

Binary Search Tree (BST)

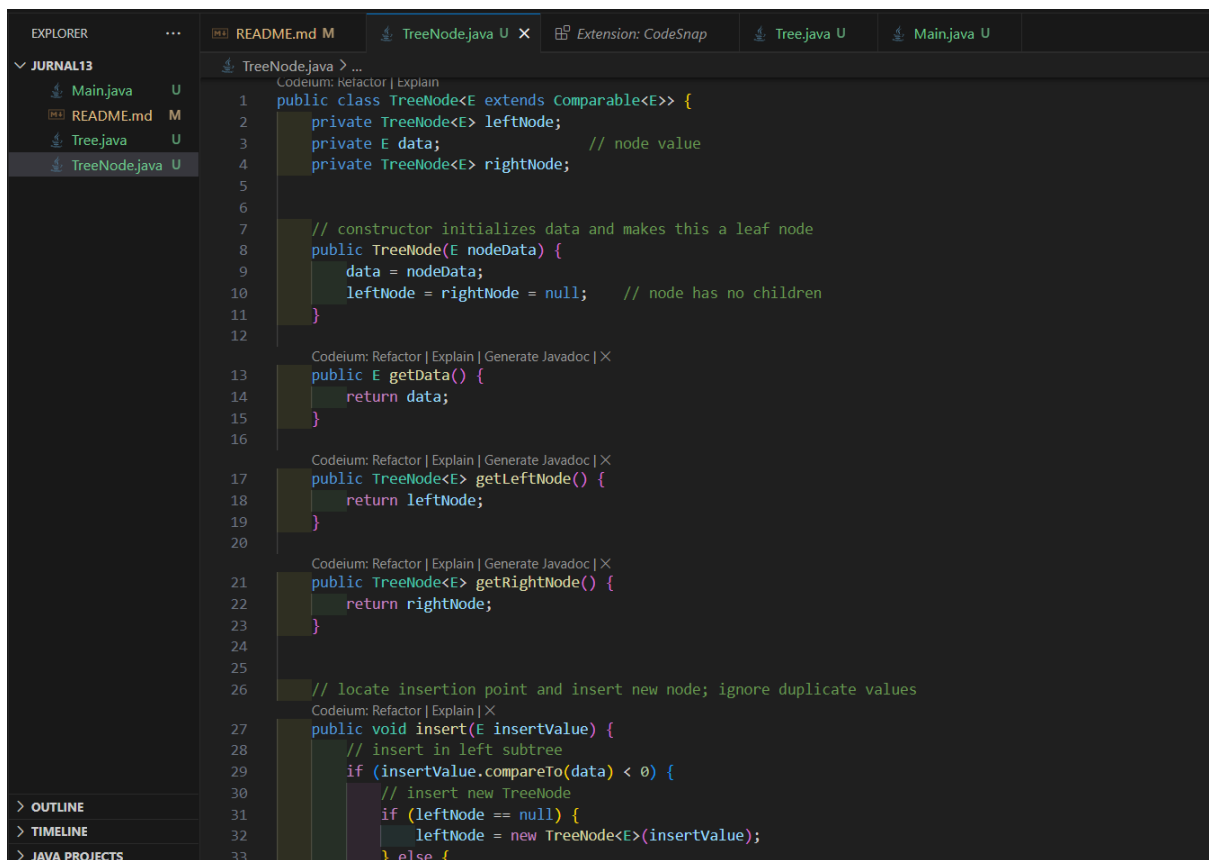
Penjelasan Jurnal 13

Nama: Arya Meidi Abiyasa

NIM: 607062300073

Kelas: D3IF-47-04

TreeNode.java



```
1 public class TreeNode<E> extends Comparable<E>> {
2     private TreeNode<E> leftNode;
3     private E data; // node value
4     private TreeNode<E> rightNode;
5
6
7     // constructor initializes data and makes this a leaf node
8     public TreeNode(E nodeData) {
9         data = nodeData;
10        leftNode = rightNode = null; // node has no children
11    }
12
13    Codeium: Refactor | Explain | Generate Javadoc | X
14    public E getData() {
15        return data;
16    }
17
18    Codeium: Refactor | Explain | Generate Javadoc | X
19    public TreeNode<E> getLeftNode() {
20        return leftNode;
21    }
22
23    Codeium: Refactor | Explain | Generate Javadoc | X
24    public TreeNode<E> getRightNode() {
25        return rightNode;
26    }
27
28    // locate insertion point and insert new node; ignore duplicate values
29    Codeium: Refactor | Explain | X
30    public void insert(E insertValue) {
31        // insert in left subtree
32        if (insertValue.compareTo(data) < 0) {
33            // insert new TreeNode
34            if (leftNode == null) {
35                leftNode = new TreeNode<E>(insertValue);
36            } else {
37                leftNode.insert(insertValue);
38            }
39        } else if (insertValue.compareTo(data) > 0) {
40            // insert in right subtree
41            if (rightNode == null) {
42                rightNode = new TreeNode<E>(insertValue);
43            } else {
44                rightNode.insert(insertValue);
45            }
46        }
47    }
48 }
```

```
▼ JURNAL13
  Main.java U
  README.md M
  Tree.java U
  TreeNode.java U

TreeNode.java > ...
1  public class TreeNode<E extends Comparable<E>> {
27  public void insert(E insertValue) {
34      leftNode.insert(insertValue); // continue traversing left subtree recursively
35  }
36  }
37  // insert in right subtree
38  else if (insertValue.compareTo(data) > 0) {
39      // insert new TreeNode
40      if (rightNode == null) {
41          rightNode = new TreeNode<E>(insertValue);
42      } else {
