FlexiTrack

Supervisor: Chan Jun Wei
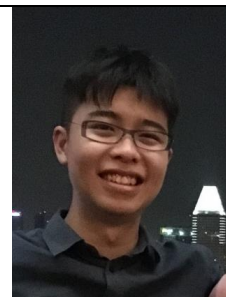

| Antonia Devina | Eddie Tan | Guo ShiMin | Jing Min |

Acknowledgements

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

- This program uses Natty Time (http://natty.joestelmach.com/)

- This program is build based off Address Book Level 4's code sample

Licence : MIT

About Us

Guo Shimin

- Role: Developer
- Responsibilities: Logic, Command
- Features implemented:
    - Help Command
    - Mark Command
    - Unmark Command
    - Block Command
    - Change Storage Path Command
- Code written: [functional code][test code][docs]
- Other major contributions:
    - Did the initial refactoring from AddressBook to FlexiTrack(Partially) [#19]
    - Set up Travis

(Eddie) Tan Yong Sheng

- Role: Developer
- Responsibilities: Documentation, Command, UI
- Features implemented:
    - Find Command
    - Add Command(Recurring task and event)

- Code written: [functional code][test code][docs]
- Other major contributions:
    - Change the UI design (colors, sizing)
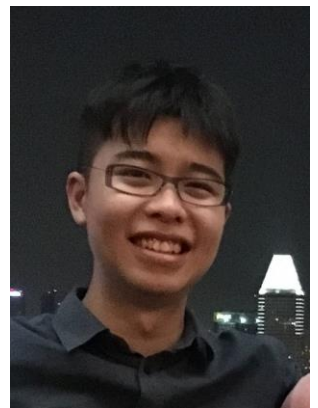   - Bug fixing

---

Antonia Devina

- Role: Developer
- Responsibilities: Integration, Scheduling and Testing
- Features implemented:
    - Mark Command
    - Undo Command
    - List Command
    - Find time (Gap) Command
- Code written: [functional code][test code][docs]
- Other major contributions:
    - Did the initial refactoring from AddressBook to FlexiTrack
        - Fix the basic function to work for flexi track
        - Fix the initial test cases to fit FlexiTrack
    - Implement natty and the DateTimeInfo class and DateTimeInfoParser class
    - Remove Tag and all the classes associated with tag

Jing Min

- Role: Developer

- Responsibilities: Threading, Testing

- Features implemented:
    - Edit Command
    - Sort the Task
    - Shortcut commands
    - Changed undo command to make it easier for redo

- o Redo Command

- Code written: [functional code][test code][docs]

- Other major contributions:

  - o Cleaninging up code
  - o Bug fixing
  - o Design of undo and redo

Mentor

Chan Jun Wei

Contributors

We welcome contributions. See Contact Us page for more info.

User Guide

- [Quick Start](#)
- [Features](#)
- [Time Format](#)
- [FAQ](#)
- [Command Summary](#)

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 `FlexiTrack.jar` from the [releases](#) tab.
3. Copy the file to the folder you want to use as the home folder for your Address Book.

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:
   - `add`add CS2103 tutorial 3 by/ Saturday : adds a task with the title of `CS2103 tutorial 3` to the FlexiTrack.
   - `delete`3 : deletes the 3rd contact shown in the current list
   - `exit` : exits the app
7. Refer to the [Features](#) section below for details of each command.

Features

Command Format

- Words in `square brackets ([])` are the parameters.
- Items within `arrow signs (<>)` are optional.
- Items with `...` after them can have multiple instances.
- The order of parameters is fixed except for edit command.

Adding a task: add

Shortcut : a

Adds a task to the FlexiTrack.

Format: `add [task title] < by/ [deadline] >`

Examples:

- `add CS2103 tutorial 3`
- `add CS2103 tutorial 3 by/ Saturday`
- `a CS2103 tutorial 3 by/ tmr 9am`

Adding an event: add

Shortcut : a

Adds a event to the FlexiTrack.

Format: `add [event title] from/ [starting time] to/ [ending time]`

Examples:

- `add Bintan trip from/ Saturday to/ Sunday`
- `a CS2103 Lecture from/ Friday 2pm to/ Friday 4pm`

Adding an recurring task: add

Shortcut : a

Adds a task(recursive) to the FlexiTrack.

Format: `` `add [task title] fr/ [number of occurrence] ``

Examples:

- `` `add Plan meet-up for assignment fr/ 5 ``
- `` `add Watch DareDevil Season 1 EP fr/ 10 ``

Adding an recurring task (with deadline): add

Shortcut : a

Adds a task(recursive) to the FlexiTrack.

Format: `add [task title] fr/ [number of occurrence] ty/ [daily | weekly | monthly] from/ [starting time] to/ [ending time]`

Examples:

- `add Submit PC1222 Labsheet fr/ 5 ty/ week by/ Tuesday 5pm`
- `add complete CS2103 post-lecture quiz fr/ 10 ty/ week by/ Sunday 10pm`

Adding an recurring event: `add`

Shortcut : `a`

Adds a event(recursive) to the FlexiTrack.

Format: `add [event title] fr/ [number of occurrence] ty/ [daily | weekly | monthly] from/ [starting time] to/ [ending time]`
Examples:

- `add attend PC1222 tutorial fr/ 5 ty/ weekly from/ Tuesday 5pm to/ Tuesday 6pm`
- `add attend CS2103 lecture fr/ 10 ty/ weekly from/ Fri 2pm to/ Fri 4pm`

Block multiple time slot for an event : `block`

Shortcut : `b`

Block another time slot for an unconfirmed existing event.

Format: `block [Description] from/ [starting time] to/ [ending time]`
The new block period must not overlapping current block task. New event will not be allow to add in if the period of the new event overlapping any blocked task from block list.
Examples:

- `block for cs2103 project from/ 5pm to/ 7pm`

Find free time slots: `gap`

Shortcut: `g`

Find and list free time slots in the schedule that is equal to or longer than the specified timing (in hours).

