420-SF2-RE Data Structures and Object-Oriented Programming

Library Management System

Sebastian Bobos

May 11, 2025

Table of Contents

Project Description

This small-scale Java project aims to replicate the most important features that a real library system could. Members can borrow books, and return them, and will have to pay a fee of 5 cents per day past the expected return date, and librarians will process their returns, and can manage users with late fees. Users can view all publications, filter by type, and can search for publications based on name (and author, for more precise searches).

Data structures

The library system class stores all different types of users and different types of publications in an ArrayList, since it is a resizable collection. However, for the member's borrowed books, it is stored in a HashMap, with each book having its corresponding due date. I chose not to have the due date of a book as a field in the class because there can be more available copies of a single book and each copy could have different due dates.

Other Requirements

The project contains two class hierarchies: The user class, with child classes BasicMember, StudentMember, and Librarian, as well as the Publication class, with child classes Novel, ReferenceBook, and Magazine. The interface for this project Borrowable, with methods Borrow and CalculateFees, which is implemented by the BasicMember and StudentMember, since they are the only users that should be able to borrow books. The runtime polymorphism is applied in this interface, since the implementing classes override the interfaces' methods to implement them differently (e.g. since basic members cannot borrow reference books). The text IO is implemented as a simple database to store the library system classes' list of publications (novels, reference books, magazines stored separately) and users (basic, student, and librarian stored separately). On runtime, all information is read from the csv files to configure each ArrayList of the library system class with the current data. The comparable class is implemented for the member classes to sort them by their fees when the librarian would view users having fees. The comparator class is used in the publication class to sort by title (also default), author, or year. The main use of the comparator is when viewing all publications.

Program Features and Screenshots

The main features of the project are searching for books and borrowing books.

Here are test cases for the search method:

```java
@Test  firedespire *
public void testSearch_Title() {
    User user = new BasicMember( email: "email",  name: "john",  password: "123");

    List<Publication> expected = new ArrayList<>();
    expected.add(LibrarySystem.referenceBooks.get(1));

    List<Publication> result = user.search( title: "The story of water");

    Assertions.assertEquals(expected, result);

    System.out.println(result);
}
```

```java
@Test  firedespire *
public void testSearch_TitleAndAuthor_Wrong() {
    User user = new BasicMember( email: "email",  name: "john",  password: "123");

    List<Publication> expected = new ArrayList<>();

    List<Publication> result = user.search( title: "The story of water",  author: "");

    Assertions.assertEquals(expected, result);

    System.out.println(result);
}
```

```java
@Test  firedespire *
public void testSearch_TitleAndAuthor_Right() {
    User user = new BasicMember( email: "email",  name: "john",  password: "123");

    List<Publication> expected = new ArrayList<>();
    expected.add(LibrarySystem.referenceBooks.get(1));

    List<Publication> result = user.search( title: "The story of water",  author: "John");

    Assertions.assertEquals(expected, result);

    System.out.println(result);
}
```

In the first case, we are searching simply by title, and the method returns a single book with title “The story of water” as we searched for. In the second case, the title is the same but the author

parameter has the wrong author name, so since there is no book with that title, it returns an empty list. In the third case, the author is correct, so we get the same book from the first case. Here is the output of those test cases:

```
C:\Users\seb\.jdks\openjdk-23.0.2\bin\java.exe ...
[ReferenceBook{year=1998, title='The story of water', author='John', availableCopies=3}]
[]
[ReferenceBook{year=1998, title='The story of water', author='John', availableCopies=3}]

Process finished with exit code 0
```

Moving on, here are test cases for the borrow method:

```
@Test  Sebastian Bobos +1 *
public void testBorrow_Magazine () {
    StudentMember student = new StudentMember( email: "email",  name: "Johnny",  password: "pass",  studentID: 123);
    Magazine magazine = new Magazine( year: 1997,  title: "Magazine",  author: "Johnny",  availableCopies: 2,  contentSummary: "Summary");

    Librarian librarian = new Librarian( email: "email",  name: "Ben",  password: "123");
    librarian.addPublication(magazine);

    Assertions.assertFalse(student.borrow(magazine));

    librarian.removePublication(magazine);
}
```

```
@Test  Sebastian Bobos +1 *
public void testBorrow_Novel () {
    StudentMember student = new StudentMember( email: "email",  name: "Johnny",  password: "pass",  studentID: 123);
    Novel novel = new Novel( year: 1997,  title: "Magazine",  author: "Johnny",  availableCopies: 2);

    Librarian librarian = new Librarian( email: "email",  name: "Ben",  password: "123");
    librarian.addPublication(novel);

    Assertions.assertTrue(student.borrow(novel));

    librarian.removePublication(novel);
}
```

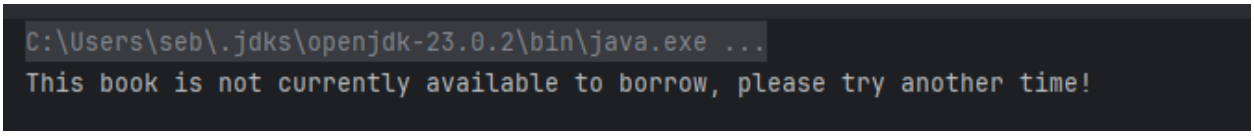
```
@Test  Sebastian Bobos +1 *
public void testBorrow_ReferenceBook () {
    StudentMember student = new StudentMember( email: "email",  name: "Johnny",  password: "pass",  studentID: 123);
    ReferenceBook referenceBook = new ReferenceBook( year: 1997,  title: "Magazine",  author: "Johnny",  availableCopies: 2);

    student.borrow(referenceBook);

    Assertions.assertTrue(student.borrow(LibrarySystem.referenceBooks.getFirst()));
}
```

In the first case, the member is trying to borrow a magazine. Because of the library's policy that does not allow anyone to borrow a magazine, the method returns false, the magazine is not

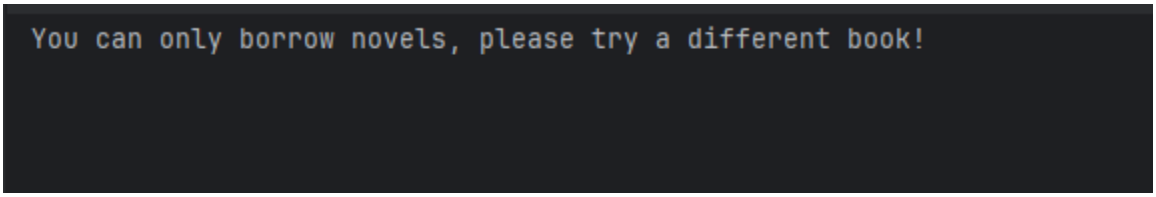
borrowed, and a message is displayed showing the reason why the publication could not be borrowed:

In the other test cases, the member is trying to borrow a novel in one and a reference book in the other. Since the member is part of the StudentMember class, allowing him to borrow reference books, both methods return true indicating its success in borrowing the book.

Other cases accounted for are when the publication's number of available copies is 0, since it means that there is no book for the member to borrow at this time. Here is the displayed message:



Finally, since a basic member cannot borrow reference books, the method returns false and displays the following message:



Challenges

I was not able to implement the payFees method as it would require a third party to work properly. I also struggled with my unit tests because of the project's inconsistency. First, my main issue was that the books' available copies field would change after one unit test when testing the borrow method, which led to the first test passing but it failing after being used multiple times. The same happened to the majority of my test cases, even though they worked on the first try. However, since it did not affect the project itself, I did not change the way that they worked.

Learning Outcomes

This project allowed me to learn most about git and github (using it through git bash), as well as project structure. Learning to use git bash was beneficial because it is general (can be used with any IDE), and is easy to implement in other projects. I also learned how to use the LocalDateTime java class which allowed me to manage the fee amount when returning a book late.