

**LAPORAN
WORKSHOP STRUKTUR DATA
“P15: Pohon”**



**NAMA : Charles Wijaya
NIM : 2255301032
KELAS : 2 TI D
DOSEN : Silvana Rasio Henim, S.S.T., M.T.
ILB : Hazimah Fatin Bachrum, S.S.T.**

**PROGRAM STUDI TEKNIK INFORMATIKA
POLITEKNIK CALTEX RIAU
TA 2023 / 2024**

Dasar Teori

Pengertian:

Tree adalah kumpulan node yang saling terhubung satu sama lain dalam suatu kesatuan yang membentuk layaknya struktur pohon.

Suatu struktur data yang tidak linier yang menggambarkan hubungan yang hirarkis (one to many dan tidak linier antara elemen elemennya).

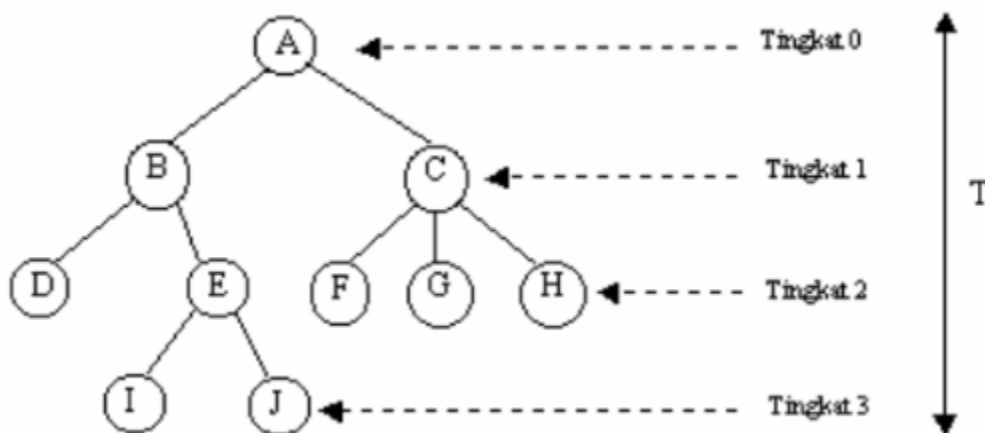
Berdasarkan jumlah elemen nodenya Tree dapat dibagi dua, yaitu:

- Tree Statik → isi node nya tetap karena bentuk pohonnya sudah ditemukan.
- Tree Dinamik → isi nodenya berubah ubah karena proses penambahan dan penghapusan.

Node Root

- Node Root dalam sebuah tree adalah suatu node yang memiliki hirarki tertinggi dan dapat juga memiliki node node anak. Semua node dapat ditelusuri dari node root tersebut.
- Node Root adalah node khusus yang tercipta pertama kalinya.
- Node node lain di bawah node root saling terhubung satu sama lain dan disebut subTree.

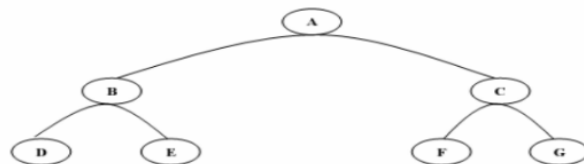
Contoh gambar Tree beserta tingkatannya:



Gambar 6.1. Contoh sebuah Pohon beserta Tingkatnya

Istilah-istilah dalam Tree:

1. Predecessor → node yang berada di atas node tertentu.
2. Successor → node yang berada di bawah node tertentu.
3. Ancestor → Seluruh node yang terletak sebelum node tertentu dan berada pada jalur yang sama.
4. Descendant: seluruh node yang terletak sesudah node tertentu dan berada pada jalur yang sama
5. Parent → Predecessor satu level diatas satu node.
6. Child → Predecessor satu level di bawah satu node.
7. Sibling → Node-node yang memiliki parent yang sama
8. Subtree → Suatu node beserta decendantnya.
9. Size: banyaknya node dalam suatu tree
10. Height: banyaknya tingkatan dalam suatu tree
11. Root: Node khusus yang tidak memiliki predecessor Leaf: Node-node dalam tree yang tidak memiliki successor.

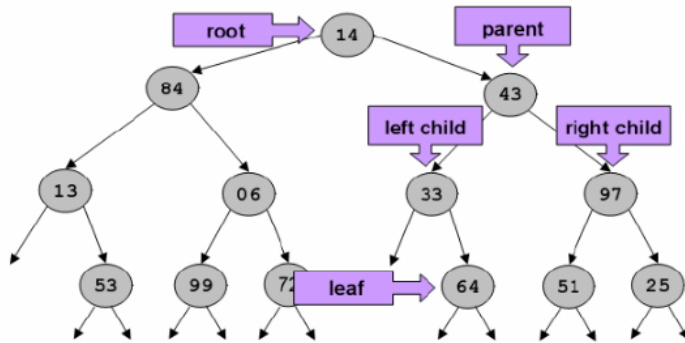


Ancestor (F)	= C,A
Descendant (C)	= F,G
Parent (D)	= B
Child (A)	= B,C
Sibling (F)	= G
Size	= 7
Height	= 3
Root	= A
Leaf	= D,E,F,G
Degree (C)	= 2

Binary Tree

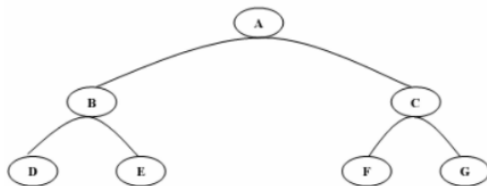
Pengertian:

Binary Tree adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah.

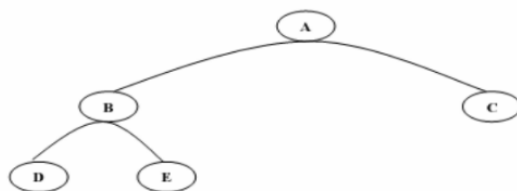


Jenis binary tree:

- **Full Binary Tree** → Binary tree yang mana semua nodenya (kecuali leaf) memiliki dua anak dan tiap subtree memiliki panjang path yang sama.



- **Complete Binary Tree** → Binary tree yang sama seperti full binary tree namun tiap path memiliki panjang yang berbeda.

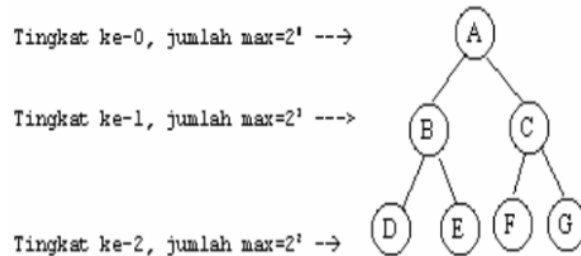


- **Skewed Binary Tree** → Binary tree yang semua nodenya (kecuali leaf) hanya memiliki satu anak.



Degree dan size binary tree:

1. Jumlah maksimum node pada setiap tingkat adalah 2^n .
2. Node pada binary tree maksimum berjumlah $2^{n+1}-1$.



Gambar 6.4. Pohon Biner Tingkat 2 Lengkap

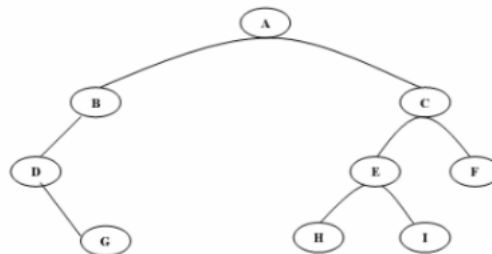
Transverse pada Tree

Pengertian:

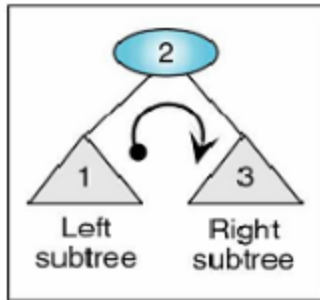
Operasi kunjungan terhadap node-node dalam pohon dimana masing-masing node akan dikunjungi sekali.

Jenis-jenis transerve:

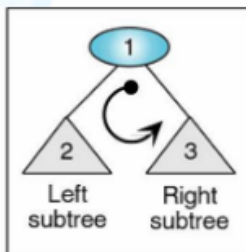
Misal terdapat Tree sebagai berikut:



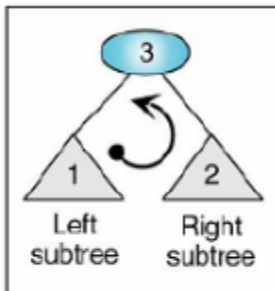
1. InOrder



2. PreOrder

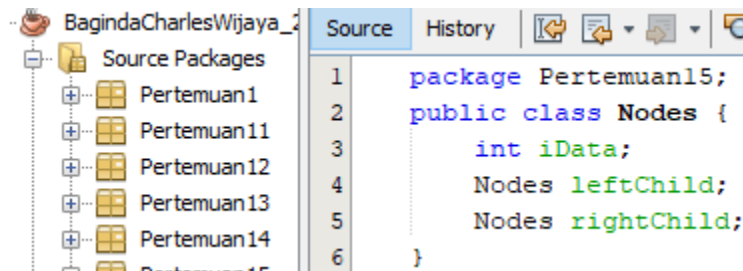


3. PostOrder



Class 1

Screenshot Program



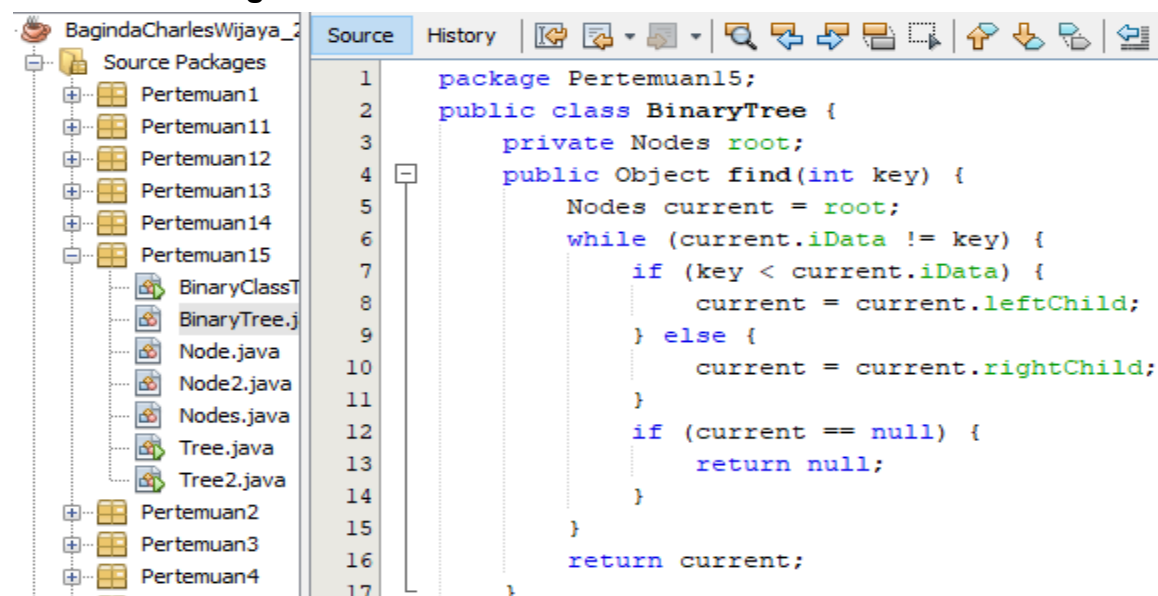
Output Program - None

Analisa program

Line	Kode Program	Analisa
2-6	<pre>public class Nodes { int iData; Nodes leftChild; Nodes rightChild; }</pre>	Sebuah class yang dapat menampung data berupa integer yang dapat menyimpan data lian dengan format class yang sama, yang dimana akan dilabelkan dengan kiri dan kanan turunan dari node tersebut, yang menyimpan integer juga.

Class 2

Screenshot Program



BagindaCharlesWijaya_2

Source Packages

- Pertemuan1
- Pertemuan11
- Pertemuan12
- Pertemuan13
- Pertemuan14
- Pertemuan15
 - BinaryClassT
 - BinaryTree.j
 - Node.java
 - Node2.java
 - Nodes.java
 - Tree.java
 - Tree2.java
- Pertemuan2
- Pertemuan3
- Pertemuan4
- Pertemuan5
- Pertemuan6
- Pertemuan7
- Pertemuan8
- TEST

Test Packages

Libraries

Test Libraries

Services Files X

BagindaCharlesWijaya

```
18 public void insert(int id) {
19     Nodes newNode = new Nodes();
20     newNode.iData = id;
21     if (root == null) {
22         root = newNode;
23     } else {
24         Nodes current = root;
25         Nodes parent;
26         while (true) {
27             parent = current;
28             if (id < current.iData) {
29                 current = current.leftChild;
30                 if (current == null) {
31                     parent.leftChild = newNode;
32                     return;
33                 }
34             } else {
35                 current = current.rightChild;
36                 if (current == null) {
37                     parent.rightChild = newNode;
38                     return;
39                 }
40             }
41         }
42     }
43 }
44
45 public boolean delete(int id) {
46     Nodes current = root;
47     Nodes parent = root;
48     boolean isLeftChild = true;
49     while (current.iData != id) {
50         parent = current;
51         if (id < current.iData) {
52             isLeftChild = true;
53             current = current.leftChild;
54         } else {
55             isLeftChild = false;
56             current = current.rightChild;
57         }
58         if (current == null) {
59             return false;
60         }
61     }
62     if (current.leftChild == null && current.rightChild == null) {
63         if (current == root) {
64             root = null;
65         }
66     }
67 }
```



```
64         } else if (isLeftChild) {
65             parent.leftChild = null;
66         } else {
67             parent.rightChild = null;
68         }
69     } else if (current.rightChild == null) {
70         if (current == root) {
71             root = current.leftChild;
72         } else if (isLeftChild) {
73             parent.leftChild = current.leftChild;
74         } else {
75             parent.rightChild = current.leftChild;
76         }
77     } else if (current.leftChild == null) {
```

```
79         root = current.rightChild;
80     } else if (isLeftChild) {
81         parent.leftChild = current.rightChild;
82     } else {
83         parent.rightChild = current.rightChild;
84     }
85 } else {
86     Nodes successor = getSuccessor(current);
87     if (current == root) {
88         root = successor;
89     } else if (isLeftChild) {
90         parent.leftChild = successor;
91     } else {
92         parent.rightChild = successor;
93     }
94     successor.leftChild = current.leftChild;
95 }
96 return true;
97 }
98
99 private Nodes getSuccessor(Nodes delNode) {
100     Nodes successorParent = delNode;
101     Nodes successor = delNode;
102     Nodes current = delNode.rightChild;
103     while (current != null) {
104         successorParent = successor;
105         successor = current;
106         current = current.leftChild;
107     }
108     if (successor != delNode.rightChild) {
109         successorParent.leftChild = successor.rightChild;
110         successor.rightChild = delNode.rightChild;
111     }
112     return successor;
113 }
114
115 public Nodes minimum() {
116     Nodes current, last = null;
117     current = root;
118     while (current != null) {
119         last = current;
120         current = current.leftChild;
121     }
122     return last;
123 }
124 }
```

