

Lim Kang Yee – Project Portfolio for treasurerPro (tP)

1. Introduction

This project portfolio briefly introduces the project, treasurerPro and outlines my contributions to the project and showcases the key features I implemented.

1.1. About the Team

My team of 5 members consists of 4 year 2 Computer Science Undergraduate students, including me, and another year 4 Computer Engineering Undergraduate student.

1.2. About the Project

This project is part of the module 'Software Engineering Project CS2103T' where we were tasked to develop a basic command line interface desktop application by morphing or enhancing an existing AddressBook desktop application. Our team decided to incorporate and morph the AddressBook application as part of our all-in-one application which enables treasurers or members of Co-Curricular Activities (CCA) Clubs and Societies to manage their club finances, reimbursements, inventory and member's contact details easily. The duration of our project was 13 weeks.

1.3. Key to the Icons and Formatting Used in the Document



This symbol indicates extra information or definition.

Model : Text with this font and grey highlight indicates a component, class or object in the architecture of the application.

command : Text with this blue font and grey highlight indicates a command that can be inputted by the user.

1.4. Introduction of treasurerPro

This desktop application consists of 6 tabs, a command box for users to input their commands and a response box for Leo, our lion mascot. Each tab serves a different purpose that helps treasurers and members better manage their club or society's finances.

Home Tab keeps track of individual transactions. Members Tab keep track of all contact details of members. Reimbursements Tab keeps track of reimbursement records for members that have spent money for the club activities. Inventory Tab keeps track of items for sale. Cashier Tab helps the club with on-site sales and directly inputs all revenue into the system. Overview Tab allows

treasurers to plan the club's finances. This is what our application looks like when it is first opened (graphical user interface for treasurerPro):

