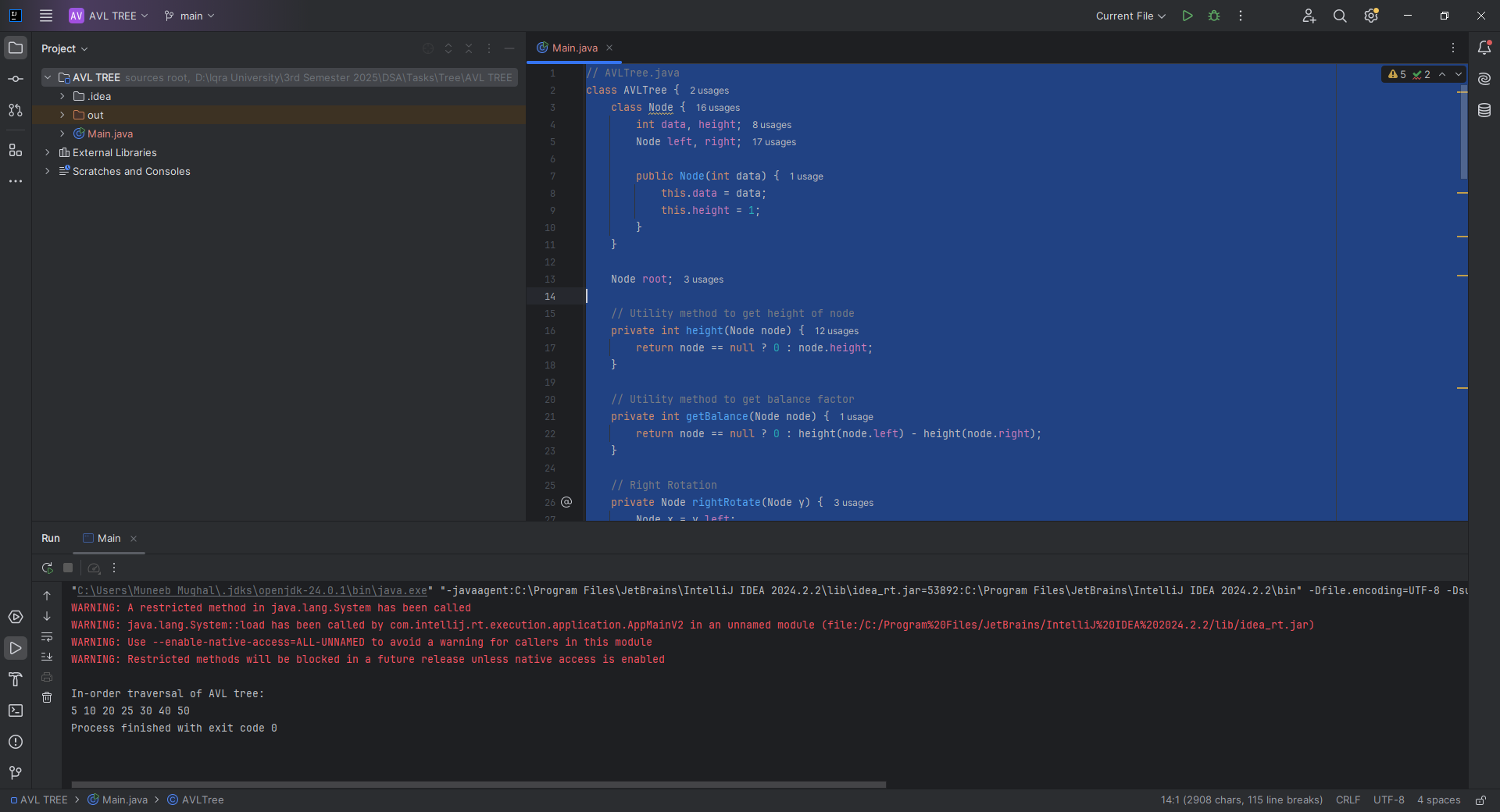
**CLASS TASK All:**

**CODE:**

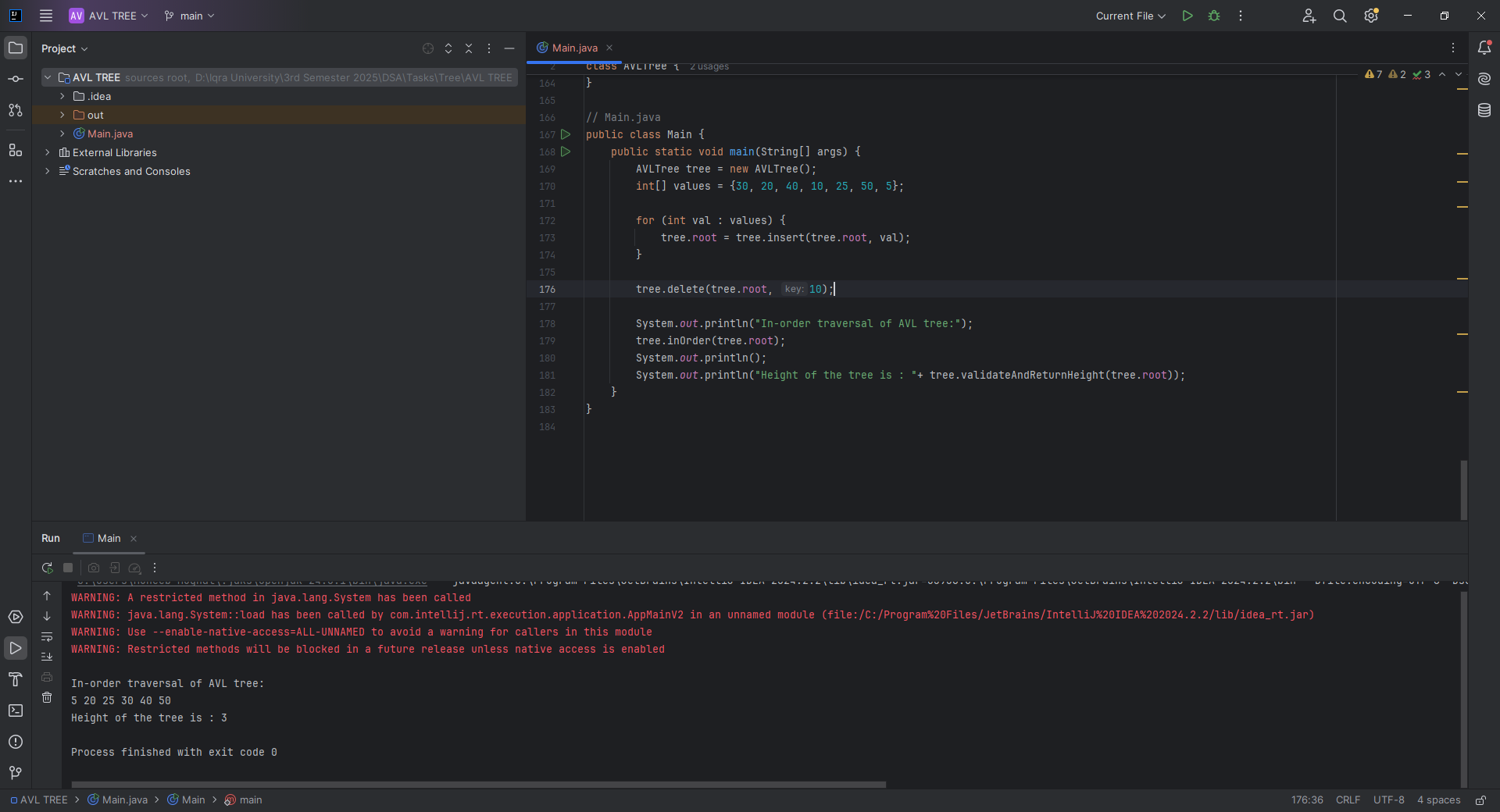
// AVLTree.java  
class AVLTree {  
 class Node {  
 int data, height;  
 Node left, right;  
  
 public Node(int data) {  
 this.data = data;  
 this.height = 1;  
 }  
 }  
  
 Node root;  
  
