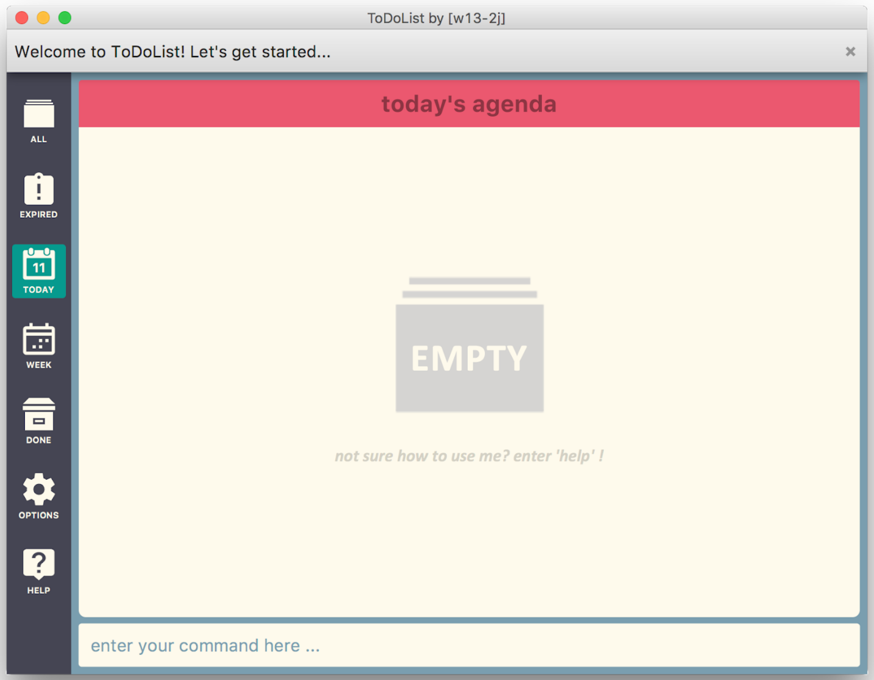
ToDoList

Project Manual


ToDoList by [w13-2j]
Welcome to ToDoList! Let's get started...
today's agenda
ALL
EXPIRED
TODAY
WEEK
DONE
OPTIONS
HELP
EMPTY
not sure how to use me? enter 'help' !
enter your command here ...

Supervisor: Akshay Narayan **Extra Feature:** GoodGUI

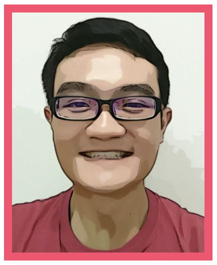
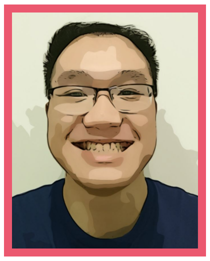
		
Huang Lie Jun (A0123994W) Team Lead User Experience	**Xiao Yuxin (A0131334W)** Documentation Database and Logic	**Zhang Jiyi (A0130620B)** Integration Parser and Logic

Table of Contents

1 Acknowledgements and Credits

We would like to thank all external library authors, teams, individuals who have contributed to *ToDoList* in one way or another. Without your contribution, we would not have been able to be as successful as we are today. Special thanks goes to our **CS2103T lecturer Dr. Henry Chia** for the inspiring and engaging lectures, **CS2103T tutor Akshay Narayan** as well as our **CS2101 tutor Chong Peck Marn Sarah** for their patience, guidance and invaluable advice.

1.1 Media and Assets

We are extremely grateful to **Freepik** and **IconFinder** for all the image assets and resources, such as icons, glyphs, as well as for providing us with inspiration for our graphics user interface design. We are also thankful to **colorhunt.co** for their amazing colour palettes.

1.2 External Libraries

We have used **JavaFx** and **ControlsFx** for our graphics user interface (GUI) design. They have been immensely useful for graphics element such as panel views, notification panes, layout and CSS styling. We have also used **SceneBuilder**, supplementary to JavaFx in handling FXML files with ease.

We have utilised **Prettytime** and **Natty** for date-time formatting, and natural language parsing. This has allowed us to provide our users with more flexibility and improve the user experience.

Another library we have used is **JUnit**, for unit testing. It has helped us to deliver reliable, error-free product like ToDoList.

We have also used **GSON** to convert Java objects into JSON format for easy parsing and handling.

2 Introducing ToDoList

ToDoList is a desktop scheduler programmed in Java, designed to facilitate daily planning of tasks. It mainly functions to receive input from keyboard and does not require any internet connection.

2.1 Objectives and Target Audience

Our aim is to provide an **elegant and intuitive solution** to scheduling tasks in a highly efficient and stressful environment. Our product will serve as an **enabler** for our users to excel by **assisting them to focus** on their schedule and outstanding tasks.

ToDoList can be used by many, but we are mainly targeting **office executives** who are **comfortable with using keyboard**.

2.2 Design Philosophy

Our design philosophy focuses on three elements: Simple, Swift and Smart.

Simple refers to the simplicity in the usage of ToDoList. Users will only require minimum effort to create tasks. This is established through **flexible commands scheme**. User will also take minimum effort to view and perceive tasks. This is established through **visual guides**, **colour labelling**, **clean formatting** of the interface.

Swift refers to how fast users can interact with the application. By providing **keyboard shortcuts** and **mouse control redundancies**, we allow users to interact with the program as flexibly and seamlessly as possible. **Command short forms** also enable the users to minimise typing and spend more time on completing their tasks.

Smart refers to accuracy in the interpretation of users' intention as well as the response to assist user in their goals. We achieve smart interpretation through utilising **natural language processing libraries** such as **Natty** and **PrettyTime**, allowing users the freedom from fixed syntax and command formats. We achieve smart responses through **meaningful feedback notifications** with **action prompts** and **suggestions**.

3 User Guide

3.1 Getting Started: Story Behind ToDoList

Congratulations to be our new user!

Inspired by **Wunderlist** and **todoist**, *ToDoList* displays your schedule in an **intuitive and simple** interface. By doing so, it helps you to **focus and complete** your outstanding tasks efficiently. To fit in aptly with your busy schedule, *ToDoList* focuses on **keyboard manipulation and shortcuts** for **swift control**. *ToDoList* is also **smart** enough to understand your schedule through natural commands.

Through meticulous and astute design, we aim to create a seamless experience for users like you, so as to turn task-scheduling into an enjoyable routine. Excited yet? Follow us through this user guide for a tour of the amazing features we have to offer in *ToDoList*!

3.2 Setting Up: Getting You Ready

Before we begin, let us get you ready for the application in 3 steps:

1

Install Java
As *ToDoList* runs on Java, be sure to install Java JDK 8u73 or later. Note that earlier Java versions may not be compatible with our application.

You can get the latest JDK here:
http://www.oracle.com/technetwork/java/javase/downloads/index.html

2

Download *ToDoList*
To download our application, you may visit our GitHub repository for our latest release.

You can get our application here:
https://github.com/cs2103jan2016-w13-2j/main/releases

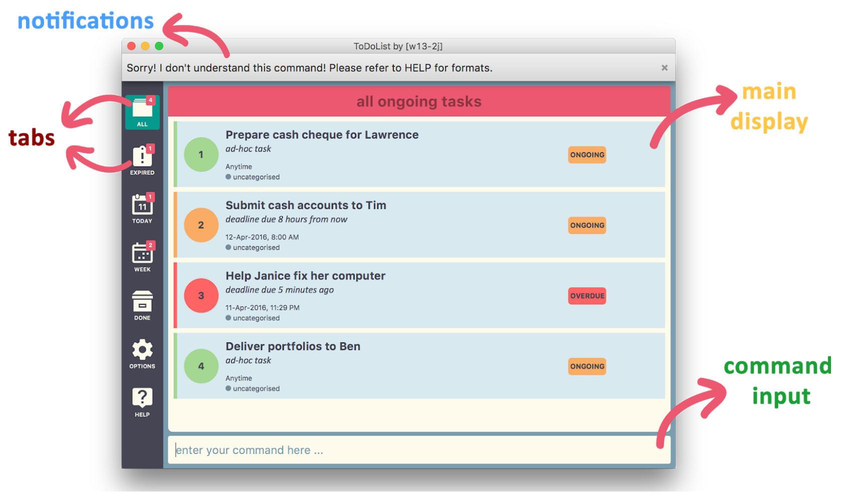
3

Launching *ToDoList*
To start using our application, you simply double-click on the downloaded release (*.jar file) to execute it from the download directory.

It is that simple! Welcome to *ToDoList*!

3.3 Façade Overview: Beauty and the Beast

Upon executing *ToDoList*, you will be greeted by a beautiful interface, as follows:



ToDoList by [w13-2j]

Sorry! I don't understand this command! Please refer to HELP for formats.

all ongoing tasks

1 Prepare cash cheque for Lawrence
ad-hoc task
Anytime
● uncategorised
ONGOING

2 Submit cash accounts to Tim
deadline due 8 hours from now
12-Apr-2016, 8:00 AM
● uncategorised
ONGOING

3 Help Janice fix her computer
deadline due 5 minutes ago
11-Apr-2016, 11:29 PM
● uncategorised
OVERDUE

4 Deliver portfolios to Ben
ad-hoc task
Anytime
● uncategorised
ONGOING

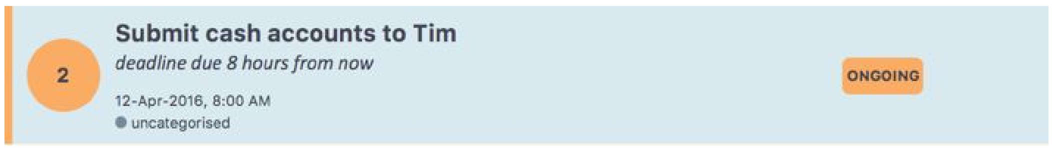
enter your command here ...

(Figure 1: Anatomy of ToDoList)

3.3.1 Main Display

Main Display (see #1 in Figure 1) is where the main content is shown in the *ToDoList* interface. It displays the content of the tab which you are currently on. In Figure 1, we begin with "Home" as our current tab. With no tasks previously created, you will see the thumbs-up glyph, signifying that you have no outstanding tasks.

A typical task item will look like this (see Figure 2.0):



2 Submit cash accounts to Tim
deadline due 8 hours from now
12-Apr-2016, 8:00 AM
● uncategorised
ONGOING

(Figure 2: Anatomy of a task item)

3.3.2 Tabs

Tabs (see #2 in Figure 1) indicates the current tab you are on by highlighting the corresponding tab icon. You may also choose which tabs to visit, either through commands or by using mouse click(s).

3.3.3 Notification Bar

Notification Bar (see #3 in Figure 1) provides notifications, reminders and general feedbacks as you interact with ToDoList. The notification bar is designed to automatically hide after a specified duration to reduce distraction of expired feedbacks.

