Library Management System

Author: Mukash uulu Daniiar

December, 2019

Table of Contents

1.Acknowledgement:

First of all, I would like to thank my lecturer Mr. Nurlan Shaidulaev for helping me to acquire knowledge of "Java Programming Language". At the same time, he gave me the opportunity to learn something new related to our module like constructors, methods, arrays, encapsulation etc.

Beside from my lecturer, I like to thank my other classmates for helping to understand the assignment related questions more clearly.

Several terms used inside of project report:

LIS – Library Management System

2. Introduction:

This assignment is based on developing a LIS (Library Management System) using Java Programming Language. For that we used GUI (Graphical User Interface) in this development so that it will become more users friendly to interact.

Besides, we also added database to store important data related to our project.

4.Explanations:

In this documentation we have given explanations of how to interact successfully with this a LIS. We have explained here step by step so that it will surely help users to become more user friendly with it. Below are our explanations:

Required software:

Before execute this program, users need to do some works so that it will run properly into their system. First, they need to make sure their system is having “JDK”. If they don’t have it then they can download from this below link:

https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Depending on their system (Windows 64bit/32bit) they need to download and install. Then they need to add the “JAVA” files to their system “PATH” so that the system can run the program from CMD (Command Prompt). The path will show something like this “C:\Program Files (x86)\Java\jre1.8.0_25\bin;”. Now just add the address besides the current path directory and save it.

The other way they can execute this program in to download the IDE (Integrated Development Environment) on their system. They can download ECLIPSE or NETBEANS or IntelliJ depending on the windows (32bit/64bit).

NETBEANS:

https://netbeans.org/downloads/

ECLIPSE:

http://www.eclipse.org/downloads/

INTELLIJ:

https://www.jetbrains.com/idea/download/

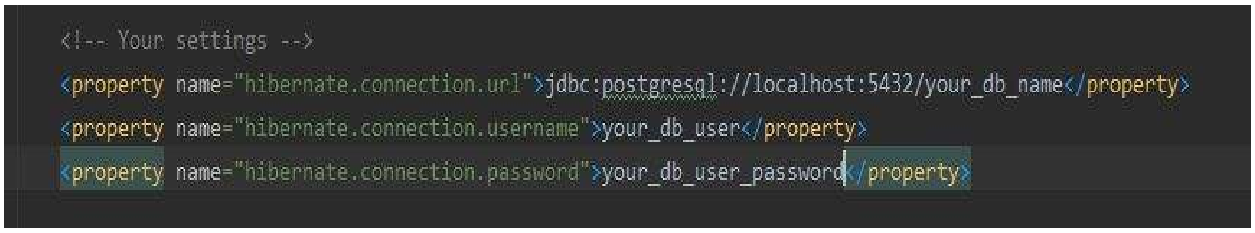
We developed this program using “**IntelliJ**”.

Also, we need to install PostgreSQL database which allow us to efficiently store, modify, get, and delete required data for our project. If they don’t have it then they can download from this below link:

https://www.postgresql.org/download/

4.Project Configuration:

After PostgreSQL installation we need to create one database user which will be used as daily database user, also we need to create database to hold our data. After creation of database and user change the values of properties in hibernate.cfg.xml



```
<!-- Your settings -->
<property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/your_db_name</property>
<property name="hibernate.connection.username">your_db_user</property>
<property name="hibernate.connection.password">your_db_user_password</property>
```

Figure 1: Configuration file

5.Excution Procedure:

When user executes this program, it will show the startup GUI (Graphical User Interface).

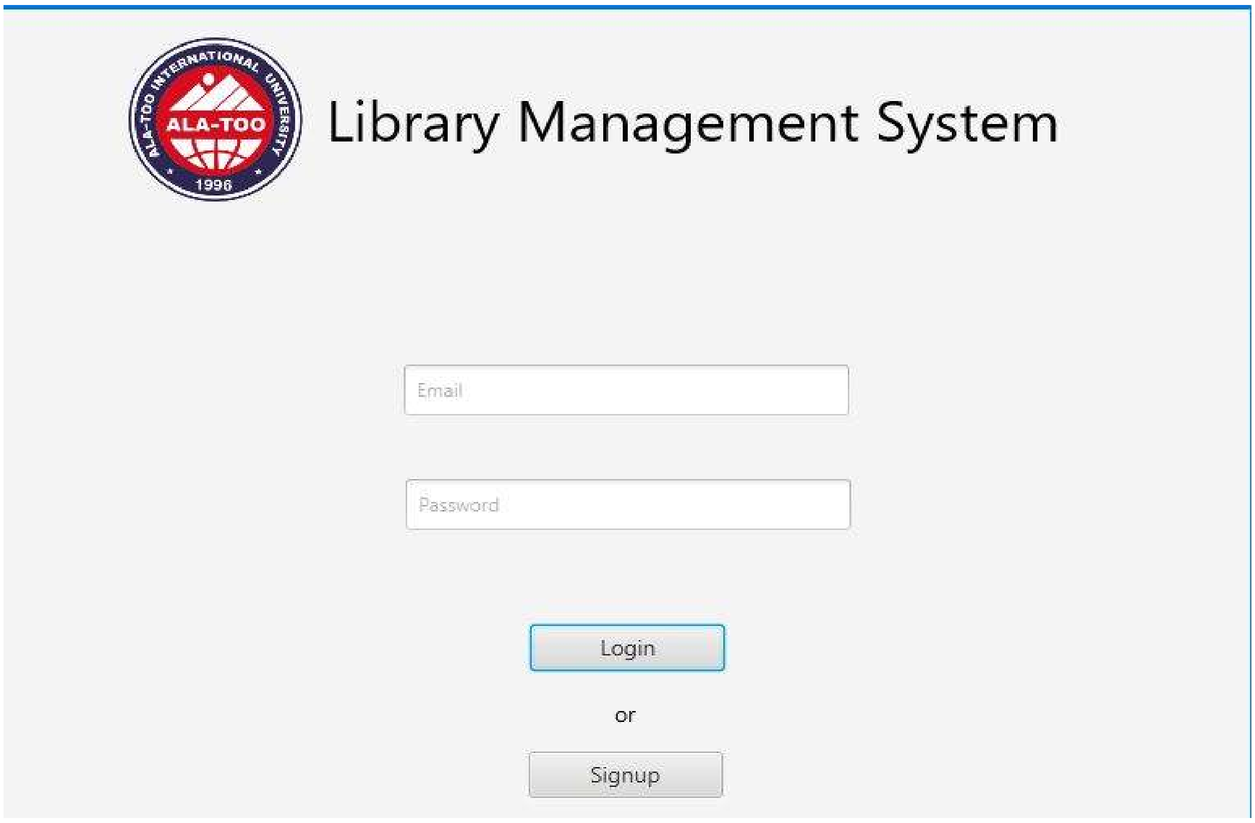


Figure 2: Login Screen

Login GUI allows to login with **librarian** and ordinary **user** account.

Now user have to option where to login with his **email** and **password** or user can create new account for himself by making registration to the system. Let's consider the second option and look at signup process.

6.Signup procedure:

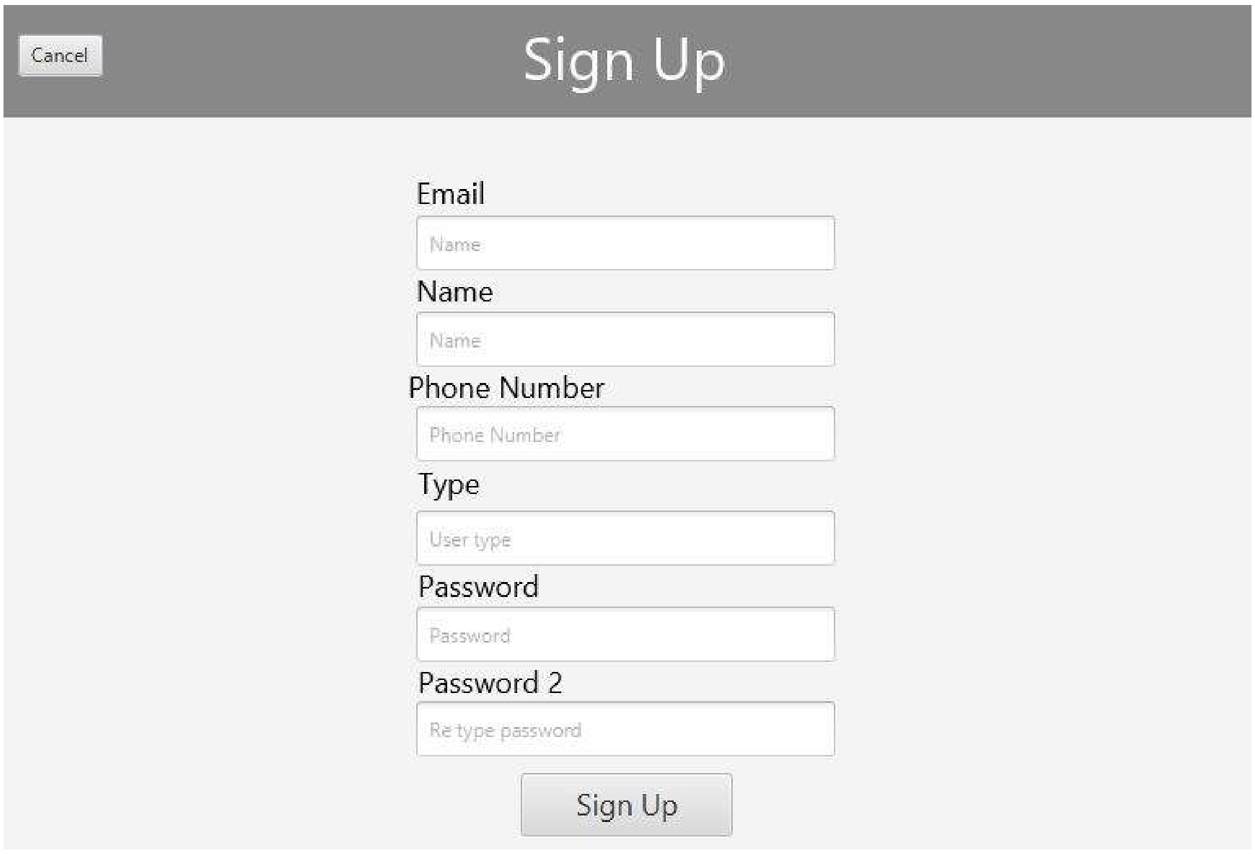


Figure 3: Signup Screen

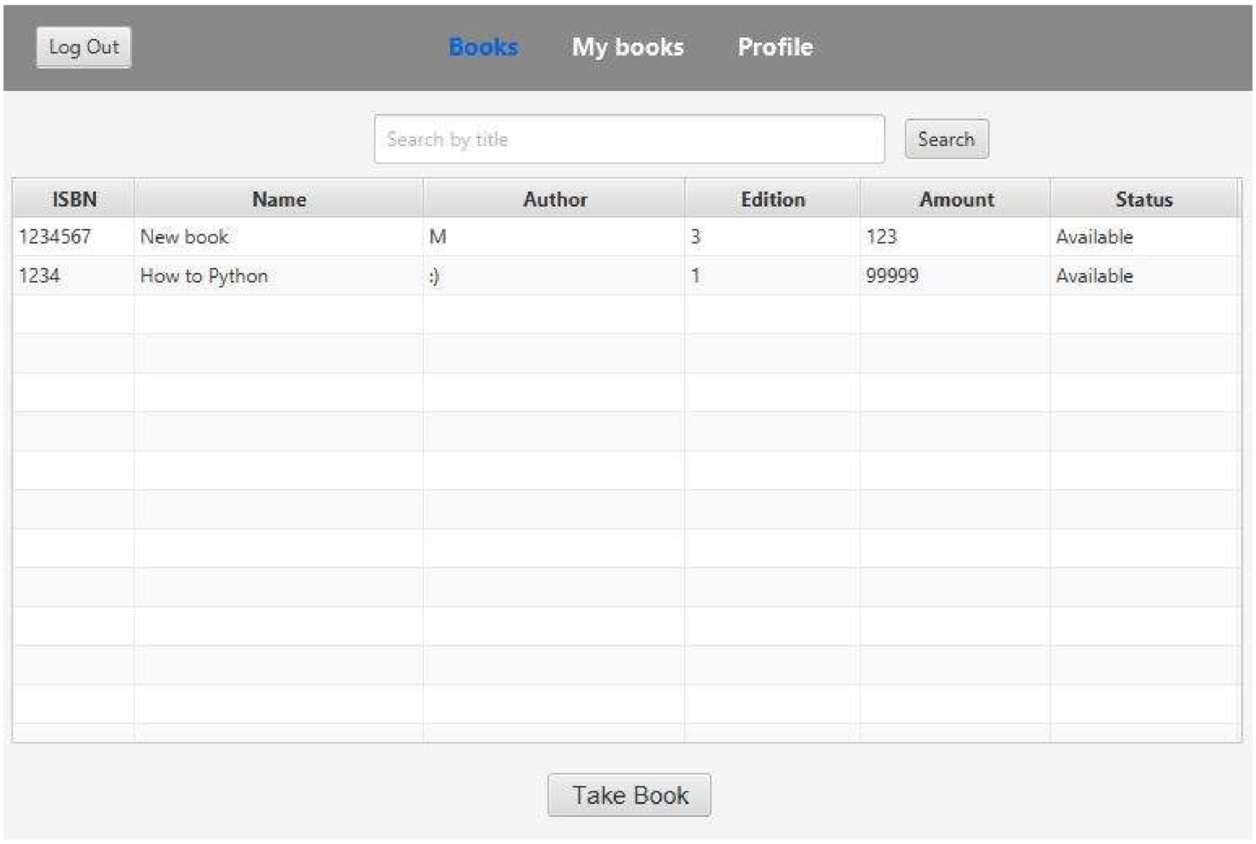
Here user needs to fill all required data about himself like name, email address, phone number, type of user like Student, Staff Member, and etc. Finally, user needs to provide password, both **password and password2 need to be identical**. Also, none of the form fields can be empty or user will see an error telling that some fields are empty. If user made **successful registration** he will be returned to **Login screen**. If user miss clicked the signup button he can be immediately returned to **login screen** by pressing **Cancel** button.

