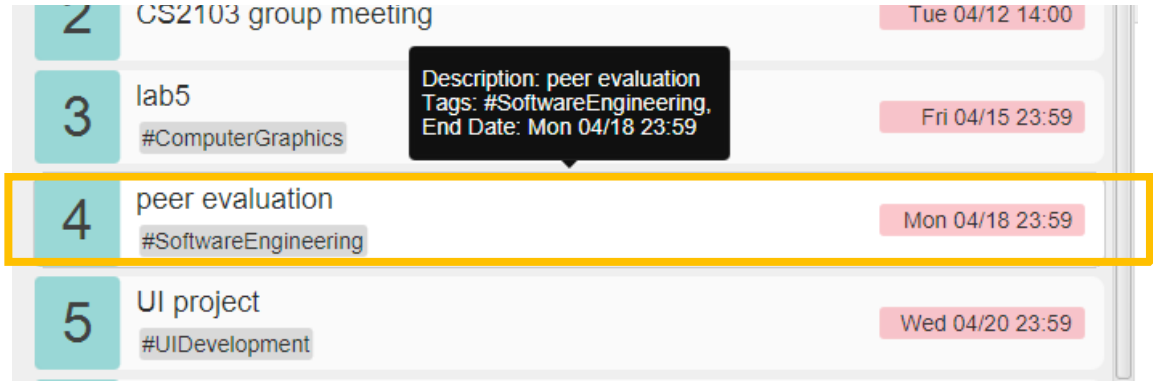
Understanding the User Interface 2

2.1 List Pane



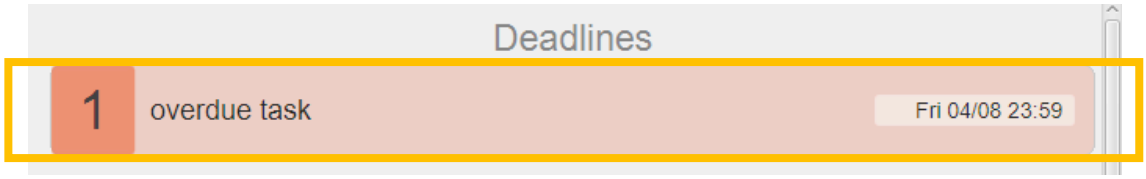
The first part is the list pane which displays list of tasks. On startup, J.Listee will show you overdue tasks, today's tasks, untimed tasks and reserved tasks, automatically ordered by time within each group.

Tasks



The list pane contains blocks of tasks. You can use **arrow keys** (↑ and ↓) or click to focus a specific task when the command bar is not focused. The focused task will be highlighted and a tooltip containing all of the task's information will appear. The focused task can then be managed using keyboard shortcut, which will be detailed in Feature Details.

Overdue Tasks

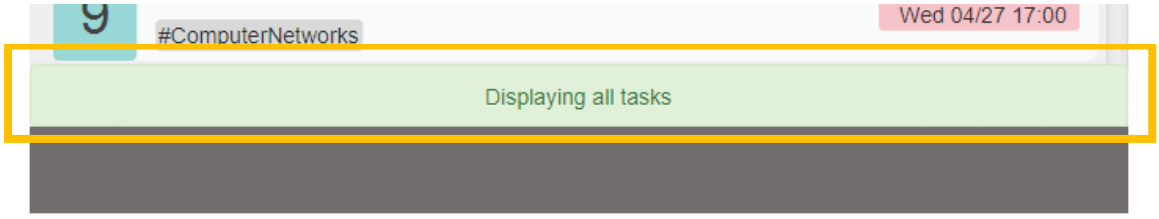


Overdue tasks will be colored in red to remind you. And they are constantly synchronized by **J.Listee**, which means as soon as a task becomes overdue, you will immediately be reminded!

Conflict Tasks

When you input a time that clashes with other tasks, the conflicted tasks will blink for a while to capture your attention and serve as a warning.

2.2 Message Bar

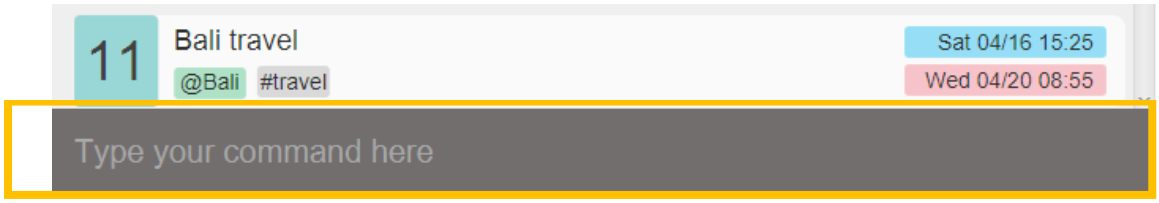


Feature Details

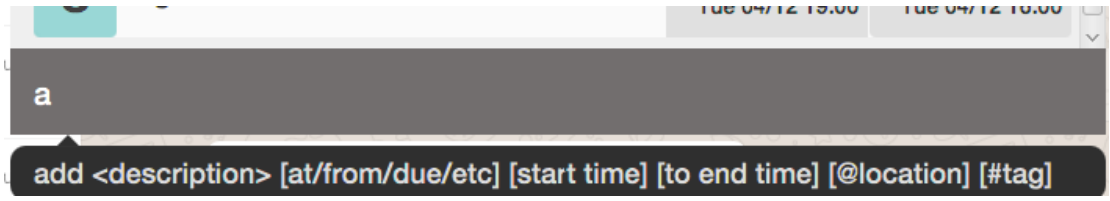
3

Every time you make some operations, the message bar containing feedback messages will be automatically triggered and disappear after a while.

2.3 Command Bar



The command bar is where you type your command and tell **J.Listee** what to do next. You can use **arrow keys** (↑ and ↓) to retrieve previous or later commands you have entered when the command bar is focused. Usually when an operation is made, the focus will be on some tasks in the list pane. Then you can **press space bar** to focus command bar again and type your next command.



When you are typing in the command bar, command suggestion will be shown. So follow them if you forget the command format!

There are a few notes to take note of for feature :

Symbol	Representation
< >	compulsory input
[ ]	optional input

Symbol	Representation
@	must add symbol if indicating location
#	must add symbol if indicating hashtag

3.1 Add Tasks

Untimed Tasks

Representing task without time

Example *"add wash clothes some time @home #laundry #sigh"*

Format : <add> <description> [@location] [#tags]

Must-Have Arguments: Task Description

Optional Arguments: location, tags (cannot contain white space)

Remarks: Task Description, location, tags can be inserted in random order such as
add @home #sigh wash clothes #laundry

Deadline Tasks

Representing task with 1 end time

Example *"add watch BigBang Theory this Monday @mac-common"*

Format : <add> <description> <<preposition> time> [@location] [#tags]

Must-Have Arguments: Task description, deadline time
Optional Arguments: location, tags (tags cannot contain whitespace)

Remarks: Task Description, location, tags and time can be inserted in random order such as : add this Monday @ mac-common watch BigBangTheory, note that the last preposition will then be date.

