

```
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.map.Map;
8
9 /**
10  * JUnit test fixture for {@code Map<String, String>}s constructor and kernel
11  * methods.
12  *
13  * @author David P. & Zach B.
14  */
15
16 public abstract class MapTest {
17
18     /**
19      * Invokes the appropriate {@code Map} constructor for the implementation
20      * under test and returns the result.
21      *
22      * @return the new map
23      * @ensures constructorTest = {}
24      */
25     protected abstract Map<String, String> constructorTest();
26
27     /**
28      * Invokes the appropriate {@code Map} constructor for the reference
29      * implementation and returns the result.
30      *
31      * @return the new map
32      * @ensures constructorRef = {}
33      */
34     protected abstract Map<String, String> constructorRef();
35
36     /**
37      *
38      * Creates and returns a {@code Map<String, String>} of the implementation
39      * under test type with the given entries.
40      *
41      * @param args
42      *         the (key, value) pairs for the map
43      * @return the constructed map
44      * @requires <pre>
45      * [args.length is even] and
46      * [the 'key' entries in args are unique]
47      * </pre>
48      * @ensures createFromArgsTest = [pairs in args]
49      */
50     private Map<String, String> createFromArgsTest(String... args) {
51         assert args.length % 2 == 0 : "Violation of: args.length is even";
52         Map<String, String> map = this.constructorTest();
53         for (int i = 0; i < args.length; i += 2) {
54             assert !map.containsKey(args[i]) : ""
55                 + "Violation of: the 'key' entries in args are unique";
56             map.add(args[i], args[i + 1]);
57         }
58     }
59 }
```

```

58         return map;
59     }
60
61     /**
62     *
63     * Creates and returns a {@code Map<String, String>} of the reference
64     * implementation type with the given entries.
65     *
66     * @param args
67     *     the (key, value) pairs for the map
68     * @return the constructed map
69     * @requires <pre>
70     *     [args.length is even] and
71     *     [the 'key' entries in args are unique]
72     * </pre>
73     * @ensures createFromArgsRef = [pairs in args]
74     */
75     private Map<String, String> createFromArgsRef(String... args) {
76         assert args.length % 2 == 0 : "Violation of: args.length is even";
77         Map<String, String> map = this.constructorRef();
78         for (int i = 0; i < args.length; i += 2) {
79             assert !map.containsKey(args[i]) : ""
80                 + "Violation of: the 'key' entries in args are unique";
81             map.add(args[i], args[i + 1]);
82         }
83         return map;
84     }
85
86     // TODO - add test cases for constructor, add, remove, removeAny, value,
87     // hasKey, and size
88
89     /**
90     * Tests adding to an initially empty map.
91     */
92     @Test
93     public void testAddToEmptyMap() {
94         Map<String, String> map = this.createFromArgsTest();
95         Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A");
96         map.add("Alice", "A");
97         assertEquals(map, expectedMap);
98     }
99
100    /**
101    * Tests adding a new pair to a non-empty map.
102    */
103    @Test
104    public void testAddPairToNonEmptyMap() {
105        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
106            "B");
107        Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A",
108            "Bob", "B", "Charlie", "C");
109        map.add("Charlie", "C");
110        assertEquals(map, expectedMap);
111    }
112
113    /**
114    * Tests adding multiple pairs to a map.

```

```
115     */
116     @Test
117     public void testAddMultiplePairs() {
118         Map<String, String> map = this.createFromArgsTest("Charlie", "C");
119         Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A",
120             "Charlie", "C", "Bob", "B");
121         map.add("Bob", "B");
122         map.add("Alice", "A");
123         assertEquals(map, expectedMap);
124     }
125
126     /**
127     * Tests removing the only pair in the map.
128     */
129     @Test
130     public void testRemovePairLeavingEmptyMap() {
131         Map<String, String> map = this.createFromArgsTest("David", "D");
132         Map<String, String> expectedMap = this.createFromArgsRef();
133         map.remove("David");
134         assertEquals(map, expectedMap);
135     }
136
137     /**
138     * Tests removing a pair from a non-empty map.
139     */
140     @Test
141     public void testRemovePairFromNonEmptyMap() {
142         Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
143             "B", "David", "D");
144         Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A",
145             "Bob", "B");
146         map.remove("David");
147         assertEquals(map, expectedMap);
148     }
149
150     /**
151     * Tests removing multiple pairs from a non-empty map.
