

```
1 import static org.junit.Assert.assertEquals;
2
3 /**
4  * JUnit test fixture for {@code Map<String, String>}s constructor and
5  * kernel
6  * methods.
7  *
8  * @author Put your name here
9  */
10 public abstract class MapTest {
11     /**
12      * Invokes the appropriate {@code Map} constructor for the
13      * implementation
14      * under test and returns the result.
15      *
16      * @return the new map
17      * @ensures constructorTest = {}
18      */
19     protected abstract Map<String, String> constructorTest();
20
21     /**
22      * Invokes the appropriate {@code Map} constructor for the
23      * reference
24      * implementation and returns the result.
25      *
26      * @return the new map
27      * @ensures constructorRef = {}
28      */
29     protected abstract Map<String, String> constructorRef();
30
31     /**
32      *
33      * Creates and returns a {@code Map<String, String>} of the
34      * implementation
35      * under test type with the given entries.
36      *
37      * @param args
38      *         the (key, value) pairs for the map
39      * @return the constructed map
40      * @requires <pre>
41      * [args.length is even] and
42      * [the 'key' entries in args are unique]
43      * </pre>
44      * @ensures createFromArgsTest = [pairs in args]
45      */
46 }
```

```

48     private Map<String, String> createFromArgsTest(String... args) {
49         assert args.length % 2 == 0 : "Violation of: args.length is
even";
50         Map<String, String> map = this.constructorTest();
51         for (int i = 0; i < args.length; i += 2) {
52             assert !map.containsKey(args[i]) : ""
53                 + "Violation of: the 'key' entries in args are
unique";
54             map.add(args[i], args[i + 1]);
55         }
56         return map;
57     }
58
59     /**
60     *
61     * Creates and returns a {@code Map<String, String>} of the
reference
62     * implementation type with the given entries.
63     *
64     * @param args
65     *     the (key, value) pairs for the map
66     * @return the constructed map
67     * @requires <pre>
68     * [args.length is even] and
69     * [the 'key' entries in args are unique]
70     * </pre>
71     * @ensures createFromArgsRef = [pairs in args]
72     */
73     private Map<String, String> createFromArgsRef(String... args) {
74         assert args.length % 2 == 0 : "Violation of: args.length is
even";
75         Map<String, String> map = this.constructorRef();
76         for (int i = 0; i < args.length; i += 2) {
77             assert !map.containsKey(args[i]) : ""
78                 + "Violation of: the 'key' entries in args are
unique";
79             map.add(args[i], args[i + 1]);
80         }
81         return map;
82     }
83
84     /*
85     * Test cases for constructors
86     */
87
88     @Test
89     public final void testNoArgumentConstructor() {

```

```
90      /*
91       * Set up variables and call method under test
92       */
93      Map<String, String> m = this.constructorTest();
94      Map<String, String> mExpected = this.constructorRef();
95      /*
96       * Assert that values of variables match expectations
97       */
98      assertEquals(mExpected, m);
99  }
100
101  /*
102   * Test cases for kernel methods
103   */
104
105  @Test
106  public final void testAddEmpty() {
107      /*
108       * Set up variables
109       */
110      Map<String, String> m = this.createFromArgsTest();
111      Map<String, String> mExpected = this.createFromArgsRef("1",
112  "red");
113      /*
114       * Call method under test
115       */
116      m.add("1", "red");
117      /*
118       * Assert that values of variables match expectations
119       */
120      assertEquals(mExpected, m);
121  }
122
123  @Test
124  public final void testAddOne() {
125      /*
126       * Set up variables
127       */
128      Map<String, String> m = this.createFromArgsTest("1", "red");
129      Map<String, String> mExpected = this.createFromArgsRef("1",
130  "red", "2",
131  "green");
132      /*
133       * Call method under test
134       */
135      m.add("2", "green");
136      /*
```

```
135         * Assert that values of variables match expectations
136         */
137         assertEquals(mExpected, m);
138     }
139
140     @Test
141     public final void testAddMoreThanOne() {
142         /*
