

COMP 4004 A/B – Fall 2025

Assignment 1 – Due October 17, 2025, at 23:59

Library Management System

About this Assignment

This assignment focuses on implementing a library system using a flavour of Test-Driven Development (TDD). You'll build a **text-based system** that demonstrates key software engineering principles in a manageable scope, following a responsibility-based TDD approach and create acceptance tests. This is a **4-week assignment** with the following deliverables, including derived responsibilities from use cases, unit and acceptance testing, GitHub submissions corresponding to test and code submissions, responsibility traceability, and a video demonstration.

This assignment is the first of three assignments. The **second and third assignments** will build off of the library system you developed in the first assignment.

Learning Objectives

By completing this assignment, you will:

- Apply skills to derive responsibilities from use cases
- Implement traceability across use cases and responsibilities
- Develop an understanding of the Test-Driven Development (TDD) methodology
- Implement responsibility-based software design using TDD methodology
- Develop skills in using a source control management system (GitHub).
- Create automated unit and acceptance test suites

Required Tools and Technologies

- **IDE:** IntelliJ IDEA Community Edition 2025.2
- **Java:** Oracle OpenJDK 22
- **Build Tool:** Maven (version that IntelliJ provides)
- **Testing Framework:** JUnit 5 (Jupiter), version 5.13.4
- **Version Control:** Git with GitHub

System Overview

Actors:

- **Borrowers:** Can log in to the library system, borrow a book, return a book, put a book on hold, and logout of the library system.

System Functionality:

- User authentication with username/password
- Book collection display during borrowing
- Single book borrowing with eligibility validation
- Single book return processing
- Single book hold placement (via borrow book workflow)
- Real-time book availability tracking
- Session management
- **Text-based system** (for assignments 1 and 2)

Borrowing Rules

- **Operations are single-item transactions: each borrow or return handles one book.** (to simplify the system)
- A maximum of three books can be borrowed per borrower
- Books have a 14-day borrowing period (use date strings in YYYY-MM-DD format, e.g., "2025-10-17")
- Only available books can be borrowed
- No renewals (to simplify the system)

Hold System Rules

- Hold queue operates on a first-come, first-served basis
- Only one hold per borrower per book is allowed
- Holds can only be placed on books that are currently checked out or already on hold by another borrower
- Books on hold cannot be borrowed by other borrowers
- If a borrower attempts to place a hold on a book they already have on hold, the system displays an appropriate message and returns to the book selection
- When displaying the collection, books with holds appear as "On Hold" to all users except the holder, who sees them as available and can borrow them normally
- Borrowers **cannot place a hold** on a book they already have checked out.
- Borrowers **cannot place a hold** on an **Available** book (they must borrow it directly).
- Borrowers at the 3-book borrowing limit **may still place holds**, but cannot borrow until below the limit.

- For this assignment, implement a simple queue structure (no complex priority handling required)

System initialization

The system starts with a predefined collection of **20** books (unique titles and authors) and **3** borrower accounts with usernames and passwords. Students will create their own test data.

Data Persistence:

- This system operates entirely in memory
 - No file or database persistence required
 - System state resets each time the application restarts
 - All books return to "Available" status, and borrower accounts reset to zero books on restart
-

Use Cases:

UC-01: User Login

ID: UC-01

Name: User Login

Subject: Library Management System

Actors: Borrower

Triggering Event: System initialization completes (with default books and users) or user logout

Description:

The borrower logs into the system using username/password, and the system authenticates and creates a session.

Preconditions:

- The system has been initialized with exactly 20 books in the collection
- The system has been initialized with exactly 3 borrower accounts with valid usernames and passwords
- All book statuses are initially set to Available
- All borrower accounts have zero borrowed books initially

Sequence:

1. The system prompts for the borrower's authentication credentials
2. The borrower enters their username and password
3. The system validates credentials.

4. The system sets the authenticated borrower as the current user (session established).
5. The system checks if any books held by the borrower have become available (i.e., returned by the previous borrower) and displays a notification with book details.
6. The system presents available operations: borrow a book, return a book, and logout

Extensions:

- **3a. Invalid credentials:**
 - 3a.1 The system displays an authentication error.
 - 3a.2 Return to step 1.

Post-condition: The borrower is authenticated and logged in with an active session.

Resulting Event: The borrower can begin operations with the library management system.

Constraints:

- The system must validate account credentials before granting access.
- The system must maintain the session until logout.

