DOM and SAX XML Parsers:

There are 3 main parsers for which I have given sample code:

- DOM Parser
- SAX Parser
- StAX Parser

XML:

```
<employees>
  <employee id="111">
    <firstName>Rakesh</firstName>
    <lastName>Mishra</lastName>
    <location>Bangalore</location>
  </employee>
  <employee id="112">
    <firstName>John</firstName>
    <lastName>Davis</lastName>
    <location>Chennai</location>
  </employee>
  <employee id="113">
    <firstName>Rajesh</firstName>
    <lastName>Sharma</lastName>
    <location>Pune</location>
  </employee>
</employees>
```

Using DOM Parser

The DOM Parser loads the complete XML content into a Tree structure. And we iterate through the Node and NodeList to get the content of the XML. The code for XML parsing using DOM parser is given below.

```
public class DOMParserDemo {

  public static void main(String[] args) throws Exception {
    //Get the DOM Builder Factory
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();

    //Get the DOM Builder
    DocumentBuilder builder = factory.newDocumentBuilder();

    //Load and Parse the XML document
    //document contains the complete XML as a Tree.
    Document document =
      builder.parse(
        ClassLoader.getSystemResourceAsStream("xml/employee.xml"));
```

```java
    List<Employee> empList = new ArrayList<>();

    //Iterating through the nodes and extracting the data.
    NodeList nodeList = document.getDocumentElement().getChildNodes();

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

      //We have encountered an <employee> tag.
      Node node = nodeList.item(i);
      if (node instanceof Element) {
        Employee emp = new Employee();
        emp.id = node.getAttributes().
            getNamedItem("id").getNodeValue();

        NodeList childNodes = node.getChildNodes();
        for (int j = 0; j < childNodes.getLength(); j++) {
          Node cNode = childNodes.item(j);

          //Identifying the child tag of employee encountered.
          if (cNode instanceof Element) {
            String content = cNode.getLastChild().
                getTextContent().trim();
            switch (cNode.getNodeName()) {
              case "firstName":
                emp.firstName = content;
                break;
              case "lastName":
                emp.lastName = content;
                break;
              case "location":
                emp.location = content;
                break;
            }
          }
        }
        empList.add(emp);
      }

    }

    //Printing the Employee list populated.
    for (Employee emp : empList) {
      System.out.println(emp);
    }

  }
}

class Employee{
  String id;
  String firstName;
  String lastName;
  String location;

  @Override
  public String toString() {
    return firstName+" "+lastName+"("+id+")"+location;
  }
}
```

Using SAX Parser

SAX Parser is different from the DOM Parser where SAX parser doesn't load the complete XML into the memory, instead it parses the XML line by line triggering different events as and when it encounters different elements like: opening tag, closing tag, character data, and comments and so on. This is the reason why SAX Parser is called an event based parser.
Along with the XML source file, we also register a handler which extends the DefaultHandler class. The DefaultHandler class provides different callbacks out of which we would be interested in:

- **startElement()** – triggers this event when the start of the tag is encountered.
- **endElement()** – triggers this event when the end of the tag is encountered.
- **characters()** – triggers this event when it encounters some text data.

The code for parsing the XML using SAX Parser is given below:

```
import java.util.ArrayList;
import java.util.List;
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) throws Exception {
    SAXParserFactory parserFactor = SAXParserFactory.newInstance();
    SAXParser parser = parserFactor.newSAXParser();
    SAXHandler handler = new SAXHandler();
    parser.parse(ClassLoader.getSystemResourceAsStream("xml/employee.xml"),
                 handler);

    //Printing the list of employees obtained from XML
    for ( Employee emp : handler.empList){
      System.out.println(emp);
    }
  }
}
/**
 * The Handler for SAX Events.
 */
class SAXHandler extends DefaultHandler {

  List<Employee> empList = new ArrayList<>();
  Employee emp = null;
  String content = null;
  @Override
  //Triggered when the start of tag is found.
  public void startElement(String uri, String localName,
                           String qName, Attributes attributes)
                           throws SAXException {

    switch(qName){
      //Create a new Employee object when the start tag is found
      case "employee":
        emp = new Employee();
        emp.id = attributes.getValue("id");
        break;
    }
  }

  @Override
  public void endElement(String uri, String localName,
                         String qName) throws SAXException {
   switch(qName){
     //Add the employee to list once end tag is found
     case "employee":
       empList.add(emp);
       break;
     //For all other end tags the employee has to be updated.
     case "firstName":
       emp.firstName = content;
       break;
     case "lastName":
       emp.lastName = content;
       break;
     case "location":
       emp.location = content;
       break;
   }
  }

  @Override
  public void characters(char[] ch, int start, int length)
          throws SAXException {
    content = String.copyValueOf(ch, start, length).trim();
  }

}

class Employee {

  String id;
  String firstName;
  String lastName;
  String location;

  @Override
  public String toString() {
    return firstName + " " + lastName + "(" + id + ")" + location;
  }
}
```

Using StAX Parser

StAX stands for Streaming API for XML and StAX Parser is different from DOM in the same way SAX Parser is. StAX parser is also in a subtle way different from SAX parser.

