Bay Wei Heng - Project Portfolio

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 is optimized for event planners who prefer typing to using the mouse.

Summary of contributions

- **Major Enhancement**: Implementation of event-person interactions
  - Description: Persons can be registered for/deregistered from events, and a particular event’s participants can be displayed. These functions form the core of the application.
  - Design Decisions: There were various design decisions I had to make through the course of implementing these interactions. Two of the most notable ones are:
    - Editing of persons is done is a mutable manner.
    - Deletion of persons still registered for an event is not allowed.
  - Justification: I made the above choices for the following reasons:
    - Mutable edit - Allows editing of persons to be done in an efficient manner; events automatically receive the edited participant’s data, without the need to duplicate the data and update each event manually.
    - Deletion constraint - Acts as a safeguard to ensure that every participant’s data is present during event registration.
  - Highlights: This enhancement was done across multiple components:
    - Modified the Model component to allow navigability from events to persons
    - Added new commands to the Logic component, and provided their implementation
    - Ensured that the UI component updates whenever a person is edited, even when it is done in a mutable manner, and similarly when a person is registered for/deregistered from an event.
- **Minor Enhancement**: Overhaul of undo-redo functionality
  - Previous Implementation: In the previous iteration of the product, the **entire** memory contents of the application were duplicated and stored to facilitate undoing/redoing commands.
  - Current Implementation: In EPIC v1.5, *every* undoable command now learns how to undo/redo itself in the pre-processing stage. This knowledge is stored alongside the command.

    - Justification: The new implementation only requires essential, and hence minimal, information to be stored, **significantly** reducing memory consumption of EPIC.
- **Code contributed**: [Functional code] [Test code]
- **Other contributions**:
    - Project management:
        - Set up team repository on GitHub
        - Set up continuous integration tools such as Travis CI and Appveyor
        - Set up coverage checkers such as Coveralls
        - Managed releases v1.2, v1.3, and v1.4 on GitHub
    - Documentation:
        - Editor for the user and developer guides, ensuring consistency and checking for language errors. Relevant PRs: #103 and #147
        - Added various example outcomes for previous commands in the User Guide
    - Community:
        - PRs reviewed (with non-trivial review comments): #41
        - Posted bugs of teammates as issues on GitHub: #107 and #155
        - Reported bugs and suggestions for other teams in the class. Relevant issues on GitHub: #110 (has follow-up with team), #89 (suggested fix), #112

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.

Interacting with the GUI

- Clicking on an event in the Events Pane selects that event, allowing you to see all persons registered for that event in the Attendees Pane, as well as perform actions such as toggling attendance. Selecting an event can also be done with the `select` command in the command box.
- Clicking on the tick/cross symbol next to a person's details in the Attendees Pane allows you to toggle that person's attendance for the currently selected event. Toggling the attendance can also be done with the `toggle` command.

