Raynold Ng - Project Portfolio

Introduction

This Project Portfolio details my contributions to past projects, showing that I am capable of producing functional code, and work in a team on a large project.

PROJECT: Event Planning isn't Complicated (EPIC)

Overview

Event Planning isn't Complicated (EPIC) is a desktop application used for **event planning and registration for large organisations**. EPIC handles event attendance management with simplicity and efficiency. By storing all of the contacts in a common database, the same person can be registered for multiple events without having to re-enter his or her details for each event. EPIC is optimized for event planners who prefer typing to using the mouse.

Summary of contributions

- **Major enhancement**: overhauled UI to a three panel system that displays all persons, events and attendees of the selected event in their respective panels.
    - What it does: A three-panel system displaying stored persons, events and attendees. It auto updates itself when there are changes to data.
    - Justification: This feature **enhances the user experience** tremendously as the user can view all three panels without having to cycle between lists. Being able to view all three lists in a clean and intuitive interface empowers the user to type out commands that require parameters from all three lists without breaking his workflow.
    - Highlights: This enhancement leverages several patterns such as the Model View Controller (MVC) Pattern, Observer Pattern, and JavaFX Beans Convention to elegantly trigger UI updates upon data changes. Using such patterns made UI updates **automatic and resource-efficient** The feature was implemented in a modular fashion, which makes maintenance and testing easy.
- **Minor enhancement**: added the `find-attendance` command that filters the attendance panel and the `list-attendance` command that resets it.
- **Code contributed**: [Functional code] [Test code]
- **Other contributions**:
    - Project management:
        - Managed releases v.1.4.1- v.1.5rc on GitHub
    - Enhancements to existing features:
        - Fixed failing test cases due to incorrect data (Pull requests #110)

        - Wrote additional tests for existing features (Pull requests #148, #170)
    - Documentation:
        - Did cosmetic tweaks to existing contents of the User Guide: #158
    - Community:
        - PRs reviewed (with non-trivial review comments): #44
        - Reported bugs and suggestions for other teams in the class (examples: 1, 2)

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.

Layout

The UI consists of the 5 main layout regions (Figure 1):

