

JAVA XML OVERVIEW

What is XML?

XML is a simple text based language which was designed to store and transport data in plain text format. It stands for Extensible Markup Language. Following are some of the salient features of XML.

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

Example

```
<?xml version="1.0"?>
<Class>
  <Name>First</Name>
  <Sections>
    <Section>
      <Name>A</Name>
      <Students>
        <Student>Rohan</Student>
        <Student>Mohan</Student>
        <Student>Sohan</Student>
        <Student>Lalit</Student>
        <Student>Vinay</Student>
      </Students>
    </Section>
    <Section>
      <Name>B</Name>
      <Students>
        <Student>Robert</Student>
        <Student>Julie</Student>
        <Student>Kalie</Student>
        <Student>Michael</Student>
      </Students>
    </Section>
  </Sections>
</Class>
```

Advantages

Following are the advantages that XML provides:

- **Technology agnostic** - Being plain text, XML is technology independent. It can be used by any technology for data storage and transmission purpose.
- **Human readable**- XML uses simple text format. It is human readable and understandable.
- **Extensible** - in XML, custom tags can be created and used very easily.
- **Allow Validation** - Using XSD, DTD and XML structure can be validated easily.

Disadvantages

Following are the disadvantages of XML usage:

- **Redundant Syntax** - Normally XML file contains lot of repetitive terms.
- **Verbose**-Being a verbose language, XML file size increases the transmission and storage costs.

JAVA XML PARSERS

What is XML Parsing?

Parsing XML refers to going through XML document to access data or to modify data in one or other way.

What is XML Parser?

XML Parser provides way how to access or modify data present in an XML document. Java provides multiple options to parse XML document. Following are various types of parsers which are commonly used to parse XML documents.

- **Dom Parser** - Parses the document by loading the complete contents of the document and creating its complete hierarchical tree in memory.
- **SAX Parser** - Parses the document on event based triggers. Does not load the complete document into the memory.
- **JDOM Parser** - Parses the document in similar fashion to DOM parser but in more easier way.
- **StAX Parser** - Parses the document in similar fashion to SAX parser but in more efficient way.
- **XPath Parser** - Parses the XML based on expression and is used extensively in conjunction with XSLT.
- **DOM4J Parser** - A java library to parse XML, XPath and XSLT using Java Collections Framework , provides support for DOM, SAX and JAXP.

There are JAXB and XSLT APIs available to handle XML parsing in Object Oriented way. We'll elaborate each parser in detail in next chapters.

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.

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.

JAVA DOM PARSER - PARSE XML DOCUMENT

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 : singh
Nick Name : jazz
Marks : 90

JAVA DOM PARSER - QUERY XML DOCUMENT

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="Lamborghini">
    <carname>Lamborghini 001</carname>
    <carname>Lamborghini 002</carname>
    <carname>Lamborghini 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 : Lamborghini
car name : Lamborghini 001
car type :
car name : Lamborghini 002
car type :
car name : Lamborghini 003
car type :

```

JAVA DOM PARSER - CREATE XML DOCUMENT

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>
```

JAVA DOM PARSER - MODIFY XML DOCUMENT

Demo Example

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.txt");
            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("Lamborghini");

            // 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("Lamborghini 001");
            }
            if("Ferrari 202".equals(eElement.getTextContent()))
                eElement.setTextContent("Lamborghini 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="Lamborghini">
<carname type="formula one">Lamborghini 001</carname>
<carname type="sports">Lamborghini 002</carname>
</supercars></cars>

```

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

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: singh
Nick Name: jazz
Marks: 90
End Element :student

```

JAVA SAX PARSER - QUERY XML DOCUMENT

Demo Example

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

```

<?xml version="1.0"?>
<ltclass>

```

```

<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.java {
    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

```

JAVA SAX PARSER - CREATE XML DOCUMENT

It is better to use StAX parser for creating XML than using SAX parser. Please refer the [Java StAX Parser](#) section for the same.

JAVA SAX PARSER - MODIFY XML DOCUMENT

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 JDOM PARSER - OVERVIEW

JDOM is an open source, java based library to parse XML document and it is typically java

developer friendly API. It is java optimized, it uses java collection like List and Arrays. It works with DOM and SAX APIs and combines the best of the two. It is of low memory footprint and is nearly as fast as SAX.

Environment Setup

In order to use JDOM parser, you should have jdom.jar in your application's classpath. Download [jdom-2.0.5.zip](#).

When to use?

You should use a JDOM 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
- You are a java developer and want to leverage java optimized parsing of XML.

What you get?

When you parse an XML document with a JDOM parser, you get the flexibility to get back a tree structure that contains all of the elements of your document without impacting the memory footprint of the application. The JDOM provides a variety of utility functions you can use to examine the contents and structure of the document in case document is well structured and its structure is known.

Advantages

JDOM gives java developers flexibility and easy maintainability of xml parsing code. It is light weight and quick API.

