Chapter 11
Algorithms and APIs

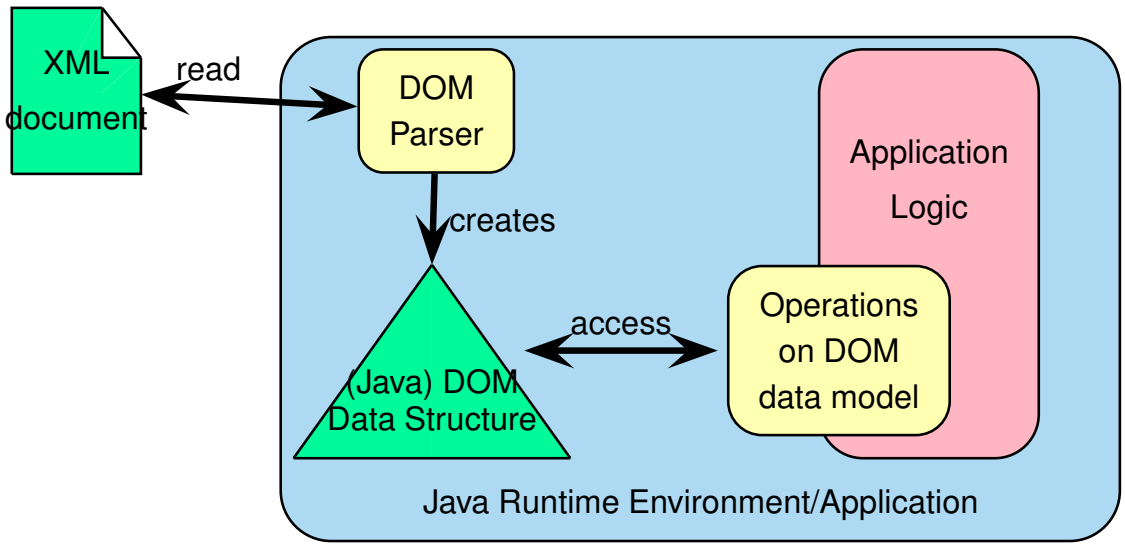
- XML as a data structure:
  - *abstract datatype* with API: DOM
  - (mainly main-memory) implementations; used e.g. in Java applications
  - low-level API with variable-based access
- Databases?
  - high-level API: XPath, XQuery
  - mapping to relational model (Oracle, IBM DB2) or ObjectTypes (Oracle, DB2)
  - “Native” storage: Software AG-Tamino
  - classical database functionality: multiuser, transactions, recovery

Algorithms and APIs (Cont’d)

- Stream Processing:
  - XML data transfer as sequence of events
  - SAX (Simple Application Interface for XML), StAX (Streaming API for XML)
- XML as Data Exchange Format in Web Services
  - serialize application objects as XML
  - SOAP: generic [not discussed in this course]
  - JAXB: ”model-aware” infrastructure

11.1 DOM

- DOM (Document Object Model) defines a platform- and language-independent object-oriented *interface* (i.e., an *abstract datatype*) for generating, processing and manipulating XML data.



DOM

- DOM is a specification of an interface/abstract datatype for the XML data model, *not a* data model and *not a* programming language!
- implementations in Java, C++, etc; usually main-memory-based;
  specialized Java interface definitions:
  - this course: JDOM jdom.jar, org.jdom.*
  - another alternative: dom4j
  - not recommended: org.w3c.dom.* (the plain dom is an implementation that exists in nearly all programming languages and does not make use of Java's advantages);
- language base of the DOM specification: OMG-IDL
- Main-memory-based: only for relatively small application programs
  (most of the "lightweight"-tools used in the course are internally based on DOM)

DOM: Principles

- only one document in a DOM
- step-by-step-access to the data:
  based on variable assignments in the surrounding imperative/object-oriented programming language and on iterators (cf. proceeding in the network data model):
  - document: represents the complete document,
    - Query-Methods, e.g. NodeList getElementsByName(string)
  - class “Node”: getNodeType(), getChildren(), getFirstChild(), getNextSibling(), getParentNode(), ...
  - class “Element”: getName(), getAttributes(), getContent(), ...
  - class “Attribute”: getName(), getValue(), ...
  - corresponding methods for generating and changing nodes.
- additionally, XPath and XSLT can be applied to instances of Document and Element;
- based on DOM, XPath and XQuery can be implemented (cf. Apache Xerces (XML/DOM)/Xalan (XSLT)/Xindice (DB))
- often inefficient (no indexes, query optimization)

DOM – sample code fragment: Stepwise access

(taken from LanguageElement.java from MARS, using JDOM)

```
// given: Element element;

protected Set<InputVariableDefinition> getInputVariableDefinitions(
          boolean includeJoinVariables)
{ Set<InputVariableDefinition> definitions = new HashSet<InputVariableDefinition>();

  @SuppressWarnings("unchecked")
  List<Element> elements = element.getChildren();
  for (Element e : elements)
  { String elementName = e.getName();
    if (!elementName.equals("Opaque"))
    { String name = e.getAttributeValue("name", "");
      InputVariableDefinition variable = null;
      if (elementName.equals("has-input-variable"))
        variable = new InputVariableDefinition(name, InputVariableDefinition.INPUT);
      else if (elementName.equals("uses-variable") && includeJoinVariables)
        variable = new InputVariableDefinition(name, InputVariableDefinition.USE);
      if (variable != null)
      { String use = e.getAttributeValue("use", "");
        if (use.length() > 0) variable.setUse(use);
        definitions.add(variable);
      }
    }
  }
  return definitions;
}
```

