Vivian Lee - Project Portfolio

PROJECT: Duke Email Manager

About the Project

My team consists of 4 software engineering students, who chose to morph the task manager to an email manager called Duke Email Manager, catered exclusively to students in School Of Computing to help them better manage their emails and tasks. The various features of Duke Email Manager includes a task manager and email manager that are switchable between each other, a Graphical User Interface (GUI) that uses Command Line Interface to enter the commands.

This is what our project looks like:

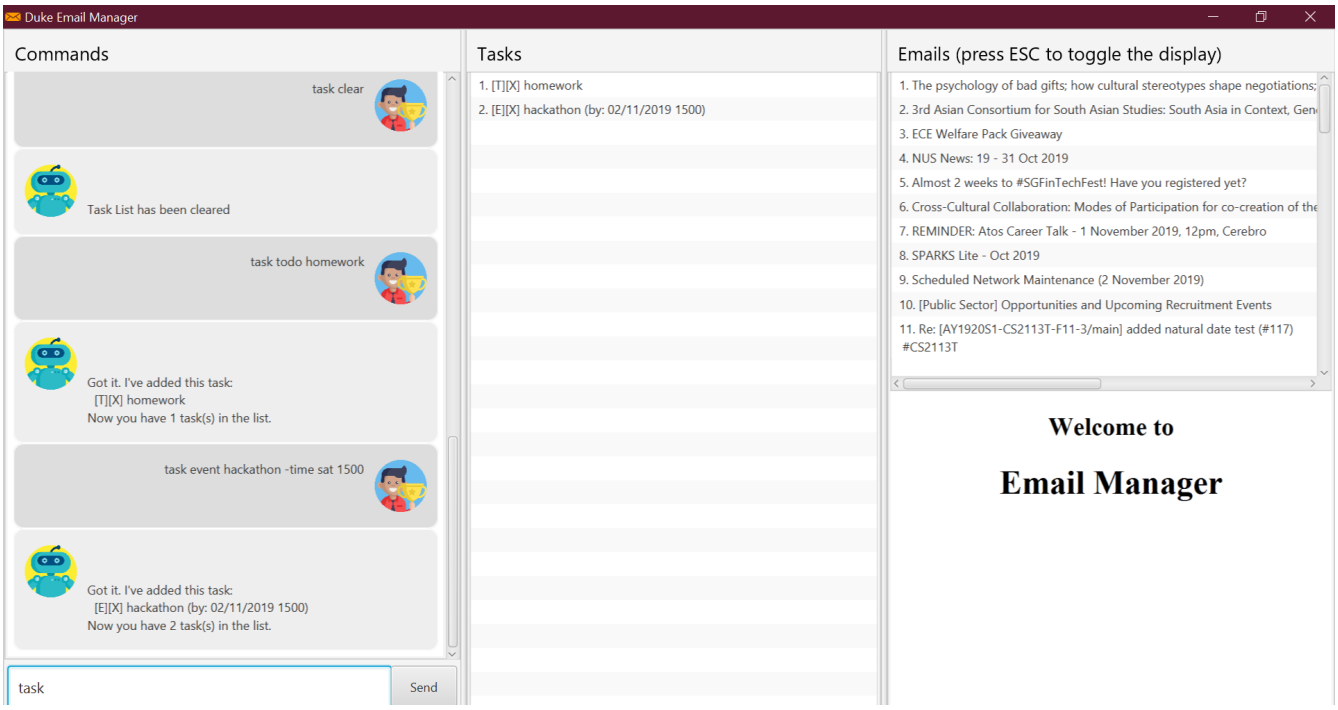

Figure 1. Duke Email Manager

My role was to write codes to enhance the task manager. The following sections illustrate my implementations in more detail, as well as relevant documentation I have contributed to the user and developer guides in relation to these implementations.

Summary of contributions

This section summarizes the contribution I made towards the project. They mainly include, but not limited to coding, documentation, and other helpful contributions to the team project.

