Zhang Yuanyu - Project Portfolio

PROJECT: TeaPet

Overview

TeaPet is a personal manager application for primary school form teachers. The user interacts with it via a Command Line Interface (CLI) along with a Graphical User Interface (GUI) created with JavaFX. The application is written in Java and only Java 11 supports this application. TeaPet is meant for online tracking purposes. With that being said, there are future plans to expand the application to an online version.

Summary of contributions

- **Major enhancements**:
  1. Added **the ability for form teachers to track administrative information of the class**
     - What it does: The form teacher will be able to save, delete and view administrative details of the class.
     - Justification: The form teacher has to record administrative information of the class daily, hence this feature is essential for that purpose. With this feature, the form teacher can easily track the administrative information of the class whenever and wherever.
     - Highlights: This enhancement allows teachers to only save the most updated administrative list as today's date to prevent mutability of other dates. The teacher is also able to view the dates that contains administrative information of the class.
  2. Sorted the student list alphabetically
     - What it does: The student list is be sorted in alphabetical order for easy viewing.
     - Justification: Class lists are always sorted in alphabetical order, so it is only logical for TeaPet to do the same.
     - Highlights: When teacher adds, deletes or edits a student, TeaPet will re-sort the class list accordingly, without the need to restart TeaPet.
- **Minor enhancements**:
  1. Added a temperature and attendance attribute to student class with the necessary restraints
  2. Changed the way student edit function works. Previously, editing one field of the student results in that field to be edited and the rest being changed back to default value. Now, editing one field of the student still updates that specific field, but the rest of the fields are left untouched.
- **Code contributed**: [Functional code and Test code]_
- **Other contributions**:

- Project management:
  - Git master of the project. Managed the team's repository and was in charge of the managing of the pull requests to prevent unnecessary merging of pull requests.
  - Helped teammates to fix their errors that resulted in travis build to fail.
  - Helped teammates with testing errors.
  - Explained to teammates the flow of AB3 to better help them understand the code and therefore aiding them in implementing their own features.
- Enhancements to existing features:
  - Sorted the student list in alphabetical order (Pull request #274)
  - Loosened the restrain of tags to allow better flexibility of creating a tag (Pull request #228)
  - Added new restrains for name to make naming of a student to be more logical (Pull request #258)
- Documentation:
  - Updated User Guide to include Admin feature
  - Updated User Guide with FAQs and the glossary
  - Updated Developer Guide to include Admin feature
- Community:
  - PRs reviewed (with non-trivial review comments): #118, #112, #96, #78, #74, #67
  - Contributed to forum discussions: 12
  - Reported bugs and suggestions for other teams: Practical Exam Dry Run

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.*

Student Administration [Done by Zhang Yuan Yu]

TeaPet's Class Administration feature is used to keep track of administrative details such as daily attendance and temperature recordings.

Displaying of the class's last updated administrative information: `admin`

Shows the last updated administrative information in the student list.

Format:

```
admin
```

Expected Outcome:

```
The Student list now displays last updated ADMIN details
```

Figure 1. After using `admin` command

Displaying of the dates with administrative information of the class: `admin dates`

Shows a list of dates that contains administrative information of the class.

Format:

```
admin dates
```

Expected Outcome:

```
List of dates with admin details of the class displayed!
```

