NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY

MILTARY COLLEGE OF SIGNALS

Data Structures & Algorithms

(CS-250)

OPEN-ENDED LAB

Submitted by: MUHAMMAD AHMAD SULTAN

CMS ID: 408709

RANK: NC

COURSE: BESE-28

SECTION: C

Submitted to: Mam Muntaha Noor

Dated: 08-01-2024

Catalog of Open-Ended Lab

Table of Contents

Title of The Open-Ended Lab

BookTrack Nexus

Library Management System

A Comprehensive Library Management System in Java, integrating Data Structures & Algorithms, Object-Oriented Programming, and Database Management with MySQL, ensuring efficient and secure operations.

About me

Allow me to introduce myself. I am *Muhammad Ahmad Sultan*, a highly motivated second-year student at the ***Military College of Signals***, currently pursuing a **Bachelor's degree in Software Engineering**. With unwavering determination, I am committed to achieving my goals and making significant strides in my academic journey.

Muhammad Ahmad Sultan

Developer

▪ Abstract:

The Java-based Library Management System (LMS) utilizes the Swing framework and MySQL Workbench for efficient resource, user, and transaction management. Rooted in Object-Oriented Programming, the system integrates data structures and algorithms to create a comprehensive solution. With robust search and sorting capabilities, users can quickly locate books, while streamlined borrowing processes and transaction history tracking enhance user interactions. The integrated return mechanism ensures accurate inventory and user record updates, contributing to a user-friendly and organized library environment. This system aims to efficiently manage library resources, user interactions, and transactions. It incorporates Object-Oriented Programming (OOP) concepts, data structures, and algorithms to create a comprehensive and user-friendly solution.

▪ History of Library Management Systems:

Library Management Systems have evolved over the years, transitioning from manual card catalogs to digital databases. The shift to automation has significantly improved the efficiency of library operations, enabling easy cataloging, tracking, and management of library resources. The advent of computer technology has facilitated the development of sophisticated Library Management Systems, streamlining tasks for librarians and enhancing the user experience. The concept of Library Management Systems dates back to the early days of computerization when libraries began to adopt automated methods to manage their resources efficiently. Over time, these systems evolved to incorporate advanced features, such as user authentication, transaction tracking, and seamless user interfaces.

▪ Objective:

The primary objective of the Library Management System is to provide a user-friendly platform for efficiently managing library resources, user interactions, and transactions. The system aims to automate tasks such as book and user management, borrowing and returning books, maintaining transaction history, and implementing search and sorting functionalities.

▪ Scope:

The Library Management System encompasses a comprehensive scope, focusing on efficient management of both books and users. It ensures seamless storage and retrieval of information, facilitating the smooth handling of borrowing and returning transactions while keeping availability statuses up-to-date. The system also maintains a detailed transaction history, recording pertinent information about books borrowed and returned by users. To enhance user experience, robust search functionalities for books and users are implemented, complemented by effective sorting operations for organized accessibility of information within the library system.

▪ Functionality & Implementation:

1. Main_Window.java (Splash Screen)

Functionality:

Displays a splash screen upon application launch.

Provides an initial visual interface before loading the main application.

Implementation:

Utilizes Java Swing for GUI components.

May include a timer for a predetermined display time.

Invokes subsequent login functionality after the splash screen.

2. Login.java (User Authentication)

Functionality:

Manages user authentication to access the library system.

Validates user credentials against stored information.

Implementation:

Incorporates Swing components for a user-friendly login interface.

Utilizes JavaConnect.java for database connection.

Implements action listeners to handle login attempts.

Navigates to the Loading.java upon successful authentication.

3. Loading.java (Loading Screen)

Functionality:

Displays a loading screen after successful user authentication.

Prepares the application for user interaction.

Implementation:

Utilizes Swing components for visual representation.

May include background tasks or thread management for loading operations.

Transitions to the Home.java after the loading process is complete.

4. Home.java (Menu and Dashboard)

Functionality:

Serves as the main dashboard for users post-authentication.

Provides access to various functionalities and features.

Implementation:

Uses Swing components to create an intuitive dashboard.

Includes buttons or menu items for different operations (e.g., AddBook, Issue, Return, Statistics, Registered Students, etc.).

Implements action listeners to handle user interactions and navigation.

5. AddBook.java (Add Books)

Functionality:

Allows users to input and add new books to the library system.

Implementation:

Incorporates Swing components for book information input.

Utilizes JavaConnect.java for database interaction.

Implements event handling for the addition of new books.

6. JavaConnect.java (Database Connection)

Functionality:

Manages the connection between the Java application and MySQL database.

Implementation:

Utilizes JDBC to establish and manage connections.

May include methods for executing queries, updates, and handling database transactions.

Ensures secure and efficient communication between the application and the database.

7. AllBooks.java (View All Books)

Functionality:

Displays a comprehensive list of all books available in the library.

Implementation:

Utilizes Swing components for an organized and user-friendly display.

Retrieves book information from the database using JavaConnect.java.

Allows users to scroll through the list of books.

8. Issue.java (Issue Books)

Functionality:

Manages the process of issuing books to registered students.

Implementation:

Incorporates Swing components for student and book selection.

Updates the database to reflect the issued books and maintains transaction records.

May include validation to ensure proper book availability.

9. Return.java (Return Books)

Functionality:

Handles the return of borrowed books, updating availability and transaction records.

Implementation:

Uses Swing components for book return interface.

Interacts with JavaConnect.java to update the database.

Calculates any applicable fines or penalties.

10. Statistics.java (Transaction History)

Functionality:

Presents statistical information about library transactions, such as issue and return history.

Implementation:

Utilizes Swing components to display statistical data.

Retrieves transaction information from the database using JavaConnect.java.

May include graphical representations for enhanced data visualization.

11. RegisteredStudents.java (View Registered Students)

Functionality:

Displays a list of all registered students in the library system.

Implementation:

Utilizes Swing components to organize and present student information.

Retrieves student data from the database using JavaConnect.java.

Allows users to browse through the list of registered students.

12. Student.java (Add Student)

Functionality:

Enables the addition of new students to the library system.

Implementation:

Incorporates Swing components for student information input.

Interacts with JavaConnect.java for database updates.

Implements event handling for student registration.

13. About.java

Functionality:

Provides information about the Library Management System application.

Implementation:

Utilizes Swing components to display application details.

May include version information, development credits, and other relevant details.

14. Register.java (User Registration)

Functionality:

Manages the registration of new users, such as librarians or administrators.

Implementation:

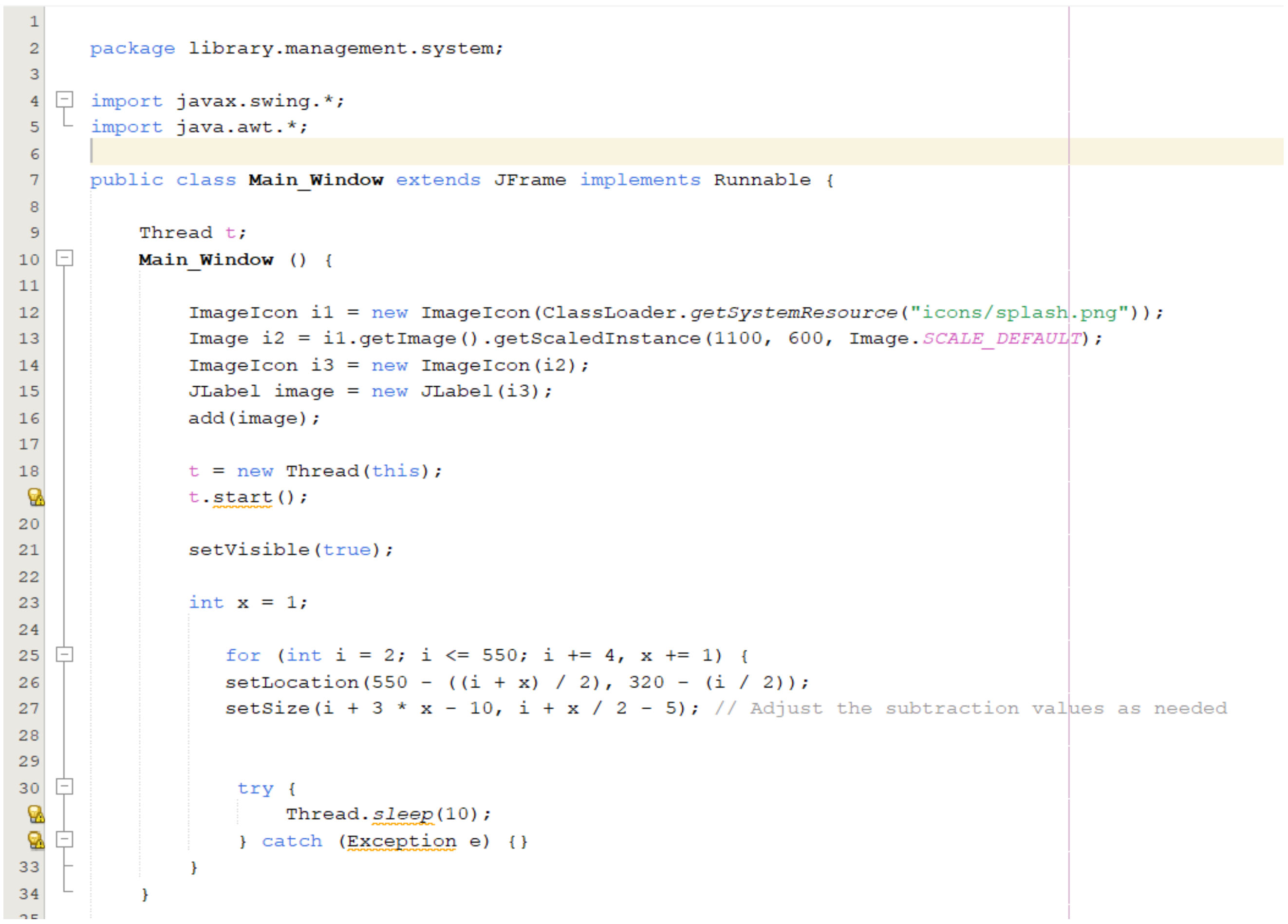
Incorporates Swing components for user registration input.

Utilizes JavaConnect.java for database interaction to store user information securely.

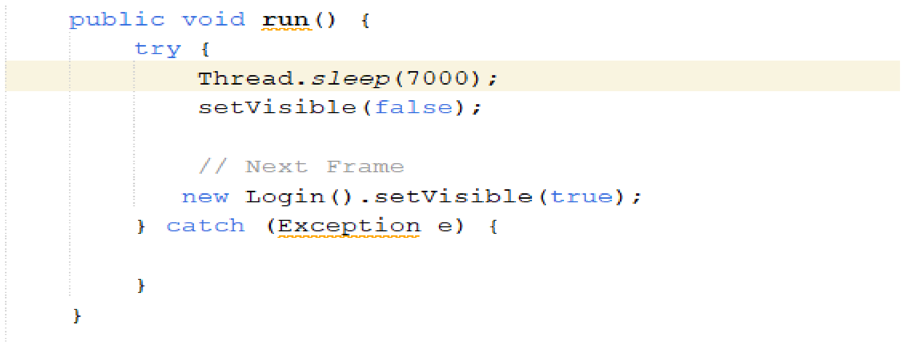
Implements event handling for user registration processes.

Java Classes

- *Main_Window.java*

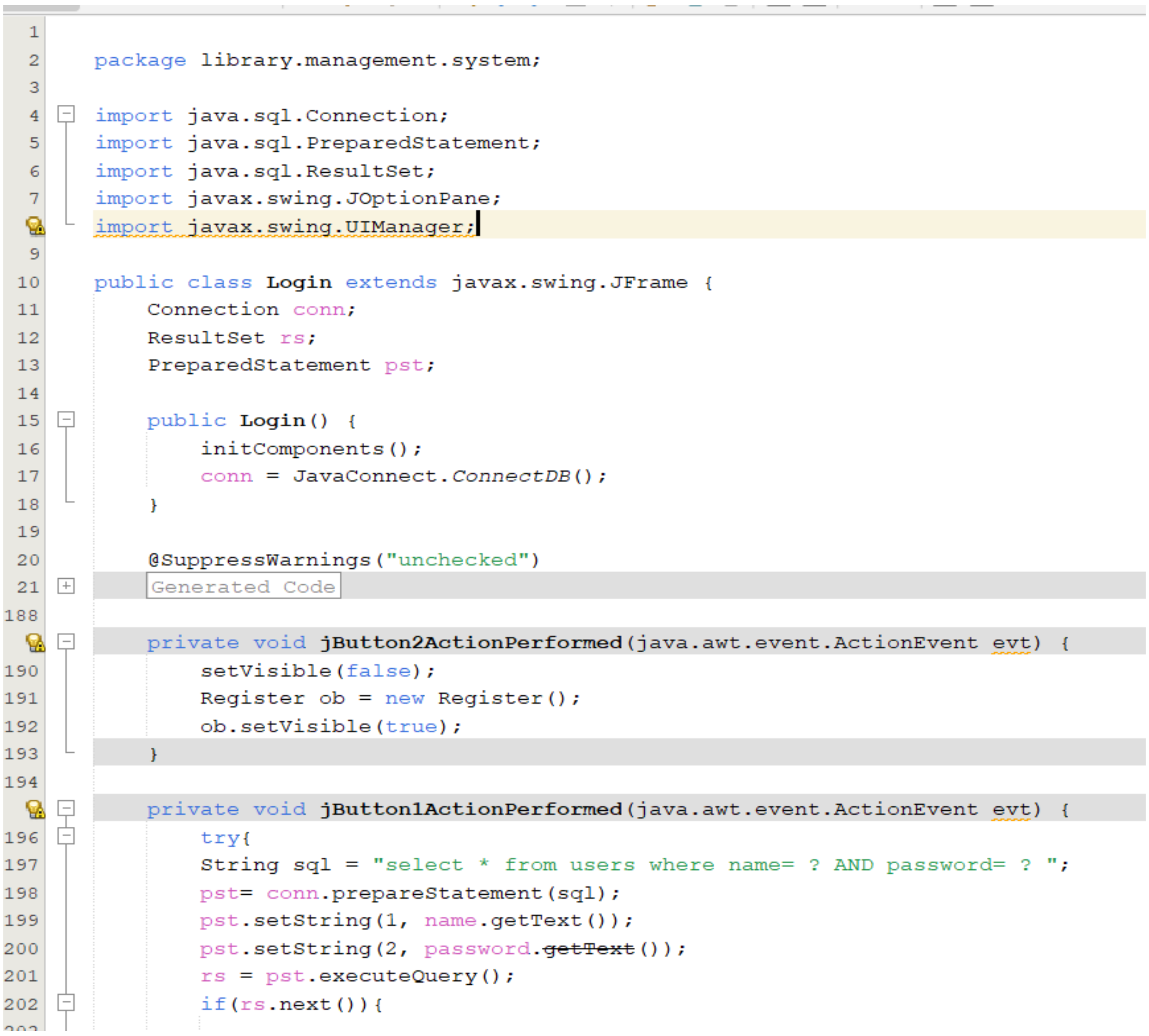


```java

package library.management.system;

import javax.swing.*;
import java.awt.*;

public class Main_Window extends JFrame implements Runnable {

    Thread t;
    Main_Window () {

        ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icons/splash.png"));
        Image i2 = i1.getImage().getScaledInstance(1100, 600, Image.SCALE_DEFAULT);
        ImageIcon i3 = new ImageIcon(i2);
        JLabel image = new JLabel(i3);
        add(image);

        t = new Thread(this);
        t.start();

        setVisible(true);

        int x = 1;

          for (int i = 2; i <= 550; i += 4, x += 1) {
          setLocation(550 - ((i + x) / 2), 320 - (i / 2));
          setSize(i + 3 * x - 10, i + x / 2 - 5); // Adjust the subtraction values as needed


           try {
               Thread.sleep(10);
           } catch (Exception e) {}
        }
    }
```



