Nguyen Dang Huu - Project Portfolio

# PROJECT: VolunCHeer

## Overview

This project portfolio page serves to document my contributions to the VolunCHeer project.

VolunCHeer is an open-sourced Command Line Interface (CLI) management application designed for Volunteering Project Managers. We aim to help our target users alleviate the haystack of managing multiple projects, beneficiaries, and volunteers in an efficient and effective manner.

Team T08-1 consists of Zhao Jun Ru, Wong Kai Wen Jeremy, Liow Zhu Hui and myself.
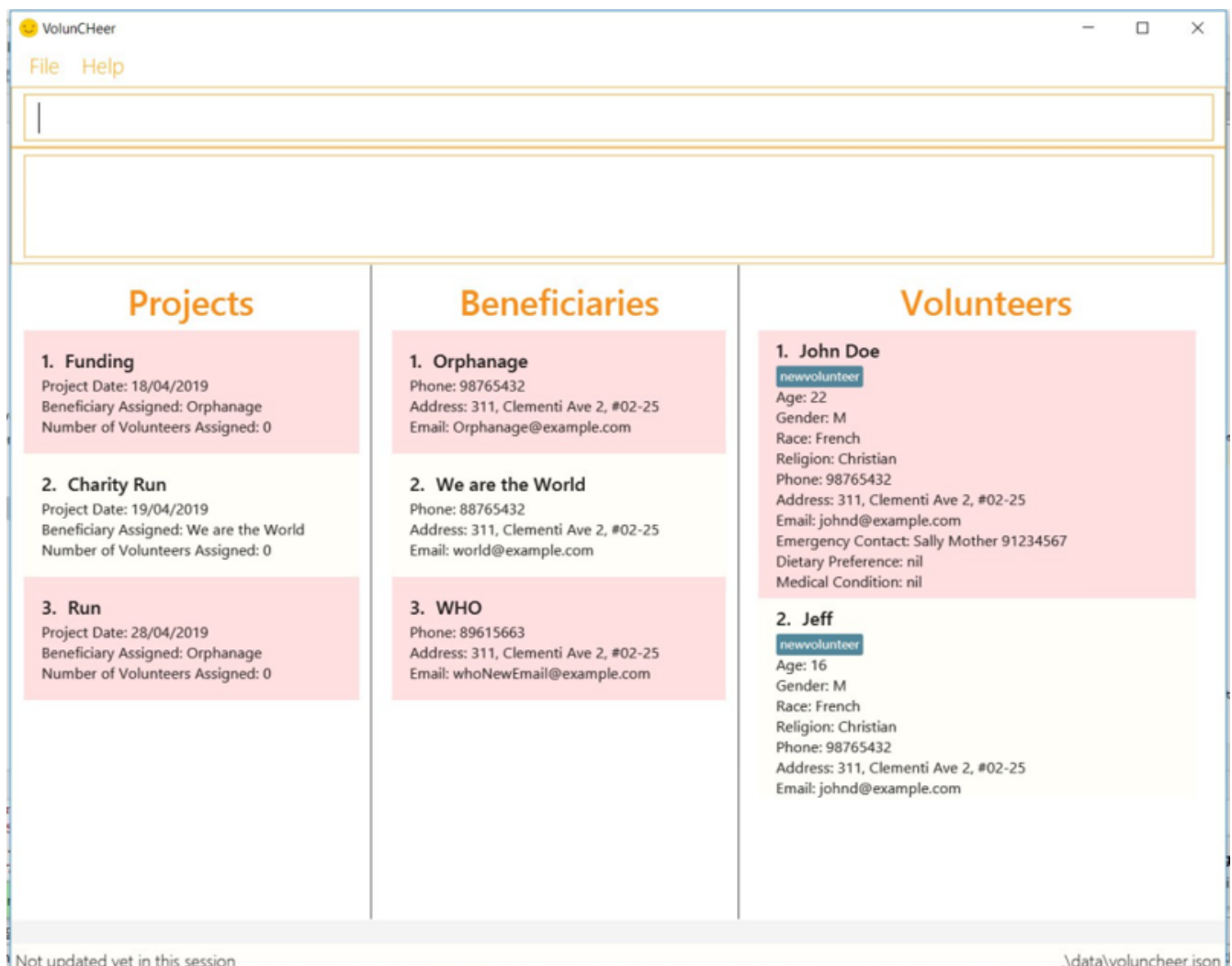


The user interacts with VolunCHeer using a command-line interface, and has a GUI created with JavaFX. VolunCHeer is written in Java, and has about 10 kLoC.

