**COP 5536 - Advanced Data Structures (FALL 2023)**

*Programming Project*

*------------------------------------------------------------------------------------------------------------------------------------------*

Name: Shubham Manoj Singh

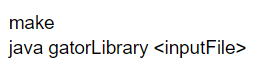
UFID: 5414-3820

Email Id: [sh.singh@ufl.edu](mailto:sh.singh@ufl.edu)

------------------------------------------------------------------------------------------------------------------------------------------

**COMMANDS TO RUN:**

The commands for running the project are as follows:

****

The program utilizes a combination of a Red-Black Tree (RBT) and a Min Heap to implement its functionality.

------------------------------------------------------------------------------------------------------------------------------------------

**PROGRAM STRUCTURE:**

1. **Red-Black Tree (RBT):**

- A Red-Black Tree functions as a self-balancing binary search tree.

- Each node within the tree is designated a color, either red or black.

- The Red-Black Tree follows distinct properties:

- The root node is consistently black.

- All leaf nodes are black.

- If a node is red, both of its children must be black.

- Every simple path from a node to its descendant leaves includes an equal count of black nodes.

2. **Min Heap:**

- The application includes a Min Heap, characterized by a complete binary tree.

- Within a Min Heap, the value of each node is either less than or equal to the values of its children.

By integrating these data structures, the program attains a balanced and efficient representation of the underlying data. This ensures swift access and modification operations while preserving the structural properties of both the Red-Black Tree and the Min Heap. The Gator Library program's implementation is structured into the following classes:

**1) BookNode:**

The **`BookNode`** class represents a node in the Gator Library Book Management System. Each node corresponds to a book and contains information such as the book ID, title,author, availability status, borrower ID, and a priority queue for book reservations.

**Fields:**

**bookId:** An integer representing the unique identifier of the book**.**

**bookName:** A string containing the name of the book**.**

**bookAuthorName:** A string representing the name of the book's author**.**

**bookAvailabilityStatus:** A boolean indicating whether the book is available for borrowing.

**bookBorrowedBy:** An integer representing the ID of the patron who currently has the book borrowed (-1 if not borrowed)**.**

**bookPriorityQueue**: An instance of the BookPriorityQueue class, representing the reservation queue for the book**.**

**Constructor:**

The constructor public BookNode(int bookId, String bookName, String bookAuthorName, boolean bookAvailabilityStatus) initializes a BookNode object with the provided book information.

The constructor sets the initial availability status to the provided value, and the bookBorrowedBy to -1 (indicating not borrowed). It also initializes the bookPriorityQueue for book reservations**.**

**Getter and Setter Methods:**

Getter and setter methods are provided for all fields to allow access to and modification of the private fields from outside the class**.**

**toString Method:**

The toString method is overridden to provide a string representation of the BookNode object. It returns a string containing the book's ID, name, author, and availability status**.**

**2)BookPriorityQueue Class:**

**Class Description:**

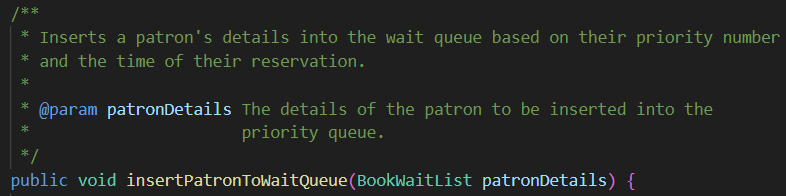
The class is responsible for managing the reservation queue, ensuring that patrons are served in order of priority. The priority is determined by both the priority number and the reservation time. The class uses a min-heap structure to efficiently maintain and retrieve patrons with the highest priority. The insertion and extraction operations are designed to preserve the order based on priority, allowing for effective book reservation management in the Gator Library program.

