

- J Binary Tree Representation.java
- J Inorder Traversal of Binary Tree.java
- J Iterative Inorder Traversal of Binary Tree.java
- J Iterative Preorder Traversal of Binary Tree.java
- J Level order Traversal OR Level order traversal in spiral form.java
- J Post-order Traversal of Binary Tree using 1 stack.java
- J Post-order Traversal of Binary Tree using 2 stack.java
- J Post-order Traversal of Binary Tree.java
- J Preorder Traversal of Binary Tree.java
- J Preorder, Inorder, and Postorder Traversal in one Traversal.java
- Traversals.iml

```

1  class Node{
2      int data;
3      Node left;
4      Node right;
5      Node(int key){
6          this.data = key;
7      }
8  }
9

```

Codeium: Refactor | Explain

```

10 public class Binary_Tree_Representation {
11     Codeium: Refactor | Explain | Generate Javadoc | X
12     public static void main(String[] args) {
13         /*
14             Real World Representation the below tree into the code.
15             1
16             /\
17            2 3
18           /\ /
19          4 5 6
20             \
21             7
22         */
23         Node root = new Node(1);
24         root.left = new Node(2);
25         root.right = new Node(3);
26         root.left.left = new Node(4);
27         root.left.right = new Node(5);
28         root.right.left = new Node(6);
29         root.right.left.right = new Node(7);
30     }
31 }

```

Codeium: Refactor | Explain

```
public class Inorder_Traversal_of_Binary_Tree {
```

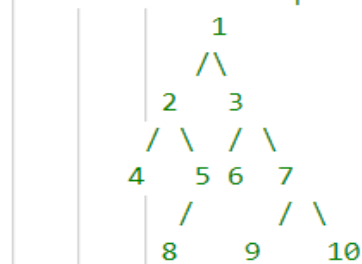
Codeium: Refactor | Explain | Generate Javadoc | ✕

```
    public static void inorderTraversal(Node node){
        if(node == null){
            return;
        }
        inorderTraversal(node.left);
        System.out.print(node.data + " ");
        inorderTraversal(node.right);
    }
}
```

Codeium: Refactor | Explain | Generate Javadoc | ✕

```
public static void main(String[] args) {
```

```
    /*
       Real World Representation the below tree into the code.
```



```
    */
    Node root = new Node(1);
    root.left = new Node(2);
    root.right = new Node(3);
    root.left.left = new Node(4);
    root.left.right = new Node(5);
    root.left.right.left = new Node(8);
    root.right.left = new Node(6);
    root.right.right = new Node(7);
    root.right.right.left = new Node(9);
    root.right.right.right = new Node(10);
    inorderTraversal(root);
}
```

```
}
```

> Task :Inorder_Traversal_of_Binary_Tree.main()

4 2 8 5 1 6 3 9 7 10

```

Codeium: Refactor | Explain
31  ✓ public class Iterative_Inorder_Traversa_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | ✕
32  ✓     public static List<Integer> inorderTraversal(Node root) {
33          ArrayList<Integer> res = new ArrayList<>();
34          Stack<Node> st = new Stack<>();
35          Node node = root;
36  ✓     while(true){
37  ✓         if(node != null){
38             st.push(node);
39             node = node.left;
40         }
41  ✓         else{
42             if(st.isEmpty()) break;
43             node = st.pop();
44             res.add(node.data);
45             node = node.right;
46         }
47     }
48     return res;
49 }

Codeium: Refactor | Explain | Generate Javadoc | ✕
50  public static void main(String[] args) {
51      /*
52          Real World Representation the below tree into the code.
53              1
54             /\
55            2  3
56           /\ /\
57          4  5 6 7
58         /\ /\
59        8  9 10
60      */
61      Node root = new Node(1);
62      root.left = new Node(2);
63      root.right = new Node(3);
64      root.left.left = new Node(4);
65      root.left.right = new Node(5);
66      root.left.right.left = new Node(8);
67      root.right.left = new Node(6);
68      root.right.right = new Node(7);
69      root.right.right.left = new Node(9);
70      root.right.right.right = new Node(10);
71      System.out.println(inorderTraversal(root));
72  }
73  }
74

```

> Task :Iterative_Inorder_Traversa_of_Binary_Tree.main()
 [4, 2, 8, 5, 1, 6, 3, 9, 7, 10]