After registration user will be allowed to login to library management system.

7.User GUI:

7.1 Books:

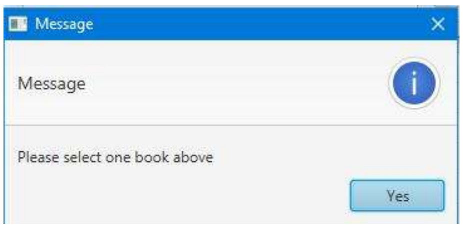
After successful authentication user will see Books screen with all books that are currently registered at library management system by librarian.



ISBN	Name	Author	Edition	Amount	Status
1234567	New book	M	3	123	Available
1234	How to Python	:)	1	99999	Available

Figure 4: Books available at LIS

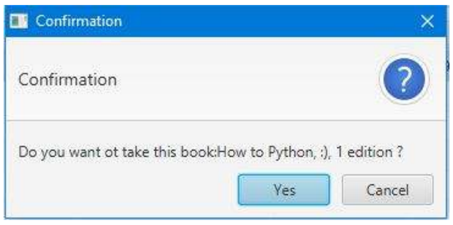
Here user can **choose book** and take it by clicking **Take Book** button, if no books is chosen he will see alert '**Please select one book above'**



Please select one book above

Figure 5: No books selected alert

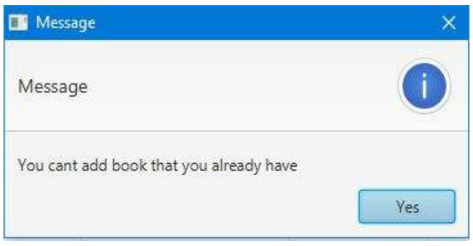
If user selected book he will see confirmation box asking whether he wants to add this book to his list.



Do you want ot take this book:How to Python, :), 1 edition ?

Taking book will decrease number of available books. If number of specific books reaches zero the book will be unavailable to all users and its status will become **Unavailable**.

If user **already took this book** he will see the alert box telling that user **can't add this book to his list**.



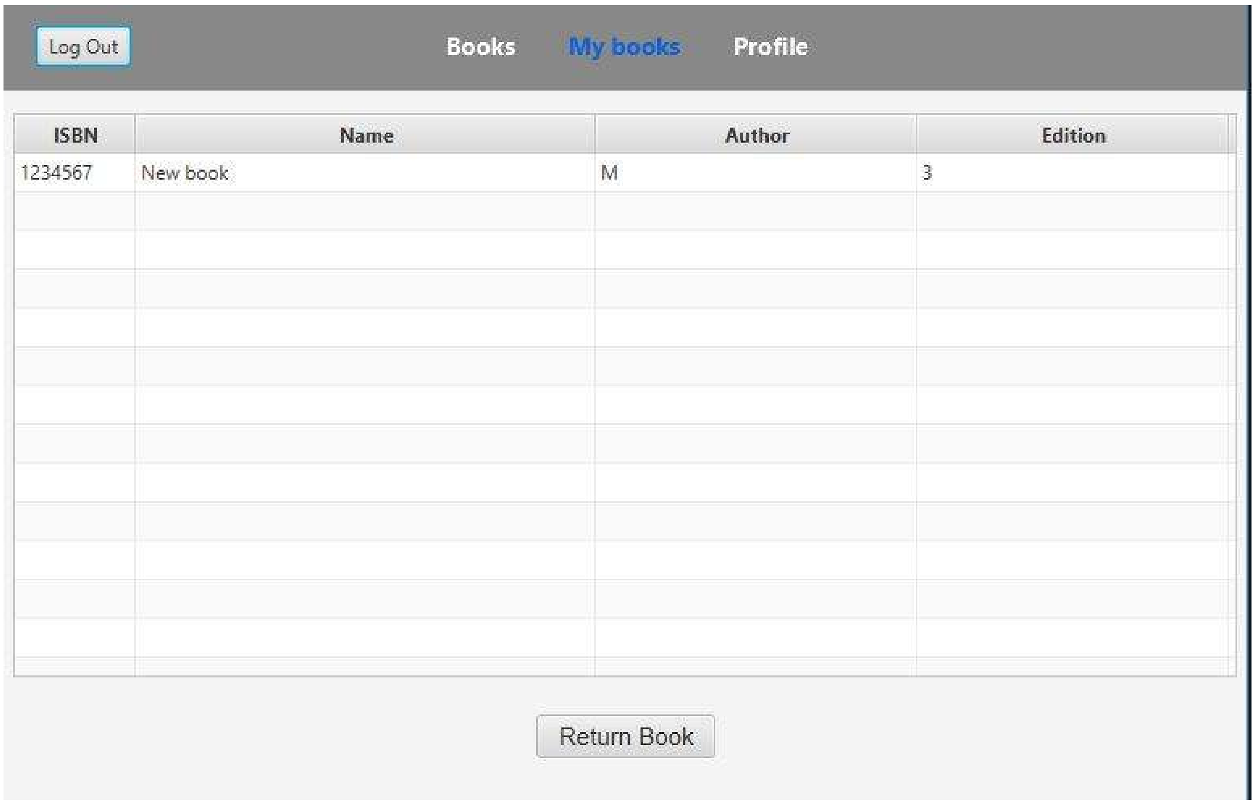
You cant add book that you already have

Figure 7: User already having specific book

All the books that was taken can be seen at **My Books** tab

7.2 User Books:

Inside of **My Books** tab we can see all the books that was taking by user.



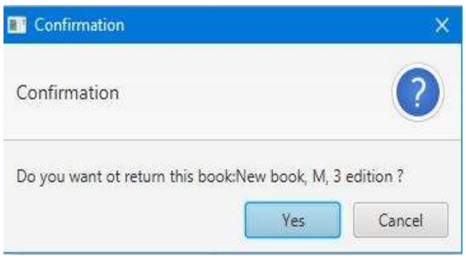
ISBN	Name	Author	Edition
1234567	New book	M	3

Figure 9: Books that was taken by current user

By clicking **Return Book** user can return book. If no book was chosen user will see alert that no was book was taken, similar to **Books** Screen.

If book was select user will see confirmation box where asking whether he wants to return specific book.

Returning Book will remove selected book from user's book list and add to the general book pool and Increase the number of available books.



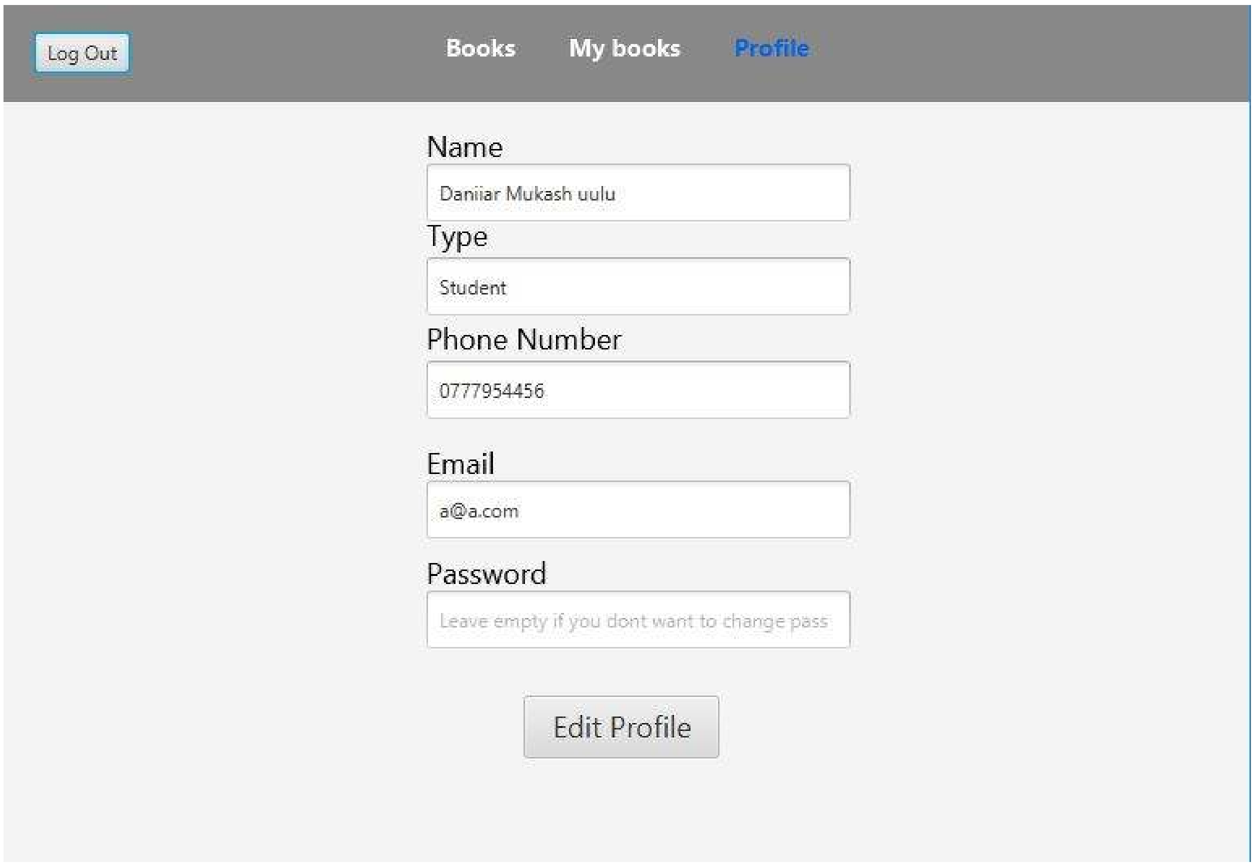
Confirmation

Do you want ot return this book:New book, M, 3 edition ?

Figure 8: Return Book confirmation

7.3 User Profile:

Inside of **Profile Tab** we can see all the information about current user and fields that prepopulated with data which is equals to user's data.



Name: Daniiar Mukash uulu

Type: Student

Phone Number: 0777954456

Email: a@a.com

Password: Leave empty if you dont want to change pass

Figure 9: User Profile

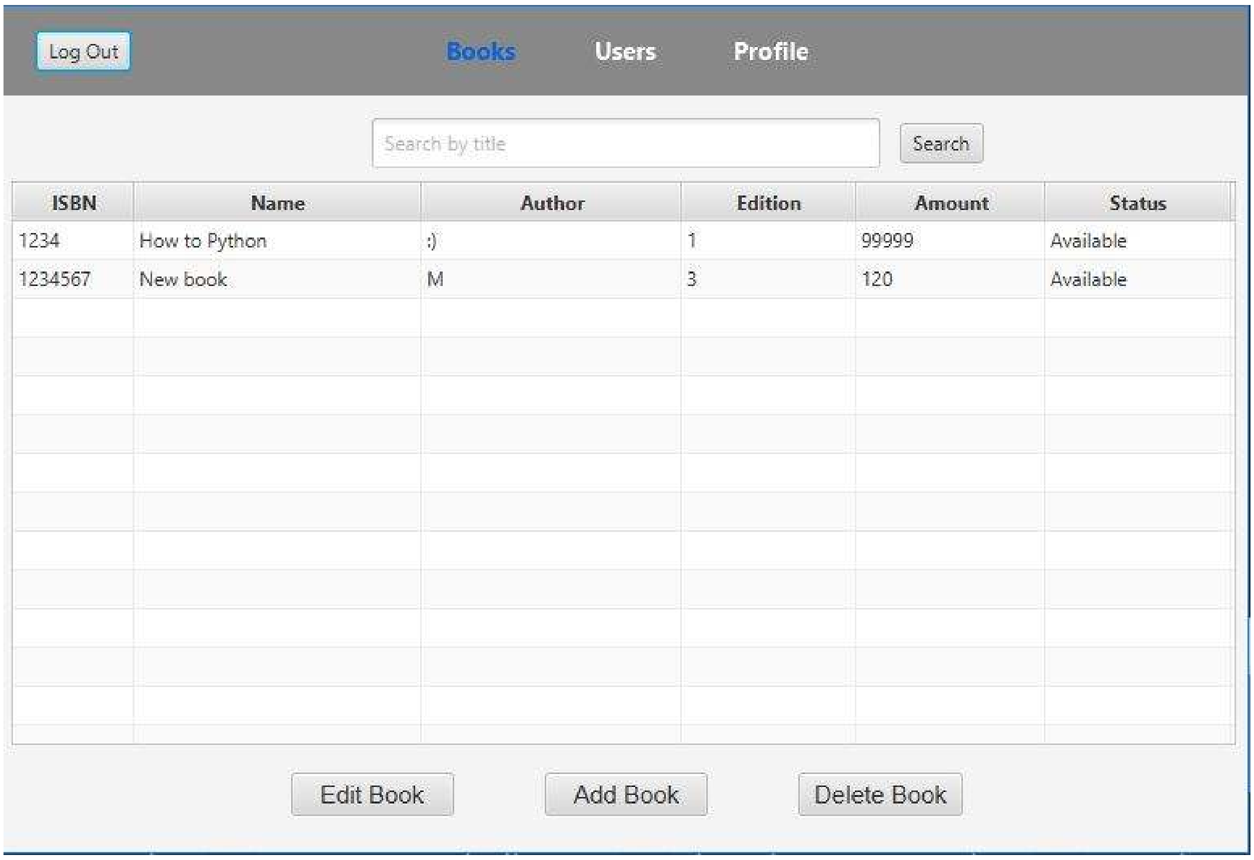
Here user can edit data about himself. **None of the fields can be empty except for the password field**, if user is not willing to change his current password he can leave this field empty.

By clicking **Edit Profile** user will update their profile with the data from corresponding fields.

8. Librarian GUI

8.1 Books:

Now let's **Login as Librarian**. First thing that librarian sees is the same table with all available books at LIS but with some more control. Librarian can add, delete, and edit books from this **Books** screen.



ISBN	Name	Author	Edition	Amount	Status
1234	How to Python	:)	1	99999	Available
1234567	New book	M	3	120	Available

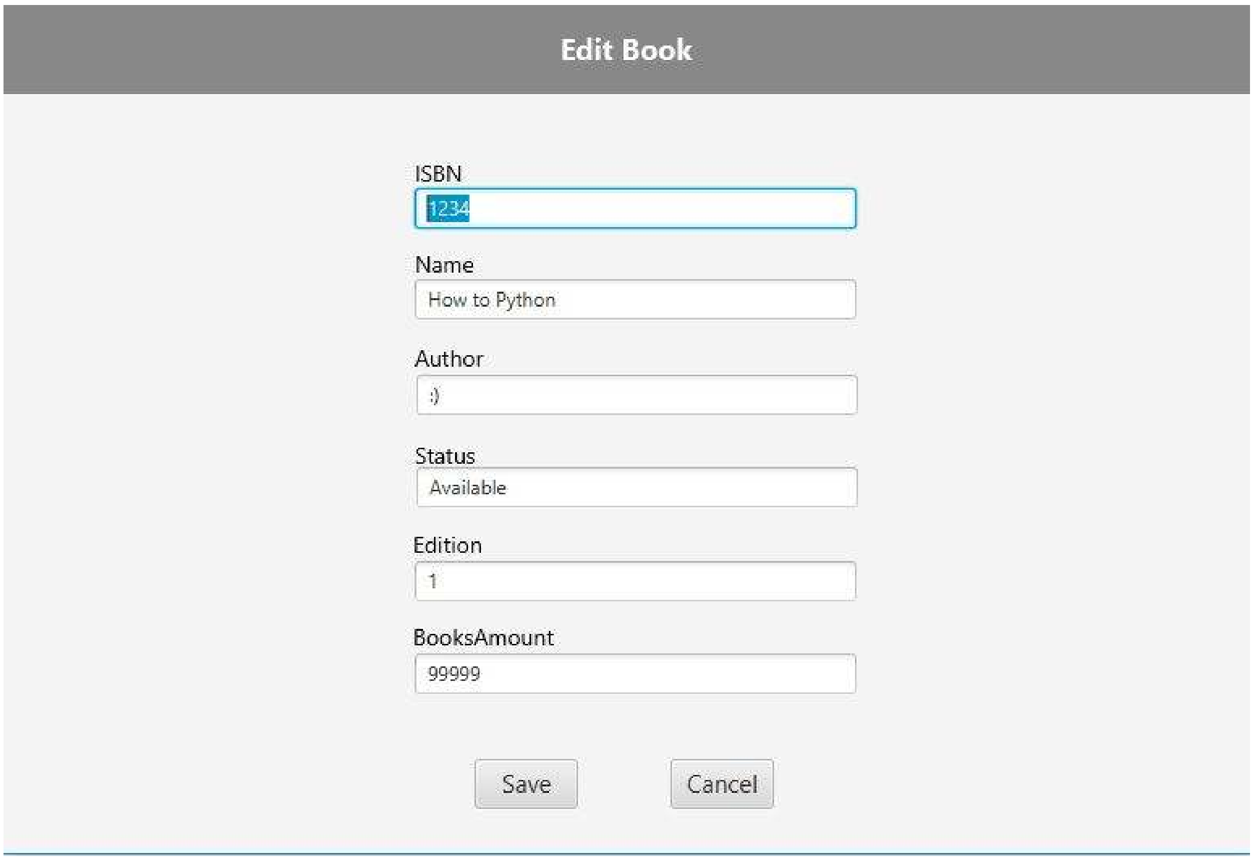
Figure 10: Librarian Books screen

Button Below correspond to actions they describe.

By selecting book and clicking Edit Book librarian will see another screen with book edition form, if no book was selected librarian will see alert box similar to user **Books** screen.

8.1.1 Edit Book:

By Editing special book librarian can increase number of this book, change Name, Author, Edition



ISBN: 1234
Name: How to Python
Author: :)
Status: Available
Edition: 1
BooksAmount: 99999

Figure11: Edit Book Screen

By clicking **Save** librarian will save altered data to specific book, after successful editing librarian will return to **Books** Screen, if librarian want to cancel the process it can be done by clicking **Cancel** button.

8.1.2 Add Book:

Adding book have the same screen as Book edit but without prepopulated fields. Here librarian can add new book to LIS, new book will be visible for all users and they will be taking this book.

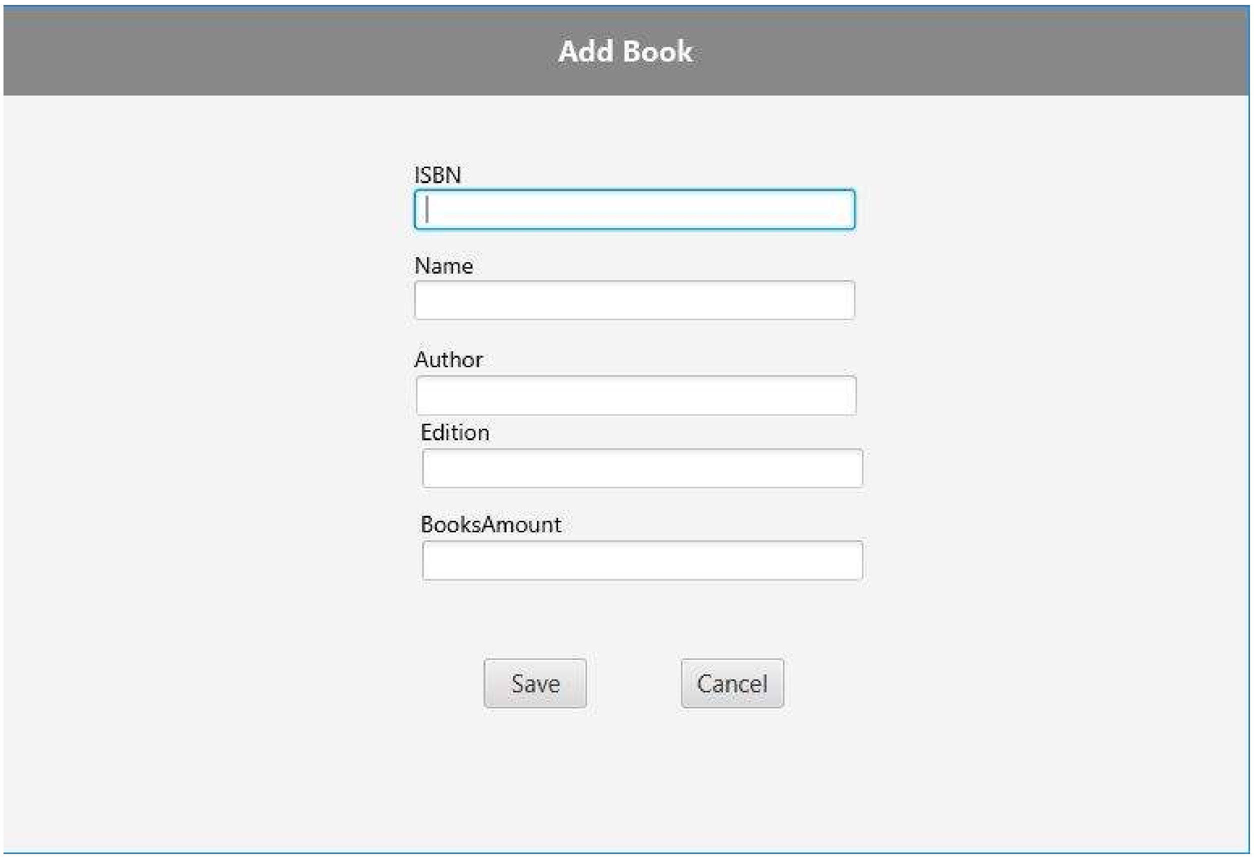
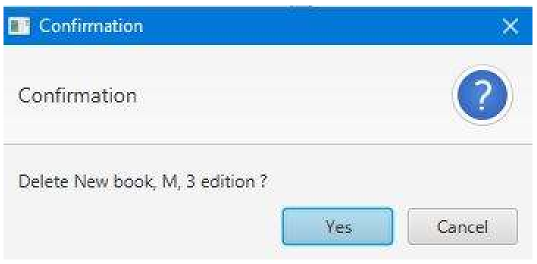


Figure 12: Adding Book to LIS

When adding book none of the fields can be empty or librarian will see error message saying that no fields can be empty. By clicking **Save** librarian will save new book and will return to **Books** screen, but by clicking **Cancel** button librarian can cancel process of adding.

8.1.3 Delete Book:

Librarian can delete book from LIS by selecting specific book and clicking **Delete Book** button, if no books were selected librarian will see alert box that no book was selected, similar to **Edit Book**. Deleted **book will be removed from all users that took this book**, so be careful with this functionality, but before deleting librarian will see confirmation box. Confirmation box is protection from miss clicking, but it cant guarantee full safety.

