```java
import components.binarytree.BinaryTree;

/**
 * Utility class with implementation of {@code BinaryTree} static, generic
 * methods height and isInTree.
 *
 * @author Put your name here
 *
 */
public final class BinaryTreeMethods {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private BinaryTreeMethods() {
    }

    /**
     * Returns the height of the given {@code BinaryTree<T>}.
     *
     * @param <T>
     *            the type of the {@code BinaryTree} node labels
     * @param t
     *            the {@code BinaryTree} whose height to return
     * @return the height of the given {@code BinaryTree}
     * @ensures height = ht(t)
     */
    public static <T> int height(BinaryTree<T> t) {
        assert t != null : "Violation of: t is not null";

        int leftH = 0, rightH = 0;
        BinaryTree<T> left = t.newInstance();
        BinaryTree<T> right = t.newInstance();
        T root = null;

        if (t.size() != 0) {

            root = t.root();
            t.disassemble(left, right);
        }

        // This line added just to make the component compilable.
        return 1;
    }

    /**
     * Returns true if the given {@code T} is in the given {@code BinaryTree<T>}
     * or false otherwise.
     *
     * @param <T>
     *            the type of the {@code BinaryTree} node labels
     * @param t
     *            the {@code BinaryTree} to search
     * @param x
     *            the {@code T} to search for
     * @return true if the given {@code T} is in the given {@code BinaryTree},
     *         false otherwise
     * @ensures isInTree = [true if x is in t, false otherwise]
     */
```

```
64      public static <T> boolean isInTree(BinaryTree<T> t, T x) {
65          assert t != null : "Violation of: t is not null";
66          assert x != null : "Violation of: x is not null";
67
68          boolean inLeft = false, inRight = false, inRoot = false;
69          BinaryTree<T> left = t.newInstance();
70          BinaryTree<T> right = t.newInstance();
71          T root = null;
72
73          if (t.size() != 0) {
74
75              root = t.root();
76              t.disassemble(left, right);
77
78              inRoot = x.equals(root);
79              inLeft = isInTree(left, x);
80              inRight = isInTree(right, x);
81
82              t.assemble(root, left, right);
83          }
84
85          return inLeft || inRight || inRoot;
86      }
87
88      /**
89       * Main method.
90       *
91       * @param args
92       *            the command line arguments
93       */
94      public static void main(String[] args) {
95          SimpleReader in = new SimpleReader1L();
96          SimpleWriter out = new SimpleWriter1L();
97
98          out.print("Input a tree (or just press Enter to terminate): ");
99          String str = in.nextLine();
100         while (str.length() > 0) {
101             BinaryTree<String> t = BinaryTreeUtility.treeFromString(str);
102             out.println("Tree = " + BinaryTreeUtility.treeToString(t));
103             out.println("Height = " + height(t));
104             out.print("  Input a label to search "
105                     + "(or just press Enter to input a new tree): ");
106             String label = in.nextLine();
107             while (label.length() > 0) {
108                 if (isInTree(t, label)) {
109                     out.println("    \"" + label + "\" is in the tree");
110                 } else {
111                     out.println("    \"" + label + "\" is not in the tree");
112                 }
113                 out.print("  Input a label to search "
114                         + "(or just press Enter to input a new tree): ");
115                 label = in.nextLine();
116             }
117             out.println();
118             out.print("Input a tree (or just press Enter to terminate): ");
119             str = in.nextLine();
120         }
121
122         in.close();
```

```
123         out.close();
124     }
125
126 }
127
```