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

import org.junit.Test;

import components.map.Map;
import components.map.Map1L;
import components.queue.Queue;
import components.queue.Queue1L;
import components.set.Set;
import components.set.Set1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

public class GlossaryTest {

    /*
     * Tests for getElements
     */

    // Routine case with strings and one line definitions
    @Test
    public void getElementsTest1() {
        Map<String, String> elementMap = new Map1L<>();
        Map<String, String> expectedMap = new Map1L<>();
        expectedMap.add("word1", "def1");
        expectedMap.add("word2", "def2");
        expectedMap.add("word3", "def3");
        Queue<String> elementQueue = new Queue1L<>();
        Queue<String> expectedQueue = new Queue1L<>();
        expectedQueue.enqueue("word1");
        expectedQueue.enqueue("word2");
        expectedQueue.enqueue("word3");

        SimpleReader in = new SimpleReader1L("test/getElementsTest1.html");
        Glossary.getElements(elementMap, elementQueue, in);
        assertEquals(elementMap, expectedMap);
        assertEquals(elementQueue, expectedQueue);

        in.close();
    }

    // Challenging case with symbols, numbers, multi-line definitions,
    // and extra blank spaces
    @Test
    public void getElementsTest2() {
        Map<String, String> elementMap = new Map1L<>();
        Map<String, String> expectedMap = new Map1L<>();
        expectedMap.add("word", "this is one long def ");
        expectedMap.add("110100101010111", "@#$%^*()(*^%$#!@#$%^*)");
        Queue<String> elementQueue = new Queue1L<>();
        Queue<String> expectedQueue = new Queue1L<>();
        expectedQueue.enqueue("word");
        expectedQueue.enqueue("110100101010111");

        SimpleReader in = new SimpleReader1L("test/getElementsTest2.html");
        Glossary.getElements(elementMap, elementQueue, in);
        assertEquals(elementMap, expectedMap);
        assertEquals(elementQueue, expectedQueue);

        in.close();
    }

    /*
     * Tests for outputHeader
     */

    // Routine case with just strings
    @Test
    public void outputHeaderTest1() {
        SimpleReader inExpected = new SimpleReader1L(
                "test/expectedHeaderEasy.html");
        SimpleReader inActual = new SimpleReader1L("test/actualHeader.html");
        SimpleWriter out = new SimpleWriter1L("test/actualHeader.html");
        Queue<String> wordQ = new Queue1L<>();
        wordQ.enqueue("Oranges");
        wordQ.enqueue("Apples");
        wordQ.enqueue("Tomato");
        Glossary.outputHeader(wordQ, out);
        Set<String> expectedSet = new Set1L<>();
        Set<String> actualSet = new Set1L<>();

        while (!inExpected.atEOS()) {
            expectedSet.add(inExpected.nextLine());
        }
        while (!inActual.atEOS()) {
            actualSet.add(inActual.nextLine());
        }

        assertEquals(expectedSet, actualSet);

        inExpected.close();
        inActual.close();
    }

    // Routine case with just strings
    @Test
    public void outputHeaderTest2() {
        SimpleReader inExpected = new SimpleReader1L(
                "test/expectedHeaderHard.html");
        SimpleReader inActual = new SimpleReader1L("test/actualHeader.html");
        SimpleWriter out = new SimpleWriter1L("test/actualHeader.html");
        Queue<String> wordQ = new Queue1L<>();
        wordQ.enqueue("10010101001111");
        wordQ.enqueue("Hello-there");
        wordQ.enqueue("#$%^&*()12345678");
        Glossary.outputHeader(wordQ, out);
        Set<String> expectedSet = new Set1L<>();
        Set<String> actualSet = new Set1L<>();

        while (!inExpected.atEOS()) {
            expectedSet.add(inExpected.nextLine());
        }
        while (!inActual.atEOS()) {
            actualSet.add(inActual.nextLine());
        }

        assertEquals(expectedSet, actualSet);

        inExpected.close();
        inActual.close();
    }

    /*
     * Test for processItem
     */

    // Routine case
    @Test
    public void processItemTest1() {
        SimpleReader inExpected = new SimpleReader1L(
                "test/expectedProcess.html");
        SimpleReader inActual = new SimpleReader1L("test/actualProcess.html");
        SimpleWriter out = new SimpleWriter1L("test/actualProcess.html");

