# Cheyanne Sim - Project Portfolio

## PROJECT: Sharkie

---

## Overview

Sharkie is a desktop expense tracker application. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 10 kLoC.

## Summary of contributions

- **Major enhancements**:
  - Added People's command, `owe`
    - What it does: allows the user to keep track of what they owe to other people in the addressbook.
    - Justification: This is one of the main feature of Sharkie.
    - Highlights: The implementation was quite challenging as it required an in-depth analysis of design alternatives. For instance, how to store the Transactions, since a Transaction has association to Description, Amount, Date and Tag, and Date itself have different formats to consider. I have implemented the logic, model, storage and ui for this command.
  - Added People's command, `returned`
    - What it does: allows the user to record that he/she has already returned the debt(s).
  - Modified People's command, `find`
    - What it does: allows the user to find people based on their name, phone, email or tags. I have also added the the tags "Debt" and "Loan" to people whom the user owes and lends, so that the user can filter the people whom they lend or owe using the commands `people find t/debt` and `people find t/loan` respectively.
- **Minor enhancements**:
  - Edit existing AB3 commands
    - Removed address field of AB3 as it is unnecessary for Sharkie.
- **Code contributed**: [Functional codes & Test codes]
- **Other contributions**:
  - Enhancements to existing features:
    - Wrote additional tests for exsiting features to increase coverage from 73% to 76% (Pull requests #182)
    - Bug fixing. (Pull request #247)

- Documentation:
  - Update AboutUs page: (Pull request #53)
  - Updated User Guide: (Pull requests #43, #196)
  - Updated Developer Guide: (Pull requests #60, #120, #124, #138, #196, #256)
- Community:
  - PRs reviewed (with non-trivial review comments): (Pull requests #95, #87, #98, #102, #117, #128)
  - Reported bugs and suggestions for other teams in the class: Reported bugs for W17-2 (Issues #1, #2, #3, #4, #5, #6, #7, #9, #9, #10, #11, #12, #13)

# 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.*

# Features

## Contact Management

If you would like to keep the contact details of a person so that you can use the debts and loans features, **Sharkie** can help you do so!

**Sharkie** notes down and remembers contacts that you have entered in a contact list, for easy reference later on! Should their contact details change, you can also edit them or delete them. If you need to find a particular person's contact details, **Sharkie** will look through all your contacts and quickly help you find the contact that you are looking for!

| | |
|---|---|
| **NOTE** | You may visit [people-commands] for more details on how you can manage your contacts with **Sharkie**. |

# Commands

**Sharkie** is filled with a variety of commands that can aid in your financial tracking journey.

For ease of reference, we have segregated them into three parts: general commands, people commands and wallet commands.

**Command Format**

- Words in angle brackets are the parameters to be supplied by the user e.g. in `add n/<name>`, `<name>` is a parameter which can be used as `add n/John Doe`.
- Items in square brackets are optional e.g. `$/<amount> [d/<date:dd/mm/yyyy>]` can be used as `$/5 d/21/02/2020` or as `$/5`.
- People commands are used when you want to do things related to the people tab, e.g. `people add n/<name> p/<phone number> e/<email address>`
- Wallet commands are used when you want to do things related to the wallet tab, e.g. `wallet expense n/<item> $/<price> [d/<date:dd/mm/yyyy>] [t/<tag>]`
- Parameters can be in any order e.g. if the command specifies `n/<name> p/<phone number>`, `p/<phone number> n/<name>` is also acceptable.

## Recording the money you owe: `owe`

Suppose you owe a person money and you want to record the debt, the command you would enter is the `people owe` command.

**Format**: `people owe <person's index> n/<description> $/<amount> [d/<date: dd/mm/yyyy>]`

**Command Format**

The following are the restrictions of `people owe` command, which you would need to take note of:

- The `<person's index>` you entered should be a positive integer, e.g. 1, 2, 3, …
- The `<amount>` should be a valid amount

**NOTE**

The `<person's index>` above refers to the index number shown in the displayed person list in **Sharkie**. It indicates a specific person in the contact list whom you owe money to.
*Still confused? Find out more about what is a person's index.*

The amount of money recorded will be added under your friend's "debts" section. Debts represent the amount of money you owe your friends.
*Still confused? Find out more about the differences between debts and loans.*

**TIP**

The `<date: dd/mm/yyyy>` is optional. If `<date: dd/mm/yyyy>` is not specified, the date that you record the debt will be used.

**Example**:

- Suppose you owe "Grace", who is the fourth person in the contact list, "$5" for "food" on "10 October 2020".

- The command you would enter is `people owe 4 n/food $/5.00 d/10/10/2020`
- This records that you owe "Grace", the fourth person in the contact list, "$5.00" for "food" on "10/10/2020".

