Developer Guide

- Introduction
- Setting Up
- Design
- Implementation
- Testing
- Dev Ops
- Appendix A: User Stories
- Appendix B: Use Cases
- Appendix C: Non Functional Requirements
- Appendix D: Glossary
- Appendix E : Product Survey

Introduction

Taskell is a simple software for users to keep track of their daily tasks and manage their busy schedule. Keyboard lovers will be able to experience the full benefit of Taskell as it implements a command-line interface.

This developer guide will help you understand the design and implementation of Taskell. You get to know how Taskell works and how you can contribute for further development. This guide follows a top-down approach by giving an overview of the essential components first, followed by thorough explanation subsequently.

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)
4. **Buildship Gradle Integration** plugin from the Eclipse Marketplace

Importing the Project into Eclipse

1. Fork this repository, 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' configuration 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

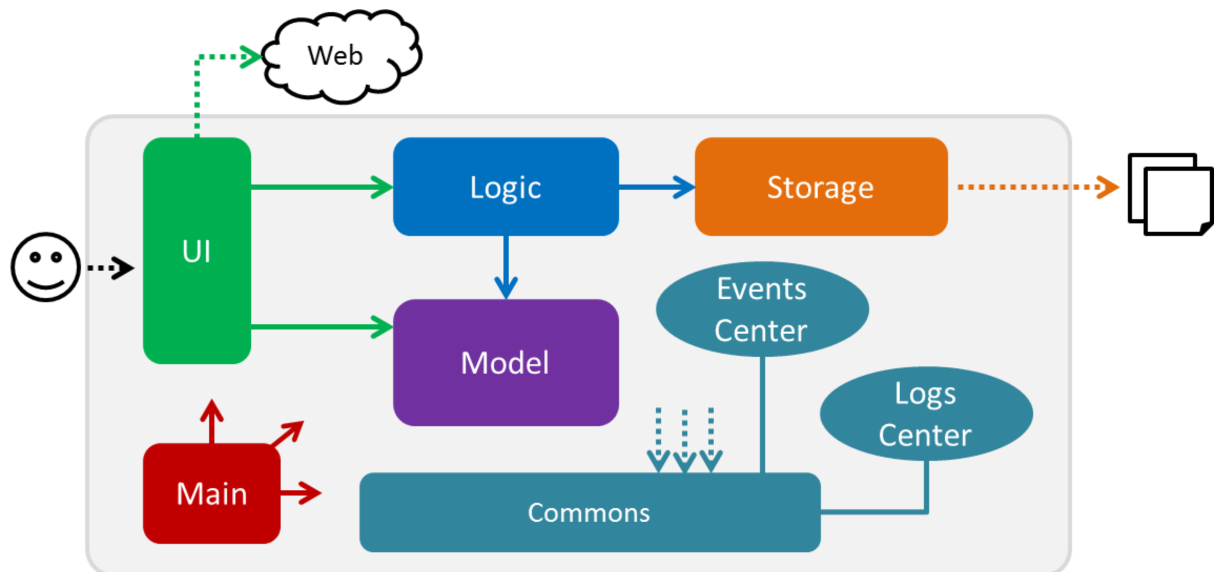

Figure 1: Architecture Diagram

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

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

- At application launch: Initializes the components in the correct sequence, and connects them up with each other.
- At shut down: Shuts down the components and invokes 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 : Used by components to communicate with other components using events (i.e. a form of *Event Driven* design)(written using Google's Event Bus library)
- LogsCenter : Used by many classes to write log messages to the Application's log file.

The rest of the Application consists four components.

- **UI** : UI of the Application.
- **Logic** : Command executor.
- **Model** : Data Holder of the Application in-memory.
- **Storage** : Data read from, and written to the hard disk.
- **History** : Data holder of Application's command history (for undo only).

