Java Fundamentals End Assignment

Inholland University of Applied Sciences
Teachers: Mark de Haan & Wim Wiltenburg
Submission deadline: Before the exam

Introduction

In order to complete the course of Java Fundamentals students need to complete two tasks:

1. An end assignment, consisting of 4 parts that result in one application.
2. A classroom exam session, extending the functionality of the end assignment with three new features.

The document before you details the requirements for this end assignment.

General requirements:

- Unless otherwise specified, requirements are must-have for a passing grade.
- The application must be written in JavaFX.
- The application must be submitted as a zip (not .rar, .7z, or other formats) file containing the following:
  - /src – directory
  - pom.xml
  - Readme.md
- The name of your zip file must be: [your-name]-[your-student-number]-[end-assignment].zip, e.g. ***wim-wiltenburg-123456-end-assignment.zip***
- Both code quality and style will be reviewed:
  - **Style**:
    Java Naming Convention, appropriate naming of classes, variables and methods.
    Use proper formatting
    Profanity will lose you points
  - **Code Quality**:
    1. Clear and concise code. Should someone other than you read the code, they should be able to understand it. Don't assume they would. Teachers are people too.
    2. Methods must not be too long. A general rule of thumb is that 30 lines for a method is too long.
    3. Use Object Oriented Principles.
    4. Think about maintainability.
- Your application must be able to start up in the IDE of the reviewer.
- The screenshots in this document are indicative. Feel free to arrange and design it to your own taste, as long as the required functionality is there.
- If you have specific usernames/password that are needed to log into the application, please share them, e.g. in a README.md file in the root of the project.
- Before creating your zip file, do a code reformatting on all your files, and optimize your imports!

Objectives

Your task is to build an application for a library. The users of the application will be able to lend and receive items. It will also be possible to manage the collection of items, and to manage the members of the library. The application consists of:

- A login screen
- Different views:
    - A screen for lending and receiving items
    - A screen for managing the collection
    - A screen for managing the members
- The login screen should be a separate window. The other functionality should all be usable in one main window.
- It is allowed to use separate (modal) dialogs for adding and editing the item collection and the members. The choice to use separate dialogs or the main window is up to you.
- **Do not** use pop-up windows (message- or dialogboxes) to display error messages.

Database object

The data in the application is maintained by a database object (not a real database) and is initialized at startup.

Important: For learning purposes, the data is not allowed to be static, and therefore the Singleton Pattern must **NOT** be used. Populate the database object with initial data.

Make sure to provide usernames & passwords in your README.md.

User experience

The example screen designs in this document are just an indication of what the UI could look like. The screen design is up to you, as long as a similar (or better) level of usability is achieved.

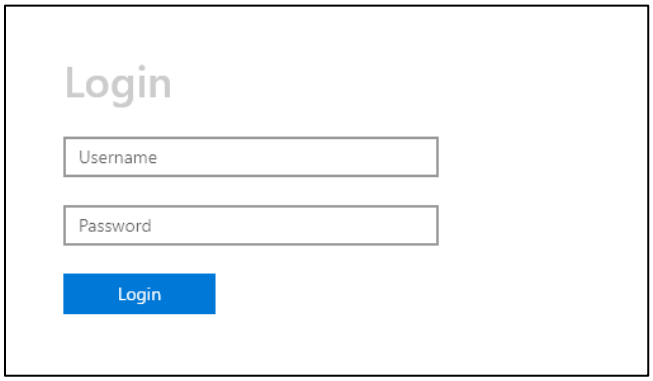
Assignment part 1 - Login Screen

The login screen is a basic screen with username and password. You need to have a proper mechanism in place to validate the input and grant the user access to the application.

First, make sure you have an in-memory database class that contains a collection of users. Initialize at least 2 example users in the constructor of your database class.

Then develop a window that allows users to login using the usernames and passwords stored in the collection. When the application starts, this window should be displayed.

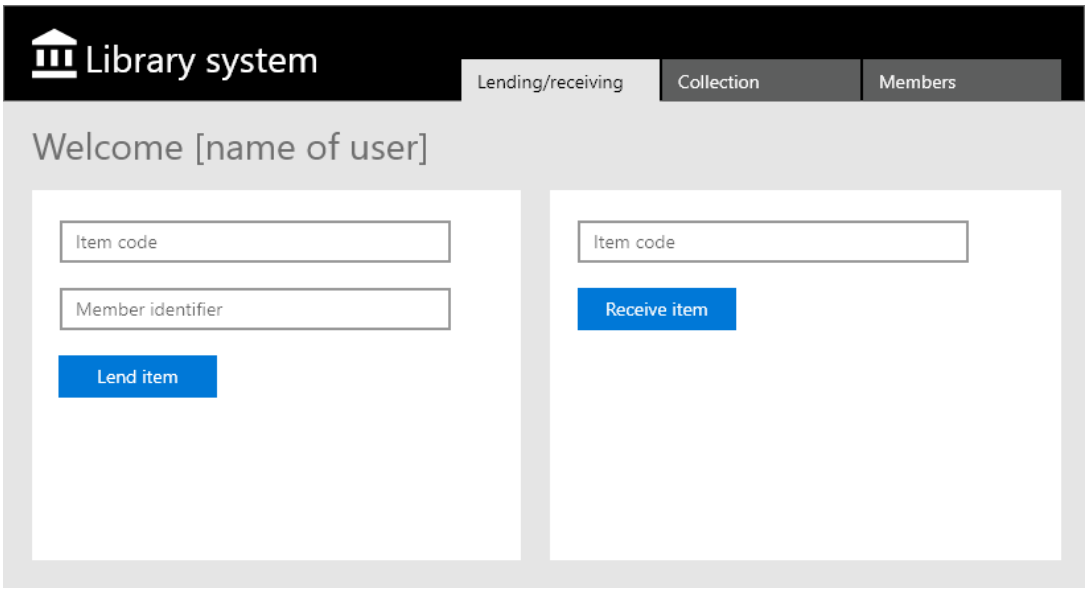
An example of what the window could look like:



- After clicking the 'Login' button with a correct username and password combination, the login window should close and a new (empty) main window should open.
- The functionality in the main window will be developed in the next assignments.
- The user must get a clear and descriptive message if something goes wrong during the login process. An example is to display: 'Invalid username/password combination' instead of 'Login failed'.
- Note that, as per the general requirements listed before, a pop-up dialog should not be used to display the error message.

Assignment part 2 - Main window

Once the user is logged in, the main window is shown.



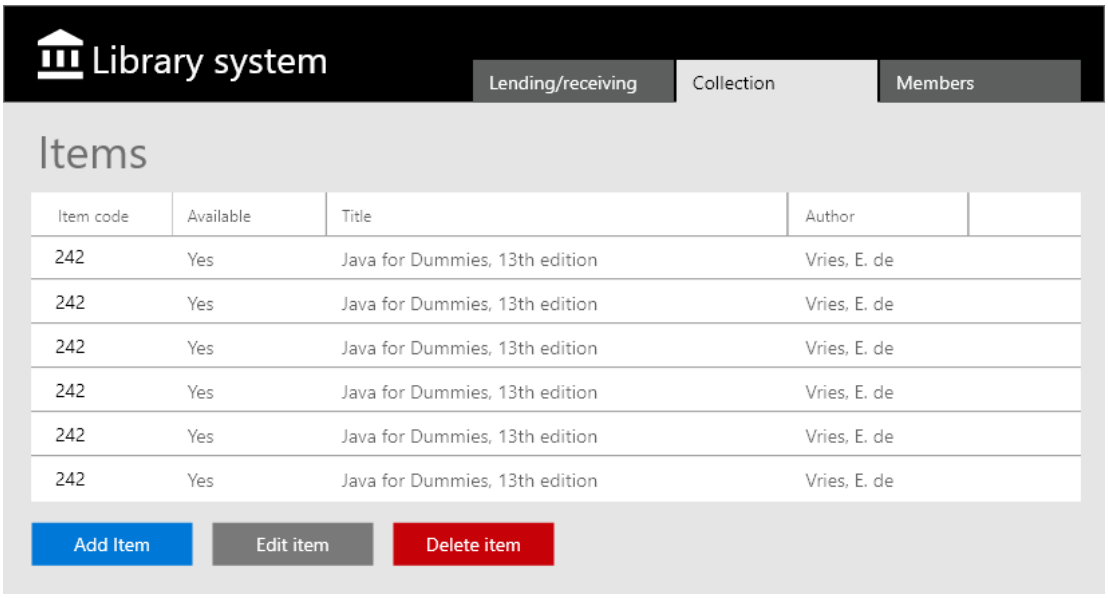
The main screen has two functionalities: Lending out items and receiving them. These are shown side by side in the main window.

Make sure your in memory database contains a few example members and items. Then develop the following functionality for the main screen:

- The full name of the logged in user should be displayed on the screen.
- When the 'Lend item' button is clicked, the status of that item should change in the system. The lending date should also be recorded for that item. The user should receive feedback that the item status has changed.
- When an item code is entered in the receiving part of the screen, and the lending date of that item is more than 3 weeks in the past, a message should be displayed that the item is too late. Display the number of days that the item is too late in the message.
- When the 'Receive item' button is clicked, the item status should change and the lending date should be cleared.
- When an incorrect item or member code is entered, a clear message should be displayed.

Assignment part 3 - Managing the collection

When the user opens the 'Collection' part of the application, the list of items in the collection should be displayed.



This screen provides standard CRUD (Create, Read, Update, Delete) functionality for items. How this is implemented, is up to you.

