Jel Lim – Project Portfolio

Overview

JelphaBot is a desktop Task Manager that aids NUS students in tracking tasks efficiently. The user interacts with it using a CLI, but it has a GUI created with JavaFX. It is written in Java, and has about 13 kLoC.

Summary of contributions

- Code contributed: [View on RepoSense]
- Major enhancement: Added the start and stop a timer feature for tasks as well as a viewing panel for the user's overall productivity for the day.
  - What it does: `start <index>` command allows the user to start a timer for the task specified by the index provided. The user may stop this timer with `stop <index>` command. The productivity panel will show the total time spent on as well as the number of tasks completed that are due today.
  - Justification: If the user wants to gauge the amount of time spent on their tasks and how many tasks they have completed or an overview of their productivity in that day, it is easy to view all this information in one tab.
  - Highlights: This enhancement works with existing as well as future commands. An in-depth analysis of design alternatives was necessary to decide the layout of the productivity panel and how effective it was in improving overall user experience. The implementation was also challenging as it was difficult to decide how much data to present to users.
  - Credits: Stop and Start command classes were adapted from existing commands such as Edit command.
- Minor enhancement: Added a command history that allows the user to navigate to previous commands using up/down keys.

Other contributions:

- Issues management:
  - Out of a total of 95 issues from versions 1.0 to 1.4. I managed 42 of them on GitHub (examples: #163 #263).
- User stories management:
  - There was a total of 30 user stories, for versions 1.0 to 1.4. I managed 10 of them.
- Documentation:
  - Developer Guide:
    - Section 3.1, Delete Sequence diagram, whole section of 3.3, 4.1, 4.4, 4.7
    - Appendix C, Use Case diagram, whole section of appendix C8, C9, D, E, F7, F8, F9, F10

  - Language improvements to User Guide : PR #283, #300, #316
  - Replaced existing UML diagrams to ones that corresponds with our final product: PR #198, #326
  - Added Use Case diagram and Activity diagrams for start command: PR #326
  - Added Sequence diagram for stop command: PR #328
- Enhancement to existing features:
  - Added DateTime model to Task model to integrate deadlines.
  - Added
- Community:
  - Reviewed Pull Request: #19
  - Reported 13 bugs and offered suggestions during PE dry run: issues
  - Reviewed other team's DG: (examples: 1, 2, 3)
- Tools:
  - Integrated CI/CD tools (Appveyor and Coveralls) to the team repository.

Contributions to User Guide (Extracts)

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

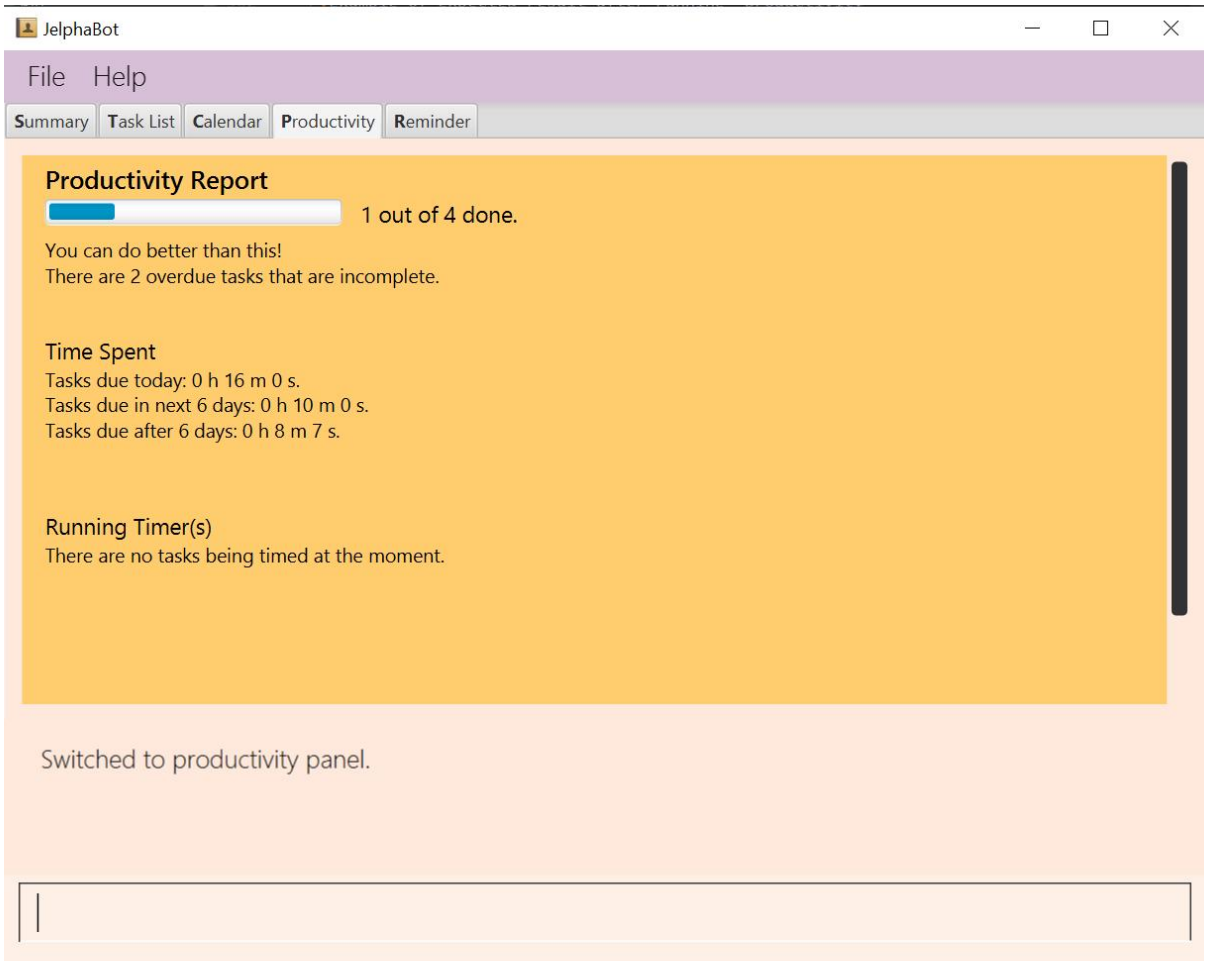
Productivity Tracking (Jel)

