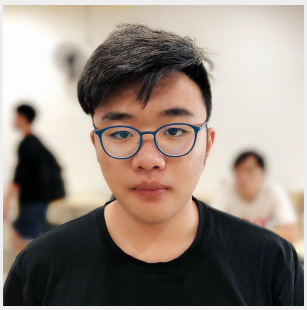
Xuerneas - Project Portfolio

PROJECT: TutorAid

Introduction



[*GitHub*]
Hi, I'm Xu Tunan, a Year Two SoC Student from National University of Singapore (NUS). This Project Portfolio is aimed to introduce our project — TutorAid, as well as state my own contributions in this project.

Overview of the TutorAid

TutorAid is a desktop application that designed to help the teaching assistants in NUS. It is a combination of the Calendar, the Earning Tracker, the Notepad, as well as the Student Profile Manager.
Those features of TutorAid was selected specifically for the target user - tutors, based on their exact needs.

The user interacts with TutorAid using a Command Line Interface (CLI), with a Graphical User Interface (GUI) created with JavaFX for user to better view the interactions.

Role

- My role in this project was to design and write the codes for the task-commands in Calendar feature as well as the undo and redo features. The sections below explain those enhancement specifically.

Summary of contributions

- ***Major enhancement***: added **the ability to undo/redo previous commands**
  - What it does: allows the user to undo all previous commands one at a time. Preceding undo commands can be reversed by using the redo command.
  - Justification: This feature improves the product significantly because a user can make mistakes in commands and the app should provide a convenient way to rectify them.

    - Highlights: This enhancement affects existing commands and commands to be added in future. It required an in-depth analysis of design alternatives. The implementation too was challenging as it required changes to existing commands.
    - Credits: The implementation of the undo and redo commands were inspired by Address Book 4, however more challenging in this project due to the complexity.
- ***Major enhancement***: added **the task management commands (Part of Calendar Feature)**
    - What it does: allows the user to `add`, `delete`, `edit`, `find`, `list` tasks.
    - Justification: This feature is one of the most important features in the TutorAid since it aimed to help tutors and this feature is really useful for tutors.
- ***Minor enhancement***: added a sorting method in TaskTime so that in each task, their task times would be sorted automatically.
    - Justification: This enhancement is added to show each task clearer. Also, users do not need to worry that they key in the task time in wrong order anymore.
- **Code contributed**: [**All Commits**][**Code Contribution**]
- **Other contributions**:
    - Project management:
        - Managed bugs reported by other users in Practical Exam Dry Run: #296, #300, #301
    - Enhancements to existing features:
        - Wrote additional tests for existing features to increase code coverage (Pull requests #232, #311, #320)
    - Documentation:
        - Added detailed implementation documentation for undo/redo feature and most part of the calendar feature in User Guide, including diagrams. (Pull requests #220, #341)
        - Added detailed implementation documentation with diagrams for undo/redo feature and add task command, class diagrams of Storage and Model, as well as user stories and use cases in Developer Guide. (Pull requests #118, #161, #220, #311)
    - Community:
        - Reviewed Pull Request with feedback : #164, #343
    - Tools:
        - Set up Coveralls for the test coverage of the project.

Contributions to the User Guide

Given below are some sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users. Because of the limitation of number of pages, some contributions like the detailed implementation of task commands and undo/redo commands are not shown below.