Format: `find time [number of hours] < [number of slots to find] >`
If there is there is a time slot longer than the required free time slot, then the free time period will be return to you By default, find time will only give a single free slot when the number of slots required is not keyed in.
Examples:

- `gap 3`

  You have a minimum of 3 hours free time slot between: today 5pm - 9pm.
- `g 5 3`

  You have a minimum of 5 hours free time slot between: Monday 2pm - 9pm, Tuesday 1pm - 6pm and Saturday 9am - 5pm.

Deleting a task or event : `delete`

Shortcut : `d`

Deletes the specified task/event from the FlexiTrack.

Format: `delete [index]`

Deletes the task/event 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, ...

Examples:

- `delete 2`
  Deletes the 2nd task/event in the address book.
- `d 1`
  Deletes the 1st task/event in the results of the `find` command.

Clear the FlexiTrack : `clear`

Shortcut : `c`

Clears the FlexiTrack, resetting it to a blank slate.

Format: `clear`

The command can be undone as long as the user does not exit FlexiTrack after clearing.

Edit a task or event: `edit`

Shortcut : `e`

Edits the specified task/event from the FlexiTrack.

Format: `edit [index] <by/ [deadline]> <n/ [title]> <from/ [starting time]> <to/ [ending time]>`

- Edits the task/event 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, ...
- Edit parameters must fit the type of task / event being edited. e.g. duedate should only be edited on a task.
- Floating tasks can be converted into tasks or events by editing the appropriate parameter.
- User cannot edit a floating task into an event with only a starting time but no ending time or vice versa.

Examples:

- `edit 2 n/ Name Edited`
  Edits the title of the task/event.

- `e 1 from/ today to/ tomorrow`
  Edits the start and end times of the specified event.

Mark a task as complete : `mark`

Shortcut : `m`

Mark an existing task to complete and move it to the bottom of the list.
Format: `mark [index]`
Mark the task/event 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, ...
Examples:

- `mark 5`

Mark a task as complete : `unmark`

Shortcut : `u`

Mark an existing task to complete and move it to the bottom of the list.
Format: `unmark [index]`
Unmark the taks/event 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, ...
Examples:

- `unmark 5`

Finding a task or an event containing any keyword in their title: `find`

Shortcut : `f`

Finds a task ot an event whose title contain any of the given keywords.
Format: `find KEYWORD [MORE_KEYWORDS]`

- The search is non case sensitive. e.g `soccer` will match `Soccer`
- The order of the keywords does not matter. e.g. `soccer dinner` will match `dinner soccer`
- Only the task/event title is searched.
- Only full words will be matched e.g. `socc` will not match `soccer` (unless 'f/' keyword is used)
- Task or event matching at least one keyword will be returned (i.e. `OR` search).
  e.g. `soccer` will match `soccer training`

- Search by exact task name can be activated with the shortcut 'f/' before the task name.

Examples:

- `find Soccer`
  Returns `Soccer training` but not `soccer training`
- `find assignment dinner mid-term`
  Returns Any task/event having `assignment`, `dinner`, or `mid-term` in the title
- `f attend CS2103 lecture`
  Returns Any task/event having `attend`, `CS2103`, or `lecture`

Finding a specific task or an event containing an exact phrase in their title: `find f/`

Shortcut : `f f/`

Finds a task of an event whose title contain any of the given keywords.
Format: `find f/ EXACT PHRASE`

List: `list`

Shortcut : `l`

Lists tasks and events that match the specified filter.
Format: `list <filter>`
Accepted filters include:

- future (shortcut : f)
- past (p)
- next week (nw)
- next month (nm)
- last week (lw)
- last month (lm)
- mark (m)
- unmark (um)
- block (b) Unmarked floating tasks will be listed in all the filter except in block and mark filters.

Examples:

- `list next month`
  Returns a list of next month's tasks and events

- `l b`
  Returns a list of all the blocked events

Undo operations : `undo`

Shortcut : `un`

Undo the previous operation.

Format: `undo`

The command will only undo commands entered during the current session of FlexiTrack Undo works for: add, delete, clear, mark, unmark, block.

Redo operations : `redo`

Shortcut : `re`

Redo the previously undone operation.

Format: `redo`

The command will only redo commands undone during the current session of FlexiTrack

Specify storage location: `cs`

Specify the storage location where the program save the data.

Format: `cs [path]`

Examples:

- `cs data/newStorage`
- `cs newDataStorage`

Limitation: This feature Only allow user to change storage path within the FlexiTrack folder.

[path] can only contains alphanumeric or following special character '\', '/', '-', ':', '.', '_'.

Exiting the program : `exit`

Shortcut : `q`

Exits the program.

Format: `exit`

Viewing help : `help`
Shortcut : `h`

Format: `help` or `help [command word]`
Examples:

- `help add`
- `help edit`
- `h delete`

Help is also shown if you enter an incorrect command e.g. `abcd` default help message will show a list of all command word, e.g. enter `help`

Saving the data

Address book data are saved in the hard disk automatically after any command that changes the data.
There is no need to save manually.

Time Format

FlexiTrack support various timing input. Here are some examples!

Exact timing

User Input	Timing information read by FlexiTrack
21 June 2018 4pm	Jun 21 16:00:00
1st January 7.20	Jan 01 07:20
April 22nd 5am	Apr 22 05:00

Relative timing

If today is 1st of February a relative timing input is also possible with FlexiTrack

User Input	Timing information read by FlexiTrack
Tomorrow 4pm	Feb 02 16:00:00
Next week 720am	Feb 08 07:20
3 weeks 2 pm	Feb 22 14:00
next month 8am	Mar 01 08:00

World-wide event

User Input	Timing information read by FlexiTrack
Christmas morning 9am	Dec 25 2016 16:00
Easter dinner 7pm	Apr 16 2016 19:00

Notes on FlexiTrack timing

1. FlexiTrack does support year. However, make sure that you also specify the hour of the timing as FlexiTrack will choose timing over year when it is uncertain.
2. When you do not specify the exact timing, FlexiTrack will assign your task to be 7.59 for due date and starting time, and 16.59 for ending time.