Each of the five components

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

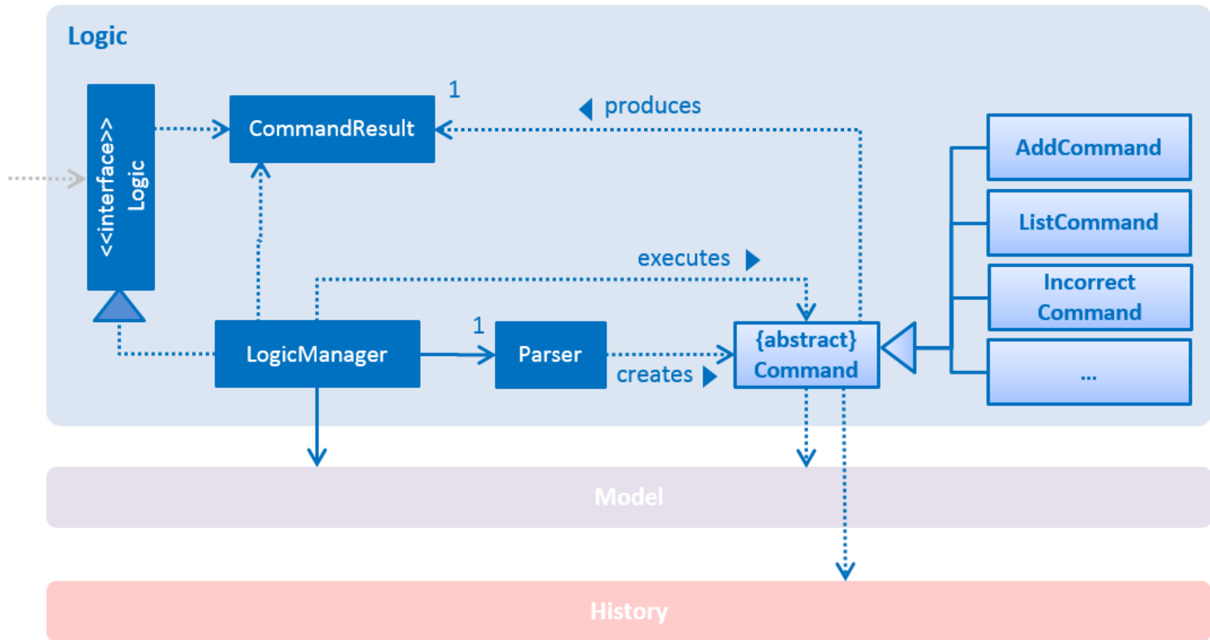

Figure 2: Logic Class Diagram

The Logic component above defines it's API in the Logic.java interface and exposes its functionality using the LogicManager.java class.

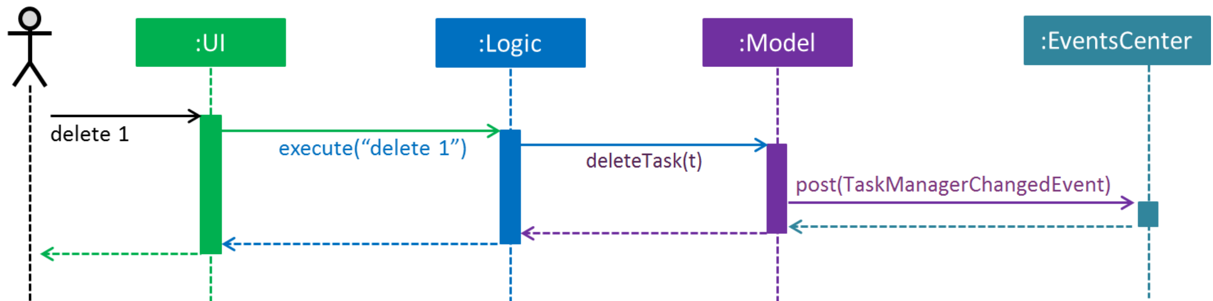

Figure 3: Sequence Diagram for Delete Task

The Sequence Diagram above 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 is changed, instead of asking the Storage to save the updates to the hard disk.

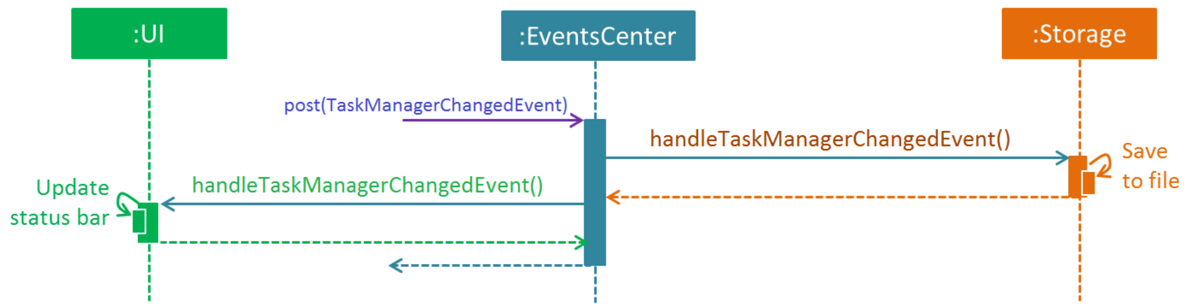

Figure 4: Sequence Diagram for Delete Task Event Handling

The diagram above shows how the EventsCenter reacts to that event, which eventually results in the updates being saved to the hard disk. The status bar of the UI is 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

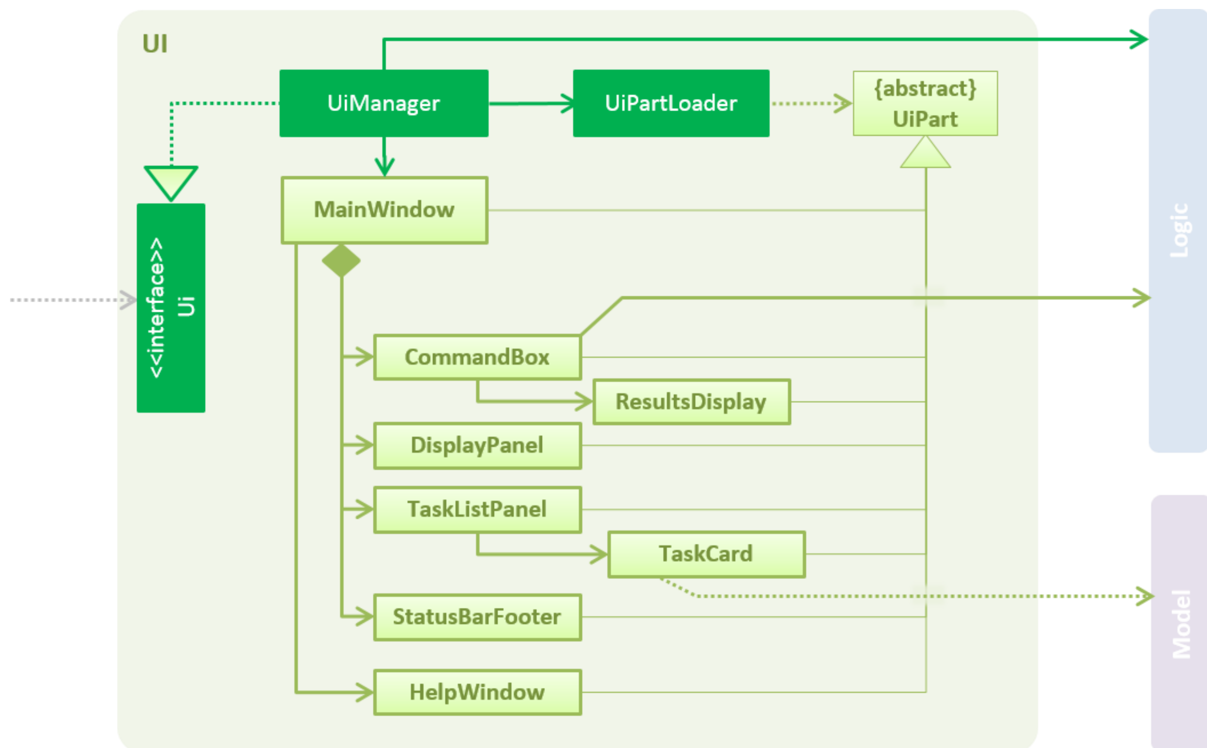

Figure 5: UI Class Diagram

The diagram above gives an overview of how the UI component is implemented.

API : 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 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 changes.
- Responds to events raised from various parts of the Application and updates the UI accordingly.
- Uses Agenda API from JFXtras to display calendar view with task events.

Logic Component

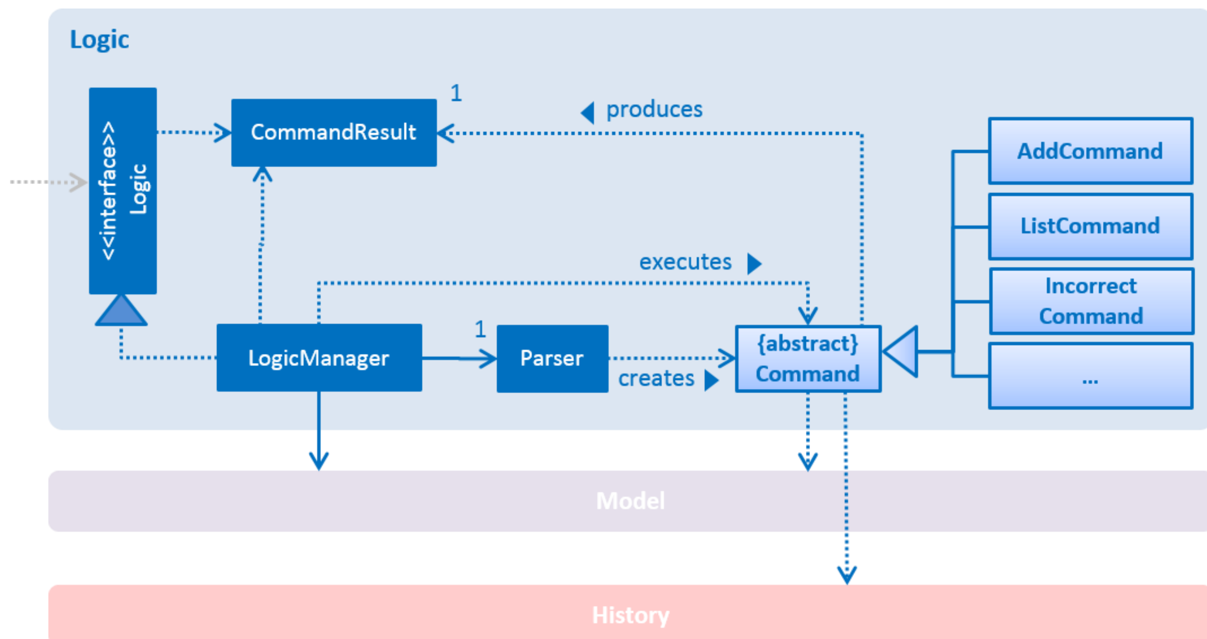

Figure 6: Logic Class Diagram

The diagram above gives an overview of how the Logic component is implemented.

API : Logic.java

The Logic component,

- Uses the Parser class to parse the user command: results in a Command object which is executed by the LogicManager.
- Affects the Model (e.g. adding a task) and/or raise events.
- Executes the necessary command and the result is encapsulated as a CommandResult to be passed back to the UI.

