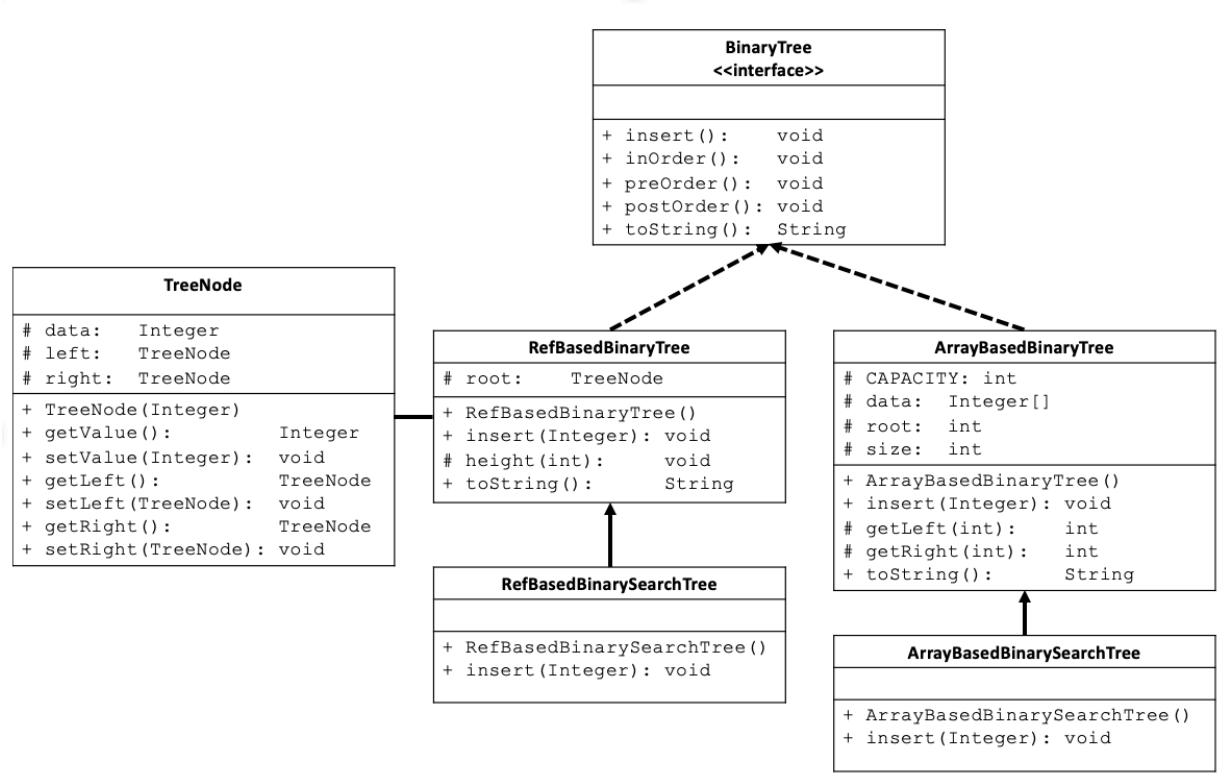
Lab 9

Objectives

- Extending Binary Trees to make Binary Search Trees
- Practice with extending a class and overriding methods

Part I – Extending BinaryTree

In this lab you will implement the `ArrayBasedBinarySearchTree.java` and `RefBasedBinarySearchTree.java` that will `extend` the `ArrayBasedBinaryTree.java` and `RefBasedBinaryTree.java` respectively (as shown in the UML).



RECALL: A Binary Search Tree maintains the invariant that for every element in the tree, every element in its left subtree must be smaller than the parent and every element in its right subtree must be larger than the parent

1. Download all the files provided to you in this lab to your Lab9 folder.
2. You will be implementing the necessary methods in `ArrayBasedBinarySearchTree.java` that extends `ArrayBasedBinarySearchTree.java`
3. Understand which methods `ArrayBasedBinarySearchTree` will inherit from the super class
4. Implement the required methods that you will **override** from the super class (insertion will be much different as the insert must maintain the invariant of the Binary Search Tree). Implement two versions of the `insert` function: an **iterative** version and then a **recursive** version
5. Look at the `main` method. Hand draw what the tree will look like after the calls to `insert` in the `main` and write out the expected in-order, pre-order and post-order traversals.
6. Compile and run and compare the output with your expected results from Step 5.

CHECK POINT – get your lab TA to check off after you have completed this. They will want to see the methods you implemented in `ArrayBasedBinarySearchTree` and see you run the `main` in each.

7. Repeat steps 3-6 for `RefBasedBinarySearchTree.java`

CHECK POINT – get your lab TA to check off after you have completed this. They will want to see the methods you implemented in `RefBasedBinarySearchTree` and see you run the `main` in each.

Part II – Adding functionality

In this part of the lab you will be adding functionality to `RefBasedBinaryTree.java` and `RefBasedBinarySearchTree.java`

For each method description below do the following:

1. Implement and test the method in `RefBasedBinaryTree.java`
2. Determine whether `RefBasedBinarySearchTree.java` should inherit the implementation from `RefBasedBinaryTree.java` or if it should override it. Ask yourself, will the algorithm be different given the constraints of a Binary Search Tree
3. If you determined you should implement the method in `RefBasedBinarySearchTree.java`, implement two versions of that method: an **iterative** version and then a **recursive** version

```
/*
 * Method name: sum
 * Purpose: computes the sum of all elements in this BinaryTree
 * Parameters: none
 * Returns: int – the sum
 */

/*
 * Method name: find
 * Purpose: determines whether val is in this BinaryTree
 * Parameters: int val
 * Returns: boolean – true if val is found, false otherwise
 */

/*
 * Method name: getMax
 * Purpose: gets and returns the largest value in this BinaryTree
 * Parameters: none
 * Throws: TreeEmptyException if called on an empty tree
 * Returns: int – the largest value
 */


/*
 * Method name: levelOrder
 * Purpose: prints all values in this BinaryTree in a level order
 * Parameters: none
 * Returns: nothing
 */
```

CHECK POINT – get your lab TA to check off after you have completed this. They will want to see the methods you implemented and see you run tests in the `main` for each.