

# Developer Guide

---

- [Introduction](#)
- [Setting Up](#)
- [Design](#)
  - [Architecture](#)
  - [UI](#)
  - [Logic](#)
  - [Model](#)
  - [Storage](#)
  - [Common](#)
- [Implementation](#)
  - [Logging](#)
  - [Configuration](#)
- [Testing](#)
- [Dev Ops](#)
  - [Build Automation](#)
  - [Continuous Integration](#)
  - [Making a Release](#)
- [Appendix A: User Stories](#)
- [Appendix B: Use Cases](#)
- [Appendix C: Non Functional Requirements](#)
- [Appendix D: Glossary](#)
- [Appendix E : Product Survey](#)

## Introduction

---

*DearJim* is a revolutionary task manager designed to help you organise your tasks that is simple and easy to use. *DearJim* is a Java desktop application that has a GUI, and the main mode of input in *DearJim* is through keyboard commands.

This guide describes the design and implementation of *DearJim*. It will help you understand how *DearJim* works and how you can further contribute to its development. We have organised this guide in a top-down manner so that you can understand the big picture before moving on to the more detailed sections.

---

# Setting up

---

## Prerequisites

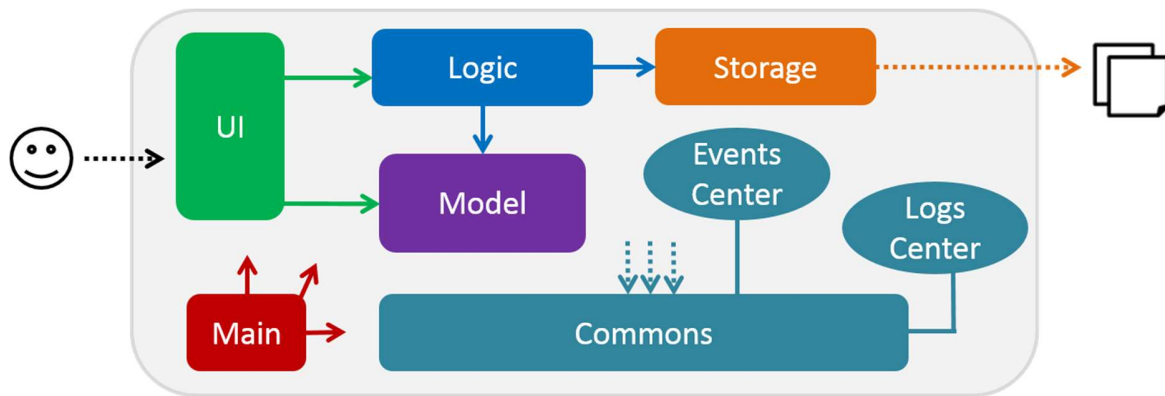
1. **JDK 1.8.0\_60** or later  
Having any Java 8 version is not enough.  
This app will not work with earlier versions of Java 8.
2. **Eclipse IDE**
3. **e(fx)clipse** plugin for Eclipse (Do the steps 2 onwards given in [this page](#))
4. **Buildship Gradle Integration** plugin from the Eclipse Marketplace

## Importing the project into Eclipse

1. Fork this 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.

# Design

## Architecture



*The Architecture Diagram of DearJim*

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`. 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](#)) 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.