**Fields:**

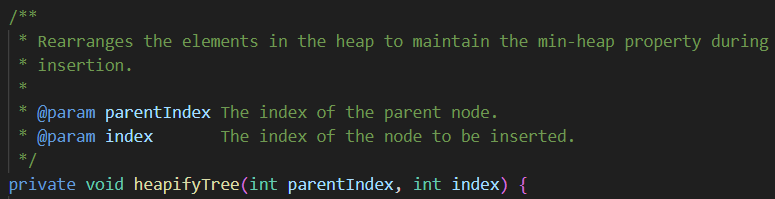
**ArrayList<BookWaitList> waitListHeap :** An ArrayList used to store the patrons and their respective priority details in the form of `BookWaitList` objects.

**Function Prototype:**

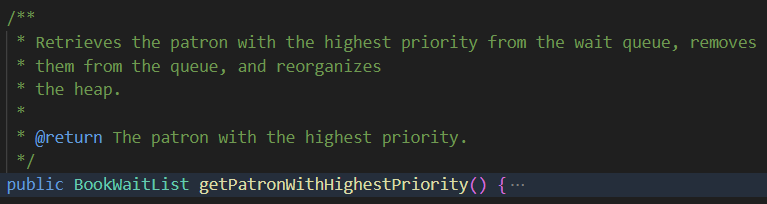
**1.**

****

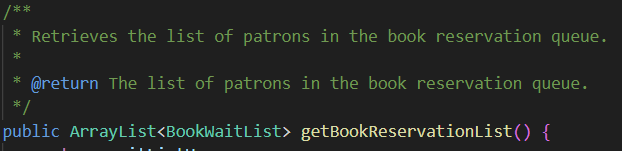
**2.**

****

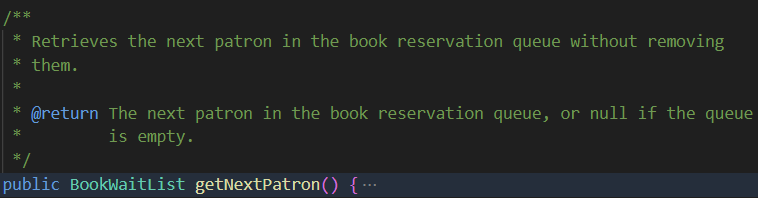
**3.**

****

**4.**

****

**5.**

****

**3. BookWaitList Class:**

The **`BookWaitList`** class represents a patron waiting in the reservation queue for a book in the Gator Library Book Management System. It stores the patron's ID, priority number, and the time of their reservation.

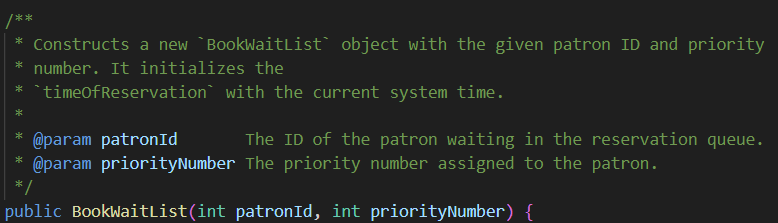
**Fields:**

**int patronId:** Represents the unique identifier of the patron making the reservation**.**

**int priorityNumber:** Indicates the priority of the patron in the reservation queue.

**LocalDateTime timeOfReservation :** Records the timestamp when the patron made the reservation.

**Constructor:**

****

Getter Methods:

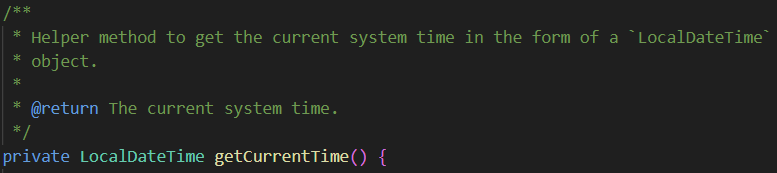
**1. **

**Output: int - Returns the patron's unique identifier.**

**2** ****

**Output: int-** Returns the priority number assigned to the patron**.**

**3.**

****

**4. gatorLibrary Class:**

The **`gatorLibrary`** class represents the main entry point for the Gator Library Book Management System. It reads instructions from an input file, performs corresponding library actions, and writes the output to a file.