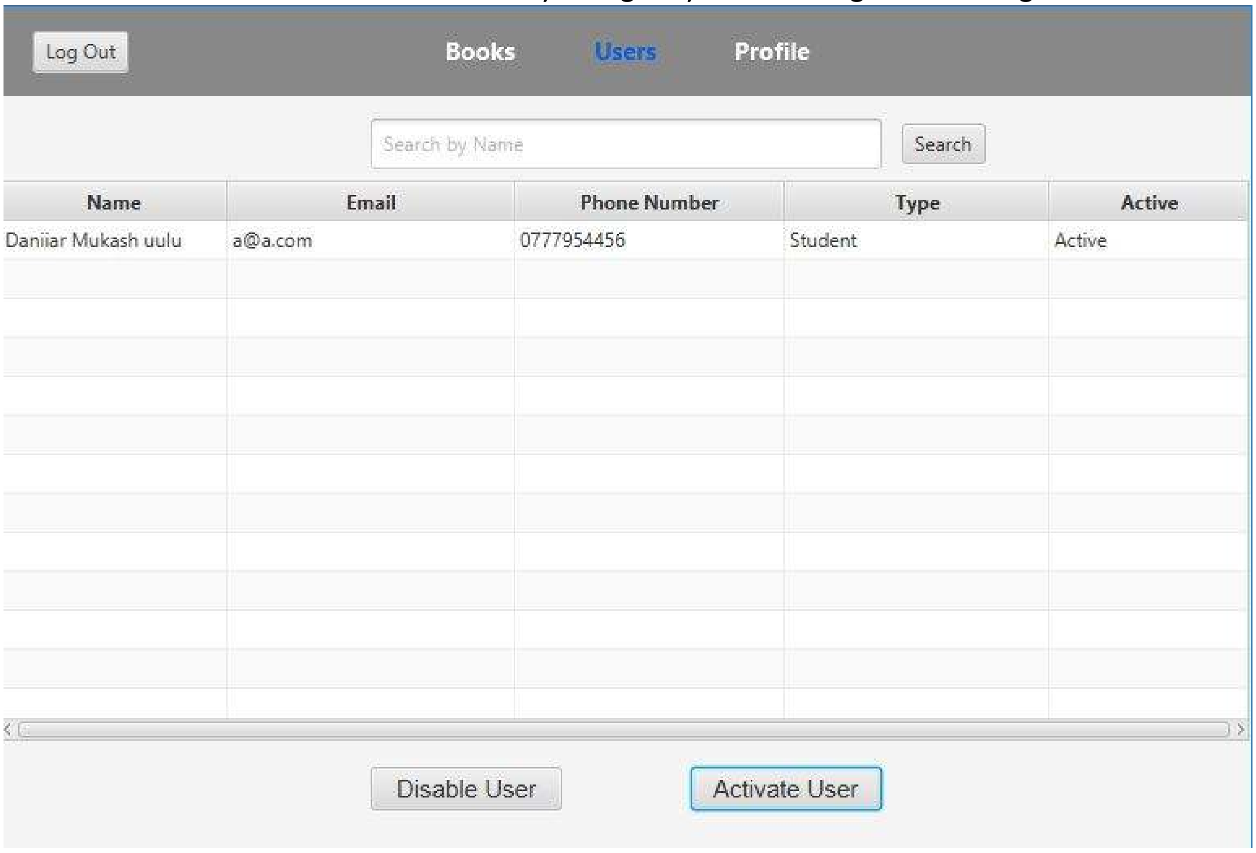


Delete New book, M, 3 edition ?

Figure 13: Confirmation before Book deletion

8.2 Users:

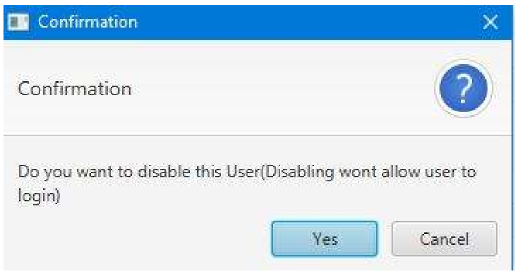
Users screen allow to control user's ability to login by deactivating or activating their accounts



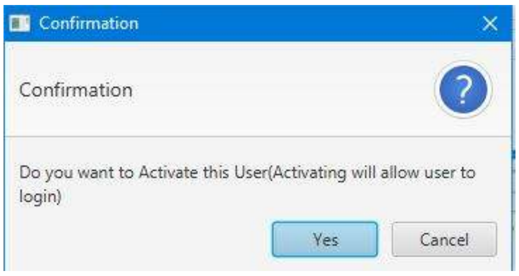
Name	Email	Phone Number	Type	Active
Daniiar Mukash uulu	a@a.com	0777954456	Student	Active

Figure 14: User accounts that registered at LIS

By selecting user and clicking Disable User librarian can ban users from LIS, and by Clicking Activate User librarian can grant users ability to login. Before Disabling and Activating users program will show confirmation box to prevent miss clicking. If user is already active or deactivated librarian should see alert box saying that user is already activated or disabled.



Do you want to disable this User(Disabling wont allow user to login)



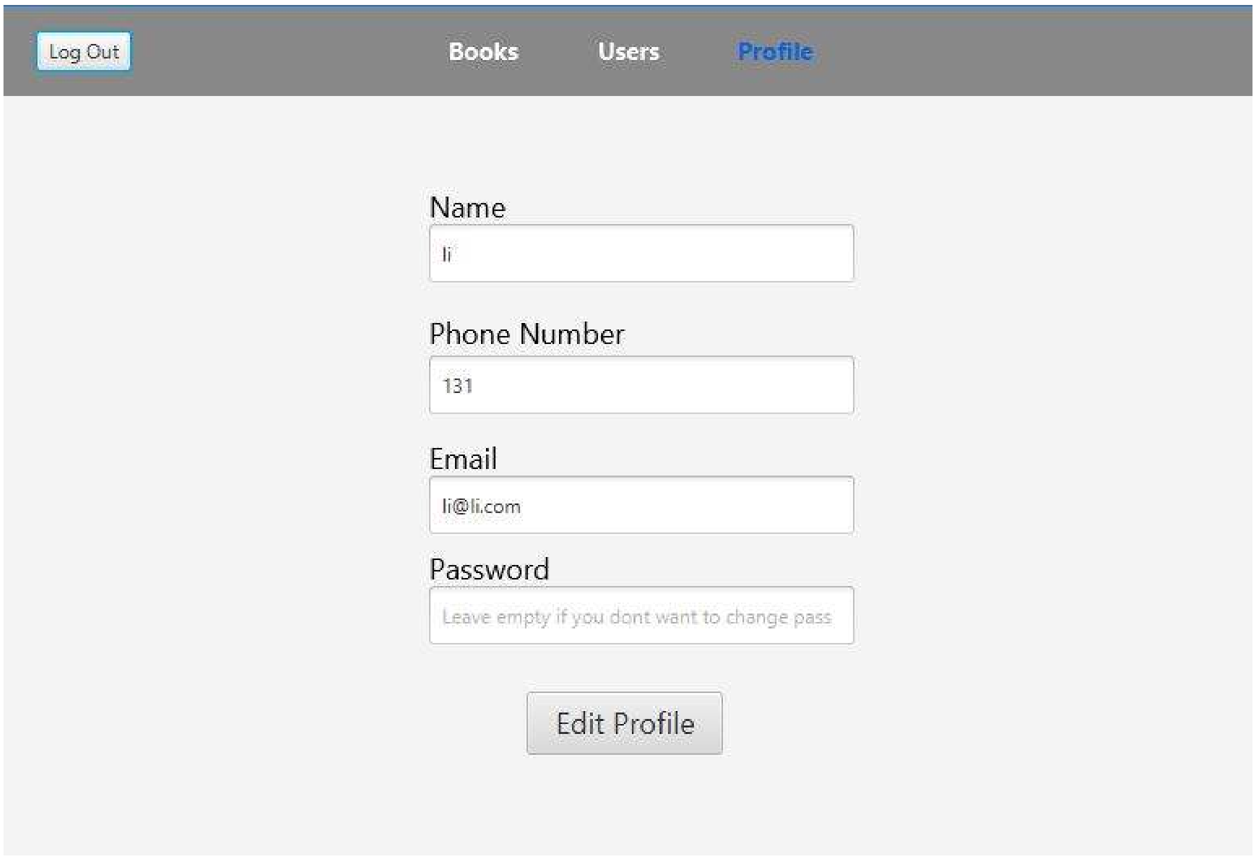
Do you want to Activate this User(Activating will allow user to login)

Figure 15: Deactivate confirmation

Figure 16: Activate confirmation

8.3 Profile:

Profile screen the same as ordinary user screen prepopulated with librarian data. Similarly, none of the fields can be empty except for the password. To save changes in librarian profile press Edit Profile button.



Name

li

Phone Number

131

Email

li@li.com

Password

Leave empty if you dont want to change pass

Figure 16: Librarian Profile screen

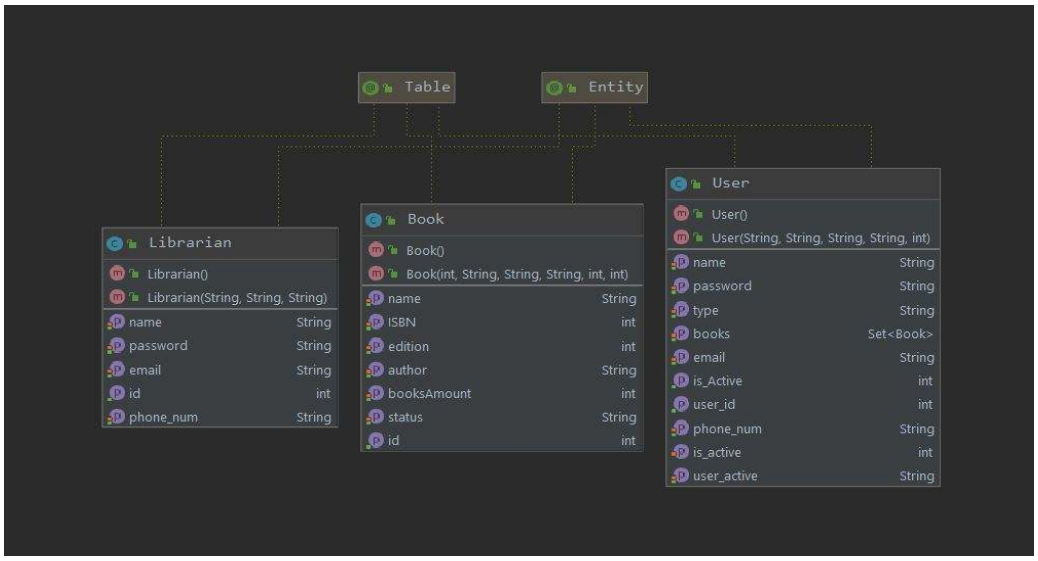
9. Object Oriented Explanation

In object-oriented programming, for example, an object is a self-contained entity that consists of both data and procedures to manipulate the data. In other way, object oriented is the software engineering concept where it is represented using the “OBJECTS”. Below are the objected oriented parts we used in this “Library Management System”:

Database connection:

Connection to the database is made with **Hibernate ORM** which allowed to simplify the development process and database design.

UML diagram:



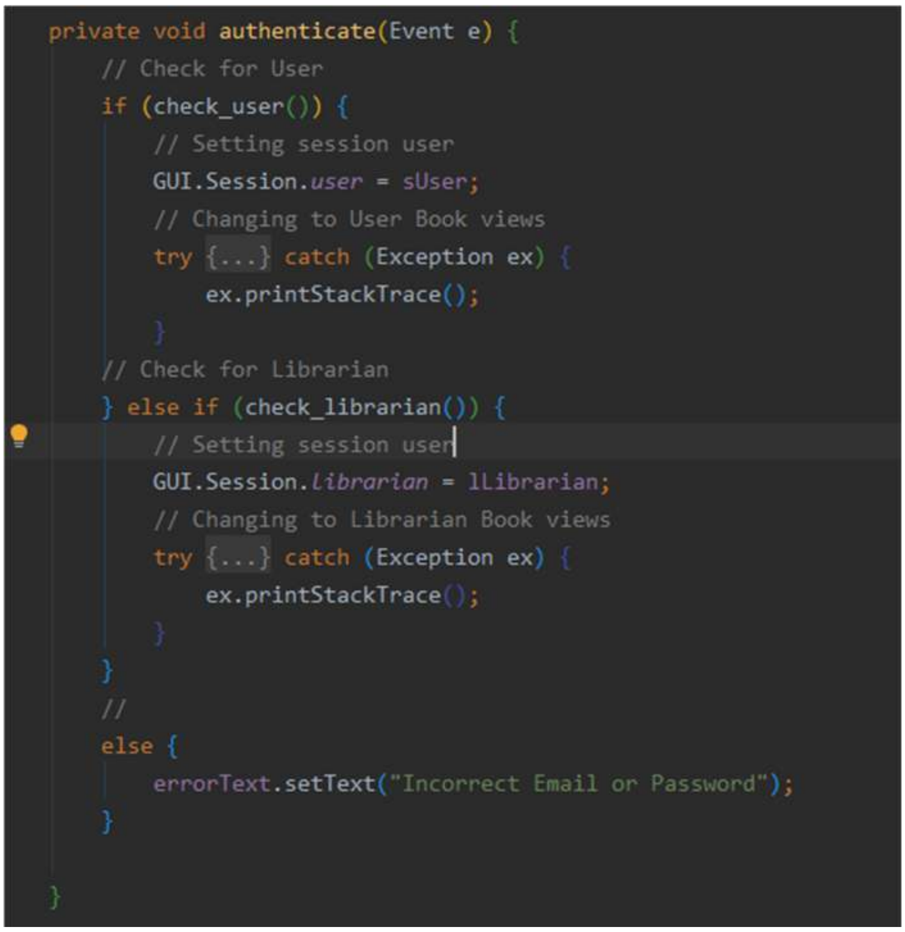
Classes in this diagram represents tables in database, because of ORM every entity of table becomes the instance of that class and its very easy to manage these entities.

