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

/**
 * 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
     * @return the constructed map
     * @requires <pre>
     * [args.length is even]  and
     * [the 'key' entries in args are unique]
     * </pre>
     * @ensures createFromArgsTest = [pairs in args]
     */
    private Map<String, String> createFromArgsTest(String... args) {
        assert args.length % 2 == 0 : "Violation of: args.length is even";
        Map<String, String> map = this.constructorTest();
        for (int i = 0; i < args.length; i += 2) {
            assert !map.hasKey(args[i])
                    : "" + "Violation of: the 'key' entries in args are unique";
            map.add(args[i], args[i + 1]);
        }
        return map;
    }

    /**
     *
     * Creates and returns a {@code Map<String, String>} of the reference
```

```
63      * implementation type with the given entries.
64      *
65      * @param args
66      *            the (key, value) pairs for the map
67      * @return the constructed map
68      * @requires <pre>
69      * [args.length is even]  and
70      * [the 'key' entries in args are unique]
71      * </pre>
72      * @ensures createFromArgsRef = [pairs in args]
73      */
74     private Map<String, String> createFromArgsRef(String... args) {
75         assert args.length % 2 == 0 : "Violation of: args.length is even";
76         Map<String, String> map = this.constructorRef();
77         for (int i = 0; i < args.length; i += 2) {
78             assert !map.hasKey(args[i])
79                     : "" + "Violation of: the 'key' entries in args are unique";
80             map.add(args[i], args[i + 1]);
81         }
82         return map;
83     }
84
85     // TODO - add test cases for constructor, add, remove, removeAny, value, hasKey, and size
86     /**
87      * Test for constructor with an empty map.
88      */
89     @Test
90     public void constructorTestEmpty() {
91         Map<String, String> map = this.constructorTest();
92         Map<String, String> mapExp = this.constructorRef();
93
94         assertEquals(mapExp, map);
95     }
96
97     /**
98      * Test constructor with non empty map.
99      */
100    @Test
101    public void constructorTestOne() {
102        Map<String, String> map = this.constructorTest();
103        map.add("hi", "bye");
104        Map<String, String> mapExp = this.createFromArgsRef("hi", "bye");
105
106        assertEquals(mapExp, map);
107    }
108
109    /**
110     * Test add with empty map
111     */
112    @Test
113    public void testAddEmpty() {
114        Map<String, String> map = this.constructorTest();
115        Map<String, String> mapExp = this.createFromArgsRef();
116
117        assertEquals(mapExp, map);
118    }
119
```

```java
    /**
     * Test add with one pair
     */
    @Test
    public void testAddOne() {
        Map<String, String> map = this.constructorTest();
        map.add("hi", "bye");
        Map<String, String> mapExp = this.createFromArgsRef("hi", "bye");

        assertEquals(mapExp, map);
    }

    /**
     * Test remove with one pair
     */
    @Test
    public void testRemoveOne() {
        Map<String, String> map = this.createFromArgsTest("hi", "bye");
        map.add("hi", "bye");
        Pair<String, String> pair = map.remove("hi");

        Map<String, String> mapExp = this.createFromArgsRef("hi", "bye");
        Pair<String, String> pairExp = mapExp.remove("hi");
        assertEquals(pairExp, pair);
        assertEquals(mapExp, map);
    }

    /**
     * Test remove with two pairs
     */
    @Test
    public void testRemoveTwo() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        map.add("c", "d");
        map.remove("a");
        map.remove("c");
        Map<String, String> mapExp = this.createFromArgsRef("hi", "bye");
        mapExp.remove("hi");
        assertEquals(mapExp, map);
    }

    /**
     * Test for removeAny on map of length one.
     */
    @Test
    public void testRemoveAnyOne() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        Pair<String, String> rem = map.removeAny();
        Map<String, String> mapExp = this.createFromArgsRef("a", "b");
        Pair<String, String> remExp = mapExp.removeAny();
        assertEquals(remExp, rem);
        assertEquals(mapExp, map);
    }

