Library System Manager

By Alex Huang

Data Structures and Object-Oriented Programming
Yi Wang

Table of Contents

Project Description

Taken from Deliverable 1

1. Scenario

The Vanier College Library needs a simple system that can manage its book catalog and track student's book exchanges. The system needs to allow students to search and borrow books, and let librarians add, issue, and return them.

2. Design Paradigm

Functionalities that will be demonstrated:

- Class hierarchies and OOP
- Interfaces
- Runtime polymorphism (overriding and overloading)
- Java collections: `List`, `Queue`
- File I/O with CSV files
- `Comparable` and `Comparator` for sorting
- Test-Driven Development with JUnit tests
- Git version control

3. Expected Output

- **Librarian**
  - Add/remove books from the library
  - Issue books to a student
  - Process returns (i.e. make the book available for borrowing again).
- **Students**
  - Search the catalog using a keyword
  - Borrow books (if available)
  - Return previously borrowed books
- Changes should be reflected in the appropriate CSV files

4. Class Hierarchies

- **Hierarchy 1:** `User` (abstract) → `Student`, `Librarian`
- **Hierarchy 2:** `Book` (abstract) → `NormalBook`, `ReferenceBook`

5. Interface

`Issuable` interface:

- Contract to keep track of the student who returns the book and the student who is borrowing the book
- Implemented by `NormalBook`, but **not** by `ReferenceBook` (e.g. dictionary).
    - Future-proof way add a new loanable resource type (DVD, equipment, board game, etc.) by implementing `Issuable` to inherit checkout/return behavior without touching existing class hierarchy

6. Runtime Polymorphism

- **Overriding**
    - Common `displayBook()` in the abstract `Book` class
    -

Book	**Display info**
Normal	Book issue status, expected return date
Reference	Shelf location, specify book is not issuable

- **Overloading**
    - In `LibrarySystem`, we provide two ways to lend:

public boolean issueBook(NormalBook book, Student s)

public boolean issueBook(String isbn, String studentId)

7. Text I/O

- `addNewBooksToLibrary(String path)` in `Librarian` that reads a CSV file and adds all books to a the `LibrarySystem`'s list of books
    - Provides an efficient alternative to add books to the library. Only accessible by `Librarian`s
- `exportBooks()` in `LibrarySystem` that exports all current books into a CSV file
    - Allows easy overview of current books. Librarians could also use this to quickly delete a book by removing a line from the CSV, then running the above method to update the list.

8. Comparable and Comparator

public class NormalBook implements Comparable<NormalBook>

- This will compare books based on their due date, useful for students

public static class BookComparator implements Comparator<Book>

- This will compare books based on either title, pages, author, or isbn (default)

9. UML Class Diagram

- Navigate to `doc/uml_class_diagram.pdf`

10. Deliverable 2

For Deliverable 2:

- All class and interface definitions
- Documentation for methods
- JUnit tests
- Partial implementations:
    - Basic CSV read/write
    - Book display

11. Deliverable 3

Complete implementation of all methods

12. Deliverable 4

Project report

- Navigate to `doc/project_report.pdf`

Project Demo

First, we instantiate a Librarian and a Student (subclasses of User)

```
// Instantiate a Librarian and a Student
Librarian librarian = new Librarian( name: "Alice", Gender.FEMALE);
librarians.add(librarian);
Student student = new Student( name: "Bob", Gender.MALE);
students.add(student);
```

Then,

```
// Import books from CSV
System.out.println("=== Importing books from CSV ===");
librarian.addNewBooksToLibrary( path: "src/main/resources/new_books.csv");
```

(file input)

The target CSV:
7 columns: NormalBook (title,author,isbn,pages,status,borrower,dueDate)
6 columns: ReferenceBook (title,author,isbn,pages,shelfLocation,availableCopies)

```
1  Intro to java,John,123-456,400,,,
2  Data Structures,Yi Wang,456-123,250,,,
3  Programming encyclopedia,Arthur,0123,600,A15,7
4  Database,Ben,789-1,150,,,
5
```