Output Program - None

Analisa Program

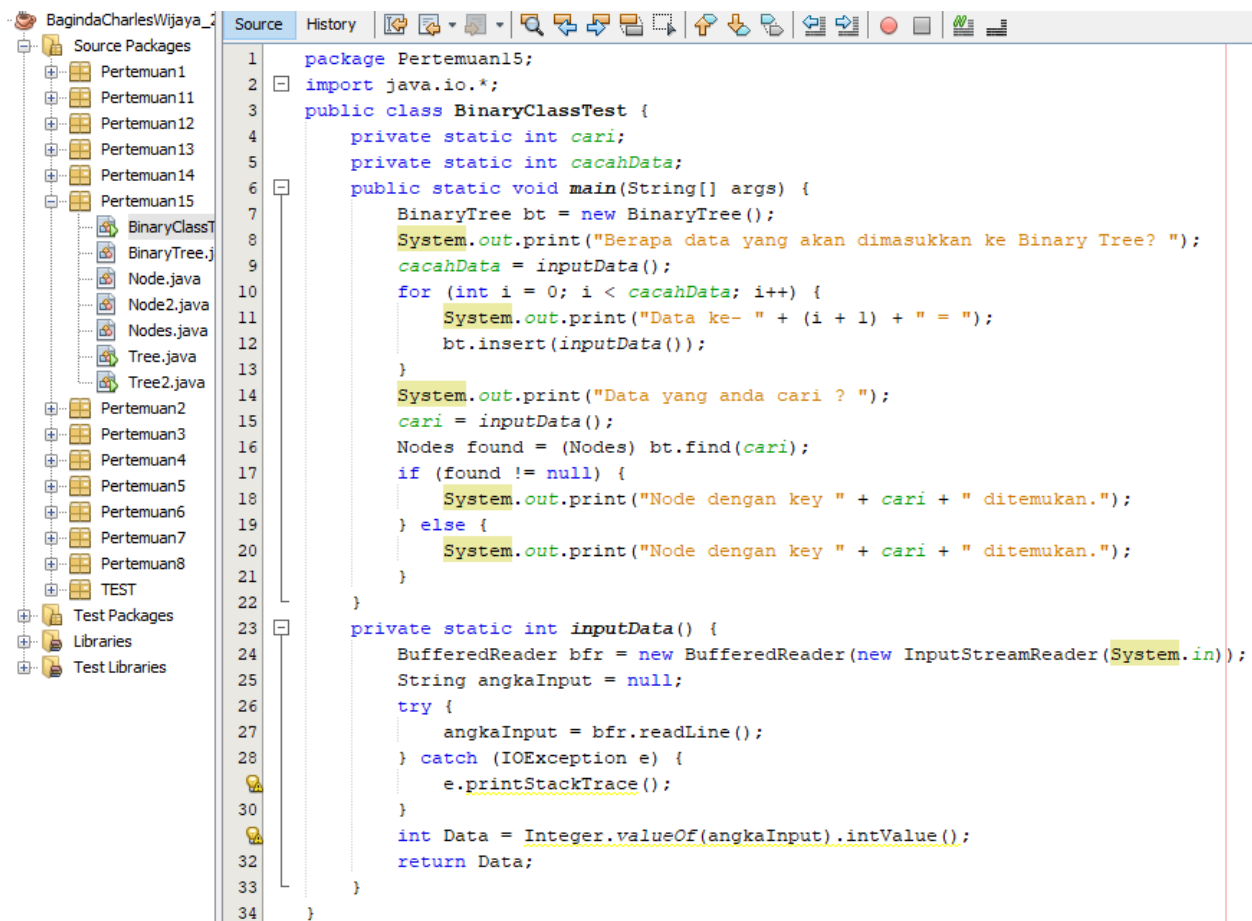
Line	Kode Program	Analisa
3-17	<pre>private Nodes root; public Object find(int key) { Nodes current = root; while (current.iData != key) { if (key < current.iData) { current = current.leftChild; } else { current = current.rightChild; } if (current == null) { return null; } } return current; }</pre>	<p>Membuat object dari nodes dengan nama root agar dapat diakses class nodes. Lalu ada method yang dapat mencari data yang ada sesuai dengan argumen yang dimasukan, dan membuat nodes bantu menjadi root. Akan membuat while loop yang mencari terus yang dimana selama bukan key yang dicari, akan terus mencari sampai masuk kondisi jika null, maka akan langsung return. Jika key nilainya lebih kecil dari current, maka akan berpindah banding ke yang kiri, dan sebaliknya kekanan, sampai dengan kondisi while tidak terpenuhi, dan setelah ketemu akan di retur nilainya.</p>
18-43	<pre>public void insert(int id) { Nodes newNode = new Nodes(); newNode.iData = id; if (root == null) { root = newNode; } else { Nodes current = root; Nodes parent; while (true) { parent = current; if (id < current.iData) { current = current.leftChild; if (current == null) { parent.leftChild = newNode; return; } } else { current = current.rightChild; if (current == null) { parent.rightChild = newNode; return; } } } } }</pre>	<p>Memasukan data yang ada dengan bantuan nodes baru yang ingin dimasukan, dengan memasukan data yang sesuai dengan argumen yang diberikan. Jika rootnya kosong akan langsung meletakkan kedalam root, dan sebaliknya akan membuat nodes root dan parent, dan selama true, akan membuat parent sama dengan current yang dijadikan root sementara. Jika datanya kurang dari data root, akan menjadikan current sama dengan leftnya yang juga dicek apakah null, jika iya langsung membuat left childnya jadi new node tersebut, dan langsung return.hal yang sama jika else, masuk ke kanan. Jika data ada akan mengulang dari awal, sampai dicari terus ke left atau right yang null.</p>

44-97	<pre> public boolean delete(int id) { Nodes current = root; Nodes parent = root; boolean isLeftChild = true; while (current.iData != id) { parent = current; if (id < current.iData) { isLeftChild = true; current = current.leftChild; } else { isLeftChild = false; current = current.rightChild; } if (current == null) { return false; } } if (current.leftChild == null && current.rightChild == null) { if (current == root) { root = null; } else if (isLeftChild) { parent.leftChild = null; } else { parent.rightChild = null; } } else if (current.rightChild == null) { if (current == root) { root = current.leftChild; } else if (isLeftChild) { parent.leftChild = current.leftChild; } else { parent.rightChild = current.leftChild; } } else if (current.leftChild == null) { if (current == root) { root = current.rightChild; } else if (isLeftChild) { parent.leftChild = current.rightChild; } else { parent.rightChild = current.rightChild; } } else { Nodes successor = getSuccessor(current); </pre>	<p>Membuat method untuk menghapus yang menggunakan loop nantinya untuk membandingkan yang mana jika parent yang dijadikan current awal yang merupakan root, akan dibandingkan nilainya, yang jika lebih kecil maka left child, dan sebaliknya right child sampai ketemu dengan nilai yang dicari, dan jika tidak ketemu maka akan membuat return langsung false. Namun jika tidak return, akan membandingkan lagi. Jika keduanya null, maka akan mengecek jika current yang didapatkan ialah root, maka root dinulikan. Jika left child, dia yang dihapus, sama dengan right child nya. Jika right child yang null akan selalu menjadikan hal yang didapatkan menjadi left child yang sekarang. Begitu juga sebaliknya jika leftchild yang null. Dan jika tidak ada opsi yang terpenuhi diatas, maka akan menggunakan method selanjutnya untuk membantu.</p>
-------	---	--

