```java
import components.binarytree.BinaryTree;

/**
 * Utility class with implementation of binary search tree static, generic
 * methods isInTree (and removeSmallest).
 *
 * @mathdefinitions <pre>
 * IS_BST(
 *   tree: binary tree of T
 *  ): boolean satisfies
 *  [tree satisfies the binary search tree properties as described in the
 *   slides with the ordering reported by compareTo for T, including that
 *   it has no duplicate labels]
 * </pre>
 *
 * @author Put your name here
 *
 */
public final class BinarySearchTreeMethods {

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

    /**
     * Returns whether {@code x} is in {@code t}.
     *
     * @param <T>
     *            type of {@code BinaryTree} labels
     * @param t
     *            the {@code BinaryTree} to be searched
     * @param x
     *            the label to be searched for
     * @return true if t contains x, false otherwise
     * @requires IS_BST(t)
     * @ensures isInTree = (x is in labels(t))
     */
    public static <T extends Comparable<T>> boolean isInTree(BinaryTree<T> t,
            T x) {

        boolean inTree = false;
        BinaryTree<T> left = t.newInstance();
        BinaryTree<T> right = t.newInstance();
        T node = t.root();

        if (t.size() > 1) {

            t.disassemble(left, right);

            // goto left or right branch
            if (node.compareTo(x) == -1) {
                inTree = isInTree(left, x);
            } else if (node.compareTo(x) == 1) {
                inTree = isInTree(right, x);
            }

            t.assemble(node, left, right);
```

```
65
66          }
67
68          if (!inTree) {
69              inTree = node.equals(x);
70          }
71
72          // This line added just to make the component compilable.
73          return inTree;
74      }
75
76      /**
77       * Removes and returns the smallest (left-most) label in {@code t}.
78       *
79       * @param <T>
80       *            type of {@code BinaryTree} labels
81       * @param t
82       *            the {@code BinaryTree} from which to remove the label
83       * @return the smallest label in the given {@code BinaryTree}
84       * @updates t
85       * @requires IS_BST(t) and |t| > 0
86       * @ensures <pre>
87       * IS_BST(t)  and  removeSmallest = [the smallest label in #t]  and
88       *  labels(t) = labels(#t) \ {removeSmallest}
89       * </pre>
90       */
91      public static <T> T removeSmallest(BinaryTree<T> t) {
92
93          // TODO - fill in body
94
95          // This line added just to make the component compilable.
96          return null;
97      }
98
99      /**
100      * Main method.
101      *
102      * @param args
103      *            the command line arguments
104      */
105     public static void main(String[] args) {
106         SimpleReader in = new SimpleReader1L();
107         SimpleWriter out = new SimpleWriter1L();
108
109         /*
110          * Input tree labels and construct BST.
111          */
112         out.println("Input the distinct labels for a binary search tree "
113                 + "in the order in which you want them inserted.");
114         out.println("Press Enter on an empty line to terminate your input.");
115         out.println();
116         out.print("Next label: ");
117         String str = in.nextLine();
118         BinaryTree<String> t = new BinaryTree1<String>();
119         while (str.length() > 0) {
120             BinaryTreeUtility.insertInTree(t, str);
121             out.println();
122             out.println("t = " + BinaryTreeUtility.treeToString(t));
123             out.println();
```

```
124             out.print("Next label: ");
125             str = in.nextLine();
126         }
127         /*
128          * Input strings and check whether each is in the BST or not.
129          */
130         out.println();
131         out.print("  Input a label to search "
132                 + "(or just press Enter to input a new tree): ");
133         String label = in.nextLine();
134         while (label.length() > 0) {
135             if (isInTree(t, label)) {
136                 out.println("    \"" + label + "\" is in the tree");
137             } else {
138                 out.println("    \"" + label + "\" is not in the tree");
139             }
140             out.print("  Input a label to search "
141                     + "(or just press Enter to terminate the program): ");
142             label = in.nextLine();
143         }
144         /*
145          * Output BST labels in order.
146          */
147 //         out.println();
148 //         out.println("Labels in BST in order:");
149 //         while (t.size() > 0) {
150 //             label = removeSmallest(t);
151 //             out.println("   " + label);
152 //         }
153 
154         in.close();
155         out.close();
156     }
157 }
158 
```