Printing the books currently in library into the console:

```
=== Current Catalog ===
Intro to java by John
    AVAILABLE
    Pages: 400
Data Structures by Yi Wang
    AVAILABLE
    Pages: 250
Programming encyclopedia by Arthur:
    Shelf location: A15
    Total copies: 7
    Pages: 600
    NOTE: Book is not issuable, put it back in place after use
Database by Ben
    AVAILABLE
    Pages: 150
```

books.csv is also updated (file output):

```
1   Intro to java,John,123-456,400,AVAILABLE,,
2   Data Structures,Yi Wang,456-123,250,AVAILABLE,,
3   Programming encyclopedia,Arthur,0123,600,A15,7
4   Database,Ben,789-1,150,AVAILABLE,,
5
```

Borrowing

```
// Student borrows book by ISBN
System.out.println("\n=== Bob borrows ISBN 123-456 ===");
issueBook( isbn: "123-456", student.getId());

// Student borrows book by NormalBook object
System.out.println("\n=== Bob borrows Data Structures ===");
issueBook((NormalBook) books.get(1), student);
```

```
// Borrowing a ReferenceBook yields an error
System.out.println("\n=== Bob borrows ReferenceBook: ISBN 0123 ===");
issueBook( isbn: "0123", student.getId());
```

```
=== Bob borrows ReferenceBook: ISBN 0123 ===
Exception in thread "main" java.lang.IllegalArgumentException Create breakpoint : cannot borrow anything other than a NormalBook
    at LibrarySystem.issueBook(LibrarySystem.java:56)
    at LibrarySystem.main(LibrarySystem.java:218)
```

Printing results in console:

```
=== Bob's Borrowed Books ===
Intro to java by John
    UNAVAILABLE. Expected return: 06/08/2025
    Pages: 400
Data Structures by Yi Wang
    UNAVAILABLE. Expected return: 06/08/2025
    Pages: 250
```

```
=== Catalog After Borrow ===
Intro to java by John
    UNAVAILABLE. Expected return: 06/08/2025
    Pages: 400
Data Structures by Yi Wang
    UNAVAILABLE. Expected return: 06/08/2025
    Pages: 250
Programming encyclopedia by Arthur:
    Shelf location: A15
    Total copies: 7
    Pages: 600
    NOTE: Book is not issuable, put it back in place after use
Database by Ben
    AVAILABLE
    Pages: 150
```

Updated books.csv:

```
1   Intro to java,John,123-456,400,BORROWED,Bob,2025-06-08
2   Data Structures,Yi Wang,456-123,250,BORROWED,Bob,2025-06-08
3   Programming encyclopedia,Arthur,0123,600,A15,7
4   Database,Ben,789-1,150,AVAILABLE,,
5
```

```
// Bob cannot borrow same book twice
System.out.println("\n=== Cannot borrow same book twice ===");
System.out.println("Successfully borrowed?: " + issueBook((NormalBook) books.get(1), student));
```

```
=== Cannot borrow same book twice ===
Successfully borrowed?: false
```

Returning

```
// Student returns Intro to Java book
System.out.println("\n=== Bob returns ISBN 123-456 ===");
boolean returned = student.returnBook(student.getBorrowedBooks().getFirst());
System.out.println("Return successful? " + returned);
```

```
=== Bob returns ISBN 123-456 ===
Return successful? true
```

Print results in console:

```
=== Bob's Borrowed Books ===
Data Structures by Yi Wang
    UNAVAILABLE. Expected return: 06/08/2025
    Pages: 250

=== Catalog After Borrow ===
Intro to java by John
    PROCESSING. Available soon.
    Pages: 400
Data Structures by Yi Wang
    UNAVAILABLE. Expected return: 06/08/2025
    Pages: 250
Programming encyclopedia by Arthur:
    Shelf location: A15
    Total copies: 7
    Pages: 600
    NOTE: Book is not issuable, put it back in place after use
Database by Ben
    AVAILABLE
    Pages: 150
```

