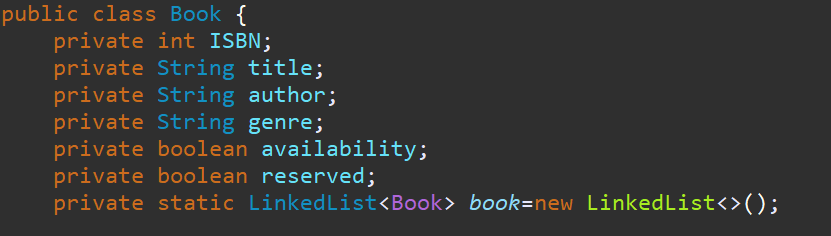
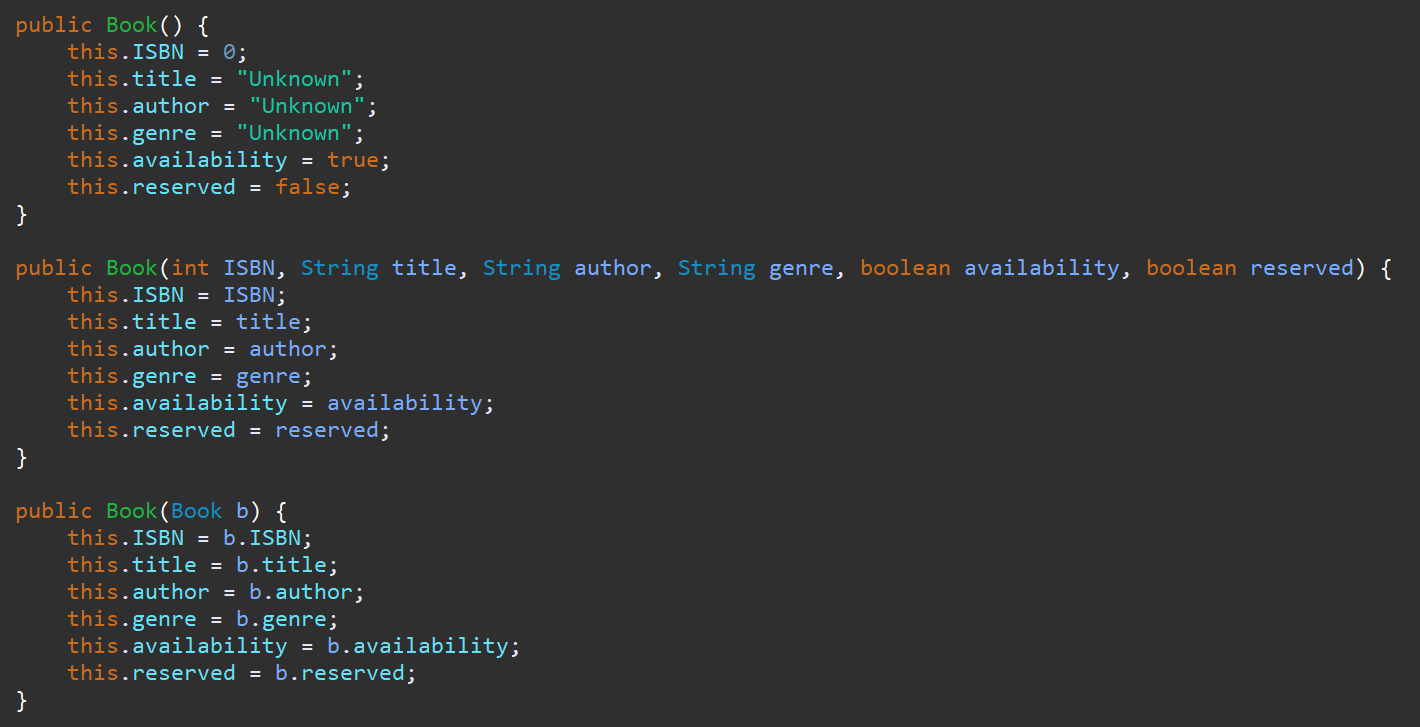
**Java Final Project**

*Documentation for the Book class*

The Book class is a normal class with 6 private parameters, which are: int ISBN (the International Standard Book Number), String title (title of the book), String author (The name of the author), String genre (the genre of the book), Boolean availability (the availability of the book), Boolean reserved (if the book is reserved), and a private LinkedList<Book> book that acts as a data base to store the books.



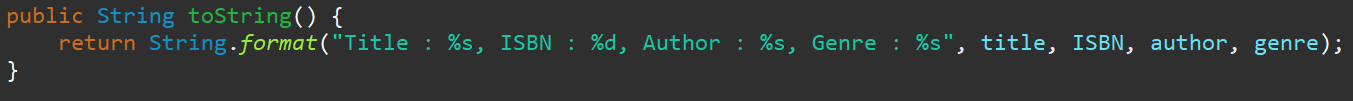
The class consist of three constructors, a default constructor (with no parameters entered), a normal constructor (with parameters), and a copy constructor (with an object of type Book entered), to declare a new object of type Book.





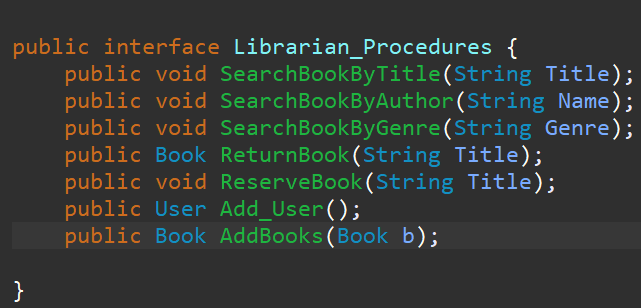
And the class also consists of getters and setters for the parameters. The getters help to get a parameter in the subclasses because the attributes are private and the setters help to change or set the parameters.

There is also a toString() method that prints the information of the book.



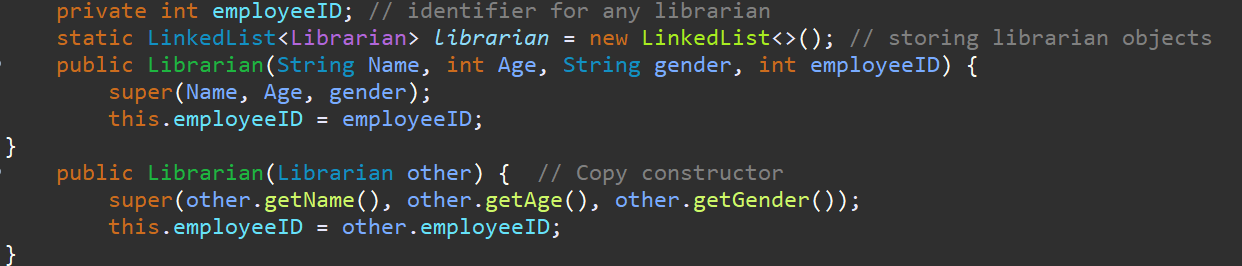
*Documentation for the Librarian\_Procedures interface*

The Librarian\_Procedures is an interface, whose methods are implemented by the Librarian class. There are several methods in the interface, such as SearchBookByTitle(String Title) (search for a book by its title), SearchBookByAuthor(String Name) (search for a book by its author’s name), SearchBookByGenre(String Genre) (search for a book by its genre), Add\_User() (add a new user to the system), AddBooks(Book b) (add a new book to the library), ReturnBook(String Title) (returns the borrowed book), and ReserveBook(String Title) (reserve a book).

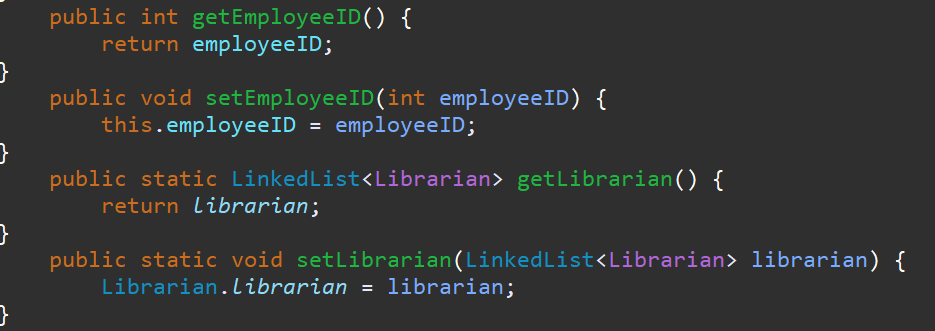


*Documentation for the Librarian class*

The Librarian class is a class that inherits the Person abstract class, and implements the Librarian\_Procedures interface. It consists of 1 private attribute, the int employeeID (the ID number of the employee which is the librarian), and a private LinkedList<Librarian> librarian which acts as a database to store the librarians. It has 2 constructors, a normal one and a copy constructor.

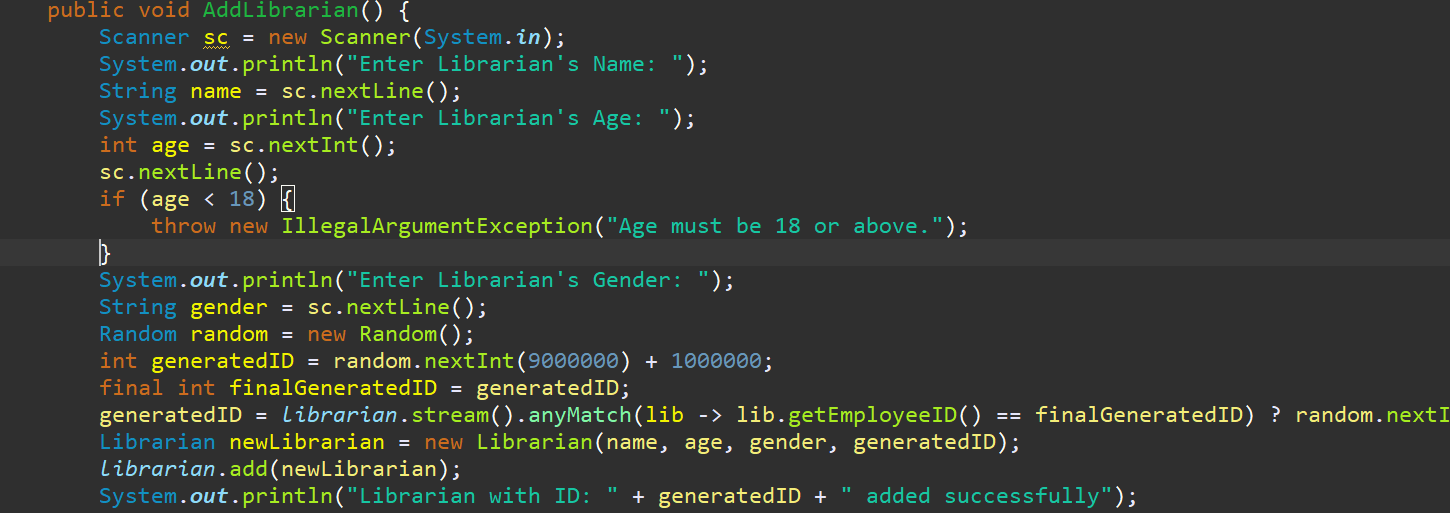


It also contains getters and setters for the parameters.

