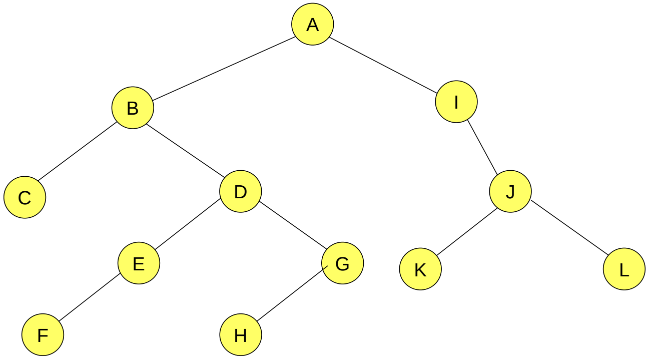
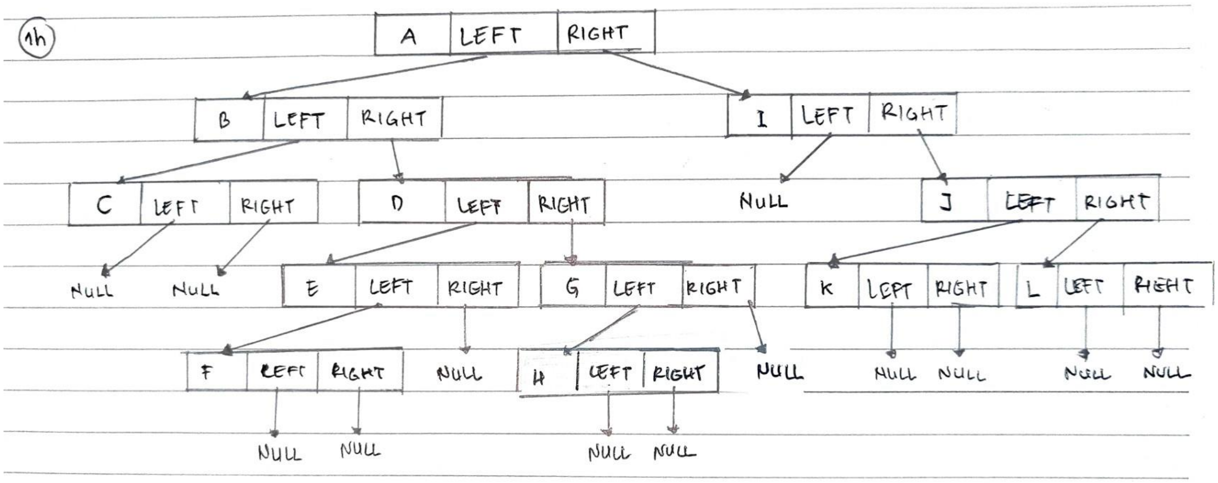
Nama : Ruth Aulya Silalahi
Kelas : 12 IF1
NIM : 11S20018
Topik : Tree, Binary Tree, Binary Search Tree

