

Lim Yi Jie - Project Portfolio for TimeBook

About the project

Our team of 5 members were tasked to either enhance or morph an existing command line interface desktop address book application for our Software Engineering project. We chose to morph the original application into a group scheduling application called TimeBook. TimeBook aims to help busy NUS undergraduates keep track of the schedules of the user and his or her friends, and aid the user to arrange meetings with his or her friends.

This is a screenshot of our application.

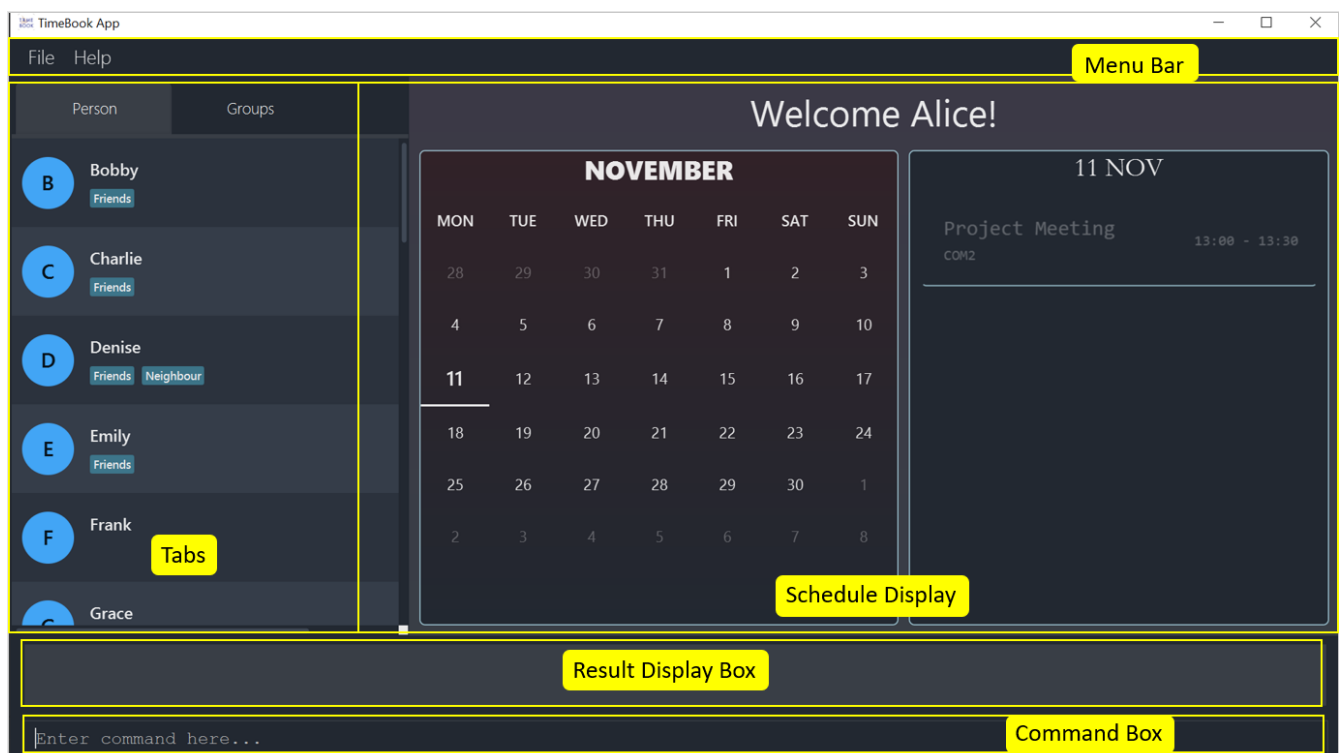



Figure 1. GUI for TimeBook

The problem assigned to me was that it is tedious and difficult to identify common free time slots among NUS undergraduates by inspecting separate NUSMods schedules especially when the group size is large. Therefore, my role was to design and write codes to display the combined schedules of NUS undergraduates from a group in the GUI elegantly and intuitively. In the following sections, my contribution and the relevant documentations that I have made will be elaborated in more detail.

Note the common symbols and formatting used in this document.

	Important information.
show	A grey highlight indicates that this is a command that can be executed in the command line
bluetext	A blue text with grey highlight indicates that this is a class or component in TimeBook.

Overview of contributions

This section provides an overview of the code, documentation and administrative contributions to the team project

Main feature added: The schedule graphic of multiple schedules in TimeBook.

