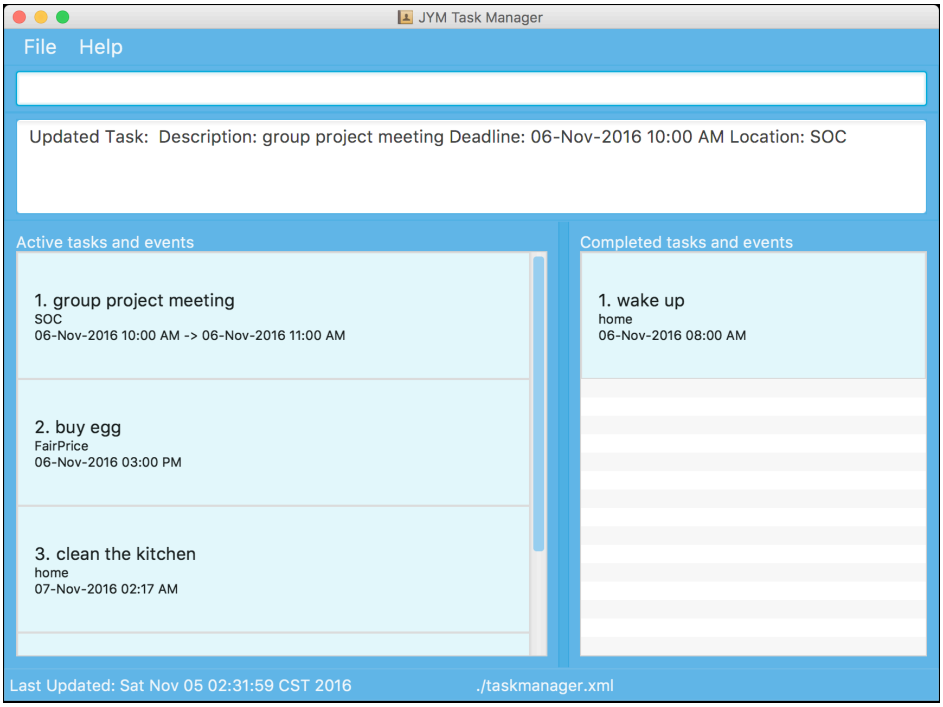
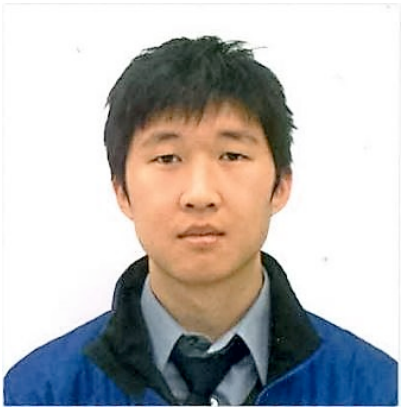
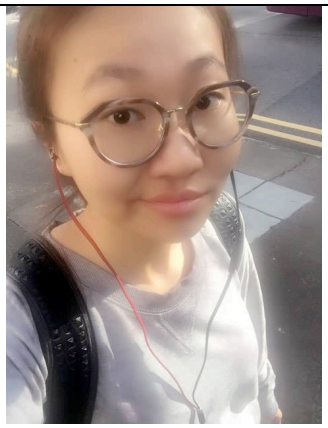
# JYM




Supervisor: Nishant Budhdev


| | | |
|---|---|---|
|  |  |  |
| Murray Chen | Zihua Huang | Yaquan Wang |

# Acknowledgements

Some parts of this sample application were inspired by these excellent Addressbook Repositories by *Damith C. RAJAPAKSE*.

# About Us

## Nishant Budhdev

Role: Project Mentor

## Murray Chen

Role: Team Lead, Developer

Responsibilities: Code Quality

## Zihua Huang

Role: Developer

Responsibilities: Integration, Git Expert, Scheduling and Tracking

## Yaquan Wang

Role: Developer

Responsibilities: Documentation, Testing, Deliverables and Deadlines

# User Guide



## 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 version of JYM from the releases releases (../../../releases) tab

3. Copy the file to the folder you want to use as the home folder for your JYM.
4. Double click the file to start the app. The GUI should appear in a few seconds.
5. Type the command in the command box and press `Enter` to execute it.

   e.g. typing **`help`** and pressing `Enter` will open the help window.
6. Some example commands you can try:

- **`list`** : lists all tasks and events.
- **`add`** `Insert Program Stub for Testing due tomorrow at 5pm` :
  adds a task named with the given description to the task list.
- **`delete`** `3` : deletes the 3rd task shown in the current list.
- **`exit`** : exits the app

Refer to the Features section below for details of each command.

## Features

**Command Format** [T15-C4][JYM]

*Words in `UPPER_CASE` are the parameters.*

Items in `SQUARE_BRACKETS` are optional.

