

Urgenda

Urgenda

File Edit View Select Help

Showing ALL TASKS





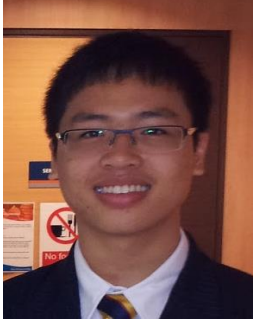
No.	Task	Start	End
Overdue Tasks			
1	Overdue task	Today 10:28PM	to Today 10:28PM
Today's Tasks			
2	★ Today Important task deadl...		by Today 10:28PM
3	Today task long long long l...	Today 10:28PM	to Today 10:28PM
4	Today Detailed task long long long Today Detailed location	Today 10:28PM	to Today 10:28PM Date Added: 19/03/16 10:28:06 Date Modified: 19/03/16 10:28:06
Other Tasks			
5	★ Important task	Today 10:28PM	to Today 10:28PM
6	Detailed task long long long long long long long long	Today 10:28PM	to Today 10:28PM Date Added: 19/03/16 10:28:06 Date Modified: 19/03/16 10:28:06

Welcome to Urgenda! Your task manager is ready for use.
Press ALT + F1 for help.

Type your to-do here!

Supervisor: Ng Hui Xian Lynnette

Extra feature: Flexi Commands

				
Koh Kim Wee	Ang Kang Soon	Cheng Tze Jin	Tan Hui Kee Joanne	Vo Hoang Khai
Team lead	Code Quality	Integration	Documentation	Eclipse Expert
Testing	Integration	Code Quality	Scheduling and Tracking	Testing

Credits

- Gson
- PrettyTime

Table of Contents

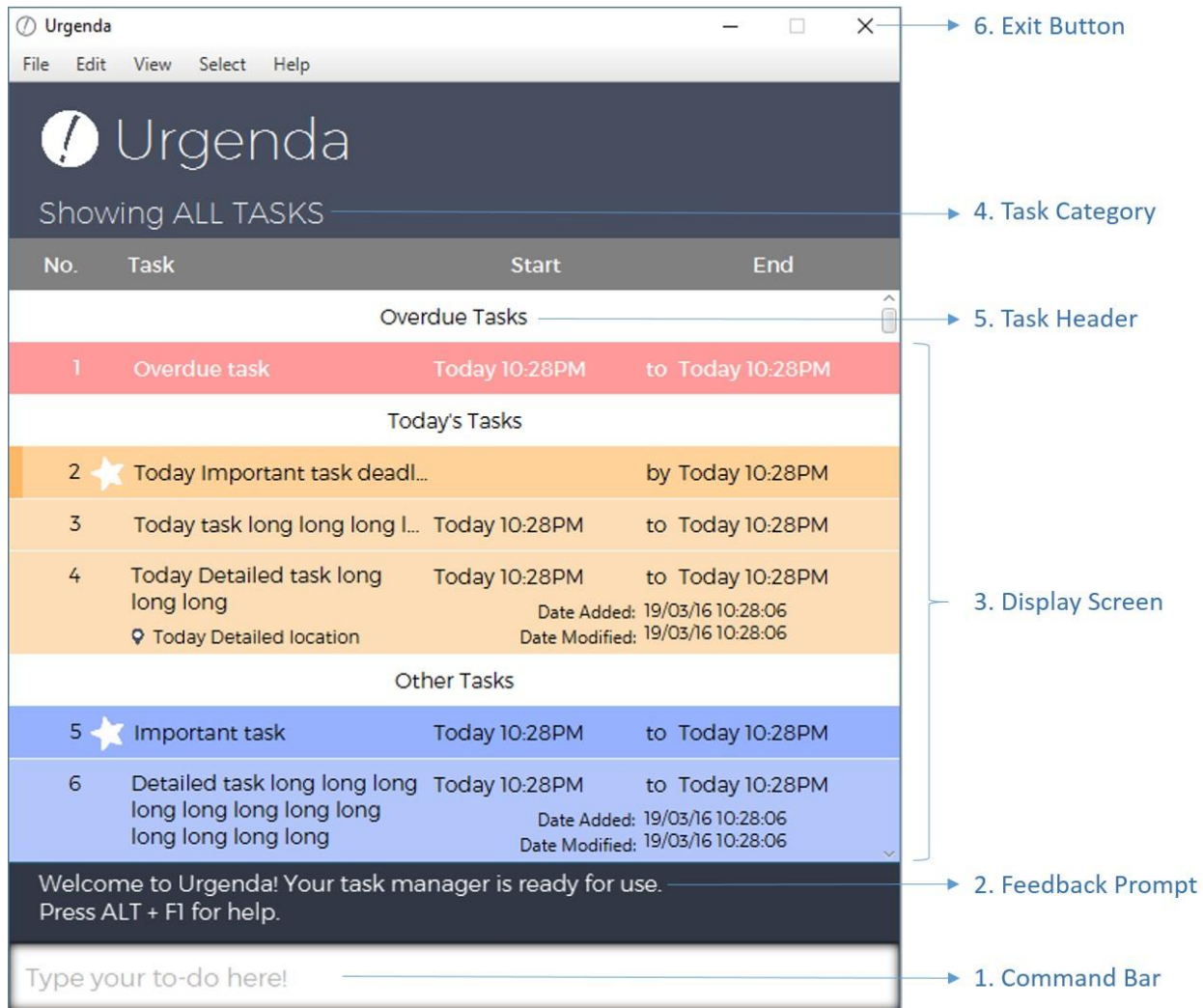
User Guide	5
Getting Started	5
Feature Details	6
Add	6
Delete	6
Mark as Completed	7
Edit.....	7
Search	8
Exit	8
Advanced Features	9
Show more details	9
Archive.....	9
Undo/Redo	10
Block Multiple Timeslot.....	10
Prioritise	11
Shortcuts.....	11
Full list of all possible commands and command tags	11
Developer Guide.....	13
Architecture.....	13
UI Component	14
Main Class.....	14
MainController Class	15
DisplayController Class	15
TaskController Class	15
TaskDetailsController Class	15
FXML Files	15
Logic Component.....	16
Logic Class.....	18
Command Component	18
Parser Component.....	20

Parser Class.....	21
Storage Component.....	22
Storage Class.....	23
Appendix A: User stories. As a user,	27
Appendix B: Non Functional Requirements	30
Appendix C: Product survey	31

User Guide

This guide helps you with understanding how to use Urgenda effectively as a task manager for your daily needs.

Getting Started



1. User input/command bar: Enter your to-dos here easily using the given Command Tags.
2. Feedback prompt: The outcome to any changes that you make to your tasks is shown here.
3. Display screen: Your tasks are categorised and displayed here according to time and priority by default.
4. Task category: This shows the type or category that the displayed tasks belong to (e.g. ALL TASKS, OVERDUE TASKS, #Assignment, etc.)
5. Task header: This indicates each sub-category for different tasks.
6. Exit button: Click to exit the program. Alternatively, type exit in the command bar to exit.

Feature Details

Add

To create a new task, **Add** or **Create** are the command words, with the task name following thereafter, ie "Dinner with Mum"

Use **Command Tags** to add in details for the task, "**at**" for time, "**@**" for location, etc. For the **full list of Command Tags**, refer to Shortcuts

Tasks are separated into 3 categories, Events, Deadlines and Untimed (Floatings).

- Event: Task is given with a start time and end time. Example: **Add Dinner with Mum at 23/3/2016 7:00pm to 8:00pm**
 - If only a start time is given, then end time will automatically be set to 1 hour after the start time.
- Deadline: Task is given with only an end time. Example: **Add Return home by 23/3/2016 7:00pm**
- Untimed (Floating): Task is given with no start time and end time. Example: **Add Dinner with Mum**

No.	Task	Start	End
Other Tasks			
1	abc		
2	task		
"task" added			
add Dinner with Mum at 23/3/2016 7:00pm to 8:00pm			

Figure 1: Adding a task by typing into the command bar

No.	Task	Start	End
Other Tasks			
1	Dinner with Mum	23 Mar 7:00PM	to 23 Mar 8:00PM
2	abc		
3	task		
"Dinner with Mum" on 23/3, 19:00 - 20:00 added			

Figure 2: Added task is displayed on the window with the selector on it

Delete

To delete an existing task, **Delete**, **Del**, **Erase** or **Remove** are the command words that can be used to delete a task. Note that deleting a task *IS NOT* completing a task.

Specify the task number, the task description or simply highlight the task to be deleted.

