Calvin Chen Xingzhu - Project Portfolio

PROJECT: AddMin+

Overview

My team of 4 software engineering students and I were tasked with enhancing a basic command line interface desktop addressbook application for our Software Engineering project. We chose to morph it into an employee records management cum communication system called AddMin+. This enhanced application enables office managers to file and recall employee data; manage employee work schedule and leave application; and email employees directly without opening an email application.

This is what our project looks like:

Figure 1. The graphical user interface for AddMin+.

My role was to design and write the codes for the fetch and allocate features. The following sections illustrate these enhancements in more detail, as well as the relevant documentation I have added to the user and developer guides in relation to these enhancements.

Summary of contributions

- **Major enhancement**: added features to allow association of employees with events which includes **automated/manual allocation of employees to event commands** and a **deallocate command**, as well as the viewing of changes the users made via the **fetch event** command. All related commands have **GUI features** implemented.
    - What it does: allows the user to check for availability of employees and allocate them to an event with certain filter requirements. The algorithm ensures that employees cannot be allocated to events with conflicting time periods.
    - Justification: This feature improves the product significantly because a user can now allocate employees to events, which is a key component for events management companies that our app is targeting. The GUI features further provide convenience for users.
    - Highlights: This enhancement affects existing commands and commands to be added in future. It required an in-depth analysis of design alternatives. The implementation too was challenging as it required the knowledge of both `Event` and `Employee` classes.
- **Minor enhancement**: added command history to allow user to view previous commands using up/down arrow keys.
- **Code contributed**: [Functional code] [Test code] *{give links to collated code files}*
- **Other contributions**:
    - Project management:
        - Ensured integration of various components (e.g. storage, logic, model) during the morphing phase.
        - Managed releases `v1.3` - `v1.5rc` (3 releases) on GitHub
    - Enhancements to existing features:
        - Updated GUI to show both employee and event list
    - Documentation:
        - Did cosmetic tweaks to existing contents of the User Guide: #14
    - Community:
        - PRs reviewed (with non-trivial review comments): #12, #32, #19, #42
        - Reported bugs and suggestions for other teams in the class (examples: 1, 2, 3)

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.

Event-Specific Management

Automated allocation of Employees to Events: `allocate` (also a GUI feature)

Automatically chooses and allocates employees that meet the requirements to events .

Format: `allocate EVENT_INDEX [n/NUMBER] [t/TAG]`

TIP Fields in [] are optional. Random selection of employee to allocate if supply exceeds demand of event.

- Allocates a `NUMBER` of employees to the event at the specified `EVENT_INDEX` filtered based on `TAG`.
- The `EVENT_INDEX` refers to the index number shown in the displayed event list.
- The `NUMBER` refers to the number of employees to be allocated to the event.
- Both `EVENT_INDEX` and `NUMBER` **must be a positive integer** 1, 2, 3, …
- If no `NUMBER` is specified, it is assumed to be the current manpower count required by the event.

Examples:

- `allocate 1`
  Allocates available employees to the 1st event.
- `allocate 2 n/3 t/female`
  Allocates 3 employees who are tagged as 'female' to the 2nd event.

Manually allocation of Employees to Events: `allocatem`

Manually chooses and allocates employees to events.

Format: `allocatem EVENT_INDEX n/EMPLOYEE_INDEX`

- Allocates an employee with `EMPLOYEE_INDEX` to the event at the specified `EVENT_INDEX`.
- The `EVENT_INDEX` refers to the index number shown in the displayed event list.
- The `EMPLOYEE_INDEX` refers to the index number shown in the displayed employee list.
- Both `EVENT_INDEX` and `EMPLOYEE_INDEX` **must be a positive integer** 1, 2, 3, …

Examples:

- `allocatem 1 n/2`
  Allocates the 2nd employee on the employee list to the 1st event on the event list.

Fetch Full Details of an Event: `fetch_ev` (also a GUI feature)

Fetches an event by displaying a pop-up window with full details of the event.