- What: Enables users to see an elegant individual or group 7-day schedule graphic in the form of a time table in TimeBook. The `show` command does this.
- Justification: This is the key feature of TimeBook as it directly addresses the problem of having to tediously cross reference every separate schedule in order to identify free time slots for group meetings.
- Highlights: This feature that I have implemented fits well with the other features that my teammates developed. For example, one of my teammates developed an NUSMods parser that makes use of parsing NUSMods links into data that is then displayed using my feature. Implementing this feature is not easy as I had to first create an empty time table and figure out a way to add events (represented by blocks) into this empty time table without distorting the other contents in the time table. In addition, I had to design the graphic such that resizing the application or adjusting the screen resolution will not distort the time table. Finally, I had to test my feature using TestFX to make sure that the graphic generated is correct.
- Credits: TestFX for providing a framework for automated GUI testing.

Additional features that enhances the schedule graphic shown in TimeBook.

1. Scroll the schedule graphic when it does not fit the user's screen through the Command Line Interface (CLI). This is done through the `scroll` command.
2. Toggle the schedule graphic to show the graphics for subsequent week's schedule. This is done through the `togglenext` command.
3. Look at only schedules belonging to some members of a group. This is done through the `lookat` command.

Other contributions:

- Application
 - Developed GUI color scheme and UI layout such as schedule window display, tabs panel and default home page to make significant cosmetic improvement to the application. [#86](#) [#93](#) [#125](#) [#254](#)
 - Wrote test cases for the application itself and its GUI that increased coverage by about 5%. [#263](#) [#268](#) [#274](#)
- Community
 - Helped to develop non-trivial UI components for my team members. [#237](#)
- Documentation
 - Made user guide and developer guide more reader-friendly.
- Tools

- Set up TestFX for GUI Testing. [#206](#)
- Created a google spreadsheet for the team to share bugs the they have found.

Contributions to User Guide

Our team had to override the original addressbook documentation with the instructions for TimeBook. This section shows my contributions to the User Guide and my ability to convey instructions for the application that my team has developed to target users. I included the sections **Show person: show** and **Show group: show** in the User Guide, and I decided to omit **Show person: show** in this section as these two parts in the User Guide are very similar.

The documentation I wrote for **Show person: show** can still be accessed [here](#).

Show group: show

Want to schedule your next group meeting? You can view a group's schedule up to 4 weeks in advance.

Format: **show g/GROUP_NAME**

- Shows the description, members and aggregated schedules for this group.

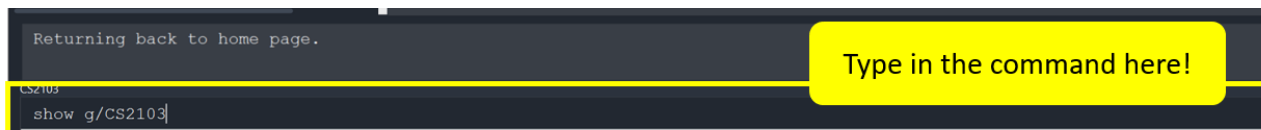
IMPORTANT

When you first start using TimeBook, it will create a hidden person called **Alice** to store your details and schedule. This profile will be a member of every group you create.

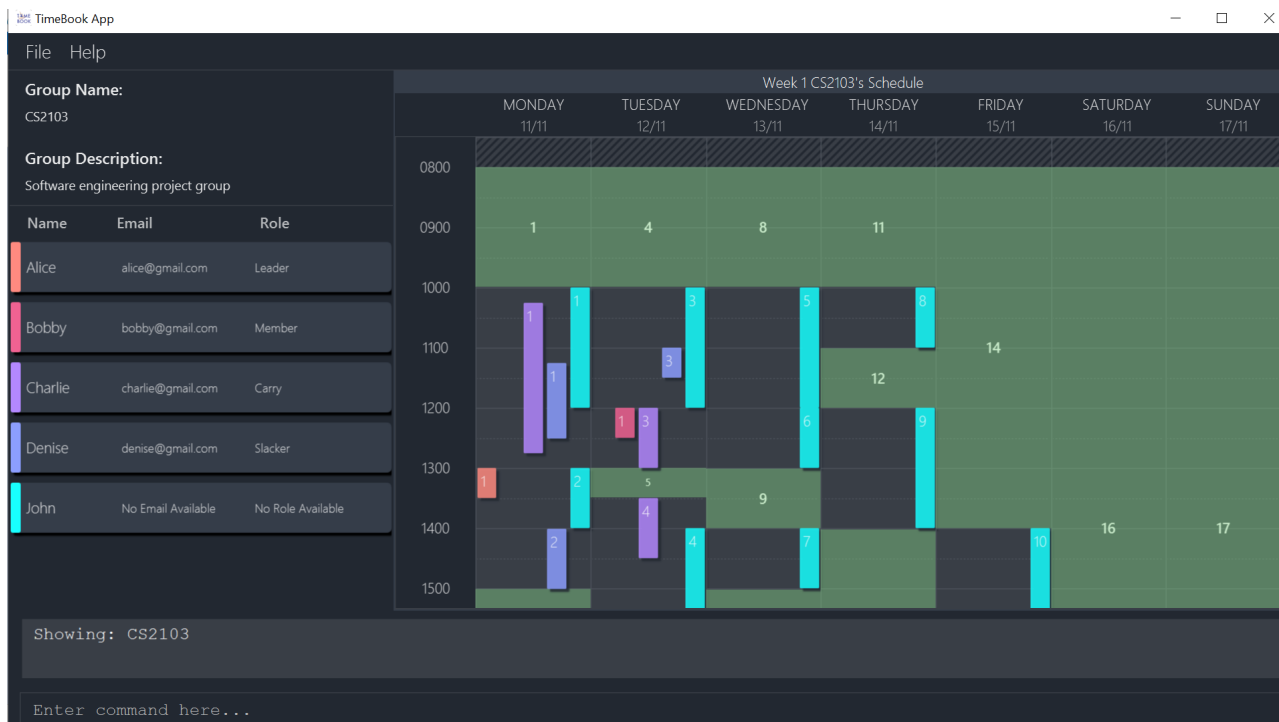
To personalize TimeBook and make it yours, rename **Alice** by entering the command **edituser n/Your Name Here**. If you'd like to fill in more details, you can do so using the **edituser** command.