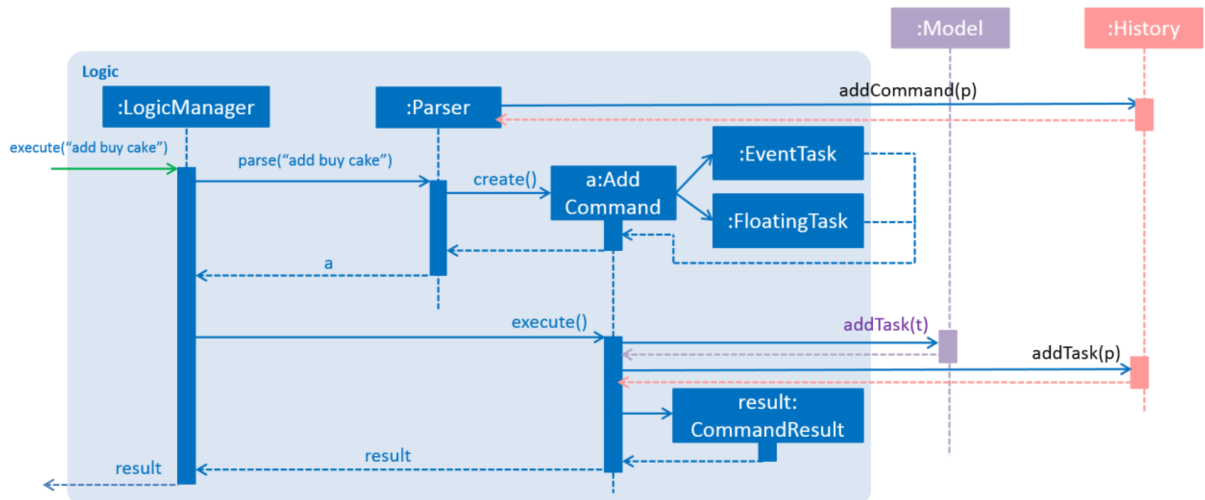

Figure 7: Add Task Sequence Diagram for Logic

The diagram above shows the Sequence Diagram for interactions within the Logic component for the execute("add buy cake") API call.

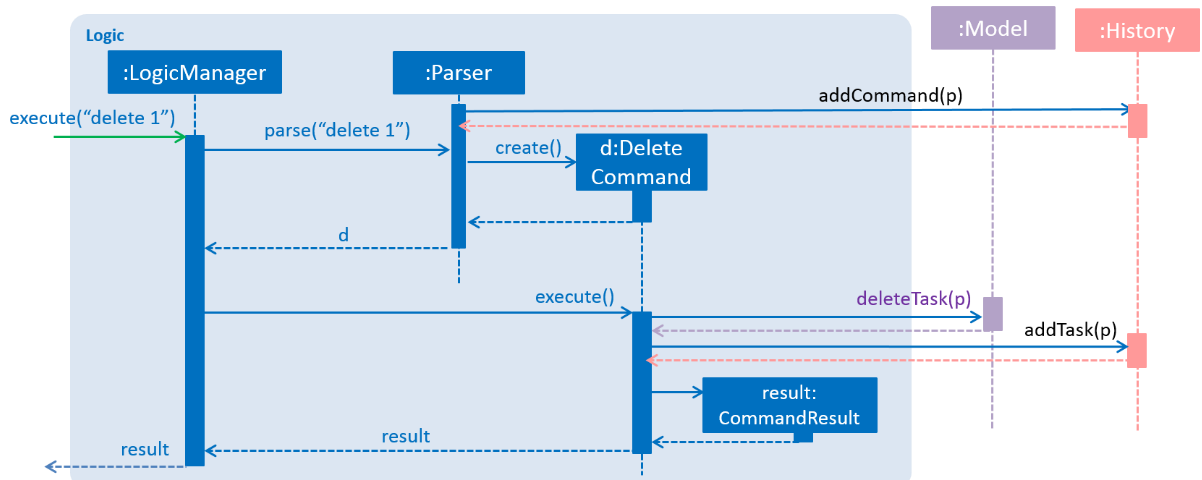

Figure 8: Delete Task Sequence Diagram for Logic

The diagram above shows the Sequence Diagram for interactions within the Logic component for the execute("delete 1") API call.

