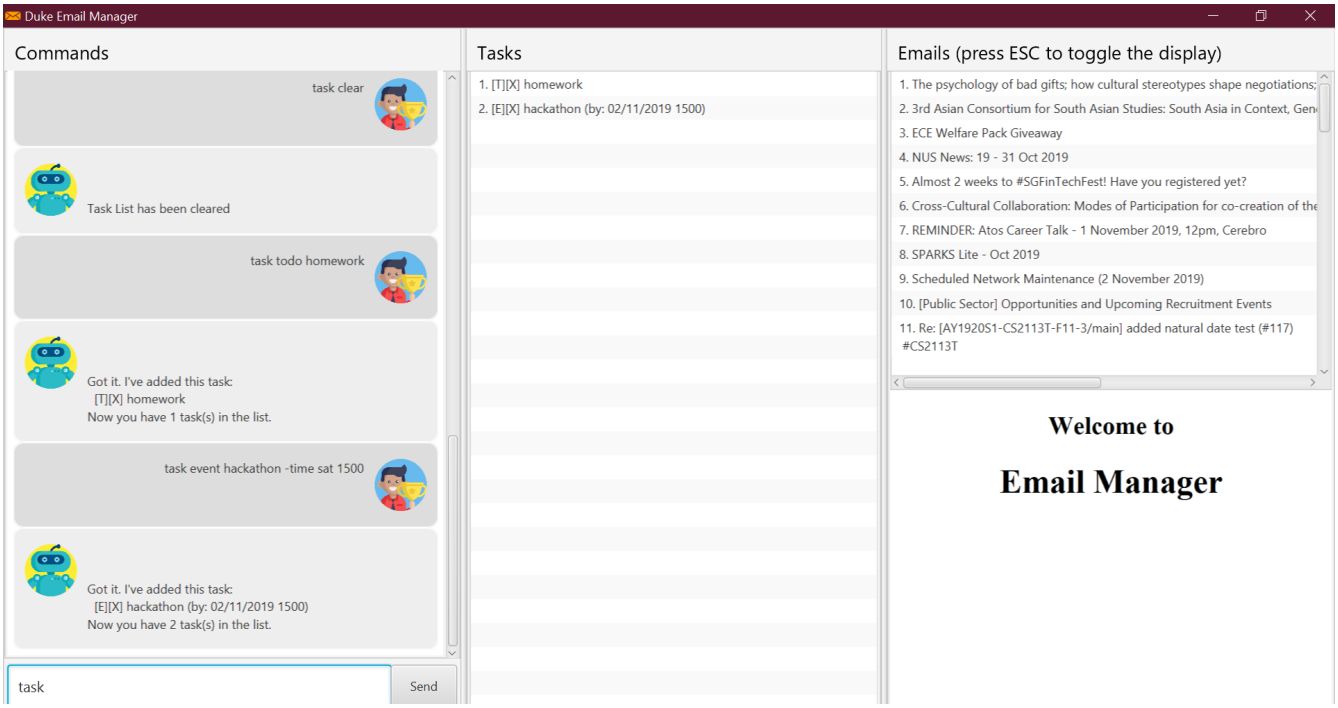
Vivian Lee - Project Portfolio

PROJECT: Duke Email Manager

About the Project

My team consists of 4 software engineering students, who were tasked with enhancing a basic command line interface of a personal task manager (Duke) for our Software Engineering Project. We 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. The various features of Duke Email Manager includes a task manager and email manager that are switchable between each other. Duke Email Manager uses Graphical User Interface (GUI) for students to use its functionalities by typing in commands.

This is what our project looks like:



My role was to write codes to enhance the task manager in our project. These includes codes for more natural date format feature and sorting feature. The following sections illustrate these implementations in more detail, as well as relevant documentation I have contributed to the user and developer guides in relation to these implementations.

Note the following symbols and formatting used in this document:

	This is a tip. Follow these tips to aid your development of Email Manager.
	This is a note. Read these for additional information.

□	This is a warning. Heed these warnings to avoid making mistakes that will hamper your development efforts.