Examples:

1. Type **show g/CS2103** in the command line as shown below



2. Hit the **Enter** key and you should see the details of this group in a similar window below.



You can use the command **scroll** to help you scroll the schedule view without touching your mouse!



You can use the command **togglenext** to see the schedule for next week! You may see this schedule up to 4 weeks in advance. After you've reached the fourth week's schedule, the next **togglenext** command will start back on the first week's schedule.



Unfortunately, TimeBook does not let you see a schedule that has already passed.

Look at some members: **lookat**

Suppose that all of your group members are extremely busy and the common free time is too short to do any productive work, you may use the **lookat** command to see if you can meet with some of your group members. You must be viewing a group's schedule in order for this command to work.

Format:

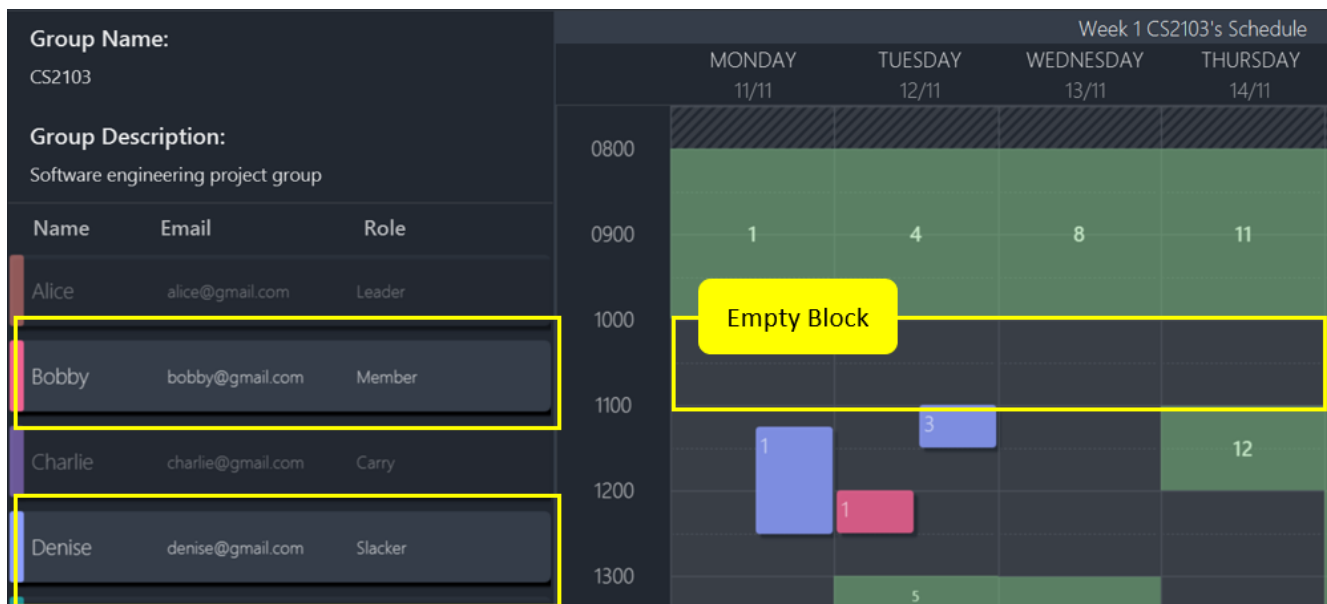
- **lookat** **[n/Name]...**

The **lookat** command takes in any number of **n/Name** and highlights the group members according to name.