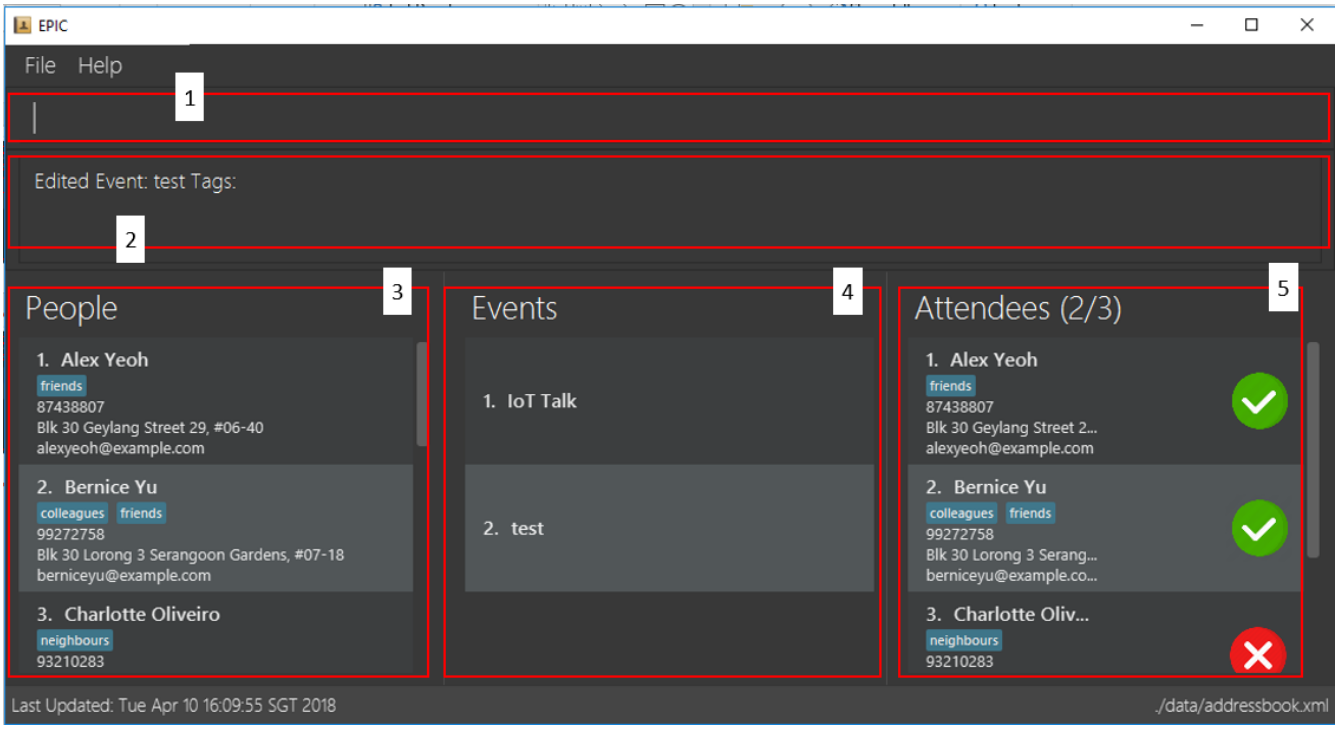

Figure 1. Components of the GUI

1. Command Box: text box to receive user inputs.
2. Result Display: status bar that displays the result of user commands.
3. People List Panel: panel that lists all persons stored in EPIC.
4. Event List Panel: panel that lists all events stored in EPIC.
5. Attendance List Panel: panel that lists all registrants in the selected event.

Selecting an Event: `select-event`

Selects the event identified by the index number used in the Events List Pane. The Attendance List Panel will display the persons registered for that event.

Format: `select-event INDEX`

TIP	By default, the first event in the Event List Panel is the selected event.

Listing all Event Participants: `find-registrant`

Finds registrants whose names contain any of the given keywords.

Format: `find-registrant KEYWORD [MORE_KEYWORDS]`

- The search is case-insensitive. e.g hans will match Hans.
- Only the name is searched.
- Only full words will be matched. e.g. Han will not match Hans.

If your event contains too many participants, it can be difficult to find information quickly. The `find-registrant` command can be used to narrow down participants by their name. Only persons whose names contain any of the given keys are displayed.

In our example, we'll apply a filter to display only participants from the IoT Talk Event with the name "Alex".

1. Select the IoT Event by executing `select-event 1`. Observe that Alex Yeoh is registered
2. Execute `find-registrant alex`, this filters the attendance panel, hiding anyone whose name does not contain the word alex (case-insensitive).
3. To remove the filter, execute `list-attendance`.

When a filter is applied, the header of the attendance panel will display `filtered`, and the attendance statistic will be computed based on the filtered list. See Figure 2 below for UI changes.

