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

import org.junit.Test;

import components.map.Map;
import components.map.Map.Pair;

/**
 * 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;
    }
```

```
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      /**
87       * Test constructor.
88       */
89      @Test
90      public final void testConstructor() {
91          Map<String, String> map = this.constructorTest();
92          Map<String, String> mapExpected = this.constructorRef();
93
94          assertEquals(mapExpected, map);
95      }
96
97      /**
98       * Test case for add.
99       */
100     @Test
101     public final void addTestOne() {
102         Map<String, String> map = this.createFromArgsTest("play", "time");
103         Map<String, String> mapExpected = this.createFromArgsRef("play", "time",
104                 "money", "change");
105
106         map.add("money", "change");
107
108         boolean result = map.hasKey("money") && map.hasValue("change");
109
110         assertTrue(result);
111         assertEquals(mapExpected, map);
112     }
113
114     /**
115      * Test case for add.
116      */
117     @Test
118     public final void addTestTwo() {
```

```
119         Map<String, String> map = this.createFromArgsTest("make", "bake",
120                 "change", "craze");
121         Map<String, String> mapExpected = this.createFromArgsRef("make", "bake",
122                 "change", "craze", "fry", "dry");
123 
124         map.add("fry", "dry");
125 
126         boolean result = map.hasKey("fry") && map.hasValue("dry");
127 
128         assertTrue(result);
129         assertEquals(mapExpected, map);
130     }
131 
132     /**
133      * Test case for remove.
134      */
135     @Test
136     public final void removeTestOne() {
137         Map<String, String> map = this.createFromArgsTest("make", "bake");
138         Map<String, String> mapExpected = this.createFromArgsRef();
139 
140         map.remove("make");
141 
142         boolean result = map.hasKey("make") && map.hasValue("bake");
143 
144         assertEquals(result, false);
145         assertEquals(mapExpected, map);
146     }
147 
148     /**
149      * Test case for remove.
150      */
151     @Test
152     public final void removeTestTwo() {
153         Map<String, String> map = this.createFromArgsTest("make", "bake",
154                 "craze", "haze", "fry", "dry");
155         Map<String, String> mapExpected = this.createFromArgsRef("make", "bake",
156                 "fry", "dry");
157 
158         map.remove("craze");
159 
160         boolean result = map.hasKey("craze") && map.hasValue("haze");
161 
162         assertEquals(result, false);
163         assertEquals(mapExpected, map);
164     }
165 
166     /**
167      * Test case for removeAny.
168      */
169     @Test
170     public final void removeAnyTestOne() {
171         /**
172          * Setup expected and actual objects
173          */
174         Map<String, String> map = this.createFromArgsTest("play", "time",
175                 "money", "change");
176         Map<String, String> mapExpected = this.createFromArgsRef("play", "time",
177                 "money", "change");
```

```
178
179         /**
180          * Call the method on the actual object
181          */
182         Pair<String, String> p = map.removeAny();
183
184         /**
185          * Evaluate the values given from the call
186          */
187         assertEquals(mapExpected.hasKey(p.key()), true);
188         assertEquals(mapExpected.hasValue(p.value()), true);
189
190         Pair<String, String> pair = mapExpected.remove(p.key());
191
192         assertEquals(p, pair);
193         assertEquals(mapExpected, map);
194
195     }
196
197     /**
198      * Test value on a map of 1.
199      */
200     @Test
201     public final void valueTestMapOne() {
202         Map<String, String> map = this.createFromArgsTest("red", "apple");
203         Map<String, String> mapExpected = this.createFromArgsRef("red",
204                 "apple");
205
206         String value = map.value("red");
207
208         assertEquals("apple", value);
209         assertEquals(mapExpected, map);
210
211     }
212
213     /**
214      * Test value on a map of 3.
215      */
216     @Test
217     public final void valueTestMapThree() {
218         Map<String, String> map = this.createFromArgsTest("red", "apple",
219                 "green", "kiwi", "purple", "grapes");
220         Map<String, String> mapExpected = this.createFromArgsRef("red", "apple",
221                 "green", "kiwi", "purple", "grapes");
222
223         String value = map.value("purple");
224
225         assertEquals("grapes", value);
226         assertEquals(mapExpected, map);
227     }
228
229     /**
230      * Test value on a map of 6.
231      */
232     @Test
233     public final void valueTestMapSix() {
234         Map<String, String> map = this.createFromArgsTest("red", "apple",
235                 "green", "kiwi", "purple", "grapes", "clear", "water", "blue",
236                 "blueberry", "yellow", "banana");
```

```
237         Map<String, String> mapExpected = this.createFromArgsRef("red", "apple",
238                 "green", "kiwi", "purple", "grapes", "clear", "water", "blue",
239                 "blueberry", "yellow", "banana");
240 
241         String value = map.value("blue");
242 
243         assertEquals("blueberry", value);
244         assertEquals(mapExpected, map);
245     }
246 
247     /**
248      * Test has key on a map of 1.
249      */
250     @Test
251     public final void hasKeyTestMapOne() {
252         Map<String, String> map = this.createFromArgsTest("red", "apple");
253         Map<String, String> mapExpected = this.createFromArgsRef("red",
254                 "apple");
255 
256         boolean bool = map.hasKey("red");
257 
258         assertEquals(true, bool);
259         assertEquals(mapExpected, map);
260 
261     }
262 
263     /**
264      * Test has key on a map of 3.
265      */
266     @Test
267     public final void hasKeyTestMapThree() {
268         Map<String, String> map = this.createFromArgsTest("red", "apple",
269                 "green", "kiwi", "purple", "grapes");
270         Map<String, String> mapExpected = this.createFromArgsRef("red", "apple",
271                 "green", "kiwi", "purple", "grapes");
272 
273         boolean bool = map.hasKey("green");
274 
275         assertEquals(true, bool);
276         assertEquals(mapExpected, map);
277 
278     }
279 
280     /**
281      * Test has key on a map of 6.
282      */
283     @Test
284     public final void hasKeyTestMapSix() {
285         Map<String, String> map = this.createFromArgsTest("red", "apple",
286                 "green", "kiwi", "purple", "grapes", "clear", "water", "blue",
287                 "blueberry", "yellow", "banana");
288         Map<String, String> mapExpected = this.createFromArgsRef("red", "apple",
289                 "green", "kiwi", "purple", "grapes", "clear", "water", "blue",
290                 "blueberry", "yellow", "banana");
291 
292         boolean bool = map.hasKey("clear");
293 
294         assertEquals(true, bool);
295         assertEquals(mapExpected, map);
```

```
296
297     }
298
299     /**
300      * Test size on an empty map.
301      */
302     @Test
303     public final void sizeTestEmptyMap() {
304         Map<String, String> map = this.createFromArgsTest();
305
306         int i = map.size();
307
308         assertEquals(0, i);
309     }
310
311     /**
312      * Test size on non-empty map.
313      */
314     @Test
315     public final void sizeTestNonEmptyMap() {
316         Map<String, String> map = this.createFromArgsTest("red", "apples",
317                 "orange", "oranges", "purple", "grapes", "green", "kiwis");
318
319         int i = map.size();
320
321         assertEquals(4, i);
322     }
323
324 }
325
```