

```
1 import static org.junit.Assert.assertEquals;
2 import static org.junit.Assert.assertFalse;
3 import static org.junit.Assert.assertTrue;
4
5 import org.junit.Test;
6
7 import components.set.Set;
8
9 /**
10  * JUnit test fixture for {@code Set<String>}'s constructor and kernel methods.
11  *
12  * @author Put your name here
13  *
14  */
15 public abstract class SetTest {
16
17     /**
18      * Invokes the appropriate {@code Set} constructor for the implementation
19      * under test and returns the result.
20      *
21      * @return the new set
22      * @ensures constructorTest = {}
23      */
24     protected abstract Set<String> constructorTest();
25
26     /**
27      * Invokes the appropriate {@code Set} constructor for the reference
28      * implementation and returns the result.
29      *
30      * @return the new set
31      * @ensures constructorRef = {}
32      */
33     protected abstract Set<String> constructorRef();
34
35     /**
36      * Creates and returns a {@code Set<String>} of the implementation under
37      * test type with the given entries.
38      *
39      * @param args
40      *         the entries for the set
41      * @return the constructed set
42      * @requires [every entry in args is unique]
43      * @ensures createFromArgsTest = [entries in args]
44      */
45     private Set<String> createFromArgsTest(String... args) {
46         Set<String> set = this.constructorTest();
47         for (String s : args) {
48             assert !set.contains(
49                 s) : "Violation of: every entry in args is unique";
50             set.add(s);
51         }
52         return set;
53     }
54
55     /**
56      * Creates and returns a {@code Set<String>} of the reference implementation
57      * type with the given entries.
```

```
58     *
59     * @param args
60     *         the entries for the set
61     * @return the constructed set
62     * @requires [every entry in args is unique]
63     * @ensures createFromArgsRef = [entries in args]
64     */
65     private Set<String> createFromArgsRef(String... args) {
66         Set<String> set = this.constructorRef();
67         for (String s : args) {
68             assert !set.contains(
69                 s) : "Violation of: every entry in args is unique";
70             set.add(s);
71         }
72         return set;
73     }
74
75     /**
76     * Tests adding a new element to an empty set.
77     */
78     @Test
79     public void testAddToEmptySet() {
80         Set<String> set = this.createFromArgsTest();
81         Set<String> expectedSet = this.createFromArgsRef("Alice");
82         set.add("Alice");
83         assertEquals(set, expectedSet);
84     }
85
86     /**
87     * Tests adding a new element to a non-empty set.
88     */
89     @Test
90     public void testAddElementToNonEmptySet() {
91         Set<String> set = this.createFromArgsTest("Alice", "Bob");
92         Set<String> expectedSet = this.createFromArgsRef("Alice", "Bob",
93             "Charlie");
94         set.add("Charlie");
95         assertEquals(set, expectedSet);
96     }
97
98     /**
99     * Tests removing the only element in the set.
100    */
101    @Test
102    public void testRemoveElementLeavingEmptySet() {
103        Set<String> set = this.createFromArgsTest("David");
104        Set<String> expectedSet = this.createFromArgsRef();
105        set.remove("David");
106        assertEquals(set, expectedSet);
107    }
108
109    /**
110    * Tests removing an element from a non-empty set.
111    */
112    @Test
113    public void testRemoveElementFromNonEmptySet() {
114        Set<String> set = this.createFromArgsTest("Alice", "Bob", "David");
```

```
115     Set<String> expectedSet = this.createFromArgsRef("Alice", "Bob");
116     set.remove("David");
117     assertEquals(set, expectedSet);
118 }
119
120 /**
121  * Tests removing any element from a set with multiple elements.
122  */
123 @Test
124 public void testRemoveAnyElementFromMultiple() {
125     Set<String> set = this.createFromArgsTest("Alice", "Bob", "David");
126     Set<String> expectedSet = this.createFromArgsRef("Alice", "Bob",
127         "David");
128     String removedElement = set.removeAny();
129     assertTrue(expectedSet.contains(removedElement));
130     expectedSet.remove(removedElement);
131     assertEquals(set, expectedSet);
132 }
133
134 /**
135  * Tests removing any element from a set with a single element.
136  */
137 @Test
138 public void testRemoveAnyElementFromSingle() {
139     Set<String> set = this.createFromArgsTest("Alice");
140     Set<String> expectedSet = this.createFromArgsRef("Alice");
141     String removedElement = set.removeAny();
142     assertTrue(expectedSet.contains(removedElement));
143     expectedSet.remove(removedElement);
144     assertEquals(set, expectedSet);
145 }
146
147 /**
148  * Tests checking the presence of an element in a single-element set.
149  */
150 @Test
151 public void testContainsSingleElement() {
152     Set<String> set = this.createFromArgsTest("Alice");
153     assertTrue(set.contains("Alice"));
154 }
155
156 /**
157  * Tests checking the presence of elements in a multiple-element set.
158  */
159 @Test
160 public void testContainsMultipleElements() {
161     Set<String> set = this.createFromArgsTest("Alice", "Bob", "Charlie");
162     assertTrue(set.contains("Alice"));
163     assertTrue(set.contains("Bob"));
164     assertTrue(set.contains("Charlie"));
165 }
166
167 /**
168  * Tests checking the absence of an element in a set.
169  */
170 @Test
171 public void testContainsNotFound() {
```

```
172     Set<String> set = this.createFromArgsTest("Alice", "Bob", "Charlie");
173     assertFalse(set.contains("David"));
174 }
175
176 /**
177  * Tests the size of an empty set.
178  */
179 @Test
180 public void testSizeOfEmptySet() {
181     Set<String> set = this.createFromArgsTest();
182     int size = set.size();
183     int expectedSize = 0;
184     assertEquals(size, expectedSize);
185 }
186
187 /**
188  * Tests the size of a set with a single element.
189  */
190 @Test
191 public void testSizeOfSingleElementSet() {
192     Set<String> set = this.createFromArgsTest("Alice");
193     int size = set.size();
194     int expectedSize = 1;
195     assertEquals(size, expectedSize);
196 }
197
198 /**
199  * Tests the size of a set with multiple elements.
200  */
201 @Test
202 public void testSizeOfMultipleElementsSet() {
203     Set<String> set = this.createFromArgsTest("Alice", "Bob", "Charlie");
204     int size = set.size();
205     int expectedSize = 3;
206     assertEquals(size, expectedSize);
207 }
208 }
209
```