Chapter -9
Xml and Java

What is XML?

- XML stands for “eXtensible Markup Language”.
- XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable

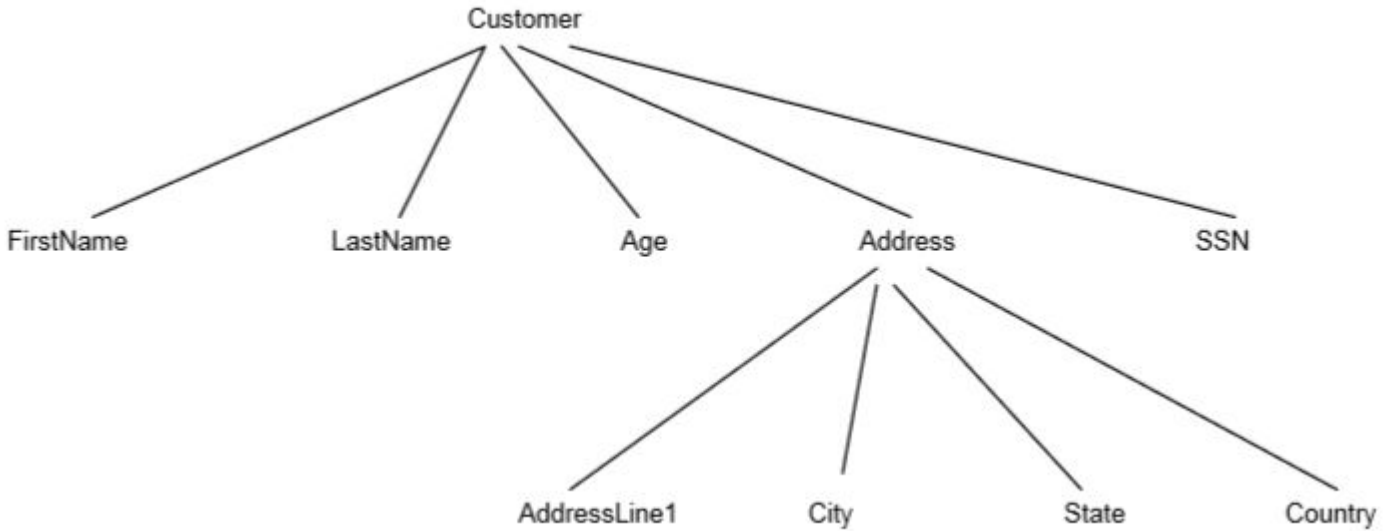
What is an XML Document?

- We all know that there are several ways for storing data and information. For instance, we use a text file to store the data line by line, we use database where tables are used to store data etc,.
- XML document is just another way for storing data and information but uses a tree like structure.
- An XML document/file consist of several tags that represent the data or information. These tags are also referred as nodes.
- Following is how a XML tag looks like:
    - <lastName>John</lastName>

- The good thing with XML is that, the tag names can be anything. However, there is just one rule that we need to strictly follow.
- **Rule: Every tag that is opened must be closed.**
- If the above rule is followed, then we say that the XML document as well formed.
- Example :

```
<?xml version="1.0"?>
<customer>
        <firstName>John</firstName>
        <lastName>Smith</lastName>
        <age>20</age>
        <ssn>23324</ssn>
        <address>
                <addressline1>Apt 2222</addressline1>
                <city>Columbus</city>
                <state>OH</state>
                <country>USA</country>
        </address>
</customer>
```

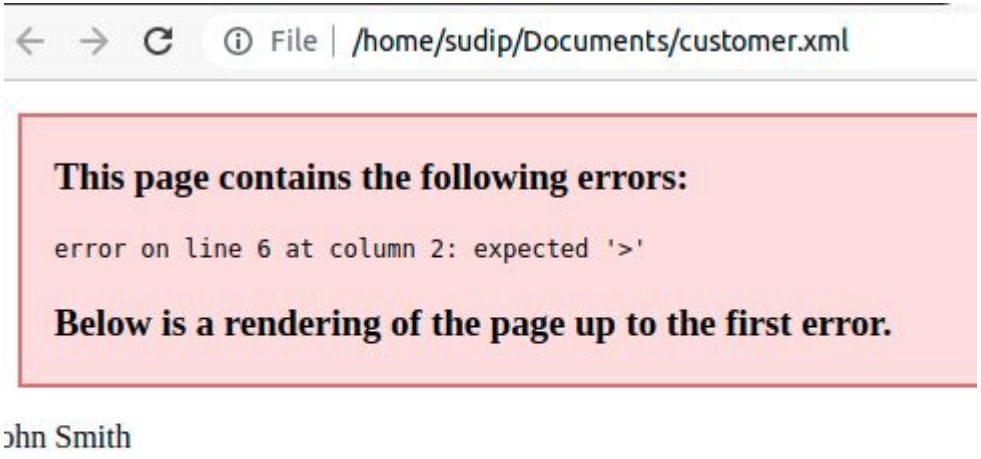
Following figure shows the tree representation of the above xml document:



- One simple way of verifying whether an XML document is well-formed or not is by opening the xml file using any browser like Internet Explorer.
- If the document is well-formed, you'll see the complete XML in the browser.
- If we fail to close some tag, the browser will display an error as shown below indicating that the XML document is not well formed. The error will also list the element name where the violation occurred.

Valid XML Document

Invalid XML Document



Why XML is important?

- The main reason why XML has tasted success is because it is 100% language independent and platform independent.
- It can be used to represent any complex data or information without any limitations on size with user defined tags.
- The only rule is that the document must be well formed. This is the reason why XML though very simple, is yet so powerful.
- Since enterprise applications involve complex data representation and processing, the flexibility that XML offers make it an ideal candidate for using in such situations.

XML Validation

- As I said before XML is used to represent data or information.
- Whenever we have any data, the first thing we need to do before processing the data is to verify whether the data is valid or not,
- So the question is how to validate the data represented by XML document?
- The simplest way is by using a DTD.

Document Type Definition (DTD)

- A DTD basically defines the following things:
  - The order of elements in the XML
  - The valid list child elements of a particular element
  - The list of valid attributes for a particular element
- All the validations defined using DTD are stored in a file with the extension ".dtd" and is referenced by the XML document.
- A DTD defines the validation rules for elements in an XML document using the ELEMENT declaration.
- Syntax
  - <!ELEMENT element name content-model>

The content-model basically tells the type of content the element can have.

There are four types of content models an element can have as listed below:

1. EMPTY
2. ANY
3. Children Only
4. Text with mixed children

1. EMPTY

- This indicates the element will not have any content
- DTD

  <!ELEMENT format EMPTY>

- XML

  <format></format> or <format/>

2. ANY:

This indicates that the element can have anything.

DTD

```
<!ELEMENT address ANY>
```

XML

```
<address> Here's a bit of sample text  </address>
```

3. Children :

- DTD

  <!ELEMENT account (accountNumber, accountBalance) >

- XML

  <account>
  <accountNumber>1234</accountNumber>
  <accountBalance>100.23</accountBalance>

  </account>

4. Text with mixed children :

- DTD

  <!ELEMENT description (#PCDATA|b|code)* >

- XML

<description>

This is a test

 description

The child elements can be

<code> in </code>

in any

order

</description>

Sample DTD

<!ELEMENT customer (firstName,lastName,age,ssn,address) >

<!ELEMENT firstName #PCDATA >

<!ELEMENT lastName #PCDATA >

<!ELEMENT age #PCDATA >

<!ELEMENT ssn #PCDATA >

<!ELEMENT address (addressLine1, city, state, country) >

<!ELEMENT addressLine1 #PCDATA >

<!ELEMENT city #PCDATA >

<!ELEMENT state #PCDATA >

<!ELEMENT country #PCDATA >

- The above DTD tells that the customer element should have the five child nodes namely
  - firstName,lastName,age,ssn,address in the same order.
- It then defines the content types of every element.
- All the text elements are defined as PCDATA which stands for Parsed Character DATA.
- We’ll see what parsing is in the next section.
- The address element in turn defines its child elements along with the order of the elements.

Though DTD provides a simple way of validating an XML document, it has certain limitations listed below.

It can only validate the order of the elements. It cannot validate the list of valid values an element can have.

Less flexible.

A DTD cannot be extended using inheritance.

No support for validating numeric and boolean data

To overcome the above limitations, a new validation scheme is created which is called as XML Schema. Let’s see what this is and how we can use xml schema to validate xml documents in a better way.

XML Schema

- Unlike a DTD, an XML Schema is itself an XML document with elements, attributes etc.
- XML Schemas overcame all the limitations of DTD and had now become the standard for validating XML documents.
- The good thing about XML Schema is that it is closely associated with Object Oriented data models.
- One of the major concerns with DTD is the lack of support of various data types.
- There is no way that using a DTD we can validate the type of data an element can have. This is where schema comes in real handy.
- It contains several built in primitive data types such as string, integer, float etc for validating the data.
- XML Schema can also be used to build complex data types using simple data types. First, let’s look at the important simple data types listed in the following table.

Data Type	Description
string	Used for text data
boolean	Used for boolean data (True/False)
Float	Used for 32 bit decimal numbers
Double	Used for 64 bit decimal numbers

- Using the above data types, an element named score will be defined as shown below:
  - <xsd:element name=”score” type=”xsd:int”/>
- Following are some of the examples using different data types:
  - <xsd:element name=”firstName” type=”xsd:string” />
  - <xsd:element name=”expiration” type=”xsd:date”/>
  - <xsd:element name=”financialIndicator” type=”xsd:boolean/>
- All the above defines data types for simple elements.
- However, using XML schema we can build complex data structures from simple data structures.
- This is where XML schema exhibits its true power. It allows us build complex data structures by defining the order of elements, valid values for different elements and so on.

XML Parsing

- Though XML provides infinite flexibility to represent data of any complexity, it’s of no use if we cannot read the data back from XML file.
- This is where parsing an XML document comes into picture. Parsing an XML document is nothing but reading the data back from the document.
- The application that parses an XML document is called an XML parser.
- The purpose of an XML parser is to make some interfaces available to an application so that it can modify and read the contents of an XML document.
- The generation of these interfaces is based on two XML standards namely SAX and DOM.

SAX

- SAX is abbreviated for Simple API for XML.
- SAX parsing is based on event model in which sequences of events are generated for each and every tag in the XML document.
- Based on the type of events, the application can take appropriate action. The various events a SAX based parser generates are:
- Start of Document
- Start of Tag
- End Of Tag
- End of Document etc.
- SAX based parsing can be used only for reading the data from XML and not for modifying the contents.
- Moreover, SAX parsing is more efficient in situations where the XML document is huge and we only want to extract a small piece of data from it.

DOM

- DOM stands for Document Object Model.
- In this model, an XML document is represented as a tree of nodes.
- A parser based on this model, can traverse the through the nodes to read the data or modify the data by removing the nodes.
- In DOM parsing, the entire XML must be loaded into the memory before reading or modifying the document.
- Because of this reason, DOM based parsing should be used only when the XML document is small enough not to cause any memory issues.
- Any XML parser that is constructed will be based on either SAX or DOM model.
- There are several XML parsers available in the market for free of cost. Most notable ones are the parsers from Apache, BEA and Sun Microsystems.
- In this chapter, we use the parser from Apache called as Xerces parser to parse the XML documents.

Difference between SAX and DOM

- SAX is based on event model.
- SAX will never load the XML into memory.
- SAX parsing is used for reading XML documents only and cannot be used to modify its contents.
- SAX is used for reading small portion of information from large XML documents.

- DOM is tree based.
- DOM will load the entire XML into memory before parsing.
- DOM is used to read and modify XML data.
- DOM is usually used with small XML documents.