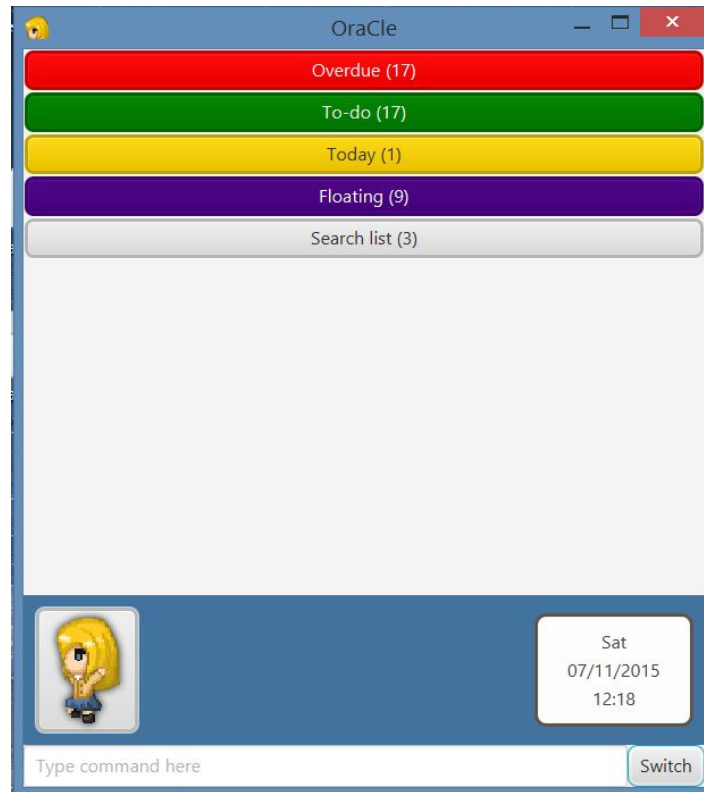


OraCle



Supervisor: *Chan Jun Wei*

Extra feature: *Good GUI, FlexiCommands*


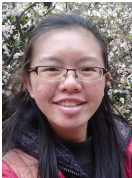

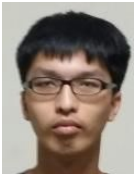
			
Terence	Ka Yi	Jia Min	Darryl
Team Leader, Logic, Code Quality, Integration	Parser, Testing, Integration	Deliverables and deadlines, Documentation, Scheduling and Tracking	GUI implementation, Eclipse expert

TABLE OF CONTENTS

User Guide

1. Getting Started
2. How to use OraCle
 - 2.1. HashTags
 - 2.2. Formats
 - 2.3. Adding
 - 2.4. Deleting
 - 2.5. Undo/Redo
 - 2.6. Searching
 - 2.7. Marking
 - 2.8. Editing
 - 2.9. Set storage location
 - 2.10. Help
 - 2.11. Exit
3. GUI Commands
 - 3.1. Open
 - 3.2. Close
 - 3.3. Pin
 - 3.4. Unpin
 - 3.5. Show
 - 3.6. Switch/Main/Log
4. Troubleshooting
5. Reference list

Developer Guide

1. Introduction
2. Library used
3. Architecture
 - 3.1. GUI Component
 - 3.2. Logic Component
 - 3.2.1. Initialisation
 - 3.2.2. Executing commands
 - 3.2.3. Updating the Model
 - 3.2.4. The Undo/Redo Stacks
 - 3.2.5. Extending Logic Class
 - 3.3. Parser Component
 - 3.3.1. InputParser
 - 3.3.2. DateTimeParser
 - 3.3.3. Other Useful Knowledge
 - 3.4. Storage Component
4. Testing
5. Future Implementations

Appendices

USER GUIDE

Welcome to OraCle, a task manager that works as both a to-do list and a calendar that helps you manage your tasks and events! With simple and intuitive commands, you'll be able to get started in no time at all! Note that everything can be done with just your keyboard in OraCle, which makes using OraCle a breeze. Excited? Let's get started!

1. Getting started

a. Get OraCle

Get the latest release of OraCle by downloading OraCle.jar from:

<http://github.com/CS2103-Aug2015-w15-4j/main/releases>

Create a folder and place OraCle.jar inside. By default, OraCle will create all its required data in this folder. You may choose to change this setting later.

b. Launch OraCle

Simply double click on the OraCle.jar file to start OraCle. Upon opening OraCle, you will be presented with the following screen.

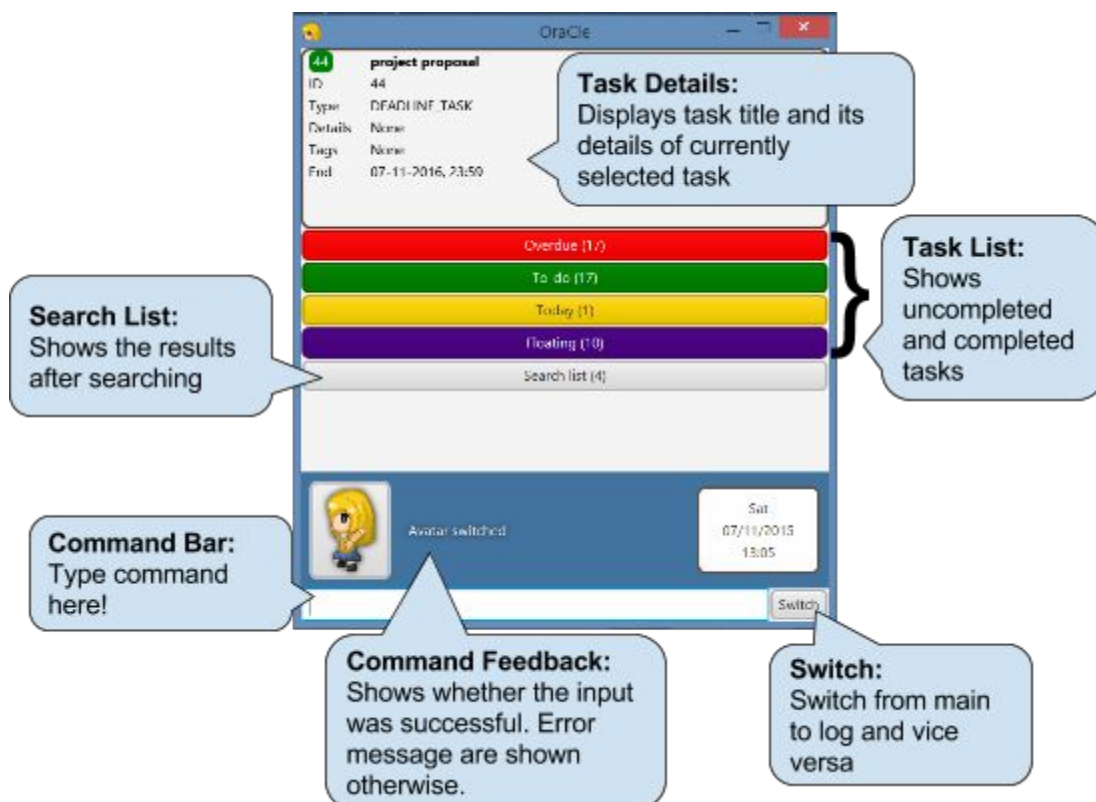


Figure 1. Main view of Oracle

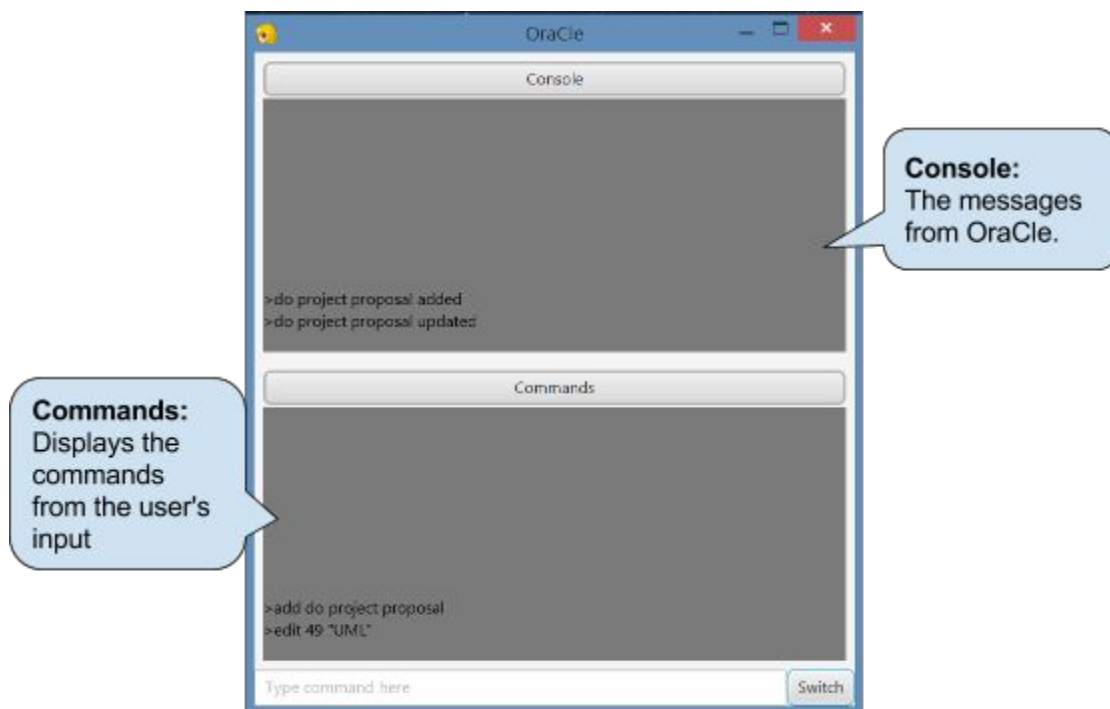


Figure 2. Log view of OraCle.

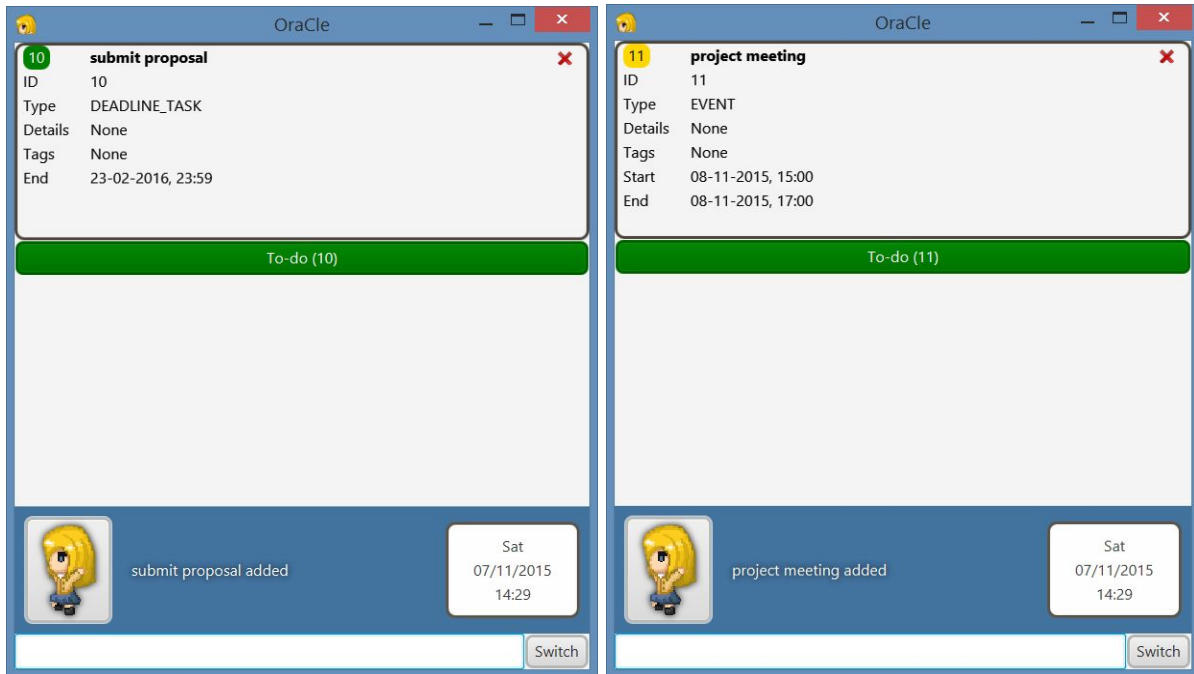
c. Start using OraCle!