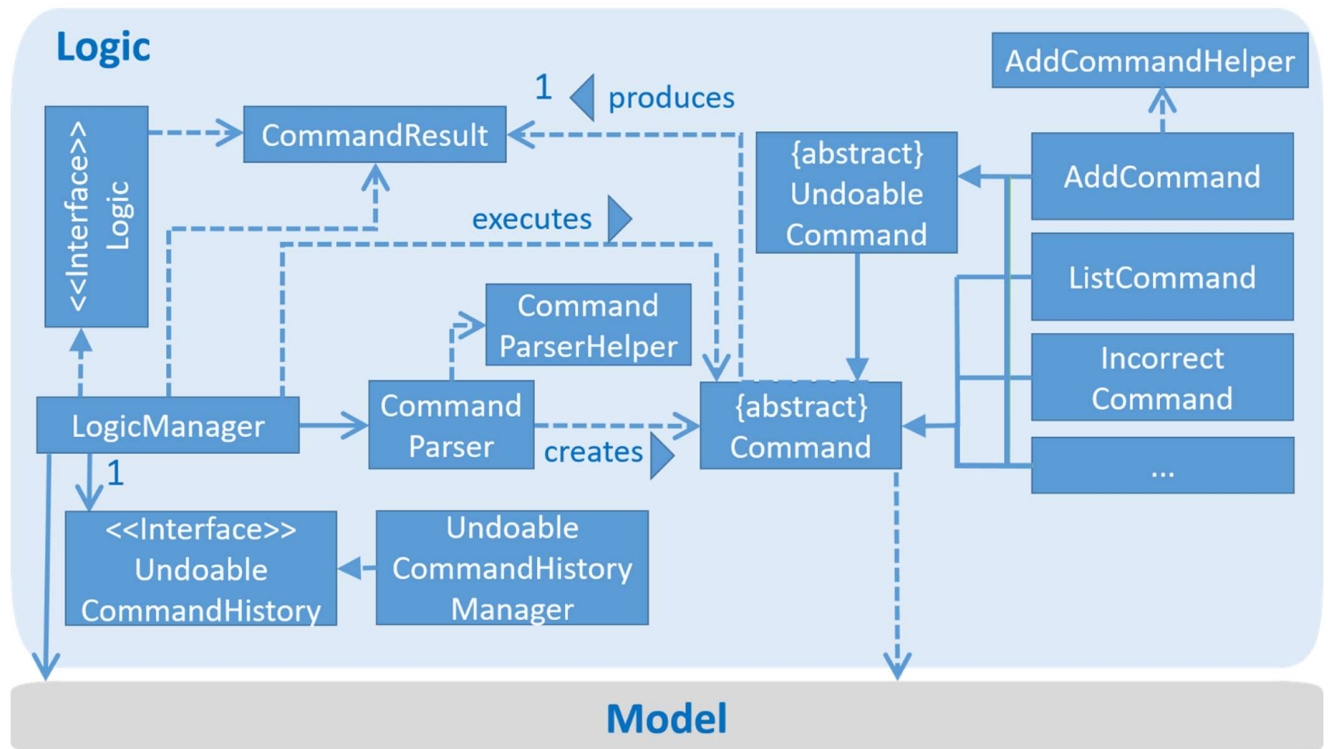
The rest of the App consists four components.

- `UI` : The UI of the 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

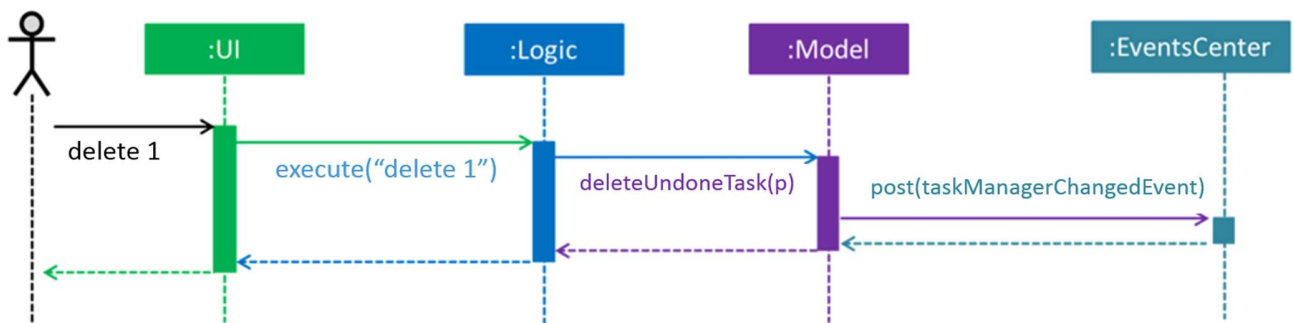
- Defines its *API* in an interface with the same name as the Component.
- Exposes its functionality using a `{Component Name}Manager` class.

For example, the Logic component (see the class diagram given below) defines its API in the `Logic.java` interface and exposes its functionality using the `LogicManager.java` class.



