Data Binding with JAXB and EclipseLink

Data externalisation or marshalling is the process of converting a graph of objects into a format suitable for transmission between running processes. There are numerous forms of data exchange formats, typically specific to a given domain area, e.g. finance, medicine, biology. In the context of inter-process communication (IPC) however, there are really only three modes of externalisation that we are concerned with:

- **Binary data formats**, e.g. the CDR format used by CORBA. These have the benefit of being highly structured, highly compact and highly efficient. However, the scoping and development of binary standards also highly laborious and difficult to implement for a heterogeneous distributed system.
- **Semi-compiled formats**, e.g. the serialized forms used in Java and C#. Serialization is highly efficient and flexible, but only works in a homogenous distributed system, i.e. a J2EE or .NET platform.
- **Unicode Formats**, e.g. CSV, XML and JSON. These forms are the least efficient but most flexible forms of data externalization and are ideal as "glueware" in heterogeneous systems. In particular, JSON and XML formats combine the flexibility and portability of Unicode formats with a very fine-grained degree of structure.

In this example we will see how an XML schema document can be used as a DDL (Data Definition Language) for creating a platform and language neutral mechanism for data exchange. We are going to use the JAXB API (Java Architecture for XML Processing, part of the core Java SDK) to generate a suite of Java classes from an XML schema. We will then use the framework to convert objects to XML and JSON and vice versa. The specific implementation that we will use is EclipseLink (http://www.eclipse.org/eclipselink/).

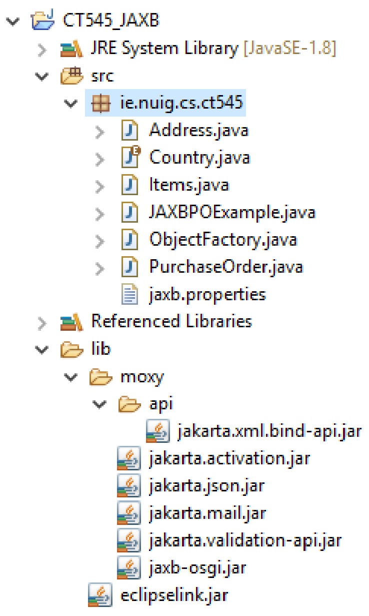
XML Schema 1.1. is a recommended standard from the W3C consortium (www.w3c.org). A schema allows us to create a grammar for a legal XML document, i.e. to specify the sequence and multiplicity of elements and the legal types to use. You should familiarise yourself with the standard at https://www.w3.org/XML/Schema , in particular, the Specifications and Development section (especially XML Schema Part 2: Datatypes).

1. Open the XML Schema po.xsd in a text editor and examine the structure of the document. Much like a programming language, the XML schema standard allows us to build complex types from a small set of simple types. These simple types can be translated into programming types use a set of language bindings. Note that the namespaces in the schema point at http://cs.nuig.ie/ct545/. Also note that the complex type *Items* has 1 : n relationship with an Item.

2. Open a command prompt in the CT545_1920_DataBindingExample directory containing the schema po.xsd and compile the schema using the XML to Java Converter tool (XJC) with the following command:
   ```
   xjc -npa po.xsd
   ```
   The schema compiler tool generates a Java class for each complexType defined in the po.xsd document. The members of each generated Java class are the same as the elements inside

