```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
     * implementation 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 createFromArgsRef = [pairs in args]
     */
    private Map<String, String> createFromArgsRef(String... args) {
        assert args.length % 2 == 0 : "Violation of: args.length is even";
        Map<String, String> map = this.constructorRef();
        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;
    }

    /*
     * Test cases for constructors
     */

    @Test
    public final void testNoArgumentConstructor() {
        /*
         * Set up variables and call method under test
         */
        Map<String, String> m = this.constructorTest();
        Map<String, String> mExpected = this.constructorRef();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, m);
    }

    /*
     * Test cases for kernel methods
     */

    @Test
    public final void testAddEmpty() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest();
        Map<String, String> mExpected = this.createFromArgsRef("1",
"red");
        /*
         * Call method under test
         */
        m.add("1", "red");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, m);
    }

    @Test
    public final void testAddOne() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red");
        Map<String, String> mExpected = this.createFromArgsRef("1",
"red", "2",
                "green");
        /*
         * Call method under test
         */
        m.add("2", "green");
        /*
```

```
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, m);
    }

    @Test
    public final void testAddMoreThanOne() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red",
"2",
                "green");
        Map<String, String> mExpected = this.createFromArgsRef("1",
"red", "2",
                "green", "3", "blue");
        /*
         * Call method under test
         */
        m.add("3", "blue");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, m);
    }

    @Test
    public final void testRemoveLeavingZero() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red");
        Map<String, String> mExpected = this.createFromArgsRef();
        /*
         * Call method under test
         */
        m.remove("1");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, m);
    }

    @Test
    public final void testRemoveLeavingOne() {
        /*
         * Set up variables
```

```java
         */
        Map<String, String> m = this.createFromArgsTest("1", "red", "2",
                "green");
        Map<String, String> mExpected = this.createFromArgsRef("1", "red");
        /*
         * Call method under test
         */
        m.remove("2");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, m);
    }

    @Test
    public final void testRemoveLeavingMoreThanOne() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red", "2",
                "green", "3", "blue");
        Map<String, String> mExpected = this.createFromArgsRef("1", "red", "3",
                "blue");
        /*
         * Call method under test
         */
        m.remove("2");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, m);
    }

    @Test
    public final void testRemoveAnyLeavingEmpty() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red");
        Map<String, String> m2 = this.createFromArgsTest("1", "red");
        /*
         * Call method under test
         */
        int expected = 0;
        Map.Pair<String, String> pair = m.removeAny();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(expected, m.size());
        assertEquals(m2.hasKey(pair.key()), true);

    }

    @Test
    public final void testRemoveAnyLeavingNotEmpty() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red",
"2",
                "blue");
        Map<String, String> m2 = this.createFromArgsTest("1", "red",
"2",
                "blue");
        /*
         * Call method under test
         */
        int expected = 1;
        Map.Pair<String, String> pair = m.removeAny();
        int mSize = m.size();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(expected, mSize);
        assertEquals(m2.hasKey(pair.key()), true);
    }

    @Test
    public final void testSizeEmpty() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest();
        Map<String, String> mExpected = this.createFromArgsRef();
        Map<String, String> m2Expected = this.createFromArgsRef();
        /*
         * Call method under test
         */
        int mSize = m.size();
        int mExpectedSize = mExpected.size();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpectedSize, mSize);
        assertEquals(m2Expected, m);
    }

    @Test
    public final void testSizeOne() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red");
        Map<String, String> mExpected = this.createFromArgsRef("1",
"red");
        Map<String, String> m2Expected = this.createFromArgsRef("1",
"red");
        /*
         * Call method under test
         */
        int mSize = m.size();
        int mExpectedSize = mExpected.size();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpectedSize, mSize);
        assertEquals(m2Expected, m);
    }

    @Test
    public final void testSizeMoreThanOne() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red",
"2",
                "green");
        Map<String, String> m2Expected = this.createFromArgsTest("1",
"red",
                "2", "green");
        /*
         * Call method under test
         */
        int mSize = m.size();
        int mExpectedSize = 2;
        /*
         * Assert that values of variables match expectations
```

```java
         */
        assertEquals(mExpectedSize, mSize);
        assertEquals(m2Expected, m);
    }

    @Test
    public final void testValueOne() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red");
        /*
         * Call method under test
         */
        String mKey = m.value("1");
        String mExpected = "red";
        Map<String, String> m2Expected = this.createFromArgsTest("1", "red");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, mKey);
        assertEquals(m2Expected, m);
    }

    @Test
    public final void testValueMoreThanOne() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red", "2",
                "green");
        /*
         * Call method under test
         */
        String mKey = m.value("1");
        String mExpected = "red";
        Map<String, String> m2Expected = this.createFromArgsTest("1", "red",
                "2", "green");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, mKey);
        assertEquals(m2Expected, m);
    }

    @Test
    public final void testHasKeyTrue() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red");
        Map<String, String> m2Expected = this.createFromArgsTest("1",
"red");
        /*
         * Call method under test
         */
        boolean mKey = m.hasKey("1");
        boolean mExpected = true;
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, mKey);
        assertEquals(m2Expected, m);
    }

    @Test
    public final void testHasKeyFalse() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest("1", "red");
        Map<String, String> m2Expected = this.createFromArgsTest("1",
"red");
        /*
         * Call method under test
         */
        boolean mKey = m.hasKey("2");
        boolean mExpected = false;
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, mKey);
        assertEquals(m2Expected, m);
    }

    @Test
    public final void testHasKeyFalseOnEmpty() {
        /*
         * Set up variables
         */
        Map<String, String> m = this.createFromArgsTest();
        Map<String, String> m2Expected = this.createFromArgsTest();
        /*
         * Call method under test
         */
        boolean mKey = m.hasKey("2");
        boolean mExpected = false;
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(mExpected, mKey);
        assertEquals(m2Expected, m);
    }
}

```