

```
1 import static org.junit.Assert.assertEquals;
6
7 /**
8  * JUnit test fixture for {@code Set<String>}'s constructor and
   kernel methods.
9  *
10 * @author Shyam Sai Bethina and Yihone Chu
11 *
12 */
13 public abstract class SetTest {
14
15     /**
16      * Invokes the appropriate {@code Set} constructor for the
   implementation
17      * under test and returns the result.
18      *
19      * @return the new set
20      * @ensures constructorTest = {}
21      */
22     protected abstract Set<String> constructorTest();
23
24     /**
25      * Invokes the appropriate {@code Set} constructor for the
   reference
26      * implementation and returns the result.
27      *
28      * @return the new set
29      * @ensures constructorRef = {}
30      */
31     protected abstract Set<String> constructorRef();
32
33     /**
34      * Creates and returns a {@code Set<String>} of the
   implementation under
35      * test type with the given entries.
36      *
37      * @param args
38      *         the entries for the set
39      * @return the constructed set
40      * @requires [every entry in args is unique]
41      * @ensures createFromArgsTest = [entries in args]
42      */
43     private Set<String> createFromArgsTest(String... args) {
44         Set<String> set = this.constructorTest();
```

```
45         for (String s : args) {
46             assert !set.contains(
47                 s) : "Violation of: every entry in args is
unique";
48             set.add(s);
49         }
50         return set;
51     }
52
53     /**
54      * Creates and returns a {@code Set<String>} of the reference
implementation
55      * type with the given entries.
56      *
57      * @param args
58      *      the entries for the set
59      * @return the constructed set
60      * @requires [every entry in args is unique]
61      * @ensures createFromArgsRef = [entries in args]
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     /**
74      * Test the no argument constructor.
75      */
76     @Test
77     public void testConstructor() {
78         Set<String> test = this.constructorTest();
79         Set<String> expected = this.constructorRef();
80
81         assertEquals(expected, test);
82     }
83
84     /**
85      * Test add using an edge case.
```

```
86     */
87     @Test
88     public void testAdd() {
89         Set<String> test = this.createFromArgsTest();
90         test.add("hello");
91
92         Set<String> expected = this.createFromArgsRef();
93         expected.add("hello");
94
95         assertEquals(expected, test);
96     }
97
98     /**
99     * Test add using an routine case.
100    */
101    @Test
102    public void testAdd2() {
103        Set<String> test = this.createFromArgsTest();
104        test.add("hello");
105        test.add("there");
106
107        Set<String> expected = this.createFromArgsRef();
108        expected.add("hello");
109        expected.add("there");
110
111        assertEquals(expected, test);
112    }
113
114    /**
115    * Test add using an routine case.
116    */
117    @Test
118    public void testAdd3() {
119        Set<String> test = this.createFromArgsTest();
120        test.add("hello");
121        test.add("there");
122        test.add("my");
123
124        Set<String> expected = this.createFromArgsRef();
125        expected.add("hello");
126        expected.add("there");
127        expected.add("my");
128
129        assertEquals(expected, test);
```

```
130     }
131
132     /**
133      * Test add using an challenging case.
134      */
135     @Test
136     public void testAdd4() {
137         Set<String> test = this.createFromArgsTest();
138         test.add("");
139
140         Set<String> expected = this.createFromArgsRef();
141         expected.add("");
142
143         assertEquals(expected, test);
144     }
145
146     /**
147      * Test remove using an edge case.
148      */
149     @Test
150     public void testRemove1() {
151         Set<String> test = this.createFromArgsTest("hello");
152         String testRemoved = test.remove("hello");
153
154         Set<String> expected = this.createFromArgsRef("hello");
155         String expectedRemoved = test.remove("hello");
156
157         assertEquals(expected, test);
158         assertEquals(expectedRemoved, testRemoved);
159     }
160
161     /**
162      * Test remove using an routine case.
163      */
164     @Test
165     public void testRemove2() {
166         Set<String> test = this.createFromArgsTest("hello",
167 "there");
168         String testRemoved = test.remove("there");
169
170         Set<String> expected = this.createFromArgsRef("hello",
171 "there");
172         String expectedRemoved = test.remove("there");
```

```
172         assertEquals(expected, test);
173         assertEquals(expectedRemoved, testRemoved);
174     }
175
176     /**
177      * Test remove using an routine case.
178      */
179     @Test
180     public void testRemove3() {
181         Set<String> test = this.createFromArgsTest("hello",
182 "there", "general");
183         String testRemoved = test.remove("general");
184
185         Set<String> expected = this.createFromArgsRef("hello",
186 "there",
187         "general");
188         String expectedRemoved = test.remove("general");
189
190         assertEquals(expected, test);
191         assertEquals(expectedRemoved, testRemoved);
192     }
193
194     /**
195      * Test remove using an challenging case.
