Qiu Jing Ying - Project Portfolio

Overview

CardiBuddy (CardiB) is a flash cards application that allows students to generate their own flash cards and test themselves through a simple Command Line Interface. It is specially designed for SoC students as a desktop application as they are generally comfortable with typing.

Summary of contributions

- **Major enhancement**: added **the ability to insert images into flashcards**
  - What it does: allows the user to add images from their local drive into flashcards.
  - Justification: This feature improves the product significantly because users are able to include images such as diagrams to illustrate more complex concepts (such as UML diagrams for CS2103).
  - Highlights: This enhancement affects the Model and Logic section of Cardi Buddy. It required an in-depth analysis of design alternatives. The implementation too was challenging as it required changes to existing commands such as the add command and the creation of a new type of object (Imagecard object). In addition, changes were made to the UI in the FlashcardListPanel to display images in the flashcard.
- **Minor enhancement**: added a filter command that allows the user to filter decks via tags.
- **Minor enhancement**: created three different answer types to questions in flashcards (T/F, MCQ and Short Answer) to make testing easier.
- **Minor enhancement**: added an insert image button that shows a popup window for the user to choose the .png and .jpg files they want to insert into their flashcards so that they do not have to key in their filepath.
- **Code contributed**: [Functional code]
- **Other contributions**:
  - Project management:
    - Morphed entire code base from AB3 to Cardi Buddy #84, #85, #96
    - Managed releases `v1.3.1` (1 release) on GitHub for the product to be tested in the Mock Practical Examination
  - Enhancements to existing features:
    - Re-designed UI (Pull requests #114, #199)
    - Improved on existing find command (renamed to search command) that allows the user to search for decks and cards that contains all of the keywords instead of any of the keywords.
    - Refactored commands from AB3 such as AddCommand to AddDeckCommand, AddFlashcardCommand and AddImageCardCommand as well as the DeleteCommand to

DeleteCardCommand and DeleteDeckCommand.

    - Wrote additional tests for the classes in Deck, Tag and Flashcard folders to increase coverage from 13% to 27% (Pull requests #200)

  - Documentation:
    - Did cosmetic tweaks to existing contents of the User Guide: #211
    - Wrote the whole README and designed the Ui mock-up: #54, #57
  - Community:
    - PRs reviewed (with non-trivial review comments): #119
    - Reported bugs and suggestions for other teams in the class (examples: 1, 2)
  - Tools:
    - Integrated automated hosting platform (Netlify) to the project (#60)

Contributions to the User Guide

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

Adding a normal card: `add q/QUESTION a/ANSWER`

Creates a new card to be added to the deck. Format: `add q/QUESTION a/ANSWER`

TIP These are the three types of `ANSWER` and their corresponding formats:
1. T/F — `T` or `F`
2. MCQ — `A)CHOICE_A B)CHOICE_B C)CHOICE_C` with the correct option being entered first, the order of the next two does not matter
3. Short Answer — `ANY_TEXT`

- The deck to be added in must be opened for you to add a card.

Examples:

- `open 2`
  `add q/How does one go about solving recursion problems? a/Wishful thinking`
  Adds a new card with a `Short Answer` type belonging to the opened deck indexed at 2.
- `open 1`
  `add q/Is defensive code desirable at all times? a/F`
  Adds a new card with a `T/F` answer type belonging to the opened deck indexed at 1.

Adding an image card: `add p/file:FILE_PATH_TO_IMAGE q/QUESTION a/ANSWER`

Creates a new image card to be added to the deck. Format: `add p/file:FILE_PATH_TO_IMAGE q/QUESTION a/ANSWER`

- The deck to be added in must be opened for you to add a card.

- The images can only be PNG or JPG files.
- The FILE_PATH_TO_IMAGE must end with .png or .jpg suffix.

Examples:

- `open 1`
  `add p/file:/Users/qiujingying/Documents/GitHub/cs2103/main/docs/images/ArchitectureDiagram.png q/What type of diagram is this? a/B)Architecture Diagram A)Sequence Diagram C)Object Diagram`
  Adds a new card with an image and `MCQ` answer type belonging to the opened deck indexed at 1. Note that the correct answer is `B` as it is the first choice entered.