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 FlexiTrack folder.

Command Summary

Command	Shortcut	Format
Add task	a	add [task title] <fr/ [number of recurrances] ty/ [day | week | month]> <by/ [deadline]>
Add event	a	add [event title] <fr/ [number of recurrances] ty/ [day | week | month]> from/ [starting time] to/ [ending time]
Block	b	block [description] from/ [starting time] to/ [ending time]
Find time	g	gap [number of hours] < [number of slots to find] >
Delete	d	delete [index]
Clear	c	clear
Edit	e	edit [index] <by/ [deadline]> <n/ [title]> <from/ [starting time]> <to/ [ending time]>
Mark	m	mark [index]
Unmark	u	unmark [index]
Find	f	find [key words] < [key words] >
List	l	list <filter>

Command	Shortcut	Format
Select	s	`select [index]`
Undo	un	`undo`
Redo	rd	`redo`
Change Storage Path	cs	`cs [path]`
Exit	q	`exit`
Help	h	`help <command word>`

Developer Guide

- [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](#)

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'.

- o 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)
- o If Eclipse auto-changed any settings files during the import process, you can discard those changes.

Design

Undo/Redo Implementation

Each undoable command has an execute() method and an executeUndo() method.

2 static Command stacks are maintained. One in the UndoCommand class (doneCommandStack) and the other in the RedoCommand class (undoneCommandStack). From application startup, whenever an undoable command is entered, the created command object is first populated with the data of whatever the command changes, before the execute() method is called. The changes are then applied and the entire command object is pushed into the doneCommandStack.

Subsequently, when the UndoCommand is entered, the last command entered is popped out of the doneCommandStack and the executeUndo() method is called. The command is then pushed into the undoneCommandStack.

When the RedoCommand is entered, the last command undone is popped out of the undoneCommand Stack and the execute() method is called, effectively reexecuting the undone command. The command is then pushed into the doneCommandStack to ensure that the we can undo and redo the same command over and over.

This method of implemeting the undo/redo functionality was chosen due to its simplicity of implementation, as well as the relatively light memory usage, especially when compared to the 'save entire state' method of implementation.

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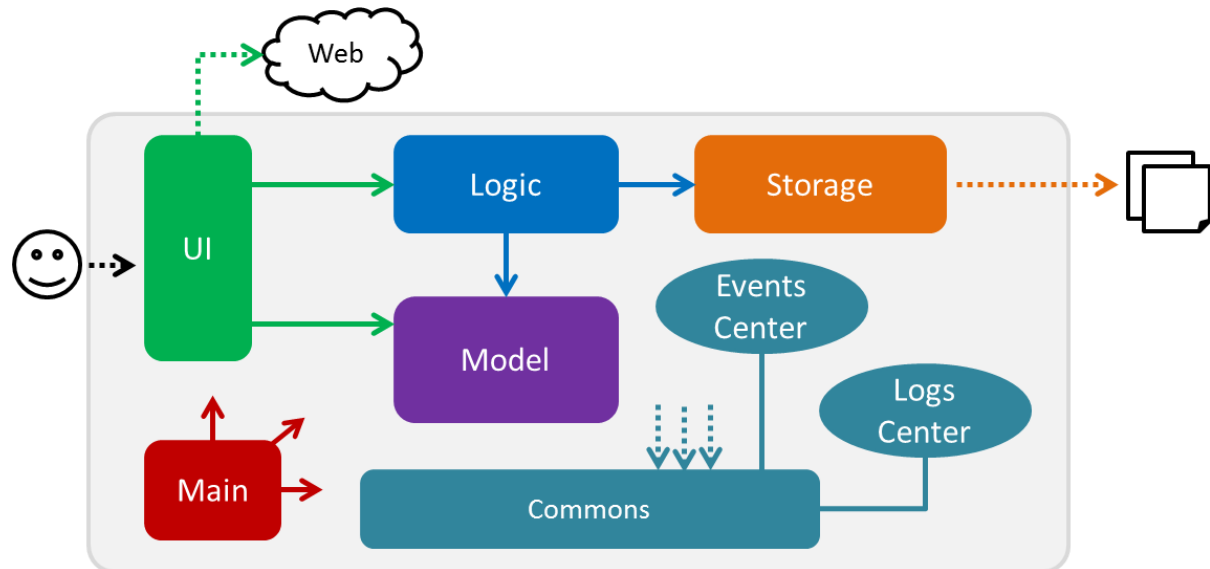
The **Architecture Diagram** given above explains the high-level design of the App. Given below is a quick overview of each component.

`Main` has only one class called [`MainApp`](../src/main/java/seedu/ /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](#)) 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.
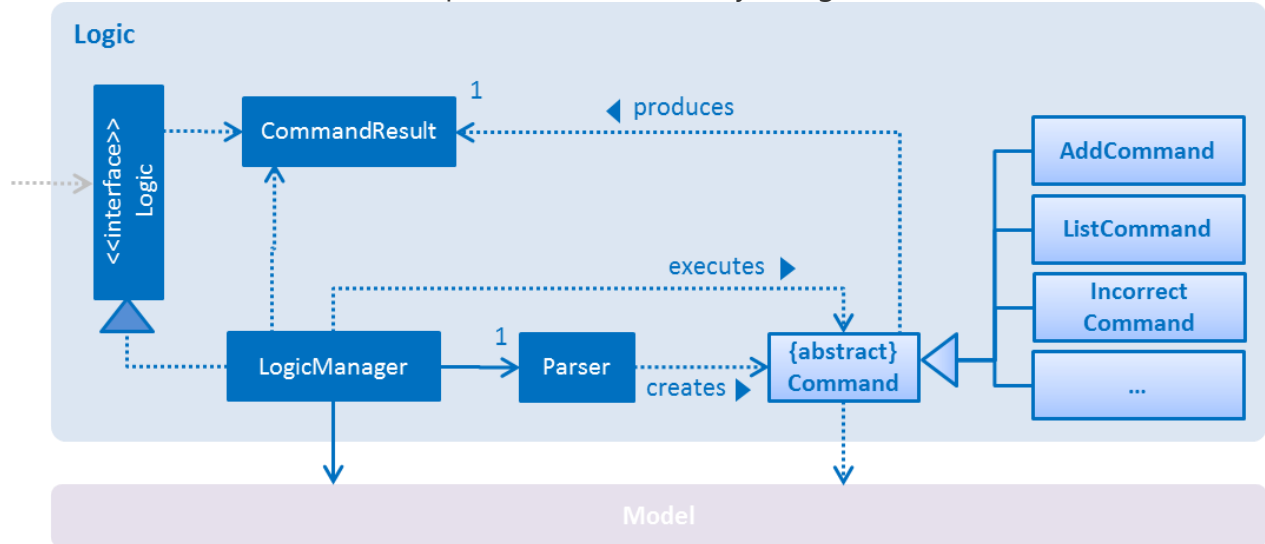