Example:

- **Delete Dinner with Mum**
- **Delete 4**

No.	Task	Start	End
Other Tasks			
1	cs2103 v0.1		by Tomorrow 11:59PM
2	Company's D&D	23 Mar 7:00PM	to 23 Mar 8:00PM
3	Dinner with Mum		
"housekeeping" removed			
Delete Dinner with Mum			

Figure 3: Deleting the task by typing the description into the command bar.

No.	Task	Start	End
Other Tasks			
1	cs2103 v0.1		by Tomorrow 11:59PM
2	Company's D&D	23 Mar 7:00PM	to 23 Mar 8:00PM
"Dinner with Mum" removed			

Figure 4: Task is deleted and remove from display.

Mark as Completed

To mark a task as done, **Done**, **Completed**, **Do**, **Mark**, **Finish**, **Fin** are the command words that can be used. Similar to Delete, specify the task number, the task description or simply highlight the task to be marked as completed.

Example:

- **Mark Dinner with Mum**
- **Done 3**

No.	Task	Start	End
Other Tasks			
1	cs2013 v0.1		by Tomorrow 11:59PM
2	Return home		by 23 Mar 7:00PM
3	Dinner with Mum		
"task" removed			
mark Dinner with Mum			

Figure 5: Marking the task by typing the description into the command bar.

No.	Task	Start	End
Other Tasks			
1	cs2013 v0.1		by Tomorrow 11:59PM
2	Return home		by 23 Mar 7:00PM
Done "Dinner with Mum!"			

Figure 6: Task is marked as done, stored in archives and removed from display.

Edit

To edit an existing task, **Edit**, **Update**, **Change** and **Mod** are the commands that can be used.

Use **Command Tags** to change details for the task, "at" for time, "@" for location, etc. For the **full list of Command Tags**, refer to Shortcuts

If the task already has those details, then the new details will overwrite the old details.

Similar to Delete and Mark as completed, specify the task number, the task description or simply highlight the task to be edited.

Example:

- **_Change Dinner with Mum @ NEX**
- **_Edit 4 Dinner with Mum and Dad**

No.	Task	Start	End
Other Tasks			
1	cs2103 v0.1		by Tomorrow 11:59PM
2	company's d&d	23 Mar 7:00PM	to 23 Mar 8:00PM
3	dinner with mum		
Welcome to Urgenda! Your task manager is ready for use. Press ALT + F1 if you need help.			
edit 3 at 3/21 6.30pm			

Figure 7: Editing task no. 3 and adding time to it

No.	Task	Start	End
Other Tasks			
1	cs2103 v0.1		by Tomorrow 11:59PM
2	dinner with mum	21 Mar 6:30PM	to 21 Mar 7:30PM
3	company's d&d	23 Mar 7:00PM	to 23 Mar 8:00PM
"dinner with mum" has been edited to "dinner with mum" on 21/3. 18:30 - 19:30			

Figure 8: Task no. 3 had time added to it, displayed accordingly in order of the timings.

Search

To search for a word, **Search**, **Find**, **Show**, **View** or **List** are the commands that can be used.

searches for the tags, and thereafter displaying the tasks.

- When searching for **#cow**, tasks with tags **#cows** and **#cowmilk** will also appear in the search result.

When searching for a date, all tasks with this particular date set as the deadline will be displayed. When

searching for a word or phrase, all tasks that contain this particular work or phrase in their task description(s) will be displayed. When searching for time, time block with assigned tasks will be displayed.

Entering the search commands alone will simply show all existing tasks.

Example: **Search boss**

- *Report to boss by 4/4/2016 3:00pm*
- *Meeting with boss at 6/4/2016 2:00pm to 5:00pm*

No.	Task	Start	End
Other Tasks			
1	cs2103 v0.1	by Tomorrow 11:59PM	
2	dinner with mum	21 Mar 6:30PM	to 21 Mar 7:30PM
3	company's d&d	23 Mar 7:00PM	to 23 Mar 8:00PM
4	meeting with boss	25 Mar 10:00AM	to 25 Mar 12:00PM
5	company retreat	08 Apr 10:00AM	to 10 Apr 6:00PM
Welcome to Urgenda! Your task manager is ready for use. Press ALT + F1 if you need help.			
search company			

Figure 9: Searching for tasks with the word “company” from among all the tasks

No.	Task	Start	End
Other Tasks			
1	company's d&d	23 Mar 7:00PM	to 23 Mar 8:00PM
2	company retreat	08 Apr 10:00AM	to 10 Apr 6:00PM
These are all the tasks found containing "company"			

Figure 10: Tasks that contains the word “company” is displayed for the user.

Exit

To exit Urgenda, click the top right exit button, or type **exit**

Advanced Features

Show more details

Showmore is a specific command to expand a specific task for the user to view more details about that task, such as the location, the tags for that task, or for any other additional details.

No.	Task	Start	End
Other Tasks			
1	★ cs2103 v0.1	by Tomorrow 11:59PM	
2	dinner with mum	21 Mar 6:30PM	to 21 Mar 7:30PM
3	company's d&d	23 Mar 7:00PM	to 23 Mar 8:00PM
4	meeting with boss	25 Mar 10:00AM	to 25 Mar 12:00PM
5	company retreat	08 Apr 10:00AM	to 10 Apr 6:00PM
Welcome to Urgenda! Your task manager is ready for use. Press ALT + F1 if you need help.			
showmore			

Figure 11: Requesting for more details to be shown for the task no. 1, where the selector is at.

No.	Task	Start	End
Other Tasks			
1	★ cs2103 v0.1	by Tomorrow 11:59PM	
	📍	Date Added: 19/03/16 06:41:46 Date Modified: 19/03/16 06:41:46	
2	dinner with mum	21 Mar 6:30PM	to 21 Mar 7:30PM
3	company's d&d	23 Mar 7:00PM	to 23 Mar 8:00PM
4	meeting with boss	25 Mar 10:00AM	to 25 Mar 12:00PM
5	company retreat	08 Apr 10:00AM	to 10 Apr 6:00PM
Showing more details for "cs2103 v0.1"			

Figure 12: More details shown for the first task.

Archive

To show previously completed tasks, the commands are **Archive** and **Showarchive**

Undo/Redo

Undo and **Redo** are classic features, allowing you to undo the previous action taken. This will result in any changes made to certain task(s) being restored. A maximum of 10 consecutive actions are possible for both **undo** and **redo**. For example:

1. **delete** *Dinner with Mom* (Task with Dinner with Mom is deleted)
2. **undo** (Task Dinner with Mom is restored with all its previous details)

However, when undo is implemented at least once, any new commands (other than **undo** and **redo**) will discard all saved redo actions after the new action has been implemented. For example:

3. **delete** *Fix lightbulb* (Task “fix lightbulb” is deleted)
4. **undo** (Task “fix lightbulb” is restored)
5. **add** *Meeting with boss* (Task “Meeting with boss” is added)
6. **redo** (No tasks to redo, previously undone actions are cleared.)

No.	Task	Start	End
Other Tasks			
1	★ cs2103 v0.1		by Tomorrow 11:59PM
"cs2103 v0.1" by 20/3, 23:59 marked as important!			
undo			

Figure 13: Previously, task no.1 is marked as important. Command for undo is given

No.	Task	Start	End
Other Tasks			
1	cs2103 v0.1		by Tomorrow 11:59PM
Undo: "cs2103 v0.1" by 20/3, 23:59 unmarked as important			

Figure 14: Task no. 1 is returned to its previous state of not being marked as important.

No.	Task	Start	End
Other Tasks			
1	cs2103 v0.1		by Tomorrow 11:59PM
Undo: "cs2103 v0.1" by 20/3, 23:59 unmarked as important			
redo			

Figure 15: Command for redo is given to re-mark task no. 1 as important.

No.	Task	Start	End
Other Tasks			
1	★ cs2103 v0.1		by Tomorrow 11:59PM
Redo: "cs2103 v0.1" by 20/3, 23:59 marked as important!			

Figure 16: Task no. 1 is returned to its original state of being marked as important.

Block Multiple Timeslot

In the event that you have a task with an unconfirmed timing, **Block**, **Alloc** and **Reserve** are commands to allow you to enter multiple timeslots for this task to prevent clashes in future.

- Example: **block** *Dental Appointment at 10am-12pm, 3pm-5pm*

Release and **Unblock** are keywords that you can use to release certain reserved timeslots that for the indicated task.

Alternatively, the task editing keywords (**edit**, **update**, **change** and **mod**) can be used to confirm the timing of the task and release all other reserved timeslots.

Prioritise

This feature allows you to mark certain tasks as important. These tasks will be displayed with a logo next to the task.