TIP You can drag and drop the image you want to add into a terminal window to obtain its file path. File paths may differ across operating systems (Windows vs Mac).

Adding an image card via the Image button: `q/QUESTION a/ANSWER`

This is a shortcut for users who do not want to search for the file path. Format: `q/QUESTION a/ANSWER`

- The deck to be added in must be opened, before you click on the button, for you to add a card.
- Do note the differences in the command to be entered. There is no `add` or `p/IMAGE_FILEPATH` inputs that are required.

Examples:

- `open 1`
  click on button and choose image
  `q/What type of diagram is this? a/B)Architecture Diagram A)Sequence Diagram C)Object Diagram`
  Adds a new card with an image and `MCQ` answer type belonging to the opened deck indexed at 1.

Searching

Searching for a deck: `search deck`

Searches for the decks with titles that contain any of the given keywords or all of the keywords concatenated with the `&` symbol. Format: `search deck KEYWORD [&] [MORE_KEYWORDS]`

- If the search has a `&` symbol, only decks with the words concatenated before and after the symbol will be returned.
- The search is case insensitive. e.g cs2040 will match CS2040
- The order of the keywords does not matter. e.g. Science Module will match Module Science
- Only titles of the decks are searched.
- The keyword needs to match a word within the deck's title exactly . e.g. cs will not match with

cs2030

Examples:

- `search deck database`
  Displays decks with the word `database` in the titles.
- `search deck database & relational`
  Displays decks with both of the words `database` and `relational` in the titles.

Searching for a card: `search card`

Finds the cards with a question that contain any of the given keywords. Format: `search card KEYWORD [&] [MORE_KEYWORDS]`

- A deck needs to be opened for the `search card` command to work. The command will only search for cards in the opened deck.
- If the search has a `&` symbol, only cards with the words concatenated before and after the symbol, in the question, will be returned.
- The search is case insensitive. e.g programming will match Programming.
- The order of the keywords does not matter. e.g. Javascript programming language will match with programming language Javascript.
- The keyword needs to match a word within the question exactly. e.g. Java will not match with Javascript.

Examples:

- `search card principle`
  Displays cards with the word `principle` in the questions.
- `search card diagram & UML`
  Displays decks with both of the words `diagram` and `UML` in the questions.

Filtering

Filtering by tags: `filter`

Filters across all decks and only displays the decks with the specific tag. Format: `filter TAGNAME [&] [MORE_TAGNAMES]`

- If the search has a `&` symbol, only decks with the tags concatenated before and after the symbol will be returned.
- Filtering by tag is case insensitive. e.g hard will match Hard

Examples:

- `filter hard`
  Displays decks with the tag `hard`.
- `filter hard & coremodule`

Displays decks with both the tags `hard` and `coremodule`.

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.

Flashcards

Design

Users are able to add two different types of cards — cards with images and cards without. These cards have three types of answers — True/False, MCQ and short answers.

Model Component

The following classes can be found inside *cardibuddy/model/flashcard.*

The add feature revolves around 2 abstract classes: `Card` and `Answer`.

The `Card` class is extended by two card classes: `Flashcard` and `Imagecard`.

The `Answer` class is extended by three answer classes: `TfAnswer`, `McqAnswer` and `ShortAnswer`.

Logic Component

To add a card, a deck must first be opened. This can be checked from accessing the `LogicToUiManager` which stores the currently opened deck. Subsequently, The `Parser` classes will separate the relevant arguments from the user input and execute commands from the `Command` classes. These `Parser` and Command` classes are part of the **Logic** component of CardiBuddy, and can be found within the *cardibuddy/logic* package.