Main Classes and Methods:

In library management system every functionality is divided into separate reusable code. To give detailed explanation on each function I want to start from the begining of Execution Procedure or Login Screen and give view of the thing that happening under the hood of Library Management System, this way you will get better understanding how specific functions are related and some nuances that need to covered, which will help for the future users or maintainers of this program

Login:

When user try to login with his credentials how its determined whether user is a librarian or an ordinary user. **Login process** consist of three main methods **authenticate(), check_librarian(), check_user().**

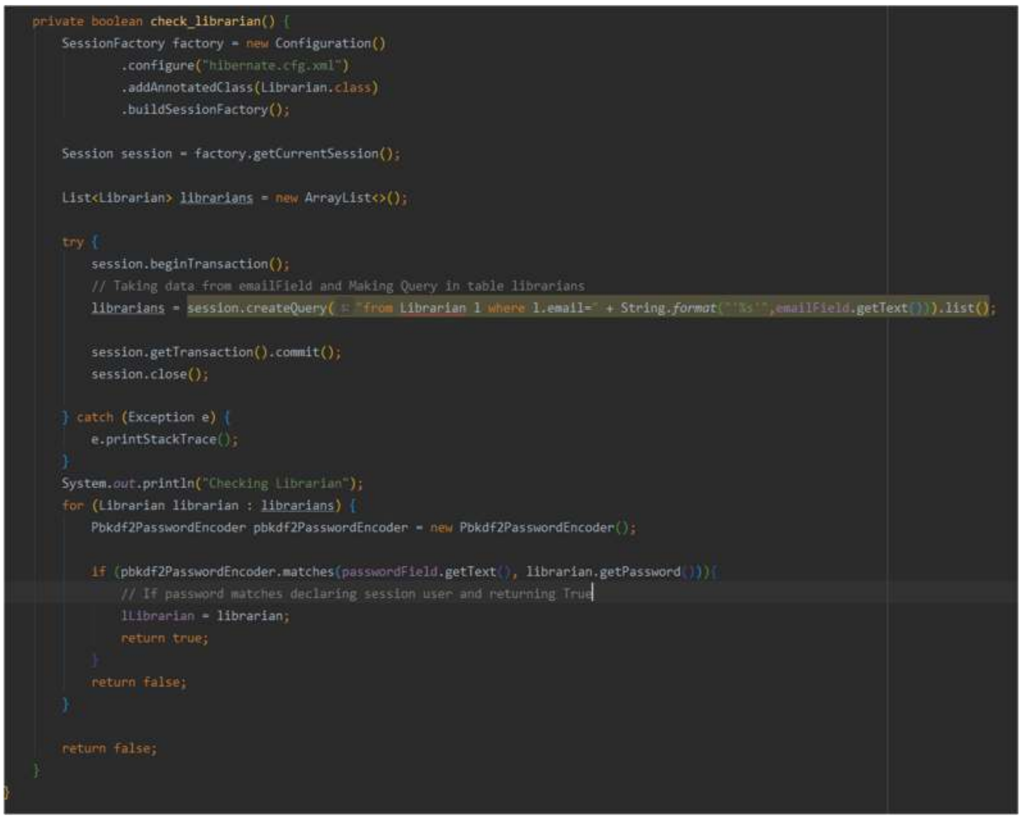


```
private void authenticate(Event e) {
    // Check for User
    if (check_user()) {
        // Setting session user
        GUI.Session.user = sUser;
        // Changing to User Book views
        try {...} catch (Exception ex) {
            ex.printStackTrace();
        }
    // Check for Librarian
    } else if (check_librarian()) {
        // Setting session user
        GUI.Session.Librarian = lLibrarian;
        // Changing to Librarian Book views
        try {...} catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    //
    else {
        errorText.setText("Incorrect Email or Password");
    }

}
```

Authenticate Method

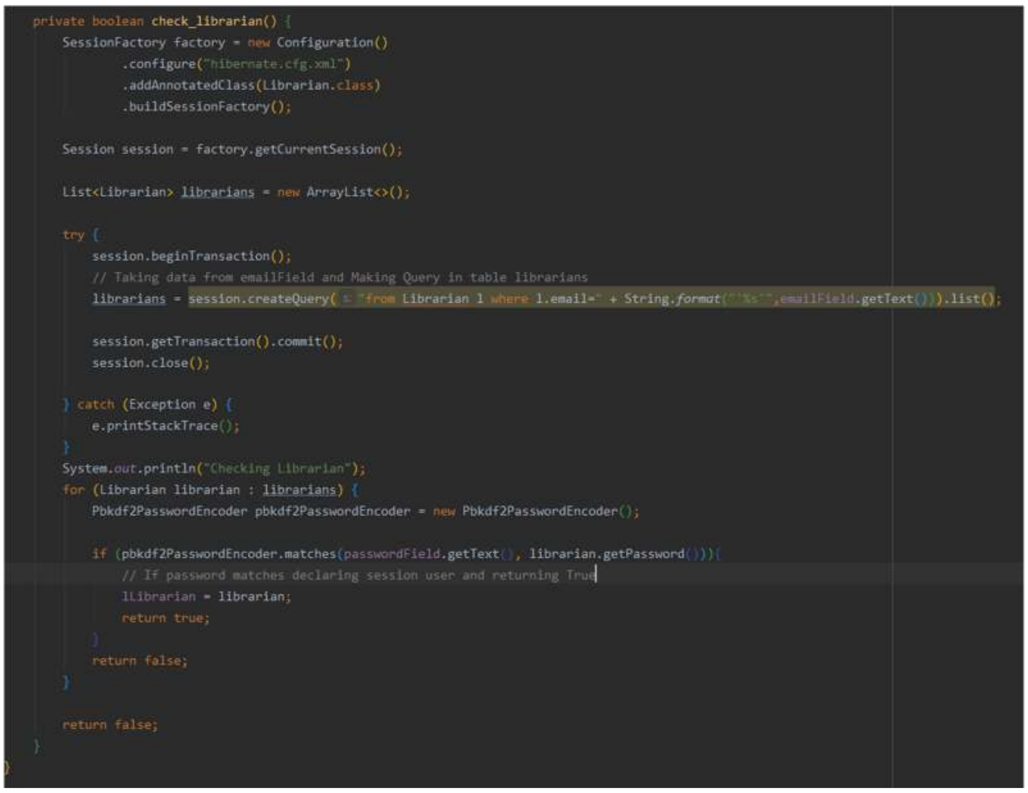
authenticate() executes two methods mentioned above if one them return True then users belongs to that category, after it sets User for this session and changes the view to Books view of each user category, if none of them is matching displays an error.

check_user() and **check_librarian()** are almost the same function but with difference in table that they make query and that **user can be active or inactive**.



```java
private boolean check_librarian() {
    SessionFactory factory = new Configuration()
            .configure("hibernate.cfg.xml")
            .addAnnotatedClass(Librarian.class)
            .buildSessionFactory();

    Session session = factory.getCurrentSession();

    List<Librarian> librarians = new ArrayList<>();

    try {
        session.beginTransaction();
        // Taking data from emailField and Making Query in table librarians
        librarians = session.createQuery( s: "from Librarian l where l.email=" + String.format("'%s'",emailField.getText())).list();

        session.getTransaction().commit();
        session.close();

    } catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("Checking Librarian");
    for (Librarian librarian : librarians) {
        Pbkdf2PasswordEncoder pbkdf2PasswordEncoder = new Pbkdf2PasswordEncoder();

        if (pbkdf2PasswordEncoder.matches(passwordField.getText(), librarian.getPassword())){
            // If password matches declaring session user and returning True
            lLibrarian = librarian;
            return true;
        }
        return false;
    }

    return false;
}
}
```

Check user

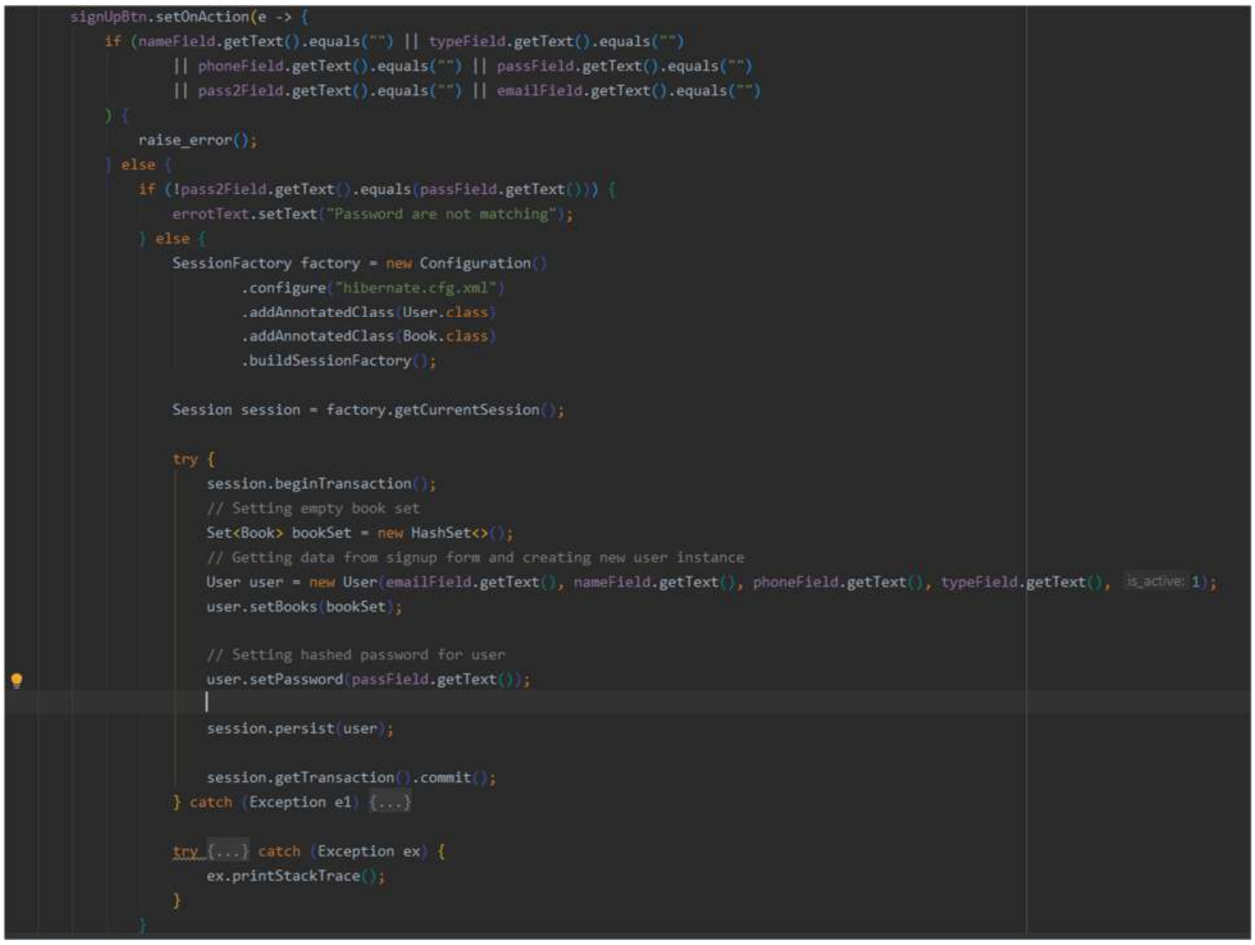


```java
private boolean check_librarian() {
    SessionFactory factory = new Configuration()
            .configure("hibernate.cfg.xml")
            .addAnnotatedClass(Librarian.class)
            .buildSessionFactory();

    Session session = factory.getCurrentSession();

    List<Librarian> librarians = new ArrayList<>();

    try {
        session.beginTransaction();
        // Taking data from emailField and Making Query in table librarians
        librarians = session.createQuery( s: "from Librarian l where l.email=" + String.format("'%s'",emailField.getText())).list();

        session.getTransaction().commit();
        session.close();

    } catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("Checking Librarian");
    for (Librarian librarian : librarians) {
        Pbkdf2PasswordEncoder pbkdf2PasswordEncoder = new Pbkdf2PasswordEncoder();

        if (pbkdf2PasswordEncoder.matches(passwordField.getText(), librarian.getPassword())){
            // If password matches declaring session user and returning True
            lLibrarian = librarian;
            return true;
        }
        return false;
    }

    return false;
}
}
```

Check librarian

Signup:

If user wants to create account in LIS he or she must fill the signup form and submit it. Signup process is bound to **Signup Button.** Every time user clicks it function take data from fields validates it and if every thing is correct saves user to database.