Codeium: Refactor | Explain

```
31 public class Iterative_Preorder_Traversal_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | ✕
32     public static List<Integer> preorderTraversal(Node root) {
33         List<Integer> res = new ArrayList<>();
34         if(root == null) return res;
35         Stack<Node> st = new Stack<>();
36         st.push(root);
37         while(!st.empty()){
38             Node node = st.pop();
39             res.add(node.data);
40             if(node.right != null) st.push(node.right);
41             if(node.left != null) st.push(node.left);
42         }
43         return res;
44     }
    Codeium: Refactor | Explain | Generate Javadoc | ✕
45     public static void main(String[] args) {
46         /*
47             Real World Representation the below tree into the code.
48             1
49             /\
50            2 3
51           /\ /\
52          4 5 6 7
53         /\ /\
54        8 9 10
55         */
56         Node root = new Node(1);
57         root.left = new Node(2);
58         root.right = new Node(3);
59         root.left.left = new Node(4);
60         root.left.right = new Node(5);
61         root.left.right.left = new Node(8);
62         root.right.left = new Node(6);
63         root.right.right = new Node(7);
64         root.right.right.left = new Node(9);
65         root.right.right.right = new Node(10);
66         System.out.println(preorderTraversal(root));
67     }
68 }
```

```
> Task :Iterative_Preorder_Traversal_of_Binary_Tree.main()
[1, 2, 4, 5, 8, 3, 6, 7, 9, 10]
```

Codeium: Refactor | Explain

```
38 public class Level_order_Traversal {
    Codeium: Refactor | Explain | Generate Javadoc | X
39     public static List<List<Integer>> levelOrder(Node root) {
40         Queue<Node> queue = new LinkedList<>();
41         List<List<Integer>> wrapList = new LinkedList<>();
42         if(root == null) return wrapList;
43         queue.add(root);
44         while(!queue.isEmpty()){
45             int levelNum = queue.size();
46             List<Integer> subList = new LinkedList<>();
47             for(int i = 0; i < levelNum; i++){
48                 if(queue.peek().left != null) queue.add(queue.peek().left);
49                 if(queue.peek().right != null) queue.add(queue.peek().right);
50                 subList.add(queue.remove().data);
51             }
52             wrapList.add(subList);
53         }
54         return wrapList;
55     }
    Codeium: Refactor | Explain | Generate Javadoc | X
56     public static void main(String[] args) {
57         /*
58          Real World Representation the below tree into the code.
59          |
60          |   |   |
61          |   |   |   3
62          |   |   |  /\
63          |   |   | 9  20
64          |   |   |  /\  \
65          |   |   | 15  7
66          |   |   |
67          |   |   |
68          |   |   |
69          |   |   |
70          |   |   |
71          |   |   |
72          |   |   |
73          |   |   |
          */
74         Node root = new Node(1);
75         root.left = new Node(9);
76         root.right = new Node(20);
77         root.right.left = new Node(15);
78         root.right.right = new Node(7);
79         System.out.println(levelOrder(root));
80     }
81 }
```

```
> Task :Level_order_Traversal.main()
[[1], [9, 20], [15, 7]]
```

```

75 // FOR SPIRAL ORDER
76 public static void reverse(List<Integer> subList){
77     int i = 0, j = subList.size() - 1;
78     while(i < j){
79         int temp = subList.get(i);
80         subList.set(i, subList.get(j));
81         subList.set(j, temp);
82         i++; j--;
83     }
84 }
85 ArrayList<Integer> findSpiral(Node root) {
86     ArrayList<Integer> res = new ArrayList<>();
87     Queue<Node> queue = new LinkedList<>();
88     List<List<Integer>> wrapList = new LinkedList<>();
89     if(root == null) return res;
90     int k = 0;
91     queue.add(root);
92     while(!queue.isEmpty()){
93         int levelNum = queue.size();
94         List<Integer> subList = new LinkedList<>();
95         for(int i = 0; i < levelNum; i++){
96             if(queue.peek().left != null) queue.add(queue.peek().left);
97             if(queue.peek().right != null) queue.add(queue.peek().right);
98             subList.add(queue.remove().data);
99         }
100         if(k % 2 == 0){
101             reverse(subList);
102         }
103         k++;
104         wrapList.add(subList);
105     }
106
107     for(List<Integer> subList : wrapList){
108         for(int num : subList){
109             res.add(num);
110         }
111     }
112     return res;
113 }
114

```

```

> Task :Level_Order_Traversal_Spiral.main()
[1, 9, 20, 7, 15]

```

```

31 Codeium: Refactor | Explain
public class Post_order_Traversal_of_Binary_Tree_using_1_stack {
32     Codeium: Refactor | Explain | Generate Javadoc | X
    public static List<Integer> postorderTraversal(Node root) {
33         List<Integer> res = new ArrayList<>();
34         if(root == null) return res;
35         Stack<Node> st = new Stack<>();
36         Node curr = root;
37         while(curr != null || !st.isEmpty()){
38             if(curr != null){
39                 st.push(curr);
40                 curr = curr.left;
41             }
42             else{
43                 Node temp = st.peek().right;
44                 if(temp == null){
45                     temp = st.pop();
46                     res.add(temp.data);
47                     while(!st.isEmpty() && temp == st.peek().right){
48                         temp = st.pop();
49                         res.add(temp.data);
50                     }
51                 }
52                 else{
53                     curr = temp;
54                 }
55             }
56         }
57         return res;
58     }
}

39 Codeium: Refactor | Explain | Generate Javadoc | X
60 public static void main(String[] args) {
61     /*
62     Real World Representation the below tree into the code
63     1
64     / \
65    2   3
66   / \ / \
67  4  5 6  7
68   / \ / \
69  8   9 10
70 */
71     Node root = new Node(1);
72     root.left = new Node(2);
73     root.right = new Node(3);
74     root.left.left = new Node(4);
75     root.left.right = new Node(5);
76     root.left.right.left = new Node(8);
77     root.right.left = new Node(6);
78     root.right.right = new Node(7);
79     root.right.right.left = new Node(9);
80     root.right.right.right = new Node(10);
81     System.out.println(postorderTraversal(root));
82 }
83

```

> Task :Post_order_Traversal_of_Binary_Tree_using_1_stack.main()
[4, 8, 5, 2, 6, 9, 10, 7, 3, 1]

```

31 public class PostOrder_Traversal_of_Binary_Tree_using_2_stack {
    Codeium: Refactor | Explain | Generate Javadoc | X
32 public static List<Integer> postorderTraversal(Node root) {
33     List<Integer> res = new ArrayList<>();
34     if(root == null) return res;
35     Stack<Node> st1 = new Stack<>();
36     Stack<Node> st2 = new Stack<>();
37     st1.push(root);
38     while(!st1.isEmpty()){
39         Node node = st1.peek();
40         st2.push(st1.pop());
41         if(node.left != null) st1.push(node.left);
42         if(node.right != null) st1.push(node.right);
43     }
44     while(!st2.isEmpty()){
45         res.add(st2.pop().data);
46     }
47     return res;
48 }
    Codeium: Refactor | Explain | Generate Javadoc | X
49 public static void main(String[] args) {
50     /*
51      Real World Representation the below tree into the code.
52      |
53      |   |   |
54      |   / \   |
55      | 2   3   |
56      / \ / \ / \
57      4 5 6 7
58      | / \ / \
59      8 9 10
60      */
61     Node root = new Node(1);
62     root.left = new Node(2);
63     root.right = new Node(3);
64     root.left.left = new Node(4);
65     root.left.right = new Node(5);
66     root.left.right.left = new Node(8);
67     root.right.left = new Node(6);
68     root.right.right = new Node(7);
69     root.right.right.left = new Node(9);
70     root.right.right.right = new Node(10);
71     System.out.println(postorderTraversal(root));
72 }
73

```

> Task :PostOrder_Traversal_of_Binary_Tree_using_2_stack.main()
[4, 8, 5, 2, 6, 9, 10, 7, 3, 1]


```

27 public class Postorder_Traversal_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | X
28     public static void postOrderTraversal(Node node){
29         if(node == null){
30             return;
31         }
32         postOrderTraversal(node.left);    // left
33         postOrderTraversal(node.right);   // right
34         System.out.print(node.data + " "); // root
35     }
    Codeium: Refactor | Explain | Generate Javadoc | X
36     public static void main(String[] args) {
37         /*
38          Real World Representation the below tree into the code.
39          1
40          /\
41         2 3
42        /\ /\
43       4 5 6 7
44        /\ /\
45       8 9 10
46         */
47         Node root = new Node(1);
48         root.left = new Node(2);
49         root.right = new Node(3);
50         root.left.left = new Node(4);
51         root.left.right = new Node(5);
52         root.left.right.left = new Node(8);
53         root.right.left = new Node(6);
54         root.right.right = new Node(7);
55         root.right.right.left = new Node(9);
56         root.right.right.right = new Node(10);
57         postOrderTraversal(root);
58     }
59 }
60