These commands allow the user to add the different types of flashcards and answers into a deck:

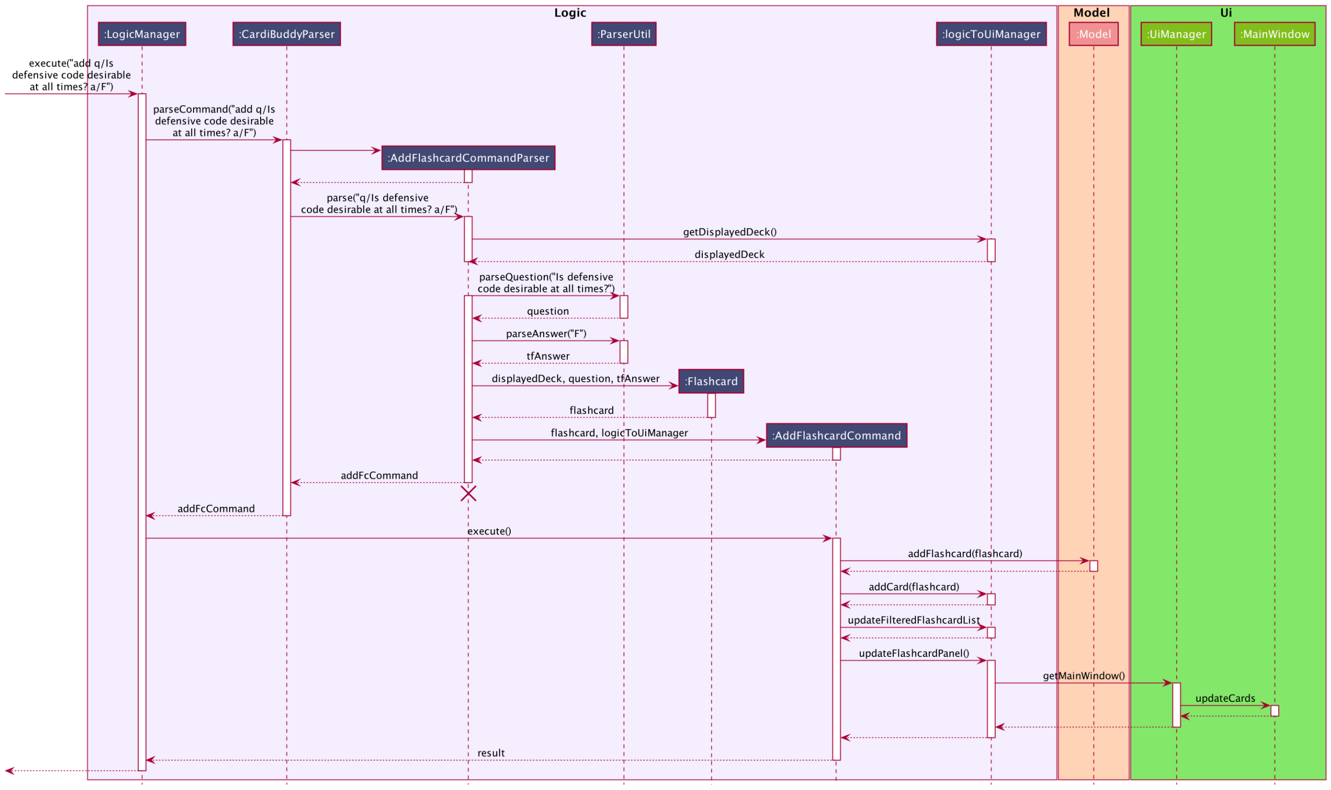
- The user will first open a deck.
- *Adding a Flashcard:* The user will enter `add` followed by `q/` with their question and `a/` with their answer.
- *Adding an Imagecard:* The user will enter `add` followed by `p/` with the filepath to the image, `q/` with their question and `a/` with their answer.
- *Adding a TfAnswer*: The user will enter either `T` or `F` for their answer after the `a/` prefix. Only capital, single-lettered answers are accepted and a `WrongTfException` will be thrown if the user enters `t`, `f`, `True` or `False`.
- *Adding an MCQAnswer*: The user will enter `A)CHOICE_A B)CHOICE_B C)CHOICE_C`, with the correct choice positioned first, for their answer after the `a/` prefix. In other words, if `C)CHOICE_C B)CHOICE_B A)CHOICE_A` is entered by the user, `C` will be taken as the correct answer. A `WrongMcqAnswerException` will be thrown if the user input does not have all three options in capital letters with parentheses.

