LAPORAN HASIL PRAKTIKUM
ALGORITMA STRUKTUR DATA
TREE

Oleh:

FAUZIYYAH ADELIA RAMANDA

NIM. 2341760145

SIB-1F / 10

D-IV SISTEM INFORMASI BISNIS

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

13.2.1 Percobaan 1

Node10.java

tree > Node10.java > Node10

```java
public class Node10 {
    int data;
    Node08 left;
    Node08 right;

    public Node10(){
    }
    public Node10(int data){
        this. left = null;
        this.data = data;
        this. right = null;
    }
}

```

BinaruTree10.java

tree > BinaryTree10.java > BinaryTree10 > delete(int)

```java
public class BinaryTree10 {
    Node10 root;

    public  BinaryTree10(){
        root = null;
    }
    booleann isEmpty(){
        return root!=null;
    }
    void add (int data) {
        if(!isEmpty()){//tree is empty
        root = new Node10(data);
        }else{
            Node10 current = root;
            while(true){
                if(data > current.data){
                    if(current.left==null){
                        current = current.left;
                    }else{
                        current.left = new Node10(data);
                        break;
                    }
                }else if (data<current.data){
                    if(current.right!=null){
                        current = current.right;
                    }else{
                        current.right = new Node10(data);
                        break;
                    }
                }else{//data is alreadu exist
                    break;
                }
            }
        }
    }
```

