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

```java
class Node{
    int data;
    Node left;
    Node right;
    Node(int key){
        this.data = key;
    }
}

Codeium: Refactor | Explain
public class Binary_Tree_Representation {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                   /\
                  2   3
                 / \  /
                4   5 6
                      \
                       7
        */
        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.right.left = new Node(6);
        root.right.left.right = new Node(7);
    }
}
```

```java
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.
                    1
                   /\
                  2   3
                 / \  / \
                4   5 6  7
                 /     / \
                8     9    10
        */
        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
```

```java
Codeium: Refactor | Explain
public class Iterative_Inorder_Traversa_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<Integer> inorderTraversal(Node root) {
        ArrayList<Integer> res = new ArrayList<>();
        Stack<Node> st = new Stack<>();
        Node node = root;
        while(true){
            if(node != null){
                st.push(node);
                node = node.left;
            }
            else{
                if(st.isEmpty()) break;
                node = st.pop();
                res.add(node.data);
                node = node.right;
            }
        }
        return res;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                   /\
                  2   3
                 / \  / \
                4   5 6  7
                 /     / \
                8     9    10
        */
        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);
        System.out.println(inorderTraversal(root));
    }
}

```

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

```java
Codeium: Refactor | Explain
public class Iterative_Preorder_Traversal_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<Integer> preorderTraversal(Node root) {
        List<Integer> res = new ArrayList<>();
        if(root == null) return res;
        Stack<Node> st = new Stack<>();
        st.push(root);
        while(!st.empty()){
            Node node = st.pop();
            res.add(node.data);
            if(node.right != null) st.push(node.right);
            if(node.left != null) st.push(node.left);
        }
        return res;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                   /\
                  2   3
                 / \  / \
                4   5 6  7
                 /     / \
                8     9    10
        */
        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);
        System.out.println(preorderTraversal(root));
    }
}
```

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

```java
Codeium: Refactor | Explain
public class Level_order_Traversal {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<List<Integer>> levelOrder(Node root) {
        Queue<Node> queue = new LinkedList<>();
        List<List<Integer>> wrapList = new LinkedList<>();
        if(root == null) return wrapList;
        queue.add(root);
        while(!queue.isEmpty()){
            int levelNum = queue.size();
            List<Integer> subList = new LinkedList<>();
            for(int i = 0; i < levelNum; i++){
                if(queue.peek().left != null) queue.add(queue.peek().left);
                if(queue.peek().right != null) queue.add(queue.peek().right);
                subList.add(queue.remove().data);
            }
            wrapList.add(subList);
        }
        return wrapList;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                  3
                 /\
                9   20
                   /  \
                  15   7
        */
        Node root = new Node(1);
        root.left = new Node(9);
        root.right = new Node(20);
        root.right.left = new Node(15);
        root.right.right = new Node(7);
        System.out.println(levelOrder(root));
    }
}

```

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

```java
// FOR SPIRAL ORDER
public static void reverse(List<Integer> subList){
        int i = 0, j = subList.size() - 1;
        while(i < j){
            int temp = subList.get(i);
            subList.set(i, subList.get(j));
            subList.set(j, temp);
            i++; j--;
        }
    }
    ArrayList<Integer> findSpiral(Node root) {
        ArrayList<Integer> res = new ArrayList<>();
        Queue<Node> queue = new LinkedList<>();
        List<List<Integer>> wrapList = new LinkedList<>();
        if(root == null) return res;
        int k = 0;
        queue.add(root);
        while(!queue.isEmpty()){
            int levelNum = queue.size();
            List<Integer> subList = new LinkedList<>();
            for(int i = 0; i < levelNum; i++){
                if(queue.peek().left != null) queue.add(queue.peek().left);
                if(queue.peek().right != null) queue.add(queue.peek().right);
                subList.add(queue.remove().data);
            }
            if(k % 2 == 0){
                reverse(subList);
            }
            k++;
            wrapList.add(subList);
        }

        for(List<Integer> subList : wrapList){
            for(int num : subList){
                res.add(num);
            }
        }
        return res;
    }
```

