

# Homework #1

## Programming Web Services

### ID2208

# XML Processing

*Mihhail Matskin and  
Hooman Peiro Sajjad ([shps@kth.se](mailto:shps@kth.se))  
KTH – ICT School  
VT 2015*

# Administrative Issues

- We appreciate 2 members per group in all activities:
  - Home works
  - Project
- More than 2 members per group are not appreciated
  - Exceptions are considered case-wise

# Course Mailing Lists

- Subscribe to student mailing list of the course:
  - [https://mailman.ict.kth.se/mailman/listinfo/id2208\\_students](https://mailman.ict.kth.se/mailman/listinfo/id2208_students)
- If you faced any technical problem during homework, please send it first to “students” mailing list:  
[id2208\\_students@mailman.ict.kth.se](mailto:id2208_students@mailman.ict.kth.se)  
if your classmates don't answer then send question to the teachers
- We continuously monitor the students mailing list, and get involved if necessary.

# Homeworks

- **3 Homework**
- In-time submission and approval of all Homework, gives you 5 Bonus points
- You can get: (maximum) 5 Bonus Points

# Project

- 1 Project
- In-time submission and approval of Project, gives you 5 Bonus points
- You can get: (maximum) 5 Bonus Points

# Total Bonus Points

- In Total you can get:
- **10 Bonus Points from Project + Homeworks**
  - **5 homework bonus points**
  - **5 project bonus points**

# Homework schedule

Start Date	Due Date	Deliverable
2015-01-26	2015-02-02	Homework 1
2015-02-02	2015-02-09	Homework 2
2015-02-09	2015-02-16	Homework 3
2015-02-16	2015-03-02	Project

# Homework #1

## **XML** **Processing**



# Homework1

- XML Processing
- Aim: Understanding and getting a hands-on experience with XML processing and transformation technologies

# XML Processing

XML processing typically includes three phases:

## 1.Processing input XML

- Validating and Parsing XML documents (DOM, SAX)
- Querying and extracting information (XQuery)
- Associating the XML information to objects (JAXB)

