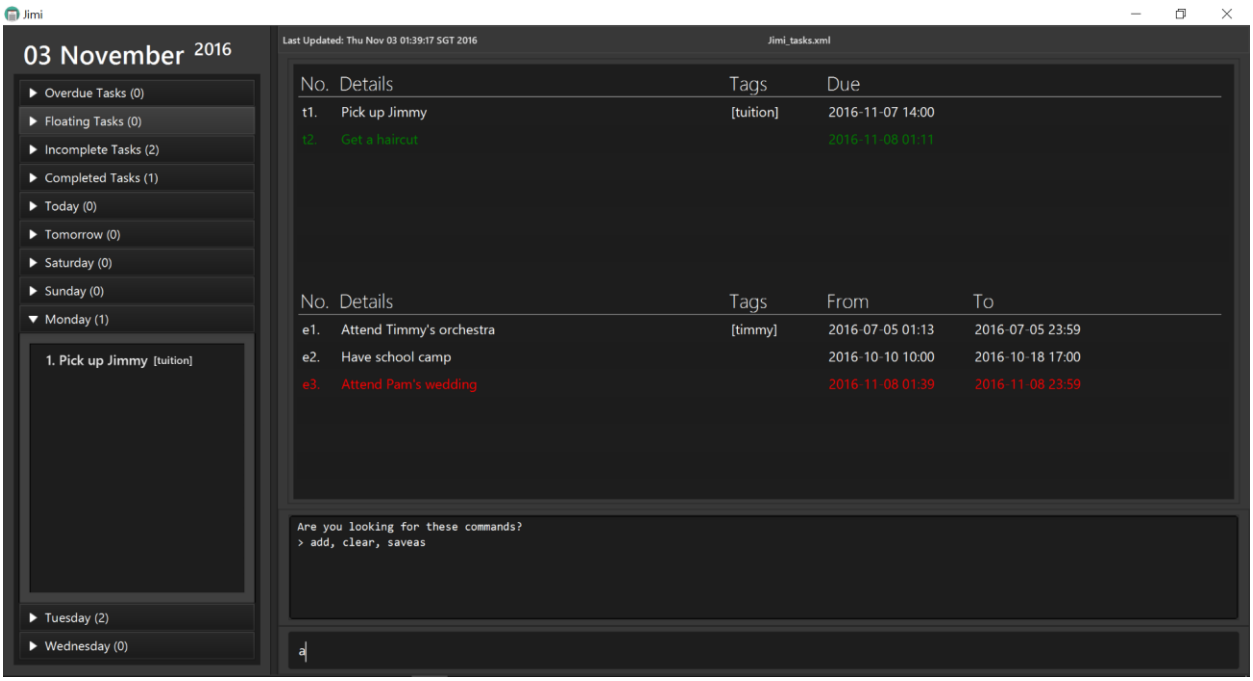
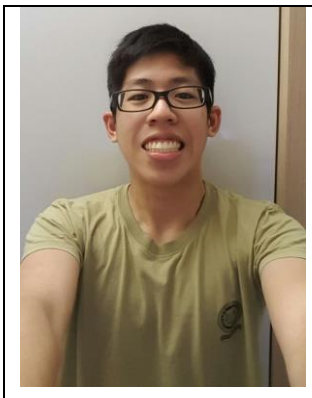
Jimi

The task manager for people like Jim.



Supervisor: Nirandika Wanigasekara

			
Clarence Chee	Zhang Yunxuan	Chong Ze Xuan	Chong Wei Yin

Acknowledgements

Some parts of this application were inspired by the sample project created by the SE-EDU initiative.

The Natty Date Parser library is utilized under the MIT license.

Some parts of this sample application were inspired by the excellent Java FX tutorial by *Marco Jakob*.

About Us

We are the team in charge of developing Jimi, a command-line based task manager that is designed for fast typers.

Our team of four, comprises computer engineers taking a stab at software engineering. We are familar with both OOP and Java - evidently from our code.

Clarence is the team leader and is in charge of overseeing the overall project coordination as well as managing the integration of different components of the software made by the team.

Ze Xuan is in charge of managing and assigning tasks to the team members as well as ensures that the project deliverables are completed on time.

Wei Yin takes charge of the overall documentation of the project, which includes the User Guide and the Developer Guide, among other things.

Lastly, **Yun Xuan** looks after the overall code quality of the project and ensures that coding standards are met as well as proper testing of the product.

Project Team

Nirandika Wanigasekara(https://github.com/nirandiw)

Role: Project Mentor

Clarence Chee @cheec

- Components in charge of : Logic
- Aspects/tools in charge of: Team Lead, Integration, Code quality control
- Features implemented:
    - Add tasks/events
    - Edit tasks/events
    - Set new save directory

    - Near match search
- Code written:[functional code][test code][docs]
- Other major contributions:
    - Set up JimiParser
    - Set up FilteredListManager
    - Morph Model and ModelManager
    - Update EditCommand parsing of updated commands
    - Update indexed commands to work with new indices
    - Refactoring/Morphing AddressBook -> Jimi
    - UI beautifying with Ze Xuan
    - Nag about code quality/style/admin issues
    - Real time command suggestions in console
    - Up/down arrow keys to cycle input history

Chong Ze Xuan @syltaris

- Components in charge of : UI
- Aspects/tools in charge of: Scheduling and Tracking, Deliverables and Deadlines, Git
- Features implemented:
    - Edit tasks/events
    - Show complete/incomplete tasks/events
    - Show tasks/events on a certain day
    - Complete a task/event
- Code written:[functional code][test code][docs]
- Other major contributions:

    - Morph Model and ModelManager
    - Set up the fixing of tasks
    - Implement AgendaListPanel and TaskListPanel for GUI
    - Create MainWindow layout
    - Update indexed commands to work with new indices
    - Set up FilteredListManager
    - Test Script for Manual Testing

Chong Wei Yin @ShadowLoner17

- Components in charge of : Storage
- Aspects/tools in charge of: Documentation
- Features implemented:
    - Delete tasks/events
    - Set new save directory
- Code written: [functional code][test code][docs]
- Other major contributions:
    - Set tasks with priority
    - Updating of documents
    - Implement Junit Testing for Deadlines/Events
    - Implement Gui Tests
    - Upload SampleData for Manual Testing

Zhang Yunxuan @yunxz

- Components in charge of : Model
- Aspects/tools in charge of: Testing, Code Quality Control
- Features implemented:
    - Find task/event using keyword
    - Undo action
    - Redo action
- Code written: [functional code][test code][docs]
- Other major contributions:
    - Set up Travis
    - Create DateTime class for Events
    - Implement Natty

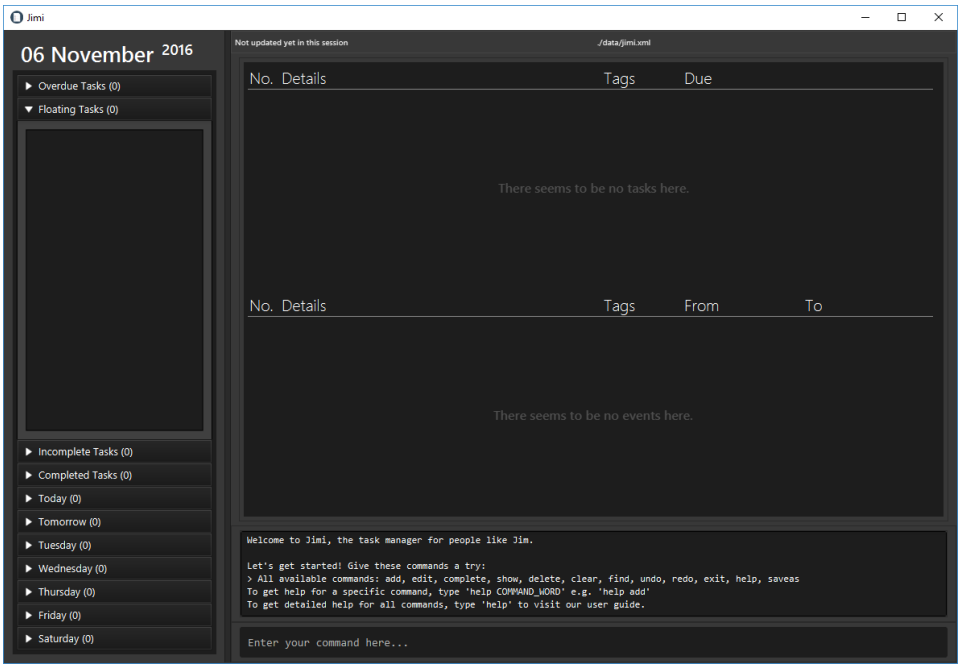
User Guide

At this point, we know you are just as excited as we are about Jimi. But before you start throwing your money at us (even though Jimi is entirely free), you should first learn **how to use Jimi properly**. What follows should guide you on how to setup, install and use Jimi easily.

Guide Map

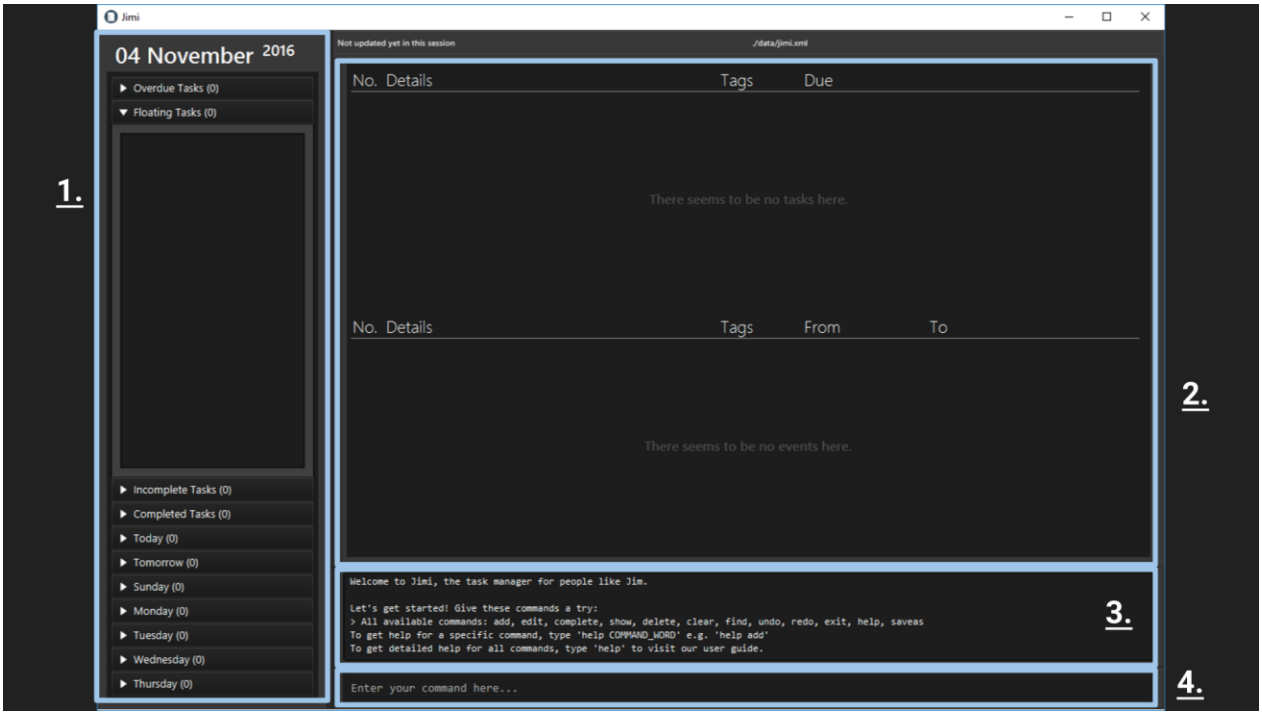
Quick Start

1. Ensure you have Java version `1.8.0_60` or later installed in your Computer.
   Having any Java 8 version is not enough.
   This app will not work with earlier versions of Java 8.
2. Download the latest `Jimi.<version>.jar` from the releases tab.
3. Copy the file to the folder you want to use as the home folder for Jimi.
4. Double-click the file to start the app. The window should appear in a few seconds.