VolunCHeer's codebase draws from SE-EDU's AddressBook Level 4.

# Role

I am the Lead Developer, who ensure the application is bug free, or report bugs and assign developers to fix them. I am also the adviser for implementation of commands and bug fixes.

# Summary of contributions

- **Major enhancement**: added **Beneficiary management feature**

  - What it does: Beneficiary Management feature allows the user to manage beneficiary, including:

    - add a beneficiary: to open a new record of a beneficiary.

    - edit a beneficiary: to edit an existing record of a beneficiary.

    - find a beneficiary by name: to help the user quickly navigate to the wanted beneficiary record.

    - list beneficiaries: to list all beneficiary records.

    - summarise beneficiaries' data: to enable the user to see most active beneficiary based on attached project list.

  - Justification: This feature improve the product significantly because:

    - The beneficiary management feature allows the dynamic of tracking beneficiary records for the user to revise, and reuse every time a project initiation process starts with a known beneficiary.

    - The feature efficiently saves time in terms of searching beneficiaries's details and navigate through beneficiary records.

    - Moreover, it also allows the user to do consideration on which beneficiary to come for in terms of funding based on the activeness measurement of beneficiaries.

  - Highlights: This enhancement affects the existing commands such as Undo and Redo Command. It also requires the analysis of resources we have in term of time and performance of algorithm in the context of VolunCHeer. Moreover, it requires challengingly synchronization with related Projects and storage files. The summary table also requires adaptation of Ui components and learning of JavaFX.

  - Credits: The implementation of the order list is based on the code written for the person list in the Address Book Level 4 made by SE-EDU initiative.

- **Minor enhancement**: updated Storage Management #55

  - What it does: Storage is ensured to capture any change the user makes in the application reflects them in the storage file. When the user reopens the application, these changes are correctly viewed or if the data is corrupted, such as duplicated data or two beneficiaries have the same attached project, the application will prompt the user.

  - Justification: The data storage is important because the main purpose of the application is to reserve these data. Hence, storage management is essential for the application to work properly and serve its purposes.

- **Minor enhancement**: quality control of synchronization of assigning beneficiary to project #61 #125
- **Minor enhancement**: UI update up to version 1.3 #43 #67
- **Minor enhancement**: Bug reports and fixed #67 #72
- **Code contributed**: My code contribution can be viewed here
- **Other contributions**:
  - Managing issue tracker: #31 #61
  - Documentation:
    - Contribution to DG in early stage #2 #3
    - Did cosmetic tweaks to existing contents of the Developer Guide: #142
  - Giving substantial comments on Github #64 #124

# 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.*

# Beneficiary Management

### Adding a beneficiary: `addBeneficiary` / `ab`

Similar to the previous adding command, this adds a beneficiary to the list of Beneficiaries

Format: `addBeneficiary n/NAME a/ADDRESS e/EMAIL p/PHONE_NUMBER`

Example:

- `addBeneficiary n/Orphanage p/98765432 e/Orphanage@example.com a/311, Clementi Ave 2, #02-25`
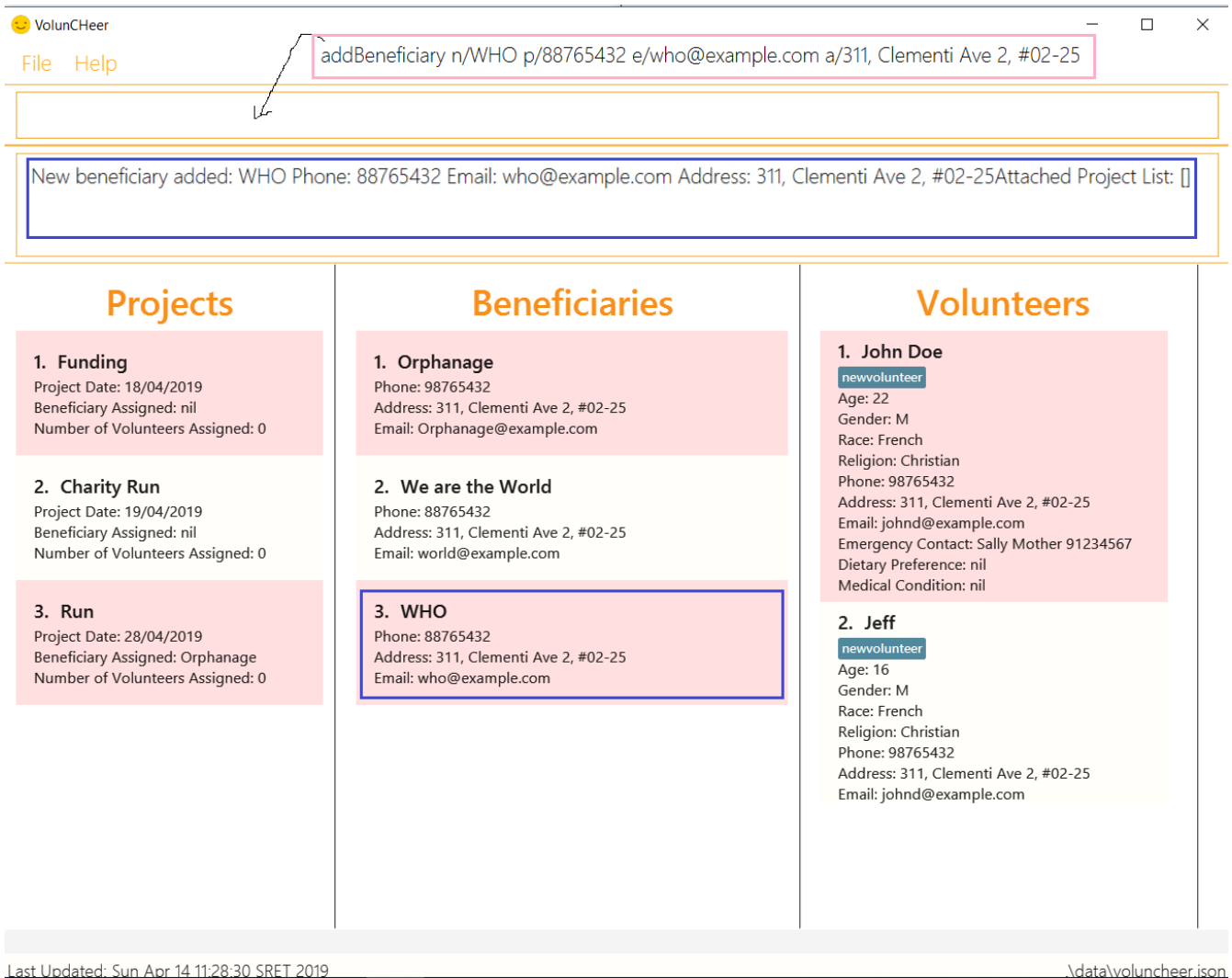
*Figure 1. Add Beneficiary Command Result (pink: input, blue: output)*

In the figure above, after the add command, we can observe a new beneficiary card is shown on the GUI.

- The beneficiary will be used to assign to a project, this means that the project will benefit this beneficiary, i.e. Orphanage Home, Nursing home, etc.

- When add a new beneficiary, the project lists assigned to it will be empty. You can assign projects to it by assign command stated.

## Editing a beneficiary: `editBeneficiary` / `eb`

In case of incorrect information, we also allow you to edit the beneficiary at the given INDEX

Format: `editBeneficiary INDEX (must be a positive integer) [n/NAME] [p/PHONE] [e/EMAIL] [a/ADDRESS]`

Examples:

- `editBeneficiary 1 n/Old Folk Home p/91234567`

*Figure 2. Edit Beneficiary Command Result (pink: input, blue: output)*

In the figure, we can see that the WHO information including phone number and email has changed, compared to the last figure.

| NOTE | When a beneficiary is edited, the data of the beneficiary in its attached projects is in sync, meaning that that data is automatically updated in the mentioned projects. |
|---|---|

## Deleting a beneficiary: `deleteBeneficiary` / `db`

Of course, once a beneficiary is no longer associated with you, it can be removed by providing the INDEX.

Format: `deleteBeneficiary i/INDEX -D`

| IMPORTANT | `-D` is optional and should not be misused (see below) |
|---|---|

- There are two modes of deletion: **soft delete mode** and **hard delete mode**.

- In the **soft delete mode**, there is a safe check to help you avoid deleting beneficiary that has attached projects, leaving the projects unassigned.

- In the **hard delete mode**, the beneficiary and all its attached projects will be deleted.

- Default is **soft delete mode**. To switch to **hard delete mode**, include `-D` in your command.

Examples:

- `deleteBeneficiary i/1` **soft delete mode**

- `deleteBeneficiary i/1 -D` **hard delete mode**



*Figure 3. Delete Beneficiary Command (Soft Delete Mode) Result (pink: input, blue: output)*

In Figure 3, we are trying to soft delete a beneficiary which was assigned to project **Run**. Hence, a message appears and informs us to switch to hard delete mode.

*Figure 4. Delete Beneficiary Command (Hard Delete Mode) Result (pink: input, blue: output)*

In Figure 4, the beneficiary and its attached projects have been deleted successfully.

## Listing all beneficiaries: `listBeneficiary` / `lb`

As before, you can show a list of all Beneficiaries in the beneficiary pool.

Format: `listBeneficiary`

| | |
|---|---|
| **TIP** | The command can be used to get back to full list after several commands which change the list. |

## Locating beneficiaries by name: `findBeneficiary` / `fb`

TO facilitate searching for beneficiary, you can locate a specific one easily with via given keyword/keywords.

Format: `findBeneficiary KEYWORD [MORE_KEYWORDS]`

- The search is case insensitive. e.g `orphanage` will match `Orphanage`
- The order of the keywords does not matter. e.g. `Orphanage Nursing` will match `Nursing Orphanage`
- Only the name is searched.
- Only full words will be matched e.g. `Orphan` will not match `Orphanage`
- beneficiaries matching at least one keyword will be returned (i.e. `OR` search). e.g. `Orphanage Nursing` will return `Orphanage Rainbow` and `Nursing Home`

Examples:

- `find Nursing`
  Returns `Nursing Home` and `Nursing Center`

**Summarising all beneficiaries:** `summariseBeneficiary` / `sb`

Sometimes we have a beneficiary assigned to many projects and we just want to see a list of everything it is attached to. This command opens a pop up summary table of the beneficiaries for easy view. You can use even the arrow in header cells **number of Projects** to sort beneficiaries by the number of attached projects.

Format: `summariseBeneficiary`

| Beneficiary Na... | No. Projects | List of attached projects |
|---|---|---|
| Orphanage | 2 | [Funding, Run] |
| We are the World | 1 | [Charity Run] |
| WHO | 0 | [] |

*Figure 5. Beneficiary Summary Table*

| TIP | The command can be used to consider future partners or fundraising. |
|---|---|

# 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.*

# Beneficiary Management Feature

## Implementation

Beneficiary is implement in order to manage the information a benefited volunteer organization. These organizations interact with the user's organization through projects. Hence, `Beneficiary` class has a bidirectional navigability with `Project` class, as shown in the Figure 10.
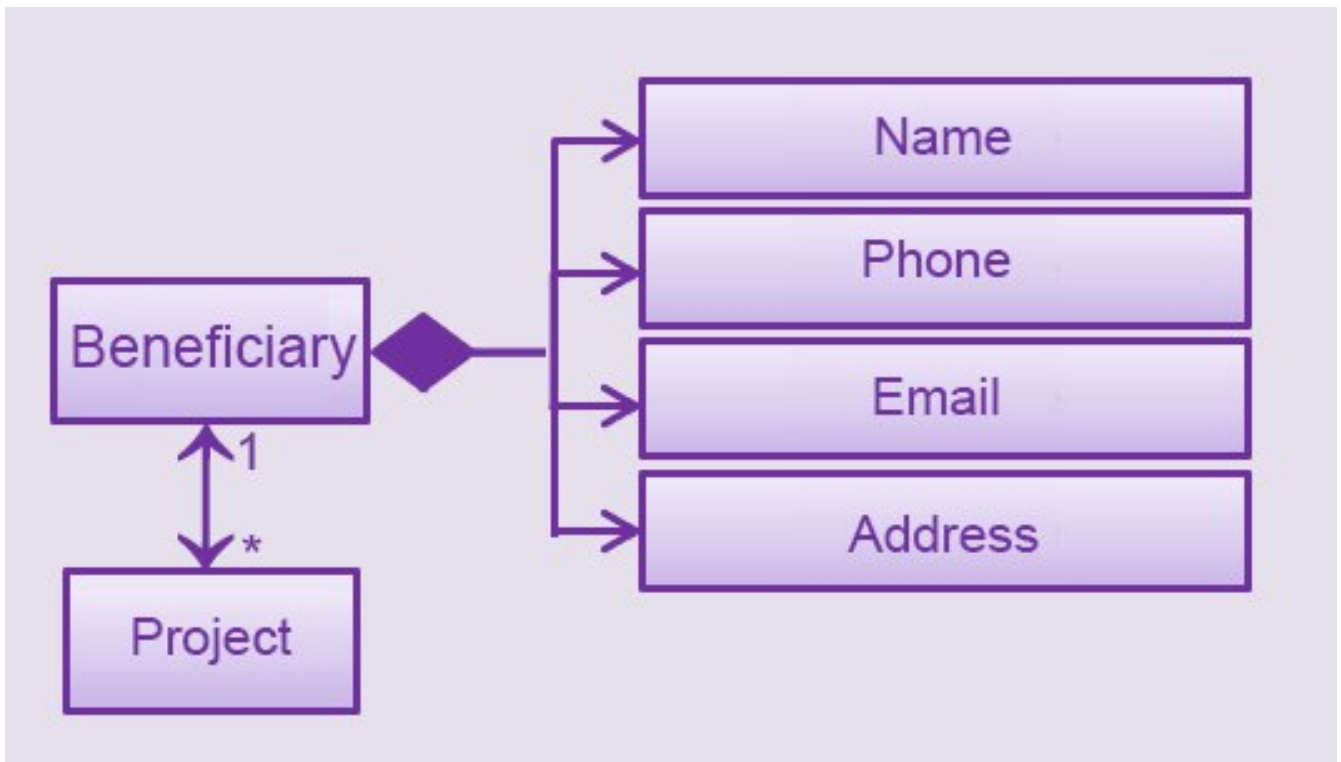
*Figure 6. Structure of the* `Beneficiary` *class including its attributes, and its bidirectional navigability with* `Project` *class.*

This means that if an operation such as deletion is done on a beneficiary, this should be updated on the projects that the beneficiary is assigned to. The figure below shows how the delete beneficiary command works:



*Figure 7. Beneficiary deletion sequence diagram, hard deletion mode.*

> **NOTE**    "-D" indicates that the deletion is in the hard mode, meaning that the respective projects that are attached to this beneficiary will be deleted.

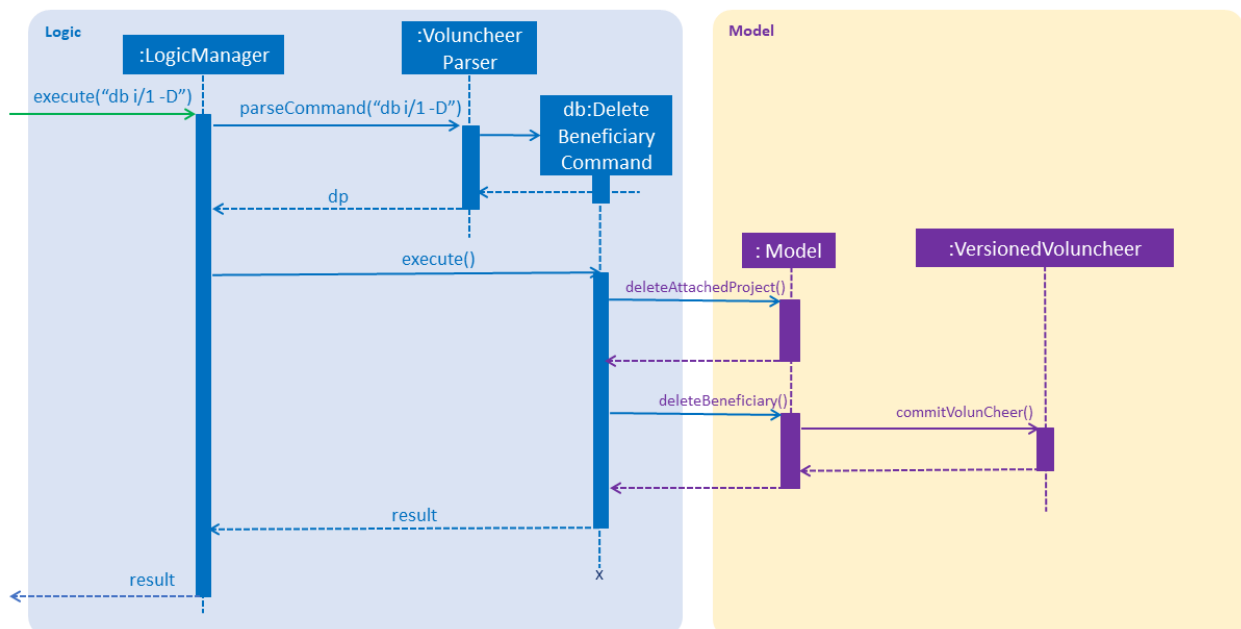1. The `DeleteBeneficiaryParser` parses the index of the beneficiary that is required to delete. The Parser constructs a `DeleteBeneficiaryCommand` with constructor as shown below:

```
public DeleteBeneficiaryCommand(Index targetIndex, boolean isHardDeleteMode) {
    this.targetIndex = targetIndex;
    this.isHardDeleteMode = isHardDeleteMode;
}
```

2. Method **deleteAttachedProjects(model, beneficiaryToDelete)** then calls the `ModelManager` to update the deletion of the respective projects.

```
    private void deleteAttachedProjects(Model model, Beneficiary beneficiaryToDelete)
{
        HashSet<ProjectTitle> attachedProjects =
beneficiaryToDelete.getHashAttachedProjectLists();
        List<Project> projectsToDelete = new
ArrayList<>(model.getFilteredProjectList());
        for (Project p : projectsToDelete) {
            if (attachedProjects.contains(p.getProjectTitle())) {
                model.deleteProject(p);
            }
        }
    }
```

3. The `ModelManager` is then called to update the deletion of the beneficiary and update all the changes.

```
model.deleteBeneficiary(beneficiaryToDelete);
model.commitAddressBook();
```

In order to view the synchronization, you can observe via project pool. This is to alleviate the worries of looking at attached projects when dealing with beneficiary, as a beneficiary can have multiple projects.
However, the Beneficiary Management Feature support the viewing of these information via Summarise Command. The Summarise Command generates the summarised statistics information of beneficiary based on their activeness.

| NOTE | The activeness of a beneficiary is measured by the number of projects that beneficiary has collaborated with the user's organization |