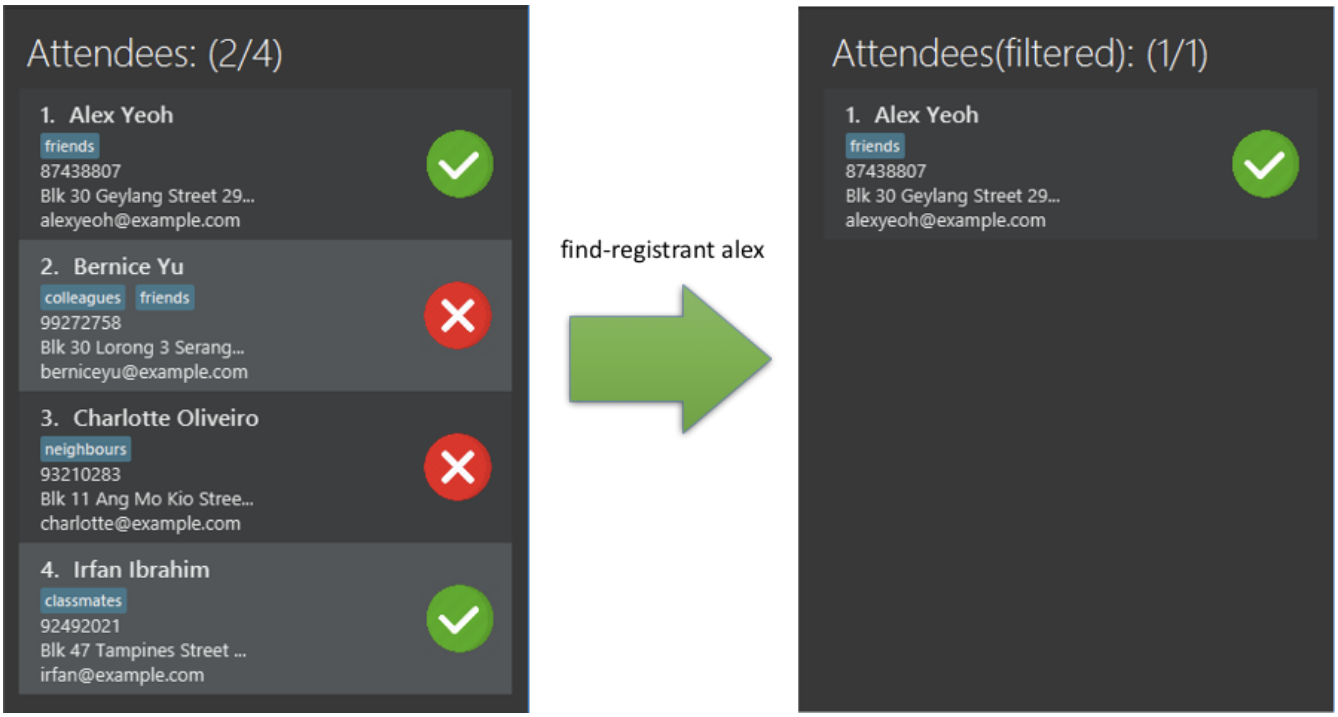

Figure 2. Illustration of `find-registrants` command (only the Attendees Pane is shown for clarity).

- The search is case insensitive. e.g hans will match Hans.
- Only the name is searched.
- Only full words will be matched. e.g. Han will not match Hans.

Listing all Event Participants: `list-attendance`

Lists all participants for the specified event, removing any filter.
Format: `list-registered EVENT_NAME`

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.

Find Registrant Feature

The `find-registrant` command enables users to filter registrants of an event with keywords. Only registrants whose names contain any of the given keywords will be listed. The user can further toggle the attendance of the registrants with the `toggle` command.

Figure 3 below shows how the `find-registrant` command is processed in the Logic component.