5. Type any command in the command box below and press `Enter` to execute it.
   e.g. typing **`help`** and pressing `Enter` will open this user guide in another window.
6. Some example commands you can try:
   - **`add`** `"do laundry"` **`due`** `tomorrow` : adds a task named `do laundry` due `tomorrow` to Jimi.
   - **`delete`** `t1` : deletes the 1st task shown in the current task list
   - **`exit`** : exits the app
7. Refer to the Features section below for details of each command.

Explaining the UI



Jimi's UI is simple to use and compact, everything you need is right at your fingertips, literally.
The UI comprises four main parts:

1. A summary panel, containing important information at a glance. To expand the drop down lists, you can either click them, or use the `show` command.
2. Task/event tables, here is where you would find all your tasks/events. The top table is for tasks, and the bottom for events. Changes in the tasks/events will be updated in real time here too.
3. A console box, essentially the voice of Jimi. It displays error messages, feedback from commands, tips and so on. This is basically your babysitter through your first usage, so pay attention to what it has to offer.
4. The command line, is where you would tell Jimi what to do. Enter your single line commands with the appropriate format and let Jimi do the dirty work.

Command Summary

Command	Format
Help	`help [COMMAND_WORD]`
Add	`add "TASK_DETAILS" [t/TAG] [p/PRIORITY]`
	`add "TASK_DETAILS" due DATE_TIME [t/TAG] [p/PRIORITY]`
	`add "EVENT_DETAILS" on|from START_DATE_TIME [to END_DATE_TIME] [t/TAG] [p/PRIORITY]`
Find	`find "KEYWORD [MORE_KEYWORDS]..."`
	`find ["KEYWORD [MORE_KEYWORDS]..."] on|from DATE_TIME [to DATE_TIME]`
Complete	`complete INDEX`
Delete	`delete INDEX`
Edit	`edit INDEX NEW_DETAILS`
Show	`show SECTION`
Undo	`undo`
Redo	`redo`
SaveAs	`saveas NEW_DIRECTORY`
Clear	`clear`
Exit	`exit`

General Information About Command Format

- Commands have to follow a certain format as shown in the table above.
- Replace words in `UPPER_CASE` with your input.
- Items in `[]` are optional.
- Items followed by ellipses, `...`, means you can have multiple instances of that item.
- Items seperated by `|` simply means any of the items will work. E.g. `on|from`, typing `on` instead of `from` and vice versa are fine.
- The order of your input text is fixed. For instance, `add DATE_TIME due "TASK_DETAILS"` is invalid.
- Some commands allow shorter command words for advanced users. Some commands, due to their critical nature e.g. `exit` or `clear`, you are required to type the full command word.
- Command words are also all case-insensitive i.e. `add` works as well as `ADD`.

Below are accepted shortcuts of all commands.

Command	Default Command Word	Shortcuts
Help	`help`	`h`, `he`, `hel`
Add	`add`	`a`, `ad`
Find	`find`	`f`, `fi`, `fin`
Complete	`complete`	`c`, `co`, `com`, ... , `complet`
Delete	`delete`	`d`, `de`, `del`, ... , `delet`
Edit	`edit`	`e`, `ed`, `edi`
Show	`show`	`s`, `sh`, `sho`
Undo	`undo`	`u`, `un`, `und`
Redo	`redo`	`r`, `re`, `red`
SaveAs	`saveas`	None

Command	Default Command Word	Shortcuts
Clear	`clear`	None
Exit	`exit`	None

Input of Task/Event Name

- You have to always surround task/event names with double quotation marks, `"`.
- Other than that, you can type anything within those double quotation marks.

Input of Tags and Priorities

- For tags, you can only type single words made up of numbers and letters. E.g. `t/IMPT`.
- For priorities, you can only type the following: `low`, `med`, `high`, `none`. E.g. `p/low`.
- Priorities are also case-insensitive i.e. `p/low` works just as well as `p/LOW` or `p/lOw`.
- A priority of `low` highlights your item in green; `med`, in yellow; and `high` in red.
- If you fail to specify a priority when adding an item, Jimi defaults it to `none`.
- For both, you cannot leave out the prefixes i.e. `t/` and `p/`.

Input of Date and Time in Commands

- The input of date and time is flexible.
- eg:
    - Tomorrow 2pm
    - Next Monday
    - 7/11/2016
    - 1 day from now
    - tmr
    - tdy
    - next week
- You can either input date, time or both.
    - If no time is given, the current time will be used instead.
    - If no date is given, the current date will be used instead.

- However, you cannot input none of them.
- The start-date & time of the events cannot be earlier prior to the end-date & time.

- Given the nature of natural language processing, Jimi can't gurantee that it will interpret your specified date/time with 100% certainty. If it ever happens that Jimi misinterprets your dates/times, you can either `undo` or `edit` to make changes.
- A failsafe option, however, is to simply type proper calender dates i.e. `27 oct` or `1 feb 2pm`.

Input of Index

- In order to differentiate the indexes of the tasks and events in the command inputs:
    - Indices are case-insensitive.
    - The index of tasks should be preceded by the letter 't'.
    - The index of events should be preceded by the letter 'e'.
    - Eg:
        - complete **t1**
        - delete **e3**
    - You can't go wrong if you follow the index that's written under the `No.` column of the tables.

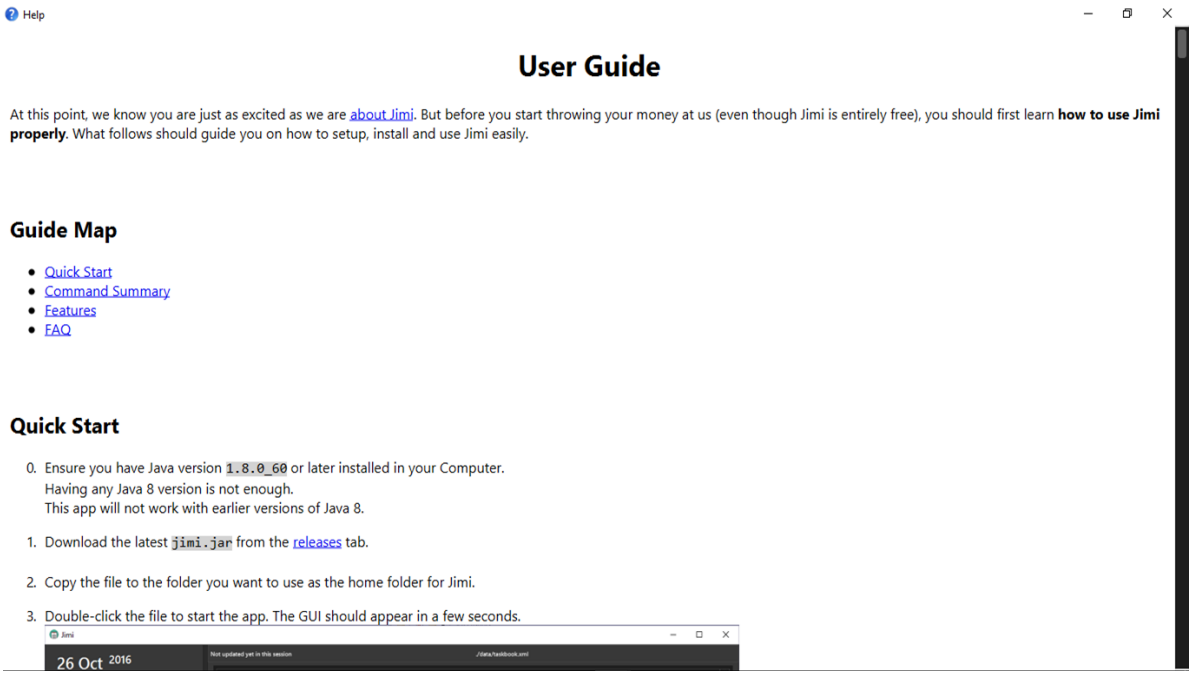
Features

Viewing help : `help`

Teaches you how to use Jimi.

Format: `help [COMMAND_WORD]`

Typing `help` will open this user guide in an offline version.



Typing `help COMMAND_WORD` e.g. `help add`, will show help for that specific command in the console box.

You can also type `help a`, equivalent of `help add` as all shortcuts mentioned above applies too.

Adding a task: `add`

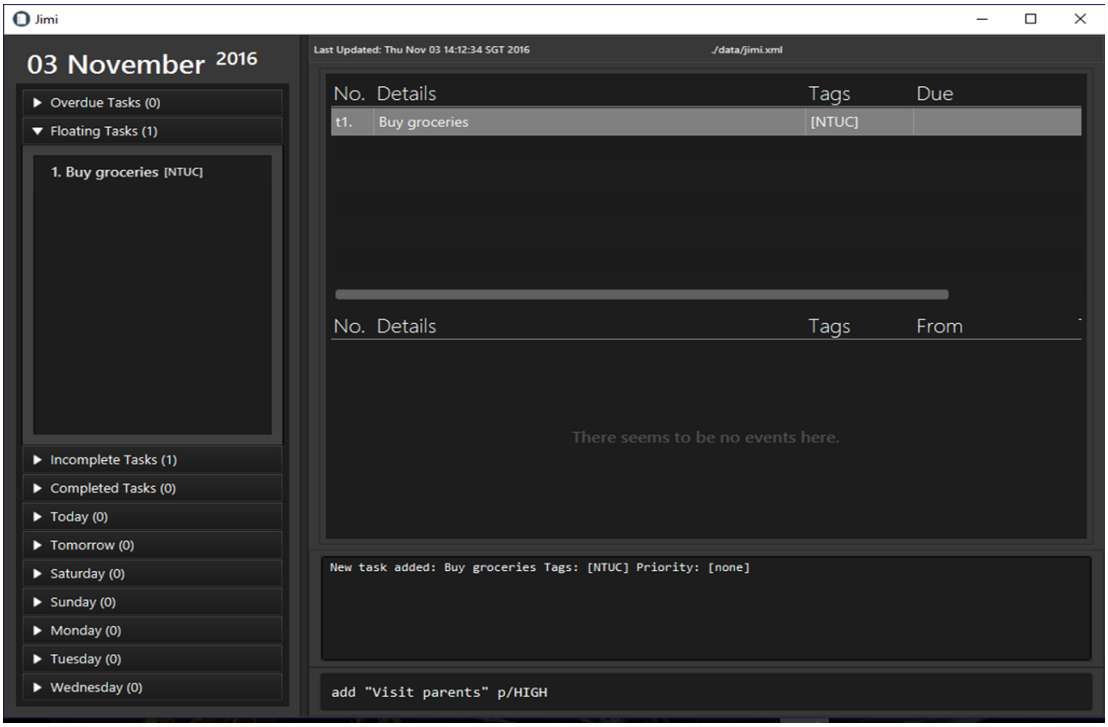
Adding a floating task to Jimi.

Format: `add "TASK_DETAILS" [t/TAG] [p/PRIORITY]`

- Floating tasks are tasks without any deadlines.

