Isaac Gideon Tan - Project Portfolio

PROJECT: PrisonBook

Overview

Prisonbook is a desktop address book application used for Prison Guards/Inmates management. The user interacts with it using a CLI, and it has a GUI created with JavaFX. It is written in Java, and has about 10 kLoC.

Summary of contributions

- **Major enhancement**: added **Added login and logout security system with access levels**
  - What it does: Allows users of the PrisonBook to log in and logout with their usernames and passwords. The system also controls access levels (such as read/write permissions) based on the user rights.
  - Justification: risonBook must have this feature to improve the security of the Prison. Not all prison guards using the PrisonBook should be able to edit and make changes to the database. They should also only be able to view information pertinent to them to avoid security breaches.
  - Highlights: This enhancement affects the logic of the entire PrisonBook and requires thoughtful design to implement.
- **Minor enhancement**: added Prisoner and Guard tags to facilitate all further functionality to our PrisonBook
- **Code contributed**: [Functional code (https://github.com)] [Test code (https://github.com)] *{give links to collated code files}*
- **Other contributions**:
  - Project management:
    - Code Quality Manager
    - Opened and Tagged Issues
    - Assigned issues to different team members
  - Enhancements to existing features:
    - Command Aliases to shorten the amount of typing that users have to do
  - Documentation:
    - Updated documentation accordingly
  - Community:

- Tools:

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.*

Logging in: `login` [0] (v1.2)

Logs in to your account on the PrisonBook
Format: `login user/USER_NAME pw/PASSWORD`

Examples:

- `login user/prisonwarden pw/password3`

FOR USER ACCEPTANCE TESTING PURPOSES: The following user accounts have been pre-added

- Username: prisonguard | Password: password1 | Security Level: 1
- Username: prisonleader | Password: password2 | Security Level: 2
- Username: prisonwarden | Password: password3 | Security Level: 3

Logging out: `logout` [0] (v1.3)

Logs out of your account
After logging out, you will not be able to undo actions that were completed before logging out.

Format: `logout`

Check Log-in Status: `status` [0] (v1.4)

Checks your current log-in status
Format: `status`

Add new user: `adduser` [3] (v1.4)

Adds new user to have access to the PrisonBook
Format: `adduser user/NEW_USERNAME pw/NEW_PASSWORD sl/SECURITY_LEVEL`

Higher security levels allow users to access a greater range of commands: Security levels for new users must be from levels 1 to 3, refer to the list below for the differences in security levels.

- Security Level 1: Most commands that only require read access.
- Security Level 2: Most commands that require write access.
- Security Level 3: Complete access to all commands.

Examples:

- `adduser user/newuser pw/newpassword sl/1`

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.

