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

import org.junit.Test;

import components.map.Map;

/**
 * JUnit test fixture for {@code Map<String, String>}'s constructor and kernel
 * methods.
 *
 * @author Charan Nanduri, Evan Frisbie
 *
 */
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();

    // TODO - add test cases for constructor, add, remove, removeAny, value,
    // hasKey, and size

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

```
41      * under test type with the given entries.
42      *
43      * @param args
44      *            the (key, value) pairs for the map
45      * @return the constructed map
46      * @requires <pre>
47      * [args.length is even]  and
48      * [the 'key' entries in args are unique]
49      * </pre>
50      * @ensures createFromArgsTest = [pairs in args]
51      */
52     private Map<String, String> createFromArgsTest(String... args)
    {
53         assert args.length % 2 == 0 : "Violation of: args.length is
    even";
54         Map<String, String> map = this.constructorTest();
55         for (int i = 0; i < args.length; i += 2) {
56             assert !map.hasKey(args[i]) : ""
57                     + "Violation of: the 'key' entries in args are
    unique";
58             map.add(args[i], args[i + 1]);
59         }
60         return map;
61     }
62
63     /**
64      *
65      * Creates and returns a {@code Map<String, String>} of the
    reference
66      * implementation type with the given entries.
67      *
68      * @param args
69      *            the (key, value) pairs for the map
70      * @return the constructed map
71      * @requires <pre>
72      * [args.length is even]  and
73      * [the 'key' entries in args are unique]
74      * </pre>
75      * @ensures createFromArgsRef = [pairs in args]
76      */
77     private Map<String, String> createFromArgsRef(String... args) {
78         assert args.length % 2 == 0 : "Violation of: args.length is
    even";
79         Map<String, String> map = this.constructorRef();
80         for (int i = 0; i < args.length; i += 2) {
```

```
 81             assert !map.hasKey(args[i]) : ""
 82                     + "Violation of: the 'key' entries in args are
    unique";
 83             map.add(args[i], args[i + 1]);
 84         }
 85         return map;
 86     }
 87
 88     /*
 89      * Testing Constructor cases
 90      */
 91     @Test
 92     public final void testNoArgConst() {
 93         Map<String, String> map = this.constructorTest();
 94         Map<String, String> exp = this.constructorRef();
 95         assertEquals(exp, map);
 96         assertEquals(exp.size(), map.size());
 97     }
 98
 99     @Test
100     public final void testEmptyArgConst() {
101         Map<String, String> map = this.createFromArgsTest("", "");
102         Map<String, String> exp = this.createFromArgsRef("", "");
103         assertEquals(exp, map);
104         assertEquals(exp.size(), map.size());
105     }
106
107     @Test
108     public final void testNonEmptyArgConst() {
109         Map<String, String> map = this.createFromArgsTest("one",
    "two", "three",
110                 "four");
111         Map<String, String> exp = this.createFromArgsRef("one",
    "two", "three",
112                 "four");
113         assertEquals(exp, map);
114         assertEquals(exp.size(), map.size());
115     }
116
117     /*
118      * Testing Kernel Method Cases
119      */
120     @Test
121     public final void testAddEmptyMap() {
122         Map<String, String> map = this.createFromArgsTest();
```

```
123         Map<String, String> exp = this.createFromArgsRef();
124         map.add("one", "two");
125         exp.add("one", "two");
126         assertEquals(exp, map);
127         assertEquals(exp.size(), map.size());
128     }
129 
130     @Test
131     public final void testAddNonEmptyMap() {
132         Map<String, String> map = this.createFromArgsTest("one",
    "two", "three",
133                 "four");
134         Map<String, String> exp = this.createFromArgsRef("one",
    "two", "three",
135                 "four");
136         map.add("five", "six");
137         exp.add("five", "six");
138         assertEquals(exp, map);
139         assertEquals(exp.size(), map.size());
140     }
141 
142     @Test
143     public final void testRemovePair() {
144         Map<String, String> map = this.createFromArgsTest("one",
    "two");
145         Map<String, String> exp = this.createFromArgsRef("one",
    "two");
146         Map.Pair<String, String> pair = map.remove("one");
147         Map.Pair<String, String> pExp = exp.remove("one");
148         assertEquals(exp, map);