Types of Cards

Flashcard

To add a `Flashcard` in an opened deck, the user will enter `add q/QUESTION a/ANSWER`. A sample command would be `add q/Is defensive code desirable at all times? a/F`.

The following sequence diagram shows how a `Flashcard` is created from the above command and displayed immediately in the flashcard panel to the user:



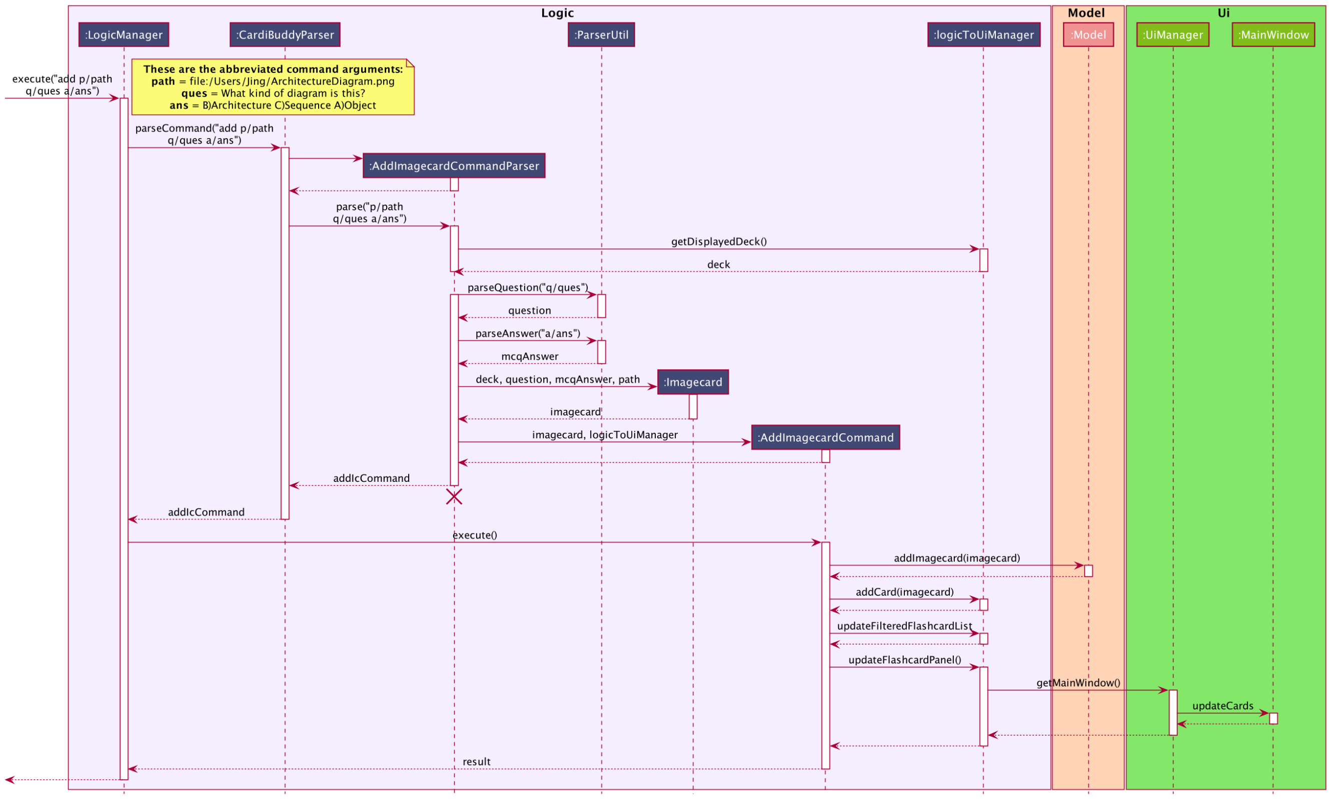
Logic
Model
Ui
:LogicManager
:CardiBuddyParser
:ParserUtil
:logicToUiManager
:Model
:UiManager
:MainWindow
execute("add q/Is defensive code desirable at all times? a/F")
parseCommand("add q/Is defensive code desirable at all times? a/F")
:AddFlashcardCommandParser
parse("q/Is defensive code desirable at all times? a/F")
getDisplayedDeck()
displayedDeck
parseQuestion("Is defensive code desirable at all times?")
question
parseAnswer("F")
tfAnswer
displayedDeck, question, tfAnswer
:Flashcard
flashcard
flashcard, logicToUiManager
:AddFlashcardCommand
addFcCommand
addFcCommand
execute()
addFlashcard(flashcard)
addCard(flashcard)
updateFilteredFlashcardList
updateFlashcardPanel()
getMainWindow()
updateCards
result

