Library Inventory System Documentation

1. High-Level Overview

This Java program is a console-based application that manages a small library. Think of it as a digital bookshelf.

How it works internally:

Instead of a database, this system uses a fundamental Java structure called an Array to store data.

1. We define what a "Book" looks like using a **Class** (a blueprint).
2. We create another **Class** (the system controller) that manages an array of these Book objects.
3. The system uses a menu loop to let the user choose actions (like adding, searching, or sorting books).

Key Concepts for Beginners:

- **Objects:** Each book is an individual "object" containing its own title, author, price, and ID.
- **The Inventory Array (`Book[] inventory`):** Imagine a long shelf with 100 fixed slots, numbered 0 to 99. Each slot can hold one Book object.
- **The `bookCount` variable:** Since the array has 100 slots but might only hold 5 books, we need a variable to keep track of how many slots are actually filled.

2. Class Breakdown: `Book`

Purpose: This class acts as a template or blueprint. It defines the data that *every* single book in the library must have. It doesn't manage the library; it just represents one book.

Attributes (Fields)

These are private variables that store information about the book. They are `private` so other parts of the code cannot mess with them directly (Encapsulation).

- `int id`: A unique number for the book (e.g., 101, 102).
- `String title`: The name of the book.
- `String author`: Who wrote the book.
- `double price`: How much it costs (allows decimal points).
- `boolean isAvailable`: A true/false flag. `true` means it's in the library; `false` means it's checked out.

Constructor method: `public Book(...)`

This is a special method called exactly once when a "new" book is created. It takes the initial information (ID, title, author, price) and fills in the attributes above. It automatically sets `isAvailable` to `true`.

Getters and Setters

Since the attributes are private, we need public "doors" to access them.

- **Getters (e.g., `getId`, `getTitle`):** Allow other parts of the program to *read* the data.
- **Setters (e.g., `setAvailable`):** Allow other parts of the program to *change* the data safely. We only have a setter for availability because the ID, title, etc., shouldn't change once created.

Method: `toString()`

When Java needs to print an object as text (like in `System.out.println`), it looks for this method. We customize it to return a neatly formatted string containing all the book's details grouped together with pipes (`|`).

3. Class Breakdown: `LibraryInventorySystem` (The Controller)

Purpose: This is the main engine of the program. It holds the array of books and contains all the logic for the menu options.

Key Fields (Variables)

- `private Book[] inventory`: This is the core data structure—an array that can hold `Book` objects.
- `private int bookCount`: Tracks how many books are currently stored. It starts at 0.
- `private final int MAX_CAPACITY = 100`: A constant defining the maximum size of the array.
- `private static int nextId = 101`: This variable belongs to the *library system itself*, not any specific book. It keeps track of the next available ID number to ensure every book gets a unique one.
- `Scanner scanner`: The tool used to read input typed by the user on the keyboard.

Constructor: `public LibraryInventorySystem()`

Sets up the empty library. It initializes the `inventory` array with 100 empty slots and ensures `bookCount` is 0.

4. Detailed Function (Method) Explanations

The Core Engine: `runMenu()`

This method runs a continuous loop (`do-while`) that keeps the program alive.

1. It prints the 16 menu options.
2. It asks for a number input.
3. It uses a `while (!scanner.hasNextInt())` loop to ensure the user actually typed a number, preventing crashes if they type letters.

4. It uses a `switch` statement. Think of this as a railway switch that redirects the program flow to different methods based on the number entered (e.g., if input is '1', it calls `addBooks()`).
5. The loop stops only when option 16 is chosen.

   Beginner Note on Scanner Issues:

   You will see `scanner.nextLine()` frequently after `scanner.nextInt()` or `scanner.nextDouble()`. When you type a number and press Enter, Java reads the number but leaves the "Enter" keypress (newline) in the buffer. The subsequent `nextLine()` consumes that leftover "Enter" so it doesn't mess up the next text input.

Feature 1: `addBooks()`

Allows adding one or more books.