	<pre> if (current == root) { root = successor; } else if (isLeftChild) { parent.leftChild = successor; } else { parent.rightChild = successor; } successor.leftChild = current.leftChild; } return true; } </pre>	
99-11 3	<pre> private Nodes getSuccessor(Nodes delNode) { Nodes successorParent = delNode; Nodes successor = delNode; Nodes current = delNode.rightChild; while (current != null) { successorParent = successor; successor = current; current = current.leftChild; } if (successor != delNode.rightChild) { successorParent.leftChild = successor.rightChild; successor.rightChild = delNode.rightChild; } return successor; } </pre>	<p>Method ini memiliki tujuan untuk menemukan suksesor dari node yang akan dihapus. Pencarian dimulai dari anak kanan node yang akan dihapus. Selama perulangan while, Method mencari node dengan nilai terkecil di sebelah kiri anak kanan node tersebut, yang kemudian dianggap sebagai suksesor. Selama pencarian, pointer successorParent menyimpan node parent dari suksesor, dan successor menyimpan suksesor itu sendiri. Jika suksesor bukan anak kanan langsung dari node yang akan dihapus, langkah-langkah tambahan dilakukan untuk memperbarui hubungan antara node-node terkait. Suksesor yang ditemukan akhirnya dikembalikan oleh Method ini dan kemudian digunakan dalam operasi penghapusan untuk menggantikan node yang akan dihapus dengan suksesor tersebut.</p>
40-51	<pre> public Nodes minimum() { Nodes current, last = null; current = root; while (current != null) { last = current; current = current.leftChild; } return last; } </pre>	<p>Membuat current dan last menjadi null, dan current menjadi root. Menggunakan loop while sampai current ialah null, dengan aksi membuat node last menjadi current, dan ambil leftchild dari current karena lebih kecil sampai ke null, dan return node lastnya.</p>

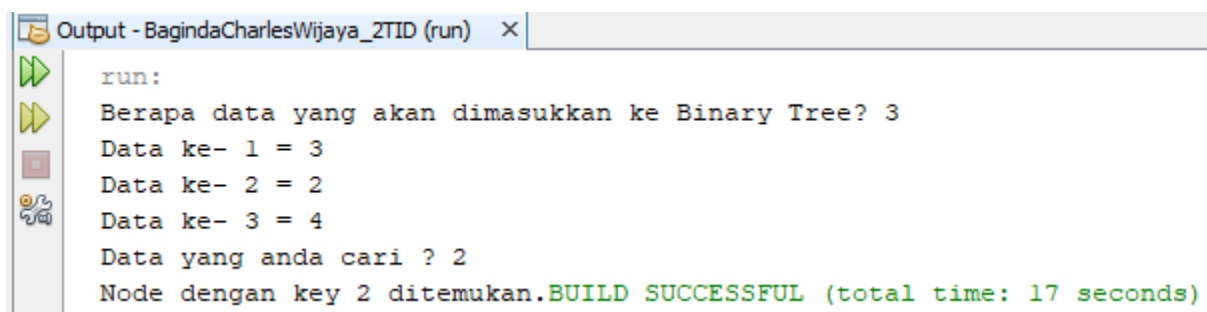
Class 3

Screenshot Program



```
1 package Pertemuan15;
2 import java.io.*;
3 public class BinaryClassTest {
4     private static int cari;
5     private static int cacahData;
6     public static void main(String[] args) {
7         BinaryTree bt = new BinaryTree();
8         System.out.print("Berapa data yang akan dimasukkan ke Binary Tree? ");
9         cacahData = inputData();
10        for (int i = 0; i < cacahData; i++) {
11            System.out.print("Data ke- " + (i + 1) + " = ");
12            bt.insert(inputData());
13        }
14        System.out.print("Data yang anda cari ? ");
15        cari = inputData();
16        Nodes found = (Nodes) bt.find(cari);
17        if (found != null) {
18            System.out.print("Node dengan key " + cari + " ditemukan.");
19        } else {
20            System.out.print("Node dengan key " + cari + " ditemukan.");
21        }
22    }
23    private static int inputData() {
24        BufferedReader bfr = new BufferedReader(new InputStreamReader(System.in));
25        String angkaInput = null;
26        try {
27            angkaInput = bfr.readLine();
28        } catch (IOException e) {
29            e.printStackTrace();
30        }
31        int Data = Integer.valueOf(angkaInput).intValue();
32        return Data;
33    }
34 }
```

Output Program



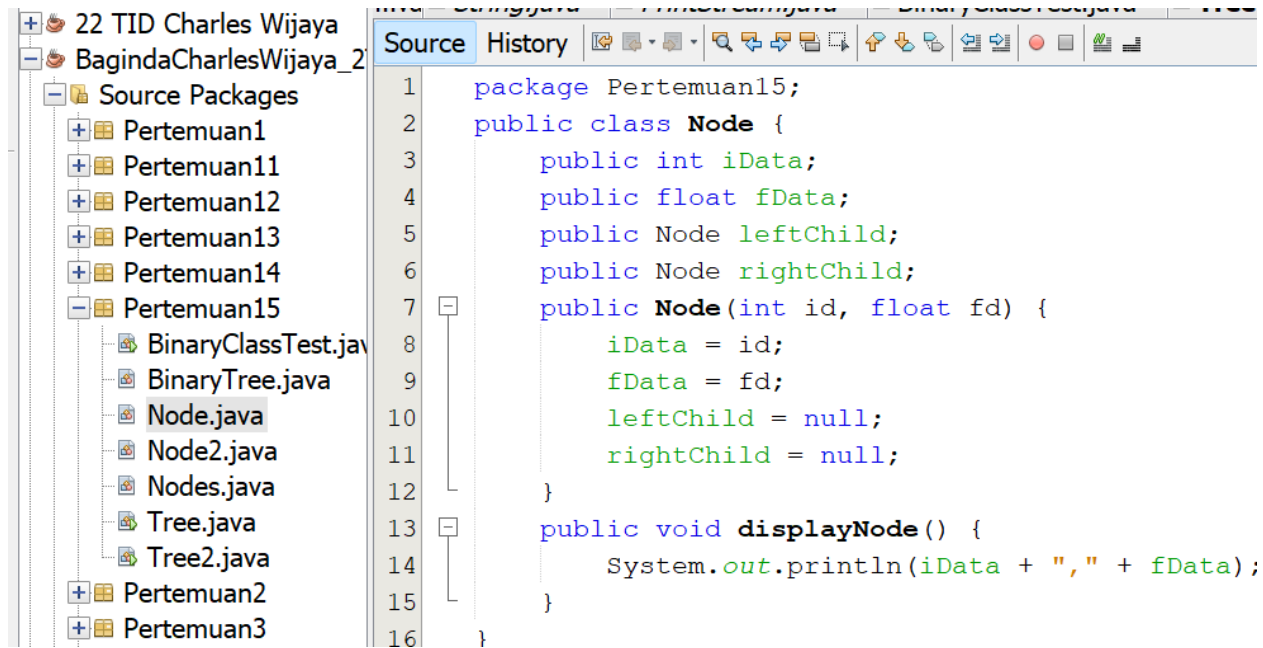
```
run:
Berapa data yang akan dimasukkan ke Binary Tree? 3
Data ke- 1 = 3
Data ke- 2 = 2
Data ke- 3 = 4
Data yang anda cari ? 2
Node dengan key 2 ditemukan.BUILD SUCCESSFUL (total time: 17 seconds)
```

