JAXB & Dozer

Laboratory of Service Design and Engineering

2011/2012

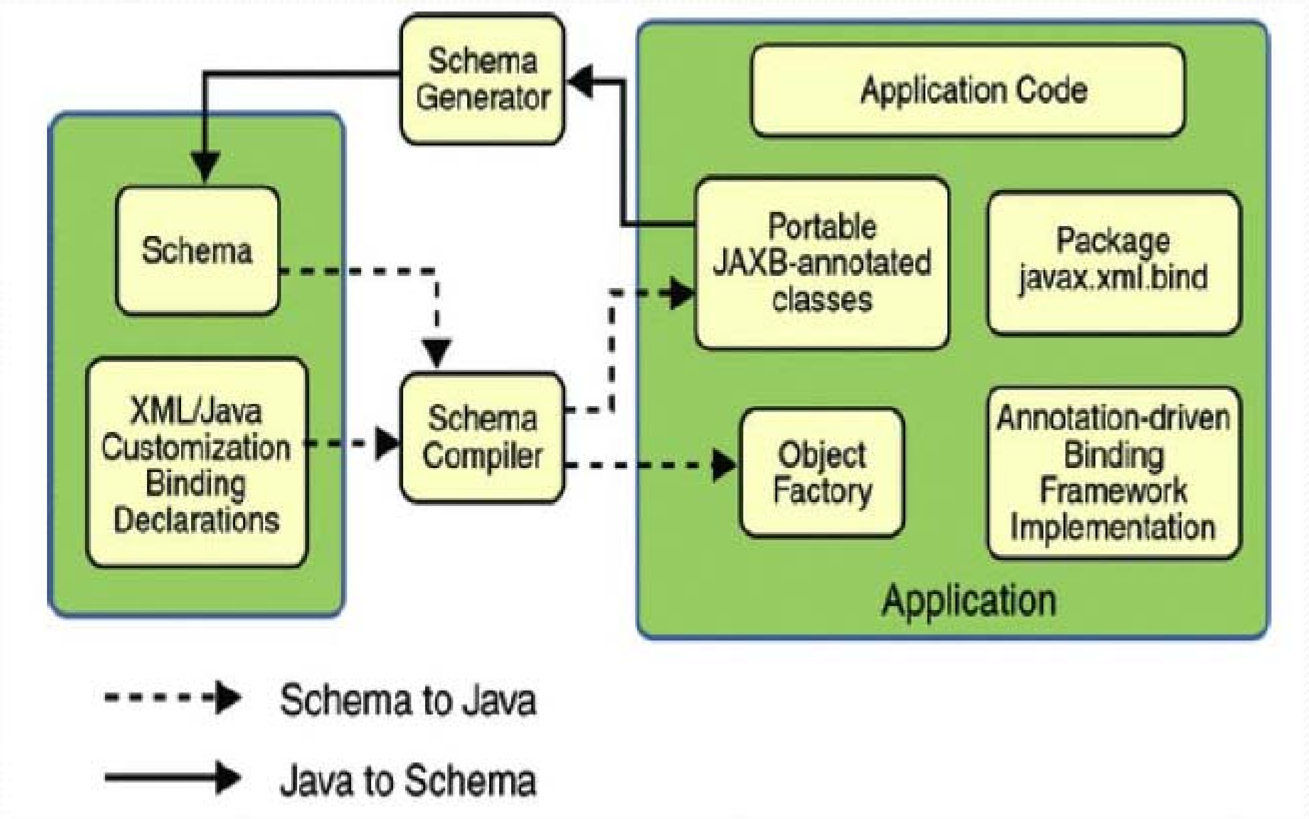
Outline

- Introduction to JAXB
- XML schema
- Java Annotation
- Example: from schema to java representations
- Example: Generate an XML document from an Object Model
- Exercise
- Dozer

Introduction to JAXB

- JAXB provides a fast and convenient way to bind between XML schemas and Java classes
- JAXB provides two main features:
  - the ability to marshal Java objects into XML
  - the ability to un-marshal XML back into Java objects
- Project site:
  - https://jaxb.dev.java.net/

JAXB Architecture



XML Schema

Node Selection

- An XML schema describes the structure of an XML document
- XML Schema is an XML-based alternative to DTD
- An XML Schema is written in XML
- XML Schema is a W3C Recommendation
  - *http://www.w3.org/2001/XMLSchema*

Node Selection

- XML Schema:
  - defines **elements** that can appear in a document
  - defines **attributes** that can appear in a document
  - defines **data types** for elements and attributes
  - defines which elements are child elements
  - defines the order of child elements
  - defines whether an element is empty or can include text
  - defines default and fixed values for elements and attributes

XSD Example

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             elementFormDefault="qualified">
  .....
</xsd:schema>
```

- xmlns:xsd="..." indicates that the elements and data types used in this schema
  - come from the *http://www.w3.org/2001/XMLSchema* namespace.
  - should be prefixed with **xsd:**

XML Schema built-in data types

- The most common built-in data types are:
  - xsd:string
  - xsd:decimal
  - xsd:integer
  - xsd:boolean
  - xsd:date
  - xsd:time
- The complete built-in data type hierarchy
  - http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

Example

- XML Elements

```
<lastname>Refsnes</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

- Corresponding element definitions

```
<xsd:element name="lastname" type="xsd:string"/>
<xsd:element name="age" type="xsd:integer"/>
<xsd:element name="dateborn" type="xsd:date"/>
```

Complex Data types

- A complex element is an XML element that contains other elements and/or attributes.
- There are four kinds of complex elements:
  - empty elements
  - elements that contain only other elements
  - elements that contain only text (and attributes)
  - elements that contain both other elements and text
- For each kind, there are many ways to write it in a XSD document. We see only one way, that is compatible with JAXB

Empty elements

- XML element

```
<product prodid="1345" />
```

- Corresponding XSD elements

```
<xsd:element name="product" type="prodtype"/>

<xsd:complexType name="prodtype">
  <xsd:attribute name="prodid" type="xsd:positiveInteger"/>
</xsd:complexType>
```

Elements that contain only other elements

- XML elements

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

- Corresponding XSD elements

```
<xsd:element name="person" type="persontype"/>

<xsd:complexType name="persontype">
  <xsd:sequence>
    <xsd:element name="firstname" type="xsd:string"/>
    <xsd:element name="lastname" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Elements that contain only text (and attributes)

- XML element

```
<shoesize country="france">35</shoesize>
```

- Corresponding XSD elements
  - Note: you must define an extension OR a restriction within the simpleContent element

```
<xsd:element name="shoesize" type="shoetype"/>

<xsd:complexType name="shoetype">
  <xsd:simpleContent>
    <xsd:extension base="xsd:integer">
      <xsd:attribute name="country" type="xsd:string" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Elements that contain both other elements and text

- XML element

```
<P>
<B>Mixed content</B> lets you embed <I>child elements</I>
</P>
```

- Corresponding XSD elements

```
<xsd:complexType name="ChunkType" mixed="true">
    <xsd:choice maxOccurs="unbounded">
      <xsd:element name="B" type="ChunkType"/>
      <xsd:element name="I" type="ChunkType"/>
    </xsd:choice>
</xsd:complexType>

<xsd:complexType name="TextType">
  <xsd:sequence>
    <xsd:element name="P" type="ChunkType" />
  </xsd:sequence>
</xsd:complexType>
```

XSD Indicators

- Order indicators are used to define the order of the elements
  - all, choice, sequence
- Occurrence indicators are used to define how often an element can occur
  - maxOccurs, minOccurs
- Group indicators are used to define related sets of elements.
  - group name, attributeGroup name

Group Name

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:element name="person" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

AttributeGroup Name

```
<xsd:attributeGroup name="elementAttrGroup">
  <xsd:attribute name="refURI" type="xsd:anyURI"
                 use="optional">
  </xsd:attribute>
  <xsd:attribute name="id" type="xsd:integer">
  </xsd:attribute>
</xsd:attributeGroup>

<xsd:complexType name="SomeEntity">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attributeGroup ref="elementAttrGroup" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Substitution

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>

<xs:complexType name="custinfo">
  <xs:sequence>
    <xs:element ref="name"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="customer" type="custinfo"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

```
<customer>
  <name>John Smith</name>
</customer>
```

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

References

- Tutorial:
  - http://www.w3schools.com/schema/default.asp
  - http://www.xfront.com/files/xml-schema.html
- XML schema validator:
  - http://tools.decisionsoft.com/schemaValidate/

Java Annotation

Annotation

- Annotations provide data about a program that is not part of the program itself. They have no direct effect on the operation of the code they annotate.
- Annotations can be applied to a program's declarations of classes, fields, methods, and other program elements.

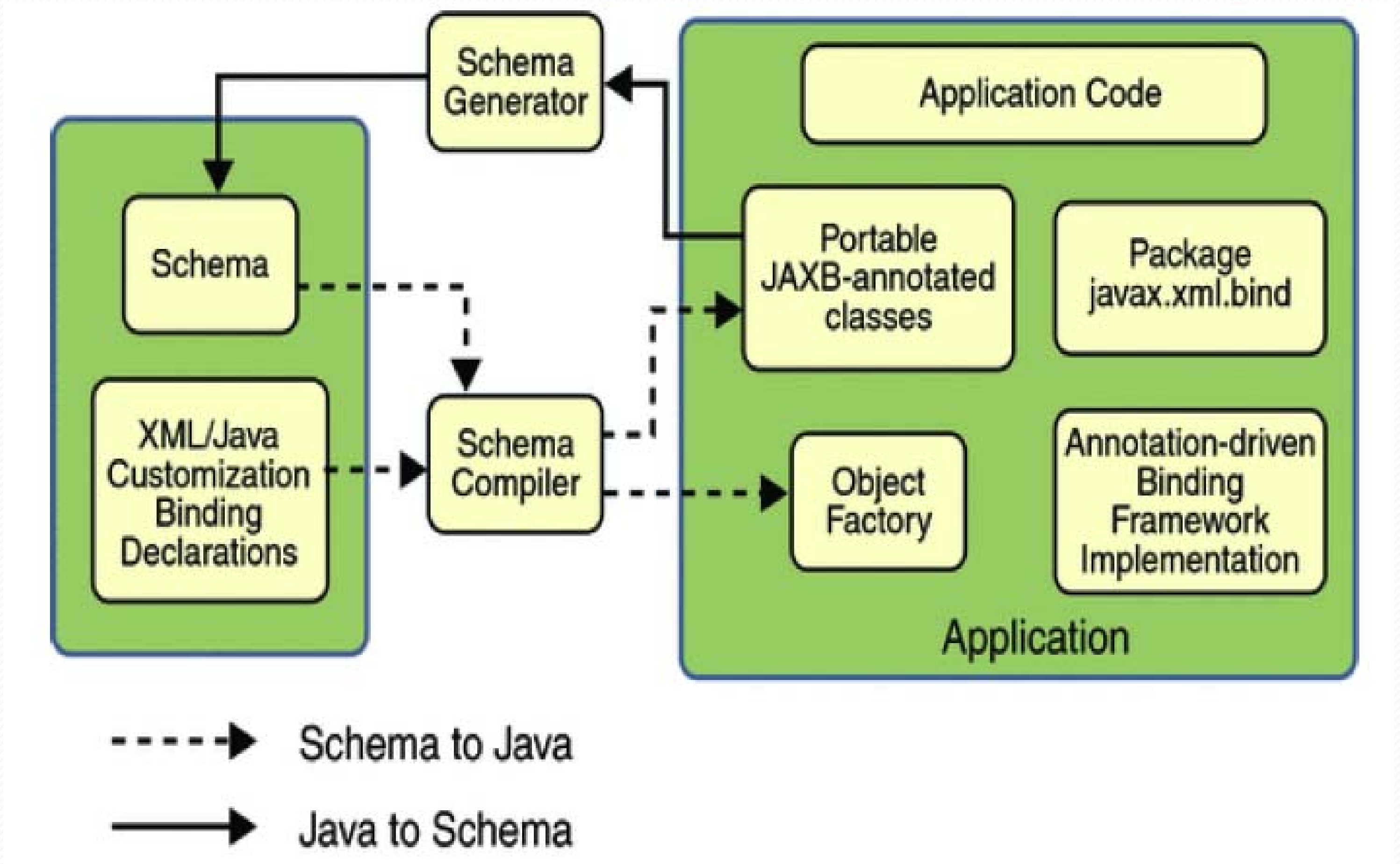
```
@Author(name = "Benjamin Franklin")
class MyClass() { }
```

Annotation

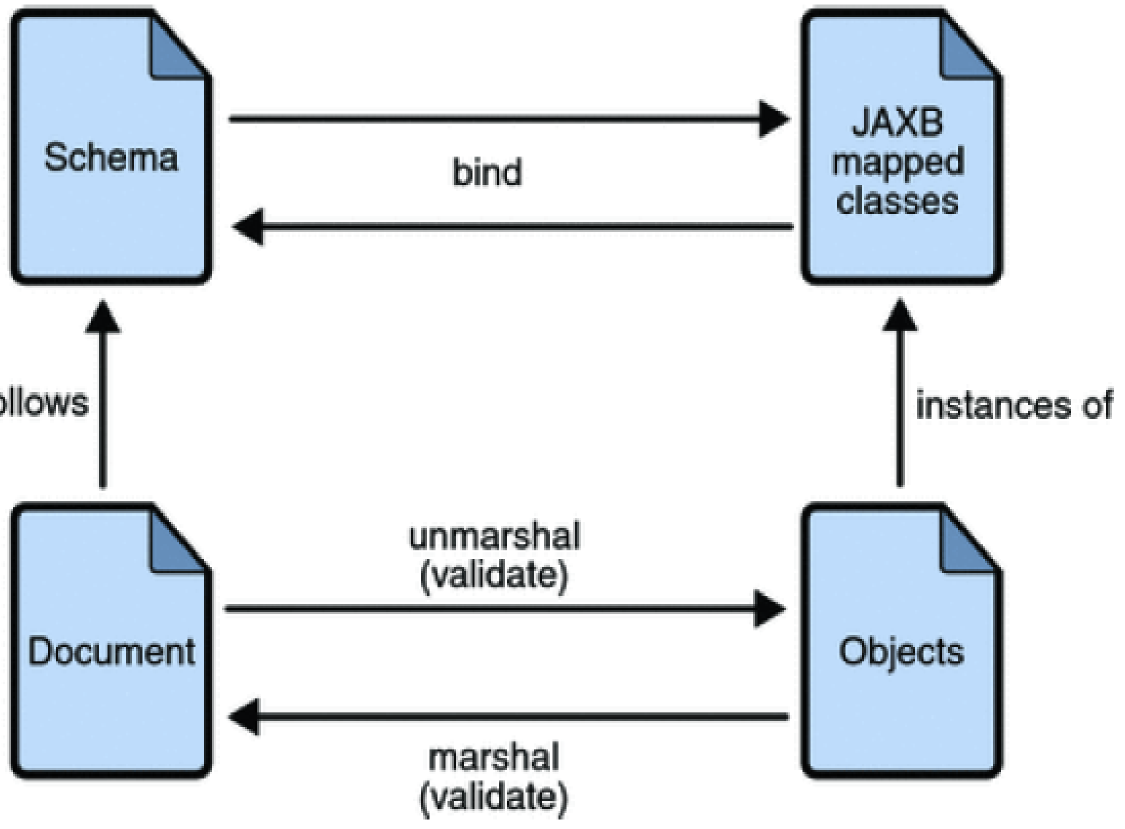
- Some uses of Annotation:
  - **Information for the compiler:** Annotations can be used by the compiler to detect errors or suppress warnings.
  - **Compiler-time and deployment-time processing:** Software tools can process annotation information to generate code, XML files, and so forth.
  - **Runtime processing:** Some annotations are available to be examined at runtime

JAXB

JAXB Architecture



JAXB Binding Process



Source: http://java.sun.com/javaee/5/docs/tutorial/doc/bnazg.html

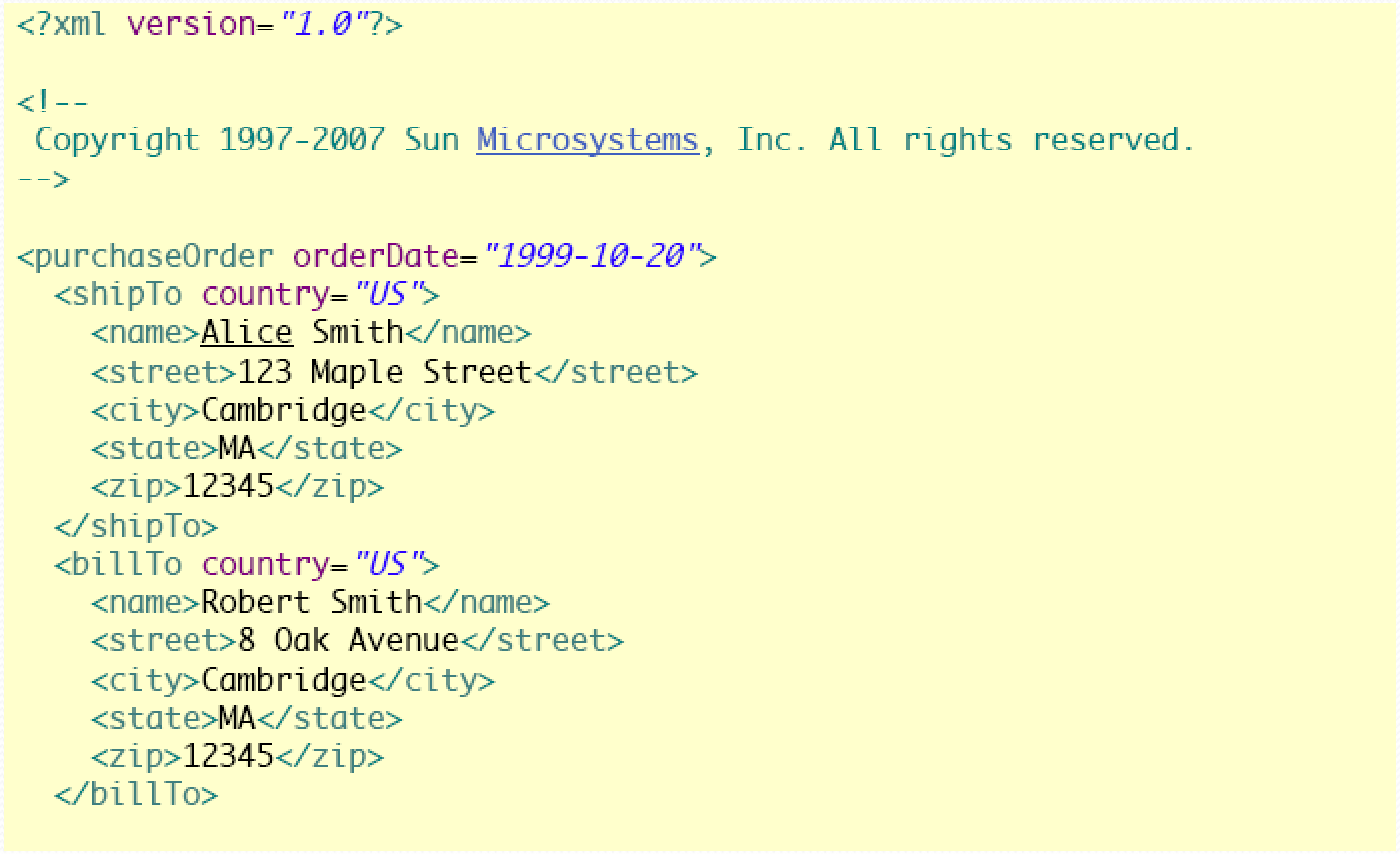
JAXB API

- JAXB version 2 is part of Java SE version 1.6
  - If you use Java SE version 1.5, download the library from https://jaxb.dev.java.net/
  - Package: javax.xml.bind

Class / Interface	Description
JAXBContext	The JAXBContext class provides the client's entry point to the JAXB API. AXBContext is an abstract class that manages the XML/Java binding.
Marshaller	The Marshaller interface serializes Java content trees back into XML data optionally validates it.
Unmarshaller	The Unmarshaller interface deserializes XML data into newly created Java content trees, optionally validating the XML data as it is unmarshalled.

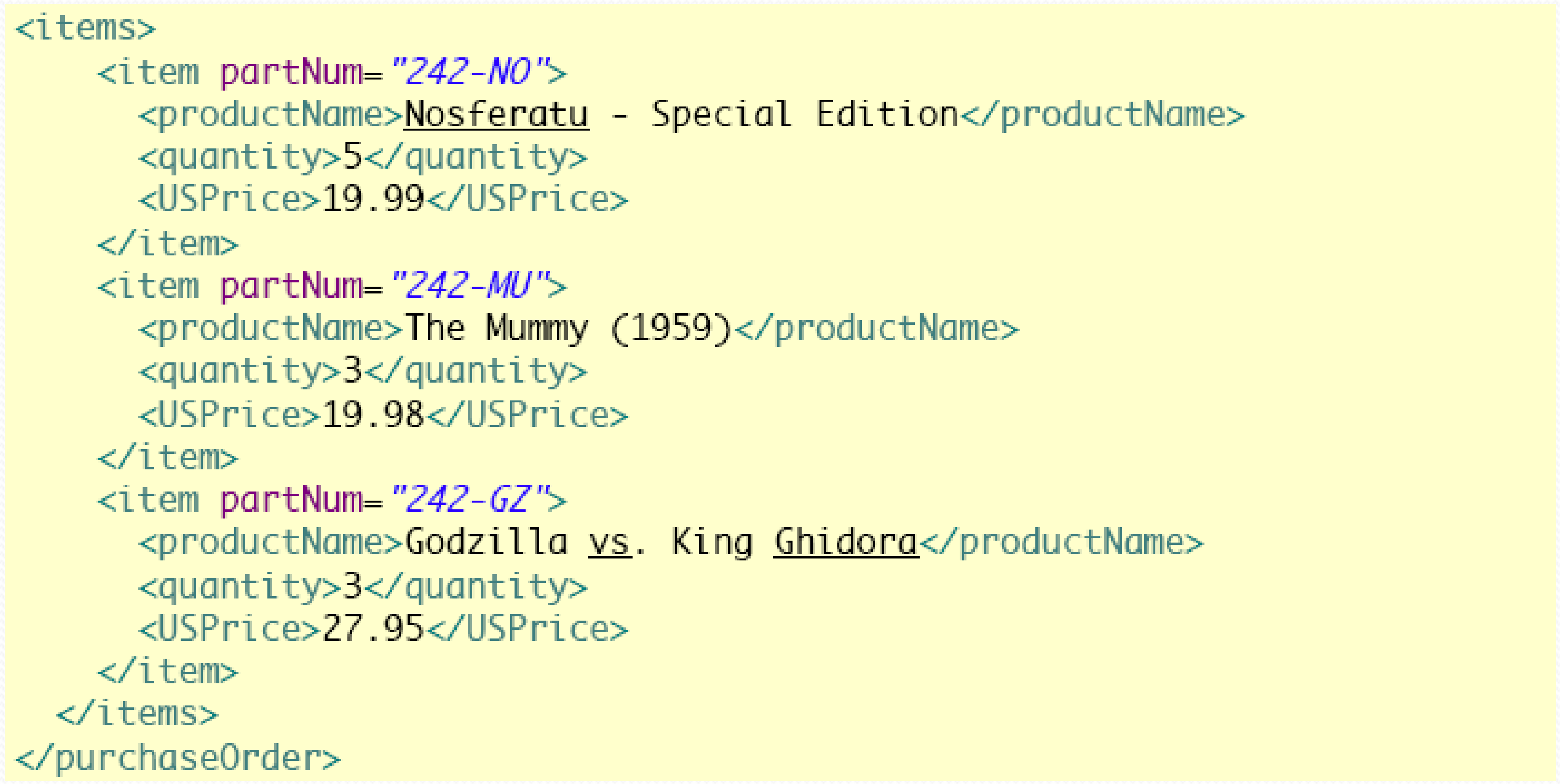
Example
From schema to java representations

XML



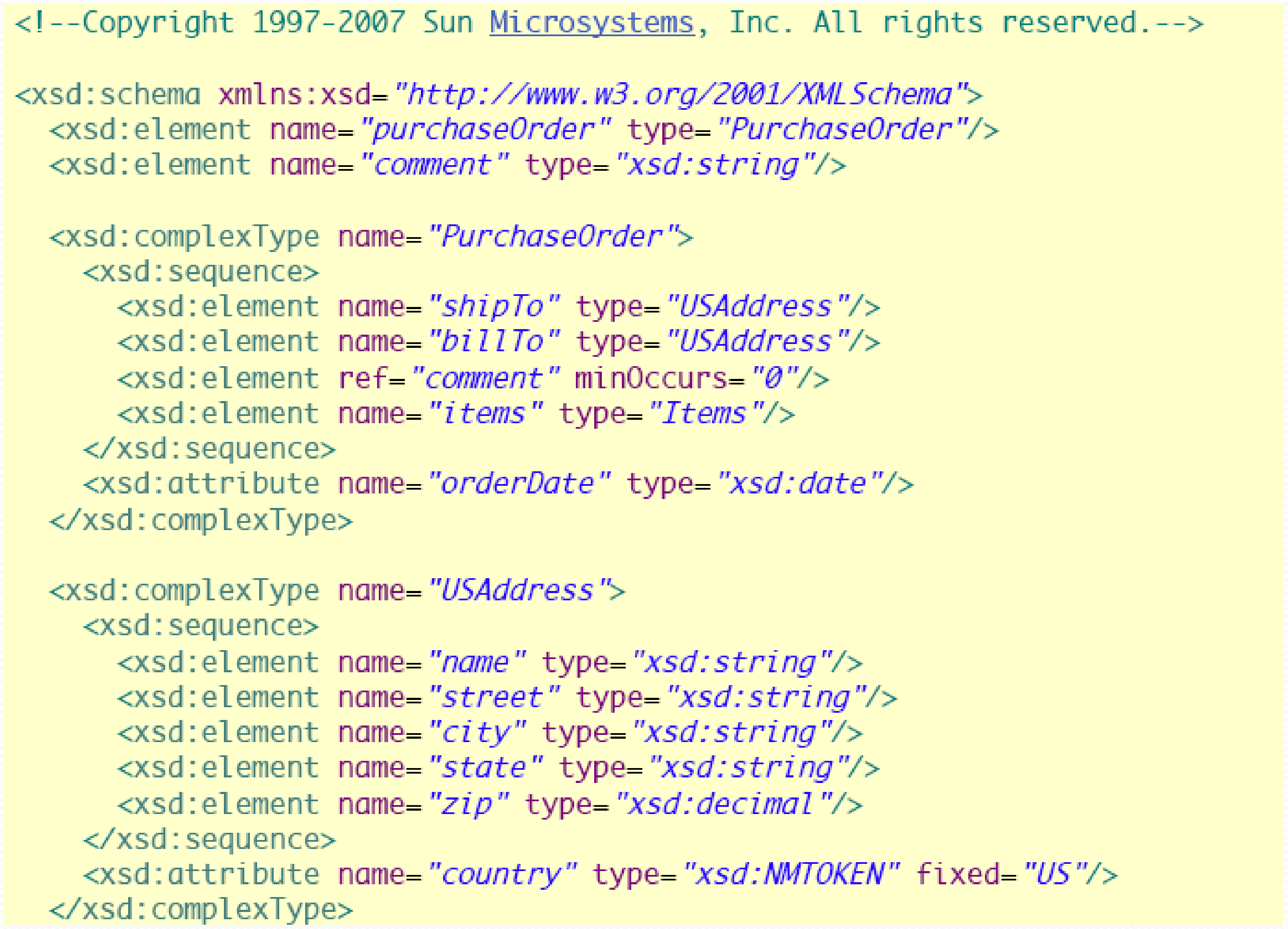
```xml
<?xml version="1.0"?>

