LAPORAN PRAKTIKUM
STRUKTUR DATA NON LINIER
MODUL 2

Dosen Pengampu

JB. Budi Darmawan S.T., M.Sc.

DISUSUN OLEH

Yohanis Celvin D. P. U. Pati

245314033

PROGRAM STUDI INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS SANATA DHARMA

YOGYAKARTA

2025

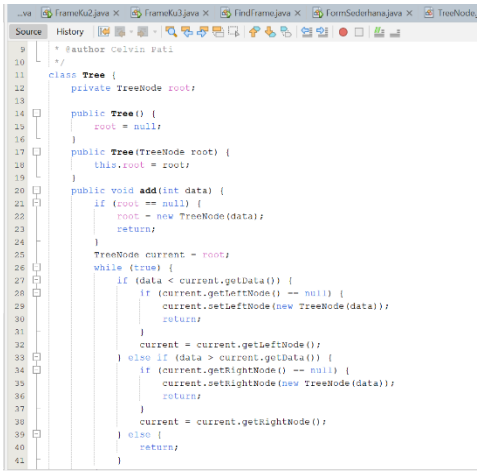
A. DIAGRAM UML

TreeNode		
-	data	: int
-	leftNode	: TreeNode
-	rightNode	: TreeNode
+	TreeNode ()	: Constuctor
+	TreeNode (int)	: Constuctor
+	getData()	: int
+	setData(int)	: void
+	getLeftNode()	: TreeNode
+	setLeftNode(TreeNode)	: void
+	getRightNode()	: TreeNode
+	setRightNode(TreeNode)	: void

Tree		
–	root	: TreeNode
+	Tree()	: Constructor
+	Tree (TreeNode)	: Constructor
+	add(int)	: void
+	getNode(int)	: TreeNode
+	getRoot()	: TreeNode
+	setRoot(TreeNode)	: void

B. SOURCE CODE

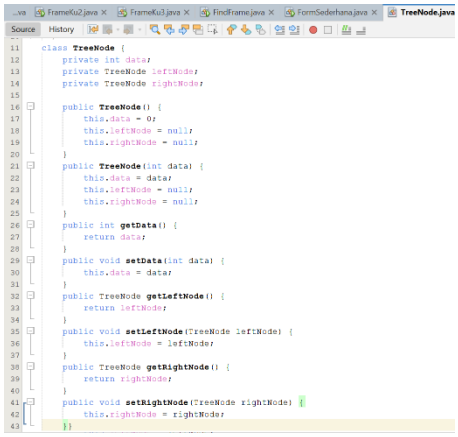
- **Tree**



```
 * @author Calvin Pati
 */
class Tree {
    private TreeNode root;

    public Tree() {
        root = null;
    }
    public Tree(TreeNode root) {
        this.root = root;
    }
    public void add(int data) {
        if (root == null) {
            root = new TreeNode(data);
            return;
        }
        TreeNode current = root;
        while (true) {
            if (data < current.getData()) {
                if (current.getLeftNode() == null) {
                    current.setLeftNode(new TreeNode(data));
                    return;
                }
                current = current.getLeftNode();
            } else if (data > current.getData()) {
                if (current.getRightNode() == null) {
                    current.setRightNode(new TreeNode(data));
                    return;
                }
                current = current.getRightNode();
            } else {
                return;
            }
```