Analisa program

Line	Kode Program	Analisa
4-5	<pre>private static int cari; private static int cacahData;</pre>	Membuat dua variabel yang dapat dipakai dalam class
6-10	<pre>BinaryTree bt = new BinaryTree(); System.out.print("Berapa data yang akan dimasukkan ke Binary Tree? "); cacahData = inputData();</pre>	Memanggil object binary tree, dan menanyakan jumlah data didalamnya
11-13	<pre>for (int i = 0; i < cacahData; i++) { System.out.print("Data ke- " + (i + 1) + " = "); bt.insert(inputData()); } System.out.print("Data yang anda cari ? "); cari = inputData(); Nodes found = (Nodes) bt.find(cari); if (found != null) { System.out.print("Node dengan key " + cari + " ditemukan."); } else { System.out.print("Node dengan key " + cari + " ditemukan."); }</pre>	Memasukan data data kedalam binary tree sesuai dengan jumlah data yang diinputkan. Lalu menggunakan method untuk mencari data yang telah diinputkan tadi dengan bantuan function pada tree, dan akan memberikan notifikasi sesuai dengan nilai yang direturn oleh method.
	<pre>private static int inputData() { BufferedReader bfr = new BufferedReader(new InputStreamReader(System.in)); String angkaInput = null; try { angkaInput = bfr.readLine(); } catch (IOException e) { e.printStackTrace(); } int Data = Integer.valueOf(angkaInput).intValue(); return Data; }</pre>	Inputan dengan menggunakan bufferedreader, dan akan mengembalikan nilai int.

Main Class

Screenshot Program:



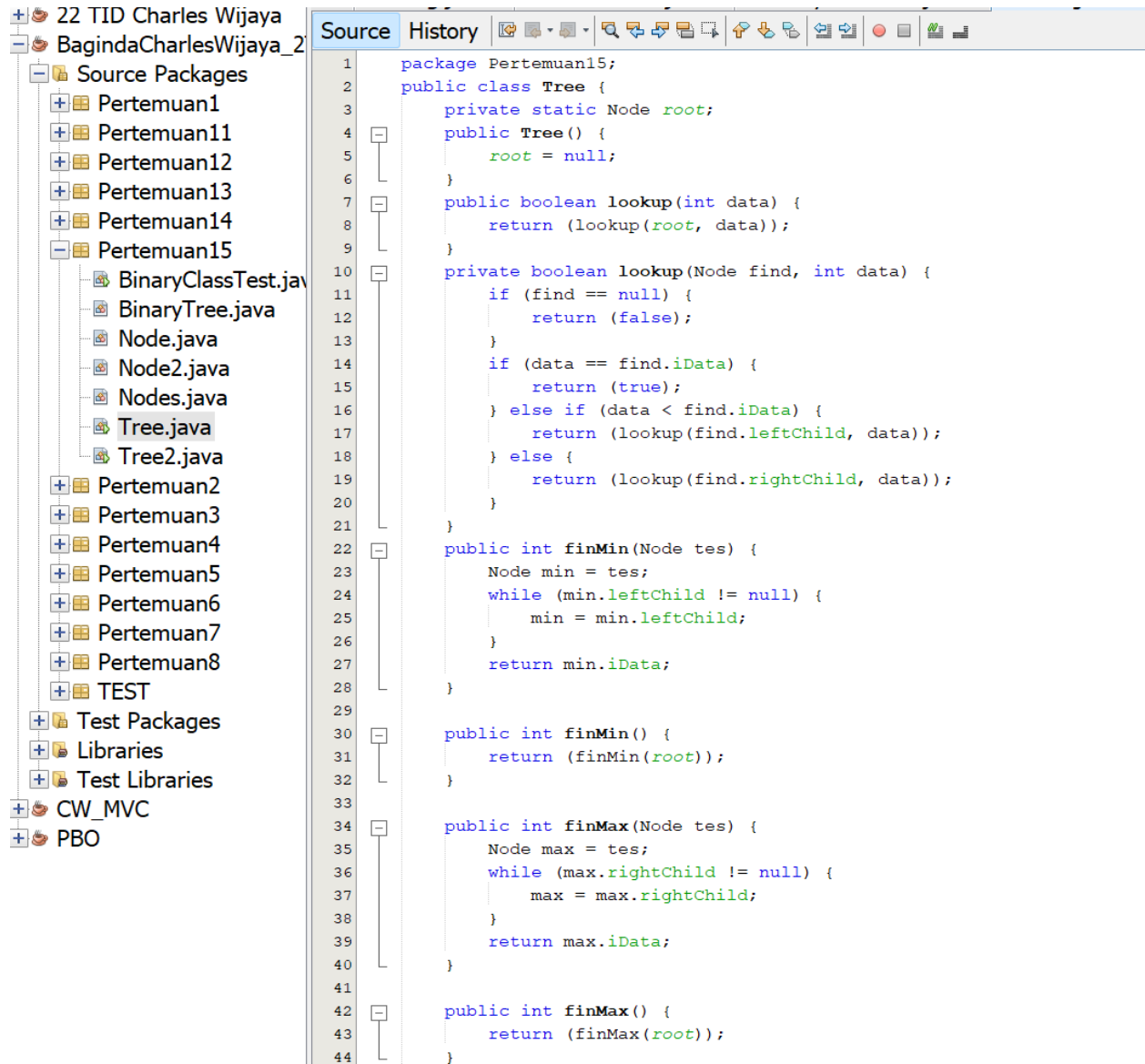
Output Program - None

Analisa program

Line	Kode Program	Analisa
3-6	<pre>public int iData; public float fData; public Node leftChild; public Node rightChild;</pre>	Membuat variabel yang dibutuhkan untuk tree, yaitu left dan right node, serta isinya dengan data yang membuatnya lebih uniq.
7-12	<pre>public Node(int id, float fd) { iData = id; fData = fd; leftChild = null; rightChild = null;} }</pre>	constructor yang ketika dibuat object akan langsung ada datanya
13-15	<pre>public void displayNode() { System.out.println(iData + "," + fData); }</pre>	Method untuk print isi nodenya.

Class 4

Screenshot Program



22 TID Charles Wijaya
 BagindaCharlesWijaya_2

- Source Packages
 - Pertemuan1
 - Pertemuan11
 - Pertemuan12
 - Pertemuan13
 - Pertemuan14
 - Pertemuan15
 - BinaryClassTest.java
 - BinaryTree.java
 - Node.java
 - Node2.java
 - Nodes.java
 - Tree.java
 - Tree2.java
 - Pertemuan2
 - Pertemuan3
 - Pertemuan4
 - Pertemuan5
 - Pertemuan6
 - Pertemuan7
 - Pertemuan8
 - TEST
- Test Packages
- Libraries
- Test Libraries
- CW_MVC
- PBO

```

45 public int size() {
46     return (size(root));
47 }
48 private int size(Node node) {
49     if (node == null) {
50         return (0);
51     } else {
52         return (size(node.leftChild) + 1 + size(node.rightChild));
53     }
54 }
55 public void insert(int id, float fd) {
56     Node baru = new Node(id, fd);
57     if (root == null) {
58         System.out.println("Node baru " + id + " sebagai root");
59         root = baru;
60     } else {
61         Node current = root;
62         Node parent;
63         while (true) {
64             parent = current;
65             if (id < current.idData) {
66                 current = current.leftChild;
67                 if (current == null) {
68                     System.out.println("Insert " + id + " sebagai anak kiri dari " + parent.idData);
69                     parent.leftChild = baru;
70                     break;
71                 }
72             } else {
73                 current = current.rightChild;
74                 if (current == null) {
75                     System.out.println("Insert " + id + " sebagai anak kanan dari " + parent.idData);
76                     parent.rightChild = baru;
77                     break;
78                 }
79             }
80         }
81     }
82 }
83 public void printTree() {
84     printTree(root);
85     System.out.println();
86 }
87 public void printTree(Node localroot) {
88     if (localroot != null) {
89         printTree(localroot.leftChild);
90         localroot.displayNode();
91         printTree(localroot.rightChild);
92     }
93 }
94 public void InOrder() {
95     InOrder(root);
96     System.out.println();
97 }
98 public void InOrder(Node localroot) {
99     if (localroot != null) {
100         InOrder(localroot.leftChild);
101         localroot.displayNode();
102         InOrder(localroot.rightChild);
103     }
104 }