*Items with `...` after them can have multiple instances.*

The order of parameters is fixed.

## Viewing help : `help`

Format: `help`

## Adding a task or event: `add`

Adds a task or event to the task list.

Format: `add [TASK/EVENT] DESCRIPTION [due/at/by DATE START_TIME END_TIME] [PRIORITY]`

Adding tasks or events can also be done through simple English.

If only one time is specified, it will be interpreted as a deadline. Otherwise, the event will use the input as start and end time.
If no command keyword is specified, the app assumes the given command is to add, and will interpret the input as a task or event depending on whether a start and end time is given or not.

Examples:

`do laundry at home JULY 24 5 PM`

*Adds a task with the description `do laundry` and the deadline `5PM 07/24`*

`write sql queries by tomorrow 9pm`

Adds a task with the description `write sql queries` and the deadline `9PM [tomorrow]`, with tomorrow being whatever date the next day is.

* `dinner with jack tomorrow at 5 pm to 6pm`

Adds an event with the description `dinner with jack` with the time 5 to 6 pm tomorrow.

## Listing all tasks and events: `list`

Shows a list of all the tasks and/or events in the program.

Format: `list`

The list command alone lists all upcoming tasks and events.

When appended with task or event, the program will show only the requested input (tasks or events).

By default list shows upcoming active tasks and events.

## Finding all tasks and events with a given keyword in the description or title: `find`

Searches for tasks and events whose descriptions or titles contain any of the given keywords or dates.

Format: `find KEYWORD [MORE_KEYWORDS]`

- By appending certain keywords onto the search, one can filter results. e.g. `completed` will search for completed tasks only, `ordered` will search for tasks with all the keywords in the given order and grouped together.
- Search is not case sensitive. e.g. `ChiCKEN` will match `cHIcken`
- Order of the keywords does not matter. e.g. `do this` will match `this do` . This can be changed by adding specific keywords.
- The description and date are both searched. e.g. `eat dinner july 30` will result in tasks and events from july 30 matching `eat dinner`
- Tasks matching at least one keyword will be returned (not including dates). These settings can be changed by setting flags in the command.

Examples:

*`find write SQL queries july 21`*

*Returns upcoming tasks with `write SQL queries` in the description on july 21.*

`find birthday all`

Returns all events with `birthday` in the description or title.

* `find CS2103 final`

Returns upcoming tasks and events with `CS2103 final` in the description

## Undo mistaken commands: `undo` v1.5-C4][JYM]

Reverses the last command done. Repeated calls to this command will undo each command in the reverse order of that which they were called. Can only go back up to 20 commands in history via undos.

Format: `undo`

## Marking a task complete: `complete`

Marks the given task as completed from the active task list. Can be reversed if done immediately after.

Format: `complete INDEX`

> Marks the task at the specified `INDEX`.
> The index refers to the index number shown in the most recent listing
>
> The index **must be a positive integer** 1, 2, 3, …
>
> Can mark multiple indices to be completed.

Examples:

```
list
```

```
complete 2
```

*Marks the 2nd task in the active task list complete.*

```
find write test for
```

```
complete 1
```

Marks the 1st tasks as complete in the results of the find command

## Updating a task: `update`

Updates a given task. [T15-C4][JYM]

Format: `update INDEX [DESCRIPTION] [by/at DATE TIME]`

> Updates the tasks at the specified `INDEX`.
>
> Will update the task depending on what is supplied in the input. If no date or time is provided, the original task/event time will stay the same. Likewise with the description.

Examples:

*`update 2 by 09/08/2016 8PM`*

*Updates the second task to have an updated deadline.*

`update 1 Redo Mission class because failed code quality check by tomorrow 9pm`

Updates the first task to have the updated description `Redo Mission class because failed code quality check` and the updated date `9PM [tomorrow]` with tomorrow being whatever date the next day is.

## Clearing all entries: `clear`

Clears all tasks and events from the program.

Format: `clear`

## Exiting the program: `exit`

Exits the program

Format: `exit`

## Setting the data storage location: `saveto`

Sets the data storage path. Must be a valid path.

Format: `saveto PATH`

Example:

* `saveto MyDropbox`

It will create a folder `MyDropbox` under the current path where the program is and save the data file when user start to add task.

### Deleting tasks: `delete`

Deletes tasks or events for when you wish to remove them entirely from the list.

Format: `delete INDEX`

### Saving the data

Data are saved in the hard disk automatically after any command that changes the data.

There is no need to save manually.

# FAQ

**Q**: How do I transfer my data to another Computer?

**A**: Install the app in the other computer and overwrite the empty data file it creates with the file that contains the data of your previous session.