3.3.4 Command Line

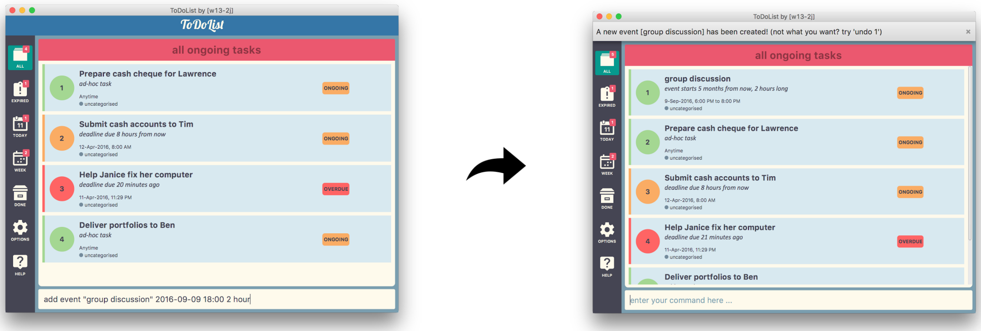
Command Line (see #4 in Figure 1) is where you will input your instructions to control and interact with *ToDoList*.

3.4 Basic Functions: For Beginners

After a quick introduction to the *ToDoList* interface, you are now ready to explore some of the basic functions:

3.4.1 Creating

There are 3 types of tasks that you may create, namely: **event**, **deadline** and **undated task**. As presented in Figure 3, to create a new item, use the command **add**.



```
add event "group discussion" 2016-09-09 18:00 2 hour
```

1 Prepare cash cheque for Lawrence — *ad-hoc task* — Anytime — uncategorised — ONGOING
2 Submit cash accounts to Tim — *deadline due 8 hours from now* — 12-Apr-2016, 8:00 AM — uncategorised — ONGOING
3 Help Janice fix her computer — *deadline due 20 minutes ago* — 11-Apr-2016, 11:29 PM — uncategorised — OVERDUE
4 Deliver portfolios to Ben — *ad-hoc task* — Anytime — uncategorised — ONGOING

A new event [group discussion] has been created! (not what you want? try 'undo 1')

1 group discussion — *event starts 5 months from now, 2 hours long* — 9-Sep-2016, 6:00 PM to 8:00 PM — uncategorised — ONGOING
2 Prepare cash cheque for Lawrence — *ad-hoc task* — Anytime — uncategorised — ONGOING
3 Submit cash accounts to Tim — *deadline due 8 hours from now* — 12-Apr-2016, 8:00 AM — uncategorised — ONGOING
4 Help Janice fix her computer — *deadline due 21 minutes ago* — 11-Apr-2016, 11:29 PM — uncategorised — OVERDUE
Deliver portfolios to Ben

enter your command here ...

(Figure 3: Created an event in ToDoList)

<u>**Creating an event**</u>
An **event** is a type of task that spans over a certain duration, with a start and end timing.
To add an event, you have to specify the `<TITLE>` of the event, the `<START-DATE>` in the format DD-MMM-YYYY, and the `<START-TIME>` in the format HH:MM. Acceptable `<TIME-UNIT>` fields include: day , hour , min.

<u>**Creating an deadline**</u>
A **deadline** is a type of task that has only the end timing.
To add a deadline, you will have to specify the `<TITLE>` of the deadline, the `<END-DATE>` in the format DD-MMM-YYYY, the `<END-TIME>` in the format HH:MM.

<u>**Creating an undated task**</u>
An **undated task** is a type of task that has no start or end timing. It is usually an ad-hoc task. To add a floating task, specify the `<TITLE>` of the floating task.

<u>The structure is as follows:</u>

```
add event <TITLE> <START-DATE> <START-TIME> <QUANTITY> <TIME-UNIT>
add deadline <TITLE> <END-DATE> <END-TIME>
add task <TITLE>
```

Example(s) :
```
add event “group discussion” 2016-10-24 15:00 3 hour
add deadline “submit-proposal” 2016-04-09 23:59
add task “buy-coffee-for-boss”
```

3.4.2 Editing

To edit a certain field of a task, use the command **edit** , followed by the `<TITLE>` of the task, the `<FIELD-NAME>` you would like to edit, and the `<NEW-VALUE>` to replace the old field value.

Valid`<FIELD-NAME>`are:

- TITLE , START-DATE , END-DATE , START-TIME , END-TIME , START (start date-time),
- END (end date-time), QUANTITY , TIME-UNIT , DURATION (quantity-timeunit),
- CATEGORY and REMINDER.

The structure is as follows:
```
edit <TITLE> <FIELD-NAME> <NEW VALUE>
```

Example(s):
```
edit “submit proposal” end 08:30
edit “submit proposal” category school
```

3.4.3 Deleting

To remove a certain task, use the command delete , followed by the <TITLE> (or index) of the task you wish to delete. Do note that deleted items will not be archived but instead will be removed from the application for good. Multiple tasks can be deleted at once.

The structure is as follows:
delete `<TITLE>` , ...

Example(s):
```
delete “my mistake”
```

NOTE:
In case you have accidentally deleted something important that you wish to recover, refer to our Undo and Redo features at **3.4.8 Undo and Redo**.

3.4.4 Archiving

Figure 4 presents to you how to mark a task in your agenda as completed: use the command **done** , followed by the `<TITLE>` (or index) of the item. Similarly for recovering an archived task, use the command **undone**, followed by the `<TITLE>` of the item.

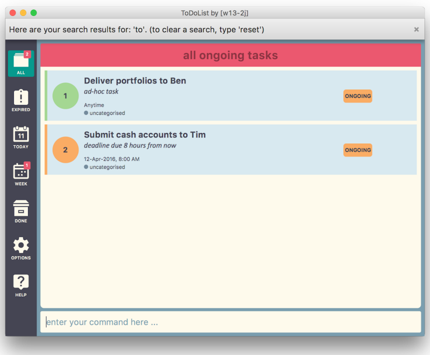
The structure is as follows:

```
done/undone <TITLE> , ...
```

Example(s):

```
done “buy coffee for boss”
```

3.4.5 Searching



ToDoList by [w13-2j]

Here are your search results for: 'to'. (to clear a search, type 'reset')

all ongoing tasks

1 Deliver portfolios to Ben
ad-hoc task
Anytime
● uncategorised
ONGOING

2 Submit cash accounts to Tim
deadline due 8 hours from now
12-Apr-2016, 8:00 AM
● uncategorised
ONGOING

enter your command here ...

(Figure 5: Searching for a task)

To look for a task in *ToDoList*, use the command **search** , followed by the `<TITLE>` of the item you are looking for, which is presented by Figure 5.

The structure is as follows:

```
search <TITLE>
```

Example(s):

```
search “group discussion”
```

3.4.6 Filtering

To view tasks belonging to a specific category, use the command **filter** , followed by a valid `<CATEGORY>`.

The structure is as follows:

```
filter <CATEGORY>
```

Example(s):

```
filter personal
```

3.4.7 Sorting

To sort the current list of tasks in your agenda automatically by a certain field, use the command **sort**, followed by the `<FIELD-NAME>` and the order `<ascending|descending>`:

Valid keywords for field name are:

- TITLE , START-DATE , END-DATE ,
- START-TIME , END-TIME , START (start date-time), END (end date-time),
- QUANTITY ,TIME-UNIT ,DURATION (QUANTITY-TIMEUNIT) ,
- CATEGORY and REMINDER.

The structure is as follows:

```
sort <FIELD-NAME> <ascending|descending>
```

Example(s):

```
sort end ascending
```

3.4.8 Undo and Redo

To undo several steps, simply type in command **undo**, followed by number of steps. Similarly to redo several steps, just type in command **redo** followed by number of steps.

The structure is as follows:

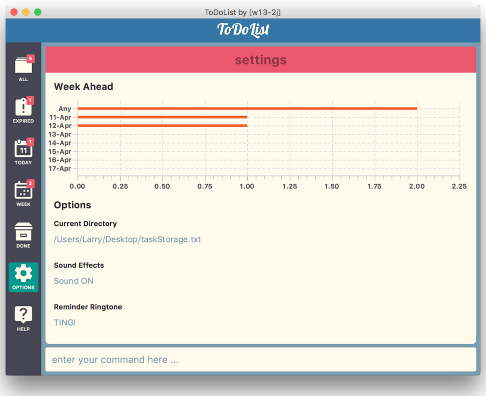
```
undo/redo <#STEPS>
```

Example(s):

```
undo 3
```

NOTE:
If an action is performed after undo, you will be unable to use **redo** to return to the previously undone state(s).

3.4.9 Locating your Data



settings

Week Ahead

Options

Current Directory
/Users/Larry/Desktop/taskStorage.txt

Sound Effects
Sound ON

Reminder Ringtone
TING!

enter your command here ...

(Figure 5: Searching for a task)

To view your data file, you can locate and open it at the same root directory where you have placed executable release (*.jar file). To change the default folder, simply use the command **save** , following with the field `<directory>`. (refer to Figure 5)

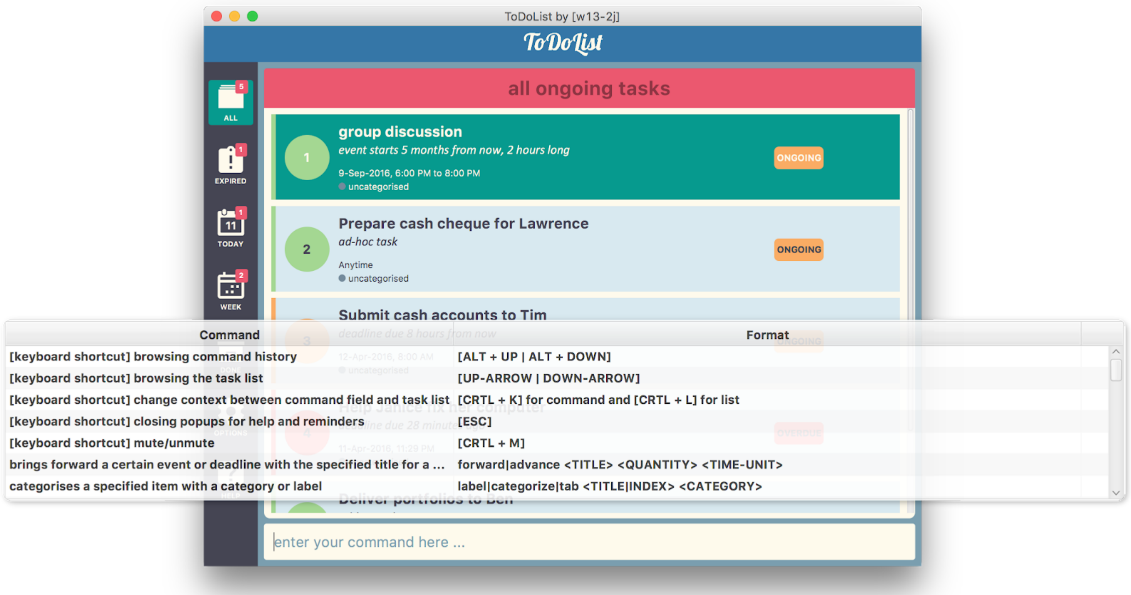
<u>The structure is as follows:</u>

```
save <directory>
```

<u>Example(s):</u>

```
save C:/Users/Larry/Documents/
```

3.4.10 Getting Help


ToDoList by [w13-2j]
ToDoList
all ongoing tasks
ALL
EXPIRED
TODAY
WEEK
group discussion
event starts 5 months from now, 2 hours long
9-Sep-2016, 6:00 PM to 8:00 PM
uncategorised
ONGOING
Prepare cash cheque for Lawrence
ad-hoc task
Anytime
uncategorised
ONGOING
Submit cash accounts to Tim
enter your command here ...