Format: `fetch_ev EVENT_INDEX`

> - The `EVENT_INDEX` refers to the index number shown in the displayed event list.
> - The `EVENT_INDEX` **must be a positive integer** 1, 2, 3, …

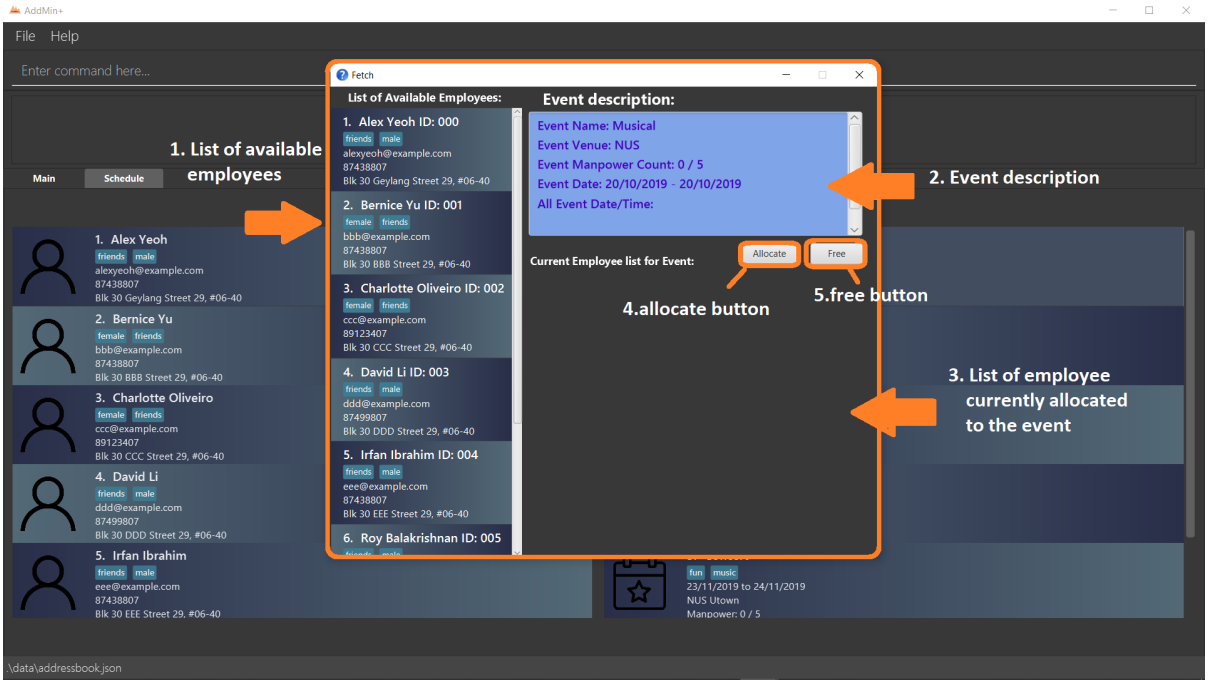
Examples:

- `fetch_ev 2`
  Returns the 2rd event from the event list

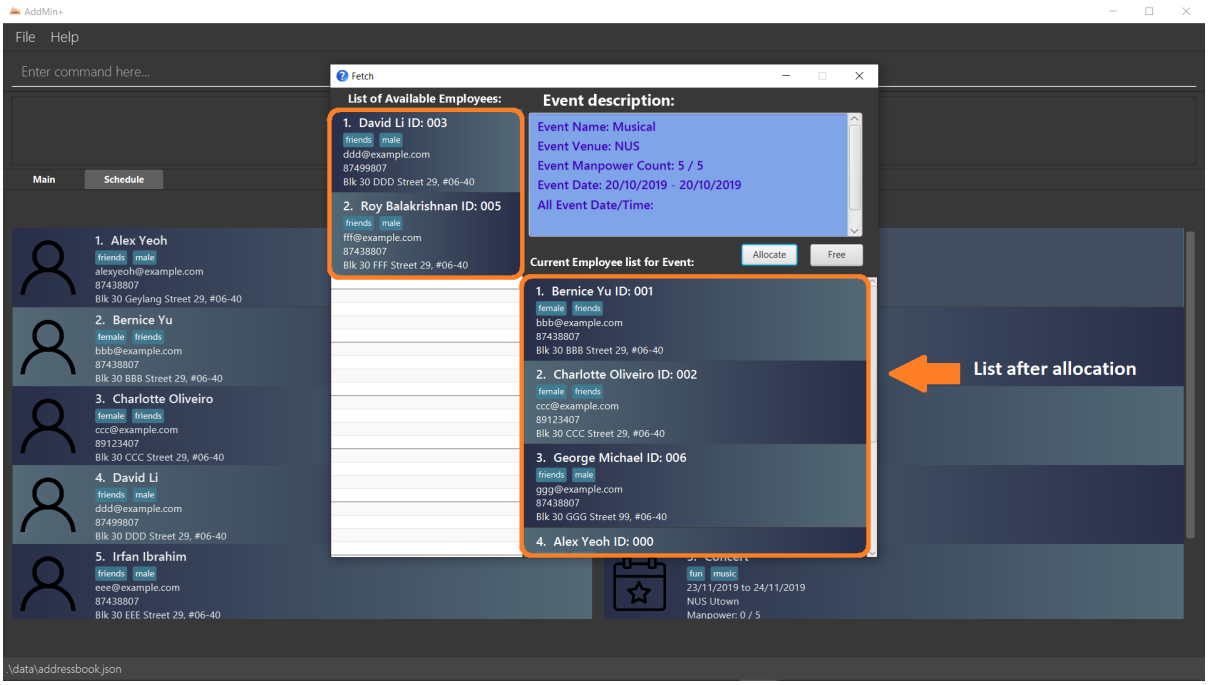
GUI Guide for event fetch and allocation commands

Step 1. For `fetch_ev`, simply **double-click the event** in the list as shown in the figure below:

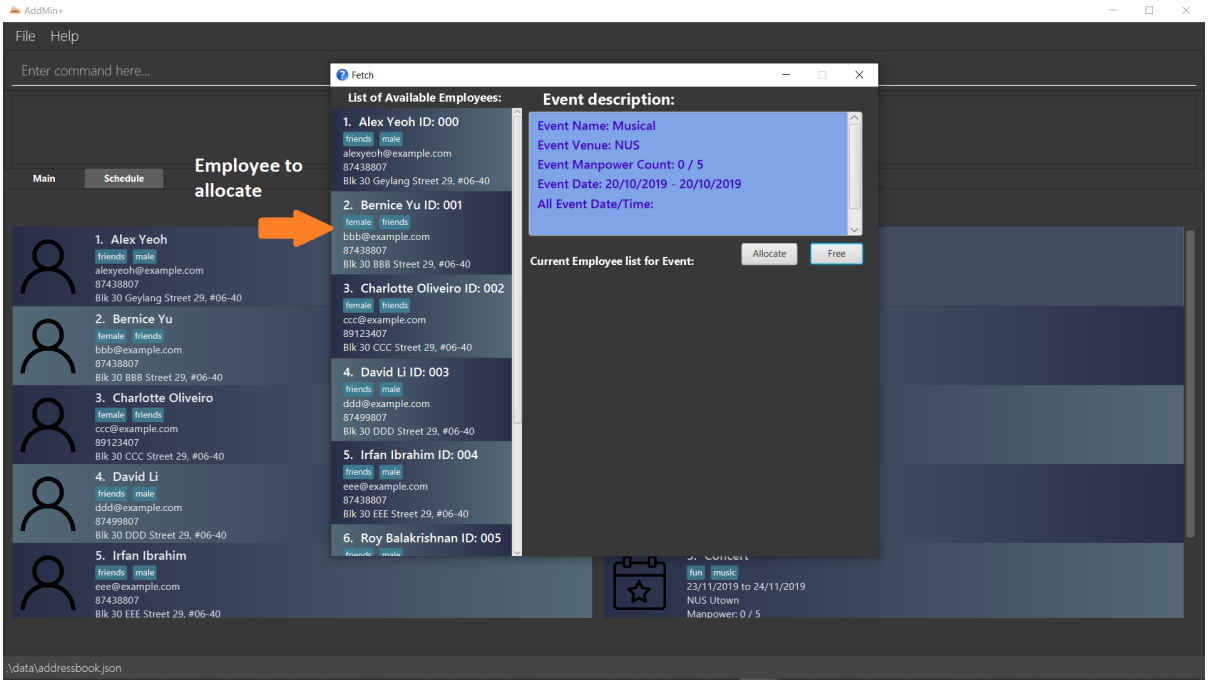
Step 2: After successfully fetching the event, the following **Fetch Window** should show:



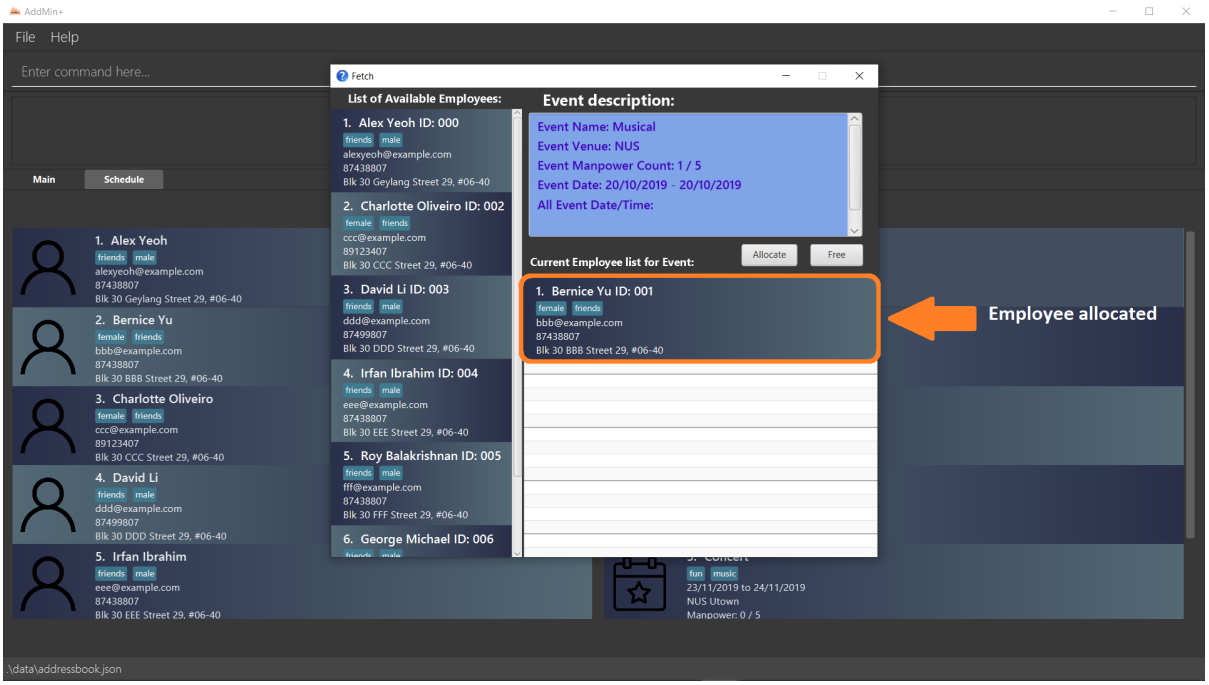
Step 3: To perform a `allocate` command without number/filter specification, click the **allocate** button as shown in the **Fetch Window** in step 2. The two lists will be updated again as shown in the following figure:



Step 4: To perform a `free` command, click the **free** button as shown in the **Fetch Window** in step 2. The two lists will be updated as shown in the following figure. Now, if you are interested to allocate a particular employee to an event, continue to step 5.



Step 5: To **allocate** a particular employee to an event, double-click the employee card on the left list. Notice the employee to allocate has moved to the list on the right as shown in the figure below:



Step 6: Finally, to **free** a particular employee to an event, double-click the employee card on the right as shown in the figure above in step 5:

NOTE	If the GUI features are not working as intended, kindly use the command line interface to execute the command instead.

