Writing XML

Let's create a simple XML file. We'll use the example with the users from the last lesson. We'll create a new project named `XmlSaxWriting`, and add a new class to the project:

```
public class User {

    private String name;
    private int age;
    private LocalDate registered;
    public static DateTimeFormatter dateTimeFormatter =
DateTimeFormatter.ofPattern("M/d/yyyy");

    public User(String name, int age, LocalDate registered) {
        this.name = name;
        this.age = age;
        this.registered = registered;
    }

    @Override
    public String toString() {
           return getName();
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public LocalDate getRegistered() {
        return registered;
    }

}
```

For simplicity's sake, we'll write the code right in the `main()` method. All we're really doing is testing out SAX's functionality. At this point, you should already know how to design object-oriented applications properly.

We create an `XMLStreamWriter` using `XMLOutputFactory`. Of course, we could only store a single object to XML (e.g. some settings). Here, we'll learn how to store a list of several objects. If you only want to store one object, you'll only need to make very minor changes

First, let's create a list of some test users:

```
ArrayList<User> users = new ArrayList<>();
LocalDate date1 = LocalDate.of(2000, Month.MARCH, 21);
LocalDate date2 = LocalDate.of(2012, Month.OCTOBER, 30);
LocalDate date3 = LocalDate.of(2011, Month.JANUARY, 1);
users.add(new User("John Smith", 22, date1));
users.add(new User("James Brown", 31, date2));
users.add(new User("Tom Hanks", 16, date3));
```

Now we'll create an `XMLStreamWriter` instance using `XMLOutputFactory`, which itself is created by the `newInstance()` factory method. Unfortunately, we can't use a try-with-resources block because `XMLStreamWriter` doesn't support it. We pass the instance as a parameter to `FileWriter`, as we did with text files.

```
XMLOutputFactory xof = XMLOutputFactory.newInstance();
XMLStreamWriter xsw = null;
try{
    xsw = xof.createXMLStreamWriter(new FileWriter("file.xml"));
}
catch (Exception e){
        System.err.println("Unable to write the file: " + e.getMessage());
}
finally{
    try{
        if (xsw != null){
                xsw.close();
        }
    }
    catch (Exception e){
            System.err.println("Unable to close the file: " +
e.getMessage());
    }
}
```

Since we can't use try-with-resources, the code is a bit complicated;

Now, let's get to the actual writing. First, let's add in the document header:

```
xsw.writeStartDocument();
```

Then (as you should know by now) the root element has to follow which contains the rest of the XML. We use the `writeStartElement()` and `writeEndElement()` methods for writing elements. The first method takes the name of the element we're opening as a parameter. The second method determines the element name on its own from the document context and it doesn't have any parameters. Let's open the root element, which is the `<users>` element in our case:

```
xsw.writeStartElement("users");
```

Next, we'll move on to writing individual users so the code can be placed in a foreach loop.

We write the value to the element using the `writeCharacters()` method, which takes its value as a parameter. Similarly, we can add an element attribute using the `writeAttribute()` method, whose parameters are the attribute name and its value. The value is always of the `String` type, so we have to convert the age to a `String` in our case. Looping and writing the `<user>` elements looks like this (without the nested elements):

```
for (User u : user)
{
    xsw.writeStartElement("user");
    xsw.writeAttribute("age", Integer.toString(u.getAge()));
    xsw.writeEndElement();
}
```

We'll add one more `endElement()` to close the root element and `endDocument()` to close the whole document. Like with text files, we have to empty the buffer using the `flush()` method. The entire application code now looks like this:

```
//  a collection of test users
ArrayList<User> users = new ArrayList<>();
LocalDate date1 = LocalDate.of(2000, Month.MARCH, 21);
LocalDate date2 = LocalDate.of(2012, Month.OCTOBER, 30);
LocalDate date3 = LocalDate.of(2011, Month.JANUARY, 1);
users.add(new User("John Smith", 22, date1));
users.add(new User("James Brown", 31, date2));
users.add(new User("Tom Hanks", 16, date3));

// writes the users
XMLOutputFactory xof = XMLOutputFactory.newInstance();
XMLStreamWriter xsw = null;
try {
    xsw = xof.createXMLStreamWriter(new FileWriter("file.xml"));
    xsw.writeStartDocument();
    xsw.writeStartElement("users");

    for (User u : users) {
        xsw.writeStartElement("user");
        xsw.writeAttribute("age", Integer.toString(u.getAge()));
        xsw.writeEndElement();
    }

    xsw.writeEndElement();
    xsw.writeEndDocument();
    xsw.flush();
}
catch (Exception e) {
        System.err.println("Unable to write the file: " + e.getMessage());
}
finally {
    try {
        if (xsw != null) {
                xsw.close();
        }
    }
    catch (Exception e) {
            System.err.println("Unable to close the file: " +
e.getMessage());
    }
}
```

Let's run the program and make sure that everything works. The output of the program should look like this (`program folder/file.xml`):

```
<?xml version="1.0" ?><users><user age="22"></user><user
age="31"></user><user age="16"</user></users>
```

The data looks fine, but there is no formatting. Let's fix it. We'll create a new method named `format(String file)` under the `main()` method. It'll accept the path to the file to format as a parameter.

```
private static void format(String file) {
    // TODO we'll write the formatter body here
```

```
}
```

Add we'll add the following lines to the method body:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse(new InputSource(new InputStreamReader(new
FileInputStream(file))));

// Gets a new transformer instance
Transformer xformer = TransformerFactory.newInstance().newTransformer();
// Sets XML formatting
xformer.setOutputProperty(OutputKeys.METHOD, "xml");
// Sets indent
xformer.setOutputProperty(OutputKeys.INDENT, "yes");
Source source = new DOMSource(document);
Result result = new StreamResult(new File(file));
xformer.transform(source, result);
```

The resulting format of the file should look like this:

```
<?xml version="1.0" ?>
<users>
  <user age="22"></user>
  <user age="31"></user>
  <user age="16"></user>
</users>
```

We can see that SAX recognized that there is no value in the `<user>` element, except for an attribute, and generated this element as unpaired. Now, let's add 2 additional elements into the `<user>` element, moreover, their name and the registration date fields. To do this, we'll create a static `dateTimeFormatter` on the `User` class:

```
public static DateTimeFormatter dateTimeFormatter =
DateTimeFormatter.ofPattern("M/d/yyyy");
```

Making it a static member is ok here since it's an auxiliary formatter for working with a class field, which we can make publicly accessible.

And we'll add the following to the loop code:

```
xsw.writeStartElement("name");
xsw.writeCharacters(u.getName());
xsw.writeEndElement();
xsw.writeStartElement("registered");
xsw.writeCharacters(User.dateTimeFormatter.format(u.getRegistered()));
xsw.writeEndElement();
```

We'll put the code where we're witing the `<user>` element, that is, between its `writeAttribute()` and `writeEndElement()`. To be sure, let's show the complete code of the loop section:

```
for (User u : users) {
    xsw.writeStartElement("user");
    xsw.writeAttribute("age", Integer.toString(u.getAge()));
    xsw.writeStartElement("name");
```

```
    xsw.writeCharacters(u.getName());
    xsw.writeEndElement();
    xsw.writeStartElement("registered");
    xsw.writeCharacters(User.dateTimeFormatter.format(u.getRegistered()));
    xsw.writeEndElement();
    xsw.writeEndElement();
}
```

That's it!