The *Sequence Diagram* below shows how the components interact for the scenario where the user issues the command `delete 3`.
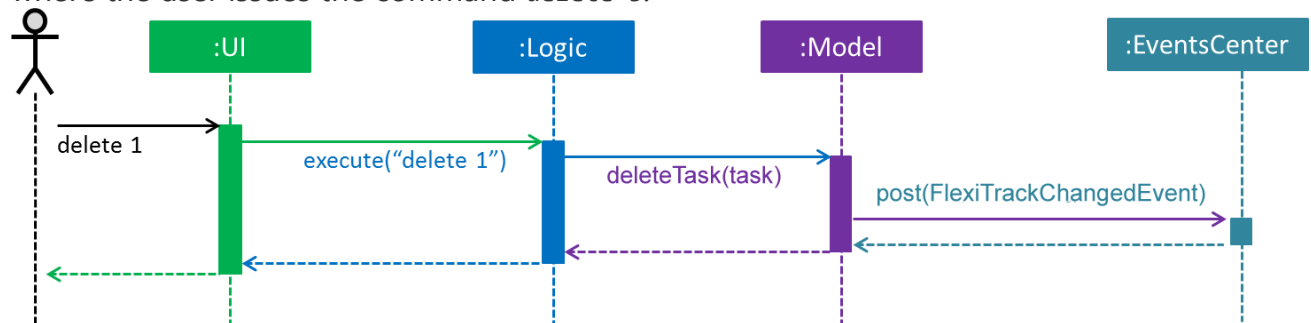
Note how the `Model` simply raises a `FlexiTrackChangedEvent` when the FlexiTrack data are changed, instead of asking the `Storage` to save the updates to the hard disk.
The diagram below shows how the `EventsCenter` reacts to that event, which eventually results in the updates being saved to the hard disk and the status bar of the UI being updated to reflect the 'Last Updated' time.
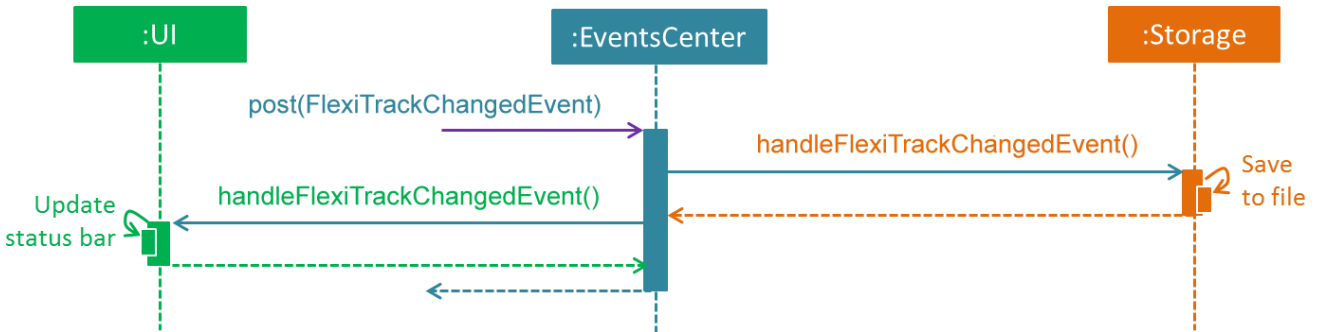
Note how the event is propagated through the `EventsCenter` to
the `Storage` and `UI` without `Model` having to be coupled to either of them. This is an
example of how this Event Driven approach helps us reduce direct coupling between
components.
The sections below give more details of each component.

