Zhao Jun Ru - Project Portfolio

PROJECT: VolunCHeer V1.3

Overview

VolunCheer is a desktop application for project managers who wish to keep track of their ongoing / upcoming projects as well as their beneficiary and volunteer pool.

Summary of contributions

- **Major enhancement**: implemented the Project class whereby it creates a new `Project` object to the VolunCHeer application*
  - What it does: the `Project` class is implemented with relevant functions that ties the Beneficiary pool and Volunteer pool together
  - Features involved: the relevant features include basic functions such as `addProject`, `deleteProject` and `listProject`. The added features are the `complete` feature and the `assign` feature which will be elaborated in further details.
  - Justification: The Project class forms the basis of the VolunCHeer application whereby the aim is to manage and organise projects better and save users form the hustle of paper works.
  - Highlights: The methods implemented involves interactions between the 3 objects namely `Project Beneficiary` and `Volunteer`. It provides user with the ability to add and delete `Projects`, assign the specific `Beneficiary` and a list of `Volunteers` to `Project` of interest as well as setting projects status as complete to keep track of ongoing and completed `Projects`.
  - Explanation of features implemented
    1. Assign Beneficiary feature **allows the user to assign a existing `Beneficiary` to the `Project` selected**
       - Justification: A `Beneficiary` is usually attached to a `Project` and the beneficiary details are very important and usually tracked in a separate document / file. Thus we decide to keep Beneficiary information as a separate list and assigned to the project when required such that it can be managed separately.
       - Highlights: This enhancement requires data from both the project list as well as the beneficiary list, there was thus some difficulty trying to implement the command. Many adjustments had to be made to keep the project class and beneficiary class independent and yet synchronised.
    2. Assign Volunteer feature **allows the user to assign a Required number of `Volunteers` to the `Project` selected**
       - Justification: Projects usually require varying number of volunteers, we thus decide to make it flexible by allowing user to key in the required number themselves. Furthermore, should there be any requirement for the `Volunteers` they can filter the `Volunteer` list with our `Map Command` and `Sort` function.

3. Complete project **allows the user to mark a project as completed in the project list** this is indicated by a colour change - to red for the project title.
    - Justification: This is thought to be a useful feature as project managers would want to keep track of what are the ongoing projects while still have a copy of their completed projects.
    - Highlights: This enhancement indicates a change of status of the specified project, it is a simple yet useful feature for the user.

- **Minor enhancement**: ListProject and DeleteProject commands are inherited from addressBook and successfully implemented on the project class.
- **Other contributions**:
    - Project management:
        - Manage project submissions and deadlines.
        - Managed setting up of Milestones v1.2 v1.3 and v1.4 #42
        - Managed releases `v1.3` and `v1.4` on GitHub #73 #122 #123
        - Setting up of issues to track progress #36 #74 #76 #110 #119 #120
        - Managed overall merging of project and solve bugs raised.
        - Merging and Integration of team’s repositories. #141
    - Documentation:
        - Did the UserGuide for v1.1: #6
        - Subsequent updates of documents on individual features.
    - Tools:
        - Integrated asciidoc for pdf releases of documentations as well as easy editing of adoc
        - Integrated Ruby for pdf document releases

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.

Project Management

Adding a project: `addProject` / `ap`

One of the first things to do when you use the app is to start adding projects to track, and this is the command to use.

Format: `addProject n/PROJECT_NAME d/DATE`

- Please enter DATE in DD/MM/YYYY format, making sure that the date should be after today.
- Project Name does not accept duplicates in the project list, so make you name everything differently!
- Projects are automatically sorted in ascending date order for easier tracking or Project tasks.

Now let us look at what happens when the command `addProject p/Old Folk Home Visit d/25/05/2019` is entered on screen.

[addProject] | *addProject.png*

Figure 1. When `addProject p/Old Folk Home Visit d/25/05/2019` *is executed.*

Deleting a project : `deleteProject` / `dp`

When a project is completed or cancelled, VolunCHeer allows you to easily delete it by stating the project order in the list.
Format: `deleteProject INDEX`

- This INDEX refers to the index of the project in the project list. If you are unsure of the order, **PLEASE** use 'listProject' to view all projects and get the correct index. If you delete the wrong projects, please refer to Undoing previous command : `undo`.
- Error message is shown if the INDEX entered is invalid

This is how the project list changes upon execution.

[deleteProject] | *deleteProject.png*

Figure 2. When `deleteProject 3` *command is executed.*

Listing all projects : `listProject` / `lp`

When you want to take a look at all your projects, this command helps you do so.