Model Component

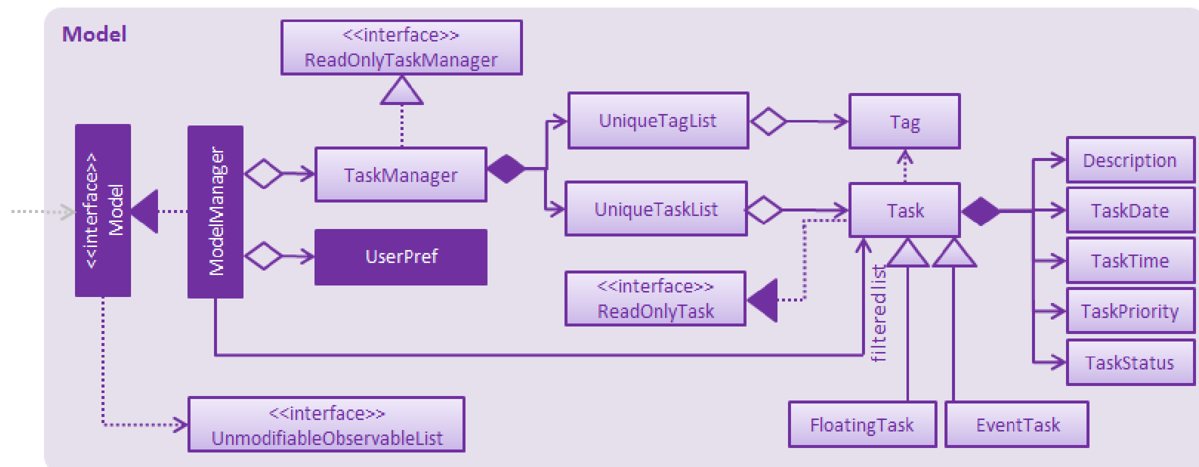

Figure 9: Model Class Diagram

The diagram above gives an overview of how the Model component is implemented.

API : [Model.java]

The Model component,

- 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

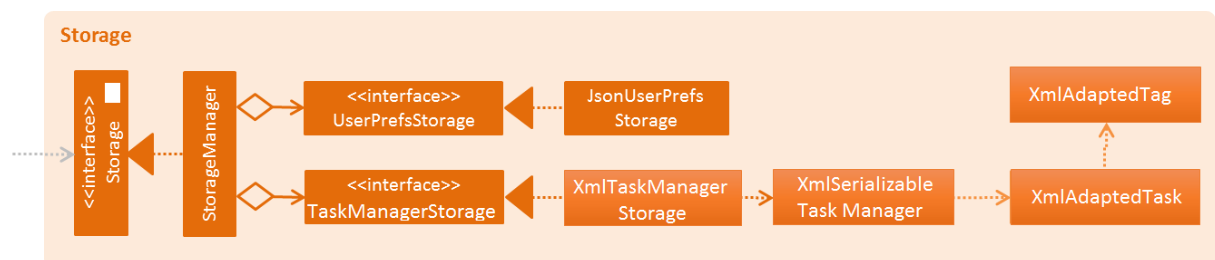

Figure 10: Storage Class Diagram

The diagram above gives an overview of how the Storage component is implemented.

API : [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.

