Introduction to XML

1

What is XML?

- introduced by W3C in 1998 (latest recommendation from 2004: http://www.w3.org/TR/REC-xml)
- stands for eXtensible Markup Language
- is more general than HTML, but simpler than SGML (which in turn was invented by Canadians, see Ottawa Citizen High Tech article)
  - Aside: Lynne Truss' "Eats, shoots, and leaves" explains why printers are focusing on markup languages
- is used to describe *metadata*
- you can define your own set of tags!
  - an XML document does not "do" anything on its own

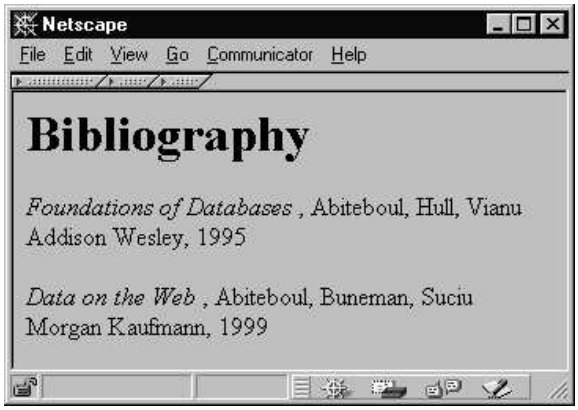
2

XML: Why Yet Another Markup Language?

- content is independent from rendering
- meta-data makes search easier
- standard tags enable data interchange across tools
- format for data and object persistence, human readable and editable
- no need for a custom parser anymore

3

From HTML to XML



4

HTML describes the presentation

HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
        Abiteboul, Hull, Vianu
        <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
        Abiteoul, Buneman, Suciu
        <br> Morgan Kaufmann, 1999
```

5

XML

```
<bibliography>
    <book>   <title> Foundations… </title>
             <author> Abiteboul </author>
             <author> Hull </author>
             <author> Vianu </author>
             <publisher> Addison Wesley </publisher>
             <year> 1995 </year>
    </book>
    …
</bibliography>
```

XML describes the content

6

XML Terminology

tags: book, title, author, …
start tag: <book>, end tag: </book>
elements:
<book>…</book>,<author>…</author>
elements are nested
empty element: <red></red> abbrv. <red/>
an XML document: single root element

well formed XML document: if it has matching tags

7

More XML: Attributes

```
<book price = “55” currency = “USD”>
  <title> Foundations of Databases </title>
  <author> Abiteboul </author>
   …
  <year> 1995 </year>
</book>
```

attributes are alternative ways to represent data

8

More XML: Oids and References

```
<person id="o555">  <name> Jane </name> </person>

<person id="o456">  <name> Mary </name>
                    <children idref="o123 o555"/>
</person>

<person id="o123" mother="o456"><name>John</name>
</person>
```

oids and references in XML are just syntax

9

Recipes in XML (will be used later)

- Define our own ***"Recipe Markup Language"***
- Choose markup tags that correspond to concepts in this application domain
  - *recipe*, *ingredient*, *amount*, ...
- No canonical choices
  - granularity of markup?
  - structuring?
  - elements or attributes?
  - ...

10

Example (1/2)

```
<collection>
  <description>Recipes suggested by Jane Dow</description>

  <recipe id="r117">
    <title>Rhubarb Cobbler</title>
    <date>Wed, 14 Jun 95</date>

    <ingredient name="diced rhubarb" amount="2.5" unit="cup"/>
    <ingredient name="sugar" amount="2" unit="tablespoon"/>
    <ingredient name="fairly ripe banana" amount="2"/>
    <ingredient name="cinnamon" amount="0.25" unit="teaspoon"/>
    <ingredient name="nutmeg" amount="1" unit="dash"/>

    <preparation>
      <step>
        Combine all and use as cobbler, pie, or crisp.
      </step>
    </preparation>
```

11

Example (2/2)

```
    <comment>
      Rhubarb Cobbler made with bananas as the main sweetener.
      It was delicious.
    </comment>

