```java
import static org.junit.Assert.assertEquals;

import org.junit.Test;

import components.map.Map;

/**
 * JUnit test fixture for {@code Map<String, String>}'s constructor and kernel
 * methods.
 *
 * @author Put your name here
 *
 */
public abstract class MapTest {

    /**
     * Invokes the appropriate {@code Map} constructor for the implementation
     * under test and returns the result.
     *
     * @return the new map
     * @ensures constructorTest = {}
     */
    protected abstract Map<String, String> constructorTest();

    /**
     * Invokes the appropriate {@code Map} constructor for the reference
     * implementation and returns the result.
     *
     * @return the new map
     * @ensures constructorRef = {}
     */
    protected abstract Map<String, String> constructorRef();

    /**
     *
     * Creates and returns a {@code Map<String, String>} of the implementation
     * under test type with the given entries.
     *
     * @param args
     *            the (key, value) pairs for the map
```

```
41      * @return the constructed map
42      * @requires <pre>
43      * [args.length is even]  and
44      * [the 'key' entries in args are unique]
45      * </pre>
46      * @ensures createFromArgsTest = [pairs in args]
47      */
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.hasKey(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.hasKey(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       * Boundary Test case with map1 = <>
86       */
87      @Test
88      public void test1() {
89          /*
90           * Set up variables and call method under test
91           */
92          Map<String, String> map1 = this.createFromArgsTest();
93          Map<String, String> expectedMap1 =
this.createFromArgsRef();
94
95          /*
96           * Assert that values of variables match expectations
97           */
98          assertEquals(map1, expectedMap1);
99      }
100
101     /**
102      * Routine Test case with map1 = <"Hello","Bye">
103      */
104     @Test
105     public void test2() {
106         /*
107          * Set up variables and call method under test
108          */
109         Map<String, String> map1 =
this.createFromArgsTest("Hello", "Bye");
110         Map<String, String> expectedMap1 =
this.createFromArgsRef("Hello",
111                 "Bye");
112
113         /*
114          * Assert that values of variables match expectations
115          */
116         assertEquals(map1, expectedMap1);
117     }
118
```

```
119     /**
120      * Challenging Test case with map1 = <"Hello, ""> 
121      */
122     @Test
123     public void test3() {
124         /*
125          * Set up variables and call method under test
126          */
127         Map<String, String> map1 =
    this.createFromArgsTest("Hello", "");
128         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Hello", "");
129 
130         /*
131          * Assert that values of variables match expectations
132          */
133         assertEquals(map1, expectedMap1);
134     }
135 
136     /**
137      * Boundary Test case with map1 = <>, addedOn = <"hello"> and
    <"bye">
138      */
139     @Test
140     public void test4() {
141         /*
142          * Set up variables and call method under test
143          */
144         Map<String, String> map1 = this.createFromArgsTest();
145         Map<String, String> expectedMap1 =
    this.createFromArgsRef("hello",
146                 "bye");
147         String addedOnKey = "hello";
148         String addedOnValue = "bye";
149         map1.add(addedOnKey, addedOnValue);
150 
151         /*
152          * Assert that values of variables match expectations
153          */
154         assertEquals(map1, expectedMap1);
155     }
156 
157     /**
158      * Routine Test case with map1 = <"Good","Luck">, addedOn =
```

```
    <"hello"> and
159      * <"bye">
160      */
161     @Test
162     public void test5() {
163         /*
164          * Set up variables and call method under test
165          */
166         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck");
167         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Good",
168                 "Luck", "hello", "bye");
169         String addedOnKey = "hello";
170         String addedOnValue = "bye";
171         map1.add(addedOnKey, addedOnValue);
172 
173         /*
174          * Assert that values of variables match expectations
175          */
176         assertEquals(map1, expectedMap1);
177     }
178 
179     /**
180      * Challenging Test case with map1 = <"Good","Luck">, addedOn
    = <"Bad"> and
181      * <"Luck">
182      */
183     @Test
184     public void test6() {
185         /*
186          * Set up variables and call method under test
187          */
188         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck");
189         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Good",
190                 "Luck", "Bad", "Luck");
191         String addedOnKey = "Bad";
192         String addedOnValue = "Luck";
193         map1.add(addedOnKey, addedOnValue);
194 
195         /*
196          * Assert that values of variables match expectations
```

```
197          */
198         assertEquals(map1, expectedMap1);
199     }
200 
201     /**