You can use the `add` command to start adding tasks to OraCle. For example, try

```
>> add submit proposal by 23 feb
```

```
>> add project meeting tmr 3-5pm
```

It will immediately be added and displayed.



Congratulations, you have successfully started using OraCle! You can have OraCle's lovely avatar pat you on the back by clicking her!

d. (Optional) Set storage location

Want to import a data file from another computer or change the location where OraCle saves its data files? You can use the `set` command to set the location of the data file that OraCle will read the tasks from and save them.

```
>> set folder <file directory>
```

The next section will teach you more commands and help you learn more about OraCle's features and how to use them.

2. How to use OraCle

OraCle is designed to be intuitive and simple to use, so that managing your tasks can be done quickly. Thus, commands have similar input formats and use a one-shot approach, where each action requires just one line of input to perform.

Now let's get started on some things you need to know before you can use this User Guide effectively.

2.1. Hashtags

We want to give you flexibility in choosing how to organize your tasks, thus OraCle supports the use of hashtags. You can spot the use of such hashtags in the following sections by their **red** font, e.g. **#tag**. Here are a few ideas on how to use them effectively:

1. Organizing by category:

Hashtags can be used to relate tasks from the same category. For example, you can label all tasks for your DIY project as **#DIYproj**. This way, it is easy to search for all tasks you need to complete to finish your DIY project. Just use "search **#DIYproj**"!

2. Priority:

Hashtags can also be used to indicate priority. For example, you can label high priority tasks with **#1**, and low priority tasks with **#3**. Alternatively, you can use Eisenhower decision matrix for time management and label your tasks with **#urgent**, **#impt**, **#notUrgent**, **#notImpt**! It is entirely up to you!

3. Location:

Another possibility is using hashtags to indicate location so you can see what needs to be done when you're heading to a specific place. For instance, if you tag items to be bought with **#grocery** throughout the week, all you need to do is search **#grocery** for your grocery list!

2.2. Formats

OraCle uses many short forms in this User Guide to make things easier for you to glance through and quickly understand the relevant section. Thus, this section was specially created to help you understand what they mean.

add	Words in blue indicate OraCle's commands. You can use them exactly as they are.
<title>	< > indicate fields and the name of the field. You should replace the entire <title> with an appropriate title.
[<date/time>]	[] indicate optional fields. You can choose to include this field if you wish.
#<tags> on <start time> by <end time>	Anything located before a field indicates a prefix keyword required for OraCle to detect the field.

CASE sensitivity	Oracle's commands and keywords are not case sensitive. So, you can choose to use ADD or Add or aDD and OraCle will still detect them as the add command.
------------------	--

For example, if you want to add a task with title “**send email**” with a tag of “**proj**”, to follow given command format:

```
>> add <title> [#<tags>]
```

```
>> add send email #proj.
```

Notice that ‘<title>’ has been replaced with ‘**send email**’, ‘<tags>’ with ‘**proj**’. The keyword ‘#’ has been kept and the ‘[]’ brackets are not included.

Date/Time

OraCle accepts most formats for <date/time>. You will find that they are indicated by **purple** in this User Guide,

Dates can be split into 2 unique formats.

Relative dates:

- **by today 15:00**
- **on next Tues 3-5pm**
- **on next mon to next thurs**

Relative dates require the **keywords on** or **by** to detect. If there are two dates or times, you can must use the **to** keyword to separate the start and end. The first date/time will be detected as the start date/time and the second date/time will be detected as the end date/time.

Actual dates:

- **2 jul 2-4.30pm**
- **3/12/14 8 am to 6.30pm**
- **august 23, 2011 to sep 10, 2012**
- **3/12/15 15:00 5/12/15 17:00**

Actual dates have no **keyword** requirements, though you can choose to include them. You may also use the **to** keyword to separate start and end date/times if you wish. If OraCle detects two different dates, it will assume the first date is the start date/time and the second is the end date/time. If only a date is added, like **3/12/15**, the default time will be set to **23:59**.

Tips & reminders!

Look out for these boxes for useful tips and reminders!

2.3. Adding

You can add tasks to OraCle very easily with the **add** command that lets you add all your information such as **date**, **time**, and any **notes** on the task all at once!

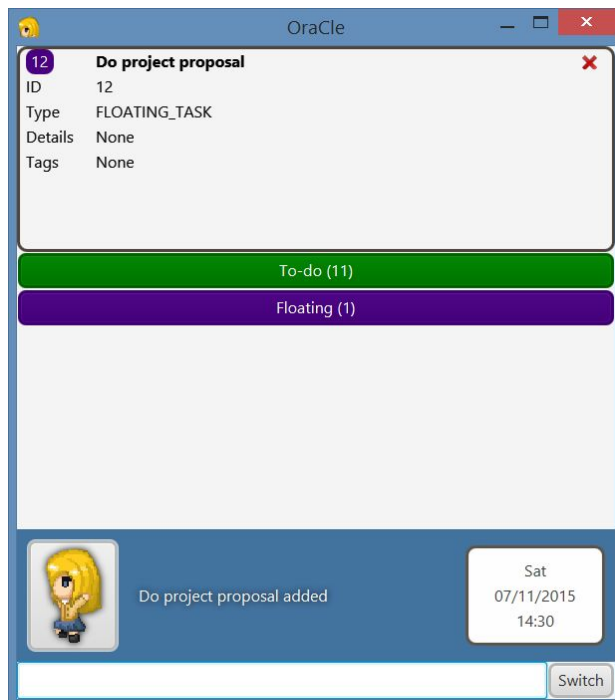
```
add <title> [<date/time>] [#<tags>] ["<description>"]
<title>      Name of the task to be added
<date/time>  Deadline or start and end date/times for events
<tags>       Categories to add to
<description> Details of the task
```

Friendly reminder!

Make sure your **start/end** fields are after **title** if you are using relative dates, otherwise OraCle will not be able to read your command properly.

Let us say you wish to add a task called “Do project proposal”.

```
>> add Do project proposal
```



You will immediately be able to see the new task added as a **Floating Task**¹. If you wish to add more information during the **add**, simply include the optional fields in the [].

¹ Task with no dates/times

For example, if you wished to add the same task, but with a due date of **20th of October** and an **office** hashtag, enter:

```
>> add Do homework by 20th October #office
```

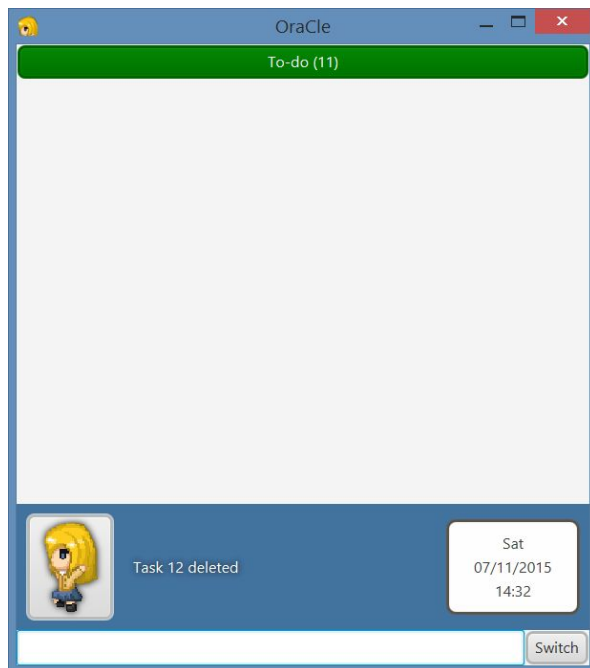
2.4. Deleting

Deleting is even simpler than adding. Only one field is needed for this function!

```
delete <ID>  
<ID> ID of the task to be deleted
```

Let's say you wish to delete the "Do project proposal" task.

```
>> delete 12
```



With that, you have removed the task from the application!

Useful tips!

Think **delete** takes too long to type for a busy person like you? Try **X** instead! For example, type **X 3** to delete task 3 instead!

To make using OraCle a breeze, we offer shortcuts for other commands as well, as listed in

4 Reference List.

2.5. Undo/Redo

What if you realise you had made a mistake, and that you actually weren't supposed to delete that task? This is where the **undo** function comes in handy!

undo

Undo the last **add** or **delete** command

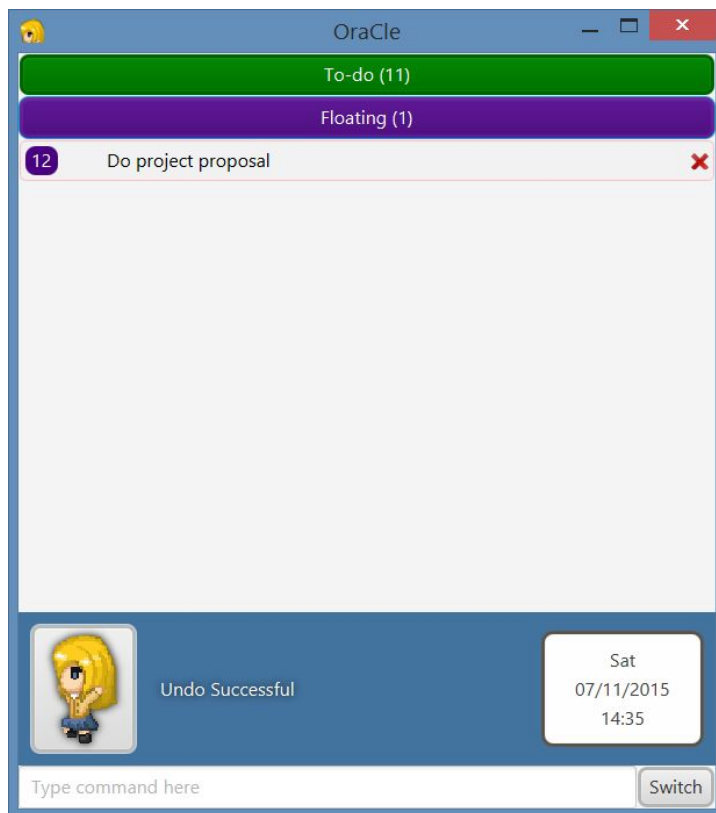
If you change your mind about **undo**, simply enter **redo**!

redo

Reverse last **undo** command

>> **undo**

And voila, it is back!



You can undo all the commands up to the time you last launched OraCle!