A. Bagian Pemahaman Konsep

1. Perhatikan diagram generic tree berikut ini:

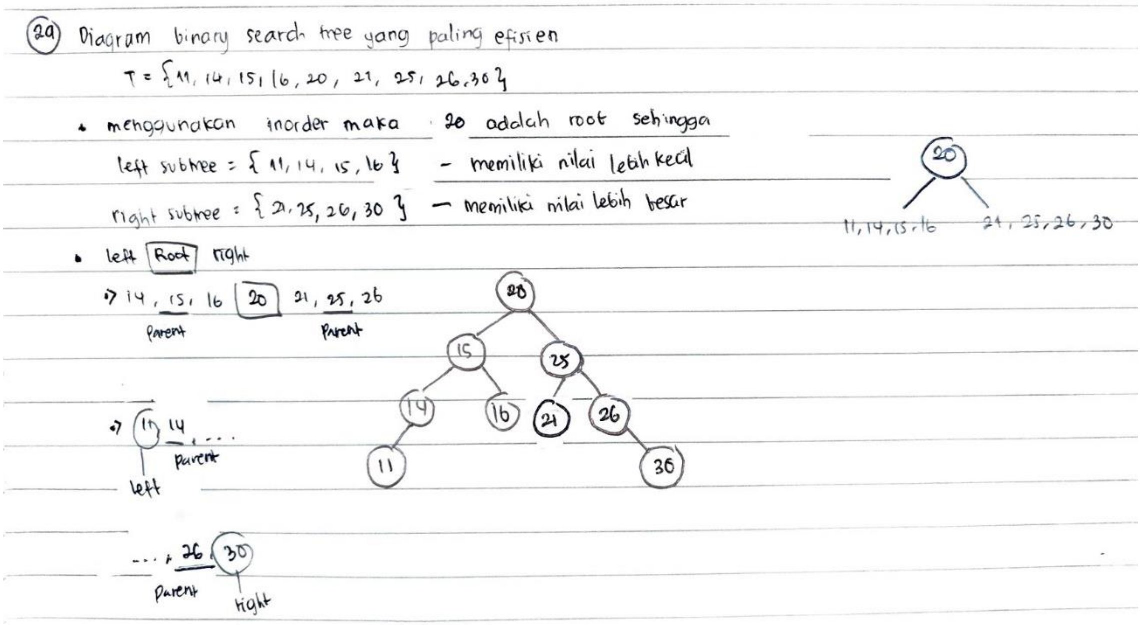


   a. Root: Node A
   b. Eksternal node (daun): C, F, H, K, L
   c. Kedalaman pohon: 4
   d. Tinggi pohon: 4
   e. Kedalaman node E: 3
   f. Descendant dari node B: C, D, E, G, F, H
   g. Ancestor dari node J: I, A
   h. Skema implementasi generic tree dengan representasi linked list:



2. Diberikan data sebagai berikut: T = {11, 14, 15, 16, 20, 21, 25, 26, 30}

   a. Diagram binary search tree yang paling efisien



2a Diagram binary search tree yang paling efisien

T = {11, 14, 15, 16, 20, 21, 25, 26, 30}

- menggunakan inorder maka 20 adalah root sehingga

left subtree = {11, 14, 15, 16} – memiliki nilai lebih kecil

right subtree = {21, 25, 26, 30} – memiliki nilai lebih besar

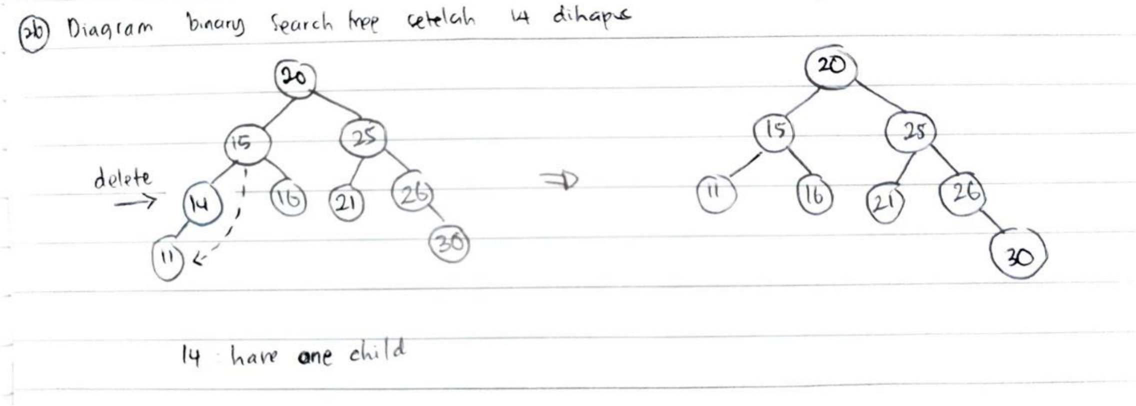
- left Root right

⇒ 14, 15, 16 20 21, 25, 26
Parent Parent

⇒ 11 14, . . .
left parent

. . . , 26 30
parent right

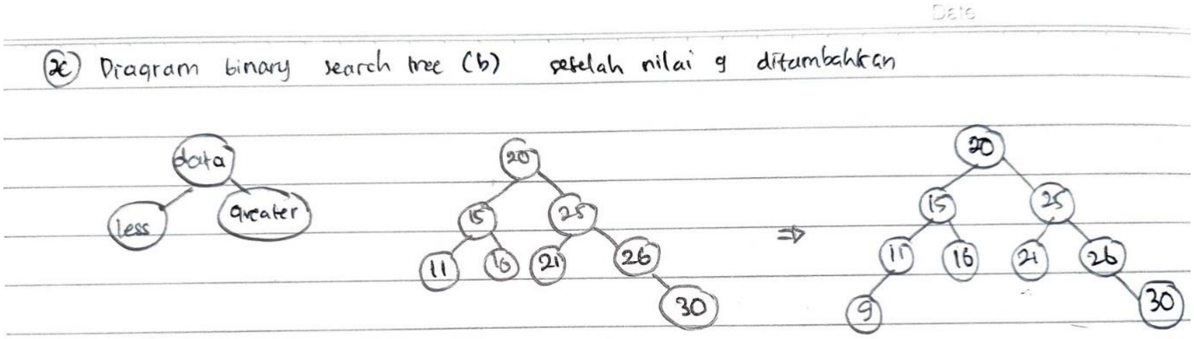
   b. Diagram binary search tree setelah nilai 14 dihapus