**Q**: How do I backup my data?

**A**: The easy way is to use the `saveto PATH` command, where the PATH points to your cloud folder (Google drive, dropbox, iCloud). In that way, everytime when you save your data, it will automatically save inside the cloud folder. Also, remember to set your cloud folder sync automatically.

# Command Summary

| Command | Format |
| --- | --- |
| | `DESCRIPTION [at LOCATION] [at/by DATE TIME]` OR `DESCRIPTION [by/at` |

| | |
|---|---|
| Add | `DATE START_TIME to END_TIME]` [JYM] |
| Update | `update INDEX [DESCRIPTION] [by/at DATE START_TIME to END_TIME]` |
| Clear | `clear` |
| Undo | `undo` |
| Complete | `complete INDEX` |
| Find | `find KEYWORD [MORE_KEYWORDS]` |
| List | `list` |
| Help | `help` |
| Saveto | `saveto PATH` |
| Delete | `delete INDEX` |

# Developer Guide



# 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 (Do the steps 2 onwards given in this page (http://www.eclipse.org/efxclipse/install.html#for-the-ambitious))
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

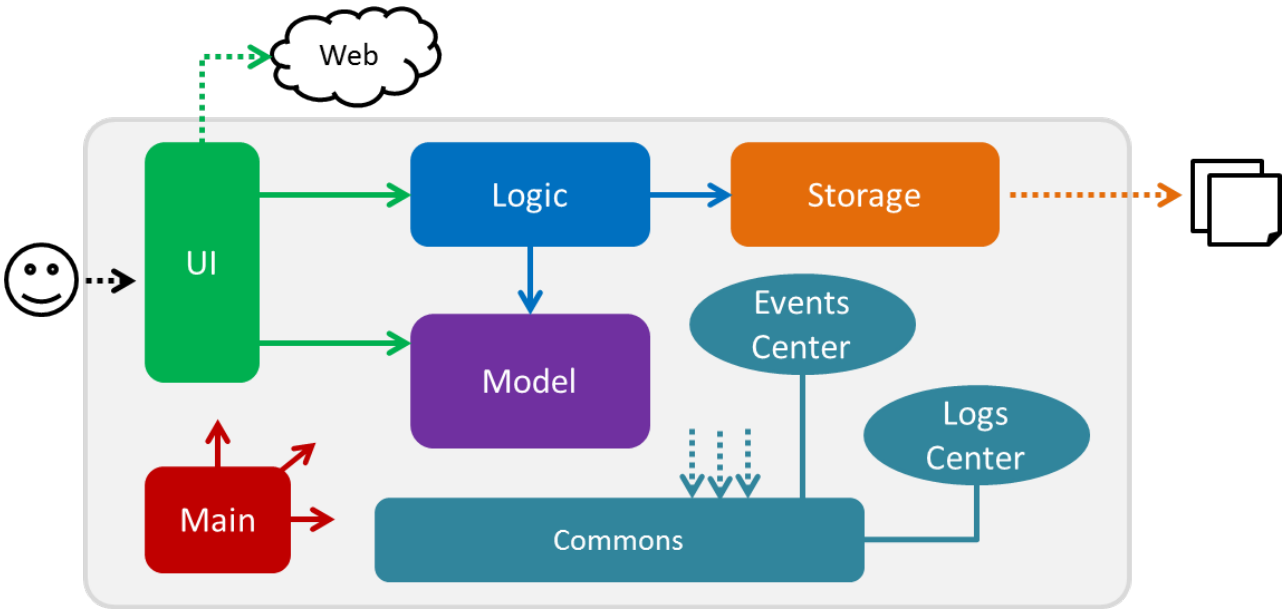
in the prerequisites above) [T15-C4][JYM]

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.

# Design

## Architecture




The ***Architecture Diagram*** given above explains the high-level design of the App. Given below is a quick overview of each component.

