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

import java.util.Comparator;

import org.junit.Test;

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

/**
 * @author Ethan Jones
 */

public class GlossaryTest {
    /**
     * Testing outputHeader.
     *
     * Just a print method, only testing if it is printing as expected. There
     * are no other cases that I could account for in this.
     */
    @Test
    public void testCreateIndexHeader() {
        SimpleWriter out = new SimpleWriter1L("data/outputHeaderTest");
        SimpleReader in = new SimpleReader1L("data/outputHeaderTest");
        Glossary.createIndexHeader(out);
        assertEquals("<html>", in.nextLine());
        assertEquals("<head>", in.nextLine());
        assertEquals("<title>Glossary</title>", in.nextLine());
        assertEquals("</head>", in.nextLine());
        assertEquals("<body>", in.nextLine());
        assertEquals("<h2>Glossary</h2>", in.nextLine());
        assertEquals("<hr>", in.nextLine());
        assertEquals("<h3>Index:</h3>", in.nextLine());
        assertEquals("<ul>", in.nextLine());
        out.close();
        in.close();
    }

    /**
     * Testing outputFooter.
     *
     * Just a print method, only testing if it is printing as expected. There
     * are no other cases that I could account for in this.
     */
    @Test
    public void testCreateIndexFooter() {
        SimpleWriter out = new SimpleWriter1L("data/outputFooterTest");
        SimpleReader in = new SimpleReader1L("data/outputFooterTest");
        Glossary.createIndexFooter(out);
        assertEquals("</ul>", in.nextLine());
        assertEquals("</body>", in.nextLine());
        assertEquals("</html>", in.nextLine());
        out.close();
```

```
60          in.close();
61      }
62
63      /**
64       * Testing outputWordFile using Application.
65       *
66       * Testing outputWordFile using only one word.
67       */
68      @Test
69      public void testOutputWordFileApplication() {
70          SimpleWriter out = new SimpleWriter1L("data/outputWordFileTest");
71          SimpleReader in = new SimpleReader1L("data/outputWordFileTest");
72          Glossary.createHTMLWords(out, "application",
73                  "the act of putting to a special use or purpose.");
74          assertEquals("<html>", in.nextLine());
75          assertEquals("<head>", in.nextLine());
76          assertEquals("<title>application</title>", in.nextLine());
77          assertEquals("</head>", in.nextLine());
78          assertEquals("<body>", in.nextLine());
79          assertEquals(
80                  "<h2><b><i><font color=\"red\">application</font></i></b></h2>",
81                  in.nextLine());
82          assertEquals("<blockquote>the act of putting to a special use or "
83                  + "purpose.</blockquote>", in.nextLine());
84          assertEquals("<hr />", in.nextLine());
85          assertEquals("<p>Return to <a href=\"index.html\">index</a>.</p>",
86                  in.nextLine());
87          assertEquals("</body>", in.nextLine());
88          assertEquals("</html>", in.nextLine());
89          assertEquals("", in.nextLine());
90          out.close();
91          in.close();
92      }
93
94      /**
95       * Testing outputWordFile using no words.
96       *
97       * Since this usually is expecting words, I decided to use no words The
98       * reaction should be that it prints code with a blank instead of a word
99       */
100     @Test
101     public void testOutputWordFileNoWords() {
102         SimpleWriter out = new SimpleWriter1L("data/outputWordFileTest");
103         SimpleReader in = new SimpleReader1L("data/outputWordFileTest");
104         Glossary.createHTMLWords(out, "", "");
105         assertEquals("<html>", in.nextLine());
106         assertEquals("<head>", in.nextLine());
107         assertEquals("<title></title>", in.nextLine());
108         assertEquals("</head>", in.nextLine());
109         assertEquals("<body>", in.nextLine());
110         assertEquals("<h2><b><i><font color=\"red\"></font></i></b></h2>",
111                 in.nextLine());
112         assertEquals("<blockquote></blockquote>", in.nextLine());
113         assertEquals("<hr />", in.nextLine());
114         assertEquals("<p>Return to <a href=\"index.html\">index</a>.</p>",
115                 in.nextLine());
116         assertEquals("</body>", in.nextLine());
117         assertEquals("</html>", in.nextLine());
118         assertEquals("", in.nextLine());
```

