Joshua Lee Kai Sheng - Project Portfolio

PROJECT: AddSurance

Overview

AddSurance is a desktop address book application. It targets technologically savvy insurance agents who wish to maintain a huge list of contacts, at the same time, manage appointments with (potential) clients. Interaction with this application is largely based on the Command Line Interface, aided by a simple GUI. This application is written in Java and has about 20 kLoC. Some of the tools and technologies used for this project include *Intellij, Github, SourceTree, SceneBuilder, JAVAFX, CalendarFX 8 by Dirk Lemmermann and Gmail API.*

Summary of contributions

Major enhancement

Implemented a package of scheduling functions to manage the user's appointments. This includes adding a `Calendar`, creating an `Appointment` model, as well as implementing the `addappointment`, `deleteappointment`, `zoomin`, `zoomout`, `gobackward`, `goforward` commands.

What it does

- Allows the user to manage upcoming appointments with (potential) clients and keep track of past appointments.

Justification

- Insurance agents tend to have many appointment arrangements with their huge list of (potential) clients. This necessitates an appointment management function for the address book.

Highlights

- This enhancement requires good adherence to the existing architecture as it touches **all components** of the architecture. A new `Appointment` model was created to support persistent storage of appointments in the address book. Besides, numerous commands were implemented for this enhancement. The most challenging part of all was the integration of third party calendar libraries, provided by **CalendarFX 8**, into the application.

Minor enhancement

Implemented a delete dated persons function which allows the user to delete persons added before a date who match all the tags specified by the user.

Justification

- Insurance agents tend to have many obsolete contacts especially those of non-clients. With this function, insurance agents can delete a group of people, as indicated by some tags, that was added before a date to clear up some space in the address book.

Highlights

- This enhancement requires in-depth, cross-component implementation despite being a minor enhancement. The Single Responsibility Principle was closely adhered to to ensure that the implementations do not violate the existing architecture.

Code contributed: [Functional Code][Adapted Functional Code][Test Code][Adapted Test Code]

Other contributions

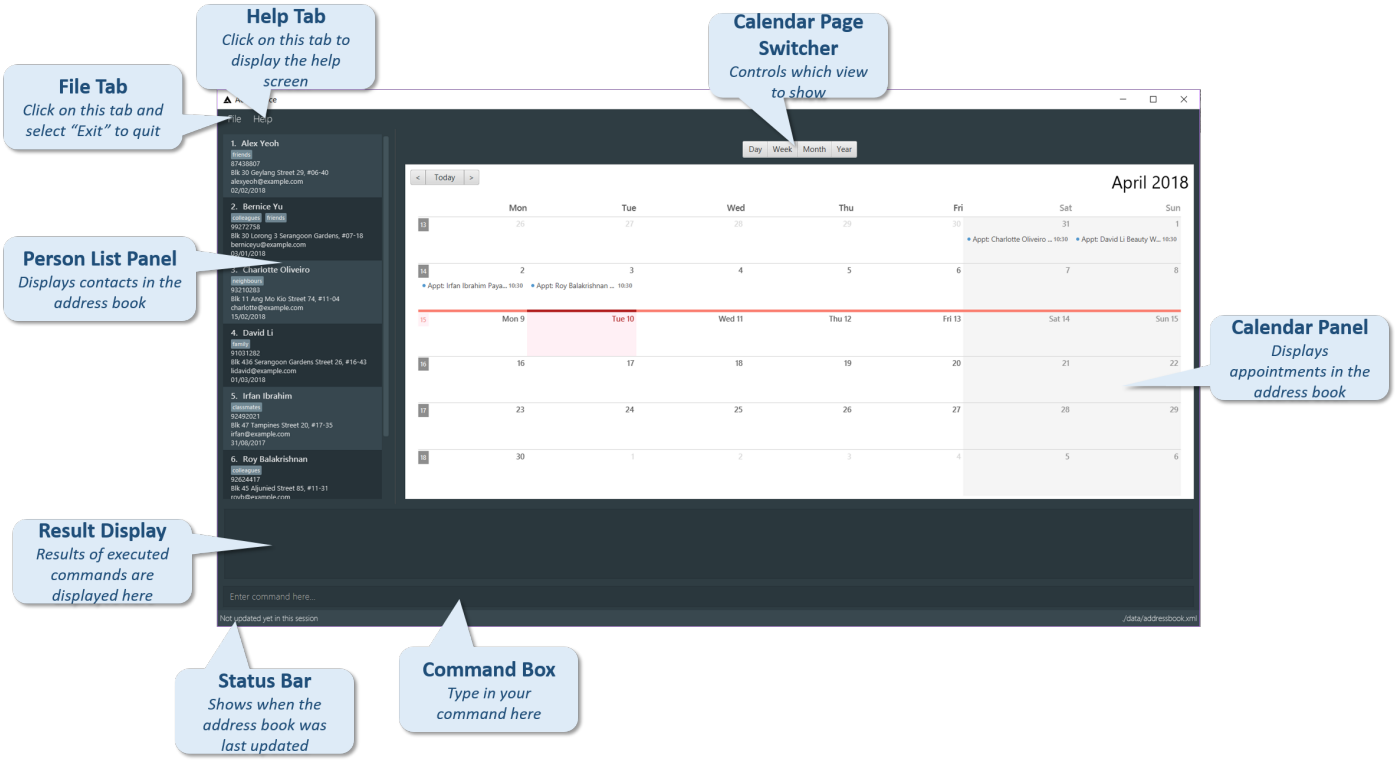
- **Project management**:
  - Team Lead
  - Set up the team repository and important tools for the project, such as Travis and Coveralls, on GitHub
  - Maintained the team repository:
    - Managed the team workflow and continual integration of the project (PR: #94, #56, #44, #176)
    - Managed issue tracking, labelling and milestones
  - Managed releases `v1.3` - `v1.5` on GitHub
- **Documentation**:
  - Managed the Developer Guide:
    - Updated the Product Scope, User Stories, Use Cases, Non Functional Requirements and Glossary sections (PR: #31, #154)
    - Updated diagrams to match the current implementation (PR: #154, #178)
  - Managed the User Guide:
    - Updated the Introduction section to match the current user profile (PR: #38, #162)

