```
 1 import static components.utilities.Tokenizer.isCondition;
13
15  * {@code Tokenizer} utility class with methods to tokenize an input stream and
18 public final class Tokenizer {
19
21      * Private members --------------------------------------------------------
23
25      * Definition of whitespace separators.
27     private static final String SEPARATORS = " \t\n\r";
28
30      * Private constructor so this utility class cannot be instantiated.
32     private Tokenizer() {
34
36      * Returns the token "kind" corresponding to the given {@code token}.
43     private static String tokenKind(String token) {
55
56     /**
57      * Returns the first "word" (maximal length string of characters not in
58      * {@code SEPARATORS}) or "separator string" (maximal length string of
59      * characters in {@code SEPARATORS}) in the given {@code text} starting at
60      * the given {@code position}.
61      *
62      * @param text
63      *            the {@code String} from which to get the word or separator
64      *            string
65      * @param position
66      *            the starting index
67      * @return the first word or separator string found in {@code text} starting
68      *         at index {@code position}
69      * @requires 0 <= position < |text|
70      * @ensures <pre>
71      * nextWordOrSeparator =
72      *   text[position, position + |nextWordOrSeparator|)  and
73      * if entries(text[position, position + 1)) intersection entries(SEPARATORS) = {}
74      * then
75      *   entries(nextWordOrSeparator) intersection entries(SEPARATORS) = {}  and
76      *   (position + |nextWordOrSeparator| = |text|  or
77      *    entries(text[position, position + |nextWordOrSeparator| + 1))
78      *      intersection entries(SEPARATORS) /= {})
79      * else
80      *   entries(nextWordOrSeparator) is subset of entries(SEPARATORS)  and
81      *   (position + |nextWordOrSeparator| = |text|  or
82      *    entries(text[position, position + |nextWordOrSeparator| + 1))
83      *      is not subset of entries(SEPARATORS))
84      * </pre>
85      */
86     private static String nextWordOrSeparator(String text, int position) {
87         assert text != null : "Violation of: text is not null";
88         assert 0 <= position : "Violation of: 0 <= position";
89         assert position < text.length() : "Violation of: position < |text|";
90
91         boolean done = false;
92         int i = 0;
93         String word = "";
94
95         while (!done) {
96             if (text.length() < position + i + 1) {
```

```
 97                 done = true;
 98             } else {
 99                 if (SEPARATORS.contains(text.subSequence(position+i,position+i+1))) {
100                     done = true;
101                 } else {
102                     word = word + text.charAt(position + i);
103                 }
104                 i++;
105             }
106         }
107 
108         return word;
109 
110     }
111 
113      * Public members --------------------------------------------------------
115 
117      * Token to mark the end of the input. This token cannot come from the input
120     public static final String END_OF_INPUT = "### END OF INPUT ###";
121 
122     /**
123      * Tokenizes the entire input getting rid of all whitespace separators and
124      * returning the non-separator tokens in a {@code Queue<String>}.
125      *
126      * @param in
127      *            the input stream
128      * @return the queue of tokens
129      * @updates in.content
130      * @requires in.is_open
131      * @ensures <pre>
132      * tokens =
133      *   [the non-whitespace tokens in #in.content] * <END_OF_INPUT>  and
134      * in.content = <>
135      * </pre>
136      */
137     public static Queue<String> tokens(SimpleReader in) {
138         assert in != null : "Violation of: in is not null";
139         assert in.isOpen() : "Violation of: in.is_open";
140 
141 
142 
143         Queue<String> tokens = new Queue2<String>();
144         String next = in.nextLine();
145         String newToken = "";
146 
147 
148         while (!in.atEOS()) {
149             int i = 0;
150             while (i < next.length()) {
151 
152                 newToken = nextWordOrSeparator(next, i);
153 
154                 if (!newToken.isEmpty()) {
155                     tokens.enqueue(newToken);
156                 }
157 
158                 i += newToken.length() + 1;
```

```
159             }
160             next = in.nextLine();
161         }
162 
163         // one more time through
164         int i = 0;
165         while (i < next.length()) {
166 
167             newToken = nextWordOrSeparator(next, i);
168 
169             if (!newToken.isEmpty()) {
170                 tokens.enqueue(newToken);
171             }
172 
173             i += newToken.length() + 1;
174         }
175 
176         tokens.enqueue(END_OF_INPUT);
177 
178         return tokens;
179     }
180 
181     /*
182      * Main test method ----------------------------------------------------
183      */
184 
185     /**
186      * Main method.
187      *
188      * @param args
189      *            the command line arguments
190      */
191     public static void main(String[] args) {
192         SimpleReader in = new SimpleReader1L();
193         SimpleWriter out = new SimpleWriter1L();
194         /*
195          * Get input file name
196          */
197         out.print("Enter input file name: ");
198         String fileName = in.nextLine();
199         /*
200          * Tokenize input with Tokenizer implementation from library.
201          */
202         SimpleReader file = new SimpleReader1L(fileName);
203         Queue<String> q1 = components.utilities.Tokenizer.tokens(file);
204         file.close();
205         /*
206          * Tokenize input with Tokenizer implementation under test.
207          */
208         file = new SimpleReader1L(fileName);
209         Queue<String> q2 = Tokenizer.tokens(file);
210         file.close();
211         /*
212          * Check that the two Queues are equal.
213          */
214         out.println();
215         if (q2.equals(q1)) {
```

```
216             out.println("Input appears to have been tokenized correctly.");
217         } else {
218             out.println("Error: input tokens are not correct.");
219             out.println("Expected: " + q1);
220             out.println();
221             out.println("Actual: " + q2);
222         }
223         out.println();
224         /*
225          * Generate expected output in file "data/expected-output.txt"
226          */
227         out.println("*** Generating expected output ***");
228         SimpleWriter tOut = new SimpleWriter1L("data/expected-output.txt");
229         for (String token : q1) {
230             tOut.println(tokenKind(token) + ": <" + token + ">");
231         }
232         tOut.close();
233         /*
234          * Generate actual output in file "data/actual-output.txt"
235          */
236         out.println("*** Generating actual output ***");
237         tOut = new SimpleWriter1L("data/actual-output.txt");
238         for (String token : q2) {
239             tOut.println(tokenKind(token) + ": <" + token + ">");
240         }
241         tOut.close();
242 
243         in.close();
244         out.close();
245     }
246 
247 }
248 
```