Simon Lam - Project Portfolio

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.

Summary of contributions

This section entails a summary of my specific enhancements, code contributions and other helpful increments towards the TeaPet application.

I implemented the student particulars feature, update student image feature, and part of the personal scheduler feature, which include:

Enhancements

Student particulars

Student Creation and Deletion

- **What it does:** Users are able to add details of their students into the application. Students are created by specifying their full names at the point of their creation. Additional details such as phone number, email, address, temperature and NOK details can be stored in the application as well.
- **Justification:** Teachers usually have many students to spend their attention on, and having physical copies of a whole bunch of data can be a hassle. With TeaPet, teachers can now enjoy the luxury of keeping track of all necessary details of their students with just their laptop. There may also be a possibility where a teacher no longer teaches a student. Therefore, TeaPet provides the functionality of removing students whom they no longer teach.
- **Highlights:** This enhancement allows the user to leave optional data unfilled. It might be quite difficult for teachers to immediately gather all necessary details from students such as NOK particulars. Therefore, TeaPet allows the creation of students to leave in empty optional fields. The only compulsory field is the name of the student.

Student editing

- **What it does:** Users are able to edit student details of students in the student list.

- **Justification:** There is always a possibility for teachers to key in information wrongly. Therefore, instead of removing the student and creating a new one, TeaPet provides the functionality to simply edit the details of a student within the student list.

Update Student Image

- **What it does:** Users are able to add in pictures of students into the student list.
- **Justification:** As mentioned, teachers usually have a large class of students and it may be quite challenging to remember each of their name. Hence, TeaPet provides the functionality to update images of your students. This allows teachers to identify students base of their pictures for easier recognition in classrooms, even if its their first few times meeting them.

Personal Scheduler

Event Creation

- **What it does:** Users are able to create events and add them into their personal scheduler. These events are created by specifying the event name, start time, end time.
- **Justification:** Teachers are always extremely busy and have many activities to attend. Some of which are teaching classes, meetings with teachers, meeting with parents, consultations etc. I wanted to make a feature which are able to manage the time of teachers effectively, both inside and outside of classrooms.
- **Highlights:** The events can be set to recur weekly, daily, or not at all, which reduces the hassle of re-typing in repeated commands if they have to take part in a particular event everyday. The events can also be color coded by users such that they can easily identify and categorise them in their personal schedule if it gets too packed.

Code Contributed

Please click here to see my code contributions dashboard.

Other contributions

- Project management:
  - Team lead of the project, ensuring everyone is up to date and on the same page for the tasked project.
  - In charge of defining, assigning, and tasking of project tasks.
  - Ensures project deliverables are done on time.
- Documentation:
  - Updated User Guide to include Student, Schedule and Additional features. (Pull requests #49, #216, #217, #225, #242)

    - Updated User Guide with a simple and easy to use Command Summary page.
    - Updated User Guide with a product survey: (Pull requests #62)
    - Updated Developer Guide to include Student, Schedule and Additional features. (Pull requests #245, #251)
- Community
    - PRs reviewed (with non-trivial review comments): #30, #66, #67
    - Reported bugs and suggestions for other teams in the class: (Examples: PE Dry Run)
- Tools
    - Integrated a Github plugin (Coverall) to the team project #259
    - Integrated a Github plugin (Travis) to the team project #265
    - Integrated a Github plugin (Netlify) to the team project #265
    - Integrated a third party library (iCalendarAgenda) to the project #108

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.

Navigating the User Guide

The aim of this User Guide is to provide you with all the information you need to fully utilise TeaPet. We understand the pains of using a Command Line Interface (CLI) program and have bested our efforts into ensuring a very readable guide on how to use our program.

If you need help setting up TeaPet, you can go to the [QuickStart] section.

If you want to find out more about TeaPet's features and commands, you can go to the [Features] section.

If you need an overview regarding the usage of TeaPet's commands, head on to the [CommandSummary] section.

Take note of the following symbols and formatting used in this document:

Table 1. A Summary of symbols used in our User Guide.

Symbol/ Format	Meaning
kbd:[Enter]	This symbol indicates the enter button on the keyboard.
`command`	A grey highlight indicates that this is a command that can be executed by the program. Some `[command]` will have brackets around them, this indicates that the command is optional.

NOTE	This symbol indicates notes.
WARNING	This symbol indicates warnings.
TIP	This symbol indicates tips.

Student Particulars [Done by Simon Lam]

TeaPet records down personal particulars of students such as address, contact number and Next of Kin (NOK) particulars. Thereafter, you are able to view, update or delete those information of specific students when deemed necessary.

Display default student view: `student`

TeaPet syncs the images of students found in the image folder into the student list. More information about updating student images can be found here. TeaPet then displays a summarised list of the student details.

Format: `student`

Expected Outcome:

```
The student list now displays DEFAULT details.
[HELP ON STUDENT COMMANDS]
1. Display detailed list: student detailed
2. Add student: student add name/NAME [phone/PHONE] [email/EMAIL] [adr/ADDRESS]
[temp/TEMPERATURE] [att/ATTENDANCE] [nok/NAME-RELATIONSHIP-PHONE] [tag/TAG]
3. Edit student: student edit INDEX [name/NAME] [phone/PHONE] [email/EMAIL]
[adr/ADDRESS] [temp/TEMPERATURE] [att/ATTENDANCE] [nok/NAME-RELATIONSHIP-PHONE]
[tag/TAG]
4. Delete student: student delete INDEX
5. Find student: student find NAME
```

Figure 1. After using `student` command

Display detailed student view: `student detailed`

Displays a detailed version of the class list with all information.

Format: `student detailed`

Expected Outcome:

```
The student list now displays ALL details.
```

Figure 2. After using `student detailed` command

Add Student: `add`

Adds a student into the student list.

Format:

```
student add name/NAME [phone/PHONE] [email/EMAIL] [adr/ADDRESS] [temp/TEMPERATURE]
[att/ATTENDANCE] [nok/NAME-RELATIONSHIP-PHONE] [tag/TAG]
```

- Adds a new student with the given attributes.
- The student name **cannot be empty**.

NOTE	The address of student is not restricted as it can be subjective to the student and teacher.

