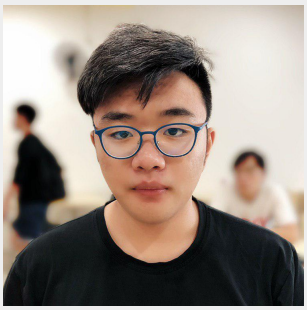
Xuerneas - Project Portfolio

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 enhancements in detail.

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`

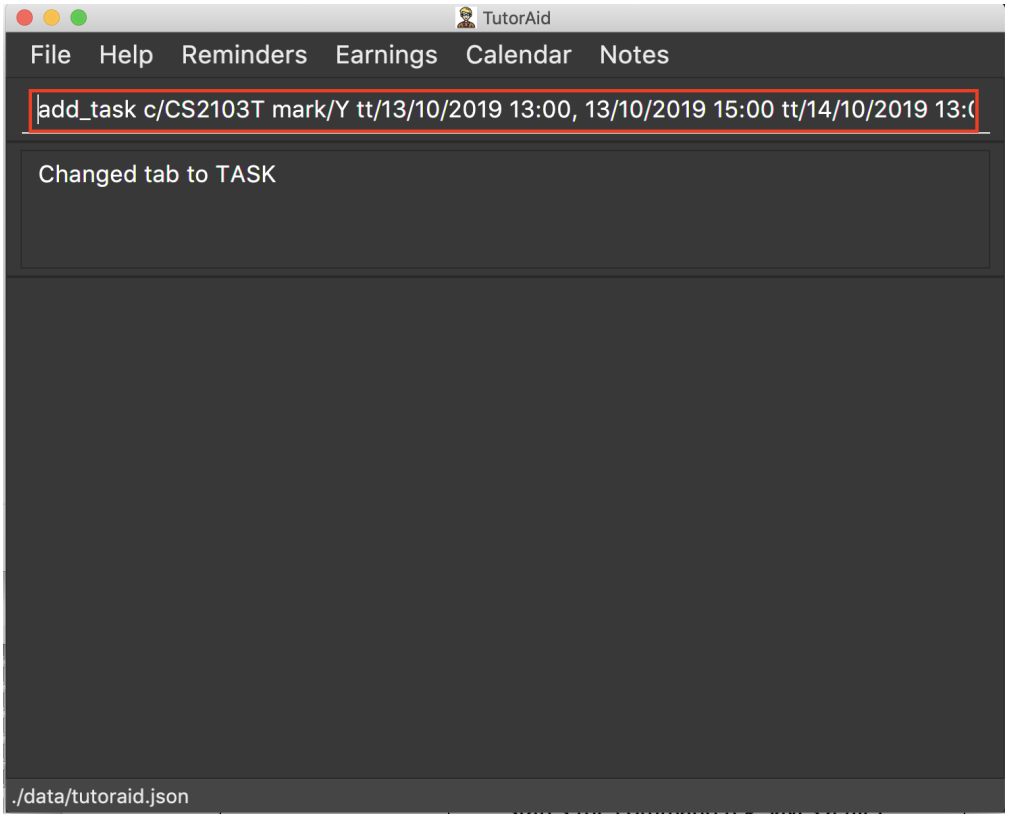
Adds a task to one or more time slots.

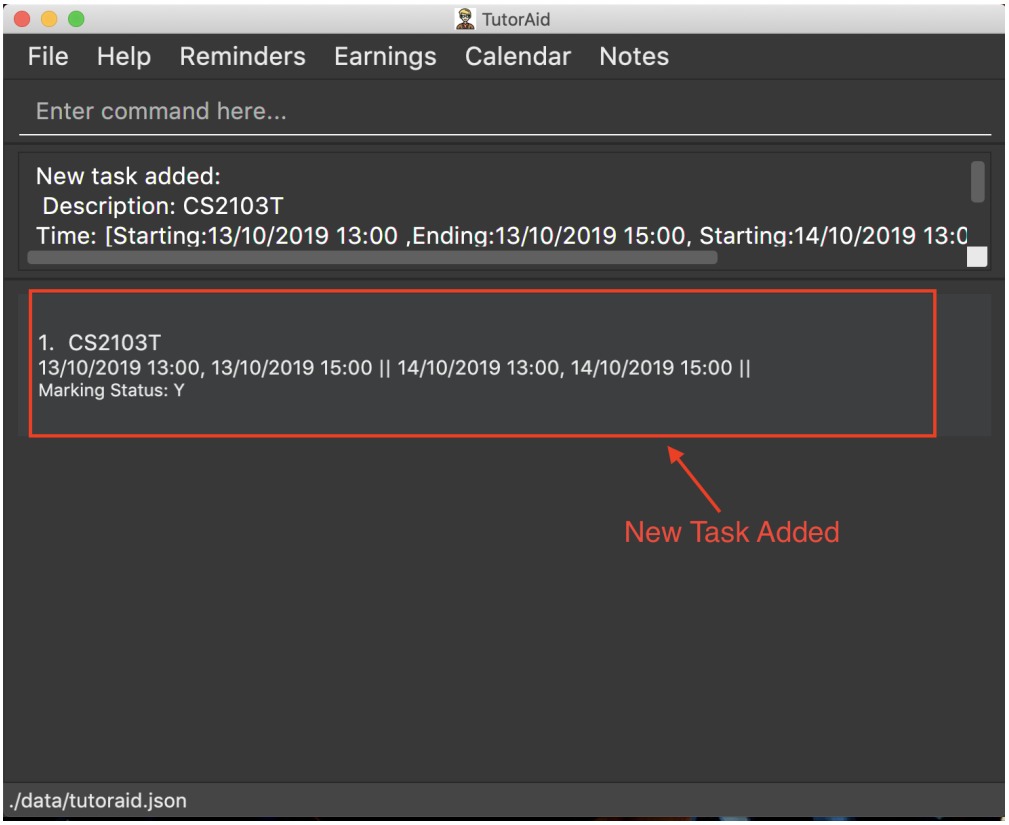
Format: `add_task c/MODULE mark/STATUS tt/TASK_TIME…`

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. Mark indicates whether a Reminder will be created for this task. The Reminder created will have the Task's `MODULE` as its `DESCRIPTION` and the Task's `TASK_TIME` as its `DATE`.

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`





- The Reminder created will have the Task's `MODULE` as its `DESCRIPTION` and the Task's `TASK_TIME` as its `DATE`.

