Java SAX Parser - Parse XML Document

Java SAX(Simple API for XML) parser is an API in Java to parse XML documents. SAX parser is an event based parser and uses a Handler class to handle the events. The call back methods such as startElement(), characters(), endElement() etc., are implemented inside the Handler class to obtain the details of elements and their attributes. These call back methods are called when the parser identifies the respective events.

Parse XML Using Java SAX parser

Following are the steps we need to follow to parse an XML document in Java using SAX parser –

- **Step 1:** Implementing a Handler class
- **Step 2:** Creating a SAXParser Object
- **Step 3:** Reading the XML
- **Step 4:** Creating object for Handler class
- **Step 5:** Parsing the XML Document
- **Step 6:** Retrieving the Elements

Step 1: Implementing a Handler class

Application program must implement a handler class to handle the events inside the XML document. After implementing the Handler class, it must be registered with the SAX parser.

As discussed in the previous chapter, the DefaultHandler class implements ContentHandler interface. It has the methods, startDocument(), endDocument(), startElement(), endElement() and characters() functions that help us parse the XML documents. We write the code inside these methods according to our requirement.

```
class UserHandler extends DefaultHandler {

   public void startDocument() {
      ...
   }

   public void startElement(String uri, String localName, String qName,
Attributes attributes) {
      ...
```

```
   }

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

   public void endElement(String uri, String localName, String qName) {
      ...
   }

   public void endDocument() {
      ...
   }
}
```

Step 2: Creating a SAXParser Object

The SAXParserFactory class is used to create a new factory instance which in turn is used to create the SAXParser object as follows –

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
```

Step 3: Reading the XML

Read the XML file by specifying the proper file path as follows –

```
File xmlFile = new File("input.xml");
```

Instead of reading files, we can create an InputStream of the XML content as follows –

```
StringBuilder xmlBuilder = new StringBuilder();
xmlBuilder.append(""<?xml version="1.0"?> <rootElement></rootElement>"");
ByteArrayInputStream inputStream = new ByteArrayInputStream(
xmlBuilder.toString().getBytes("UTF-8"));
```

Step 4: Creating object for Handler class

Create an object for the already implemented UserHandler class in first step as follows –

```
UserHandler userHandler = new UserHandler();
```

Step 5: Parsing the XML Document

The SAXParser class has the parse() method that takes two arguments, one is the file and the other is the DefaultHandler object. This function parses the given file as XML document using the functions implemented inside the DefaultHandler class.

```
saxParser.parse(xmlFile, userHandler);
```

The SAXParser class also has the function parse() that takes the content as InputStream –

```
saxParser.parse(inputStream, userHandler);
```

Step 6: Retrieving the Elements

After following the above five steps, we can now easily retrieve the required information about the elements. We should write the required code inside the methods of our Handler class in first step. All the methods available inside the ContentHandler interface are discussed in the previous chapter and in this chapter, we will implement these methods to retrieve the basic information about elements such as element name, text content and attributes.

Retrieving Element Name

Element name can be obtained from the **startElement()** method of ContentHandler interface. The third argument of this method is the name of the Element and it is of String type. We can implement this method in our Handler class and get the name of an Element.

Example

In the following example, we have taken XML content in the form of a String using StringBuilder class and converted into bytes using ByteArrayInputStream.

In the UserHandler class, we have implemented the startElement() method and printed the name of the Element. Since, we have only single element in the XML content, that becomes the root element of the document.

</> Open Compiler

```
import java.io.ByteArrayInputStream;
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;

//Implementing UserHandler Class
class UserHandler extends DefaultHandler{
   public void startElement(String uri, String localName, String qName,
Attributes attributes)
      throws SAXException {
      System.out.println("Root element is "+qName);
   }
}

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

         //Creating a SAXParser Object
         SAXParserFactory factory = SAXParserFactory.newInstance();
         SAXParser saxParser = factory.newSAXParser();

         //Reading the XML
         StringBuilder xmlBuilder = new StringBuilder();
         xmlBuilder.append("<college>XYZ College</college>");
         ByteArrayInputStream input = new
ByteArrayInputStream(xmlBuilder.toString().getBytes("UTF-8"));

         //Creating UserHandler object
         UserHandler userhandler = new UserHandler();

         //Parsing the XML Document
         saxParser.parse(input, userhandler);

      }  catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Root Element name, 'college' is printed on the output screen.

```
Root element is college
```

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

Retrieving TextContent

To retrieve text content of an element, we have **characters()** method in ContentHandler interface. There is character array, start and length arguments in this method. As soon as the parser sees the content after ">" symbol, this method is called. The start argument carries the index of the first character after ">" symbol and length has the number of characters before it encounters "<" symbol.