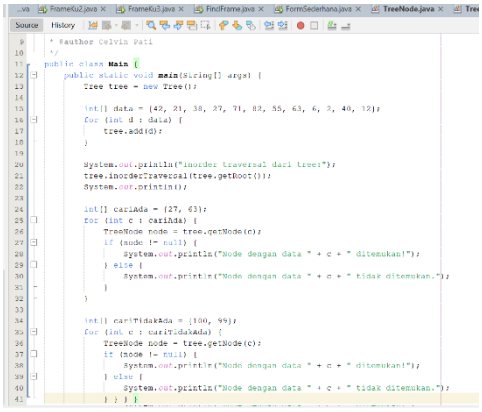
```
        }
    }
    public TreeNode getNode(int data) {
        TreeNode current = root;
        while (current != null) {
            if (data == current.getData()) {
                return current;
            } else if (data < current.getData()) {
                current = current.getLeftNode();
            } else {
                current = current.getRightNode();
            }
        }
        return null;
    }
    public TreeNode getRoot() {
        return root;
    }
    public void setRoot(TreeNode root) {
        this.root = root;
    }
    public void inorderTraversal(TreeNode node) {
        if (node != null) {
            inorderTraversal(node.getLeftNode());
            System.out.print(node.getData() + " ");
            inorderTraversal(node.getRightNode());
        }
    }
}
```

- **TreeNode**



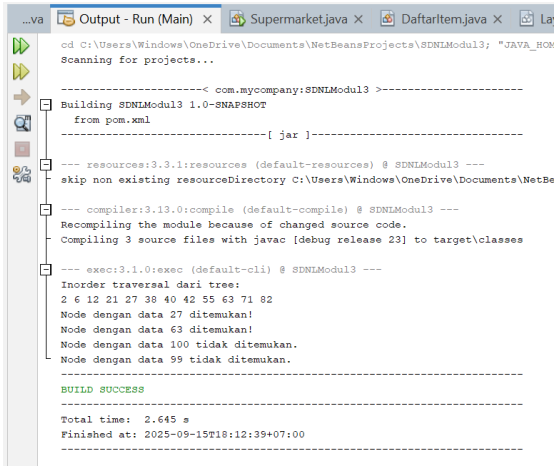
```
class TreeNode {
    private int data;
    private TreeNode leftNode;
    private TreeNode rightNode;

    public TreeNode() {
        this.data = 0;
        this.leftNode = null;
        this.rightNode = null;
    }
    public TreeNode(int data) {
        this.data = data;
        this.leftNode = null;
        this.rightNode = null;
    }
    public int getData() {
        return data;
    }
    public void setData(int data) {
        this.data = data;
    }
    public TreeNode getLeftNode() {
        return leftNode;
    }
    public void setLeftNode(TreeNode leftNode) {
        this.leftNode = leftNode;
    }
    public TreeNode getRightNode() {
        return rightNode;
    }
    public void setRightNode(TreeNode rightNode) {
        this.rightNode = rightNode;
    }
```

- **Main**



```
/**
 * @author Celvin Pati
 */
public class Main {
    public static void main(String[] args) {
        Tree tree = new Tree();

        int[] data = {42, 21, 38, 27, 71, 82, 55, 63, 6, 3, 40, 12};
        for (int d : data) {
            tree.add(d);
        }

        System.out.println("Inorder traversal dari tree:");
        tree.inorderTraversal(tree.getRoot());
        System.out.println();

        int[] cariAda = {27, 63};
        for (int c : cariAda) {
            TreeNode node = tree.getNode(c);
            if (node != null) {
                System.out.println("Node dengan data " + c + " ditemukan!");
            } else {
                System.out.println("Node dengan data " + c + " tidak ditemukan.");
            }
        }

        int[] cariTidakAda = {100, 99};
        for (int c : cariTidakAda) {
            TreeNode node = tree.getNode(c);
            if (node != null) {
                System.out.println("Node dengan data " + c + " ditemukan!");
            } else {
                System.out.println("Node dengan data " + c + " tidak ditemukan.");
            } } }
```