<!--
 Copyright 1997-2007 Sun Microsystems, Inc. All rights reserved.
-->

<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Cambridge</city>
    <state>MA</state>
    <zip>12345</zip>
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Cambridge</city>
    <state>MA</state>
    <zip>12345</zip>
  </billTo>
```

XML



```xml
<items>
    <item partNum="242-NO">
      <productName>Nosferatu - Special Edition</productName>
      <quantity>5</quantity>
      <USPrice>19.99</USPrice>
    </item>
    <item partNum="242-MU">
      <productName>The Mummy (1959)</productName>
      <quantity>3</quantity>
      <USPrice>19.98</USPrice>
    </item>
    <item partNum="242-GZ">
      <productName>Godzilla vs. King Ghidora</productName>
      <quantity>3</quantity>
      <USPrice>27.95</USPrice>
    </item>
  </items>
</purchaseOrder>
```

XSD



```
<!--Copyright 1997-2007 Sun Microsystems, Inc. All rights reserved.-->

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="purchaseOrder" type="PurchaseOrder"/>
  <xsd:element name="comment" type="xsd:string"/>

  <xsd:complexType name="PurchaseOrder">
    <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items" type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
  </xsd:complexType>

  <xsd:complexType name="USAddress">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zip" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
  </xsd:complexType>
```

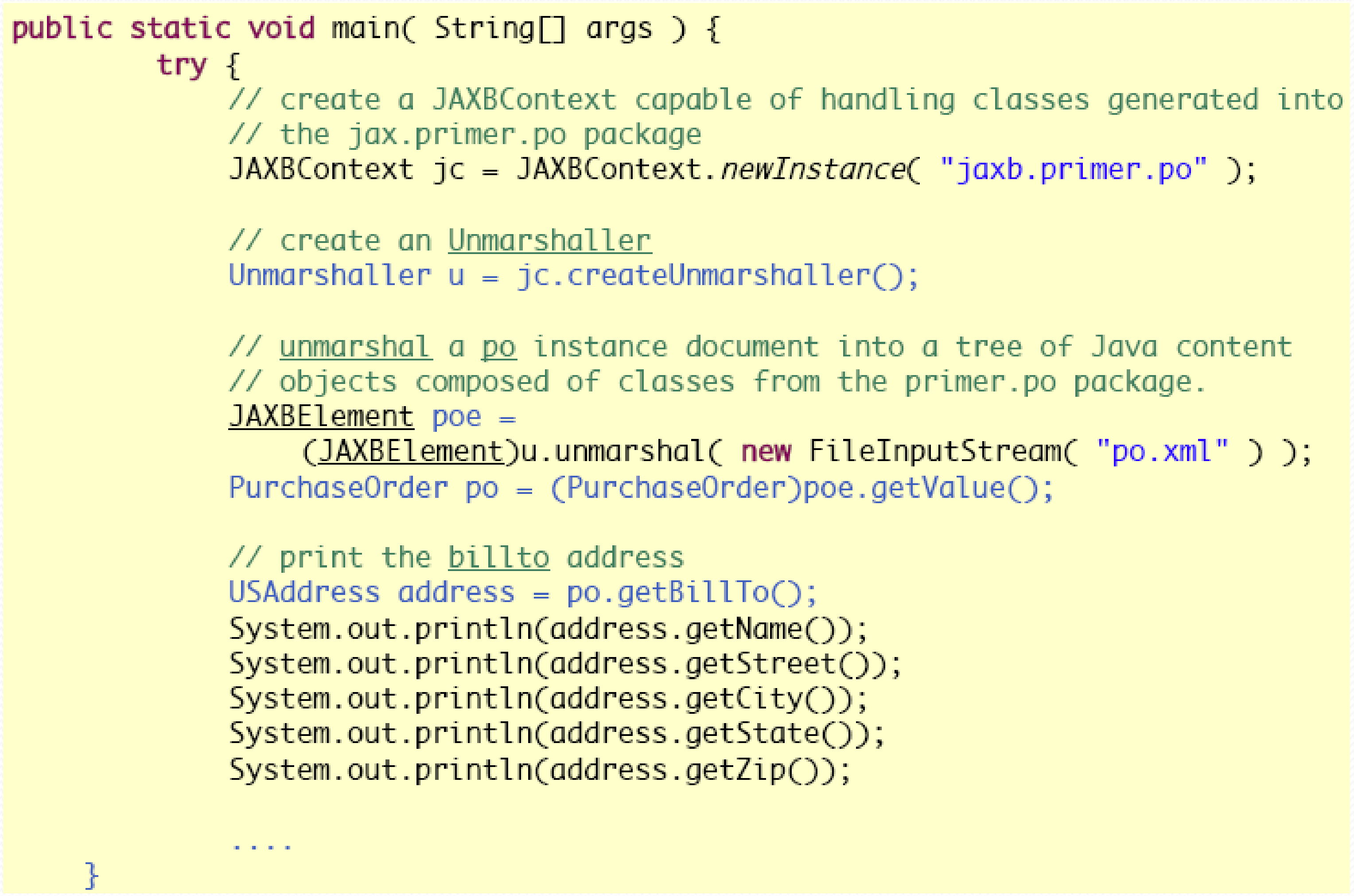
XSD

```
<xsd:complexType name="Items">
    <xsd:sequence>
      <xsd:element name="item" minOccurs="1" maxOccurs="unbounded"
type="Item"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Item">
   <xsd:sequence>
     <xsd:element name="productName" type="xsd:string"/>
     <xsd:element name="quantity">
    <xsd:simpleType>
             <xsd:restriction base="xsd:positiveInteger">
            <xsd:maxExclusive value="100"/>
             </xsd:restriction>
    </xsd:simpleType>
     </xsd:element>
     <xsd:element name="USPrice" type="xsd:decimal"/>
     <xsd:element ref="comment" minOccurs="0"/>
     <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="partNum" type="SKU" use="required"/>
  </xsd:complexType>
```

XSD

```
<!-- Stock Keeping Unit, a code for identifying products -->
  <xsd:simpleType name="SKU">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

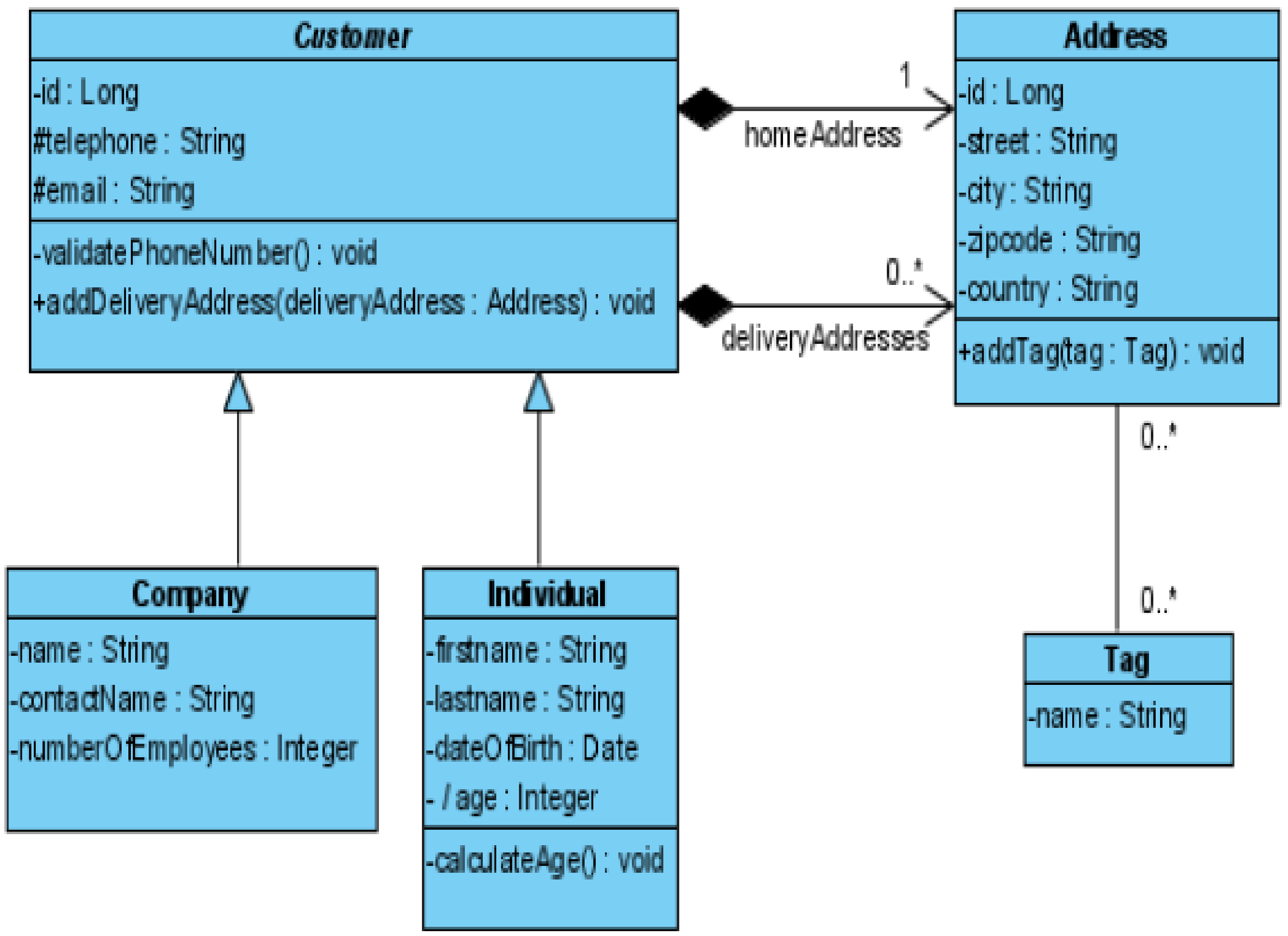
Unmarshaller



```
public static void main( String[] args ) {
        try {
            // create a JAXBContext capable of handling classes generated into
            // the jax.primer.po package
            JAXBContext jc = JAXBContext.newInstance( "jaxb.primer.po" );

            // create an Unmarshaller
            Unmarshaller u = jc.createUnmarshaller();

            // unmarshal a po instance document into a tree of Java content
            // objects composed of classes from the primer.po package.
            JAXBElement poe =
                (JAXBElement)u.unmarshal( new FileInputStream( "po.xml" ) );
            PurchaseOrder po = (PurchaseOrder)poe.getValue();

            // print the billto address
            USAddress address = po.getBillTo();
            System.out.println(address.getName());
            System.out.println(address.getStreet());
            System.out.println(address.getCity());
            System.out.println(address.getState());
            System.out.println(address.getZip());

            ....
    }
```