        - Rearranged the features into three logical sections for better clarity (PR: #162)
        - Tidied up the Command Summary section (PR: #162, #178)
    - Updated README to match the current value proposition (PR: #38, #92)
- **Repackaging of the product**:
    - Renamed the address book (PR: #123, #154, #38)
    - Changed the logo and theme of the address book (PR: #123, #161)
    - Enhanced the UI of the address book (PR: #161, #180)
- **Enhancement to existing features**:
    - Created command aliases for all existing commands together with the associated tests (PR: #34)
- **Community**:
    - Reviewed PRs for teammates with non-trivial review comments (PR: #77, #81, #118, #163, #170)
    - Reported bugs and given suggestions for other teams during mock User Acceptance Testing (PR: #163, #193, #194, #190, #186, #178, #175, #173, #168, #164, #188)

Contributions to the User Guide

Given below are some sections I contributed to the User Guide. They showcase my ability to write documentation targeting end-users.

UI Layout and Explanations



Deleting dated persons : `deletebefore` or `db`

Deletes all persons added before a specified date matching all specified tags.
Format: `deletebefore d/DATE t/TAG…` or `db d/DATE t/TAG…`

- Persons deleted must have all the specified `TAG` s.
- The specified `TAG` s can be in any order.
- **At least one** `TAG` must be provided.
- The input `DATE` **must be in the following format: dd/MM/yyyy**.

You are encouraged to **attach tags to persons** representing different groups. This way, you can delete persons of a particular group who were added before a date. *For example, to delete contacts relating to non-clients in batch, you can attach a **"nonclient"** tag to these persons and use the* `deletebefore` *command.*

Examples:

- `deletebefore d/12/12/2017 t/nonclients`
  Deletes all non-clients added before 12/12/2017 in the address book.

Scheduling Features

The calendar view is optimized for fullscreen mode.

Currently, there is a glitch in the scaling of the calendar Year Page. We are working with the third party provider to resolve this issue.

Adding an appointment : `addappointment` or `aa`

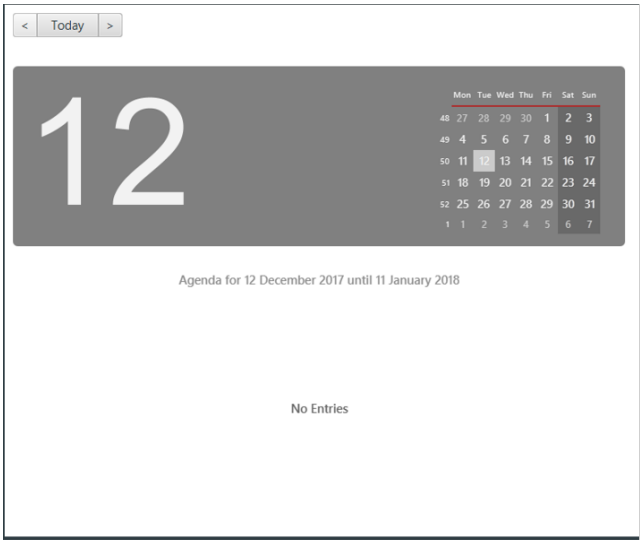
Adds an appointment to the address book that will be shown in the calendar panel.
Format: `addappointment n/NAME d/DATE st/STARTTIME et/ENDTIME l/LOCATION` or `aa n/NAME d/DATE st/STARTTIME et/ENDTIME l/LOCATION`

- Adds an appointment with the person named `NAME`, with the specified `DATE`, `STARTTIME`, `ENDTIME` and `LOCATION` respectively
- The input `STARTTIME` **must precede** `ENDTIME`
- The input `DATE` **must be in the following format: dd/MM/yyyy**.
- The input `STARTTIME` and `ENDTIME` **must be in the following format: HH:mm (24hr format)**.
- The calendar panel will be updated accordingly.

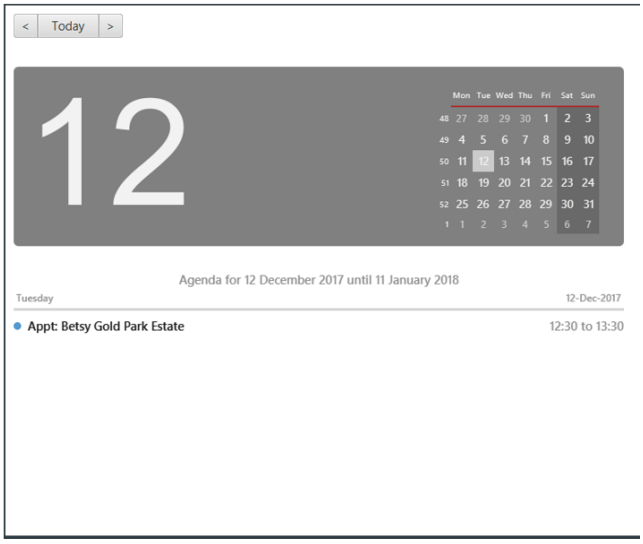
The appointment added **need not be related to any person existing in the address book**. This gives you the flexibility to keep track of appointments with anyone, without having to add that person into the address book.