Command Summary

- **Help** : `help`
- **Log** :
  `login user/USERNAME pass/PASSWORD`
  `register user/USERNAME pass/PASSWORD`
  `logout`
- **Tab** :
  `change_tab tab/TAB_DESTINATION`
- **Task:**
  `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** :
  `add_reminder rd/DESCRIPTION rt/REMINDER_TIME…`
  `delete_reminder 1`
  `find_reminder_by_description DESCRIPTION …`
  `find_reminder_by_date DATE …`

`list_reminder`

- **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 c/MODULE_CODE type/CLASS_TYPE note/NOTE_CONTENT`
  `editnote INDEX c/MODULE_CODE type/CLASS_TYPE note/NOTE_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, the manual testing instructions of some task and undo/redo commands, as well as some of the use cases 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,

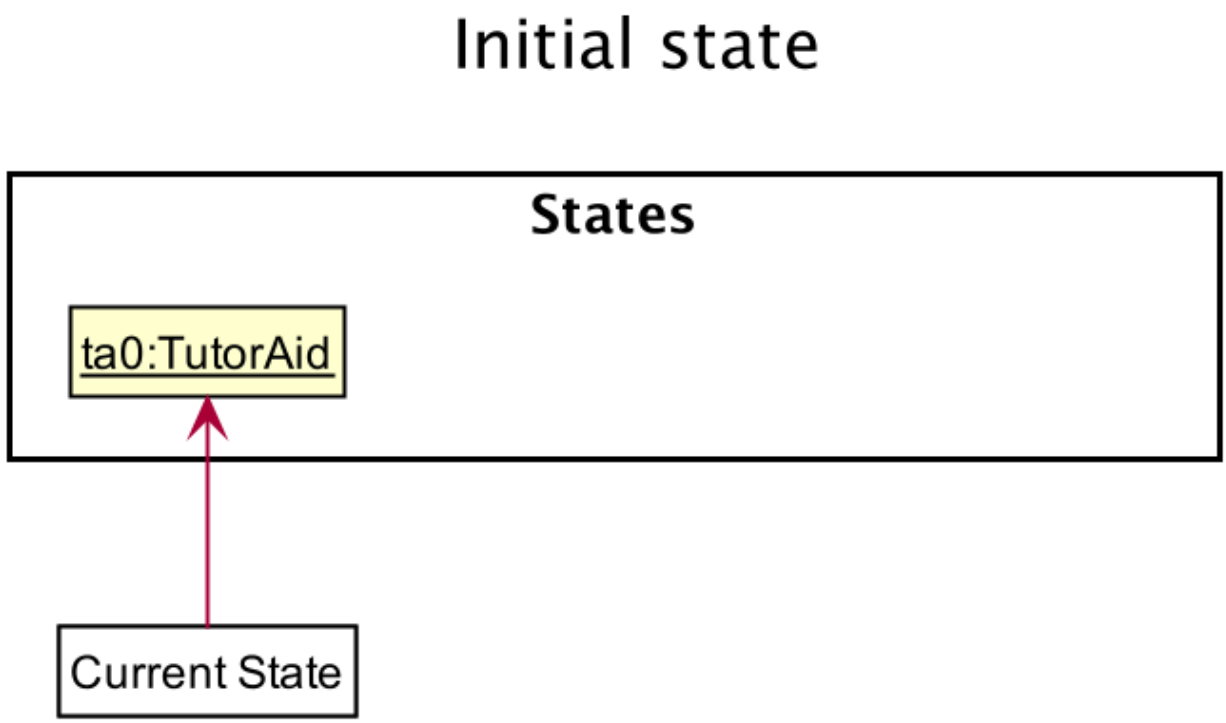
it implements the following operations:

- `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.

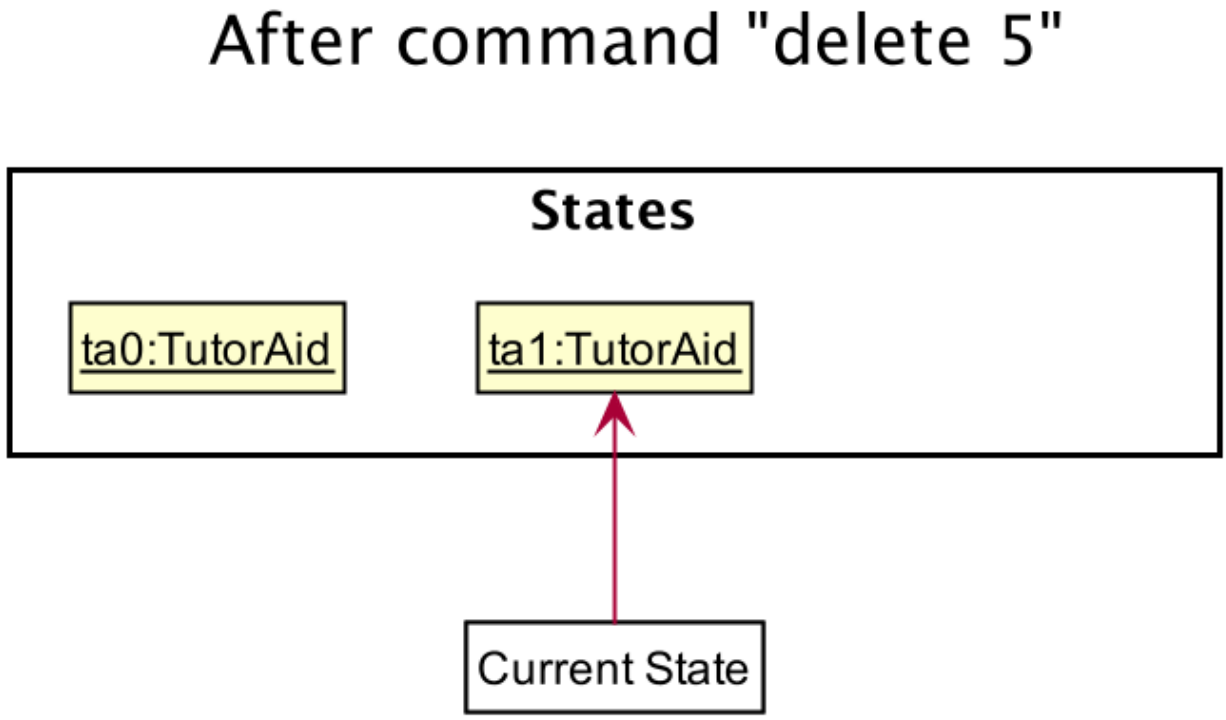
These operations are exposed in the `Model` interface as `Model#commitTutorAid()`, `Model#undoTutorAid()` and `Model#redoTutorAid()` respectively.

