PRIFYSGOL BANGOR UNIVERSITY

Java Technologies Mini-Project 3

Working with XML

Cameron Gray

Introduction

This week, we will look at the use of Extensible Mark-up Language (XML) and how to process XML files in Java using either the Simple API for XML-Processing (SAX) or the Document Object Model (DOM) APIs. Both of these APIs provide parsers enabling Java applications to read, validate, and manipulate the data stored in an XML file. Both parsers are part of the Java API for XML Processing (JAXP).

DOM parsers and their use are described in Horstmann's *Big Java*. For information about SAX parsers see: http://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html.

To provide additional help I have provided some example programs which make use of a SAX parser. Note that the first example **`SAXParserExample`** uses the **`Employee`** class and the file **`employees.xml`**.

For those of you know very little about XML you should begin by looking at Chapter 22 in Horstmann's *Big Java* (4th ed.). When reading this chapter make sure that you distinguish between XML and the various Java APIs for processing XML documents. Also have a look at the Additional Notes in the Appendix.

Finally, when creating your own XML documents use an IDE to provide maximum support (e.g. syntax checking, formatting and code completion).

Thus NetBeans will generate XML files that, depending on user choice, are either:

- Well-formed XML documents
- DTD constrained documents
- XML schema constrained documents

To obtain any of the above right click on the package in which the XML files are to be stored and then select **new --> XML Document**. To begin with choose the default option for the nature of the XML document.

The reason XML is used on the Internet is its' universal and self-documenting nature. The description of the data is contained in the hierarchy of elements in the document and (if present) in the schema or type definition. This means that programs on different systems, written in different languages can consume the same data and 'understand' it in the same way: whether the data is presented in the same way and for the same purpose is neither here nor there.

Exercise 1: Consume the BBC Weather RSS Feed (15%)

RSS, short for *Really Simple Syndication*, is a type of XML data feed. It is popular with news, weather and other current affairs sites, such as blogs, because it allows them to offer their content in an easy to consume form. To begin work on this exercise visit http://www.bbc.co.uk/weather/, and use the **Find a Forecast** feature to locate the current weather for your preferred UK location. On each forecast page, you will find a set of social media and syndication icons, as shown below:

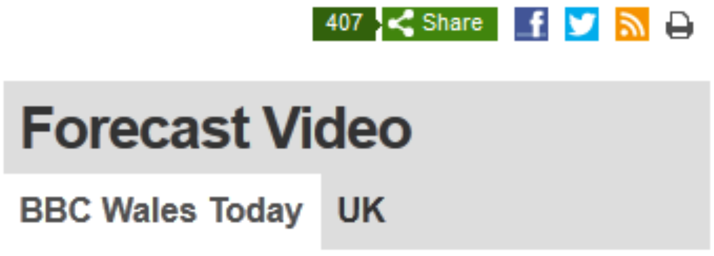


Figure 1 BBC Find a Forecast

The RSS Feed Icon is the orange one with the white dot and two white arcs. Click on the icon to make the Feeds menu appear, and select **Observations** which should cause a weather report to appear. Right-click on the associated link in the address box and copy it to the clipboard for later. For example, the URL for Bangor is

http://open.live.bbc.co.uk/weather/feeds/en/2656397/observations.rss.

Note that if you are using Google Chrome, the raw XML feed is displayed, however, Mozilla Firefox or Internet Explorer attempts to 'prettify' the output. To obtain raw XML right-click and select **View Page Source** from the context menu. You should get something similar to what appears below:

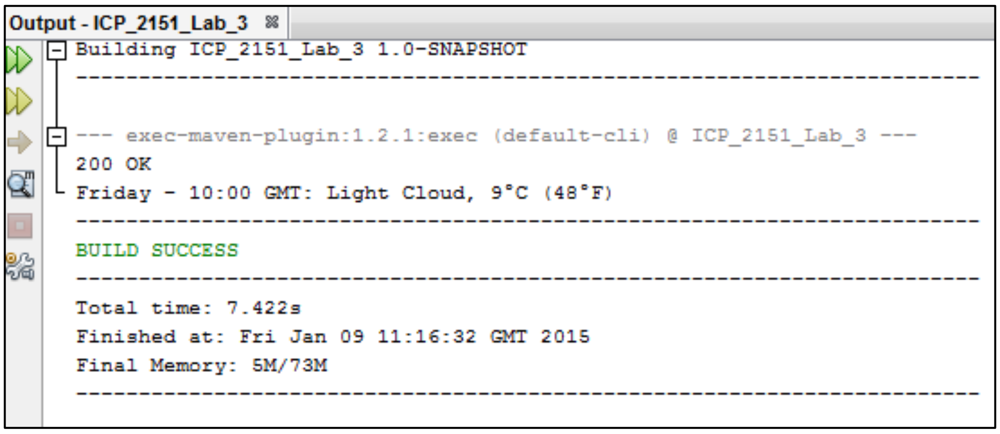
```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:georss="http://www.georss.org/georss" version="2.0">
<channel>
    <atom:link
ref="http://open.live.bbc.co.uk/weather/feeds/en/2656397/observations.rss"
rel="self" type="application/rss+xml" />
    <title>BBC Weather - Observations for  Bangor, United Kingdom</title>
    <link>http://www.bbc.co.uk/weather/2656397</link>
    <description>Latest observations for Bangor from BBC Weather, including
weather, temperature and wind information</description>
    <language>en</language>
    <copyright>Copyright: (C) British Broadcasting Corporation, see
http://www.bbc.co.uk/terms/additional_rss.shtml for more details</copyright>
    <pubDate>Fri, 09 Jan 2015 10:38:07 +0000</pubDate>
    <item>
      <title>Friday - 10:00 GMT: Light Cloud, 9°C (48°F)</title>
      <link>http://www.bbc.co.uk/weather/2656397</link>
      <description>Temperature: 9°C (48°F), Wind Direction: South South Westerly,
Wind Speed: 19mph, Humidity: 87%, Pressure: 1018mb, Rising, Visibility:
Good</description>
      <pubDate>Fri, 09 Jan 2015 10:38:07 +0000</pubDate>
      <guid isPermaLink="false">http://www.bbc.co.uk/weather/2656397-2015-01-
09T10:38:07.000Z</guid>
      <georss:point>53.22647 -4.13459</georss:point>
    </item>
</channel></rss>
```

Now that you have a URL and content, your task is to write a *command-line client* to retrieve the document, parse the XML and display the current observation headline (the text highlighted in red). The current headline is found in the `<rss >` `<channel >` `<item >` `<title>` element body.

Your client must be contained within a Maven project, even though you will not need any external dependencies.

You may not simply use String processing to extract this element from the retrieved URL.

The client should produce output similar to that below. Notice that the content selected for display belongs to the second occurrence of a <title> element; do not display the content of the first occurrence of this tag.



```
Building ICP_2151_Lab_3 1.0-SNAPSHOT
------------------------------------------------------------------------

--- exec-maven-plugin:1.2.1:exec (default-cli) @ ICP_2151_Lab_3 ---
200 OK
Friday - 10:00 GMT: Light Cloud, 9°C (48°F)
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 7.422s
Finished at: Fri Jan 09 11:16:32 GMT 2015
Final Memory: 5M/73M
------------------------------------------------------------------------
```

You will need to use the following class from the Java Class Library to access the site and recover the XML file.

`java.net.URL`

You will also need to use one of the following classes depending upon your choice of parser.

`org.xml.sax.helpers.DefaultHandler`

`javax.xml.parsers.DocumentBuilderFactory`

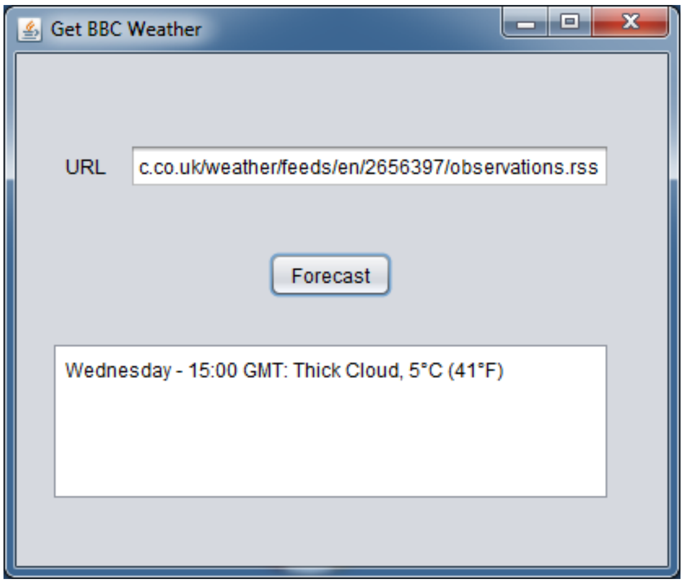
You may notice that depending on the type of Java XML parser you have chosen to employ, you do not need to explore every element and attribute in the XML document. Think about the differences between the types of parsers available and what that means when dealing with more complex or larger XML documents.