**Expected Outcome**:

- Your debt to "Grace" will increase by "$5".

```
Increased debt to Grace by $5.00. You now owe Grace $10.00.
```

## Recording the money you return: `returned`

Suppose you have returned a person a debt, and you want to remove the debt recorded, the command that you would enter is the `people returned` command.

**Format**: `people returned <person's index> [i/<debt's index>]`

**Command Format**

The following are the restrictions of `people returned` command, which you would need to take note of:

- The `<person's index>` and `<debt's index>` you entered should be positive integers, e.g. 1, 2, 3, …

**NOTE**

The `<person's index>` above refers to the index number shown in the displayed person list in **Sharkie**. It indicates a specific person in the contact list whom you returned the money to.
*Still confused? Find out more about what is a person's index.*

The `<debt's index>` above refers to the index number shown in the displayed debt list in **Sharkie**. It indicates a specific debt under the person whom you returned the money to.
*Still confused? Find out more about what is a debt's index.*

Debts represent the amount of money you owe your friends.
*Still confused? Find out more about the differences between debts and loans.*

**TIP**

The `<debt's index>` is optional. Sharkie will remove all debts for the person if the `<debt's index>` is not specified.

**Example**:

- Suppose that you have just returned "Grace", the fourth person in the contact list, the first debt in her debt list.

- The command that you would enter is `people returned 4 i/1`.
- This records that you have returned the money for the first debt of "Grace", the fourth person in the contact list.

**Expected Outcome**:

- The first debt of "Grace" will be removed from her debt list and the unsettled debts to "Grace" will be shown.

```
Reduced debt to Grace by $5.00. You now owe Grace $5.00.
```

## Recording the money you lend: `lend`

Suppose you lend a person money and you want to record the loan, the command you would enter is the `people lend` command.

**Format**: `people lend <person's index> n/<description> $/<amount> [d/<date: dd/mm/yyyy>]`

**Command Format**

The following are the restrictions of `people lend` command, which you would need to take note of:

- The `<person's index>` you entered should be a positive integer, e.g. 1, 2, 3, ...
- The `<amount>` should be a valid amount.

**NOTE**

The `<person's index>` above refers to the index number shown in the displayed person list in **Sharkie**. It indicates a specific person in the contact list whom you lent money to.
*Still confused? Find out more about what is a person's index.*

The amount of money recorded will be added under your friend's "loans" section. Loans represent the amount of money you lend your friends.
*Still confused? Find out more about the differences between debts and loans.*

**TIP**

The `<date: dd/mm/yyyy>` is optional. If `<date: dd/mm/yyyy>` is not specified, the date that you record the loan will be used.

**Example**:

- Suppose you lent "Syin Yi", who is the fifth person in the contact list, "$5" for "dinner" on "10 October 2020".
  - The command you would enter is `people lend 5 n/dinner $/5.00 d/10/10/2020`
  - This records that you owe "Syin Yi", the fifth person in the contact list, "$5.00" for "dinner" on "10/10/2020".

**Expected Outcome**:

- Your loan to "Syin Yi" will increase by "$5".

```
Increased loan to Syin Yi by $5.00. Syin Yi now owes you $8.00.
```

## Listing all contacts: `list`

Suppose that you have just executed the `people find` command. And now, you would like to see the entire list of people in your contact list. The command you would enter is our `people list` command.

**Format**: `people list`

**Example**:

- Suppose you want to view the entire list of people in your contact list.
  - The command that you would enter is `people list`.
  - This will list out your entire contact list.

**Expected Outcome**:

- The details of everyone in the contact list, including their name, phone, email address, debts and loans, will be listed.

```
Listed all persons.
```

## Editing a person : `edit`

Suppose a person has changed his contact details, and you want to update them, the command that you would enter is the `people edit` command.

**Format**: `people edit <person's index> [n/<name>] [p/<phone number>] [e/<email>]`

**Command Format**

The following are the restrictions of `people edit` command, which you would need to take note of:

- The `<person's index>` you entered should be a positive integer, e.g. 1, 2, 3, …
- You should provide at least one of the optional fields.

| | |
|---|---|
| **WARNING** | Existing values will be updated to the new values that you have inputted. |

| | |
|---|---|
| **NOTE** | The `<person's index>` above refers to the index number shown in the displayed person list in **Sharkie**. It indicates a specific person in the contact list.<br>*Still confused? Find out more about what is a person's index.* |

**Example**:

- Suppose you want to update John's email, and John is the first person in your contact list.
  - The command you would enter is `people edit 1 e/johndoe@example.com`.
  - This edits the email address of the first person, John, to be "johndoe@example.com".

