Lab 8

Reminder: Your code is to be designed and written by only you and not to be shared with anyone else. See the Course Outline for details explaining the policies on Academic Integrity. Submissions that violate the Academic Integrity policy will be forwarded directly to the Computer Science Academic Integrity Committee.
All materials provided to you for this work are copyrighted, these and all solutions you create for this work cannot be shared in any form (digital, printed or otherwise). Any violations of this will be investigated and reported to Academic Integrity.

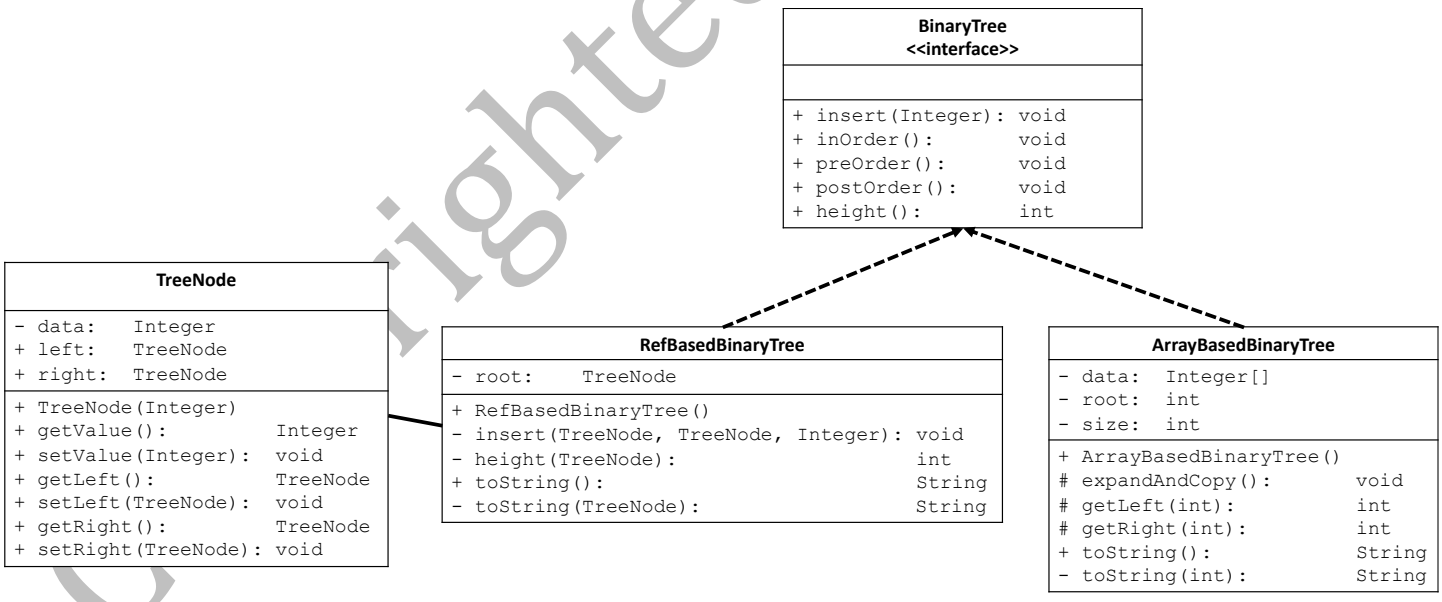
Objectives

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

Implementing a BinaryTree interface

NOTE: these are **NOT** Binary Search Trees in this lab!

The following is a UML representation of a `BinaryTree` abstract data type. We have provided you with the `BinaryTree` interface and the classes `ArrayBasedBinaryTree`, `RefBasedBinaryTree` and `TreeNode` classes. There are methods in `ArrayBasedBinaryTree` and `RefBasedBinaryTree` have been left as stubs for you to complete. We have provided you with the `Lab8Tester` and `main` methods in each of the `ArrayBasedBinaryTree` and `RefBasedBinaryTree` classes to help with your testing.



1. Start by completing the implementation of `ArrayBasedBinaryTree`
   This class has stubs marked with TODOs. We suggest you implement them in the following order:
   - `constructor`
   - `getLeft` and `getRight`
   - `insert`
   - `inOrder`, `preOrder` and `postOrder`
     A small `main` is included in this class that will allow you to test these traversal methods by compiling and running `ArrayBasedBinaryTree`:
     - `javac ArrayBasedBinaryTree.java`
     - `java ArrayBasedBinaryTree`
   - `height`

Tips for implementing `getLeft` and `getRight`:

- 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?

CHECK POINT – get help from your lab TA if you are unable to complete this part.

2. Understanding `RefBasedBinaryTree` implementation

   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 methods are correct.

CHECK POINT – get help from your lab TA if you are unable to complete this part.

3. Complete the implementation of `RefBasedBinaryTree`

   This class has stubs marked with TODOs for you to completes. We suggest you implement them in the following order:
   - `constructor`
   - `height` and `height` helper method
   - `inOrder`, `preOrder` and `postOrder`
     A small `main` is included in this class that will allow you to test these traversal methods by compiling and running `RefBasedBinaryTree`:
     - `javac RefBasedBinaryTree.java`
     - `java 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 from the right and the left subtrees.

CHECK POINT – get help from your lab TA if you are unable to complete this part.