MOUSTASK

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

TABLE OF CONTENTS

1. User Guide 3
1.1. Introduction of MousTask 3
1.2. Target Users of MousTask 3
1.3. Important Information Regarding Commands and Sub-Commands 3
1.4. Requesting Help 3
1.5. Adding a Task 3
1.5.1. Adding a Timed Task 4
1.5.2. Adding a Deadline Task 4
1.5.3. Adding a Floating Task 4
1.6. Displaying All Tasks 4
1.7. Searching For Tasks 4
1.7.1. Searching For Tasks by Keywords 5
1.7.2. Searching For Tasks by Date 5
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 6
1.7.6. Searching For a Free Slot 7
1.7.7. Searching For Tasks Using Multiple Commands 7
1.8. Editing a Task 8
1.9. Deleting a Task 9
1.10. Clearing All Tasks 9
1.11. Undoing a Command 9
2. Developer Guide 10
2.1. Purpose of Developer Guide 10
2.2. Architecture 10
2.2.1. Command Line Interface (CLI) 10
2.2.2. Logic 10
2.3. Design Descriptions 11
2.3.1. Helper Classes 11
2.3.2. Behavior of Main Components 14
2.3.3. Behavior of Command Classes 16
2.4. Instruction for Testing 17
2.5. Possible Future Work 17
2.6. Change Log 18

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. Therefore 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 between 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.

NOTE: When typing in a date in MousTask, typing in the time is always optional, e.g. 2012-12-25 and 2012-12-25 15:00 are both valid dates.

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

1.5.2. Adding a Deadline Task

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

Usage: .add <event> .by <DAY OF THE WEEK>

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). You may also type in the day of the week that corresponds to the exact date.

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

Example: *.add project .by Tuesday*

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.

Example: *.search meeting*

1.7.2. SEARCHING FOR TASKS BY DATE

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

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

Usage: *.search .by <DAY OF THE WEEK>*

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 or the day of the week. 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*

Example: *.search .by tuesday*

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.

NOTE: When typing in a date in MousTask, typing in the time is always optional, e.g. 2012-12-25 and 2012-12-25 15:00 are both valid dates.

```
Command: .search .from 2012-11-15 .to 2012-11-15
============================================================
                  *** MOUSTASK LIST ***
                     (TOTAL TASKS: 1)
============================================================
TASK #1
meeting with friends
------------------------------------------------------------
FROM  : 2012-11-15 18:00
------------------------------------------------------------
TO    : 2012-11-15 22:00
------------------------------------------------------------
STATUS: UNDONE
```

```
Command: .search .by tuesday
============================================================
                  *** MOUSTASK LIST ***
                     (TOTAL TASKS: 1)
============================================================
TASK #1
eat lunch
------------------------------------------------------------
VENUE : mcdonalds
------------------------------------------------------------
FROM  : 2012-11-13 15:00
------------------------------------------------------------
TO    : 2012-11-13 17:00
------------------------------------------------------------
STATUS: UNDONE
```

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.

Example: *.search .venue mcdonalds*

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
============================================================
                  *** MOUSTASK LIST ***
                     (TOTAL TASKS: 3)
============================================================
TASK #1
project meeting
------------------------------------------------------------
FROM  : 2012-11-10 13:00
------------------------------------------------------------
TO    : 2012-11-10 15:00
------------------------------------------------------------
STATUS: UNDONE
============================================================
============================================================
TASK #2
meeting with friends
------------------------------------------------------------
FROM  : 2012-11-15 18:00
------------------------------------------------------------
TO    : 2012-11-15 22:00
------------------------------------------------------------
STATUS: UNDONE
============================================================
============================================================
TASK #3
eat lunch
------------------------------------------------------------
VENUE : mcdonalds
------------------------------------------------------------
FROM  : 2012-11-13 15:00
------------------------------------------------------------
TO    : 2012-11-13 17:00
------------------------------------------------------------
STATUS: UNDONE
```

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.

Example: *.search .status undone*

1.7.6. SEARCHING FOR A FREE SLOT

Usage: *.search .free <integer> .hours*

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

To search for the earliest free slot available between your tasks, type the command .free after the .search command followed by the number of hours and days required for the free slot. For searching of free hours, type the command .hours after the number and type the command .days after the number for the searching of free days. A free slot will be returned and it will have the label **FREE SLOT**, along with the start date and time of the free slot and the end date and time of the free slot

