```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
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 David P. & Zach B.
 *
 */
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

    /**
     * Tests adding to an initially empty map.
     */
    @Test
    public void testAddToEmptyMap() {
        Map<String, String> map = this.createFromArgsTest();
        Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A");
        map.add("Alice", "A");
        assertEquals(map, expectedMap);
    }

    /**
     * Tests adding a new pair to a non-empty map.
     */
    @Test
    public void testAddPairToNonEmptyMap() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
                "B");
        Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A",
                "Bob", "B", "Charlie", "C");
        map.add("Charlie", "C");
        assertEquals(map, expectedMap);
    }

    /**
     * Tests adding multiple pairs to a map.
     */
    @Test
    public void testAddMultiplePairs() {
        Map<String, String> map = this.createFromArgsTest("Charlie", "C");
        Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A",
                "Charlie", "C", "Bob", "B");
        map.add("Bob", "B");
        map.add("Alice", "A");
        assertEquals(map, expectedMap);
    }

    /**
     * Tests removing the only pair in the map.
     */
    @Test
    public void testRemovePairLeavingEmptyMap() {
        Map<String, String> map = this.createFromArgsTest("David", "D");
        Map<String, String> expectedMap = this.createFromArgsRef();
        map.remove("David");
        assertEquals(map, expectedMap);
    }

    /**
     * Tests removing a pair from a non-empty map.
     */
    @Test
    public void testRemovePairFromNonEmptyMap() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
                "B", "David", "D");
        Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A",
                "Bob", "B");
        map.remove("David");
        assertEquals(map, expectedMap);
    }

    /**
     * Tests removing multiple pairs from a non-empty map.
     */
    @Test
    public void testRemoveMultiplePairs() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
                "B", "David", "D");
        Map<String, String> expectedMap = this.createFromArgsRef("Bob", "B");
        map.remove("David");
        map.remove("Alice");
        assertEquals(map, expectedMap);
    }

    /**
     * Tests removing any pair from a map with multiple pairs.
     */
    @Test
    public void testRemoveAnyPairFromMultiple() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
                "B", "David", "D");
        Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A",
                "Bob", "B", "David", "D");
```

```java
        Map.Pair<String, String> removedPair = map.removeAny();
        assertTrue(expectedMap.hasKey(removedPair.key()));
        expectedMap.remove(removedPair.key());
        assertEquals(map, expectedMap);
    }

    /**
     * Tests removing any pair from a map with a single pair.
     */
    @Test
    public void testRemoveAnyPairFromSingle() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A");
        Map<String, String> expectedMap = this.createFromArgsRef("Alice", "A");
        Map.Pair<String, String> removedPair = map.removeAny();
        assertTrue(expectedMap.hasKey(removedPair.key()));
        expectedMap.remove(removedPair.key());
        assertEquals(map, expectedMap);
    }

    /**
     * Tests retrieving the value associated with a key in a single-pair map.
     */
    @Test
    public void testGetValueSinglePair() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A");
        String value = map.value("Alice");
        String expectedValue = "A";
        assertEquals(value, expectedValue);
    }

    /**
     * Tests retrieving the value associated with a key in a multiple-pair map.
     */
    @Test
    public void testGetValueMultiplePairs() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
                "B", "Charlie", "C");
        String value = map.value("Bob");
        String expectedValue = "B";
        assertEquals(value, expectedValue);
    }

    /**
     * Tests checking the presence of a key in a single-pair map.
     */
    @Test
    public void testHasKeyInSinglePair() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A");
        boolean hasKey = map.hasKey("Alice");
        assertTrue(hasKey);
    }

    /**
     * Tests checking the presence of keys in a multiple-pair map.
     */
    @Test
    public void testHasKeyInMultiplePairs() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
                "B", "Charlie", "C");
        assertTrue(map.hasKey("Alice"));
        assertTrue(map.hasKey("Bob"));
        assertTrue(map.hasKey("Charlie"));
    }

    /**
     * Tests checking the absence of a key in a map.
     */
    @Test
    public void testHasKeyNotFound() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
                "B", "Charlie", "C");
        boolean hasKey = map.hasKey("David");
        assertFalse(hasKey);
    }

    /**
     * Tests checking the absence of a key in an empty map.
     */
    @Test
    public void testHasKeyInEmptyMap() {
        Map<String, String> map = this.createFromArgsTest();
        boolean hasKey = map.hasKey("Alice");
        assertFalse(hasKey);
    }

    /**
     * Tests checking the absence of keys in a map with multiple pairs.
     */
    @Test
    public void testHasKeyMultipleNotFound() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
                "B", "Charlie", "C");
        assertFalse(map.hasKey("David"));
        assertFalse(map.hasKey("Eve"));
    }

    /**
     * Tests the size of an empty map.
     */
    @Test
    public void testSizeOfEmptyMap() {
        Map<String, String> map = this.createFromArgsTest();
        int size = map.size();
        int expectedSize = 0;
        assertEquals(size, expectedSize);
    }

    /**
     * Tests the size of a map with a single pair.
     */
    @Test
    public void testSizeOfSinglePairMap() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A");
        int size = map.size();
        int expectedSize = 1;
        assertEquals(size, expectedSize);
    }

    /**
     * Tests the size of a map with multiple pairs.
     */
    @Test
    public void testSizeOfMultiplePairsMap() {
        Map<String, String> map = this.createFromArgsTest("Alice", "A", "Bob",
                "B", "Charlie", "C");
        int size = map.size();
        int expectedSize = 3;
        assertEquals(size, expectedSize);
    }
}

```