2.6. Searching

Are you looking for a particular task? `show` displays details of your chosen task if used with `<ID>`. If you are unable to find that single task that you wanted to delete, `show` is also able to show tasks according to your search criteria. (Alternatively, you can use command word `search`, which is equivalent to `show`.) This way, you can easily find the task you want without needing to navigate through the entire list of tasks for it.

```
show <ID>
search [<keywords>] [<date/time>] [#<tags>] [<status>][<task type>]
<ID>          ID of the task you would like to view details of
<keywords>    Shows tasks that match the keyword within the <title> or <description>
<date/time>   Date or date range to search
<tags>        Shows tasks that match the <tags> have been input
<status>      Shows tasks that match the status needed (todo, done, overdue)
<task type>   Task types floating, deadline, event
```

2.7. Changing task status

When you finish your tasks, easily mark them as done using the `flag done` command. You can also change a task status from `done` to `todo`.

```
flag status <ID>
<ID>    ID of the task
Status  Status of the task to flag as. Valid statuses are: todo, done
```

Useful tips!

Alternatively, just type `V <ID>` to mark a task as done! For example, if you've completed your homework (with task ID 3), simply type `V 3`.

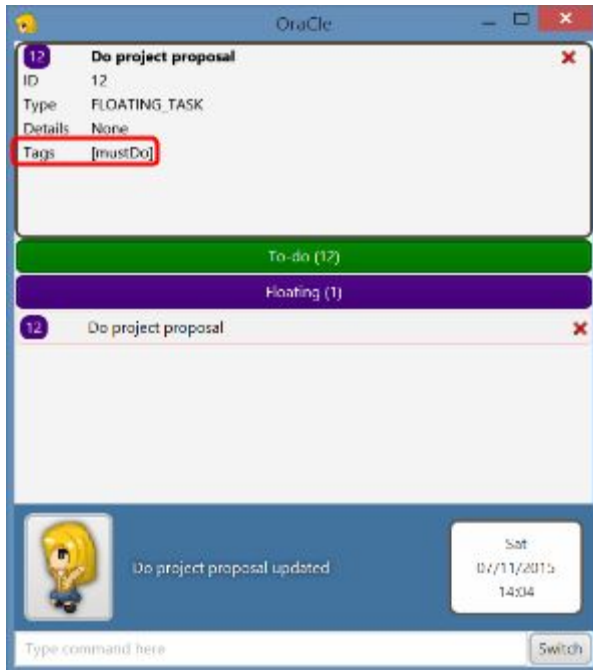
2.8. Editing

Made a mistake? Forgot to add something? No worries, simply use `edit` to change the relevant fields!

```
edit <ID> <new data 1> <new data 2> .....
<ID>          ID of the task to be edited
<new data>    Data to replace existing details, of type title, tags or description (same format as in
               add)
```

For example, if you would like to replace your previous tags with the tag of `mustDo`, you can use `edit` to replace all tags immediately.

```
>> edit 12 #mustDo
```



2.9. Set storage location

Want to sync your files so that you may access your tasks across different computers? Use `set folder` to store your tasks in a folder controlled by a cloud syncing service (e.g. Dropbox). Otherwise, your tasks are stored in the folder containing OraCle.jar by default. You can also use `set avatar` to customise avatar images.

```
set <object> <name>
```

Object Object to be changed. Valid objects are: `folder`, `avatar`

New details File directory (for `folder`) or image file name (for `avatar`)

For example, to store your tasks in a Dropbox folder in the Desktop, use:

```
>> set folder C:\Users\Jimmy\Desktop\Dropbox\
```

If you wish to view the file produced by OraCle, you may look at `Data.txt` in the folder specified.

2.10. Help

Some people may have photographic memories, but most of us have too much to remember. Hence, we've made it simple for you to check command formats and shortcuts within OraCle itself, using the help command.

[help](#)

Displays list of commands and corresponding formats.

Entering [help](#) brings up a list of commands (which can be seen in **4 Reference list**).

>> [help](#)

2.11. Exit

To close the program, use [exit](#), Alt-F4, or click the application exit button (the cross) on the top right. All changes to your data are automatically saved each time you enter a command, so even if the program does not exit properly, all your data remain stored!

[exit](#)

Exit OraCle.

>> [exit](#)

3. GUI Commands

All of OraCle's clickable buttons are accessible with the keyboard. These are classified under GUI commands. They are around to encourage you to leave your hands on the keyboard and keep your fingers busy, with a minimal need to touch the mouse.

3.1. Open

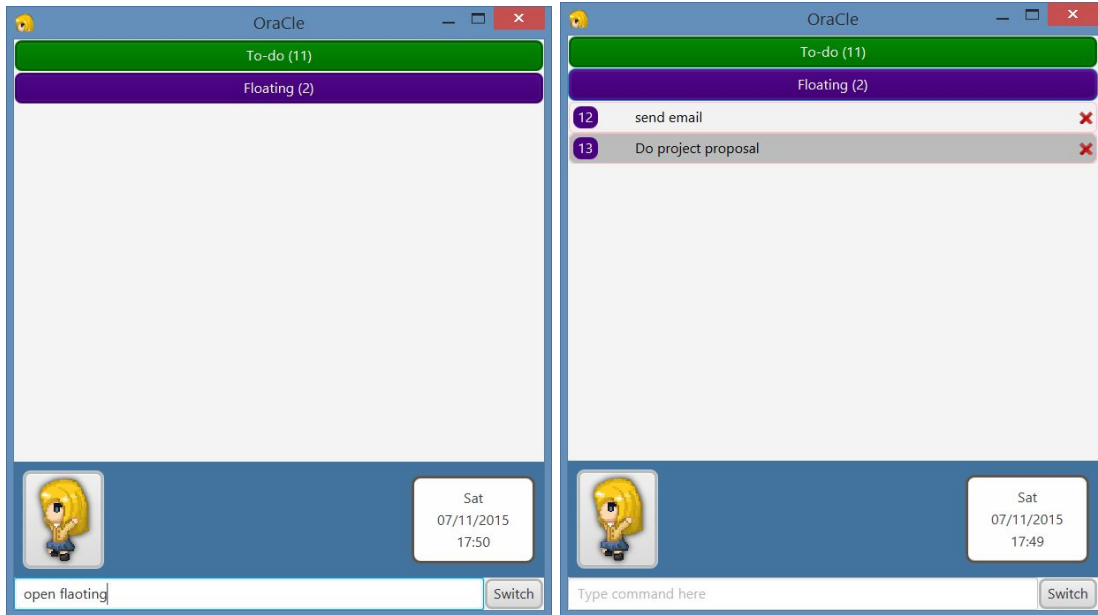
Want to see a list of tasks in a specific list? Use [open](#) to view a task list.

[open](#) <Name of task list>/<List position from top>

<Name of task list> Task list to be opened: [floating](#), [to-do](#), [today](#), [overdue](#), [search list](#)

<List position from top> Range from [1](#) to [5](#)

>> [open](#) [floating](#)



Task lists that were previously opened will be closed when you open your new task list.

3.2. Close

To close an open task list, simply call [close](#).

[close](#) <Name of task list>/<List position from top>

<Name of task list> Task list to be closed: [floating](#), [to-do](#), [today](#), [overdue](#), [search list](#)

<List position from top> Task list number: [1](#) to [5](#)

>> [close](#) [floating](#)

3.3. Pin

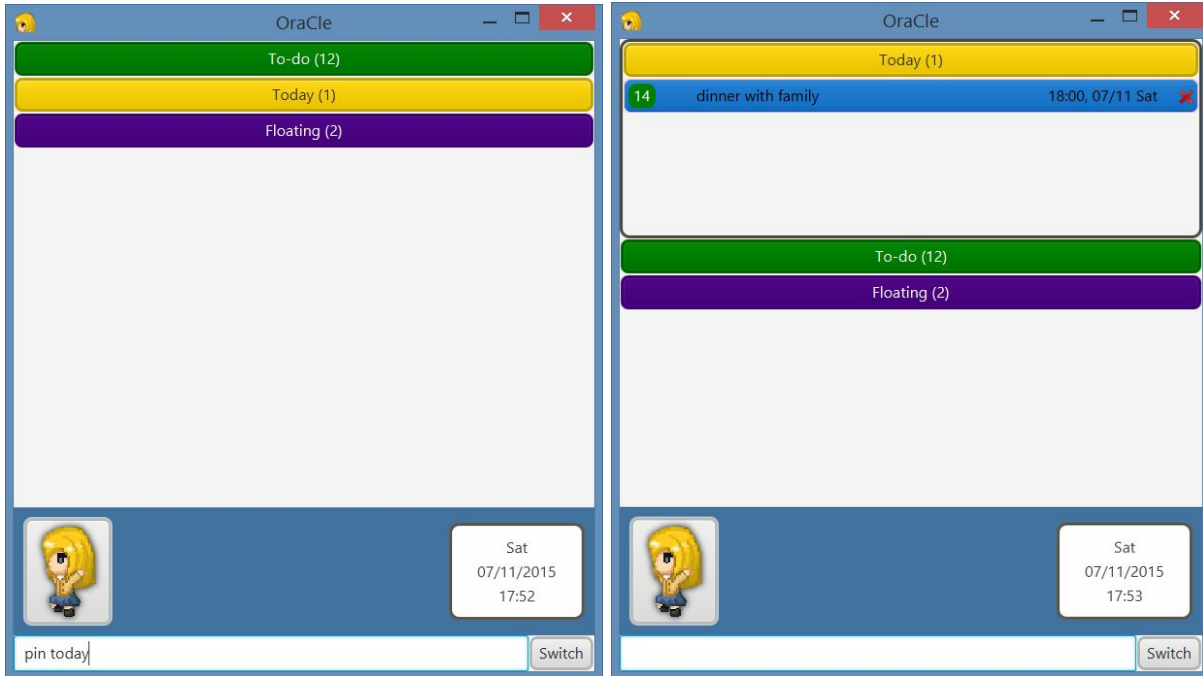
Want to follow up keep a specified task list open at all times? Use [pin](#)! Task lists you pinned will remain at the top of OraCle until you unpin them or exit OraCle. (By default, if the Overdue task list has items, it will start out pinned)

[pin](#) <Name of task list>/<List position from top>

<Name of task list> Task list to be pinned: [floating](#), [to-do](#), [today](#), [overdue](#), [search list](#)

<List position from top> Task list number: [1](#) to [5](#)

>> [pin](#) [today](#)



3.4. Unpin

You can unpin the task list too!

unpin
Unpin the task list.

