

```

1 import static org.junit.Assert.assertEquals;
2
3 /**
4  * JUnit test fixture for {@code Map<String, String>}s constructor and kernel
5  * methods.
6  *
7  * @author Qinuo Shi & Yiming Cheng
8  */
9
10 public abstract class MapTest {
11
12     /**
13      * Invokes the appropriate {@code Map} constructor for the 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 reference
23      * implementation and returns the result.
24      *
25      * @return the new map
26      * @ensures constructorRef = {}
27      */
28     protected abstract Map<String, String> constructorRef();
29
30     /**
31      *
32      * Creates and returns a {@code Map<String, String>} of the implementation
33      * under test type with the given entries.
34      *
35      * @param args
36      *         the (key, value) pairs for the map
37      * @return the constructed map
38      * @requires <pre>
39      * [args.length is even] and
40      * [the 'key' entries in args are unique]
41      * </pre>
42      * @ensures createFromArgsTest = [pairs in args]
43      */
44     private Map<String, String> createFromArgsTest(String... args) {
45         assert args.length % 2 == 0 : "Violation of: args.length is even";
46         Map<String, String> map = this.constructorTest();
47         for (int i = 0; i < args.length; i += 2) {
48             assert !map.containsKey(args[i]) : ""
49                 + "Violation of: the 'key' entries in args are unique";
50             map.add(args[i], args[i + 1]);
51         }
52         return map;
53     }
54
55     /**
56      *
57      * 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     // TODO - add test cases for constructor, add, remove, removeAny, value,
85     // hasKey, and size
86     /**
87      * test for constructor without elements.
88      */
89     @Test
90     public void testDefaultConstructor() {
91         Map<String, String> t = this.constructorTest();
92         Map<String, String> texpected = this.constructorRef();
93
94         assertEquals(t, texpected);
95     }
96
97     /**
98      * test for constructor with elements.
99      */
100    @Test
101    public void testConstructorOne() {
102        Map<String, String> t = this.createFromArgsTest("q", "0");
103        Map<String, String> texpected = this.createFromArgsRef("q", "0");
104
105        assertEquals(t, texpected);
106    }
107
108    /**
109     * test for adding one pair of elements to constructor without elements.
110     */
111    @Test
112    public void testAddToNew() {
113        Map<String, String> t = this.createFromArgsTest();
114        Map<String, String> texpected = this.createFromArgsRef("q", "0");
115
116        t.add("q", "0");
117
118        assertEquals(t, texpected);

```

```
119     }
120
121     /**
122     * test for constructor with more elements.
123     */
124     @Test
125     public void testConstructorElements() {
126         Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
127             "2", "r", "3", "t", "4");
128         Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
129             "1", "e", "2", "r", "3", "t", "4");
130
131         assertEquals(t, texpected);
132     }
133
134     /**
135     * test for add one pair of elements to constructor with elements.
136     */
137     @Test
138     public void testAddOne() {
139         Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1");
140         Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
141             "1", "e", "2");
142
143         t.add("e", "2");
144
145         assertEquals(t, texpected);
146     }
147
148     /**
149     * test for add some pairs of elements to constructor without elements.
150     */
151     @Test
152     public void testAddMany() {
153         Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1");
154         Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
155             "1", "e", "2", "r", "3", "t", "4");
156
157         t.add("e", "2");
158         t.add("r", "3");
159         t.add("t", "4");
160
161         assertEquals(t, texpected);
162     }
163
164     /**
165     * test for remove elements from constructors with elements.
166     */
167     @Test
168     public void testRemoveZero() {
169         Map<String, String> t = this.createFromArgsTest("q", "0");
170         Map<String, String> texpected = this.createFromArgsRef();
171
172         Map.Pair<String, String> one = t.remove("q");
173
174         assertEquals(t, texpected);
175         assertEquals(one.key(), "q");
```

```
176         assertEquals(one.value(), "0");
177     }
178
179     /**
180      * test for remove elements from constructors with elements.
181      */
182     @Test
183     public void testRemove() {
184         Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
185             "2", "r", "3");
186         Map<String, String> texpected = this.createFromArgsRef("q", "0");
187
188         Map.Pair<String, String> two = t.remove("w");
189         Map.Pair<String, String> three = t.remove("e");
190         Map.Pair<String, String> four = t.remove("r");
191
192         assertEquals(t, texpected);
193         assertEquals(two.key(), "w");
194         assertEquals(two.value(), "1");
195         assertEquals(three.key(), "e");
196         assertEquals(three.value(), "2");
197         assertEquals(four.key(), "r");
198         assertEquals(four.value(), "3");
199     }
200
201     /**
202      * test for remove elements from constructors with elements.
203      */
204     @Test
205     public void testRemoveMany() {
206         Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
207             "2", "r", "3", "t", "4");
208         Map<String, String> texpected = this.createFromArgsRef("w", "1", "e",
209             "2", "r", "3", "t", "4");
210
211         Map.Pair<String, String> one = t.remove("q");
212
213         assertEquals(t, texpected);
214         assertEquals(one.key(), "q");
215         assertEquals(one.value(), "0");
216     }
217
218     /**
219      * test for remove any pair of elements.
220      */
221     @Test
222     public void testRemoveToZero() {
223         Map<String, String> t = this.createFromArgsTest("q", "0");
224         Map<String, String> texpected = this.createFromArgsRef("q", "0");
225
226         Map.Pair<String, String> one = t.removeAny();
227
228         Map.Pair<String, String> oneexpected = texpected.remove(one.key());
229         assertEquals(t, texpected);
230         assertEquals(one.key(), oneexpected.key());
231         assertEquals(one.value(), oneexpected.value());
232     }
```

```
233
234 /**
235  * test for remove any pair of elements.