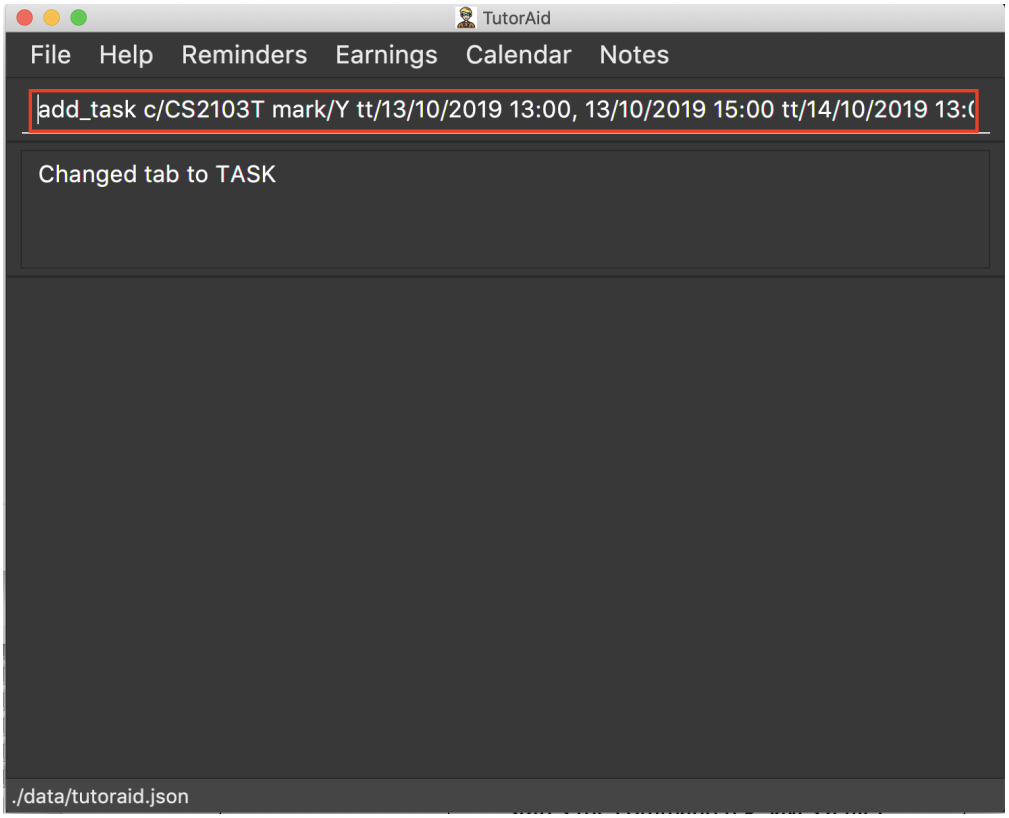
Adding task: `add_task`

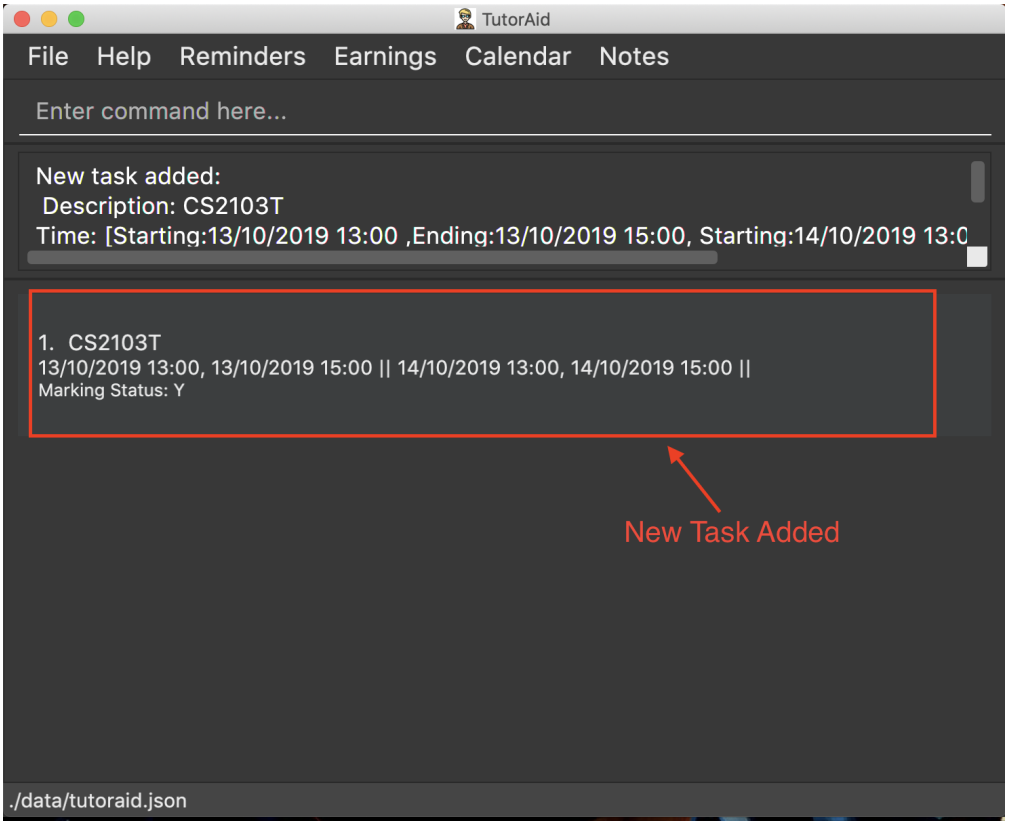
Adds a task to one or more time slots.
Format: `add_task c/MODULE mark/STATUS tt/TASK_TIME…`

TIP	A task can have more than one time slots. STATUS should only be Y or N. `TASK_TIME` should be in the format "dd/MM/YYYY HH:mm, dd/MM/YYYY HH:mm". If there are multiple task times, they will be automatically sorted based on their starting time.

