XML

Objectives

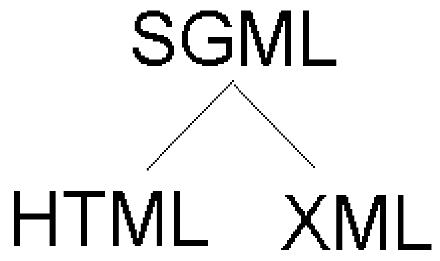
- **At the end of this session, you will be able to,**
  - Write XML document
  - Write DTD and confirm XML document to a DTD
  - Write XSD and confirm XML document to a XSD
  - Understand the function of parser
  - Distinguish between SAX – DOM and XSLT Parser
  - Write Simple XML Parser (Validating and Non-Validating)
  - Write Java Code to Read XML data

What is XML?

- XML stands for EXtensible Markup Language
  - you can create any element “name”
- XML is a markup language much like HTML
- XML was designed to describe data
- XML tags are not predefined. You must define your own tags

What is XML? (Contd…)

- XML is a W3C Recommendation
- Based on Standard Gen



age (SGML)

- XML uses a Document Type Definition (DTD) or an XML Schema (XSD) to provide grammar for the data

XML Versions

- **XML 1.0 (First Edition)**
  - XML 1.0 was released as a W3C Recommendation 10, February 1998
- **XML 1.0 (Second Edition)**
  - XML 1.0 (SE) was released as a W3C Recommendation 6, October 2000
    Second Edition is only a correction to XML 1.0 that incorporates the first-edition errata (bug fixes)
- **XML 1.0 (Third Edition)**
  - Third Edition is only a correction to XML 1.0 that incorporates the first- and second-edition errata (bug fixes)
- **XML 1.1**
  - XML 1.1 was released as a Working Draft 13, December 2001, and as a Candidate Recommendation 15, October 2002.
    XML 1.1 allows almost any Unicode characters to be used in names.

How XML Differs from HTML ??

- **HTML tags are predefined while as XML tags are user defined.**

- **XML and HTML were designed with different goals:**
  - XML was designed to describe data and to focus on what data is
  - HTML was designed to display data and to focus on how data looks

- **XML when used in conjuction with HTML, vastly extends the capability of web pages to:**
  - Deliver virtually any type of document
  - Sort, filter, rearrange, find and manipulate the information
  - Present highly structured information

XML Document

```xml
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

How can XML be used?

- **XML can separate data from HTML**
  - With XML, your data is stored outside your HTML
- **XML is used to exchange data**
  - With XML, data can be exchanged between incompatible systems
- **XML and B2B**
  - With XML, information can be exchanged over the Internet
- **XML can be used to create new languages**
  - XML is the mother of WAP(Standard used for mobile devices) and WML(Wireless Markup Language)
- **XML is a cross-platform, software and hardware independent tool for transmitting information**

Anatomy of an XML document

- **Three Parts**
  - Prolog
  - Document Element [Root Element]
  - Epilog

Prolog (Optional)

- **XML Declaration**
  - States that this is an XML Document. Must be the first line in the XML document.

    <?xml version="1.0" encoding="ISO-8859-1"?>

- **Document Type Declaration**
  - Defines the type and the structure of the document. If used, it must come after the XML declaration.

    <!DOCTYPE catalog SYSTEM “catalog.dtd”> e.g. of external subset

- **Processing Instructions**
  - Provides information that the XML processor passes on to the application. There can be one or more processing instructions that can appear in prolog.

    <?xml-stylesheet type="text/css" href="cd_catalog.css"?>

- **Comments**

    <!-- Edited with XML Spy v4.2 -->

Document Element

- Also Known as Root element.

- It contains the document's information content.
- The element content can be character data, other (nested) elements, or combination of both.

```
<note>
    This is a note.
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

- Document element is <note>. It contains 4 nested elements.

Epilog (Optional)

- The area after the root element is known as epilog
- Some markup constructs like comments, white spaces can come after the root element
- This is not an official term used in XML standard

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE CATALOG SYSTEM "cd_catalog.dtd">
<?xml-stylesheet type="text/css"
    href="cd_catalog.css"?>

<!-- Edited with XML Spy v4.2 -->

<CATALOG>
    <CD>
      <TITLE>Empire Burlesque</TITLE>
      <ARTIST>Bob Dylan</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>Columbia</COMPANY>
      <PRICE>10.90</PRICE>
      <YEAR>1985</YEAR>
    </CD>
    <CD>
      <TITLE>Unchain my heart</TITLE>
      <ARTIST>Joe Cocker</ARTIST>
      <COUNTRY>USA</COUNTRY>
      <COMPANY>EMI</COMPANY>
      <PRICE>8.20</PRICE>
      <YEAR>1987</YEAR>
    </CD>
</CATALOG>