The class diagram for the Logic Component of DearJim

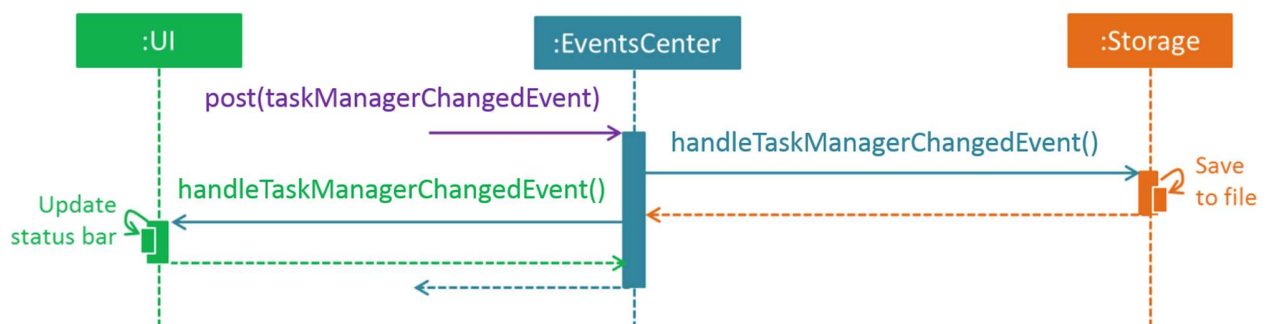
The *Sequence Diagram* below shows how the components interact for the scenario where the user issues the command delete 1.



*The sequence diagram for the scenario 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.

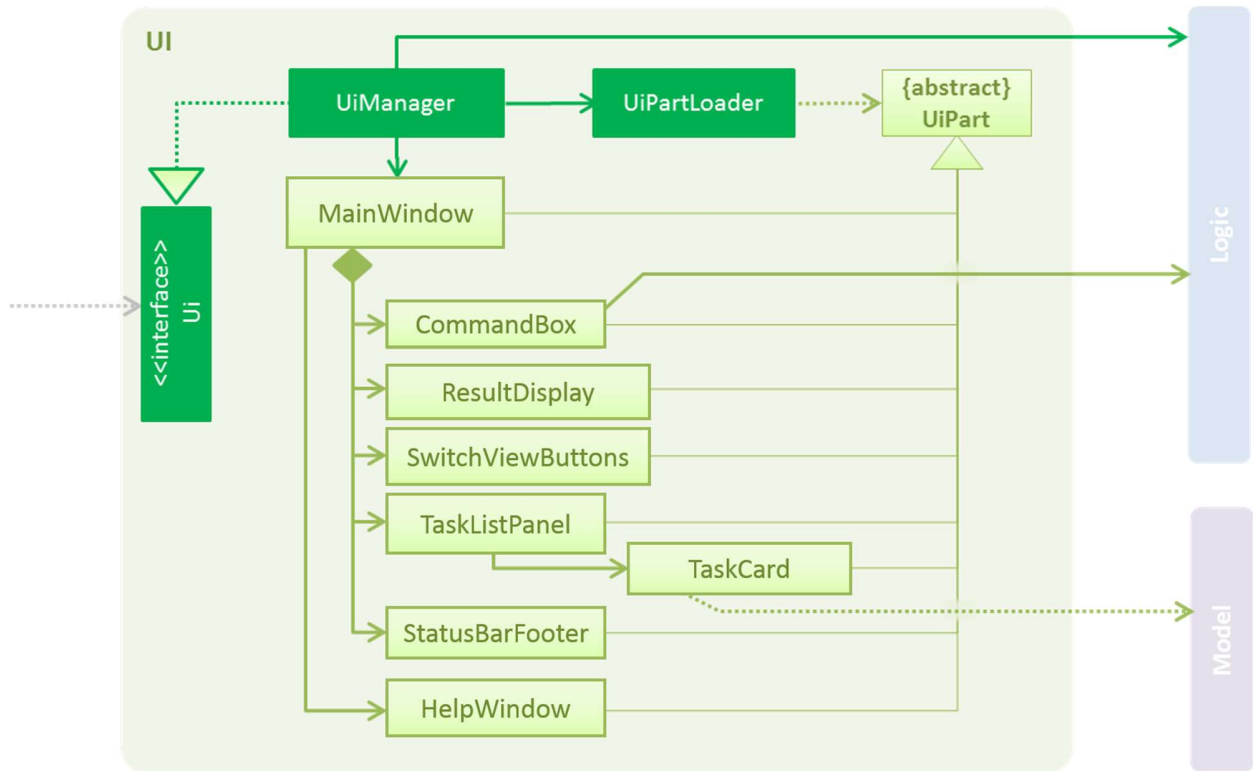


*The sequence diagram showing the TaskManagerChangedEvent and effects on Storage and UI*