Examples:

- `add_task c/CS2103T mark/Y tt/13/09/2019 13:00, 20/09/2019 16:00 tt/21/09/2019 13:00, 21/09/2019 15:00`





- `add_task c/MA1521 Tutorial mark/N tt/02/11/2020 14:00, 02/11/2020 15:00`

Command Summary

- **Help** : `help`
- **Log** :
  `login user/USERNAME pass/PASSWORD`
  `register user/USERNAME pass/PASSWORD`
  `logout`
- **Tab** : `tab TAB_DESTINATION`
- **Calendar:**
  `add_task c/MODULE mark/STATUS tt/TASK_TIME…`
  `edit_task INDEX [mark/STATUS] [tt/TASK_TIME]`
  `delete_task 1`
  `find_task_by_module MODULE …`
  `find_task_by_date DATE …`
  `list_task`
- **Reminder** : `reminder INDEX STATUS`
- **Earnings** :
  `add_earnings d/DATE c/CLASSID amt/AMOUNT`
  `update_earnings d/DATE c/CLASSID amt/AMOUNT type/TYPE`
  `delete_earnings d/DATE c/CLASSID`
  `find_earnings k/KEYWORD …`
  `claim_earnings d/DATE c/CLASSID`
  `filter_earnings VARIABLE`

- **Note** :
  `addnote mod/MODULE_CODE c/CONTENT`
  `editnote INDEX mod/MODULE_CODE c/CONTENT`
  `deletenote INDEX`
  `findnote KEYWORD`
  `listnote`
- **Student List** :
  `add n/NAME c/CLASSID`
  `delete INDEX`
  `edit INDEX n/NAME pic/PICTURE r/RESULT att/ATTENDANCE part/PARTICIPATION c/CLASS`
  `list`
  `find NAME`
  `set_pic INDEX pic/FILENAME`
  `assign_class INDEXES c/CLASSID`
  `list_class CLASSID`
  `mark_attendance INDEXES`
  `mark_participation INDEXES`
- **Undo** : `undo`
- **Redo** : `redo`
- **Clear** : `clear`
- **Exit** : `exit`

Contributions to the Developer Guide

> *Given below are some 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. Because of the limitation of number of pages, some contributions like the explanation of the add_task command, the considerations for undo/redo command, as well as the use case are not shown below.*

Undo/Redo feature

The undo/redo mechanism is facilitated by `VersionedTutorAid`. It extends `TutorAid` with an undo/redo history, stored internally as an `tutorAidStateList` and `currentStatePointer`. Additionally, it implements the following operations:

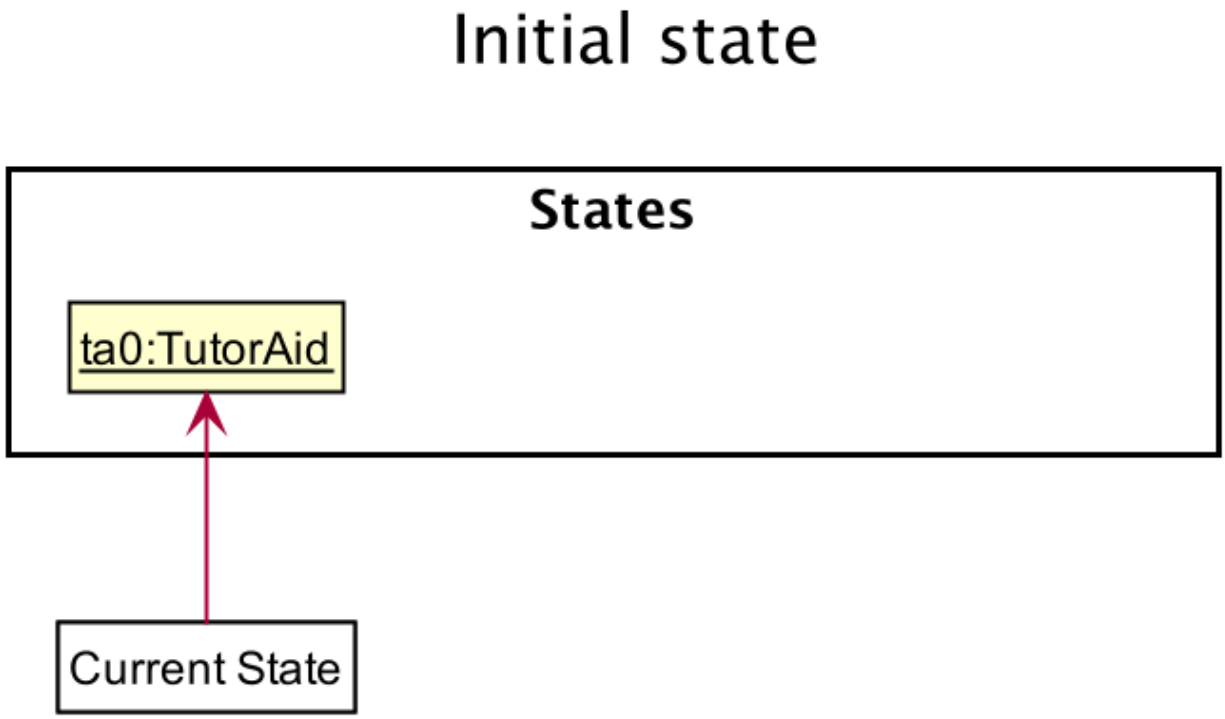
- `VersionedTutorAid#commit()` — Saves the current tutor aid state in its history.
- `VersionedTutorAid#undo()` — Restores the previous tutor aid state from its history.
- `VersionedTutorAid#redo()` — Restores a previously undone tutor aid state from its history.

These operations are exposed in the `Model` interface as `Model#commitTutorAid()`, `Model#undoTutorAid()` and `Model#redoTutorAid()` respectively.

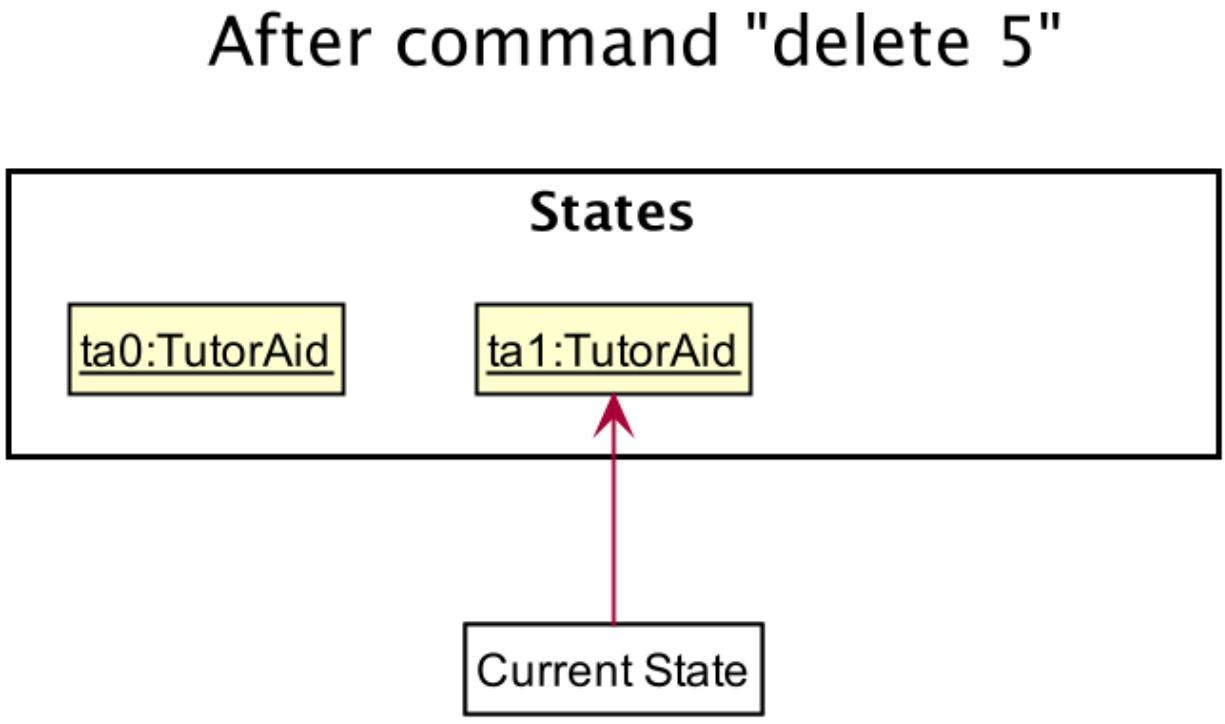
Given below is an example usage scenario and how the undo/redo mechanism behaves at each

step.