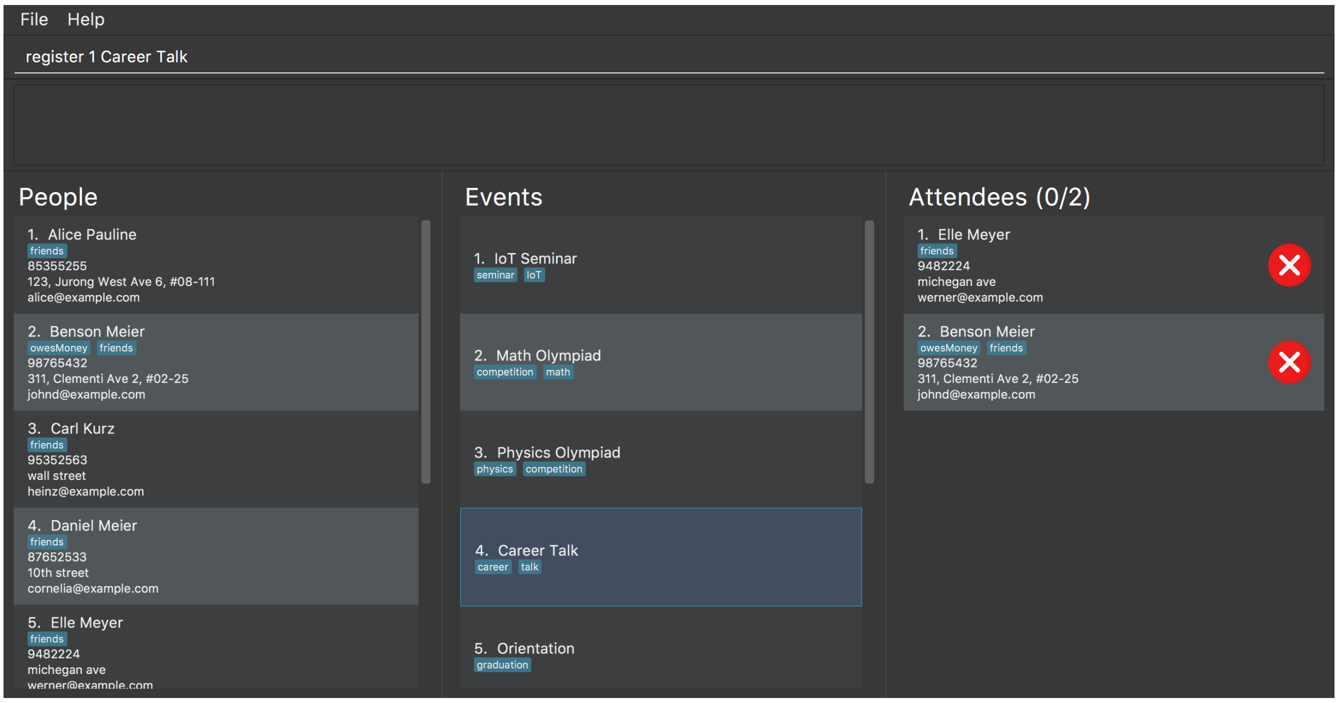
Registering a Person for an Event: `register`

Registers the specified person for an event in EPIC.
Format: `register INDEX EVENT_NAME`

- Registers the person at the specified `INDEX`. The index refers to the index number shown in the Persons Pane.
- The index **must be a positive integer**: 1, 2, 3, …
- `EVENT_NAME` must match the name of an event in EPIC exactly.

Examples:

- `find Betsy`
  `register 1 Computing Seminar`
  Registers the 1st person in the results of the `find` command for Computing Seminar.
- `list`
  `register 2 AY201718 Graduation`
  Registers the 2nd person in EPIC for AY201718 Graduation. See the figures below for UI changes.