1. Asks how many books to add.
2. Checks if there is enough room in the array (`MAX_CAPACITY`).
3. Runs a `for` loop for the number of books specified.
4. **Input Validation:** Inside the loop, it asks for Title, Author, and Price. It uses `do-while` loops to refuse empty titles/authors or negative prices, forcing the user to try again until valid data is entered.
5. **Creation:** Once data is valid, it takes the current `nextId` (e.g., 101), increments it for the next time (to 102), and creates a new book object: `inventory[bookCount] = new Book(...)`.
6. It puts the new book into the next available slot pointed to by `bookCount` and then increases `bookCount` by 1.

Feature 2: `displayAllBooks()`

1. Checks if `bookCount` is 0. If so, tells the user it's empty.
2. Uses a `for` loop starting from index 0 up to (but not including) the current `bookCount`.
3. At every step, it calls `System.out.println(inventory[i])`, which automatically triggers the `toString()` method of the book at that slot, printing its details nicely.

Features 3 & 4: `searchByTitle()` and `searchByAuthor()`

These work almost identically.

1. Ask the user for a search term.
2. Convert the search term to lowercase to make searching easier.
3. Loop through every book in the inventory.
4. Get the book's title (or author), convert it to lowercase, and check if it `.contains()` the search term. This allows partial matches (e.g., searching "harry" finds "Harry Potter").
5. If a match is found, print the book and set a `found` flag to true.

CRITICAL HELPER FUNCTION: `findIndexById(int id)`

This is a private helper used by many other functions. The user knows Book IDs (like 105), but the computer only knows array indices (like slot 4).

1. This function takes an ID as input.
2. It loops through the inventory looking for a book whose `.getId()` matches the input.
3. If found, it returns the **array index (`i`)** where that book lives.
4. If the loop finishes without finding it, it returns **-1** (an error code indicating "not found").

Features 5 & 6: `issueBook()` and `returnBook()`

1. Ask the user for the Book ID.
2. Call the helper `findIndexById(id)` to find where that book is located in the array.
3. If index is -1, say "Book not found".
4. If found, check its current status using `.isAvailable()`.
   - For issuing: if it IS available, use `.setAvailable(false)`.
   - For returning: if it IS NOT available, use `.setAvailable(true)`.

Feature 7: `removeBook()` (Important Logic)

Removing items from an array is tricky because arrays don't automatically shrink.

1. Ask for ID and find its index using the helper function. Let's say we find the book to remove at index [2].
2. We need to fill the gap at [2]. We do this by shifting everything after it one step to the left.
3. A loop starts at the removal index [2]. It copies the book from [3] into [2], then [4] into [3], etc.
4. After shifting, the last used slot holds a duplicate. We set it to `null` to clear it out.
5. We decrease `bookCount` by 1 because we have one less book.

Features 8 & 9: Sorting (`sortByPrice`, `sortByTitle`)

These implementations use a classic beginner algorithm called **Bubble Sort**.

- It uses nested loops (a loop inside a loop).
- The inner loop compares neighboring books (e.g., book at `[j]` vs book at `[j+1]`).
- If they are in the wrong order (e.g., for price sorting, if the left price is greater than the right price), it **swaps** them using a temporary variable.
- It repeats this process over and over until the whole list is sorted.
- *Note:* Title sorting uses `.compareToIgnoreCase()`, which returns a positive number if the first string comes after the second alphabetically.

Reporting Features (10-15)

- **Show Issued/Available Books:** Loops through the inventory and uses an `if` statement checking the `isAvailable()` boolean.

- **Show Books by Price Range:** Asks for min and max prices, then loops and uses an `if` statement checking if the book's price is between those two numbers.
- **Total Inventory Value:** Sets a `total` variable to 0. Loops through all books, adding each book's price to the `total`.
- **Most Expensive/Cheapest:** Assumes the first book (index 0) is both the most expensive and cheapest to start. Loops through the rest, updating these variables if it finds a book whose price is higher or lower than the current champions.
- **Count of Books per Author (Complex):** This uses "parallel arrays". It creates one array to hold unique author names and another array to hold the count for that author index. It loops through the inventory; for each book, it checks if that author is already in the unique list. If yes, it increases their count. If no, it adds the new author to the list and starts their count at 1.