

Jerry Kwong - Project Portfolio

Project: MyProject

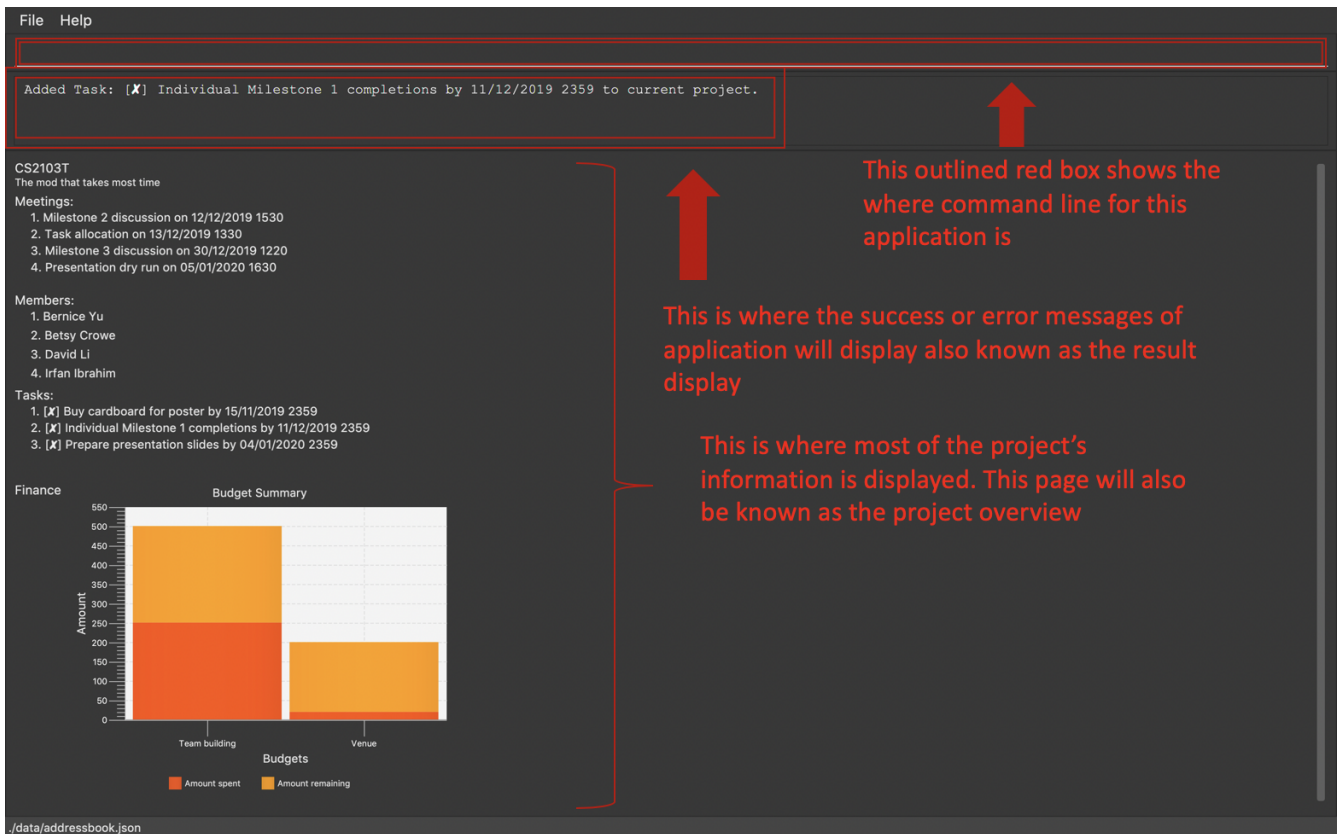
Project background

MyProject is an command line desktop application that I helped design and implement as part of a team, for a school project for the NUS School of Computing. My team consisted of 5 computing students including me, and the task for this project was to modify a given application called the AddressBook into our own application, that serves a different purpose. As second year university students, we have always found it difficult to manage many different types of information across many different projects during our first year, and hence we decided to modify the AddressBook application into *MyProject*. Since we fit into the target demographic as well, it was easier for us to relate to the problem we were trying to solve, which helped us when we were deciding what features to implement for this project.

Overview

MyProject is a desktop application with a command line interface, which aims to be a one-stop project management application that will satisfy all the project management needs of a student. Our application specialises in storing, processing and displaying the information of the projects in a meaningful way to help students better manage their workload when it comes to dealing with multiple different projects at the same time. *MyProject* provides basic functionality such as storing the members, meetings and tasks of a given project, and also storing the contact information of the members. *MyProject* also boasts more complex functions such as calculating optimal meeting times based on the NUSMODS timetable of each member, and also tracking the finances of the project with different types of budgets.

A typical usage of the project looks like this:



My team worked together tirelessly to put this application together. Each of us are in charge of different parts of the application's functionality, and my role in the team was to integrate the Person model in the original AddressBook application into our application, and use it as a way to keep track of the member's information.

In addition to that, I also implemented a way for the application to track each member's performance in terms of a few tangible attributes such by keep track of how many tasks they have done. This is to allow anyone using the application to optimise their workflow, and better manage the progress of their project since they would know who has done what tasks and how each member is contributing to the team. This would also aid in more even work distribution.

In the next section, I will summarise all my contributions to the project.

Summary of contributions

- Major enhancements: Added the performance overview functionality of *MyProject*.
 - What it does: This feature consolidates all the information available, and uses it to calculate the performance of each member. The information is stored, and displayed to the user when the the user executes the `showProjectOverview` command. The calculated attributes of the performance is then displayed in the form of tables for easy comparison between the members. It will also show the individual performances of each member by itself. To see a clearer view of this, you may look at my contributions to the user guide which will be in the subsequent sections.
 - Justifications: This feature allows the user to conveniently keep track of what each member has done and their contributions to the project. This can help with distributing the workload between the members, and also helps if there is a need to assign credit to each member for

their amount of work done.

- Highlights: This feature does not require any additional storage as it is only calculated when the user requests for it, and all the information is already available within the `Person` and `Project` objects themselves. The feature also provides both an overview of their performances for easy comparison and a more individual view where the attributes are grouped and listed by according to each member should there be a need to check the specific member's performance. The user also only has to type in the `showPerformanceOverview` command and everything else will be handled internally this is because all the information needed is already supplied during the day to day interactions with the application when the user `assignTask` to members and `markAttendance` for meetings. (Both of which are commands that can be executed by the user).
- Challenges: One of the challenges I faced when implementing this feature is finding a way to implement the `PerformanceOverview` model. I initially considered making it an attribute of the existing `Project` model which my teammate implemented, however I quickly realised that this would not be feasible, as calculating the performance of a member requires a information from both the `Project` and `Person`. Therefore I chose to implement `PerformanceOverview` as a new model instead which created less dependencies between the `Project` and `Person` models, and also reduced the need for the `Project` model to be associated with the `Person` model.
- Minor enhancements: I also added the functionality to keep track of the members information within the application, such as what projects they are involved in, and the commands to add and remove members from projects. To facilitate my performance overview functionality, I also implemented 3 more commands which are the `assignTask`, `unassignTask` and `markAttendance` command. This is to supply the application with useful information to be used in the performance calculations when the performance overview is requested by the user.
- Code contributed: [[Functional code](#)] ← Click this link to see my code contributions.
- Other contributions:
 - Project Management:
 - Helped to report and sometimes fix the bugs which I found during the routine testing of our application.
 - Documentation:
 - Did the first round of general formatting of our user guide which included fixing the inconsistent formatting, and improving the overall organisation and style of the whole user guide. See more here → [[Pull request #69](#)]

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.

Add a new member: `addMember` [Checkout]

This command is similar to adding a person to your contacts shown previously, except that you

have to be checked out into a project, and the person will be added to the working project as well.
Format: `addMember n/NAME [p/PHONE_NUMBER] [e/EMAIL] [a/address] [t/tag]...`

TIP Adding a member only requires his/her name!

However it will be good to add as much information as possible.

Example:

- `addMember n/John Doe a/John street, block 123, #01-01`
- `addMember n/Betsy Crowe e/betsycrowe@example.com t/friend`

To help you better understand how to use this command, here is a step-by-step guide, using the second example.

Step 1: You type in the `addMember` command, followed by all the information you want to store, which in this case is her email and also her tag as a friend. With this, before you press enter your screen should look like this:

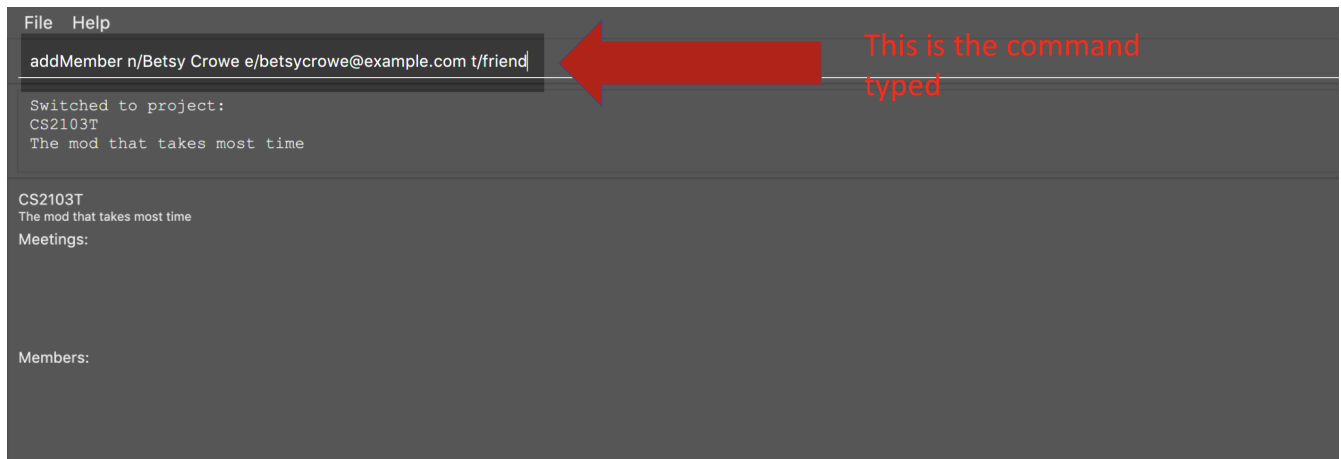


Figure 1. `addMember` Command user input

If your screen looks like the picture above, just press enter and you're done! Adding a new member is just a simple one step process.