Enhancement Added

Major enhancement:

Natural Date Format feature

- **What it does:** Students can define the date (where needed) in terms of a day of the week. The natural date format assumes that the student wants to input the date of the next nearest day of the week.
- **Justification:** Increase flexibility, reduces the time and effort needed to input the date. students.
- **Highlights:** This feature also works well with other commands like `update`, where students can update the date and time of the task using natural date format as well.
- **Code contributed:** #91, #92, #93, #99, #117

Sorting feature

- **What it does:** Students can sort the task list according to how they prefer. The task list can be sorted according to `priority`, `status` or `time`
- **Justification:** Makes it easier for students to find the task when the task list is sorted.
- **Highlights:** The task list is sorted according to time by default. Once the sort command is executed to change the sort type, the task list will be sorted according to the sort type until the user exits the program.
- **Code contributed:** #192

Minor Enhancement:

Priority feature

- **What it does:** Students can set priorities for tasks according to its urgency or severity. (Future implementation: Tasks will be sorted and displayed in the GUI according to the priorities.)
- **Justification:** Student can distinguish higher priority task from lower priority ones.
- **Highlights:** This feature works well with several other commands, like the `update` command, or any of the add task commands (`todo`, `deadline`, `event`)
- **Code contributed:** #79, #121

Snooze feature

- **What it does:** Students can snooze a deadline or event task either by indicating a specific snooze duration or by default of 3 days (when no specific duration is keyed).
- **Justification:** The time of deadline and event task might change, hence students can extend the time by snoozing the task.
- **Code contributed:** #99

Other contributions:

Enhancements to existing features:

- Added key binding to allow scrolling of inputs in the GUI input text field by pressing the `up` and `down` arrow keys: #74
- Added `clear` command on top of having just a `delete` command to clear the task list: #119

Documentation:

- Added documentations for my features in the developer guide and user guide, which include some use cases and test guide.

Community:

- PRs reviewed: #81, #114, #185
- C-Tagging adopted by group mate and was further enhanced to allow for more tags to be included in a task: #60, #62

Contributions to the User Guide

We had to update our original User Guide with instructions for the enhancements we had added. The following is an excerpt from our Duke Email Manager User Guide, showing additions that I have made for some of my features.

Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

Set Priority Command: `set`

Format: `set ITEM_NUMBER -priority PRIORITY`

Sets a priority to a task.

Examples:

`set 1 -priority high`

`set 2 -priority med`

NOTE	The PRIORITY input is restricted to only **high**, **medium**/**med** or **low** (case insensitive). Any other PRIORITY inputted will be invalid.

WARNING	This command can override the priority set to a task by the command `update ITEM_NUMBER -priority PRIORITY` and vice versa. It also overrides the priority set to a task by the `todo`, `deadline` or `event` command, but not the other way round.

Snooze a task: `snooze`

Format: `snooze ITEM_NUMBER [-by NO_OF_DAYS]`

Snoozes the task by the NO_OF_DAYS.

Examples:

`snooze 3`

`snooze 1 -by 4`

NOTE	If the NO_OF_DAYS is not input, the `snooze` command will automatically snooze the task by 3 days. Only tasks of type `deadline` and `event` can be snoozed.

WARNING	Once a task is snoozed, there is no undo option. To modify the date and time of the task, use the `update` command.

Sort task list: `sort`

Format: `sort SORT_TYPE`

This command sorts the task list according to the SORT_TYPE.

Examples:

`sort priority`

`sort status`

NOTE	Task list is sorted according to `time` by default. The list can be sorted by `priority`, `status` and `time`.

Sorted by	How task list is displayed
`priority`	tasks with higher priority will be at the top of the task list

`status`	tasks that are not completed will be displayed at the top of the task list
`time`	deadline or events tasks with nearest date and time will be at the top of the task list.

Clear task list: `clear`

Format: `clear`

This command deletes all tasks in the list.

WARNING	Once executed, you will not be able to undo this command.

TIP	If you execute this command by accident, you may return to your last saved state by closing *Email Manager* without using the `bye` command.

Contributions to the Developer Guide

I mainly contributed to the documentation of the features that I have implemented, which are mainly for the task section. This includes use cases and test guides that are related to my features.

*Given below are an **extract** of what I contributed to the Developer Guide. They showcase my ability to write technical documentation and the technical depth of my contributions to the project.*

Natural Dates Support

Natural dates support helps to speed up the process at which users enter their task details so that their task can be added into the task list quickly.