Example:

Let's use the same schedule exactly like the one shown [above](#).

- Simply type **lookat n/Bobby n/Denise** if you are interested to inspect both of their schedules! You should see that Bobby's and Denise's schedule become highlighted.



Empty blocks represent the free time slots for the group members that you are looking at. The green blocks still represent the common free time slots for the entire group!

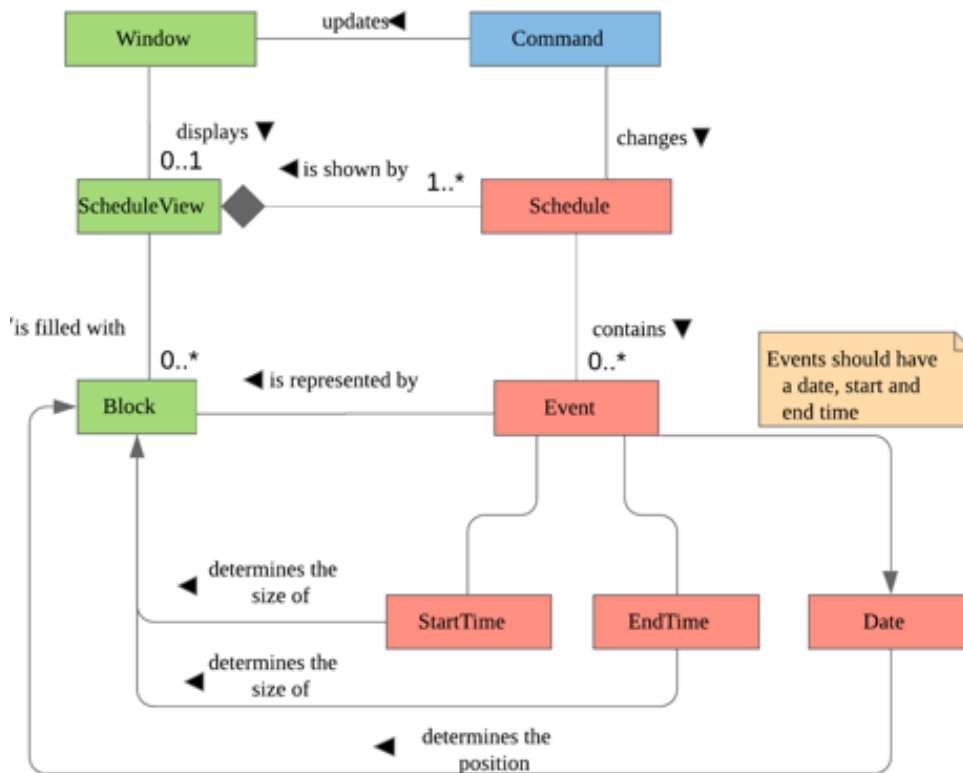
Contributions to Developer Guide

This section shows my contributions to the developer guide and my ability to convey technical information for my feature in the application that my team has developed.

Visual Representation of individual's or group's schedule feature

The visual representation refers to the beautiful graphics you see when you view a group or an individual's schedule in TimeBook. All of these graphics are created in the `ScheduleView` class. The model below illustrates the purpose of the `ScheduleView` class in TimeBook.

Object oriented domain model



The **ScheduleView** class in TimeBook follows the above model closely. Let's walk you through how the graphics are created:

1. Each **PersonTimeslot** object must first have a date, a start time and an end time.
2. Obtain a list of **PersonTimeslot** that has no clashes. This means that there should not be any overlapping time slots in the list.
3. Sort the list of according to their date first, followed by start times.
4. For each date, create a **container** to stack **Block** objects. Eventually, this **container** represents the graphic for one date in the **ScheduleView** object.
 - a. If the first **PersonTimeslot** in the list starts after 8am (TimeBook's schedule start time), stack an empty **Block** in the **container** with the height the same as the duration between 8am and the start time of the first **PersonTimeslot** object in the list.
5. For each **PersonTimeslot** object in the list, stack a coloured **Block** in the same **container**. This **Block** should have the same height as the duration between the start and end time of the **PersonTimeslot** object.
6. Stack in empty **Block** to fill the gaps between the end time of the current **PersonTimeslot** and the start time of the next **PersonTimeslot** in the list.

