```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;
    }

    // CONSTRUCTORS ------------------------------------------------------------

    /**
     * Test for no argument constructor case.
     */
    @Test
    public final void noArgConstructorTest() {
        // Setup
        Map<String, String> m = this.createFromArgsTest();
        Map<String, String> mExpected = this.createFromArgsRef();

        // Eval
        assertEquals(mExpected, m);
    }

    /**
     * Test for single pair argument constructor case.
     */
    @Test
    public final void singleArgConstructorTest() {
        // Setup
        Map<String, String> m = this.createFromArgsTest("1", "a");
        Map<String, String> mExpected = this.createFromArgsRef("1", "a");

        // Eval
        assertEquals(mExpected, m);
    }

    /**
     * Test for 2 pairs argument constructor case.
     */
    @Test
    public final void twoArgConstructorTest() {
        // Setup
        Map<String, String> m = this.createFromArgsTest("1", "a", "2", "b");
        Map<String, String> mExpected = this.createFromArgsRef("1", "a", "2",
                "b");

        // Eval
        assertEquals(mExpected, m);
    }

    // KERNEL ---------------------------------------------------------------
    // add ------------------------------------------------------------------
    /**
     * Test adding a single k v pair to an empty map.
     */
    @Test
    public final void testAddSinglePairToEmpty() {
        // Setup
        Map<String, String> m = this.createFromArgsTest();
        Map<String, String> mExpected = this.createFromArgsRef("1", "a");
        // Call
        m.add("1", "a");
        // Eval
        assertEquals(mExpected, m);
    }

    // remove ---------------------------------------------------------------
    /**
     * Test removing a single k v pair resulting in empty map.
     */
    @Test
    public final void testRemoveSinglePairToEmpty() {
        // Setup
        Map<String, String> m = this.createFromArgsTest("1", "a");
        Map<String, String> mExpected = this.createFromArgsRef();
        Map.Pair<String, String> pair;
        // Unable to declare a map.pair object without simplepair
        //  messy workaround
        Map<String, String> pairExpected = this.createFromArgsRef("1", "a");
        // Call
        pair = m.remove("1");
        // Eval
        assertEquals(pairExpected.remove("1"), pair);
        assertEquals(mExpected, m);
    }

    // removeAny ------------------------------------------------------------
    /**
     * Test removing any k v pair
     */
    @Test
    public final void testRemoveAny() {
        // Setup
        Map<String, String> m = this.createFromArgsTest("1", "a", "1", "b");
        Map<String, String> mExpected = this.createFromArgsRef("1", "a");
        Map<String, String> mExpected2 = this.createFromArgsRef("1", "b");
        //create boolean to check if the pair removed was in m
        boolean goodRemove = false;
        // Call
        m.removeAny();
        if (m.equals(mExpected) || m.equals(mExpected2)) {
            goodRemove = true;
        }
        // Eval
        assertEquals(true, goodRemove);
    }
    // value -----------------------------------------------------------------
    /**
     * Test reporting the value v at key k.
     */
    @Test
    public final void testValue() {
        // Setup
        Map<String, String> m = this.createFromArgsRef("1", "a");
        // Call
        String value = m.value("1");
        // Eval
        assertEquals("a", value);
    }
    // hasKey ----------------------------------------------------------------
    /**
     * Test removing a single k v pair resulting in empty map.
     */
    @Test
    public final void testHasKey() {
        // Setup
        Map<String, String> m = this.createFromArgsRef("1", "a");
        // Call
        boolean trueKey = m.hasKey("1");
        boolean falseKey = m.hasKey("2");
        // Eval
        assertEquals(true, trueKey);
        assertEquals(false, falseKey);
    }
    // size ------------------------------------------------------------------
    /**
     * Test reporting the size of a map.
     */
    @Test
    public final void testMapSize() {
        // Setup
        Map<String, String> m0 = this.createFromArgsRef();
        Map<String, String> m1 = this.createFromArgsRef("1", "a");
        Map<String, String> m2 = this.createFromArgsRef("1", "a", "2", "b");
        int zeroRef = 0;
        int oneRef = 1;
        int twoRef = 2;

        // Call
        int zero = m0.size();
        int one = m1.size();
        int two = m2.size();

        // Eval
        assertEquals(zeroRef, zero);
        assertEquals(oneRef, one);
        assertEquals(twoRef, two);
    }


    /**
     * Test size = 0;
     */
    @Test
    public final void testSizeEmpty() {
        // Setup
        Map<String, String> m = this.createFromArgsTest();
        Map<String, String> mExpected = this.createFromArgsRef();
        // Call
        // Eval
        assertEquals(mExpected.size(), m.size());
    }

    /**
     * Test size = 1;
     */
    @Test
    public final void testSize1() {
        // Setup
        Map<String, String> m = this.createFromArgsTest("1", "a");
        Map<String, String> mExpected = this.createFromArgsRef("1", "a");
        // Call
        // Eval
        assertEquals(mExpected.size(), m.size());
    }

    /**
     * Test size = 0 after -1 using remove();
     */
    @Test
    public final void testSize1minus1() {
        // Setup
        Map<String, String> m = this.createFromArgsTest("1", "a");
        Map<String, String> mExpected = this.createFromArgsRef();
        // Call
        m.remove("1");
        // Eval
        assertEquals(mExpected.size(), m.size());
    }

    /**
     * Test size = 0 after -1 using removeAny();
     */
    @Test
    public final void testSize1minusAny() {
        // Setup
        Map<String, String> m = this.createFromArgsTest("1", "a");
        Map<String, String> mExpected = this.createFromArgsRef();
        // Call
        m.removeAny();
        // Eval
        assertEquals(mExpected.size(), m.size());
    }

```

```
}

```