

Sahil S/O Sanjeev Gathani - Project Portfolio

What you can expect from this document

This document serves as an overview of my contributions to the StudyBuddyPro project. This project was part of an introductory software engineering module known as **CS2103T**. Moreover, the document aims to highlight my technical competence and documentation skills by using the StudyBuddyPro project as an example.

StudyBuddyPro was developed by [me](#), [Samuel Lim](#), [Chan Jun Ren](#), [Chen Kai Bin](#) and [Jasmine Yeo Jia Min](#), all sophomore students from National University of Singapore (NUS) studying Computer Science.

What our project is about

StudyBuddyPro is a desktop application designed for university students in computing-related fields to supplement their revision. StudyBuddyPro aims to consolidate commonly used studying tools into one integrated platform so the student does not have to fuss about switching between different applications. This prevents disorganisation and improves convenience.

StudyBuddyPro was developed for use through a Command Line Interface (CLI) to take advantage of the typing proficiency of computing students. However, a Graphical User Interface (GUI) is also provided for easier viewing, therefore providing the best of both worlds.

StudyBuddyPro's 3 main features are summarized in the table below.

Feature	Purpose
Flashcard	Create flashcards for quick revision and effective retention
Notes	Create general-purpose notes
Cheatsheet	Create and customize cheatsheets from relevant content found in Flashcard and Note features

Summary of my contributions to our project

This section provides an overview of the technical and non-technical contributions I made to the development of StudyBuddyPro. They highlight my ability to:

- 1) Design, model and implement features that are appropriate for our target users.
- 2) Work in a software development team setting and handle non-technical tasks.
- 3) Design test cases and write quality code that abides by software engineering principles.

You can find an overall summary of my contributions at my [RepoSense](#).

- **Major Feature Enhancements**

- Developed the Flashcard **Model**, **Logic** and **Storage** components. This enhancement provides the functionality for one of the three core features in the StudyBuddyPro application. Without it, the application would be missing a key revision tool and the application would be less attractive to the target market.

- **What this enhancement included:**

- The **Flashcard** model and its components such as **Question**, **Answer**, 'Title' and **Statistics** objects and their respective storage.
 - Basic commands to use the flashcard model such as **add** and **list**. Advanced features such as the **remind** feature.

- **How deep this feature was:**

- Implementing this enhancement required a deep technical understanding of the architecture of the software as it modified 3 out of the 4 major components.

- **How complete this feature was:**

- The enhancement is fully functional and goes beyond the basic needs of the user.
 - Advanced features like the **remind** feature makes use of spaced repetition techniques and automatic reminders so the user can effectively revise.

- **How much effort was needed to develop this feature:**

- The enhancement modified and improved existing commands, while also linking various commands which was very tricky to implement.

- **Minor Feature Enhancements**

- Code quality
 - Refactored **CommandResult** that all other features used to have separate classes to make codebase more object-oriented. (Pull request [#195](#))
 - Refactored **Storage** that all other features used to abide by the interface segregation principle. (Pull request [#340](#))

- **Other contributions**

- Project Management
 - Set-up the GitHub developers page for our team.
 - Integrated Travis CI, a continuous integration service, into our repository.
 - Managed the issue tracker for our repository by assigning issues, cross-referencing pull requests and closing issues when completed. (Pull request [#340](#), Issue [#301](#))
- Testing (Increased coverage by more than 5%)
 - Wrote all test cases for Flashcard model. (Pull requests [#321](#), [#333](#), [#337](#))
 - Wrote all test cases for overall StudyBuddyPro **Storage** component. (Pull request [#340](#))
- Documentation
 - Added comprehensive manual testing for **remind** feature in Developer Guide.
 - Wrote the introduction for the User Guide in a concise yet inviting manner.

My contributions to the User Guide

Given below are sections I contributed to the User Guide. They showcase my ability to write clear and inviting documentation that helps the non-technical end user use the application. Please note that some sections may be slightly truncated or edited for formatting.

Quick Start

NOTE Please ensure you have Java 11 or above installed before proceeding!

1. Download the latest version of [StudyBuddyPro.jar](#) [here](#).
2. Place the file in the folder you want to set as the home directory. All data and miscellaneous files associated with StudyBuddyPro will be placed in this folder.
3. Double-click [StudyBuddyPro.jar](#) to launch the application. The GUI should appear in a few seconds, and should look like the screen shown below. If not, please refer to the first question in the [FAQ](#) for help!