Storage Component

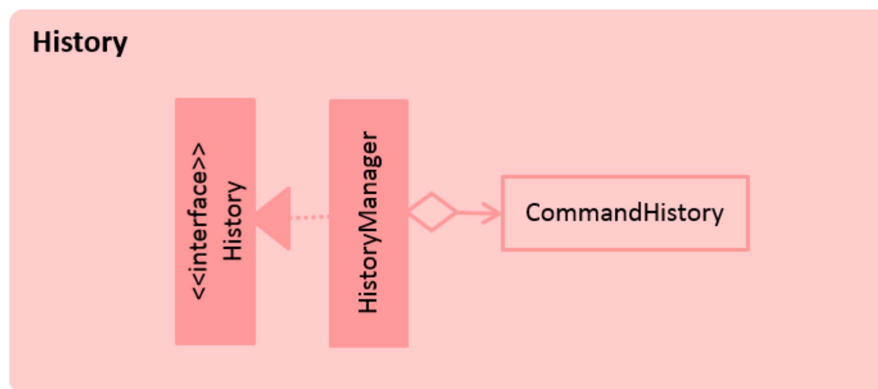

Figure 11: History Class Diagram

The diagram above gives an overview of how the History component is implemented.

API : History.java
The History component,

- stores the commands that UndoCommand can execute (add/delete/done/undone/edit)
- exposes list of command input strings for UI display
- updates list of command history every time a command is executed

Common Classes

Classes used by multiple components are in the seedu.taskmanager.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 : Critical problem detected which may possibly cause the termination of the application
- WARNING : Program can continue, but with caution
- INFO : Information showing the noteworthy actions by the Application
- 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 Application name, logging level) through the configuration file (default: config.json). To reset properties in the configuration file, delete config.json and run Taskell again.

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.

- 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 Application 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.taskell.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.taskell.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.taskell.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.

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 repository with the version number. e.g. v0.1
3. Create a new release using GitHub and upload the JAR file your created.

Managing Dependencies

A project often depends on third-party libraries. For example, Taskell depends on the Jackson library 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 repository (this bloats the repository 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 user guide	refer to the different commands when I forget how to use the application.
* * *	user	add a task	take note of all my tasks.
* * *	user	delete a task	remove task that I no longer need.
* * *	user	find a task by its description	locate details of tasks without having to go through the entire list.
* * *	user	categorize my tasks	group and view tasks of similar type.
* * *	user	view all the tasks, sorted by day, month	plan my schedule.
* * *	user	edit task	make changes to the task created.
* * *	user	have a start and end time for a event	take note of the duration of the event
* * *	user	set deadlines for a task	remember the task is due.
* * *	user	undo my previous action	correct any mistakes made
* * *	user	mark a task as done	focus on the uncompleted tasks.
* * *	user	have flexible command format	have various options to execute a command
* * *	user	specify a folder with cloud syncing service as the storage location	easily access my task manager from different computers.
* * *	user	I want to see a list of completed tasks	view all the tasks that I have done
*	user	delete tasks based on a certain index	delete a few tasks instead of one
*	user	set some of my task recursively	schedule them on a daily/weekly/monthly basis
*	user	be able to block multiple timeslots, and release the timeslots when timing is confirmed	schedule in events which have uncertain timings more efficiently.
*	user	sort tasks by priority	view the most important tasks.
*	user	edit my notification time period	customise if I wanted to be reminded earlier or later.
*	user	use the history command	saves time typing repeated commands
*	user	view the task in either calendar form or list form	switch between the two display format

Appendix B : Use Cases

Use case: Add task

MSS

1. User requests to add tasks either with or without deadline
2. Taskell adds the task

Use case ends

Extensions

2a. The user did not follow the given format to add a task or deadline

> 2a1. Taskell shows the help message
> Use case resumes at step 1

Use case: Delete task

MSS

1. User requests to list tasks
2. Taskell shows a list of uncompleted tasks
3. User requests to delete a specific task in the list
4. Taskell deletes the task

Use case ends

Extensions

2a. The list is empty

3a. The given index is invalid

> 3a1. Taskell shows an error message
> Use case resumes at step 2

Use case: Done task

MSS

1. User requests to list tasks
2. Taskell shows a list of uncompleted tasks
3. User requests to mark a specific task in the list as completed
4. Taskell marks the task as completed

Use case ends

Extensions

2a. The list is empty

2b. User tries to mark a completed task as completed

3a. The given index is invalid

> 3a1, 2b1, 2a1. Taskell shows an error message
> Use case resumes at step 2

Use case: Undone task

MSS

1. User requests to list tasks
2. Taskell shows a list of completed tasks
3. User requests to mark a specific task in the list as uncompleted
4. Taskell marks the task as uncompleted.