UI component
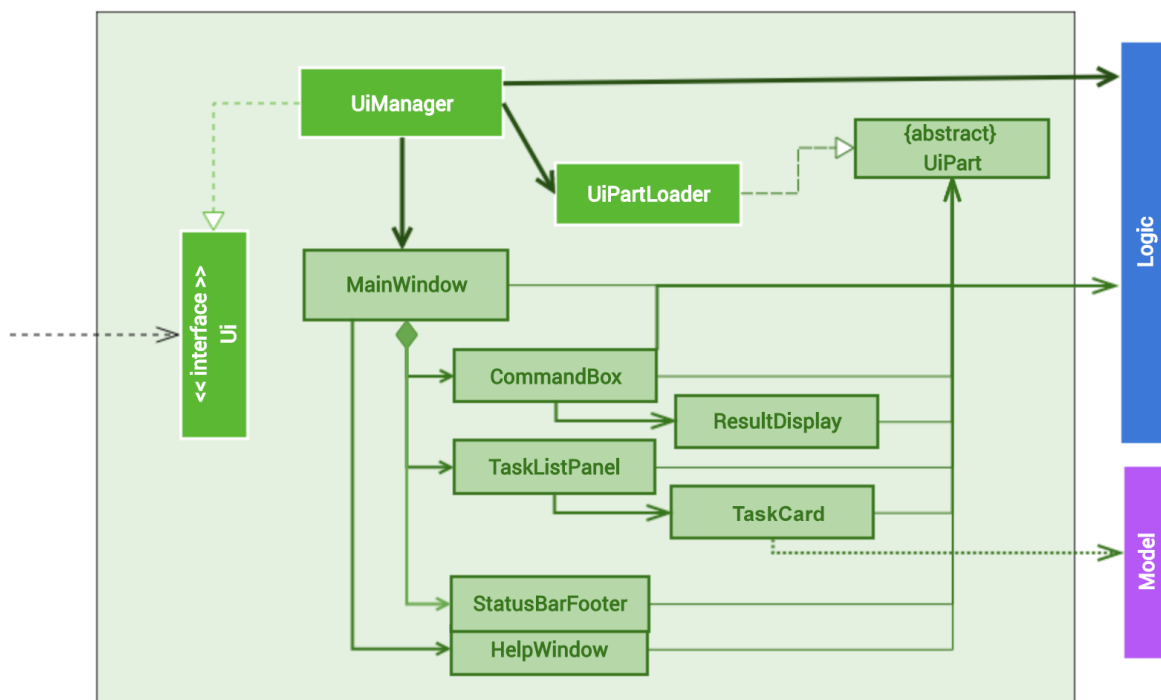
API : `Ui.java`
The UI consists of a `MainWindow` that is made up of parts
e.g.`CommandBox`, `ResultDisplay`, `TaskListPanel`, `StatusBarFooter`etc. All these, including

the `MainWindow`, inherit from the abstract `UiPart` class and they can be loaded using the `UiPartLoader`.

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

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

The `UI` component,

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

Logic component
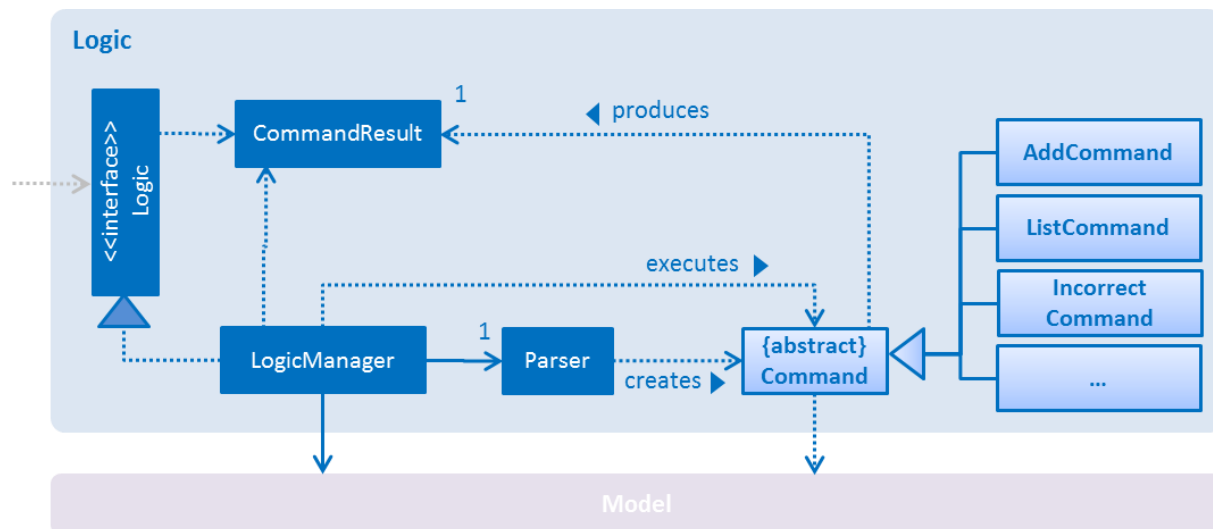
API : `Logic.java`

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`.

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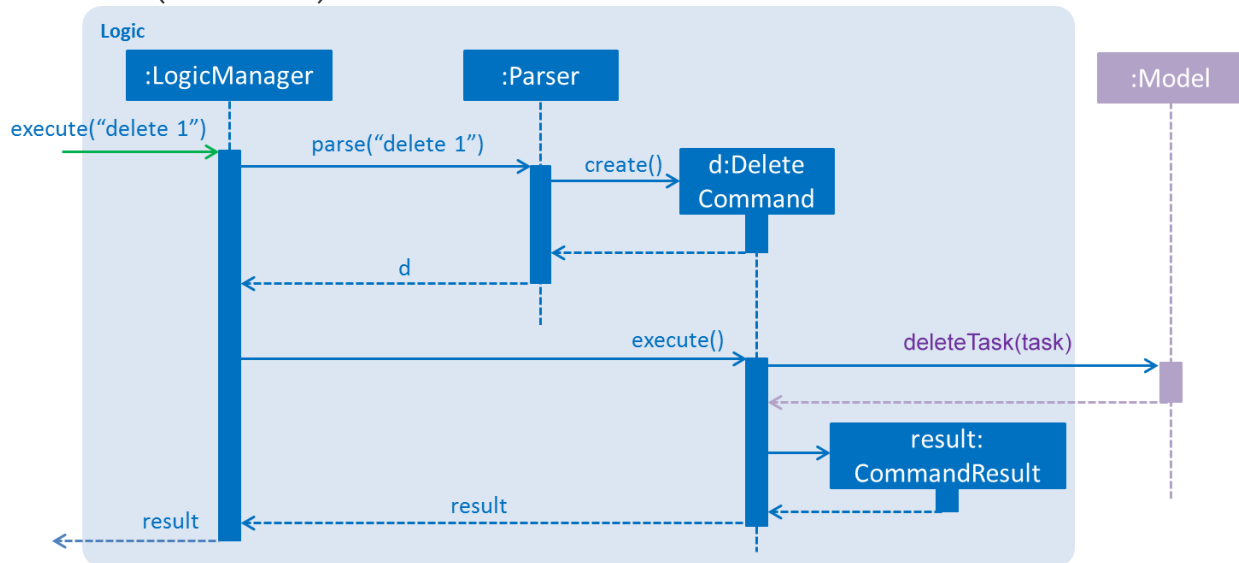