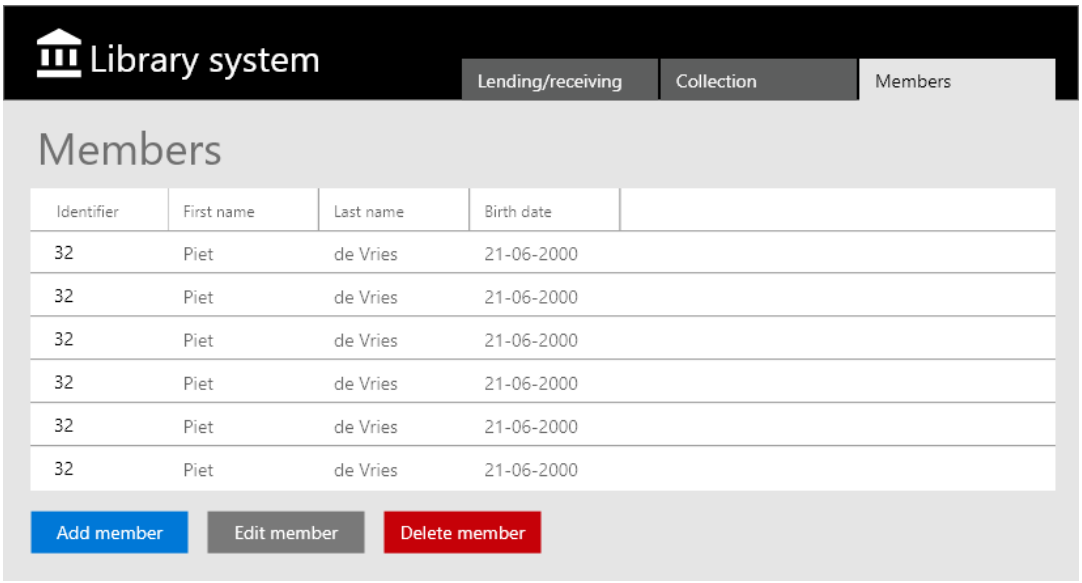
- When adding items, the item code should be automatically generated. This should be an (integer) number, preferably the highest currently used item code + 1.
- Ensure that when the item collection is modified, the changes are displayed immediately. The user should not have to click buttons or restart the application to see the changes.
- When an item is lent out to a member, the 'Available' column should display the value 'No'. When an item is not lent out, the column should display 'Yes'.

The following 'should have' requirement can be implemented:

- A search functionality that works on parts of both title and author is not mandatory for a passing grade, but will give points

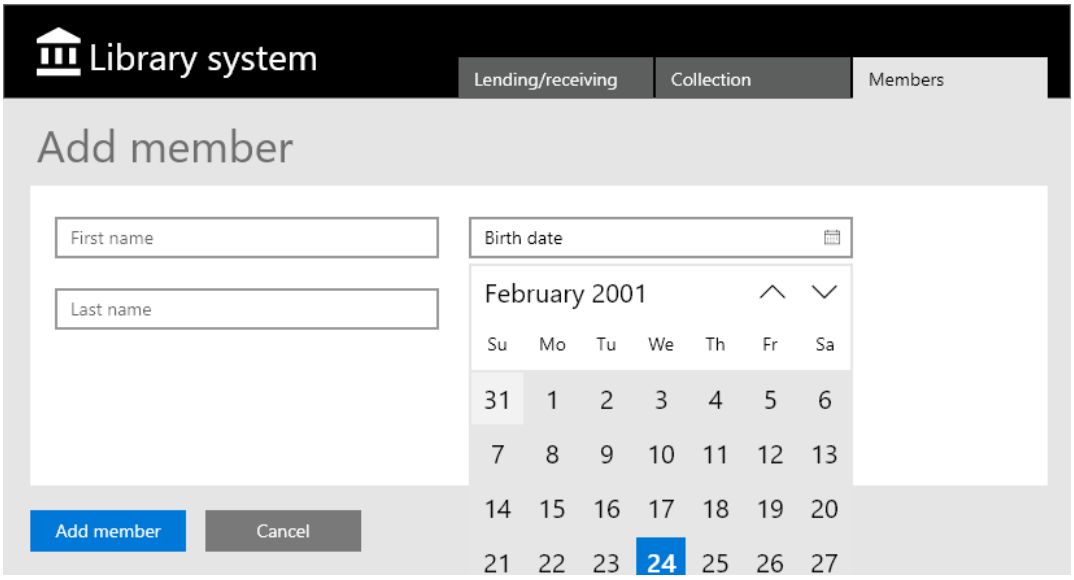
Assignment part 4 - Managing members

This assignment is similar to the previous assignment, but adds extra functionality.



This screen provides standard CRUD (Create, Read, Update, Delete) functionality for members. How this is implemented, is up to you.

- When adding a member, the identifier code should be automatically generated. This should be an (integer) number, preferably the highest currently used identifier + 1.
- When closing the application, the state of the database (the collection items and members) should be serialized to a file. When starting the application, the file should be deserialized. This ensures changes will persist throughout application restarts. Make sure the application does not crash when the serialized file is missing.
- When adding or editing a member, the birth date must be set in an intuitive matter, by both allowing the user to type the date in the correct format (dd-mm-yyyy) or selecting from a date picker.



- As with the previous assignment, a search functionality that works on parts of both first and last name is not mandatory for a passing grade, but will give points