Submitted by -

|  |  |  |  |
| --- | --- | --- | --- |
| **Saurabh**  **saurabh.24scse1010928@galgotiasuniversity.ac.in** | **Active** | **Member** |  |
| **Arpit Poddar**  **arpitpoddar999@gmail.com** | **Active** | **Admin** |  |
| **Ashutosh Kumar Ray**  **ashutosh.24scse1010947@galgotiasuniversity.ac.in** | **Active** | **Member** |  |
| **Abhishek sharma**  **abhishek.24scse1010225@galgotiasuniversity.ac.in** | **Active** | **Member** |  |

**1. Introduction**

This project is a console-based Java application simulating a basic library management system. It leverages file handling for data persistence and adheres to object-oriented programming principles. The system allows users to register, log in, search for books, issue or return them, and manage their personal activity reports.

**2. Project Objectives**

The primary objectives of this project were to:

* Develop a user-friendly library system for basic library operations.
* Implement **CRUD** (Create, Read, Update, Delete) operations for users and books.
* Utilize **file handling** (text files) as a simple mechanism for data persistence, avoiding the need for a database.
* Apply core **Object-Oriented Programming (OOP)** concepts such as inheritance, encapsulation, and polymorphism.
* Ensure robust **data validation** and comprehensive **error handling** throughout the system.

**3. Technologies Used**

The project was developed using the following technologies:

* **Java**: The core programming language used for implementing all application logic, object-oriented structures, and file input/output (I/O) operations.
* **Text Files**: users.txt and books.txt serve as local storage for user and book data, respectively, eliminating the need for an external database.
* **Scanner Class**: Used for reading user input from the console, enabling interactive menu navigation.
* **Java's Collection Framework**: Specifically, ArrayList is used to manage collections of user and book objects in memory.

**4. Features**

The Library Management System includes the following key features:

* **User Registration & Login**: Allows new users to register with unique IDs and existing users to log in securely.
* **Book Search**: Users can search for books by their title or author.
* **Issue Book**: Facilitates the borrowing of books by logged-in users. Includes dynamic book addition if a requested book is not found in the system.
* **Return Book**: Enables users to return borrowed books, updating their status and availability.
* **Add Book**: Functionality for administrators (or authenticated users) to add new books to the library's collection.
* **Generate Borrowed Books Report**: Provides users with a report of the books they have currently borrowed.

**5. Code Structure**

The project's code is organized into distinct layers to promote modularity and maintainability:

**1. Model Layer**

Defines the core entities of the system:

* Person: A base class (or interface) representing common attributes for individuals in the system.
* User: Extends Person, adding specific user attributes like password and a borrowedBooksList.
* Book: Represents a book with attributes such as title, author, and its current status (e.g., issued or available).

**2. DAO Layer (Data Access Object)**

Responsible for interacting with the data persistence layer (text files):

* BookDAO: Handles loading book data from and saving it to books.txt.
* UserDAO: Manages loading user data from and saving it to users.txt.

**3. LibrarySystem Class**

Contains the core application logic:

* Implements methods for user registration, login, book searching, issuing, and returning.
* Acts as the central orchestrator for interactions between the user interface and the DAO layer.

**4. Main Method**

* Provides the entry point for the application.
* Presents a **menu-driven interface** to the user using the Scanner class for interaction.

**6. Sample Outputs**

Here's an example of user interaction with the system:

=== GUVI Library Management System ===

Menu:

0. Register

1. Login

2. Search Book

3. Issue Book

...

Choose an option: 0

Enter new user ID: 101

Enter your name: Alice

Enter a password: pass123

Registration successful.

**7. Error Handling**

The system incorporates various error handling mechanisms to ensure robustness:

* **Try-catch blocks**: Used to gracefully handle IOException during file operations (reading from and writing to users.txt and books.txt).
* **Validation messages**: Provided to the user in case of incorrect login credentials or invalid input.
* **Null checks**: Implemented to prevent NullPointerExceptions, for example, by checking if a user is logged in before allowing certain operations.
* **Input validation**: Includes checks like scanner.hasNextInt() to handle invalid menu input from the user.

**8. Data Validation**

To maintain data integrity and a smooth user experience, the project includes:

* **Unique User ID Check**: Ensures that each registered user has a unique ID, preventing duplicate user accounts.
* **Password Matching**: Verifies that the entered password matches the stored password during the login process.
* **Book Title Checks**: Performed before issuing or returning books to ensure the book exists and its status is appropriate.
* **Dynamic Book Addition Prompt**: If a book is not found during an issue request, the system prompts the user to add the book.

**9. Future Enhancements**

The current system provides a solid foundation that can be extended with several enhancements:

* **Role-Based Access Control**: Implement distinct roles (e.g., admin, librarian, regular user) with different access privileges.
* **Graphical User Interface (GUI)**: Develop a more interactive and visually appealing interface using JavaFX or Swing.
* **Database Integration**: Migrate from text file storage to a robust database system (e.g., MySQL, SQLite) for better data management, scalability, and concurrent access.
* **Advanced Library Features**: Introduce functionalities such as due dates, fine calculation for overdue books, and a book reservation system.
* **Unit Testing**: Implement unit tests using frameworks like JUnit to ensure the correctness and reliability of individual code components.

**10. Conclusion**

This project effectively demonstrates the practical application of Java's core features, including Object-Oriented Programming (OOP), file handling, and the Collection Framework. It serves as a strong foundational project that can be further developed into a more comprehensive and sophisticated library management system with advanced features like GUI and database integration.

**11. GitHub Link**

The source code for this project is available on GitHub:

https://github.com/BerlinXAP/Library-managment-system/upload/main

Main code

import java.io.\*;

import java.util.\*;

public class Guvi{

// === Model Classes ===

// Layered architecture to perform CRUD operation

static class Person {

protected int id;

protected String name;

public Person(int id, String name) {

this.id = id;

this.name = name;

}

}

static class User extends Person {

private String password;

private List<Book> borrowedBooks;

public User(int id, String name, String password) {

super(id, name);

this.password = password;

this.borrowedBooks = new ArrayList<>();

}

public int getId() { return id; }

public String getName() { return name; }

public boolean checkPassword(String password) { return this.password.equals(password); }

public void borrowBook(Book book) { borrowedBooks.add(book); }

public void returnBook(Book book) { borrowedBooks.remove(book); }

public List<Book> getBorrowedBooks() { return borrowedBooks; }

}

static class Book {

private int id;

private String title;

private String author;

private boolean isIssued;

public Book(int id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

this.isIssued = false;

}

public int getId() { return id; }

public String getTitle() { return title; }

public String getAuthor() { return author; }

public boolean isIssued() { return isIssued; }

public void setIssued(boolean issued) { this.isIssued = issued; }

public void showInfo() {

System.out.println(id + ": " + title + " by " + author + (isIssued ? " (Issued)" : ""));

}

}

// === DAO Classes ===

static class BookDAO {

private static final String FILE\_NAME = "books.txt";

// Implement IO file connectivity from Java

// Auto-create files to perform CRUD

public static List<Book> loadBooks() {

List<Book> books = new ArrayList<>();

try (BufferedReader br = new BufferedReader(new FileReader(FILE\_NAME))) {

String line;

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

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

int id = Integer.parseInt(parts[0]);

String title = parts[1];

String author = parts[2];

boolean isIssued = Boolean.parseBoolean(parts[3]);

Book book = new Book(id, title, author);

book.setIssued(isIssued);

books.add(book);

}

} catch (IOException e) {

System.out.println("No existing book file found. Starting fresh.");

}

return books;

}

public static void saveBooks(List<Book> books) {

try (PrintWriter pw = new PrintWriter(new FileWriter(FILE\_NAME))) {

for (Book book : books) {

pw.println(book.getId() + "," + book.getTitle() + "," + book.getAuthor() + "," + book.isIssued());

}

} catch (IOException e) {

System.out.println("Error saving books: " + e.getMessage());

}

}

}

static class UserDAO {

private static final String FILE\_NAME = "users.txt";

public static List<User> loadUsers() {

List<User> users = new ArrayList<>();

try (BufferedReader br = new BufferedReader(new FileReader(FILE\_NAME))) {

String line;

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

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

int id = Integer.parseInt(parts[0]);

String name = parts[1];

String password = parts[2];

users.add(new User(id, name, password));

}

} catch (IOException e) {

System.out.println("No existing user file found. Starting fresh.");

}

return users;

}

public static void saveUsers(List<User> users) {

try (PrintWriter pw = new PrintWriter(new FileWriter(FILE\_NAME))) {

for (User user : users) {

pw.println(user.getId() + "," + user.getName() + "," + user.password);

}

} catch (IOException e) {

System.out.println("Error saving users: " + e.getMessage());

}

}

}

// === Core Library System ===

static class LibrarySystem {

List<Book> books;

List<User> users;

User currentUser;

int nextBookId = 1;

public LibrarySystem() {

books = BookDAO.loadBooks();

users = UserDAO.loadUsers();

for (Book b : books) if (b.getId() >= nextBookId) nextBookId = b.getId() + 1;

}

public void registerUser(Scanner sc) {

System.out.print("Enter new user ID: ");

int newId = sc.nextInt(); sc.nextLine();

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

String name = sc.nextLine();

System.out.print("Enter a password: ");

String password = sc.nextLine();

// Input validation

for (User user : users) if (user.getId() == newId) {

System.out.println("User ID already exists."); return; }

users.add(new User(newId, name, password));

UserDAO.saveUsers(users);

System.out.println("Registration successful.");

}

public void login(Scanner sc) {

System.out.print("Enter user ID: ");

int userId = sc.nextInt(); sc.nextLine();

System.out.print("Enter password: ");

String pwd = sc.nextLine();

for (User user : users) if (user.getId() == userId && user.checkPassword(pwd)) {

currentUser = user;

System.out.println("Login successful. Welcome, " + user.getName());

return;

}

System.out.println("Invalid credentials."); // Error messages and feedback

}

public void searchBook(Scanner sc) {

sc.nextLine();

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

String query = sc.nextLine().toLowerCase();

boolean found = false;

for (Book book : books) {

if (book.getTitle().toLowerCase().contains(query) ||

book.getAuthor().toLowerCase().contains(query)) {

book.showInfo(); found = true;

}

}

if (!found) System.out.println("No book found.");

}

public void issueBook(Scanner sc) {

if (currentUser == null) {

System.out.println("Login first."); return;

}

sc.nextLine();

System.out.print("Enter book title to issue: ");

String title = sc.nextLine();

for (Book book : books) if (book.getTitle().equalsIgnoreCase(title)) {

if (!book.isIssued()) {

book.setIssued(true);

currentUser.borrowBook(book);

BookDAO.saveBooks(books);

System.out.println("Book issued.");

} else System.out.println("Book is already issued.");

return;

}

System.out.print("Book not found. Add and issue it? (yes/no): ");

if (sc.nextLine().equalsIgnoreCase("yes")) {

System.out.print("Enter author: ");

String author = sc.nextLine();

Book newBook = new Book(nextBookId++, title, author);

newBook.setIssued(true);

books.add(newBook);

currentUser.borrowBook(newBook);

BookDAO.saveBooks(books);

System.out.println("Book added and issued.");

}

}

public void returnBook(Scanner sc) {

if (currentUser == null) {

System.out.println("Login first."); return;

}

sc.nextLine();

System.out.print("Enter book title to return: ");

String title = sc.nextLine();

for (Book book : currentUser.getBorrowedBooks()) {

if (book.getTitle().equalsIgnoreCase(title)) {

book.setIssued(false);

currentUser.returnBook(book);

BookDAO.saveBooks(books);

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

return;

}

}

System.out.println("Book not in your list.");

}

public void generateReport() {

if (currentUser == null) {

System.out.println("Login first."); return;

}

// Accuracy of Output

System.out.println("=== Report for " + currentUser.getName() + " ===");

if (!currentUser.getBorrowedBooks().isEmpty()) {

for (Book b : currentUser.getBorrowedBooks()) System.out.println("- " + b.getTitle());

} else System.out.println("No books borrowed.");

}

public void addBook(Scanner sc) {

if (currentUser == null) {

System.out.println("Login first."); return;

}

sc.nextLine();

System.out.print("Enter book title: ");

String title = sc.nextLine();

System.out.print("Enter author: ");

String author = sc.nextLine();

books.add(new Book(nextBookId++, title, author));

BookDAO.saveBooks(books);

System.out.println("Book added.");

}

}

// === Main Method ===

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

LibrarySystem lib = new LibrarySystem();

int choice = -1;

System.out.println("=== GUVI Library Management System ===");

do {

System.out.println("\nMenu:");

System.out.println("0. Register");

System.out.println("1. Login");

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

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

System.out.println("4. Return Book");

System.out.println("5. Generate Report");

System.out.println("6. Add Book");

System.out.println("7. Exit");

System.out.print("Choose an option: ");

if (scanner.hasNextInt()) {

choice = scanner.nextInt();

} else {

System.out.println("Invalid input.");

scanner.next(); // Consume bad input

continue;

}

switch (choice) {

case 0 -> lib.registerUser(scanner);

case 1 -> lib.login(scanner);

case 2 -> lib.searchBook(scanner);

case 3 -> lib.issueBook(scanner);

case 4 -> lib.returnBook(scanner);

case 5 -> lib.generateReport();

case 6 -> lib.addBook(scanner);

case 7 -> {

BookDAO.saveBooks(lib.books);

UserDAO.saveUsers(lib.users);

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

}

default -> System.out.println("Invalid option.");

}

} while (choice != 7);

scanner.close();

    }

}