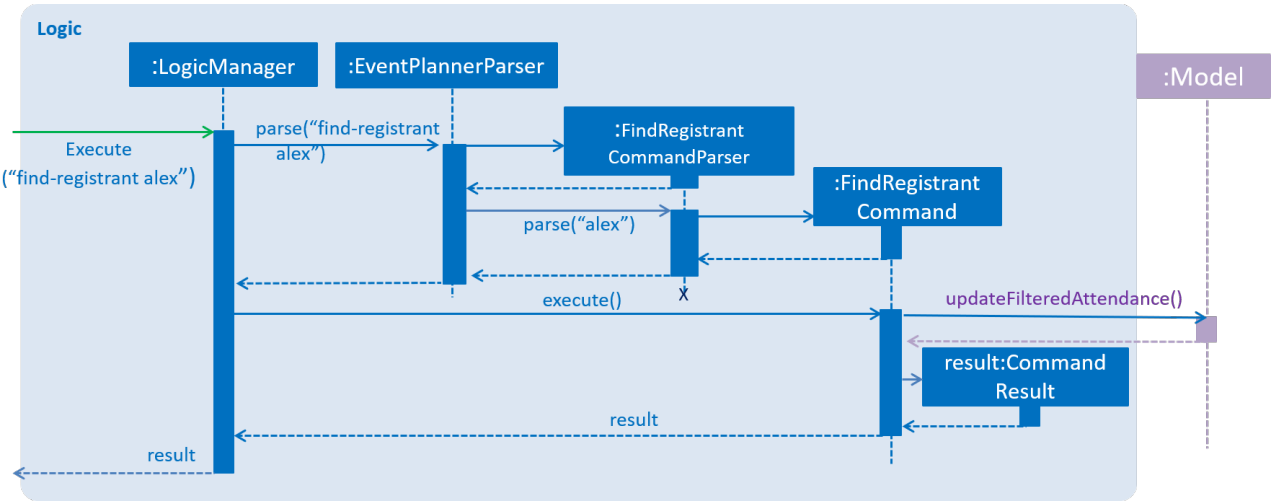

Figure 3. Sequence Diagram for `find-registrant` command

Current Implementation

The `find-registrant` command is passed to the `EventPlannerParser` object when then reads the parses the input and returns a `FindRegistrantCommand` object. When the `FindRegistrantCommand` object is executed, the Model filters the list of registrants and the Attendance List Panel is updated.

Design Considerations

Aspect: Keyword Matching Pattern

- **Alternative 1 (current choice)**: Match any of the keywords
  - Pros: The user can find multiple registrants using multiple keywords.
  - Cons: If the user will not be able to search by full name as other registrants might be matched as well.
- **Alternative 2**: Match all keywords
  - Pros: The user can search by full name.
  - Cons: Matching all keywords might be too restrictive and would not a useful filter to narrow down registrants.

- **Alternative 3** Fuzzy Search
  - Pros: The user can filter registrants with approximate keywords.
  - Cons: It is harder to implement fuzzy search.

UI Component

The UI component is responsible for interfacing with the user. The UI component has three main responsibilities:

- It executes user commands using the Logic component.
- It binds itself to some data in the Model component so that it can auto-update itself when data in the Model component changes.
- It responds to events raised from various parts of the App and updates itself accordingly.

Functional Overview

The UI consists of a `MainWindow` that contains the 5 main layout regions (Figure 4):

