Main.java

```
/**
 *
 * Solomiya Pobutska
 * CISC 3130
 * Assignment #4
 *
 * */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Main {

    public static void main(String args[]) throws IOException {
        BinaryTree tree = new BinaryTree();
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        System.out.println("Enter the number of elements in the array :");
        int n = Integer.parseInt(br.readLine());
        int[] inputArray = new int[n];
        String strArray[] = br.readLine().split(" ");

        for (int i = 0; i < strArray.length - 1; i++) {
            inputArray[i] = Integer.parseInt(strArray[i]);
            tree.insert(inputArray[i]);
        }


        System.out.println("\nInorder Traversal := ");
        tree.printInorder(tree.root);
        System.out.println("\nPost Order Traversal := ");
        tree.printPostorder(tree.root);
        System.out.println("\nPre Order Traversal := ");
        tree.printPreorder(tree.root);


        System.out.println("\nCount := " + tree.count(tree.root));
        System.out.println("Children := ");
        tree.children(tree.root);



        tree.insert(21);
        tree.deleteKey(1);
        tree.insert(0);
        tree.deleteKey(10);
        tree.deleteKey(11);
        tree.deleteKey(5);
        tree.deleteKey(200);
        tree.insert(10);


        System.out.println("Inorder Traversal := ");
        tree.printInorder(tree.root);
        System.out.println("\nPost Order Traversal := ");
        tree.printPostorder(tree.root);
        System.out.println("\nPre Order Traversal := ");
        tree.printPreorder(tree.root);


        System.out.println("\nCount := " + tree.count(tree.root));
        System.out.println("Children := ");
        tree.children(tree.root);
        br.close();
    }
}
```

BinaryTree.java

```
public class BinaryTree {
    Node root;

    public BinaryTree(int key) {
        root = new Node(key);
    }

    BinaryTree() {
        root = null;
    }

    /*
     * Given a binary tree, print its nodes according to the “bottom-up"
     * Post-order traversal.
     */
    void printPostorder(Node node) {
        if (node == null)
            return;

        // first recur on left subtree
        printPostorder(node.left);

        // then recur on right subtree
        printPostorder(node.right);

        // now deal with the node
        System.out.print(node.key + " ");
    }

    /* Given a binary tree, print its nodes in in-order */
    void printInorder(Node node) {
        if (node == null)
            return;
        /* first recur on left child */
        printInorder(node.left);

        /* then print the data of node */
        System.out.print(node.key + " ");

        /* now recur on right child */
        printInorder(node.right);
    }

    /* Given a binary tree, print its nodes in pre-order */
    void printPreorder(Node node) {
        if (node == null)
            return;

        /* first print data of node */
        System.out.print(node.key + " ");

        /* then recur on left subtree */
        printPreorder(node.left);

        /* now recur on right subtree */
        printPreorder(node.right);
    }

    /**
     * Insert a node
     */
    void insert(int key) {
        root = insertRec(root, key);
    }

    Node insertRec(Node root, int key) {
        /* If the tree is empty, return a new node */
        if (root == null) {
            root = new Node(key);
            return root;
        }
        /* Otherwise, recur down the tree */
        if (key < root.key)
            root.left = insertRec(root.left, key);
        else if (key > root.key)
            root.right = insertRec(root.right, key);
        /* return the (unchanged) node pointer */
        return root;
    }

    /**
     * Delete a node.
     */
    void deleteKey(int key) {
        root = deleteRec(root, key);
    }
```

```
Node deleteRec(Node root, int key) {
    /* Base Case: If the tree is empty */
    if (root == null)
        return root;

    /* Otherwise, recur down the tree */
    if (key < root.key)
        root.left = deleteRec(root.left, key);
    else if (key > root.key)
        root.right = deleteRec(root.right, key);

        // if key is same as root's key, then This is the node
        // to be deleted
    else {
        // node with only one child or no child
        if (root.left == null)
            return root.right;
        else if (root.right == null)
            return root.left;

        // node with two children: Get the in-order successor (smallest
        // in the right subtree)
        root.key = minValue(root.right);

        // Delete the in-order successor
        root.right = deleteRec(root.right, root.key);
    }

    return root;
}

// A utility function to search a given key in BST
public Node search(Node root, int key) {
    // Base Cases: root is null or key is present at root
    if (root == null || root.key == key)
        return root;

    // val is greater than root's key
    if (root.key > key)
        return search(root.left, key);

    // val is less than root's key
    return search(root.right, key);
}

   int minValue(Node root) {
    int minv = root.key;
    while (root.left != null) {
        minv = root.left.key;
        root = root.left;
    }
    return minv;
}
```

```
       int count(Node node) {

        if(node == null)
            return 0;
        int countNumber = 1;
        if (node.left != null)
            countNumber += count(node.left); // Recur the left subtree
        if (node.right != null)
            countNumber += count(node.right); // Recur the right subtree
        return countNumber;
    }

       void children(Node node) {
        if (node == null)
            return;

        Node temp = node;
        /* first print data of node */
        System.out.println("Node with key " + node.key + " :=> " +
(count(temp) - 1));

        /* then recur on left subtree */
        children(node.left);

        /* now recur on right subtree */
        children(node.right);
    }

}
```

Node.java

```
public class Node {
    int key;
    Node left, right;

    public Node(int key) {
        this.key = key;
        left = right = null;
    }
}
```

Output:

Set #1 :

```
Enter the number of elements in the array :
21
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -999
Inorder Traversal :=
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Post Order Traversal :=
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Pre Order Traversal :=
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Count := 20
Children :=
Node with key 1 :=> 19
Node with key 2 :=> 18
Node with key 3 :=> 17
Node with key 4 :=> 16
Node with key 5 :=> 15
Node with key 6 :=> 14
Node with key 7 :=> 13
Node with key 8 :=> 12
Node with key 9 :=> 11
Node with key 10 :=> 10
Node with key 11 :=> 9
Node with key 12 :=> 8
Node with key 13 :=> 7
Node with key 14 :=> 6
Node with key 15 :=> 5
Node with key 16 :=> 4
Node with key 17 :=> 3
Node with key 18 :=> 2
Node with key 19 :=> 1
Node with key 20 :=> 0
Inorder Traversal :=
0 2 3 4 6 7 8 9 10 12 13 14 15 16 17 18 19 20 21
Post Order Traversal :=
0 10 21 20 19 18 17 16 15 14 13 12 9 8 7 6 4 3 2
Pre Order Traversal :=
2 0 3 4 6 7 8 9 12 10 13 14 15 16 17 18 19 20 21
Count := 19
Children :=
Node with key 2 :=> 18
Node with key 0 :=> 0
Node with key 3 :=> 16
Node with key 4 :=> 15
Node with key 6 :=> 14
Node with key 7 :=> 13
Node with key 8 :=> 12
Node with key 9  :=> 11
Node with key 12 :=> 10
Node with key 10 :=> 0
Node with key 13 :=> 8
Node with key 14 :=> 7
Node with key 15 :=> 6
Node with key 16 :=> 5
Node with key 17 :=> 4
Node with key 18 :=> 3
Node with key 19 :=> 2
Node with key 20 :=> 1
Node with key 21 :=> 0
```

Set #2 :

```
Enter the number of elements in the array :
4
3 1 5 -999
Inorder Traversal :=
1 3 5
Post Order Traversal :=
1 5 3
Pre Order Traversal :=
3 1 5
Count := 3
Children :=
Node with key 3 :=> 2
Node with key 1 :=> 0
Node with key 5 :=> 0
Inorder Traversal :=
5
Post Order Traversal :=
5
Pre Order Traversal :=
5
Count := 1
Children :=
Node with key 5 :=> 0
```

Set #3 :

```
Enter the number of elements in the array :
1
-999
Inorder Traversal :=
Post Order Traversal :=
Pre Order Traversal :=
Count := 0
Children :=
Inorder Traversal :=
Post Order Traversal :=
Pre Order Traversal :=
Count := 0
Children :=
```