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.

Storage component

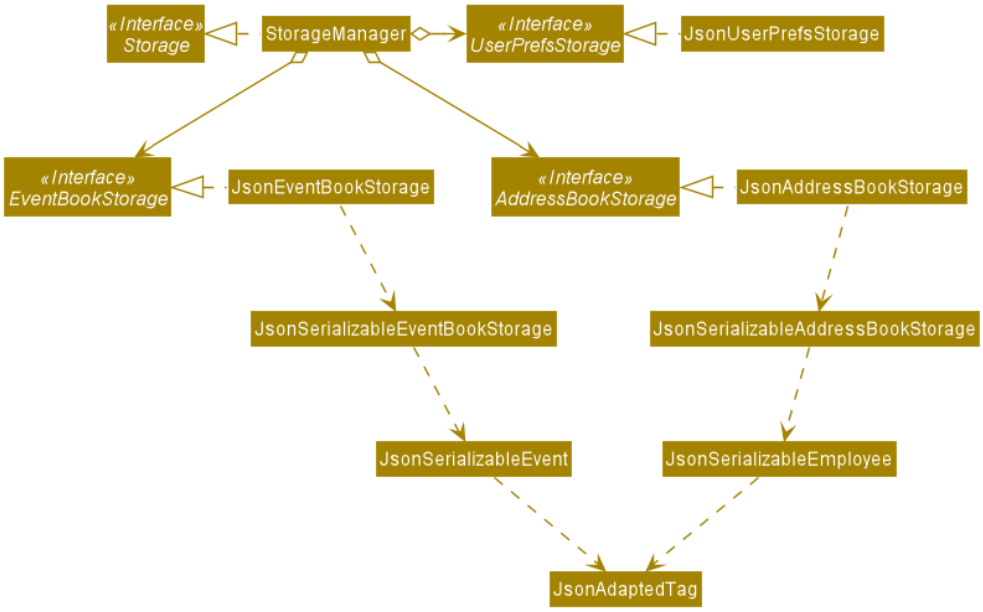


Figure 2. Structure of the Storage Component

API : `Storage.java`

The `Storage` component,

- can save `UserPref` objects in json format and read it back.
- can save the App data in json format and read it back.

Automated allocation of Employees to Events feature

Implementation

The `AutoAllocateCommand` has an auto-allocation mechanism which is facilitated by methods in `Event`. The `AutoAllocateCommand` takes in three arguments:

1. `eventIndex` - index of event in the displayed event list
2. `ManpowerCountToAdd` - number of employees to allocate [optional]
3. `tagList` - a set of tags to filter the employees [optional]

Additionally, the `AutoAllocateCommand` uses the following operations:

- `Event#isAvailableForEvent()` — Checks if an employee is available for the event.
- `AutoAllocateCommand#createAvailableEmployeeListForEvent()` — Creates a list of employees available for the event, filtered by the tags specified by user.
- `AutoAllocateCommand#getManpowerNeededByEvent()` — Calculates the number of employees currently required by the event.
- `AutoAllocateCommand#createEventAfterManpowerAllocation()` — Creates a new event with a updated manpower list.

Given below is an example usage scenario and how the auto allocation mechanism behaves at each step.

