Shamus Neo Zhi Kai - Project Portfolio for OwlMoney

Hello there, I am *Shamus*. I am currently pursuing a degree in Information Security at the National University of Singapore (NUS).
I enjoy server and network administration as well as building and breaking applications.

This portfolio serves to document my extensive involvement and contribution in a team-based project CS2113T (Software Engineering & Object-Oriented Programming) module by NUS. This project spans over a period of about eight weeks, and was completed together with a team of five members (consisting of Mong Zheng Wei, Brian Tan Kian Ming, Terence Tan Wee Theng, Valerie Tan Yi Jia and myself).

About the project

My team and I were tasked with enhancing a basic Command Line Interface (CLI) based personal assistant, named Duke for our Software Engineering Project. We chose to morph it into a financial tracker and management application called **OwlMoney**. This enhanced application targets undergraduates and fresh graduates and aims to help them track and manage their finances. Some features of **OwlMoney** include tracking of both expenses and expenditures as well as tracking credit cards and investments such as investment bonds.
My role as a team leader was to perform project management as well as to write the baseline code for the team to get started. On top of that, I was also in charge of writing codes for the `investment`, `bonds`, `storage` and some parts of the `commands` section.
The following sections illustrates these enhancements in more details, illustrates the relevant sections I have contributed to the project as a whole, as well as the relevant sections I have added to the user and developer guides in relation to these enhancements.

Summary of Contributions

This section shows a summary of my coding, documentation, and other helpful contributions to the team project.

Enhancement added: I added the ability to add `investment` accounts and `bonds`.

- **What it does:** The `investment` account allows the user to create an account specially dedicated for `investment` purposes.
- **Justification:** As we are targeting undergraduates and fresh graduates who usually do not have good financial literacy, I believe that **OwlMoney** can play a role in educating them on separating spending from `investment`. Therefore, no spending transactions can be made on the `investment` account and no investments can be bought in a savings account.
- **Highlights:** This enhancement automatically credits interest for `bonds` added into the `investment` account every **6** months from the date of purchase, eliminating the need for the user to manually calculate and enter the interest in every half-yearly. This prevents errors caused by users forgetting to enter the interest since the interest only credits twice a year.

- **Credits:** A portion of recording `transactions` and similar features across all bank accounts were inherited from a common parent class `bank` which was coded by both Brian Tan Kian Ming and I.

Code contributed:

- [Commits] [Pull Requests] [RepoSense Code Contribution Dashboard]

Other contributions:

- **Project Management:**
  - There were a total of 4 releases, from version 1.0 to version 1.3. I managed the team's repository and was responsible for producing all of the releases on Github.
- **Enhancements to existing features:**
  - Updated the persistent storage to use `OpenCSV` instead of writing into a conventional `.txt` file. This provides better clarity and organisation of stored data in secondary storage.
- **Documentation:**
  - Responsible for maintaining **over 70%** of content on both the **User Guide** and **Developer Guide** as well as making it presentable in the `.adoc` format which can be rendered and converted to various other formats since it is platform independent.
- **Community:**
  - Contributed to **forum discussions** (examples: PRs #61, #60, #57, #26, #8)
  - Reported bugs (examples: PRs #11)
- **Tools:**
  - Integrated a third party library OpenCSV to the project for persistent `storage`.
  - Integrated Travis & AppVeyor **Continuous Integration (CI)** tools. (examples: PRs #119)
  - Integrated Coveralls & Codecov **Code coverage** tools. (examples: PRs #133)
  - Integrated Codacy **Code analytics** tool.
  - Integrated Netlify & GitHub Deployment **Deployment preview** tool to the team's repository (examples: PRs #119, #101, #103)
  - Updated **Gradle build configuration** to automate and simplify the build process (examples: PRs #101)
  - Added a new **Github plugin** (project-bot) to automate triaging of issues in the team's Project Board

Contributions to User Guide

The following sections illustrate my ability in writing documentation targeting end-users to guide them in using the various features of the application.

Adding investment bank account `/add /investment`

Want to start **investing** to grow your wealth? No problem! All you need to do is to add an investment account!
Here's how you can use the `/add /investment` command.

Command Syntax

`/add /investment /name ACCOUNT_NAME /amount AMOUNT`

> **WARNING** There can only be a maximum of **3** investment accounts.