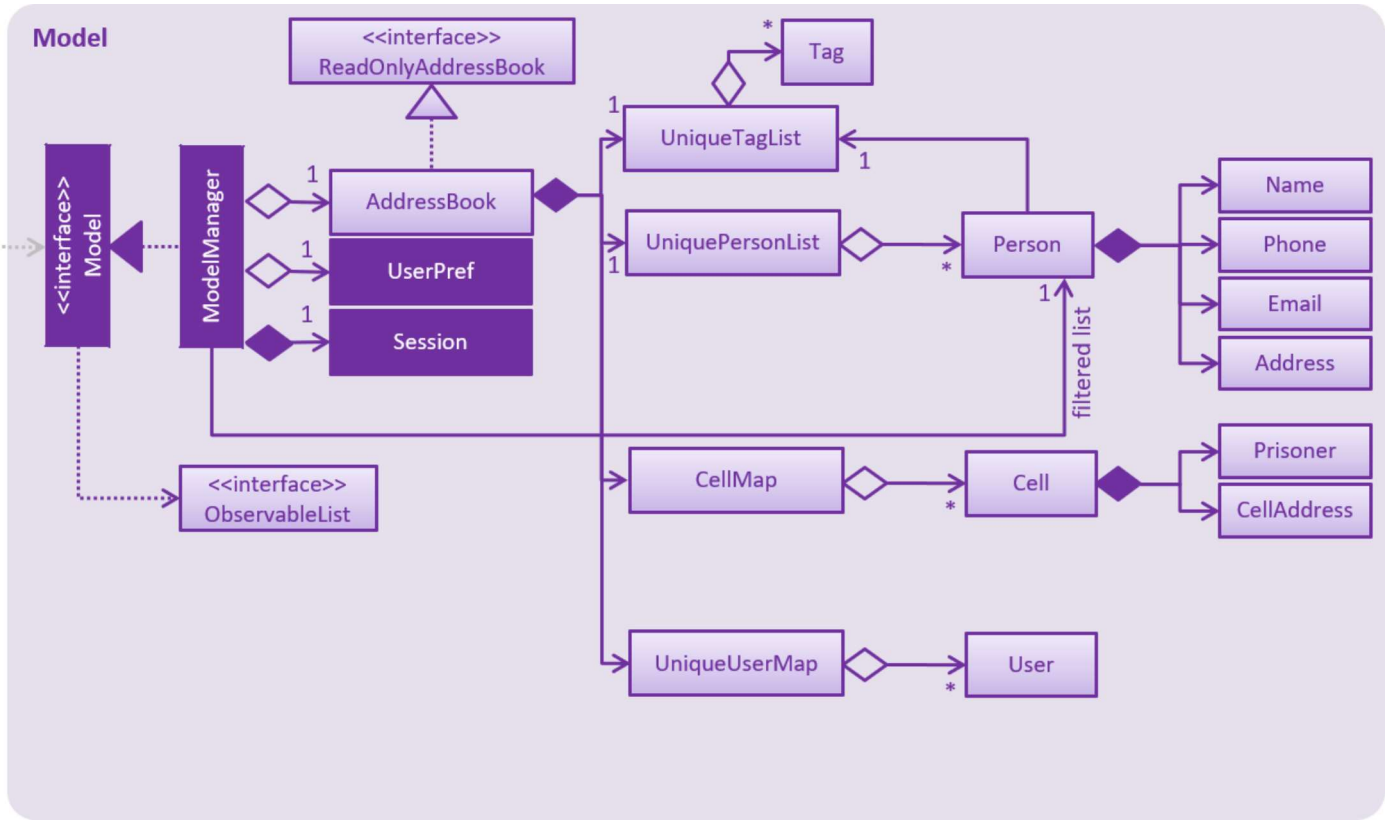
Login and Security Feature

Current Implementation

The Login and Security feature in the PrisonBook is implemented using Users and Sessions.

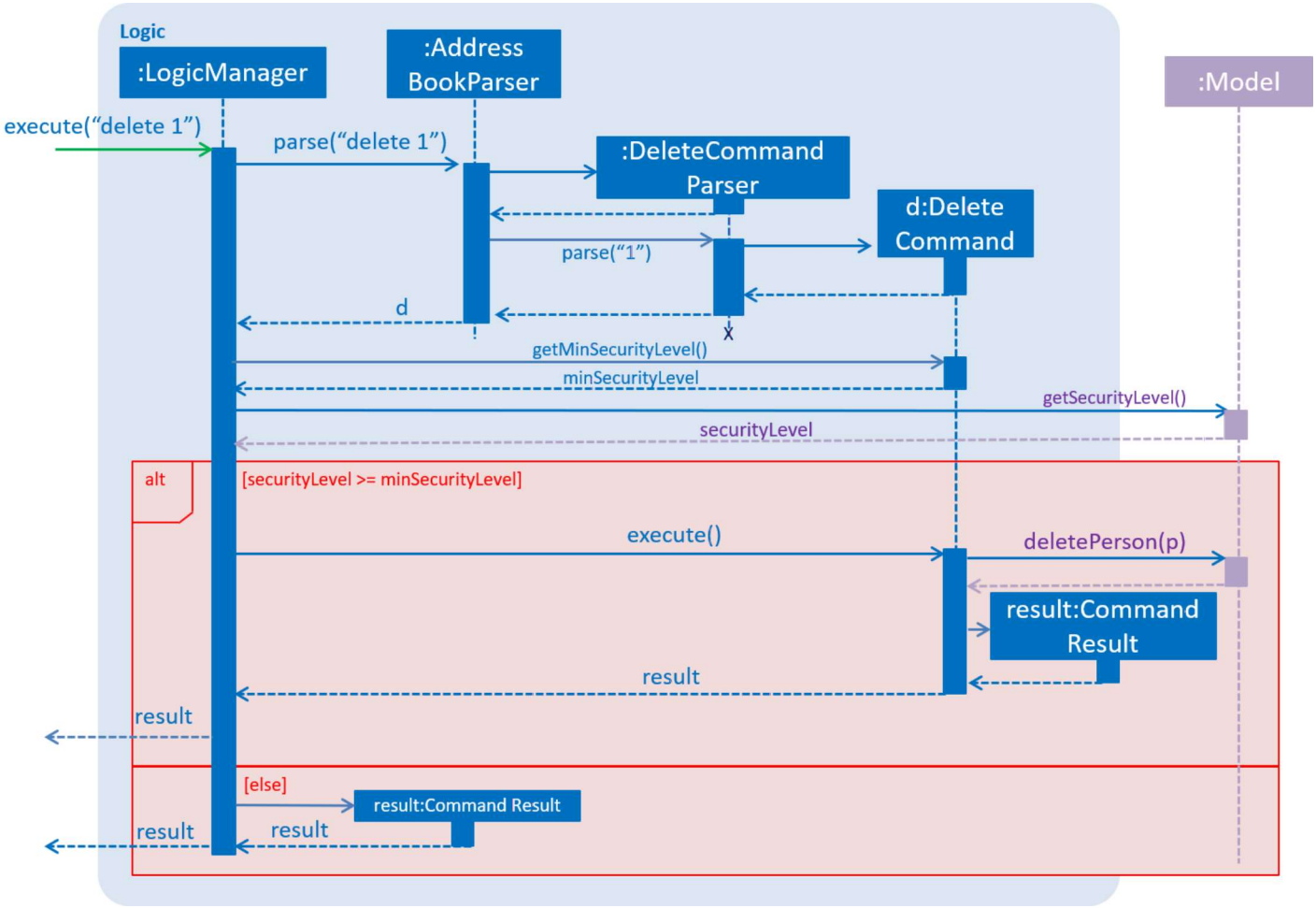
Sessions

Sessions are implemented with a `Session` class which resides inside `Model Manager`. It supports the logging in and logging out of a user as well as support restricted access for specific commands. For example read access (list command) vs write access (add command and edit command).



As can be seen from the diagram, a Session is initialised upon the creation of the ModelManager. The session stores information such as the logged-in user's username, as well as the security level of the user.

The accessibility of each command by the user is implemented on the logic level. Each command has a static `minimumSecurityLevel`. The usage of a command goes through the sequence diagram as follows (the example command being used here is the Delete command).



The `LogicManager` checks the `minSecurityLevel` of the command against the `securityLevel` of the current `Session` in the Model. If `the securityLevel` greater than or equal to the `minSecurityLevel`, the LogicManager will then call the `execute()` method on the Command.