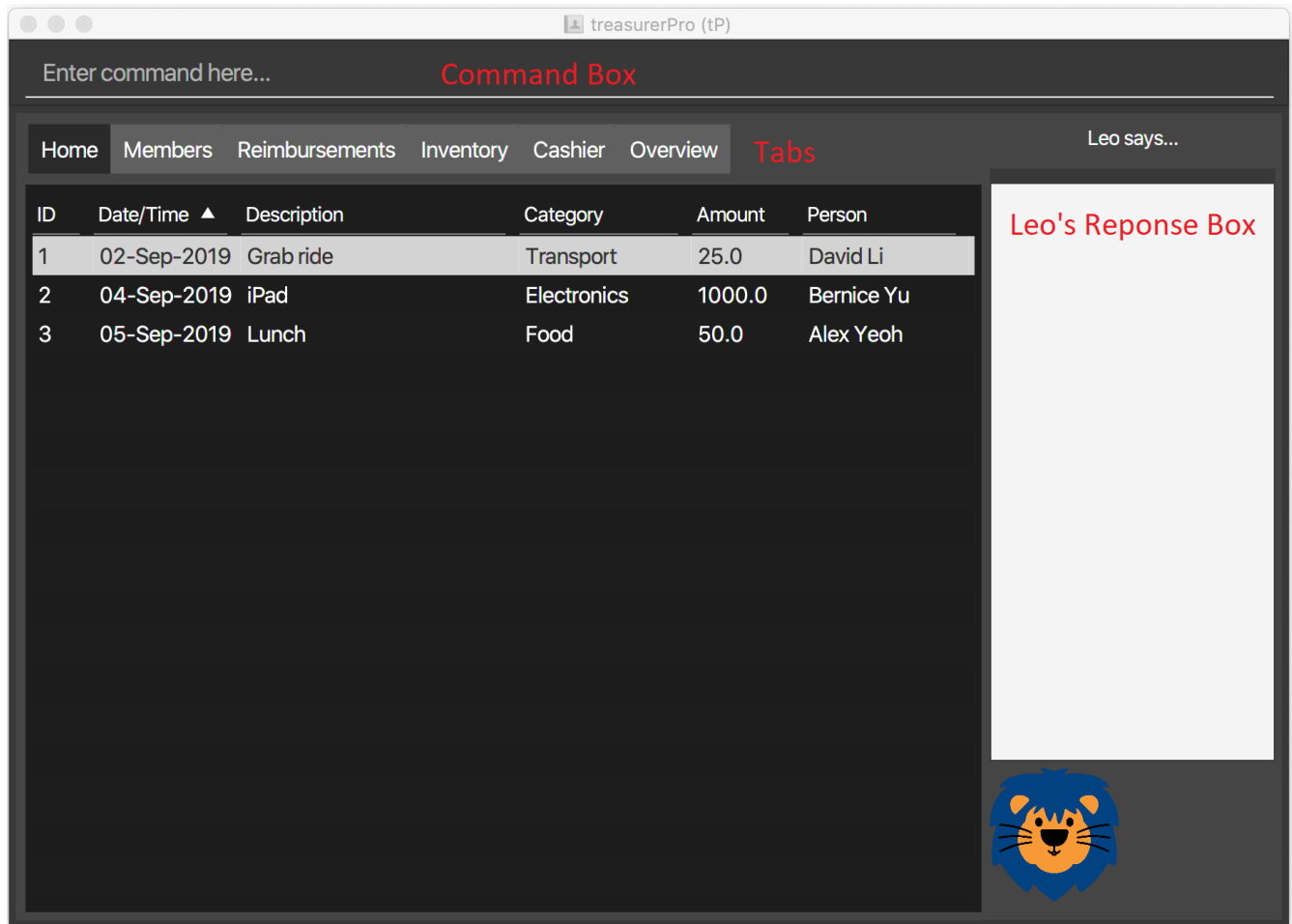


Figure 1. Graphical User Interface of treasurerPro

2. Summary of Contributions

My role was to design and write the code for the features of Home Tab. The following sections shows a summary of these features, as well as the relevant documentation I have added to the user and developer guides in relation to these features.

Enhancements

- Deleting Transactions of a Specific Member
 - What it does: The Delete Command allows all transactions linked to the specified member in the command to be deleted at once.
 - Justification: This feature allows the user to clear all transaction records linked to a specific member if the member has left the CCA since a member's details can only be deleted in the Members Tab if the member does not have existing transaction records.
 - Highlights: This feature is a faster way to delete transactions of a specific member, especially if the member has many transaction records. This works well with the deletion of member feature in Members Tab.
- Sorting Transactions in a Specific Order

- What it does: Each transaction records consists of a date, description, category, amount of money and person accountable for the transaction. Thus, the Sort Command helps to sort the transactions by the alphabetical order of the person's name, by the date (from oldest to most recent) or amount (from smallest to largest).
- Justification: It is useful for users to keep track of transactions and view the transaction records according to different priority.
- Highlights: This command can be extended easily to allow for sorting of transactions in the reverse order or with other different orders.

Code Contributed

Please click these links to view the code I have contributed for Home Tab: [Overall Commit Activity and Code](#), [Functional Code](#) , [Test Code](#)

Other Contributions

- Community
 - Reviewed Pull Requests (With Non-trivial Review Comments): [\(PR #98\)](#)
 - Added Detailed Explanation of Added Code in Pull Requests to Help Understanding By Other Members: [\(PR #42\)](#), [\(PR #201\)](#)
 - Helped to Debug Code in Other Member's Package: [\(PR #126\)](#)
- Integration
 - Integrated **Transaction Tab** with **Reimbursements Tab**: [\(PR #49\)](#)
 - Integrated the Original **AddressBook** into the **Members Tab** in Graphical User Interface: [\(PR #42\)](#)
 - Integrated the Edit and Delete Command of **AddressBook** with the Logic of **Transaction Tab**: [\(PR #49\)](#), [\(PR #85\)](#)
- Documentation
 - Added Implementation Details for Home Tab, Model Component and Storage Component to the Developer Guide: [\(PR #154\)](#), [\(PR #199\)](#)
 - Added to User Stories and the Guide to Use the Home Tab to the User Guide: [\(PR#22\)](#), [\(PR #197\)](#)

3. Contributions to the User Guide

The following section shows my additions to the treasurerPro User Guide for the **Home Tab** features which showcases my ability to write documentation to guide non-technical end-users on how to use treasurerPro.