Examples:

- `add "Buy groceries" t/NTUC`
- `add "Visit parents" p/HIGH`

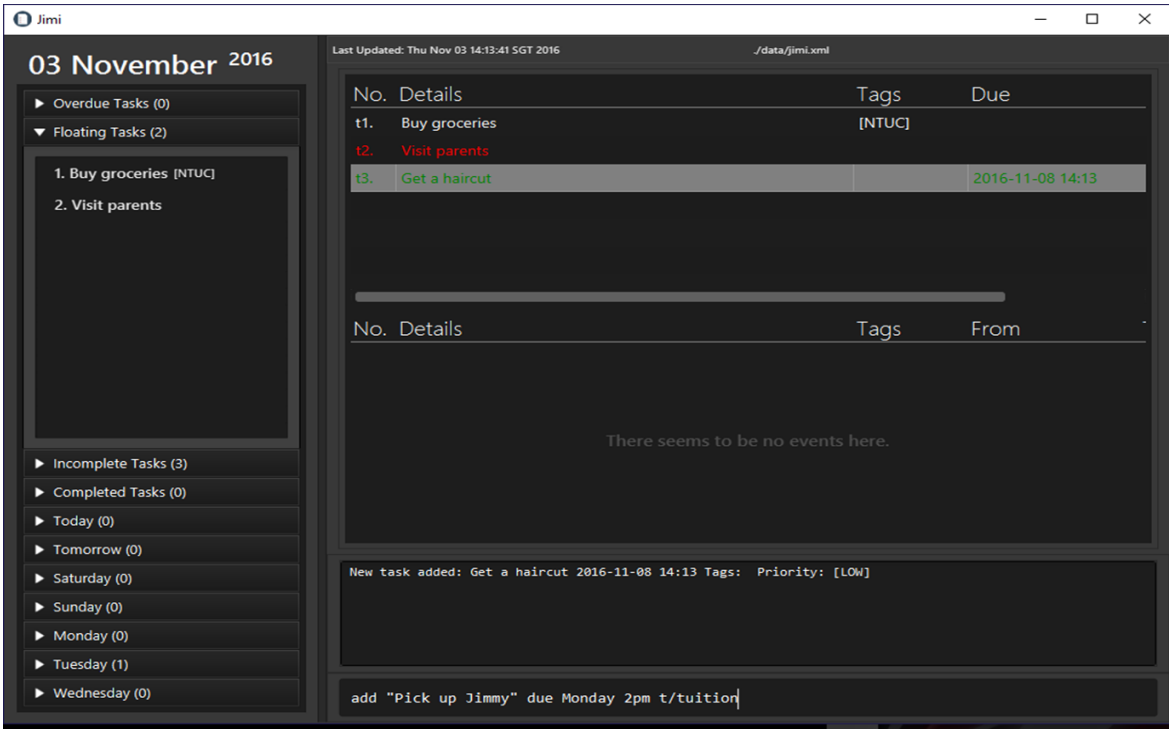


Adds a task with a deadline to Jimi.

Format: `add "TASK_DETAILS" due DATE_TIME [t/TAG] [p/PRIORITY]`

Examples:

- `add "Get a haircut" due Tuesday p/LOW`
- `add "Pick up Jimmy" due Monday 2pm t/tuition`



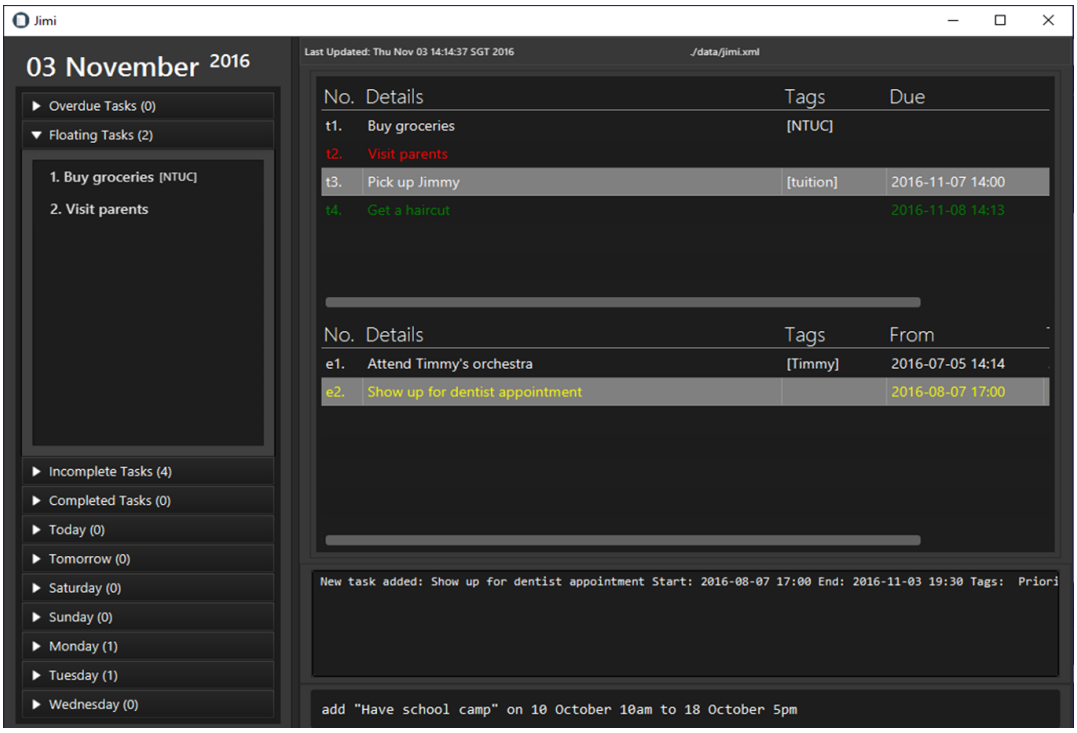
Adds an event to Jimi.

Format: `add "EVENT_DETAILS" on START_DATE_TIME [to END_DATE_TIME] [t/TAG] [p/PRIORITY]`

- If the event is more than a day long, you may include the end date_time.
- You may define the end time of the event if you wish. If you do not, however, Jimi will assume your event lasts till midnight of the start day.

Examples:

- `add "Attend Timmy's orchestra" on 5th July t/Timmy`
- `add "Show up for dentist appointment" on 8-7-2016 5:00pm to 7:30pm p/MED`
- `add "Have school camp" on 10 October 10am to 18 October 5pm`



Finding all tasks relevant to keywords you input: `find`

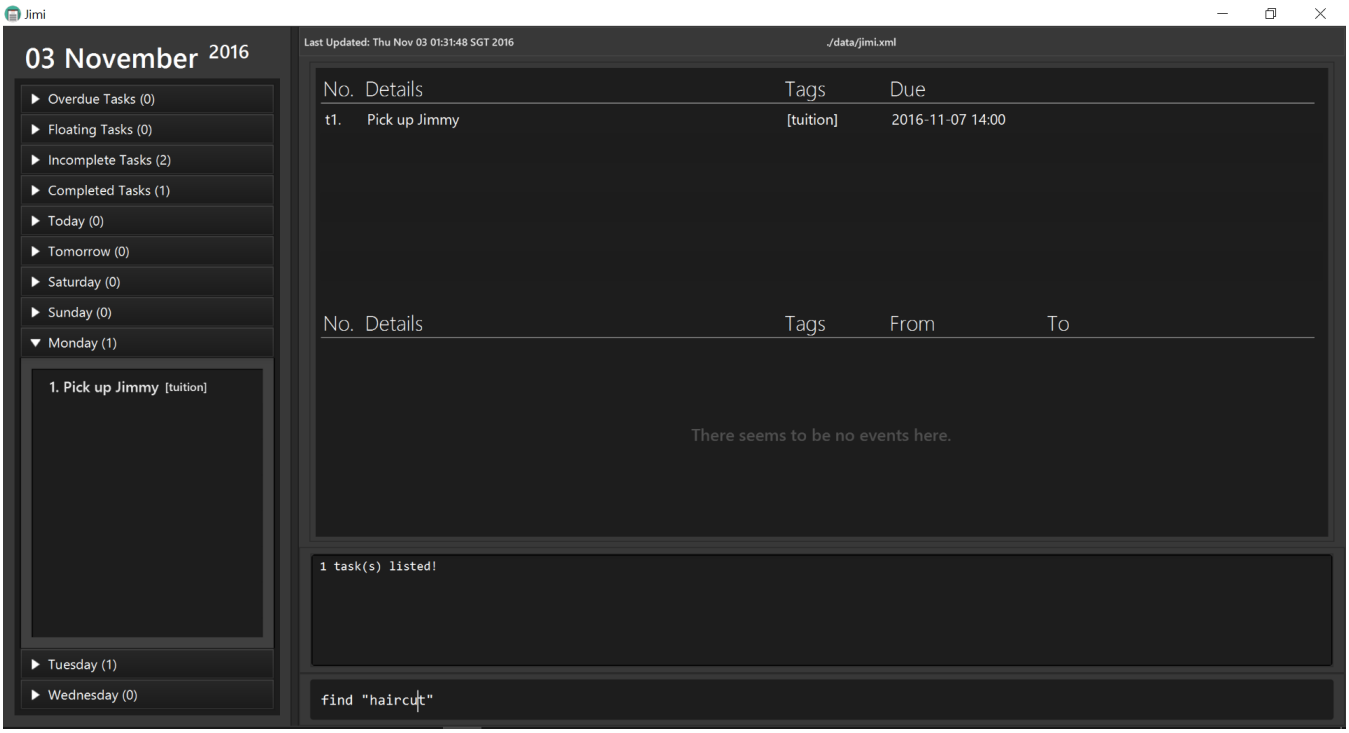
Finds and lists all tasks in Jimi whose name contains any of the argument keywords.

Format: `find "KEYWORD [MORE_KEYWORDS]"`

- The keywords must be specified in quotes.
- The order of the keywords you type in does not matter. e.g. `Essay writing` will match `Writing essay`
- Task details, tags and priorities can be search. e.g. `find "high"` will cover high priority tasks too.
- Searching takes into account typos too, to a certain extent. e.g. `find "apolet"` will match `apple`.
- Tasks with details/tags/priorities matching at least one keyword will be returned. e.g. `Writing` will match `Writing essay`

Examples:

- `find "Jimmy"`
- `find "buy attend do get"`



Finding all tasks according to the dates you specify: `find`

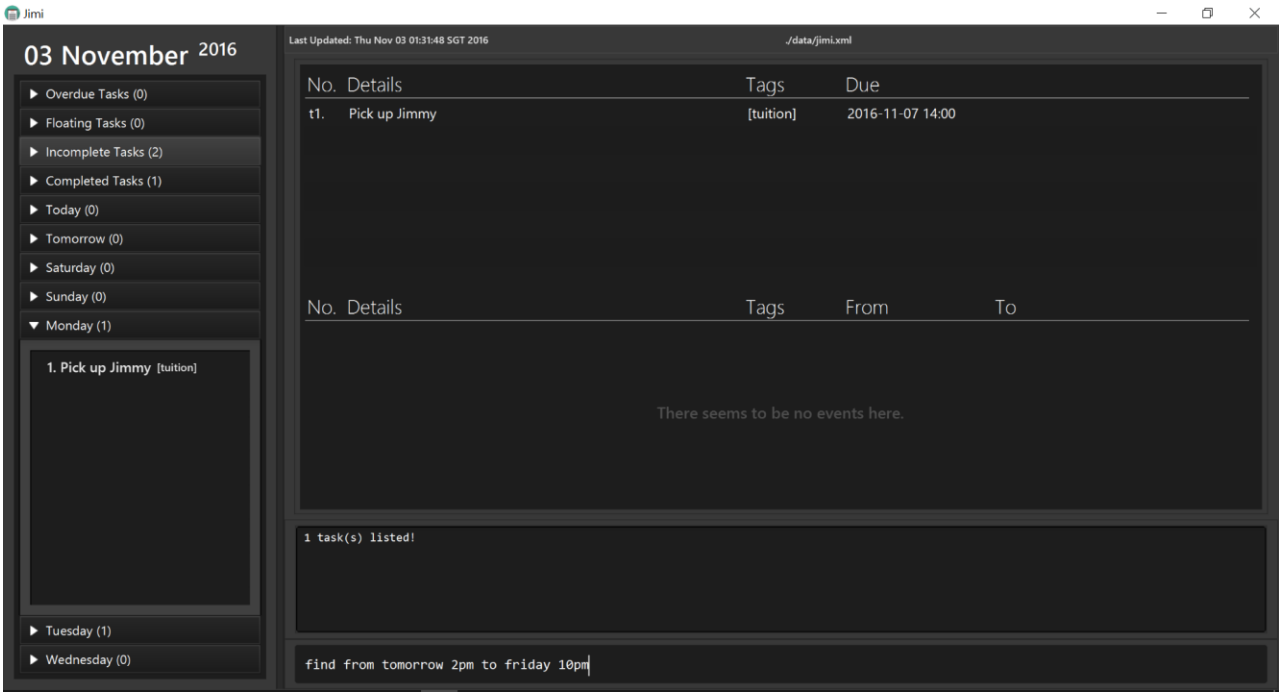
Finds and lists all tasks and events in Jimi whose dates matches the requirements specified.