Model component
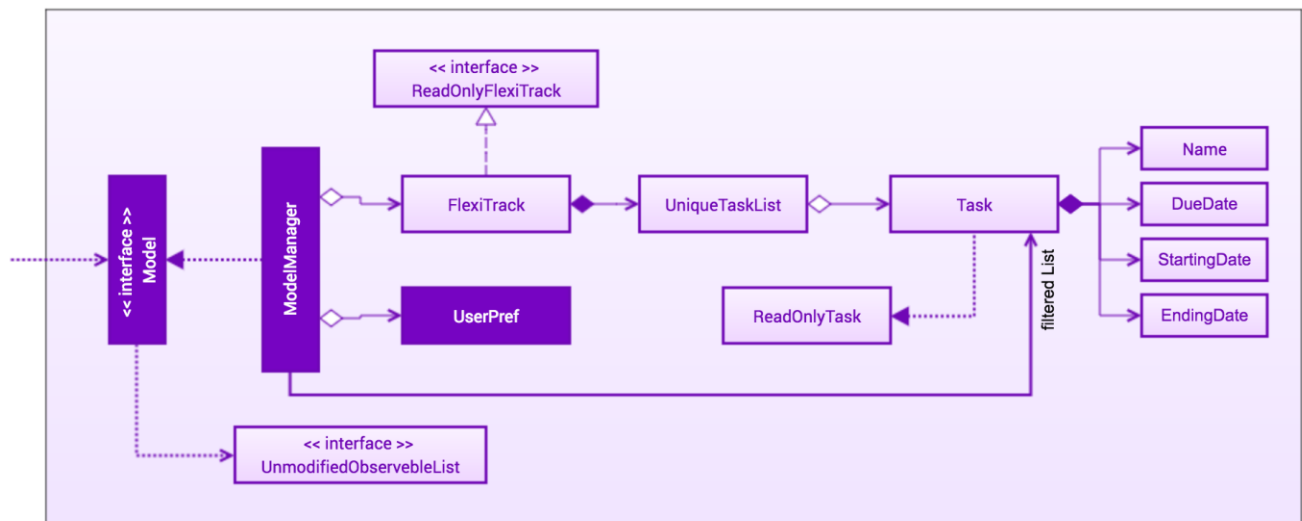
API : `Model.java`
The `Model`,

- stores a `UserPref` object that represents the user's preferences.
- stores the FlexiTrack 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
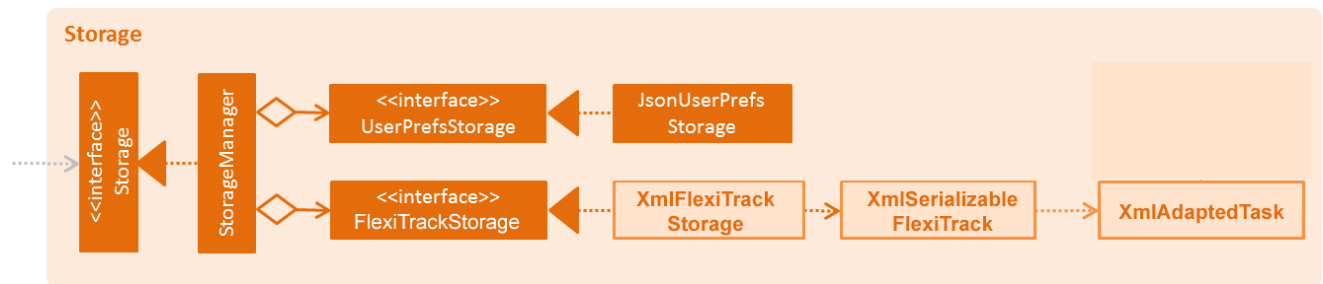
API : `Storage.java`
The `Storage` component,

- can save `UserPref` objects in json format and read it back.
- can save the Flexi Track data in xml format and read it back.

Common classes

Classes used by multiple components are in the `seedu.flexitrack.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` : 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.

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

Using Gradle:

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

We have two types of tests:

1. **GUI Tests** - These are *System Tests* that test the entire App by simulating user actions on the GUI. These are in the `guitests` package.

2. **Non-GUI Tests** - These are tests not involving the GUI. They include,

   i. *Unit tests* targeting the lowest level methods/classes.
      e.g. `seedu.flexitrack.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.flexitrack.storage.StorageManagerTest`

iii. Hybrids of unit and integration tests. These test are checking multiple code units as well as how the are connected together.
e.g. `seedu.flexitrack.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. [Crete a new release using GitHub](#) and upload the JAR file your created.

Managing Dependencies

A project often depends on third-party libraries. For example, FlexiTrack 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 (must have) – * * *, Medium (nice to have) – * *, Low (unlikely to have) – *

Priority	As a ...	I want to ...	So that I can...
* * *	As a user	I want add a task by specifying a task description only,	So that I can record tasks that need to be done 'some day'.
* * *	As a user	I want to add an event,	So that I can track the events that I have.
* * *	As a user	I want to add a deadline to a task,	So that I can organised my time better.
* * *	As a user	I want to find upcoming tasks,	So that I can decide what needs to be done soon.
* * *	As a user	I want to edit a task,	So that I can change the details of a task.
* * *	As a user	I want to delete a task,	So that I can get rid of tasks that I no longer care to track.
* * *	As a user	I want to undo an operation,	So that I do not need to be afraid to make a mistake.
* * *	As a new user	I want to view more information about a particular command,	So that I can learn how to use various commands.
* * *	As a new user	I want to mark a task as	So that I will be reminded that I have completed the task and not to worry