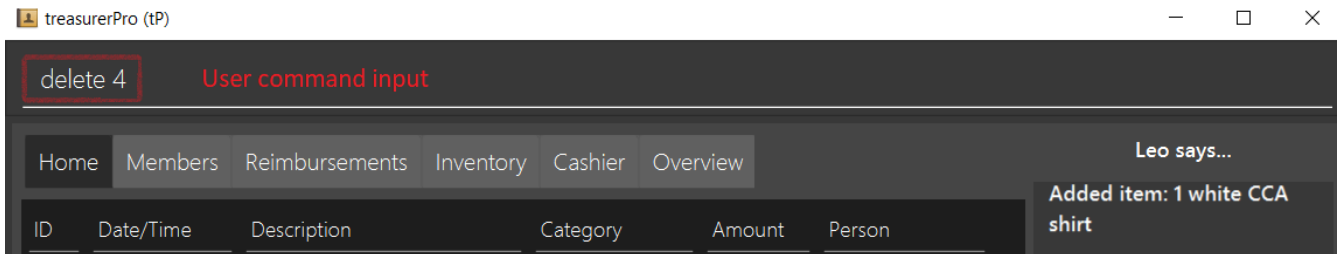
3.1. Current Enhancement

{Start of First Extract from User Guide}

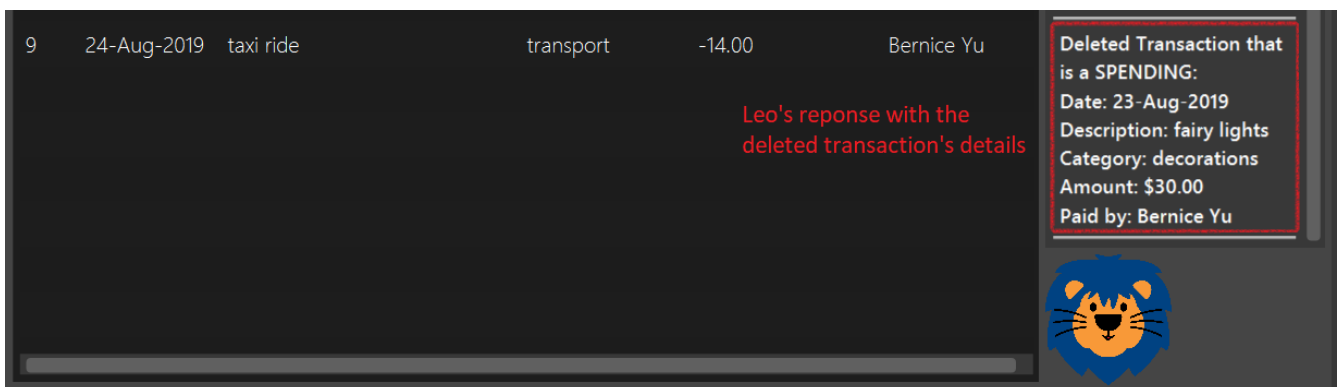
5.1.2. To Delete Transactions in the Home Tab

This command allows you to delete either all transactions of a person or a single transaction of a specific ID from the table.

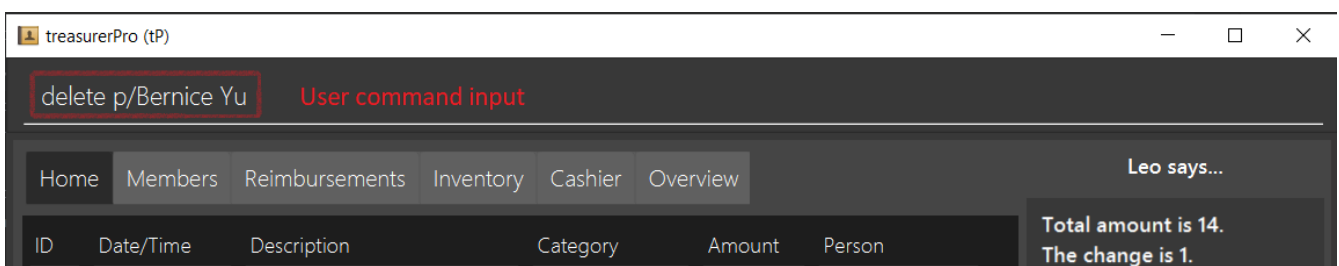
- Command: **delete ID** or **delete p/PERSON**
- Examples
 - Delete by ID: **delete 1** - Deletes the transaction at the index shown in the table. If the table shows a filtered list of transactions due to the find command detailed in [Section 5.1.5. To Find Transactions that Match Keywords in the Home Tab](#), the 1st transaction in the filtered list is deleted.
 - Delete by Person: **delete p/Alex Yeoh** - Deletes all transactions of the person in full list of transactions.
- Steps for Deleting by ID
 1. Type the command with the ID of the transaction to be deleted as shown in the screenshot below:



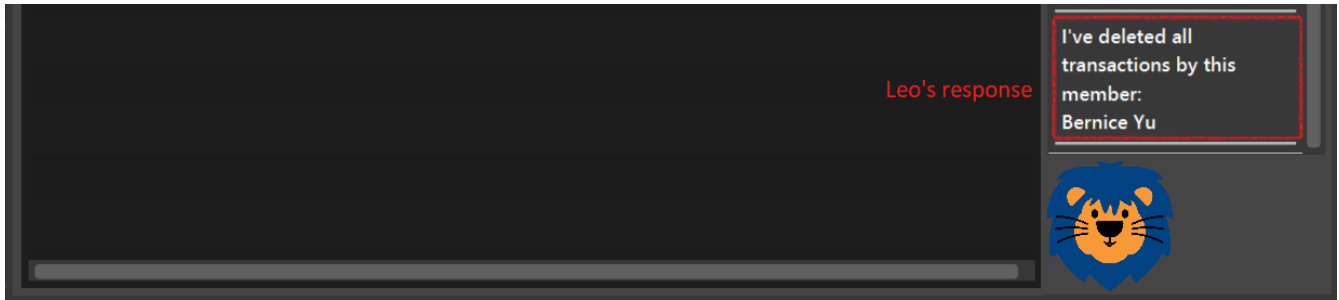
2. Hit **Enter**. Leo will respond with a success message and the transaction will be removed from the table as shown below:



- Steps for Deleting by Person
 1. Type the command with the person's name to delete all transactions related to that person, as shown in the screenshot below:



2. Hit **Enter**. Leo will respond with the success message and the transaction(s) will be removed from the table as shown below:



For both delete commands, if the transaction(s) deleted was part of a pending reimbursement record, it will also be removed from that reimbursement record. On the other hand, if the person entered is not part of our database as shown in the Members Tab, Leo will respond with a message to inform you. If the person does not have any transactions, Leo will also respond with a message to inform you.

{End of First Extract}

{Start of Second Extract from User Guide}

5.1.4. To Sort Transactions in the Home Tab

This command sorts the table of transactions into the specified order for viewing and carrying out of subsequent commands.

- To sort
 - By date (from oldest to most recent): **sort date**
 - By name (from alphabetical order of name): **sort name**
 - By amount (from smallest to largest in amount): **sort amount**
 - Undo sort: **sort reset**



The undo sort command allows you to view the table of transactions in the order originally shown when the application was initially opened.

- Steps
 1. Type the any one of the above command in blue font into the command box.
 2. Hit **Enter**. The table will now be sorted in the specified order for you.

{End of Second Extract}

4. Contributions to the Developer Guide

The following section shows my contribution to the treasurerPro Developer Guide for the **Home Tab** features which showcases my ability to write documentation of technical concepts and the technical-depth of my contributions to the project.

4.1. Current enhancement

{Start of Extract from Developer Guide}

3.1.2. Delete Feature

This section explains the implementation of the delete command in Home Tab. This feature allows for 2 types of deletion, by the index shown in the table or by the person's name. Inputting the person's name will cause all transactions linked to that person to be deleted. The following activity diagram shows the steps needed to delete a new transaction:

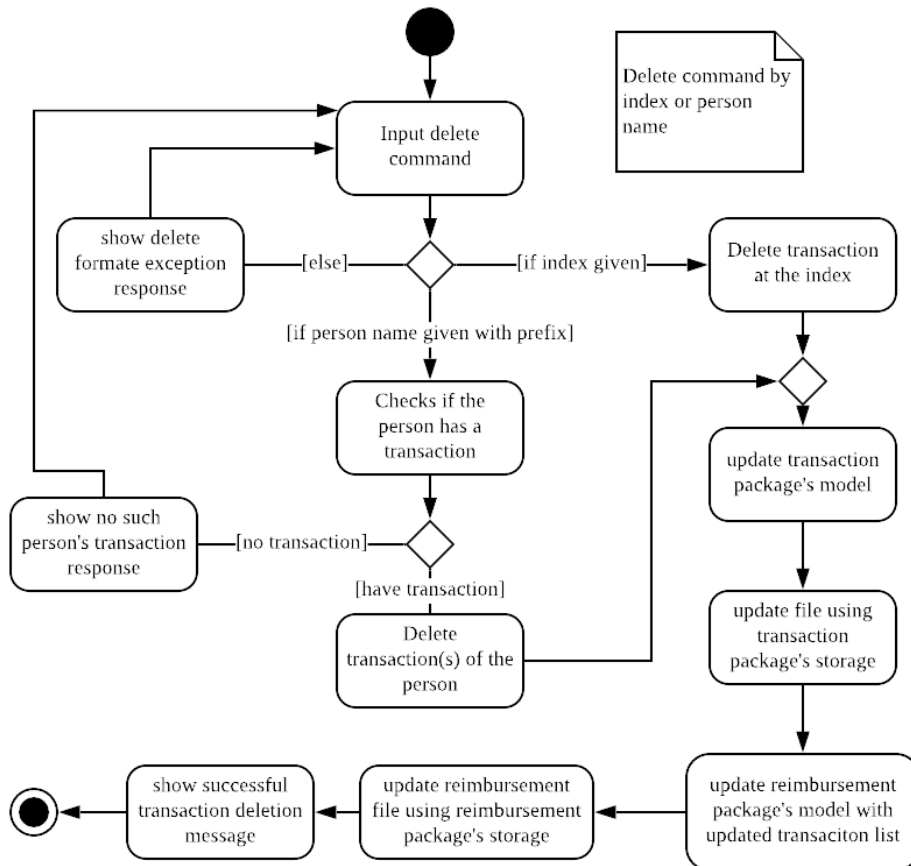


Figure 17. Activity diagram of the delete command being executed in Home Tab (transaction package)

The above activity diagram assumes the index to be within the bounds of the table but if it is not, a response will be shown about the incorrect input. Also, as shown above, responses will be shown to indicate if an input is incorrect or when a successful deletion is done.

For the implementation, the `DeleteCommandParser` which is represented in 2.3. Logic component: Figure 5 as a generic parser creates either a `DeleteIndexCommand` or `DeleteNameCommand` object according to the user input. The following sequence diagram shows how the `DeleteNameCommand` is created with the definition of `CheckAndGetPersonByNameOnlyModel#getPersonByName` defined in 3.1.1. Add Command Feature:

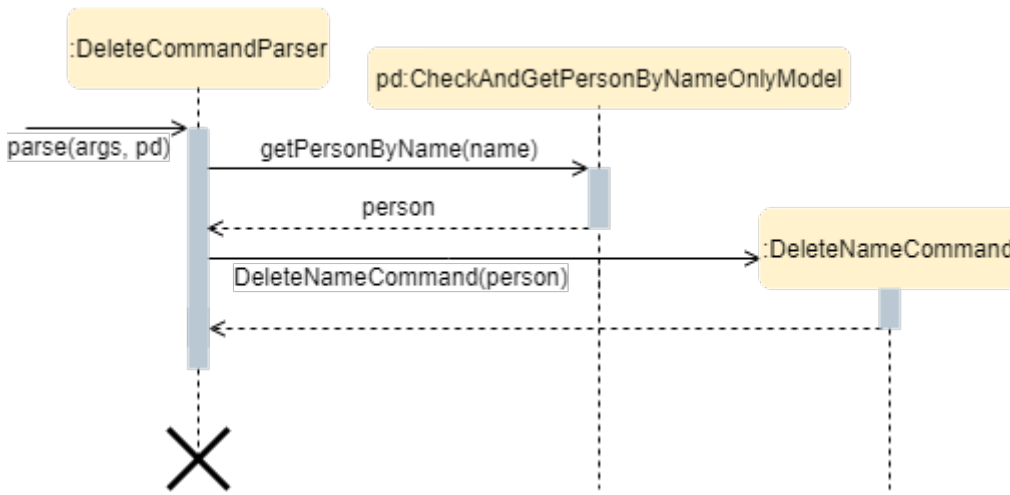


Figure 18. Sequence diagram of the parser for the delete command to delete by the **Person** 's name