- The SAX Parser pushes the data but StAX parser pulls the required data from the XML.
- The StAX parser maintains a cursor at the current position in the document allows to extract the content available at the cursor whereas SAX parser issues events as and when certain data is encountered.

XMLInputFactory and XMLStreamReader are the two class which can be used to load an XML file. And as we read through the XML file using XMLStreamReader, events are generated in the form of integer values and these are then compared with the constants in XMLStreamConstants. The below code shows how to parse XML using StAX parser:

```
import java.util.ArrayList;
import java.util.List;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

public class StaxParserDemo {
  public static void main(String[] args) throws XMLStreamException {
    List<Employee> empList = null;
    Employee currEmp = null;
    String tagContent = null;
    XMLInputFactory factory = XMLInputFactory.newInstance();
    XMLStreamReader reader =
        factory.createXMLStreamReader(
        ClassLoader.getSystemResourceAsStream("xml/employee.xml"));

    while(reader.hasNext()){
      int event = reader.next();

      switch(event){
        case XMLStreamConstants.START_ELEMENT:
          if ("employee".equals(reader.getLocalName())){
            currEmp = new Employee();
            currEmp.id = reader.getAttributeValue(0);
          }
          if("employees".equals(reader.getLocalName())){
            empList = new ArrayList<>();
          }
          break;

        case XMLStreamConstants.CHARACTERS:
          tagContent = reader.getText().trim();
          break;
        case XMLStreamConstants.END_ELEMENT:
          switch(reader.getLocalName()){
            case "employee":
              empList.add(currEmp);
              break;
            case "firstName":
              currEmp.firstName = tagContent;
              break;
            case "lastName":
              currEmp.lastName = tagContent;
              break;
            case "location":
              currEmp.location = tagContent;
              break;
          }
          break;

        case XMLStreamConstants.START_DOCUMENT:
          empList = new ArrayList<>();
          break;
      }

    }

    //Print the employee list populated from XML
    for ( Employee emp : empList){
      System.out.println(emp);
    }

  }
}

class Employee{
  String id;
  String firstName;
  String lastName;
  String location;

  @Override
  public String toString(){
    return firstName+" "+lastName+"("+id+") "+location;
  }
}
```

DOM XML Parser in Java

DOM Stands for Document Object Model and it represent an XML Document into tree format which each element representing tree branches. DOM Parser creates an In Memory tree representation of XML file and then parses it, so it requires more memory and it's advisable to have increased the heap size for DOM parser in order to avoid **Java.lang.OutOfMemoryError**:java heap space .

Parsing XML file using DOM parser is quite fast if XML file is small but if you try to read a large XML file using DOM parser there is more chances that it will take a long time or even may not be able to load it completely simply because it requires lot of memory to create XML Dom Tree. Java provides support DOM Parsing and you can parse XML files in Java using DOM parser. DOM classes are in w3c.dom package while DOM Parser for Java is in JAXP (Java API for XML Parsing) package.

SAX XML Parser in Java

Difference between DOM and XML parsers in Java

SAX Stands for Simple API for XML Parsing. This is an event based XML Parsing and it parse XML file step by step so much suitable for large XML Files. SAX XML Parser fires an event when it encountered opening tag, element or attribute, and the parsing works accordingly. It’s recommended to use SAX XML parser for parsing large XML files in Java because it doesn't require to load whole XML file in Java and it can read a big XML file in small parts. Java provides support for SAX parser and you can parse any XML file in Java using SAX Parser, I have covered an example of reading XML file using SAX Parser here. One disadvantage of using SAX Parser in java is that reading XML file in Java using SAX Parser requires more code in comparison of DOM Parser.

Difference between DOM and SAX XML Parser

Here are few high-level differences between DOM parser and SAX Parser in Java:

1) DOM parser loads whole XML document in memory while SAX only loads a small part of the XML file in memory.

2) DOM parser is faster than SAX because it access whole XML document in memory.

3) SAX parser in Java is better suitable for large XML file than DOM Parser because it doesn't require much memory.

4) DOM parser works on Document Object Model while SAX is an event based XML parser.

That’s all on the difference between SAX and DOM parsers in Java, now it’s up to you on which XML parser you going to choose. I recommend using DOM parser over SAX parser if XML file is small enough and go with SAX parser if you don’t know the size of XML files to be processed or they are large.