```java
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
import components.xmltree.XMLTree;
import components.xmltree.XMLTree1;

/**
 * Program to convert a series of XML RSS (version 2.0) feed from a list of
 * given URLs into the corresponding HTML output files.
 *
 * @author Yiming Cheng
 *
 */
public final class RSSAggregator {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private RSSAggregator() {
    }

    /**
     * Outputs the "opening" tags in the generated HTML file. These are the
     * expected elements generated by this method:
     *
     * <html> <head> <title>the channel tag title as the page title</title>
     * </head> <body>
     * <h1>the page title inside a link to the <channel> link</h1>
     * <p>
     * the channel description
     * </p>
     * <table border="1">
     * <tr>
     * <th>Date</th>
     * <th>Source</th>
     * <th>News</th>
     * </tr>
     *
     * @param channel
     *            the channel element XMLTree
     * @param out
     *            the output stream
     * @updates out.content
     * @requires [the root of channel is a <channel> tag] and out.is_open
     * @ensures out.content = #out.content * [the HTML "opening" tags]
     */
    private static void outputHeader(XMLTree channel, SimpleWriter out) {
        assert channel != null : "Violation of: channel is not null";
        assert out != null : "Violation of: out is not null";
        assert channel.isTag() && channel.label().equals("channel") : ""
                + "Violation of: the label root of channel is a <channel> tag";
        assert out.isOpen() : "Violation of: out.is_open";
        //find the content of the title
        String name = " ";
        if (getChildElement(channel, "title") <= -1
                || channel.child(getChildElement(channel, "title"))
                        .numberOfChildren() == 0) {
            name = "No Title";
        } else {
            name = channel.child(getChildElement(channel, "title")).child(0)
                    .label();
```

```
63          }
64          //find the description that is provided
65          String description = " ";
66          if (getChildElement(channel, "description") <= -1
67                  || channel.child(getChildElement(channel, "description"))
68                          .numberOfChildren() == 0) {
69              description = "No Description";
70          } else {
71              description = channel.child(getChildElement(channel, "description"))
72                      .child(0).label();
73          }
74          //print the information that the page needs
75          out.println("<html>");
76          out.println("<head>");
77          out.println("<title>" + name + "</title>");
78          out.println("</head>");
79          out.print("<body>");
80          out.println(
81                  "<h1>" + "<a href=\""
82                          + channel.child(getChildElement(channel, "link"))
83                                  .child(0).label()
84                          + "\">" + name + "</a>" + "</h1>");
85          out.println("<p>");
86          out.println(description);
87          out.println("</p>");
88          out.println(" <table border=\"1\">");
89          out.println("<tr>");
90          out.println("<th>Date</th>");
91          out.println("<th>Source</th>");
92          out.println("<th>News</th>");
93          out.println("</tr>");
94
95      }
96
97      /**
98       * Outputs the "closing" tags in the generated HTML file. These are the
99       * expected elements generated by this method:
100      *
101      * </table>
102      * </body> </html>
103      *
104      * @param out
105      *            the output stream
106      * @updates out.contents
107      * @requires out.is_open
108      * @ensures out.content = #out.content * [the HTML "closing" tags]
109      */
110     private static void outputFooter(SimpleWriter out) {
111         assert out != null : "Violation of: out is not null";
112         assert out.isOpen() : "Violation of: out.is_open";
113
114         out.println("</table>");
115         out.println("</body>");
116         out.println("</html>");
117     }
118
119     /**
120      * Finds the first occurrence of the given tag among the children of the
121      * given {@code XMLTree} and return its index; returns -1 if not found.
122      *
123      * @param xml
124      *            the {@code XMLTree} to search
```

```
125      * @param tag
126      *            the tag to look for
127      * @return the index of the first child of type tag of the {@code XMLTree}
128      *         or -1 if not found
129      * @requires [the label of the root of xml is a tag]
130      * @ensures <pre>
131      * getChildElement =
132      *  [the index of the first child of type tag of the {@code XMLTree} or
133      *   -1 if not found]
134      * </pre>
135      */
136     private static int getChildElement(XMLTree xml, String tag) {
137         assert xml != null : "Violation of: xml is not null";
138         assert tag != null : "Violation of: tag is not null";
139         assert xml.isTag() : "Violation of: the label root of xml is a tag";
140
141         //compare the information
142
143         int index = -1;
144         int number = xml.numberOfChildren();
145         int i = 0;
146         while (index == -1 && number > i) {
147             if (xml.child(i).isTag()) {
148                 if (xml.child(i).label().equals(tag)) {
149                     index = i;
150                 }
151             }
152             i++;
153         }
154         return index;
155     }
156
157     /**
158      * Processes one news item and outputs one table row. The row contains three
159      * elements: the publication date, the source, and the title (or
160      * description) of the item.