Imagecard

To add an `Imagecard` in an opened deck, the user will enter `add p/file:IMAGE_FILEPATH q/QUESTION a/ANSWER`. A sample command would be `add p/file:/Users/Jing/ArchitectureDiagram.png q/What kind of diagram is this? a/B)Architecture C)Sequence A)Object`.

When an `ImagecardCard` is displayed in the `FlashcardPanel`, the image will be retrieved via the stored `IMAGE_FILEPATH` from the user's computer. If the file path is invalid, the middle part of the card will be blank and an image will not be shown. More information regarding the implementation of the `Ui` can be found inside *cardibuddy/ui/ImagecardCard*.

The following sequence diagram shows how an `Imagecard` is created from the above command and displayed immediately in the flashcard panel to the user:


Logic
Model
Ui
:LogicManager
:CardiBuddyParser
:ParserUtil
:logicToUiManager
:Model
:UiManager
:MainWindow
These are the abbreviated command arguments:
path = file:/Users/Jing/ArchitectureDiagram.png
ques = What kind of diagram is this?
ans = B)Architecture C)Sequence A)Object
execute("add p/path q/ques a/ans")
parseCommand("add p/path q/ques a/ans")
:AddImagecardCommandParser
parse("p/path q/ques a/ans")
getDisplayedDeck()
deck
parseQuestion("q/ques")
question
parseAnswer("a/ans")
mcqAnswer
deck, question, mcqAnswer, path
:Imagecard
imagecard
imagecard, logicToUiManager
:AddImagecardCommand
addIcCommand
addIcCommand
execute()
addImagecard(imagecard)
addCard(imagecard)
updateFilteredFlashcardList
updateFlashcardPanel()
getMainWindow()
updateCards
result

It is largely similar to the sequence diagram for the creation of a `Flashcard` but with an extra `IMAGE_FILEPATH` argument.