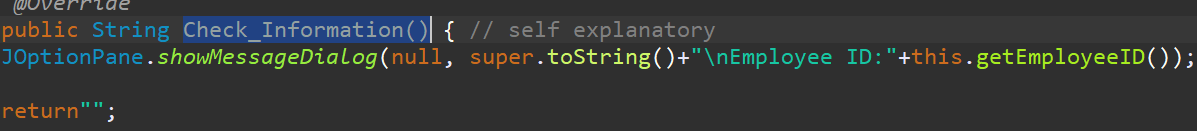


In this class we can find 1 concrete method AddLibrarian(), 1 method implementation for the inherited abstract method Check\_Information(), and 5 implementations for the methods declared in the interface Librarian\_Procedures:

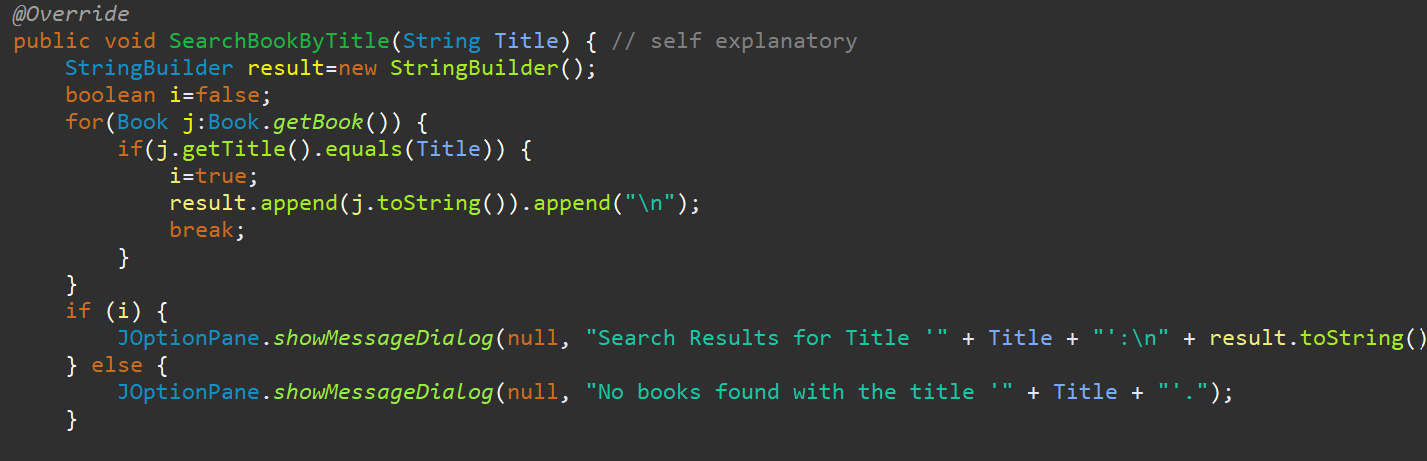
* AddLibrarian() method asks the user to enter his/her name, gender, and age. It also generates a random number, while checking if the number was already given before for one of the librarians in the LinkedList.



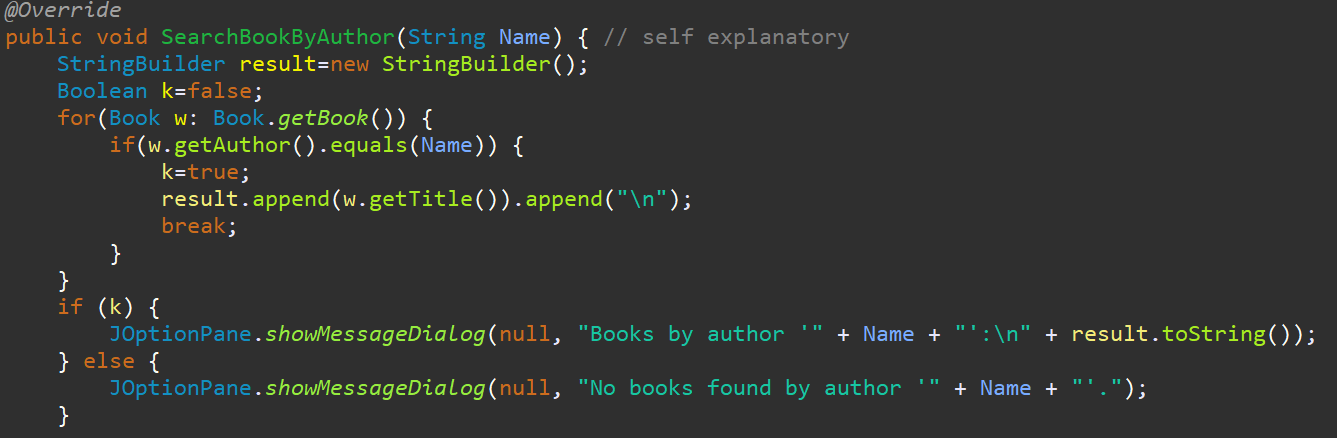
* Check\_Information() method is overridden. It allows the librarian to check his/her information, by printing his/her name, age, gender, and employeeID.



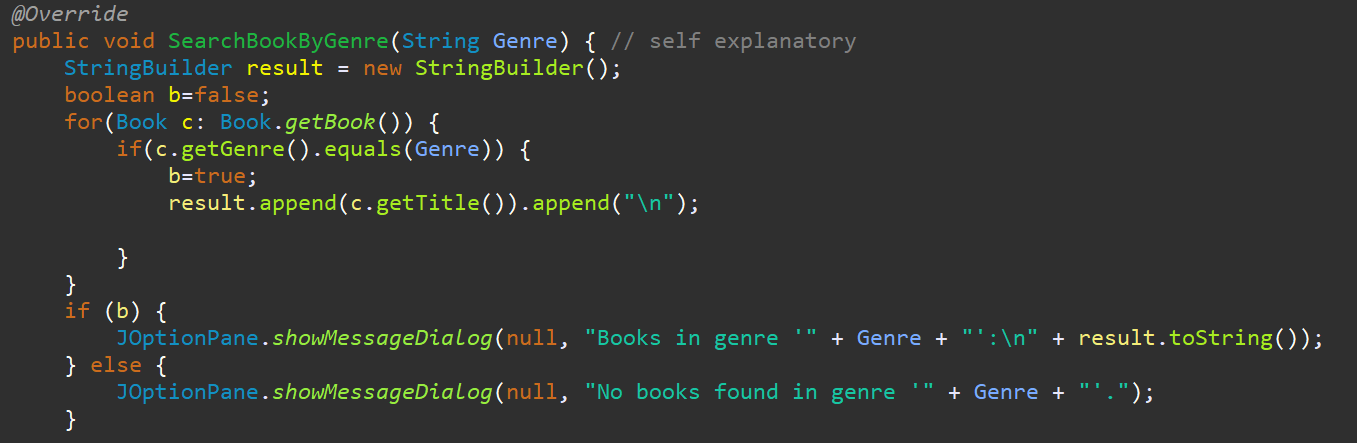
* SearchBookByTitle(String Title) method is to search for a book given a title.



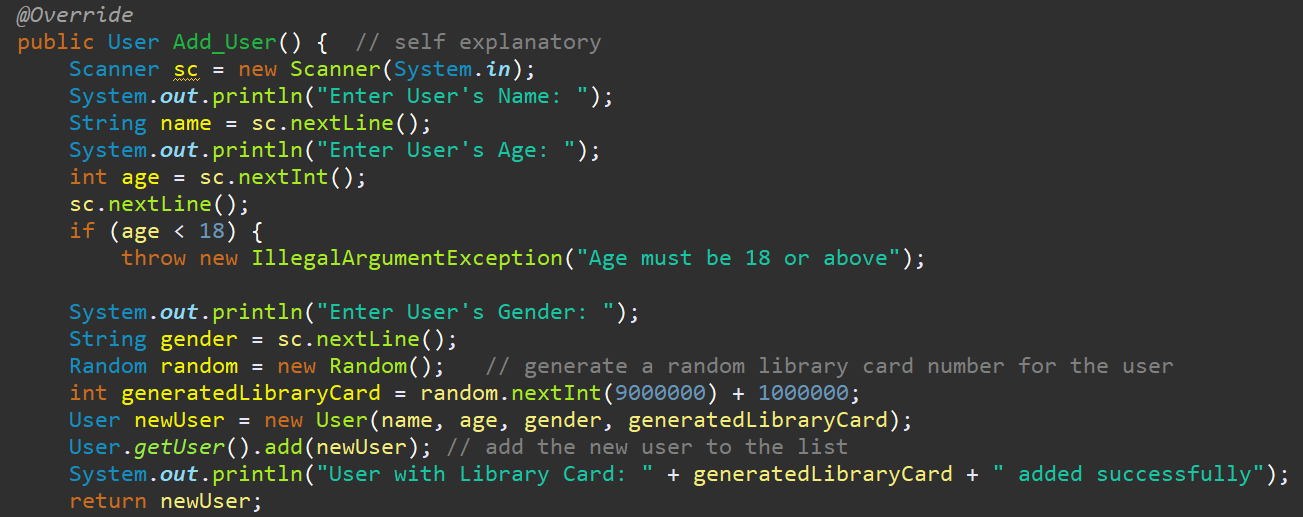
* SearchBookByAuthor(String Name) method is to search for the books that belong to the given author.



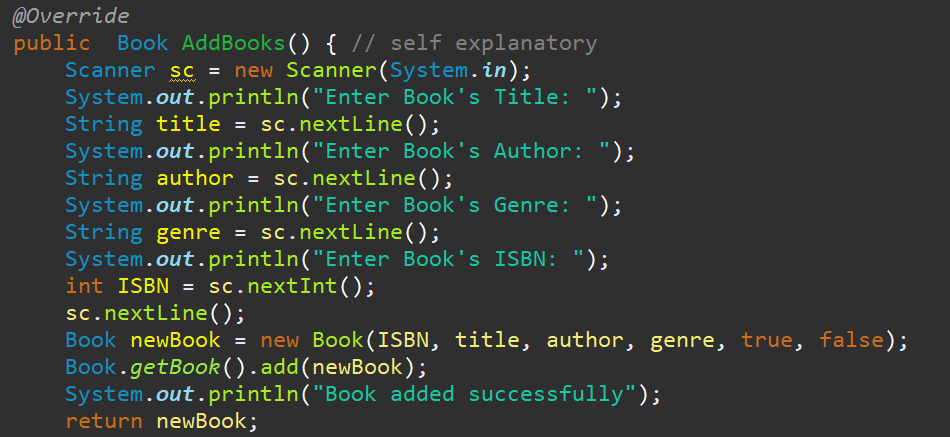
* SearchBookByGenre(String Genre) method is to search for the books belonging to the given genre .