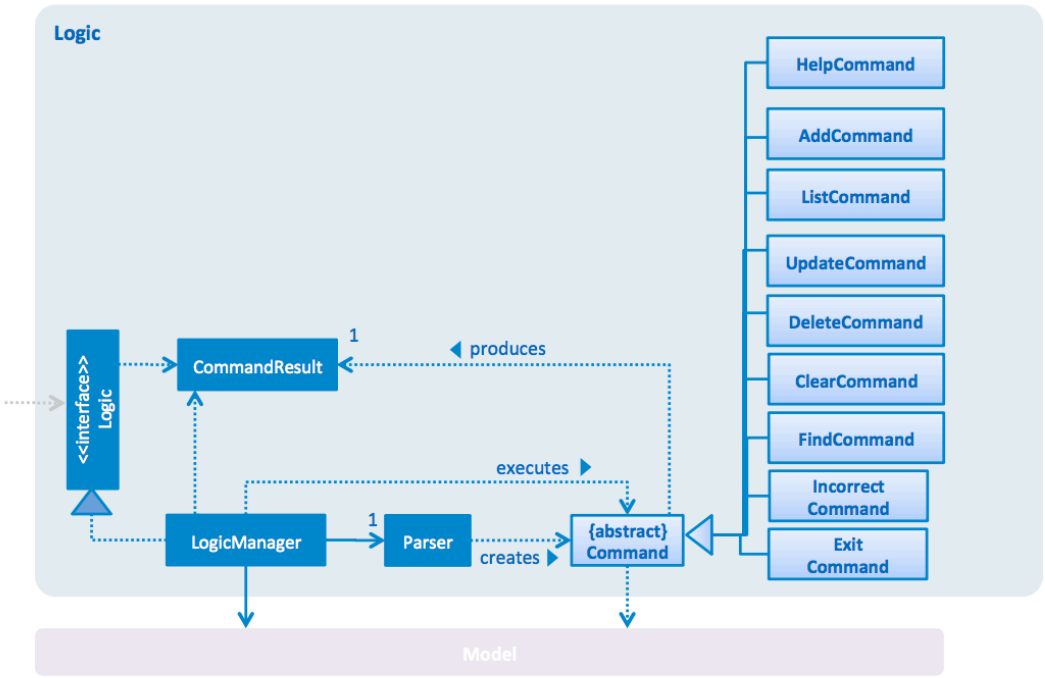
`Main` has only one class called `MainApp` [T15-C4][JYM] (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/src/main/java/jym/manager/MainApp.java). It is responsible for,
*At app launch: Initializes the components in the correct sequence, and connect them up with each other.*
At shut down: Shuts down the components and invoke cleanup method where necessary.

**`Commons`** represents a collection of classes used by multiple other components.
Two of those classes play important roles at the architecture level.
*`EventsCentre` : This class (written using Google's Event Bus library (https://github.com/google/guava/wiki/EventBusExplained))*
*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.

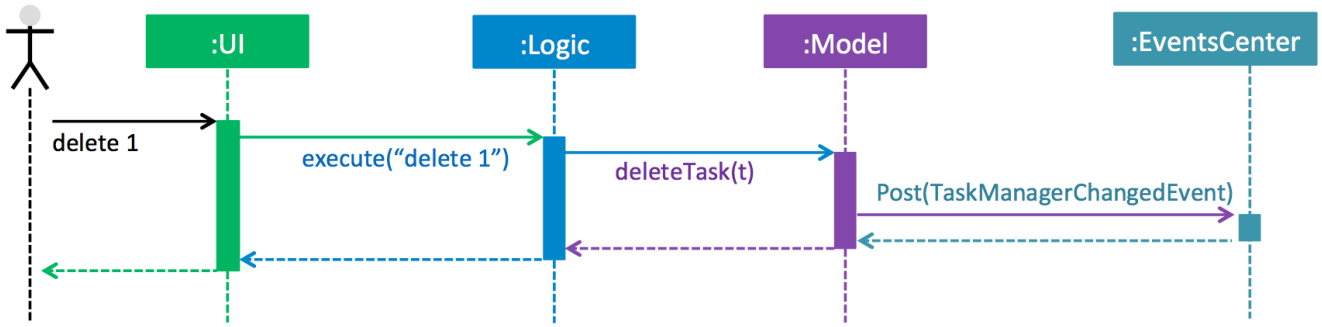
The rest of the App consists four components.
***`UI`** : The UI of tha App.*
**`Logic`** : The command executor.
***`Model`** : Holds the data of the App in-memory.*
**`Storage`** : Reads data from, and writes data to, the hard disk.

Each of the four components
*Defines its API 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 given below) defines it's API in the `Logic.java`
interface and exposes its functionality using the `LogicManager.java` class.

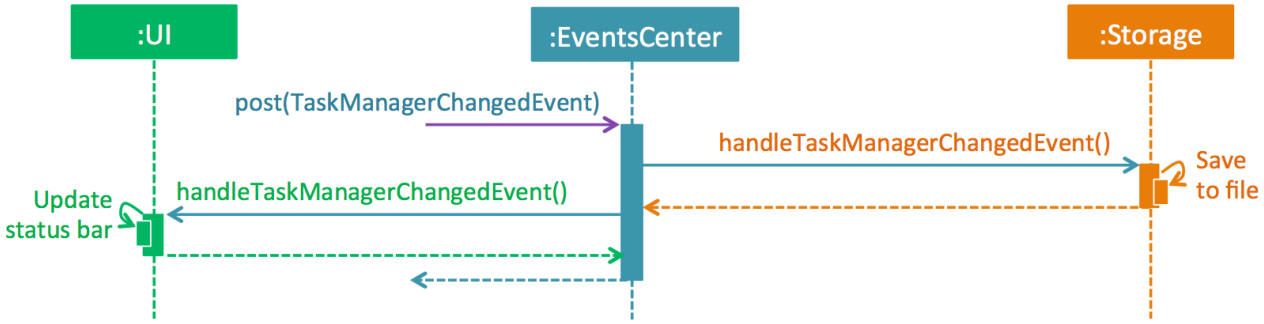



The *Sequence Diagram* below shows how the components interact for the scenario where the user issues the
command `delete 1`.