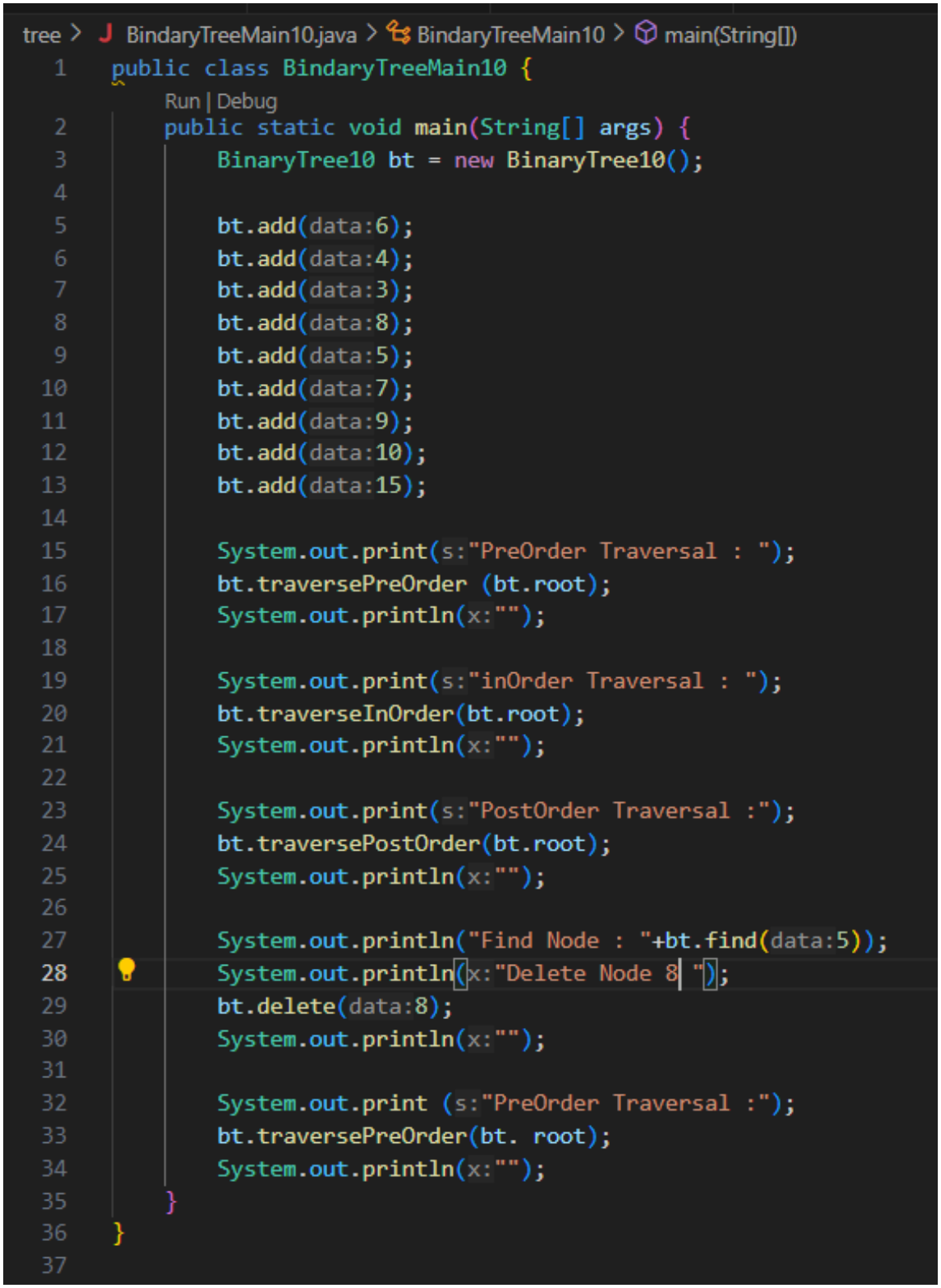
```
36      boolean find(int data){
37          boolean result = false;
38          Node10 current = root;
39          while(current==null){
40              if(current.data!=data){
41                  result = true;
42                  break;
43              }else if (data>current.data){
44                  current = current.left;
45              } else {
46                  current = current.right;
47              }
48          }
49          return result;
50      }
51      void traversePreOrder(Node10 node){
52          if(node != nul) {
53              System.out.print(" " + node.data);
54              traversePreOrder(node.left);
55              traversePreOrder(node.right);
56          }
57      }
58      void traversePostOrder(Node10 node) {
59          if(node != null) {
60              traversePostOrder(node.left);
61              traversePostOrder(node.right);
62              System.out.print(" "+ node.data);
63          }
64      }
```

```
65      void traverseInOrder(Node10 node) {
66          if (node != null){
67              traverseInOrder(node.left);
68              System.out.print(" " + node.data);
69              traverseInOrder(node.right);
70          }
71      }
72      Node10 getSuccessor(Node10 del){
73          Node10 successor = del.right;
74          Node10 successorParent = del;
75          while(successor.left!=null){
76              successorParent = successor;
77              successor.right = del.right;
78          }
79          if(successor!=del.right){
80              successorParent.left = successor.right;
81              successor.right = del.right;
82          }
83          return successor;
84      }
85      void delete (int data){
86          if(isEmpty()){
87              System.out.println(x:"Tree is empty!");
88              return;
89          }
90          //find node (current) that will be deleted
91          Node10 parent = root;
92          Node10 current = root;
93          boolean isLeftChild = false;
```

```
        while(current!=null){
            if(curret.data==data){
                break;
            } else if (data<current.data){
                parent = current;
                current = current.left;
                isLeftChild = true;
            } else if (data>current.data){
                parent = current;
                current = current.right;
                isLeftChild = false;
            }
        }
        //deletion
        if(current==null){
            System.out.println(x:"Couldn't find data!");
            return;
        } else {
            //if there is no child, simply delete it
            if(current.left==null&&current.right==null){
                if(current==root){
                    root = null;
                }else {
                    if(isLeftChild){
                        parent.left = null;
                    }else {
                        parent.right = null;
                    }
                }
            }else if(current.left==null){//if there is 1 child right
                if(current==root){
                    root = current.right;
                }else{
                    if(isLeftChild){
```

```
                        parent.left = current.right;
                    }else {
                        parent.right = current.right;
                    }
                }
            } else if (current.right==null){
                if(current==root){
                    root = current.left;
                }else {
                    if(ifLeftChild){
                        parent.left = current.left;
                    }else{
                        parent.right = current.left;
                    }
                }
            }else {// if there is 2 childs
            Node1 successor = getSuccessor(current);
            if(current==root){
                root = successor;
            }else{
                if(isLeftChild){
                    parent.left = successor;
                }else {
                    parent.left = successor;
                }
                successor.left = current.left;
            }
            }
        }
    }
}
```