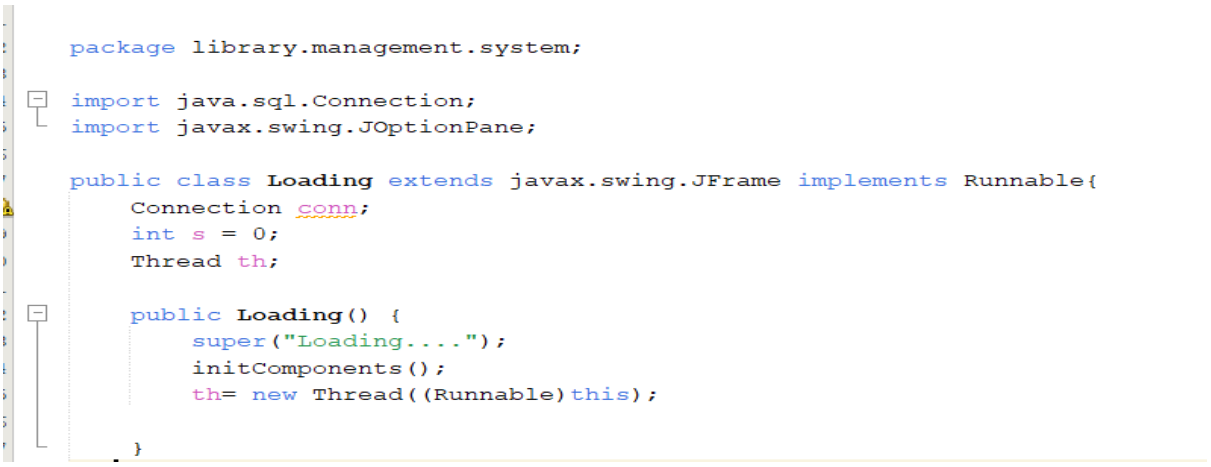
```java
    public void run() {
        try {
            Thread.sleep(7000);
            setVisible(false);

            // Next Frame
           new Login().setVisible(true);
        } catch (Exception e) {

        }
    }
```

- *Login.java*



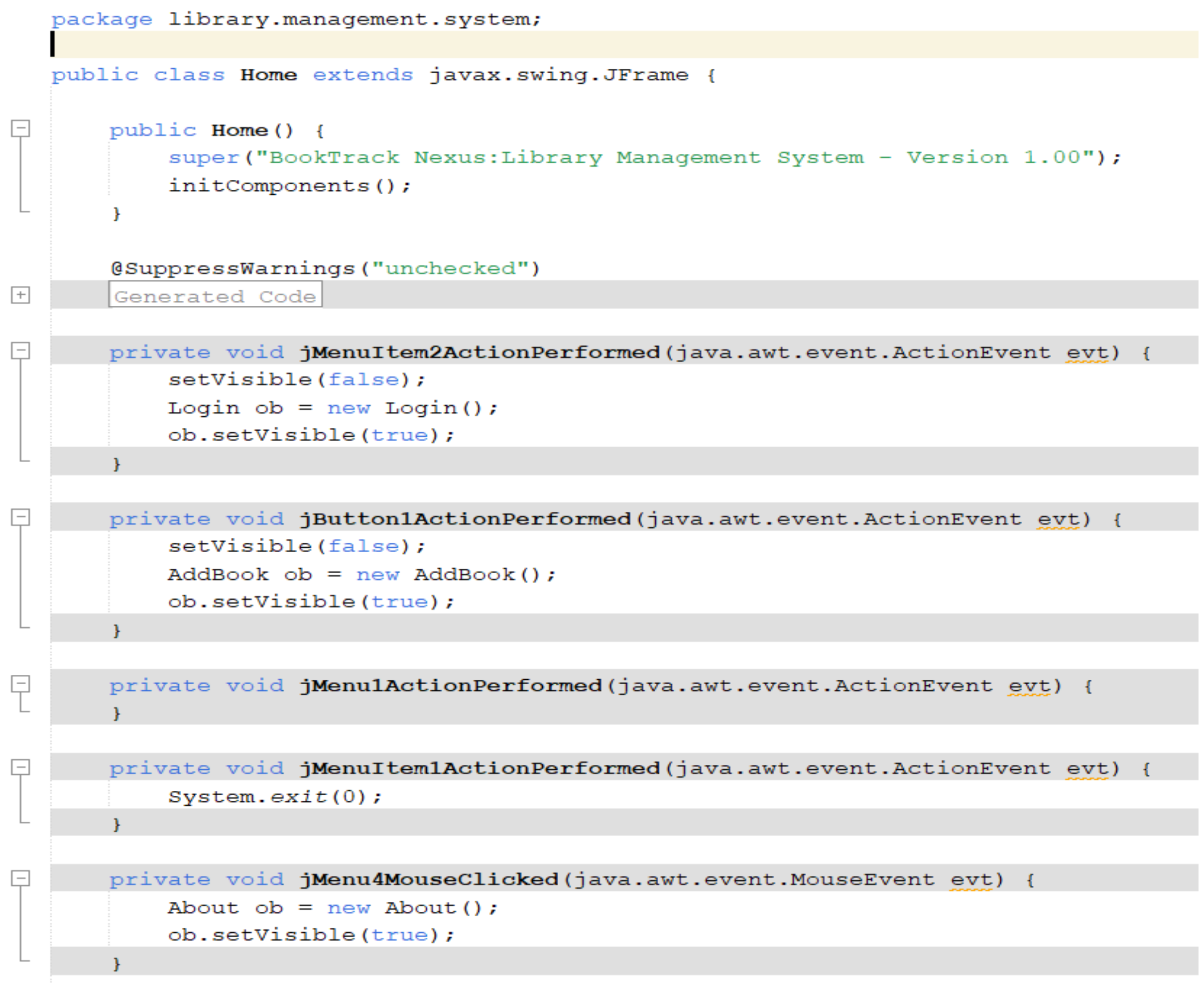
```

package library.management.system;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
import javax.swing.UIManager;

public class Login extends javax.swing.JFrame {
    Connection conn;
    ResultSet rs;
    PreparedStatement pst;

    public Login() {
        initComponents();
        conn = JavaConnect.ConnectDB();
    }

    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        setVisible(false);
        Register ob = new Register();
        ob.setVisible(true);
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        try{
        String sql = "select * from users where name= ? AND password= ? ";
        pst= conn.prepareStatement(sql);
        pst.setString(1, name.getText());
        pst.setString(2, password.~~getText~~());
        rs = pst.executeQuery();
        if(rs.next()){
```

- *Loading.java*



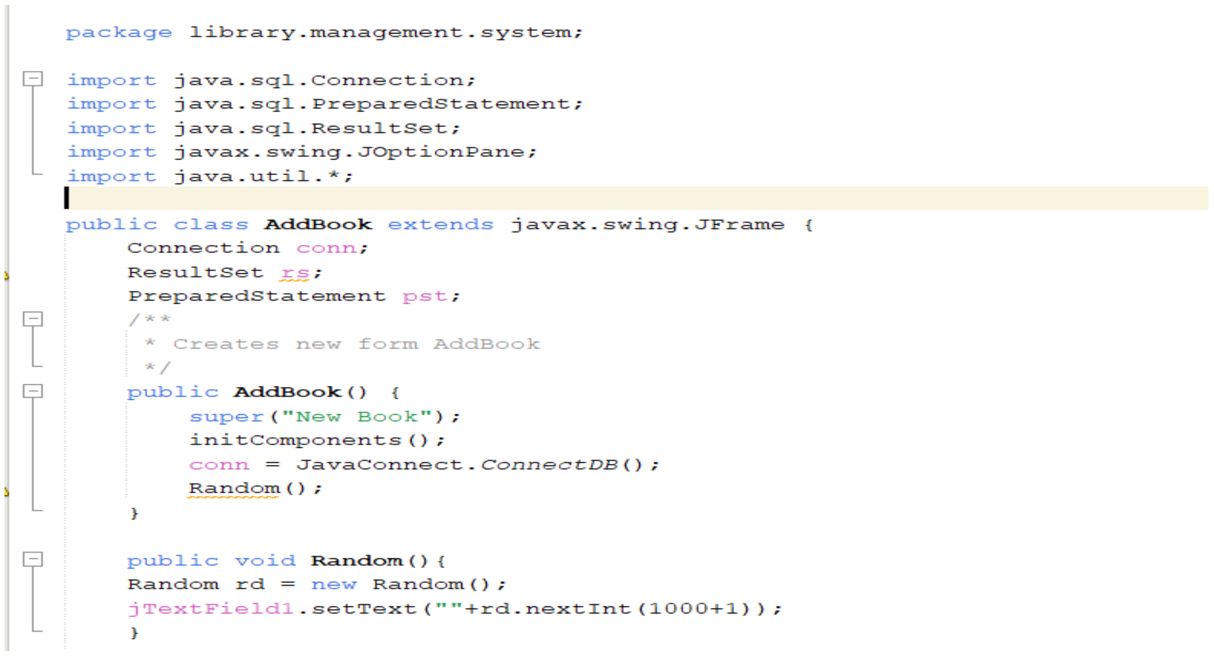
```

package library.management.system;

import java.sql.Connection;
import javax.swing.JOptionPane;

public class Loading extends javax.swing.JFrame implements Runnable{
    Connection conn;
    int s = 0;
    Thread th;

    public Loading() {
        super("Loading....");
        initComponents();
        th= new Thread((Runnable)this);

    }
```

- *Home.java*



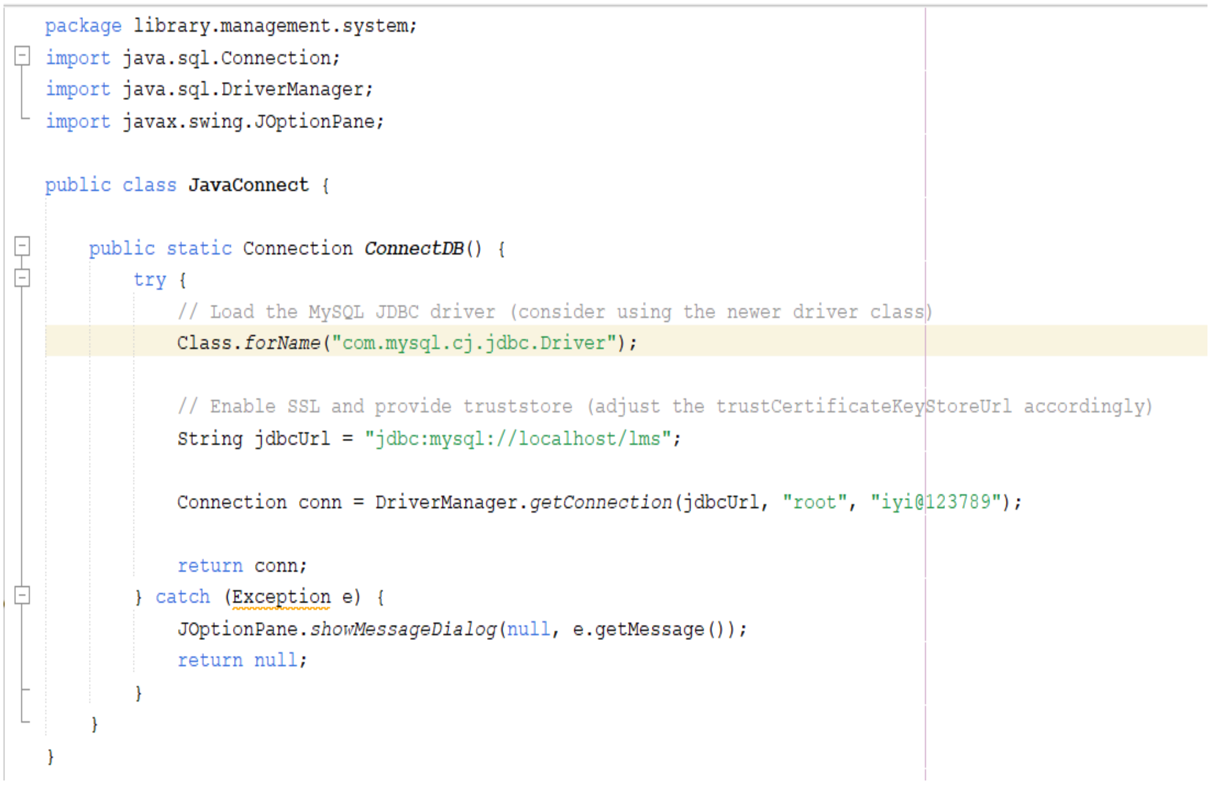
```
package library.management.system;

public class Home extends javax.swing.JFrame {

    public Home() {
        super("BookTrack Nexus:Library Management System - Version 1.00");
        initComponents();
    }

    @SuppressWarnings("unchecked")
    Generated Code

    private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
        setVisible(false);
        Login ob = new Login();
        ob.setVisible(true);
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        setVisible(false);
        AddBook ob = new AddBook();
        ob.setVisible(true);
    }

    private void jMenu1ActionPerformed(java.awt.event.ActionEvent evt) {
    }

    private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
        System.exit(0);
    }

    private void jMenu4MouseClicked(java.awt.event.MouseEvent evt) {
        About ob = new About();
        ob.setVisible(true);
    }
```

- *AddBook.java*



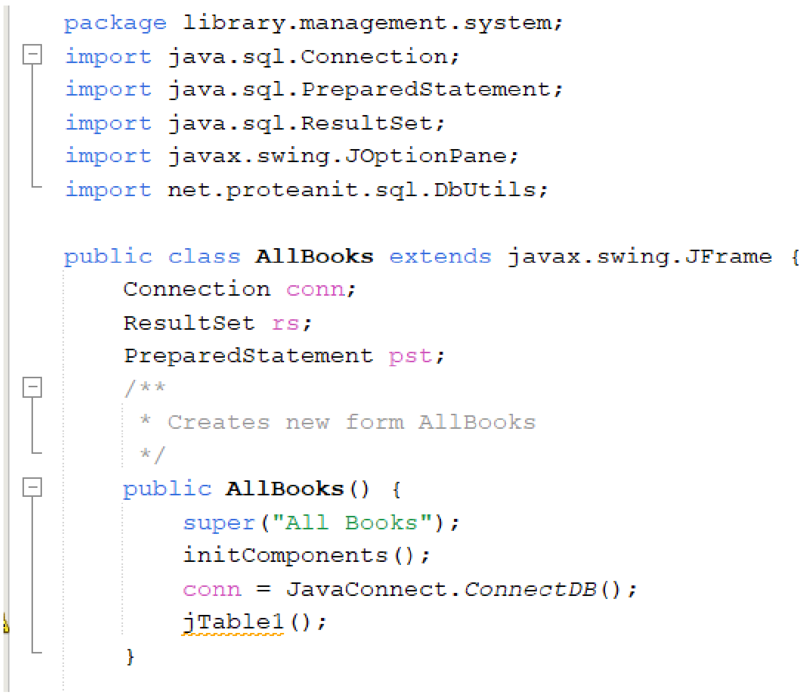
```
package library.management.system;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
import java.util.*;

public class AddBook extends javax.swing.JFrame {
    Connection conn;
    ResultSet rs;
    PreparedStatement pst;
    /**
     * Creates new form AddBook
     */
    public AddBook() {
        super("New Book");
        initComponents();
        conn = JavaConnect.ConnectDB();
        Random();
    }

    public void Random(){
    Random rd = new Random();
    jTextField1.setText(""+rd.nextInt(1000+1));
    }
```

- *JavaConnect.java*



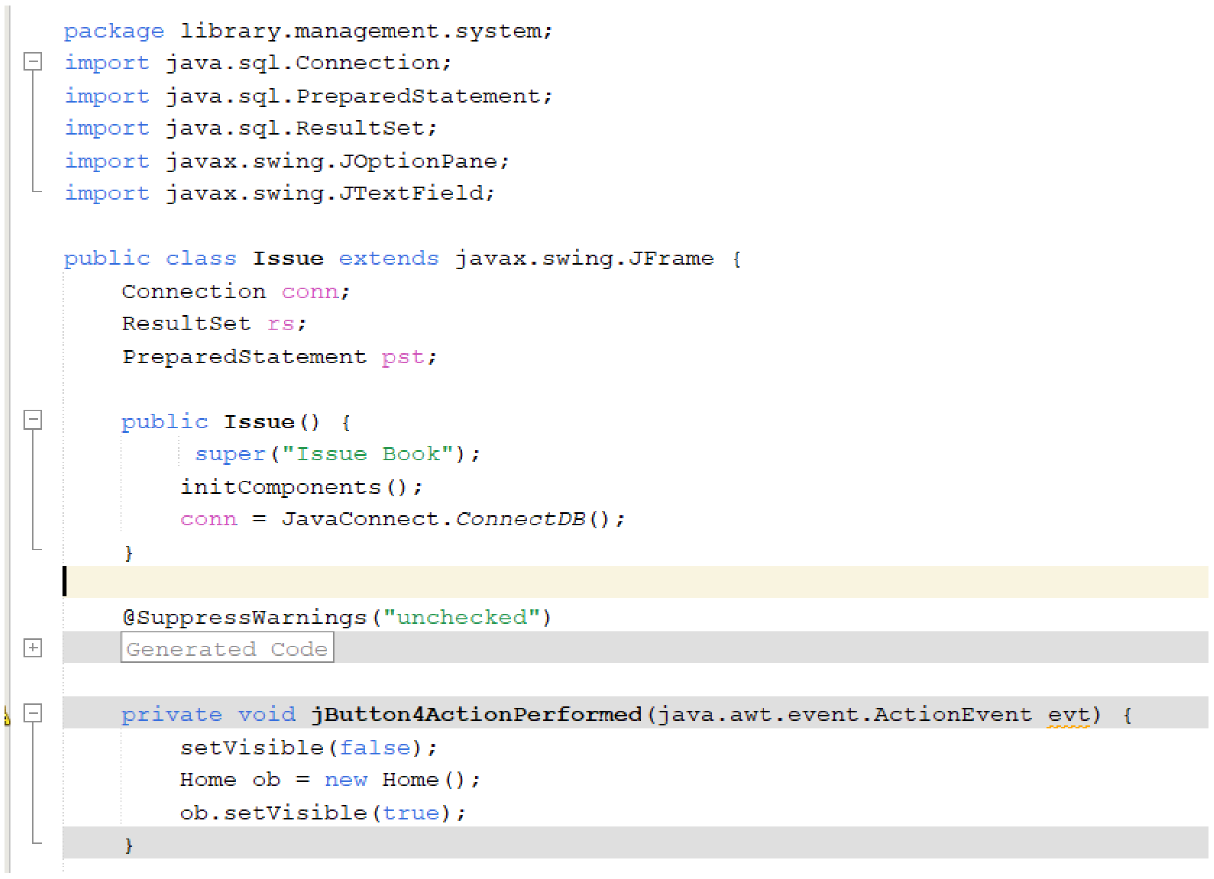
```java
package library.management.system;
import java.sql.Connection;
import java.sql.DriverManager;
import javax.swing.JOptionPane;

public class JavaConnect {

    public static Connection ConnectDB() {
        try {
            // Load the MySQL JDBC driver (consider using the newer driver class)
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Enable SSL and provide truststore (adjust the trustCertificateKeyStoreUrl accordingly)
            String jdbcUrl = "jdbc:mysql://localhost/lms";

            Connection conn = DriverManager.getConnection(jdbcUrl, "root", "iyi@123789");

            return conn;
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage());
            return null;
        }
    }
}
```

- *AllBooks.java*



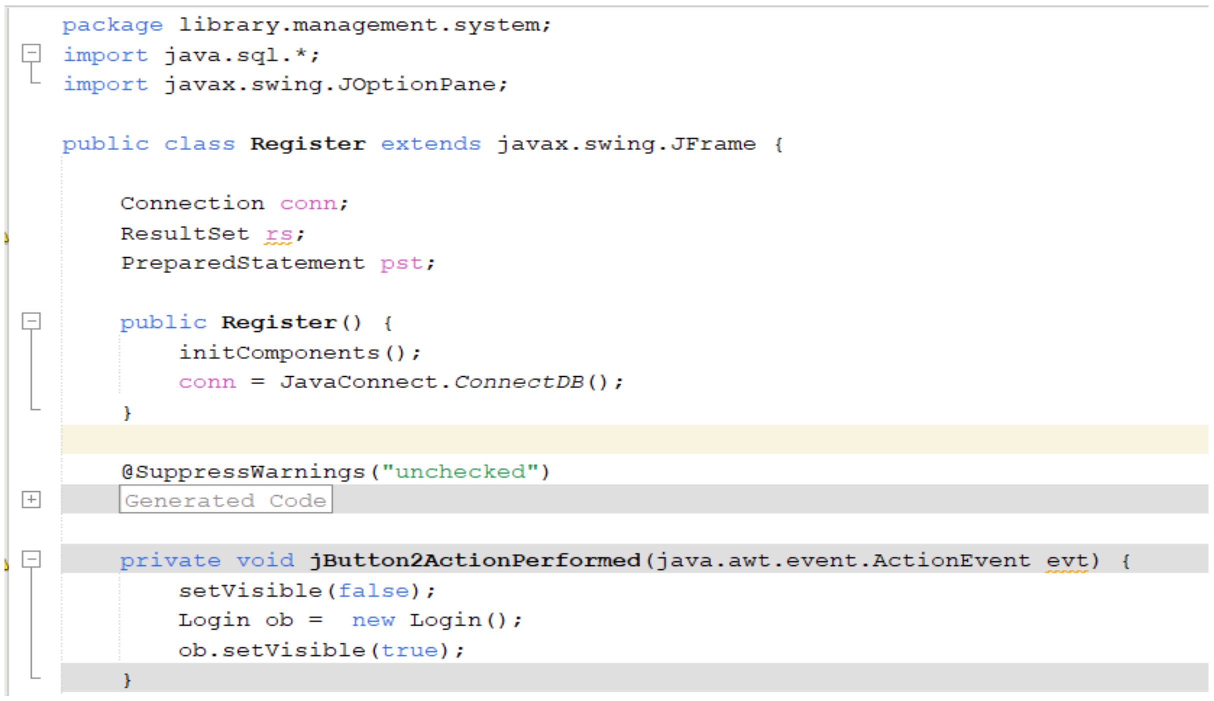
```java
package library.management.system;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
import net.proteanit.sql.DbUtils;

public class AllBooks extends javax.swing.JFrame {
    Connection conn;
    ResultSet rs;
    PreparedStatement pst;
    /**
     * Creates new form AllBooks
     */
    public AllBooks() {
        super("All Books");
        initComponents();
        conn = JavaConnect.ConnectDB();
        jTable1();
    }
```

- ***Issue.java***



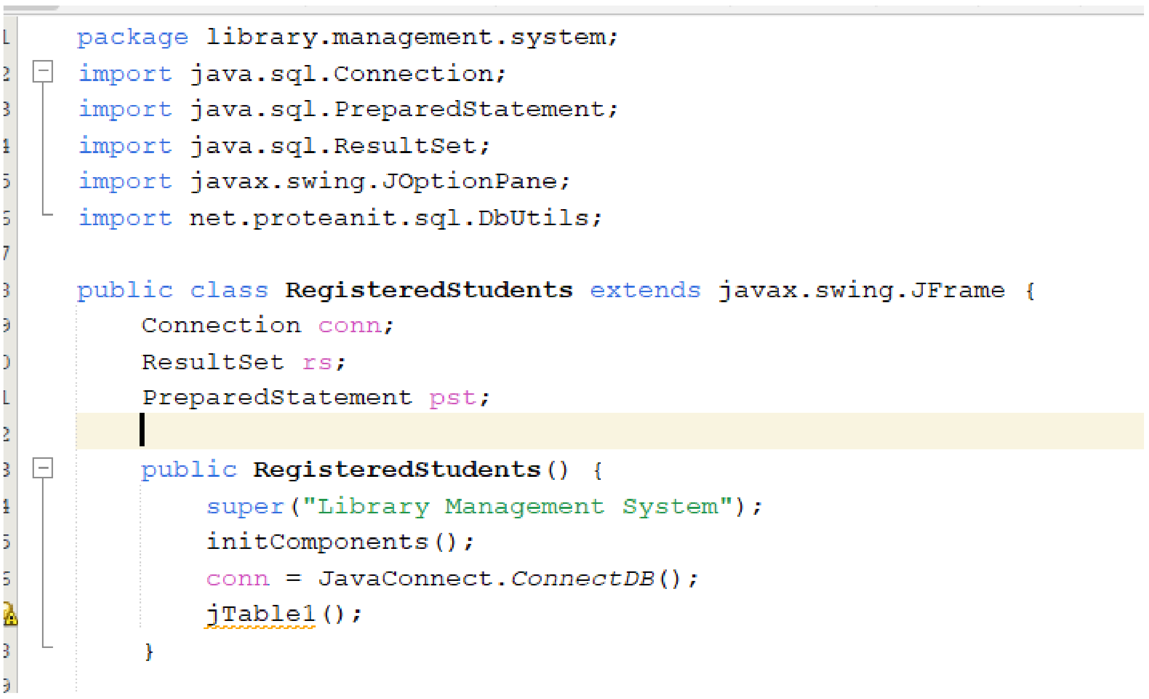
```java
package library.management.system;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

public class Issue extends javax.swing.JFrame {
    Connection conn;
    ResultSet rs;
    PreparedStatement pst;

    public Issue()  {
        super("Issue Book");
        initComponents();
        conn = JavaConnect.ConnectDB();
    }

    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt)  {
        setVisible(false);
        Home ob = new Home();
        ob.setVisible(true);
    }
```

- ***Register.java***



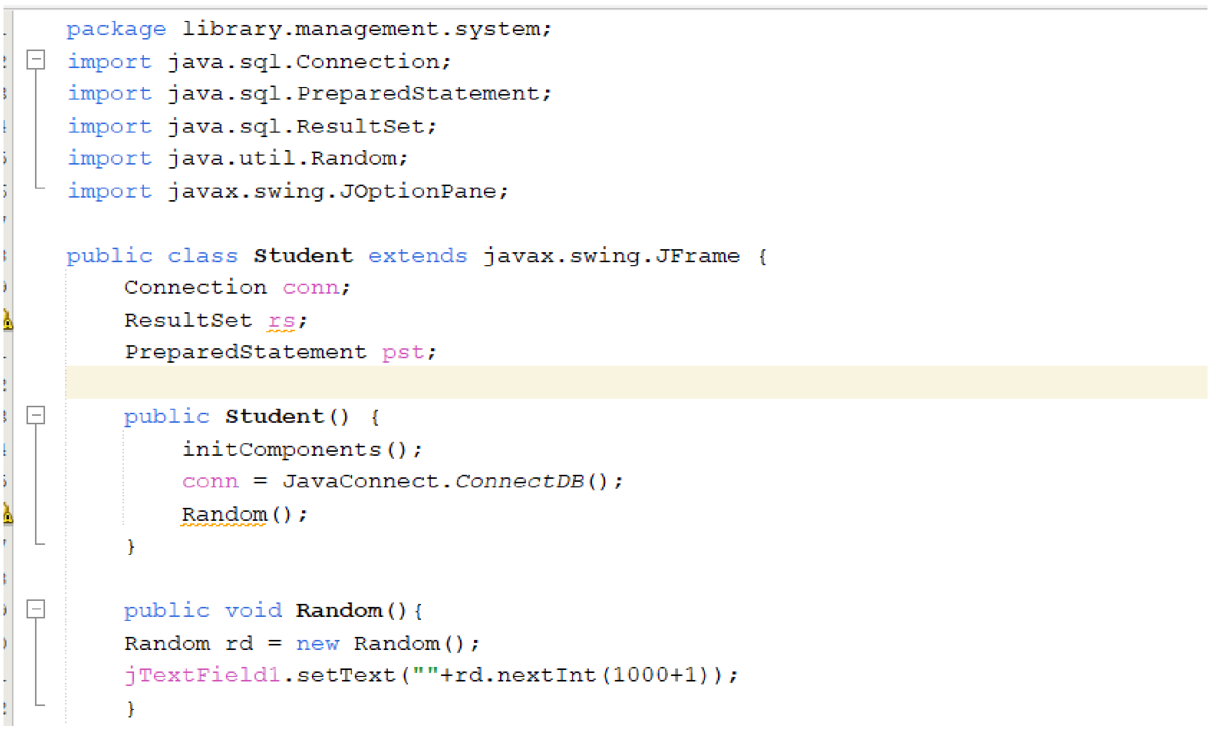
```java
package library.management.system;
import java.sql.*;
import javax.swing.JOptionPane;

public class Register extends javax.swing.JFrame {

    Connection conn;
    ResultSet rs;
    PreparedStatement pst;

    public Register()  {
        initComponents();
        conn = JavaConnect.ConnectDB();
    }

    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)  {
        setVisible(false);
        Login ob =  new Login();
        ob.setVisible(true);
    }
```