43          rightNode.insert(insertValue); // continue traversing right subtree recursively
44      }
45  }
46  // else if (insertValue.compareTo(data) == 0) {
47  //     // insert new TreeNode
48  //     if (leftNode == null) {
49  //         leftNode = new TreeNode<E>(insertValue);
50  //     } else {
51  //         leftNode.insert(insertValue); // continue traversing left subtree recursively
52  //     }
53  // }
54  else { // eklenecek nodeun değeri root'un kiyle aynıysa sola gönder
55      // insert new TreeNode
56      if (leftNode == null) {
57          leftNode = new TreeNode<E>(insertValue);
58      } else {
59          leftNode.insert(insertValue); // continue traversing left subtree recursively
60      }
61  }
62  }
63  }

64  @Override
65  public String toString() {
66      return "TreeNode [leftNode=" + leftNode + ", data=" + data + ", rightNode=" + rightNode + "];"
67  }
68  }
69  }
```

Kelas `TreeNode` adalah struktur dasar dari sebuah binary search tree (BST) yang memungkinkan penyimpanan data secara terstruktur dengan cara yang memudahkan pencarian dan penambahan data baru.

Setiap `TreeNode` dapat memiliki referensi ke dua anak, yaitu `leftNode` dan `rightNode`, serta menyimpan nilai data (`data`) yang merupakan tipe generik `E`.

Konstruktor kelas ini menginisialisasi node sebagai leaf node (node tanpa anak) dengan data yang diberikan.

Metode `insert` digunakan untuk menambahkan elemen baru ke dalam BST. Jika nilai yang dimasukkan lebih kecil dari nilai pada node saat ini, elemen baru akan ditempatkan di subtree kiri; jika lebih besar, di subtree kanan; dan jika sama, kode yang dikomentari menunjukkan bahwa elemen tersebut akan ditempatkan di subtree kiri.

Kelas ini menggunakan generic type `E` yang harus bisa dibandingkan, memungkinkan `TreeNode` untuk bekerja dengan berbagai tipe data selama tipe tersebut mengimplementasikan interface `Comparable`.

Secara keseluruhan, kelas `TreeNode` ini memberikan fungsi-fungsi dasar yang diperlukan untuk membangun dan mengelola binary search tree. Kode ini dapat digunakan sebagai fondasi untuk struktur data yang lebih kompleks atau aplikasi yang memerlukan penyimpanan dan pengambilan data yang efisien.