**Expected Outcome**:

- In the list of people shown, John's email will be "johndoe@example.com".

```
Edited Person: John Doe Phone: 91234568 Email: johndoe@example.com You owe: $0.00
You lent: $0.00 Tags:
```

## Deleting a person : `delete`

Suppose you would like to delete a person's contact details, the command that you would enter is the `people delete` command.

**Format**: `people delete <person's index>`

**Command Format**

The following are the restrictions of `people delete` command, which you should take note of:

- The `<person's index>` you entered should be a positive integer, e.g. 1, 2, 3, …

| | |
|---|---|
| **WARNING** | Remember to check and ensure that the `<person's index>` that you have inputted corresponds to the correct person. |

| | |
|---|---|
| **NOTE** | The `<person's index>` above refers to the index number shown in the displayed person list in **Sharkie**. It indicates a specific person in the contact list.<br>*Still confused? Find out more about what is a person's index.* |

**Example**:

- Suppose you want to delete "Betsy" from your contact list, and "Betsy" is the first person in the list.
  - The command that you would enter is `people delete 1`.
  - **Sharkie** will delete "Betsy" from the contact list.

**Expected Outcome**:

- "Betsy" will no longer be shown on the list of people.

```
Deleted Person: Betsy Phone: 91234567 Email: something@email.com You owe: $0.00 You
lent: $0.00 Tags:
```

# 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.*

# Architecture

## Architecture

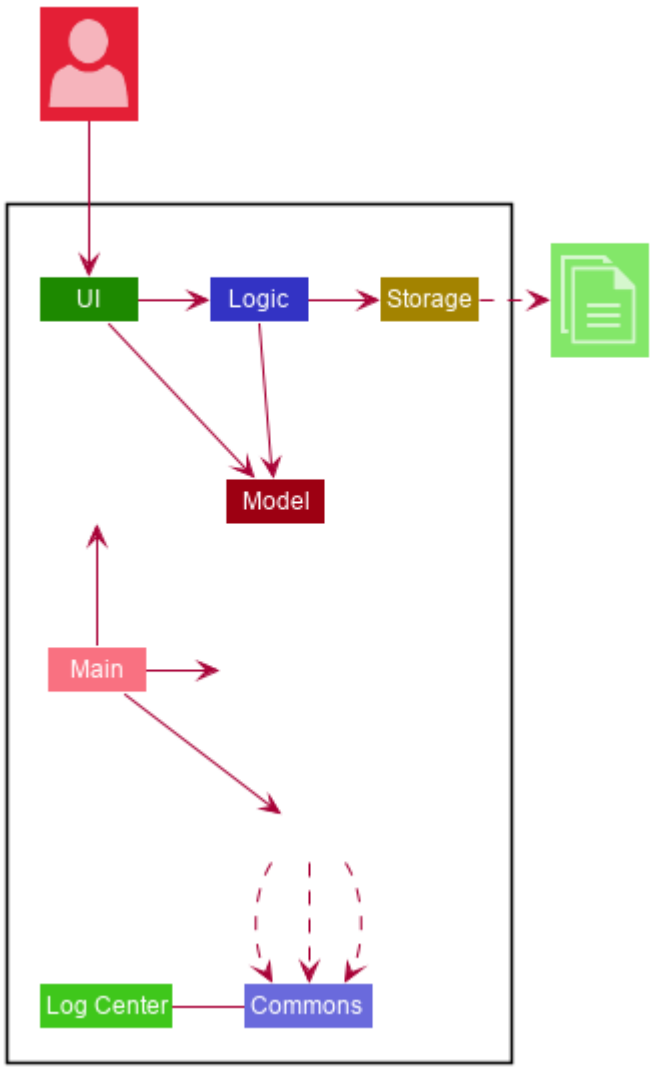



*Figure 1. Architecture Diagram*

The ***Architecture Diagram*** given above explains the high-level design of Sharkie. Given below is a quick overview of each component. **Sharkie** uses the same architecture design as Address Book 3

(AB3).

| | |
|---|---|
| **TIP** | The `.puml` files used to create diagrams in this document can be found in the diagrams folder. Refer to the Using PlantUML guide to learn how to create and edit diagrams. |

`Main` has two classes called `Main` and `MainApp`. It is responsible for,

- At app launch: Initializes the components in the correct sequence, and connects them up with each other.
- At shut down: Shuts down the components and invokes cleanup method where necessary.

`Commons` represents a collection of classes used by multiple other components. The following class plays an important role at the architecture level:

- `LogsCenter` : Used by many classes to write log messages to the App's log file.

The rest of the App consists of four components.

- **`UI`**: The UI of the App.
- **`Logic`**: The command executor.
- **`Model`**: Holds the data of the App in-memory.
- **`Storage`**: Reads data from, and writes data to, the hard disk.

