Object Oriented Programming 2

Rolf Haenni & Andres Scheidegger

Exercises 5

1. Marshal and Unmarshal Person Data with Annotations

Open a new Eclipse Java project. Import the *Person* class and the *MaritalStatus* enum from Exercise 1. Add a new class *Persons*, which contains a list of Person objects. Add also a class *PersonRegistry* with a main method. In this method create a *Persons* object and add several *Person* objects to it.

Now annotate your *Persons* and your *Person* classes with the JAXB annotations necessary to write and read all person data to / from an XML file.

In the main method create a JAXB marshaller and marshal your Persons object to the console and to an XML file. Finally, create an unmarshaller and try to read in the XML file again and to print the person data to the console.

Play with your annotations and try to get as close as possible to the XML file for person data from Exercise 4, task 6.

Hint: You will note, that the XML type for dates is a different one, then we use in Java. Implement an appropriate *XMLAdapter* (see slides).

2. Marshal and Unmarshal Person Data with Annotations

This is basically the same as task 1, however a bit more complicated.

Use the given classes (directory *topic05\Sources\Exercise5Task2\src\university*) to create an instance of the class *University* and store it in an XML file. In a second step, try to read in again the generated XML file and show the content on the screen.

Given classes:

- UniMain.java (main class)
- University.java
- Person.java
- Student.java
- Employee.java
- Professor.java

The resulting XML file should look like to following example (next page):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<university>
    <name>bfh</name>
    <staff>
        <staffmember xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xsi:type="student">
            <name>Max</name>
            <email>max@bfh.ch</email>
            <grade>B</grade>
        </staffmember>
        <staffmember xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xsi:type="professor">
            <name>Mr. X</name>
            <email>prof.x@bfh.ch</email>
            <subject>IT Security</subject>
        </staffmember>
        <staffmember xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xsi:type="employee">
            <name>Erni Schmidt</name>
            <email>erni.schmidt@bfh.ch</email>
        </staffmember>
    </staff>
    <students>
        <student>
            <name>Max</name>
            <email>max@bfh.ch</email>
            <grade>B</grade>
        </student>
        <student>
            <name>Tom</name>
            <email>tom@bfh.ch</email>
            <grade>c</grade>
        </student>
    </students>
</university>
```

Hint: The Person class in an abstract class. Annotate the fields with *@XmlElement* to include them in the XML file.

Hint2: Use the annotation *@XmlSeeAlso* in the abstract class to establish the links in the class hierarchy.

3. Generate Java Classes from XML Schema

Use the given XML schema fax.xsd (directory *topic05\Sources\Exercise5Task3\Schema*) to generate the corresponding JAXB classes (use *xjc*). Write a program that creates a valid fax instance and marshals it. The result should look like the example on next page.

In a second step, try to validate and read in the generated XML file and show the content on the screen.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns1:fax xmlns:ns1="http://www.bfh.ch/fax/">
    <header>
        <from name="MyCompany" ns1:faxno="012 345 67 89"/>
        <to name="YourCompany" ns1:faxno="098 765 43 21"/>
        <priority>URGENT</priority>
        <pages>1</pages>
    </header>
    <body>This is the message in the body.</body>
</ns1:fax>
```

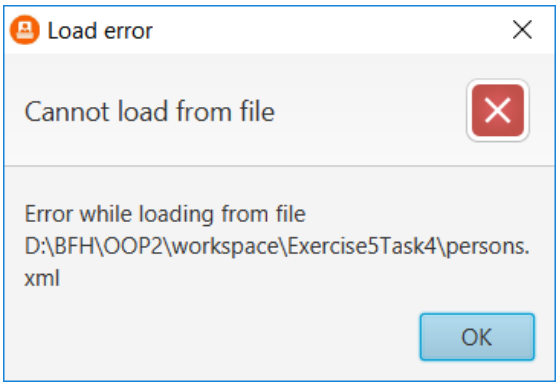
4. Combine JavaFX Application and JAXB

Now it is time to get back to our JavaFX Person Registry application as we left it back in Exercise 3. The goal is to add persistence of the person data via JAXB, based on the XML schema from Exercise 4, task 6.

Copy your application to a new JavaFX project and add a folder with the schema. Use *xjc* to create the Java classes. Now, implement a new *PersonDAO* using JAXB marshalling and unmarshalling. Change your application to use the new DAO.

Find out how you can set the *xsi:schemaLocation* attribute in the XML file where the person data is stored. Provide a valid path to the schema file and verify, that the XML file is validated in your Eclipse project (no warnings or errors shown on the generated XML file).