143         * Set up variables
144         */
145         Map<String, String> m = this.createFromArgsTest("1", "red",
146             "2",
147             "green");
148         Map<String, String> mExpected = this.createFromArgsRef("1",
149             "red", "2",
150             "green", "3", "blue");
151         /*
152         * Call method under test
153         */
154         m.add("3", "blue");
155         /*
156         * Assert that values of variables match expectations
157         */
158         assertEquals(mExpected, m);
159     }
160
161     @Test
162     public final void testRemoveLeavingZero() {
163         /*
164         * Set up variables
165         */
166         Map<String, String> m = this.createFromArgsTest("1", "red");
167         Map<String, String> mExpected = this.createFromArgsRef();
168         /*
169         * Call method under test
170         */
171         m.remove("1");
172         /*
173         * Assert that values of variables match expectations
174         */
175         assertEquals(mExpected, m);
176     }
177
178     @Test
179     public final void testRemoveLeavingOne() {
180         /*
181         * Set up variables
```

```
180         */
181         Map<String, String> m = this.createFromArgsTest("1", "red",
182 "2",
183         "green");
184         Map<String, String> mExpected = this.createFromArgsRef("1",
185 "red");
186         /*
187         * Call method under test
188         */
189         m.remove("2");
190         /*
191         * Assert that values of variables match expectations
192         */
193         assertEquals(mExpected, m);
194     }
195
196     @Test
197     public final void testRemoveLeavingMoreThanOne() {
198         /*
199         * Set up variables
200         */
201         Map<String, String> m = this.createFromArgsTest("1", "red",
202 "2",
203         "green", "3", "blue");
204         Map<String, String> mExpected = this.createFromArgsRef("1",
205 "red", "3",
206         "blue");
207         /*
208         * Call method under test
209         */
210         m.remove("2");
211         /*
212         * Assert that values of variables match expectations
213         */
214         assertEquals(mExpected, m);
215     }
216
217     @Test
218     public final void testRemoveAnyLeavingEmpty() {
219         /*
220         * Set up variables
221         */
222         Map<String, String> m = this.createFromArgsTest("1", "red");
223         Map<String, String> m2 = this.createFromArgsTest("1", "red");
224         /*
225         * Call method under test
226         */
```

```
223         int expected = 0;
224         Map.Pair<String, String> pair = m.removeAny();
225         /*
226          * Assert that values of variables match expectations
227          */
228         assertEquals(expected, m.size());
229         assertEquals(m2.containsKey(pair.key()), true);
230
231     }
232
233     @Test
234     public final void testRemoveAnyLeavingNotEmpty() {
235         /*
236          * Set up variables
237          */
238         Map<String, String> m = this.createFromArgsTest("1", "red",
239 "2",
240 "2",
241 "blue");
242         Map<String, String> m2 = this.createFromArgsTest("1", "red",
243 "2",
244 "blue");
245         /*
246          * Call method under test
247          */
248         int expected = 1;
249         Map.Pair<String, String> pair = m.removeAny();
250         int mSize = m.size();
251         /*
252          * Assert that values of variables match expectations
253          */
254         assertEquals(expected, mSize);
255         assertEquals(m2.containsKey(pair.key()), true);
256     }
257
258     @Test
259     public final void testSizeEmpty() {
260         /*
261          * Set up variables
262          */
263         Map<String, String> m = this.createFromArgsTest();
264         Map<String, String> mExpected = this.createFromArgsRef();
265         Map<String, String> m2Expected = this.createFromArgsRef();
266         /*
267          * Call method under test
268          */
269         int mSize = m.size();
270         int mExpectedSize = mExpected.size();
```

```
268      /*
269       * Assert that values of variables match expectations
270       */
271      assertEquals(mExpectedSize, mSize);
272      assertEquals(m2Expected, m);
273  }
274
275  @Test
276  public final void testSizeOne() {
277      /*
278       * Set up variables
