**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** or later

**1.8.0\_60**

Having any Java 8 version is not enough.

This app will not work with earlier versions of Java 8.

1. **Eclipse** IDE
2. **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))
3. **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

1. Click Browse , then locate the project's directory
2. Click

Finish

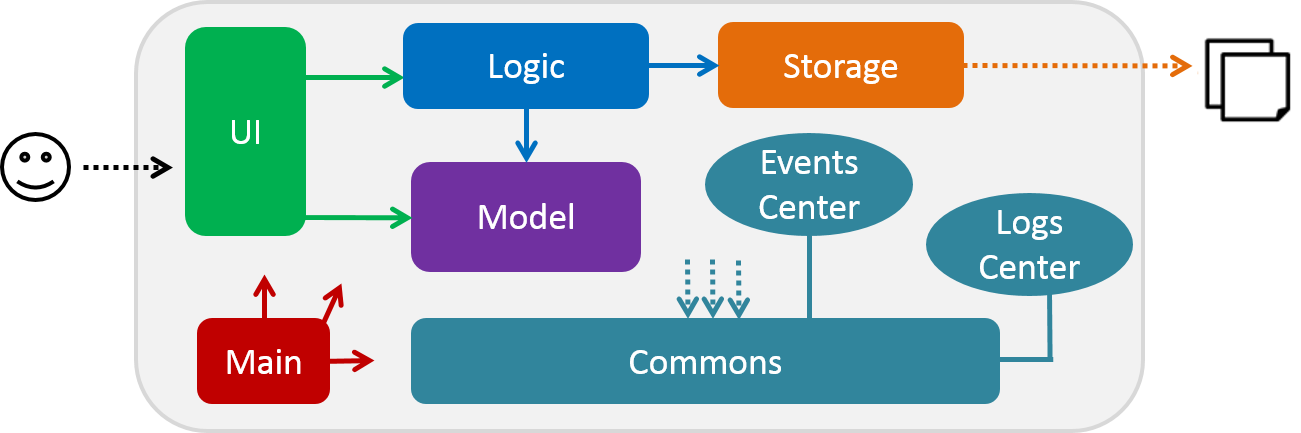
If you are asked whether to 'keep' or 'overwrite' config files, choose to 'keep'.

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.

has only one class called . It is responsible for,

Main

MainApp

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.

represents a collection of classes used by multiple other components. Two of those classes play important roles at the architecture level.

**Commons**

: This class (written using [Google's Event Bus library](https://github.com/google/guava/wiki/EventBusExplained)) is used by components to communicate with other

EventsCentre

components using events (i.e. a form of *Event Driven* design)

: Used by many classes to write log messages to the App's log file.

LogsCenter

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.