Format: `find ["KEYWORD [MORE_KEYWORDS]..."] on|from DATE_TIME [to DATE_TIME]`

- You can also input just a single date to search for tasks and events relevant to that day.
- You can search the tasks and events by dates along with keywords as well.
- Simply append the dates to the keywords.

Examples:

- `find from tuesday to wednesday`
- `find "attend" from tomorrow to next month`



Marking a task as complete: `complete`

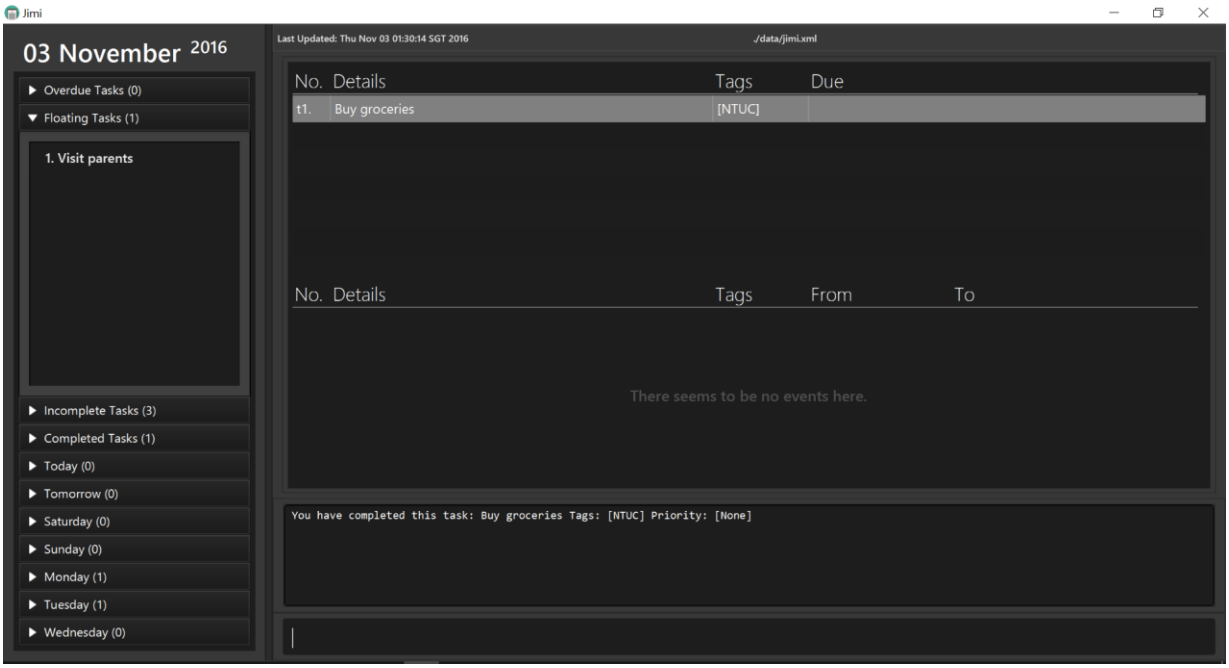
Marks an existing task as complete.

Format: `complete TASK_INDEX`

- Jimi will mark the task as completed at the specified `TASK_INDEX`.
- Jimi will then move the completed task to a completed task list.
- If you want to revert the task back as incomplete, use the undo command.

Example:

- `complete t1`



Deleting a task/event: `delete`

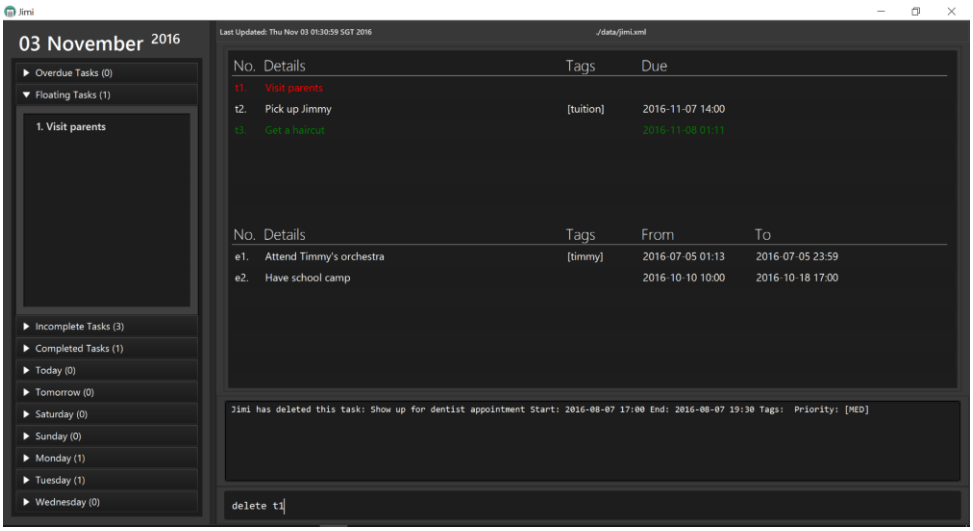
Deletes the specified task/ event from Jimi.

Format: `delete INDEX`

- Jimi will delete the task specified by `INDEX`.
- If you need to recover your deleted task/event, use the undo command.
- You can also delete a range of tasks/events by specifying the `to` keyword as well.

Examples:

- `delete e2` Deletes the 2nd event in Jimi.
- `delete t1` Deletes the 1st task in the Jimi.
- `delete t1 to t5` Deletes the 1st task to 5th task in Jimi.



Editing a detail: `edit`

Edits the specified detail of any task or event.

Format: `edit INDEX NEW_DETAILS`

- Jimi edits the task/event specified by `INDEX`, `NEW_DETAILS` are simply the edits you want to make.
- You can edit everything from the items name to its priority. You can leave out fields that you do not wish to edit too.
- Although all fields are optional, they can't all be empty!
- The format of `NEW_DETAILS` that Jimi recognizes when editting:
    - ["NEW_TASK_DETAILS"] [due NEW_DATETIME] | [[on NEW_START_DATETIME][to NEW_END_DATETIME]] [t/NEW_TAG] [p/NEW_PRIORITY]
- Here are some examples:

If you type	**Jimi will**
edit t1 "finish this user guide"	edit just the name of task t1.
edit t4 t/bobz	edit just the tag of task t4.
edit e2 "go to concert" to monday p/LOW	edit name, end date and priority of event e2.
edit e1 on sunday	edit the start date of event e1.
edit e6 to tmr	edit the end date of event e6.
edit t2 due 8pm	edit the deadline of task t2.
edit e9 on tmr to next monday	edit the start date and end date of event e9.

When editting an event, you can either edit just the start date alone or the end date alone or both. But if you wish to convert to an event from a task, `on|from NEW_START_DATETIME` is no longer optional, as will be shown below.

- Using edit, you may also convert between item types. That is, you may freely convert between floating tasks (dateless tasks), events, and deadline tasks.
- Below is the format of `NEW_DETAILS` that Jimi recognizes when converting:

Converting to	**What to type for `NEW_DETAILS`**	**Examples**
Dateless Task	dateless	`edit e1 dateless`
Tagless item	tagless	`edit t1 tagless`
No priority item	p/none	`edit t4 p/none`
Deadline Task	["NEW_TASK_DETAILS"] due NEW_DATETIME [t/NEW_TAG] [p/NEW_PRIORITY]	`edit e3 due tomorrow p/HIGH`
Event	["NEW_TASK_DETAILS"] on|from NEW_START_DATETIME [to NEW_END_DATETIME] [t/NEW_TAG] [p/NEW_PRIORITY]	`edit t1 "skip CS2103 lecture" on 29 oct t/IMPT`

- If you ever make a mistake, don't be afraid to use the undo command.

Showing section: `show`

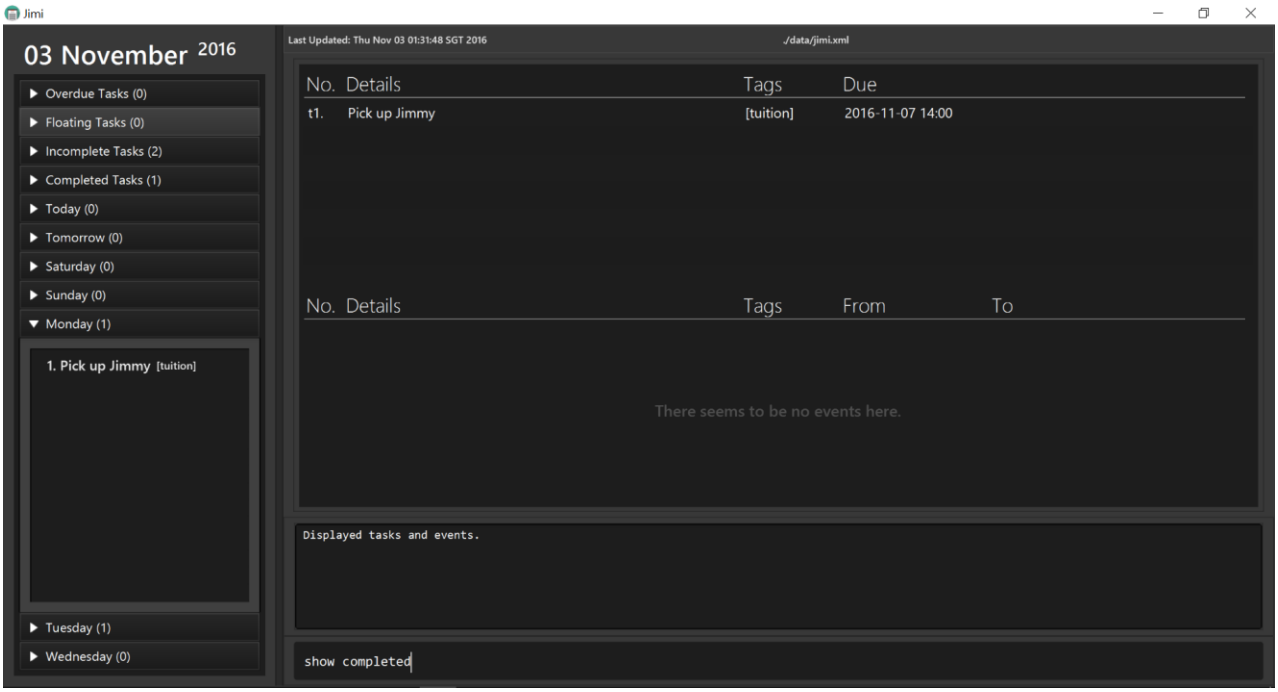
Expands and lists sections from the left summary panel, or displays all tasks and events.

Format: `show SECTION`

- `SECTION` is case-insensitive.
- For the sections with two words, you can type just the first word of the two.
- To display all tasks and events, please input `show all` as the command.