Command	Format
[keyboard shortcut] browsing command history	[ALT + UP | ALT + DOWN]
[keyboard shortcut] browsing the task list	[UP-ARROW | DOWN-ARROW]
[keyboard shortcut] change context between command field and task list	[CRTL + K] for command and [CRTL + L] for list
[keyboard shortcut] closing popups for help and reminders	[ESC]
[keyboard shortcut] mute/unmute	[CRTL + M]
brings forward a certain event or deadline with the specified title for a ...	forward|advance <TITLE> <QUANTITY> <TIME-UNIT>
categorises a specified item with a category or label	label|categorize|tab <TITLE|INDEX> <CATEGORY>

Should you be confused with the use of *ToDoList*, do not worry! Help is just a command away. Use the command **help** to trigger the help menu.

3.4.11 Exiting the Program

To terminate the application and close the application window, simply **click the close button on the window** or type **exit** in the command line.

3.5 Advanced Functions: For Expert Users

Very familiar with the basic functions already? Here are some additional features for a veteran like you!

3.5.1 Recurring Tasks

To set recurring events or deadlines, you may replace `<START-DATE>` or `<END-DATE>` in the add feature with `<every-ddd-...>` expression.

The structure is as follows:

```
add recurring deadline <INTERVAL> <TITLE> <END-DATE> <END_TIME>
add recurring event <INTERVAL> <TITLE> <START-DATE> <START_TIME>
<QUANTITY> <TIME_UNIT>
```

Example(s):

```
add recurring deadline 1-day “submit minutes” 2016-05-05 11:59
```

3.5.2 Natural Commands

3.5.2.1 Event Dates

Try entering special days and occasions by name instead of by date.

Example(s):

```
Valentines = 14-Feb-2016
```

3.5.2.2 Smart Time

Try specifying breakfast , lunch and dinner for time fields.

These are the assigned values:

```
breakfast = 9:00AM
lunch = 13:00PM
dinner = 7:00PM
```

3.5.3 Shortcuts

3.5.3.1 List Number as Title

You can address tasks by their list-index numbers in replacement for `<TITLE>` .

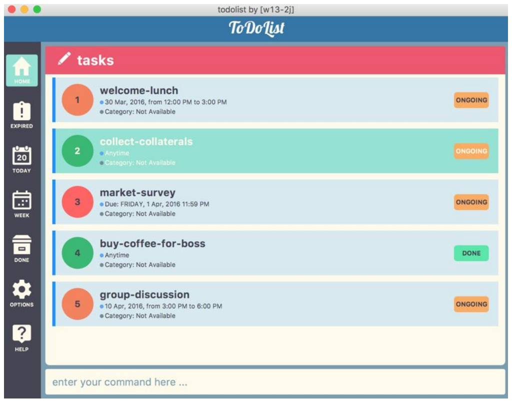
Example(s):

```
done 1, 2, 4
delete 3, 5
```

3.5.3.2 GUI Navigation

Use `UP-ARROW` and `DOWN-ARROW` to scroll and browse tasks in your current tasks list.

3.5.4 Day & Night Mode



1 welcome-lunch
30 Mar, 2016, from 12:00 PM to 3:00 PM
Category: Not Available

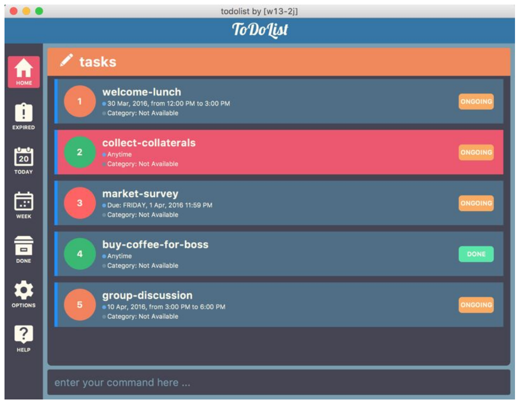
2 collect-collaterals
Anytime
Category: Not Available

3 market-survey
Due: FRIDAY, 1 Apr, 2016 11:59 PM
Category: Not Available

4 buy-coffee-for-boss
Anytime
Category: Not Available

5 group-discussion
10 Apr, 2016, from 3:00 PM to 6:00 PM
Category: Not Available

enter your command here ...



1 welcome-lunch
30 Mar, 2016, from 12:00 PM to 3:00 PM
Category: Not Available

2 collect-collaterals
Anytime
Category: Not Available

3 market-survey
Due: FRIDAY, 1 Apr, 2016 11:59 PM
Category: Not Available

4 buy-coffee-for-boss
Anytime
Category: Not Available

5 group-discussion
10 Apr, 2016, from 3:00 PM to 6:00 PM
Category: Not Available

enter your command here ...

(Figure 6: ToDoList in Day/Night Mode)

In *ToDoList*, we put a lot of emphasis on the comfort of using our application. To avoid straining your eyes under difficult lighting, you can switch between different themes to view your schedule more comfortably using the commands **day-mode** and **night-mode**.

3.5.5 Reminders

To set reminders when adding a task, use the **add-remind** command. It functions as add feature but with reminders set to trigger upon `<START-TIME>` or `<END-TIME>` of event and deadline respectively.

To set a reminder for an existing task, use the command **remind**. It sets a reminder for the event or deadline with the specified title to trigger upon `<START-TIME>` or `<END-TIME>` of event and deadline respectively.

To set a reminder for an existing task that triggers at a certain duration before the `<START-TIME>`, use the command **remind-bef**.

The structure is as follows:

```
add-remind
remind <TITLE>
remind-bef <TITLE> <QUANTITY> <TIME-UNIT>
```

Example(s):

```
add-remind event dinner 2016-09-09 19:00 1 hour
```

(will trigger a reminder for dinner 2016-09-09 at 7:00PM)

4 Developer Guide

4.1 Introduction: Contributing to ToDoList

Glad to have you on-board! At ToDoList, we welcome all creative and passionate contributors to experiment with our application. Most importantly, have fun and enjoy contributing to our project!

In this developer guide, you will first be introduced to the high-level architecture (section 4.2.1) of ToDoList, followed by a walkthrough of the internal structures and components (section 4.2.2 to section 4.2.5) in detail. After which, you will be provided a quick glance-through of our Application Program Interface (APIs) (section 4.3) to prepare you for contributing to ToDoList. Once you are ready, you can read implementation and testing (section 4.4) for instructions on how to start developing on ToDoList. Finally, You will be able to identify some of the existing issues and possible future work (section 4.5).

This developer guide assumes that you have some basic knowledge of Java, JavaFx and object-oriented programming. Should you be unfamiliar, it is never too late to learn!

Take a Java course at Udacity for free:
https://www.udacity.com/course/intro-to-java-programming--cs046

4.2 Under the Hood: Software Design Architecture

4.2.1 Architecture

The following diagram (Figure 1) presents the high-level design of the software.

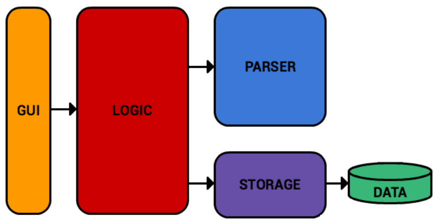


Figure 7.0: Architecture of ToDoList

ToDoList contains **4** major components: **Graphics User Interface (GUI)**, **Logic**, **Parser** and **Storage**. The following are the main roles of each component

- **GUI:** get user input through command line, display information to the user
- **Logic:** handle and process user commands
- **Parser(s)**: interprets the user commands and then pass back to **logic** component

- **Storage**: store, order and retrieve data

Packages Overview

The following is our architecture in greater details, organised into packages:

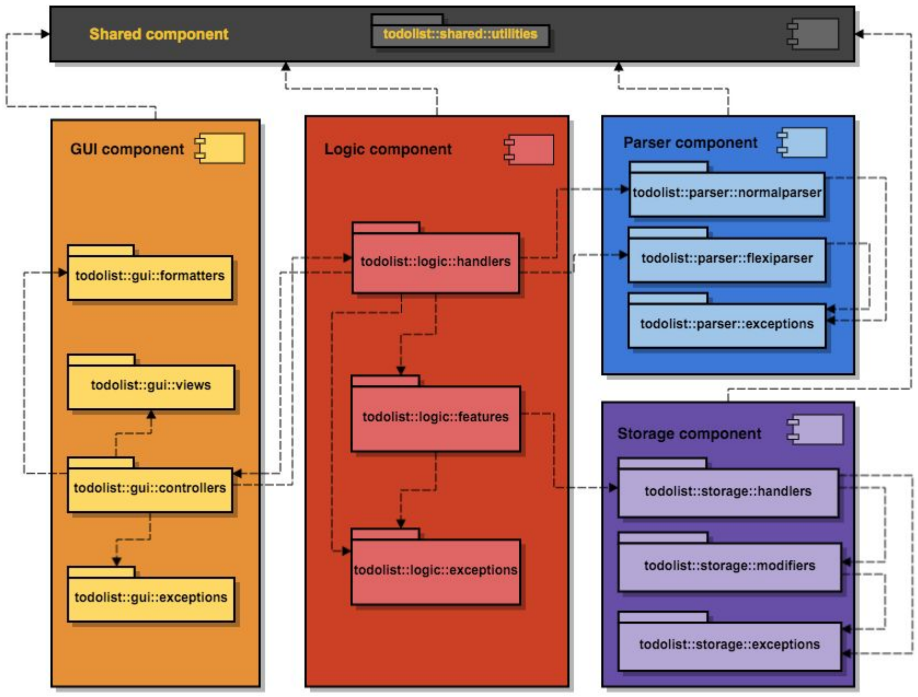

Shared component
todolist::shared::utilities
GUI component
todolist::gui::formatters
todolist::gui::views
todolist::gui::controllers
todolist::gui::exceptions
Logic component
todolist::logic::handlers
todolist::logic::features
todolist::logic::exceptions
Parser component
todolist::parser::normalparser
todolist::parser::flexiparser
todolist::parser::exceptions
Storage component
todolist::storage::handlers
todolist::storage::modifiers
todolist::storage::exceptions

Figure 8.0: Package Diagram of ToDoList

Sequence Overview

The following Figure 9.0, a sequence diagram, illustrating the call of functions in a typical usage scenario:

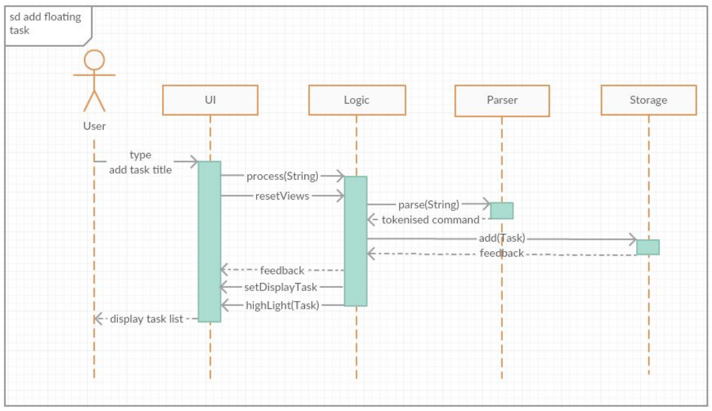


Figure 9.0: Package Diagram of ToDoList

Overarching Principle: Single Responsibility

Every class in each component has the responsibility over one part of the functionality provided by the component. By observing this principle faithfully, we are able to tidy up

ToDoList and reduces conflicts and coupling between classes, improving cohesiveness of the program.

