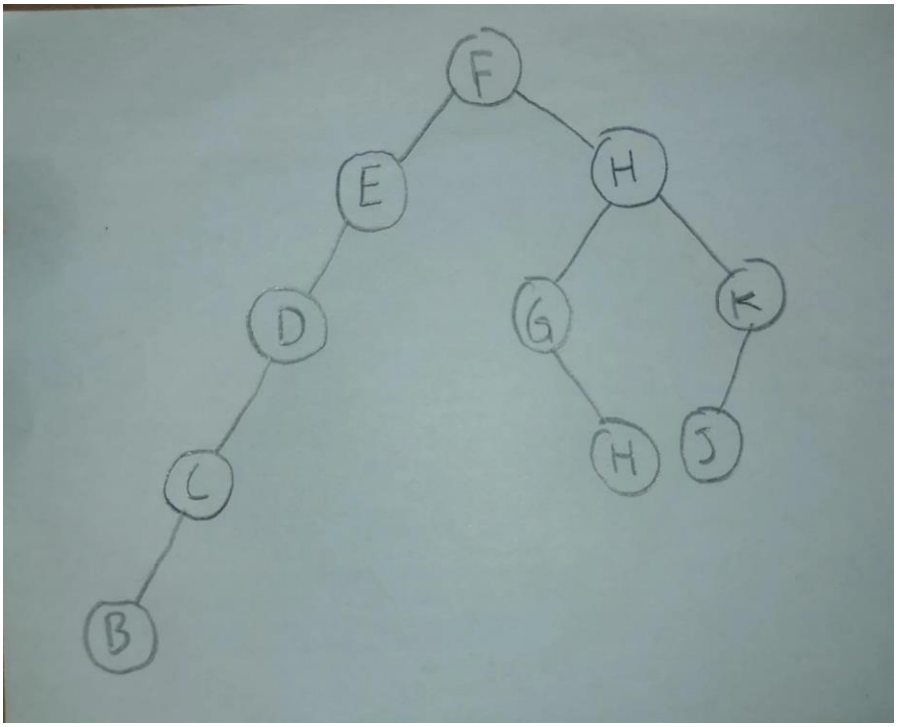
Nama: Angel Levyne

Kelas: D3IF-47-02

NIM: 607062330045

Tugas: Jurnal 13 ISD

Gambaran Binary Seacrh Tree(BST)



Pre-Order : F E D C B H G H K J

In-Order : B C D E F G H H J K

Postorder : B C D E H G J K H F

Penjelasan Coding

Class TreeNode.java

```
public class TreeNode<E extends Comparable<E>> {
```

Kelas publik TreeNode yang menggunakan parameter generik **E**. Parameter E dibatasi dengan **Comparable<E>,** yang berarti tipe **E** harus bisa dibandingkan (pengimplementasian antarmuka **Comparable**).

```
    private TreeNode<E> leftNode;
    private E data;
    private TreeNode<E> rightNode;
```

- **leftNode**: sebuah referensi ke anak kiri dari tipe **TreeNode<E>.**
- **data**: elemen data yang disimpan dalam node, bertipe **E**.
- **rightNode**: sebuah referensi ke anak kanan dari tipe **TreeNode<E>.**

```
    public TreeNode(E nodeData) {
        data = nodeData;
        leftNode = rightNode = null;
    }
```

Bagian ini mendefinisikan konstruktor untuk kelas **TreeNode**, konstruktor ini mengambil argumen **nodeData** yang merupakan data yang akan disimpan dalam node. Di dalam konstruktor:

- **data** diinisialisasi dengan nilai **nodeData**
- **leftNode** dan **rightNode** diinisialisasi dengan null, yang berarti pada awalnya, node ini tidak memiliki anak kiri maupun anak kanan

```
    public E getData() {
        return data;
    }
```