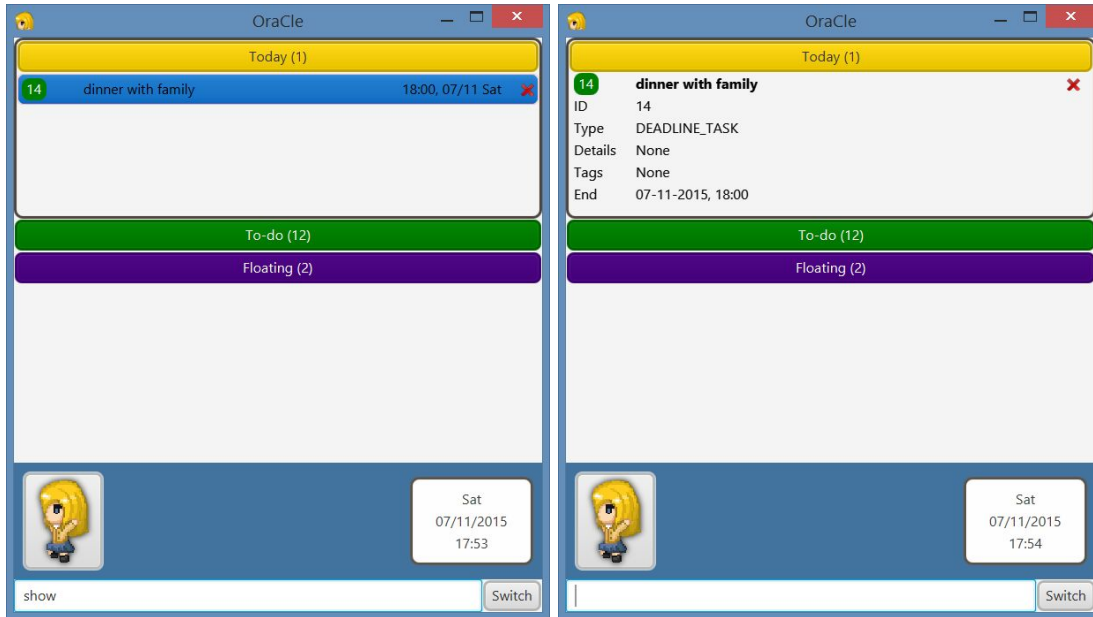
>> **unpin**

3.5. Show

Want to view the description and tags of a particular task? Use **show** to view the details. In this mode, you can press the up and down arrow keys to view the details of other tasks in the same list.

show
Opens a task in **Focus Mode**

>> **show**



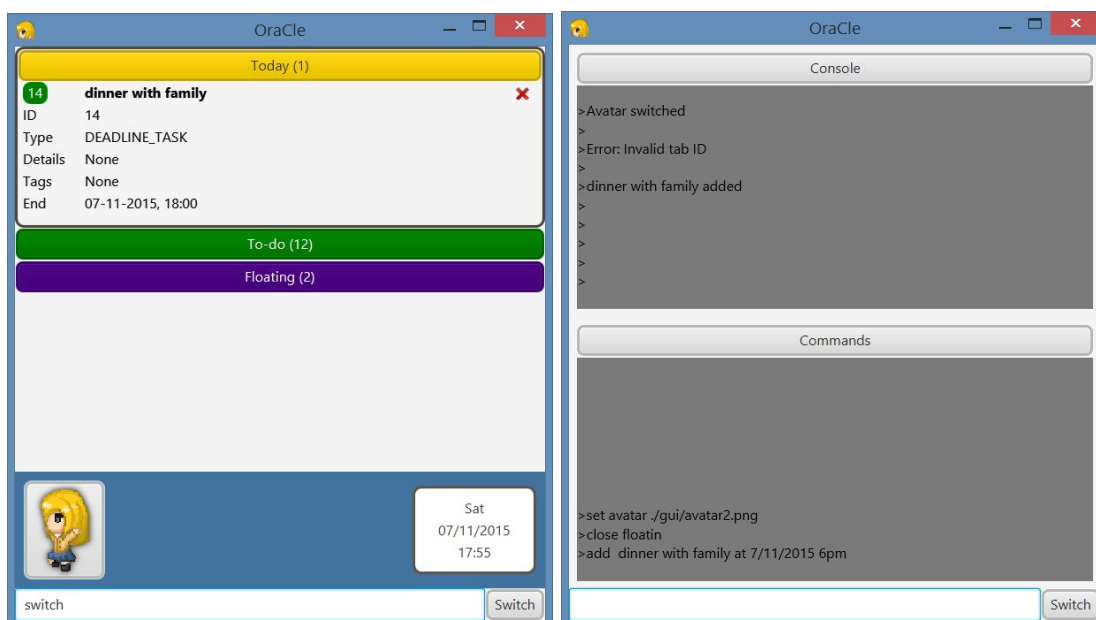
3.6. Switch/Main/Log

Oracle offers a log view that displays a record of your past inputs and their corresponding output messages. To switch between this and the default view, use [log](#), [main](#) or [switch](#). You can also access previously entered commands by pressing the up arrow key when in the command bar.

[switch/main/log](#)

Switches from Task View to History, main or log respectively

>> [switch](#)



Useful tips!

Try using ALT + T to switch between the views! While we are at switching views, try out ALT + Space when window switching instead of your usual ALT + Tab to toggle OraCle!

4. Troubleshooting

We try our best to be as intuitive as possible, but there are times when OraCle misinterprets your command. In such cases, ensure that you have used our keywords or symbols correctly. For example, you may have keyed in the following:

```
>> add watch day The Day After Tomorrow movie
```

Note that **Tomorrow** is detected as a **date**, and hence does not appear in **title**. To solve this, add a backslash \ in front of **Tomorrow**. Similarly, adding \ in front of keywords will allow the application to know that you wish to use the keyword as a word and ignore it.

```
>> add watch day The Day After \Tomorrow movie
```

Keywords and special symbols that require use of backslash if used in **title/search keyword** in Oracle are as follows:

Keyword/symbol	Example	Title/keyword without \ used	Title/search keyword with \ used
Double quotation marks “...”	read \“The Little Prince\”	read	read “The Little Prince”
Date/time section	prepare \1 Apr joke	prepare joke	prepare 1 Apr joke
Hashtags #	visit \#5 building	visit building	visit #5 building

The following keywords and special symbols apply to show command only.

Keyword/symbol	Example	Search keyword if \ is not used	Search keyword if \ is used
Task status todo/done/overdue	\overdue books	books	overdue books
Numbers at beginning of search keyword	\3 apples	[displays task 3]	3 apples
Task type floating/deadline/event	proj \deadline	proj	proj deadline

5. Reference list

Field	Syntax	Examples
Title	None	proj meeting
Description	"<description>"	"bring proposal"
Date (if year is omitted, default year is set to current year)	dd/mm/yy or dd-mm-yy or dd.mm.yy dd/mm or dd-mm or dd.mm or dd/mm/yyyy or dd-mm-yyyy or dd.mm.yyyy	22/3/15 or 22/03/2015 or 22-3-15 or 22.3.15 or 22/3
	dd MMM yyyy or dd MMM	3rd Feb 2014 or 2 january
	<natural language date>	on next Mon or by 3 days or tmr or today
Time	<natural language time>	2pm or 3.30-4pm or 8.00am or 23:59
Tag	#<tag>	#cs2103

COMMAND HELP

Commands	Shortcuts	Description
add/insert	+	Adds a task
edit / update	e;	Replaces task fields with new data indicated in user input
delete/cancel	X	Deletes task
show/search/ display	s; Ctrl-F	Shows tasks according to criteria (search/filter function) or details of a task
flag /mark	flag	Change task status to done or todo
done	V	Mark task as complete
undo	Ctrl-Z	Undo previous action
set	set	Set location of data file or choice of avatar
help	h; ?	Displays list of commands/details of a command
exit/quit	q;	Exit OraCle

GUI CONTROL COMMANDS

Commands	Keyboard Shortcuts	Fields	Description
open		<Name of task list> <List position from top>	Opens a selected task list
close		<Name of task list> <List position from top>	Close a selected task list
pin		<Name of task list> <List position from top>	Pins a selected task list
unpin	Ctrl+U	None	Unpin a selected task list
	Shift+\		Clears Focus Mode
show	\	None	Activates Focus Mode
	None	<TaskID of task>	Opens a task in Focus Mode
switch main log	Alt+T	None	Switches from Task View to History Switches View to main or log respectively
	Ctrl+Space	None	Minimises/expands application window

DEVELOPER GUIDE

1. Introduction

OraCle is a task management tool that combines both functionalities of to-do lists and calendars. It is a Java Desktop application which uses a very flexible graphical user interface (GUI), allowing easy shifting of objects.

In this guide, you will learn about the design and implementation of OraCle, beginning with an overview of the architecture. The subsequent sections cover each component in detail, as well as how to conduct testing. We hope this guide provides a useful introduction to OraCle and makes it easy for you to get started on contributing to OraCle's development. Welcome aboard and all the best!

2. Understanding Oracle

2.1 Understanding the core experience

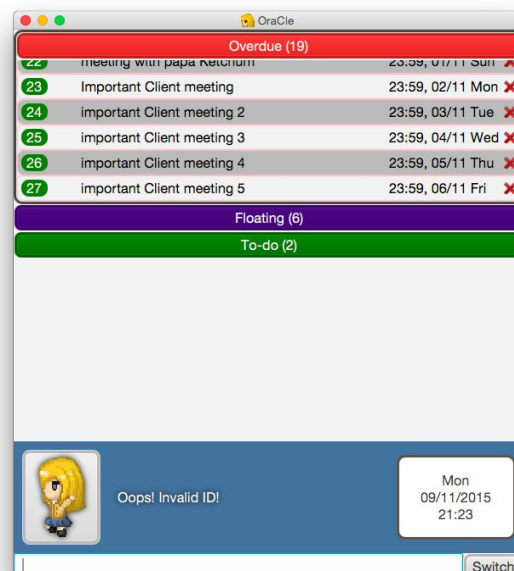
Oracle was designed to be an interactive experience between you and your Avatar. By having text based commands, we further reinforce this idea by making the interaction similar to how you would communicate with another person in real life.

2.2 Recommended guidelines

User Interaction

In order to stay consistent with Oracle's theme ,the following should be structured close to what a person would normally say in natural speech.

- Commands and their syntax
- Response messages
- Error messages



2.3 Libraries

OraCle uses the following third-party libraries in its implementation, so do familiarise yourself with the relevant libraries for your component.

1. **Lucene**
Used by Logic to handle full-featured text search.
2. **GSON** <http://sites.google.com/site/gson/>
Used by Logic and Storage to convert between Java objects and JSON objects for storing user's data.
3. **Natty** <http://natty.joestelmach.com/>
Natural date parser used by Parser to parse natural language dates.
4. **JavaFX**
Java library used by GUI to create the rich user interface.

2.4 Design Patterns

Command Design Pattern

“The command design pattern is a behavioural design pattern that encapsulates method calls in objects and thus allowing us to issue requests without knowledge of the requested operation or the requesting object.” extract from <http://www.oodeesign.com/command-pattern.html>

The Command Design Pattern provides us with the ability to queue commands and perform undo/redo operations and is used extensively in the Logic component.

Model-View-Controller Pattern (MVC)

“The central component of MVC, the *model*, captures the behavior of the application in terms of its **problem domain**, independent of the user interface.^[11] The model directly manages the data, logic and rules of the application. A *view* can be any output representation of information, such as a chart or a diagram; multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The third part, the *controller*, accepts input and converts it to commands for the model or view.” extract from <https://en.wikipedia.org/wiki/Model-view-controller>

Oracle uses the MVC pattern to implement its GUI. The Model Class represents the Model, the GUI_Controller & Logic act as the Controller and the rest of the GUI represents the View.

Builder Pattern

The builder pattern is a design pattern for creating objects that have multiple parameters. This pattern allows you to construct objects using by passing the builder each of the relevant parameters in any order, before the object is constructed at once.

This pattern is used by ParsedCommand in Parser, to handle its many initialization parameters during construction.

Chain-of-Responsibility Pattern

“In object-oriented design, the chain-of-responsibility pattern is a design pattern consisting of a source of command objects and a series of processing objects. Each processing object contains logic that defines the types of command objects that it can handle; the rest are passed to the next processing object in the chain. A mechanism also exists for adding new processing objects to the end of this chain.” extract from https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern

Oracle uses Chain-Of-Responsibility Pattern for date and time parsing in Parser, passing input through the different parsers, until the input can be parsed by the relevant parser and the result returned.