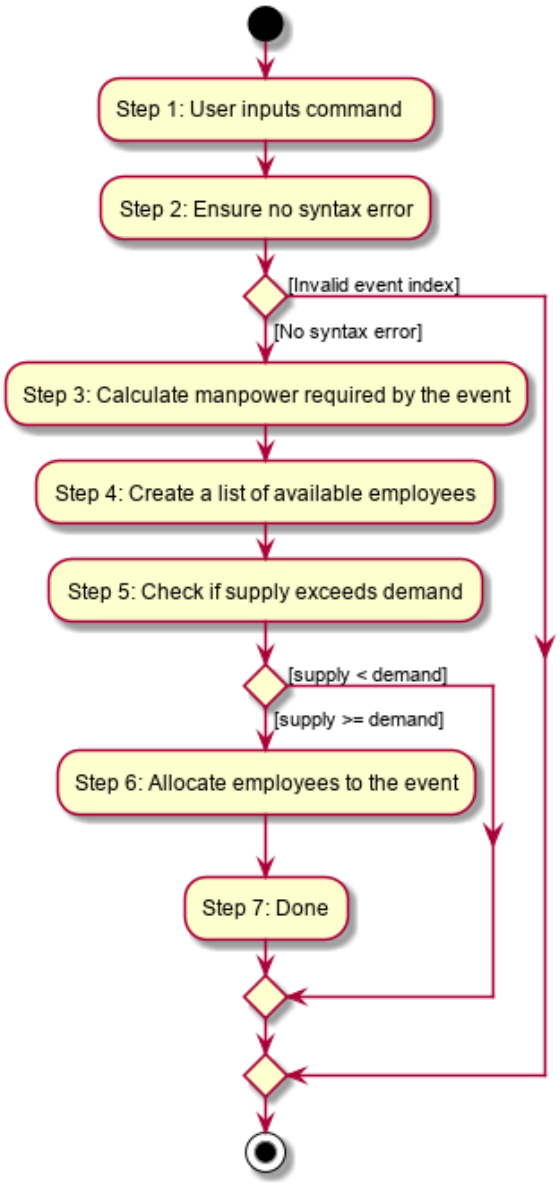


Figure 3. Program flow of the Auto Allocate Feature

Step 1. The user executes `allocate 1 n/2 t/female` with the intention to allocate 2 employees with tag [female] to the 1st event displayed in the event list.

Step 2. The command checks if `eventIndex` is valid and if `ManpowerCountToAdd` is specified.

NOTE	If `ManpowerCountToAdd` is not specified, it is assumed to be the maximum number possible for the event. Validity of other command arguments e.g. if `ManpowerCountToAdd` is a positive integer is checked by `AutoAllocateCommandParser` and not within the command `AutoAllocateCommand`.

Step 3. The command calls its own method `AutoAllocateCommand#getManpowerNeededByEvent()` to get the number of employees required by the specified event.

Step 4. The command calls its own method `AutoAllocateCommand#createAvailableEmployeeListForEvent()` to create a filtered list of employees

based on the `tagList` and if employee satisfies `Event#isAvailableForEvent()`.

Step 5. The command checks if supply (number of employees in filtered list in step 4) exceeds demand (number of employees required by event, generated in step 3).

NOTE	If demand exceeds supply, an exception will be thrown to the user. If the supply exeeds demand, employees will be randomly selected instead.

Step 6. The command calls `Event#createEventAfterManpowerAllocation()` to create a new event with a updated manpower list.

NOTE	For storage purposes, only the `Employee#EmployeeId` is saved in the event's manpower list.

Step 7. Done.

The following sequence diagram shows how the auto allocation works:

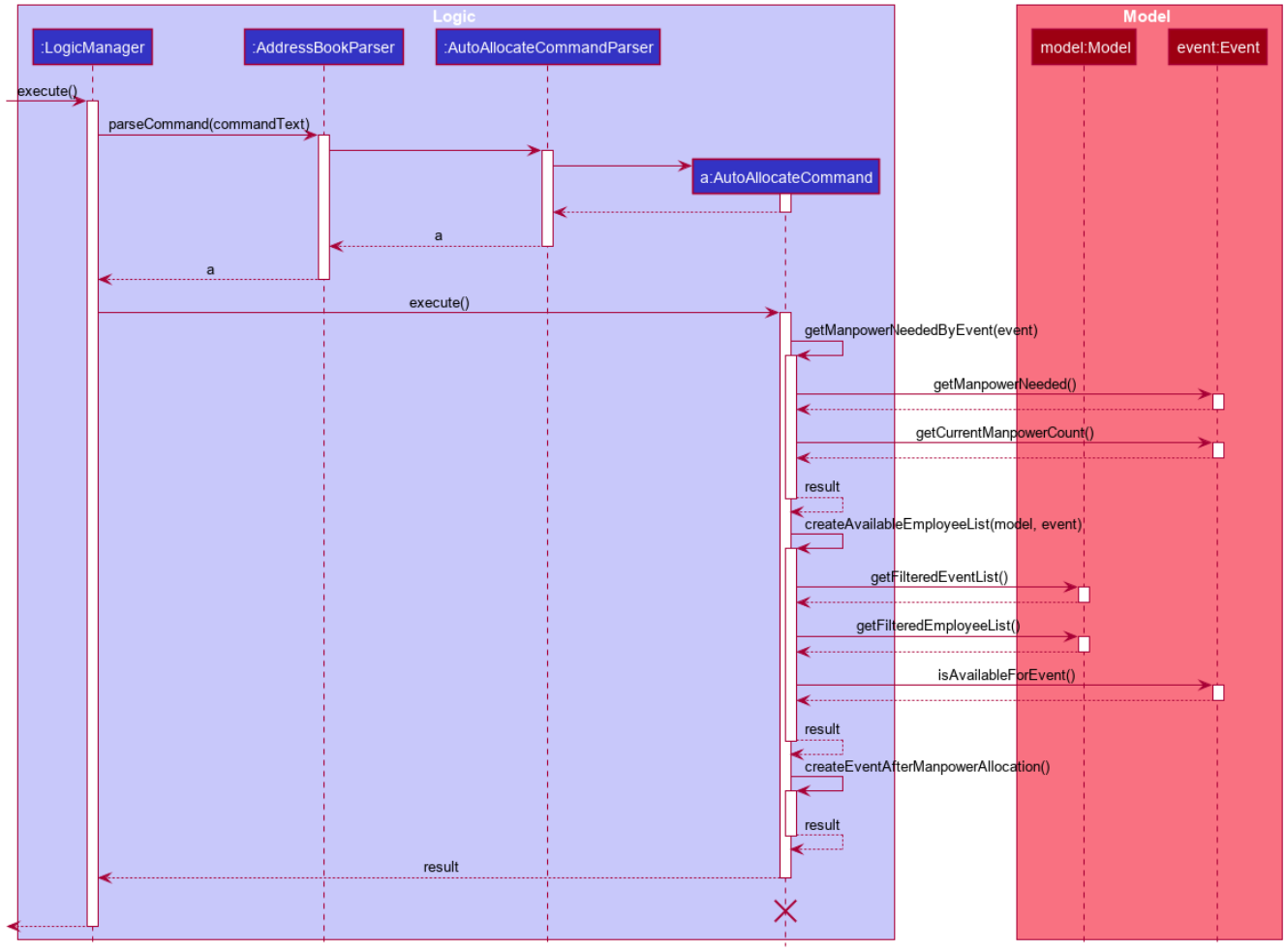


Figure 4. Sequence Diagram of the AutoAllocate Command

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

Design Considerations

Aspect: Storage of employees associated with event after successful command

Feature	Alternative 1	Alternative 2
Storage of employees associated with event after successful command	Saves only the `Employee#EmployeeId` associated with the event. **Pros**: Easy to implement. Will use less memory. **Cons**: Future accesses require more time. I decided to proceed with this option because it creates less dependencies.	Saves all fields of `Employee` associated with the event. **Pros**: Easy retrieval in the future. **Cons**: Changes in `Employee` attributes have to be reflected in the event. This meant that `EditCommand` and `DeleteCommand` for `Employee` have to be heavily modified.
How to update the changes in the manpower list of an event after allocation of employee.	Directly modifies the `EventManpowerAllocatedList` of the specified event **Pros**: Easy to implement. **Cons**: May cause unwanted behaviours if testing is not done properly.	Create a new event with a newly created and updated manpower list. **Pros**: Defensive programming. **Cons**: Harder to implement. I decided to proceed with this option because it complies with the Law of Demeter which states that objects should not navigate internal structures of other objects.

PROJECT: DUKE

{Optionally, you may include other projects in your portfolio.}