```java
package binarytree;

// Java program to print left view of binary tree

/* Class containing left and right child of current
node and key value*/


import java.util.LinkedList;
import java.util.Queue;

/* Class to print the left view */
class BinaryTree55 {
    Node root;
    static int max_level = 0;

    // recursive function to print left view
    void leftViewUtil(Node node, int level)
    {
        // Base Case
        if (node == null)
            return;

        // If this is the first node of its level
        if (max_level < level) {
            System.out.print(" " + node.data);
            max_level = level;
        }

        // Recur for left and right subtrees
        leftViewUtil(node.left, level + 1);
        leftViewUtil(node.right, level + 1);
    }

    // A wrapper over leftViewUtil()
    void leftView()
    {
        leftViewUtil(root, 1);
    }

    /* testing for example nodes */
    public static void main(String args[])
    {
        /* creating a binary tree and entering the nodes */
        BinaryTree55 tree = new BinaryTree55();
        tree.root = new Node(12);
        tree.root.left = new Node(10);
        tree.root.right = new Node(30);
        tree.root.right.left = new Node(25);
        tree.root.right.right = new Node(40);

        tree.leftView();
    }
}


  class PrintRightView {
    // Binary tree node
    private static class Node {
        int data;
        Node left, right;

        public Node(int data)
        {
            this.data = data;
            this.left = null;
```

```
            this.right = null;
        }
    }

    // function to print right view of binary tree
    private static void printRightView(Node root)
    {
        if (root == null)
            return;

        Queue<Node> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            // number of nodes at current level
            int n = queue.size();

            // Traverse all nodes of current level
            for (int i = 1; i <= n; i++) {
                Node temp = queue.poll();

                // Print the left most element at
                // the level
                if (i == 1)
                    System.out.print(temp.data + " ");

                // Add left node to queue
                if (temp.left != null)
                    queue.add(temp.left);

                // Add right node to queue
                if (temp.right != null)
                    queue.add(temp.right);
            }
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        // construct binary tree as shown in
        // above diagram
        Node root = new Node(10);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(7);
        root.left.right = new Node(8);
        root.right.right = new Node(15);
        root.right.left = new Node(12);
        root.right.right.left = new Node(14);

        printRightView(root);
    }
}

// This code is contributed by
// Manne SreeCharan
```