After having a class that creates the graphics for TimeBook, we require another class to control what graphics to show. As such, we made use of an abstract class **ScheduleViewManager** to control the creation of **ScheduleView** objects. The two classes that extend from **ScheduleViewManager** are **IndividualScheduleViewManager** and **GroupScheduleViewManager** and each of them controls the creation

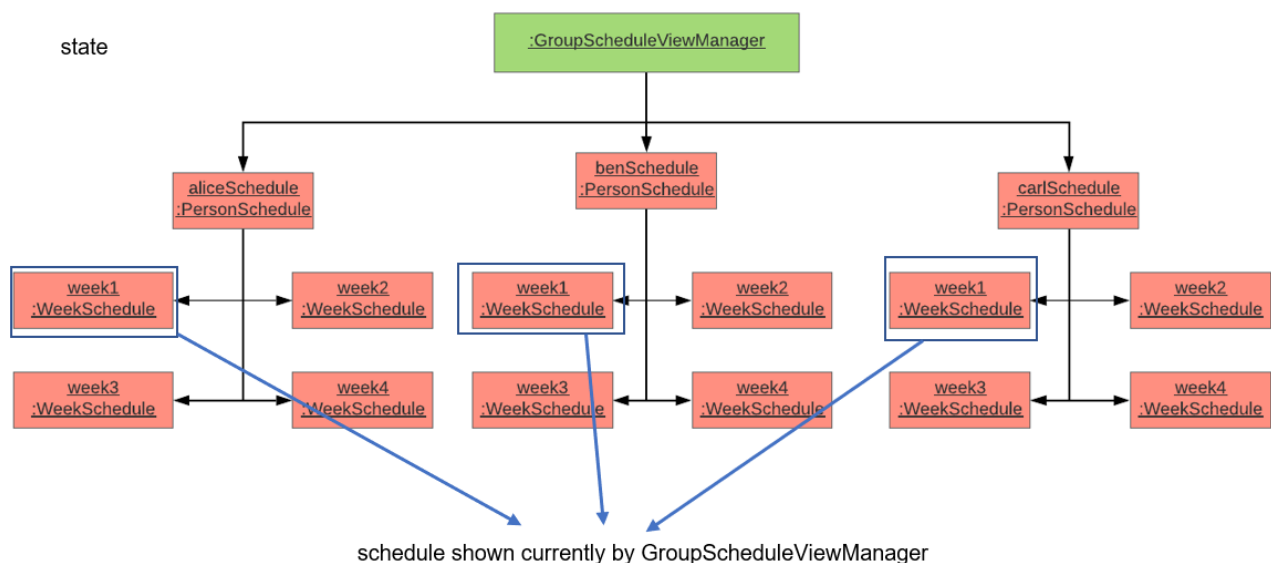
of individual and group schedules respectively.

The following methods are implemented in `ScheduleViewManager` to control the schedules displayed in the window.

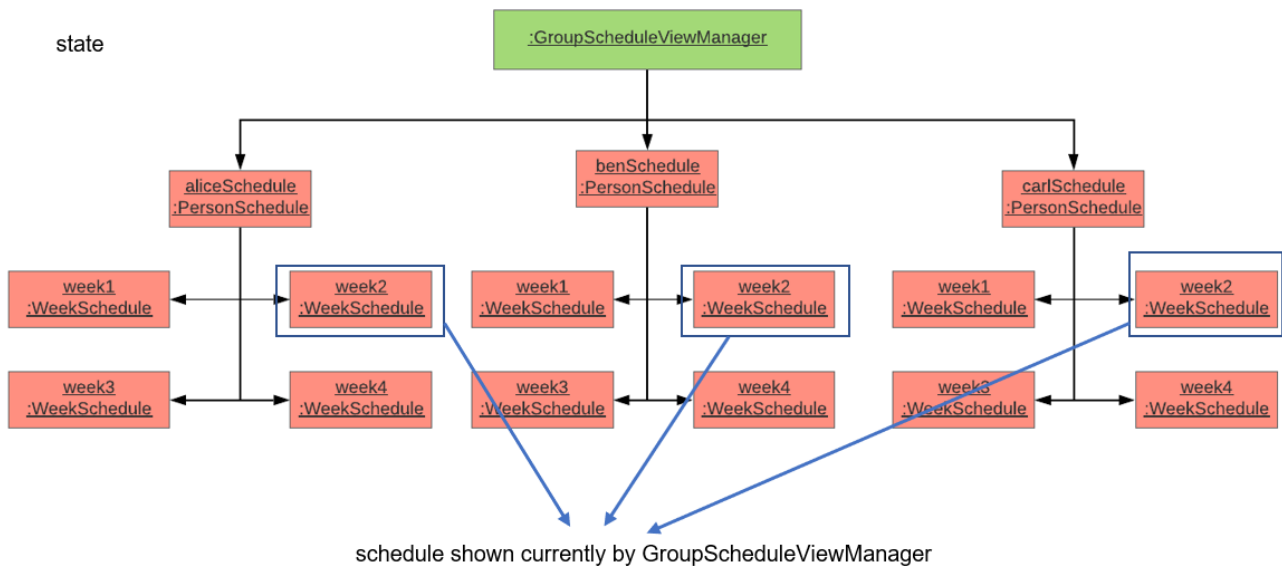
- `ScheduleViewManager#getInstanceOf(ScheduleDisplay)` — Instantiates the `ScheduleViewManager` with a given `ScheduleDisplay` object. The `ScheduleDisplay` object contains all the information needed to generate a schedule view.
- `ScheduleViewManager#scrollNext()` — Scrolls the schedule shown down. Once it reaches the bottom, it will start back at the top.
- `ScheduleViewManager#toggleNext()` — Modifies the schedule shown to show the next week's schedule. The schedule shown can at most show up to 4 weeks in advance. Once the fourth week is reached, it will start back at the first week.
- `ScheduleViewManager#filterPerson(List<Name>)` Filters the schedule shown to the given list of names. This method only works when the schedule shown is a group's schedule.

