Open-Ended Lab

**Proposal Submission: Library Management System**

**Group members:**

* Shahid Sohail(64129)
* Neha Imran(63967)
* Areej Imran(63921)

1. **Problem Definition**

**Problem Overview:**

Managing books in libraries and keeping track of their availability or issuance can be challenging, especially as the collection grows. Manual tracking often leads to mistakes. We propose creating a Library Management System to automate the process of managing, issuing, and tracking books' availability.

**Proposed Application:**

The system will have the following core functions:

1. **Add New Books**: Allow adding books to the system.
2. **Issue Books**: Mark books as issued when borrowed.
3. **Return Books**: Mark books as available when returned.
4. **View Available Books**: List books that are currently available for borrowing.

A **Singly Linked List** will be used to manage the books dynamically, providing flexibility for the system to grow or shrink as needed.

**Justification of Data Structure:**

The **Singly Linked List** is chosen for its benefits:

* **Dynamic Size**: The linked list adjusts as books are added or removed, unlike arrays, which need resizing.
* **Efficient Insertions/Deletions**: Operations like adding or removing books are faster because there is no need to shift elements as in an array.
* **Traversal**: The system will frequently need to traverse the list to check or update book status, which the linked list handles efficiently.

**Algorithmic Challenges & Solutions:**

1. **Issuing a Book**:
   * **Challenge**: Locate an available book and mark it as issued.
   * **Solution**: Traverse the list to find the first available book and change its status.
2. **Returning a Book**:
   * **Challenge**: Find the returned book and mark it as available.
   * **Solution**: Search the list by title and update its status to available.
3. **Viewing Available Books**:
   * **Challenge**: Display all books that are available.
   * **Solution**: Traverse the list and list all books marked as available.

**Plan to Address Algorithmic Challenges**:

* **Search/Update**: Searching for books in the linked list takes **O(n)** time, which is acceptable for typical library sizes. Updating the status (issuing/returning) can be done in **O(1)** if the book is found early in the list.

**Deliverables:**

* **Functional System**: The system will support adding, issuing, returning, and viewing books.
* **Code Implementation**: It will include classes for managing books and the linked list structure, with methods to handle operations.
* **Testing**: The system will be tested with sample books to ensure correct functionality.
* **Documentation**: The code will be documented with comments explaining the methods and addressing algorithmic challenges.

1. **Data Structure Design:**

**1. Book Class**

The **Book class** represents a book in the library and contains the book's title, author, and availability status (whether it is issued or available).

* **Attributes**:
  + title: Stores the title of the book.
  + author: Stores the author of the book.
  + isIssued: A boolean variable that keeps track of whether the book is issued or available.
* **Methods**:
  + issueBook(): Marks the book as issued.
  + returnBook(): Marks the book as available.
  + isAvailable(): Returns true if the book is available, false if it is issued.
  + toString(): Provides a string representation of the book (for displaying book details).

**package** LibraryManagnmentSystem;

**class** Book {

String title;

String author;

**boolean** isIssued;

**public** Book(String title, String author) {

**this**.title = title;

**this**.author = author;

**this**.isIssued = **false**; // Initially, the book is available

}

**public** **void** issueBook() {

isIssued = **true**;

}

**public** **void** returnBook() {

isIssued = **false**;

}

**public** **boolean** isAvailable() {

**return** !isIssued;

}

@Override

**public** String toString() {

**return** "Title: " + title + ", Author: " + author + ", Available: " +(isIssued ? "No" : "Yes");

}

}

#### ****2. Library Class (LinkedList Implementation)****

The **Library class** implements a **Singly Linked List** to manage the collection of books. Each node in the list contains a book and a pointer to the next node.

* **Attributes**:
  + Node: A private inner class that represents a node in the linked list. Each node contains a Book object and a reference to the next node (next).
  + head: The first node in the linked list, which represents the start of the list of books.
* **Methods**:
  + addBook(Book book): Adds a book to the end of the linked list.
  + issueBook(String title): Searches for an available book by title and issues it.
  + returnBook(String title): Finds a book by title and returns it (marks it as available).
  + viewAvailableBooks(): Traverses the linked list and prints out the books that are available for borrowing.

