```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();

    final String a = "a", b = "b", c = "c", d = "d", e = "e", f = "f";

    /**
     * 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
    public void testAdd() {

        Map<String, String> map1 = this.createFromArgsRef(this.a, this.b,
                this.c, this.d);
        Map<String, String> map2 = this.createFromArgsRef(this.a, this.b,
                this.c, this.d, this.e, this.f);
        map1.add(this.c, this.f);
        assertEquals(map2, map1);
    }

    @Test
    public void testRemove() {

        Map<String, String> map1 = this.createFromArgsRef(this.a, this.b,
                this.c, this.d);
        Map<String, String> map2 = this.createFromArgsRef(this.a, this.b,
                this.c, this.d, this.e, this.f);
        map2.remove(this.e);
        assertEquals(map1, map2);
    }

    @Test
    public void testRemoveAny() {
        Map<String, String> map1 = this.createFromArgsRef(this.a, this.b,
                this.c, this.d);
        Map<String, String> map2 = this.createFromArgsRef();
        while (map1.size() != 0) {
            map1.removeAny();
        }
        assertEquals(map2, map1);
    }

    @Test
    public void testValue() {
        Map<String, String> map = this.createFromArgsRef(this.a, this.b, this.c,
                this.d);
        String test = map.value(this.c);
        assertEquals(this.b, test);
    }

    @Test
    public void testHasKey() {
        Map<String, String> map = this.createFromArgsRef(this.a, this.b, this.c,
                this.d);
        boolean test = map.hasKey(this.a);
        assertEquals(true, test);
    }

    @Test
    public void testSize() {
        Map<String, String> map = this.createFromArgsRef(this.a, this.b, this.c,
                this.d, this.e, this.f);
        int test = map.size();
        assertEquals(3, test);
    }

    /*
     * m = {}; m = {"one", 1} m = {("one" , 1),("zero", 0)} m = {("one" ,
     * 1),("zero", 0),("negative one", -1)} m = {("one" , 1),("negative one",
     * -1)} p = {"zero", 0} m = {("negative one", -1)} p = {"zero", 0} m =
     * {("negative one", -1), ("cipher", 0)} p = {"zero", 0} m =
     * {("negative one", -1), ("cipher", 0), ("zero", 0)} p = {"zero", 0} m =
     * {("cipher", 0), ("zero", 0)} p = {"zero", 0} m = {("zero", 0)} p =
     * {"zero", 0} m = {} p = {"zero", 0}
     */

}
```