Urgent, Important, Impt, Pri or **Pin** are the commands to prioritise a task. Example:

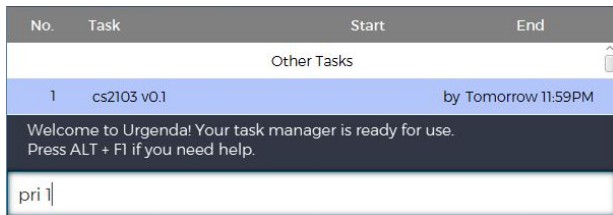


Figure 17: Command to mark task no. 1 is as given.

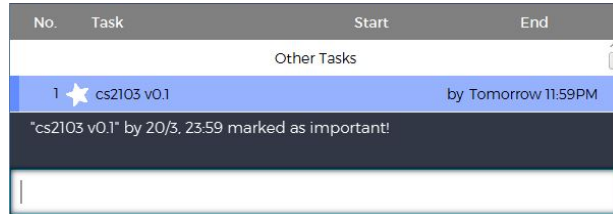


Figure 18: Task no. 1 is displayed with the star to show as important

Shortcuts

- Ctrl + Z / Ctrl + Y : Undoing/Redoing what was previously written in the command bar
- Ctrl + D : Displays that default view of showing all tasks
- ▲ : Cycle through previous commands from command history
- ▼ : Cycle through next commands from command history
- Ctrl + ▲, Ctrl + ▼ : Changes the highlighted task on graphical display
- Alt + F4 : Exits Urgenda
- Ctrl + Alt + D : Quick launch of Urgenda

Full list of all possible commands and command tags

Action	Command Words to be used	Command Tags to be used If the tags have an * before it, they are optional.
Creating a new task	□ Add, Create	<ul style="list-style-type: none"> • Task Description • *Time: at • *Location: @ • *Date: tomorrow, today, DD/MM, DD Month, next week/day • *Tag: #
Deleting a task	□ Delete, del, erase, remove	<ul style="list-style-type: none"> • Task Description • Task Number
Completing a task	□ Done, Completed, Mark, Finish, Fin	<ul style="list-style-type: none"> • Task Description • Task Number

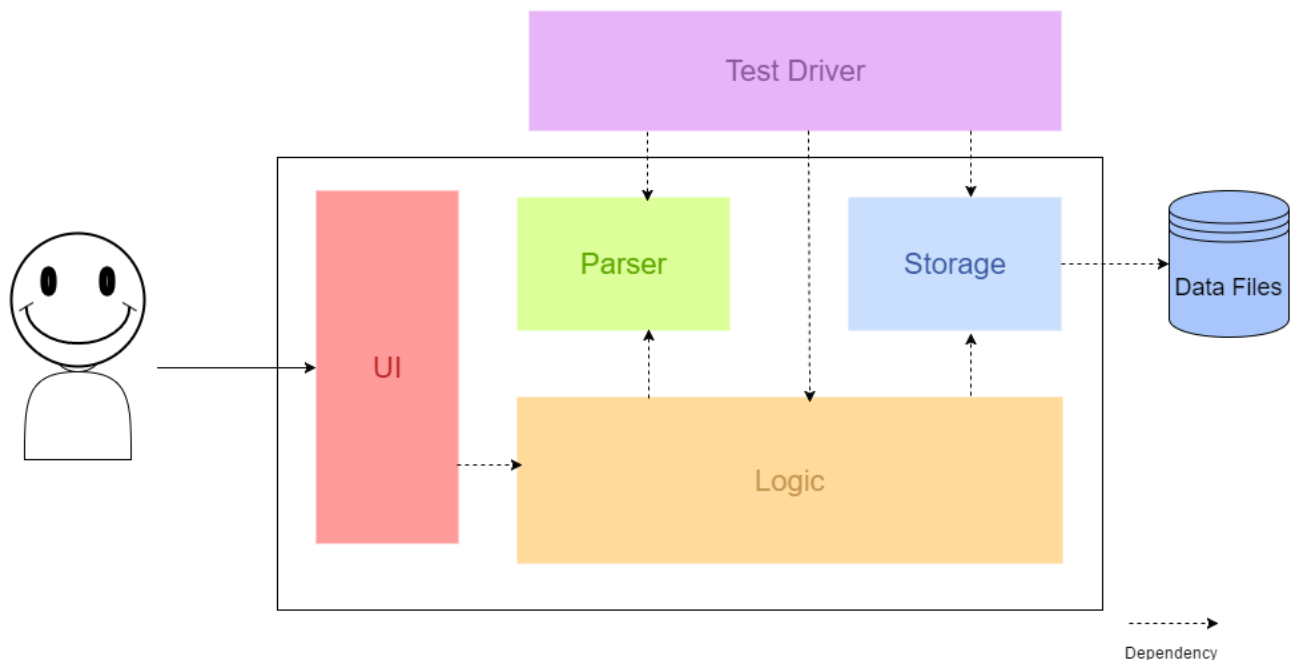
Editing a task	<input type="checkbox"/> Edit, Update, Change, Mod	<ul style="list-style-type: none"> • Task Description • Task Number <input type="checkbox"/> *Time: at • *Location: @ • *Date: tomorrow, today, DD/MM, DD Month, next week/day • *Tag: #
Searching for a task	<input type="checkbox"/> Search, Find, Show, View, List, #	<ul style="list-style-type: none"> • *Task Description • *Time: at • *Location: @ • *Date: tomorrow, today, DD/MM, DD Month, next week/day • *Tag: #
Exiting the program	<input type="checkbox"/> Exit	
Show details	<input type="checkbox"/> Showmore	<ul style="list-style-type: none"> • Task Description • Task Number
Accessing archives	<input type="checkbox"/> Archive, Showarchive	
Undo/Redo an action	<input type="checkbox"/> Undo, Redo	
Prioritisation	<input type="checkbox"/> Urgent, Important, Pri, Impt, Pin	<ul style="list-style-type: none"> • Task Description • Task Number
Adding multiple timeslots	<input type="checkbox"/> Block, Alloc, Reserve	<ul style="list-style-type: none"> • Task Description *Time: at • *Location: @ • *Date: tomorrow, today, DD/MM, DD Month, next week/day • *Tag: # • Additional timeslots: ,
Releasing multiple timeslots	<input type="checkbox"/> Release, Unblock	<ul style="list-style-type: none"> • Task Description • Task Number • At least 1 timeslot

Developer Guide

Urgenda is a text-based task manager designed for users who are quick on the keyboard, or generally prefer using the keyboard over mouseclicks. It is a Java application that has a main GUI for users to interact mainly through typing.

This guide describes the implementation of Urgenda from a top-down approach, going from the big picture down to the small details, starting from the UI through which the user interacts with, to the Storage where the files are stored. This guide intends to allow easy assimilation of anyone who would like to contribute and add on to it.

