Bryan Ng - Project Portfolio

Introduction

[github] [linkedin]

Hello! My name is Bryan. I'm currently a second year SoC (School of Computing) student in NUS majoring in Computer Science.

The purpose of this portfolio page is to document the contributions I have made in the development of TutorAid, which is a project that my team of 4 other Computer Science students and I completed for the module CS2103T. This project has definitely been greatly beneficial to my own learning and self-development in becoming a more competent software engineer.

Overview - TutorAid

TutorAid is a desktop application that was made for teaching assistants in NUS School of Computing. It is a Teaching Assistant (TA) management system that helps TAs improve their workflow and save time by better organizing all the information they need in one central location. TutorAid accomplishes this with its main features - tracking earnings, notes, tasks, reminders and students.

TutorAid is a Command Line Interface (CLI) based tool (i.e. you type in text input to execute commands) to cater to computing professionals who are highly adept at typing but also provides a Graphical User Interface (GUI) interface (i.e. graphical elements such as buttons are included) for users to easily view and interact with TutorAid.

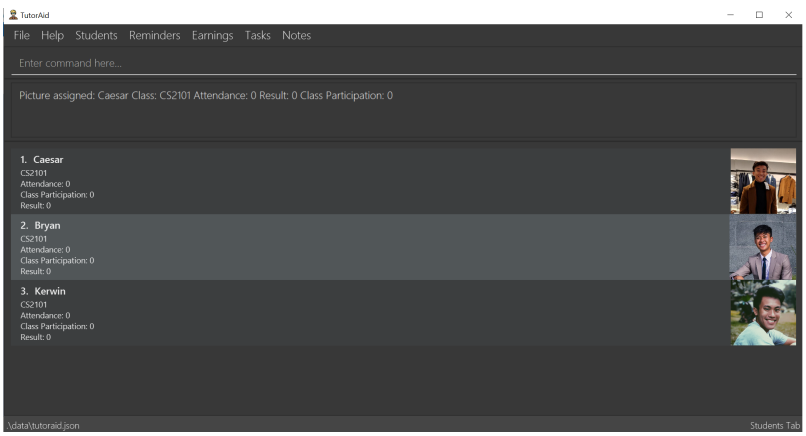


Figure 1. A view of TutorAid

Role

My main role was to prototype and develop the learning of wrong commands into custom commands feature and the Student Profile feature. The custom commands feature required handling command words that are not recognized by TutorAid carefully instead of just throwing an exception for unknown invalid commands. I created the base classes to facilitate this whole event flow, then linked it to the Model and Storage components since the custom commands have to be saved for future use. I also wrote rigorous tests for quality assurance of our product and to ensure that any possible bugs are kept to a minimal.

Summary of contributions

Given below is a brief write-up of my implemented features and other helpful contributions to the project.

- **Major enhancement**: Added **the learning of wrong commands into custom commands feature**
    - What it does: It gives the user the ability to learn commands which they have typed wrongly into TutorAid as a basic command. These custom commands will be saved and pop up as suggestions in the future.
    - Justification: This feature improves the product significantly because TutorAid has many features with 43 different commands currently and is still growing. It is extremely difficult for the user to remember all these command words without constantly referring to the user guide. They are highly likely to forget commands or key in invalid commands based on their own typing habits. Learning these typing habits and having them show up as suggested commands will help users improve their efficiency in using TutorAid as the issue of forgetting command words is minimized.
    - Implementation: This enhancement alters the flow of command execution in TutorAid. Executing an unknown command that TutorAid does not recognize changes the command execution pathway and sets TutorAid into "learner mode". Command parsing is now facilitated by the use of a TreeMap which stores a list of all available commands and their actions as key-value pairs. During execution, TutorAid will check against this TreeMap to see if the input command word exists. If it does not, this input command word is saved onto a Stack in the Model component to be retrieved in the next command. If the next command is still unknown, it is also added to the Stack. When a valid command that exists is entered, the last unknown command is retrieved from the top on the Stack and is mapped to the action of the aforementioned valid command. This key-value pair is then added to the TreeMap and saved into our .json file. TutorAid is then set back into normal mode. The TreeMap is also used to generate command suggestions based on the user’s input.