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

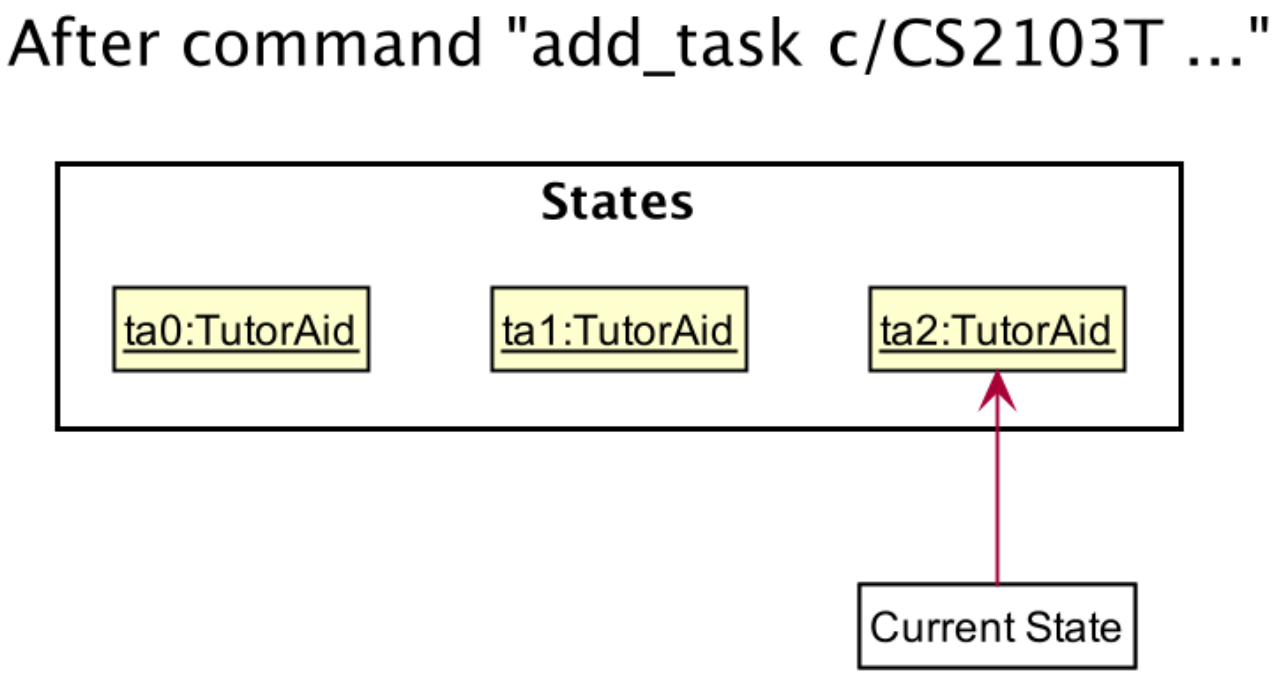
Step 1. The user launches the application for the first time. The `VersionedTutorAid` will be initialized with the initial tutor aid state, and the `currentStatePointer` pointing to that single tutor aid state.



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.

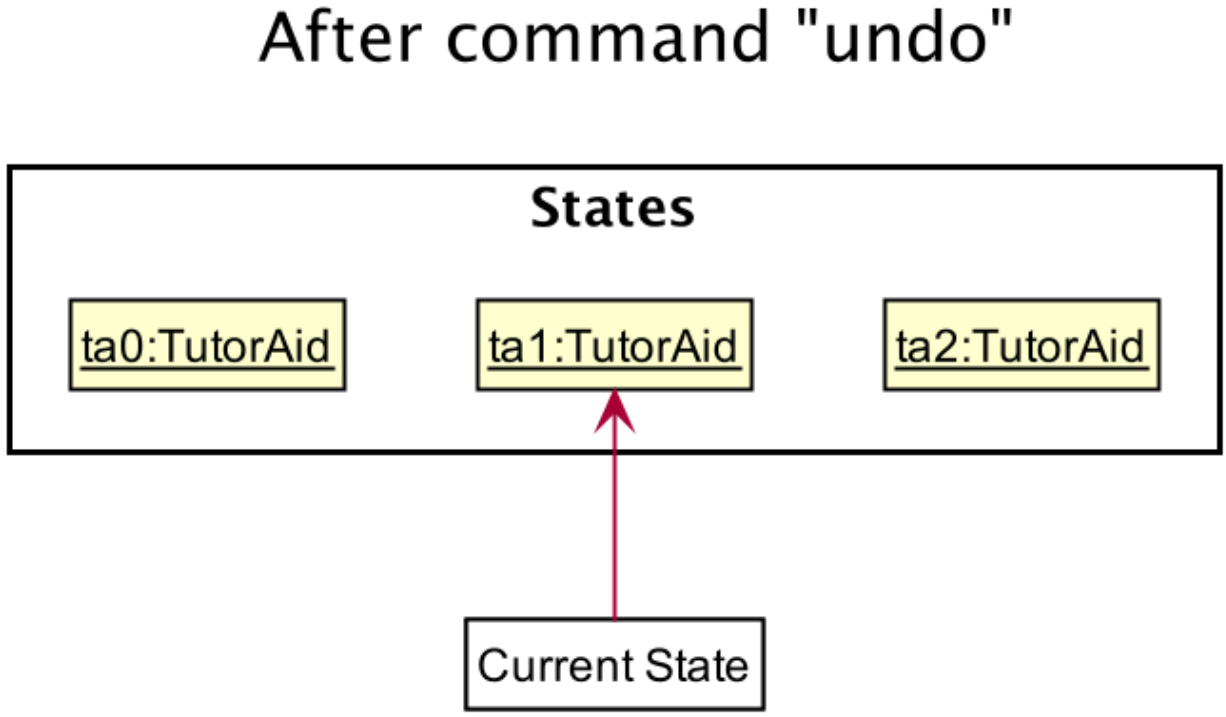


Step 3. The user executes `add_task c/CS2103T ⋯` to add a new task. The `add_task` command also calls `Model#commitTutorAid()`, causing another modified tutor aid state to be saved into the `tutorAidStateList`.



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`.