A sample usage of the `ScheduleViewManager` is described below.

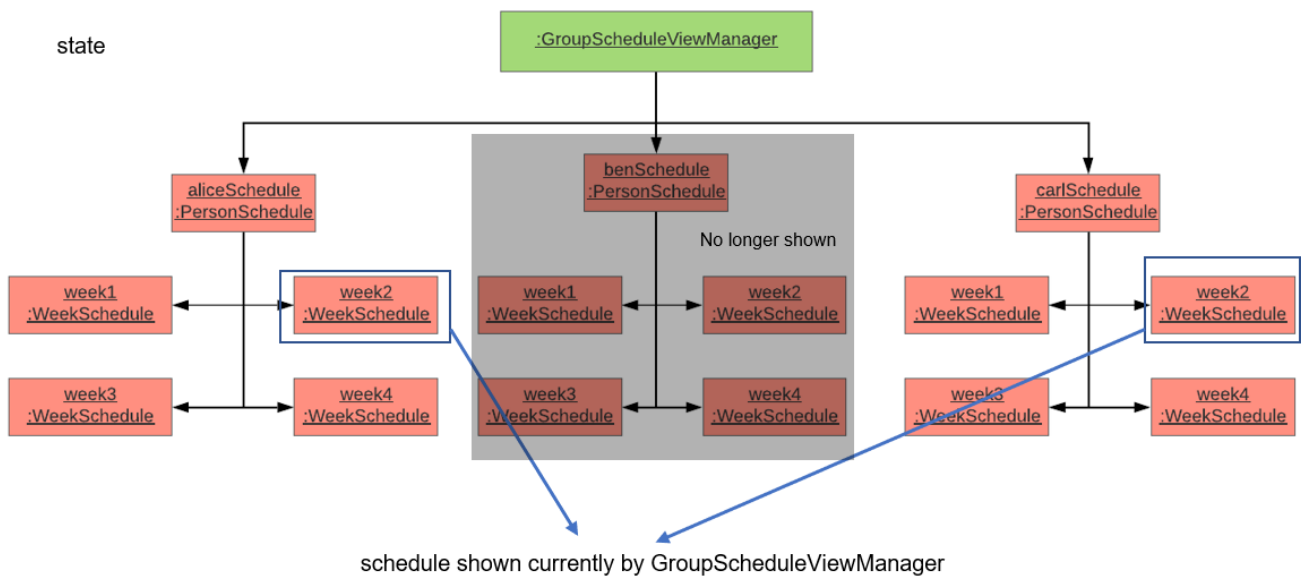
Step 1. The user wants to view a group called "Three musketeers" consisting of 3 members, Alice, Ben and Carl in TimeBook and executes the command `show g/Three musketeers` in the command line. The state of `ScheduleViewManager` will be initialised to show only the group's schedule for the first week as shown in the object diagram below.