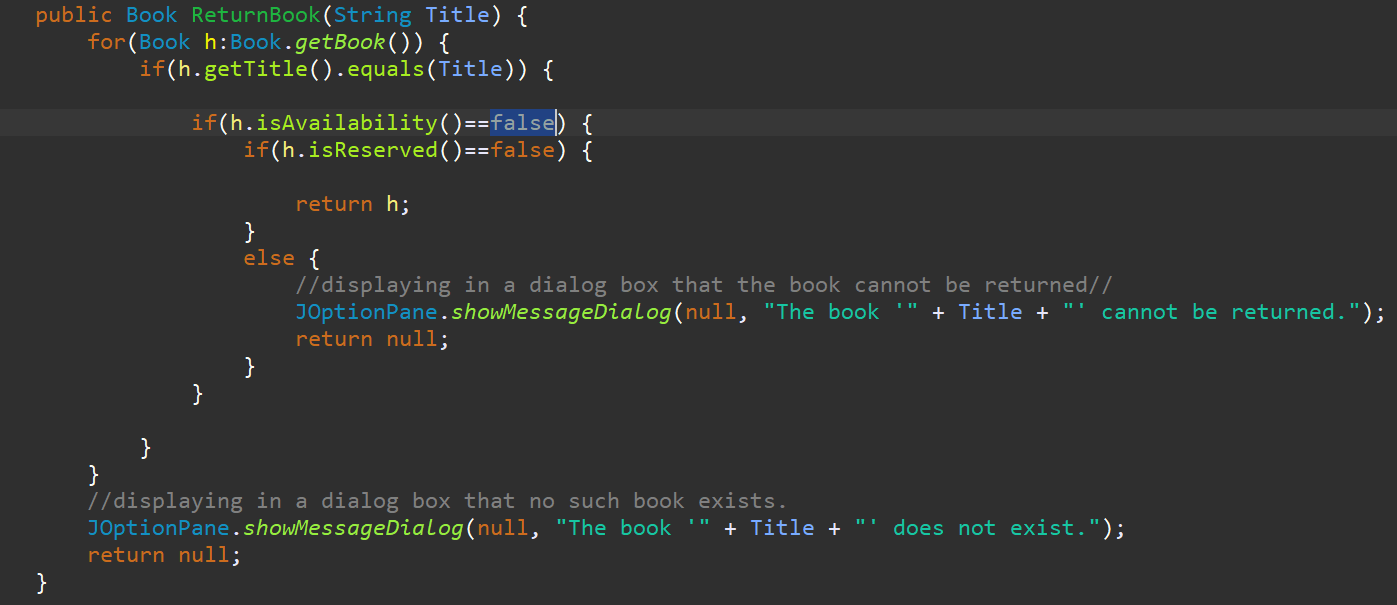
* Add\_User() method adds a new user to the system by asking the name, age, and the gender of the user. For the library card number, it generates it randomly and makes sure that no two users possess the same card number.

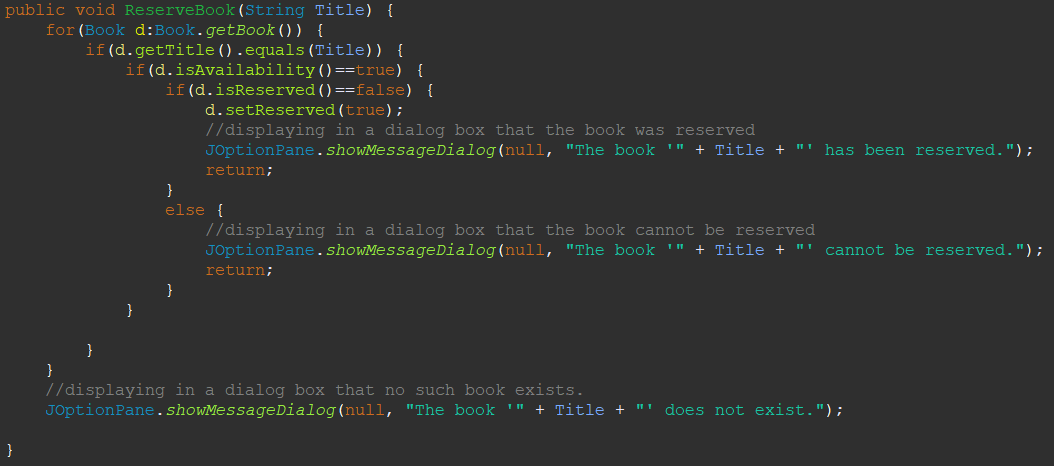


* AddBooks() method adds a new book to the system, by asking the title, author’s name, genre, and ISBN.



* ReturnBook(String Title) searches a book in the LinkedList based on its title. If a book is located and it’s availability and reserve status are false, the program will return the book. If not, the program will inform the user that “The book ‘Title’ cannot be returned. In case none of the books have the title, the program will print “The book ‘Title’ does not exist” message .

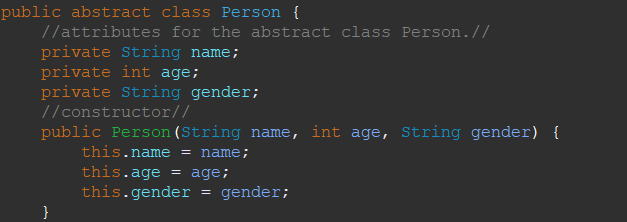


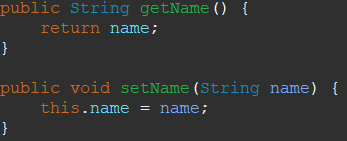
* ReserveBook(String Title) searches for a book by its title. If it is located, available and not reserved, the program will set the reserve attribute of the book to true and will inform the user that “the book ‘title’ has been reserved.”. If not, the program will print “The book ‘Title’ cannot be reserved”. In case no books were titled as the given parameter, the program will print “The book ‘Title’ does not exist” message.
* 

*Documentation for the code responsible for creating and implementing the functionalities of the users:*

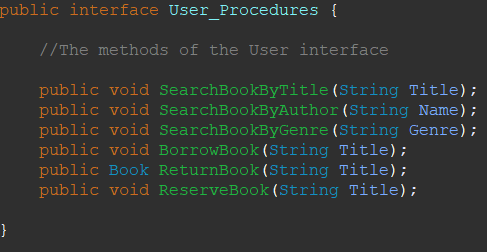
First of all, users within this library application have the following characteristics stored and saved: Name, age, gender, and a library card number. Since the first three characteristics are common to everyone, regardless of their status as a user or a librarian, a general abstract class named person has been created to be extended by the user and the librarian classes.

The abstract person class:

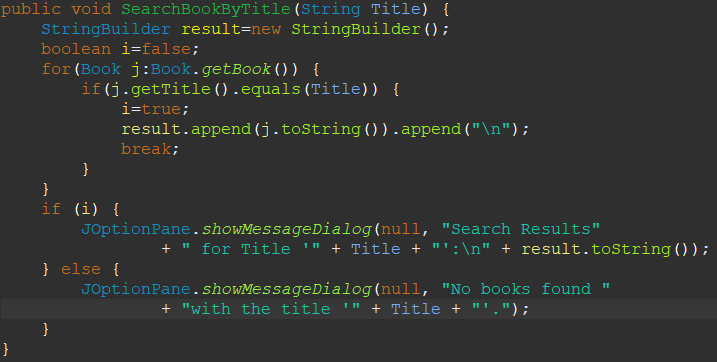


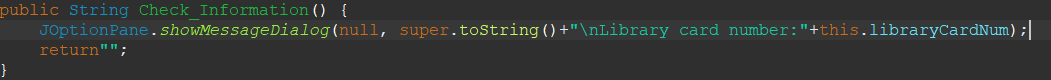
As mentioned above, the class includes the name, age , and gender attributes and has getters and setters for each one of them. For instance: this code represents the getters and setter for the name attribute. Additionally, the class has one abstract method named Check\_Information, which will be implemented and used to show the user or the librarian his/her personal information. Finally, the class also has a method called toString, which represents the information for either the user or the librarian.

The User\_Procedures Interface:

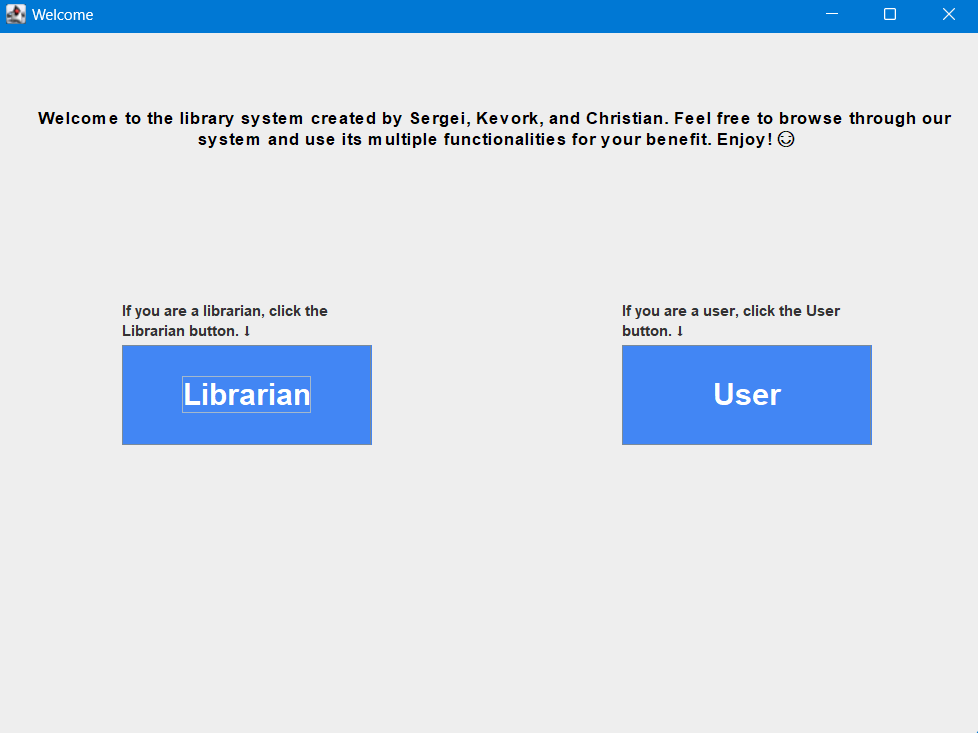
Coming to the User\_Procedures class, it is an interface which will be implemented by the user class. It contains six different methods, three of them enable the user to search for a book based on its title,author, or genre, and the other three are needed to borrow, return, or reserve a book.

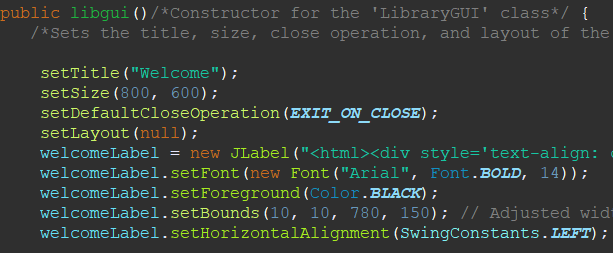
The User class:

Moving on to the user class, every user is distinguished from the other by a library card number,which is auto generated and used to login and use the functionalities of the program. The user class contains a LinkedList of users, which will be used to store the information of every registered user in the program. In addition to having getters and setters for the library card number and the User LinkedList, there is also the implementation for the six methods of the User\_Procedures. For example, the following code constitutes the SearchBookByTitle method:

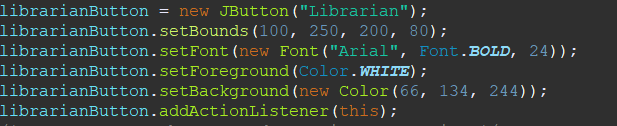
Here, we request from the user the title of the book he/she is searching for and go through the linkedlist and check whether the provided title matches a book. If it does, a dialog box appears showing the title,ISBN, author, and the genre of the book. If it doesn’t match neither of the books, a dialog box appears indicating that no books with the given title were found. The process and the code for the methods SearchBookByAuthor and SearchBookByGenre are similar, except that the searching process is based on the author’s name and the genre of the book for each method respectively. For the BorrowBook method, a search is completed with the title provided.If the book being searched for is found, available, and not reserved, then a dialog box appears displaying that the book has been borrowed and it accordingly becomes unavailable. Otherwise, If the book is reserved, then another dialog box appears indicating that the book cannot be borrowed. In case the book is not found, a dialog box appears showing that no such book exists in the database. Continuing with the ReturnBook method, it again searches through the linkedlist, checks if the borrowed book is neither available nor reserved, and accordingle either returns the information of the book, or informs the user that the book cannot be returned.In case no books were found, it displays a dialog box indicating the negative result of the conducted search. Additionally, the ReserveBook method allows a user to reserve a book if it is originally not reserved and available. Finally, the last method within the user class is the Check\_Information method, which is used to display a dialog box showing the information of the user.

