Ang Wei Heng Kendrick - Project Portfolio

PROJECT: Scheduler

Overview

People who hold leadership positions, especially those in student-led clubs, have to usually schedule interviews with those interested in applying to leadership positions during recruitment period. Usually, there are clashing preferred time slots between interviewees, which the interviewer has to slowly and manually resolve. We therefore came up with Scheduler, a desktop application used to automate and streamline the process of interview scheduling. Scheduler matches interviewees with interview time slots to generate an optimal interview timetable. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 20 kLoC.

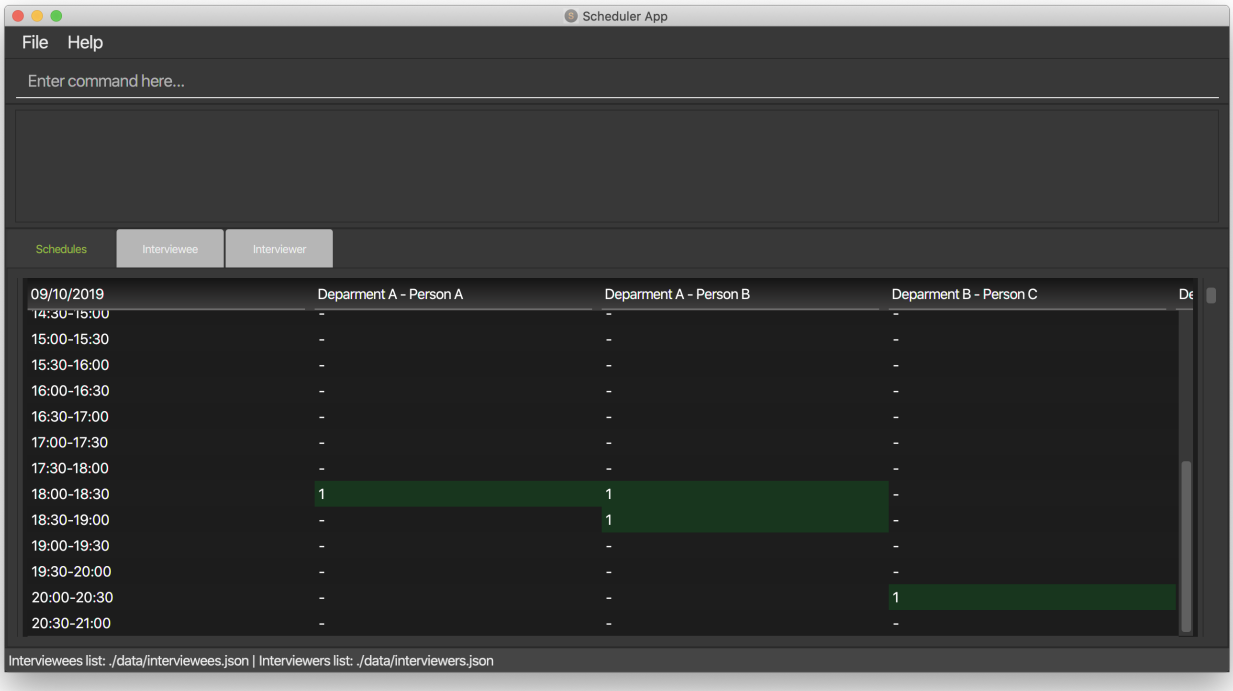

Figure 1. Graphical user interface of Scheduler

Summary of contributions

- **Major enhancement**: added the **ability to manually `add`, `edit` and `delete` entities (interviewees and interviewers)** in Scheduler.
    - What it does: This allows the user to manually alter the data currently in Scheduler .
    - Justification: This feature is necessary to the product as interview schedules change

frequently - interviewees may suddenly be unable to attend an interview, withdraw from the interview. Interviewers may also be forced to withdraw due to urgent business. There may also be more applicants, and hence interviewees. In these cases, the ability to manually change data in Scheduler is essential.