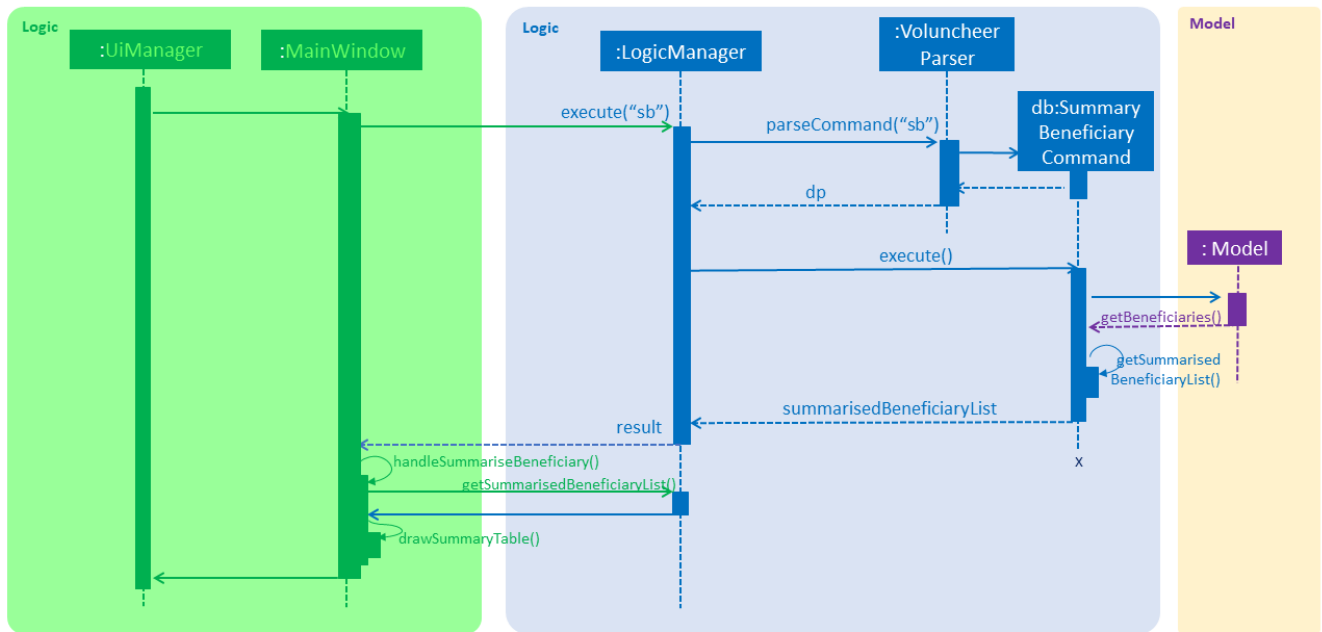The sequence diagram below shows how the Summarise Command works.

*Figure 8. Summarise beneficiary command sequence diagram.*

1. `SummaryBeneficiaryCommand` calls the `Model` to get the beneficiary list.

2. A summary list is generated and passed to `Logic`.

3. The Ui component which is `MainWindow` does handling of summarised list by generate a summary table and show on the screen.

## Design Considerations

| Aspect | Alternatives | Pros (+)/ Cons(-) |
|---|---|---|
| Implementation of Synchronization | **Update the by linear search for designed object** | + : It is easier to implement because the code base are list based. Moreover, the use of the application is limited to only local use without a large amount of data. Hence, this method gives a good performance in the context.<br><br>- : Unoptimized in terms of complexity, which requires more work for scaling of the application. |
| | Hash Table of the data | + : It has a better time complexity and reduce the work in scaling stage since this data structure is more optimized (O(1) can be achieved).<br><br>- : Take more resources to implement. |
| Display and use of attached project list | The beneficiary card shows the list | + : The synchronization can be observed throughout the execution of commands.<br><br>- : The beneficiary card is full with information and not reader friendly. Moreover, it is unnecessary to see the projects when operating single operations such as add, and edit |
| | **Generation of summary table** | + : The summary gives a good way to look at the statistics of the beneficiary list. As it allows the dynamic of sorting in ascending or descending order of the list based on the beneficiary's activeness<br><br>- : The adaptation of Ui is required. |