NOTE	Next-of-kin relationships allowed: Father, Mother, Sister, Brother, Grandfather, Grandmother

Example:

- `student add name/Jim phone/90045722 email/jim@example.com adr/Bishan St 13 Blk 154 #08-18 tag/monitor nok/James-Father-91234567 temp/36.6 att/Present`
  Adds a student named Jim into the student list along with his details.

Expected Outcome:

```
New student added: Jim Phone: 90045722 Email: jim@example.com Address: Bishan St 13
Blk 154 #08-18 Temperature: 36.6 Attendance: Present NextOfKin: James-Father-91234567
Tags: [monitor]
```

Figure 3. After using `student add` command

Edit Student: `edit`

Edits personal details of students.

Format:

```
student edit INDEX [name/NAME] [phone/PHONE] [email/EMAIL] [adr/ADDRESS]
[temp/TEMPERATURE] [att/ATTENDANCE] [nok/NAME-RELATIONSHIP-PHONE] [tag/TAG]
```

Example:

- `student edit 1 phone/90023413`

Edits the student phone number in index 1 to a new phone number.

Expected Outcome:

```
Edited Student: Simon Lam Phone: 90023413 Email: simonlam@example.com Address: Blk 30
Geylang Street 29,
#06-40 Temperature: 36.5 Attendance: Sick Remark:  Tags: [Sheares]
```

Delete Student: `delete`

Deletes the student and all his personal details from the student list.

Format:

```
student delete INDEX
```

Example:

- `student delete 1` Deletes the student at index 1.

Expected Outcome:

```
Deleted Student: Simon Lam Phone: 90023413 Email: simonlam@example.com Address: Blk 30
Geylang Street 29, #06-40 Temperature: 36.5 Attendance: Sick Remark:  Tags: [Sheares]
```

Clear Student List: `clear`

Clears all data from the student list.

Format:

```
student clear
```

Example:

```
`student clear` Deletes the entire student list
```

Expected Outcome:

```
Student list has been cleared!
```

Find Student: `find`

Finds the student information from the student list and displays it.

Format:

```
student find NAME
```

Example:

- `student find Simon` Finds the information a student named Simon.

Expected Outcome:

```
1 students listed!
```

Personal Scheduler [Done by Simon Lam and Gary Lim]

TeaPet's Personal Scheduler allows you to record down your events for the week, which will be sorted according to date and time. You will then be able to easily view your schedule as you need it.

Display your scheduler: `schedule` [Done by Simon Lam and Gary Lim]

Displays your schedule in this current week.

Format: `schedule`

Expected Outcome:

```
This is your schedule for the week
Schedule helps you to keep track of your events.
[HELP ON SCHEDULE COMMANDS]
add event: schedule add eventName/EVENT_DESCRIPTION startDateTime/YYYY-MM-DDTHH:MM
endDateTime/YYYY-MM-DDTHH:MM recur/RECUR_DESCRIPTION color/COLOR_CODE
edit event: schedule edit INDEX [eventName/EVENT_DESCRIPTION] [startDateTime/YYYY-MM-
DDTHH:MM] [endDateTime/YYYY-MM-DDTHH:MM] [recur/RECUR_DESCRIPTION] [color/COLOR_CODE]
delete event: schedule delete INDEX
get index of a event: schedule indexGet/EVENT_DESCRIPTION
get all indexes of events in schedule: schedule indexAll
change view mode of schedule: schedule view mode/SCHEDULE_MODE date/YYYY-MM-DD
export schedule: schedule export
Type the following commands for more info!
```

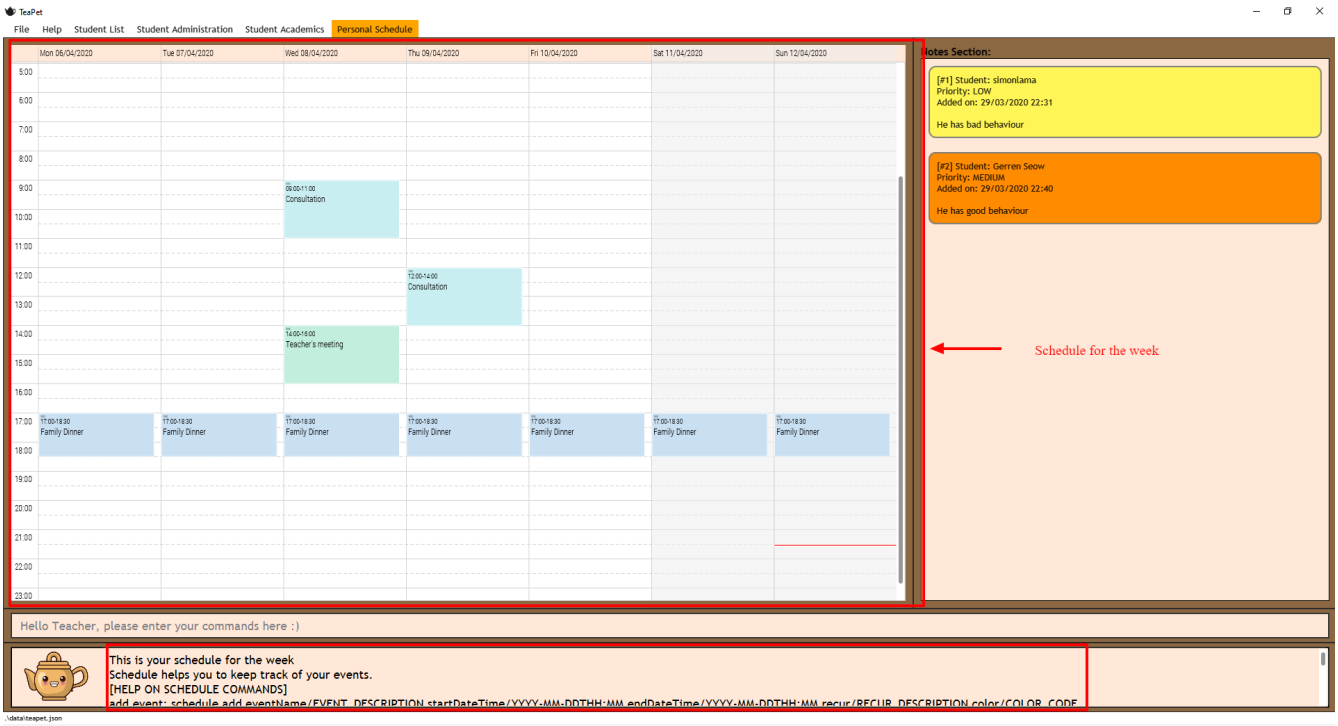

Figure 4. After using `schedule` command

Adding an event to schedule: `schedule add` [Done by Simon Lam and Gary Lim]

Adds an event to your personal scheduler.