JelphaBot also comes with a productivity tracking that allows you track the progress of your tasks in that week. You would be able to see the progress bar fill up as you complete more tasks!

Track productivity : `productivity`

Apart from the function to switch tabs by pressing kbd:[Ctrl] + kbd:[tab] on your keyboard, you can enter the `productivity` command or its shortcuts `:P` or `:p` to manually switch to the productivity tab. The productivity panel will then show you your productivity for the day.

Format: `productivity`
Shortcut: `:P` or `:p`



Productivity Report
1 out of 4 done.
You can do better than this!
There are 2 overdue tasks that are incomplete.

Time Spent
Tasks due today: 0 h 16 m 0 s.
Tasks due in next 6 days: 0 h 10 m 0 s.
Tasks due after 6 days: 0 h 8 m 7 s.

Running Timer(s)
There are no tasks being timed at the moment.

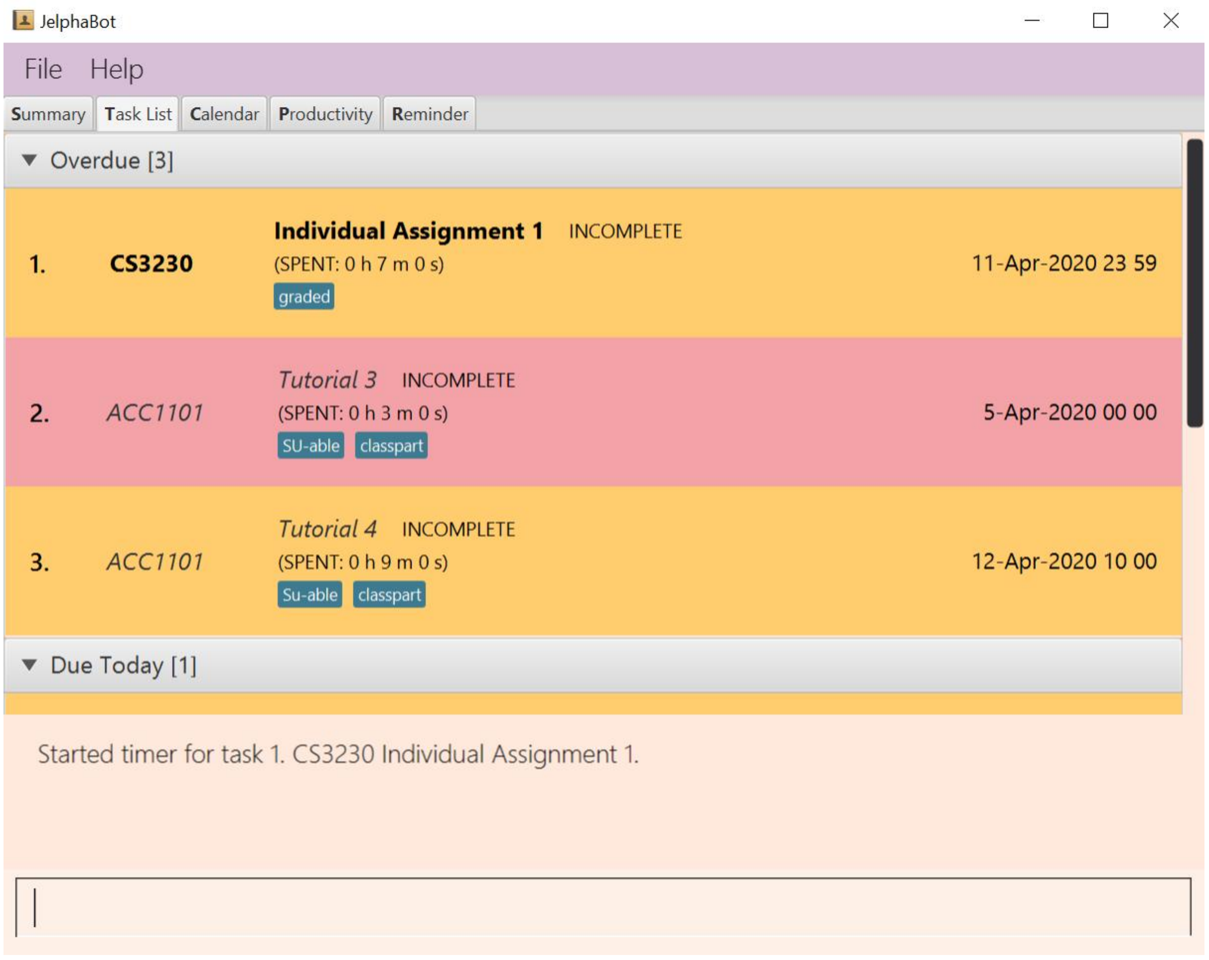
Switched to productivity panel.