Examples:

- `show monday`
- `show completed`
- `show all`



Undoing previous action: `undo`

Undoes the previous action done in Jimi.

Format: `undo`

Only actions that make changes to the data of Jimi are undo-able.

i.e. `add`, `delete`, `edit`, `clear`, `complete` You can only undo actions done in the current session, if you exit from Jimi you cannot undo any actions done in the previous session when a new session is started.

Redoing previously undone action: `redo`

Redoes the previously undone action done in the task manager.

Format: `redo`

Only actions that make changes to the data of Jimi are redo-able.

i.e. `add`, `delete`, `edit`, `clear`, `complete`

Setting save directory : saveas

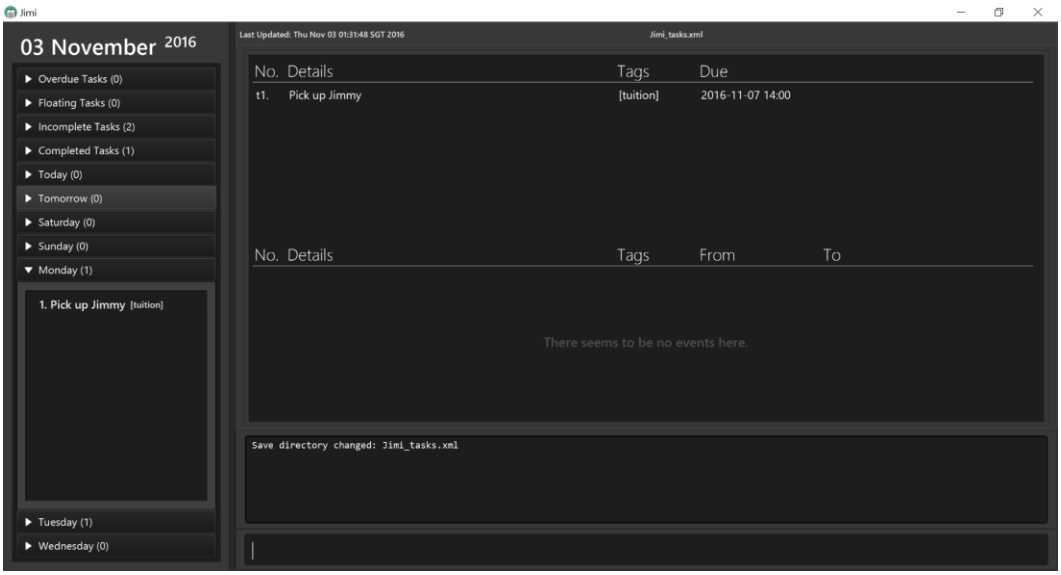
Saves a copy of the existing storage file to a new directiory, also all future saves save to this new directory.

Format: `saveas NEW_DIRECTORY`

- `NEW_DIRECTORY` should be in the format: `FILE_PATH/FILE_NAME.xml`
- **WARNING**: this command overwrites the specified .xml file in the new save directory. It does **NOT** load files from the new directory, if you wish to load files from a new directory, you would have to manually transfer the content of the .xml files over.
- If you want to reset the save directory back to default of `<home_folder_of_installation>/data/jimi.xml`, type `saveas reset`

Example:

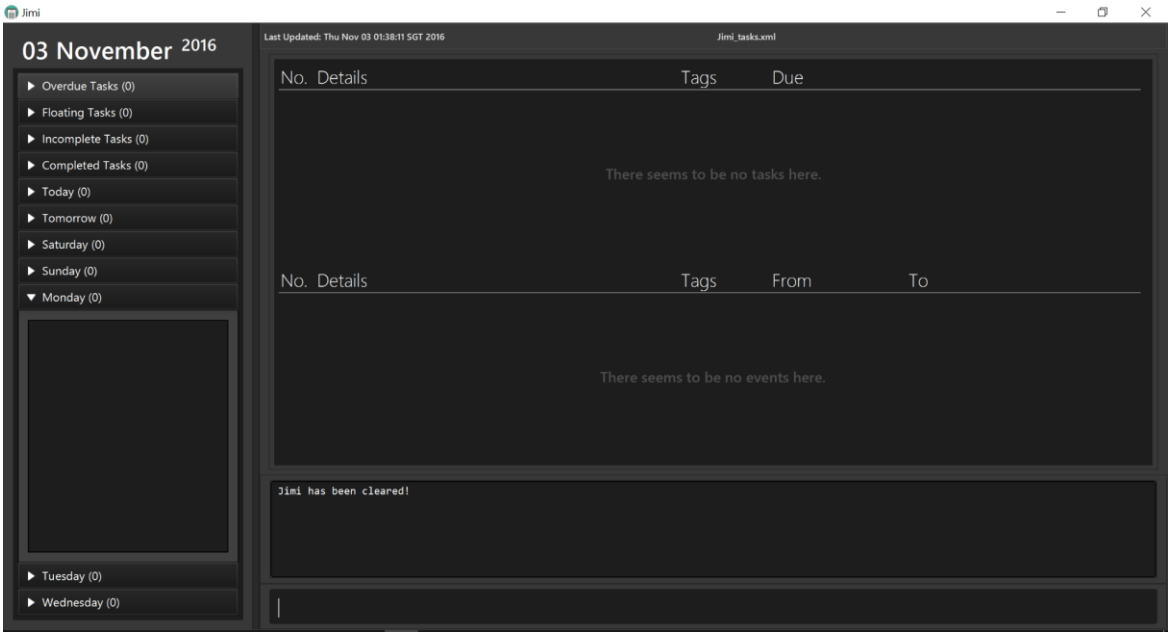
- `saveas Jimi_tasks.xml`



Clearing all entries : `clear`

Clears all entries of tasks and events from Jimi.

Format: `clear`



If you want to undo your clear, use the undo command.

Exiting the program : `exit`

Exits the program.

Format: `exit`

Before exiting the program, ensure you have no unwanted actions that need to be reverted.

Saving the data

The tasks and events in Jimi are saved in your hard disk automatically if you made any changes or added any new items.

You do not need to worry about saving the data manually.

FAQ

How do I transfer my data to another Computer?

Simply copy the data file specified by the save location to the other computer. Install the app in the other computer and then specify the location of the data file you want to load using the `saveas` command, or just overwrite the default data file with the old file.

Is there a way to be notified of upcoming tasks or events that are due soon?

Jimi will display all overdue tasks at the top Agenda box, so you will always be notified of the most important details first. To display upcoming tasks or events, do use the find command to list them.

What happens if I enter a wrong command?

Jimi will display an error message that tells you that the command you entered is incorrect.

What happens if I enter an invalid format?

Jimi will display an error message that tells you that the format for the command you entered is incorrect and will also display the correct command format required for that command.

Can I still use the mouse to use Jimi's functionalities?

As Jimi is specifically catered to use the command-line interface, support for mouse-input is limited. You can display certain sections of the summary panel by clicking on the headers with the mouse although that is pretty much what Jimi can allow you to do with the mouse.

How do I see all of my completed tasks?

You can do so by using the `show completed` command which will `show` all of your completed tasks in the main window.

Can I use this with Google Cloud or my favourite cloud service?

As of now, Jimi does not support any online functionality. However, there are plans for the development of online cloud services integration so users can easilly access their data from multiple devices with ease.

How can I delete multiple tasks/events at once?

You can do so by using the `delete` command follow by the range of indices of tasks or events you want to remove.

Jimi misinterpreted my dates/times! Why is it so dumb?

We apologise for the mishap. Please submit an issue to us at our issue tracker if you would like Jimi to accept particular dates/times.

Regarding your question, given the nature of natural language processing, it is physically impossible for Jimi to interpret your dates/times with a 100% certainty. Much like how Apple's Siri and Google's Google Assistant fail to recognize accents from time to time, Jimi is similar in that regard. If it ever happens that Jimi misinterprets your dates/times, you can either `undo` or `edit` to make changes.

A failsafe option, however, is to simply type proper calender dates i.e. `27 oct` or `1 feb 2pm`, Jimi will never get that wrong!

Why is the find command so slow sometimes?

We are experimenting with a near-match search agorithm to help better your experience. Please bear with us; improving the run-time of the `find` command is an important focus of ours and we look to fix this issue in the near future.

Developer Guide

Jimi is a simple task manager specifically catered for people like Jim. It is a Java desktop application that has both a Text UI and a GUI. Jimi handles most, if not all, input via the command line interface (CLI).

This guide describes the design and implementation of Jimi. Here, we share the inner workings of how Jimi works and how you can further contribute to its development. We have organised this guide in a top-down fashion so that you first understand the big picture before moving on to the finer details.

Guide Map

Setting up

Prerequisites

1. **JDK 1.8.0_60** or later

   Having any Java 8 version is not enough.
   This app will not work with earlier versions of Java 8.

2. **Eclipse** IDE
3. **e(fx)clipse** plugin for Eclipse (Follow the steps from starting from step 2 in this page).
4. **Buildship Gradle Integration** plugin from the Eclipse Marketplace.

Importing the project into Eclipse

1. Fork this repo, and clone the fork to your computer.
2. Open Eclipse. (Note: Ensure you have installed the **e(fx)clipse** and **buildship** plugins as given in the prerequisites above.)
3. Click `File` > `Import`.
4. Click `Gradle` > `Gradle Project` > `Next` > `Next`.
5. Click `Browse`, then locate the project's directory.
6. Click `Finish`.
    - If you are asked whether to 'keep' or 'overwrite' config files, choose to 'keep'.
    - Depending on your connection speed and server load, it can even take up to 30 minutes for the set up to finish. (This is because Gradle downloads library files from servers during the project set up process.)
    - If Eclipse auto-changed any settings files during the import process, you can discard those changes.

Troubleshooting project setup

Problem: Eclipse reports compile errors after new commits are pulled from Git

* Reason: Eclipse fails to recognize new files that appeared due to the Git pull.

- Solution: Refresh the project in Eclipse:
  Right click on the project (in Eclipse package explorer), choose Gradle -> Refresh Gradle Project.

Problem: Eclipse reports some required libraries missing

* Reason: Required libraries may not have been downloaded during the project import.

- Solution: Run tests using Gradle once (to refresh the libraries).

Design

Architecture

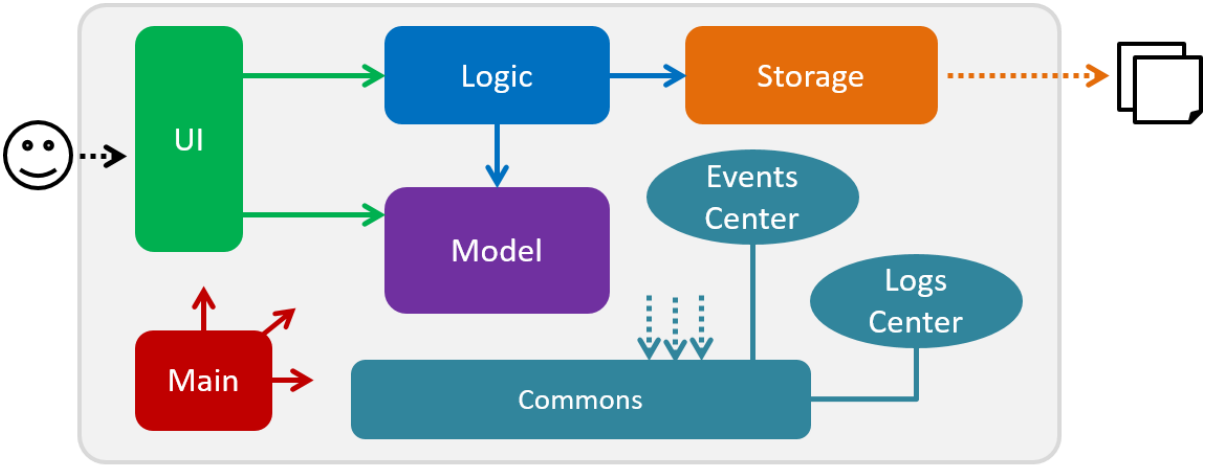


Fig 1.1 Architecture Diagram

The ***Architecture Diagram*** shown in fig 1.1 explains the high-level design of the App. Given below is a quick overview of each component.