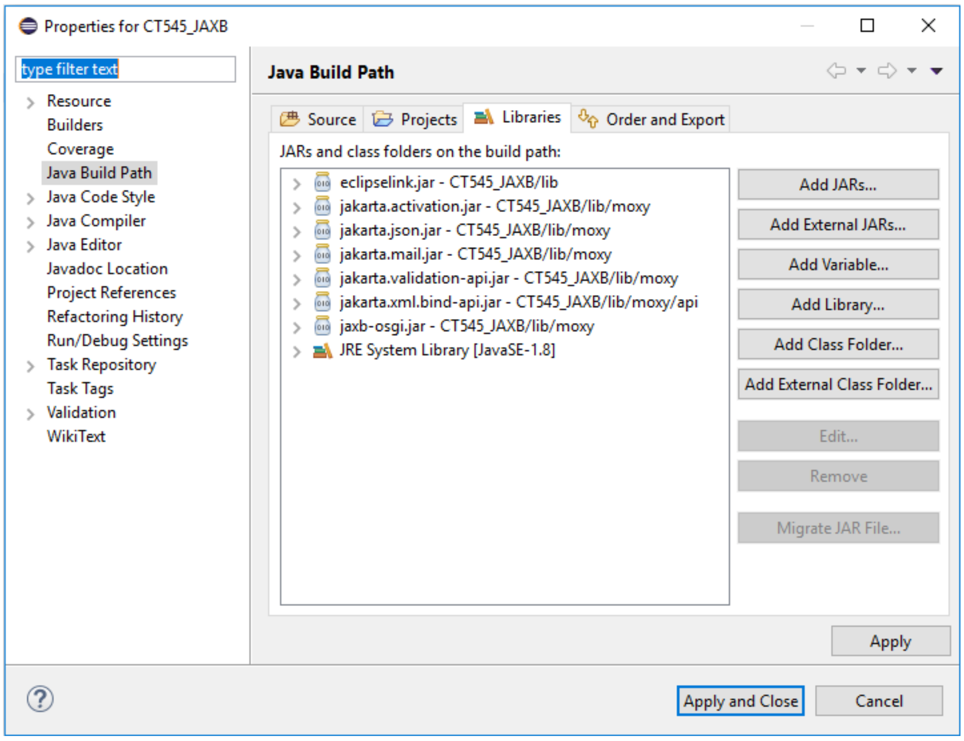
the corresponding complexType, and the class contains getter and setter methods for these fields. The generated classes include an object factory for manufacturing instances of PurchaseOrder.

3. Create a new Eclipse (Standard Edition) project and copy in all the files in the ie.nuig.cs.ct545 folder into the src folder of your project. Then copy the lib folder into your project directory. You should have the following folder structure after you have completed these steps:



```
CT545_JAXB
  JRE System Library [JavaSE-1.8]
  src
    ie.nuig.cs.ct545
      Address.java
      Country.java
      Items.java
      JAXBPOExample.java
      ObjectFactory.java
      PurchaseOrder.java
      jaxb.properties
  Referenced Libraries
  lib
    moxy
      api
        jakarta.xml.bind-api.jar
      jakarta.activation.jar
      jakarta.json.jar
      jakarta.mail.jar
      jakarta.validation-api.jar
      jaxb-osgi.jar
    eclipselink.jar
```

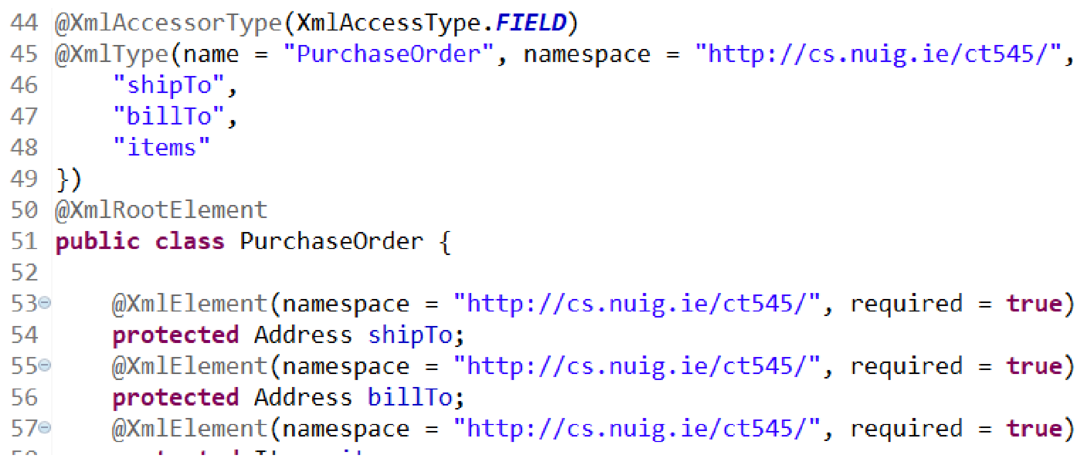
4. Right click on your project title -> Properties -> Java Build Path -> Libraries. Add the 7 libraries in the lib folder to the classpath of the project.