Event Tasks

Representing task with start and end time

Example *"add Bali Travel from 4/16 2pm to 4/20 4pm @Bali #travel"*

Format: <add> <description> <<from> start time <to>end time> [<@>location][#tags]

Must-have arguments: Task description, start time, end time

Optional arguments: location, tags (tags cannot contain whitespace)

3.2 Delete Tasks

This command enables you to delete existing tasks from **J.Listee**.
For example, if you want to delete task 1, 2, 3 and 5, you can try the following

"delete 1-3, 5" *"del 1-3, 5"*

Format:
<delete> or <del> <starting task number - ending task number>
<delete> or <del> <tasknumber, tasknumber, tasknumber>

Must-have arguments: at least one task number

Optional arguments: more task number separated by English comma or linked by minus

Shortcut: Select the specific task and press Backspace or Delete, the task will be deleted.

3.3 Update Tasks

Tasks that is not reserved

Example *“update 2 more work +end tonight 2359”*

Update the 2nd task’s description to “more work” and change end time to “tonight 2359”

Format:
<update> / <edit> <task number> [+ OR - <timegroup> time] [task description] [@location] OR [-@] [#hashtags][-#deletehashtags]

Must-have arguments:
a task number indicating the specific task

Optional arguments:
new task description, removing start time, new start time, new end time, removing existing location OR new location, removing specific tags, new tags

Tasks that is reserved

Example *“update 3 2 +start today 4pm”*

Update the 3rd task, 2nd time slot’s start time to “today 4pm”

Format:
<update> <task number > [<reservation index> < [-] OR [+] timegroup [time]> [task description] [@location] OR [-@] [#hashtags][-#deletehashtags]

Must-have arguments:
a task number indicating the specific task, a reservation index indicating the specific time slot reserved

Optional arguments:
new task description, new start time, new end time, removing existing location OR new location, removing specific tags, new tags

3.4 Postpone Tasks

This command enables you to postpone the deadline of an existing task for some minutes, hours, days, months, years.

Example *“postpone 3 3years 2months 8days”*

Format:
<postpone> <task number> <time>

Must-have arguments:
a task number indicating a specific task, time

Shortcut:
Select the specific task and press p, the deadline of the task will be postponed for 1 day.

3.5 Search Tasks

This command enables you to see specific tasks in a time period or with some specific information.

Example *“show /today @school assignment”*

Which means to show today’s task at school with the keyword assignment

Format:
<show> [task description] [/taskgroup] [<preposition> time] [@location] [#hashtags]

Must-have arguments:
some filter

3.6 Reserve Tasks

This command enables you to reserve multiple time slots for later use.

Example *“reserve meet friends from thurs 4pm to Thursday 6pm or friday 4pm to 6pm #dinner”*

Which means to the reserved “meet friends” task will have 2 timeslots, one from Thusday 4pm to 6pm, the other from Friday 4pm to 6pm

Format:
reserve <task description> <<from> start time <to> end time> [<and/or> <from> start time <to> end time] [@location] [#hashtags]

Must-have arguments:
Task description and at least one timeslot

Optional arguments:
more timeslots linked by and/or

3.7 Confirm timeslot

This command enables you to confirm one of the time slots you have reserved.

Example *“confirm 2 3”*

Which means to confirm the 3rd reserved time slot of the 2nd task

Format:
<confirm> <task number> <reservation index>

Must-have arguments:
a task number indicating the specific task and the index of a timeslot reserved

Shortcut:

Select the specific task and press the number key, and the corresponding timeslot of the task will be confirmed.

3.8 Mark as done/undone

This command enables you to mark existing tasks as done/undone. The completed tasks will then disappear from the current page. You can type “show /done” to see the tasks just marked as done.

Example “undone 1-3, 5”

Which means to mark 1, 2, 3, 5 as undone (the tasks must be completed at the first place)

Format:
<done> <starting task number - ending task number>
<done> <tasknumber, tasknumber, tasknumber>

Must-have arguments:
at least one task number

Optional arguments:
more task number separated by English comma or linked by minus

Shortcut:
Select the specific task and press d, if the task is completed, it will be marked as undone, otherwise it’ll be marked as done.

3.9 Change storage location

This command enables you to change the storage location to the folder you want. After entering this command, a folder choosing window will pop up, and you can choose the folder you want to put the file in.

Example *“change filepath”*

Format:
change filepath

Shortcut:
Press ctrl+f to pop up the folder chooser.

3.10 Undo/Redo

This command enables you to undo or redo the most recent operation.

Example *“redo”*

Format:
undo/redo

Shortcut:
Press ctrl+z to undo last operation; Press ctrl+y to redo last undid operation

3.11 Help Screen

This command enables you to see help screen.

Example *“help”*

Format:
help
Shortcut:
Press ctrl+h to see help screen.

3.12 Exit

This command enables you to exit from **J.Listee.**

Example “ exit ”

Format:
exit or quit
Shortcut:
Press Escape to exit.

Summary of Commands (in alphabetical order) 4

Command Type	Command	Format	Example
add deadline	add	<add> [description] [<preposition> time] [@location] [#tags]	add good friday this friday @home #yay #study
add event	add	<add> [description] [<from> start time <to> end time] [*@*location] [#tag]	add meeting with prof @com1 #work from 1st april 2pm to 5th april 3pm
add untimed task	add	<add> [description] [@location] [#tags]	add @room clean
change a filepath for txt file	change filepath		
confirm a reserve task starting from left most as index 1	confirm cfm	<confirm> <tasknumber> <reserve task index>	confirm 2 3
delete task	delete del	<del> or <delete> <starting task number - ending task number> <del> or <delete> <tasknumber, tasknumber, tasknumber>	delete 1-3, 5 delete 1,2,3,4,5 delete 1-5
done task	done	<done> <starting task number - ending task number> <done> tasknumber, tasknumber, tasknumber>	done 1-3, 5 done 1,2,3,4,5 done 1-5
edit other task (not reserved)	update edit	<update> <task number > [</timegroup> time] [task description] [@location] OR [-@] [#hashtags][-#deletehashtags]	update 2 more work /end tonight 2359 edit 2 /both today from 3pm to 6pm -@

Command Type	Command	Format	Example
edit reserve task	update edit	<update> <task number > [<reservation index> <[-]/ timegroup [time]> [task description] [@location] OR [-@] [#hashtags][-#deletehashtags]	update 1 1 /start today 4pm edit 2 meeting manager @office edit 3 -start
exit from app	exit		
help screen	help		
postpone a task	postpone pp	<postpone> <task number> <<time>>	postpone 1 2year postpone 2 2hours 4mins postpone 3 3years 2months 8days
redo previous command	redo		
reserve multiple slots of time	reserve res	reserve <task description> [<from> start time <to> end time] [and/or] [<from> start time <to> end time] [@location] [#hashtags]	reserve meet friends from thursday 4pm to thursday 6pm or friday 4pm to 6pm #dinner
search for a certain group of task, particular location or hashtag	show search	<show> [task description] [/taskgroup][<preposition> time] [@location] [#hashtags]	show assignment due friday or show /today @school assignment

