Bookshelf.me Part 5: Final Report

Members:

- Jennifer Dooley
- Soham Shah
- Girishkumar Ramkumar
- Samuel Taylor

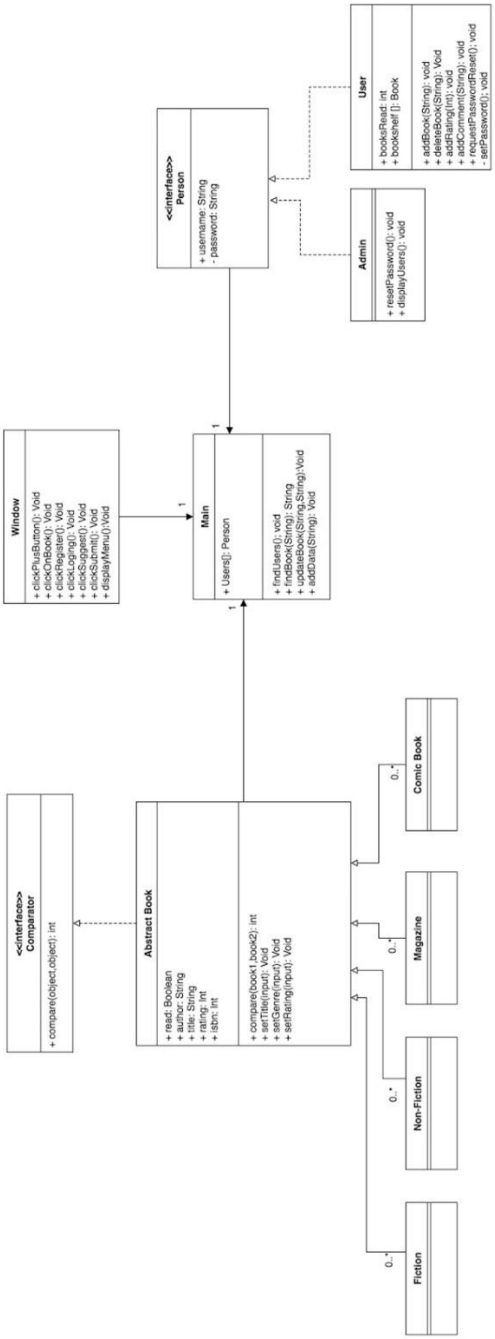
Implemented Features:

ID	Title/Description
UR - 1	User Register
UR - 2	User Login
UR - 4	User Add Book
UR - 6	User Rate Book
UR - 7	User Statistics (number of books read)
UR - 8	Admin View Storage
UR - 9	User Comment/Review
FR - 1	Pre-populated bookshelf
FR - 2	Fill in file path for adding books
NFR - 1	Adding books is straightforward
NFR - 2	Program starts without bugs
UR - 3	Admin view user credentials

Non-implemented Features:

ID	Title/Description
BR - 1	Pay by credit card
BR - 2	Contact Customer Service
UR - 5	User add book by filepath
UR - 10	Book Suggestions

Class Diagrams Before:



Class Diagrams After:

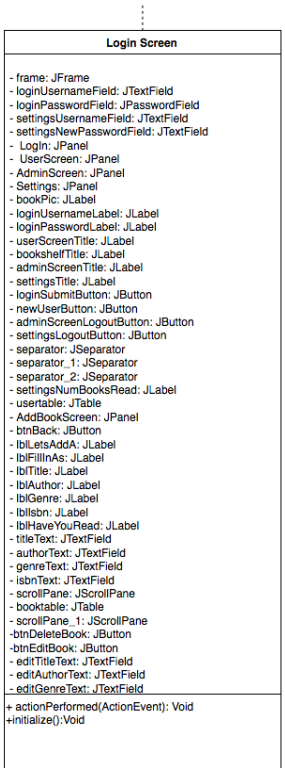
What Changed?

The biggest change was instead of using a main class, we used an authenticator class that works a little different. Originally the idea was to use the main class to contain the users and some data about the user's books. Now the authenticator is used to verify users/admins exist and to add new ones if they are not present in the database. Another change made to the class diagram was the book class. It no longer has subclasses of different types. Also, we didn't implement the comparison interface. Instead, we just display the user's bookshelf in the order the books are in the array.

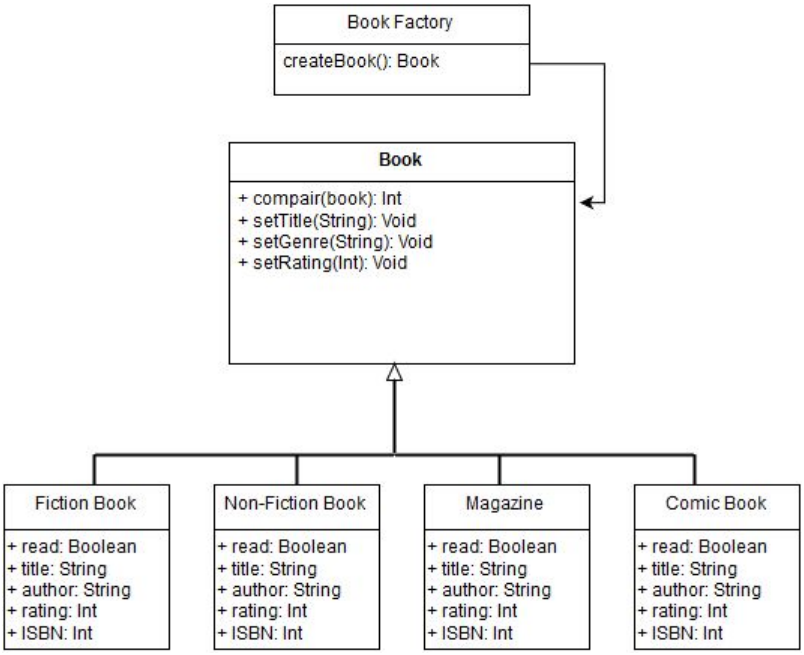
Design Patterns

Singleton - We used the singleton method to create a single instance of our authenticator class. This way, there was only one object of the authenticator which contained all the data for the users.

The Blob - The window class within our project shows resemblance to the blob (monster) anti design pattern. There are many functions nested within the class that could potentially be broken down into smaller classes to avoid conflicts as the project grows. We also considered using the MVC design pattern going forward in order to scale the program.



The Factory - If we had kept the fiction, non-fiction, magazine, and comic book classes in the project the factory design pattern would have been helpful with handling these different types. The book factory class would have made a certain type of book class depending on the input type of the book.



The Takeaway:

Throughout the course of this project and class we have learned about the process of creating, designing, and implementing systems. Designing the UML diagrams before starting to code helped us to stay more focused. We knew what needed to be implemented and how. We also knew where to begin programming as we had a high level idea of how everything would tie together. Also, looking at the design patterns helped with recognizing patterns within classes to analyze data within them efficiently. While we only used a single design pattern, we noticed how we could have used certain design patterns to make our code easier to maintain as we added features. Something we noticed is that as we continued adding features to our code, our window class got a lot more complicated. Because we did not follow the MVC framework, we had a lot of if checks in the window rendering code itself. In retrospect, we've gained a greater appreciation for why MVC is used in industry and understand how having a better approach can save time in the long run.