**JAX P**

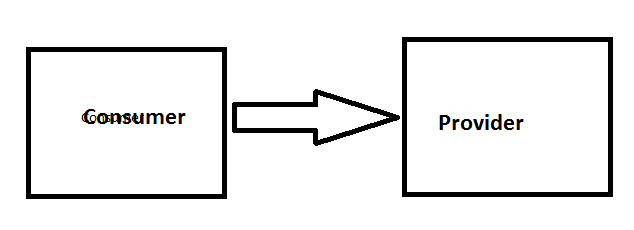
**Use Case:**

**For Sim Activation:**

For example, if we want to get a Airtel new connection. We will fill the form and the POC of the Airtel will fill that application details in an excel sheet or User interface. In the backend an XML fie is generated. It is send to central office i.e Begumpet office.

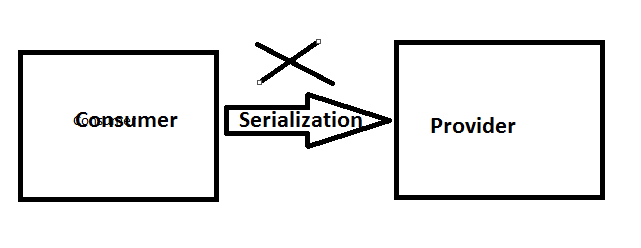
So, in Business to Business appications, XML is mandatory.

**In webservices:**



Java applications i.e. Consumer here should hold the data in terms of an object. And the object shold be sent to Provider.

**Can we use Serialization to transfer the data?**

We cannot use serialization, because serialization is the concept of only Java. So, provider should be Java appication. Here, interoperability is lost in serialization. 

**Then how can we transfer the data between consumer and provider ?**

Convert the object of data into a XML equalent reperesentation of object. If we send the XML, then if there is any change the XML then the XML logic for reading the data should be modified.

So, provider should not read the directy from an XML. So, we have to send the XML as an object.

**JAX P :**  
Java API for XML Processing, this is not meant for converting XML to object. It just can read or write the data from an XML.

Initially huge adoption of XML is not there. But later due to the heavy increase in the use of XML, slowly API to read the XML use started increasing.

Initially Java has not provided any support for XML reading API. Java community people requested Sun Micro systems to provide the API. But Sun has not shown interest in supporting the XML reading API. Java people now has no support to reading the contents of an XML. So, third party vendors came into picture.

To read the contenets of an file we have a IO Streams. But it is very difficult for normal Java programmers. So, vendors gave few classes and methods, to read the XML.

**Third party libraries:**

SAX4J

DOM4J

XML4J

**What is standard API ? What are third party library ? Why Sun has not provided Standard API ?**

Java being an Open Source platform, it is always late in the market. Because open source has some financial issues.

For example, if Java immediatly started investing in cloud, lets assume cloud is not successful then money is wasted.For commercial vendors, fund ll be coming from partner itself.

**Standard API:**

API - API is always partial.

Interfaces and abstract classes will only be there. The concrete implementation of interfaces/abstract classes are not present in API. So, APIs are partial.

**API vs Standard API:**

API is bunch of libraries. API is a generic term. Third party provided library is also called as API.

**Example**: log4J

APIs provided from Sun are ony called as Standard APIs.

**Example**: JAX P

**Java follows 3 stages of software development for deveoping a Standard API: [SAI]**

As we know Java is Open source technoogy stack.

*a. Specification*

*b. API*

*c. Implementation*

**Specification :**

It is just like a white paper or open paper. Anyone can write a specification. It may be diagramatic or any representaions. We can just keep our idea on paper here.

We can call this as **Draft Specification document**. People who has some idea can give there ideas also. We can finally make this specification meaningful.

It is blue print or a representaion of something. Its not needed to represent in terms of techinical.

**Example**: Building plan. How many rooms will be there, and where it will be there only will be there. Here use a 12 inch brick is used or 9mm rod use it etc will not be there in specification.

For XML reading, should we read in sequential way or random way, will be identified as a part of XML Specification.

**API :**

For building a house we hire architect will be called. He will guide us how to build a house. How much inch brick is needed and how much rod is needed etc.

API will specify the structural representations only.

**Implementation :**

Actual implementaion i.e concrete representation is specifed here.

**Not only for financial issues, open source also has lack of support too.**

**For example**:

.Net is not an open source. We have to get license from Microsoft. If we are working on .Net, we found a bug. Then Microsoft has to definatly address the bug. As we are buying the license, they have to solve issues 24/7.

If we have some issue in Java, as it is open source one, it is not that active in supporting to sove the bug we got.

So, enterprise/business applications are not preferred to use Java. As if issues comes no one will be there to solve the bugs. So, .Net was very famous initially along with Java.

**Lot of open source patforms like PHP, Ruby On Rails etc are there ? But why Java is famous than other language ?**

**Or**

**Why three stages are needed for standard API ?**

**For Example:**

If Sun provided a library called java.xml.parsingAPI library. Directly Java has provided libraries, without using any of 3 stages formula.

While developing the applications in Java, we will provide jxmlReader class. It has a method called jReadElement() method.

*class JXMLReader{  
 jReadElement(Element Name){  
 }  
}*

If we are using the above, and we have seen a bug. Then as Sun is a open source, it will not concentrate that much on solving the bug.

***So, don't go for Java is the ony solution left ?? "No"***

So, Java said that I have provided Java Standard API. So, various vendors can provide the concrete implementaions for that interface and abstrace classes.

**For Example:**

**API** *interface jReader(){  
 }  
interface jElement{  
 ReadElement(String ){  
 String getTagName();  
 String getTagVaue();   
 }  
}*

**Java:***class jSunReaderimpl implements jReader{   
}*

*class jSunElement implements jElement{  
 Reader.ReadElement(OrderItems);  
 Element.getTagName();  
 Element.getTagValue();*

*}*

*So,* we can start reading it. We are just using the implementation class here. If we found a bug in API then we can just continue switching to other vendor.

**Apache:***class jFastReaderimpl implements jReader{   
}*

*class jSunElement implements jElement{  
 Reader.ReadElement(OrderItems);  
 Element.getTagName();  
 Element.getTagValue();*

*}*

So, here we are not tightly coupling. So we can easily move to other vendors using this. As vendors provide implementations for these vendors. Few implementations of these are commercial implementaions too, it gives 24/7 support. So we can work like .Net in Java. We can get few memory saves also, if we select few vendors.

For every API, Java will provide open source implementation of its own. First Java provides Sun Implementaion or Reference Implementaions.

JAXP is as API, so, Sun itself should provide one reference implementaion.

**Why vendors will provide implementaions for the API provided ?**

So SUN went into the market and started promoting their APIs. They requested vendors to provide implementaions.

**Then vendors asked what is the use for them ?**

Then SUN said, I will provide you my brand. I will treat you as a partnered vendor of Sun i.e SUN PARTNERED VENDOR. So, that small company will be recognized as SUN PARTNER. Then people will start to treat you with TRUST.

**Then vendors asked how can we get complete information API ?**

"From initial thought onwards we will involve you in the specifications. Then you can contribute to that idea better" - said SUN.

So, Sun has to provide a reference implementation and then the vendors will provide there implementation.

**As SUN is open source vendors, how will it get revenues ?**

**Certifications** - SCJP, SCWCD, ...

**Support and Services** - Hire SUN staff as architects of SUN. There will be annual support packages too.

**Sun Trainers -** Costly

**Note:**

***API:***

*interface jReader(){  
 }*

*ReadElement(String ){  
 String getTagName();  
 String getTagVaue();*

*}*

***Sun:***

*class jSunReaderimpl implements jReader{   
}*

*class jSunElement implements jElement{  
 Reader.ReadElement(OrderItems);  
 Element.getTagName();  
 Element.getTagValue();*

*}*

***Working:***

class ReadXML{  
pubic void process(file("PurchaseOrder")){  
 JReader reader= new JReader;  
 // Wrong, we need to create from implementation  
 JReader reader= new jSunReaderimpl;// Right  
 }  
}

If we want to modify this to Apache implementation then again code change  
is needed.

**Apache:**

*class jFastReaderimpl implements jReader{   
}*

*class jSunElement implements jElement{  
 Reader.ReadElement(OrderItems);  
 Element.getTagName();  
 Element.getTagValue();*

*}*

***Working:***

class ReadXML{  
pubic void process(file("PurchaseOrder")){  
 JReader reader= new JReader;  
 ***// Wrong, we need to create from implementation*** JReader reader= new jFastReaderimpl;// Right  
 }  
 }

It should not be loosely coupled. It should be loosely coupled.

For this interface SUN provided implementation is jSunReaderimpl and Apache provided implementation is jFastReaderimpl.

We cant remember al the class names. So, there should be some way to remember different vendor implementations.

**SUN:**

*class jSunReaderimpl implements jReader{   
}*

*class jSunElement implements jElement{  
 Reader.ReadElement(OrderItems);  
 Element.getTagName();  
 Element.getTagValue();   
}*

**Working:**

*class POAccessories{  
 public void POManager(File XMLFile){ jReader reader=new jReader();  
 //Wrong, Should use implementation of API  
 jReader reader=new jSunReaderImpl();   
 jElement element = reader.ReadElement(ItemCode);*

*}   
}*

Here to switch to different vendors, we need to change the code. It is difficult as above code is tightly coupled.

**From above, how can we know the class names of different vendors ?**

So, we should find some way easily to move from one vendor to other vendor.

Factory comes into picture.

**Factory:**

Factories are the classes who knows how to create the object of another classes.

Creating an object would be complicated, so we have to avoid such logic in the code at multiple places.

jReader is an interface for which I have implementation i.e. *jSunReaderImpl*, my class wants to use *jSunReaderImpl*, but I dont know the impementation class, but I want to use the object of that class. Go for Factory Method. Factory takes care of identifying the implementations and instantiates and returns.

So, we dont need to know the actual implementation cass name. So, we can say it is loosely coupled.

**Sun:**

*class jReaderSunFactory{  
 jReader CreatejReader(){  
 //Return sun provided implementaion class object  
 return jSunReaderimpl; -- Object  
 }  
}*

**Internally for SUN:**

*class jSunReaderimpl implements jReader{   
}*

*class jSunElement implements jElement{  
 Reader.ReadElement(OrderItems);  
 Element.getTagName();  
 Element.getTagValue();   
}*

If vendor changes the implementaion class names, then we dont even care. As we are just referring to the Factory method. So, class changes then we dont need to change the code at-all. So, it is loosly coupled.

Here, CreatejReader internally is constructed in such a way that, if we call that method, then automatically in has to get the details of jReader and jElement interface also. All must be linked internally.

So, in JDBC, connection, statement and Resutset are all interconnected.

**Apache:**

*class jReaderFastFactory{  
 jReader CreatejReader(){  
 //Return sun provided implementaion class object  
 return jFastReaderimpl; -- Object  
 }  
}*

**Internally for Apache:**

*class jFastReaderimpl implements jReader{   
}*

*class jSunElement implements jElement{  
 Reader.ReadElement(OrderItems);  
 Element.getTagName();  
 Element.getTagValue();   
}*

Here we are referring to the concrete factory methods. So, again whole code must be modified.

So, we use Factory of Factory methods.

*class jReaderSunReaderFactoryImpl implements jSunReaderimpl {*

*}*

*class jReaderFastReaderFactoryImpl implements jFastReaderimpl {*

*}*

Internally they have same methods only.

**Now [ Factory of Classes ]**

***Sun:***

*jReaderFactory jFactory=new jReaderSunReaderFactoryImpl ();  
jReader reader=jFactory.createjReader();*

***Apache:***

*jReaderFactory jFactory=new jReaderFastReaderFactoryImpl ();  
jReader reader=jFactory.createjReader();*

But the above also has a problem. Here also a we need to change the code during switching to other vendor.

So again a factory is needed i.e Super Factory method. It goes on.

**Abstract Super class factory :**

*public jReaderFactory{  
 createJReaderFactory()  
}*

*class jReaderFactoryFactory(){  
 CreatejReaderFactory("SUN"); or  
 CreatejReaderFactory("APACHE"); or  
}*

**Instead of writing the Factory Factory classes, we can write above as :**

***Sun:***

*jReaderFactory jFactory=new jReaderSunReaderFactoryImpl ();  
jReader reader=jFactory.createjReader();*

***Apache:***

*jReaderFactory jFactory=new jReaderFastReaderFactoryImpl ();  
jReader reader=jFactory.createjReader();*

These can be converted to [Abstract Factory Class],

*jReaderFactory.createReader("SUN");*

*or*

*jReaderFactory.createReader("APACHE");*

All these are called as ***Abstract Factory Design Pattern.***

**How to hard code the SUN or APACHE ?**

* If I have a valaue, I want to pass it as input to a method, pass it to a parameters to a method.
* If I have a valaue, I want to pass it as input to all the methods of the class, pass it as an Attribute.
* If I have a valaue, I want to pass it as input to all the objects of a class, pass it as a static variable.
* If I have a valaue, I want to pass it as a global to all the classes to appications running on same JVM then pass it as an SYSTEM variable.

**SYSTEM PROPERTY is the global variable that is there in Java.**

*String implementaion=null;*

*implementation=SYSTEM.getProperty("VENDOR\_NAME.NAME");*

While running the program, we should pass :

*-d VENDOR.NAME="SUN";*

**If we use Static variable also it soves the problem , then why shold we take it as a SYSTEM Property ?**

Class Loaders.

Implementation vendor jar will be loaded by class loader. All the applications has to use the same jar. Because it cannot be overridden.

Jar we are putting at the CLASS\_PATH, that will be loaded by the CLASS LOADERS, so that the classes that are loaded by the CLASS LOADER will be applicable to all the applications that are running under same JVM under the same class loader. It should be GLOBAL to the class loader scope, It shouldnot be gobal to the application level scope. So, we are passing it as a SYSTEM PROPERTY.

SYSTEM PROPERTY is a JVM level input data.

**Why Factory classes ?**

**Basic principle adopted by Java for API's ?**

**What is Abstract Factory Design Pattern ?**

**Problems in working with third party library ?**

When we are working with 3rd party library we should properly take care of the documentations. Else the limitations may not be understood.

So, we should got for JAXP API.

**JAX APIs implementation present in the market:**

*XERCES2 - Default implementation. - Provided byApache  
CRIMSON  
ORACLE V2 Parser  
XML4J*

**Versions:**

From JDK 1.4, JAX P is provided - But no direct support.

JAXP API.jar, JAXP\_API \_impl.jar will not be there inside JDK.

From JDK 1.5, we have direct support for working with JAXP.

JAXP API.jar, JAXP\_API \_impl.jar(XERCES implementation) will be there inside JDK directly.

**Note:**

In general, if we give a book to few guys. Each guy will have his own style of reading a book. Some starts with prefix, some with first unit etc. Similarly, **In general, in Universe, there are 2 ways of reading the contents of an XML: [Universal Access Methodology ] :**

**SAX** *- Simple Access for XML API*

**DOM** *- Document Object Model*

**SAX:**

Simple Access for XML API.

It allows to read the contents of an XML in a **sequential access model**. SAX will start reading the contents of an XML by placing the pointer at the top of the XML before the root. It move the pointer from top to the bottom from one element to the other.

SAX consumes less amount of memory while reading the contents of an XML.

For example, at any point of time the pointer will read only one portion at a time. So we can say it consumes less memory.

SAX uses **Event based processing model/mechanism**, in reading the contents of an XML.

**Event based processing model:**

AWT and Swings in Core Java works on this model.

There are four actors in this mechanism:

*a. Source  
b. Event  
c. Listener and  
d. Handler*

**General:**

Lets take a button. If we click on a button, then it is called as an Event.   
Who knows whether I clicked the button or not ? i.e button will know. So, button should know the logic to store the data in the data base. If we clicked on DELETE button, will it also store the data in the database ? No. So, buttons should not store or delete. So, button is not tightly coupled with the actions.

So, button should tell that someone has pressed me. So that some one will be able to listen what type of action has been happen on the button. Then others will perfor the action.

**Event**: Here, an action that has been taken place on a button.

**Source**: The component that triggers the action i.e pressing the button.

**Why source will generate the event ?**

When someone has clicked the button, then button cant do anything. So, it should tell someone that event is triggered.

Event will not perform the action. It just carries the action which is performed on it.

*class button{ --> Save  
 public void onCick(){  
 }  
}*

*class button{ --> Delete  
 public void onCick(){  
 }  
}*

... etc.

So, we have to create two object on the same button i.e. save and delete.

*class button{ -->* Generic component.  
 *public void onCick(){  
 }  
 }*

So, button should not do the action by its own. Then it shoud tell someone that it has been clicked.

Event is an informational object that carries, the action that carries on the source.

**It has 2 parts :**

a. Type of action [single click or double click etc...]

b. Source [which button has been clicked, save or delete or etc..]

*Event event = new Event();  
event.setAction=clicked;  
event.setSource=this;* //Then publish the event.

Event is a object that encapsulates the information about the action on the source and on which source the action has been performed. Event dont know how to handle the action.

Neither the source or event knows what action to perform. Programmer knows it.

**Handler :** The programmer has to write the logic to perform in a seperate class which is known as Handler.

A source can trigger multiple events like save, delete, doube click, single click..etc.

So, handler will contain multiple different methods(routines) for each event.

Logic that has been written in the handler class is based on the type of the event generated by the source. Handler class needs both action and the source. As source is carried by events, always these methods will contain the source as an input. So, in JavaScript method we pass events.

**Listeners :** It acts as a mediator between the source and the handler. Listener will places an ear on the source. Whenever a source has produced an event, then the listener will check the type of the event and then listener will identify the source. Handler will register with the listener.

Listener is a kind of logical entry with source and handler. From which source the event is called, then the listener calls the corresponding handler.

If the same event is called again. Then we dont need to duplicate the logic, we just need to make an entry in the source to same handler. Because neither the handler or the source is taking to each other directly. So we can reuse the handler or we can bind the multiple sources to the same handler.

**Now if we click on a button, we should not save it.**

*Then, change the code in the BUTTON ?* ***NO****.*

*Then, change the code in the EVENT ?* ***NO****.*

*Then, change the code in the HANDLER ?* ***NO****.*

Nothing will be done, just we need to remove the entry from the listener. Because everyone is loosly coupled, that is the purpose of event based processing mechanism. Same button can be binded to any action, neither the source or event of handler is modified. This is the loosly coupled mechanism.

==========================================

SAX will allows to read XML in sequential way and uses event based processing mechanism.

**Why SAX adopted the event based mechanism ?**

**In SAX, who is source, event, handler or listener ?**

**Source :**

A person who is capable of generating the event is Source. In SAX XML is the source.

**Events :**

As we know the SAX places a pointer at the top of the XML and moves forward.

Start tag , End tag, Data... all these are considered as an Events.

**Handler:**

The action to be performed should be written in the handler component.

If start tag is encountered, START TAG method is called.  
 If end tag is encountered, END TAG method is called.  
 If data is encountered, DATA method is called.

**Listener:**

The intermediator which takes source and handler as input, will identifies the type of event by incrementing the pointer and calls the corresponding methods in the handler class.

Here, **listener will acts as a parser.** We should pass the source and the handler.

**===================================================**

**SAX in JAX P:**

When ever we pass XML as an input to SAX Engine, this will place the pointer to the top of the XML.

Based on the content it is pointing events are being triggered.

*a. START DOCUMNET  
b. START ELEMENT  
c. CHARACTERS  
d. END ELEMENT  
e. END DOCUMNET*

Like this many different types of events. If an API can generate more events only the API will be strong.

We cannot call any class and call it has a handler class. Because we are not going to call the methods, parser is going to call it. How would the parser will know which methods are ther and for which event which method has to be triggered ?

So, when we are writing this handler class, we should write this class, by implementing or extending from the standard interface or by a class.

We need to write this class extending from a default handler. Which is not a interface or abstract class. It is a concrete class, with a dummy methods. No logic will be there inside it. We can write the logic only for the event we need.

***public void startDocument() {  
 }***

***public void startElement(String uri, String localName, String qname, Attributes attr) {  
}***

***public void characters(char[] ch, int start, int length) {*** *// Whole xml is passed as a character array.  
 //Start/Offset - Position where the data is encountered.* ***}***

***public void endElement(String uri, String localName, String qname) {  
 }***

***public void endDocument() {  
 }***

If we dont want to do anything, then dont even call that event.

To the mediator/parser we should, pass the XML and object of the Handler.

Object because parser is not going to instantiate this class, it is going to call the methods. To call the methods, we need to have the object of the class, we dont need the class. So, we dont need to take care of initializing and instantiating the handler. We shoud just pass the object of the handler.

SAX Parser, will reads the contents of the XML and triggers the events and calls the methods of the handler.

SAX Parser is coming from JAX P API. As API is partial it contains interfaces and abstract class. SAX Parser will also be an interface / abstract class.

To call the methods, we need to create the object of the SAX Parser. We can create object of a SAX Pareser in following way:

*SAXParser parser = new SAXParser(); //* ***Wrong***

As it is interface/abstract class, we cannot create the object of the SAXParser. We need to find the implementation class that is provided by the implementation class which is provided by the implementation vendor for that interface or abstract class.

So, we need to go to SAXParserFactory. It is a factory which knows what is the corresponding implementation class provided by the vendor for the interface or abstract class.

SAXParserFactory is coming from the API... So we use the Abstract Factory Design Pattern.

*SAXParserFactory --> Abstract Factory class.*

***SAXParserFactory parser = SAXParserFactory.newInstance();***

When newInstance is called, it will check for SYSTEM PROPERTY. In system property we can specify which impementation we are using(CRIMSON, XERCES,...etc). If nothing is set,then the default implemetation i.e. XERCES is called, when we are working with JDK 1.5 or above. But in JDK 1.4 there is no direct support is there. So, no default implementation in 1.4.

*Parse can you please parse XML(absolute path of XML), Handler object.*

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

***SAXParser parser = factory.newSAXParser();***

***parser.parse(new File("D:\\po.xml"), new POHandler());***

**Alt SHIFT S V**

=====================================================================

**DOM:**

Another universal access methodology.

DOM will read the contents of an XML in random manner. That is, there is an API which will help us to access specifid element in a random manner.

Trees will not support in random access. Cost of traversal in trees are high. Because we dont know how many childs are there in the nodes.Binary trees came into picture. Each node has 2 child nodes here. Problem is that binary tree is costly.Then NRI Binary tree came in to picture....

DOM reads the contents of XML in a hierarchial process model. Anything that is written in terms of parent and child, is called as a Hierarchial model. So, DOM is going to represent the data in the tree representation. It represents the contents of the tree as nodes of the tree. Each node has a relation with the others. As tree contains the contents of an XML, it is called as **In-Memory Content Tree.** Once the tree is constructed, the pointer is placed at the top of the tree. One can move the pointer to different nodes of the tree, by using the **traversal mechanisms.**

**Traversal mechanisms has different methods:** *Next Node  
Previous node.  
First child.  
Second child   
Any element data etc...*

To read the contents of an XML, the XML should be represented in a hierarchial model, then only the methods that are there in the DOM will allows us to traverse. We have to place that in an object form because anything that is there in the JVM memory is represented in terms of object. In which object we should store or represent the structure of a JVM as a tree ? **DOCUMENT OBJECT.**

So, Document Object represents the tree representation of an XML, having the relation in a hierarchial model. Once the tree has been constructed, it places a pointer at the top of the Tree. The pointer is been given as a reference to us in the document. So, we have top node of the tree now. We use traversal API to traverse. Unless until we create a document object, we cannot traverse.

**How to create a document object ?**

As **DOCUMENT** is a class coming from JAX P API, it is interface or abstract class. We dont know the implementtaion of it, So we have to go to the Factory. So we should use Document Factory?

DocumentFactory document= new DOCUMENT(); //WRONG, because

When comes to the DOM it not only reads that XML, it can also create a new XML also. To support such a kind of API, DOM has provided one object called as **DOCUMENT BUILDER.**

A Builder has an API for reading the XML and also writing the new XML... Check Builder Design Pattern.

Again Document builder is coming from API. So, it must be interface or abstract class, we dont know how to create an object for it. So, we have to go to DocumnetBuiderFactory.

*DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance(); DocumentBuilder db = dbf.newDocumentBuilder();  
 Document doc = db.parse(new File("games.xml"));*

Now, doc object has a tree structure of XML, and a pointer is placed at the top of the tree.

There will be one root node, that is DOCUMENT node.

*DOCUMENT --> Purcahse-order--> Item --> Item-code--> etc...*

All the nodes are same, as all are represented in a node interface. But based on the characteristice of the node, we can distinguish between one node to the another node.

**There are 4 types are nodes are there:**

*a. DOCUMENT NODE  
b. ELEMENT NODE   
c. TEXT NODE  
d. ATTRIBUTE NODE*

For DOCUMENT node to the ELEMENT(purchase-order) node, we can call as get First child. That is purchase-oder node.

*Node node=doc.getFirstChild();  
 or   
 doc.getFirstChild().getFirstChild().getName();  
 or  
 doc.getFirstChild().getFirstChild().getFirstChild().getValue();*

Element node will have name and text node will have value.

**What is Builder Design Pattern ?**

Java SAX Parser – Overview

SAX (the Simple API for XML) is an event-based parser for xml documents. Unlike a DOM parser, a SAX parser creates no parse tree. SAX is a streaming interface for XML, which means that applications using SAX receive event notifications about the XML document being processed an element, and attribute, at a time in sequential order starting at the top of the document, and ending with the closing of the ROOT element.

* Reads an XML document from top to bottom, recognizing the tokens that make up a well-formed XML document
* Tokens are processed in the same order that they appear in the document
* Reports the application program the nature of tokens that the parser has encountered as they occur
* The application program provides an "event" handler that must be registered with the parser
* As the tokens are identified, callback methods in the handler are invoked with the relevant information

When to use?

You should use a SAX parser when:

* You can process the XML document in a linear fashion from the top down
* The document is not deeply nested
* You are processing a very large XML document whose DOM tree would consume too much memory. Typical DOM implementations use ten bytes of memory to represent one byte of XML
* The problem to be solved involves only part of the XML document
* Data is available as soon as it is seen by the parser, so SAX works well for an XML document that arrives over a stream

Disadvantages of SAX

* We have no random access to an XML document since it is processed in a forward-only manner
* If you need to keep track of data the parser has seen or change the order of items, you must write the code and store the data on your own
* Java SAX XML parser stands for Simple API for XML (SAX) parser.
* SAX is an event-driven, serial-access mechanism for accessing XML documents.
* This mechanism is frequently used to transmit and receive XML documents.
* SAX is a state independent processing, where the handling of an element does not depend on the other elements. StAX is state dependent processing.
* SAX is an event-driven model. When using the SAX parser we provide the callback methods, and the parser invokes them as it reads the XML data.
* In SAX we cannot go back to an earlier part of the document and we can only process element by element, one by one from the start to the end.

When to Use SAX

SAX is fast and efficient and it is useful for state-independent filtering. SAX parser calls a method when an element tag is encountered and calls a different method when text is found.

SAX requires much less memory than DOM since SAX does not create an internal tree structure of the XML data, as a DOM does.

Parsing an XML File Using SAX

In the following we are going to see a demo application which output all SAX events. It is extending DefaultHandler from package org.xml.sax.helpers as follows.

**public** **class** Main **extends** DefaultHandler {

The following code sets up the parser and gets it started:

SAXParserFactory spf = SAXParserFactory.newInstance();

spf.setNamespaceAware(true);

spf.setValidating(true);

parser = spf.newSAXParser();

parser.parse(file, this);

These lines of code create a SAXParserFactory instance, as determined by the setting of the javax.xml.parsers.SAXParserFactory system property.

The factory is set up to support XML namespaces by setting setNamespaceAware to true, and then a SAXParser instance is obtained from the factory by newSAXParser() method.

And then it handles the start-document and end-document events:

**public** void startDocument() {

System.out.println(**"Start document: "**);

}

**public** void endDocument() {

System.out.println(**"End document: "**);

}

After that it uses the System.out.println to print message once the method is called by the parser.

When a start tag or end tag is encountered, the name of the tag is passed as a String to the startElement or the endElement method, as appropriate.

When a start tag is encountered, any attributes it defines are passed in an Attributes list.

**public** void startElement(String uri, String localName, String qname, Attributes attr) {

System.out.println(**"Start element: local name: "** + localName + **" qname: "** + qname + **" uri: "**

+ uri);

}

Characters within the element are passed as an array of characters, along with the number of characters and an offset into the array that points to the first character.

**public** void characters(char[] ch, int start, int length) {

System.out.println(**"Characters: "** + **new** String(ch, start, length));

}

The complete code.

**import** java.io.File;

**import** java.io.IOException;

//from www .j a v a 2 s .c o m

**import** javax.xml.parsers.ParserConfigurationException;

**import** javax.xml.parsers.SAXParser;

**import** javax.xml.parsers.SAXParserFactory;

**import** org.xml.sax.Attributes;

**import** org.xml.sax.SAXException;

**import** org.xml.sax.helpers.DefaultHandler;

**public** **class** Main **extends** DefaultHandler {

**private** **static** Main handler = null;

**private** SAXParser parser = null;

**public** **static** **void** main(String args[]) {

**if** (args.length == 0) {

System.out.println(**"No file to process. Usage is:"** + **"\njava TrySAX <filename>"**);

**return**;

}

**File** xmlFile = **new** **File**(args[0]);

handler = **new** Main();

handler.process(xmlFile);

}

**private** **void** process(**File** file) {

SAXParserFactory spf = SAXParserFactory.newInstance();

spf.setNamespaceAware(true);

spf.setValidating(true);

System.out.println(**"Parser will "** + (spf.isNamespaceAware() ? **""** : **"not "**)

+ **"be namespace aware"**);

System.out.println(**"Parser will "** + (spf.isValidating() ? **""** : **"not "**) + **"validate XML"**);

**try** {

parser = spf.newSAXParser();

System.out.println(**"Parser object is: "** + parser);

} **catch** (SAXException e) {

e.printStackTrace(System.err);

System.exit(1);

} **catch** (ParserConfigurationException e) {

e.printStackTrace(System.err);

System.exit(1);

}

System.out.println(**"\nStarting parsing of "** + file + **"\n"**);

**try** {

parser.parse(file, **this**);

} **catch** (IOException e) {

e.printStackTrace(System.err);

} **catch** (SAXException e) {

e.printStackTrace(System.err);

}

}

**public** **void** startDocument() {

System.out.println(**"Start document: "**);

}

**public** **void** endDocument() {

System.out.println(**"End document: "**);

}

**public** **void** startElement(String uri, String localName, String qname, Attributes attr) {

System.out.println(**"Start element: local name: "** + localName + **" qname: "** + qname + **" uri: "**

+ uri);

}

**public** **void** endElement(String uri, String localName, String qname) {

System.out.println(**"End element: local name: "** + localName + **" qname: "** + qname + **" uri: "**

+ uri);

}

**public** **void** characters(**char**[] ch, **int** start, **int** length) {

System.out.println(**"Characters: "** + **new** String(ch, start, length));

}

**public** **void** ignorableWhitespace(**char**[] ch, **int** start, **int** length) {

System.out.println(**"Ignorable whitespace: "** + **new** String(ch, start, length));

}

}

The code above generates the following result.



## Error Handler

The parser can generate three kinds of errors:

* a fatal error
* an error
* a warning

When a fatal error occurs, the parser cannot continue.

For nonfatal errors and warnings, default error handler would not generate exceptions and no messages are displayed.

The following line installs the our own error handler.

reader.setErrorHandler(new MyErrorHandler());

The MyErrorHandler class implements the standard org.xml.sax.ErrorHandler interface, and defines a method to obtain the exception information that is provided by any SAXParseException.

The complete code.

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

/\*www . ja va2 s. c om\*/

import org.xml.sax.ErrorHandler;

import org.xml.sax.InputSource;

import org.xml.sax.SAXException;

import org.xml.sax.SAXParseException;

import org.xml.sax.XMLReader;

class MyErrorHandler implements ErrorHandler {

public void warning(SAXParseException e) throws SAXException {

show("Warning", e);

throw (e);

}

public void error(SAXParseException e) throws SAXException {

show("Error", e);

throw (e);

}

public void fatalError(SAXParseException e) throws SAXException {

show("Fatal Error", e);

throw (e);

}

private void show(String type, SAXParseException e) {

System.out.println(type + ": " + e.getMessage());

System.out.println("Line " + e.getLineNumber() + " Column " + e.getColumnNumber());

System.out.println("System ID: " + e.getSystemId());

}

}

// Installation and Use of an Error Handler in a SAX Parser

public class SAXCheck {

static public void main(String[] arg) throws Exception {

boolean validate = false;

validate = true;

SAXParserFactory spf = SAXParserFactory.newInstance();

spf.setValidating(validate);

XMLReader reader = null;

SAXParser parser = spf.newSAXParser();

reader = parser.getXMLReader();

reader.setErrorHandler(new MyErrorHandler());

InputSource is = new InputSource("test.xml");

reader.parse(is);

}

}

## XML Schema validation

We can turn on XML Schema validation during parsing with a SAXParser.

import java.io.File;

/\*w w w . java2s. com\*/

import javax.xml.XMLConstants;

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

import javax.xml.validation.Schema;

import javax.xml.validation.SchemaFactory;

public class Main {

public static void main(String args[]) throws Exception {

String language = XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI;

SchemaFactory factory = SchemaFactory.newInstance(language);

Schema schema = factory.newSchema(new **File**("yourSchema"));

SAXParserFactory spf = SAXParserFactory.newInstance();

spf.setSchema(schema);

SAXParser parser = spf.newSAXParser();

// parser.parse(...);

}

}

## DefaultHandler

The following code shows that when using the DefaultHandler we don't need to implement all methods. We only need to provide the implementation for the methods we care about.

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

/\*from ww w . ja va 2 s. co m\*/

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

public class Main {

public static void main(String args[]) throws Exception {

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser saxParser = factory.newSAXParser();

DefaultHandler handler = new DefaultHandler() {

public void startElement(String uri, String localName, String qName,

Attributes attributes) throws SAXException {

System.out.println(qName);

}

public void characters(char ch[], int start, int length)

throws SAXException {

System.out.println(new String(ch, start, length));

}

};

saxParser.parse(args[0], handler);

}

}

The following code handles SAX errors by override the error handler methods from DefaultHandler

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.SAXParseException;

import org.xml.sax.helpers.DefaultHandler;

public class Main {

public static void main(String[] argv) throws Exception {

SAXParserFactory factory = SAXParserFactory.newInstance();

factory.setValidating(true);

SAXParser parser = factory.newSAXParser();

SaxHandler handler = new SaxHandler();

parser.parse("sample.xml", handler);

}

}

class SaxHandler extends DefaultHandler {

public void startElement(String uri, String localName, String qName, Attributes attrs)

throws SAXException {

if (qName.equals("order")) {

}

}

public void error(SAXParseException ex) throws SAXException {

System.out.println("ERROR: [at " + ex.getLineNumber() + "] " + ex);

}

public void fatalError(SAXParseException ex) throws SAXException {

System.out.println("FATAL\_ERROR: [at " + ex.getLineNumber() + "] " + ex);

}

public void warning(SAXParseException ex) throws SAXException {

System.out.println("WARNING: [at " + ex.getLineNumber() + "] " + ex);

}

}

## ContentHandler

The following code chooses to implement the ContentHandler interface and provide implementation for all necessary methods.

It also implements the ErrorHandler interface.

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

// www . ja va2s . c om

import org.xml.sax.Attributes;

import org.xml.sax.ContentHandler;

import org.xml.sax.ErrorHandler;

import org.xml.sax.InputSource;

import org.xml.sax.Locator;

import org.xml.sax.SAXException;

import org.xml.sax.SAXParseException;

import org.xml.sax.XMLReader;

public class Main {

static public void main(String[] arg) throws Exception {

String filename = "yourXML.xml";

// Create a new factory that will create the parser.

SAXParserFactory spf = SAXParserFactory.newInstance();

// Create the XMLReader to be used to parse the document.

SAXParser parser = spf.newSAXParser();

XMLReader reader = parser.getXMLReader();

// Specify the error handler and the content handler.

reader.setErrorHandler(new MyErrorHandler());

reader.setContentHandler(new MyContentHandler());

// Use the XMLReader to parse the entire file.

InputSource is = new InputSource(filename);

reader.parse(is);

}

}

class MyContentHandler implements ContentHandler {

private Locator locator;

/\*\*

\* The name and of the SAX document and the current location within the

\* document.

\*/

public void setDocumentLocator(Locator locator) {

this.locator = locator;

System.out.println("-" + locator.getLineNumber() + "---Document ID: "

+ locator.getSystemId());

}

/\*\* The parsing of a document has started.. \*/

public void startDocument() {

System.out.println("-" + locator.getLineNumber()

+ "---Document parse started");

}

/\*\* The parsing of a document has completed.. \*/

public void endDocument() {

System.out.println("-" + locator.getLineNumber()

+ "---Document parse ended");

}

/\*\* The start of a namespace scope \*/

public void startPrefixMapping(String prefix, String uri) {

System.out.println("-" + locator.getLineNumber()

+ "---Namespace scope begins");

System.out.println(" " + prefix + "=\"" + uri + "\"");

}

/\*\* The end of a namespace scope \*/

public void endPrefixMapping(String prefix) {

System.out.println("-" + locator.getLineNumber()

+ "---Namespace scope ends");

System.out.println(" " + prefix);

}

/\*\* The opening tag of an element. \*/

public void startElement(String namespaceURI, String localName, String qName,

Attributes atts) {

System.out.println("-" + locator.getLineNumber()

+ "---Opening tag of an element");

System.out.println(" Namespace: " + namespaceURI);

System.out.println(" Local name: " + localName);

System.out.println(" Qualified name: " + qName);

for (int i = 0; i < atts.getLength(); i++) {

System.out.println(" Attribute: " + atts.getQName(i) + "=\""

+ atts.getValue(i) + "\"");

}

}

/\*\* The closing tag of an element. \*/

public void endElement(String namespaceURI, String localName, String qName) {

System.out.println("-" + locator.getLineNumber()

+ "---Closing tag of an element");

System.out.println(" Namespace: " + namespaceURI);

System.out.println(" Local name: " + localName);

System.out.println(" Qualified name: " + qName);

}

/\*\* Character data. \*/

public void characters(char[] ch, int start, int length) {

System.out.println("-" + locator.getLineNumber() + "---Character data");

showCharacters(ch, start, length);

}

/\*\* Ignorable whitespace character data. \*/

public void ignorableWhitespace(char[] ch, int start, int length) {

System.out.println("-" + locator.getLineNumber() + "---Whitespace");

showCharacters(ch, start, length);

}

/\*\* Processing Instruction \*/

public void processingInstruction(String target, String data) {

System.out.println("-" + locator.getLineNumber()

+ "---Processing Instruction");

System.out.println(" Target: " + target);

System.out.println(" Data: " + data);

}

/\*\* A skipped entity. \*/

public void skippedEntity(String name) {

System.out.println("-" + locator.getLineNumber() + "---Skipped Entity");

System.out.println(" Name: " + name);

}

/\*\*

\* Internal method to format arrays of characters so the special whitespace

\* characters will show.

\*/

public void showCharacters(char[] ch, int start, int length) {

System.out.print(" \"");

for (int i = start; i < start + length; i++)

switch (ch[i]) {

case '\n':

System.out.print("\\n");

break;

case '\r':

System.out.print("\\r");

break;

case '\t':

System.out.print("\\t");

break;

default:

System.out.print(ch[i]);

break;

}

System.out.println("\"");

}

}

class MyErrorHandler implements ErrorHandler {

public void warning(SAXParseException e) throws SAXException {

show("Warning", e);

throw (e);

}

public void error(SAXParseException e) throws SAXException {

show("Error", e);

throw (e);

}

public void fatalError(SAXParseException e) throws SAXException {

show("Fatal Error", e);

throw (e);

}

private void show(String type, SAXParseException e) {

System.out.println(type + ": " + e.getMessage());

System.out.println("Line " + e.getLineNumber() + " Column "

+ e.getColumnNumber());

System.out.println("System ID: " + e.getSystemId());

}

}

## Locator

The following code shows how to access Locator interface from DefaultHandler.

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

/\* w ww.j a va2 s. c o m\*/

import org.xml.sax.Attributes;

import org.xml.sax.Locator;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

public class Main{

public static void main(String[] args) throws Exception {

SAXParserFactory factory = SAXParserFactory.newInstance();

factory.setValidating(true);

SAXParser parser = factory.newSAXParser();

parser.parse("sample.xml", new SampleOfXmlLocator());

}

}

class SampleOfXmlLocator extends DefaultHandler {

private Locator locator;

public void setDocumentLocator(Locator locator) {

this.locator = locator;

}

public void startElement(String uri, String localName, String qName, Attributes attrs)

throws SAXException {

if (qName.equals("order")) {

System.out.println("here process element start");

} else {

String location = "";

if (locator != null) {

location = locator.getSystemId(); // XML-document name;

location += " line " + locator.getLineNumber();

location += ", column " + locator.getColumnNumber();

location += ": ";

}

throw new SAXException(location + "Illegal element");

}

}

}

ContentHandler Interface

This interface specifies the callback methods that the SAX parser uses to notify an application program of the components of the XML document that it has seen.

* **void startDocument()** - Called at the beginning of a document.
* **void endDocument()** - Called at the end of a document.
* **void startElement(String uri, String localName, String qName, Attributes atts)** - Called at the beginning of an element.
* **void endElement(String uri, String localName,String qName)** - Called at the end of an element.
* **void characters(char[] ch, int start, int length)** - Called when character data is encountered.
* **void ignorableWhitespace( char[] ch, int start, int length)** - Called when a DTD is present and ignorable whitespace is encountered.
* **void processingInstruction(String target, String data)** - Called when a processing instruction is recognized.
* **void setDocumentLocator(Locator locator))** - Provides a Locator that can be used to identify positions in the document.
* **void skippedEntity(String name)** - Called when an unresolved entity is encountered.
* **void startPrefixMapping(String prefix, String uri)** - Called when a new namespace mapping is defined.
* **void endPrefixMapping(String prefix)** - Called when a namespace definition ends its scope.

Attributes Interface

This interface specifies methods for processing the attributes connected to an element.

* **int getLength()** - Returns number of attributes.
* **String getQName(int index)**
* **String getValue(int index)**
* **String getValue(String qname)**

Demo Example

**Here is the input xml file we need to parse:**

<?xml version="1.0"?>

<class>

<student rollno="393">

<firstname>dinkar</firstname>

<lastname>kad</lastname>

<nickname>dinkar</nickname>

<marks>85</marks>

</student>

<student rollno="493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>vinni</nickname>

<marks>95</marks>

</student>

<student rollno="593">

<firstname>jasvir</firstname>

<lastname>singn</lastname>

<nickname>jazz</nickname>

<marks>90</marks>

</student>

</class>

*UserHandler.java*

package com.tutorialspoint.xml;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

public class UserHandler extends DefaultHandler {

boolean bFirstName = false;

boolean bLastName = false;

boolean bNickName = false;

boolean bMarks = false;

@Override

public void startElement(String uri,

String localName, String qName, Attributes attributes)

throws SAXException {

if (qName.equalsIgnoreCase("student")) {

String rollNo = attributes.getValue("rollno");

System.out.println("Roll No : " + rollNo);

} else if (qName.equalsIgnoreCase("firstname")) {

bFirstName = true;

} else if (qName.equalsIgnoreCase("lastname")) {

bLastName = true;

} else if (qName.equalsIgnoreCase("nickname")) {

bNickName = true;

}

else if (qName.equalsIgnoreCase("marks")) {

bMarks = true;

}

}

@Override

public void endElement(String uri,

String localName, String qName) throws SAXException {

if (qName.equalsIgnoreCase("student")) {

System.out.println("End Element :" + qName);

}

}

@Override

public void characters(char ch[],

int start, int length) throws SAXException {

if (bFirstName) {

System.out.println("First Name: "

+ new String(ch, start, length));

bFirstName = false;

} else if (bLastName) {

System.out.println("Last Name: "

+ new String(ch, start, length));

bLastName = false;

} else if (bNickName) {

System.out.println("Nick Name: "

+ new String(ch, start, length));

bNickName = false;

} else if (bMarks) {

System.out.println("Marks: "

+ new String(ch, start, length));

bMarks = false;

}

}

}

*SAXParserDemo.java*

package com.tutorialspoint.xml;

import java.io.File;

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

public class SAXParserDemo {

public static void main(String[] args){

try {

File inputFile = new File("input.txt");

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser saxParser = factory.newSAXParser();

UserHandler userhandler = new UserHandler();

saxParser.parse(inputFile, userhandler);

} catch (Exception e) {

e.printStackTrace();

}

}

}

class UserHandler extends DefaultHandler {

boolean bFirstName = false;

boolean bLastName = false;

boolean bNickName = false;

boolean bMarks = false;

@Override

public void startElement(String uri,

String localName, String qName, Attributes attributes)

throws SAXException {

if (qName.equalsIgnoreCase("student")) {

String rollNo = attributes.getValue("rollno");

System.out.println("Roll No : " + rollNo);

} else if (qName.equalsIgnoreCase("firstname")) {

bFirstName = true;

} else if (qName.equalsIgnoreCase("lastname")) {

bLastName = true;

} else if (qName.equalsIgnoreCase("nickname")) {

bNickName = true;

}

else if (qName.equalsIgnoreCase("marks")) {

bMarks = true;

}

}

@Override

public void endElement(String uri,

String localName, String qName) throws SAXException {

if (qName.equalsIgnoreCase("student")) {

System.out.println("End Element :" + qName);

}

}

@Override

public void characters(char ch[],

int start, int length) throws SAXException {

if (bFirstName) {

System.out.println("First Name: "

+ new String(ch, start, length));

bFirstName = false;

} else if (bLastName) {

System.out.println("Last Name: "

+ new String(ch, start, length));

bLastName = false;

} else if (bNickName) {

System.out.println("Nick Name: "

+ new String(ch, start, length));

bNickName = false;

} else if (bMarks) {

System.out.println("Marks: "

+ new String(ch, start, length));

bMarks = false;

}

}

}

This would produce the following result:

Roll No : 393

First Name: dinkar

Last Name: kad

Nick Name: dinkar

Marks: 85

End Element :student

Roll No : 493

First Name: Vaneet

Last Name: Gupta

Nick Name: vinni

Marks: 95

End Element :student

Roll No : 593

First Name: jasvir

Last Name: singn

Nick Name: jazz

Marks: 90

End Element :student

Demo Example

**Here is the input text file we need to Query for rollno: 393**

<?xml version="1.0"?>

<class>

<student rollno="393">

<firstname>dinkar</firstname>

<lastname>kad</lastname>

<nickname>dinkar</nickname>

<marks>85</marks>

</student>

<student rollno="493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>vinni</nickname>

<marks>95</marks>

</student>

<student rollno="593">

<firstname>jasvir</firstname>

<lastname>singn</lastname>

<nickname>jazz</nickname>

<marks>90</marks>

</student>

</class>

*UserHandler.java*

package com.tutorialspoint.xml;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

public class UserHandler extends DefaultHandler {

boolean bFirstName = false;

boolean bLastName = false;

boolean bNickName = false;

boolean bMarks = false;

String rollNo = null;

@Override

public void startElement(String uri,

String localName, String qName, Attributes attributes)

throws SAXException {

if (qName.equalsIgnoreCase("student")) {

rollNo = attributes.getValue("rollno");

}

if(("393").equals(rollNo) &&

qName.equalsIgnoreCase("student")){

System.out.println("Start Element :" + qName);

}

if (qName.equalsIgnoreCase("firstname")) {

bFirstName = true;

} else if (qName.equalsIgnoreCase("lastname")) {

bLastName = true;

} else if (qName.equalsIgnoreCase("nickname")) {

bNickName = true;

}

else if (qName.equalsIgnoreCase("marks")) {

bMarks = true;

}

}

@Override

public void endElement(String uri,

String localName, String qName) throws SAXException {

if (qName.equalsIgnoreCase("student")) {

if(("393").equals(rollNo)

&& qName.equalsIgnoreCase("student"))

System.out.println("End Element :" + qName);

}

}

@Override

public void characters(char ch[],

int start, int length) throws SAXException {

if (bFirstName && ("393").equals(rollNo)) {

//age element, set Employee age

System.out.println("First Name: " +

new String(ch, start, length));

bFirstName = false;

} else if (bLastName && ("393").equals(rollNo)) {

System.out.println("Last Name: " +

new String(ch, start, length));

bLastName = false;

} else if (bNickName && ("393").equals(rollNo)) {

System.out.println("Nick Name: " +

new String(ch, start, length));

bNickName = false;

} else if (bMarks && ("393").equals(rollNo)) {

System.out.println("Marks: " +

new String(ch, start, length));

bMarks = false;

}

}

}

*SAXQueryDemo.java*

package com.tutorialspoint.xml;

import java.io.File;

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;

import org.xml.sax.SAXException;

import org.xml.sax.helpers.DefaultHandler;

public class SAXQueryDemo {

public static void main(String[] args){

try {

File inputFile = new File("input.txt");

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser saxParser = factory.newSAXParser();

UserHandler userhandler = new UserHandler();

saxParser.parse(inputFile, userhandler);

} catch (Exception e) {

e.printStackTrace();

}

}

}

class UserHandler extends DefaultHandler {

boolean bFirstName = false;

boolean bLastName = false;

boolean bNickName = false;

boolean bMarks = false;

String rollNo = null;

@Override

public void startElement(String uri,

String localName, String qName, Attributes attributes)

throws SAXException {

if (qName.equalsIgnoreCase("student")) {

rollNo = attributes.getValue("rollno");

}

if(("393").equals(rollNo) &&

qName.equalsIgnoreCase("student")){

System.out.println("Start Element :" + qName);

}

if (qName.equalsIgnoreCase("firstname")) {

bFirstName = true;

} else if (qName.equalsIgnoreCase("lastname")) {

bLastName = true;

} else if (qName.equalsIgnoreCase("nickname")) {

bNickName = true;

}

else if (qName.equalsIgnoreCase("marks")) {

bMarks = true;

}

}

@Override

public void endElement(String uri,

String localName, String qName) throws SAXException {

if (qName.equalsIgnoreCase("student")) {

if(("393").equals(rollNo)

&& qName.equalsIgnoreCase("student"))

System.out.println("End Element :" + qName);

}

}

@Override

public void characters(char ch[],

int start, int length) throws SAXException {

if (bFirstName && ("393").equals(rollNo)) {

//age element, set Employee age

System.out.println("First Name: " +

new String(ch, start, length));

bFirstName = false;

} else if (bLastName && ("393").equals(rollNo)) {

System.out.println("Last Name: " +

new String(ch, start, length));

bLastName = false;

} else if (bNickName && ("393").equals(rollNo)) {

System.out.println("Nick Name: " +

new String(ch, start, length));

bNickName = false;

} else if (bMarks && ("393").equals(rollNo)) {

System.out.println("Marks: " +

new String(ch, start, length));

bMarks = false;

}

}

}

This would produce the following result:

Start Element :student

First Name: dinkar

Last Name: kad

Nick Name: dinkar

Marks: 85

End Element :student

**It is better to use StAX parser for creating XML than using SAX parser.**

Demo Example

**Here is the input xml file we need to Modify by appending <Result>Pass<Result/>**

**at the end of </marks> tag**

<?xml version="1.0"?>

<class>

<student rollno="393">

<firstname>dinkar</firstname>

<lastname>kad</lastname>

<nickname>dinkar</nickname>

<marks>85</marks>

</student>

<student rollno="493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>vinni</nickname>

<marks>95</marks>

</student>

<student rollno="593">

<firstname>jasvir</firstname>

<lastname>singn</lastname>

<nickname>jazz</nickname>

<marks>90</marks>

</student>

</class>

*SAXModifyDemo.java*

package com.tutorialspoint.xml;

import java.io.\*;

import org.xml.sax.\*;

import javax.xml.parsers.\*;

import org.xml.sax.helpers.DefaultHandler;

public class SAXModifyDemo extends DefaultHandler {

static String displayText[] = new String[1000];

static int numberLines = 0;

static String indentation = "";

public static void main(String args[]) {

try {

File inputFile = new File("input.txt");

SAXParserFactory factory =

SAXParserFactory.newInstance();

SAXModifyDemo obj = new SAXModifyDemo();

obj.childLoop(inputFile);

FileWriter filewriter = new FileWriter("newfile.xml");

for(int loopIndex = 0; loopIndex < numberLines; loopIndex++){

filewriter.write(displayText[loopIndex].toCharArray());

filewriter.write('\n');

System.out.println(displayText[loopIndex].toString());

}

filewriter.close();

}

catch (Exception e) {

e.printStackTrace(System.err);

}

}

public void childLoop(File input){

DefaultHandler handler = this;

SAXParserFactory factory = SAXParserFactory.newInstance();

try {

SAXParser saxParser = factory.newSAXParser();

saxParser.parse(input, handler);

} catch (Throwable t) {}

}

public void startDocument() {

displayText[numberLines] = indentation;

displayText[numberLines] += "<?xml version=\"1.0\" encoding=\""+

"UTF-8" + "\"?>";

numberLines++;

}

public void processingInstruction(String target,

String data) {

displayText[numberLines] = indentation;

displayText[numberLines] += "<?";

displayText[numberLines] += target;

if (data != null && data.length() > 0) {

displayText[numberLines] += ' ';

displayText[numberLines] += data;

}

displayText[numberLines] += "?>";

numberLines++;

}

public void startElement(String uri, String localName,

String qualifiedName, Attributes attributes) {

displayText[numberLines] = indentation;

indentation += " ";

displayText[numberLines] += '<';

displayText[numberLines] += qualifiedName;

if (attributes != null) {

int numberAttributes = attributes.getLength();

for (int loopIndex = 0; loopIndex < numberAttributes;

loopIndex++){

displayText[numberLines] += ' ';

displayText[numberLines] += attributes.getQName(loopIndex);

displayText[numberLines] += "=\"";

displayText[numberLines] += attributes.getValue(loopIndex);

displayText[numberLines] += '"';

}

}

displayText[numberLines] += '>';

numberLines++;

}

public void characters(char characters[],

int start, int length) {

String characterData = (new String(characters, start, length)).trim();

if(characterData.indexOf("\n") < 0 && characterData.length() > 0) {

displayText[numberLines] = indentation;

displayText[numberLines] += characterData;

numberLines++;

}

}

public void endElement(String uri, String localName,

String qualifiedName) {

indentation = indentation.substring(0, indentation.length() - 4) ;

displayText[numberLines] = indentation;

displayText[numberLines] += "</";

displayText[numberLines] += qualifiedName;

displayText[numberLines] += '>';

numberLines++;

if (qualifiedName.equals("marks")) {

startElement("", "Result", "Result", null);

characters("Pass".toCharArray(), 0, "Pass".length());

endElement("", "Result", "Result");

}

}

}

This would produce the following result:

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

<class>

<student rollno="393">

<firstname>

dinkar

</firstname>

<lastname>

kad

</lastname>

<nickname>

dinkar

</nickname>

<marks>

85

</marks>

<Result>

Pass

</Result>

</student>

<student rollno="493">

<firstname>

Vaneet

</firstname>

<lastname>

Gupta

</lastname>

<nickname>

vinni

</nickname>

<marks>

95

</marks>

<Result>

Pass

</Result>

</student>

<student rollno="593">

<firstname>

jasvir

</firstname>

<lastname>

singn

</lastname>

<nickname>

jazz

</nickname>

<marks>

90

</marks>

<Result>

Pass

</Result>

</student>

</class>

Java DOM Parser – Overview

The Document Object Model is an official recommendation of the World Wide Web Consortium (W3C). It defines an interface that enables programs to access and update the style, structure,and contents of XML documents. XML parsers that support the DOM implement that interface.

When to use?

You should use a DOM parser when:

* You need to know a lot about the structure of a document
* You need to move parts of the document around (you might want to sort certain elements, for example)
* You need to use the information in the document more than once

What you get?

When you parse an XML document with a DOM parser, you get back a tree structure that contains all of the elements of your document. The DOM provides a variety of functions you can use to examine the contents and structure of the document.

Advantages

The DOM is a common interface for manipulating document structures. One of its design goals is that Java code written for one DOM-compliant parser should run on any other DOM-compliant parser without changes.

A DOM is a standard tree structure, where each node contains one of the components from an XML structure.

The two most common types of nodes in XML document are element nodes and text nodes.

With Java DOM API we can create nodes, remove nodes, change their contents, and traverse the node hierarchy.

When to Use DOM

The Document Object Model standard is designed for XML documents manipulation.

DOM was intended to be language-neutral. Java DOM Parser does not take advantage of Java's object-oriented features.

Mixed-Content Model

Text and elements is intermixed in a DOM hierarchy. That kind of structure is called mixed content in the DOM model.

For example, we have the following xml structure:

<yourTag>This is an <bold>important</bold> test.</yourTag>

The hierarchy of DOM nodes is as follows, where each line represents one node:

ELEMENT: yourTag

+ TEXT: This is an

+ ELEMENT: bold

+ TEXT: important

+ TEXT: test.

The yourTag element contains text, followed by a sub-element, followed by additional text.

Types of Nodes

To support mixed content, DOM nodes are very simple. The "content" of the tag element identifies the kind of node it is.

For example, the <yourTag> node content is the name of the element, yourTag.

The DOM Node API defines nodeValue(), nodeType(), and nodeName() methods.

For the element node <yourTag>, nodeName() returns yourTag, while nodeValue() returns null.

For the text node + TEXT: This is an, nodeName() returns #text, and nodeValue() returns "This is an".

Example

The following code shows how to use DOM parser to parse a xml file and get a org.w3c.dom.Document object.

/\*w ww .j ava2s . c o m\*/

**import** java.io.File;

**import** javax.xml.parsers.DocumentBuilder;

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** org.w3c.dom.Document;

**public** **class** Main {

**public** **static** **void** main(String[] args) **throws** Exception {

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

DocumentBuilder db = null;

db = dbf.newDocumentBuilder();

Document doc = db.parse(**new** **File**(**"games.xml"**));

}

}

Example 2

The following code shows how to do a DOM dump.

**import** javax.xml.parsers.DocumentBuilder;

**import** javax.xml.parsers.DocumentBuilderFactory;

/\*from ww w. j ava 2 s . co m\*/

**import** org.w3c.dom.Document;

**import** org.w3c.dom.Node;

**import** org.w3c.dom.NodeList;

**import** org.xml.sax.ErrorHandler;

**import** org.xml.sax.InputSource;

**import** org.xml.sax.SAXException;

**import** org.xml.sax.SAXParseException;

**public** **class** Main{

**static** **public** **void** main(String[] arg) **throws** Exception{

String filename = **"input.xml"**;

**boolean** validate = true;

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

dbf.setValidating(validate);

dbf.setNamespaceAware(true);

dbf.setIgnoringElementContentWhitespace(true);

DocumentBuilder builder = dbf.newDocumentBuilder();

builder.setErrorHandler(**new** MyErrorHandler());

InputSource is = **new** InputSource(filename);

Document doc = builder.parse(is);

TreeDumper td = **new** TreeDumper();

td.dump(doc);

}

}

**class** TreeDumper {

**public** **void** dump(Document doc) {

dumpLoop((Node)doc,**""**);

}

**private** **void** dumpLoop(Node node,String indent) {

**switch**(node.getNodeType()) {

**case** Node.CDATA\_SECTION\_NODE:

System.out.println(indent + **"CDATA\_SECTION\_NODE"**);

**break**;

**case** Node.COMMENT\_NODE:

System.out.println(indent + **"COMMENT\_NODE"**);

**break**;

**case** Node.DOCUMENT\_FRAGMENT\_NODE:

System.out.println(indent + **"DOCUMENT\_FRAGMENT\_NODE"**);

**break**;

**case** Node.DOCUMENT\_NODE:

System.out.println(indent + **"DOCUMENT\_NODE"**);

**break**;

**case** Node.DOCUMENT\_TYPE\_NODE:

System.out.println(indent + **"DOCUMENT\_TYPE\_NODE"**);

**break**;

**case** Node.ELEMENT\_NODE:

System.out.println(indent + **"ELEMENT\_NODE"**);

**break**;

**case** Node.ENTITY\_NODE:

System.out.println(indent + **"ENTITY\_NODE"**);

**break**;

**case** Node.ENTITY\_REFERENCE\_NODE:

System.out.println(indent + **"ENTITY\_REFERENCE\_NODE"**);

**break**;

**case** Node.NOTATION\_NODE:

System.out.println(indent + **"NOTATION\_NODE"**);

**break**;

**case** Node.PROCESSING\_INSTRUCTION\_NODE:

System.out.println(indent + **"PROCESSING\_INSTRUCTION\_NODE"**);

**break**;

**case** Node.TEXT\_NODE:

System.out.println(indent + **"TEXT\_NODE"**);

**break**;

**default**:

System.out.println(indent + **"Unknown node"**);

**break**;

}

NodeList list = node.getChildNodes();

**for**(**int** i=0; i<list.getLength(); i++)

dumpLoop(list.item(i),indent + **" "**);

}

}

**class** MyErrorHandler **implements** ErrorHandler {

**public** **void** warning(SAXParseException e) **throws** SAXException {

show(**"Warning"**, e);

**throw** (e);

}

**public** **void** error(SAXParseException e) **throws** SAXException {

show(**"Error"**, e);

**throw** (e);

}

**public** **void** fatalError(SAXParseException e) **throws** SAXException {

show(**"Fatal Error"**, e);

**throw** (e);

}

**private** **void** show(String type, SAXParseException e) {

System.out.println(type + **": "** + e.getMessage());

System.out.println(**"Line "** + e.getLineNumber() + **" Column "**

+ e.getColumnNumber());

System.out.println(**"System ID: "** + e.getSystemId());

}

}

Error handler

The following code shows how to handle error when parsing the XML with DOM parser.

**import** java.io.IOException;

/\* w w w. j av a 2s .com\*/

**import** javax.xml.parsers.DocumentBuilder;

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** javax.xml.parsers.ParserConfigurationException;

**import** org.w3c.dom.Document;

**import** org.xml.sax.ErrorHandler;

**import** org.xml.sax.InputSource;

**import** org.xml.sax.SAXException;

**import** org.xml.sax.SAXParseException;

**public** **class** DOMCheck {

**static** **public** **void** main(String[] arg) {

**boolean** validate = true;

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

dbf.setValidating(validate);

dbf.setNamespaceAware(true);

**try** {

DocumentBuilder builder = dbf.newDocumentBuilder();

builder.setErrorHandler(**new** MyErrorHandler());

InputSource is = **new** InputSource(**"person.xml"**);

Document doc = builder.parse(is);

} **catch** (SAXException e) {

System.out.println(e);

} **catch** (ParserConfigurationException e) {

System.err.println(e);

} **catch** (IOException e) {

System.err.println(e);

}

}

}

**class** MyErrorHandler **implements** ErrorHandler {

**public** **void** warning(SAXParseException e) **throws** SAXException {

show(**"Warning"**, e);

**throw** (e);

}

**public** **void** error(SAXParseException e) **throws** SAXException {

show(**"Error"**, e);

**throw** (e);

}

**public** **void** fatalError(SAXParseException e) **throws** SAXException {

show(**"Fatal Error"**, e);

**throw** (e);

}

**private** **void** show(String type, SAXParseException e) {

System.out.println(type + **": "** + e.getMessage());

System.out.println(**"Line "** + e.getLineNumber() + **" Column "** + e.getColumnNumber());

System.out.println(**"System ID: "** + e.getSystemId());

}

}

Example 3

The following code shows how to recursively visit all nodes in a DOM tree.

**import** java.io.File;

/\*w w w . j a v a 2s .c o m\*/

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** org.w3c.dom.Document;

**import** org.w3c.dom.Node;

**import** org.w3c.dom.NodeList;

**public** **class** Main {

**public** **static** **void** main(String[] argv) **throws** Exception{

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

factory.setValidating(true);

factory.setExpandEntityReferences(false);

Document doc = factory.newDocumentBuilder().parse(**new** **File**(**"file.xml"**));

visit(doc, 0);

}

**public** **static** **void** visit(Node node, **int** level) {

NodeList list = node.getChildNodes();

**for** (**int** i = 0; i < list.getLength(); i++) {

Node childNode = list.item(i);

visit(childNode, level + 1);

}

}

}

Example 4

The following code shows how to convert an XML Fragment into a DOM Fragment

**import** java.io.File;

**import** java.io.StringReader;

//from w w w .j av a 2s . c om

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** org.w3c.dom.Document;

**import** org.w3c.dom.DocumentFragment;

**import** org.w3c.dom.Element;

**import** org.w3c.dom.Node;

**import** org.xml.sax.InputSource;

**public** **class** Main {

**public** **static** **void** main(String[] argv) **throws** Exception {

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

factory.setValidating(true);

Document doc = factory.newDocumentBuilder().parse(**new** **File**(**"infilename.xml"**));

String fragment = **"<fragment>aaa</fragment>"**;

factory = DocumentBuilderFactory.newInstance();

Document d = factory.newDocumentBuilder().parse(**new** InputSource(**new** StringReader(fragment)));

Node node = doc.importNode(d.getDocumentElement(), true);

DocumentFragment docfrag = doc.createDocumentFragment();

**while** (node.hasChildNodes()) {

docfrag.appendChild(node.removeChild(node.getFirstChild()));

}

Element element = doc.getDocumentElement();

element.appendChild(docfrag);

}

}

Example 5

The following code shows how to parse an XML string: Using DOM and a StringReader.

**import** java.io.StringReader;

/\* ww w . jav a2 s. c o m\*/

**import** javax.xml.parsers.DocumentBuilder;

**import** javax.xml.parsers.DocumentBuilderFactory;

**import** org.w3c.dom.CharacterData;

**import** org.w3c.dom.Document;

**import** org.w3c.dom.Element;

**import** org.w3c.dom.Node;

**import** org.w3c.dom.NodeList;

**import** org.xml.sax.InputSource;

**public** **class** Main {

**public** **static** **void** main(String arg[]) **throws** Exception{

String xmlRecords = **"<data><employee><name>A</name>"**

+ **"<title>Manager</title></employee></data>"**;

DocumentBuilder db = DocumentBuilderFactory.newInstance().newDocumentBuilder();

InputSource is = **new** InputSource();

is.setCharacterStream(**new** StringReader(xmlRecords));

Document doc = db.parse(is);

NodeList nodes = doc.getElementsByTagName(**"employee"**);

**for** (**int** i = 0; i < nodes.getLength(); i++) {

Element element = (Element) nodes.item(i);

NodeList name = element.getElementsByTagName(**"name"**);

Element line = (Element) name.item(0);

System.out.println(**"Name: "** + getCharacterDataFromElement(line));

NodeList title = element.getElementsByTagName(**"title"**);

line = (Element) title.item(0);

System.out.println(**"Title: "** + getCharacterDataFromElement(line));

}

}

**public** **static** String getCharacterDataFromElement(Element e) {

Node child = e.getFirstChild();

**if** (child **instanceof** CharacterData) {

CharacterData cd = (CharacterData) child;

**return** cd.getData();

}

**return** **""**;

}

}

The code above generates the following result.



DOM interfaces

The DOM defines several Java interfaces. Here are the most common interfaces:

* **Node** - The base datatype of the DOM.
* **Element** - The vast majority of the objects you'll deal with are Elements.
* **Attr** Represents an attribute of an element.
* **Text** The actual content of an Element or Attr.
* **Document** Represents the entire XML document. A Document object is often referred to as a DOM tree.

Common DOM methods

When you are working with the DOM, there are several methods you'll use often:

* **Document.getDocumentElement()** - Returns the root element of the document.
* **Node.getFirstChild()** - Returns the first child of a given Node.
* **Node.getLastChild()** - Returns the last child of a given Node.
* **Node.getNextSibling()** - These methods return the next sibling of a given Node.
* **Node.getPreviousSibling()** - These methods return the previous sibling of a given Node.
* **Node.getAttribute(attrName)** - For a given Node, returns the attribute with the requested name.

Steps to Using DOM

Following are the steps used while parsing a document using DOM Parser.

* Import XML-related packages.
* Create a DocumentBuilder
* Create a Document from a file or stream
* Extract the root element
* Examine attributes
* Examine sub-elements

**Import XML-related packages**

import org.w3c.dom.\*;

import javax.xml.parsers.\*;

import java.io.\*;

**Create a DocumentBuilder**

DocumentBuilderFactory factory =

DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

**Create a Document from a file or stream**

StringBuilder xmlStringBuilder = new StringBuilder();

xmlStringBuilder.append("<?xml version="1.0"?> <class> </class>");

ByteArrayInputStream input = new ByteArrayInputStream(

xmlStringBuilder.toString().getBytes("UTF-8"));

Document doc = builder.parse(input);

**Extract the root element**

Element root = document.getDocumentElement();

**Examine attributes**

//returns specific attribute

getAttribute("attributeName");

//returns a Map (table) of names/values

getAttributes();

**Examine sub-elements**

//returns a list of subelements of specified name

getElementsByTagName("subelementName");

//returns a list of all child nodes

getChildNodes();

Demo Example

**Here is the input xml file we need to parse:**

<?xml version="1.0"?>

<class>

<student rollno="393">

<firstname>dinkar</firstname>

<lastname>kad</lastname>

<nickname>dinkar</nickname>

<marks>85</marks>

</student>

<student rollno="493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>vinni</nickname>

<marks>95</marks>

</student>

<student rollno="593">

<firstname>jasvir</firstname>

<lastname>singn</lastname>

<nickname>jazz</nickname>

<marks>90</marks>

</student>

</class>

**Demo Example:**

*DomParserDemo.java*

package com.tutorialspoint.xml;

import java.io.File;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.DocumentBuilder;

import org.w3c.dom.Document;

import org.w3c.dom.NodeList;

import org.w3c.dom.Node;

import org.w3c.dom.Element;

public class DomParserDemo {

public static void main(String[] args){

try {

File inputFile = new File("input.txt");

DocumentBuilderFactory dbFactory

= DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(inputFile);

doc.getDocumentElement().normalize();

System.out.println("Root element :"

+ doc.getDocumentElement().getNodeName());

NodeList nList = doc.getElementsByTagName("student");

System.out.println("----------------------------");

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = nList.item(temp);

System.out.println("\nCurrent Element :"

+ nNode.getNodeName());

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

System.out.println("Student roll no : "

+ eElement.getAttribute("rollno"));

System.out.println("First Name : "

+ eElement

.getElementsByTagName("firstname")

.item(0)

.getTextContent());

System.out.println("Last Name : "

+ eElement

.getElementsByTagName("lastname")

.item(0)

.getTextContent());

System.out.println("Nick Name : "

+ eElement

.getElementsByTagName("nickname")

.item(0)

.getTextContent());

System.out.println("Marks : "

+ eElement

.getElementsByTagName("marks")

.item(0)

.getTextContent());

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

This would produce the following result:

Root element :class

----------------------------

Current Element :student

Student roll no : 393

First Name : dinkar

Last Name : kad

Nick Name : dinkar

Marks : 85

Current Element :student

Student roll no : 493

First Name : Vaneet

Last Name : Gupta

Nick Name : vinni

Marks : 95

Current Element :student

Student roll no : 593

First Name : jasvir

Last Name : singn

Nick Name : jazz

Marks : 90

Demo Example

**Here is the input xml file we need to query:**

<?xml version="1.0"?>

<cars>

<supercars company="Ferrari">

<carname type="formula one">Ferarri 101</carname>

<carname type="sports car">Ferarri 201</carname>

<carname type="sports car">Ferarri 301</carname>

</supercars>

<supercars company="Lamborgini">

<carname>Lamborgini 001</carname>

<carname>Lamborgini 002</carname>

<carname>Lamborgini 003</carname>

</supercars>

<luxurycars company="Benteley">

<carname>Benteley 1</carname>

<carname>Benteley 2</carname>

<carname>Benteley 3</carname>

</luxurycars>

</cars>

**Demo Example:**

*QueryXmlFileDemo.java*

package com.tutorialspoint.xml;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.DocumentBuilder;

import org.w3c.dom.Document;

import org.w3c.dom.NodeList;

import org.w3c.dom.Node;

import org.w3c.dom.Element;

import java.io.File;

public class QueryXmlFileDemo {

public static void main(String argv[]) {

try {

File inputFile = new File("input.txt");

DocumentBuilderFactory dbFactory =

DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(inputFile);

doc.getDocumentElement().normalize();

System.out.print("Root element: ");

System.out.println(doc.getDocumentElement().getNodeName());

NodeList nList = doc.getElementsByTagName("supercars");

System.out.println("----------------------------");

for (int temp = 0; temp < nList.getLength(); temp++) {

Node nNode = nList.item(temp);

System.out.println("\nCurrent Element :");

System.out.print(nNode.getNodeName());

if (nNode.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) nNode;

System.out.print("company : ");

System.out.println(eElement.getAttribute("company"));

NodeList carNameList =

eElement.getElementsByTagName("carname");

for (int count = 0;

count < carNameList.getLength(); count++) {

Node node1 = carNameList.item(count);

if (node1.getNodeType() ==

node1.ELEMENT\_NODE) {

Element car = (Element) node1;

System.out.print("car name : ");

System.out.println(car.getTextContent());

System.out.print("car type : ");

System.out.println(car.getAttribute("type"));

}

}

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

This would produce the following result:

Root element :cars

----------------------------

Current Element :supercars

company : Ferrari

car name : Ferarri 101

car type : formula one

car name : Ferarri 201

car type : sports car

car name : Ferarri 301

car type : sports car

Current Element :supercars

company : Lamborgini

car name : Lamborgini 001

car type :

car name : Lamborgini 002

car type :

car name : Lamborgini 003

car type :

Demo Example

**Here is the XML we need to create:**

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<cars><supercars company="Ferrari">

<carname type="formula one">Ferrari 101</carname>

<carname type="sports">Ferrari 202</carname>

</supercars></cars>

**Demo Example:**

*CreateXmlFileDemo.java*

package com.tutorialspoint.xml;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Attr;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import java.io.File;

public class CreateXmlFileDemo {

public static void main(String argv[]) {

try {

DocumentBuilderFactory dbFactory =

DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder =

dbFactory.newDocumentBuilder();

Document doc = dBuilder.newDocument();

// root element

Element rootElement = doc.createElement("cars");

doc.appendChild(rootElement);

// supercars element

Element supercar = doc.createElement("supercars");

rootElement.appendChild(supercar);

// setting attribute to element

Attr attr = doc.createAttribute("company");

attr.setValue("Ferrari");

supercar.setAttributeNode(attr);

// carname element

Element carname = doc.createElement("carname");

Attr attrType = doc.createAttribute("type");

attrType.setValue("formula one");

carname.setAttributeNode(attrType);

carname.appendChild(

doc.createTextNode("Ferrari 101"));

supercar.appendChild(carname);

Element carname1 = doc.createElement("carname");

Attr attrType1 = doc.createAttribute("type");

attrType1.setValue("sports");

carname1.setAttributeNode(attrType1);

carname1.appendChild(

doc.createTextNode("Ferrari 202"));

supercar.appendChild(carname1);

// write the content into xml file

TransformerFactory transformerFactory =

TransformerFactory.newInstance();

Transformer transformer =

transformerFactory.newTransformer();

DOMSource source = new DOMSource(doc);

StreamResult result =

new StreamResult(new File("C:\\cars.xml"));

transformer.transform(source, result);

// Output to console for testing

StreamResult consoleResult =

new StreamResult(System.out);

transformer.transform(source, consoleResult);

} catch (Exception e) {

e.printStackTrace();

}

}

}

This would produce the following result:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<cars><supercars company="Ferrari">

<carname type="formula one">Ferrari 101</carname>

<carname type="sports">Ferrari 202</carname>

</supercars></cars>

**Here is the input xml file we need to modify:**

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<cars>

<supercars company="Ferrari">

<carname type="formula one">Ferrari 101</carname>

<carname type="sports">Ferrari 202</carname>

</supercars>

<luxurycars company="Benteley">

<carname>Benteley 1</carname>

<carname>Benteley 2</carname>

<carname>Benteley 3</carname>

</luxurycars>

</cars>

**Demo Example:**

*ModifyXmlFileDemo.java*

package com.tutorialspoint.xml;

import java.io.File;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.NamedNodeMap;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

public class ModifyXmlFileDemo {

public static void main(String argv[]) {

try {

File inputFile = new File("input.xml");

DocumentBuilderFactory docFactory =

DocumentBuilderFactory.newInstance();

DocumentBuilder docBuilder =

docFactory.newDocumentBuilder();

Document doc = docBuilder.parse(inputFile);

Node cars = doc.getFirstChild();

Node supercar = doc.getElementsByTagName("supercars").item(0);

// update supercar attribute

NamedNodeMap attr = supercar.getAttributes();

Node nodeAttr = attr.getNamedItem("company");

nodeAttr.setTextContent("Lamborigini");

// loop the supercar child node

NodeList list = supercar.getChildNodes();

for (int temp = 0; temp < list.getLength(); temp++) {

Node node = list.item(temp);

if (node.getNodeType() == Node.ELEMENT\_NODE) {

Element eElement = (Element) node;

if ("carname".equals(eElement.getNodeName())){

if("Ferrari 101".equals(eElement.getTextContent())){

eElement.setTextContent("Lamborigini 001");

}

if("Ferrari 202".equals(eElement.getTextContent()))

eElement.setTextContent("Lamborigini 002");

}

}

}

NodeList childNodes = cars.getChildNodes();

for(int count = 0; count < childNodes.getLength(); count++){

Node node = childNodes.item(count);

if("luxurycars".equals(node.getNodeName()))

cars.removeChild(node);

}

// write the content on console

TransformerFactory transformerFactory =

TransformerFactory.newInstance();

Transformer transformer = transformerFactory.newTransformer();

DOMSource source = new DOMSource(doc);

System.out.println("-----------Modified File-----------");

StreamResult consoleResult = new StreamResult(System.out);

transformer.transform(source, consoleResult);

} catch (Exception e) {

e.printStackTrace();

}

}

}

This would produce the following result:

-----------Modified File-----------

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<cars>

<supercars company="Lamborigini">

<carname type="formula one">Lamborigini 001</carname>

<carname type="sports">Lamborigini 002</carname>

</supercars></cars>

## Attribute

The following code shows how to add attribute to an element.

import java.io.StringWriter;

/\* w ww . j a v a 2s .c om\*/

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.DOMImplementation;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

public class Main {

public static void main(String[] argv) throws Exception {

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

DOMImplementation impl = builder.getDOMImplementation();

Document doc = impl.createDocument(null, null, null);

Element e1 = doc.createElement("api");

doc.appendChild(e1);

Element e2 = doc.createElement("java");

e1.appendChild(e2);

e2.setAttribute("url", "http://www.java2s.com");

//transform the DOM for showing the result in console

DOMSource domSource = new DOMSource(doc);

Transformer transformer = TransformerFactory.newInstance().newTransformer();

transformer.setOutputProperty(OutputKeys.OMIT\_XML\_DECLARATION, "yes");

transformer.setOutputProperty(OutputKeys.METHOD, "xml");

transformer.setOutputProperty(OutputKeys.ENCODING, "ISO-8859-1");

transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");

transformer.setOutputProperty(OutputKeys.INDENT, "yes");

StringWriter sw = new StringWriter();

StreamResult sr = new StreamResult(sw);

transformer.transform(domSource, sr);

System.out.println(sw.toString());

}

}

The code above generates the following result.



Copying Attributes

public void dupAttributes(Document doc) {

Element root = doc.getDocumentElement();

Element personOne = (Element)root.getFirstChild();

Element personTwo = (Element)personOne.getNextSibling();

Element personThree = (Element)personTwo.getNextSibling();

Attr deptAttr = personOne.getAttributeNode("dept");

personOne.removeAttributeNode(deptAttr);

String deptString = deptAttr.getValue();

personTwo.setAttribute("dept",deptString);

personThree.setAttribute("dept",deptString);

String mailString = personOne.getAttribute("mail");

personTwo.setAttribute("mail",mailString);

String titleString = personOne.getAttribute("title");

personOne.removeAttribute("title");

personThree.setAttribute("title",titleString);

}

Deleting Two Attributes

public void delAttribute(Document doc) {

Element root = doc.getDocumentElement();

Element person = (Element)root.getFirstChild();

person.removeAttribute("extension");

person.removeAttribute("dept");

}

## Element

import java.io.StringWriter;

/\* w w w . j a va 2s . c o m\*/

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.DOMImplementation;

import org.w3c.dom.Document;

import org.w3c.dom.Node;

public class Main {

public static void main(String[] argv) throws Exception {

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

DOMImplementation impl = builder.getDOMImplementation();

Document doc = impl.createDocument(null, null, null);

Node root = doc.createElement("A");

doc.appendChild(root);

Node stanza = doc.createElement("B");

root.appendChild(stanza);

Node line = doc.createElement("C");

stanza.appendChild(line);

line.appendChild(doc.createTextNode("test"));

line = doc.createElement("Line");

stanza.appendChild(line);

line.appendChild(doc.createTextNode("test"));

//transform the DOM for showing the result in console

DOMSource domSource = new DOMSource(doc);

Transformer transformer = TransformerFactory.newInstance().newTransformer();

transformer.setOutputProperty(OutputKeys.OMIT\_XML\_DECLARATION, "yes");

transformer.setOutputProperty(OutputKeys.METHOD, "xml");

transformer.setOutputProperty(OutputKeys.ENCODING, "ISO-8859-1");

transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");

transformer.setOutputProperty(OutputKeys.INDENT, "yes");

StringWriter sw = new StringWriter();

StreamResult sr = new StreamResult(sw);

transformer.transform(domSource, sr);

System.out.println(sw.toString());

}

}

The code above generates the following result.



The following code shows how to remove an element from parent.

public void deleteFirstElement(Document doc) {

Element root = doc.getDocumentElement();

Element child = (Element)root.getFirstChild();

root.removeChild(child);

}

## Text Node

The following code shows how to add text node to an element.

import java.io.StringWriter;

//from w w w .jav a2s . c om

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.bootstrap.DOMImplementationRegistry;

import org.w3c.dom.ls.DOMImplementationLS;

import org.w3c.dom.ls.LSOutput;

import org.w3c.dom.ls.LSSerializer;

public class Main {

public static void main(String[] argv) throws Exception {

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

dbf.setNamespaceAware(true);

DocumentBuilder db = dbf.newDocumentBuilder();

Document doc = db.newDocument();

Element root = doc.createElementNS(null, "person"); // Create Root Element

Element item = doc.createElementNS(null, "name"); // Create element

item.appendChild(doc.createTextNode("Jeff"));

root.appendChild(item); // Attach element to Root element

item = doc.createElementNS(null, "age"); // Create another Element

item.appendChild(doc.createTextNode("28"));

root.appendChild(item); // Attach Element to previous element down tree

item = doc.createElementNS(null, "height");

item.appendChild(doc.createTextNode("1.80"));

root.appendChild(item); // Attach another Element - grandaugther

doc.appendChild(root); // Add Root to Document

DOMImplementationRegistry registry = DOMImplementationRegistry

.newInstance();

DOMImplementationLS domImplLS = (DOMImplementationLS) registry

.getDOMImplementation("LS");

LSSerializer ser = domImplLS.createLSSerializer(); // Create a serializer

// for the DOM

LSOutput out = domImplLS.createLSOutput();

StringWriter stringOut = new StringWriter(); // Writer will be a String

out.setCharacterStream(stringOut);

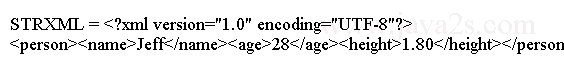
ser.write(doc, out); // Serialize the DOM

System.out.println("STRXML = " + stringOut.toString()); // DOM as a String

}

}

The code above generates the following result.



The following code shows how to edit text by insertion and replacement.

public void edit3(Document doc) {

int count;

int offset;

Element root = doc.getDocumentElement();

Element place = (Element)root.getFirstChild();

Text name = (Text)place.getFirstChild().getFirstChild();

Text directions = (Text)place.getLastChild().getFirstChild();

offset = 7;

name.insertData(offset," black");

offset = 5;

count = 4;

directions.replaceData(offset,count,"right");

}

Modifying Text by Cutting and Pasting

public void edit(Document doc) {

int length;

int count;

int offset;

Element root = doc.getDocumentElement();

Element place = (Element)root.getFirstChild();

Text name = (Text)place.getFirstChild().getFirstChild();

Text directions = (Text)place.getLastChild().getFirstChild();

length = name.getLength();

count = 4;

offset = length - 4;

name.deleteData(offset,count);

length = directions.getLength();

count = 6;

offset = length - count;

String bridge = directions.substringData(offset,count);

name.appendData(bridge);

count = 5;

offset = 4;

directions.deleteData(offset,count);

}

Modifying Text by Replacement

public void edit(Document doc) {

Element root = doc.getDocumentElement();

Element place = (Element)root.getFirstChild();

Text name = (Text)place.getFirstChild().getFirstChild();

Text directions = (Text)place.getLastChild().getFirstChild();

name.setData("AAA");

directions.setData("BBB");

}

## Comment

The following code shows how to create comments node for XML.

import java.io.File;

/\*from w w w . j ava 2s .c om\*/

import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Comment;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

public class Main {

public static void main(String[] argv) throws Exception {

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

factory.setValidating(true);

factory.setExpandEntityReferences(false);

Document doc = factory.newDocumentBuilder().parse(new **File**("filename"));

Element element = doc.getDocumentElement();

Comment comment = doc.createComment("A Document Comment");

element.getParentNode().insertBefore(comment, element);

}

}

## Processing Instruction

The following code shows how to add ProcessingInstruction.

public void addProcessingInstruction(Document doc) {

Element root = doc.getDocumentElement();

Element folks = (Element)root.getLastChild();

ProcessingInstruction pi;

pi = (ProcessingInstruction)doc.createProcessingInstruction(

"validate",

"phone=\"lookup\"");

root.insertBefore(pi,folks);

}

## CDATA

The following code shows how to add CDATA to XML document.

public void addCDATA(Document doc) {

Element root = doc.getDocumentElement();

Element place = (Element)root.getFirstChild();

Element directions = (Element)place.getLastChild();

String dirtext =/\*from w ww. j ava 2 s .c o m\*/

">>>\n" +

"<<<\n" +

"&&&\n" +

"<><><>.";

CDATASection dirdata = doc.createCDATASection(dirtext);

directions.replaceChild(dirdata,directions.getFirstChild());

}

## Clone

public void duplicatePerson(Document doc) {

Element root = doc.getDocumentElement();

Element origPerson = (Element)root.getFirstChild();

Element newPerson = (Element)origPerson.cloneNode(true);

root.appendChild(newPerson);

}