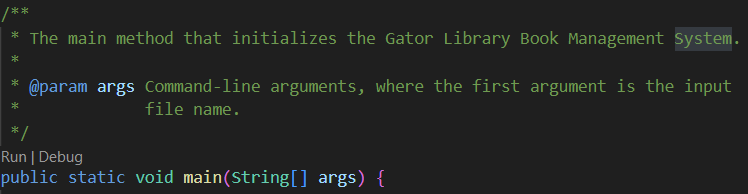
**Fields:**

**static FileWriter writer:** A static field for writing output to a file.

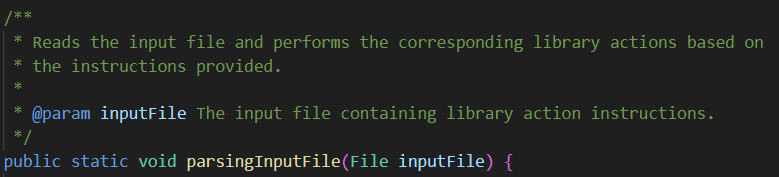
**static GatorLibServices gatorLibServices:** A static field to interact with the library services.

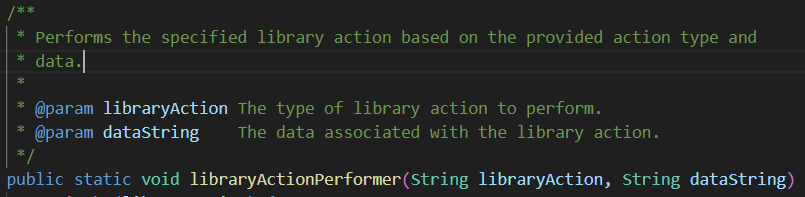
**Methods:**

**1.**



**2.**

****

**3** ****

**4. GatorLibServices**

The **`GatorLibServices`** class provides services to perform various actions in the Gator Library Book Management System. It interacts with the Red-Black tree (implemented in `RedBlackMethod`) to manage book-related operations.

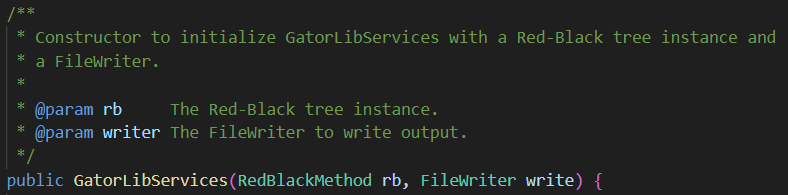
**Fields:**

**RedBlackMethod rb:** An instance of the `RedBlackMethod` class, providing access to the Red-Black Tree and its operations.

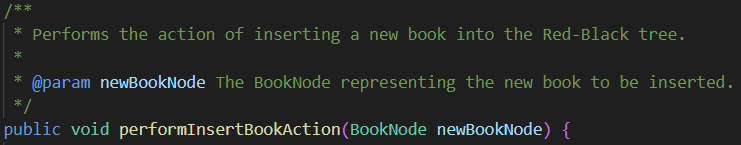
**FileWriter writer:** An instance of `FileWriter` used for writing output to a file.