Each of the four components

- Defines its *API* in an `interface` with the same name as the Component.
- Exposes its functionality using a `{Component Name}Manager` class.

For example, the `Logic` component (see the class diagram given below) defines it's API in the `Logic.java` interface and exposes its functionality using the `LogicManager.java` class.

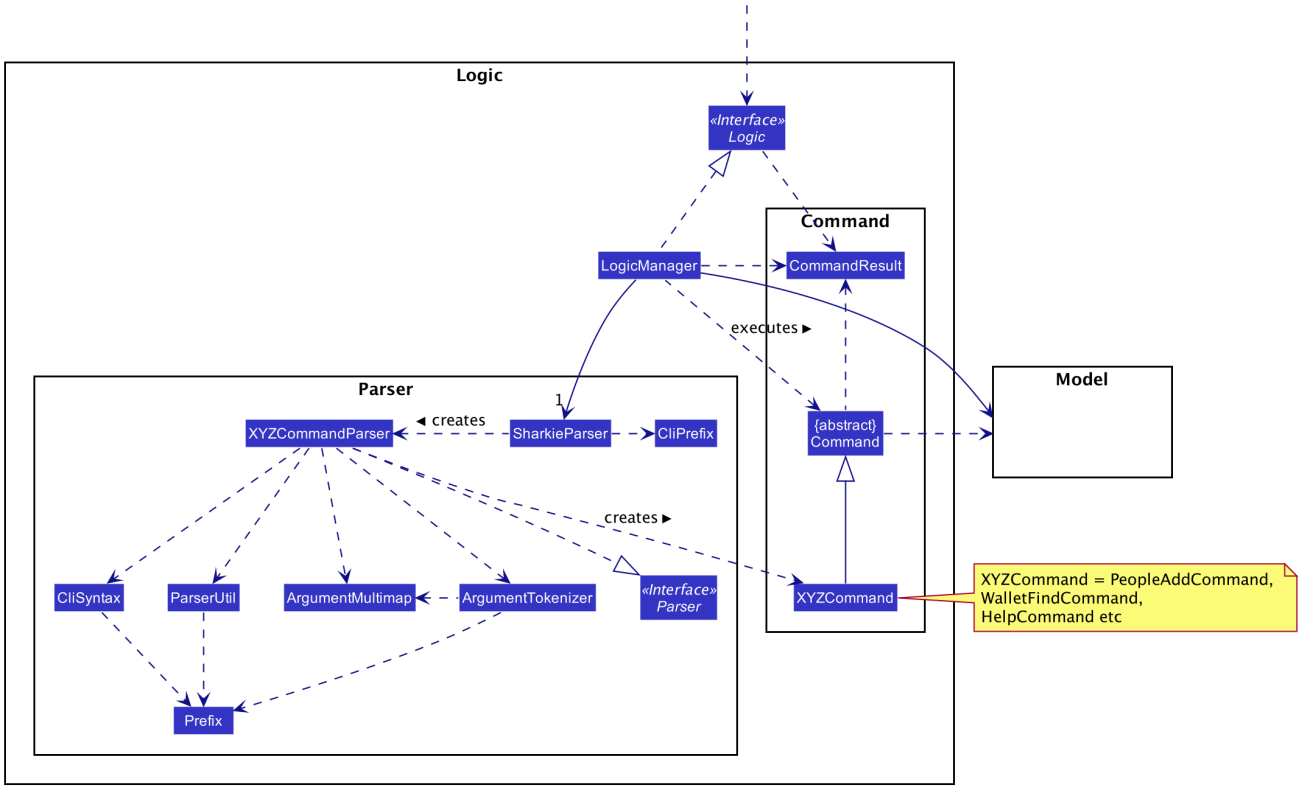



*Figure 2. Class Diagram of the Logic Component*

## How the architecture components interact with each other

The *Sequence Diagram* below shows how the components interact with each other for the scenario where the user issues the command `people delete 1`.