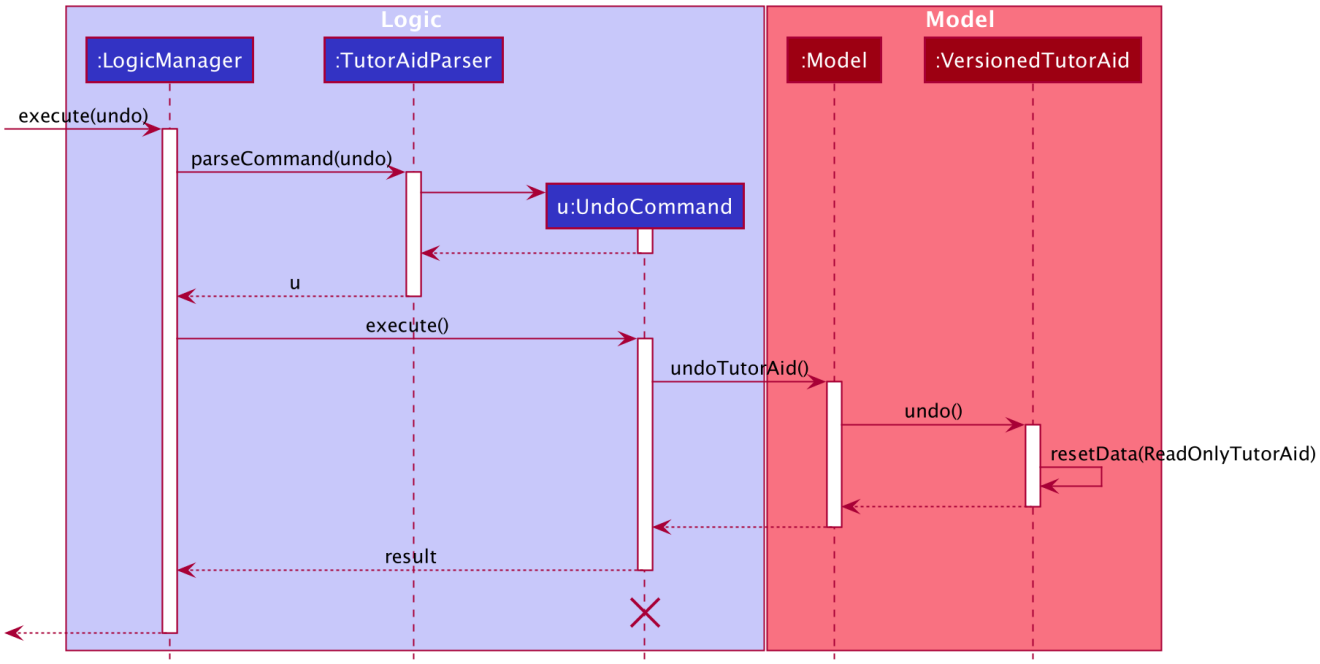
Step 4. The user now decides that adding the task was a mistake, and decides to undo that action by executing the `undo` command. The `undo` command will call `Model#undoTutorAid()`, which will shift the `currentStatePointer` once to the left, pointing it to the previous tutor aid state, and restores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

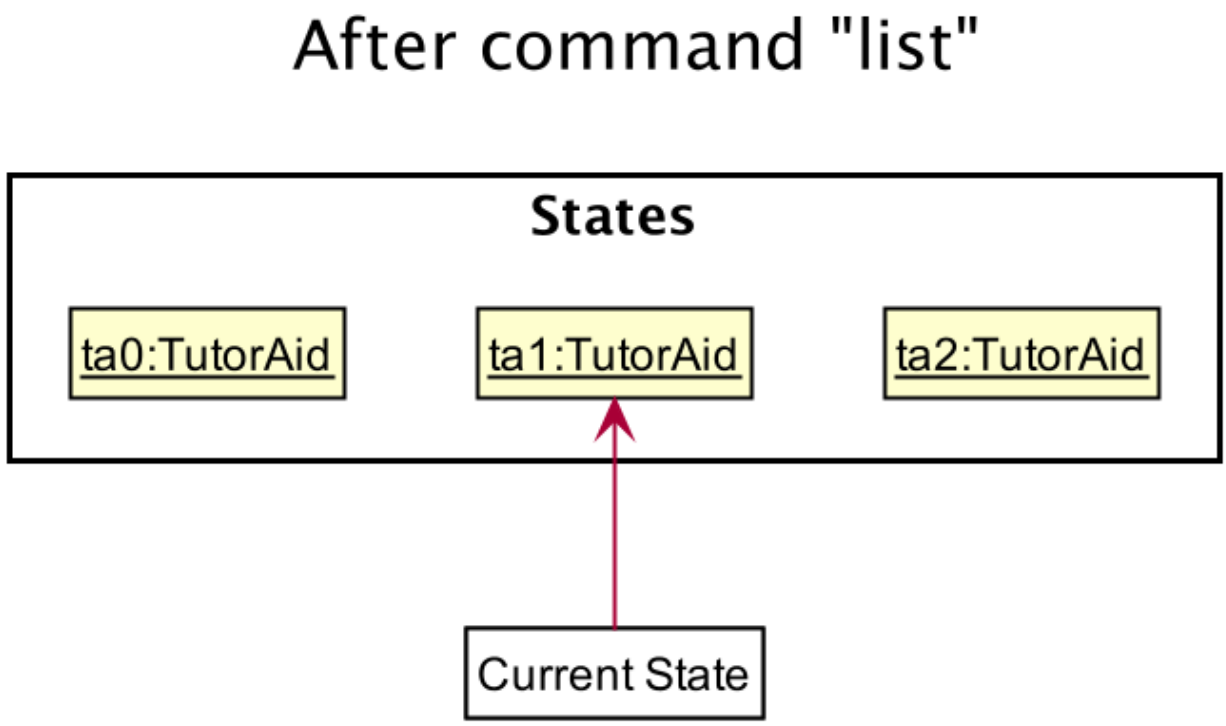
The lifeline for `UndoCommand` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

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.

Design Considerations

Aspect: How undo & redo executes

- **Alternative 1 (current choice):** Saves the entire tutor aid.
  - Pros: Easy to implement.
  - Cons: May have performance issues in terms of memory usage.
- **Alternative 2:** Individual command knows how to undo/redo by itself.
  - Pros: Will use less memory (e.g. for `delete`, just save the person being deleted).
  - Cons: We must ensure that the implementation of each individual command are correct.

