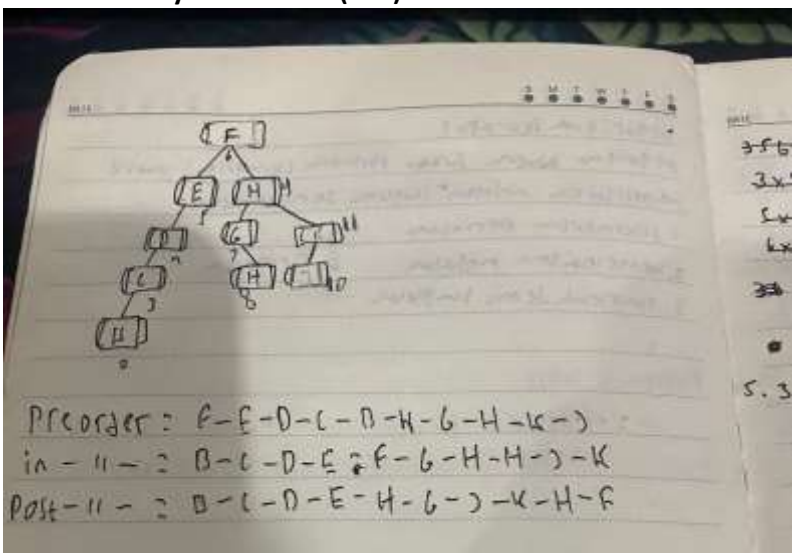
Gambar Binary Search Tree (BST)



Kodingan & Penjelasan

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

    //Atribut private
    private TreeNode<E> leftNode;
    private E data;
    private TreeNode<E> rightNode;

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

Kelas TreeNode mewakili sebuah node dalam struktur data pohon (Tree). Kelas ini memiliki tiga atribut private: leftNode yaitu referensi ke node anak kiri dari node ini, Data: nilai yang disimpan dalam node ini yang dapat dibandingkan dengan nilai lain (menggunakan interface Comparable), dan rightNode: referensi ke node anak dari node ini.

Kelas ini juga memiliki konstruktor yang digunakan untuk membuat instance TreeNode baru. Konstruktor ini memiliki parameter nodeData yang merupakan nilai yang disimpan pada node ini. Saat konstruktor ini dipanggil, atribut data disetel ke nilai nodeData, dan atribut leftNode dan rightNode diinisialisasi dengan nilai null. Artinya node ini tidak mempunyai anak kiri dankanan

```
//Getter
    public E getData() {    //Untuk mendapatkan data node saat ini
        return data;
    }

    public TreeNode<E> getLeftNode() {  //Untuk mendapatkan referensi ke node
kiri
        return leftNode;
    }
```

```
    public TreeNode<E> getRightNode() { //Untuk mendapatkan referensi ke node
kanan
        return rightNode;
    }
```

Kode diatas adalah metode pengambil kelas TreeNode. Getter adalah metode mengambil nilai suatu atribut. Kelas TreeNode ini memiliki tiga metode penggambil: getData(): metode ini digunakan untuk mengambil nilai atribut data dari node saat ini. getLeftNode(): metode ini digunakan untuk mendapatkan referensi ke node anak kiri dari node saat ini, getRightNode(): metode ini digunakan untuk mendapatkan referensi ke node anak kanan dari node saat ini.

Selain itu, setiap metode getter memiliki tipe data kembalian yang sesuai drngan atribut yang diterimanya, yaitu E untuk getData() dan TreeNode untuk getLeftNode() dan getRightNode(). Hal ini penting agar bisa mendapatkan jenis data yang tepat sesuat kebutuhan

```
//Metode insert --> untuk memasukkan nilai baru ke dalam pohon biner
    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);
            }
        }
    }
```

Kode diatas adalah metode insert kelas TreeNode. Metode ini memasukkan nilai bar uke dalam pohon biner. Pertama, metode insert menggunakan metode compareTo membandingkan nilai yang akan dimasukkan (insertValue) dengan nilai node (data) saat ini. Jika insertValue lebih kecil dari data, metode akan memeriksa apakah leftNode sudah ada. Jika tidak, simpul kiri baru akan dibuat dengan nilai insertValue. Jika sudah ada, metode insert dipanggil lagi pada node kiri dan nilai insertValue dimasukkan ke dalam subtree kiri. Jika

insertValue lebih besar dari data, metode akan memeriksa apakah node yang benar sudah ada. Jika tidak, node kanan baru akan dibuat dengan nilai insertValue. Jika sudah ada, metode insert dipanggil lagi pada node kanan dan nilai insertValue dimasukkan ke dalam subtree kanan. Jika insertValue adalah data, metode akan menyisipkan nilai insertValue ke node kiri. Hal ini karena dalam pohon biner, nilai yang sama ditempatkan pada simpul kiri

