Library Management System

Overview

Objective

You will be creating a Library Management System in which you can perform all CRUD operations, in addition to advanced search, book issuing, Serialization, and Deserialization.

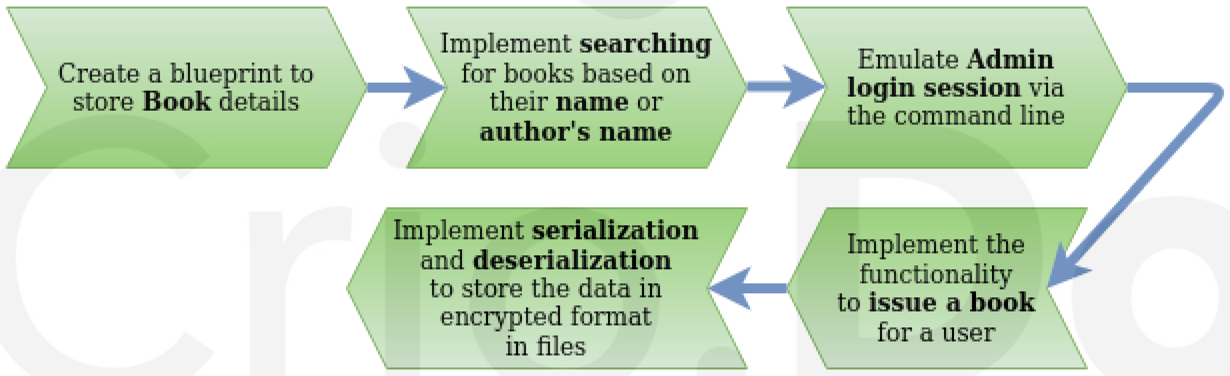
Project Context

Management systems mostly use software across all schools and colleges to keep record of data. One such system is the Library Management System, which is used to keep records of books present in a library.

These Systems comprise of information regarding Books, where we can perform all the operations that are required in an actual system such as CRUD operation, advanced search, book issuing, and also to store the information within files using the process known as Serialization.

Project Stages

The project consists of the following stages:


Create a blueprint to store **Book** details
Implement **searching** for books based on their **name** or **author's name**
Emulate **Admin login session** via the command line
Implement the functionality to **issue a book** for a user
Implement **serialization** and **deserialization** to store the data in encrypted format in files

High-Level Approach

- The project will be a typical command line application in Java.
- We'll be accepting data (book and user details) to be stored.
- We'll be adding the functionality to retrieve book details.
- The admin login session will be emulated via the command line.
- Only during the admin login session can a user be issued a book.
- All of the data will be stored in encrypted format in text files to escape the initial volatile nature of the application.

Task 1

Getting started

Our project is going to be a command line application in Java. Like most typical command line applications, it'll contain a switch case that will display a set of options to the user to perform the actions and make necessary queries.

We'll start off by creating an object for storing details of books and also adding some code to our `public static void main(String[] args)`.

Requirements

- Create a class Book, for addition of book details.
- Add required fields such as `bookId`, `bookName`, `writerName`, `price` and `quantity`.
- In your main code, make use of Java Collections to efficiently store the accepted book details.

References

- Classes and Objects in Java
- Collections in Java
- ArrayList in Java

Expected Outcome

You should be able to store the details of various books in the form of objects of type `Book`, while using an efficient data structure at the same time.

Task 2

Addition and Deletion of book details

Next up we'll be adding the functionality to add/delete book details.

Requirements

- You should be able to accept details for a specific book and store the same in an object of type `Book`.
- Every time you add the details of a new book in your already present collection (preferably ArrayList), generate a unique `bookId` for the same.
- Accept a query for deleting book details and perform the same. You should check whether the book details requested to be deleted are present or not.
- Note that you have an attribute called `quantity` for each object of type `Book`. This attribute can come in handy when you want to reduce the quantity of a specific book.

Moreover, you should be able to check whether the number to be subtracted is actually lesser or equal to the already present `quantity` or not.

Bring it On!

- Can you raise red flags in the following cases:
  - Books details being absent.
  - In case of reducing the `quantity`, if the number to be subtracted is greater than the `quantity`.

Expected Outcome

You should be able to add and delete book details.

Task 3

Searching

We should be able to search for a particular book based on its name or the writer's name.

Requirements

- Search for book details by `bookName`. If found, display the details or display `Book Absent`.
- Search for book details by `writerName`. If found, display the details or display `Writer Absent`.
- Add the ability to search for a subsequence of `bookName` or `writerName`.
- The search functionality should not be case sensitive.

References

- Searching characters and substring in a String in Java

Expected Outcome

You should be able to implement an efficient searching functionality for books.

Task 4

Issue Book

In this task, we'll be implementing the functionality for users to issue a book. Only admins will be allowed to issue books to users.

Requirements

- In order to issue a book, we'll need a user. For the same, create a `User` entity with fields `userId` (should be auto-generated), `password`, `role` (`Admin`/`User`).
- Create a class `IssueBook` having attributes `bookId` and `userId`. The objects of this class will basically be created whenever a user issues a book.
- You should accept various user details from the command line.
- In order to issue a book for a `User`, emulate a login session for `Admin` in the command line. On issuing, add the details as an object of type `IssueBook` to an `ArrayList` of the same type.

Bring it On!

- Can you raise red flags in the following cases:
- If a book is already issued to a user then it should not be reissued.
- If a book is unavailable currently.

Expected Outcome

You should be able to emulate `Admin` login session from the command line for issuing books to a `User` as per request.

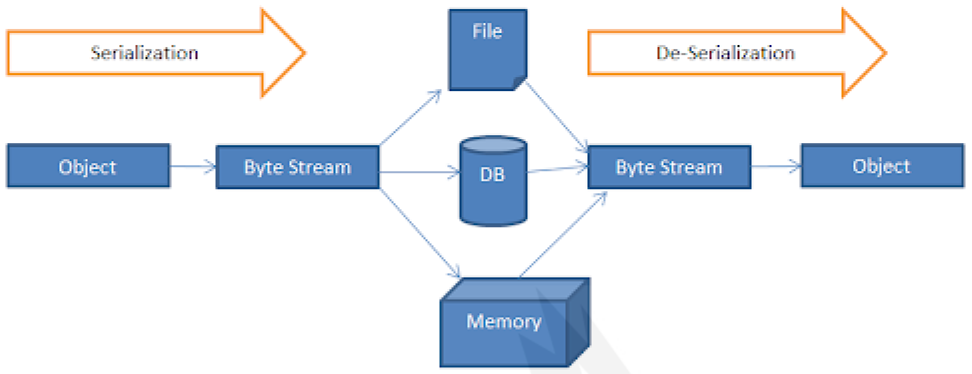
Task 5

Serialization and Deserialization using Files

Till now your main program should run fine. It should be able to add and remove book details. Add users and admins and enable admin login sessions to issue books for users. But if you stop your program execution, all the data gets deleted. That means that we'll need a way to store our data so that we can access and manipulate it at a later time.

Requirements

- Create three files: `userDetails.txt`, `bookDetails.txt` and `issueBook.txt`.
- All the data should be stored in these files in an encrypted format.
- Utilize the concept of serialization to store the data to the files.
- Utilize the concept of deserialization to access the data from the files.



References

- Serialization and Deserialization in Java with Example

Expected Outcome

The main objective of this milestone is to replace the volatile concept and store everything permanently in unreadable files.