4.2.2 GUI Component

The Graphical User Interface (GUI) component consists mainly of views and controllers, managed by a **MainApp** class. This allows us to modify and change displays according to user input and logic output. Figure 10 below is a UML class diagram illustrating the internal constructs of the **GUI** component:

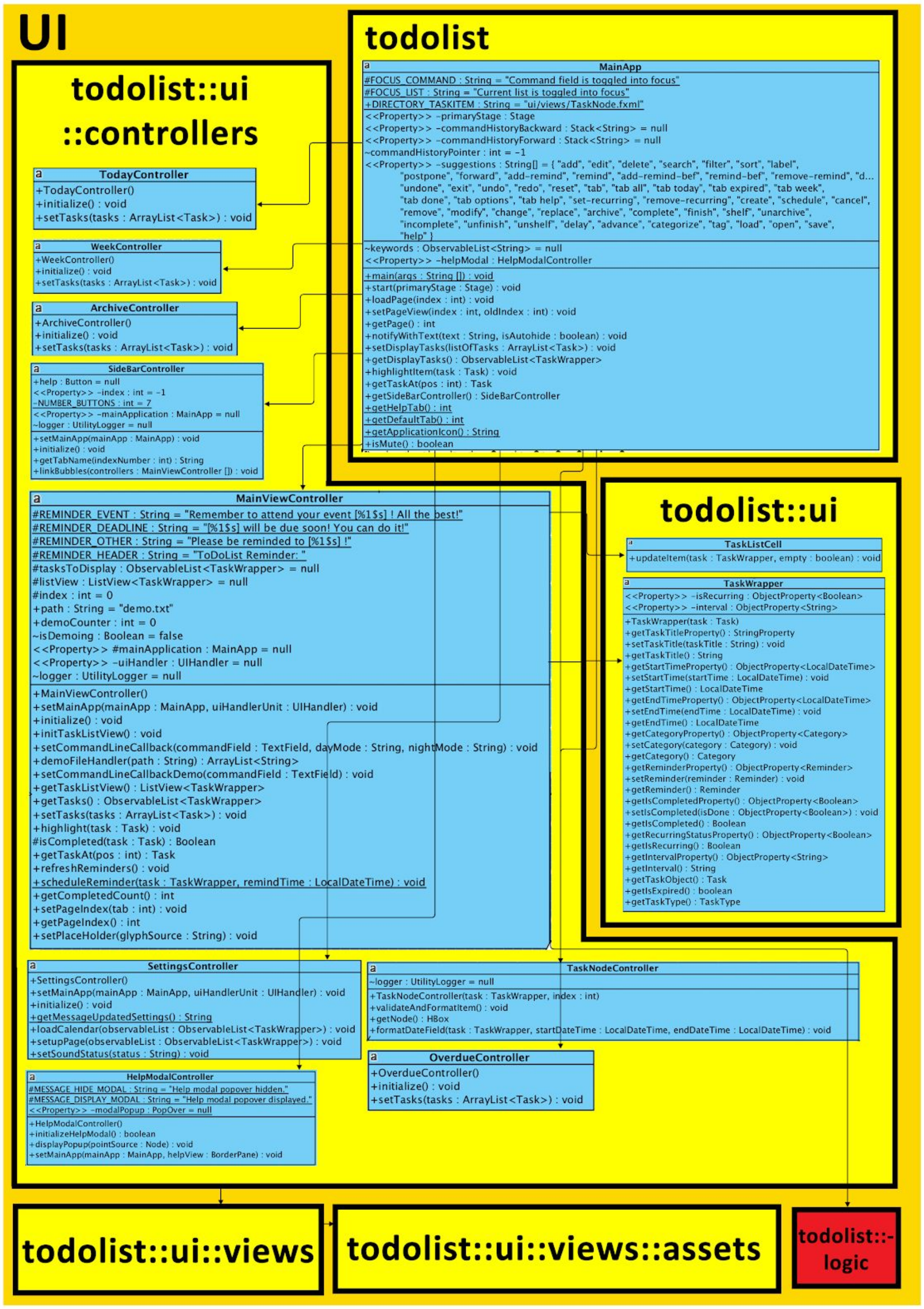

UI
todolist
MainApp
#FOCUS_COMMAND : String = "Command field is toggled into focus"
#FOCUS_LIST : String = "Current list is toggled into focus"
+DIRECTORY_TASKITEM : String = "ui/views/TaskNode.fxml"
<<Property>> -primaryStage : Stage
<<Property>> -commandHistoryBackward : Stack<String> = null
<<Property>> -commandHistoryForward : Stack<String> = null
~commandHistoryPointer : int = -1
~keywords : ObservableList<String> = null
<<Property>> -helpModal : HelpModalController
+main(args : String []) : void
+start(primaryStage : Stage) : void
+loadPage(index : int) : void
+setPageView(index : int, oldIndex : int) : void
+getPage() : int
+notifyWithText(text : String, isAutohide : boolean) : void
+setDisplayTasks(listOfTasks : ArrayList<Task>) : void
+getDisplayTasks() : ObservableList<TaskWrapper>
+highlightItem(task : Task) : void
+getTaskAt(pos : int) : Task
+getSideBarController() : SideBarController
+getHelpTab() : int
+getDefaultTab() : int
+getApplicationIcon() : String
+isMute() : boolean
todolist::ui ::controllers
TodayController
WeekController
ArchiveController
SideBarController
MainViewController
SettingsController
HelpModalController
TaskNodeController
OverdueController
todolist::ui
TaskListCell
TaskWrapper
todolist::ui::views
todolist::ui::views::assets
todolist::-logic

Figure 10: UML Class Diagram for GUI Component

GUI Pattern: Model-View Controller

In **GUI**, we follow the model-view controller pattern closely. This is observable by analysing how the **MainApp** functions. For every view, **MainApp** loads its controller and initialize the controller's' **logic**. This allows the controller of a particular view to receive user input from the view, send the input to logic to modify the model, and update the view with processed logic output. This pattern reduces dependencies and decouples view from the model or data.

4.2.3 Logic Component

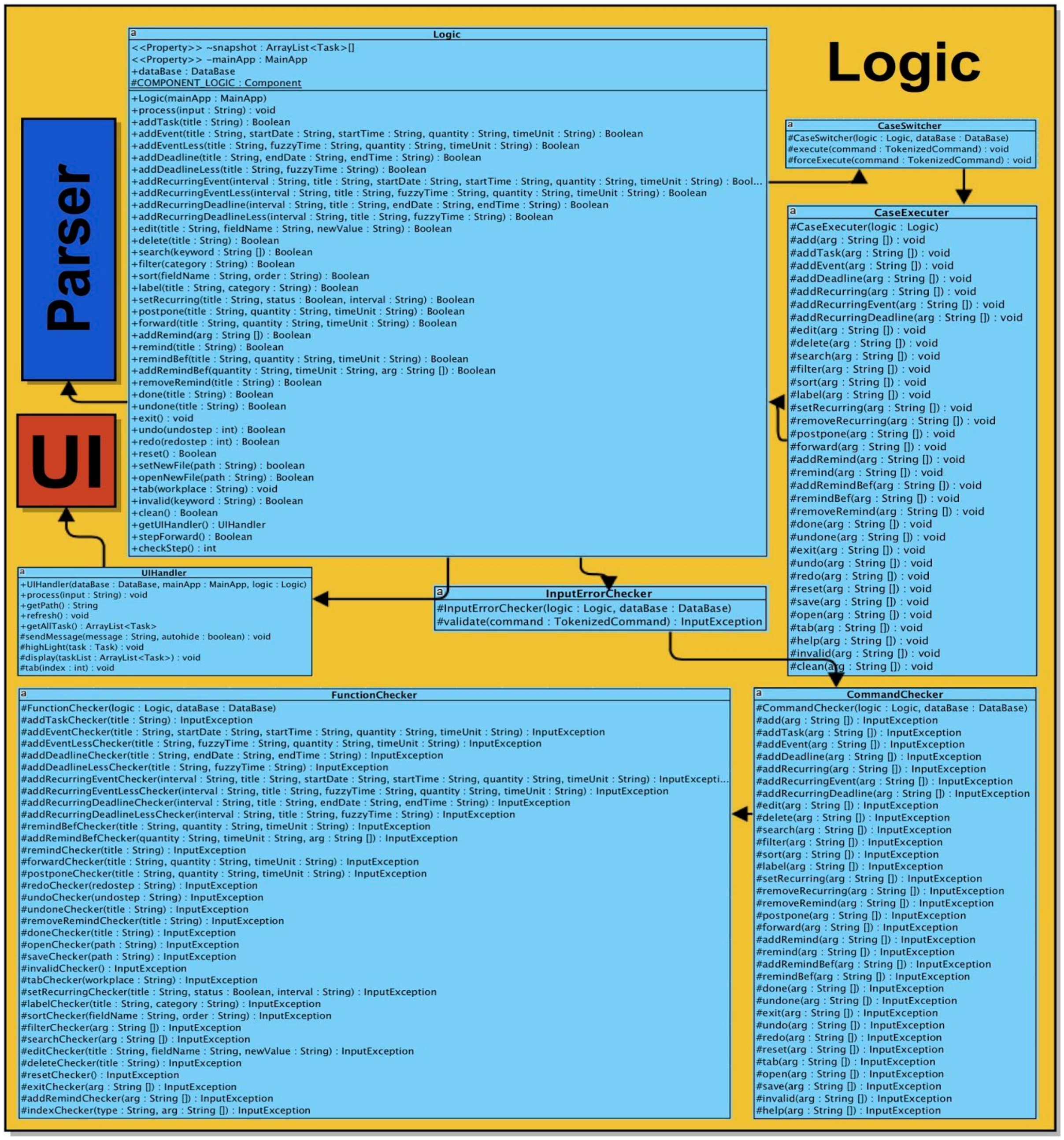

Logic
Parser
UI
Logic
<<Property>> ~snapshot : ArrayList<Task>[]
<<Property>> ~mainApp : MainApp
+dataBase : DataBase
#COMPONENT_LOGIC : Component
+Logic(mainApp : MainApp)
+process(input : String) : void
+addTask(title : String) : Boolean
+addEvent(title : String, startDate : String, startTime : String, quantity : String, timeUnit : String) : Boolean
+addEventLess(title : String, fuzzyTime : String, quantity : String, timeUnit : String) : Boolean
+addDeadline(title : String, endDate : String, endTime : String) : Boolean
+addDeadlineLess(title : String, fuzzyTime : String) : Boolean
+addRecurringEvent(interval : String, title : String, startDate : String, startTime : String, quantity : String, timeUnit : String) : Bool...
+addRecurringEventLess(interval : String, title : String, fuzzyTime : String, quantity : String, timeUnit : String) : Boolean
+addRecurringDeadline(interval : String, title : String, endDate : String, endTime : String) : Boolean
+addRecurringDeadlineLess(interval : String, title : String, fuzzyTime : String) : Boolean
+edit(title : String, fieldName : String, newValue : String) : Boolean
+delete(title : String) : Boolean
+search(keyword : String []) : Boolean
+filter(category : String) : Boolean
+sort(fieldName : String, order : String) : Boolean
+label(title : String, category : String) : Boolean
+setRecurring(title : String, status : Boolean, interval : String) : Boolean
+postpone(title : String, quantity : String, timeUnit : String) : Boolean
+forward(title : String, quantity : String, timeUnit : String) : Boolean
+addRemind(arg : String []) : Boolean
+remind(title : String) : Boolean
+remindBef(title : String, quantity : String, timeUnit : String) : Boolean
+addRemindBef(quantity : String, timeUnit : String, arg : String []) : Boolean
+removeRemind(title : String) : Boolean
+done(title : String) : Boolean
+undone(title : String) : Boolean
+exit() : void
+undo(undostep : int) : Boolean
+redo(redostep : int) : Boolean
+reset() : Boolean
+setNewFile(path : String) : boolean
+openNewFile(path : String) : Boolean
+tab(workplace : String) : void
+invalid(keyword : String) : Boolean
+clean() : Boolean
+getUIHandler() : UIHandler
+stepForward() : Boolean
+checkStep() : int
CaseSwitcher
#CaseSwitcher(logic : Logic, dataBase : DataBase)
#execute(command : TokenizedCommand) : void
#forceExecute(command : TokenizedCommand) : void
CaseExecuter
#CaseExecuter(logic : Logic)
#add(arg : String []) : void
#addTask(arg : String []) : void
#addEvent(arg : String []) : void
#addDeadline(arg : String []) : void
#addRecurring(arg : String []) : void
#addRecurringEvent(arg : String []) : void
#addRecurringDeadline(arg : String []) : void
#edit(arg : String []) : void
#delete(arg : String []) : void
#search(arg : String []) : void
#filter(arg : String []) : void
#sort(arg : String []) : void
#label(arg : String []) : void
#setRecurring(arg : String []) : void
#removeRecurring(arg : String []) : void
#postpone(arg : String []) : void
#forward(arg : String []) : void
#addRemind(arg : String []) : void
#remind(arg : String []) : void
#addRemindBef(arg : String []) : void
#remindBef(arg : String []) : void
#removeRemind(arg : String []) : void
#done(arg : String []) : void
#undone(arg : String []) : void
#exit(arg : String []) : void
#undo(arg : String []) : void
#redo(arg : String []) : void
#reset(arg : String []) : void
#save(arg : String []) : void
#open(arg : String []) : void
#tab(arg : String []) : void
#help(arg : String []) : void
#invalid(arg : String []) : void
#clean(arg : String []) : void
UIHandler
+UIHandler(dataBase : DataBase, mainApp : MainApp, logic : Logic)
+process(input : String) : void
+getPath() : String
+refresh() : void
+getAllTask() : ArrayList<Task>
#sendMessage(message : String, autohide : boolean) : void
#highLight(task : Task) : void
#display(taskList : ArrayList<Task>) : void
#tab(index : int) : void
InputErrorChecker
#InputErrorChecker(logic : Logic, dataBase : DataBase)
#validate(command : TokenizedCommand) : InputException
FunctionChecker
#FunctionChecker(logic : Logic, dataBase : DataBase)
#addTaskChecker(title : String) : InputException
#addEventChecker(title : String, startDate : String, startTime : String, quantity : String, timeUnit : String) : InputException
#addEventLessChecker(title : String, fuzzyTime : String, quantity : String, timeUnit : String) : InputException
#addDeadlineChecker(title : String, endDate : String, endTime : String) : InputException
#addDeadlineLessChecker(title : String, fuzzyTime : String) : InputException
#addRecurringEventChecker(interval : String, title : String, startDate : String, startTime : String, quantity : String, timeUnit : String) : InputExcepti...
#addRecurringEventLessChecker(interval : String, title : String, fuzzyTime : String, quantity : String, timeUnit : String) : InputException
#addRecurringDeadlineChecker(interval : String, title : String, endDate : String, endTime : String) : InputException
#addRecurringDeadlineLessChecker(interval : String, title : String, fuzzyTime : String) : InputException
#remindBefChecker(title : String, quantity : String, timeUnit : String) : InputException
#addRemindBefChecker(quantity : String, timeUnit : String, arg : String []) : InputException
#remindChecker(title : String) : InputException
#forwardChecker(title : String, quantity : String, timeUnit : String) : InputException
#postponeChecker(title : String, quantity : String, timeUnit : String) : InputException
#redoChecker(redostep : String) : InputException
#undoChecker(undostep : String) : InputException
#undoneChecker(title : String) : InputException
#removeRemindChecker(title : String) : InputException
#doneChecker(title : String) : InputException
#openChecker(path : String) : InputException
#saveChecker(path : String) : InputException
#invalidChecker() : InputException
#tabChecker(workplace : String) : InputException
#setRecurringChecker(title : String, status : Boolean, interval : String) : InputException
#labelChecker(title : String, category : String) : InputException
#sortChecker(fieldName : String, order : String) : InputException
#filterChecker(arg : String []) : InputException
#searchChecker(arg : String []) : InputException
#editChecker(title : String, fieldName : String, newValue : String) : InputException
#deleteChecker(title : String) : InputException
#resetChecker() : InputException
#exitChecker(arg : String []) : InputException
#addRemindChecker(arg : String []) : InputException
#indexChecker(type : String, arg : String []) : InputException
CommandChecker
#CommandChecker(logic : Logic, dataBase : DataBase)
#add(arg : String []) : InputException
#addTask(arg : String []) : InputException
#addEvent(arg : String []) : InputException
#addDeadline(arg : String []) : InputException
#addRecurring(arg : String []) : InputException
#addRecurringEvent(arg : String []) : InputException
#addRecurringDeadline(arg : String []) : InputException
#edit(arg : String []) : InputException
#delete(arg : String []) : InputException
#search(arg : String []) : InputException
#filter(arg : String []) : InputException
#sort(arg : String []) : InputException
#label(arg : String []) : InputException
#setRecurring(arg : String []) : InputException
#removeRecurring(arg : String []) : InputException
#removeRemind(arg : String []) : InputException
#postpone(arg : String []) : InputException
#forward(arg : String []) : InputException
#addRemind(arg : String []) : InputException
#remind(arg : String []) : InputException
#addRemindBef(arg : String []) : InputException
#remindBef(arg : String []) : InputException
#done(arg : String []) : InputException
#undone(arg : String []) : InputException
#exit(arg : String []) : InputException
#undo(arg : String []) : InputException
#redo(arg : String []) : InputException
#reset(arg : String []) : InputException
#tab(arg : String []) : InputException
#open(arg : String []) : InputException
#save(arg : String []) : InputException
#invalid(arg : String []) : InputException
#help(arg : String []) : InputException

Figure 11: UML Class Diagram for Logic Component

Logic Pattern: Command

The ***Logic*** class acts as a command centre. By obtaining a command line from GUI, it sends it for parsing; and receives a tokenized command object. This object encapsulates the user requests, and is thrown to the ***CaseSwitcher*** class to call the function that the user is requesting for. The Command pattern observed by the ***Logic*** class allows us to process parameterized requests, or specifically, handle requests of different parameters and formats.

4.2.4 Parser Component

The **parser** component consists of 2 classes (see Figure 12):

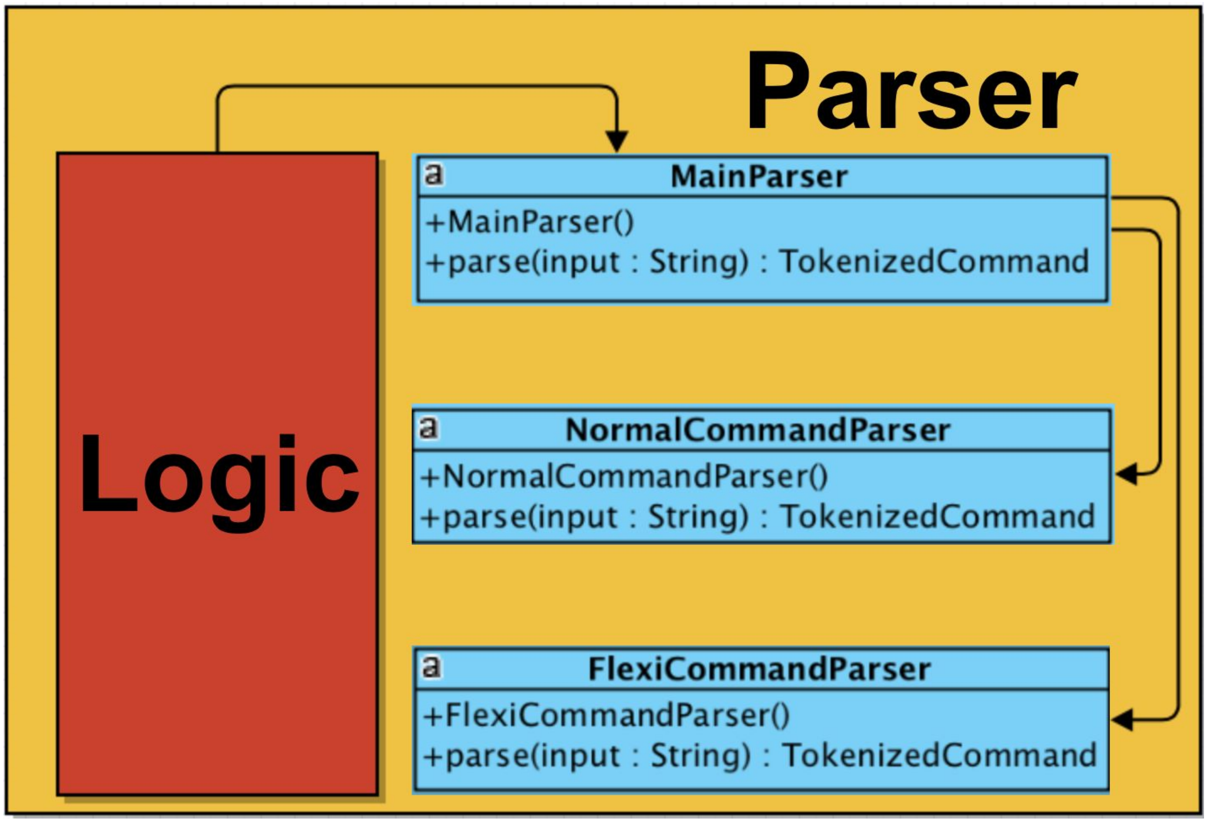

Parser
Logic
a MainParser
+MainParser()
+parse(input : String) : TokenizedCommand
a NormalCommandParser
+NormalCommandParser()
+parse(input : String) : TokenizedCommand
a FlexiCommandParser
+FlexiCommandParser()
+parse(input : String) : TokenizedCommand

Figure 12: UML Class Diagram for Parser Component

The following is the pattern and principle involved in the design.