Logic

Logic.java

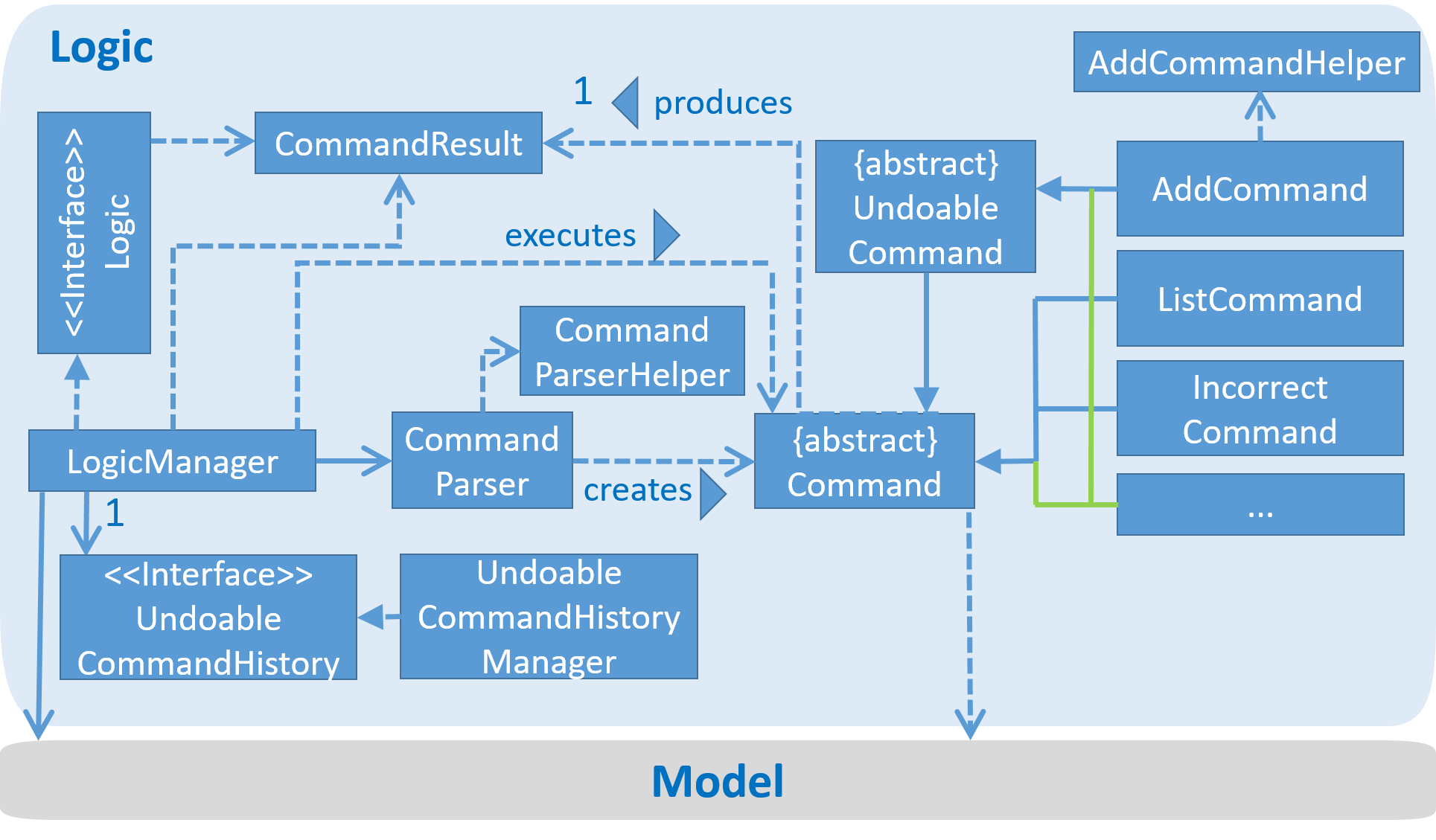
For example, the

component (see the class diagram given below) defines its API in the

interface and

exposes its functionality using the class.