```
119         out.close();
120         in.close();
121     }
122
123     /**
124      * Testing my sort method using 15 words.
125      *
126      * This is a routine case, just a lot of different words
127      */
128     @Test
129     public void testSorting() {
130         Queue<String> wordList = new Queue1L<>();
131         Queue<String> wordListExpected = new Queue1L<>();
132         wordList.enqueue("frank");
133         wordList.enqueue("cook");
134         wordList.enqueue("important");
135         wordList.enqueue("adult");
136         wordList.enqueue("distributor");
137         wordList.enqueue("taste");
138         wordList.enqueue("conversation");
139         wordList.enqueue("peace");
140         wordList.enqueue("knit");
141         wordList.enqueue("arrest");
142         wordList.enqueue("denial");
143         wordList.enqueue("tease");
144         wordList.enqueue("determine");
145         wordList.enqueue("assertive");
146         wordList.enqueue("restoration");
147         wordListExpected.enqueue("adult");
148         wordListExpected.enqueue("arrest");
149         wordListExpected.enqueue("assertive");
150         wordListExpected.enqueue("conversation");
151         wordListExpected.enqueue("cook");
152         wordListExpected.enqueue("denial");
153         wordListExpected.enqueue("determine");
154         wordListExpected.enqueue("distributor");
155         wordListExpected.enqueue("frank");
156         wordListExpected.enqueue("important");
157         wordListExpected.enqueue("knit");
158         wordListExpected.enqueue("peace");
159         wordListExpected.enqueue("restoration");
160         wordListExpected.enqueue("taste");
161         wordListExpected.enqueue("tease");
162         Comparator<String> queueSort = new Glossary.StringOrder();
163         wordList.sort(queueSort);
164         assertEquals(wordList, wordListExpected);
165     }
166
167     /**
168      * Testing createMap method with the word "Application.
169      *
170      * This is an edge case because this is only testing one word, as testing no
171      * words would not result in anything useful
172      */
173     @Test
174     public void testCreateMapApplication() {
175         SimpleWriter out = new SimpleWriter1L("data/ApplicationMap");
176         out.println("application");
177         out.println("the act of putting to a special use or purpose");
```

```
178         out.println();
179         SimpleReader in = new SimpleReader1L("data/ApplicationMap");
180         Queue<String> wordList = new Queue1L<>();
181         Queue<String> wordListExpected = new Queue1L<>();
182         wordListExpected.enqueue("application");
183         Map<String, String> map = Glossary.createMap(in, wordList);
184         Map<String, String> mapExpected = new Map1L<>();
185         mapExpected.add("application",
186                 "the act of putting to a special use or purpose");
187         out.close();
188         in.close();
189         assertEquals(map, mapExpected);
190         assertEquals(wordList, wordListExpected);
191     }
192 
193     /**
194      * Testing createMap using Application and Chauvinist.
195      *
196      * This is a normal case with only two words
197      */
198     @Test
199     public void testCreateMapApplicationChauvinist() {
200         SimpleWriter out = new SimpleWriter1L("data/Application_Chauvinist");
201         out.println("chauvinist");
202         out.println("a person who is aggressively and blindly patriotic, "
203                 + "especially one devoted to military glory");
204         out.println();
205         out.println("application");
206         out.println("the act of putting to a special use or purpose");
207         out.println();
208         SimpleReader in = new SimpleReader1L("data/Application_Chauvinist");
209         Queue<String> wordList = new Queue1L<>();
210         Queue<String> wordListExpected = new Queue1L<>();
211         wordListExpected.enqueue("application");
212         wordListExpected.enqueue("chauvinist");
213         Map<String, String> map = Glossary.createMap(in, wordList);
214         Map<String, String> mapExpected = new Map1L<>();
215         mapExpected.add("application",
216                 "the act of putting to a special use or purpose");
217         mapExpected.add("chauvinist",
218                 "a person who is aggressively and blindly patriotic, "
219                         + "especially one devoted to military glory");
220         out.close();
221         in.close();
222         assertEquals(map, mapExpected);
223         assertEquals(wordList, wordListExpected);
224     }
225 
226     /**
227      * Testing createMap using 15 words.
228      *
229      * This is a challenge case because it is giving 15 words in an order that
230      * is not sorted properly and seeing if all 15 words are sorted
231      * alphabetically and have the proper keys.
232      *
233      */
234     @Test
235     public void testCreateMap15Words() {
236         SimpleWriter out = new SimpleWriter1L("data/15Words");
```