    <nutrition calories="170" fat="28%"
               carbohydrates="58%" protein="14%"/>
    <related ref="42">Garden Quiche is also yummy</related>
  </recipe>
</collection>
```

12

Building on the XML Notation

- Defining the **syntax** of our recipe language
  - DTD, XML Schema, ...
- Showing recipe documents in **browsers**
  - XPath, XSLT
- Recipe collections as **databases**
  - XQuery
- Building a **Web-based** recipe editor
  - HTTP, Servlets, JSP, ...
- ...

13

XML Concepts

- An XML document that follows the syntax rules is considered *well-formed*
- But there is no restriction on the nature, order and number of tags in a well-formed XML document!
  - in order to impose some restrictions, you need to define *validity* criteria in a separate document…

14

Validity: Motivation

- We have designed our Recipe Markup Language
- ...but so far only **informally** described its **syntax**
- *How can we make tools that check that an XML document is a **syntactically correct** Recipe Markup Language document (and thus meaningful)?*
- Implementing a specialized validation tool for Recipe Markup Language is *not* the solution...

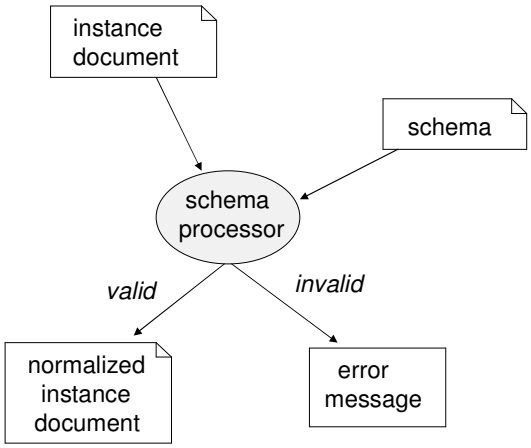
15

XML Languages

- ***XML language***:
  a set of XML documents with some semantics
- ***schema***:
  a formal definition of the syntax of an XML language
- ***schema language***:
  a notation for writing schemas

16

Validation



17

Why use Schemas?

- Formal but human-readable descriptions
- Data validation can be performed with existing schema processors

18

General Requirements

- Expressiveness
- Efficiency
- Comprehensibility

19

Regular Expressions

- Commonly used in schema languages to describe **sequences** of **characters** or **elements**
- Σ: an alphabet (typically Unicode characters or element names)
- $\sigma \in \Sigma$ matches the string σ
- α? matches zero or one α
- α* matches zero or more α's
- α+ matches one or more α's
- $\alpha\ \beta$ matches any concatenation of an α and a β
- $\alpha \mid \beta$ matches the union of α and β

20

Examples

- A regular expression describing **integers**:

```
0|-?(1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*
```

- A regular expression describing the valid contents of `table` elements in XHTML:

```
caption? ( col | colgroup )* thead? tfoot? ( tbody | tr )+
```

21

DTD – Document Type Definition

- Defined as a subset of the DTD formalism from SGML
- Specified as an integral part of XML 1.0
- A starting point for development of more expressive schema languages
- Considers elements, attributes, and character data – processing instructions and comments are mostly ignored

22

Document Type Declarations

- Associates a DTD schema with the instance document

```
<?xml version="1.1"?>
<!DOCTYPE collection SYSTEM
"http://www.brics.dk/ixwt/recipes.dtd">
<collection>
...
</collection>
```

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<!DOCTYPE collection [ ... ]>
```

23

Element Declarations

<!ELEMENT *element-name content-model* >

Content models:

- **EMPTY**
- **ANY**
- ***mixed content***: (#PCDATA | e_1 | e_2 | ... | e_n) *
- ***element content***: regular expression over element names
  (concatenation is written with "**,**")

Example:

```
<!ELEMENT table
        (caption?,(col|colgroup)*,thead?,tfoot?,(tbody|tr)+) >
```

24

Attribute-List Declarations

<!ATTLIST *element-name attribute-definitions* >

Each attribute definition consists of

- an attribute name
- an attribute *type*
- a *default declaration*

Example:

```
<!ATTLIST input maxlength CDATA #IMPLIED
               tabindex CDATA #IMPLIED>
```

25

Attribute Types

- CDATA: any value
- *enumeration*: $(s_1|s_2|...|s_n)$
- ID: must have unique value
- IDREF (/ IDREFS): must match some ID attribute(s)
- ...