    - Highlights:
        - This enhancement deeply affects the foundational structure of Scheduler in `Logic` and `Model`.
        - It required an in-depth understanding of the existing code base, as well as other concepts like regex expressions to ensure input validity and the Jackson library to convert objects to JSON format.
        - The implementation was also challenging as it required changes to existing commands and deletion of irrelevant fields. The large number of possible input arguments for these commands (e.g the `add` command has 8 possible input prefixes) also each required extensive testing to ensure users could not input invalid values.
    - Credits: Joshua Bloch, author of Effective Java, for his ideas on Builder patterns as a way to generate objects with optional parameters.
    - Relevant pull requests: #46, #60, #66, #86, #90, #102, #156
- **Minor enhancement**: Added the `clear` command, allowing the user to clear all entities and schedules from Scheduler's internal model.
- **Code contributed**: [View on RepoSense]
- **Other contributions**:
    - Project management:
        - Contributed non-trivially to pull request code discussion: (pull requests #90, #158)
        - Coordinated assignment of bugs after mock practical exam: (github issues page)
    - Enhancements to existing features:
        - Wrote additional tests for existing features to increase coverage to above 75%: (pull requests #148, #154)
    - Documentation:
        - Did cosmetic and structural tweaks to existing contents of the User Guide to improve readability: (pull request #156)
    - Community:
        - Contributed to forum discussion: (example 1)

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.

Add interviewees/interviewers manually `add`

Manually add a new entity to the database. The command format differs on the preamble supplied, which should be `interviewee` or `interviewer`:

- If `interviewee` is supplied, refer to Adding a new interviewee below.
- If `interviewer` is supplied, refer to Adding a new interviewer below.

Adding a new interviewee

Manually add a new interviewee to the database.

Format: `add interviewee n/NAME p/PHONE f/FACULTY y/YEAR_OF_STUDY ep/PERSONAL_EMAIL ew/NUS_WORK_EMAIL d/DEPARTMENT… s/SLOT… [t/TAG]…`

Example:

```
add interviewee n/John Doe p/91234567 f/School of Computing y/1 ep/john_doe@gmail.com
ew/john_doe@u.nus.edu d/Marketing s/17/10/2019 13:30-14:00 t/friends
```

Notes:

- If a name prefix `n/` is supplied, the `NAME` given should not match (case-insensitive) that of other interviewees already in Scheduler.
    - `John Doit` and `John Do it` are considered different names.

Adding a new interviewer

Manually add a new interviewer to the database.

Format: `add interviewer n/NAME p/PHONE ew/NUS_WORK_EMAIL d/DEPARTMENT s/SLOT… [t/TAG]…`

Example:

```
add interviewer n/Mary Jane p/98765432 ew/mary_jane@u.nus.edu d/Marketing s/20/09/2019
18:00-18:30
```

Notes:

- If a name prefix `n/` is supplied, the `NAME` given should not match (case-insensitive) that of other interviewers already in Scheduler.
    - `John Doit` and `John Do it` are considered different names.

Constraints:

- A `n/NAME` must only contain alphanumeric characters and spaces, and should not be blank.
- A `p/PHONE` must only contain numbers, and should be at least 3 digits long.
- A `f/FACULTY` can take any value in English, but should not be blank.
- A `y/YEAR_OF_STUDY` must be a positive integer within the range [1,5].
- A `ep/PERSONAL_EMAIL` or `ew/NUS_WORK_EMAIL` must be of format local-part@domain.
  - The local-part must only contain alphanumeric characters and special characters (!#$%&'*+/=?`{|}~^.-), excluding the parenthesis.
  - The domain name must at least be 2 characters long, start and end with alphanumeric characters and consist of alphanumeric characters, with a period or a hyphen for the characters in-between, if any.
- A `d/DEPARTMENT` should be in English, and should not be blank.
- A `t/TAG` must only be alphanumeric and one word, i.e `t/catLover` is correct while `t/cat lover` is not.
- `s/SLOT` must follow the format: `dd/MM/yyyy HH:mm-HH:mm`, where **dd/MM/yyyy** refers to a **date**, the leftmost **HH:mm** refers to the **start duration** and the rightmost **HH:mm** refers to the **end duration**.
- The date **dd/MM/yyyy** must strictly be a valid date, i.e 30/02/2019 cannot be supplied as a date.
- The **start duration** and **end duration** of a `s/SLOT` must follow these constraints:
  - A **start duration** and **end duration** must be in 24-hour format.
  - The range of **start durations** and **end durations** available for input must be within Working Hours.
  - The **start duration** must be earlier than the **end duration**, and be in increments of `duration`. The time elapsed from the **start duration** to **end duration** must also follow the number of minutes as specified by the value of `duration` in User Preferences. Otherwise, scheduled data will not be displayed properly in the user interface. See Duration of Timeslot for more details.