register 1 Career Talk

People

1. Alice Pauline
friends
85355255
123, Jurong West Ave 6, #08-111
alice@example.com

2. Benson Meier
owesMoney friends
98765432
311, Clementi Ave 2, #02-25
johnd@example.com

3. Carl Kurz
friends
95352563
wall street
heinz@example.com

4. Daniel Meier
friends
87652533
10th street
cornelia@example.com

5. Elle Meyer
friends
9482224
michegan ave
werner@example.com

Events

1. IoT Seminar
seminar IoT

2. Math Olympiad
competition math

3. Physics Olympiad
physics competition

4. Career Talk
career talk

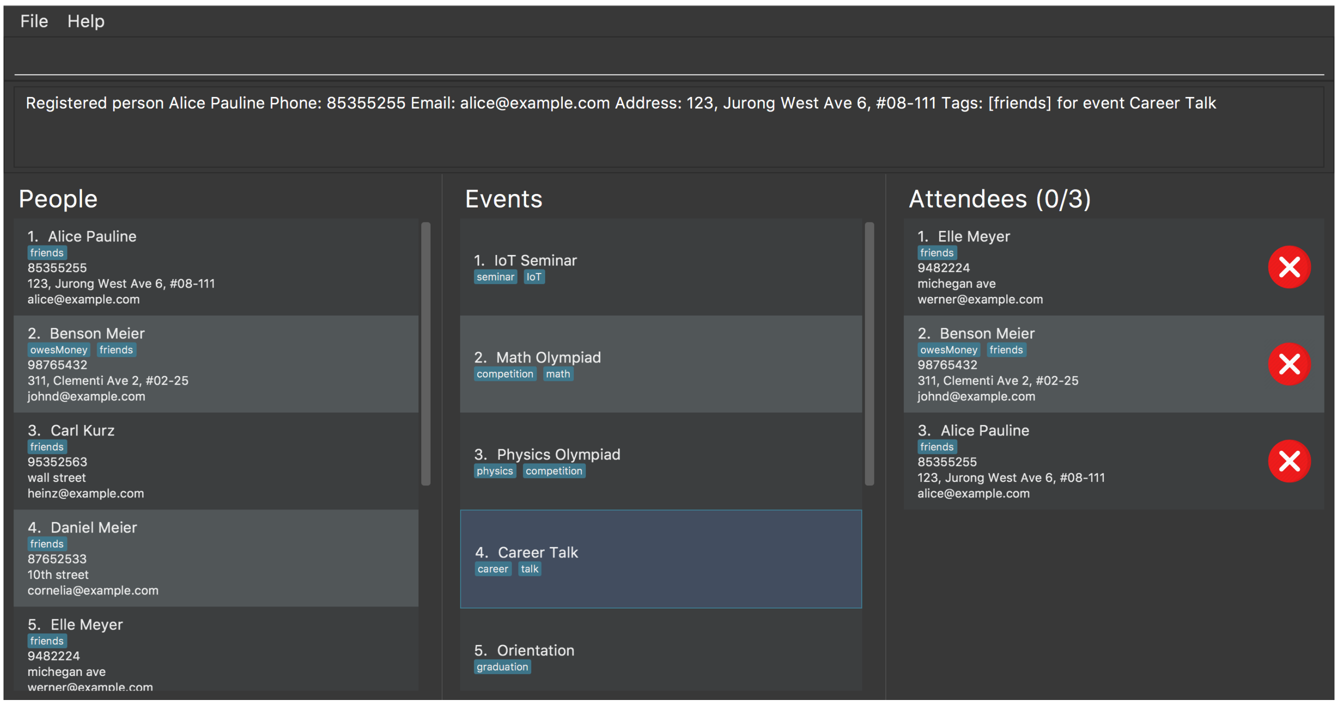
5. Orientation
graduation

Attendees (0/2)

1. Elle Meyer
friends
9482224
michegan ave
werner@example.com

2. Benson Meier
owesMoney friends
98765432
311, Clementi Ave 2, #02-25
johnd@example.com

Figure 1. Before the `register` command is executed



Registered person Alice Pauline Phone: 85355255 Email: alice@example.com Address: 123, Jurong West Ave 6, #08-111 Tags: [friends] for event Career Talk

People

1. Alice Pauline
friends
85355255
123, Jurong West Ave 6, #08-111
alice@example.com

2. Benson Meier
owesMoney friends
98765432
311, Clementi Ave 2, #02-25
johnd@example.com

3. Carl Kurz
friends
95352563
wall street
heinz@example.com

4. Daniel Meier
friends
87652533
10th street
cornelia@example.com

5. Elle Meyer
friends
9482224
michegan ave
werner@example.com

Events

1. IoT Seminar
seminar IoT

2. Math Olympiad
competition math

3. Physics Olympiad
physics competition

4. Career Talk
career talk

5. Orientation
graduation

Attendees (0/3)

1. Elle Meyer
friends
9482224
michegan ave
werner@example.com

2. Benson Meier
owesMoney friends
98765432
311, Clementi Ave 2, #02-25
johnd@example.com

3. Alice Pauline
friends
85355255
123, Jurong West Ave 6, #08-111
alice@example.com

Figure 2. After the `register` command is executed

Deregistering a Person from an Event: `deregister`

Deregisters the specified person from an event in the EPIC.
Format: `deregister INDEX EVENT_NAME`

- Deregisters the person at the specified `INDEX`. The index refers to the index number shown in the Persons Pane.
- The index **must be a positive integer**: 1, 2, 3, …
- `EVENT_NAME` must match the name of an event in EPIC exactly.
- The person to be deregistered must be already in the event.