Exercise 2: Building a Weather App (20%)

Use the NetBeans GUI Builder (Matisse) to build a simple Swing based application to display the information recovered in the previous exercise. The GUI should consist of:

- a `JTextBox` for the URL;
- a `JButton` to obtain a forecast;
- a `JTextArea` to present the results.

The application you develop should have an interface looking something like the following:



```
Wednesday - 15:00 GMT: Thick Cloud, 5°C (41°F)
```

Figure 2 Getting the Forecast for Bangor

You will also need to alter the Maven POM to cause the resulting JAR to run your GUI class automatically. This will involve using the Maven Assembly Plugin, and is also known as 'creating an executable JAR'.

You will find instructions on using this plugin at

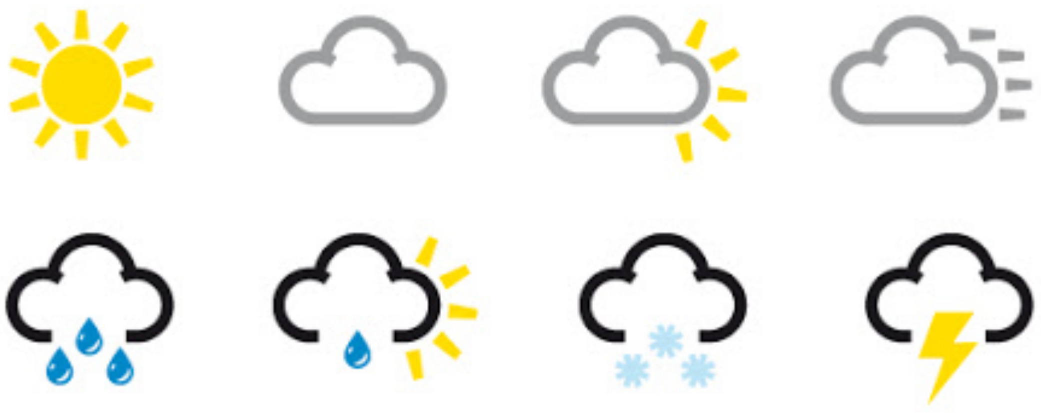
http://maven.apache.org/plugins/maven-assembly-plugin/usage.html.

Exercise 3: Add Weather Symbols (15%)

Using the text description of the weather condition generate a BBC style symbol for that condition and place this symbol onto the frame used for the GUI. The condition is contained in the **`<title>`** element and occurs after the colon symbol and is terminated by the first comma in the text. Thus in the element below the weather condition is "Light Cloud"

```
<title>Friday - 10:00 GMT: Light Cloud, 9°C (48°F)</title>
```

Some BBC style weather symbols are listed below:



Weather Symbols - From Top Left: Sunny, Cloudy, Light Clouds, Windy
Heavy Rain, Rain Showers/Light Rain, Snow, Lightning

When there is no symbol that matches the description, you will need to make a decision on how best to present the information to the user.

The images can be produced in a number of ways. For instance, you could build a local library of image files by searching the Web. These images can then be used build instances of the class **`ImageIcon`** which can then be used to construct a **`JLabel`** . Alternatively, you could use 2D graphics to produce your own set of weather symbols. If you decide on the second option speak with the lecturer to obtain further guidance.

To obtain a list of standard weather conditions consult the XML file below:

http://www.mikeafford.com/tv-graphics/projects/bbc-weather-symbols.html

Exercise 4: Adding Searching for Other Locations (20%)

As you may have noticed, the location information is not included as a name in the URL at the BBC. However each such geographic location must be uniquely identified in some way. The BBC uses the GeoNames ID system, which is highlighted in bold in the URL below:

http://open.live.bbc.co.uk/weather/feeds/en/**2656397**/observations.rss

In order to match names of specific locations to IDs (e.g. Bangor, Gwynedd to 2656397), you will need to register with the GeoNames service at http://www.geonames.org/login. Once your account details have been confirmed via the e-mail, you will need to enable the 'search' web service at http://www.geonames.org/manageaccount.