Edit interviewees/interviewers manually `edit`

Manually edit an entity in the database. The command format depends on the `r/ROLE` prefix supplied:

- If `interviewee` is supplied, refer to Edit an existing interviewee below.
- If `interviewer` is supplied, refer to Edit an existing interviewer below.

Edit an existing interviewee

Manually edit an existing interviewee in the database.

Format: `edit NAME r/interviewee [n/NAME] [p/PHONE] [f/FACULTY] [y/YEAR_OF_STUDY] [ep/PERSONAL_EMAIL] [ew/NUS_WORK_EMAIL] [d/DEPARTMENT]… [s/SLOT]… [t/TAG]…`

Notes:

- An empty tag prefix `t/` can be supplied to reset the tags of an interviewee.
- If a department prefix `d/` is supplied, at least one valid, non-empty department must be provided.
- If a slot prefix `s/` is supplied, at least one valid, non-empty slot must be provided.
- If a name prefix `n/` is supplied, the `NAME` given should not match (case-insensitive) that of other interviewees.
    - `John Doit` and `John Do it` are considered different names.

Example:

```
edit John Doe r/interviewee p/91234567 f/School of Computing s/05/11/2019 18:30-19:00
```

Edit an existing interviewer

Manually edit an existing interviewer in the database.

Format: `edit NAME r/interviewer [n/NAME] [p/PHONE] [t/TAG]… [d/DEPARTMENT] [ew/NUS_WORK_EMAIL] [s/SLOT]…`

Notes:

- An empty tag prefix `t/` can be supplied to reset the tags of an interviewer.
- If a slot prefix `s/` is supplied, at least one valid, non-empty slot must be provided.
- If a name prefix `n/` is supplied, the `NAME` given should not match (case-insensitive) that of other interviewers.
    - `John Doit` and `John Do it` are considered different names.

Example:

```
edit John Doe r/interviewer p/91234567 s/05/11/2019 18:30-19:00
```

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.

Logic component

This section shows how the `Logic` component is structured. The class diagram below shows an overview of the `Logic` component:

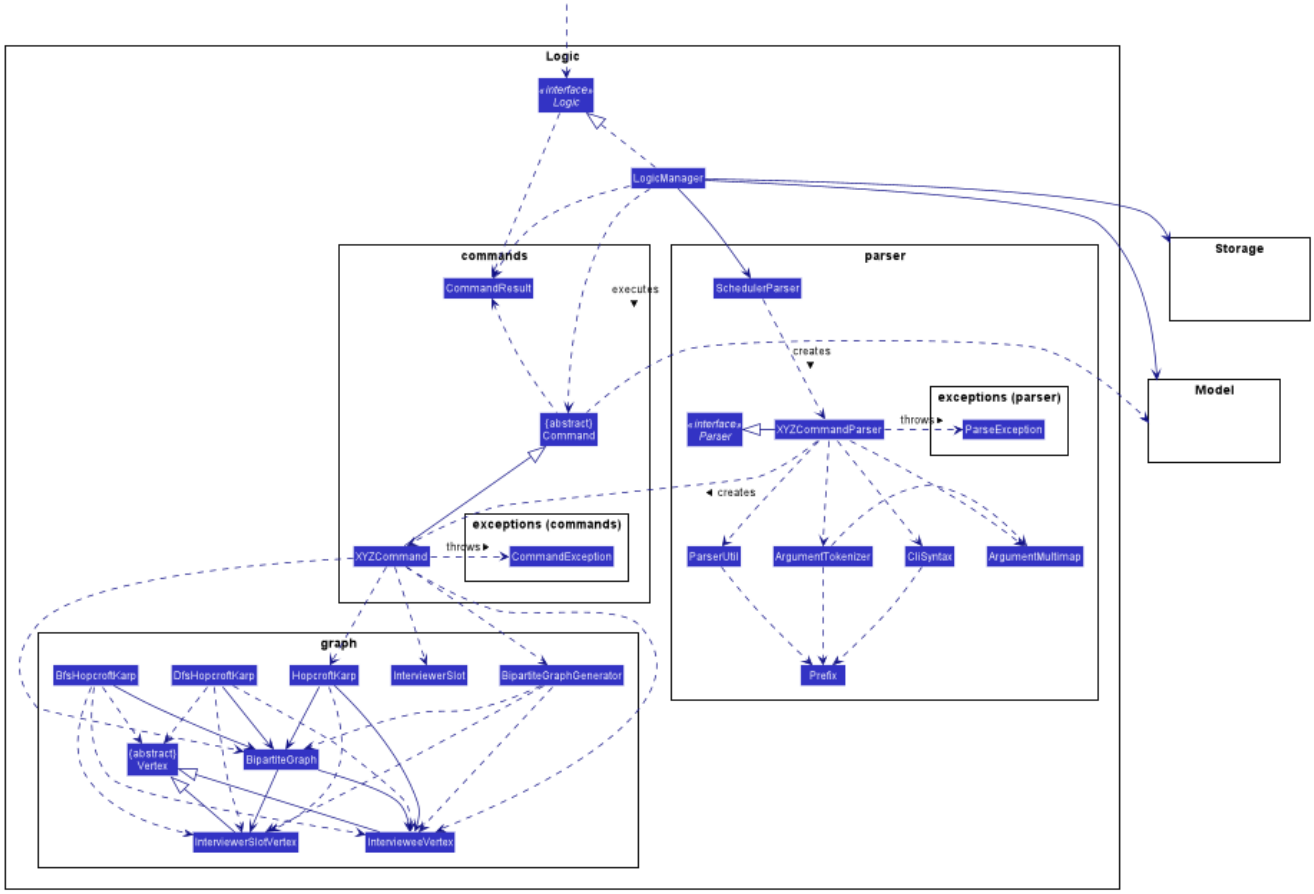

Figure 2. Structure of the Logic Component

- `XYZ` in `XYZCommand` can either be: Add, Edit, Delete, Clear, Exit, Help, Import, Export, Email, Schedule or Display.
- `XYZ` in `XYZCommandParser` can either be Add, Edit, Delete, Import, Export, Email, Schedule or Display.
- The dependencies of `XYZCommand` with the `graph` package shown in the Class Diagram only applies to `ScheduleCommand`.

API : `Logic.java`

The `Logic` component mainly handles the parsing of user input and creation of commands to execute, which in turn affects the `Model` and `Storage` components.

A command entered by the user is processed as follows:

1. `Logic` uses the `SchedulerParser` class to parse the user command, producing a `Command` object.
2. This `Command` object is executed by `LogicManager`, which may affect `Model` (e.g adding an interviewee).
3. The execution produces a result, which is encapsulated in a `CommandResult` object and returned to `LogicManager`.
4. `LogicManager` the processes the `CommandResult`. The `CommandResult` can instruct to `Ui` to perform certain actions, such as displaying help to the user.
5. Finally, `LogicManager` calls the `Storage` component to save the current `Model` to the hard disk.

Given below is the general Sequence Diagram for interactions within the `Logic` component for the execution of the `add`, `edit` and `delete` API calls. Refer to Section 3: Implementation for how different `Command` objects are executed and how they influence other components.

