MOUSTASK

			
Ching Sieh Kae Team Lead, Lead Programmer	Shawn Lee Lead Tester, Constraints Planner	Arnold Koroa Resources Manager, Tester	Loy Yusong Documentation, Deadline Watcher

Contents

Contents 2

1. User Guide 4

1.1 Introduction of Moustask 4

1.2 Target Users of Moustask 4

1.3. Important Information Regarding Commands and Sub-Commands 4

1.4. Requesting Help 4

1.5. Adding a Task 4

1.5.1. Adding a Timed Task 4

1.5.2. Adding a Deadline Task 5

1.5.3 Adding a Floating Task 5

1.6. Displaying All Tasks 5

1.7. Searching For Tasks 5

1.7.1. Searching For Tasks by Keywords 5

1.7.2. Searching For Tasks by Date 6

1.7.3. Searching For Tasks by Venue 6

1.7.4. Searching For Tasks by Categories 6

1.7.5. Searching For Tasks by Status 7

1.7.6. Searching For Tasks Using Multiple Commands 7

1.8. Editing A Task 7

1.9. Deleting A Task 8

1.10. Clearing All Tasks 8

1.11 Undoing A Command 8

2. Developer Guide 9

2.1. Purpose of Developer Guide 9

2.2. Architecture 9

2.2.1. Command Line Interface (CLI) 9

2.2.2. Logic 9

2.2.3. Storage 9

2.3. Design Descriptions 9

2.3.1. Helper Classes 9

2.3.1.1. Task Classes 10

2.3.1.2. Command Classes 10

2.3.1.3. TypeTaskPair Class 11

2.3.2. Main Components 12

2.3.2.1. CLI 12

2.3.2.2. Logic 12

2.3.2.3. Storage 12

2.4. Future Work 13

2.5. Change Log 13

1. User Guide

1.1. Introduction of Moustask

Moustask is a to-do task list manager. Moustask allows the user to arrange his schedule on-the-go efficiently and swiftly. The users can store 3 types of tasks in Moustask. They are timed tasks, deadline tasks and floating tasks. Timed tasks are tasks which have a start and end date. Deadline tasks are tasks which only have an end date and floating tasks are tasks with no start and end date. With these 3 kinds of tasks, the users can easily search for tasks to fit into his or her schedule.

1.2. Target Users of Moustask

The target users are users who spend as much time offline as they do online. Hence no installation or internet connection is needed to use Moustask. Another feature of Moustask is that it only accepts commands typed using the keyboard and hence no mouse is needed. People who spend a lot of time managing their tasks often find it a hassle to have to interchange the use of mouse and keyboard to arrange their schedules, thus if all they have to use is the keyboard the users will be able to save time when managing their schedules.

1.3. Important Information Regarding Commands and Sub-Commands

Please take note that all commands and sub-commands should be prefixed with a '.' (dot). The program will not recognize the commands entered otherwise.

1.4. Requesting Help

Usage: *.help*

To display all the commands that can be used, type the command *.help* and enter.

1.5. Adding a Task

Usage: *.add <event> .at <venue> (RED FIELDS ARE OPTIONAL)*

To add a task, type the command *.add* followed by the description of the task. After typing the description, you may choose to enter a venue by typing *.at* and the name of the venue. You may also choose not to enter a venue.

There are 3 types of task that you can add. Timed task are tasks which have a start time and date and an end time and date, deadline tasks are tasks which only have an end time and date and floating tasks, which are tasks with no start and end time and date.

1.5.1. Adding a Timed Task

Usage: *.add <event> .at <venue> .from <YYYY-MM-DD> <HOUR MINUTES> .to <YYYY-MM-DD> <HOUR MINUTES>*

To add a timed task, type the command *.from* after the description (or venue name) and type the start date in the format YYYY-MM-DD (E.g. 2012-12-25) and the start time in the 24-hour format (E.g. 23:30 for 11.30 pm). Type the command TO after that and type the end date and time in the same format as mentioned previously.