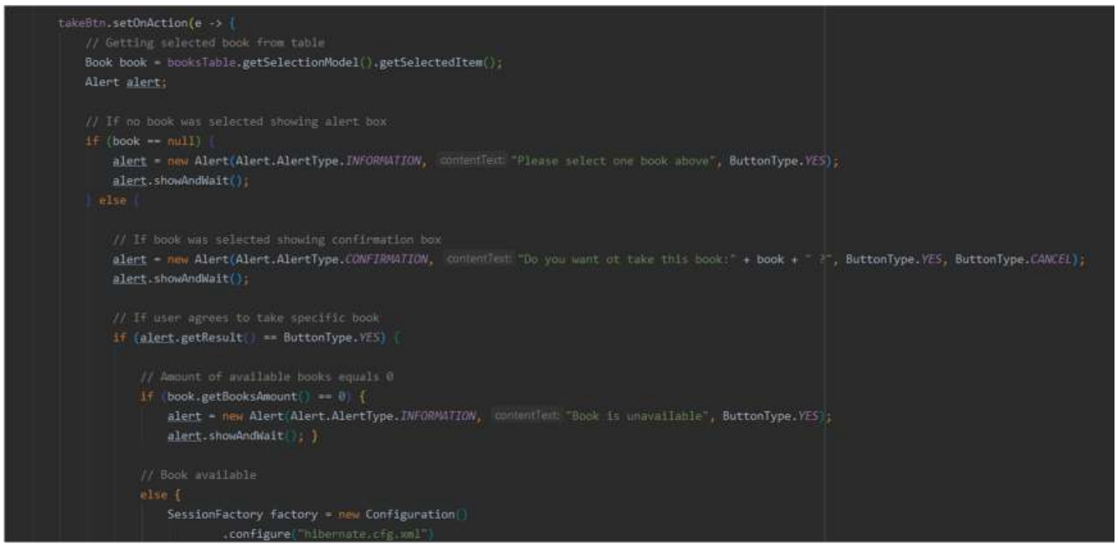


```java
signUpBtn.setOnAction(e -> {
    if (nameField.getText().equals("") || typeField.getText().equals("")
            || phoneField.getText().equals("") || passField.getText().equals("")
            || pass2Field.getText().equals("") || emailField.getText().equals("")
    ) {
        raise_error();
    } else {
        if (!pass2Field.getText().equals(passField.getText())) {
            errotText.setText("Password are not matching");
        } else {
            SessionFactory factory = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .addAnnotatedClass(User.class)
                    .addAnnotatedClass(Book.class)
                    .buildSessionFactory();

            Session session = factory.getCurrentSession();

            try {
                session.beginTransaction();
                // Setting empty book set
                Set<Book> bookSet = new HashSet<>();
                // Getting data from signup form and creating new user instance
                User user = new User(emailField.getText(), nameField.getText(), phoneField.getText(), typeField.getText(), is_active: 1);
                user.setBooks(bookSet);

                // Setting hashed password for user
                user.setPassword(passField.getText());

                session.persist(user);

                session.getTransaction().commit();
            } catch (Exception e1) {...}

            try {...} catch (Exception ex) {
                ex.printStackTrace();
            }
        }
```

Signup on button click

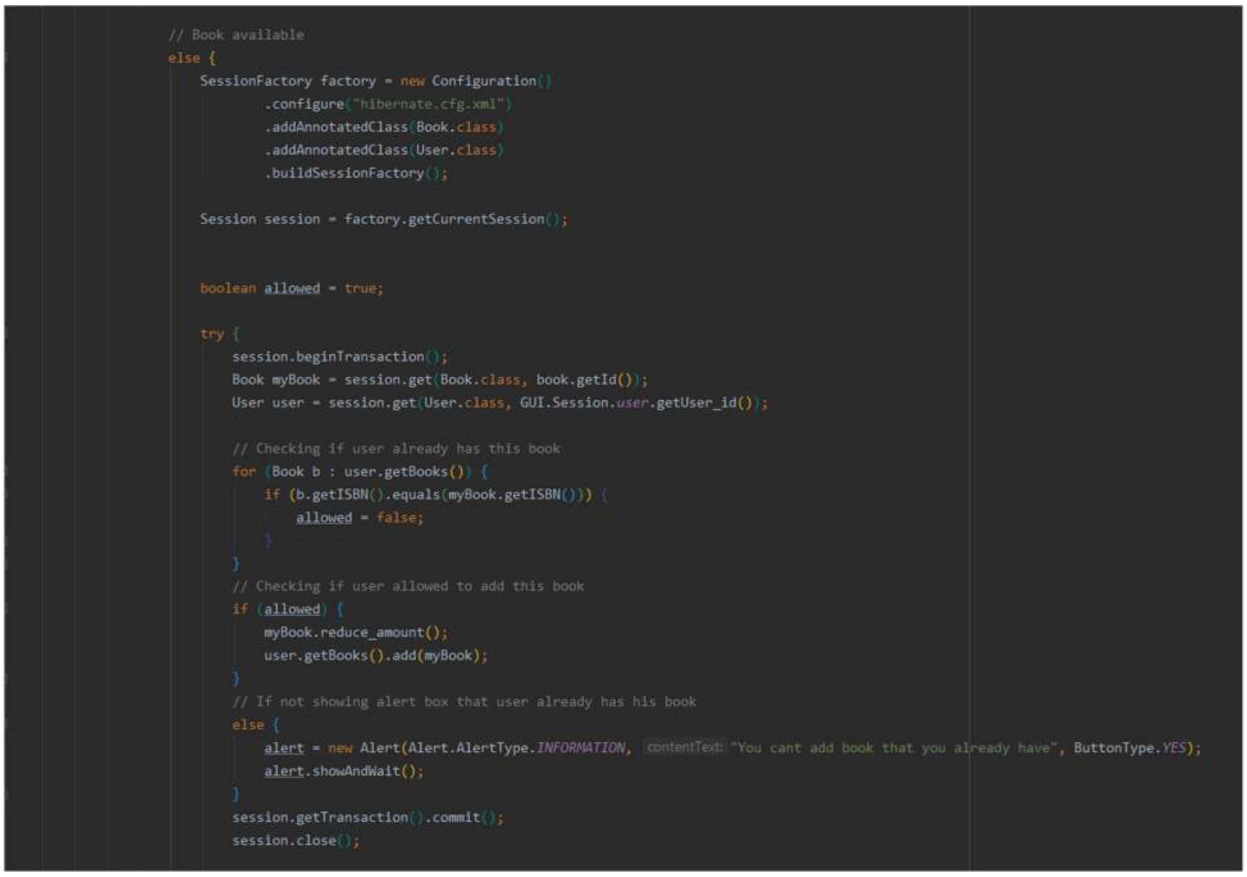
Books:

Books view for both user and librarian are same, there is table view for all available books in LIS and they both can search by their title, but the functionality is different. Let's first look at **user's Book View.**



```java
takeBtn.setOnAction(e -> {
    // Getting selected book from table
    Book book = booksTable.getSelectionModel().getSelectedItem();
    Alert alert;

    // If no book was selected showing alert box
    if (book == null) {
        alert = new Alert(Alert.AlertType.INFORMATION, contentText: "Please select one book above", ButtonType.YES);
        alert.showAndWait();
    } else {

        // If book was selected showing confirmation box
        alert = new Alert(Alert.AlertType.CONFIRMATION, contentText: "Do you want ot take this book:" + book + " ?", ButtonType.YES, ButtonType.CANCEL);
        alert.showAndWait();

        // If user agrees to take specific book
        if (alert.getResult() == ButtonType.YES) {

            // Amount of available books equals 0
            if (book.getBooksAmount() == 0) {
                alert = new Alert(Alert.AlertType.INFORMATION, contentText: "Book is unavailable", ButtonType.YES);
                alert.showAndWait(); }

            // Book available
            else {
                SessionFactory factory = new Configuration()
                        .configure("hibernate.cfg.xml")
```

In the picture above described process of taking book first by checking whether the book is available or not.

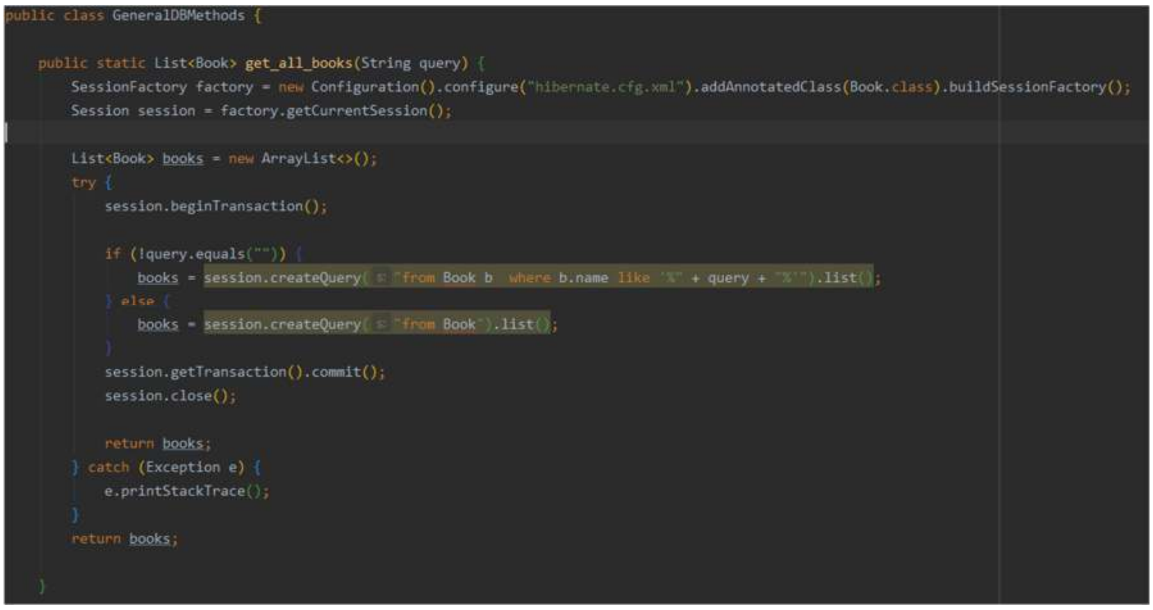


```java
// Book available
else {
    SessionFactory factory = new Configuration()
            .configure("hibernate.cfg.xml")
            .addAnnotatedClass(Book.class)
            .addAnnotatedClass(User.class)
            .buildSessionFactory();

    Session session = factory.getCurrentSession();


    boolean allowed = true;

    try {
        session.beginTransaction();
        Book myBook = session.get(Book.class, book.getId());
        User user = session.get(User.class, GUI.Session.user.getUser_id());

        // Checking if user already has this book
        for (Book b : user.getBooks()) {
            if (b.getISBN().equals(myBook.getISBN())) {
                allowed = false;
            }
        }
        // Checking if user allowed to add this book
        if (allowed) {
            myBook.reduce_amount();
            user.getBooks().add(myBook);
        }
        // If not showing alert box that user already has his book
        else {
            alert = new Alert(Alert.AlertType.INFORMATION,  contentText: "You cant add book that you already have", ButtonType.YES);
            alert.showAndWait();
        }
        session.getTransaction().commit();
        session.close();
```

Adding book to users list

After that verifying that user don't has this book already in his books list.

