

**Project #5: RSS News Aggregator**

---

```

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 an XML RSS (version 2.0) feed from a given URL into the
 * corresponding HTML output file.
 *
 * @author Danny Kan (kan.74@osu.edu)
 *
 * @version 02252022
 */
public final class RSSAggregator {

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

    /**
     * Processes one XML RSS (version 2.0) feed from a given URL converting it
     * into the corresponding HTML output file.
     *
     * @param url
     *     the URL of the RSS feed
     * @param file
     *     the name of the HTML output file
     * @param out
     *     the output stream to report progress or errors
     * @updates out.content
     * @requires out.is_open
     * @ensures <pre>
     * [reads RSS feed from url, saves HTML document with table of news items
     * to file, appends to out.content any needed messages]
     * </pre>
     */
    private static void processFeed(String url, String file, SimpleWriter out) {
        /**
         * Initialize an XMLTree object from a given RSS (Really Simple
         * Syndication) (Version 2.0) News Feed. If successful, the input is a

```

```

    * valid XML document.
    */
XMLTree xml = new XMLTree1(url);

/*
 * Check the root of the XMLTree to satisfy the condition that the input
 * provided is indeed a valid RSS (Really Simple Syndication) (Version
 * 2.0) news feed.
 */
if (xml.label().equals("rss") && xml.hasAttribute("version")
    && xml.attributeValue("version").equals("2.0")) {
    XMLTree channel = xml.child(0);

    // outputHeader method call:
    outputHeader(channel, out);

    int i = 0;
    while (i < channel.numberOfChildren()) {
        if (channel.child(i).isTag()
            && channel.child(i).label().equals("item")) {
            XMLTree item = channel.child(i);
            // processItem method call:
            processItem(item, out); // prints a row.
        }
        i++;
    }

    // outputFooter method call:
    outputFooter(out);

} else {
    out.println(
        "ERROR - The provided URL is not a valid RSS (Really Simple Syndication) (Version
2.0) XML document.");
}

/**
 * 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";

    // <html>
    out.println("<html>");

    out.println("<head>");
    out.println("<title>");
    int titleIdx = getChildElement(channel, "title");
    String titleContent = channel.child(titleIdx).child(0).label();
    if (channel.child(titleIdx).numberOfChildren() == 0) {
        out.println("Empty Title");
    } else {
        out.println(titleContent);
    }
    out.println("</title>");
    out.println("</head>");

    // <body>
    out.println("<body>");

    int linkIdx = getChildElement(channel, "link");

```

```

String linkContent = channel.child(linkIdx).child(0).label();
out.println("<h1><a href=\"" + linkContent + "\">" + titleContent
    + "</a></h1>");

out.println("<p>");
int descriptionIdx = getChildElement(channel, "description");
if (channel.child(descriptionIdx).numberOfChildren() == 0) {
    out.println("No description");
} else {
    String descriptionContent = channel.child(descriptionIdx).child(0)
        .label();
    out.println(descriptionContent);
}
out.println("</p>");

// <table>
out.println("<table border = \"1\">");

out.println("<tr>");
out.println("<th>Date</th>");
out.println("<th>Source</th>");
out.println("<th>News</th>");
out.println("</tr>");
}

/**
 * Outputs the "closing" tags in the generated HTML file. These are the
 * expected elements generated by this method:
 *
 * </table>
 * </body> </html>
 *
 * @param out
 *     the output stream
 * @updates out.contents
 * @requires out.is_open
 * @ensures out.content = #out.content * [the HTML "closing" tags]
 */
private static void outputFooter(SimpleWriter out) {
    assert out != null : "Violation of: out is not null";
    assert out.isOpen() : "Violation of: out.is_open";

    out.println("</table>");
    out.println("</body>");