**class** Library {

**class** Node {

Book book;

Node next;

**public** Node(Book book) {

**this**.book = book;

**this**.next = **null**;

}

}

**private** Node head;

**public** Library() {

head = **null**;

}

**public** **void** addBook(Book book) {

Node newNode = **new** Node(book);

**if** (head == **null**) {

head = newNode;

} **else** {

Node temp = head;

**while** (temp.next != **null**) {

temp = temp.next;

}

temp.next = newNode;

}

}

**public** **void** issueBook(String title) {

Node current = head;

**while** (current != **null**) {

**if** (current.book.title.equals(title) && current.book.isAvailable()) {

current.book.issueBook();

System.***out***.println("Book issued: " + title);

**return**;

}

current = current.next;

}

System.***out***.println("Book not available or already issued.");

}

**public** **void** returnBook(String title) {

Node current = head;

**while** (current != **null**) {

**if** (current.book.title.equals(title) && !current.book.isAvailable()) {

current.book.returnBook();

System.***out***.println("Book returned: " + title);

**return**;

}

current = current.next;

}

System.***out***.println("This book was not issued.");

}

**public** **void** viewAvailableBooks() {

Node current = head;

System.***out***.println("Available Books:");

**while** (current != **null**) {

**if** (current.book.isAvailable()) {

System.***out***.println(current.book);

}

current = current.next;

}

}

}

1. **Class diagram:**

Book

- title: String

- author: String

- isIssued: boolean

Book (title, author)

+ issueBook()

+ returnBook()

+ isAvailable()

+ toString()

Library

- head: Node

Library ()

+ addBook(book):void + +issueBook(title):void

+ returnBook(title):void

+ viewAvailableBooks():void

Node

- book: Book

- next: Node

+ Node(book:Book)

**1.. \***

1. **Time Complexity Analysis:**

**addBook(Book book)**:

* 1. **Time Complexity**: O(n) — We have to traverse the list to find the last node to insert the new book.
  2. **Space Complexity**: O(1) — We are only adding a new node, not using extra space for other operations.

**issueBook(String title)**:

* 1. **Time Complexity**: O(n) — We may need to traverse the entire list to find the book by its title.
  2. **Space Complexity**: O(1) — No additional space is required for issuing a book.

**returnBook(String title)**:

* 1. **Time Complexity**: O(n) — We need to traverse the linked list to find the book and return it.
  2. **Space Complexity**: O(1) — No extra space is required.

**viewAvailableBooks()**:

* 1. **Time Complexity**: O(n) — We need to traverse the entire linked list to check each book's availability.
  2. **Space Complexity**: O(1) — We only output the available books, not storing them elsewhere.

1. **Algorithm Development**

This section outlines the key algorithms required for managing a library using a Singly Linked List. The primary operations are:

1. Adding a book to the library.
2. Issuing a book.
3. Returning a book.
4. Viewing available books.

### ****1. Add Book Algorithm****

The **addBook()** algorithm adds a new book to the library. This involves creating a new node in the linked list, which contains the book's details, and then linking it to the existing list of books.

##### **Algorithm: Add a Book**

1. Create a new node containing the book.
2. If the list is empty (i.e., the head is null), set the head to point to the new node.
3. If the list is not empty, traverse to the last node and set its next pointer to the new node.

**CODE IMPLEMENTATION:**

**public** **void** addBook(Book book) {

Node newNode = **new** Node(book);

**if** (head == **null**) {

head = newNode;

} **else** {

Node temp = head;

**while** (temp.next != **null**) {

temp = temp.next;

}

temp.next = newNode;

}

}

##### **Time and Space Complexity**:

* **Time Complexity**: O(n), where n is the number of books in the library. The time complexity is dominated by the traversal of the linked list to find the last node.
* **Space Complexity**: O(1), since only one new node is created during the operation.

### ****2. Issue Book Algorithm****

The **issueBook()** algorithm is responsible for issuing a book to a user. It involves:

1. Searching for the book by its title.
2. Checking if the book is available.
3. If available, mark it as issued.
4. If not available or already issued, print an error message.