3. Architecture

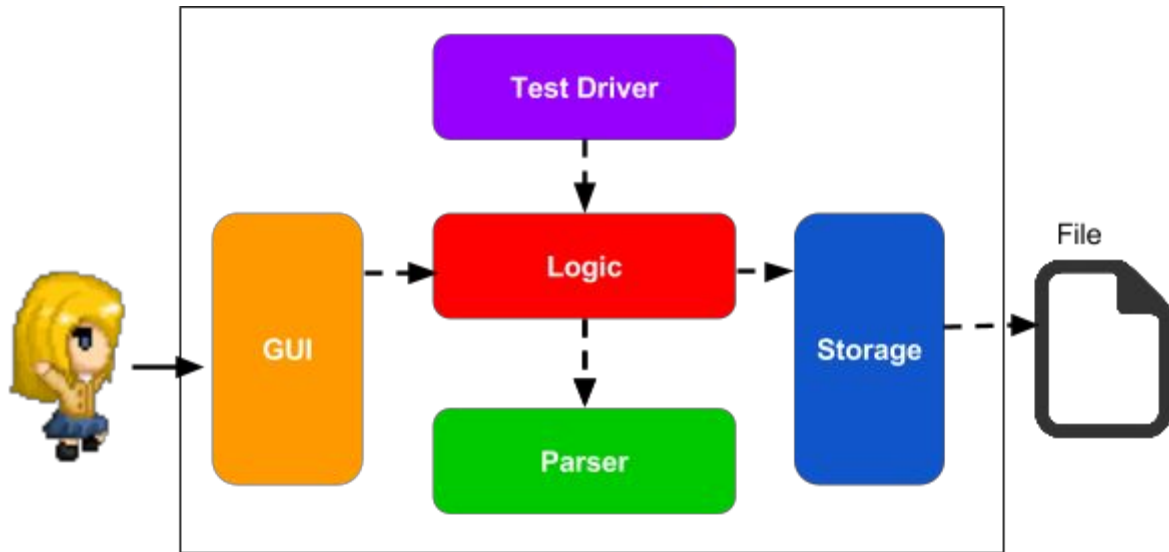


Figure 3. Components of OraCle

OraCle consists of 5 main components.

1. The GUI component consists of GUI controller to respond to user input and maintain the view for users to interact.
2. The Logic consists of the command classes and decides which command class to call upon an action.
3. The Parser component filters the user's commands into usable data.
4. The Storage component consists of text file. This file is the storage area for OraCle to store user's tasks. The file is also loaded when the user launches OraCle.
5. The Test Driver consists of JUnit test cases for GUI, Parser, Logic and Storage for testing.

3.1. GUI Component

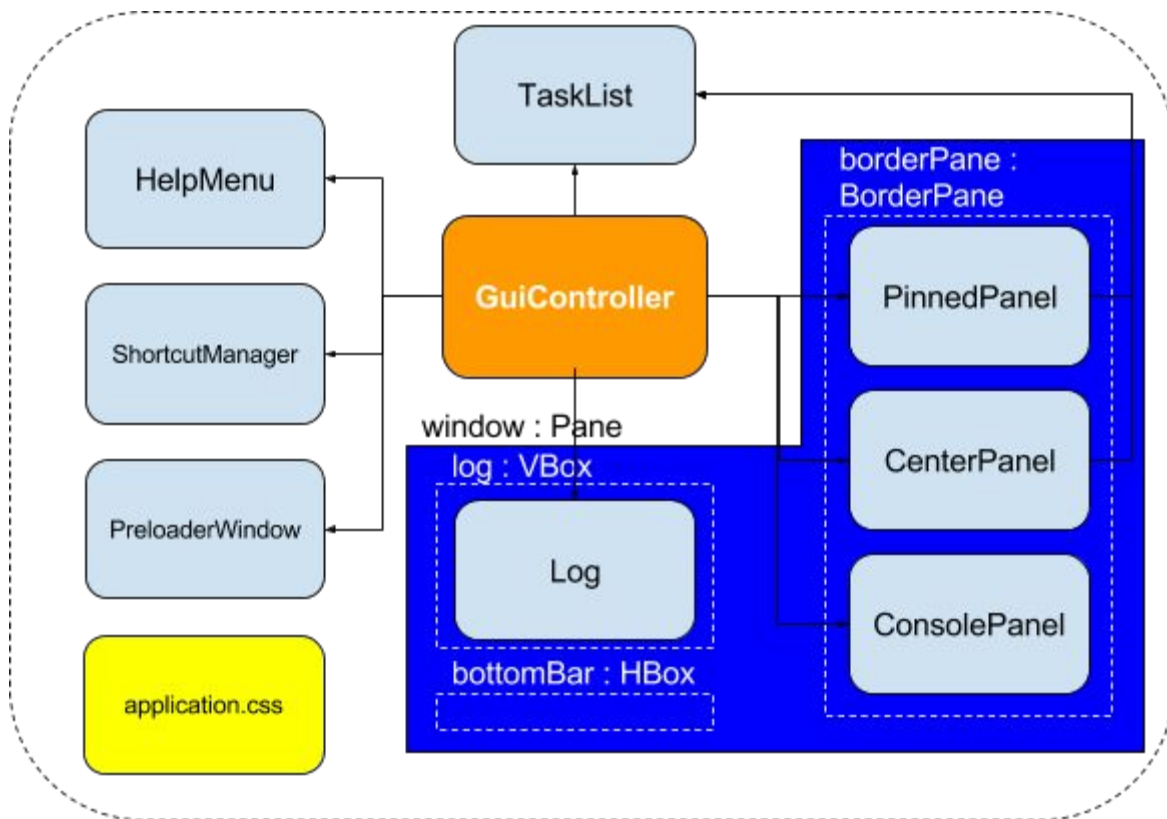


Figure 3.1.1. Components of GUI

The GUI is meant to give the user access to all information and sort them as neatly as possible. To do this, the `TaskList` class is used, and is thus the core of the displayable containing all the `Tasks`. Thus, it is vital for you to know how `TaskList` works and how it integrates into the entire GUI.

When a `Task` is added to the `TaskList`, it is automatically added to the `listView : ListView` displayable, using the `createMinimisedDisplay(Task task)` function. If a `TaskList` has items inside, the `borderPane : BorderPane` automatically refreshes and shows it to the user. `createDetailedDisplay(Task task)` is used via `focusTask()` only when the user wishes to view more information on the `Task`, outputting it to `detailedView`. The `TaskList` can then toggle between the `listView` and `detailedView` through clicking the `name` button or GUI commands.

The `TaskLists` are then placed within either the `PinnedPanel` or `CenterPanel` of the `borderPane : BorderPane`, where the user can choose which `TaskLists` to put in which panel. Note that at any point in time, `PinnedPanel` can thus hold and pin only one `TaskList` at any point in time.

TaskList	
#	<code>listView : ListView</code>
#	<code>name : Button</code>
#	<code>detailedView : ScrollPane</code>
+	<code>addTask(Task task) : void</code>
+	<code>deleteTask(Task task) : void</code>
+	<code>openList() : void</code>
+	<code>closeList() : void</code>
+	<code>focusTask() : void</code>
+	<code>createMinimisedDisplay(Task task) : GridPane</code>
+	<code>createDetailedDisplay(Task task) : GridPane</code>

The `PinnedPanel` and `CenterPanel` are then placed within pane in the `GuiController` class, which handles interaction between them and their `TaskLists`.

Now that you know how the `TaskList` fits into the structure of GUI, let us take a look at a sequence diagrams to see how GUI initialises and uses them.

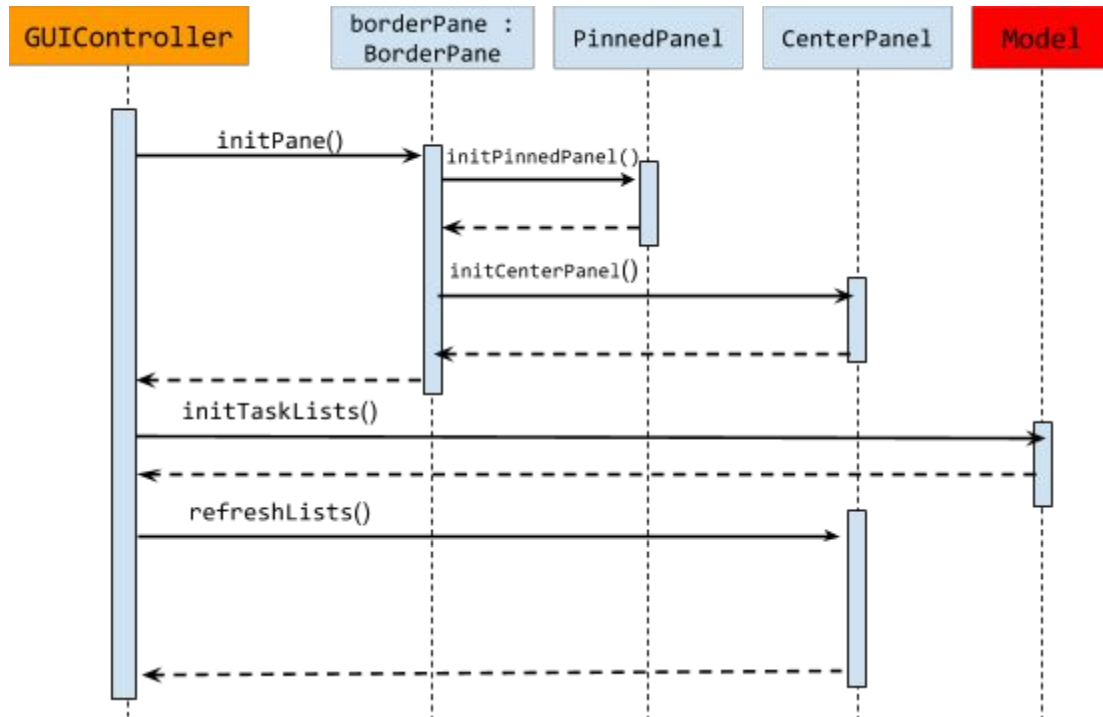


Figure 3.1.2. Initialisation of GUI

As you can see, `GuiController` first instantiates the frames, `PinnedPanel` and `CenterPanel`. Following that, it draws from the `Model` the `Task` information, sorting them into `TaskLists`, before using `refreshLists()` to put them into the `CenterPanel`. `PinnedPanel` is thus only used when the `pin(TaskList list)` function is used, and remains empty until then.

Things to note are that `window : Pane` is a frame for `borderPane : BorderPane` located within `GuiController`. This frame thus allows for the ability of OraCle to switch between `borderPane : BorderPane` and `log : VBox`, which is a log of all user inputs and console outputs for that particular instance of OraCle. The frame `bottomBar : HBox`, also located in `GuiController`, containing the `userInputField : TextField` and `windowSwitch : Button`.

The rest of the GUI classes explain themselves in their functionality. The `ConsolePanel`, however, contains a special avatar feature for OraCle. At the moment it is a very basic `Image` that acts as a `Button`, allowing for sound clips to be played when clicked.

Possible upgradable features of OraCle would be `createDetailedDisplay(Task task)`, which is currently very simplistic compared to the rest of OraCle. Organisation of `TaskLists` (currently consists of 5, which can be found in `taskListNames : String[]`) and `TaskList` functionality can be modified as well depending on user feedback. Aesthetics wise, more application of CSS as well as an upgrade to the avatar feature can be implemented.