161      *
162      * @param item
163      *            the news item
164      * @param out
165      *            the output stream
166      * @updates out.content
167      * @requires [the label of the root of item is an <item> tag] and
168      *           out.is_open
169      * @ensures <pre>
170      * out.content = #out.content *
171      *   [an HTML table row with publication date, source, and title of news item]
172      * </pre>
173      */
174     private static void processItem(XMLTree item, SimpleWriter out) {
175         assert item != null : "Violation of: item is not null";
176         assert out != null : "Violation of: out is not null";
177         assert item.isTag() && item.label().equals("item") : ""
178                 + "Violation of: the label root of item is an <item> tag";
179         assert out.isOpen() : "Violation of: out.is_open";
180
181         //find the information about the chart
182         out.println("<tr>");
183         String publicationDate = " ";
184         //compare the date
185         if (getChildElement(item, "pubDate") != -1) {
186             publicationDate = item.child(getChildElement(item, "pubDate"))
```

```
187                     .child(0).label();
188         } else {
189             publicationDate = "No data available";
190         }
191         out.println("<td>" + publicationDate + "</td >");
192         String sourceOfLink = " ";
193         String URL = "";
194         //find the URL that the chart would be contained
195         if (getChildElement(item, "source") != -1) {
196             sourceOfLink = item.child(getChildElement(item, "source")).child(0)
197                     .label();
198             URL = item.child(getChildElement(item, "source"))
199                     .attributeValue("url");
200             out.println("<td><a href=\"" + URL + "\">" + sourceOfLink
201                     + "</a></td>");
202         } else {
203             sourceOfLink = "No source available";
204             out.println("<td>" + sourceOfLink + "</td>");
205 
206         }
207         String newsTitle = " ";
208         // provide the title of the RSS
209         if (getChildElement(item, "title") > -1
210                 && item.child(getChildElement(item, "title"))
211                         .numberOfChildren() != 0) {
212             newsTitle = item.child(getChildElement(item, "title")).child(0)
213                     .label();
214         } else if (getChildElement(item, "descrption") > -1
215                 && item.child(getChildElement(item, "description"))
216                         .numberOfChildren() != 0) {
217             newsTitle = item.child(getChildElement(item, "description"))
218                     .child(0).label();
219         } else {
220             newsTitle = "No title available";
221         }
222         String newsLink = "";
223         if (getChildElement(item, "link") > -1) {
224             newsLink = item.child(getChildElement(item, "link")).child(0)
225                     .label();
226             out.println("<td><a href=\"" + newsLink + "\">" + newsTitle + "</a>"
227                     + "</td>");
228 
229         } else {
230             out.println("<td>" + newsTitle + "</td>");
231         }
232         out.println("</tr>");
233 
234     }
235 
236     /**
237      * Processes one XML RSS (version 2.0) feed from a given URL converting it
238      * into the corresponding HTML output file.
239      *
240      * @param url
241      *            the URL of the RSS feed
242      * @param file
243      *            the name of the HTML output file
244      * @param out
245      *            the output stream to report progress or errors
246      * @updates out.content
247      * @requires out.is_open
248      * @ensures <pre>
```

```
249      * [reads RSS feed from url, saves HTML document with table of news items
250      *   to file, appends to out.content any needed messages]
251      * </pre>
252      */
253     private static void processFeed(String url, String file, SimpleWriter out) {
254         XMLTree xml = new XMLTree1(url);
255         //prompt the user to type valid xml address
256         if (!xml.label().equals("rss")
257                 && xml.attributeValue("version") == "2.0") {
258             out.print("The file could not be processed.");
259         } else {
260             XMLTree channel = xml.child(0);
261             SimpleWriter outFile = new SimpleWriter1L(file);
262             outputHeader(channel, outFile);
263             //the code would run the different items from the address
264             for (int a = 0; a < xml.child(0).numberOfChildren(); a++) {
265                 if (xml.child(0).child(a).label().equals("item")) {
266                     processItem(channel.child(a), outFile);
267                 }
268             }
269             outputFooter(outFile);
270         }
271     }
272
273     /**
274      * Main method.
275      *
276      * @param args
277      *            the command line arguments; unused here
278      */
279     public static void main(String[] args) {
280         SimpleReader in = new SimpleReader1L();
281         SimpleWriter out = new SimpleWriter1L();
282         //prompt the users to type the names of the xml that they want to process
283         out.print("type the name of an XML file containing a list of URLs");
284         String input = in.nextLine();
285         XMLTree xml = new XMLTree1(input);
286         //prompt the users to type the name of the file that they want to process
287         out.println("Enter a filename that you want to process.");
288         String file = in.nextLine();
289         SimpleWriter outFile = new SimpleWriter1L(file);
290         //process the different urls
291         for (int a = 0; a < xml.numberOfChildren(); a++) {
292             String url = xml.child(a).attributeValue("url");
293             String newfile = xml.child(a).attributeValue("file");
294             processFeed(url, newfile, outFile);
295         }
296         //print the basic structure of html
297         outFile.println("<html>");
298         outFile.println("  <head>");
299         outFile.println(
300                 "     <title>" + xml.attributeValue("title") + "</title>");
301         outFile.println("  </head>");
302         outFile.println("  <body>");
303         outFile.println("     <h2>" + xml.attributeValue("title") + "</h2>");
304         outFile.println("<ul>");
305         for (int i = 0; i < xml.numberOfChildren(); i++) {
306             outFile.print("      <li><a href=\"");
307             outFile.print(xml.child(i).attributeValue("file"));
308             outFile.print(
309                     "\" >" + xml.child(i).attributeValue("name") + "</a></li>");
310             outFile.println();
```

```
311         }
312         outFile.println("    </ul>");
313         outFile.println("  </body>");
314         outFile.println("</html>");
315 
316         in.close();
317         out.close();
318         outFile.close();
319     }
320 
321 }
322 
```