Project Portfolio for StudyBuddyPro

Done By: Jasmine Yeo Jia Min

This portfolio is meant to be a well-documented data point of my Software Engineering experience on the project. It showcases my contributions to the project and my documentation skills.

Overview

StudyBuddyPro is a desktop Command Line Interface (CLI) application intended for Computing School students of National University of Singapore (NUS). It aims to lessen target users' revision time consumption without compromising the quality of studies with a centralized platform.

StudyBuddyPro provides three main features: Flashcard, Notes and Cheatsheet. These features are used to revise using flashcards, take notes and generate cheatsheets respectively. Additionally, StudyBuddyPro is equipped with a Graphic User Interface (GUI).

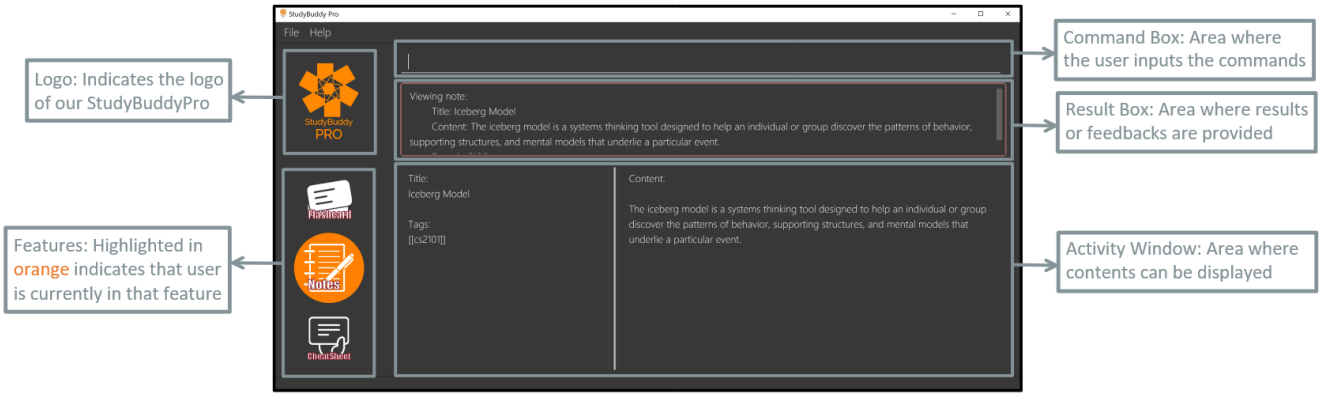

Figure 1. StudyBuddyPro's GUI with explanations

The above image shows how StudyBuddyPro looks like with the descriptions of each functional areas and icons.

Syntax

Please do take a look at the syntax notations below as they are used throughout the document.

`syntax 1` — For commands or class objects

syntax 2 — For figures, tables, functionality or variables

Callouts Signs

Please do refer to the signs below as they are used throughout the document.

WARNING	Indicates information that are to be adhere as potential problems may be encountered if you are not careful.
IMPORTANT	Indicates information that are crucial to understand so that you will be able to follow the flow of the document.
NOTE	Indicates information that are note-worthy. Do read them for more information and better understandings.

Summary of contributions

The sections below are a summary of my contributions to the project. It highlights my major contributions and states my minor, code and other contributions.

Major Enhancement 1: Automated the Generation of Cheatsheets

This enhancement allows users to automatically generate cheatsheets according to the *tags* that they specify. From this, it improves the product significantly as users can sift out information to be included in a specific cheatsheet. Thus, it reduces the users' time taken in collating cheatsheets.

This enhancement's implementation makes use of the other features in the StudyBuddyPro to create the contents in the cheatsheets. As such, users can make full use of this auto-generation enhancement by using *tagging* functionality in those features.

Major Enhancement 2: Customization of Generated Cheatsheets

This enhancement allows users to remove contents in the generated cheatsheets by specifying its content indexes or tags. From this, it improves the cheatsheets' usability as users can determine which contents to be included or excluded in the cheatsheets.

This enhancement ensures all contents are placed under the *tags* specified even after users' personalization. Hence, users can be assured that the cheatsheets contain no irrelevant contents.

Minor Enhancement 1: Switching between the Features

This enhancement allows users to toggle amongst the features in the StudyBuddyPro. Also, it allows *global* commands, like `exit` or `filterall` to be used in any features.

