Evon Dong Bing Bing – Project Portfolio for treasurerPro (tP)

1. Introduction

This project portfolio briefly introduces the project, treasurerPro and outlines my contributions and explanations about my features.

1.1. About the team

My team consists of five software engineering students who are all taking the module CS2103T. Four of us, including me, are year 2 Computer Science students and the other is a year 4 Computer Engineering student.

1.2. About the project

This project is part of the module - Software Engineering Project, CS2103T. We were tasked to develop a basic command line interface [1] (CLI) desktop application by morphing or enhancing an existing AddressBook desktop application over a period of 2 months. My team decided to morph and incorporate the AddressBook application as part of our all-in-one expense tracker. This application enables treasurers and members of Co-Curricular Activities (CCA) Clubs and Societies to regulate their club expenses, manage the stocks in the inventory and retrieve records of reimbursements of its members and their contact details easily.

1.3. Typographical Conventions

This section covers the key icons and formatting used in this document.

This symbol indicates important information or definition.

`command` : Text with blue font and grey highlight indicates a command that can be inputted into the command line by the user and be executed by the application.

`Model` : Text with grey highlight indicates a component, class, or an object in the architecture of the application.

1.4. Introduction of treasurerPro

This section briefly summarises the main features of treasurerPro.

This desktop application consists of 6 tabs, a command box for user inputs and a response box for Leo, our lion mascot. Each tab serves a different purpose, but every tab aims to facilitate seamless and effortless managing of the organisation's finances. The following shows the various tabs and their respective purposes:

- `Home Tab`: keeps track of individual transactions
- `Members Tab`: records contact details of all members
- `Reimbursement Tab`: handles reimbursement records for members who fork in their money
- `Inventory Tab`: keeps track of items in the inventory
- `Cashier Tab`: supports cashiering duties and directly input sales into the system
- `Overview Tab`: gives an analysis of the organisation's finances for future planning

This is what the application looks like when it starts up:

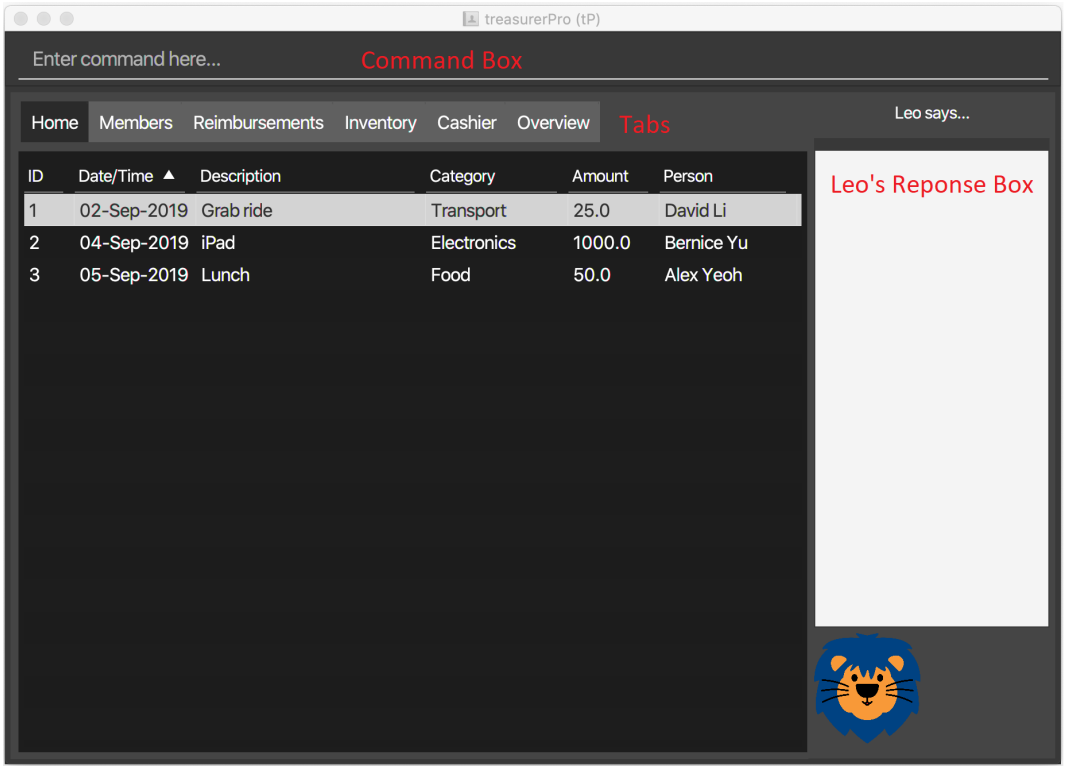


Figure 1. Graphical User Interface (GUI) of treasurerPro

2. Summary of contributions

My role was to design the layout and code the features for the `Cashier Tab`. The following sections presents the enhancements that I have implemented in greater detail, as well as the relevant documentation I have done in the user guide and developer guide.

2.1. Main Enhancements

The following illustrates some crucial enhancements that I have added for `Cashier Tab`.

- Adding, Deleting and Editing of Sales Items
  - What it does: It allows the user to add, delete and update the sales items added in the table shown on the tab.
  - Justification: These features incorporates the essential elements of a cash register and are fundamental for the `Cashier Tab`. By expediting the process of calculating and recording the

quantity of sales items, it enables the cashier-in-charge to perform cashiering duties more efficiently.

    - Highlights: The tab monitors the current stock in the inventory to ensure that the quantity keyed in does not exceed the stock available. This eliminates any hindrance involving manual checks for the availability of a certain item.
      Furthermore, upon any mistyped words or description inputs that does not match an existing item in the inventory, suggestions will be offered. This further augments the hassle-free aspect of this tab.

- Checking Out of the Cart
    - What it does: This checkout feature calculates and displays the total amount and the change to be returned.

- Justification: This feature allows the stock to be updated accordingly automatically on the `Inventory Tab` and abolish any need for manual logging of the stock.

- Highlights: This feature ensures accountability by prohibiting checkout if the cashier has not been set. Additionally, the sales made will be recorded and updated on the `Home Tab`. The sales made will also contribute towards the revenue and be used in the financial analysis.

2.2. Code contributed

The code that I wrote for my features can be found in the following links: Functional Code, Test Code

2.3. Other contributions

The following section leads to the relevant GitHub pull requests [PR] in relation to the specific contributions.

- Enhancements:
    - Wrote tests for several `Inventory` classes: (PR #148) [PR]
    - Refactored code to write to and from `Inventory` and `Transaction` package and update the respective models: (PR #202) [PR]
- Community:
    - Reviewed pull requests and offered suggestions (with non-trivial review comments): (PR #221) [PR] (PR #196) [PR]
    - Integrated `Cashier Tab` with other packages, fixed and added some `Inventory` classes: (PR #96) [PR] (PR #94) [PR]
    - Standardized decimal places for all amounts attributes in all packages for calculation and display: (PR #297) [PR]
- Documentation:
    - Updated the developer guide with diagrams and information about `Inventory` and `Cashier` parsers: (PR #209) [PR]

- Added implementation details for the `Cashier Tab`: (PR #209) [PR] (PR #164) [PR] (PR #162) [PR]
- Added guide to use `Cashier Tab` in user guide: (PR #209) [PR] (PR #212) [PR]
- Enhanced user guide to make it more user-friendly and updated `Members` Tab: (PR #303) [PR] (PR #252) [PR]
- Amended README document to make it more comprehensible with a better format: (PR #230) [PR]
- Updated glossary and FAQ questions: (PR #301) [PR]

3. Contributions to the User Guide

The following section illustrates my contribution to the treasurerPro User Guide for features specific to the `Cashier Tab`.

3.1. Current enhancement

{Start of First Extract from User Guide}

5.5.1. Add a Sales Item to the Table

This command enables you to add a sales item into the table.

- Command: `add [c/CATEGORY] d/DESCRIPTION q/QUANTITY`

The quantity that you input must be less than or equal to the stock available in the Inventory Tab. Else, Leo will display a message prompting input of a smaller quantity or another item.
The category field is optional. If you are unsure about the description of the desired item, you can refer to the Inventory Tab or simply key in the category without any other fields. Leo will display all the items in the specified category that are available for sale.

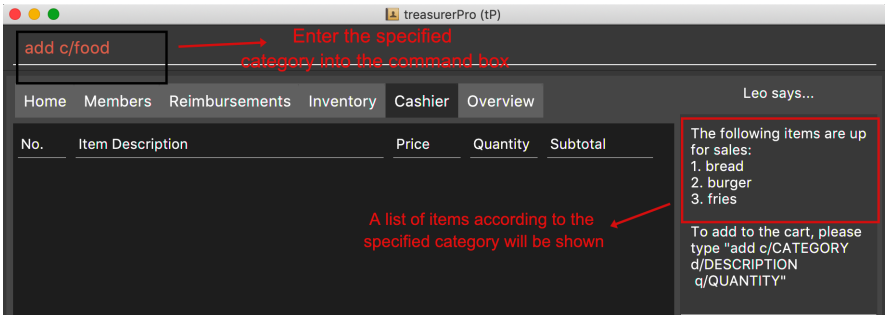
Additionally, if the description is misspelled or does not match any of the items in the inventory, Leo will recommend items with similar description that you might be looking for.

Examples:

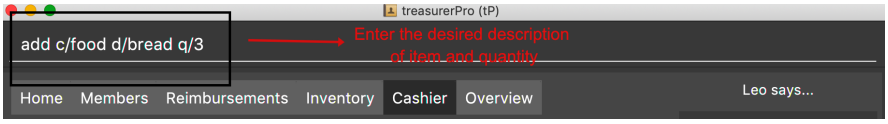
- `add c/food` - Displays all items that are under the 'food' category in the response box
- `add c/stationary d/pancake q/3` - Adds 3 similar items which have the description "pancake"
- `add d/pancake q/3` - Adds 3 similar items which have the description "pancake"

The system will allow a valid item to be added even if the category of the item does **not** match with the specified category inputted.

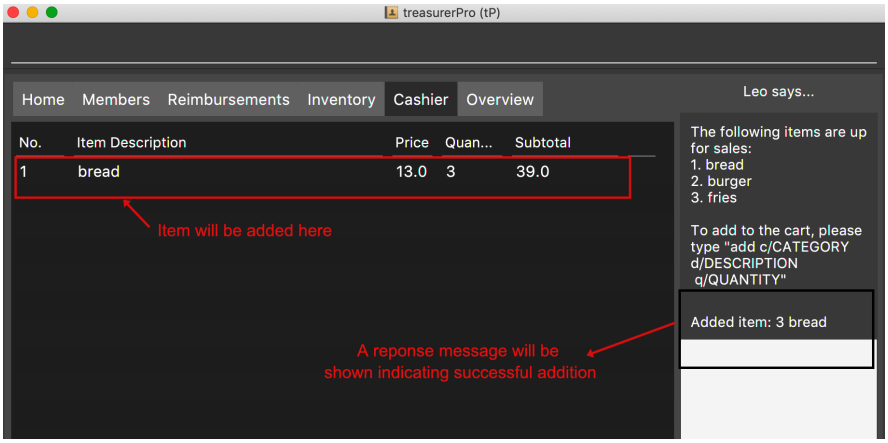
Steps: 1. Type the command with a category specified as shown in the screenshot below:



2. Enter the desired description and quantity according to the items suggested by Leo as shown below:



3. Hit `Enter`



Leo will respond to the successful addition with a response message. The newly added item will be shown on the table.