> Note how the `Model` simply raises a `TaskManagerChangedEvent` when the Task Manager data are changed,
> instead of asking the `Storage` to save the updates to the hard disk.

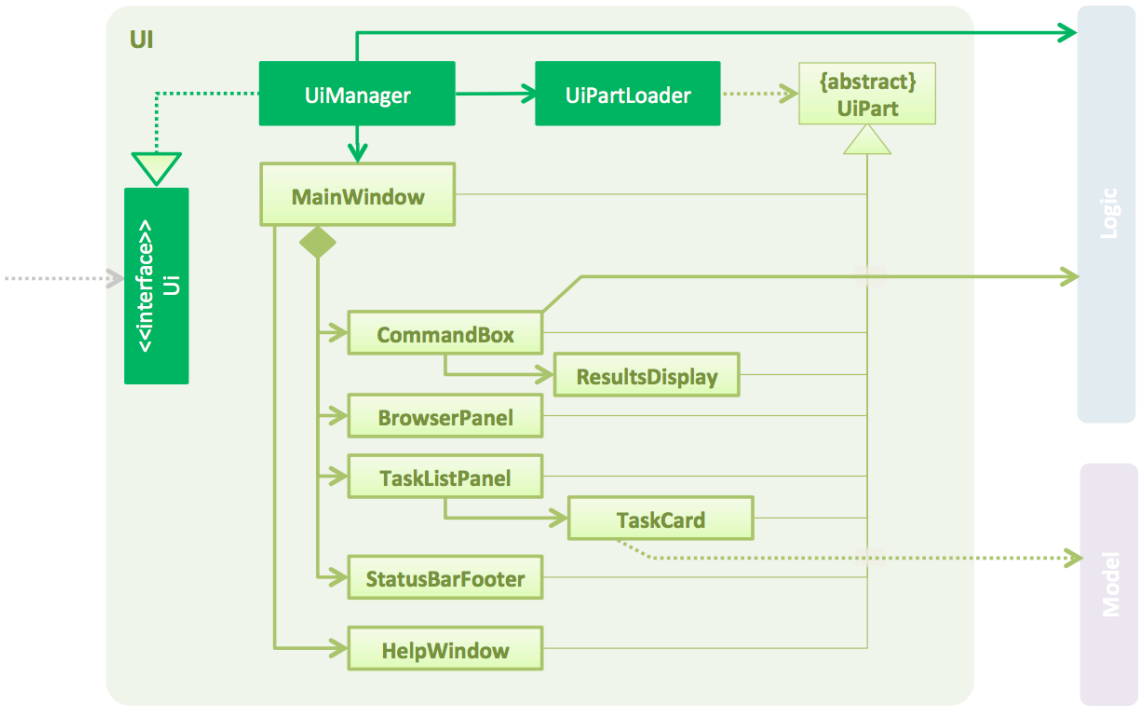
The diagram below shows how the `EventsCenter` reacts to that event, which eventually results in the updates
being saved to the hard disk and the status bar of the UI being updated to reflect the 'Last Updated' time.




> 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




**API** : `Ui.java` (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/src/main/java/jym/manager/ui/Ui.java)

The UI consists of a `MainWindow` that is made up of parts e.g. `CommandBox` , `ResultDisplay` , `TaskListPanel` ,
`StatusBarFooter` , `BrowserPanel` etc. All these, including the `MainWindow` , inherit from the abstract `UiPart` class
and they can be loaded using the `UiPartLoader` .