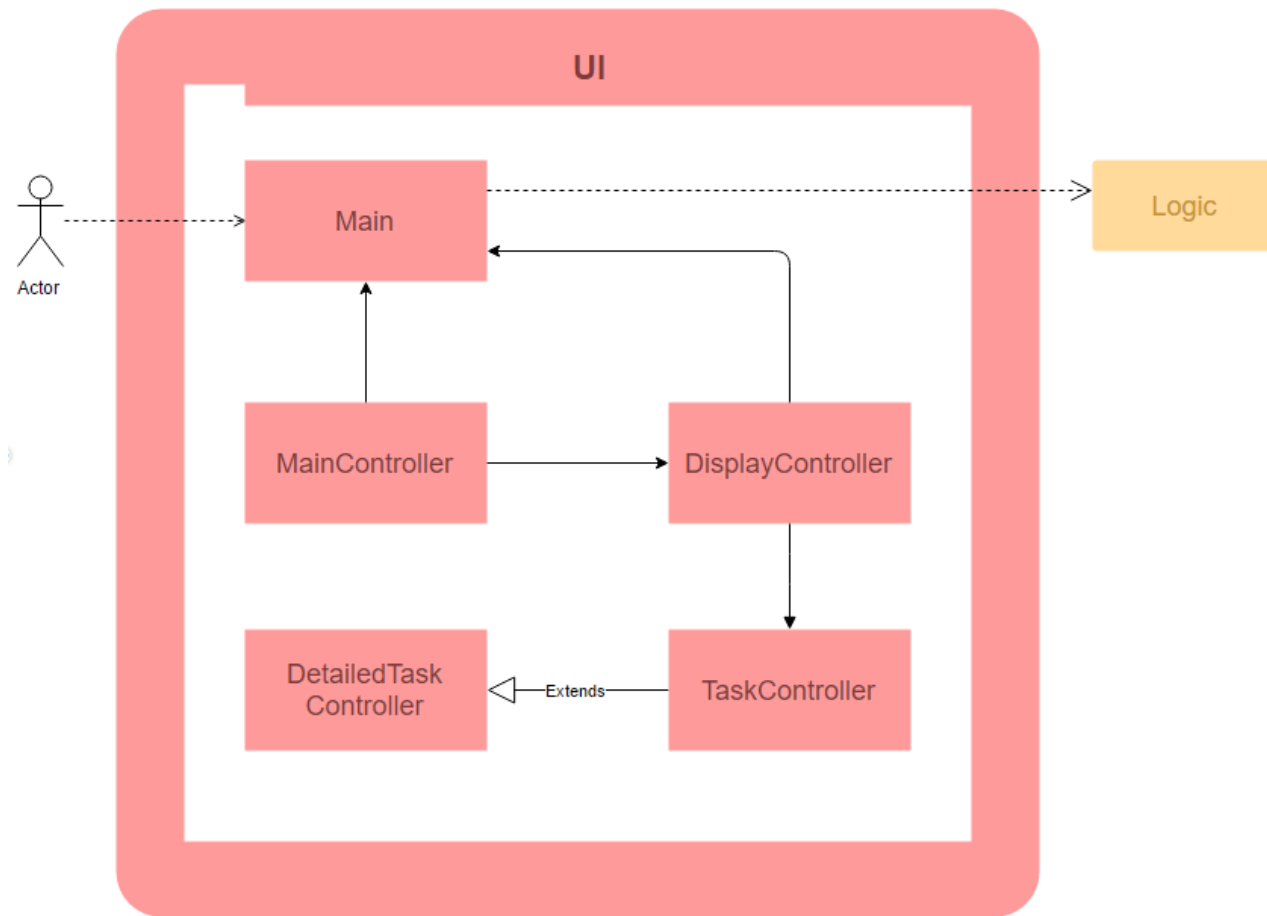
Architecture



Urgenda consists of 4 main components, with the interaction for the user through the UI

1. The **UI** component uses JavaFX with FXML files for displaying the UI to the user, with the controllers in java files to control the display
2. The **Logic** component is the main brainchild of Urgenda. Every other component interacts with **Logic** to provide information and details that are required for Urgenda to run smoothly
3. The **Parser** component is the parser of Urgenda, parsing natural language used by a typical user into processable variables and attributes for **Logic** to utilise for maximum effectiveness.
4. The **Storage** component keeps all the Tasks, data and settings in textfiles on the user's computer, allowing Urgenda to instantly restore all the user's previous tasks when starting up Urgenda every time.

UI Component



The UI component is the sole component responsible for creating and maintaining Urgenda's graphical user interface. It is also the only component that directly handles any user interaction with Urgenda. User input is mainly through the command line input through the input bar at the bottom of the window. Minor click functionalities are also enabled to enhance the user experience, but do not provide any extra functions beyond that available through command line input.

JavaFX and CSS are used to setup the layout and design for the graphical user interface.

Main Class

The Main class acts as the intermediary between the controllers for all UI components with the back-end Logic component. This abstracts all command execution flow from the UI class.

MainController Class

The MainController class handles all user interaction with Urgenda, for both command line input as well as click inputs. Several event handlers set in this class, including `commandLineListener`, will call the UI class's `handleCommandLine` method as pass the command line String as the argument. Other event handlers include accelerated keyboard shortcuts for undo, redo, and help functions. This class also displays feedback from any user interaction to the user.

DisplayController Class

The DisplayController class handles the display area where relevant tasks are displayed to the user. The display area is set up using the `setDisplay` method, where the task panels are created and added to the display according to the task list argument passed. This class also handles the setup of the design for tasks with different task types and tasks to show details for.

TaskController Class

The TaskController class handles the set up of the panel for one single task. The task index as ordered in the display area, description, prioritisation, as well as relevant dates and times, are shown on the panel. This class also applies the task design style for the task, and formats the dates and times with respect to the current date and time.

TaskDetailsController Class

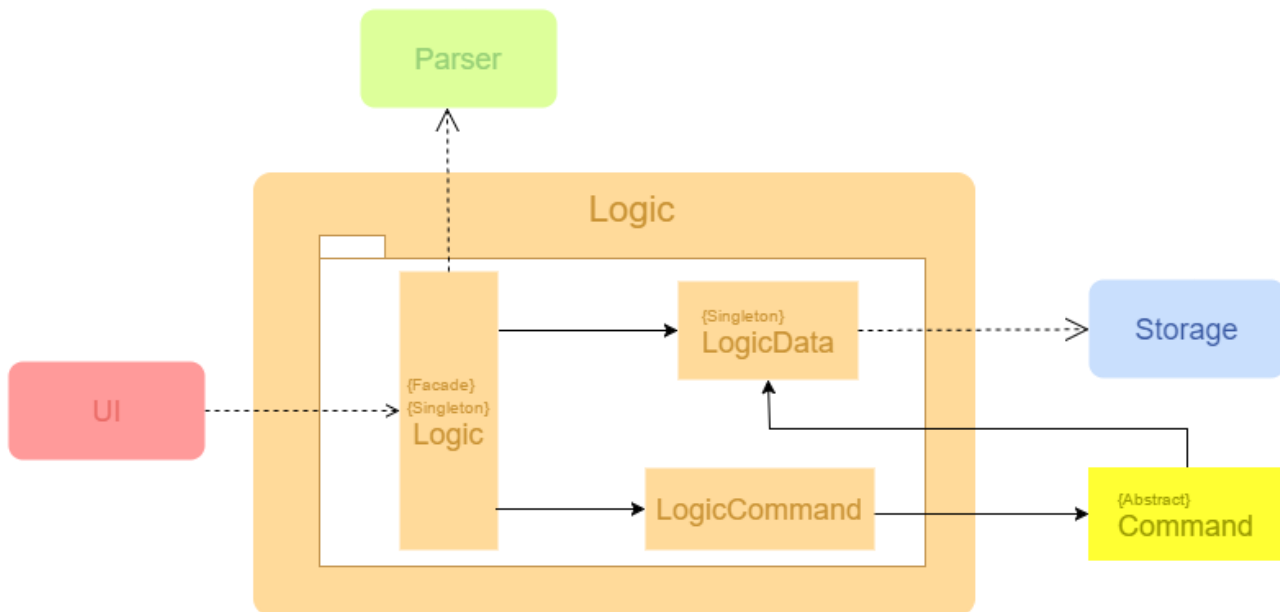
The TaskDetailsController class is an extended class from TaskController. This class is invoked to create the task panel when the relevant task is required to show more details, such as location, date created and modified, and long descriptions.

FXML Files

Each controller class will load an FXML file, which sets the layout of the UI window according to the design set by these FXML files. There are a total of 5 FXML files: `Main.fxml`, `DisplayView.fxml`, `TaskView.fxml`, and `DetailedTaskView.fxml` for the main UI window, and `HelpSplash.fxml` for setting the help window. FXML files can be opened and edited using SceneBuilder 2.0, which is an official visual layout tool for JavaFX applications from Oracle. More information about SceneBuilder 2.0 can be found at:

(<http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html>)

Logic Component



The Logic component is accessible through the Logic class using the facade pattern, in which it is in charge of handling the execution of user inputs from the UI component. This component only relies on the Parser component and Storage component and works independently from the UI component. Furthermore, the Command component is part of the Logic of Urgenda which encompasses the functionalities of the different commands given by the user.

The table below shows the classes in Logic component and their functions:

Class	Function
Logic (Facade)	Main handler for external calls from other components. Also has the Singleton pattern as there should always be only one Logic handling the processes in Urgenda.
LogicData	Class in Logic component that stores the Tasks temporarily when Urgenda is running. Most data manipulation and edits are done through LogicData. It is also responsible for generation of the current state. Has the Singleton pattern as well to ensure that all data manipulation is done on the only LogicData.
Class	Function
LogicCommand	Class where the Commands are being stored in the Logic component. Execution of commands as well as undo/redo of these commands will be carried out by LogicCommand.