Format: `schedule add eventName/EVENT_DESCRIPTION startDateTime/START_DATETIME endDateTime/END_DATETIME recur/RECUR_DESCRIPTION color/COLOR_CODE`

NOTE	The format of startDateTime and endDateTime is in YYYY-MM-DDThh:mm format, where time is in the 24-hour format. Example: 7th April 2020 10AM will be 2020-04-07T10:00
NOTE	RECUR_DESCRIPTION can only be either of these: `none`, `daily` or `weekly`.
NOTE	Events which are further away in the future have a darker color code. This is intentional.
NOTE	The prefixes are meant to be longer due the the emphasis on clarity as there are other features in this application which uses similar prefixes as well.
TIP	COLOR_CODE is from 0 to 23 inclusive.

Figure 5. Color code for TeaPet's calendar

Example:

- **Non-Recurring Event** `schedule add eventName/Teachers Meeting startDateTime/2020-04-02T08:00 endDateTime/2020-04-07T10:00 recur/none color/21`

Creates an event in the schedule with the description "Teachers Meeting" from "2nd Apr 2020, 0800" to "7th Apr 2020, 1000" with "no recurrence" and a color group of "21".

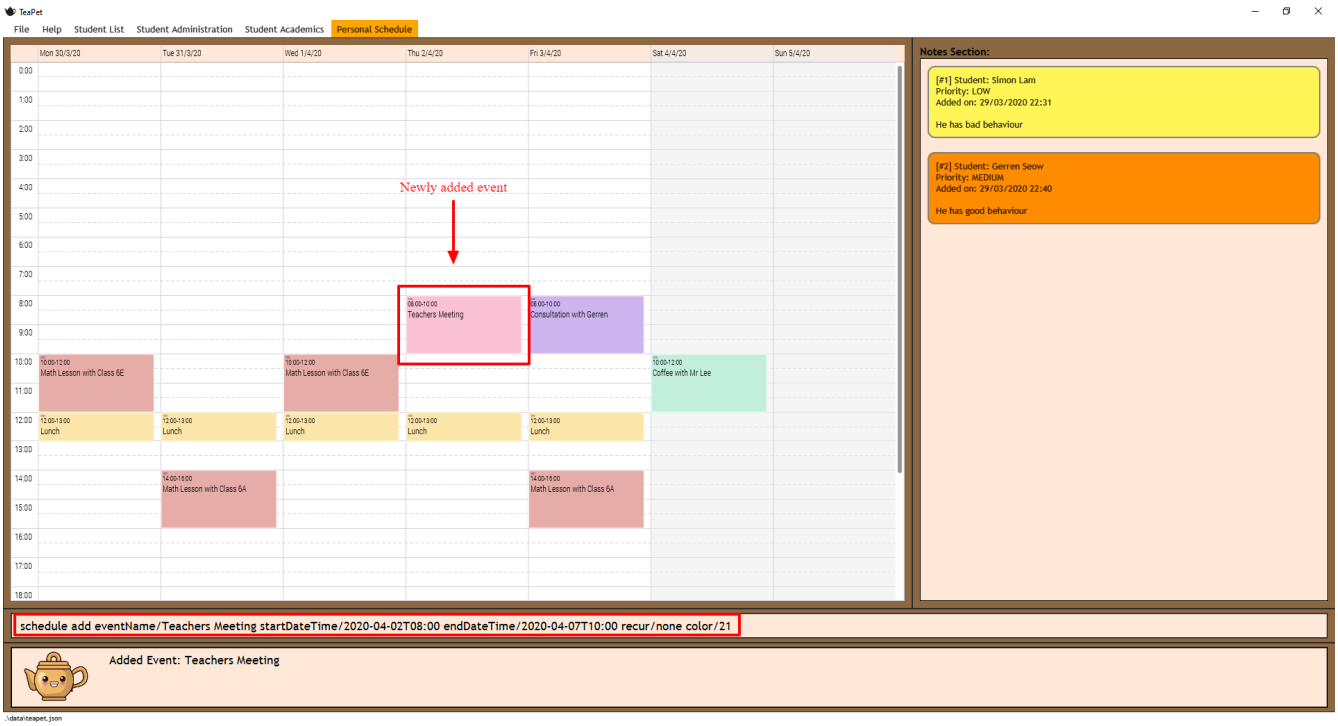

Figure 6. Adding an event to the schedule

Additional Features [Done by Simon Lam]

Update Profile Picture

TeaPet's student list allows you to upload image of your students into your application. The following steps will help you upload photos of your students into the student list.

Step 1. Locate the image folder. It is in the root directory folder!

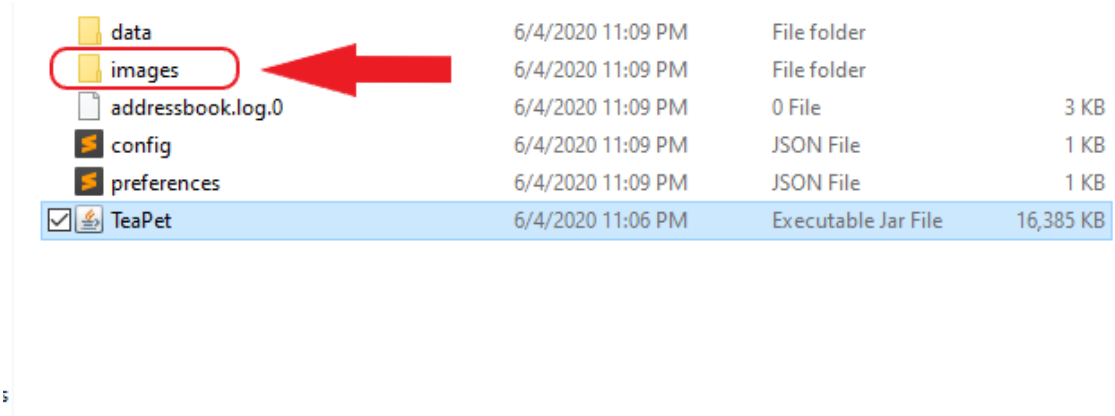

Figure 7. Location of image folder

Step 2. Open the image folder and drag the image of your student into the folder.