```

```

> Task :Postorder_Traversal_of_Binary_Tree.main()
4 8 5 2 6 9 10 7 3 1

```

```

28 Codeium: Refactor | Explain
public class Preorder_Traversal_of_Binary_Tree {
29     Codeium: Refactor | Explain | Generate Javadoc | X
    public static void preOrderTraversal(Node node){
30         if(node == null){
31             return;
32         }
33         System.out.print(node.data + " ");
34         preOrderTraversal(node.left);
35         preOrderTraversal(node.right);
36     }
37     Codeium: Refactor | Explain | Generate Javadoc | X
    public static void main(String[] args) {
38         /*
39         Real World Representation the below tree into the code.
40             1
41            /\
42           2 3
43          /\ /\
44         4 5 6 7
45          /\ /\
46         8 9 10
47         */
48         Node root = new Node(1);
49         root.left = new Node(2);
50         root.right = new Node(3);
51         root.left.left = new Node(4);
52         root.left.right = new Node(5);
53         root.left.right.left = new Node(8);
54         root.right.left = new Node(6);
55         root.right.right = new Node(7);
56         root.right.right.left = new Node(9);
57         root.right.right.right = new Node(10);
58         preOrderTraversal(root);
59     }
60 }
61

```

> Task :Preorder_Traversal_of_Binary_Tree.main()

1 2 4 5 8 3 6 7 9 10 |

```

6 Codeium: Refactor | Explain
class Pair{
7     Node node;
8     int num;
9     Pair(Node node, int num){
10         this.node = node;
11         this.num = num;
12     }
13 }
14 Codeium: Refactor | Explain
class Node{
15     int data;
16     Node left;
17     Node right;
18     Node(int key){
19         this.data = key;
20     }
21 }
22

```

```

23 Codeium: Refactor | Explain
public class Preorder_Inorder_and_Postorder_Traversal_in_one_Traversal {
24     Codeium: Refactor | Explain | Generate Javadoc | ×
25     public static void preInPostTraversal(Node root){
26         Stack<Pair> st = new Stack<>();
27         st.push(new Pair(root, 1));
28         ArrayList<Integer> pre = new ArrayList<>();
29         ArrayList<Integer> in = new ArrayList<>();
30         ArrayList<Integer> post = new ArrayList<>();
31         if(root == null) return;
32         while(!st.isEmpty()){
33             Pair it = st.pop();
34
35             // this is the part of pre
36             // Increment 1 to 2
37             // push the left side of the tree
38             if(it.num == 1){
39                 pre.add(it.node.data);
40                 it.num++;
41                 st.push(it);
42
43                 if(it.node.left != null){
44                     st.push(new Pair(it.node.left, 1));
45                 }
46             }
47
48             // this is the part of in
49             // Increment 2 to 3
50             // push the right side of the tree
51             else if(it.num == 2){
52                 in.add(it.node.data);
53                 it.num++;
54                 st.push(it);
55
56                 if(it.node.right != null){
57                     st.push(new Pair(it.node.right, 1));
58                 }
59             }
60
61             // this is the part of Post
62             // Don't push it back again
63             else{
64                 post.add(it.node.data);
65             }
66         }
67         System.out.println(pre);
68         System.out.println(in);
69         System.out.println(post);
70     }
71     Codeium: Refactor | Explain | Generate Javadoc | ×
72     public static void main(String[] args) {
73         /*
74          Real World Representation the below tree into the code.
75          1
76         / \
77        2   3
78       / \ / \
79      4  5 6  7
80     / \ / \
81     8  9 10
82
83     */
84     Node root = new Node(1);
85     root.left = new Node(2);
86     root.right = new Node(3);
87     root.left.left = new Node(4);
88     root.left.right = new Node(5);
89     root.left.right.left = new Node(8);
90     root.right.left = new Node(6);
91     root.right.right = new Node(7);
92     root.right.right.left = new Node(9);
93     root.right.right.right = new Node(10);
94     preInPostTraversal(root);
95 }

```

```

> Task :Preorder_Inorder_and_Postorder_Traversal_in_one_Traversal.main()
[1, 2, 4, 5, 8, 3, 6, 7, 9, 10]
[4, 2, 8, 5, 1, 6, 3, 9, 7, 10]
[4, 8, 5, 2, 6, 9, 10, 7, 3, 1]

```