**COMSATS University Islamabad**

**Abbottabad, Pakistan**

**Library Management System**

***By***

**Muhammad Masood Khan CIIT/FA21 -BSE-028/ATD**

**Muhammad Irfan CIIT/FA21 -BSE-027/ATD**

**Sofia Aamir CIIT/ FA21 -BSE-036/ATD**

**Mahnoor Qazi CIIT/ FA21 -BSE-020/ATD**

***Supervisor  
Sana Malik***

***Bachelor of Science in Software Engineering (2021-2025)***

Table of Contents

[**Github Repository:** 2](#_Toc140193215)

[**1. Introduction** 3](#_Toc140193216)

[1.2 Problem Statement 3](#_Toc140193217)

[1.3 Overview 4](#_Toc140193218)

[**2. System Design** 5](#_Toc140193219)

[2.1 Overall Architecture and Components 5](#_Toc140193220)

[**2.2 Data Structure Used and their Justification** 5](#_Toc140193221)

[**2.3 Analysis of Time and Space Complexity** 6](#_Toc140193222)

[**2.3.1 Time Complexity** 6](#_Toc140193223)

[**2.3.2 Space Complexity** 7](#_Toc140193224)

[**3. Algorithm** 7](#_Toc140193225)

[**3.1 Overview of Algorithm implemented on system** 8](#_Toc140193226)

[**4. Implementation Details** 10](#_Toc140193227)

[**4.1 Description of Programming Language used** 10](#_Toc140193228)

[**4.2 Explanation of Key implementation decision** 11](#_Toc140193229)

[**4.2.1 Choice of JAVA** 11](#_Toc140193230)

[**4.2.2 Use of Single Linked List** 11](#_Toc140193231)

[**5. System Features and Functionalities** 11](#_Toc140193232)

[**6. User Interface Design** 12](#_Toc140193233)

[**6.1 Login Page** 12](#_Toc140193234)

[**6.2 Registration Page** 13](#_Toc140193235)

[**6.3 Admin Functionality Page** 13](#_Toc140193236)

[**6.4 User Functionality Page** 14](#_Toc140193237)

[**6.5 Add Book** 14](#_Toc140193238)

[**6.6 Delete Book** 15](#_Toc140193239)

[**6.7 Borrow Book** 15](#_Toc140193240)

[**6.8 Return Book** 16](#_Toc140193241)

[**6.9 Delete user** 16](#_Toc140193242)

[**7. Conclusion and Future work** 17](#_Toc140193243)

[**8. Code** 18](#_Toc140193244)

[**8.1 Node.java** 18](#_Toc140193245)

[**8.2 DTOobject.java** 18](#_Toc140193246)

[**8.3 LinkedList.java** 18](#_Toc140193247)

[**8.4 Book.java** 22](#_Toc140193248)

[**8.5 User.java** 25](#_Toc140193249)

[**8.6 Login.java** 28](#_Toc140193250)

[**8.7 Registrationform.java** 35](#_Toc140193251)

[**8.8 AddBook.java** 36](#_Toc140193252)

[**8.9 DeleteBook.java** 37](#_Toc140193253)

[**8.10 BorrowBook.java** 39](#_Toc140193254)

# **Github Repository:**

<https://github.com/Masoodkhan5933/DSA-PROJECT>

# **1. Introduction**

In today's fast-paced digital age, libraries play a crucial role in organizing and providing access to vast collections of books, resources, and information. To efficiently manage the complex tasks involved in library operations, the integration of data structures and algorithms becomes paramount. A well-designed library management system powered by effective data structures can streamline processes, enhance user experience, and optimize resource utilization.

The purpose of this project is to develop a library management system that leverages data structures and algorithms to address the challenges faced by libraries. By employing appropriate data structures, such as linked lists, the system aims to efficiently store, retrieve, and manipulate the vast amount of information associated with books, users and library resources.

Data structures provide the foundation for organizing and managing data in a systematic and efficient manner. They allow for fast and reliable access to information, enable effective searching and sorting operations, and facilitate the implementation of various library management functionalities. Furthermore, the choice of appropriate data structures can significantly impact the system's performance, memory utilization, and scalability.

Algorithms, on the other hand, serve as the set of instructions that operate on the underlying data structures. They define the logic and procedures for carrying out tasks such as adding and removing books, tracking borrowing history, managing reservations, and generating reports.