Examples:

- `list`
  `deregister 2 AY201718 Graduation`
  Deregisters the 2nd person in EPIC from AY201718 Graduation.
- `find Betsy`
  `deregister 1 Computing Seminar`
  Deregisters the 1st person in the results of the `find` command from Computing Seminar.

Listing all Event Participants: `list-registered`

Lists all participants for the specified event.
Format: `list-registered EVENT_NAME`

- `EVENT_NAME` must match the name of an event in EPIC exactly.

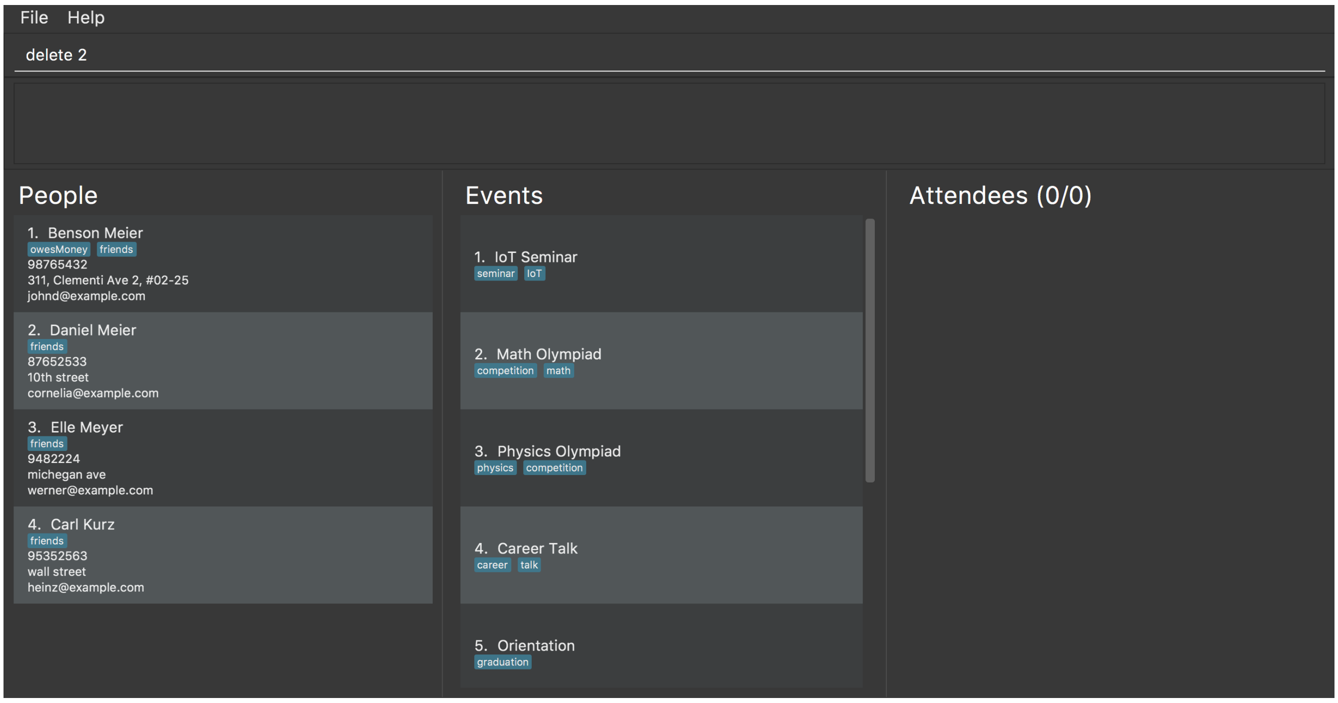
Undoing Commands : `undo`

Restores EPIC to the state before the previous *undoable* command was executed.
There is no guarantee that the relative order of persons/events will be preserved after the undoing of a `delete` or `delete-event` command.
Format: `undo`