Minor Enhancement 2: Clearing Specific Features's Data

This enhancement allows users to clear specific feature's data in the StudyBuddyPro. As such, users do not need to clear the entire data if desired.

Code Contribution

Please do refer to the link here my functional codes and here for my test codes.

Other Contributions:

- Designed the GUI for `view` in the Cheatsheet feature (#166, #245)
- Did cosmetic tweaks to the existing contents of the Developer Guide (#74, #177, #192, #203)
- Reported bugs and suggestions for other teams in the class

Contributions to the User Guide (UG)

The sections below are my contributions to the UG of StudyBuddyPro. They showcase my ability to write documentation that targets the end-users.

> **IMPORTANT** All commands in the sections assume that the user is in the *cheatsheet* mode.

Creating a Cheatsheet: `add`

Adds a cheatsheet from user input title <TITLE> and content <CONTENT>. Flashcards and notes in StudyBuddyPro that have the specified tag will be used as contents in the cheatsheet.

```
Format: add t/TITLE [tag/TAG]...
```

Example usage:

```
add t/CS2100 Midterm CheatSheet tag/cs2100midterm
```

> **IMPORTANT** Assuming that there is a flashcard object with the tag "cs2100midterm"

Expected output:

```
New cheatsheet added:
Title: CS2100 Midterm CheatSheet
Tags: [cs2100midterm]
1 content(s) have been successfully generated from the other modes.
```

Editing a Cheatsheet: `edit`

Edits cheatsheet's title, tag, content by a specified <CHEATSHEET_INDEX>. At least one of the optional fields must be specified to edit.

```
Format: edit (index) [t/TITLE] [tag/TAG]...
```

IMPORTANT	• Only `t/TITLE` optional field will overwrite its field. • All other optional fields will remove itsn existing content(s). • Any invalid `c/CONTENT_INDEX` or `tag/TAG` will be **ignored**.

Example CheatSheet of index 8:

```
Title: cs2100 cheatsheet
Tags: [cs2100finals][formula]
Contents: [ 1. Question: What is 110 Binary in its Decimal Form?; Answer: 6 ]
          [ 2. 10 + 10 = 20]
```

Example usage:

```
edit 8 t/cs2100 final cheatsheet tag/formula
```

Expected output:

NOTE	The actual implementation does not show the contents in the feedback box. Please do use `view` command to view them!

```
Edited Cheatsheet:
Title: cs2100 final cheatsheet
Tags: [cs2100finals]
Contents: [ 1. Question: What is 110 Binary in its Decimal Form?; Answer: 6 ]
```

Viewing Cheatsheets: `view`

Views a cheatsheet by the specified index.

```
Format: view (index)
```

Example CheatSheet of index 1:

```
Title: cs2100 cheatsheet
Tags: [cs2100finals][important]
Contents: [ 1. Question: What is 110 Binary in its Decimal Form?; Answer: 6 ]
          [ 2. 10 + 10 = 20]
```

Example usage:

```
view 1
```

Expected output:

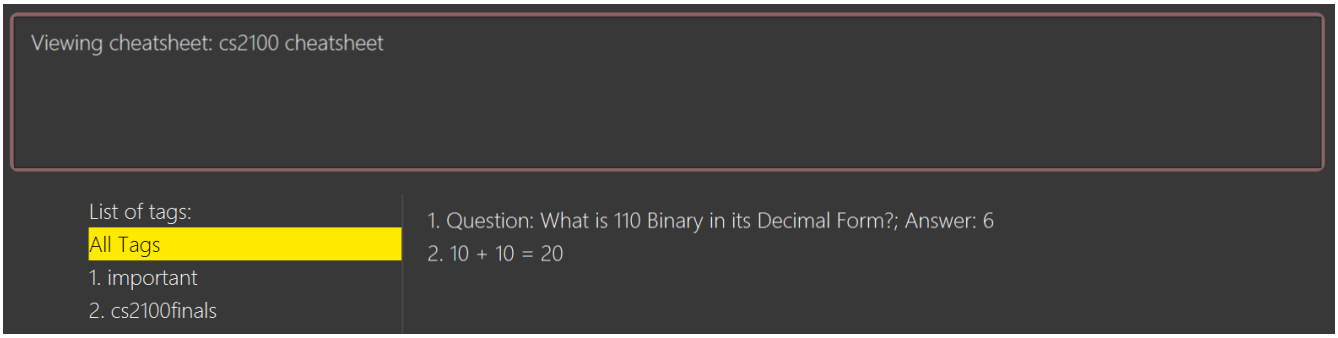

Figure 2. Screenshot for `view 1` expected output

The above screenshot shows the cheatsheet with index 1 in the GUI of StudyBuddyPro.