The following sequence diagram shows how the command to delete transactions according to the specified name works after it is created which is referenced in [2.3. Logic component: Figure 5](#):

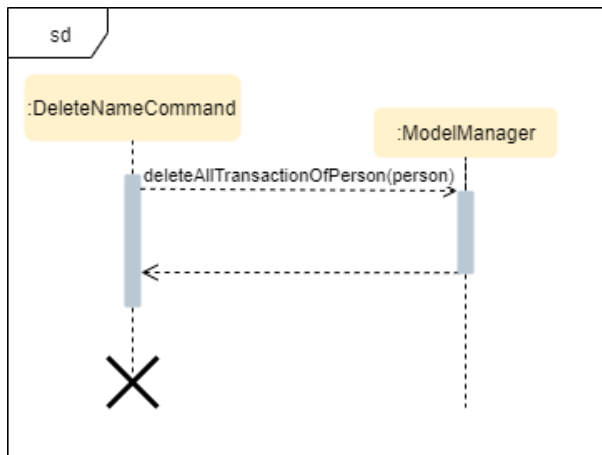


Figure 19. Sequence diagram of the delete command being executed in Home Tab (transaction package)

In addition, the **Model#resetPredicate()** method defined in [3.1.1. Add Command Feature](#) is not called in the **DeleteNameCommand**. If the prior input is a find command and the list at the start of the activity diagram shows a filtered list, the table in the UI will continue to show the filtered list after the current delete command. To view the full transaction list, the user would be required to enter the back command where **BackCommand** calls **Model#resetPredicate()**. The sequence diagram for the **BackCommand** is shown in the following section [3.1.3. Back Command Feature](#)

After this, the list of transactions and Reimbursement Tab is updated as shown in [3.1.1. Add Command feature: Figure 14](#) and [3.1.1. Add Command feature: Figure 15](#) respectively.

The implementation of the command to delete transactions according to the specified index would be similar but does not require interaction with the **CheckAndGetPersonByNameOnlyModel** from the **AddressBook** in the person package inside its branch in **DeleteCommandParser**. In **DeleteIndexCommand**, it also calls the **Model#findTransactionInFilteredListByIndex(index)** and **Model#deleteTransaction(index)** instead which deletes the **Transaction** object according to its position in the filtered transaction list.

3.1.3. Back Command Feature

This section explains the implementation of the back command feature in Home Tab. The **BackCommand** is not initialised by a specific command parser as shown in as shown in 2.3. **Logic component: Figure 5** but initialised by the **TransactionTabParser** instead. The following detailed sequence diagram shows how the back command works:

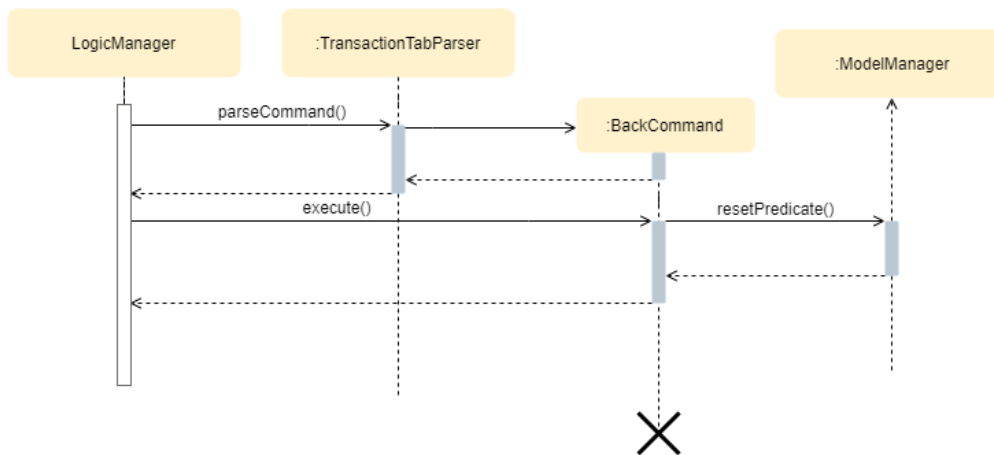


Figure 20. Sequence diagram of the back command being executed in Home Tab (transaction package)

3.1.4. Sort Command Feature

This section explains the implementation of the sort command feature in Home Tab. The **SortCommand** allows for 3 types of sort, by name in alphabetical order, by amount (from least to most) and by date (from oldest to most recent).