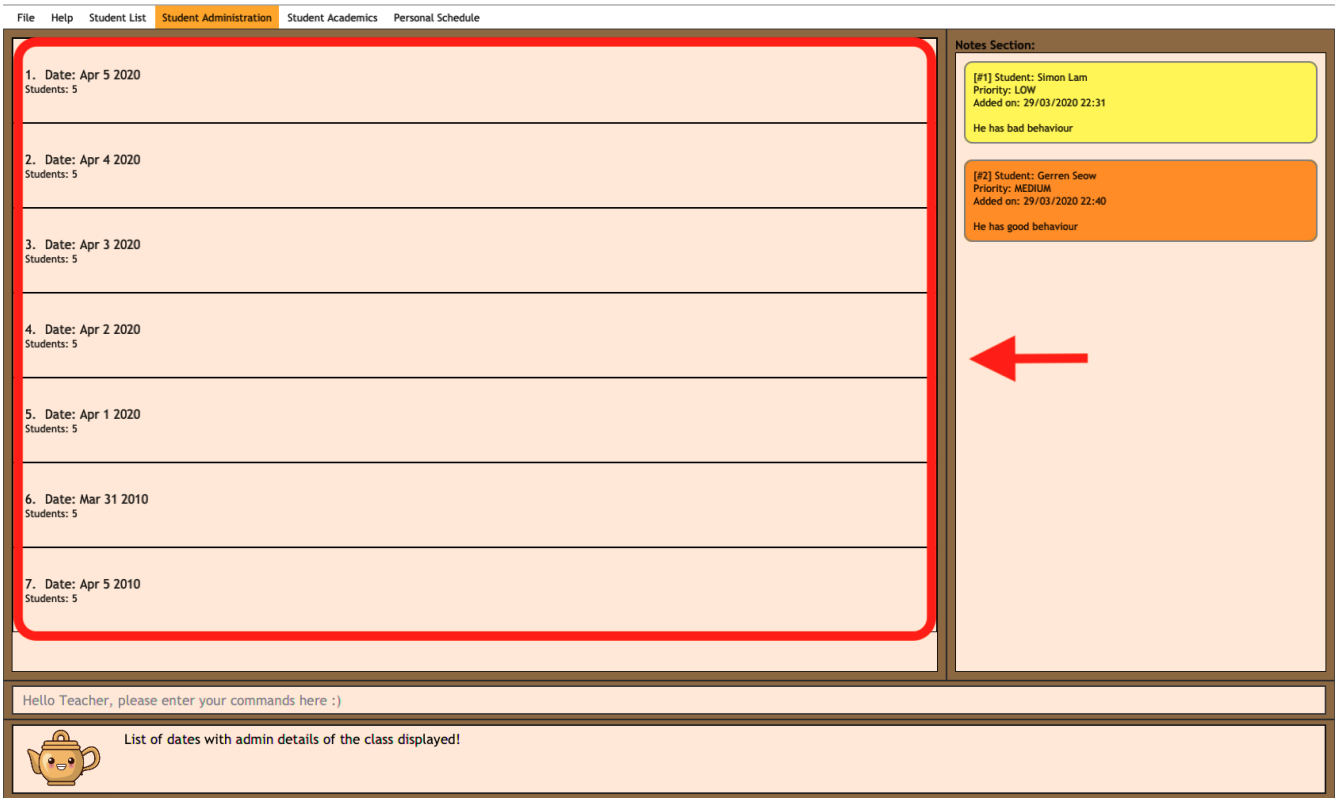

Figure 2. After using `admin dates` command

Fetching of administrative information of the class at the specified date: `admin fetch`

Retrieves the administrative information of the class at the date provided.

Format:

```
admin fetch DATE
```

- Date should be written in **YYYY-MM-DD** format.
- If date provided is not in data base, an error message will be shown.

Example:

- `admin fetch 2020-04-02`
  Retrieves the administrative information of the class at on April 2 2020.

Expected Outcome:

```
Class admin details for Apr 2 2020 listed!
```

Saves today's administrative information of the class: `admin save`

Saves today's administrative information of the class.

Format:

```
admin save
```

- Takes a screenshot of the most updated class administrative details and saves it as today's date.
- If the class administrative information has been saved before earlier on the same day, saving it again will result duplicates, resulting in an error and an error message.
- If there are changes to the class administrative information today and you wish to save it again, you would have to delete today's date from the list of dates and save it again.
- Old dates and their administrative details cannot be edited to prevent mutation of data.