## 2.Business Logic Process

- Processing information according to your business logic

## 3.Processing Output XML

- Building XML document model and directly serializing to XML
- Applying XSLT

# Document Object Model (DOM)

# DOM

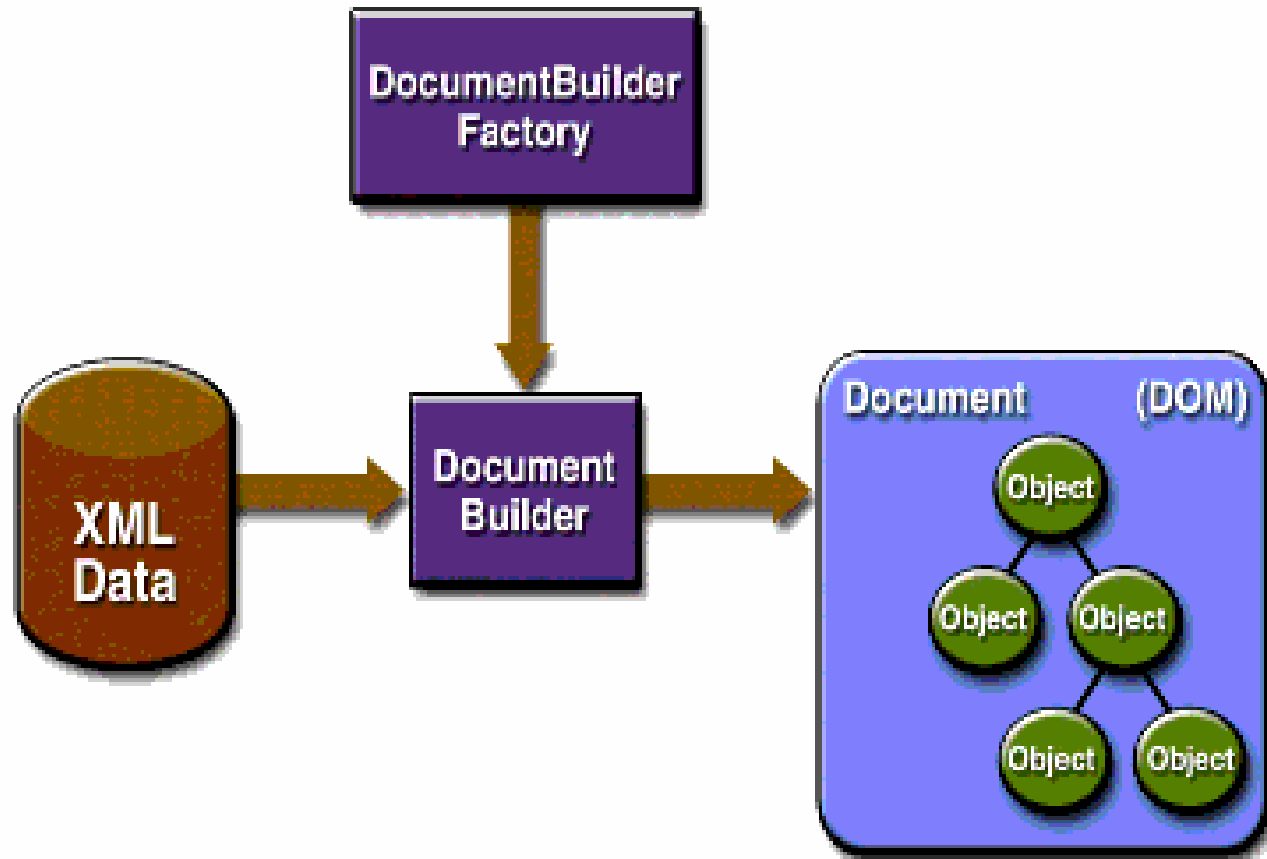
- Standard tree structure,
- Each node contains one of the components from an XML structure.
- Two common types of nodes: **element nodes** , **text nodes**

<sentence>*This is an* <bold> *important* </bold> *idea.*</sentence>



```
ELEMENT: sentence
+ TEXT: This is an
+ ELEMENT: bold
  + TEXT: important
+ TEXT: idea.
```

# DOM – Processing Model



# DOM (1)

//Get a factory object for DocumentBuilder objects

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
```

// to make the parser a validating parse

```
factory.setValidating(true);
```

//To parse a XML document with a namespace,

```
factory.setNamespaceAware(true);
```

// to ignore whitespace between elements.

```
factory.setIgnoringElementContentWhitespace(true);
```

// specifies the schema language for validation

```
factory.setAttribute(
```

```
"http://java.sun.com/xml/jaxp/properties/schemaLanguage,"
```

```
http://www.w3.org/2001/XMLSchema)
```

//specifies the XML schema document to be used for validation.

```
factory.setAttribute( "http://java.sun.com/xml/jaxp/properties/  
schemaSource", " YourXSDName");
```

# DOM (2)

//Get a DocumentBuilder (parser) object

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

//Parse the XML input file to create a document object that represents the input XML file.

```
Document document = builder.parse(new File(XMLFileName));
```

//Process the DOM tree, beginning with the document node to produce the output.

// For example :

```
Node root = document.getFirstChild()
```

```
NodeList children = root.getChildNodes();
```

```
for (Node child = root.getFirstChild(); child != null;  
      child = child.getNextSibling()) {  
    processNode(child);  
}
```

// look at sample DOM processing program

```
<xsd:schema>
<xsd:element name="transcript">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="university" type="xsd:string"/>
      <xsd:element name="degree" type="xsd:string"/>
      <xsd:element name="year" type="xsd:int"/>
      <xsd:element name="courses">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="course" minOccurs="0"
              maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```



# SAX Parsing Model

# SAX

An event-driven, serial-access mechanism for accessing XML documents.

You provide the callback methods, and the parser invokes them as it reads the XML data.

It is unidirectional, you cannot "back up" to an earlier part of the document, or rearrange it, any more than you can back up a serial data stream or rearrange characters you have read from that stream.