- *RegisteredStudents.java*



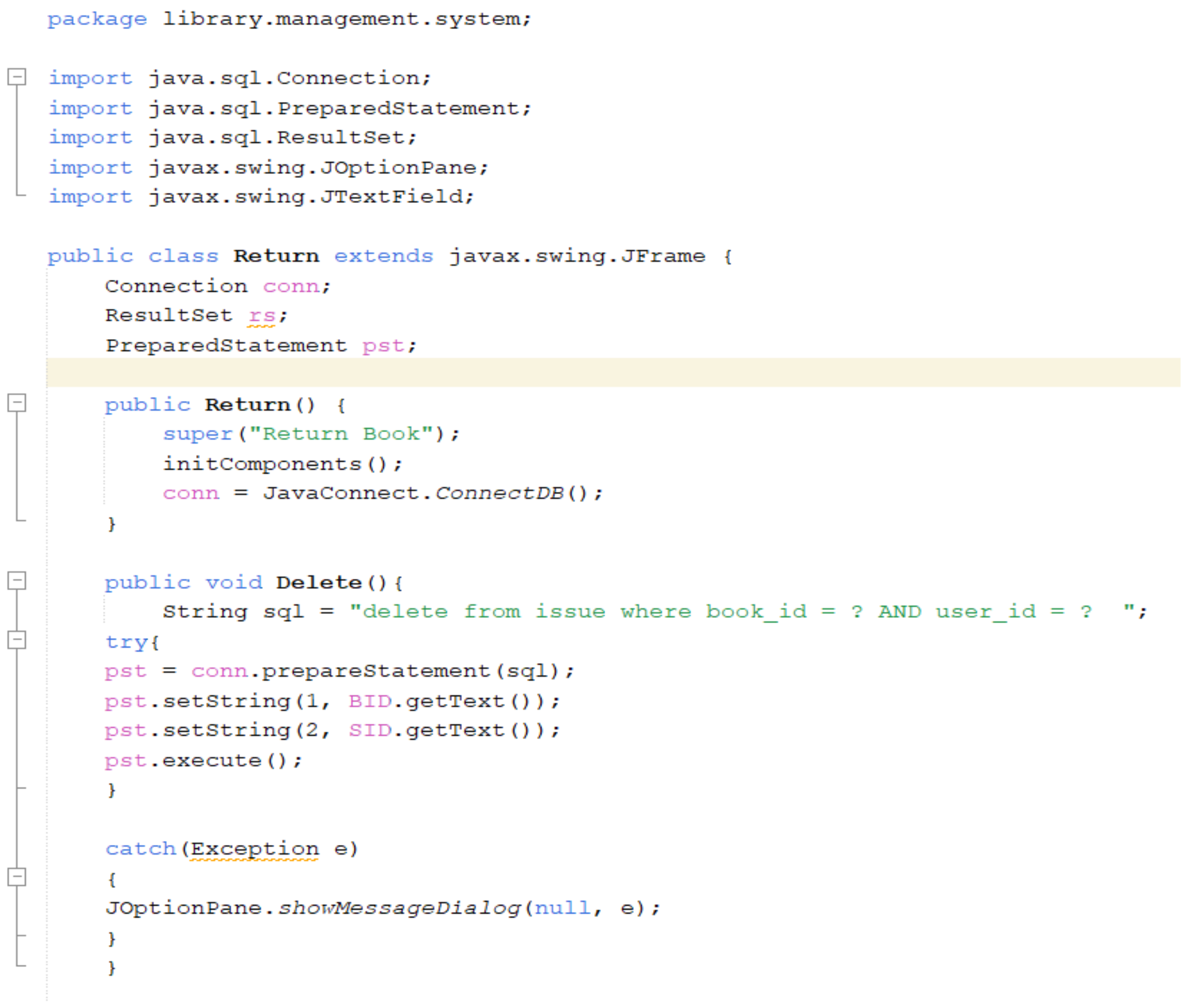
```java
package library.management.system;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
import net.proteanit.sql.DbUtils;

public class RegisteredStudents extends javax.swing.JFrame {
    Connection conn;
    ResultSet rs;
    PreparedStatement pst;

    public RegisteredStudents() {
        super("Library Management System");
        initComponents();
        conn = JavaConnect.ConnectDB();
        jTable1();
    }
```

- *Student.java*



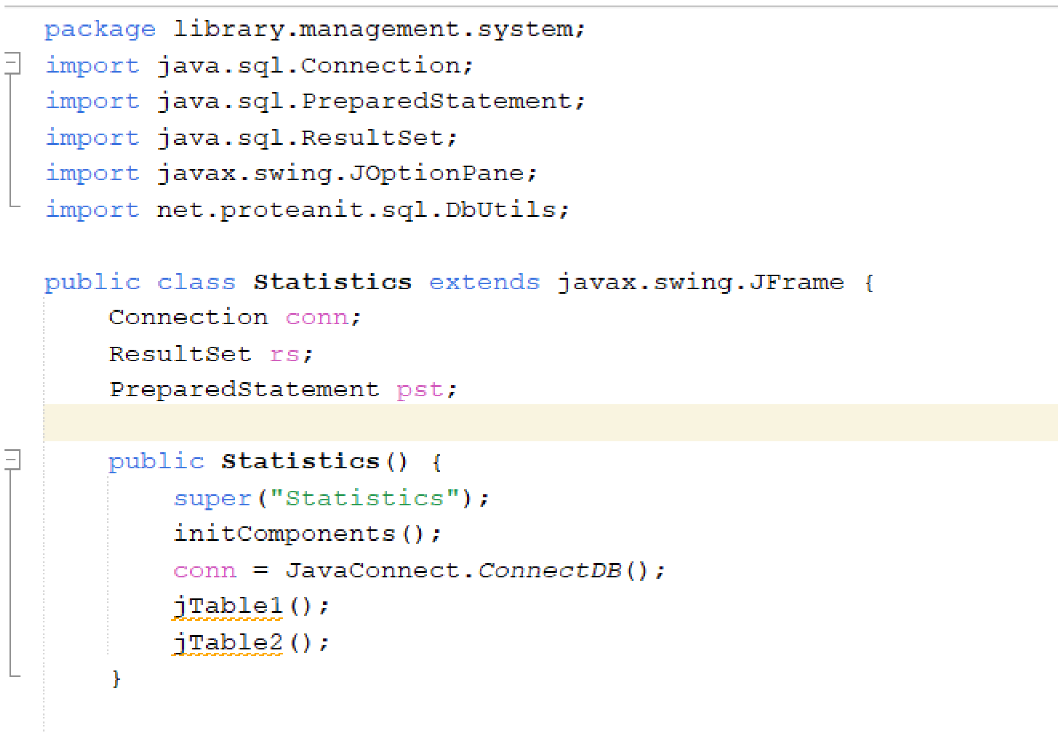
```java
package library.management.system;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Random;
import javax.swing.JOptionPane;

public class Student extends javax.swing.JFrame {
    Connection conn;
    ResultSet rs;
    PreparedStatement pst;

    public Student() {
        initComponents();
        conn = JavaConnect.ConnectDB();
        Random();
    }

    public void Random(){
    Random rd = new Random();
    jTextField1.setText(""+rd.nextInt(1000+1));
    }
```

- *Return.java*



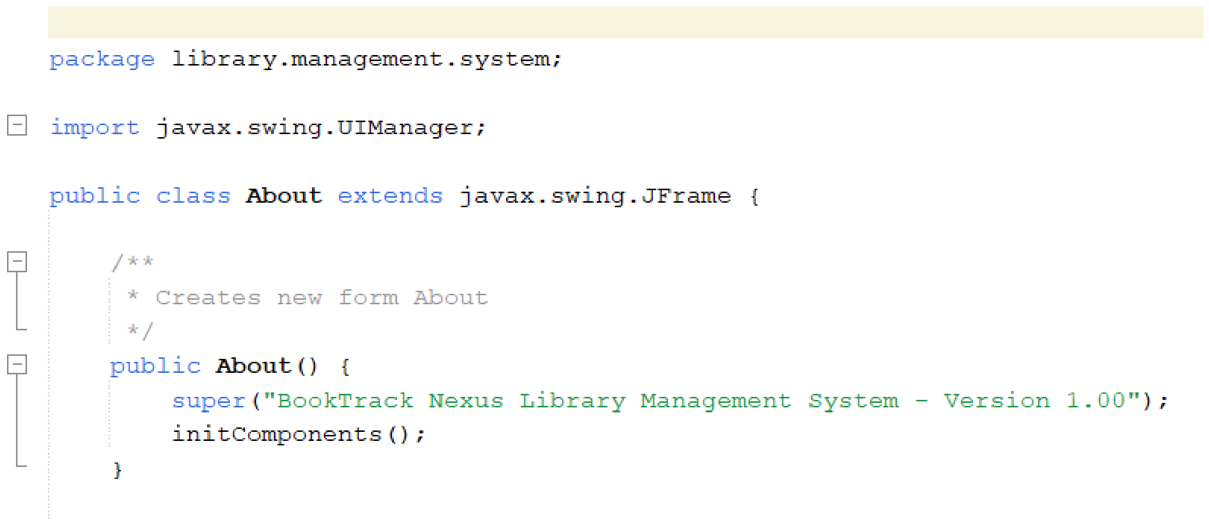
```java
package library.management.system;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

public class Return extends javax.swing.JFrame {
    Connection conn;
    ResultSet rs;
    PreparedStatement pst;

    public Return() {
        super("Return Book");
        initComponents();
        conn = JavaConnect.ConnectDB();
    }

    public void Delete(){
        String sql = "delete from issue where book_id = ? AND user_id = ?  ";
    try{
    pst = conn.prepareStatement(sql);
    pst.setString(1, BID.getText());
    pst.setString(2, SID.getText());
    pst.execute();
    }

    catch(Exception e)
    {
    JOptionPane.showMessageDialog(null, e);
    }
    }
```

- *Statistics.java*



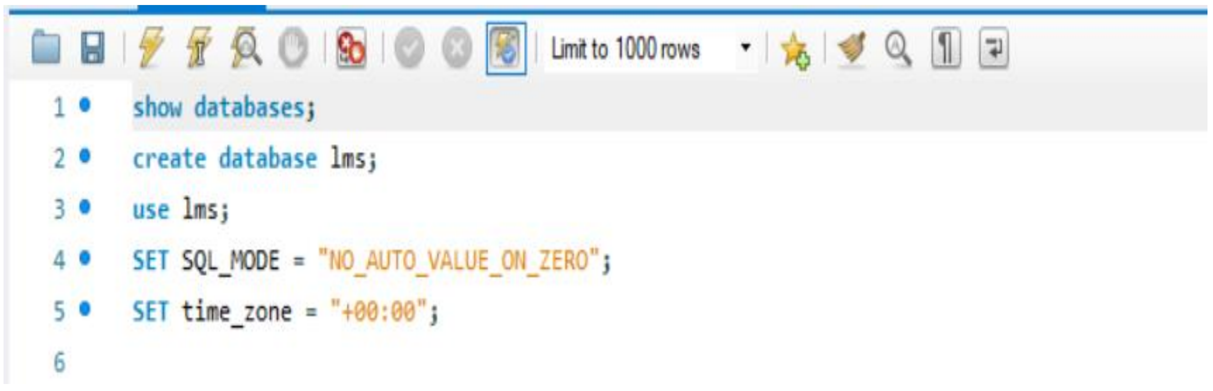
```java
package library.management.system;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;
import net.proteanit.sql.DbUtils;

public class Statistics extends javax.swing.JFrame {
    Connection conn;
    ResultSet rs;
    PreparedStatement pst;

    public Statistics() {
        super("Statistics");
        initComponents();
        conn = JavaConnect.ConnectDB();
        jTable1();
        jTable2();
    }
```

- *About.java*

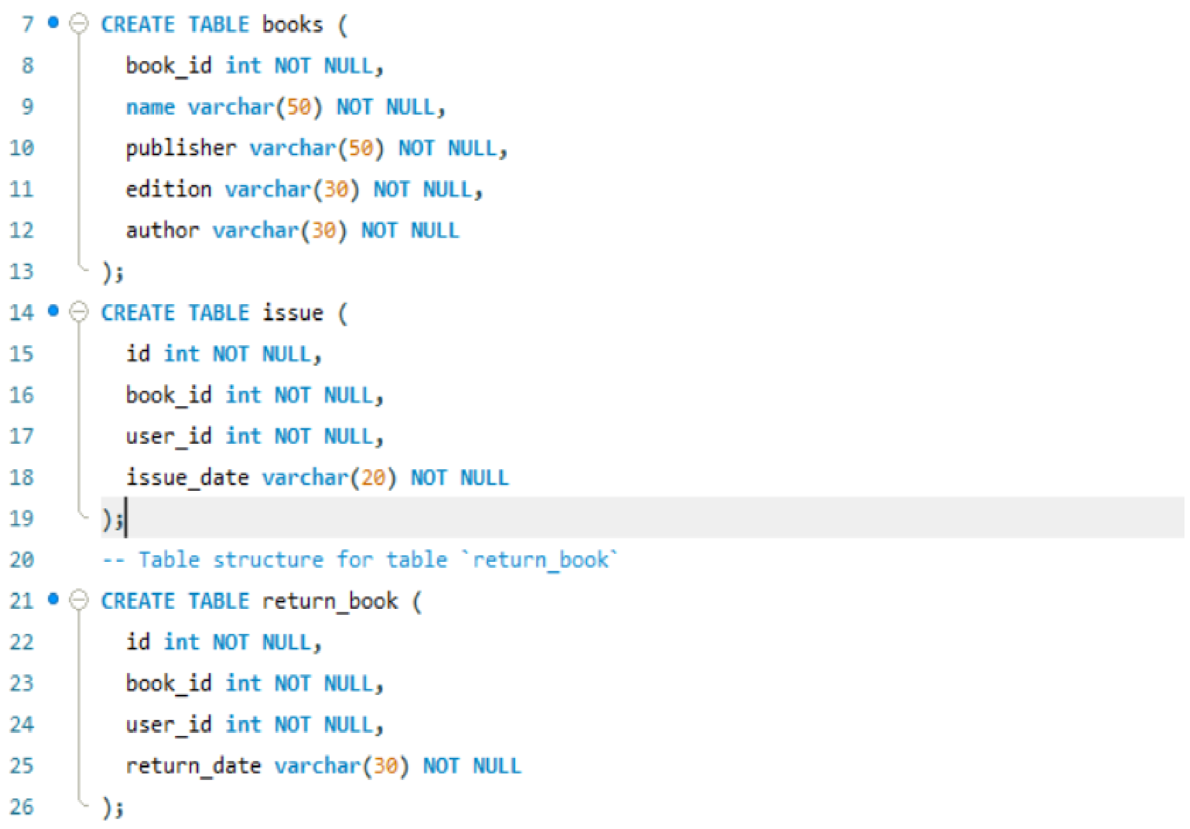


```java
package library.management.system;

import javax.swing.UIManager;

public class About extends javax.swing.JFrame {

    /**
     * Creates new form About
     */
    public About() {
        super("BookTrack Nexus Library Management System - Version 1.00");
        initComponents();
    }
```