Main has only one class called MainApp. It is responsible for:

- Initializing the components in the correct sequence, and connect them up with each other when app launches.
- Shutting down the components and invoke cleanup method where necessary when app shuts down.

Commons represents a collection of classes used by multiple other components. Two of those classes play important roles at the architecture level.
* EventsCenter : This class (written using Google's Event Bus library) is used by components to communicate with other components using events (i.e. a form of *Event Driven* design)
* LogsCenter : Used by many classes to write log messages to the App's log file.

The rest of the App consists four components.

- **UI** : Displays interactions with the user.
- **Logic** : Executes the commands.
- **Model** : Holds the data of the App in-memory.
- **Storage** : Reads data from, and writes data to, the hard disk.

Each of the four components

- Defines its *API (Application program interface)* in an interface with the same name as the Component.
- Exposes its functionality using a {Component Name}Manager class.

For example, the Logic component (see the class diagram in fig 2.1) defines it's API in the Logic.java interface and exposes its functionality using the LogicManager.java class.

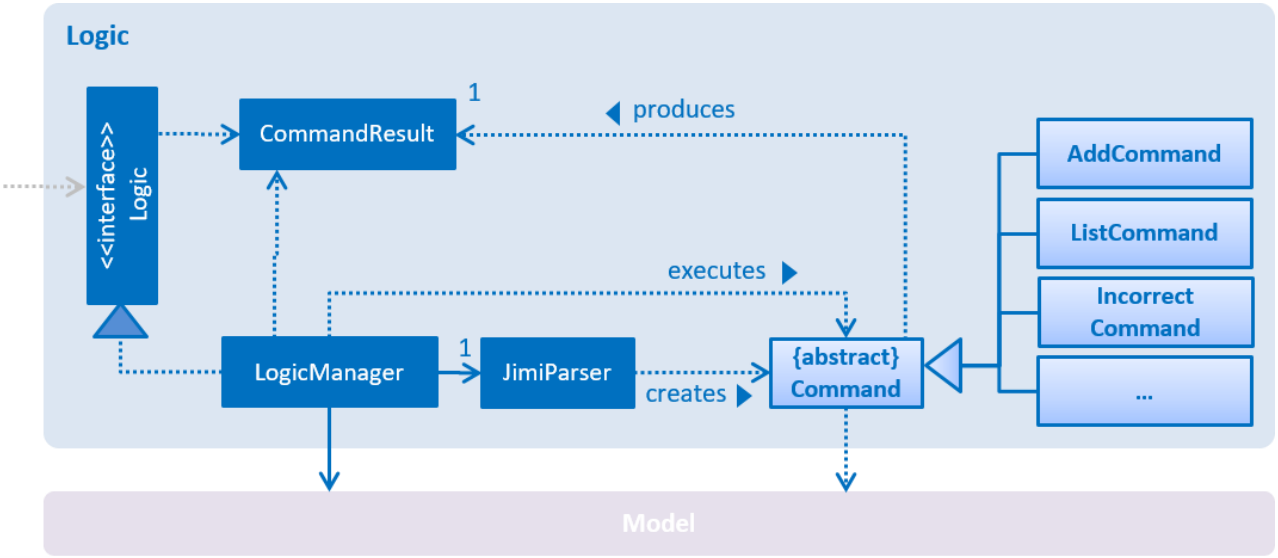


Fig 2.1 Logic Class Diagram

The *Sequence Diagram* shown in fig 2.2 shows how the components interact for the scenario where the user issues the command `delete t3`.

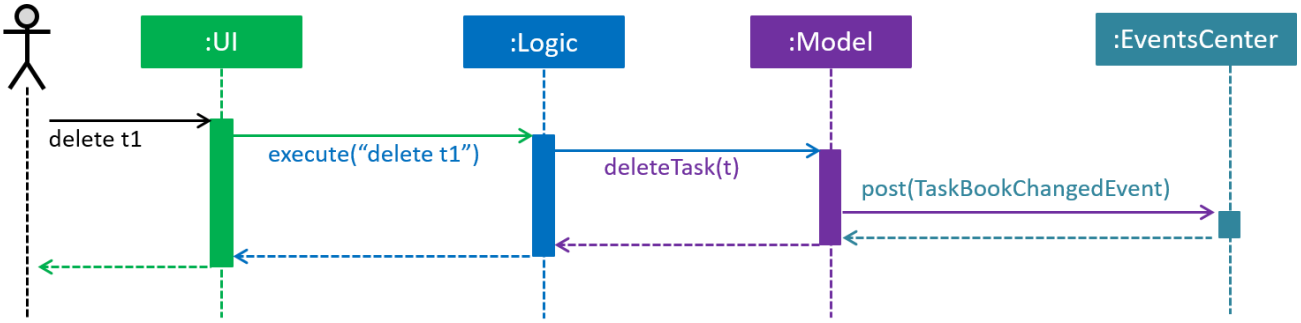


Fig 2.2 Sequence Diagram for Delete Command

Note how the Model simply raises a TaskBookChangedEvent when Jimi's data changes, instead of asking the Storage to save the updates to the hard disk.

The diagram shown in fig 2.3 shows how the EventsCenter reacts to that event, which eventually results in the updates being saved to the hard disk and the status bar of the UI being updated to reflect the 'Last Updated' time.

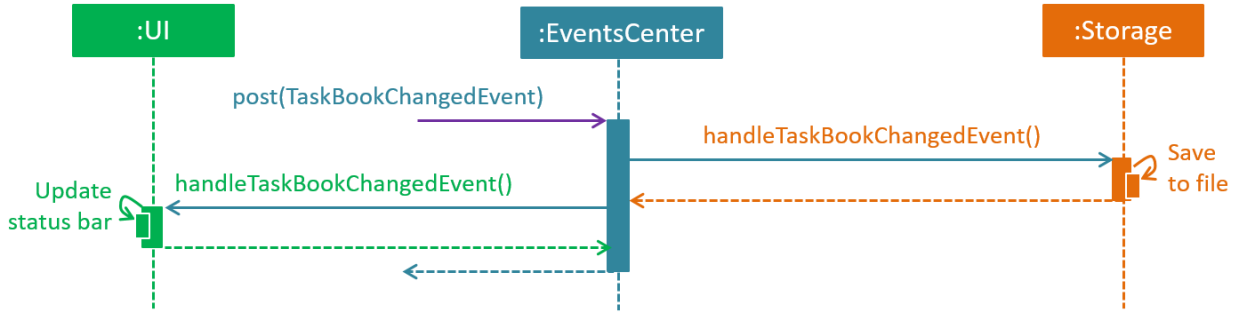


Fig 2.3 Event Sequence Diagram for Delete Command

Note how the event is propagated through the EventsCenter to the Storage and UI without Model having to be coupled to either of them. This is an example of how this Event Driven approach helps us reduce direct coupling between components.

The sections below give more details of each component.

UI component

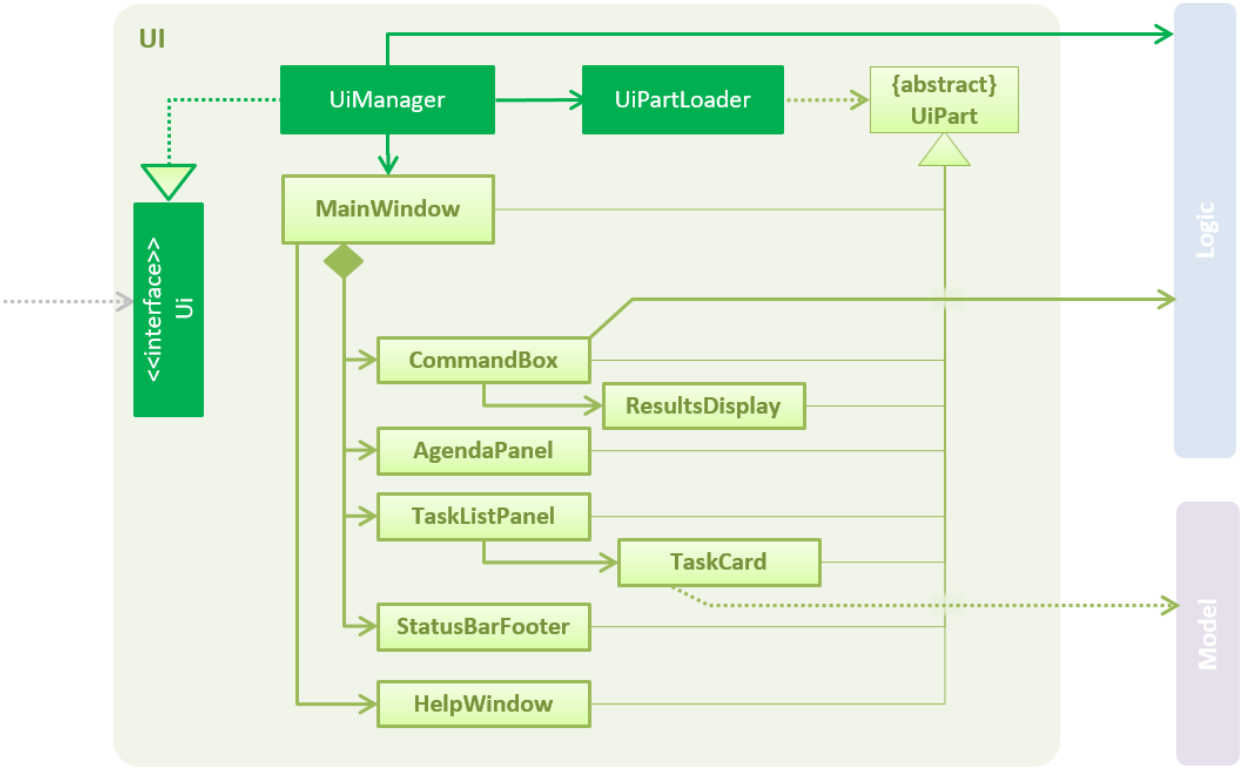


Fig 3.1 User Interface Class Diagram

API : Ui.java

As shown in fig 3.1, the UI consists of a MainWindow that is made up of parts e.g.CommandBox, ResultDisplay, TaskListPanel, StatusBarFooter, AgendarPanel etc. All these, including the MainWindow, inherit from the abstract UiPart class and they can be loaded using the UiPartLoader.

The UI component uses JavaFx UI framework. The layout of these UI parts are defined in matching .fxml files that are in the src/main/resources/view folder.
For example, the layout of the MainWindow is specified in MainWindow.fxml

The UI component:

- Executes user commands using the Logic component.
- Binds itself to some data in the Model so that the UI can auto-update when data in the Model change.
- Responds to events raised from various parts of the App and updates the UI accordingly.