NOTE	The filename of your image must of this format: 1. Filename of the image must be the same as the student. 4. File is in .png format. For example, a student with name **Simon Lam** must have a image file with name **Simon Lam** in .png format.

TIP	For ideal optimization of the image, its dimensions for its length and width should be roughly equal.

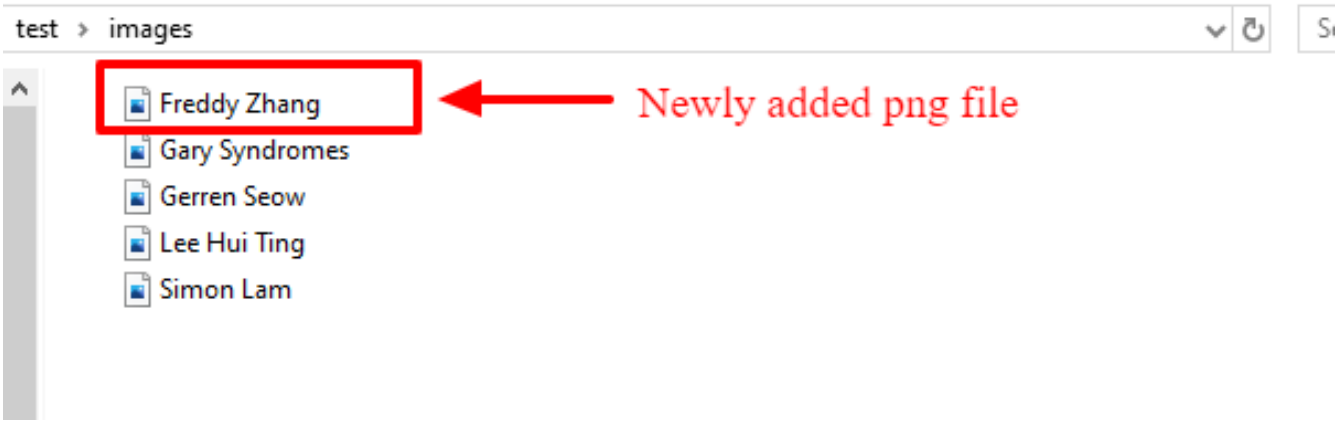

Figure 8. Dragging png file into image folder

Step 3. Type in the `student` command in the user interface. TeaPet will update the images and now you can see pictures of your students in your student list!

WARNING	Editing the name of the student will change the student image back to the default image. To solve this, you have to edit the png file in the Image folder as well after you edit the name of the student and then enter the `student` command.

Figure 9. Before using the student command

Figure 10. After using the student command

All in one help window

Suppose you are lost and you need help regarding the many commands in TeaPet, you can easily type in `help` or simply press your `F1` key to bring up this user guide!

Format: `help`

Expected Outcome:

```
Opened help window.
```

Figure 11. Displayed help window

Personal Scheduler Commands

Here are the commands to manage scheduler. They require the prefix `schedule`.

Table 2. Personal Scheduler commands of TeaPet

Command	Format	Expected outcome
`schedule edit`	`schedule edit 2 eventName/Teachers Meeting`	Edits an event in your personal scheduler.
`schedule add`	`schedule add eventName/EVENT_DESCRIPTION startDateTime/YYYY-MM-DDTHH:MM endDateTime/YYYY-MM-DDTHH:MM recur/RECUR_DESCRIPTION color/COLOR_CODE`	Adds an event into the scheduler.
`schedule delete`	`schedule delete INDEX`	Deletes an event in your personal scheduler.
`schedule indexGet`	`schedule indexGet/EVENT_DESCRIPTION`	Get the index of an event in your personal scheduler.
`schedule indexAll`	`schedule indexAll`	Get all indexes of your events in your personal scheduler.
`schedule view`	`schedule view mode/SCHEDULE_MODE date/YYYY-MM-DD`	Change the view mode of your Personal Schedule as how you need it.
`schedule export`	`schedule export`	Exports all events currently in schedule into a ".ics" file. The .ics file type can easily be used with Google Calendar and other calendar applications.

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.

Student particulars feature

The student particulars feature keeps track of essential student details. The feature comprises of commands namely,

- `StudentAddCommand` - Adds the student particulars into the class list
- `StudentEditCommand` - Edits the particulars of a student
- `StudentDeleteCommand` - Deletes the student information
- `StudentFindCommand` - Finds information of the required student
- `StudentClearCommand` - Deletes all student details from the student list

The student commands all share similar paths of execution and is illustrated in the following sequence diagram below, which shows the sequence diagram for the StudentAddCommand.

The commands when executed, will interface with the methods exposed by the `Model` interface to perform the related operations (See logic component for the general overview).