Priority	As a ...	I want to ...	So that I can...
		done,	about it.
* * *	As an advanced user	I want to use shorter versions of a command,	So that type a command faster.
* * *	As a user	I want to specify a specific location to store my data,	So that I can choose to store the data file in a local folder controlled by a cloud syncing service, which allowing me to access data from other computers.
* * *	As a user	I want to block a multiple time slot for uncertain time of an event,	So that I will not plan something on the uncertain timing.
* * *	As a user	I want to find a free time slot in my schedule,	So that can plan my time better.
* *	As a user	I want to add a recurring task,	So that I do not have to keep adding the same task every week/day.
* *	As a user	I want to find a task that is similar to what I'm searching,	So that I can see similar tasks.
* *	As a user	I want to find a task with the exact name,	So that I can locate the exact task I'm looking for.
* *	As a user	I want to launch the program with key combination (ALT+SPACE),	So that I can minimized time spent on clicking.

Priority	As a ...	I want to ...	So that I can...
* *	As a user	I want to do everything by typing,	So that I can do everything much faster.
* *	As a user	I want to do multiple undo,	So that If I have a mistake i do not have to trace back and correct it manually.
* *	As a user	I want to do multiple redo,	So that if I decided not to take the changes from undo, I can go back to use my original file.
* *	As a user	I want to have a high flexibility of command format,	So that I do not need to memorize or worry about the format of the input.
* *	As a new user	I want to have a guided tour for the app,	So that I will know how to use the app.
* *	As a user	I want to receive feedback while typing,	So that I know that i have successfully key in what I typed.
*	As a user	I want to add a tag to a task,	So that I know at a glance what the task is for.
*	As a user	I want to sort the task by the tag,	So that I can see related tasks together.
*	As a user	I want to add a sub-task under a task,	So that I can break down my task into smaller tasks.
*	As a user	I want to add a note to a task,	So that I will not forget the details about the task, if there is any.

Priority	As a ...	I want to ...	So that I can...
*	As a user	I want to mark a sub-task as done,	So that I will not worry about it anymore.
*	As a user	I want to set reminder for a task,	So that I will not forget to do the task.
*	As a user	I want to set/pick the reminder time,	So that I can assign reminder according to how long the task need to be completed.

Appendix B : Use Cases

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

Use case: UC01 – add new task or event

MSS

1.User request to add a new task or an event 2.System add the data Use case ends

Extensions

1a. The user input is invalid

1a1. System output an invalid input message Use case ends
1b. The user input existing active task or event title

1b1. System output warning message that there is existing active task or event title Use case ends
1c. The user wish to add an event on top of another event

1b1. System shows warning that the event clash continue 2

Use case: UC02 – Edit a task

MSS

1.User request to edit an existing task giving the index number and the new task
2.System edit the task Use case ends

Extensions

1a. The user input is invalid

1a1. System output an invalid input message Use case ends

Use case: UC03 – delete

MSS

1.User request to delete an existing task/event
2.System delete the task/event Use case ends

Extensions

1a. The user input an invalid list number

1a1. System output an invalid input message Use case ends

Use case: UC04 – mark

MSS

1.User request to mark an event as done
2.System mark the event
Use case ends

Extensions

1a. The user input an invalid list number

1a1. System output an invalid list number Use case ends

Use case: UC05 – unmark

MSS

1.User request to unmark an event as not done
2.System unmark the event
Use case ends

Extensions

1a. The user input an invalid list number

1a1. System output an invalid list number Use case ends

Use case: UC06 – specify storage

MSS

1.User request to specify storage
2.System move the storage to the specify directory
Use case ends

Extensions

1a. The user input an invalid directory

1a1. System output an invalid directory message Use case ends

Use case: UC07 – block multiple time slot

MSS

1.User request to block an extra time slot for an event
2.System block the extra time slot
Use case ends

Extensions

1a. The user input an invalid list number

1a1. System output an invalid list number message Use case ends
1b. The user select a busy timing (unavailable timing)

1b1. System output an unavailable timing message Use case ends

Use case: UC09 – Find a free time with a specified length and number of occurrence

MSS

1.User request to list the task in specific order 2.System shows the list of task at the particular timing Use case ends

Extensions

1a. The user input a command word not in the list of choice

1a1. System output an invalid input and how to use list command Use case ends

Use case: UC09 – Find a free time with a specified length and number of occurrence

MSS

1.User request to find a specified length of free time (optional: and number of occurrence)
2.System shows the available timing Use case ends

Extensions

1a. The user spell out the length of the free time

1a1. System output an message to repeat command with digit Use case ends
1b. The user spell out the number of occurrence of the free time

1b1. System output an message to repeat command with digit Use case ends
1c. The user put in an invalid argument

1c1. System output a message on how to use the gap command Use case ends
The implementation of finding a free time is similar to the delete command shown earlier section. Below is the sequence diagram of finding a free