LogicManager.java

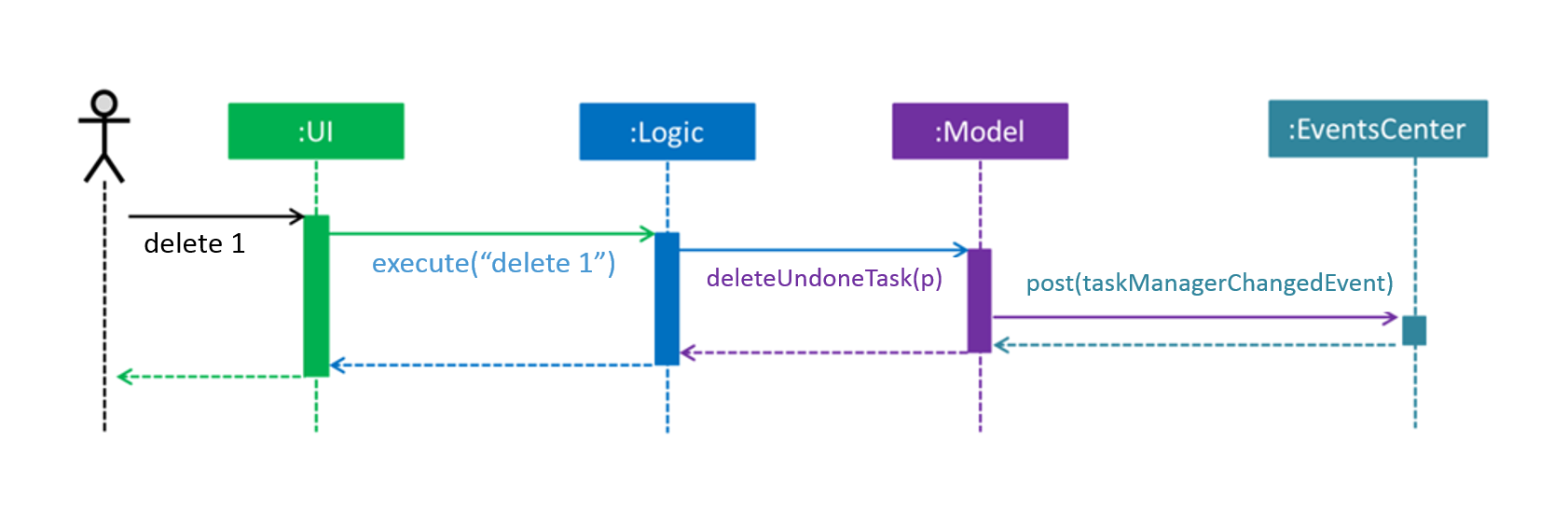


*The class diagram for the Logic Component of DearJim*

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

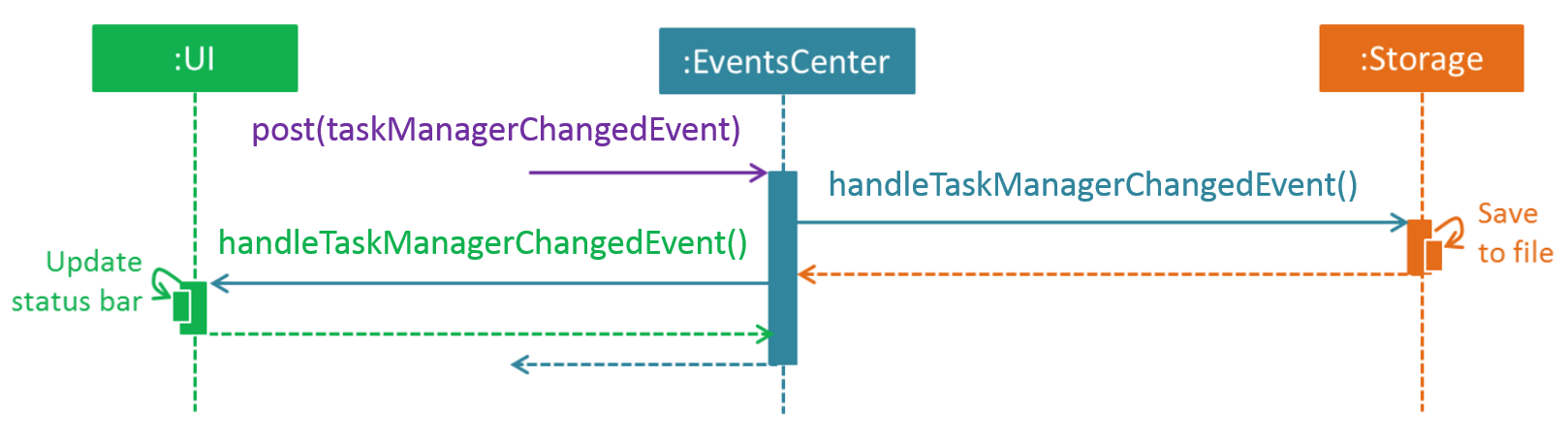
asking the Storage to save the updates to the hard disk.

when the Task Manager data are changed, instead of

The diagram below shows how the reacts to that event, which eventually results in the updates being saved to the

EventsCenter

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*

*and*

Note how the event is propagated through the

*Storage*



*UI*

to the

and

without

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.

