Swing IDB-BISEW
Xaml:
Class: 107
What is an xml?
Ans.

- **Xml** (eXtensible Markup Language) is a mark up language.
- XML is designed to store and transport data.
- Xml was released in late 90’s. it was created to provide an easy to use and store self describing data.
- XML became a W3C Recommendation on February 10, 1998.
- XML is not a replacement for HTML.
- XML is designed to be self-descriptive.
- XML is designed to carry data, not to display data.
- XML tags are not predefined. You must define your own tags.
- XML is platform independent and language independent.

What a well-formed xml document is?
Ans.

An XML document is called well-formed if it satisfies certain rules, specified by The W3C. These rules are:

- **A well-formed XML document must have a corresponding end tag for all of its start tags.**

- **Nesting of elements within each other in an XML document must be proper. For example, <tutorial><topic>XML</topic></tutorial> is a correct way of nesting but <tutorial><topic>XML</tutorial></topic> is not.**

- **In each element two attributes must not have the same value. For example, <tutorial id="001"><topic>XML</topic></tutorial> is right,but <tutorial id="001" id="w3r"><topic>XML</topic></tutorial> is incorrect.**

- **Markup characters must be properly specified. For example, <tutorial id="001"><topic>XML</topic></tutorial> is right, not <tutorial id="001" id="w3r"><topic>XML</topic></tutorial>**

- **An XML document can contain only one root element. So, the root element of an xml document is an element which is present only once in an xml document and it does not appear as a child element within any other element.**

OR….
An XML document with correct syntax is called "Well Formed".
The syntax rules were described in the previous chapters:

- **XML documents must have a root element**
- **XML elements must have a closing tag**
- **XML tags are case sensitive**
- **XML elements must be properly nested**
- **XML attribute values must be quoted**

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
```

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

Example of well-formed xml document:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
```

Check it here: https://validator.w3.org/check

What constitutes a valid xml document?

XML Validation

A well formed XML document can be validated against DTD or Schema.
A well-formed XML document is an XML document with correct syntax. It is very necessary to know about valid XML document before knowing XML validation.

Valid XML document

It must be well formed (satisfy all the basic syntax condition)
It should be behave according to predefined DTD or XML schema

---

Rules for well formed XML

- **It must begin with the XML declaration.**
- **It must have one unique root element.**
- **All start tags of XML documents must match end tags.**
- **XML tags are case sensitive.**
- **All elements must be closed.**
- **All elements must be properly nested.**
- **All attributes values must be quoted.**
- **XML entities must be used for special characters.**

What the components an xml document are and how they are used?
Check it to know more:
https://www.scribd.com/doc/4712578/Components-of-an-XML-Document

components of XML:

Processing Instructions:

An XML Documents usually begins with the XML declaration statement called the Processing Instructions .This statement provides information on how the XML file should be processed.

e.g. <?xml version =”1.0” encoding=”UTF-8”?>

The Processing Instruction statement uses the encoding property to specify the encoding scheme used to create the XML file

Tags:

Tags are used to specify a name for a given piece of information. It is a means of identifying data. Data is marked up using tags.

Elements:

Elements are the basic units used to identify and describe the data in XML. They are the building blocks of an XML document. Elements are represented using tags.

Content:

Content refers to the information represented by the elements of an XML document. Consider the following example:
<name>ram</name >
Here ram is content

Attributes:

Attributes provide additional information about the elements for which they are declared. An attribute consists of a name-value pair. Consider the following example:

<Student_name S_ID = “101”>shanshak </ Student_name >

Entities:

An entity is a name that is associated with a block of data, such as chunk of text or a reference to an external file that contains textual or binary information. It is a set of information that can be specifying a single name.

Comments:

Comments are statements used to explain the XML code. They are used to provide documentation information about the XML file or the application to which the file belongs. The parser ignores comments entries during code execution.

What xml dtd is and how it is defined?

A DTD defines the legal elements of an XML document
In simple words we can say that a DTD defines the document structure with a list of legal elements and attributes.
XML schema is a XML based alternative to DTD.
Actually DTD and XML schema both are used to form a well formed XML document.
We should avoid errors in XML documents because they will stop the XML programs.

XML schema

It is defined as an XML language
Uses namespaces to allow for reuses of existing definitions
It supports a large number of built in data types and definition of derived data types

What DTD is and how it is defined

What is DTD

DTD stands for Document Type Definition. It defines the legal building blocks of an XML document. It is used to define document structure with a list of legal elements and attributes.

Purpose of DTD

Its main purpose is to define the structure of an XML document. It contains a list of legal elements and define the structure with the help of them.

Checking Validation

Before proceeding with XML DTD, you must check the validation. An XML document is called "well-formed" if it contains the correct syntax.
A well-formed and valid XML document is one which have been validated against DTD.
Visit http://www.xmlvalidation.com to validate the XML file.

Valid and well-formed XML document with DTD

Let's take an example of well-formed and valid XML document. It follows all the rules of DTD.

employee.xml

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
 <firstname>Mr</firstname>
 <lastname>reza</lastname>
 <email>info@coderbd.com</email>
</employee>
```

In the above example, the DOCTYPE declaration refers to an external DTD file. The content of the file is shown in below paragraph.

employee.dtd

```
<!ELEMENT employee (firstname,lastname,email)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Description of DTD

<!DOCTYPE employee : It defines that the root element of the document is employee.
<!ELEMENT employee: It defines that the employee element contains 3 elements "firstname, lastname and email".
<!ELEMENT firstname: It defines that the firstname element is #PCDATA typed. (parse-able data type).
<!ELEMENT lastname: It defines that the lastname element is #PCDATA typed. (parse-able data type).
<!ELEMENT email: It defines that the email element is #PCDATA typed. (parse-able data type).

XML DTD with entity declaration

A doctype declaration can also define special strings that can be used in the XML file.
An entity has three parts:

1. An ampersand (&)
2. An entity name
3. A semicolon (;)

Syntax to declare entity:

```
<!ENTITY entity-name "entity-value">
```

Let's see a code to define the ENTITY in doctype declaration.

author.xml

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE author [
 <!ELEMENT author (#PCDATA)>
 <!ENTITY sj "Mr Reza">
]>
```

6. **<author>&sj;</author>**

What namespaces are and why you use it?
XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.
This XML carries HTML table information:

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

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

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

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.
A user or an XML application will not know how to handle these differences.

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.
This XML carries information about an HTML table, and a piece of furniture:

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

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

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a **namespace** for the prefix must be defined.
The namespace can be defined by an **xmlns** attribute in the start tag of an element.
The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

```
<root>

<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>

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

</root>
```

In the example above:
The xmlns attribute in the first <table> element gives the h: prefix a qualified namespace.
The xmlns attribute in the second <table> element gives the f: prefix a qualified namespace.
When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.
Namespaces can also be declared in the XML root element:

```
<root xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="https://www.w3schools.com/furniture">

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

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

</root>
```

Note: The namespace URI is not used by the parser to look up information.
The purpose of using an URI is to give the namespace a unique name.
However, companies often use the namespace as a pointer to a web page containing namespace information.

Uniform Resource Identifier (URI)

A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.
The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Universal Resource Name** (URN).

Default Namespaces

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:
xmlns="*namespaceURI*"
This XML carries HTML table information:

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

This XML carries information about a piece of furniture:

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

Namespaces in Real Use

XSLT is a language that can be used to transform XML documents into other formats.
The XML document below, is a document used to transform XML into HTML.
The namespace "http://www.w3.org/1999/XSL/Transform" identifies XSLT elements inside an HTML document:

```
<?xml version="1.0" encoding="UTF-8"?>

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

<xsl:template match="/">
<html>
<body>
 <h2>My CD Collection</h2>
 <table border="1">
   <tr>
    <th style="text-align:left">Title</th>
    <th style="text-align:left">Artist</th>
   </tr>
   <xsl:for-each select="catalog/cd">
   <tr>
    <td><xsl:value-of select="title"/></td>
```

```
     <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
 </table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Difference between PCDATA and CDATA in DTD

PCDATA - Parsed Character Data
XML parsers normally parse all the text in an XML document.
CDATA - (Unparsed) Character Data
The term CDATA is used about text data that should not be parsed by the XML parser.
Characters like "<" and "&" are illegal in XML elements.

Sax API?

SAX (Simple API for XML)

A SAX Parser implements SAX API. This API is an event based API and less intuitive.

Features of SAX Parser

It does not create any internal structure.
Clients does not know what methods to call, they just overrides the methods of the API and place his own code inside method.
It is an event based parser, it works like an event handler in Java.

Advantages

1) It is simple and memory efficient.
2) It is very fast and works for huge documents.

Disadvantages

1) It is event-based so its API is less intuitive.
2) Clients never know the full information because the data is broken into pieces.

XML DOM?

What is XML DOM

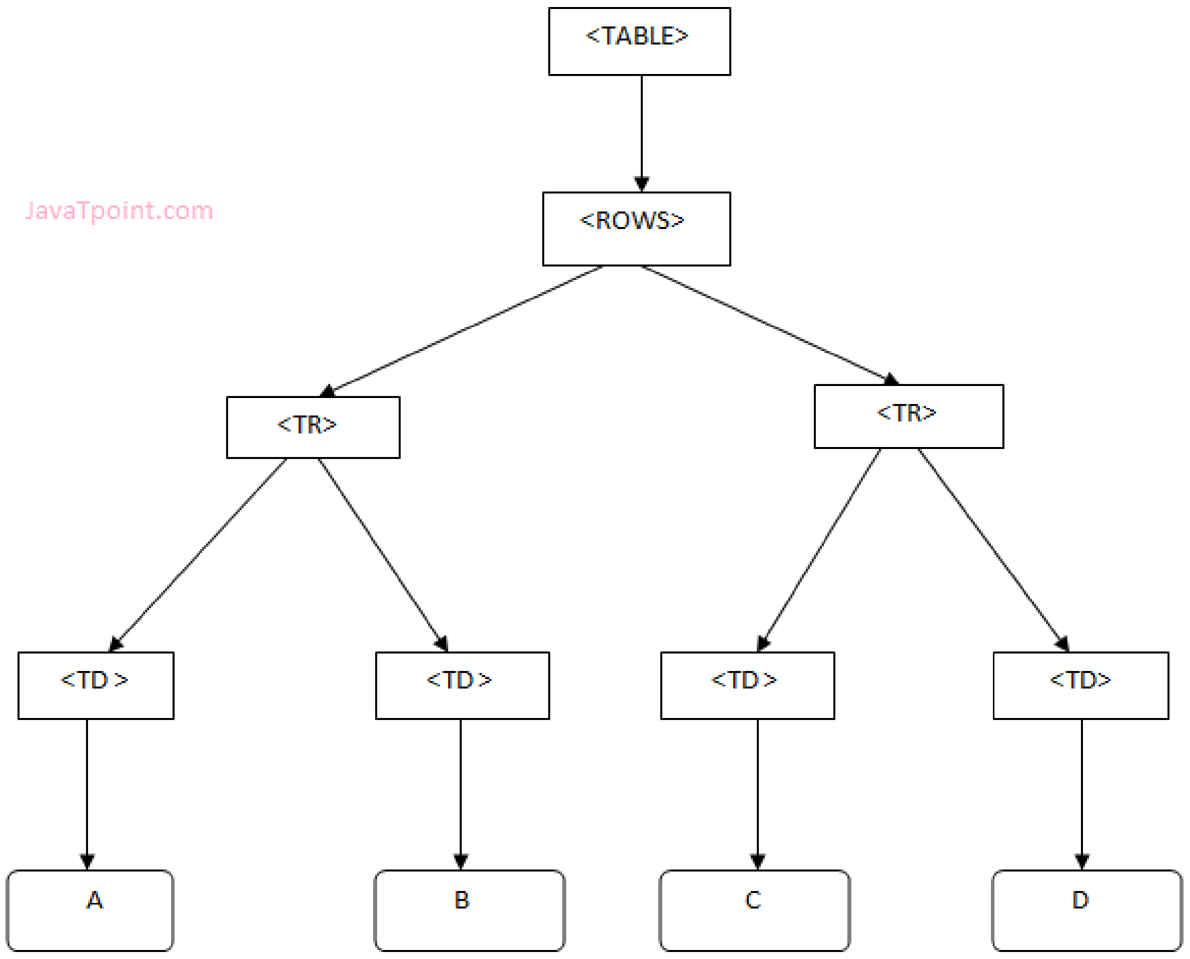
DOM is an acronym stands for Document Object Model. It defines a standard way to access and manipulate documents. The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.
As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The Document Object Model can be used with any programming language.
XML DOM defines a standard way to access and manipulate XML documents.

What does XML DOM

The XML DOM makes a tree-structure view for an XML document.
We can access all elements through the DOM tree.
We can modify or delete their content and also create new elements. The elements, their content (text and attributes) are all known as nodes.
For example, consider this table, taken from an HTML document:

1. <TABLE>
2. <ROWS>
3. <TR>
4. <TD>A</TD>
5. <TD>B</TD>
6. </TR>
7. <TR>
8. <TD>C</TD>
9. <TD>D</TD>
10. </TR>
11. </ROWS>
12. </TABLE>

The Document Object Model represents this table like this:



Difference between sax api and dom?

DOM XML Parser in Java

DOM parser is a tree-based API. A tree-based API is centered around a tree structure and therefore provides interfaces on components of a tree (which is a DOM document) such as **Document** interface,**Node** interface, **NodeList** interface, **Element** interface, **Attr** interface and so on.
A DOM parser creates a tree structure in memory from the input document and then waits for requests from client. A DOM parser always serves the client application with the **entire document no matter how much is actually needed** by the client. With DOM parser, method calls in client application have to be explicit and forms a kind of chained method calls.

SAX XML Parser in Java

SAX parser is a event-based API. Usually an event-based API provides interfaces on handlers. There are four handler interfaces, **ContentHandler** interface, **DTDHandler** interface, **EntityResolver** interface and **ErrorHandler** interface.

SAX parser **does not create any internal structure**. Instead, it takes the occurrences of components of an input document as events, and tells the client what it reads as it reads through the input document. SAX parser serves the client application always **only with pieces of the document at any given time**. With SAX parser, some custom methods are called [“**callback**” methods] when some certain events occur during parsing on xml document. These methods do not have to be called explicitly by the client, though we could call them explicitly.