```

```

        out.println("</html>");
    }

    /**
     * Finds the first occurrence of the given tag among the children of the
     * given {@code XMLTree} and return its index; returns -1 if not found.
     *
     * @param xml
     *      the {@code XMLTree} to search
     * @param tag
     *      the tag to look for
     * @return the index of the first child of type tag of the {@code XMLTree}
     *      or -1 if not found
     * @requires [the label of the root of xml is a tag]
     * @ensures <pre>
     *      getChildElement =
     *      [the index of the first child of type tag of the {@code XMLTree} or
     *      -1 if not found]
     * </pre>
     */
    private static int getChildElement(XMLTree xml, String tag) {
        assert xml != null : "Violation of: xml is not null";
        assert tag != null : "Violation of: tag is not null";
        assert xml.isTag() : "Violation of: the label root of xml is a tag";

        int firstOccurenceldx = -1;
        int i = 0;
        while (i < xml.numberOfChildren()) {
            if (xml.child(i).isTag() && xml.child(i).label().equals(tag)) {
                firstOccurenceldx = i;
            }
            i++;
        }
        return firstOccurenceldx;
    }

    /**
     * Processes one news item and outputs one table row. The row contains three
     * elements: the publication date, the source, and the title (or
     * description) of the item.
     *
     * @param item
     *      the news item
     * @param out

```

```

*      the output stream
* @updates out.content
* @requires [the label of the root of item is an <item> tag] and
*      out.is_open
* @ensures <pre>
* out.content = #out.content *
* [an HTML table row with publication date, source, and title of news item]
* </pre>
*/
private static void processItem(XMLTree item, SimpleWriter out) {
    assert item != null : "Violation of: item is not null";
    assert out != null : "Violation of: out is not null";
    assert item.isTag() && item.label().equals("item") : ""
        + "Violation of: the label root of item is an <item> tag";
    assert out.isOpen() : "Violation of: out.is_open";

    // <tr>
    out.println("<tr>");

    /*
    * Checks to see if there is a "pubDate" tag. If so, prints the content
    * inside. Otherwise, prints "No date available."
    */
    int pubDateIdx = getChildElement(item, "pubDate");
    String pubDateContent = item.child(pubDateIdx).child(0).label();
    if (pubDateIdx == -1) {
        out.println("<td>No date available</td>");
    } else {
        out.println("<td>" + pubDateContent + "</td>");
    }

    /*
    * Checks to see if there is a "source" tag. If so, prints the attribute
    * value of "url" - the link. Otherwise, prints "No source available."
    */
    int sourceIdx = getChildElement(item, "source");
    if (sourceIdx == -1) {
        out.println("<td>No source available</td>");
    } else {
        out.println("<td><a href=\"\"
            + item.child(sourceIdx).attributeValue("url") + "\">"
            + item.child(sourceIdx).child(0).label() + "</a></td>");
    }
}

```

```

// <td>
out.println("<td>");

    int linkIdx = getChildElement(item, "link");
    String linkContent = item.child(linkIdx).child(0).label();
    if (linkIdx != -1) {
        out.print("<a href=\"" + linkContent + "\">");
    }

    int titleIdx = getChildElement(item, "title");
//    String titleContent = item.child(titleIdx).child(0).label();
    if (titleIdx != -1) {
        if (item.child(titleIdx).numberOfChildren() == 0) {
            int descriptionIdx = getChildElement(item, "description");
            if (descriptionIdx != -1) {
                if (item.child(descriptionIdx).numberOfChildren() == 0) {
                    out.print("No description available");
                } else {
                    String descriptionContent = item.child(descriptionIdx)
                        .child(0).label();
                    out.print(descriptionContent);
                }
            }
        } else {
            out.print(item.child(titleIdx).child(0).label());
        }
    }

    if (linkIdx != -1) {
        out.println("</a>");
    }

// </td>
out.println("</td>");
// </tr>
out.println("</tr>");
}

/**
 * Main method.
 *
 * @param args
 *      the command line arguments; unused here
 */