Metode ini merupakan getter yang mengembalikan **data** yang disimpan dalam node saat ini.

```
    public TreeNode<E> getLeftNode() {
        return leftNode;
    }
```
Getter yang mengembalikan referensi ke anak kiri dari node saat ini.

```
    public TreeNode<E> getRightNode() {
        return rightNode;
    }
```
Getter yang mengembalikan referensi ke anak kanan dari node saat ini.

```
    public void insert(E insertValue) {
        if (insertValue.compareTo(data) < 0) {
            if (leftNode == null) {
                leftNode = new TreeNode<E>(insertValue);
            } else {
                leftNode.insert(insertValue);
            }
        }
        else if (insertValue.compareTo(data) > 0) {
            if (rightNode == null) {
                rightNode = new TreeNode<E>(insertValue);
            } else {
                rightNode.insert(insertValue);
            }
        }
        else {
            if (leftNode == null) {
                leftNode = new TreeNode<E>(insertValue);
            } else {
                leftNode.insert(insertValue);
            }
        }
    }
```
Memasukkan nilai baru **(insertValue)** ke dalam tree.

1. Pertama, nilai yang akan dimasukkan **(insertValue)** dibandingkan dengan data dalam node saat ini menggunakan metode **compareTo**.
    - Jika **insertValue** < data (hasil **compareTo** kurang dari 0):
    - Jika **leftNode** masih **null**, maka node baru dengan **insertValue** dibuat dan ditetapkan sebagai **leftNode**.

- Jika **leftNode** tidak **null**, maka metode **insert** dipanggil secara rekursif pada leftNode.

2. Jika **insertValue** >(hasil **compareTo** lebih dari 0):
   - Jika **rightNode** masih **null**, maka node baru dengan **insertValue** dibuat dan ditetapkan sebagai **rightNode**.
   - Jika **rightNode** tidak **null**, maka metode insert dipanggil secara rekursif pada **rightNode**.
3. Jika **insertValue** == data (hasil **compareTo** sama dengan 0):
   - Perilakunya di sini mirip dengan jika **insertValue** lebih kecil dari data, yakni mencoba memasukkannya ke **leftNode**.

```
    @Override
    public String toString() {
        return "TreeNode [leftNode=" + leftNode + ", data=" + data + ",
rightNode=" + rightNode + "]";
    }
}
```

Menimpa (**override**) metode **toString** dari class **Object**. Metode ini memberikan bentuk string dari objek **TreeNode**, yang mencakup informasi tentang **leftNode**, **data**, dan **rightNode**.

Class Tree.java

```
public class Tree<E extends Comparable<E>> {
    private TreeNode<E> root;

    public Tree() {
        root = null;
    }
```

Class public **Tree** yang menggunakan parameter generik **E**, yang harus merupakan turunan dari **Comparable<E>.** Kelas ini memiliki satu variabel instance **root**, yang merupakan referensi ke node akar dari tipe **TreeNode<E>**.

```
    public void insertNode(E insertValue) {
        System.out.print(insertValue + " ");
        if (root == null) {
            root = new TreeNode<E>(insertValue);
        } else {
            root.insert(insertValue);
        }
    }
```

Memasukkan nilai baru (**insertValue**) ke dalam tree. Pertama mencetak nilai yang akan dimasukkan. Jika **root** masih **null**, maka node baru dengan **insertValue** dibuat dan ditetapkan sebagai **root**. Jika **root** sudah ada, metode **insert** pada **root** dipanggil untuk memasukkan nilai baru ke dalam tree.

```
    private void preorderHelper(TreeNode<E> node) {
        if (node == null) {
            return;
        }

        System.out.printf("%s ", node.getData());
        preorderHelper(node.getLeftNode());
        preorderHelper(node.getRightNode());
    }


    public void preorderTraversal() {
        preorderHelper(root);
    }
```

Melakukan traversal preorder pada pohon, di mana setiap node dikunjungi sebelum anak-anaknya. Metode **preorderTraversal** memulai traversal dari **root**, sedangkan

preorderHelper adalah metode rekursif yang mengunjungi node saat ini, kemudian anak kiri, dan terakhir anak kanan.

```
    private void inorderHelper(TreeNode<E> node) {
        if (node == null) {
            return;
        }

        inorderHelper(node.getLeftNode());
        System.out.printf("%s ", node.getData());
        inorderHelper(node.getRightNode());
    }


    public void inorderTraversal() {
        inorderHelper(root);
    }
```