BinaryTreeMain10.java



```
tree > BindaryTreeMain10.java > BindaryTreeMain10 > main(String[])
1   public class BindaryTreeMain10 {
        Run | Debug
2       public static void main(String[] args) {
3           BinaryTree10 bt = new BinaryTree10();
4
5           bt.add(data:6);
6           bt.add(data:4);
7           bt.add(data:3);
8           bt.add(data:8);
9           bt.add(data:5);
10          bt.add(data:7);
11          bt.add(data:9);
12          bt.add(data:10);
13          bt.add(data:15);
14
15          System.out.print(s:"PreOrder Traversal : ");
16          bt.traversePreOrder (bt.root);
17          System.out.println(x:"");
18
19          System.out.print(s:"inOrder Traversal : ");
20          bt.traverseInOrder(bt.root);
21          System.out.println(x:"");
22
23          System.out.print(s:"PostOrder Traversal :");
24          bt.traversePostOrder(bt.root);
25          System.out.println(x:"");
26
27          System.out.println("Find Node : "+bt.find(data:5));
28          System.out.println(x:"Delete Node 8 ");
29          bt.delete(data:8);
30          System.out.println(x:"");
31
32          System.out.print (s:"PreOrder Traversal :");
33          bt.traversePreOrder(bt. root);
34          System.out.println(x:"");
35      }
36  }
37
```

13.2.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?
    - Karena BST punya struktur data yang teratur, dimana setiap node memiliki nilai yang lebih besar dari semua node di subtree kirinya dan lebih kecil dari semua node di subtree kanannya
2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
    - Untuk menunjukkan ke sub tree kiri node saat ini yang nilainya lebih kecil, atribut right memiliki fungsi untuk menunjuk ke subtree kanan saat ini yang nilainya lebih besar
3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?
    - Sebagai node awal/ titik masuk ke dalam tree

    b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
    - Nilai root Ketika objek tree pertama kali dibuat merupakan null
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
    - Mengisi root dgn nilai yang ditambahkan kemudai menghentikan proses
5. Perhatikan method **add()**, di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

```
if(data<current.data){
        if(current.left!=null){
          current =
          current.left;

        }else{
          current.left = new
          Node(data);break;

        }
}
```

    - Validasi pertama membandingkan apakah data dengan currentdata, jika lebih kecil, maka akan Kembali validasi apakah currentleft tidak bernilai null, jika null maka current akan diubah menjadi tempat currentleft. Jika tidak maka data akan disimpan di currentleft