You are **not allowed to add appointments that clash in time.**

Examples:



Before



After

- `aa n/Betsy d/12/12/2017 st/12:30 et/13:30 l/Gold Park Estate`
  Creates an appointment with Betsy on 12/12/2017 from 12:30pm to 1:30pm at Gold Park Estate.

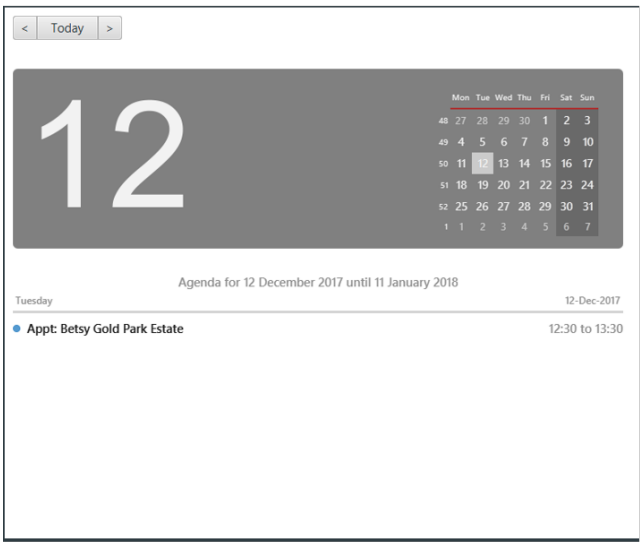
Deleting an appointment : `deleteappointment` or `da`

Deletes an appointment in the address book.
Format: `deleteappointment n/NAME d/DATE st/STARTTIME et/ENDTIME l/LOCATION` or `da n/NAME d/DATE st/STARTTIME et/ENDTIME l/LOCATION`

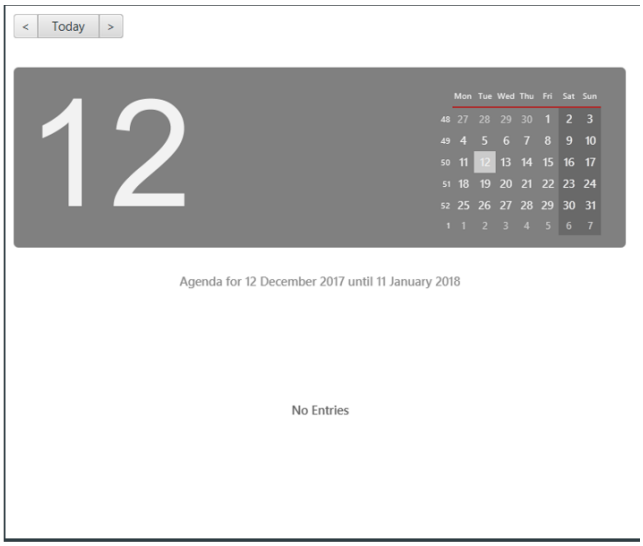
- Deletes an appointment that matches all the input `NAME`, `DATE`, `STARTTIME`, `ENDTIME` and `LOCATION`.
- The input `DATE` **must be in the following format: dd/MM/yyyy**.
- The input `STARTTIME` and `ENDTIME` **must be in the following format: HH:mm (24hr format)**.
- The calendar panel will be updated accordingly.

In the current version, the deleteappointment command requires you to input all fields of the appointment to be deleted. This command will be upgraded to the enhanced deleteappointment command which will be available in v2.0.

Examples:



Before



After

- `da n/Betsy d/12/12/2017 st/12:30 et/13:30 l/Gold Park Estate`
  Deletes the appointment previously arranged with Betsy on 12/12/2017 from 12:30pm to 1:30pm at Gold Park Estate.

Finding an appointment : `findappointment` or `fa [Coming in v2.0]`

Finds appointments with details containing any of the given keywords or part thereof.
Format: `findappointment KEYWORD…` or `fa KEYWORD…`

- Keywords are words that are part of any fields of an appointment, i.e. `NAME`, `DATE`, `STARTTIME`, `ENDTIME` and `LOCATION`
- The search is case insensitive. e.g. `gold park` will match `Gold Park`
- All the details of an appointment will be searched.
- Partial words will be matched e.g. `gol` will match `gold park`

Examples:

- `findappointment 12/12/2018`
  Returns any appointment that is scheduled on 12/12/2018
- `findappointment Betsy Tim John`
  Returns any appointment with `Betsy`, `Tim`, or `John`

Editing an appointment : `editappointment` or `ea [Coming in v2.0]`

Edits an appointment in the address book.
Format: `editappointment INDEX [n/NAME] [d/DATE] [st/STARTTIME] [et/ENDTIME] [l/LOCATION]` or `ea INDEX [n/NAME] [d/DATE] [st/STARTTIME] [et/ENDTIME] [l/LOCATION]`

- Edits an appointment at the specified `INDEX`. The index refers to the index number shown in the last appointment listing. The index must be a positive integer 1, 2, 3, …
- At least one of the optional fields must be provided.
- Existing values will be updated to the input values.
- The input `DATE` **must be in the following format: dd/MM/yyyy**.
- The input `STARTTIME` and `ENDTIME` **must be in the following format: HH:mm (24hr format)**.
- The calendar panel will be updated accordingly.

