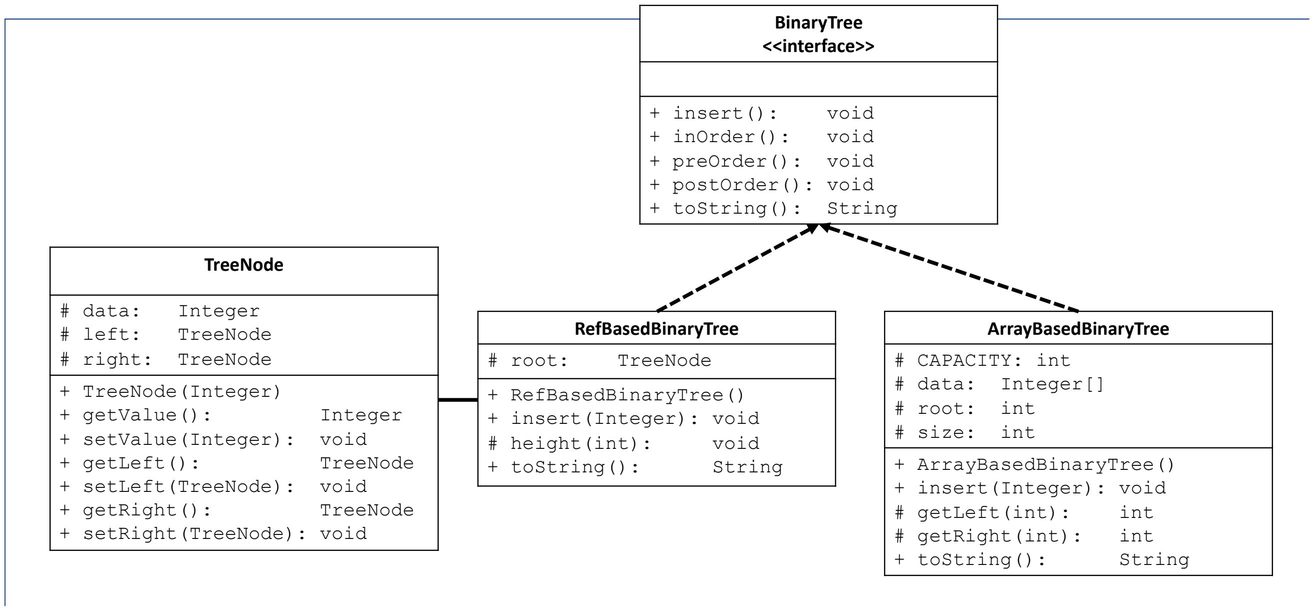
Lab 8

Objectives

- Introduction to Binary Trees
- Practice with implementing an interface with both reference and array based implementations

Part I – implementing a BinaryTree interface

The following is a UML representation of a `BinaryTree` abstract data type. We have provided you with the interface `BinaryTree.java` and the classes `ArrayBasedBinaryTree.java`, `RefBasedBinaryTree.java` and `TreeNode.java`. The methods in `ArrayBasedBinaryTree` and `RefBasedBinaryTree` have been left as stubs for you to complete.



1. Start by completing the implementation of `ArrayBasedBinaryTree`

   A small `main` is included in the class that will allow you to test in isolation by compiling and running:

   ```
   javac ArrayBasedBinaryTree.java
   java ArrayBasedBinaryTree
   ```

   When you are complete, it should insert the given elements and have the expected output for the calls to the traversal and `toString` methods. You will need to draw a picture of the tree that is created and inserted into in main. You can then traverse the drawn tree by hand to compare it to your output to ensure your method implementations are correct.

Tips:

- The calculation of the indices of the left and right subtree is dependent on the index of the root is initialized to in the constructor.
  - If the root is initialized to 0 in the constructor, the calculations to determine the children of a current node are:
    - index of the left child = 2 * index of the current node + 1
    - index of the right child = 2 * index of the current node + 2

- If the root is initialized to 1 in the constructor, the calculations to determine the children of a current node are:
  - index of the left child = 2 * index of the current node
  - index of the right child = 2 * index of the current node + 1
- convince yourself:
  - draw a tree of height 3, number the elements in a level order starting at 0. Do the indices of the left and right children match the calculation described above?
  - repeat with a numbering starting at 1

- The traversal methods are the simplest to write recursively
  - you will need helper methods that takes the index of a tree element as a parameter much like the recursive list methods you wrote.
  - Think carefully about the basecase:
    - How do you know you have reached an index that is out of bound of the array?
    - How do you know you have reached a leaf node?

CHECKPOINT (Ungraded) – Now might be a good time to check-in with the TA if you are aren't sure you have completed the tasks as expected. Remember, please don't hesitate to ask questions if you are unclear about anything.

Next you will work on `RefBasedBinaryTree`

NOTE: the insertion algorithm is not the same as the `ArrayBasedBinaryTree` implementation so the traversals will not have the same output.

Take the time to understand what the `insert` method is doing by hand-drawing the tree that will be created by the calls to insert in the `main` method. You will use this drawing to ensure your traversal and `toString` methods are correct.

CHECKPOINT (Ungraded) – Now might be a good time to check-in with the TA if you are aren't sure you have completed the tasks as expected. Remember, please don't hesitate to ask questions if you are unclear about anything.

Complete the implementation of the methods in `RefBasedBinaryTree`
Again, a small `main` is included in the class allowing you to compile and run:
```
javac RefBasedBinaryTree.java
java RefBasedBinaryTree
```
When you are complete it should insert the given elements and have the expected output for the calls to the traversal and `toString` methods.

Tips:
- The traversal methods are the simplest to write recursively – you will need helper methods that take a `TreeNode` as a parameter much like the recursive list methods you wrote.
- The height method is also easiest to do recursively. Start with the recursive template as you did with the traversal methods and then reason about what the recursive call on the left and then the right will give you – this will help you understand what to do at the current node.

SUBMISSION (Graded) – Submit **`RefBasedBinaryTree.java`** file into the Lab 8 page on ConneX.