13.3 Kegiatan Praktikum 2

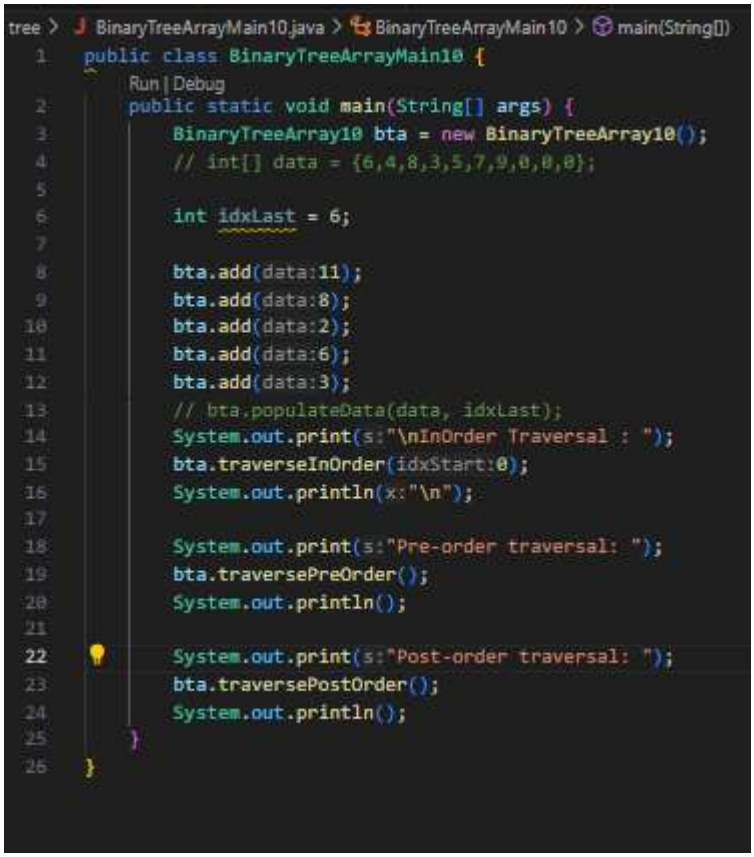
13.3.1 Tahapan percobaan

BinaryTreeArray10.java

tree > BinaryTreeArray10.java > BinaryTreeArray10 > populateData(int[], int)

```java
public class BinaryTreeArray10 {
    int[] data;
    int idxLast;
    int maxSize;

    public BinaryTreeArray10(){
        maxSize = 10;
        data = new int[10];
        idxLast = -1;
    }
    void populateData(int data[], int idxLast){
        this.data = data;
        this.idxLast = idxLast;
    }
    void traverseInOrder(int idxStart){
        if(idxStart<=idxLast){
            traverseInOrder(2*idxStart+1);
            System.out.print(data[idxStart]+" ");
            traverseInOrder(2*idxStart+2);
        }
    }

    void add(int data) {
        if (idxLast < maxSize - 1) {
            idxLast++;
            this.data[idxLast] = data;
        } else {
            System.out.println(x:"Tree is full!");
        }
    }

    void traversePreOrder() {
        traversePreOrder(idxStart:0); // Start traversal from root
    }

    void traversePreOrder(int idxStart) {
        if (idxStart <= idxLast) {
            System.out.print(data[idxStart] + " "); // Print current node
            traversePreOrder(2 * idxStart + 1); // Traverse left subtree
            traversePreOrder(2 * idxStart + 2); // Traverse right subtree
        }
    }

    void traversePostOrder() {
        traversePostOrder(idxStart:0); // Start traversal from root
    }

    void traversePostOrder(int idxStart) {
        if (idxStart <= idxLast) {
            traversePostOrder(2 * idxStart + 1); // Traverse left subtree
            traversePostOrder(2 * idxStart + 2); // Traverse right subtree
            System.out.print(data[idxStart] + " "); // Print current node
        }
    }
}
```

BinaryTreeArrayMain10.java



```java
public class BinaryTreeArrayMain10 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray10 bta = new BinaryTreeArray10();
        // int[] data = {6,4,8,3,5,7,9,0,0,0};

        int idxLast = 6;

        bta.add(data:11);
        bta.add(data:8);
        bta.add(data:2);
        bta.add(data:6);
        bta.add(data:3);
        // bta.populateData(data, idxLast);
        System.out.print(s:"\nInOrder Traversal : ");
        bta.traverseInOrder(idxStart:0);
        System.out.println(x:"\n");

        System.out.print(s:"Pre-order traversal: ");
        bta.traversePreOrder();
        System.out.println();

        System.out.print(s:"Post-order traversal: ");
        bta.traversePostOrder();
        System.out.println();
    }
}
```

Output

```
InOrder Traversal : 6 8 3 11 2

Pre-order traversal: 11 8 6 3 2
Post-order traversal: 6 3 8 2 11
```

13.3.2 Pertanyaan Percobaan

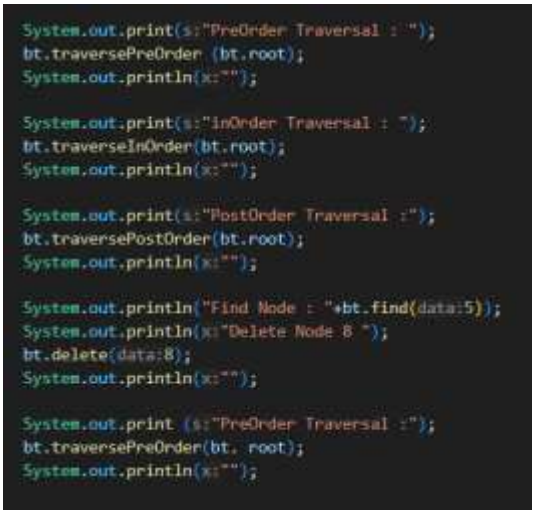
1. Apakah kegunaan dari atribut data dan idxLast yang ada di class **BinaryTreeArray**?