Parser Pattern: Facade
The ***Parser*** class provides a simple interface to the logic component, including an API "parse(String)". The ***Parser*** class will then make use of other classes in parser component, which are ***NormalCommandHandler*** and ***FlexiCommandHandler*** to parse the string from the **logic** component. This design makes it clearer for the **logic** component to understand and make use of the functionality of the **parser** component. At the same time, it also reduces the dependency of the **logic** component on the internal workings of the **parser** component.

4.2.5 Storage Component

The **storage** component consists of 4 classes (see Figure 13):

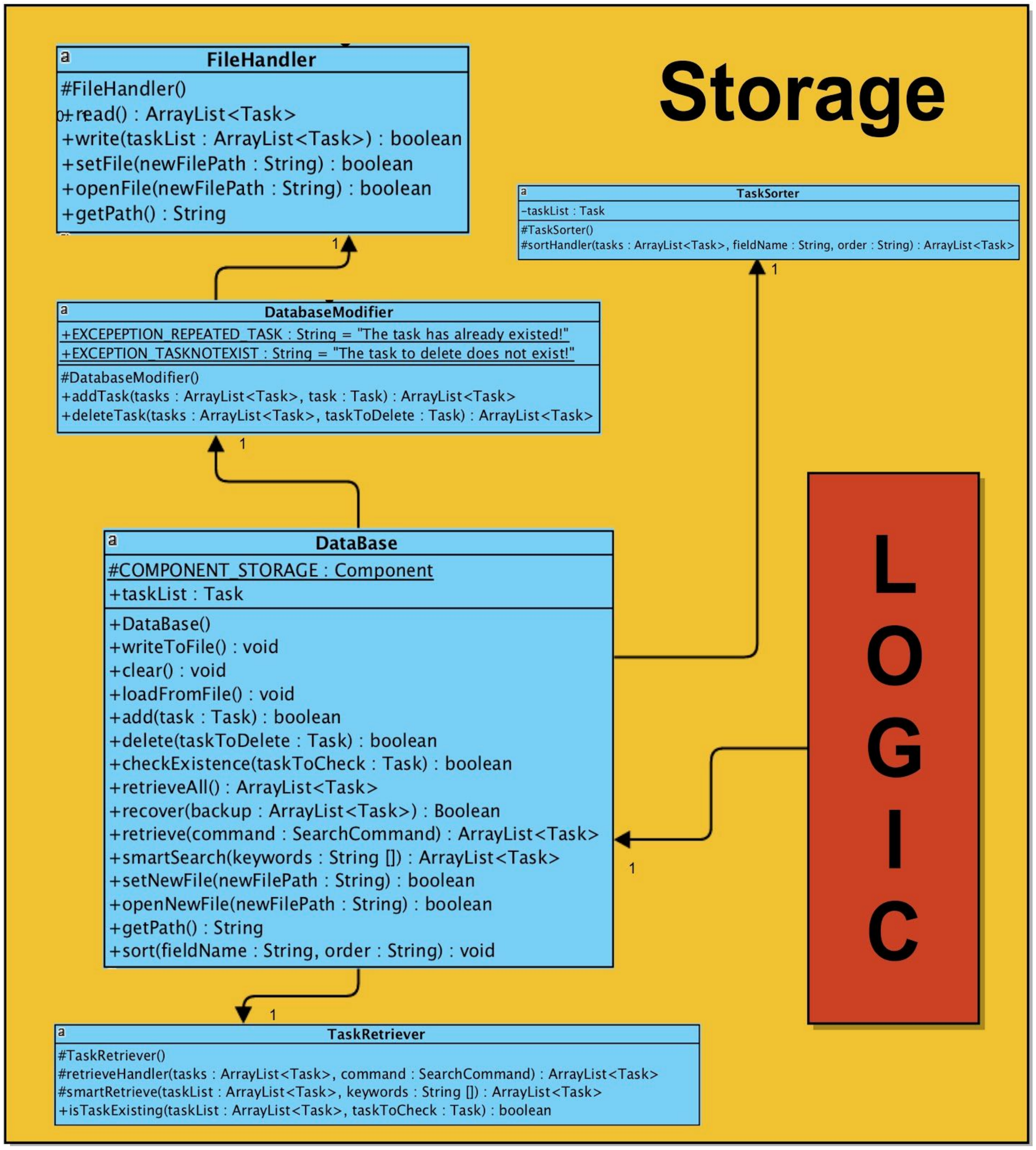

Storage
FileHandler
#FileHandler()
+read() : ArrayList<Task>
+write(taskList : ArrayList<Task>) : boolean
+setFile(newFilePath : String) : boolean
+openFile(newFilePath : String) : boolean
+getPath() : String
TaskSorter
-taskList : Task
#TaskSorter()
#sortHandler(tasks : ArrayList<Task>, fieldName : String, order : String) : ArrayList<Task>
DatabaseModifier
+EXCEPEPTION_REPEATED_TASK : String = "The task has already existed!"
+EXCEPTION_TASKNOTEXIST : String = "The task to delete does not exist!"
#DatabaseModifier()
+addTask(tasks : ArrayList<Task>, task : Task) : ArrayList<Task>
+deleteTask(tasks : ArrayList<Task>, taskToDelete : Task) : ArrayList<Task>
DataBase
#COMPONENT_STORAGE : Component
+taskList : Task
+DataBase()
+writeToFile() : void
+clear() : void
+loadFromFile() : void
+add(task : Task) : boolean
+delete(taskToDelete : Task) : boolean
+checkExistence(taskToCheck : Task) : boolean
+retrieveAll() : ArrayList<Task>
+recover(backup : ArrayList<Task>) : Boolean
+retrieve(command : SearchCommand) : ArrayList<Task>
+smartSearch(keywords : String []) : ArrayList<Task>
+setNewFile(newFilePath : String) : boolean
+openNewFile(newFilePath : String) : boolean
+getPath() : String
+sort(fieldName : String, order : String) : void
LOGIC
TaskRetriever
#TaskRetriever()
#retrieveHandler(tasks : ArrayList<Task>, command : SearchCommand) : ArrayList<Task>
#smartRetrieve(taskList : ArrayList<Task>, keywords : String []) : ArrayList<Task>
+isTaskExisting(taskList : ArrayList<Task>, taskToCheck : Task) : boolean
1
1
1
1
1

Figure 13: UML Class Diagram for Storage Component

The following is the pattern and principle involved in the design:

Storage Pattern: Facade

The ***DatabaseHandler*** class provides a simple interface to the **logic** component, including APIs such as ***add*** and ***delete***. The ***DatabaseHandler*** will then make use of other classes in the **storage** component, such as ***TaskRetriever***, ***TaskSorter*** and ***FileHandler***, to handle the command from the **logic** component. This design makes it clearer for the **logic** component to understand and make use of the functionality of the **storage** component. At the same time, it also reduces the dependency of the **logic** component on the internal workings of the **storage** component.

4.3 What You Are Here For: Application Program Interface

In this section, you will be introduced to the important Application Programming Interfaces (APIs) of each individual components, which you may utilise in your development.

4.3.1 GUI: User Experience and Designer

If you are a **GUI** designer or programmer, you may find the following method calls useful:

Modifier and Type	Method and Description
todolist.MainApp	
static **String**	**getApplicationIcon**() getApplicationIcon returns the path for the application icon.
Stack<**String**>	**getCommandHistoryBackward**() getCommandHistoryBackward returns the stack of command history saved before the current visited entry.
Stack<**String**>	**getCommandHistoryForward**() getCommandHistoryForward returns the stack of command history saved after the current visited entry.
static int	**getDefaultTab**() getDefaultTab returns the index referring to the default tab.
javafx.collections.ObservableList<**TaskWrapper**>	**getDisplayTasks()** getDisplayTask returns the current displayed list of wrapped task.
HelpModalController	**getHelpModal()** getHelpModal returns a HelpModalController that controls the logic for help display.
static int	**getHelpTab()** getHelpTab returns the tab number that refers to the help tab.
int	**getPage()** getPage returns the current page index.
javafx.stage.Stage	**getPrimaryStage()** getPrimaryStage returns the current stage that is hosting the display.
SideBarController	**getSideBarController()** getSideBarController returns the current sideBarController that controls the logic of the sidebar.

static String[]	**getSuggestions()** getSuggestions returns the array of suggestions that displays for autocomplete.
Task	**getTaskAt(int pos)** getTaskAt returns a task given its position in the current displayed list, or a null if such as task does not exist.
void	**highlightItem(Task task)** highlightItem sets the corresponding task item on focus.
boolean	**isMute()** isMute returns the status of the sound settings.
void	**loadPage(int index)** loadPage sets the current page index to the given index.
static void	**main(String[] args)** Starts the application with launch() command.
void	**notifyWithText(String text, boolean isAutohide)** notifyWithText triggers a notification with or without autohide.
void	**setDisplayTasks(ArrayList<Task> listOfTasks)** setDisplayTasks replaces and overwrites the current list of tasks to display.
void	**setHelpModal(HelpModalController helpModal)** setHelpModal sets the given helpModalController as the helpModalController for the help table.
void	**setPageView(int index, int oldIndex)** setPageView loads the corresponding page into the main display area.
void	**setPrimaryStage(javafx.stage.Stage primaryStage)** setPrimaryStage sets the current primaryStage as the given stage.
void	**start(javafx.stage.Stage primaryStage)** **(non-Javadoc)**
todolist.ui.controllers.SideBarController	
int	**getIndex**()
MainApp	**getMainApplication()**
String	**getTabName(int indexNumber)**
void	**initialize()**
void	**linkBubbles(MainViewController[] controllers)**