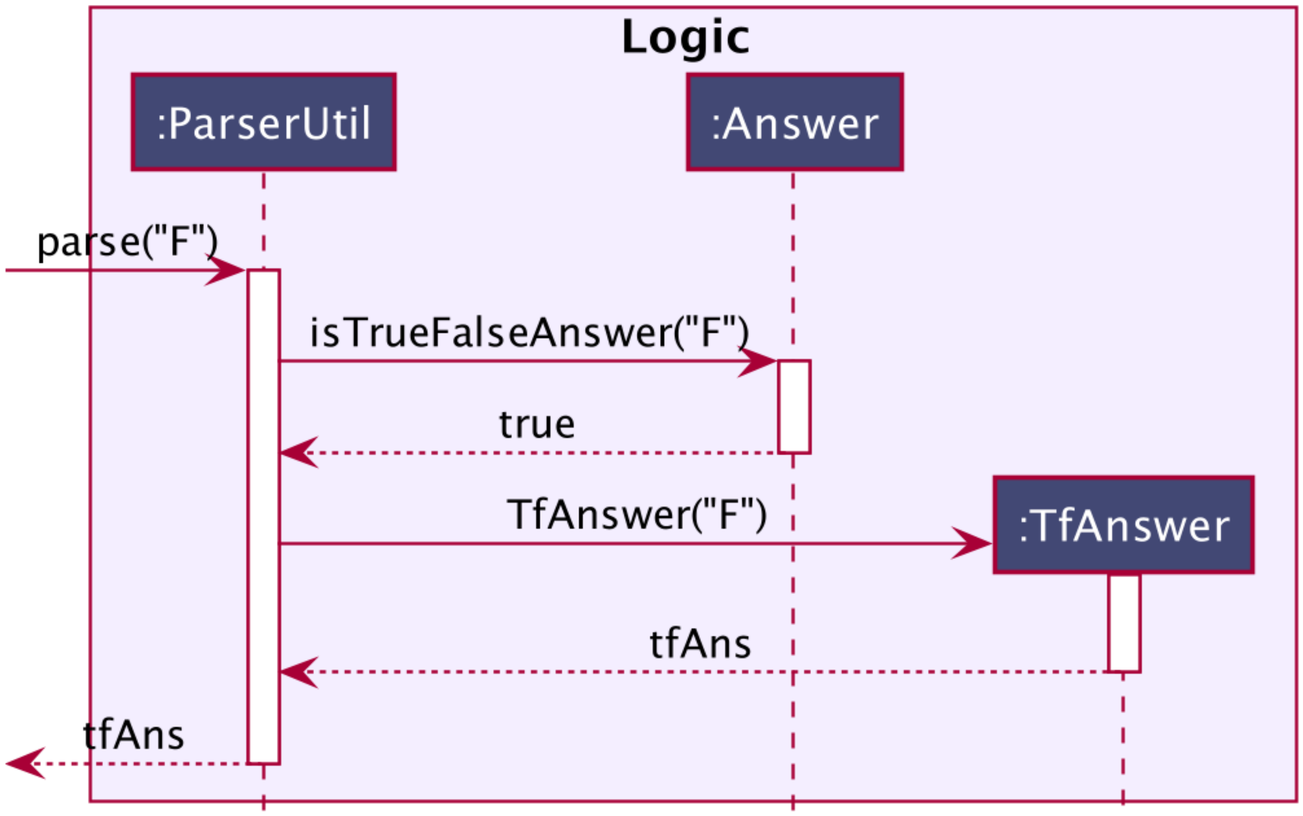
Types of Answers

When adding cards, the ParserUtil will parse the different answer inputs to create one of the three different types of answers.

TfAnswer

For a `TfAnswer` to be associated with a `Card`, the user will have to enter either `T` or `F` after the answer prefix `a/`. A sample command was mentioned in the example for `Flashcard` above: `add q/Is defensive code desirable at all times? a/F`.