{End of First Extract from User Guide}

{Start of Second Extract from User Guide}

5.5.5 Checkout All Sales Items

This command enables you to perform a checkout of all the sales items in the table.

- Command: `checkout AMOUNT_PAID_BY_CUSTOMER`

The amount inputted should be the amount that the customer will be paying. This amount must be greater than or equal to the total amount listed on the bottom row of the table. If the amount paid is greater than the total amount, Leo will display the amount of change that the cashier should return.

After checking out, all items in the table will be cleared and the cashier will be reset.

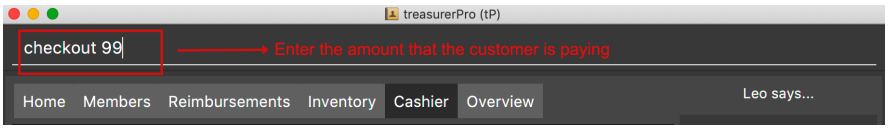
You must set the cashier before checking out. Else, checkout cannot proceed.

- Example:

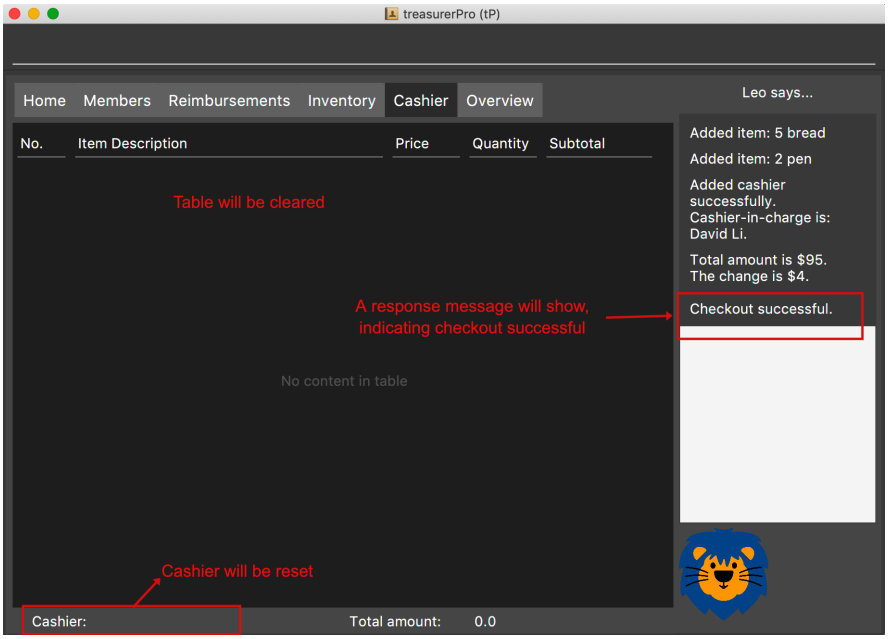
- `checkout 850` - Customer pays $850 to cashier

- Steps:

1. Type the command and specify the amount that the customer is paying as shown in the screenshot below:



2. Hit `Enter`



If checkout is successful, Leo will respond with a response message. The table will be cleared and the cashier will be reset.
The `Home Tab` will be updated with this transaction and the remaining stock in the `Inventory Tab` will also be updated.

{End of Second Extract from User Guide}

4. Contributions to the Developer Guide

The following section shows my contribution to the treasurerPro Developer Guide for features specific to the `Cashier Tab`.

4.1. Current enhancement

{Start of First Extract from Developer Guide}

3.1.1 Add Sales Item feature

This feature allows the addition of sales items to the cart.

Adding of a sales item to the cart will require an input of its description and quantity. An optional

field for category is provided to guide the cashier to find the desired item. If the category field is input with other unspecified description and quantity fields, `Model` will search all the sales items in the `Inventory List` according to the specified category and suggestions would be shown by Leo, the assistant.
If description and quantity field are both valid, the `ModelManager` will add the item into the sales list.