Examples:

```
<!ATTLIST p align (left|center|right|justify) #IMPLIED>

<!ATTLIST recipe id ID #IMPLIED>
<!ATTLIST related ref IDREF #IMPLIED>
```

26

Attribute Default Declarations

- #REQUIRED
- #IMPLIED (= optional)
- "*value*" (= optional, but default provided)
- #FIXED "*value*" (= required, must have this value)

Examples:

```
<!ATTLIST form
      action CDATA #REQUIRED
      onsubmit CDATA #IMPLIED
      method (get|post) "get"
      enctype CDATA "application/x-www-form-urlencoded" >

<!ATTLIST html
      xmlns CDATA #FIXED "http://www.w3.org/1999/xhtml">
```

27

Checking Validity with DTD

A DTD processor (also called a *validating* XML parser)

- parses the input document (includes checking well-formedness)
- checks the root element name
- for each element, checks its contents and attributes
- checks uniqueness and referential constraints (ID/IDREF(S) attributes)

28

RecipeML with DTD (1/2)

```
<!ELEMENT collection (description,recipe*)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT recipe
  (title,date,ingredient*,preparation,comment?,
   nutrition,related*)>
<!ATTLIST recipe id ID #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT ingredient (ingredient*,preparation)?>
<!ATTLIST ingredient name CDATA #REQUIRED
                     amount CDATA #IMPLIED
                     unit CDATA #IMPLIED>
```

29

RecipeML with DTD (2/2)

```
<!ELEMENT preparation (step*)>
<!ELEMENT step (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT nutrition EMPTY>
<!ATTLIST nutrition calories CDATA #REQUIRED
                    carbohydrates CDATA #REQUIRED
                    fat CDATA #REQUIRED
                    protein CDATA #REQUIRED
                    alcohol CDATA #IMPLIED>
<!ELEMENT related EMPTY>
<!ATTLIST related ref IDREF #REQUIRED>
```

30

Problems with the DTD description

- `calories` should contain a non-negative number
- `protein` should contain a value on the form *N*`%` where *N* is between 0 and 100;
- `comment` should be allowed to appear anywhere in the contents of `recipe`
- `unit` should only be allowed in an elements where `amount` is also present
- nested `ingredient` elements should only be allowed when `amount` is absent

–DTD schema permits in some cases too much and in other cases too little!

31

Other Limitations of DTD

1. Cannot constraint **character data**
2. Specification of **attribute values** is too limited
3. The support for **modularity**, **reuse**, and **evolution** is too primitive
4. The normalization features lack **content defaults** and proper **whitespace** control
5. It does not itself use an **XML syntax**
6. No support for **namespaces**
7. .....

32

Requirements for XML Schema

- W3C's proposal for replacing DTD

Design principles:

- More expressive than DTD
- Use XML notation
- Self-describing
- Simplicity

Technical requirements:

- Namespace support
  - allow reuse of types and schemas
  - avoid naming clashes
- User-defined datatypes
- Inheritance (OO-like)
- ...

33

XML Schema

- a 2001 W3C recommendation
- allows the definition of elements and attributes using the XML syntax
- supports many primitive types
- allows the creation of complex types

34

XML Schemas

```
<xsd:element name="paper" type="papertype"/>
<xsd:complexType name="papertype">
    <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" minOccurs="0"/>
        <xsd:element name="year"/>
        <xsd: choice> < xsd:element name="journal"/>
                      <xsd:element name="conference"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:element>
```

DTD: <!ELEMENT paper (title,author*,year, (journal|conference))>

35

Elements v.s. Types in XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name"
                   type="xsd:string"/>
      <xsd:element name="address"
                   type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="person"
                   type="ttt">
<xsd:complexType name="ttt">
  <xsd:sequence>
    <xsd:element name="name"
                   type="xsd:string"/>
    <xsd:element name="address"
                   type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

DTD: <!ELEMENT person (name,address)>

36

XML Schema: Types and Structure

- Types:
  - Simple types (integers, strings, ...)
  - Complex types (regular expressions, like in DTDs)
- Element-type-element alternation:
  - Root element has a complex type
  - That type is a regular expression of elements
  - Those elements have their complex types...
  - ...
  - On the leaves we have simple types

37

Types and Declarations

- **Simple type definition**:
  defines a family of Unicode text strings
- **Complex type definition**:
  defines a content and attribute model
- **Element declaration**:
  associates an element name with a simple or complex type
- **Attribute declaration**:
  associates an attribute name with a simple type

38

Element and Attribute Declarations

Examples:

```
- <element name="serialnumber"
           type="nonNegativeInteger"/>

- <attribute name=”alcohol"
             type=”r:percentage"/>
```

39

Example (1/3)

Instance document:

```
<b:card
  xmlns:b="http://businesscard.org">
  <b:name>John Doe</b:name>
  <b:title>CEO, Widget Inc.</b:title>
  <b:email>john.doe@widget.com</b:email>
  <b:phone>(202) 555-1414</b:phone>
  <b:logo b:uri="widget.gif"/>
</b:card>
```

40

Example (2/3)

Schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://businesscard.org"
        targetNamespace="http://businesscard.org">