```



```

public static void main(String[] args) {
    /*
     * Open input and output streams.
     */
    SimpleReader in = new SimpleReader1L();
    SimpleWriter out = new SimpleWriter1L();

    //  /*
    //  * Prompt the user to enter the URL of an XML RSS (Really Simple
    //  * Syndication) (Version 2.0) news feed.
    //  */
    //  out.print(
    //      "Enter the URL of an RSS (Really Simple Syndication) (Version 2.0) news feed: ");
    //  String userUrl = in.nextLine();
    //
    //  /*
    //  * Initialize an XMLTree object from a given RSS (Really Simple
    //  * Syndication) (Version 2.0) News Feed. If successful, the input is a
    //  * valid XML document.
    //  */
    //  XMLTree xml = new XMLTree1(userUrl);
    //
    //  /*
    //  * Check the root of the XMLTree to satisfy the condition that the input
    //  * provided is indeed a valid RSS (Really Simple Syndication) (Version
    //  * 2.0) news feed.
    //  */
    //  if (xml.label().equals("rss") && xml.hasAttribute("version")
    //      && xml.attributeValue("version").equals("2.0")) {
    //
    //      /*
    //      * Prompt the user to enter the name of an output file including the
    //      * .html extension.
    //      */
    //      out.print(
    //          "Enter the name of an output file including the .html extension: ");
    //      String fileName = in.nextLine();
    //
    //      /*
    //      * Prepare the output stream to output to a .html file.
    //      */
    //      SimpleWriter fileOut = new SimpleWriter1L(fileName);
    //      XMLTree channel = xml.child(0);
    //

```

```

//      // outputHeader method call:
//      outputHeader(channel, fileOut);
//
//      int i = 0;
//      while (i < channel.numberOfChildren()) {
//          if (channel.child(i).label().equals("item")) {
//              XMLTree item = channel.child(i);
//              // processItem method call:
//              processItem(item, fileOut); // prints a row.
//          }
//          i++;
//      }
//
//      // outputFooter method call:
//      outputFooter(fileOut);
//
//      } else {
//          out.println(
//              "ERROR - The provided URL is not a valid RSS (Really Simple Syndication) (Version
2.0) XML document.");
//      }

```

```

/*
 * Prompt the user to enter the URL of an XML RSS (Really Simple
 * Syndication) (Version 2.0) news feed.
 */
out.print(
    "Enter the URL of an RSS (Really Simple Syndication) (Version 2.0) news feed: ");
String userUrl = in.nextLine();

```

```

/*
 * Initialize an XMLTree object from a given RSS (Really Simple
 * Syndication) (Version 2.0) News Feed. If successful, the input is a
 * valid XML document.
 */
XMLTree xml = new XMLTree1(userUrl);

if (xml.label().equals("feeds") && xml.hasAttribute("title")) {
    /*
     * Prompt the user to enter the name of an output file including the
     * .html extension.
     */
    out.print(
        "Enter the name of an output file including the .html extension: ");
}

```

```

String fileName = in.nextLine();

/*
 * Prepare the output stream to output to a .html file.
 */
SimpleWriter fileOut = new SimpleWriter1L(fileName);

/*
 * Beginning tag nodes:
 */
fileOut.println("<html>");
fileOut.println("<head>");
fileOut.println(
    "<title>" + xml.attributeValue("title") + "</title>");
fileOut.println("</head>");
fileOut.println("<body>");
fileOut.println("<h1>" + xml.attributeValue("title") + "</h1>");
fileOut.println("<ul>");

int i = 0;
while (i < xml.numberOfChildren()) {
    if (xml.child(i).label().equals("feed")) {
        /*
         * Initialization of variables:
         */
        String url = xml.child(i).attributeValue("url");
        String name = xml.child(i).attributeValue("name");
        String file = xml.child(i).attributeValue("file");

        /*
         * Prepare the output stream.
         */
        SimpleWriter feed = new SimpleWriter1L(file);

        // processFeed method call:
        processFeed(url, file, feed);

        fileOut.println("<li><a href=\"\" + file + \"\">" + name
            + "</a></li>");
        /*
         * Close the output stream.
         */
        feed.close();
    }
}

```

```
        i++;
    }

    /*
     * Ending tag nodes:
     */
    fileOut.println("</ul>");
    fileOut.println("</body>");
    fileOut.println("</html>");

    /*
     * Close the output stream.
     */
    fileOut.close();

}
/*
 * Close input and output streams.
 */
in.close();
out.close();
}
}
```