Activity 9.1

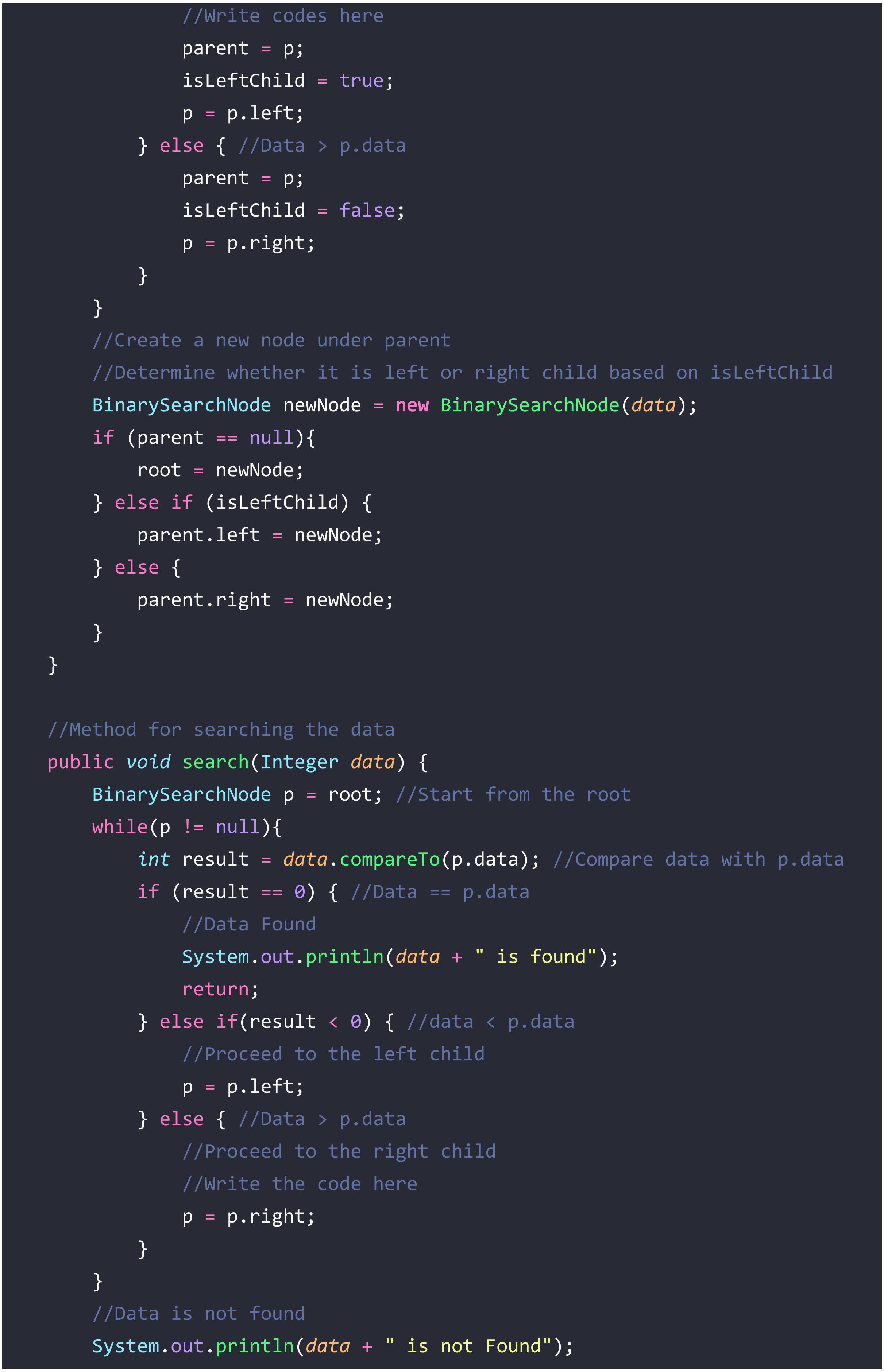
Ramzy Izza Wardhana - 21/472698/PA/20322