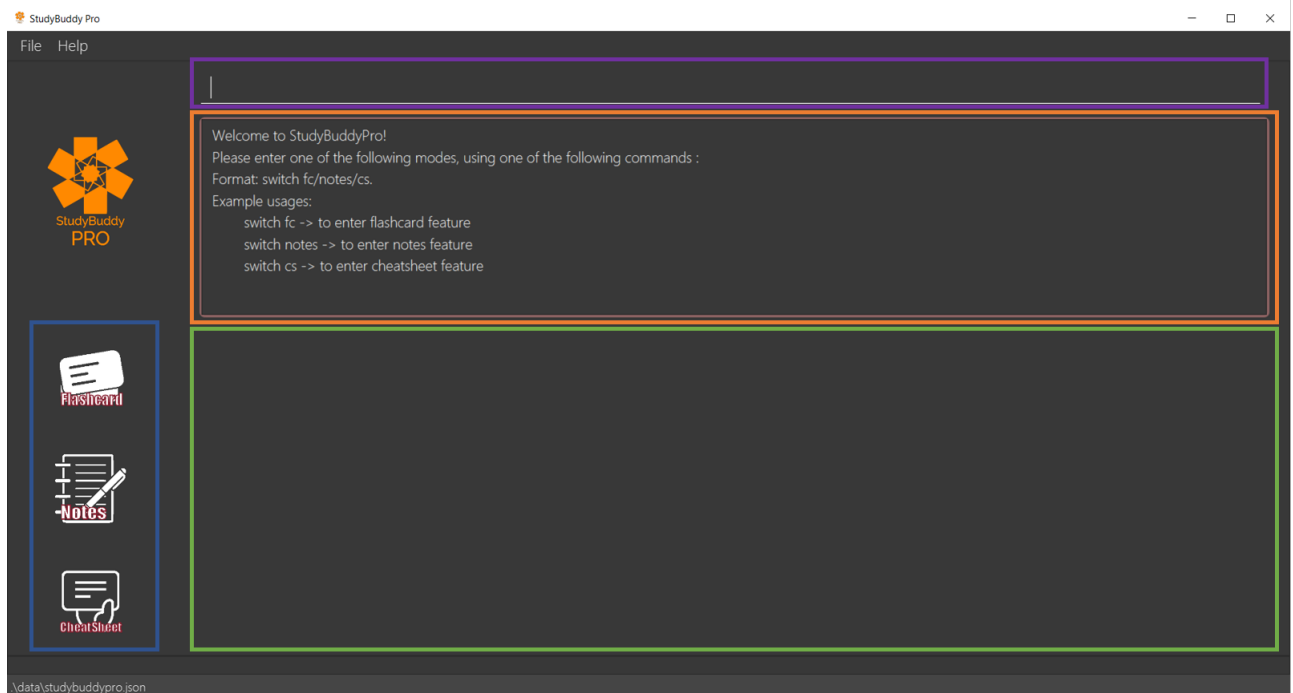


Figure 1. GUI of StudyBuddyPro application displayed on startup

4. Type a command in the command box execute it by pressing *Enter*. Typed commands will appear in the CLI highlighted in the purple box in the diagram above. Refer to the [Command Summary](#) section for a quick overview of all the available commands!
5. Output from the command will be shown in the boxes highlighted in orange and green in the diagram above. The green box is used to display a flashcard, note or cheatsheet while the orange box outputs feedback from commands.

TIP

The blue box in the diagram above with the "Flashcards", "Notes" and "Cheatsheets" logo can be used to quickly check which mode you are in! Switching into a mode will highlight the relevant mode's logo in an orange circle, as shown in this [figure](#).