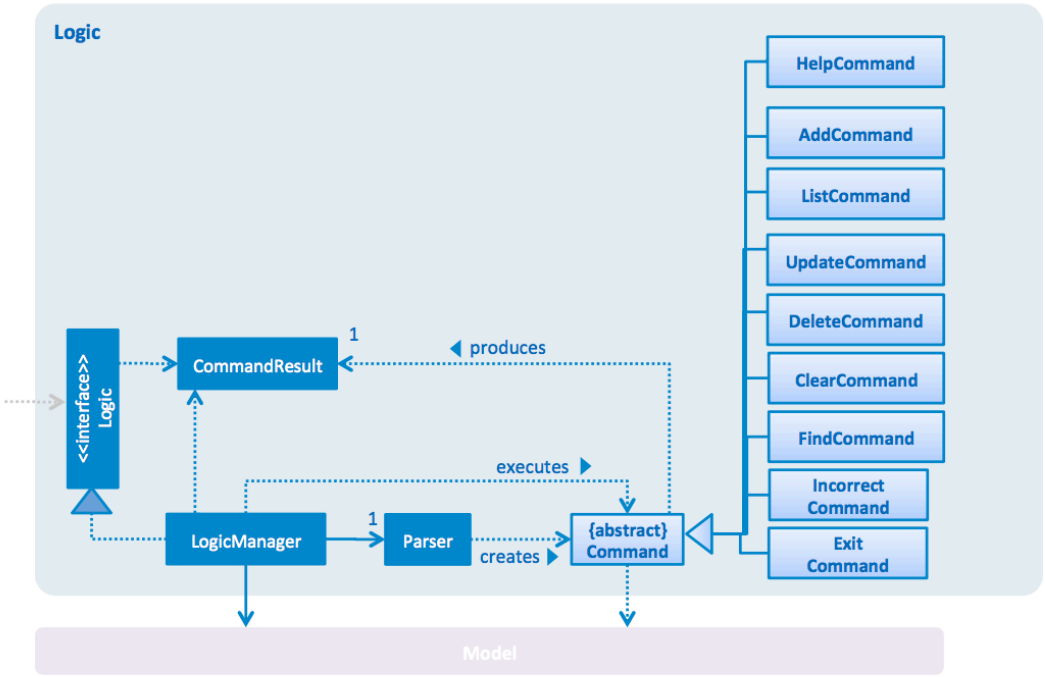
The `UI` component uses JavaFx UI framework. The layout of these UI parts are defined in matching `.fxml` files
that are in the `src/main/resources/view` folder.

For example, the layout of the `MainWindow` (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/src/main/java/jym/manager/ui/MainWindow.java) is specified in

`MainWindow.fxml` (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/src/main/resources/view/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

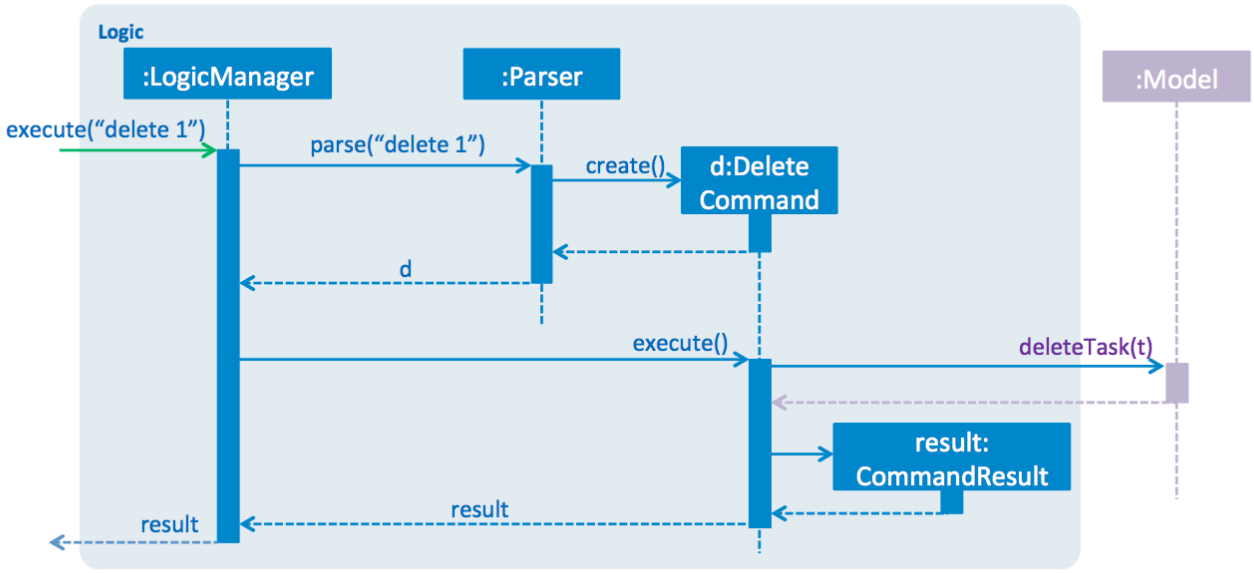



**API** : `Logic.java` (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/src/main/java/jym/manager/logic/Logic.java)