NOTE Undoable commands: those commands that modify EPIC's content. To see the full list of undoable commands, refer to [List of undoable commands]

Examples:

- `select 1`
  `list`
  `undo`
  The `undo` command fails since neither `list` nor `select` is undoable.
- `delete 1`
  `clear`
  `undo`
  `undo`
  Undoes the `clear` command, followed by the `delete 1` command.
- `delete 2`
  `undo`
  Undoes the `delete 2` command. See the figures below for UI changes. In particular, After the `undo` command is executed shows that relative order of persons might not be preserved.



delete 2

1. Benson Meier
owesMoney friends
98765432
311, Clementi Ave 2, #02-25
johnd@example.com

2. Daniel Meier
friends
87652533
10th street
cornelia@example.com

3. Elle Meyer
friends
9482224
michegan ave
werner@example.com

4. Carl Kurz
friends
95352563
wall street
heinz@example.com

1. IoT Seminar
seminar IoT

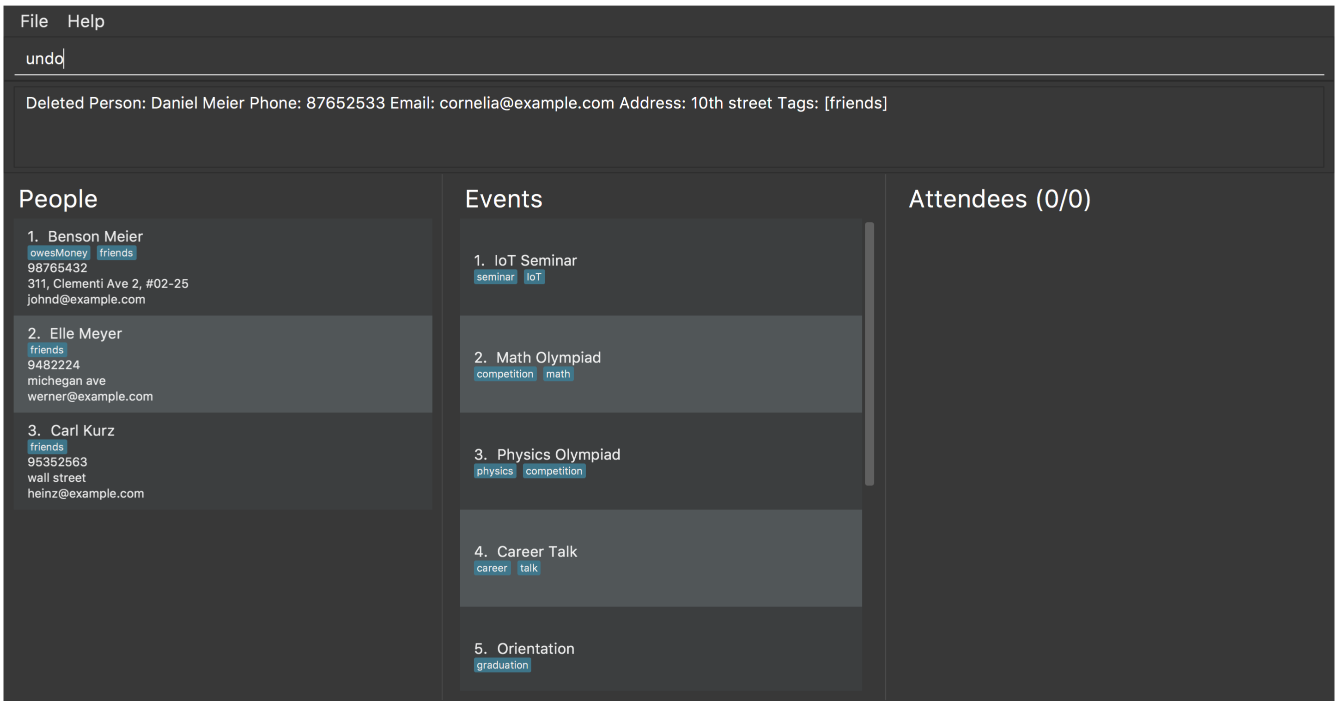
2. Math Olympiad
competition math

3. Physics Olympiad
physics competition

4. Career Talk
career talk

5. Orientation
graduation

Figure 3. Before the `delete 2` command is executed



undo

Deleted Person: Daniel Meier Phone: 87652533 Email: cornelia@example.com Address: 10th street Tags: [friends]

