Homework 9

Ramzy Izza Wardhana - 21/472698/PA/20322

Note: The method printTree() is located at the end of BinarySearchTree Class

BinarySearchNode.java

```java
public class BinarySearchNode {
    Integer data;
    BinarySearchNode left;
    BinarySearchNode right;

    BinarySearchNode(Integer data){
        this.data = data;
        this.left = null;
        this.right = null;
    }

    public String toString() {
        return "[" + data + ", " + left + ", " + right + "]";
    }
}
```

BinarySearchTree.java

```java
public class BinarySearchTree {
    private BinarySearchNode root;
    public String label;
    public BinarySearchTree left;
    public BinarySearchTree right;

    //Constructor
    BinarySearchTree() {
        root = null;
    }

    //Method for inserting data
    public void insert (Integer data) {
        BinarySearchNode p = root; //Start from the root
        BinarySearchNode parent = null; //Parent of p, Intially == null
        boolean isLeftChild = false; //True if p is the left child of parent

        while(p != null){
            int result = data.compareTo(p.data);
```

```
            if(result == 0) { //Data == p.data
                //Data already in the tree, return
                System.out.println(data + " Already Exist");
                return;
            } else if (result < 0) {
                //Write codes here
                parent = p;
                isLeftChild = true;
                p = p.left;
            } else { //Data > p.data
                parent = p;
                isLeftChild = false;
                p = p.right;
            }
        }
        //Create a new node under parent
        //Determine whether it is left or right child based on isLeftChild
        BinarySearchNode newNode = new BinarySearchNode(data);
        if (parent == null){
            root = newNode;
        } else if (isLeftChild) {
            parent.left = newNode;
        } else {
            parent.right = newNode;
        }
    }

    //Method for searching the data
    public void search(Integer data) {
        BinarySearchNode p = root; //Start from the root
        while(p != null){
            int result = data.compareTo(p.data); //Compare data with p.data
            if (result == 0) { //Data == p.data
                //Data Found
                System.out.println(data + " is found");
                return;
            } else if(result < 0) { //data < p.data
                //Proceed to the left child
                p = p.left;
            } else { //Data > p.data
                //Proceed to the right child
                //Write the code here
```

```
                p = p.right;
            }
        }
        //Data is not found
        System.out.println(data + " is not Found");
    }

    public void delete(Integer data) {
        BinarySearchNode p = root; //Start from the root
        BinarySearchNode parent = null; //Parent of p, initially == null
        boolean isLeftChild = false; //True if p is the left child of parent
        while (p != null) {
            int result = data.compareTo(p.data); //Data == p.data
            if (result == 0) { //Found the data
                if(p. left == null && p.right == null) {//P is external node
                    if(parent == null) {
                        root = null;
                    } else if (isLeftChild) {
                        parent.left = null;
                        break;
                    } else {
                        parent.right = null;
                        break;
                    }
                } else if (p.left == null) { //p only has the right subtree
                    //Replace with the right child
                    //Write codes here
                    if (parent == null) {
                        root = p.right;
                    } else if (isLeftChild){
                        parent.left = p.right;
                        break;
                    } else {
                        parent.right = p.right;
                        break;
                    }
                } else if (p.right == null) {//p only has the left subtree
                    if(parent == null) {
                        root = p.left;
                    } else if (isLeftChild){
                        parent.left = p.left;
                        break;
```

```
                } else {
                    parent.right = p.left;
                    break;
                }

            } else { //P has both right and left subtrees
                //Find the smallest node from the right subtree
                BinarySearchNode x = findMin(p);
                //Replace p with x
                if (parent == null) {
                    root = x;
                } else if(isLeftChild) {
                    parent.left = x;
                } else{
                    parent.right = x;
                }
                x.right = p.right;
                x.left = p.left;
                p.right = null;
                p.left = null;
            }
        } else if (result < 0) { //data < p.data
            parent = p;
            isLeftChild = true;
            p = p.left;
        } else { //data > p.data
            parent = p;
            isLeftChild = false;
            p = p.right;
        }
    }
    //Data has been succesfully deleted
    System.out.println(data + " Has been deleted");
    //Data is not found
    System.out.println(data + " Is not found");
}

//Method for finding the smallest node of the right subtree
public BinarySearchNode findMin(BinarySearchNode parent) {
    BinarySearchNode p = parent.right;
    //Traverse to the leftmost of this subtree to fin the smallest node
    while (p.left != null) {
```

```
            p = p.left;
        }
        return p; //Return the smallest node
    }

    public String toString(){
        return inOrder(root);
    }

    private String inOrder(BinarySearchNode p){
        if (p == null)
            return "";
        return inOrder(p.left) + "" + p.data + " " + inOrder(p.right);
    }

    //Method for outputting the binary tree
    public void printTree() {
        System.out.println("\nResult of Binary Tree (Left [root] to Right
[Child]): ");
        printTree(root, "");
    }
    private void printTree(BinarySearchNode node, String indent) {
        if (node == null) {
            return;
        }
        printTree(node.right, indent + "\t");
        //Root
        if (indent.isEmpty()) {
            System.out.println("   " + "[" + node.data + "]");
        } //Child with edge
        else {
            System.out.println(indent + "|--- [" + node.data + "]");
        }
        printTree(node.left, indent + "\t");
    }
}
```

TestBinarySearchTree.java

```
public class TestBinarySearchTree {
    public static void main(String[] args) {
```

```
        //Create a BST
        BinarySearchTree bst = new BinarySearchTree();
        bst.insert(5);
        bst.insert(3);
        bst.insert(8);
        bst.insert(2);
        bst.insert(4);
        bst.insert(6);
        bst.insert(7);
        bst.printTree();

        // //Search data from BST
        System.out.println("\nData Searching on Binary Tree: ");
        bst.search(2);
        bst.search(9);

        // //Delete data from the BST
        System.out.println("\nData Deleting on Binary Tree: ");
        bst.delete(8);
        bst.printTree();
        bst.delete(6);
        bst.printTree();
        bst.delete(7);
        bst.printTree();
        bst.delete(2);
        bst.printTree();
    }
}
```

Output

```
PS C:\Users\themi\Downloads\java-prak-asd>  c:; cd 'c:\Us
ws\java-prak-asd_a7071df7\bin' 'TestBinarySearchTree'

Result of Binary Tree (Left [root] to Right [Child]):
        |--- [8]
                        |--- [7]
                |--- [6]
  [5]
                |--- [4]
        |--- [3]
                |--- [2]

Data Searching on Binary Tree:
2 is found
9 is not Found

Data Deleting on Binary Tree:
8 Has been deleted
8 Is not found

Result of Binary Tree (Left [root] to Right [Child]):
                |--- [7]
        |--- [6]
  [5]
                |--- [4]
        |--- [3]
                |--- [2]
6 Has been deleted
6 Is not found

Result of Binary Tree (Left [root] to Right [Child]):
        |--- [7]
  [5]
                |--- [4]
        |--- [3]
                |--- [2]
7 Has been deleted
7 Is not found

Result of Binary Tree (Left [root] to Right [Child]):
  [5]
                |--- [4]
        |--- [3]
                |--- [2]
2 Has been deleted
2 Is not found

Result of Binary Tree (Left [root] to Right [Child]):
  [5]
                |--- [4]
        |--- [3]
```