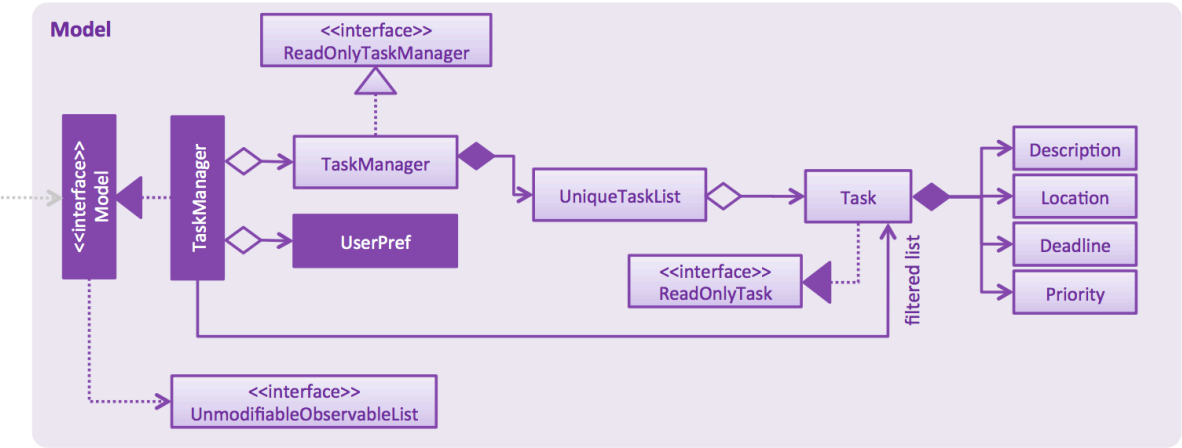
1. `Logic` uses the `Parser` class to parse the user command.
2. This results in a `Command` object which is executed by the `LogicManager`.
3. The command execution can affect the `Model` (e.g. adding a task) and/or raise events.
4. The result of the command execution is encapsulated as a `CommandResult` object which

is passed back to the `Ui`. [T15-C4][JYM]

Given below is the Sequence Diagram for interactions within the `Logic` component for the `execute("delete 1")` API call.



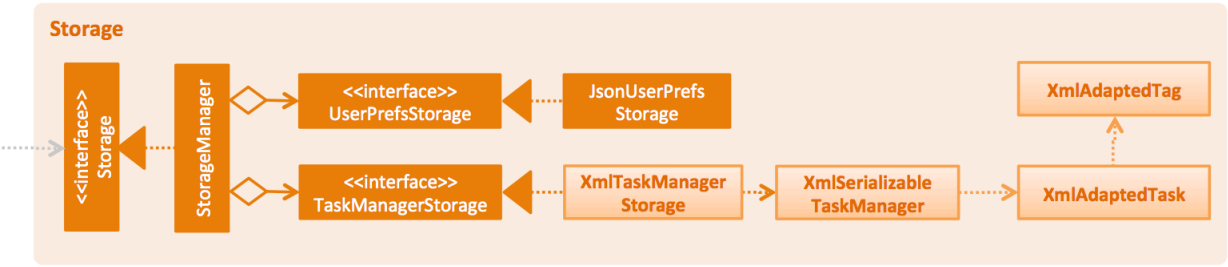

# Model component



**API** : `Model.java` [T15-C4][JYM]
(/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/src/main/java/jym/manager/model/Model.java)

The `Model` ,
*stores a `UserPref` object that represents the user's preferences.*
stores the Task Manager 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




**API** : `Storage.java`
(/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/src/main/java/jym/manager/storage/Storage.java)

The `Storage` component,
*can save `UserPref` objects in json format and read it back.*
can save the Task Manager data in xml format and read it back.

## Common classes

Classes used by multiple components are in the `jym.manager.commons` package.

# Implementation

# Logging [T15-C4][JYM]

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` : Critical problem detected which may possibly cause the termination of the application
- `WARNING` : Can continue, but with caution
- `INFO` : Information showing the noteworthy actions by the App
- `FINE` : 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**:

If you are not using a recent Eclipse version (i.e. *Neon* or later), enable assertions in JUnit tests
as described here (http://stackoverflow.com/questions/2522897/eclipse-junit-ea-vm-option).

- 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 (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/docs/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,

3. *Unit tests* targeting the lowest level methods/classes.

   e.g. `jym.manager.commons.UrlUtilTest`
4. *Integration tests* that are checking the integration of multiple code units
   (those code units are assumed to be working).

   e.g. `jym.manager.storage.StorageManagerTest`
5. Hybrids of unit and integration tests. These test are checking multiple code units as well as
   how the are connected together.

   e.g. `jym.manager.logic.LogicManagerTest`

**Headless GUI Testing** : [T15-C4][JYM]
Thanks to the TestFX (https://github.com/TestFX/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 (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/docs/UsingGradle.md#running-tests) to learn how to run tests in headless mode.