Format: `listProject`

Assigning a Beneficiary to Project: `assignB`

Projects are generally associated with certain beneficiaries. VolunCHeer allows you to attach them easily with this command. It assigns the Beneficiary at the provided INDEX to the Project with ProjectTitle indicated.

Format: `assignBeneficiary p/PROJECT_TITLE i/INDEX`

- The assigned Beneficiary can then be seen under the Project card as shown below.
- There can be only one beneficiary for each project, however, one beneficiary can be assigned to multiple projects.

IMPORTANT The index **must be a positive integer** `1, 2, 3, …`

After assigning a beneficiary, the project will have its adata updated as seen below.

[assignB] | *assignB.png*

Figure 3. When `assignB p/Old Folks Home Visit i/1` *command executed*

TIP Use listBeneficiary to view a full list of Beneficiary to assign. Use summaryBeneficiary command to view the Projects attached to each Beneficiary.

Assigning one or more Volunteers to Project: `assignV`

We also provide an easy method to assign a specific number of volunteers to the indicated Project

Format: `assignVolunteer p/PROJECT_TITLE rv/REQUIRED_NUMBER_OF_VOLUNTEERS`

TIP The number of volunteers assigned to the Project can be seen under the Project card as shown below.

[assignV] | *assignV.png*

Figure 4. When `assignV p/Old Folks Home Visit rv/2` *is executed.*

TIP Use the commands listed in Filtering & Exporting to filter out the desired list of volunteers.

Mark project as complete: 'complete'

Once a project is done, you can mark it as complete to distinguish it from your other projects. Simply provide an INDEX to indicate which project you would like to complete.

Format: `complete i/INDEX`

[complete] | *complete.png*

Figure 5. When `complete i/1` *command is executed*

NOTE Once marked as complete, project title will be displayed in red colour font

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`

[AddBeneficiary] | *AddBeneficiary.png*

Figure 6. 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`

[EditBeneficiary] | *EditBeneficiary.png*

Figure 7. 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**

[SoftDelete] | *SoftDelete.png*

Figure 8. 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.

[HardDelete] | *HardDelete.png*

Figure 9. 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`

[SummaryBeneficiary] | *SummaryBeneficiary.png*

Figure 10. Beneficiary Summary Table

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

Volunteer

Adding a volunteer: `addVolunteer` / `av`

As like before, this adds a volunteer to the volunteer pool

Format: `addVolunteer n/NAME y/AGE g/GENDER r/RACE [rg/RELIGION] a/ADDRESS e/EMAIL p/PHONE_NUMBER ec/EMERGENCY_CONTACT [dp/DIETARY_PREFERENCE] [m/MEDICAL_CONDITION] [t/TAG]…`

Alternative Format: `av n/NAME y/AGE g/GENDER r/RACE [rg/RELIGION] a/ADDRESS e/EMAIL p/PHONE_NUMBER ec/EMERGENCY_CONTACT [dp/DIETARY_PREFERENCE] [m/MEDICAL_CONDITION] [t/TAG]…`

- "Add Successful!" message is prompted upon successfully adding a volunteer
- An invalid message will be prompted if a Volunteer with the same exact name is present in the existing database
- Parameters for Religion, Dietary Preference, Medical Condition are optional and set to 'nil' by default

TIP A volunteer can have any number of tags (including 0)

Examples:

- `addVolunteer n/John Doe y/18 g/male r/eurasian rg/christian a/John street, block 123, #01-01 e/johnd@example.com p/98765432 ec/Mary, Mother, 92221111 dp/vegetarian m/asthma`
- `av n/Sarah Soh y/22 g/female r/chinese rg/buddhist a/betsy ave 6, 02-08 e/sarah08@example.com p/92345678 ec/Johnny, Husband, 81234568`

Deleting a volunteer : `deleteVolunteer` \ `dv`

After a volunteer has left, it can be deleted by this command by referencing its index in the list.

Format: `deleteVolunteer INDEX`
Alternative Format: `dv INDEX`

- Deletes the volunteer at the specified `INDEX`.
- The index refers to the index number shown in the displayed volunteer list.
- The index **must be a positive integer** 1, 2, 3, …
- Error message is shown if the given index is invalid

Examples:

- `listVolunteer`
  `deleteVolunteer 2`
  Deletes the 2nd volunteer in the volunteer list.
- `findVolunteer Betsy`
  `dv 1`
  Deletes the 1st volunteer in the searched volunteer list.

