Ahmad Siddiq Priaji (22/496854/PA/21370)

**BinarySearchNode.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**

public class BinarySearchTree {

    private BinarySearchNode root;

    //constructor

    BinarySearchTree() {

        root = null;

    }

    //methods for inserting a data

    public void insert(Integer data) {

        BinarySearchNode p = root; //start from the root

        BinarySearchNode parent = null; //parent of p, initially=null

        boolean isLeftChild = false; //true id 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 exists");

                return;

            } 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;

            }

        }

        //Crate 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;

        }

    }

    //methods for searching a data

    public void search(Integer data) {

        BinarySearchNode p = root; //start from the root

        while (p != null) {

            int result = data.compareTo(p.data); //compare data and p.data

            if (result == 0) { //data == p.data

                System.out.println(data + " is found");

                return;

            } else if (result < 0) { //data < p.data

                //proccess the left child

                p = p.left;

            } else { //data > p.data

                //proccess the right child

                p = p.right;

            }

        }

        //data is not found

        System.out.println(data + " is not found");

    }

    //methods for deleting a data

    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 an external node

                    if (parent == null) root = null;

                    else if (isLeftChild) parent.left = null;

                    else parent.right = null;

                } else if (p.left == null) { //p only has right subtree

                    //replace with the right child

                    if (parent == null) root = p.right;

                    else if (isLeftChild) parent.left = p.right;

                    else parent.right = p.right;

                } else if (p.right == null) { //p only has left subtree

                    //replace with the left child

                    if (parent == null) root = p.left;

                    else if (isLeftChild) parent.left = p.left;

                    else parent.right = p.left;

                } else { //p has both right and left subtrees

                    //find the smallest node from the right subtree

                    BinarySearchNode x = findMin(p);

                    //replace p with x 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;

                }

                //data has been succesfully deleted

                System.out.println(data + " has been deleted");

                return;

            } 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 is not found

         System.out.println(data + " is not found");

    }

    //methods for finding the smallest node of a right subtree

    private BinarySearchNode findMin(BinarySearchNode parent) {

        BinarySearchNode p = parent.right;

        //traverse to the leftmost node of this subtree to find the smallest node

        while (p.left != null) {

            p = p.left;

        }

        return p;

    }

    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 the 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);

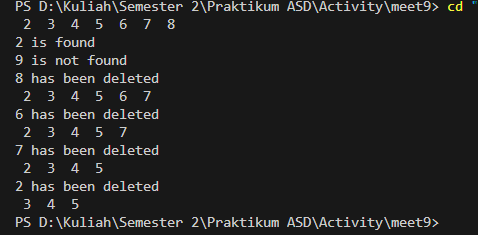
        System.out.println(bst);

        bst.delete(2);

        System.out.println(bst);

    }

}

**Hasil**