JDOM classes

The JDOM defines several Java classes. Here are the most common classes:

- **Document** - Represents the entire XML document. A Document object is often referred to as a DOM tree.
- **Element** - Represents an XML element. Element object has methods to manipulate its child elements, its text, attributes and namespaces.
- **Attribute** Represents an attribute of an element. Attribute has method to get and set the value of attribute. It has parent and attribute type.
- **Text** Represents the text of XML tag.
- **Comment** Represents the comments in a XML document.

Common JDOM methods

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

- **SAXBuilder.build(xmlSource())** - Build the JDOM document from the xml source.
- **Document.getRootElement()** - Get the root element of the XML.
- **Element.getName()** - Get the name of the XML node.
- **Element.getChildren()** - Get all the direct child nodes of an element.
- **Node.getChildren(Name)** - Get all the direct child nodes with a given name.
- **Node.getChild(Name)** - Get first child node with given name.

JAVA JDOM PARSER - PARSE XML DOCUMENT

Steps to Using JDOM

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

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

Import XML-related packages

```
import java.io.*;  
import java.util.*;  
import org.jdom2.*;
```

Create a DocumentBuilder

```
SAXBuilder saxBuilder = new SAXBuilder();
```

Create a Document from a file or stream

```
File inputFile = new File("input.txt");  
SAXBuilder saxBuilder = new SAXBuilder();  
Document document = saxBuilder.build(inputFile);
```

Extract the root element

```
Element classElement = document.getRootElement();
```

Examine attributes

```
//returns specific attribute  
getAttribute("attributeName");
```

Examine sub-elements

```
//returns a list of subelements of specified name  
getChildren("subelementName");  
//returns a list of all child nodes  
getChildren();  
//returns first child node  
getChild("subelementName");
```

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

```

import java.io.File;
import java.io.IOException;
import java.util.List;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;

public class JDomParserDemo {
    public static void main(String[] args) {
        try {
            File inputFile = new File("input.txt");

            SAXBuilder saxBuilder = new SAXBuilder();

            Document document = saxBuilder.build(inputFile);

            System.out.println("Root element : "
                + document.getRootElement().getName());

            Element classElement = document.getRootElement();

            List<Element> studentList = classElement.getChildren();
            System.out.println("-----");

            for (int temp = 0; temp < studentList.size(); temp++) {
                Element student = studentList.get(temp);
                System.out.println("\nCurrent Element : "
                    + student.getName());
                Attribute attribute = student.getAttribute("rollno");
                System.out.println("Student roll no : "
                    + attribute.getValue() );
                System.out.println("First Name : " + student.getChild("firstname").getText());
                System.out.println("Last Name : " + student.getChild("lastname").getText());
                System.out.println("Nick Name : " + student.getChild("nickname").getText());
                System.out.println("Marks : " + student.getChild("marks").getText());
            }
        } catch (JDOMException e) {
            e.printStackTrace();
        } catch (IOException ioe) {
            ioe.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
```

JAVA JDOM PARSER - QUERY XML DOCUMENT

Demo Example

Here is the input xml file we need to query:

```
<?xml version="1.0"?>
<ltcars>
  <ltsupercars company="Ferrari">
    <ltcarname type="formula one">>Ferarri 101</carname>
    <ltcarname type="sports car">>Ferarri 201</carname>
    <ltcarname type="sports car">>Ferarri 301</carname>
  </supercars>
  <ltsupercars company="Lamborghini">
    <ltcarname>>Lamborghini 001</carname>
    <ltcarname>>Lamborghini 002</carname>
    <ltcarname>>Lamborghini 003</carname>
  </supercars>
  <ltluxurycars company="Benteley">
    <ltcarname>>Benteley 1</carname>
    <ltcarname>>Benteley 2</carname>
    <ltcarname>>Benteley 3</carname>
  </luxurycars>
</cars>
```

Demo Example:

QueryXmlFileDemo.java

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;

public class QueryXmlFileDemo {
```



```

public static void main(String[] args) {
    try {
        File inputFile = new File("input.txt");

        SAXBuilder saxBuilder = new SAXBuilder();
        Document document = saxBuilder.build(inputFile);

        System.out.println("Root element : "
            + document.getRootElement().getName());

        Element classElement = document.getRootElement();

        List<Element> supercarList = classElement.getChildren("supercars");
        System.out.println("-----");

        for (int temp = 0; temp < supercarList.size(); temp++) {
            Element supercarElement = supercarList.get(temp);
            System.out.println("\nCurrent Element : "
                + supercarElement.getName());
            Attribute attribute = supercarElement.getAttribute("company");
            System.out.println("company : "
                + attribute.getValue() );
            List<Element> carNameList = supercarElement.getChildren("carname");
            for (int count = 0;
                count < carNameList.size(); count++) {
                Element carElement = carNameList.get(count);
                System.out.print("car name : ");
                System.out.println(carElement.getText());
                System.out.print("car type : ");
                Attribute typeAttribute = carElement.getAttribute("type");
                if(typeAttribute !=null)
                    System.out.println(typeAttribute.getValue());
                else{
                    System.out.println("");
                }
            }
        }
    }catch(JDOMException e){
        e.printStackTrace();
    }catch(IOException ioe){
        ioe.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 : Lamborghini
car name : Lamborghini 001
car type :
car name : Lamborghini 002
car type :
car name : Lamborghini 003
car type :

```

JAVA JDOM PARSER - CREATE XML DOCUMENT

Demo Example

Here is the XML we need to create:

```
<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <supercars company="Ferrari">
    <carname type="formula one">Ferrari 101</carname>
    <carname type="sports">Ferrari 202</carname>
  </supercars>
</cars>
```

Demo Example:

CreateXmlFileDemo.java

```
import java.io.IOException;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.output.Format;
import org.jdom2.output.XMLOutputter;

public class CreateXmlFileDemo {
    public static void main(String[] args) {

        try{
            //root element
            Element carsElement = new Element("cars");
            Document doc = new Document(carsElement);

            //supercars element
            Element supercarElement = new Element("supercars");
            supercarElement.setAttribute(new Attribute("company", "Ferrari"));

            //supercars element
            Element carElement1 = new Element("carname");
            carElement1.setAttribute(new Attribute("type", "formula one"));
            carElement1.setText("Ferrari 101");

            Element carElement2 = new Element("carname");
            carElement2.setAttribute(new Attribute("type", "sports"));
            carElement2.setText("Ferrari 202");

            supercarElement.addContent(carElement1);
            supercarElement.addContent(carElement2);

            doc.getRootElement().addContent(supercarElement);

            XMLOutputter xmlOutput = new XMLOutputter();

            // display ml
            xmlOutput.setFormat(Format.getPrettyFormat());
            xmlOutput.output(doc, System.out);
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

This would produce the following result:

```
<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <supercars company="Ferrari">
    <carname type="formula one">Ferrari 101</carname>
    <carname type="sports">Ferrari 202</carname>
  </supercars>
</cars>
```

JAVA JDOM PARSER - MODIFY XML DOCUMENT

Demo Example

Here is the input text 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

```
import java.io.File;
import java.io.IOException;
import java.util.List;

import org.jdom2.Attribute;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import org.jdom2.output.Format;
import org.jdom2.output.XMLOutputter;

public class ModifyXMLFileDemo {
    public static void main(String[] args) {
        try {
            File inputFile = new File("input.txt");
            SAXBuilder saxBuilder = new SAXBuilder();
            Document document = saxBuilder.build(inputFile);
            Element rootElement = document.getRootElement();

            //get first supercar
            Element supercarElement = rootElement.getChild("supercars");

            // update supercar attribute
            Attribute attribute = supercarElement.getAttribute("company");
            attribute.setValue("Lamborghini");

            // loop the supercar child node
            List<Element> list = supercarElement.getChildren();
            for (int temp = 0; temp < list.size(); temp++) {
                Element carElement = list.get(temp);
                if("Ferrari 101".equals(carElement.getText())){
                    carElement.setText("Lamborghini 001");
                }
                if("Ferrari 202".equals(carElement.getText())){
                    carElement.setText("Lamborghini 002");
                }
            }
        } catch (JDOMException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

    }
}

//get all supercars element
List<Element> supercarslist = rootElement.getChildren();
for (int temp = 0; temp < supercarslist.size(); temp++) {
    Element tempElement = supercarslist.get(temp);
    if("luxurycars".equals(tempElement.getName())){
        rootElement.removeContent(tempElement);
    }
}

XMLOutputter xmlOutput = new XMLOutputter();

// display xml
xmlOutput.setFormat(Format.getPrettyFormat());
xmlOutput.output(document, System.out);
} catch (JDOMException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

This would produce the following result:

```

<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <supercars company="Lamborghini">
    <carname type="formula one">Lamborghini 001</carname>
    <carname type="sports">Lamborghini 002</carname>
  </supercars>
</cars>

```

JAVA STAX PARSER - OVERVIEW

StAX is a JAVA based API to parse XML document in a similar way as SAX parser does. But there are two major difference between the two APIs

- StAX is a PULL API where as SAX is a PUSH API. It means in case of StAX parser, client application need to ask StAX parser to get information from XML whenever it needs but in case of SAX parser, client application is required to get information when SAX parser notifies the client application that information is available.
- StAX API can read as well as write XML documents. Using SAX API, xml can be only be read.

Environment Setup

In order to use StAX parser, you should have stax.jar in your application's classpath. Download [stax-1.2.0.jar](#).

Following are the features of StAX API

- 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" reader which acts as an iterator and iterates over the event to get the required information. Another reader available is "cursor" reader which acts as a pointer to xml nodes.

- As the events are identified, xml elements can be retrieved from the event object and can be processed further.

When to use?

You should use a StAX 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 StAX 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

XMLStreamReader Class

This class provide iterator of events which can be used to iterate over events as they occur while parsing the XML document

- **StartElement asStartElement()** - used to retrieve value and attributes of element.
- **EndElement asEndElement()** - called at the end of a element.
- **Characters asCharacters()** - can be used to obtain characters such a CDATA, whitespace etc.

XMLStreamWriter Class

This interface specifies methods for creating an event.

- **add(Event event)** - Add event containing elements to XML.

XMLStreamReader Class

This class provide iterator of events which can be used to iterate over events as they occur while parsing the XML document

- **int next()** - used to retrieve next event.
- **boolean hasNext()** - used to check further events exists or not
- **String getText()** - used to get text of an element
- **String getLocalName()** - used to get name of an element

XMLStreamWriter Class

This interface specifies methods for creating an event.

- **writeStartElement(String localName)** - Add start element of given name.
- **writeEndElement(String localName)** - Add end element of given name.

- **writeAttribute(String localName, String value)** - Write attribute to an element.

JAVA STAX PARSER - PARSE XML DOCUMENT

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>
```