After you press enter you will be able to see the member reflected in the project like this:

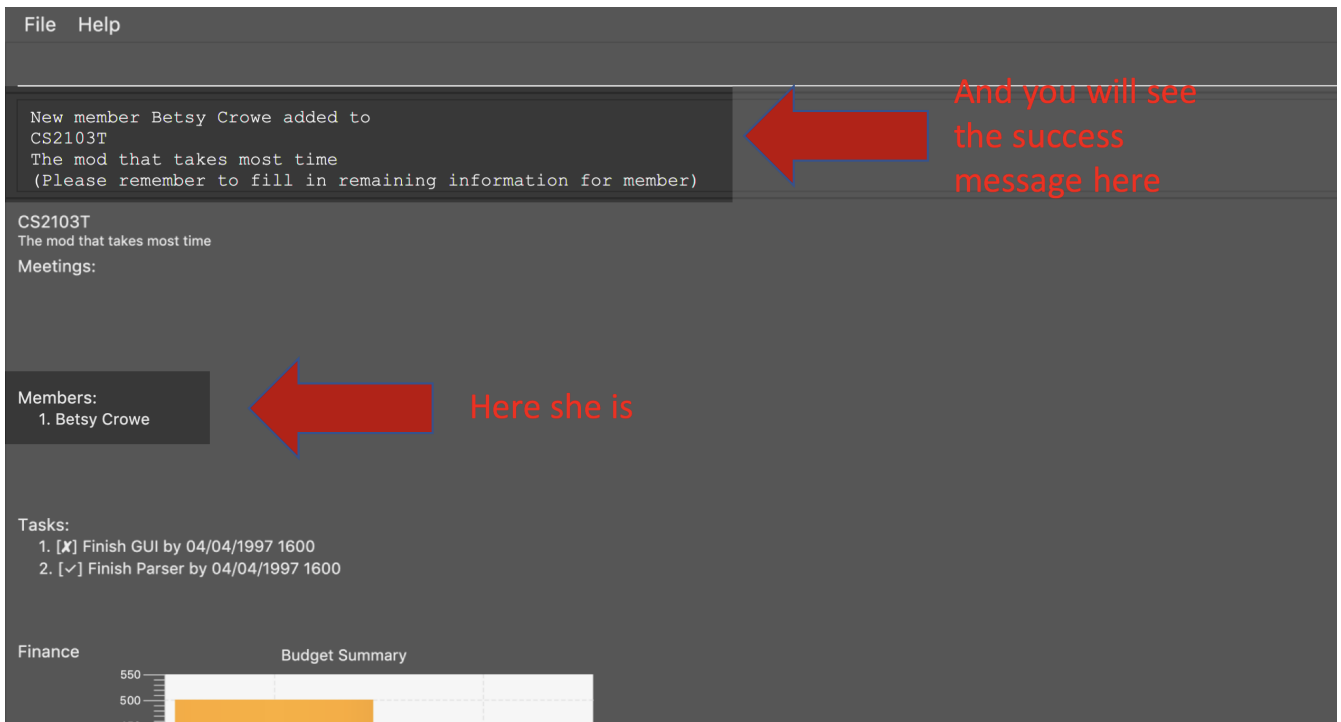


Figure 2. AddMemberCommand success project overview display

And you will also be able to see her in your contacts like this:

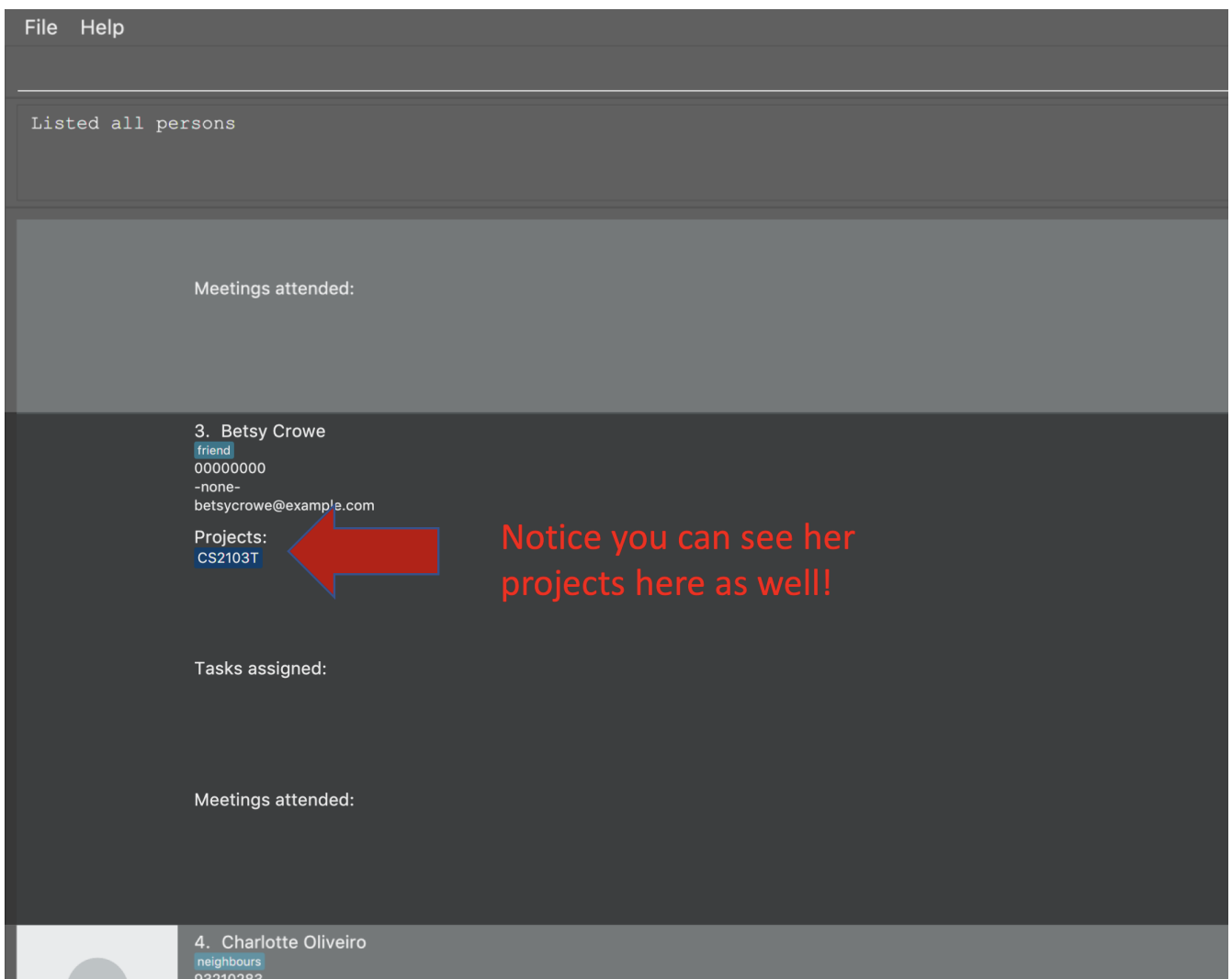


Figure 3. New Member displayed in your contacts as a new person added.

If you look closely at Figure 5, you will see that some of the information does not really seem right. That is because you have not input it yet, but don't worry you can always do that later with our `edit` command which you will further down the guide.

Add from your contacts: `addFromContacts` [Checkout]

This command helps to add a person you already saved in your contacts into your project, reducing the need to type his information all over again. All you need to do is enter the index he is listed at.

Format: `addFromContacts INDEX...`

INDEX is the number which the person is listed at, and it should be a positive integer eg. 1, 2, 3,

TIP

You can put multiple indexes to add multiple people to your project at once isn't that convenient! E.g. `addFromContacts 1 3 5`

Example:

- `addFromContacts 1`

To help you better understand how to use this command, here is a step-by-step guide.

Step 1. Find the person you want to add, and take note of the index which the person is listed at. Referring to the picture below, let's say you want to add 'Bernice Yu' into your project. Notice her index is 2.