Example:

- `admin save`
  Saves the administrative information of the class with today's date, taking April 8 2020 as an example.

Expected Outcome:

```
This admin list has been saved for Apr 8 2020
```

Deleting of administrative information of the class at the specified date: `admin delete`

Deletes the administrative information of the class at the specified date.

Format:

```
admin delete DATE
```

- Date should be written in **YYYY-MM-DD** format.
- If date provided is not in data base, an error message will be shown.

Example:

- `admin delete 2020-04-08`
  Deletes the administrative information of the class at on April 8 2020.

Expected Outcome:

```
Admin list has been deleted for Apr 8 2020
```

Student Particulars Commands

Here are the commands to manage students. They require the prefix `student`.

Table 1. Student commands of TeaPet

Command	Format	Expected outcome
`student detailed`	`student detailed`	Displays the detailed details of the class
`student add`	`student add name/NAME [phone/PHONE] [email/EMAIL] [adr/ADDRESS] [nok/NOK] [temp/TEMPERATURE] [att/ATTENDANCE]`	Adds a student into the class with the respective particulars
`student edit`	`student edit INDEX [phone/PHONE] [email/EMAIL] [adr/ADDRESS] [nok/NOK] [temp/TEMPERATURE] [att/ATTENDANCE]`	Edits the student at the index with the respective particulars
`student delete`	`student delete INDEX`	Deletes the student at the index
`student clear`	`student clear`	Deletes the entire class list
`student find`	`student find KEYWORD`	Searches through the class list and filter students whose names have the specified KEYWORD

Student Administration Commands

Here are the commands to manage students. They require the prefix `admin`.

Table 2. Student Administration commands of TeaPet

Command	Format	Expected outcome
`admin dates`	`admin dates`	Shows the dates with admin information of the class
`admin save`	`admin save`	Saves the most updated administrative information of the class as today's date
`admin fetch`	`academics fetch DATE`	Fetches the administrative information of the class at the specified date
`admin delete`	`admin delete DATE`	Deletes the administrative information of the class at the specified date

Glossary [Done by Zhang Yuanyu]

The table below provides the prefixes used in certain commands. The definitions are provided for you to reference easier and have a better understanding of what they do.

Table 3. Command Prefix

Prefix	Attributes	Used in the following Command(s)
name/	Name of student	Student Particulars, Notes
phone/	Phone number	Student Particulars
email/	Email address	Student Particulars
adr/	Address	Student Particulars
tag/	Tag	Student Particulars
nok/	Next of Kin details	Student Particulars
temp/	Temperature	Student Particulars
att/	Attendance	Student Particulars
stu/	Student name and score obtained	Student Academcis
desc/	Assessment description	Student Academcis
type/	Assessment type	Student Academcis
date/	Date	Student Academcis, Personal Scheduler
eventName/	Event name	Personal Scheduler

Prefix	Attributes	Used in the following Command(s)
startDateTime/	Starting date and time of event	Personal Scheduler
endDateTime/	Ending date and time of event	Personal Scheduler
recur/	Recurring type	Personal Scheduler
color/	Color code	Personal Scheduler
indexGet/	Get index of event	Personal Scheduler
mode/	Event view mode	Personal Scheduler
cont/	Content of note	Notes
pr/	Note priority level	Notes

FAQ [Done by Zhang Yuanyu]

This section will provide answers to all Frequently Asked Questions by our users.

1. *How do I transfer my data to another Computer?*

   Install the app in the other computer and overwrite the empty data file it creates with the file that contains the data of your previous TeaPet folder.

2. *How do I transfer the information in TeaPet to the co-form teacher?*

   Right now, TeaPet does not support the transfer of data, but the feature will be coming soon in the near future!

3. *Why can't I see my personal schedule from a while back?*

   To see the schedule for a specific week, you could use the command `schedule view mode/weekly date/DATE`, where date is one of the date in the week you are seeking for.

