

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

451

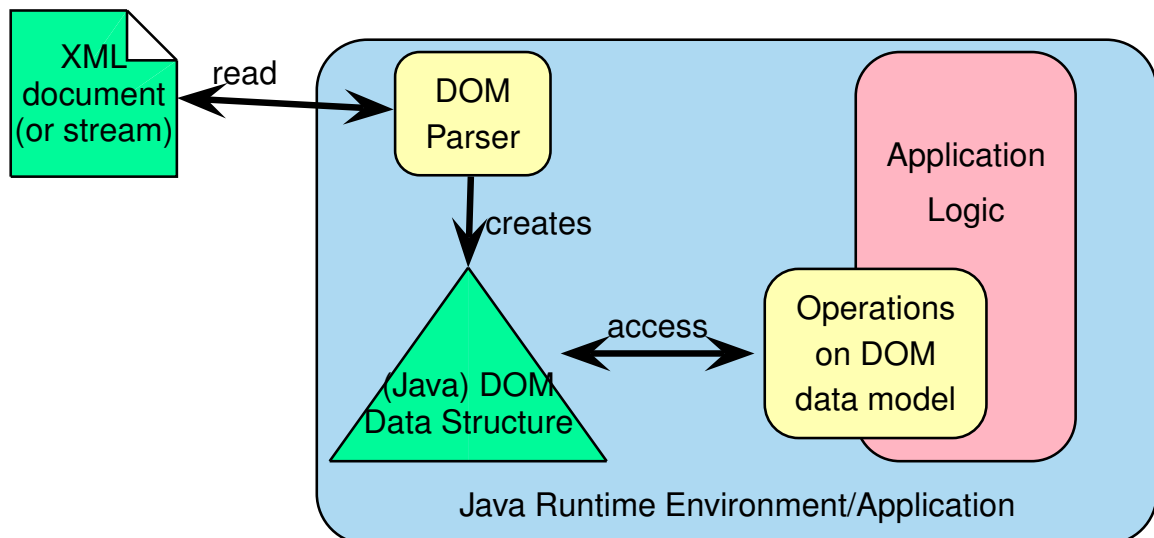
Algorithms and APIs (Cont'd)

- XML as Data Exchange Format in Web Services
 - serialize application objects as XML
 - SOAP: generic [not discussed in this course]
 - JAXB: "model-aware" infrastructure
- Stream Processing:
 - XML data transfer as sequence of events
 - SAX (Simple Application Interface for XML), StAX (Streaming API for XML)
- an intermediate rule-based concept:
 - `apache.commons.digester`

452

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.



453

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)

454

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 getElementsByTagName(string)`
 - class “Node”: `getNodeName()`, `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)

455

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;
}
```

456

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) {...}
}
```

457

11.2 JAXB - The Java API for XML Binding

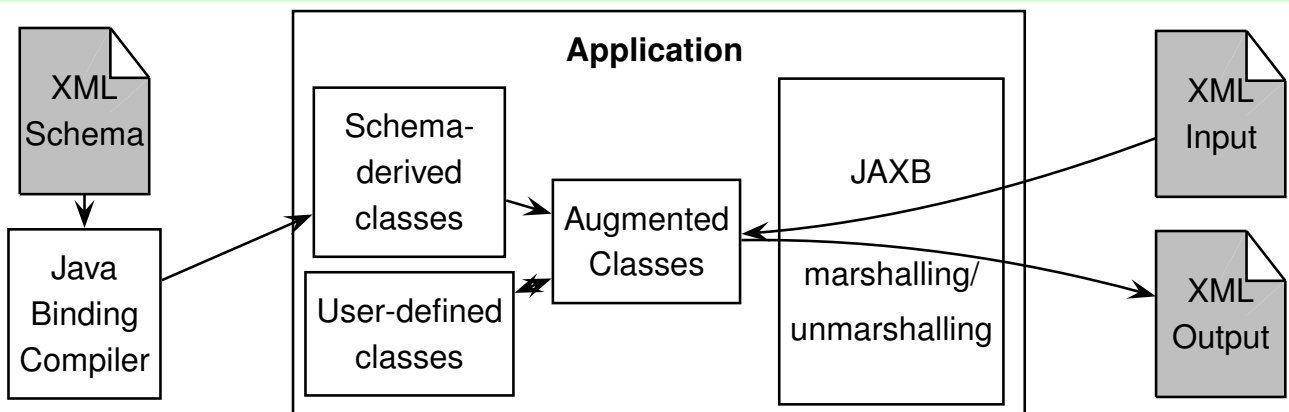
- Part of the Java Web Services Developer Pack
- SUN's "Java Web Service Tutorial"
<http://java.sun.com/webservices/tutorial.html>
- XML elements describe objects with properties,
- correspond to classes of an application,
- derive interface with setX/getX methods (= Java Beans) as skeletons for these classes (automatically generated from an XML Schema description),
- user derives classes from these interfaces by adding behavior,
- application logics implemented by using these classes,
- import/export of XML instances of these classes via generic mappings (derived from the XSD).