<!-- Edited with XML Spy v4.2 -->
```

Prolog (Optional)

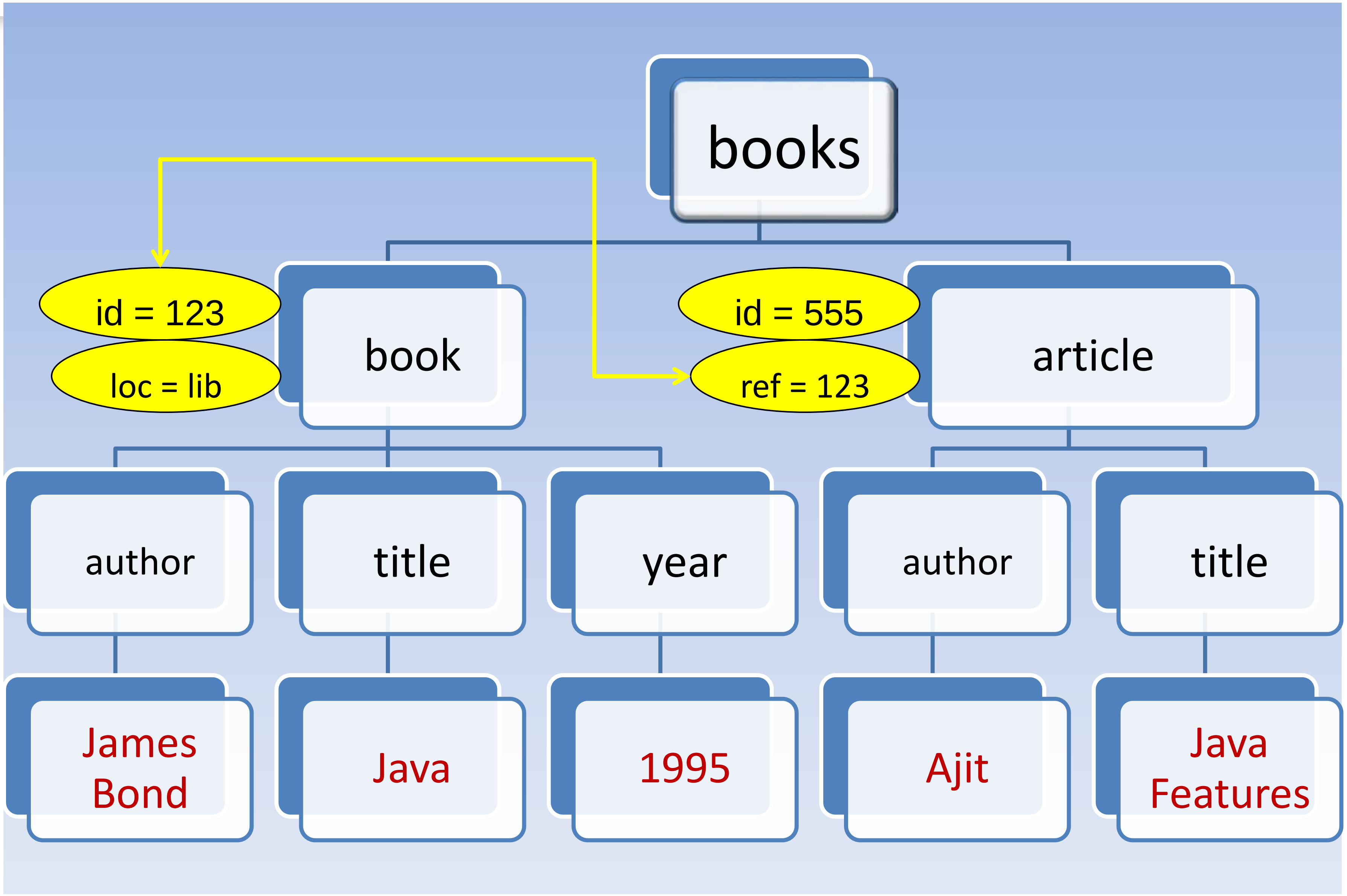
Document Element

Epilog (Optional)

Example

```
<books>
    <book id="123" loc="lib">
      <author>James Bond</author>
      <title>Java </title>
      <year>1995</year>
    </book>
    <article id="555" ref="123">
      <author>Ajit</author>
      <title>Java Features </title>
    </article>
</ books >
```

Example


books
id = 123
loc = lib
book
id = 555
ref = 123
article
author
title
year
author
title
James Bond
Java
1995
Ajit
Java Features

XML Syntax

- **All XML documents must have a root element**

  e.g.

  ```
  <root>
      <child>
          <subchild>.....</subchild>
      </child>
  </root>
  ```

- **All XML elements must have a closing tag**

  e.g. `<p>This is a paragraph </p>`

- **XML tags are case sensitive**

  e.g.

  ```
  <Message>This is incorrect</message>
  <message>This is correct</message>
  ```

- **All XML elements must be properly nested**

  e.g.

  ```
  <b><i>Invalid Nesting</b></i>
  <b><i>Valid Nesting</i></b>
  ```

XML Syntax (Contd...)

- **Attribute values must always be quoted**

  e.g. `<note date="12/11/2002"></note>`

- **Comments in XML**
  - The syntax for writing comments in XML is similar to that of HTML

    e.g. `<!-- This is a comment -->`

- **Naming Rules**
  - The name must begin with a letter or underscore, followed by zero or more letters, digits, periods, hyphens or underscores.
  - Names beginning with xml (any combination of upper and lower case) are reserved for standardization. Don’t use.
  - Names cannot contain spaces

XML Elements

- **XML Elements have relationships**
  - Elements are related as parents and children

```
<book>
  <title>My First XML</title>
  <prod id="33-657" media="paper"></prod>
  <chapter>Introduction to XML
    <para>What is HTML</para>
    <para>What is XML</para>
  </chapter>
  <chapter>XML Syntax
    <prod id="12-234" />
    <para>Elements must have a closing tag</para>
    <para>Elements must be properly nested</para>
  </chapter>
</book>
```

XML Elements

- **Elements have Content**
  - An element can have
    - element content
    - mixed content
    - simple content
    - empty content
    - An element can also have attributes

XML Attributes

- Attributes are used to provide additional information about elements
- XML elements can have attributes in the start tag, just like HTML. e.g. <note type=”reminder”>
- Attributes are enclosed in single or double quotes

  e.g. <file type="gif">computer.gif</file>

Use of Elements vs. Attributes

- Example 1

