A. Introduction

This laboratory exercise required the design and implementation of a tree interface, a binary search tree node and a binary search tree.

Helper methods were created to interact with the object of these classes and interface.

B. Design Approach

The lab required the creation of an abstract Tree interface class with works with generic classes with functions like insert, contain, empty to interact with classes that implement this interface.

A BinaryNode class was created to create class that produces node objects for classes that implement Tree interface.

A BinaryTree abstract class was created to implement some required methods of the Tree interface. A BinarySearchTree class was created and it inherits from the BinaryTree class and implements the rest of the methods of BinaryTree class.

The final class is the SimulationController which accepts integer arguments and places them into a BinarySearchTree<Integer> object and then prints out details such as if an integer was added to the tree or not, number of elements at each depth, maximum element and minimum element in the BinarySearchTree<Integer> object.

C. Method

Three arguments were selected for this simulation run.

1. 5,3,2,4,7,6,8 since it gives a balanced tree
2. 6,5,4,7,7,5,4 since it has recurring objects. It will be interesting to see if objects are added to tree or prevented
3. 2,3,4,5,6,7,8 since it is expected a linear-like tree is produced and it will be interesting to see the number of elements at each depth.

D. Data & Analysis

1. 5,3,2,4,7,6,8 insertion

"5","3","2","4","7","6","8"

Output:

```
Inserting values
============================
```

```
Insert 5: Insertion completion: true
Insert 3: Insertion completion: true
Insert 2: Insertion completion: true
Insert 4: Insertion completion: true
Insert 7: Insertion completion: true
Insert 6: Insertion completion: true
Insert 8: Insertion completion: true

Print tree
==============================

In-order String: [2, 3, 4, 5, 6, 7, 8]
Post-order String: [2, 4, 3, 6, 8, 7, 5]
Pre-order String: [5, 3, 2, 4, 7, 6, 8]

Largest element in tree
==============================
8

Smallest element in tree
==============================
2

Number of elements at each depth
==============================
Number of elements at depth 2: 4
Number of elements at depth 1: 2
Number of elements at depth 0: 1

Empty tree
==============================
Tree emptied
Is tree empty? true
In-order String: []
Post-order String: []
Pre-order String: []
```

2. 6,5,4,7,7,5,4 insertion

```
Inserting values
==============================

Insert 6: Insertion completion: true
Insert 5: Insertion completion: true
Insert 4: Insertion completion: true
Insert 7: Insertion completion: true
Insert 7: Insertion completion: false
```

```
Insert 5: Insertion completion: false
Insert 4: Insertion completion: false

Print tree
==============================

In-order String: [4, 5, 6, 7]
Post-order String: [4, 5, 7, 6]
Pre-order String: [6, 5, 4, 7]

Largest element in tree
==============================
7

Smallest element in tree
==============================
4

Number of elements at each depth
==============================
Number of elements at depth 2: 1
Number of elements at depth 1: 2
Number of elements at depth 0: 1

Empty tree
==============================
Tree emptied
Is tree empty? true
In-order String: []
Post-order String: []
Pre-order String: []
```

3. 2,3,4,5,6,7,8 insertion

```
Inserting values
==============================

Insert 2: Insertion completion: true
Insert 3: Insertion completion: true
Insert 4: Insertion completion: true
Insert 5: Insertion completion: true
Insert 6: Insertion completion: true
Insert 7: Insertion completion: true
Insert 8: Insertion completion: true

Print tree
==============================
```

```
In-order String: [2, 3, 4, 5, 6, 7, 8]
Post-order String: [8, 7, 6, 5, 4, 3, 2]
Pre-order String: [2, 3, 4, 5, 6, 7, 8]

Largest element in tree
==============================
8

Smallest element in tree
==============================
2

Number of elements at each depth
==============================
Number of elements at depth 6: 1
Number of elements at depth 5: 1
Number of elements at depth 4: 1
Number of elements at depth 3: 1
Number of elements at depth 2: 1
Number of elements at depth 1: 1
Number of elements at depth 0: 1

Empty tree
==============================
Tree emptied
Is tree empty? true
In-order String: []
Post-order String: []
Pre-order String: []
```

The SimulationController gave as expected outputs.

For case 1, a perfect tree was expected and it was obtained with 4 nodes at the lowest depth, decreasing by half as tree is traversed from bottom up.

For case 2, objects were prevented from re-entry, thus preventing duplicate elements in the tree, as the lab instructed.

For case 3, objects are inserted to produce a linear-like tree, with an element at each depth.

E. Conclusion

BinarySearchTree class created behaves and performs as lab instructions required.

F. References

Lysecky, R. V., & Lysecky, F. V. (2013). *Data Structures and Algorithms*. Los Gatos, CA. Zyante Inc.

Oracle Corporation. (2018, October 6). Java API String. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/lang/String.html.