Example: *.add must finish CS2103T online assignment .from 2012-12-20 13:00 .to 2012-12-31 24:00*

1.5.2. Adding a Deadline Task

Usage: *.add <event> .at <venue> .by <YYYY-MM-DD> <HOUR MINUTES>*

To add a deadline task, type the command *.by* after the description (or venue name) and type the end date in the format YYYY-MM-DD (E.g. 2012-12-25) and the end time in the 24-hour format (E.g. 23:30 for 11.30 pm).

Example: *.add Project Due Date .by 2012-12-20 12:30*

1.5.3. Adding a Floating Task

Usage: *.add <event> .at <venue>*

To add a floating task, simply enter the command after typing the description (or venue name).

Example: *.add Read books*

1.6. Displaying All Tasks

Usage: *.display*

To display all tasks entered into the list so far, simply type the command *.display* and enter. Tasks displayed will be numbered starting from top to bottom.

1.7. Searching For Tasks

Usage: *.search*

Searching for tasks can be done by typing the command *.search*. You can search for tasks using various commands to filter search results. Search results will be numbered starting from top to bottom.

1.7.1. Searching For Tasks by Keywords

Usage: *.search <keywords>*

To search for tasks using keywords, type the keywords to search for after the *.search* command. The tasks with any keywords used appearing in their task description will be displayed.

```
Command: .search meeting

[Displaying 2 task(s)]
1. project meeting | Venue: SoC discussion room 9
Category: TIMED | Status: UNDONE
Start: 2012-10-15 15:00 | End: 2012-10-15 20:00

2. meeting with project tutor for CS2103 | Venue: COM1 basement
Category: TIMED | Status: UNDONE
Start: 2012-10-10 08:00 | End: 2012-10-10 12:00
```

1.7.2. Searching For Tasks by Date

Usage: *.search .by <integer> .days*

Usage: *.search .by <integer> .months*

Usage: *.search .from <YYYY-MM-DD> <HOUR MINUTES> .to <YYYY-MM-DD> <HOUR MINUTES>*

To search for those tasks that have deadlines or will happen within certain days or months, type the command *.by* after the *.search* command followed by the number of days or months. For days, type the command *.days* after the number and for months, type the command *.months* after the number. The tasks that will happen within the number of days or months specified will be displayed.

Example: *.search .by 7 .days*

To search for tasks within a specific time range, the command *.from* after the *.search* command and type the start date in the format YYYY-MM-DD (E.g. 2012-12-25) and the start time in the 24-hour format (E.g. 23:30 for 11.30 pm). Type the command TO after that and type the end date and time in the same format as mentioned previously.

```
Command: .search .from 2012-10-10 00:00 .to 2012-10-20 23:59

[Displaying 2 task(s)]
1. project meeting | Venue: SoC discussion room 9
Category: TIMED | Status: UNDONE
Start: 2012-10-15 15:00 | End: 2012-10-15 20:00

2. meeting with project tutor for CS2103 | Venue: COM1 basement
Category: TIMED | Status: UNDONE
Start: 2012-10-10 08:00 | End: 2012-10-10 12:00
```

1.7.3. Searching For Tasks by Venue

Usage: *.search .venue <keywords>*

To search for those tasks that take place at a certain venue, type the command *.venue* after the *.search* command followed by the name of the venue. Those tasks that take place at the venue will be displayed.

```
Command: .search .venue mcdonalds

[Displaying 2 task(s)]
1. meal with daddy | Venue: mcdonalds
Category: TIMED | Status: UNDONE
Start: 2012-10-15 20:00 | End: 2012-10-15 22:00

2. buy happy meal hello kitty doll | Venue: mcdonalds
Category: FLOATING | Status: UNDONE
```