Viewing Cheatsheets of Specific Tag: `show`

Views a cheatsheet's content for a specified tag. User must be in a `view` command before using `show` command.

```
Format: show (index)
```

NOTE Assuming user is in the `view 1` command from above example in `view` command.

Example usage:

```
show 1
```

Expected output:

NOTE Currently, cheatsheets only allow contents that match all the specified tags. Hence, `show` command will only show color toggling at the tags segment at the moment.

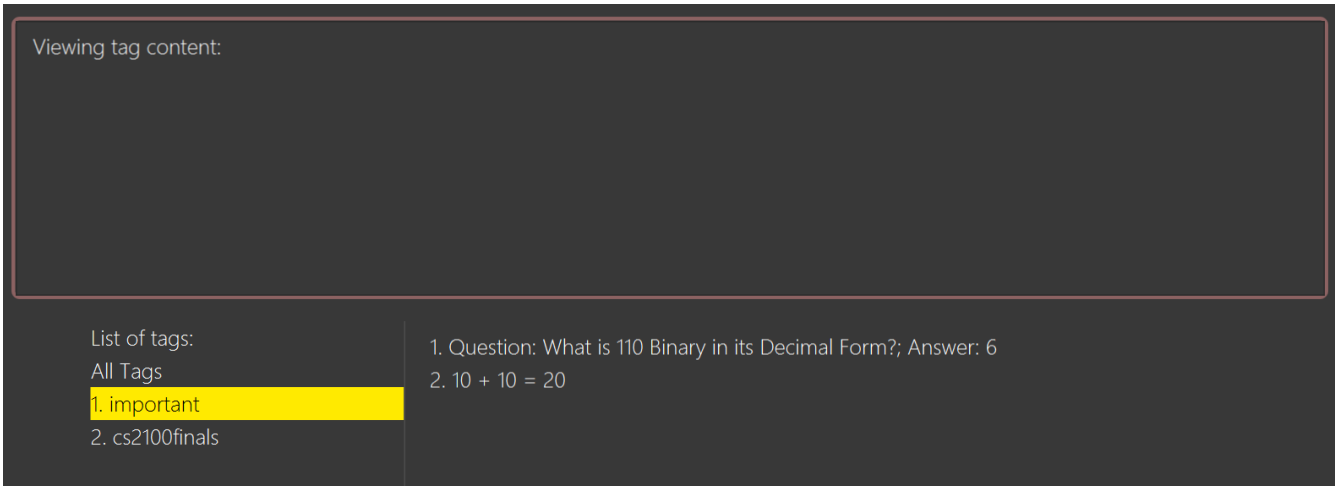

Figure 3. Screenshot for `show 1` expected output

The above screenshot shows the cheatsheet's content with tag index 1 in the GUI of StudyBuddyPro.

Listing Cheatsheets: `list`

Lists all cheatsheets found in StudyBuddyPro.

```
Format: list
```

Updating a Cheatsheet: `update` (Coming in v2.0)

Updates cheatsheet's contents by a specified <CHEATSHEET_INDEX>. Tags specified are added into the list of tags for the cheatsheet if it not already exist.

WARNING	This command may overwrite any customization of contents done prior to it as the cheatsheet's contents will be regenerated.

```
Format: update (index) [tag/TAG]...
```

Example CheatSheet of index 8:

```
Title: cs2100 cheatsheet
Tags: [cs2100finals]
Contents: [ 1. Question: What is 110 Binary in its Decimal Form?; Answer: 6 ]
```

Example usage 1:

```
update 8
```

Expected output:

NOTE	The actual implementation does not show the contents in the feedback box. Please do use `view` command to view them!

```
Updated Cheatsheet:
Title: cs2100 final cheatsheet
Tags: [cs2100finals]
Contents: [ 1. Question: What is 110 Binary in its Decimal Form?; Answer: 6 ]
          [ 2. Binary is in bits of 1 and 0.]
```

Example usage 2:

```
update 8 tag/formula
```

Expected output:

NOTE	The actual implementation does not show the contents in the feedback box. Please do use `view` command to view them!

```
Updated Cheatsheet:
Title: cs2100 final cheatsheet
Tags: [cs2100finals][formula]
Contents: [ 1. Question: What is 110 Binary in its Decimal Form?; Answer: 6 ]
          [ 2. 10 + 10 = 20]
```

Contributions to the Developer Guide (DG)

The sections below are my contributions to the DG of StudyBuddyPro. They showcase my ability to write technical documentation. Also, it portrays my technical depth of my contributions to StudyBuddyPro.

