```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 Zhuoyang Li + Xinci Ma
 *
 */
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();

    /**
```

```
36      *
37      * Creates and returns a {@code Map<String,
  String>} of the implementation
38      * under test type with the given entries.
39      *
40      * @param args
41      *            the (key, value) pairs for the map
42      * @return the constructed map
43      * @requires <pre>
44      * [args.length is even]  and
45      * [the 'key' entries in args are unique]
46      * </pre>
47      * @ensures createFromArgsTest = [pairs in args]
48      */
49     private Map<String, String>
  createFromArgsTest(String... args) {
50         assert args.length % 2 == 0 : "Violation of:
  args.length is even";
51         Map<String, String> map =
  this.constructorTest();
52         for (int i = 0; i < args.length; i += 2) {
53             assert !map.hasKey(args[i]) : ""
54                     + "Violation of: the 'key'
  entries in args are unique";
55             map.add(args[i], args[i + 1]);
56         }
57         return map;
58     }
59
60     /**
61      *
62      * 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,
 86     // hasKey, and size
 87
 88     //no argument constructor
 89     @Test
 90     public final void testConstructor1() {
 91         Map<String, String> test =
   this.constructorTest();
 92         Map<String, String> ref =
   this.constructorRef();
 93         assertEquals(ref, test);
 94         assertTrue(test.size() == 0);
 95     }
 96
 97     //argument constructor
 98     @Test
 99     public final void testConstructor2() {
100         Map<String, String> test =
   this.createFromArgsTest("a", "1", "c", "d");
101         Map<String, String> ref =
```

```
    this.createFromArgsRef("a", "1", "c", "d");
102         assertEquals(ref, test);
103         assertTrue(test.size() == 2);
104     }
105 
106     //add one pair
107     @Test
108     public void testAdd1() {
109         Map<String, String> test =
    this.createFromArgsTest("a", "1");
110         test.add("b", "2");
111         Map<String, String> ref =
    this.createFromArgsRef("a", "1", "b", "2");
112 
    assertTrue(test.toString().equals(ref.toString()));
113     }
114 
115     //add pairs to an empty map
116     @Test
117     public void testAdd2() {
118         Map<String, String> test =
    this.createFromArgsTest();
119         test.add("a", "2");
120         test.add("b", "3");
121         Map<String, String> ref =
    this.createFromArgsRef("a", "2", "b", "3");
122 
    assertTrue(test.toString().equals(ref.toString()));
123     }
124 
125     @Test
126     public final void testRemove1() {
127         Map<String, String> test =
    this.createFromArgsTest("a", "b", "c", "d");
128         Map<String, String> ref =
    this.createFromArgsRef("a", "b", "c", "d");
129         test.remove("a");
130         ref.remove("a");
131 
    assertTrue(test.toString().equals(ref.toString()));
```

```
132     }
133
134     @Test
135     public final void testRemove2() {
136         Map<String, String> test =
    this.createFromArgsTest("c", "d");
137         Map<String, String> ref =
    this.createFromArgsRef();
138         test.remove("c");
139
    assertTrue(test.toString().equals(ref.toString()));
140     }
141
142     @Test
143     public final void testRemoveAny1() {
144         Map<String, String> test =
    this.createFromArgsTest("a", "b", "c", "d",
145                 "e", "f", "g", "h", "i", "j");
146         Map<String, String> ref =
    this.createFromArgsRef("a", "b", "c", "d",
147                 "e", "f", "g", "h", "i", "j");
148         test.removeAny();
149         ref.removeAny();
150         assertEquals(test.size(), ref.size());
151     }
152
153     @Test
154     public final void testRemoveAny2() {
155         Map<String, String> test =
    this.createFromArgsTest("1", "2");
156         test.removeAny();
157
158         assertEquals(test.size(), 0);
159     }
160
161     @Test
162     public final void testValue1() {
163         Map<String, String> test =
    this.createFromArgsTest("a", "b", "c", "d");
164         assertEquals(test.value("a"), "b");
```

```
165     }
166
167     @Test
168     public final void hasKey1() {
169         Map<String, String> test =
    this.createFromArgsTest("a", "b", "c", "d");
170         assertTrue(test.hasKey("a"));
171     }
172
173     @Test
174     public final void hasKey2() {
175         Map<String, String> test =
    this.createFromArgsTest("a", "b", "c", "d");
176         assertTrue(!test.hasKey("e"));
177     }
178
179     @Test
180     public final void size1() {
181         Map<String, String> test =
    this.createFromArgsTest("a", "b", "c", "d");
182         assertEquals(test.size(), 2);
183     }
184
185     @Test
186     public final void size2() {
187         Map<String, String> test =
    this.createFromArgsTest();
188         assertEquals(test.size(), 0);
189     }
190
191 }
192
```