Design Considerations

Aspect: Implementation for checking of security level for command access

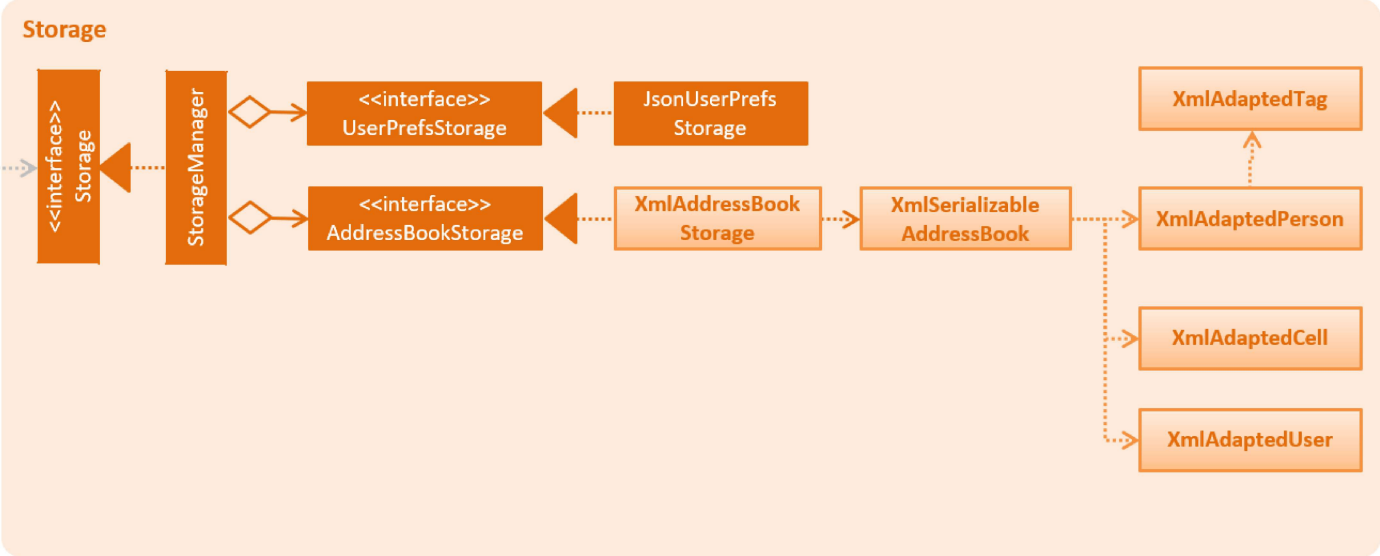
- **Alternative 1 (current choice):** Implemented by the `LogicManager`
  - Pros: Decreases coupling between `Command` and `Session`. This allows for flexibility in future implementation changes.
  - Cons: Sequence during execution needs to go back and forth with the `LogicManager`
- **Alternative 2:** Implement at the `Command` level

- Pros: Easy to understand, the sequence during execution keeps moving on to the next step instead of going back and forth with the LogicManager.
- Cons: Commands are also responsible for checking if they should be run. Violates Single Responsibility Principle and Separation of Concerns as `HistoryManager` now needs to do two different things.

Users

Users are implemented with a `User` class, as well as a `UniqueUserMap` class which also reside in the `Model Manager`.
The `UniqueUserMap` is implemented using a Hashmap which maps `usernames` as Keys and `User` as Values. Additionally, there is an internal list implemented via an `ObservableList` of all the Users in the Hashmap.

To save the database of the users, a new type is added to the storage so that it is able to store the Users in the storage file.



Upon initialization of the AddressBook, the AddressBook will read the user data from the storage file and create the internal list of Users. The list is then iterated through and mapped to the HashMap.

When adding and deleting users, both the Hashmap and the internal list are updated so as to reflect accurately in the model and stored correctly in the storage.

Currently, the user data, including passwords are stored in plaintext, which may pose a security problem. For future implementations, we will include hashing of the passwords before storage to as to increase the security of the application.

Design Considerations

Aspect: Data Structures for UniqueUserMap

- **Alternative 1 (current choice):** Hashmap + ObservableList

- Pros: Easy to access users, can immediately call up the correct `User` object by passing in the username key. Whereas the ObservableList is used to complement the storage of the user data, where it is necessary to iterate through all the Users in order to store in the storage .xml file.
- Cons: Duplicate information stored in two different data structures.

- **Alternative 2:** ObservableList only
  - Pros: No duplication of Users. Storing is also easy as it uses the existing AddressBook storage framework which works with ObservableLists.
  - Cons: Everytime we want to access a specific User, we have to iterate through the entire list of Users in order to do so.
- **Alternative 3:** HashMap only
  - Pros: Fast access of users in O(1) time
  - Cons: Unable to be stored efficiently with the current method of storing which works with XML file.

