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 4 Specifications: Red-Black Trees

Release date: 10-03-2025 at 06:00
Due date: 14-03-2025 at 23:59
Total marks: 90

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

Red-Black trees are variations of binary search trees in which each node is coloured red or black. These colours balance the tree by enforcing rules regarding the structure of red and black nodes. These trees are often used in database systems for efficient traversal and retrieval of data for indexed columns.

3 Tasks

For this practical, you will be implementing a ClassList that will use a Red-Black tree to store the names and surnames.

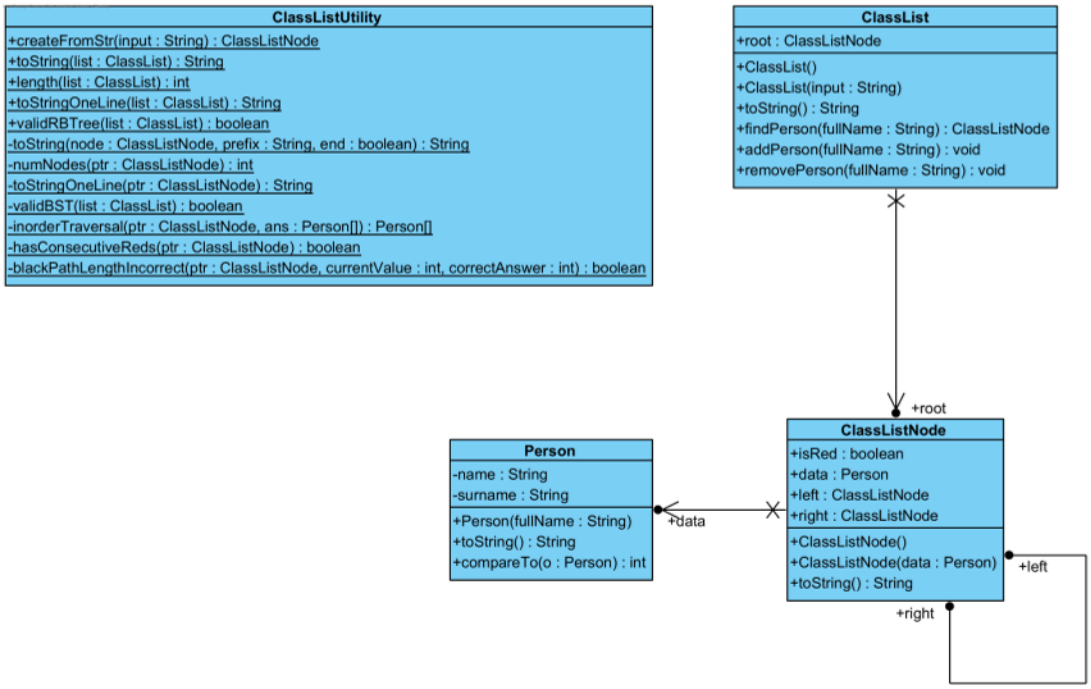

Figure 1: UML

3.1 Person

This class is given to you. Do not change this, since it will be overwritten on FitchFork. This class will be used as the data of the nodes in the ClassList.

3.1.1 Members

- name
  - This is the name of the person.
- surname
  - This is the surname of the person.

3.1.2 Functions

- Person
  - This is the constructor which takes in a full name, and then creates a Person object.

- toString
  - This returns a string representation of a Person object.
- compareTo
  - This allows Person to implement the Comparable interface.
  - Comparison is done first on surnames, and then if the surnames are equal, then according to firstName.
  - **Important:** `.equals` is not overwritten. Thus use `.compareTo()==0` instead of `.equals`.

3.2 ClassListNode

This class is given to you. Do not change this, since it will be overwritten on FitchFork. This class will be used as the nodes in the ClassList.

3.2.1 Members

- data
  - This is the data of the node.
- isRed
  - This shows whether the current node is red or black.
- left,right
  - These are the children of the current node

3.2.2 Functions

- ClassListNode
  - These are the constructors. There is a default constructor and a constructor that takes in data
- toString
  - This returns a string representation of the node.

3.3 ClassListUtility

This class is given to you. Do not change this, since it will be overwritten on FitchFork. This class contains helper functions which you can use for testing. **Note:** this file will not count for coverage in the testing task.

3.3.1 Functions

- createFromStr
  - This helper function converts the output from toStringOneLine to a Red-Black Tree. This is used in the String constructor for ClassList.
- toString
  - This returns a String representation of the ClassList.

- length
  - This starts with the toStringOneLine output.
  - This returns the size of the ClassList.
- toStringOneLine
  - This returns a one line String representation of the ClassList, which can be used to recreate any ClassList.
  - The first number in this is the length of the List. After this is the word True or False. This shows whether the ClassList is a valid Red-Black tree.
- validRBTree
  - This returns true if the passed-in ClassList is a valid Red-Black Tree.
- private helper functions
  - The rest of the functions in this file are helper functions which the class uses.

3.4 ClassList

3.4.1 Members

- root
  - This is the root of the Red-Black tree that is used to store the ClassList.

3.4.2 Functions

- ClassList
  - These functions are given to you. Do not change these, since this might lose you marks on FitchFork.
  - There is a default constructor that creates an empty tree, and a String constructor which recreates the ClassList from a String. The format of the String is the same as `ClassListUtility.toStringOneLine()`.
- toString
  - This function is given to you. Do not change this, since this might lose you marks on FitchFork.
  - This returns a string representation of the list using `ClassListUtility.toString()`.
- findPerson
  - This function returns a ClassListNode where the data has the same name and surname as the passed-in full name.
  - If the passed-in data was not found, then return `null`.
- addPerson
  - This function adds a person to the ClassList using the bottom-up Red-Black Tree insertion algorithm.
  - Don't add duplicates to the ClassList. Thus if the passed-in data is already in the list, then do not add it again.

- removePerson
  - This function removes a person from the ClassList using the top-down Red-Black Tree removal algorithm.
  - Because this algorithm is top-down, if the passed-in data is not in the tree, the tree might still change. This is the intended behaviour.

4 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.

5 Upload checklist

The following files should be in the root of your archive

- ClassList.java
- Main.java
- Any textfiles needed by your Main

6 Allowed libraries

- No imports allowed

7 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!**