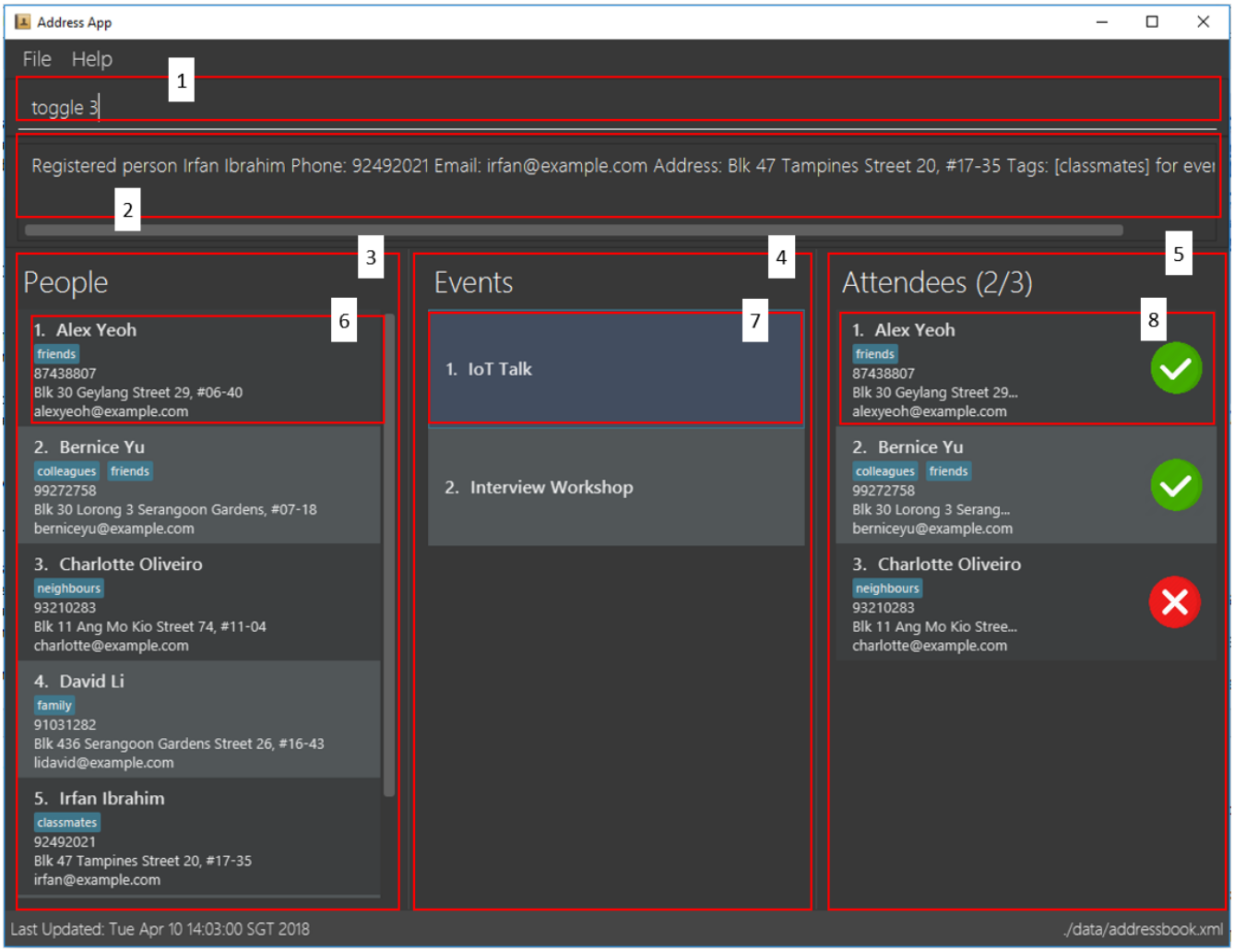

Figure 4. Overview of the Graphical User Interface of EPIC

1. Command Box: text box to receive user inputs.
2. Result Display: status bar that displays the result of user commands.

3. People List Panel: pane that lists all persons.
4. Event List Panel: pane that lists all events.
5. Attendance List Panel: pane that lists all registrants in the user selected event.
6. Person Card: a card that lists a person's contact information.
7. Event Card: a card that lists an event's details.
8. Attendance Card: a card that lists a registrant's contact information and displays if the registrant has attended or not.

NOTE	If no `EpicEvent` has been selected, the Attendance List Panel will be empty

Technical Overview

The UI component (Figure 5) uses the JavaFX UI framework. The UI consists of a `MainWindow` that is made up of parts e.g.`CommandBox`, `ResultDisplay`, `PersonListPanel`, `StatusBarFooter` etc. All these, including the `MainWindow`, inherit from the abstract `UiPart` class.