Enable also validation when unmarshalling the XML file and adapt your error handling code. If validation fails while unmarshalling the following *Alert* should be shown to the user:



At the same time the stack trace should be printed to the console.

5. PersonDAO with StAX

Write a StAX-based implementation of the *PersonDAO* interface. Integrate it with your Person Registry application and test it. In a first step do not use namespaces and schema. In a second step you may introduce a namespace and in a third step you may try validating against a schema.

6. Reading a RSS Feed with StAX

From Wikipedia, the free encyclopedia: http://en.wikipedia.org/wiki/RSS

« **RSS** (**Rich Site Summary**; originally **RDF Site Summary**; often called **Really Simple Syndication**) is a type of web feed[2] which allows users to access updates to online content in a standardized, computer-readable format. These feeds can, for example, allow a user to keep track of many different websites in a single news aggregator. The news aggregator will automatically check the RSS feed for new content, allowing the content to be automatically passed from website to website or from website to user. This passing of content is called web syndication. Websites usually use RSS feeds to publish frequently updated information, such as blog entries, news headlines, audio, video. An RSS document (called "feed", "web feed",[3] or "channel") includes full or summarized text, and metadata, like publishing date and author's name. A standard XML file format ensures compatibility with many different machines/programs. »

Task: Write a simple program, a RSS client, that retrieves a RSS feed from an URL, e.g. https://www.tagesanzeiger.ch/rss.html, and display the result on the screen.
Here an example from 17.3.2018.

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom">
      <channel>
            <title>Tages-Anzeiger Front</title>
            <link>http://tagesanzeiger.ch</link>
            <description>RSS von Tages-Anzeiger</description>
            <language>de-de</language>
            <copyright>Tages-Anzeiger</copyright>
            <atom:link href=http://tagesanzeiger.ch
                        rel="self" type="application/rss+xml" />
            <image>
                  <title>Tages-Anzeiger Front</title>
                  <link>http://tagesanzeiger.ch</link>
                  <url>https://files.newsnetz.ch//rss_logos/2.png</url>
            </image>
            <item>
                  <title>Der digitale Verteilkampf hat begonnen</title>
                  <description>
                        <![CDATA[Economiesuisse hat plötzlich etwas gegen
                        Privateigentum. Das Motiv ist klar. ]]>
                  </description>
                  <category><![CDATA[Video]]></category>
                  <link>http://tagesanzeiger.ch/wirtschaft/standard/Der-digitale-
                        Verteilkampf-hat-begonnen/story/31259561</link>
                  <guid isPermaLink="false">
                        http://tagesanzeiger.ch/31259561
                  </guid>
                  <pubDate>Sat, 17 Mar 2018 12:16:06 +0200</pubDate>
            </item>
            <!-- further items -->
      </channel>
</rss>
```

1. Analyse the structure of the XML document.
2. Write a `RSSFeedParser` based on StAX, that can be used with the 2 main programs `ReadTest.java` (read directly from an URL) and `ReadFileTest.java` (read form an xml file). Use the given classes `Feed.java` and `FeedMessage.java` (see *topic05\Sources\Exercise5Task6\src\rss*) to represent the RSS Feeds. Hint: the method `openStream()` of class `java.net.URL` creates an `InputStream` from an `URL` object.

7. Manipulating a DOM

This task shows the use of DOM classes to read and modify an XML document. An original XML document is read by the *XMLDomProcessing* class and its contents is printed out adding the node information of each text component. After that, some modifications are carried out on this document (see method *modifyDocument()*) and the modified document is printed out again.

1. Compile and run the original class (see *topic05\Sources\Exercise5Task7\src\sonnet*)
   There is not much to say to this task: Just compile the file and run it by passing it the file name *sonnet.xml*. Find out what is modified in the given xml document and how this is done.
   Be aware, that printing the DOM in this task is done "by hand", which is not advisable for productive code. See presentation for a better solution.

2. Modify the title element
   Change the title of the sonnet (in method *modifyDocument()*): Add some stars to the title, e.g.: "*** Sonnet 130 ***".

3. Insert new lines
   Change the lines of the sonnet: Insert before and after every line element a new line element, e.g. "---------".

4. Insert a comment
   Insert a comment before the author element. User the following command to create a comment element: *Comment comment = doc.createComment("A comment ...");*

8. PersonDAO with DOM

Write a DOM-based implementation of the *PersonDAO* interface. Integrate it with your Person Registry application and test it. Use namespaces and schema attributes in the generated instance document, when storing person data, and validate against the schema, when loading.