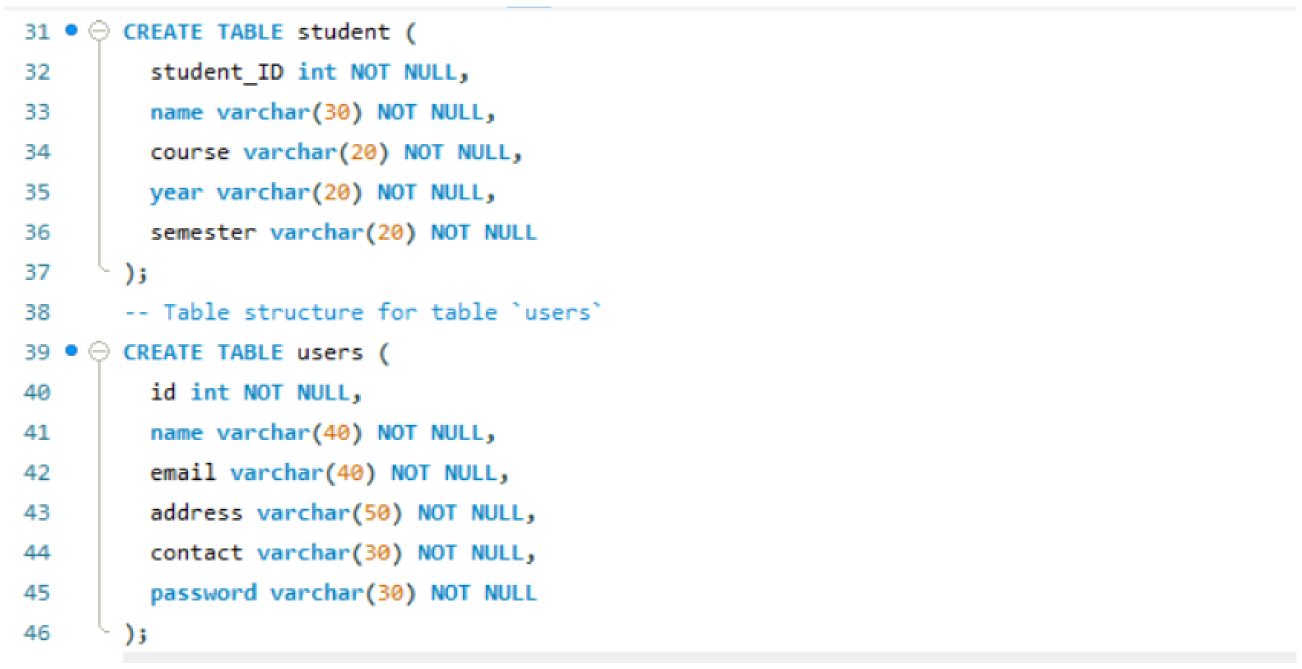
MySQL Database Queries



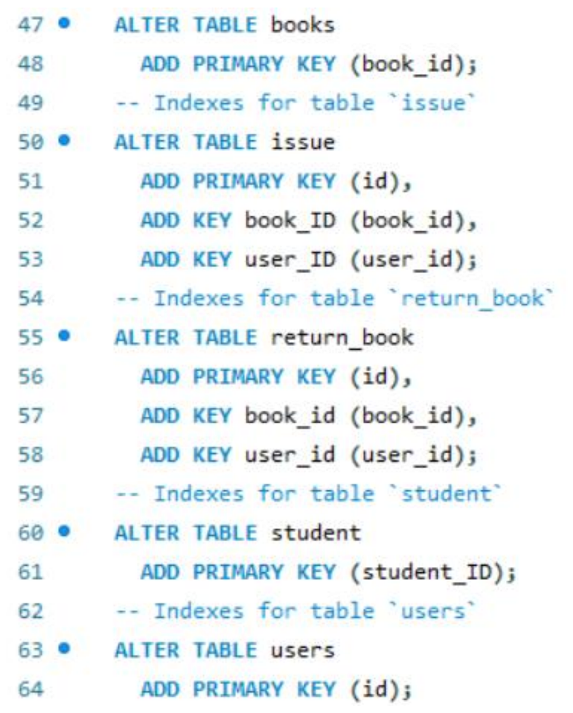
```sql
show databases;
create database lms;
use lms;
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";

```



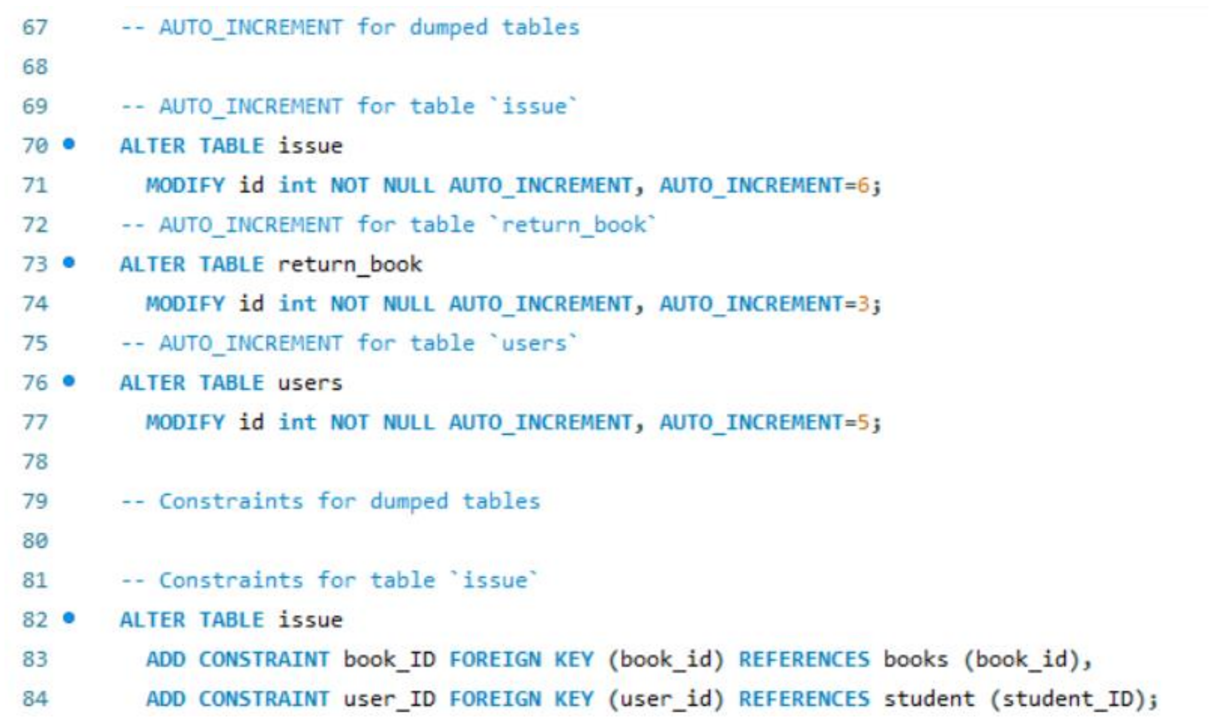
```sql
CREATE TABLE books (
  book_id int NOT NULL,
  name varchar(50) NOT NULL,
  publisher varchar(50) NOT NULL,
  edition varchar(30) NOT NULL,
  author varchar(30) NOT NULL
);
CREATE TABLE issue (
  id int NOT NULL,
  book_id int NOT NULL,
  user_id int NOT NULL,
  issue_date varchar(20) NOT NULL
);
-- Table structure for table `return_book`
CREATE TABLE return_book (
  id int NOT NULL,
  book_id int NOT NULL,
  user_id int NOT NULL,
  return_date varchar(30) NOT NULL
);
```



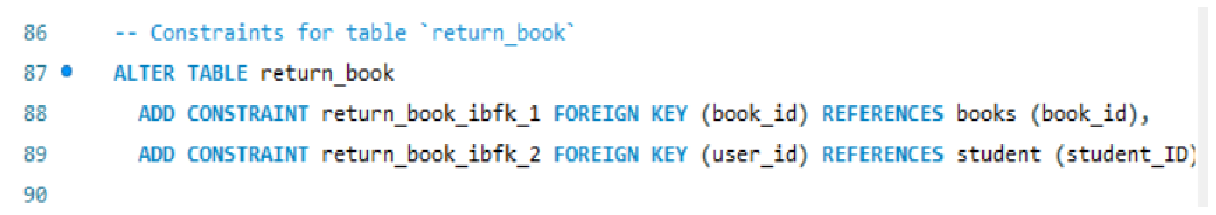
```sql
CREATE TABLE student (
  student_ID int NOT NULL,
  name varchar(30) NOT NULL,
  course varchar(20) NOT NULL,
  year varchar(20) NOT NULL,
  semester varchar(20) NOT NULL
);
-- Table structure for table `users`
CREATE TABLE users (
  id int NOT NULL,
  name varchar(40) NOT NULL,
  email varchar(40) NOT NULL,
  address varchar(50) NOT NULL,
  contact varchar(30) NOT NULL,
  password varchar(30) NOT NULL
);
```



```sql
ALTER TABLE books
  ADD PRIMARY KEY (book_id);
-- Indexes for table `issue`
ALTER TABLE issue
  ADD PRIMARY KEY (id),
  ADD KEY book_ID (book_id),
  ADD KEY user_ID (user_id);
-- Indexes for table `return_book`
ALTER TABLE return_book
  ADD PRIMARY KEY (id),
  ADD KEY book_id (book_id),
  ADD KEY user_id (user_id);
-- Indexes for table `student`
ALTER TABLE student
  ADD PRIMARY KEY (student_ID);
-- Indexes for table `users`
ALTER TABLE users
  ADD PRIMARY KEY (id);
```



```sql
-- AUTO_INCREMENT for dumped tables

-- AUTO_INCREMENT for table `issue`
ALTER TABLE issue
  MODIFY id int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;
-- AUTO_INCREMENT for table `return_book`
ALTER TABLE return_book
  MODIFY id int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=3;
-- AUTO_INCREMENT for table `users`
ALTER TABLE users
  MODIFY id int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;

-- Constraints for dumped tables

-- Constraints for table `issue`
ALTER TABLE issue
  ADD CONSTRAINT book_ID FOREIGN KEY (book_id) REFERENCES books (book_id),
  ADD CONSTRAINT user_ID FOREIGN KEY (user_id) REFERENCES student (student_ID);
```



```sql
-- Constraints for table `return_book`
ALTER TABLE return_book
  ADD CONSTRAINT return_book_ibfk_1 FOREIGN KEY (book_id) REFERENCES books (book_id),
  ADD CONSTRAINT return_book_ibfk_2 FOREIGN KEY (user_id) REFERENCES student (student_ID)

```

Explanation:

1.Database Initialization:

The initial command "SHOW DATABASES;" lists existing databases.

"CREATE DATABASE LMS;" creates a new database named "lms."

"USE LMS;" selects the "lms" database for subsequent operations.

2. SQL Configuration:

"SET SQL_MODE = 'NO_AUTO_VALUE_ON_ZERO';" configures SQL mode to disallow automatic increment for zero values.

"SET time_zone = '+00:00';" sets the time zone for the database to UTC.

3. Books Table:

"CREATE TABLE books" defines the structure for the 'books' table with fields: book_id, name, publisher, edition, and author.

"ALTER TABLE books ADD PRIMARY KEY (book_id);" establishes 'book_id' as the primary key.

4. Issue Table:

"CREATE TABLE issue" defines the 'issue' table with fields: id, book_id, user_id, and issue_date.

"ALTER TABLE issue" adds primary and foreign key constraints linking 'book_id' to 'books' and 'user_id' to 'student.'

5. Return Book Table:

"CREATE TABLE return_book" defines the 'return_book' table with fields: id, book_id, user_id, and return_date.

"ALTER TABLE return_book" adds primary and foreign key constraints linking 'book_id' to 'books' and 'user_id' to 'student.'

6. Student Table:

"CREATE TABLE student" defines the 'student' table with fields: student_ID, name, course, year, and semester.

"ALTER TABLE student ADD PRIMARY KEY (student_ID);" establishes 'student_ID' as the primary key.

7. Users Table:

"CREATE TABLE users" defines the 'users' table with fields: id, name, email, address, contact, and password.

"ALTER TABLE users ADD PRIMARY KEY (id);" sets 'id' as the primary key.

8. Auto-increment and Constraints:

"ALTER TABLE issue MODIFY id" and "ALTER TABLE return_book MODIFY id" configure auto-increment for 'id' in the 'issue' and 'return_book' tables, respectively.

"ALTER TABLE users MODIFY id" sets auto-increment for 'id' in the 'users' table.

Foreign key constraints ensure data integrity, linking 'book_id' and 'user_id' in 'issue' and 'return_book' tables to 'books' and 'student' tables, respectively.

▪ Detailed Implementation:

• *Technologies Used*

Java Swing (GUI Development)

Java Swing is employed to create a graphical user interface (GUI) for the Library Management System. It enables the development of interactive and visually appealing interfaces, enhancing the user experience.

MySQL Workbench (Database Management)

MySQL Workbench serves as the database management system for the application. It facilitates the creation and management of tables storing crucial information about books and users. Key concepts include relational tables, establishing relationships, and optimizing database performance through efficient indexing.

- ***Database Concepts***

The application utilizes MySQL Workbench for efficient database management. Key database concepts include:

Relational Tables

Structured tables are designed to store information about books and users. This relational approach ensures data integrity and facilitates efficient data retrieval.

Relationships Between Tables

Establishing relationships between tables, such as a one-to-many relationship between books and transactions, ensures the coherence of the database and supports complex queries.

Efficient Indexing for Faster Retrieval

Strategic indexing is implemented to enhance data retrieval speed. Indexing allows the system to locate and retrieve specific records more swiftly, contributing to overall system performance.

▪ *Data Structures and Algorithms*

Arrays and Lists

Arrays and lists are fundamental data structures used for storing and managing information about books and users. They enable efficient storage and retrieval operations, ensuring optimal performance.

Sorting Algorithms

Sorting algorithms are implemented to arrange book records in ascending order by title or author. This enhances the readability of the data and facilitates quicker search operations.

- ***Additional Data Structures***

Hash Tables: Hash tables can be utilized for rapid search operations, especially when looking up information based on unique identifiers like book ISBNs. This ensures constant-time complexity for retrieval, enhancing search efficiency.

Trees (e.g., Binary Search Trees): Binary Search Trees (BST) can be employed for maintaining a sorted order of records. For instance, a binary search tree based on book titles allows for efficient insertion, deletion, and search operations while preserving a hierarchical structure.

Queues and Stacks: Queues and stacks can be useful in managing transactions. For instance, a queue can represent a first-in, first-out (FIFO) system for book borrowings, ensuring fairness in serving requests. Stacks may be used for tracking the history of returned books.

Lists: Linked lists can be utilized to store and manage data dynamically. This flexibility is particularly useful for scenarios where the number of books or users may change frequently. A linked list of transactions, for example, provides an adaptable structure for handling borrowing and return history.

- ***DSA Concepts***

Object-Oriented Programming (OOP)

The system leverages Object-Oriented Programming principles for code organization and modularity. Classes are utilized for encapsulation and abstraction, promoting a modular and maintainable codebase.