```
237         out.println("frank");
238         out.println("direct and unreserved in speech");
239         out.println();
240         out.println("cook");
241         out.println("to prepare (food) by the use of heat, as by boiling, "
242                 + "baking, or roasting.");
243         out.println();
244         out.println("important");
245         out.println("of much or great significance or consequence");
246         out.println();
247         out.println("adult");
248         out.println("a person who is fully grown or developed or of age.");
249         out.println();
250         out.println("distributor");
251         out.println("a person or thing that distributes");
252         out.println();
253         out.println("taste");
254         out.println("to try or test the flavor or quality of (something) by"
255                 + " taking some into the mouth");
256         out.println();
257         out.println("conversation");
258         out.println("informal interchange of thoughts, information, etc., by "
259                 + "spoken words; oral communication between persons");
260         out.println();
261         out.println("peace");
262         out.println("the nonwarring condition of a nation, group of "
263                 + "nations, or the world.");
264         out.println();
265         out.println("knit");
266         out.println("to join closely and firmly, as members or parts");
267         out.println();
268         out.println("arrest");
269         out.println("to seize (a person) by legal authority or warrant");
270         out.println();
271         out.println("denial");
272         out.println("an assertion that something said or believed is false");
273         out.println();
274         out.println("tease");
275         out.println("to irritate or provoke with persistent petty distractions,"
276                 + " trifling jests, or other annoyances, "
277                 + "often in a playful way");
278         out.println();
279         out.println("determine");
280         out.println("to conclude or ascertain, as after "
281                 + "reasoning, observation, etc.");
282         out.println();
283         out.println("assertive");
284         out.println("confidently aggressive or self-assured");
285         out.println();
286         out.println("restoration");
287         out.println("the act of restoring");
288         out.println();
289         SimpleReader in = new SimpleReader1L("data/15Words");
290         Queue<String> wordList = new Queue1L<>();
291         Queue<String> wordListExpected = new Queue1L<>();
292         wordListExpected.enqueue("frank");
293         wordListExpected.enqueue("cook");
294         wordListExpected.enqueue("important");
295         wordListExpected.enqueue("adult");
```

```
296         wordListExpected.enqueue("distributor");
297         wordListExpected.enqueue("taste");
298         wordListExpected.enqueue("conversation");
299         wordListExpected.enqueue("peace");
300         wordListExpected.enqueue("knit");
301         wordListExpected.enqueue("arrest");
302         wordListExpected.enqueue("denial");
303         wordListExpected.enqueue("tease");
304         wordListExpected.enqueue("determine");
305         wordListExpected.enqueue("assertive");
306         wordListExpected.enqueue("restoration");
307         Comparator<String> queueSort = new Glossary.StringOrder();
308         wordListExpected.sort(queueSort);
309         Map<String, String> map = Glossary.createMap(in, wordList);
310         Map<String, String> mapExpected = new Map1L<>();
311         mapExpected.add("frank", "direct and unreserved in speech");
312         mapExpected.add("cook",
313                 "to prepare (food) by the use of heat, as by boiling, "
314                         + "baking, or roasting.");
315         mapExpected.add("important",
316                 "of much or great significance or consequence");
317         mapExpected.add("adult",
318                 "a person who is fully grown or developed or of age.");
319         mapExpected.add("distributor", "a person or thing that distributes");
320         mapExpected.add("taste",
321                 "to try or test the flavor or quality of (something) by"
322                         + " taking some into the mouth");
323         mapExpected.add("conversation",
324                 "informal interchange of thoughts, information, etc., by "
325                         + "spoken words; oral communication between persons");
326         mapExpected.add("peace",
327                 "the nonwarring condition of a nation, group of "
328                         + "nations, or the world.");
329         mapExpected.add("knit",
330                 "to join closely and firmly, as members or parts");
331         mapExpected.add("arrest",
332                 "to seize (a person) by legal authority or warrant");
333         mapExpected.add("denial",
334                 "an assertion that something said or believed is false");
335         mapExpected.add("tease",
336                 "to irritate or provoke with persistent petty distractions,"
337                         + " trifling jests, or other annoyances, "
338                         + "often in a playful way");
339         mapExpected.add("determine", "to conclude or ascertain, as after "
340                 + "reasoning, observation, etc.");
341         mapExpected.add("assertive", "confidently aggressive or self-assured");
342         mapExpected.add("restoration", "the act of restoring");
343         out.close();
344         in.close();
345         assertEquals(map, mapExpected);
346         assertEquals(wordList, wordListExpected);
347     }
348 
349     /**
350      * Testing processWord using chauvinist.
351      *
352      * Testing if processing a word works properly
353      */
354     @Test
```