Customizable Auto-generated Cheatsheet Feature

IMPORTANT	All the operations assume the user is in the *cheatsheet* mode.

Implementation

This feature has a two-step implementation. The first step is to auto-generate cheatsheet, and the second step is to enable removal of contents in the generated cheatsheet.

Step 1: Auto-generation

The auto-generation mechanism is used in the `AddCheatSheetCommand` during creation of the cheatsheet. After creation, the cheatsheet is then stored in the `studyBuddyBook`.

It is involved in the following operations:

- `AddCheatSheetCommand#execute()` — Creates the cheatsheet
- `AddCheatSheetCommand#getRelevantContents()` — Gets all the contents from *flashcard* and *notes* according to the *tags* specified

The first operation is exposed in the `Model` interface as `Model#setCheatSheet()`.

The following *figure* shows a high-level view of how the auto-generation operation works.

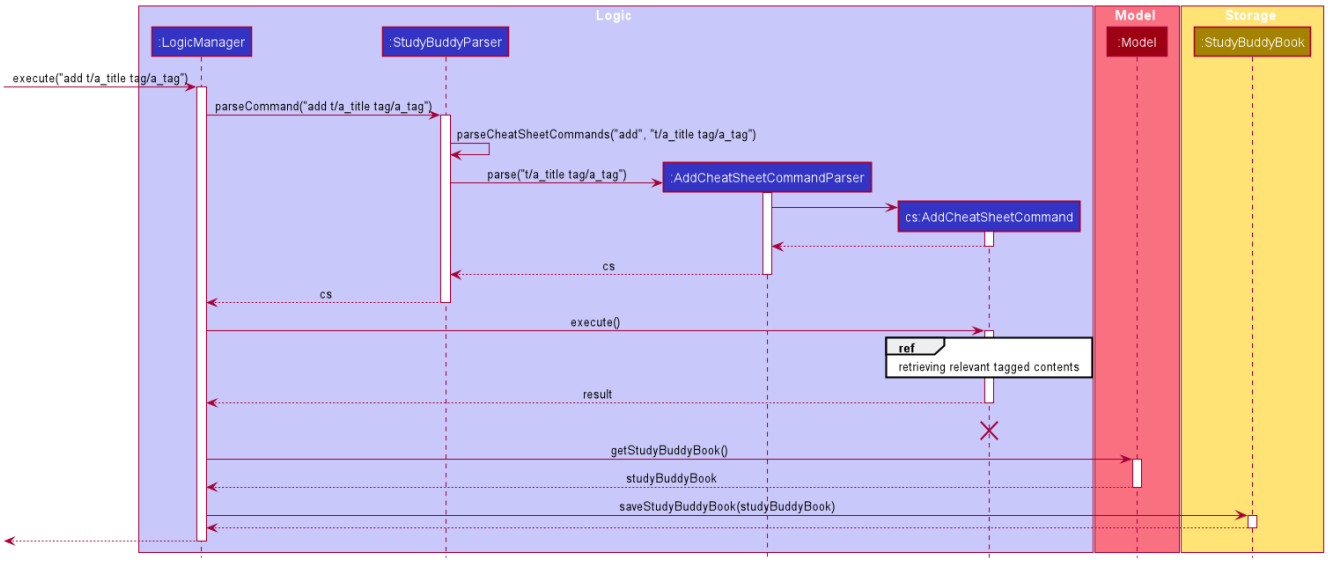

Figure 4. Sequence diagram to illustrate auto-generation operation

From the sequence diagram above, it portrays the relationships between the components to execute the creation mechanism. The *figure* below explains the details within the sequence diagram.