TIP Use the list volunteers commands to check the correct index of the volunteer to be deleted

Editing a volunteer : `editVolunteer` \ `ev`

Similar to beneficiary, we can update volunteer particulars by the given index.

Format: `editVolunteer INDEX [n/NAME] [y/AGE] [g/GENDER] [r/RACE] [rg/RELIGION][p/PHONE] [a/ADDRESS] [e/EMAIL] [ec/EMERGENCYCONTACT] [dp/DIETARYPREFERENCE] [mc/MEDICALCONDITION] [t/TAG]…`

Alternative Format: `ev INDEX [n/NAME] [y/AGE] [g/GENDER] [r/RACE] [rg/RELIGION][p/PHONE] [a/ADDRESS] [e/EMAIL] [ec/EMERGENCYCONTACT] [dp/DIETARYPREFERENCE] [mc/MEDICALCONDITION] [t/TAG]…`

- Edits the volunteer at the specified `INDEX`. The index refers to the index number shown in the displayed volunteer list. 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.
- When editing tags, the existing tags of the volunteer will be removed i.e adding of tags is not cumulative.
- You can remove all the volunteer's tags by typing `t/` without specifying any tags after it.

Examples:

- `editVolunteer 1 p/91234567 e/johndoe@example.com`
  Edits the phone number and email address of the 1st volunteer to be `91234567` and `johndoe@example.com` respectively.
- `ev 2 n/Betsy Crower t/`
  Edits the name of the 2nd volunteer to be `Betsy Crower` and clears all existing tags.

Locating volunteers by name: `findVolunteer` \ `fv`

Searching for volunteers works similarly to beneficiaries.

Format: `find KEYWORD [MORE_KEYWORDS]`

Alternative Format: `fv KEYWORD [MORE_KEYWORDS]`

- The search is case insensitive. e.g `hans` will match `Hans`
- The order of the keywords does not matter. e.g. `Hans Bo` will match `Bo Hans`
- Only the name is searched.
- Only full words will be matched e.g. `Han` will not match `Hans`
- volunteers matching at least one keyword will be returned (i.e. `OR` search).
- e.g. `Hans Bo` will return `Hans Gruber`, `Bo Yang`

Examples:

- `findVolunteer John`
  Returns `john` and `John Doe`
- `fv Betsy Tim John`
  Returns any volunteer having names `Betsy`, `Tim`, or `John`

Listing all volunteers : `listVolunteer` \ `lv`

Shows a list of all volunteers in the volunteer pool.

Format: `listVolunteer`

Alternative Format: `lv`

Filtering & Exporting

Assigning mapping index to each volunteer : `map`

We know that some volunteers suit a certain project better than others. To help with finding these volunteers, the map command assigns the volunteers with points 3, 2 or 1 according to the selection criteria that you set.

Format: `map t/(POINTS)(CRITERIA) t/(POINTS)(CRITERIA) t/(POINTS)(CRITERIA)`

- The t/ refers to any of the following tags.
- There are three types of tags, the age of volunteer (y/), race (r/) and medical condition (m/).
- You can enter at most 3 tags and at least 1 tag as the selection criteria.
- Each volunteer is internally assigned points which will be used used for sorting later on.
- The age criteria has comparators >,<,= which relate to the age given afterwards.
- See examples below for a clearer picture.

Examples:

- `map y/3>18 r/2chinese m/1NIL` Gives volunteers above the AGE of 18 3 points, RACE chinese 2 points and MEDICAL_CONDITION of NIL 1 point.
- `map m/3NIL` Only gives volunteers with no MEDICAL_CONDITION 3 points.

[MapCommand] | *MapCommand.png*

Figure 11. map command execution

Upon executing a successful map command, the message on figure 12 will appear.

Sorting volunteers according to points : `sort`

After mapping, we can then sort the volunteers into order, with the most suitable volunteers being on top.

Format: `sort`

- The map function should be called before sort to generate the points
- Volunteers with equal points will not be sorted in any particular order
- Selection of the volunteers, such as with the assignV command, can be done after sorting.

[SortBefore] | *SortBefore.jpg*

Figure 12. Before sorting

[SortAfter] | *SortAfter.jpg*

Figure 13. After sorting

As can be seen in figure 13 Alice was previously at index 3. After sorting, she has shifted up to index 2 in figure 14.

Extracting multiple volunteers from sorted list : `extract`

Not everyone will have VolunCHeer, which is frankly their loss. Nonetheless, this command allows you share a list of certain volunteer particulars by extracting it into a Microsoft Excel file.