1.7.4. Searching For Tasks by Categories

Usage: *.search .category <timed|deadline|floating>*

To search for tasks that are either timed tasks, deadline tasks or floating tasks, type the command *.category* after the *.search* command followed by the type of the task.

```
Command: .search .category timed

[Displaying 3 task(s)]
1. project meeting | Venue: SoC discussion room 9
Category: TIMED | Status: UNDONE
Start: 2012-10-15 15:00 | End: 2012-10-15 20:00

2. meeting with project tutor for CS2103 | Venue: COM1 basement
Category: TIMED | Status: UNDONE
Start: 2012-10-10 08:00 | End: 2012-10-10 12:00
```

```
3. meal with daddy | Venue: mcdonalds
Category: TIMED | Status: UNDONE
Start: 2012-10-15 20:00 | End: 2012-10-15 22:00
```

1.7.5. Searching For Tasks by Status

Usage: *.search .status <done|undone|impossible>*

To search for tasks that are either done, undone or cannot be done, type the command *.status* after the *.search* command followed by the type of status. The tasks with the specified status will be displayed.

```
Command: .search .status undone

[Displaying 4 task(s)]
1. project meeting | Venue: SoC discussion room 9
Category: TIMED | Status: UNDONE
Start: 2012-10-15 15:00 | End: 2012-10-15 20:00

2. meeting with project tutor for CS2103 | Venue: COM1 basement
Category: TIMED | Status: UNDONE
Start: 2012-10-10 08:00 | End: 2012-10-10 12:00

3. lunch with daddy | Venue: mcdonalds
Category: TIMED | Status: UNDONE
Start: 2012-10-15 20:00 | End: 2012-10-15 22:00

4. buy happy meal hello kitty doll | Venue: mcdonalds
Category: FLOATING | Status: UNDONE
```

1.7.6. Searching For Tasks Using Multiple Commands

Usage: *.search <SEARCHCOMMAND> .or <SEARCHCOMMAND>*

Usage: *.search <SEARCHCOMMAND> .and <SEARCHCOMMAND>*

Usage: *.search <SEARCHCOMMAND> .not <SEARCHCOMMAND>*

To search for tasks using multiple search commands, type the first search command and its required fields followed by the command *.or*, *.and* or *.not*. The command *.or* allows you to search for tasks that matches any of the multiple search fields as defined. The command *.and* allows you to search for tasks that matches all of the search fields. The command *.not* allows you to filter out search results that do not match the search field typed after the *.not* command. You may use as many *.or*, *.and* and *.not* commands in a single line of command as you wish.

```
Command: .search .from 2012-10-04 00:00 .to 2012-10-16 23:59 .not .venue mcdonalds .and CS2103

[Displaying 1 task(s)]
1. meeting with project tutor for CS2103 | Venue: COM1 basement
Category: TIMED | Status: UNDONE
Start: 2012-10-10 08:00 | End: 2012-10-10 12:00
```

1.8. Editing a Task

NOTE: You cannot change the type of task to a different category, e.g. changing a timed task to a floating task.

Usage: *.edit <task number> <event> .at <venue> .from <YYYY-MM-DD> <HOUR MINUTES> .to <YYYY-MM-DD> <HOUR MINUTES> .status <done|undone|impossible>*

Usage: *.edit <task number> <event> .at <venue> .by <YYYY-MM-DD> <HOUR MINUTES> .status <done|undone|impossible>*

Usage: *.edit <task number> <event> .at <venue> .status <done|undone|impossible>*

To edit a task, you will first have to search for it using any type of search command. Type the command *.edit* followed by the task number of the task you want to edit as displayed in the search results. Type the new description of the task and the venue name (which again, is optional). Depending on what type of task it is, you will have to type the appropriate commands (*.from*, *.to* for timed tasks, *.by* for deadline tasks and nothing for floating tasks) followed by the new date and time. You may also update the status of the task if you wish. If not, you can simply not type anything after the date and time.

```
Command: .search CS2103
[Displaying 1 task(s)]
1. meeting with project tutor for CS2103 | Venue: COM1 basement
Category: TIMED | Status: UNDONE
Start: 2012-10-10 08:00 | End: 2012-10-10 12:00

Command: .edit 1 meeting with project tutor and module coordinator for CS2103T
Task meeting with project tutor for CS2103 has been updated.

Command: .search CS2103T
[Displaying 1 task(s)]
1. meeting with project tutor and module coordinator for CS103T | Venue: COM1 basement
Category: TIMED | Status: UNDONE
```

