```java
// Functions for printing Boundary of a binary tree anticlockwise from root.
// Boundary => root node - leftmost sided nodes - leave nodes - rightmost sided node
public ArrayList<Integer> printBoundary(Node root) {
    // store root node, leftmost sided nodes, leave nodes and rightmost sided node
    // to the boundary respectively, Since rightmost sided node extracted from
    // root node, add them to the boundary in reverse order(anticlockwise)
    ArrayList<Integer> boundary = new ArrayList<>();
    // 1. add root node
    boundary.add(root.data);
    // 2. Extract leftmost sided nodes
    Node curr = root.left;
    while (curr != null) {
        if (curr.left == null && curr.right == null) {
            break;
        }
        boundary.add(curr.data);
        Node temp = curr.left;
        if (temp == null) {
            // left node is empty, try right node
            curr = curr.right;
        } else {
            curr = temp;
        }
    }
    // 3. Extract all the leave nodes
    dfs(root, boundary);
    // 4. Extract rightmost sided nodes
    ArrayList<Integer> rightMostNodes = new ArrayList<>();
    curr = root.right;
```

```
    ArrayList<Integer> rightMostNodes = new ArrayList<>();
    curr = root.right;
    while (curr != null) {
        if (curr.left == null && curr.right == null) {
            break;
        }
        rightMostNodes.add(curr.data);
        Node temp = curr.right;
        if (temp == null) {
            // right node is empty, try left node
            curr = curr.left;
        } else {
            curr = temp;
        }
    }
    // 5. add rightmost nodes to the boundary in reverse order
    for (int i = rightMostNodes.size() - 1; i >= 0; i--) {
        boundary.add(rightMostNodes.get(i));
    }
    return boundary;
}
// DFS for get all the leave nodes from left to right
void dfs(Node node, ArrayList<Integer> boundary) {
    if (node == null) {
        return;
    }
    // leave node
    if (node.left == null && node.right == null) {
        boundary.add(node.data);
        return;
    }
    // go leftmost subtree first, then rightmost
    dfs(node.left, boundary);
    dfs(node.right, boundary);
}
```