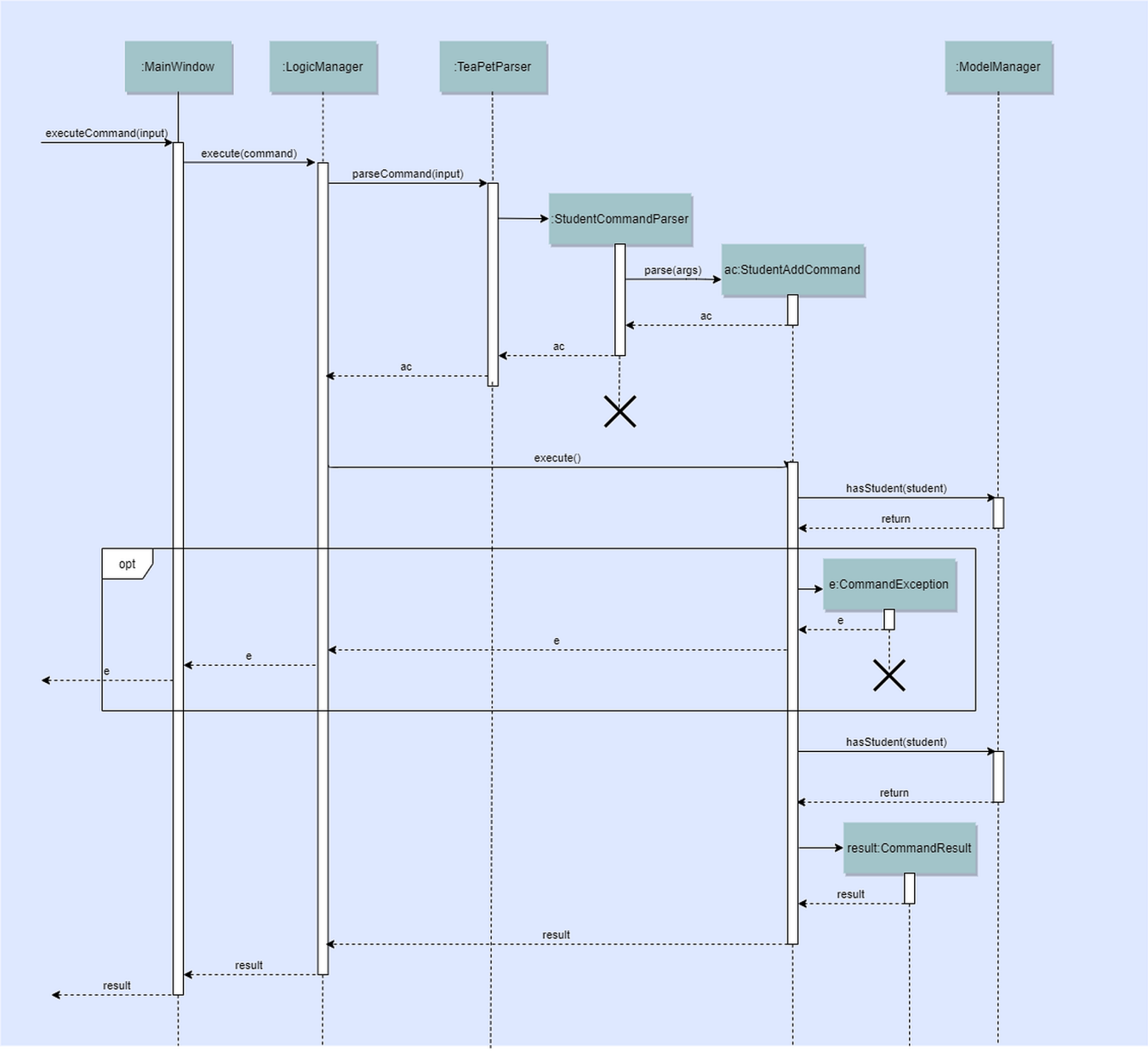

Figure 12. Sequence Diagram for StudentAddCommand

These are the common steps among the Student Commands:

1. The `TeaPetParser` will assign the `StudentCommandParser` to parse the user input.
2. The `StudentCommandParser#parse` will take in a string of user input consisting of the arguments.
3. The arguments are tokenized and the respective models of each argument are created.

Student Add command

Implementation

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

1. After the successful parsing of user input, the `StudentAddCommand#execute(Model model)` method is called which validates the student defined.

2. As student names are unique, if a duplicate student is defined, a `CommandException` is thrown which will not add the defined student.
3. The method `Model#addStudent(Student student)` will then be called to add the student. The command box will be reflected with the `StudentAddCommand#MESSAGE_SUCCESS` constant and a new `CommandResult` will be returned with the success message.

NOTE	If the format or wording of adding a student contains error(s), the behaviour of TeaPet will be that either a unknown command or wrong format error message will be displayed.

4. The newly created student is added to the `UniqueStudentList`.

The following activity diagram summarizes what happens when a user executes the `student add` command:

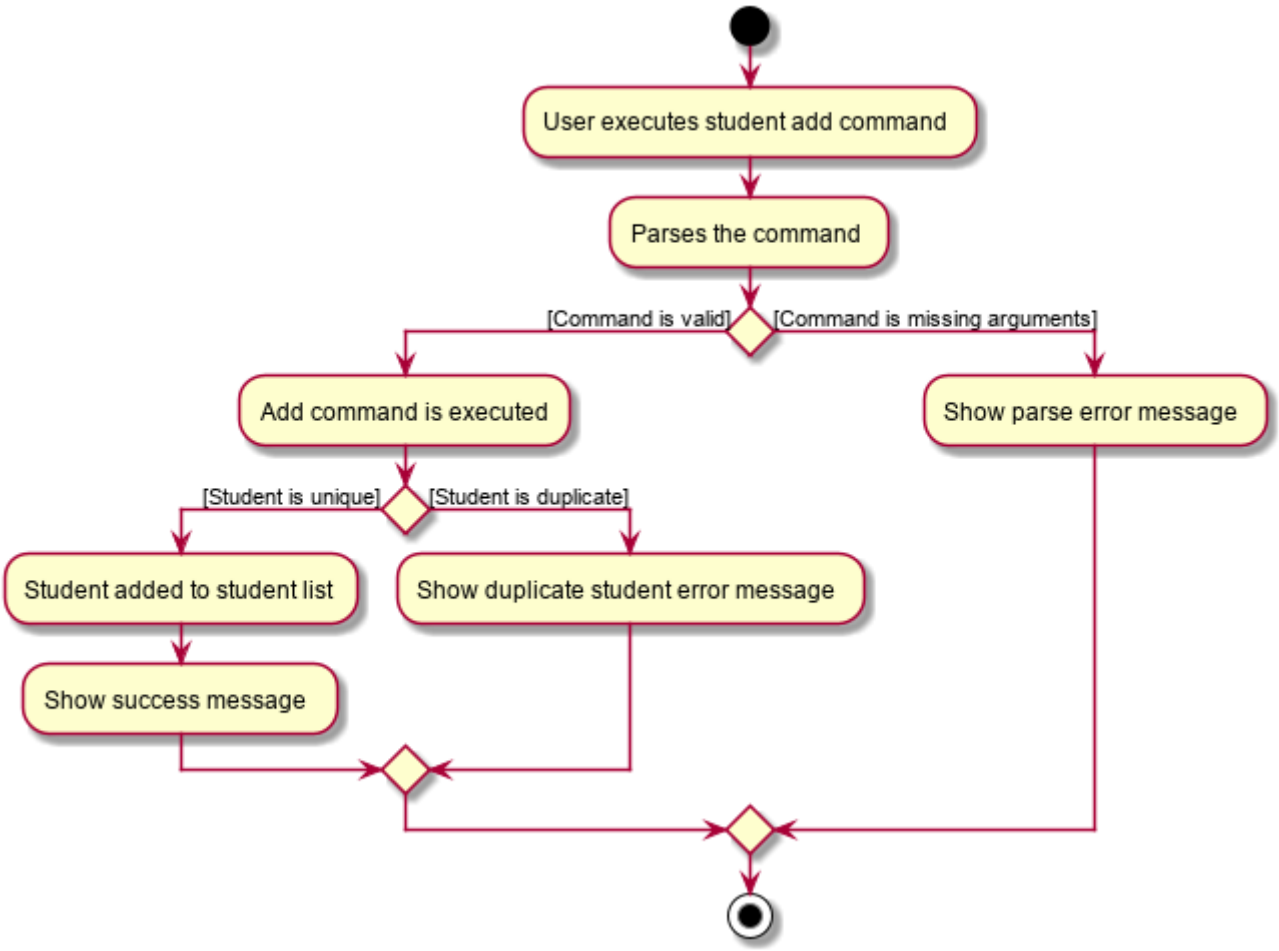

Figure 13. Activity Diagram for StudentAddCommand

Student Edit command

Implementation

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

1. After the successful parsing of user input, the `StudentEditCommand#execute(Model model)` method is called which checks if the `Index` is defined as an argument when instantiating the

`StudentEditCommand(Index, index, EditStudentDescriptor editStudentDescriptor)` constructor. It uses the `StudentEditCommand.EditStudentDescriptor` to create a new edited student.

2. A new `Student` with the newly updated values will be created which replaces the existing `Student` object using the `Model#setStudent(Student target, Student editedStudent)` method.
3. The filtered student list is then updated with the new `Student` with the `Model#updateFilteredStudentList(PREDICATE_SHOW_ALL_STUDENTS)` method.
4. The command box will be reflected with the `StudentEditCommand#MESSAGE_SUCCESS` constant and a new `CommandResult` will be returned with the success message.

Student Delete command

Implementation

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

1. After the successful parsing of user input, the `StudentDeleteCommand#execute(Model model)` method is called which checks if the `Index` is defined as an argument when instantiating the `StudentDeleteCommand(Index index)` constructor.

> **NOTE** The `Index` must be within the bounds of the student list.

2. The `Student` at the specified `Index` is then removed from the `UniqueStudentList#students` observable list using the `Model#deleteStudent(Index index)` method.
3. The command box will be reflected with the `StudentDeleteCommand#MESSAGE_SUCCESS` constant and a new `CommandResult` will be returned with the success message.

Student Find command

Implementation

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

1. After the successful parsing of user input, the `StudentFindCommand#execute(Model model)` method is called which checks if the `NameContainsKeywordsPredicate(keywords)` is defined as part of the argument when instantiating the `StudentFindCommand(NameContainsKeywordsPredicate predicate)` constructor.
2. The `Student` is then searched through the `UniqueStudentList#students` list using the `Model#hasStudent(Student student)` method to check if the `Student` already exists. If the `Student` does not exist, a `StudentNotFoundException` will be thrown and the `Student` will not be displayed.
3. The existing `UniqueStudentList#internalList` is then cleared and updated using the `Model#updateFilteredStudentList(Predicate predicate)` method.
4. A new `CommandResult` will be returned with the success message.

Student Clear command

Implementation

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

1. After the successful parsing of the user input, the `StudentClearCommand#execute(Model model)` method is called.
2. The `Model#setTeaPet(ReadOnlyTeaPet teaPet)` method is then called which triggers the `TeaPet#resetData(ReadOnlyTeaPet newData)` method and creates a brand new student list to replace the old one.
3. A new `CommandResult` will be returned with the success message.

Design Considerations

Aspect: Command Syntax

- **Current Implementation:**
  - Current implementation of the feature follows just the command word syntax For example, `student`.
- **Alternatives Considered:**
  - We considered using the forward slash `/` before the command word, for example `/add`. However, we realise that it is redundant and will make inputs more tedious and confusing for users.

Aspect: Command Length:

- **Current Implementation:**
  - Commands are shortened as much as possible without much loss in clarity. For example, instead of using `/temperature`, we used `/temp` instead to input the students temperature into the application. Although this may be initially unfamiliar to users, it should be easy to pick up and make it less tedious during input.
- **Alternatives Considered:**
  - We considered using more descriptive arguments such that arguments are clear and succinct. However, this will definitely decrease the user expereince as the command will be too long to type.

Import image feature

This feature was included in TeaPet to help teachers easily identify the students using their pictures instead of just names. This feature utilises the `StudentCard#updateImage` method to update the images of students.

The feature comprises of one command namely, `DefaultStudentDisplayCommand`

- Updates the student list to show updated images of students and displays the student list.

This is further illustrated in the following sequence diagram below.