2b Diagram binary Search tree setelah 14 dihapus

14 have one child

   c. Diagram binary search tree setelah nilai 9 ditambahkan

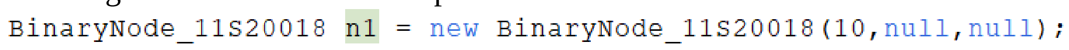


2c Diagram binary search tree (b) setelah nilai 9 ditambahkan

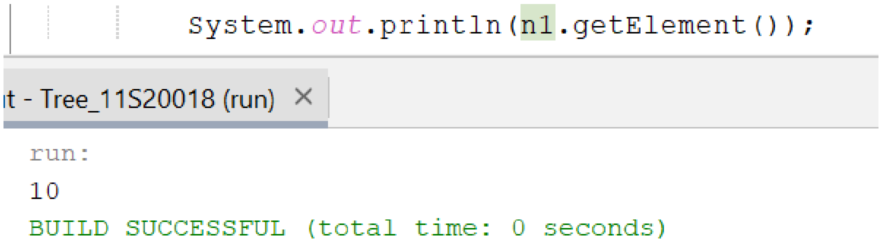
B. Implementasi

1. Binary Tree
   a) Kode kelas BinaryNode
   Link:
   BinaryTree_11S20018\BinaryNode_11S20018.java
   b) Kode kelas BinaryTree
   Link:
   BinaryTree_11S20018\BinaryTree_11S20018.java
   c) Kode untuk menguji method yang ada
   Link:
   BinaryTree_11S20018\TestBinaryNode_11S20018.java
   BinaryTree_11S20018\TestBinaryTree_11S20018.java

Method getElement: untuk mendapatkan nilai elemen



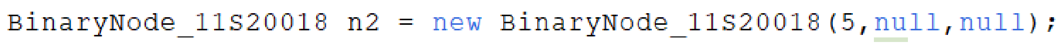
```
BinaryNode_11S20018 n1 = new BinaryNode_11S20018(10,null,null);
```



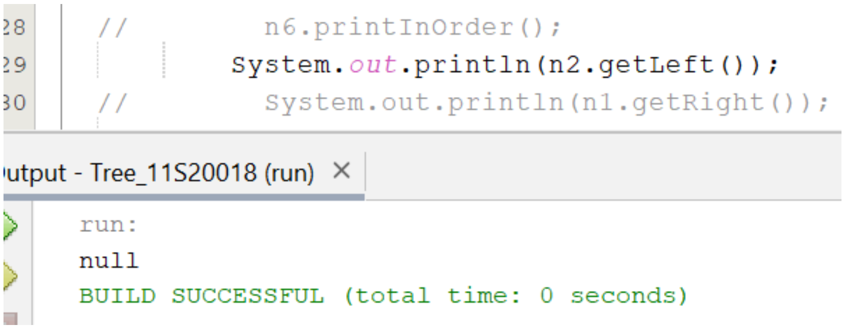
```
        System.out.println(n1.getElement());
```

```
run:
10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method getLeft: untuk mendapatkan node sebelah kiri dari sebuah node



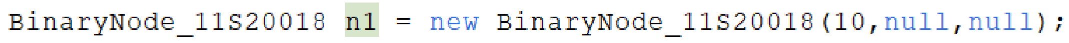
```
BinaryNode_11S20018 n2 = new BinaryNode_11S20018(5,null,null);
```



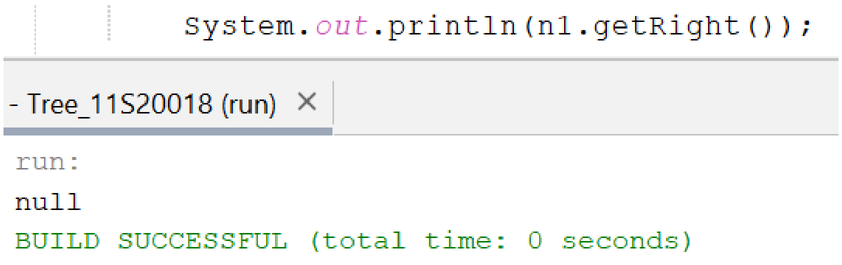
```
//          n6.printInOrder();
        System.out.println(n2.getLeft());
//          System.out.println(n1.getRight());
```

```
run:
null
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method getRight: untuk mendapatkan node sebelah kana dari sebuah node



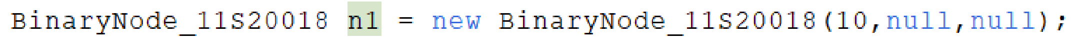
```
BinaryNode_11S20018 n1 = new BinaryNode_11S20018(10,null,null);
```