□□	**This feature only works with `deadline` and `event` tasks.**

Implementation

The following sequence diagram below illustrates how this feature works:

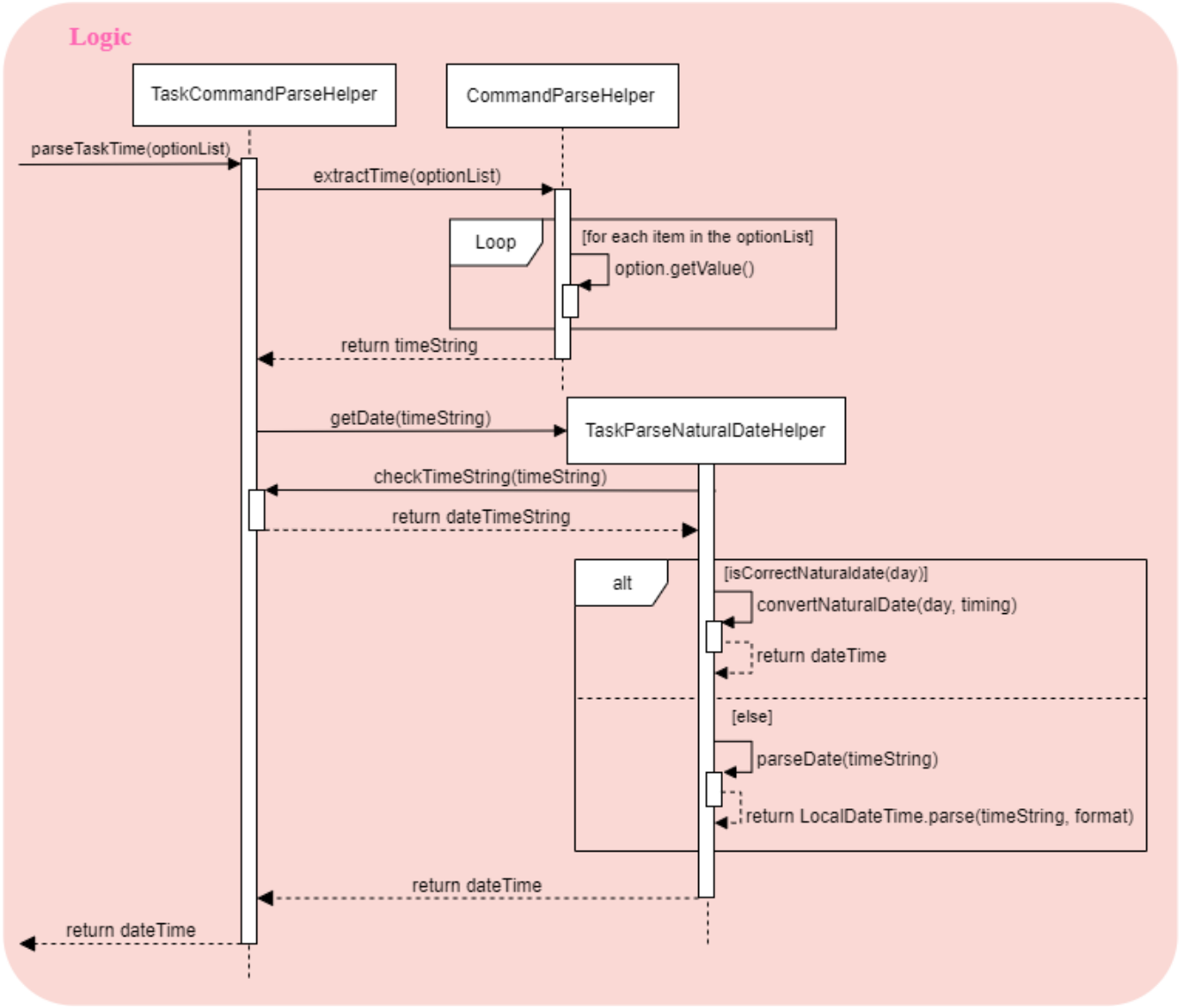

Figure 2. Natural Dates Support Sequence diagram

As seen from the diagram above, the Natural Dates support is facilitated by three classes, namely `TaskCommandParseHelper`, `CommandParseHelper` and `TaskParseNaturalDateHelper`.

Given below is an example usage scenario and how Natural Dates Support behaves at each step.

Step 1: The user launches the application. The input type is currently in `email` mode. The user wishes to add a task and keys in `flip` to switch input type to `task` mode.

Step 2: The user executes `deadline homework -time Mon 1200` to add a new deadline task. `UI` component captures the input and passes to `Logic` component to parse the input.

- `CommandParseHelper` takes in the `input`, parses and extracts the options and stores it inside ArrayList<Option> `optionList`, then passes the `input` and `optionList` to `TaskCommandParseHelper`.

Step 3: `TaskCommandParseHelper` takes in the command, parses and extracts the time string of the task by calling `CommandParseHelper#extractTime(optionList)`.