3.2. Logic Component

As previously mentioned and can be seen below, the Logic component follows the command design pattern.

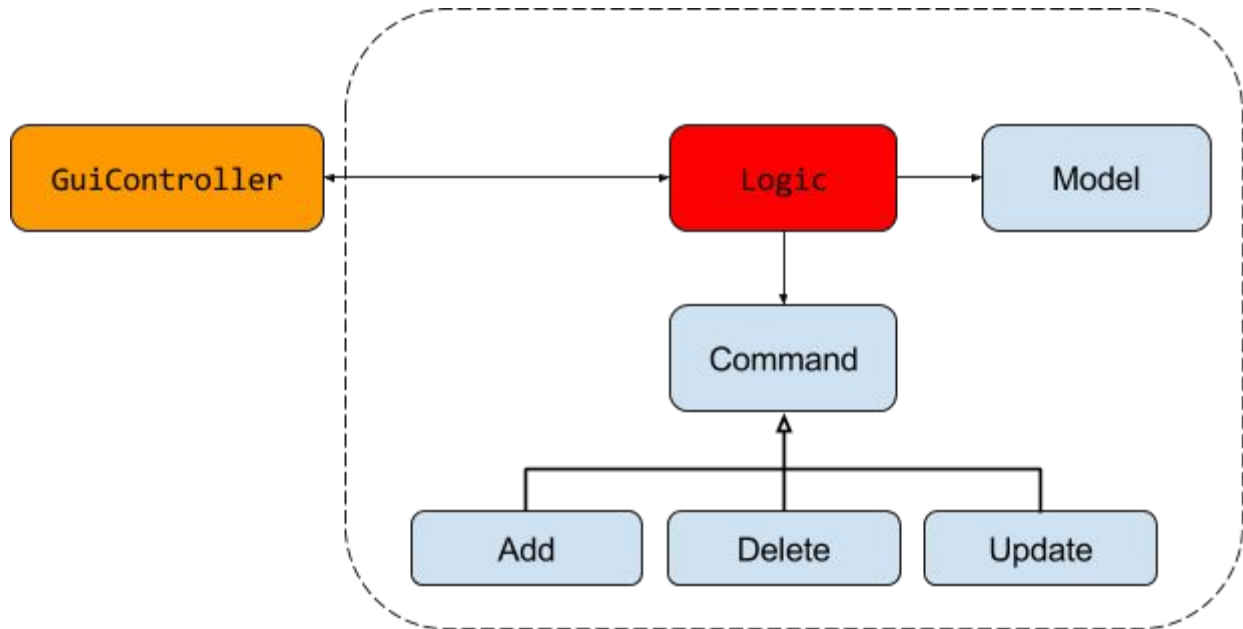


Figure 3.2.a Components of Logic

Logic acts as the “brain” of Oracle and is responsible for a majority of the operations within Oracle. User inputs are passed to the Logic Class from the **GuiController**. The **Logic** Class then determines which command to execute based on the user input. The individual commands themselves call upon the **Storage** Class to store data if necessary.

After the command is executed, the **Model** is updated with the relevant fields and the updated **Model** is then returned to the **GuiController** for displaying.

3.2.1 Initialisation

When the **Logic** Class is initialised, it instantiates a new **Storage** and **Model** Object. These Objects are shared across the different commands to ensure that all commands utilise the same, most updated **Storage** and **Model** data.

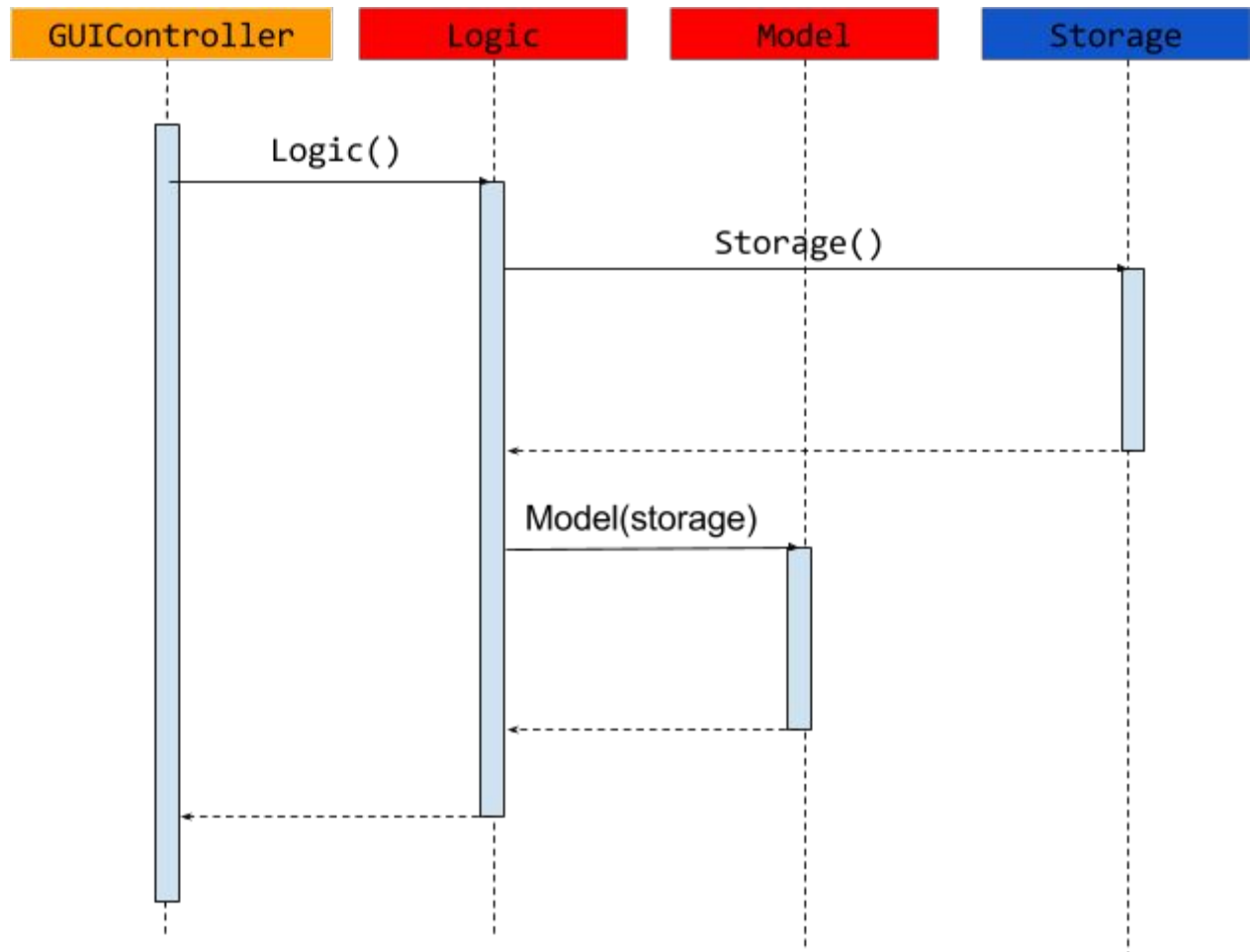


Figure 3.2.1.a Initialisation Sequence Diagram

1. When **Logic** is initialised, a common **Storage** class is first instantiated.
2. Next, a **Model** Object is instantiated with the freshly created **storage** Object assigned to it.

3.2.2 Executing commands

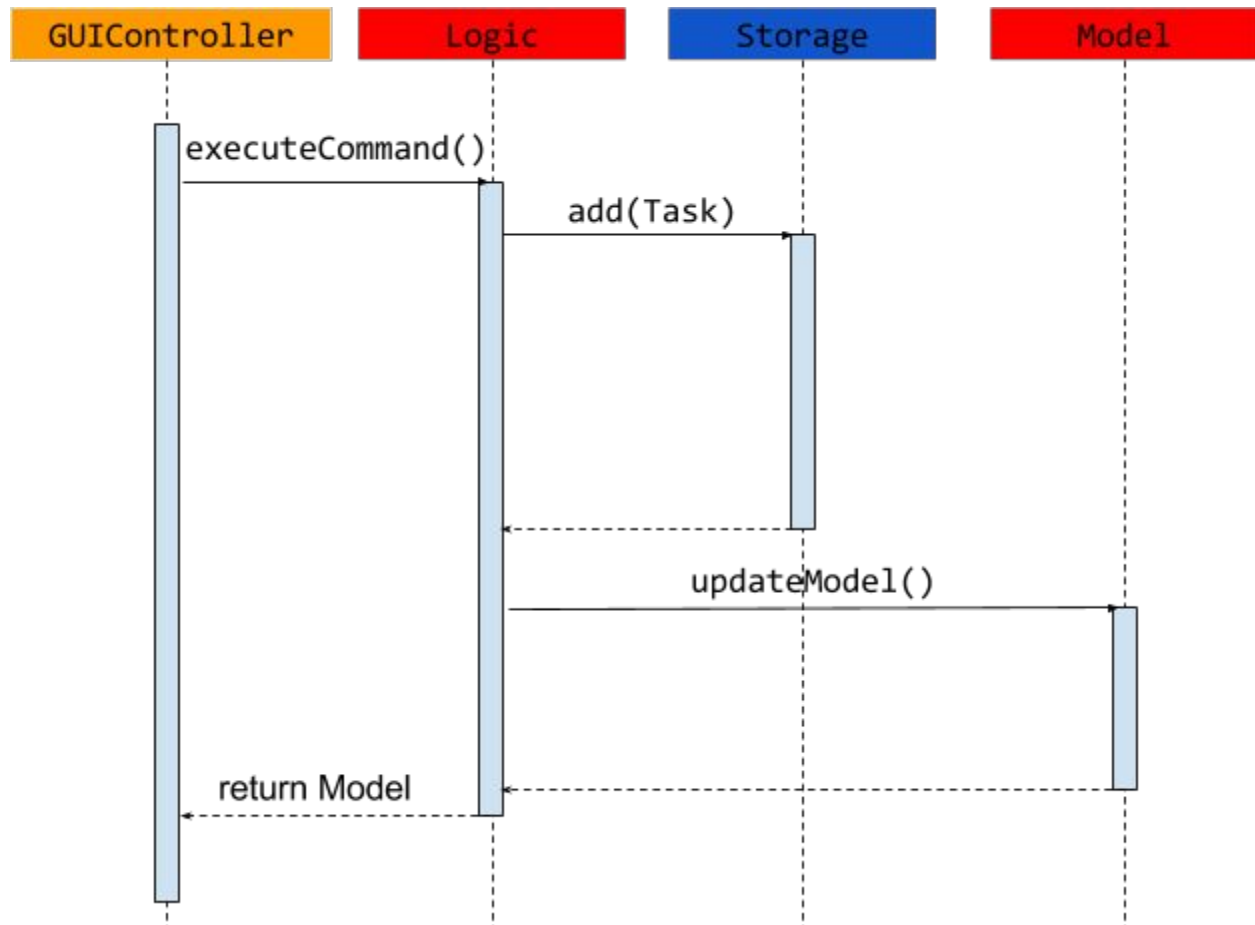


Figure 3.2.2.a Sequence Diagram for executing an add command

To illustrate how the logic component works, let's go through how a simple add command would look like through a sequence diagram.

1. `executeCommand` is called with `ParsedCommand` provided as the argument.
2. The corresponding command will then be executed. For this case, the **add** command is executed.
3. Since adding a task involves updating the `Storage`, the `Storage` class' `add` method is called.
4. After a successful add command, the Model then needs to be updated with all the necessary new information. Calling the appropriate `updateModel` method will automatically update all relevant fields. (see the Model class API for more information)
5. `Model` is now updated for the `GuiController` to pick up the new information.