Efficient Search Operations

Arrays and Lists for Search

Data structures like arrays and lists are employed for efficient search operations. The system allows users to search for books based on title, author, or ISBN, leveraging these data structures for quick and accurate results.

Algorithmic Search Implementations

Algorithmic search techniques are implemented to enhance search functionality. These may include binary search for sorted data structures, ensuring optimal search efficiency.

Input and Output Specifications

Input

I. User input via the GUI for operations like adding books, issuing books, etc.
II. Data input from the database for displaying information about books, users, and transactions.

Output

I. Displaying information on the GUI based on user queries and system operations.
II. Transaction records stored in the database for future reference.

▪ Usage of OOP Concepts in Library Management System

1. Classes and Objects:

Different classes, such as Book, User, Issue, Return, and JavaConnect, represent entities within the library management system.

Objects of these classes are instantiated to represent individual books, users, transactions, and database connections.

2. Inheritance:

Inheritance establishes relationships between classes in the library management system.

A base class like LibraryItem can be created, extended by subclasses like Book, Issue, and Return to inherit common attributes and behaviors related to library items.

3. Encapsulation:

Encapsulation is applied to classes to encapsulate data and functionality within the library management system.

For example, the Book class encapsulates book-specific details like name, author, and publisher, providing methods to access and modify this information securely.

4. Method Overriding:

Method overriding is employed when there is a need to provide custom implementations of inherited methods in the library management system.

For instance, the displayDetails() method in the Book class can override the same method in the LibraryItem class to display book-specific information.

5. Constructor:

Constructors are utilized in various classes of the library management system to initialize object properties when they are created.

For example, a constructor in the Issue class can be used to set initial values such as issue ID, book ID, user ID, and issue date.

6. Polymorphism:

Polymorphism is applied in the library management system to treat objects of different classes as objects of a common superclass or interface.

For flexibility and code reusability, a method that processes book transactions can accept both Book and Issue objects, treating them as instances of a common LibraryItem superclass.

By leveraging these OOP principles, the library management system project ensures a well-structured and maintainable codebase, facilitating efficient management of library resources and user interactions

Future Enhancements for Library Management System

1. User Roles and Permissions:

Implement role-based access control for librarians, administrators, and users.

2. Advanced Search and Filtering:

Improve search functionality with advanced filters like genre and publication date.

3. Integration with External APIs:

Integrate external book databases or library catalog APIs for real-time updates.

4. Mobile Application Development:

Develop a mobile app for on-the-go access to the library system.

5. RFID Technology Integration:

Automate book transactions with RFID technology.

6. User Notifications and Alerts:

Implement notifications for due dates, overdue books, and announcements.

7. Data Analytics and Reporting:

Introduce analytics for user behavior and library usage trends.

8. Integration with Learning Management Systems:

Sync with educational institutions' learning platforms for seamless access.

9. Fine Management System:

Automate fine calculations and provide online payment options.

▪ Output and Execution Flow in Library Management System

1. Splash Screen (Main_Window.java): Upon launching the Library Management System, users are greeted with a visually appealing splash screen, providing a brief and engaging introduction to the application. The splash screen serves as an initial point of interaction and sets the tone for the user experience.

2. Login Window (Login.java and Register.java): Following the splash screen, users are directed to the login window, where they can authenticate themselves to access the library system. This window also provides the option for new users to register. The registration process ensures that users can sign up securely, expanding the user base and enhancing system accessibility.

3. Loading Screen (Loading.java): Upon successful login or registration, users are directed to a loading screen. This screen indicates that the system is initializing and preparing for user interaction. It ensures a smooth transition between authentication and the main application, providing a seamless user experience.

4. Home Screen (Home.java): After the loading process, users are presented with the main home screen, featuring a dashboard that serves as a central hub for various library management operations. The home screen is designed with user-friendly navigation, allowing users to access functionalities like adding students, adding books, viewing transaction history, searching, sorting, issuing and returning books, signing out, accessing information about the application and the developer, and exiting the system.

5. Dashboard Operations:

Add Student and Add Book: Users can input information to add new students and books to the system, contributing to an updated and comprehensive database.

View Transaction History: The transaction history feature enables users to view a detailed log of all library transactions, providing transparency and accountability.

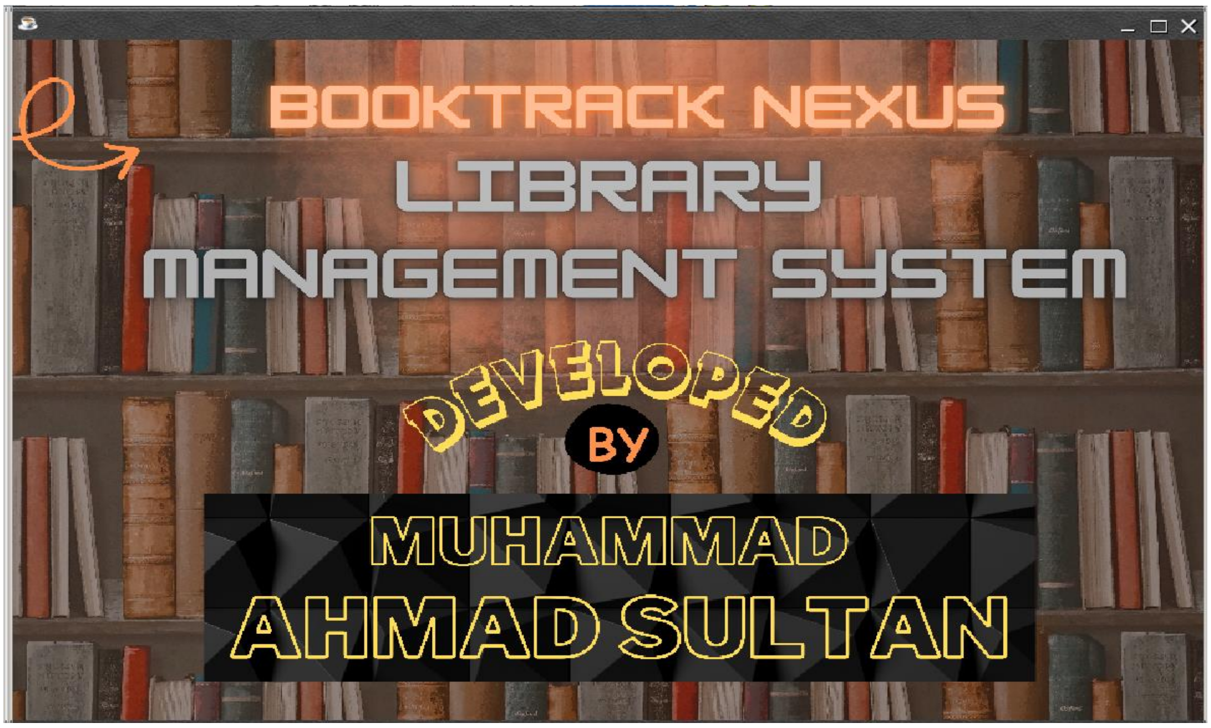
Search and Sort: The search and sort functionalities allow users to quickly locate specific books or students based on various criteria, enhancing the efficiency of information retrieval.

Issue and Return Books: Users can seamlessly manage the borrowing and returning processes, ensuring accurate tracking of book availability and transaction records.

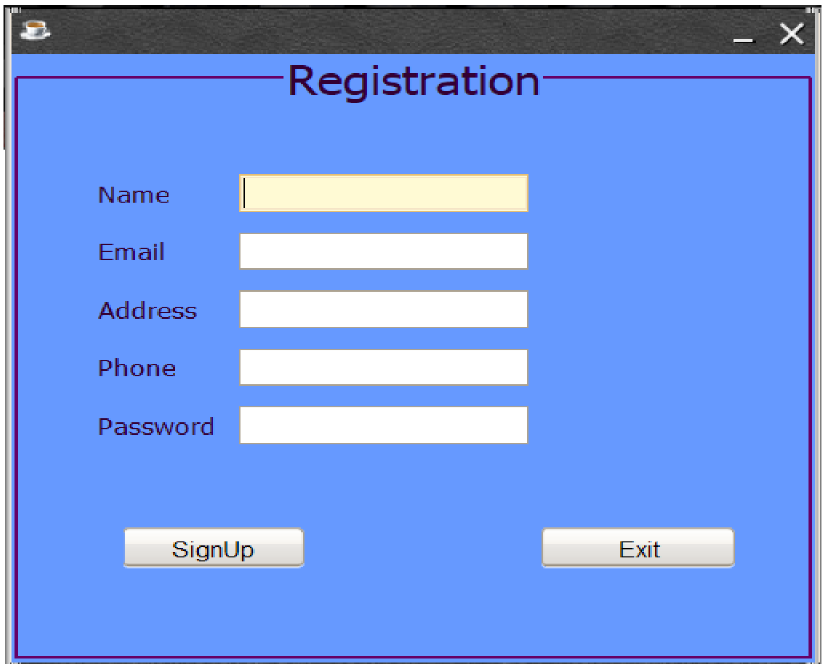
About and Exit: The dashboard provides information about the application and the developer, creating a user-friendly environment. Users can also choose to exit the system when done with their library tasks.

