```java
import static org.junit.Assert.assertEquals;

/**
 * JUnit test fixture for {@code Map<String, String>}'s constructor and kernel
 * methods.
 *
 * @author Qinuo Shi & Yiming Cheng
 *
 */
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 without elements.
     */
    @Test
    public void testDefaultConstructor() {
        Map<String, String> t = this.constructorTest();
        Map<String, String> texpected = this.constructorRef();

        assertEquals(t, texpected);
    }

    /**
     * test for constructor with elements.
     */
    @Test
    public void testConstructorOne() {
        Map<String, String> t = this.createFromArgsTest("q", "0");
        Map<String, String> texpected = this.createFromArgsRef("q", "0");

        assertEquals(t, texpected);
    }

    /**
     * test for adding one pair of elements to constructor without elements.
     */
    @Test
    public void testAddToNew() {
        Map<String, String> t = this.createFromArgsTest();
        Map<String, String> texpected = this.createFromArgsRef("q", "0");

        t.add("q", "0");

        assertEquals(t, texpected);
```

```java
    }

    /**
     * test for constructor with more elements.
     */
    @Test
    public void testConstructorElements() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
                "2", "r", "3", "t", "4");
        Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
                "1", "e", "2", "r", "3", "t", "4");

        assertEquals(t, texpected);
    }

    /**
     * test for add one pair of elements to constructor with elements.
     */
    @Test
    public void testAddOne() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1");
        Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
                "1", "e", "2");

        t.add("e", "2");

        assertEquals(t, texpected);
    }

    /**
     * test for add some pairs of elements to constructor without elements.
     */
    @Test
    public void testAddMany() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1");
        Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
                "1", "e", "2", "r", "3", "t", "4");

        t.add("e", "2");
        t.add("r", "3");
        t.add("t", "4");

        assertEquals(t, texpected);
    }

    /**
     * test for remove elements from constructors with elements.
     */
    @Test
    public void testRemoveZero() {
        Map<String, String> t = this.createFromArgsTest("q", "0");
        Map<String, String> texpected = this.createFromArgsRef();

        Map.Pair<String, String> one = t.remove("q");

        assertEquals(t, texpected);
        assertEquals(one.key(), "q");
        assertEquals(one.value(), "0");
    }

    /**
     * test for remove elements from constructors with elements.
     */
    @Test
    public void testRemove() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
                "2", "r", "3");
        Map<String, String> texpected = this.createFromArgsRef("q", "0");

        Map.Pair<String, String> two = t.remove("w");
        Map.Pair<String, String> three = t.remove("e");
        Map.Pair<String, String> four = t.remove("r");

        assertEquals(t, texpected);
        assertEquals(two.key(), "w");
        assertEquals(two.value(), "1");
        assertEquals(three.key(), "e");
        assertEquals(three.value(), "2");
        assertEquals(four.key(), "r");
        assertEquals(four.value(), "3");
    }

    /**
     * test for remove elements from constructors with elements.
     */
    @Test
    public void testRemoveMany() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
                "2", "r", "3", "t", "4");
        Map<String, String> texpected = this.createFromArgsRef("w", "1", "e",
                "2", "r", "3", "t", "4");

        Map.Pair<String, String> one = t.remove("q");

        assertEquals(t, texpected);
        assertEquals(one.key(), "q");
        assertEquals(one.value(), "0");
    }

    /**
     * test for remove any pair of elements.
     */
    @Test
    public void testRemoveToZero() {
        Map<String, String> t = this.createFromArgsTest("q", "0");
        Map<String, String> texpected = this.createFromArgsRef("q", "0");

        Map.Pair<String, String> one = t.removeAny();

        Map.Pair<String, String> oneexpected = texpected.remove(one.key());
        assertEquals(t, texpected);
        assertEquals(one.key(), oneexpected.key());
        assertEquals(one.value(), oneexpected.value());
    }
```

```java

    /**
     * test for remove any pair of elements.
     */
    @Test
    public void testRemoveAny() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
                "2", "r", "3", "t", "4");
        Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
                "1", "e", "2", "r", "3", "t", "4");

        Map.Pair<String, String> one = t.removeAny();

        Map.Pair<String, String> oneexpected = texpected.remove(one.key());
        assertEquals(t, texpected);
        assertEquals(one.key(), oneexpected.key());
        assertEquals(one.value(), oneexpected.value());
    }

    /**
     * test for one value in the constructor.
     */
    @Test
    public void testValueO() {
        Map<String, String> t = this.createFromArgsTest("q", "0");
        Map<String, String> texpected = this.createFromArgsRef("q", "0");

        String string = t.value("q");
        String stringexpected = texpected.value("q");

        assertEquals(t, texpected);
        assertEquals(string, "0");
        assertEquals(stringexpected, "0");
    }

    /**
     * test for one value in the constructor.
     */
    @Test
    public void testValueM() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
                "2", "r", "3", "t", "4");
        Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
                "1", "e", "2", "r", "3", "t", "4");

        String string = t.value("q");
        String stringexpected = texpected.value("q");

        assertEquals(t, texpected);
        assertEquals(string, "0");
        assertEquals(stringexpected, "0");
    }

    /**
     * test for some values in the constructor.
     */
    @Test
    public void testValueS() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
                "2");
        Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
                "1", "e", "2");

        String string = t.value("q");
        String stringexpected = texpected.value("q");
        String string2 = t.value("w");
        String stringexpected2 = texpected.value("w");

        assertEquals(t, texpected);
        assertEquals(string, "0");
        assertEquals(stringexpected, "0");
        assertEquals(string2, "1");
        assertEquals(stringexpected2, "1");
    }

    /**
     * test for hasKey with no element.
     */
    @Test
    public void testHasKeyZero() {
        Map<String, String> t = this.constructorTest();
        Map<String, String> texpected = this.constructorRef();

        boolean b = t.hasKey("q");
        boolean bexp = texpected.hasKey("q");

        assertEquals(t, texpected);
        assertEquals(b, false);
        assertEquals(bexp, false);
    }

    /**
     * test for hasKey with one element, true.
     */
    @Test
    public void testHasKeyOneTrue() {
        Map<String, String> t = this.createFromArgsTest("q", "0");
        Map<String, String> texpected = this.createFromArgsRef("q", "0");

        boolean b = t.hasKey("q");
        boolean bexp = texpected.hasKey("q");

        assertEquals(t, texpected);
        assertEquals(b, true);
        assertEquals(bexp, true);
    }

    /**
     * test for hasKey with one element, false.
     */
    @Test
    public void testHasKeyOneFalse() {
        Map<String, String> t = this.createFromArgsTest("q", "0");
        Map<String, String> texpected = this.createFromArgsRef("q", "0");

        boolean b = t.hasKey("x");
        boolean bexp = texpected.hasKey("x");

        assertEquals(t, texpected);
        assertEquals(b, false);
        assertEquals(bexp, false);
    }

    /**
     * test for hasKey with more elements, true.
     */
    @Test
    public void testHasKeyManyTrue() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
                "2", "r", "3");
        Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
                "1", "e", "2", "r", "3");

        boolean b = t.hasKey("q");
        boolean bexp = texpected.hasKey("q");

        assertEquals(t, texpected);
        assertEquals(b, true);
        assertEquals(bexp, true);
    }

    /**
     * test for hasKey with more elements, false.
     */
    @Test
    public void testHasKeyManyFalse() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
                "2", "r", "3");
        Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
                "1", "e", "2", "r", "3");

        boolean b = t.hasKey("o");
        boolean bexp = texpected.hasKey("o");

        assertEquals(t, texpected);
        assertEquals(b, false);
        assertEquals(bexp, false);
    }

    /**
     * test for 0 size.
     */
    @Test
    public void testSizeZero() {
        Map<String, String> t = this.createFromArgsTest();
        Map<String, String> texpected = this.createFromArgsRef();

        int size = t.size();
        int sizeexp = texpected.size();

        assertEquals(t, texpected);
```

```java
        assertEquals(size, 0);
        assertEquals(sizeexp, 0);
    }

    /**
     * test for size 1 .
     */
    @Test
    public void testSizeOne() {
        Map<String, String> t = this.createFromArgsTest("q", "0");
        Map<String, String> texpected = this.createFromArgsRef("q", "0");

        int size = t.size();
        int sizeexp = texpected.size();

        assertEquals(t, texpected);
        assertEquals(size, 1);
        assertEquals(sizeexp, 1);
    }

    /**
     * test for larger size.
     */
    @Test
    public void testSizeMany() {
        Map<String, String> t = this.createFromArgsTest("q", "0", "w", "1", "e",
                "2", "r", "3");
        Map<String, String> texpected = this.createFromArgsRef("q", "0", "w",
                "1", "e", "2", "r", "3");

        int size = t.size();
        int sizeexp = texpected.size();

        assertEquals(t, texpected);
        assertEquals(size, 4);
        assertEquals(sizeexp, 4);
    }
}
```