Flashcard Feature

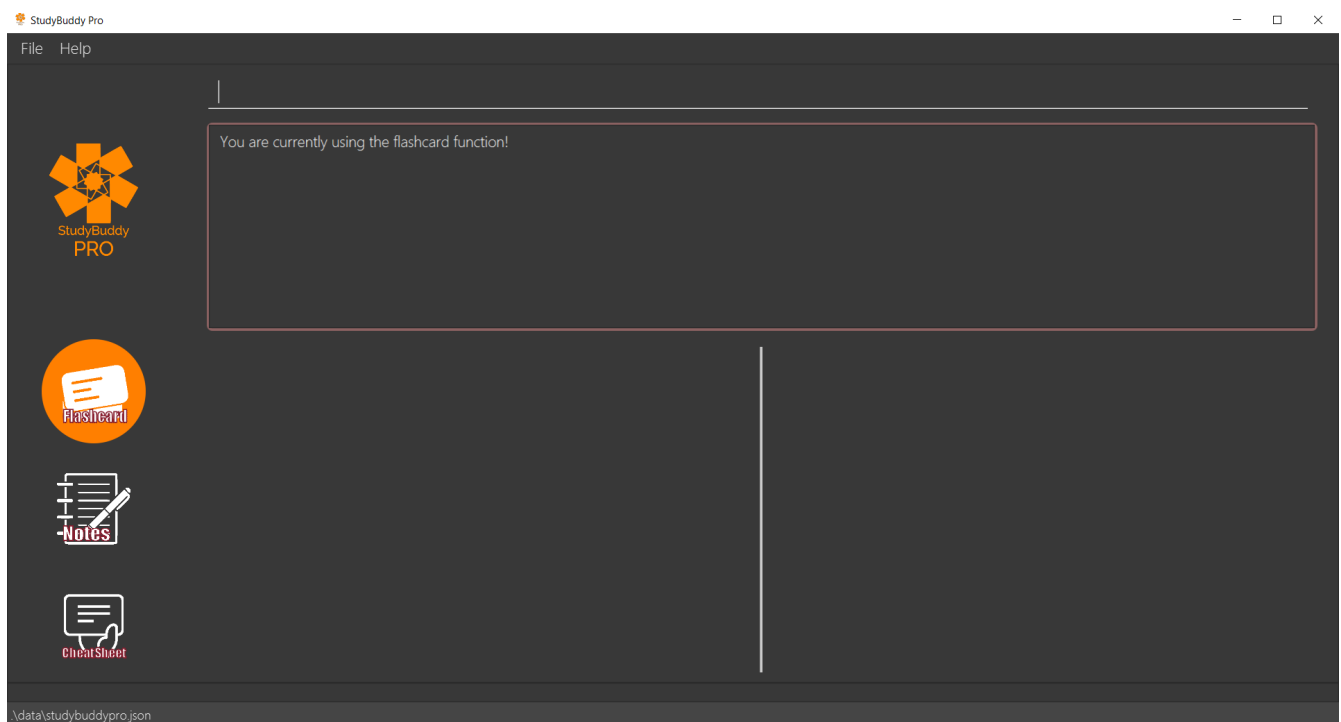
TIP

Good news - StudyBuddyPro comes with some preloaded flashcards, specially catered for you as a computing student! Be sure to take a look! Psst - here's a hint for our more tech-savvy users: You can delete your *flashcards.json* file in the StudyBuddyPro data folder to restore these default flashcards at any time. Of course, your current flashcards will be deleted as well!

Sick and tired of cramming all your revision at the last minute? Why not give our Flashcards feature a try! This feature can help you create your very own flashcards to help you consistently revise. With our built-in reminder features and timetrial modes to test yourself, use this feature and be on track to better revising habits today!

IMPORTANT

All the operations in this section assume that the user is in the *flashcard* mode. To be sure you are *flashcard* mode, please ensure you used the **switch fc** command before this. Your screen should now look like the one found in the screenshot below.



Find out what flashcards to revise today, or ones you may have missed: **remind**

This feature helps the user check which flashcards are due for revision today and which flashcards overdue for revision. StudyBuddyPro automatically sets the date the flashcard should next be

viewed at for optimal learning. These increments also scale with time i.e. newer flashcards will have to be viewed more often.

TIP Be sure to check in everyday to see which flashcards you have due!

NOTE StudyBuddyPro only marks a flashcard as revised and removes it from the due and overdue flashcard list when you see the flashcard's *answer* not just its question! For example, simply using the **view** command without the **show** command to reveal the flashcard's answer will not trick the system. Sorry, it's for your own good!

Example usage:

```
remind
```

If no flashcards due for revision today and no overdue flashcards:

Expected output:

```
Well done - No due or overdue flashcards!
```

If there are flashcards due for revision today but no overdue flashcards:

Expected output:

```
Here are the flashcards due today:  
1. Math Question 1 - What is 2 x 2?
```

If there are no flashcards due for revision today but there are overdue flashcards:

Expected output:

```
Here are your overdue flashcards:  
1. Math Question 1 - What is 2 x 2? (Was due on 2019-10-30)
```

If there are both flashcards due for revision today and overdue flashcards:

Expected output:

```
Here are the flashcards due today:  
1. Math Question 1 - What is 2 x 2?  
Here are your overdue flashcards:  
1. Math Question 2 - What is 3 x 2? (Was due on 2019-10-30)
```

My contributions to the Developer Guide

Given below are sections I contributed to the Developer Guide. They showcase my ability to explain complicated technical information in an easy-to-read, digestible manner. This was achieved through the use of examples and industry standard technical diagrams. Please note that some sections may be slightly truncated or edited for formatting.

