```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;
    }

    // TODO - add test cases for constructor, add, remove, removeAny, value, hasKey, and size
    /**
     * Test for constructor with an empty map.
     */
    @Test
    public void constructorTestEmpty() {
        Map<String, String> map = this.constructorTest();
        Map<String, String> mapExp = this.constructorRef();

        assertEquals(mapExp, map);
    }

    /**
     * Test constructor with non empty map.
     */
    @Test
    public void constructorTestOne() {
        Map<String, String> map = this.constructorTest();
        map.add("hi", "bye");
        Map<String, String> mapExp = this.createFromArgsRef("hi", "bye");

        assertEquals(mapExp, map);
    }

    /**
     * Test add with empty map
     */
    @Test
    public void testAddEmpty() {
        Map<String, String> map = this.constructorTest();
        Map<String, String> mapExp = this.createFromArgsRef();

        assertEquals(mapExp, map);
    }

```

```java
    /**
     * Test add with one pair
     */
    @Test
    public void testAddOne() {
        Map<String, String> map = this.constructorTest();
        map.add("hi", "bye");
        Map<String, String> mapExp = this.createFromArgsRef("hi", "bye");

        assertEquals(mapExp, map);
    }

    /**
     * Test remove with one pair
     */
    @Test
    public void testRemoveOne() {
        Map<String, String> map = this.createFromArgsTest("hi", "bye");
        map.add("hi", "bye");
        Pair<String, String> pair = map.remove("hi");

        Map<String, String> mapExp = this.createFromArgsRef("hi", "bye");
        Pair<String, String> pairExp = mapExp.remove("hi");
        assertEquals(pairExp, pair);
        assertEquals(mapExp, map);
    }

    /**
     * Test remove with two pairs
     */
    @Test
    public void testRemoveTwo() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        map.add("c", "d");
        map.remove("a");
        map.remove("c");
        Map<String, String> mapExp = this.createFromArgsRef("hi", "bye");
        mapExp.remove("hi");
        assertEquals(mapExp, map);
    }

    /**
     * Test for removeAny on map of length one.
     */
    @Test
    public void testRemoveAnyOne() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        Pair<String, String> rem = map.removeAny();
        Map<String, String> mapExp = this.createFromArgsRef("a", "b");
        Pair<String, String> remExp = mapExp.removeAny();
        assertEquals(remExp, rem);
        assertEquals(mapExp, map);
    }

    /**
```

```java
     * Test for removeAny on map of length two.
     */
    @Test
    public void testRemoveAnyTwo() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        map.add("c", "d");
        Pair<String, String> rem = map.removeAny();
        Map<String, String> mapExp = this.createFromArgsRef("a", "b", "c", "d");
        assertEquals(true, mapExp.hasKey(rem.key()));
        assertEquals(mapExp, map);
    }

    /**
     * Test for value on map of length one.
     */
    @Test
    public void testValueOne() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        Pair<String, String> rem = map.removeAny();
        String val = rem.value();
        Map<String, String> mapExp = this.createFromArgsRef("a", "b");
        Pair<String, String> remExp = mapExp.removeAny();
        String valExp = remExp.value();
        assertEquals(valExp, val);
        assertEquals(mapExp, map);
    }

    /**
     * Test for value on map of length two.
     */
    @Test
    public void testValueTwo() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        map.add("c", "d");
        Pair<String, String> rem = map.remove("a");
        String val = rem.value();
        Map<String, String> mapExp = this.createFromArgsRef("a", "b", "c", "d");
        Pair<String, String> remExp = mapExp.remove("a");
        String valExp = remExp.value();
        assertEquals(valExp, val);
        assertEquals(mapExp, map);
    }

    /**
     * Test for hasKey one map of length one.
     */
    @Test
    public void testHasKeyOne() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        Map<String, String> mapExp = this.createFromArgsRef("a", "b");
        assertEquals(true, map.hasKey("a"));
        assertEquals(mapExp, map);
    }

    /**
     * Test for hasKey one map of length two.
     */
    @Test
    public void testHasKeyTwo() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        map.add("c", "d");
        boolean key = false;
        Map<String, String> mapExp = this.createFromArgsRef("a", "b", "c", "d");
        if (map.hasKey("a") || map.hasKey("c")) {
            key = true;
        }
        assertEquals(true, key);
        assertEquals(mapExp, map);
    }

    /**
     * Test for size on empty map.
     */
    @Test
    public void testSizeEmpty() {
        Map<String, String> map = this.constructorTest();
        Map<String, String> mapExp = this.createFromArgsRef();
        assertEquals(mapExp.size(), map.size());
        assertEquals(0, map.size());
        assertEquals(mapExp, map);
    }

    /**
     * Test for size on map of length one.
     */
    @Test
    public void testSizeOne() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        Map<String, String> mapExp = this.createFromArgsRef("a", "b");
        assertEquals(mapExp.size(), map.size());
        assertEquals(1, map.size());
        assertEquals(mapExp, map);
    }

    /**
     * Test for size on map of length three.
     */
    @Test
    public void testSizeTwo() {
        Map<String, String> map = this.constructorTest();
        map.add("a", "b");
        map.add("c", "d");
        map.add("e", "f");
        Map<String, String> mapExp = this.createFromArgsRef("a", "b", "c", "d", "e", "f");
        assertEquals(mapExp.size(), map.size());
        assertEquals(3, map.size());
        assertEquals(mapExp, map);
    }
```

```

}

```