Examples:

- `findappointment Betsy`
  `editappointment 1 d/25/12/2017 l/Parkway Parade`
  Edits the date and location of the first appointment with Betsy in the listing to 25/12/2017 and Parkway Parade respectively.

Deleting an appointment (enhanced) : `deleteappointment` or `da` `[Coming in v2.0]`

Deletes the specified appointment in the address book.
Format: `deleteappointment INDEX` or `da INDEX`

- Deletes the appointment at the specified `INDEX`.
- The index refers to the index number shown in the most recent listing.
- The index **must be a positive integer** 1, 2, 3, …
- The calendar panel will be updated accordingly.

Examples:

- `findappointment 12/12/2018`
  `deleteappointment 1`
  Deletes the 1st appointment in the results of the `findappointment` command

Configuring calendar theme : `configcalendar` or `cc` `[Coming in v2.0]`

Configures the calendar view.
Format: `configcalendar s/STYLE` or `cc s/STYLE`

- Set the calendar theme to the specified predefined `STYLE`.

Examples:

- `cc s/STYLE_2`
  Set the calendar theme to the predefined `STYLE_2`.

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.

Delete Dated Persons Function

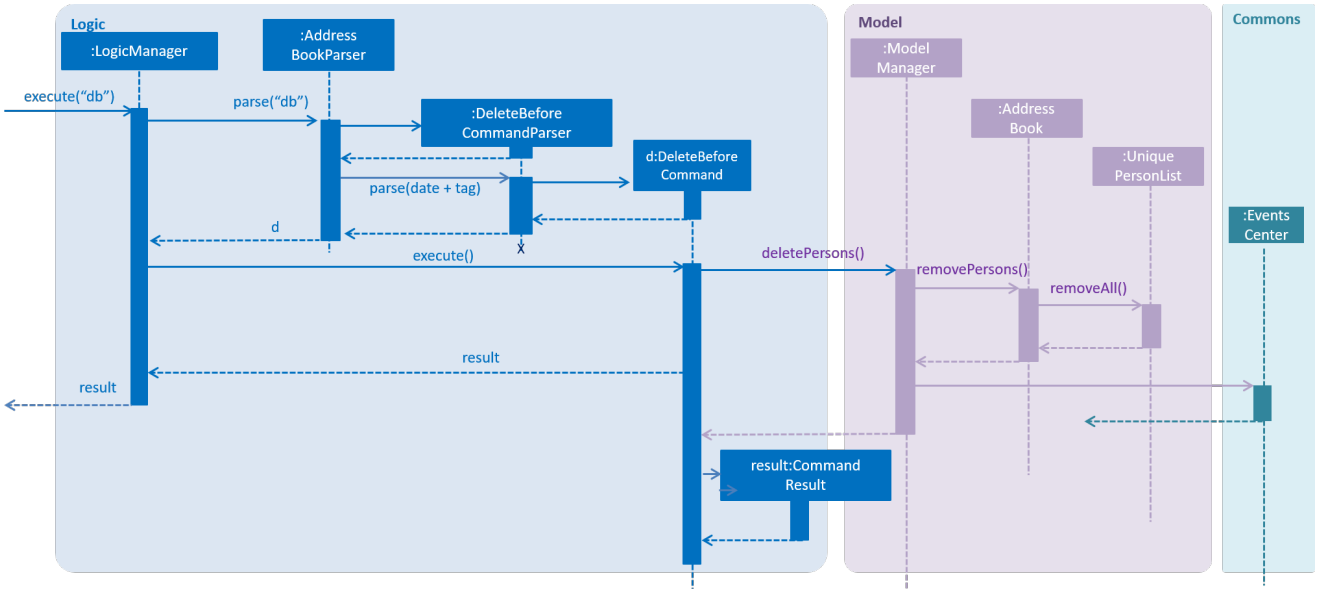
Current Implementation

The delete dated persons mechanism supports deleting all persons added before a date who possess all the tags specified by the user. This function is facilitated by adding a `DateAdded` attribute whenever a `Person` is added to the address book. This date stamp is added automatically by the application whenever the user adds a new person. Furthermore, to check if a `Person` is to be deleted, additional predicate classes are implemented.

The command for this function, `DeleteBeforeCommand` inherits from the `UndoableCommand`. This allows user to undo the delete should he/she does it by mistake. The `DeleteBeforeCommand` takes in `d/DATE` and `[t/TAG…]` as parameters. The `TAG` s parameter allows the user to delete only persons with the specified tags and not everyone added before a certain date. This enhances the flexibility of this delete function.

The implementation of this function adheres closely to the Object Oriented Principle as well as the Single Responsibility Principle.

