# Lab Worksheet 2: Binary Search Trees

The goal of this worksheet is:

- To understand how Binary Search Trees work.
- To understand Java API documentation.
- To write code to implement and use a Binary Search Tree.

Download the `W2-BSTs.zip` file from Brightspace and import it into IntelliJ.

The `doc` folder contains documentation for the data structures we use in this worksheet, including descriptions of all the methods that are available. Before you start programming, you should examine the data structures that are contained within this documentation. Note that there are two packages:

- The dsa.iface package contains all the interfaces for the data structures.
- The dsa.impl package contains implementations of the data structures.

Pay particular attention to the methods available in the ITree and IBinaryTree interfaces, and the ProperLinkedBinaryTree class.

You will write your code in the Java files that are in the `src` folder. This project contains a main() method in the Main class. This will add and remove elements to/from a Binary Search Tree and print the shape of the tree afterwards. To get the correct output, you must implement the Binary Search Tree methods, however.

**Your task** is to implement the following methods in the `BinarySearchTree` class in the `dsa.impl` package. The algorithm for each method is included in the comments of the class. You should implement all four methods in the order they are listed below.

- `private IPosition<T> find( IPosition<T> node, T element )`: Return the node/position that contains `element`. If `element` is not contained in the tree, return the external node that was reached during the search. Refer to the lecture slides to see how this method should work.

  **Hint:** When implementing this method, you will need to compare `element` with the elements stored in the nodes. For this, you will need to use the compareTo() method (do not use == to compare the value of objects).

- `public void insert( T element )`: Insert `element` into the appropriate place in the tree. You can use the `find()` method to find the correct external node to expand.

  **Hint**: When you have found the correct external node to expand, you can use the `expandExternal()` method from `ProperLinkedBinaryTree` to do the insertion. Remember that `BinarySearchTree` is a subclass of `ProperLinkedBinaryTree`.

  After you complete this method, the output of running the main() method in Main should be correct for the first tree. See the next page for details about how the tree is printed.

- `public boolean contains( T element ):` Check if the tree contains `element`.

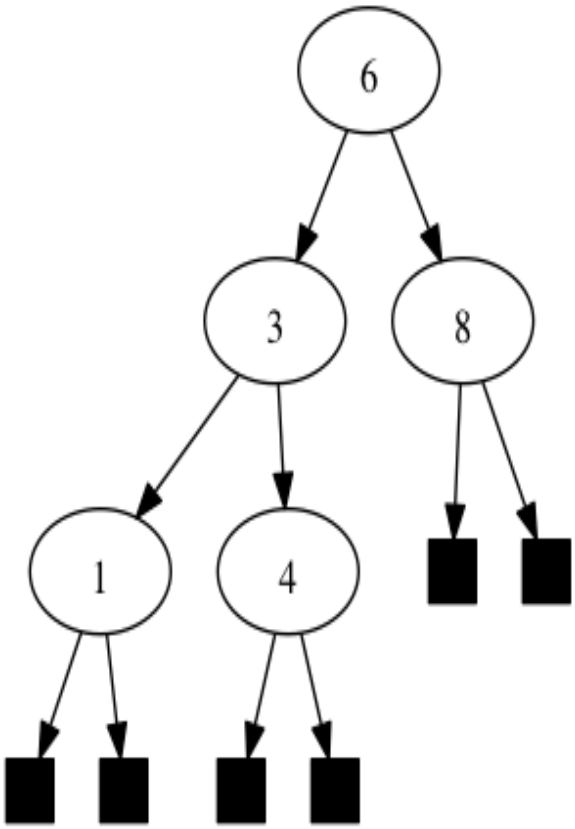
  **Hint**: if the `find()` method returns an internal node when looking for `element`, it means that the tree contains it.

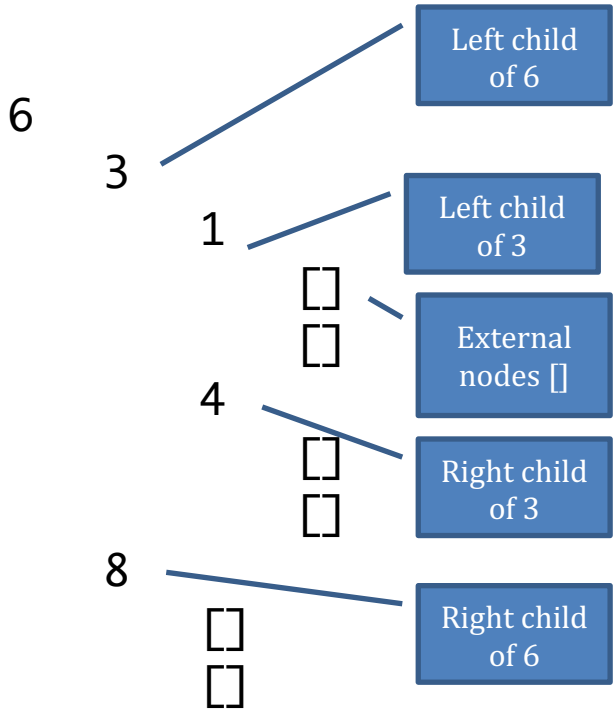
- `public void remove( T element )`: Remove element from the `tree`. Remember that the process to remove an element can change depending on whether the node's children are internal or external.

  **Hint**: Once you have found the correct node to remove from the tree, you can use the remove() method from `ProperLinkedBinaryTree` to do the removal for you. Remember, this may not be the node that originally contained `element`.

## Explaining the Output

The `main()` method in the `Main` class performs some operations and outputs a simple representation of a Binary Search Tree (from left-to-right instead of from top-to-bottom). An example of the kind of output this produces is below.



# Expected Output

After you have implemented all the methods of the Binary Search Tree, the output of running the main() method in the Main class should be as follows:

```
FIRST TREE----------------------------------------
23
      12
            1
                  []
                  7
                        []
                        []
            13
                  []
                  22
                        18
                              []
                              []
                        []
      44
            43
                  []
                  []
            55
                  []
                  []
SECOND TREE----------------------------------------
23
      12
            7
                  []
                  []
            13
                  []
                  22
                        18
                              []
                              []
                        []
      44
            43
                  []
                  []
            55
                  []
                  []
```

```
THIRD TREE----------------------------------------
43
      12
            7
                  []
                  []
            13
                  []
                  22
                        18
                              []
                              []
                        []
      44
            []
            55
                  []
                  []
```