**Methods:**

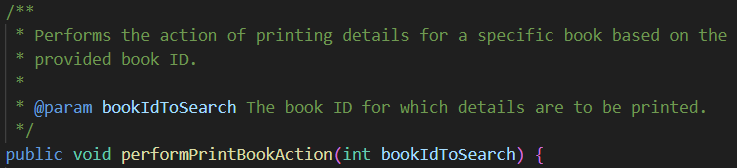
**1.**

****

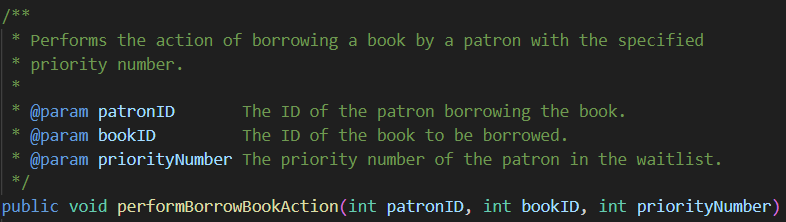
**2.**

****

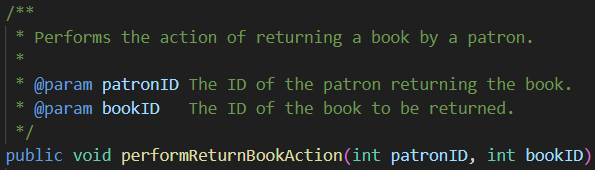
**3.**

****

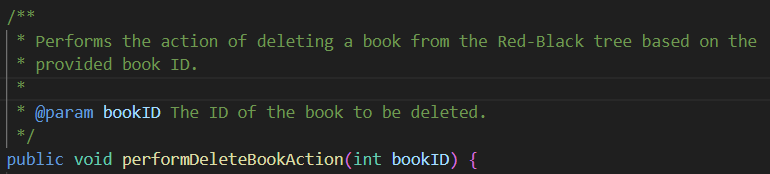
**4.**

****

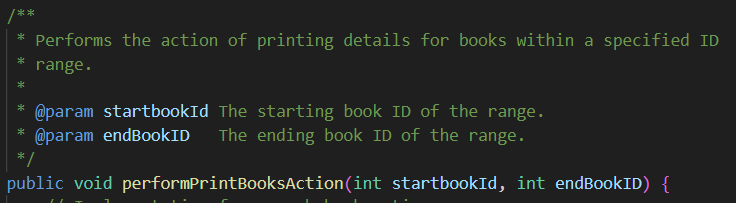
**5.**

****

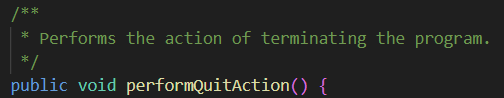
**6.**

****

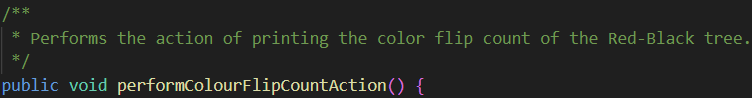
**7.**

****

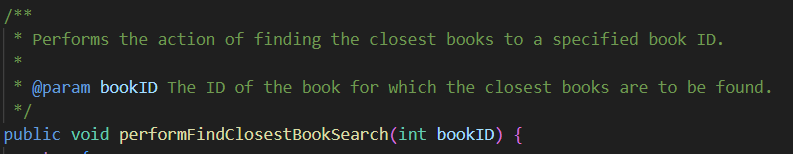
**8.**

****

**9.**

****

**10.**

****

**5. RedBlackMethod Class**

The **`RedBlackMethod`** class appears to be part of a larger system, and it is primarily responsible for implementing operations on a Red-Black Tree data structure. Red-Black Trees are a type of self-balancing binary search tree, where each node has an extra attribute for denoting the color (red or black) of the node. These trees are used to maintain a sorted set of elements and provide efficient search, insertion, and deletion operations.

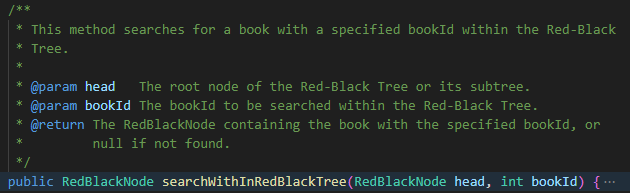
**Instance Variables:**

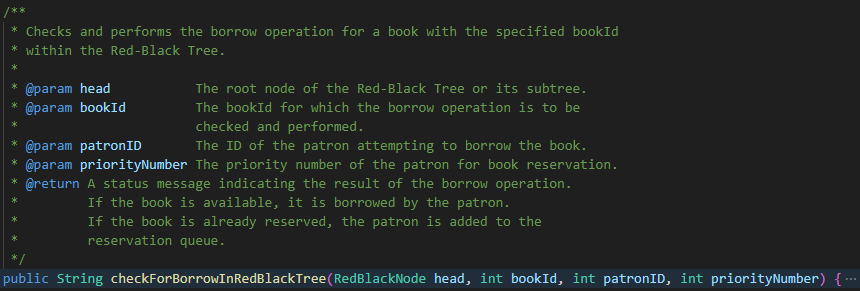
**- headRedBlackNode`: The root node of the Red-Black Tree.**