If the description inputted does not match any valid item, the `Model` will call the `getRecommendedItems(description)` method to show a list of suggestions.

The following is a code snippet from `getRecommendedItems(description)`:

```
if (description.length() >= 3) {
    char[] arr = description.toCharArray();
    ArrayList<String> combinations = getCombination(arr, arr.length);
    for (int j = 0; j < combinations.size(); j++) {
        if (combinations.get(j).contains(itemDescription) // itemDescription refers to
the actual description of an item in the inventory
                || itemDescription.contains(combinations.get(j))) {
            recommendedItems.add(item.getDescription());
            continue;
        }
    }
}
```

The `getCombination(arr, arr.length)` method in line 3 of the first snippet returns an ArrayList containing all subsets of descriptions that are of at least length 3. While iterating through the `InventoryList`, these subsets are compared with the actual descriptions of all items in the inventory to check if either contains the other.

The following sequence diagram shows how the `AddCommand` works which is referenced in 2.3. Logic component: Figure 5:

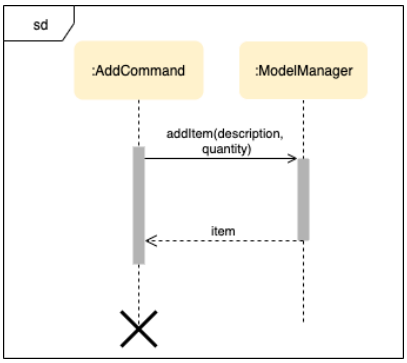


Figure 2. Sequence Diagram of Add Command in Cashier Tab (cashier package)

`AddCommandParser` will carry out multiple checks to check the validity of the inputs. `hasItemInInventory(description)` and `hasSufficientQuantityToAdd(description, quantity)` methods will be called to ensure the item has sufficient stock left in the inventory.
There will also be checks to ensure that the item specified is available for sale.

The following activity diagram shows the steps proceeding after the user input an add command:

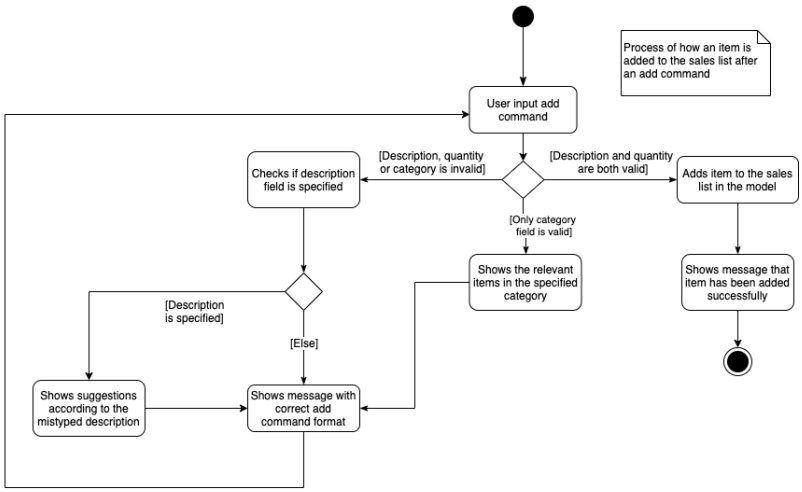


Figure 3. Activity Diagram of Add Command (cashier package)

{End of First Extract from Developer Guide}

{Start of Second Extract from Developer Guide}

3.1.2 Checkout Feature

This feature records all the sales items in the table as one sales transaction under the `Sales` category.

The `Home Tab` will be updated with the new transaction labelled as `Items sold`. The remaining stock of the sales items will also be updated on the `Inventory Tab`.