StAXParserDemo.java

```
package com.tutorialspoint.xml;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Iterator;

import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;

public class StAXParserDemo {
    public static void main(String[] args) {
        boolean bFirstName = false;
        boolean bLastName = false;
        boolean bNickName = false;
        boolean bMarks = false;
        try {
            XMLInputFactory factory = XMLInputFactory.newInstance();
            XMLEventReader eventReader =
                factory.createXMLEventReader(
                    new FileReader("input.txt"));

            while(eventReader.hasNext()){
                XMLEvent event = eventReader.nextEvent();
                switch(event.getEventType()){
                    case XMLStreamConstants.START_ELEMENT:
                        StartElement startElement = event.asStartElement();
                        String qName = startElement.getName().getLocalPart();
                        if (qName.equalsIgnoreCase("student")) {
```

```

        System.out.println("Start Element : student");
        Iterator<Attribute> attributes = startElement.getAttributes();
        String rollNo = attributes.next().getValue();
        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;
    }
    break;
case XMLStreamConstants.CHARACTERS:
    Characters characters = event.asCharacters();
    if(bFirstName){
        System.out.println("First Name: "
            + characters.getData());
        bFirstName = false;
    }
    if(bLastName){
        System.out.println("Last Name: "
            + characters.getData());
        bLastName = false;
    }
    if(bNickName){
        System.out.println("Nick Name: "
            + characters.getData());
        bNickName = false;
    }
    if(bMarks){
        System.out.println("Marks: "
            + characters.getData());
        bMarks = false;
    }
    break;
case XMLStreamConstants.END_ELEMENT:
    EndElement endElement = event.asEndElement();
    if(endElement.getName().getLocalPart().equalsIgnoreCase("student")){
        System.out.println("End Element : student");
        System.out.println();
    }
    break;
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (XMLStreamException e) {
    e.printStackTrace();
}
}
}

```

This would produce the following result:

```

Start Element : student
Roll No : 393
First Name: dinkar
Last Name: kad
Nick Name: dinkar
Marks: 85
End Element : student

```

```

Start Element : student
Roll No : 493
First Name: Vaneet
Last Name: Gupta

```

Nick Name: vinni
Marks: 95
End Element : student

Start Element : student
Roll No : 593
First Name: jasvir
Last Name: singn
Nick Name: jazz
Marks: 90
End Element : student

JAVA STAX PARSER - QUERY XML DOCUMENT

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>
```

StAXParserDemo.java

```
package com.tutorialspoint.xml;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Iterator;

import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.Characters;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;

public class StAXQueryDemo {
    public static void main(String[] args) {
        boolean bFirstName = false;
        boolean bLastName = false;
        boolean bNickName = false;
        boolean bMarks = false;
        boolean isRequestRollNo = false;
        try {
            XMLInputFactory factory = XMLInputFactory.newInstance();
            XMLEventReader eventReader =
```



```

factory.createXMLStreamReader(
    new FileReader("input.txt"));

String requestedRollNo = "393";
while(eventReader.hasNext()){
    XMLEvent event = eventReader.nextEvent();
    switch(event.getEventType()){
        case XMLStreamConstants.START_ELEMENT:
            StartElement startElement = event.asStartElement();
            String qName = startElement.getName().getLocalPart();
            if (qName.equalsIgnoreCase("student")) {
                Iterator<Attribute> attributes = startElement.getAttributes();
                String rollNo = attributes.next().getValue();
                if(rollNo.equalsIgnoreCase(requestedRollNo)){
                    System.out.println("Start Element : student");
                    System.out.println("Roll No : " + rollNo);
                    isRequestRollNo = true;
                }
            } 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;
            }
            break;
        case XMLStreamConstants.CHARACTERS:
            Characters characters = event.asCharacters();
            if(bFirstName && isRequestRollNo){
                System.out.println("First Name: "
                    + characters.getData());
                bFirstName = false;
            }
            if(bLastName && isRequestRollNo){
                System.out.println("Last Name: "
                    + characters.getData());
                bLastName = false;
            }
            if(bNickName && isRequestRollNo){
                System.out.println("Nick Name: "
                    + characters.getData());
                bNickName = false;
            }
            if(bMarks && isRequestRollNo){
                System.out.println("Marks: "
                    + characters.getData());
                bMarks = false;
            }
            break;
        case XMLStreamConstants.END_ELEMENT:
            EndElement endElement = event.asEndElement();
            if(endElement.getName().getLocalPart().equalsIgnoreCase("student") &&
isRequestRollNo){
                System.out.println("End Element : student");
                System.out.println();
                isRequestRollNo = false;
            }
            break;
    }
}
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (XMLStreamException e) {
    e.printStackTrace();
}
}

```

```
}
```

This would produce the following result:

```
Start Element : student
Roll No : 393
First Name: dinkar
Last Name: kad
Nick Name: dinkar
Marks: 85
End Element : student
```

JAVA STAX PARSER - CREATE XML DOCUMENT

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:

StAXCreateXMLDemo.java

```
package com.tutorialspoint.xml;

import java.io.IOException;
import java.io.StringWriter;

import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

public class StAXCreateXMLDemo {
    public static void main(String[] args) {
        try {
            StringWriter stringWriter = new StringWriter();

            XMLOutputFactory xMLOutputFactory = XMLOutputFactory.newInstance();
            XMLStreamWriter xMLStreamWriter =
xMLOutputFactory.createXMLStreamWriter(stringWriter);

            xMLStreamWriter.writeStartDocument();
            xMLStreamWriter.writeStartElement("cars");

            xMLStreamWriter.writeStartElement("supercars");
            xMLStreamWriter.writeAttribute("company", "Ferrari");

            xMLStreamWriter.writeStartElement("carname");
            xMLStreamWriter.writeAttribute("type", "formula one");
            xMLStreamWriter.writeCharacters("Ferrari 101");
            xMLStreamWriter.writeEndElement();

            xMLStreamWriter.writeStartElement("carname");
            xMLStreamWriter.writeAttribute("type", "sports");
            xMLStreamWriter.writeCharacters("Ferrari 202");
            xMLStreamWriter.writeEndElement();

            xMLStreamWriter.writeEndElement();
            xMLStreamWriter.writeEndDocument();

            xMLStreamWriter.flush();
```

```

        XMLStreamWriter.close();

        String xmlString = stringWriter.getBuffer().toString();

        stringWriter.close();

        System.out.println(xmlString);

    } catch (XMLStreamException e) {
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        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>

```

JAVA STAX PARSER - MODIFY XML DOCUMENT

Demo Example

Here is the XML we need to modify:

```

<?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>singh</lastname>
    <nickname>jazz</nickname>
    <marks>90</marks>
</student>

```

Demo Example:

StAXModifyDemo.java

```

package com.tutorialspoint.xml;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;

```

```

import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;

import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;
import org.jdom2.output.Format;
import org.jdom2.output.XMLOutputter;

public class StAXModifyDemo {
    public static void main(String[] args) {
        try {
            XMLInputFactory factory = XMLInputFactory.newInstance();
            XMLStreamReader eventReader =
                factory.createXMLStreamReader(
                    new FileReader("input.txt"));
            SAXBuilder saxBuilder = new SAXBuilder();
            Document document = saxBuilder.build(new File("input.txt"));
            Element rootElement = document.getRootElement();
            List<Element> studentElements = rootElement.getChildren("student");
            while(eventReader.hasNext()){
                XMLEvent event = eventReader.nextEvent();
                switch(event.getEventType()){
                    case XMLStreamConstants.START_ELEMENT:
                        StartElement startElement = event.asStartElement();
                        String qName = startElement.getName().getLocalPart();

                        if (qName.equalsIgnoreCase("student")) {
                            Iterator<Attribute> attributes = startElement.getAttributes();
                            String rollNo = attributes.next().getValue();
                            if(rollNo.equalsIgnoreCase("393")){
                                //get the student with roll no 393
                                for(int i=0;i < studentElements.size();i++){
                                    Element studentElement = studentElements.get(i);

                                    if(studentElement.getAttribute("rollno").getValue().equalsIgnoreCase("393")){
                                        studentElement.removeChild("marks");
                                        studentElement.addContent(new Element("marks").setText("80"));
                                    }
                                }
                            }
                        }
                        break;
                }
            }
            XMLOutputter xmlOutput = new XMLOutputter();
            // display xml
            xmlOutput.setFormat(Format.getPrettyFormat());
            xmlOutput.output(document, System.out);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (XMLStreamException e) {
            e.printStackTrace();
        } catch (JDOMException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

This would produce the following result:

```
<student rollno="393">
  <firstname>dinkar</firstname>
  <lastname>kad</lastname>
  <nickname>dinkar</nickname>
  <marks>80</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>singh</lastname>
  <nickname>jazz</nickname>
  <marks>90</marks>
</student>
```

JAVA XPATH PARSER - OVERVIEW

The XPath is an official recommendation of the World Wide Web Consortium (W3C). It defines a language to find information in an XML file. It is used to traverse elements and attributes of an XML document. XPath provides various type of expressions which can be used to enquire relevant information from the XML document.

What is XPath?