458

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



459

JAXB - EXAMPLE

[Filename: java/JAXB/books.xml]

```
<?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>
        <Discount>Buy one get Web Services Part 1 free</Discount>
      </promotion>
      <publicationDate>2002-11-01</publicationDate>
    </book>
  </books>
</BookCollection>
```

- values for `xd:date` and `xs:time` must conform to the syntax required for these XML types (cf. Slide 282)

460

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">
  .....
  <!-- 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`:

```
## since java6, jaxb is included in JDK
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
```

463

JAXB - STEPS

1. `xjc -p pkgname xsdfile -d gensrc`:
 - generates class files (with local classes) from the XML Schema specification (put them into `gensrc/pkgname`),
 - all classes *classname* have only the standard constructor *classname*().
 - generates an *ObjectFactory.java* with methods of the form

```
public classname createclassname() {
    return new classname(); }
```
2. `javac -d . 'find gensrc -name '*.java''`
compiles all files found in the `gensrc` subtree
(generates package directory *pkgname* in ".").
3. write a java application which
 - (a) creates an Unmarshaller (which uses the ObjectFactory),
 - (b) unmarshalls an XML file into objects (using the ObjectFactory).

464

JAXB: Binding XML Schema to Java Classes

```
<xs:element name="BookCollection">
```

```
<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 BookCollection.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

```
Class BookCollection
private Books books;
public Books getBooks();
public void setBooks(Books b);
```

```
Class Books
private List<BookType> book;
public List<BookType> getBook();
returns a "live list" - no setBook(),
but use getBook.add(newItem)
```

```
Class BookType
...
```

465

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();
```

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

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

466

JAXB - Example Usage

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import javax.xml.bind.Marshaller;
import java.io.File;
import java.util.List;
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("JAXBbooks");
      Unmarshaller unmarshaller = jc.createUnmarshaller();

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

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

      // continue next page

      // continue next page

      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());
        }
        XMLGregorianCalendar date = book.getPublicationDate();
        System.out.println("Date " + date);
        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();
      }

      // continue next page

      // write the result to an XML file:
```

[Filename: java/JAXB/JAXBbooks.java]

467

468

```

// add an element to a live list:
book.getLanguage().add("Kiswaheli");
System.out.println();
}

// continue next page

// 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(); }
}}

```

469

JAXB - ANOTHER EXAMPLE: MONDIAL

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

```

<?xml version="1.0"?>
<mondial>
  <country name="Austria" area="83850" indep_date="1918-11-12" capital="cty-Austria-Vienna">
    <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>

```

470

JAXB example: Mondial

- annotation of the country/@capital IDREFS attribute:
⇒ public City getCapital()
- countries have at most one population subelement, cities may have several ones.
- “indep_date” attribute of country and “year” attribute of city/@population are mapped to XMLGregorianCalendar.

from java/JAXB/gensrc/JAXBmondial/Country.java:

```
public class Country {
    :
    @XmlAttribute(name = "indep_date")
    @XmlSchemaType(name = "date")
    protected XMLGregorianCalendar indepDate;
    :
    @XmlSchemaType(name = "IDREF")
    protected City capital;
    :
    public City getCapital() {
        return capital;
    }
}
```

471

JAXB - Example: 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>
<!-- continue next page -->
```

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

472

```

</xs:attribute>
</xs:complexType>
<!-- continue next page -->

<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>

```

473

JAXB - Example Usage

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

```

package JAXBmondial;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
import java.io.File;
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 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++)
                {
                    prov = (Province) provList.get(j);
                    System.out.println("    Province name: " + prov.getName().trim());
                    System.out.println("                area: " + prov.getArea().toString());
                    System.out.println("                pop : " + prov.getPopulation().getValue());
                    List cityList = prov.getCity();
                    for (int k = 0; k < cityList.size() ; k++)
                    {
                        city = (City) cityList.get(k);
                        System.out.println("      City name: " + city.getName().trim());
                    }
                }
            }
        }
    }
}

```

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`
- ⇒ methods of `my_xxx` not applicable
- ⇒ [Different alternatives.](#)

475

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 annotation as `@XmlAttribute` – and then they will be exported when marshalling them.
- ⇒ 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.

476

JAXB - Example Usage with extended class definition

```
package JAXBmondial;
import java.util.List;

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() {
        List provList = getProvince();
        for (int j = 0; j < provList.size() ; j++) {
            Province prov = (Province) provList.get(j);
            List cityList = prov.getCity();
            for (int k = 0; k < cityList.size() ; k++) {
                City city = (City) cityList.get(k);
                System.out.println(city.getName().trim());
            }
        }
    }
}}
```

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

477

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.