Logic component

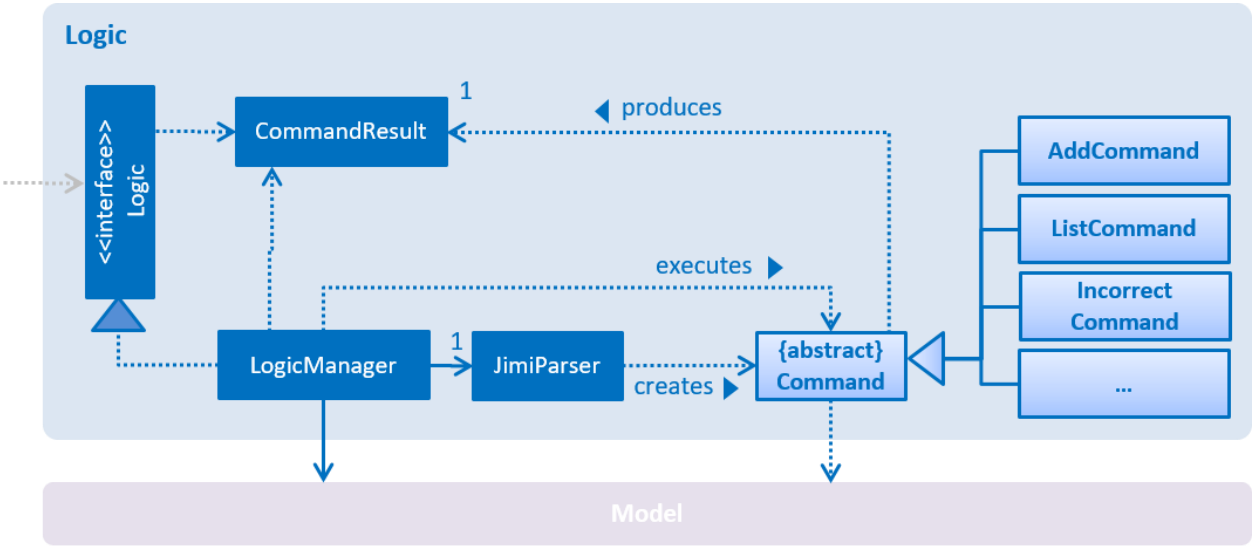


Fig 2.1 Logic Class Diagram

API : `Logic.java`

The Logic component:

- uses the JimiParser class to parse the user command.
- Creates a Command object which is executed by the LogicManager.
- Changes the model (e.g. when adding a task) and/or raise events along with the command execution.
- Encapsulates the result of the command execution as a CommandResult object and passes it back to the Ui.

Shown in fig 4.1 is the Sequence Diagram for interactions within the Logic component for the execute("delete t1") API call.

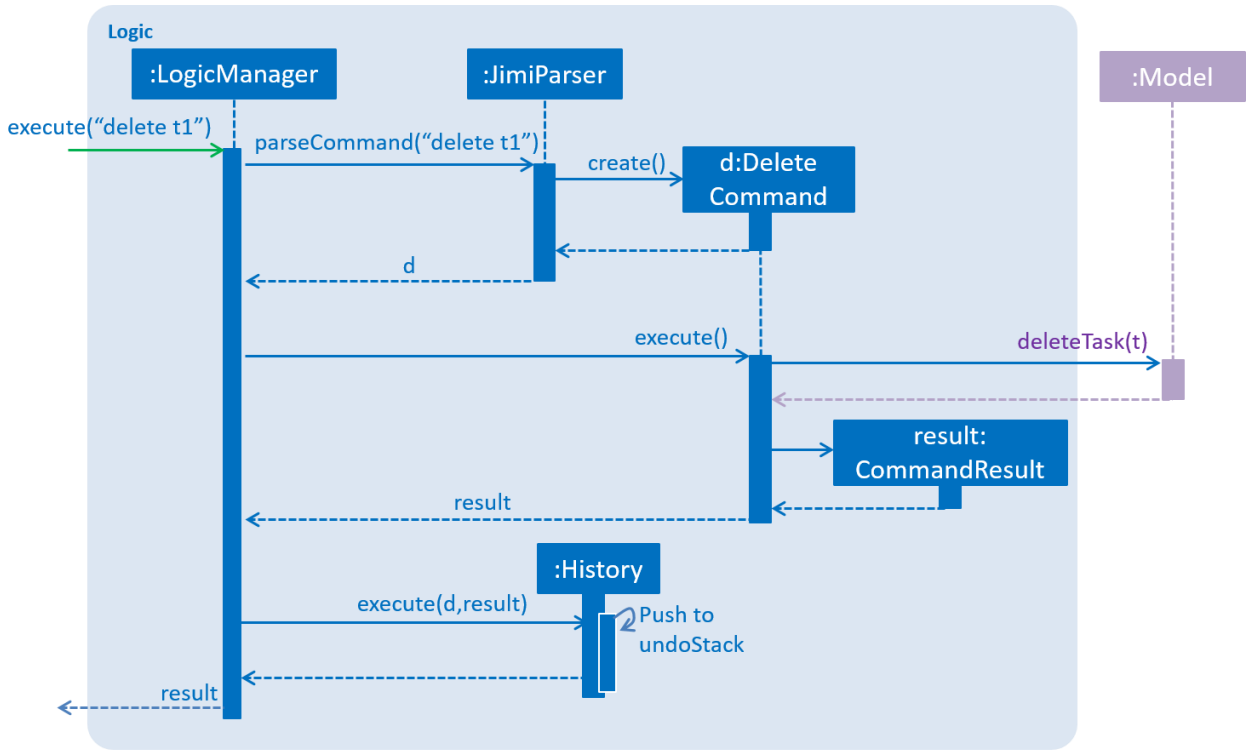


Fig 4.1 Logic Sequence Diagram for Delete Command

Model component

API : [Model.java]

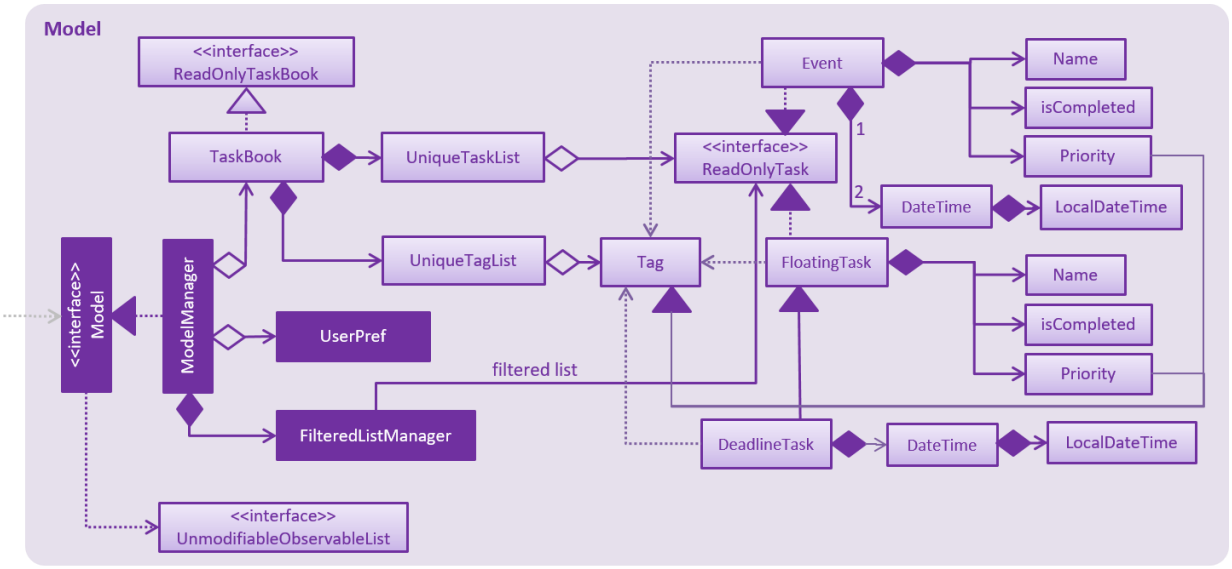


Fig 5.1 Model Class Diagram

The Model component:

- stores a UserPref object that represents the user's preferences.
- stores Jimi's data.
- exposes a UnmodifiableObservableList<ReadOnlyTask> that can be 'observed' e.g. the UI can be bound to this list so that the UI automatically updates when the data in the list change.
- does not depend on any of the other three components.

Storage component

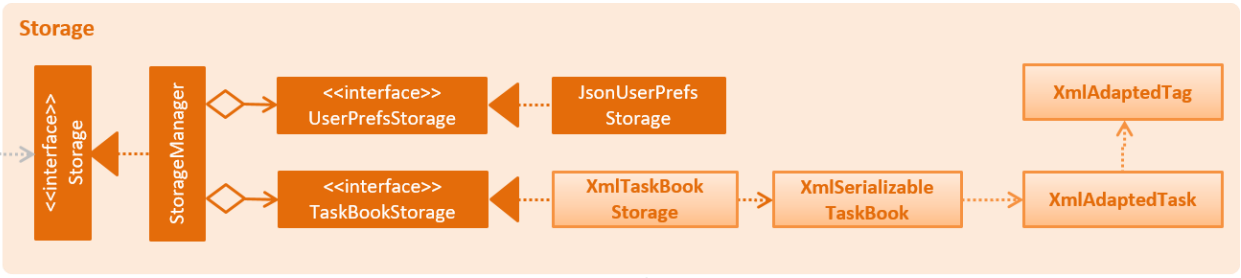


Fig 6.1 Storage Class Diagram

API : `Storage.java`

The Storage component:

- saves UserPref objects in json format and reads it back.
- saves Jimi's data in xml format and reads it back.

Common classes

Classes used by multiple components are in the seedu.jimi.commons package.

Implementation

Logging

We are using java.util.logging package for logging. The LogsCenter class is used to manage the logging levels and logging destinations.

- The logging level can be controlled using the logLevel setting in the configuration file (See Configuration)
- The Logger for a class can be obtained using LogsCenter.getLogger(Class) which will log messages according to the specified logging level
- Currently log messages are output through: Console and to a .log file.

Logging Levels

- SEVERE : Shows that critical problems, which may possibly cause the termination of the application, are detected.
- WARNING : Shows that application can continue operation, but with caution.
- INFO : Shows information of the noteworthy actions by the App
- FINE : Shows details that is not usually noteworthy but may be useful in debugging e.g. print the actual list instead of just its size

Configuration

Certain properties of the application can be controlled (e.g App name, logging level) through the configuration file (default: config.json).

Testing

Tests can be found in the ./src/test/java folder.

In Eclipse:

- To run all tests, right-click on the src/test/java folder and choose Run as > JUnit Test
- To run a subset of tests, you can right-click on a test package, test class, or a test and choose to run as a JUnit test.

Using Gradle:

- See UsingGradle.md for how to run tests using Gradle.

We have two types of tests:

1. **GUI Tests** - These are *System Tests* that test the entire App by simulating user actions on the GUI. These are in the guitests package.
2. **Non-GUI Tests** - These are tests not involving the GUI. They include,

1. *Unit tests* targeting the lowest level methods/classes.
   e.g. seedu.jimi.commons.UrlUtilTest
2. *Integration tests* that are checking the integration of multiple code units (those code units are assumed to be working).
   e.g. seedu.jimi.storage.StorageManagerTest
3. Hybrids of unit and integration tests. These test are checking multiple code units as well as how the are connected together.
   e.g. seedu.jimi.logic.LogicManagerTest

Headless GUI Testing : Thanks to the TestFX library we use, our GUI tests can be run in the *headless* mode. In the headless mode, GUI tests do not show up on the screen. That means the developer can do other things on the Computer while the tests are running.
See UsingGradle.md to learn how to run tests in headless mode.

Troubleshooting tests

Problem: Tests fail because NullPointerException when AssertionError is expected

- **Reason**: Assertions are not enabled for JUnit tests. This can happen if you are not using a recent Eclipse version (i.e. *Neon* or later)
- **Solution**: Enable assertions in JUnit tests as described here.
- Delete run configurations created when you ran tests earlier.

Dev Ops

Build Automation

See UsingGradle.md to learn how to use Gradle for build automation.

Continuous Integration

We use Travis CI to perform *Continuous Integration* on our projects. See UsingTravis.md for more details.

Making a Release

Here are the steps to create a new release.

1. Generate a JAR file using Gradle.
2. Tag the repo with the version number. e.g. v0.1
3. Crete a new release using GitHub and upload the JAR file you created.