Step 1. The user launches the application for the first time. The `VersionedTutorAid` will be initialized with the initial tutor aid state, and the `currentStatePointer` pointing to that single tutor aid state.

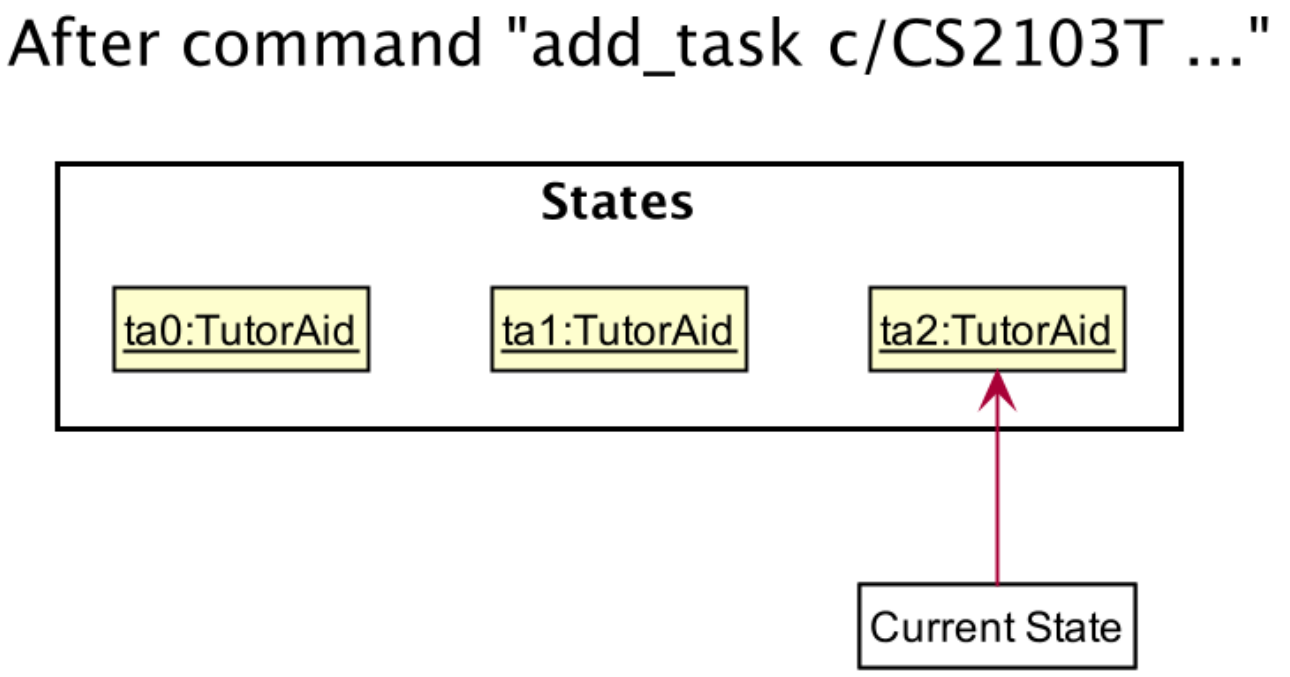


Step 2. The user executes `delete 5` command to delete the 5th person in the tutor aid. The `delete` command calls `Model#commitTutorAid()`, causing the modified state of the tutor aid after the `delete 5` command executes to be saved in the `tutorAidStateList`, and the `currentStatePointer` is shifted to the newly inserted tutor aid state.



