```java
import components.binarytree.BinaryTree;

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

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

    /**
     * Returns the {@code String} prefix representation of the given
     * {@code BinaryTree<T>}.
     *
     * @param <T>
     *            the type of the {@code BinaryTree} node labels
     * @param t
     *            the {@code BinaryTree} to convert to a {@code String}
     * @return the prefix representation of {@code t}
     * @ensures treeToString = [the String prefix representation of t]
     */
    public static <T> String treeToString(BinaryTree<T> t) {
        // i need help with understanding what are we turning into a string exactly
        // the whole tree? the root? or one value inside the tree?
        return t.toString();
    }

    /**
     * Returns a copy of the the given {@code BinaryTree}.
     *
     * @param t
     *            the {@code BinaryTree} to copy
     * @return a copy of the given {@code BinaryTree}
     * @ensures copy = t
     */
    public static BinaryTree<Integer> copy(BinaryTree<Integer> t) {
        BinaryTree<Integer> copied = new BinaryTree<>(t);
        return copied;
    }

//    /**
//     * 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)
//     */
```

```
58 //    public static <T> int height(BinaryTree<T> t) {
59 //        assert t != null : "Violation of: t is not null";
60 //        int height = 0;
61 //        BinaryTree<T> ls = t.newInstance();
62 //        BinaryTree<T> rs = t.newInstance();
63 //        int leftS = 0;
64 //        int rightS = 0;
65 //        if (t.size() > 0) {
66 //            T root = t.disassemble(ls, rs);
67 //            leftS = leftS + height(ls) + 1;
68 //            rightS = rightS + height(rs) + 1;
69 //            t.assemble(root, ls, rs);
70 //        }
71 //        height = Math.max(leftS, rightS);
72 //        return height;
73 //    }
74 //
75 //    /**
76 //     * Returns true if the given {@code T} is in the given {@code BinaryTree<T>}
77 //     * or false otherwise.
78 //     *
79 //     * @param <T>
80 //     *            the type of the {@code BinaryTree} node labels
81 //     * @param t
82 //     *            the {@code BinaryTree} to search
83 //     * @param x
84 //     *            the {@code T} to search for
85 //     * @return true if the given {@code T} is in the given {@code BinaryTree},
86 //     *         false otherwise
87 //     * @ensures isInTree = [true if x is in t, false otherwise]
88 //     */
89 //    public static <T> boolean isInTree(BinaryTree<T> t, T x) {
90 //        assert t != null : "Violation of: t is not null";
91 //        assert x != null : "Violation of: x is not null";
92 //        boolean check = false;
93 //        for (T w : t) {
94 //            if (w.equals(x)) {
95 //                check = true;
96 //            }
97 //        }
98 //        return check;
99 //    }
100 //
101 //    /**
102 //     * Main method.
103 //     *
104 //     * @param args
105 //     *            the command line arguments
106 //     */
107 //    public static void main(String[] args) {
108 //        SimpleReader in = new SimpleReader1L();
109 //        SimpleWriter out = new SimpleWriter1L();
110 //
111 //        out.print("Input a tree (or just press Enter to terminate): ");
112 //        String str = in.nextLine();
113 //        while (str.length() > 0) {
114 //            BinaryTree<String> t = BinaryTreeUtility.treeFromString(str);
```

```
115 //            out.println("Tree = " + BinaryTreeUtility.treeToString(t));
116 //            out.println("Height = " + height(t));
117 //            out.print("  Input a label to search "
118 //                    + "(or just press Enter to input a new tree): ");
119 //            String label = in.nextLine();
120 //            while (label.length() > 0) {
121 //                if (isInTree(t, label)) {
122 //                    out.println("    \"" + label + "\" is in the tree");
123 //                } else {
124 //                    out.println("    \"" + label + "\" is not in the tree");
125 //                }
126 //                out.print("  Input a label to search "
127 //                        + "(or just press Enter to input a new tree): ");
128 //                label = in.nextLine();
129 //            }
130 //            out.println();
131 //            out.print("Input a tree (or just press Enter to terminate): ");
132 //            str = in.nextLine();
133 //        }
134 //
135 //        in.close();
136 //        out.close();
137 //    }
138
139 }
140
```