

```
1 import static org.junit.Assert.assertEquals;
2
3 /**
4  * JUnit test fixture for {@code Set<String>}'s constructor and kernel methods.
5  *
6  * @author Put your name here
7  */
8 public abstract class SetTest {
9
10     /**
11      * Invokes the appropriate {@code Set} constructor for the implementation
12      * under test and returns the result.
13      *
14      * @return the new set
15      * @ensures constructorTest = {}
16      */
17     protected abstract Set<String> constructorTest();
18
19     /**
20      * Invokes the appropriate {@code Set} constructor for the reference
21      * implementation and returns the result.
22      *
23      * @return the new set
24      * @ensures constructorRef = {}
25      */
26     protected abstract Set<String> constructorRef();
27
28     /**
29      * Creates and returns a {@code Set<String>} of the implementation under
30      * test type with the given entries.
31      *
32      * @param args
33      *         the entries for the set
34      * @return the constructed set
35      * @requires [every entry in args is unique]
36      * @ensures createFromArgsTest = [entries in args]
37      */
38     private Set<String> createFromArgsTest(String... args) {
39         Set<String> set = this.constructorTest();
40         for (String s : args) {
41             assert !set.contains(
42                 s) : "Violation of: every entry in args is unique";
43             set.add(s);
44         }
45         return set;
46     }
47
48     /**
49      * Creates and returns a {@code Set<String>} of the reference implementation
50      * type with the given entries.
51      *
52      * @param args
53      *         the entries for the set
54      * @return the constructed set
55      * @requires [every entry in args is unique]
56      * @ensures createFromArgsRef = [entries in args]
57      */
58 }
```

```
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 testNoArgumentConstructor() {
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 testTwoArgumentConstructor() {
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      */
```

```
119     @Test
120     public final void testAddToNonEmpty() {
121         // Setup
122         Set<String> s = this.createFromArgsTest("a", "b", "c");
123         Set<String> sExpected = this.createFromArgsRef("a", "b", "c", "e");
124         // Call
125         s.add("e");
126         // Eval
127         assertEquals(sExpected, s);
128     }
129
130     // REMOVE -----
131     /**
132      * Testing remove(); to empty.
133      */
134     @Test
135     public final void testRemoveToEmpty() {
136         // Setup
137         Set<String> s = this.createFromArgsTest("a");
138         Set<String> sExpected = this.createFromArgsRef();
139         // Call
140         s.remove("a");
141         // Eval
142         assertEquals(sExpected, s);
143     }
144
145     /**
146      * Testing remove(); to non-empty.
147      */
148     @Test
149     public final void testRemoveToNonEmpty() {
150         // Setup
151         Set<String> s = this.createFromArgsTest("a", "b", "c", "e");
152         Set<String> sExpected = this.createFromArgsRef("a", "b", "c");
153         // Call
154         s.remove("e");
155         // Eval
156         assertEquals(sExpected, s);
157     }
158
159     // REMOVEANY -----
160     /**
161      * Testing removeAny() with set.size() == 1.
162      */
163     @Test
164     public final void testRemoveAnySizeOne() {
165         // Setup
166         Set<String> s = this.createFromArgsTest("a");
167         Set<String> sExpected = this.createFromArgsRef("a");
168         String removedEntry;
169         // Call
170         removedEntry = s.removeAny();
171         // Eval
172         assertEquals(sExpected.contains(removedEntry), true);
173         sExpected.remove(removedEntry);
174         assertEquals(sExpected, s);
175     }
```

```
176
177  /**
178   * Testing removeAny() with set.size() == 3.
179   */
180  @Test
181  public final void testRemoveAnySizeThree() {
182      // Setup
183      Set<String> s = this.createFromArgsTest("a", "b", "c");
184      Set<String> sExpected = this.createFromArgsRef("a", "b", "c");
185      String removedEntry;
186      // Call
187      removedEntry = s.removeAny();
188      // Eval
189      assertEquals(sExpected.contains(removedEntry), true);
190      sExpected.remove(removedEntry);
191      assertEquals(sExpected, s);
192  }
193
194
195  // CONTAINS -----
196  /**
197   * Testing contains(); return true.
198   */
199  @Test
200  public final void testContainsTrue() {
201      // Setup
202      Set<String> s = this.createFromArgsTest("a", "b", "c", "e");
203      // Call
204      // Eval
205      assertEquals(true, s.contains("a"));
206  }
207
208  /**
209   * Testing contains(); return false.
210   */
211  @Test
212  public final void testContainsFalse() {
213      // Setup
214      Set<String> s = this.createFromArgsTest("a", "b", "c", "e");
215      // Call
216      // Eval
217      assertEquals(false, s.contains("d"));
218  }
219  // SIZE -----
220
221  /**
222   * Testing size(); set = {}.
223   */
224  @Test
225  public final void testSize0() {
226      // Setup
227      Set<String> s = this.createFromArgsTest();
228      // Call
229      // Eval
230      assertEquals(0, s.size());
231  }
232
```

```
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
```