Code:-

```
package Tree;
// AVL tree implementation in Java

// Create node
class node {
    int item, height;
    node left, right;

    node(int d) {
        item = d;
        height = 1;
    }
}

// Tree class
class AVLTree {
    node root;

    int height(node N) {
        if (N == null)
            return 0;
        return N.height;
    }

    int max(int a, int b) {
        return (a > b) ? a : b;
    }

    node rightRotate(node y) {
        node x = y.left;
        node T2 = x.right;
        x.right = y;
        y.left = T2;
        y.height = max(height(y.left), height(y.right)) + 1;
        x.height = max(height(x.left), height(x.right)) + 1;
        return x;
    }

    node leftRotate(node x) {
        node y = x.right;
        node T2 = y.left;
        y.left = x;
        x.right = T2;
        x.height = max(height(x.left), height(x.right)) + 1;
        y.height = max(height(y.left), height(y.right)) + 1;
        return y;
    }

    // Get balance factor of a node
    int getBalanceFactor(node N) {
        if (N == null)
            return 0;
        return height(N.left) - height(N.right);
    }

    // Insert a node
    node insertNode(node node, int item) {

        // Find the position and insert the node
        if (node == null)
            return (new node(item));
```

```
        if (item < node.item)
            node.left = insertNode(node.left, item);
        else if (item > node.item)
            node.right = insertNode(node.right, item);
        else
            return node;

        // Update the balance factor of each node
        // And, balance the tree
        node.height = 1 + max(height(node.left), height(node.right));
        int balanceFactor = getBalanceFactor(node);
        if (balanceFactor > 1) {
            if (item < node.left.item) {
                return rightRotate(node);
            } else if (item > node.left.item) {
                node.left = leftRotate(node.left);
                return rightRotate(node);
            }
        }
        if (balanceFactor < -1) {
            if (item > node.right.item) {
                return leftRotate(node);
            } else if (item < node.right.item) {
                node.right = rightRotate(node.right);
                return leftRotate(node);
            }
        }
        return node;
    }

    node nodeWithMimumValue(node node) {
        node current = node;
        while (current.left != null)
            current = current.left;
        return current;
    }

    // Delete a node
    node deleteNode(node root, int item) {

        // Find the node to be deleted and remove it
        if  (root == null)
            return root;
        if (item < root.item)
            root.left = deleteNode(root.left, item);
        else if (item > root.item)
            root.right = deleteNode(root.right, item);
        else {
            if ((root.left == null) || (root.right == null)) {
                node temp = null;
                if (temp == root.left)
                    temp = root.right;
                else
                    temp = root.left;
                if (temp == null) {
                    temp = root;
                    root = null;
                } else
                    root = temp;
            } else {
                node temp = nodeWithMimumValue(root.right);
```

```
                root.item = temp.item;
                root.right = deleteNode(root.right, temp.item);
            }
        }
        if (root == null)
            return root;

        // Update the balance factor of each node and balance the tree
        root.height = max(height(root.left), height(root.right)) + 1;
        int balanceFactor = getBalanceFactor(root);
        if (balanceFactor > 1) {
            if (getBalanceFactor(root.left) >= 0) {
                return rightRotate(root);
            } else {
                root.left = leftRotate(root.left);
                return rightRotate(root);
            }
        }
        if (balanceFactor < -1) {
            if (getBalanceFactor(root.right) <= 0) {
                return leftRotate(root);
            } else {
                root.right = rightRotate(root.right);
                return leftRotate(root);
            }
        }
        return root;
    }

    void preOrder(node node) {
        if (node != null) {
            System.out.print(node.item + " ");
            preOrder(node.left);
            preOrder(node.right);
        }
    }

    // Print the tree
    private void printTree(node currPtr,  String indent, boolean last) {
        if (currPtr != null) {
            System.out.print(indent);
            if (last) {
                System.out.print("R----");
                indent += "   ";
            } else {
                System.out.print("L----");
                indent += "|  ";
            }
            System.out.println(currPtr.item);
            printTree(currPtr.left, indent, false);
            printTree(currPtr.right, indent, true);
        }
    }

    // Driver code
    public static void main(String[] args) {
        AVLTree tree = new AVLTree();
        tree.root = tree.insertNode(tree.root, 33);
        tree.root = tree.insertNode(tree.root, 13);
        tree.root = tree.insertNode(tree.root, 53);
        tree.root = tree.insertNode(tree.root, 9);
```

```
        tree.root = tree.insertNode(tree.root, 21);
        tree.root = tree.insertNode(tree.root, 61);
        tree.root = tree.insertNode(tree.root, 8);
        tree.root = tree.insertNode(tree.root, 11);
        tree.printTree(tree.root, "", true);
        tree.root = tree.deleteNode(tree.root, 13);
        System.out.println("After Deletion: ");
        tree.printTree(tree.root, "", true);
    }
}
```

OUTPUT:-