People

1. Benson Meier
owesMoney friends
98765432
311, Clementi Ave 2, #02-25
johnd@example.com

2. Elle Meyer
friends
9482224
michegan ave
werner@example.com

3. Carl Kurz
friends
95352563
wall street
heinz@example.com

Events

1. IoT Seminar
seminar IoT

2. Math Olympiad
competition math

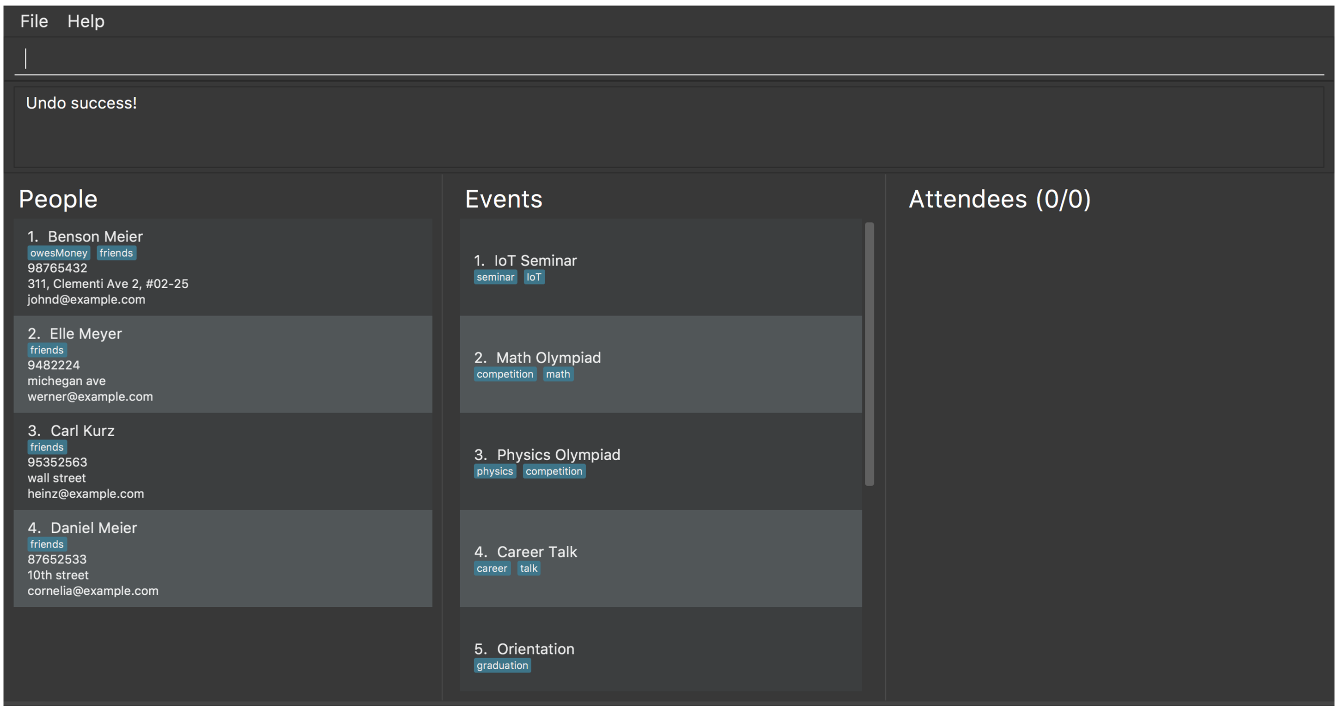
3. Physics Olympiad
physics competition

4. Career Talk
career talk

5. Orientation
graduation

Attendees (0/0)

Figure 4. After the `delete 2` command is executed



Undo success!