List of Task Group, Preposition, Time Group

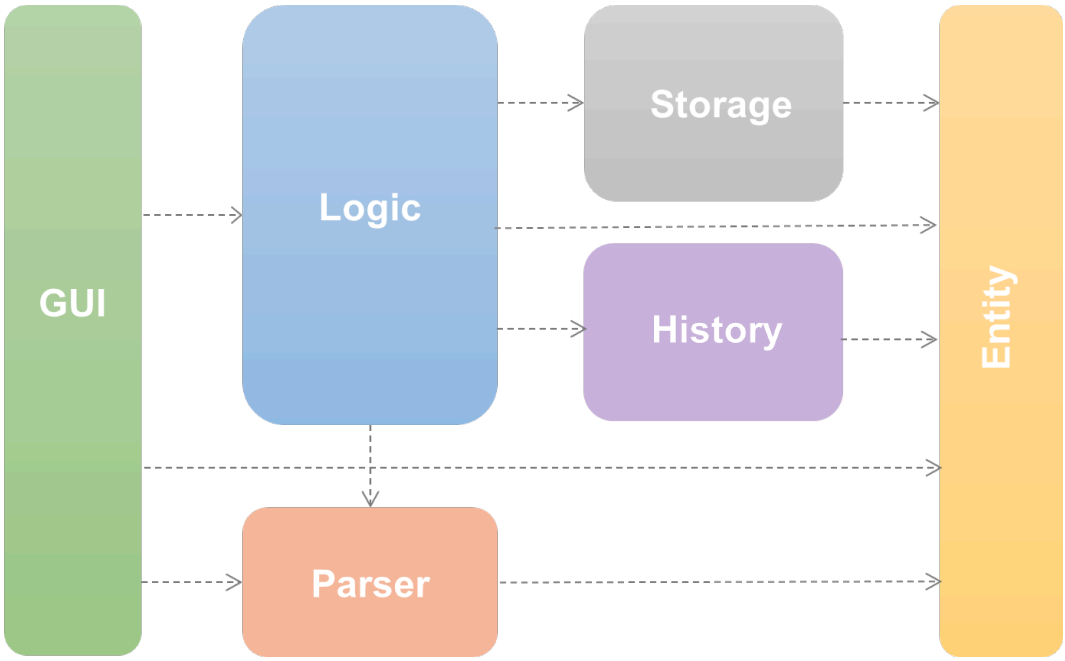
5

Task group for show / search	Prepositions before time	Time group for update
/today	from…to….	/end
/tomorrow	by	/start
/overdue	on	/etime
/done	at	/stime
/reserved	due	-alltime
/events	during	-end
/deadlines	in	-start
/untimed	for	/both
	this	
	before	
	after	
	next	

Command Type	Command	Format	Example
undo previous command	undo		
undone a done task	undone	<undone> <starting task number - ending task number> <undone> tasknumber, tasknumber, tasknumber>	undone 1-3, 5 undone 1,2,3,4,5 undone 1-5

J.Listee V0.5
Developer guide

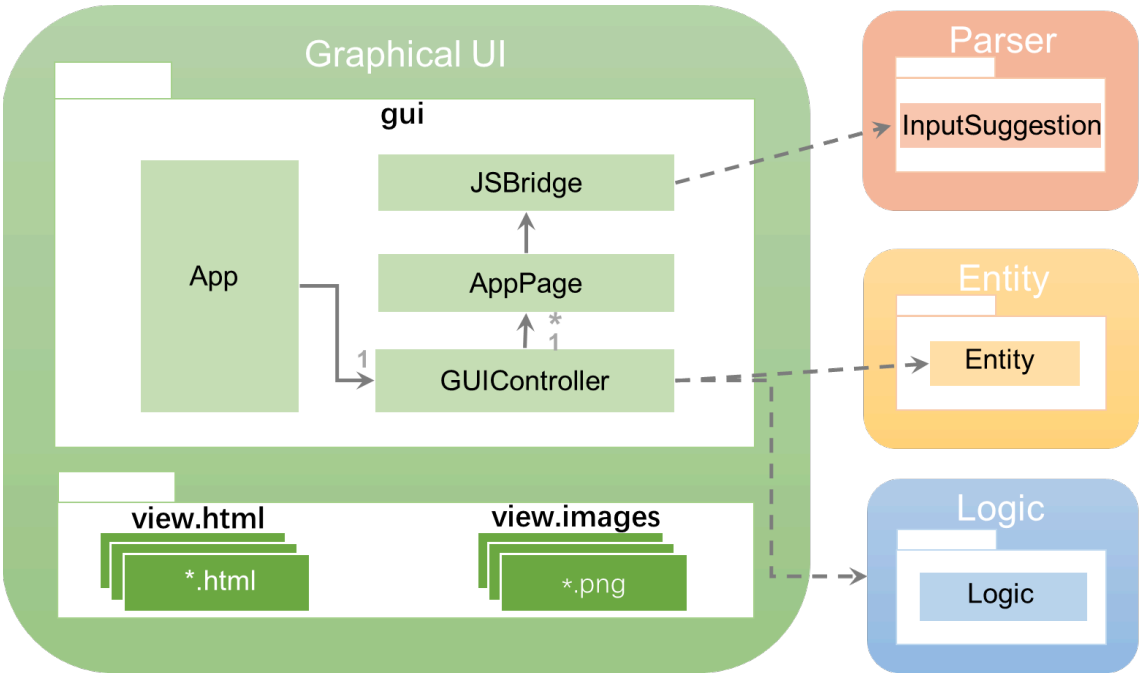
Architecture 1



J.Listee is made up of six main components.
The **GUI** component consists of JavaFX's HTML files which define the layout that users interact with and the Java files which communicate with these HTML files through JavaScript.
The **Logic** component serves as the interface between GUI and the respective components.
The **Storage** component consists of two parts. One is a task file which saves the user's tasks so that he can access his to-do list whenever he starts **J.Listee**. The other is a log file which saves the storage location of user's task file.
The **Parser** component consists of a parser class which is used to analyze user's command and then return the command to Logic for following execution.
The **History** component is where the previous display objects are stored in case user wants to undo/redo certain commands.
The **Entity** component represents objects involved in the management of tasks such as the tasks to do and commands that the user wants the app to execute.

GUI component

2



The GUI component is made up of two packages, gui and view. The gui package contains the Java files that control what users see while the view package contains HTML files that describe the layout that users interact with and also images that are used in the application.

App Class

The App class is the entry of **J.Listee**. App extends from JavaFX's Application class and overrides its start method. This method is the starting point of the whole application and very importantly, initializes the main frame that are required for the GUI. This method also calls another method to judge whether it's necessary for user to choose storage location and show corresponding page according to the judgement.

GUIController Class

The GUIController class is the main driver for the GUI component. It controls what users see and handles user inputs by passing them to the Logic component.

Notable APIs:

Return Type	Method and Description
void	**createFile(String userChosenfile)** : Call Logic to write into the log file the user chosen storage location and create the text file that holds the contents of the task list for storage.
void	**initializeList(String filePath)** : Initialize the start page which display deadlines, events and floating tasks
void	**displayWelcome ()** : Set the current scene of stage to welcome page.
void	**displayList (Display display)** : Set the current scene of stage to showList page.
void	**displayHelp() :** Set the current scene of stage to help page.
void	**handelUserInput(String command)** : Decides what to do when command is entered.
void	**changeFilePath (String newPath) :** change storage location by pass the new path to Logic.