4. *How do I retrieve back all the class list in TeaPet if I accidentally cleared all the content?*

   Right now TeaPet does not support a backup feature, hence it would be best if you do not accidentally use the clear command. The backup feature will be coming soon in the near future!

5. *TeaPet is not working on my computer. How do I fix it?*

   Ensure that your computer is running on Java 11 and not other versions. TeaPet does not support other versions of Java.

6. *Sometimes the text size of TeaPet is too small. How do I fix it?*

   TeaPet runs best on full screen mode. Hence, it is strongly encouraged that you maximise the screen when using TeaPet for clarity.

7. *Sometimes I forget the various commands to use, where can I find the list of commands?*

   You could view enter help tab by clicking F1, or by pressing the `help` button in the tabs section located at the top of TeaPet, which will then lead you to this user guide to provide you with the help you need!

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.

Class administration feature

The class administration feature keeps track of essential student administrative details. The feature comprises of four commands namely.

The structure of the Admin commands are as shown below:

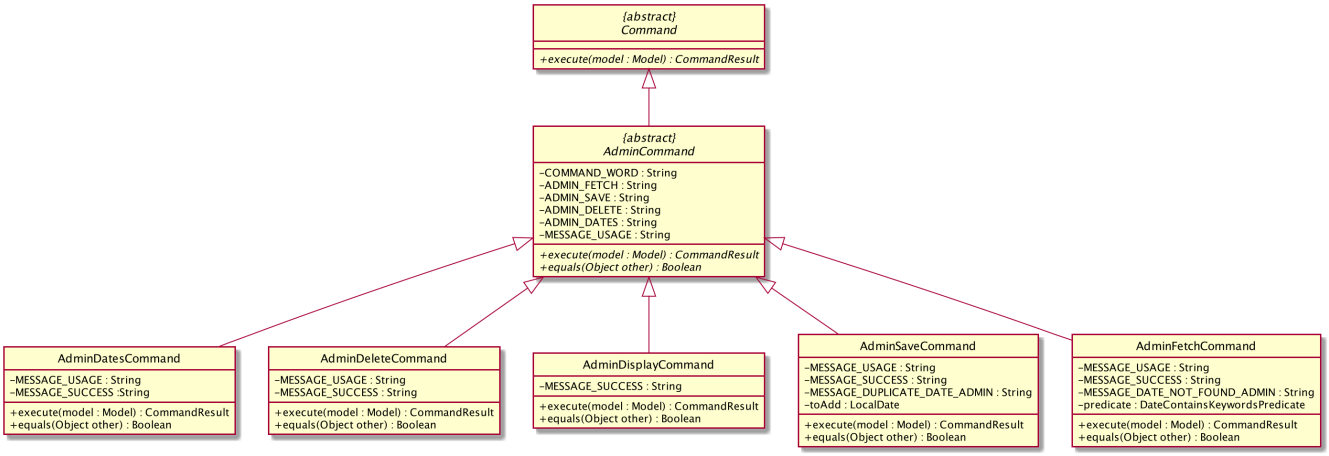

Figure 3. Admin Commands Diagram

These are the various admin commands to try:

- `admin` - Displays the most updated class administrative details.
- `admin dates` - Displays the dates that hold administrative information of the class.
- `admin save` - Saves today's administrative information of the class.
- `admin delete` - Deletes the administrative information of the class at the specified date.
- `admin fetch` - Fetches the administrative information of the class at the specified date.

Admin Display Command

Implementation

The following is a detailed explanation of the operations which `admin` performs.

Step 1. The `AdminDisplayCommand#execute(Model model)` method is executed which does not take in any arguments.

Step 2. The method `Model#updateFilteredStudentList(PREDICATE_SHOW_ALL_STUDENTS)` will then be called to update the filtered student list to show all current students in the student list.

NOTE	If the class list is empty, a blank page will be shown.

Step 3. The command box will be reflected with the `AdminDisplayCommand#MESSAGE_SUCCESS` constant and a new `CommandResult` will be returned with the message.