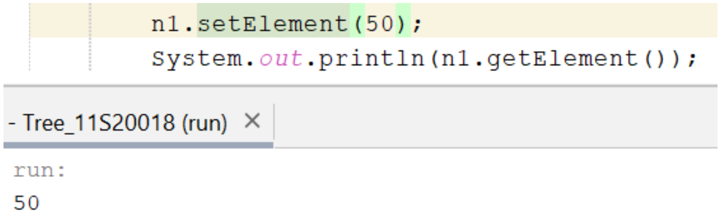
```
        System.out.println(n1.getRight());
```

```
run:
null
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method setElement: mengatur ulang nilai elemen



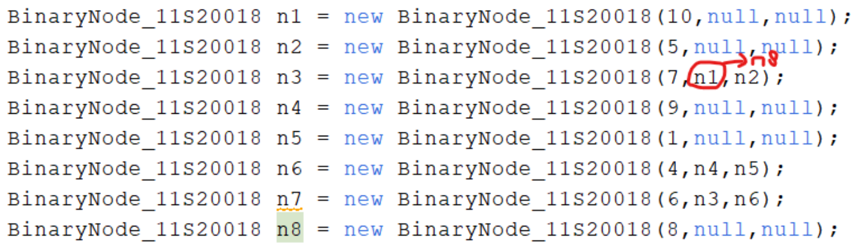
```
BinaryNode_11S20018 n1 = new BinaryNode_11S20018(10,null,null);
```



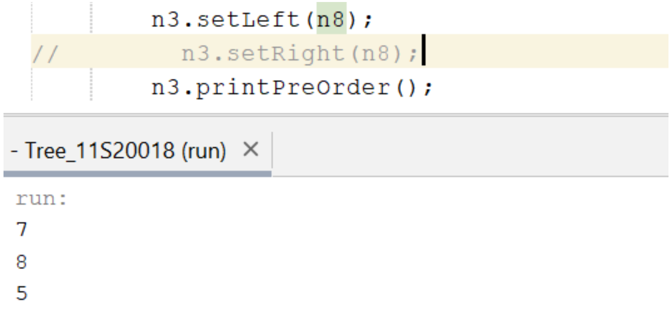
```
        n1.setElement(50);
        System.out.println(n1.getElement());
```

```
run:
50
```

Method setLeft: mengatur ulang nilai sebelah kiri dari sebuah node



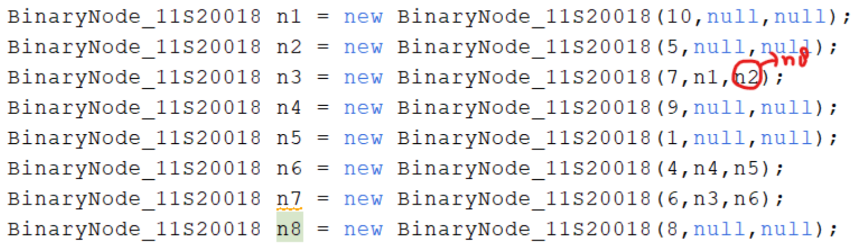
```
BinaryNode_11S20018 n1 = new BinaryNode_11S20018(10,null,null);
BinaryNode_11S20018 n2 = new BinaryNode_11S20018(5,null,null);
BinaryNode_11S20018 n3 = new BinaryNode_11S20018(7,n1,n2);
BinaryNode_11S20018 n4 = new BinaryNode_11S20018(9,null,null);
BinaryNode_11S20018 n5 = new BinaryNode_11S20018(1,null,null);
BinaryNode_11S20018 n6 = new BinaryNode_11S20018(4,n4,n5);
BinaryNode_11S20018 n7 = new BinaryNode_11S20018(6,n3,n6);
BinaryNode_11S20018 n8 = new BinaryNode_11S20018(8,null,null);
```



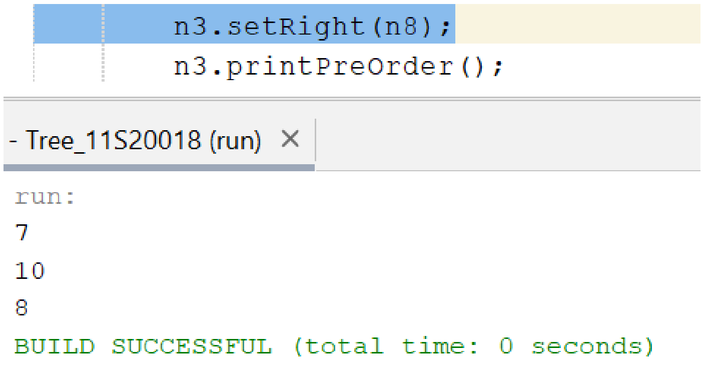
```
        n3.setLeft(n8);
//          n3.setRight(n8);
        n3.printPreOrder();
```

```
run:
7
8
5
```

Method setRight: mengatur ulang nilai sebelah kanan dari sebuah node



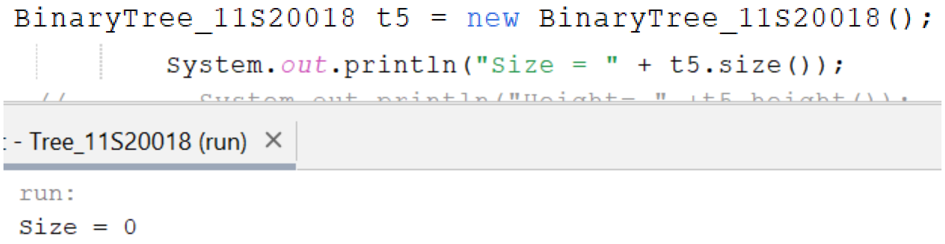
```
BinaryNode_11S20018 n1 = new BinaryNode_11S20018(10,null,null);
BinaryNode_11S20018 n2 = new BinaryNode_11S20018(5,null,null);
BinaryNode_11S20018 n3 = new BinaryNode_11S20018(7,n1,n2);
BinaryNode_11S20018 n4 = new BinaryNode_11S20018(9,null,null);
BinaryNode_11S20018 n5 = new BinaryNode_11S20018(1,null,null);
BinaryNode_11S20018 n6 = new BinaryNode_11S20018(4,n4,n5);
BinaryNode_11S20018 n7 = new BinaryNode_11S20018(6,n3,n6);
BinaryNode_11S20018 n8 = new BinaryNode_11S20018(8,null,null);
```