Following is a sequence diagram describing how the delete dated persons function works:



Design Considerations

Aspect: Implementation of `DeleteBeforeCommand`

- **Alternative 1 (current choice):** Follows the general design of the `delete` command
  - Pros: Consistencies in the implementation of commands and command parsers
  - Cons: Requires additional methods for removing persons in the `AddressBook` and `UniquePersonList` classes
- **Alternative 2:** Extends the `delete` command
  - Pros: Does not require "exploratory" modifications in the `Model` component
  - Cons: Needs to call the `delete` command multiple times to delete all targeted persons

Aspect: Implementation of `DateAdded`

- **Alternative 1 (current choice):** Add `DateAdded` as an attribute of a `Person`
  - Pros: Consistencies in implementation of the `Person` model that is in line with the OOP and Single Responsibility Principle
  - Cons: Requires additional tests and overheads in maintaining an extra attribute
- **Alternative 2:** Include the `DateAdded` directly inside the `Person` class
  - Pros: Less overhead needed to maintain the `DateAdded` attribute
  - Cons: Reduces consistencies in the existing architecture and reduces coherence of the `Person` class

Aspect: Who adds the `DateAdded` attribute?

- **Alternative 1 (current choice):** The application automatically creates a `DateAdded` attribute for every `Person` added
  - Pros: Avoids unnecessary errors in user-input date
  - Cons: Additional stubs are needed to carry out tests in order to manually fix and manipulate `DateAdded`
- **Alternative 2:** Let the user input the current date
  - Pros: The user might input the wrong date or date with invalid format
  - Cons: Ease of implementation

Scheduling and Calendar Module

This entire section is dedicated to explaining the whole appointment scheduling and calendar module in view of its scale and significance.

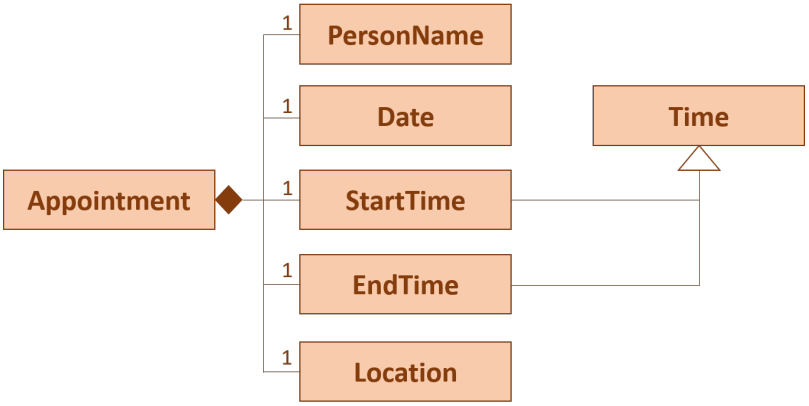
In general, the implementation of this module follows closely the current architecture of the system as shown in Section 2.1. This is to ensure consistency in the design of the system to keep it simple as *simplicity is consistency*.

A number of Software Engineering Principles are used in the implementation of this module, namely **OOP**, **SOLID**, **YAGNI**, **DRY** principles. Besides, the commands related to this module are also implemented using the **Command Design Pattern**.

Appointment Model

Current Implementation

The `Appointment` model is created to support the scheduling feature of the address book. It stores the information of appointments arranged by the user. The class diagram for the `Appointment` model is as follows:



Similar to the `Person` model, the `Appointment` model consists of an `Appointment` class which has its attributes implemented as separate classes to observe the **Single Responsibility Principle**. The attributes include:

i. `PersonName` ⇒ the name of the person the user is having appointment with

ii. `Date` ⇒ the date of the appointment

iii. `StartTime` ⇒ the start time of the appointment

iv. `EndTime` ⇒ the end time of the appointment

v. `Location` ⇒ the arranged location of the appointment

The `Appointment` s are stored in a `UniqueAppointmentList` in the `AddressBook`. The existing `Model` interface is extended with functionalities to manage the filtered list of `Appointment` s. This extension is kept as minimised as possible in light of the **YAGNI** Principle.

Design Considerations

Aspect: Model Manager for Appointment

- **Alternative 1 (current choice):** The existing `ModelManager` is extended to handle the filtered list of `Appointment` s
  - Pros: This design adheres to the **DRY** Principle as there will not be a duplicated `ModelManager` for the `Appointment` model with repeated functionalities as the existing one
  - Cons: Additional methods added to the `Model` interface might increase coupling for existing classes that are dependent on the `ModelManager`. For example, modifications made for `Appointment` in the `ModelManager` might now affect other commands unrelated to `Appointment` which call the `ModelManager`.
- **Alternative 2:** Create another `Model` interface implemented by another `ModelManager`
  - Pros: Ease of implementation and minimal impact on classes coupled with the existing `ModelManager`
  - Cons: Violates the **DRY** Principle and causes redundant codes