Tree.java

```

JURNAL13
  Main.java U
  README.md M
  Tree.java U
  TreeNode.java U

Tree.java > ...
Codeium: Refactor | Explain
1 public class Tree<E extends Comparable<E>> {
2     private TreeNode<E> root;
3
4     // constructor initializes an empty tree of integers
5     public Tree() {
6         root = null;
7     }
8
9     // insert a new node into the binary search tree
10    Codeium: Refactor | Explain | X
11    public void insertNode(E insertValue) {
12        System.out.print(insertValue + " "); // additional method
13        if (root == null) {
14            root = new TreeNode<E>(insertValue); // create root node
15            // System.out.print(insertValue + " "); // ilave method | if it is put in this if statement it will print
16        } else {
17            root.insert(insertValue);
18            // System.out.print(insertValue + " ");
19        }
20    }
21
22    // =====
23    // recursive method to perform preorder traversal
24    Codeium: Refactor | Explain | X
25    private void preorderHelper(TreeNode<E> node) {
26        if (node == null) {
27            return; // exit the method
28        }
29
30        System.out.printf(format: "%s ", node.getData()); // output node data
31        preorderHelper(node.getLeftNode()); // traverse left subtree
32        preorderHelper(node.getRightNode()); // traverse right subtree
33    }
34    // begin preorder traversal
35
36    Tree.java > ...
37    public class Tree<E extends Comparable<E>> {
38
39        public void preorderTraversal() {
40            preorderHelper(root);
41        }
42
43        // =====
44
45        // recursive method to perform inorder traversal
46        Codeium: Refactor | Explain | X
47        private void inorderHelper(TreeNode<E> node) {
48            if (node == null) {
49                return;
50            }
51
52            inorderHelper(node.getLeftNode()); // traverse left subtree
53            System.out.printf(format: "%s ", node.getData()); // output node data
54            inorderHelper(node.getRightNode()); // traverse right subtree
55        }
56
57        // begin inorder traversal
58        Codeium: Refactor | Explain | X
59        public void inorderTraversal() {
60            inorderHelper(root);
61        }
62
63        // =====
64
65        // recursive method to perform postorder traversal
66        Codeium: Refactor | Explain | X
67        private void postorderHelper(TreeNode<E> node) {
68            if (node == null) {
69                return;
70            }
71
72            postorderHelper(node.getLeftNode()); // traverse left subtree
73            postorderHelper(node.getRightNode()); // traverse right subtree
74            System.out.printf(format: "%s ", node.getData()); // output node data
75        }
76    }
77
78    > OUTLINE
79    > TIMELINE
80    > JAVA PROJECTS
```

```
1 public class Tree<E extends Comparable<E>> {
68 }
69
70 // begin postorder traversal
71 Codeium: Refactor | Explain | X
72 public void postorderTraversal() {
73     postorderHelper(root);
74 }
75
76 // =====
77
78 // carrano
79 Codeium: Refactor | Explain | X
80 private boolean searchBSTHelper(TreeNode<E> node, E key) {
81     boolean result = false;
82
83     if (node != null) {
84         if (key.equals(node.getData())) {
85             result = true;
86         } else if (key.compareTo(node.getData()) < 0) {
87             result = searchBSTHelper(node.getLeftNode(), key);
88         } else {
89             result = searchBSTHelper(node.getRightNode(), key);
90         }
91     }
92     return result;
93 }
94
95 Codeium: Refactor | Explain | Generate Javadoc | X
96 public void searchBST(E key) {
97     boolean hasil = searchBSTHelper(root, key);
98
99     if (hasil) {
100         System.out.println("Data " + key + " ditemukan pada tree");
101     } else {
102         System.out.println("Data " + key + " tidak ditemukan pada tree");
103     }
104 }
```

Kelas Tree merupakan implementasi dari binary search tree (BST) yang memungkinkan penyimpanan dan pengelolaan data secara terurut dan efisien.