A description of the service can be found at http://www.geonames.org/export/geonames-search.html.

One way to use the GeoNames search service is to supply a URL with search parameters to your web browser. For example, the URL below will return a single search result, in English, for the 'place' or text 'london' and user 'eesa03' (i.e. Steve Marriott):

http://api.geonames.org/search?q=london&maxRows=1&lang=en&username=eesa03

This search returns;

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<geonames style="MEDIUM">
    <totalResultsCount>5699</totalResultsCount>
    <geoname>
        <toponymName>London</toponymName>
        <name>London</name>
        <lat>51.50853</lat>
        <lng>-0.12574</lng>
        <geonameId>2643743</geonameId>
        <countryCode>GB</countryCode>
        <countryName>United Kingdom</countryName>
        <fcl>P</fcl>
        <fcode>PPLC</fcode>
    </geoname>
</geonames>
```

Using this web service amend your weather application so that instead of getting the user to supply a lengthy URL he or she simply has to supply *the name of the location*, for which a weather forecast is desired. Obviously, this means that you should alter the GUI to accommodate this modification. Once the user has entered the location name, your program should pass it to the appropriate GeoNames service. The service will then return an XML document, which can be parsed so that the geonameId value marked in red, can be extracted and used to build the required URL for the BBC weather feed. For example, the URL for London would become;

http://open.live.bbc.co.uk/weather/feeds/en/**2643743**/observations.rss

You must remember to use your own username in the search URL supplied to GeoNames. Additionally, the GeoNames service will return IDs for more places than the BBC can return weather forecasts for, although forecasts are not limited to just the UK. When this happens the feed will return an HTTP 404 – Not Found error. You will need to handle this event gracefully; an error message would be appropriate.

Adjust your weather application to accommodate these changes.

Exercise 5: Additional Information from the Description (10%)

This exercise involves parsing and extracting the information provided in the `description` element of the BBC Weather RSS feed. In particular you will need to recover the *six* pieces of information highlighted in red below:

```
<description>Temperature: 9°C (48°F), Wind Direction: South South
Westerly, Wind Speed: 19mph, Humidity: 87%, Pressure: 1018mb,
Rising, Visibility: Good</description>
```

Having recovered this information you will need to find some suitable way of presenting it visually. Marks will be awarded for use of advanced Swing components such as `JTables`, `JTrees` and `JLists` or further Java2D drawing/image processing where appropriate. This aspect of the mini-project is free form, however the assessor will ask questions regarding your design choices. This means that you cannot simply decide to use the most complicated component possible to get good marks. The choice must be appropriate for the information you are trying to display.

Exercise 6: Serialising Search Data to XML (20%)

This final exercise requires you to serialise[1] all of the search data, and retrieved GeoNames IDs back into an XML file. The file needs to be *written to* as each search is completed so that the information doesn't need to be held in memory. You should use the Streaming API for XML (StAX) in Java to act

[1] Object serialization is the process of saving an object's state to a sequence of bytes, as well as the process of rebuilding those bytes into a live object at some future time.

http://www.oracle.com/technetwork/articles/java/javaserial-1536170.html as both a reader (i.e. read from an XML document) and a writer (write to an XML document). You'll find Oracle's tutorial for StAX at http://docs.oracle.com/javase/tutorial/jaxp/stax/index.html.

The XML document you are going to create should conform to the basic structure shown below (the elements emphasised in italics must be replaced with the actual values):

