```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 an XML RSS (version 2.0) feed from a given URL into the
 * corresponding HTML output file.
 *
 * @author VishalKumar
 *
 */
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";

        // Get positions of child, link description
        int titlePos = getChildElement(channel, "title");
        int linkPos = getChildElement(channel, "link");
        int descPos = getChildElement(channel, "description");

        // print title
        if (channel.child(titlePos).numberOfChildren() > 0) {
            out.println("<html><head><title>"
                    + channel.child(titlePos).child(0).label() + "</title>");
        } else {
            out.println("<html><head><title>" + "Empty Title" + "</title>");
        }

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

        // Print 1st header with hyperLink to page
        out.print("<h1><a href=\"" + channel.child(linkPos).child(0).label()
                + "\">" + channel.child(titlePos).child(0).label()
                + "</a> </h1>");
        out.println("<p>");

        // print description of channel
        if (channel.child(descPos).numberOfChildren() > 0) {
            out.println(channel.child(descPos).child(0).label());
        } else {
            out.println("No description");
        }

        // print closing text of the header
        out.println("</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";

        // print closing text of the html page
        out.println("</table>");
        out.print("</body></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 pos = -1;
        // loop through tree and find first position of desired tag
        for (int i = 0; i < xml.numberOfChildren() && pos == -1; i++) {
            String name = xml.child(i).label();
            if (name.equals(tag)) {
                pos = i;
            }
        }
        //return the position
        return pos;
    }

    /**
     * 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 positions of date, source, title and link
        int datePos = getChildElement(item, "pubDate");
        int sourcePos = getChildElement(item, "source");
        int titlePos = getChildElement(item, "title");
        int linkPos = getChildElement(item, "link");
        int descPos = getChildElement(item, "description");

        out.print("<tr>");
        // print the date
        if (datePos != -1) {
            out.println(
                    "<td>" + item.child(datePos).child(0).label() + "</td>");
        } else {
            out.println("<td> No date availible </td>");
        }

        // print the source
        if (sourcePos != -1) {
            out.println("<td><a href=\""
                    + item.child(sourcePos).attributeValue("url") + "\">"
                    + item.child(sourcePos).child(0).label() + "</a></td>");
        } else {
            out.println("<td> No source availible </td>");
        }

        // print the article title or the article description with a hyperlink to the article
        if (titlePos != -1) {
            if (item.child(titlePos).numberOfChildren() > 0) {
                out.print("<td><a href=\""
                        + item.child(linkPos).child(0).label() + "\">");
                out.print(item.child(titlePos).child(0).label() + "</a></td>");
            } else {
                out.println("<td> No title availible </td>");
            }

        } else if (descPos != 1) {
            if (item.child(descPos).numberOfChildren() > 0) {
                out.print("<td><a href=\""
                        + item.child(linkPos).child(0).label() + "\">");
                out.print(item.child(descPos).child(0).label() + "</a></td>");
            } else {
                out.println("<td> No description availible </td>");
            }

        } else {
            out.println("<td> No title availible </td>");
        }
        out.print("</tr>");

    }

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

        // get url and file name from user
        out.print("Enter the URL of an RSS 2.0 feed: ");
        String url = in.nextLine();
        out.print(
                "Enter the name of an output file. Make sure to include the .html extension at the end!: ");
        String fileName = in.nextLine();

        // setup output steam to file
        SimpleWriter write = new SimpleWriter1L(fileName);

        // construct the xml tree objects
        XMLTree root = new XMLTree1(url);
        XMLTree channel = root.child(0);

        // print the header of the html page
        outputHeader(channel, write);

        // check to see if the tree is rss 2.0
        if (root.label().equals("rss")
                && root.attributeValue("version").equals("2.0")) {
            // loop through the channel tree and process each <item> tag
            int i = 0;
            while (i < channel.numberOfChildren()) {
                if (channel.child(i).label().equals("item")) {
                    processItem(channel.child(i), write);
                }
                i++;
            }
        } else {
            out.print("Link not valid RSS 2.0");
        }

        // print the footer of the html page
        outputFooter(write);

        // close input and output streams
        in.close();
        out.close();
        write.close();
    }

}
```