NOTE	If the wording of the `admin` command contains error(s), an unknown command message will be displayed.

Admin Dates Command

Implementation

The following is a detailed explanation of the operations which `admin dates` performs.

Step 1. The `AdminDatesCommand#execute(Model model)` method is executed which does not take in any arguments.

Step 2. The method `Model#updateFilteredDateList(PREDICATE_SHOW_ALL_DATES)` will then be called to update the filtered date list to show all current dates in the date list.

NOTE	If the date list is empty, a blank page will be shown.

Step 3. The command box will be reflected with the `AdminDatesCommand#MESSAGE_SUCCESS` constant and a new `CommandResult` will be returned with the message.

NOTE	If the format or wording of the `admin dates` command contains error(s), an unknown command or a wrong format message will be displayed.

Admin Save Command

Implementation

The following is a detailed explanation of the operations which `admin save` performs.

Step 1. The `AdminSaveCommand#execute(Model model)` method is executed which takes in today's date as an argument.

Step 2. The method `Model#updateFilteredStudentList(PREDICATE_SHOW_ALL_STUDENTS)` will then be called to update the filtered student list to show all current students in the student list.

Step 3. Sequentially, a date constructor will then called, creating a date object with today's date and `Model#getFilteredStudentList()`

Step 4. The method `Model#addDate(Date date)` will then be called to add the date. This will then trigger the `UniqueDateList#addDate(Date toadd)` method, which will throw `DuplicateDateException` if the date that is been added exists, with the duplicate dates error message.

Step 5. The command box will be reflected with the `AdminSaveCommand#MESSAGE_SUCCESS` constant and a new `CommandResult` will be returned with the message.

NOTE	If the format or wording of saving of a date contains error(s), an unknown command or wrong format error message will be displayed.

The following activity diagram summarizes what happens when a user executes admin save command:

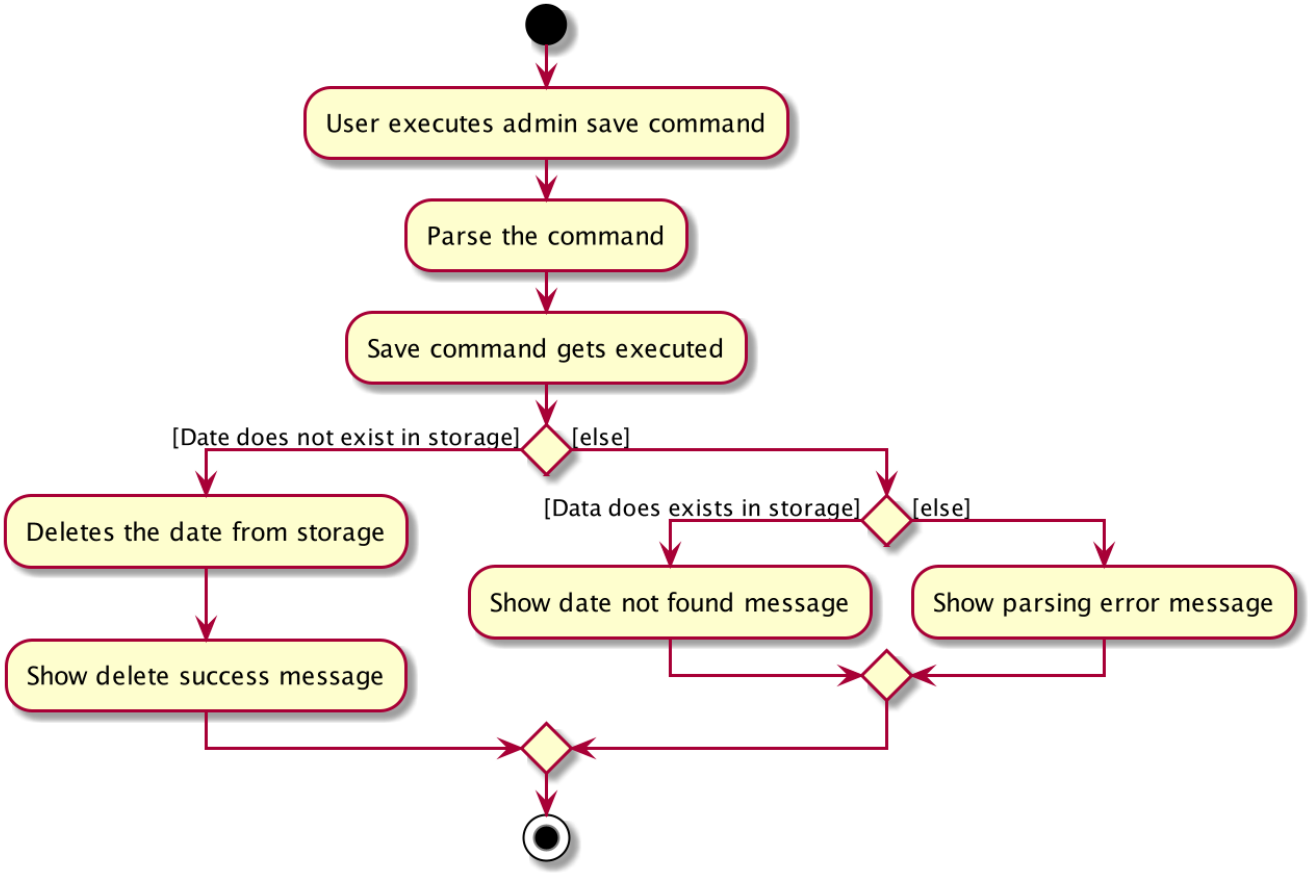

Figure 4. Admin Save Activity Diagram

Design Considerations

Aspect: Which date to save

- **Alternative 1 (current choice):** Saves the most updated administrative list as today's date.
  - Pros: Easy to implement and prevents mutation of dates.
  - Cons: The user will be unable to overwrite dates.
- **Alternative 2:** Saves the most updated administrative list as any date.
  - Pros: The user can mutate any dates as he or she wishes.
  - Cons: Hard to implement, and possible accidental mutation of dates.

Aspect: Allow overwriting of data

- **Alternative 1 (current choice):** Saving a date that exists in the storage is not allowed.
  - Pros: Easy to implement and prevent accidental mutation of data
  - Cons: Hard to implement.

- **Alternative 2:** Saving a date that exists in the storage is allowed.
  - Pros: User can make necessary changes to the dates where errors exists.
  - Cons: Hard to implement and could result in accidental mutation of dates.

Admin Delete Command

Implementation

The following is a detailed explanation of the operations which `admin save` performs.

Step 1. The `AdminDeleteCommand#execute(Model model)` method is executed which takes in a DateContainsKeywordsPredicate object as an argument. User input will be parsed first to a DateContainsKeywordsPredicate object before passing to the `AdminDeleteCommand` constructor.

NOTE	Date is to be entered in YYYY-MM-DD format, or a ParseException will be thrown and an error message will be displayed.

Step 2. The method `Model#updateFilteredStudentList(DateContainsKeywordsPredicate predicate)` will then be called to update the filtered date list to show the date that matches the given predicate. If no such date is found after searching through the `UniqueDateList#dates`, a DateNotFoundException will be thrown with an error message displayed.

Step 3. After the date has been found, the method `Model*deleteDate(Date target)` will then be called to remove the specified date from `UniqueDateList`.