void	**setIndex(int index)**
void	**setMainApp(MainApp mainApp)** **CORE FUNCTIONS**
todolist.ui.controllers.MainViewController	
ArrayList<**String**>	**demoFileHandler**(**String** path) demoFileHandler is used in demo to manipulate demo data and files.
int	**getCompletedCount**() getCompletedCount returns the number of completed tasks in the current display list.
MainApp	**getMainApplication**() getMainApplication returns the link-back to the MainApp
int	**getPageIndex()** getPageIndex retrieves the page index of the current page.
Task	**getTaskAt(int pos)** getTaskAt returns the task at the position specified in the list view, or null if it is not found.
javafx.scene.control.ListView<**TaskWrapper**>	**getTaskListView()** getTaskListView returns the current list view of tasks.
javafx.collections.ObservableList<**TaskWrapper**>	**getTasks()** getTasks returns the list of tasks stored.
UIHandler	**getUiHandler()** getUiHandler returns a UIHandler reference.
void	**highlight(Task task)** highlight gets the index of the task to highlight and put that index item on focus.
void	**initialize()** initialize prepares the task list.
void	**initTaskListView()** initTaskListView initializes the task list by setting the list properties and default format.
void	**refreshReminders()** refreshReminders updates all the reminders to be triggered.
static void	**scheduleReminder(TaskWrapper task, LocalDateTime remindTime)**

	scheduleReminder sets the task reminder to trigger at the given timing.
void	**setCommandLineCallback(javafx.scene.control.TextField commandField,String dayMode, String nightMode)** setCommandLineCallback takes in a textfield and sets a function as the action callback function.
void	**setCommandLineCallbackDemo(javafx.scene.control.TextField commandField)** **setCommandLineCallbackDemo sets the command callback for demonstration.**
void	**setMainApp(MainApp mainApp, UIHandler uiHandlerUnit)** **setMainApp takes a MainApp reference and stores it locally as the mainApplication reference.**
void	**setMainApplication(MainApp mainApplication)** **setMainApplication takes in the a link to the main application.**
void	**setPageIndex(int tab)** **setPageIndex takes in the tab index and sets the page with the index as the current page.**
void	**setPlaceHolder(String glyphSource)** **setPlaceHolder takes in a glyphSource directory and sets the glyph as the placeholder for empty list view(s).**
void	**setTasks(ArrayList<Task> tasks)** **setTasks takes a list of tasks and sets it as the list of tasks to display by associating with the list view.**
void	**setUiHandler(UIHandler uiHandler)** **setUiHandler takes in the link to a UI handler.**
todolist.ui.controllers.TaskNodeController	
void	**formatDateField**(**TaskWrapper** task,**LocalDateTime** startDateTime,**LocalDateTime** endDateTime) formatDateField takes in a task, start date-time, end date-time to format date range field for display.
javafx.scene.layout.HBox	**getNode**() getNode returns the formatted task view for display.
void	**validateAndFormatItem()** **validateAndFormatItem checks the validity of the display content and formats it for display.**
todolist.ui.controllers.ArchiveController	

void	**initialize()** **(non-Javadoc)**
void	**setTasks(ArrayList<Task> tasks)** **(non-Javadoc)**
todolist.ui.controllers.HelpModalController	
void	**displayPopup(javafx.scene.Node pointSource)** displayPopup shows the help table, given a source node to display as the table.
org.controlsfx.control.PopOver	**getModalPopup()** getModalPopup returns a PopOver to access the help table popup.
boolean	**initializeHelpModal()** initializeHelpModal initialises the PopOver for displaying the help table.
void	**setMainApp(MainApp mainApp, javafx.scene.layout.BorderPane helpView)** Is called by the main application to give a reference back to itself.
todolist.ui.controllers.OverdueController	
void	**displayPopup(javafx.scene.Node pointSource)** displayPopup shows the help table, given a source node to display as the table.
org.controlsfx.control.PopOver	**getModalPopup()** getModalPopup returns a PopOver to access the help table popup.
boolean	**initializeHelpModal()** initializeHelpModal initialises the PopOver for displaying the help table.
void	**setMainApp(MainApp mainApp, javafx.scene.layout.BorderPane helpView)** Is called by the main application to give a reference back to itself.
todolist.ui.controllers.SettingsController	
static String	**getMessageUpdatedSettings()** getMessageUpdatedSettings returns the standard output message for display when the settings view gets updated.
void	**initialize()** **(non-Javadoc)**
void	**loadCalendar(javafx.collections.ObservableList<TaskWrapper> observableList)** Deprecated.
void	**setMainApp(MainApp mainApp, UIHandler uiHandlerUnit)** setMainApp takes a MainApp reference and stores it locally as the mainApplication

	reference.
void	**setSoundStatus(String status)** setSoundStatus takes in a status string descriptor and sets the sound status display text accordingly.
void	**setupPage(ArrayList<Task> tasks)** setupPage takes in an array list of tasks and filters for a neat weekly summary.
todolist.ui.controllers.TodayController	
void	**initialize()** **(non-Javadoc)**
void	**setTasks(ArrayList<Task> tasks)** **(non-Javadoc)**
todolist.ui.controllers.WeekController	
void	**initialize()** **(non-Javadoc)**
void	**setTasks(ArrayList<Task> tasks)** **(non-Javadoc)**
todolist.ui.formatters.DateTimeFormatter (Deprecated)	

(Table 3: GUI API)

4.3.2 Logic: Back-End Utilities Manager

The **logic** component provides one API for the **GUI** component to call. The following table (Table 3) presents the **APIs** offered by the **logic** component. You can simply call ***Logic*** class if you want to call these methods.

Modifier and Type	**Method and Description**
todolist.logic.Logic	
Boolean	**addDeadline(String title, String endDate, String endTime)**
Boolean	**addDeadlineLess(String title, String fuzzyTime)** This method adds a new deadline.(less argument and fuzzy time)
Boolean	**addEvent(String title, String startDate, String startTime, String quantity, String timeUnit)** This method adds a new event with start date and duration

Boolean	**addEventLess(String title, String fuzzyTime, String quantity, String timeUnit)** This method adds a new event with start date and duration(less argument and fuzzy time)
Boolean	**addRecurringDeadline(String interval, String title, String endDate, String endTime)** This method adds an recurring deadline
Boolean	**addRecurringDeadlineLess(String interval, String title, String fuzzyTime)** This method adds an recurring deadline.(less argument and fuzzy time)
Boolean	**addRecurringEvent(String interval, String title, String startDate, String startTime, String quantity, String timeUnit)** This method adds an recurring event
Boolean	**addRecurringEventLess(String interval, String title, String fuzzyTime, String quantity, String timeUnit)** This method adds an recurring event.(less argument and fuzzy time)
Boolean	**addRemind(String[] arg)** This method adds a task with remind and triggers the remind at the deadline.
Boolean	**addRemindBef(String quantity, String timeUnit, String[] arg)** This method adds a task with remind and triggers the remind a duration before the deadline.
Boolean	**addTask(String title)** This method adds a new floating task.
int	**checkStep()** This method returns the current step number
Boolean	**clean()** This method clears all the tasks
Boolean	**delete(String title)** This method takes in the title of a task and deletes it.
Boolean	**done(String title)** This method takes in the title of a task and marks it as done.
Boolean	**edit(String title, String fieldName, String newValue)** This method edits a task.

void	**exit()** This method terminates the application.
Boolean	**filter(String category)** This method takes in the name of a category and displays tasks of that category.
Boolean	**forward(String title, String quantity, String timeUnit)** This method forwards a task by a duration.
MainApp	**getMainApp()** This method returns the current main app
ArrayList<Task>[]	**getSnapshot()** This method returns the snapshot array
UIHandler	**getUIHandler()** This method returns the current UIHandler
Boolean	**invalid(String keyword)** This method call UIHandler to display a message when flexi command cannot be parsed
Boolean	**label(String title, String category)** This method takes in the title of a task and labels it with a category.
Boolean	**openNewFile(String path)** This method opens a file from a given new file path
Boolean	**postpone(String title, String quantity, String timeUnit)** This method postpones a task by a duration
void	**process(String input)** This method takes in raw user input and process it by calling parser
Boolean	**redo(int redostep)** This method takes in an integer and redo that number of steps.
Boolean	**remind(String title)** This method adds remind to an existing task and triggers the remind at the deadline.
Boolean	**remindBef(String title, String quantity, String timeUnit)** This method adds remind to an existing task and triggers the remind a duration before the deadline.

Boolean	**removeRemind(String title)** This method takes in the title of a task and removes its reminder.
Boolean	**reset()** This method resets the view
Boolean	**search(String[] keyword)** This method takes in the title of a task and displays it.
void	**setMainApp(MainApp mainApp)** This method sets the main app
boolean	**setNewFile(String path)** This method save all the tasks to a new file path
Boolean	**setRecurring(String title, Boolean status, String interval)** This method edits the recurring status of a task.
Boolean	**sort(String fieldName, String order)** This method sorts all tasks in according to the field name and order.
Boolean	**stepForward()** This method increases the internal step counter and takes a snapshot
void	**tab(String workplace)** This method call UIHandler to change tab
Boolean	**undo(int undostep)** This method takes in an integer and undo that number of steps.
Boolean	**undone(String title)** This method takes in the title of a task and marks it as undone.

(Table 4: Logic API)

4.3.3 Parser: Command and Language Interpreter

Modifier and Type	Method and Description
todolist.parser.MainParser	
TokenizedCommand	**parse(String** input) This method takes in a string and parse it.

(Table 5: Parser API)

4.3.4 Storage: Data and File Handler

If you are the developer for the **logic** component, you may make use of the APIs provided by the **storage** component. Table 6 below represents the list of APIs of the **storage** component, which you can call through database class.

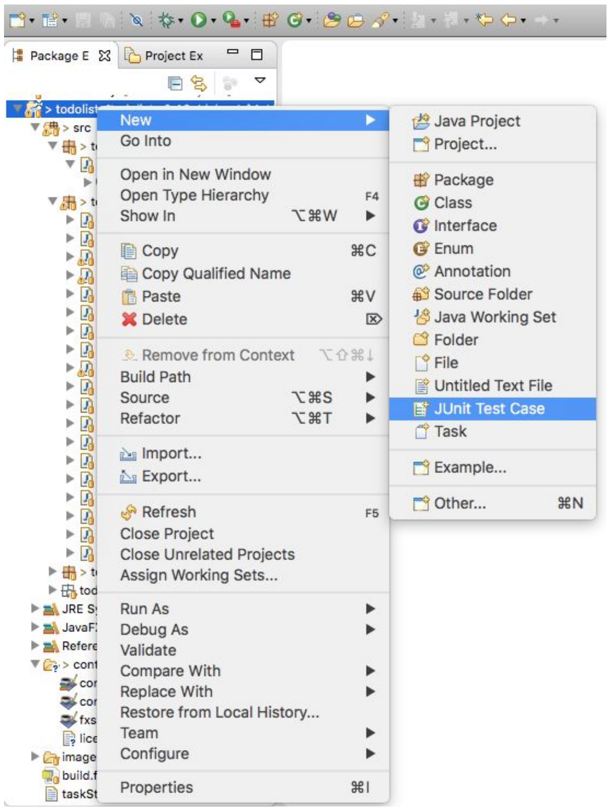
Modifier and Type	**Method and Description**
todolist.storage.Database	
boolean	**add**(**Task** task) This method handles the writing into the text file with the add command.
boolean	**checkExistence**(**Task** taskToCheck) This method returns whether a task is in the text file.
void	**clear**() This method is to clear the database and the local file
boolean	**delete**(**Task** taskToDelete) This method handles the updating of text file when the specified task is to be deleted.
String	**getPath**() This method returns the path of the local text file that stores all the tasks.
void	**loadFromFile()** This method read from local file
boolean	**openNewFile(String newFilePath)** This method opens a new file for task storage in another directory
Boolean	**recover(ArrayList<Task> backup)**
ArrayList<**Task**>	**retrieve(SearchCommand command)** This method search for and then return tasks from the database according to the command.
ArrayList<**Task**>	**retrieveAll()** This method returns all the tasks in the database.
boolean	**setNewFile(String newFilePath)** This method is to set new file for the storage of data.
ArrayList<**Task**>	**smartSearch(String[] keywords)** This method searches for tasks whose names containing the passed in string as a substring
void	**sort(String fieldName, String order)** This method sorts the tasks in the text file in the specific order of the specific field.
void	**writeToFile()** This method write the the list of the tasks into local file

(Table 6: Storage API)

4.4 Some Assembly Required: Implementation and Testing

When you are ready to add in new functionalities to the software, the Test Driven Development (TDD) approach should be applied. Before you start to implement these functions, you may start with writing some automated unit test-cases based on the behaviours you are going to implement. While you are implementing the new functionality, it is better to code in the way that is more testable, and then run the test case after the implementation of every single new function.