149         assertEquals(pair, pExp);
150         assertEquals(exp.size(), map.size());
151     }
152 
153     @Test
154     public final void testRemoveMultiplePairs() {
155         Map<String, String> map = this.createFromArgsTest("one",
    "two", "three",
156                 "four");
157         Map<String, String> exp = this.createFromArgsRef("one",
    "two", "three",
158                 "four");
159         Map.Pair<String, String> pair = map.remove("one");
160         Map.Pair<String, String> pExp = exp.remove("one");
161         assertEquals(exp, map);
```

```
162         assertEquals(pair, pExp);
163         assertEquals(exp.size(), map.size());
164     }
165 
166     @Test
167     public final void testRemoveAny() {
168         Map<String, String> map = this.createFromArgsTest("one",
    "two", "three",
169                 "four");
170         Map<String, String> exp = this.createFromArgsRef("one",
    "two", "three",
171                 "four");
172         Map.Pair<String, String> pair = map.removeAny();
173         assertTrue(exp.hasKey(pair.key()));
174         assertEquals(exp.value(pair.key()), pair.value());
175         Map.Pair<String, String> pExp = exp.remove(pair.key());
176         assertEquals(exp, map);
177         assertEquals(pExp, pair);
178         assertEquals(exp.size(), map.size());
179     }
180 
181     @Test
182     public final void testRemoveAnyEmptyMap() {
183         Map<String, String> map = this.createFromArgsTest("one",
    "two");
184         Map<String, String> exp = this.createFromArgsRef("one",
    "two");
185         Map.Pair<String, String> pair = map.removeAny();
186         assertTrue(exp.hasKey(pair.key()));
187         assertEquals(exp.value(pair.key()), pair.value());
188         Map.Pair<String, String> pExp = exp.remove(pair.key());
189         assertEquals(exp, map);
190         assertEquals(pExp, pair);
191         assertEquals(exp.size(), map.size());
192     }
193 
194     @Test
195     public final void testVal() {
196         Map<String, String> map = this.createFromArgsTest("one",
    "two", "three",
197                 "four");
198         Map<String, String> exp = this.createFromArgsRef("one",
    "two", "three",
199                 "four");
200         assertEquals(exp.value("three"), map.value("three"));
```

```
201         assertEquals(exp, map);
202         assertEquals(exp.size(), map.size());
203     }
204 
205     @Test
206     public final void testKeyEmpty() {
207         Map<String, String> map = this.createFromArgsTest();
208         Map<String, String> exp = this.createFromArgsRef();
209         assertEquals(exp.hasKey("one"), map.hasKey("one"));
210         assertEquals(exp, map);
211         assertEquals(exp.size(), map.size());
212     }
213 
214     @Test
215     public final void testKeyNonEmptyAndNoKey() {
216         Map<String, String> map = this.createFromArgsTest("one",
    "two");
217         Map<String, String> exp = this.createFromArgsRef("one",
    "two");
218         assertEquals(exp.hasKey("four"), map.hasKey("four"));
219         assertEquals(exp, map);
220         assertEquals(exp.size(), map.size());
221     }
222 
223     @Test
224     public final void testKeyExists() {
225         Map<String, String> map = this.createFromArgsTest("one",
    "two",
226                 "threee", "four");
227         Map<String, String> exp = this.createFromArgsRef("one",
    "two", "threee",
228                 "four");
229         assertEquals(exp.hasKey("four"), map.hasKey("four"));
230         assertEquals(exp, map);
231         assertEquals(exp.size(), map.size());
232     }
233 
234     @Test
235     public final void testSizeEmpty() {
236         Map<String, String> map = this.createFromArgsTest();
237         Map<String, String> exp = this.createFromArgsRef();
238         assertEquals(exp.size(), map.size());
239         assertEquals(exp, map);
240     }
241 
```

```
242     @Test
243     public final void testSizeOne() {
244         Map<String, String> map = this.createFromArgsTest("one",
    "two");
245         Map<String, String> exp = this.createFromArgsRef("one",
    "two");
246         assertEquals(exp.size(), map.size());
247         assertEquals(exp, map);
248     }
249 
250     @Test
251     public final void testSizeMultiple() {
252         Map<String, String> map = this.createFromArgsTest("one",
    "two", "three",
253                 "four");
254         Map<String, String> exp = this.createFromArgsRef("one",
    "two", "three",
255                 "four");
256         assertEquals(exp.size(), map.size());
257         assertEquals(exp, map);
258     }
259 
260 }
```