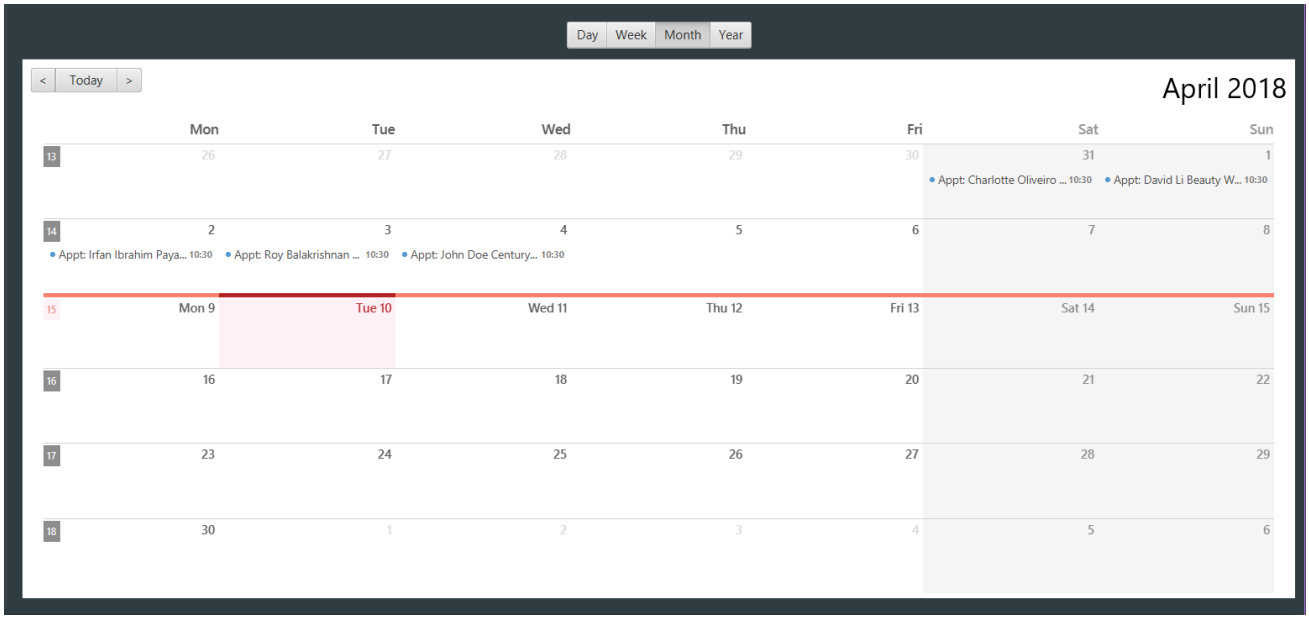
Aspect: Creation of StartTime and EndTime attributes

- **Alternative 1 (current choice):** `StartTime` and `EndTime` are created as separate classes which extends the `Time` class
  - Pros: This implementation adheres to the Polymorphism and Inheritance concepts in OOP. Both `StartTime` and `EndTime` has a *is a* relationship with `Time`. The separation of `StartTime` and `EndTime` into two classes also support possible future extensions to these attribute classes. For example, it is now possible to set different limits to `StartTime` and `EndTime` with this implementation.
  - Cons: The `StartTime` and `EndTime` have to adhere to the **Liskov Substitution Principle** which means that they cannot be more restrictive than the parent `Time` class.
- **Alternative 2:** Create `StartTime` and `EndTime` as separate unrelated classes
  - Pros: Avoids coupling between `StartTime` and `EndTime` due to an additional `Time` superclass
  - Cons: Violates the **DRY** Principle as `StartTime` and `EndTime` share some functionalities, such as the checking of the validity of an input date

Calendar UI

Current Implementation

The calendar is implemented using third party libraries provided by CalendarFX. The calendar panel view is shown below:



The setting up of the `CalendarView` is done by first creating a `Calendar` named *Appointments* which is added to a `CalendarSource`. The `CalendarSource` is then added to the `CalendarView`. The implementation details are as shown in the following code snippet:

```
CalendarView calendarView;
Calendar calendar;

calendar = new Calendar("Appointments");
CalendarSource calendarSource = new CalendarSource("My Calendar");
calendarSource.getCalendars().addAll(calendar);

calendarView.getCalendarSources().addAll(calendarSource);
```

The `Appointment` s stored in the address book is loaded as `Entries` into the `Calendar` upon start up. The list of `Appointment` s is passed into the `CalendarPanel` class. For each of these `Appointment` s, its attributes are used to create an `Entry`, which is then added to the `Calendar`.

When there are changes to the filtered list of `Appointment` s due to addition or deletion of appointments, the `Calendar` is updated using **Event Handlers**. The event handling for the addition of a new `Appointment` is shown below:

```
@Subscribe
private void handleNewAppointmentAddedEvent(NewAppointmentAddedEvent event) {
    logger.info(LogsCenter.getEventHandlingLogMessage(event));
    loadEntry(event.getAppointmentAdded());
}
```

Design Considerations

Aspect: Updating of Calendar due to addition or deletion of appointments

- **Alternative 1 (current choice):** Use of event handling to update the `Calendar`
  - Pros: Event handling is suitable for UI purposes due to its flexibility
  - Cons: The flow of program is less logical and obvious
- **Alternative 2:** The `DeleteAppointmentCommand` and `AddAppointmentCommand` call the `UIManager` to invoke the update to the calendar
  - Pros: Associations between the `Logic` and `UI` components are clearer
  - Cons: This will introduce a two way dependency between the `Logic` and `UI` components which may lead to confusion and make the `Logic` component coupled to the `UI` component

Aspect: Creation of Calendar

- **Alternative 1 (current choice):** Use third party libraries
  - Pros: Readily available hence time saving
  - Cons: There might be possible bugs in the libraries provided by third parties

There is an existing bug in the `YearPage` for the `Calendar`, which causes the `YearPage` to fail to scale its size according to the size of the `Pane` container

- **Alternative 2:** Create our own `CalendarView`

- Pros: More flexibility with the design of the `CalendarView`
- Cons: Time consuming and manpower intensive