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

Natural Date Format feature

- **What it does:** Students can define the date (where needed) in International Organization for Standardization (ISO) date format, or define the date 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.
    - For example, if the date today is Thursday, 31/10/2019, and the input date is "Wed", the date will be saved as 06/11/2019, which is the date of the coming Wednesday.

      This feature also include the time. If no time is inputted after a day of the week, the time will be indicated as `0000`.

    - For example, "Thu" will give a date and time of '31/10/2019 0000', while "Thu 1200" will give '31/10/2019 1200'.
- **Justification:** This feature significantly reduces the time and effort needed to input the date and time, especially if students are not exactly sure of the time. It gives more flexibility to students when inputting dates.
- **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:** Please click the links below to examine code contributed to the creation and maintenance of this feature:
    - Initial implementation of natural date format: #91
    - Renamed variable and fix bug which causes error in Todo task: #92
    - Fix code to allow update command to take in natural date format: #93
    - Refactor all methods related to natural date into the same class: #99
    - Refactor natural date class to allow easier testing and added test code for natural date format feature: #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:** Students might have many tasks in the task list and it is hard for them to look through the list to find the task that they are looking for. This feature 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:** Please click the links below to examine code contributed to the creation and maintenance of this feature:
    - Implementation of sort command: #192

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:** With so many tasks in hand, students need a way to indicate which tasks are of higher priority so that they can complete those tasks first before proceeding to the lower priorities 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:** Please click the links below to examine code contributed to the creation and maintenance of this feature:
    - Initial implementation of priority feature: #79
    - Added fixed input for priorities: #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.
- **Highlights:**
- **Code contributed:** Please click the links below to examine code contributed to the creation and maintenance of this feature:
    - Implementation of snooze feature: #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

Documentation:

- Updated the user 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 Natural Date Format, priority, snooze and clear 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. == Contributions to the Developer Guide

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 `priority`**: tasks with higher priority will be at the top of the task list. **Sorted by `status`**: tasks that are not completed will be displayed at the top of the task list. **Sorted by `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 sections 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. The benefits of having this Natural Dates support are:

- Reduce the time and effort needed to key in the date and time for deadline and event tasks.

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

Implementation

The following sequence diagram below illustrates how this feature works:

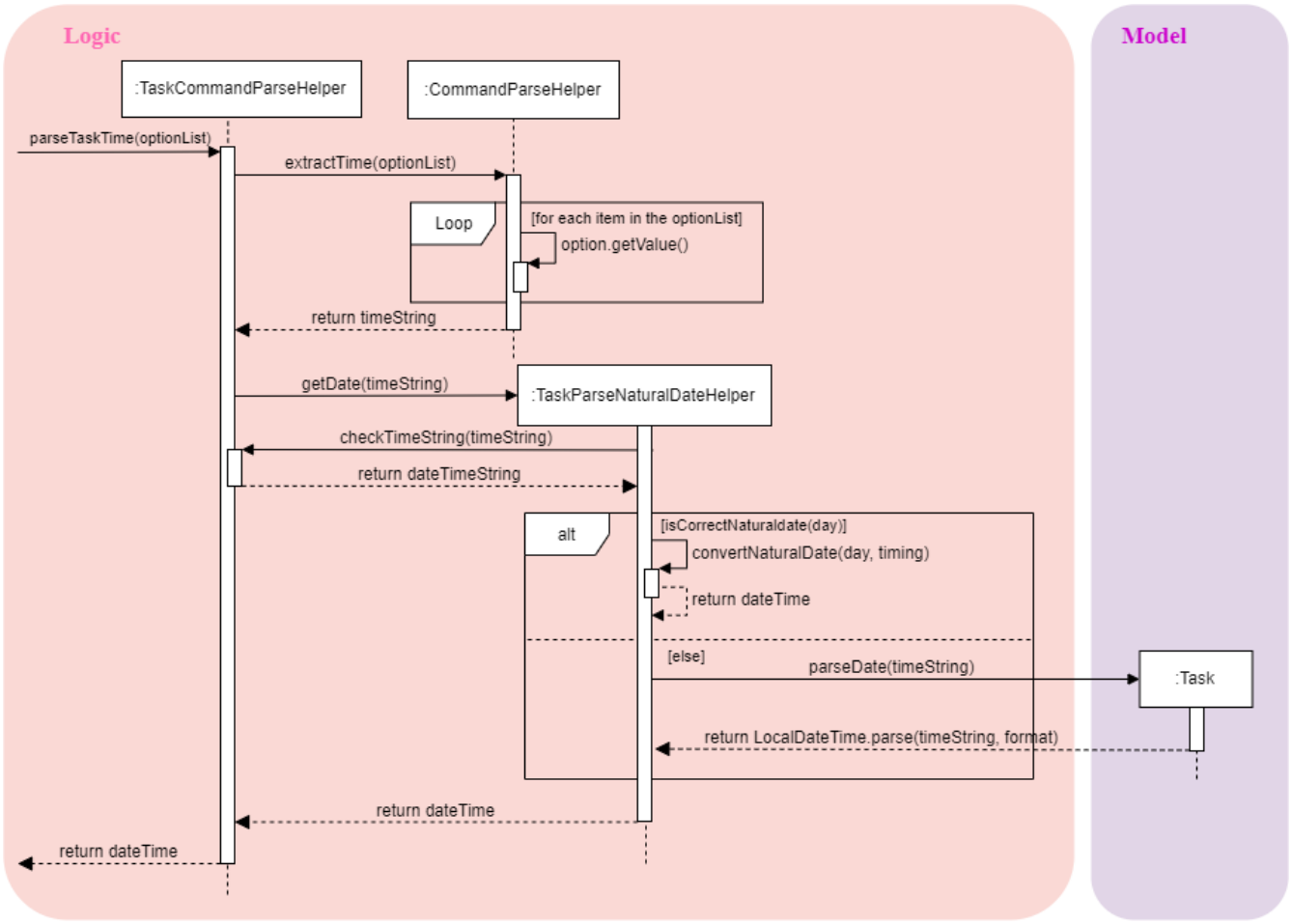


Figure 5: Natural Dates Support Sequence diagram

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

`TaskParseNaturalDateHelper` is under the Command component. It implements the following operations:

- `TaskParseNaturalDateHelper#isCorrectNaturalDate(day)` - Checks if `day` is a day of the week
- `TaskParseNaturalDateHelper#convertNaturalDate(day, time)` - Converts string day and time to local date and time in LocalTimeDate format

- `TaskParseNaturalDateHelper#getDate(timeString)` - Returns a dateTime in LocalDateTime format

`TaskCommandParseHelper` and `CommandParseHelper` are under the Parser component. It implements the following operations:

- `TaskCommandParseHelper#parseTaskTime(optionList)` - Parses time string extracted from optionList and returns a dateTime in LocalDateTime format
- `TaskCommandParseHelper#checkTimeString(timeString)` - Checks if time string contains time component and returns a pair with day as key and timing as value
- `CommandParseHelper#extractTime(optionList)` - Extracts and returns the time string from the input

`Task` is under the Entity component. It implements the following operations:

- `Task#parseDate(timeString)` - Converts timeString to LocalDateTime format if timeString is of `dd/MM/uuuu HHmm` format

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 `Task#parseDate(timeString)` is called.

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

- The next nearest date is returned according to the input day
  - E.g. When the timeString inputted is `sun 1200`, the date of the next nearest Sunday is returned.

Design Considerations

Aspect: Handling of parsed time string

- Alternative 1 (chosen): `TaskCommandParseHelper#checkTimeString(timeString)` parses and returns the time string as a string pair that stores day as key and time as value.
    - Pros: Easier to use the day and time in the pair in other methods without having to extract it from time string again.
    - Cons: Does not check whether day and time in the pair is valid, the pair can be any string. `TaskCommandParseHelper#checkTimeString(timeString)` is called from another class which increase dependency.
- Alternative 2: Parse the time string and extract day or time before each use in `TaskParseNaturalDateHelper`
    - Pros: Able to validate if time string is valid before it is called by other methods. Easier to write tests.
    - Cons: Might have duplicated codes.

