

```
1 import static org.junit.Assert.assertEquals;
2
3 import org.junit.Test;
4
5 import components.set.Set;
6 import components.simplewriter.SimpleWriter;
7 import components.simplewriter.SimpleWriter1L;
8
9 /**
10  * JUnit test fixture for {@code Set<String>}'s constructor and
11  * kernel methods.
12  * @author Bashir Ali and Kwasi Fosu
13  *
14  */
15 public abstract class SetTest {
16
17     /**
18      * Invokes the appropriate {@code Set} constructor for the
19      * implementation
20      * under test and returns the result.
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
28      * reference
29      * implementation and returns the result.
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      * Test cases for constructors
77      */
78
79     @Test
80     public final void testNoArgumentConstructor() {
81         /*
82          * Set up variables and call method under test
83          */
84         Set<String> s = this.constructorTest();
85         Set<String> sExpected = this.constructorRef();
86         /*
87          * Assert that values of variables match expectations
88          */
89         assertEquals(sExpected, s);
90     }
91
92     /**
93      * Test cases for kernel methods
94      */
95
96     @Test
97     public final void testAddEmpty() {
98         /*
99          * Set up variables
100         */
101         Set<String> s = this.createFromArgsTest();
102         Set<String> sExpected = this.createFromArgsRef("red");
103         /*
104          * Call method under test
105          */
106         s.add("red");
107         /*
108          * Assert that values of variables match expectations
109          */
110         assertEquals(sExpected, s);
```

```
111     }
112
113     @Test
114     public final void testAddNonEmptyOne() {
115         /*
116          * Set up variables
117          */
118         Set<String> s = this.createFromArgsTest("red");
119         Set<String> sExpected = this.createFromArgsRef("red",
120 "blue");
121         /*
122          * Call method under test
123          */
124         s.add("blue");
125         /*
126          * Assert that values of variables match expectations
127          */
128         assertEquals(sExpected, s);
129     }
130
131     @Test
132     public final void testAddNonEmptyMoreThanOne() {
133         /*
134          * Set up variables
135          */
136         Set<String> s = this.createFromArgsTest("red", "blue",
137 "green");
138         Set<String> sExpected = this.createFromArgsRef("red",
139 "blue", "green",
140 "yellow");
141         /*
142          * Call method under test
143          */
144         s.add("yellow");
145         /*
146          * Assert that values of variables match expectations
147          */
148         assertEquals(sExpected, s);
149     }
```

```
147
148  @Test
149  public final void testRemoveLeavingEmpty() {
150      /*
151       * Set up variables
152       */
153      Set<String> s = this.createFromArgsTest("red");
154      Set<String> sExpected = this.createFromArgsRef();
155      /*
156       * Call method under test
157       */
158      String x = s.remove("red");
159      /*
160       * Assert that values of variables match expectations
161       */
162      assertEquals(sExpected, s);
163      assertEquals("red", x);
164  }
165
166  @Test
167  public final void testRemoveLeavingOne() {
168      /*
169       * Set up variables
170       */
171      Set<String> s = this.createFromArgsTest("red", "blue");
172      Set<String> sExpected = this.createFromArgsRef("red");
173      /*
174       * Call method under test
175       */
176      String x = s.remove("blue");
177      /*
178       * Assert that values of variables match expectations
179       */
180      assertEquals(sExpected, s);
181      assertEquals("blue", x);
182  }
183
184  @Test
185  public final void testRemoveLeavingMoreThanOne() {
```

```
186      /*
187       * Set up variables
188       */
189      Set<String> s = this.createFromArgsTest("red", "green",
"blue");
190      Set<String> sExpected = this.createFromArgsRef("red",
"blue");
191      SimpleWriter out = new SimpleWriter1L();
192      /*
193       * Call method under test
194       */
195      String x = s.remove("green");
196      out.print(x);
197      /*
198       * Assert that values of variables match expectations
199       */
200      assertEquals(sExpected, s);
201      assertEquals("green", x);
202  }
203
204  @Test
205  public final void testSizeEmpty() {
206      /*
207       * Set up variables
208       */
209      Set<String> s = this.createFromArgsTest();
210      Set<String> sExpected = this.createFromArgsRef();
211      /*
212       * Call method under test
213       */
214      int i = s.size();
215      /*
216       * Assert that values of variables match expectations
217       */
218      assertEquals(sExpected, s);
219      assertEquals(0, i);
220  }
221
222  @Test
```

```
223     public final void testSizeOne() {
224         /*
225          * Set up variables
226          */
227         Set<String> s = this.createFromArgsTest("red");
228         Set<String> sExpected = this.createFromArgsRef("red");
229         /*
230          * Call method under test
231          */
