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

import org.junit.Test;

import components.set.Set;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * JUnit test fixture for {@code Set<String>}'s constructor and
kernel methods.
 *
 * @author Bashir Ali and Kwasi Fosu
 *
 */
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);
        }
```

```java
        return set;
    }

    /**
     * Test cases for constructors
     */

    @Test
    public final void testNoArgumentConstructor() {
        /*
         * Set up variables and call method under test
         */
        Set<String> s = this.constructorTest();
        Set<String> sExpected = this.constructorRef();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    /**
     * Test cases for kernel methods
     */

    @Test
    public final void testAddEmpty() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest();
        Set<String> sExpected = this.createFromArgsRef("red");
        /*
         * Call method under test
         */
        s.add("red");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    @Test
    public final void testAddNonEmptyOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red");
        Set<String> sExpected = this.createFromArgsRef("red",
"blue");
        /*
         * Call method under test
         */
        s.add("blue");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    @Test
    public final void testAddNonEmptyMoreThanOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red", "blue",
"green");
        Set<String> sExpected = this.createFromArgsRef("red",
"blue", "green",
                "yellow");
        /*
         * Call method under test
         */
        s.add("yellow");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    @Test
    public final void testRemoveLeavingEmpty() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red");
        Set<String> sExpected = this.createFromArgsRef();
        /*
         * Call method under test
         */
        String x = s.remove("red");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals("red", x);
    }

    @Test
    public final void testRemoveLeavingOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red", "blue");
        Set<String> sExpected = this.createFromArgsRef("red");
        /*
         * Call method under test
         */
        String x = s.remove("blue");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals("blue", x);
    }

    @Test
    public final void testRemoveLeavingMoreThanOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red", "green",
"blue");
        Set<String> sExpected = this.createFromArgsRef("red",
"blue");
        SimpleWriter out = new SimpleWriter1L();
        /*
         * Call method under test
         */
        String x = s.remove("green");
        out.print(x);
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals("green", x);
    }

    @Test
    public final void testSizeEmpty() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest();
        Set<String> sExpected = this.createFromArgsRef();
        /*
         * Call method under test
         */
        int i = s.size();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals(0, i);
    }

    @Test
```

```java
    public final void testSizeOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red");
        Set<String> sExpected = this.createFromArgsRef("red");
        /*
         * Call method under test
         */
        int i = s.size();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals(1, i);
    }

    @Test
    public final void testSizeMoreThanOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red", "blue",
"green");
        Set<String> sExpected = this.createFromArgsRef("red",
"blue", "green");
        /*
         * Call method under test
         */
        int i = s.size();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals(3, i);
    }

    @Test
    public final void testContainsEmpty() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest();
        Set<String> sExpected = this.createFromArgsRef();
        /*
         * Call method under test
         */
        boolean contains = s.contains("red");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals(false, contains);
    }

    @Test
    public final void testTrueWhenContainsOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red");
        Set<String> sExpected = this.createFromArgsRef("red");
        /*
         * Call method under test
         */
        boolean containsTrue = s.contains("red");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals(true, containsTrue);
    }

    @Test
    public final void testFalseWhenContainsOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red");
        Set<String> sExpected = this.createFromArgsRef("red");
        /*
         * Call method under test
         */
        boolean containsFalse = s.contains("blue");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals(false, containsFalse);
    }

    @Test
    public final void testTrueWhenContainsMany() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red", "green",
"yellow");
        Set<String> sExpected = this.createFromArgsRef("red",
"green",
                "yellow");
        /*
         * Call method under test
         */
        boolean containsTrue = s.contains("red");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals(true, containsTrue);
    }

    @Test
    public final void testFalseWhenContainsMany() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red", "green", "yellow");
        Set<String> sExpected = this.createFromArgsRef("red", "green",
                "yellow");
        /*
         * Call method under test
         */
        boolean containsFalse = s.contains("blue");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals(false, containsFalse);
    }

    @Test
    public final void testRemoveAnyLeavingEmpty() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red");
        Set<String> sExpected = this.createFromArgsRef();
        /*
         * Call method under test
         */
        String removed = s.removeAny();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals("red", removed);
    }

    @Test
    public final void testRemoveAnyLeavingOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red", "blue");

        /*
         * Call method under test
         */
        String removed = s.removeAny();
        int size = s.size();
        int expectedSize = 1;

        /*
         * Assert that values of variables match expectations
         */
        assertEquals(expectedSize, size);
    }

    @Test
    public final void testRemoveAnyLeavingMoreThanOne() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("red", "blue",
 "green");

        /*
         * Call method under test
         */
        String removed = s.removeAny();
        int size = s.size();
        int expectedSize = 2;

        /*
         * Assert that values of variables match expectations
         */
        assertEquals(expectedSize, size);
    }

}
```