Sebastian Lie - Project Portfolio

PROJECT: Teacher's Notebook

Overview

This project portfolio details my key contributions to Project Teacher's Notebook.

Teacher's Notebook is a desktop application designed by my group mates and I for a software engineering module in National University of Singapore (NUS). It's target audience is secondary or primary school teachers in Singapore. aims to improve the efficiency of administrative processes in medical institutes. The project was created under the constraint of the fact that the user only interacts with the application through a Command Line Interface (CLI). It is written in Java, and has about 10 kLoC.

The main features of the project are:

- The ability to manage students and their
- The ability to
- The ability to
- The ability to

Summary of Contributions

- **Major Enhancement**: added the ability to upload and display photos of students
  - What it does
  - Justification
- **Major Enhancement**: added the autocomplete feature to the gui
  - What it does
- **Minor Enhancement**: added the ability to scroll through previously entered commands with the up and down arrow keys
- Code Contributed [Functional Code] [RepoSense]
  - **Other contributions**:
    - Project management:
      - Managed releases `v1.3` - `v1.5rc` (3 releases) on GitHub
      - Fixed issues brought up by peers. (Pull requests #61, Pull requests #85)
    - Enhancements to existing features:
      - Added 2 extra panels to the GUI, and added display picture to the student card (Pull

requests #61, Pull requests #85)

- improved test coverage from 25 to 33 %, PR #189, #190, #191: 46-48% (Pull requests #190, Pull requests #189)

- Documentation:
  - Did cosmetic tweaks to existing contents of the User Guide: #14
- Community:
  - PRs reviewed (with non-trivial review comments): #12, #32, #19, #42
  - Reported bugs and suggestions for other teams in the class (examples: 1, 2, 3)

- Other contributions:
- Added Test coverage: improved test coverage from 25 to 33 %, PR #189, #190, #191: 46-48%

Contributions to User Guide

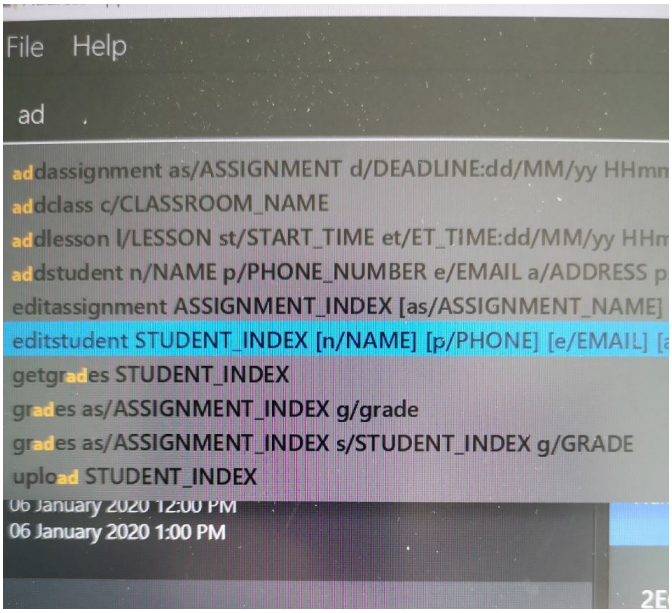
Given below are sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

User-Friendly Features

The features here do not necessarily contribute directly to a teacher's everyday job, but improve the usability of Teacher's Notebook and make using the application a more forgiving, painless and seamless experience.

Autocomplete

The Autocomplete feature allows a user to toggle through suggestions that guess at what command the user wants to type in.
Upon typing any letters that resemble commands, a popup menu will appear as shown below:



The user can then choose the first option of the menu using CTRL, use SHIFT + UP and SHIFT + DOWN to cycle through menu options,
or use ESC to close the pop window. If the popup menu is closed, no autocomplete features will be available.

Usage:

`SHIFT + Arrow DOWN` and `SHIFT + Arrow UP` Keys to toggle between autocomplete options
`CTRL` Key to choose the first autocomplete suggestions
`ESC` Key to close the autocomplete suggestions

History

The History feature allows a user to toggle through their previously entered commands,
regardless of whether the user command was successful. As this may clash with autocomplete suggestion toggling, we recommend closing
autocomplete suggestions before using the History feature.

Usage:

`Arrow UP and DOWN` Key to toggle through previous commands

Undo/Redo: Undo/Redo

The undo/redo feature allows a user to undo any command, and there is no limit on the number of actions that can be undone or redone.
After the undo/redo command the application will be in the state before/after the action was made.
Undo/redo cannot, however, undo actions made during previous activation of Teacher's Notebook.
This is to say, once the application is closed, all actions done cannot be undone.