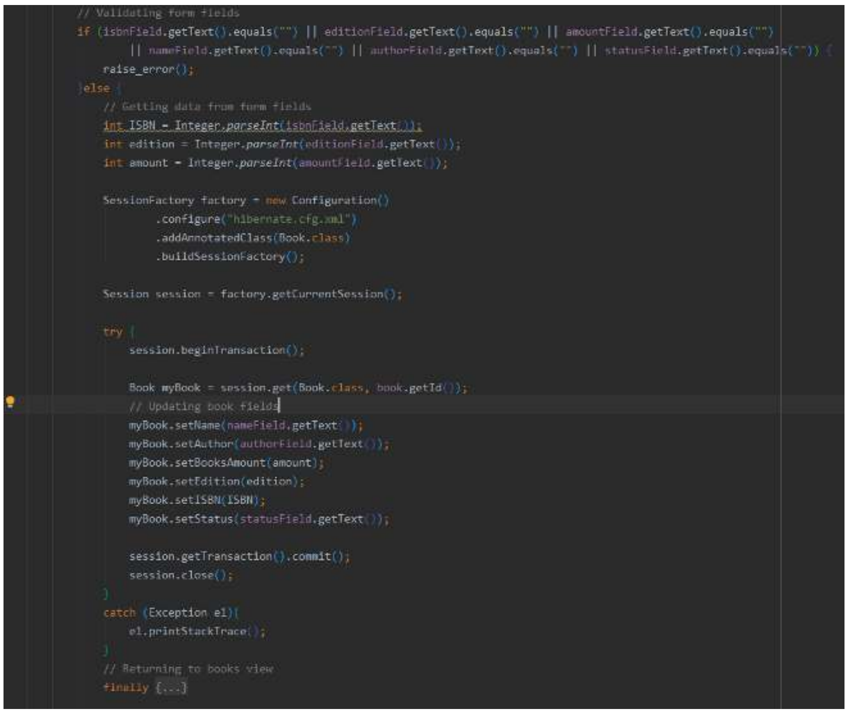
Process of searching is done by **get_all_books()** static method which returns list of all books if passed arguments is empty string or if other string passed searches for book that contains that string.



```java
public class GeneralDBMethods {

    public static List<Book> get_all_books(String query) {
        SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Book.class).buildSessionFactory();
        Session session = factory.getCurrentSession();

        List<Book> books = new ArrayList<>();
        try {
            session.beginTransaction();

            if (!query.equals("")) {
                books = session.createQuery( s: "from Book b  where b.name like '%" + query + "%'").list();
            } else {
                books = session.createQuery( s: "from Book").list();
            }
            session.getTransaction().commit();
            session.close();

            return books;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return books;

    }
```

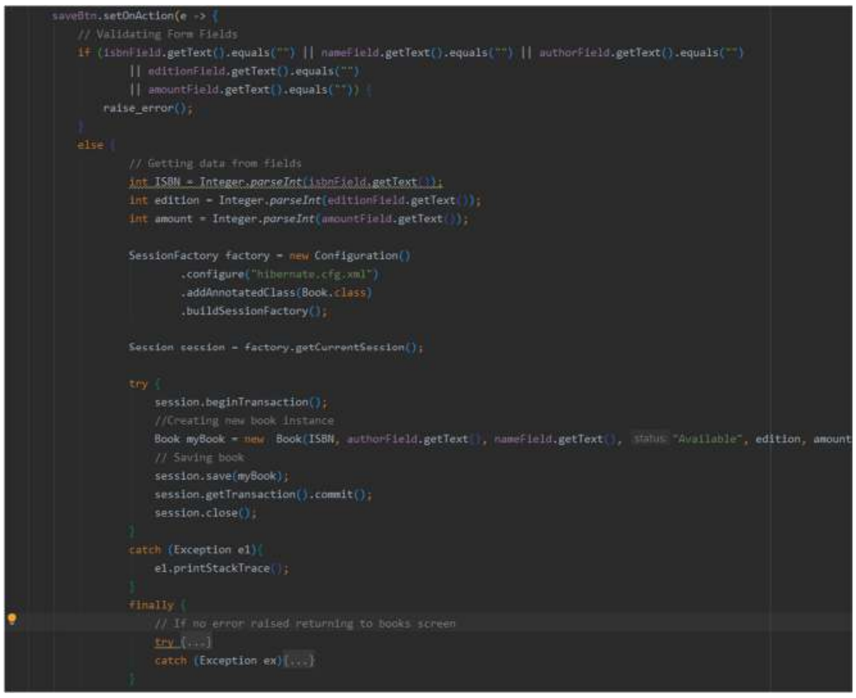
Getting books

But Librarian has 3 function available which is **Edit Book**, **Add Book**, **Delete Book**. Under the hood first two method are very similar to what we saw above. First **Edit Book** is similar to **Take Book** from user view but instead of adding book to users Book List and modifying user instance we just update specific Book instance. Second **Add Book** is similar to Signup process, but here instead of user we create book instance.



```
        // Validating form fields
        if (isbnField.getText().equals("") || editionField.getText().equals("") || amountField.getText().equals("")
                || nameField.getText().equals("") || authorField.getText().equals("") || statusField.getText().equals("")) {
            raise_error();
        }else {
            // Getting data from form fields
            int ISBN = Integer.parseInt(isbnField.getText());
            int edition = Integer.parseInt(editionField.getText());
            int amount = Integer.parseInt(amountField.getText());

            SessionFactory factory = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .addAnnotatedClass(Book.class)
                    .buildSessionFactory();

            Session session = factory.getCurrentSession();

            try {
                session.beginTransaction();

                Book myBook = session.get(Book.class, book.getId());
                // Updating book fields
                myBook.setName(nameField.getText());
                myBook.setAuthor(authorField.getText());
                myBook.setBooksAmount(amount);
                myBook.setEdition(edition);
                myBook.setISBN(ISBN);
                myBook.setStatus(statusField.getText());

                session.getTransaction().commit();
                session.close();
            }
            catch (Exception e1){
                e1.printStackTrace();
            }
            // Returning to books view
            finally {...}
```

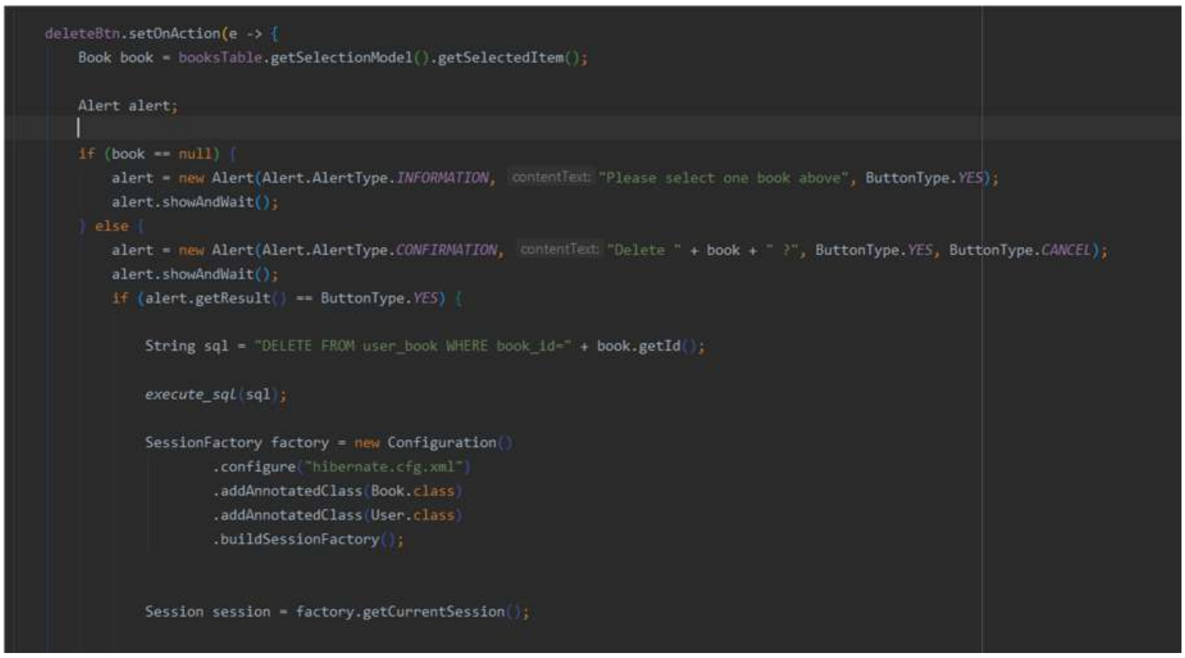
Adding book



```
    saveBtn.setOnAction(e -> {
        // Validating Form Fields
        if (isbnField.getText().equals("") || nameField.getText().equals("") || authorField.getText().equals("")
                || editionField.getText().equals("")
                || amountField.getText().equals("")) {
            raise_error();
        }
        else {
                // Getting data from fields
                int ISBN = Integer.parseInt(isbnField.getText());
                int edition = Integer.parseInt(editionField.getText());
                int amount = Integer.parseInt(amountField.getText());

                SessionFactory factory = new Configuration()
                        .configure("hibernate.cfg.xml")
                        .addAnnotatedClass(Book.class)
                        .buildSessionFactory();

                Session session = factory.getCurrentSession();

                try {
                    session.beginTransaction();
                    //Creating new book instance
                    Book myBook = new  Book(ISBN, authorField.getText(), nameField.getText(), status: "Available", edition, amount
                    // Saving book
                    session.save(myBook);
                    session.getTransaction().commit();
                    session.close();
                }
                catch (Exception e1){
                    e1.printStackTrace();
                }
                finally {
                    // If no error raised returning to books screen
                    try {...}
                    catch (Exception ex){...}
                }
```

Editing book

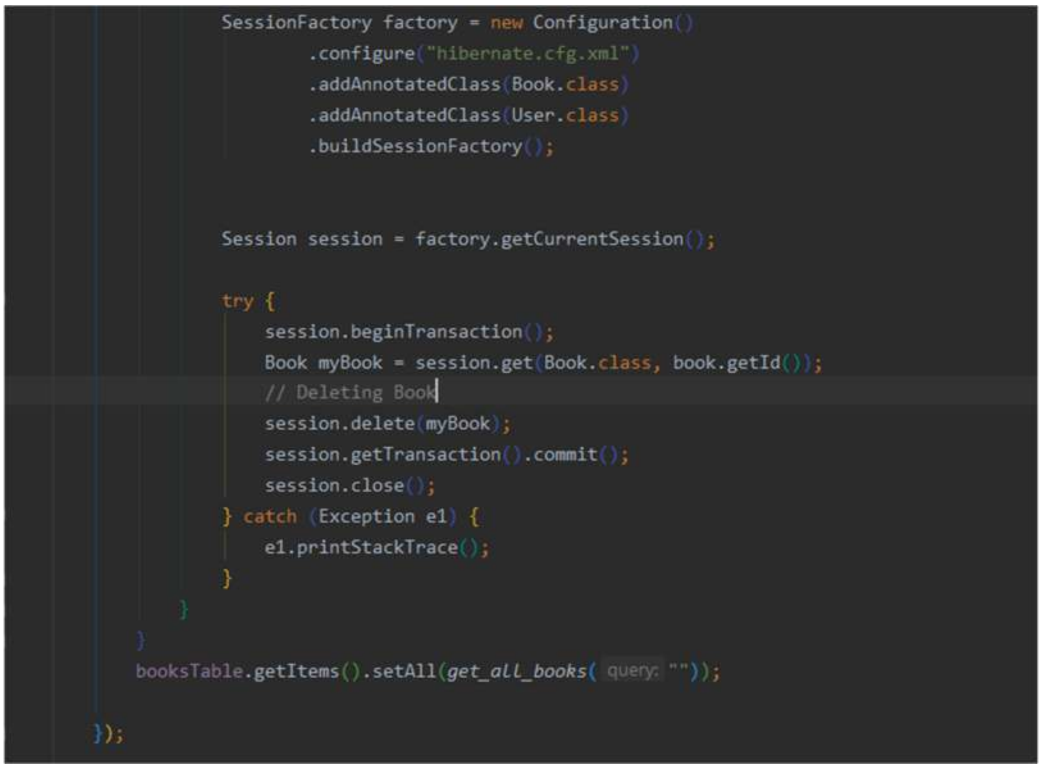
But with Delete Book situation is a bit different. When librarian deletes book, it may be linked to users by many to many relations, so database won't allow to delete until they have relation, also there is no Entity class in my program that will allow to access the database with Hibernate ORM. To overcome this problem program run custom **sql** with **execute_sql** method that deletes relation between users and specific book is it exists.



```
deleteBtn.setOnAction(e -> {
    Book book = booksTable.getSelectionModel().getSelectedItem();

    Alert alert;

    if (book == null) {
        alert = new Alert(Alert.AlertType.INFORMATION,  contentText: "Please select one book above", ButtonType.YES);
        alert.showAndWait();
    } else {
        alert = new Alert(Alert.AlertType.CONFIRMATION,  contentText: "Delete " + book + " ?", ButtonType.YES, ButtonType.CANCEL);
        alert.showAndWait();
        if (alert.getResult() == ButtonType.YES) {

            String sql = "DELETE FROM user_book WHERE book_id=" + book.getId();

            execute_sql(sql);

            SessionFactory factory = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .addAnnotatedClass(Book.class)
                    .addAnnotatedClass(User.class)
                    .buildSessionFactory();


            Session session = factory.getCurrentSession();
```

As always showing the confirmation box

After deleting all relations, we can easily delete book from database.