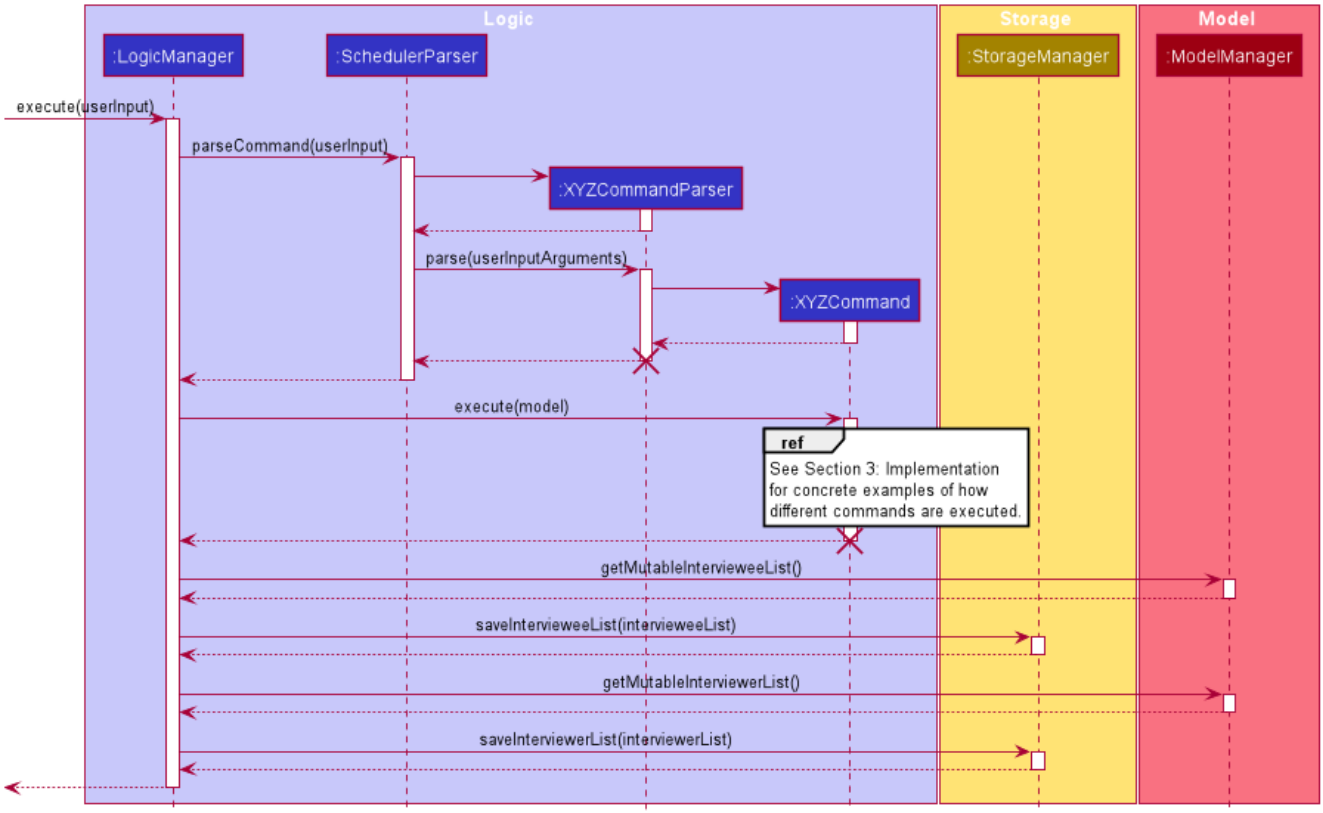

Figure 3. General interactions Inside the Logic Component for the `add`, `edit` and `delete` Commands

NOTE	The lifeline for `XYZCommandParser` and `XYZCommand` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

Add, Edit and Delete feature

The `add`, `edit` and `delete` features allow interviewees and interviewers to be manually added to Scheduler.

The flow of logic for a successful execution of these commands can be summarised as follows:

1. Parse user input and populate an `ArgumentMultimap` instance with a mapping of prefixes (from user input) to String arguments following those prefixes.
2. Check the validity of these arguments using the `ParserUtil` class.
3. If the arguments are valid, carry out the necessary modifications to the `ModelManager` class, which contains the underlying `IntervieweeList` and `InterviewerList` , which we store our entities in.

The Activity Diagram below further illustrates the general flow of logic from command input (by a user) to command execution:

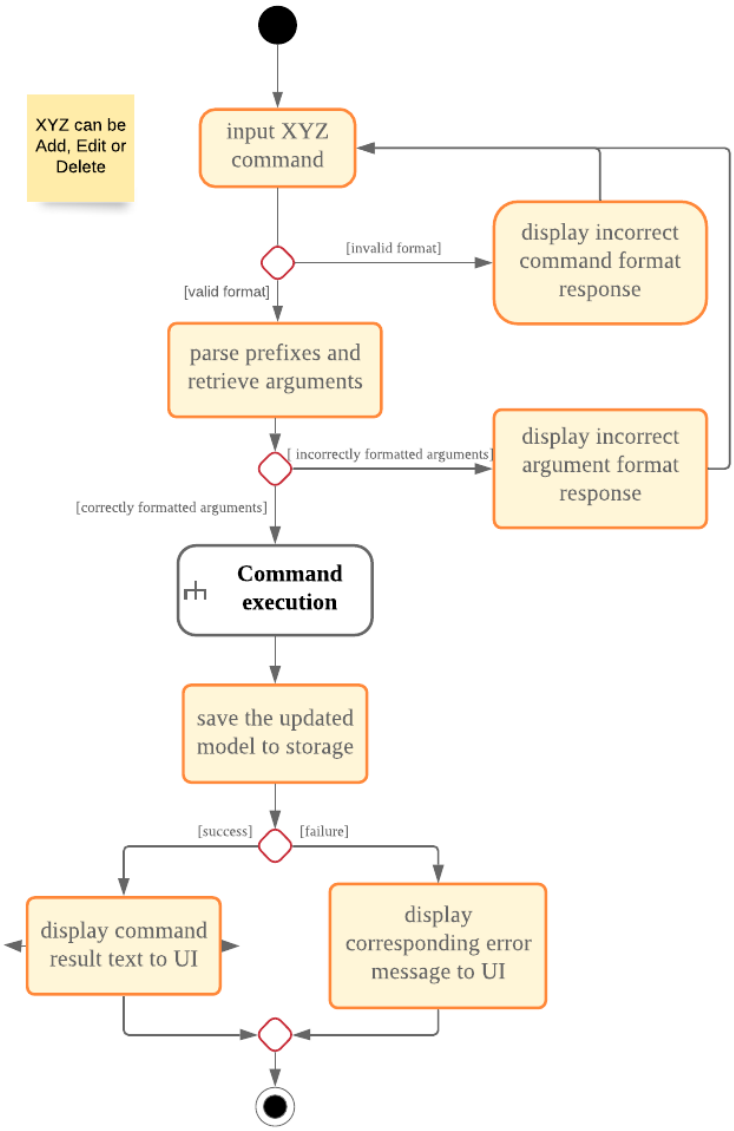

Figure 4. A general Activity Diagram illustrating logic flow in the `Logic` component.

NOTE	The box labelled "Command execution" is a **rake**. It indicates that part of the activity is given as a separate diagram. Each of add, edit and delete provide their own versions of "Command execution".

Add Interviewee/Interviewer feature

The `add` command feature allows a user to add an `Interviewee` or `Interviewer` object to the underlying `IntervieweeList` or `InterviewerList` of `ModelManager`.

1. Upon successful parsing of the `add` command arguments, an instance of `Interviewee`/`Interviewer` is created. (depending on the preamble supplied in the user input)
    a. I.e: `add interviewee n/John Doe`. The preamble starts after the command word "add" and before the first prefix "n/".
2. ModelManager#addInterviewee() is then called to add the entity to its corresponding list in `ModelManager`.
    a. The underlying `UniquePersonList` of `IntervieweeList` and `InterviewerList` ensures that no duplicate entities are present at any time, checked by Interviewee#isSamePerson() and Interviewer#isSamePerson().

The following **Sequence Diagram** illustrates how the `add interviewee` command works. The `add interviewer` command works in a similar manner.