```
        n3.setRight(n8);
        n3.printPreOrder();
```

```
run:
7
10
8
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method size: mengembalikan ukuran dari subtree suatu node

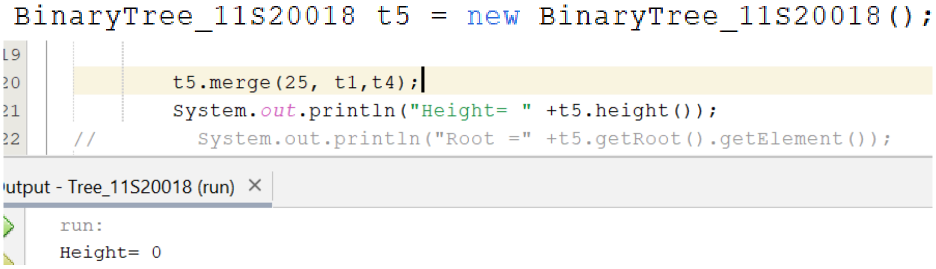


```
BinaryTree_11S20018 t5 = new BinaryTree_11S20018();
        System.out.println("Size = " + t5.size());
//          System.out.println("Height= " +t5.height());
```

: - Tree_11S20018 (run) ×

```
run:
Size = 0
```

Method height: mengembalikan tinggi dari subtree pada suatu node



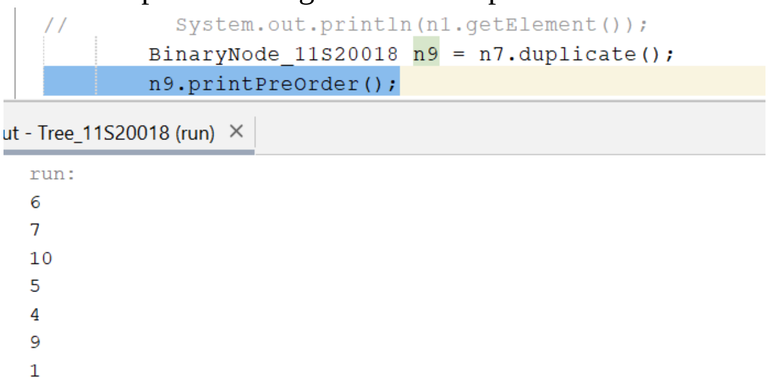
```
BinaryTree_11S20018 t5 = new BinaryTree_11S20018();

        t5.merge(25, t1,t4);
        System.out.println("Height= " +t5.height());
//          System.out.println("Root =" +t5.getRoot().getElement());
```

utput - Tree_11S20018 (run) ×

```
run:
Height= 0
```

Metode duplicate: mengembalikan duplikat tree

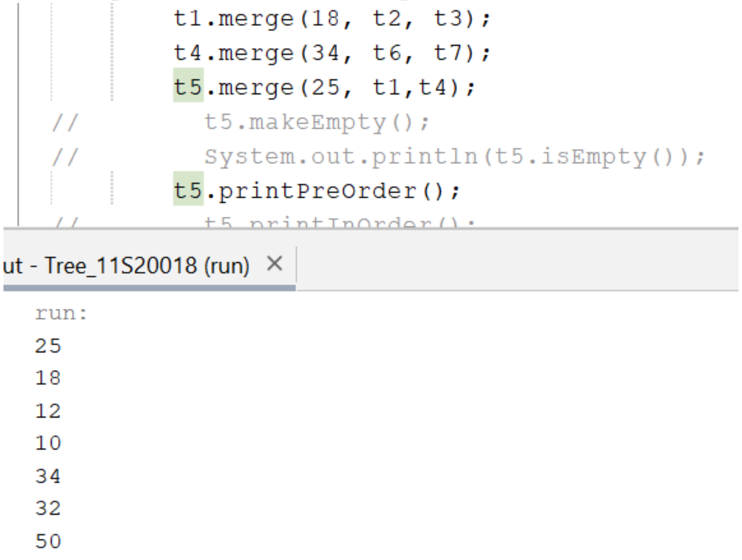


```
//          System.out.println(n1.getElement());
        BinaryNode_11S20018 n9 = n7.duplicate();
        n9.printPreOrder();
```