AppPage Class

The AppPage class is an abstract class which extends from JavaFX's StackPane class. Each child of AppPage will load a html file from html package, and use JSBridge class as its JavaScript interface object which acts as a bridge between Java and JavaScript.
The class uses Singleton Pattern to ensure the consistency of command history across app pages.

JSBridge Class

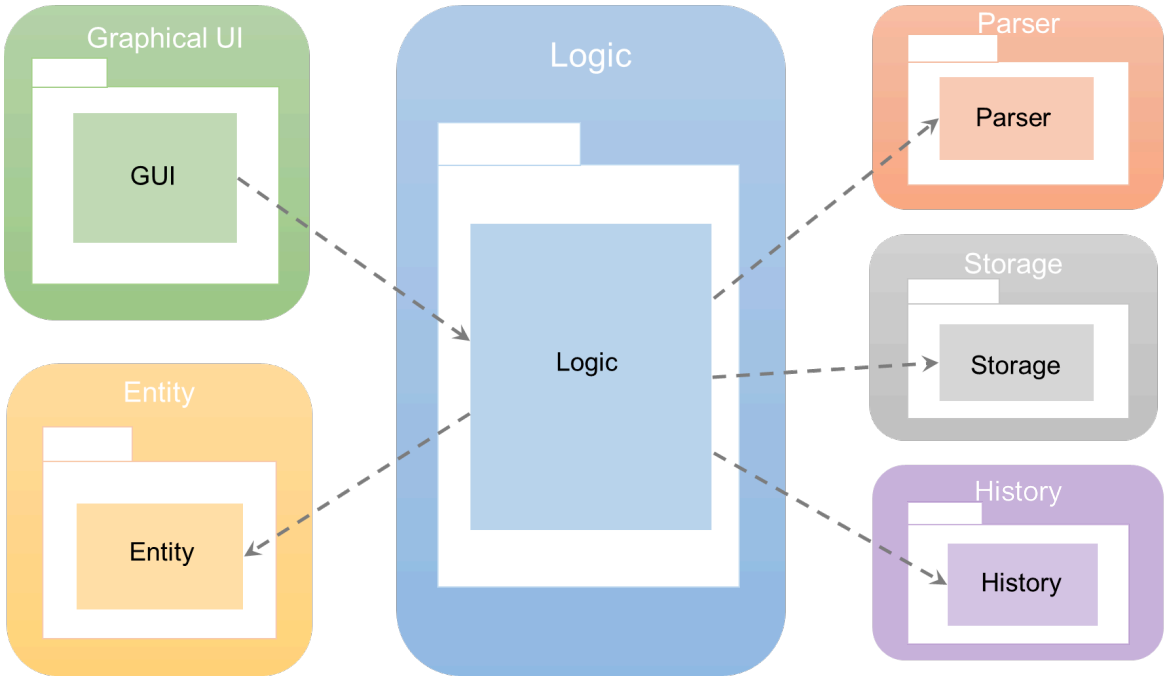
The JSBridge class communicates Java and JavaScript in the html file. It calls some methods of GUIController. For example, it will call handelUserInput whenever user inputs command. This ensures that the logic is handled by GUIController to avoid unnecessary coupling between AppPage and Logic.

Notable APIs:

Return Type	Method and Description
void	**getInstance()** : Return the only instance of this class.
void	**receiveCommand(String userInput)** : Add user input command to command history, check special commands, and call Logic to handle it.
void	**getCommandSuggestion (String cmd)** : Get command suggestion from Parser.
void	**getPreviousCmd()** : Get previous command.
void	**getLaterCmd() :** Get later command.
void	**chooseFolder() :** Pop up a filechooser window and ask for new filepath.

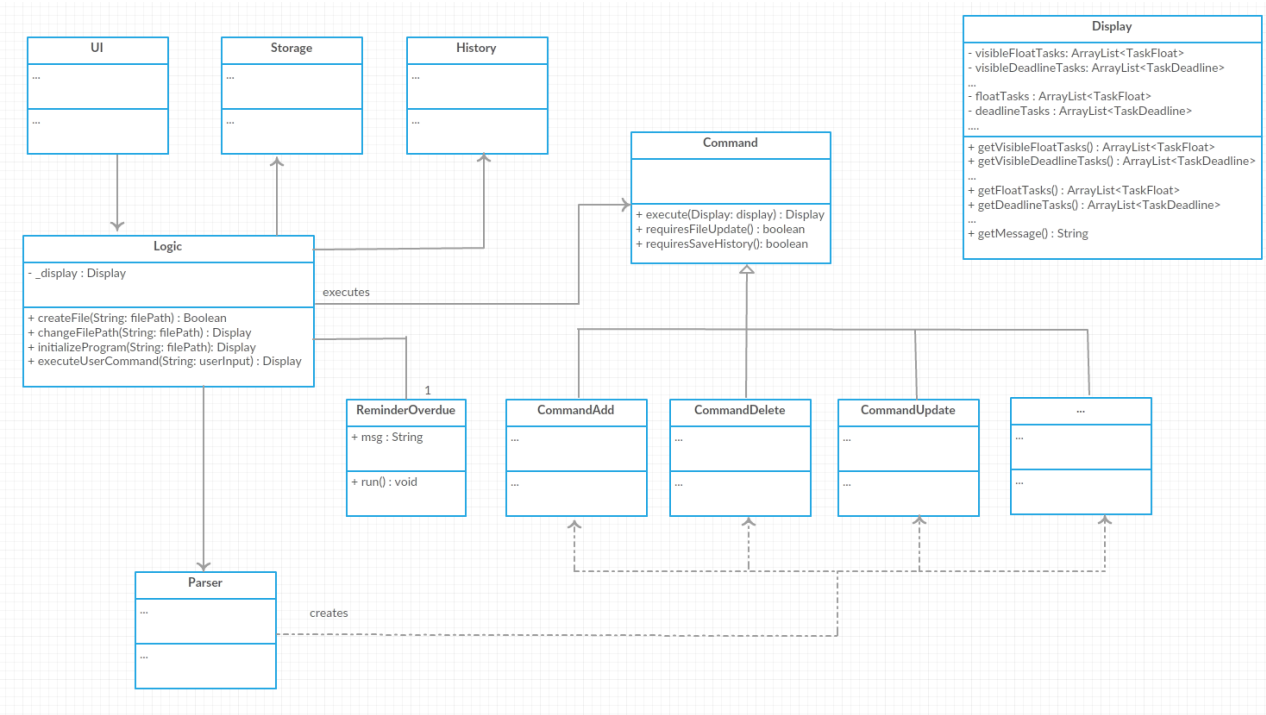
Logic component

3



Logic serves as the interface between UI and the sub-components. It takes in the user inputs from the UI component and executes them as commands before returning the result to UI.

Logic Class



Logic serves as the interface between GUI and the respective components.
As seen above, Logic implements the command pattern and Liskov substitution principle where a general command is executed at logic. This helps to reduce coupling as Logic will not need to know which command (add, delete, etc) or the internal details of a command. Logic only needs to call the execute method.