1.9. Deleting a Task

Usage: *.delete <task number>*

To delete a task, you will first have to search for it using any type of search command. Type the command followed by the task number of the task you want to delete as displayed in the search results.

```
Command: .search .venue mcdonalds
[Displaying 2 task(s)]
1. lunch with daddy | Venue: mcdonalds
Category: TIMED | Status: UNDONE
Start: 2012-10-15 20:00 | End: 2012-10-15 22:00

2. buy happy meal hello kitty doll | Venue: mcdonalds
Category: FLOATING | Status: UNDONE

Command: .delete 2
Task buy happy meal hello kitty doll has been deleted.

Command: .search .venue mcdonalds
[Displaying 1 task(s)]
1. lunch with daddy | Venue: mcdonalds
Category: TIMED | Status: UNDONE
Start: 2012-10-15 20:00 | End: 2012-10-15 22:00
```

1.10. Clearing All Tasks

Usage: *.clear*

To clear all the tasks entered into the list so far, type the command *.clear* and enter.

1.11. Undoing a Command

Usage: *.undo*

To undo a command entered, type the command *.undo* and enter. The commands that can be undone are add, edit, delete and clear.

2. Developer Guide

2.1. Purpose of Developer Guide

The purpose of this guide is to update an incoming developer to the project on what is the design of the program and how it is being implemented. This guide will help the developer understand the architecture of the program and bring him into the project as soon as possible.

2.2. Architecture

There are three main components in Moustask: Command Line Interface (CLI), Logic and Storage. The architecture of the program is shown in Fig. 1.

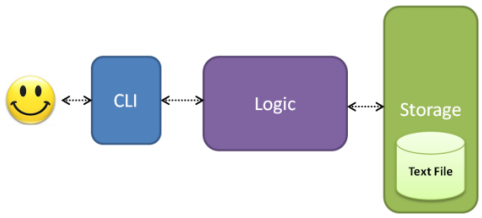

Fig. 1 Architecture of Moustask

2.2.1. Command Line Interface (CLI)

Command Line Interface (CLI) is in charge of listening to user input and passing it to Logic. It is also responsible for receiving the command results from Logic and displaying them to the user.

2.2.2. Logic

Logic is responsible for parsing the user command and invoking the correct operation to the to-do list.

2.2.3. Storage

Storage is responsible for storing and loading the tasks in the to-do list in Logic to/from the database, which is a text file in our current version of the program.

As mentioned, Moustask consists of three main components: CLI, Logic and Storage. Aside from the three main components, Moustask also has several helper classes.

2.3. Design Descriptions

2.3.1. Helper Classes

Helper classes are classes used by the main components to process commands. The understanding of the helper classes will allow you to understand the relationship between CLI, Logic and Storage much better.

2.3.1.1. Task Classes

Each task is stored in a task class. Task classes are AbstractTask, FloatingTask, DeadlineTask and TimedTask. The AbstractTask class defines a generic (parent) task while FloatingTask, DeadlineTask and TimedTask inherit from AbstractTask and defines a specific task type. This relationship between tasks can is summarised in Fig. 2.

