

Intro to Programming

Group 2

System Development NQF5

Intro to Programming

[Untitled Project - Slow Living 1.png](https://drive.google.com/open?id=1Y_eG0aoNR21k5_2uRhs2eahvyiTOy69N&usp=drive_copy)

Library Management System

**GROUP 2**

**SYSTEM DEVELOPMENT NQF5**

**SYSTEM SUPPORT NQF5**

**ADVANCED PROGRAMMING**

**CERTIFICATION OF ORIGINALITY**

I hereby declare that the project entitled “Library Management System” submitted as part of the coursework is the result of my group’s original work. This project has been completed in partial fulfilment of the requirements of Information Technology at Gauteng City College. All the work presented in this documentation, including code, research, design, and implementation, was carried out by the members of our group, unless otherwise acknowledged.

We affirm that this submission has not been previously submitted in part or in full for the award of any kind and does not contain any material copied or plagiarized from any source without proper citation. Every effort has been made to ensure the accuracy and integrity of the information provided, and we accept full responsibility for the authenticity of the project.

We understand that any violation of this declaration may lead to disciplinary action in accordance with the academic policies of the institution.

Submitted By:

**Name/ID number/Signature:**

Anthony Bompetshi, AB0892087. A.BOMPETSHI

Takudzwa Arimenda, GN517512. T.ARIMENDA

Tshifhiwa Nkhumeleni, 0307185256087. T. NKHUMELENI

Ndivhuwo Matahela, 0207165555088. N.MATAHELA

Lesego Moatlhodi, 0412185090088. MOTHEO.

Nonjabulo Ndlovu, 0310070572089. N.NDLOV

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Project  Task weeks | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| brainstorming |  |  |  |  |  |  |  |  |
| Analysis & Objectives |  |  |  |  |  |  |  |  |
| Aims |  |  |  |  |  |  |  |  |
| Design & Development |  |  |  |  |  |  |  |  |
| Implementation |  |  |  |  |  |  |  |  |
| Testing and prototyping |  |  |  |  |  |  |  |  |
| Finalization and documentation |  |  |  |  |  |  |  |  |
| Submission & presentation |  |  |  |  |  |  |  |  |

**Library**

This program simulates a basic **library login and registration system** for users and staff (admin). It uses file handling (FileReader, BufferedReader, FileWriter, BufferedWriter) to store and retrieve credentials from .txt files, offering functionalities such as user login, staff login, registration for both, and file data display. The program is entirely **console-based** and maintains a looped interface through recursive or repeated calls to Entry().

**Static Variables and File Initialization**

static final String FILENAME = "Register.txt";

static final String FILENAME2 = "Saff.txt";

The program uses two static file paths: Register.txt for storing **user data** and Saff.txt (likely a typo of "Staff.txt") for **staff/admin data**. These files are accessed throughout the program using buffered readers and writers.

Additionally, it defines variables like Name, User, Surname, and Password as shared state across methods. This simplifies input storage but isn't ideal for multiple concurrent users—refactoring to pass data through method parameters would improve clarity and structure.

**Entry() – Main Menu Handler**

The Entry() method displays the main menu and uses a switch block to direct the user to different options such as login, registration, or exiting the program. Input is captured using a Scanner. If the user chooses an invalid number, a default case prompts them again. However, the absence of a loop causes the program to exit on invalid input without returning to the menu. A while(true) loop would be a more user-friendly way to keep the menu active until exit is explicitly selected.

**UserLogin() – User Credential Authentication**

This method prompts the user to input their username and password. It then reads through each line of Saff.txt to check if the credentials match. If a match is found, it prints a success message and exits the method. If not, it recursively calls UserLogin() again. This recursion can cause a stack overflow with repeated invalid attempts. A better approach would be to use a while(!found) loop for repeated prompts.

**RegisterUserFile() – User Registration**

This method allows a user to register by entering their name, surname, username, and password. It creates a string with the user's data formatted in a readable way and writes it to Register.txt using BufferedWriter in append mode (true). This is good because it avoids overwriting existing data. After writing, the program returns to the main menu via Entry().

Improvement: Always calling Entry() after data entry is functional, but consider letting the main loop (in main) handle flow, which promotes separation of concerns and reduces tight coupling.

**StaffLogin() – Admin Authentication**

Similar to UserLogin(), this method verifies credentials for staff members by scanning Saff.txt. Upon success, it invokes the Admin() method. If the login fails, it recursively restarts. Again, recursion should be replaced with looping to prevent call stack growth on failure.

**Admin() – Admin Menu**

Once logged in, staff are presented with three options:

1. Register a new staff member
2. Read file content (user or staff)
3. Exit

This menu functions similarly to the main Entry() menu but is specific to admin tasks. Choosing option 1 invokes AdminFile() to add a new staff member; option 2 calls readFromFile() to view saved data; and option 3 exits.

A logical enhancement here would be to wrap this menu in a loop so the admin can perform multiple actions without logging in again.

**AdminFile() – Staff Registration**

The logic is almost identical to RegisterUserFile() but writes to the staff file Saff.txt. The format and process are the same. An improvement here would be to unify this logic into a single reusable method, accepting the file path as a parameter to reduce code duplication.

**readFromFile() – Display Saved User/Staff Data**

This method allows admin users to select which file to view (1 for user, 2 for staff). It reads each line and prints it to the console. The Entry() call at the end causes the program to return to the main menu, which may not be appropriate in this context since only admin should have this access. A better design would be to return to the Admin() menu instead.

Also, the Scanner here is used for menu selection. Wrapping the read logic in a loop would allow reading multiple files without going back to the main menu.

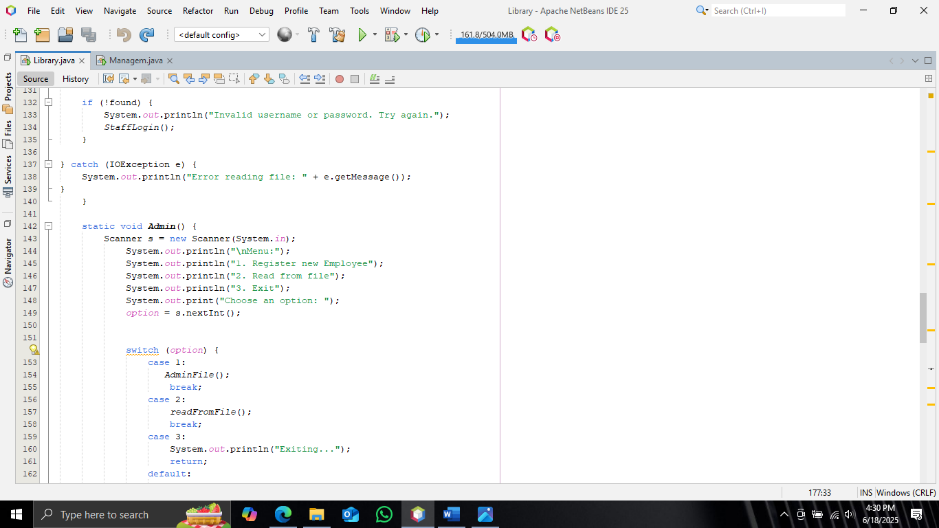
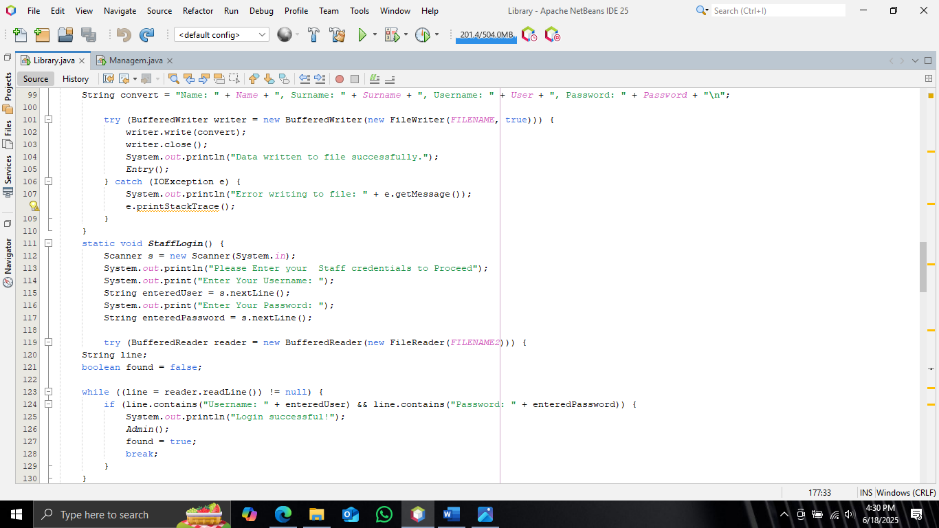
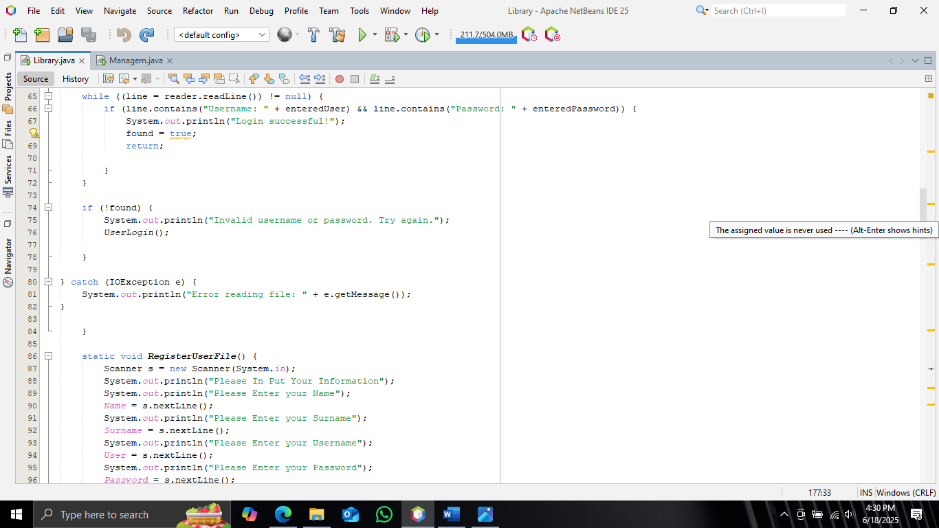
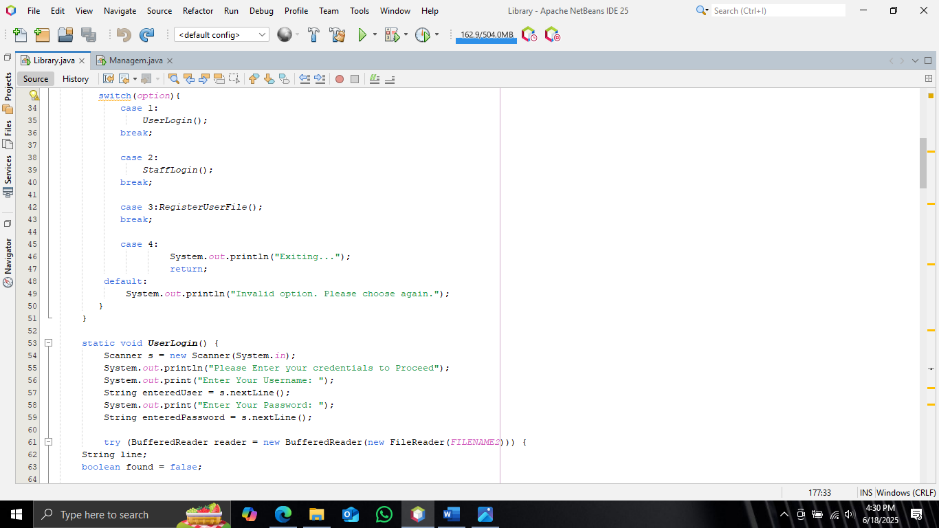
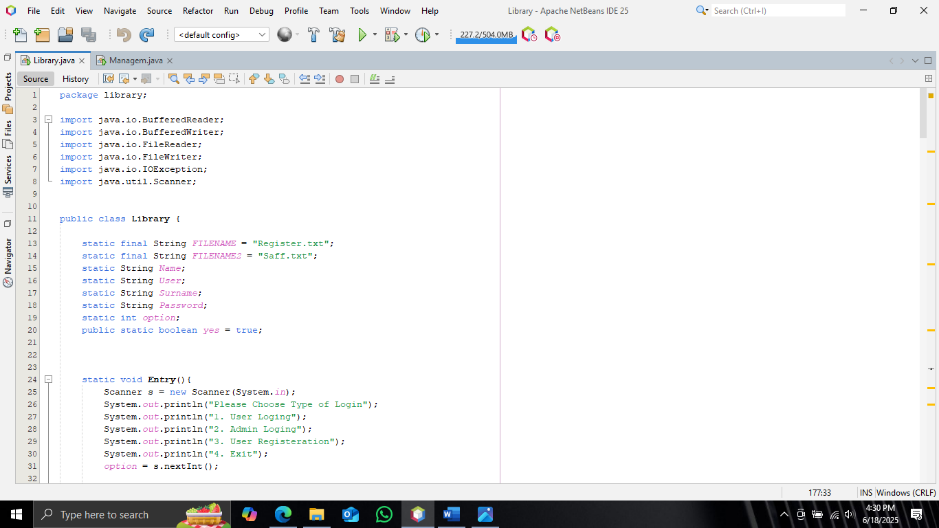
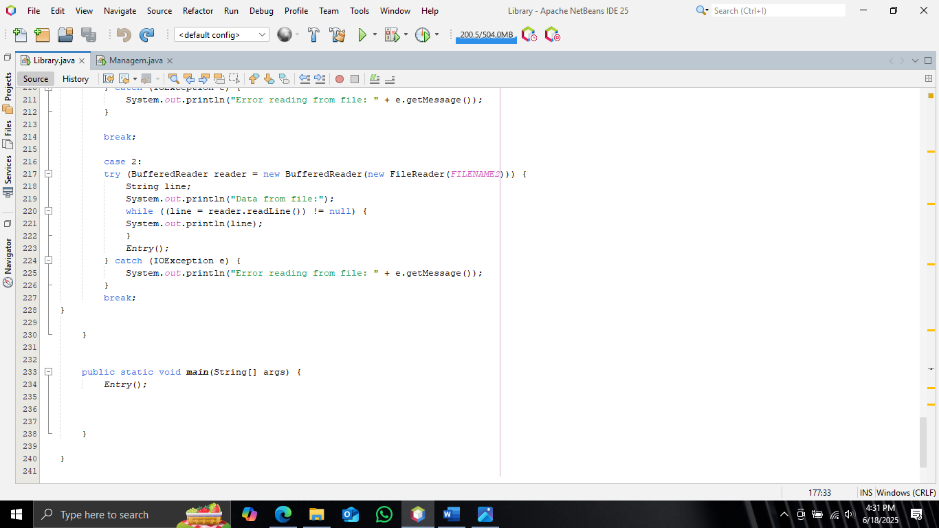
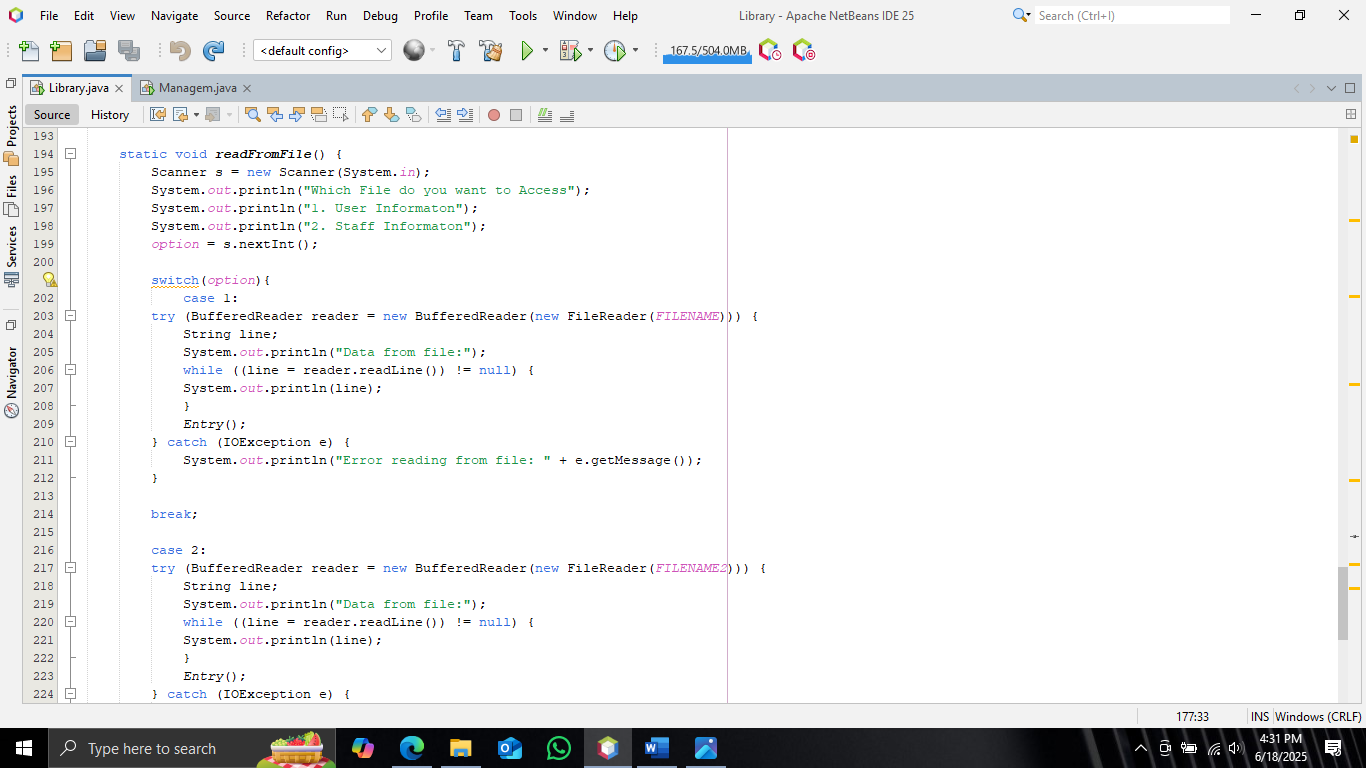
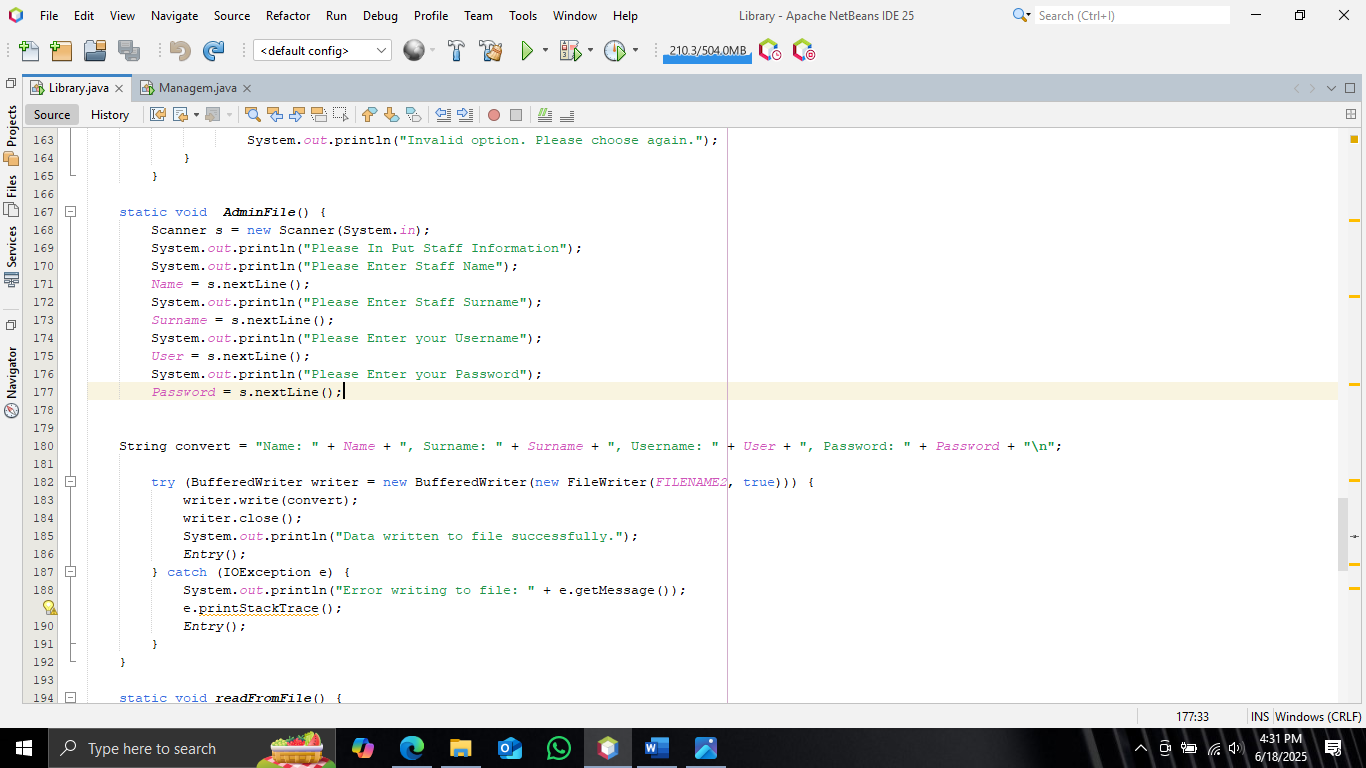
**main() – Program Entry Point**

public static void main(String[] args) {

Entry();

}

The main method simply calls the Entry() function, which starts the user interface. This keeps the program’s entry simple, which is good. But as mentioned earlier, Entry() itself should probably be structured as a loop rather than recursive calls between menu methods to better control the program flow and prevent stack overflow in longer sessions.



User management

The program serves as a basic text-based management system for searching and deleting user records from a file named Register.txt. It is divided into three main operations: displaying a menu, searching for a user, and deleting a user's entry. The main() method initiates the program by calling the menu() method, which presents a user-friendly interface in the console. This menu operates inside an infinite loop and offers three choices: search, search and delete, or exit. The loop ensures that the program continues to run until the user explicitly chooses to exit. The user’s input is captured using the Scanner class, and input validation is performed to ensure only numerical input is accepted, avoiding runtime exceptions caused by invalid entries like letters or symbols. If the input is not a number, the program informs the user and prompts again, demonstrating a basic but effective form of error handling.

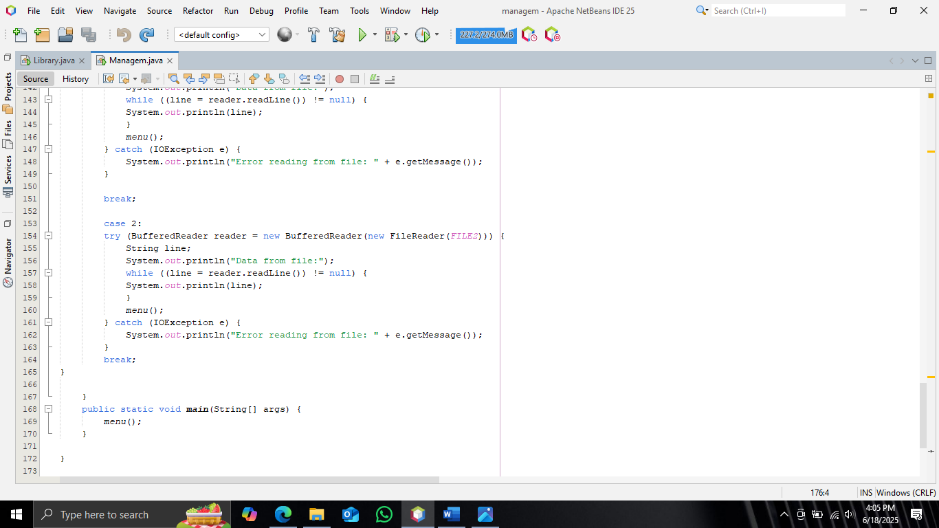
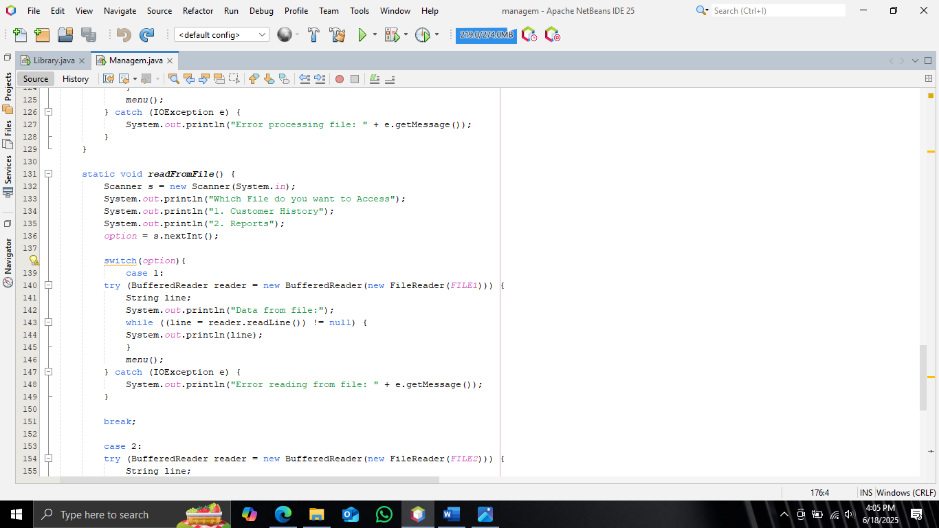
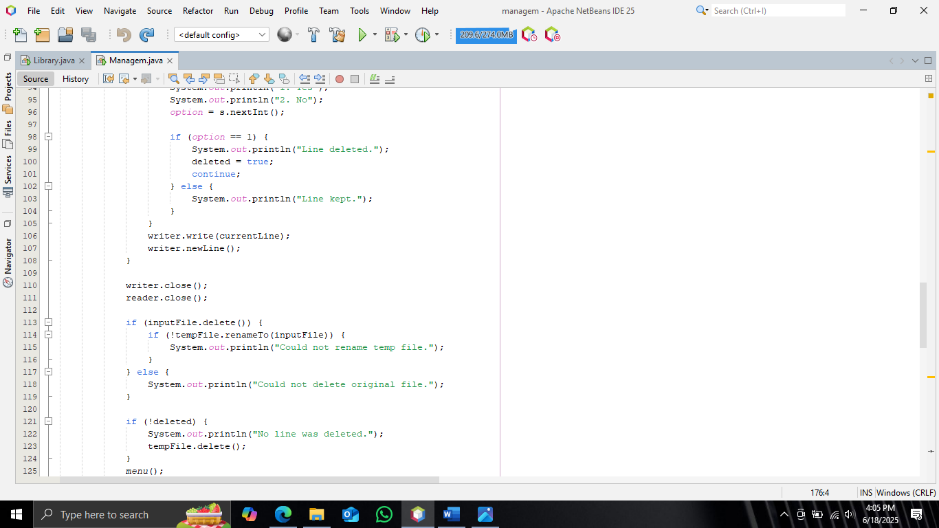
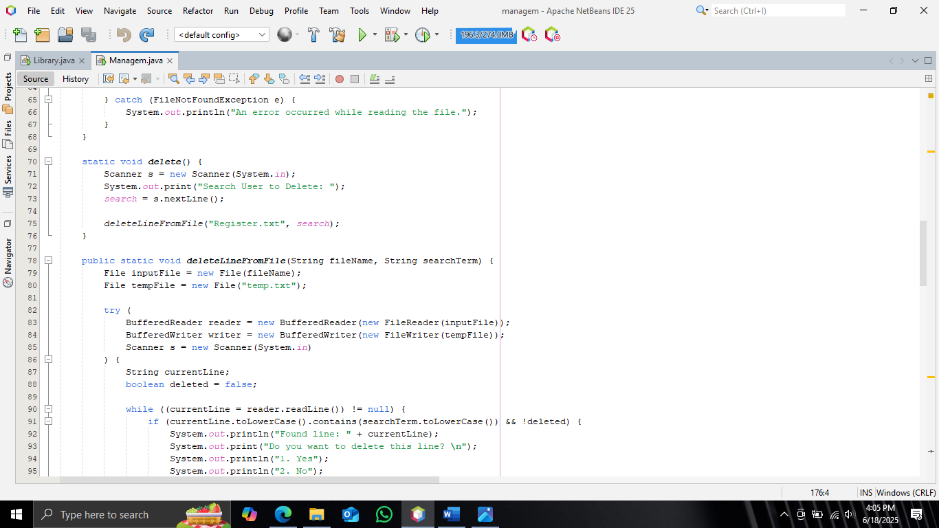
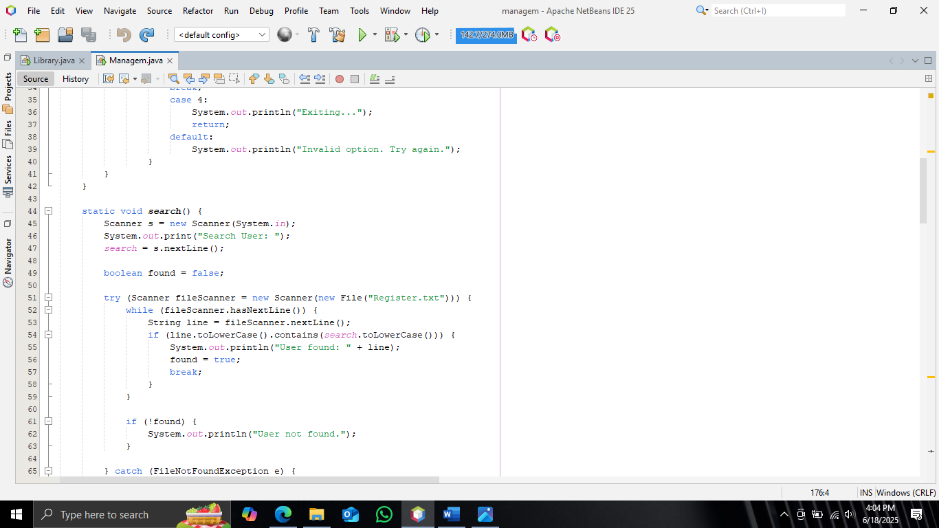
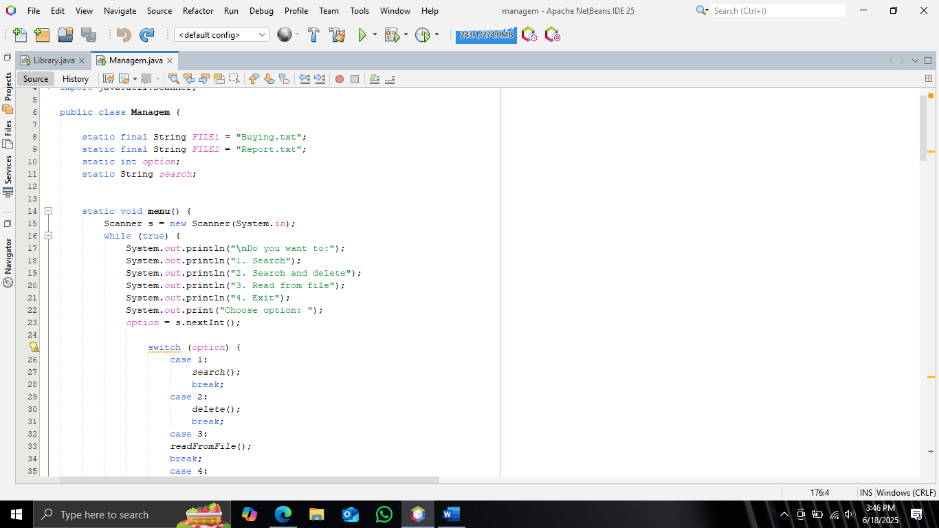
When the user selects option 1, the search() method is called. This method prompts the user to enter a search term, which is expected to be a part of a line representing a user record in the Register.txt file. The program reads the file line-by-line using another Scanner and checks each line to see if it contains the search term, using a case-insensitive comparison (toLowerCase()). If a match is found, it displays the line and informs the user that the user has been found. If no match is found after scanning the entire file, it prints a message indicating that the user was not found. This is a simple implementation of a linear search algorithm over file lines, effective for small to moderately sized text files.

If the user chooses option 2, the delete() method is invoked. This method behaves similarly to search(), in that it first prompts the user for a search term. However, instead of just displaying the matching line, it passes the search term to another method called deleteLineFromFile(). This method handles the deletion by opening both the original file (Register.txt) and a temporary file (temp.txt). It reads the original file line-by-line using a BufferedReader and writes to the temporary file using a BufferedWriter. When it encounters a line that contains the search term (again, in a case-insensitive way), it prints the line and asks the user to confirm whether they want to delete it. If the user selects “Yes,” that line is skipped (not written to the temp file), effectively deleting it. If the user selects “No,” the line is written as-is to the temp file. This approach allows the program to safely modify file contents by copying only the desired lines to a new file and then replacing the original file with this new version. This ensures that in the case of any errors or user cancellation, the original file remains untouched until everything is completed successfully.

After processing all lines, the program attempts to delete the original file and rename the temporary file to Register.txt. This two-step approach to file modification is a well-established best practice because it reduces the risk of data loss due to file corruption, failed writes, or crashes during the operation. If no deletion occurs (i.e., if the user declines to delete or the search term is not found), the temporary file is deleted as a cleanup step. This logical flow not only protects data integrity but also maintains a clean working directory.

Throughout the program, thoughtful use of try-catch blocks ensures that file I/O operations are managed gracefully. Any FileNotFoundException or IOException is caught, and an appropriate message is shown to the user, which prevents the program from crashing unexpectedly. Furthermore, the use of static variables for option and search simplifies access across multiple methods, although in a larger, object-oriented program, using instance variables and encapsulation would be recommended for better maintainability and reusability.

Overall, this code demonstrates solid foundational practices in file handling, user interaction, and defensive programming. It is well-structured, easy to follow, and practical for small-scale text-based data management. Enhancements such as looping through multiple deletions, exporting logs, or integrating with a structured format like CSV could further improve its robustness and functionality. Nonetheless, as it stands, the program effectively accomplishes its goal in a clean and user-responsive manner.



**1. About Us**

The **Library User Communication System** is a console-based Java application designed to improve interaction between **library users** and **administrators**. The purpose of this system is to keep users informed about their book statuses and library policies, while also giving them a way to raise concerns or submit enquiries.

The project is built with simplicity and functionality in mind — perfect for libraries in schools, universities, or communities that want a lightweight, easy-to-use communication platform.

**2. System Overview**

The system bridges the communication gap between users and library management by offering the following core features:

* **Notification Delivery:** Admin notifications such as overdue books and policy updates
* **Enquiry Submission:** Users can write and send their messages or concerns
* **Return-to-Menu Function:** Easy navigation throughout the program

**3. Key Features**

| **Feature** | **Description** |
| --- | --- |
| **1. Admin Notifications** | Users receive updates like overdue reminders or policy changes. |
| **2. Policy Updates** | Admins can include notices about holidays, new rules, etc. |
| **3. Enquiry Section** | Users can write and save enquiries for library staff. |
| **4. Return to Menu** | Users can go back to the main menu after every action. |
| **5. File Integration** | Notifications and enquiries are handled through .txt files. |

**4. How It Works**

1. **Admin Messages** are preloaded into a file (notifications.txt).
2. When the program runs:
   * Users can read these notifications.
   * They can also submit their own messages (enquiries), which are saved in enquiries.txt.
3. After each action, users are returned to the menu to make another selection or exit.

**6. Technologies Used**

* **Java** (console application)
* **File I/O:** BufferedReader, BufferedWriter, FileReader, FileWriter
* **User Input:** Scanner class

**7. Challenges Faced**

| **Challenge** | **Description** |
| --- | --- |
| **File Handling Errors** | If notifications.txt doesn't exist, the system must handle the error gracefully. |
| **Input Validation** | Ensuring users only enter numbers for menu selection and valid text for enquiries. |
| **Return-to-Menu Logic** | Building a reliable loop structure that keeps the user in control. |
| **No Admin Interface** | Admins must edit notifications manually in the .txt file; no interface yet. |
| **No Data Persistence** | Without a database, files can be accidentally deleted or overwritten. |

**8. Conclusion**

This Java-based Library Communication System provides a simple and effective way to manage user notifications and allow user feedback. It helps reduce the workload on librarians by automating basic communication.

With future enhancements like a GUI, admin dashboard, and database integration, this system can scale into a full-featured communication module for any library.

**"About Us"**

**2. Contact Information**

This subsection displays official communication details of the library, enabling users to reach out for support or assistance.

* **Email:** contact@READREALM.org
* **Phone:** 065 588 8114

**3. Address**

This provides the physical location of the library for users who wish to visit or locate their nearest branch.

JOHANNESBURG, SANDTON, BK 12345  
South Africa

**4. Rules and Guidelines**

* Keep noise levels low.
* Return books on time.
* No food or drinks in the reading areas.

**5. Social Media Links**

Users are encouraged to stay c

Connected with the library through social media for updates, events, and announcements:

* **Facebook:** **https://facebook.com/READREALM**
* **Twitter:** **https://twitter.com/READREALM**
* **Instagram:** **https://instagram.com/READREALM**

**6. Library Profile (Opening Hours)**

Details about when users can visit the library and which branches are operational:

**Opening Hours:**

* Monday – Friday: 9:00 AM – 6:00 PM
* Saturday: 10:00 AM – 4:00 PM
* Sunday: Closed

**Branches:**

* SOWETO
* SOWETO
* CAPETOWN

**7. Return to Menu Functionality**

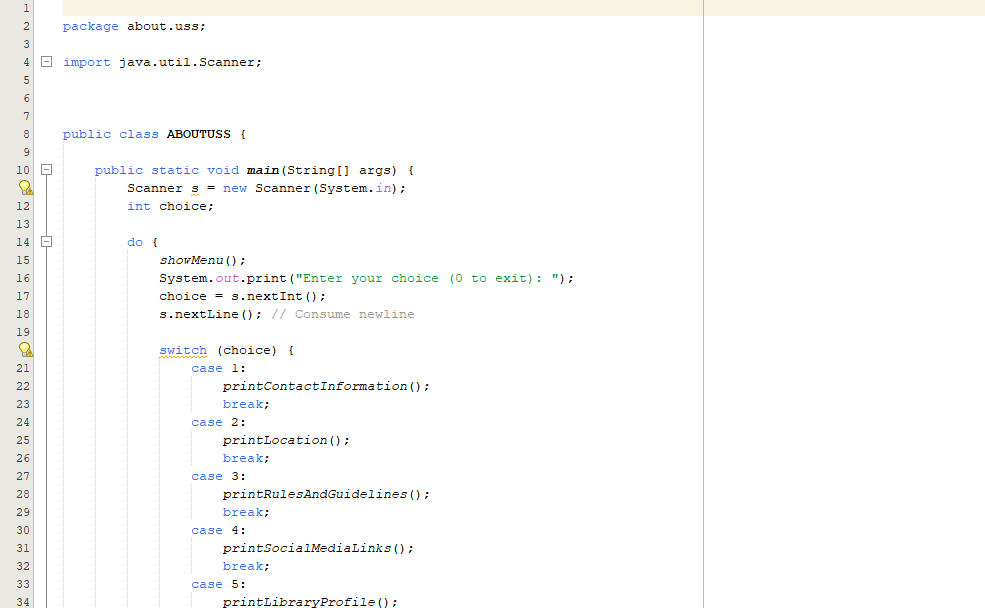
A user-friendly "Return to Menu" feature is included at the end of every section. This ensures that users do not have to restart the program after viewing information, enhancing navigation flow and overall usability.

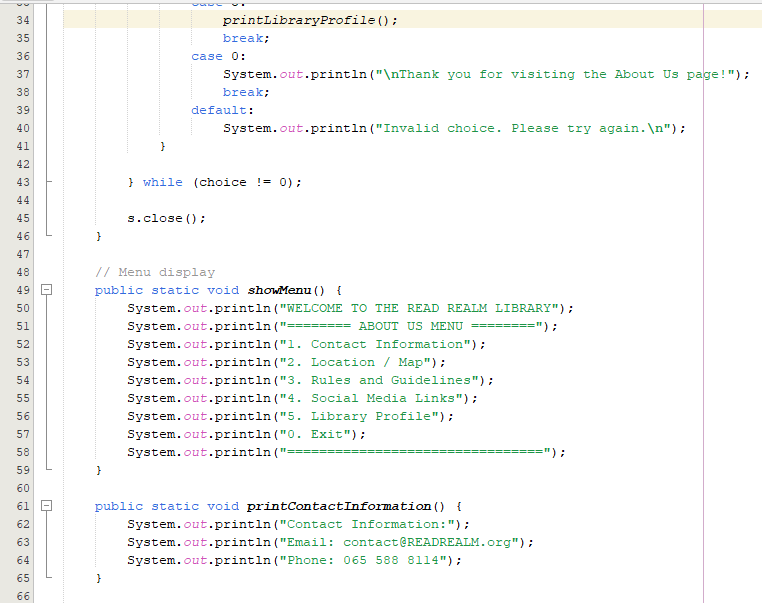
This is implemented using looping and method-calling mechanisms in Java, allowing users to return to the main menu with ease.

**8. Challenges Faced During Development**

| **Challenge** | **Description** |
| --- | --- |
| **User Navigation** | Designing a return-to-menu system without confusion required careful loop and method structure. |
| **Text Formatting** | Ensuring content like addresses and rules were displayed cleanly in the console took extra formatting logic. |
| **Readability** | Making long text (rules, social links) readable on the console required spacing and grouping. |
| **Back Function Design** | Providing smooth return functionality from every section without crashing the loop was a critical issue resolved through conditional handling. |
| **User Engagement** | Displaying content clearly without GUI was challenging. Efforts were made to use text symbols to enhance user experience. |

**About us Codes:**

****

****

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Documentation: User Communication System**

**1. Overview**

The **User Communication System** is a Java-based console application designed for library environments to streamline communication between library users and administrative staff. It acts as a two-way communication interface where:

* Admin can share important updates (like overdue notices and policy changes)
* Users can submit enquiries directly to management
* A return-to-menu function ensures ease of navigation within the system

This module aims to increase efficiency, reduce manual workload, and foster better engagement between users and library services.

**2. System Features**

**a) Admin Notifications**

The system reads from a file (notifications.txt) to display current messages from the library admin. These notifications may include:

* Overdue book alerts
* Upcoming policy changes
* Library closures or schedule changes

**File source:** notifications.txt

**📃 b) Policy Updates**

Policy notices are included within the same admin notifications. These may inform users about:

* New rules or conduct expectations
* Extended or reduced operating hours
* Special services or holiday adjustments

This keeps users up to date without needing to contact the library directly.

**c) User Enquiry Submission**

The system provides an option for users to type a message or enquiry to management, which is then saved to a file.

**File destination:** enquiries.txt

These messages could be:

* Book-related questions
* Service complaints
* Suggestions for improvement
* Membership or access requests

**d) Return to Menu Function**

After completing any action, users are returned to the main menu without needing to restart the program. This provides a continuous flow, allowing multiple actions in one session.

* Implemented using loop and method structures in Java
* Enhances usability for non-technical users

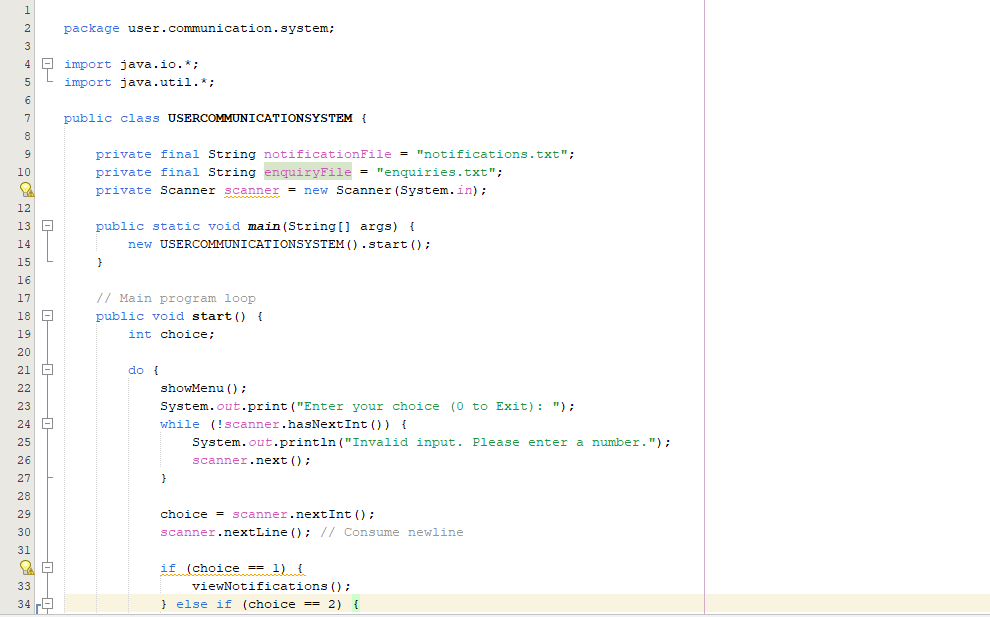
**3. Technical Implementation**

* **Language:** Java
* **Type:** Console application
* **Storage:** Text file I/O (read/write)
* **Modules:** Notification display, enquiry writing, enquiry viewing
* **User Interaction:** Menu-driven using Scanner and loop logic