C. OUTPUT



```
...va | Output - Run (Main) | Supermarket.java | DaftarItem.java | La
cd C:\Users\Windows\OneDrive\Documents\NetBeansProjects\SDNLModul3; "JAVA_HOM
Scanning for projects...

----------------------< com.mycompany:SDNLModul3 >----------------------
Building SDNLModul3 1.0-SNAPSHOT
  from pom.xml
--------------------------------[ jar ]---------------------------------

--- resources:3.3.1:resources (default-resources) @ SDNLModul3 ---
skip non existing resourceDirectory C:\Users\Windows\OneDrive\Documents\NetBe

--- compiler:3.13.0:compile (default-compile) @ SDNLModul3 ---
Recompiling the module because of changed source code.
Compiling 3 source files with javac [debug release 23] to target\classes

--- exec:3.1.0:exec (default-cli) @ SDNLModul3 ---
Inorder traversal dari tree:
2 6 12 21 27 38 40 42 55 63 71 82
Node dengan data 27 ditemukan!
Node dengan data 63 ditemukan!
Node dengan data 100 tidak ditemukan.
Node dengan data 99 tidak ditemukan.
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.645 s
Finished at: 2025-09-15T18:12:39+07:00
------------------------------------------------------------------------
```

D. ANALISA

1. **Kelas TreeNode**
    - **Konstruktor TreeNode(int data)**

      Method ini berfungsi untuk membuat node baru dengan nilai data, dan leftNode & rightNode akan atomatis diinisialisasikan null.
    - **getData() / setData(int data)**

      kedua method ini berfungsi untuk mengaambil daan juga bisa mengubah nilai data dari sebuah node.
    - **getLeftNode() / setLeftNode(TreeNode leftNode)**

      method ini juga berfungsi sebagai akses atau setter untuk subtree kiri.
    - **getRightNode() / setRightNode(TreeNode rightNode)**

      method ini juga sama berfungsi sebagai akses/setter tapi untuk subtree kanan.
    - **Konsep Tree**

      Jadi kaitannya dengan konsep Tree yaitu setiap node bisa punya maksimal 2 anak: left dan right, sesuai dengan definisi Binary Tree.
2. **Kelas Tree**
    - **Konstruktor Tree()**

      Method ini berfungsi untuk membuat Tree kosong dengan root = null.
    - Konstruktor Tree(TreeNode root)

      Metod ini juga berfungsi untuk membuat tree tapi dengan root yang sudah ditentukan.
    - **add(int data)**

      Method ini berfungsi untuk menambah node bar uke tree sesuai aturan Binary Search Tree:
        - jika data lebih kecil maka masuk ke subtree kiri.
        - jika lebih besar maka masuk ke subtree kanan.
        - jika sama maka diabaikan (tidak boleh duplikat).
    - **getNode(int data)**

      Method ini berfungsi untuk mencari node dengan nilai tertentu. Jadi Traversal dilakukan dari root ke bawah dengan mengikuti aturan BST.

- **getRoot() / setRoot(TreeNode root)**

  Method ini berfungsi untuk mengambil atau mengubah node root pada tree.
- **inorderTraversal(TreeNode node)**

  Method ini berfungsi untuk melakukan traversal inorder (kiri -> root -> kanan). Pada BST, dan hasilnya akan urut dari kecil ke besar.
- **Konsep Tree**

  Jadi hubungan kelas ini dengan konsep Tree yaitu kelas ini mengimplementasikan strutur Binary Search Tree, dengan operasi dasar: insert, search, traesal.

3. **Kelas Main**
   - **Tree tree = new Tree();**

     Method ini berfungsi untuk membuat tree baru yang masih Korong.
   - **Loop tree.add(d);**

     Method ini berfungsi untuk menambah kumpulan data ke dalam tree dengan mengikuti aturan BST.
   - **tree.inorderTraversal(tree.getRoot());**

     Method ini berfungsi untuk menampilkan isi tree dengan cara urut atau ascending.
   - **Pencarian node (getNode)**

     Method ini berfungsi untuk mengecek apakah data tertentu ada di dalam tree atau tidak, lalu mencetak hasilnya.
   - **Konsep Tree**

     Jadi kaitan nya kelas main ini dengan konsep Tree yaitu mengimplementasi nyata dari tree, dengan menunjukan cara data disusun secara hierarki dan bisa ditelusuri dengan mudah.

E. REFERENSI