  <element name="card" type="b:card_type"/>
  <element name="name" type="string"/>
  <element name="title" type="string"/>
  <element name="email" type="string"/>
  <element name="phone" type="string"/>
  <element name="logo" type="b:logo_type"/>
  <attribute name="uri" type="anyURI"/>
```

41

Example (3/3)

```
  <complexType name="card_type">
    <sequence>
      <element ref="b:name"/>
      <element ref="b:title"/>
      <element ref="b:email"/>
      <element ref="b:phone" minOccurs="0"/>
      <element ref="b:logo" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="logo_type">
    <attribute ref="b:uri" use="required"/>
  </complexType>
</schema>
```

42

Connecting Schemas and Instances

```
<b:card xmlns:b="http://businesscard.org“
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance"
  xsi:schemaLocation="http://businesscard.org
                      business_card.xsd">
  <b:name>John Doe</b:name>
  <b:title>CEO, Widget Inc.</b:title>
  <b:email>john.doe@widget.com</b:email>
  <b:phone>(202) 555-1414</b:phone>
  <b:logo b:uri="widget.gif"/>
</b:card>
```

43

Simple Types (Datatypes) – Primitive

string	*any Unicode string*
boolean	true, false, 1, 0
decimal	3.1415
float	6.02214199E23
double	42E970
dateTime	2004-09-26T16:29:00-05:00
time	16:29:00-05:00
date	2004-09-26
hexBinary	48656c6c6f0a
base64Binary	SGVsbG8K
anyURI	http://www.brics.dk/ixwt/
QName	rcp:recipe, recipe
…	

44

Derivation of Simple Types – Restriction

Constraining facets:

- length
- minLength
- maxLength
- pattern
- enumeration
- whiteSpace
- maxInclusive
- maxExclusive
- minInclusive
- minExclusive
- totalDigits
- fractionDigits

45

Examples

```
<simpleType name="score_from_0_to_100">
  <restriction base="integer">
    <minInclusive value="0"/>
    <maxInclusive value="100"/>
  </restriction>
</simpleType>

<simpleType name="percentage">
  <restriction base="string">
    <pattern value="([0-9]|[1-9][0-9]|100)%"/>
  </restriction>
</simpleType>
```

regular expression

46

Simple Type Derivation – List

```
<simpleType name="integerList">
  <list itemType="integer"/>
</simpleType>
```

matches whitespace separated lists of `integers`

47

Simple Type Derivation – Union

```
<simpleType name="boolean_or_decimal">
  <union>
    <simpleType>
      <restriction base="boolean"/>
    </simpleType>
    <simpleType>
      <restriction base="decimal"/>
    </simpleType>
  </union>
</simpleType>
```

48

Built-In Derived Simple Types

- normalizedString
- token
- language
- Name
- NCName
- ID
- IDREF
- integer
- nonNegativeInteger
- unsignedLong
- long
- int
- short
- byte
- ...

49

RecipeML with XML Schema (1/5)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:r="http://www.brics.dk/ixwt/recipes"
        targetNamespace="http://www.brics.dk/ixwt/recipes"
        elementFormDefault="qualified">

  <element name="collection">
    <complexType>
      <sequence>
        <element name="description" type="string"/>
        <element ref="r:recipe" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
    <unique name="recipe-id-uniqueness">
      <selector xpath=".//r:recipe"/>
      <field xpath="@id"/>
    </unique>
    <keyref name="recipe-references" refer="r:recipe-id-uniqueness">
      <selector xpath=".//r:related"/>
      <field xpath="@ref"/>
    </keyref>
  </element>
```

50

RecipeML with XML Schema (2/5)

```
<element name="recipe">
  <complexType>
    <sequence>
      <element name="title" type="string"/>
      <element name="date" type="string"/>
      <element ref="r:ingredient" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="r:preparation"/>
      <element name="comment" type="string" minOccurs="0"/>
      <element ref="r:nutrition"/>
      <element ref="r:related" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="NMTOKEN"/>
  </complexType>
</element>
```

51

RecipeML with XML Schema (3/5)

```
<element name="ingredient">
  <complexType>
    <sequence minOccurs="0">
      <element ref="r:ingredient" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="r:preparation"/>
    </sequence>
    <attribute name="name" use="required"/>
    <attribute name="amount" use="optional">
      <simpleType>
        <union>
          <simpleType>
            <restriction base="r:nonNegativeDecimal"/>
          </simpleType>
          <simpleType>
            <restriction base="string">
              <enumeration value="*"/>
            </restriction>
          </simpleType>
        </union>
      </simpleType>
    </attribute>
    <attribute name="unit" use="optional"/>
  </complexType>
</element>
```

52

RecipeML with XML Schema (4/5)

```
<element name="preparation">
  <complexType>
    <sequence>
      <element name="step" type="string" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<element name="nutrition">
  <complexType>
    <attribute name="calories" type="r:nonNegativeDecimal" use="required"/>
    <attribute name="protein" type="r:percentage" use="required"/>
    <attribute name="carbohydrates" type="r:percentage" use="required"/>
    <attribute name="fat" type="r:percentage" use="required"/>
    <attribute name="alcohol" type="r:percentage" use="optional"/>
  </complexType>
</element>

<element name="related">
  <complexType>
    <attribute name="ref" type="NMTOKEN" use="required"/>
  </complexType>
</element>
```

53

RecipeML with XML Schema (5/5)

```
<simpleType name="nonNegativeDecimal">
  <restriction base="decimal">
    <minInclusive value="0"/>
  </restriction>
</simpleType>

<simpleType name="percentage">
  <restriction base="string">
    <pattern value="([0-9]|[1-9][0-9]|100)%"/>
  </restriction>
</simpleType>

</schema>
```

54

Problems with the XML Schema description

- calories should contain a non-negative number
- protein should contain a value of the form N% where N is between 0 and 100, solved
- comment should be allowed to appear anywhere in the contents of recipe
- unit should only be allowed in an elements where amount is also present
- nested ingredient elements should only be allowed when amount is absent

– even XML Schema has insufficient expressiveness!

55

Other Limitations of XML Schema

- The details are extremely complicated (and the spec is unreadable)
- Declarations are (mostly) context insentitive
- It is impossible to write an XML Schema description of XML Schema
- Cannot require specific root element
- Element defaults cannot contain markup
- The type system is overly complicated
- ….

56

Strengths of XML Schema

- Namespace support
- Data types (built-in and derivation)
- Modularization
- Type derivation mechanism

57

Some XML-based Standards

- MathML (http://www.w3.org/Math/)
- CML (Chemical Markup Language), see http://wwmm.ch.cam.ac.uk/moin/ChemicalMarkupLanguage
- MusicXML (http://www.recordare.com/xml.html)
- XMI (XML Metadata Interchange), see http://www.omg.org/technology/documents/formal/xmi.htm

58

XML Parsing

- Many XML parsers are available: JAXP, XERCES…
- Two “standardized” parsing methods:
  - SAX
    - event-driven
    - serial-access
    - element-by-element processing
  - DOM
    - creates a tree structure of objects
    - stores it in memory
    - easier to navigate, but more memory needed

59

JAXP

- Java API for XML Processing (JAXP) available as a separate library or as a part of Sun JDK 1.4 (*javax.xml.parsers* package)
- Implementation independent way of writing Java code for XML
- Supports the SAX and DOM parser APIs and the XSLT standard
- Allows to plug-in implementation of the parser or the processor

60

SAX: Simple API for XML

- Event driven processing of XML documents
- Parser sends events to programmer's code (start and end of every component)
- Programmer decides what to do with every event
- SAX parser doesn't create any objects at all, it simply delivers events

61

SAX Features

- SAX API acts like a data stream
- Stateless
- Events are not permanent
- Data not stored in memory
- Impossible to move backward in XML data
- Impossible to modify document structure
- Fastest and least memory intensive way of working with XML

62

Basic SAX Events

- **`startDocument`** – receives notification of the beginning of a document
- **`endDocument`** – receives notification of the end of a document
- **`startElement`** – gives the name of the tag and any attributes it might have
- **`endElement`** – receives notification of the end of an element
- **`characters`** – parser will call this method to report each chunk of character data

63

Additional SAX Events

- **`ignorableWhitespace`** – allows to react (ignore) whitespace in element content
- **`warning`** – reports conditions that are not errors or fatal errors as defined by the XML 1.0 recommendation, e.g. if an element is defined twice in a DTD
- **`error`** – *nonfatal* error occurs when an XML document fails a validity constraint
- **`fatalError`** – a non-recoverable error e.g. the violation of a well-formedness constraint; the document is unusable after the parser has invoked this method

64

SAX Events in a Simple Example

```
<?xml version="1.0"?>
<xmlExample>
   <heading>
   This is simple example.
   </heading>
   That is all folks.
</xmlExample>
```

```
startDocument()
startElement():
       xmlExample
startElement():heading
characters():
       This is simple example
endElement():heading
characters():
       That is all folks
endElement():xmlExample
endDocument()
```

65

SAX2 Handlers Interfaces

- **ContentHandler** - receives notification of the logical content of a document (`startDocument, startElement, characters` etc.)
- **ErrorHandler** - for XML processing errors generates events (`warning, error, fatalError`) instead of throwing exception (this decision is up to the programmer)
- **DTDHandler** - receives notification of basic DTD-related events, reports notation and unparsed entity declarations
- **EntityResolver** – handles the external entities

66

DefaultHandler Class

- Class **`org.xml.sax.helpers.DefaultHandler`**
- Implements all four handle interfaces with null methods
- Programmer can derive from `DefaultHandler` his own class and pass its instance to a parser
- Programmer can override only methods responsible for some events and ignore the rest

67

Parsing XML Document with JAXP SAX

- All examples for this part based on: *Simple API for XML by Eric Armstrong*
- Import necessary classes:

```
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;
```

68

Extension of DefaultHandler Class

```
public class EchoSAX extends DefaultHandler
{ public void startDocument()
      throws SAXException
      {//override necessary methods}
  public void endDocument()
      throws SAXException {...}
  public void startElement(...)
      throws SAXException{...}
  public void endElement(...)
      throws SAXException{...}
}
```

69

Overriding of Necessary Methods

```
public void startDocument() throws SAXException{
   System.out.println("DOCUMENT:");
}
public void endDocument() throws SAXException{
   System.out.println("END OF DOCUMENT:");
}
public void endElement(String namespaceURI, String sName,
  String qName) throws SAXException {
   String eName = sName; // element name
   if ("".equals(eName)) eName = qName; // not namespaceAware
   System.out.println("END OF ELEMENT: "+eName);
}
```

70

Overriding of Necessary Methods (2)

```
public void startElement(String namespaceURI, String sName,
 String qName, Attributes attrs) throws SAXException {
  String eName = sName; // element name
  if ("".equals(eName)) eName = qName; // not namespaceAware
  System.out.print("ELEMENT: ");
  System.out.print(eName);
  if (attrs != null) {
       for (int i = 0; i < attrs.getLength(); i++) {
              String aName = attrs.getLocalName(i);//Attr name
              if ("".equals(aName)) aName = attrs.getQName(i);
              System.out.print(" ");
              System.out.print(aName
                     +"=\""+attrs.getValue(i)+"\"");
          }
       }
       System.out.println("");
}
```

71

Creating new SAX Parser Instance

- `SAXParserFactory` -creates an instance of the parser determined by the system property

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

- `SAXParser` - defines several kinds of `parse()` methods. Every parsing method expects an XML data source (file, URI, stream) and a `DefaultHandler` object (or object of any class derived from `DefaultHandler`)

```
SAXParser saxParser = factory.newSAXParser();
saxParser.parse( new File("test.xml"), handler);
```

72

Validation with SAX Parser

- After creation of `SAXParserFactory` instance set the validation property to `true`

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

factory.setValidating(true);
```

73

Parsing of the XML Document

```
public static void main(String argv[]){
   // Use an instance of ourselves as the SAX event handler
   DefaultHandler handler = new EchoSAX();
   // Use the default (non-validating) parser
   SAXParserFactory factory = SAXParserFactory.newInstance();
   //Set validation on
   factory.setValidating(true);
   try {
        // Parse the input
        SAXParser saxParser = factory.newSAXParser();
        saxParser.parse( new File("test.xml"), handler);
        } catch (Throwable t) {t.printStackTrace();}
         System.exit(0);
    }
```

74

Document Object Model (DOM)

- DOM - a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents, originally defined in OMG Interface Definition Language
- DOM treats XML document as a tree
- Every tree node contains one of the components from XML structure (element node, text node, attribute node etc.)

75

DOM Features

- Document‘s tree structure is kept in the memory
- Allows to create and modify XML documents
- Allows to navigate in the structure
- DOM is language neutral –does not have all advantages of Java‘s OO features (which are available in e.g. JDOM)

76

Kinds of Nodes

- **Node** - primary datatype for the entire DOM, represents a single node in the document tree; all objects implementing the Node interface expose methods for dealing with children, not all objects implementing the Node interface may have children
- **Document** - represents the XML document, the root of the document tree, provides the primary access to the document's data and methods to create them

77

Kinds of Nodes (2)

- **Element** - represents an element in an XML document, may have associated attributes or text nodes
- **Attr**- represents an attribute in an ELEMENT object
- **Text** - represents the textual content of an ELEMENT or ATTR
- Other (COMMENT, ENTITY)

78

Common DOM Methods

- `Node.getNodeType()` – the type of the underlying object, e.g. `Node.ELEMENT_NODE`
- `Node.getNodeName()` - value of this node, depending on its type, e.g. for elements it's tag name, for text nodes always string `#text`
- `Node.getFirstChild()` and `Node.getLastChild()` - the first or last child of a given node
- `Node.getNextSibling()` and `Node.getPreviousSibling()` - the next or previous sibling of a given node
- `Node.getAttributes()` – collection containing the attributes of this node (if it is an element node) or null

79

Common DOM Methods (2)

- `Node.getNodeValue()` - value of this node, depending on its type, e.g. value of an attribute but `null` in case of an element node
- `Node.getChildNodes()` - collection that contains all children of this node
- `Node.getParentNode()` -parent of this node
- `Element.getAttribute(name)` - an attribute value by name
- `Element.getTagName()` -name of the element
- `Element.getElementsByTagName()` -collection of all descendant Elements with a given tag name

80

Common DOM Methods (3)

- `Element.setAttribute(name,value)`-adds a new attribute, if an attribute with that name is already present in the element, its value is changed
- `Attr.getValue()`-the value of the attribute
- `Attr.getName()`-the name of this attribute
- `Document.getDocumentElement()`-allows direct access to the child node that is the root element of the document
- `Document.createElement(tagName)`-creates an element of the type specified

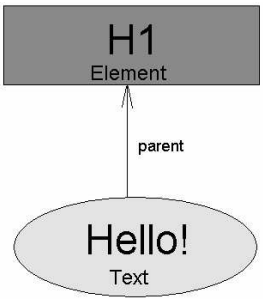
81

Text Nodes

- Text inside an element (or attribute) is considered as a child of this element (attribute) not the value of it!

```
<H1>Hello!</H1>
```

  - element named `H1` with value `null` and one text node child, which value is `Hello!` and name `#text`



82

Parsing XML with JAXP DOM

- Import necessary classes:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.*;
import org.w3c.dom.DOMException
```

83

Creating new DOM Parser Instance

- `DocumentBuilderFactory` -creates an instance of the parser determined by the system property

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

- `DocumentBuilder` -defines the API to obtain DOM Document instances from an XML document (several parse methods)

```
DocumentBuilder builder = factory.newDocumentBuilder();
document = builder.parse( new File("test.xml") );
```

84

Recursive DOM Tree Processing

```
private static void scanDOMTree(Node node){
    int type = node.getNodeType();
    switch (type){
         case Node.ELEMENT_NODE:{
                  System.out.print("<"+node.getNodeName()+">");
                  NamedNodeMap attrs = node.getAttributes();
                  for (int i = 0; i < attrs.getLength(); i++){
                           Node attr = attrs.item(i);
                           System.out.print(" "+attr.getNodeName()+
                                    "=\"" + attr.getNodeValue() +  "\"");
                  }
                  NodeList children = node.getChildNodes();
                  if (children != null)   {
                           int len = children.getLength();
                           for (int i = 0; i < len; i++)
                                    scanDOMTree(children.item(i));
                  }
                  System.out.println("</"+node.getNodeName()+">");
                  break;
         }
```

85

Recursive DOM Tree Processing (2)

```
    case Node.DOCUMENT_NODE:{
         System.out.println("<?xml version=\"1.0\" ?>");
         scanDOMTree(((Document)node).getDocumentElement());
         break;
    }
    case Node.ENTITY_REFERENCE_NODE:{
         System.out.print("&"+node.getNodeName()+";");
         break;
    }
    case Node.TEXT_NODE:{
         System.out.print(node.getNodeValue().trim());
         break;
    }
    //...
}
```

86

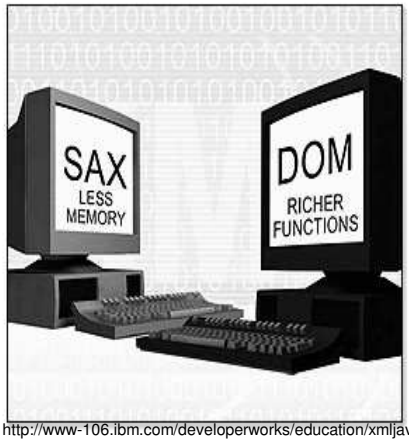
Simple DOM Example

```
public class EchoDOM{
   static Document document;
   public static void main(String argv[]){
        DocumentBuilderFactory factory =
                DocumentBuilderFactory.newInstance();
        try {
                DocumentBuilder builder =
                        factory.newDocumentBuilder();
                document = builder.parse( new File("test.xml"));
        } catch (Exception e){
                System.err.println("Sorry, an error: " + e);
        }
        if(document!=null){
                scanDOMTree(document);
        }
   }
```

87

SAX vs. DOM

- DOM
  - More information about structure of the document
  - Allows to create or modify documents
- SAX
  - You need to use the information in the document only once
  - Less memory



From http://www-106.ibm.com/developerworks/education/xmljava/

88

XSLT

- eXtensible Stylesheet Language Templates
- allows the transformation of one XML document into another by specifying transformation rules

89

XSLT History

- Nov. 1999 W3C issued XSLT as a recommendation.
  - language, written in XML, to take XML data with any desired structure and to generate something which the server can supply to the user as an HTML page, a PDF document or in XML form.
- Also in Nov.1999, language allowing access to parts of a document: *XML Path Language (XPath)*
  - describes, in the form of expressions, which elements and attributes are to be selected for processing

90

XSLT Examples: XML Document

```
<source>
   <title>XSL</title>
   <author>John Smith</author>
</source>
```

91

Simple XSLT Example

- Stylesheet:

```
<xsl:stylesheet version = '1.0'
    xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:template match="/">
    <h1>
        <xsl:value-of select="//title"/>
    </h1>
    <h2>
        <xsl:value-of select="//author"/>
    </h2>
</xsl:template>
</xsl:stylesheet>
```

- Sample output:

```
<h1>XSL</h1>
<h2>John Smith</h2>
```

92

XSLT Stylesheets

```
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="2.0">
  ...
</xsl:stylesheet>
```

- An XSLT stylesheet contains *template rules*
- The processor finds the *most specific* rule for the document root
- It then executes the template *body*

93

Use of XPath in XSLT

- Specifying *patterns* for template rules
- Selecting *nodes* for processing
- Computing *boolean* conditions
- Generating *text* contents for the output document

94

Template Rules

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

- Find the template rules that *match* the contex node (initially the whole document)
- Select the *most specific* one
- *Evaluate* the body (a *sequence constructor*)

95

Patterns and Matching

- A *pattern* is a restricted XPath expression
  - it is a union of path expressions
  - each path expression contains a number of steps separated by / or //
  - each step may only use the `child` or `attribute` axis
- A pattern *matches* a node if
  - starting from *some* node in the tree:
  - the given node is *contained* in the resulting sequence

```
rcp:recipe/rcp:ingredient//rcp:preparation
```

96

Presenting a Business Card

```
<card xmlns="http://businesscard.org">
  <name>John Doe</name>
  <title>CEO, Widget Inc.</title>
  <email>john.doe@widget.inc</email>
  <phone>(202) 555-1414</phone>
  <logo uri="widget.gif"/>
</card>
```

```
- <card>
    <name>John Doe</name>
    <title>CEO, Widget Inc.</title>
    <email>john.doe@widget.inc</email>
    <phone>(202) 456-1414</phone>
    <logo uri="widget.gif"/>
  </card>
```

97

Using XSLT

```
<?xml-stylesheet type="text/xsl"
   href="businesscard.xsl"?>
<card xmlns="http://businesscard.org">
  <name>John Doe</name>
  <title>CEO, Widget Inc.</title>
  <email>john.doe@widget.inc</email>
  <phone>(202) 555-1414</phone>
  <logo uri="widget.gif"/>
</card>
```

98

XSLT for Business Cards (1/2)

```
<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:b="http://businesscard.org"
     xmlns="http://www.w3.org/1999/xhtml">

  <xsl:template match="b:card">
    <html>
      <head>
        <title><xsl:value-of select="b:name/text()"/></title>
      </head>
      <body bgcolor="#ffffff">
        <table border="3">
          <tr>
            <td>
              <xsl:apply-templates select="b:name"/><br/>
              <xsl:apply-templates select="b:title"/><p/>
              <tt><xsl:apply-templates select="b:email"/></tt><br/>
```

99

XSLT for Business Cards (2/2)

```
              <xsl:if test="b:phone">
                Phone: <xsl:apply-templates select="b:phone"/><br/>
              </xsl:if>
            </td>
            <td>
              <xsl:if test="b:logo">
                <img src="{b:logo/@uri}"/>
              </xsl:if>
            </td>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="b:name|b:title|b:email|b:phone">
    <xsl:value-of select="text()"/>
  </xsl:template>

</xsl:stylesheet>
```

100