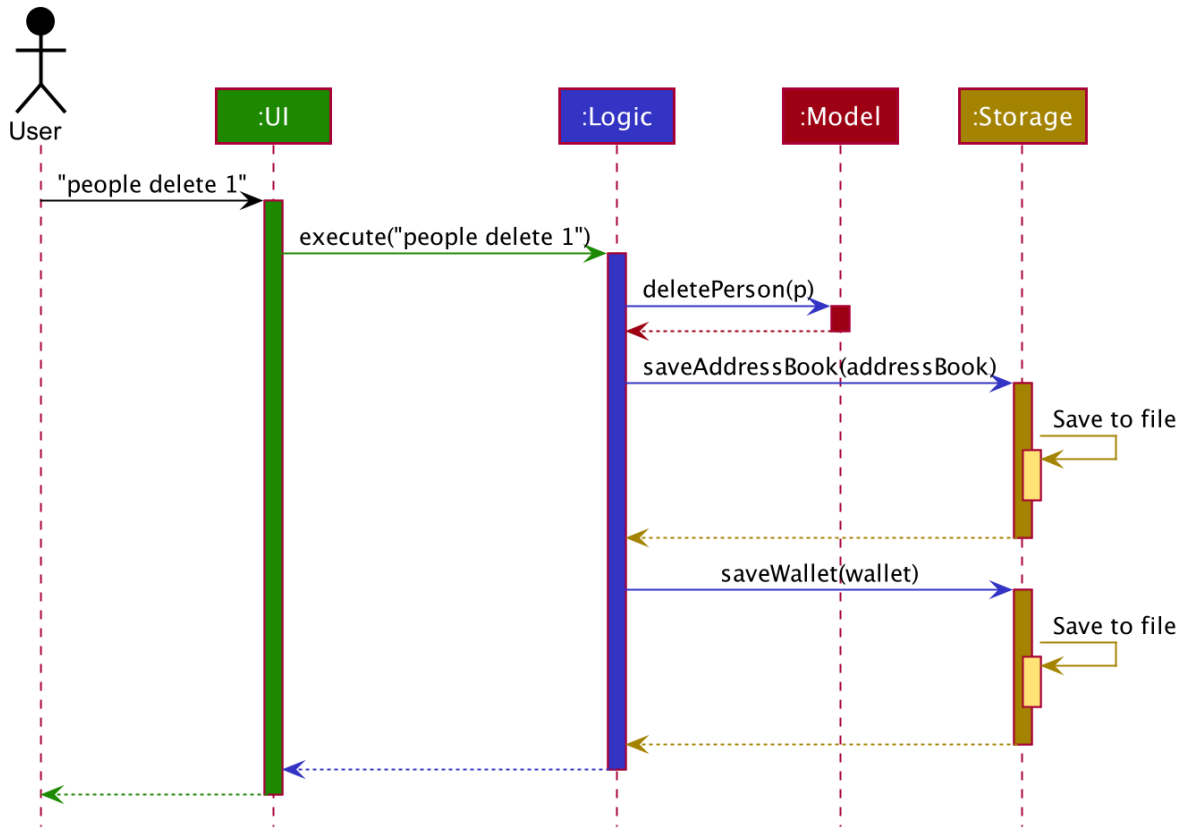



*Figure 3. Component interactions for `people delete 1` command*

The sections below give more details of each component.

# Implementation

## People Owe Command

The `people owe` command is implemented in the class `PeopleOweCommand`.

This command can be accessed from `Logic#execute()`. It records a `debt` of an indicated `Amount` to the `Person` specified by the index.

The following activity diagram illustrates what happens when the user enters a `people owe` command:

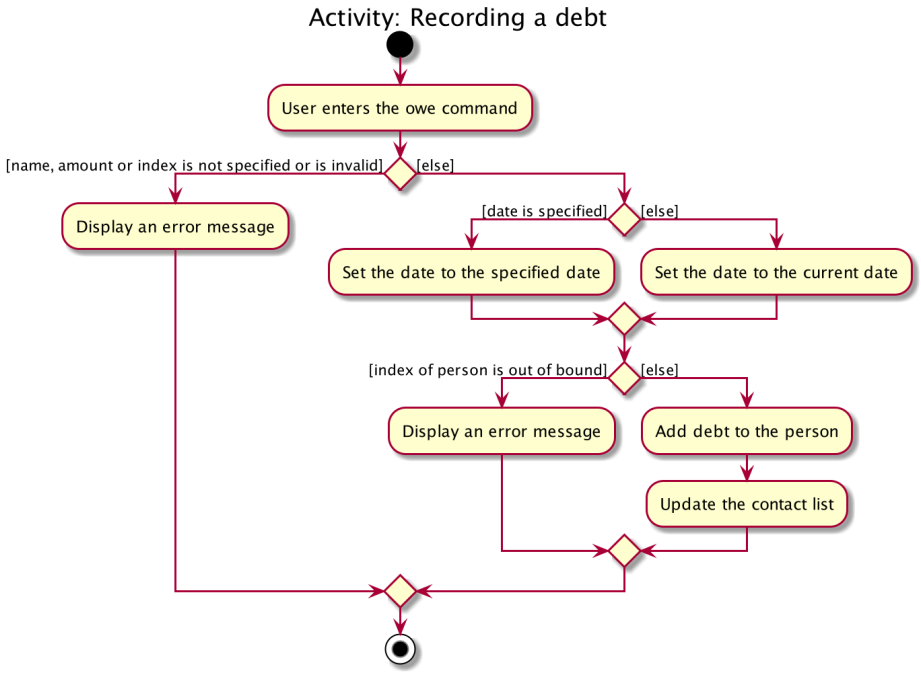


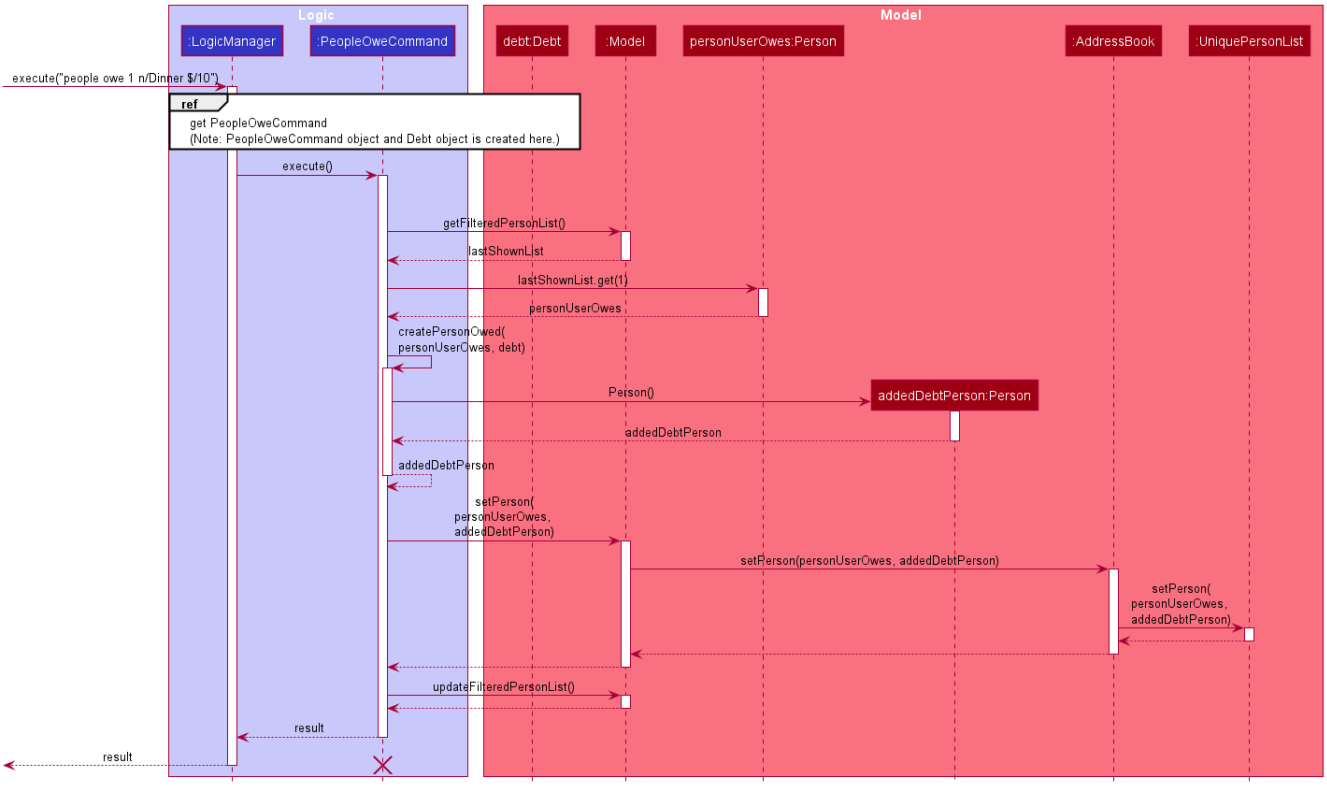

*Figure 4. Activity diagram of the recording a debt*

### Implementation of `people owe` command

1. When entering the debt command, the user will specify the `Person` using the index of the `Person` in the list shown in the GUI.
2. The user should also specify the debt `Description`, `Amount` and optionally, the `Date`.
3. The `PeopleOweCommandParser` will create a `Debt` object based on the details provided, and return the resulting `PeopleOweCommand`.
4. When the `LogicManager` is executing the `PeopleOweCommand`, it will extract the indicated person from the list of `Persons` obtained from the `Model` via `Model#getFilteredPersonList()`
5. A new `Person` with the added `Debt` is created.
6. This new `Person` replace the initial `Person` at the indicated index via `Model#setPerson()` for immutability.