Remind Feature

Overview

This feature aims to help the user stay on track with the user's revision schedule through two other sub-features. The first sub-feature is the **remind** command which helps the user keep track of which flashcards are due or overdue for revision. The second sub-feature is integrated with the **exit** command and is illustrated in the activity diagram below. For example, a typical user may use the **exit** command to exit the application without realizing they still had due or overdue flashcards left to revise. StudyBuddyPro will automatically warn the user about these unrevised flashcards after which the user can decide if they wish to revise these flashcards or exit StudyBuddyPro anyway.

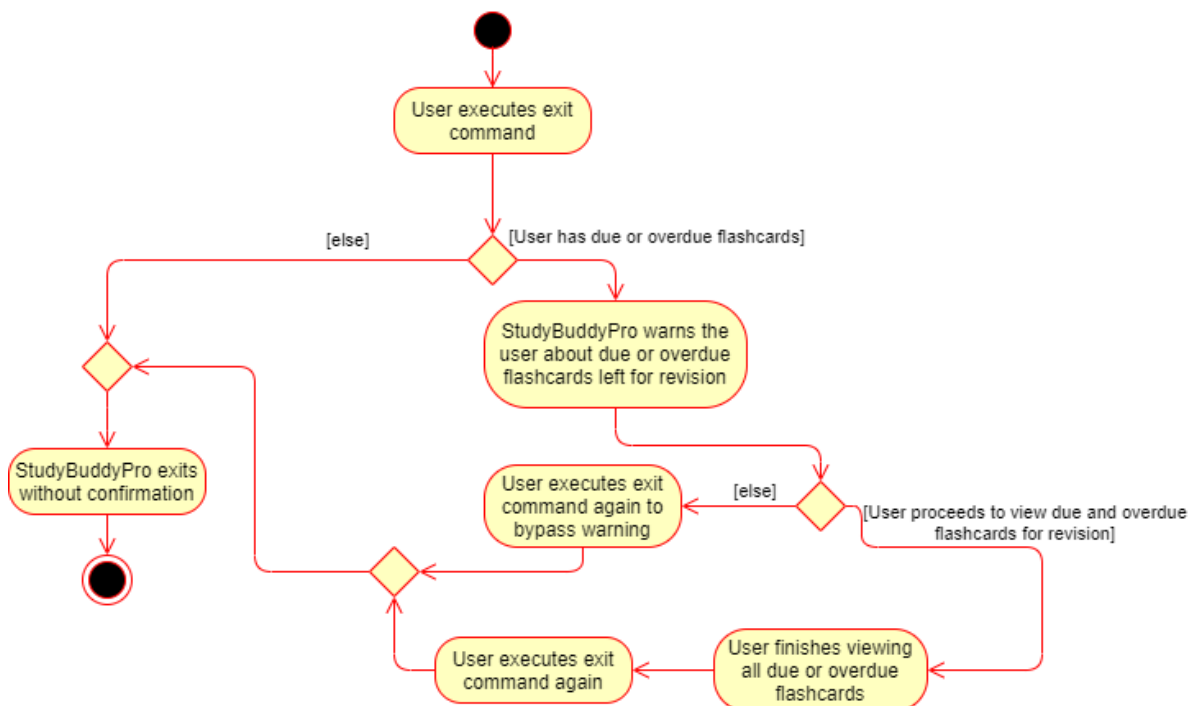


Figure 2. Activity diagram of user trying to exit StudyBuddyPro through the **exit** command

Implementation of **Statistics** class

In order to fully understand how the remind feature was implemented, it is important to understand how a flashcard stores the relevant data fields it needs such as its last viewed date. All the relevant statistics pertaining to each flashcard is contained within a **Statistics** object. Each **Statistics** object also has a **ScheduleIncrement**, which helps keep track of when the flashcard should be next viewed for optimum spaced repetition learning. The relationships between the **Flashcard**, **Statistics** and **ScheduleIncrement** classes are summarized in the class diagram below.

