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

import java.util.Iterator;

import org.junit.Test;

import components.map.Map;
import components.map.Map.Pair;
import components.map.Map1L;
import components.set.Set;
import components.set.Set1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;

public class GlossarySearchTest {

    /*
     * Test for complete (correct) list of terms + definition
     */

    @Test
    public void getTerms_termsTxt() {

        //create expected map
        Map<String, String> expectedMap = new Map1L<>();
        expectedMap.add("meaning",
                "something that one wishes to convey, especially by language");
        expectedMap.add("term", "a word whose definition is in a glossary");
        expectedMap.add("word",
                "a string of characters in a language, which has at least one character");
        expectedMap.add("definition",
                "a sequence of words that gives meaning to a term");
        expectedMap.add("glossary",
                "a list of difficult or specialized terms, with their definitions usually near the end of a book");
        expectedMap.add("language",
                "a set of strings of characters, each of which has meaning");
        expectedMap.add("book", "a printed or written literary work");

        //get actual map
        SimpleReader testFile = new SimpleReader1L("terms.txt");
        Map<String, String> actualMap = GlossarySearch.getTerm(testFile);

        //compare the two and return whether they are equal or not
        boolean areEqual = compareMaps(expectedMap, actualMap);

        //run assert to check if true
        assertEquals(true, areEqual);

        //close streams
        testFile.close();
    }

    //function for checking if maps are ==
    public static boolean compareMaps(Map<String, String> expected,
            Map<String, String> actual) {

```

```
57          boolean isEqual = true;
58
59          if (expected.size() == actual.size()) {
60
61              Iterator<Map.Pair<String, String>> iter = actual.iterator();
62              while (iter.hasNext()) {
63                  Pair<String, String> p = iter.next();
64
65                  if (!expected.hasKey(p.key())) {
66                      isEqual = false;
67                  }
68              }
69
70          } else {
71              isEqual = false;
72          }
73
74          return isEqual;
75      }
76
77      /*
78       * Test for alphabetize
79       */
80
81      @Test
82      public void alphabetize_termsTxt() {
83
84          Set<String> expected = new Set1L<>();
85          expected.add("book");
86          expected.add("definition");
87          expected.add("glossary");
88          expected.add("language");
89          expected.add("meaning");
90          expected.add("term");
91          expected.add("word");
92
93          Map<String, String> expectedMap = new Map1L<>();
94          expectedMap.add("meaning",
95                  "something that one wishes to convey, especially by language");
96          expectedMap.add("term", "a word whose definition is in a glossary");
97          expectedMap.add("word",
98                  "a string of characters in a language, which has at least one character");
99          expectedMap.add("definition",
100                 "a sequence of words that gives meaning to a term");
101         expectedMap.add("glossary",
102                 "a list of difficult or specialized terms, with their definitions usually near
    the end of a book");
103         expectedMap.add("language",
104                 "a set of strings of characters, each of which has meaning");
105         expectedMap.add("book", "a printed or written literary work");
106
107         Set<String> actual = GlossarySearch.alphabetize(expectedMap);
108
109         assertEquals(expected, actual);
110     }
111
112     /*
```

```
113      * Test for correct separator set
114      */
115
116     @Test
117     public void generateElements_termsTxt() {
118
119         Set<Character> expected = new Set1L<>();
120         expected.add(' ');
121         expected.add(',');
122         expected.add('\t');
123
124         Set<Character> actual = new Set1L<>();
125         String test = " \t, ";
126
127         GlossarySearch.generateElements(test, actual);
128
129         assertEquals(expected, actual);
130     }
131
132     /*
133      * Test for nextWord functionality
134      */
135
136     @Test
137     public void nextWord1() {
138
139         String test = "This is a test";
140         int position = 0;
141         Set<Character> sep = new Set1L<>();
142         sep.add(' ');
143         sep.add(',');
144         sep.add('\t');
145
146         String actual = GlossarySearch.nextWord(test, position, sep);
147         String expected = "This";
148
149         assertEquals(expected, actual);
150     }
151
152     @Test
153     public void nextWord2() {
154
155         String test = "This is a test";
156         int position = 13;
157         Set<Character> sep = new Set1L<>();
158         sep.add(' ');
159         sep.add(',');
160         sep.add('\t');
161
162         String actual = GlossarySearch.nextWord(test, position, sep);
163         String expected = "t";
164
165         assertEquals(expected, actual);
166
167     }
168
169 }
```

170