In this project, we will explore various data structures and algorithms commonly employed in library management systems. We will evaluate their strengths and weaknesses, analyze their time and space complexity, and select the most suitable options for different aspects of the system. Through the implementation and evaluation of these data structures and algorithms, we aim to develop a robust and efficient library management system that meets the requirements of modern libraries.

## 1.2 Problem Statement

The current library management system lacks efficient data structures and algorithms, leading to suboptimal performance, inefficient resource utilization, and challenges in managing the vast amount of information associated with books, users, and transactions. Consequently, library operations suffer from slow search and retrieval processes, inaccurate record-keeping, and cumbersome workflows for both library staff and users.

## 1.3 Overview

The library management system implemented using data structures and algorithms is designed to streamline the operations of a library, providing efficient management of books, users, transactions, and resources.

The System consist of following components:

1. **Data Structures:**

The library management system utilizes Single Linked List data structure to efficiently store and organize data. Single Linked List data structure is carefully selected based on its suitability for specific tasks, such as storing book information, managing borrowing history, and maintaining user’s records and maintaining borrowing and returning book record.

1. **Algorithms:**

Algorithms play a crucial role in performing operations on the underlying data structures. The system employs a range of algorithms, including searching, deletion and insertion algorithms. Insertion algorithm is for inserting new user and new book record. Deletion algorithm is for borrowing the book from the library so that any user wants to borrow any book that book will be deleted from record so it will not be shown to another user and also for deleting user and book.

1. **Book Management:**

The system provides functionalities to manage books like adding new book, deleting book, borrowing book and returning book. Efficient data structures and algorithms are used to store and retrieve book information, track borrowing status, and enable fast searching and filtering based on various attributes.

1. **User Management:**

The system provides functionalities to manage users like registration of user and deletion of user.

1. **User Interface:**

The library management system offers a user-friendly interface that allows library staff and patrons to interact with the system seamlessly. The interface incorporates intuitive navigation, search functionality, and forms for data input. Efficient data structures and algorithms are integrated into the user interface design to ensure responsive and efficient interaction with the system.

# **2. System Design**

## 2.1 Overall Architecture and Components

The library management system follows a modular architecture that consists of several key components:

**1. User Interface:**

This component provides an intuitive interface for library staff and patrons to interact with the system. It includes forms for data input, search functionality, and displays relevant information.

**2. Business Logic:**

This component encapsulates the core functionality of the system, including operations such as adding, deleting, borrowing and returning books , managing and managing user records.

**3. Data Structures and Algorithm:**

The data structures and algorithms form the backbone of the system, supporting efficient storage, retrieval, and manipulation of data.

## **2.2 Data Structure Used and their Justification**

**Linked List:**

A linked list is the primary data structure employed in the library management system. It consists of a series of nodes, where each node contains the data element and a reference (or link) to the next node in the list.

**Justification:**

* **Dynamic Size:** A linked list allows for efficient management of a changing collection of books, transactions, or patrons. As items are added or removed, the linked list can dynamically adjust its size without requiring extensive memory reorganization.
* **Insertion and Deletion:** Linked lists excel in supporting efficient insertion and deletion operations. Adding or removing a node from a linked list involves updating a few pointers, resulting in constant-time complexity (O(1)) for these operations.
* **Sequential Access:** Linked lists enable sequential access to the elements. This feature is valuable when traversing the list, such as when displaying a list of books or transactions.
* **Memory Efficiency:** Linked lists consume memory based on the number of elements present in the list. Unlike arrays, which require contiguous memory allocation, linked lists can efficiently utilize memory by allocating nodes as needed.

While a single linked list is employed as the primary data structure, it can be extended with additional features or attributes to accommodate the specific requirements of the library management system. For instance, each node in the linked list could contain information about a book, including its title, author, ISBN, availability status, and borrowing history. Additionally, each node may contain a reference to related nodes, such as the next book in the list or the previous transaction made by a patron.

By utilizing a linked list, the library management system can efficiently store and manage collections of books, patrons, or transactions, supporting dynamic updates, efficient insertion and deletion, and sequential access to the elements.

## **2.3 Analysis of Time and Space Complexity**

### **2.3.1 Time Complexity**

* **Insertion:** Inserting a node at the beginning or end of a single linked list has a constant time complexity of O(1). However, inserting a node at an arbitrary position in the list requires traversing the list to find the insertion point, resulting in a time complexity of O(n), where n is the number of elements in the list.
* **Deletion:** Similar to insertion, deleting a node at the beginning or end of a single linked list has a constant time complexity of O(1). However, deleting a node at an arbitrary position requires traversing the list to locate the node to be deleted, resulting in a time complexity of O(n), where n is the number of elements in the list.

