UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS212 - Data structures and algorithms

Practical 2 Specifications: Binary Search Trees

Release date: 24-02-2025 at 06:00
Due date: 28-02-2025 at 23:59
Total marks: 84

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually; no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of the built-in Java data structures. Doing so will result in a zero mark. You may only use native 1-dimensional arrays where applicable.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a zero mark. Runtime errors are considered unsafe programming.
- Read the entire specification before you start coding.
- **Ensure your code compiles with Java 8**
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Please note that there is a late deadline which is 1 hour after the initial deadline, but you will lose 20% of your mark.

2 Overview

Binary Search Trees(BST) are Binary Trees in which a node's left children have values less than the node itself and its right children have values greater than the node itself. The primary advantage of BST is its hierarchical structure, which makes it efficient for searching insertions and deletions.

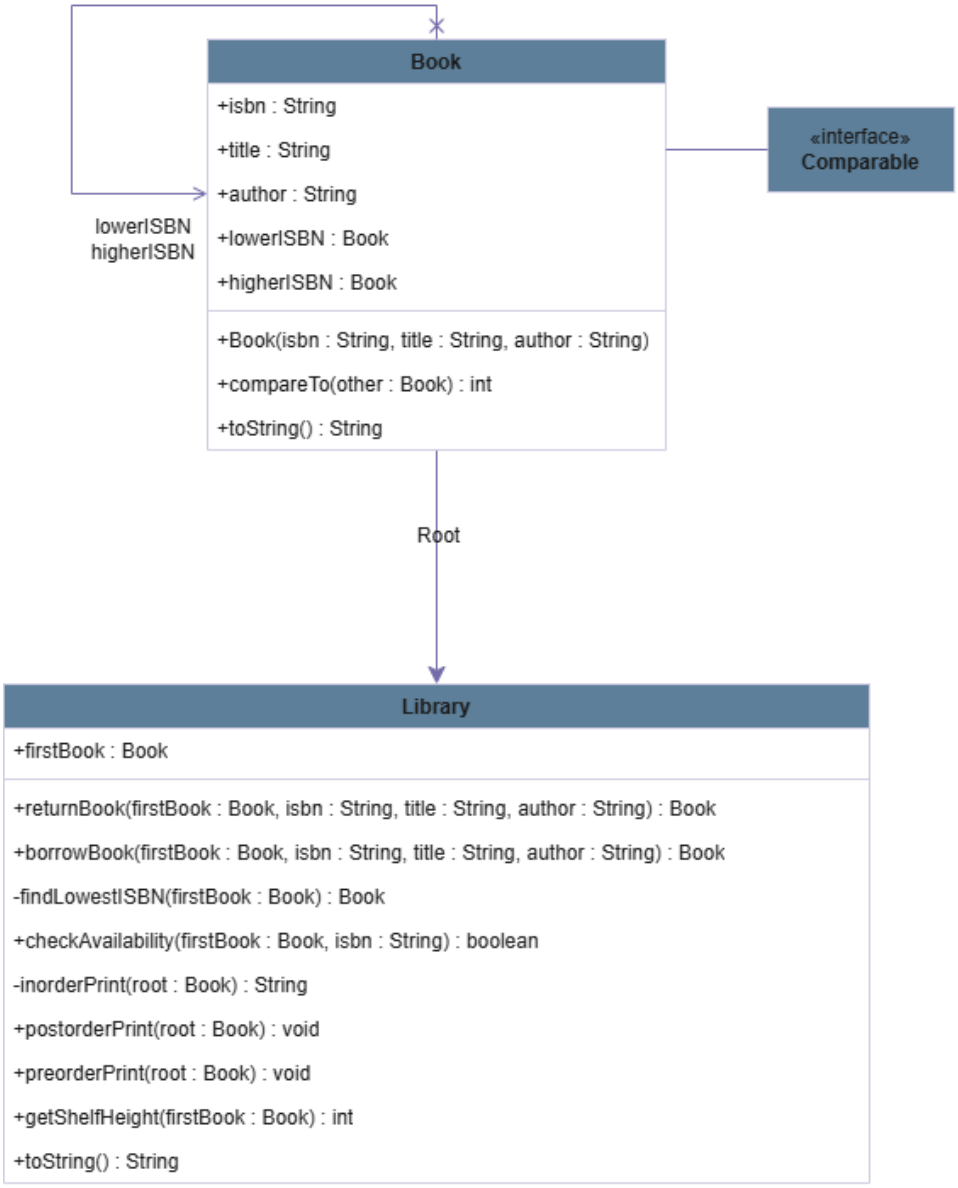

Figure 1: Binary Search Tree Structure for Library System

2.1 Book

This will represent a node in the Binary Search Tree

2.1.1 Members

isbn

- This represents an ISBN number belonging to a book in the Library

title

- This represents a book title belonging to a book in the Library

author

- This represents the author of a book in the Library

lowerISBN

- This represents an ISBN number belonging to a book in the Library

higherISBN

- This represents an ISBN number belonging to a book in the Library

2.1.2 Functions

Book

- This is the constructor to create Book objects.

compareTo

- Implements Comparable<Book> interface
- Compares books based on their ISBN numbers
- Returns negative if this book's ISBN is less than other's
- Returns positive if this book's ISBN is greater than other's
- Returns zero if ISBNs are equal
- Do not edit this as it will negatively affect your marks

toString

- Returns a formatted string representation of the Book and its left and right nodes id
- Includes ISBN, title, and author in a readable format
- Do not edit this as it will negatively affect your marks

2.2 Library

This will represent the Binary Search Tree implementation of the Library system

2.2.1 Members

firstBook

- This represents the root of the BST

2.2.2 Functions

returnBook

- This function adds a new book to the library by inserting it into the BST
- Returns the root node of the updated BST
- Please use compareTo method for comparisons
- Do nothing if there are duplicate books

borrowBook

- This function removes a book from the library by deleting it from the BST
- Returns the root node of the updated BST
- Please use compareTo method for comparisons