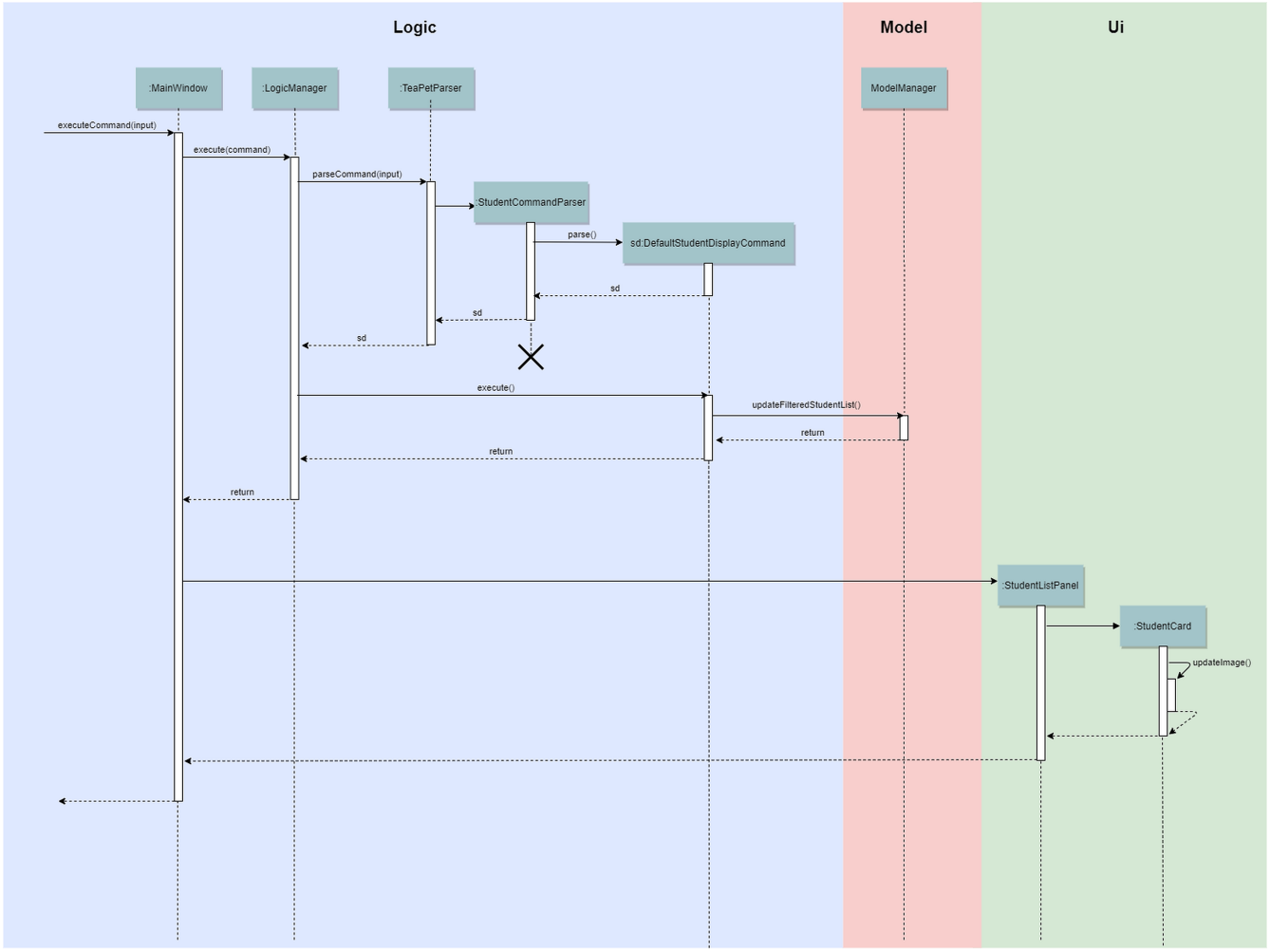

Figure 14. Sequence Diagram for `DefaultStudentDisplayCommand`

Implementation

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

1. After the successful parsing of user input, the `DefaultStudentDisplayCommand#execute(Model model)` method is called. It does not require validation as it does not write into the student list.
2. The `StudentCardDefault#updateImage` method is then called which checks the image folder for the required png file and updates the student card.

> **NOTE** The name of the png file must match the name of the student.

3. If any view other than the student list view is showing on the `MainWindow`, the `MainWindow#handleDefaultStudent()` method will be called and the student list is now visible on the `MainWindow`.

The following activity diagram summarizes what happens when a user executes the `student` command:

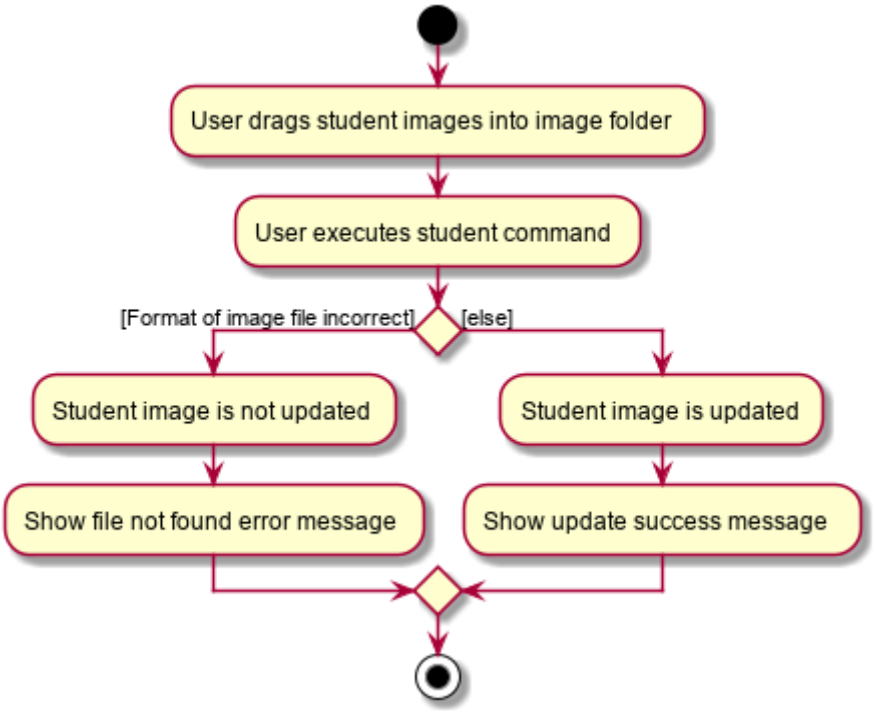

Figure 15. Activity Diagram Import Image Feature

Design Considerations

Aspect: Command Syntax

- **Current Implementation:**
  - Current implementation of the commands follows the command word syntax, followed by the arguments necessary for execution. For example, `student add/edit/delete/find`.
- **Alternatives Considered:**
  - We considered using a whole new command, `student refresh` to solely refresh and update images of the students. However, we realised that it would be more convenient for the user if we just add this functionality into the `student` command instead as it is able to both update the images and display the student list concurrently.

Use case: UC01 - Add student

MSS

1. User enters a student name, followed by optional attributes such as emergency contacts, through the command line.
2. TeaPet adds the student and his/her attributes to the class list.
3. TeaPet displays feedback to the user that a new student is being added.

   Use case ends.

Extensions

1a. Student is invalid.

1a1. TeaPet shows an error message.

Use case ends.

1b. Student particulars keyed in by user are invalid.

1b1. TeaPet shows an error message.

Use case ends.

Use case: UC02 - Edit student

MSS

1. User specifies which student, using the name, and what particulars he/she wants to edit in the command line.
2. TeaPet edits the student's particulars in the class list as instructed by the commands.
3. TeaPet displays feedback to the user that the student has been edited, followed by the changes made.

Use case ends.

Extensions

1a. Student is invalid.

1a1. TeaPet shows an error message.

Use case ends.

1b. Student particulars keyed in by user are invalid.

1b1. TeaPet shows an error message.

Use case ends.

Use case: UC03 - Delete student

MSS

1. User specifies which student, using the index, he/she wants to remove.
2. TeaPet removes the student from the class list.
3. TeaPet displays feedback to the user that the student is being removed.

Use case ends.

Extensions

1a. Student index entered by user is invalid.

1a1. TeaPet shows an error message.

Use case ends.

Use case: UC04 - Display Schedule

MSS

1. User keys in the command to display events.
2. TeaPet displays the events in chronological order.

Use case ends.

Extensions

1a. Command is invalid.

1a1. TeaPet shows an error message.

Use case ends.

Use case: UC05 - Display student class list.

MSS

1. User enters the command to display the class list.
2. TeaPet displays the class list with the students' tags, mobile number, email, and notes.

Use case ends.

Extensions

1a. Command is invalid.

1a1. TeaPet shows an error message.

Use case ends.

Use case: UC18 - Add event into schedule

MSS

1. Teacher wishes to add an event into the scheduler
2. Teacher enters the event details.

3. TeaPet saves the item and displays it on the scheduler.

   Use case ends.

Extensions

   2a. Item is missing details

   - 2a1. Teapet displays an error message.
     Use case resumes at step 2.

   Use case ends.

Use case: UC19 - Delete event from schedule

Preconditions 1. Event exists in scheduler.

MSS

1. Teacher lists the events in calendar (UC12)
2. Teacher wishes to delete an event.
3. Teacher confirms the deletion.
4. TeaPet deletes the event from the scheduler.

   Use case ends.

Extensions

   2a. Teacher reconsiders and chooses not to remove the event.

   Use case ends.

Use case: UC20 - Update student profile picture

Preconditions 1. Png files in image folder is in correct format.

MSS

1. Teacher wants to update image of students.
2. Teacher adds in the respective images into the image folder.
3. TeaPet confirms the process.
4. TeaPet updates the profile pictures of students in the student list.

   Use case ends.

Manual Test: Student particulars

1. Adding a student from class list with the specific name entered by user.
    a. Test case: `student add name/John Tan Jun Wei phone/83391329 email/john@gmail.com temp/36.0 adr/Punggol Street 22 nok/James-father-93259589`
    Expected: Student John Tan Jun Wei has been added to the class list.
    b. Test case: `student add name/John Tan Jun Wei phone/83393129 email/john@gmail.com temp/3@.5 adr/Punggol Street 22 nok/James-father-93259589`
    Expected: No student is added. Error details shown in the status message. Status bar remains the same.
    c. Test case: `student add name/John Tan Jun Wei phone/839 email/john@gmail.com temp/36.0 adr/Punggol Street 22 nok/James-father-93259589`
    Expected: An error message is shown as the phone number of student is invalid.
    d. Test case: `student add name/John Tan Jun Wei phone/83391329 email/john@gmail.com temp/36.0 adr/Punggol Street 22 nok/James-dad-93259589`
    Expected: An error message is shown as the relationship of NOK is not recognised.
    e. Test case: `student add name/John333 phone/83391329 email/john@gmail.com temp/36.0 adr/Punggol Street 22 nok/James-father-93259589`
    Expected: An error message is shown as the name of student cannot contain numbers.
2. Deleting a student from class list with the specific name entered by user.
    a. Test case: `student delete 1`
    Expected: The student at the first index is deleted from the list. Status message displays that the specified student has been deleted.
    b. Test case: `delete Tan John Wei Jun`
    Expected: No student is deleted. Error details shown in the status message. Status bar remains the same.
    c. Other incorrect delete commands to try: `delete`, `delete 10` (where the specified student is not a student in the class list due to the index being out of bounds.) *{give more}*
    Expected: Similar to test case b. === Manual Test: Scheduler
3. Adding an event to the scheduler
    a. Prerequisites: The scheduler already contain an event with name "Coffee Break", startDateTime "2020-04-04T12:00", endDateTime "2020-04-04T13:00". The recurrence type and color do not matter as long as they are valid.
    b. Test case: `schedule add eventName/Consultation startDateTime/2020-04-10T10:00 endDateTime/2020-04-10T12:00 recur/none color/5`
    Expected: An event with name Consultation is added to the scheduler.
    c. Test case: `schedule add eventName/Coffee Break startDateTime/2020-04-04T12:00 endDateTime/2020-04-04T13:00 recur/none color/5`
    Expected: An error message is shown due to duplicate events being created.
    d. Test case: `schedule add eventName/ `
    Expected: An error message is shown due to invalid command.
    e. Test case: `schedule add eventName/Consultation startDateTime/2020-04-10T10:00 endDateTime/2020-04-10T12:00 recur/none color/24`
    Expected: An error message is shown due to an invalid color code.

f. Test case: `schedule add eventName/Consultation startDateTime/2020-04-10T10:00 endDateTime/2020-04-10T12:00 recur/fortnightly color/5` Expected: An error message is shown due to invalid recurrence type.

g. Test case: `schedule add eventName/Consultation startDateTime/2020-04-10T13:00 endDateTime/2020-04-10T12:00 recur/none color/5` Expected: An error message is shown due to the invalid date time range.