- **Minor enhancement**: Added **the Student Profile feature**
    - What it does: It allows the user to add students and their vital information to a list for easy viewing and management.
    - Justification: TAs have too many things to remember about individual students like their participation, attendance and results. Having an organized bird's eye view of everything in one place helps to reduce the time wasted in retrieving information from many different places.
    - Implementation: This feature was implemented by refactoring and enhancing the existing Person class with additional functionality such as setting a picture to a student profile.
- **Code contributed**: [All commits] [Project Code Dashboard]
- **Other contributions**:
    - Project management:
        - Managed bugs reported by other users in PED and assigned team members to issues: #287, #266, #290
        - Managed release `v1.3` on Github
    - Enhancements to existing features:
        - Wrote tests for existing features to increase code coverage from 36% to 42% (Pull request #316)
    - Documentation:
        - Added detailed implementation documentation for the custom shortcut feature in Developer Guide, including diagrams (Pull requests #215)
        - Customized and updated ReadMe for TutorAid
    - Community:
        - Reviewed and gave feedback to team members. PRs reviewed: #243, #214
    - Tools:
        - Set up Codacy for code quality

Contributions to the User Guide

Given below is an excerpt I wrote in our User Guide for the learning of wrong commands into custom commands feature.

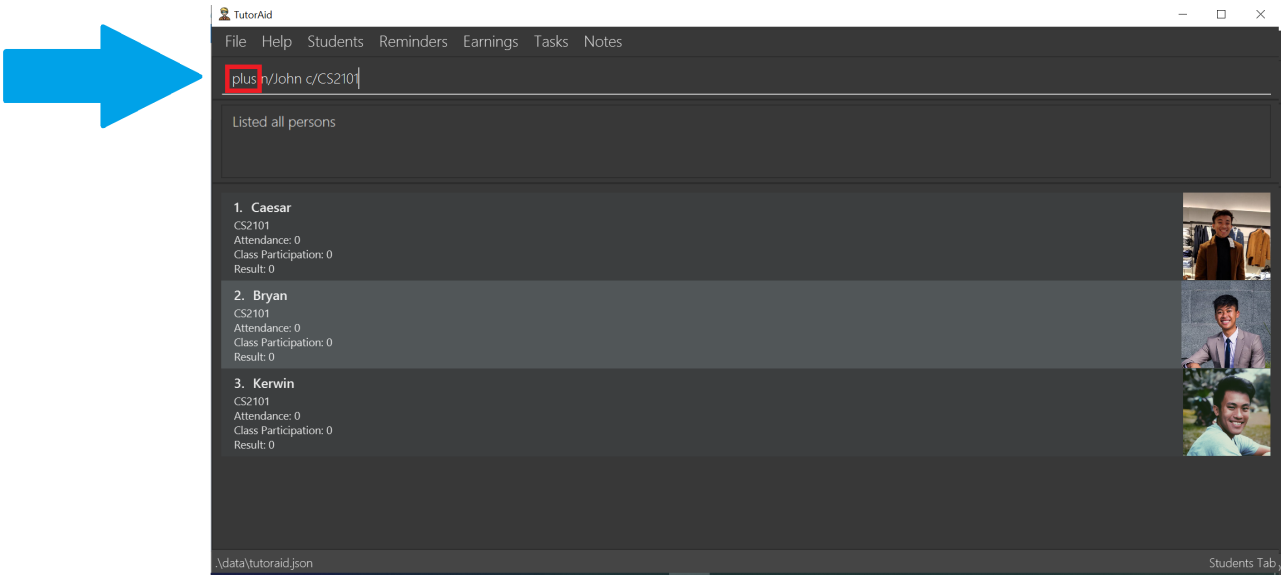
Learn wrong commands as custom commands

To help map the command you entered wrongly to the command you originally intended to use.