Figure 1. Example of expected result after running `productivity`

NOTE	The progress bar and the text following it only shows tasks that are due on the day JelphaBot is running.

Starting timer for a task : `start`

You can start a timer for your task.
Format: `start INDEX`


JelphaBot

File Help

Summary | Task List | Calendar | Productivity | Reminder

Overdue [3]

1. CS3230 **Individual Assignment 1** INCOMPLETE (SPENT: 0 h 7 m 0 s) graded 11-Apr-2020 23 59

2. ACC1101 *Tutorial 3* INCOMPLETE (SPENT: 0 h 3 m 0 s) SU-able classpart 5-Apr-2020 00 00

3. ACC1101 *Tutorial 4* INCOMPLETE (SPENT: 0 h 9 m 0 s) Su-able classpart 12-Apr-2020 10 00

Due Today [1]

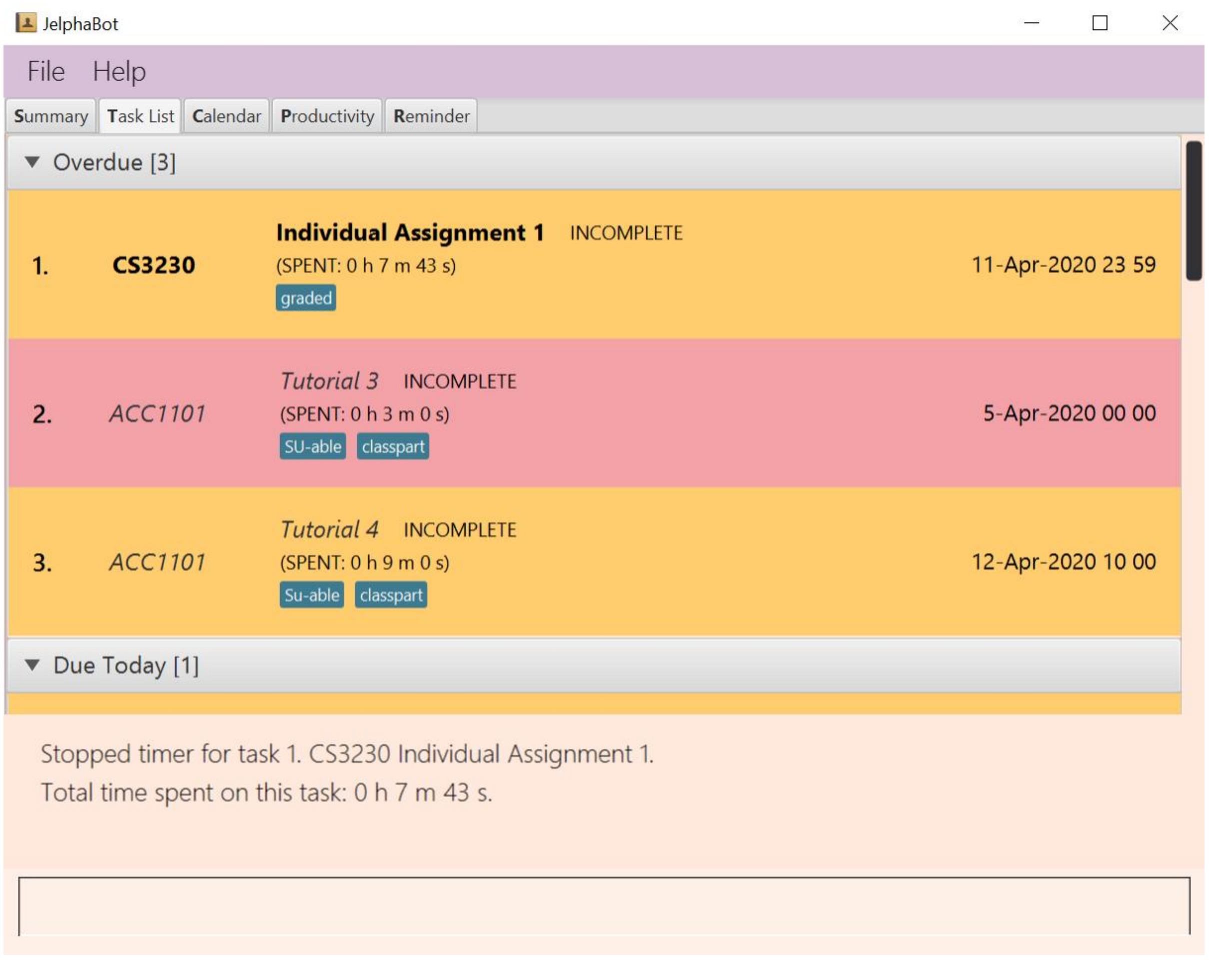
Started timer for task 1. CS3230 Individual Assignment 1.