Notable APIs:

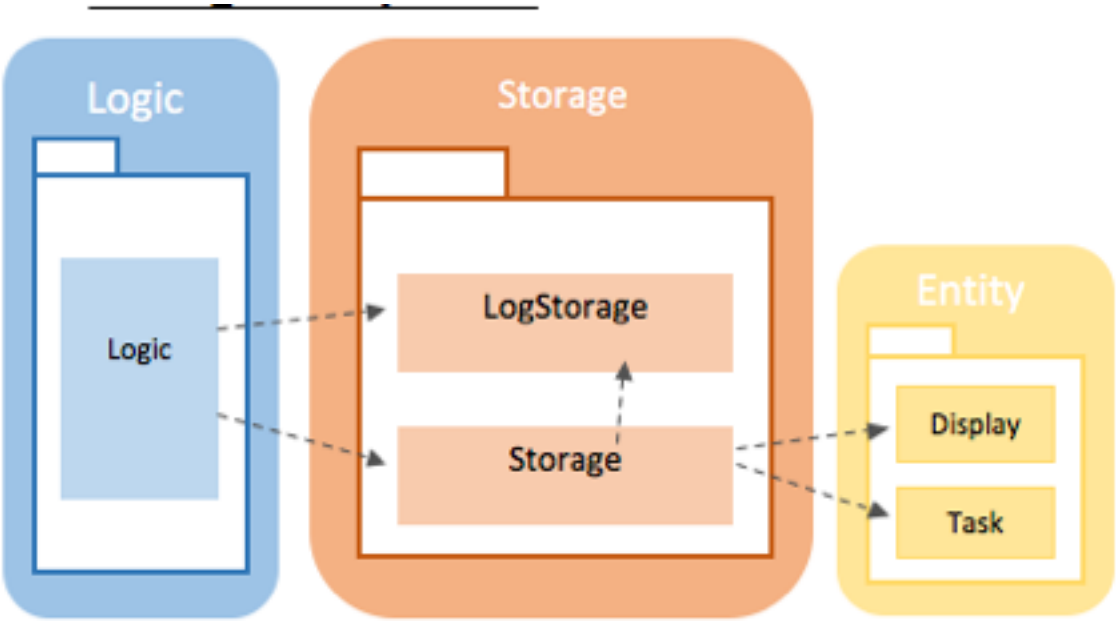
Return Type	Method and Description
void	**createFile(String filepath)** : This method is called on the very first startup of the program. The file path, which is specified by the user, is then sent to storage. The result is a Boolean which indicates whether the file is created successfully.
Display	**initializeProgram(String filePath)** : initializeProgram is called every start up the program to retrieve the display from storage.
Display	**ExecuteUserCommand(String userInput)** : The method is called by GUI whenever the user enters in a command. User inputs are sent to the parser to be processed. The result is a command which is then executed at Logic. All commands implement the command interface. Logic then sends the output to History and/or Storage to be saved if necessary before returning it to GUI.
Display	**changeFilePath(String filePath):** Calls storage to change the file path of the stored text file.

ReminderOverdue Class

Logic also contains a reminderOverdue timertask which is constantly checking for overdue tasks.

Storage component

4



The Storage Component composes of two classes: LogStorage and Storage. This component saves the user's tasks so that he can access his to do list whenever he starts **J.Listee**.

LogStorage Class

The LogStorage class is used for reading and writing the log file which contains the file path of the text file used as storage. When first starting up **J.Listee**, LogStorage writes the location of the storage into the log file. Upon subsequence uses of **J.Listee**, LogStorage automatically reads the log file for the storage's location.

Notable APIs:

Return Type	Method and Description
String	readLog() : Reads the log file that contains the filepath of the task list. If the log file doesn't exist, creates the log file.
void	writeLogFile(String filePath) : Writes the filepath into the log file.

Storage Class

The Storage class is used for manipulating to the human-readable and editable text file which contains the storage of all the tasks used in **J.Listee**. It creates the text file, moves its location, retrieves the data stored in the file for Logic to manipulate upon opening up **J.Listee**, and saves the changes to the file the user has created after operations that update the task list.Patterns:

Singleton – Only one instance of the Storage class is needed. Instead of using a public constructor to instantiate this class, the method getInstance() is used to prevent more than one instance of Storage being created.

Principles:

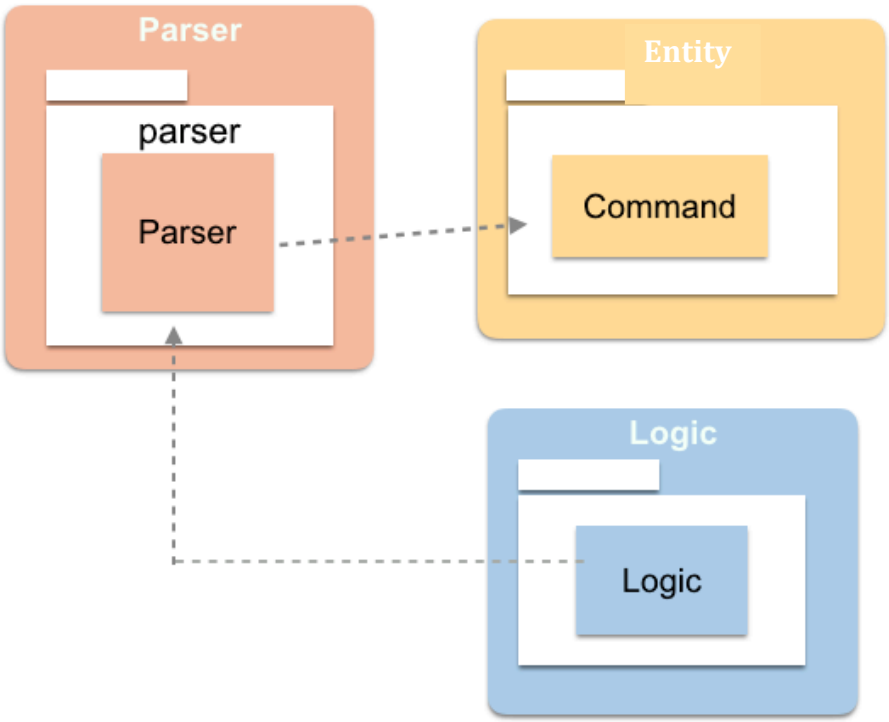
Single Responsibility Principle – Every class has its own responsibility. Therefore, LogStorage only reads and writes to the log file, and Storage only reads and writes to the text file that contains the task list.

Notable APIs:

Return Type	Method and Description
Storage	getInstance() : Instantiates the Storage class according to the Singleton pattern
void	createFile(String filepath) : Creates the text file that holds the contents of the task list for storage.
Display	getDisplay(String filepath) : Reads the contents of the storage file and sends it as a Display object for the Logic component.
void	saveFile(Display thisDisplay) : Writes all the contents of the Display object to the text file containing the storage of the task list.
void	changeFilePath(String newFilePath) : Changes the location of the storage file. If a valid storage file already exists in the new filepath, read from that file instead.

