Lab 8

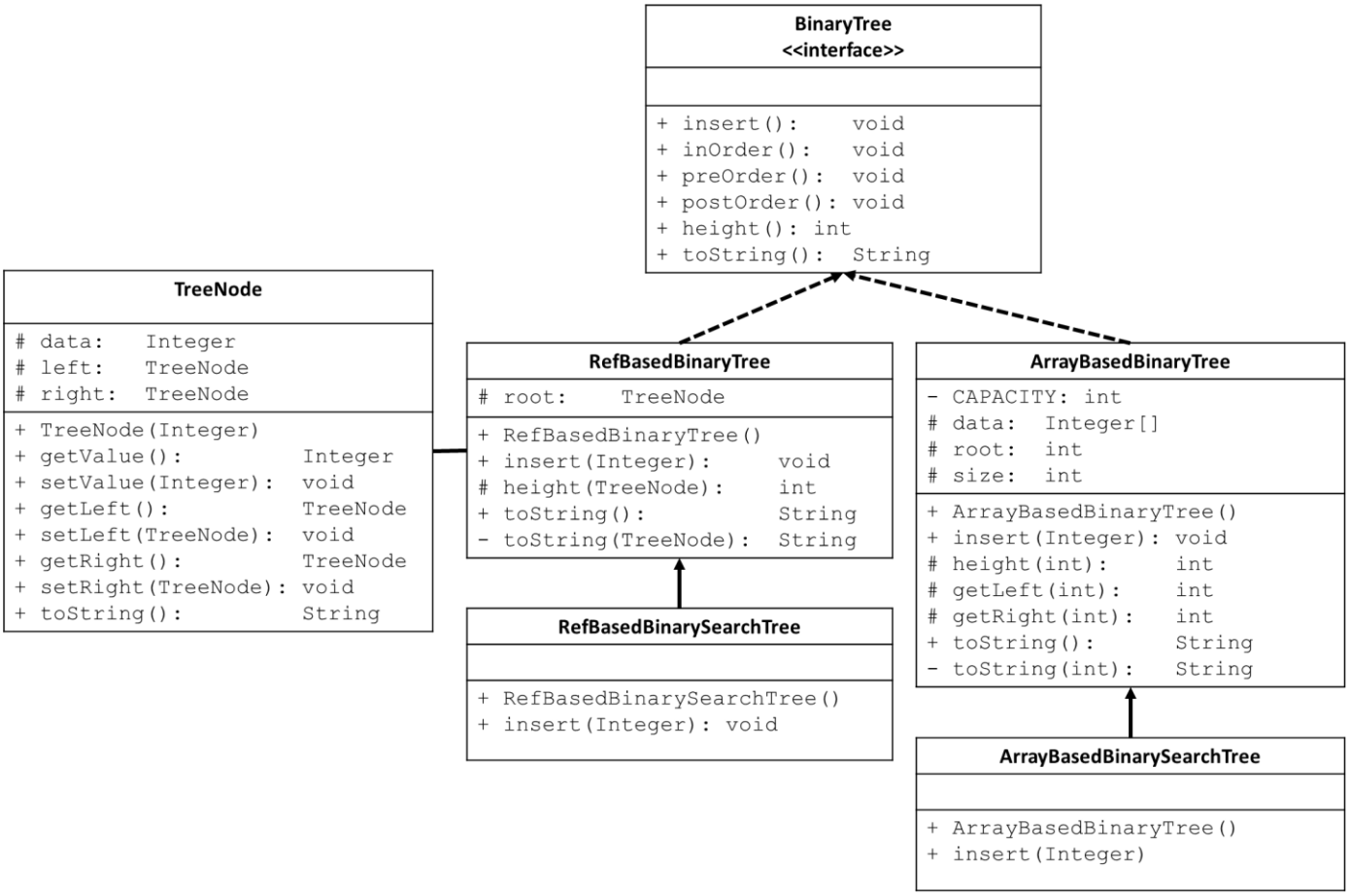
Objectives

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

Implementing the `BinaryTree` interface

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

Note: In this lab you are working with **Binary Trees**, *not* **Binary Search Trees**!



Tips:

- The calculation of the indices of the left and right subtree depends on the initial value assigned to the root index 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 base case:
        - 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?

1. Start by completing the implementation of **`ArrayBasedBinaryTree`**.
   this class has stubs marked with TODOs. We suggest you implement them in the following order:
   a) **`constructor`**
   b) **`getLeft`** and **`getRight`**
   c) **`insert`**
   d) **`inOrder`**, **`preOrder`**, and **`postOrder`**.
   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
   ```
   e) **`height`**

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

2. Next, complete the implementation of **`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 is created by the calls to insert in the **`main`** method. Use this drawing to ensure your traversal and **`toString`** methods are correct.

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

3. Open **`RefBasedBinaryTree.java`**. We will start with a code tracing exercise. Work with a partner in lab and draw out the tree as values 2, 3, 4, 5, 6, and 7 are inserted into an initially empty tree. Does the final tree look like you expected? Double check that your tree is correct with a TA if you need to.

   Complete the implementation of the methods in `RefBasedBinaryTree`.

   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 subtree will give you
     - what the recursive call on the right subtree will give you

     This will help you understand what to do at the current node, given the results provided by the recursive calls on the left and right subtrees.

   The class has stubs marked with TODOs to complete. We suggest you implement them in the following order:
   a) **`constructor`**
   b) **`height`** and the **`height`** helper method
   c) **`inOrder`**, **`preOrder`**, and **`postOrder`**.

   Again, a small **`main`** is included in the class allowing you to test the traversal methods by compiling and running **`RefBasedBinaryTree`** with the following:

```
javac RefBasedBinaryTree.java
java RefBasedBinaryTree
```

CHECKPOINT – LAB COMPLETE – Remember to demonstrate your completed lab to your TA.