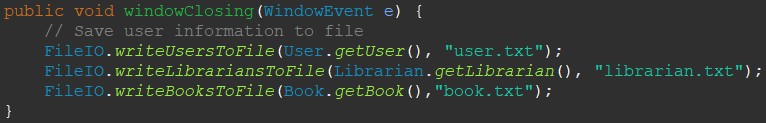
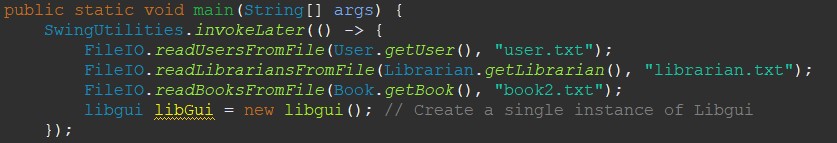
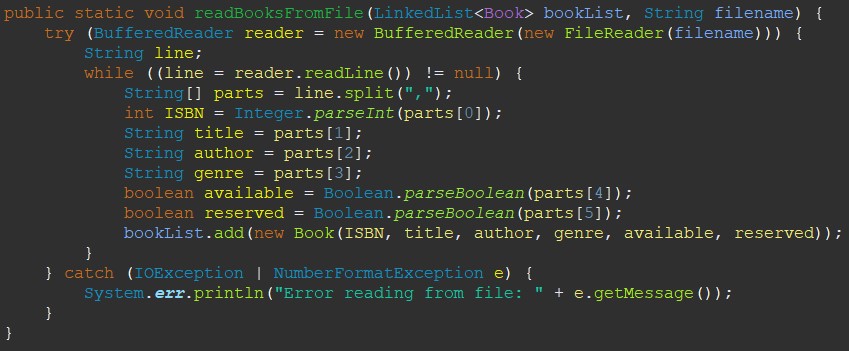
The Library GUI:

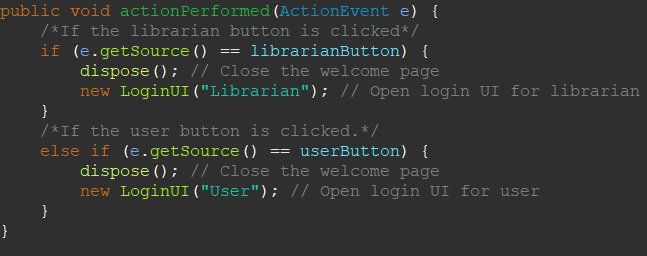
When running the program, the first window that appears on the screen is the following:

In order to display the welcome message in the center of the window, the following code was used:

This is the constructor for the class libgui that extends JFrame and ActionListener,which is for handling button click events.

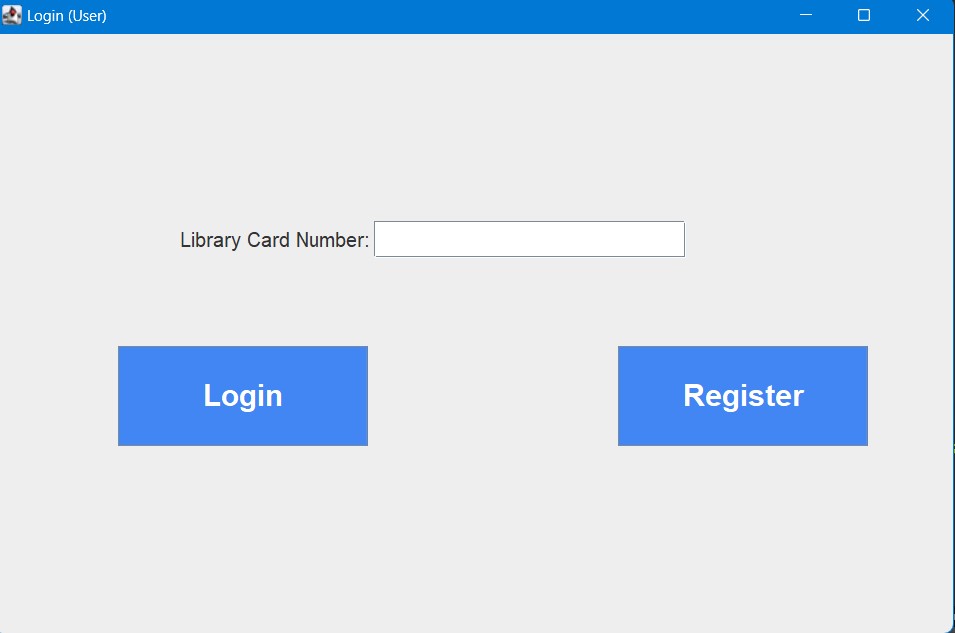
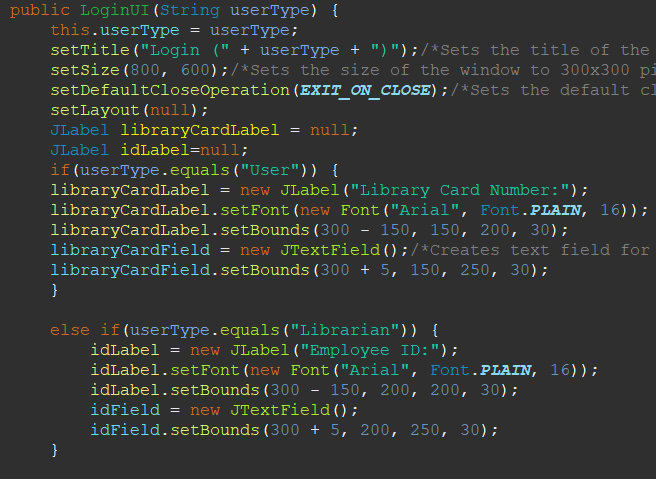
The setTitle(“Welcome”) is responsible for the text in the red box. SetSize() states the size of the entire page.In order to include in the page a welcome message, a label was created and given specific characteristics(such as its position,font size and theme).Similarily for the texts that are above the Librarian and the User buttons. Moving on to the buttons, this block of code stands for the librarian button. First line of code indicates what text the button will hold.Then, the positioning of the button is stated. Moving on, the font of the text is specified along with the color of the background.This is also the case for the user button. For every element to appear on the window, be it a button or a label, it should be added in this manner. For example, this code adds the welcome label containing the introductory text and the librarian button.

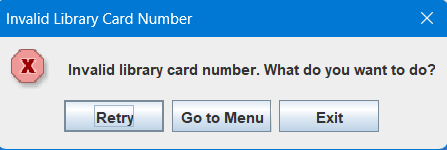
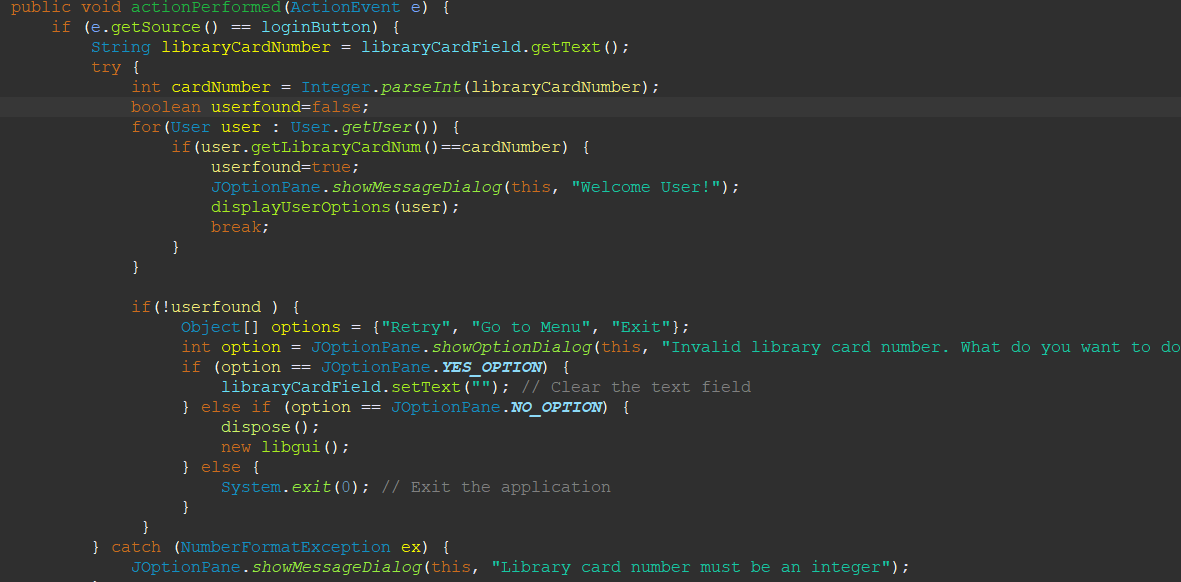
Continuing with the code, the setVisible(true) allows the JFrame to be visible,and this method serves the purpose of calling the proper methods in order to write whatever is found in the User,Librarian and Books LinkedLists to three separate files. When the program launches again, it can retrieve from the text files all of the information previously stored and arrange them in the LinkedLists through this code, which reads the data, stores them in the linkedlists and creates a single instance of libgui.

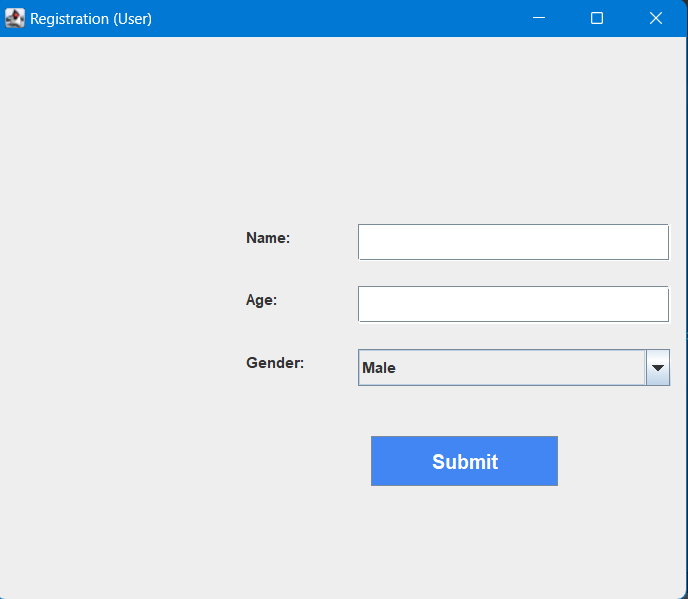
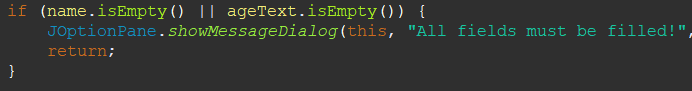
The readBooksFromFile method has the following implementation, where the program reads the textfile line by line, stores the title, author , and the other necessary information related to the book, and directly adds the book with the given specifications to the book linkedlist. The other methods to read and write are of similar structure.

This code detacts whether the librarian or user button was clicked and accordingly,closes the first window through the “dispose()” line and opens the correct one through the “newLoginUI” line, which either takes “Librarian” or “User” String parameter, based on which a specific page will be opened.