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



The class diagram for the UI component of DearJim

### API: `Ui.java`

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

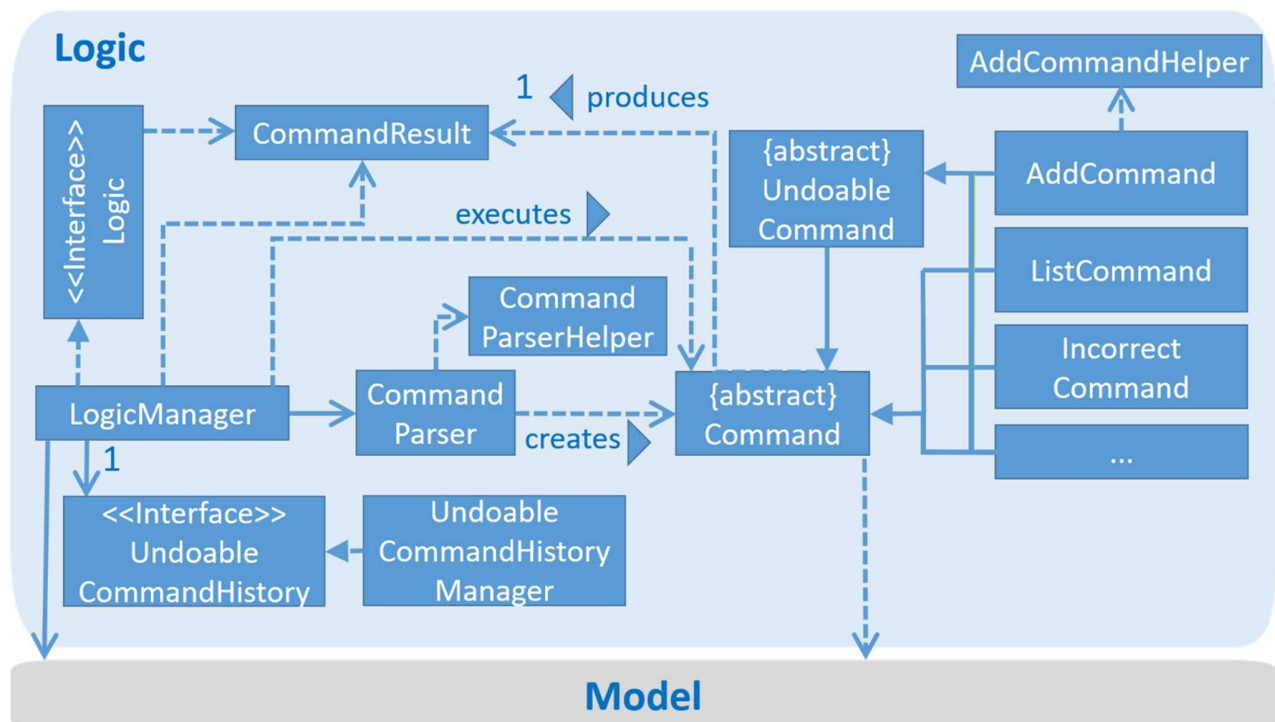
The UI component uses JavaFx UI framework. The layouts of these UI parts are defined in matching `.fxml` files that are in the `src/main/resources/view` folder.

For example, the layout of the `MainWindow` is specified in `MainWindow.fxml`

The UI component,

- Executes user commands and updates the user input history (to navigate the previous and next inputs using `Up` and `Down` arrow keys) using the `Logic` component.
- Binds itself to some data in the `Model` so that the UI can automatically update itself when data in the `Model` changes.
- Responds to events raised from various parts of the App and updates the UI accordingly.