279       */
280      Map<String, String> m = this.createFromArgsTest("1", "red");
281      Map<String, String> mExpected = this.createFromArgsRef("1",
282  "red");
283      Map<String, String> m2Expected = this.createFromArgsRef("1",
284  "red");
285      /*
286       * Call method under test
287       */
288      int mSize = m.size();
289      int mExpectedSize = mExpected.size();
290      /*
291       * Assert that values of variables match expectations
292       */
293      assertEquals(mExpectedSize, mSize);
294      assertEquals(m2Expected, m);
295  }
296
297  @Test
298  public final void testSizeMoreThanOne() {
299      /*
300       * Set up variables
301       */
302      Map<String, String> m = this.createFromArgsTest("1", "red",
303  "2",
304  "green");
305      Map<String, String> m2Expected = this.createFromArgsTest("1",
306  "red",
307  "2", "green");
308      /*
309       * Call method under test
310       */
311      int mSize = m.size();
312      int mExpectedSize = 2;
313      /*
314       * Assert that values of variables match expectations
```

```
311         */
312         assertEquals(mExpectedSize, mSize);
313         assertEquals(m2Expected, m);
314     }
315
316     @Test
317     public final void testValueOne() {
318         /*
319          * Set up variables
320          */
321         Map<String, String> m = this.createFromArgsTest("1", "red");
322         /*
323          * Call method under test
324          */
325         String mKey = m.value("1");
326         String mExpected = "red";
327         Map<String, String> m2Expected = this.createFromArgsTest("1",
"red");
328         /*
329          * Assert that values of variables match expectations
330          */
331         assertEquals(mExpected, mKey);
332         assertEquals(m2Expected, m);
333     }
334
335     @Test
336     public final void testValueMoreThanOne() {
337         /*
338          * Set up variables
339          */
340         Map<String, String> m = this.createFromArgsTest("1", "red",
"2",
341             "green");
342         /*
343          * Call method under test
344          */
345         String mKey = m.value("1");
346         String mExpected = "red";
347         Map<String, String> m2Expected = this.createFromArgsTest("1",
"red",
348             "2", "green");
349         /*
350          * Assert that values of variables match expectations
351          */
352         assertEquals(mExpected, mKey);
353         assertEquals(m2Expected, m);
354     }
```



```
355
356     @Test
357     public final void testHasKeyTrue() {
358         /*
359          * Set up variables
360          */
361         Map<String, String> m = this.createFromArgsTest("1", "red");
362         Map<String, String> m2Expected = this.createFromArgsTest("1",
"red");
363         /*
364          * Call method under test
365          */
366         boolean mKey = m.containsKey("1");
367         boolean mExpected = true;
368         /*
369          * Assert that values of variables match expectations
370          */
371         assertEquals(mExpected, mKey);
372         assertEquals(m2Expected, m);
373     }
374
375     @Test
376     public final void testHasKeyFalse() {
377         /*
378          * Set up variables
379          */
380         Map<String, String> m = this.createFromArgsTest("1", "red");
381         Map<String, String> m2Expected = this.createFromArgsTest("1",
"red");
382         /*
383          * Call method under test
384          */
385         boolean mKey = m.containsKey("2");
386         boolean mExpected = false;
387         /*
388          * Assert that values of variables match expectations
389          */
390         assertEquals(mExpected, mKey);
391         assertEquals(m2Expected, m);
392     }
393
394     @Test
395     public final void testHasKeyFalseOnEmpty() {
396         /*
397          * Set up variables
398          */
399         Map<String, String> m = this.createFromArgsTest();
```

```
400      Map<String, String> m2Expected = this.createFromArgsTest();
401      /*
402       * Call method under test
403       */
404      boolean mKey = m.containsKey("2");
405      boolean mExpected = false;
406      /*
407       * Assert that values of variables match expectations
408       */
409      assertEquals(mExpected, mKey);
410      assertEquals(m2Expected, m);
411    }
412 }
413
```