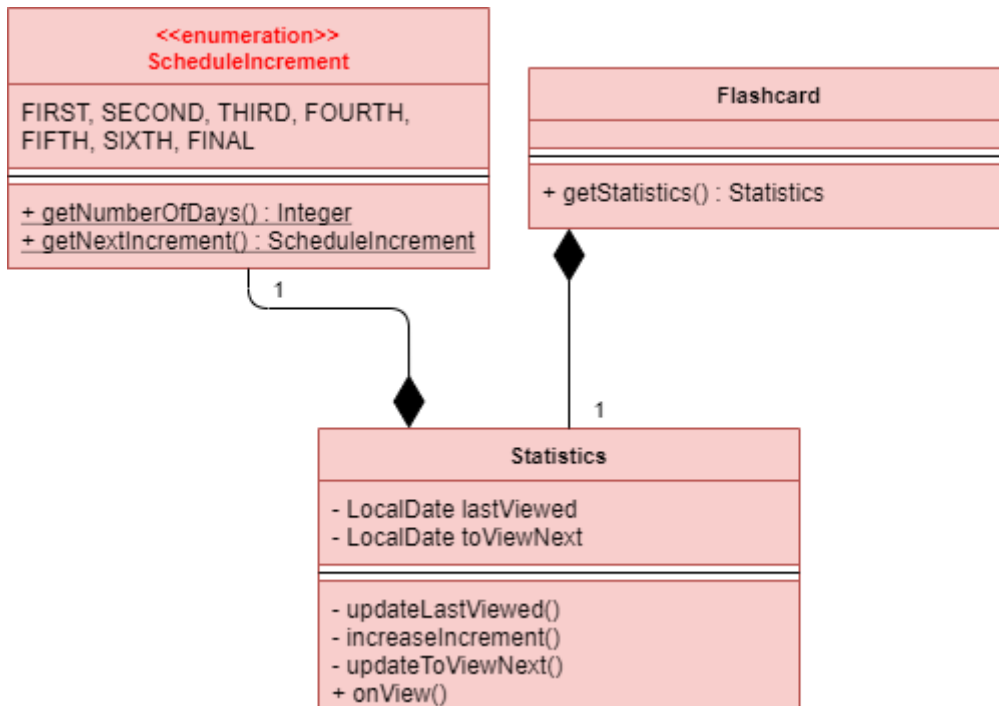


Figure 3. Class diagram of *Flashcard*, *Statistics* and *ScheduleIncrement* classes

The interactions between the relevant classes can be illustrated through the following scenario:

Today's date is 10/10/2019. Jim decides to view a flashcard which was previously viewed on 7/10/2019 and was scheduled to be viewed on 8/10/2019.

Step 1. Jim views the flashcard, either through a *timetrial* or by specifically finding the relevant flashcard and viewing the flashcard's answer using the *view* and *show* commands.

Step 2. When the relevant command is called, the *Flashcard* object's *Statistics#onView()* method is invoked to update its statistics.

Step 3. The *onView()* method calls its helper functions in the specific order shown in the code snippet below. The order is significant as calling the *updateToViewNext()* method before the *increaseIncrement()* method would result in the wrong *ScheduleIncrement* being used to update the *toViewNext* attribute.

```

/**
 * Updates all fields from the described methods. To be used when a {@Link Flashcard} is viewed.
 * Ensures lastViewed and toViewNext will never conflict and be the same date
 */
public void onView() {
    LocalDate currentDate = LocalDate.now();
    if (toViewNext.isEqual(currentDate) || toViewNext.isBefore(currentDate)) {
        updateLastViewed();
        increaseIncrement();
        updateToViewNext();
    } else {
        updateLastViewed();
    }
}

```

Figure 4. Code snippet of *Statistics#onView()*

Step 4. In this case, the "if" clause of the code snippet above is triggered since the *toViewNext* date

(8/10/2019) was **before** the current date (10/10/2019).

Step 5. The various helper methods execute their relevant functions to update their respective fields. For example, the **lastViewed** date is now updated to 10/10/2019.

Implementation of **remind** command

IMPORTANT | The following commands assume that the user is in the *flashcard* mode.

The **remind** command is facilitated by the **RemindFeatureUtil** class and extends the abstract **Command** class as summarized in the class diagram below. This diagram also provides a summary of the **Logic** components of the **remind** command.