152     */
153     @Test
154     public void testRemoveMultiplePairs() {
155         Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
156             "B", "David", "D");
157         Map<String, String> expectedMap = this.createFromArgsRef("Bob", "B");
158         map.remove("David");
159         map.remove("Alice");
160         assertEquals(map, expectedMap);
161     }
162
163     /**
164     * Tests removing any pair from a map with multiple pairs.
165     */
166     @Test
167     public void testRemoveAnyPairFromMultiple() {
168         Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
169             "B", "David", "D");
170         Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A",
171             "Bob", "B", "David", "D");
```

```
172     Map.Pair<String, String> removedPair = map.removeAny();
173     assertTrue(expectedMap.containsKey(removedPair.key()));
174     expectedMap.remove(removedPair.key());
175     assertEquals(map, expectedMap);
176 }
177
178 /**
179  * Tests removing any pair from a map with a single pair.
180  */
181 @Test
182 public void testRemoveAnyPairFromSingle() {
183     Map<String, String> map = this.createFromArgsTest("Alice", "A");
184     Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A");
185     Map.Pair<String, String> removedPair = map.removeAny();
186     assertTrue(expectedMap.containsKey(removedPair.key()));
187     expectedMap.remove(removedPair.key());
188     assertEquals(map, expectedMap);
189 }
190
191 /**
192  * Tests retrieving the value associated with a key in a single-pair map.
193  */
194 @Test
195 public void testGetValueSinglePair() {
196     Map<String, String> map = this.createFromArgsTest("Alice", "A");
197     String value = map.value("Alice");
198     String expectedValue = "A";
199     assertEquals(value, expectedValue);
200 }
201
202 /**
203  * Tests retrieving the value associated with a key in a multiple-pair map.
204  */
205 @Test
206 public void testGetValueMultiplePairs() {
207     Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
208         "B", "Charlie", "C");
209     String value = map.value("Bob");
210     String expectedValue = "B";
211     assertEquals(value, expectedValue);
212 }
213
214 /**
215  * Tests checking the presence of a key in a single-pair map.
216  */
217 @Test
218 public void testHasKeyInSinglePair() {
219     Map<String, String> map = this.createFromArgsTest("Alice", "A");
220     boolean hasKey = map.containsKey("Alice");
221     assertTrue(hasKey);
222 }
223
224 /**
225  * Tests checking the presence of keys in a multiple-pair map.
226  */
227 @Test
228 public void testHasKeyInMultiplePairs() {
```

```
229     Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
230         "B", "Charlie", "C");
231     assertTrue(map.containsKey("Alice"));
232     assertTrue(map.containsKey("Bob"));
233     assertTrue(map.containsKey("Charlie"));
234 }
235
236 /**
237  * Tests checking the absence of a key in a map.
238  */
239 @Test
240 public void testHasKeyNotFound() {
241     Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
242         "B", "Charlie", "C");
243     boolean hasKey = map.containsKey("David");
244     assertFalse(hasKey);
245 }
246
247 /**
248  * Tests checking the absence of a key in an empty map.
249  */
250 @Test
251 public void testHasKeyInEmptyMap() {
252     Map<String, String> map = this.createFromArgsTest();
253     boolean hasKey = map.containsKey("Alice");
254     assertFalse(hasKey);
255 }
256
257 /**
258  * Tests checking the absence of keys in a map with multiple pairs.
259  */
260 @Test
261 public void testHasKeyMultipleNotFound() {
262     Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
263         "B", "Charlie", "C");
264     assertFalse(map.containsKey("David"));
265     assertFalse(map.containsKey("Eve"));
266 }
267
268 /**
269  * Tests the size of an empty map.
270  */
271 @Test
272 public void testSizeOfEmptyMap() {
273     Map<String, String> map = this.createFromArgsTest();
274     int size = map.size();
275     int expectedSize = 0;
276     assertEquals(size, expectedSize);
277 }
278
279 /**
280  * Tests the size of a map with a single pair.
281  */
282 @Test
283 public void testSizeOfSinglePairMap() {
284     Map<String, String> map = this.createFromArgsTest("Alice", "A");
285     int size = map.size();
```

```
286         int expectedSize = 1;
287         assertEquals(size, expectedSize);
288     }
289
290     /**
291     * Tests the size of a map with multiple pairs.
292     */
293     @Test
294     public void testSizeOfMultiplePairsMap() {
295         Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
296             "B", "Charlie", "C");
297         int size = map.size();
298         int expectedSize = 3;
299         assertEquals(size, expectedSize);
300     }
301 }
302
```