EventsCenter

Storage

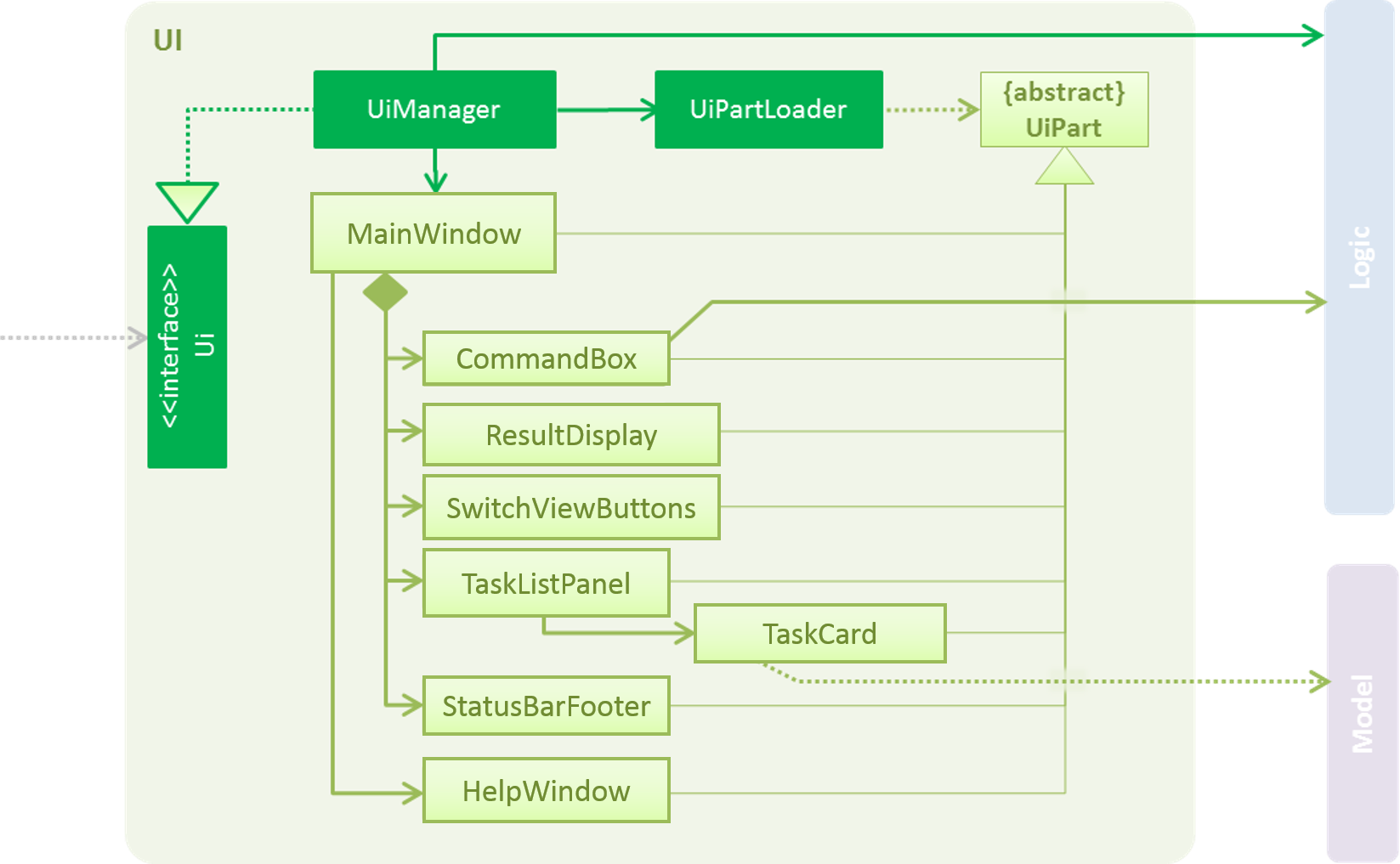


UI

Model

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 .

.fxml

vaFx UI framework. The layouts of these UI parts are defined in matching folder.



The UI component uses Ja

src/main/resources/view

files that are in the

For example, the layout of the is specified in

MainWindow

MainWindow.fxml

The component,



UI

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

change.

Model

Binds itself to some data in the InputHistory so that it can fetch the user input history, to allow the user to navigate his



Up

Down

previous or next inputs using the

and

Arrow keys, updating the UI in doing so.

Responds to events raised from various parts of the App and updates the UI accordingly.

## Logic component

## C:\Users\fishe\AppData\Local\Microsoft\Windows\INetCacheContent.Word\NewLogicDiagram.png

*The class diagram for the Logic component of DearJim*

#### API :

Logic.java

1. uses the CommandParser class to parse the user command.

Logic

1. This results in a Command object which is executed by the .

LogicManager

1. The command execution can affect the Model (e.g. adding a task) and/or raise events.
2. The command execution can update the History if command executed is an .

UndoableCommand

1. The result of the command execution is encapsulated as a object which is passed back to the .

CommandResult



UI

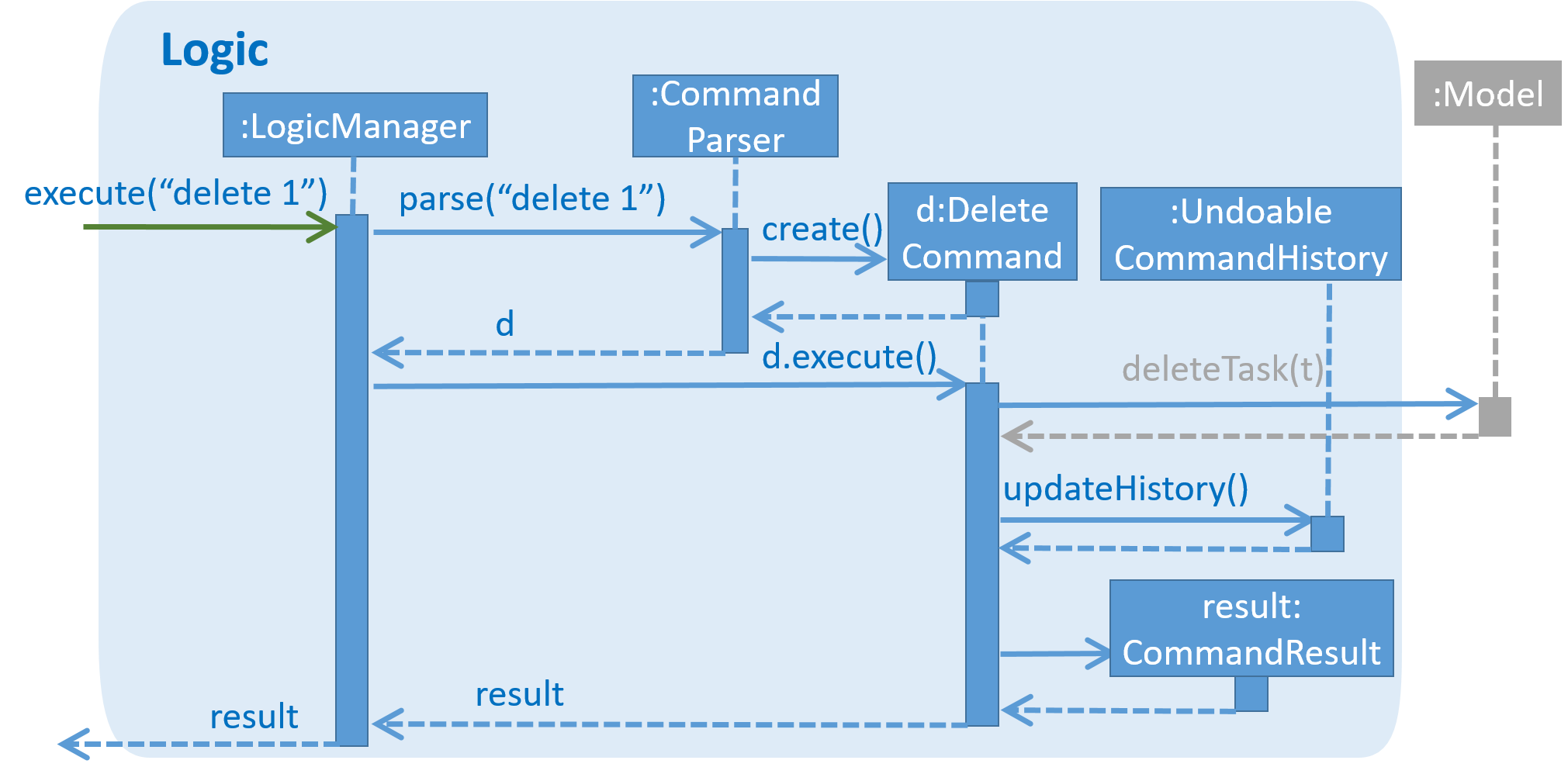
execute("delete 1")