Usage:

Enter `undo` or `redo`

Help

Triggers a popup window with a link to the user guide.

Usage:

Enter `help`

Clear

Clears all data from the notebook. If triggered erroneously, the clear command can be undone to reclaim all data.

Usage:

Enter `clear` to clear all data

Exit

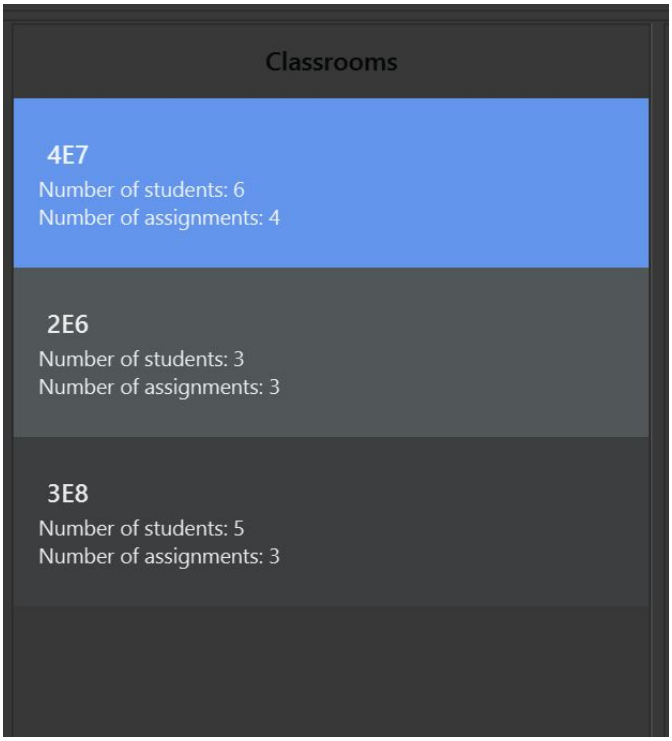
Exits the application and stops all teacher's notebook processes.

Usage:

Enter `exit`

Classes

The user can view his or her classes in the middle panel of the GUI of Teacher's Notebook as shown below.



The class colored in blue shows the class that the user is currently viewing. Viewing a class enables a user to see the full profiles of the students,
As well as full information about assignments. Changing classes will allow the user to view the students and assignments of another class. The number of students, and the number of assignments are displayed for each class

Adding: addclass

Adds a new class to the list of existing classes. The name of the new class must not be empty, and it must be different from all the current class names.

Format: `addclass c/CLASS_NAME`

Examples:

- `addclass c/4E7`
- `addclass c/3E8`

Deleting: deleteclass

Allows a user to delete the selected class, if the user no longer needs the class or entered its name wrongly.
Classroom must be in the current list of classrooms, and class name cannot be empty.

Format: `deleteclassroom c/CLASS_NAME`

Examples:

- `deleteclassroom c/4E7`
- `deleteclassroom c/3E8`

Setting: setclass

Allows a user to delete the selected class, if the user no longer needs the class or entered its name wrongly.
Classroom must be in the current list of classrooms, and class name cannot be empty.

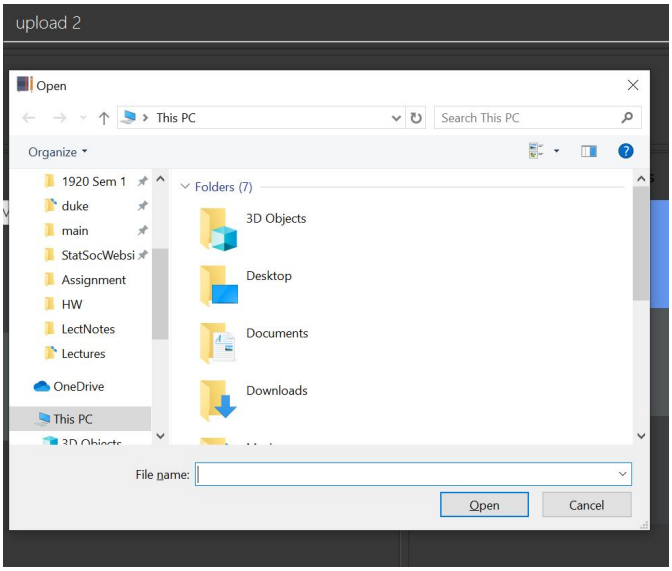
Format: `setclass c/CLASS_NAME`

Examples:

- `setclass c/4E7`
- `setclass c/3E8`

Uploading: upload

Allows a user to upload a PNG/JPG file to set the display picture of the student at the specified index in the student list of the current class.
Upon entering the command, a window will popup as shown below:



And the user will be prompted to select a PNG/JPG file from their computer to set as the new display picture of the student.
The user may cancel the upload operation by clicking on cancel in the window that pops up.