**4. Challenges Faced During Development**

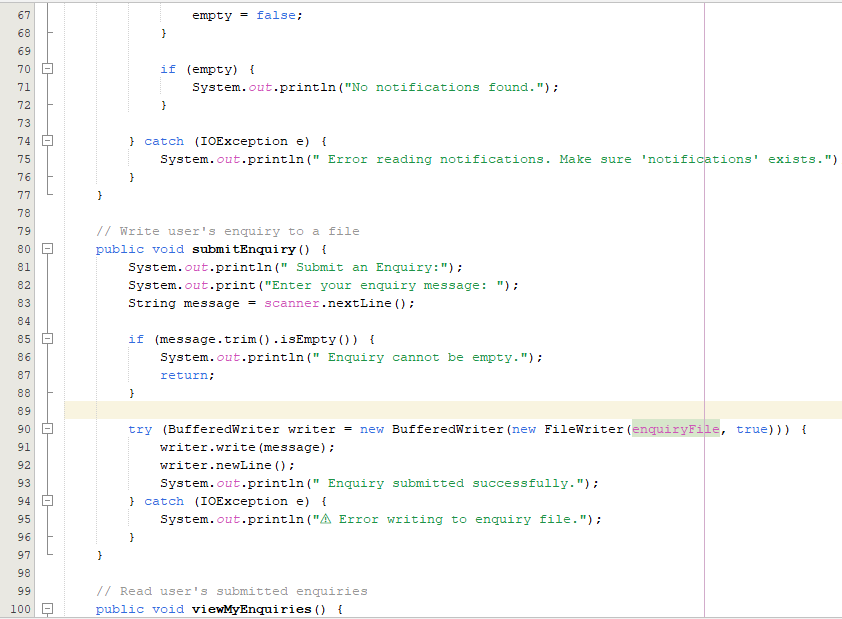
| **Challenge** | **Explanation** |
| --- | --- |
| **File Handling Errors** | Ensuring the system handles missing or empty .txt files without crashing. Default notices and error messages were added. |
| **Input Validation** | Managing user input (e.g., invalid numbers or empty enquiries) to avoid bugs or infinite loops. |
| **Return to Menu Logic** | Designing a seamless loop structure so users could return after each action without disrupting flow. |
| **Data Persistence** | Using .txt files is simple but not scalable; errors can occur if files are accidentally deleted or accessed improperly. |
| **User-Friendly Console** | Making the interface readable and easy to use without GUI required structured formatting and spacing. |
| **File Access Conflicts** | Preventing read-write errors when multiple users might try to write to the same file (future challenge in multi-user systems) |

**User Communication System codes:**

****

A screenshot of a computer program

AI-generated content may be incorrect.



A screenshot of a computer program

AI-generated content may be incorrect.

**LIBRARY MANAGEMENT SYSTEM – DATABASE DOCUMENTATION**

1.**PURPOSE**

This document provides a clear overview of the database schema for managing library members and tracking their book purchase transactions. It supports data storage, integrity, and easy reporting in a library system.

2**. DATABASE TABLES**

2.1. **Members Table**

Stores information about each user registered in the library.

**FIELD NAME DATA TYPE DESCRIPTION**

User\_ID VARCHAR(10) Unique identifier (e.g., 'M001')

First\_Name VARCHAR(50) Member's first name

Last\_Name VARCHAR(50) Member's last name

Username VARCHAR(50) Login username

Password VARCHAR(50) Encrypted password

Contact\_Info VARCHAR(100) Phone number or address

Email VARCHAR(100) Email address

**Example**:

CREATE TABLE Members (

User\_ID VARCHAR(10) PRIMARY KEY,

First\_Name VARCHAR(50),

Last\_Name VARCHAR(50),

Username VARCHAR(50),

Password VARCHAR(50),

Contact\_Info VARCHAR(100),

Email VARCHAR(100)

);

2.2. **BUY\_TRANSACTION TABLE**

Captures every book purchase made by members.

**Field Name Data Type Description**

Transaction\_ID INT Auto-incrementing purchase ID

User\_ID VARCHAR(10) Links to Members table

Book\_ID VARCHAR(10) ID of the purchased book

Amount DECIMAL(10,2) Total cost of purchase

Date\_of\_Purchase DATE When the purchase was made

**Example**:

CREATE TABLE Buy\_Transaction (

Transaction\_ID INT PRIMARY KEY AUTO\_INCREMENT,

User\_ID VARCHAR(10),

Book\_ID VARCHAR(10),

Amount DECIMAL(10,2),

Date\_of\_Purchase DATE,

FOREIGN KEY (User\_ID) REFERENCES Members(User\_ID),

FOREIGN KEY (Book\_ID) REFERENCES Book(Book\_ID)

);

> The User\_ID is a foreign key linking to Members, and Book\_ID links to Book (assuming a Book table exists with Book\_ID as primary key).

3. **Relationships**

One Member → Many Transactions (1:N)

One Book → Many Transactions (1:N)

4. **Sample Data Highlights**

**Members** (Example)

INSERT INTO Members VALUES

('M001', 'John', 'Doe', 'johndoe', 'pass123', '123-456-7890', 'john@example.com');

**Buy\_Transaction** (Example)

INSERT INTO Buy\_Transaction (User\_ID, Book\_ID, Amount, Date\_of\_Purchase) VALUES

('M001', 'B001', 120.50, '2024-01-15');

5. **Usage Scenario**

Track each user's total purchase history.

Generate reports on most sold books.

Notify members via email after transactions.

Use data for loyalty programs or discounts.

6**. Security & Best Practices**

Passwords should be stored with hashing (e.g., SHA256).

Emails should be unique for identification.

Enable foreign key constraints to maintain integrity.

The Library Management System is a software application designed to manage and automate library operations such as book tracking, member management, borrowing, returning, and reporting.

PURPOSE:

The Library Management System is designed to automate and streamline the operations of a library, making it more efficient, organized, and user-friendly. Its primary purpose is to digitally manage books, members, and transactions, reducing manual work and minimizing errors. The system helps librarians track book availability, record loans and returns, manage member details, and generate reports for better decision-making. Additionally, it provides users with an easy search functionality to quickly locate books by title, author, or category. By maintaining a centralized database, the LMS ensures accurate record-keeping, prevents book losses, and improves overall library services. It is particularly useful in schools, colleges, and public libraries, where managing large collections manually can be time-consuming and inefficient. Ultimately, the system enhances accessibility, productivity, and security in library operations.

System Objectives:

- Automate book cataloguing and inventory management.

- Track book loans, returns, and overdue fines.

- Manage library members (students, faculty, etc.).

- Generate reports (borrowed books, overdue books, etc.).

- Provide an efficient search system for books and members.

Nonjabulo Ndlovu

* **ROLE:**
* creating database using XAMPP.

**DATABASE** : Library Management System

**-Tables:**

- BookCatalog (Book\_ID, Book\_name, Author, Genre, Price, Availability, Language, Rating, Age Restriction)

-Regular\_Users(user\_id,first\_name,last\_name,username,password,email,contact\_info)

-Fines(fine\_id, user\_id,Book\_ID,Days\_late, fine\_Amount,paid\_syatus)

-LoanedBooks(record\_id, Book\_ID, user\_id,borrow\_date,due\_date,return\_date,reservation\_id)

-Reservation(reservation\_id, , Book\_ID, user\_id, reservation\_date, status)

-Admins(admin\_id, first\_name,last\_name,username,password,email,position)

-ReturnedBooks(return\_id, , user\_id,Book\_ID,record\_id,return\_date,days\_late,fine\_Amount)

* Populating the tables.

CHALLENGES:

Ensuring that database is normalized to avoid redundancy.

Ensuring referential integrity with foreign keys between tables.

**Coding roles for library system management: BY Tshifhiwa Nkhumeleni**

* Borrowing and returning books
* Rating and reviewing books
* Fines
* Book management
* challenges

The system is developed using **Java**, leveraging **Object-Oriented Programming** principles. Core components include:

* Classes for Book, User, Review, Fine, and Transaction
* Use of ArrayList, HashMap, and Date for storing and manipulating data
* Input/output handled through the console or a GUI for interaction
* Optionally backed by a database for persistent storage

**Borrowing and Returning Books**

**Borrowing Books**

* Users enter their ID and the Book ID to borrow a book.
* The system checks:
* Book availability
* User’s current borrowed books (limit control)
* A timestamp is generated and stored as the borrow date.
* A due date is calculated (e.g., 30 days from borrow date).

**Returning Books**

* Users enter their ID and Book ID to return a book.
* System checks the return date against the due date.
* If the book is late, a fine is calculated.
* Book availability status is updated.

Example:

static void returnBook() {

System.out.print("Enter your User ID: ");

String userId = scanner.nextLine();

System.out.print("Enter Book ID to return: ");

String bookId = scanner.nextLine();

File loanFile = new File(LOANS\_FILE);

List<String> updatedLines = new ArrayList<>();

boolean returned = false;

try (BufferedReader br = new BufferedReader(new FileReader(loanFile))) {

String line;

while ((line = br.readLine()) != null) {

String[] data = line.split(",");

if (data[0].equals(bookId) && data[1].equals(userId)) {

returned = true; // Skip this line (returning)

} else {

updatedLines.add(line);

}

}

} catch (IOException e) {

System.out.println("Error reading loans file.");

}

if (returned) {

try (BufferedWriter bw = new BufferedWriter(new FileWriter(LOANS\_FILE))) {

for (String line : updatedLines) {

bw.write(line);

bw.newLine();

}

System.out.println("Book returned successfully.");

} catch (IOException e) {

System.out.println("Error updating loans file.");

}

} else {

System.out.println("No matching loan record found.");

}

**Rating and Reviewing Books**

* After reading a book, users can rate it (1 to 5 stars) and leave a comment.
* The system stores reviews linked to the user and book.
* Ratings are averaged per book and can be used for sorting or recommendation.

**Example:**

class Review {

String userId;

String bookId;

String comment;

int rating; // 1 to 5

public Review(String userId, String bookId, String comment, int rating) {

this.userId = userId;

this.bookId = bookId;

this.comment = comment;

this.rating = rating;

}

}

**Fines**

* A fine is charged for late book returns.
* Fine amount can be fixed (e.g., R 200 ).
* A Fine Details class keeps track of:
  + Fine amount
  + Reason (e.g., "Late return")
  + Date issued

**Book Management**

* Admins/librarians can:
  + Add new books
  + Remove books
  + Edit book details (title, author, category)
  + Check current availability
* Each book has:
  + Unique ID
  + Title, Author, Year, Genre
  + Availability status
  + Total rating and number of reviews

Example:

public void start() {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nLibrary Management System");

System.out.println("1. Add Book");

System.out.println("2. Update Book");

System.out.println("3. Delete Book");

System.out.println("4. List All Books");

System.out.println("5. Search Books");

System.out.println("6. Check Out Book");

System.out.println("7. Check In Book");

System.out.println("8. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

addBook(inputBookDetails(scanner));

break;

case 2:

System.out.print("Enter Book ID to update: ");

int updateId = scanner.nextInt();

scanner.nextLine();

Book updateBook = inputBookDetails(scanner);

updateBook(updateId, updateBook.getTitle(), updateBook.getAuthor());

break;

case 3:

System.out.print("Enter Book ID to delete: ");

int deleteId = scanner.nextInt();

deleteBook(deleteId);

break;

case 4:

listBooks();

break;

case 5:

System.out.print("Enter search query (title or author): ");

String query = scanner.nextLine();

searchBooks(query);

break;

case 6:

System.out.print("Enter Book ID to check out: ");

int checkoutId = scanner.nextInt();

checkOutBook(checkoutId);

break;

case 7:

System.out.print("Enter Book ID to check in: ");

int checkinId = scanner.nextInt();

checkInBook(checkinId);

break;

case 8:

System.out.println("Exiting...");

return;

default:

System.out.println("Invalid choice, please enter a number between 1 and 8.");

break;

}

}

}

**challenges**

* The code took long than I expected
* I couldn’t finish coding because of low shedding
* One of my code was not working properly I get help from one of our members in the group ( Lesego M )
* I created multiple code and they didn’t work so I had to do it over again until I get it right
* I didn’t have money to buy data to do research it was challenging and annoying at the same time because I was running out of time

### **Conclusion**

A Library System Management platform must be robust, user-friendly, and scalable. It should offer seamless experiences in borrowing, returning, reviewing, and managing books while ensuring fair enforcement of rules like fines. Despite the challenges in design and implementation, a well-developed system significantly improves library efficiency and user satisfaction.

Our roles:

|  |  |
| --- | --- |
| ***SYSTEM:*** | ***ROLE:*** |
| Book Catalog   * Show Book Name, Genre,   Language, Age restrictions if needed.   * Display Book prices for buy options * Release/Published date * Details about the book | Anthony |
| ‘About’ section   * Contact Info * Address * Rules and guidelines * Social Media Links * Library Profile (opening hours) * Return to menu/back system | Takudzwa |
| Log in/sign up incl membership/regular Staff & Admin etc.   * User must be able to sign up to membership, but it must be costly * Log in as Admin/User * Option to create an account (incl.   ID number, First/last name, Terms and Conditions agreement, set password (two step)).   * Show name of library on top (e.g.   Cozy Library).   * Exit function. | Takudzwa/ Lesego M |
| Buy transections   * User must inert ID * Amount Received in Account * Date of Purchase * Remove book from database permanently | Anthony / Tshifiwa |
| Fines (ADMIN)  • Deduct money from user accounts for overdue books/ damaged books | Tshifiwa / Lesego M |
| Book Loaning   * Track user in admin section * Borrow book * User should enter ID number to loan book . * Should be given a 30 due date * Reservation for unavailable books | Tshifiwa / Anthony |

|  |  |
| --- | --- |
|  |  |
| User Communication system   * User should receive notification from Admin for overdue books etc. * Policy updates * And a section for user to insert an enquiry to management * Return to Menu function | Anthony / Takudzwa |
| Book Search and filtering   * Efficient search functionality * Book Name * Author * Availability * Price range | Anthony |
| User Management System (ADMIN)   * Member registration * Update details * View / Search * Add/remove members * Borrowing/purchase/rent history * Access all members details * Manage reservations * Generate reports(fines, borrowed   etc) | Lesego M |
| Book Management (ADMIN)   * Add/update or remove books * Remove books if purchased * Stock * Insert book ID, Price etc (adds to data base) | Anthony / Lesego M |
| Returning books   * User ID must be entered * User mut insert book id * It must update on the data base   (txt file) | Tshifiwa / Takudzwa |
| Reviews and Ratings   * User should Enter ID * User should enter book ID * Opinion/Comment * Rating (1-5) | Tshifiwa |
| Reservations   * User should enter ID * User should Enter Book ID * User should set date | Anthony/ Lesego M |

|  |  |
| --- | --- |
| * Set reservation if book is currently non available   (e.g. “would you like to reserve the book”)   * Add to txt file data base |  |
| Log Out  • User should log out from System | Lesego M/ Takudzwa |

NB: return function must be available

Data base management

|  |  |
| --- | --- |
| ***DATABASE:*** | ***ROLE:*** |
| BookCatalog   * Book\_ID (200 books) * Book\_Name (200) * Author * Genre * Price * Availability * Language * Rating (1-5) * Age Resetriction (yes/no) | Ndivhuwo |
| RegularUsers   * User\_ID (300 users) (“U001”) * First\_Name (300) * Last\_Name * Username (e.g.   ‘young.45Emkay’)   * Password * Contact\_Info * Email | Nonjabulo / Ndivhuwo |
| Members   * User\_ID (300 users) (“M001”) * First\_Name * Last\_Name * Username * Password * Contact\_Info * Email | Ndivhuwo |
| BuyTransection   * User\_ID * Book\_ID * Amount * DateOf\_Purchase | Anthony |
| Fines  • Fine\_ID (“F1003”) | Ndivhuwo/Nonjabulo |
| * User\_ID * Book\_ID * Days\_late * Fine\_Amount * Paid\_Status (unpaid/paid) |  |
| LoanedBooks   * User\_ID * Record\_ID (BR1001) * Book\_ID * Borrow\_Date * Due\_Date * Return\_Date (null/ 2025-05-10) * Reservation\_ID(null,RZ001) | Nonjabulo |
| Reservation   * User\_ID * Reservation\_ID(“RZ001”) * Book\_ID * Reservation\_Date * Status (ready/not ready/cancelled) | Ndivhuwo |
| Admin   * Admin\_ID (“A001”) (50 people) * First\_Name * Last\_Name * Username * Position (Liberian, Manager etc) * Password * Email | Nonjabulo |
| ReturnedBook   * User\_ID * Book\_ID * Record\_ID (FOREIGN KEY<Loanedbooks) * Return\_Date * Days\_late (FOREIGN KEY) * Fine\_Amount (FOREIGN KEY) | Ndivhuwo |