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

/**
 * JUnit test fixture for {@code Map<String, String>}'s constructor and kernel
 * methods.
 *
 * @author Gabe Azzarita and Ty Fredrick
 *
 */
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
     * implementation type with the given entries.
     *
```

```
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     // Testing default contructor
86     @Test
87     public final void testForEmptyConstructor() {
88         Map<String, String> test = this.constructorTest();
89         Map<String, String> ref = this.constructorRef();
90
91         assertEquals(test, ref);
92     }
93
94     // Test constructor with arguments
95     @Test
96     public final void testForNonEmptyConstructor() {
97         Map<String, String> test = this.createFromArgsTest("A", "B", "1", "2");
98         Map<String, String> ref = this.createFromArgsRef("A", "B", "1", "2");
99
100        assertEquals(test, ref);
101    }
102
103    // Testing add function on an empty map
104    @Test
105    public final void testForAddEmpty() {
106        Map<String, String> test = this.createFromArgsTest();
107        Map<String, String> ref = this.createFromArgsRef("A", "B");
108
109        test.add("A", "B");
110
111        assertEquals(test, ref);
112    }
113
114    // Testing add on a non-empty map
115    @Test
116    public final void testForAdd() {
117        Map<String, String> test = this.createFromArgsTest("A", "B");
118        Map<String, String> ref = this.createFromArgsRef("A", "B", "1", "2");
119
120        test.add("1", "2");
121
122        assertEquals(test, ref);
123    }
```

```
124
125     // Testing add with multiple add calls
126     @Test
127     public final void testForAddMultiple() {
128         Map<String, String> test = this.createFromArgsTest();
129         Map<String, String> ref = this.createFromArgsRef("A", "B", "1", "2");
130
131         test.add("A", "B");
132         test.add("1", "2");
133
134         assertEquals(test, ref);
135     }
136
137     // Testing remove, and checking that it returns correct pair
138     @Test
139     public final void testForRemove() {
140         Map<String, String> test = this.createFromArgsTest("A", "B");
141         Map<String, String> ref = this.createFromArgsRef("A", "B");
142
143         Pair<String, String> testRemoved = test.remove("A");
144         Pair<String, String> refRemoved = ref.remove("A");
145
146         assertEquals(test, ref);
147         // Make sure remove function returns pair correctly
148         assertEquals(testRemoved, refRemoved);
149     }
150
151     // Testing remove with multiple calls
152     @Test
153     public final void testForRemoveMultiple() {
154         Map<String, String> test = this.createFromArgsTest("A", "B", "1", "2");
155         Map<String, String> ref = this.createFromArgsRef();
156
157         test.remove("1");
158         test.remove("A");
159
160         assertEquals(test, ref);
161     }
162
163     // Testing remove with multiple calls
164     @Test
165     public final void testForRemoveMultipleHard() {
166         Map<String, String> test = this.createFromArgsTest("1", "i", "2", "ii",
167                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
168                 "viij", "9", "ix", "10", "x");
169         Map<String, String> ref = this.createFromArgsRef("3", "iii", "4",
170                 "iiij", "5", "v", "6", "vi", "7", "vii");
171
172         test.remove("1");
173         test.remove("2");
174         test.remove("8");
175         test.remove("9");
176         test.remove("10");
177         assertEquals(test, ref);
178     }
179
180     // Testing removeAny with one pair
181     @Test
182     public final void testRemoveAnyOnePair() {
```

```
183         Map<String, String> test = this.createFromArgsTest("1", "i");
184
185         Map<String, String> ref = this.createFromArgsRef("1", "i");
186
187         Map.Pair<String, String> element = test.removeAny();
188
189         assertEquals(test.hasKey(element.key()), false);
190         assertEquals(ref.hasKey(element.key()), true);
191         assertEquals(test.size(), ref.size() - 1);
192     }
193
194     // Testing removeAny with multiple pairs
195     @Test
196     public final void testRemoveAny() {
197         Map<String, String> test = this.createFromArgsTest("1", "i", "2", "ii",
198                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
199                 "viij", "9", "ix", "10", "x");
200         Map<String, String> ref = this.createFromArgsRef("1", "i", "2", "ii",
201                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
202                 "viij", "9", "ix", "10", "x");
203
204         Map.Pair<String, String> element = test.removeAny();
205
206         // Check that pair is no longer in test, but is still in ref
207         assertEquals(test.hasKey(element.key()), false);
208         // Make sure removeAny properly returns element by referencing ref Map
209         assertEquals(ref.hasKey(element.key()), true);
210         assertEquals(ref.value(element.key()), element.value());
211         // Make sure size is updated
212         assertEquals(test.size(), ref.size() - 1);
213     }
214
215     // Testing size with empty map
216     @Test
217     public final void testSizeZero() {
218         Map<String, String> test = this.constructorTest();
219         assertEquals(0, test.size());
220     }
221
222     // Testing size with a non-empty map
223     @Test
224     public final void testSizeNonZero() {
225         Map<String, String> test = this.createFromArgsTest("1", "i", "2", "ii",
226                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
227                 "viij", "9", "ix", "10", "x");
228
229         final int ten = 10;
230
231         assertEquals(ten, test.size());
232     }
233
234     // Testing hasKey when map contains key
235     @Test
236     public final void testHasKeyTrue() {
237         Map<String, String> test = this.createFromArgsTest("1", "i", "2", "ii",
238                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
239                 "viij", "9", "ix", "10", "x");
240         Map<String, String> ref = this.createFromArgsRef("1", "i", "2", "ii",
241                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
```

```
242                 "viij", "9", "ix", "10", "x");
243
244         assertEquals(test.hasKey("1"), true);
245         assertEquals(test.hasKey("5"), true);
246
247         // Make sure that hasKey does not change map
248         assertEquals(test, ref);
249     }
250
251     // Testing hasKey when map does not contain key
252     @Test
253     public final void testHasKeyFalse() {
254         Map<String, String> test = this.createFromArgsTest("1", "i", "2", "ii",
255                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
256                 "viij", "9", "ix", "10", "x");
257         Map<String, String> ref = this.createFromArgsRef("1", "i", "2", "ii",
258                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
259                 "viij", "9", "ix", "10", "x");
260
261         assertEquals(test.hasKey("i"), false);
262         assertEquals(test.hasKey("11"), false);
263
264         // Make sure that hasKey does not change map
265         assertEquals(test, ref);
266     }
267
268     // Routine cases for value
269     @Test
270     public final void testValue() {
271         Map<String, String> test = this.createFromArgsTest("1", "i", "2", "ii",
272                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
273                 "viij", "9", "ix", "10", "x");
274         Map<String, String> ref = this.createFromArgsRef("1", "i", "2", "ii",
275                 "3", "iii", "4", "iiij", "5", "v", "6", "vi", "7", "vii", "8",
276                 "viij", "9", "ix", "10", "x");
277
278         assertEquals("vi", test.value("6"));
279         assertEquals("viij", test.value("8"));
280
281         // Make sure that value does not change map
282         assertEquals(test, ref);
283     }
284
285 }
286
```