236  */
237 @Test
238 public void testRemoveAny() {
239     Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
240         "2", "r", "3", "t", "4");
241     Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
242         "1", "e", "2", "r", "3", "t", "4");
243
244     Map.Pair<String, String> one = t.removeAny();
245
246     Map.Pair<String, String> oneexpected = texpected.remove(one.key());
247     assertEquals(t, texpected);
248     assertEquals(one.key(), oneexpected.key());
249     assertEquals(one.value(), oneexpected.value());
250 }
251
252 /**
253  * test for one value in the constructor.
254  */
255 @Test
256 public void testValue0() {
257     Map<String, String> t = this.createFromArgsTest("q", "0");
258     Map<String, String> texpected = this.createFromArgsRef("q", "0");
259
260     String string = t.value("q");
261     String stringexpected = texpected.value("q");
262
263     assertEquals(t, texpected);
264     assertEquals(string, "0");
265     assertEquals(stringexpected, "0");
266 }
267
268 /**
269  * test for one value in the constructor.
270  */
271 @Test
272 public void testValueM() {
273     Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
274         "2", "r", "3", "t", "4");
275     Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
276         "1", "e", "2", "r", "3", "t", "4");
277
278     String string = t.value("q");
279     String stringexpected = texpected.value("q");
280
281     assertEquals(t, texpected);
282     assertEquals(string, "0");
283     assertEquals(stringexpected, "0");
284 }
285
286 /**
287  * test for some values in the constructor.
288  */
289 @Test
```

```
290     public void testValueS() {
291         Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
292             "2");
293         Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
294             "1", "e", "2");
295
296         String string = t.value("q");
297         String stringexpected = texpected.value("q");
298         String string2 = t.value("w");
299         String stringexpected2 = texpected.value("w");
300
301         assertEquals(t, texpected);
302         assertEquals(string, "0");
303         assertEquals(stringexpected, "0");
304         assertEquals(string2, "1");
305         assertEquals(stringexpected2, "1");
306     }
307
308     /**
309      * test for hasKey with no element.
310      */
311     @Test
312     public void testHasKeyZero() {
313         Map<String, String> t = this.constructorTest();
314         Map<String, String> texpected = this.constructorRef();
315
316         boolean b = t.containsKey("q");
317         boolean bexp = texpected.containsKey("q");
318
319         assertEquals(t, texpected);
320         assertEquals(b, false);
321         assertEquals(bexp, false);
322     }
323
324     /**
325      * test for hasKey with one element, true.
326      */
327     @Test
328     public void testHasKeyOneTrue() {
329         Map<String, String> t = this.createFromArgsTest("q", "0");
330         Map<String, String> texpected = this.createFromArgsRef("q", "0");
331
332         boolean b = t.containsKey("q");
333         boolean bexp = texpected.containsKey("q");
334
335         assertEquals(t, texpected);
336         assertEquals(b, true);
337         assertEquals(bexp, true);
338     }
339
340     /**
341      * test for hasKey with one element, false.
342      */
343     @Test
344     public void testHasKeyOneFalse() {
345         Map<String, String> t = this.createFromArgsTest("q", "0");
346         Map<String, String> texpected = this.createFromArgsRef("q", "0");
```

```
347
348     boolean b = t.containsKey("x");
349     boolean bexp = texpected.containsKey("x");
350
351     assertEquals(t, texpected);
352     assertEquals(b, false);
353     assertEquals(bexp, false);
354 }
355
356 /**
357  * test for hasKey with more elements, true.
358  */
359 @Test
360 public void testHasKeyManyTrue() {
361     Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
362         "2", "r", "3");
363     Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
364         "1", "e", "2", "r", "3");
365
366     boolean b = t.containsKey("q");
367     boolean bexp = texpected.containsKey("q");
368
369     assertEquals(t, texpected);
370     assertEquals(b, true);
371     assertEquals(bexp, true);
372 }
373
374 /**
375  * test for hasKey with more elements, false.
376  */
377 @Test
378 public void testHasKeyManyFalse() {
379     Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
380         "2", "r", "3");
381     Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
382         "1", "e", "2", "r", "3");
383
384     boolean b = t.containsKey("o");
385     boolean bexp = texpected.containsKey("o");
386
387     assertEquals(t, texpected);
388     assertEquals(b, false);
389     assertEquals(bexp, false);
390 }
391
392 /**
393  * test for 0 size.
394  */
395 @Test
396 public void testSizeZero() {
397     Map<String, String> t = this.createFromArgsTest();
398     Map<String, String> texpected = this.createFromArgsRef();
399
400     int size = t.size();
401     int sizeexp = texpected.size();
402
403     assertEquals(t, texpected);
```

```
404         assertEquals(size, 0);
405         assertEquals(sizeexp, 0);
406     }
407
408     /**
409     * test for size 1 .
410     */
411     @Test
412     public void testSizeOne() {
413         Map<String, String> t = this.createFromArgsTest("q", "0");
414         Map<String, String> texpected = this.createFromArgsRef("q", "0");
415
416         int size = t.size();
417         int sizeexp = texpected.size();
418
419         assertEquals(t, texpected);
420         assertEquals(size, 1);
421         assertEquals(sizeexp, 1);
422     }
423
424     /**
425     * test for larger size.
426     */
427     @Test
428     public void testSizeMany() {
429         Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
430             "2", "r", "3");
431         Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
432             "1", "e", "2", "r", "3");
433
434         int size = t.size();
435         int sizeexp = texpected.size();
436
437         assertEquals(t, texpected);
438         assertEquals(size, 4);
439         assertEquals(sizeexp, 4);
440     }
441 }
442
```