```
<weatherSearches>
    <search date="date search ran">
        <term>entered search term</term>
        <found>true/false</found>
        <geoNameID>filled in if found</geoNameID>
    </search>
    …
</weatherSearches>
```

The <search /> element is repeated each time the user runs another search. You should ensure that the file is updated after each write, so that your code can be tested completely. If the file you have chosen to use already exists, you can erase the contents of the file on each new execution of your program. You will need to handle the fact that your program is running from a JAR and so will not necessarily have a file system location to use. There is a suitable method in the `java.io.File` class to assist in creating temporary files safely. You will also need to find a suitable way to inform the user of the location of this file.

Useful Links

In addition to looking at Horstmman I strongly recommend the following two sites for more information about XML.

http://www.ibm.com/developerworks/xml/

http://www.w3schools.com/DTD/default.asp

Another important source of information is the W3 consortium site which maintains a section devoted to XML. The home page is listed below:

http://www.w3.org/XML/

Perhaps the most useful area at W3 can be found at

http://www.w3.org/standards/xml/

This link will take you to an authoritative description of the XML and the various standards associated with this markup language.

For an interesting article on XML design by Uche Ogbuji see

http://www.ibm.com/developerworks/xml/library/x-eleatt.html

For information about the various Java APIs associated with XML checkout the following links:

http://docs.oracle.com/javase/tutorial/jaxp/index.html

http://www.ibm.com/developerworks/java/library/x-jaxp/

Submission

Use **Blackboard** to submit your source code files. Each source code file must

- Contain a program header
- Contain an appropriate level of Javadoc style comments
- Follow a consistent style of indentation
- Follow the usual Java conventions for class and variable names

Failure to comply with the above will result in a marks penalty of 5%.

The deadline for submitting your work is published on Blackboard. Late submissions will be penalised in line with School policy.

When submitting work it is your responsibility to ensure that all work submitted is

- Consistent with stated requirements
- Entirely your own work
- On time

Please note that there are **severe penalties** for submitting work which is not your own. *If you have used code that you have found on the Internet or from any other source* then you **must** signal that fact with appropriate program comments. Note also that to obtain a mark you **must** attend a lab session and be prepared to demonstrate your program and to answer questions about the coding. Non-attendance at labs will result in your work not being marked.

Appendix 1: Namespaces

Due to the fact that various XML authors may make use of the same tags to represent different data (especially since quite often a word will have multiple possible meanings), naming collisions can occur. In order to prevent such collisions, we typically declare our XML tags as belonging to a namespace. For example, imagine the following XML document representing a university, which has many courses and many students:

```
<?xml version="1.0"?>
<university>
      <student>
            <id>6</id>
            <name>John Smith</name>
            <average-grade>67</average-grade>
      </student>
      <course>
            <id>5</id>
            <name>Computing Laboratory 2</name>
            <credits>30</credits>
      </course>
</university>
```

In the above XML document, there are two collisions – one between `id` tags and another between `name` tags. In the first instance, our id and name tags are in reference to a student ID and a student name whereas, in the second we are referring to a course ID and course name. This would likely cause confusion when attempting to validate our XML documents against a DTD as it would appear (by choosing the same name for these tags) that the id and name tags should store the same information whereas in reality, this is not what we want. In order to distinguish between these, we can use namespaces e.g.

```
<?xml version="1.0"?>
<university    xmlns:student    =    "http://www.bangor.ac.uk/xmlns-student"
xmlns:course = "http://www.bangor.ac.uk/xmlns-course">
      <student:student>
            <student:id>6</id>
            <student:name>John Smith</name>
            <student:average-grade>67</average-grade>
      </student:student>
      <course:course>
            <course:id>5</id>
            <course:name>Computing Laboratory 2</name>
            <course:credits>30</credits>
      </course:course>
</university>
```

In the above example, we have used two namespaces in order to ensure that our tags are now unique in meaning – naming collisions no longer occur and we cannot confuse one tag for another. The second line of the XML document, which would normally be used to declare the root element of an XML document now also declares two namespaces – `student` and `course`. We assign each namespace a URL purely in order to ensure that these namespaces are unique. The theory is that since we are the only one to which a particular URL belongs, by using our URL as part of our namespace, this should ensure that our namespace is truly unique – using the Bangor University is probably not the best example since in theory every student / staff member of the university might make use of this domain in order to define a namespace so it would probably be better to make use of a more personal domain name should you happen to own one.

Whilst URLs are very commonly used in practice in order to make namespaces unique, we are in fact allowed to use any URI i.e. we are not limited to URLs. Notice that

```
http://www.bangor.ac.uk/xmlns-student
```

and

```
http://www.bangor.ac.uk/xmlns-course
```

aren't actually URLs which exist, the URL is included as part of the namespace purely for the purposes of making it unique, for the same reason that Java packages are also typically named after (reversed) domains e.g. `uk.ac.bangor`.

Having declared our namespaces, we simply prefix each tag with the namespace to which it should belong.