Tab Change feature

Tab Change feature is a type of Command that allows users to change to respective windows for using different features in the system. Tab change function is implemented in using both CLI and GUI. User can execute tab change by typing command in the command box or by interaction with the GUI component (Menu Bar)

Implementation

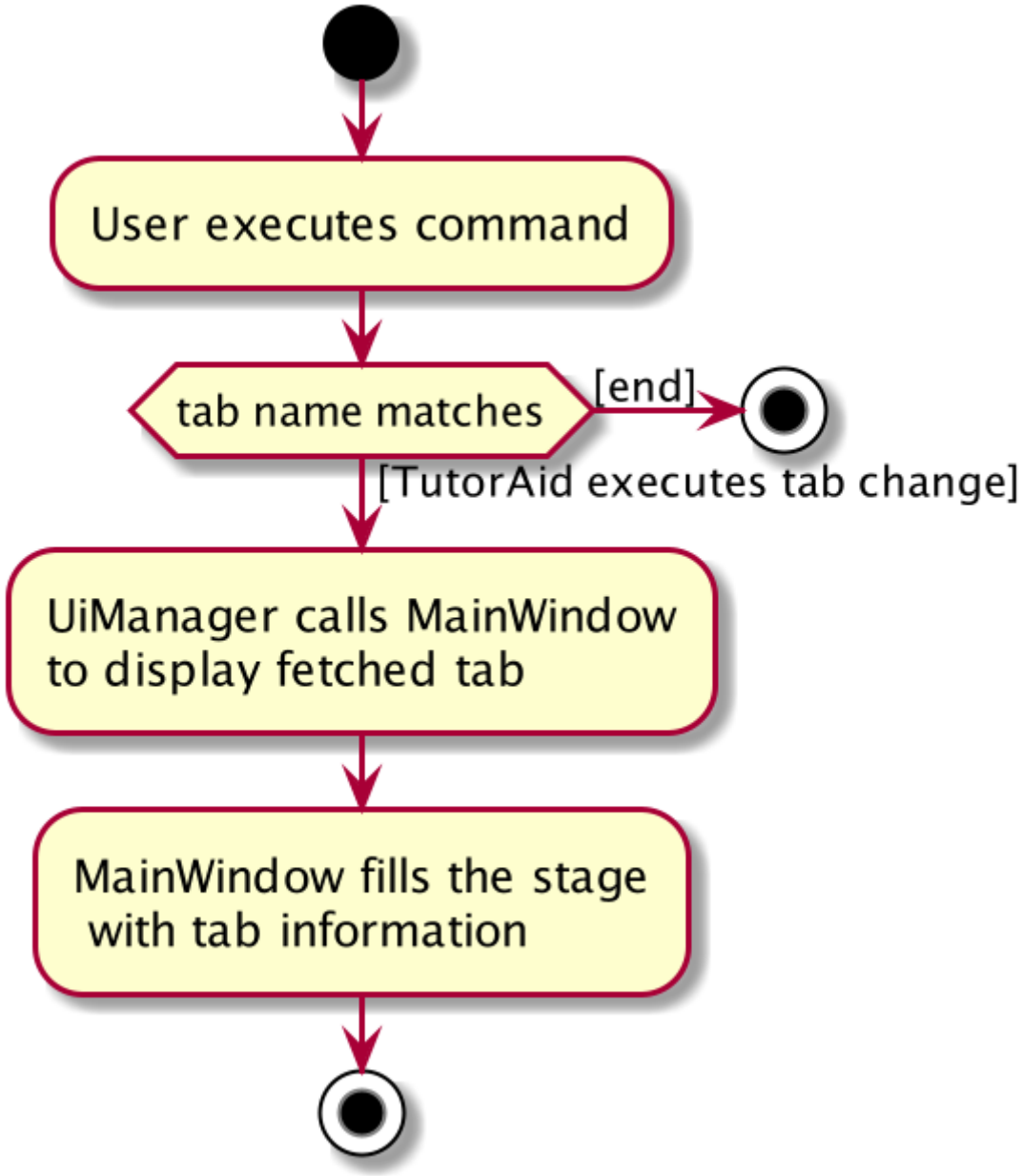


Figure 1. Activity Diagram

User will execute `change_tab` command with the parameters of `tab/TAB`. Using this, TutorAid will display the list and the view that the user wishes to see, effectively changing tabs.

Logging

We are using `java.util.logging` package for logging. The `LogsCenter` class is used to manage the logging levels and logging destinations.

- The logging level can be controlled using the `logLevel` setting in the configuration file (See Configuration)
- The `Logger` for a class can be obtained using `LogsCenter.getLogger(Class)` which will log messages according to the specified logging level
- Currently log messages are output through: `Console` and to a `.log` file.

Logging Levels

- `SEVERE` : Critical problem detected which may possibly cause the termination of the application
- `WARNING` : Can continue, but with caution
- `INFO` : Information showing the noteworthy actions by the App
- `FINE` : Details that is not usually noteworthy but may be useful in debugging e.g. print the actual list instead of just its size

Configuration

Certain properties of the application can be controlled (e.g user prefs file location, logging level) through the configuration file (default: `config.json`).

Documentation

Refer to the guide here.

Testing

Refer to the guide here.

Dev Ops

Refer to the guide here.

Appendix A: Product Scope

Target user profile:

- has a need to manage classes and related tasks
- has a need to track earnings
- prefer desktop apps over other types

- can type fast
- prefers typing over mouse input
- is reasonably comfortable using CLI apps

Value proposition: manage students, tasks, notes, earnings and reminders faster than a typical mouse/GUI driven app. Helps to improve workflow by organizing all teaching-related information in one common place and hence saves time.

Appendix B: 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	track all the information about my students	remember key info about them easily when I need it
* * *	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
* *	tutor	see who’s not coming for class	check up on them

Priority	As a …	I want to …	So that I can…
* *	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

{More to be added}

Appendix C: Use Cases

(For all use cases below, the **System** is the `TutorAid` and the **Actor** is the `user`, unless specified otherwise)

Use case: Delete task

MSS

1. User requests to list tasks
2. TutorAid shows a list of tasks
3. User requests to delete a specific task in the list
4. TutorAid deletes the person

   Use case ends.

Extensions

2a. The list is empty.

  2a1. TutorAid tells user that there is no task.

  Use case ends.

3a. The given index is invalid.

  3a1. TutorAid shows an error message.

  Use case resumes at step 2.

Use case: Delete Reminder

MSS

1. User requests to list reminders
2. TutorAid shows a list of reminders
3. User requests to delete a specific reminder in the list
4. TutorAid deletes the reminder

   Use case ends.