Step 4: The extracted time string will go through `TaskParseNaturalDateHelper#getDate(timeString)`, which calls `TaskCommandParseHelper#checkTimeString(timeString)` and retrieves dateTimeString = new Pair<>(day, time).

Step 5: If `TaskParseNaturalDateHelper#isCorrectNaturalDate(day)` is true, `TaskParseNaturalDateHelper#convertNaturalDate(day, time)` is called, else `TaskParseNaturalDateHelper#parseDate(timeString)` is called.

Step 6: `TaskCommandParseHelper#parseTaskTime(optionList)` retrieve dateTime from `TaskParseNaturalDateHelper` and returns it.

Sorting of Task List

The task list can be sorted according to what the user prefer.

□□	**Task list can be sorted by `priority`, `status` and `time` only.**

Implementation

The following sequence diagram below illustrates how the sort command is parsed:

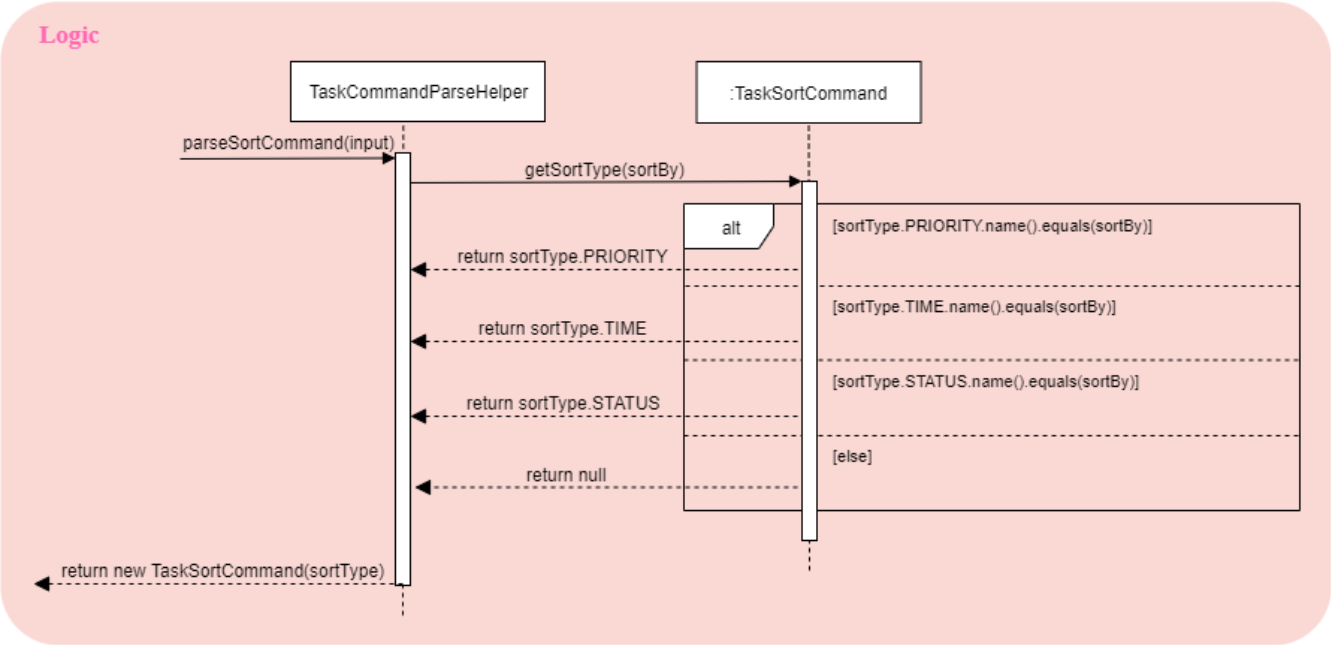

Figure 3. parseSortCommand Sequence diagram

As seen from the diagram above, the parsing of sort command is facilitated by two class, namely `TaskCommandParseHelper` and `TaskSortCommand`.

Given below is an example usage scenario and how `parseSortCommand` behaves at each step.

Step 1: The user launches the application. The input type is currently in `email` mode. The user wishes to check the task list and key in `flip` to switch input type to `task` mode.

Step 2: The user executes `sort priority` to sort the task list according to its priority. `UI` component captures the input and passes to `Logic` component to parse the input.

Step 3: `TaskCommandParseHelper#parseSortCommand(input)` is called and extracts the sort type called sortBy.