DOM – sample code fragment: XPath

(taken from ServiceRegistry.java from MARS)

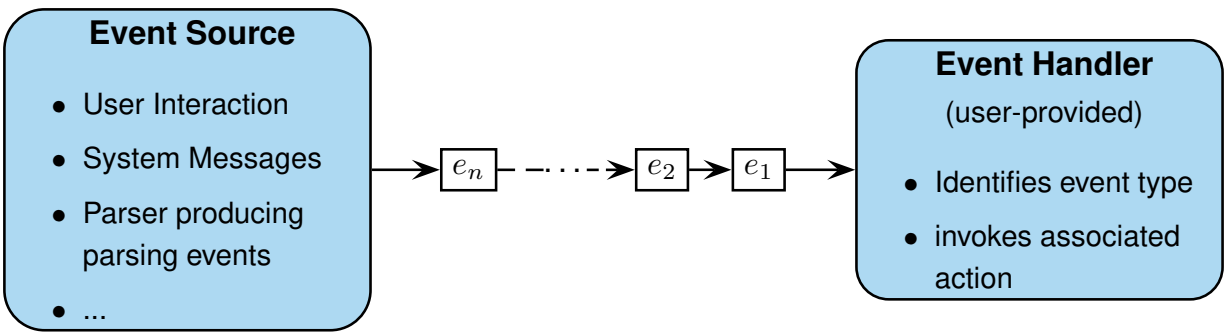
- similar to the JDBC statement concept for SQL in Java:

```
public Element getTaskDescr(Element serviceDescr, String task)
{
  Element taskDescr = null;
  try
  {  XPath xpath = XPath.newInstance(
        "./lsr:has-task-description/lsr:TaskDescription[" +
        "contains(lsr:describes-task/@rdf:resource,$task)]");
     xpath.addNamespace(Namespaces.RDF_NS);
     xpath.addNamespace(Namespaces.MARS_NS);
     xpath.addNamespace(Namespaces.LSR_NS);
     xpath.setVariable("task", task);
     taskDescr = (Element) xpath.selectSingleNode(serviceDescr);
  }
  catch (Exception e) {...}
}
```


11.2 SAX: Event-Based XML Processing

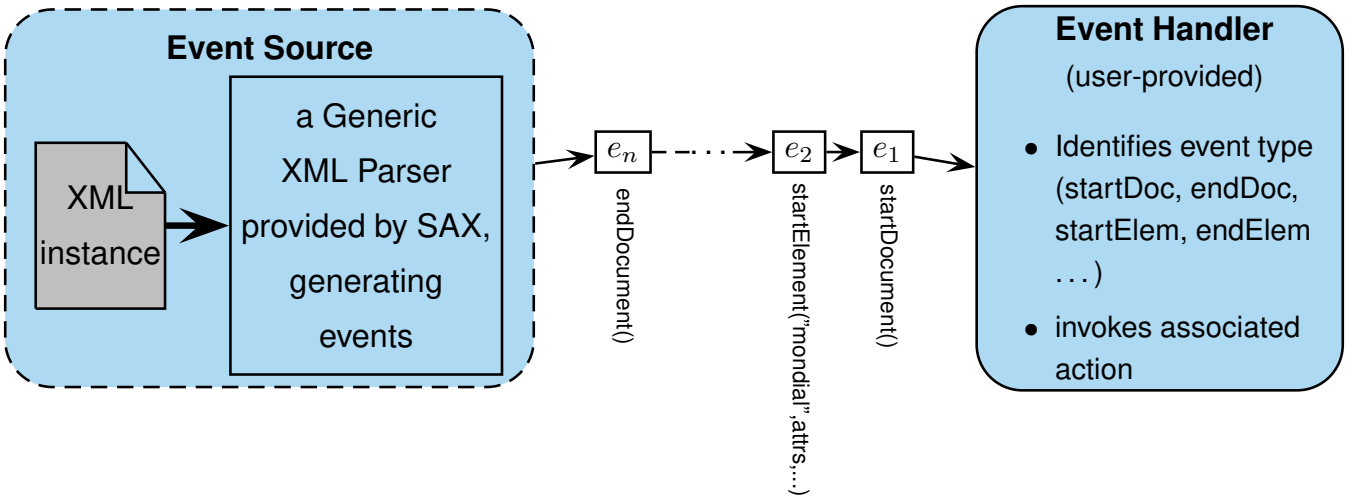
- SAX ("The Simple API for XML") is an *event-based interface/model*

Event-Based Processing as a *general Design Pattern*



- The application programmer provides the Event Handler implementation, containing actions for each type of event;
- kind of *rule-based*;
- programmer is *not* in charge of the control flow

EVENT-BASED XML PARSING WITH SAX



Represents/processes an XML document as a sequence of events (depth-first traversal), e.g.

- startDocument(), endDocument()
- startElement(Name, attributesList) – attributes not split
- endElement(Name)
- characters(string)

XML PARSING WITH SAX

SAX: parse XML from a file (in general: char stream).

- import classes: javax.xml.parsers.*, org.xml.sax.*
- a generic XML Parser is parameterized with a *Content Handler* (plus *Error Handler*, *DTD Handler*, and *EntityResolver*) implementation.
- The most trivial Content Handler is the *DefaultHandler* that does nothing: the document is parsed, events are detected, but no action is performed (DTD / XML Schema validation can be switched on).
- Event handler programmed wrt. a "*push* API".
- Normally, the user-provided Content Handler extends the DefaultHandler, overwriting (some of) its Event Methods.
- With the content handler implementation, the user provides "actions" in form of Java code, associated with specific events (and even dependent on context information).
- If during parsing of the XML document, a specific event occurs, the code of the associated action from the content handler is invoked (*"callback"*).

SAX: Applications

Only events are signaled: linear processing based on incoming sequence of events.

- ... among many other things, one can generate a DOM tree structure,
- validation according to a DTD (using the automaton as given on Slide 176) in linear time,
- stream-processing of XML input
  - start processing already when input document is not yet complete
  - filtering for elements that are relevant for a given application
  - linear search for something, e.g., names of countries
    (Exercise: sketch the behavior of the event handler on relevant events)
  - if the stream is a list of elements of the same structure:
    generate a database entry for each element (use JDBC)
- if necessary: application needs to maintain context.

SAX Example Code

Consider a very simple application that

- detects all elements with attributes
- for each element, output the element's name
- for each element, output the name-value pairs of its attributes

```
>java PrintAttributes mondial.xml > bla.out
>less bla.out
element: country
 - attribute: 'car_code' value: 'AL' type: 'ID'
 - attribute: 'area' value: '28750' type: 'CDATA'
 - attribute: 'capital' value: 'cty-cid-cia-Albania-Tirane' type: 'IDREF'
 - attribute: 'memberships' value: 'org-BSEC org-CE org-CCC org-ECE org-EBRD org-EU ...' type: 'IDREFS'
element: encompassed
 - attribute: 'continent' value: 'europe' type: 'IDREF'
 - attribute: 'percentage' value: '100' type: 'CDATA'
element: ethnicgroups
 - attribute: 'percentage' value: '3' type: 'CDATA'
element: ethnicgroups
 - attribute: 'percentage' value: '95' type: 'CDATA'
element: religions
 - attribute: 'percentage' value: '70' type: 'CDATA'
...
```

Class “PrintAttributes.java”:

```
import java.io.IOException;
import javax.xml.parsers.*;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class PrintAttributes {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("usage: PrintAttributes <url>");
            System.exit(1);
        }
        String url = args[0];    // ... prepare a contentHandler:
        DefaultHandler handler = new ContentHandlerPrintAttributes(
                "printing attributes of document at url '" + url + "'");
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            SAXParser parser = factory.newSAXParser();
            parser.parse(url, handler);  // <<<<<<<<< and now it runs ...
        } catch (IOException e1) {
            e1.printStackTrace();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        }
    }
}
```

[see java/SAX/PrintAttributes.java]

Class “ContentHandlerPrintAttributes.java”:

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ContentHandlerPrintAttributes extends DefaultHandler {
    public ContentHandlerPrintAttributes(String message) {
        System.out.println(message);
    }

    // react on opening elements:
    public void startElement(String url, String localName, String qName,
            Attributes attrs) throws SAXException {
        if (attrs.getLength() > 0) {
            String elementName;
            if(qName == null || qName.equals("")) elementName = localName;
            else elementName = qName;
            System.out.println("element: " + elementName);
            for (int i = 0; i < attrs.getLength(); i++) {
                System.out.println(" - attribute: '" + attrs.getQName(i)
                        + "' value: '" + attrs.getValue(i) + "' type: '"
                        + attrs.getType(i)+"'");
            }
            System.out.println();
        }
    }
    // methods for endElement(), startDocument(), endDocument(), characters() omitted
    // all other "parsing events" are ignored in this case
}
```

[see java/SAX/ContentHandlerPrintAttributes.java]

SAX: Applications to XPath Query Answering

Forward queries

XPath-queries like //country[@car_code='D']/population can be answered very (time- and memory-)efficient,

- use the sequence of events (linear)
- maintain some context (often LOGSPACE/additional LOGTIME sufficient)

... works only for queries, that contain only forward steps,

General queries

which XPath expressions can be *transformed* in equivalent forward-expressions (and with what efforts)?

- "XPath: Looking forward"; F. Bry et al ; 2002; LMU München
- theory: complexity, connections to linear temporal logic
  For every linear temporal logic formula that uses past and future operators, there is an equivalent formula that uses only future operators
  ... but in general of exponential size.

11.3 StAX - The Streaming API for XML

SAX appears somewhat sturdy to many software developers, since passing the control flow to a subordinate parser is not a common design pattern.

- like SAX: on-the fly processing, no in-memory representation for good performance
- learnt from SAX: idea of "XML Event Stream"
  import: javax.xml.stream.* (rt.jar)
  - can be used as inputs and outputs and inputs ...
    TODO: piping through successive XML-to-XML transformations/stylesheets?
  - not based on char, but "XML Stream" as a basic concept,
  - a char stream (File, HTTP) can be converted into an XML Event Stream by an XML parser; see example's main() method.
- StAX on XML Event Input Stream:
  control flow is not passed to the parser (**unlike** *SAX*), but XML events are accessed using an *iterator*, controlled by the Java program using the StAX API (*Pull-API*).
  [Note: iterators are a common design pattern, not only applied to collections, but as we see here also to streams: init(), next(), ...]
- application code: same as for SAX, only operational embedding done differently.

StAX Example: Exam Registration

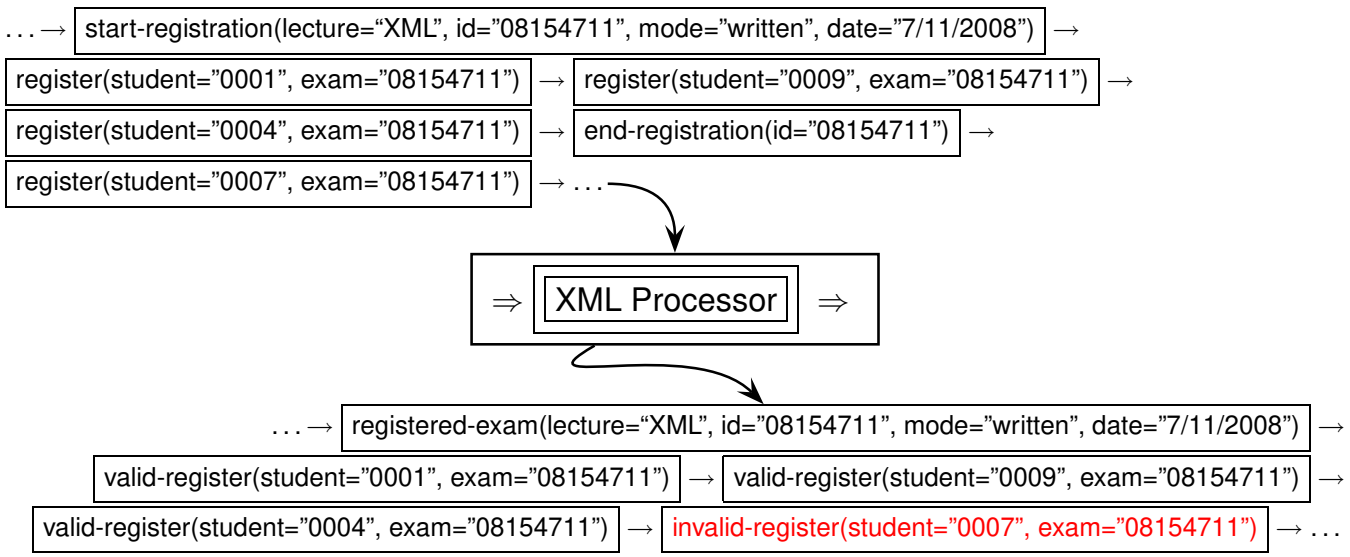
Assume the administration of exams in a student's office ("Prüfungsamt"):

- The *subject* (e.g., "Semi-structured Data and XML") and ID of lectures/exams,
- whether the exam is *written* or *oral*,
- for written exams, the date of the exam,
- for oral exams, a number of dates is given when the single exams are held.
- the registration period *starts* when receiving an incoming XML message `start-registration`
- the registration period *ends* when receiving an incoming XML message `end-registration`
- for all students that did (`register`) correctly, the student's relevant details are extracted and written to some output stream (`valid-register`; in the example, we use `STDOUT`.)
- students that register before beginning or after the end of registration, are not accounted for the exam; an XML message `invalid-register` goes to `STDOUT`,

StAX Example: Exam Registration (Cont'd)

- the registration data of the students comes in via a continuous input stream;
- the program should allow the management of registrations for multiple exams at one time (all incoming over the same input stream).

Example stream:



StAX Example Cont'd:

Consider the following XML sequence as input stream:

```
<?xml version="1.0" encoding="UTF-8"?>
<stream>
  <register student="0007" exam="08154711"/>
  <start-registration id="08154711" mode="written">
    <subject>Semistructured Data and XML</subject>
    <date>07/11/2008</date>
  </start-registration>
  <register student="0001" exam="08154711"/>
  <register student="0009" exam="08154711"/>
  <start-registration id="12345678" mode="oral">
    <subject>Dental Hygiene</subject>
    <dates>
      <date>17/9/2008</date>
      <date>18/9/2008</date>
    </dates>
  </start-registration>
  <end-registration id="12345678"/>
  <register student="0004" exam="08154711"/>
  <register student="0004" exam="12345678"/>
  <register student="0007" exam="12345678"/>
  <end-registration id="08154711"/>
  <register student="0007" exam="08154711"/>
</stream>
```

[Filename: java/StAX/exam.xml]

StAX Example Cont'd (2):

Code for the `Exam` bean, containing the exam's properties and some constants):

```
import java.util.ArrayList;
import java.util.List;

public class Exam {
  public static final String DATE = "date";
  public static final String SUBJECT = "subject";
  public static final String ID = "id";
  public static final String MODE = "mode";
  public static final String DATES = "dates";
  public static final String STARTOFREG = "start-of-registering";
  public static final String ENDOFREG = "end-of-registering";

  private String id;
  private boolean oral;
  private String subject;
  private String date;
  private List<String> dates;
  private boolean registeringClosed = false;
  private String startOfReg;
  private String endOfReg;

  public Exam(String id, String mode) {
    this.id = id;
    this.oral = "oral".equals(mode);
    this.dates = new ArrayList();
  }
```

```
  public String getId() { return id; }
  public void setDate(String date) { this.date = date; }
  public String getDate() { return date; }
  public void setDates(List<String> dates) {this.dates = dates; }
  public List<String> getDates() { return dates; }
  public void setSubject(String subject) { this.subject = subject; }
  public String getSubject() { return subject; }
  public boolean isOral() { return oral; }
  public boolean isWritten() { return (!oral); }

  public String getMode() {
    if (oral) return "oral";
    return "written";
  }
  public boolean isRegisteringClosed() {
    return registeringClosed;
  }
  public void setRegisteringClosed(boolean registeringClosed) {
    this.registeringClosed = registeringClosed;
  }
  public String getEndOfReg() {
    return endOfReg;
  }
  public String getStartOfReg() {
    return startOfReg;
  }
  public void setStartOfReg(String startOfReg) {
    this.startOfReg = startOfReg;
  }
  public void setEndOfReg(String endOfReg) {
    this.endOfReg = endOfReg;
  }
}
```

[Filename: java/StAX/Exam.java]

StAX Example Cont'd (3):

Code for the main parser class, containing the `main` method:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.OutputStream;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLStreamWriter;


public class ExamStreamParser {

  FileInputStream inputStream;
  OutputStream outputStream;

  public ExamStreamParser(FileInputStream in, OutputStream out) {
    this.inputStream = in;
    this.outputStream = out;
  }

  public void startParsing() {
    try {
      XMLInputFactory inputFactory = XMLInputFactory.newInstance();
      XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
      XMLStreamReader parser = inputFactory.createXMLStreamReader(inputStream);
      XMLStreamWriter writer = outputFactory.createXMLStreamWriter(outputStream);
      Exam currentExam = null;
      Map<String,Exam> exams = new HashMap<String,Exam>();
          boolean goOn = true;
```

```
while (goOn) {
int event = parser.next();
switch(event) {
case XMLStreamConstants.END_DOCUMENT:
   parser.close();
   writer.flush();
   writer.close();
   goOn = false;
   break;
case XMLStreamConstants.START_ELEMENT:
  // start-registration and its subelelements
  if("start-registration".equals(parser.getLocalName())) {
      currentExam = new Exam(parser.getAttributeValue(null, Exam.ID),parser.getAttributeValue(null, Exam.MOD
      currentExam.setStartOfReg(getDate());
      break;
    }
  if(Exam.SUBJECT.equals(parser.getLocalName())) {
      currentExam.setSubject(parser.getElementText()); break;
    }
  if(Exam.DATE.equals(parser.getLocalName())) {
      if(currentExam.isWritten()) currentExam.setDate(parser.getElementText());
      else currentExam.getDates().add(parser.getElementText());
      break;
    }
  if("end-registration".equals(parser.getLocalName())) {
      String examId = parser.getAttributeValue(null,Exam.ID);
    Exam exam = exams.get(examId);
    if(exam == null) {
      System.err.println("no such exam with id '"+examId+"' open for registration!");
      break;
    }
      exam.setEndOfReg(getDate());
      exam.setRegisteringClosed(true);
      break;   // no output is generated.
    }
  // register and its subelements
  if("register".equals(parser.getLocalName())) {
      String studentId = parser.getAttributeValue(null, "student");
      String examId = parser.getAttributeValue(null, "exam");
      if(exams.containsKey(examId)) {
        Exam exam = exams.get(examId);
        if(! exam.isRegisteringClosed()) {
          writer.writeStartElement("valid-register");
        writer.writeAttribute("student", studentId);
        writer.writeAttribute("exam", examId);
        writer.writeEndElement();
      } else {
          writer.writeStartElement("invalid-register");
          writer.writeAttribute("student", studentId);
        writer.writeAttribute("exam", examId);
        writer.writeStartElement("message");
        writer.writeCharacters("invalid registration! registration for exam '" + exam.getId()
          + "' (" + exam.getSubject() + ") has ended on " + exam.getEndOfReg());
        writer.writeEndElement();
      }
      } else {
        writer.writeStartElement("invalid-register");
        writer.writeAttribute("student", studentId);
      writer.writeAttribute("exam", examId);
      writer.writeStartElement("message");
      writer.writeCharacters("invalid registration! exam '"+examId+"' is not (yet?) open for registration.")
      writer.writeEndElement();writer.writeEndElement();
      }
      writer.writeCharacters("\n");
      break;
    }
    break;
```



```
        case XMLStreamConstants.END_ELEMENT:
          if("start-registration".equals(parser.getLocalName())) {
            exams.put(currentExam.getId(),currentExam);
            writer.writeStartElement("registered-exam");
            writer.writeAttribute(Exam.ID, currentExam.getId());
            writer.writeAttribute(Exam.MODE, currentExam.getMode());
            writer.writeCharacters("\n  ");
              writer.writeStartElement(Exam.SUBJECT);
            writer.writeCharacters(currentExam.getSubject());
            writer.writeEndElement();
            writer.writeCharacters("\n  ");
              writer.writeStartElement(Exam.STARTOFREG);
            writer.writeCharacters(currentExam.getStartOfReg());
            writer.writeEndElement();
            writer.writeCharacters("\n  ");
              if(currentExam.isWritten()) {
              writer.writeStartElement(Exam.DATE);
              writer.writeCharacters(currentExam.getDate());
              writer.writeEndElement();
            } else {
              writer.writeStartElement(Exam.DATES);
              for(Iterator<String> i=currentExam.getDates().iterator();i.hasNext();) {
                writer.writeStartElement(Exam.DATE);
                writer.writeCharacters(i.next());
                writer.writeEndElement();
              }
              writer.writeEndElement();
            }
              writer.writeCharacters("\n");
              writer.writeEndElement();writer.writeCharacters("\n");
            currentExam = null; // it's better to provoke
            // a nullpointer exception than to edit the wrong exam object
            break;
          }
        }
      }
    } catch (XMLStreamException e) {
      e.printStackTrace();
    }
  }

  private String getDate() {
    DateFormat format = new SimpleDateFormat();
    Date date = new Date();
    return format.format(date);
  }

  public static void main(String[] args) {
    try {
      ExamStreamParser examStreamParser = new ExamStreamParser(new FileInputStream(args[0]), System.out);
      examStreamParser.startParsing();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```

StAX Summary

- Pro's:
  - Quick and performant like SAX,
  - is a "Pull API"
  - can handle XML Streams
- Con's:
  - Common pattern: huge `switch` command performing a case differentiation over all relevant events

$\Rightarrow$ look and feel is not much different from writing a SAX event handler $\Rightarrow$ Push vs Pull: *so what?*

- can easily produce XML output via `XMLStreamWriter` (e.g. to another StAX application) $\Rightarrow$ same feature can be provided by a SAX Event Handler as well $\Rightarrow$ *so what?*
- TODO: can SAX also be applied on an XML events' input stream?

SAX and StAX: Applications

Stream-based processing can be applied to XML data on multiple levels:

- low-level applications:
  SAX is often used for building a DOM from ASCII XML input: "opening tag with attributes", "text", "closing tag" can immediately be translated into the DOM constructors.
- low-level streaming of an XML instance:
  answering XPath (forward-axes only) queries; optionally maintaining some context (e.g., stack).
- higher level "application-level events":
  the XML stream is not seen as the traversal of a large instance, but as a sequence of (independent) XML fragments that are seen as application-level events (cf. the StAX example)
  [RFID applications, time series of stock quotes, RSS feeds]

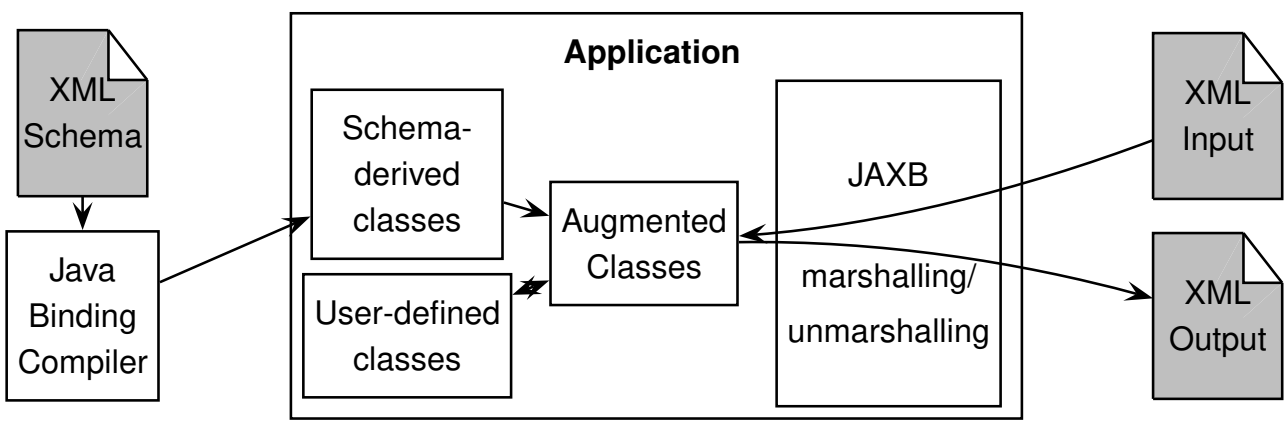
11.4 JAXB - The Java API for XML Binding

- Part of the Java Web Services Developer Pack
- SUN's "Java Web Service Tutorial"
  http://java.sun.com/webservices/tutorial.html
- 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 implemened 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 instance documents into Java objects,
- methods for *marshalling* Java objects back into XML instance documents.

Architecture



JAXB - EXAMPLE

[Filename: java/JAXB/books.xml]

```
<?xml version="1.0"?>
<Collection>
  <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>
        <Discount>Buy one get Web Services Part 1 free</Discount>
      </promotion>
      <publicationDate>2002-11-01</publicationDate>
    </book>
  </books>
</Collection>
```


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="Collection">
 <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>
```

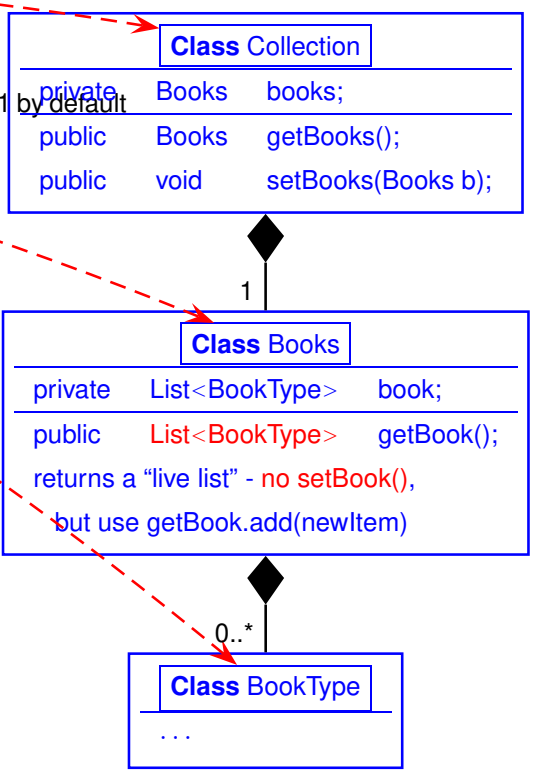
JAXB HowTo

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

```
## Java Architecture for XML Binding (JAXB)

 mkdir myjaxb
 cd myjaxb
 mkdir classes gen-src
 ## download jaxb and adjust below HOME:
 java -jar JAXB2_20070122.jar
 export JAXB_HOME=whereeveritis/jaxb20
 export JAXB_LIB=$JAXB_HOME/lib
 export JAXB_JAR=$JAXB_LIB/jaxb-api.jar:$JAXB_LIB/jaxb-libs.jar:$JAXB_LIB/jaxb-xjc.jar

 $JAXB_HOME/bin/xjc.sh -p JAXBbooks books.xsd -d gen-src
 # created classes can the be found in gen-src/JAXBbooks
 javac -d classes -classpath $JAXB_JAR ‘find gen-src -name ’*.java’‘
 javac -d classes -classpath $JAXB_JAR:classes JAXBbooks.java
 java -classpath classes:$JAXB_JAR JAXBbooks books.xml

 $JAXB_HOME/bin/xjc.sh -p JAXBmondial mondial-jaxb.xsd -d gen-src
 javac -d classes -classpath $JAXB_JAR ‘find gen-src -name ’*.java’‘
 javac -d classes -classpath $JAXB_JAR:classes JAXBmondial.java
 java -classpath classes:$JAXB_JAR JAXBmondial mondial-jaxb.xml
```

JAXB: Binding XML Schema to Java Classes

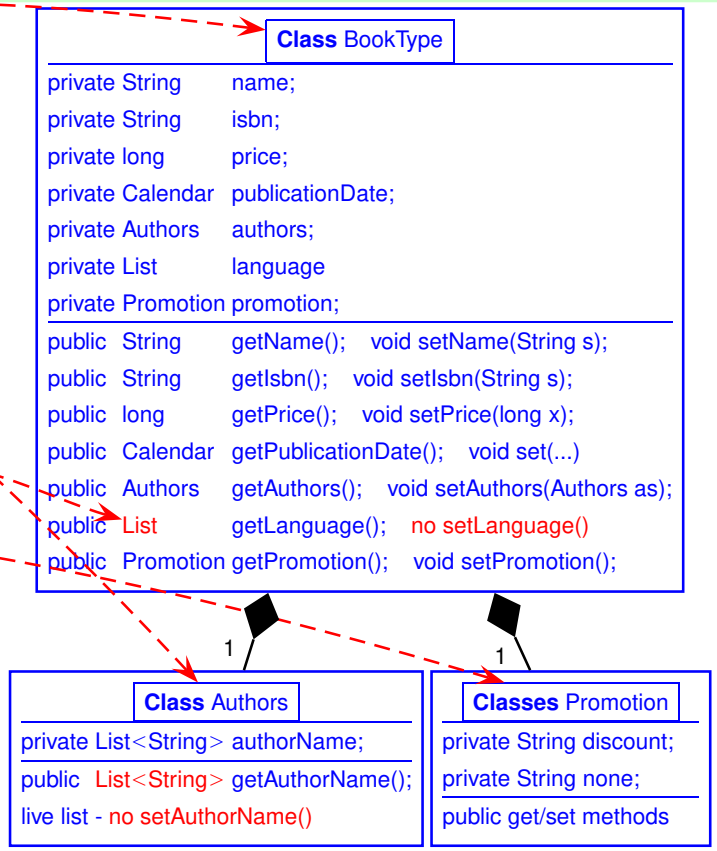
```
<xs:element name="Collection">
  <xs:complexType>
   <xs:sequence>
    <xs:element name ="books">      minOccurs = maxOccurs = 1 by default
      <xs:complexType>
       <xs:sequence>
        <xs:element name="book" type="bookType"
            minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
      </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```



- elements that have complexTypes are mapped to classes (for local type declarations: local classes like Collection.Books),
- 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>
```



JAXB - Example Usage

[Filename: java/JAXB/JAXBbooks.java]

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import javax.xml.bind.Marshaller;
import java.util.List;

import java.io.File;
import org.w3c.dom.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;

// import java content classes generated by binding compiler
import JAXBbooks.*;

/**
  * This shows how to use JAXB to unmarshal an xml file
  * Then display the information from the content tree
  */

public class JAXBbooks {

  public static void main (String args[]) {
    try
    {
      JAXBContext jc = JAXBContext.newInstance("JAXBbooks");
      Unmarshaller unmarshaller = jc.createUnmarshaller();
      // unmarshaller.setValidating(true);  not supported in new version

      Collection collection= (Collection)
             unmarshaller.unmarshal(new File( "books.xml"));

      Collection.Books books = collection.getBooks();
      List bookList = books.getBook();
```



```
      for( int i = 0; i < bookList.size();i++ )
      {
        System.out.println("Book  details "   );
        BookType book =(BookType) bookList.get(i);
        System.out.println("Book Name: " +  book.getName().trim());
        System.out.println("Book ISBN: " +  book.getIsbn().trim());
        System.out.println("Book Price: " +  book.getPrice());
        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 (int j = 0; j < authors.getAuthorName().size();j++)
        {
          String authorName = (String) authors.getAuthorName().get(j);
          System.out.println("Author Name " + authorName.trim());
        }
        for (int j = 0; j < book.getLanguage().size();j++)
        {
          String language = (String) book.getLanguage().get(j);
          System.out.println("Language " + language.trim());
        }
        // add an element to a live list:
        book.getLanguage().add("Kisuaheli");
        System.out.println();
      }

      // write the result 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");
      Transformer transformer = factory.newTransformer();
      transformer.transform(docSource, result);
    }catch (Exception e ) {
        e.printStackTrace();
    }
  }
}
```

JAXB - Another Example

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

```
<?xml version="1.0"?>
<mondial>
 <country name="Austria" area="83850" indep_date="1918-11-12"  capital="cty-Austria-
  <population>8023244</population>
  <province name="Styria" area="16386">
   <population>1203000</population>
   <city name="Graz">
    <population year="1994-01-01">238000</population>
   </city>
  </province>
  <province name="Salzburg" area="7154">
   <population>501000</population>
   <city name="Salzburg">
    <population year="1994-01-01">144000</population>
   </city>
  </province>
  <province name="Vienna" area="415">
   <population>1583000</population>
    <city name="Vienna" id="cty-Austria-Vienna">
     <population year="1994-01-01">1583000</population>
    </city>
  </province>
 </country>
</mondial>
```


JAXB - Example: XSD

[Filename: java/JAXB/mondial-jaxb.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="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:integer" 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"/>
     </jaxb:property>
    </xs:appinfo>
   </xs:annotation>
  </xs:attribute>
 </xs:complexType>
```

```
<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:integer" 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:decimal">
   <xs:attribute name="year" type="xs:date" use="optional"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
</xs:schema>
```

- annotation of the country/@capital IDREFS attribute:
  ⇒ public City getCapital()
- countries have at most one population subelement, cities may have several ones.

JAXB - Example Usage

[Filename: java/JAXB/JAXBmondial.java]

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import java.io.File;
import java.util.List;

// import java content classes generated by binding compiler
import JAXBmondial.*;

/**
 * This shows how to use JAXB to unmarshal an xml file
 * Then display the information from the content tree
 */

public class JAXBmondial {

 public static void main (String args[]) {
 try {
  JAXBContext jc = JAXBContext.newInstance("JAXBmondial");
  Unmarshaller unmarshaller = jc.createUnmarshaller();
  //unmarshaller.setValidating(true);

  Mondial mondial =
    (Mondial) unmarshaller.unmarshal(new File("mondial-jaxb.xml"));
  List countryList  = mondial.getCountry();
  Province prov;
  City city;

  for ( int i = 0; i < countryList.size();i++ )
  {
    Country country  = (Country) countryList.get(i);
    System.out.println("Country: " + country.getName()  );
    System.out.println("   pop: " +
        country.getPopulation().getValue());

    // Java knows from the annotation of the IDREF attribute
    // that this is a city

    City c = country.getCapital();
    System.out.println("   cap: " +  c.getName());

    List provList = country.getProvince();
    for (int j = 0; j < provList.size() ; j++)
    {
```

JAXB: Summary

- allows for easy and lightweighted unmarshalling, bean-based manipulation and marshalling of XML data,
- integrates well with XML files and with DOM trees
- higher level of abstraction from XML representation, compared with DOM and SAX.

Issues

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

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

⇒ use own classes that extend the classes and add further methods.

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 Exchange 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.

11.5 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 274.
- 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.
- TODO: check if JAXB declares automatically as date (which class in Java) and whether set-methods of JAXB-created classes automatically convert data.