BinarySearchNode.java

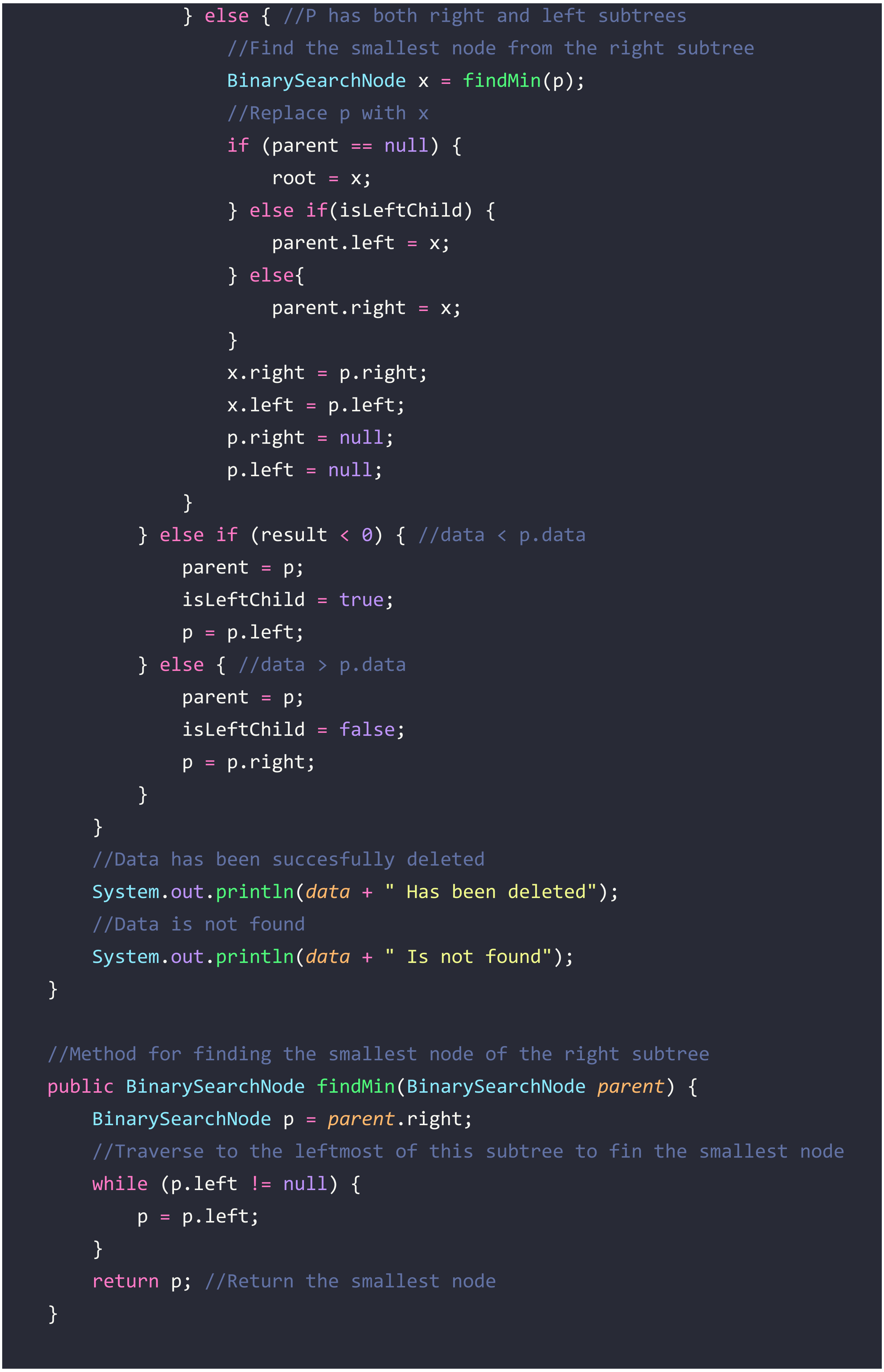
```java
public class BinarySearchNode {
    Integer data;
    BinarySearchNode left;
    BinarySearchNode right;

    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;

    //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;
                    }
```