Example
Generate an XML document from Object Model

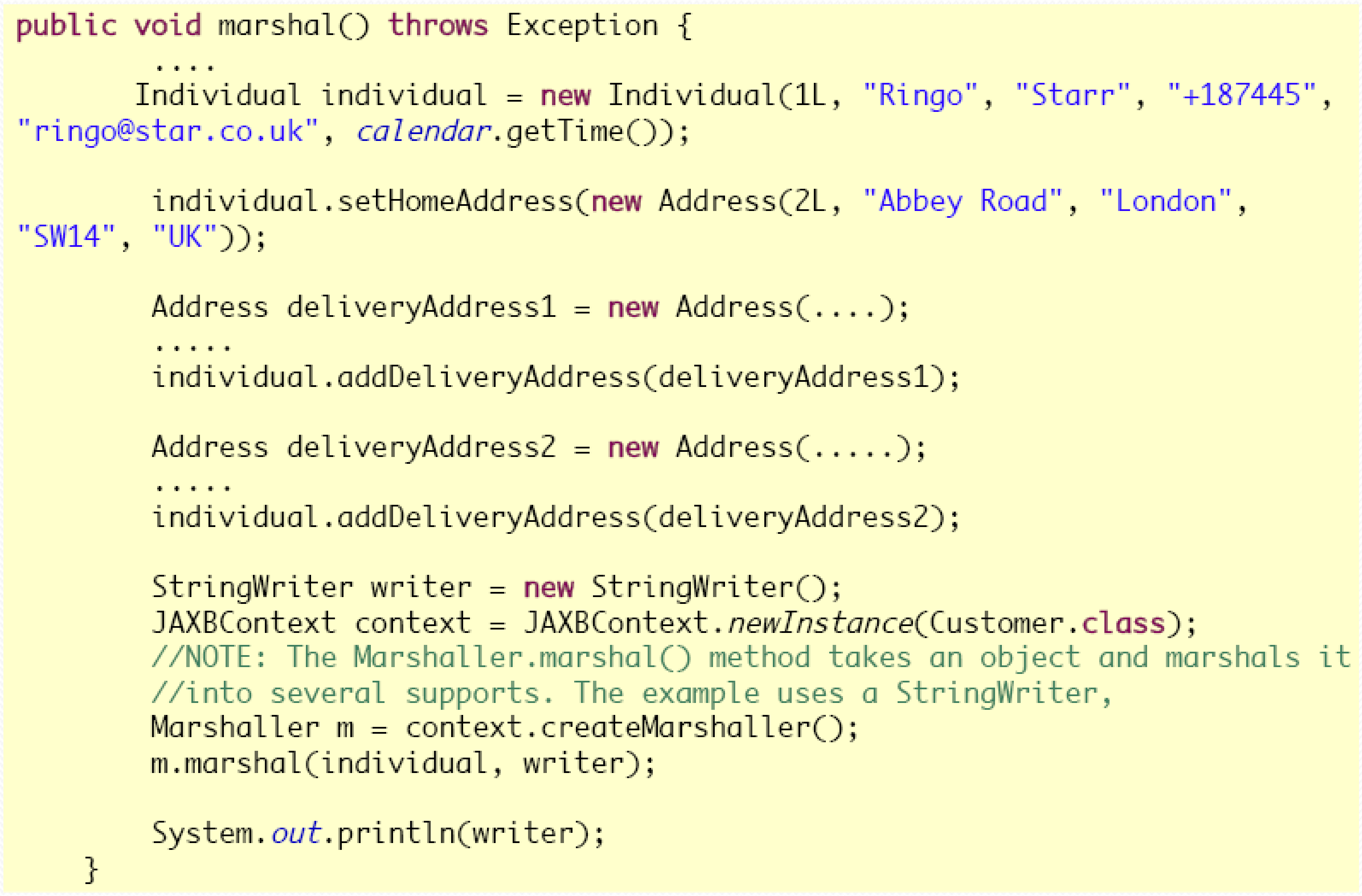
Business Model



Basic Implementation

```
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public abstract class Customer {
    protected Long id;
     protected String telephone;
     protected String email;
     private Address homeAddress;
     private List<Address> deliveryAddresses = new
                                             ArrayList<Address>();

    //getter and setter
}
```

Basic Implementation



```
public void marshal() throws Exception {
        ....
       Individual individual = new Individual(1L, "Ringo", "Starr", "+187445",
"ringo@star.co.uk", calendar.getTime());

        individual.setHomeAddress(new Address(2L, "Abbey Road", "London",
"SW14", "UK"));

        Address deliveryAddress1 = new Address(....);
        .....
        individual.addDeliveryAddress(deliveryAddress1);

        Address deliveryAddress2 = new Address(.....);
        .....
        individual.addDeliveryAddress(deliveryAddress2);

        StringWriter writer = new StringWriter();
        JAXBContext context = JAXBContext.newInstance(Customer.class);
        //NOTE: The Marshaller.marshal() method takes an object and marshals it
        //into several supports. The example uses a StringWriter,
        Marshaller m = context.createMarshaller();
        m.marshal(individual, writer);

        System.out.println(writer);
    }
```

XML Document

```
<customer>
    <deliveryAddresses>
        <city>London</city>
        <country>UK</country>
        <id>3</id>
        <street>Findsbury Avenue</street>
        <tags>
            <name>working hours</name>
        </tags>
        <tags>
            <name>mind the dog</name>
        </tags>
        <zipcode>CE451</zipcode>
    </deliveryAddresses>
    <deliveryAddresses>.....</deliveryAddresses>
    <email>ringo@star.co.uk</email>
    <homeAddress> ...</homeAddress>
    <id>1</id>
    <telephone>+187445</telephone>
</customer>
```

Customize Implementation

```
<individual id="1">
 <delivery>
  <address>
    <street>Findsbury</street>
    <zip>CE451</zip>
    <city>London</city>
    <country>UK</country>
  </address>
  <address>
    <street>Camden</street>
    <zip>NW487</zip>
    <city>Brighton</city>
    <country>UK</country>
  </address>
 </delivery>
 (...)
</individual>
```

Customize Implementation: Customer

```
@XmlTransient
public abstract class Customer {

    @XmlAttribute
    protected Long id;
    protected String telephone;
    protected String email;
    protected Address homeAddress;
    @XmlElementWrapper(name = "delivery")
    @XmlElement(name = "address")
    protected List<Address> deliveryAddresses =
                                      new ArrayList<Address>();

   //getter and setter
}
```