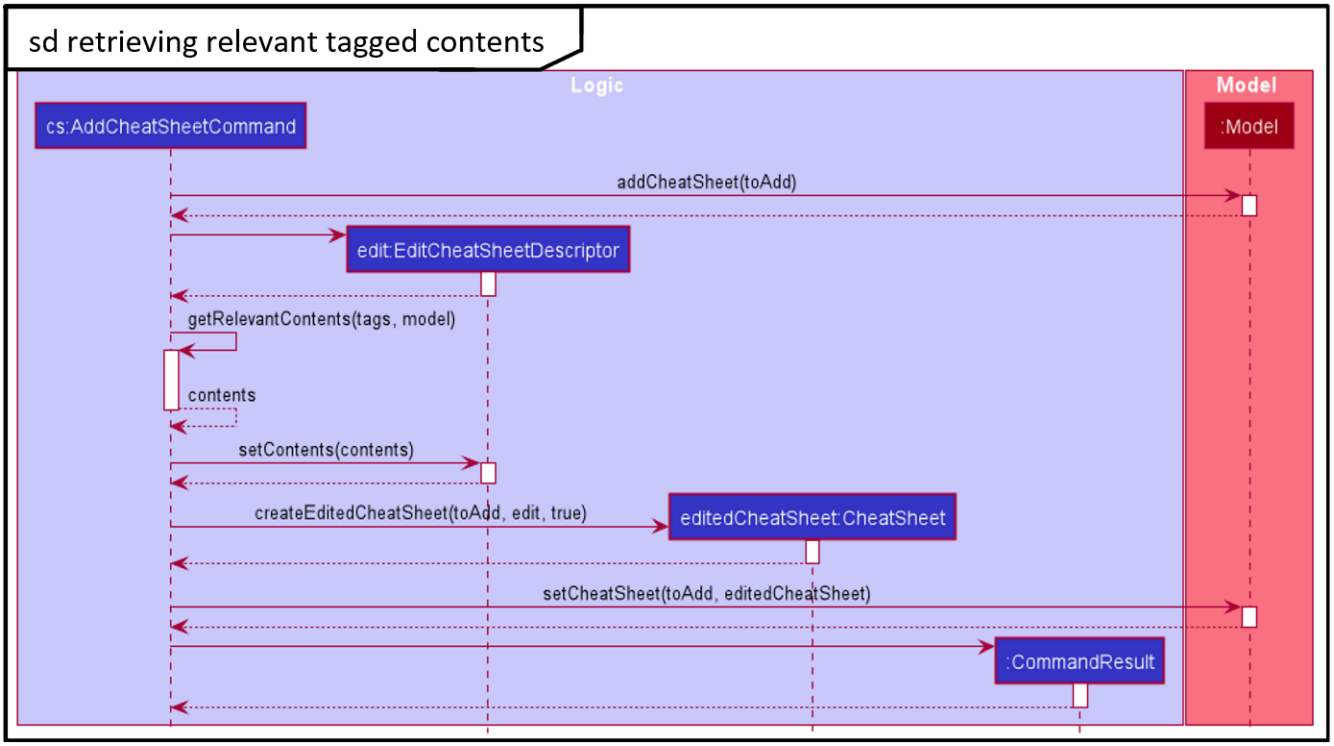

Figure 5. Detailed sequence diagram for retrieval of tagged contents

The above sequence diagram shows the complete action flow for the execution of methods. Here is a brief summary of the steps taken to create the cheatsheet:

1. A new *cheatsheet* object is created with the parsed *title* and *tags*.
2. Another new *cheatsheet* object is created with the relevant contents extracted according to the *tags* specified.
3. The first *cheatsheet* object is replaced with the second *cheatsheet* object while retaining its *title* and *tags*.

Different *cheatsheet* objects are created to ensure that the *cheatsheet* object itself is not modifiable.

Step 2: Customizing contents

The customization is based on the contents that the user wants to remove. The customization feature is used in the `EditCheatSheetCommand` during the editing of the cheatsheet. After the customization, the cheatsheet is then stored in the `studyBuddyBook`.

It is involved in the following operations:

- `EditCheatSheetCommand#execute()` — Edits the cheatsheet
- `EditCheatSheetCommand#updateContents()` — Retrieves the contents to be retained in the cheatsheet

The first operation is exposed in the `Model` interface as `Model#setCheatSheet()`.

The following *figure* shows the activity flow how the customization feature works using an example command called `edit 1 c/1 c/3 c/7`.

