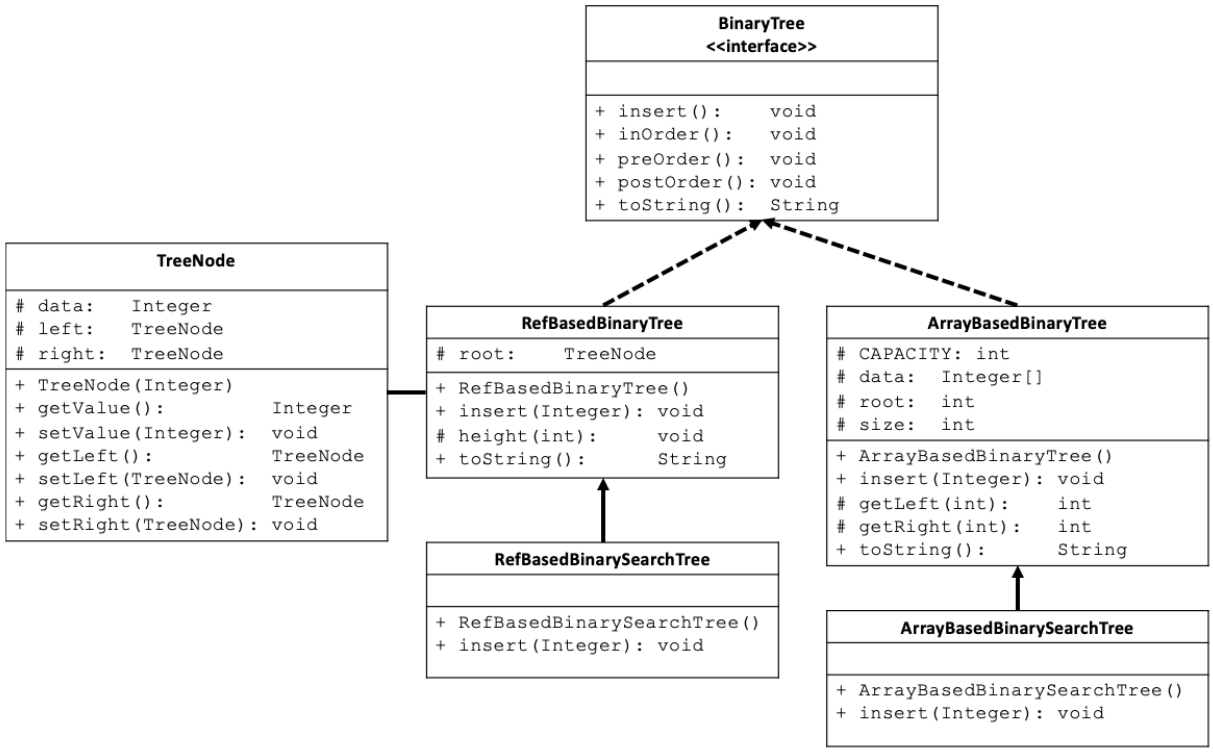
# Lab 9

## 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 node $n$ in the tree, all node's in $n$'s left subtree have a key less than $n$'s, and all nodes in $n$'s right subtree have a key larger than $n$'s.
Download all the files provided to you in this lab to your Lab9 folder.

1. You will be implementing the necessary methods in `ArrayBasedBinarySearchTree.java` that extends `ArrayBasedBinarySearchTree.java`
2. Understand which methods `ArrayBasedBinarySearchTree` will inherit from the super class
3. 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
4. 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.
5. Compile and run and compare the output with your expected results from Step 5.
6. Repeat steps 3-6 for `RefBasedBinarySearchTree.java`

**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.

## 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
 */
```

**SUBMISSION (Graded)** – Submit the **`RefBasedBinaryTree.java`** and **`RefBasedBinarySerchTree.java`** files into the Lab 9 page on ConneX.