The following sequence diagram shows how the add operation works:

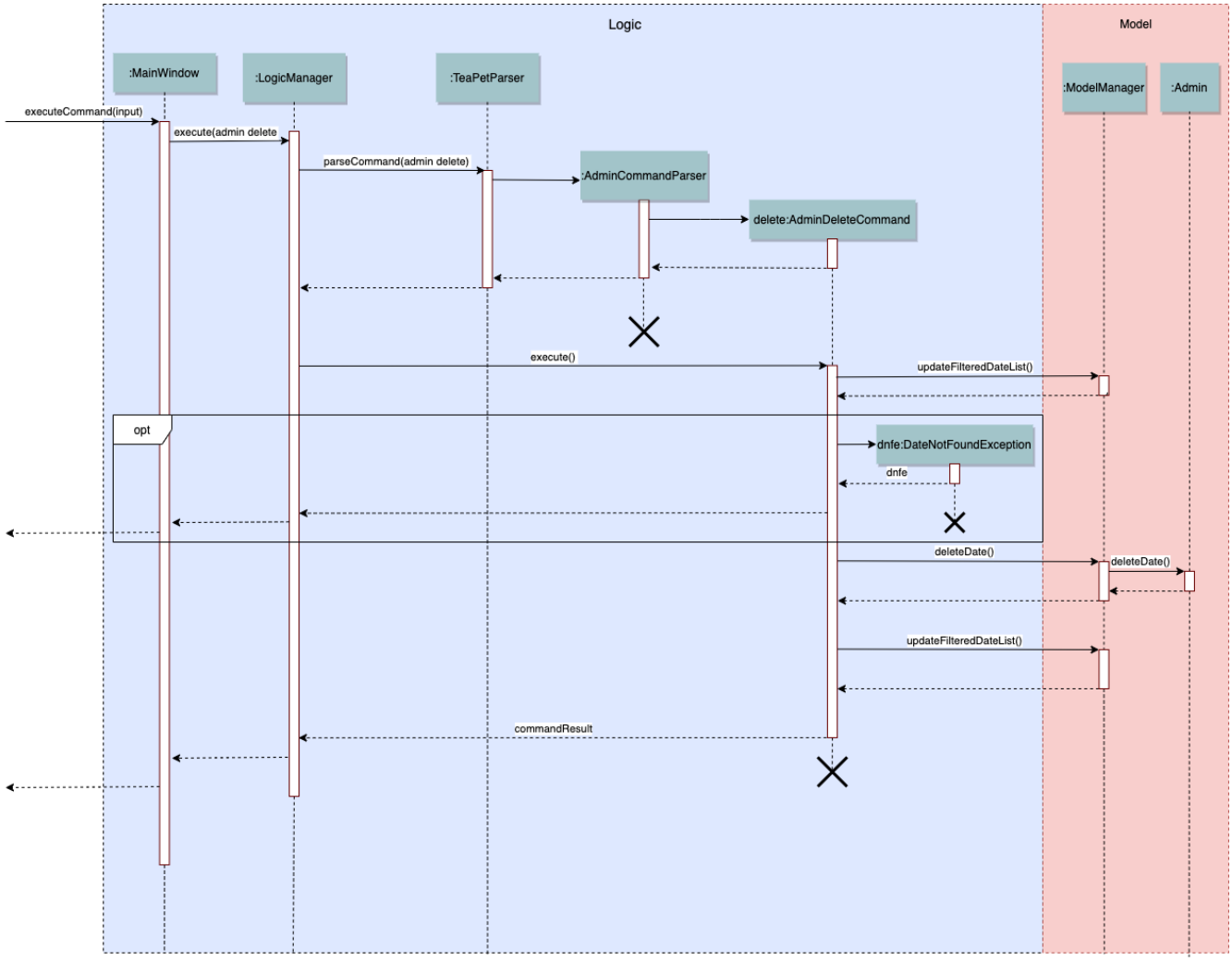

Figure 5. Admin Delete Sequence Diagram

Admin Fetch Command

Step 1. The `AdminFetchCommand#execute(Model model)` method is executed which takes in a DateContainsKeywordsPredicate object as an argument. User input will be parsed first to a DateContainsKeywordsPredicate object before passing to the `AdminFetchCommand` constructor.