Format: `extract NUMBER_OF_VOLUNTEERS t/PARTICULAR [t/OTHER_PARTICULARS]…`

- This command requires at least one type of particular from the volunteers, up to all type of particulars.
- If the NUMBER_OF_VOLUNTEERS exceeds the total number of volunteers in the list, the file will just extract all volunteers in VolunCHeer.
- This command can be called before map and sort if order is not an issue.

Examples:

*`extract [1][20]` Extracts the first 20 volunteers in the sorted list. *`extract [5][15]` Extracts volunteer number 5 to 15 in the list.

[Export] | *Export.png*

Figure 14. Extracted volunteer details

The Excel file will look like figure 15.

Listing entered commands : `history`

Lists all the commands that you have entered in reverse chronological order.
Format: `history`

NOTE Pressing the kbd:[&uarr;] and kbd:[&darr;] arrows will display the previous and next input respectively in the command box.

Undoing previous command : `undo`

Restores the VolunCHeer application to the state before the previous *undoable* command was executed.
Format: `undo`

NOTE Undoable commands: those commands that modify the VolunCHeer application's main content (`addProject`, `addVolunteer`, `delete`, `edit` and `clear`).

Examples:

- `delete 1`
  `list`
  `undo` (reverses the `delete 1` command)
- `select 1`
  `list`
  `undo`
  The `undo` command fails as there are no undoable commands executed previously.
- `delete 1`
  `clear`
  `undo` (reverses the `clear` command)
  `undo` (reverses the `delete 1` command)

Redoing the previously undone command : `redo`

Reverses the most recent `undo` command.
Format: `redo`

Examples:

- `delete 1`
  `undo` (reverses the `delete 1` command)
  `redo` (reapplies the `delete 1` command)
- `delete 1`
  `redo`
  The `redo` command fails as there are no `undo` commands executed previously.
- `delete 1`
  `clear`
  `undo` (reverses the `clear` command)
  `undo` (reverses the `delete 1` command)
  `redo` (reapplies the `delete 1` command)
  `redo` (reapplies the `clear` command)

Clearing all entries : `clear`

Clears all entries from the specific list requested by user.
Format: `clear`

Exiting the program : `exit`

Exits the program.
Format: `exit`

Saving the data

All data for the application are saved in the hard disk automatically after any command that changes the data.
There is no need to save manually.

Attendance taking `[coming in v2.0]`

Track attendance of the volunteers and award frequent volunteers with certificates or promote to team leader.

Manage funding and sponsorships `[coming in v2.0]`

Manage funds and sponsors for individual projects and track project spending.

Auto-completion of command `[coming in v2.0]`

Quick Auto-completion of command to enhance typing speed

FAQ

Q: How do I transfer my data to another Computer?
A: Install the app in the other computer and overwrite the empty data file it creates with the file that contains the data of your previous VolunCHeer application folder.

Command Summary

- **AddProject** `addProject n/PROJECT_TITLE d/DATE b/BENEFICIARY [t/TAG]…`
  e.g. `addProject n/Charity Run d/081219 b/Sunshine Old Folks Home`
- **AddVolunteer** `addVolunteer n/NAME y/AGE a/ADDRESS e/EMAIL p/PHONE_NUMBER g/EMERGENCY_CONTACT r/RACE d/DIETARY_PREFERENCE m/MEDICAL CONDITION [t/TAG]…`
  e.g. `addVolunteer n/John Doe y/18 a/John street, block 123, #01-01 e/johnd@example.com p/98765432 g/98292998 r/chinese d/vegetarian m/asthma`
- **AddBeneficiary** `addBeneficiary n/NAME a/ADDRESS e/EMAIL p/PHONE_NUMBER`
  e.g. `addBeneficiary n/Orphanage p/98765432 e/Orphanage@example.com a/311, Clementi Ave 2, #02-25`
- **EditBeneficiary** `editBeneficiary INDEX (must be a positive integer) [n/NAME] [p/PHONE] [e/EMAIL] [a/ADDRESS]`
  e.g. `editBeneficiary 1 n/Old Folk Home p/91234567`
- **DeleteBeneficiary** `deleteBeneficiary i/INDEX -D` e.g. `deleteBeneficiary i/1 -D`
- **ListBeneficiary** `listBeneficiary`
- **FindBeneficiary** `findBeneficiary KEYWORD` e.g. `findBeneficiary Old`
- **SummariseBeneficiary** `summariseBeneficiary`

