Muhammad Irham Rasyidi Bin Zainal - Project Portfolio

PROJECT: Loanbook

Overview

Link to GitHub Repo: https://github.com/CS2103-AY1819S1-F10-2/main

Loanbook is a desktop application targeted towards bicycle shop owners who are proficient with interacting with a command-line interface (CLI). This application allows them to manage their bicycle loaning business, from the loaning and returning process to bicycle management.

The user interacts with the application using a CLI and it has a GUI created with JavaFX. The code base has approximately 20,000 lines of code.

This project was morphed from an existing project, Addressbook Level 4.

Summary of contributions

- **Major enhancement**: added **the assignment of unique Loan IDs to Loans upon addition**
  - What it does: When the user adds a new Loan, the app will automatically assign a unique Loan ID to that Loan.
  - Justification: When performing a bicycle Loan, it is convenient to be able to have a way to uniquely identify that transaction. For example, the customer (who rented out the bicycle) can have the Loan ID printed on the receipt (or e-receipt). This will smoothen the returning process since the shop owner can identify the Loan to return easily.
  - Highlights: The implementation of this feature was fairly troublesome, mostly due to the existing tests, but also due to the existing implementation of the `add` command (which is the command that assigns new Loan IDs to Loans).
    - In the original implementation of the `add` command, the `AddCommandParser` produces an `AddCommand` that already contains the `Loan` to add into the Loanbook. This could be done in the Addressbook since all the relevant fields of

a Loan can be extracted from the typed command. However, with the LoanID, the actual Loan cannot be fully formed until the `execute()` method of `AddCommand`, which has access to the Model.

    - Many of the existng tests which failed were troublesome to fix since the expected loans to be added now required knowledge on the current state of the Model. The edit command is also another culprit since Loan ID is not an editable field, meaning when trying to edit an existing Loan to a specified Loan, the ID of the actual loan may not be able to match the expected loan.

- **Minor enhancement**: added a `resetall` command that reset the entire Loanbook.
- **Minor enhancement**: added password authentication to `resetloans` and `resetall` commands.
- **Code contributed**: [Reposense code]
- **Other contributions**:
    - Project management:
    - Enhancements to existing features:
        - Refactored the `LoanTime` class to extend the `DataField` class and make time-related operations simpler. (Pull request #73)
    - Community:
        - PRs reviewed (with non-trivial review comments): #251, #253
        - Some parts of the Loan ID feature I added was adopted by other class mates (#253)

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.*

Hard reset all loans from loan book : `resetloans`

Removes all loans from the loan book and resets the Loan ID counter. This operation requires password authentication.

Format: `resetloans x/CURRENT_PASSWORD`

List of Parameters:

`x/CURRENT_PASSWORD` : Password used in the LoanBook.

Example:

- `resetloans x/a12345`

This operation will not modify the bicycles in the Loan Book. To reset the entire loan book, including the bicycles, see the `resetall` command.

This operation will erase the data of ALL loans! Do this at your own peril.

Hard reset the entire loan book : `resetall`

Resets the entire loan book. This includes the removal of all loans and bikes from the loan book and the Loan ID counter being reset. This operation requires password authentication.

Format: `resetall x/CURRENT_PASSWORD`

List of Parameters:

`x/CURRENT_PASSWORD` : Password used in the LoanBook.

Example:

- `resetall x/a12345`

This operation will erase the data of ALL loans and bikes! Do this at your own peril.

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.

Unique Loan ID

To keep individual loan transactions identifiable, we have a Loan ID for every loan. This will be automatically assigned when the user creates a Loan using the `add` command, and will be unique among all Loans in the Loanbook.

Current Implementation

In the current implementation, the Loan IDs are represented using integers. The added loans will be assigned IDs in increasing order starting from 0 (i.e. the first loan will be assigned ID 0, the second loan will be assigned ID 1, the third loan, ID 2, and so on).

> Since this value will be a represented by a Java `int`, the maximum possible ID has been set to be 999,999,999 to prevent any integer overflow. The user will not be allowed to add a Loan using the `add` command after that maximum ID has been assigned.

The running ID mechanism is facilitated by the `LoanIdManager` class, which is a component within the `LoanBook` class. After initialization using the last used Loan ID value, it stores a running integer value which increments whenever the next Loan ID is generated using `getNextAvailableLoanId()`.

Since the `LoanIdManager` is a component within the `LoanBook` class, this makes it possible to capture the internal state of the `LoanIdManager` at any point for future use. It allows for simple integration with the existing undo/redo mechanism, and can be stored within the same XML file that contains all other Loanbook details.

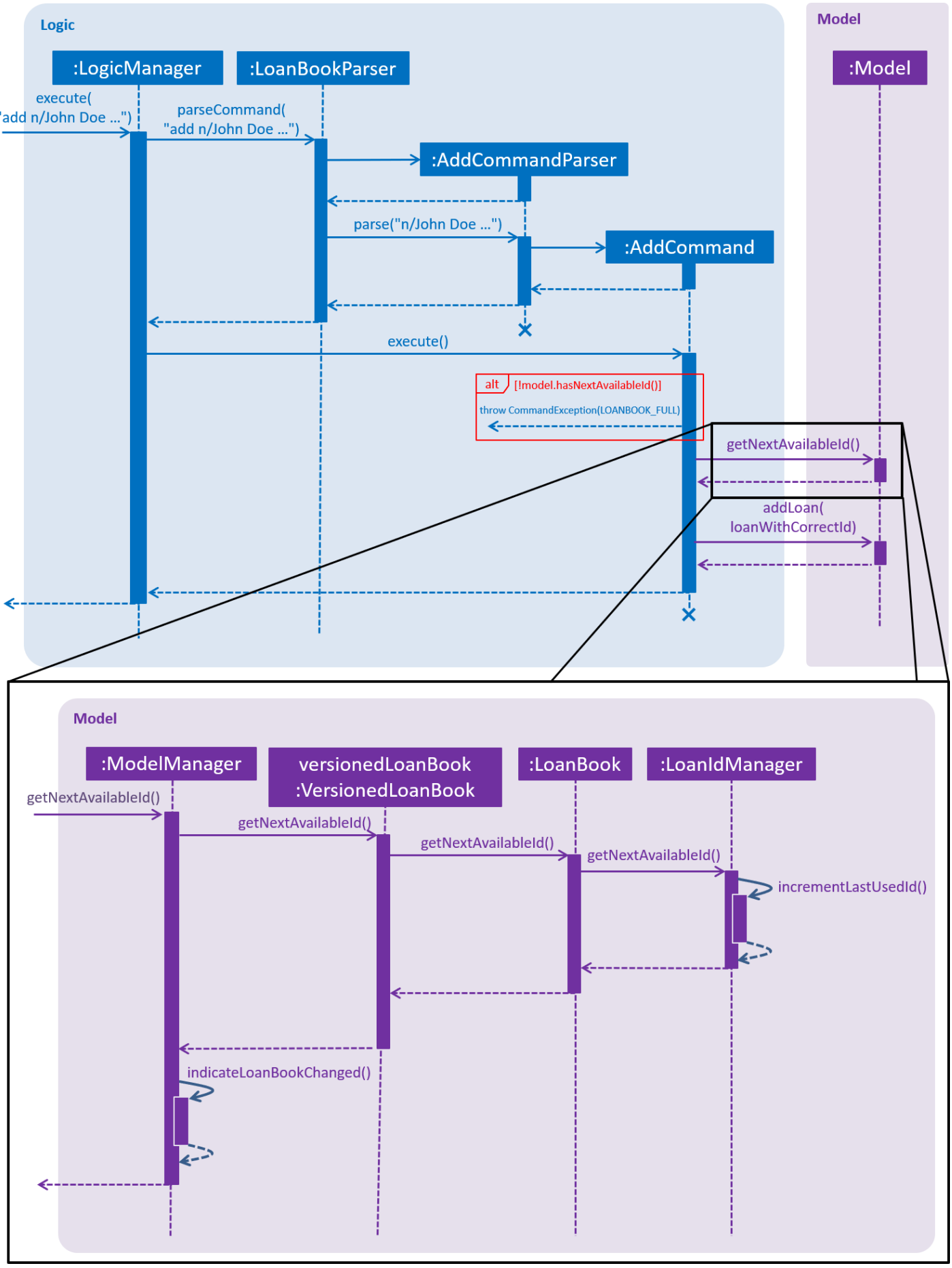
The following steps are taken when the user adds a new Loan using the `add` command:

Step 1: After typing an `add` command, the `AddCommandParser` will parse the user input and construct a new `Loan` instance with a placeholder Loan ID, which is then used to construct a new `AddCommand` instance.

Step 2: When `AddCommand` gets executed, the next available Loan ID will be generated via the `Model` interface. This will increment the internal running ID within the `LoanIdManager` of the `VersionedLoanBook`.

Step 3: A new `Loan` instance will be generated using the generated Loan ID value. This new `Loan` instance gets inserted into the `VersionedLoanBook` and gets committed.

The following sequence diagram shows how the `add` command functions, including the retrieval of the next Loan ID from the Model:



Design Considerations

Aspect: Using an Integer for the Loan ID

- **Alternative**: Use a `String` to represent the Loan ID
  - Pros: More Loan ID values can be created given the same number of characters.
    a. There would be no technical limit to the number of Loan IDs possible. If all Loan IDs of a certain length get exhausted, the next generated Loan ID can be one character longer.
  - Cons:
    a. Generating the next available Loan ID may be more complicated for a string as compared to an integer (though not by a significant amount).
  - **Possible Implementation**: A running counter, starting from 0, that resets at the start of every day. The Loan ID is formed by taking the current date, in "YYYYMMDD" format, and appending it with said running counter.
    - Pros: Simple to implement.
    - Cons: Depending on the end-user, the fact that the Loan ID displays the number of loans created during that day may be undesirable.
  - **Possible implementation**: Generate a string with a fixed number of characters.
    - Pros: The ID does not allow the number of loans created to be determined. This may be useful if this behaviour is desirable by the end-user.
    - Cons:
      a. The model needs to keep track of all the IDs that have been used before to prevent duplicates.
      b. The ID may be troublesome to type in the command-line, especially if the Loan ID is used for other commands.

Aspect: Having a LoanIdManager class simply to keep track of a running counter

- **Alternative**: Have a static integer variable in the `Loan` or the `VersionedLoanBook` class
  - Pros:
    a. The process of incrementing the running value is simpler and has lesser steps than with a `LoanIdManager` class.
  - Cons:
    a. It is more complicated to capture the state of the running value. The undo/redo mechanism may not work with a static variable.

Last updated 2018-11-12 21:34:21 SGT