During the execution of the command, `getCashier()` method will be called which will return a person. This person will be used to create a `Transaction` object. If the cashier is null, the command cannot proceed and Leo will prompt the user to set a cashier.
If the amount inputted is valid and cashier has been set, the `ModelManager` will create a new transaction of the sales made.

The following sequence diagram shows how the checkout command is executed:

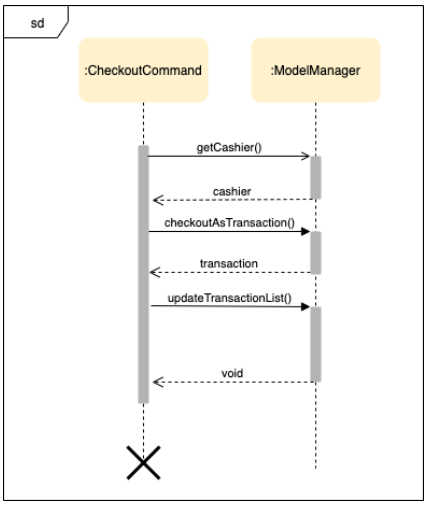


Figure 4. Sequence Diagram of Checkout Command (cashier package)

The `Cashier Logic` will call relevant methods to update the inventory list and newly-generated transaction to the respective `.txt file`.

To update the view on the `Inventory Tab` and `Transaction Tab`,`readInUpdatedList()` method of inventory model will called to read in the entire inventory data file and transaction will be added to the transaction model.

The following sequence diagram shows how the transaction and inventory are updated:

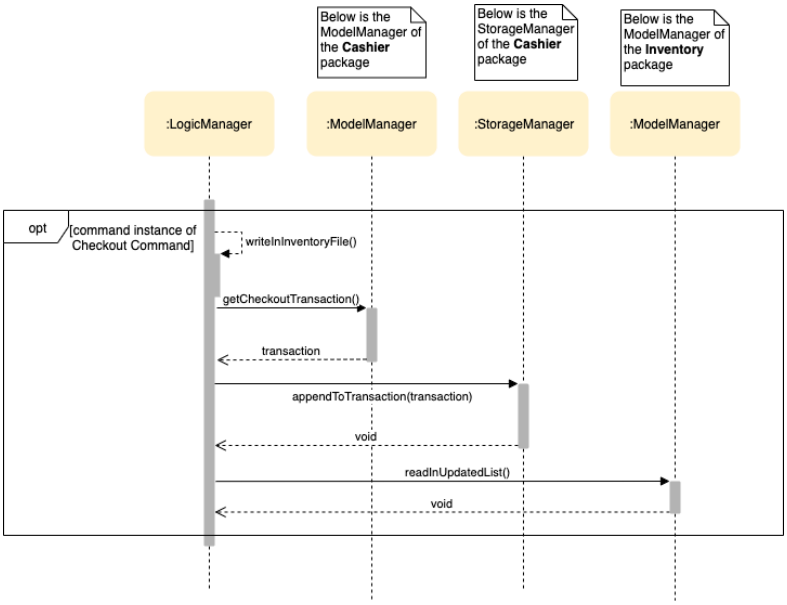


Figure 5. Sequence Diagram of how transaction and inventory get updated (cashier package)

As seen below, if the amount inputted is less than the total amount of items, the user will be prompted to key in a valid value.

The following activity diagram shows the steps after the user input a checkout command:

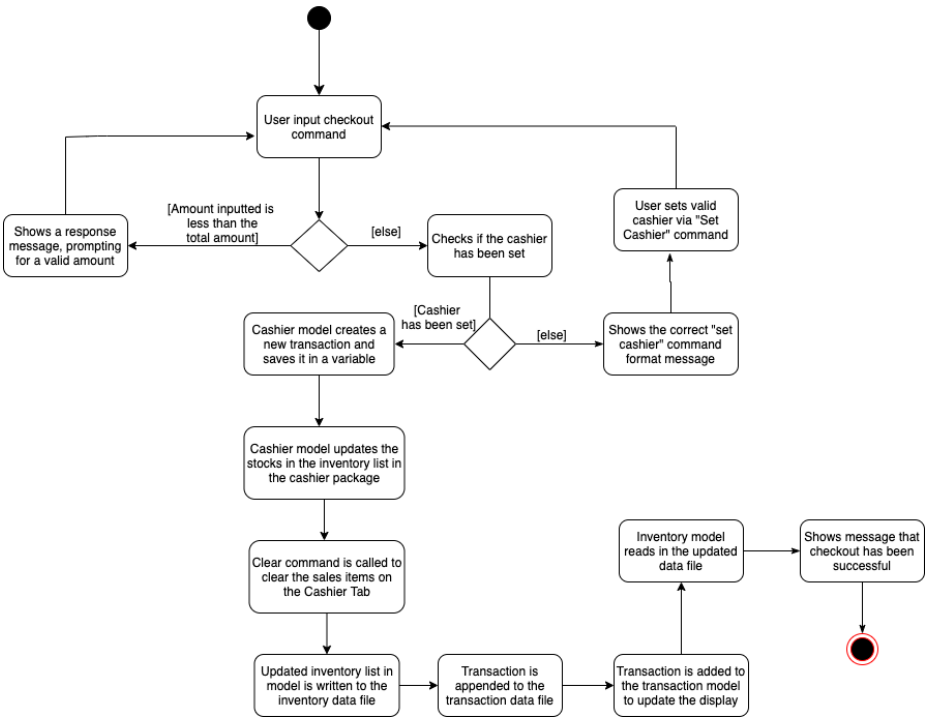


Figure 6. Activity Diagram of Checkout Command (cashier package)

3.1.5 Overall Design Considerations

This section explains the design considerations for some crucial implementations in the Cashier

Tab.

Alternative 1	Alternative 2	Conclusion and Explanation
An ArrayList is used to store the list of sales item shown on the `Cashier Tab`. **Pros**: Elements are be accessed directly more efficiently. **Cons**: Adding and removing from the head of the list is less time-efficient for ArrayList.	A LinkedList is used to store the list of sales item. **Pros**: Time performance is better when elements are accessed from the head of the list. **Cons**: Performance is poor when accessing directly via the index.	Alternative 1 is selected. An ArrayList has better performance with respect to time when accessing each elements directly. As the sales list will be updated and accessed regularly, an ArrayList is more fitting.
The Transaction, Inventory and Person `Model` interfaces are passed as parameters into Cashier's `Logic` to call relevant methods to update the inventory and transactions. **Pros**: Cashier's Logic can access all public methods in the respective `Model`. **Cons**: It might result in unintended modification to some of the data in the Models.	Interfaces that only contains required methods are created. The methods are called via these interfaces to update the data. **Pros**: Only necessary methods will be accessed, preventing any unwanted changes through other methods. This follows the Facade Pattern. **Cons**: If more methods are needed, they need to be added to these interfaces.	Alternative 2 was implemented as only a few methods are required from each `Model`, so the new interfaces can act as facades and restrict access to all public methods in the models. This prevents in Cashier's `Logic` from causing any unintended modification to any of the data in the Models.
The Cashier Storage directly writes to and from the data file of the inventory and transaction. **Pros**: It can access the data file directly without any dependencies. **Cons**: The data files can be modified from 2 sources, introducing more chances of bugs.	The Cashier Storage accesses the methods from the Transaction and Inventory storage via their `Logic` to update the data. **Pros**: The data files are only modified from 1 source, ensuring cohesiveness in the format of data stored. **Cons**: It introduces more dependencies on the storage of other packages.	Alternative 2 is implemented to enforce defensive programming, so that the data files are not modified via 2 different methods and eliminate any chances of uncoordinated data in the data files.

{End of Second Extract from Developer Guide}

[1] command line interface (CLI) is a text-based user interface (UI) that allows the user to interact with the system using commands