```
<person sex="female">
    <firstname>Anna</firstname>  <lastname>Smith</lastname>
</person>
```

- Example 2

```
<person>
    <sex>female</sex>
    <firstname>Anna</firstname>  <lastname>Smith</lastname>
</person>
```

Use of Elements vs. Attributes

```
<note day="12" month="11" year="2002"
    to="Tove" from="Jani" heading="Reminder"
    body="Don't forget me this weekend!">
    </note>
```

Avoid using Attributes

- **Some of the problems with using attributes are:**
  - attributes cannot contain multiple values (child elements can)
  - attributes are not easily expandable (for future changes)
  - attributes cannot describe structures (child elements can)
  - attributes are more difficult to manipulate by program code
  - attribute values are not easy to test against a Document Type Definition (DTD) - which is used to define the legal elements of an XML document

- **If you use attributes as containers for data, you end up with documents that are difficult to read and maintain**
- **Try to use elements to describe data**
- **Use attributes only to provide information that is not relevant to the data**

When to use Attribute

- **Assigning ID references to elements**
  - These ID references can be used to access XML elements in much the same way as the NAME or ID attributes in HTML. This example demonstrates this:

```
<messages>
  <note id="p501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
</messages>
```

Entity Reference

- **There are 5 predefined entity references in XML**

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

XML CDATA

- All the text in an XML document is parsed by the parser.
  - The parser does this because XML elements can contain other elements
- Only the text inside a CDATA section is ignored by the parser

```
<script>
  <![CDATA[
    function matchwo(a,b)
    {
      if (a < b && a < 0) then
      {
        return 1
      }
      else
      {
        return 0
      }
    }
  ]]>
</script>
```

- Nested CDATA sections are not allowed

“Well-formed” XML Documents

- **XML with correct syntax is well-formed XML**
  - All elements must have an end tag
  - All elements must be cleanly nested
  - All attribute values must be enclosed in quotation marks
  - Each document must have a unique first element, the root node

XML Namespaces

- **Name conflicts**
  - This XML document carries information in a table:

```
<table>
  <tr>
  <td>Apples</td>
  <td>Bananas</td>
  </tr>
</table>
```

  - This XML document carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

XML Namespaces (Contd…)

Solving Name Conflicts Using a Prefix

- This XML document carries information in a table:

```
<h:table>
  <h:tr>
  <h:td>Apples</h:td>
  <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

- This XML document carries information about a piece of furniture:

```
<f:table>
   <f:name>African Coffee Table</f:name>
   <f:width>80</f:width>
   <f:length>120</f:length>
</f:table>
```

XML Namespaces (Contd…)

- **XML Namespaces provide a method to avoid element name conflicts**
  - This XML document carries information in a table:

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
  <h:td>Apples</h:td>
  <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

- **This XML document carries information about a piece of furniture:**

```
<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

XML Namespaces – xmlns Attribute

- **The XML namespace attribute is placed in the start tag of an element and has the following syntax:**

  xmlns:namespace-prefix="namespaceURI“

  - All child elements with the same prefix are associated with the same namespace.
  - Imp: the address used to identify the namespace is not used by the parser to look up information. The only purpose is to give the namespace a unique name. However, very often companies use the namespace as a pointer to a real web page containing information about the namespace.

XML Namespaces – Default Namespace

- **Defining a default namespace for an element saves us from using prefixes in all the child elements.**

  xmlns="namespaceURI“

- **This XML document carries information in a table:**

```
<table xmlns="http://www.w3.org/TR/html4/">
  <tr>
  <td>Apples</td>
  <td>Bananas</td>
  </tr>
</table>
```

- **This XML document carries information about a piece of furniture:**

```
<table xmlns="http://www.w3schools.com/furniture">
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

Document Type Definition (DTD)

Why….. DTD?

- Define the legal building blocks of an XML document
- It defines the document structure with a list of legal elements
- With DTD, each of your XML files can carry a description of its own format with it
- With a DTD, independent groups of people can agree to use a common DTD for interchanging data
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid
- You can also use a DTD to verify your own data

What … .DTD?

- **Defines structure of the document**
  - Allowable tags and their attributes
  - Attribute values constraints
  - Nesting of tags
  - Number of occurrences of tags
  - Entity definitions

How….DTD?

- A DTD can be declared inline in your XML document, or as an external reference

- If the DTD is included in your XML source file, it should be wrapped in a DOCTYPE definition with the following syntax:

  <!DOCTYPE root-element [element-declarations]>

Internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

External DTD

- **If the DTD is external to your XML source file, it should be wrapped in a DOCTYPE definition with the following syntax:**

```
<!DOCTYPE root-element SYSTEM "filename">
                                    or
<!DOCTYPE root-element PUBLIC "filename">
```

External DTD

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

External DTD

- **And this is a copy of the file "note.dtd" containing the DTD:**

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Building Blocks of XML Documents

- **Seen from a DTD point of view, all XML documents are made up by the following simple building blocks:**
  - Elements
  - Tags
    - used to markup elements
  - Attributes
    - provide extra information about elements
  - Entities
    - are variables used to define common text
  - PCDATA (Parsed Character Data)
    - PCDATA is text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded.
  - CDATA (Character Data)
    - CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

DTD Declaration

<! KEYWORD parameter1, parameter2, parameter3, …. .. parameterN>

KEYWORD can be

- ELEMENT
- ATTLIST
- ENTITY
- NOTATION

DTD - Elements

- **In a DTD, XML elements are declared with a DTD element declaration**

```
<!ELEMENT element-name category>
or
<!ELEMENT element-name (element-content)>
```

- **There are five categories of element content**

1. ANY (Any well-formed XML data)
2. EMPTY (May not contain any text or child elements)
3. Text Only (Contains any text but no child element)
4. Element Only (Contains only child elements and no text outside of those children)
5. Mixed (May contain a mixture of child elements and/or text data)

DTD – Elements Examples

- **Empty elements**

  ```
  <!ELEMENT br EMPTY>
  XML example:
  <br/>
  ```

- **Elements with any contents**

  ```
  <!ELEMENT element-name ANY>
  example:
  <!ELEMENT note ANY>
  ```

- **Elements with only character data**

  ```
  <!ELEMENT element-name (#PCDATA)>
  example:
  <!ELEMENT from (#PCDATA)>
  ```

DTD – Elements Examples

- **Elements with Children**

```
<!ELEMENT element-name  (child-element-name)>
or
<!ELEMENT element-name  (child-element-name, child-element-name,.....)>
example:
<!ELEMENT note (to, from, heading, body)>
```

  - When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document.
  - In a full declaration, the children must also be declared, and the children can also have children.

```
<!ELEMENT to      (#PCDATA)>
<!ELEMENT from    (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body    (#PCDATA)
```

DTD – Elements Examples

- **Elements with Mixed contents**
  example:<!ELEMENT note (#PCDATA|to|from|header|message)*>

- **Declaring only one occurrence of the same element**

  <!ELEMENT element-name (child-name)>

  example:

  <!ELEMENT note (message)>

- **Declaring minimum one occurrence of the same element**

  <!ELEMENT element-name (child-name+)>

  example:

  <!ELEMENT note (message+)>

DTD – Elements Examples

- **Declaring zero or more occurrence of the same element**

  <!ELEMENT element-name (child-name*)>

  example:

  <!ELEMENT note (message*)>

- **Declaring zero or one occurrences of the same element**

  <!ELEMENT element-name (child-name?)>

  example:

  <!ELEMENT note (message?)>

- **Declaring either/or content**

  example:

  <!ELEMENT note (to,from,header,(message|body))>

Occurrence Indicator

Indicator	Occurrence	
(no indicator)	Required	One and only one
?	Optional	None or one
*	Optional, repeatable	None, one, or more
+	Required, repeatable	One or more

DTD Attributes

- **Syntax**
  - <!ATTLIST element-name attribute-name
  - attribute-type default-value>
- **DTD example:**
  - <!ATTLIST payment type CDATA "check">
- **XML example:**
  - <payment type="check" />

DTD Attributes: Attribute Defaults

Value	**Explanation**
value	The default value of the attribute
#REQUIRED	The attribute value must be included in the element
#IMPLIED	The attribute does not have to be included
#FIXED value	The attribute value is fixed

DTD Attributes: Examples

- **Specifying a Default attribute value**

- **DTD:**
  - <!ELEMENT square EMPTY>
  - <!ATTLIST square width CDATA "0">
- **Valid XML:**
  - <square width="100" />
  - In the example above, the "square" element is defined to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

DTD Attributes: Examples

- **Specifying Implied Attribute Value**
  - <!ATTLIST element-name attribute-name
  - attribute-type #IMPLIED>
- **Example**
  - DTD:
    <!ATTLIST contact fax CDATA #IMPLIED>
  - Valid XML:
    <contact fax="555-667788" />
  - Valid XML:
    <contact />
- **Use the #IMPLIED keyword if you do not want to force the author to include an attribute, and you do not have an option for a default value**

DTD Attributes: Examples

- **Specifying Fixed Attribute Value**
  - <!ATTLIST element-name attribute-name
  - attribute-type #FIXED "value">
- **Example**
  - DTD:
    <!ATTLIST sender company CDATA #FIXED "Microsoft">
  - Valid XML:
    <sender company="Microsoft" />
  - Invalid XML:
    <sender company="W3Schools" />
- **Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.**

DTD Attributes: Examples

- Enumerated attribute values
- Syntax:

  <!ATTLIST element-name

  attribute-name (en1|en2|..) default-value>
- DTD example:

  <!ATTLIST payment type (cheque|cash) "cash">

- XML example:

  <payment type="cheque" />

  or

  <payment type="cash" />
- Use enumerated attribute values when you want the attribute values to be one of a fixed set of legal values.

DTD Attributes: Examples

- **ID, IDREF and IDREFS**
  - <!ELEMENT item (#PCDATA)>
  - <!ATTLIST item stock-code ID #REQUIRED goes-with IDREF #IMPLIED>
  - Example:
  - <item stock-code=“s034”>Electric Goods</item>
  - <item stock-code=“s036” goes-with=“s034”> fan </item>

DTD Entity

- **Entities are variables used to define shortcuts to common text**
  - Entity references are references to entities
  - Entities can be declared internal, or external

- **Syntax:**

  <!ENTITY entity-name "entity-value"> → Internal Entity

  <!ENTITY entity-name SYSTEM "URI/URL"> → External Entity

DTD Entity

- **Internal Entity declaration**

DTD Example:

```
<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">
```

XML example:

```
<author>&writer;&copyright;</author>
```

DTD Entity: Examples

- **External Entity declaration**
  - DTD Example:
  - <!ENTITY writer
  - SYSTEM "http://www.w3schools.com/dtd/entities.dtd">
  - <!ENTITY copyright
  - SYSTEM "http://www.w3schools.com/dtd/entities.dtd">
  - XML example:
  - <author>&writer;&copyright;</author>

Limitations of DTD

- DTD itself is not in XML format – more work for parsers
- Does not express data types (weak data typing)
- No namespace support
- Document can override external DTD definitions
- No DOM support
- XML Schema is intended to resolve these issues but, DTDs are going to be around for a while

XML Schema Definition (XSD)

XML Schemas

- “XML Schema” is an XML-based alternative to DTDs
- An XML Schema describes the structure of an XML document
- The XML Schema language is also referred to as XML Schema Definition (XSD)

Schema Definition

- **An XML Schema defines:**
  - Elements that can appear in a document
  - Attributes that can appear in a document
  - Which elements are child elements
  - Order of child elements
  - Number of child elements
  - Whether an element is empty or can include text
  - Data types for elements and attributes
  - Default and fixed values for elements and attributes

Why XML Schemas?

- **DTDs provide weak specification language**
  - We can not put any restriction on text content
  - We have very little control over mixed content (text plus elements)
  - We have little control over ordering of elements
- **DTDs are written in a strange (non-XML) format**
  - We need separate parsers for DTDs and XML
- **The XML Schema Definition language solves these problems**
  - XSD gives you much more control over structure and content
  - XSD is written in XML

Referring to a Schema

- To refer to a DTD in an XML document, the reference goes before the root element:

```
<?xml version="1.0"?>
<!DOCTYPE rootElement SYSTEM "url">
<rootElement> … </rootElement>
```

- To refer to an XML Schema in an XML document, the reference goes in the root element:

```
<?xml version="1.0"?>
<rootElement
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          (The XML Schema Instance reference is required)
xsi:noNamespaceSchemaLocation="url.xsd">
          (This is where the XML Schema definition can be found)
  ...
</rootElement>
```

The XSD Document

- Since the XSD is written in XML, it can get confusing which we are talking about
- Except for the additions to the root element of our XML data document, we will discuss about the XSD schema document
- The file extension is .xsd
- The root element is <schema>
- The XSD starts like this:

  -
    ```
    <?xml version="1.0"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    ```

<schema>

- **The <schema> element may have attributes:**
  - xmlns:xs="http://www.w3.org/2001/XMLSchema"
    - This is necessary to specify where all our XSD tags are defined

“Simple” and “Complex” Elements

- **A “simple” element is one that contains text and nothing else**
  - It cannot have attributes
  - It cannot contain other elements
  - It cannot be empty
  - However, the text can be of many different types, and may have various restrictions applied to it
- **If an element is not simple, it is “complex”**
  - A complex element may have attributes
  - It may be empty, or it may contain text, other elements, or both text and other elements

Defining a simple element

- **A simple element is defined as**
  <xs:element name="*name*" type="*type*" />
  where:
  - *name* is the name of the element
  - the most common values for *type* are

xs:boolean	xs:integer
xs:date	xs:string
xs:decimal	xs:time

- **Other attributes a simple element may have:**
  - default="*default value*" if no other value is specified
  - fixed="*value*" no other value may be specified

Defining an Attribute

- Attributes themselves are always declared as simple types
- **An attribute is defined as**
  <xs:attribute name="*name*" type="*type*" />
  where:
  - *name* and *type* are the same as for xs:element
- **Other attributes a simple element may have:**

default="*default value*"	if no other value is specified
fixed="*value*"	no other value may be specified
use="optional"	the attribute is not required (default)
use="required"	the attribute must be present

Restrictions, or “facets”

- The general form for putting a restriction on a text value is:
  -
```
<xs:element  name="name">              (or xs:attribute)
   <xs:restriction base="type">
       ... the restrictions ...
    </xs:restriction>
</xs:element>
```
- For example:
  -
```
<xs:element  name="age">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="0">
        <xs:maxInclusive value="140">
    </xs:restriction>
</xs:element>
```

Restrictions on Numbers

- **minInclusive: number must be ≥ the given *value***
- **minExclusive: number must be > the given *value***
- **maxInclusive: number must be ≤ the given *value***
- **maxExclusive: number must be < the given *value***
- **totalDigits: number must have exactly *value* digits**
- **fractionDigits: number must have no more than *value* digits after the decimal point**

Restrictions on Strings

- **Length: the string must contain exactly value characters**
- **minLength: the string must contain at least value characters**
- **maxLength: the string must contain no more than value characters**
- **Pattern: the value is a regular expression that the string must match**
- **whiteSpace: not really a “restriction”; tells what to do with whitespace**
  - value="preserve“: Keeps all whitespace
  - value="replace“: Changes all whitespace characters to spaces
  - value="collapse“: Removes leading and trailing whitespace, and replaces all sequences of whitespace with a single space

Enumeration

- An enumeration restricts the value to be one of a fixed set of values
- Example:

  -
    ```
    <xs:element name="season">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Spring"/>
          <xs:enumeration value="Summer"/>
          <xs:enumeration value="Autumn"/>
          <xs:enumeration value="Fall"/>
          <xs:enumeration value="Winter"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    ```

Complex Elements

- A complex element is defined as

```
<xs:element  name="name">
    <xs:complexType>
        ... information about the complex type...
    </xs:complexType>
</xs:element>
```

- **Example:**

```
<xs:element name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- Remember that attributes are always simple types

Declaration and Use

- So far we've been talking about how to *declare* types, not how to *use* them
- To *use* a type we have declared, use it as the value of **type="..."**
  - Examples:
    - `<xs:element name="student" type="person"/>`
    - `<xs:element name="professor" type="person"/>`
  - Scope is important: you cannot use a type if is local to some other type

xs:sequence

- We've already seen an example of a complex type whose elements must occur in a specific order:

```
<xs:element   name="person">
   <xs:complexType>
      <xs:sequence>
         <xs:element  name="firstName"  type="xs:string" />
         <xs:element  name="lastName"  type="xs:string" />
      </xs:sequence>
   </xs:complexType>
 </xs:element>
```

Text element with attributes

- **If a text element has attributes, it is no longer a simple type**

```
<xs:element name="population">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension  base="xs:integer">
                    <xs:attribute  name="year" type="xs:integer">
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
</xs:element>
```

Predefined Date and Time Types

- **xs:date: A date in the format *CCYY-MM-DD*, for example, 2002-11-05**
- **xs:time: A date in the format *hh:mm:ss* (hours, minutes, seconds)**
- **xs:dateTime: Format is *CCYY-MM-DDThh:mm:ss***
- **Allowable restrictions on dates and times:**
  - enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, pattern, whiteSpace

Predefined Numeric Types

- **Some of the predefined numeric types:**

xs:decimal	xs:positiveInteger
xs:byte	xs:negativeInteger
xs:short	xs:nonPositiveInteger
xs:int	xs:nonNegativeInteger
xs:long	

- **Allowable restrictions on numeric types:**
  - enumeration, minInclusive, maxExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace

Sample XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="customername" type="xs:string"/>
    <xs:element name="salesperson" type="xs:string"/>
     <xs:element name="shippingaddress" type="xs:string"/>
     <xs:element name="orderdate" type="xs:string"/>
     <xs:element name="order">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="orderid"/>
                            <xs:element ref="customername"/>
                            <xs:element ref="orderdate"/>
                            <xs:element ref="salesperson"/>
                            <xs:element ref="shippingaddress"/>
                    </xs:sequence>
            </xs:complexType>
     </xs:element>
            <xs:element name="orderid">
            <xs:simpleType>
                    <xs:restriction base="xs:byte">
                            <xs:enumeration value="1"/>
                            <xs:enumeration value="2"/>
                    </xs:restriction>
            </xs:simpleType>
     </xs:element>
     <xs:element name="orders">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="order" maxOccurs="unbounded"/>
                    </xs:sequence>
            </xs:complexType>
     </xs:element>
</xs:schema>
```

Sample XML with XSD

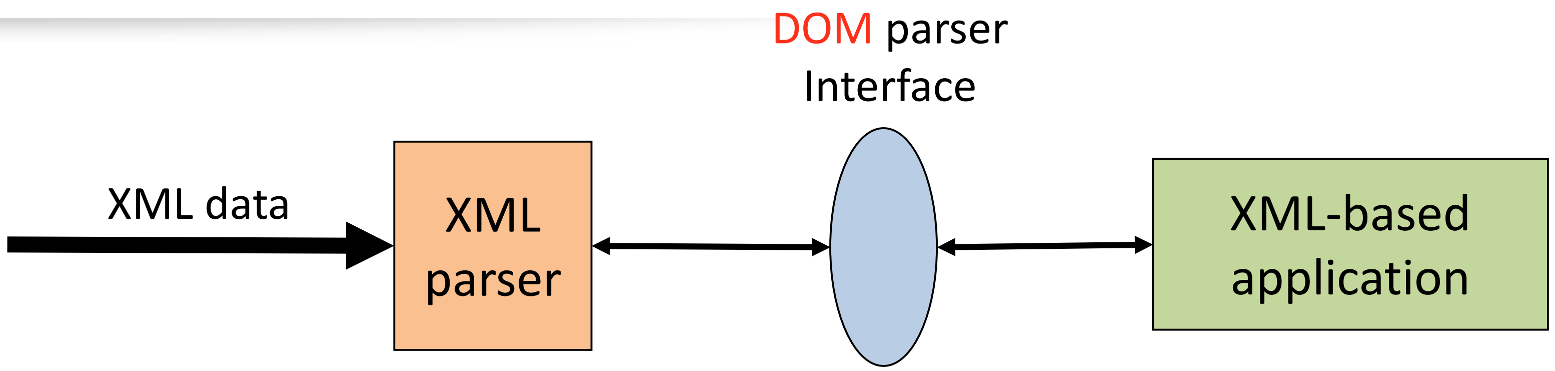
```
<?xml version="1.0"?>
<orders
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=“example2.xsd">
    <order>
         <orderid>1</orderid>
        <customername>Nova Systems</customername>
        <orderdate>10-april-2001</orderdate>
        <salesperson>Jack Nicholson</salesperson>
        <shippingaddress>12,green park</shippingaddress>
    </order>
    <order>
        <orderid>2</orderid>
        <customername>Iflex Systems</customername>
        <orderdate>20-april-2001</orderdate>
        <salesperson>Bill Gates</salesperson>
        <shippingaddress> M.G.Road </shippingaddress>
    </order>
</orders>
```

Java API for XML Processing (JAX-P)

XML Parser

- To read and update, create and manipulate an XML document, we need an XML parser
- The parser loads the document into the computer's memory
- Once the document is loaded, its data can be manipulated using the DOM
- The DOM treats the XML document as a tree

XML Processing: The XML Parser



- The parser must verify that the XML is syntactically correct
- Such data is said to be *well-formed*
- A parser MUST stop processing if the data isn't well-formed
  - E.g., stop processing and "throw an exception" to the XML-based application. The XML 1.0 spec *requires* this behaviour

Introduction to JAXP

1. The Java API for XML Processing (JAX-P) is for processing XML data
2. The JAXP APIs are defined in the **javax.xml.parsers** package
3. **SAXParserFactory** - give you a SAXParser
4. **DocumentBuilderFactory** - give you a DocumentBuilder
5. **TransformerFactory** - give you a XSLT transformer

   creates a DOM-compliant Document object

Overview of the Packages

- The libraries that define JAX-P APIs are as follows:

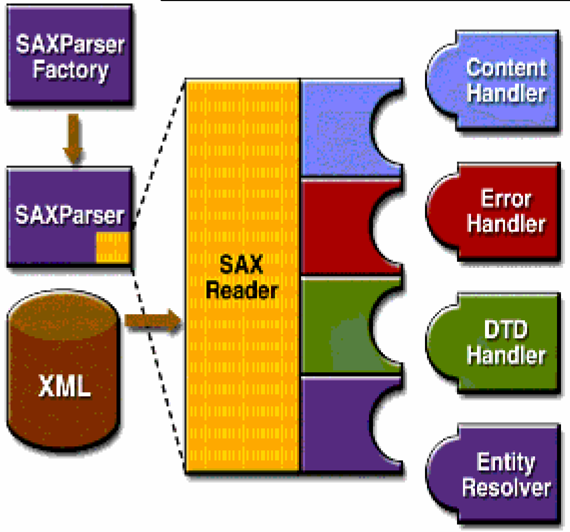
- **javax.xml.parsers:** The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers
- **org.w3c.dom:** Defines the Document class (a DOM) as well as classes for all the components of a DOM
- **org.xml.sax:** Defines the basic SAX APIs
- **javax.xml.transform:** Defines the XSLT APIs that let you transform XML into other forms
- **javax.xml.stream:** Provides StAX-specific transformation APIs

SAX – DOM - XSLT

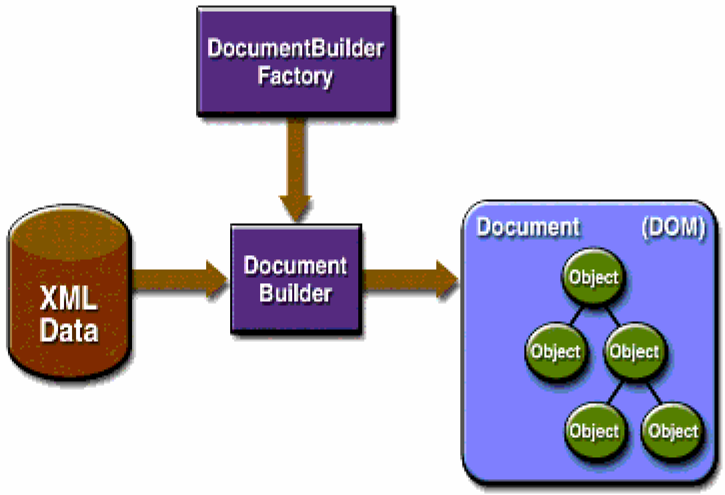
- The Simple API for XML (SAX) is the event-driven, serial-access mechanism that does element-by-element processing.
- The API for this level reads and writes XML to a data repository or the web
- The DOM API provides a tree structure of objects.
- DOM API can be use to manipulate the hierarchy of application objects it encapsulates.
- The DOM API is ideal for interactive applications because the entire object model is present in memory
- DOM requires reading the entire XML structure and holding the object tree in memory, so it is much more CPU- and memory-intensive.
- The XSLT APIs defined in javax.xml.transform let you write XML data to a file or convert it into other forms

SAX – DOM - XSLT

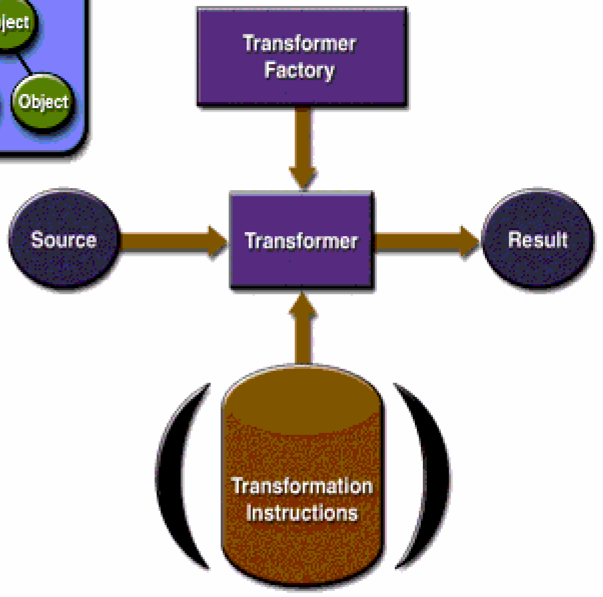
SAX Parser



DOM Parser



XSLT Parser



Document Object Model

- A DOM is a standard tree structure
- In a tree where each node contains one of the components from an XML structure
- The two most common types of nodes are element nodes and text nodes

Creating DOM with JAX-P

- Instantiate the Factory

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

- Get a Parser

```
DocumentBuilder db = dbf.newDocumentBuilder();
```

  - By default, the factory returns a non-validating parser

- Parse the File

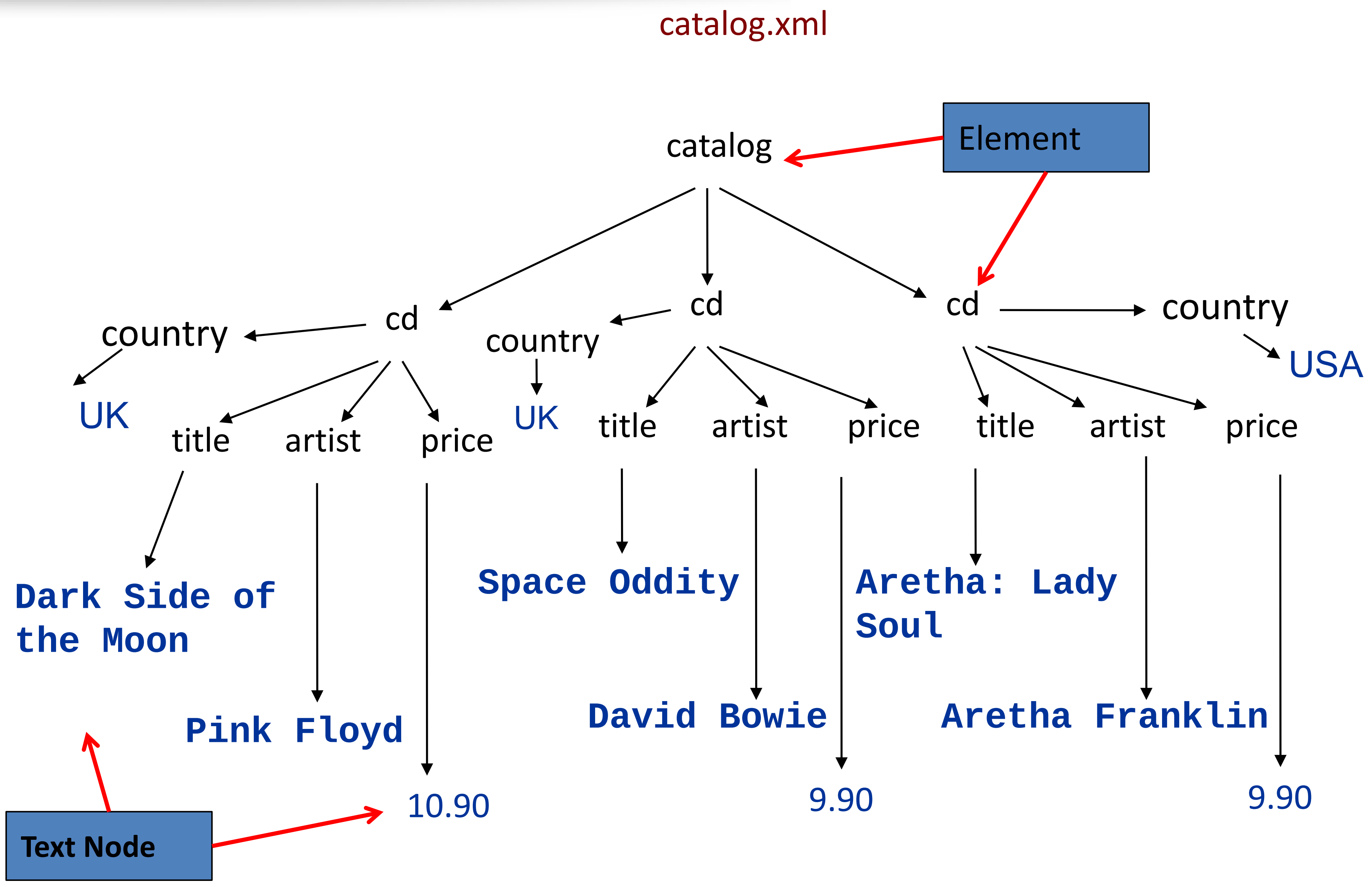
```
Document dom = db.parse(new File(filename));
```

  – *parse ()* method will return DOM

An XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
    <cd country="UK">
      <title>Dark Side of the Moon</title>
      <artist>Pink Floyd</artist>
      <price>10.90</price>
    </cd>
    <cd country="UK">
        <title>Space Oddity</title>
        <artist>David Bowie</artist>
        <price>9.90</price>
    </cd>
    <cd country="USA">
        <title>Aretha: Lady Soul</title>
        <artist>Aretha Franklin</artist>
        <price>9.90</price>
   </cd>
</catalog>
```

DOM (Document Object Model)



DOM Node Types

Node	Node Name	Node Value	Attributes
Attr	Name of attribute	Value of attribute	null
CDATASection	#cdata-section	Content of the CDATA section	null
Comment	#comment	Content of the comment	null
Document	#document	null	null
DocumentFragment	#documentFragment	null	null
DocumentType	Document Type name	null	null
Element	Tag name	null	null
Entity	Entity name	null	null
EntityReference	Name of entity referenced	null	null
Notation	Notation name	null	null
ProcessingInstruction	Target	Entire content excluding the target	null
Text	#text	Content of the text node	null

Sample Code to Read XML

```
NodeList list = dom.getElementsByTagName("catalog");
for (int i = 0; i < list.getLength(); i++) {
            NodeList childList = list.item(i).getChildNodes();
for (int j = 0; j < childList.getLength(); j++) {
            NodeList grandChildList = childList.item(j).getChildNodes();
for (int k = 0; k < grandChildList.getLength(); k++) {
            NodeList lastLevel = grandChildList.item(k).getChildNodes();
    for (int l = 0; l < lastLevel.getLength(); l++) {
            if (lastLevel.item(l).getNodeType() == Node.TEXT_NODE) {
System.out.println(lastLevel.item(l).getNodeValue());
}
}
}
}
}
```

Summary

- **In this session, we have covered,**
  - Introduction to JAXP
  - Comparison between SAX – DOM – XSLT Parser
  - Document Object Model
  - Creating DOM with JAX-P
  - Accessing XML Using JAX-P

Summary

- **In this session, we have covered:**
  - Introduction to XML
  - XML Syntax (well formed XML)
  - XML Namespace
  - DTD
  - XSD
  - Introduction to DOM

Thank You