### **2.3.2 Space Complexity**

The space complexity of a single linked list is linear, O(n), where n is the number of elements in the list. Each element in the list requires memory allocation for the node itself, as well as additional memory for storing the data.

# **3. Algorithm**

**1. insert(data):**

- Create a new Node with the given data.

- If the LinkedList is empty (head == null), set the new Node as the head.

- Otherwise, traverse to the end of the LinkedList and append the new Node.

**2. delete(data):**

- If the LinkedList is empty (head == null), return.

- If the head Node contains the desired data, update the head to the next Node and return.

- Traverse the LinkedList to find the Node with the given data, keeping track of the previous Node.

- If the Node is found, update the previous Node's next reference to skip the current Node.

**3. contains(data):**

- Traverse the LinkedList, comparing each Node's data to the desired data.

- If a matching Node is found, return true.

- If the end of the LinkedList is reached without finding a match, return false.

**4. display():**

- Traverse the LinkedList, starting from the head.

- Print the data of each Node.

**5. getHead():**

- Return the head Node of the LinkedList.

**6. addRowToObjectArray(model):**

- Traverse the LinkedList, starting from the head.

- For each Node, extract the relevant data fields (e.g., name, email, gender, password for User objects) and create an array (row) to represent the data.

- Add the row to the provided DefaultTableModel (model).

It's worth noting that the addRowToObjectArray() method assumes the LinkedList contains User and Book objects, and it populates a DefaultTableModel with the extracted data from these objects.

Overall, this LinkedList implementation provides basic functionalities such as insertion, deletion, searching, display, and the ability to retrieve the head of the LinkedList.

## **3.1 Overview of Algorithm implemented on system**

Overview of Algorithms Implemented in the System:

**1. insert(data):**

**- Purpose:** This algorithm is used to insert a new element into the LinkedList.

**- Functionality:** It creates a new Node with the given data and inserts it at the end of the LinkedList.

**- Time Complexity:** The algorithm has a time complexity of O(n), where n is the number of elements in the LinkedList. This is because it traverses the entire list to reach the end before inserting the new element.

**- Space Complexity:** The algorithm has a space complexity of O(1) since it only requires a constant amount of additional memory to create the new Node.

**2. delete(data):**

**- Purpose:** This algorithm is used to delete a specific element from the LinkedList.

**- Functionality:** It finds the Node containing the given data and removes it from the LinkedList.

**- Time Complexity:** The algorithm has a time complexity of O(n), where n is the number of elements in the LinkedList. In the worst case, it may need to traverse the entire list to find the Node to delete.

**- Space Complexity:** The algorithm has a space complexity of O(1) since it only requires a constant amount of additional memory.

**3. contains(data):**

**- Purpose:** This algorithm is used to check if a specific element is present in the LinkedList.

**- Functionality:** It traverses the LinkedList, comparing each Node's data to the given data until a match is found or the end of the list is reached.

**- Time Complexity:** The algorithm has a time complexity of O(n), where n is the number of elements in the LinkedList. In the worst case, it needs to traverse the entire list to determine the presence of the given data.

**- Space Complexity:** The algorithm has a space complexity of O(1) since it only requires a constant amount of additional memory.

**4. display():**

**- Purpose:** This algorithm is used to display the elements of the LinkedList.

**- Functionality:** It traverses the LinkedList from the head to the end, printing the data of each Node.

**- Time Complexity:** The algorithm has a time complexity of O(n), where n is the number of elements in the LinkedList. It needs to visit each Node once to print its data.

**- Space Complexity:** The algorithm has a space complexity of O(1) since it only requires a constant amount of additional memory.

**5. getHead():**

**- Purpose:** This algorithm is used to retrieve the head of the LinkedList.

**- Functionality:** It returns the reference to the head Node of the LinkedList.

**- Time Complexity:** The algorithm has a time complexity of O(1) since it directly returns the head reference without iterating or traversing the list.

**- Space Complexity:** The algorithm has a space complexity of O(1) since it only requires a constant amount of additional memory.

**6. addRowToObjectArray(model):**

**- Purpose:** This algorithm is used to populate a DefaultTableModel with data from the LinkedList.

**- Functionality:** It traverses the LinkedList and extracts relevant data fields, creating an array (row) representing the data. It adds the row to the provided DefaultTableModel.