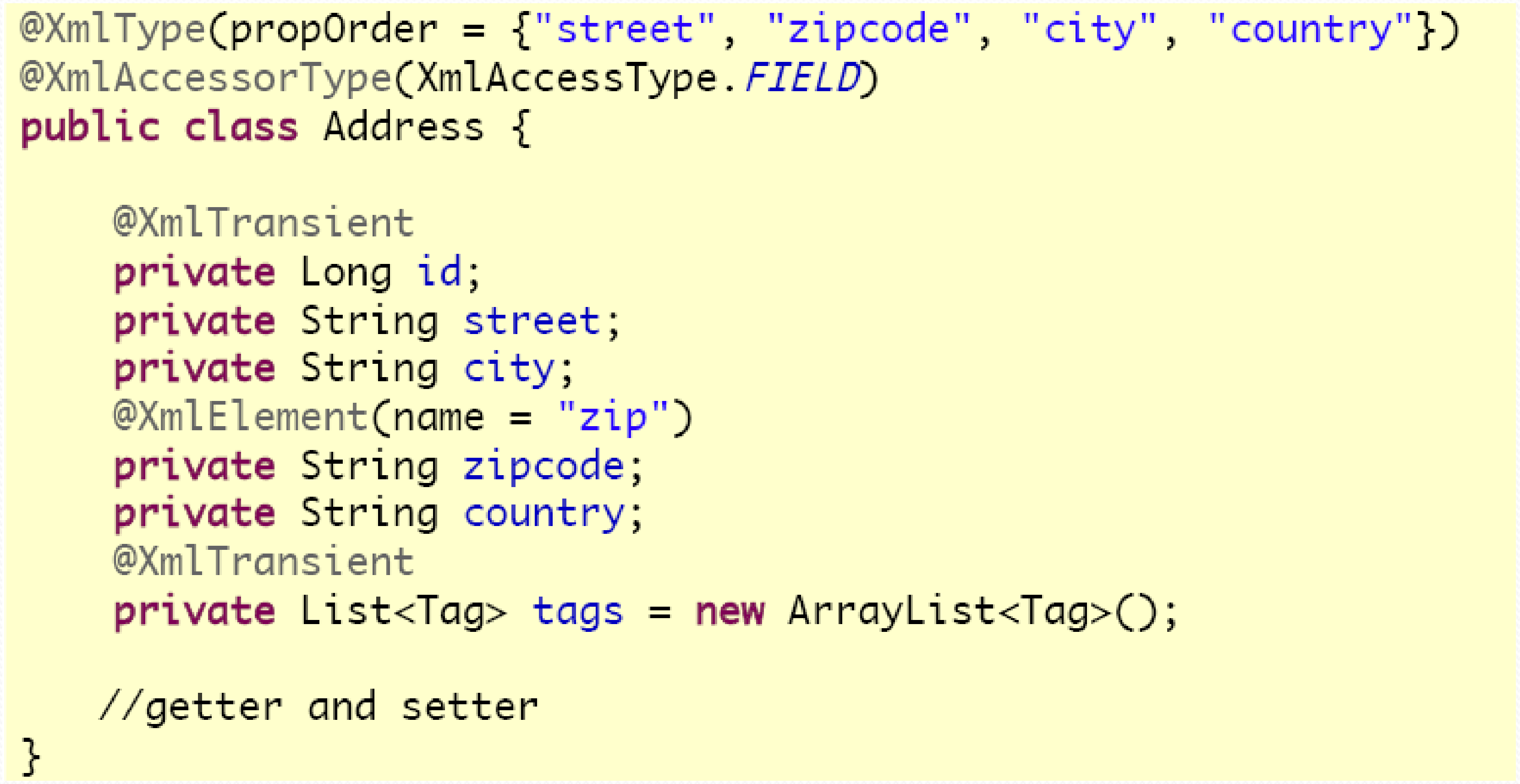
Customize Implementation: Individual

```
@XmlRootElement(name = "individual", namespace =
"http://www.watermelon.example/customer")
@XmlType(propOrder = {"id", "lastname", "firstname",
"dateOfBirth", "telephone", "email", "homeAddress",
"deliveryAddresses"})
@XmlAccessorType(XmlAccessType.FIELD)

public class Individual extends Customer {

    private String firstname;
    private String lastname;
    @XmlJavaTypeAdapter(DateAdapter.class)
     //required format: "dd/MM/yyyy"
    private Date dateOfBirth;

   //getter, setter, constructors

}
```

Customize Implementation: DateAdapter

```java
import javax.xml.bind.annotation.adapters.XmlAdapter;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

public class DateAdapter extends XmlAdapter<String, Date> {

    DateFormat df = new SimpleDateFormat("dd/MM/yyyy");

    public Date unmarshal(String date) throws Exception {
        return df.parse(date);
    }

    public String marshal(Date date) throws Exception {
        return df.format(date);
    }
}
```

Customize Implementation: Address



```java
@XmlType(propOrder = {"street", "zipcode", "city", "country"})
@XmlAccessorType(XmlAccessType.FIELD)
public class Address {

    @XmlTransient
    private Long id;
    private String street;
    private String city;
    @XmlElement(name = "zip")
    private String zipcode;
    private String country;
    @XmlTransient
    private List<Tag> tags = new ArrayList<Tag>();

   //getter and setter
}
```

Exercise

- Download code from web site.
- Run the code and do following
  - Run simple Java to XML
  - Perform Marshalling
  - Perform Un-Marshalling

- Generate java classes from JAXB using “xjc” command.
- Make java beans according to old “employee.xml” and do marshalling

Domain Objects vs Transfer Objects

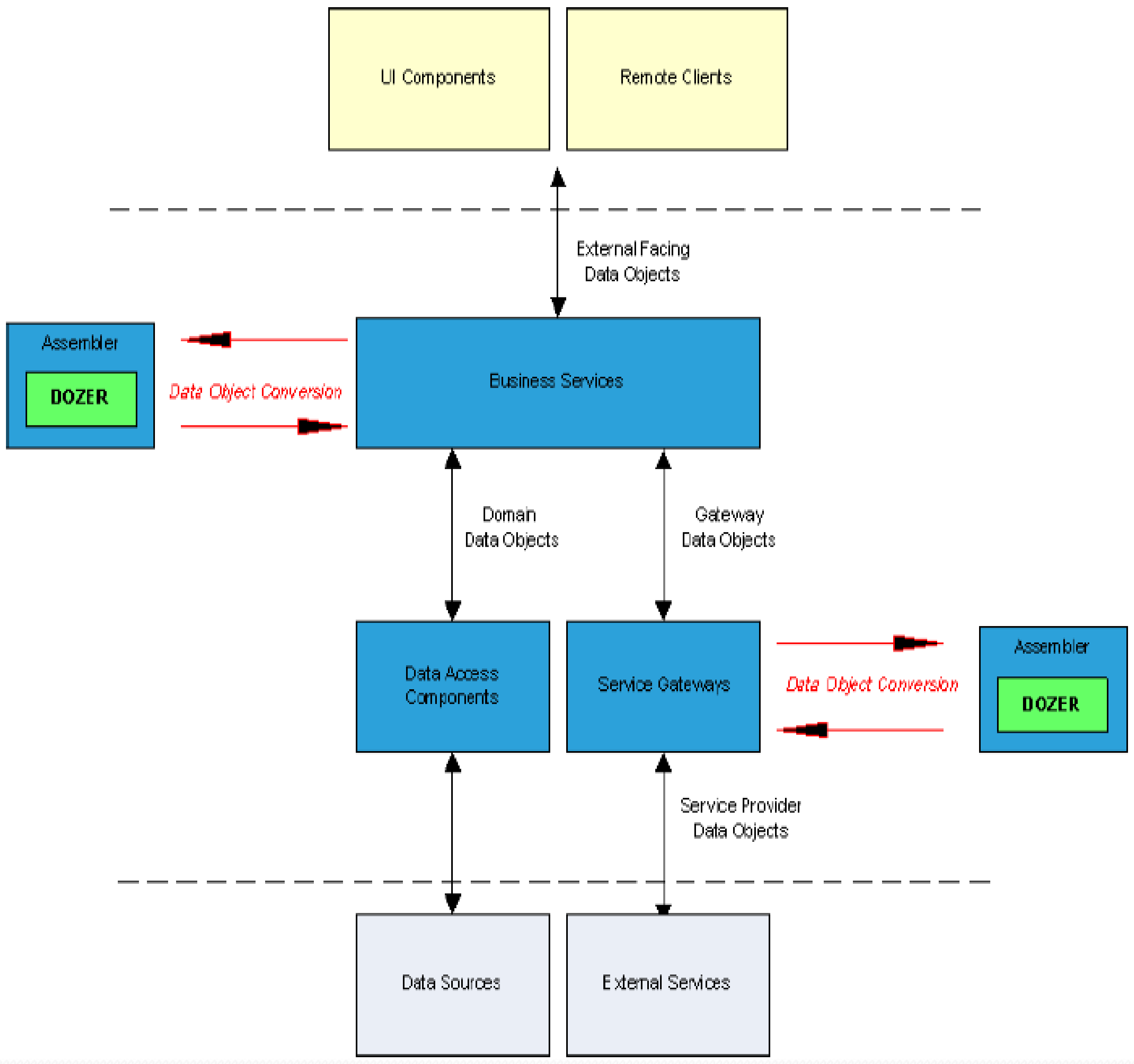
- I would like to keep domain objects separate from logics that manage the transformation into XML documents
- I propose to define two different packages:
  - entity: here we put pure domain objects
  - transfer.xml: here we put jaxb objects
- We need a mechanism to map domain objects into jaxb objects

Dozer

- Dozer is a Java Bean to Java Bean mapper that recursively copies data from one object to another
- Dozer supports mapping between attribute names and between types.
- Standard conversions are provided automatically
- You are allowed to specify custom conversions via XML

Dozer

- With Dozer, your internal domain objects are not exposed to external presentation layers or to external
- consumers.
- Dozer maps your domain objects to external APIs calls
- and vice-versa.