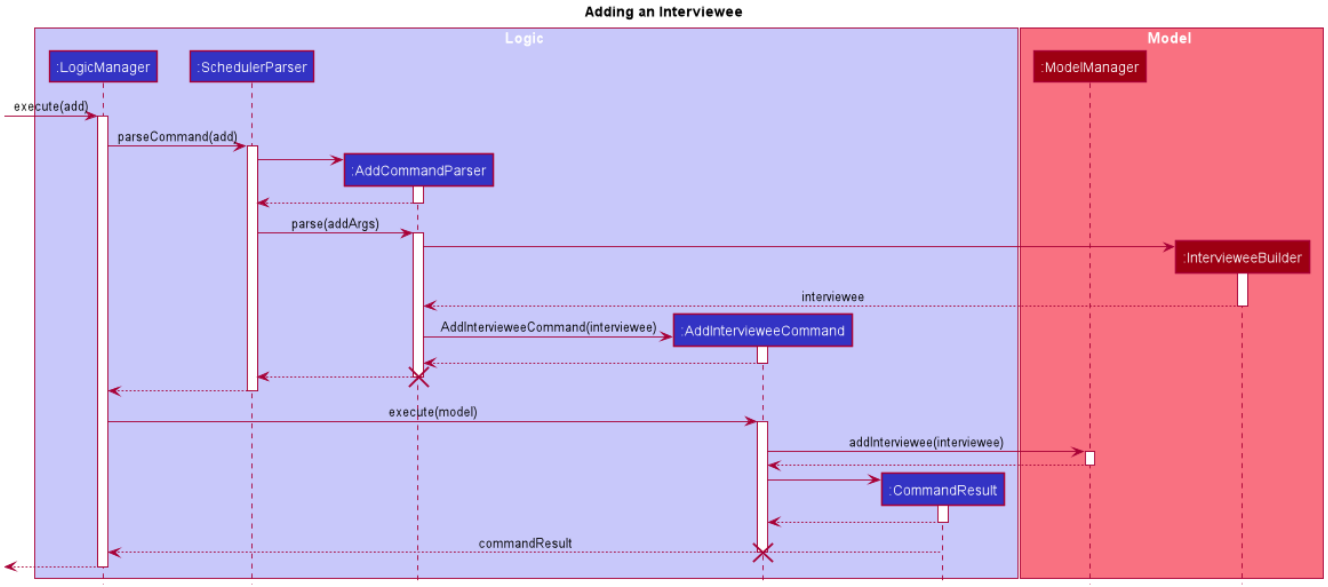

Figure 5. Add Interviewee Sequence Diagram

NOTE	The lifeline for `AddCommandParser` and `AddIntervieweeCommand` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifeline reaches the end of diagram.

- See the general logic sequence diagram in Section 2.3: Logic component for a more complete view of how `Storage` is affected as well.
- For brevity, the input arguments to functions referenced in this guide may be omitted.

To better illustrate the flow of events from the moment a user inputs an add command till completion of the command, the continuation of the **rake** from the general activity diagram is shown below:

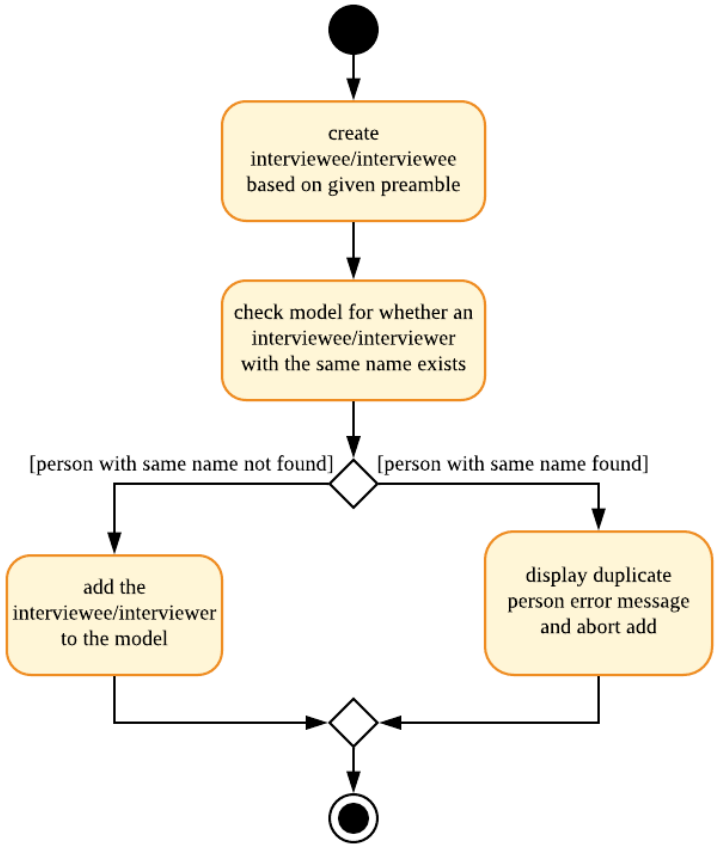

Figure 6. Activity diagram snippet of the add command