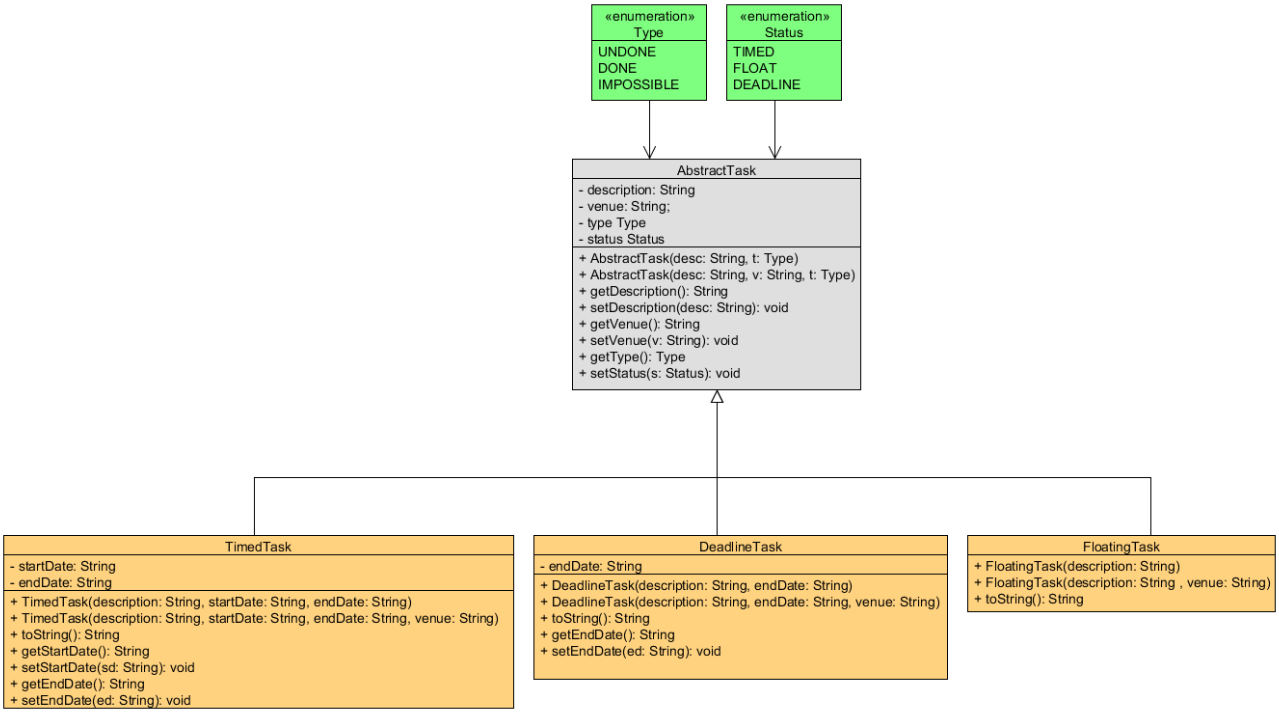

Fig. 2 UML diagram for the task classes

2.3.1.2. Command classes

The command classes are Add, Delete, Edit, Clear, Search. These classes represent the different operations that can be performed to the to-do list and they create and / or modify task objects as when needed. All classes under this category implement either the Command or UndoableCommand interface depending on whether or not the command class can be undone. UndoableCommand in this case does not mean that the command cannot be done; rather it means that the command can be reverted to restore any changes made to any tasks in the list by the command. The way the command classes implement the interfaces is summarised in Fig. 3.