Given below is the Sequence Diagram for interactions within the

Logic

component for the

API call.



*The sequence diagram for the interactions within the component for*

*Logic*

*execute("delete 1")*

## Model componentC:\Users\fishe\AppData\Local\Microsoft\Windows\INetCacheContent.Word\ModelClassDiagram.png

*The class diagram for the Model component of DearJim*

#### API :

Model.java

The ,

Model

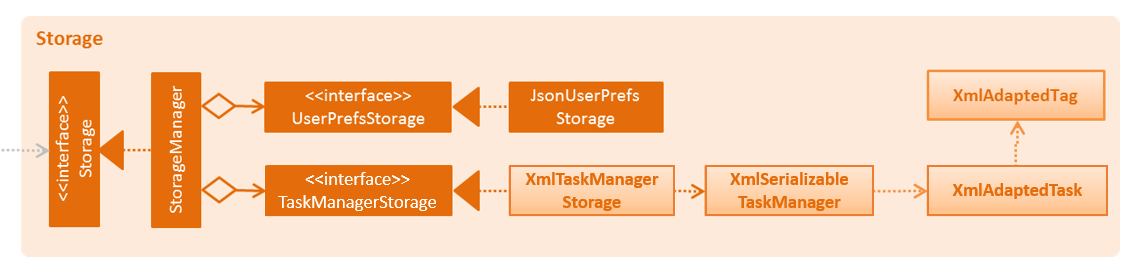
stores the Task Manager data. exposes a

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 component,

Storage

can save objects in json format and read it back.

UserPref

can save the Task Manager data in xml format and read it back.

## Common classes

Classes used by multiple components are in the package.

seedu.taskmanager.commons

# Implementation

LogsCenter

## Logging

We are using logging destinations.

java.util.logging

package for logging. The

class is used to manage the logging levels and

The logging level can be controlled using the logLevel setting in the configuration file (See Configuration)

The for a class can be obtained using LogsCenter.getLogger(Class) which will log messages according to the

Logger

specified logging level

Currently log messages are output through:

and to a

Console

file.

.log

#### Logging Levels

Critical problem detected which may possibly cause the termination of the application

SEVERE : WARNING INFO : In

FINE : D

: Application can continue running, but minor errors may occur formation showing the noteworthy actions by the App

etails 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 folder.

./src/test/java

#### 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 folder and choose >

src/test/java

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 package.

guitests

1. **Non-GUI Tests** - These are tests not involving the GUI. They include,
   1. *Unit tests* targeting the lowest level methods/classes. e.g.

seedu.taskmanager.commons.UrlUtilTest

* 1. *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

* 1. Hybrids of unit and integration tests. These tests are checking multiple code units as well as how the are connected together.

seedu.taskmanager.logic.LogicManagerTest

e.g.