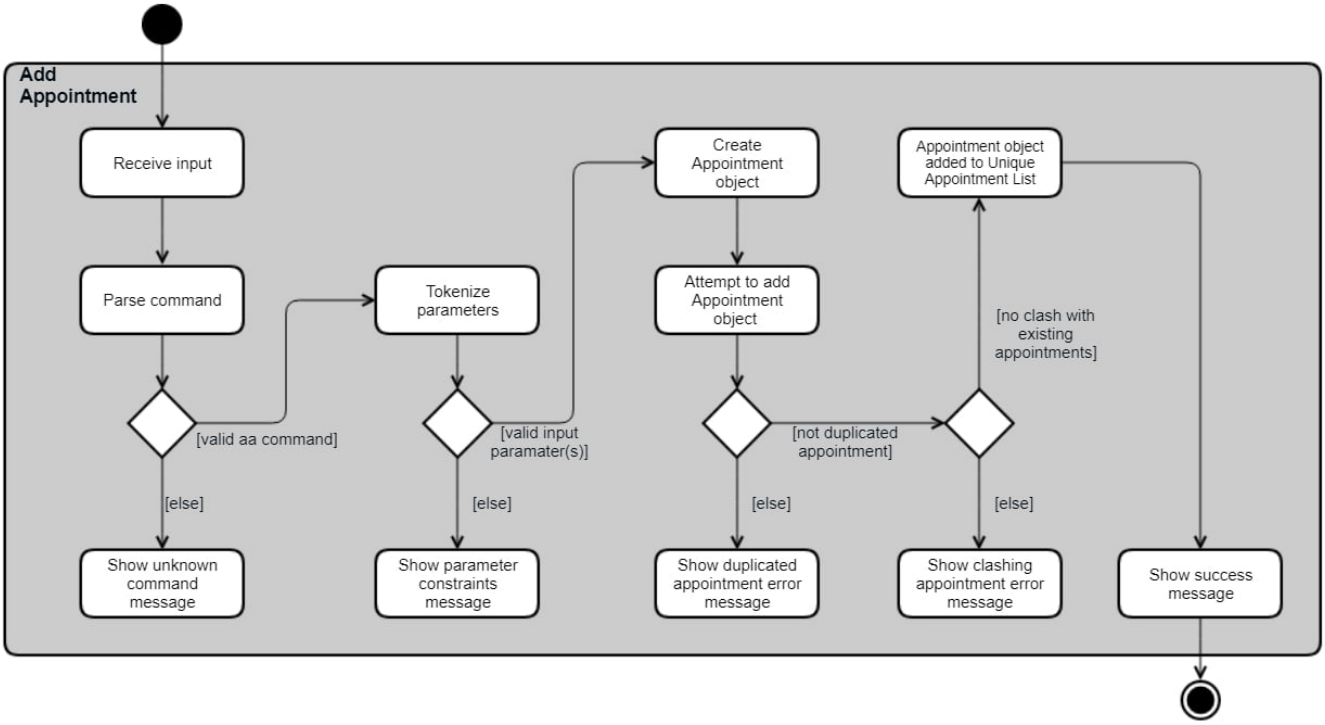
Add Appointment Function

Current Implementation

The add `Appointment` function is implemented as the `AddAppointmentCommand` together with its associated `AddAppointmentCommandParser` class. This function supports the adding of an `Appointment` to the address book. The `AddAppointmentCommand` inherits from the `UndoableCommand` to allow user to undo the addition of an `Appointment`. The implementation of the `AddAppointmentCommand` is similar to that of the `AddCommand`.

To enhance the logic of this function, checks are implemented to ensure that the `StartTime` of an appointment precedes the `EndTime`. Besides, the function also disallows the creation of clashing appointments in time.

Following is an activity diagram describing the flow of the add `Appointment` function:



Design Considerations

Aspect: What `Event` to raise when adding a new `Appointment`?

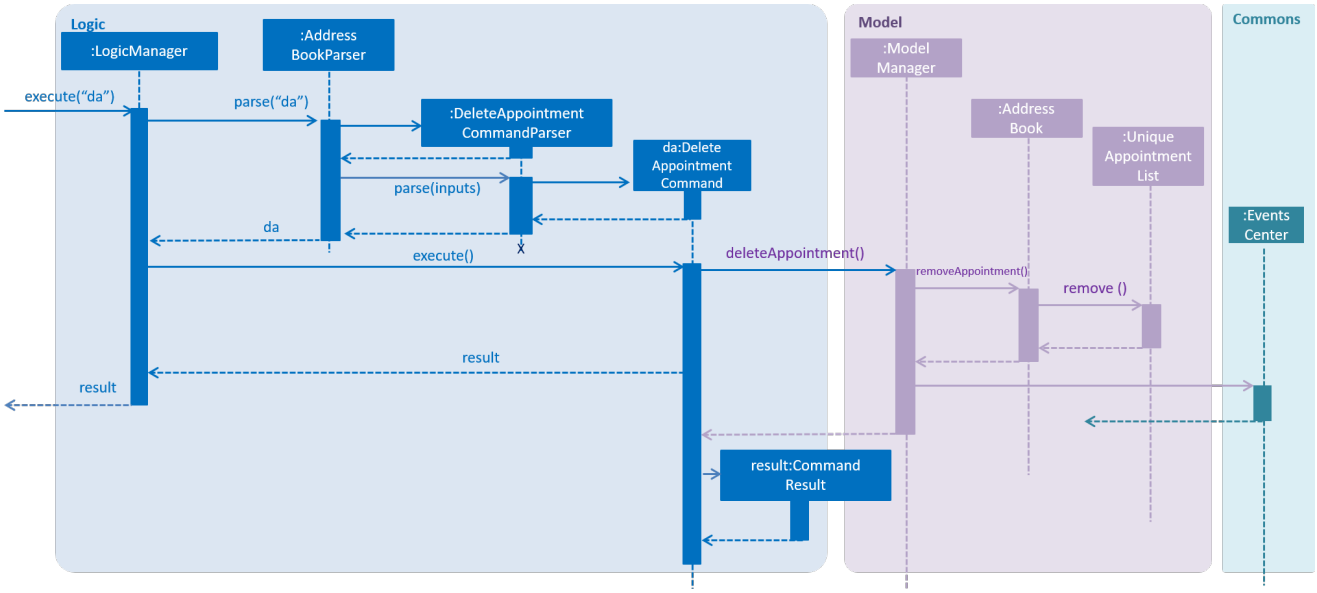
- **Alternative 1 (current choice):** Raise a specially created `NewAppointmentAddedEvent`
  - Pros: Do not have to respond to unrelated or unnecessary events
  - Cons: Overhead in maintaining the new `Event` class
- **Alternative 2:** Raise the existing `AddressBookChangedEvent`
  - Pros: Do not have to implement a new `Event` class
  - Cons: The `Calendar` will refresh whenever `AddressBookChangedEvent` is raised, including when only the `Person` list is changed

