```
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class CreateBinaryApna {


    static class Node {
        int data;
        Node left,right;

        public Node(int data){

            this.data=data;
            this.left=null;
            this.right=null;
        }

    }

    static class BinaryTree{

        int idx = -1;
                public Node buildTree(int [] nodes) {
                    idx++;
                    if (nodes[idx]==-1) {
                        return null;
                    }

                   Node newNode=new Node(nodes[idx]);
                   newNode.left= buildTree(nodes);
                   newNode.right=buildTree(nodes);


                   return newNode;
                }

                public void preOrder(Node root){

                    if (root==null) {
                       return;
                    }
                    System.out.print(root.data+" ");
                    preOrder(root.left);
                    preOrder(root.right);


                }

                public void inOrder(Node root){

                    if (root==null) {
                        return;
                    }

                    preOrder(root.left);
                    System.out.print(root.data+" ");
                    preOrder(root.right);
                }

                public void postOrder(Node root){

                    if (root==null) {
                        return;
                    }

                    postOrder(root.left);
                    postOrder(root.right);
```

```
            System.out.print(root.data+" ");
        }

        public int countofNodes(Node root){

                  if (root==null) {
                   return 0;
                  }

            return countofNodes(root.left)+countofNodes(root.right)+1;

         }

         public int sumofNodes(Node root){

            if (root==null) {
             return 0;
            }

     return sumofNodes(root.left)+sumofNodes(root.right)+root.data;

  }


  public int height(Node root){

    if (root==null) {
     return 0;
    }

return Math.max(height(root.left), height(root.right))+1;

}

int maxi=0;
public int diameter(Node root){

    if (root==null) {
        return 0;
    }

    int lh=height(root.left);
    int rh=height(root.right);

    maxi=Math.max(lh+rh+1, maxi);

    diameter(root.left);
    diameter(root.right);
    return maxi;


}

public int iterativeDiameter(Node root){

    Stack<Node> stack= new Stack<>();
    HashMap<Node, Integer> map= new HashMap<>();

    int diameter=0;

    if (root==null) {
       return 0;
    }
    stack.add(root);

    while (!stack.isEmpty()) {

        Node current=stack.peek();
```

```
                if (current.left!=null && !map.containsKey(current.left)) {
                   stack.push(current.left);
                }

                else if (current.right!=null && !map.containsKey(current.right)) {
                    stack.push(current.right);
                }

                else{

                    stack.pop();
                    int ln=map.getOrDefault(current.left, 0);
                    int rn=map.getOrDefault(current.right, 0);

                    map.put(current, 1+Math.max(ln, rn));

                    diameter=Math.max(diameter, ln+rn);

                }
            }

            return diameter+1;

        }

// Use your custom Node class here instead of TreeNode
public int countNodes(Node root) {
    if (root == null) {
        return 0; // If the node is null, return 0
    }
    // Recursively count nodes in the left and right subtrees, plus 1 for the current node
    return 1 + countNodes(root.left) + countNodes(root.right);
}


        public boolean subTree(Node root, Node given){

            if (given==null) {
                return true;
            }

            if (root==null) {
                return false;
            }
            HashMap<Node, Boolean> map= new HashMap<>();
            Stack<Node> stack= new Stack<>();
            stack.add(root);
            map.put(root, true);

            while (!stack.isEmpty()) {

                Node current=stack.peek();

                if (current.left!=null && !map.containsKey(current.left)) {
                    stack.push(current.left);
                    map.put(current.left, true);
                }
                else if (current.right!=null && !map.containsKey(current.right)) {
                    stack.push(current.right);
                    map.put(current.right, true);
                }

                else{

                   stack.pop();

                //    System.out.println("Current Data: "+ given.data+ "left: "+given.left.data+
"right: "+ given.right.data );
                   if (current.left!=null && current.right!=null && current.data==given.data &&
```

```
current.left.data==given.left.data && current.right.data==given.right.data) {

                    return true;

                   }

               }

           }

           return false;


       }




       public int kthLevelsum(Node root, int k){

       Queue<Node> queue= new LinkedList<>();
       queue.add(root);
       queue.add(null);
       int countLevel=1;
       int sum=0;


       while (!queue.isEmpty()) {

           Node curr=queue.poll();

           if (curr!=null) {

               if (countLevel==k) {
                   sum=sum+curr.data;
               }

               if (curr.left!=null) {
                   queue.add(curr.left);
               }

               if (curr.right!=null) {
                   queue.add(curr.right);
               }
           }

           else if (!queue.isEmpty()) {
               countLevel++;
               queue.add(null);

           }


       }
               return sum;

       }

           public void levelOrder(Node root){

                           Queue<Node> q= new LinkedList<>();
                           q.add(root);
                           q.add(null);

                           while (!q.isEmpty()) {

                               Node curr=q.remove();
                               if (curr==null) {
                                   System.out.println();
```

```
                                if (q.isEmpty()) {
                                    break;
                                }
                                else{
                                    q.add(null);
                                }
                            }

                            else{

                                System.out.print(curr.data+" ");
                                if (curr.left!=null) {
                                    q.add(curr.left);
                                }

                                if (curr.right!=null) {
                                    q.add(curr.right);
                                }


                            }

                        }
                    }
                }


                public static void main(String[] args) {

                    // int [] nodes={1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1};
                    // int [] nodes={1, 2, 6, 7, 8, -1, -1, -1, -1, 4, 5, 9, -1, -1, -1, -1,
3, -1, -1};
                    int[] nodes = {10, 5, 1, -1, -1, 7, -1, -1, 15, 12, -1, -1, 18, -1, -1};
                    int[] nodes1 = {15, 12, -1, -1, 18, -1, -1};
                    BinaryTree bnt= new BinaryTree();
                    Node root=bnt.buildTree(nodes);
                    bnt.idx=-1;
                Node root1=bnt.buildTree(nodes1);
            //      System.out.println("Hello: "+root.left.left.data);

            //     System.out.println("Resultant value preOrder: ");
            //     // System.out.println(root.data);
            //    bnt.preOrder(root);
            //    System.out.println("Resultant value InOrder: ");
            //    bnt.inOrder(root);
            //    System.out.println("Resultant value postOrder: ");
            //    bnt.postOrder(root);
            //    System.out.println("Resultant value levelOrder: ");
            //    bnt.levelOrder(root);
            //    System.out.println();
            //    System.out.print("Total Number of nodes: "+ bnt.countofNodes(root));
               System.out.println();
               System.out.print("Sum of nodes: "+ bnt.sumofNodes(root));
               System.out.println();
               System.out.print("Height of nodes: "+ bnt.height(root));
               System.out.println();
               System.out.print("Diameter of node: "+ bnt.diameter(root));
               System.out.println();
               System.out.print("Diameter of node: "+ bnt.iterativeDiameter(root));
               System.out.println();
               System.out.print("Subtree present in node or not: "+ bnt.subTree(root, root1));
               System.out.println();
               System.out.print("kth Level sum: "+    bnt.kthLevelsum(root, 2));


    }
}
```