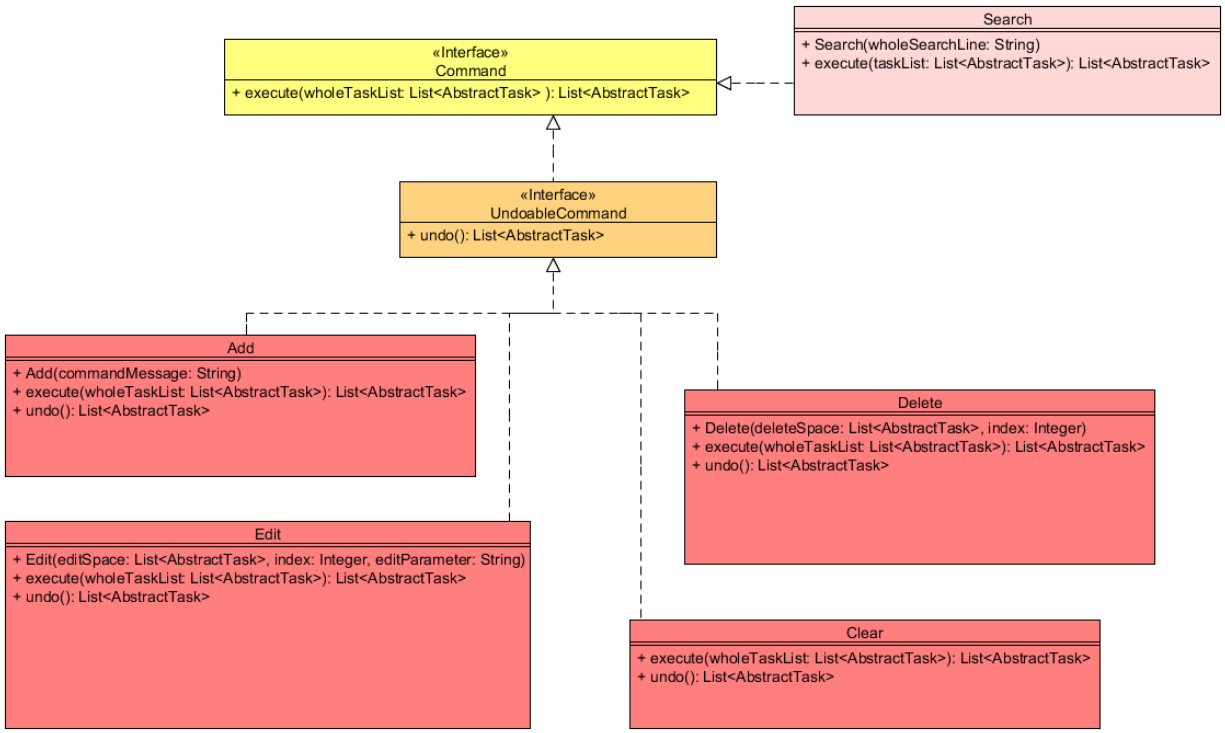

Fig. 3 The command interfaces and the classes that implement them

Important APIs:

Commands implementing the Command interface:

```
public List<AbstractTask> execute(List<AbstractTask> taskList)
```

This command takes the reference to the to-do task list to be read/modified as an argument and returns a list of AbstractTask objects created or modified by this command. Invoking this method causes the command to be executed.

Commands implementing the UndoableCommand interface:

```
public List<AbstractTask> undo()
```

This method returns a list of AbstractTask objects created or modified by this command. Invoking this method on a command object results in any changes to the to-do task list caused by the execution of the command the object represents to be undone.

2.3.1.3. TypeTaskPair Class

This class represents a result of the command issued by the user and the operation that was done to produce this result. This class is returned from Logic to CLI for every command received by Logic such that CLI can display the result properly. The definition of TypeTaskPair is shown in the class diagram in Fig. 4.

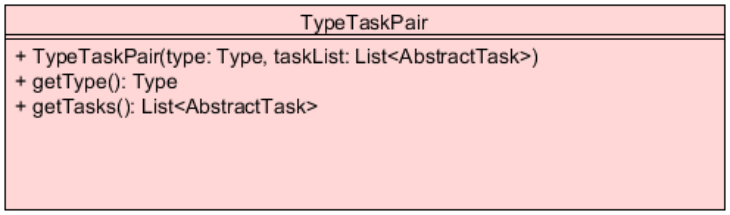

Fig. 4 Class diagram of TypeTaskPair

Important APIs:

`public List<AbstractTask> getTasks()`
Returns the list of results of the operations.
`public Type getType()`
Returns the type of operation done to produce the results stored in the results list.

