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

/**
 * JUnit test fixture for {@code Set<String>}'s constructor and kernel methods.
 *
 * @author Yunlong Zhang
 *
 */
public abstract class SetTest {

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

    /**
     * Invokes the appropriate {@code Set} constructor 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) {
```

```
64              assert !set.contains(
65                      s) : "Violation of: every entry in args is unique";
66              set.add(s);
67          }
68          return set;
69      }
70
71      @Test
72      public void testAdd() {
73          Set<String> s1 = this.createFromArgsRef("a", "b", "c", "d", "e");
74          Set<String> s2 = this.createFromArgsRef("a", "b", "c", "d");
75          String ad = "e";
76          s1.add(ad);
77          assertEquals(s1, s2);
78
79      }
80
81      @Test
82      public void testRemove() {
83          Set<String> s1 = this.createFromArgsRef("a", "b", "c", "d");
84          Set<String> s2 = this.createFromArgsRef("a", "b", "c", "d", "e");
85          String ad = "e";
86          s1.remove(ad);
87          assertEquals(s1, s2);
88
89      }
90
91      @Test
92      public void testContains() {
93          boolean result = true;
94          Set<String> s2 = this.createFromArgsRef("a", "b", "c", "d", "e");
95          String ad = "c";
96          boolean test = s2.contains(ad);
97          assertEquals(result, test);
98
99      }
100
101     @Test
102     public void testSize() {
103         Set<String> s2 = this.createFromArgsRef("a", "b", "c", "d", "e");
104         int cont = s2.size();
105         int result = 5;
106         assertEquals(result, cont);
107
108     }
109
110     @Test
111     public void testRemoveAny() {
112         Set<String> s2 = this.createFromArgsRef("a", "b", "c", "d", "e");
113         Set<String> s1 = this.createFromArgsRef("a", "b", "c", "d", "e");
114         String ad = s2.removeAny();
115         boolean contain = s1.contains(ad);
116         boolean containResult = true;
117         boolean size;
118         boolean sizeResult = true;
119         int sizeS2 = s2.size() + 1;
120         if (s1.size() == sizeS2) {
121             size = true;
122         } else {
```

```
123             size = false;
124         }
125         assertEquals(containResult, contain);
126         assertEquals(sizeResult, size);
127 
128     }
129 
130 }
```