Format: `upload STUDENT_INDEX`

Example:

- `upload 3`

NOTE	The index provided must be valid and the directory of the image uploaded is assumed to not change. If the image is moved or deleted after it is set as the display picture, the display picture will be empty when the user starts up Teacher's Notebook.

Resetting display picture

Resets the display picture of the student to the default.

Format: resetdisplaypic

Example:

- `resetdisplaypic`

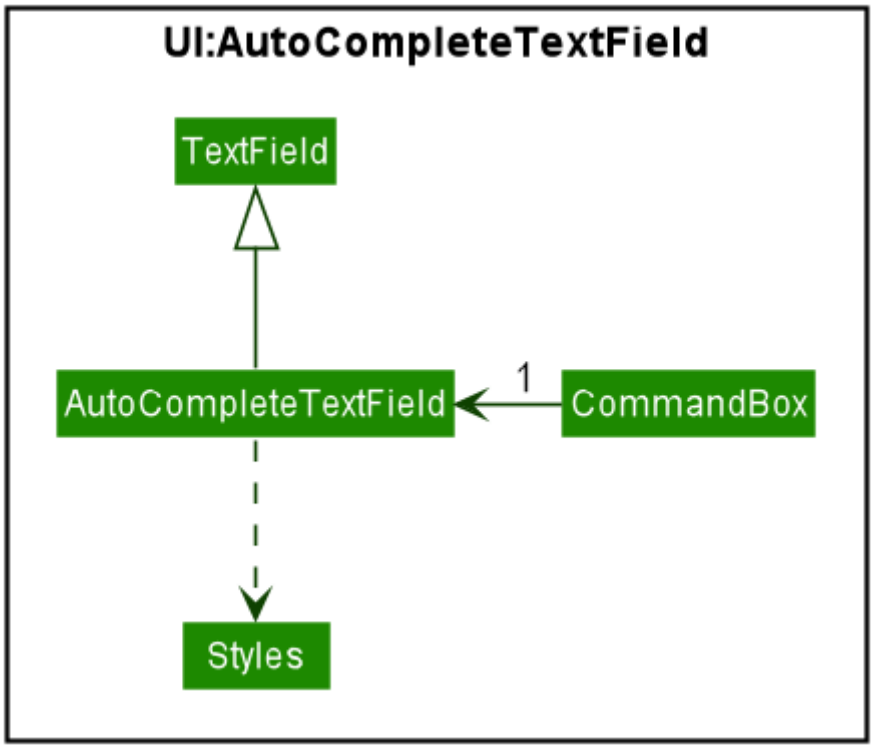
Contributions to the Developer Guide

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.

Autocomplete feature

Current Implementation

The AutoComplete feature is facilitated by 2 classes: the `AutoCompleteTextField`, and the class `Styles`. Both are represented in the class diagram below.



The `AutoCompleteTextField` adds a `ChangeListener` to textProperty() that notifies `AutoCompleteTextField` whenever the user inputs new text, i.e when the text entered changes.

Step 1. User triggers the listener when user enters text.

Step 2. From the text entered, `AutoCompleteTextField` attempts suggesting existing commands. If input matches any existing commands, it proceeds to step 3. Otherwise, it does nothing and waits user input.