202      * Boundary Test case with map1 = <"Good","Luck">
203      */
204     @Test
205     public void test7() {
206         /*
207          * Set up variables and call method under test
208          */
209         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck");
210         Map<String, String> expectedMap1 =
    this.createFromArgsRef();
211         map1.remove("Good");
212 
213         /*
214          * Assert that values of variables match expectations
215          */
216         assertEquals(map1, expectedMap1);
217     }
218 
219     /**
220      * Routine Test case with map1 = <"Good","Luck", "Bad",
    "Luck">
221      */
222     @Test
223     public void test8() {
224         /*
225          * Set up variables and call method under test
226          */
227         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck",
228                 "Bad", "Luck");
229         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Bad",
230                 "Luck");
231         map1.remove("Good");
232 
233         /*
234          * Assert that values of variables match expectations
235          */
```

```
236         assertEquals(map1, expectedMap1);
237     }
238 
239     /**
240      * Challenging Test case with map1 = <"Good","Luck", "",
    "Luck">
241      */
242     @Test
243     public void test9() {
244         /*
245          * Set up variables and call method under test
246          */
247         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck", "",
248                 "Luck");
249         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Good",
250                 "Luck");
251         map1.remove("");
252 
253         /*
254          * Assert that values of variables match expectations
255          */
256         assertEquals(map1, expectedMap1);
257     }
258 
259     /**
260      * Boundary Test case with map1 = <"","">
261      */
262     @Test
263     public void test10() {
264         /*
265          * Set up variables and call method under test
266          */
267         Map<String, String> map1 = this.createFromArgsTest("",
    "");
268         Map<String, String> expectedMap1 =
    this.createFromArgsRef("", "");
269         String value = map1.value("");
270         String expectedValue = "";
271 
272         /*
273          * Assert that values of variables match expectations
274          */
```

```
275         assertEquals(map1, expectedMap1);
276         assertEquals(value, expectedValue);
277     }
278 
279     /**
280      * Routine Test case with map1 = <"Good","Luck">
281      */
282     @Test
283     public void test11() {
284         /*
285          * Set up variables and call method under test
286          */
287         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck");
288         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Good",
289                 "Luck");
290         String value = map1.value("Good");
291         String expectedValue = "Luck";
292 
293         /*
294          * Assert that values of variables match expectations
295          */
296         assertEquals(map1, expectedMap1);
297         assertEquals(value, expectedValue);
298     }
299 
300     /**
301      * Challenging Test case with map1 = <"Good","">
302      */
303     @Test
304     public void test12() {
305         /*
306          * Set up variables and call method under test
307          */
308         Map<String, String> map1 = this.createFromArgsTest("Good",
    "");
309         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Good", "");
310         String value = map1.value("Good");
311         String expectedValue = "";
312 
313         /*
314          * Assert that values of variables match expectations
```

```
315          */
316         assertEquals(map1, expectedMap1);
317         assertEquals(value, expectedValue);
318     }
319 
320     /**
321      * Boundary Test case with map1 = <"","">
322      */
323     @Test
324     public void test13() {
325         /*
326          * Set up variables and call method under test
327          */
328         Map<String, String> map1 = this.createFromArgsTest("",
    "");
329         Map<String, String> expectedMap1 =
    this.createFromArgsRef("", "");
330         Boolean value = map1.hasKey("");
331         Boolean expectedValue = true;
332 
333         /*
334          * Assert that values of variables match expectations
335          */
336         assertEquals(map1, expectedMap1);
337         assertEquals(value, expectedValue);
338     }
339 
340     /**
341      * Routine Test case with map1 = <"Good","Luck">
342      */
343     @Test
344     public void test14() {
345         /*
346          * Set up variables and call method under test
347          */
348         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck");
349         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Good",
350                 "Luck");
351         Boolean value = map1.hasKey("Good");
352         Boolean expectedValue = true;
353 
354         /*
```

```
355          * Assert that values of variables match expectations
356          */
357         assertEquals(map1, expectedMap1);
358         assertEquals(value, expectedValue);
359     }
360 
361     /**
362      * Challenging Test case with map1 = <"Good","Luck">
363      */