Extensions

2a. The list is empty.

   2a1. TutorAid tells user that there are no reminders.

   Use case ends.

3a. The given index is invalid.

   3a1. TutorAid shows an error message.

   Use case resumes at step 2.

Use case: Add Earnings

MSS

1. User adds an earnings by specifying its details
2. TutorAid shows a success message

   Use case ends.

Extensions

- 1a. The arguments provided are invalid.
  - 1a1. TutorAid shows an error message.

    Use case ends.
- 1b. The mandatory arguments are not provided.
  - 1b1. TutorAid shows an error message.

    Use case ends.

Use case: Delete Earnings

MSS

1. User requests to change tab to earnings
2. TutorAid shows a list of earnings
3. User requests to delete a specific earnings in the list
4. TutorAid deletes the earnings

   Use case ends.

Extensions

- 2a. The list is empty.

  Use case ends

- 3a. The given index is invalid.
  - 3a1. TutorAid shows an error message.

    Use case resumes at step 2.

Use case: Find Earnings

MSS

1. User requests to find earnings with specified keyword(s)
2. TutorAid shows a success message

   Use case ends.

Extensions

- 1a. The mandatory arguments are not provided.
  - 1a1. TutorAid shows an error message.

    Use case ends.

- 2a. The arguments provided have no match.
  - 2a1. TutorAid shows an empty list.

    Use case ends.

Use case: Claim Earnings

MSS

1. User requests to claim earnings with specified arguments
2. TutorAid shows a success message

Use case ends.

Extensions

- 1a. The mandatory arguments are not provided.
  - 1a1. TutorAid shows an error message.

    Use case ends.
- 1b. The arguments provided are invalid.
  - 1b1. TutorAid shows an error message.

    Use case ends.

Use case: Automate Earnings

MSS

1. User requests to change tab to earnings
2. TutorAid shows a list of earnings
3. User requests to automate a specific earnings in the list
4. TutorAid add the earnings into a list.

   Use case ends.

Extensions

- 2a. The list is empty.

  Use case ends
- 3a. The mandatory arguments are not provided.
  - 3a1. TutorAid shows an error message.

    Use case ends.
- 3b. The arguments provided are invalid.
  - 3b1. TutorAid shows an error message.

    Use case ends.
- 3c. The given index is invalid.
  - 3c1. TutorAid shows an error message.

    Use case resumes at step 2.

Use case: Add Automated Earnings

MSS

1. User requests to add automated earnings
2. TutorAid shows a success message

   Use case ends.

Extensions

- 1a. No new earnings are added.
  - 1a1. TutorAid shows an error message.

    Use case ends

{More to be added}

Appendix D: Non Functional Requirements

1. Should work on any mainstream OS as long as it has Java 11 or above installed.
2. Should be able to hold up to 1000 tasks without a noticeable sluggishness in performance for typical usage.
3. A user with above average typing speed for regular English text (i.e. not code, not system admin commands) should be able to accomplish most of the tasks faster using commands than using the mouse.

Appendix E: Glossary

Mainstream OS

Windows, Linux, Unix, OS-X

Private contact detail

A contact detail that is not meant to be shared with others

Appendix F: Instructions for Manual Testing

Given below are instructions to test the app manually.

NOTE	These instructions only provide a starting point for testers to work on; testers are expected to do more *exploratory* testing.

Launch

1. Initial launch
    a. Download the jar file and copy into an empty folder
    b. Double-click the jar file
    Expected: Shows the GUI with a set of sample contacts. The window size may not be optimum.
2. Saving window preferences
    a. Resize the window to an optimum size. Move the window to a different location. Close the window.
    b. Re-launch the app by double-clicking the jar file.
    Expected: The most recent window size and location is retained.

Registering an account

1. Registering an account
    a. Test case: `register user/bryan pass/Pa55w0rd`
    Expected: Success message shown in status bar.
    b. Test case: `register user/abc pass/hello`
    Expected: No account created. Error details shown in the status message. Status bar remains the same.
    c. Other incorrect register commands to try: `register`, `register bryan Pa55w0rd`
    Expected: Similar to previous.

Logging In

1. Logging into TutorAid with personal account
    a. Test case: `login user/bryan pass/Pa55w0rd`
    Expected: Login Window closes and Main Window pops up.
    b. Test case: `login user/wad efsf pass/hello`
    Expected: Not logged in. Error details shown in the status message. Status bar remains the same.
    c. Other incorrect login commands to try: `login`, `login bryan Pa55w0rd`
    Expected: Similar to previous.

Deleting a person

1. Deleting a person while all persons are listed
    a. Prerequisites: List all persons using the `list` command. Multiple persons in the list.
    b. Test case: `delete 1`
    Expected: First person is deleted from the list. Details of the deleted person shown in the

status message.

c. Test case: `delete 0`
Expected: No person is deleted. Error details shown in the status message.

d. Other incorrect delete commands to try: `delete`, `delete x` (where x is larger than the list size), `delete Tom`
Expected: Similar to previous.

Adding an earning

1. Adding earnings to TutorAid

   a. Prerequisites: Arguments are valid and mandatory parameters are provided.

   b. Test case: `add_earnings d/11/10/2014 type/lab c/CS1101S amt/90.30`
   Expected: Adds an earnings of $90.30 of a CS1101S Lab lesson to TutorAid on 11/10/2014

   c. Test case: `add_earnings d/10/03/2019 type/tut`
   Expected: No earnings is added Error details shown in the status message. Status bar remains the same.

   d. Other incorrect add earnings commands to try: `add_earnings`, `add_earnings d/today type/c` and `add_earnings 2`
   Expected: Similar to previous.

Deleting an earning

1. Deleting earnings while all earnings are listed

   a. Prerequisites: List all earnings either using tab button on application or `change_tab tab/earnings` command. Multiple earnings in the list.

   b. Test case: `delete_earnings 1`
   Expected: First earnings is deleted from the list. Details of the deleted earnings shown in the status message.

   c. Test case: `delete_earnings 0`
   Expected: No earnings is deleted. Error details shown in the status message. Status bar remains the same.

   d. Other incorrect delete earnings command to try: `delete_earnings`, `delete_earnings x` (where x is larger than the list size).
   Expected: Similar to previous.

Finding an earning

1. Finding earnings

   a. Test case: `find_earnings CS`
   Expected: All earnings that contains "CS" (regardless case of the letters) will show in TutorAid. Partial match of "CS" will be shown as well. Number of earnings found were stated in the status message.

b. Test case: `find_earnings`
Expected: No earnings will be shown. Error details shown in the status message. Status bar remains the same.