```
Command: .search .free 2 .hours
============================================================
                  *** MOUSTASK LIST ***
                     (TOTAL TASKS: 1)
============================================================
TASK #1
FREE SLOT
------------------------------------------------------------
FROM  : 2012-11-12 16:17
------------------------------------------------------------
TO    : 2012-11-12 18:17
------------------------------------------------------------
STATUS: UNDONE
```

1.7.7. 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 match any of the multiple search fields as defined. The command *.and* allows you to search for tasks that match 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 .status undone .and eat .not .venue KFC
============================================================
                  *** MOUSTASK LIST ***
                     (TOTAL TASKS: 1)
============================================================
TASK #1
eat lunch
------------------------------------------------------------
VENUE : mcdonalds
------------------------------------------------------------
FROM  : 2012-11-13 15:00
------------------------------------------------------------
TO    : 2012-11-13 17:00
------------------------------------------------------------
STATUS: UNDONE
```

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. You can also choose to enter both dates or either.

```
Command: .search eat
============================================================
                  *** MOUSTASK LIST ***
                     (TOTAL TASKS: 1)
============================================================
TASK #1
eat lunch
------------------------------------------------------------
VENUE : mcdonalds
------------------------------------------------------------
FROM  : 2012-11-13 15:00
------------------------------------------------------------
TO    : 2012-11-13 17:00
------------------------------------------------------------
STATUS: UNDONE
============================================================
                     (TOTAL TASKS: 1)
                *** END OF TASKS LIST ***
============================================================

Command: .edit 1 .venue KFC
Task <eat lunch at mcdonalds from 2012-11-13 15:00 to 2012-11-13 17:00> has been updated to Task <eat lunch at
KFC from 2012-11-13 15:00 to 2012-11-13 17:00>.

Command: .search eat
============================================================
                  *** MOUSTASK LIST ***
                     (TOTAL TASKS: 1)
============================================================
TASK #1
eat lunch
------------------------------------------------------------
VENUE : KFC
------------------------------------------------------------
FROM  : 2012-11-13 15:00
------------------------------------------------------------
TO    : 2012-11-13 17:00
------------------------------------------------------------
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.

```
============================================================
                  *** MOUSTASK LIST ***
                     (TOTAL TASKS: 1)
============================================================
TASK #1
eat lunch
------------------------------------------------------------
VENUE : KFC
------------------------------------------------------------
FROM  : 2012-11-13 15:00
------------------------------------------------------------
TO    : 2012-11-13 17:00
------------------------------------------------------------
STATUS: UNDONE
============================================================
                     (TOTAL TASKS: 1)
                *** END OF TASKS LIST ***
============================================================

Command: .delete 1
Task <eat lunch at KFC from 2012-11-13 15:00 to 2012-11-13 17:00> has been deleted.