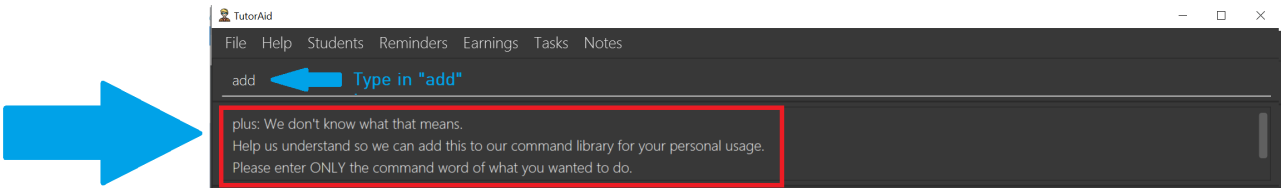
Example: Let's say you frequently use the `add` command but can never remember it and always type in `plus` instead. This feature helps you map `plus` to `add` so you no longer need to remember the `add` command.

To learn the wrong command `plus` as `add`:

1. You want to do an `add` command but carelessly type in `plus` instead and hit Enter to execute.

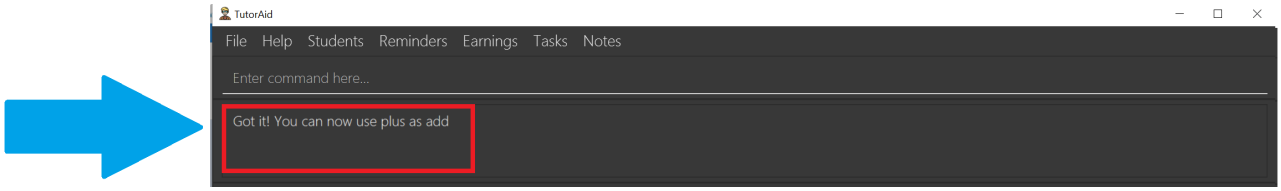


2. Oh no! TutorAid does not know what `plus` means! You realise you've entered an unknown command. Thankfully, TutorAid offers to help you learn `plus`. You should type in `add` now since it's what you actually intended to do. Hit Enter!

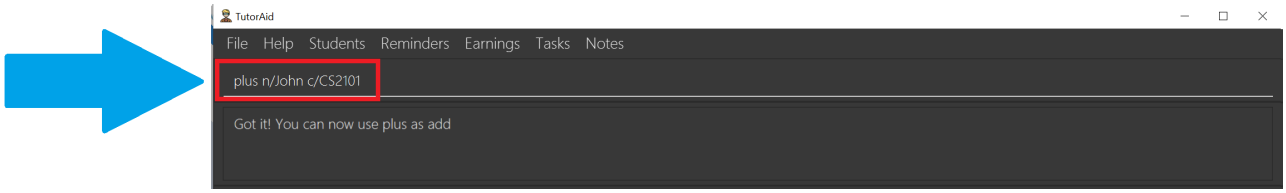


NOTE You can also choose to discard the wrong command at this stage and carry on with normal operations if you do not want to map `plus` to `add`. Just type `cancel`.

