Project 3

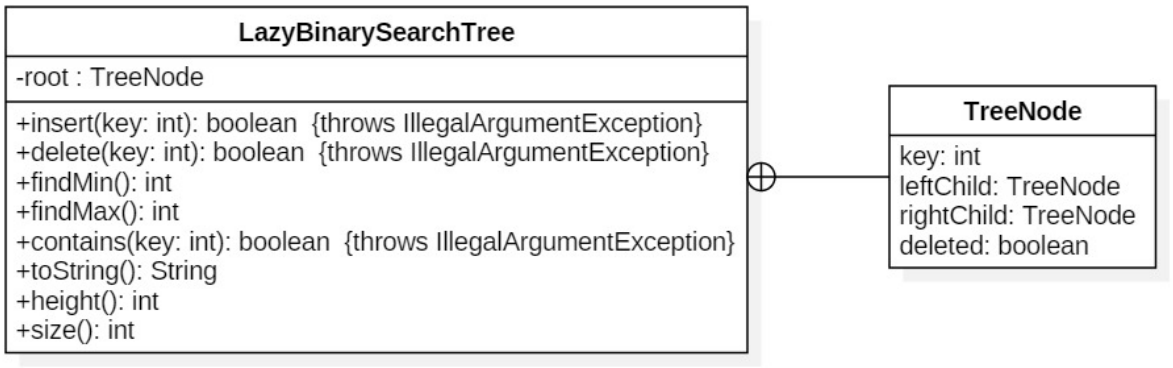
Binary Search Trees with Lazy Deletion

The task of this project is to implement in Java a binary search tree with lazy deletion. The BST class should contain a nested tree node class that is used to implement the BST.

Specification

The project must implement the following specification exactly, which includes identifier names, method signatures, the presence or absence of exceptional behavior, etc. Anything not clearly indicated by the UML class diagram (such as the access level of `TreeNode` fields, note the TreeNode class itself is private nested inside LazyBinarySearchTree class) or explicitly stated in the natural language description is left up to your discretion. You may also add helper methods and additional fields as you see fit – in fact, you may find it necessary to do so!

Structure



Note that the (+) relation in UML denotes nesting of scope, and not just composition of use.

Behavior

`insert` should insert a new element to a leaf node. The valid set of keys is all integers in the range $[1,99]$. If the new element would be a duplicate of a non-deleted element already in the tree, then `insert` should do nothing. However, if the new element is not a duplicate of a non-deleted element, but is a duplicate of a deleted element, then `insert` should "undelete" the deleted element in-place rather than physically inserting a new copy of the element. The return value of `insert` should indicate whether `insert` logically (as opposed to physically) inserted a new element.

`delete` should not physically remove an element from the tree. Rather, it should mark the specified element as logically deleted. If the specified element is not in the tree or is already marked as deleted, then `delete` should do nothing. The return value of `delete` should indicate whether `delete` logically deleted an element.

`findMin` should return the value of the minimum non-deleted element, or `-1` if none exists.

`findMax` should return the value of the maximum non-deleted element, or `-1` if none exists.

`contains` should return whether the given element both exists in the tree and is non-deleted.

`toString` should perform an pre-order traversal of the tree and print the value of each element, including elements marked as deleted. However, elements that are marked as deleted should be preceded by a single asterisk. Every pair of adjacent elements should be separated by whitespace in the printing, but no whitespace should occur between an asterisk and the element with which it is associated. Leading and trailing whitespace is tolerable, but it will be ignored. (no additional messages should be printed, either) An example of the output is as follows:

```
45 30 2 *5 47 50 *60
```

`height` should return the height of the tree, including "deleted" elements.

`size` should return the count of elements in the tree, including "deleted" ones.

The valid set of keys is all integers in the range $[1,99]$. Every method that accepts a key argument should throw an `IllegalArgumentException` with an appropriate message iff the argument is invalid. No method specified herein should (intentionally) throw any other exception.

Do not define any package (for this project, anyway). I.e. you should use the default package.

Along with the LazyBinarySeacrTree class include another class that has a main function. You can name this class anything just clearly indicate the name in Readme File. This class will take two command line arguments. The first argument will be the input file name and second will be output file name. The input file will be given to the program and the output file will be generated by the program. This main class will create an instance of LazyBinarySearchTree and do the operations specified in the input file.

The format of the input file will be

```
Insert:98
Insert:67
Insert:55
Insert:45
PrintTree
Delete:84
Delete:45
Contains:45
FindMin
FindMax
PrintTree
Height
Size
Insert:84
Insert:32
Insert:132
```

```
PrintTree
Insert:980
Insert
hih
```

{Insert and Delete will be followed by ":" semicolon and the key to be inserted. You have to check for validity of the key as well as the line in the file. "PrintTree" corresponds to toString method in the class. All commands are case sensitive.}

The corresponding correct output file will be

```
True
True
True
True
98 67 55 45
False
True
False
55
98
98 67 55 *45
3
4
True
True
Error in insert: IllegalArgumentException raised
98 67 55 *45 32 84
Error in insert: IllegalArgumentException raised
Error in Line: Insert
Error in Line: hih
```

Submission

Submit the following items through eLearning:

1. **README.txt**
   This should identify who you are (name, NetID, etc.), which project you are submitting, what files comprise your project, how you developed and compiled your project (e.g. what IDE or text editor, which version of Java, what compiler options, etc.), and any other information you believe the grader should know or you want the grader to know. Please

include sample commands that corresponding output. If some methods do not work completely please indicate.

2. **LazyBinarySearchTree.java**
   This java file should not include a `main` method, since it is intended to be used like library code. (that is also how it will be tested) . Have the file inside default package
3. **Main Class**
   Give an appropriate name and indicate it in the readme file. Should take two command line argument and process the input file and write the output file accordingly. Have the file inside default package

All items should be submitted as a single zipped file named with your lowercase NetID. The file structure should resemble the following example:

```
*-- abc123789.zip
    |-- README.txt
    |-- LazyBinarySearchTree.java
    |-- Main.java
```

Evaluation

This project will be evaluated primarily according to the correctness of the various tree methods specified above, and secondarily according to the quality of your code apart from its correctness. Correctness will be verified by empirical testing, whereas code quality will be verified by visual inspection. Testing for correctness will involve installing your `LazyBinarySearchTree` implementation into a test suite that will "take it through the paces" as if it were library code – this is why exactly adhering to the specifications for naming and method signatures is important. Code quality comprises engineering and design (e.g. modularity), documentation and comments (e.g. Javadoc), and style and layout (e.g. visually grouping lines into logical "paragraphs").

The rubric is as follows:

Category	Weight
`insert`	10%
`delete`	5%
`findMax`	20%
`findMin`	20%
`contains`	5%
`toString`	10%
`Height`	10%
`size`	5%
Code Quality /Error Checking	15%

Note that your source code should include a suitable Javadoc for every public method and class.

Commentary

My hope is that by reducing the scope of the project (you no longer have to grapple with the vagaries of direct end-user interaction), you will be able to focus more on (1) understanding and correctly implementing the data structures and algorithms that we are studying in this class, and (2) developing the skills and habits required to write clean, clear, and robust code. The tradeoff is that your data structure implementation will be tested more thoroughly.