JARs and class folders on the build path:

```
eclipselink.jar - CT545_JAXB/lib
jakarta.activation.jar - CT545_JAXB/lib/moxy
jakarta.json.jar - CT545_JAXB/lib/moxy
jakarta.mail.jar - CT545_JAXB/lib/moxy
jakarta.validation-api.jar - CT545_JAXB/lib/moxy
jakarta.xml.bind-api.jar - CT545_JAXB/lib/moxy/api
jaxb-osgi.jar - CT545_JAXB/lib/moxy
JRE System Library [JavaSE-1.8]
```

5. Open the file PurchaseOrder.java and add the annotation @XmlRootElement to the generated class and the import javax.xml.bind.annotation.XmlRootElement.



```java
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "PurchaseOrder", namespace = "http://cs.nuig.ie/ct545/",
    "shipTo",
    "billTo",
    "items"
})
@XmlRootElement
public class PurchaseOrder {

    @XmlElement(namespace = "http://cs.nuig.ie/ct545/", required = true)
    protected Address shipTo;
    @XmlElement(namespace = "http://cs.nuig.ie/ct545/", required = true)
    protected Address billTo;
    @XmlElement(namespace = "http://cs.nuig.ie/ct545/", required = true)
```

6. Open the class JAXBPOExample.java and examine the import statements and code in the main() method. The class creates an instance of the generated class PurchaseOrder and then marshals and unmarshals the state of the object graph in XML and JSON format.

7. Run the main method in the JAXBPOExample.java class. Examine the content of the screen output and the XML and JSON files generated in the process.

8. Extra - execute the Python script JAXBPOExample.py to prove that the JSON format is language neutral: python JAXBPOExample.py

Example order.xml output

```xml
<?xml version="1.0" encoding="UTF-8"?>
<purchaseOrder xmlns:ns0="http://cs.nuig.ie/ct545/"
orderNumber="55522-BABA" orderDate="2967-04-03+01:00">
   <ns0:shipTo country="Ireland">
      <ns0:name>John Doe</ns0:name>
      <ns0:street>123 Castle Road</ns0:street>
      <ns0:city>Oranmore</ns0:city>
      <ns0:county>Galway</ns0:county>
   </ns0:shipTo>
   <ns0:billTo country="Ireland">
      <ns0:name>John Doe</ns0:name>
      <ns0:street>123 Castle Road</ns0:street>
      <ns0:city>Oranmore</ns0:city>
      <ns0:county>Galway</ns0:county>
   </ns0:billTo>
   <ns0:items>
      <ns0:item partNumber="123ABC">
         <ns0:productName>11ft Trout Fly Road
         </ns0:productName>
         <ns0:quantity>1</ns0:quantity>
         <ns0:price>250.00</ns0:price>
         <ns0:shipDate>2967-04-03+01:00</ns0:shipDate>
      </ns0:item>
      <ns0:item partNumber="177AAA">
         <ns0:productName>14ft Salmon Fly Road
         </ns0:productName>
         <ns0:quantity>1</ns0:quantity>
         <ns0:price>450.00</ns0:price>
         <ns0:shipDate>2967-04-03+01:00</ns0:shipDate>
      </ns0:item>
   </ns0:items>
</purchaseOrder>
```

Example order.json output

```
{
   "orderNumber" : "55522-BABA",
   "orderDate" : "2967-04-03+01:00",
   "shipTo" : {
      "country" : "Ireland",
      "name" : "John Doe",
      "street" : "123 Castle Road",
      "city" : "Oranmore",
      "county" : "Galway"
   },
   "billTo" : {
      "country" : "Ireland",
      "name" : "John Doe",
      "street" : "123 Castle Road",
      "city" : "Oranmore",
      "county" : "Galway"
   },
   "items" : {
      "item" : [ {
         "partNumber" : "123ABC",
         "productName" : "11ft Trout Fly Road",
         "quantity" : 1,
         "price" : 250.00,
         "shipDate" : "2967-04-03+01:00"
      }, {
         "partNumber" : "177AAA",
         "productName" : "14ft Salmon Fly Road",
         "quantity" : 1,
         "price" : 450.00,
         "shipDate" : "2967-04-03+01:00"
      } ]
   }
}
```