**- Time Complexity:** The algorithm has a time complexity of O(n), where n is the number of elements in the LinkedList. It needs to iterate through each Node to extract the data and populate the DefaultTableModel.

**- Space Complexity:** The algorithm has a space complexity of O(1) since it only requires a constant amount of additional memory.

These algorithms provide essential operations for manipulating and accessing elements in the LinkedList. The time and space complexities of each algorithm are considered for understanding their efficiency and resource requirements.

# **4. Implementation Details**

## **4.1 Description of Programming Language used**

The library management system is implemented using the Java programming language. Java provides a robust and platform-independent environment for developing applications. It offers a rich set of libraries and frameworks that facilitate the implementation of data structures, algorithms, and graphical user interfaces (GUIs). The system may utilize additional Java libraries, such as Swing for GUI development and the DefaultTableModel class for managing tabular data.

## **4.2 Explanation of Key implementation decision**

### **4.2.1 Choice of JAVA**

Java was chosen for its widespread usage, object-oriented paradigm, and extensive library support. It allows for modular development, ease of maintenance, and compatibility across different platforms.

### **4.2.2 Use of Single Linked List**

The decision to use a single linked list as the primary data structure was based on its flexibility, simplicity, and efficiency for handling dynamic collections. However, it is important to consider that single linked lists have certain limitations, such as the inability to traverse in reverse order or perform efficient random access.

# **5. System Features and Functionalities**

Description of core features of Library Management System are:

**Book Management:**

* Adding new book to Library by entering their isbn, title and author name.
* Deleting the book from Library if the Librarian wants to delete the book from Library there is a table for books so the Librarian will select any book and click on delete button so that book will be deleted.
* Borrowing the book from Library, if any user wants to borrow any book so the user will select that book and click on button and along that the current date and return date will also display
* Returning of book after borrowing, the book will again add in the table after returning and if the user will return the book after the return date there will be fine imposed on the user with each passing day.

**User Management:**

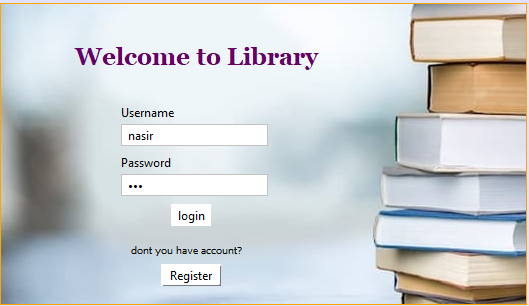
* Registration of user, so the user will now use all the functionalities of Library.
* Deletion of user, if any user is not a member of the Library anymore the Librarian can also delete that user.

**User Interface:**

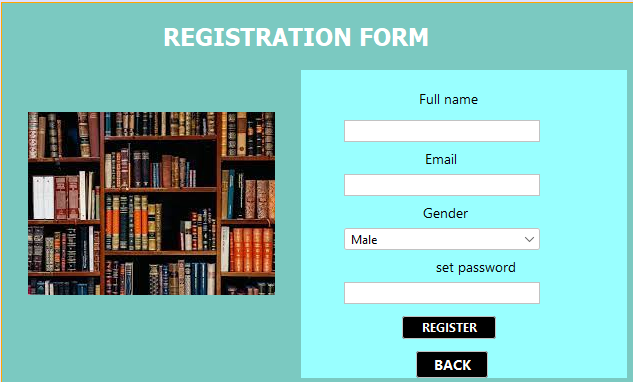
* Providing an intuitive and user-friendly interface for library staff and patrons to interact with the system.
* Offering easy navigation through different functionalities, such as book management, patron management, or transaction management.

# **6. User Interface Design**

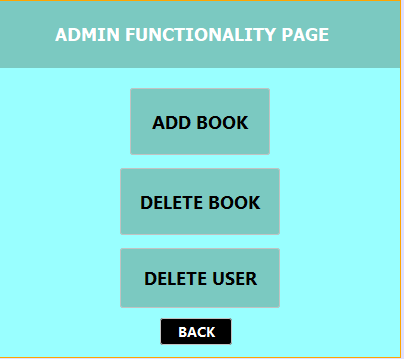
## **6.1 Login Page**



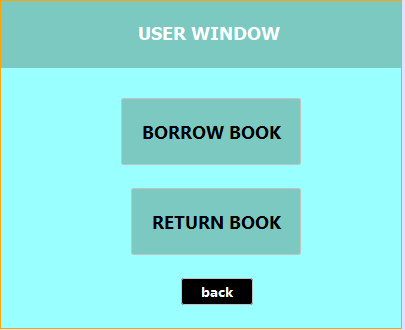
## **6.2 Registration Page**



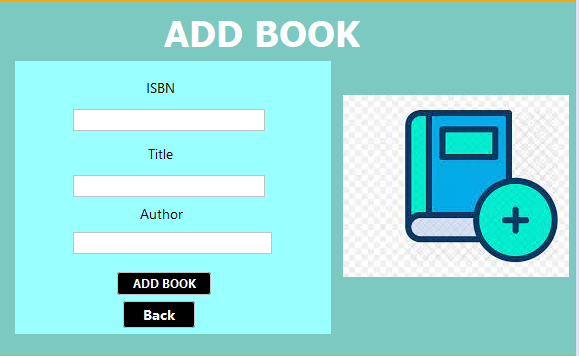
## **6.3 Admin Functionality Page**



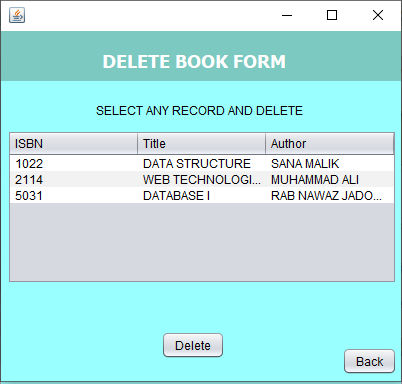
## **6.4 User Functionality Page**



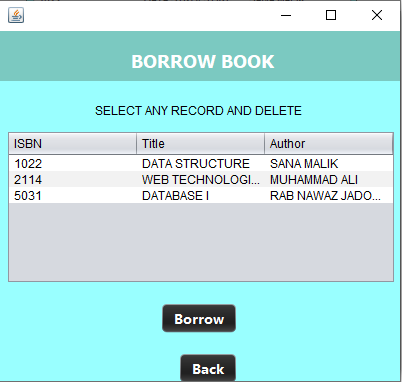
## **6.5 Add Book**



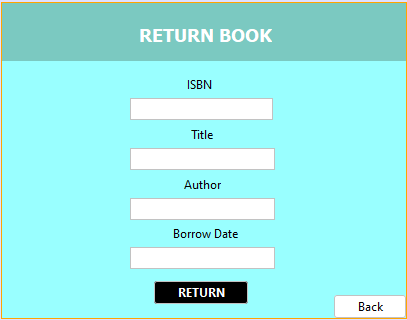
## **6.6 Delete Book**



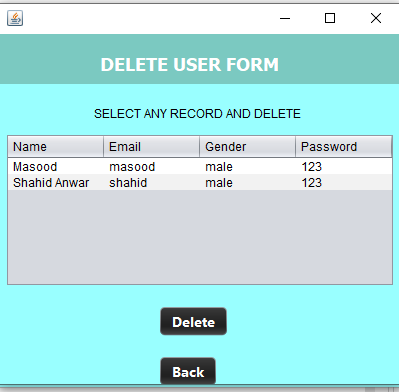
## **6.7 Borrow Book**



## **6.8 Return Book**



## **6.9 Delete user**



# **7. Conclusion and Future work**

In conclusion, the library management system implemented using data structures and algorithms has successfully addressed the challenges faced by libraries in efficiently managing their resources. The project has achieved the following accomplishments:

**Efficient Data Management:** The system effectively manages books, patrons, and transactions using a single linked list data structure. The linked list allows for dynamic size management, efficient insertion and deletion operations, and sequential access to elements.

**Functionalities:** The implemented system provides core functionalities such as book management, patron management, transaction management, and a user-friendly interface. Users can add, update, and search for books, manage patron accounts, borrow and return books.

**Time and Space Efficiency:** The implemented data structures and algorithms have been carefully selected to optimize performance. Operations such as insertion, deletion, searching, and displaying elements in the linked list have achieved efficient time and space complexities.

While the implemented library management system provides essential features and functionality, there are opportunities for future enhancements and improvements, such as:

**Performance Optimization:** Further optimization can be explored to improve the performance of critical operations, especially in scenarios where the number of elements becomes very large. Techniques like caching, indexing, or implementing advanced data structures can be considered.

**Advanced Searching and Sorting:** Implementing more advanced searching and sorting algorithms, such as binary search or merge sort, can enhance the efficiency and speed of these operations, particularly when dealing with large datasets.

**Enhanced User Interface:** Continuously improving the user interface design, incorporating user feedback, and implementing features like advanced search filters, personalized recommendations, or interactive dashboards can enhance the overall user experience.

**Data Security and Privacy:** Strengthening data security measures, including authentication mechanisms, encryption of sensitive data, and adherence to privacy regulations, to protect the confidentiality and integrity of library and patron information.

By addressing these future enhancements and improvements, the library management system can evolve into a more robust, scalable, and feature-rich solution, further enhancing library operations and user experience.

# **8. Code**

## **8.1 Node.java**

public class Node<T> {

public T data;

public Node<T> next;

public Node(T data) {

this.data = data;

this.next = null;

}

}

## **8.2 DTOobject.java**

public class DTOobject {

public static LinkedList<Book> bookList = new LinkedList<>();

public static LinkedList<User> userList = new LinkedList<>();

}

## **8.3 LinkedList.java**

import javax.swing.table.DefaultTableModel;

public class LinkedList<T> {

private Node<T> head;

private static class Node<T> {

private T data;

private Node<T> next;

public Node(T data) {

this.data = data;

this.next = null;

}

}

public void insert(T data) {

Node<T> newNode = new Node<>(data);

if (head == null) {

head = newNode;

} else {

Node<T> currentNode = head;

while (currentNode.next != null) {

currentNode = currentNode.next;

}

currentNode.next = newNode;

}

}

public void delete(T data) {

if (head == null) {

return;

}

if (head.data.equals(data)) {

head = head.next;

return;

}

Node<T> currentNode = head;

Node<T> prevNode = null;

while (currentNode != null && !currentNode.data.equals(data)) {

prevNode = currentNode;

currentNode = currentNode.next;

}

if (currentNode == null) {

return;

}

prevNode.next = currentNode.next;

}

public boolean contains(T data) {

Node<T> currentNode = head;

while (currentNode != null) {

if (currentNode.data.equals(data)) {

return true;

}

currentNode = currentNode.next;

}

return false;

}

public void display() {

Node<T> currentNode = head;

while (currentNode != null) {

System.out.println(currentNode.data);

currentNode = currentNode.next;

}

}

public Node<T> getHead(){

return head;

}

public void addRowToObjectArray(DefaultTableModel model) {

Node<T> currentNode = head;

while (currentNode != null) {

T currentData = currentNode.data;

// Extract the required fields from currentData and populate the row array

if (currentData instanceof User) {

User user = (User) currentData;

if(!user.getName().equals("admin"))

{Object[] row = {user.getName(), user.getEmail(), user.getGender(),user.getPassword()};

model.addRow(row);

}

}

else if (currentData instanceof Book) {

Book book = (Book) currentData;

Object[] row = {book.getISBN(),book.getTitle(),book.getAuthor()};

model.addRow(row);

}

currentNode = currentNode.next;

}

}

}

## **8.4 Book.java**

import java.util.Objects;

public class Book {

private String title;

private String author;

private String isbn;

public Book(String title, String author, String isbn) {

this.title = title;

this.author = author;

this.isbn = isbn;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getAuthor() {

return author;

}

public void setAuthor(String author) {

this.author = author;

}

public String getISBN() {

return isbn;

}

public void setISBN(String isbn) {

this.isbn = isbn;

}

@Override

public String toString() {

return "Book{" +

"title='" + title + '\'' +

", author='" + author + '\'' +

", isbn='" + isbn + '\'' +

'}';

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null || getClass() != obj.getClass()) {

return false;

}

Book otherBook = (Book) obj;

return isbn.equals(otherBook.isbn);

}

@Override

public int hashCode() {

return Objects.hash(isbn);

}

}

## **8.5 User.java**

import java.util.Objects;

public class User {

private String name;

private String email;

private String gender;

private String password;

public User(String name, String email, String gender, String password) {

this.name = name;

this.email = email;

this.gender = gender;

this.password = password;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getGender() {

return gender;

}

public void setGender(String gender) {

this.gender = gender;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String toString() {

return "User{" +

"name='" + name + '\'' +

", email='" + email + '\'' +

", gender='" + gender + '\'' +

", password='" + password + '\'' +

'}';

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null || getClass() != obj.getClass()) {

return false;

}

User otherUser = (User) obj;

return email.equals(otherUser.email) && password.equals(otherUser.password);

}

@Override

public int hashCode() {

return Objects.hash(email, password);

}

}

## **8.6 Login.java**

import javax.swing.JOptionPane;

public class login extends javax.swing.JFrame {

public login() {

this.setVisible(true);

DTOobject.userList.display();