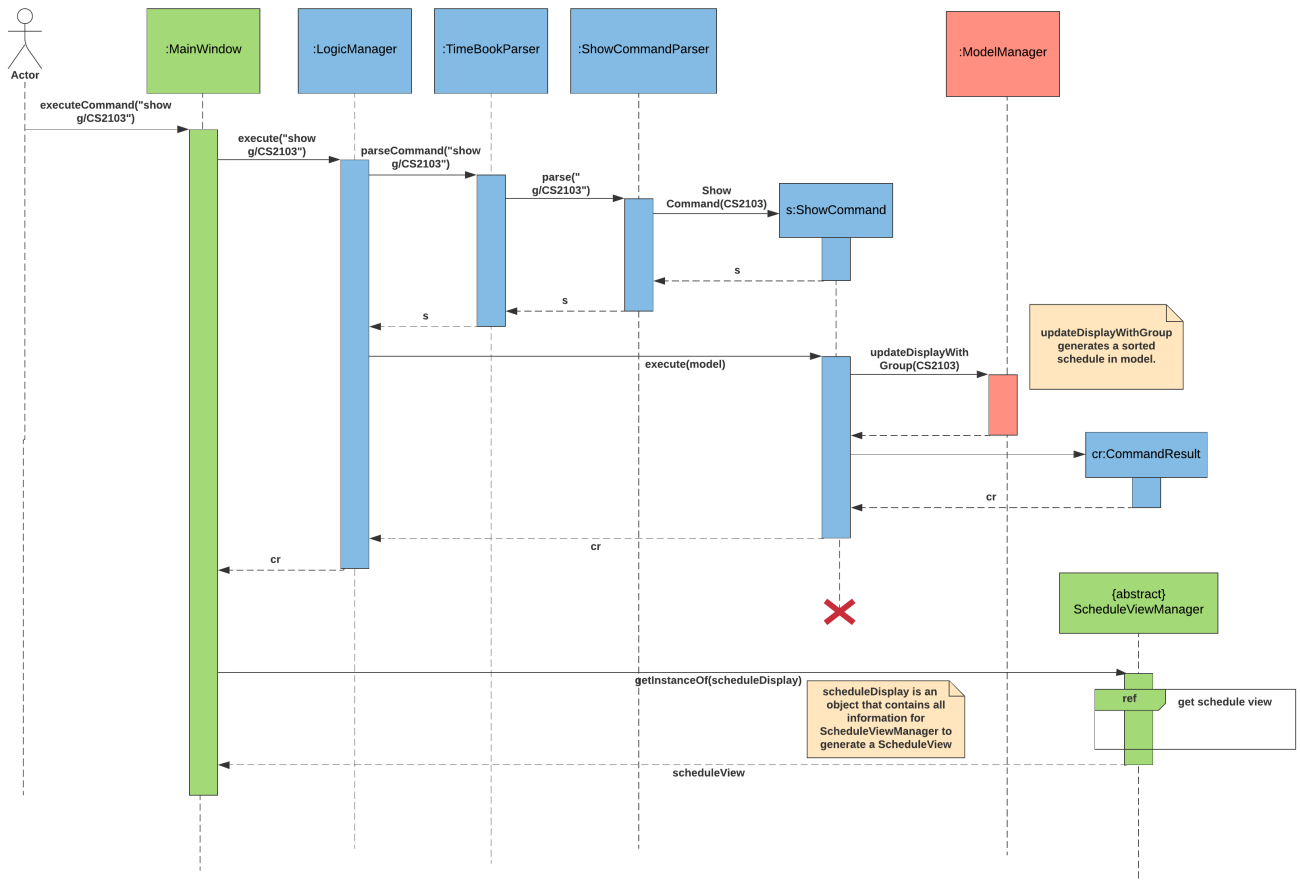
Step 2. Suppose the user thinks that arranging a group meeting on the first week is too rushed, so he executes the `toggleNext` command to view the group's schedule for the next week. The state of `ScheduleViewManager` is then modified to show the second week of the group's schedule as shown in the diagram below.