Because the program is written in Java, we use **JUnit** for the automated testing. If you are using Eclipse for programming, in order to create a new test case, you can right click on the project and then go to **New → JUnit Test Case**, which is shown in Figure 14 below:



(Figure 14: TDD JUnit Testing)

5 Appendices

5.1 Appendix A: Command Sheet

Command	Description
`add|schedule|create event <TITLE> <START-DATE> <START-TIME> <QUANTITY> <TIME-UNIT>`	creates a new event
`add|schedule|create deadline <TITLE> <END-DATE> <END-TIME>`	creates a new deadline
`add|schedule|create task <TITLE>`	creates a new floating task
`add|schedule|create recurring event <INTERVAL> <TITLE> <START-DATE> <START-TIME> <QUANTITY> <TIME-UNIT>`	create a new recurring event
`add|schedule|create recurring deadline <INTERVAL> <TITLE> <START-DATE> <START-TIME>`	create a new recurring deadline
`edit|modify|change|replace <TITLE> <FIELD-NAME> <NEW-VALUE>`	overwrites the specified field of an item with the new value
`delete|cancel|remove <TITLE|INDEX>`	deletes the item with the specified title from your agenda
`done|archive|complete|shelf|finish <TITLE|INDEX>`	marks the item with the specified title as completed
`undone|unarchive|incomplete|unshelf|unfinish <TITLE|INDEX>`	marks the item with the specified title as incompleted
`reset`	clears existing search and filters
`save <FILE-PATH>`	saves the current schedule to a file path
`open|load <FILE-PATH>`	sets new file source
`search <TITLE>`	highlights the item with the specified title
`filter <CATEGORY>`	filters the display for items belonging to a specific view / category
`sort <FIELD-NAME> <ASCENDING|DESCENDING>`	orders the items in the current view list by field(s) in the specified order, first by the earliest field specified
`label|categorize|tab <TITLE|INDEX> <CATEGORY>`	categorises a specified item with a category or label
`undo <#STEPS>`	reverts the last few changes
`redo <#STEPS>`	recovers the last few changes you have undone
`postpone|delay <TITLE> <QUANTITY> <TIME-UNIT>`	postpones a certain event or deadline with the specified title for a certain duration
`forward|advance <TITLE> <QUANTITY>`	brings forward a certain event or deadline with the specified title

`<TIME-UNIT>`	for a certain duration
`remind <TITLE|INDEX>`	sets reminder for the event or deadline with the specified title to trigger upon <START-TIME> or <END-TIME> of event and deadline
`remove-remind <TITLE|INDEX>`	removes reminder for the event or deadline
`set-recurring <TITLE> <QUANTITY>-<TIME-UNIT>`	sets a task as a recurring task
`remove-recurring <TITLE>`	sets a recurring task as a normal task
`add-remind...`	functions as **add** feature but with reminders set to trigger upon <START-TIME> or <END-TIME> of event and deadline respectively
`add-remind-bef <QUANTITY> <TIME-UNIT>...`	functions as **add** feature but with reminders set to trigger <DURATION> before event <START-TIME> or deadline <END-TIME>
`remind-bef <TITLE> <QUANTITY> <TIME-UNIT>`	sets reminder for event or deadline with specified title to trigger <DURATION> before the event <START-TIME> or the deadline <END-TIME>. <DURATION> is expressed as <QUANTITY-TIMEUNIT>
`tab <VIEW>`	navigates to the specified view
`help`	display the help sheet
`clean`	removes all the tasks in the schedule
`exit`	quits and closes the application
`[UP-ARROW | DOWN-ARROW]`	`[keyboard shortcut] browsing the task list`
`[ALT + UP | ALT + DOWN]`	`[keyboard shortcut] browsing command history`
`[ESC]`	`[keyboard shortcut] closing popups for help and reminders`
`[TAB]`	`[keyboard shortcut] browsing tabs or pages`
`[CRTL + K] for command and [CRTL + L] for list`	`[keyboard shortcut] change context between command field and task list`
`[CRTL + M]`	`[keyboard shortcut] mute/unmute`

5.2 Appendix B: Units Sheet

Field	Values and Format
`<TITLE>`	a string with no spaces or a string with spaces surrounded by “
`<START-DATE>`	YYYY-MM-DD or MM-DD
`<START-TIME>`	HH:MM
`<QUANTITY>`	a positive integer
`<TIME-UNIT>`	minute, hour, day, week, month
`<END-DATE>`	YYYY-MM-DD or MM-DD
`<END-TIME>`	HH:MM
`<FIELD-NAME>`	title , done, undone, start-time, end-time
`<NEW-VALUE>`	a value following the format specified for the particular field
`<VIEW>`	all, expired, today, week, done, options, help
`<CATEGORY>`	name of an existing category, or all
`<ASCENDING|DESCENDING>`	ascending, descending
`<#STEPS>`	a positive integer
`<INTERVAL>`	<QUANTITY>-<TIME-UNIT>

5.3 Appendix C: Units Sheet

[Likely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
addItem	create an item with different fields	can keep track of new events / deadlines / tasks
displayView	view events, tasks and deadlines in summary	can determine my schedule and workload
editItem	edit the fields of an item in his schedule	can adapt to changes in my schedule
deleteItem	delete unwanted items from his schedule	can reduce the clutter on the main list
archiveItem	archive items on the schedule	can mark and review completed tasks without cluttering the inbox
searchItem	search the schedule for an item	can verify the details as well as to check if the item has been successfully entered into the system
filterView	filter the schedule by categories or labels, as well as by presets such as overdue, today, this week, etc.	can effectively look at a specific group of items of a particular property that is of interest
sortView	sort, order schedule by fields / manually	can view the schedule in priorities
reschedule	postpone or bring forward deadlines and events by a certain duration	can adjust my schedule without having to edit fields specifically
setDirectory	specify a directory at which all the data for the program will be saved	can backup or recover the data through the preferred directory
changeStates	undo / redo the last few actions	can have room to make mistakes and recover swiftly
isOverdue	maintain an overdue inbox	can easily find out things that urgently require my attention
setLabel	categorize and label tasks	can classify them into respective genres
setReminder	set and receive notifications, reminders for upcoming events, deadlines	will be able to remember to complete and fulfil the tasks on time
addRecurring	enter recurring tasks automatically	will not have to re-enter the event every time a session is over

[Somewhat Likely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
toggleSettings	change global settings	have the option to make changes to the schedule without having to toggle settings for every single item on the agenda
resize	dynamically resize the application without readability compromised	can use the application care-freely with different resolution and window size

onMouseDown	navigate and interact with the GUI by clicking on buttons	can have the option of controlling the application by mouse
toggleContextMenu	invoke contextual menus	can be provided more tailored options upon inspecting a specific item on the schedule
toggleViewMode	choose between day and night mode	can reduce strains on the eyes when viewing the application under different lighting conditions
toggleCalendar	toggle between a calendar view and an agenda view	may easily keep track of any free time available in the schedule
exportData	share events, deadlines and tasks with collaborators	can conveniently show and convey the schedule with others without elaborating
setHighlight	view syntactical highlights while entering a command	can know if the application is recognizing the command accurately and correctly
autoComplete	complete my command automatically	can save time entering similar or repeated commands

[Unlikely]

ID	I can … (i.e. Functionality)	so that I … (i.e. Value)
setSprite	modify categories or labels with custom colours and icons	may associate with the tasks easily without having to read the category field all the time
addChild	create child task(s) that are associated with a parent task	can impose some hierarchy among the tasks or deadlines
verifyLogin	associate the schedule with an online service account	can establish a secured log-in before viewing my schedule in private, as well as to be able to backup and recover data from the online account
exportToGooCal	export or import data from google calendar	can choose not to rely on the desktop application alone when organising the schedule
synchronize	synchronize data with an online server	may access and edit the schedule from different terminals and yet be able to always acquire the latest and most updated version across all workstations

5.4 Appendix D: Non-Functional Requirements

- The application should run on Windows 7 or later
- The application should not crash unexpectedly and should produce error messages
- The application should be able to hold up to any amount of tasks, only limited by storage space
- The application should be able to save and quit without errors

5.5 Appendix E: Product Surveys

Product: todoist **Documented by**: Lie Jun

Strengths:
1. It supports keyboard shortcut and manipulation.
2. It allows users to archive the completed tasks.
3. It has clean and useful labels.
4. It supports email, which allows people like Jim to check email and then update the task list conveniently.
5. It supports synchronization from multi-platforms.
6. It does not require Internet connection for updating or access to tasks.
7. It supports 'today' and 'next 7 days' views.

Weaknesses:
1. It only supports deadline and floating task, but not event.
2. It does not allow users to check for free time slots or to block certain time slots.
3. It relies heavily on GUI.

Product: Smartsheet **Documented by**: Jiyi

Strengths:
1. It allows users to add new properties (like deadline, duration, etc.) on their own, so users can create different types of task, like deadline, event, and floating task.
2. It supports email reminder.
3. It allows users to set their own font size, color, style, etc.

Weaknesses:
1. It mostly relies on the use of a mouse with little keyboard shortcut.
2. It requires Internet connection for updating or checking the task list.
3. It has too many labels and buttons, which may take very long time to get used to.
4. It does not allow users to check for free slots.

Product: Wunderlist **Documented by**: Yuxin

Strengths:
1. It is desktop software that does not require Internet connection for updating or checking tasks.
2. It supports recurring tasks.
3. It supports keyboard shortcut, requiring few clicks to create or update tasks.
4. It has clean and useful labels and supports categorization of tasks.
6. It supports desktop reminders.
7. It supports 'today' and 'this week' views.

Weaknesses:
1. It only supports floating tasks and deadlines but not events.
2. It is unable to block or check for free slots.