The following sequence diagram shows how the sort command works which is referenced in 2.3. **Logic component: Figure 5**:

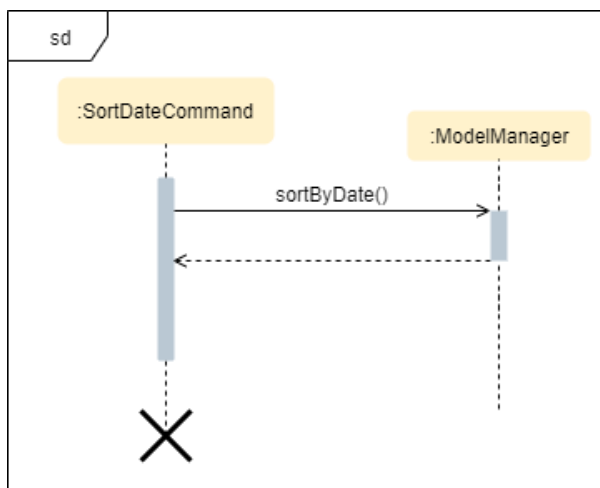


Figure 21. Sequence diagram of the sort command being executed in Home Tab (transaction package)

When a user inputs the sort command, it is only checked that it is one of the 3 types or it will show a response about the incorrect user input. When it is successfully sorted, there will also be a response message shown.

Similar to the delete command in 3.1.2. **Delete Command Feature**, the `Model#resetPredicate()` method is not called. If the UI table is showing a filtered list of transactions, the back command has to be entered to call that method from **BackCommand** to reset the predicate.

3.1.5. Overall Design Considerations

This section's table explains the design considerations for some implementations in the Home Tab.

Alternative 1	Alternative 2	Conclusion and Explanation
<code>ModelManager</code> contains 2 attributes that point to a <code>TransactionList</code> object in original order and a <code>TransactionList</code> object for viewing in the UI table that can be sorted such that when <code>sort reset</code> is called, the <code>TransactionList</code> for viewing can be set to be equals to the original one.	<code>ModelManager</code> contains only the shown <code>TransactionList</code> that can be sorted and reads from the data file to get the <code>TransactionList</code> object in original order when <code>sort reset</code> is called.	Alternative 1 was implemented. Alternative 1 allows the data file to be exported in the desired order anytime while <code>treasurerPro</code> is running while alternative 2 meant that the data file would be updated only when <code>treasurerPro</code> is exited. The implementation is shown below the table in Figure 22 .
The Members Tab's <code>Model</code> interface is passed as parameters into Transaction Tab's <code>Logic</code> to give <code>Logic</code> access to all public methods of <code>ModelManager</code> .	A new interface is made to allow the only used method of Members Tab's <code>ModelManager</code> to be accessed in Transaction Tab's <code>Logic</code> .	Alternative 2 was implemented. The new interface acts as a facade for <code>ModelManager</code> which prevent unwanted modifications to <code>AddressBook</code> . The interface implemented is <code>CheckAndGetPersonByNameOnlyModel</code> as shown below the table in Figure 23 .
An <code>ArrayList</code> is used to store <code>Transaction</code> objects in <code>TransactionList</code> .	A <code>LinkedList</code> is used to store <code>Transaction</code> objects in <code>TransactionList</code> .	Alternative 1 was implemented. An <code>ArrayList</code> has better performance for the set and get methods than a <code>LinkedList</code> which would be used frequently in <code>ModelManager</code> .

```
public class ModelManager implements Model {
    private final Logger logger = LogsCenter.getLogger(getClass());
    private final TransactionList transactionList;
    private TransactionList filteredList;
```

Figure 22. Code snippet of `ModelManager` class with 2 `TransactionList` objects

```
public interface CheckAndGetPersonByNameModel {

    /**
     * Retrieves the person with the same name from the Address Book.
     * @param name Specified name
     * @return Person of specified name
     */
    Person getPersonByName(String name);
```

Figure 23. Code snippet of `CheckAndGetPersonByNameOnly` facade class for `ModelManager` from Members Tab

{End of Extract}