Command: .search eat
Your search returns no result.
```

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 the design and implementation of the program. This guide will help the developer understand the architecture of the program so that he can work on 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.

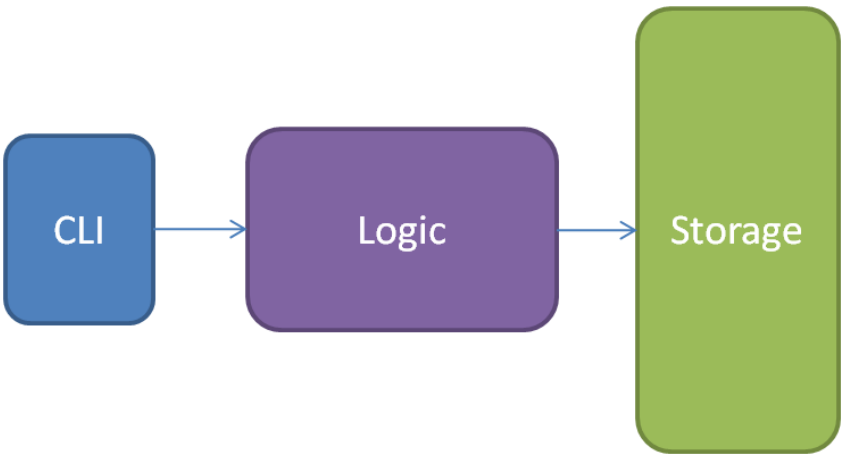

Fig. 1.1 Architecture of MousTask

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. The details for Logic's subcomponents will be discussed in more detail in the next section.

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 (v0.5) of the program.

2.3. Design Descriptions

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

2.3.1. 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 be summarised in Fig. 2.

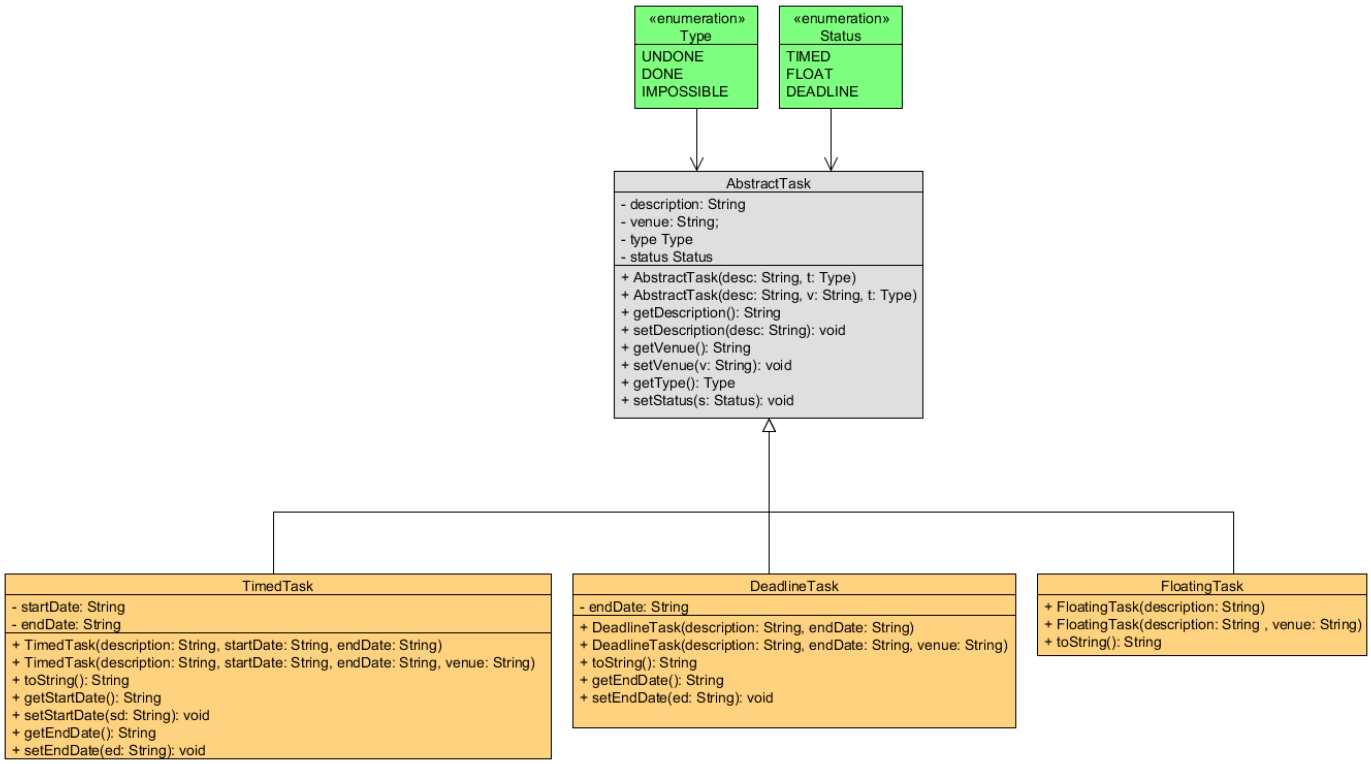

Fig. 2 Class diagram for the task classes