UI Components
Remote Clients
External Facing Data Objects
Assembler
DOZER
Data Object Conversion
Business Services
Domain Data Objects
Gateway Data Objects
Data Access Components
Service Gateways
Data Object Conversion
Assembler
DOZER
Service Provider Data Objects
Data Sources
External Services

Dozer Installation

- Download Dozer and extract the archive http://sourceforge.net/projects/dozer/files/dozer/5.1/dozer-5.1.jar/download
- Add ${dozer.home}/dist/dozer.jar to your classpath.
- Add required thirdparty runtime jars to your classpath http://dozer.sourceforge.net/dependencies.html

Example: XML

```xml
<group>
    <person id="1">
      <name>Veronica</name>
      <surname>Rizzi</surname>
      <sex>female</sex>
    </person>
     <person id="2">
      <name>Gaia</name>
      <surname>Trecarichi</surname>
      <sex>female</sex>
    </person>
</group>
```

Example: XSD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="group" type="PersonsJAXB"/>

  <xsd:complexType name="PersonsJAXB">
    <xsd:sequence>
      <xsd:element name="person" minOccurs="1"
                   maxOccurs="unbounded" type="PersonJAXB"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="PersonJAXB">
   <xsd:sequence>
     <xsd:element name="name" type="xsd:string"/>
     <xsd:element name="surname" type="xsd:string" minOccurs="0"/>
     <xsd:element name="sex" type="xsd:string"/>
   </xsd:sequence>
   <xsd:attribute name="id" type="xsd:integer" use="required"/>
  </xsd:complexType>

</xsd:schema>
```

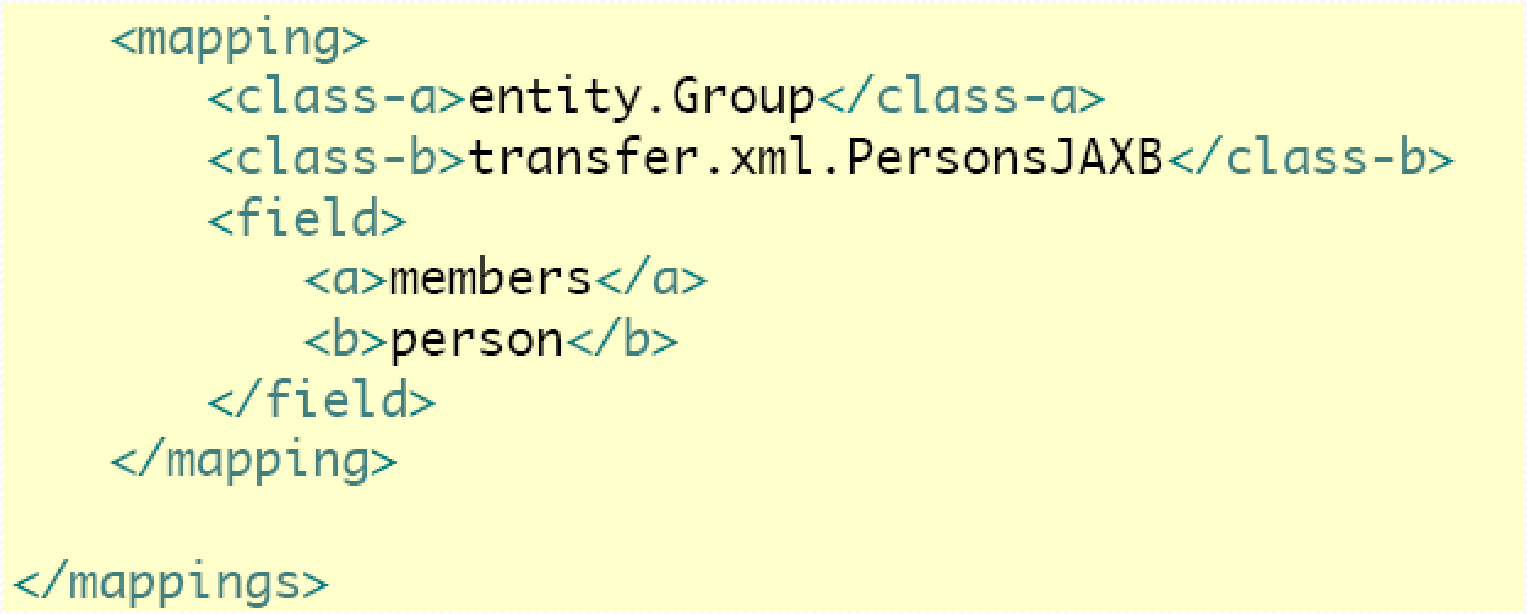
Example: Packages

- Entity
  - Group
  - Person
- transfer.xml
  - PersonsJAXB
  - PersonJAXB
  - ObjecFactory
- test
  - MappingTest

Mappings

```
<mappings xmlns="http://dozer.sourceforge.net"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">

    <mapping>
        <class-a>entity.Person</class-a>
        <class-b>transfer.xml.PersonJAXB</class-b>
        <field>
            <a>firstName</a>
            <b>name</b>
        </field>
        <field>
            <a>familyName</a>
            <b>surname</b>
        </field>
        <field>
            <a>gender</a>
            <b>sex</b>
        </field>
    </mapping>
```

Mappings



```
    <mapping>
        <class-a>entity.Group</class-a>
        <class-b>transfer.xml.PersonsJAXB</class-b>
        <field>
            <a>members</a>
            <b>person</b>
        </field>
    </mapping>

</mappings>
```

Mapper example

```
private Person mapPerson(PersonJAXB personJAXB) {

      // add all of your mappings to a list
      List myMappingFiles = new ArrayList();
      myMappingFiles.add("file:./config/dozer.xml");

      DozerBeanMapper mapper = new DozerBeanMapper();
      mapper.setMappingFiles(myMappingFiles);
      Person person = (Person) mapper.map(personJAXB,
                                          Person.class);

      return person;
   }
```

References

- http://en.wikipedia.org/wiki/JAXB
- http://java.sun.com/javaee/5/docs/tutorial/doc/bnazf.
- http://www.devx.com/Java/Article/34069/0/page/1
- http://dozer.sourceforge.net/

Assignment

- You have to model the part of scientific domain which deals with bibliography data. The model would contain the main concepts in this particular domain e.g. publication, book, patent, researcher, group of researcher, conference, journal, university, department. For each of these concepts, identify the attributes, make relation among them and finally make a domain model.

- Define an XML document which represents these concepts.
- Define the XML schema.
- Write java application which does the marshalling and un-marshalling (use JAXB).
- Use proper directory structure.
- Separate domain object from presentation objects and map them through Dozer.

Assignment: Rules

- The assignment is due on 26-Oct (Mid-Night). You will be informed about where to submit assignment. Please also check website for the updates.
- On 27-Oct I'll copy your directory and I use that copy for the evaluation.
- You have to provide a runnable code, the required XML and XSD documents and a brief report (pdf format).