public class  Main{

    public static void main(String[] args) {

        Tree<Character> tree = new Tree<>();

        char[] pohon = {'F','E','H','D','H','C','B','H','K','J'};

        for(char BTS : pohon){

            System.out.printf("%s",BTS);

            tree.insertNode(BTS);

        }

        char huruf1 = 'K';

        char huruf2 = 'A';

        System.out.println();

        System.out.println("Huruf yang ingin dicari adalah "+huruf1+" dan "+huruf2);

        tree.searchBST('A');

        tree.searchBST('K');

    }

}

1. Class Main ini adalah class yg akan mengeksekusi fungsi yg ada yg telah disediakan di class Tree dan . Di kelas **Main**, program dimulai dengan pembuatan instance dari kelas **Tree** yang akan menyimpan elemen bertipe **Character**. Sebuah array karakter bernama **pohon** didefinisikan dengan nilai-nilai tertentu. Kemudian, setiap karakter dalam array tersebut dimasukkan ke dalam pohon menggunakan metode **insertNode** dari kelas **Tree**. Setelah itu, program mencoba mencari dua karakter ('K' dan 'A') dalam pohon menggunakan metode **searchBST**.

class Tree<E extends Comparable<E>> {

    private TreeNode<E> root;

    // constructor initializes an empty Tree of integers

    public Tree() {

        root = null;

    }

    // insert a new node in the binary search tree

    public void insertNode(E insertValue) {

        if (root == null) {

            root = new TreeNode<E>(insertValue); // create root node

        } else {

            root.insert(insertValue); // call the insert method

        }

    }

    // begin preorder traversal

    public void preorderTraversal() {

        preorderHelper(root);

    }

    private void preorderHelper(TreeNode<E> node) {

        if (node == null) {

            return;

        }

        System.out.printf("%s ", node.getData()); // output node data

        preorderHelper(node.getLeftNode()); // traverse left subtree

        preorderHelper(node.getRightNode()); // traverse right subtree

    }

    // begin inorder traversal

    public void inorderTraversal() {

        inorderHelper(root);

    }

    // recursive method to perform inorder traversal

    private void inorderHelper(TreeNode<E> node) {

        if (node == null) {

            return;

        }

        inorderHelper(node.getLeftNode()); // traverse left subtree

        System.out.printf("%s ", node.getData()); // output node data

        inorderHelper(node.getRightNode()); // traverse right subtree

    }

    // begin postorder traversal

    public void postorderTraversal() {

        postorderHelper(root);

    }

    // recursive method to perform postorder traversal

    private void postorderHelper(TreeNode<E> node) {

        if (node == null) {

            return;

        }

        postorderHelper(node.getLeftNode()); // traverse left subtree

        postorderHelper(node.getRightNode()); // traverse right subtree

        System.out.printf("%s ", node.getData()); // output node data

    }

    public void searchBST(E key) {

        boolean hasil = searchBSTHelper(root, key);

        if (hasil)

            System.out.println("Huruf "+ key +" ditemukan");

        else

            System.out.println("Huruf "+key+" tidak ditemukan");

    }

    // Carrano

    public boolean searchBSTHelper(TreeNode<E> node, E key) {

        boolean result = false;

        if (node != null) {

            if (key.equals(node.getData()))

                result = true;

            else if (key.compareTo(node.getData()) < 0)

                result = searchBSTHelper(node.getLeftNode(), key);

            else

                result = searchBSTHelper(node.getRightNode(), key);

        }

        return result;

    }

}

1. Class Tree

Kelas **Tree** bertanggung jawab untuk mengelola operasi pada pohon. Atribut utama dari kelas ini adalah **root**, yang merupakan referensi ke node akar dari pohon. Konstruktor **Tree** menginisialisasi **root** sebagai **null**, menandakan bahwa pohon kosong pada awalnya. Metode **insertNode** digunakan untuk menambahkan node baru ke dalam pohon. Jika pohon masih kosong (**root** masih **null**), node baru akan menjadi akar. Jika tidak, metode **insert** pada objek **TreeNode** digunakan untuk menemukan posisi yang tepat bagi node baru. Kelas ini juga menyediakan metode untuk traversal pohon dalam tiga cara: preorder, inorder, dan postorder. Setiap metode traversal ini memiliki metode rekursif helper yang digunakan untuk menelusuri pohon dan mencetak data dari setiap node. Selain itu, kelas **Tree** juga memiliki metode **searchBST** yang digunakan untuk mencari nilai tertentu dalam pohon. Metode ini menggunakan metode rekursif **searchBSTHelper** untuk melakukan pencarian, dan akan mencetak pesan yang sesuai berdasarkan hasil pencarian.

class TreeNode<E extends Comparable<E>> {

    private TreeNode<E> leftNode;

    private E data; // node value

    private TreeNode<E> rightNode;

    // constructor initializes data and makes this a leaf node

    public TreeNode(E nodeData) {

        data = nodeData;

        leftNode = rightNode = null; // node has no children

    }

    public E getData() {

        return data;

    }

    public TreeNode<E> getLeftNode() {

        return leftNode;

    }

    public TreeNode<E> getRightNode() {

        return rightNode;

    }

    public void insert(E insertValue) {

        // insert in left subtree

        if (insertValue.compareTo(data) < 0) {

            // insert new TreeNode

            if (leftNode == null) {

                leftNode = new TreeNode<E>(insertValue);

            } else { // continue traversing left subtree recursively

                leftNode.insert(insertValue);

            }

        }

        // insert in right subtree

        else if (insertValue.compareTo(data) > 0) {

            // insert new TreeNode

            if (rightNode == null) {

                rightNode = new TreeNode<E>(insertValue);

            } else { // continue traversing right subtree recursively

                rightNode.insert(insertValue);

            }

        }

    }

}

1. Class TreeNode

Kelas **TreeNode** merepresentasikan setiap node dalam pohon. Kelas ini memiliki atribut untuk menyimpan data node serta referensi ke anak kiri (**leftNode**) dan anak kanan (**rightNode**). Konstruktor **TreeNode** menginisialisasi node sebagai daun (leaf node) dengan anak kiri dan kanan diatur ke **null**. Metode **insert** digunakan untuk menambahkan nilai baru ke dalam subtree yang sesuai. Jika nilai baru lebih kecil dari data node saat ini, maka akan dimasukkan ke subtree kiri. Jika lebih besar, maka akan dimasukkan ke subtree kanan. Jika subtree yang sesuai masih kosong (**null**), node baru akan dibuat di sana.