Java XML Overview

Java XML is simply working with an XML document from a Java program. Imagine, we have a file "products.xml" where we have product details such as name, brand and price.

Now, we want to update prices for some products using Java programming. Before writing such java programs to access XML documents, we should know basics of XML.

What is XML?

XML stands for E**X**tensible **M**arkup **L**anguage. It is a text-based markup language which is used to store and transport data. It is self-descriptive and both human-readable and, machine-readable. Following are some notable points on XML –

- XML is a markup language.
- XML is a tag based language like HTML.
- XML tags are not predefined like HTML.
- You can define your own tags which is why it is called extensible language.
- XML tags are designed to be self-descriptive.
- XML is W3C Recommendation for data storage and data transfer.

XML Document

An XML document is the collection of elements that define data in a well structured and organized manner. An XML document has two sections, namely, **document prolog** and **document elements**.

Syntax

Following is the syntax of an XML document –

```
<?xml ?>
<root_element>
    <element></element>
    ...
</root_element>
```

Where,

- **<?xml ?>** is the XML declaration statement. If included, it must be kept in the first line.
- **<root_element>** is the root element and it is the parent of all other elements.
- **<element>** is the sub element of the root element.

Example

Following example shows Employee details with **<Employee>** as the root element and **<name>**, **<role>**, **<salary>** as sub elements. Data for each element is enclosed between opening and closing tags.

```
<?xml version="1.0" ?>
<Employee>
    <name>Kiran</name>
    <role>developer</role>
    <salary>25,000</salary>
</Employee>
```

Learn **Java** in-depth with real-world projects through our **Java certification course**. Enroll and become a certified expert to boost your career.

Elements in XML

An element is the building block of an XML document. It consists of an opening tag, content and a closing tag. In an xml document, there should always be a root element, inside which we can write many sub elements. Elements can also have any number of attributes inside them.

Syntax

Following is the syntax of an XML element –

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

Where,

- **<root>** is the root element of the XML document.

- **<child>** is the child element and its parent is the root element.
- **<subchild>** is the sub child and its parent is the child element.

Example

Let us see an example where DOB(date of birth) is further structured into date, month and year. Here, **<DOB>** is the root element and **<date>**, **<month>**, **<year>** are child elements.

```
<DOB>
    <date>27</date>
    <month>March</month>
    <year>2000</year>
</DOB>
```

Tags in XML

Tags in XML are self-explanatory and user defined. These are enclosed in less than (**<**) and greater than (**>**) symbols. XML is case sensitive and hence opening and closing tags should have same name.

Example

In the following example, we have written an address element with opening and closing tags.

```
<address>Hyderabad</address>
```

Now, let us see some **incorrect** ways of writing XML tags:

```
<Address></address>
<ADDRESS></address>
```

> *Since, XML is case sensitive unlike HTML, it throws the error:* ***Opening and ending tag mismatch****.*

Attributes in XML

Elements in XML can have attributes. Attributes are **name-value** pairs that provide further specific information about a particular element. An element can have any number of attributes.

Syntax

Following is the syntax for XML attributes –

```
<element_name attribute_name="value" >content</element_name>
```

Where,

- **element_name** is the name of the element.
- **attribute_name** is the name of the attribute.
- **value** is the value of the corresponding attribute.

Example

Now, let's look at the following example where we have four attributes, name, class, marks and DOB for the 'Student' element.

```
<Student name="Kiran" class="8" marks="50" DOB="27-03-2000"></Student>
```

Using sub elements to replace attributes

Instead of attributes, sub elements can also be used in elements to achieve the same purpose as of attributes. The same student example can also be written as follows:

```
<Student>
    <name>Kiran</name>
    <class>8</class>
    <marks>50</marks>
    <DOB>27-03-2000</DOB>
</Student>
```

> *It is always a best practice to use sub elements instead of attributes. Because, sub elements can further be extended whereas attributes cannot be*

extended.

In the above example, If we further want the date of birth as date, month and year then it can be done by using sub elements for DOB element as follows :

```
<Student>
    <name>Kiran</name>
    <class>8</class>
    <marks>50</marks>
    <DOB>
        <date>27</date>
        <month>03</month>
        <year>2000</year>
    </DOB>
</Student>
```

XML Declaration

XML declaration describes the basic format information such as version, encoding and standalone status about the entire XML document. If an XML declaration is included in the document, it must be written in the first line.

By default, if declaration is not mentioned, XML parser considers the document is in version 1.0

Syntax

Following is the syntax of XML declaration –

```
<?xml
    version="version_number"
    encoding="encoding_type"
    standalone="standalone_status"
?>
```

Where,

- XML declaration starts with the character sequence **<?xml** and ends with the character sequence **?>**

- **version** is the version number of the XML used
- **encoding** is the character encoding used for the content of XML document
- **standalone** is a boolean attribute whose default value is set to 'no'. This tells whether the XML document is standalone or uses information from external source to parse the document such as DTD(Document Type Definition). The default value is set to 'no'.

Example

Following example uses XML 1.0 version with encoding type UTF-16 and it is standalone.

```
<?xml
    version="1.0"
    encoding="UTF-16"
    standalone="yes"
?>
```

XML Comments

Comments in XML are used to explain the document's purpose and details. It is always a best practice to include comments in the document because it makes the task simpler to the one who is reading the document for the first time. XML follows the same syntax as of HTML.

Syntax

Following is the syntax for both single line and multi line XML comments –

```
<!-- comment here -->
```

Example

Let us say we have collected the information of departments in a college in the year 2015. These records might be changed over the years. So, mentioning this in the comments helps the one who is editing to know when these details have been collected.

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<!-- Following information is collected in the year 2015 -->
<college>
    <Department>
```

```
        <name>CSE</name>
        <code>CS</code>
        <faculty_strength>25</faculty_strength>
    </Department>
    <Department>
        <name>ECE</name>
        <code>EC</code>
        <faculty_strength>20</faculty_strength>
    </Department>
</college>
```

XML Namespaces

XML namespaces are used to resolve name conflicts in the XML document. When two or more XML fragments are added, then there is a chance that these XML code fragments might use some tags with same name. Then, this confuses the XML parser. To avoid these kind of name conflicts, XML Namespaces are used.

Example

Assume we have created an XML element holding the information about a coffee table –

```
<table>
    <shape>Oval</shape>
    <material>Wood</material>
    <seat_count>3</seat_count>
    <cost>15000</cost>
</table>
```

Suppose we have created another element which holds information about a dining table as –

```
<table>
    <shape>Rectangle</shape>
    <material>Marble</material>
    <seat_count>6</seat_count>
    <cost>25000</cost>
</table>
```

When the above two XML code fragments are added together (in a single file), there will be a name conflict. Though the name of both the elements is same the information

provided by them varies. There are two ways to resolve these name conflicts in XML. They are –

- Using Prefix
- Using namespace declaration

Using Prefix

We can differentiate the elements by adding prefix to them. To resolve the above name conflict, we can add the prefix 'c' to the element holding the info about the coffee table and similarly we can add the prefix 'd' for the other element (dining table).

Example

Let us take the same table example and try to resolve the name conflict using prefixes.

```
<!-- Coffee Table -->
<c:table>
    <shape>Oval</shape>
    <material>Wood</material>
    <seat_count>3</seat_count>
    <cost>15000</cost>
</table>

<!-- Dining Table -->
<d:table>
    <d:shape>Rectangle</d:shape>
    <d:material>Marble</d:material>
    <d:seat_count>6</d:seat_count>
    <d:cost>25000</d:cost>
</d:table>
```

Drawbacks of Using Prefixes

While using prefixes there still might be a chance where two elements have same prefix along with the name. In such cases the conflict prevails.

Suppose if we add another element providing info about a dressing table, to differentiate, we need to use the prefix 'd'. This again brings a conflict between dining table and dressing table. Hence, using prefix can solve the conflict to some extent but not completely.

Using "namespace" Declaration

XML namespace declaration is used to resolve name conflicts effectively. A new attribute named 'xmlns' is used.

Syntax

Following is the syntax for XML namespace −

```
<element-name xmlns:prefix="URI">
```

Where,

- **element-name**: Element name on which the namespace is used.
- **xmlns**: A compulsory keyword to declare namespace.
- **prefix**: Namespace prefix
- **URI**: Namespace identifier

Example

The following example uses XML namespace declaration for three table tags. Now, the conflict between dining table and dressing table is resolved by differentiating them in their namespace URI.

```
<!-- Coffee Table -->
<h:table xmlns:h="/coffee">
<c:table>
    <shape>Oval</shape>
    <material>Wood</material>
    <seat_count>3</seat_count>
    <cost>15000</cost>
</table>

<!-- Dining Table -->
<d:table xmlns:h="/dining">
    <d:shape>Rectangle</d:shape>
    <d:material>Marble</d:material>
    <d:seat_count>6</d:seat_count>
    <d:cost>25000</d:cost>
</d:table>
```

```
<!-- Dressing Table -->
<d:table xmlns:h="/dressing">
    <d:brand>Trevi Furniture</d:brand>
    <d:material>Engineered wood</d:material>
    <d:cost>15000</d:cost>
</d:table>
```