Example

- `/add /investment /name DBB Vickers Account /amount 10000`

Adds an investment account named `DBB Vickers Account` which has an initial amount of `$10000` inside that you can start investing with.

Editing investment bank account `/edit /investment`

Changes made to your investment account? Here is how you can do it!

Command Syntax

`/edit /investment /name ACCOUNT_NAME [/newname ACCOUNT_NAME] [/amount AMOUNT]`

> **WARNING** At least one of `/newname`, `/amount` must be used.

Example

- `/edit /investment /name DBB Vickers Account /newname OBOB Securities Account`

Edits the name of the account from `DBB Vickers Account` to `OBOB Securities Account`.

- `/edit /investment /name DBB Vickers Account /amount 50000`

Edits the amount in `DBB Vickers Account` to `$50000`.

- `/edit /investment /name DBB Vickers Account /newname OBOB Securities Account /amount 50000`

Edits the name of the account from `DBB Vickers Account` to `OBOB Securities Account` and the amount in the account to `$50000`.

Deleting investment bank Account `/delete /investment`

You can also close your investment bank account on `OwlMoney` as well!

Command Syntax

`/delete /investment /name ACCOUNT_NAME`

WARNING	All transactions (e.g. bonds) related to the investment bank account will be deleted.

Example

- `/delete /investment /name DBB Vickers Account`

Deletes an investment account named `DBB Vickers Account`.

Adding investment bonds `/add /bonds`

Signed up for a bond? Finding it difficult to keep up with the interest? No worries! **OwlMoney** allows efficient tracking of your semi-annual coupon interest!

Command Syntax

`/add /bonds /from ACCOUNT_NAME /name BOND_NAME /amount AMOUNT /rate BOND_RATE /date DATE /year YEARS`

WARNING	An investment account needs to be created first to add bonds. Bonds can only be added to investment accounts.

Example

- `/add /bonds /from DBB Vickers Account /name June SSB /amount 1000 /rate 1.92 /date 1/1/2019 /year 1`

Adds a bond named `June SSB` charged to `DBB Vickers Account` at $`1000` with an interest rate of `1.92`% bought on `1/1/2019` for `1` year(s).

Editing Bond details `/edit /bonds`

Change in your investment details? Edit them here!

Command Syntax `/edit /bonds /from ACCOUNT_NAME /name BOND_NAME [/rate BOND_RATE] [/year YEARS]`

WARNING	At least one of `/rate` or `/year` must be present when editing. Only `/rate` and `/year` can be edited. Editing `/rate` will only result in future bond coupon interest crediting to be modified, all past interest credited will not be edited. `/year` can only be edited to a year higher than the original year.

Example

- `/edit /bonds /from DBB Vickers Account /name June SSB /rate 1.98`

Edits the bond named `June SSB` charged to `DBB Vickers Account` with a new interest rate of `1.98`%.

Deleting bonds `/delete /bonds`

Sold your bonds? Delete it from **OwlMoney**!

Command Syntax `/delete /bonds /from ACCOUNT_NAME /name BOND_NAME`

Example

- `/delete /bonds /from DBB Vickers Account /name June SSB`

Deletes the bond named `June SSB` charged to `DBB Vickers Account`.

Contributions to Developer Guide

The following sections illustrate my ability in writing documentation to provide developers insights on the design of the application. It also showcase the technical depth of my contributions to the project.