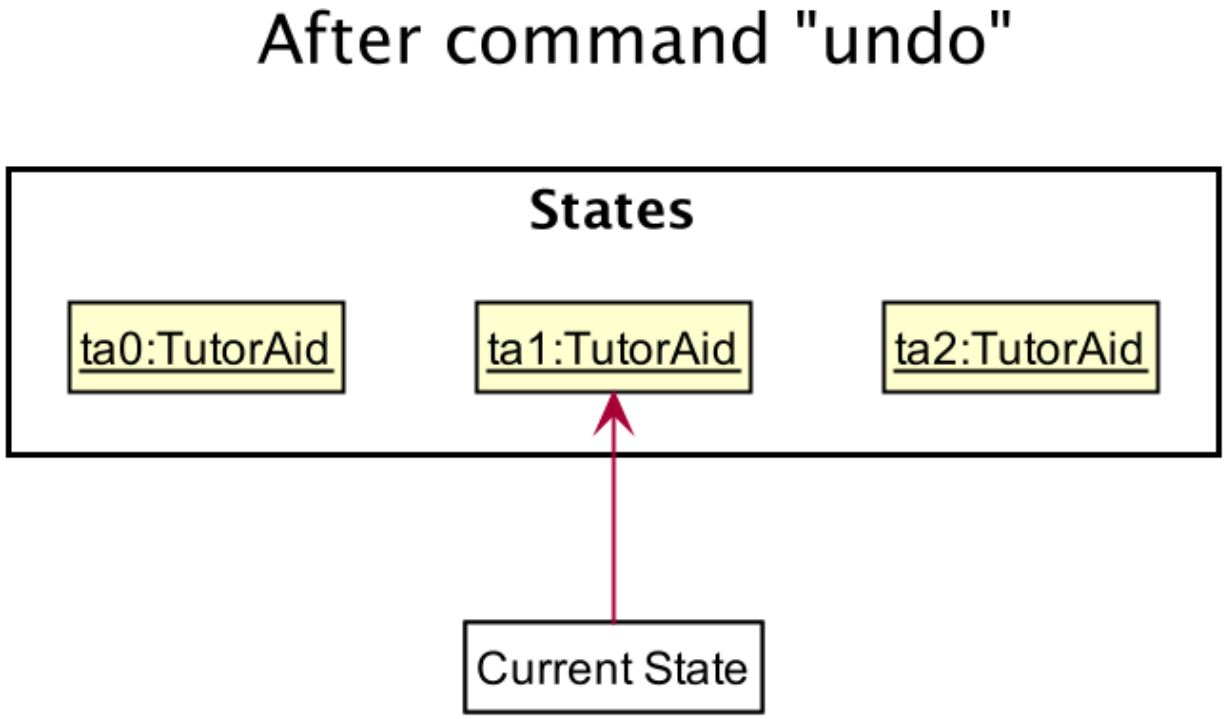
Step 3. The user executes `add_task c/CS2103T ⋯` to add a new task. The `add_task` command also calls

`Model#commitTutorAid()`, causing another modified tutor aid state to be saved into the `tutorAidStateList`.



NOTE If a command fails its execution, it will not call `Model#commitTutorAid()`, so the tutor aid state will not be saved into the `tutorAidStateList`.

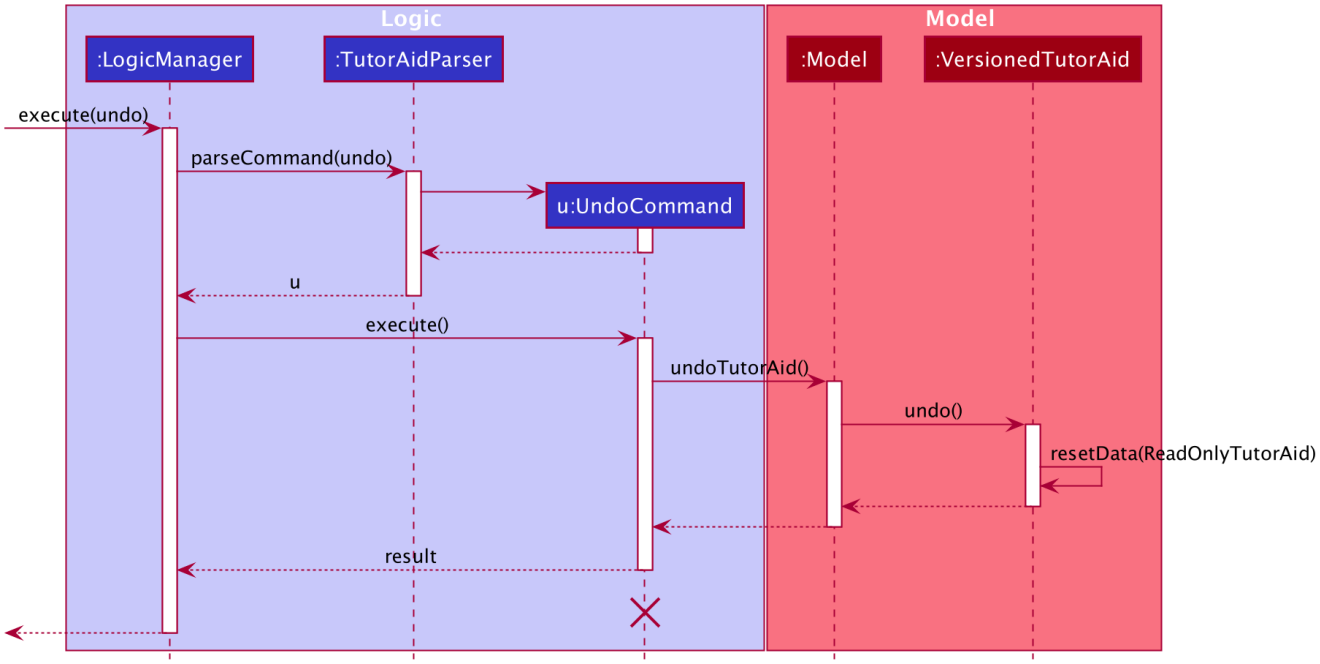
Step 4. The user now decides that adding the task was a mistake, and decides to undo that action by executing the `undo` command. The `undo` command will call `Model#undoTutorAid()`, which will shift the `currentStatePointer` once to the left, pointing it to the previous tutor aid state, and restores the tutor aid to that state.