**Headless GUI Testing** : 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 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 C](https://travis-ci.org/)I 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

1. [Crete a new release using GitHub](https://help.github.com/articles/creating-releases/) 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](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.

1. Include those libraries in the repo (this bloats the repo size)
2. 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 |
| \* | 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

, unless specified otherwise)

### Use case: UC01 - Add a task

user

#### MSS

* 1. User enters an command, specifying details of the task to be added

add

* 1. TaskManager parses the command, and adds the task to the current task list

add

* 1. 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 command format

add

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

add

and an example command

add

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 command

list

1. TaskManager parses the command

list

1. 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 command format

list

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

list

and an example command

list

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

edit

INDEX

1. User enters the new values

command, specifying the

of the task in the list to be edited, the fields to be edited and their

1. TaskManager parses the command and looks for the task in the list

edit

1. TaskManager edits the requested fields on the specified task according to the command entered
2. 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 command

edit

and an example command

edit

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 command format

edit

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

edit

and an example command

edit

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

delete

INDEX

1. User enters the

command, specifying the

of the task in the list to be deleted

1. TaskManager parses the command and looks for the task in the list

delete

1. TaskManager deletes the task from the list
2. 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 command format

delete

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

delete

command and an example Use case resumes at step 3

delete

command

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

#### MSS

1. User enters an command

undo

1. TaskManager parses the command

undo

1. TaskManager attempts to identify the latest stored undoable command, reversing the action of that command
2. 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 command, followed by some arguments

undo

1a1. TaskManager parses the

undo

*Use case resumes at step 3*

command, ignoring the arguments that follow

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 command

undo

4a1. User enters the

redo

*Use case ends.*

command (UC06)

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

#### MSS

1. User enters a
2. User enters a

command successfully (UC05) command

1. TaskManager parses the command

undo

redo

redo

undo

1. TaskManager attempts to identify the latest stored command that was undone by an command

, redoing the effects of that

1. 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 command

redo

redo

redo

1b3. TaskManager parses the

*Use case ends.*

command and indicates that there is nothing to

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

redo

1a1. TaskManager parses the

redo

*Use case resumes at step 3*

command, ignoring the arguments that follow

# Appendix C : Non Functional Requirements

1. Should work on any mainstream OS as long as it has Java

1.8.0\_60

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

or higher installed.

# Appendix E : Product Survey

|  |  |  |
| --- | --- | --- |
| **Product** | **Strength** | **Weaknesses** |
| Wunderlist | 1. Beautiful background 2. Cloud sync 3. Able to create folders to group similar tasks 4. Able to add tags to tasks to filter them | 1. No start date or repeat options for tasks 2. No options for subtasks |
| Todo.txt | 1. Works on many platforms, can be accessed on devices that support Dropbox 2. Easily editable format, can be edited in plain text and then displayed with neat styles 3. Can edit with any text editor 4. Easy syncing - can sync through Dropbox 5. Command line support - can edit using command line by a supplied bash script | 1. No support for recurring tasks 2. No options for subtasks 3. Only supports Dropbox, not flexible |
| Google Calendar | 1. Can be synced to mobile devices 2. Alerts via notifications on phones 3. Switches between views easily 4. Minimalistic interface | 1. Requires an Internet connection to be used 2. Cannot be brought up with a keyboard shortcut |
| Remember the milk | 1. Able to support email, text, IM, Twitter, and mobile notifications 2. Able to share lists and tasks with others 3. Synchronises across on all devices 4. Organize with priorities, due dates, repeats, lists, tags 5. Search tasks and notes, and save favorite searches 6. Integrates with Gmail, Google Calendar, Twitter, Evernote, and more | 1. Free version lacks features: E.g. splitting into subtasks 2. Lack keyboard shortcuts |

**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.