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

/**
 * JUnit test fixture for {@code Set<String>}'s constructor and kernel methods.
 *
 * @author Zhuoyang Li + Xinci Ma
 *
 */
public abstract class SetTest {

    /**
     * Invokes the appropriate {@code Set} constructor for the implementation
     * under test and returns the result.
     *
     * @return the new set
     * @ensures constructorTest = {}
     */
    protected abstract Set<String> constructorTest();

    /**
     * Invokes the appropriate {@code Set} constructor for the reference
     * implementation and returns the result.
     *
     * @return the new set
     * @ensures constructorRef = {}
     */
    protected abstract Set<String> constructorRef();

    /**
     * Creates and returns a {@code Set<String>} of the implementation under
     * test type with the given entries.
     *
     * @param args
     *            the entries for the set
     * @return the constructed set
     * @requires [every entry in args is unique]
     * @ensures createFromArgsTest = [entries in args]
     */
    private Set<String> createFromArgsTest(String... args) {
        Set<String> set = this.constructorTest();
        for (String s : args) {
            assert !set.contains(
                    s) : "Violation of: every entry in args
is unique";
            set.add(s);
        }
        return set;
    }

    /**
     * Creates and returns a {@code Set<String>} of the reference implementation
     * type with the given entries.
     *
     * @param args
     *             the entries for the set
     * @return the constructed set
     * @requires [every entry in args is unique]
     * @ensures createFromArgsRef = [entries in args]
     */
    private Set<String> createFromArgsRef(String... args) {
        Set<String> set = this.constructorRef();
        for (String s : args) {
            assert !set.contains(
                    s) : "Violation of: every entry in args is unique";
            set.add(s);
        }
        return set;
    }

    /**
     * Tests the default constructor by creating two sets using the tested
     * constructor and the reference constructor, then asserts that both sets
     * are equal.
     */
    @Test
    public final void testConstructor() {
        Set<String> s = this.constructorTest();
        Set<String> sExpected = this.constructorRef();
        assertEquals(sExpected, s);
    }

    @Test
    public final void testAddToEmptySet() {
        Set<String> s = this.createFromArgsTest();
        Set<String> sExpected = this.createFromArgsTest("apple");
        s.add("apple");
        assertEquals(s, sExpected);
    }

    @Test
    public final void testAddToSetWithOneEntry() {
        Set<String> s = this.createFromArgsTest("apple");
        Set<String> sExpected = this.createFromArgsTest("apple", "banana");
        s.add("banana");
        assertEquals(s, sExpected);
    }

    @Test
    public final void testAddToSetWithThreeEntries() {
        Set<String> s = this.createFromArgsTest("apple", "banana", "orange");
        Set<String> sExpected = this.createFromArgsTest("apple", "banana",
                "orange", "kiwi");
        s.add("kiwi");
        assertEquals(s, sExpected);
    }

    @Test
    public final void testRemoveLastEntry() {
        Set<String> s = this.createFromArgsTest("apple");
        Set<String> sExpected = this.createFromArgsTest();
        String removed = s.remove("apple");
        assertEquals("apple", removed);
        assertEquals(sExpected, s);
    }

    @Test
    public final void testRemoveAnyFromSetWithOneEntry() {
        Set<String> s = this.createFromArgsTest("apple");
        Set<String> sExpected = this.createFromArgsRef();
        String removed = s.removeAny();
        assertEquals(sExpected, s);
        assertEquals("apple", removed);
    }

    @Test
    public final void testRemoveAnyFromSetWithMultipleEntries() {
        Set<String> s = this.createFromArgsTest("apple", "banana", "orange");
        Set<String> sExpected = this.createFromArgsRef("apple", "banana",
                "orange");
        String removed = s.removeAny();
        assertTrue(sExpected.contains(removed));
        sExpected.remove(removed);
        assertEquals(sExpected, s);
    }

    @Test
    public final void testContainsWithOneEntry() {
        Set<String> s = this.createFromArgsTest("apple");
        assertTrue(s.contains("apple"));
    }

    @Test
    public final void testDoesNotContainWithOneEntry() {
        Set<String> s = this.createFromArgsTest("apple");
        assertFalse(s.contains("banana"));
    }

    @Test
    public final void testContainsWithThreeEntries() {
        Set<String> s = this.createFromArgsTest("apple", "banana", "orange");
        assertTrue(s.contains("banana"));
    }

    @Test
    public final void testDoesNotContainWithThreeEntries() {
        Set<String> s = this.createFromArgsTest("apple", "banana", "orange");
        assertFalse(s.contains("kiwi"));
    }

    @Test
    public final void testContainsWithEmptySet() {
        Set<String> s = this.createFromArgsTest();
        assertFalse(s.contains("apple"));
    }

    @Test
    public final void testSizeWithNoEntries() {
        Set<String> s = this.createFromArgsTest();
        assertEquals(0, s.size());
    }

    @Test
    public final void testSizeWithOneEntry() {
        Set<String> s = this.createFromArgsTest("apple");
        assertEquals(1, s.size());
    }

    @Test
    public final void testSizeWithThreeEntries() {
        Set<String> s = this.createFromArgsTest("apple",
 "banana", "orange");
        assertEquals(3, s.size());
    }
}

```