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

Kode di atas merupakan implementasi metode toString() dari kelas TreeNode. Metode ini mengubah objek kelas TreeNode menjadi representasi string. Metode toString() ini menggunakan format string yang terdiri dari tiga bagian: leftNode: Representasi string dari node kiri dari node saat ini. Data: Sebuah string yang mewakili nilai dari node saat ini.rightNode: Representasi string dari node kanan dari node saat ini. Setiap bagian yang diapit tanda kurung siku ([ ]) menghasilkan representasi string dari atribut terkait. Untuk mendapatkan representasi string dari objek TreeNode, metode toString() kelas TreeNode dipanggil secara rekursif pada node kiri dan kanan

Binary Tree

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

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

public class Tree {: Kelas Tree didefinisikan dengan parameter tipe E dan harus mengimplementasikan antarmuka Comparable. Artinya tipe E harus sebanding dengan dirinya sendiri. Contoh tipe E yang bisa digunakan adalah Integer, String, atau tipe lainnya yang mengimplementasikan antarmuka Comparable. private TreeNode root;: Atribut root adalah node yang mewakili akar pohon biner. Atribut ini dinyatakan private, artinya hanya dapat diakses di dalam kelas Tree itu sendiri. Atribut root bertipe TreeNode. Artinya node ini dapat menyimpan nilai bertipe E. public Tree() {: Konstruktor untuk kelas Tree telah ditentukan. Konstruktor ini tidak memiliki parameter dan hanya menginisialisasi atribut root dengan nilai null. Artinya pada saat objek Tree dibuat, pohon binernya masih kosong dan tidak berisi node

```
 //Metode insertNode --> Menambahkan nilai (interValue) ke dalam tree
 //Jika root masih null, buat node baru dan dijadikan root
 //Jika root tidak null, panggil metode insert pada root + nilai keposisi
sesuai
    public void insertNode(E insertValue) {
        System.out.print(insertValue + " ");
```

```
        if (root == null) {
            root = new TreeNode<E>(insertValue);
        } else {
            root.insert(insertValue);
        }
    }
```

Kode di atas merupakan implementasi metode insertNode dari kelas Tree. Metode ini menyisipkan node baru ke dalam pohon biner. Deskripsi kodenya adalah sebagai berikut: public void insertNode(E insertValue) {: Metode insertNode didefinisikan dengan parameter tipe E yang disebut insertValue. Metode ini tidak mengembalikan nilai dan hanya digunakan untuk memasukkan node baru ke dalam pohon biner. System.out.print(insertValue + " ");: Sebelum memasukkan node baru, nilai yang akan dimasukkan dicetak ke layar menggunakan metode System.out.print. metode System.out.print ini digunakan untuk mencetak urutan proses yang mengisi node di pohon biner. if (root == null) {: Jika atribut root dari pohon biner masih kosong, maka node baru akan dibuat dan ditetapkan sebagai root dari pohon biner. root
= new TreeNode(insertValue);: Jika atribut root dari pohon biner masih kosong, node baru dibuat dan ditetapkan sebagai akar dari pohon biner. Node baru dibuat menggunakan konstruktor TreeNode dengan parameter tipe E bernama insertValue. } else {: Jika atribut root dari pohon biner sudah mempunyai nilai, node baru dimasukkan ke dalam pohon biner menggunakan metode insert pada node root. root.insert(insertValue);: Jika atribut root dari pohon biner sudah memiliki nilai, node baru dimasukkan ke dalam pohon biner menggunakan metode penyisipan node root. Metode penyisipan menyisipkan simpul baru dengan menjalankan pohon biner dari akar ke simpul yang bersangkutan

```
//Metode helper --> metode rekursif melakukan traversal preorder pada Tree
    private void preorderHelper(TreeNode<E> node) {
        if (node == null) {
            return;
        }

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

    //Melakukan metode preorder traversal
    //Root-Kiri-Kanan
    public void preorderTraversal() {
        preorderHelper(root);
    }
```

Kode di atas merupakan implementasi metode yang melakukan pre-order traversal pada pohon biner. Metode Helper: preorderHelper Metode preorderHelper adalah metode rekursif yang digunakan untuk melakukan preorder traversal pada pohon biner. Metode ini memiliki parameter node yang menentukan node yang sedang diproses. if (node== null) { return; }: Jika node saat ini adalah null, metode ini mengembalikan nilai dan tidak mengambil tindakan lebih lanjut. System.out.printf("%s ", node.getData());: Jika node saat ini bukan null, nilai node ini dicetak ke layar menggunakan System.out.printf. Nilai keluaran adalah nilai yang

disimpan dalam node ini. preorderHelper(node.getLeftNode());: Setelah mencetak nilai dari node saat ini, metode ini memanggil dirinya sendiri dengan parameter node.getLeftNode(), yang berarti node kiri dari node saat ini. Hal ini menyebabkan metode melintasi node di sebelah kiri. preorderHelper(node.getRightNode());: Setelah melintasi node kiri, metode ini memanggil dirinya sendiri dengan parameter node.getRightNode() yang berarti node di sebelah kanan node saat ini. Hal ini akan menyebabkan metode melewati node yang benar.