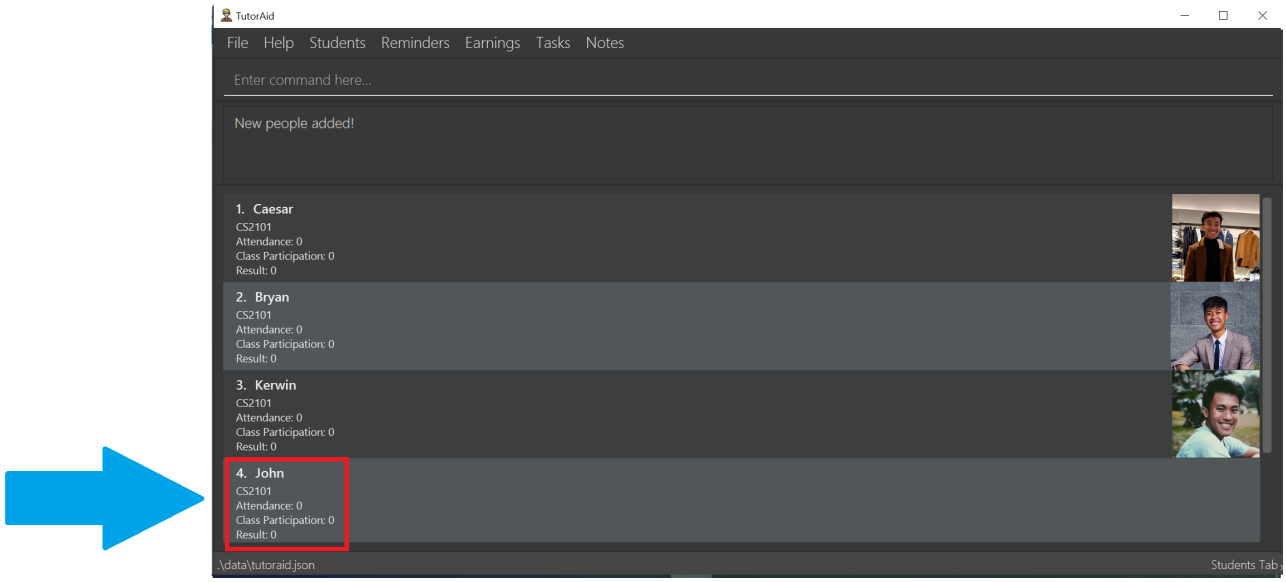
3. The result box tells you that `plus` has now been mapped to `add`.



4. Let's test our new command by trying to add a student named John in our CS2101 class. Type in `plus n/John c/CS2101` and hit Enter.



5. You should see that the command is successful and a new student called John in CS2101 has been added!

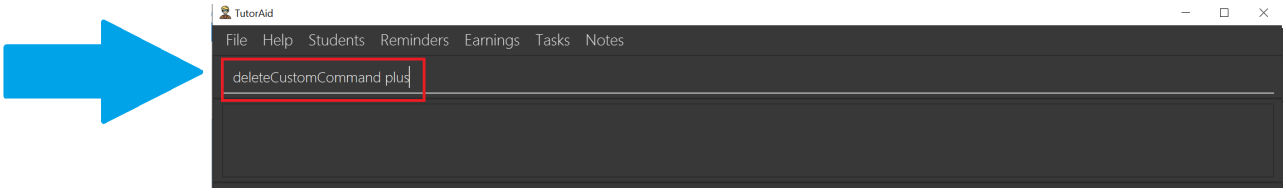


Delete a custom command: 'deleteCustomCommand'

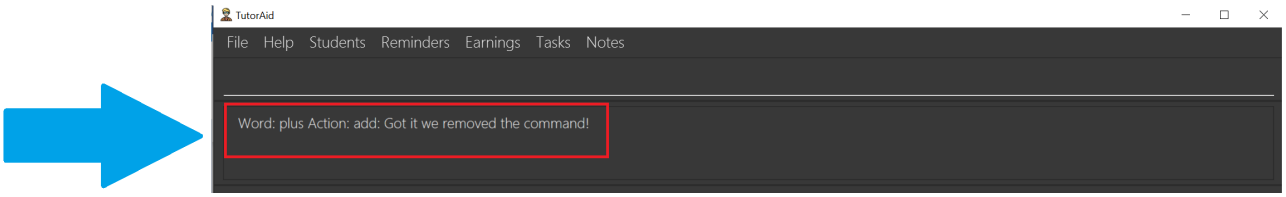
To delete a custom command you previously added. Format: `deleteCustomCommand CUSTOMCOMMAND`

Example: Let's try to delete the `plus` command you learned as `add` previously.

1. Type in `deleteCustomCommand plus` and hit Enter.



2. The result box informs you that they've deleted the custom command `plus`.



3. You should no longer be able to use `plus` as `add`.

NOTE	You can't use `deleteCustomCommand` to delete basic commands like `add`, `delete`, `list` etc. You can only delete custom commands you added.

Contributions to the Developer Guide

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.

Learn wrong commands as custom commands feature

The main point of this feature is to learn a user's typing habits so as to give them a more seamless experience while using TutorAid by adapting to their typing style so that they do not have to memorize built in commands. This is done by having TutorAid trigger in and out of "learner" mode.