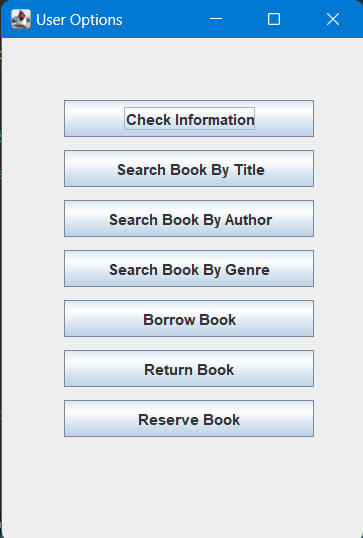
In case of clicking on the user button, the subsequent page will be revealed:

Here, a new class named LoginUI has been created containing a “Library Card Number” Label and besides it a text field where the user can input his/her card number and then click on the login button, which will traverse through the User LinkedList and check whether a user with the given card number exists. The code responsible for this is in the screenshot below. If the parameter passed is “User”, then the code in the red box will be executed and a page as the one on the right will be displayed on the screen. However, if the “Librarian” parameter is passed to the constructor, then the code below the red box will be executed, which will result in the replacement of the “Library Card Number” Label with the “Employee ID” label.

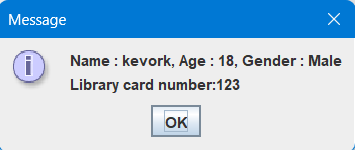
 In case the user inputs a correct library card number and clicks on the login button, a “welcome user” dialog box will appear on the screen and take the user to another window where other functionalities can be accessed from. If no such user exists, then the picture with the red border appears. If Retry is clicked, the code in the red box gets executed and subsequently, the user can enter a library card number again. If Go to Menu is clicked, then the code in yellow gets executed and takes the user to the main menu. Finally, if the exit button gets clicked, the program closes.

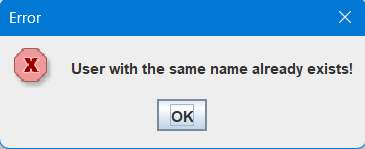
If the user clicks on the Register button, the constructor of the “RegistrationForm” class gets executed and the adjacent window appears, allowing the user to enter his/her name, age and gender. Once clicking on the submit button, the application checks whether all of the fields have been filled with this code.”1” If not, the application returns back to the registration page. Assuming all the fields have been filled, the application checks whether there is another user with the same name as the new one. If there exist two users with the same name, a dialog box once again appears indicating that the application cannot proceed.”2

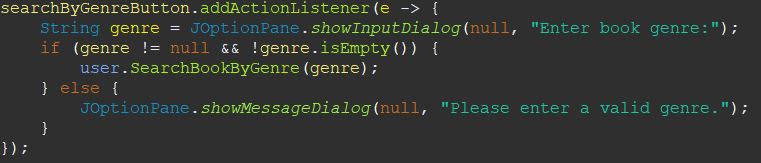
“1”.

Once everything goes smoothly and no problems arise, the system generates and shows to the user a library card number, after which it takes the user back to the main page. Pretending that a user logged in, this page appears.”3” Here the user can click on the check information button to view his personal data.”4”

“3”

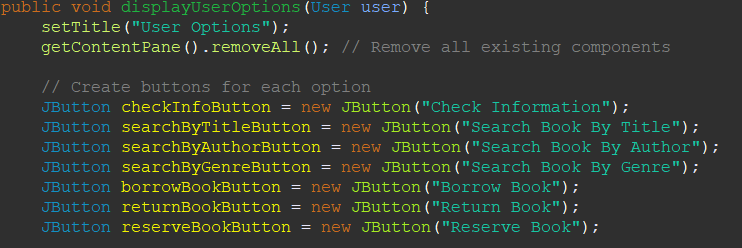
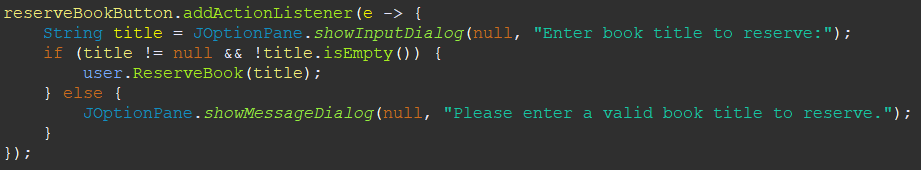
“4”.

“2”

If the Search Book By Genre button is clicked, the program requires the user to enter the genre of the book so that it can search through the book database and check if any books exist in the specified genre. For example, the code for this button is:

It checks if the user actually fills out the text field. If not, it displays a box indicating that a valid genre needs to be given and goes back to the User options page. If the user enters a valid genre, then the SearchBookByGenre method found in the user class gets called in order to perform the searching process.

Other explanations of codes:

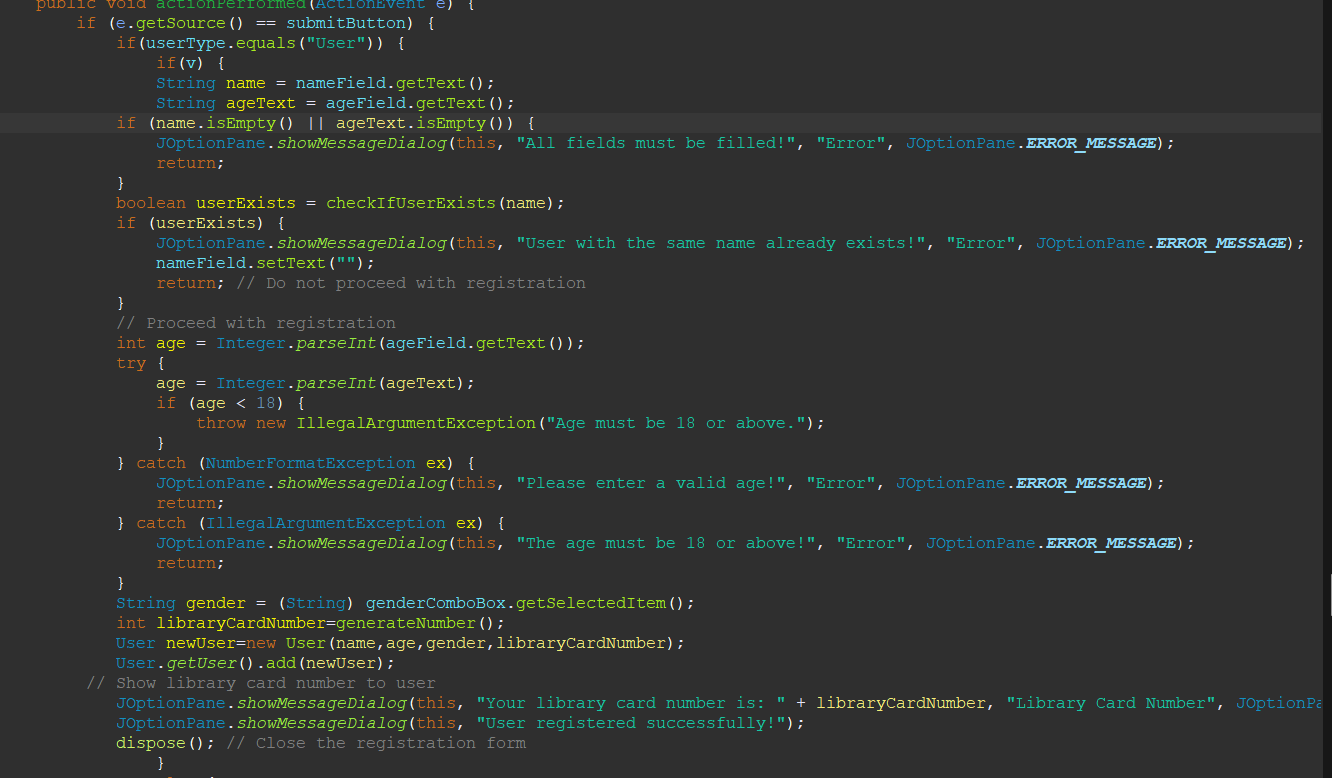
1. When the user logs in and as previously mentioned, a page containing 7 buttons appears on the screen. Each of the 7 buttons have action listeners, which are responsible to perform what is necessary. For example, the reserveBookButton has this action listener: The code implies that first of all, if the box where the user should input the name of the book remains empty, a dialog box appears indicating that a valid book title must be entered. Otherwise, the reservebook method will be called with the title of the book as a parameter.Similarily, all of the other buttons have the same structure of code.
2. Registration Form class: There is a fundemental point that needs to be elaborated in the registration class. A person using this program can register a user in two ways. Either by first clicking on the user button and then proceeding to click the register button, Or by logging in as a librarian and clicking on the AddUser button. In the first case, when the person has filled out the registration form, the program should automatically return back to the main menu. In the second case however, when the person inputs the users information and clicks on the submit button, the program should return to the page that contains buttons, allowing the librarian to perform specific tasks. In order to satisfy these conditions, there exists two constructors for the Registration form class. One serving for the first case and having only a string parameter, and the other having a JFrame paramter in addition to the string parameter. Furthermore, a boolean named “v” has been created for two purposes, which are to keep track of the constructor called and know what window to open after the execution of the code.

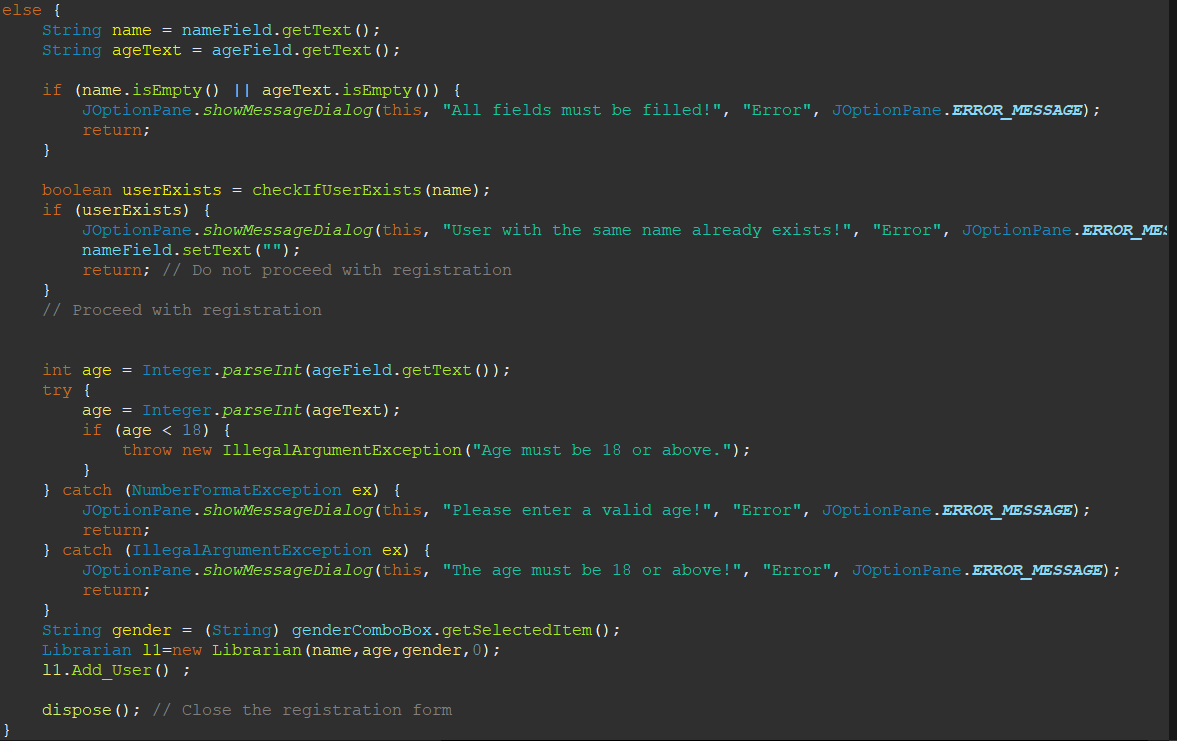




The second constructor with two parameters.