Updated books.csv:

```
1  Intro to java,John,123-456,400,PROCESSING,,
2  Data Structures,Yi Wang,456-123,250,BORROWED,Bob,2025-06-08
3  Programming encyclopedia,Arthur,0123,600,A15,7
4  Database,Ben,789-1,150,AVAILABLE,,
5
```

```
// Alice, librarian, processes the return
System.out.println("\n=== Alice processes return ===");
librarian.processReturn();
```

Result: Intro to Java is now Available

```
Intro to java,John,123-456,400,AVAILABLE,,
Data Structures,Yi Wang,456-123,250,BORROWED,Bob,2025-06-08
Programming encyclopedia,Arthur,0123,600,A15,7
Database,Ben,789-1,150,AVAILABLE,,
```

Search catalog

```
// Search catalog by keyword ("data")
System.out.println("\n=== Search by keyword "data" ===");
List<Book> results = searchBooks( keyword: "data");
results.forEach(Book::displayBook);
```

```
=== Search by keyword "data" ===
Data Structures by Yi Wang
    UNAVAILABLE. Expected return: 06/08/2025
    Pages: 250
Database by Ben
    AVAILABLE
    Pages: 150
```

Sorting

With BookComparator

```
// Sort catalog of books by title
System.out.println("\n=== Sort by Title ===");
books.sort(new Book.BookComparator( criteria: "title"));
books.forEach(Book::displayBook);
```

```
=== Sort by Title ===
Data Structures by Yi Wang
    UNAVAILABLE. Expected return: 06/08/2025
    Pages: 250
Database by Ben
    AVAILABLE
    Pages: 150
Intro to java by John
    AVAILABLE
    Pages: 400
Programming encyclopedia by Arthur:
    Shelf location: A15
    Total copies: 7
    Pages: 600
    NOTE: Book is not issuable, put it back in place after use
```

With NormalBook Comparable

Bob's books before he returned any

```
// Sort Bob's books by due date
System.out.println("\n=== Sort by Due Date ===");
// Set first borrowed book's due date to a later date than the second
student.getBorrowedBooks().getFirst().setDueDate(LocalDate.now().plusMonths( monthsToAdd: 2));
student.getLoansByDueDate().forEach(Book::displayBook);
```

Print result in console:

```
=== Sort by Due Date ===
Data Structures by Yi Wang
    UNAVAILABLE. Expected return: 06/08/2025
    Pages: 250
Intro to java by John
    UNAVAILABLE. Expected return: 07/11/2025
    Pages: 400
```

Interface method example (calculateLateFee)

```
// Calculate late fee
System.out.println("\n=== Calculate late fee ===");
NormalBook bookWithLateFee = student.getBorrowedBooks().getFirst();
// Set imaginary overdue date
bookWithLateFee.setDueDate(LocalDate.now().minusDays( daysToSubtract: 5));
System.out.printf("5 day late fee: %.2f$", bookWithLateFee.calculateLateFee());
```

```
=== Calculate late fee ===
5 day late fee: 2.50$
```

Challenges

No Console UI
I could have connected the backend logic for searching, borrowing, and returning books with a console-based menu. All operations run only via a method call instead of through user interaction.

User Authentication
I could have implemented sign-up and login checks to verify the credentials of the User subclasses.

Static Fields & JUnit Setup
It was initially difficult to test the LibrarySystem methods due to its static fields, since it broke isolation between tests. I worked around it by resetting all statics using a method with an `@BeforeEach` annotation.

Learning Outcomes

Test-Driven Development
I learnt the usefulness of writing tests and thinking about edge cases before fully implementing all methods. It forced me to think more deeply about the relation between different methods (such as between a `Student`'s `returnBook()` and a `Librarian`'s `processReturn()`). It also made verifying the correctness of my code a lot easier.

Git version control system
I realized the convenience of using the Git version control system. It allowed me to try different method implementations without worrying about irreversibly breaking the project. This also forced me to commit early and often, with clear messages. Using a VCS also made it easier to work on the project at school and at home without using external hardware.

Deepened OOP understanding
Designing the two class hierarchies reinforced principles of inheritance, encapsulation, and abstract classes. Having to think about the structure of the project on my own reinforced these principles further.