# SAX (1)

// example of the default (non-validating) parser

```
SAXParserFactory factory = SAXParserFactory.newInstance();
```

```
SAXParser saxParser = factory.newSAXParser();
```

```
saxParser.parse(new File(XMLFileName), handler);
```

# SAX (2)

```
.....  
// Parse for Education Section  
  
saxp.parse("Input.xml", new YourParserHandler(...));
```

---

```
static class YourParserHandler extends DefaultHandler {  
.....
```

# SAX (3)

@Override

public void **startDocument()** throws SAXException

```
{  
    ....  
}
```

@Override

public void **endDocument()** throws SAXException

```
{  
    ....  
}
```

@Override

public void **characters**(char[] arg0, int arg1, int arg2) throws SAXException

```
{  
    .....  
}
```

# SAX (3)

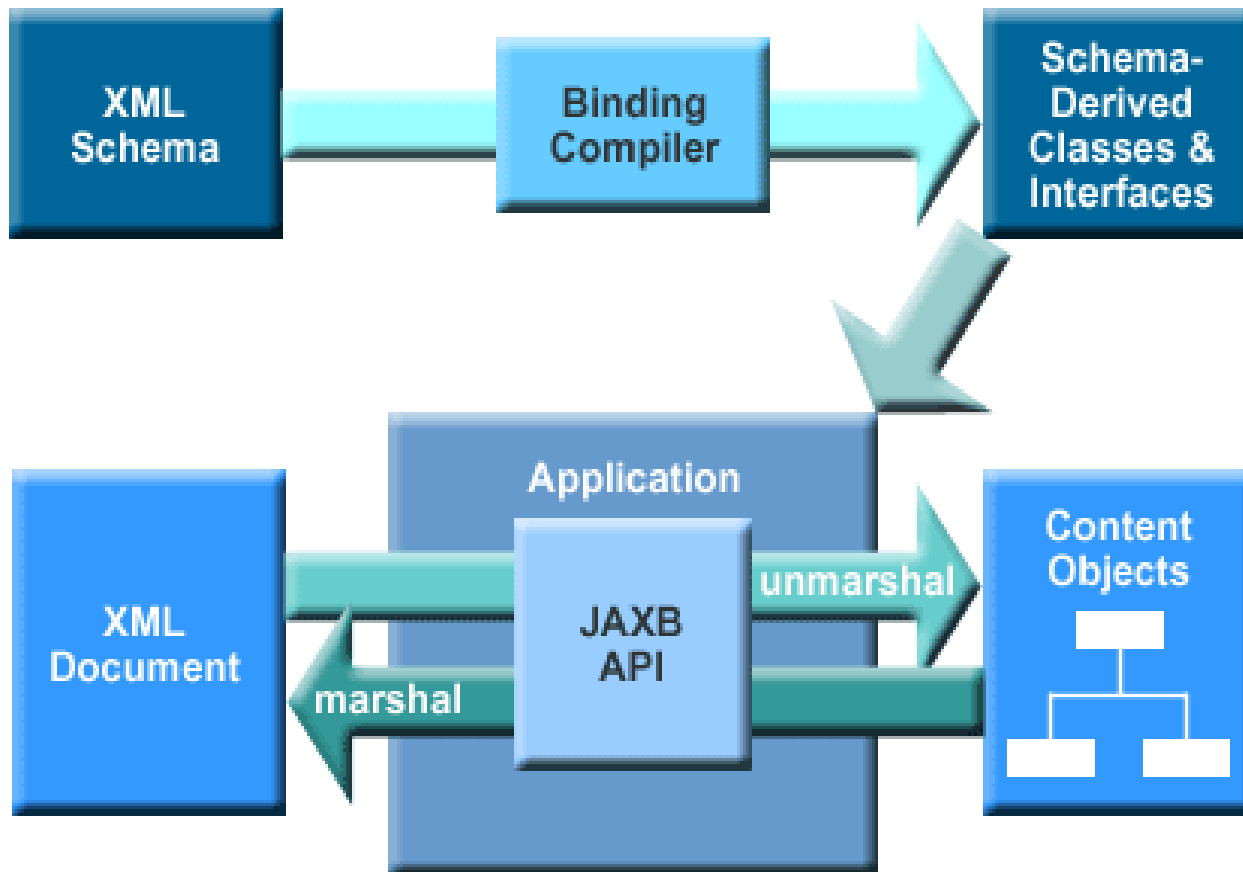
```
public void startElement(String namespaceURI, String
localName, // local name String qName, // qualified name
Attributes attrs) throws SAXException
{
    //Start of Element tag
}
```

```
public void endElement (String namespaceURI, String
localName, // local name String qualifiedName // qualified name)
throws SAXException
{
    //End of Element tag
}
```