Implementation

A `CommandResult` object now has an added boolean `isUnknown` instance variable to indicate if the command entered is an unknown command. Using this, TutorAid shifts between "learner" mode and normal mode:

Figure 2. How TutorAid goes from normal to "learner" mode

Figure 3. How TutorAid goes from "learner" to normal mode

The effect of being in "learner" mode results in the `TutorAidParser` using different methods to parse the command. This will be elaborated on in the sample use scenario below.

While in "learner" mode, TutorAid can map wrong commands to the actions of known commands. These wrong/custom command keywords and their mappings are stored locally in TutorAid.json with the help of `JsonAdaptedCommand` and the fact that commands are now modelled as a `CommandObject` that contains their `CommandWord` and `CommandAction`.

A TreeMap is now being used in the process of parsing commands. New command classes such as `UnknownCommand`, `NewCommand` and `CancelCommand` were also created.

Given below is an example usage scenario and how the learn custom command mechanism behaves at each step.

Step 1. The user launches the application for the first time. The `TutorAidParser` will be initialized and all basic commands and previous existing custom commands will be added to its TreeMap via `TutorAidParser#initialiseBasicCommands()`.

Step 2. The user enters `plus` instead of `add`. `TutorAidParser` does a lookup in its TreeMap in the `TutorAidParser#parseCommand(String userInput)` method and returns a new `UnknownCommand` since the TreeMap does not contain the keyword `plus`.

Step 3. The `UnknownCommand` is executed by `Logic` and `plus` is saved on the `savedCommand` Stack in the `Model` component. Subsequently, the `CommandResult` is passed to `MainWindow` to display the corresponding text on the GUI. This `CommandResult` triggers TutorAid into "learner" mode via the earlier explained implementation.

Step 4. TutorAid prompts the user to type in a valid command to map the unknown command to. The user types in another wrong command that is not recognised. This time, he enters `ad` instead of `add`. Since TutorAid is in "learner" mode, the method `TutorAidParser#checkCommand(String userInput, String prevUnknownCommand)` is now called instead of `TutorAidParser#parseCommand(String userInput)`. Note that `prevUnknownCommand` is the last saved command retrieved from the `savedCommand` Stack.

NOTE	Logic#execute(String userInput) has been changed to Logic#execute(String userInput, boolean isUnknown) to account for different execution pathways when in normal and "learner" mode.

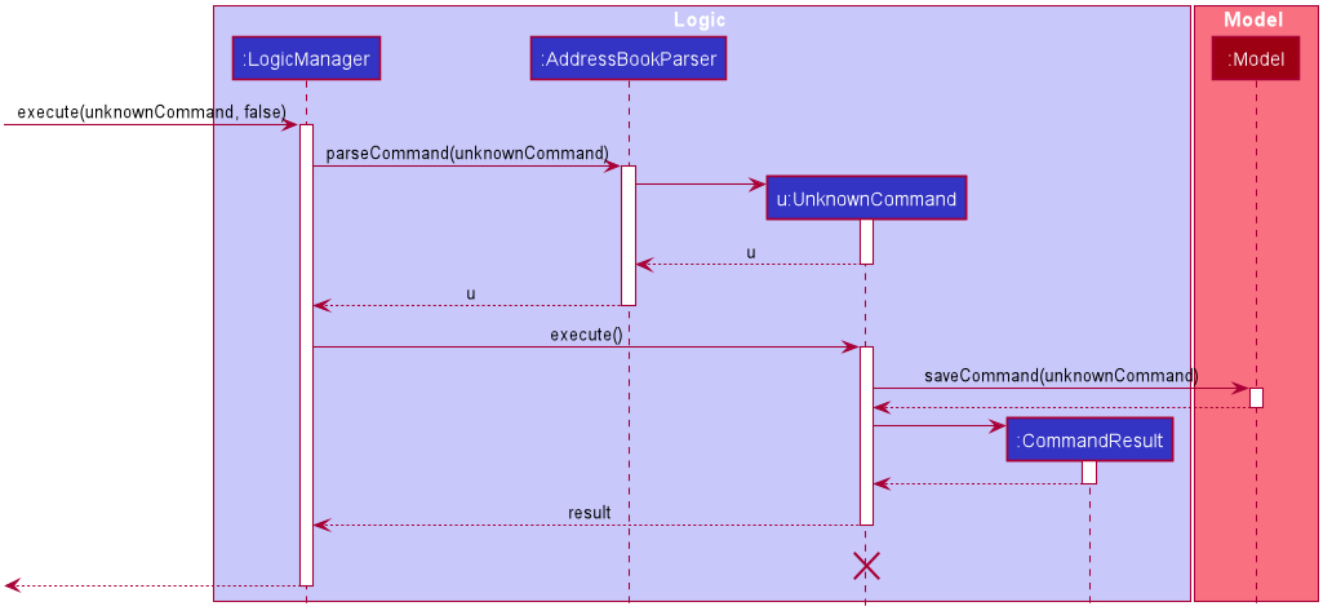
Step 5. In `TutorAidParser#checkCommand(String userInput, String prevUnknownCommand)`, the command is still not recognized and another `UnknownCommand` is returned and `ad` is also saved. Steps 3-5 occurs continuously as long as an unknown command is being supplied or until the user cancels the operation by typing in `cancel`.