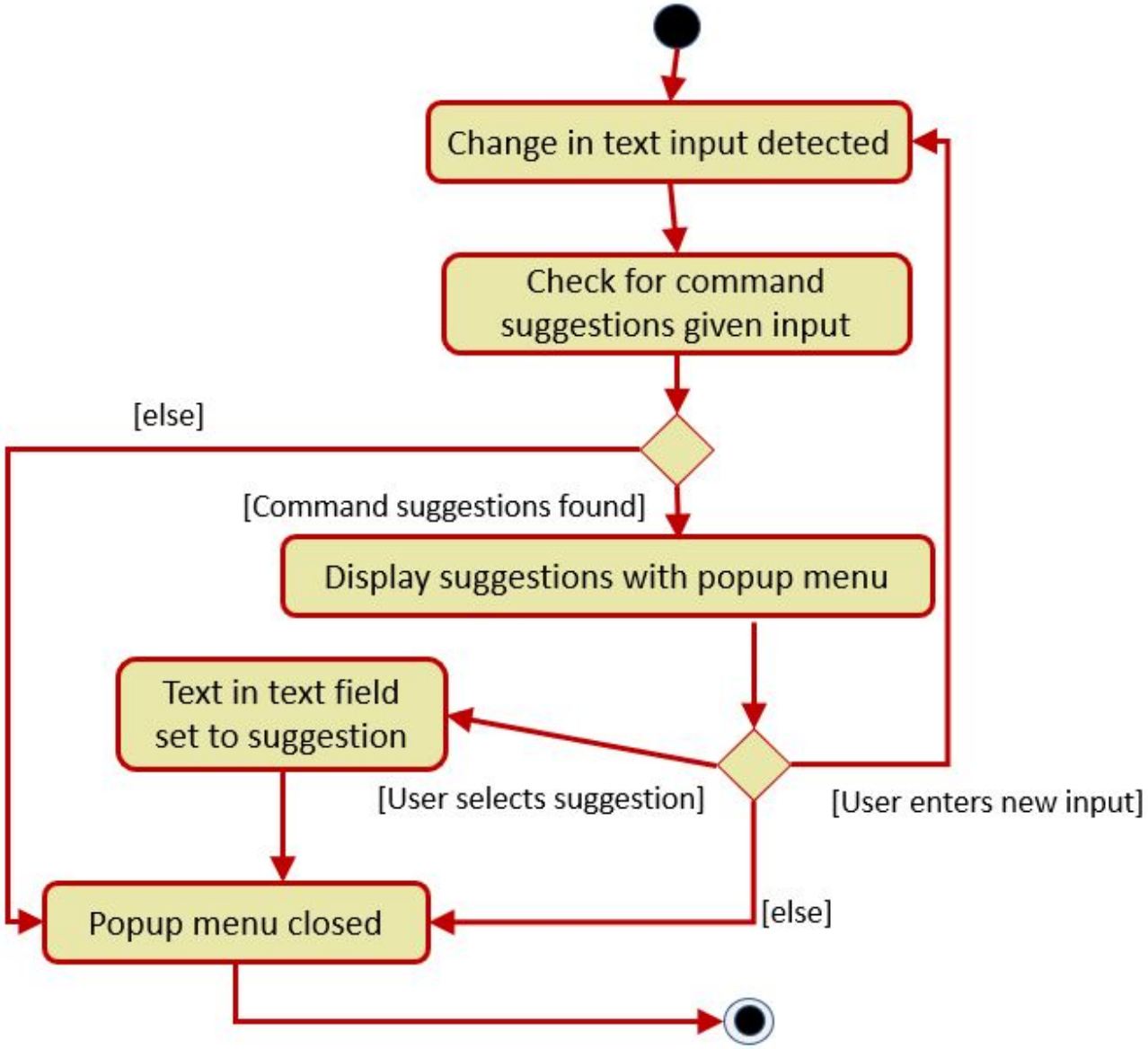
Step 3. `AutoCompleteTextField` calls `Styles#buildTextFlow`to highlight the portion of each command suggestion where the user's input matches the suggestion.

Step 4. `AutoCompleteTextField` then displays a popup menu using the javafx class `ContextMenu`.

Step 5. If the user selects a suggestion, the textfield is set to that suggestion, and the popup menu is closed. If the user enters more text, `AutoCompleteTextField` returns to step 1.

Step 6. Otherwise, the popup menu is closed.

This sequence of steps is illustrated below with an activity diagram.



Upload feature

Current Implementation

The uploading of a student's picture is facilitated by UploadPictureCommandParser, UploadPictureCommand, ModelManager, FileChooser and DisplayPicture, while the displaying of the student's picture is facilitated by MainWindow and StudentCard.
`UploadPictureCommandParser` implements `Parser`. `UploadPictureCommand` extends `Command`, and represents the logic that will be executed once the user activates the upload command and chooses a file.
The `UploadPictureCommand` command communicates with `ModelManager` to replace the old `Student` object with a new `Student` object that has its display picture field changed.

This change is then reflected in the UI when `MainWindow` loads the `StudentCard` for the new student, which will load the image and display it in the UI.

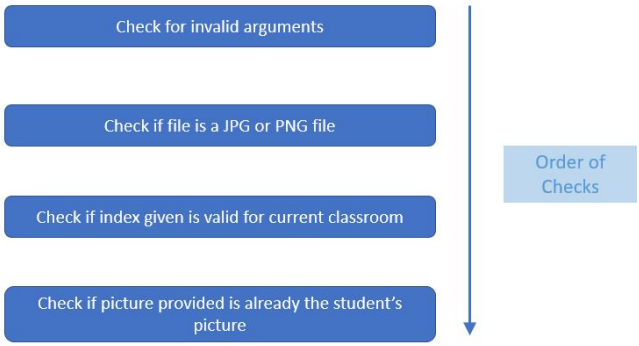
The main operation implemented is `UploadPictureCommand#execute`.