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 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.

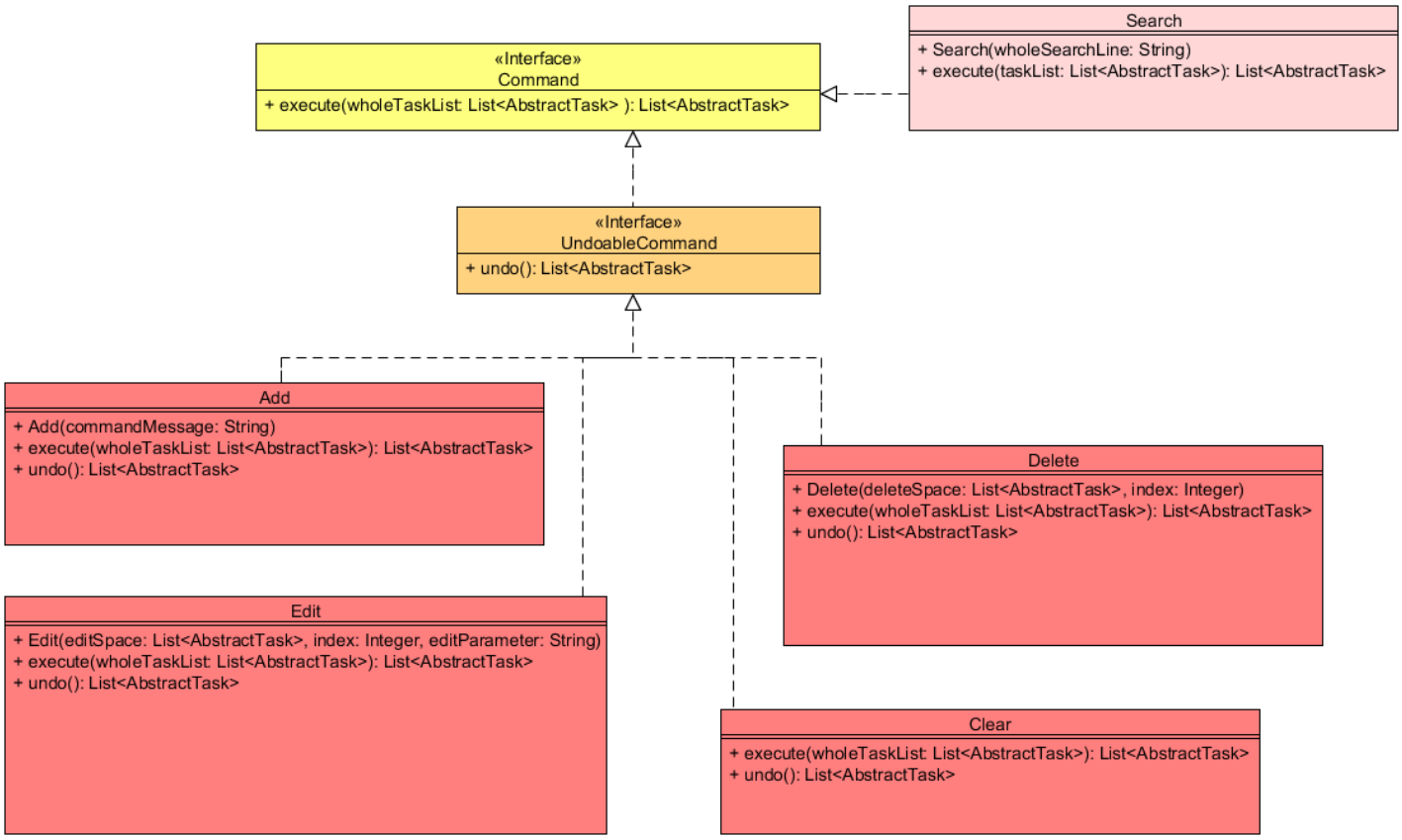

Fig. 3 Relationship between command interfaces and classes

Important APIs for the Command Classes:

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 will revert any changes done to the to-do task list by the command object.

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.

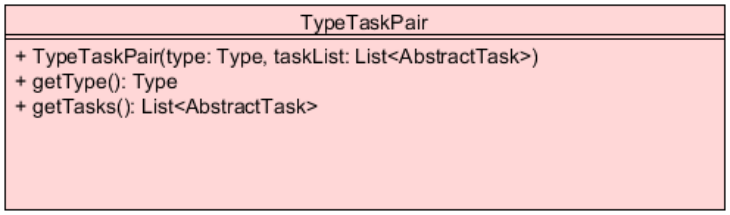

Fig. 4 Class diagram of TypeTaskPair

Important APIs for TypeTaskPair Class:

`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.

2.3.1.4 DateTime Class

This class helps to represent a date and time given a string containing either the exact date or the day of the week. This class is used by many classes where date is involved in the operations, e.g. Search, Edit and Add, to generate dates where the exact date may not be specified by the user. The definition of DateTime is shown in the class diagram in Fig. 5