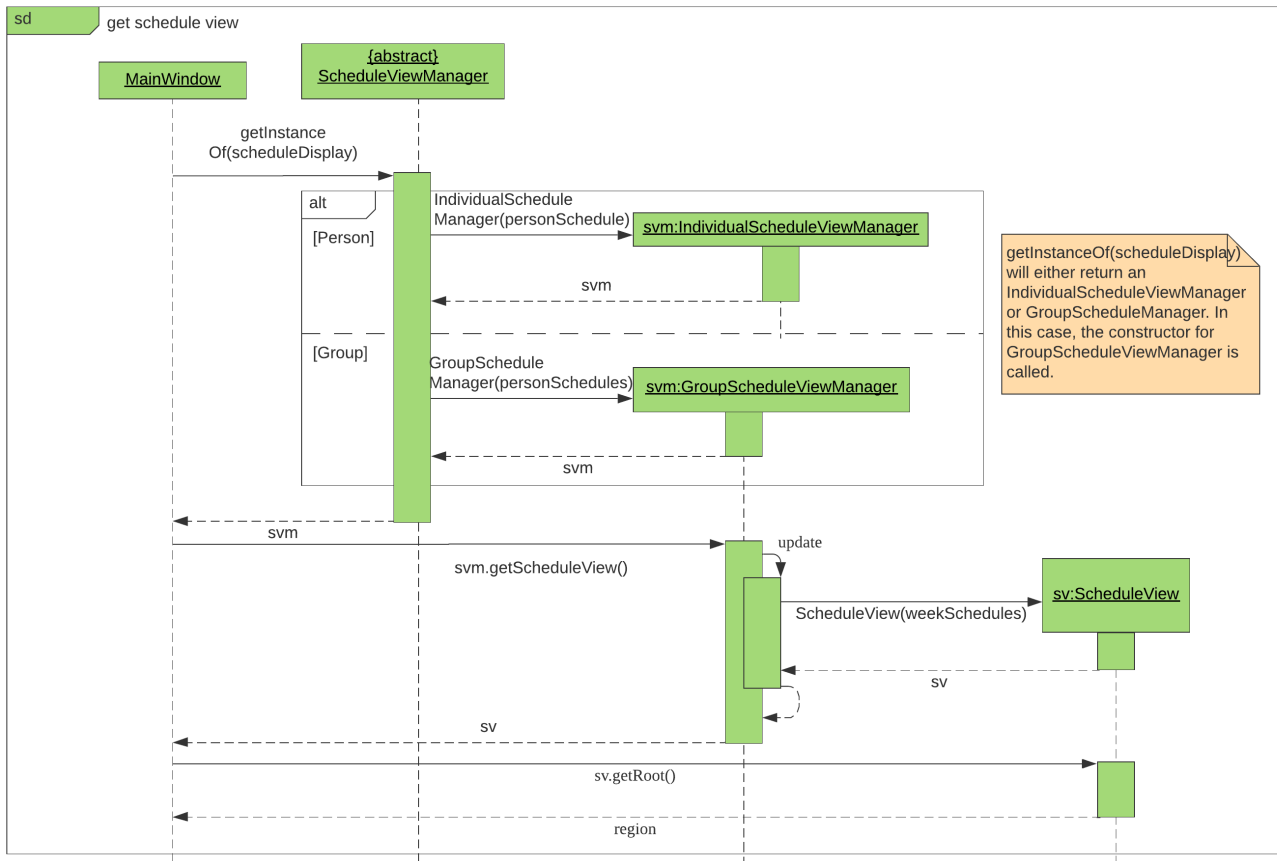
Step 3. Suppose the user wants to organise a group meeting with some of his group members and still want to keep track of the entire group schedule's free time slots, he executes the **lookat** command to inspect Alice's and Carl's schedules. The state of **ScheduleViewManager** is once again modified to only show the specified group members' schedules in the object diagram below.



Now that we have the full picture of how the graphics are created and controlled, we are ready to show how the user obtain a visual representation of a person or group's schedule using the **show** command. The following sequence diagram shows the sequence of events that lead to changes in the UI when an example of the **show** command is executed for a group called **CS2103**.



In order to make the diagram look less messy, a reference diagram shown below is created to show what happens in the **get schedule view** frame.



Details of how the graphics are created within the **ScheduleView** have been described above and thus, are omitted in the diagram.

Design Considerations

Aspect:	Choice	Pros	Cons
---------	--------	------	------

Amount of detail present in schedule view.	1. Enable users to see schedules up to 1 week in advance.	1. Easy to implement. 2. Less likely for bugs when invoking other commands such as select and popup.	1. Users may experience difficulty to plan meetings 2 or more weeks in advance.
	2. Enable users to see schedules up to 4 weeks in advance. (Current choice)	1. Most users should be able to plan most of their meetings. (Up to 1 month in advance).	1. Slightly more challenging to implement. 2. Slower as each request will take 4 times as long.
	2. Enable users to see schedules up to an indefinite weeks in advance.	1. Every users should be able to plan their meetings.	1. Difficult to implement. 2. Slow requests as every query will regenerate a new set of graphics.

We chose to allow users to see schedules up to 4 weeks in advance mainly due to usability. We recognise that most group meetings do not happen within a short period of 1 week as it may seem rushed for everyone in a group. We also found that it is unnecessary to enable users to see their schedules after the 1 month mark since it is most likely to not have been updated yet.

Aspect:	Choice	Pros	Cons
Viewing some group member's schedule in a group using the lookat command.	1. Filter and only to the specified group members from the command. Does not recalculate and display the free time slots for the filtered group members. (Current choice)	1. Easier to implement. 2. Faster for each query when schedules are more complicated. 3. Enable users to see the filtered schedules and still keep track of the entire group's schedule.	1. Users may be misled to think that the lookat command is not working as it does not update the displayed free time slots.
	2. filters, recalculate and display the common free time slot for the filtered members.	1. There will not be any misleading empty blocks in a group's schedule.	1. Difficult to implement. 2. Each query will take a lot longer to process the locations data.

We understand that users may want to inspect the schedules of some of his or her group members while still keeping track of the entire group's common free time slots. This would be useful for users who want to organise partial group meetings with some of his or her group members before or after the official group meeting (where everyone attends). Furthermore, filtering a group member can easily be done by just creating an entirely new group and adding the filtered group members in this newly created group.