- **List** : `list`
- **EditProject** `editProject PROJECT_NAME [n/NAME] [d/DATE] [b/BENEFICIARY] [t/TAG]…`
  e.g. `editProject Charity Run d/010319`
- **EditVolunteer** `edit INDEX [n/NAME] [p/PHONE] [e/EMAIL] [a/ADDRESS] [t/TAG]…`
  e.g. `editVolunteer 1 p/91234567 e/johndoe@example.com`
- **Find** : `find KEYWORD [MORE_KEYWORDS]`
  e.g. `find James Jake`
- **DeleteProject** : `delete PROJECT_TITLE` e.g. `delete Charity Run`
- **DeleteVolunteer** : `delete INDEX`
  e.g. `delete 3`
- **Select** : `select INDEX`
  e.g.`select 2`
- **Map** `map t/SELECTION t/SELECTION t/SELECTION`
  e.g. `map y/18 > r/chinese m/NIL`
- **Sort** `sort`
- **Extract** `extract VOLUNTEERS_REQUIRED+` e.g. `extract 20`
- **History** : `history`
- **Undo** : `undo`
- **Redo** : `redo`
- **Clear** : `clear`
- **Export** : `export`
- **Import** : `import`
- **Exit** * : `exit`
- **Help** : `help`

== 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.

=== Project Complete Feature The complete feature allows users to indicate a project as completed.

==== Implementation To facilitate the complete feature, an association with a new `Complete` class is added to the `Project` class:

[ProjectClassDiagram] | *ProjectClassDiagram.PNG*

Figure 15. Structure of the attributes of a Project in the Model component.

The diagram shows that the Project class is associated with the Complete class.

The following sequence diagram shows how the complete command works:

[CompleteSequenceDiagram] | *CompleteSequenceDiagram.png*

Figure 16. Figure Sequence diagram for the complete command.

1. The `CompleteCommandParser` parses the user input to obtain the target project index and constructs a ne `CompleteCommand` with this index.
2. The logic portion of the complete command will be executed by the `CompleteCommand` method. To mark a Project object as complete:
    1. The **CompleteCommand()** method creates a `targetProject` based on the provided project index.
    2. In the **executeCommandResult()** method then creates a `editedProject` with `Complete` attribute set to “true”. The `editedProject` is created with ProjectBuilder as shown below:

```
    Project editedProject = new
ProjectBuilder(targetProject).withComplete(true).build();
```

3. In the executeCommandResult() method

```
    model.setProject(targetProject, editedProject)
```

is called to replace `Project`’s complete attribute from “false” to “true” in the VolunCHeer in-memory.

==== Design Considerations

Aspect	Alternatives	Pros (+)/ Cons(-)
Implementation of ‘CompleteCommand’	**Add a Complete attribute to Project (current choice) -Completed projects indicated “Red”**	+ : It is easy to tag complete status as an attribute to the `Project` as we can make use of current implementations such as model.setProject(Project,Project) that sets the `Project`’s complete attribute to “true”. - : Unable to have a observable list of complete projects.
	Create a new CompletedProjectList that consists of all the complete projects, a listComplete command to show all completed tasks..	+ : Will use less memory (e.g. for deleteVolunteer, just save the volunteer being deleted). - : We must ensure that the implementation of each individual command are correct.

=== Assign Feature Assigning a Beneficiary / VolunteerList to Project.

==== Implementations Since the implementation of commands AssignBeneficiary and AssignVolunteer are similar, we will describe the implementation of AssignBeneficiary command only and provide the difference between the two.

[assignSequence] | *assignSequence.png*

Figure 17. Sequence diagram to show how the AssignBeneficiaryCommand works.

1. The **AssignBeneficiaryCommand(ProjectTitle, Index)** takes in the targetProject’s projectTitle attribute and targetBeneficiary’s index.
2. The **executeCommandResult()** method
   1. Sets up projectToAssign by calling a predicate to compare with the `ProjectTitle` in `FilteredProjectList`:

      ```
      model. getFilteredProjectList(). filtered(equalProjectTitle).get(0);
      ```

   2. **updateBeneficiary(model)** methods updates the `Beneficiary` object so that ProjectTitle is tracked within the Beneficiary class.
   3. editedProject is created using ProjectBuilder to take in the `Beneficiary` assigned. The following method is called to store the `Project` in VolunCHeer with specific `Beneficiary` attached to it.

      ```
      model.setProject(projectToAssign, editedProject)
      ```