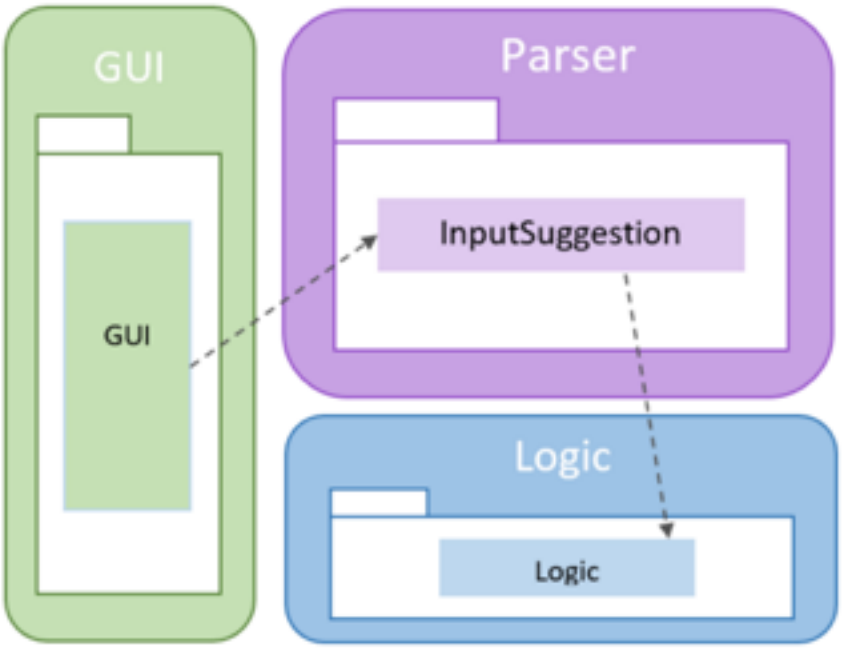
Parser component

5



The Parser component consists of a Parser class which is used to analyze user's command and then return the command to Logic for following execution.

InputSuggestion Class

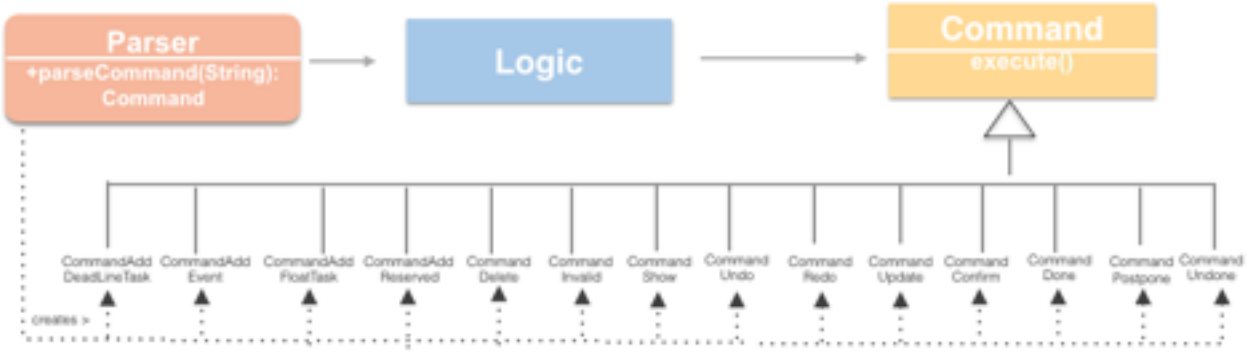


The InputSuggestion class is used for giving command suggestions to the user as he is typing. While the user is typing, his current input is given to this class. InputSuggestion will then return to the UI a String with a command suggestion based on what is currently typed in so that the user can see.

APIs	
Return Type	**Method and Description**
InputSuggestion	getInstance() : Instantiates the InputSuggestion class according to the Singleton pattern
String	getSuggestedInput(String currentInput) : Uses the currentInput which is what the user has currently typed and returns a String that contains a suggestion and command format for the command they typed out or are currently typing.

JListeeParser Class

The Parser class objective is to break down the input string of the user and determine type of command, creates a Command object and passes to the Logic class for execution.



Notable APIs:

Return Type	Method and Description
Command	**parseCommand(String inputLine)** : Returns the Command object with defined fields in each Command object depending the type of command.

Example:
User input: “add CS2103 V0.1 (20/3/16 23:59) @online #work”

Parser creates **Command** object, noting that it is a deadline task as it only consists of date and time, hence initialising taskDescription, Location, Date, Time, Tags.

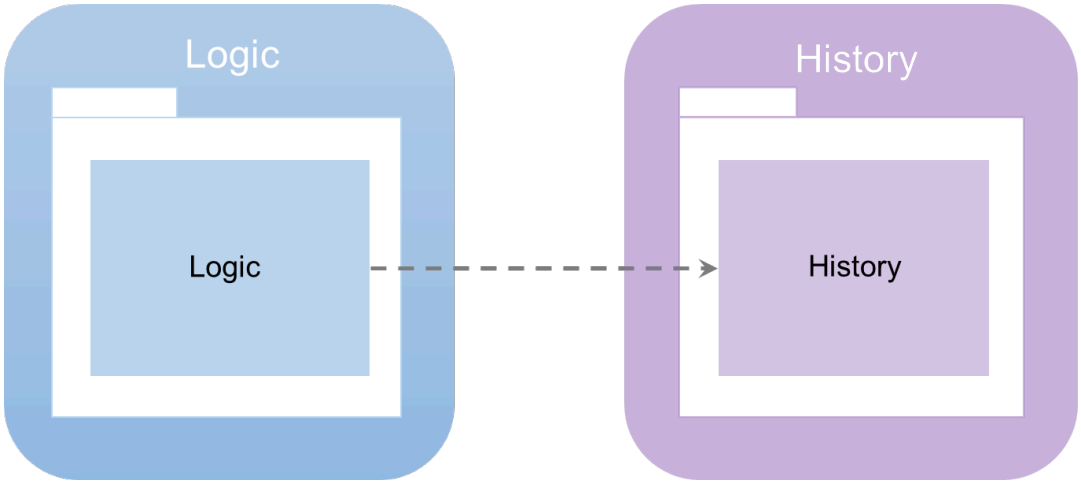
Command deadLineTask = new CommandAddDeadlineTask (taskDescription, location, endDateTime, tagLists);

return deadLineTask;

History component

6

History is where the previous display objects are stored in case user wants to undo/redo certain commands.

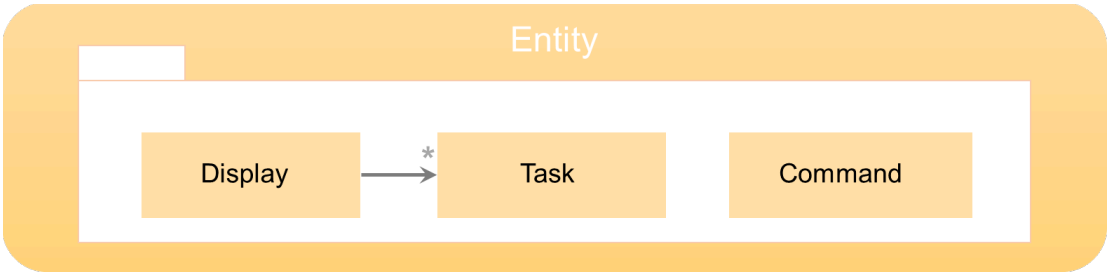


The History Class is where the previous display objects are stored in case user wants to undo/redo certain commands.

Return Type	Method and Description
void	**saveDisplay(Display display)** : Saves the argument display into a list of Display objects.
Display	**getDisplay(int offset)** : This method is to facilitate the undo/redo commands. An offset of -1 will return the previous display while an offset of 1 will return the next display (only possible if user has previous issued an undo command)

Entity component

7



The Entity component contains the classes that represent the various elements that are required in managing user's tasks. Logic manipulates these classes and all the other components will use the data within these classes to do their jobs.

Command Class

This class represents user's commands. When user enter some commands, Parser will analyze it and create a Command object. It's a generic object and has ten types of children: CommandAddDeadlineTask, CommandAddEvent, CommandAddFloatTask, CommandAddReserved, CommandDelete, CommandInvalid, CommandRedo, CommandShow, CommandUndo, CommandUpdate.

Display Class

This class represents information that needs to be displayed on the app. Every Display object will have several lists of different tasks and a message.

Task Class

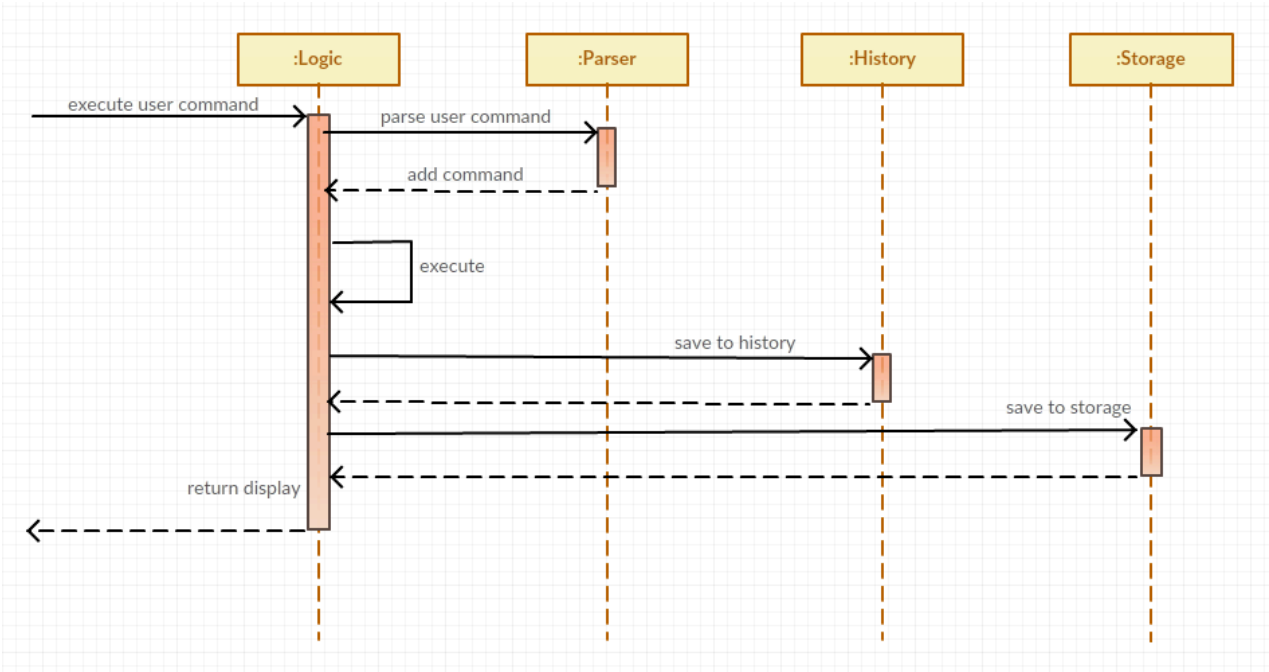
This class represents user's tasks. It's a generic object and has four types of children: TaskDeadline, TaskEvent, TaskFloat, TaskRserved.

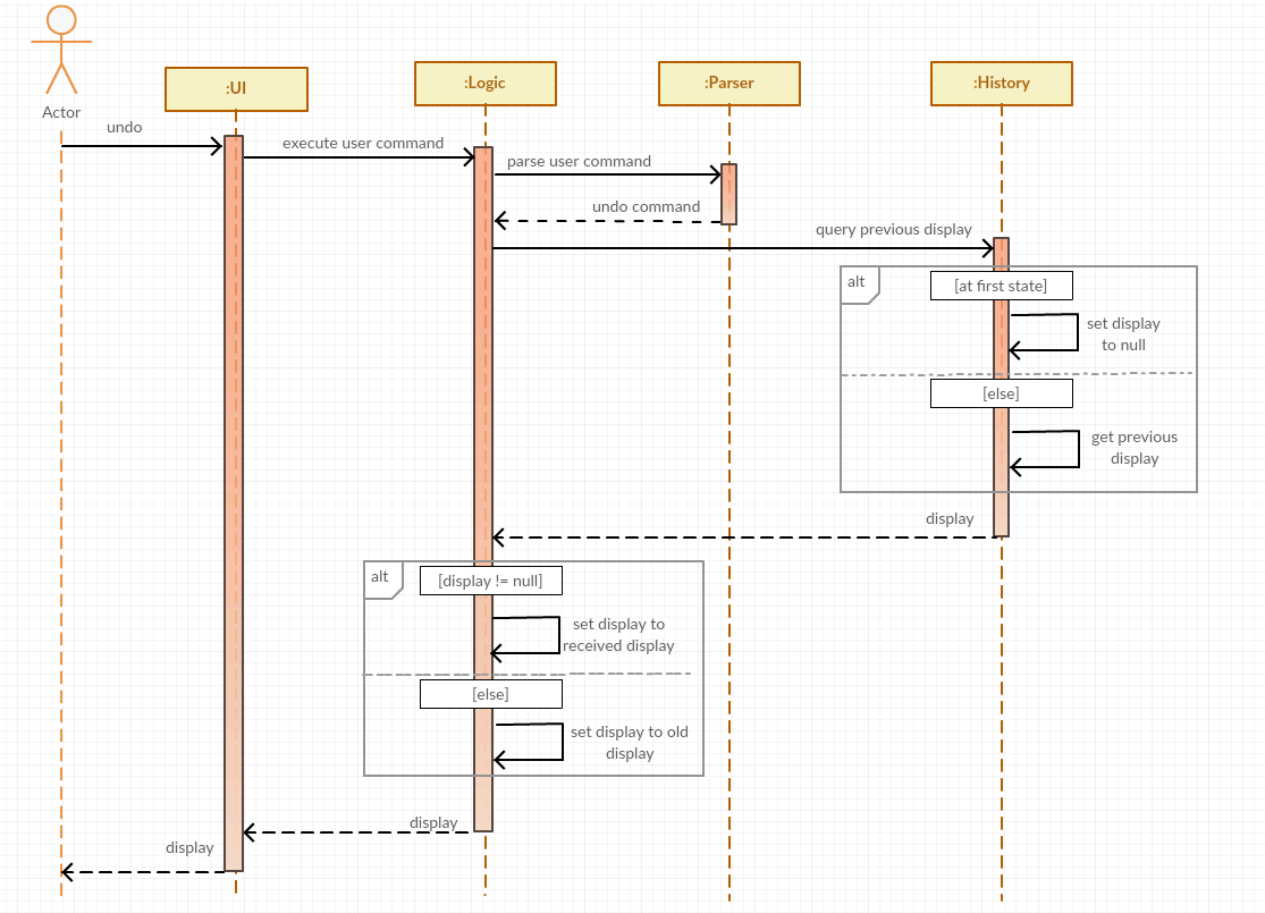
Sequence Diagram

8

Sequence diagram for CRUD commands

The figure below illustrates the process of a typical add command:

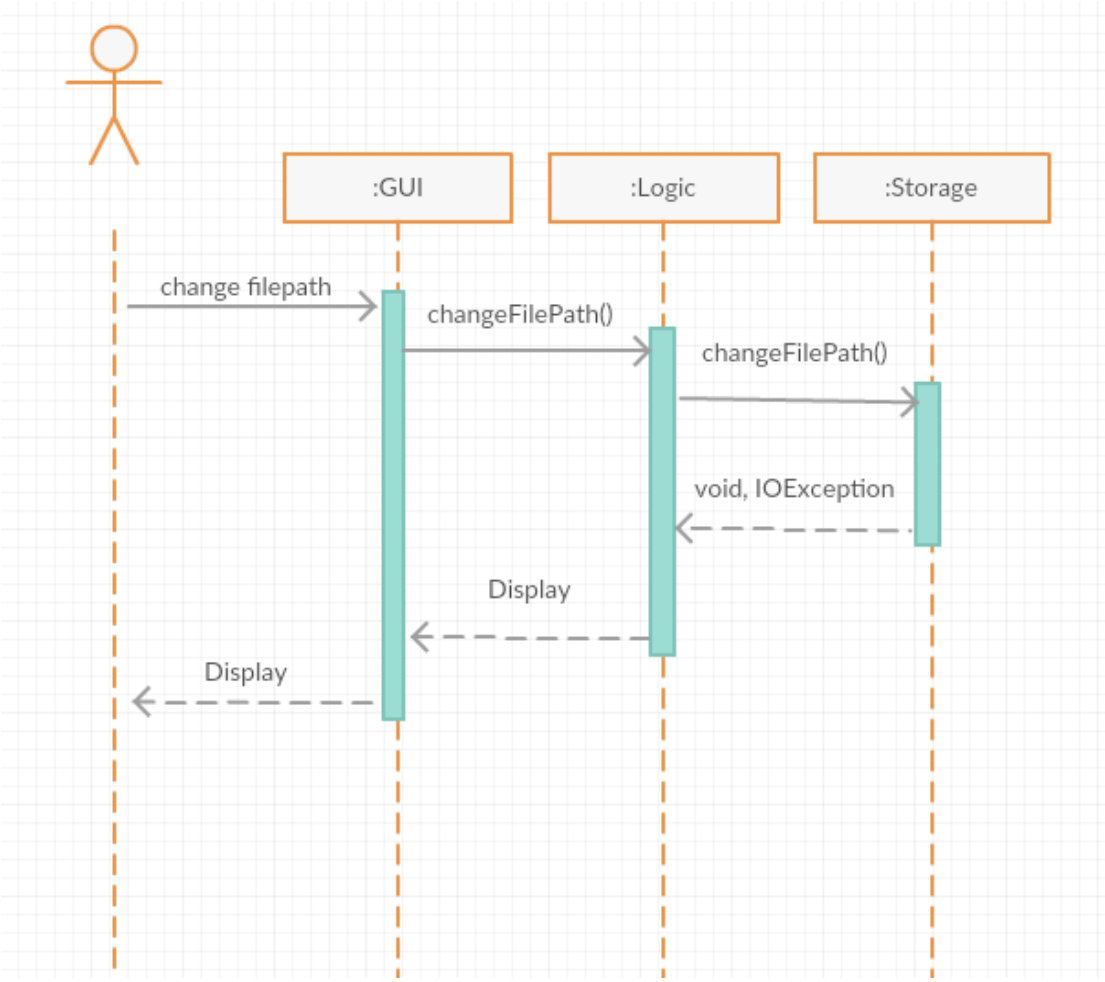




Sequence diagram for a typical undo command

As seen above, the logic first calls the parser to parse the command. After which during execution of the command, logic will query history for previous display. History will then check to see if there are any previous displays available. If there are available displays, History retrieves it and sends it to logic. If not, History sends back a null display. Logic then proceeds to check if the display received is valid. If valid display is received, logic simply sends it to UI to be displayed. If not, Logic will retrieve the old display and send it to UI.

Sequence diagram for change file path



Testing

9

The automated testing framework used for **J.Listee** is JUnit. We used Equivalence Partitioning as well as checking boundary cases in order to have efficient and effecting testing. Inputs within the same partition are expected to behave in the same way, so it is inefficient and redundant to test all the cases within a single partition.

Below is a sample of the Integration Test.

```java
/*************************
 * Input Validation Tests *
*************************/

@Test
public void testInvalidCommand() {
    display = Logic.executeUserCommand("This is an invalid command.");
    String expected = "You have specified an invalid command";
    String actual = display.getMessage();
    assertEquals(expected, actual);
}

@Test
public void testEmptyCommand() {
    display = Logic.executeUserCommand("");
    String expected = "You have specified an invalid command";
    String actual = display.getMessage();
    assertEquals(expected, actual);
}

/**********************
 * Adding Tasks Tests *
 **********************/

@Test
public void testAddFloating() {
    display = Logic.executeUserCommand("add Floating Test @NUS #tag");
    String expected = "added: \"Floating Test\"";
    String actual = display.getMessage();

    assertEquals(expected, actual);
    assertTrue(display.toString().contains("floatTasks=["
        + "Description: Floating Test\r\n"
        + "Location: NUS\r\n"
        + "Tags: #tag\r\n\r\n]"));
}

@Test
public void testAddDeadline() {
    display = Logic.executeUserCommand("add Deadline Test due 14th Apr 3pm @NUS #tag");
    String expected = "added: \"Deadline Test\"";
    String actual = display.getMessage();

    assertEquals(expected, actual);
    assertTrue(display.toString().contains("deadlineTasks=["
        + "Description: Deadline Test\r\n"
        + "Deadline: 14/04/16 15:00\r\n"
        + "Location: NUS\r\n"
        + "Tags: #tag\r\n\r\n]"));
}
```

Acknowledgements

10

1. **J. Listee** transfers data between Java and JavaScript using JSON (JavaScript Object Notation), which is a lightweight data-interchange format.
2. **J.Listee** use JKeymaster to enable quick launch by shortcut. JKeyMaster is a java library that provides single interface to register Global Hotkeys for several platforms.
3. **J.Listee** uses Natty to parse dates for natural input command.