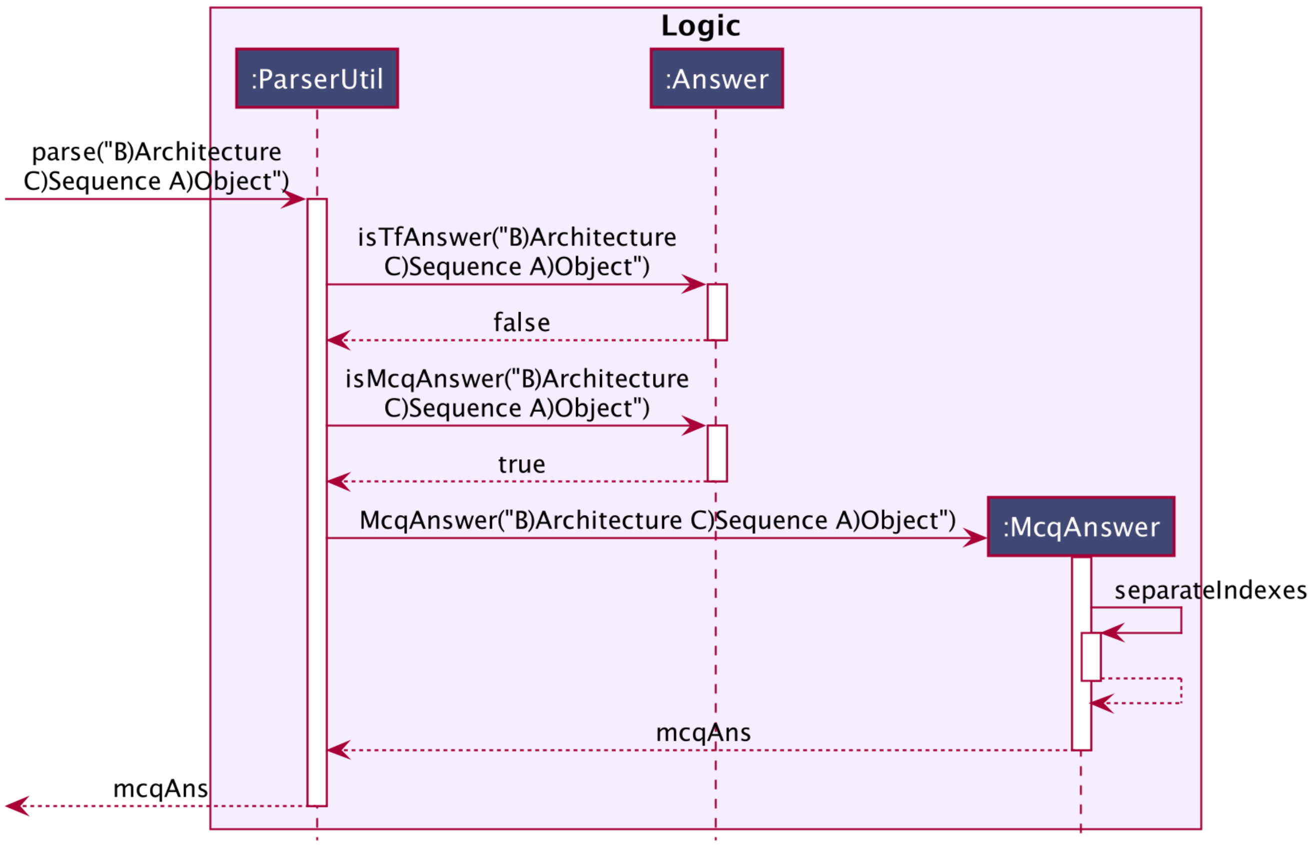
The following sequence diagram shows how the `ParserUtil` class creates a `TfAnswer` answer based on the given sample command:



McqAnswer

For an `McqAnswer` to be associated with a `Card`, the user will have to enter `A)CHOICE_A B)CHOICE_B C)CHOICE_C`, with the correct choice positioned first, for their answer after the answer prefix `a/`. A sample command was mentioned in the example for `Imagecard` above: `add p/file:/Users/Jing/ArchitectureDiagram.png q/What kind of diagram is this? a/B)Architecture C)Sequence A)Object`.