ut - Tree_11S20018 (run) ×

```
run:
6
7
10
5
4
9
1
```

Method printPreOrder: menampilkan tree tranversal secara preOrder

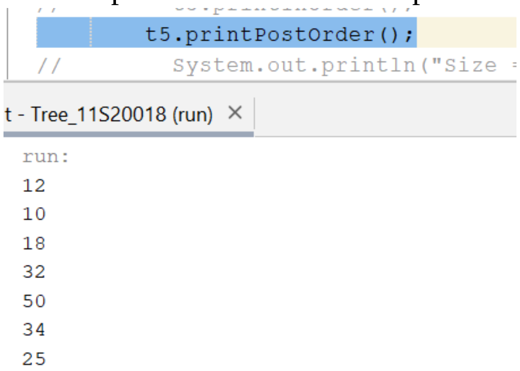


```
        t1.merge(18, t2, t3);
        t4.merge(34, t6, t7);
        t5.merge(25, t1,t4);
//          t5.makeEmpty();
//          System.out.println(t5.isEmpty());
        t5.printPreOrder();
//          t5.printInOrder();
```

ut - Tree_11S20018 (run) ×

```
run:
25
18
12
10
34
32
50
```

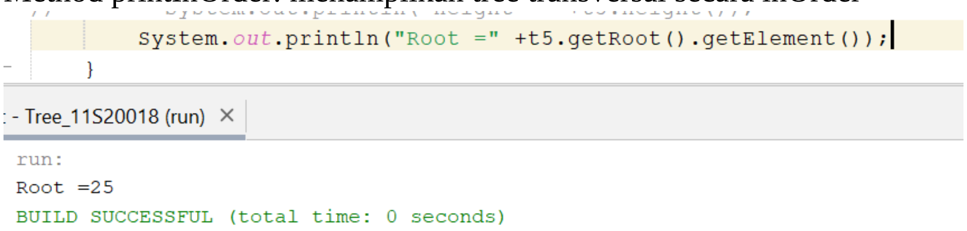
Method printPostOrder: menampilkan tree transversal secara postOrder



```
        t5.printPostOrder();
//      System.out.println("Size =
```

```
run:
12
10
18
32
50
34
25
```

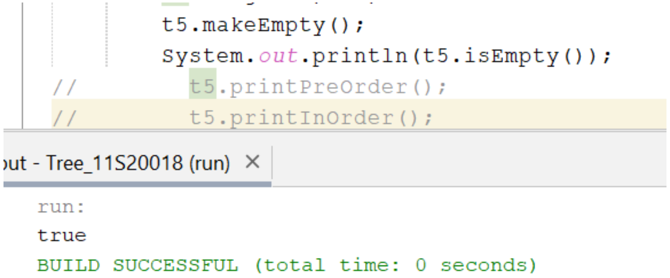
Method printInOrder: menampilkan tree transversal secara inOrder



```
        System.out.println("Root =" +t5.getRoot().getElement());
    }
```

```
run:
Root =25
BUILD SUCCESSFUL (total time: 0 seconds)
```

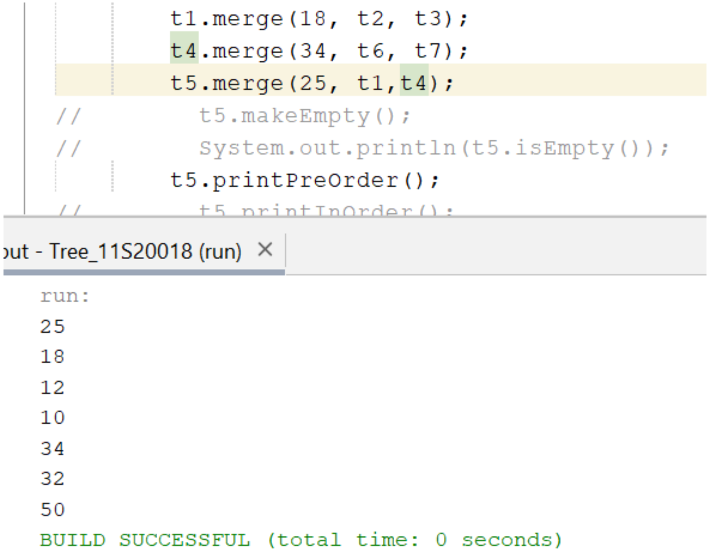
Method makeEmpty: mengosongkan nilai tree



```
        t5.makeEmpty();
        System.out.println(t5.isEmpty());
//        t5.printPreOrder();
//        t5.printInOrder();
```

```
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method merge: menggabungkan sebuah nilai menjadi nilai node

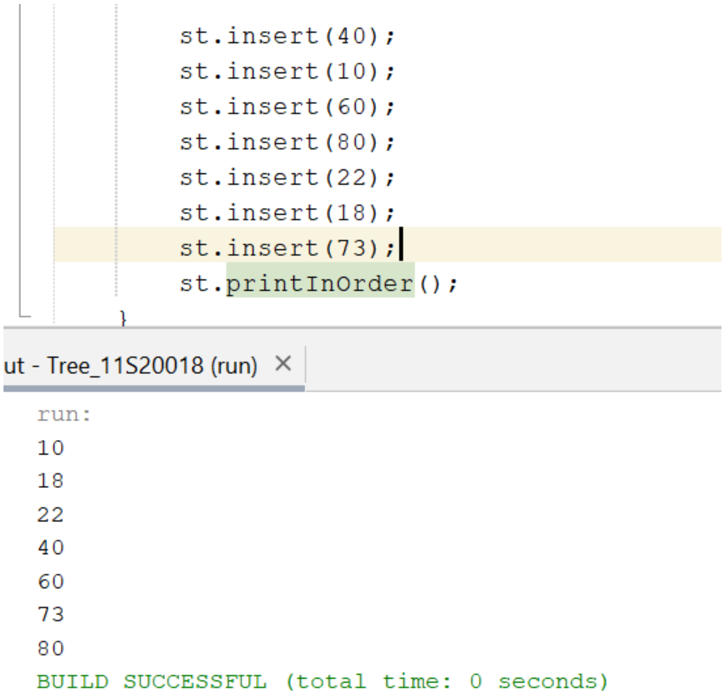


