****

University of Central Punjab

**Data Structure and Algorithms - Lab**

**AVL TREE**

**Name:** Abdullah Maqbool

**Reg no:** L1F22BSSE0391

**Section:** P4

**Submitted to:** Ma`am Javaria Tanveer

**All Tasks (1 to 6)**

**Note: (I did all the tasks in a single Class)**

**Java Code:**

public class All\_Tasks {  
 class Node {  
 int key, height;  
 Node left, right;  
  
 Node(int d) {  
 key = d;  
 height = 1;  
 }  
 }  
  
 Node root;  
  
 // (task 1)  
 Node insert(Node node, int key) {  
 if (node == null)  
 return new Node(key);  
 if (key < node.key)  
 node.left = insert(node.left, key);  
 else if (key > node.key)  
 node.right = insert(node.right, key);  
 else  
 return node;  
 node.height = 1 + Math.*max*(height(node.left), height(node.right));  
 int balance = getBalance(node);  
 if (balance > 1 && key < node.left.key)  
 return rightRotate(node);  
 if (balance < -1 && key > node.right.key)  
 return leftRotate(node);  
 if (balance > 1 && key > node.left.key) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
 if (balance < -1 && key < node.right.key) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
 return node;  
 }  
  
 // (task 1)  
 Node deleteNode(Node root, int key) {  
 if (root == null)  
 return root;  
 if (key < root.key)  
 root.left = deleteNode(root.left, key);  
 else if (key > root.key)  
 root.right = deleteNode(root.right, key);  
 else {  
 if ((root.left == null) || (root.right == null)) {  
 Node temp = (root.left != null) ? root.left : root.right;  
 if (temp == null) {  
 temp = root;  
 root = null;  
 } else  
 root = temp;  
 } else {  
 Node temp = minValueNode(root.right);  
 root.key = temp.key;  
 root.right = deleteNode(root.right, temp.key);  
 }  
 }  
 if (root == null)  
 return root;  
 root.height = Math.*max*(height(root.left), height(root.right)) + 1;  
 int balance = getBalance(root);  
 if (balance > 1 && getBalance(root.left) >= 0)  
 return rightRotate(root);  
 if (balance > 1 && getBalance(root.left) < 0) {  
 root.left = leftRotate(root.left);  
 return rightRotate(root);  
 }  
 if (balance < -1 && getBalance(root.right) <= 0)  
 return leftRotate(root);  
 if (balance < -1 && getBalance(root.right) > 0) {  
 root.right = rightRotate(root.right);  
 return leftRotate(root);  
 }  
 return root;  
 }  
  
 Node minValueNode(Node node) {  
 Node current = node;  
 while (current.left != null)  
 current = current.left;  
 return current;  
 }  
  
 int height(Node N) {  
 if (N == null)  
 return 0;  
 return N.height;  
 }  
  
 int getBalance(Node N) {  
 if (N == null)  
 return 0;  
 return height(N.left) - height(N.right);  
 }  
  
 Node rightRotate(Node y) {  
 Node x = y.left;  
 Node T2 = x.right;  
 x.right = y;  
 y.left = T2;  
 y.height = Math.*max*(height(y.left), height(y.right)) + 1;  
 x.height = Math.*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 = Math.*max*(height(x.left), height(x.right)) + 1;  
 y.height = Math.*max*(height(y.left), height(y.right)) + 1;  
 return y;  
 }  
  
 void insertNode(int key) {  
 root = insert(root, key);  
 }  
  
 // (task 2)  
 void createSampleTree() {  
 insertNode(30);  
 insertNode(20);  
 insertNode(40);  
 insertNode(10);  
 insertNode(25);  
 }  
  
 // (task 3)  
 void inOrder(Node node) {  
 if (node != null) {  
 inOrder(node.left);  
 System.*out*.print(node.key + " ");  
 inOrder(node.right);  
 }  
 }  
  
 // (task 3)  
 void preOrder(Node node) {  
 if (node != null) {  
 System.*out*.print(node.key + " ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
  
 // (task 3)  
 void postOrder(Node node) {  
 if (node != null) {  
 postOrder(node.left);  
 postOrder(node.right);  
 System.*out*.print(node.key + " ");  
 }  
 }  
  
 // (task 4)  
 int findMin() {  
 if (root == null) return -1;  
 Node curr = root;  
 while (curr.left != null)  
 curr = curr.left;  
 return curr.key;  
 }  
  
 // (task 4)  
 int findMax() {  
 if (root == null) return -1;  
 Node curr = root;  
 while (curr.right != null)  
 curr = curr.right;  
 return curr.key;  
 }  
  
 // (task 5)  
 int treeHeight() {  
 return height(root);  
 }  
  
 // (task 6)  
 int countNodes(Node node) {  
 if (node == null)  
 return 0;  
 return 1 + countNodes(node.left) + countNodes(node.right);  
 }  
  
 // (task 6)  
 int countLeafNodes(Node node) {  
 if (node == null)  
 return 0;  
 if (node.left == null && node.right == null)  
 return 1;  
 return countLeafNodes(node.left) + countLeafNodes(node.right);  
 }  
  
 // (task 6)  
 int countInternalNodes(Node node) {  
 if (node == null || (node.left == null && node.right == null))  
 return 0;  
 return 1 + countInternalNodes(node.left) + countInternalNodes(node.right);  
 }  
  
 public static void main(String[] args) {  
 All\_Tasks tree = new All\_Tasks();  
 tree.createSampleTree();  
 System.*out*.print("InOrder: ");  
 tree.inOrder(tree.root);  
 System.*out*.println();  
 System.*out*.print("PreOrder: ");  
 tree.preOrder(tree.root);  
 System.*out*.println();  
 System.*out*.print("PostOrder: ");  
 tree.postOrder(tree.root);  
 System.*out*.println();  
 System.*out*.println("Min: " + tree.findMin());  
 System.*out*.println("Max: " + tree.findMax());  
 System.*out*.println("Height: " + tree.treeHeight());  
 System.*out*.println("Total Nodes: " + tree.countNodes(tree.root));  
 System.*out*.println("Leaf Nodes: " + tree.countLeafNodes(tree.root));  
 System.*out*.println("Internal Nodes: " + tree.countInternalNodes(tree.root));  
 }  
}

**Output Screenshots:**