UC-02: Borrow Book

ID: UC-02

Name: Borrow Book

Description: A borrower borrows one available book from the collection

Subject: Library Management System

Actors: Borrower

Triggering event: Borrower is authenticated, and selects borrow book option from available functionality

Sequence:

1. The system displays the borrower's current book count (0-3)
2. The system displays all books in the collection with the following information:
 - 2.1 Book title and author
 - 2.2 Availability status: Available, Checked Out, or On Hold.
 - 2.3 If checked out, shows Due: [date]
3. The borrower selects one available book
4. The system presents the selected book and borrowing details for confirmation
5. The borrower confirms the borrowing transaction
6. System verifies book availability (book is not currently borrowed or on hold)
7. The system verifies eligibility to borrow a book (borrower currently has fewer than 3 books borrowed).
8. The system calculates due date (14 days from current date)
9. The system records borrowing transaction with borrower information
10. The system updates borrower account and book availability status
11. The system provides borrowing confirmation with due date
12. The borrower acknowledges completion

13. The system returns to available functionality

Extensions:

- 6.a. Selected book is unavailable:
 - 6a.1 If the book's status is **Checked Out** or **On Hold** (by another borrower), the system offers to place a hold.
 - 6a.2 If the borrower already has a hold on this book, display "*You already have a hold on this book*" and return to Step 2.
 - 6a.3 If the borrower currently has this book checked out, display "*You already have this book checked out*" and return to Step 2 (no hold allowed).
 - 6a.4 If the book is **Available** but the borrower is at the **3-book limit**, the system prevents borrowing but still allows a hold.
 - 6a.5 The system records the hold in the queue and confirms placement.
 - 6a.6 Return to Step 2 (display collection again).
- 7a. Borrower already has 3 books borrowed:
 - 7a.1 The system notifies borrower maximum borrowing limit reached
 - 7a.2 Return to available functionality

Post-condition: The selected book is **checked out** to the borrower with a due date.

Resulting event: Book's availability status is shown as "Checked Out" in future displays

Constraints:

- Only one book can be borrowed per transaction
- A maximum of 3 books can be borrowed simultaneously
- Books must be available (not already borrowed)
- Books have a 14-day borrowing period
- Borrower privacy must be maintained (no personal information of other borrowers displayed)

UC-03: Return Book

ID: UC-03

Name: Return Book

Description: A borrower returns a borrowed book

Subject: Library Management System

Actors: Borrower

Triggering event: Borrower is authenticated, and selects the return book option from available functionality

Sequence:

1. The system presents the borrower's currently borrowed books with due dates

2. The borrower selects a book to return
3. The system checks for pending holds on the book
4. The system updates the book's availability status to "Available" and removes the book from borrower's account
5. The system provides a return confirmation
6. The borrower acknowledges completion
7. The system returns to available functionality

Extension:

- 1a. Borrower has no books currently borrowed.
 - 1a.1 System notifies borrower that no books are currently borrowed
 - 1a.2 Return to available functionality
- 3a. A hold is present on the returned book
 - **3a.1** The system updates the book's status to **On Hold** and reserves it for the next borrower in the queue.
 - **3a.2** The system records that this borrower is the **current holder** and ensures they are **notified on next login** (UC-01 step 5).
 - **3a.3** The book is **not automatically checked out**; the holder must borrow it explicitly via **UC-02**.
 - **3a.4** The system removes the book from the original borrower's account.
 - **3a.5** Continue at Step 5 in the main sequence

Post-condition: Book is returned and availability updated to "Available" or "On Hold"

Resulting event: Book shows as Available in future displays

Constraints: Single item transaction

UC-04: User Logout

ID: UC-04

Name: User Logout

Description: Borrower logs out and system returns to authentication prompt

Subject: Library Management System

Actors: Borrower

Triggering event: Borrower is authenticated, and selects logout option from available functionality

Sequence:

1. The system confirms logout with the borrower
2. The system clears current user session
3. The system returns to user authentication (UC-01)

Post-condition: No user is logged in and the user session has been removed

Resulting event: System prompts for user authentication

TDD Implementation Requirements

Step 1: Identify Responsibilities

Your Task: Analyze the use cases and identify the system's responsibilities. Each responsibility should represent a distinct, testable behaviour that the system must provide.

First, from the use cases provided in the previous pages, identify the responsibilities of the library system. Please note that a step of a use case may involve **one or more** responsibilities. Also, a **single responsibility may be realized using one or more methods** of some **class(es)**.

Starting Responsibilities (Provided)

To help you get started, here are 3 responsibilities:

Responsibility ID	Responsibility description	UC step(s) relevant to this responsibility	Number of Tests
RESP-01	System book collection initialization	UC-01 preconditions	
RESP-02	System user accounts initialization	UC-01 preconditions	
RESP-03	User Authentication and credentials validation	UC-01 steps 1, 2 and 3	

Your task is to identify the remaining **15-20 responsibilities** (excluding the three responsibilities provided) by analyzing the remaining use case steps.

Step 2: TDD Commit Strategy (to GitHub)

The GitHub repository should have the following naming convention: A1-<yourLastName>-<yourFirstName>-<yourStudentNumber> must be set as a **private project**, and you must invite all six TAs and the instructor to your repository. The TAs' and the instructor's GitHub usernames are on BrightSpace in the Course Information section.

Initial Setup

Your first commit can be an "Initialize System" commit before starting the TDD cycle to set up your basic project structure, dependencies, and initial framework.

R-TEST JUnit naming convention and GitHub commit convention:

JUnit Method Naming Convention (in IntelliJ test directory)

- Format: RESP_XX_test_Y
- Where:
 - XX = responsibility number (zero-padded: 01, 02, 03...)
 - Y = test number for that responsibility (1, 2, 3...)
- Example: RESP_01_test_1

GitHub Commit Guidelines:

- **Commit Message:** R-TEST-XX
 - Where XX = responsibility number (zero-padded)
 - Example: R-TEST-01
- **Commit Description:** List all test methods included
 - Format: Each test method name on a separate line
 - Example:
RESP_01_test_1
RESP_01_test_2
RESP_01_test_3

Test Requirements:

- Tests must compile and call actual system methods (no mocks) to ensure full integration testing at each step

R-CODE commit: Implement code to make tests pass

- Commit message: "R-CODE-X"
- Commit description: Include the responsibility ID, X, that this code implements
 - Example: "R-CODE-03"
- Write minimal code to satisfy the tests

REFAC commit: Refactor code without changing the external behaviour.

- **Commit message:** "REFAC"
- Commit description: Briefly describe what was refactored

- Example: "REFAC - Extracted authentication logic into separate method"
- Refactor without changing external behaviour.

Step 3: Acceptance Testing

Note that Acceptance Tests must be committed to GitHub using the commit message "A-TEST-X". Where X refers to the acceptance test. The commit description should provide details about your acceptance test.

Write exactly two A-TEST commits covering these scenarios:

A-TEST-01: "Multi-User Borrow and Return with Availability Validated"

- **Path:** UC-01 → UC-02 → UC-04 → UC-01 → UC-02 → UC-04 → UC-01 → UC-03 → UC-04 → UC-01 → UC-02 → UC-04
- **Test:** User1 logs in, borrows "The Great Gatsby", logs out. User2 logs in, sees book checked out. User1 logs back in, returns book. User2 sees book available again.

A-TEST-02: "Initialization and Authentication with Error Handling"

- **Path:** UC-01 complete workflow with error handling
- **Test:** System starts with 20 books and 3 borrower accounts, valid login succeeds and displays menu options and logouts. Another user enters invalid credentials, rejected with error message and retry prompt.

Note: Only these two acceptance tests are required. Additional acceptance tests are optional but not required for full marks.

Delivery Requirements

Submit completed grading grid (will provide at a later date):

- Repository URL and video link
- A Table that lists all responsibilities with use case traceability and the number of tests for each responsibility
- Confirmation that all tests pass
- The grading grid that you fill up must be renamed to A1-<yourLastName>-<yourFirstName>-<yourStudentNumber> and must be submitted to Brightspace by the due date

Video Demonstration Requirements (No more than 20 minutes)

1. **Installation Demo:** Clone from GitHub, show compilation success
 2. **Testing Demo:** Run all unit tests, run acceptance tests (all green)
 3. **System Demo:** This will be you starting the library system and demonstrating the following: user authentication (log in), borrow one book, return one book, logout. Log in with the same user and select Borrow Book, and show that the book that was originally borrowed indicates that it is checked out, then logout of the system
- Repository URL and video link
 - Record using Zoom (recommended)
 - Upload completed video to YouTube (unlisted), OneDrive, or Google Drive
 - Include a shareable link in your grading grid
-

Timeline and Milestones (meant as a guide)

Week 1: Foundation

- Install IntelliJ and setup a Maven project using JUnit 5 (Jupiter)
- Review the assignment and derive responsibilities from the use cases. Remember all use cases and their sequences and preconditions/postconditions must be covered by all the responsibilities.
- Set up a GitHub account (if you don't have one) and create a private project with the specified naming convention and invite all the TAs and the instructor to your repository.

Week 2: Core Functionality

- Review your responsibilities.
- Work on implementing about 50% of your responsibilities using the test-driven development cycle: test-code-refactor(as needed).

Week 3: Integration

- Complete remaining responsibilities and acceptance testing scenarios

Week 4: Documentation and Delivery

- Video creation, documentation, and final testing
- Complete the grading grid and submit the assignment to BrightSpace