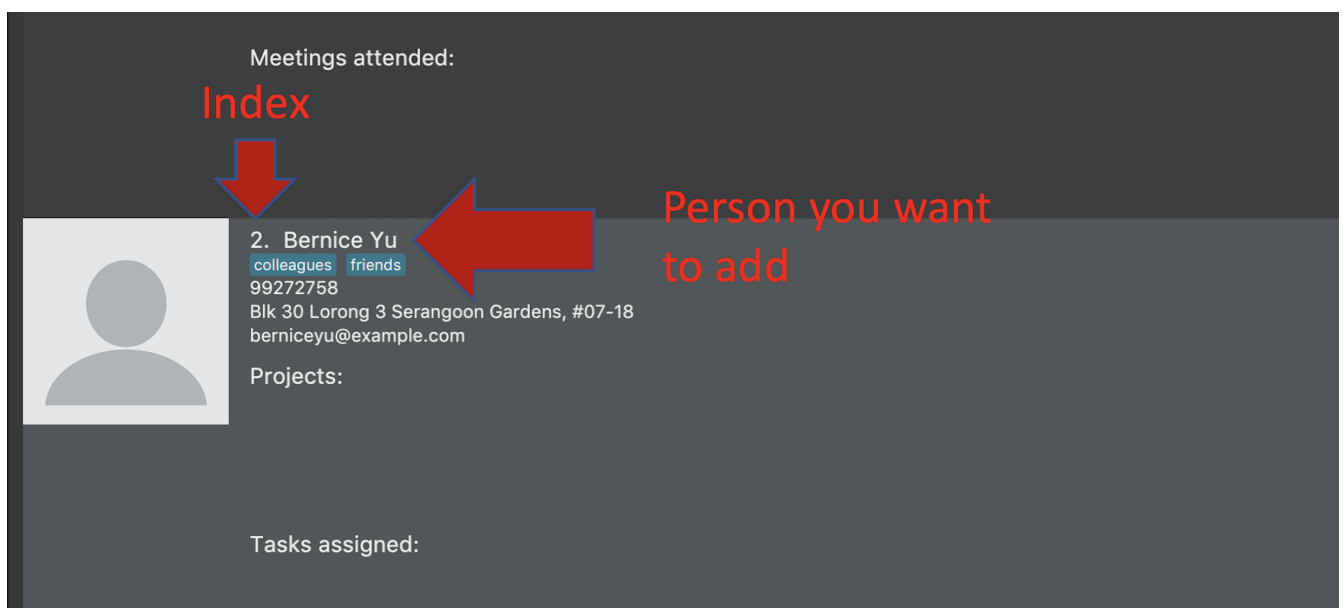


Figure 4. Finding the person to add

Step 2. Type in the `addFomContacts` command with the index 2

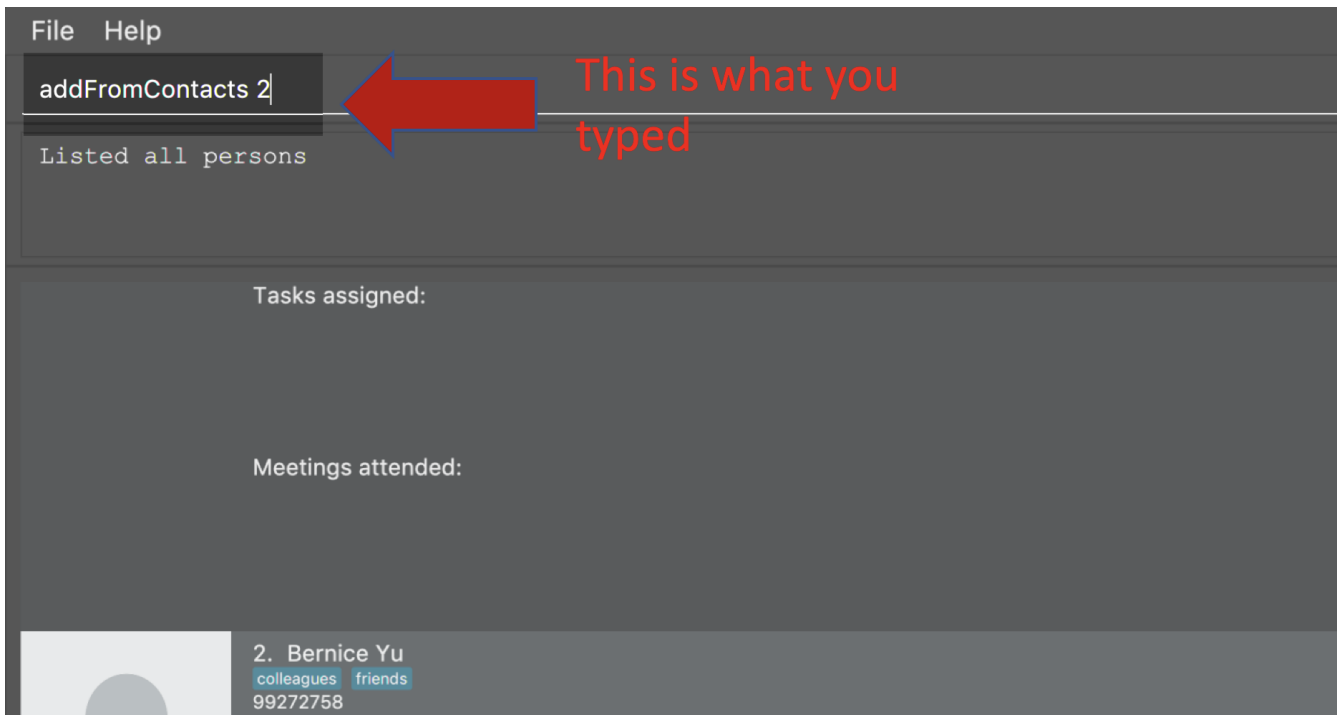
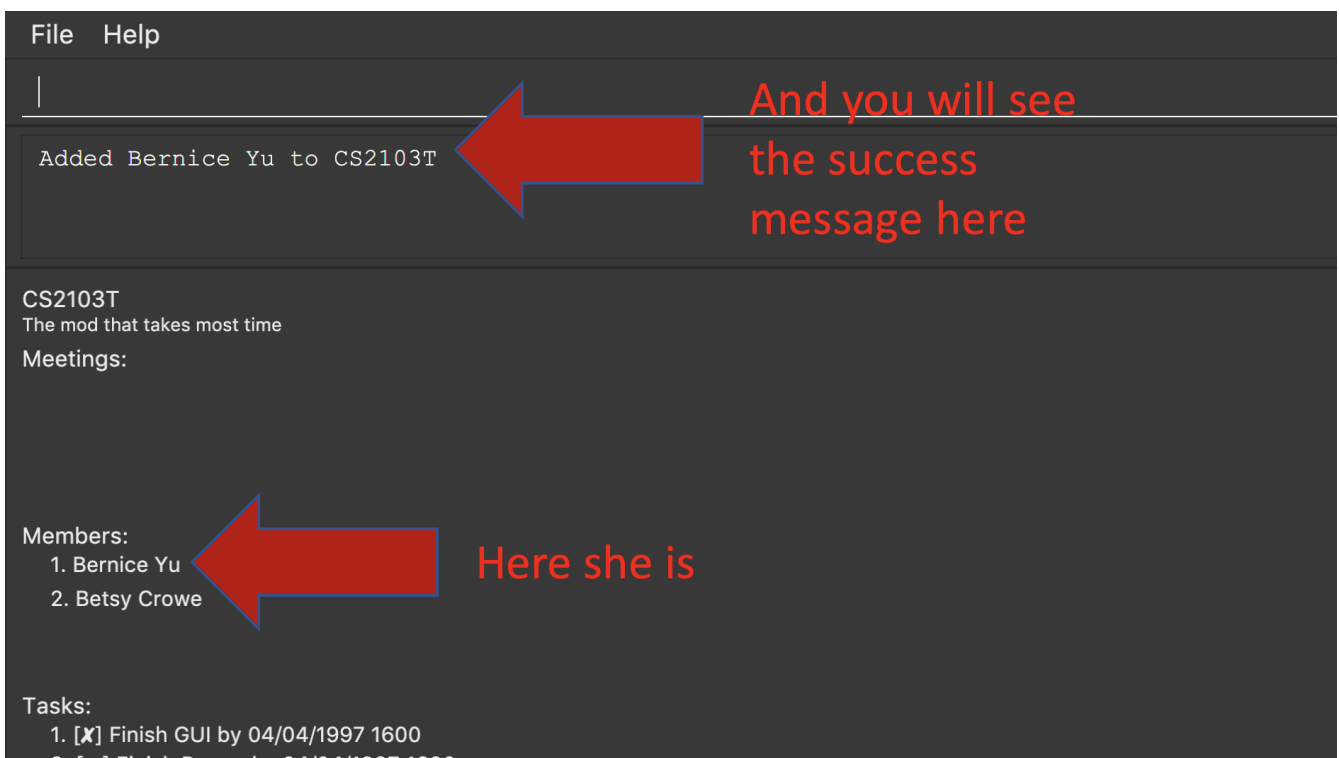


Figure 5. `addFromContacts` command input.

Step 3. Press enter and you're done! Refer to the picture below to see what your screen should look like.



Remove a member: `removeMember` [Checkout]

Removes the specified person from the current working project.

Format: `removeMember INDEX`

`INDEX` refers to the index that the person is listed at under the members section of the project overview.

Example:

- `removeMember 1`

To help you better understand how to use this command, here is a step-by-step guide.

Step 1. Find the person you want to remove from the project. Let's say you want to remove 'Bernice Yu' because she just dropped your module. Notice her index is at 1



Figure 6. Finding the person you want to remove

Step 2. Type in the `removeMember` command with the index of 1

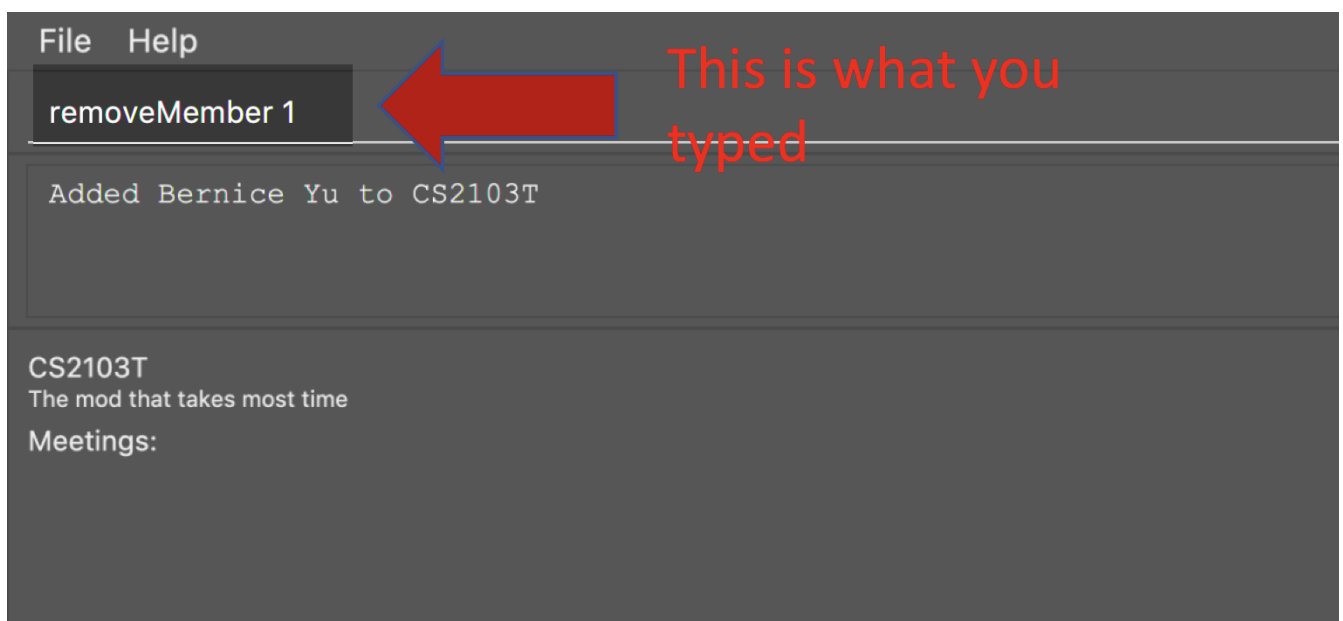


Figure 7. `removeMember` command user input.

Step 3. Press enter and you're done! She is now no longer a member of your project.

Performance tracking

Knowing that it is hard to keep track of what every needs to do, or has done so far, MyProject gives you a simple way of tracking the performance of each member as well. The following section will give you all the details you need to know about this feature.

Assigning a task: `assignTask` [Checkout]

This command allows you to assign a task to one or more of your members in the project.

Format: `assignTask TASK_INDEX PERSON_INDEX...`

`TASK_INDEX` refers to the index at which the task is displayed at.

`PERSON_INDEX` refers to the index at which the member is displayed at. (You can input more than 1 index)

Example:

- `assignTask 1 1`

TIP

You can assign a task to multiple people easily just by including all of their indexes E.g. `assignTask 1 1 3 5` this assigns task 1 to member 1,3 and 5.

To help you better understand how to use this command, here is a step-by-step guide.

Step 1. Find the task you want to assign and the members you want to assign the task to, and take note of their indexes.