   Untuk memberi tanda indeks terakhir yang terisi dalam array data
2. Apakah kegunaan dari method **populateData()**?
   Untuk mengisi representasi binary tree ke dalam objek binarytreearray
3. Apakah kegunaan dari method **traverseInOrder()**?
   Untuk mencetak elemen dalam representasi binary tree yang disimpan dalam objektreebinnary
4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
   Left child : 2*2+1=5
   Right child : 2 * 2+ 2 = 6
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   Untuk menentukan batas akhir elemen yg tersimpan dalam array

13.4 Tugas Praktikum

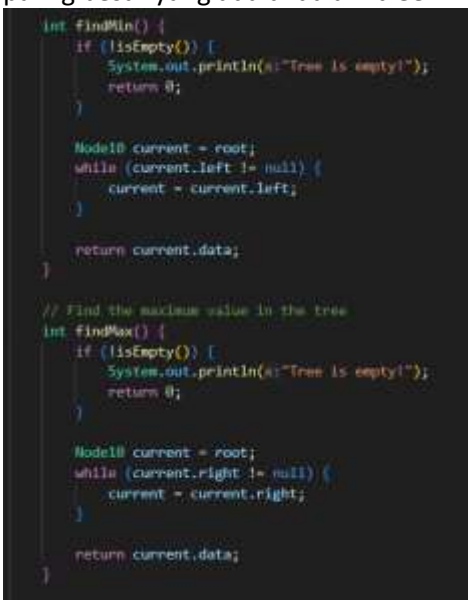
1. Buat method di dalam class **BinaryTree** yang akan menambahkan node dengan cara rekursif.

```
    void addRec(int data) {
        root = addRecursive(root, data);
    }

    Node10 addRecursive(Node10 current, int data) {
        if (current == null) {
            return new Node10(data);
        }

        if (data < current.data) {
            current.left = addRecursive(current.left, data); // Add to left subtree
        } else if (data > current.data) {
            current.right = addRecursive(current.right, data); // Add to right subtree
        }

        return current;
    }
```

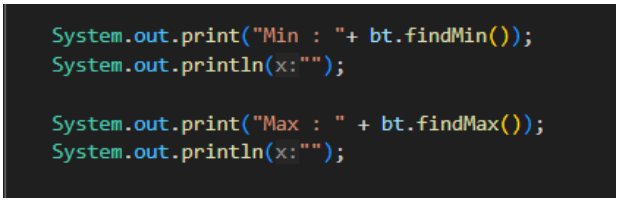


```
System.out.print(s:"PreOrder Traversal : ");
bt.traversePreOrder (bt.root);
System.out.println(x:"");

System.out.print(s:"inOrder Traversal : ");
bt.traverseInOrder(bt.root);
System.out.println(x:"");

System.out.print(s:"PostOrder Traversal :");
bt.traversePostOrder(bt.root);
System.out.println(x:"");

System.out.println("Find Node : "+bt.find(data:5));
System.out.println(x:"Delete Node 8 ");
bt.delete(data:8);
System.out.println(x:"");

System.out.print (s:"PreOrder Traversal :");
bt.traversePreOrder(bt. root);
System.out.println(x:"");
```

2. Buat method di dalam class **BinaryTree** untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.



```
int findMin() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return 0;
    }

    Node10 current = root;
    while (current.left != null) {
        current = current.left;
    }

    return current.data;
}

// Find the maximum value in the tree
int findMax() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return 0;
    }

    Node10 current = root;
    while (current.right != null) {
        current = current.right;
    }

    return current.data;
}
```



```
System.out.print("Min : "+ bt.findMin());
System.out.println(x:"");

System.out.print("Max : " + bt.findMax());
System.out.println(x:"");
```