2.3.2. Main Components

2.3.2.1. CLI

CLI listens to user input, passes it to Logic, and displays the results returned by Logic. CLI will receive a TypeTaskPair object from Logic. CLI will then read the type field in the TypeTaskPair object so that CLI can know what message to display to the user. CLI will then display the tasks contained in the results list of the TypeTaskPair object accordingly.

2.3.2.2. Logic

Logic receives a string from CLI. This string is the command from user input. Logic will then decide the type of operation requested by the user using the first word of the command string. Logic then creates the appropriate Command object, passing the command parameter via its constructor, and call execute() to perform the operation on the task list. From the executed Command object, Logic will receive a list of tasks that has been created / modified by the Command object. The executed Command object, if undoable, will then be pushed into a stack to keep track of the commands executed so that they can be undone if requested by the user. Logic will finally create and return a TypeTaskPair to CLI for display. Additionally, Logic also loads the task list from the database on start-up and stores the task list to the database every time the list is modified. The loading and storing of tasks are done by calling Storage methods.

Important APIs:

`public TypeTaskPair processCommand(String userCommand)`
Receives a string containing the user's command and returns a TypeTaskPair describing the operation done and the results of the operation.

2.3.2.3. Storage

Storage loads and stores task lists from and to an internal database. In the current version of the program this database is simply a text file. At start-up, Logic will call `loadTaskList()`, which will return the tasks to be loaded to a task list in Logic. Whenever the task list is modified, Logic will call `writeTaskList(List<AbstractTask> taskList)` to update the file with the newest task list.

Important APIs

`public void writeTaskList(List<AbstractTask> taskList)`

Receives a task list as argument and saves it to the database.

`public List<AbstractTask> loadTaskList()`

Returns a list of AbstractTask objects loaded from the database.

The relationship between CLI, Logic and Storage is shown in the sequence diagram in Fig. 5.

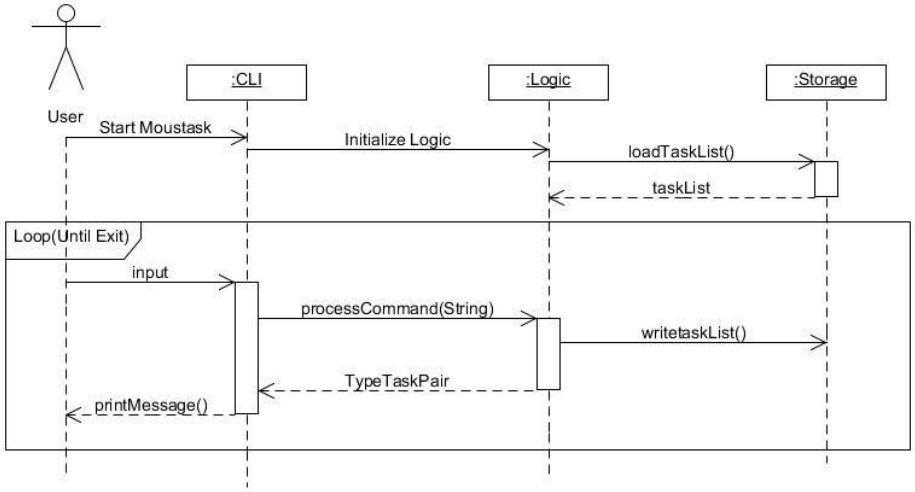

Fig. 5 Sequence diagram showing the relationship between CLI, Logic and Storage

2.4. Future Work

i. Produce unit tests for every component and for the whole program as well

ii. Apply defensive coding

iii. Add logging capabilities

2.5. Change Log

V0.1:

Wrote the first version of the Developer Guide

Produced a prototype of Moustask

Updated the user guide with examples of Search, Edit and Delete

V0.0:

Wrote the first version of the user guide