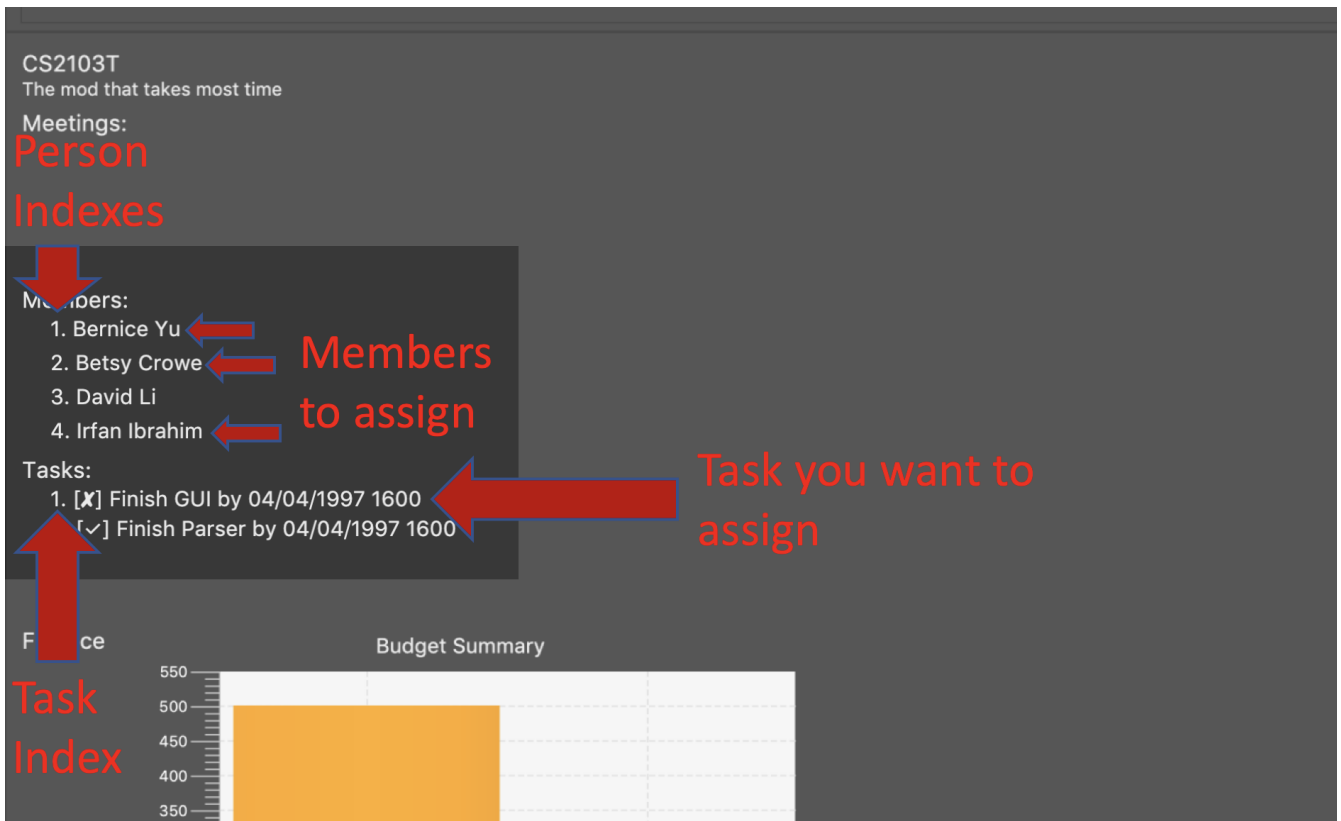


Figure 8. Finding the task to assign and the members to assign to

Step 2. Type in the `assignTask` command along with the correct indexes.



Figure 9. Typing the `assignTask` command with the correct input

Step 3. Press enter and you're done! You will now see the tasks reflected under the respective members in your contacts.

You will see a success message:

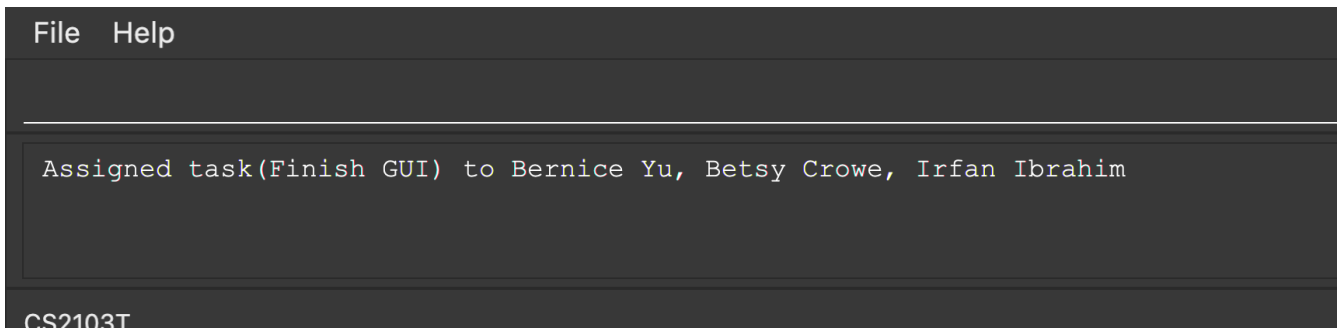


Figure 10. `assignTask` command success message

And also see the task reflected under the members:

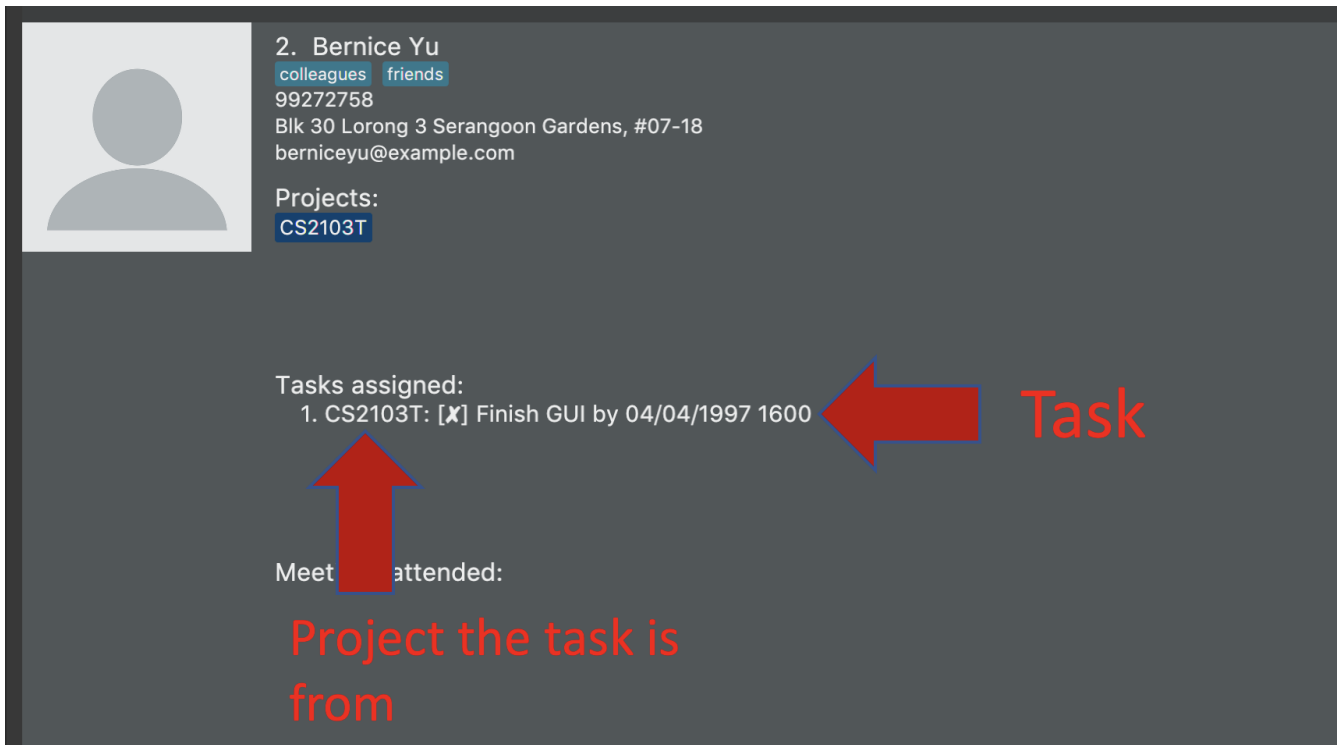


Figure 11. How the assigned task is shown

Unassigning a task: `unassignTask`

With the ability to assign task, naturally you should be able to unassign a task as well, and you can do that with this command.

Format: `unassignTask PERSON_INDEX TASK_INDEX...`

`PERSON_INDEX` refers to the index of the person as displayed in your contacts

`TASK_INDEX` refers to the indexes of the tasks as displayed under the specific person

Example:

- `unassignTask 1 1`

TIP

You can unassign one or more tasks at once from a person by specifying all the task indexes. E.g. `unassignTask 1 1 3 5` this unassigns tasks 1, 3, and 5 from the first person.

To help you better understand how to use this command, here is a step-by-step guide.

Step 1. Identify the tasks you want to unassign and the person you want to unassign the tasks from, and take note of their indexes.

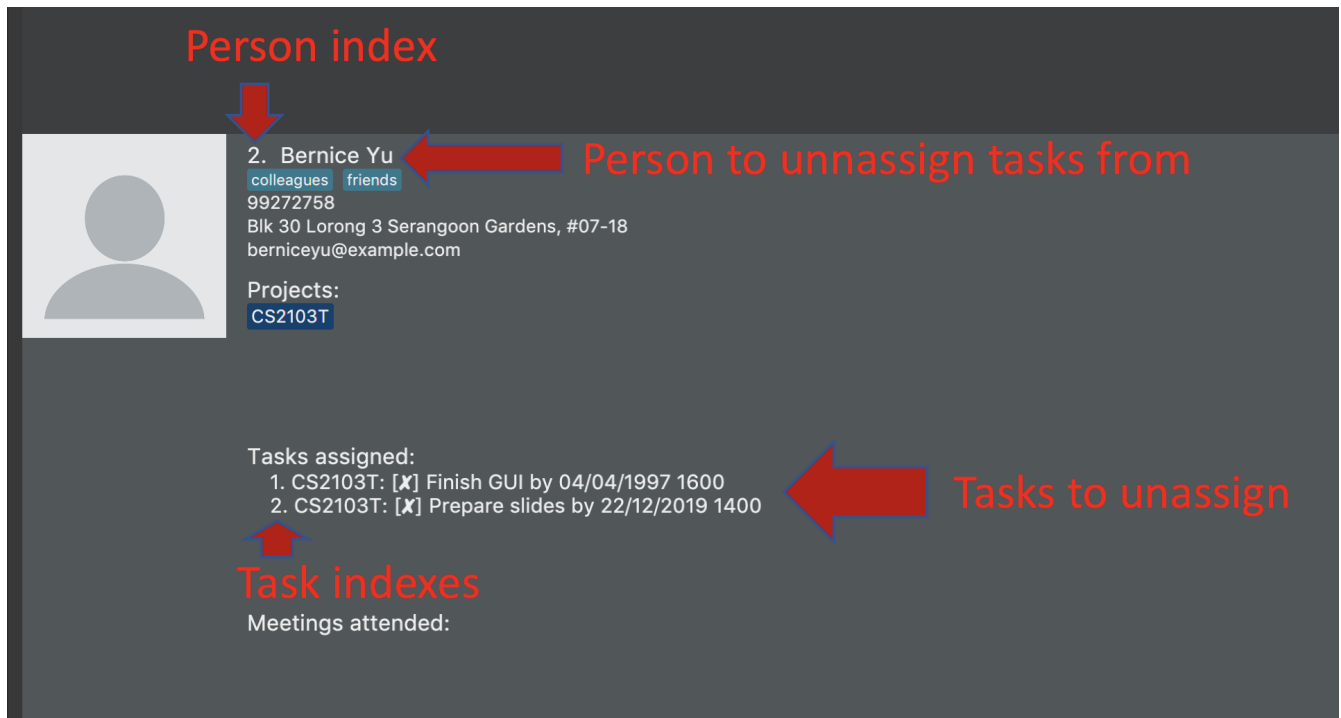


Figure 12. Finding the tasks to unassign, and the person to unassign the tasks from.