```
        t1.merge(18, t2, t3);
        t4.merge(34, t6, t7);
        t5.merge(25, t1,t4);
//        t5.makeEmpty();
//        System.out.println(t5.isEmpty());
        t5.printPreOrder();
//        t5.printInOrder();
```

```
run:
25
18
12
10
34
32
50
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Binary Search Tree
    a) Kode kelas BinaryNode
       Link:
       BinarySearchTree_11S20018\BinaryNode_11S20018.java
    b) Kode kelas BinarySearchTree
       Link:
       BinarySearchTree_11S20018\BinarySearchTree_11S20018.java
    c) Kode untuk menguji method yang ada
       Link:
       BinarySearchTree_11S20018\TestBinarySearchTree_11S20018.java

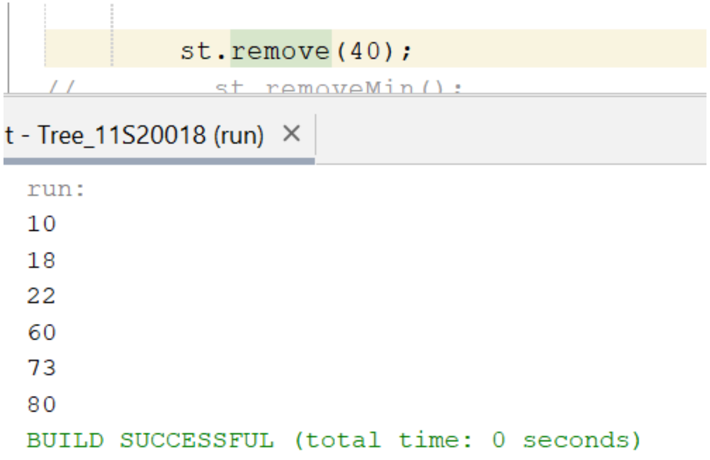
Method insert: menambahkan nilai



```
        st.insert(40);
        st.insert(10);
        st.insert(60);
        st.insert(80);
        st.insert(22);
        st.insert(18);
        st.insert(73);
        st.printInOrder();
    }
```

```
run:
10
18
22
40
60
73
80
BUILD SUCCESSFUL (total time: 0 seconds)
```

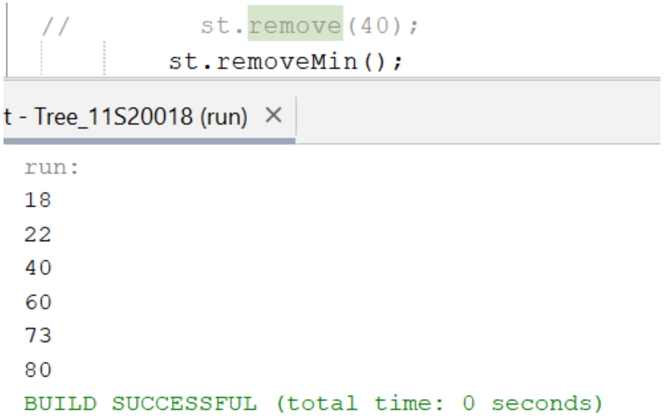
Method remove: menghapus nilai x pada subtree



```
        st.remove(40);
//        st.removeMin();
```

```
run:
10
18
22
60
73
80
BUILD SUCCESSFUL (total time: 0 seconds)
```

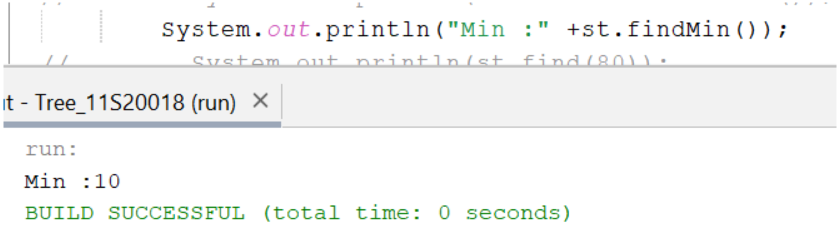
Method removeMin: menghapus nilai minimum pada subtree



```
//          st.remove(40);
        st.removeMin();
```

```
run:
18
22
40
60
73
80
BUILD SUCCESSFUL (total time: 0 seconds)
```

