/*

Given a Binary Tree, find its Boundary Traversal. The traversal should be in the following order: defined as the path from the root to the left-most node ie- the leaf node you could reach when you always travel preferring the left subtree over the right subtree. All the leaf nodes except for the ones that are part of left or right boundary. defined as the path from the right-most node to the root. The right-most node is the leaf node you could reach when you always travel preferring the right subtree over the left subtree. Exclude the root from this as it was already included in the traversal of left boundary nodes.

If the root doesn't have a left subtree or right subtree, then the root itself is the left or right boundary.

```
     1
    / \
   2    3
  / \  / \
 4   5 6  7
    / \
   8   9
```

Output:

```
  1 2 4 8 9 6 7 3
```

*/

// Solution

```
import java.util.*;
// Creating Node structure
class Node{
   Node left;
   int data;
   Node right;
   Node(int data){
      this.data = data;
      left = null;
```

```
        right = null;
    }
}
public class BoundaryTraversal {
    public static void addLeftSideNodes(Node root,ArrayList<Integer> res){
        Node temp = root;
        while(temp != null){
            if(temp.left != null && temp.right != null)
                res.add(temp.data);
            if(temp.left != null)
                temp = temp.left;
            else if(temp.right != null)
                temp = temp.right;
            else
                break;
        }
    }

    public static void addRightSideNodes(Node root,ArrayList<Integer> res){
        Node temp = root;
        Stack<Integer> stack = new Stack<Integer>();
        while(temp != null){
            if(temp.left != null && temp.right != null)
                stack.push(temp.data);
            if(temp.right != null)
                temp = temp.right;
            else if(temp.left != null)
                temp = temp.left;
```

```
        else
            break;
    }
    while(stack.size() > 0){
        res.add(stack.peek());
        stack.pop();
    }
}

public static void addLeadNodes(Node root,ArrayList<Integer> res){
    if(root.left == null && root.right == null)
        res.add(root.data);
    if(root.left != null) addLeadNodes(root.left, res);
    if(root.right != null) addLeadNodes(root.right, res);
}

public static void printBoundaryNodes(Node root){
    ArrayList<Integer> answer = new ArrayList<Integer>();
    addLeftSideNodes(root, answer);
    addLeadNodes(root, answer);
    if(root.right != null)
        addRightSideNodes(root.right, answer);

    for (int i = 0; i < answer.size(); i++) {
        System.out.print(answer.get(i) + " ");
    }
}
public static void main(String[] args) {
```

```
        Node tree = new Node(1); // root node is created with value 1

        // Inserting the left side nodes
        tree.left = new Node(2);
        tree.left.left = new Node(4);
        tree.left.right = new Node(5);
        tree.left.right.left = new Node(8);
        tree.left.right.right = new Node(9);

        // Inserting the right side nodes
        tree.right = new Node(3);
        tree.right.left = new Node(6);
        tree.right.right = new Node(7);

        printBoundaryNodes(tree);
    }
}
```

// Output: