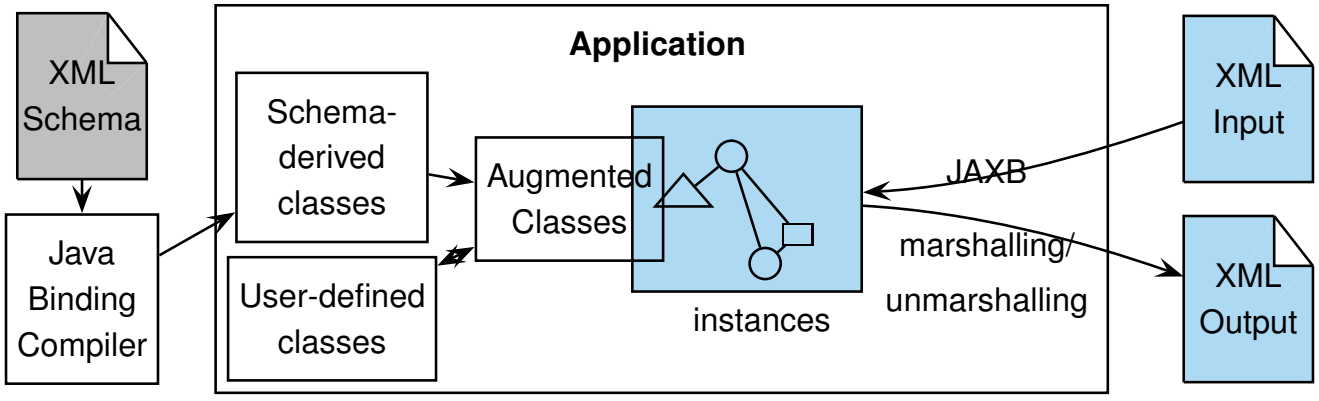
11.7 JAXB - The Java/Jakarta API for XML Binding

- API changes its position within Java from time to time
  - was published first in 2003 as part of the Java Web Services Developer Pack (which was replaced in 2006 by GlassFish)
  - From Java SE 6 until Java SE 10 (Sept. 2018), it was part of Java SE.
  - Part of Jakarta EE (formerly Java Platform, Enterprise Edition (Java EE)) current implementations e.g. `https://javaee.github.io/jaxb-v2/` (2020)
- XML elements describe objects with properties,
- correspond to classes of an application,
- derive interface with setX/getX methods (= Java Beans) as skeletons for these classes (automatically generated from an XML Schema description),
- user derives classes from these interfaces by adding behavior,
- application logics implemented by using these classes,
- import/export of XML instances of these classes via generic mappings (derived from the XSD).

JAXB Architecture

- map XML Schemas to Java classes (get/set methods),
- methods for *unmarshalling* XML data (file, DOM instance, javax XMLEvents stream, etc.) into Java objects,
- methods for *marshalling* Java objects back into XML data (DOM instance, SAX, javax XML Events stream, etc.),
- for input/output XML data formats see online documentation.

Architecture



JAXB - Example

```
<?xml version="1.0"?>
<BookCollection>
  <books>
    <book isbn="111-1234">
      <name>Learning JAXB</name>
      <price>34</price>
      <authors>
        <authorName>Jane Doe</authorName>
      </authors>
      <language>English</language>
      <language>French</language>
      <promotion>
        <Discount>10% until March 2003</Discount>
      </promotion>
      <publicationDate>2003-01-01</publicationDate>
    </book>
    <book isbn="112-0815">
      <name>Java Web Services Today and Beyond</name>
      <price>29</price>
      <authors>
        <authorName>John Brown</authorName>
        <authorName>Peter T.</authorName>
      </authors>
      <language>English</language>
      <promotion> <None/> </promotion>
      <publicationDate>2002-11-01</publicationDate>
    </book>  </books>  </BookCollection>
```

Values for xs:date and xs:time must conform to the syntax required for these XML types (cf. Slide 309)

[Filename: java/JAXB/books.xml]

JAXB - Example: XSD

[Filename: java/JAXB/books.xsd]

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:jaxb="http://java.sun.com/xml/ns/jaxb" jaxb:version="2.0">

<xs:element name="BookCollection">
 <xs:complexType>
  <xs:sequence>
   <xs:element name ="books">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="book" type="bookType"
                  minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>

  <!-- continue next page -->
```

```
<xs:complexType name="bookType">
 <xs:sequence>
  <xs:element name="name" type="xs:string"/>
  <xs:element name="price" type="xs:long"/>
  <xs:element name="authors" >
   <xs:complexType>
    <xs:sequence>
     <xs:element name="authorName" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="language" type="xs:string" minOccurs="1"  maxOccurs="unbounded"/>
  <xs:element name="promotion">
   <xs:complexType>
    <xs:choice>
     <xs:element name="Discount" type="xs:string" />
     <xs:element name="None" type="xs:string"/>
    </xs:choice>
   </xs:complexType>
  </xs:element>
  <xs:element name="publicationDate" type="xs:date"/>
 </xs:sequence>
  <xs:attribute name="isbn" type="xs:string" />
</xs:complexType>
</xs:schema>
```


JAXB HowTo

- README file in java/JAXB/JAXB-README.txt:

```
mkdir gensrc
xjc -p JAXBbooks books.xsd -d gensrc
# created classes can then be found in gensrc/JAXBbooks:
#  BookType.java, BookCollection.java, and ObjectFactory.java
# compile: generated classes can then be found in ./JAXBbooks
javac -d . $(find gensrc -name '*.java')
javac JAXBbooks.java
java JAXBbooks books.xml

xjc -p JAXBmondial mondial-jaxb.xsd -d gensrc
javac -d . $(find gensrc -name '*.java')
javac JAXBmondial/JAXBmondial.java
java JAXBmondial/JAXBmondial mondial-jaxb.xml
javac JAXBmondial/MyCountry.java
javac JAXBmondial/MondialObjectFactory.java
javac JAXBmondial/JAXBmondialExt.java
java JAXBmondial/JAXBmondialExt mondial-jaxb.xml
```

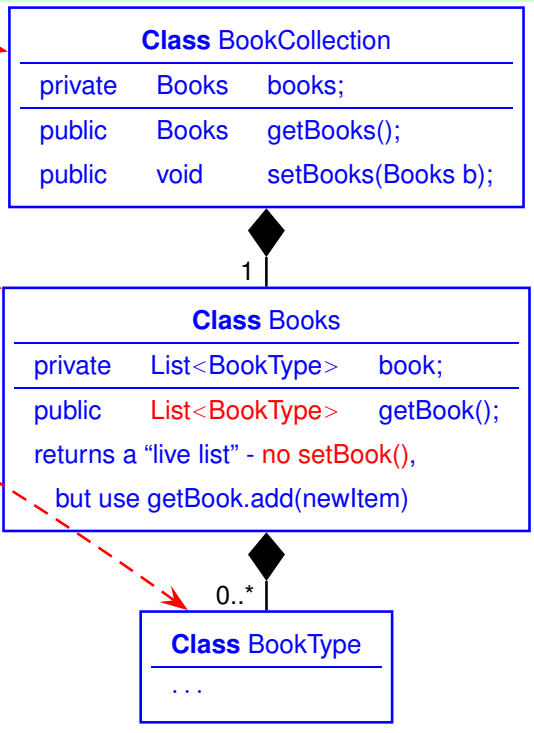
JAXB - STEPS

1. `xjc -p` *pckgname xsdfile* `-d gensrc`:
   - generates class files (with local classes) from the XML Schema specification (put them into `gensrc/`*pckgname*),
   - all classes *classname* have only the standard constructor *classname()* and getXXX()/setXXX(...) methods.
   - generates an *ObjectFactory.java* with methods of the form
     `public` *classname* `create`*classname*`() {`
     `return new` *classname*`(); }`
2. `javac -d . $(find gensrc -name '*.java')`
   compiles all files found in the `gensrc` subtree
   (generates package directory *pckgname* in ".").
3. write a a java application which
   (a) creates an Unmarshaller (which uses the ObjectFactory),
   (b) unmarshalls an XML file into objects (using the ObjectFactory),
   (c) optionally uses a Marshaller to transform the object graph back to XML.

JAXB: Binding XML Schema to Java Classes (1)

```
<xs:element name="BookCollection">
  <xs:complexType>
    <xs:sequence>
      <xs:element name ="books">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="book" type="bookType"
                minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

minOccurs = maxOccurs = 1 by default



- elements that have complexTypes are mapped to classes (for local type declarations: local classes like BookCollection.Books),
- Note: class names are automatically capitalized
- elements of simpleTypes and attributes are mapped to instance properties,
- multivalued properties are handled by lists; updates not via setXXX(), but via list modifications.

JAXB: Binding XML Schema to Java Classes (2)

```
<xs:complexType name="bookType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="price" type="xs:string"/>
    <xs:element name="authors" >
      <xs:complexType>
        <xs:sequence>
          <xs:element name="authorName" type="xs:string"
            minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="language" type="xs:string"
    minOccurs="1" maxOccurs="unbounded"/>
    <xs:element name="promotion">
      <xs:complexType>
        <xs:choice>
          <xs:element name="Discount" type="xs:string" />
          <xs:element name="None" type="xs:string"/>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="publicationDate" type="xs:date"/>
  </xs:sequence>
  <xs:attribute name="isbn" type="xs:string"/>
</xs:complexType>
```

Class BookType

```
private String          name;
private String          isbn;
private long            price;
private XMLGregorianCalendar publicationDate;
private Authors         authors;
private List            language
private Promotion       promotion;
public  String          getName();    void setName(String s);
public  String          getIsbn();    void setIsbn(String s);
public  long            getPrice();   void setPrice(long x);
public  XML-G-C.        getPublicationDate();   void set(...)
public  Authors         getAuthors();   void setAuthors(Authors as);
public  List            getLanguage();   no setLanguage()
public  Promotion       getPromotion();   void setPromotion();
```

1 — 1

Class Authors

```
private List<String> authorName;
public  List<String> getAuthorName();
live list - no setAuthorName()
```

Class Promotion

```
private String discount;
private String none;
public get/set methods
```


JAXB - Example Usage

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import javax.xml.bind.Marshaller;
import java.io.File;
import javax.xml.datatype.XMLGregorianCalendar;
import org.w3c.dom.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import JAXBbooks.*; // import classes generated by binding compiler

public class JAXBbooks {

  public static void main (String args[]) { try
    { // generate the context using the JAXBBooks directory
      // where the generated classes and the ObjectFactory are
      JAXBContext jc = JAXBContext.newInstance(ObjectFactory.class);
      Unmarshaller unmarshaller = jc.createUnmarshaller();

      //   continue next page
```

[Filename: java/JAXB/JAXBbooks.java]


```
      BookCollection collection =
```

```
    //   continue next page


    BookCollection collection =
         (BookCollection) unmarshaller.unmarshal(new File( "books.xml"));
    BookCollection.Books books = collection.getBooks();

    for (BookType book : books.getBook()) {
      System.out.println("Book details "   );
      System.out.println("Book Name: " +  book.getName().trim());
      System.out.println("Book ISBN: " +  book.getIsbn().trim());
      System.out.println("Book Price: " +  book.getPrice());
      if (book.getPromotion().getDiscount() != null)
      System.out.println("Book promotion: " +  book.getPromotion().getDiscount().trim());
      System.out.println("No of Authors " + book.getAuthors().getAuthorName().size());

      BookType.Authors authors = book.getAuthors();
      for (String authorName : authors.getAuthorName())
          System.out.println("Author Name " + authorName.trim());
      XMLGregorianCalendar date = book.getPublicationDate();
      System.out.println("Date " + date);
      for (String language: book.getLanguage())
          System.out.println("Language " + language.trim());
      // add an element to a live list:
      book.getLanguage().add("Kisuaheli");
    }
```



```
      for (String language: book.getLanguage())
          System.out.println("Language " + language.trim());
      // add an element to a live list:
      book.getLanguage().add("Kisuaheli");
    }

    // transform the result into a DOM and write to an XML file:
    Marshaller m = jc.createMarshaller();
    DOMResult domResult = new DOMResult();
    m.marshal(collection, domResult);
    Document doc = (Document) domResult.getNode();
    // transformer stuff is only for writing DOM tree to file/stdout
    TransformerFactory factory = TransformerFactory.newInstance();
    Source docSource = new DOMSource(doc);
    StreamResult result = new StreamResult("foo.xml");
    System.out.println("look into foo.xml for the result");
    Transformer transformer = factory.newTransformer();
    transformer.transform(docSource, result);
  } catch (Exception e) { e.printStackTrace(); }
}}
```

Marshaller

- generate an instance m of javax.xml.bind.Marshaller,
- The call of m.marshal(collection, destination) supports multiple kinds of destination (see javadoc for javax.xml.bind.Marshaller):
  (File)OutputStream (including System.out), SAX ContentHandler, DOM(Result) (as above), XMLStreamWriter, XMLEventWriter etc.
- Some methods and properties for controlling output:
  - `m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true)`
    `m.setProperty("com.sun.xml.bind.indentString", "  ")`
    formatted output with indentation __ (default: ____),
  - `m.setProperty("com.sun.xml.bind.xmlHeaders", "\n<!DOCTYPE mondial SYSTEM 'mondial.dtd'>")`
    outputs doctype declaration,
  - `m.setProperty(Marshaller.JAXB_FRAGMENT, true)`
    if result should be output as a *fragment* into something bigger (depends on destination, does not output xml declaration or START/END_DOCUMENT).

JAXB - Example: Mondial XSD with an annotation

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:jaxb="http://java.sun.com/xml/ns/jaxb" jaxb:version="2.0">
 <xs:element name="mondial">
  <xs:complexType> <xs:sequence>
        <xs:element name="country" type="country" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence> </xs:complexType>
 </xs:element>
 <xs:complexType name="country">
  <xs:sequence>
   <xs:element name="population" type="populationtype" minOccurs="0" maxOccurs="1" />
   <xs:element name="province" type="province" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="area" type="xs:decimal" use="optional"/>
  <xs:attribute name="car_code" type="xs:ID" use="optional"/>
  <xs:attribute name="indep_date" type="xs:date" use="optional"/>
  <xs:attribute name="capital" type="xs:IDREF" use="optional">
   <xs:annotation>  <!-- annotation of the target type  <<<<<<<< -->
    <xs:appinfo>
     <jaxb:property>
      <jaxb:baseType name="City"/>  <!-- Note: capitalization - otherwise: error !!!!!!!!!! -->
     </jaxb:property>
    </xs:appinfo>
   </xs:annotation>
  </xs:attribute>
 </xs:complexType>   <!-- continue next page -->
```

[Filename: java/JAXB/mondial-jaxb.xsd]

```
<xs:complexType name="province">
 <xs:sequence>
  <xs:element name="population" type="populationtype" minOccurs="0" maxOccurs="1" />
  <xs:element name="city" type="city" minOccurs="0" maxOccurs="unbounded" />
 </xs:sequence>
 <xs:attribute name="name" type="xs:string" use="required"/>
 <xs:attribute name="area" type="xs:decimal" use="optional"/>
</xs:complexType>

<xs:complexType name="city">
 <xs:sequence>
  <xs:element name="population" type="populationtype"  minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 <xs:attribute name="name" type="xs:string" use="required"/>
 <xs:attribute name="id" type="xs:ID"/>
</xs:complexType>

<xs:complexType name="populationtype">
 <xs:simpleContent>
  <xs:extension base="xs:nonNegativeInteger">
   <xs:attribute name="year" type="xs:nonNegativeInteger" use="optional"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
</xs:schema>
```


JAXB example: Mondial – Datatypes

(see `java/JAXB/gensrc/JAXBmondial/XXX.java`)

- annotation of the country/@capital IDREFS attribute:
  ⇒ Country: `public City getCapital()`,
- without annotation (like standard mondial):
  ⇒ Country: `public Object getCapital()`, requires casting.
- countries have only one, and cities may have several (complex) population subelements (of type “PopulationType” → class PopulationType.java):
  - Country: `public PopulationType getPopulation()`,
  - City: `public List<PopulationType> getPopulation()`,
- Populationtype: content is xs:nonNegativeInteger – `public BigInteger getValue()`,
- Country/@Area: value is xs:decimal – `public BigDecimal getArea()`,
- Date values:
  Class IndepDate (<indep_date from=“GB”>1776-07-04</indep_date>):
  `public XMLGregorianCalendar getValue()`,
  Class Organization (<organization ...> <established>1992-02-06</>...</>):
  `public XMLGregorianCalendar getEstablished()`.

JAXB – Datatypes

- Complex Types → application-specific auto-generated (bean) classes with setter/getter methods.
- Text content types and attribute value types:
  - "High-level" datatypes like xs:decimal, xs:nonNegativeInteger, xs:integer are mapped to *Java literal classes* java.math.BigDecimal, java.math.BigInteger, etc.
  - xs:int, xs:long (see Books.xsd: BookType.price) are mapped to *Java literal types* int, log, etc.
- Usage
  - XML + XML Schema for data exchange:
    use implementation-level types xs:int, xs:long etc. in XSD
  - XML + XML Schema as data model (+ ontology): comes with semantic datatypes like xs:nonNegativeInteger,
    ⇒ JAXB programs must do conversions.

JAXB - Mondial

```
<?xml version="1.0"?>
<mondial>
 <country name="Austria" area="83850" indep_date="1918-11-12" capital="cty-Austria-Vienna">
  <population>8023244</population>
  <province name="Vienna" area="415">
   <population>1583000</population>
    <city name="Vienna" id="cty-Austria-Vienna">
     <population year="1994">1583000</population>
     <population year="2013" measured="admin.">1761738</population>
    </city>
  </province>
  <province name="Upper Austria" area="11979">
   <population>1424910</population>
   <city name="Linz">
     <population year="1994">203000</population>
     <population year="2013">193511</population>
   </city>
   <city name="Wels">
     <population year="2013">59239</population>
   </city>
  </province>
 </country>
</mondial>
```

[Filename: java/JAXB/mondial-jaxb.xml]

JAXB - Example Usage

```
package JAXBmondial;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import java.io.File;
import java.math.BigInteger;
import java.math.BigDecimal;
import java.util.List;

public class JAXBmondial {
public static void main (String args[]) {
 try {
  JAXBContext jc = JAXBContext.newInstance("JAXBmondial");
  Unmarshaller unmarshaller = jc.createUnmarshaller();
  Mondial mondial = (Mondial) unmarshaller.unmarshal(new File("mondial-jaxb.xml"));

  List<Country> countryList  = mondial.getCountry();
  for (Country country: countryList)
  {
    System.out.println("Country: " + country.getName()  );
    BigDecimal area = country.getArea();
    BigInteger countrypop = country.getPopulation().getValue();
      // datatype class handling is ugly (pop is BigInteger, area is BigDecimal):
    System.out.println("   pop: " +  countrypop + ", area: " + area + ", density: "
              + new BigDecimal(countrypop).divide(area, java.math.BigDecimal.ROUND_HALF_UP));
```



```
    System.out.println("   pop: " +  countrypop + ", area: " + area + ", density: "
              + new BigDecimal(countrypop).divide(area, java.math.BigDecimal.ROUND_HALF_UP));


    // Java knows from the annotation of the IDREF attribute that this is a city:
    City cap = country.getCapital();
    System.out.println("   cap: " +  cap.getName());

    List<Province> provList = country.getProvince();
    for (Province prov: provList) {
      System.out.println("   Province name: " + prov.getName().trim());
      System.out.println("            area: " + prov.getArea().toString());
      System.out.println("            pop : " + prov.getPopulation().getValue());
      List<City> cityList = prov.getCity();
      for (City city : cityList) {
        System.out.println("            City name: "+city.getName().trim());
        List<Populationtype> poplist = city.getPopulation();
        for (Populationtype p : poplist)
          System.out.println("                pop "
                             + p.getYear() + ": " + p.getValue());
  } } }
 } catch (Exception e ) { e.printStackTrace(); }
}}
```

[Filename: java/JAXB/JAXBmondial/JAXBmondial.java]

JAXB Integration with Java Application?

Classes in an application program

- application-specific methods
- properties that are local to the Java existence of the object

JAXB-generated classes vs. user-defined classes

- user-defined class my_xxx where xxx is a subclass of:
  - useful from the java point of view: extend application class with bean functionality and marshalling.
  - cannot be communicated declaratively to the JAXB generation of the classes (annotation with xjc:superClass c in the XML Schema does only allow to make all classes subclasses of c)
- define a subclass: my_xxx extends xxx
  - after unmarshalling, the objects are only instances of xxx

  $\Rightarrow$ methods of my_xxx not applicable

  $\Rightarrow$ Different alternatives.

User-Defined Extension of JAXB-Created Classes

Manual editing of generated classes themselves

- edit the generated xxx.java files
- if instance attributes are added, they must also be added either to propOrder, or get an anntotation as @XmlAttribute – and then they will be exported when marshalling them.

$\Rightarrow$ must be manually redone/adapted after schema changes.

User-Defined Subclasses (I)

- (manually) write application subclasses my_xxx that extend the JAXB-generated classes,
- after unmarshalling, traverse the tree and re-create the objects as instances of the my_xxx subclasses.

User-Defined Subclasses (II) – Overwrite Generated Object Factory

- create the instances of the my_xxx subclasses during unmarshalling:
  JAXB allows to create the unmarshaller over a user-defined Object Factory.

JAXB - Example Usage with extended class definition

```
package JAXBmondial;

public class MyCountry extends Country {
  // a method for more comfortable manipulation:
  public void addProvince(Province p) {
      getProvince().add(p);
  }
  // a "useful" method:
  public void printCityNames() {
    for (Province prov : getProvince()) {
        for (City city : prov.getCity()) {
            System.out.println(city.getName().trim());
        }
    }
}}
```

[Filename: java/JAXB/JAXBmondial/MyCountry.java]

JAXB - Example Extended Object Factory

- original auto-generated ObjectFactory can be found in
  `java/JAXB/gensrc/JAXBmondial/ObjectFactory.java`:

  ```
  public Country createCountry() { return new Country(); }
  ```

```
package JAXBmondial;
import JAXBmondial.ObjectFactory;
public class MondialObjectFactory extends ObjectFactory {
  @Override
  public Country createCountry() {
    System.out.println("create MyCountry");
    return new MyCountry();
  }}
```

[Filename: java/JAXB/JAXBmondial/MondialObjectFactory.java]

JAXB - Example Usage with extended class definition

- tell the unmarshaller to use the modified MondialObjectFactory
- *unm*.setProperty(*jaxb-unm-propertyId*, *myClass*);
- supported Values for *jaxb-unm-propertyId*:
  in general none, only provider-dependent ones. Download most recent JAXB2.

JAXB - Example Usage with extended class definition

```
package JAXBmondial;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import java.io.File;

public class JAXBmondialExt {
 public static void main (String args[]) { try {
  JAXBContext jc = JAXBContext.newInstance("JAXBmondial");
  Unmarshaller unm = jc.createUnmarshaller();
  unm.setProperty("com.sun.xml.bind.ObjectFactory", new MondialObjectFactory()); // <<<<
  Mondial mondial = (Mondial) unm.unmarshal(new File("mondial-jaxb.xml"));

  for ( Country x : mondial.getCountry() ) {  // <<<<<<<  only "Country" possible
    MyCountry country  = (MyCountry) x;   // <<<<<<<<<
    System.out.println("Country: " + country.getName()  );
    country.printCityNames();  // <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
  }} catch (Exception e ) { e.printStackTrace(); }
}}
```

[Filename: java/JAXB/JAXBmondial/JAXBmondialExt.java]

JAXB MAPPING: SUMMARY

- allows for easy and lightweight unmarshalling, bean-based manipulation and marshalling of XML data,
- higher level of abstraction from XML representation, compared with DOM and SAX,
- but still actually just a way to manipulate XML data without having to know the specific notions of the XML data model.

Minor Comments

- naming (getBook() for a list etc.) not always intuitive;
  can be customized by annotations to the XSD;
- intermediate elements (example: Books, Authors) lead to unnecessary classes;
  can often be omitted (example: Book/Language elements)

$\Rightarrow$ to get a better "modeling", do not use structures like
Country-hasProvince-Province-hasCity-City
(as in Striped RDF/XML [Semantic Web lecture]; this generates intermediate classes), but
Country-Province-City.

- Care for datatypes/classes (cf. Slide 561).

ASIDE: SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

- Generic “protocol” (nevertheless, HTTP-based)
- Any object can be serialized in XML, sent, and deserialized
  (only having the Java class code, without having an XSD).
  So far similar to the OIF (Object Interchange Format) of ODMG (cf. Slide 53 ff.).
- The XML representation is not intended to be processed on the XML level, but only by soap-unpacking it.
- Bad experience: correct packing/unpacking only between same SOAP implementations.
- Note: Instances of Java XML (DOM) are not serialized as plain XML, but as SOAP serialization of an instance of the underlying DOM implementation class.
  (not intended *for* exchanging XML, but for exchanging objects *by* XML).

⇒ when messages are designed to be XML, SOAP is not the right way, but use simple, plain HTTP!

- One does not need to have any knowledge of XML to use soap (actually, knowledge of XML doesn’t help).

⇒ so it does not fit in this course.

11.8 XML Digester

Comparison

- SAX/StAX:
  - fine granularity,
  - extremely flexible,
  - hard to write and read
- JAXB:
  - whole document is transformed into objects
  - unflexible
  - self-explaining mapping

... in-between:

- http://commons.apache.org/digester/

XML DIGESTER: PRINCIPLE

- rule-based: on simple XPath patterns ... do something.
- internally based on SAX,
- rules hook on beginElement, PCDATA, and endElement,
- provides a stack with automatical and user-defined behavior,
- building an object graph/tree by traversing the XML tree:
  - predefined rules for user-defined selective XML-to-objects mapping supported by a stack and bean-style user-defined classes.
- implementing other XML-tree-traversal-based algorithms:
  - rules with user-defined program code on the predefined hooks
- mixed functionality:
  - can be used for SAX/StAX-style filtering from the stream (and building intermediate objects) and query answering.
- Comparison with JAXB:
  transformation XML→objects, but not the other direction.

XML Digester: Stack

Predefined rule types mirror XML element tree/hierarchy:

- top-of-stack-element is always the one that is currently accessed by methods,
- ancestors are down the stack.
- push(), pop(),
- peek(n) accesses the n-th element on the stack (top=0),
- generation of objects on the stack is controlled by rules:
- ObjectCreate(*pattern*, *class*.class)
  - if an element satisfying *pattern* is started, create a new instance of class *class* and push it on the stack.
  - endElement(): pops the topmost element from the stack.

⇒ On-the-fly-filtering: specify addObjectCreate() only for relevant element types (=object classes).

- During traversal, map the element hierarchy to the created objects:
  SetNext(*pattern*, *method*): on endElement() of x satisfying *pattern*, calls *method* of the next object on the stack, *method*'s argument type must be the class of x.
  (i.e. applies peek(1).method(peek(0)))

Built-In Rules for XML-to-Objects/Beans Mapping

Rule specifications consist of a match pattern (similar to XSL patterns) and specifications of the action:

- Patterns: only *elname/.../elname* and **/elname/.../elname* where * stand for an arbitrary number of child navigation steps,
- digester.addObjectCreate(*pattern*, *class*); (see above)
- digester.addSetProperties(*pattern*, *attrname*, *property*);
  sets *property* of the top object to the value of *attrname*;
  also [...]- lists of attrnames/properties are allowed.
- digester.addBeanPropertySetter(*pattern*);
  given a node x matching the pattern, sets the property with x's name to the value of x.
- digester.addCallMethod(*pattern*, *method*, n);
  digester.addCallParam(*pattern*, i); ($i \leq n$)
  executes a *method* call to the top object with n parameters, which are set by the value(s) of the subsequent addCallParam rules.
- digester.addSetNext(*pattern*, *method*);
- see Javadoc at `http://commons.apache.org/digester/` for details.

XML Digester: Example

```
import java.io.File;
import java.util.TreeSet;
import org.apache.commons.digester3.Digester;

public class GetMillionCities {
  public static class CityCollection extends TreeSet<City>  {
      public void addCity(City c) { if (c.population > 1000000) this.add(c);}  }
  public static void listCities(CityCollection cities) {
      for (City c : cities) {
         System.out.println(c.name + " " + c.country + " " + c.population); }  }
  public static class City implements Comparable{
    String name = null;
    String country = null;
    int population = -1;
    public void setName(String n) { if (name == null) name = n; }
    public void setCountry(String code) { this.country = code; }
    // note: all PCDATA/CDATA values are strings!
    public void setPopulation(String pop) { this.population = new Integer(pop); }
    public int compareTo(Object o) {
         if (this.population < ((City)o).population) return 1; else return -1; }
  }
  // continue next page
```

[Filename: java/Digester/GetMillionCities.java]

```
  public static void main(String[] args) {
    File mondial = new File("mondial.xml");
    final Digester digester = new Digester();
    digester.push(new CityCollection());

    // note: reacts only on cities as direct children of countries
    // */city would take all cities,
    // country//city, country/*/city, mondial/*/city are not allowed;
    digester.addObjectCreate("mondial/country/city", City.class);
    digester.addSetProperties("mondial/country/city", "country", "country");
    digester.addBeanPropertySetter("mondial/country/city/name");
    digester.addCallMethod("mondial/country/city/population", "setPopulation", 1);
    digester.addCallParam("mondial/country/city/population", 0);
    // Digester (clearly) does not like population[last()]. //
    // note: at the end, the last=most recent population is the stored value
    // at </city> calls addCity() of the now-top-of-stack-object which is the CityCollection
    digester.addSetNext("mondial/country/city", "addCity");
    try { digester.setValidating(false);
          final CityCollection cities = digester.parse(mondial);
          System.out.println("####  now listing cities #### ");
          listCities(cities);
        } catch (Exception e) { e.printStackTrace(); }
} }
```


Digester: Rules

The above are shortcuts for typical rule patterns.
Every rule implementation class implements three methods:

- begin(): executed at startElement(),
- end(): executed at endElement(),
- body(): executed at PCDATA contents of an element.

Example: addObjectCreate(*pattern*, *class*)

- Class ObjectCreateRule(*class*.class)
- begin(): create object of given class, initialize from attributes in the opening tag (if required), push it on the stack;
- end(): pop object from stack.

Example: addSetNext(*pattern*, *method*)

- Class SetNextRule(*class*.class), subclass of AbstractMethodRule
- begin(): nothing,
- end(): pop object obj from stack and execute *method* of then-top with argument obj.

XML Digester: Example using Basic Rule Implementations

[Filename: java/Digester/GetMillionCities2.java]

```
import java.io.File;
import java.util.HashSet;
import org.apache.commons.digester3.Digester;
import org.apache.commons.digester3.Rule;
import org.apache.commons.digester3.ObjectCreateRule;
import org.xml.sax.Attributes;

public class GetMillionCities2 {

  public static class CityCollection extends HashSet<City>  {
      public void addCity(City c) {
        if (c.population > 1000000) this.add(c); System.out.println("ADD TO COLL " + c.name);}
  }
  public static void listCities(CityCollection cities) {
      for (City c : cities) {
         System.out.println(c.name + " " + c.country + " " + c.population); }
  }

  public static class City {
    String name = null;
    String country = null;
    int population = -1;
    public void setName(String n) { if (name == null) name = n; }
    public void setCountry(String code) { this.country = code; }
    // note: all PCDATA/CDATA values are strings!
    public void setPopulation(String pop) { this.population = new Integer(pop); }
  }
  // continue next page
```



```
  // continue next page



  public static void main(String[] args) {
    File mondial = new File("mondial.xml");
    final Digester digester = new Digester();
    digester.push(new CityCollection());
    System.out.println("INIT: " + digester.peek());

    // digester.addObjectCreate("*/city", City.class);
    digester.addRule("*/city", new ObjectCreateRule(City.class){
      public void begin(String namespace, String name, Attributes attrs) throws Exception {
          digester.push(new City());  // equivalent: super.begin(namespace, name, attrs);
          System.out.println("BEGIN: " + digester.peek() + digester.peek(1)); }
      public void end(String namespace, String name) throws Exception {
          System.out.println("END: " + digester.peek());
          // if (((City)digester.peek()).population > 1000000) {...}
          digester.pop();  // equivalent: super.end(namespace,name);
          System.out.println("ENDEND: " + digester.peek()); }
    } );

    // digester.addSetNext("*/city", "addCity");
    digester.addRule("*/city", new Rule(){
    public void begin(String namespace, String name, Attributes attrs) throws Exception {}
      public void end(String namespace, String name) throws Exception {
          System.out.println("ADD: " + digester.peek(0) + digester.peek(1));
          ((CityCollection)(digester.peek(1))).addCity((City)digester.peek(0)); }
    } );

    digester.addSetProperties("*/city", "country", "country");
    digester.addBeanPropertySetter("*/city/name");
    digester.addCallMethod("*/city/population", "setPopulation", 1);
    digester.addCallParam("*/city/population", 0);

    try {
        digester.setValidating(false);
```

```
    digester.addSetProperties("*/city", "country", "country");
    digester.addBeanPropertySetter("*/city/name");
    digester.addCallMethod("*/city/population", "setPopulation", 1);
    digester.addCallParam("*/city/population", 0);

    try {
        digester.setValidating(false);
        final CityCollection cities = digester.parse(mondial);
        System.out.println("####  now listing cities #### ");
        listCities(cities);
      } catch (Exception e) { e.printStackTrace(); }
  }
}
```

- addObjectCreate(...) replaced by addRule(...),
- addSetNext(...) replaced by addRule(...)

Rule Application Order

If multiple rules match in a situation:

- if startElement(), rules are applied in the order they have been added to the digester,
- if endElement(), rules are applied in *reverse* order.
- this guarantees if multiple rules push/pop, pop() is are applied in the correct order.

Digester: Producing (sysout) Streaming Output

- Rules can be designed to produce output immediately during the processing:

[Filename: java/Digester/PrintMillionCities.java]

```
import java.io.File;
import org.apache.commons.digester3.Digester;
import org.apache.commons.digester3.Rule;
import org.xml.sax.Attributes;   ###  note: import from SAX API ...

public class PrintMillionCities {

  public static class City {
    String name = null;
    String country = null;
    int population = -1;
    public void setName(String n) { if (name == null) name = n; }
    public void setCountry(String code) { this.country = code; }
    // note: all PCDATA/CDATA values are strings!
    public void setPopulation(String pop) { this.population = new Integer(pop); }
  }
  // continue next page
```

```
  public static void main(String[] args) {
    File mondial = new File("mondial.xml");
    final Digester digester = new Digester();

    digester.addObjectCreate("*/city", City.class);

    digester.addRule("*/city", new Rule(){
      public void begin(String namespace, String name, Attributes attrs) throws Exception {
        System.out.println("start city"); }
      public void end(String namespace, String name) throws Exception {
        City c = (City)(digester.peek(0));
        if (c.population > 1000000)
           System.out.println(c.name + " " + c.country + " " + c.population); }
    } );

    digester.addSetProperties("*/city", "country", "country");
    digester.addBeanPropertySetter("*/city/name");
    digester.addCallMethod("*/city/population", "setPopulation", 1);
    digester.addCallParam("*/city/population", 0);

    try { digester.setValidating(false);
          digester.parse(mondial);
      } catch (Exception e) { e.printStackTrace(); }   } }
```


Digester: A “Native” Stack Application - Evaluation of Artihmetic Term Trees

- Arithmetic terms as trees
- depth-first-evaluation: push values on the stack, operator: pop two values, compute the results, push it on the stack.
- Possible XML representation of terms:

```
<term><!-- ((90 div (19 - (3*8)))+3) -->
  <plus>
    <div>
      <val>90</val>
      <minus>
        <val>19</val>
        <mult>
          <val>3</val>
          <val>8</val>
        </mult>
      </minus>
    </div>
    <val>3</val>
  </plus>
</term>
```

[Filename: java/Digester/arithm-tree-example.java]

[Filename: java/Digester/ArithmTerms.java]

```
import java.io.File;
import org.apache.commons.digester3.Digester;
import org.apache.commons.digester3.Rule;

public class ArithmTerm {
  public static void main(String[] args) {
    File term = new File("arithm-tree-example.xml");
    final Digester digester = new Digester();
    //  Rule: push values as Integers on the stack
    digester.addRule("*/val", new Rule(){
     public void body(String namespace, String name, String text) throws Exception {
        digester.push(new Integer(Integer.parseInt(text)));
        System.out.println("value:" + text );
      }
    } );
    //  Rule: operators: combine the two top stack values accordingly:
    digester.addRule("*/plus", new Rule(){
      public void end(String namespace, String name) throws Exception {
        int n = (Integer)(digester.pop()) + (Integer)(digester.pop());
        digester.push(new Integer(n));
        System.out.println("plus: " + n); }
      } );                                  585

    digester.addRule("*/minus", new Rule(){
      public void end(String namespace, String name) throws Exception {
        int min = (Integer)(digester.pop());
        int n = (Integer)(digester.pop()) - min;
        digester.push(new Integer(n));
        System.out.println("minus: " + n ); }
      } );
    digester.addRule("*/mult", new Rule(){
      public void end(String namespace, String name) throws Exception {
        int n = (Integer)(digester.pop()) * (Integer)(digester.pop());
        digester.push(new Integer(n));
        System.out.println("mult: " + n); }
    } );
    digester.addRule("*/div", new Rule(){
      public void end(String namespace, String name) throws Exception {
        Integer div = (Integer)(digester.pop());
        int n = (Integer)(digester.pop()) / div;
        digester.push(new Integer(n));
        System.out.println("div: " + n); }
    } );
    try { digester.setValidating(false);
          Integer result = digester.parse(term);
  System.out.println("result: " + result);
      } catch (Exception e) { e.printStackTrace(); }   } }
```

11.9 Aside: Use of Date and Time Datatypes

- XML Schema: simple datatypes for dateTime
- represented by String literals in attribute values or text contents
  - xs:dateTime: *yyyy-mm-dd*T*hh:mm:ss*[.*xx*][{+|-}*hh:mm*]
  - xs:time: *hh:mm:ss*[{+|-}*hh:mm*]
  - xs:duration: P[*n*Y][*n*M][*n*D][T[*n*H][*n*M][*n*[.*n*]S]], where *n* can be any natural number
  - xs:dayTimeDuration, xs:yearMonthDuration: restrictions of xs:duration.
- for XQuery handling with specific operations (similar to those known from SQL) cf. Slide 309.
- map to appropriate classes for processing by Java (and by this e.g. with JDBC from and to SQL databases).

Aside: Date and Time in Java

Java provides several classes for handling date and time:

- Datatype: import java.util.GregorianCalendar;

Create from string representation:

- import java.text.DateFormat; import java.text.SimpleDateFormat;

```
public static String datedefaultpattern = "yyyy-MM-dd'T'HH:mm:ss";
// input: string s (following the XML Schema pattern)
DateFormat df = new SimpleDateFormat(datedefaultpattern);
GregorianCalendar typedvalue = df.parse(s);
// result: typedvalue as an object
```

- see java.util.GregorianCalendar for method documentation.
- JAXB: uses javax.xml.datatype.XMLGregorianCalendar class (see e.g. the generated gensrc/Organization.java)