Step 1. The user wants to upload a display picture for their student. User identifies the student's index number and enters the command `upload INDEX`.

Step 2. `MainWindow` detects the `upload` command word and uses `FileChooser` to open a window for the user to select a picture from their computer. `MainWindow` then saves the path of the selected file as a string and adds a prefix "f/" to the input arguments. The path of the file is also appended to the end of the input string before the input arguments are passed to `NotebookParser`

Step 3. `NotebookParser` parses the input and detects the `upload` command word, and calls `UploadPictureCommandParser`. This in turn checks user's input for errors before calling `UploadPictureCommand`, which checks whether the file is a PNG or JPG file.

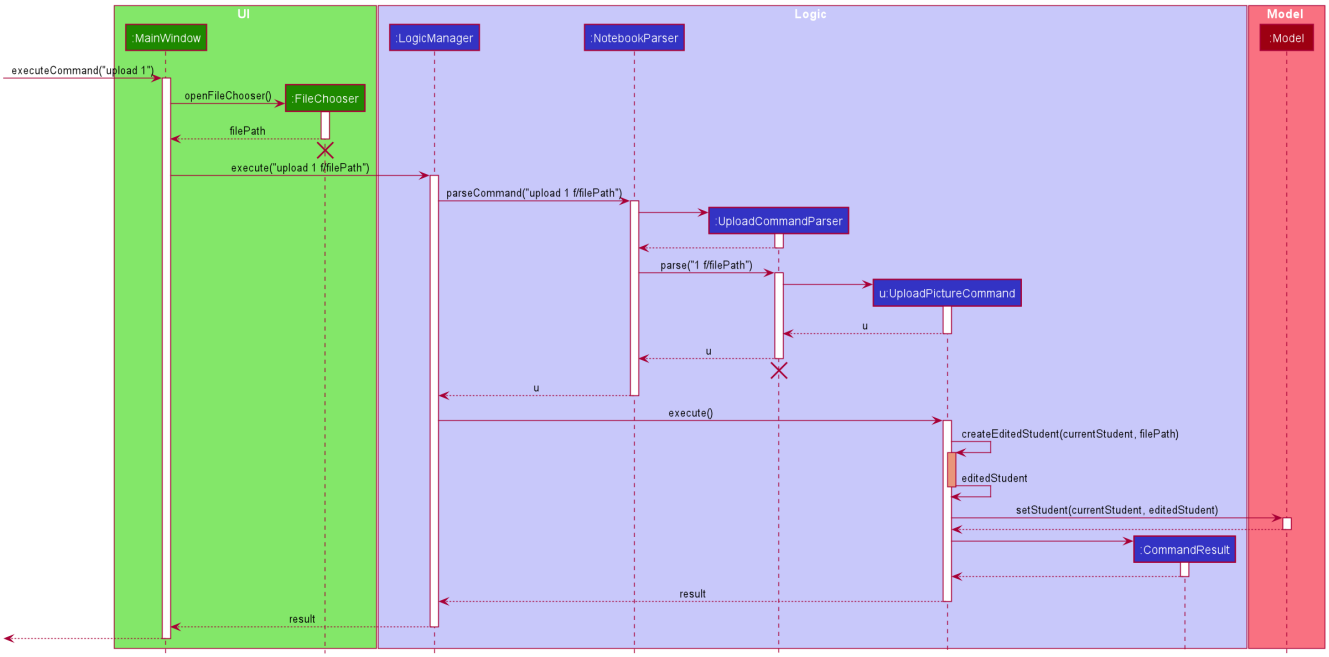
Step 4. `LogicManager` executes the `UploadPictureCommand`, which checks the validity of the index provided, as well as whether the picture selected is different from the picture already displayed. A visual representation of the order of checks for the input arguments are displayed in the diagram below.



Step 5. After these checks, the `createEditedStudent` method in `UploadPictureCommand` is called to create a new `Student` object, editedStudent, which has its display filepath changed to the file the user chose.

Step 6. `UploadPictureCommand` calls Model to replace the current student with the editedStudent. The command result is then passed back all the way to `MainWindow` which refreshes the GUI, and displays the new picture for the chosen student.

The following diagram illustrates the process above.



Design Considerations

Alternative implementations: