**TermiLib – A CLI-powered library management system that handles book inventory, user roles, and borrowing records efficiently with persistent file storage.**

**Project Report**

**Table of Contents**

1. Title Page
2. Abstract
   1. Introduction
   2. 1.1 Problem Description
   3. 1.2 Project Description
   4. 1.3 Objective
   5. 1.4 Scope
   6. Task Distribution
   7. Features of the System
   8. System Functionality
   9. Modules or Entities
   10. 5.1 Book Entity and Management
   11. 5.2 User Entity and Management
   12. 5.3 Transaction Entity and Management
   13. 5.4 Authentication Module
   14. 5.5 Utility Modules
       * 5.5.1 File Handling
       * 5.5.2 Input Validation
   15. Implementation Details
   16. 6.1 Data Structures
   17. 6.2 File Handling and Persistence
   18. 6.3 Input Validation
   19. 6.4 Exception Handling
   20. 6.5 User Roles and Access Control
   21. 6.6 Menu System and Flow
   22. Tools and Technologies Used
   23. Project Timeline
   24. Code and Screenshots of Outputs
   25. 9.1 Main Class Snippet (TermiLib.java)
   26. 9.2 Login and Registration
   27. 9.3 Admin Menu and Operations
       * 9.3.1 Book Management
       * 9.3.2 User Management
       * 9.3.3 Transaction Management & Reports
   28. 9.4 Student Menu and Operations
   29. 9.5 Data Persistence (File Examples)
   30. 9.6 Error Handling and Validation Examples
   31. Beyond Current Focus (Limitations)
   32. Future Enhancements
   33. Conclusion
   34. Reference Citations

**1. Title Page**  
**(You would create this as a separate first page)**

**(Centered on Page)**

**TermiLib**

A CLI-powered library management system that handles book inventory, user roles, and borrowing records efficiently with persistent file storage.

A Project Report

Submitted in partial fulfillment of the requirements for  
[Course Name/Purpose of Submission]

by

Muhammad Ahmad Sultan  
[Your Student ID, if applicable]

[Date of Submission]

[Institution/Department, if applicable]

**2. Abstract**

TermiLib is a Command Line Interface (CLI) based Library Management System developed in Java. The system aims to provide a robust and user-friendly platform for managing library operations, including book inventory, user accounts (Admin and Student), and book borrowing/returning transactions. Key features include distinct user roles with specific functionalities, comprehensive CRUD (Create, Read, Update, Delete) operations for books and users, persistent data storage using text files, input validation, and basic reporting. The system utilizes arrays for in-memory data management and employs exception handling to ensure stability. TermiLib is designed to be menu-driven, offering a clear and efficient workflow for both administrators and students interacting with the library's resources.

**1. Introduction**

**1.1 Problem Description**  
Traditional library management often involves manual record-keeping, which can be time-consuming, error-prone, and inefficient. Tracking book availability, user borrowing history, due dates, and managing user accounts manually can lead to discrepancies, lost records, and a poor user experience. There is a need for a digitized system that can automate these processes, provide quick access to information, ensure data integrity, and streamline library operations for both staff and patrons. A CLI-based system offers a lightweight and accessible solution, especially in environments where graphical interfaces may not be necessary or readily available.

**1.2 Project Description**  
"TermiLib – A CLI-powered library management system that handles book inventory, user roles, and borrowing records efficiently with persistent file storage." This project implements a comprehensive library management system accessible via a command-line interface. It allows administrators to manage the book catalog (add, update, remove, search books) and user accounts. Students can register, search for books, borrow, and return them. All data related to books, users, and transactions is persistently stored on disk in text files, ensuring data retention across sessions. The system incorporates input validation and error handling to maintain data integrity and prevent crashes.

**1.3 Objective**  
The primary objectives of the TermiLib project are:

* To develop a functional CLI-based library management system.
* To implement separate login systems and functionalities for Admin and Student users.
* To provide comprehensive CRUD operations (Add, View, Update, Search) for managing books and users.
* To manage book borrowing and returning transactions.
* To store all system records persistently using file I/O.
* To ensure data integrity through proper input validation.
* To build a stable system with robust exception handling to prevent crashes.
* To create a menu-driven interface for easy navigation and usability.

**1.4 Scope**  
The scope of the TermiLib system includes:

* **User Management:** Registration for students, login for students and admins. Admins can view, search, update, and remove users.
* **Book Management:** Admins can add, view all/available, update, remove, and search books by title, author, ISBN, or category.
* **Transaction Management:** Students can borrow and return books. Admins can view all transactions, view overdue books, and force return books.
* **Data Persistence:** All book, user, and transaction records are stored in text files (books.txt, users.txt, transactions.txt).
* **Reporting:** Basic reports like most borrowed books, active borrowers, and system statistics.
* **Input Validation:** Validation for fields like age, year, email, and general input formats.
* **Error Handling:** Basic error handling to prevent crashes from invalid input or unexpected situations.

**Out of Scope for the current version:**

* Graphical User Interface (GUI).
* Advanced fine calculation system.
* Online book reservation or renewal.
* Integration with external databases (uses flat files).
* Concurrency control for multiple simultaneous users (designed for single-user CLI interaction at a time).
* Complex search algorithms or natural language processing.

**2. Task Distribution**

This project, "TermiLib," was developed by Muhammad Ahmad Sultan. All aspects of the project, including design, coding, testing, and documentation, were handled by the aforementioned individual.

*(If this were a group project, you'd list members and their responsibilities, e.g.:)*

* *Muhammad Ahmad Sultan: Project Lead, Backend Logic (Managers, File I/O), Admin Module.*
* *Member 2: User Interface (CLI Menus), Student Module, Input Validation.*
* *Member 3: Data Models, Testing, Documentation.*

**3. Features of the System**

TermiLib offers a range of features essential for library management:

* **Menu-Driven Interface:** Intuitive and easy-to-navigate command-line menus for all user interactions.
* **Dual User Roles:**
  + **Admin:** Full control over book inventory, user management, transaction oversight, and system reports.
  + **Student:** Can register, search books, view available books, borrow books, return books, and view their borrowing history.
* **Secure Login System:** Separate authentication for Admins and Students. A default admin account is created on first run.
* **Book Inventory Management:**
  + Add new books with details like title, author, ISBN, publication year, category, and total copies.
  + View all books or only available books.
  + Update existing book information.
  + Remove books from the inventory (if no copies are borrowed).
  + Search books by title, author, ISBN, or category.
* **User Account Management (Admin):**
  + View all registered users.
  + Search users by name, username, or email.
  + Update user details (name, age, email, password).
  + Remove user accounts (cannot remove the last admin).
* **Transaction Management:**
  + Students can borrow available books (limit of 5 books per student).
  + Students can return borrowed books.
  + Admins can view all transactions, list overdue books, and forcefully return a book using a transaction ID.
  + Automatic due date calculation (14 days from borrowing).
* **Persistent Data Storage:**
  + Book, user, and transaction records are saved to text files (books.txt, users.txt, transactions.txt) in a data subdirectory, ensuring data is retained between sessions.
* **Input Validation:**
  + Validation for various fields like age (16-100), publication year (1000-2024), email format, non-empty fields, and numeric inputs.
* **Exception Handling:**
  + Robust error handling to prevent system crashes due to invalid inputs (e.g., non-numeric input where numbers are expected) or file I/O issues.
* **Reporting (Admin):**
  + Display most borrowed books.
  + List active borrowers.
  + Show system statistics (total books, available books, borrowed books, total users, total transactions, active borrowings).
* **Automatic ID Generation:** Unique IDs are automatically generated for new books, users, and transactions.

**4. System Functionality**

TermiLib operates through a hierarchical menu system. Upon starting the application, the user is presented with a Login Menu.

1. **Initial Interaction:**
   * Users can choose to Login, Register as a Student, or Exit.
   * A default admin account (admin/admin) is created if it doesn't exist, facilitating initial system access.
   * Student registration prompts for username, password, name, age, and email, with validation.
2. **Post-Login:**
   * **Admin:** If login is successful as an Admin, the Admin Menu is displayed. This menu provides access to:
     + Book Management (Add, View, Update, Remove, Search)
     + User Management (View, Search, Update, Remove)
     + Transaction Management (View All, View Overdue, Force Return)
     + Reports (Most Borrowed, Active Borrowers, Statistics)
     + Logout
   * **Student:** If login is successful as a Student, the Student Menu is displayed. This menu allows:
     + Search Books
     + View Available Books
     + Borrow Book (by ID)
     + Return Book (by ID)
     + View My Borrowing History
     + Logout
3. **Operations:**
   * All operations involving data modification (add, update, remove) trigger an immediate save to the respective data files.
   * Search operations allow users to find specific records based on various attributes.
   * The system provides feedback messages for successful operations, errors, or when no data is found.
   * Exception handling ensures that incorrect input types (e.g., text instead of numbers) do not crash the application but instead prompt the user to try again.
4. **Data Flow:**
   * User input is captured via Scanner.
   * The TermiLib main class routes requests to the appropriate manager (BookManager, UserManager, TransactionManager, AuthManager).
   * Managers interact with model objects (Book, User, Transaction) and handle the business logic, including array manipulation.
   * For persistence, managers use FileHandler to read from and write to text files.
   * InputValidator is used by various classes to ensure data integrity before processing.

**5. Modules or Entities**

The system is organized into several key modules and entities:

**5.1 Book Entity and Management (models/Book.java, managers/BookManager.java)**

* **Entity (Book.java):** Represents a book with attributes: id, title, author, isbn, publicationYear, category, totalCopies, and availableCopies. It includes methods for converting to/from a file string (toFileString, fromFileString) and checking availability (isAvailable).
* **Management (BookManager.java):** Handles all operations related to books.
  + addBook(): Adds a new book to the books array and saves to file, ensuring ISBN uniqueness and generating a new ID.
  + viewAllBooks()/viewAvailableBooks(): Displays all books or only those with available copies.
  + updateBook(): Modifies details of an existing book based on its ID. Allows selective updates.
  + removeBook(): Removes a book if no copies are currently borrowed. Shifts array elements.
  + searchBy...(): Provides search functionality by title, author, ISBN, or category.
  + Handles loading books from books.txt on startup and saving changes back.

**5.2 User Entity and Management (models/User.java, managers/UserManager.java)**

* **Entity (User.java):** Represents a user with attributes: id, username, password, name, age, email, and role (ADMIN or STUDENT). Includes methods for converting to/from a file string.
* **Management (UserManager.java):** Manages user accounts.
  + addUser(): Adds a new user (typically for student registration or initial admin setup) to the users array and saves to file.
  + viewAllUsers(): Displays all registered users.
  + updateUser(): Allows an admin to modify user details (name, age, email, password).
  + removeUser(): Removes a user, preventing removal of the last admin. Shifts array elements.
  + searchBy...(): Provides search functionality by name, username, or email.
  + getUserById()/getUserByUsername(): Retrieves user objects.
  + Handles loading users from users.txt on startup and saving changes.

**5.3 Transaction Entity and Management (models/Transaction.java, managers/TransactionManager.java)**

* **Entity (Transaction.java):** Represents a borrowing record with attributes: id, userId, bookId, borrowDate, dueDate, returnDate, and status (BORROWED, RETURNED, OVERDUE). Dates are handled using java.time.LocalDate. Includes methods for file string conversion and checking if overdue.
* **Management (TransactionManager.java):** Manages book borrowing and returning processes.
  + borrowBook(): Creates a new transaction when a student borrows a book. Updates book availability via BookManager. Checks borrowing limits.
  + returnBook(): Updates an existing transaction when a student returns a book. Updates book availability.
  + forceReturnBook(): Allows an admin to mark a book as returned using the transaction ID.
  + viewAllTransactions()/viewUserHistory()/viewOverdueBooks(): Displays transaction records based on different criteria.
  + showMostBorrowedBooks()/showActiveBorrowers(): Generates simple reports.
  + Handles loading transactions from transactions.txt and saving changes. Depends on BookManager and UserManager for context.

**5.4 Authentication Module (auth/AuthManager.java)**

* **Management (AuthManager.java):** Responsible for user authentication.
  + authenticate(): Verifies user credentials (username and password) against records fetched via UserManager.
  + isAdmin()/isStudent(): Helper methods to check the role of an authenticated user.
  + It takes a UserManager instance in its constructor to access user data for authentication.

**5.5 Utility Modules (utils/)**

\*\*5.5.1 File Handling (`utils/FileHandler.java`)\*\*

\* Provides centralized methods for reading from and writing to text files.

\* `readFromFile()`: Reads all lines from a specified file within the `data` directory into a string array. Creates the `data` directory if it doesn't exist.

\* `writeToFile()`: Writes an array of strings to a specified file, overwriting existing content.

\* Handles basic file I/O exceptions by printing error messages.

\*\*5.5.2 Input Validation (`utils/InputValidator.java`)\*\*

\* Contains static methods for validating common input types.

\* Includes checks for `isValidEmail()`, `isValidAge()`, `isValidYear()`, `isValidUsername()`, `isValidPassword()`, `isNotEmpty()`, and `isValidISBN()`.

\* These methods are used throughout the application, particularly in the `TermiLib` main class and manager classes, to ensure data integrity before processing or storage.

**6. Implementation Details**

**6.1 Data Structures**  
The primary data structure used for storing records of entities (Books, Users, Transactions) in memory is **arrays**.

* Book[] books in BookManager
* User[] users in UserManager
* Transaction[] transactions in TransactionManager  
  Each array has a fixed maximum capacity (e.g., MAX\_BOOKS = 1000). A separate integer variable (e.g., bookCount) tracks the current number of active records in each array.  
  When a record is removed, elements are shifted to fill the gap, and the count is decremented. New records are added at the end of the active data, incrementing the count.

**6.2 File Handling and Persistence**  
Data persistence is achieved by storing records in plain text files within a data subdirectory.

* data/books.txt
* data/users.txt
* data/transactions.txt  
  Each line in these files typically represents one record, with fields separated by commas (CSV-like format).
* The FileHandler.java class encapsulates file reading and writing operations.
* Each model class (Book, User, Transaction) has toFileString() and static fromFileString(String line) methods to serialize and deserialize objects to/from their string representations for file storage.
* Manager classes call load...() methods in their constructors to populate arrays from files and save...() methods after any data modification.

**6.3 Input Validation**  
Input validation is implemented at multiple levels:

* **TermiLib.java (Main class):** Handles NumberFormatException for menu choices. Validates specific inputs during registration (e.g., age range, non-empty fields) and book addition (e.g., publication year, positive copy numbers).
* **InputValidator.java:** Provides reusable static methods for common validation tasks like email format, age range, year range, username/password policies, ISBN format, and checking for non-empty strings.
* **Manager Classes:** Perform specific business rule validations (e.g., BookManager checks for duplicate ISBNs, UserManager prevents deletion of the last admin, TransactionManager checks book availability and borrowing limits).  
  Validation messages guide the user to correct their input.

**6.4 Exception Handling**  
The system employs try-catch blocks to handle potential runtime exceptions:

* **Main Loop (TermiLib.java):** A general try-catch (Exception e) block wraps the main menu loop to catch any unhandled exceptions, print an error message, and allow the program to continue running instead of crashing. This meets the "software should not crash" requirement.
* **Numeric Input:** Integer.parseInt(scanner.nextLine()) is often wrapped in try-catch (NumberFormatException e) to handle cases where users enter non-numeric text for numeric fields.
* **File I/O (FileHandler.java):** Basic try-catch blocks are used around file operations, printing error messages to System.err if issues occur (e.g., IOException).  
  The goal is to gracefully handle common errors and provide informative feedback to the user.

**6.5 User Roles and Access Control**  
The system distinguishes between two user roles: "ADMIN" and "STUDENT".

* The User model has a role attribute.
* AuthManager authenticates users and determines their role.
* The TermiLib main class displays different menus (showAdminMenu(), showStudentMenu()) based on the currentUser.getRole().
* Admin users have privileged access to manage books, all users, all transactions, and view system-wide reports.
* Student users have restricted access to search/borrow/return books and view their own history.

**6.6 Menu System and Flow**  
The system is entirely menu-driven, providing a clear flow for users.

* TermiLib.java contains methods for each menu (showLoginMenu, showAdminMenu, showStudentMenu, showBookManagementMenu, etc.).
* Users make selections by entering numbers corresponding to menu options.
* switch statements are used to handle user choices and navigate to sub-menus or call appropriate operational methods.
* "Back" options in sub-menus allow users to return to the previous menu, ensuring proper navigation.
* Logout functionality returns the user to the main Login Menu.