232         int i = s.size();
233         /*
234          * Assert that values of variables match expectations
235          */
236         assertEquals(sExpected, s);
237         assertEquals(1, i);
238     }
239
240     @Test
241     public final void testSizeMoreThanOne() {
242         /*
243          * Set up variables
244          */
245         Set<String> s = this.createFromArgsTest("red", "blue",
246 "green");
247         Set<String> sExpected = this.createFromArgsRef("red",
248 "blue", "green");
249         /*
250          * Call method under test
251          */
252         int i = s.size();
253         /*
254          * Assert that values of variables match expectations
255          */
256         assertEquals(sExpected, s);
257         assertEquals(3, i);
258     }
259
260     @Test
261     public final void testContainsEmpty() {
```

```
260      /*
261      * Set up variables
262      */
263      Set<String> s = this.createFromArgsTest();
264      Set<String> sExpected = this.createFromArgsRef();
265      /*
266      * Call method under test
267      */
268      boolean contains = s.contains("red");
269      /*
270      * Assert that values of variables match expectations
271      */
272      assertEquals(sExpected, s);
273      assertEquals(false, contains);
274  }
275
276  @Test
277  public final void testTrueWhenContainsOne() {
278      /*
279      * Set up variables
280      */
281      Set<String> s = this.createFromArgsTest("red");
282      Set<String> sExpected = this.createFromArgsRef("red");
283      /*
284      * Call method under test
285      */
286      boolean containsTrue = s.contains("red");
287      /*
288      * Assert that values of variables match expectations
289      */
290      assertEquals(sExpected, s);
291      assertEquals(true, containsTrue);
292  }
293
294  @Test
295  public final void testFalseWhenContainsOne() {
296      /*
297      * Set up variables
298      */
```



```
299         Set<String> s = this.createFromArgsTest("red");
300         Set<String> sExpected = this.createFromArgsRef("red");
301         /*
302          * Call method under test
303          */
304         boolean containsFalse = s.contains("blue");
305         /*
306          * Assert that values of variables match expectations
307          */
308         assertEquals(sExpected, s);
309         assertEquals(false, containsFalse);
310     }
311
312     @Test
313     public final void testTrueWhenContainsMany() {
314         /*
315          * Set up variables
316          */
317         Set<String> s = this.createFromArgsTest("red", "green",
"yellow");
318         Set<String> sExpected = this.createFromArgsRef("red",
"green",
319             "yellow");
320         /*
321          * Call method under test
322          */
323         boolean containsTrue = s.contains("red");
324         /*
325          * Assert that values of variables match expectations
326          */
327         assertEquals(sExpected, s);
328         assertEquals(true, containsTrue);
329     }
330
331     @Test
332     public final void testFalseWhenContainsMany() {
333         /*
334          * Set up variables
335          */
```

```
336     Set<String> s = this.createFromArgsTest("red", "green",
337     "yellow");
337     Set<String> sExpected = this.createFromArgsRef("red",
338     "green",
339     "yellow");
339     /*
340     * Call method under test
341     */
342     boolean containsFalse = s.contains("blue");
343     /*
344     * Assert that values of variables match expectations
345     */
346     assertEquals(sExpected, s);
347     assertEquals(false, containsFalse);
348 }
349
350 @Test
351 public final void testRemoveAnyLeavingEmpty() {
352     /*
353     * Set up variables
354     */
355     Set<String> s = this.createFromArgsTest("red");
356     Set<String> sExpected = this.createFromArgsRef();
357     /*
358     * Call method under test
359     */
360     String removed = s.removeAny();
361     /*
362     * Assert that values of variables match expectations
363     */
364     assertEquals(sExpected, s);
365     assertEquals("red", removed);
366 }
367
368 @Test
369 public final void testRemoveAnyLeavingOne() {
370     /*
371     * Set up variables
372     */
```

```
373         Set<String> s = this.createFromArgsTest("red", "blue");
374
375         /*
376          * Call method under test
377          */
378         String removed = s.removeAny();
379         int size = s.size();
380         int expectedSize = 1;
381
382         /*
383          * Assert that values of variables match expectations
384          */
385         assertEquals(expectedSize, size);
386     }
387
388     @Test
389     public final void testRemoveAnyLeavingMoreThanOne() {
390         /*
391          * Set up variables
392          */
393         Set<String> s = this.createFromArgsTest("red", "blue",
394 "green");
395
396         /*
397          * Call method under test
398          */
399         String removed = s.removeAny();
400         int size = s.size();
401         int expectedSize = 2;
402
403         /*
404          * Assert that values of variables match expectations
405          */
406         assertEquals(expectedSize, size);
407     }
408 }
409
```