## Logic component

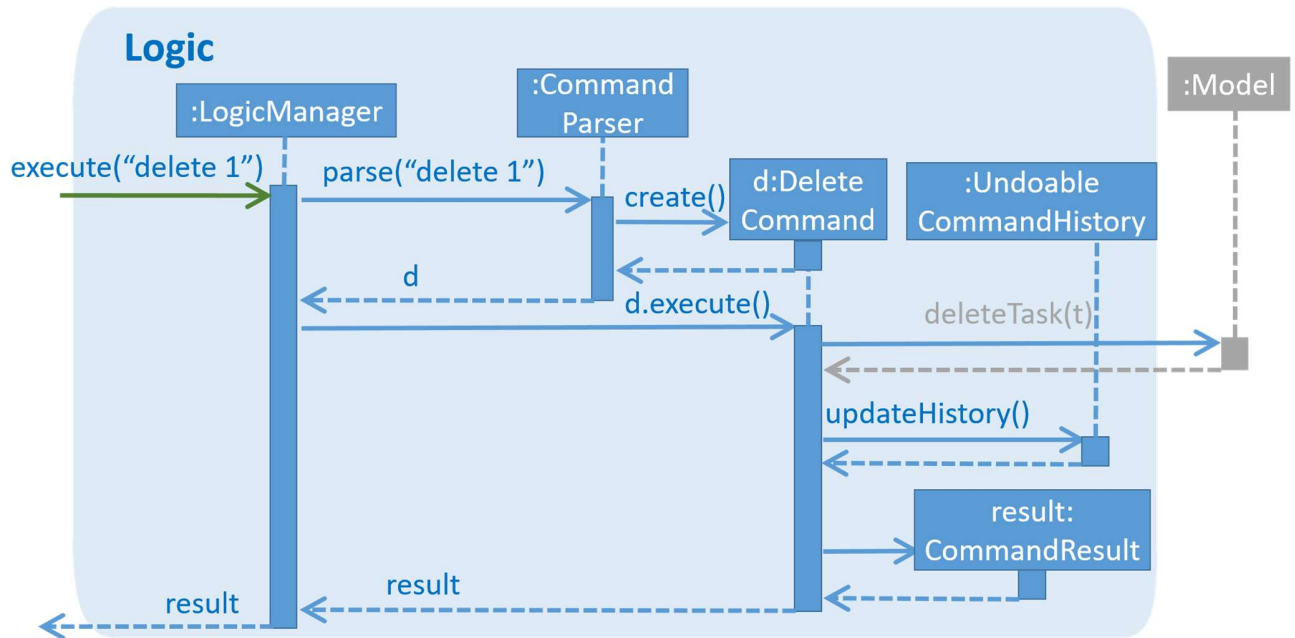


The class diagram for the Logic component of DearJim

API : `Logic.java`

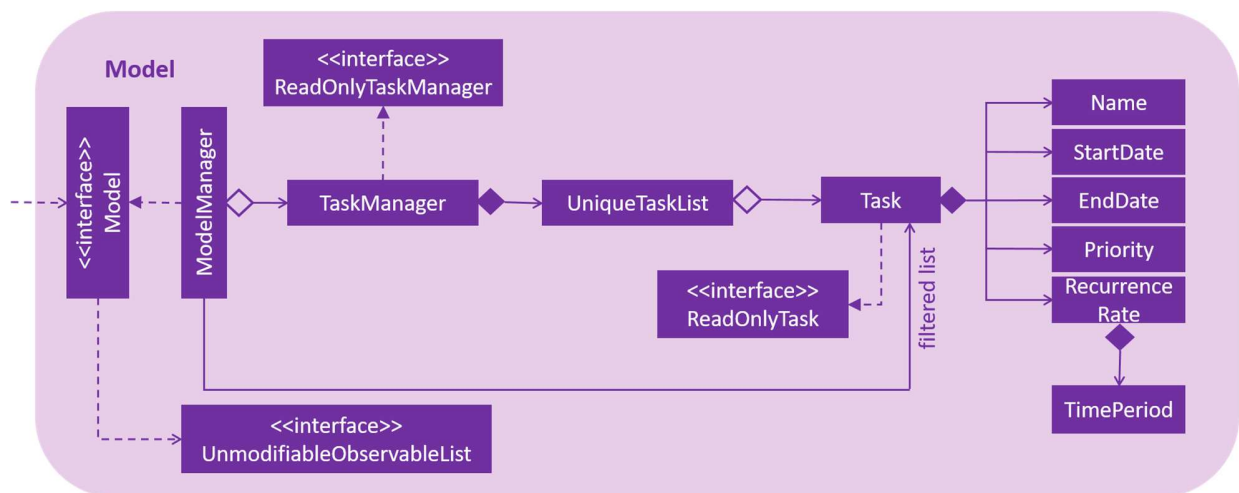
1. Logic uses the `CommandParser` class to parse the user command.
2. This results in a `Command` object which is executed by the `LogicManager`.
3. The command execution can affect the Model (e.g. adding a task) and/or raise events.
4. The command execution can update the History if command executed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

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



The sequence diagram for the interactions within the Logic component for execute("delete 1")

## Model component



The class diagram for the Model component of DearJim

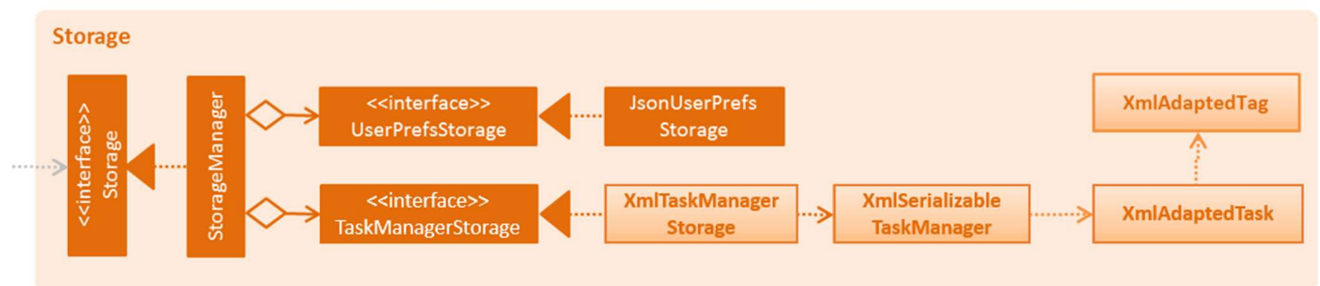
API: Model.java



The Model,

- 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



*The class diagram for the Storage component of DearJim*

**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.

## Common classes

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

Examples of these classes include the `StringUtil` class, which has the `containsIgnoreCase(String, String)` method, and `CollectionUtil` class, which has the `isAnyNull(Object...)` method.

## 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` : Application can continue running, but minor errors may occur
- `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](#).

- 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 `guittests` package.
2. **Non-GUI Tests** - These are tests not involving the GUI. They include,
  - i. *Unit tests* targeting the lowest level methods/classes.  
e.g. `seedu.taskmanager.commons.UrlUtilTest`
  - ii. *Integration tests* that are checking the integration of multiple code units (those code units are assumed to be working).  
e.g. `seedu.taskmanager.storage.StorageManagerTest`
  - iii. Hybrids of unit and integration tests. These tests are checking multiple code units as well as how they are connected together.  
e.g. `seedu.taskmanager.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 repo 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, DearJim 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 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 - \* \* \* Medium - \* \* Low - \*

Priority	As a ...	I want to ...	So that I can...
* * *	new user	see instructions on how to use the app	refer to instructions when I forget how to use the App
* * *	user	add tasks to the todo list	be reminded of what to do
* * *	user	add tasks with deadline	remember by what time I have to complete a task
* * *	user	add tasks with timeframe	know what I have to attend an event
* * *	user	view upcoming tasks in chronological order	check what I need to do / have to do soon
* * *	user	search for details on a task or event	view what needs to be done
* * *	user	delete a task that has been completed	completely remove it from my task manager
* * *	user	mark a task as completed upon completion	keep an archive of tasks that I have completed
* * *	user	view tasks that I have completed	recall what I have completed
* * *	user	undo my actions	reverse any mistake that I have made
* * *	user	redo my undo	correct any accidental undo
* * *	user	edit tasks	keep them up to date
* * *	user	store my tasks in different locations	share the tasks with multiple devices
* *	user	sync my task with other computers	keep track of what to do anywhere
* *	user	enter synonyms for the commands	use natural language to accomplish what I want
*	user	tag tasks	classify them and search for them according to these tags

Priority	As a ...	I want to ...	So that I can...
*	user	call up the todo list with a simple keystroke	start the application anytime during my workflow

---

## Appendix B : Use Cases

---

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

### Use case: UC01 - Add a task

#### MSS

1. User enters an add command, specifying details of the task to be added
2. TaskManager parses the add command, and adds the task to the current task list
3. TaskManager saves the current task list to storage and updates the GUI to display the updated list with the newly added task

Use case ends

#### Extensions

1a. User enters a task name that needs to be escaped as it contains values that CommandParser is unable to parse correctly.