364     @Test
365     public void test15() {
366         /*
367          * Set up variables and call method under test
368          */
369         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck");
370         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Good",
371                 "Luck");
372         Boolean value = map1.hasKey("Bad");
373         Boolean expectedValue = false;
374 
375         /*
376          * Assert that values of variables match expectations
377          */
378         assertEquals(map1, expectedMap1);
379         assertEquals(value, expectedValue);
380     }
381 
382     /**
383      * Boundary Test case with map1 = <>
384      */
385     @Test
386     public void test16() {
387         /*
388          * Set up variables and call method under test
389          */
390         Map<String, String> map1 = this.createFromArgsTest();
391         Map<String, String> expectedMap1 =
    this.createFromArgsRef();
392         int value = map1.size();
393         int expectedValue = 0;
394 
395         /*
```

```
396          * Assert that values of variables match expectations
397          */
398         assertEquals(map1, expectedMap1);
399         assertEquals(value, expectedValue);
400     }
401 
402     /**
403      * Routine Test case with map1 = <"Bad","Luck">
404      */
405     @Test
406     public void test17() {
407         /*
408          * Set up variables and call method under test
409          */
410         Map<String, String> map1 = this.createFromArgsTest("Bad",
    "Luck");
411         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Bad",
412                 "Luck");
413         int value = map1.size();
414         int expectedValue = 1;
415 
416         /*
417          * Assert that values of variables match expectations
418          */
419         assertEquals(map1, expectedMap1);
420         assertEquals(value, expectedValue);
421     }
422 
423     /**
424      * Challenging Test case with map1 = <"","">
425      */
426     @Test
427     public void test18() {
428         /*
429          * Set up variables and call method under test
430          */
431         Map<String, String> map1 = this.createFromArgsTest("",
    "");
432         Map<String, String> expectedMap1 =
    this.createFromArgsRef("", "");
433         int value = map1.size();
434         int expectedValue = 1;
435 
```

```
436         /*
437          * Assert that values of variables match expectations
438          */
439         assertEquals(map1, expectedMap1);
440         assertEquals(value, expectedValue);
441     }
442 
443     /**
444      * Boundary Test case with map1 = <"","">
445      */
446     @Test
447     public void test19() {
448         /*
449          * Set up variables and call method under test
450          */
451         //Setup
452         Map<String, String> map1 = this.createFromArgsTest("",
    "");
453         Map<String, String> expectedMap1 =
    this.createFromArgsRef("", "");
454 
455         //Call
456         Map.Pair<String, String> capture = map1.removeAny();
457 
458         //Evaluation
459         assertEquals(true, expectedMap1.hasKey(null));
460         expectedMap1.remove("");
461         assertEquals(map1, expectedMap1);
462 
463         /*
464          * Assert that values of variables match expectations
465          */
466         assertEquals(map1, expectedMap1);
467     }
468 
469     /**
470      * Routine Test case with map1 = <"Good","Luck", "Bad",
    "Luck">
471      */
472     @Test
473     public void test20() {
474         /*
475          * Set up variables and call method under test
476          */
```

```
477         //Setup
478         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck",
479                 "Bad", "Luck");
480         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Good",
481                 "Luck", "Bad", "Luck");
482 
483         //Call
484         Map.Pair<String, String> capture = map1.removeAny();
485 
486         //Evaluation
487         assertEquals(true, expectedMap1.hasKey(capture.key()));
488         expectedMap1.remove(capture.key());
489         assertEquals(map1, expectedMap1);
490 
491         /*
492          * Assert that values of variables match expectations
493          */
494         assertEquals(map1, expectedMap1);
495     }
496 
497     /**
498      * Challenging Test case with map1 = <"Good","Luck",
    "","Luck">
499      */
500     @Test
501     public void test21() {
502         /*
503          * Set up variables and call method under test
504          */
505         //Setup
506         Map<String, String> map1 = this.createFromArgsTest("Good",
    "Luck", "",
507                 "");
508         Map<String, String> expectedMap1 =
    this.createFromArgsRef("Good",
509                 "Luck", "", "Luck");
510 
511         //Call
512         Map.Pair<String, String> capture = map1.removeAny();
513 
514         //Evaluation
515         assertEquals(true, expectedMap1.hasKey(capture.key()));
```

```
516         expectedMap1.remove(capture.key());
517         assertEquals(map1, expectedMap1);
518 
519         /*
520          * Assert that values of variables match expectations
521          */
522         assertEquals(map1, expectedMap1);
523     }
524 
525 }
```