**7. Tools and Technologies Used**

* **Programming Language:** Java (Version: JDK 8 or higher, as java.time.LocalDate is used)
* **Development Environment (IDE):** Any standard Java IDE such as:
  + Eclipse
  + IntelliJ IDEA
  + Visual Studio Code with Java extensions
* **Build Tool (Implied):** Standard Java compiler (javac) and runtime (java). No external build tools like Maven or Gradle are explicitly part of this simple CLI project structure.
* **Core Java Libraries:**
  + java.util.Scanner: For CLI input.
  + java.io.\* (BufferedReader, BufferedWriter, FileReader, FileWriter, File): For file handling.
  + java.nio.file.\* (Files, Paths): For directory creation and path management.
  + java.time.LocalDate, java.time.format.DateTimeFormatter: For handling dates in transactions.
  + java.util.regex.Pattern: Used in InputValidator for email validation.
  + java.util.Map (HashMap): Used in TransactionManager for generating reports (e.g., most borrowed books).
* **Operating System:** Platform-independent (Windows, macOS, Linux) due to Java's nature.
* **Version Control (Assumed):** Git (commonly used, though not explicitly stated in code).

**8. Project Timeline (Example)**

This is a sample timeline for a project of this nature. Adjust if your actual timeline differed.

* **Week 1: Planning and Design (e.g., [Start Date] - [End Date])**
  + Understanding requirements.
  + Designing data models (Book, User, Transaction).
  + Planning class structure (Managers, Utils, Auth).
  + Defining file storage format.
  + Outlining menu flows.
* **Week 2: Core Implementation - Models and Basic Managers (e.g., [Start Date] - [End Date])**
  + Implementing Book.java, User.java, Transaction.java models with toFileString/fromFileString.
  + Implementing FileHandler.java.
  + Basic BookManager and UserManager (add, view, array storage, initial file load/save).
* **Week 3: Advanced Manager Features and Authentication (e.g., [Start Date] - [End Date])**
  + Implementing AuthManager.java and login/registration logic in TermiLib.java.
  + Implementing update, remove, search functions in BookManager and UserManager.
  + Developing TransactionManager for borrow/return logic.
  + Integrating InputValidator.java.
* **Week 4: Menu System, Reporting, and Refinement (e.g., [Start Date] - [End Date])**
  + Building all CLI menus and ensuring proper navigation in TermiLib.java.
  + Implementing reporting features in TransactionManager.
  + Comprehensive testing of all functionalities.
  + Implementing robust exception handling.
  + Code cleanup and commenting.
* **Week 5: Documentation and Final Review (e.g., [Start Date] - [End Date])**
  + Writing the project report.
  + Capturing screenshots.
  + Final testing and bug fixes.
  + Preparing for submission.

**9. Code and Screenshots of Outputs**

The complete codebase has been provided separately. Key snippets and descriptions of expected outputs are below.

**9.1 Main Class Snippet (TermiLib.java)**  
The TermiLib.java class serves as the entry point and orchestrates the application flow. It initializes managers, handles user sessions, and displays menus.

// Snippet from TermiLib.java

public class TermiLib {

private static Scanner scanner = new Scanner(System.in);

private static BookManager bookManager = new BookManager();

private static UserManager userManager = new UserManager();

private static AuthManager authManager = new AuthManager(userManager);

private static TransactionManager transactionManager = new TransactionManager(bookManager, userManager);

private static User currentUser = null;

public static void main(String[] args) {

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

System.out.println(" Welcome to TermiLib Management System ");

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

initializeDefaultAdmin(); // Creates admin/admin if not present

while (true) {

try {

if (currentUser == null) {

showLoginMenu();

} else {

if (currentUser.getRole().equals("ADMIN")) {

showAdminMenu();

} else {

showStudentMenu();

}

}

} catch (Exception e) { // General catch to prevent crash

System.out.println("An error occurred: " + e.getMessage());

System.out.println("Please try again...");

}

}

}

// ... other menu methods like showLoginMenu(), showAdminMenu(), etc.

}

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Java

IGNORE\_WHEN\_COPYING\_END

**9.2 Login and Registration**

* **Screenshot 1: Welcome Screen & Login Menu**
  + Description: Shows the initial welcome message and the login menu with options: 1. Login, 2. Register as Student, 3. Exit.
  + Example Output:
  + ==========================================
  + Welcome to TermiLib Management System
  + ==========================================
  + Default admin created - Username: admin, Password: admin (if first run)
  + ========== LOGIN MENU ==========
  + 1. Login
  + 2. Register as Student
  + 3. Exit
  + Enter your choice:

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).

IGNORE\_WHEN\_COPYING\_END

* **Screenshot 2: Student Registration Process**
  + Description: Shows the prompts for student registration (username, password, name, age, email) and a success message.
  + Example Output:
  + ========== STUDENT REGISTRATION ==========
  + Username: newstudent
  + Password: password123
  + Full Name: New Student
  + Age: 20
  + Email: newstudent@example.com
  + Registration successful! You can now login.

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).

IGNORE\_WHEN\_COPYING\_END

* **Screenshot 3: Successful Admin Login**
  + Description: Shows the login prompt, admin credentials being entered, and the subsequent admin menu.
  + Example Output:
  + Username: admin
  + Password: admin
  + Login successful! Welcome Administrator
  + ========== ADMIN MENU ==========
  + 1. Book Management
  + 2. User Management
  + 3. Transaction Management
  + 4. Reports
  + 5. Logout
  + Enter your choice:

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).

IGNORE\_WHEN\_COPYING\_END

**9.3 Admin Menu and Operations**

\*\*9.3.1 Book Management\*\*

\* \*\*Screenshot 4: Book Management Menu\*\*

\* Description: Shows the options available under Book Management for an admin.

\* Example Output:

```

========== BOOK MANAGEMENT ==========

1. Add Book

2. View All Books

3. Update Book

4. Remove Book

5. Search Books

6. Back to Main Menu

Enter your choice:

```

\* \*\*Screenshot 5: Adding a New Book\*\*

\* Description: Shows the prompts for adding a book and the success message.

\* \*\*Screenshot 6: Viewing All Books\*\*

\* Description: Shows a list of books currently in the system.

\* \*\*Screenshot 7: Searching for a Book (e.g., by Title)\*\*

\* Description: Shows the search prompt and the results.

\*\*9.3.2 User Management\*\*

\* \*\*Screenshot 8: User Management Menu\*\*

\* Description: Shows options for managing users.

\* \*\*Screenshot 9: Viewing All Users\*\*

\* Description: List of registered users (admin and students).

\*\*9.3.3 Transaction Management & Reports\*\*

\* \*\*Screenshot 10: Transaction Management Menu\*\*

\* Description: Options for viewing transactions and overdue books.

\* \*\*Screenshot 11: Reports Menu\*\*

\* Description: Options for system reports.

\* \*\*Screenshot 12: System Statistics Report\*\*

\* Description: Shows total books, users, transactions, etc.

\* Example Output:

```

========== SYSTEM STATISTICS ==========

Total Books: 100

Available Books: 85

Borrowed Books: 15

Total Users: 5

Total Transactions: 20

Active Borrowings: 15

```

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).

IGNORE\_WHEN\_COPYING\_END

**9.4 Student Menu and Operations**

* **Screenshot 13: Student Menu**
  + Description: Shows the menu available to a logged-in student.
  + Example Output:
  + ========== STUDENT MENU ==========
  + 1. Search Books
  + 2. View Available Books
  + 3. Borrow Book
  + 4. Return Book
  + 5. My Borrowing History
  + 6. Logout
  + Enter your choice:

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).

IGNORE\_WHEN\_COPYING\_END

* **Screenshot 14: Student Borrowing a Book**
  + Description: Shows the student entering a book ID to borrow, and the success/failure message with due date.
* **Screenshot 15: Student Viewing Borrowing History**
  + Description: List of books borrowed by the student, with their status.

**9.5 Data Persistence (File Examples)**

* **Screenshot 16: Content of data/books.txt**
  + Description: A screenshot of a text editor showing a few lines from books.txt, illustrating the CSV format.
  + Example Content:
  + 1,The Great Gatsby,F. Scott Fitzgerald,9780743273565,1925,Classic,5,3
  + 2,To Kill a Mockingbird,Harper Lee,9780061120084,1960,Fiction,3,3

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).

IGNORE\_WHEN\_COPYING\_END

* **Screenshot 17: Content of data/users.txt**
  + Description: Example content from users.txt.
* **Screenshot 18: Content of data/transactions.txt**
  + Description: Example content from transactions.txt.