- **Output Display:**

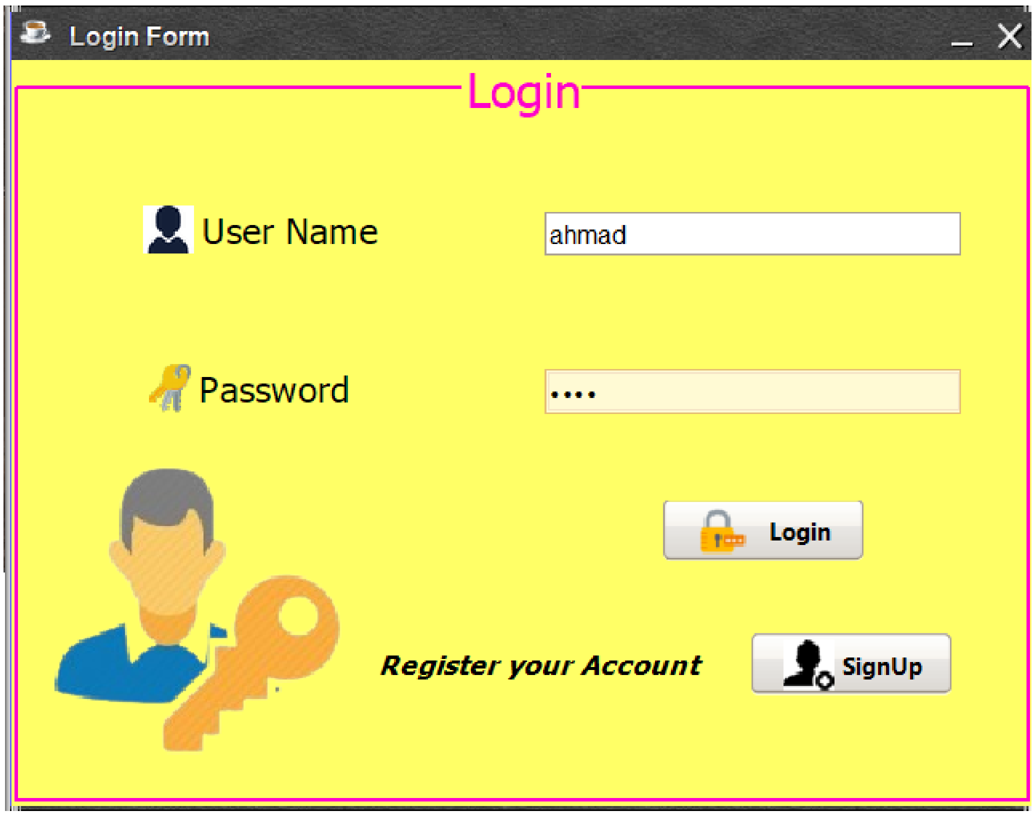
  - *Main_Window.java*



  - *Register.java*



❖ *Login.java*

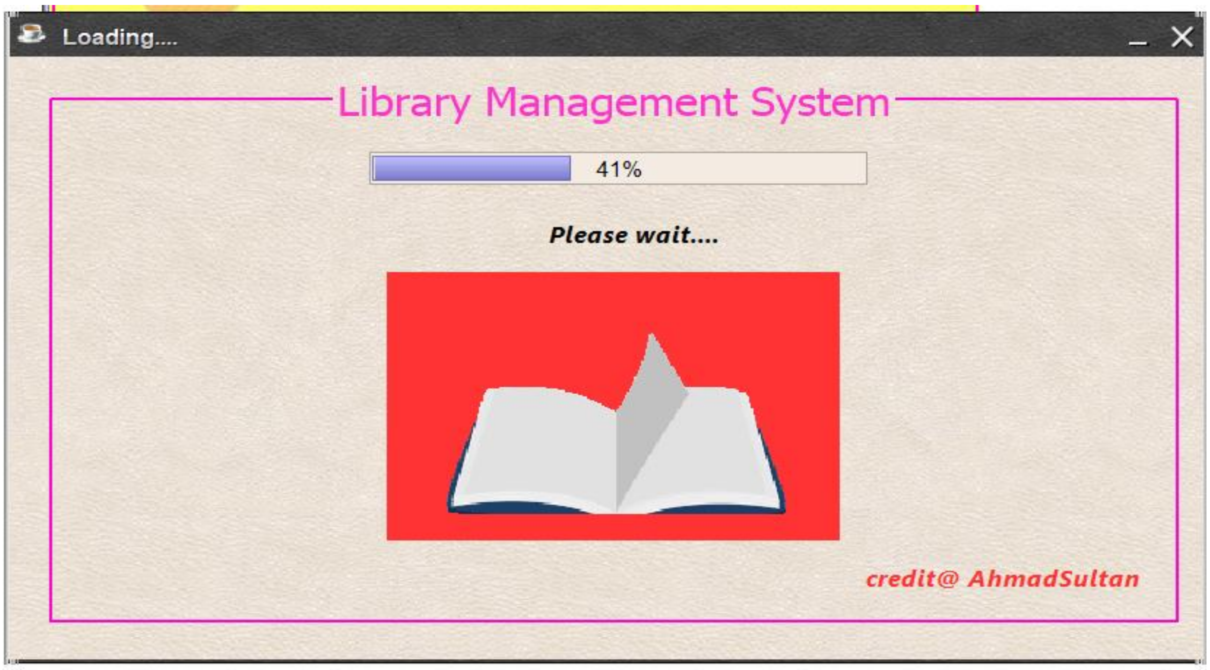


User Name: ahmad

Password: ••••

Register your Account

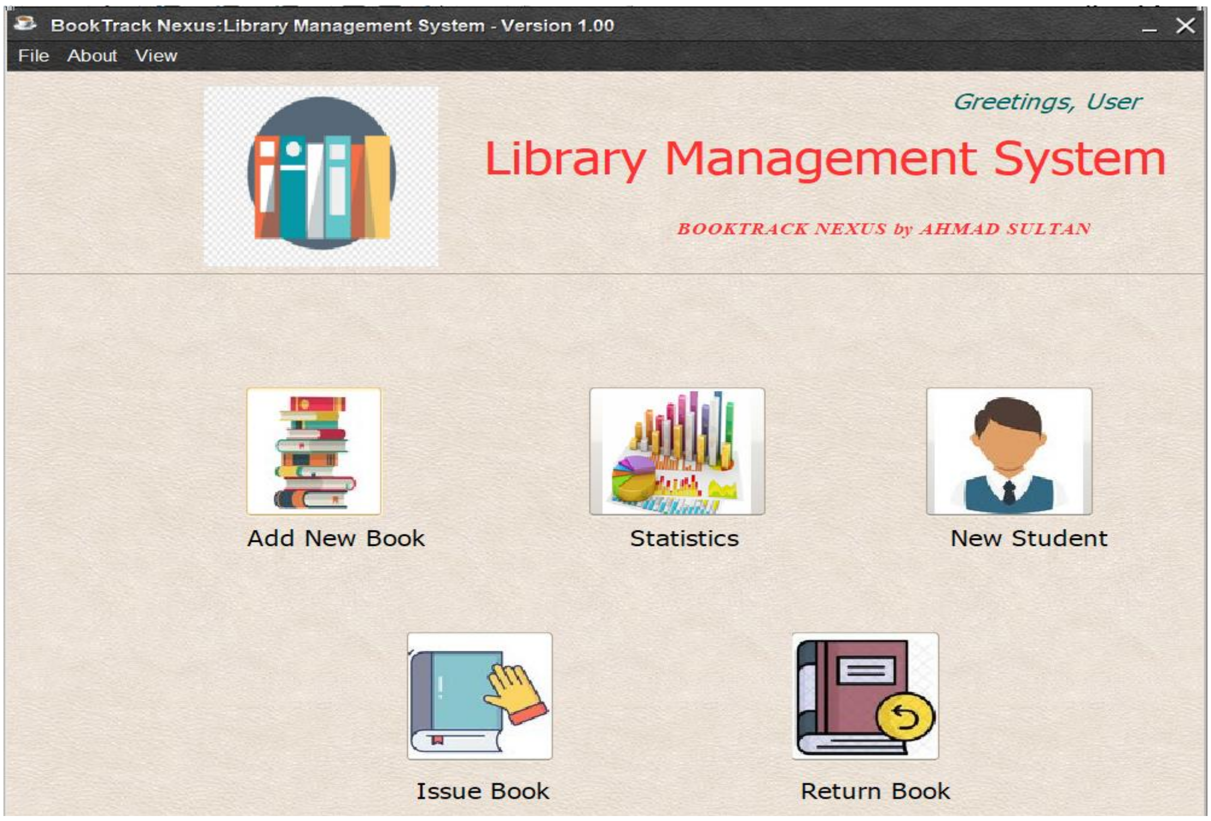
❖ *Loading.java*



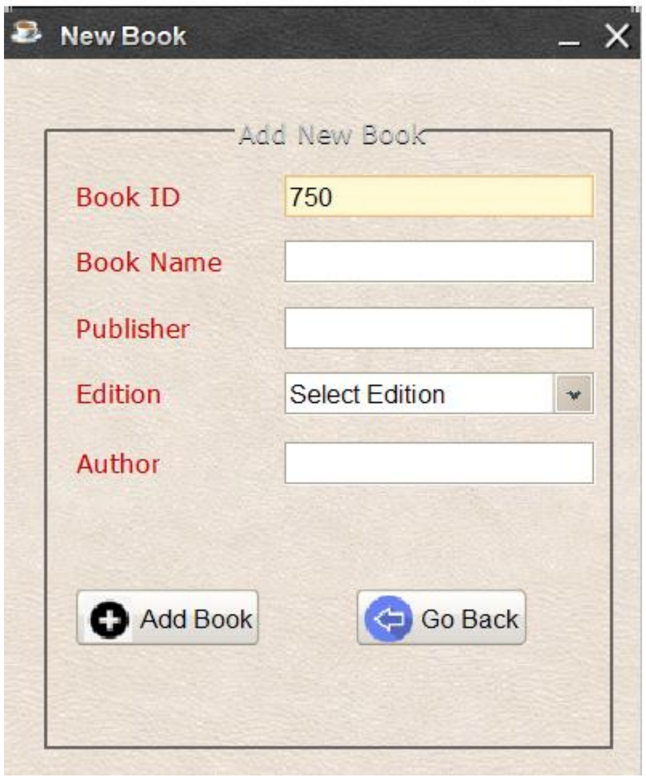
Please wait....

credit@ AhmadSultan

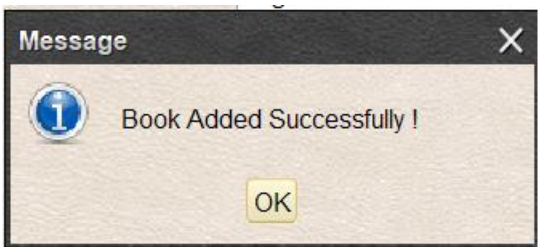
❖ *Home.java*


BookTrack Nexus:Library Management System - Version 1.00
File
About
View
Greetings, User
Library Management System
BOOKTRACK NEXUS by AHMAD SULTAN
Add New Book
Statistics
New Student
Issue Book
Return Book

❖ *AddBook.java*



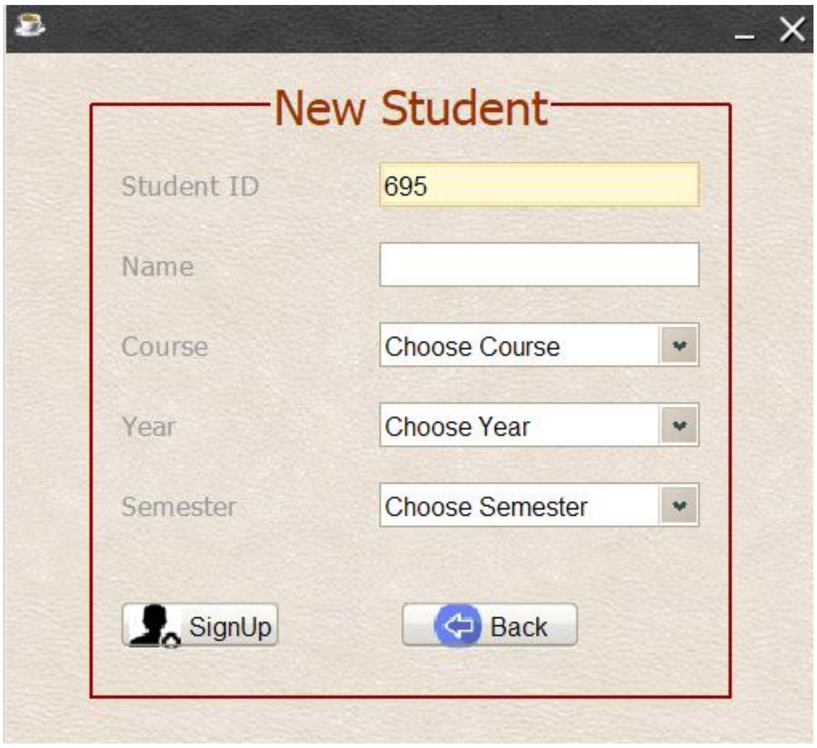
Book ID	750
Book Name	
Publisher	
Edition	Select Edition
Author	


Message

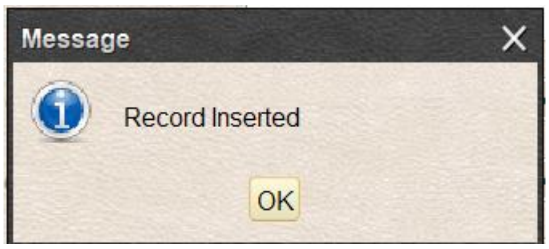
Book Added Successfully !

OK

❖ *Student.java*


New Student

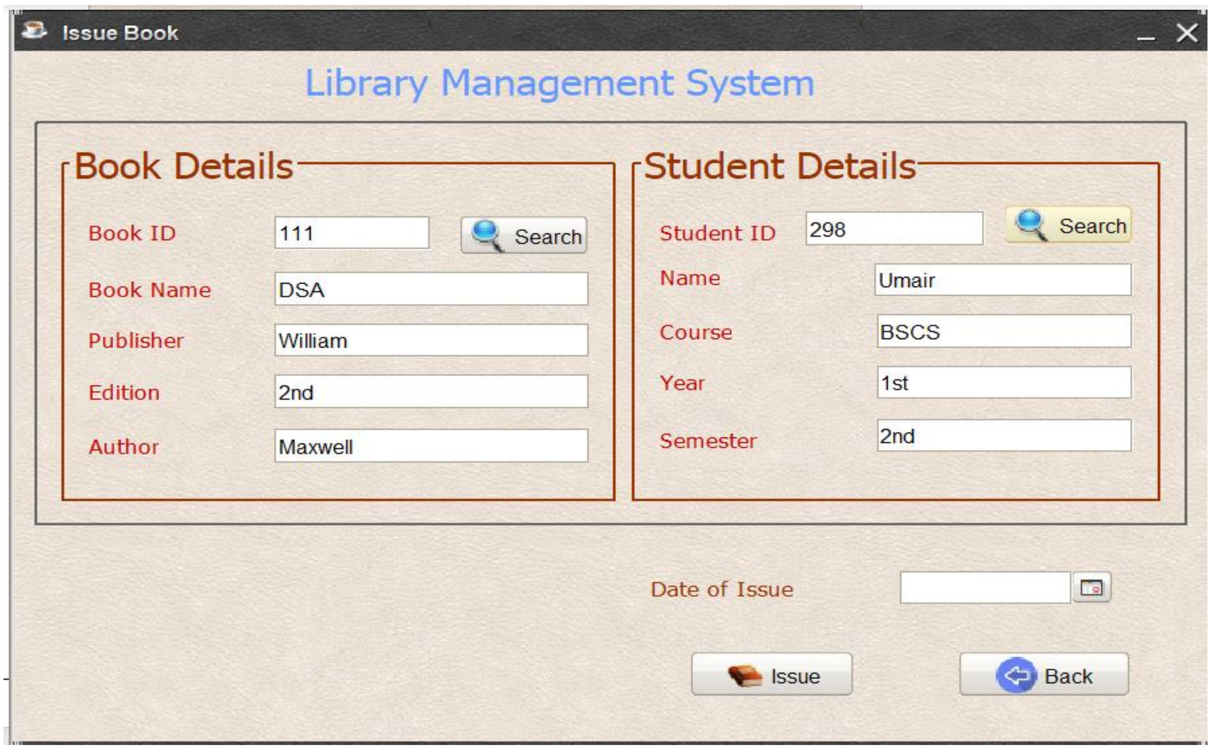
Student ID	695
Name	
Course	Choose Course
Year	Choose Year
Semester	Choose Semester