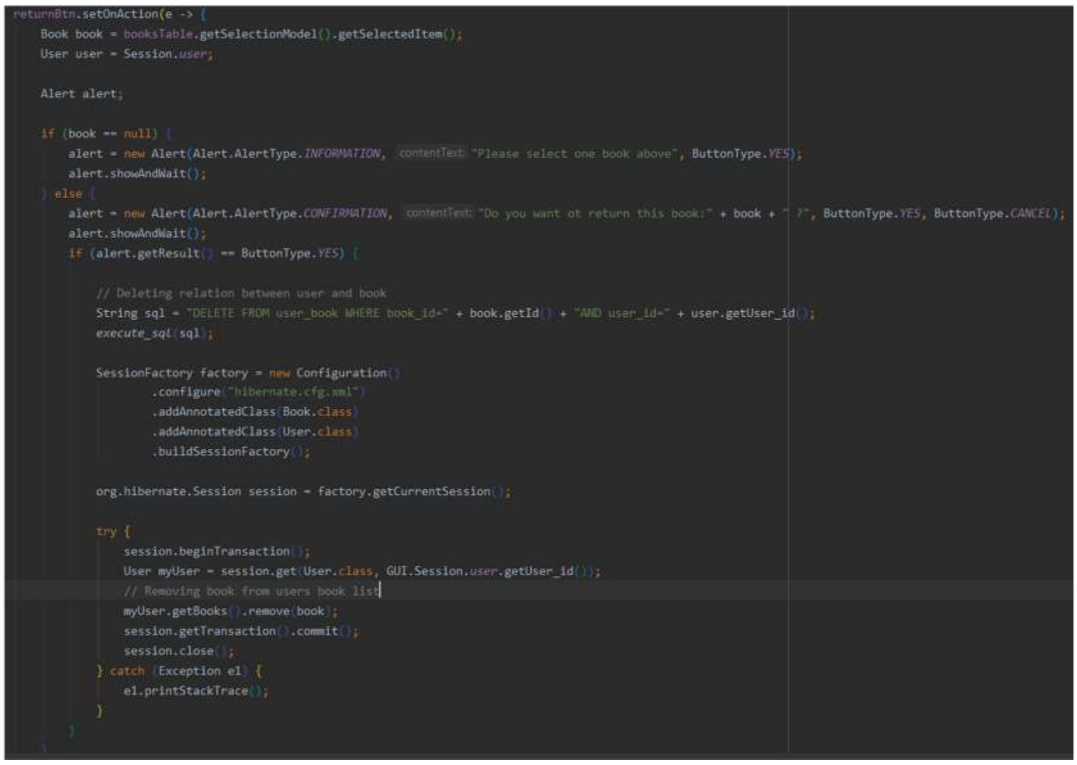


```
            SessionFactory factory = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .addAnnotatedClass(Book.class)
                    .addAnnotatedClass(User.class)
                    .buildSessionFactory();


            Session session = factory.getCurrentSession();

            try {
                session.beginTransaction();
                Book myBook = session.get(Book.class, book.getId());
                // Deleting Book
                session.delete(myBook);
                session.getTransaction().commit();
                session.close();
            } catch (Exception e1) {
                e1.printStackTrace();
            }
        }
    }
    booksTable.getItems().setAll(get_all_books( query: ""));

});
```

Deleting Book

Users Books:

This view allows to user see all the books that they took from library and if they willing to return them back. Process of returning is similar to Adding Book, but now we just remove book from relational table and update user instance.

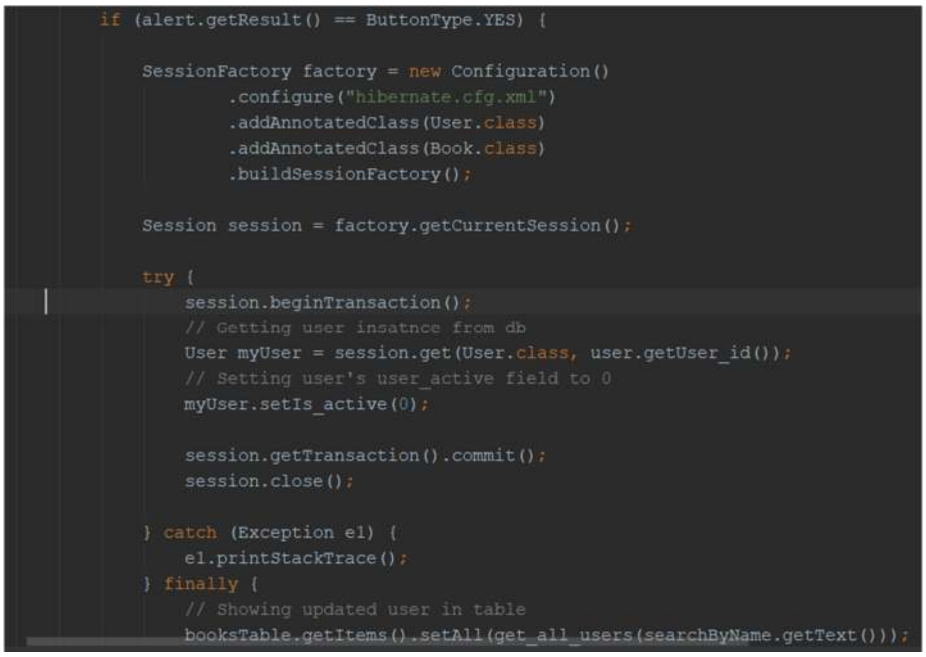


```java
returnBtn.setOnAction(e -> {
    Book book = booksTable.getSelectionModel().getSelectedItem();
    User user = Session.user;

    Alert alert;

    if (book == null) {
        alert = new Alert(Alert.AlertType.INFORMATION, contentText: "Please select one book above", ButtonType.YES);
        alert.showAndWait();
    } else {
        alert = new Alert(Alert.AlertType.CONFIRMATION, contentText: "Do you want ot return this book:" + book + " ?", ButtonType.YES, ButtonType.CANCEL);
        alert.showAndWait();
        if (alert.getResult() == ButtonType.YES) {

            // Deleting relation between user and book
            String sql = "DELETE FROM user_book WHERE book_id=" + book.getId() + "AND user_id=" + user.getUser_id();
            execute_sql(sql);

            SessionFactory factory = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .addAnnotatedClass(Book.class)
                    .addAnnotatedClass(User.class)
                    .buildSessionFactory();

            org.hibernate.Session session = factory.getCurrentSession();

            try {
                session.beginTransaction();
                User myUser = session.get(User.class, GUI.Session.user.getUser_id());
                // Removing book from users book list
                myUser.getBooks().remove(book);
                session.getTransaction().commit();
                session.close();
            } catch (Exception e1) {
                e1.printStackTrace();
            }
        }
    }
```

Returning Book back to library

Users:

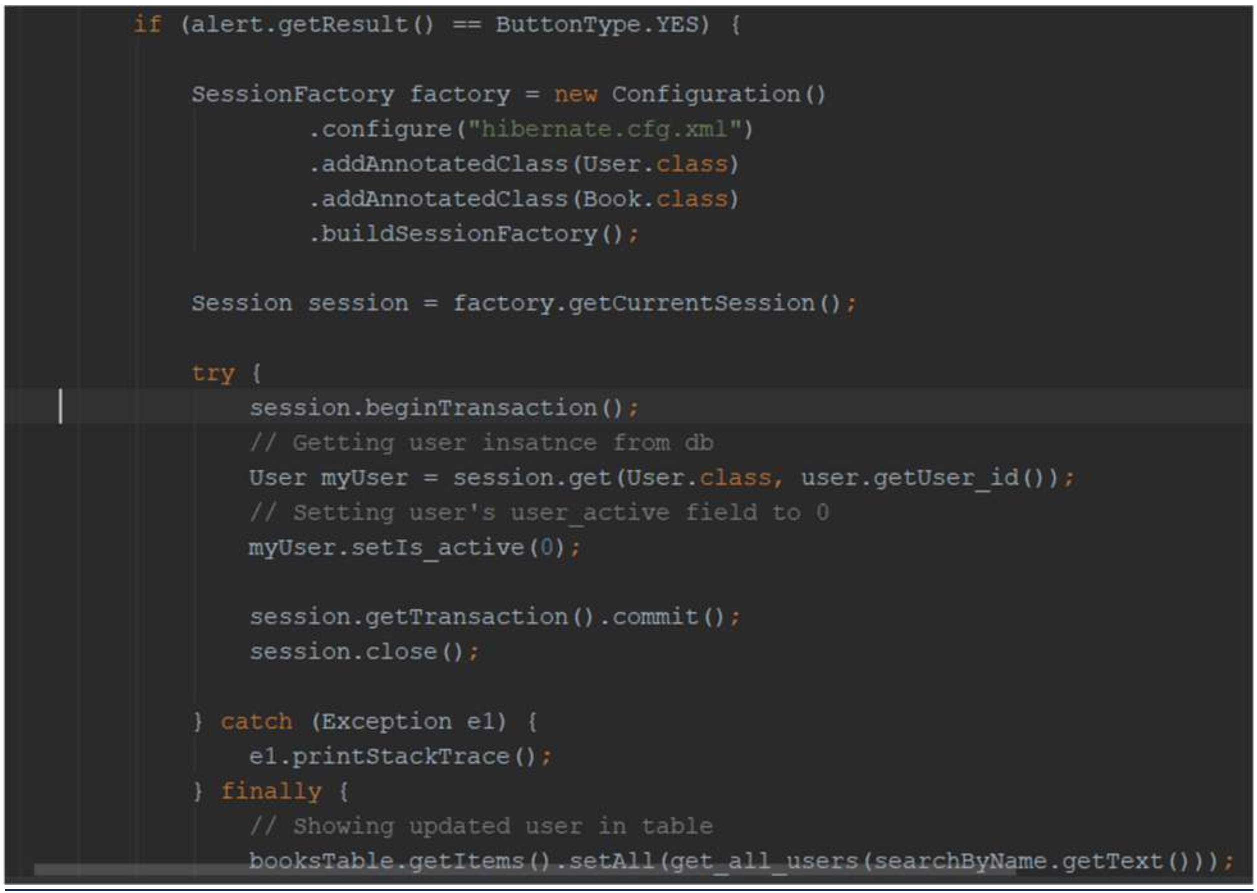
Users view available for librarian gives ability to control whether user can login or not, by updating user_active field in selected user to 0 or 1, because login view checks not only password but whether user is active or not.


```java
    if (alert.getResult() == ButtonType.YES) {

        SessionFactory factory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(User.class)
                .addAnnotatedClass(Book.class)
                .buildSessionFactory();

        Session session = factory.getCurrentSession();

        try {
            session.beginTransaction();
            // Getting user insatnce from db
            User myUser = session.get(User.class, user.getUser_id());
            // Setting user's user_active field to 0
            myUser.setIs_active(0);

            session.getTransaction().commit();
            session.close();

        } catch (Exception e1) {
            e1.printStackTrace();
        } finally {
            // Showing updated user in table
            booksTable.getItems().setAll(get_all_users(searchByName.getText()));
```

Changing User's user_active field to 0

Profile:

Profile views for both User and Librarian are exact copy of each other except for the fields that each view contains. As always similar process goes here we query user from data base validate fields and then update existing user with new data.


```java
    if (alert.getResult() == ButtonType.YES) {

        SessionFactory factory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(User.class)
                .addAnnotatedClass(Book.class)
                .buildSessionFactory();

        Session session = factory.getCurrentSession();

        try {
            session.beginTransaction();
            // Getting user insatnce from db
            User myUser = session.get(User.class, user.getUser_id());
            // Setting user's user_active field to 0
            myUser.setIs_active(0);

            session.getTransaction().commit();
            session.close();

        } catch (Exception e1) {
            e1.printStackTrace();
        } finally {
            // Showing updated user in table
            booksTable.getItems().setAll(get_all_users(searchByName.getText()));
```

Updating user instances

Assumption

Even this program covers most of user management it can’t give GUI for creating librarian accounts, but to solve this problem we have librarian driver. By running this driver program creates initial migration and create new librarian account.