478

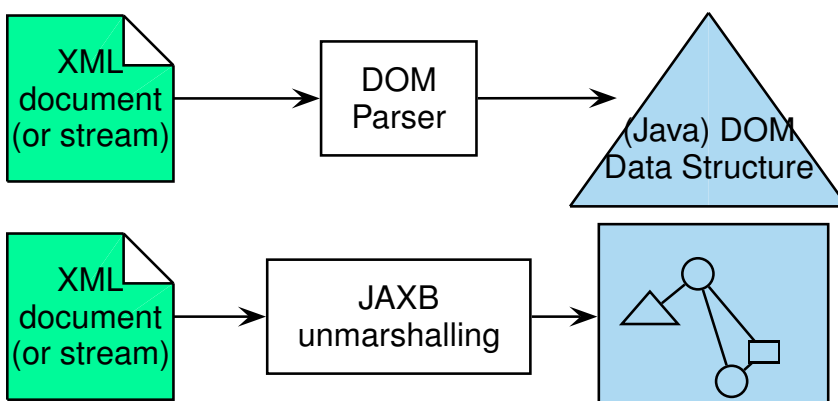
ASIDE: SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

- 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.

481

11.3 XML Stream Processing

- reading from a file or from an HTTP connection both is actually reading char by char from a stream
- the stream is parsed, resulting in something that can be used by application:



482

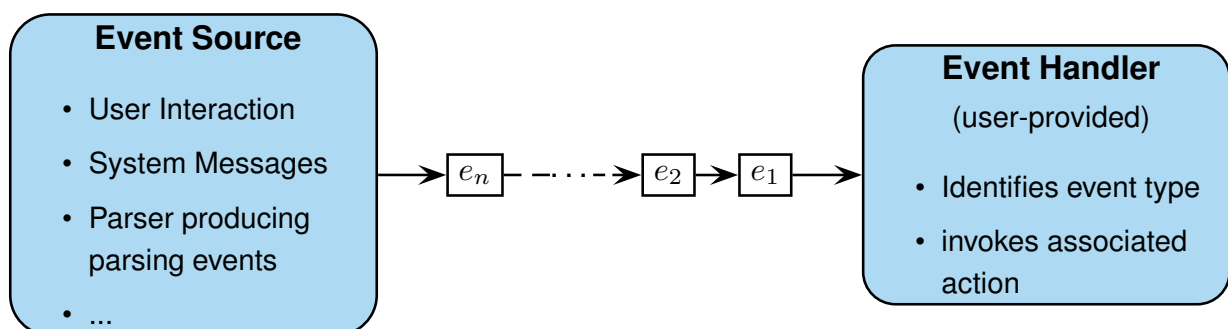
PARSING: GENERAL CONCEPTS

- **Compiler Construction in general:**
 1. input: a sequence of characters
 2. lexical analysis ("lexer") extracts the keywords (e.g. "begin", "end", "for") and outputs a sequence of *tokens*
 3. syntactical (grammar) analysis: check grammatical structure and generate the *parse tree* (e.g. via automaton)
 4. tools: lex & yacc/bison
 5. interpreter, optimizer, compiler, visualizer etc. process the parse tree
 - **XML:**
 1. lexical: split unicode input sequence into opening tags, closing tags, attributes, PCDATA, processing instructions, etc.
 2. syntactical and structural: is it well-formed?
 3. processing: build DOM tree, build JAXB structure, visualize, ...
 - the above DOM and JAXB are actually parser+specific processing
- ⇒ **XML Stream Processing: works on the tokens sequence!**

483

EVENT-BASED PROCESSING AS A *general Design Pattern*

- A stream of (high-level) items that carry some inherent semantics can be seen as a stream of "events"
(in contrast to a simple 0-1-stream, a byte stream or similar low-level streams)

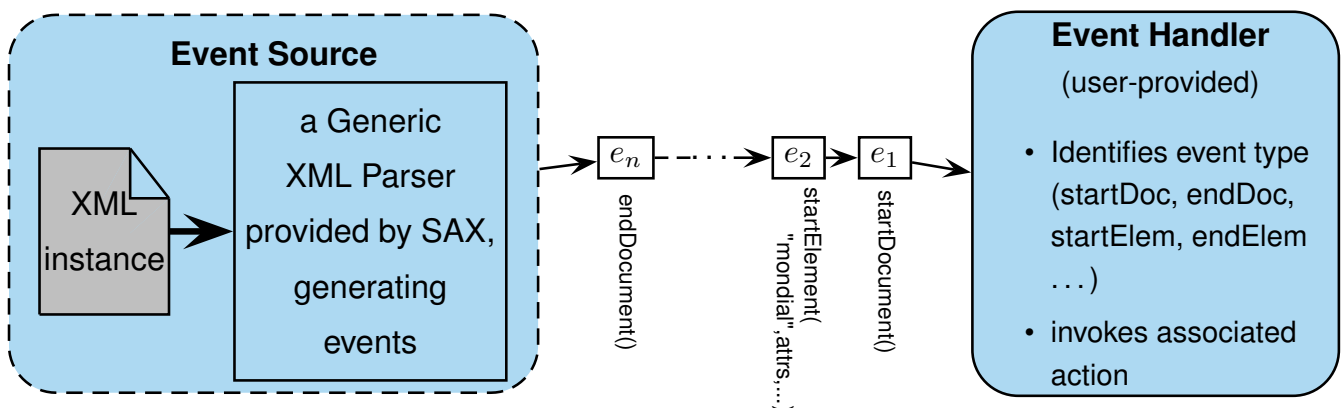


- The application programmer provides the Event Handler implementation, containing actions for each type of event;
- kind of *rule-based*;
- programmer is *not* in charge of the control flow

484

11.4 Event-based XML Parsing with SAX

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



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)

485

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”).

486

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 177) 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.

487

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'
...
```

488

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]

489

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]

490

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.

491

11.5 XML Streams/StAX - The Streaming API for XML

Higher abstraction level (than character-based XML) for XML data exchange:

`javax.xml.stream (rt.jar)`

Reconsider SAX

- on-the fly processing, no in-memory representation for good performance
- idea of “XML Event Stream”: a char stream (File, HTTP) can be converted into an XML Event Stream by an XML parser; see example’s `main()` method.
- SAX does not make the XML Event Stream accessible, but only via the Event Handler.

Generalization and Abstraction: XML Streams

- XMLEvents: StartDocument, StartElement, Character, EndElement,
- XMLStreamWriter, XMLStreamReader,
- [XML Streams also can be connected *directly* as an *abstract* means to exchange XML](#)

492

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 Unicode 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
[RFID applications, time series of stock quotes, RSS feeds]

493

XML Streams: Application Scenarios

- READ: usage analogous to SAX: process an XML file input as an 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.
- WRITE AND READ: streamed data exchange between processed on the XML level

494

Interfaces XMLStreamWriter, XMLStreamReader

(only some comments; see also following examples)

Reader

- `int event = r.next()` and then switch based on event type
javax.xml.stream.XMLStreamConstants.XX:
START_DOCUMENT, START_ELEMENT, ATTRIBUTE, CHARACTERS, END_ELEMENT, ...
- goal-driven access methods when on START_ELEMENT:
`r.getLocalName()`, `r.getAttributeValue(name)`,
`r.getAttributeCount()`, `getAttributeValue(n)` for iteration,
`r.getElementText()` (reads also the next EndElement from the stream!)
- goal-driven access method when on CHARACTERS: `r.getText()`

Writer

- Writer: `w.writeStartDocument()`, `w.writeStartElement(name)`,
`w.writeAttribute(name, value)`, `w.writeCharacters(text)`: obvious;
- `w.writeEndElement()`: closes the innermost open element;
- `w.writeEndDocument()`: closes all open elements.
- `w.flush()`: force write any data to the underlying output mechanism.

495

StAX EXAMPLE: EXAM REGISTRATION

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

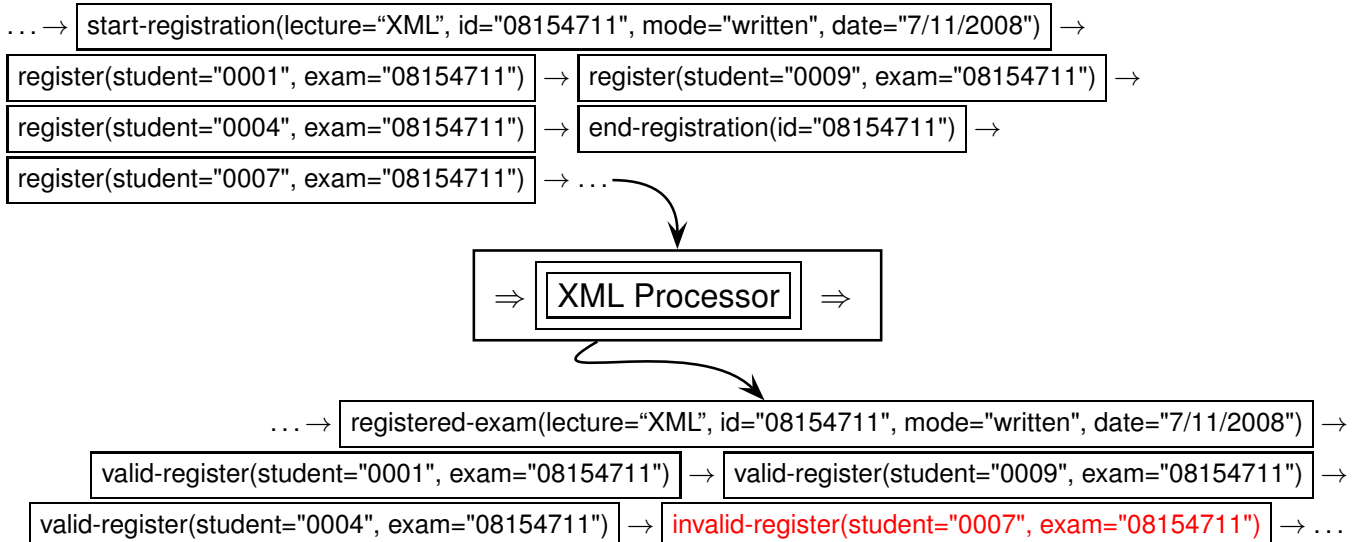
- The *subject* (e.g., "Semi-structured Data and XML") and ID of lectures/exams,
- whether the exam is *written* or *oral*,
- for written exams, the date of the exam,
- for oral exams, a number of dates is given when the single exams are held.
- the registration period *starts* when receiving an incoming 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`,

496

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:



497

StAX Example Cont'd:

Consider the following XML sequence as input stream:

498


```
<?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]

499

StAX Example Cont'd (2):

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

500

```

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();
    }
}

```

501

```
// continue next page
```

```

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]

502

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.io.FileOutputStream;
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;

            // continue next page

            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 subelements
                        if("start-registration".equals(parser.getLocalName())) {
                            currentExam = new Exam(parser.getAttributeValue(null, Exam.ID), parser.getAttributeValue(null, Exam.MOD), parser.getAttributeValue(null, Exam.DATE));
                            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;
                        }
                    }
                }
            }
            // continue next page
        } catch (XMLStreamException e) {
            e.printStackTrace();
        }
    }
}
```

```

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.STARTOFFREQ);
        writer.writeCharacters(currentExam.getStartOffreg());
        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();
    }
}
}
}

```

[Filename: java/StAX/ExamStreamParser.java]

505

StAX COMPARISON WITH SAX

SAX: • “Push” API

- Common pattern: methods for each event type, where `startElement()` and `endElement()` contain large `ifs`.

StAX: • “Pull” API

- Common pattern: huge `switch` command whose cases again contain large `if`.
- Performance: no difference (inside, StAX’s `next()` does the same as the SAX builtin stream reader algorithm that call the `EventHandler`)
- The actual code to be written is not much different in both cases.
- SAX maps a unicode input stream directly to the `EventHandler` calls.
- StAX makes the [intermediate abstraction level](#) of XML event streams accessible
 - can easily produce XML output via `XMLStreamWriter` (e.g. to another StAX appl.) (programmatically, the same feature can be provided by a SAX Event Handler as well using StAX’s `xml.stream` classes)

506

Example: XML Stream Communication

```
import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.io.OutputStream;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamWriter;

public class XMLStreamTestWriter implements Runnable
{
    OutputStream outputStream;

    public XMLStreamTestWriter(OutputStream out) {
        this.outputStream = out;
    }

    public void run() {
        try {
            XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
            XMLStreamWriter writer = outputFactory.createXMLStreamWriter(outputStream);
            writer.writeStartElement("foo");
            int i=1;
            while (i<100) {
                writer.writeStartElement("bla");
                writer.writeCharacters(" " + i);
                writer.writeEndElement();
                System.out.print("Write <bla> " + i + "</bla> ");
                //writer.flush(); // if not uncommented: strictly alternating
                // comment out flush: sleep < 700 causes alternating after blocks of 2...5 elements
                try{ java.lang.Thread.sleep(50); }
                catch (Exception e) { e.printStackTrace(); }
                i++;
            }
            // writer.writeEndElement(); // close </foo> is done by the next line:
            writer.writeEndDocument(); // docu: closes all tags, but does not send anything else
            writer.flush();
            writer.close();
        } catch (Exception e) { e.printStackTrace(); }
        System.out.println("Writer finished");
    }

    public static void main(String[] args) throws Exception{
        PipedOutputStream pos = new PipedOutputStream();
        PipedInputStream pis = new PipedInputStream();
        pis.connect(pos);
        new Thread (new XMLStreamTestWriter(pos)).start();
        new Thread (new XMLStreamTestReader(pis)).start();
    }
}
```

[Filename: java/StAX/XMLStreamTestWriter.java]

- underlying: connected PipedOutput/InputStream

507

Example: XML Stream Communication (Cont'd)

```
import java.io.PipedInputStream;
import java.io.InputStream;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamReader;

public class XMLStreamTestReader implements Runnable {
    InputStream inputStream;

    public XMLStreamTestReader(PipedInputStream in) {
        this.inputStream = in;
    }

    public void run() {
        try {
            XMLInputFactory inputFactory = XMLInputFactory.newInstance();
            XMLStreamReader parser = inputFactory.createXMLStreamReader(inputStream);
            boolean goOn = true;
            while (goOn) {
                int event = 0;
                try {
                    event = parser.next();
                    switch(event) {
                        case XMLStreamConstants.START_ELEMENT:
                            System.out.println("Read start element " + parser.getLocalName());
                            break;
                        case XMLStreamConstants.CHARACTERS:
                            System.out.println("Read " + parser.getText());
                            break;
                        case XMLStreamConstants.END_ELEMENT:
                            System.out.println("Read end element " + parser.getLocalName());
                            break;
                        case XMLStreamConstants.END_DOCUMENT: // never happens!
                            System.out.println("Read end document");
                            goOn = false;
                        default: System.out.println("Read something else. event: " + event);
                    }
                } catch(Exception e) { parser.close(); goOn = false; }
                parser.close();
                System.out.println("Reader finished");
            }
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

[Filename: java/StAX/XMLStreamTestReader.java]

508

11.6 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/>

509

XML DIGESTER: PRINCIPLE

- rule-based:
- on (restricted) XPath patterns ... do something.
- internally based on SAX,
- rules hook on beginElement and endElement,
- support for tailorable object generation, setting of properties and method calls,
- provides a stack with automatic and user-defined behavior.

510

XML DIGESTER: STACK

Stack

Default supported behavior mirrors XML hierarchy:

- top element is always the one that is currently processed,
- ancestors are down the stack.
- generation of objects on the stack is controlled by rules:
- `ObjectCreate(pattern, class.class)`
 - if an element satisfying *pattern* is opened, 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).

511

Stack: Additional Methods

- `push()`, `pop()`,
- `peek(n)` accesses the *n*-th element on the stack (top=0),
- 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. apply `peek(1).method(peek(0))`)

512

Actions/Rules

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

- `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.

513

XML Digester: Example

[Filename: java/Digester/GetMillionCities.java]

```
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;
        String country = null;
        int population = -1;
        public void setName(String name) { this.name = name; }
        public void setCountry(String code) { this.country = code; }
        // note: all PCDATA/CDATA values are strings!
        public void makePopulation(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
```

514


```

public static void main(String[] args) {
    File res = 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 and country/*/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", "makePopulation", 1);
    digester.addCallParam("mondial/country/city/population", 0);
    // at </city> calls addCity of the now-top-of-stack-object
    digester.addSetNext("mondial/country/city", "addCity");
    try {
        digester.setValidating(false);
        final CityCollection cities = digester.parse(res);
        System.out.println("#### now listing cities #### ");
        listCities(cities);
    } catch (Exception e) { e.printStackTrace(); }
}
}

```

515

RULES

The above are shortcuts for typical rule patterns.

Rule implementations are based on three optional 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 with opening tags (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 from stack.

516

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;
        String country = null;
        int population = -1;
        public void setName(String name) { this.name = name; }
        public void setCountry(String code) { this.country = code; }
        // note: all PCDATA/CDATA values are strings!
        public void makePopulation(String pop) { this.population = new Integer(pop); }
    }

    // continue next page                    517
    // continue next page
```

```
public static void main(String[] args) {
    File res = 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", "makePopulation", 1);
    digester.addCallParam("*/city/population", 0);
```

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

```

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

try {
    digester.setValidating(false);
    final CityCollection cities = digester.parse(res);
    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 applied in the correct order.

519

Digester: Producing 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;

public class PrintMillionCities {

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

    // continue next page

```

```

public static void main(String[] args) {
    File res = new File("mondial.xml");

```

520

```

}
// continue next page

public static void main(String[] args) {
    File res = 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", "makePopulation", 1);
    digester.addCallParam("*/city/population", 0);

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

```

521

11.7 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-ddThh:mm:ss[.xx][{+|-}hh:mm]*
 - xs:time: *hh:mm:ss[{+|-}hh:mm]*
 - xs:duration: *P[nY][nM][nD][T[nH][nM][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 282.
- map to appropriate classes for processing by Java (and by this e.g. with JDBC from and to SQL databases).

522

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.

523

11.8 Web Services (Overview)

- History: RPC (Remote Procedure Call)
 - call a specific procedure at a specific server
(client stub → marshalling → message → unmarshalling → server stub → server).
- History: OMG Standard (Object Management Group) CORBA (1989, “Common Object Request Broker Architecture”; cf. Slides 38 ff.):
 - Middleware, usually applied in an Intranet,
 - central ORB bus where services can connect,
 - service registry (predecessor of WSDL and UDDI ideas)
 - description of service interfaces in object-oriented style
(IDL - interface description language, similar to C++ declarations)
 - exchanging objects between services via OIF (Object Interchange Format)⇒ RPC abstraction (call abstract functionality) by the ORB as a broker.
- XML-RPC and SOAP+WSDL+UDDI are XML-based variants of RPC+Corba.
- SOA (“Service-Oriented Architecture”).

524

HTTP: HYPERTEXT TRANSFER PROTOCOL (OVERVIEW)

- HTTP 0.9 (1991), HTTP 1.0 (1995), HTTP 1.1 (1996).
- Application Layer Protocol, based on a (reliable) transport protocol (usually TCP “Transmission Control Protocol” that belongs to the “Internet Protocol Suite” (IP)) [see Telematics lecture].
- Request-Response Protocol: open connection, send something, receive response (upon completion), close connection
- usually associated with Web Browsing and HTML:
send (HTTP GET) URL, get URL (=resource) contents
⇒ this is already a (very basic) Web Service
also: send HTTP POST URL+Data (Web Forms) get answer
⇒ this is also a (still basic) Web Service; “Hidden Web”
- common protocol used for communication with and between Web Services ...

525

(JAVA) SERVLET

- a piece of software that should be made available as a Web Service
- implements the methods of the Servlet interface
(Java: `javax.servlet.http.HttpServlet`, subclasses `GenericServlet`, `HttpServlet`)

WEB (SERVICE|SERVLET) CONTAINER

- a piece of software that extends a Web Server with infrastructure to provide the runtime environment to run servlets as Web Services,
- hosts one or more Web Services that extend the container’s base URL,
- the servlets’ code must be accessible to the Web Service Container, usually located in a specific directory,
- controls the lifecycle of the servlets: (`init()`, `destroy()`)
- maps the incoming communication from ports via the URLs to the appropriate servlet invocation
Method `service(httpContents)`, mapped to `doGet(httpContents)`,
`doPut(httpContents)`.

526

ABSTRACTION LEVELS

- a Web Services Container contains several “projects” (eclipse terminology) or “applications”:
 - from the programmer’s view, a “project” is an (e.g., eclipse) project, as a package it is a single .war file, at the end, it is a subdirectory in the container.
Each project has an (internal) name (its directory name in the container), e.g. `project1` or `mondial`.
- Each project consists of one or more servlets:
 - each servlet has an (internal) name (relative to its directory name in the container), e.g. `project1` contains servlets `servlet1a` and `servlet1b`.
 - each servlet’s code is a class that extends `javax.HttpServlet`;

527

ABSTRACTION LEVELS: URL MAPPING

HTTP connections (GET, POST) received by the server are transparently forwarded to the servlets.

URLs are e.g.

`http://www.example.org/services/2011/oneservice/handle1`

`http://www.example.org/mondial/sqlonline`

- the Web Service Container has a base url;
`http://www.example.org`.
- the Web Service Container maps *relative paths* to projects (later: by tomcat’s `server.xml`), e.g.
`/services/2011/oneservice` to `project1`,
`/mondial` to `mondial`.
- each project’s configuration (later: `web.xml`) maps path tails to servlet ids, and servlet ids to servlet classes, e.g.
`/handle1` and `/` to `servlet1a` to `Project1.MyServletA`,
`/handle2` to `servlet1b` to `Project1.MyServletB`.

528

TOMCAT BASIC INSTALLATION

- Web Server with Web Service Container: Download and install Apache Tomcat
 - can *optionally, but not necessarily* be combined with the Apache Webserver
 - can be installed in the CIP Pool
- set environment variable (catalina is tomcat's Web Service Container)

```
export CATALINA_HOME=~/apache-tomcat-x.x.x
```
- configure server: edit

```
$CATALINA_HOME/conf/server.xml:  
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->  
<Connector port="8080" .../>
```
- start/stop tomcat:

```
$CATALINA_HOME/bin/startup.sh  
$CATALINA_HOME/bin/shutdown.sh
```
- logging goes to

```
$CATALINA_HOME/logs/catalina.out
```

529

GENERAL SERVLET PREPARATION DIRECTORY STRUCTURE

MyProject project directory (anywhere outside tomcat)

MyProject/build.xml the ant file for compiling and deploying – see later.

MyProject/src the .java (and other) sources

MyProject/lib jars that are needed for building, but should not be copied to the Web Service Container.

Put javax.servlet.jar there (tomcat has own classes for servlets, this would create conflicts).

MyProject/WebRoot roughly, all this content is copied later to the Web Service Container.

MyProject/WebRoot/WEB-INF

the whole content of MyProject/WebRoot except WEB-INF is visible later (e.g., HTML pages can be placed here); the contents of WEB-INF is used by the Web Service Container.

MyProject/WebRoot/WEB-INF/web.xml web application configuration – see later.

MyProject/WebRoot/WEB-INF/classes compiled binary stuff,

MyProject/WebRoot/WEB-INF/lib used jars.

530

HTTP METHODS GET AND POST

HTTP GET and POST: request-response paradigm

HTTP GET should be used only if invocation does *not* change

- Request consists only of URL+parameters:

`http://www.example.org/mondial?type=city&code=D&name=Berlin&province=Berlin`

HTTP POST should be used if it has side effects or changes the state of the Web Service

- Request URL consists only of the plain URL,
- parameters (e.g. queries using forms) or any other information is sent via a stream

⇒ often also queries use POST

Response: always as a stream.

- other HTTP methods PUT (resource), DELETE (resource) are used in REST (Representational State Transfer) “architectures” (e.g. the eXist XML database and document management system uses REST)

531

SERVLET PROGRAMMING

- servlets handle *incoming* HTTP connections via `doGet()` (=react-on-get) and `doPost()` (=react-on-post) methods:

```
public class MyServletA extends HttpServlet
{ public void init(ServletConfig cfg) throws ServletException
  { ... initialization ... }
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException
  { ... }
  protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException
  { ... }
}
```

- `doGet()` and `doPost()` both read the `HttpServletRequest` and write the `HttpServletResponse` object.
- the `HttpServletRequest` is different between GET (simpler) and POST (including a stream).

532

SERVLET PROGRAMMING - SOME METHODS

doGet/doPost(HttpServletRequest req, HttpServletResponse resp) throws ...

```
{ String path = req.getPathInfo();
```

tail of the URL path, e.g. do

```
if (path.equals("/handle1")) { ... }
```

```
java.util.Map<java.lang.String,java.lang.String[]> req.getParameterMap();
```

(doGet) returns a java.util.Map of the parameters of this request.

```
ServletInputStream req.getInputStream(); or
```

```
java.io.BufferedReader req.getReader();
```

(doPost) retrieves the body of the request as binary data using a ServletInputStream.

(doPost) retrieves the body of the request as character data using a BufferedReader.

```
PrintWriter out = resp.getWriter();
```

yields a Writer to the response – send character text.

```
ServletOutputStream os = resp.getOutputStream();
```

yields an output stream. Don't forget os.flush().

533

INVOKING A NEW HTTP CONNECTION

Strongly stripped fragment (without try-catch etc.):

```
public static StringBuffer connectAndSend(String url, StringBuffer content) {  
    HttpURLConnection.setFollowRedirects(true);  
    HttpURLConnection con = (HttpURLConnection) new URL(url).openConnection();  
    con.setRequestMethod("POST");  
    con.setDoInput(true);  
    con.setDoOutput(true);  
    con.setRequestProperty("Connection", "keep-alive");  
    con.setConnectTimeout(timeout);  
    con.setRequestProperty("Content-type", "text/xml");  
    write(con.getOutputStream(), content);  
    con.getOutputStream().close();  
    con.connect();  
    StringBuffer response = read(con.getInputStream());  
    con.getInputStream().close();  
    return response;    }  
}
```

534

A NOTE ON MULTITHREADING

- servlets can be instantiated by the container permanently or on-demand.
- if multiple requests for the same servlet come in, the servlet container can run multiple threads on the same instance of a servlet.
 - be careful with instance variables,
 - implement mutual exclusion if necessary
- the server can also create (and remove) additional instances of a servlet.

535

THE PROJECT'S WEB.XML

```
<web-app>
  <!-- Define servlet names and associate them with classfiles -->

  <servlet>
    <servlet-name>servlet1a</servlet-name>
    <servlet-class>org.dbis.project1.MyServletA</servlet-class>
    <init-param> ... </init-param>
  </servlet>
  <servlet> ... </servlet>

  <!-- define mapping of path tails to servlets -->

  <servlet-mapping>
    <servlet-name>servlet1a</servlet-name>
    <url-pattern>/handle1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>servlet1a</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>servlet1b</servlet-name>
    <url-pattern>/handle2</url-pattern>
  </servlet-mapping>
</web-app>
```

536

SERVLET DEPLOYMENT

- upon startup, tomcat deploys all servlets that are available in
 `$CATALINA_HOME/webapps`
 (considering path mappings etc. in `$CATALINA_HOME/server.xml`)

Two alternatives how to make servlets available there:

- create a `project1.war` file (web archive, similar to jar) and copy it into
 `$CATALINA_HOME/webapps`.
 (e.g. build.xml targets "dist" and "deploy")
 (tomcat will unpack and deploy it upon startup)
- create a directory `project1`, copy everything that is in the `WebRoot` directory there.
 (e.g. build.xml target "deploy")

537

TOMCAT'S CONF/SERVER.XML

The URL paths to the projects have to be defined.

This is done in the `<Host>` element:

```
<Host>
  <Context path="/services/2011/oneservice" reloadable="false" docBase="project1"/>
  :
</Host>
```

(since the standard path `mondial` for `mondial` is used, no explicit entry is needed)

- `reloadable`: automatically reloads the servlet if the code is changed (e.g. a new `.war`).
 Should be done only during development.
- the `path` attribute is key. There can be multiple paths that are mapped to the same `docBase`.

538

SOME COMMENTS

- HTTP connections are Unicode.
- exchanging XML via HTTP must be programmed via String/StringBuffer
 - out: serialize XML,
 - in: put SAX or StAX on the Unicode stream.
- exchanging StAX's XMLStream:
 - out: have an XMLStream, run StAX's next() iterator on it,
 - serialize each event, put it in the HTTP stream
 - in: create a naked StAX XMLStreamReader on the HTTP stream that just provides the XMLStream.