7. The `filteredPersons` in the `Model` is then updated.
8. `CommandResult` is returned.

The following sequence diagram summarizes what happens during the execution of a `people owe` command:



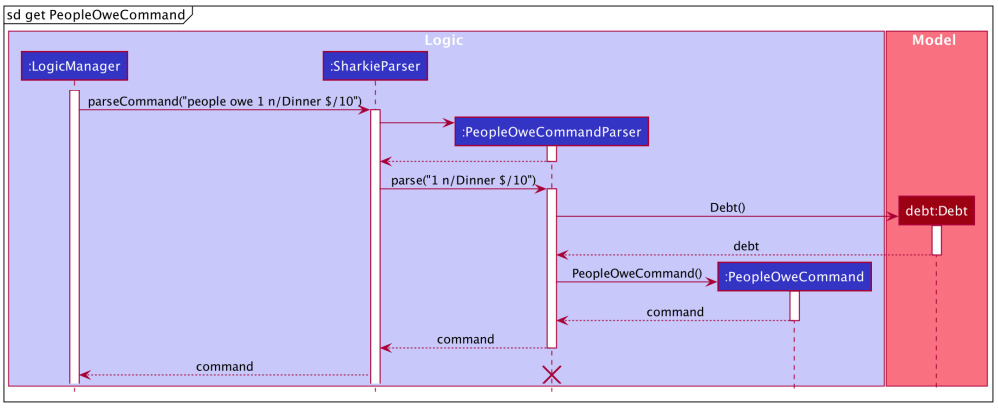




*Figure 5. Sequence diagram of the `people owe` command*

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

## Design Considerations

**Aspect: Keeping track of `Debt` of a `Person`.**

- **Alternative 1 (current choice):** Each `Person` has a list of `Debt` objects, each `Debt` object has `Description`, `Amount` and `Date`.

- Pros: Able to record more information about a `Debt`.
- Cons: `Sharkie` only allows the return of a `Debt` all at once, i.e., the user cannot return a `Debt` partially.

- **Alternative 2:** Each `Person` has one `Debt` object.
  - Pros: Easier to store the `Debt` object, only have to keep track of the total debt `Amount` and the `Date` of the debt. The user can return any `Amount` to the `Person`, and `Sharkie` will just deduct the total `Amount` of debt accordingly.
  - Cons: Storing the `Date` is problematic, as it questions whether the `Date` of the first borrowing or latest borrowing should be stored. Furthermore, there is no breakdown of `Debt` details if the user wants to recall why he owed a `Person` money.

# Appendix D: Non-Functional Requirements

1. Sharkie should work on any mainstream OS as long as it has Java 11 or above installed.
2. Sharkie should be able to hold up to 100 persons and 100 transactions without a noticeable sluggishness in performance for typical usage.
3. University students with above average typing speed for regular English text (i.e. not code, not system admin commands) should be able to accomplish most of the tasks faster using commands than using the mouse.
4. Sharkie should be for a single user.
5. Sharkie needs to be developed incrementally with high cohesion and utilising CS2103T coding standards for maintainability.
6. The data used by Sharkie should be stored locally and should be in a human editable file.
7. The Sharkie JAR file size should be less than 100Mb.

# Appendix G: Instructions for Manual Testing

1. Recording the money you owe a person
   a. Prerequisites: The person whom you owe exists in the person list.
   b. Test case: `people owe 1 n/Breakfast $/5.00`
   Expected: A debt named *Breakfast* with *$5.00* is added into the debt list of the first person.
   c. Test case: `people owe 1 n/Breakfast $/5.00 d/02/02/2020`
   Expected: A loan named *Breakfast* with *$5.00*, recorded under the date *02/02/2020* is added into the debt list of the first person. Total amount of money, which you lent to the first person is shown in the result display.
   d. Test case: `people owe 0 n/Breakfast $/5.00`
   Expected: The loan is not recorded. Error details shown in the result display.
   e. Other invalid `people owe` commands to try:
      - `people owe`,
      - `people owe 1`,

- `people owe 1 n/Laksa`,
- `people owe 1 $/5.00`,
- `people owe n/Laksa $/5.00`,
- `people owe x n/Breakfast $/12.00` (where x is larger than the person list size),
- `people owe x n/Breakfast $/12.00` (where x is a negative number),
- `people owe x n/Breakfast $/12.00` (where x is a non-integer),
- `people owe 1 n/Breakfast $/x` (where x is a negative number),
- `people owe 1 n/Breakfast $/x` (where x is greater 92233720368547758.07),
- `people owe 1 n/Breakfast $/x` (where x has more than 2 decimal places) or
- `people owe 1 n/Breakfast $/x` (where x is not a number).
  Expected: Similar to previous

