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

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;

/**
 * JUnit test cases for the Glossary class methods. Author: Logan Oden
 */
public class GlossaryTest {

    /**
     * Challenging test case for the nextWordOrSeparator method. Tests when the
     * first character is a separator.
     */
    @Test
    public void testNextWordOrSeparator_Challenging() {
        Set<Character> separators = new Set1L<>();
        separators.add(' ');
        separators.add(',');

        String text = "Hello, world!";
        int position = 5;

        String result = Glossary.nextWordOrSeparator(text, position,
                separators);
        assertEquals(", ", result);
    }
```

```
38
39      /**
40       * Routine test case for the nextWordOrSeparator method.
  Tests when the
41       * first character is not a separator.
42       */
43      @Test
44      public void testNextWordOrSeparator_Routine() {
45          Set<Character> separators = new Set1L<>();
46          separators.add(' ');
47          separators.add(',');
48
49          String text = "Hello, world!";
50          int position = 0;
51
52          String result = Glossary.nextWordOrSeparator(text,
  position,
53                  separators);
54          assertEquals("Hello", result);
55      }
56
57      /**
58       * Challenging test case for the mapFromInputLines
  method. Tests with an
59       * input file containing unexpected data.
60       */
61      @Test
62      public void testMapFromInputLines_Challenging() {
63          SimpleReader input = new SimpleReader1L("data\
  \terms_test.txt");
64
65          Map<String, String> resultMap =
  Glossary.mapFromInputLines(input);
66
67          assertEquals(resultMap.hasKey("Java"), true);
68          assertEquals(resultMap.hasKey("key"), false);
69      }
70
71      /**
72       * Routine test case for the mapFromInputLines method.
```

```
   Tests with a standard
 73      * input file.
 74      */
 75     @Test
 76     public void testMapFromInputLines_Routine() {
 77         SimpleReader input = new SimpleReader1L("data\
   \terms_test.txt");
 78
 79         Map<String, String> expectedMap = new Map1L<>();
 80         expectedMap.add("Java", "A programming language.");
 81         expectedMap.add("JUnit", "A testing framework for
   Java.");
 82
 83         Map<String, String> resultMap =
   Glossary.mapFromInputLines(input);
 84
 85         assertEquals(expectedMap, resultMap);
 86     }
 87
 88     /**
 89      * Edge test case for the mapFromInputLines method. Tests
   with an empty
 90      * input file.
 91      */
 92     @Test
 93     public void testMapFromInputLines_Edge() {
 94         SimpleReader input = new SimpleReader1L("data\
   \empty_file.txt");
 95
 96         Map<String, String> resultMap =
   Glossary.mapFromInputLines(input);
 97
 98         assertTrue(resultMap.size() == 0);
 99     }
100
101     /**
102      * Challenging test case for the SortingKeys method.
   Sorting a map with
103      * entries in arbitrary order.
104      */
```

```
105     @Test
106     public void testSortingKeys_Challenging() {
107         Map<String, String> mapToSort = new Map1L<>();
108         mapToSort.add("C", "Definition of C");
109         mapToSort.add("A", "Definition of A");
110         mapToSort.add("B", "Definition of B");
111 
112         Queue<String> expectedQueue = new Queue1L<>();
113         expectedQueue.enqueue("A");
114         expectedQueue.enqueue("B");
115         expectedQueue.enqueue("C");
116 
117         Queue<String> resultQueue =
    Glossary.sortingKeys(mapToSort);
118 
119         assertEquals(expectedQueue, resultQueue);
120     }
121 
122     /**
123      * Routine test case for the SortingKeys method. Sorting
    a map with entries
124      * in alphabetical order.
125      */
126     @Test
127     public void testSortingKeys_Routine() {
128         Map<String, String> mapToSort = new Map1L<>();
129         mapToSort.add("Java", "A programming language.");
130         mapToSort.add("Python", "A high-level programming
    language.");
131         mapToSort.add("C++", "A general-purpose programming
    language.");
132 
133         Queue<String> expectedQueue = new Queue1L<>();
134         expectedQueue.enqueue("C++");
135         expectedQueue.enqueue("Java");
136         expectedQueue.enqueue("Python");
137 
138         Queue<String> resultQueue =
    Glossary.sortingKeys(mapToSort);
139 
```

```
140         assertEquals(expectedQueue, resultQueue);
141     }
142
143     /**
144      * Edge test case for the SortingKeys method. Sorting an
    empty map.
145      */
146     @Test
147     public void testSortingKeys_Edge() {
148         Map<String, String> mapToSort = new Map1L<>();
149
150         // Test with an empty map
151         Queue<String> resultQueue =
    Glossary.sortingKeys(mapToSort);
152
153         assertTrue(resultQueue.length() == 0);
154     }
155
156     /**
157      * Challenging test case for the processTerm method.
    Tests with a term
158      * containing special characters.
159      */
160     @Test
161     public void testProcessTerm_Challenging() {
162         String term = "Java@Programming!";
163         String definition = "A programming language.";
164
165         Map<String, String> glossary = new Map1L<>();
166         glossary.add(term, definition);
167         Queue<String> terms = new Queue1L<>();
168         terms.enqueue(term);
169
170         Map.Pair<String, String> resultPair =
    glossary.removeAny();
171
172         Glossary.processTerm(terms, resultPair, "data");
173
174         assertNotNull(resultPair);
175         assertEquals(term, resultPair.key());
```

```
176         assertEquals(definition, resultPair.value());
177     }
178
179     /**
180      * Routine test case for the processTerm method. Tests
    with a standard term
181      * and definition.
182      */
183     @Test
184     public void testProcessTerm_Routine() {
185         String term = "Java";
186         String definition = "A programming language.";
187
188         Map<String, String> glossary = new Map1L<>();
189         glossary.add(term, definition);
190         Queue<String> terms = new Queue1L<>();
191         terms.enqueue(term);
192
193         Map.Pair<String, String> resultPair =
    glossary.removeAny();
194
195         Glossary.processTerm(terms, resultPair, "data");
196
197         assertNotNull(resultPair);
198         assertEquals(term, resultPair.key());
199         assertEquals(definition, resultPair.value());
200     }
201
202     /**
203      * Edge test case for the processTerm method. Tests with
    an empty term and
204      * definition.
205      */
206     @Test
207     public void testProcessTerm_Edge() {
208         String term = "";
209         String definition = "";
210
211         Map<String, String> glossary = new Map1L<>();
212         glossary.add(term, definition);
```

```
213         Queue<String> terms = new Queue1L<>();
214         terms.enqueue(term);
215 
216         Map.Pair<String, String> resultPair =
    glossary.removeAny();
217         Glossary.processTerm(terms, resultPair, "data");
218     }
219 
220     /**
221      * Challenging test case for the processTerm method.
    Tests with a term and
222      * definition containing only spaces.
223      */
224     @Test
225     public void testProcessTerm_Challenging2() {
226         String term = "   ";
227         String definition = "    ";
228 
229         Map<String, String> glossary = new Map1L<>();
230         glossary.add(term, definition);
231         Queue<String> terms = new Queue1L<>();
232         terms.enqueue(term);
233 
234         Map.Pair<String, String> resultPair =
    glossary.removeAny();
235 
236         Glossary.processTerm(terms, resultPair, "data");
237 
238         assertNotNull(resultPair);
239         assertEquals(term, resultPair.key());
240         assertEquals("    ", resultPair.value());
241     }
242 
243 }
244 
```