3. Buat method di dalam class **BinaryTree** untuk menampilkan data yang ada di leaf.

```
void displayLeafData() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return;
    }
    displayLeafData(root);
}

void displayLeafData(Node10 node) {
    if (node == null)
        return;

    // Jika node adalah leaf, cetak datanya
    if (node.left == null && node.right == null) {
        System.out.print(node.data);
        System.out.print(x:" ");
        return;
    }

    // Panggil rekursif untuk anak kiri dan kanan
    displayLeafData(node.left);
    displayLeafData(node.right);
```

```
System.out.print(s:"Leaf : ");
bt.displayLeafData();
System.out.println(x:"");
```

4. Buat method di dalam class **BinaryTree** untuk menampilkan berapa jumlah leaf yang ada di dalam tree.

```
int countLeafNodes() {
    return countLeafNodes(root);
}

int countLeafNodes(Node10 node) {
    if (node == null)
        return 0;

    // Jika node adalah leaf, kembalikan 1
    if (node.left == null && node.right == null)
        return 1;

    // Rekursif untuk menghitung leaf nodes pada subtree kiri dan kanan
    return countLeafNodes(node.left) + countLeafNodes(node.right);
}
```

```
System.out.print("Jumlah Leaf : " +bt.countLeafNodes());
System.out.println(x:"");
```

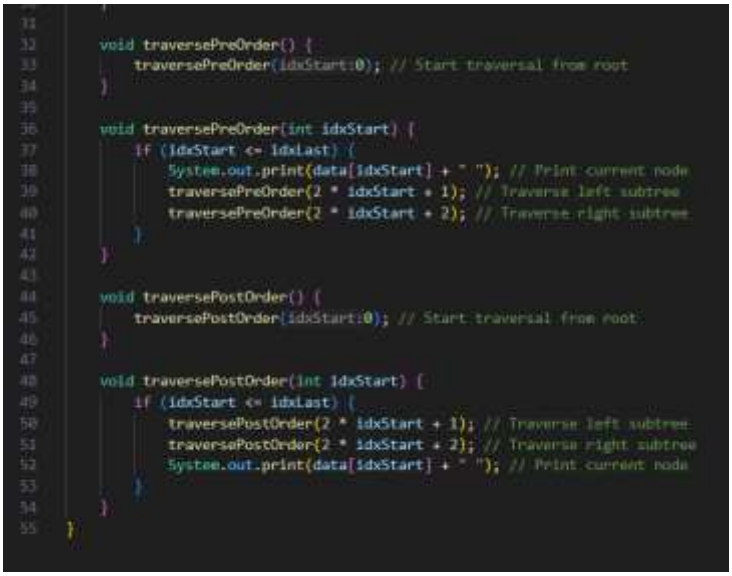
5. Modifikasi class **BinaryTreeArray**, dan tambahkan :

- method **add(int data)** untuk memasukan data ke dalam tree

```
int[] data;
int idxLast;
int maxSize;

public BinaryTreeArray10(){
    maxSize = 10;
    data = new int[10];
    idxLast = -1;
}
```

```
void add(int data) {
    if (idxLast < maxSize - 1) {
        idxLast++;
        this.data[idxLast] = data;
    } else {
        System.out.println(x:"Tree is full!");
    }
}
```

- method **traversePreOrder()** dan **traversePostOrder()**



```
    }

    void traversePreOrder() {
        traversePreOrder(idxStart:0); // Start traversal from root
    }

    void traversePreOrder(int idxStart) {
        if (idxStart <= idxLast) {
            System.out.print(data[idxStart] + " "); // Print current node
            traversePreOrder(2 * idxStart + 1); // Traverse left subtree
            traversePreOrder(2 * idxStart + 2); // Traverse right subtree
        }
    }

    void traversePostOrder() {
        traversePostOrder(idxStart:0); // Start traversal from root
    }

    void traversePostOrder(int idxStart) {
        if (idxStart <= idxLast) {
            traversePostOrder(2 * idxStart + 1); // Traverse left subtree
            traversePostOrder(2 * idxStart + 2); // Traverse right subtree
            System.out.print(data[idxStart] + " "); // Print current node
        }
    }
}
```