Step 4: `TaskSortCommand#getSortType(sortBy)` is then called and returns the sort type according to sortBy

- E.g. If sortBy = "priority", sortType.PRIORITY will be returned.

Step 5: `TaskCommandParseHelper` returns new `TaskSortCommand(sortType)`

The following sequence diagram below illustrates how the sort command works:

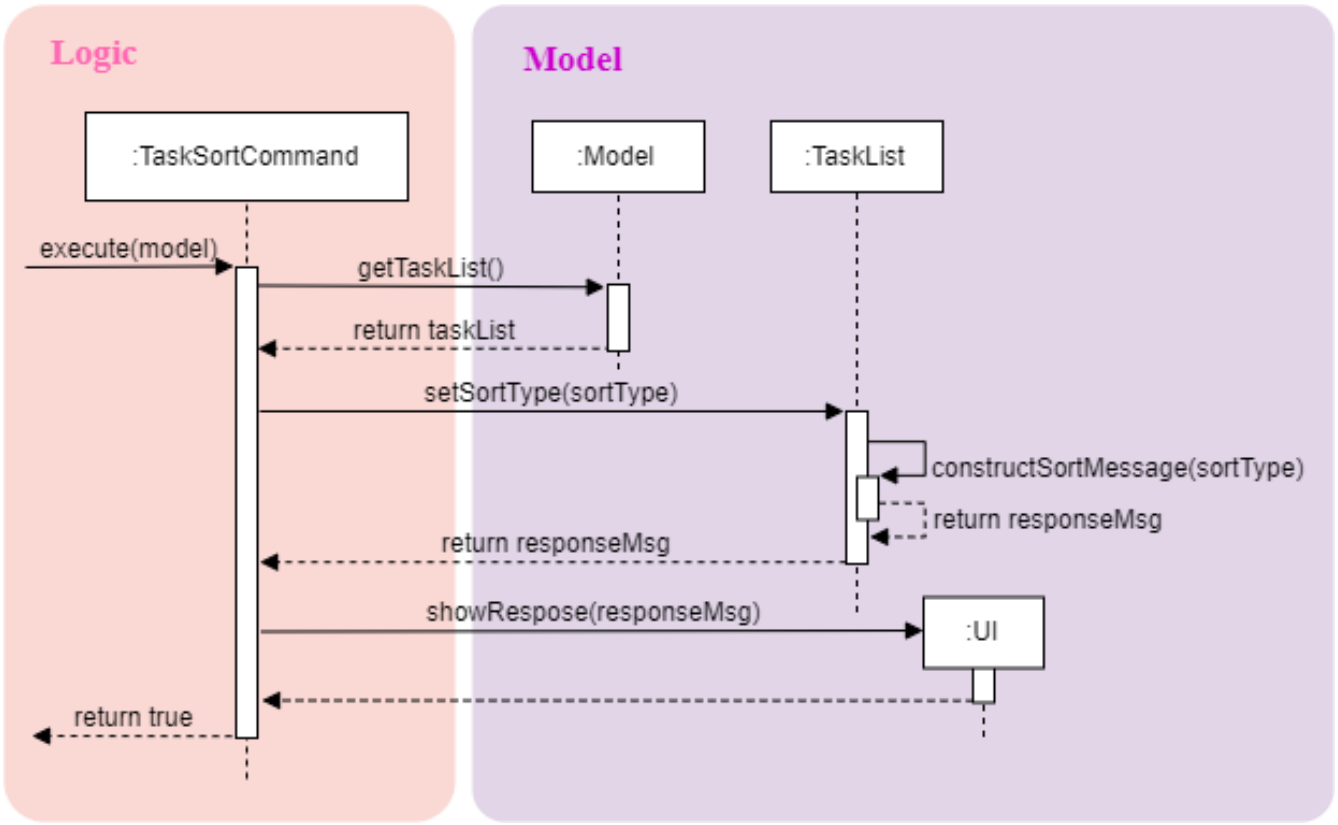

Figure 4. TaskSortCommand Sequence diagram

As seen from the diagram above, the sort command is facilitated by four class, namely `TaskSortCommand`, `Model`, `TaskList` and `UI`.

Given below is an example usage scenario and how `TaskSortCommand` behaves at each step.

Step 1: The user executes `sort status` to sort the task list according to whether it is completed or not.

Step 2: `Model#getTaskList()` is called and the current taskList is returned.

Step 3: `TaskList#setSortType(sortType)` is called and returns a response message after calling `TaskList#constructSortMessage(sortType)`.