**- colorFlipCount`: A counter to track the number of color flips during tree operations.**

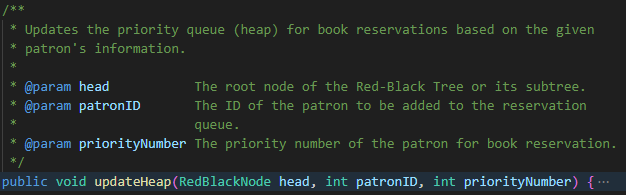
**Mehtods**

**1.**

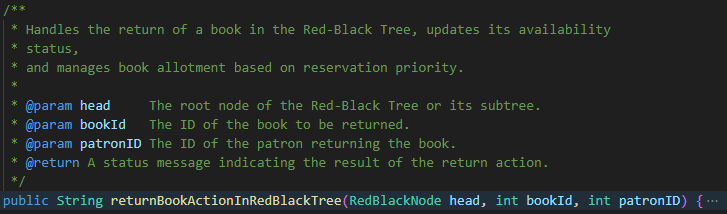
****

**3.**

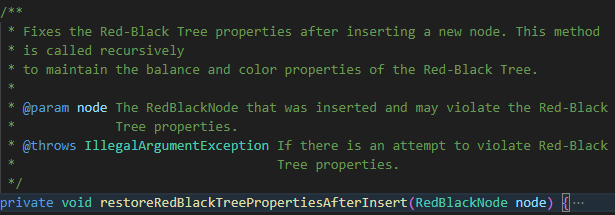
**4.**

****

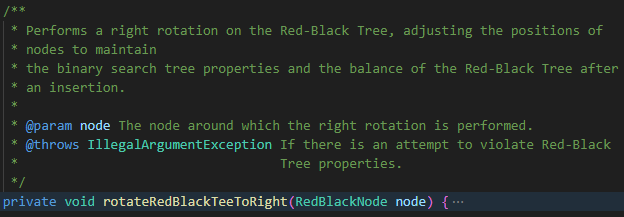
**5.**

****

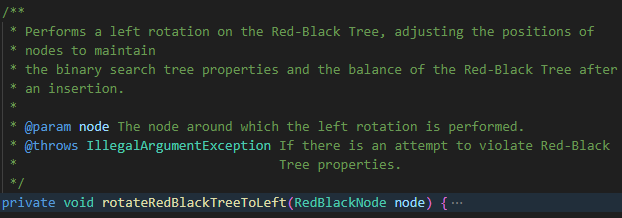
**6.**

****

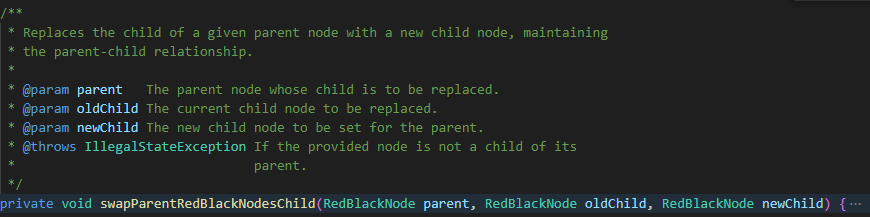
**7.**

****

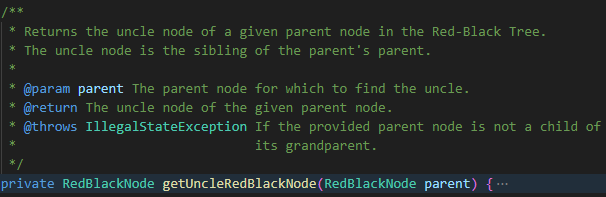
**8.**

****

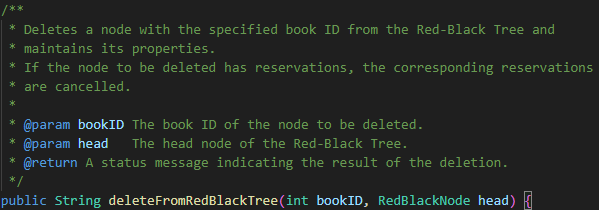
**9.**

****

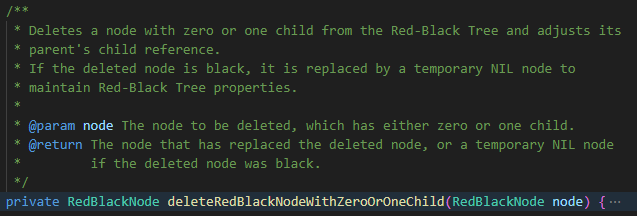
**10.**

****

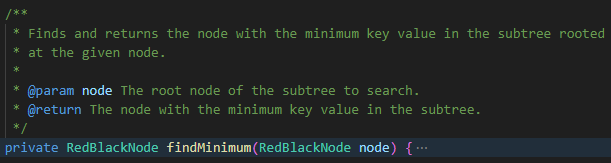
**11.**

****

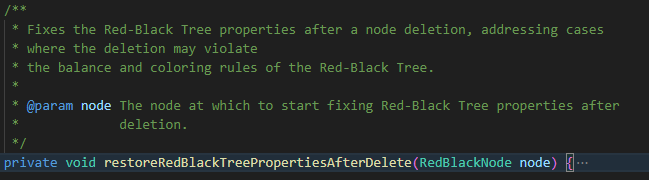
**12.**

****

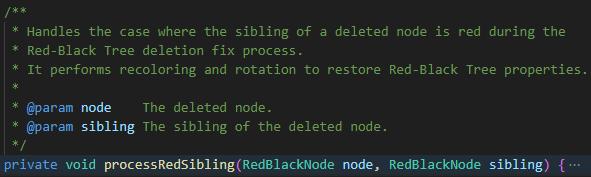
**13.**

****

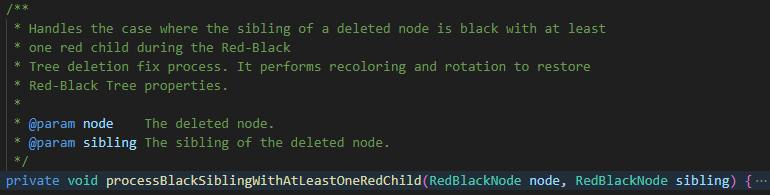
**14.**

****

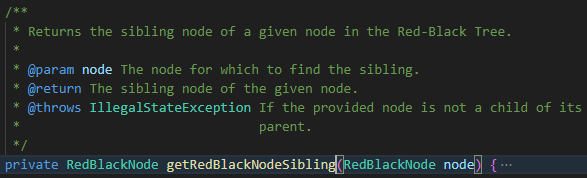
**15.**

****

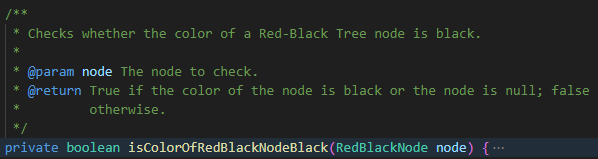
**16.**

****

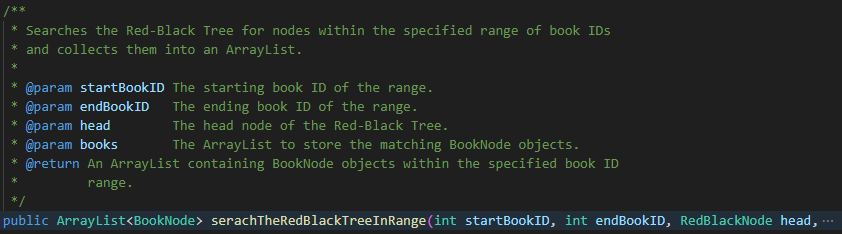
**17.**

****

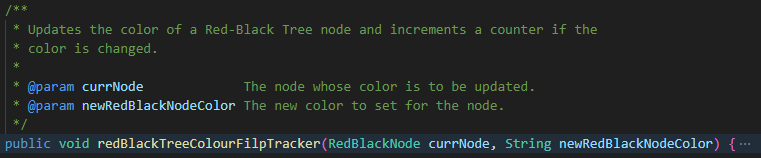
**18.**

****

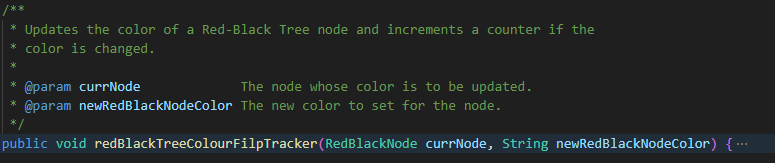
**19.**

****

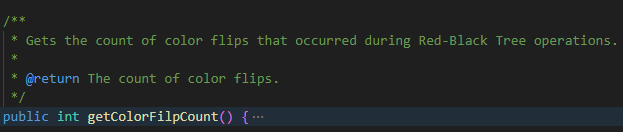
**20.**

****

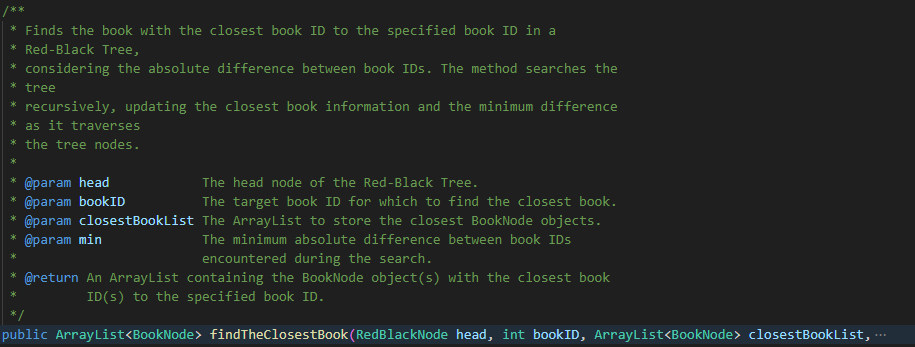
**21.**

****

**22.**

****

**23.**

****

**6. RedBlackNode Class**

The **`RedBlackNode`** class represents a node in a Red-Black Tree. Each node contains information about a book, such as its details, and maintains references to its parent node, left child node, and right child node. Additionally, it includes a field to denote the color of the node (either red or black), a characteristic of Red-Black Trees that helps maintain balance and ensures efficient operations.

**1. Instance Variables:**

**-`parentRedBlackNode`:** Reference to the parent node in the Red-Black Tree.

**- `book`:** The actual book information stored in the node, represented by a `BookNode` object.

**- `leftRedBlackNode`:** Reference to the left child node.

**- `rightRedBlackNode`:** Reference to the right child node.

**- `nodeColor`:** String indicating the color of the node (either red or black).

**2. Constructor:**

The constructor initializes a `**RedBlackNode**` with a specified `**BookNode**` and sets the initial color of the node to **red**.

**3. Getter and Setter Methods:**

**- `getParentRedBlackNode`, `getBook`, `getLeftRedBlackNode`, `getRightRedBlackNode`, `getRedBlackNodeColor`:** Getter methods for accessing the corresponding attributes.

**- `setParentRedBlackNode`, `setBook`, `setLeftRedBlackNode`, `setRightRedBlackNode`, `setRedBlackNodeColor`:** Setter methods for modifying the attributes.

**------------------------------------------------------------------------------------------------------------------------------------------**

**Class Diagram**