```
355     public void testProcessWordChauvinist() {
356         SimpleWriter out = new SimpleWriter1L("data/processWordTest");
357         SimpleReader in = new SimpleReader1L("data/processWordTest");
358         Glossary.processWord(out, "chauvinist",
359                 "a person who is aggressively and blindly patriotic, "
360                         + "especially one devoted to military glory",
361                 "data");
362         assertEquals("<li><a href=\"chauvinist.html\">chauvinist</a></li>",
363                 in.nextLine());
364         out.close();
365         in.close();
366     }
367
368     /**
369      * Testing processWord using application.
370      *
371      * Again, testing if processing a word works as intended with a different
372      * word
373      */
374     @Test
375     public void testProcessWordApplication() {
376         SimpleWriter out = new SimpleWriter1L("data/processWordTest2");
377         SimpleReader in = new SimpleReader1L("data/processWordTest2");
378         Glossary.processWord(out, "application",
379                 "the act of putting to a special use or purpose.", "data");
380         assertEquals("<li><a href=\"application.html\">application</a></li>",
381                 in.nextLine());
382         out.close();
383         in.close();
384     }
385
386     /**
387      * Testing processFile using Application and Chauvinist.
388      *
389      * Routine case, just testing if this works properly
390      */
391     @Test
392     public void testProcessFile() {
393         SimpleWriter out = new SimpleWriter1L("data/processFileTest");
394         SimpleReader in = new SimpleReader1L("data/processFileTest");
395
396         Glossary.processFile("data/Application_Chauvinist", "data", out);
397         assertEquals("<li><a href=\"Application.html\">Application</a></li>",
398                 in.nextLine());
399         assertEquals("<li><a href=\"Chauvinist.html\">Chauvinist</a></li>",
400                 in.nextLine());
401         out.close();
402         in.close();
403     }
404
405     /**
406      * Testing processFile using 15 different words.
407      *
408      * Challenge case, seeing if all 15 wors are properly capetalized and
409      * alphabetized
410      */
411     @Test
412     public void testProcessFile15Words() {
413         SimpleWriter out = new SimpleWriter1L("data/processFileTest");
```