Step 6. The user inputs a valid command. `TutorAidParser#checkCommand(String userInput, String prevUnknownCommand)` now finds the valid command in the TreeMap. With the latest `prevUnknownCommand` retrieved from the Stack, the unknown command and action of the valid command is added as a key-value pair to the TreeMap and a `NewCommand` object is returned and executed to store this new command mapping with the help of `Model`.

Step 7. `NewCommand` has a `CommandResult` with an `isUnknown` value of `false`. This will trigger TutorAid back to normal mode. Normal commands can then be performed as `TutorAidParser#parseCommand(String userInput)` will now be called again instead of `TutorAidParser#checkCommand(String userInput, String prevUnknownCommand)`.

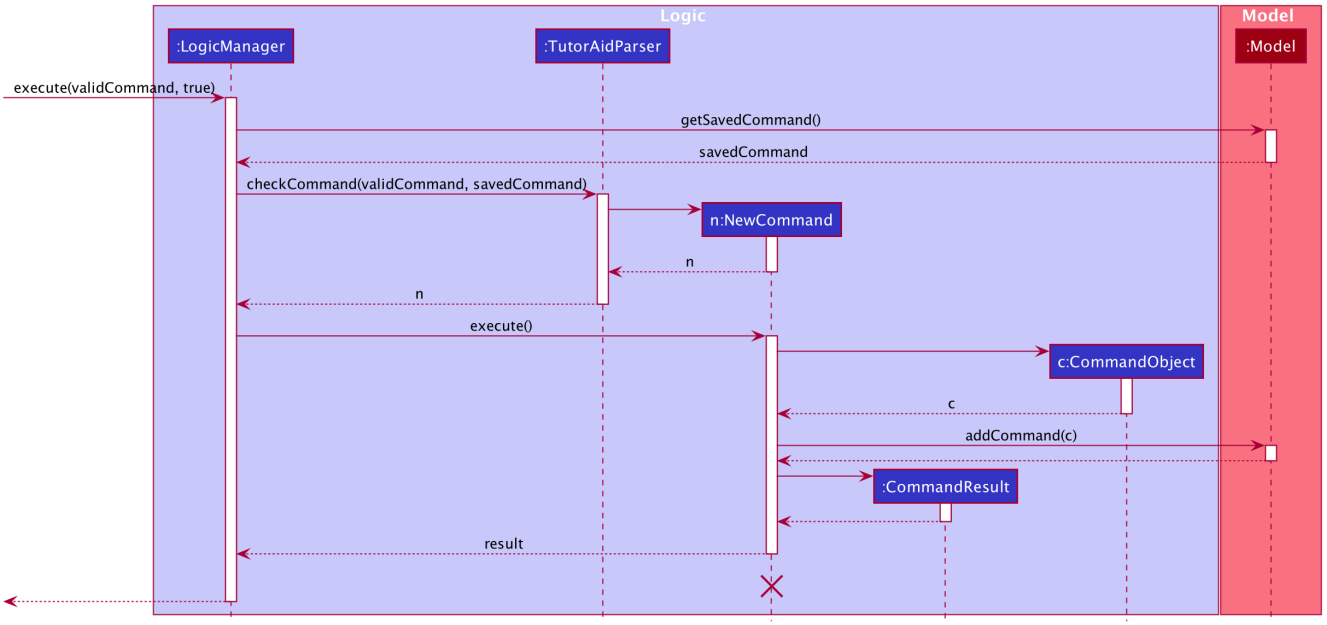
The following sequence diagrams shows how the learn custom command operation works:

Firstly an unknown command is supplied,

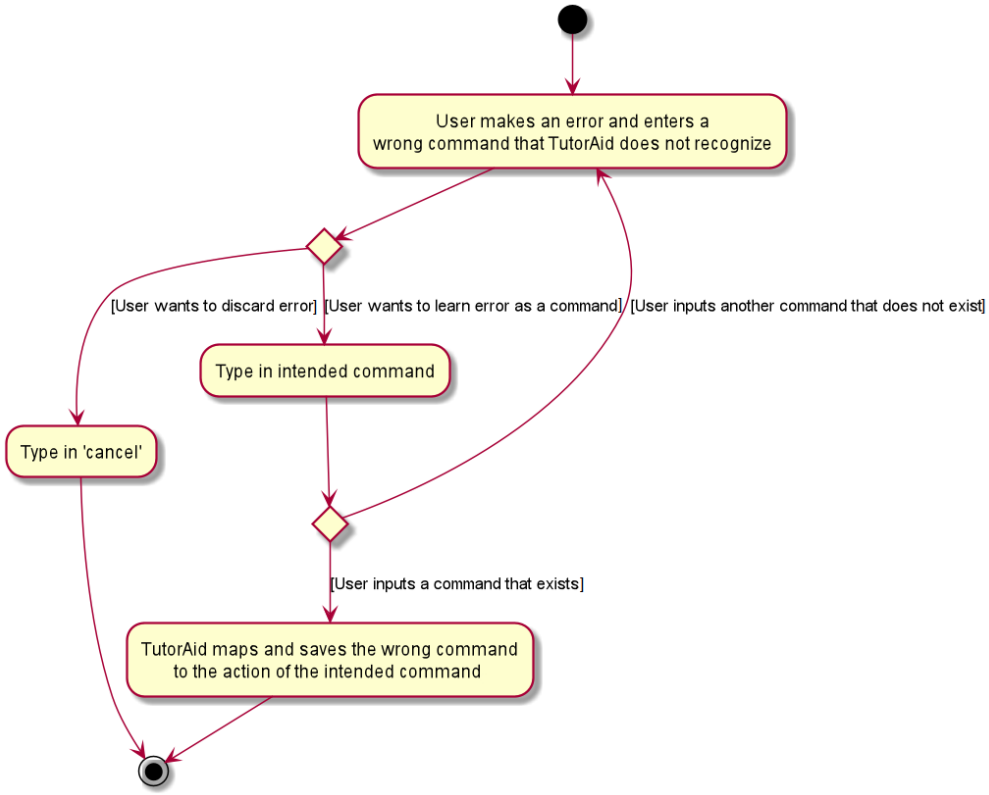


NOTE The lifeline for `UnknownCommand` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

After which, a known command is supplied,



The following activity diagram summarizes what happens when a user executes a new command:



Design Considerations

Aspect: Data structure to support the learning of custom commands

- **Alternative 1 (current choice):** Add all `CommandObject` objects in the `ObservableList<CommandObject>` into a TreeMap.
    - Pros: Future queries to determine if a command exists or not only requires O(1) time.
    - Cons: The first iteration to populate the TreeMap still takes O(n) time.
- **Alternative 2:** Iterate through the `ObservableList<CommandObject>` to check if the command exists.
    - Pros: We do not need to maintain a separate data structure, and just reuse what is already in the codebase. We also do not need to waste time populating a TreeMap.
    - Cons: Every single query costs O(n) time to check if the command exists.