        Set<Character> separators = new Set1L<>();
        separators.add(' ');
        separators.add(',');
        String word = "harvest";
        String def = "the process or period of gathering in crops";
        Map<String, String> pairMap = new Map1L<>();
        pairMap.add(word, def);
        Glossary.processItem(word, def, out, separators, pairMap);

        Set<String> expectedSet = new Set1L<>();
        Set<String> actualSet = new Set1L<>();
        while (!inExpected.atEOS()) {
            expectedSet.add(inExpected.nextLine());
        }
        while (!inActual.atEOS()) {
            actualSet.add(inActual.nextLine());
        }
        assertEquals(expectedSet, actualSet);

        inExpected.close();
        inActual.close();
    }

    // Routine case with a linking term
    @Test
    public void processItemTest2() {
        SimpleReader inExpected = new SimpleReader1L(
                "test/expectedProcess2.html");
        SimpleReader inActual = new SimpleReader1L("test/actualProcess.html");
        SimpleWriter out = new SimpleWriter1L("test/actualProcess.html");

        Set<Character> separators = new Set1L<>();
        separators.add(' ');
        separators.add(',');
        String word = "harvest";
        String def = "the process or period of gathering in crops";
        Map<String, String> pairMap = new Map1L<>();
        pairMap.add(word, def);
        pairMap.add("crops", "a cultivated plant");
        Glossary.processItem(word, def, out, separators, pairMap);

        Set<String> expectedSet = new Set1L<>();
        Set<String> actualSet = new Set1L<>();
        while (!inExpected.atEOS()) {
            expectedSet.add(inExpected.nextLine());
        }
        while (!inActual.atEOS()) {
            actualSet.add(inActual.nextLine());
        }

        assertEquals(expectedSet, actualSet);

        inExpected.close();
        inActual.close();
    }

    /*
     * Tests for nextWordOrSeparator
     */

    // Routine case
    @Test
    public void nextWordOrSeparatorTest1() {
        Set<Character> separators = new Set1L<>();
        separators.add(' ');
        String text = "Hello there";
        int position = 0;
        String next = Glossary.nextWordOrSeparator(text, position, separators);
        String expectedNext = "Hello";
        assertEquals(next, expectedNext);
    }

    // Routine case with different separator
    @Test
    public void nextWordOrSeparatorTest2() {
        Set<Character> separators = new Set1L<>();
        separators.add(' ');
        separators.add('/');
        String text = "10110101/2 = 5055050.5";
        int position = 0;
        String next = Glossary.nextWordOrSeparator(text, position, separators);
        String expectedNext = "10110101";
        assertEquals(next, expectedNext);
    }

    // Testing return if separator is the first char
    @Test
    public void nextWordOrSeparatorTest3() {
        Set<Character> separators = new Set1L<>();
        separators.add('.');
        String text = ".TheFirstCharacterIsASeparator";
        int position = 0;
        String next = Glossary.nextWordOrSeparator(text, position, separators);
        String expectedNext = ".";
        assertEquals(next, expectedNext);
    }

    // Challenging case
    @Test
    public void nextWordOrSeparatorTest4() {
        Set<Character> separators = new Set1L<>();
        separators.add('.');
        separators.add(' ');
        String text = "!@#$%^&*(1234)-=~TheseAreSpecialSymbols and numbers";
        int position = 0;
        String next = Glossary.nextWordOrSeparator(text, position, separators);
        String expectedNext = "!@#$%^&*(1234)-=~TheseAreSpecialSymbols";
        assertEquals(next, expectedNext);
    }

    /*
     * Test for outputHeader, only one because it's a straightforward method and
     * parameter has no effect on method besides where it prints
     */

    // Routine test case
    @Test
    public void outputFooterTest1() {
        SimpleReader inExpected = new SimpleReader1L(
                "test/expectedFooter.html");
        SimpleReader inActual = new SimpleReader1L("test/actualFooter.html");
        SimpleWriter out = new SimpleWriter1L("test/actualFooter.html");
        Glossary.outputFooter(out);

        Set<String> expectedSet = new Set1L<>();
        Set<String> actualSet = new Set1L<>();
        while (!inExpected.atEOS()) {
            expectedSet.add(inExpected.nextLine());
        }
        while (!inActual.atEOS()) {
            actualSet.add(inActual.nextLine());
        }

        assertEquals(expectedSet, actualSet);
        inExpected.close();
        inActual.close();
    }

}
```