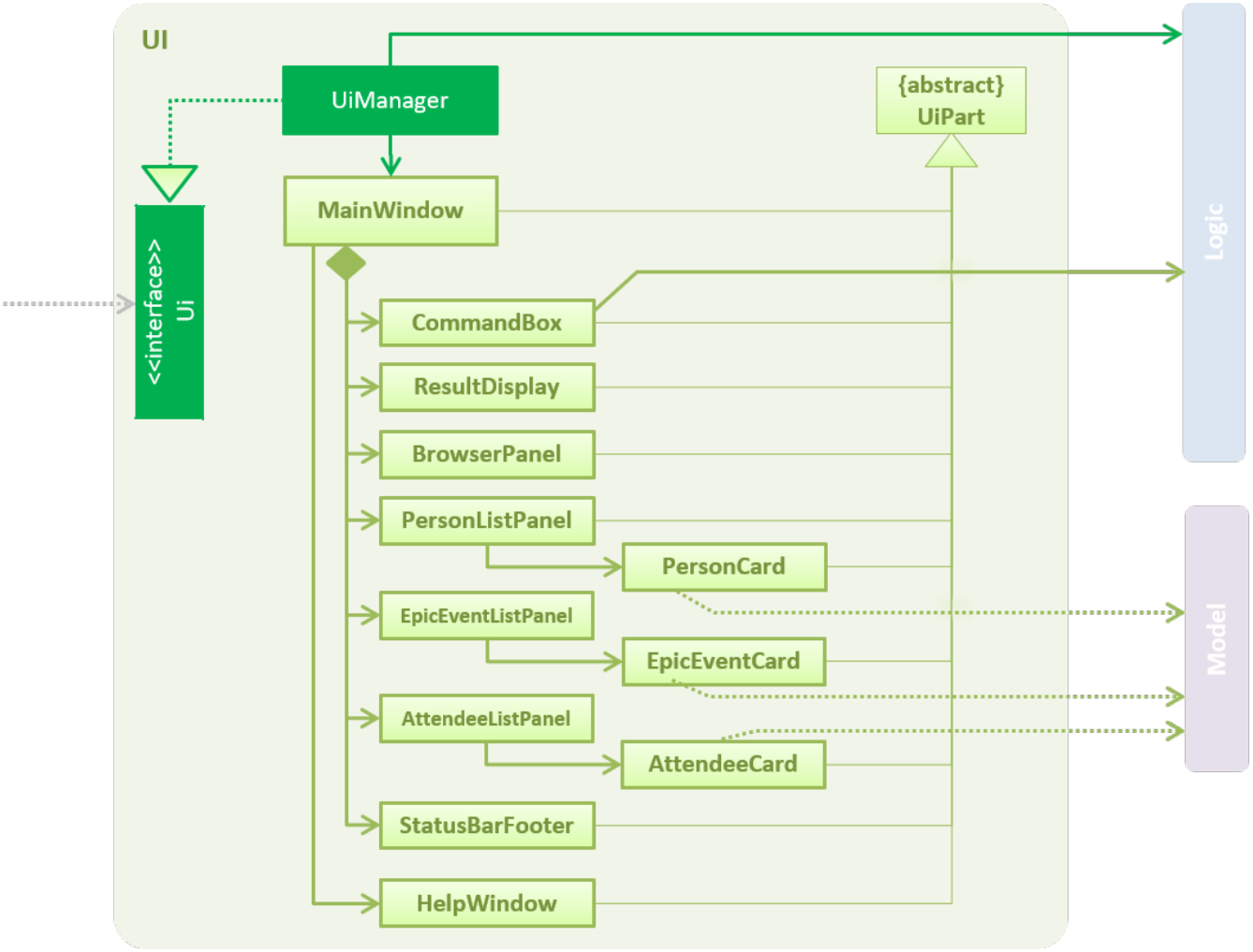

Figure 5. Structure of the UI Component

API : `Ui.java`

The layout of these UI parts is defined in matching `.fxml` files that are in the `src/main/resources/view` folder. For example, the layout of the `MainWindow` is specified in

`MainWindow.fxml`

Three-Pane UI

The new UI is a three-pane UI consisting of a list of persons, events, and registrants for the selected event. Being able to view all three panes will allow the user to read off the desired command parameters without having to switch between lists.

In addition, `edit` and `edit-command` commands are now implemented in a mutable manner (see [edit-curr-implementation]). As a result, the `ObservableList` that wraps the people and event data will no longer be notified of changes. The new UI must be notified of changes to data so it can refresh itself to reflect such changes. The UI will also have to respond to new commands such as the `select-event` and `toggle`. The `select-event` could invoke a change in the contents of the Attendance List Panel and `toggle` will toggle the attendance status and is reflected in the UI in the form of the attendance icon change.

Current Implementation

The UI now consists of 3 list panes: `PersonListPanel`, `EpicEventListPanel` and `AttendanceListPanel`. Each pane is a list of `PersonCard`, `EpicEventCard` and `AttendanceCard` respectively. Since the `AttendanceCard` has the same UI as `PersonCard` with the addition of the toggle icon, it extends `PersonCard`. The `model` maintains a `selectedEpicEvent` variable whose registrants will be displayed in the `AttendancePanel`.

In order to get the `AttendanceListPanel` to instantly update its elements, we employ the JavaFX Bean Pattern. We listen to changes to object properties by using the JavaBeans API to represent the properties.

The `Person` class is implemented this way:

```
public class Person extends SimpleObjectProperty {

    // ... Person logic ...

    /**
     * Edits this person by transferring the fields of dummyPerson over.
     * Used for the mutable edit command
     */
    public void setPerson(Person dummyPerson) {
        // ... setPerson logic ...
        fireValueChangedEvent();
    }
}
```

This way, editing `Person` calls `fireValueChangedEvent()` which informs JavaFX of the content change and triggers an UI refresh.