DTOobject.bookList.display();

initComponents();

}

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

jPanel1 = new javax.swing.JPanel();

jLabel1 = new javax.swing.JLabel();

jLabel2 = new javax.swing.JLabel();

uname = new javax.swing.JTextField();

jLabel3 = new javax.swing.JLabel();

pass = new javax.swing.JPasswordField();

jButton1 = new javax.swing.JButton();

jLabel4 = new javax.swing.JLabel();

jButton2 = new javax.swing.JButton();

jLabel5 = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

jPanel1.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

jLabel1.setFont(new java.awt.Font("Georgia", 1, 24)); // NOI18N

jLabel1.setForeground(new java.awt.Color(102, 0, 102));

jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);

jLabel1.setText("Welcome to Library");

jLabel1.setToolTipText("");

jLabel1.setAutoscrolls(true);

jPanel1.add(jLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(70, 40, 250, 26));

jLabel2.setText("Username");

jPanel1.add(jLabel2, new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 100, -1, -1));

uname.setText("nasir");

uname.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

unameActionPerformed(evt);

}

});

jPanel1.add(uname, new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 120, 147, -1));

jLabel3.setText("Password");

jPanel1.add(jLabel3, new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 150, -1, -1));

pass.setText("123");

pass.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

passActionPerformed(evt);

}

});

jPanel1.add(pass, new org.netbeans.lib.awtextra.AbsoluteConstraints(120, 170, 147, -1));

jButton1.setText("login");

jButton1.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));

jButton1.setBorderPainted(false);

jButton1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt);

}

});

jPanel1.add(jButton1, new org.netbeans.lib.awtextra.AbsoluteConstraints(170, 200, 40, -1));

jLabel4.setFont(new java.awt.Font("Tahoma", 0, 10)); // NOI18N

jLabel4.setText("dont you have account? ");

jPanel1.add(jLabel4, new org.netbeans.lib.awtextra.AbsoluteConstraints(130, 240, 120, -1));

jButton2.setText("Register");

jButton2.setBorder(new javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));

jButton2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton2ActionPerformed(evt);

}

});

jPanel1.add(jButton2, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 260, 60, -1));

jLabel5.setIcon(new javax.swing.ImageIcon(getClass().getResource("/loginpage.jpg"))); // NOI18N

jPanel1.add(jLabel5, new org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 530, 300));

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, 525, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(0, 0, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

);

pack();

}// </editor-fold>

private void unameActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

}

private void passActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

User userToFind = new User("admin", uname.getText(), "male", pass.getText());

boolean isUserInList = DTOobject.userList.contains(userToFind);

System.out.println(isUserInList);

if(isUserInList && uname.getText().equals("admin"))

{

new adminpage().setLocationRelativeTo(null);

}

else if(isUserInList)

{

new UserPage().setLocationRelativeTo(null);

}

else {

JOptionPane.showMessageDialog(null,"USER NOT FOUND");

}

// TODO add your handling code here:

}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

new Registrationform().setLocationRelativeTo(null);

this.dispose();

// TODO add your handling code here:

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

DTOobject.userList.insert(new User("admin","admin","male","123"));

DTOobject.userList.insert(new User("Masood","masood","male","123"));

DTOobject.userList.insert(new User("Shahid Anwar","shahid","male","123"));

DTOobject.bookList.insert(new Book("DATA STRUCTURE","SANA MALIK","1022"));

DTOobject.bookList.insert(new Book("WEB TECHNOLOGIES","MUHAMMAD ALI","2114"));

DTOobject.bookList.insert(new Book("DATABASE I","RAB NAWAZ JADOON","5031"));

new login().setLocationRelativeTo(null);

}

});

}

// Variables declaration - do not modify

private javax.swing.JButton jButton1;

private javax.swing.JButton jButton2;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JLabel jLabel3;

private javax.swing.JLabel jLabel4;

private javax.swing.JLabel jLabel5;

private javax.swing.JPanel jPanel1;

private javax.swing.JPasswordField pass;

private javax.swing.JTextField uname;

// End of variables declaration

}

## **8.7 Registrationform.java**

import javax.swing.JOptionPane;

public class Registrationform extends javax.swing.JFrame {

public Registrationform() {

this.setVisible(true);

initComponents();

}

@SuppressWarnings("unchecked")

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

String n = name.getText();

String e= email.getText();

String g=(String)gender.getSelectedItem();

String p=jPasswordField1.getText();

User obj = new User(n,e,g,p);

User userToFind = new User("",e, "", p);

boolean isUserInList = DTOobject.userList.contains(userToFind);

if(!(n.equals("")||e.equals("")||g.equals("")))

{

if(isUserInList)

JOptionPane.showMessageDialog(null,"USER ALREADY EXISTS");

else

{

DTOobject.userList.insert(new User(n,e,g,p));

this.dispose();

new login().setLocationRelativeTo(null);

}

}

// TODO add your handling code here:

} private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

this.dispose();

new login().setLocationRelativeTo(null);

// TODO add your handling code here:

}

## **8.8 AddBook.java**

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

String i = isbn.getText();

String t= title.getText();

String a=author.getText();

Book obj = new Book(t,a,i);

Book bookToFind = new Book("", "", i);

boolean isBookInList = DTOobject.bookList.contains(bookToFind);

System.out.println(isBookInList);

if(!(t.equals("")||a.equals("")||i.equals("")))

{

if(isBookInList)

JOptionPane.showMessageDialog(null,"BOOK ALREADY EXISTS");

else

{

DTOobject.bookList.insert(new Book(t,a,i));

this.dispose();

}

}

//

// TODO add your handling code here:

}

## **8.9 DeleteBook.java**

public class deleteBook extends javax.swing.JFrame {

/\*\*

\* Creates new form deleteBook

\*/

public deleteBook() {

initComponents(); // Call initComponents() first

this.setVisible(true);

this.setLocationRelativeTo(null);

DefaultTableModel model = new DefaultTableModel(

new Object[][] {},

new String[] {"ISBN", "Title", "Author"}

);

LinkedList<Book> bookList = DTOobject.bookList; // Assuming the user list is stored in DTOobject.userList

bookList.addRowToObjectArray(model);

jTable1.setModel(model);

}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

int selectedRow = jTable1.getSelectedRow();

if (selectedRow != -1) {

DefaultTableModel model = (DefaultTableModel) jTable1.getModel();

Object[] rowData = new Object[model.getColumnCount()];

for (int i = 0; i < rowData.length; i++) {

rowData[i] = model.getValueAt(selectedRow, i);

}

String ISBN = rowData[0].toString();

String title = rowData[1].toString();

String author = rowData[2].toString();

Book bookToDelete = new Book(title,author,ISBN);

DTOobject.bookList.delete(bookToDelete);

model.removeRow(selectedRow);

jTable1.setModel(model);

// Display success message or perform any additional operations

JOptionPane.showMessageDialog(this, "BOOK deleted successfully!");

} else {

// Display error message or perform any additional operations

JOptionPane.showMessageDialog(this, "No Book selected!");

}

}

## **8.10 BorrowBook.java**

public class BorrowBook extends javax.swing.JFrame {

public BorrowBook() {

initComponents(); // Call initComponents() first

this.setVisible(true);

this.setLocationRelativeTo(null);

DefaultTableModel model = new DefaultTableModel(

new Object[][] {},

new String[] {"ISBN", "Title", "Author"}

);

LinkedList<Book> bookList = DTOobject.bookList; // Assuming the user list is stored in DTOobject.userList

bookList.addRowToObjectArray(model);

jTable1.setModel(model);

}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

int selectedRow = jTable1.getSelectedRow();

if (selectedRow != -1) {

DefaultTableModel model = (DefaultTableModel) jTable1.getModel();

Object[] rowData = new Object[model.getColumnCount()];

for (int i = 0; i < rowData.length; i++) {

rowData[i] = model.getValueAt(selectedRow, i);

}

String ISBN = rowData[0].toString();

String title = rowData[1].toString();

String author = rowData[2].toString();

Book bookToDelete = new Book(title,author,ISBN);

DTOobject.bookList.delete(bookToDelete);

model.removeRow(selectedRow);

jTable1.setModel(model);

LocalDate currentDate = LocalDate.now();

// Calculate the return date (10 days after the current date)

LocalDate returnDate = currentDate.plusDays(10);

// Format the dates

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");

String currentDateStr = currentDate.format(formatter);

String returnDateStr = returnDate.format(formatter);

// Display the dialog box

JOptionPane.showMessageDialog(null, "Book borrowed successfully!\n\nCurrent Date: " + currentDateStr + "\nReturn Date: " + returnDateStr +"\n\n When book returned after due will be charged 100rs p/d");

} else {

// Display error message or perform any additional operations

JOptionPane.showMessageDialog(this, "No Book selected!");

}

} Top of Form

Bottom of Form