NOTE

If the `currentStatePointer` is at index 0, pointing to the initial tutor aid state, then there are no previous tutor aid states to restore. The `undo` command uses `Model#canUndoTutorAid()` to check if this is the case. If so, it will return an error to the user rather than attempting to perform the undo.

The following sequence diagram shows how the undo operation works:



NOTE

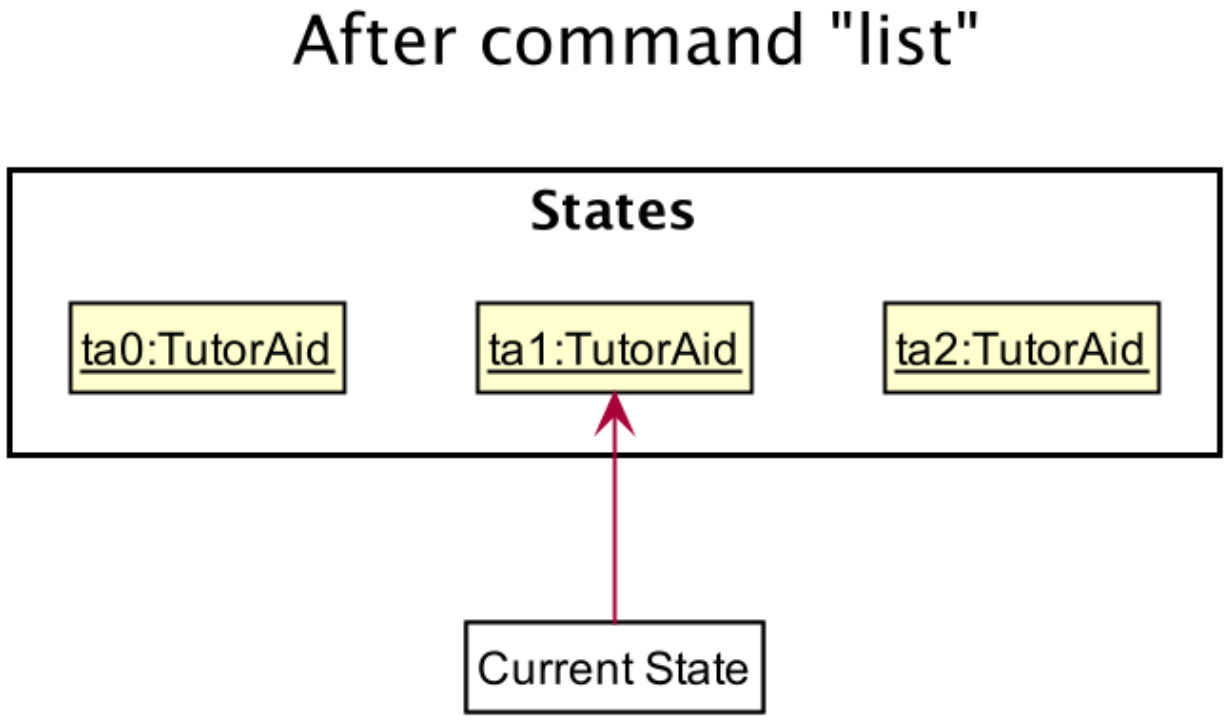
The lifeline for `UndoCommand` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

The `redo` command does the opposite — it calls `Model#redoTutorAid()`, which shifts the `currentStatePointer` once to the right, pointing to the previously undone state, and restores the tutor aid to that state.