Step 2. Type in the `unassignTask` command with the relevant inputs

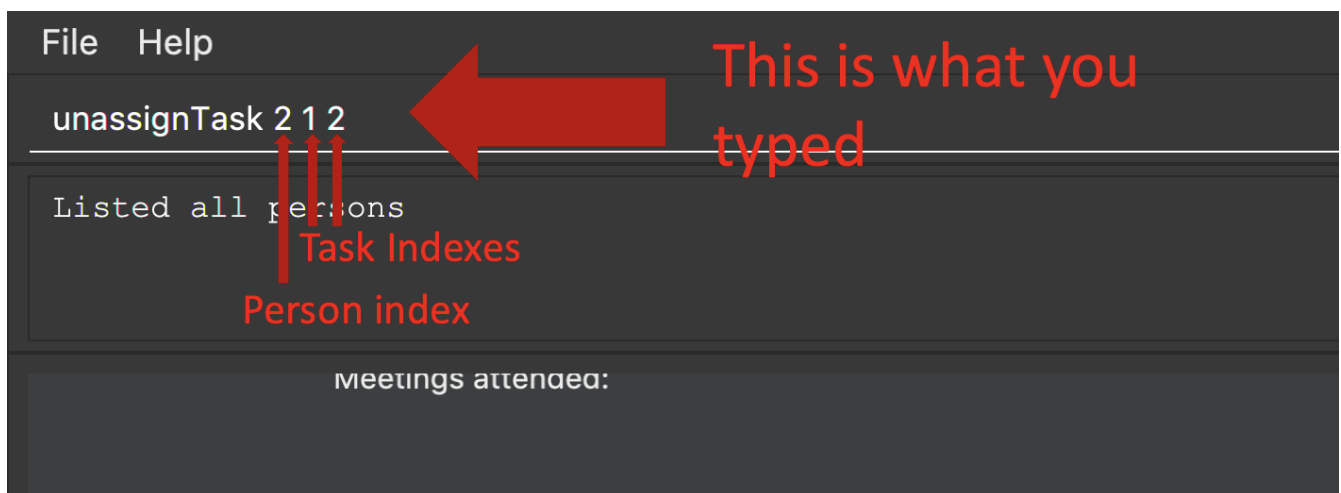


Figure 13. `unassignTask` command input

Step 3. Press enter and you're done!

You will see a success message:

```
Unassigned task(s): Finish GUI, Prepare slides  
from Bernice Yu.
```

Figure 14. `unassignTask` command success message

And you will no longer see the tasks reflected under the person.

Mark attendance: `markAttendance` [Checkout]

MyProject also allows you to mark the attendance of one or more members for a meeting.

Format: `markAttendance MEETING_INDEX PERSON_INDEX...`

`MEETING_INDEX` refers to the index of the meeting as displayed in the project overview

`PERSON_INDEX` refers to the index of the person as displayed in the project overview

Example:

- `markAttendance 1 1`

TIP

You can mark the attendance of multiple people at once, just by specifying all the indexes of the persons E.g. `markAttendance 1 1 3 4` this marks the attendance for meeting 1 for members 1, 3 and 4.

To help you better understand how to use this command, here is a step-by-step guide.

Step 1. Identify the meeting you want to mark attendance for, and the members who were present.

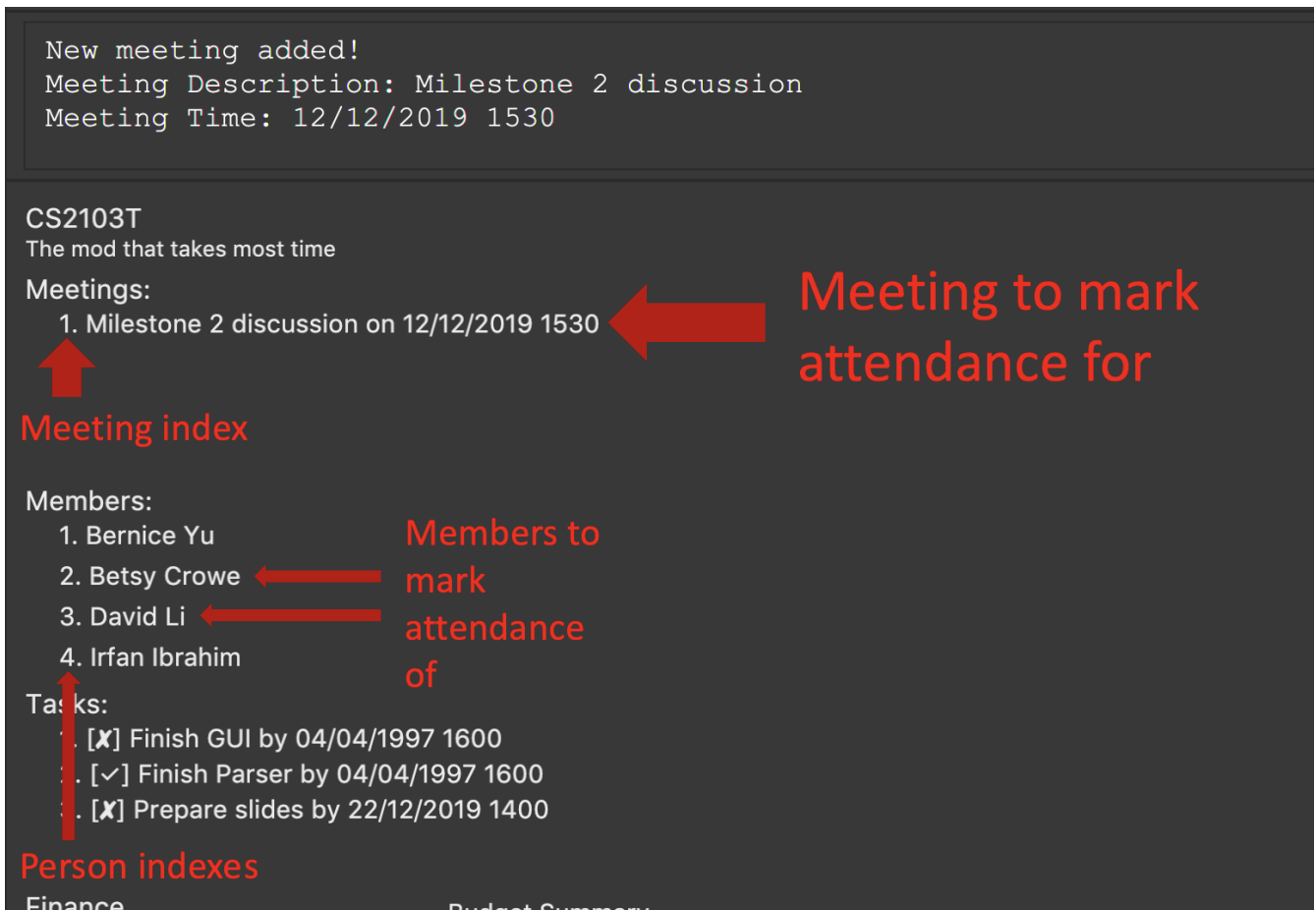


Figure 15. Finding the meeting to mark attendance for and the members to mark attendance of

Step 2. Type in the `markAttendance` command with the relevant inputs.