Message

Record Inserted

OK

❖ *Issue.java*



Issue Book

Library Management System

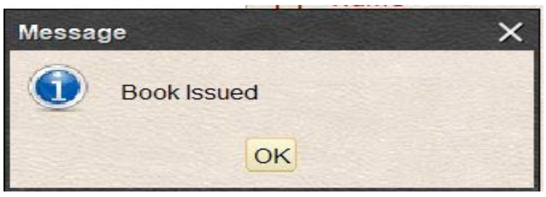
Book Details

Book ID	111
Book Name	DSA
Publisher	William
Edition	2nd
Author	Maxwell

Student Details

Student ID	298
Name	Umair
Course	BSCS
Year	1st
Semester	2nd

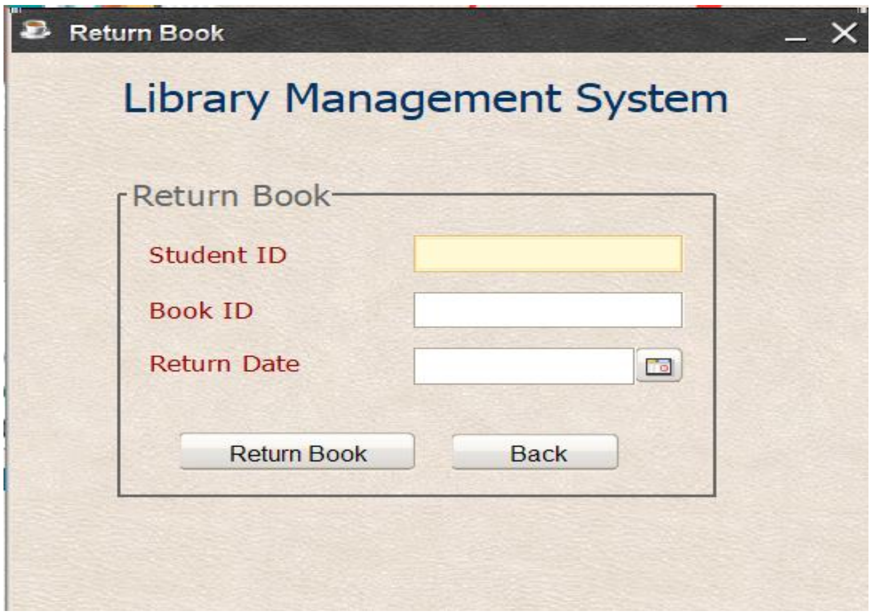
Date of Issue



Message

Book Issued

▪ *Return.java*



Return Book

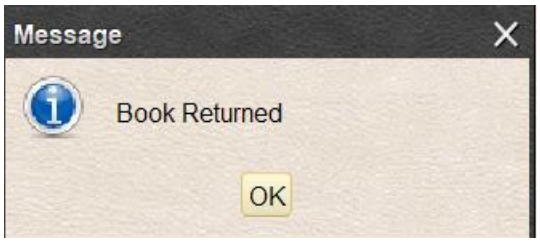
Library Management System

Return Book

Student ID

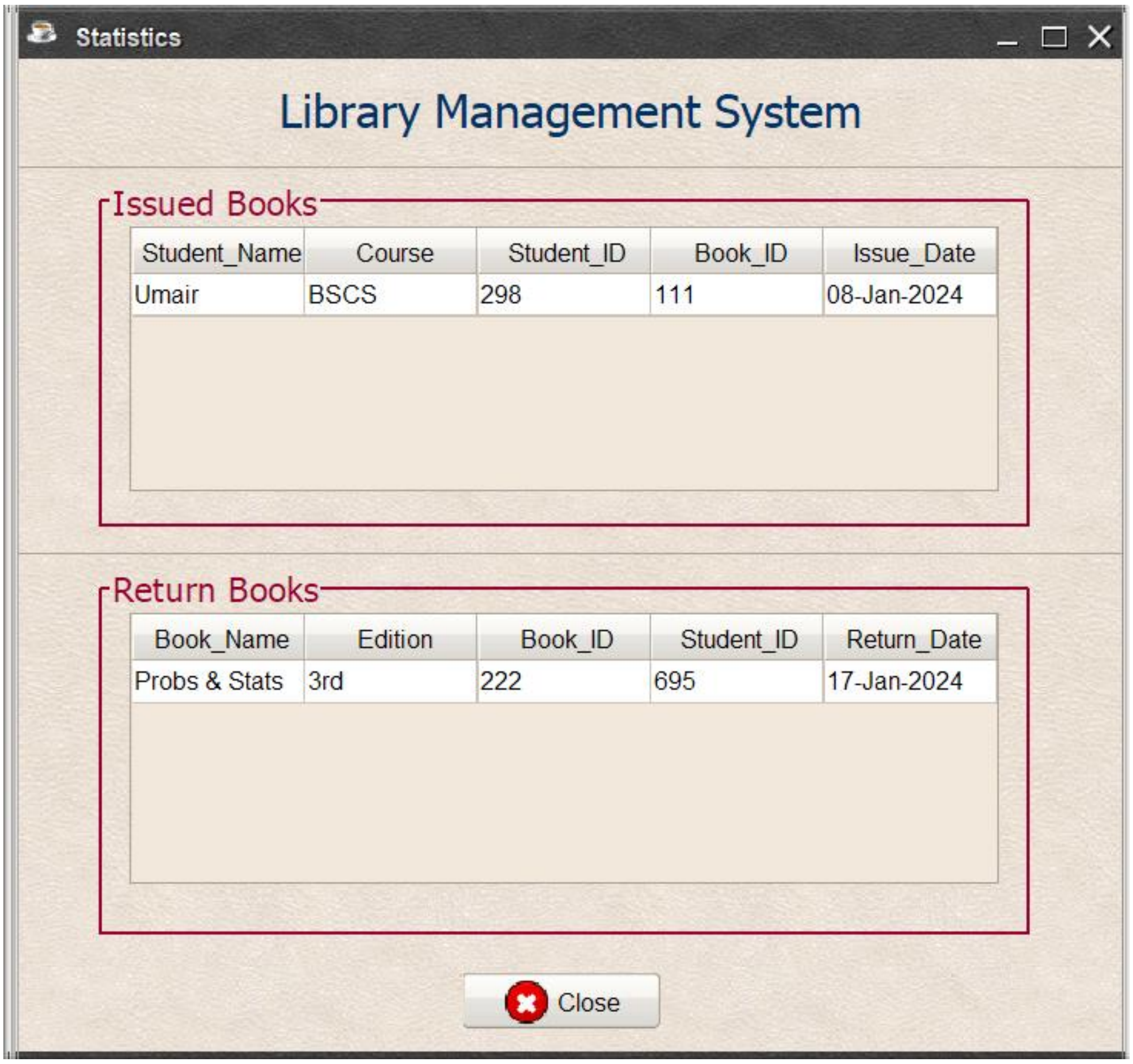
Book ID

Return Date


Message

Book Returned

❖ *Statistics.java*


Statistics

Library Management System

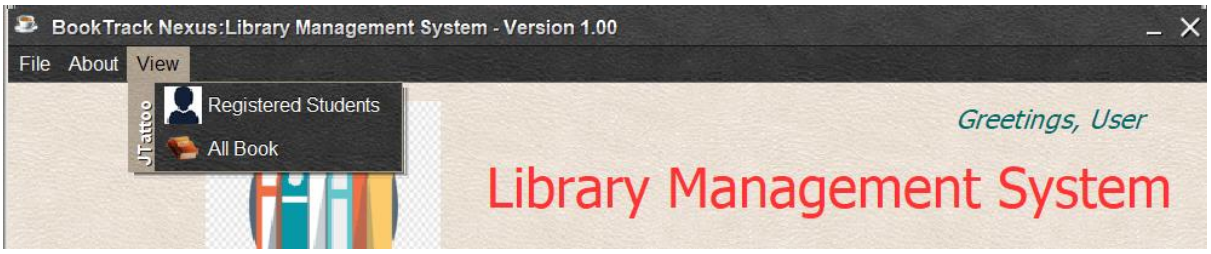
Issued Books

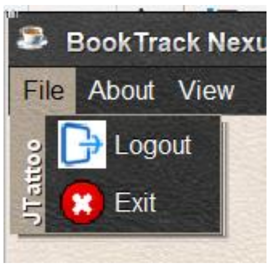
Student_Name	Course	Student_ID	Book_ID	Issue_Date
Umair	BSCS	298	111	08-Jan-2024

Return Books

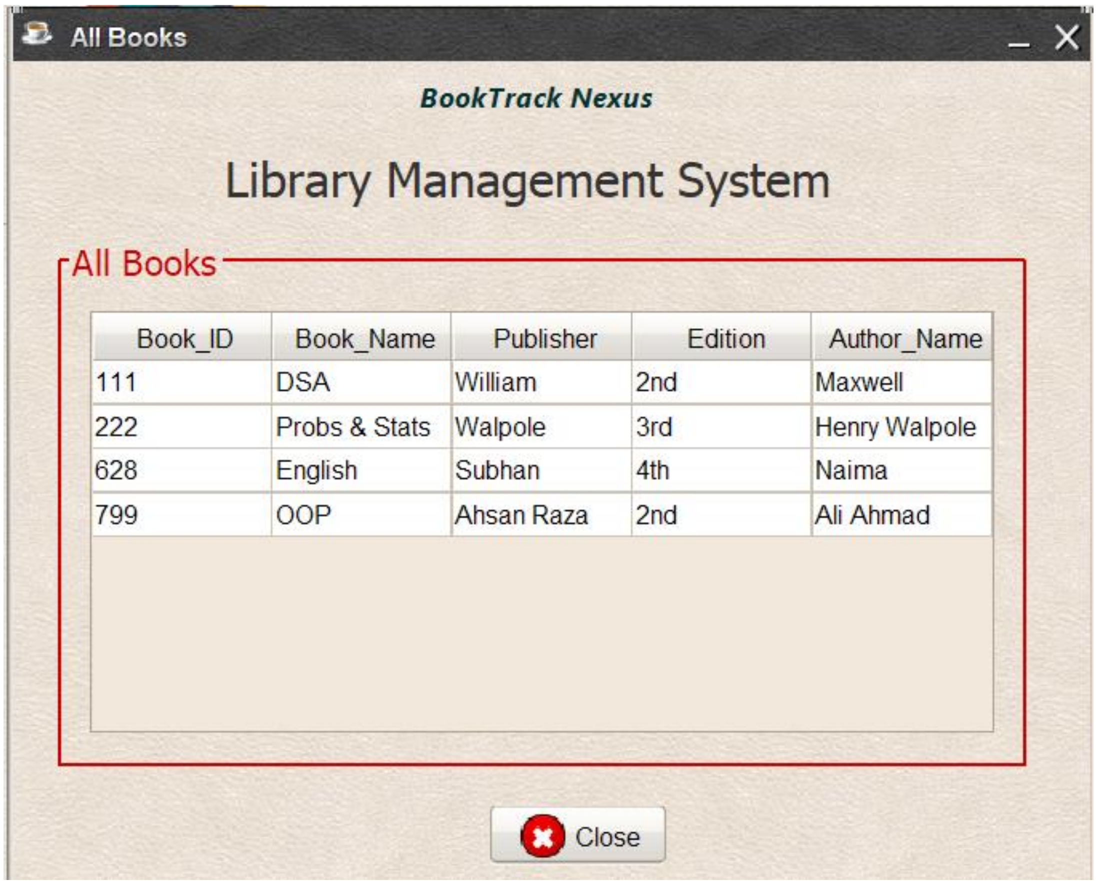
Book_Name	Edition	Book_ID	Student_ID	Return_Date
Probs & Stats	3rd	222	695	17-Jan-2024

BookTrack Nexus:Library Management System - Version 1.00


BookTrack Nexus:Library Management System - Version 1.00
File
About
View
Registered Students
All Book
JTattoo
Greetings, User
Library Management System

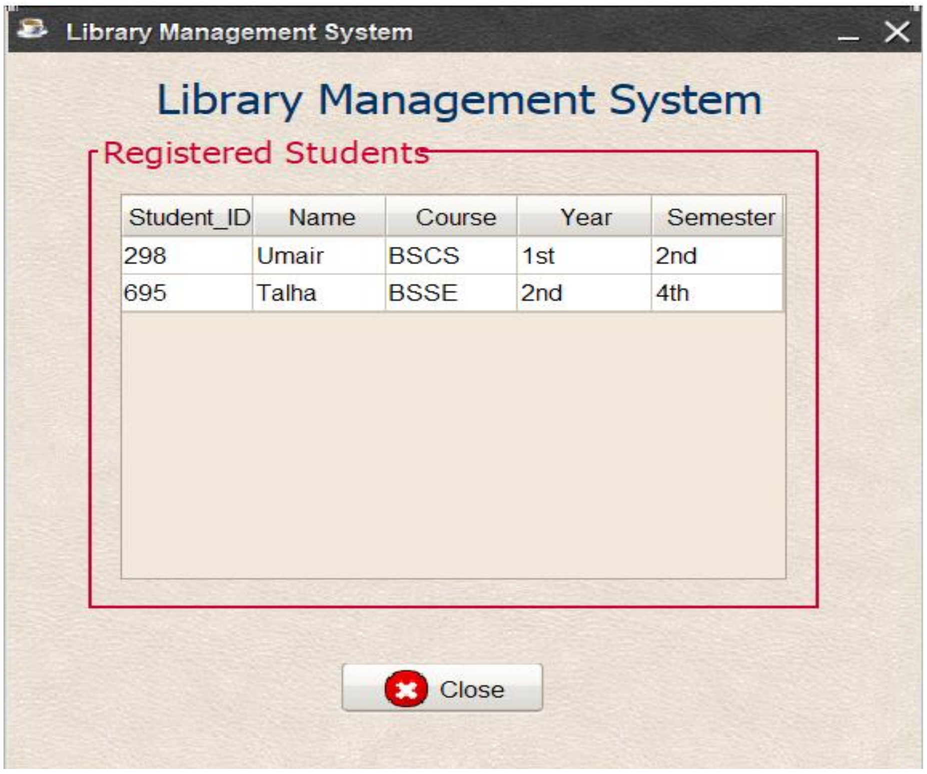


❖ *AllBooks.java*



Book_ID	Book_Name	Publisher	Edition	Author_Name
111	DSA	William	2nd	Maxwell
222	Probs & Stats	Walpole	3rd	Henry Walpole
628	English	Subhan	4th	Naima
799	OOP	Ahsan Raza	2nd	Ali Ahmad

❖ *RegisteredStudents.java*



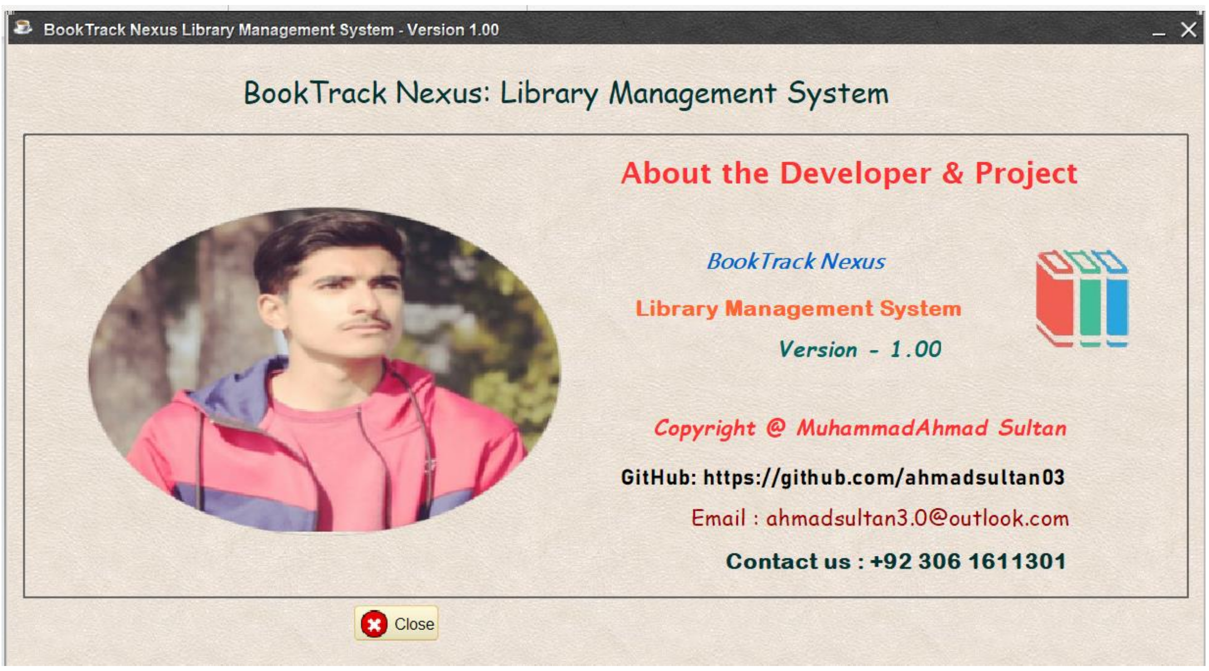
Library Management System

Library Management System

Registered Students

Student_ID	Name	Course	Year	Semester
298	Umair	BSCS	1st	2nd
695	Talha	BSSE	2nd	4th

❖ *About.java*



BookTrack Nexus Library Management System - Version 1.00

BookTrack Nexus: Library Management System

About the Developer & Project

BookTrack Nexus

Library Management System

Version - 1.00

Copyright @ MuhammadAhmad Sultan

GitHub: https://github.com/ahmadsultan03

Email : ahmadsultan3.0@outlook.com

Contact us : +92 306 1611301

Potential Questions:

- How does your Library Management System leverage data structures to optimize information retrieval and enhance overall system performance?
- In what ways does the implementation of Java Swing contribute to creating a user-friendly and intuitive interface for your library application, ensuring a seamless user experience?
- Can you elaborate on the security measures implemented in your project to safeguard user authentication and protect sensitive information within the Library Management System's database?

Conclusion:

In summary, the Library Management System project is a sophisticated and user-centric solution crafted to streamline the management of library resources, user interactions, and transactions. Developed using Java with the Swing framework and MySQL for database management, the project embodies key Object-Oriented Programming (OOP) principles, resulting in a well-structured and easily maintainable codebase. The project's execution flow unfolds seamlessly, commencing with an engaging splash screen, progressing through user authentication and loading processes, and culminating in a feature-rich home screen dashboard. This dashboard seamlessly integrates functionalities like adding students and books, viewing transaction history, searching, sorting, issuing and returning books, while also offering insights about the application and the developer. Crucially, data structures play a pivotal role in optimizing system efficiency, facilitating rapid and effective retrieval, storage, and manipulation of information regarding books, users, transactions, and student details. The incorporation of these data structures, coupled with MySQL database management and Java Swing GUI, underscores the project's commitment to delivering a robust, functional, and user-friendly Library Management System.

Reference Citations:

https://www.youtube.com/

https://www.w3schools.com/

https://www.codecademy.com/

https://www.learnjavaonline.org/

https://www.geeksforgeeks.org/learn-data-structures-and-algorithms-dsa-tutorial/

https://www.homeandlearn.co.uk/java/java_and_databases.html

THE END