# Dev Ops

## Build Automation

See UsingGradle.md (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/docs/UsingGradle.md) to learn how to use Gradle for build automation.

## Continuous Integration

We use Travis CI (https://travis-ci.org/) to perform *Continuous Integration* on our projects.
See UsingTravis.md (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/docs/UsingTravis.md) for more details.

## Making a Release

Here are the steps to create a new release.

1. Generate a JAR file using Gradle (/Users/jameshuang1996/Desktop/github/CS2103TeamC4/main/docs/UsingGradle.md#creating-the-jar-file).
2. Tag the repo with the version number. e.g. `v0.1`
3. Crete a new release using GitHub (https://help.github.com/articles/creating-releases/) and upload the JAR file your created.

## Managing Dependencies [T15-C4][JYM]

A project often depends on third-party libraries. For example, JYM depends on the Jackson library (http://wiki.fasterxml.com/JacksonHome) for XML parsing. Managing these *dependencies*
can be automated using Gradle. For example, Gradle can download the dependencies automatically, which
is better than these alternatives.

a. Include those libraries in the repo (this bloats the repo size)

b. Require 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 | see usage instructions | refer to instructions when I forget how to use the App |
| * * * | user | add a new task or event with a given deadline or date and priority | remember what tasks I have to do in the future. |
| * * * | user | add a floating task with no given deadline or date | work on long term tasks that have no given deadline. |
| * * * | user | mark a task or event as completed | refer to which tasks I have finished or not finished |
| * * * | user | find a task or event by name | locate details of tasks/events without having to go through the entire list |

| | | | |
|---|---|---|---|
| * * * | user | list upcoming tasks in sorted order [MSCA][JYM] | determine which tasks I should do next |
| * * * | user | edit tasks | update tasks if things change. |
| * * * | user | undo recent commands | revert tasks/events created by mistake. |
| * * * | user | specify the location to place the data storage | do things with it, like sync it to my Dropbox. |
| * * | user | add tasks through plain English | type more naturally than having to write in commands and flags. |

# Appendix B : Use Cases

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

## Use case: Mark Task completed.

### MSS

1. User requests to view list of active tasks
2. System shows list of tasks
3. User requests to mark a task as completed from the list.
4. System marks task as completed

   Use case ends.

### Extensions

2a. The list is empty

> Use case ends

3a. The given index is invalid

3a1. JYM shows an error message [T15-C4][JYM]

Use case resumes at step 2

### Use case: Update Task

#### MSS

1. User requests to view list of active tasks
2. System shows list of tasks
3. User requests to update a task with given description and/or deadline
4. System updates task.
   Use case ends.

#### Extensions

2a. List is empty

Use case ends

3a. Given index invalid

3a1. JYM shows error message

Use case resumes at step 2

3b. Invalid deadline supplied by user

3b1. JYM returns error message

Use case resumes at step 2

{More to be added}

## Appendix C : Non Functional Requirements

1. Program should load within 5 seconds
2. Storage file should be limited to 100MB default (can be changed by user)
3. Should work on any mainstream OS

4. Should hold up to 1000 tasks/events on the active task list at any time [T13-C4][Jayan]
5. Comes with automated unit tests.
6. Commands should be intuitive and easy to use.
7. Interface is simple and easy to understand for beginners (i.e. people seeing it for the first time are not confused by what box does what).

{More to be added}

# Appendix D : Glossary

**Mainstream OS**

| |
|---|
| Windows, Linux, Unix, OS-X |

**Task**

| |
|---|
| An activity that needs to be completed by a certain date or time. Contains a description, deadline, and priority. |

**Event**

| |
|---|
| An activity that happens within a set boundary of time, with a start and end time on a given date.<br>Contains a description, event date and time, priority, and location. |

**Floating Task**

| |
|---|
| A task with no set deadline. |

**Active Task List**

| |
|---|
| List of tasks that have yet to be marked complete and whose deadlines have not passed yet. |

# Appendix E : Product Survey

**Google Calendar Quickadd**

Satisfies many stories, but cannot add multiple events simultaneously, and does not hold support for both tasks and events.

Everything is an event, and although one can create them in a certain way to make them act like tasks, it can be tedious and not optimal to do so.

**iStudiez Pro**

Satisfies many stories, in particular having a nice GUI and being able to view the task list in many formats. However, there is no shortcut to add tasks, and you cannot type everything in one line without tabbing over to a separate text box.

**S Planner (Samsung Phone Calendar, Android 4.2.2)**

Is only a smartphone app. Harder to type in tasks and events on the phone.

Syncs with Google Calendar.

**GO Note Widget (to-do list add-on for GO Launcher)**

Clean and simple interface (clutter-free)

Can mark tasks with colored labels

No option to state the start or end time of a task

Smartphone app.