Managing Dependencies

A project often depends on third-party libraries. For example, Jimi depends on the Jackson library for XML parsing.

You can automate the managing of these *dependencies* using Gradle. Gradle can download the dependencies automatically, which can remove the hassle of manually downloading and updating the libraries yourself.

It is recommended to use Gradle to manage the dependencies for you, however, if you choose not to use Gradle for whatever reason you can also manually manage these *dependencies* by:
a. Including those libraries in the repo (this bloats the repo size)
b. Requiring developers to download those libraries manually (this creates extra work for developers)

Appendix A : User Stories

Priorities: High (must have) - * * *, Medium (nice to have) - * *, Low (unlikely to have) - *

Priority	**As a ...**	**I want to ...**	**So that I can...**
* * *	new user	list all commands	see all the functionalities of the application
* * *	user	add a new task	
* * *	user	add an event	be reminded of upcoming events to attend
* * *	user	add a floating task	keep track of things I want to complete without a deadline
* * *	user	edit an existing task	modify the details in case a task changes

Priority	As a …	I want to …	So that I can…
* * *	user	remove an existing task	delete a task I no longer care to track
* * *	user	search for tasks with keywords	view all tasks relevant to the keyword easily
* * *	user	view all incomplete tasks	see all tasks that I need to complete
* * *	user	view all completed tasks	refer to all tasks that I have completed
* * *	user	reminded of upcoming tasks	be reminded of incomplete tasks that are due soon
* * *	user	specify a storage location for a file to save the tasks	access it from my own personal location within my system
* * *	user	undo my previous action	easily undo an unwanted action
* *	user	prioritize my tasks	see which tasks are of higher importance/urgency than others
* *	user	set repeating tasks	be reminded of repeated tasks on a timely basis
* *	user	view all tasks due within a specific period of time	know tasks that are required to be completed within set period of time
* *	user	check if I am free at a certain time	know if I can add additional tasks/events to the time-slot
* *	user	do a near-match search	find the tasks I require more conveniently
* *	user	filter out tasks or events with certain characteristics	find all tasks that match the attributes I require
*	user	let the software automatically predict my required command	do what I need more conveniently and quickly

Priority	As a …	I want to …	So that I can…
*	advanced user	assign custom command shortcuts	suit my preferences for better accessibility and convenience
*	user	view current output of the input command in real time	check whether its the expected result of the command

Appendix B : Use Cases

(For all use cases below, the **System** is the TaskBook and the **Actor** is the user, unless specified otherwise)

Use case: List all commands

MSS

1. User requests a list of all commands.
2. App shows list of all commands with guides on how to use the different commands.
   Use case ends.

Use case: Add task/event

MSS

1. User requests to add a task/event.
2. App saves task/event and task/event details to the TaskBook, registers the task/event for future notification/reminders and shows confirmation of successful addition.
   Use case ends.

Extensions

1a. User enters command in invalid format.

1a1. App shows user an error message with correct format needed.
Use case resumes at step 1.

1b. User enters a event with overlapping time with another event.

1b1. App shows user a notification and continues with the addition.
Use case ends.

Use case: Complete task

MSS

1. App shows a list of days/categories.
2. User requests to list tasks/events from a selected day/category.
3. App shows a list of tasks/events from that day/category.
4. User requests to complete a specific task in the list.
5. App marks the task as completed and shows confirmation to the user.
   Use case ends.

Extensions

1a. App shows daily agenda and user requests to complete a specific task in the daily agenda.

Use case jumps to step 5.

3a. The list is empty.

Use case ends.

4a. The given index is invalid.

4a1. App shows an error message. Use case resumes at step 3.

Use case: Delete task/event

MSS

1. App shows a list of days/categories.
2. User requests to list tasks/events from a selected day/category.
3. App shows a list of tasks/events from that day/category.
4. User requests to delete a specific task/event in the list.
5. App deletes the task/event and shows confirmation to the user.
   Use case ends.

Extensions

1a. App shows daily agenda and user requests to delete a specific task/event in the daily agenda.

Use case jumps to step 5.

3a. The list is empty.

Use case ends.

4a. The given index is invalid.

4a1. App shows an error message to user.
Use case resumes at step 3.

Use case: Edit task/event.

MSS

1. App shows a list of days/categories.
2. User requests to list tasks/events from a selected day/category.
3. App shows a list of tasks/events from that day/category.
4. User requests to edit a specific task/event in the list.
5. App edits the details of the task/event and shows confirmation to the user.
   Use case ends.

Extensions

1a. App shows daily agenda and user requests to edit a specific task/event in the daily agenda.

Use case jumps to step 5.

3a. The list is empty.

Use case ends.

4a. The given index is invalid.

4a1. App shows an error message to user.
Use case resumes at step 3.

4b. User enters command in invalid format.

4b1. App shows an error message to user with correct format needed.
Use case resumes at step 3.

4c. User enters new details that are the same as the original details.

4c1. App shows an error message to user.
Use case resumes at step 3.

4d. User enters new time details that overlap with another event's time details.

4d1. App shows notification and continues the editing of the details.
Use case ends.

Use case: Shows today's agenda

MSS

1. User requests to list the agenda of the day
2. App shows a list of tasks due on that day and events held on that day. Use case ends

Extension

2a. There is no tasks due on that day or events hald on that day.

2a1. App shows an empty list.
Use case ends.

Use case: Shows list of tasks & events in a category

MSS

1. User requests to list out the tasks & events in a particular category
2. App shows a list of tasks and/or events that matches that category

2a. There is no tasks or events that matches that category.

2a1. App shows an empty list.
Use case ends.

Use case: Undo action

MSS

1. User requests to undo previous action.
2. App undoes the previous action and shows confirmation to user.
   Use case ends.

Extensions

1a. No previous action was done before.

1a1. App shows an error message to user.
Use case ends.

1b. Previous action is an invalid action to be undone.

1b1. App shows an error message to user.
Use case ends.

Use case: Find task/event

MSS

1. User requests to find a particular task/event using a particular keyword used in the details.
2. App shows a list of tasks/events matching that keyword.
   Use case ends.

Extensions

2a. No such keyword was used before in any task details.

2a1. App shows message to user and displays an empty list to user.
Use case ends.

Use case: Set save directory

MSS

1. User requests to set a new save directory for all the tasks and events.
2. App switches the save directory to the new save directory given and shows confirmation message to user.
   Use case ends.

Extensions

1a. The input new save directory is invalid.

1a1. App shows error message to user.
Use case ends.

1b. The input new save directory is the same as the original save directory.

1b1. App shows error message to user.
Use case ends.

Use case: Clear TaskBook

MSS

1. User requests to clear the TaskBook of all tasks and events.
2. App requests for confirmation with user to clear the TaskBook.
3. User confirms.
4. App clears the TaskBook of all tasks and events and show a confirmation message to user.
   Use case ends.

Extensions

1a. The TaskBook is already empty.

1a1. App shows error message to user.
Use case ends.

3a. User rejects the confirmation.

3a1. App shows message to user.
Use case ends.

Use case: Exit application

MSS

1. User requests to exit the application.
2. Application closes itself.
   Use case ends.

Appendix C : Non Functional Requirements

1. Should work on any mainstream OS as long as it has Java 1.8.0_60 or higher installed.
2. Should be able to hold up to 1000 Tasks.

3. Should come with automated unit tests and open source code.
4. Should favor DOS style commands over Unix-style commands.
5. Should support natural language processing with natural language commands.
6. Should be able to do all functions through the command-line interface.
7. Should be able to be accessed offline.
8. Should load within 1 second of opening the program.
9. Should be able to display full list of tasks within 1 second.
10. Should be able to save and backup tasks into a file for recovery or portability.
11. Should not cause data corruption when program is closed abruptly.
12. Should be able to hold tasks up to one year onwards.
13. Should recover from major errors within 1 second.

Other requirements can be found in the project constraints section of our team's module handbook.

Appendix D : Glossary

Application program interface

A set of routines, protocols, and tools for building software applications.

Mainstream OS

Windows, Linux, OS-X

Natural Language Commands

Commands formatted in a language that has developed naturally in use and is intuitive for humans to understand. (as contrasted with an artificial language or computer code).

Natural Language Processing

A branch of artificial intelligence that deals with analyzing, understanding and generating the languages that humans use naturally in order to interface with computers in both written and spoken contexts using natural human languages instead of computer languages.

Command-line interface

User interface to a computer's operating system or an application in which the user responds to a visual prompt by typing in a command on a specified line, receives a response back from the system, and then enters another command, and so forth.

Appendix E : Product Survey

Task Managers	**Strengths**	**Weaknesses**
Todoist	Has a very simple design. Offers a mobile app. Has a feature where users are encouraged to earn "Todoist Karma", to track their productivity trends as they finish their tasks.	Free version is limited in its capabilities and is not well-encrypted. Some of the mobile apps have design issues (like being unable to sort tasks). Free version does not come up with some features like reminders, filters, labels, and templates.
Trello	Can divide projects up by tasks, and then edit those tasks with descriptions, labels, checklists, and even attachments. Is particularly helpful for teams working on separate tasks toward a greater project goal, where the tasks are in need of a pipeline.	Has no good way to use this system to prioritize tasks between projects.
Google Keep	Easy on the eyes. Easy to use. Integrates with desktop/mobile very well. As expected from google, it integrates well with other google products too. Voice memos feature. Images feature. Able to retrieve deleted items in archive. Has reminders. Reminders can be set location-based. Can share lists.	No chronological representation of reminders.
Google Calendar	Events are shown clearly on a calendar interface. Integrated with Google reminders to give user	No other way to prioritise tasks other than setting a deadline for it.

Task Managers	**Strengths**	**Weaknesses**
	reminders. Allows addition of tasks with deadlines unlike most to-do apps. Puts floating tasks and tasks with deadlines together for easy reviewing and reassessing of what to do next.	
Wunderlist	Easy to use for beginners. Has reminder features. Allows creation and sharing of lists and setting up deadlines to get them done. Is useful for collaboration on these lists with other people. Comes with a mobile and a smartwatch app.	Advanced users have to pay if they want access to better features that beginners might not want to use as much. More of a power list making tool than a true GTD app.
Any.do	Can sync lists across all devices. Can share lists with other people. Allows creation of recurring tasks. Comes with a voice-entry feature, which allows you to create a list using voice commands. Comes with a feature "Any.do Moment" which focuses on just the tasks due on the day itself.	Comes with limited themes. Users have to pay to use their more advanced features.
Remember the Milk	Allows syncing across devices. Tasks can be set with deadlines and priorities. Works well with serveices like Google Calendar and Evernote. Allows easy sharing of lists and tasks with friends and colleagues. Advanced feature to break jobs into sub-tasks. Advanced feature to use colour tags to separate different kinds of lists.	Users have to pay to use their advanced features. Users have to pay for the ability to set reminders.
Clear	Comes with a mobile app. Differs from other task managers, as you can	Users have to pay to use this app.

Task Managers	**Strengths**	**Weaknesses**
	use gestures, such as swiping, to create, rearrange and mark tasks as complete on the mobile version, instead of tapping on the mobile. Allows creation of separate lists. Built-in reminder feature.	
Habitica	Comes with a mobile app. Comes as an RPG-style game to motivate users to complete tasks that are tracked in the app.	May be too quirky for more serious users or users who do not play RPGs.
OmniFocus	Flexible in that it can be as simple or as complex as what the user wants it. Allows viewing and organising tasks in different ways to suit the wants of the user. Allows user to just use a keyboard shortcut to add a task anytime while on the desktop. Desktop version syncs with some email clients to turn emails into tasks. iOS 8 devices allows addition of tasks into OmniFocus from other iOS apps. iOs 8 devices has a quick-entry button into the OmniFocus App. Syncs across all iOS devices and Mac.	Only available for iOS devices and Mac. Users have to pay to use this app.