Figure 2. Example of expected result after running `start 1`

- Each task can only have 1 running timer.
- Starts the timer for the task at the specified `INDEX` if timer was not already running.
- The index refers to the index number shown in the displayed task list.
- The index **must be a positive integer** 1, 2, 3, …
- A completed task cannot be timed.
- New timer entry under "Running Timer(s)" in the productivity tab will be added if execution is successful.

Stopping timer for a task : `stop`

You can stop a running timer for your task.
Format: `stop INDEX`



1. CS3230 Individual Assignment 1 INCOMPLETE (SPENT: 0 h 7 m 43 s) graded 11-Apr-2020 23 59
2. ACC1101 Tutorial 3 INCOMPLETE (SPENT: 0 h 3 m 0 s) SU-able classpart 5-Apr-2020 00 00
3. ACC1101 Tutorial 4 INCOMPLETE (SPENT: 0 h 9 m 0 s) Su-able classpart 12-Apr-2020 10 00

```
Stopped timer for task 1. CS3230 Individual Assignment 1.
Total time spent on this task: 0 h 7 m 43 s.
```

Figure 3. Expected result after running `stop 1`

- The task has to have a running timer.
- Stops the timer for the task at the specified `INDEX` if timer was running.
- The index refers to the index number shown in the displayed task list.
- The index **must be a positive integer** 1, 2, 3, …
- Timer entry under "Running Timer(s)" in the productivity tab will be removed if execution is successful.
- Time spent on timed task will be added to the respected time spent section in the productivity tab.

Receiving encouragement and criticism

JelphaBot automatically tracks the user’s productivity in a day and outputs the appropriate response to the user’s achievements and task completion rate.
There is no need to manually request for compliments or criticism.

Contributions to Developer Guide (Extracts)

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

Productivity feature (Jel)

JelphaBot has a productivity panel of this feature which provides an overarching view of user's overall productivity.

The view of this panel is facilitated by the productivity package that extracts the relevant data and displays them as a cohesive view. The productivity package supports the creation of `TimeSpentToday`, `RunningTimers` and `TasksCompleted` instances. Each of these classes iterate through the tasks contained within the task list.
Below is a class diagram of the related productivity classes:

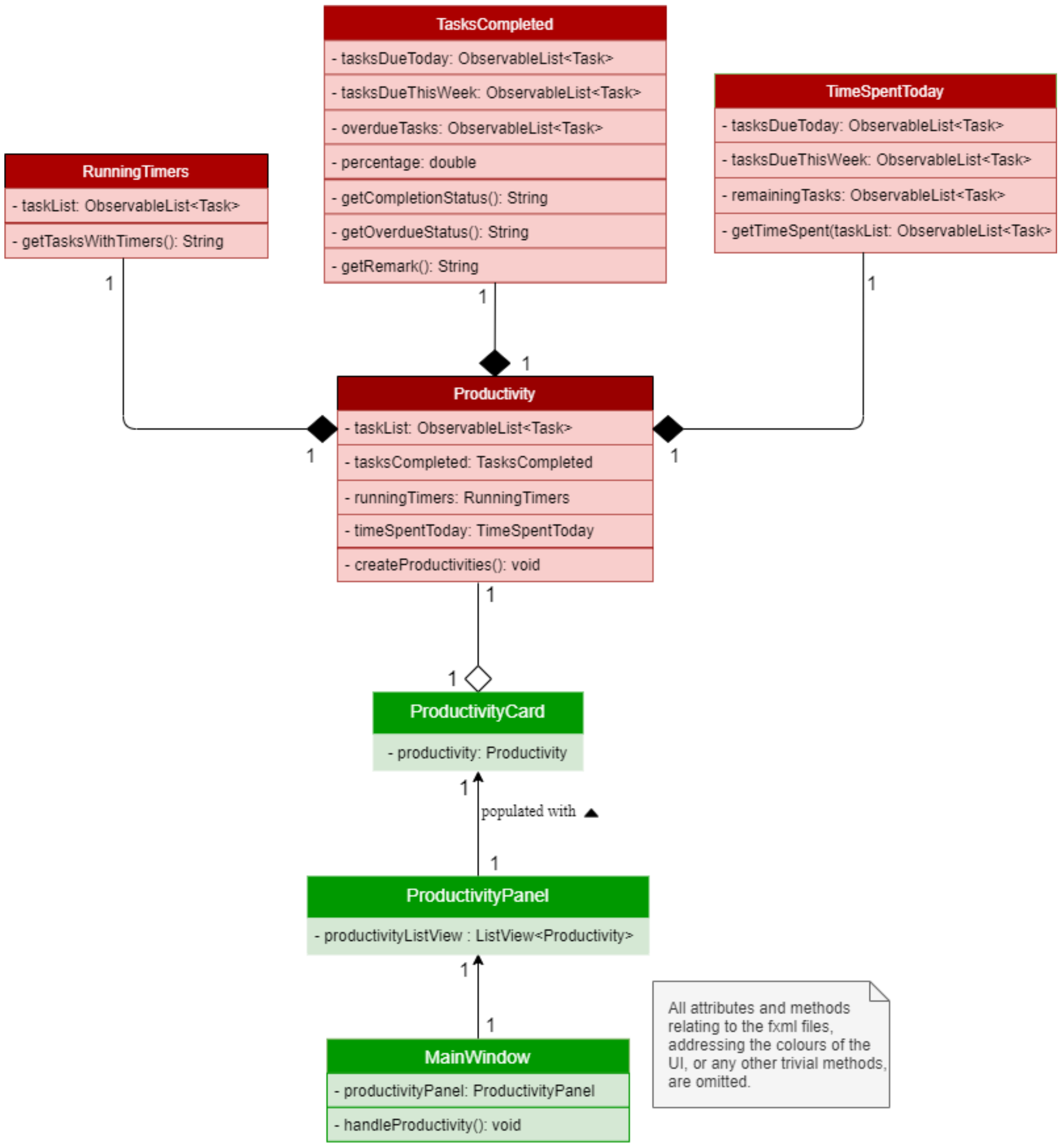

TasksCompleted
- tasksDueToday: ObservableList<Task>
- tasksDueThisWeek: ObservableList<Task>
- overdueTasks: ObservableList<Task>
- percentage: double
- getCompletionStatus(): String
- getOverdueStatus(): String
- getRemark(): String
TimeSpentToday
- tasksDueToday: ObservableList<Task>
- tasksDueThisWeek: ObservableList<Task>
- remainingTasks: ObservableList<Task>
- getTimeSpent(taskList: ObservableList<Task>
RunningTimers
- taskList: ObservableList<Task>
- getTasksWithTimers(): String
Productivity
- taskList: ObservableList<Task>
- tasksCompleted: TasksCompleted
- runningTimers: RunningTimers
- timeSpentToday: TimeSpentToday
- createProductivities(): void
ProductivityCard
- productivity: Productivity
populated with
ProductivityPanel
- productivityListView : ListView<Productivity>
MainWindow
- productivityPanel: ProductivityPanel
- handleProductivity(): void
All attributes and methods relating to the fxml files, addressing the colours of the UI, or any other trivial methods, are omitted.

Figure 4. Class diagram showing the structure and relations of `Productivity`, `ProductivityPanel` *and their components.*

This feature offers two main functions and one panel for visualisation:

- Start timer for a task.
- Stop running timer for a task.
- Productivity panel under Productivity tab.

Implementation

Text rendered onto the productivity panel is retrieved from the `Productivity` class.
A `Productivity` object is a container for the objects responsible for the sub-parts of the panel, namely `TimeSpentToday`, `RunningTimers` and `TasksCompleted`. Each of these have their respective String representations which will be used in rendering the panel.

`TimeSpentToday` implements the following operations:

- An iterator method `getTimeSpent()` which iterates through a list of `ObservableList<Task>`.
    - This iterator will extract the duration field of each task.
- A `toString()` method which returns the sum of duration (i.e. time spent) of tasks under 3 different categories: "due today", "due in next 6 days" and "due after 6 days".

NOTE	Each time `toString()` is called, `getTimeSpent()` is called thrice; once each for the 3 categories.

`RunningTimers` implements the following operations:

- An iterator method `getTasksWithTimers()` which iterates through a list of `ObservableList<Task>`.
    - This iterator will extract the description and deadline of tasks with timers that have been started.
- A `toString()` method which returns the tasks with running timers.