People

1. Benson Meier
owesMoney friends
98765432
311, Clementi Ave 2, #02-25
johnd@example.com

2. Elle Meyer
friends
9482224
michegan ave
werner@example.com

3. Carl Kurz
friends
95352563
wall street
heinz@example.com

4. Daniel Meier
friends
87652533
10th street
cornelia@example.com

Events

1. IoT Seminar
seminar IoT

2. Math Olympiad
competition math

3. Physics Olympiad
physics competition

4. Career Talk
career talk

5. Orientation
graduation

Attendees (0/0)

Figure 5. After the `undo` command is executed

Redoing Commands : `redo`

Reverses the most recent `undo` command. This will **fail** if the most recent command was neither `undo` nor `redo` (See last example below)
Format: `redo`

Examples:

- `delete 1`
  `undo`

`redo`
Redoes the `delete 1` command.

- `delete 1`
  `redo`
  The `redo` command fails as there are no `undo` commands executed previously.
- `delete 1`
  `clear`
  `undo`
  `undo`
  `redo`
  `redo`
  Redoes the `delete 1` command, followed by the `clear` command.
- `add-event n/MyEvent`
  `undo`
  `add-event n/MyEvent`
  `redo`
  The `redo` command fails as the most recent command was not an `undo/redo` command. This prevents ill-defined behavior like trying to add back an event that already exists.

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.

Edit Person/Event Feature

Previous Implementation

In AddressBook-Level4, the `edit` command was performed by creating a new `Person` and passing it to a `UniquePersonList` in the model, which would then replace the to-be-edited `Person` with it.

```
    int index = internalList.indexOf(personToEdit);
    Person editedPerson = new Person(name, ...);
    internalList.set(index, editedPerson);
```

Current Implementation

In EPIC, the `edit` and `edit-event` commands are now implemented in a mutable manner - instead of replacing the to-be-edited `Person/EpicEvent` with the new one, we edit the details of the to-be-edited `Person/EpicEvent` directly.

```
    int index = internalList.indexOf(personToEdit);
    Person editedPerson = new Person(name, ...);
    internalList.get(index).setPerson(editedPerson); // setPerson edits internal
details using those of the supplied Person
```

Design Considerations

Aspect: Implementation of `edit`

- **Alternative 1 (current choice):** Edit in a mutable manner
    - Pros: Since EPIC has both `EpicEvent` and `Person` objects, which maintain references to one another, editing a `Person/EpicEvent` in this manner automatically updates the `EpicEvent/Person` objects that is associated with it.
    - Cons: Implementation of `undo` will be more difficult.
- **Alternative 2:** Edit in an immutable manner
    - Pros: Implementation of `undo` is easier, since we can just replace the current `EventPlanner` with the previous one.
    - Cons: Editing a `Person/EpicEvent` will require passing a copy of the newly-created `Person/EpicEvent` to all objects associated with the to-be-edited version, introducing significant overhead

Changes from Previous Implementation

Instead of saving the entire event planner each time we execute an `UndoableCommand`, each `UndoableCommand` knows how to `undo/redo` itself. Each `UndoableCommand` has an `oppositeCommand` field, which is another `UndoableCommand` that, when executed, reverses the changes made by the original command. The sequence diagram for the new `undo()` implementation is shown in Sequence Diagram for new undo implementation.