{Please refer to the complete Developer Guide for the diagram!}

The **remind** command also interacts with the **Model** component, specifically through indirect interactions with each **Flashcard** object currently in **StudyBuddyPro**. This was explained earlier through Figure 3.

The following example serves to provide an overview of the high-level architecture involved for the **remind** command.

Step 1. The user types "remind" and executes the command to check what flashcards the user has due.

Step 2. **MainWindow** would call **LogicManager#execute()** to hand over control flow to the **Logic** component.

Step 3. **LogicManager** parses the user input to determine which command is being called. After determining that a **remind** command is being executed, the relevant **RemindCommand** object is created.

Step 4. The **RemindCommand** object determines which flashcards are due and overdue, if any, and returns the output packaged in a **FlashcardCommandResult** object.

Step 5. **MainWindow** extracts the relevant output to be shown to the viewer from the **FlashcardCommandResult** and displays it.

These steps are illustrated in further detail in the sequence diagram below.

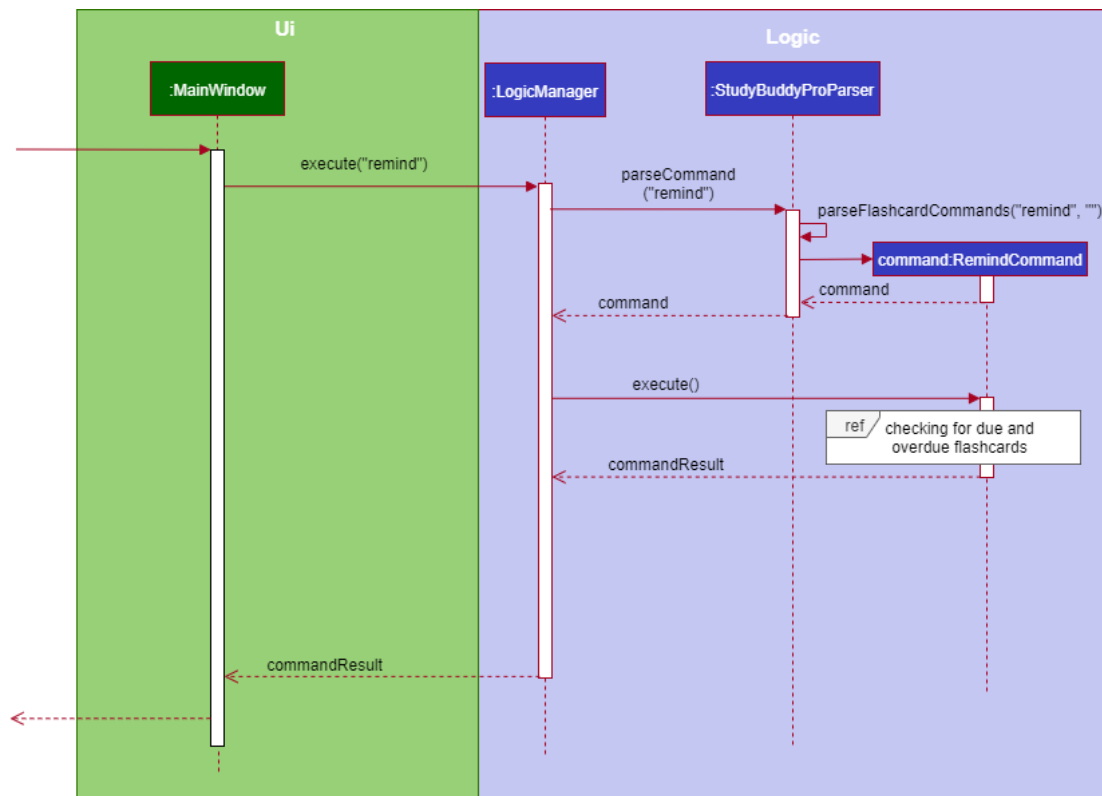


Figure 5. Sequence diagram of high-level overview of how the **remind** command works

With reference to **Step 4** above, the implementation of how the **RemindCommand** object determines which flashcards are due or overdue can be further examined. A broad overview is provided in the sequence diagram below and explained in further detail below the diagram.