Sorting of Task List

The task list can be sorted according to what the user prefer. The benefits of having this sorting feature are:

- Easier viewing of task list when the list is sorted.
- Faster checking of task list when the list is sorted.

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

Implementation

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

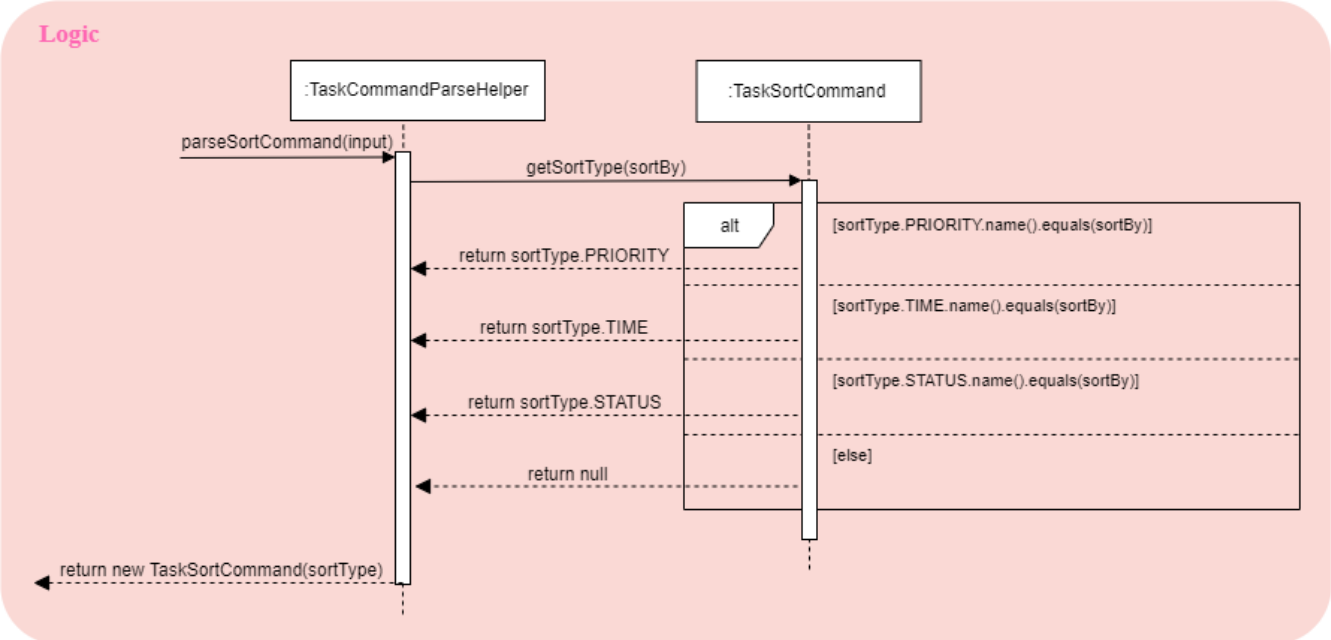


Figure 6: parseSortCommand Sequence diagram

As seen from the diagram above, the parsing of sort command is facilitated by two class, namely

`TaskCommandParseHelper` and `TaskSortCommand`.

`TaskCommandParseHelper` is under the Parser component. It implements the following operations:

- `TaskCommandParseHelper#parseSortCommand(input)` - Parse the input and extracts the sort type(sortBy) after the command `sort`
    - E.g. `sort priority` → sortBy = "priority"

`TaskSortCommand` is under the Command component. It implements the following operations:

- `TaskSortCommand#getSortType(sortBy)` - Checks sortBy and returns the sort type if sortBy is valid

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:

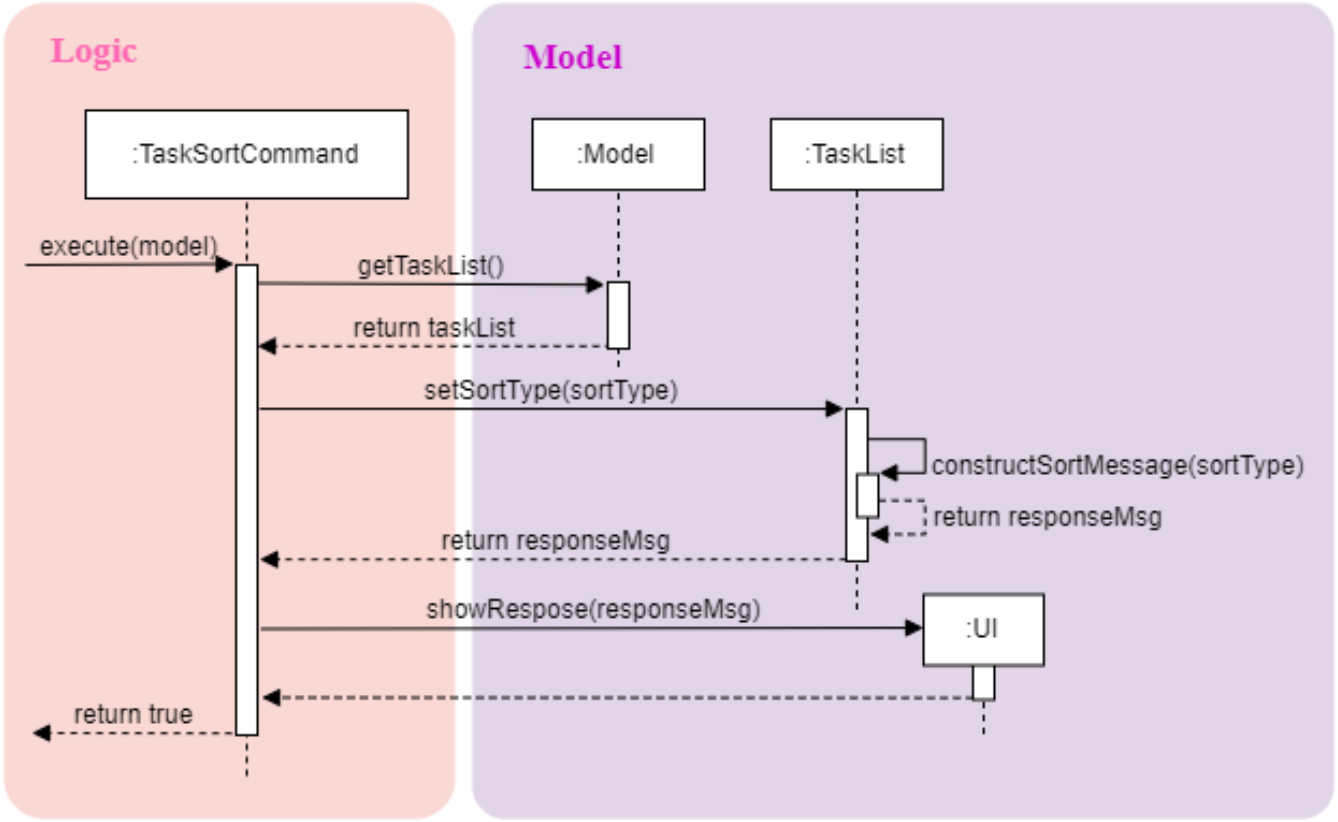


Figure 7: TaskSortCommand Sequence diagram

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

The following operations are implemented:

- `TaskSortCommand#execute(model)` - executes the sort command
- `Model#getTaskList()` - returns current task list
- `TaskList#setSortType(sortType)` - sets the sort type of the task list to `sortType`
- `TaskList#constructSortMessage(sortType)` - returns responseMsg
- `UI#showResponse(responseMSg)` - display responseMsg

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:

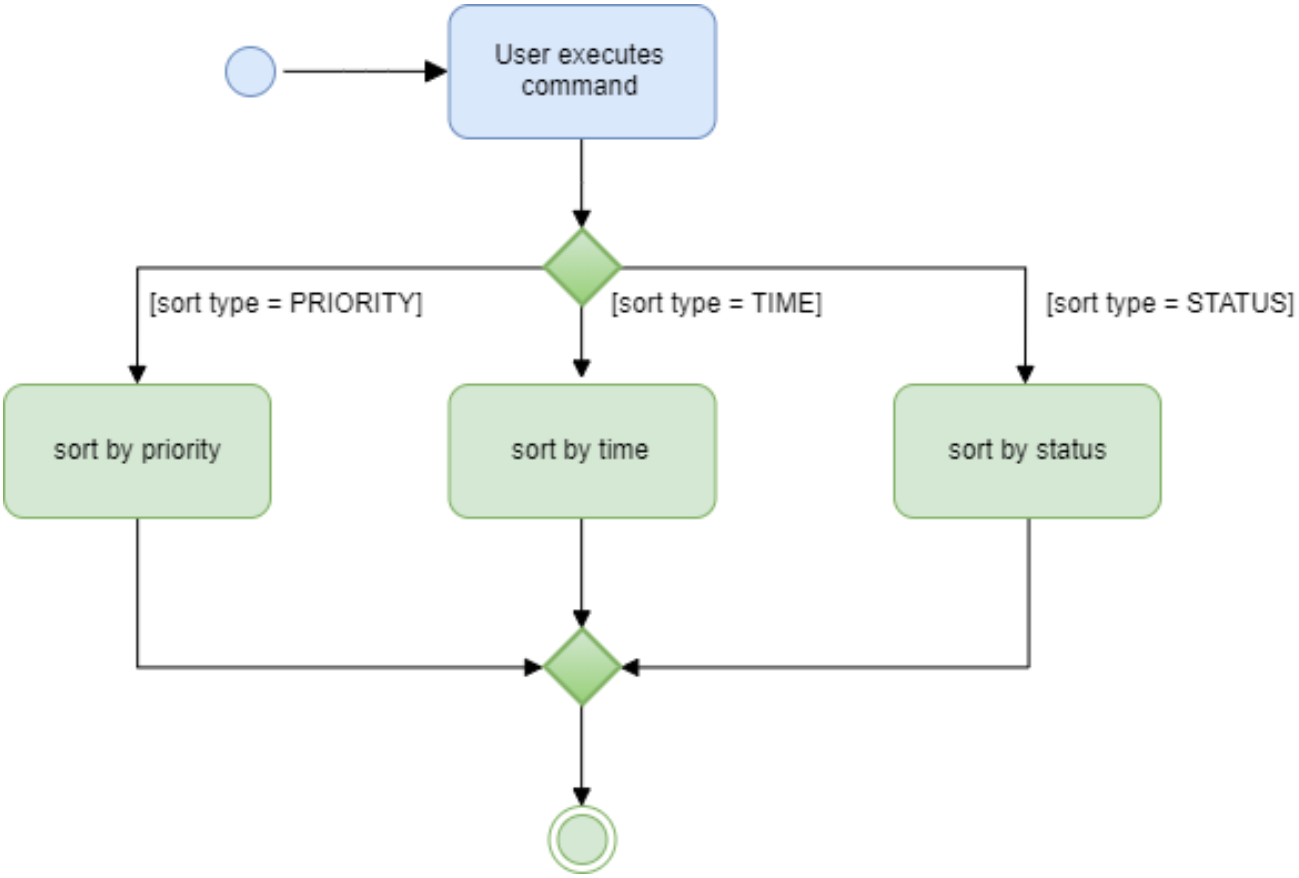


Figure 8: sortByType activity diagram

□□	**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.

□□	
sorted by `priority`	tasks with higher priority will be at the top of the task list.
sorted by `status`	tasks that are not completed will be at the top of the task list.
sorted by `time`	deadline or event tasks with nearing date and time will be at the top of the task list.

Design Considerations

Aspect: When the sorting of task list occurs

- Alternative 1 (chosen): The task list is sorted whenever the GUI is updated.
    - Pros: Ensures that the task list will always be sorted according to how the user wants.
    - Cons: Sort command is executed on every user input to keep task list sorted and task list view in GUI updated, which increases the computational load.
- Alternative 2: The task list is sorted in `TaskList` class before it is displayed by the GUI
    - Pros: Sort command is called only when needed.
    - Cons: Have to consider other commands (e.g. `update`) that will change the task list and requires the task list to be sorted.