Use case ends

Extensions

2a. The list is empty
3a. The given index is invalid
3b. User tries to mark an uncompleted task as uncompleted

> 2a1, 3a1, 3b1. Taskell shows an error message
> Use case resumes at step 2

Use case: Help task

MSS

1. User requests to view the different command
2. User enters "help"
3. User guide is displayed.

Use case ends

Extensions

2a. The user types "help" incorrectly

> 2a1. Taskell displays the error message

Use case: Find task

MSS

1. User requests to find tasks with specific keywords
2. Taskell displays the task with all the matching keywords

Use case ends

Extensions

1a. No keyword is given

> 1a1. Taskell shows an error message

Use case: Edit task

MSS

1. User requests to list tasks
2. Taskell shows a list of tasks
3. User requests to edit either the description, date, time or priority of a task
4. Taskell edits the respective field
5. Taskell displays both the original and updated version of the task.

Use case ends

Extensions

2a. The list is empty

3a. The given index is invalid

3b. The user did not key in the new field of the task

3c. The user did not key in a valid parameter

> 3a1, 3b1 and 3c1. Taskell shows an error message
> Use case resumes at step 2

Use case: Undo task

MSS

1. User enters a command
2. Taskell executes it
3. User requests to view undo commands history
4. Taskell requests to undo command at specific index
5. Taskell reverts the command

Use case ends

Extensions

3a. The user did not enter any previous command

> 3a1. Taskell shows an error message

4a. The user enters invalid index

> 4a1. Taskell shows error essage indicating index is invalid

Use case: List task

MSS

1. User requests to list either all tasks, incomplete tasks, completed tasks, task with specific start date or task with specific priority.
2. Taskell shows a list of tasks

Use case ends

Extensions

2a. The list is empty

> 2a1. Taskell shows an error message
> Use case resumes at step 2

Use case: View calendar for the week

MSS

1. User requests to view calendar
2. Taskell displays calendar

Use case ends

Extensions

NIL

Use case: Store data in cloud syncing folder

MSS

1. User requests to save all tasks
2. Taskell saves all tasks in the requested folder

Use case ends

Extensions

1a. User gives invalid file path (contains illegal symbols not allowed in file names)

> 1a1. Taskell shows an error message and still saves in previous old location.

2a. Data cannot be written to the requested folder (invalid directory or access prohibited)

> 2a1. Taskell shows an error message and still saves in previous old location.

Use case: Clear task

MSS

1. User requests to clear all tasks
2. Taskell shows pop-up to ask for confirmation
3. User confirms
4. Taskell deletes all the tasks

Use case ends

Extensions

2a. User cancels request

> 2a1. Taskell does not clear all tasks

Use case: Exit task

MSS

1. User requests to exit Taskell
2. Taskell window closes

Use case ends

Extensions

NIL

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 favour DOS style commands over Unix-style commands.
5. Each command executed under 1 second.

Appendix D : Glossary

Mainstream OS

Windows, Linux, Unix, OS-X

Floating Tasks

Tasks with no deadline

Overdue Tasks

Tasks with the deadline of the task elapsed

Appendix E : Product Survey

Task Managers	WunderList	Remember the Milk	Google Calendar	Any.do	Taskell
CRUD	Available	Available	Available	Available	Available
Undo	Not available	Available	Not Available	Available	Available
Sync across multiple platform	Available	Available	Available	Available	Available
Internet required	Required	Required	Required	Required	Not required
Calendar	Not available	Available	Available	Available	Available
Other features	Attach different files and picture inside the task	Handle some natural languages	Able to add public holiday	Do a daily review at the start/end of day	Support for Recurring Tasks