```java
import static org.junit.Assert.assertEquals;

/**
 * JUnit test fixture for {@code BinarySearchTreeMethods}'s static methods
 * isInTree (and removeSmallest).
 */
public final class BinarySearchTreeMethodsTest {

    /**
     * Constructs and return a BST created by inserting the given {@code args}
     * into an empty tree in the order in which they are provided.
     *
     * @param args
     *            the {@code String}s to be inserted in the tree
     * @return the BST with the given {@code String}s
     * @requires [the Strings in args are all distinct]
     * @ensures createBSTFromArgs = [the BST with the given Strings]
     */
    private static BinaryTree<String> createBSTFromArgs(String... args) {
        BinaryTree<String> t = new BinaryTree1<String>();
        for (String s : args) {
            BinaryTreeUtility.insertInTree(t, s);
        }
        return t;
    }

    @Test
    public void inTreeTest1() {
        /*
         * Set up variables
         */
        BinaryTree<String> t1 = createBSTFromArgs("b", "a", "c");
        BinaryTree<String> t2 = createBSTFromArgs("b", "a", "c");
        /*
         * Call method under test
         */
        boolean inTree = BinarySearchTreeMethods.isInTree(t1, "a");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(true, inTree);
        assertEquals(t2, t1);
    }

    @Test
    public void inTreeTest2() {
        /*
         * Set up variables
         */
        BinaryTree<String> t1 = createBSTFromArgs("b", "a", "c");
        BinaryTree<String> t2 = createBSTFromArgs("b", "a", "c");
        /*
         * Call method under test
         */
        boolean inTree = BinarySearchTreeMethods.isInTree(t1, "b");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(true, inTree);
        assertEquals(t2, t1);
    }

    @Test
    public void inTreeTest3() {
        /*
         * Set up variables
         */
        BinaryTree<String> t1 = createBSTFromArgs("b", "a", "c");
        BinaryTree<String> t2 = createBSTFromArgs("b", "a", "c");
        /*
         * Call method under test
         */
        boolean inTree = BinarySearchTreeMethods.isInTree(t1, "c");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(true, inTree);
        assertEquals(t2, t1);
    }

    @Test
    public void inTreeTest4() {
        /*
         * Set up variables
         */
        BinaryTree<String> t1 = createBSTFromArgs("b", "a", "c");
        BinaryTree<String> t2 = createBSTFromArgs("b", "a", "c");
        /*
         * Call method under test
         */
        boolean inTree = BinarySearchTreeMethods.isInTree(t1, "d");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(false, inTree);
        assertEquals(t2, t1);
    }

    @Test
    public void inTreeTest5() {
        /*
         * Set up variables
         */
        BinaryTree<String> t1 = createBSTFromArgs("b");
        BinaryTree<String> t2 = createBSTFromArgs("b");
        /*
         * Call method under test
         */
        boolean inTree = BinarySearchTreeMethods.isInTree(t1, "b");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(true, inTree);
        assertEquals(t2, t1);
    }

    @Test
    public void inTreeTest6() {
        /*
         * Set up variables
         */
        BinaryTree<String> t1 = createBSTFromArgs("b");
        BinaryTree<String> t2 = createBSTFromArgs("b");
        /*
         * Call method under test
         */
        boolean inTree = BinarySearchTreeMethods.isInTree(t1, "a");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(false, inTree);
        assertEquals(t2, t1);
    }

    // TODO: add here other test cases for BinarySearchTreeMethods.isInTree
    // (and for BinarySearchTreeMethods.removeSmallest)

}

```