Ho Hol Yin - Project Portfolio

PROJECT: LiBerry

Overview

This portfolio aims to highlight my contributions to our team project, 'LiBerry'.

'LiBerry' is a library management system that is designed for small communities with a lack of expertise and resources to manage a library.

This library management system is able to **manage books** in the library, **register** borrowers, **loan** and **return** books, as well as **calculate loan periods** and **fines** for any overdue loans.

I was in charge of implementing the **Generate loan slip** feature and the **Book** feature. Additionally, I am responsible for the **code quality and testability** of the team's code base throughout the entire development phase.

In this portfolio, I will illustrate my contributions in documenting the user guide and developer guide. These documentations serve to aid librarians and developers in understanding the features of the software.

Legend

I have used symbols to give a visual explanation to certain parts of this portfolio. This section aims to explain the meaning behind the usage of these symbols.

Denotes useful tips.

Denotes additional information.

Summary of contributions

This section shows a summary of my contributions to the team project and will include the main enhancement (Generating Loan Slip), code contribution and other contributions.

Major enhancement - Generate Loan Slip feature

This feature allows librarians to **automatically generate loan slips in Portable Document Format (PDF)** for every loan. I will explain in detail below what the feature is about and why it is an important feature.

What the feature does

This feature **generates loan slips** so that the librarian can immediately print the loan slip for the borrower. The printed loan slip contains all information about the loan or renew, including a list of loaned books and the due date of these loans.

Justification

Since our target group is small impoverished communities, it is unlikely they will have reliable internet access to check the status of their loans. Therefore, it will be suitable for librarians to provide them with a printed version of the loan.

Highlights

This feature is challenging to implement as it will require us to create a new PDF document from scratch. It was required of me to understand a great deal about file input and output in Java, which was something out of the scope of the course. Additionally, this document would have to be suitably styled to display a certain level of professionalism.

Why this is a major feature

This feature represents most of the core features of 'LiBerry' in a condensed PDF. I have provided a breakdown below as to how this feature achieves the required **depth**, **completeness** and **level of effort**.

- **Depth**: This feature is deeply linked with the Book, Loan and Borrower features as it has to display information relevant to these features.
- **Completeness**: This feature is complete with the intended style has been rigorously tested in many test cases.
- **Effort**: This feature required an in-depth understanding of handling external libraries as well as file input and output. These information are not taught in the course and therefore required extensive research on my part.

Credits

This feature is made possible with the 'iText 7' library. However, it was still required of me to understand both the Java `File` class and the API well such that I can use it appropriately in the context of our project.

Code Contributed

The following link shows the analysis of my code contributions:

- RepoSense link for Code Contributions

Other contributions:

Project management:

- Managed the release of `v1.3` on GitHub.
- Continually added test cases throughout development phase to ensure consistent code quality. Done through the following Pull Requests:
  - [#31]: Refactored `AddressBook3` tests to `LiBerry` tests
  - [#206]: Increased test coverage for `UI` components
  - [#260]: Increased test coverage for `Delete` and `Book` classes

Enhancements to existing features:

These enhancements are done through the following Pull Requests:

- [#176]: Added a new `LoanHistory` object in `Book` for to be displayed in the info window
- [#177]: Created a borrower panel in the Graphical User Interface to list books that the borrower has currently loaned
- [#187]: Upgraded `LoanSlipUtil` to allow it to mount multiple loans into a single loan slip

Documentation:

The documentations are updated through the following Pull Requests:

- [#148]: Illustrated the `Model` component of the system and its behaviour through class and sequence diagrams
- [#159]: Documented the feature to generate loan slip and its design considerations
- [#162]: Amended Developer Guide based on feedback from tutor

Community:

- Reviewed a Pull Request from another group regarding their user stories and UML diagrams
- Contributed a tip in the forum (as a group) on how to check code coverage when running tests
- Reported bugs and suggestions for other teams in the class.

Tools:

This tool was added through the following Pull Request:

- [#150]: Integrated a third party library (iText 7) to the project.

Contributions to the User Guide

This section outlines my contributions to the team's User Guide. It demonstrates my ability to write documentation for librarians to understand how to use the software. Please refer to the User Guide for the full documentation.

Book feature

Adding a book: `add`

Adds a new book to library records.

Format: `add t/TITLE a/AUTHOR [sn/BOOK_SN] [g/GENRE]…`

- Adds a book with the title `TITLE`, written by `AUTHOR`, classified by the genres `GENRE` and tagged with the serial number `BOOK_SN`.
- `TITLE` should be at most 50 characters long.
- `AUTHOR` should be at most 50 characters long.
- `GENRE`, if provided, should be at most 20 characters long.
- `BOOK_SN`, if provided, must be a valid serial number that starts with the prefix 'B' followed by 5 digits. They should be unique.