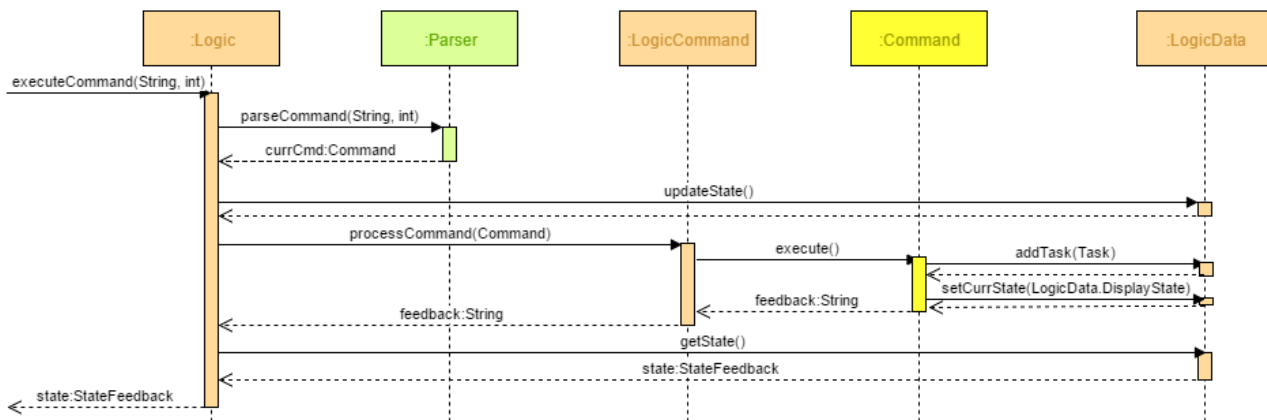
Furthermore, the table below shows the notable API for usage of Logic:

Method	Return type and function
executeCommand(String command, int index)	Returns a StateFeedback object which consists of the execution feedback, as well as the current state of the task objects relevant to display for UI. The method will be used to process all inputs by the user.
retrieveStartupState()	Returns a StateFeedback object of the system. This method is for the initial startup of Urgenda in setting up the components as well as retrieval of previously saved tasks.
displayHelp()	Returns a String which consists of the help manual of Urgenda. This method is used for the request of Urgenda's help manual.
getCurrentSaveDirectory()	Returns a String of the current location where the data is being saved on the user's computer.

Logic Class

The Logic class contains the methods that handle the core functionality of Urgenda. It can be thought of as the "processor" of Urgenda. User inputs are passed to `executeCommand(String, int)` to determine the corresponding command object based on the user input by the Parser component.

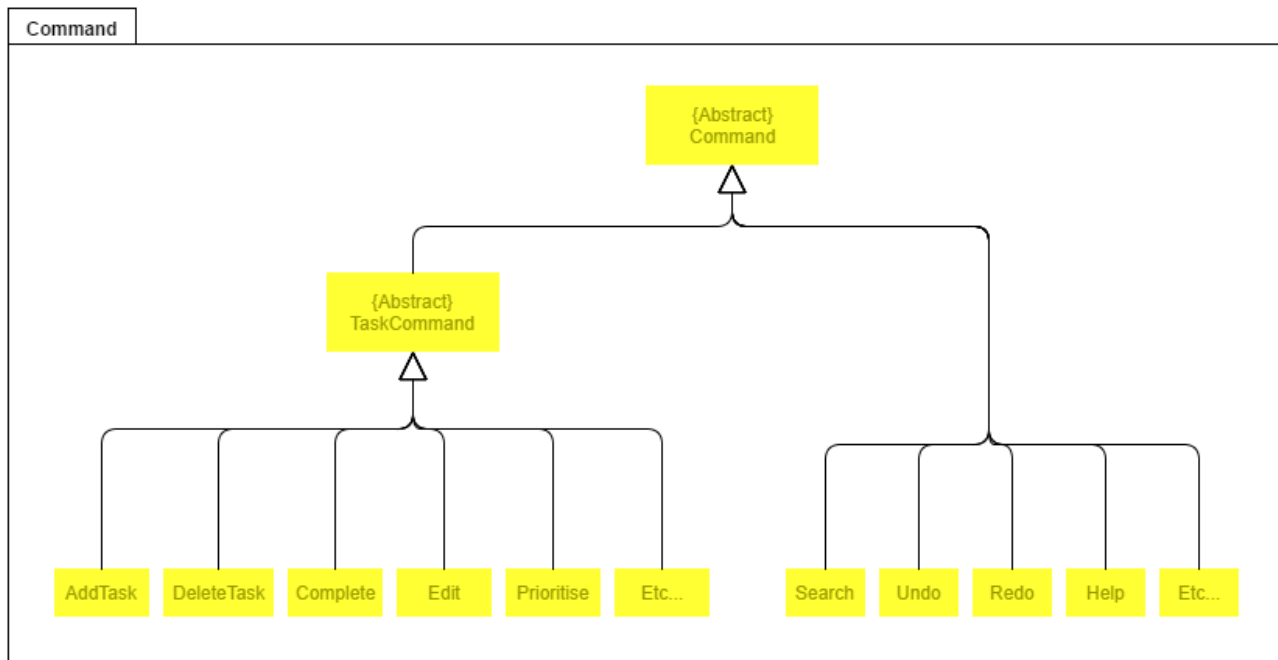
A generic example of the process flow in Logic can be seen below:



After knowing the type of command, Logic retrieves the updated state and data per launch time from LogicData via the `updateState()` method call. After which the command object will be passed to LogicCommand for process through the `processCommand(Command)` method call. The command will then be executed, and LogicData will update its relevant fields. In the case of adding a task, the task will be added to task list via the `addTask(Task)` method call and the display state will be updated correspondingly. LogicData maintains a temporary set of data same as that displayed to the user per launch time so as to facilitate number pointing of task and reduce dependency with Storage component (e.g. when user inputs delete 4, Logic is able to determine which is task 4 without having to call Storage). Storage component will then store the data to ensure no loss of user data upon unintentional early termination of Urgenda Program. More details of the storing procedure are mentioned in the Storage section. The `executeCommand(String)` method will then return the appropriate feedback to its caller method. The caller method can then decide how to update the user interface.

Command Component

The Command component is part of the Logic processing in Urgenda, where the specific commands are being executed by the program. It mainly consists the specific execution instructions of the individual command types.



Here is the abstract method that is present in Command class.

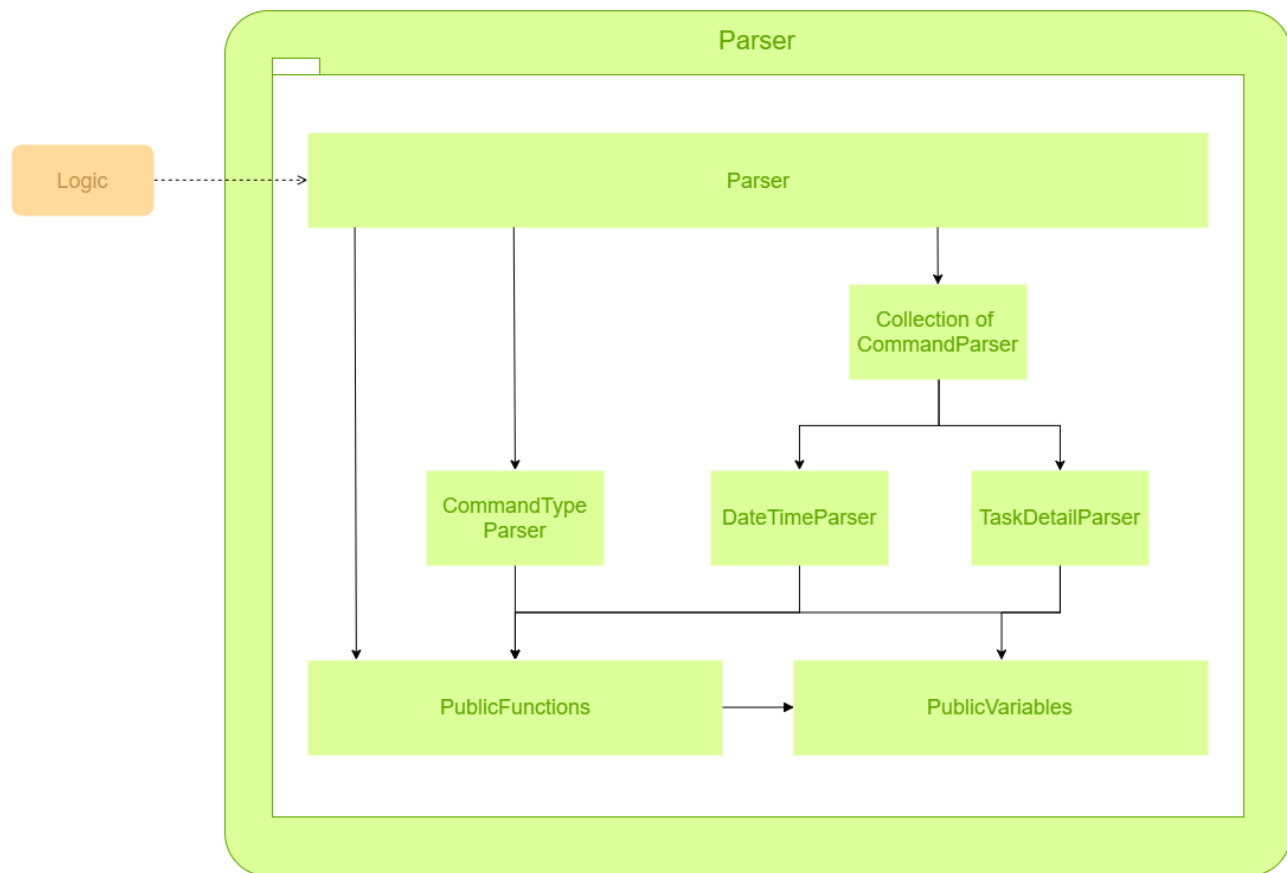
Method	Return type and function
execute()	Returns a String which represents the feedback of the command being executed. This is the abstract method for the generic execution of Commands.