##### **Algorithm: Issue a Book**

1. Search for the book by its title in the linked list.
2. If the book is found and is available, mark it as issued.
3. If the book is already issued or not found, print an error message.

**CODE IMPLEMENTATION:**

**public** **void** issueBook(String title) {

Node current = head;

**while** (current != **null**) {

**if** (current.book.title.equals(title) && current.book.isAvailable()) {

current.book.issueBook();

System.***out***.println("Book issued: " + title);

**return**;

}

current = current.next;

}

System.***out***.println("Book not available or already issued.");

}

##### **Time and Space Complexity**:

* **Time Complexity**: O(n), where n is the number of books in the library. In the worst case, the algorithm must traverse the entire list to find the book.
* **Space Complexity**: O(1), as only a few pointer variables (i.e., current) are used during traversal.

### ****3. Return Book Algorithm****

The **returnBook()** algorithm marks a book as returned. The algorithm involves:

1. Searching for the book by its title.
2. Checking if the book is issued (i.e., isIssued is true).
3. If issued, mark the book as available.
4. If not issued or not found, print an error message.

##### **Algorithm: Return a Book**

1. Search for the book by its title in the linked list.
2. If the book is found and is issued, mark it as returned.
3. If the book is not issued or not found, print an error message.

**CODE IMPLEMENTATION:**

**public** **void** returnBook(String title) {

Node current = head;

**while** (current != **null**) {

**if** (current.book.title.equals(title) && !current.book.isAvailable()) {

current.book.returnBook();

System.***out***.println("Book returned: " + title);

**return**;

}

current = current.next;

}

System.***out***.println("This book was not issued.");

}

##### **Time and Space Complexity**:

* **Time Complexity**: O(n), where n is the number of books in the library. In the worst case, the algorithm must traverse the entire list to find the book.
* **Space Complexity**: O(1), as only a reference to the current node is needed during the traversal.

### ****4. View Available Books Algorithm****

The **viewAvailableBooks()** algorithm lists all the books that are available for borrowing. This algorithm involves:

1. Traversing the linked list.
2. Checking the availability of each book.
3. Printing out the details of the books that are available.

##### **Algorithm: View Available Books**

1. Start from the head node and traverse the entire linked list.
2. For each book, check if it is available by calling isAvailable().
3. If available, print the details of the book.

**CODE IMPLEMENTATION:**

**public** **void** viewAvailableBooks() {

Node current = head;

System.***out***.println("Available Books:");

**while** (current != **null**) {

**if** (current.book.isAvailable()) {

System.***out***.println(current.book);

}

current = current.next;

}

}

##### **Time and Space Complexity**:

* **Time Complexity**: O(n), where n is the number of books. The algorithm needs to traverse the entire list to check the availability of each book.
* **Space Complexity**: O(1), as we only need a reference to the current node during the traversal.

1. **Integration and Optimization**

#### The Singly Linked List has been integrated to manage the library's books efficiently, ensuring memory efficiency and easy updates. The system uses optimized algorithms for common library operations.

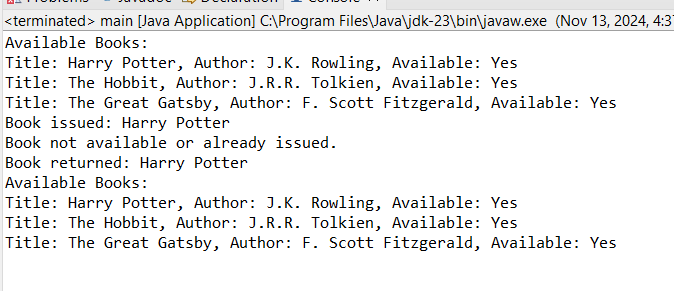
#### ****Complete Expected Outcome****

 Books are added and can be viewed with viewAvailableBooks().

 Issuing a book marks it unavailable, with error handling for already issued books.

 Returning a book marks it available again, ensuring only issued books can be returned.

 Available books are displayed, reflecting updates after issuing or returning



### Conclusion

The Library Management System efficiently manages books using a Singly Linked List, supporting key operations like adding, issuing, returning, and viewing books. The system is optimized for both performance and memory, meeting all requirements effectively.