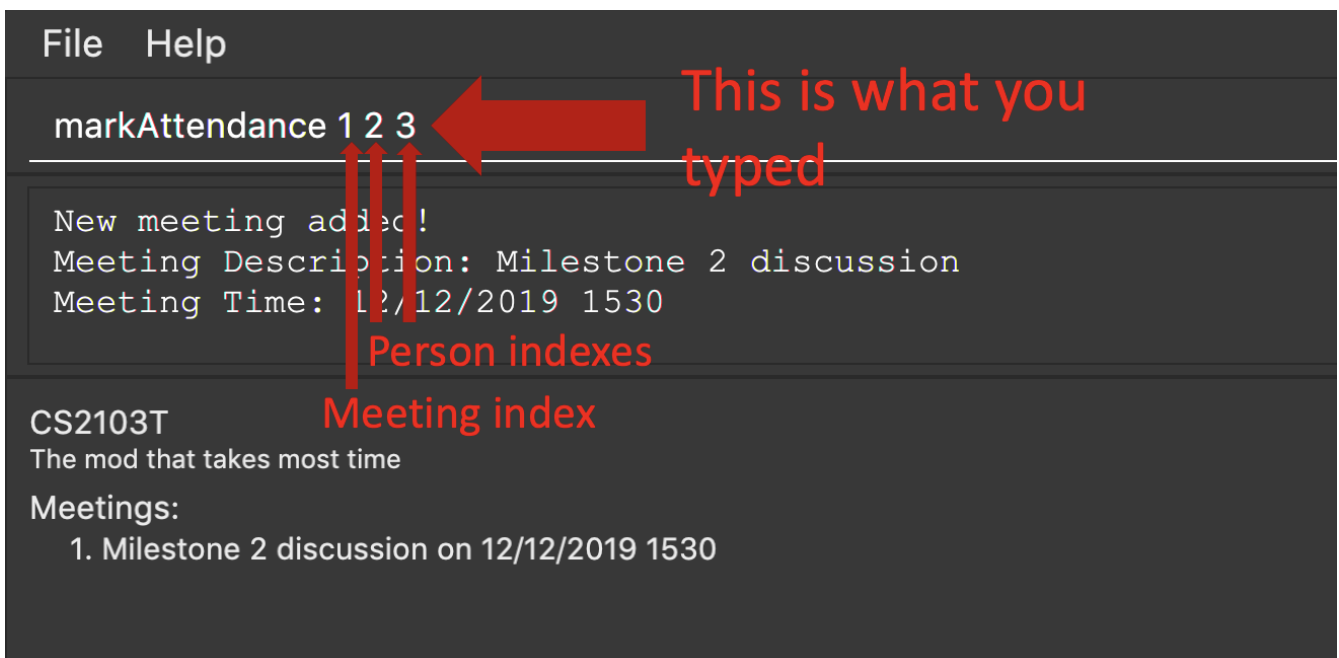


Figure 16. `markAttendance` command input

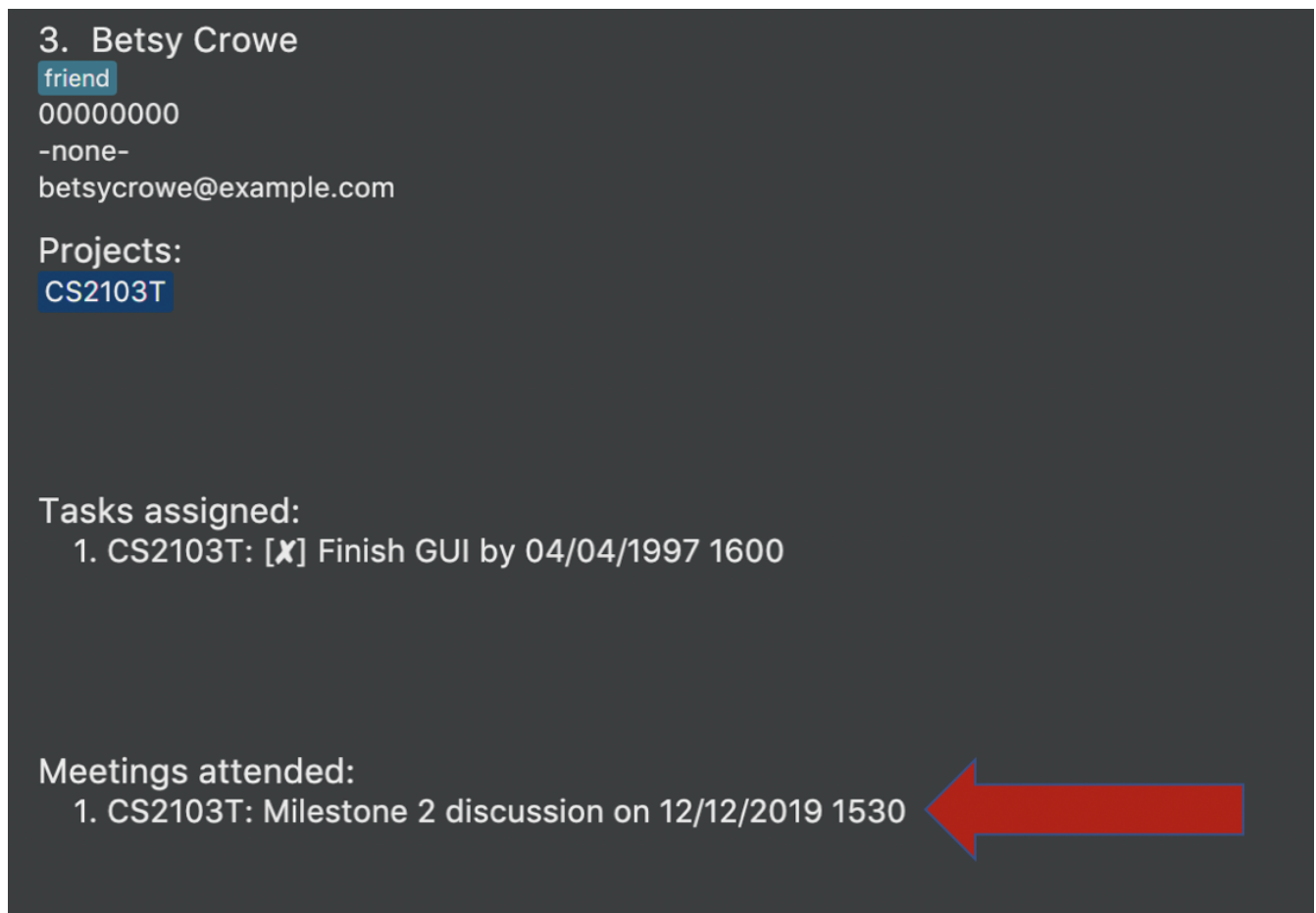
Step 3. Press enter and you're done!

You will see a success message:

```
Attendance for meeting(Milestone 2 discussion on 12/12/2019 1530) is marked for Betsy Crowe, David Li
```

Figure 17. `markAttendance` command success message

And the meeting will show up under the information of the person like this:



The screenshot shows a user profile for Betsy Crowe. It includes a 'friend' status, a unique ID '00000000', a placeholder '-none-' for a profile picture, and an email address 'betsycrowe@example.com'. Under the 'Projects:' section, 'CS2103T' is listed. The 'Tasks assigned:' section shows one task: '1. CS2103T: [X] Finish GUI by 04/04/1997 1600'. The 'Meetings attended:' section shows one meeting: '1. CS2103T: Milestone 2 discussion on 12/12/2019 1530'. A large red arrow points to this meeting entry.

```
3. Betsy Crowe
friend
00000000
-none-
betsycrowe@example.com

Projects:
CS2103T

Tasks assigned:
1. CS2103T: [X] Finish GUI by 04/04/1997 1600

Meetings attended:
1. CS2103T: Milestone 2 discussion on 12/12/2019 1530
```

Figure 18. Showing attendance marked

Viewing performance overview: `showPerformanceOverview`

After assigning the tasks and marking the attendance of members, you can use this command to give you a comprehensive view of how each member is performing within the project isn't that convenient!

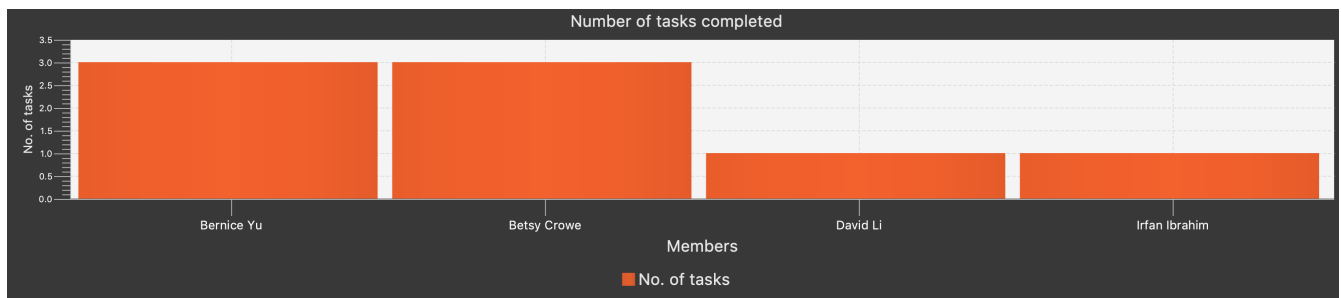
Format: `showPerformanceOverview`

There are 4 attributes that we calculate for every member which constitutes their performance:

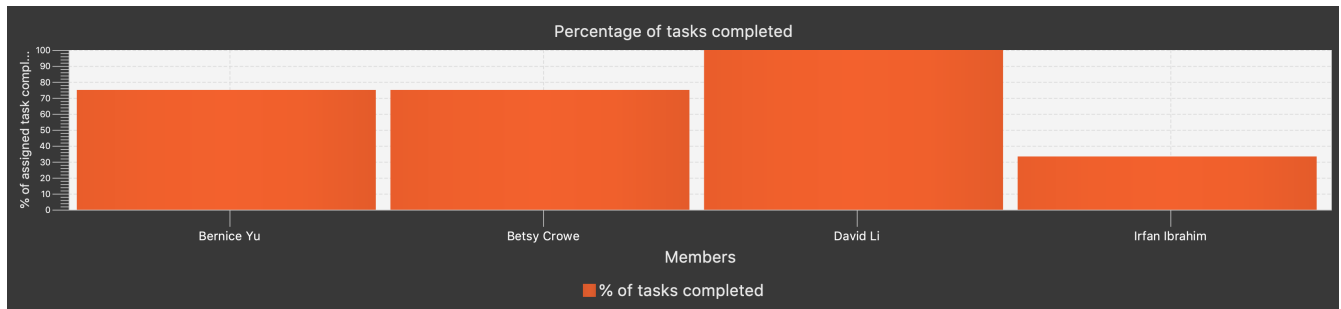
1. Number of tasks completed
2. Percentage of assigned tasks completed (Rate of task completion)
3. Number of meetings attended
4. Percentage of total number of meetings attended (Rate of attendance)

Refer to the pictures below to understand more about how we display this information in a organised manner to you.

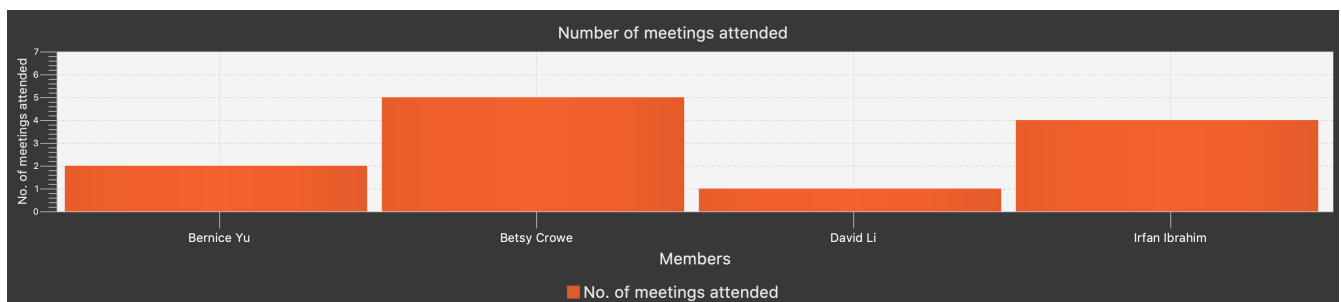
- Table showing the number of task each member completed:



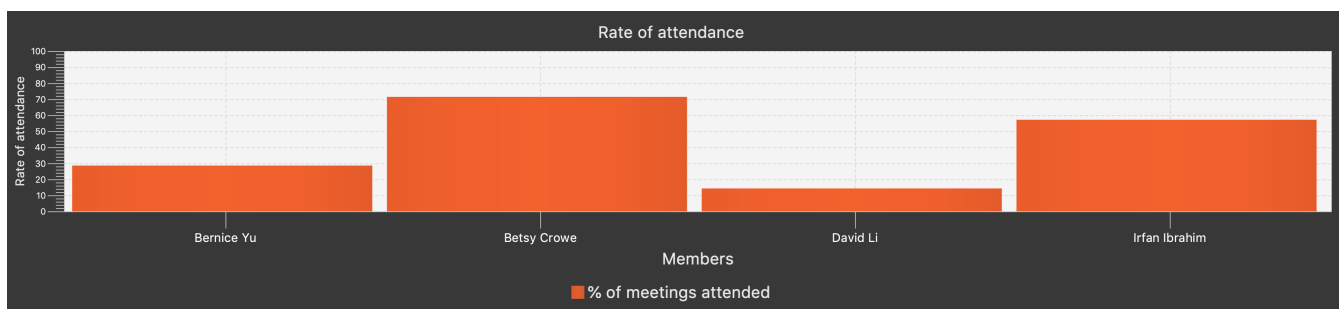
- Table showing the percentage of their assigned tasks each member completed:



- Table showing the number of meetings each member attended:



- Table showing the percentage of the total number of meetings each member attended:



- Lastly you can also view each member individually after scrolling through the tables:

1. Bernice Yu

Number of tasks done: 3

% of tasks completed: 75.0%

Number of meetings attend: 2

% of meetings attended: 28.6%

2. Betsy Crowe

Number of tasks done: 3

% of tasks completed: 75.0%

Number of meetings attend: 5

% of meetings attended: 71.4%

3. David Li

Number of tasks done: 1

% of tasks completed: 100.0%

Number of meetings attend: 1

% of meetings attended: 14.3%

4. Irfan Ibrahim

Number of tasks done: 1

% of tasks completed: 33.3%

Number of meetings attend: 4

% of meetings attended: 57.1%

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.

hello world

Member feature

Description of feature

In every project, it almost certain that there are members. Therefore we have allowed for the users to be able to track who are the members, and how many members there are within each project. Each **Project** stores its members as a list of strings representing their names, and similarly each **Person** stores the projects they are involved in as a list of strings representing the project titles. Below you will find 2 diagrams which represents this relationship.

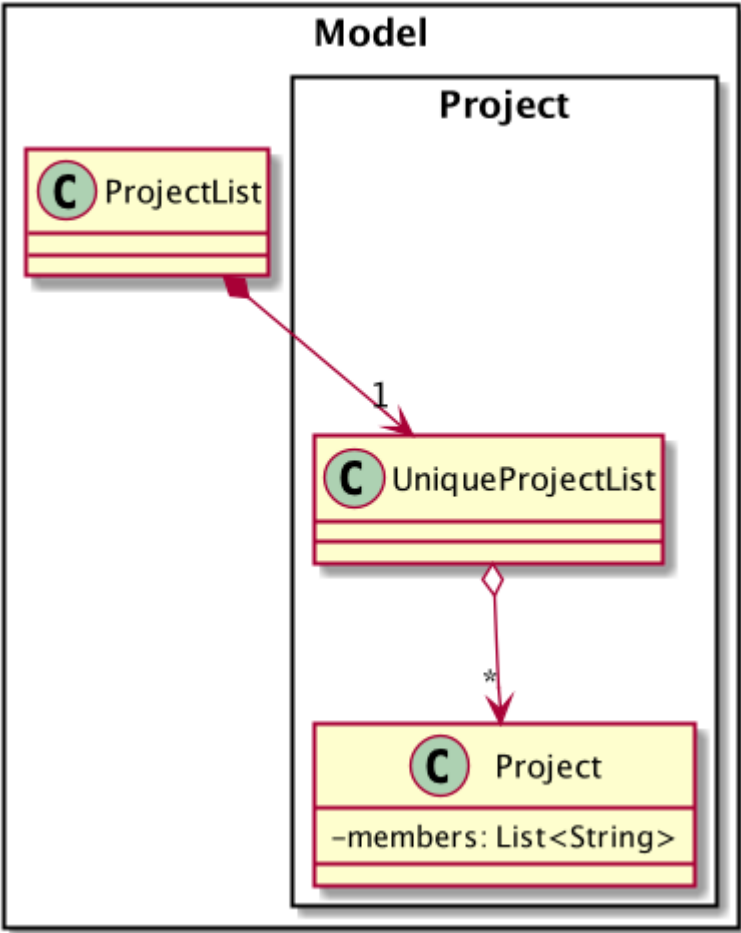


Figure 19. Class diagram of Project

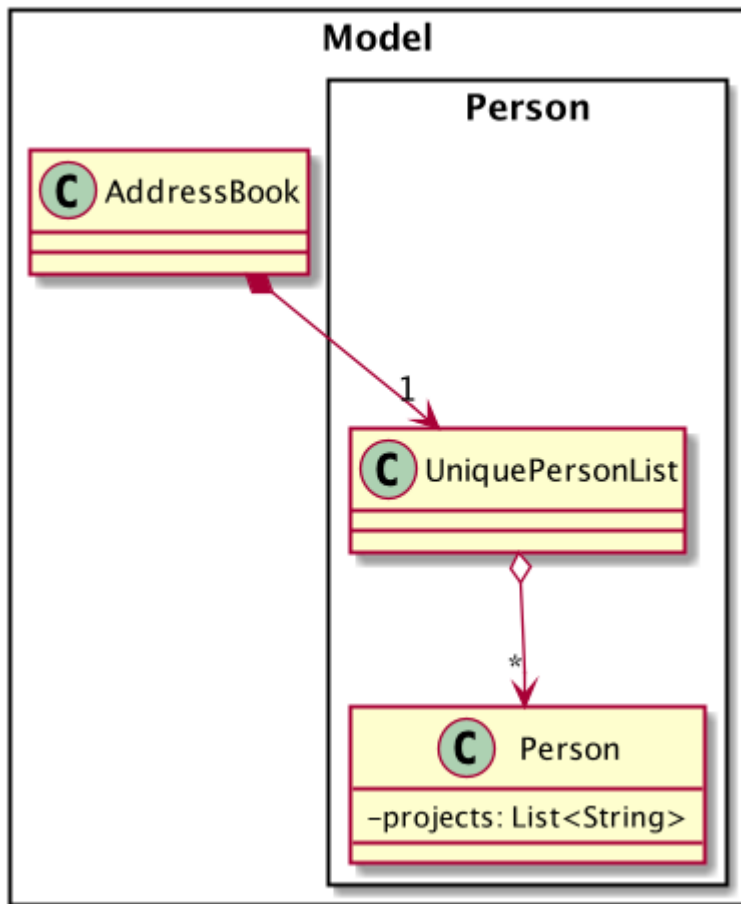


Figure 20. Class diagram of a Person

NOTE

Whenever member is mentioned it should be known that it is represented by a **Person**.

In order to facilitate this feature, 4 commands are implemented as a subclass of the **Command** class:

1. **addMember** - Adds a new **Person** to the contacts and records the name as a member of the current working project.
2. **addFromContacts** - Adds a **Person** to the current working project as a member.
3. **removeMember** - Removes the record of the **Person** as a member of the current working project.
4. **addProfilePicture** - Adds a profile picture to the specified **Person**.

Implementation

In this section, we will demonstrate how a member is added to the project from your contacts, and also how a member as a new **Person**.

We will start with adding a new **Person** to your contacts, and to your project at the same time.

Step 1. The user enters the **addMember** command with the following parameters **n/David p/94328727 e/david97@hotmail.com a/Ang Mo kio avenue 3**.

Full user input: **addMember n/David p/94328727 e/david97@hotmail.com a/Ang Mo kio avenue 3**

The user input is parsed into an `ArgumentMultiMap` by `AddMemberCommandParser#parse` so that every attribute of the person can be extracted, and put into a `NewMemberDescriptor` to be used to correctly create the `Person`.

NOTE `ArgumentMultiMap` is a class that stores all the parsed parameters from the user input.

NOTE `NewMemberDescriptor` is a class that stores any all the information on the `Person` given by the user to be used to create the `Person` subsequently.

Step 2. The information of the person is stored in the `NewMemberDescriptor`, and used to create a new instance of the `AddMemberCommand`

Every Project keeps track of which members are involved by storing a list of strings of the names of the members. Similarly a Person keeps track of which projects they are involved in by storing a list of strings of the project titles.

The following commands are implemented to support this feature:

1. `addMember` - Adds a new person to the contacts as well as to the current working project.
2. `addFromContacts` - Adds a person currently stored in your contacts to the current working project.
3. `removeMember` - Removes a person from the current working project.
4. `addProfilePicture` - Adds a profile picture to a person in the contacts.

Performance Tracking

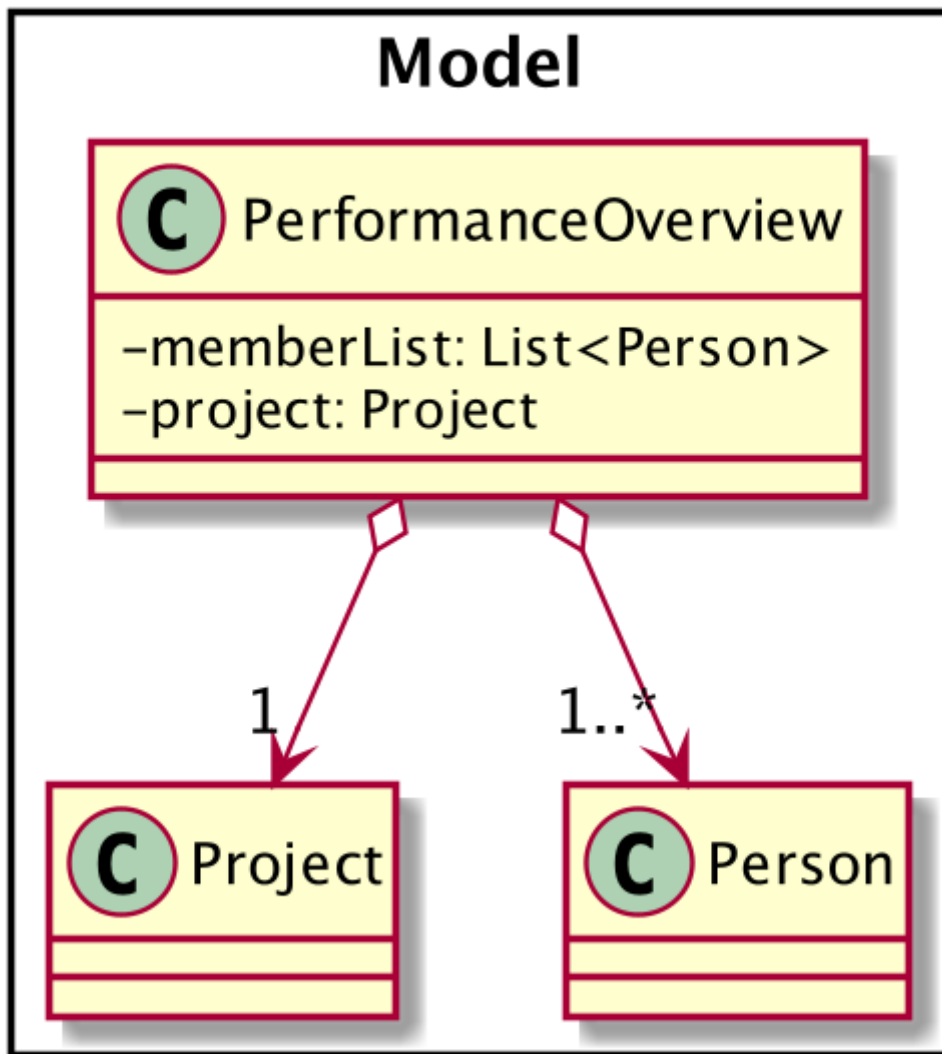
We allow users to be able to track the performance of every member in the project, by using information that the user has already input into MyProject.