# **JAVA API for XML Binding (JAXB)**

**<http://jaxb.java.net/>**

# JAXB (2) – XML Processing Model



JAXB XML processing model taken from :  
<https://netbeans.org/kb/74/websvc/jaxb.html>



# JAXB

## JAXB :

- API and tools that automate the mapping between XML documents and Java objects
- Part of *Oracle JWS DP* package

# Binding Schema to Java Class

You can compile the edited XML schema into Java classes using **XJC**.

You can find XJC at : **.../jwsdp/jaxb/bin/xjc**

Assume Purchase Order **Schema**

(<http://www.w3.org/TR/xmlschema-0/#po.xsd>)

```
> xjc -p primer.po -d src po.xsd
```

parsing a schema...

compiling a schema...

primer\po\impl\CommentImpl.java

primer\po\impl\ItemsImpl.java

primer\po\impl\JAXBVersion.java

primer\po\impl\PurchaseOrderImpl.java

primer\po\impl\USAddressImpl.java

.....

# Marshaling (Java Object to XML) - 1

*// create an ObjectFactory instance.*

*ObjectFactory objFactory = new ObjectFactory();*

*// create an empty PurchaseOrder*

*PurchaseOrder po = objFactory.createPurchaseOrder();*

*// manipulate "po" object*

*.....*

*// create a JAXBContext*

*JAXBContext jc = JAXBContext.newInstance( "primer.po" );*

*// create a Marshaller and marshal to System.out*

*Marshaller m = jc.createMarshaller();*

*m.setProperty( [Marshaller.JAXB\\_FORMATTED\\_OUTPUT](#),  
[Boolean.TRUE](#) );*

*m.[marshal](#)( po, System.out );*

# Java Object to XML (Unmarshaling)

*Try yourself:*

<http://www.oracle.com/technetwork/articles/javase/index-140168.html#unmars>

*A complete example of marshaling is here:*

[https://netbeans.org/kb/74/websvc/jaxb.html#Exercise\\_3](https://netbeans.org/kb/74/websvc/jaxb.html#Exercise_3)