IMPORTANT	The index provided after `c/` indicates the content to be *removed*, not to be *retained*.

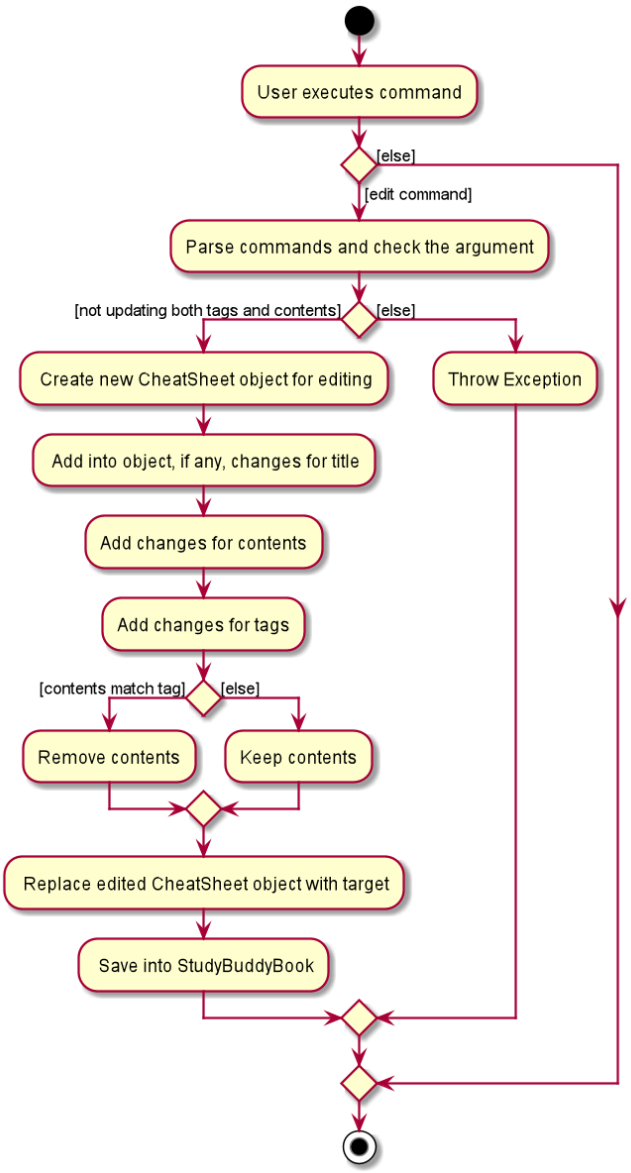

Figure 6. Activity diagram of the `edit` command for the removal of contents

The activity diagram above shows the general flow of customization of cheatsheet feature. The changes for *contents* have to come first before the changes for *tags*. This is to ensure that the *contents* are still relevant to the *tags* specified. An example is provided below.

Table 1. Example of tags and contents in a cheatsheet titled "An Example"

Tag	**Content**
tag1	content1
tag2	content2
tag2	content3

According to the above table, the system will be able to remove *tag2* first before *content2* if the order of removal is not followed. This may result in **potential errors** in the system as *content2* may not be found or the position of it is being replaced with another content.

Design Considerations

Aspect: How auto-generation is implemented

- **Alternative 1 (current choice):** Replacing the newly created cheatsheet with another cheatsheet object containing all the relevant contents
  - Pros: Retains the object originality and easier to implement.
  - Cons: Invoking the edit method to create a new cheatsheet object may be complicated and messy.
- **Alternative 2:** Reformat the way the `add` function works and abstract it such that it will be generalized.
  - Pros: Codes may be cleaner and easier to understand.
  - Cons: Harder to implement. More abstraction and modifications have to be done. Might change the format of the system.

Aspect: How customization of contents is implemented

- **Alternative 1 (current choice):** Places all contents that are not within the indexes specified by users into a new cheatsheet object and the targeted cheatsheet object with the respective changes.
  - Pros: Retains the object originality and easier to implement.
  - Cons: Large amount of contents may result in longer processing time as it loops to find all contents not removed. It is messier to comprehend.
- **Alternative 2:** Reformat the way the `edit` function works and abstract it such that it will be generalized.
  - Pros: Codes may be cleaner and easier to understand.
  - Cons: Harder to implement. More abstraction and modifications have to be done.