Additionally, here are the abstract methods present in the TaskCommand class.

Method	Return type and function
undo()	Returns a String which represents the feedback of the command being undone. This is the abstract method of the generic undo execution of each TaskCommand.
redo()	Returns a String which represents the feedback of the command being done again. This is the abstract method of the generic redo execution of each TaskCommand.

Command is an abstract class that uses the Command Pattern and holds the `execute()` method where the generic execution of `Command.execute()` can be used. Classes that extend from it will have their own implementation of the `execute()` method. `TaskCommand` is another abstract class which extends `Command` and is for commands that deal with manipulation of Task objects. `TaskCommand` has two abstract functions which are `Undo()` and `Redo()` which are also implemented separately by the child classes to revert the changes made by that command. The structure of the Command component allows the flexibility of adding new command types to Urgenda by simply extending one of the two abstract classes (`Command` and `TaskCommand`). The abstraction of the `Command` class allows new Commands to be added by just extending and implementing their unique `execute()` command.

Parser Component



The Parser component is accessible through the `Parser` class using the interface pattern. This component is invoked by the `Logic` component, and has the function of parsing a passed in user command string and return an appropriate `Command` Object. In order to do this, `Parser` will access different classes, each having its unique functions, as listed in the next section.

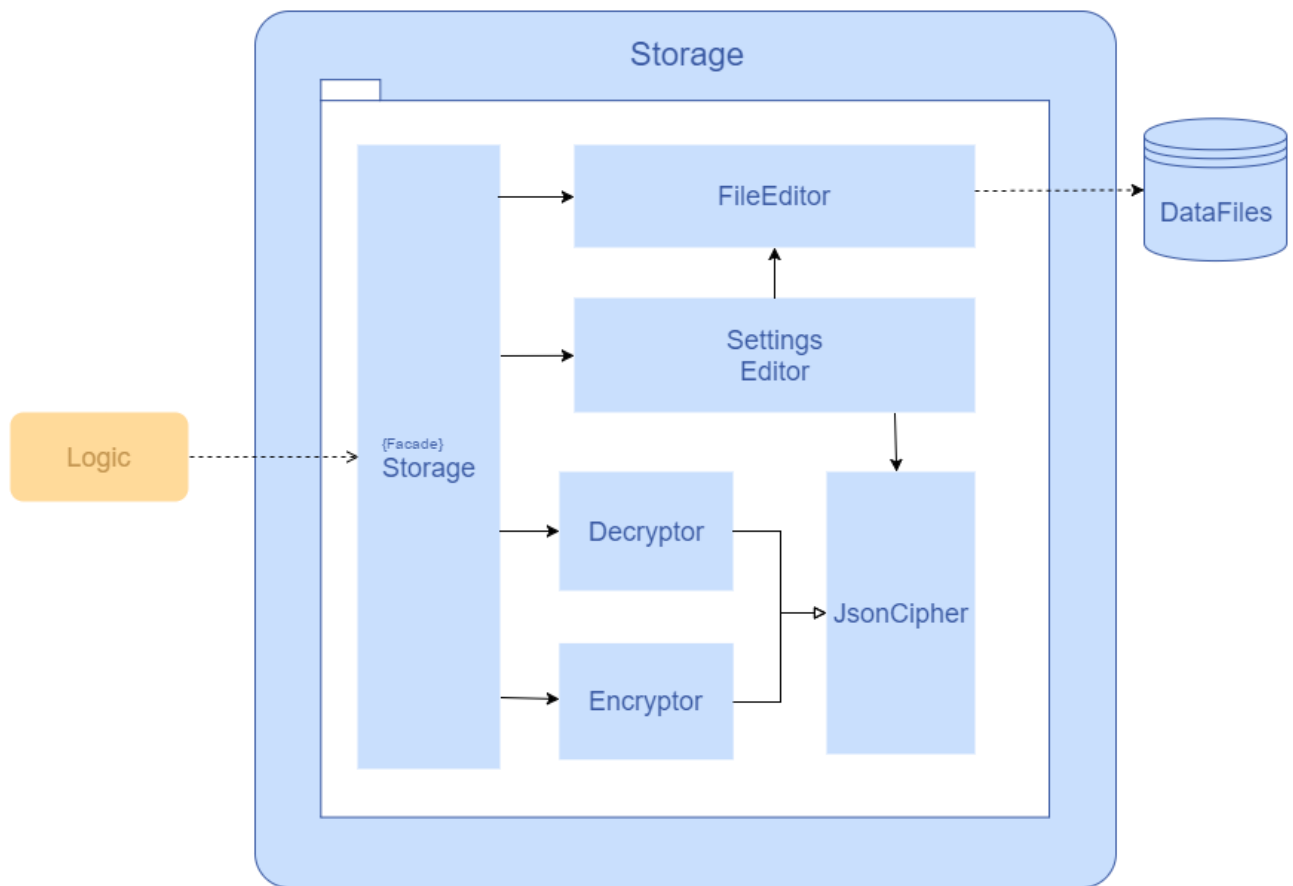
Parser Class

Class	Function
CommandTypeParser	In charge of parsing the passed in user command string to determine the type of expected returned Command object, as well as returning the string of arguments for further processing by other respective classes, such as DateTimeParser or TaskDetailsParser.
Collection of CommandParsers	Includes various classes that are in charge of generating and returned a specific type of Command object, such as AddTask or DeleteTask. The type and details of the returned Command objects depend on the private attributes String _argsString and int _index present in each of these classes, which will be set upon calling of its constructor. Each classes have its own method that invokes different functions in PublicFunctions to perform the correct parsing depending on the type of returned Command.
DateTimeParser	In charge of recognizing and parsing date and time values in the argument string. DateTimeParser relies on PrettyTimeParser, which is an external open source project that parses date and time values in natural language flexibly. The role of DateTimeParser is to make modifications to the pick-up patterns of PrettyTimeParser, as well as handling user keywords in the command. The parsing result is made directly on the variables stored in PublicVariables, and DateTimeParser returned the argument string already trimmed of date time expressions.
TaskDetailParser	In charge of recognizing and parsing other relevant details in the argument string, including task index, description, location and hashtags.
PublicFunctions	This class contains all the public functions shared between the command parser classes to perform their role.
PublicVariables	This class contains all the public variables accessible to the different command parser classes and needed to generate the appropriate type of returned Command object and correct details.

The API of parser is:

Method	Return type and function
ParseCommand(String commandString)	Returns a Command object with specific type and details, as parsed by parser.

Storage Component



The Storage component is accessible through the `Storage` class using the facade pattern, where it handles and directs file manipulation using the respective classes. Gestalt's Principle is used in this component to enhance the cohesiveness of each class and reduce the coupling, where only necessary dependencies are utilized. The functions of each class are grouped accordingly to the very meaning that each class name suggest.

Urgenda primarily has 3 files:

- `data.txt` stores all the tasks, either completed or uncompleted, in JSON format. Each line, or each string, represents one task. This file is able to be renamed or moved to other directories as per user's desire.
- `settings.txt` stores the settings of the user, such as the file name and file location, so that on start-up Urgenda can retrieve these settings, retrieve the datafile and set preferences for the user. This file cannot be moved or renamed.
- `help.txt` is where the documentation for the user is stored. If the user ever require any form of assistance in entering commands, he can bring up the help panel, which retrieves the text from this file. This file cannot be moved or renamed.

Storage Class

In every case where LogicData needs to access or edit the data in the file or the datafile itself, it goes through Storage class, which then dispatches the corresponding method using the respective classes within the Storage component.

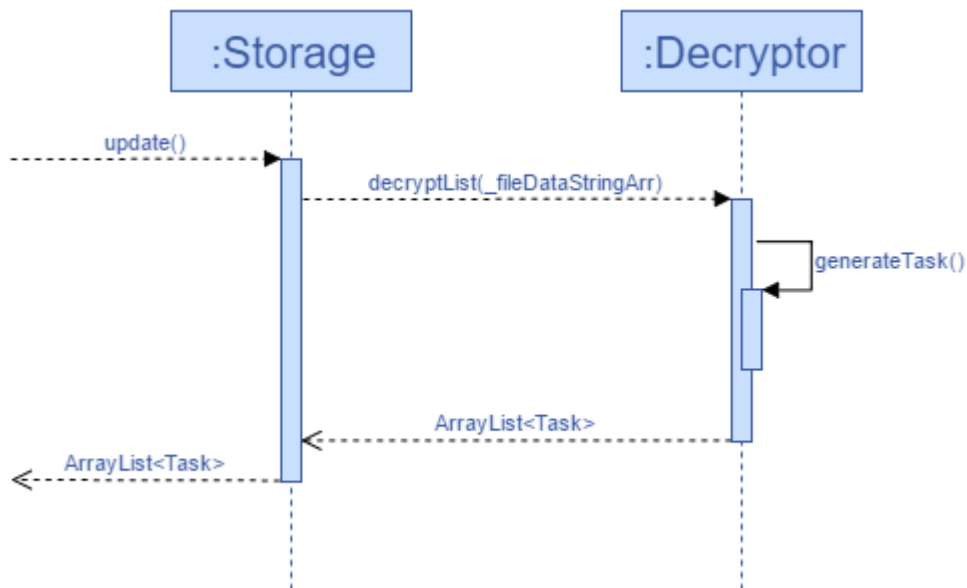
As mentioned above, apart from the Storage class which acts as the facade, all other classes have their own specific functions:

Class	Function
FileEditor	Contains all the file manipulation methods that is required in the Storage component - retrieving from file, writing to file, renaming, moving to other directories, clearing the file. Essentially, only this class can access and manipulate the actual file itself
JsonCipher	The primary ciphering tool. Storage uses the external library Gson that allows conversion of objects to a string. In this class, instead of converting directly from a Task to a String, a Task is converted to a LinkedHashMap<String, String>, then converted into a String. This allows for easier conversion back into a Task from a String. JsonCipher provides the tools required for converting from Task <=> LinkedHashMap<String, String> <=> String
Encryptor	A subclass of JsonCipher, the role of Encryptor is to encrypt all Task into a String using JsonCipher as a means of doing so.
Decryptor	A subclass of JsonCipher, the role of Decryptor is to decrypt all String into a Task using JsonCipher as a means of doing so.
SettingsEditor	This class handles all matters related to the settings of the user, in order not to mix it with the actual data file. File manipulation and encryption/decryption is done using the FileEditor class and through JsonCipher directly, since there is no need Task involved with the settings.

The main APIs of the Storage class include:

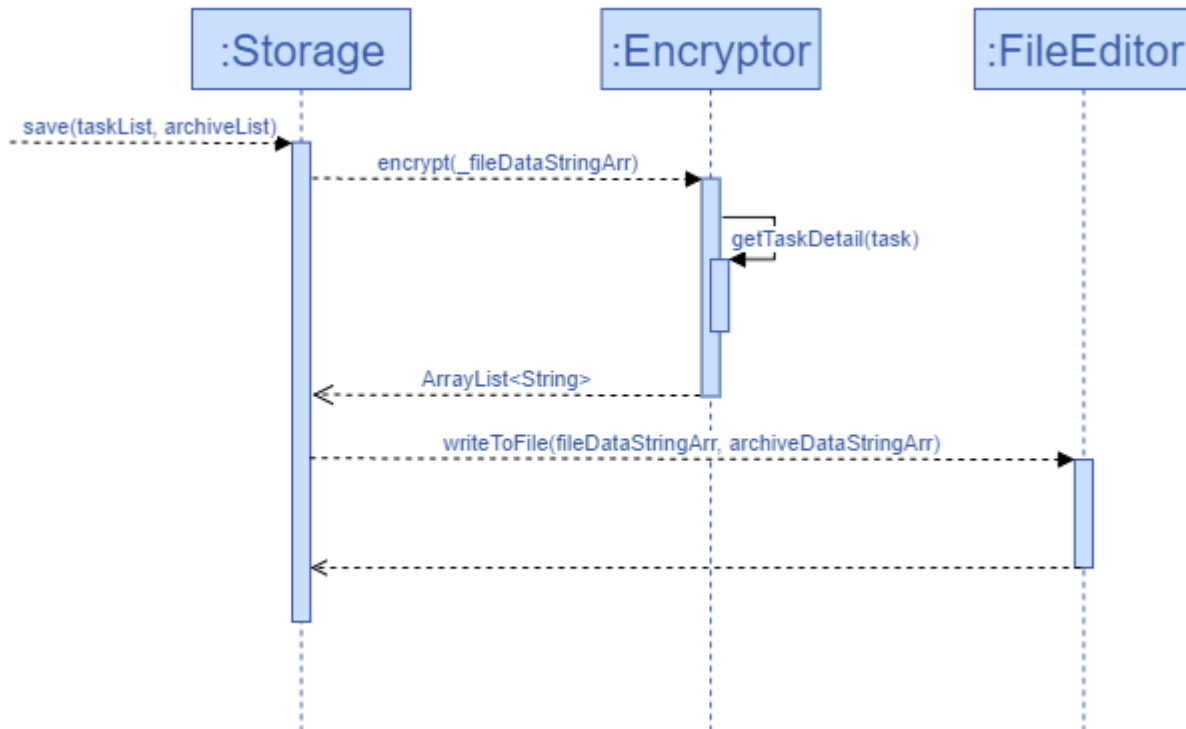
Method	Return type and function
updateArrayList()	Returns ArrayList<Task>. This method is used during startup to retrieve all tasks in the datafile and pack it into an ArrayList.
save(ArrayList<Task> tasks, ArrayList<Task> archives)	Void function. This method is used to store all tasks in the datafile, for easy retrieval, relocation to another computer.
changeFileSettings(String path, String name)	Void function. This method allows the datafile to be renamed and move to other directories/folders through Urgenda itself, with no need to enter File Explorer

updateArrayList()



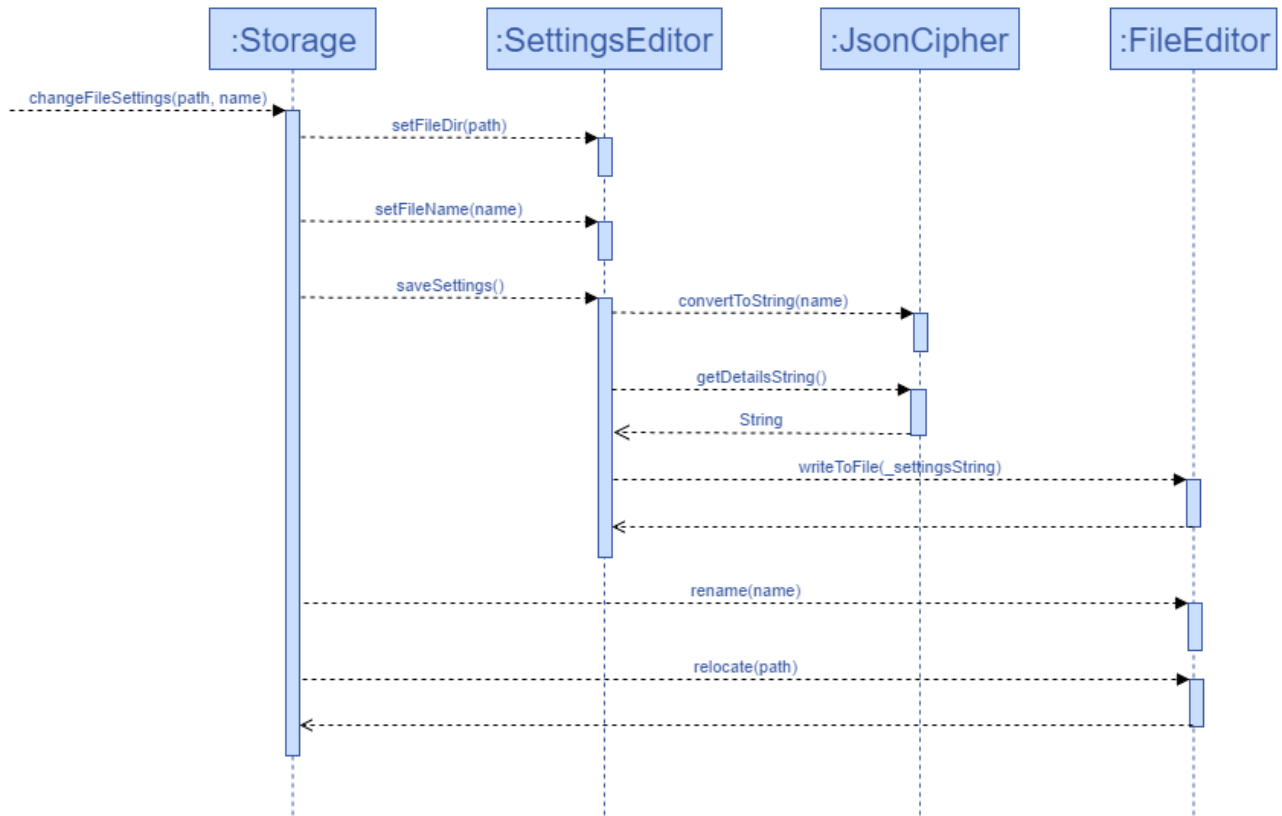
`updateArrayList()` is the generic method for `updateCurrentTaskList()` and `updateArchiveTaskList()`. With the `_fileDataStringArr` already retrieved and stored within `Storage` upon initialization, it simply has to be decrypted from JSON to actual `Tasks` objects.

save(ArrayList<Task> tasks, ArrayList<Task> archives)



`save(ArrayList<Task> tasks, ArrayList<Task> archives)` saves the current list of tasks into the specified file by writing onto it. This method can be split into two parts:

1. Encrypting the array list of Tasks into JSON format and converting it to an array list of String.
2. This part entails writing the array list of String into the specified file, where each String represents a single Task

changeFileSettings(String path, String name)

`changeFilePath(String path)` and `changeFileName(String name)` are similar methods to `changeFileSettings(String path, String name)`, whereby the latter changes both the name and the directory the datafile is saved in. `changeFileSettings(String path, String name)` has two parts to it:

- edit the preferred file name and file location in `settings.txt` `rename/relocate` the actual datafile
- with the preferred name to the preferred location

Appendix A: User stories. As a user, ...

[Likely]

ID	I can ... (i.e. Functionality)	so that I ... (i.e. Value)
add	add a task by specifying a task description only	can record tasks that I want to do
delete	delete a task	can forsake tasks that are no longer relevant
edit/update	edit a task	can rectify previous mistakes or add more info to a task
marking as done	mark done items	know that the task has been done.
exit	exit the program	can shut down the program after I am done with my tasks
archive	view tasks that have been previously completed	can have the history of my previous tasks
sync	access my to-do list from either my office or home computer	can check my to-do list anywhere even without internet accessibility
shortcuts	have keyboard shortcuts for various functions, i.e. quicklaunch, CRUD, etc,	can enjoy fast and easy access.
prioritise	mark certain tasks as a priority	know at a glance what is more important
sort	sort my tasks either by due dates, category or priorities	cater to different situations and needs
hashtags	hashtag my tasks	can search easily for tasks related to a single person/activity/event
rescheduling	re-schedule my tasks to a different timing	can have increased flexibility
free time	search for free times during a specified period where I am not occupied	can find empty time to insert tasks at said timings
quick add	quick add events	do not have to fill up the lengthy details with different fields
time clash	be alerted of possible clashes when entering a new event	can decide if the new event should still be added at said timing

history	view completed tasks together with their timestamps	can backtrack for references or recall similar tasks
ignore letter casings	CRUD commands regardless of the casings	the program will not regard my command as invalid due to casing
help	have a help section	can view all possible shortcuts and commands for easy reference
block	block off several timeslots for an unconfirmed event/meeting/activity, such that once the timing is confirmed, the other unused timeslots are released	can plan my time better and more efficiently
cycle	use the up button and down button to cycle through the displayed tasks.	can look through my entire task list
similar detection	detect commands with similar meanings	can use similar words to enter the same function
12/24 hr format	key in timings in both 12/24 hour format and it will get recognised	can naturally follow the format in emails without having to change before input
type suggestions	have a type suggestion feature to suggest what type commands are available to me given what I have already type into the dialog	more convenient for me to know the format of additional type-in commands
error checker	be warned of a typo that doesn't fit any command words or command tags	can edit to fit the requirements
embed	access the command bar using an embedded form in the taskbar	can enter commands without minimizing the window I am reading

[Unlikely]

ID	I can ... (i.e. Functionality)	so that I ... (i.e. Value)
pictures/location tag	insert pictures or contacts to tag events	have reference if it's related to the task
auto startup	enable auto-startup	can open the programme every time I turn on my computer without me having to click a button
folders	organize my tasks into folders/categories	the task list is neater and more organized
calendar	a calendar/ calendar view that details out what I have to be done on each day	can look through my tasks in time-based manner easily
settings	customize the background, interface, fonts.	can have a more aesthetically pleasant-looking program
google cal integration	sync my to-do with google calendar	can access my to-dos on different platforms
expected complete time	insert the time a task is expected to take to be completed	can plan my activities accordingly
siri/cortana	have someone to talk to, to inform me of my tasks, enable voice recognition	am less lonely during Valentine's day
share	share and print my to-dos	can have multiple copies of it and others may get a copy too
reminders	have either a single reminder or multiple reminders for tasks.	can get reminded of upcoming tasks

Appendix B: Non Functional Requirements

- Software should run on Windows 7 and later.
- Software should work on both 32-bit and 64-bit environments.
- Software should respond to any given command within 3 seconds.
- User interface should be easy for the user to find the tasks when shown.
- Data should be stored and backup readily available.
- Data should be easily copied from source file to another computer with the same software.
- All commands can be done without the usage of a mouse.
- Requires input from a keyboard.
- Data should be able to upload to an online storage and synced when user logs in from different devices where internet is available

Appendix C: Product survey

Product: Wunderlist **Documented by:** Ang Kang Soon

Strengths:

- Integration with Google Calendar
- Clean user interface
- Tasks with nearing deadlines are color coded
- Folders to categorize different to-dos
- Quick add for dates

Weaknesses:

- No keyboard shortcut to launch program window
- No suggestions
- No spelling check

Product: Todo.txt **Documented by:** Koh Kim Wee

Strengths:

- Search function works for phrases regardless of caps/small letters
- Search function also provides a - flag to exclude certain words from appearing
- Able to scroll for previous/next commands keyed

Weaknesses:

- Requires cygwin or other supporting software/knowledge to setup (not easy for people without much computing knowledge)
- Not user friendly as -h for help does not provide the full available and useful commands
- No undo function
- Numbering of tasks are not in order and have gaps after deletion (until archiving is done)
- Commands not intuitive / does not support similar words, such as del and delete
- No storage/viewing of done tasks

Product: Habitica **Documented by:** Tan Hui Kee Joanne

Strengths:

- Able to push urgent tasks to top (available with shortcut commands)
- Clear and distinct categorization between floating tasks, due date tasks and events
- Facilitates the addition of multiple tasks at one go (differentiated by one per line)
- Able to sync to different devices
- Out of the norm. Involves gaming elements to entice users.
- Completed To-Dos are automatically archived after 3 days. Can access them from Settings > Export.
- Able to add tags for easier categorizing.

Weaknesses:

- Complicated functions and naming of functions and commands are non-intuitive.
- User interface is too flamboyant, comes off as distracting and messy.
- Very few keyboard commands. Mostly require clicking with mouse (Not suitable for Jim/fast users).
- A search key that isn't very functional. Irrelevant filtering of things and tasks.
- Unable to view it in calendar mode for easy detection of free time.
- Does not facilitate multiple slots to be "blocked" when the exact timing of a task is uncertain, and release the blocked slots once the timing is finalized.

Product: Do! **Documented by:** Cheng Tze Jin

Strengths:

- Very minimalistic view, easy for first time user to know what to press, what to do.
- Adding is simplified, with or without details, all using the same button
- Colour-coding helps to categorize the different tasks and reminders
- Pinning tasks allows for important, top-priority tasks to be right at the top of the list immediately

Weaknesses:

- When trying to edit a task, too many unintuitive steps involved. “edit” button isn’t next to the task itself, but on the row for adding a new task. Small button that is difficult to press appears only after pressing the “edit” button
- Sorting by due dates, priority is hidden within settings, unable to be found easily.
- Displays date by 15/2, both of which are numbers and can be confusing when the two numbers are small enough to confuse the date.
- List of to-do too long and messy, no separation by “today”, “tomorrow”, “next week”.
- Two different areas to add a new task, redundant especially when there’s absolutely no difference.

Product: Google Calendar Quick Add Feature **Documented by:** Vo Hoang Khai

Strengths:

- Commands are intuitive, so new users would not have a hard time figuring out the correct command
- Flexibility in detecting user’s command, with many different acceptable format
- Allows user to add recursive events quickly
- Clean user interface

Weaknesses:

- Insufficient functionalities for the user (the user can’t use Quick add to view, edit, delete, search, etc)
- Does not have the flexibility to edit default settings, for example when creating recurring events, if no end date is specified the event is repeated 365 times, and this default cannot be edited.