time:
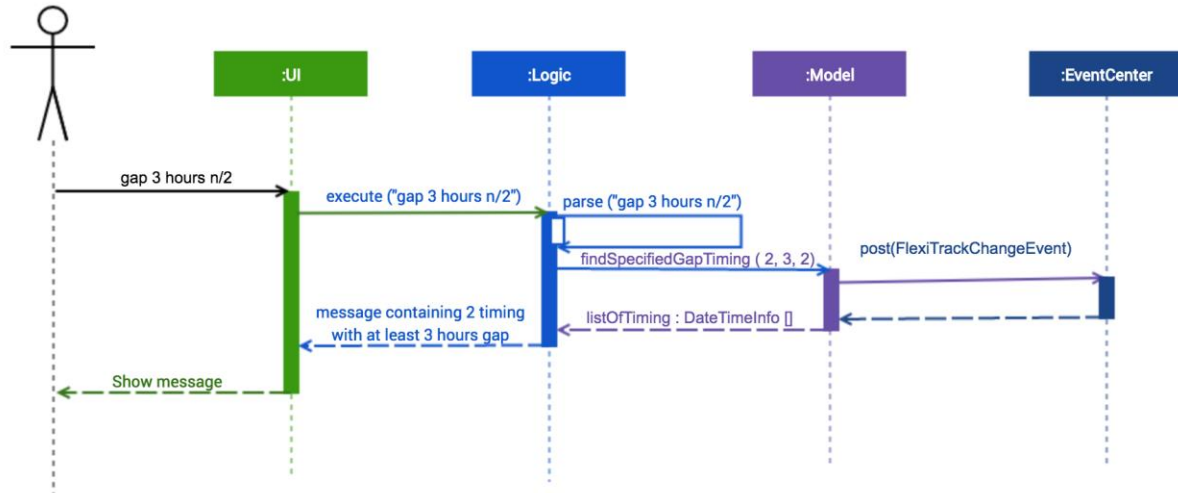
The two main implementation is parsing and preparing the argument to be passed to the GapCommand class and to find the free time itself. Below are the activities diagram for the two implementations:
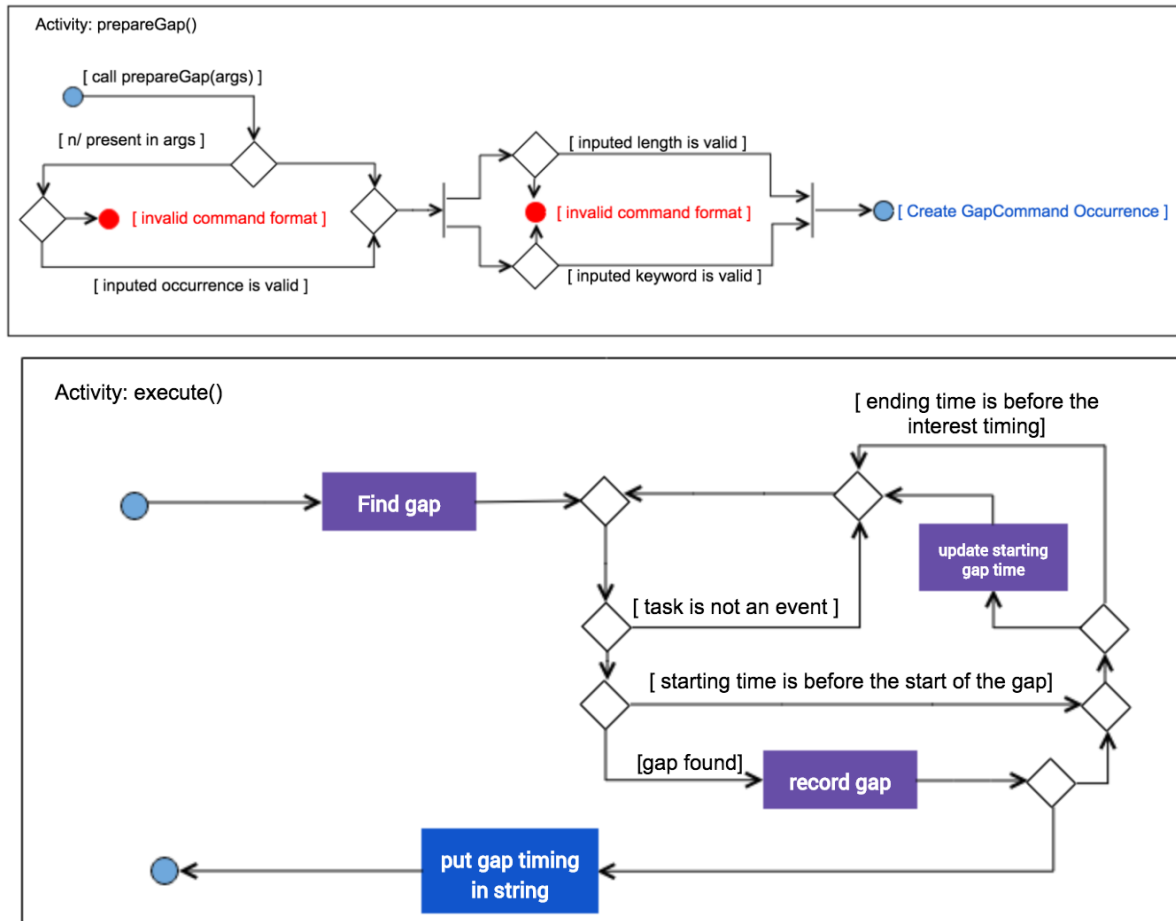
Appendix C : Non Functional Requirements

1. Should work on any mainstream OS as long as it has Java `1.8.0_60` or higher installed.
2. Should come with automated unit tests and open source code.
3. User must get the command done within three mouse clicks
4. The total file size must be less than 15MB
5. Search result must be returned in 0.3ms
6. Buttons and text field must be clearly labeled
7. Make the more important/urgent tasks stand out
8. The longer events will look different from shorter events
9. An event will look differently from a task thus easy to differentiate 10.Easy to see free slots 11.Intuitive vocabulary and design

{More to be added}

Appendix D : Glossary

Active task

A task that have not passed the deadline and have not been marked as done

Active event

An event that has not passed yet

Event

An occasion that goes on for a period of time. It has a starting time and an ending time.

Task

An occasion that need to be completed by a certain date called deadline

Mainstream OS

Windows, Linux, Unix, OS-X

Appendix E : Product Survey

Google Calendar

Pros:

Help to organise schedules and pre-planned them in advance Able to show schedules in multiple view (3 days, day, month or week) format Easy to synchronise planned schedules across multiple devices

Cons:

Unable to use single line command to add new events Each event creation take up around 2 minutes of the user time to fill up the details Unable to do partial search for a particular event name Confusing timing display on each event. (E.g 11p for night event and 11 for morning event)

Any.Do

Pros:

Help to add a quick task to the list Able to invite friends in the same event or task Able to send notification to the user before the event start Able to add new task(s) to personalised folder