1a1. TaskManager's instant parsing feature reflects to the user that his / her input is being parsed into the wrong field

1a2. User uses the double inverted commas to escape the task name

Use case resumes at step 2

2a. User enters an input that does not follow the add command format

2a1. TaskManager displays an error message on the GUI, informing the user of the correct format for the add command and an example add command

Use case resumes at step 1

2b. User is currently at done list view

2b1. TaskManger displays an error message on the GUI, informing the user that he / she is unable to perform the add command in done list view, and prompts the user to switch to undone list view instead to perform the add command

Use case resumes at step 1

3a. User identifies a mistake in the details of the task added

3a1. User edits the task details (UC03)

Use case ends

### **Use case: UC02 - List all undone tasks**

#### **MSS**

1. User enters the `list` command
2. TaskManager parses the `list` command
3. TaskManager removes any filters for the task list and updates the GUI to display the entire list of undone tasks

Use case ends

#### **Extensions**

2a. User enters an input that does not follow the `list` command format

2a1. TaskManager displays an error message on the GUI, informing the user of the correct format for the `list` command and an example `list` command

Use case resumes at step 1

2b. The list is empty

Use case ends

### **Use case: UC03 - Edit an undone task**

#### **MSS**

1. User requests to list undone tasks (UC02)
2. TaskManager shows the list of all undone tasks
3. User enters the `edit` command, specifying the INDEX of the task in the list to be edited, the fields to be edited and their new values
4. TaskManager parses the `edit` command and looks for the task in the list
5. TaskManager edits the requested fields on the specified task according to the command entered
6. TaskManager updates the GUI to display the new list of undone tasks and highlight the newly edited task

Use case ends



## Extensions

2a. The list is empty

Use case ends

3a. Index is not given

3a1. TaskManager displays an error message on the GUI, informing the user of the correct format for the `edit` command and an example `edit` command

Use case resumes at step 3

3b. User enters a task name that needs to be escaped as it contains values that `CommandParser` is unable to parse rightly.

3b1. TaskManager's instant parsing feature reflects to the user that his / her input is being parsed into the wrong field

3b2. User uses the double inverted commas to escape the task name

Use case resumes at step 3

4a. The given index is invalid

4a1. TaskManager displays an error message on the GUI, informing the user that the given index is invalid and thus cannot edit any task

Use case resumes at step 3

4b. User enters an end date that occurs before the start date.

4b1. TaskManager displays an error message on the GUI, informing the user that the end date must occur after the start date

Use case resumes at step 3

4c. User enters an input that does not follow the `edit` command format

4c1. TaskManager displays an error message on the GUI, informing the user of the correct format for the `edit` command and an example `edit` command

Use case resumes at step 3

#### **Use case: UC04 - Delete an undone task**

##### **MSS**

1. User requests to list undone tasks (UC02)
2. TaskManager shows the list of all undone tasks
3. User enters the delete command, specifying the INDEX of the task in the list to be deleted
4. TaskManager parses the delete command and looks for the task in the list
5. TaskManager deletes the task from the list
6. TaskManager updates the GUI to display the new list of undone tasks

Use case ends

##### **Extensions**

2a. The list is empty

Use case ends

4a. The given index is invalid

4a1. TaskManager displays an error message on the GUI, informing the user that the given index is invalid and thus cannot delete any task

Use case resumes at step 3

4b. User enters an input that does not follow the delete command format

4b1. TaskManager displays an error message on the GUI, informing the user of the correct format for the delete command and an example delete command

Use case resumes at step 3

#### **Use case: UC05 - Undo a previous command**

##### **MSS**

1. User enters an undo command
2. TaskManager parses the undo command
3. TaskManager attempts to identify the latest stored undoable command, reversing the action of that command
4. TaskManager saves the modified task list to storage and updates the GUI to inform the user of the changes

Use case ends.