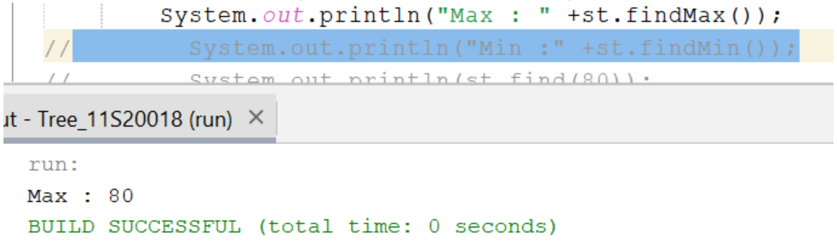
Method findMin: mengembalikan item dengan nilai terkecil pada subtree



```
        System.out.println("Min :" +st.findMin());
//          System.out.println(st.find(80));
```

```
run:
Min :10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method findMax: mengembalikan item dengan nilai terbesar pada subtree

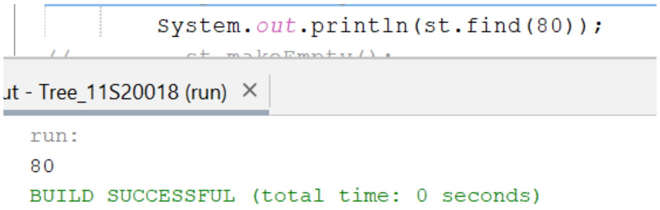


```
        System.out.println("Max : " +st.findMax());
//          System.out.println("Min :" +st.findMin());
//          System.out.println(st.find(80));
```

```
run:
Max : 80
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method find: mengembalikan item yang bernilai sama dengan x yang dicari

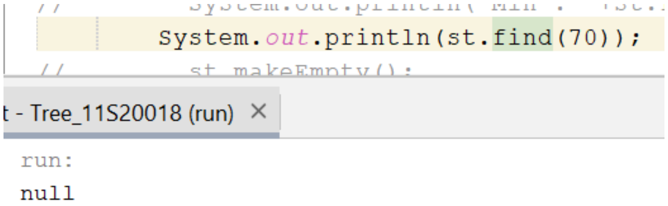
Jika ada:



```
        System.out.println(st.find(80));
//          st.makeEmpty();
```

```
run:
80
BUILD SUCCESSFUL (total time: 0 seconds)
```

Jika tidak ada:



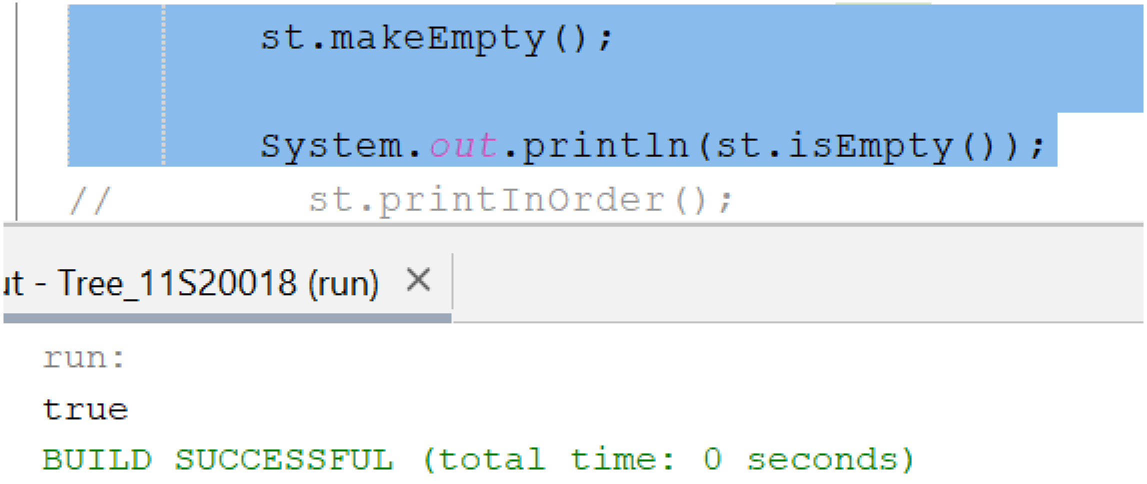
```
        System.out.println(st.find(70));
//          st.makeEmpty();
```

```
run:
null
```

Method elementAt: mengembalikan nilai elemen t (bersifat private)

Method makeEmpty: mengosongkan nilai tree

Method isEmpty: mengembalikan nilai true jika tree dalam keadaan kosong



```
        st.makeEmpty();

        System.out.println(st.isEmpty());
//          st.printInOrder();
```

```
run:
true
BUILD SUCCESSFUL (total time: 0 seconds)
```