    /**
```

```
177      * Test for removeAny on map of length two.
178      */
179     @Test
180     public void testRemoveAnyTwo() {
181         Map<String, String> map = this.constructorTest();
182         map.add("a", "b");
183         map.add("c", "d");
184         Pair<String, String> rem = map.removeAny();
185         Map<String, String> mapExp = this.createFromArgsRef("a", "b", "c", "d");
186         assertEquals(true, mapExp.hasKey(rem.key()));
187         assertEquals(mapExp, map);
188     }
189 
190     /**
191      * Test for value on map of length one.
192      */
193     @Test
194     public void testValueOne() {
195         Map<String, String> map = this.constructorTest();
196         map.add("a", "b");
197         Pair<String, String> rem = map.removeAny();
198         String val = rem.value();
199         Map<String, String> mapExp = this.createFromArgsRef("a", "b");
200         Pair<String, String> remExp = mapExp.removeAny();
201         String valExp = remExp.value();
202         assertEquals(valExp, val);
203         assertEquals(mapExp, map);
204     }
205 
206     /**
207      * Test for value on map of length two.
208      */
209     @Test
210     public void testValueTwo() {
211         Map<String, String> map = this.constructorTest();
212         map.add("a", "b");
213         map.add("c", "d");
214         Pair<String, String> rem = map.remove("a");
215         String val = rem.value();
216         Map<String, String> mapExp = this.createFromArgsRef("a", "b", "c", "d");
217         Pair<String, String> remExp = mapExp.remove("a");
218         String valExp = remExp.value();
219         assertEquals(valExp, val);
220         assertEquals(mapExp, map);
221     }
222 
223     /**
224      * Test for hasKey one map of length one.
225      */
226     @Test
227     public void testHasKeyOne() {
228         Map<String, String> map = this.constructorTest();
229         map.add("a", "b");
230         Map<String, String> mapExp = this.createFromArgsRef("a", "b");
231         assertEquals(true, map.hasKey("a"));
232         assertEquals(mapExp, map);
233     }
```

```
234
235     /**
236      * Test for hasKey one map of length two.
237      */
238     @Test
239     public void testHasKeyTwo() {
240         Map<String, String> map = this.constructorTest();
241         map.add("a", "b");
242         map.add("c", "d");
243         boolean key = false;
244         Map<String, String> mapExp = this.createFromArgsRef("a", "b", "c", "d");
245         if (map.hasKey("a") || map.hasKey("c")) {
246             key = true;
247         }
248         assertEquals(true, key);
249         assertEquals(mapExp, map);
250     }
251
252     /**
253      * Test for size on empty map.
254      */
255     @Test
256     public void testSizeEmpty() {
257         Map<String, String> map = this.constructorTest();
258         Map<String, String> mapExp = this.createFromArgsRef();
259         assertEquals(mapExp.size(), map.size());
260         assertEquals(0, map.size());
261         assertEquals(mapExp, map);
262     }
263
264     /**
265      * Test for size on map of length one.
266      */
267     @Test
268     public void testSizeOne() {
269         Map<String, String> map = this.constructorTest();
270         map.add("a", "b");
271         Map<String, String> mapExp = this.createFromArgsRef("a", "b");
272         assertEquals(mapExp.size(), map.size());
273         assertEquals(1, map.size());
274         assertEquals(mapExp, map);
275     }
276
277     /**
278      * Test for size on map of length three.
279      */
280     @Test
281     public void testSizeTwo() {
282         Map<String, String> map = this.constructorTest();
283         map.add("a", "b");
284         map.add("c", "d");
285         map.add("e", "f");
286         Map<String, String> mapExp = this.createFromArgsRef("a", "b", "c", "d", "e", "f");
287         assertEquals(mapExp.size(), map.size());
288         assertEquals(3, map.size());
289         assertEquals(mapExp, map);
290     }
```

```
291
292 }
293
```