## **Extensions**

1a. User enters an undo command, followed by some arguments

1a1. TaskManager parses the undo command, ignoring the arguments that follow  
Use case resumes at step 3

3a. There is no previous undoable command

3a1. TaskManager indicates that there is nothing to undo.  
Use case ends.

4a. User wants to reverse the undo command

4a1. User enters the redo command (UC06)  
Use case ends.

## **Use case: UC06 - Redo a command that was undone**

### **MSS**

1. User enters a undo command successfully (UC05)
2. User enters a redo command
3. TaskManager parses the redo command
4. TaskManager attempts to identify the latest stored command that was undone by an undo, redoing the effects of that command
5. TaskManager saves the modified task list to storage and updates the GUI to inform the user of the changes

Use case ends.

## **Extensions**

1a. User enters a non-undoable command

1a1. TaskManager handles the command  
Use case resumes at step 2

1b. User enters an undoable command

1b1. TaskManager handles the command and clears the history of commands to redo

1b2. User enters a redo command

1b3. TaskManager parses the redo command and indicates that there is nothing to redo  
*Use case ends.*

2a. User enters a redo command, followed by some arguments

1a1. TaskManager parses the redo command, ignoring the arguments that follow

*Use case resumes at step 3*

## 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 10000 tasks.
3. Should come with automated unit tests and open source code.
4. Should favor DOS style commands over Unix-style commands.
5. Should have less than 2 seconds processing time for each command.
6. Should be easy to learn and use.
7. Should be available without Internet connection.
8. Should be scalable and maintainable.

## Appendix D : Glossary

---

### **Mainstream OS**

Windows, Linux, Unix, OS-X

### **Scalable**

Able to work well as number of tasks grows

### **Synonyms**

Alternative names for a single command

### **Maintainable**

Code that is readable and easy to contribute towards

## Appendix E : Product Survey

Product	Strength	Weaknesses
Wunderlist	<ol style="list-style-type: none"> <li>1. Beautiful background</li> <li>2. Cloud sync</li> <li>3. Able to create folders to group similar tasks</li> <li>4. Able to add tags to tasks to filter them</li> </ol>	<ol style="list-style-type: none"> <li>1. No start date or repeat options for tasks</li> <li>2. No options for subtasks</li> </ol>
Todo.txt	<ol style="list-style-type: none"> <li>1. Works on many platforms, can be accessed on devices that support Dropbox</li> <li>2. Easily editable format, can be edited in plain text and then displayed with neat styles</li> <li>3. Can edit with any text editor</li> <li>4. Easy syncing - can sync through Dropbox</li> <li>5. Command line support - can edit using command line by a supplied bash script</li> </ol>	<ol style="list-style-type: none"> <li>1. No support for recurring tasks</li> <li>2. No options for subtasks</li> <li>3. Only supports Dropbox, not flexible</li> </ol>
Google Calendar	<ol style="list-style-type: none"> <li>1. Can be synced to mobile devices</li> <li>2. Alerts via notifications on phones</li> <li>3. Switches between views easily</li> <li>4. Minimalistic interface</li> </ol>	<ol style="list-style-type: none"> <li>1. Requires an Internet connection to be used</li> <li>2. Cannot be brought up with a keyboard shortcut</li> </ol>
Remember the milk	<ol style="list-style-type: none"> <li>1. Able to support email, text, IM, Twitter, and mobile notifications</li> <li>2. Able to share lists and tasks with others</li> <li>3. Synchronises across on all devices</li> <li>4. Organize with priorities, due dates, repeats, lists, tags</li> <li>5. Search tasks and notes, and save favourite searches</li> <li>6. Integrates with Gmail, Google Calendar, Twitter, Evernote, and more</li> </ol>	<ol style="list-style-type: none"> <li>1. Free version lacks features: E.g. splitting into subtasks</li> <li>2. Lack keyboard shortcuts</li> </ol>

**Summary:** We observed that these products have very good features, but we realised that none of these products have the specific combination of features that caters to our target audience. Therefore, we are incorporating some of the good features such as minimalistic interface and ability to sync with multiple computers while designing DearJim carefully to avoid the pitfalls found in these products, to make a targeted product for our intended audience.