Metode preorderTraversal adalah metode yang digunakan untuk memulai traversal preorder pada pohon biner. preorderHelper(root);: Metode ini hanya memanggil metode preorderHelper dengan parameter root, artinya akar pohon biner. Hal ini menyebabkan metode preorderHelper melakukan traversal preorder pohon biner mulai dari root

```
//Metode helper --> metode rekursif melakukan traversal inorder pada Tree
    private void inorderHelper(TreeNode<E> node) {
        if (node == null) {
            return;
        }

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

    //Melakukan metode inorder traversal
    //Kiri-Root-kanan
    public void inorderTraversal() {
        inorderHelper(root);
    }
```

Metode inorderHelper adalah metode rekursif yang digunakan untuk melakukan traversal urutan pohon biner, dan metode inorderTraversal adalah metode yang digunakan untuk memulai traversal urutan pohon biner. Traversal inorder mengeluarkan nilai node dari kiri kekanan

```
//Metode helper --> metode rekursif melakukan traversal postorder pada Tree
    private void postorderHelper(TreeNode<E> node) {
        if (node == null) {
            return;
        }

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

    //Melakukan metode postorder traversal
    //Kiri-Kanan-Root
    public void postorderTraversal() {
        postorderHelper(root);
```

Metode postorderHelper adalah metode rekursif yang digunakan untuk melakukan traversal postorder pada pohon biner, dan metode postorderTraversal adalah metode yang digunakan untuk memulai traversal postorder pada pohon biner. Traversal postorder menghasilkan nilai simpul dalam urutan akar kiri-kanan

```
 //Metode Search (pencarian) -->metode rekursif untuk mencari nilai E pada
Tree
    //Jika node saat ini null = elemen tidak ditemukan
    //Jika key sama dengan data di node saat ini = elemen ditemukan
    //Jika key < data di node saat ini, cari di subtree kiri
    //Jika key > data di node saat ini, cari di subtree kanan
    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;
    }

    //Memulai pencarian dari root dan mencetak hasilnya apakah elemen
ditemukan atau tidak
    public void searchBST(E key) {
        boolean hasil = searchBSTHelper(root, key);

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

Metode searchBSTHelper merupakan metode rekursif untuk mencari elemen dalam pohon pencarian sekuensial biner (BST). Metode ini memiliki dua parameter.``node'' adalah node yang sedang diproses dan ``key'' adalah elemen yang akan dicari. boolean result = false;: Variabel hasil digunakan untuk menyimpan hasil pencarian. Nilai awalnya salah, artinya item tidak ditemukan. if (simpul!= null) { … }: Jika node saat ini bukan null, pencarian dilanjutkan. if (key.equals(node.getData())) { … }: Jika key sama dengan nilai node saat ini, pencarian berhasil dan hasilnya diubah menjadi true. else if (key.compareTo(node.getData()) < 0) { … }: Jika kunci lebih kecil dari nilai node saat ini, pencarian dilanjutkan dengan node di sebelah

kiri node saat ini. else { … }: Jika kunci lebih besar dari nilai node saat ini, pencarian dilanjutkan dengan node di sebelah kanan node saat ini.

Metode Metode searchBST Metode searchBST memulai pencarian dari akar pohon BST. hasil boolean = searchBSTHelper(root, key);: Metode ini hanya memanggil metode searchBSTHelper dengan parameter root dan key (artinya akar pohon BST dan elemen yang dicari). jika (hasil) { … } else { … }: Pencarian berhasil mencetak "Kunci data ditemukan di pohon". Jika tidak, jika pencarian gagal, Anda akan melihat "Kunci data tidak ditemukan di pohon.

Jadi, metode searchBSTHelper merupakan metode rekursif yang digunakan untuk mencari elemen dalam pohon biner pencarian sekuensial (BST), sedangkan metode searchBST digunakan untuk memulai pencarian dari akar metode BST dan hasilnya adalah: Menghasilkan apakah elemen ditemukan atau tidak

Main

```
public class Main {
    public static void main(String[] args) {
        Tree<String> tree = new Tree<>();

        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");


        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();

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

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

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

    }
}
```

Metode main adalah metode yang dijalankan saat program dijalankan

Output :