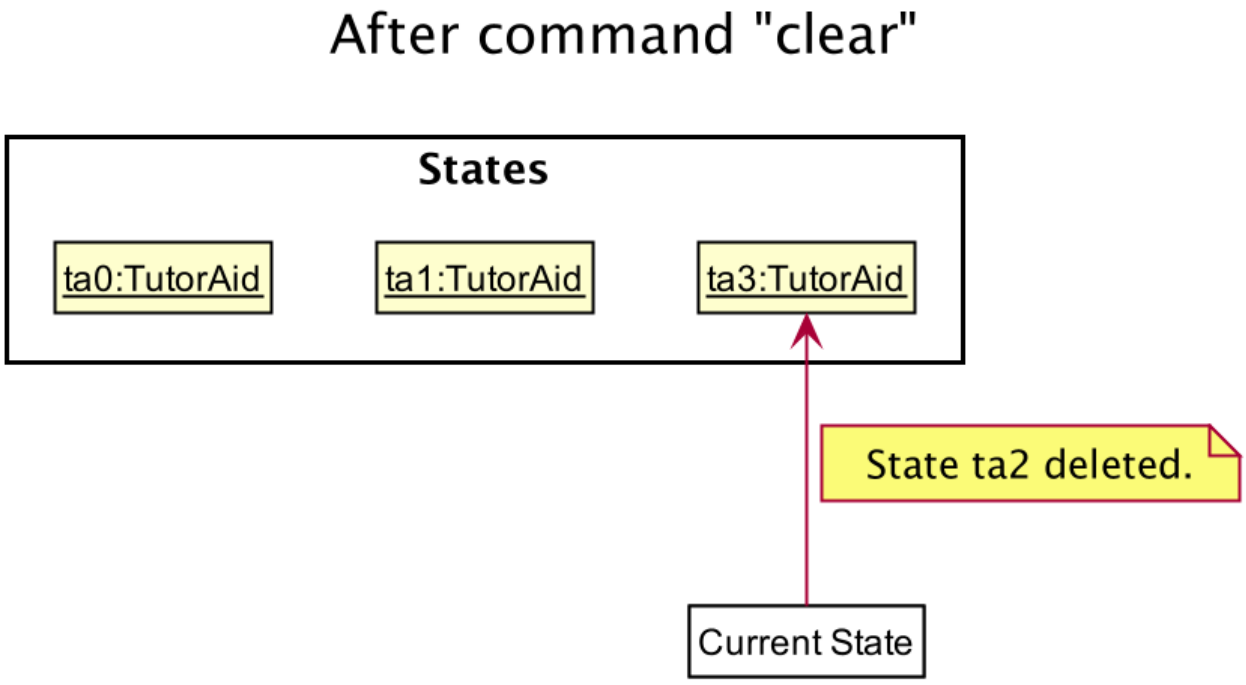
NOTE

If the `currentStatePointer` is at index `tutorAidStateList.size() - 1`, pointing to the latest tutor aid state, then there are no undone tutor aid states to restore. The `redo` command uses `Model#canRedoTutorAid()` to check if this is the case. If so, it will return an error to the user rather than attempting to perform the redo.

Step 5. The user then decides to execute the command `list`. Commands that do not modify the tutor aid, such as `list`, will usually not call `Model#commitTutorAid()`, `Model#undoTutorAid()` or `Model#redoTutorAid()`. Thus, the `tutorAidStateList` remains unchanged.



Step 6. The user executes `clear`, which calls `Model#commitTutorAid()`. Since the `currentStatePointer` is not pointing at the end of the `tutorAidStateList`, all tutor aid states after the `currentStatePointer` will be purged. We designed it this way because it no longer makes sense to redo the `add_task c/CS2103T ⋯` command. This is the behavior that most modern desktop applications follow.



Appendix A: User Stories

Priorities: High (must have) - * * *, Medium (nice to have) - * *, Low (unlikely to have) - *

Priority	As a …	I want to …	So that I can…
* * *	new user	see usage instructions	refer to instructions when I forget how to use the App
* * *	tutor	add a new class	check the details of the task when I want
* * *	tutor	edit an existing task	update task information when I need
* * *	tutor	add my earnings	check my earnings when I want to
* * *	tutor	check my existing classes	attend the classes in time
* * *	tutor	check the information of my classes	know more about my students' situation
* *	user	know what's the command format	key in the correct command
* *	forgetful tutor user	be reminded before my tutorials	go for the tutorials on time
*	caring tutor user	check the upcoming events	remind my students