Claiming an earning

1. Change the Claim status of an earning

   a. Prerequisites: List all earnings either using tab button on application or `change_tab tab/earnings` command. Multiple earnings in the list.

   b. Test case: `claim_earnings 1 claim/processing`
   Expected: First earnings will change claim status to `processing`.

   c. Test case: `claim_earnings 1`
   Expected: No change in claim status of the first earnings. Error details shown in the status message. Status bar remains the same.

   d. Other incorrect claim earnings command to try: `claim_earnings 1 claim/done`, `claim_earnings`
   Expected: Similar to previous.

Automating earnings

1. Automate the addition of earnings

   a. Prerequisites: List all earnings either using tab button on application or `change_tab tab/earnings` command. Multiple earnings in the list.

   b. Test case: `weekly_earnings 1 count/2`
   Expected: Success message shows up. No change in earnings list. First earnings will be automatically added in the next 2 weeks on the same day of the week, after invoking the `auto` command.

   c. Test case: `weekly_earnings 0 count/4`
   Expected: No earnings will be automatically added. Error details shown in the status message. Status bar remains the same.

   d. Other incorrect weekly earnings command to try: `weekly_earnings`, `weekly_earnings 3 count/15`, `weekly_earnings x count/8` (where x is larger than the list size).
   Expected: Similar to previous.

Adding automated earnings

1. Adds automated earnings to list.

   a. Prerequisites: There must have been a few pre-existing earnings that were invoked by the `weekly_earnings` command.

   b. Test case: `auto`
   Expected: Depending on the number of earnings that were invoked by `weekly_earnings` command and the day of the week, earnings will be automatically added to the list.

c. Test case: `auto 5`
Expected: No earnings are added. Error details shown in the status message. Status bar remains the same.

Saving data

Adding a task

1. Add a task and list all tasks out with the added task.

   a. Test case: `add_task c/CS2103T mark/Y tt/13/09/2019 13:00, 13/09/2019 16:00`
   Expected: A marked task with Class Id "CS2103T" and Task Time "13/09/2019 13:00 to 16:00" was added to the task list. Details of the added task shown in the status message. Task list shown in Main window.
   Reminder with Description "CS2103T" and Time "13/09/2019 13:00, 20/09/2019 16:00" will be created as well.
   A Task will be shown on that date in the Calendar view as well.

   b. Test case: `add_task`
   Expected: No task is added. Error details and correct format shown in the status message.

   c. Other incorrect command format to try: `add_task task`, `add_task c/ mark/ tt/`

Editing a task

1. Edit a task and list all tasks out with the edited task.

   a. Test case: `edit_task 1 c/CS2103`
   Expected: The first task in the task list's Class Id changed to CS2103. Details of the edited task shown in the status message. Task list shown in Main window.

   b. Test case: `edit_task c/cs2103`
   Expected: No task is changed. Error details and correct format shown in the status message.

   c. Other incorrect command format to try: `edit_task` `edit_task 1 c/`

   d. Reminder: Make sure the index of the task is not larger than the total number of tasks in task list.

Deleting a task

1. Delete a task and list all tasks out without the deleted task.

   a. Test case: `delete_task 1`
   Expected: The first task in the task list was deleted. Details of the deleted task shown in the status message. Task list shown in Main window.

   b. Test case: `delete_task`
   Expected: No task is deleted. Error details and correct format shown in the status message.

   c. Other incorrect command format to try: `delete_task 0` `delete_task 1 c/`

   d. Reminder: Make sure the index of the task is not larger than the total number of tasks in

task list.

e. Reminder: If you add a marked task then delete it, remember to delete it in Reminder also so that you can add another marked task with same Class Id.

Finding a task

1. Find tasks by their Class Id or its Task Time.

   a. Test case: `find_task_by_module cs2100`
   Expected: All tasks that Class Id contains "cs2100" (regardless case of the letters) were shown in the MainWindow. Number of tasks found were stated in the status message.

   b. Test case: `find_task_by_date 20/10/2019`
   Expected: All tasks that Task Time contains "20/10/2019" were shown in the MainWindow. Number of tasks found were stated in the status message.

   c. Test case: `find_task_by_module`
   Expected: Error details and correct format shown in the status message.

   d. Other incorrect command format to try: `find_task_by_date`, `find_task_by_date 13/10/2019 15/10/2019`

   e. Reminder: Tasks can be found by multiple numbers of Class Ids but only one Date.

Listing all tasks

1. List out all tasks.

   a. Test case: `list_task`
   Expected: All tasks were listed in Main Window.

   b. Test case: `list tasks`
   Expected: Error details shown in the status message.

Undoing/Redoing commands

1. Undo or redo previous commands.

   a. Test case: `delete_task 1` + `undo`
   Expected: The previously deleted task reappeared in the list.

   b. Test case: `delete_task 1` + `undo` + `redo`
   Expected: The recovered task was deleted again from the list.

Adding a reminder

1. Add a reminder and list all reminders out with the added reminder.

   a. Test case: `add_reminder rd/CS2103T rt/13/09/2019 13:00, 20/09/2019 16:00`
   Expected: A Reminder with Class Id "CS2103T" and Task Time "13/09/2019 13:00, 20/09/2019 16:00" was added to the reminder list. Details of the added reminder shown in the status message. Reminder list shown in Main window.

b. Test case: `add_reminder`
   Expected: No reminder is added. Error details and correct format shown in the status message.

c. Other incorrect command format to try: `add_reminder reminder`, `add_reminder rd/error rt/20/9/2019 13:00, 21/9/2019 15:00`

d. Reminder: Reminders with duplicate `description` cannot be created.

Deleting a reminder

1. Delete a reminder and list all reminders out without the deleted reminder.

   a. Test case: `delete_reminder 1`
      Expected: The first reminder in the reminder list was deleted. Details of the deleted reminder shown in the status message. Reminder list shown in Main window.

   b. Test case: `delete_reminder`
      Expected: No reminder is deleted. Error details and correct format shown in the status message.

   c. Other incorrect command format to try: `delete_reminder 0` `delete_reminder 1 rd/`

   d. Reminder: Make sure the index of the reminder is not larger than the total number of reminder in reminder list.