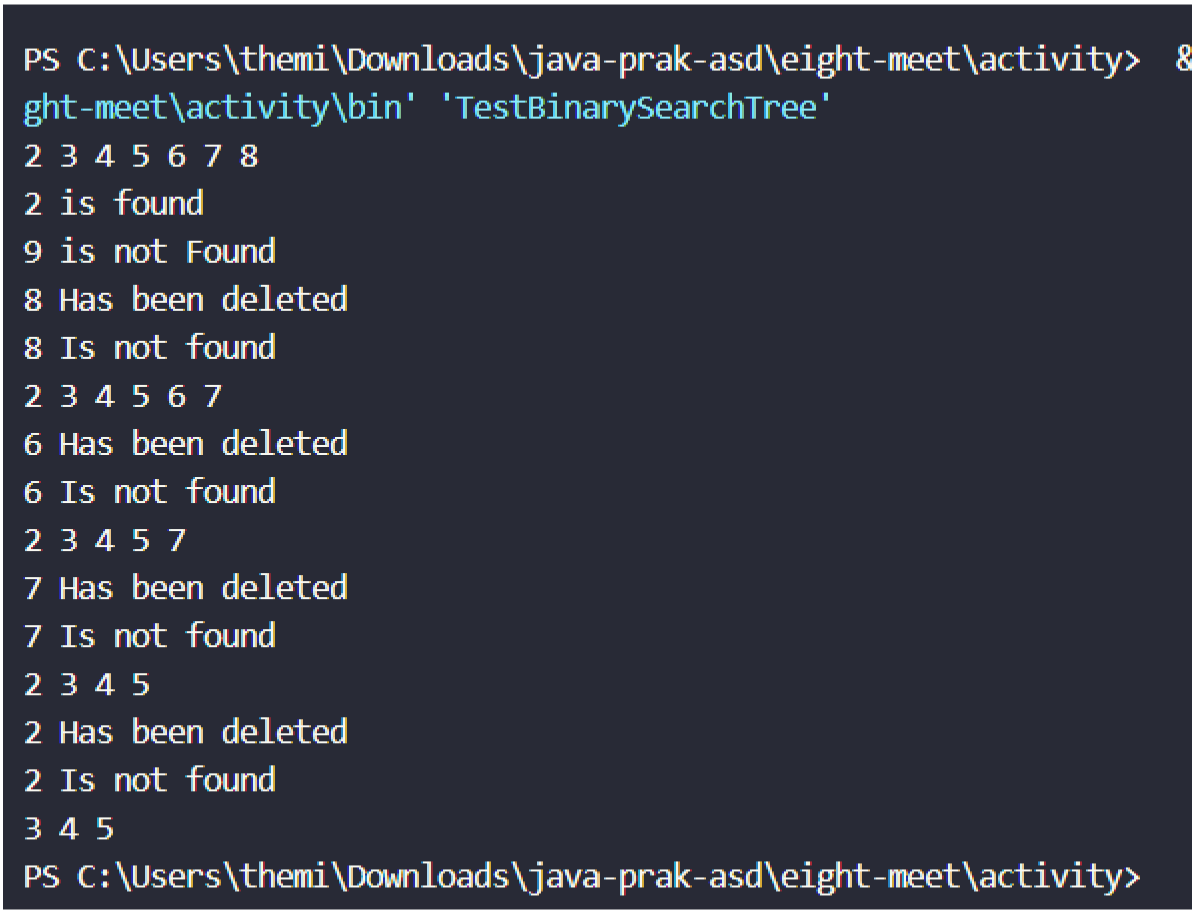


```
                } else { //P has both right and left subtrees
                    //Find the smallest node from the right subtree
                    BinarySearchNode x = findMin(p);
                    //Replace p with x
                    if (parent == null) {
                        root = x;
                    } 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);
    }
}
```

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);
        System.out.println(bst);

        //Search data from BST
        bst.search(2);
        bst.search(9);

        //Delete data from the BST
        bst.delete(8);
        System.out.println(bst);
        bst.delete(6);
        System.out.println(bst);
        bst.delete(7);
        System.out.println(bst);
        bst.delete(2);
        System.out.println(bst);
    }
}
```

Output



```
PS C:\Users\themi\Downloads\java-prak-asd\eight-meet\activity>  &
ght-meet\activity\bin' 'TestBinarySearchTree'
2 3 4 5 6 7 8
2 is found
9 is not Found
8 Has been deleted
8 Is not found
2 3 4 5 6 7
6 Has been deleted
6 Is not found
2 3 4 5 7
7 Has been deleted
7 Is not found
2 3 4 5
2 Has been deleted
2 Is not found
3 4 5
PS C:\Users\themi\Downloads\java-prak-asd\eight-meet\activity>
```