~\OneDrive - VietNam National University - HCM INTERNATIONAL UNIVERSITY\Desktop\DSA\DSA LAB NEW\Lab 7 Hash Tables\ITITSB22029_DoMinhDuy_Lab7\1\Tree.java

```java
// ------------------------------------------------------------
// Representing arithmetic expressions by binary tree
// CS 501
// Zdravko Markov
// ------------------------------------------------------------
class Tree {
   public static int countElements(Node t) {
      if (t == null)
         return 0;
      return 1 + countElements(t.leftChild) + countElements(t.rightChild); // Count current node and recurse
   }

   public static int computeHeight(Node t) {
      if (t == null)
         return -1;
      return 1 + Math.max(computeHeight(t.leftChild), computeHeight(t.rightChild)); // Recur for left and right
   }

   public static int countLeaves(Node t) {
      if (t == null)
         return 0;
      if (t.leftChild == null && t.rightChild == null)
         return 1;
      return countLeaves(t.leftChild) + countLeaves(t.rightChild);
   }

   public static boolean isFullyBalanced(Node t) {
      return checkBalance(t) != -1;
   }

   private static int checkBalance(Node t) {
      if (t == null)
         return 0;

      int leftHeight = checkBalance(t.leftChild);
      int rightHeight = checkBalance(t.rightChild);

      if (leftHeight == -1 || rightHeight == -1)
         return -1;
      if (Math.abs(leftHeight - rightHeight) > 1)
         return -1;

      return 1 + Math.max(leftHeight, rightHeight);
   }

   public static boolean isIdentical(Node t1, Node t2) {
      if (t1 == null && t2 == null)
         return true;
      if (t1 == null || t2 == null)
```

```java
         return false;
      if (t1.value != t2.value || t1.operation != t2.operation)
         return false;

      // Recursively check left and right subtrees
      return isIdentical(t1.leftChild, t2.leftChild) && isIdentical(t1.rightChild, t2.rightChild);
   }

   // ------------------------------------------------------------
   public static void main(String[] args) {
      // Define Tree 1
      Node a = node(2);
      Node b = node(3);
      Node c = node('+', a, b);
      Node d = node(5);
      Node e = node(1);
      Node f = node('-', d, e);
      Node g = node('*', c, f);
      Node h = node(8);
      Node i = node('/', g, h);

      // Define Tree 2 (identical to Tree 1)
      Node a2 = node(2);
      Node b2 = node(3);
      Node c2 = node('+', a2, b2);
      Node d2 = node(5);
      Node e2 = node(1);
      Node f2 = node('-', d2, e2);
      Node g2 = node('*', c2, f2);
      Node h2 = node(8);
      Node i2 = node('/', g2, h2);

      System.out.println("Tree:");
      showTree(0, i);
      System.out.print("Prefix: ");
      prefix(i);
      System.out.print("\nPostfix: ");
      postfix(i);
      System.out.print("\nInfix: ");
      infix(i);
      System.out.println("\nValue: " + eval(i));

      System.out.println("-------------------");

      // Compute the height of the tree
      System.out.println("Height of the tree: " + computeHeight(i));

      // Count elements in the tree
      System.out.println("Number of elements in the tree: " + countElements(i));

      // Count leaves in the tree
      System.out.println("Number of leaves in the tree: " + countLeaves(i));

      // Check if the tree is fully balanced
      System.out.println("Is the tree fully balanced? " + isFullyBalanced(i));

      // Check if the two trees are identical
      System.out.println("Are the trees identical? " + isIdentical(i, i2));
   }

   // -------------------------------------------------------------
   public static Node node(char op, Node l, Node r) {
      Node a = new Node();
      a.operation = op;
      a.leftChild = l;
      a.rightChild = r;
      return a;
   }

   public static Node node(int val) {
      Node a = new Node();
      a.value = val;
      return a;
   }

   public static void prefix(Node t) {
      if (t == null)
         return;
      if (t.leftChild == null && t.rightChild == null)
         System.out.print(t.value + " ");
      else {
         System.out.print(t.operation + " ");
         prefix(t.leftChild);
         prefix(t.rightChild);
      }
   }

   public static void postfix(Node t) {
      if (t == null)
         return;
      if (t.leftChild == null && t.rightChild == null)
         System.out.print(t.value + " ");
      else {
         postfix(t.leftChild);
         postfix(t.rightChild);
         System.out.print(t.operation + " ");
      }
   }

   public static void infix(Node t) {
      if (t == null)
         return;
      if (t.leftChild == null && t.rightChild == null)
         System.out.print(t.value);
      else {
         System.out.print("(");
         infix(t.leftChild);
         System.out.print(t.operation);
         infix(t.rightChild);
         System.out.print(")");
      }
   }

   public static double eval(Node t) {
      if (t == null)
         return 0;
      if (t.leftChild == null && t.rightChild == null)
         return t.value;

      switch (t.operation) {
         case '+':
            return eval(t.leftChild) + eval(t.rightChild);
         case '-':
            return eval(t.leftChild) - eval(t.rightChild);
         case '*':
            return eval(t.leftChild) * eval(t.rightChild);
         case '/':
            return eval(t.leftChild) / eval(t.rightChild);
         default:
            return 0;
      }
   }

   public static void showTree(int n, Node t) {
      if (t == null)
         return;
      tab(n);
      if (t.leftChild == null && t.rightChild == null)
         System.out.println(t.value);
      else {
         System.out.println(t.operation);
         showTree(n + 2, t.leftChild);
         showTree(n + 2, t.rightChild);
      }
   }

   public static void tab(int n) {
      for (int i = 0; i < n; i++)
         System.out.print(" ");
   }
}

// ------------------------------------------------------------
class Node {
   char operation;
   int value;
   Node leftChild;
   Node rightChild;
}

```