A book can have up to 5 genres (but can have no genres as well).

You do not need to specify the serial number if you wish so.
LiBerry will then auto-generate a valid serial number for the new book.

Examples:

- `add t/Harry Botter a/Raylei Jolking sn/B02010 g/children`
  Adds a children book titled "Harry Botter" by "Raylei Jolking", with the serial number "B02010", to LiBerry.
- `add t/Inferno a/Tande g/classic g/epic` Adds a book titled "Inferno" by "Tande", with the genres "classic" and "epic" to LiBerry. The serial number for this book will be automatically generated.

Deleting a book : `delete`

Deletes a book from the library records. Used when book is lost or trashed.

Format: `delete INDEX` or `delete sn/BOOK_SN`

- Deletes the book at the specified `INDEX`.
- `INDEX` refers to the index number shown in the displayed book list.
- `INDEX` **must be a positive integer** 1, 2, 3, …
- `delete INDEX` will delete the book with the book at `INDEX` position in the results list.
- `delete sn/BOOK_SN` will delete the book with this serial number.

Examples:

- `find t/harry`
  `delete 1`
  Deletes the 1st book in the results of the `find` command.
- `delete sn/B00422`
  Deletes the book with serial number `sn/B00422`.

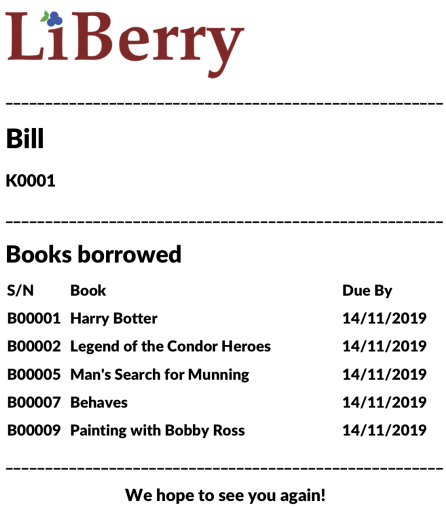
Generating a loan slip

Exiting Serve Mode: `done`

Exits Serve Mode.

Format: `done`

After loaning all books, upon the `done` command, a printable loan slip in pdf format will be generated. The loan slip will be opened in your computer's pdf viewer and also saved in the `loan_slips` folder. The figure below shows an example of how a loan slip might look like.


LiBerry

Bill

K0001

Books borrowed

S/N	Book	Due By
B00001	Harry Botter	14/11/2019
B00002	Legend of the Condor Heroes	14/11/2019
B00005	Man's Search for Munning	14/11/2019
B00007	Behaves	14/11/2019
B00009	Painting with Bobby Ross	14/11/2019

We hope to see you again!

Figure 1. Printable loan slip generated.

In the figure above, we can see that the loan slip records all the books borrowed by 'Bill'.

Contributions to the Developer Guide

This section highlights my contributions to the Developer Guide. It demonstrate my ability to aid other developers in understanding the various implementations of the features. Please refer to the Developer Guide for the full documentation.

Implementation of Book feature

Details of Implementation

The add book function is facilitated by `Catalog`. The `Catalog` stores a list of books, representing the books in the library. Additionally, it implements the following operation:

- `Catalog#addBook(book)` — Add a new book to the list of books in the catalog.

Given below is an activity diagram of a book being added to the catalog.

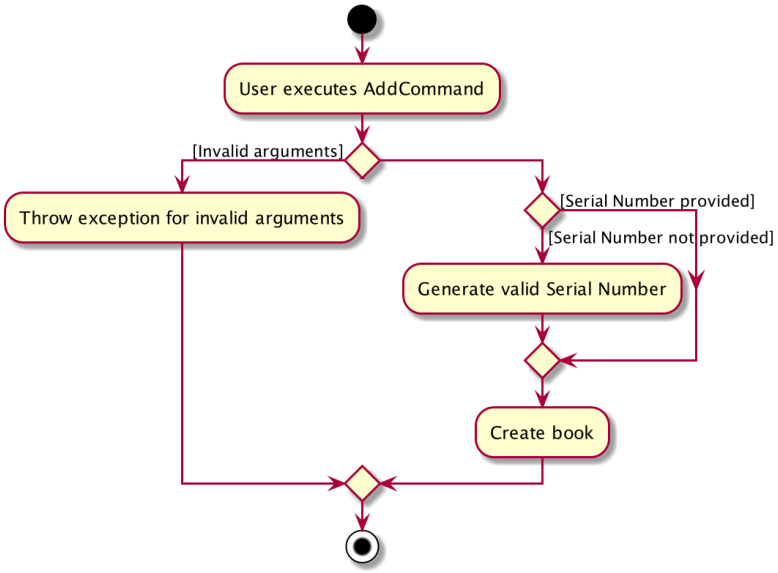


Figure 2. Activity Diagram for adding a book

The else branch of each branch node should have a guard condition `[else]` but due to a limitation of PlantUML, they are not shown.

We can clearly see how the system decides to generate a valid serial number base on whether the user input contains a valid serial number or not.

After the book is added to the system, we can now represent it with a class diagram shown below.

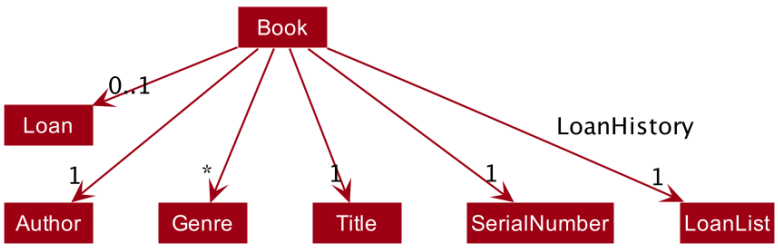


Figure 3. Class Diagram for `Book`

Notice how the book can hold either **1 or 0 loans**, depending on whether it is currently loaned out or not.

The current state of this newly-added book is further illustrated by the object diagram below.

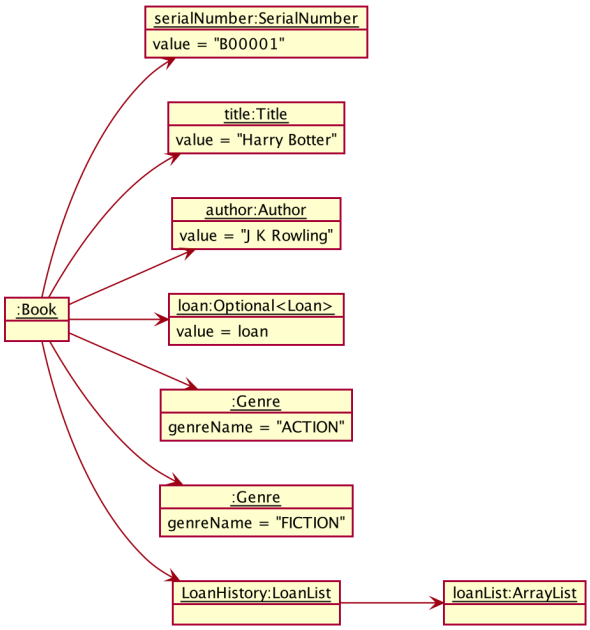


Figure 4. Object Diagram for `Book`

We can see that the book holds an `Optional<Loan>` and has an empty `LoanHistory`, making it consistent with the class diagram of `Book` above.

Design Considerations

Aspect: Data structure to store books.

- **Alternative 1 :** Store them only in a ObservableList as per the original AddressBook implementation.
  - Pros: Will be easy to implement.
  - Cons: Iterating through the list of books to retrieve one may be inefficient.
- **Alternative 2 (current choice):** Store them in a HashMap.
  - Pros: Will be easier (and more readable) to retrieve books by serial number.
  - Cons: Will incur additional memory to maintain the HashMap.

We have decided to go with Alternative 2. There is a lot of retrieval of book objects within the Book and Loan features. Therefore, the benefits of quick retrieval of book will outweigh the additional memory costs incurred to maintain the HashMap.

Aspect: Generating a unique serial number.

Since we allow librarians to provide their own valid serial number when adding a book, we cannot use the number of books or the largest serial number to generate the next serial number.

- **Alternative 1:** Use a TreeMap to store current serial numbers.
  - Pros: Will be efficient in generating the next valid serial number.

- Cons: Will incur additional memory to maintain the TreeMap. Might also result in unexpected behaviour in some edge cases.

- **Alternative 2 (Current choice):** Iterate from the beginning to obtain the first unused serial number.
  - Pros: Will be easy to implement.
  - Cons: Will be inefficient once the number of books grow.

We have decided to go with Alternative 2 and keep it simple. This is because there are some cases which leads to unexpected behaviour from Alternative 1. Furthermore, Alternative 2 is in line with the **KISS** (Keep it Simple, Stupid) principle of programming.