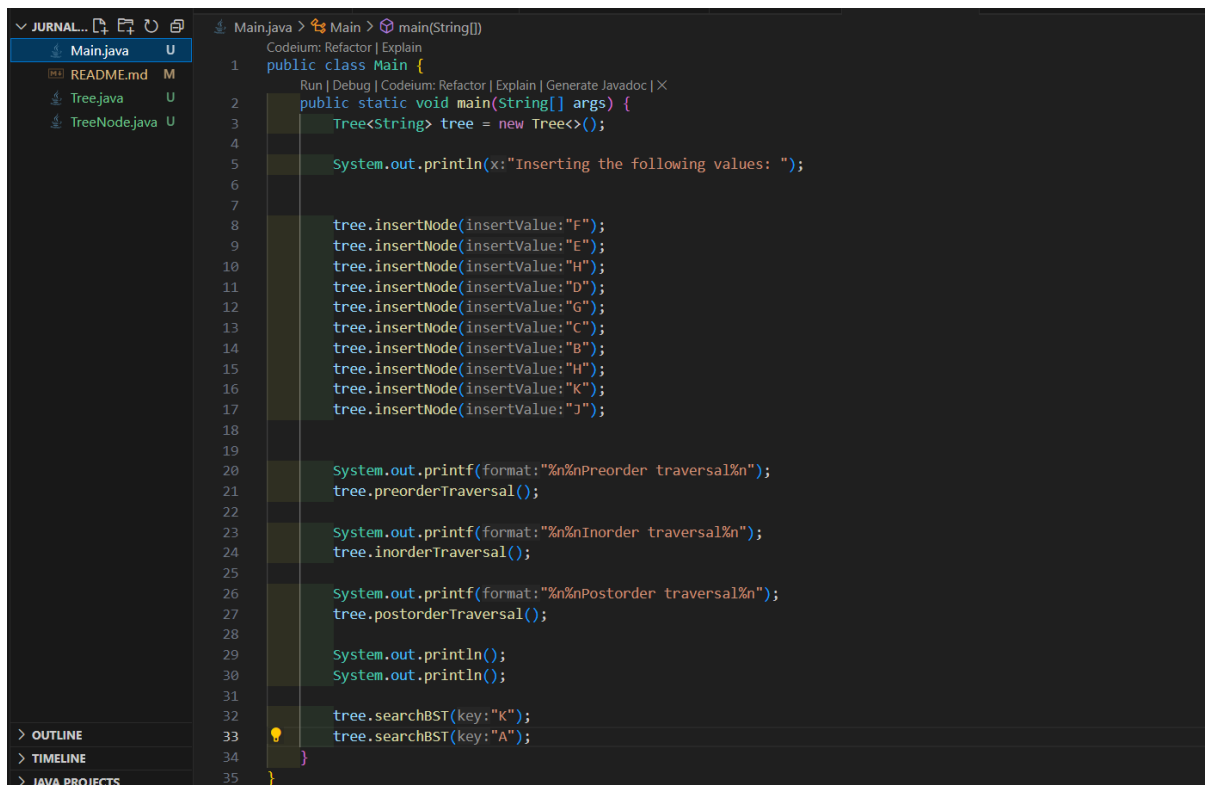
Tree memiliki metode insertNode untuk menambahkan elemen baru, dan metode traversal preorderTraversal, inorderTraversal, dan postorderTraversal untuk mengunjungi elemen-elemen dalam urutan tertentu.

Metode searchBST digunakan untuk mencari elemen tertentu dalam BST dan memberikan umpan balik apakah elemen tersebut ada atau tidak dalam struktur pohon.

Kelas ini menggunakan generic type E yang harus mengimplementasikan interface Comparable untuk memastikan elemen dapat dibandingkan dan diurutkan.

Kelas Tree ini menyediakan fungsionalitas dasar yang diperlukan untuk struktur data BST, termasuk penambahan, pencarian, dan traversal elemen. Ini berguna dalam berbagai aplikasi yang memerlukan penyimpanan data dengan cara yang memudahkan pencarian dan akses terurut.

Main.java



```
1 public class Main {
2     public static void main(String[] args) {
3         Tree<String> tree = new Tree<>();
4
5         System.out.println(x:"Inserting the following values: ");
6
7
8         tree.insertNode(insertValue:"F");
9         tree.insertNode(insertValue:"E");
10        tree.insertNode(insertValue:"H");
11        tree.insertNode(insertValue:"D");
12        tree.insertNode(insertValue:"G");
13        tree.insertNode(insertValue:"C");
14        tree.insertNode(insertValue:"B");
15        tree.insertNode(insertValue:"H");
16        tree.insertNode(insertValue:"K");
17        tree.insertNode(insertValue:"J");
18
19
20        System.out.printf(format:"%n%nPreorder traversal%n");
21        tree.preorderTraversal();
22
23        System.out.printf(format:"%n%nInorder traversal%n");
24        tree.inorderTraversal();
25
26        System.out.printf(format:"%n%nPostorder traversal%n");
27        tree.postorderTraversal();
28
29        System.out.println();
30        System.out.println();
31
32        tree.searchBST(key:"K");
33        tree.searchBST(key:"A");
34    }
35 }
```

Kelas Main merupakan titik masuk program yang menggunakan kelas Tree untuk membuat binary search tree (BST) dengan tipe data String.

Program ini menunjukkan bagaimana menambahkan elemen ke dalam BST, melakukan traversal preorder, inorder, dan postorder, serta mencari elemen tertentu dalam BST.

Output yang dihasilkan mencakup daftar elemen yang ditambahkan, representasi visual dari traversal BST, dan hasil pencarian elemen "K" dan "A".

Contoh ini menggambarkan penggunaan praktis dari BST dalam menyimpan dan mengatur data secara teratur, serta menunjukkan bagaimana melakukan operasi dasar pada struktur data tersebut.

Program ini memberikan contoh yang baik tentang bagaimana struktur data seperti BST dapat digunakan dalam aplikasi nyata untuk memanipulasi dan menampilkan data secara efisien. Jika ada pertanyaan lebih lanjut atau butuh bantuan dengan bagian tertentu, silakan bertanya!