2. Recording the money you returned to a person
   a. Prerequisites: The person whom you returned to exists in the person list.
   b. Test case: `people returned 1 i/1`
   Expected: The first debt of the first person is deleted from the debt list. Remaining amount of debt, which have yet settled by the first person is shown in the result display.
   c. Test case: `people returned 0 i/1`
   Expected: No debt is deleted. Error details shown in the result display.
   d. Other invalid `people returned` commands to try:
      - `people returned`,
      - `people returned 1`,
      - `people returned i/1`,
      - `people returned x i/1` (where x is larger than the person list size),
      - `people returned x i/1` (where x is a negative number),
      - `people returned x i/1` (where x is a non-integer value),
      - `people returned 1 i/x` (where x is larger than the debt list size),
      - `people returned 1 i/x` (where x is a negative number or zero) or
      - `people returned 1 i/x` (where x is a non-integer value)
      Expected: Similar to previous

3. Adding a person
   a. Test case: `people add n/John Doe p/91234567 e/John@example.com`
   Expected: A person named *John Doe* with phone number *91234567* and email *John@example.com* is added to the contact.
   b. Test case: `people add n/John Doe p/91234567`
   Expected: No person is added. Error details shown in the result display.
   c. Other invalid `people add` commands to try:
      - `people add`,

- `people add n/Invalid! p/99999999 e/John@example.com`,
- `people add n/John Doe p/123 e/John@example.com`,
- `people add n/John Doe p/123 e/invalid`,
- `people add n/John Doe e/John@example.com`,
- `people add p/91234567 e/John@example.com`,
- `people add n/John Doe`,
- `people add p/91234567` or
- `people add e/John@example.com`
  Expected: Similar to previous.

4. Editing the details of a person
   a. Prerequisites: The person whom you want to edit exists in the person list.
   b. Test case: `people edit 1 n/John Doe p/91234567 e/John@example.com`
   Expected: The first person name, phone number and email will be changed to *John Doe*, *91234567* and *John@example.com* respectively.
   c. Test case: `people edit 0 n/John Doe p/91234567 e/John@example.com`
   Expected: No person is edited. Error details shown in the result display.
   d. Other valid `people edit` commands to try:
      - `people edit 1 n/Bob p/88888888`,
      - `people edit 1 n/Cate e/cate@example.com`,
      - `people edit 1 p/66666666 e/cate@example.com`,
      - `people edit 1 n/Alice`,
      - `people edit 1 p/99999999` or
      - `people edit 1 e/email@example.com`,
      Expected: Similar to (b).
   e. Other invalid `people edit` commands to try:
      - `people edit`,
      - `people edit 1`,
      - `people edit 1 n/Invalid!`,
      - `people edit 1 p/123`,
      - `people edit 1 e/invalid`,
      - `people edit x n/Something` (where x is larger than the person list size),
      - `people edit x n/Somthing` (where x is a negative number) or
      - `people edit x n/Something` (where x is a non-integer value)
      Expected: Similar to (c).

5. Finding a person
   a. Prerequisites: The person whom you want to find exists in the person list.

b. Test case: `people find n/Alex Bernice`
Expected: People who are have *Alex* or *Bernice* in their name (case insensitve) will be listed.

c. Test case: `people find t/debt` Expected: People whom you owe money to will be listed. ..Test case: `people find t/loan` Expected: People whom you lend money to will be listed.

d. Test case: `people find n/Alex p/91234567`
Expected: No person found. Error details shown in the result display.

e. Other valid `people find` commands to try:

- `people find p/93210283`,
- `people find p/9321 9927`,
- `people find e/@example`,
- `people find e/irfan@example.com`,
- `people find p/phone` or
- `people find t/debt loan`
Expected: Similar to (b).

> **NOTE** `people find p/phone` is a valid command even though *phone* is not a valid phone number. However, no person will be listed since no one has *phone* as their phone number.

f. Other invalid `people find` commands to try:

- `people find d/invalidTag`
Expected: Similar to (d).

6. Listing everyone

a. Test case: `people list`
Expected: Everyone in the contacts will be listed.

7. Deleting a person while all persons are listed

a. Prerequisites: List all persons using the `people list` command. The person who you want to delete exists in the person list.

b. Test case: `people delete 1`
Expected: First contact is deleted from the list. Details of the deleted contact shown in the result display.

c. Test case: `people delete 0`
Expected: No person is deleted. Error details shown in the result display.

d. Other invalid `people delete` commands to try:

- `people delete`,
- `people delete x` (where x is larger than the person list size),
- `people delete x` (where x is a negative number) or
- `people delete x` (where x is a non-integer value)
Expected: Similar to previous.

8. Deleting everyone

    a. Test case: `people clear`
    Expected: The list of people will be empty.

| NOTE | If you have deleted everyone in the addressbook, and would like to retrieve some sample data to test **Sharkie**, simply delete **data/addressbook.json**. |
|---|---|