- **Structure Definations** - XPath defines the parts of an XML document like element, attribute, text, namespace, processing-instruction, comment, and document nodes
- **Path Expressions** XPath provides powerful path expressions select nodes or list of nodes in XML documents.
- **Standard Functions** XPath provides a rich library of standard functions for manipulation of string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values etc.
- **Major part of XSLT** XPath is one of the major element in XSLT standard and is must have knowledge in order to work with XSLT documents.
- **W3C recommendation** XPath is official recommendation of World Wide Web Consortium (W3C).

Here is the input text 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>
```

XPath Expressions

XPath uses a path expression to select node or list of nodes from an xml document. Following is the list of useful paths and expression to select any node/ list of nodes from an xml document.

Expression	Description
node-name	Select all nodes with the given name "nodename"
/	Selection starts from the root node
//	Selection starts from the current node that match the selection
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes
student	Example: Selects all nodes with the name "student"
class/student	Example: Selects all student elements that are children of class
//student	Selects all student elements no matter where they are in the document

Predicates

Predicate are used to find specifi node or a node containing specific value and are defined using [...].

Expression	Result
/class/student[1]	Selects the first student element that is the child of the class element.
/class/student[last()]	Selects the last student element that is the child of the class element.
/class/student[last()-1]	Selects the last but one student element that is the child of the class element.
//student[@rollno='493']	Selects all the student elements that have an attribute named rollno with a value of '493'

JAVA XPATH PARSER - PARSE XML DOCUMENT

Steps to Using XPath

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

- Import XML-related packages.
- Create a DocumentBuilder
- Create a Document from a file or stream
- Create an Xpath object and an XPath path expression

- Compile the XPath expression using XPath.compile() and get a list of nodes by evaluating the compiled expression via XPath.evaluate()
- Iterate over the list of nodes.
- Examine attributes
- Examine sub-elements

Import XML-related packages

```
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
import javax.xml.xpath.*;
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);
```

Build XPath

```
XPath xPath = XPathFactory.newInstance().newXPath();
```

Prepare Path expression and evaluate it

```
String expression = "/class/student";
NodeList nodeList = (NodeList) xPath.compile(expression).evaluate(doc,
XPathConstants.NODESET);
```

Iterate over NodeList

```
for (int i = 0; i < nodeList.getLength(); i++) {
    Node nNode = nodeList.item(i);
    ...
}
```

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 text 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>singh</lastname>
    <nickname>jazz</nickname>
    <marks>90</marks>
  </student>
</class>
```

Demo Example:

XPathParserDemo.java

```
package com.tutorialspoint.xml;

import java.io.File;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.xml.sax.SAXException;

public class XPathParserDemo {
    public static void main(String[] args) {
        try {
            File inputFile = new File("input.txt");
            DocumentBuilderFactory dbFactory
                = DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder;

            dBuilder = dbFactory.newDocumentBuilder();

            Document doc = dBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();

            XPath xPath = XPathFactory.newInstance().newXPath();

            String expression = "/class/student";
            NodeList nodeList = (NodeList) xPath.compile(expression).evaluate(doc,
XPathConstants.NODESET);
            for (int i = 0; i < nodeList.getLength(); i++) {
                Node nNode = nodeList.item(i);
                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 (ParserConfigurationException e) {
    e.printStackTrace();
} catch (SAXException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (XPathExpressionException e) {
    e.printStackTrace();
}
}
}

```

This would produce the following result:

```

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 : singh
Nick Name : jazz
Marks : 90

```

JAVA XPATH PARSER - QUERY XML DOCUMENT

Demo Example

Here is the input text file we need to query:

```

<?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:

XPathParserDemo.java

```

package com.tutorialspoint.xml;

import java.io.File;
import java.io.IOException;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import org.xml.sax.SAXException;

public class XPathParserDemo {
    public static void main(String[] args) {
        try {
            File inputFile = new File("input.txt");
            DocumentBuilderFactory dbFactory
                = DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder;

            dBuilder = dbFactory.newDocumentBuilder();

            Document doc = dBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();

            XPath xPath = XPathFactory.newInstance().newXPath();

            String expression = "/class/student[@rollno='493']";
            NodeList nodeList = (NodeList) xPath.compile(expression).evaluate(doc,
XPathConstants.NODESET);
            for (int i = 0; i < nodeList.getLength(); i++) {
                Node nNode = nodeList.item(i);
                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 (ParserConfigurationException e) {
    e.printStackTrace();
} catch (SAXException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (XPathExpressionException e) {
    e.printStackTrace();
}
}
}

```

This would produce the following result:

```

Current Element :student
Student roll no : 493
First Name : Vaneet
Last Name : Gupta
Nick Name : vinni
Marks : 95

```

JAVA XPATH PARSER - CREATE XML DOCUMENT

XPath parser is used to to navigate XML Document only. It is better to use DOM parser for creating XML. Please refer the Java DOM Parser section for the same.

JAVA XPATH PARSER - MODIFY XML DOCUMENT

XPath parser is used to to navigate XML Document only. It is better to use DOM parser for modifying XML. Please refer the Java DOM Parser section for the same.

JAVA DOM4J PARSER - OVERVIEW

DOM4J is an open source, java based library to parse XML document and it is highly flexible, high-performance, and memory-efficient API. It is java optimized, it uses java collection like List and Arrays. It works with DOM, SAX, XPath and XSLT. It can parse large XML document with very low memory footprint.

Environment Setup

In order to use DOM4J parser, you should have dom4j-1.6.1.jar and jaxen.jar in your application's classpath. Download [dom4j-1.6.1.zip](#).

When to use?

You should use a DOM4J 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
- You are a java developer and want to leverage java optimized parsing of XML.

What you get?

When you parse an XML document with a DOM4J parser, you get the flexibility to get back a tree structure that contains all of the elements of your document without impacting the memory footprint of the application. The DOM4J provides a variety of utility functions you can use to examine the contents and structure of the document in case document is well structured and its structure is known. DOM4J uses XPath expression to navigate through the XML document.

Advantages

DOM4J gives java developers flexibility and easy maintainability of xml parsing code. It is light weight and quick API.

DOM4J classes

The DOM4J defines several Java classes. Here are the most common classes:

- **Document** - Represents the entire XML document. A Document object is often referred to as a DOM tree.
- **Element** - Represents an XML element. Element object has methods to manipulate its child elements, its text, attributes and namespaces.
- **Attribute** Represents an attribute of an element. Attribute has method to get and set the value of attribute. It has parent and attribute type.
- **Node** Represents Element, Attribute or ProcessingInstruction

Common DOM4J methods

When you are working with the DOM4J, there are several methods you'll use often:

- **SAXReader.read(xmlSource())** - Build the DOM4J document from the xml source.
- **Document.getRootElement()** - Get the root element of the XML.
- **Element.node(index)** - Get the XML node at particular index in the element.
- **Element.attributes()** - Get all the attributes of an element.
- **Node.valueOf(@Name)** - Get the value of an attribute with given name of the element.

JAVA DOM4J PARSER - PARSE XML DOCUMENT

Steps to Using DOM4J

Following are the steps used while parsing a document using DOM4J Parser.

- Import XML-related packages.

- Create a SAXReader
- Create a Document from a file or stream
- Get the required nodes using XPath Expression by calling document.selectNodes()
- Extract the root element
- Iterate over the list of nodes.
- Examine attributes
- Examine sub-elements

Import XML-related packages

```
import java.io.*;  
import java.util.*;  
import org.dom4j.*;
```

Create a DocumentBuilder

```
SAXBuilder saxBuilder = new SAXBuilder();
```

Create a Document from a file or stream

```
File inputFile = new File("input.txt");  
SAXBuilder saxBuilder = new SAXBuilder();  
Document document = saxBuilder.build(inputFile);
```

Extract the root element

```
Element classElement = document.getRootElement();
```

Examine attributes

```
//returns specific attribute  
valueOf("@attributeName");
```

Examine sub-elements

```
//returns first child node  
selectSingleNode("subelementName");
```

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>
```

```

        &lt;lastname>singn</lastname>
        &lt;nickname>jazz</nickname>
        &lt;marks>90</marks>
    </student>
</class>

```

Demo Example:

DOM4JParserDemo.java

```

package com.tutorialspoint.xml;

import java.io.File;
import java.util.List;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.Node;
import org.dom4j.io.SAXReader;

public class DOM4JParserDemo {
    public static void main(String[] args) {
        try {
            File inputFile = new File("input.txt");
            SAXReader reader = new SAXReader();
            Document document = reader.read( inputFile );

            System.out.println("Root element :"+
                + document.getRootElement().getName());

            Element classElement = document.getRootElement();

            List<Node> nodes = document.selectNodes("/class/student" );
            System.out.println("-----");
            for (Node node : nodes) {
                System.out.println("\nCurrent Element :"+
                    + node.getName());
                System.out.println("Student roll no : "+
                    + node.valueOf("@rollno" ));
                System.out.println("First Name : " +
                    node.selectSingleNode("firstname").getText());
                System.out.println("Last Name : " +
                    node.selectSingleNode("lastname").getText());
                System.out.println("First Name : " +
                    node.selectSingleNode("nickname").getText());
                System.out.println("Marks : " + node.selectSingleNode("marks").getText());
            }
        } catch (DocumentException e) {
            e.printStackTrace();
        }
    }
}

```

This would produce the following result:

```

Root element :class
-----

Current Element :student
Student roll no :
First Name : dinkar
Last Name : kad
First Name : dinkar
Marks : 85

Current Element :student

```

Student roll no :
First Name : Vaneet
Last Name : Gupta
First Name : vinni
Marks : 95

Current Element :student
Student roll no :
First Name : jasvir
Last Name : singn
First Name : jazz
Marks : 90