Example

The following **college.xml** file has a single sub element, "department" with text content "Computer Science". Let us write a Java program to retrieve this text content along with element names using SAX API.

```
<college>
   <department>Computer Science</department>
</college>
```

The UserHandler class inherits DefaultHandler and we have implemented startElement(), endElement() and characters() method. When the parser sees the text content inside department element, this method is called and we are printing it on the console.

```
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;

//Implementing UserHandler Class
class UserHandler extends DefaultHandler {
   public void startElement( String uri, String localName, String qName,
Attributes attributes)
      throws SAXException {
      System.out.println("Start Element : " + qName);
   }
```

```
   public void endElement(String uri, String localName, String qName) {
      System.out.println("End Element : " + qName);
   }
   public void characters(char[] ch, int start, int length) throws
SAXException{
      System.out.println("Text Content : " + new String(ch, start, length));
   }
}
public class RetrieveTextContent {
    public static void main(String args[]) {
       try {

          //Creating a SAXParser Object
          SAXParserFactory factory = SAXParserFactory.newInstance();
          SAXParser saxParser = factory.newSAXParser();

          //Reading the XML
          File xmlFile = new File("college.xml");

          //Creating UserHandler object
          UserHandler userHandler = new UserHandler();

          //Parsing the XML Document
          saxParser.parse(xmlFile, userHandler);

       } catch(Exception e) {
          e.printStackTrace();
       }
    }
}
```

The text content for department element is displayed. As there is no text content inside the "college" element, it is left blank.

```
Start Element : college
Text Content :

Start Element : department
Text Content : Computer Science
End Element : department
Text Content :
```

```
End Element : college
```

Retrieving Attributes

The method startElement() has **Attributes** as last argument which has the list of attributes inside the current Element. The **getValue("attr_name")** function inside the Attributes interface is used to get the value of the specified attribute.

Example

We have added few more department elements to our "college.xml" file and also added an attribute "deptcode" to each of the departments. Let us write a java program to retrieve all the elements along with their attributes.

```
<?xml version = "1.0"?>
<college>
   <department deptcode = "DEP_CS23">
      <name>Computer Science</name>
      <staffCount>20</staffCount>
   </department>
   <department deptcode = "DEP_EC34">
      <name>Electrical and Electronics</name>
      <staffCount>23</staffCount>
   </department>
   <department deptcode = "DEP_MC89">
      <name>Mechanical</name>
      <staffCount>15</staffCount>
   </department>
</college>
```

The following Java program implements startElement() and characters() methods in UserHandler class. We have initialised two boolean variables to let us notified about deptcode and staffCount attributes in department element, so that we can use these to print the attributes in characters() method.

```
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;
```

```
//Implementing UserHandler Class
class UserHandler extends DefaultHandler{

   boolean hasDeptName=false;
   boolean hasStaffCount=false;

   public void startElement(String uri, String localName, String qName,
Attributes attributes) throws SAXException{

      if(qName.equals("college")) {
         System.out.println("Root Element : "+qName + "\n");
      }
      if(qName.equals("department")) {
         System.out.println("Current Element : "+qName);
         System.out.println("Department code : "+
attributes.getValue("deptcode"));
      }
      if(qName.equals("name")) {
         hasDeptName=true;
      }
      if(qName.equals("staffCount")) {
         hasStaffCount=true;
      }
   }

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

      if(hasDeptName) {
         System.out.println("Department Name : "+ new String(ch, start,
length));
         hasDeptName=false;
      }
      if(hasStaffCount) {
         System.out.println("Staff Count : "+ new String(ch, start, length) +
"\n");
         hasStaffCount=false;
      }
   }
}
```

```
public class ParseAttributesSAX {
   public static void main(String args[]) {
      try {

         //Creating a DocumentBuilder Object
         SAXParserFactory factory = SAXParserFactory.newInstance();
         SAXParser saxParser = factory.newSAXParser();

         //Reading the XML
         File xmlFile = new File("college.xml");

         //Creating UserHandler object
         UserHandler userHandler = new UserHandler();

         //Parsing the XML Document
         saxParser.parse(xmlFile, userHandler);

      } catch(Exception e) {
               e.printStackTrace();
      }
   }
}
```

The ouput window displays names of each element along with the attributes.

```
Root Element : college

Current Element : department
Department code : DEP_CS23
Department Name : Computer Science
Staff Count : 20

Current Element : department
Department code : DEP_EC34
Department Name : Electrical and Electronics
Staff Count : 23

Current Element : department
Department code : DEP_MC89
```

```
Department Name : Mechanical
Staff Count : 15
```