

## RSSReader.java

```
1 import components.simplereader.SimpleReader;
2 import components.simplereader.SimpleReader1L;
3 import components.simplewriter.SimpleWriter;
4 import components.simplewriter.SimpleWriter1L;
5 import components.xmltree.XMLTree;
6 import components.xmltree.XMLTree1;
7
8 /**
9  * Program to convert an XML RSS (version 2.0) feed from a given URL into the
10  * corresponding HTML output file.
11  *
12  * @author VishalKumar
13  *
14  */
15 public final class RSSReader {
16
17     /**
18      * Private constructor so this utility class cannot be instantiated.
19      */
20     private RSSReader() {
21     }
22
23     /**
24      * Outputs the "opening" tags in the generated HTML file. These are the
25      * expected elements generated by this method:
26      *
27      * <html> <head> <title>the channel tag title as the page title</title>
28      * </head> <body>
29      * <h1>the page title inside a link to the <channel> link</h1>
30      * <p>
31      * the channel description
32      * </p>
33      * <table border="1">
34      * <tr>
35      * <th>Date</th>
36      * <th>Source</th>
37      * <th>News</th>
38      * </tr>
39      *
40      * @param channel
41      *         the channel element XMLTree
42      * @param out
43      *         the output stream
44      * @updates out.content
45      * @requires [the root of channel is a <channel> tag] and out.is_open
46      * @ensures out.content = #out.content * [the HTML "opening" tags]
47      */
48     private static void outputHeader(XMLTree channel, SimpleWriter out) {
49         assert channel != null : "Violation of: channel is not null";
50         assert out != null : "Violation of: out is not null";
51         assert channel.isTag() && channel.label().equals("channel") : ""
52             + "Violation of: the label root of channel is a <channel> tag";
53         assert out.isOpen() : "Violation of: out.is_open";
54
55         // Get positions of child, link description
56         int titlePos = getChildElement(channel, "title");
57         int linkPos = getChildElement(channel, "link");
```

## RSSReader.java

```

58     int descPos = getChildElement(channel, "description");
59
60     // print title
61     if (channel.child(titlePos).numberOfChildren() > 0) {
62         out.println("<html><head><title>"
63             + channel.child(titlePos).child(0).label() + "</title>");
64     } else {
65         out.println("<html><head><title>" + "Empty Title" + "</title>");
66     }
67
68     out.println("</head><body>");
69
70     // Print 1st header with hyperLink to page
71     out.print("<h1><a href=\"" + channel.child(linkPos).child(0).label()
72         + "\">" + channel.child(titlePos).child(0).label()
73         + "</a> </h1>");
74     out.println("<p>");
75
76     // print description of channel
77     if (channel.child(descPos).numberOfChildren() > 0) {
78         out.println(channel.child(descPos).child(0).label());
79     } else {
80         out.println("No description");
81     }
82
83     // print closing text of the header
84     out.println("</p>");
85     out.println("<table border=\"1\">");
86     out.println("<tr>");
87     out.println("<th>Date</th>");
88     out.println("<th>Source</th>");
89     out.println("<th>News</th>");
90     out.println("</tr>");
91
92 )
93
94 /**
95  * Outputs the "closing" tags in the generated HTML file. These are the
96  * expected elements generated by this method:
97  *
98  * </table>
99  * </body> </html>
100  *
101  * @param out
102  *         the output stream
103  * @updates out.contents
104  * @requires out.is_open
105  * @ensures out.content = #out.content * [the HTML "closing" tags]
106  */
107 private static void outputFooter(SimpleWriter out) {
108     assert out != null : "Violation of: out is not null";
109     assert out.isOpen() : "Violation of: out.is_open";
110
111     // print closing text of the html page
112     out.println("</table>");
113     out.print("</body></html>");
114 }