The first constructor with one parameter only.

It is also noteworthy to mention that when a librarian registers a user, the addUser method is called from the librarian class, while when a default person registers a user, the creation of the user identity and everything related to it happens in the actionPerformed method.To illustrate:

This is the code for creating the user and directly adding to the User LinkedList.

This is the code for adding a user through the method in the librarian class/As can be seen here, the addUser() method is being called to add the user.

Starting off,we have the displayLibrarianOptions method which is responsible for creating a graphical user interface (GUI) to display various options for a librarian.

**Setting the Title:**

The method sets the title of the GUI window to "Librarian Options" using the setTitle function.

Removing Existing Components:

It clears the content pane of any existing components using getContentPane().removeAll(). This ensures a clean slate for adding new components.

**Creating Buttons:**

Several JButton objects are instantiated for different librarian actions, such as adding a librarian, adding a user, checking information, searching for books by title, author, or genre, borrowing, returning, or reserving a book, and adding a new book.

Each button is initialized with a label indicating the corresponding action.

**Setting Button Bounds:**

The method calculates the appropriate position for each button to ensure they are centered both horizontally and vertically within the GUI window.

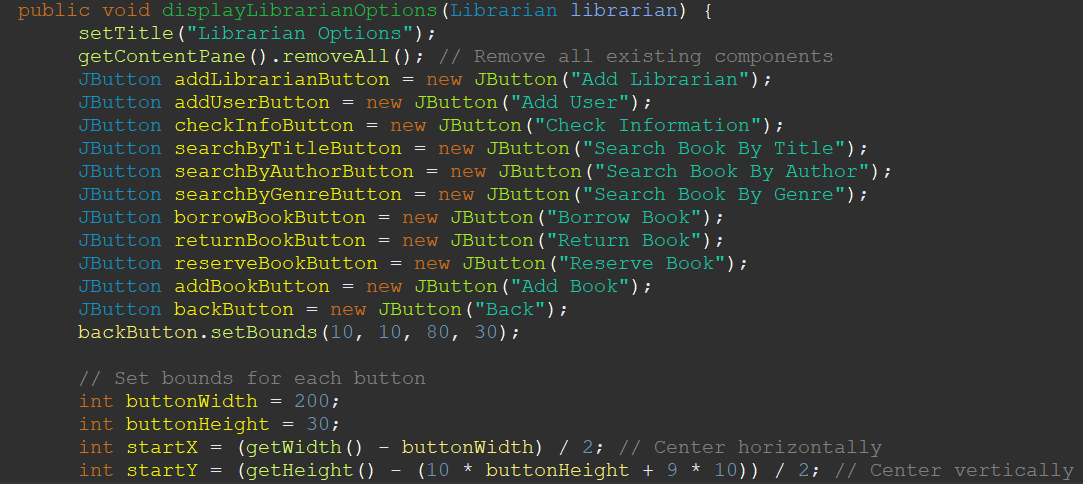
Button dimensions are specified with a width of 200 pixels and a height of 30 pixels.

The setBounds method is used to set the position and size of each button within the window.

**Back Button:**

Additionally, a back button is created and positioned at the top left corner of the window with dimensions 80x30 pixels.

This button allows the user to navigate back to a previous screen or close the current window.



Moving on, we have a block of code which determines the position of each button within the graphical user interface (GUI) window and adds action listeners to handle user interactions:

**Button Positions:**

The setBounds method is used to set the position and size of each button within the window.

The startX and startY variables are calculated to ensure the buttons are centered both horizontally and vertically.

Each button is positioned relative to the previous button, with a vertical spacing of (buttonHeight + 10) pixels between each button.

The buttonWidth and buttonHeight variables specify the dimensions of each button.

**Action Listeners:**

Action listeners are added to each button to define what happens when the button is clicked.

For example, when the back button is clicked, the current window is disposed of, and a new LoginUI window for librarians is opened.

Each button has a lambda expression that defines the action to be performed when the button is clicked.

These action listeners ensure that the GUI remains responsive to user input, executing the corresponding functionality for each button click.



Moving on with the action listeners, we have the main part which is responsible for the action listeners, that is keeping the functionality of the buttons in order.

**Add Librarian Button Action Listener:**

When the "Add Librarian" button is clicked, the current window is disposed of using the dispose() method.

Subsequently, a new instance of the RegistrationForm class is created with the role parameter set to "Librarian".

The RegistrationForm window is then made visible to the user, allowing them to register a new librarian account.

**Add User Button Action Listener:**

Clicking the "Add User" button triggers the disposal of the current window.

A new instance of the RegistrationForm class is created with the role parameter set to "User".

The RegistrationForm window is displayed, enabling the user to register a new user account.

**Check Information Button Action Listener:**

Upon clicking the "Check Information" button, a debug message ("Check Information button clicked") is printed to the console.

The Check\_Information method of the librarian object is invoked, presumably to retrieve and display relevant information.

**Search Book By Title Button Action Listener:**

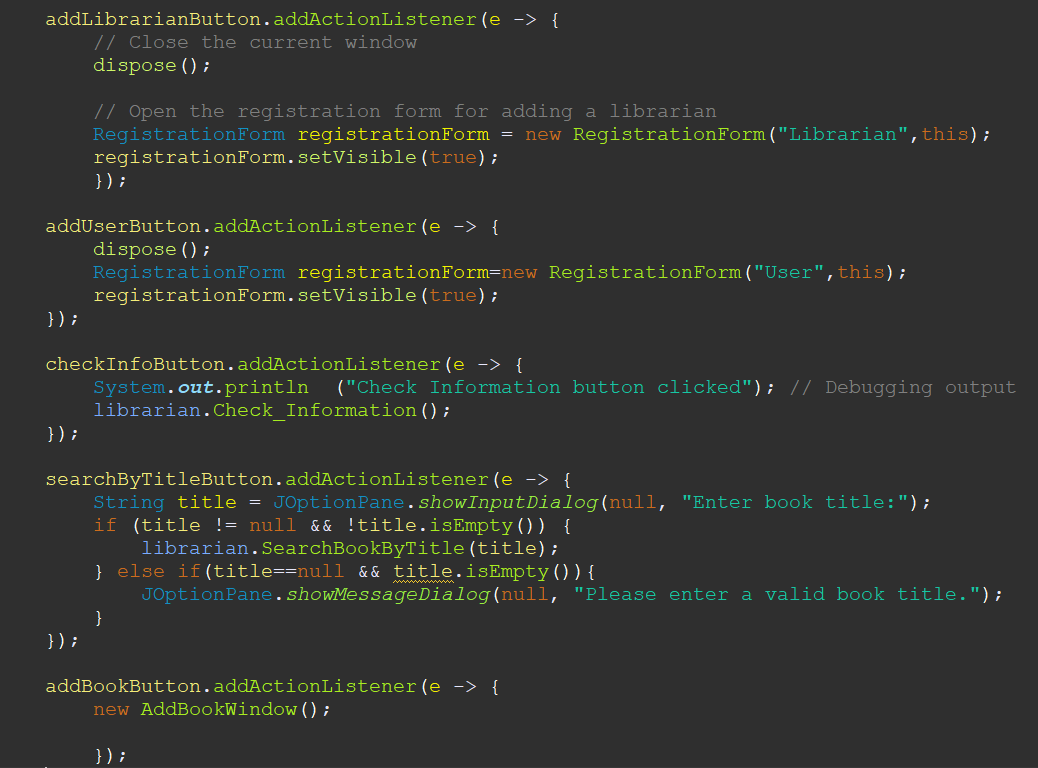
Clicking the "Search Book By Title" button prompts the user to input a book title via a dialog box.

If a non-empty title is provided, the SearchBookByTitle method of the librarian object is called with the entered title.

If the title input is empty or null, a message dialog prompts the user to enter a valid book title.

**Add Book Button Action Listener:**

The "Add Book" button triggers the instantiation of a new AddBookWindow object, allowing the user to add a new book to the system.



**Search Book By Author Button Action Listener:**

Upon clicking the "Search Book By Author" button, the user is prompted to input a book author via a dialog box.

If a non-empty author name is provided, the SearchBookByAuthor method of the librarian object is called with the entered author.

If the author input is empty or null, a message dialog prompts the user to enter a valid author name.

**Search Book By Genre Button Action Listener:**

Clicking the "Search Book By Genre" button prompts the user to input a book genre via a dialog box.

If a non-empty genre is provided, the SearchBookByGenre method of the librarian object is called with the entered genre.

If the genre input is empty or null, a message dialog prompts the user to enter a valid genre.

**Borrow Book Button Action Listener:**

When the "Borrow Book" button is clicked, the user is prompted to input a book title via a dialog box.

If a non-empty title is provided, the BorrowBook method of the librarian object is called with the entered title.

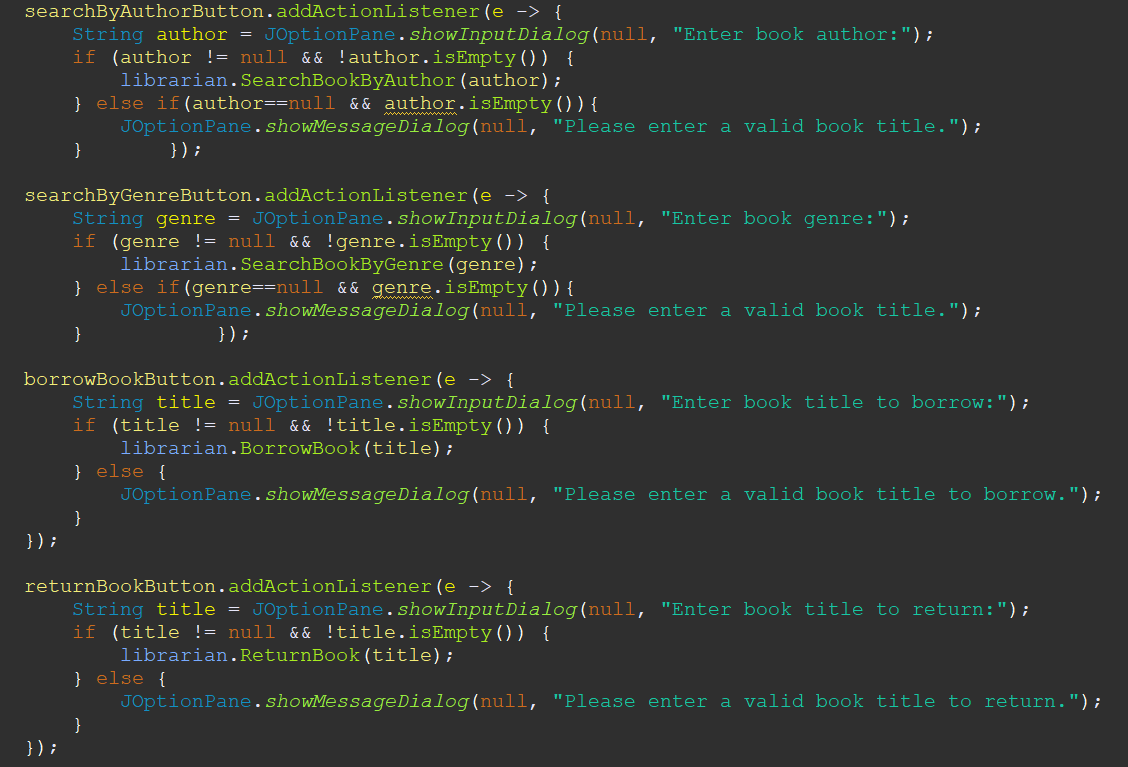
If the title input is empty or null, a message dialog prompts the user to enter a valid book title.