```

22 TID Charles Wijaya

BagindaCharlesWijaya_2

Source Packages

Pertemuan1

Pertemuan11

Pertemuan12

Pertemuan13

Pertemuan14

Pertemuan15

BinaryClassTest.java

BinaryTree.java

Node.java

Node2.java

Nodes.java

Tree.java

Tree2.java

Pertemuan2

Pertemuan3

Pertemuan4

Pertemuan5

Pertemuan6

Pertemuan7

Pertemuan8

TEST

Test Packages

Libraries

Test Libraries

CW_MVC

PBO

Source

History

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

```

public void PostOrder() {
    PostOrder(root);
    System.out.println();
}

public void PostOrder(Node localroot) {
    if (localroot != null) {
        PostOrder(localroot.leftChild);
        PostOrder(localroot.rightChild);
        localroot.displayNode();
    }
}

public void PreOrder() {
    PreOrder(root);
    System.out.println();
}

public void PreOrder(Node localroot) {
    if (localroot != null) {
        localroot.displayNode();
        PreOrder(localroot.leftChild);
        PreOrder(localroot.rightChild);
    }
}

public static void main(String args[]) {
    Tree pohon = new Tree();
    pohon.insert(12, 3);
    pohon.insert(10, 3);
    pohon.insert(15, 3);
    pohon.insert(5, 3);
    pohon.insert(17, 3);
    System.out.println("Print InOrder");
    pohon.InOrder();
    System.out.println("Print PostOrder");
    pohon.PostOrder();
    System.out.println("Print PreOrder");
    pohon.PreOrder();
    boolean tampung = pohon.lookup(12);
    if (tampung == true) {
        System.out.println("Ketemu");
    } else {
        System.out.println("TIdak Ketemu");
    }
    int nilaimin = pohon.finMin();
    System.out.println("nilai minimum = " + nilaimin);
    int nilaimax = pohon.finMax();
    System.out.println("nilai maksimum = "+nilaimax);
    int count = pohon.size();
    System.out.println("jumlah node " + count);
}

```

Output Program:

```
Output - BagindaCharlesWijaya_2TID (run) X
run:
Node baru 12 sebagai root
Insert 10 sebagai anak kiri dari 12
Insert 15 sebagai anak kanan dari 12
Insert 5 sebagai anak kiri dari 10
Insert 17 sebagai anak kanan dari 15
Print InOrder
5,3.0
10,3.0
12,3.0
15,3.0
17,3.0

Print PostOrder
5,3.0
10,3.0
17,3.0
15,3.0
12,3.0

Print PreOrder
12,3.0
10,3.0
5,3.0
15,3.0
17,3.0

Ketemu
nilai minimum = 5
nilai maksimum = 17
jumlah node 5
```

Analisa Program:

Line	Kode Program	Analisa
3-6	<pre>private static Node root; public Tree() { root = null;}</pre>	Membuat object node yang bisa dipakai di class tree, dan constructor yang membuat root menjadi null jika baru.
7-21	<pre>public boolean lookup(int data) { return (lookup(root, data)); } private boolean lookup(Node find, int data) { if (find == null) { return (false); } if (data == find.iData) {</pre>	Membuat fitur search tentang data yang ada, dan jika data sama, akan diberikan true, dan jika data lebih kecil, atau besar, akan masuk ke left atau right dengan rekursif sampai ketemu.

	<pre> return (true); } else if (data < find.iData) { return (lookup(find.leftChild, data)); } else { return (lookup(find.rightChild, data)); } } </pre>	
22–32	<pre> public int finMin(Node tes) { Node min = tes; while (min.leftChild != null) { min = min.leftChild; } return min.iData; } public int finMin() { return (finMin(root)); } </pre>	Karena left pasti angka yang lebih kecil, akan masuk ke left nodes terus sampai left node dari yang left node terakhir yaitu null, lalu di return
34-44	<pre> public int finMax(Node tes) { Node max = tes; while (max.rightChild != null) { max = max.rightChild; } return max.iData; } public int finMax() { return (finMax(root)); } </pre>	Karena kanan pasti angka yang lebih besar, akan masuk ke right nodes terus sampai right node dari yang right node terakhir yaitu null, lalu di return
45-54	<pre> public int size() { return (size(root)); } private int size(Node node) { if (node == null) { return (0); } else { return (size(node.leftChild) + 1 + size(node.rightChild)); } } </pre>	Penggunaan rekursif untuk mengembalikan nilai dari left child dan right child yang diakses, dan return yang ditambahkan 1 untuk rootnya.
55-82	<pre> public void insert(int id, float fd) { Node baru = new Node(id, fd); if (root == null) { System.out.println("Node baru " + id + " sebagai root"); } } </pre>	Memasukan node baru. Jika root kosong, node baru jadi root. Jika tidak, node baru akan dilihat apakah lebih kecil dari data sekarang atau tidak. Jika iya, maka akan memasukkannya kedalam left child, dan

	<pre> root = baru; } else { Node current = root; Node parent; while (true) { parent = current; if (id < current.iData) { current = current.leftChild; if (current == null) { System.out.println("Insert " + id + " sebagai anak kiri dari " + parent.iData); parent.leftChild = baru; break; } } else { current = current.rightChild; if (current == null) { System.out.println("Insert " + id + " sebagai anak kanan dari " + parent.iData); parent.rightChild = baru; break; } } } } } } </pre>	<p>sebaliknya. Hal ini akan diulang sampai left atau right node yang ingin diisi null.</p>
83-93	<pre> public void printTree() { printTree(root); System.out.println(); } public void printTree(Node localroot) { if (localroot != null) { printTree(localroot.leftChild); localroot.displayNode(); printTree(localroot.rightChild); } } </pre>	<p>Memprint secara inorder dengan bantuan rekursif yang dimulai dari left child</p>
94-12 7	<pre> public void InOrder() { InOrder(root); System.out.println(); } public void InOrder(Node localroot) { if (localroot != null) { </pre>	<p>Memprint secara inorder dengan bantuan rekursif yang dimulai dari left child, atau left ke right baru rootnya, dan root dulu baru left ke right</p>

	<pre> InOrder(localroot.leftChild); localroot.displayNode(); InOrder(localroot.rightChild); } } public void PostOrder() { PostOrder(root); System.out.println(); } public void PostOrder(Node localroot) { if (localroot != null) { PostOrder(localroot.leftChild); PostOrder(localroot.rightChild); localroot.displayNode(); } } public void PreOrder() { PreOrder(root); System.out.println(); } public void PreOrder(Node localroot) { if (localroot != null) { localroot.displayNode(); PreOrder(localroot.leftChild); PreOrder(localroot.rightChild); } } </pre>	
128-1 53	<pre> public static void main(String args[]) { Tree pohon = new Tree(); pohon.insert(12, 3); pohon.insert(10, 3); pohon.insert(15, 3); pohon.insert(5, 3); pohon.insert(17, 3); System.out.println("Print InOrder"); pohon.InOrder(); System.out.println("Print PostOrder"); pohon.PostOrder(); System.out.println("Print PreOrder"); pohon.PreOrder(); boolean tampung = pohon.lookup(12); if (tampung == true) { </pre>	<p>Mencoba beberapa fungsi seperti memasukkan data kedalam tree, dan memprintkan secara in Order, preorder, dan postorder. Lalu mencoba mencari data dan akan diberitahukan jika ketemu atau tidaknya. Lalu juga akan mengambil nilai min dan max dari yang di tree, ataupun melihat size treenya</p>