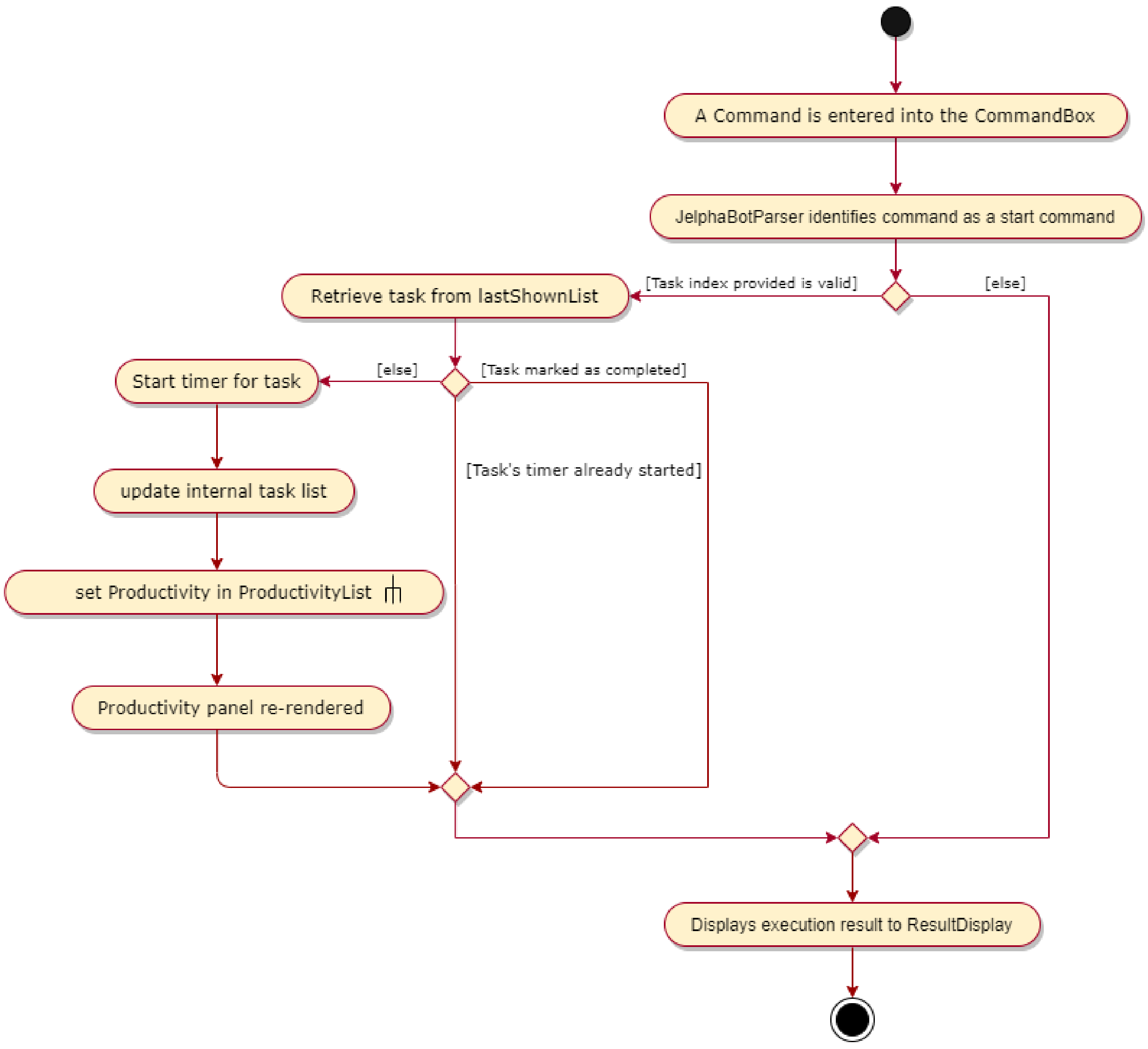
`TasksCompleted` implements the following operations:

- An iterator method `getCompletionStatus()` which iterates through a list of `ObservableList<Task>`.
    - This iterator will extract the number of tasks completed under the "due today" category.
- An iterator method `getOverdueStatus()` which iterates through a list of `ObservableList<Task>`.
    - This iterator will extract the number of tasks that are incomplete and past their due date.
- A getter method which returns the percentage of tasks completed that are under the "due today" category.
- A `toStringArray()` method which returns the task completion status, JelphaBot’s response to the user’s productivity, as well as number of overdue tasks.

Information from all three objects are subsequently rendered onto the panel through `ProductivityCard` and `ProductivityPanel`. Assuming that the task list is not empty, the following describe the flow of *start 1* and *stop 1* which modify the currently shown productivity panel:

Function 1: Starts timer for a specified task
In order to start timing a task, the user enters ***start INDEX*** command (e.g. *start 1*).