Implementation of Generate Loan Slip feature

Details of Implementation

The printing of loan slip feature is facilitated by `LoanSlipUtil`. Essentially, `LoanSlipUtil` implements the following operations:

- `LoanSlipUtil#mountLoan()` — Mounts a loan in the current loan session.
- `LoanSlipUtil#clearSession()` — Clears the loan session by unmounting all loans.
- `LoanSlipUtil#createLoanSlipInDirectory()` — Creates a pdf version of the mounted loans as a single loan slip, saved in the loan_slips folder.

Given below is the sequence diagram of the generation of loan slip during the loan of a book.

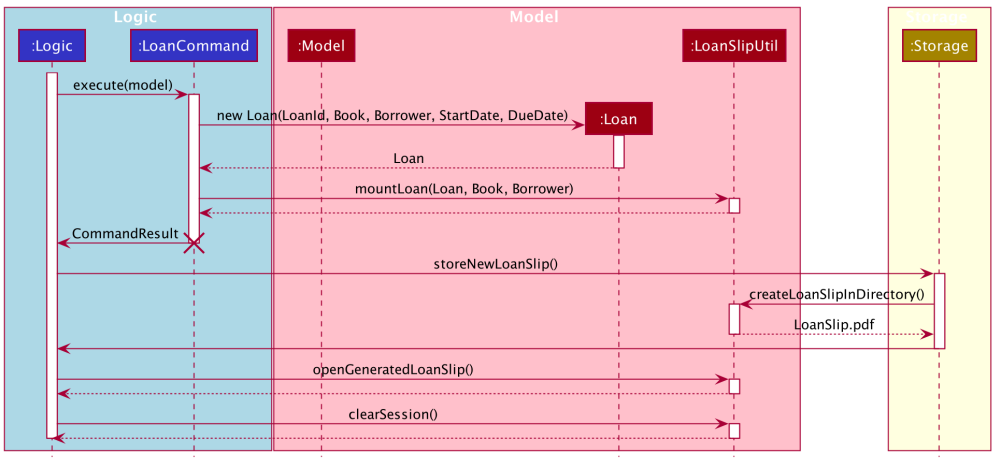


Figure 5. Sequence Diagram for the generation of a loan slip

The sequence diagram above is described by the following sequence of events:

1. `LoanCommand` is executed
2. `LoanCommand` retrieves the `Book` and the `Borrower` to create a new `Loan`

3. `LoanCommand` mounts the new loan in `LoanSlipUtil`
4. `Storage` component creates and saves a new PDF in a saved folder
5. `Logic` component opens the newly generated `LoanSlipDocument`
6. `Logic` component clears the session in `LoanSlipUtil`

Design Considerations

Aspect: How to create and use an instance of a `LoanSlipDocument`.

- **Alternative 1 :** Use the `LoanSlipDocument` constructor directly.
  - Pros: Will be straightforward to implement.
  - Cons: The `Logic` component and the `LoanCommand` object needs to know all the methods of `LoanSlipDocument` to be able to create a loan slip.
- **Alternative 2 (current choice):** Create a Facade class `LoanSlipUtil` to facilitate creation of `LoanSlipDocument`.
  - Pros: The `Logic` component and the `LoanCommand` object can now use the full functionality of `LoanSlipDocument` via the static class `LoanSlipUtil` without knowing the internal implementation of `LoanSlipDocument`.
  - Cons: There is more code to be written and maintained.

We have decided to go with Alternative 2 as it **decouples** the code, making it easier to modify in the future. On the contrary, Alternative 1 will introduce unnecessary dependencies between classes, thereby **increasing coupling** and **reducing maintainability**.

Aspect: Implementation to allow extension (loan multiple books at one go).

- **Alternative 1 (current choice):** Mount a loan in `LoanSlipUtil` for each book.
  - Pros: Will be able to mount multiple loans using `LoanSlipUtil` before generating all loans in a single loan slip.
  - Cons: Will require more code when mounting loans in the Facade class.
- **Alternative 2:** Re-create `LoanSlipDocument` whenever a new loan comes in.
  - Pros: Will only need to make adjustments to `Logic` component to contain an `Optional<LoanSlipDocument>` field and update when a new `Loan` comes in.
  - Cons: Violates Single Responsibility Principle as the Logic class will now have to change if we change the implementation of `LoanSlipDocument`.

We have decided to go with Alternative 1 as it allows us to have flexible code that is **easily extendable**. Furthermore, it adheres to good programming practices as compared to Alternative 2, which **violates the Single Responsibility Principle**.