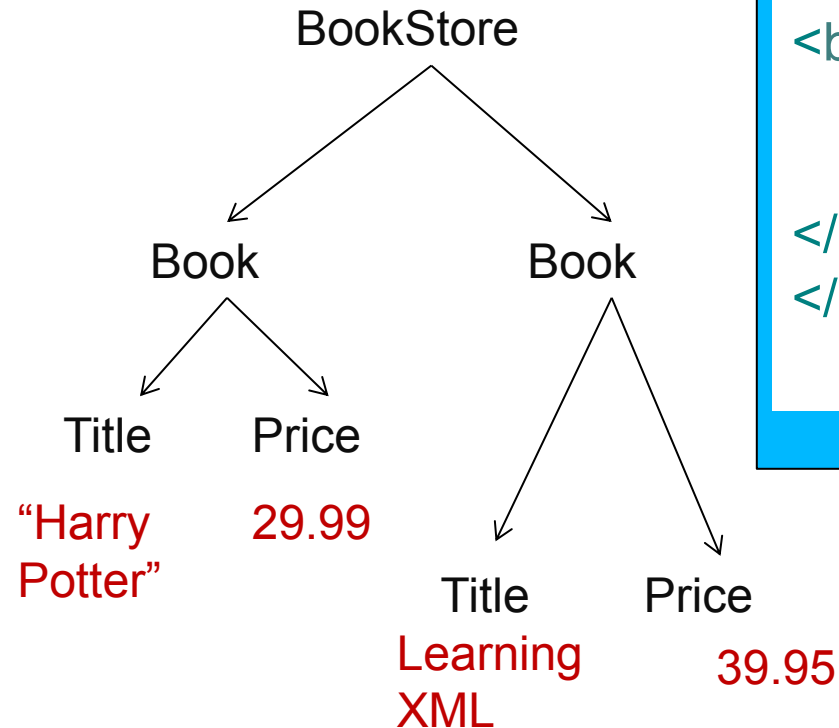
# XML Querying

# XML Querying

- 1-XPath (path expression + conditions) ✓
- 2-XSLT (XPath + transformation, output formatting) ✓
- 3-XQuery (XPath + more querying features)

# XPath (1)

Think of XML as a tree ,  
XPath = path + condition



```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="eng">Harry Potter</title>
    <price>29.99</price>
  </book>
  <book>
    <title lang="se">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>
```

# XPath (2)

XPath: Path expressions to select nodes in an XML document.

Expression	Description
<i>nodename</i>	Selects all child nodes of the named node
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

## Examples

In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
bookstore	Selects all the child nodes of the bookstore element
/bookstore	Selects the root element bookstore  <b>Note:</b> If the path starts with a slash ( / ) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang



# XPath (4)

XPath follows a hierarchical pattern to select elements.

Use **predicates ([ ])** to find a specific node or a node that contains a specific value. <http://www.w3schools.com/>

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element.  <b>Note:</b> IE5 and later has implemented that [0] should be the first node, but according to the W3C standard it should have been [1]!!
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='eng']	Selects all the title elements that have an attribute named lang with a value of 'eng'
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

# Some Sample XPATH

1- get All book titles

```
doc("YourBookstoreFile.xml")/Bookstore/Book/Title
```

2- Getting all titles (using *//*)

```
doc("BookstoreQ.xml") // Title
```

3- Reading all elements (using *\**)

```
doc(" YourBookstoreFile.xml ") // *
```

4- get All books with less than 40 (using *[condition]*)

```
doc(" YourBookstoreFile.xml ")/Bookstore/Book[Price < 40]
```

5- get the second book in the document ((using predicate *[ ]*)

```
doc(" YourBookstoreFile.xml ")/Bookstore/Book[2]
```

# XSLT

# XSLT

## Extensible Stylesheet Language Transformation (XSLT)

was originally aimed at developing an XML-based Stylesheet Language.

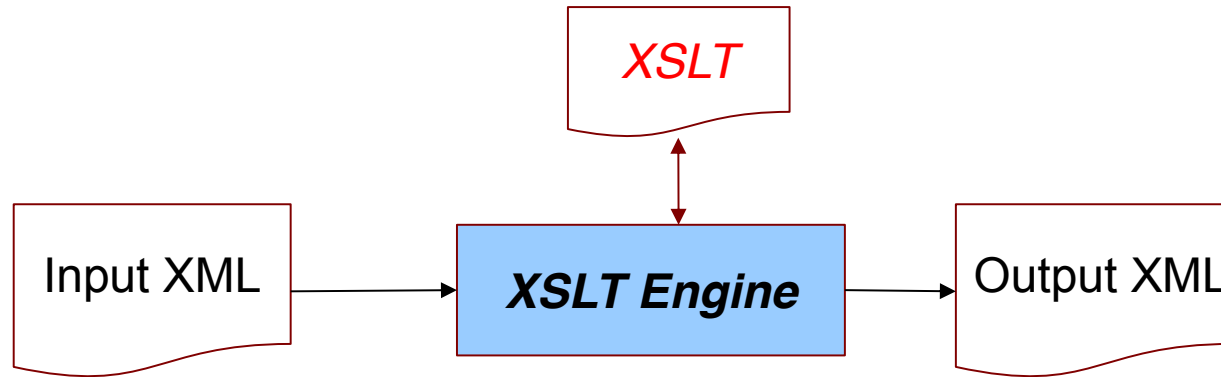
- XSLT: is a rule based transformation
- Idea is to write a template and replace

In this homework, We use **XSLT** to specify conversion from one XML document format to another XML document format.

XSLT has a larger agenda and scope than the course labs, you can find more examples at:

<http://www.w3schools.com/xsl/default.asp>

# Applying XSLT for XML Transformation



# XSLT Basics

A **Rule** based language.

A Rule (template rule) consists of:

1-A “**matching pattern**”, to match against XML elements specified using **XPath** expressions. Example:

```
<xsl:template match="XPath Expression">
```

2. A “**template**” which defines format of output document, whenever an XML element fits to the matching pattern. Example:

```
<xsl:element name="....">  
</xsl:element>
```

OR...

```
<xsl:value-of select="...."/>
```

OR....

# Input XML

```
<?xml version="1.0" encoding="UTF-8"?>
<priceList>
  <coffee>
    <name> Santos </name>
    <price>11.95</price>
    <producer>Brazil</producer>
  </coffee>
  <coffee>
    <name> Colombia </name>
    <price>12.50</price>
    <producer>JuanValdez</producer>
  </coffee>
</priceList>
```

# Target XML

```
<Coffee >  
  <CoffeeName> coffe name<CoffeeName>  
  <CoffeeProducer> coffee produce <CoffeeProducer>  
</Coffee>
```



# Designed XSLT (no loop)

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0" xmlns:ns="http://www.coffee.com" >
```

```
  <xsl:template match="/">
```

```
    <xsl:element name="ns:Coffee">
```

```
      <xsl:element name="ns:CoffeeName">
```

```
        <xsl:value-of select="/priceList/coffee/name"/>
```

```
      </xsl:element>
```

```
      <xsl:element name="ns:CoffeeProducer">
```

```
        <xsl:value-of select="/priceList/coffee/producer"/>
```

```
      </xsl:element>
```

```
    </xsl:element>
```

```
  </xsl:template>
```

```
</xsl:stylesheet>
```

# Output

```
<?xml version="1.0" encoding="UTF-8"?>  
<ns:Coffee xmlns:ns="http://www.coffee.com">  
  <ns:CoffeeName> Santos_</ns:CoffeeName>  
  <ns:CoffeeProducer> Brazil </ns:CoffeeProducer>  
</ns:Coffee>
```

# Designed XSLT (Withloop)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0" xmlns:ns="http://www.coffee.com" >
  <xsl:template match="/">
    <xsl:element name="ns:Coffee">
      <xsl:for-each select="/priceList/coffee">
        <xsl:element name="ns:CoffeeName">
          <xsl:value-of select="name"/>
        </xsl:element>
        <xsl:element name="ns:CoffeeProducer">
          <xsl:value-of select="producer"/>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

# Output

```
<?xml version="1.0" encoding="UTF-8"?>  
<ns:Coffee xmlns:ns="http://www.coffee.com">  
  <ns:CoffeeName> Santos_</ns:CoffeeName>  
  <ns:CoffeeProducer> Brazil </ns:CoffeeProducer>  
  <ns:CoffeeName> Colombia</ns:CoffeeName>  
  <ns:CoffeeProducer> Juan Valdez</ns:CoffeeProducer>  
</ns:Coffee>
```

# Homeworks!

# Problem Description -1

We would like to simulate an “Employment Service Company” like *Manpower*, *AcademicWork*, *Komet* ...

The main task of such companies is to create a profiles of job seekers and match them against the advertised jobs, provided by different companies.



# Problem Description -2

A profile of job seeker is made of :

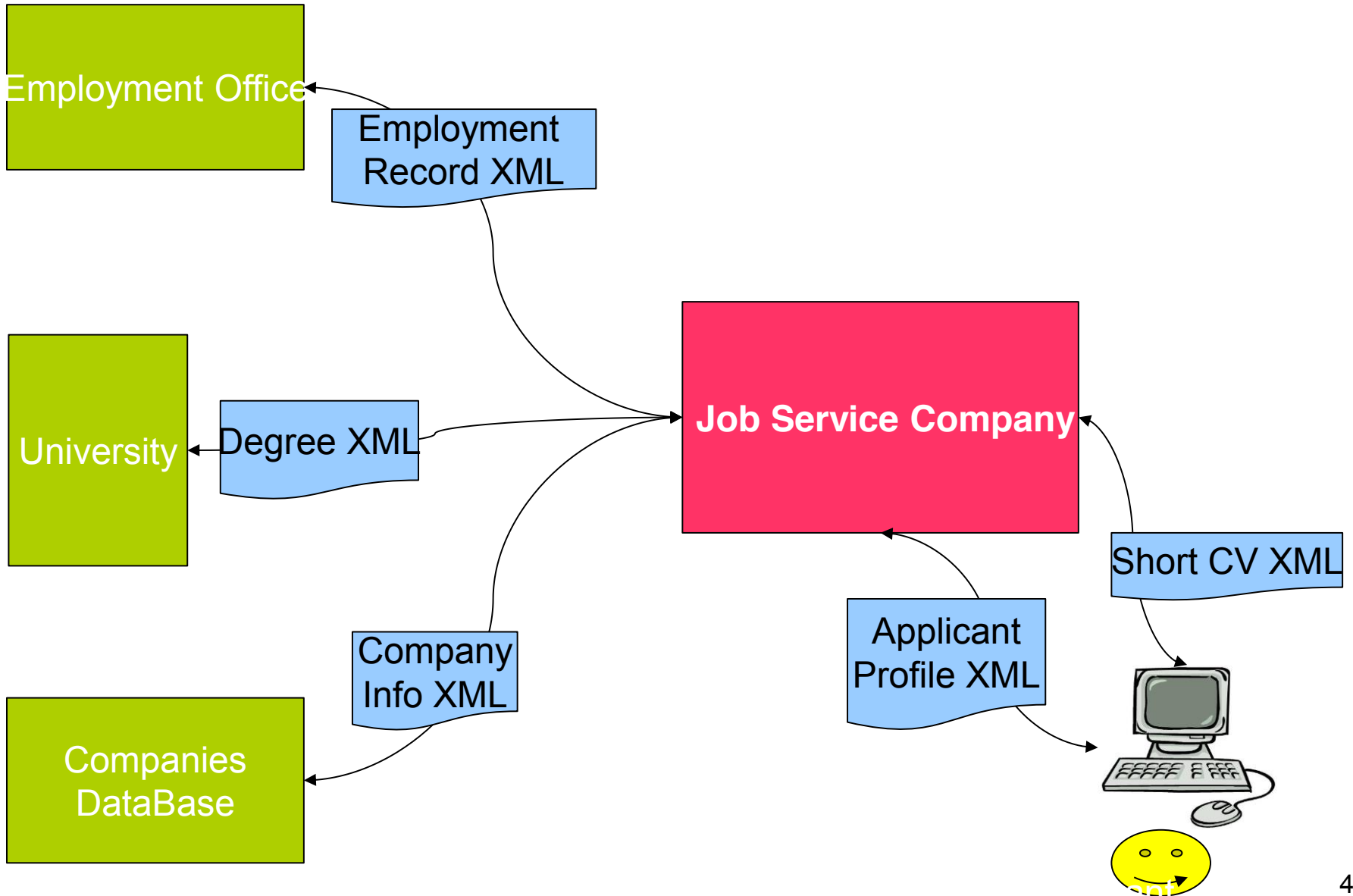
- CV,
- relevant academic degree(s)
- previous working experiences,
- information about companies the applicant worked for before,
- motivation letter,
- places desired to work,
- type of job (permanent, part time, contract,...) ,
- references and other relevant qualifications (e.g. driving license).
- .....

# Problem Description -3

- The idea is to design a automated way to collect the required information/documents from different sources to create applicant profile automatically.
  - All documents are provided in XML format.
- The content of the profile can be obtained from the following sources:
  - Degree and Transcript issued by “University”
  - Employment Records from “Employment Office”
  - Information of Companies from an Online Service (database of Companies)
  - Short CV and other materials provided by the applicant while registering in “Employment Service Company”



# Interactions



# Tasks -1

- ◆ Create appropriate schema (XSD) for each XML document (**Transcript, Employment Record, Company Info, short CV and Applicant Profile**).
- ◆ Generate sample documents (XMLs) out of those schema and populate the content (assume that applicant have **at least** one previous working experience, one academic degree, ....) and **Validate** them against your schema.
- ◆ Use ***Namespace*** in your Schemas  
Obligatoriskt!

# Tasks -2

- Write programs to map the relevant piece of information from collected documents into Applicant Profile through **ALL FOUR** different mechanisms (use each of the following methods for XML processing at least once).
  - Document Object Model (DOM)
  - Simple API for XML (SAX)
  - Extensible Stylesheet Language Transformations (XSLT)
  - JAXB

# Tasks -3

"ONLY use XSLT"

- As a part of program functionality, it should be able to calculate the GPA from Transcript and put it in appropriate place in User Profile, while mapping academic records to User Profile.

GPA = Grade Point Average, snittbetyg

- The output of the above mentioned programs will be the complete **User Profile in XML format.**