- Delete by copying using the successor

findLowestISBN

- Helper function that finds the book with the lowest ISBN in a subtree

checkAvailability

- Searches the BST to check if a book with given ISBN exists
- Returns true if the book is found, false otherwise

inorderPrint

- Performs recursive inorder traversal
- Make sure to return a string with the below Signature when traversing

```
Book{ISBN: " + root.isbn + ", Title: " + root.title + ", Author: " + root.author + "}\n"
```

postorderPrint

- Performs recursive postorder
- Make sure to return a string with the below Signature when traversing

```
Book{ISBN: " + root.isbn + ", Title: " + root.title + ", Author: " + root.author + "}\n"
```

preorderPrint

- Performs recursive preorder traversal
- Make sure to return a string with the below Signature when traversing

```
Book{ISBN: " + root.isbn + ", Title: " + root.title + ", Author: " + root.author + "}\n"
```

getShelfHeight

- Calculates the height of the BST
- Returns 0 for empty tree

toString

- Returns a formatted string representation of the Library and its contents
- Includes ISBN, title, and author in a readable format of all books currently in the Library
- Do not edit this as it will negatively affect your marks

3 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, approximately 10% of the assignment marks will be assigned to your testing skills. To do this, you will need to submit a testing main (inside the Main.java file) that will be used to test an Instructor-provided solution. You may add any helper functions to the Main.java file to aid your testing. In order to determine the coverage of your testing the jacoco tool. The following set of commands will be used to run jacoco:

```
javac *.java
rm -Rf cov
mkdir ./cov
java -javaagent:jacocoagent.jar=excludes=org.jacoco.*,destfile=./cov/output.exec
    -cp ./ Main
mv *.class ./cov
java -jar ./jacococli.jar report ./cov/output.exec --classfiles ./cov --html
    ./cov/report
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

and we will scale this ratio according to the size of the class.
The mark you will receive for the testing coverage task is determined using table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark. Remember that your main will be testing the instructor-provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

4 Upload checklist

The following files should be in the root of your archive

- Book.java
- Library.java

- Main.java
- Any textfiles needed by your Main

5 Allowed libraries

6 Submission

You need to submit your source files on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. Your code must be able to be compiled with the Java 8 standard.

For this practical, you will have 5 upload opportunities and your best mark will be your final mark. Upload your archive to the appropriate slot on the FitchFork website well before the deadline. **No late submissions will be accepted!**