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

/**
 * JUnit test fixture for {@code Set<String>}'s constructor and kernel methods.
 *
 * @author Put your name here
 *
 */
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]
```

```
62      */
63     private Set<String> createFromArgsRef(String... args) {
64         Set<String> set = this.constructorRef();
65         for (String s : args) {
66             assert !set.contains(
67                     s) : "Violation of: every entry in args is unique";
68             set.add(s);
69         }
70         return set;
71     }
72
73     // TODO - add test cases for constructor, add, remove, removeAny, contains, and size
74     // CONSTRUCTOR ------------------------------------------------------------
75     /**
76      * Testing no argument constructor.
77      */
78     @Test
79     public final void testNoArugementConstructor() {
80         // Setup
81         Set<String> s = this.createFromArgsTest();
82         Set<String> sExpected = this.createFromArgsRef();
83         // Call
84         // Eval
85         assertEquals(sExpected, s);
86     }
87
88     /**
89      * Testing two argument constructor.
90      */
91     @Test
92     public final void testTwoArugementConstructor() {
93         // Setup
94         Set<String> s = this.createFromArgsTest("a", "b");
95         Set<String> sExpected = this.createFromArgsRef("a", "b");
96         // Call
97         // Eval
98         assertEquals(sExpected, s);
99     }
100
101    // ADD --------------------------------------------------------------------
102    /**
103     * Testing add(); to empty.
104     */
105    @Test
106    public final void testAddToEmpty() {
107        // Setup
108        Set<String> s = this.createFromArgsTest();
109        Set<String> sExpected = this.createFromArgsRef("a");
110        // Call
111        s.add("a");
112        // Eval
113        assertEquals(sExpected, s);
114    }
115
116    /**
117     * Testing add(); to non-empty.
118     */
```

```java
    @Test
    public final void testAddToNonEmpty() {
        // Setup
        Set<String> s = this.createFromArgsTest("a", "b", "c");
        Set<String> sExpected = this.createFromArgsRef("a", "b", "c", "e");
        // Call
        s.add("e");
        // Eval
        assertEquals(sExpected, s);
    }

    // REMOVE -----------------------------------------------------------------
    /**
     * Testing remove(); to empty.
     */
    @Test
    public final void testRemoveToEmpty() {
        // Setup
        Set<String> s = this.createFromArgsTest("a");
        Set<String> sExpected = this.createFromArgsRef();
        // Call
        s.remove("a");
        // Eval
        assertEquals(sExpected, s);
    }

    /**
     * Testing remove(); to non-empty.
     */
    @Test
    public final void testRemoveToNonEmpty() {
        // Setup
        Set<String> s = this.createFromArgsTest("a", "b", "c", "e");
        Set<String> sExpected = this.createFromArgsRef("a", "b", "c");
        // Call
        s.remove("e");
        // Eval
        assertEquals(sExpected, s);
    }

    // REMOVEANY --------------------------------------------------------------
    /**
     * Testing removeAny() with set.size() == 1.
     */
    @Test
    public final void testRemoveAnySizeOne() {
        // Setup
        Set<String> s = this.createFromArgsTest("a");
        Set<String> sExpected = this.createFromArgsRef("a");
        String removedEntry;
        // Call
        removedEntry = s.removeAny();
        // Eval
        assertEquals(sExpected.contains(removedEntry), true);
        sExpected.remove(removedEntry);
        assertEquals(sExpected, s);
    }
```

```java

    /**
     * Testing removeAny() with set.size() == 3.
     */
    @Test
    public final void testRemoveAnySizeThree() {
        // Setup
        Set<String> s = this.createFromArgsTest("a", "b", "c");
        Set<String> sExpected = this.createFromArgsRef("a", "b", "c");
        String removedEntry;
        // Call
        removedEntry = s.removeAny();
        // Eval
        assertEquals(sExpected.contains(removedEntry), true);
        sExpected.remove(removedEntry);
        assertEquals(sExpected, s);
    }

    // CONTAINS ------------------------------------------------------------------
    /**
     * Testing contains(); return true.
     */
    @Test
    public final void testContainsTrue() {
        // Setup
        Set<String> s = this.createFromArgsTest("a", "b", "c", "e");
        // Call
        // Eval
        assertEquals(true, s.contains("a"));
    }

    /**
     * Testing contains(); return false.
     */
    @Test
    public final void testContainsFalse() {
        // Setup
        Set<String> s = this.createFromArgsTest("a", "b", "c", "e");
        // Call
        // Eval
        assertEquals(false, s.contains("d"));
    }
    // SIZE ----------------------------------------------------------------------

    /**
     * Testing size(); set = {}.
     */
    @Test
    public final void testSize0() {
        // Setup
        Set<String> s = this.createFromArgsTest();
        // Call
        // Eval
        assertEquals(0, s.size());
    }

```

```
233     /**
234      * Testing size(); non empty set.
235      */
236     @Test
237     public final void testSizeNonEmpty() {
238         // Setup
239         Set<String> s = this.createFromArgsTest("a", "b", "c", "e");
240         // Call
241         // Eval
242         assertEquals(4, s.size());
243     }
244 
245 }
246 
```