# XML Processing libraries in JWSDP

- Download and install Java Web Service Developer Pack ( JWSDP 2.0) from:  
<http://www.oracle.com/technetwork/java/javasebusiness/downloads/java-archive-downloads-jwsdp-419428.html>
- Unzip  
You can find required libraries for XML processing in `.../jwsdp/jaxb/lib` and `.../jwsdp/jaxp/lib` folder in the installed directories.

# Development Environments

You are free to use whatever IDE (NetBeans, Eclipse, .... ) you desire.

Recent versions of NetBeans includes an XSLT module.

You can find XSLT plug-ins for Eclipse at:

<http://eclipsexslt.sourceforge.net/> ,

[http://wiki.eclipse.org/XSLT\\_Project](http://wiki.eclipse.org/XSLT_Project)

# Deliverables

- Textual report explaining what you did
- The XSDs (5 xsd files)
- The 4 populated XML documents (Transcript, Employment Record,...)
- The source code of the “ mapping programs” including designed “xslt” file.
- The generated Applicant Profile.
- You **will** demonstrate your work in a presentation.

# In your work, try to use:

- Give suitable and human understandable names to XML tags.
- In the designed schema, we expect to see:
  - **Complex** and **Simple** types, **Attributes** and **Elements**.
  - Using **Restrictions** (at least THREE per schema) to narrow the ranges of values or formatting the values which an element could take.
  - Using **Extension** (if possible)



# HW #1 - Delivery

- Send your deliverables by e-mail to BOTH:

[misha@kth.se](mailto:misha@kth.se) and [shps@kth.se](mailto:shps@kth.se)

*e-mail subject: **PWS15-HW1***

*Please add your names in the body of the email*

*Attach: source code + instructions how to run your code,*

Deadline: **2 Feb 2015, 11:59 PM CET**

- Presentation: **will be announced on mailing-list**  
(Location to be decided)

**GOOD LUCK!**

# Useful Materials

- XSLT:

[http://www.globalguideline.com/xslt/XSLT\\_Introduction.php](http://www.globalguideline.com/xslt/XSLT_Introduction.php)

<http://www.w3schools.com/xsl/default.asp>

<http://www.learn-xslt-tutorial.com/>

<http://www.zvon.org/xxl/XSLTutorial/Output/contents.html#id2>

- SAX , DOM, JAXB:

<http://totheriver.com/learn/xml/xmltutorial.html>

<http://java.sun.com/webservices/docs/2.0/tutorial/doc/index.html>

- Sample code for examples throughout the tutorial is available in a zip file on course homepage.

```
<xsd:schema>
<xsd:element name="transcript">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="university" type="xsd:string"/>
      <xsd:element name="degree" type="xsd:string"/>
      <xsd:element name="year" type="xsd:int"/>
      <xsd:element name="courses">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="course" minOccurs="0"
              maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```