Upon successful execution of the command, the productivity tab displays the task being timed under the Running Timer(s) header. The following diagram shows the flow of *start 1* which modifies the current view of the productivity panel:

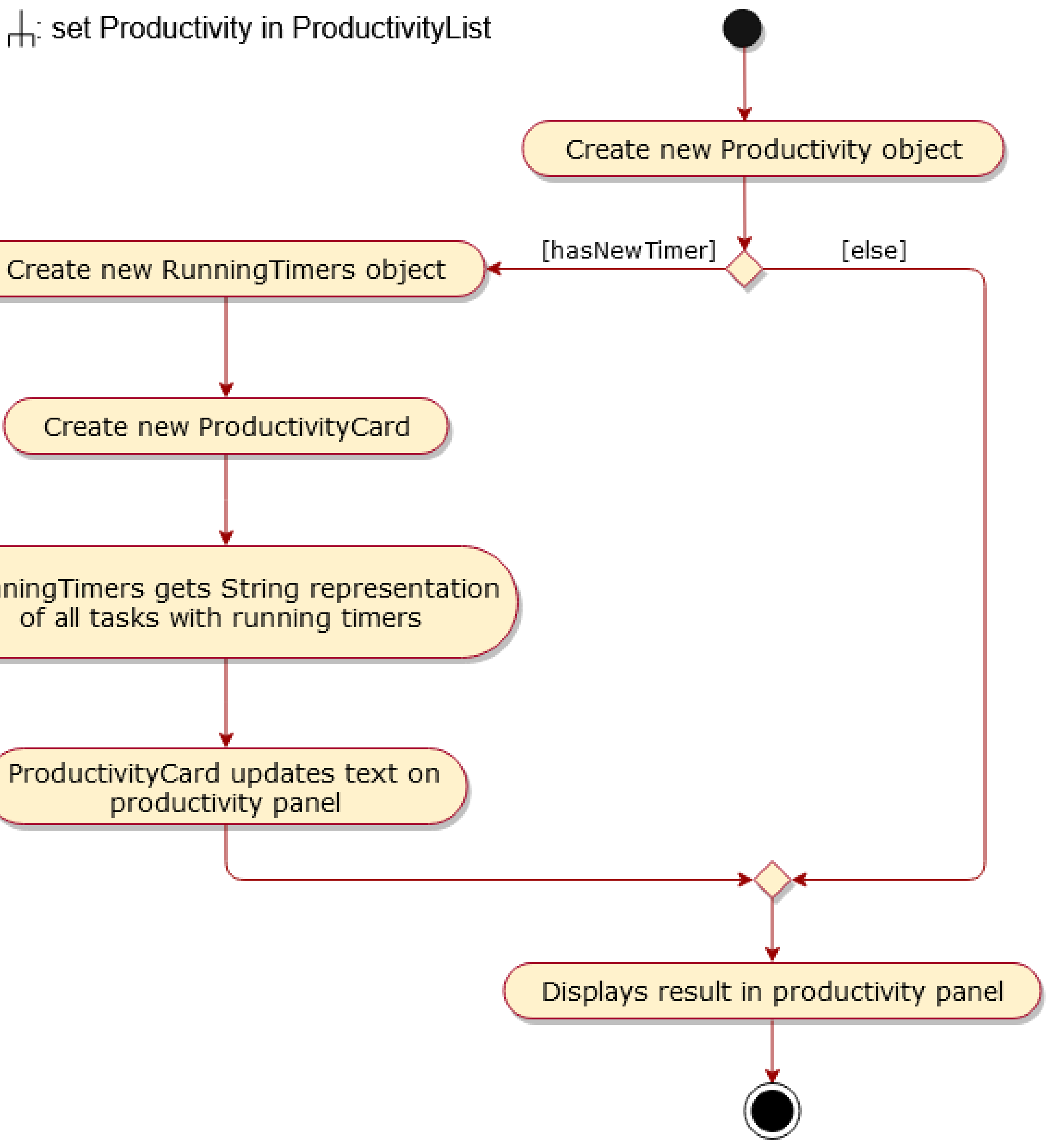


A Command is entered into the CommandBox
JelphaBotParser identifies command as a start command
[Task index provided is valid]
[else]
Retrieve task from lastShownList
[else]
[Task marked as completed]
Start timer for task
[Task's timer already started]
update internal task list
set Productivity in ProductivityList
Productivity panel re-rendered
Displays execution result to ResultDisplay

Figure 5. Activity Diagram showing the setting of `Productivity` *in the* `ProductivityList`

To update the productivity panel to reflect the changes, a new `Productivity` object will first be created, replacing the existing `Productivity` object in the `ProductivityList`. Each time a new `Productivity` object is created, its corresponding booleans will dictate whether the sub-parts (i.e. `TimeSpentToday`, `RunningTimers` and `TasksCompleted`) are to be replaced with new objects. As the command executed is ***start***, a new `RunningTimers` object is created. As detailed above, the iterator method in `RunningTimers` will be called and a new `String` representation to be displayed onto the productivity panel will be created. This `String` is subsequently rendered onto the panel under the Running Timer(s) header.