3.2.3 Updating the Model

The `Model` Class acts as the middle man between the Logic and GUI components. The Logic component updates the `Model` Class based on the user input sent through Logic's `executeCommand` method. The GUI then reads the data from the `Model` and displays

In order to keep the `Model` updated, the `Model` Class has certain methods that automate the updating of Oracle's Lists.

Model Class' Notable APIs

returns	Method and Description
void	<code>updateSearch</code> (String message, ParsedCommand searchQuery, List<Task> searchList) : Passes the recent search query into the Model so that it can automatically update the search list through <code>updateSearchList</code> .
void	<code>updateSearchList</code> () : updates the search list based on the recent search query
void	<code>updateModel</code> (String message) : updates all Model fields and changes the console message to the new message.
void	<code>updateModel</code> (String message, focusId) : updates all Model fields, change to a new console message(message) and set the task to focus on(focusId). Used for commands like Add and Edit which require refocus.
void	<code>updateModel</code> () : updates all Model fields.
void	<code>setConsoleMessage</code> (String message) : Changes the console message. Used for commands that do not require updating the Model.

3.2.4 The Undo/Redo stacks

When the undo command is executed:

1. `CommandHistory` is popped to get the command to undo.
2. The popped command's `undo` method is executed.
3. The popped command is also added to the Redo Stack.
4. Redo is marked as a valid command until the next user command(that is not undo).

3.2.5 Extending the Logic Class

If you wish to add more commands to Oracle, understand that the logic component follows the Command Design Pattern. As such, all new Command Classes should implement the Command Interface.

Logic Class' Notable APIs

returns	Method and Description
Model	<code>executeCommand(String input)</code> : Handles the execution of user inputs.
int	<code>getNewId()</code> : Issues the next available ID
boolean	<code>checkID(int id)</code> : Checks if the ID provided is currently assigned to a Task
Task	<code>searchList(List<Task> taskList, int taskId)</code> : searches the list for a particular task given it's ID.
List<Task>	<code>updateTodayList()</code> : returns a list of tasks due today (sorted in increasing order by date)
List<Task>	<code>updateMainList()</code> : returns a list of non-floating tasks that are due from today onwards (sorted in increasing order by date)
List<Task>	<code>updateFloatingList()</code> : returns a list of floating tasks
List<Task>	<code>updateOverdueList()</code> : returns a list of Overdue tasks (sorted in increasing order by date)

3.3. Parser Component

The role of **Parser** is simply to parse String user inputs into `ParsedCommand` objects, with all relevant fields formatted properly and converted to the appropriate object types. Note that this means the syntax of user inputs is entirely up to you. Furthermore, the only method in Parser used by other components is `MyParser`'s `parseCommand`, which is called by **Logic**. **Parser** is thus independent of any other component.

The structure of **Parser** component is as shown in Figure 3.3.1. More detailed views of `InputParser` and `DateTimeParser` are shown in subsequent diagrams.

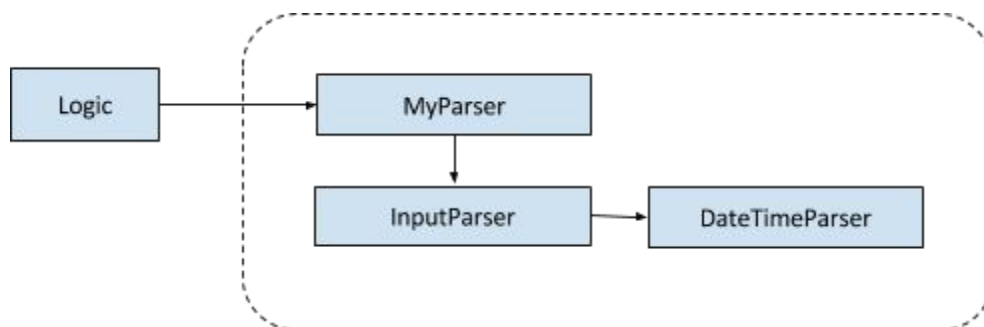


Figure 3.3.1 Component diagram for **Parser**

`MyParser` is at the highest level and contains `parseCommand`, which is called by **Logic** to parse user input to `ParsedCommand` objects. `MyParser` determines the type of command being called by the user and hence calls the relevant parser in `InputParser` to parse the input.

`InputParser` handles the actual parsing through classes that extend it. As date and time parsing are more complicated, we have `DateTimeParser` as a separate class to handle parsing of dates and times.

3.3.1. InputParser

`InputParser` is an abstract class that contains methods to extract and parse specific fields from the strings. It is extended by various parsers that handle parsing for specific command words. By adding parser classes that extend `InputParser`, you will be able to add support for new commands. The structure of `InputParser` and classes that extend it is shown in Figure 3.3.1.1.

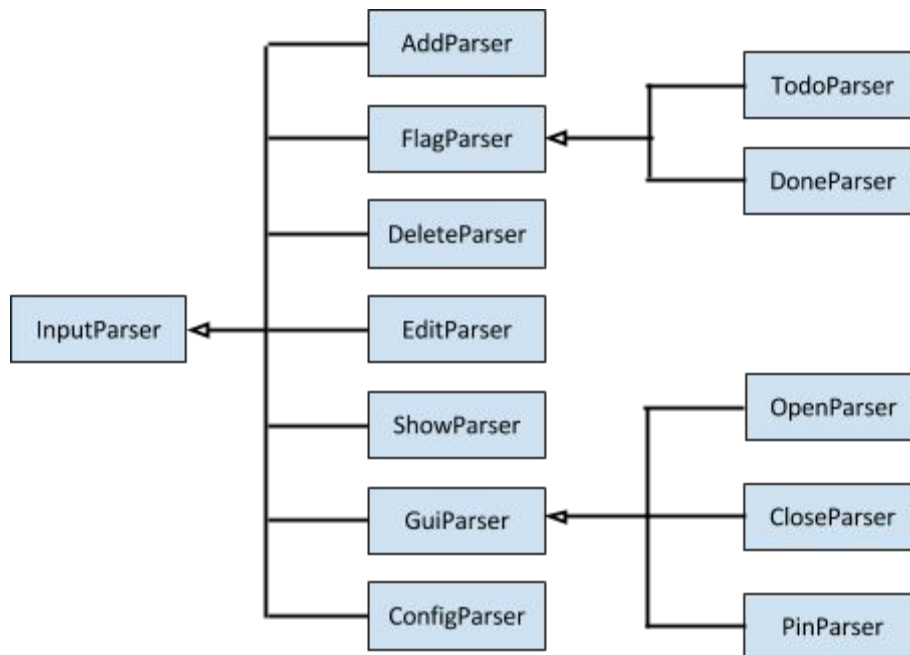


Figure 3.3.1.1 Component diagram for *InputParser*

Each of the parsers return **ParsedCommand** objects, which are objects containing the user input parsed to appropriate formats and object types. If any invalid input is detected, a **ParsedCommand** object with command type ERROR is returned with an error message. Note that **ParsedCommand** uses builder pattern to handle the large number of initialization parameters, so **ParsedCommandBuilder** is used to set the various parameters first before the **ParsedCommandBuilder** object is used to construct a **ParsedCommand** object.

3.3.2. DateTimeParser

DateTimeParser is used by **InputParser** to parse dates and times. There are four classes that extend **DateTimeParser** - **TimeParser**, which handles times, and **FormattedDateParser**, **FlexibleDateParser** and **NattyDateParser**, which handle dates of different formats². The component diagram of **DateTimeParser** is shown in Figure 3.3.2.1.

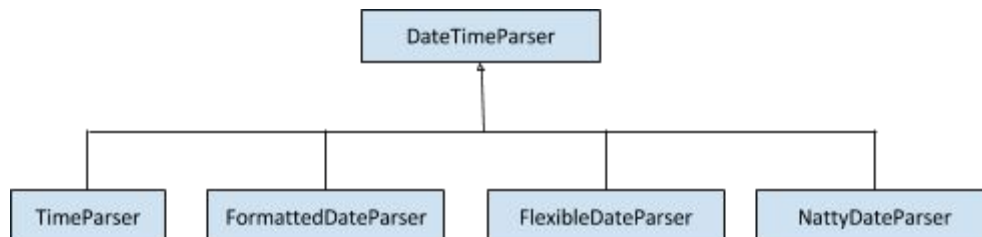


Fig 3.3.2.1 Component diagram for *DateTimeParser*

² The list of supported date and time formats can be found in Appendix F.

The four classes that extend DateTimeParser are:

- **TimeParser**: Handles parsing of twelve and twenty-four hour times.
- **FormattedDateParser**: Handles parsing of dates in numbers, such as 3/2/15
- **FlexibleDateParser**: Handles parsing of dates with months in words, such as 3 Feb
- **NattyDateParser**: Handles relative dates, such as next Monday, using the Natty natural date parsing library

Parsing of dates and times uses the chain-of-responsibility design pattern. **TimeParser** first parses for time. **FormattedDateParser**, **FlexibleDateParser** and **NattyDateParser** are then called in order. If dates are detected (whether valid or invalid), parsing stops at that particular parser, as can be seen in Figure 3.3.2.2, and a **DateTime** object is returned.