	<pre> System.out.println("Ketemu"); } else { System.out.println("Tidak Ketemu"); } int nilaimin = pohon.finMin(); System.out.println("nilai minimum = " + nilaimin); int nilaimax = pohon.finMax(); System.out.println("nilai maksimum = "+nilaimax); int count = pohon.size(); System.out.println("jumlah node " + count); } </pre>	
--	--	--

Class Tugas

Screenshot Program

The screenshot shows an IDE with a project named "22 TID Charles Wijaya" and a sub-project "BagindaCharlesWijaya_2". The project structure on the left includes several "Pertemuan" folders and a "Source Packages" folder containing files like "BinaryClassTest.java", "BinaryTree.java", "Node.java", "Node2.java", "Nodes.java", "Tree.java", and "Tree2.java". The "Source" tab is active, displaying the code for "Node2.java".

```

1  package Pertemuan15;
2  public class Node2 {
3      public String iData;
4      public String fData;
5      public Node2 leftChild;
6      public Node2 rightChild;
7      public Node2(String id, String fd) {
8          iData = id;
9          fData = fd;
10         leftChild = null;
11         rightChild = null;
12     }
13     public void displayNode() {
14         System.out.println(iData + "," + fData);
15     }
16 }

```

22 TID Charles Wijaya
BagindaCharlesWijaya_2

Source Packages

- Pertemuan1
- Pertemuan11
- Pertemuan12
- Pertemuan13
- Pertemuan14
- Pertemuan15
 - BinaryClassTest.java
 - BinaryTree.java
 - Node.java
 - Node2.java
 - Nodes.java
 - Tree.java
 - Tree2.java
- Pertemuan2
- Pertemuan3
- Pertemuan4
- Pertemuan5
- Pertemuan6
- Pertemuan7
- Pertemuan8
- TEST

Test Packages

Libraries

Test Libraries

CW_MVC

PBO

```
1 package Pertemuan15;
2 import javax.swing.JOptionPane;
3 public class Tree2 {
4     private static Node2 root;
5     public Tree2() {
6         root = null;
7     }
8     public boolean lookup(String data) {
9         return lookup(root, data);
10    }
11    private boolean lookup(Node2 find, String data) {
12        if (find == null) {
13            return false;
14        }
15        if (data.equals(find.iData)) {
16            return true;
17        } else if (data.compareTo(find.iData) < 0) {
18            return lookup(find.leftChild, data);
19        } else {
20            return lookup(find.rightChild, data);
21        }
22    }
23    public String findMin(Node2 tes) {
24        Node2 min = tes;
25        while (min.leftChild != null) {
26            min = min.leftChild;
27        }
28        return min.iData;
29    }
30    public String findMin() {
31        return findMin(root);
32    }
33    public String findMax(Node2 tes) {
34        Node2 max = tes;
35        while (max.rightChild != null) {
36            max = max.rightChild;
37        }
38        return max.iData;
39    }
40    public String findMax() {
41        return findMax(root);
42    }
```


22 TID Charles Wijaya
 BagindaCharlesWijaya_2

- Source Packages
 - Pertemuan1
 - Pertemuan11
 - Pertemuan12
 - Pertemuan13
 - Pertemuan14
 - Pertemuan15
 - BinaryClassTest.java
 - BinaryTree.java
 - Node.java
 - Node2.java
 - Nodes.java
 - Tree.java
 - Tree2.java
- Pertemuan2
- Pertemuan3
- Pertemuan4
- Pertemuan5
- Pertemuan6
- Pertemuan7
- Pertemuan8
- TEST
- Test Packages
- Libraries
- Test Libraries
- CW_MVC
- PBO

```

43 public int size() {
44     return size(root);
45 }
46 private int size(Node2 node2) {
47     if (node2 == null) {
48         return 0;
49     } else {
50         return size(node2.leftChild) + 1 + size(node2.rightChild);
51     }
52 }
53 public void insert(String id, String fd) {
54     Node2 baru = new Node2(id, fd);
55     if (root == null) {
56         System.out.println("Node baru " + id + " sebagai root");
57         root = baru;
58     } else {
59         Node2 current = root;
60         Node2 parent;
61         while (true) {
62             parent = current;
63             if (id.compareTo(current.iData) < 0) {
64                 current = current.leftChild;
65                 if (current == null) {
66                     System.out.println("Insert " + id + " sebagai anak kiri dari "
67                     parent.leftChild = baru;
68                     break;
69                 }
70             } else {
71                 current = current.rightChild;
72                 if (current == null) {
73                     System.out.println("Insert " + id + " sebagai anak kanan dari "
74                     parent.rightChild = baru;
75                     break;
76                 }
77             }
78         }
79     }
80 }
81 public void printTree() {
82     printTree(root);
83     System.out.println();
84 }
85 public void printTree(Node2 localroot) {
86     if (localroot != null) {
87         printTree(localroot.leftChild);
88         localroot.displayNode();
89         printTree(localroot.rightChild);
90     }
91 }
  
```

```

public void inOrder() {
    inOrder(root);
    System.out.println();
}

public void inOrder(Node2 localroot) {
    if (localroot != null) {
        inOrder(localroot.leftChild);
        localroot.displayNode();
        inOrder(localroot.rightChild);
    }
}

public void postOrder() {
    postOrder(root);
    System.out.println();
}

public void postOrder(Node2 localroot) {
    if (localroot != null) {
        postOrder(localroot.leftChild);
        postOrder(localroot.rightChild);
        localroot.displayNode();
    }
}

public void preOrder() {
    preOrder(root);
    System.out.println();
}

public void preOrder(Node2 localroot) {
    if (localroot != null) {
        localroot.displayNode();
        preOrder(localroot.leftChild);
        preOrder(localroot.rightChild);
    }
}

```

```

public static void main(String args[]) {
    Tree2 pohon = new Tree2();
    JOptionPane.showMessageDialog(null, "Welcome to Pohon");
    String opt[] = {"Insert", "Search", "Print", "Info", "Exit"};
    while (true) {
        int option = showOption(opt);
        switch (option) {
            case 0:
                int limit = Integer.parseInt(JOptionPane.showInputDialog("How many data you wish to insert?"));
                for (int i = 0; i < limit; i++) {
                    String data = JOptionPane.showInputDialog("Data No-:" + (i+1));
                    String uniq = JOptionPane.showInputDialog("Uniq key:");
                    pohon.insert(data, uniq);
                }
                break;
            case 1:
                String searchData = JOptionPane.showInputDialog("Enter data to search:");
                boolean found = pohon.lookup(searchData);
                if (found) {
                    JOptionPane.showMessageDialog(null, "Data found!");
                } else {
                    JOptionPane.showMessageDialog(null, "Data not found.");
                }
                break;
            case 2:
                String[] prin = {"inOrder", "preOrder", "postOrder"};
                int optionss = JOptionPane.showOptionDialog(null, "Print method:",
                    "Pohon Vers 1.31", JOptionPane.DEFAULT_OPTION,
                    JOptionPane.INFORMATION_MESSAGE, null, prin, prin[0]);
                switch (optionss) {
                    case 0:
                        pohon.inOrder();
                        break;
                    case 1:
                        pohon.preOrder();
                        break;
                    case 2:
                        pohon.postOrder();
                        break;
                }
                break;
            case 3:
                System.out.println("Size of the tree: " + pohon.size());
                System.out.println("Minimum value: " + pohon.findMin());
                System.out.println("Maximum value: " + pohon.findMax());
                break;
            case 4:
                System.exit(0);
                break;
        }
    }
}

```

```

private static int showOption(String[] options) {
    String message = "What you want to do here?";
    int option = JOptionPane.showOptionDialog(null, message,
        "Pohon Vers 1.31", JOptionPane.DEFAULT_OPTION,
        JOptionPane.INFORMATION_MESSAGE, null, options, options[0]);
    return option;
}

```

Output Program

Message X

Welcome to Pohon

OK

Pohon Vers 1.31 X

What you want to do here?

Insert Search Print Info Exit

Input X

How many data you wish to insert?

3

OK Cancel

Input X

Data No.:1

5

OK Cancel

Input X

Uniq key:

angka

OK Cancel

Input X

Data No.:2

ayam

OK Cancel

Input X

Uniq key:

hewan

OK Cancel

Input X

Data No.:3

CW

OK Cancel

Input X

Uniq key:

Manusia

OK Cancel

Input X

Enter data to search:

CW

OK Cancel

Message X

Data found!

OK

Pohon Vers 1.31 X

Print method:

inOrder preOrder postOrder

Output X

BagindaCharlesWijaya_2TID (run) x BagindaCharle

run:

Node baru 5 sebagai root

Insert Ayam sebagai anak kanan dari 5

Insert CW sebagai anak kanan dari Ayam

5, angka

Ayam, Hewan

CW, Manusia

Size of the tree: 3

Minimum value: 5

Maximum value: CW

Analisa Program:

Node

Tree

Line	Kode Program	Analisa
3-6	<pre>private static Node root; public Tree() { root = null;}</pre>	Membuat object node yang bisa dipakai di class tree, dan constructor yang membuat root menjadi null jika baru.
7-22	<pre>public boolean lookup(String data) { return lookup(root, data); } private boolean lookup(Node2 find, String data) { if (find == null) { return false; } if (data.equals(find.iData)) { return true; } else if (data.compareTo(find.iData) < 0) { return lookup(find.leftChild, data); } else { return lookup(find.rightChild, data); } }</pre>	Membuat fitur search tentang data yang ada, dan jika data sama, akan diberikan true, dan jika data lebih kecil, atau besar, akan masuk ke left atau right dengan rekursif sampai ketemu.
23-32	<pre>public String findMin(Node2 tes) { Node2 min = tes; while (min.leftChild != null) { min = min.leftChild; } return min.iData; } public String findMin() { return findMin(root); }</pre>	Karena left pasti angka yang lebih kecil, akan masuk ke left nodes terus sampai left node dari yang left node terakhir yaitu null, lalu di return
33-42	<pre>public String findMax(Node2 tes) { Node2 max = tes; while (max.rightChild != null) { max = max.rightChild; }</pre>	Karena kanan pasti angka yang lebih besar, akan masuk ke right nodes terus sampai right node dari yang right node terakhir yaitu null, lalu di return

	<pre> } return max.iData; } public String findMax() { return findMax(root); } </pre>	
43-52	<pre> public int size() { return size(root); } private int size(Node2 node2) { if (node2 == null) { return 0; } else { return size(node2.leftChild) + 1 + size(node2.rightChild); } } </pre>	<p>Penggunaan rekursif untuk mengembalikan nilai dari left child dan right child yang diakses, dan return yang ditambahkan 1 untuk rootnya.</p>
53-80	<pre> public void insert(String id, String fd) { Node2 baru = new Node2(id, fd); if (root == null) { System.out.println("Node baru " + id + " sebagai root"); root = baru; } else { Node2 current = root; Node2 parent; while (true) { parent = current; if (id.compareTo(current.iData) < 0) { current = current.leftChild; if (current == null) { System.out.println("Insert " + id + " sebagai anak kiri dari " + parent.iData); parent.leftChild = baru; break; } } else { current = current.rightChild; if (current == null) { System.out.println("Insert " + id + " sebagai anak kanan dari " + parent.iData); parent.rightChild = baru; break; } } } } </pre>	<p>Memasukan node baru. Jika root kosong, node baru jadi root. Jika tidak, node baru akan dilihat apakah lebih kecil dari data sekarang atau tidak. Jika iya, maka akan memasukkannya kedalam left child, dan sebaliknya. Hal ini akan diulang sampai left atau right node yang ingin diisi null.</p>

	<pre> } } } </pre>	
81-91	<pre> public void printTree() { printTree(root); System.out.println(); } public void printTree(Node localroot) { if (localroot != null) { printTree(localroot.leftChild); localroot.displayNode(); printTree(localroot.rightChild); } } </pre>	Memprint secara inorder dengan bantuan rekursif yang dimulai dari left child
92-12 5	<pre> public void InOrder() { InOrder(root); System.out.println(); } public void InOrder(Node localroot) { if (localroot != null) { InOrder(localroot.leftChild); localroot.displayNode(); InOrder(localroot.rightChild); } } public void PostOrder() { PostOrder(root); System.out.println(); } public void PostOrder(Node localroot) { if (localroot != null) { PostOrder(localroot.leftChild); PostOrder(localroot.rightChild); localroot.displayNode(); } } public void PreOrder() { PreOrder(root); System.out.println(); } public void PreOrder(Node localroot) { if (localroot != null) { </pre>	Memprint secara inorder dengan bantuan rekursif yang dimulai dari left child, atau left ke right baru rootnya, dan root dulu baru left ke right

	<pre> localroot.displayNode(); PreOrder(localroot.leftChild); PreOrder(localroot.rightChild); } } </pre>	
126-1 83	<pre> public static void main(String args[]) { Tree2 pohon = new Tree2(); JOptionPane.showMessageDialog(null, "Welcome to Pohon"); String opt[] = {"Insert","Search","Print","Info","Exit"}; while (true) { int option = showOption(opt); switch (option) { case 0: int limit = Integer.parseInt(JOptionPane.showInput tDialog("How many data you wish to insert?")); for (int i = 0; i < limit; i++) { String data = JOptionPane.showInputDialog("Data No-:" + (i+1)); String uniq = JOptionPane.showInputDialog("Uniq key:"); pohon.insert(data, uniq);} break; case 1: String searchData = JOptionPane.showInputDialog("Enter data to search:"); boolean found = pohon.lookup(searchData); if (found) { JOptionPane.showMessageDialog(null, "Data found!"); } else { JOptionPane.showMessageDialog(null, "Data not found."); } break; case 2: String[] prin = {"inOrder","preOrder","postOrder"}; </pre>	<p>Memberikan beberapa pilihan berupa menu sesuai dengan 5 opsi yang diberikan dan akan membuat aksi dengan switch case dengan memanggil method tertentu.</p>

	<pre> int optionss = JOptionPane.showOptionDialog(null,"P rint method:", "Pohon Vers 1.31",JOptionPane.DEFAULT_OPTION , JOptionPane.INFORMATION_MESSA GE,null,prin,prin[0]); switch (optionss) { case 0: pohon.inOrder(); break; case 1: pohon.preOrder(); break; case 2: pohon.postOrder(); break} break; case 3: System.out.println("Size of the tree: " + pohon.size()); System.out.println("Minimum value: " + pohon.findMin()); System.out.println("Maximum value: " + pohon.findMax()); break; case 4: System.exit(0); break; } } } private static int showOption(String[] options) { String message = "What you want to do here?"; int option = JOptionPane.showOptionDialog(null,m essage, "Pohon Vers 1.31",JOptionPane.DEFAULT_OPTION , JOptionPane.INFORMATION_MESSA GE,null,options,options[0]); return option; } </pre>	
--	--	--