**Return Book Button Action Listener:**

Clicking the "Return Book" button triggers a dialog box prompt for the user to input a book title.

If a non-empty title is provided, the ReturnBook method of the librarian object is called with the entered title.

If the title input is empty or null, a message dialog prompts the user to enter a valid book title.

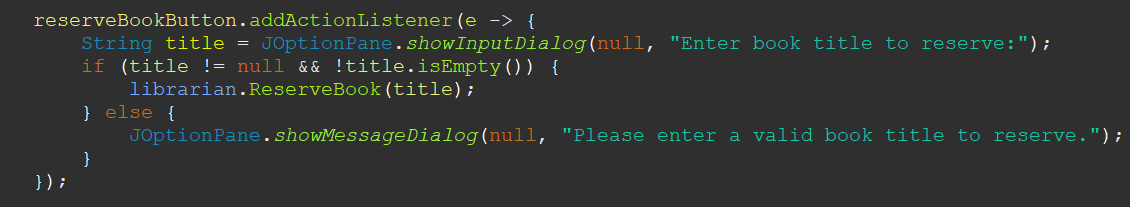


**Reserve Book Button Action Listener:**

Upon clicking the "Reserve Book" button, the user is prompted to input a book title via a dialog box.

If a non-empty title is provided, the ReserveBook method of the librarian object is called with the entered title.

If the title input is empty or null, a message dialog prompts the user to enter a valid book title.



Now we have to add the buttons which we have worked hardly on. This segment of code adds the previously initialized buttons to the frame and performs additional actions related to window behavior:

**Adding Buttons:**

Each button (addLibrarianButton, addUserButton, checkInfoButton, searchByTitleButton, searchByAuthorButton, searchByGenreButton, borrowBookButton, returnBookButton, reserveBookButton, addBookButton, backButton) is added to the frame using the add method.

By adding these buttons, they become visible and accessible within the graphical user interface (GUI) window.

**Repainting the Frame:**

After adding the buttons, the frame is repainted to reflect any changes made to its contents.

The revalidate() method is called to ensure that any layout changes are applied, and the components are laid out appropriately.

The repaint() method is invoked to redraw the frame, updating its visual appearance to reflect the addition of the buttons.

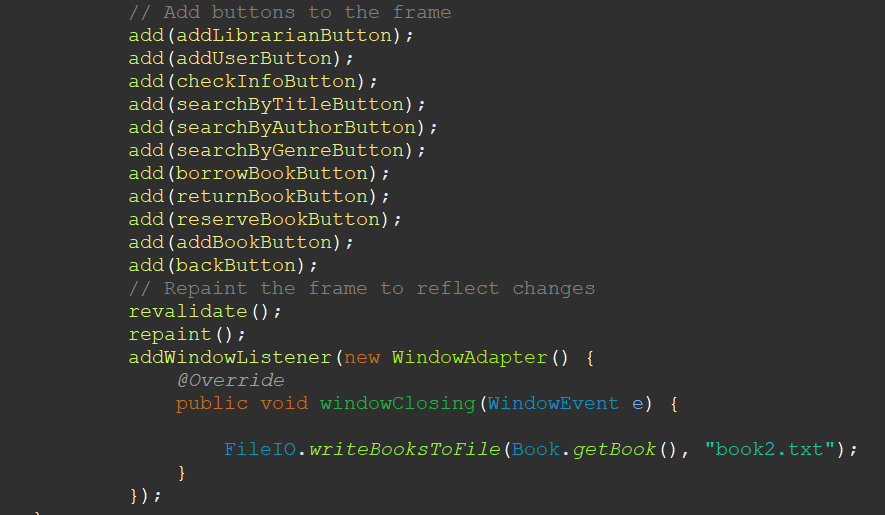
**Window Closing Action:**

A WindowAdapter is added to the frame to listen for the window closing event.

When the window is closed, the windowClosing method of the adapter is invoked.

Within this method, the writeBooksToFile method of the FileIO class is called to write the current book data to a file named "book2.txt".

This ensures that the book data is persisted to the file system before the application terminates, allowing for data retention between sessions.



Next up we have the AddBookWindow class which extends JFrame and implements the ActionListener interface. It represents a window for adding a new book to the library system.

**Constructor:**

The constructor initializes the window's properties, such as title, size, default close operation, layout, and visibility.

The window is titled "Add Book" and has a size of 800x600 pixels.

It is set to dispose when the user closes the window.

The layout is defined as a GridLayout with 5 rows, 2 columns, and gaps of 10 pixels between components.

**Text Fields:**

Four JTextField objects (titleField, authorField, genreField, isbnField) are instantiated to capture information about the new book, including title, author, genre, and ISBN.

Each text field corresponds to a specific attribute of the book.

**Labels:**

Labels (titleLabel, authorLabel, genreLabel, isbnLabel) are created to provide descriptive text for each text field.

These labels help the user understand what information to input into each field.

**Adding Components:**

The labels and text fields are added to the frame in a specific order to ensure proper alignment within the grid layout.

They are added in pairs, with each label followed by its corresponding text field.

**Submit Button:**

A "Submit" button (submitButton) is created and added to the frame.

An action listener is registered for the button, with the AddBookWindow class itself as the listener.

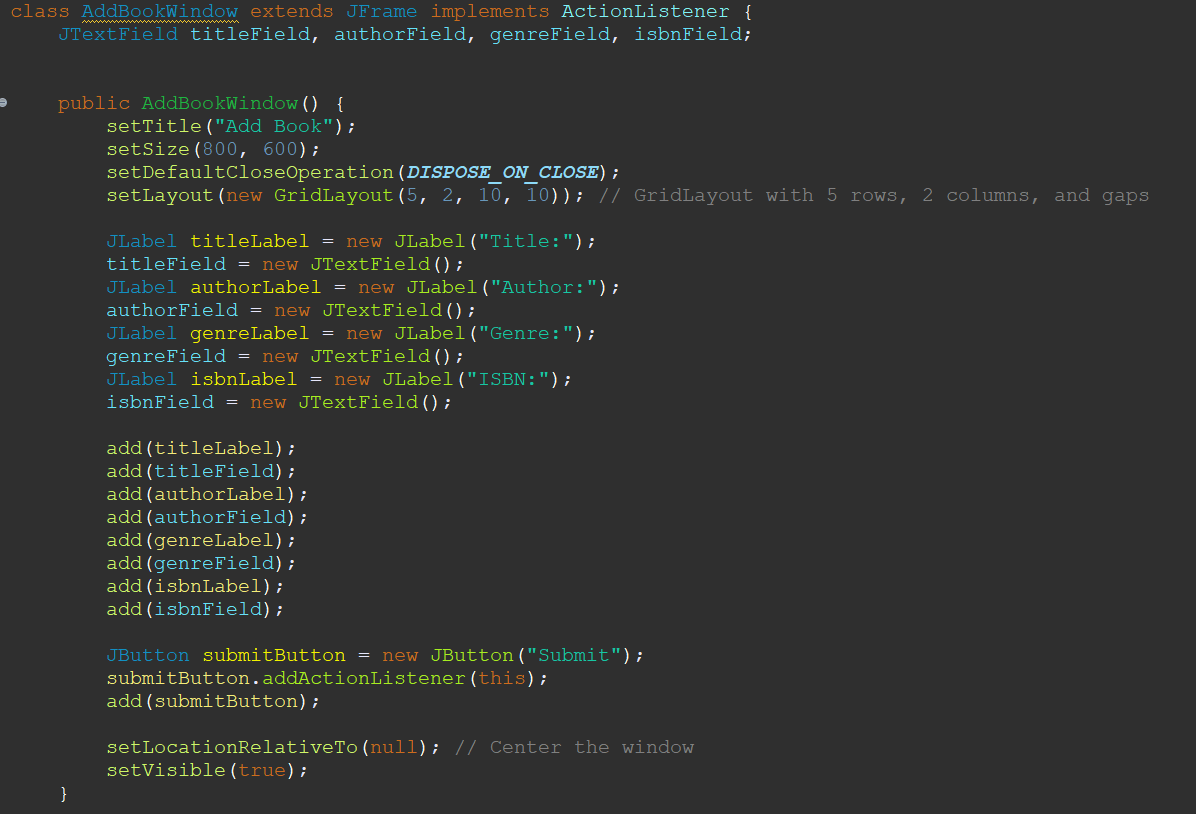
When the button is clicked, the actionPerformed method is invoked to handle the event.

**Centering the Window:**

The setLocationRelativeTo(null) method centers the window on the screen, ensuring it appears in the middle of the user's display.

**Visibility:**

Finally, the setVisible(true) method is called to make the window visible to the user.



The actionPerformed method is overridden from the ActionListener interface to handle events triggered by user interactions with components in the AddBookWindow.

**Event Handling:**

The method is invoked when an action event occurs, such as a button click.

It first checks if the event source is an instance of a JButton, ensuring that the action was triggered by a button.

**Button Identification:**

If the event source is a button, the method retrieves the specific button that triggered the event (sourceButton).

It checks if the text of the button is "Submit" to ensure that the submit button triggered the event.

**Book Details Extraction:**

Upon clicking the submit button, the method retrieves the details of the new book entered by the user from the text fields.

The title, author, genre, and ISBN are obtained from their respective text fields.

The ISBN, represented as an integer, is parsed from the text field using Integer.parseInt().

**Book Object Creation:**

Using the extracted book details, a new Book object (b11) is instantiated, representing the newly added book.

The constructor of the Book class is called with the extracted details along with default values for availability flags.

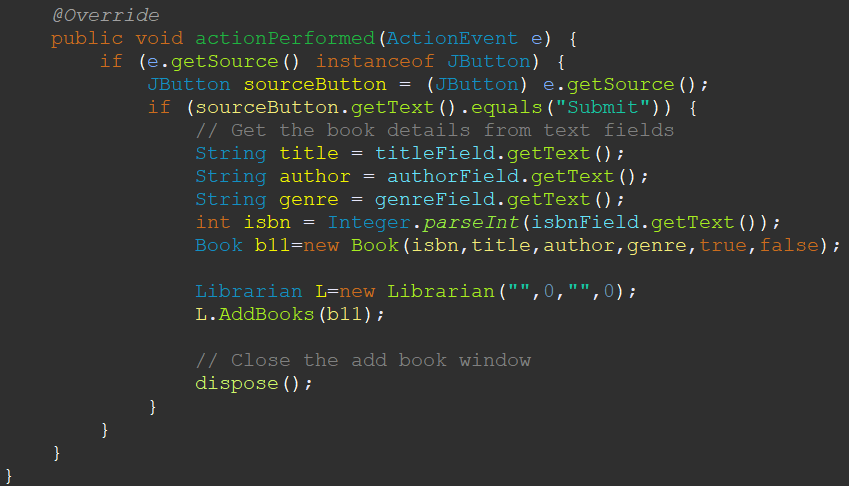
**Adding Book to Library:**

An instance of the Librarian class (L) is created to perform librarian-related operations.

The AddBooks method of the Librarian class is invoked, passing the newly created Book object as a parameter to add it to the library catalog.

**Window Disposal:**

Finally, the dispose() method is called to close the AddBookWindow after the book has been successfully added to the library.



The conditional block handles the registration process for adding a new librarian to the system.

**Input Validation:**

If the user clicked on a button other than "Submit" in the AddBookWindow, this block is executed.