Editing Bonds

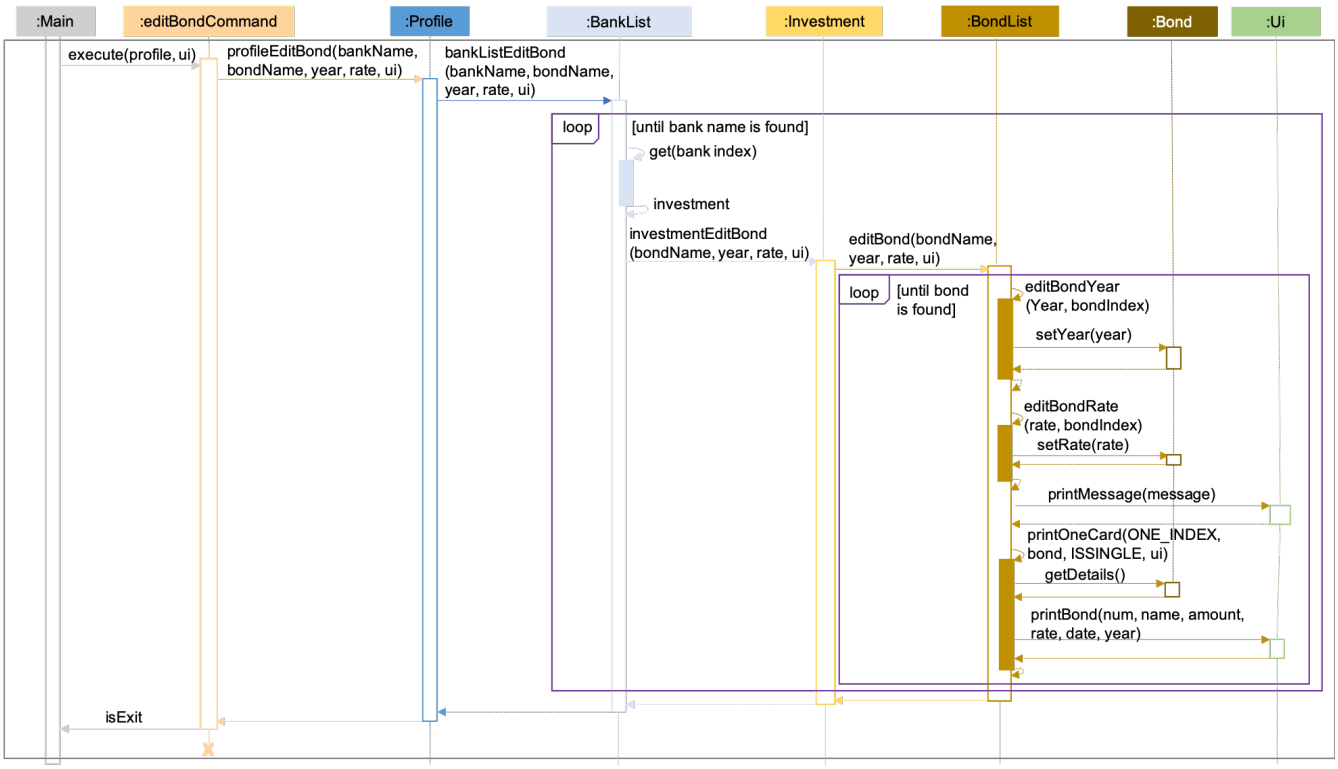

Figure 1. Sequence Diagram of Editing Bonds

NOTE	The sequence diagram presented above is assumed to be a valid command which will produce a successful result.

The sequence diagram presented above depicts the interaction between the `Logic` and `Model` component for running `EditBondCommand`.

The `EditBondCommand` requires a minimum of **3** and up to a maximum of **4** inputs:

1. Investment Account's name
2. Bond's name
3. **At least 1** of the **2** inputs:
    a. Rate
    b. Year of maturity

When the user executes the `EditBondCommand`, the following steps are taken by the application:

1. When `EditBondCommand` is executed, it will invoke `profileEditBond`.
2. Within the invocation of `profileEditBond`, a method named `bankListEditBond` will be invoked.
3. Once invoked, `bankListEditBond` will perform the following checks based on the bank name specified:
    - Check for the existence of the investment account containing the bond.

NOTE	`bankListEditBond` will throw an error if the above check fails.

4. After passing the above checks, the method `investmentEditBond` will be invoked.
5. Within `investmentEditBond`, the method named `editBond` will be invoked.
6. Once invoked, `editBond` will perform the following checks:
    - Check for the existence of the bond within the investment account.
    - Check whether the newly specified year of maturity for the bond is more than or equal to the current year of maturity through the method `editBondYear`.

NOTE	`editBond` will throw an error if the above check fails.

7. After passing all of the above checks, `editBond` will update the bond details with the new details specified using:
    - `editBondRate` → edits bond's interest or coupon rate.
    - `editBondYear` → edits year of maturity.
8. Once the bond object has been edited, the updated details of that bond object will be displayed to inform the user of the **successful** editing of the bond.