~\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)
```

```
50          return false;
51       if (t1.value != t2.value || t1.operation != t2.operation)
52          return false;
53
54       // Recursively check left and right subtrees
55       return isIdentical(t1.leftChild, t2.leftChild) && isIdentical(t1.rightChild,
    t2.rightChild);
56    }
57
58    // ------------------------------------------------------------
59    public static void main(String[] args) {
60       // Define Tree 1
61       Node a = node(2);
62       Node b = node(3);
63       Node c = node('+', a, b);
64       Node d = node(5);
65       Node e = node(1);
66       Node f = node('-', d, e);
67       Node g = node('*', c, f);
68       Node h = node(8);
69       Node i = node('/', g, h);
70
71       // Define Tree 2 (identical to Tree 1)
72       Node a2 = node(2);
73       Node b2 = node(3);
74       Node c2 = node('+', a2, b2);
75       Node d2 = node(5);
76       Node e2 = node(1);
77       Node f2 = node('-', d2, e2);
78       Node g2 = node('*', c2, f2);
79       Node h2 = node(8);
80       Node i2 = node('/', g2, h2);
81
82       System.out.println("Tree:");
83       showTree(0, i);
84       System.out.print("Prefix: ");
85       prefix(i);
86       System.out.print("\nPostfix: ");
87       postfix(i);
88       System.out.print("\nInfix: ");
89       infix(i);
90       System.out.println("\nValue: " + eval(i));
91
92       System.out.println("--------------------");
93
94       // Compute the height of the tree
95       System.out.println("Height of the tree: " + computeHeight(i));
96
97       // Count elements in the tree
98       System.out.println("Number of elements in the tree: " + countElements(i));
99
100      // Count leaves in the tree
101      System.out.println("Number of leaves in the tree: " + countLeaves(i));
102
```

```
103         // Check if the tree is fully balanced
104         System.out.println("Is the tree fully balanced? " + isFullyBalanced(i));
105 
106         // Check if the two trees are identical
107         System.out.println("Are the trees identical? " + isIdentical(i, i2));
108      }
109 
110      // -----------------------------------------------------------
111      public static Node node(char op, Node l, Node r) {
112         Node a = new Node();
113         a.operation = op;
114         a.leftChild = l;
115         a.rightChild = r;
116         return a;
117      }
118 
119      public static Node node(int val) {
120         Node a = new Node();
121         a.value = val;
122         return a;
123      }
124 
125      public static void prefix(Node t) {
126         if (t == null)
127            return;
128         if (t.leftChild == null && t.rightChild == null)
129            System.out.print(t.value + " ");
130         else {
131            System.out.print(t.operation + " ");
132            prefix(t.leftChild);
133            prefix(t.rightChild);
134         }
135      }
136 
137      public static void postfix(Node t) {
138         if (t == null)
139            return;
140         if (t.leftChild == null && t.rightChild == null)
141            System.out.print(t.value + " ");
142         else {
143            postfix(t.leftChild);
144            postfix(t.rightChild);
145            System.out.print(t.operation + " ");
146         }
147      }
148 
149      public static void infix(Node t) {
150         if (t == null)
151            return;
152         if (t.leftChild == null && t.rightChild == null)
153            System.out.print(t.value);
154         else {
155            System.out.print("(");
156            infix(t.leftChild);
```

```
157            System.out.print(t.operation);
158            infix(t.rightChild);
159            System.out.print(")");
160         }
161      }
162
163      public static double eval(Node t) {
164         if (t == null)
165            return 0;
166         if (t.leftChild == null && t.rightChild == null)
167            return t.value;
168
169         switch (t.operation) {
170            case '+':
171               return eval(t.leftChild) + eval(t.rightChild);
172            case '-':
173               return eval(t.leftChild) - eval(t.rightChild);
174            case '*':
175               return eval(t.leftChild) * eval(t.rightChild);
176            case '/':
177               return eval(t.leftChild) / eval(t.rightChild);
178            default:
179               return 0;
180         }
181      }
182
183      public static void showTree(int n, Node t) {
184         if (t == null)
185            return;
186         tab(n);
187         if (t.leftChild == null && t.rightChild == null)
188            System.out.println(t.value);
189         else {
190            System.out.println(t.operation);
191            showTree(n + 2, t.leftChild);
192            showTree(n + 2, t.rightChild);
193         }
194      }
195
196      public static void tab(int n) {
197         for (int i = 0; i < n; i++)
198            System.out.print(" ");
199      }
200   }
201
202   // -----------------------------------------------------------
203   class Node {
204      char operation;
205      int value;
206      Node leftChild;
207      Node rightChild;
208   }
209
```