```
414         SimpleReader in = new SimpleReader1L("data/processFileTest");
415 
416         Glossary.processFile("data/15Words", "data", out);
417         assertEquals("<li><a href=\"Adult.html\">Adult</a></li>",
418                 in.nextLine());
419         assertEquals("<li><a href=\"Arrest.html\">Arrest</a></li>",
420                 in.nextLine());
421         assertEquals("<li><a href=\"Assertive.html\">Assertive</a></li>",
422                 in.nextLine());
423         assertEquals("<li><a href=\"Conversation.html\">Conversation</a></li>",
424                 in.nextLine());
425         assertEquals("<li><a href=\"Cook.html\">Cook</a></li>", in.nextLine());
426         assertEquals("<li><a href=\"Denial.html\">Denial</a></li>",
427                 in.nextLine());
428         assertEquals("<li><a href=\"Determine.html\">Determine</a></li>",
429                 in.nextLine());
430         assertEquals("<li><a href=\"Distributor.html\">Distributor</a></li>",
431                 in.nextLine());
432         assertEquals("<li><a href=\"Frank.html\">Frank</a></li>",
433                 in.nextLine());
434         assertEquals("<li><a href=\"Important.html\">Important</a></li>",
435                 in.nextLine());
436         assertEquals("<li><a href=\"Knit.html\">Knit</a></li>", in.nextLine());
437         assertEquals("<li><a href=\"Peace.html\">Peace</a></li>",
438                 in.nextLine());
439         assertEquals("<li><a href=\"Restoration.html\">Restoration</a></li>",
440                 in.nextLine());
441         assertEquals("<li><a href=\"Taste.html\">Taste</a></li>",
442                 in.nextLine());
443         assertEquals("<li><a href=\"Tease.html\">Tease</a></li>",
444                 in.nextLine());
445         out.close();
446         in.close();
447     }
448 
449     /**
450      * Testing processFile, createIndexHeader, and CreateIndexFooter using 15
451      * different words. This is putting together a full glossary.
452      *
453      * This should make it so that the files are all stored in a file called
454      * fullGlossaryTest.html
455      */
456     @Test
457     public void testFullGlossary() {
458         SimpleWriter out = new SimpleWriter1L("data/fullGlossaryTest.html");
459         SimpleReader in = new SimpleReader1L("data/fullGlossaryTest.html");
460         Glossary.createIndexHeader(out);
461         Glossary.processFile("data/15Words", "data", out);
462         Glossary.createIndexFooter(out);
463         assertEquals("<html>", in.nextLine());
464         assertEquals("<head>", in.nextLine());
465         assertEquals("<title>Glossary</title>", in.nextLine());
466         assertEquals("</head>", in.nextLine());
467         assertEquals("<body>", in.nextLine());
468         assertEquals("<h2>Glossary</h2>", in.nextLine());
469         assertEquals("<hr>", in.nextLine());
470         assertEquals("<h3>Index:</h3>", in.nextLine());
471         assertEquals("<ul>", in.nextLine());
472         assertEquals("<li><a href=\"Adult.html\">Adult</a></li>",
```

```
473                 in.nextLine());
474         assertEquals("<li><a href=\"Arrest.html\">Arrest</a></li>",
475                 in.nextLine());
476         assertEquals("<li><a href=\"Assertive.html\">Assertive</a></li>",
477                 in.nextLine());
478         assertEquals("<li><a href=\"Conversation.html\">Conversation</a></li>",
479                 in.nextLine());
480         assertEquals("<li><a href=\"Cook.html\">Cook</a></li>", in.nextLine());
481         assertEquals("<li><a href=\"Denial.html\">Denial</a></li>",
482                 in.nextLine());
483         assertEquals("<li><a href=\"Determine.html\">Determine</a></li>",
484                 in.nextLine());
485         assertEquals("<li><a href=\"Distributor.html\">Distributor</a></li>",
486                 in.nextLine());
487         assertEquals("<li><a href=\"Frank.html\">Frank</a></li>",
488                 in.nextLine());
489         assertEquals("<li><a href=\"Important.html\">Important</a></li>",
490                 in.nextLine());
491         assertEquals("<li><a href=\"Knit.html\">Knit</a></li>", in.nextLine());
492         assertEquals("<li><a href=\"Peace.html\">Peace</a></li>",
493                 in.nextLine());
494         assertEquals("<li><a href=\"Restoration.html\">Restoration</a></li>",
495                 in.nextLine());
496         assertEquals("<li><a href=\"Taste.html\">Taste</a></li>",
497                 in.nextLine());
498         assertEquals("<li><a href=\"Tease.html\">Tease</a></li>",
499                 in.nextLine());
500         assertEquals("</ul>", in.nextLine());
501         assertEquals("</body>", in.nextLine());
502         assertEquals("</html>", in.nextLine());
503         out.close();
504         in.close();
505     }
506
507 }
508
```