JAVA DOM4J PARSER - QUERY XML DOCUMENT

Demo Example

Here is the input xml file we need to parse:

```
<?xml version="1.0"?>
<ltclass>
  <ltstudent rollno="393">
    <ltfirstname>dinkar</firstname>
    <ltlastname>kad</lastname>
    <ltnickname>dinkar</nickname>
    <ltmarks>85</marks>
  </student>
  <ltstudent rollno="493">
    <ltfirstname>Vaneet</firstname>
    <ltlastname>Gupta</lastname>
    <ltnickname>vinni</nickname>
    <ltmarks>95</marks>
  </student>
  <ltstudent rollno="593">
    <ltfirstname>jasvir</firstname>
    <ltlastname>singn</lastname>
    <ltnickname>jazz</nickname>
    <ltmarks>90</marks>
  </student>
</class>
```

Demo Example:

DOM4JQueryDemo.java

```
package com.tutorialspoint.xml;

import java.io.File;
import java.util.List;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.Node;
import org.dom4j.io.SAXReader;

public class DOM4JQueryDemo {
    public static void main(String[] args) {
        try {
            File inputFile = new File("input.txt");
            SAXReader reader = new SAXReader();
            Document document = reader.read( inputFile );

            System.out.println("Root element :"+
                + document.getRootElement().getName());

            Element classElement = document.getRootElement();
```

```

List<Node> nodes = document.selectNodes("/class/student[@rollno='493']" );
System.out.println("-----");
for (Node node : nodes) {
    System.out.println("\nCurrent Element : "
        + node.getName());
    System.out.println("Student roll no : "
        + node.valueOf("@rollno") );
    System.out.println("First Name : " +
node.selectSingleNode("firstname").getText());
    System.out.println("Last Name : " +
node.selectSingleNode("lastname").getText());
    System.out.println("First Name : " +
node.selectSingleNode("nickname").getText());
    System.out.println("Marks : " + node.selectSingleNode("marks").getText());
}
} catch (DocumentException e) {
    e.printStackTrace();
}
}
}

```

This would produce the following result:

```

Root element :class
-----
Current Element :student
Student roll no : 493
First Name : Vaneet
Last Name : Gupta
First Name : vinni
Marks : 95

```

JAVA DOM4J PARSER - CREATE XML DOCUMENT

Demo Example

Here is the XML we need to create:

```

<?xml version="1.0" encoding="UTF-8"?>
<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 java.io.IOException;
import java.io.UnsupportedEncodingException;

import org.dom4j.Document;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;
import org.dom4j.io.OutputFormat;
import org.dom4j.io.XMLWriter;

public class DOM4JCreateXMLDemo {
    public static void main(String[] args) {
        try {
            Document document = DocumentHelper.createDocument();
            Element root = document.addElement( "cars" );

```



```

Element supercarElement= root.addElement("supercars")
    .addAttribute("company", "Ferrai");

supercarElement.addElement("carname")
    .addAttribute("type", "Ferrari 101")
    .addText("Ferrari 101");

supercarElement.addElement("carname")
    .addAttribute("type", "sports")
    .addText("Ferrari 202");

// Pretty print the document to System.out
OutputFormat format = OutputFormat.createPrettyPrint();
XMLWriter writer;
writer = new XMLWriter( System.out, format );
writer.write( document );
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

This would produce the following result:

```

<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <supercars company="Ferrari">
    <carname type="formula one">Ferrari 101</carname>
    <carname type="sports">Ferrari 202</carname>
  </supercars>
</cars>

```

JAVA DOM4J PARSER - MODIFY XML DOCUMENT

Demo Example

Here is the XML we need to modify:

```

<?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:

CreateXmlFileDemo.java

```

package com.tutorialspoint.xml;

```

```

import java.io.File;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.Iterator;
import java.util.List;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.Node;
import org.dom4j.io.OutputFormat;
import org.dom4j.io.SAXReader;
import org.dom4j.io.XMLWriter;

public class DOM4jModifyXMLDemo {
    public static void main(String[] args) {
        try {
            File inputFile = new File("input.txt");
            SAXReader reader = new SAXReader();
            Document document = reader.read( inputFile );

            Element classElement = document.getRootElement();

            List<Node> nodes = document.selectNodes("/class/student[@rollno='493']" );
            for (Node node : nodes) {
                Element element = (Element)node;
                Iterator<Element> iterator=element.elementIterator("marks");
                while(iterator.hasNext()){
                    Element marksElement=(Element)iterator.next();
                    marksElement.setText("80");
                }
            }

            // Pretty print the document to System.out
            OutputFormat format = OutputFormat.createPrettyPrint();
            XMLWriter writer;
            writer = new XMLWriter( System.out, format );
            writer.write( document );
        } catch (DocumentException e) {
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

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>
  </student>
  <student rollno="493">
    <firstname>Vaneet</firstname>
    <lastname>Gupta</lastname>
    <nickname>vinni</nickname>
    <marks>80</marks>
  </student>
  <student rollno="593">
    <firstname>jasvir</firstname>
    <lastname>singn</lastname>
  </student>
</class>

```

```
<nickname>jazz</nickname>  
<marks>90</marks>  
</student>  
</class>
```

JAVA XSLT TUTORIAL

A detailed XSLT tutorial is coming soon. A link to XSLT tutorial will be available here.

JAVA JAXB TUTORIAL

Follow detailed JAXB tutorial: [JAXB Tutorial](#)