Implementation

The `Performance` of each `Person` is consolidated and stored in a `PerformanceOverview` object inside the model.

NOTE `Performance` of a `Person` stores the information such as tasks assigned to the person, which are specific to the person. The actual performance of the person can only be calculated in `PerformanceOverview` with information from the `Project` as well.

`PerformanceOverview` is a separate model we have implemented, which consolidates all the data from each `Person` involved in the specific project, as well as from the project itself. The following is a class diagram for `PerformanceOverview`.



A typical **PerformanceOverview** is created using the following constructor:

```
PerformanceOverview(project, memberList)
```

- **project** - This is the **Project** that the user is concerned about.
- **memberList** - This is the `List<Person>` which consists of all the **Person**(s) involved with this project. Note that they each have their own **Performance** as well.

All the calculations of the various components constituting the performance of an individual is calculated within the **PerformanceOverview**, and the results are accessed using each of the following commands:

1. **PerformanceOverview#getAttendanceOf(Person person)** - Gets the number of meetings attended by the **Person**
2. **PerformanceOverview#getRateOfAttendanceOf(Person person)** - Gets the percentage of meetings attended by the **Person**
3. **PerformanceOverview#getNumOfTaskDoneOf(Person person)** - Gets the number of tasks completed by this **Person**
4. **PerformanceOverview#getTaskCompletionRateOf(Person person)** - Gets the percentage of the

assigned tasks, which the **Person** completed

Internally, every attribute of the performance of an individual a **HashMap** for that particular attribute. For example, the task completion rate of every individual is stored in a **HashMap<String, RateOfTaskCompletion>** where the key is the string of the name of the member, while the value is the rate of task completion. Every other attribute is stored in similar fashion.

Next, we will demonstrate the process of using this feature to see the performance of each member. The following sequence diagram shows the entire process of calculating the performance of member and showing it, after the **showPerformanceOverview** command is input by the user.

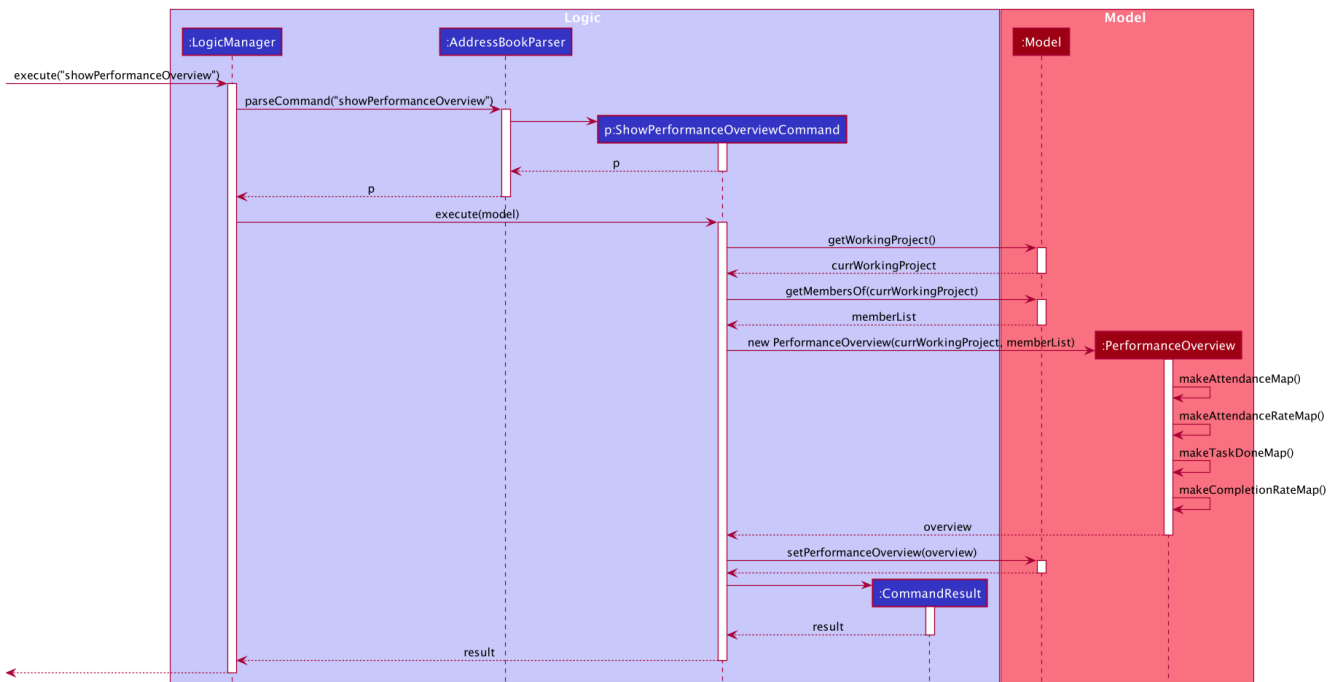


Figure 21. Sequence diagram of the **showPerformanceOverview** command execution

With reference to the sequence diagram above, here are the steps of the command execution:

Step 1. The user input is parsed, and a **ShowPerformanceOverviewCommand** is created.

Step 2. The execution of the **ShowPerformanceOverviewCommand** fetches the **Project** that the user is working on, and also the **Person(s)** involved in the project and returns them as a **List<Person>**.

Step 3. The **List<Person>** and **Project** are used to create the **PerformanceOverview**. Here you can see that when a **PerformanceOverview** is created there are 4 methods being called internally. This is where all the necessary information is taken from the project and members, and used to calculate the different attributes of the member's performance. It is also in those 4 methods, where the **HashMaps** are created.

Step 4. The **PerformanceOverview** is set in the **Model**, and displayed to the user subsequently.

The following activity diagram summarizes the general flow of the execution of the **showPerformanceOverview** command:

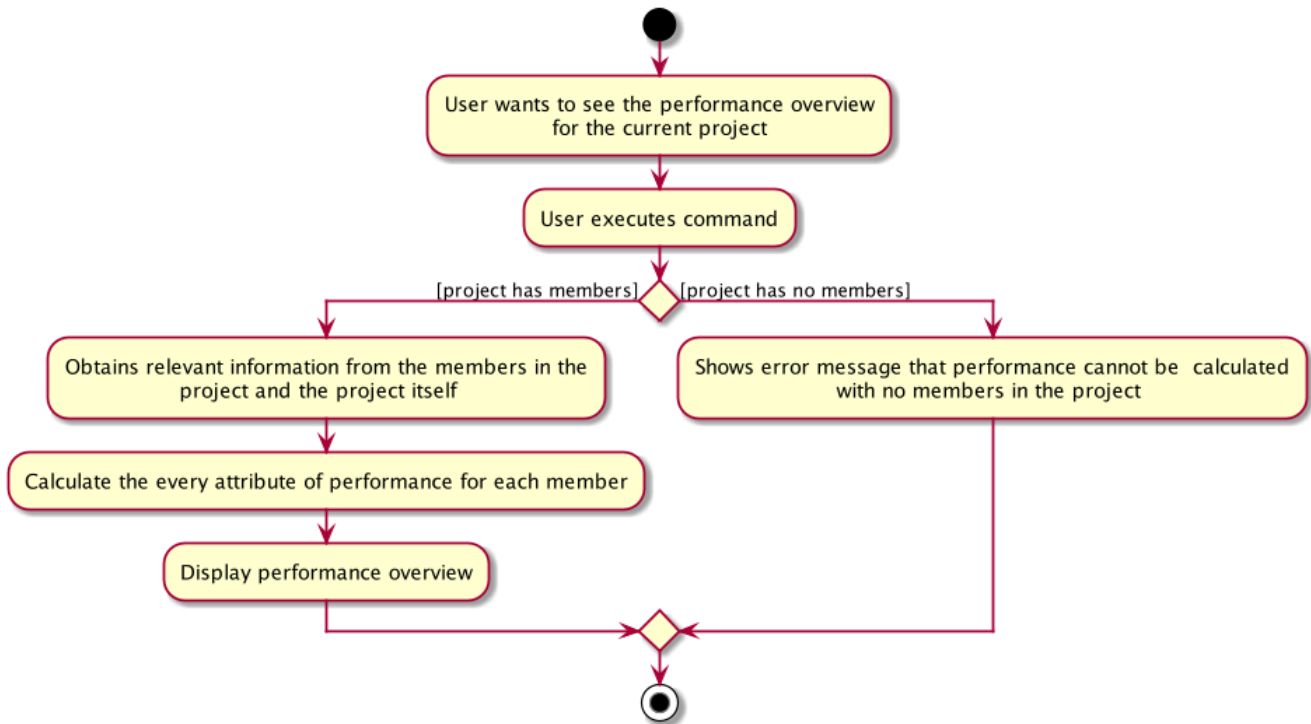


Figure 22. Activity diagram for `showPerformanceOverview`

As shown in the activity diagram, there are only 2 main flows during the execution of this command. If there are members in the project, the performance of the members can be calculated, and the performance overview will be displayed to the user. If there are no members in the project, the user will be shown an error message to tell them that the performance overview cannot be computed.

Design Considerations

Data structure of `PerformanceOverview`

- **Alternative 1(Current implementation):** Currently, every attribute is stored and paired to the member using a `HashMap`. The key is the string representation of the name of the members, and the value is the attribute itself.
 - Pros:
 - Easy to implement.
 - In order to add a new attribute to measure performance, there is only a need to add a new `HashMap`, a method to calculate the values, and a method to retrieve the values.
 - Cons:
 - It is more difficult to iterate through all `HashMaps` to retrieve the values based on your preferred ordering.