Melakukan traversal inorder pada tree, di mana setiap node dikunjungi setelah anak kirinya dan sebelum anak kanannya. **inorderTraversal** memulai traversal dari **root**, sedangkan **inorderHelper** adalah metode rekursif yang mengunjungi anak kiri, kemudian node saat ini, dan terakhir anak kanan.

```
    private void postorderHelper(TreeNode<E> node) {
        if (node == null) {
            return;
        }

        postorderHelper(node.getLeftNode());
        postorderHelper(node.getRightNode());
        System.out.printf("%s ", node.getData());
    }

    public void postorderTraversal() {
        postorderHelper(root);
    }
```

Melakukan traversal postorder pada tree, di mana setiap node dikunjungi setelah anak-anaknya. **postorderTraversal** memulai traversal dari **root**, sedangkan **postorderHelper** adalah metode rekursif yang mengunjungi anak kiri, kemudian anak kanan, dan terakhir node saat ini.

```
    private 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;
    }

    public void searchBST(E key) {
        boolean hasil = searchBSTHelper(root, key);

        if (hasil) {
            System.out.println("Data " + key + " ditemukan");
        } else {
            System.out.println("Data "+ key + " tidak ditemukan");
        }
    }
}
```

Digunakan untuk mencari elemen (**key**) dalam Tree Binary. **searchBST** memulai pencarian dari **root** dengan memanggil metode **searchBSTHelper**. **searchBSTHelper** adalah metode rekursif yang membandingkan **key** dengan **data** dalam node saat ini:

- Jika == maka result diset menjadi true.
- Jika **key** < seacrh dilanjutkan ke anak kiri.
- Jika **key** > seacrh dilanjutkan ke anak kanan.

Hasil search (**true** atau **false**) dikembalikan dan ditampilkan oleh **searchBST**.

Class Main.java

```
public class Main {
    public static void main(String[] args) {
```

Merupakan class **Main** dan metode **main** yang merupakan titik masuk program.

```
        Tree<String> tree = new Tree<>();
```

Membuat sebuah objek **tree** dari class **Tree.java** dengan tipe data **String**.

```
        System.out.println("Inserting the following values: ");


        tree.insertNode("F");
        tree.insertNode("E");
        tree.insertNode("H");
        tree.insertNode("D");
        tree.insertNode("G");
        tree.insertNode("C");
        tree.insertNode("B");
        tree.insertNode("H");
        tree.insertNode("K");
        tree.insertNode("J");
```

Mencetak "**Inserting the following values:** " lalu beberapa nilai dimasukkan ke dalam tree menggunakan metode **insertNode**. Nilai yang dimasukkan adalah **"F", "E", "H", "D", "G", "C", "B", "H", "K", dan "J".**

```
        System.out.printf("%n%nPreorder traversal%n");
        tree.preorderTraversal();

        System.out.printf("%n%nInorder traversal%n");
        tree.inorderTraversal();

        System.out.printf("%n%nPostorder traversal%n");
        tree.postorderTraversal();
```

Mencetak "**Preorder traversal**", "**Inorder traversal**" , "**Postorder traversal**" , kemudian memanggil masing-masing metode **preorderTraversal**, **inorderTraversal** dan **PostorderTraversal** pada objek **tree** untuk melakukan masing-masing traversal dan mencetaknya.

```
        System.out.println();
        System.out.println();

        tree.searchBST("K");
        tree.searchBST("A");

    }
}
```

Disini mencetak dua baris kosong untuk pemisah.Lalu, metode **searchBST** dipanggil untuk mencari nilai "**K**" dan "**A**" dalam tree. Hasil seacrh ditampilkan dan menunjukkan apakah nilai tersebut ditemukan atau tidak dalam tree.