Step 4: The UI displays the response message and `TaskSortCommand#execute(model)` returns true if the sort type in `TaskList` is set correctly.

The following activity diagram shows the method `sortByType()` that changes how the task list is sorted according to the sort type:

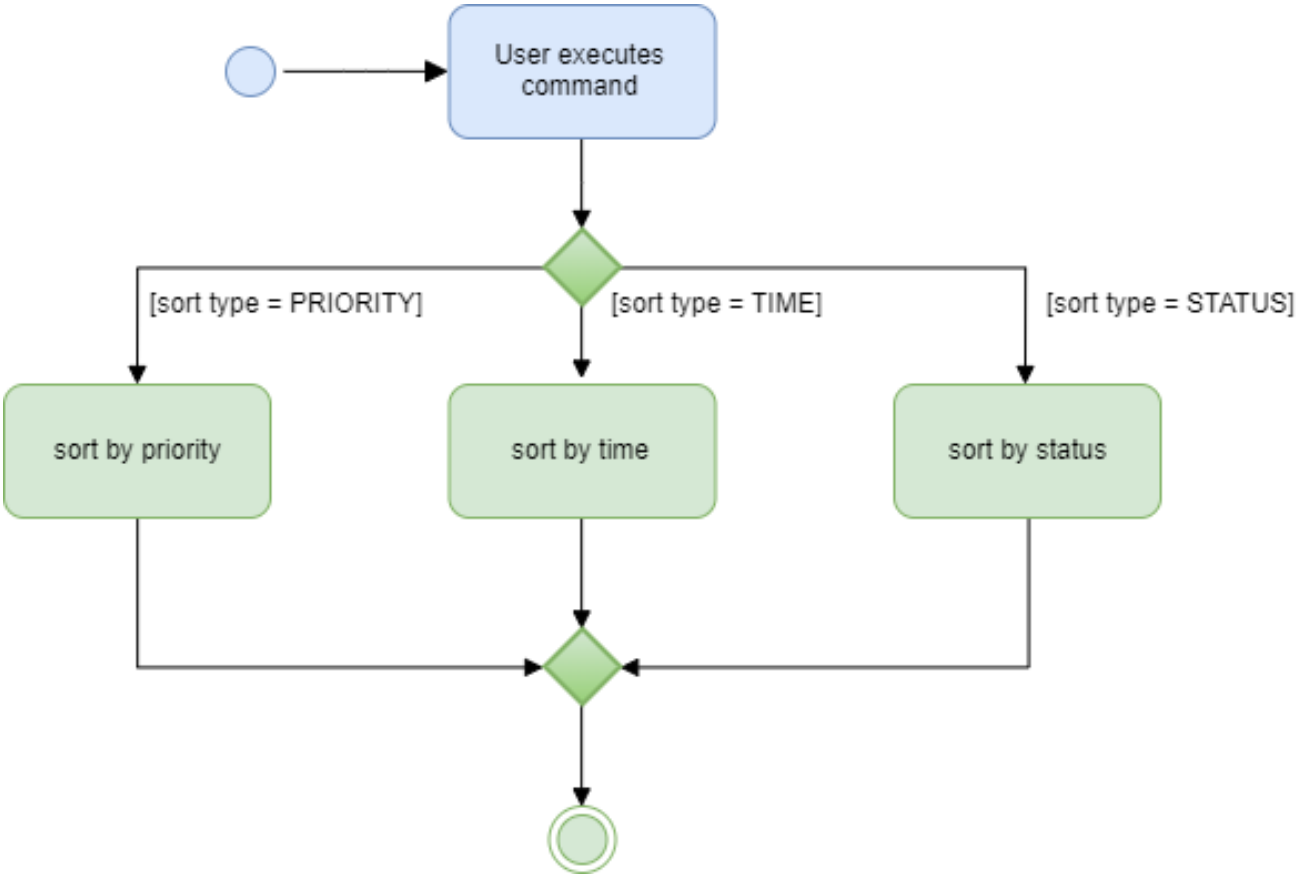

Figure 5. sortByType activity diagram

□□	**The task list is sorted according to time by default.**

Given below is an example usage scenario and how `TaskSortCommand` behaves at each step.

Step 1: The user wishes sort the task list according to the priority level of the tasks and executes `sort priority`.

Step 2: The sort type is changed to `PRIORITY` from `TIME`, and the sorting method is changed.

Step 3: `sortByType() is called by `Model#updateGuiTaskList()` on every user input to keep task list sorted.