```

## RSSReader.java

```

115
116 /**
117  * Finds the first occurrence of the given tag among the children of the
118  * given {@code XMLTree} and return its index; returns -1 if not found.
119  *
120  * @param xml
121  *         the {@code XMLTree} to search
122  * @param tag
123  *         the tag to look for
124  * @return the index of the first child of type tag of the {@code XMLTree}
125  *         or -1 if not found
126  * @requires [the label of the root of xml is a tag]
127  * @ensures <pre>
128  *   getChildElement =
129  *     [the index of the first child of type tag of the {@code XMLTree} or
130  *     -1 if not found]
131  * </pre>
132  */
133 private static int getChildElement(XMLTree xml, String tag) {
134     assert xml != null : "Violation of: xml is not null";
135     assert tag != null : "Violation of: tag is not null";
136     assert xml.isTag() : "Violation of: the label root of xml is a tag";
137
138     int pos = -1;
139     // loop through tree and find first position of desired tag
140     for (int i = 0; i < xml.numberOfChildren() && pos == -1; i++) {
141         String name = xml.child(i).label();
142         if (name.equals(tag)) {
143             pos = i;
144         }
145     }
146     //return the position
147     return pos;
148 }
149
150 /**
151  * Processes one news item and outputs one table row. The row contains three
152  * elements: the publication date, the source, and the title (or
153  * description) of the item.
154  *
155  * @param item
156  *         the news item
157  * @param out
158  *         the output stream
159  * @updates out.content
160  * @requires [the label of the root of item is an <item> tag] and
161  *         out.is_open
162  * @ensures <pre>
163  *   out.content = #out.content *
164  *   [an HTML table row with publication date, source, and title of news item]
165  * </pre>
166  */
167 private static void processItem(XMLTree item, SimpleWriter out) {
168     assert item != null : "Violation of: item is not null";
169     assert out != null : "Violation of: out is not null";
170     assert item.isTag() && item.label().equals("item") : ""
171         + "Violation of: the label root of item is an <item> tag";

```

# RSSReader.java

```

172     assert out.isOpen() : "Violation of: out.is_open";
173
174     // get positions of date, source, title and link
175     int datePos = getChildElement(item, "pubDate");
176     int sourcePos = getChildElement(item, "source");
177     int titlePos = getChildElement(item, "title");
178     int linkPos = getChildElement(item, "link");
179     int descPos = getChildElement(item, "description");
180
181     out.print("<tr>");
182     // print the date
183     if (datePos != -1) {
184         out.println(
185             "<td>" + item.child(datePos).child(0).label() + "</td>");
186     } else {
187         out.println("<td> No date available </td>");
188     }
189     // print the source
190     if (sourcePos != -1) {
191         out.println("<td><a href=\""
192             + item.child(sourcePos).attributeValue("url") + "\">"
193             + item.child(sourcePos).child(0).label() + "</a></td>");
194     } else {
195         out.println("<td> No source available </td>");
196     }
197     // print the article title or the article description with a hyperlink to the article
198     if (titlePos != -1) {
199         if (item.child(titlePos).numberOfChildren() > 0) {
200             out.print("<td><a href=\""
201                 + item.child(linkPos).child(0).label() + "\">";
202             out.print(item.child(titlePos).child(0).label() + "</a></td>");
203         } else {
204             out.println("<td> No title available </td>");
205         }
206     } else if (descPos != 1) {
207         if (item.child(descPos).numberOfChildren() > 0) {
208             out.print("<td><a href=\""
209                 + item.child(linkPos).child(0).label() + "\">";
210             out.print(item.child(descPos).child(0).label() + "</a></td>");
211         } else {
212             out.println("<td> No description available </td>");
213         }
214     } else {
215         out.println("<td> No title available </td>");
216     }
217     out.print("</tr>");
218
219 }
220
221 /**
222  * Main method.
223  *
224  * @param args
225  *      the command line arguments; unused here
226  */
227 public static void main(String[] args) {
228     SimpleReader in = new SimpleReader1L();

```

# RSSReader.java

```
229     SimpleWriter out = new SimpleWriter1L();
230
231     // get url and file name from user
232     out.print("Enter the URL of an RSS 2.0 feed: ");
233     String url = in.nextLine();
234     out.print(
235         "Enter the name of an output file. Make sure to include the .html extension at
the end!: ");
236     String fileName = in.nextLine();
237
238     // setup output steam to file
239     SimpleWriter write = new SimpleWriter1L(fileName);
240
241     // construct the xml tree objects
242     XMLTree root = new XMLTree1(url);
243     XMLTree channel = root.child(0);
244
245     // print the header of the html page
246     outputHeader(channel, write);
247
248     // check to see if the tree is rss 2.0
249     if (root.label().equals("rss")
250         && root.attributeValue("version").equals("2.0")) {
251         // loop through the channel tree and process each <item> tag
252         int i = 0;
253         while (i < channel.numberOfChildren()) {
254             if (channel.child(i).label().equals("item")) {
255                 processItem(channel.child(i), write);
256             }
257             i++;
258         }
259     } else {
260         out.print("Link not valid RSS 2.0");
261     }
262
263     // print the footer of the html page
264     outputFooter(write);
265
266     // close input and output streams
267     in.close();
268     out.close();
269     write.close();
270 }
271
272 }
```