Finding a reminder

1. Find reminders by their Description or its Reminder Time.

   a. Test case: `find_reminder_by_description cs2100`
      Expected: All reminders that description contains "cs2100" (regardless case of the letters) were shown in the MainWindow. Number of reminders found were stated in the status message.

   b. Test case: `find_reminder_by_date 20/10/2019`
      Expected: All reminders that Reminder Time contains "20/10/2019" were shown in the MainWindow. Number of reminder found were stated in the status message.

   c. Test case: `find_reminder_by_description`
      Expected: Error details and correct format shown in the status message.

   d. Other incorrect command format to try: `find_reminder_by_date`, `find_reminder_by_date 13/10/2019 15/10/2019`

   e. Reminder: Reminder can be found by multiple numbers of Description but only one Date.

Listing all reminders

1. List out all reminders.

   a. Test case: `list_reminder`
      Expected: All reminders were listed in Main Window.

   b. Test case: `list reminders`

Expected: Error details shown in the status message.

Tab Change

1. Change Tab.
    a. Test case: `change_tab tab/earnings`
    Expected: All earnings were listed in Main Window.
    b. Test case: `changetab tasks`
    Expected: Error details shown in the status message.

Listing by class

1. List out all students in the class.
    a. Test case: `list_class CS2030`
    Expected: All students in class CS2030 are shown.
    b. Test case: `list_class CS2030 CS2040`
    Expected: All students in class CS2030 and CS2040 are shown.
    c. Test case: `list_class RANDOM STRING`
    Expected: No students are shown.

Mass Assigning Class

1. Assigns the class to all students indicated.
    a. Test case: `assign_class 1,2,3 c/CS2030`
    Expected: Students at index 1,2 and 3 are assigned to class CS2030. (assuming the list size is at least 3)
    b. Test case: `assign_class 1,2,3 c/CS2030`
    Expected: Nobody is assigned to class CS2030. (assuming the list size is less than 3)
    c. Test case: `assign_class 1,2,100 c/CS2030`
    Expected: Nobody is assigned to class CS2030. (assuming the list size is 5)

Mass Marking Attendance

1. Increases the attendance of indicated students by one.
    a. Test case: `mark_attendance 1,2,3`
    Expected: Students at index 1,2 and 3 have their attendance increased (assuming the list size is at least 3)
    b. Test case: `mark_attendance 1,2,3`
    Expected: Nobody has their attendance increased (assuming the list size is less than 3)
    c. Test case: `mark_attendance 1,2,100`
    Expected: Nobody has their attendance increased (assuming the list size is 5)

Mass Marking Participation

1. Increases the participation of indicated students by one.

    a. Test case: `mark_participation 1,2,3`
    Expected: Students at index 1,2 and 3 have their participation increased (assuming the list size is at least 3)

    b. Test case: `mark_participation 1,2,3`
    Expected: Nobody has their participation increased (assuming the list size is less than 3)

    c. Test case: `mark_participation 1,2,100`
    Expected: Nobody has their participation increased (assuming the list size is 5)

Setting picture

1. Sets the profile picture of the indicated student

    a. Test case: `set_pic 1 pic/test.jpg`
    Expected: Student at index 1 is given test.jpg as his profile picture. (assuming test.jpg is in the same directory as TutorAid)

    b. Test case: `set_pic 1 pic/test.docx`
    Expected: Error occurs. Only .jpg/.png/.gif/.bmp is supported.

    c. Test case: `set_pic 1 pic/test.jpg`
    Expected: Error occurs. TutorAid can't find the picture. (assuming test.jpg is not in the same directory as TutorAid)

    d. Test case: `set_pic 100 pic/test.jpg`
    Expected: Error occurs. Index is invalid. (assuming a list size of 10 and test.jpg is in the same directory as TutorAid)

Unknown wrong command

1. Learns a wrong command the user typed in as a basic command.

    a. Test case: `gibberish`
    Expected: TutorAid prompts the user for the command word they intended to execute. (assuming gibberish is not yet learned)

    b. Test case: `ad n/Caesar c/CS1000`
    Expected: TutorAid prompts the user for the command word they intended to execute. (assuming ad is not yet learned)

        i. Test case: `cancel`
        Expected: The unknown command is discarded and normal operations can resume.

        ii. Test case: `gibberish`
        Expected: TutorAid prompts the user for the command word they intended to execute. (assuming gibberish is not yet learned)

        iii. Test case: `add n/Caesar c/CS2100`
        Expected: TutorAid can't learn full commands. Prompts the user which command word

they want to map the last unknown command to again.

iv. Test case: `add`
Expected: TutorAid learns the last unknown command as `add`

Delete Custom Command

1. Deletes a wrong custom command previously learnt.

   a. Test case: `deleteCustomCommand VALIDCUSTOMCOMMAND`
   Expected: The custom command is deleted.

   b. Test case: `deleteCustomCommand INVALIDCUSTOMCOMMAND`
   Expected: Error. Invalid command, TutorAid can't find such a command to delete.

   c. Test case: `deleteCustomCommand BASICCOMMAND`
   Expected: Error. You can't delete a basic built-in command.

Appendix G: 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	track all the information about my students	remember key info about them easily when I need it
* * *	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

Priority	As a …	I want to …	So that I can…
* * *	tutor	check the information of my classes	know more about my students' situation
* *	tutor	see who's not coming for class	check up on them
* *	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

Adding a task

1. Add a task and list all tasks out with the added task.

    a. Test case: `add_task c/CS2103T mark/Y tt/13/09/2019 13:00, 13/09/2019 16:00`
    Expected: A marked task with Class Id "CS2103T" and Task Time "13/09/2019 13:00 to 16:00" was added to the task list. Details of the added task shown in the status message. Task list shown in Main window.
    Reminder with Description "CS2103T" and Time "13/09/2019 13:00, 20/09/2019 16:00" will be created as well.
    A Task will be shown on that date in the Calendar view as well.

    b. Test case: `add_task`
    Expected: No task is added. Error details and correct format shown in the status message.

    c. Other incorrect command format to try: `add_task task`, `add_task c/ mark/ tt/`

Editing a task

1. Edit a task and list all tasks out with the edited task.

    a. Test case: `edit_task 1 c/CS2103`
    Expected: The first task in the task list's Class Id changed to CS2103. Details of the edited task shown in the status message. Task list shown in Main window.

    b. Test case: `edit_task c/cs2103`
    Expected: No task is changed. Error details and correct format shown in the status message.

    c. Other incorrect command format to try: `edit_task` `edit_task 1 c/`

d. Reminder: Make sure the index of the task is not larger than the total number of tasks in task list.