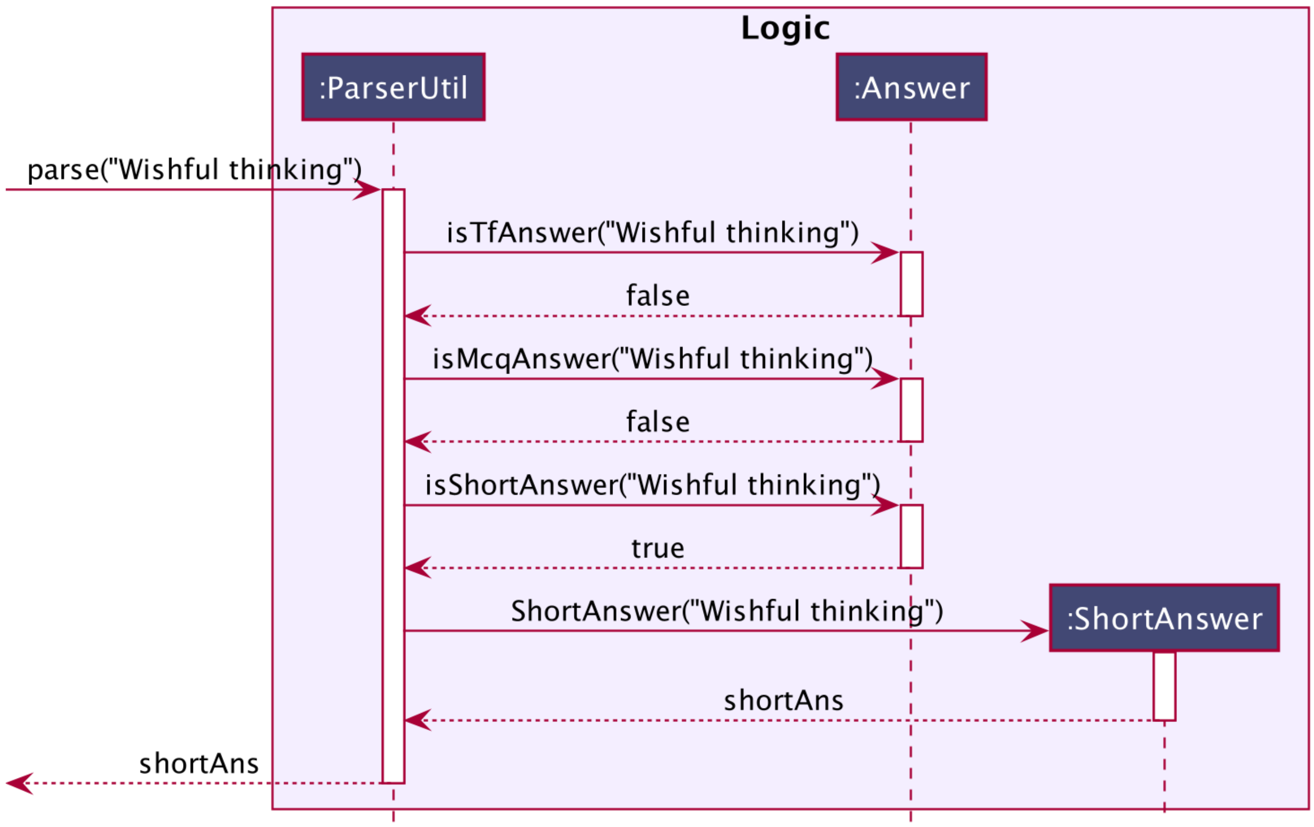
The following sequence diagram shows how the `ParserUtil` class creates an `McqAnswer` answer based on the given sample command:


Logic
:ParserUtil
:Answer
parse("B)Architecture C)Sequence A)Object")
isTfAnswer("B)Architecture C)Sequence A)Object")
false
isMcqAnswer("B)Architecture C)Sequence A)Object")
true
McqAnswer("B)Architecture C)Sequence A)Object")
:McqAnswer
separateIndexes
mcqAns
mcqAns

ShortAnswer

For a `ShortAnswer` to be associated with a `Card`, the user will have to enter an answer that does not fulfil the requirements of both `TfAnswer` and `McqAnswer` after the answer prefix `a/`. A sample command would be `add q/How does one go about solving recursion problems? a/Wishful thinking`.

The following sequence diagram shows how the `ParserUtil` class creates a `ShortAnswer` answer based on the given sample command:


Logic
:ParserUtil
:Answer
parse("Wishful thinking")
isTfAnswer("Wishful thinking")
false
isMcqAnswer("Wishful thinking")
false
isShortAnswer("Wishful thinking")
true
ShortAnswer("Wishful thinking")
:ShortAnswer
shortAns
shortAns

Design Considerations

- Due to the similarities in characteristics of the different answers and cards classes, a `Card` and `Answer` abstract class was created.
- More specific exceptions were thrown such as `WrongMcqAnswerException` and `WrongTfAnswerException` due to the large number of classes involved in the creation of a `Card` to help in pinpointing the input errors.