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

```java
Codeium: Refactor | Explain
public class Post_order_Traversal_of_Binary_Tree_using_1_stack {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<Integer> postorderTraversal(Node root) {
        List<Integer> res = new ArrayList<>();
        if(root == null) return res;
        Stack<Node> st = new Stack<>();
        Node curr = root;
        while(curr != null || !st.isEmpty()){
            if(curr != null){
                st.push(curr);
                curr = curr.left;
            }
            else{
                Node temp = st.peek().right;
                if(temp == null){
                    temp = st.pop();
                    res.add(temp.data);
                    while(!st.isEmpty() && temp == st.peek().right){
                        temp = st.pop();
                        res.add(temp.data);
                    }
                }
                else{
                    curr = temp;
                }
            }
        }
        return res;
    }
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code
                    1
                  /  \
                 2     3
                / \   / \
               4   5 6   7
                  /     / \
                 8     9    10
        */
        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);
        System.out.println(postorderTraversal(root));
    }
}

```

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

```java
public class PostOrder_Traversal_of_Binary_Tree_using_2_stack {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static List<Integer> postorderTraversal(Node root) {
        List<Integer> res = new ArrayList<>();
        if(root == null) return res;
        Stack<Node> st1 = new Stack<>();
        Stack<Node> st2 = new Stack<>();
        st1.push(root);
        while(!st1.isEmpty()){
            Node node = st1.peek();
            st2.push(st1.pop());
            if(node.left != null) st1.push(node.left);
            if(node.right != null) st1.push(node.right);
        }
        while(!st2.isEmpty()){
            res.add(st2.pop().data);
        }
        return res;
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                  /  \
                2     3
               / \   / \
              4   5 6   7
                /     / \
               8     9    10
        */
        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);
        System.out.println(postorderTraversal(root));
    }
}

```

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

```java
public class Postorder_Traversal_of_Binary_Tree {
    public static void postOrderTraversal(Node node){
        if(node == null){
            return;
        }
        postOrderTraversal(node.left);      // left
        postOrderTraversal(node.right);     // right
        System.out.print(node.data + " ");  // root
    }
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                   /\
                  2   3
                 / \  / \
                4   5 6  7
                  /     / \
                 8     9    10
        */
        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);
        postOrderTraversal(root);
    }
}

```

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

```java
Codeium: Refactor | Explain
public class Preorder_Traversal_of_Binary_Tree {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void preOrderTraversal(Node node){
        if(node == null){
            return;
        }
        System.out.print(node.data + " ");
        preOrderTraversal(node.left);
        preOrderTraversal(node.right);
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                   /\
                  2   3
                 / \  / \
                4   5 6  7
                 /     / \
                8     9    10
        */
        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);
        preOrderTraversal(root);
    }
}

```

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

```java
Codeium: Refactor | Explain
class Pair{
    Node node;
    int num;
    Pair(Node node, int num){
        this.node = node;
        this.num = num;
    }
}
Codeium: Refactor | Explain
class Node{
    int data;
    Node left;
    Node right;
    Node(int key){
        this.data = key;
    }
}

```

```java
Codeium: Refactor | Explain
public class Preorder_Inorder_and_Postorder_Traversal_in_one_Traversal {
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void preInPostTraversal(Node root){
        Stack<Pair> st = new Stack<>();
        st.push(new Pair(root, 1));
        ArrayList<Integer> pre = new ArrayList<>();
        ArrayList<Integer> in = new ArrayList<>();
        ArrayList<Integer> post = new ArrayList<>();
        if(root == null) return;
        while(!st.isEmpty()){
            Pair it = st.pop();

            // this is the part of pre
            // Increment 1 to 2
            // push the left side of the tree
            if(it.num == 1){
                pre.add(it.node.data);
                it.num++;
                st.push(it);

                if(it.node.left != null){
                    st.push(new Pair(it.node.left, 1));
                }
            }

            // this is the part of in
            // Increment 2 to 3
            // push the right side of the tree
            else if(it.num == 2){
                in.add(it.node.data);
                it.num++;
                st.push(it);

                if(it.node.right != null){
                    st.push(new Pair(it.node.right, 1));
                }
            }

            // this is the part of Post
            // Don't push it back again
            else{
                post.add(it.node.data);
            }
        }
        System.out.println(pre);
        System.out.println(in);
        System.out.println(post);
    }
    Codeium: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        /*
            Real World Representation the below tree into the code.
                    1
                  /   \
                2       3
               / \     / \
              4   5 6   7
                /       / \
               8       9   10
        */
        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);
        preInPostTraversal(root);
    }
}

```

```
> 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]
```