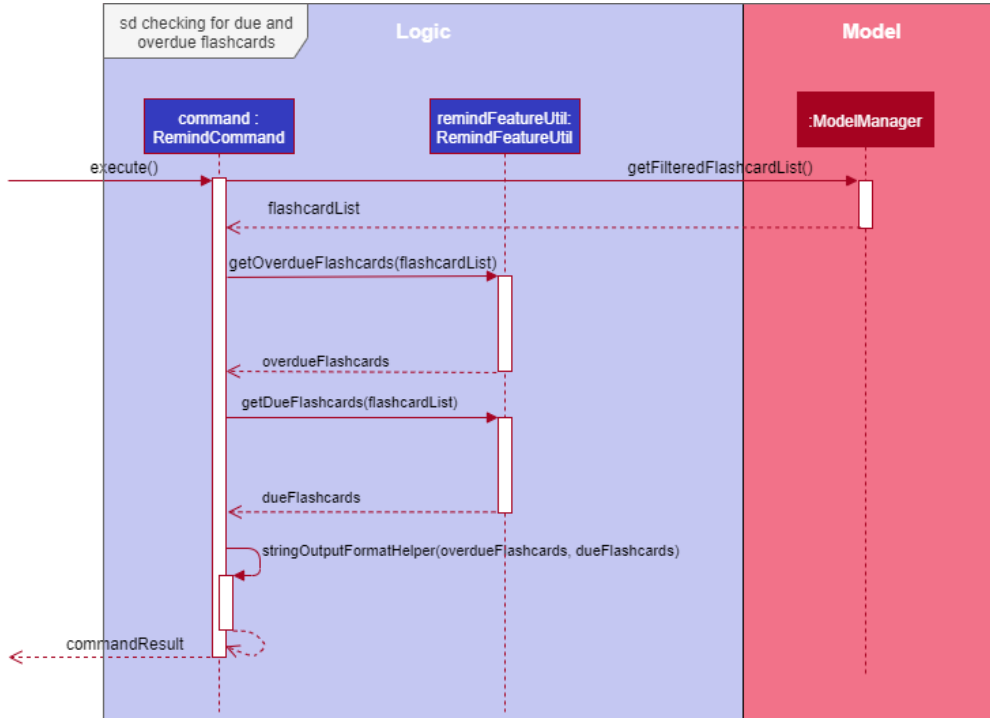


Figure 6. Sequence diagram of checking for due and overdue flashcards

- Whenever a **RemindCommand** object is created, it creates its own **RemindFeatureUtil** object.
- The **RemindCommand** object communicates directly with the **ModelManager** object through the exposed **Model** interface by calling upon the **ModelManager#getFilteredFlashcardList()** to get the current list of flashcards in the application.

- The `RemindCommand` passes the list of flashcards to its helper `RemindFeatureUtil` class which retrieves due and overdue flashcards through the `RemindFeatureUtil#getDueFlashcards()` and `RemindFeatureUtil#getOverdueFlashcards()` respectively.
- Overdue flashcards are collected by iterating through the entire flashcard list and checking if each flashcard's `toViewNext` attribute is before the current system date. A similar approach is adopted to check for due flashcards.
- The two lists of due and overdue flashcards are formatted into a String and passed back through the `FlashcardCommandResult` object created.

Design and Implementation Considerations

The following sections describes the considerations taken into account when developing this feature.

Implementation Consideration: How to keep track of when a `Flashcard` object was last viewed?

- **Alternative 1 (current choice):** Design a new `Statistics` class and make a `Flashcard` object store a `Statistics` object. The `Statistics` object then keeps track of the flashcard's last viewed date.
 - Pros: Using a separate `Statistics` class is more in line with Object-Oriented Programming (OOP) practices and means the same class can be adapted and used to track statistics of other objects in StudyBuddyPro in the future.
 - Cons: Additional class adds additional maintenance and issues such as difficulty converting this class into a format that can be saved and read.
- **Alternative 2:** Use an integer attribute field in each `Flashcard` object to keep track of how many times it was viewed.
 - Pros: Easier to maintain: The addition of a new class increases overall coupling compared to adding a single new attribute field.
 - Cons: Bad OOP practice and makes it difficult to implement future changes. It may be unclear what the integer value represents and makes debugging more difficult.

[Proposed] Future Improvements

- Improve the formula used for determining when flashcards should be revised.
 - We could take into account other factors such as user confidence level or number of times flashcard was answered correctly or incorrectly to create a more dynamic reminding schedule.
- Introduce desktop notifications for the user.
 - As a user may not open StudyBuddyPro everyday, StudyBuddyPro can be integrated with the desktop system calendar to provide notifications when flashcards are due for revision.