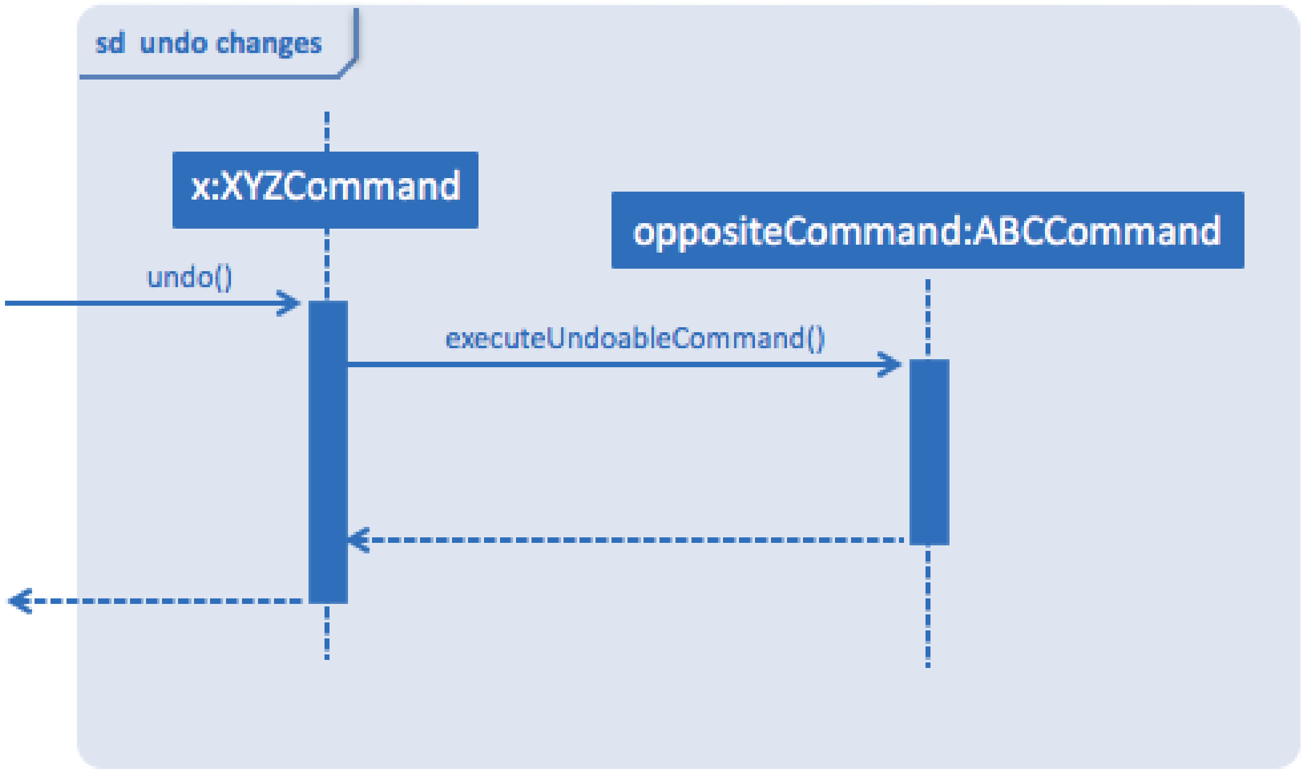


Figure 6. Sequence Diagram for new undo implementation

The `oppositeCommand` is generated in the `execute()` method, after `preprocessUndoableCommand()`. This is because generating the `oppositeCommand` requires knowledge of the actual `Person/EpicEvent` objects to be modified. For example, the `oppositeCommand` for a `deletePersonCommand` is an `addPersonCommand`, but we only know the person to be deleted after the pre-processing step.

NOTE Each `UndoableCommand` now requires its individual `generateOppositeCommand()` implementation. Hence, this method is made abstract in the abstract class `UndoableCommand`

There was no `Command` that could easily reverse the changes of a `ClearCommand`, hence a new `Command` `RestoreCommand` had to be created. Since the sole purpose of this command is to serve as the `oppositeCommand` of a `ClearCommand`, this command is not directly accessible to the user, and can only be executed when the user undoes a `ClearCommand`.

Design Considerations

Aspect: How Undo and Redo Executes

- **Alternative 1 (current choice):** Store the minimal knowledge required to undo each command inside itself.
  - Pros: Significantly less memory is used (e.g. for `delete`, just save the person being deleted). Compatible with mutable commands.
  - Cons: Implementation is more complicated.
- **Alternative 2:** Save the entire event planner after every undoable command.
  - Pros: Implementation is easy.
  - Cons: Performance issues may result due to high memory usage. Also, this is incompatible

with the mutable `edit` and `edit-event` implementations.