 // Utility method to get height of node  
 private int height(Node node) {  
 return node == null ? 0 : node.height;  
 }  
  
 // Utility method to get balance factor  
 private int getBalance(Node node) {  
 return node == null ? 0 : height(node.left) - height(node.right);  
 }  
  
 // Right Rotation  
 private Node rightRotate(Node y) {  
 Node x = y.left;  
 Node T2 = x.right;  
  
 // Perform rotation  
 x.right = y;  
 y.left = T2;  
  
 // Update heights  
 y.height = Math.*max*(height(y.left), height(y.right)) + 1;  
 x.height = Math.*max*(height(x.left), height(x.right)) + 1;  
  
 return x;  
 }  
  
 // Left Rotation  
 private Node leftRotate(Node x) {  
 Node y = x.right;  
 Node T2 = y.left;  
  
 // Perform rotation  
 y.left = x;  
 x.right = T2;  
  
 // Update heights  
 x.height = Math.*max*(height(x.left), height(x.right)) + 1;  
 y.height = Math.*max*(height(y.left), height(y.right)) + 1;  
  
 return y;  
 }  
  
 // Insert node into AVL tree  
 public Node insert(Node node, int data) {  
 if (node == null) return new Node(data);  
  
 if (data < node.data) node.left = insert(node.left, data);  
 else if (data > node.data) node.right = insert(node.right, data);  
 else return node; // Duplicates not allowed  
  
 node.height = 1 + Math.*max*(height(node.left), height(node.right));  
  
 int balance = getBalance(node);  
  
 // Left Left Case  
 if (balance > 1 && data < node.left.data)  
 return rightRotate(node);  
  
 // Right Right Case  
 if (balance < -1 && data > node.right.data)  
 return leftRotate(node);  
  
 // Left Right Case  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
  
 // Right Left Case  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
  
 // In-order traversal  
 public void inOrder(Node node) {  
 if (node != null) {  
 inOrder(node.left);  
 System.*out*.print(node.data + " ");  
 inOrder(node.right);  
 }  
 }  
}  
  
// Main.java  
public class Main {  
 public static void main(String[] args) {  
 AVLTree tree = new AVLTree();  
 int[] values = {30, 20, 40, 10, 25, 50, 5};  
  
 for (int val : values) {  
 tree.root = tree.insert(tree.root, val);  
 }  
  
 System.*out*.println("In-order traversal of AVL tree:");  
 tree.inOrder(tree.root);  
 }  
}



**Home Task 1:**

**CODE:**

public Node delete(Node root, int key) {  
 if (root == null) return root;  
  
 if (key < root.data) root.left = delete(root.left, key);  
 else if (key > root.data) root.right = delete(root.right, key);  
 else {  
 if ((root.left == null) || (root.right == null)) {  
 Node temp = (root.left != null) ? root.left : root.right;  
 if (temp == null) {  
 root = null;  
 } else {  
 root = temp;  
 }  
 } else {  
 Node temp = minValueNode(root.right);  
 root.data = temp.data;  
 root.right = delete(root.right, temp.data);  
 }  
 }  
  
 if (root == null) return root;  
  
 root.height = Math.*max*(height(root.left), height(root.right)) + 1;  
 int balance = getBalance(root);  
  
 // Rebalancing  
 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;  
}  
  
private Node minValueNode(Node node) {  
 Node current = node;  
 while (current.left != null) current = current.left;  
 return current;  
}



**Home Task 2:** Finding the Height of the AVL Tree

**CODE:**

public int validateAndReturnHeight(Node node) {  
 if (node == null) return 0;  
  
 int left = validateAndReturnHeight(node.left);  
 int right = validateAndReturnHeight(node.right);  
 int expectedHeight = 1 + Math.*max*(left, right);  
  
 if (node.height != expectedHeight) {  
 System.*out*.println("Height mismatch at node " + node.data +  
 ": Expected " + expectedHeight + ", Found " + node.height);  
 }  
  
 return expectedHeight;  
}