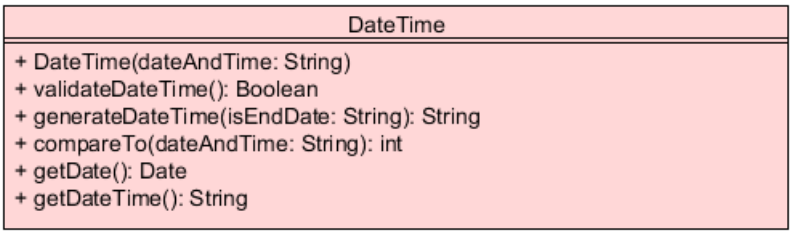

Fig. 5 Class diagram of DateTime

Important APIs for TypeTaskPair Class:

`public boolean validateDateTime()`
Returns a boolean depending on whether the date and time inside the class is a valid date format.

`public String generateDateTime(boolean isEndDate)`
Returns a string containing the date and time, the time depending whether the date is an end date or not.

2.3.2. Behavior of 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 will 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

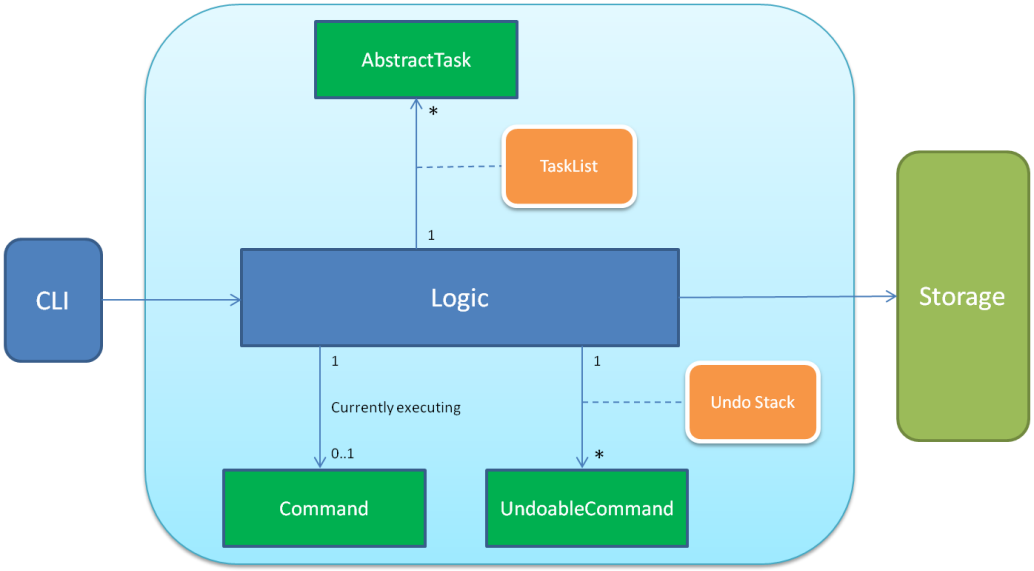

Fig. 6 Association diagram for subcomponents of Logic

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 or 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 for Logic Class:

```
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 (v0.5) 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 for Storage Class:

`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. 6.

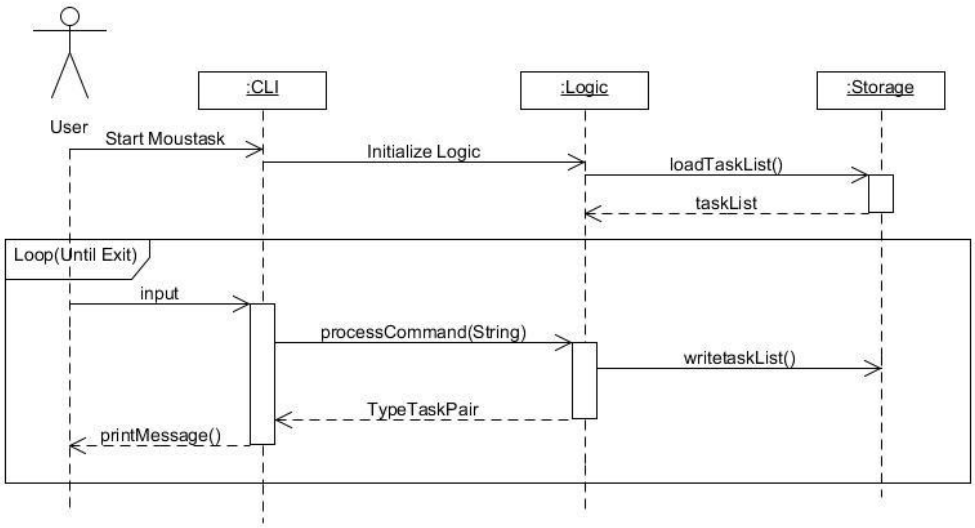

Fig. 7 Sequence diagram showing the relationship between CLI, Logic and Storage

2.3.3. BEHAVIOR OF COMMAND CLASSES

2.3.3.1 ADD

Add receives a string from Logic via Add's constructor. This string is the information of the task that is being added. Upon execution, Logic will pass the task list containing all the tasks to Add and Add will take a number of actions. Firstly, it will check if the string is an empty string. Secondly, it will replace all instances of valid action words in the string with the string "DELIMITER" + action word + the string"DELIMITER". Add will then split the new string into tokens using the string "DELIMITER" as the delimiter and place the tokens in a vector as shown in the code sample in Fig. 7. Add will then decide whether the task is a timed task, deadline task or floating task based on the action words ".from", ".by" or the lack of it in the vector of tokens. Each valid action word will then be processed and the various fields of the task will be added to the task based on the content of the vector. A task list containing the newly added task will be returned to Logic, and the new will also be added to the task list containing all the tasks.

```
private static List<String> splitStringIntoTokens(String messageToAdd,
                    List<String> addTokenList) {
              String[] addTokenArray;

              addTokenArray = messageToAdd.replaceAll("(\\s\\.\\w+)",
                          "DELIMITER$1DELIMITER").split("DELIMITER");

              for (int i = 0; i < addTokenArray.length; i++) {
                    addTokenList.add(addTokenArray[i].trim());
              }
              return addTokenList;
       }
```

Fig. 8 Code sample showing how add splits the string into tokens

2.3.3.2 EDIT

Edit receives a string and a task list containing the search results from Logic via Edit's constructor. Upon execution, Logic will pass the task list containing all the tasks to Edit and Edit will do a numbers of checks to ensure that the task lists are not empty and that the task that is to be edited can be found in the task lists. The string will contain the index number of the task that is to be edited in the search results. Edit will then create two copies of the task and stores one of them internally to be used in the event that `undo()` is executed on Edit. The same task in the task list containing all the tasks is then deleted. Edit will then split the string into tokens using the space character " " as the delimiter and place the tokens in a vector. Edit then checks for the first token that starts with the character '.' and add the tokens before it as the new description for the task. Edit will then update the various fields of the task based on the valid action words and data in the vector. Finally, Edit will return Logic a task list containing the edited task and the original task and also add the new task to the task list containing all the tasks.

2.3.3.3 CLEAR

Clear does not receive anything from Logic when Clear is created. Upon execution, Logic will pass the task list containing all the tasks to Clear and Clear will create a clone of the task list and stores one of them in the event that `undo()` is executed on Clear. Clear then clears the task list that is not stored and returns null to Logic.

2.3.3.4 DELETE

Delete receives an integer and a task list containing the search results from Logic via Delete's constructor. Upon execution, Logic will pass the task list containing all the tasks to Delete and Delete will do a number of checks to ensure that the task lists are not empty and that the task that is to be deleted can be found in the task lists. Delete will then get the task to be deleted from the search results and remove the task from the task list containing all the tasks. Delete then saves a copy of the task that has been deleted in the event that `undo()` is executed on Delete. Finally, Delete will return to Logic a task list containing the task that has been deleted.

2.3.3.5 SEARCH

Search receives a string from Logic via Search's constructor. Upon execution, Search will pass the task list containing all the tasks to Search and Search will split the string into tokens using " " as the delimiter and place the tokens into an array. The tokens in the array will then be concatenated into a string started from the first token to the last until there are no more tokens in the array or a chain command word is encountered. Upon doing so, Search will then execute the type of search based on the concatenated string and checks the task lists containing all the tasks for matches. All matches will be placed in a task list that will represent the search results. If a chain command exists in the array, the array, the task lists containing all the tasks and the search results will be passed to a method that handles chain command execution. What `execute()` will return to Logic after all commands and chain commands has been executed will be the task list containing all the search results that match all the search commands.

2.4. INSTRUCTION FOR TESTING

Our development team relies heavily on JUnit tests for unit and regression testing. For every major change that is made to the code base, the existing JUnit tests are to be run for regression testing and the program should pass all the tests. The EclEmma coverage tool is then used to check how much of the code base is covered by the existing JUnit tests. The coverage of the tests should always be above 90% for all components of the program. If this is not the case, new tests are to be written to cover the uncovered parts of the program.

2.5. POSSIBLE FUTURE WORK

- Add a Graphical User Interface for the program
- Allow even more flexible commands
- Make operations (especially search) more efficient

2.6. CHANGE LOG

V0.5:

- Added a feature that allows the user to search for a free slot
- Added a feature that allows user to search, edit or add by the day of the week
- Updated the presentation of MousTask

V0.4:

- Fixed a bug in logging that causes multiple log files to be created
- Fixed a bug in logging that causes logs to be displayed on console
- Developed unit tests for all components
- Changed how tasks are displayed
- Various bug fixes

V0.3:

- Developed unit tests for some components
- Applied defensive coding for all components
- Implemented logging capability for all components
- Allows addition and editing of dates without specifying time
- Allows addition of dates using day names
- Added a DateTime converter
- Various bug fixes

V0.2:

- Various bug fixes

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