Jurnal 11 : Implementation of Binary Tree and Tree Traversal

Nama : Abraham Putra B. S

NIM : 607062300058

Kelas : D3IF-47-01

# Penjelasan Koding

Pada implementasi binary tree dan traversal ini, terdapat tiga file yaitu:

* **BTNode2. java** untuk pembuatan tree dalam bentuk node atau bisa dibilang **POJO**. Didalamnya juga terdapat function setter dan getter untuk menyimpan dan mengambil nilainya ke dalam node atau dari node yang left maupun right.
* **BinaryTree2.java** berisi semua fungsi yang dibutuhkan mulai dari menambahkan angka ke node, mencari angkanya/traversal, menghitung jumlah node, dan melakukan preorder,inorder, postorder.
* **Main.java**, seperti biasa, untuk halaman menu, menginput angka dan menampilkan yang dibutuhkan serta preorder, inorder dan postorder.

## BTNode2.java

    /\* Class ini menggunakan Comparable agar data yang input bisa langsung dicompare/banding\*/

public class BTNode2<*E* extends *Comparable*<*E*>> implements *Comparable*<BTNode2<*E*>>{

    BTNode2 left, right;

*E* data;

    /\* Constructor untuk data yang tidak diinput atau dibuat didalam Main\*/

    public BTNode2() {

        left = null;

        right = null;

        data = null;

    }

    /\* Constructor untuk data(generic) yang bisa diinput\*/

    public BTNode2(*E* *item*) {

        left = null;

        right = null;

        data = *item*;

    }

 @Override

/\* Fungsi ini yang akan mengcompare data yang diinput dengan data yang ada di node\*/

    public *int* compareTo(BTNode2<*E*> *o*) {

        return data.compareTo(*o*.data);

    }

    /\* Fungsi untuk menetapkan input ke node kiri \*/

    public *void* setLeft(BTNode2 *n*) {

        left = *n*;

    }

    /\* Fungsi untuk menetapkan input ke node kanan \*/

    public *void* setRight(BTNode2 *n*) {

        right = *n*;

    }

    /\* Fungsi untuk mengambil nilai node kiri\*/

    public BTNode2 getLeft() {

        return left;

    }

    /\* Fungsi untuk mengambil nilai node kiri\*/

    public BTNode2 getRight() {

        return right;

    }

    /\* Fungsi untuk menetapkan data dari node\*/

    public *void* setData(*E* *d*) {

        data = *d*;

    }

    /\* Fungsi untuk mengambil data dari node\*/

    public *E* getData() {

        return data;

    }

}

## BinaryTree2.java

class BinaryTree2<*E*> {

/\* Menginisiasi BTNode2 dengan variabel root\*/

    private BTNode2 root;

    /\* Constructor untuk menyetel pada head dari tree atau root adalah null, masih kosong\*/

    public BinaryTree2() {

        root = null;

    }

    /\* Cek apakah root atau kepala dari tree kosong\*/

    public *boolean* isEmpty() {

        return root == null;

    }

    /\* Fungsi untuk menambahkan data sesuai function insert() \*/

    public *void* insert(*E* *data*) {

        root = insert(root, *data*);

    }

    /\* Ini adalah fungsi rekursif untuk menambahkan datanya.\*/

    private BTNode2 insert(BTNode2 *node*, *E* *data*) {

/\* Jika node belum ada nilai atau belum terbentuk akan membuat node denga n isi data saat itu juga\*/

        if (*node* == null)

*node* = **new** BTNode2((*Comparable*) *data*);

/\* Tapi jika sudah terbentuk atau ada nilai maka akan dicek lagi…..\*/

        else {

/\* Data yang lebih kecil dari node parentnya akan berada di kiri sedangkan data yang lebih besarnya akan ada di node kanan\*/

            if *node*.getData().compareTo(*data*) > 0){

*node*.left = insert(*node*.left, *data*);

            }else{

*node*.right = insert(*node*.right, *data*);

}

        }

        return *node*;

    }

    /\* Mengembalikan hasil dari fungsi countNodes() \*/

    public *int* countNodes() {

        return countNodes(root);

    }

    /\* Fungsi rekursif untuk menghitung banyaknya node\*/

    private *int* countNodes(BTNode2 *r*) {

/\* Jika node TRUE kosong, kembalikan 0 \*/

        if (*r* == null)

            return 0;

/\* Jika FALSE, setiap node yang ada dikanan dan kiri(anak/turunan dari root dihitung satu satu.)\*/

        else {

*int* l = 1;/\* Karena ada akar maka nodenya berarti sudah ada 1\*/

            l += countNodes(*r*.getLeft());

            l += countNodes(*r*.getRight());

            return l;

        }

    }

    /\* Mengembalikan fungsi search(). Menerima argument val atau data yang akan dicari. search(root, val) = mencari data dari semua node atau treenya\*/

    public *boolean* search(*E* *val*) {

        return search(root, *val*);

    }

    /\* Fungsi rekursif untuk mencari nilai didalam tree \*/

    private *boolean* search(BTNode2 *r*, *E* *val*) {

/\* Jika TRUE didalam Tree sama dengan nilai yang diminta, maka kembalikan true\*/

        if (*r*.getData().equals(*val*))

            return true;

/\* Jika TRUE didalam Tree, cek, apakah pada cabang kiri tidak kosong, kemudian cek lagi, apakah dibagian kiri sama dengan nilai yang diminta, maka kembalikan true \*/

        if (*r*.getLeft() != null)

            if (search(*r*.getLeft(), *val*))

                return true;

/\* Lakukan hal serupa untuk bagian kanan\*/

        if (*r*.getRight() != null)

            if (search(*r*.getRight(), *val*))

                return true;

        return false;

    }

    /\* Mengembalikan hasil fungsi inorder()\*/

    public *void* inorder() {

        inorder(root);

    }

    /\* Fungsi inorder\*/

    private *void* inorder(BTNode2 *r*) {

/\* Ketika node/root tidak kosong, maka lakukan fungsi rekursif untuk node arah kiri. Ambil dan tampilkan datanya. Kemudian Kembali ke node sebelumnya, lalu ke arah kanan, ambil dan cetak, begitupun seterusnya. Pattern: Kiri – Root - Kanan\*/

        if (*r* != null) {

            inorder(*r*.getLeft());

            System.out.print(*r*.getData() + " ");

            inorder(*r*.getRight());

        }

    }

    /\* Mengembalikan hasil fungsi preorder()\*/

    public *void* preorder() {

        preorder(root);

    }

    /\* Fungsi preorder\*/

    private *void* preorder(BTNode2 *r*) {

/\* Ketika node/root tidak kosong, langsung ambil dan cetak data. Kemudian kunjungi node kiri, rekursif node kiri dulu, langsung ambil dan cetak data. Pattern: Root – Kiri - Kanan\*/

        if (*r* != null) {

            System.out.print(*r*.getData() + " ");

            preorder(*r*.getLeft());

            preorder(*r*.getRight());

        }

    }

    /\* Mengembalikan hasil fungsi postorder()\*/

    public *void* postorder() {

        postorder(root);

    }

/\* Fungsi preorder()\*/

    private *void* postorder(BTNode2 *r*) {

/\* Ketika node/root tidak kosong, kunjugi node kiri hingga ujung. Jika sudah tidak ada lagi node kiri tapi masih ada node kanan, kunjungi node kanan. Jika dinode tersebut ada node kiri maka kunjugi node itu. Jika node itu ujungnya(tidak ada node turunan) maka ambil dan cetak. Ada node saudara, cetak juga, lalu kembali ke parent nodenya dan cetak. Pattern: Kiri – Kanan - Root\*/

        if (*r* != null) {

            postorder(*r*.getLeft());

            postorder(*r*.getRight());

            System.out.print(*r*.getData() + " ");

        }

    }

}

## Main.java

import java.util.Scanner;

/\*\*

 \* keseluruhan program diambil dari Java Program to Implement Binary Tre \*

 \* https://www.sanfoundry.com/java-program-implement-binary-tree/ \* dengan

 \* sedikit perubahan

 \*/

public class Main {

    public static *void* main(String[] *args*) {

        Scanner scan = **new** Scanner(System.in);

        /\* Creating object of BT \*/

        BinaryTree2 bt = **new** BinaryTree2();

        /\*  Perform tree operations  \*/

        System.out.println("Binary Tree Test\n");

*char* ch;

        do {

            System.out.println("\nBinary Tree Operations\n");

            System.out.println("1. insert ");

            System.out.println("2. search");

            System.out.println("3. count nodes");

            System.out.println("4. check empty");

*int* choice = scan.nextInt();

            scan.nextLine();

            switch (choice) {

                case 1 :

                System.out.println("Enter integer element to insert");

                bt.insert(scan.nextLine());

                break;

                case 2 :

                System.out.println("Enter integer element to search");

                System.out.println("Search result : "+ bt.search( scan.nextLine() ));

                break;

                case 3 :

                System.out.println("Nodes = "+ bt.countNodes());

                break;

                case 4 :

                System.out.println("Empty status = "+ bt.isEmpty());

                break;

                default :

                System.out.println("Wrong Entry \n ");

                break;

            }

            /\*  Display tree  \*/

            System.out.print("\nPost order : ");

            bt.postorder();

            System.out.print("\nPre order : ");

            bt.preorder();

            System.out.print("\nIn order : ");

            bt.inorder();

            System.out.println("\n\nDo you want to continue (Type y or n) \n");

            ch=scan.next().charAt(0);

        }while(ch=='Y'||ch=='y');

        }

    }

Hasil Program:

A number on a black background

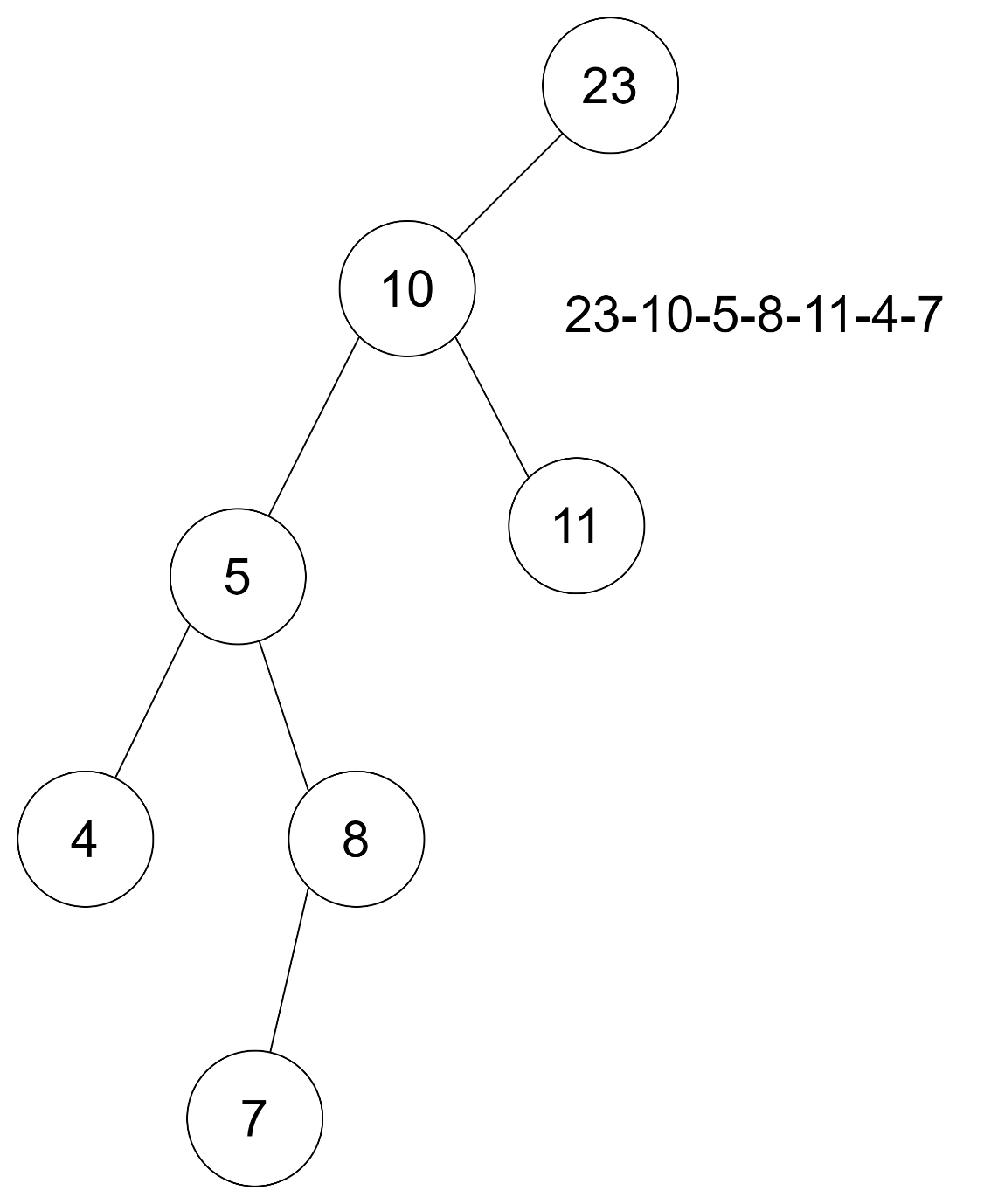
Description automatically generated

Note: Output program tidak sesuai dengan yang diharapkan. Letak angka 11 dan 4 tidak sesuai. Seharusnya 23-10-5-8-11-4-7 sama tertulis di preorder. Meskipun insert sudah dibetulkan tapi output program selalu menghasilkan demikian. Output bisa ditampilkan jika tidak memakai inputan dari user. Berikut hasilnya:

A number on a black background

Description automatically generated

# Gambaran Output dalam Tree



**Post-order: 4-7-8-5-11-10-23**

**Pemahaman saya: “Ambil dari daun dulu”**

**Pola dari postorder adalah Kiri-Kanan-Root. Pertama, mulai dari akar. Akar punya turunan yang dikiri yaitu 10, Karena 10 masih ada turunan di kiri maka kunjungi 5. 5 juga masih sama, kunjungi lagi 4. Karena 4 tidak ada turunan lagi maka dia akan dicetak. Parentnya 5 masih ada node yaitu 8. Karena 8 ada turunan yaitu 7 maka kunjungi dan cetak. Kembali ke parent, cetak 5. Kembali ke parent yaitu 10. 10 masih memiliki turunan di kanan, cetak 11. Cetak 10 dan yang terakhir cetak 23.**

**Pre-order: 23-10-5-4-8-7-11**

**Pemahaman saya: “Cetak Root dan trusss ke kiri. Root-Kiri cetak, Kanan menyusul”**

**Pola preorder adalah Root-Kiri-Kanan. Root dicetak, 10 dicetak, 5 dicetak, 4 dicetak dan mentok, kunjungi kanan yaitu saudaranya, 8. 8 dicetak. 8 ada turunan dikiri yaitu 7, cetak 7. Dan angka yang terakhir yang belum dicetak adalah 11, cetak dia juga.**

**In-order: 4-5-7-8-10-11-23**

**Pemahaman saya: “Setelah diujung kiri, jangan lupa balik ke parent supaya dicetak dulu. Trus fokus lagi ke yang ada kiri untuk dicetak”**

**Pola inorder adalah Kiri-Root-Kanan. Dari root, turun ke cabang kirinya yaitu 10. Dari 10 turun lagi ke cabang kirinya yaitu 5. Turun lagi ke cabang kirinya yaitu 4 dan cetak(karena tidak ada lagi turunan di kiri). Kembali ke parent dan cetak angka 5. 5 masih ada turunan yaitu 8, tapi jangan dulu dicetak. Setelah 8, masih ada turunan di kiri yaitu 7, cetak dia lebih dulu baru nanti parentnya. Lakukan hal serupa, anaknya (5) sudah dicetak, cetak orang tuanya(10). 11 posisinya dikanan dan child dicetak lebih dulu, baru ke 23.**