```
 1 import static org.junit.Assert.assertEquals;
 6
 7 /**
 8  * JUnit test fixture for {@code Map<String, String>}'s constructor and kernel
 9  * methods.
10  *
11  * @author Put your name here
12  *
13  */
14 public abstract class MapTest {
15
16     /**
17      * Invokes the appropriate {@code Map} constructor for the implementation
18      * under test and returns the result.
19      *
20      * @return the new map
21      * @ensures constructorTest = {}
22      */
23     protected abstract Map<String, String> constructorTest();
24
25     final String a = "a", b = "b", c = "c", d = "d", e = "e", f = "f";
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.hasKey(args[i]) : ""
55                     + "Violation of: the 'key' entries in args are unique";
56             map.add(args[i], args[i + 1]);
57         }
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.hasKey(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     @Test
87     public void testAdd() {
88
89         Map<String, String> map1 = this.createFromArgsRef(this.a, this.b,
90                 this.c, this.d);
91         Map<String, String> map2 = this.createFromArgsRef(this.a, this.b,
92                 this.c, this.d, this.e, this.f);
93         map1.add(this.c, this.f);
94         assertEquals(map2, map1);
95     }
96
97     @Test
98     public void testRemove() {
99
100        Map<String, String> map1 = this.createFromArgsRef(this.a, this.b,
101                this.c, this.d);
102        Map<String, String> map2 = this.createFromArgsRef(this.a, this.b,
103                this.c, this.d, this.e, this.f);
104        map2.remove(this.e);
105        assertEquals(map1, map2);
106    }
107
108    @Test
109    public void testRemoveAny() {
110        Map<String, String> map1 = this.createFromArgsRef(this.a, this.b,
111                this.c, this.d);
112        Map<String, String> map2 = this.createFromArgsRef();
113        while (map1.size() != 0) {
114            map1.removeAny();
115        }
116        assertEquals(map2, map1);
117    }
118
119    @Test
120    public void testValue() {
121        Map<String, String> map = this.createFromArgsRef(this.a, this.b, this.c,
122                this.d);
```

```
123         String test = map.value(this.c);
124         assertEquals(this.b, test);
125     }
126 
127     @Test
128     public void testHasKey() {
129         Map<String, String> map = this.createFromArgsRef(this.a, this.b, this.c,
130                 this.d);
131         boolean test = map.hasKey(this.a);
132         assertEquals(true, test);
133     }
134 
135     @Test
136     public void testSize() {
137         Map<String, String> map = this.createFromArgsRef(this.a, this.b, this.c,
138                 this.d, this.e, this.f);
139         int test = map.size();
140         assertEquals(3, test);
141     }
142 
143     /*
144      * m = {}; m = {"one", 1} m = {("one" , 1),("zero", 0)} m = {("one" ,
145      * 1),("zero", 0),("negative one", -1)} m = {("one" , 1),("negative one",
146      * -1)} p = {"zero", 0} m = {("negative one", -1)} p = {"zero", 0} m =
147      * {("negative one", -1), ("cipher", 0)} p = {"zero", 0} m =
148      * {("negative one", -1), ("cipher", 0), ("zero", 0)} p = {"zero", 0} m =
149      * {("cipher", 0), ("zero", 0)} p = {"zero", 0} m = {("zero", 0)} p =
150      * {"zero", 0} m = {} p = {"zero", 0}
151      */
152 
153 }
```