To listen to changes in attendance. We enable the `AttendanceList` to report changes on the element by providing a properties extractor. First, we use the JavaBeans API to represent properties of

`Attendance` that we want to listen to, which is the `hasAttendedEventProperty`, a `BooleanProperty` object.

We then use an Extractor which is a Callback containing an array of `Observables` which are then observed by the `ObservableList`.

```
Callback<Attendance, javafx.beans.Observable[]> extractor = attendance -> new
javafx.beans.Observable[] {
                attendance.getPerson(), attendance.getHasAttendedEventProperty()};
```

By using the JavaBeans API to handle UI updates, the programmer does not have to care about manually refreshing the UI upon data updates.

Design Considerations

When deciding on the UI, the following aspects of user experience were considered:

Aspect: Overall UI Design

- **Alternative 1 (current choice)**: A three-pane UI consisting of list of persons, events, and registrants for the selected event
  - Pros: The user can view all 3 lists at the same time. He would know what arguments to supply when typing commands as he can read them off the list.
  - Cons: The UI might become too cluttered as there are too many UI elements. However, given that EPIC is meant for modern computers with large displays, this should not be an issue.
- **Alternative 2 (previous choice)**: two-pane UI where the left pane is a 2 tab pane consisting of a list of persons and events, and the right pane is a list of registrants.
  - Pros: Merging the horizontal space for the list of persons and events will create more space for list of registrants. The user is likely to be more interested in the registrants' details.
  - Cons: If the user needs to access data for some tabbed pane that is not in focus to fill out a command, this would break his workflow. The user will have to delete his current command, execute a command to set focus to the desired tab, memorize the required details and reenter his previous command.
- **Alternative 3**: A common list that can display either list of persons, events or registrants for the selected event
  - Pros: We only have to make minimal changes to the UI layout.
  - Cons: Events, persons, and registrants must be displayed using the same `Card` class. This would result in tight coupling of the display graphics logic for the three lists.

Aspect: Updating UI when Data Changes

Previously, changes to `Person` would create a new `Person` object that would replace the previous object, triggering a UI refresh. Now that such changes to `Peron`, `EpicEvent` and `Attendance` objects only mutate it, the `ListView` is unable to listen to such changes.

- Alternative 1 (current choice): Use the JavaFX Beans Convention

    - Pros: Using the JavaBeans API to represent a property of an object allows property changes to be propagated to property change listeners. This is an elegant way to get the `ListView` to instantly update its elements.
    - Cons: Using the JavaBeans API to represent object properties introduces coupling between the Model and UI components.
- Alternative 2: Force all commands that change data to invoke a UI refresh
    - Pros: It is straightforward to force a `ListView` UI refresh by invoking its `refresh()` method.
    - Cons: We will have to ensure that any action that could modify data force a UI refresh. In addition, constantly invoking a UI refresh could become a resource hog.

Binding Data to Attendance Panel

- Alternative 1 (current choice): Wrap the selected `EpicEvent` in an `EpicEventObservable` object to listen for changes
    - Pros: We can listen to changes in the `EpicEventObservable` object to trigger a UI update.
    - Cons: Creating a new class introduces more code bloat.
- Alternative 1 (current choice): Bind `ObservableEpicEvent` to the Attendance Panel
    - Pros: As `ObservableEpicEvent` extends `Observable`, it can listen to when the selected event changes and update the Attendance Panel accordingly.
    - Cons: More code has to be written and maintained as we are introducing an additional layer of abstraction.
- Alternative 2: Bind `ObservableList<Attendance>` to the Attendance Panel
    - Pros: Binding data to the Attendance Panel is straightforward and the approach used to data to the People and Events Panels can be used here too.
    - Cons: Logic to handle the switching of selected events is moved up to the `MainWindow`, increasing coupling between the `MainWindow` and `AttendanceListPanel`. It is a violation of Single Responsibility Principle as the `MainWindow` should only be concerned with displaying the main layout.

Overview

NUSMods Planner is an automatic timetable generator, helping students to generate their timetable purely from specifications. More details can be found at my Github page