Delete Appointment Function

Current Implementation

The delete `Appointment` function is implemented as the `DeleteAppointmentCommand` together with its associated `DeleteAppointmentCommandParser` class. This function supports the deletion of an `Appointment` from the address book. The `DeleteAppointmentCommand` inherits from the `UndoableCommand` to allow user to undo the deletion of an `Appointment`. The implementation of the `DeleteAppointmentCommand` is similar to that of the `DeleteCommand`.

Following is a sequence diagram describing how the delete `Appointment` function works:



Design Considerations

Aspect: What fields to input when deleting an `Appointment`?

- **Alternative 1 (current choice):** Use all attributes of an `Appointment` as input fields to delete a matching `Appointment`
  - Pros: Certainty in deleting an `Appointment`
  - Cons: User has to input all fields which may be cumbersome
- **Alternative 2:** Only input the `PersonName` and `Date` of the `Appointment` to be deleted
  - Pros: User does not have to input all the attributes of an `Appointment`
  - Cons: There may be more than one `Appointments` with the same `PersonName` and `Date`, which may cause unexpected behaviour by the `DeleteAppointmentCommand`

Calendar Zooming Function

Current Implementation

The calendar zooming function is implemented by the `ZoomInCommand` and `ZoomOutCommand`. These `Command` classes are responsible for raising the `ZoomInEvent` and `ZoomOutEvent`. In turn, `ZoomInEvent` and `ZoomOutEvent` are created to support this function.

If a user tries to zoom in or out when the zoom level is at its maximum, an error message will be shown. To facilitate this, the `CalendarPanel` contains logic that will raise the `MaxZoomInEvent` or `MaxZoomOutEvent` for such situations. `ZoomInCommand` and `ZoomOutCommand` will then handle these events raised by throwing a `CommandException` containing the error message.

The actual implementation of the `ZoomInCommand` is as follows:

```
public class ZoomInCommand extends Command {
    public static final String COMMAND_WORD = "zoomin";
    public static final String COMMAND_ALIAS = "zi";

    public static final String MESSAGE_SUCCESS = "Calendar zoomed in";
    public static final String MESSAGE_MAX_ZOOM_IN = "The calendar is already zoomed in to the maximum level";

    private final Logger logger = LogsCenter.getLogger(this.getClass());
    private boolean receivedFeedback = false;
    private boolean isSuccessful = false;

    @Override
    public CommandResult execute() throws CommandException {
        registerAsAnEventHandler(this);
        raise(new ZoomInEvent());
        while (!receivedFeedback);

        if (isSuccessful) {
            return new CommandResult(MESSAGE_SUCCESS);
        } else {
            throw new CommandException(MESSAGE_MAX_ZOOM_IN);
        }
    }

    @Subscribe
    private void handleMaxZoomInEvent(MaxZoomInEvent event) {
        logger.info(LogsCenter.getEventHandlingLogMessage(event));
        receivedFeedback = true;
        isSuccessful = false;
    }

    @Subscribe
    private void handleZoomSuccessEvent(ZoomSuccessEvent event) {
        logger.info(LogsCenter.getEventHandlingLogMessage(event));
        receivedFeedback = true;
        isSuccessful = true;
    }
}
```

The implementation for `ZoomOutCommand` is highly similar to that of the `ZoomInCommand`.

Design Considerations

Aspect: Where should the event raising be implemented for the `ZoomInCommand` and `ZoomOutCommand`?

- **Alternative 1 (current choice):** Implement event raising in the `Command` superclass
  - Pros: Adheres to the **DRY** principle as `ZoomInCommand` and `ZoomOutCommand` do not have to implement a duplicated event raising functionality
  - Cons: All other command subclasses inherits the unnecessary event raising function
- **Alternative 2:** Implement event raising in a specially created subclass of `Command`, which is in turn inherited by the `ZoomInCommand` and `ZoomOutCommand`
  - Pros: Commands that do not need to raise event do not inherit the event raising function
  - Cons: Might over-complicate things for these simple commands.