Aspect: Storage location of User Data

- **Alternative 1 (current choice):** Stored in the same XML File with the AddressBook data
  - Pros: Easy to implement with current method of saving, where the entire AddressBook is upon an Address Book Changed Event. There is no need to maintain an additional stack for a User Database for UndoRedo.
  - Cons: User data should theoretically be separated from AddressBook data (persons, cells, tags).
- **Alternative 2:** Store in a separate XML File
  - Pros: Separation of storage for two different aspects. If User data file is corrupted, it does not affect the AddressBook data and vice versa.
  - Cons: Very difficult to implement with the current method of saving. Furthermore, there needs to be addtional stacks implemented to keep track of the past states of User data for the UndoRedo functions.
- Isaac Gideon Tan (zacci)

  1. Major enhancement:
     Login and Authorization Levels: Enable creation of account for guards, logging in, and authorization levels for commands that may compromise the security of the Prison data.
  2. Minor enhancement:
     Additional Attribute to store 'Role' information. To identify the person as a Prisoner or Guard

Major Feature: Users, Sessions and Security Levels

Logging in and Logging out

1. For security purposes, users of PrisonBook need to login with their user details in order to to view the information on the PrisonBook. They can also logout after finishing their session, so that the next person will need to login again.

    a. Prerequisites: You must not be logged in. (By default, you are not logged in when you first open the application.)
    Test case: Type `login user/prisonguard pw/password1`
    Expected: PrisonBook will display `Login Success`

    b. Prerequisites: You are logged in.
    Test case: Type `status` to show your current session's details
    Expected: `Username: prisonguard Security Level: 1`

    c. Prerequisites: You are logged in.
    Test case: Type `logout` to logout
    Expected: PrisonBook will display `Successfully logged out`. Furthermore the person details in the left panel will be cleared, and the cell details in the right panel will reflect that you have insufficient authority to view the information.

Commands have restricted access

1. A minimum Security Level is required to use and access certain commands.
   Refer to Section 5: Command Summary for the detailed list of commands and their respective minimum security level ranging from 0 to 3.

    a. Test case: Type `logout` to ensure that you are logged out. (By default you are not logged in when you first open the application)
    Type `list`
    Expected: PrisonBook will show `Your security level is insufficient to access this command.`

    b. Test case: Login using `login user/prisonleader pw/password2` (This user has Security Level of 2)
    Type `list`
    Expected: PrisonBook will say `Listed all persons` and show the full list of persons on the left column of the UI.
    You can test the other commands with the following user details:

        i. `login user/prisonwarden pw/password3` Security Level 3.

        ii. `login user/prisonleader pw/password2` Security Level 2.

        iii. `login user/prisonguard pw/password1` Security Level 1.

Adding a new user

1. You can add new users to the PrisonBook when you want to give access to new people.

a. Prerequisites: Have sufficient security level[3]
Test case: `adduser user/prisonguard99 pw/password2 sl/1`
Expected: `New user prisonguard99 added to PrisonBook`
You can then proceed to logout and attempt to login with the new user `prisonguard99`.

b. Prerequisites: Have sufficient security level[3]
Test case: `adduser user/prisonwarden pw/password3 sl/3`
Expected: `prisonwarden is already a user in PrisonBook`

Deleting a user

1. You can delete users from the PrisonBook

a. Prerequisites: You must be logged in with a higher security level than the user you want to delete. (Users with security level 3 can delete anyone, including other security level 3 users.)
Test case: Login with `login user/prisonleader pw/password2`, then type `deleteuser user/prisonguard99` (Which was added above in F.2.3)
Expected: `User has been successfully deleted.` You can logout and attempt to login to `prisonguard99` but you will not be able to successfully login as the user has been deleted.

b. Prerequisites: You must be logged in with a minimum security level of 2.
Test case: `delete user/nonexistentuser` Expected: `The username does not exist`

Minor Feature: Role Attribute added to Persons

1. Allow user to indicate to the PrisonBook if added person is a Guard or a Prisoner.

a. Prerequisites: Have sufficient security level[2]

b. Test case: `add n/Sample Prisoner a/Chua Chu Kang p/91234567 e/sp@example.com r/p`
Expected: New person to show up on the UI with the name Sample Prisoner and shown to be a Prisoner on the person card.

c. Test case: `add n/Sample Guard a/Chua Chu Kang p/91234567 e/sg@example.com r/g`
Expected: New person to show up on the UI with the name Sample Guard and shown to be a Guard on the person card.

Last updated 2018-04-15 14:50:08 UTC