```
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 Gabe Azzarita
 *
 */
public final class RSSReader {

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

    /*-
     * 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";

    out.println("<html>");
    out.println("<head>");
    out.println("<title>" + channel.child(0).child(0) + "</title>");
    out.println("</head>");
    out.println(" <h1>" + "<a href=" + channel.child(1).child(0) + ">"
            + channel.child(0).child(0) + "</a>" + "</h1>");

    //Description is not guaranteed to have child
    int descIndex = getChildElement(channel, "description");
    if (channel.child(descIndex).numberOfChildren() > 0) {
        out.println(" <p>" + channel.child(descIndex).child(0) + "</p>");
    } else {
        out.println(" <p> No description avaliable </p>");
    }

    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 childAt = -1;
    // Run through all children of xml, checking for desired tag
    for (int i = 0; i < xml.numberOfChildren(); i++) {
        if (tag.equals(xml.child(i).label())) {
            childAt = i;
        }
    }
    return childAt;
}

/**
 * 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";

    // Get index number for each desired component to use later
    int titleIndex = getChildElement(item, "title");
    int descIndex = getChildElement(item, "description");
    int pubDateIndex = getChildElement(item, "pubDate");
    int sourceIndex = getChildElement(item, "source");
    int linkIndex = getChildElement(item, "link");

    //Check for pubDate and source, not guaranteed
    if (pubDateIndex >= 0) {
        out.println(
                "   <td>" + item.child(pubDateIndex).child(0) + "</td>");
    } else {
        out.println("   <td> No date avaliable </td>");
    }

    if (sourceIndex >= 0) {
        String sourceLink = item.child(sourceIndex).attributeValue("url");
        if (item.child(sourceIndex).numberOfChildren() == 0) {
            out.println("   <td><a href=" + sourceLink + ">" + sourceLink
                    + "</a></td>");
        } else {
            out.println("   <td><a href=" + sourceLink + ">"
                    + item.child(sourceIndex).child(0) + "</a></td>");
        }
    } else {
        out.println("   <td> No source avaliable </td>");
    }

    // Check for title, if no title check for description, one is required
    if (item.child(0).label().equals("title")) {
        if (linkIndex == -1) {
            out.println("   <td> No link avaliable </td>");
        } else {
            out.println("   <td><a href=" + item.child(linkIndex).child(0)
                    + ">" + item.child(titleIndex).child(0) + "</a></td>");
        }
    } else if (item.child(0).label().equals("description")) {
        if (linkIndex == -1) {
            out.println("   <td> No link avaliable </td>");
        } else {
            out.println("   <td><a href=" + item.child(linkIndex).child(0)
                    + ">" + item.child(descIndex).child(0) + "</a></td>");
        }
    }
}
```

```
    /**
     * Main method.
     *
     * @param args
     *            the command line arguments; unused here
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();
        SimpleWriter out2 = new SimpleWriter1L("Index.HTML");

        // Create xml object
        out.print("Enter the URL of an RSS 2.0 news feed: ");
        String url = in.nextLine();
        XMLTree xml = new XMLTree1(url);
        XMLTree channel = xml.child(getChildElement(xml, "channel"));

        outputHeader(channel, out2);

        /*
         * Runs through all children of channel, if child is an item tag then we
         * processItem and add a new row to the table
         */
        for (int i = 0; i < channel.numberOfChildren(); i++) {
            if (channel.child(i).label().equals("item")) {
                out2.println("  <tr>");
                processItem(channel.child(i), out2);
                out2.println("  </tr>");
            }
        }

        outputFooter(out2);

        in.close();
        out.close();
    }

}
```