The following diagram shows the flow which updates the Running Timer(s) section in the productivity panel:



set Productivity in ProductivityList
Create new Productivity object
[hasNewTimer]
[else]
Create new RunningTimers object
Create new ProductivityCard
RunningTimers gets String representation of all tasks with running timers
ProductivityCard updates text on productivity panel
Displays result in productivity panel

Figure 6. Activity Diagram showing the updating of Running Timer(s) in the productivity panel

Function 2: Stops timer for a specified task

In order to stop timing a task, the user enters ***stop INDEX*** command (e.g. *stop 1*)

Upon successful execution of the command, the productivity tab removes the task being timed under the Running Timer(s) header. Removing a task from the Running Timer(s) header is similar to adding it, as illustrated by the Activity Diagram above. Under the Time Spent header, the total time spent will be increased in the respective subheaders depending on the date that the task is due.

IMPORTANT	Attempting to start a timer for a task which is marked as completed or stop a task with no active timer results in the command execution failing and an exception thrown.

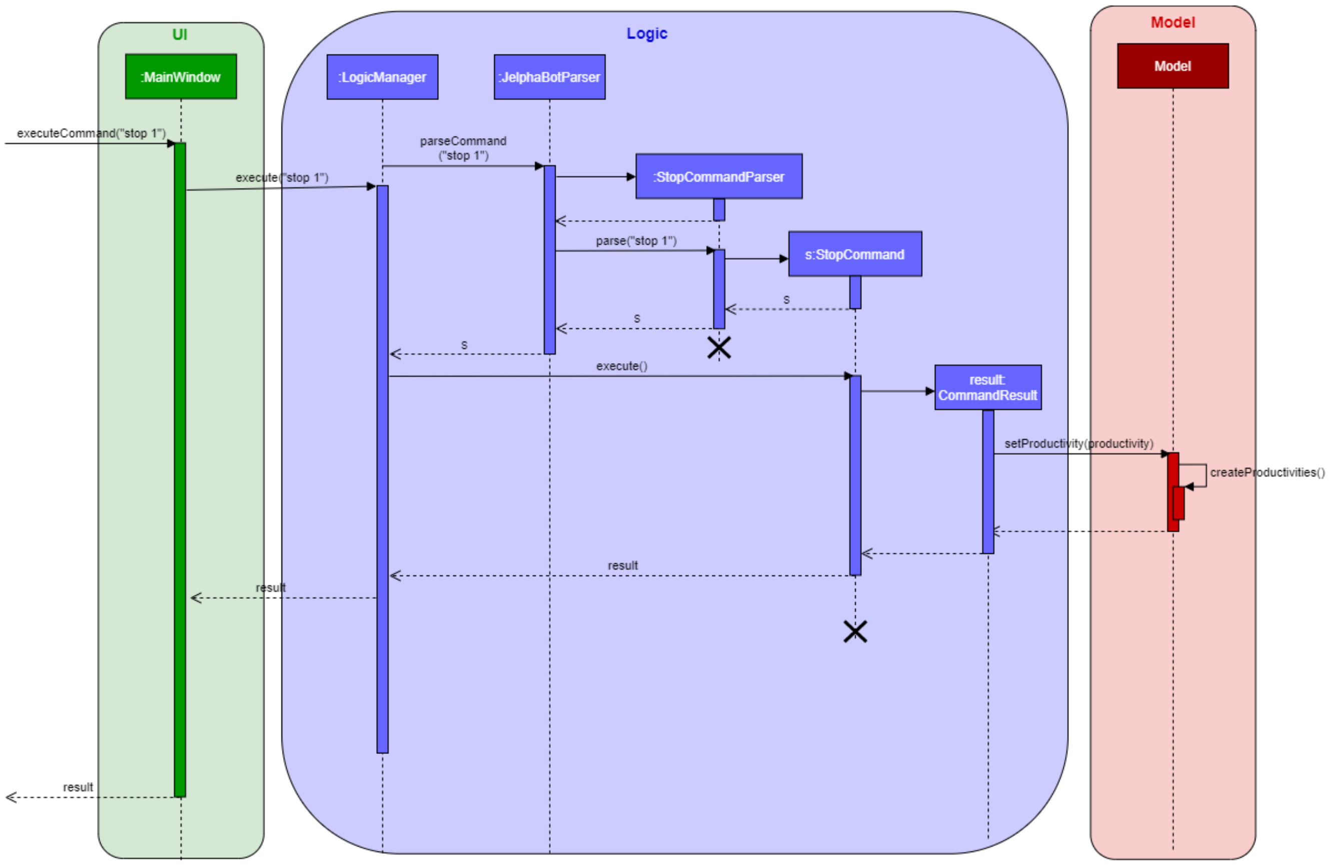


Figure 7. Sequence Diagram after running stop 1