NOTE	Date is to be entered in YYYY-MM-DD format, or a ParseException will be thrown and an error message will be displayed.

Step 2. The method `Model#updateFilteredStudentList(DateContainsKeywordsPredicate predicate)` will then be called to update the filtered date list to show the date that matches the given predicate. If no such date is found after searching through the `UniqueDateList#dates`, a DateNotFoundException will be thrown with an error message displayed.

NOTE	The sequence diagram for `admin fetch` command is similar to that of `admin delete` command.

Appendix A: 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
* * *	form teacher	take the attendance of my students	know who is present for my class
* * *	form teacher	have a schedule tracking my events	know what I need to attend/do in a day
* * *	form teacher	maintain of a list of students who have completed my homework	know who has not submitted my homework
* * *	form teacher	take down notes for student's behavior	track the behaviour of my students
* * *	form teacher	see the scores of my class	track the academic progress of my class
* * *	form teacher	add students	add new students to the class list
* * *	form teacher	delete a student	remove students that I no longer need
* * *	form teacher	find a student by name	locate details of students without having to go through the entire list
* * *	form teacher	sort students by alphabetical order	locate a student easily

Priority	As a …	I want to …	So that I can…
* * *	form teacher	update the details of my students	make necessary changes to my student's particulars
* * *	form teacher	maintain emergency contacts of my students	know who to contact in case of emergency
* *	form teacher	specify if a student is late or absent for class	know why my student is absent
* *	user	hide private contact details by default	minimize chance of someone else seeing them by accident
* *	form teacher	keep track of the sitting arrangement of the class	students who change their seats unknowingly
* *	form teacher	record the temperature of students	track the health of my students
*	form teacher	get feedback from other teachers teaching the students of my class	better understand the progress of the class

Use case: UC07 - Display admin class list.

MSS

1. User enters the command to display the administrative version of the class list.
2. TeaPet displays the class list with the students' attendance status and temperature.

   Use case ends.

Extensions

1a. Command is invalid.

1a1. TeaPet shows an error message.

Use case ends.

Use case: UC08 - Deleting a date from the list of dates.

MSS

1. User enters the command to delete a specific date from the list of dates.
2. TeaPet searches through the list of dates for the date provided by the user.
3. TeaPet removes that date from the date list.
4. TeaPet displays that the date has been deleted successfully.

Use case ends.

Extensions

1a. Command is invalid.

1a1. TeaPet shows an error message.

Use case ends.

1b. Command is in incorrect format.

1b1. TeaPet shows an error message displaying the correct format for the command.

Use case ends.

1c. Date is in incorrect format.

1c1. TeaPet shows an error message displaying the correct format for date.

Use case ends.

2a. Date provided is not in the list of dates.

2a1. TeaPet shows an error message displaying date is not found.

Use case ends.

Manual Test: Class Admin Particulars

1. Saving a date
   a. Test case: `admin save`
   Expected: Date has either been saved if today's date is not in the list of dates, or date not saved if today's date is already in the list of dates. Status message either displays that today's date has been saved or displays that current date already exists in the current list of dates respectively.

b. Other incorrect delete commands to try: `save`, `admin save DATE` where `DATE` is in YYYY-MM-DD format.
Expected: Status bar displays invalid command and incorrect command format message respectively.

2. Deleting a date

a. Test case: `admin delete DATE` where DATE is in YYYY-MM-DD format.
Expected: Date has either been deleted if the date provided exists in the list of dates, or no dates will be deleted if the date provided is not in the list. Status message either displays that the specific date has been deleted or displays that current date already exists in the current list of dates respectively.

b. Other incorrect delete commands to try: `admin delete`, `delete DATE`, `admin delete `DATE`, where `DATE` is in YYYY-MM-DD format and `` `DATE `` is in the wrong `DATE` format.
Expected: Status bar displays invalid command and incorrect command format message.