196      */
197     @Test
198     public void testRemove4() {
199         Set<String> test = this.createFromArgsTest("");
200         String testRemoved = test.remove("");
201
202         Set<String> expected = this.createFromArgsRef("");
203         String expectedRemoved = test.remove("");
204
205         assertEquals(expected, test);
206         assertEquals(expectedRemoved, testRemoved);
207     }
208
209     /**
210      * Test contains using an edge case
211      */
212     @Test
213     public void testContains1() {
214         Set<String> test = this.createFromArgsTest("hello");
215         Set<String> expected = this.createFromArgsRef("hello");
```

```
214
215     boolean testBoolean = test.contains("hello");
216
217     boolean expectedBoolean = expected.contains("hello");
218
219     assertEquals(expected, test);
220     assertEquals(expectedBoolean, testBoolean);
221
222 }
223
224 /**
225  * Test contains using an routine case
226  */
227 @Test
228 public void testContains2() {
229     Set<String> test = this.createFromArgsTest("hello",
230 "there");
231     Set<String> expected = this.createFromArgsRef("hello",
232 "there");
233
234     boolean testBoolean = test.contains("there");
235
236     boolean expectedBoolean = expected.contains("there");
237
238     assertEquals(expected, test);
239     assertEquals(expectedBoolean, testBoolean);
240
241 }
242
243 /**
244  * Test contains using an routine case
245  */
246 @Test
247 public void testContains3() {
248     Set<String> test = this.createFromArgsTest("hello",
249 "there", "general");
250     Set<String> expected = this.createFromArgsRef("hello",
251 "there",
252 "general");
253
254     boolean testBoolean = test.contains("kenobi");
255
256     boolean expectedBoolean = expected.contains("kenobi");
257
258 }
```

```
254         assertEquals(expected, test);
255         assertEquals(expectedBoolean, testBoolean);
256
257     }
258
259     /**
260      * Test contains using an challenging case
261      */
262     @Test
263     public void testContains4() {
264         Set<String> test = this.createFromArgsTest();
265         Set<String> expected = this.createFromArgsRef();
266
267         boolean testBoolean = test.contains("");
268
269         boolean expectedBoolean = expected.contains("");
270
271         assertEquals(expected, test);
272         assertEquals(expectedBoolean, testBoolean);
273
274     }
275
276     /**
277      * Test size using an edge case
278      */
279     @Test
280     public void testSize1() {
281         Set<String> test = this.createFromArgsTest();
282         Set<String> expected = this.createFromArgsRef();
283
284         int testReturn = test.size();
285         int expectedReturn = expected.size();
286
287         assertEquals(expected, test);
288         assertEquals(expectedReturn, testReturn);
289
290     }
291
292     /**
293      * Test size using an routine case
294      */
295     @Test
296     public void testSize2() {
297         Set<String> test = this.createFromArgsTest("hello",
```

```
    "there");
298     Set<String> expected = this.createFromArgsRef("hello",
    "there");
299
300     int testReturn = test.size();
301     int expectedReturn = expected.size();
302
303     assertEquals(expected, test);
304     assertEquals(expectedReturn, testReturn);
305
306 }
307
308 /**
309  * Test size using an routine case
310  */
311 @Test
312 public void testSize3() {
313     Set<String> test = this.createFromArgsTest("hello",
    "there", "general");
314     Set<String> expected = this.createFromArgsRef("hello",
    "there",
315         "general");
316
317     int testReturn = test.size();
318     int expectedReturn = expected.size();
319
320     assertEquals(expected, test);
321     assertEquals(expectedReturn, testReturn);
322
323 }
324
325 /**
326  * Test size using an challenging case
327  */
328 @Test
329 public void testSize4() {
330     Set<String> test = this.createFromArgsTest("");
331     Set<String> expected = this.createFromArgsRef("");
332
333     int testReturn = test.size();
334     int expectedReturn = expected.size();
335
336     assertEquals(expected, test);
337     assertEquals(expectedReturn, testReturn);
}
```



```
338
339     }
340
341     /**
342      * Test removeAny using an edge case
343      */
344     @Test
345     public void testRemoveAny1() {
346
347         /*
348          * Set up variables and call method under test
349          */
350         //Setup
351         Set<String> test = this.createFromArgsTest("hello");
352         Set<String> expected = this.createFromArgsRef("hello");
353
354         //Call
355         String capture = test.removeAny();
356
357         //Evaluation
358         assertEquals(true, expected.contains(capture));
359         expected.remove(capture);
360         assertEquals(expected, test);
361     }
362
363
364     /**
365      * Test removeAny using an routine case
366      */
367     @Test
368     public void testRemoveAny2() {
369
370         /*
371          * Set up variables and call method under test
372          */
373         //Setup
374         Set<String> test = this.createFromArgsTest("hello",
375 "there");
376         Set<String> expected = this.createFromArgsRef("hello",
377 "there");
378
379         //Call
380         String capture = test.removeAny();
```

```
380         //Evaluation
381         assertEquals(true, expected.contains(capture));
382         expected.remove(capture);
383         assertEquals(expected, test);
384     }
385 }
386
387 /**
388  * Test removeAny using an routine case
389  */
390 @Test
391 public void testRemoveAny3() {
392     /*
393     * Set up variables and call method under test
394     */
395     //Setup
396     Set<String> test = this.createFromArgsTest("hello",
397 "there", "general");
398     Set<String> expected = this.createFromArgsRef("hello",
399 "there",
400         "general");
401     //Call
402     String capture = test.removeAny();
403
404     //Evaluation
405     assertEquals(true, expected.contains(capture));
406     expected.remove(capture);
407     assertEquals(expected, test);
408 }
409 }
410
411 /**
412  * Test removeAny using an challenging case
413  */
414 @Test
415 public void testRemoveAny4() {
416     /*
417     * Set up variables and call method under test
418     */
419 }
```

```
422         //Setup
423         Set<String> test = this.createFromArgsTest("");
424         Set<String> expected = this.createFromArgsRef("");
425
426         //Call
427         String capture = test.removeAny();
428
429         //Evaluation
430         assertEquals(true, expected.contains(capture));
431         expected.remove(capture);
432         assertEquals(expected, test);
433
434     }
435
436     // TODO - add test cases for constructor, add, remove,
437     removeAny, contains, and size
438 }
439
```