**9.6 Error Handling and Validation Examples**

* **Screenshot 19: Invalid Menu Choice**
  + Description: Shows the system responding to an invalid numeric choice in a menu.
  + Example Output: Invalid choice! Please try again.
* **Screenshot 20: Non-Numeric Input for Numeric Field**
  + Description: Shows the system handling text input where a number (e.g., age, book ID) was expected.
  + Example Output: Please enter a valid number! or Please enter a valid age!
* **Screenshot 21: Age Validation Error during Registration**
  + Description: Shows the error message when a user tries to register with an age outside the valid range (e.g., 15 or 101).
  + Example Output: Age must be between 16 and 100!

**10. Beyond Current Focus (Limitations)**

While TermiLib fulfills its core requirements, it has certain limitations inherent in its design and scope:

* **Fixed-Size Arrays:** The use of fixed-size arrays (e.g., MAX\_BOOKS = 1000) means the system has a hard limit on the number of records it can store. If this limit is reached, no new records can be added without modifying the code and recompiling.
* **No GUI:** The CLI interface, while efficient for some, may not be as user-friendly for all users compared to a graphical interface.
* **Basic Reporting:** Reporting capabilities are rudimentary. More advanced analytics or customizable reports are not available.
* **Single Admin Type:** Only one type of admin role exists. No finer-grained permissions for administrative tasks.
* **No Real-time Concurrency:** The system is designed for single-user CLI interaction at a time. It does not handle concurrent access from multiple users simultaneously, which would be an issue for a shared, networked library system. Data files could get corrupted if multiple instances try to write at the same time.
* **File-Based Storage Scalability:** Plain text files for data storage can become inefficient and slow with very large datasets. Searching can become linear and time-consuming.
* **Limited Search:** Search functionality is basic string matching (contains/equals). No advanced search features like boolean operators, relevance ranking, or fuzzy search.
* **No Fine System:** The system tracks overdue books but does not implement any fine calculation or management.
* **Security of Password Storage:** Passwords are stored in plain text in users.txt, which is a significant security vulnerability in a real-world scenario.

**11. Future Enhancements**

TermiLib can be extended with several features to improve its functionality, usability, and robustness:

* **Database Integration:** Replace flat-file storage with a relational database (e.g., MySQL, PostgreSQL, SQLite) for better scalability, data integrity, and querying capabilities.
* **Dynamic Data Structures:** Use ArrayList or other dynamic collections instead of fixed-size arrays to remove hard limits on record counts.
* **Graphical User Interface (GUI):** Develop a GUI using Java Swing or JavaFX for a more visually appealing and intuitive user experience.
* **Web-Based Interface:** Transform the application into a web application using frameworks like Spring Boot, allowing access from any web browser.
* **Advanced Search and Filtering:** Implement more sophisticated search options (e.g., search by range of publication years, combined criteria) and filtering.
* **Fine Management System:** Add functionality to calculate and manage fines for overdue books.
* **User Roles and Permissions:** Introduce more granular user roles (e.g., Librarian, Senior Librarian) with specific permissions.
* **Password Hashing:** Implement secure password storage using hashing algorithms (e.g., bcrypt, scrypt).
* **Book Reservation and Renewal:** Allow students to reserve books that are currently borrowed and renew their borrowed books online.
* **Import/Export Data:** Add functionality to import/export library data in standard formats like CSV or XML.
* **Audit Trails/Logging:** Implement logging for important actions performed by users for auditing and troubleshooting.
* **Enhanced Reporting:** Provide more detailed and customizable reports with options for graphical representation.
* **Notification System:** Send email or in-app notifications for due date reminders or when a reserved book becomes available.

**12. Conclusion**

In a nutshell, TermiLib successfully demonstrates the development of a Command Line Interface-based Library Management System using Java. It effectively meets the core project requirements, including a menu-driven interface, separate user roles (Admin and Student) with distinct functionalities, CRUD operations for books and users, transaction management, and persistent data storage using file I/O. The system incorporates input validation and exception handling to ensure stability and data integrity.

The use of arrays for data management and flat files for persistence, while simple, serves the purpose for a small-scale application. The modular design with separate manager, model, utility, and authentication packages promotes code organization and maintainability.

While there are limitations and areas for future enhancements, TermiLib provides a functional and foundational system for library operations, showcasing an understanding of core Java programming principles, object-oriented design, and application development for the command line.

**13. Reference Citations**