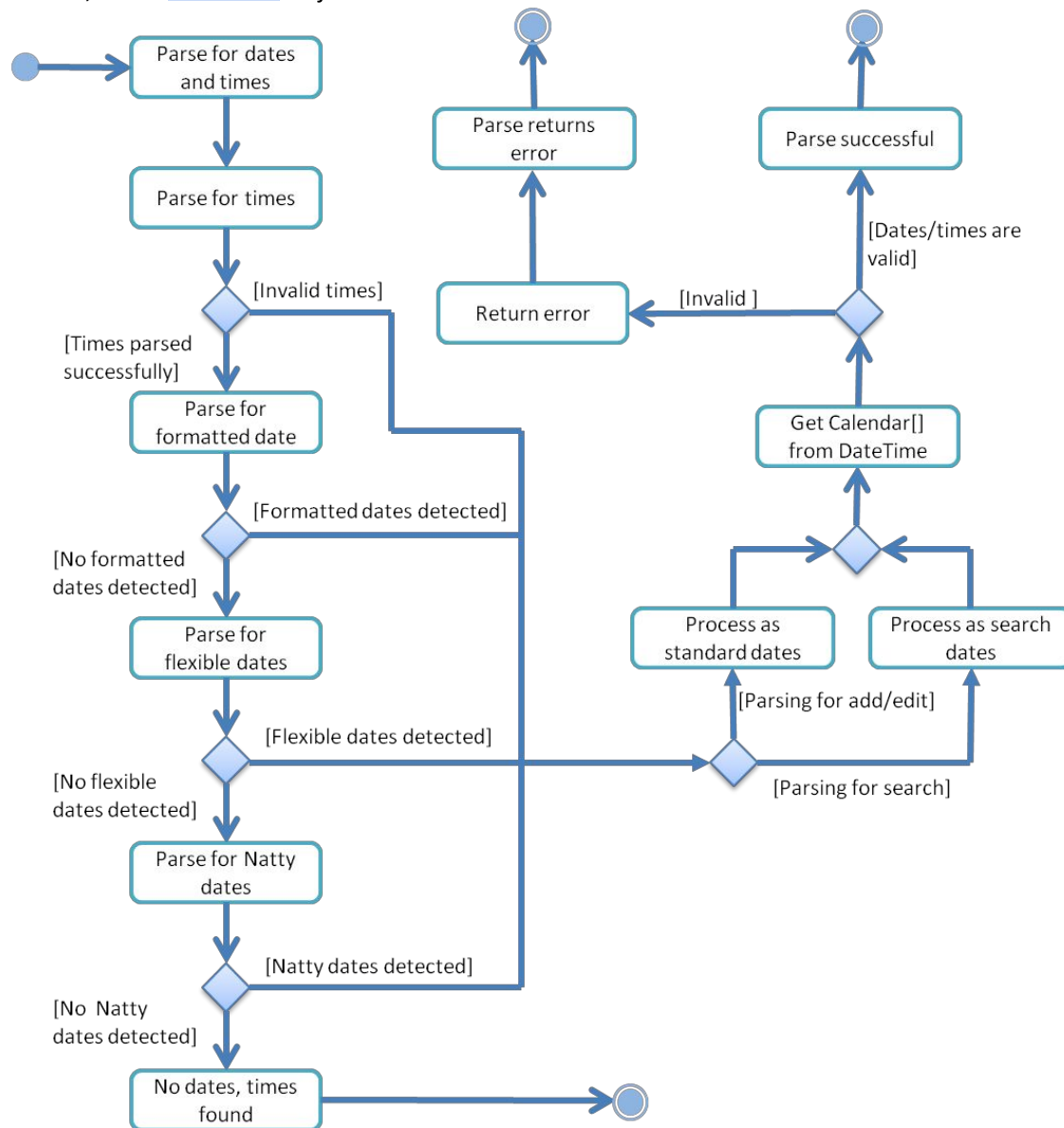


Figure 3.3.2.2: Activity diagram for date time parsing

`DateTime` objects are used to store parsed input during date and time parsing. Originally, builder pattern was intended to allow step-by-step construction of `DateTime` objects. However, a restructure of the date and time parsers has rendered the builder pattern unnecessary. Due to time constraints, this builder pattern has not been removed.

Note that most dates and all times are handled by our own parsers (`FormattedDateParser` or `FlexibleDateParser`). Furthermore, not all formats supported by Natty are supported by Oracle. Natty is very flexible, which means some inputs which are expected to return errors fail to do so. Thus, we have chosen to reduce the number of acceptable formats in exchange for better error detection, by processing the user input in `NattyDateParser` before using Natty's methods. To support a wider range of formats, you will need to edit the `standardizeNattyInput` method. Other issues found with using `Natty` are listed within Appendix E.

3.3.3. Other useful knowledge

As parsing makes extensive use of regex, it would be good to learn how to use regular expressions effectively, and to familiarise yourself with the `Java Pattern API`³. The only external library used by `Parser` is `Natty`. Useful resources may be found in Appendix G.

Notable APIs

returns	Method and Description
ParsedCommand	<code>parseCommand(String userInput)</code> : Parses string input and returns <code>ParsedCommand</code> object with appropriate fields filled in; returns <code>ParsedCommand</code> of type Error if user input is invalid

3.4. Storage Component



Figure 3.4 Storage Component

The role of `Storage` is to take in data provided by `Logic` and save it to a locally stored file. It contains the class `Storage`, which handles and saves the path details for the file's location, as well as application configuration.

Currently, user data and path details are saved in different text files. The default filenames are defined in this class. The `GSON` library is used to make these files human-readable and editable.

Upon instantiation, the `Storage` class reads the text file of tasks and stores the tasks in a `List<Task>`.

Notable APIs

returns	Method and Description
---------	------------------------

³ <https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>

void	<code>add(Task task)</code> : Saves the task
void	<code>delete(int taskID)</code> : Deletes the task with id equal to <code>taskID</code>
void	<code>addPath(String newPath)</code> : Inserts <code>newPath</code> into Path.txt
void	<code>copyAndDelete(String oldPath, String newFilePath)</code> : Copies the file from <code>oldPath</code> to <code>newFilePath</code> and deletes the old one

The output will be displayed in the format that **GSON** uses, which is simply {key:value}. An example of output by **GSON** for a **Task** is:

```
{"name":"shopping","details":"","id":1,"isCompleted":false,"tags":[],"taskType":1}
```

4. Testing

OraCle uses JUnit to perform unit tests on Logic, Parsing and Storage components based on JUnit4 framework. All unit tests are placed in the **OraCle/src/test** folder. Run the AllTestSuite.java Test Driver in order to run all test files. Do note that you are required to copy over all Test_Data files in **OraCle/src/test/Test Data** onto the OraCle/src directory in order for the tests to work.

If you encounter any fail cases when running those unit tests, try removing config and Data.txt files.

Possible loss of Database data during and after tests
 Do a backup of the database files:Data.txt and config before starting any test if you want to keep the contents of the files.

5. Future Development

1. Customisable Input Format
One area we hope to work on is increasing the flexibility of user input by allowing users to set their own preferred formats. For example, by allowing users to create their own keyboard and command shortcuts, or by allowing users to set their preferred date formats (say, for example, to 12/25/15 instead of 25/12/15).
3. Personalised Display
We hope to increase customisability of OraCle in terms of display as well, such as by allowing users to choose their own colour schemes, or arranging the layout of the display.

4. APPENDIX

Appendix A: User stories. As a user, ...

[Likely to be implemented]

ID	I can ... (i.e. Functionality)	so that I can... (i.e. Value)
add	add a task by specifying task only	record tasks that I want to do someday
search	search a task	quickly find a specific task
update	update the task	add more details to a task when necessary
edit	edit the task	change details of a task
delete	delete the task	cancel specific tasks
undo	undo the task to last previous version	quickly revert any mistakes made
sort	sort the tasks	view all completed or uncompleted tasks
sync	choose to store file in a synced folder	view it anywhere on any platform
display	view a list of tasks	see all my pending tasks
recurTask	set a task to repeat after a set period of time	set events that occur regularly

[Unlikely to be implemented]

ID	I can ... (i.e. Functionality)	so that I ... (i.e. Value)
voiceCommand	search by voice	

Appendix B: Non-functional requirements

- Launching the program takes at most 1 second
- Commands take at most 1 second to execute
- Commands that are intuitive to the user (non linux-esque)

- Program can be run on any operating system that supports Java applications (Windows, Linux, Mac OSX)

Appendix C: Product survey

Product: Todo.txt

Documented by: Jia Min

Strengths:

- is available on different OS platform: windows, mac, ios, android
- Sync with Google Tasks, Google Docs or even local and remote Git repository
- Set the priorities of todo items
- Keep track of current work, set desktop notification

Weaknesses:

- no standalone, need to install additional software, like: git or cygwin
- poor GUI

Product: Google Calendar

Documented by: Jia Min

Strengths:

- Good UI
- Quick add, powerful search function
- Import calendar

Weaknesses:

- Need to sign in by Google account
- The todo item must contain date and time

Product: Wunderlist

Documented by: Ka Yi

Strengths:

- one line input
- flexible display, allows user to rearrange tasks
- can add files, description, comments to tasks
- has timed reminders
- supports subtasks
- has multiple lists

Weaknesses:

- difficult to add recurring tasks (can only be done in side pane, not in the one line input)
- deleting and editing requires 2 steps
- cannot delete multiple items at a time
- cannot find tutorial

Product: Apple iCal

Documented by: Terence Kong

Strengths:

- Good UI
- Quick add is intuitive and easy to use
- Can be more or less operated by the keyboard alone
- Can import Calendars from other apps (like google Calendar)

Weaknesses:

- all events added to the calendar need a start/end time (Can't support floating tasks)

- Navigating the GUI with the keyboard is slow (since it's supposed to be a mouse based UI)
- Month tab doesn't seem very useful especially when you have a lot of events stacked up

Product: in-built Phone Calendar app

Documented by: Darryl

Strengths:

- Can be used on the go, lightweight and simple
- Really simple GUI

Weaknesses:

- Difficult to perform actions that require a bit more finesse (like adding events with a lot of details)
- Difficult to view tasks in the bigger picture (like in conjunction with a Calendar open or when you have a long list of events in a single day) due to the limited screen size
- Keyboard also takes up half the screen, making inputting new entries somewhat troublesome.

Appendix D: List of features

List of all features:

- Basic features**
 - Add** : Users are able to add floating tasks, deadline tasks and events using the same command ('add'), with reasonable flexibility (any order of input, flexible date inputs)
 - Edit** : Users are able to edit details of tasks/events and mark tasks as completed
 - Delete** : Users are able to delete tasks/events
 - View** : Users are able to view tasks/events in detail
- Advanced features**
 - Search** : Users are able to search tasks/events containing keywords, as well as filter tasks/events by date/time, tags and status
 - Sync** : Users are able to access task list on multiple computers
 - Keyboard shortcuts** : Users can launch program with a keyboard shortcut
- Special feature chosen: Graphical User Interface**
 - Supports viewing tasks/events in list and calendar format
 - Customizable avatar
- Extra features:**
 - Tags** : Users are able to tag tasks for easier organization
 - Command shortcuts** : Advanced users may use command shortcuts to reduce time spent typing inputs

Appendix E: Issues with Natty

Problems encountered while using Natty as parser include the following:

- Parses "**31 April**" as "**1 May**" (misinterpreting 31 April as 31 days from start of April), instead of providing an invalid date error.
- Parses dates in mm/dd/yy format instead of the typical format dd/mm/yy used in Singapore.
- Cannot handle use of '.' as date delimiters (e.g. [2.3.13](#))

4. Returns wrong date/time if 'from' is used (e.g. 2 jan from 2pm to 3pm).
5. Returns wrong date/time if whitespaces are missing (e.g. 2-4pm instead of 2 - 4pm).

Appendix F: Supported date/time formats for Parser

Time	Format	Examples	Status
12 hour formats	h:mm a - h:mm a, h.mm a - h.mm a	8:30 - 11:00 am, 3.30 pm - 5 pm	Done
	h:mm a ... h:mm a, h.mm a ... h.mm a	8:30 am to 9 pm, 3pm until 10.30 pm	Done
24 hour formats	HHMM -HHMM h	0830-1100h	Done
	HHMM h ... HHMM h	0830h to 1000 h	Done

Date	Format	Examples	Status
Date-Month-Year	dd/mm/yy, dd/mm/yyyy, dd/mm, dd.mm.yy, dd.mm.yyyy, dd-mm-yy, dd-mm-yyyy	2/3/15, 17/4, 12.3.2015, 11-12-2015	Done
	d MMM, d MMM yyyy	3 Mar, 3rd March, 3 Mar 2015, 3rd March, 2015	Done
Month-Date-Year	mm/dd/yy, mm/dd/yyyy, mm/dd, mm.dd.yy, mm.dd.yyyy, mm.dd, mm-dd-yy, mm-dd-yyyy, mm-dd	12/25/15, 3.27.2015, 4-30	will not be supported (requires allowing user to customise preferred date format)
	MMM d, MMM d yyyy	Mar 3, Mar 3 2015, March 3 2015	Done
Year-Month-Date	yy/mm/dd, yyyy/mm/dd, yy.mm.dd, yyyy.mm.dd, yy-mm-dd, yyyy-mm-dd	99/10/31, 2015/3/29, 15-5-21	Not done. yy will not be supported (requires allowing user to customise preferred date format)

*Unlisted formats are not supported

Appendix G: Useful resources

Tutorial for learning regex: <http://www.regular-expressions.info/tutorial.html>

Tool for debugging regex: <https://regex101.com/>