It first checks if the boolean variable v is false, indicating that the registration is for a librarian.

**Name and Age Validation:**

The method retrieves the name and age entered by the user from their respective text fields (nameField and ageField).

If either the name or age field is empty, an error message dialog is displayed, prompting the user to fill in all required fields.

Additionally, if the age entered is less than 18, an error message dialog notifies the user that the age must be 18 or above.

The NumberFormatException and IllegalArgumentException are caught to handle invalid age input or age less than 18, respectively.

**Check for Existing Librarian:**

The method checks if a librarian with the same name already exists in the system by invoking the checkIfLibrarianExists method.

If a librarian with the same name is found, an error message dialog is displayed, informing the user that a librarian with the same name already exists, and the registration process is halted.

**Librarian Registration:**

If all input validations pass and no librarian with the same name exists, the registration process proceeds.

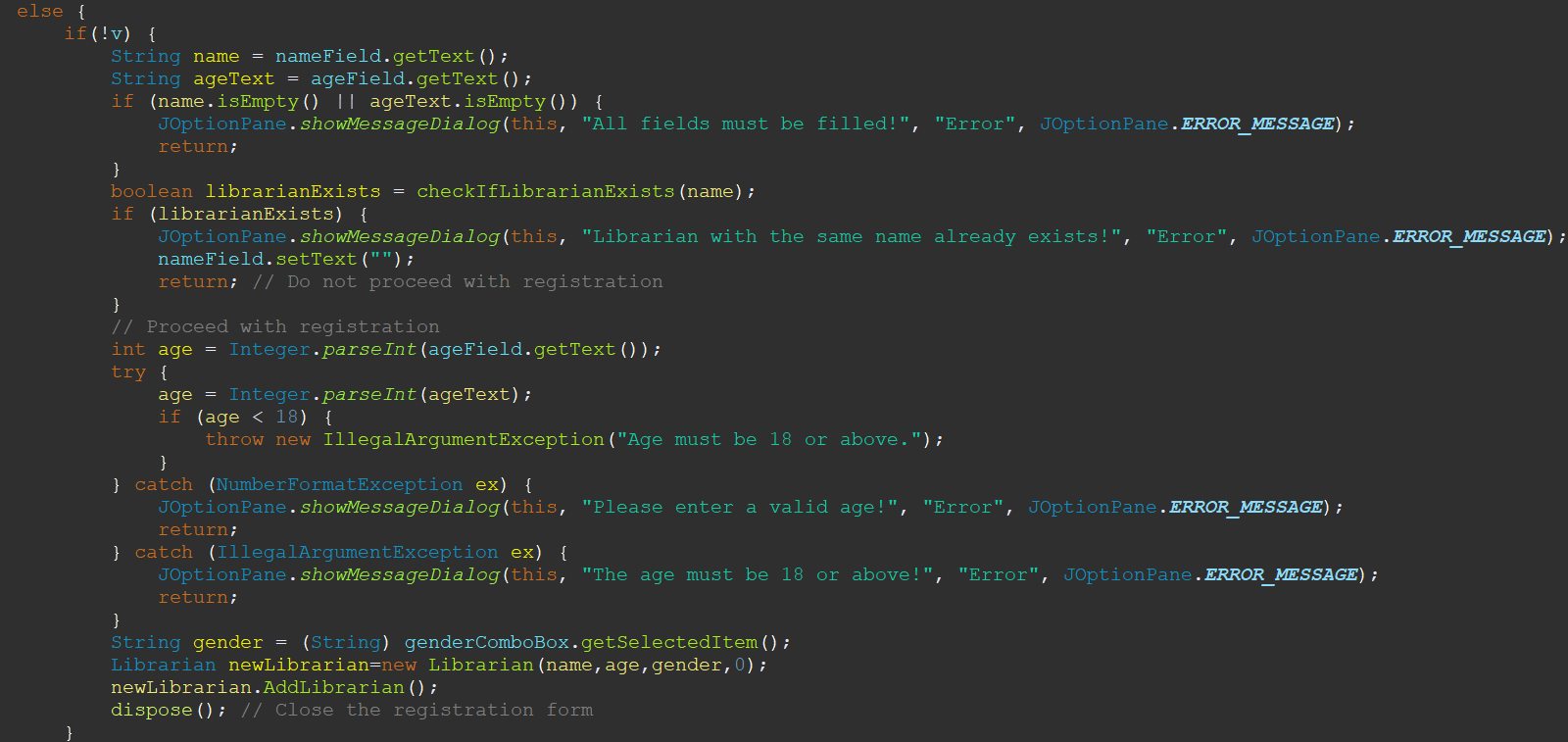
The age and gender input are parsed and retrieved from their respective fields (ageField and genderComboBox).

A new Librarian object (newLibrarian) is created with the entered name, age, gender, and default values for other attributes.

The AddLibrarian method of the Librarian class is invoked to add the new librarian to the system.

**Window Disposal:**

Finally, the registration form window is closed using the dispose() method after the registration process is completed.



This conditional block handles the registration process for adding a new user to the system.

**Input Validation:**

This block executes if the registration is not for a librarian, indicated by the absence of the boolean variable v.

It retrieves the name and age entered by the user from their respective text fields (nameField and ageField).

If either the name or age field is empty, an error message dialog is displayed, prompting the user to fill in all required fields.

Additionally, if the age entered is less than 18, an error message dialog notifies the user that the age must be 18 or above.

**Check for Existing Librarian:**

The method checks if a librarian with the same name already exists in the system by invoking the checkIfLibrarianExists method.

If a librarian with the same name is found, an error message dialog is displayed, informing the user that a librarian with the same name already exists, and the registration process is halted.

**User Registration:**

If all input validations pass and no librarian with the same name exists, the registration process proceeds.

The age and gender input are parsed and retrieved from their respective fields (ageField and genderComboBox).

An employee ID is generated for the new user using the generateNumber method.

A new Librarian object (newLibrarian) is created with the entered name, age, gender, and generated employee ID.

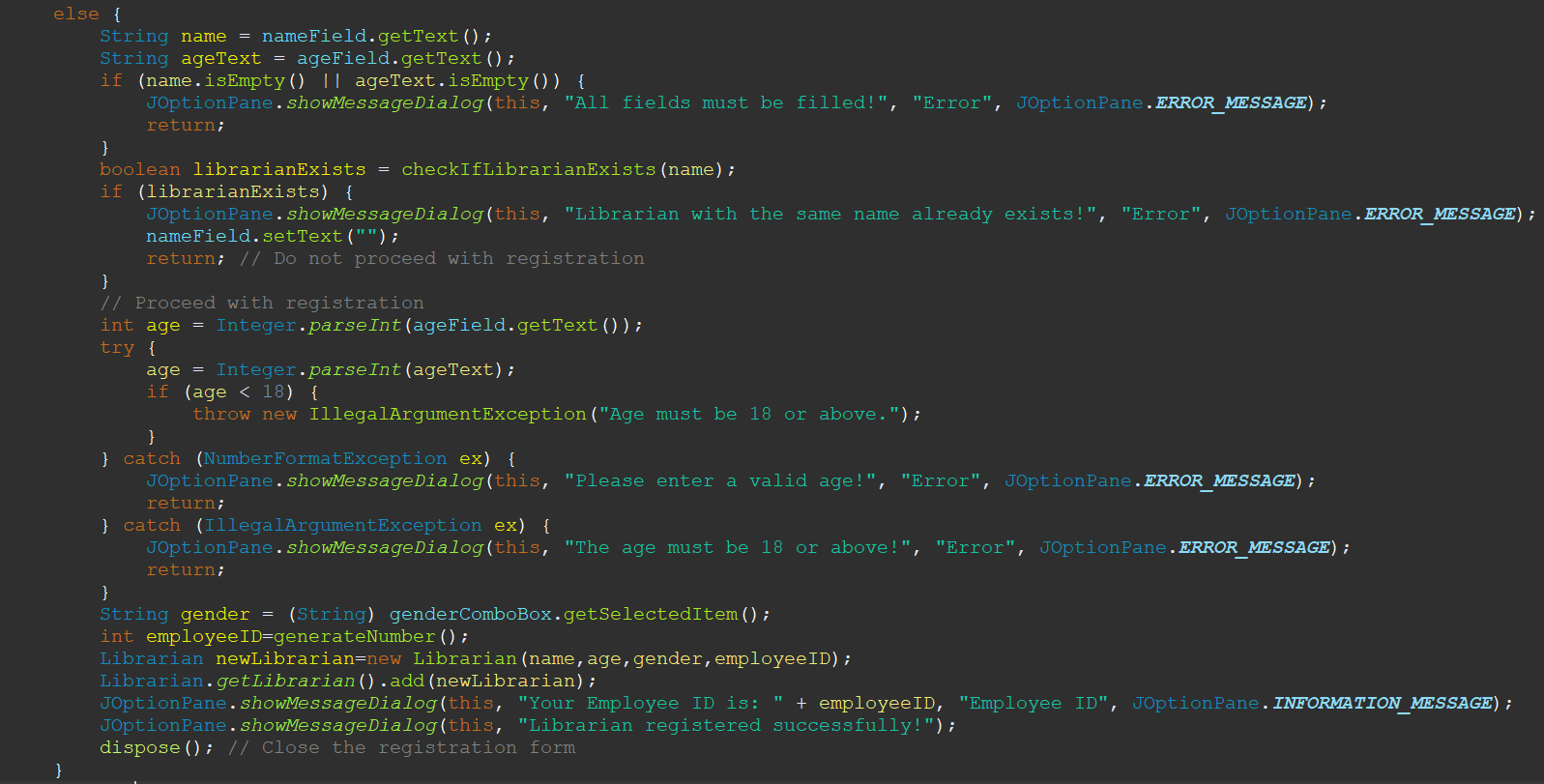
The new librarian object is added to the list of librarians using the add method of the ArrayList.

An information message dialog displays the assigned employee ID to the user.

A success message dialog notifies the user that the librarian registration was successful.

**Window Disposal:**

Finally, the registration form window is closed using the dispose() method after the registration process is completed.



Challenges faced throughout the project:

1. One obstacle encountered in the library graphical user interface was to save the data found in the three linkedlists to a file and retrieve them back at the beginning of the program. Even when the necessary code was developed and placed in the method that handles window closing belonging to the libgui class, the information of the books were being erased and not saved. The solution was that a separate window closing method should’ve been placed for the class that handles the librarian part of the program because when initially the person clicks on the librarian button and enters his employee ID, thereby logging in the program, the window closing method of the libgui class is being executed, even though no books have been added. With the addition of the method in the displayLibrarianOptions section, if the librarian added any books, they were being directly written at the instant of termination of the program.
2. Another problem that arose was when the addUser or addLibrarian method of the librarian class was being called. Without passing any of the personal data regarding the person as a parameter, the method needed to be called. Fortunately, the problem was resolved by creating a new instance of the librarian class, giving the instance the same information as the user or the other librarian, and then transfering the data back from the librarian to the user in the AddLibrarian or AddUser method.
3. One challenge faced in the SearchBookByTitle method was to display the data of several books together in a single dialog box. The problem was resolved by appending the information of the books to a StringBuilder that will be used as a mean to collectively show the titles of the books.

Division of work:  
Christian: Book class, the opening page of the gui and the login page:

Sergei:librarian procedures and librarian classes. Everything related to the librarian (registration form, the page containing the buttons of the librarian, the functionalities for the buttons)in the GUI.

Kevork: User procedures and the user classes. Everything related to the User(registration form, the page containing the buttons of the user, the functionalities for the buttons)in the GUI.