Set #4 :

```
Enter the number of elements in the array :
2
2 -999
Inorder Traversal :=
2
Post Order Traversal :=
2
Pre Order Traversal :=
2
Count := 1
Children :=
Node with key 2 :=> 0
Inorder Traversal :=
Post Order Traversal :=
Pre Order Traversal :=
Count := 0
Children :=
```

Set #5:

```
Enter the number of elements in the array :
16
11 25 75 12 37 60 90 8 15 32 45 50 67 97 95 -999
Inorder Traversal :=
8 11 12 15 25 32 37 45 50 60 67 75 90 95 97
Post Order Traversal :=
8 15 12 32 50 45 67 60 37 95 97 90 75 25 11
Pre Order Traversal :=
11 8 25 12 15 75 37 32 60 45 50 67 90 97 95
Count := 15
Children :=
Node with key 11 :=> 14
Node with key 8 :=> 0
Node with key 25 :=> 12
Node with key 12 :=> 1
Node with key 15 :=> 0
Node with key 75 :=> 9
Node with key 37 :=> 5
Node with key 32 :=> 0
Node with key 60 :=> 3
Node with key 45 :=> 1
Node with key 50 :=> 0
Node with key 67 :=> 0
Node with key 90 :=> 2
Node with key 97 :=> 1
Node with key 95 :=> 0
Inorder Traversal :=
8 11 12 25 32 40 45 50 60 67 75 90 95 97 99
Post Order Traversal :=
8 12 40 32 50 67 60 45 95 99 97 90 75 25 11
Pre Order Traversal :=
11 8 25 12 75 45 32 40 60 50 67 90 97 95 99
Count := 15
Children :=
Node with key 11 :=> 14
Node with key 8 :=> 0
Node with key 25 :=> 12
Node with key 12 :=> 0
Node with key 75 :=> 10
Node with key 45 :=> 5
Node with key 32 :=> 1
Node with key 40 :=> 0
Node with key 60 :=> 2
Node with key 50 :=> 0
Node with key 67 :=> 0
Node with key 90 :=> 3
Node with key 97 :=> 2
Node with key 95 :=> 0
Node with key 99 :=> 0
```

Set #6:

```
Enter the number of elements in the array :
10
50 40 60 30 70 20 80 10 90 -999
Inorder Traversal :=
10 20 30 40 50 60 70 80 90
Post Order Traversal :=
10 20 30 40 90 80 70 60 50
Pre Order Traversal :=
50 40 30 20 10 60 70 80 90
Count := 9
Children :=
Node with key 50 :=> 8
Node with key 40 :=> 3
Node with key 30 :=> 2
Node with key 20 :=> 1
Node with key 10 :=> 0
Node with key 60 :=> 3
Node with key 70 :=> 2
Node with key 80 :=> 1
Node with key 90 :=> 0
Inorder Traversal :=
10 20 30 40 50 60 70 80 90
Post Order Traversal :=
10 20 30 40 90 80 70 60 50
Pre Order Traversal :=
50 40 30 20 10 60 70 80 90
Count := 9
Children :=
Node with key 50 :=> 8
Node with key 40 :=> 3
Node with key 30 :=> 2
Node with key 20 :=> 1
Node with key 10 :=> 0
Node with key 60 :=> 3
Node with key 70 :=> 2
Node with key 80 :=> 1
Node with key 90 :=> 0
```

Set #7:

```
Enter the number of elements in the array :
6
30 40 20 10 50 -999
Inorder Traversal :=
10 20 30 40 50
Post Order Traversal :=
10 20 50 40 30
Pre Order Traversal :=
30 20 10 40 50
Count := 5
Children :=
Node with key 30 :=> 4
Node with key 20 :=> 1
Node with key 10 :=> 0
Node with key 40 :=> 1
Node with key 50 :=> 0
Inorder Traversal :=
10 20 30 40 50
Post Order Traversal :=
10 20 50 40 30
Pre Order Traversal :=
30 20 10 40 50
Count := 5
Children :=
Node with key 30 :=> 4
Node with key 20 :=> 1
Node with key 10 :=> 0
Node with key 40 :=> 1
Node with key 50 :=> 0
```