Java provides numerous ways to parse an XML file. For example, parsing an XML file using DOM parser, SAX parser or StAX parser.

Before getting into the details on how to parse XML files using SAX parser, let’s first see what is the difference between parsing through different parsers and when to choose one over the other.

**SAX Parser** – SAX is an acronym for Simple API for XML. SAX Parser parses the XML file line by line and triggers events when it encounters opening tag, closing tag or character data in XML file. This is why SAX parser is called an event-based parser

**DOM Parser** – DOM is an acronym for Document Object Model. Unlike SAX parser DOM parser loads the complete XML file into memory and creates a tree structure where each node in the tree represents a component of XML file. With DOM parser you can create nodes, remove nodes, change their contents and traverse the node hierarchy. DOM provides maximum flexibility while working with XML files but it comes with a cost of potentially large memory footprint and significant processor requirements in case of large XML files

**StAX Parser** – StAX is an acronym for Streaming API for XML. Stream-based parsers are very useful when your application has memory limitations. For example, a cell phone running Java Micro Edition. Similarly, if your application needs to process several requests simultaneously, for example an application server, StAX parser should be used.

**Stream-based parsing can further be classified as:**

**Pull Parsing** – In pull parsing, client application calls for methods on an XML parsing library when it needs to interact with an XML infoset. In other words, client only gets XML data when it explicitly asks for it.

**Push Parsing** – In push parsing, it is the XML parser which pushes XML data to the client, when it encounters elements in an XML infoset. In other words, parser sends the data to application irrespective of the application being ready to use it or not.

**XML File**

Below is the XML file that we are going to parse and construct Java objects

< dvd name="Bourne Series">

< movies>

< movie>

< name>The Bourne Identity< /name>

< directors>Doug Liman< /directors>

< runtime>119< /runtime>

< cast>Matt Damon, Franka Potente< /cast>

< released>2002< /released>

< /movie>

< movie>

< name>The Bourne Supremacy< /name>

< directors>Paul Greengrass< /directors>

< runtime>108< /runtime>

< cast>Matt Damon, Franka Potente, Joan Allen< /cast>

< released>2004< /released>

< /movie>

< movie>

< name>The Bourne Ultimatum< /name>

< directors>Paul Greengrass< /directors>

< runtime>115< /runtime>

< cast>Matt Damon, Edgar Ramirez, Joan Allen< /cast>

< released>2007< /released>

< /movie>

< movie>

< name>The Bourne Legacy< /name>

< directors>Tony Gilroy< /directors>

< runtime>135< /runtime>

< cast>Jeremy Renner, Rachel Weisz, Edward Norton< /cast>

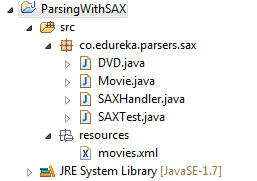
< released>2012< /released>

< /movie>

< /movies>

< /dvd>

**Project Structure**  
Here is the screen shot of project structure in Eclipse IDE

[](https://www.edureka.co/blog/wp-content/uploads/2015/06/project-structure.png)

Here is the DVD class which holds a list of movie objects

package co.edureka.parsers.sax;

import java. util. List;

public class DVD {

private String name;

private List movies;

public String get Name () {

return name;

}

public void setName (String name) {

this.name = name;

}

public List getMovies () {

return movies;

}

public void setMovies (List movies) {

this.movies = movies;

}

}

Movie object have properties like name, directors, runtime(duration) of movie, released year and cast of the movie

package co.edureka.parsers.sax;

public class Movie {

private String name;

private String directors;

private int runtime;

private int released;

private String cast;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDirectors() {

return directors;

}

public void setDirectors(String directors) {

this.directors = directors;

}

public int getRuntime() {

return runtime;

}

public void setRuntime(int runtime) {

this.runtime = runtime;

}

public int getReleased() {

return released;

}

public void setReleased(int released) {

this.released = released;

}

public String getCast() {

return cast;

}

public void setCast(String cast) {

this.cast = cast;

}

@Override

public String toString() {

return "Movie [name=" + name + ", directors=" + directors

+ ", runtime=" + runtime + ", released=" + released + ", cast="

+ cast + "]";

}

}

**Implementing the SAX Handler:**

We are going to extend the org.xml.sax.helpers. DefaultHandler class which provides many call back methods and will override the following methods:

**startElement()** – This method gets called when start of a tag is encountered

**endElement()** – This  method  gets called when end of a tag is encountered

**characters ()** – This method gets called when some text data is encountered

Note: There are many other call back methods like startDocument(), endDocument() etc. that can be overridden if required.

package co.edureka.parsers.sax;

import java.util.ArrayList;

import java.util.List;

import org.xml.sax.Attributes;

import org.xml.sax.helpers.DefaultHandler;

public class SAXHandler extends DefaultHandler{

DVD dvd=new DVD();

ListmovieList=new ArrayList();

Movie movie=null;

String content =null;

public void startElement(String namespaceURI,String localName,String qname,Attributes attributes){

if(qname.equals("dvd")){

String dvdName=attributes.getValue("name");

dvd.setName(dvdName);

}else if(qname.equals("movie")){

movie=new Movie();

}

}

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

switch(qname){

case "movie": movieList.add(movie);

break;

case "name" : movie.setName(content);

break;

case "directors" : movie.setDirectors(content);

break;

case "released" : movie.setReleased(Integer.parseInt(content));

break;

case "runtime" : movie.setRuntime(Integer.parseInt(content));

break;

case "cast" : movie.setCast(content);

break;

case "dvd" : dvd.setMovies(movieList);

break;

}

}

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

content=new String(ch, start, length);

}

public DVD getDVD(){

return dvd;

}

}

**Testing the SAX Handler**  
Now let’s test our SAXHandler. Below is the test class SAXTest where we first get an instance of SAXParser  from SAXParserFactory and call the parse method which takes two arguments: A File and a handler instance.

package co.edureka.parsers.sax;

import java.io.IOException;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.List;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.parsers.SAXParser;

import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.SAXException;

public class SAXTest {

public static void main(String[] args) throws ParserConfigurationException, SAXException, IOException {

SAXParserFactory parserFactor = SAXParserFactory.newInstance();

SAXParser parser = parserFactor.newSAXParser();

SAXHandler handler = new SAXHandler();

Path path = Paths.get("src/resources", "movies.xml");

parser.parse(path.toFile(),handler);

DVD dvd=handler.getDVD();

List movies=dvd.getMovies();

System.out.println("DVD Name : "+dvd.getName());

for(Movie movie:movies){

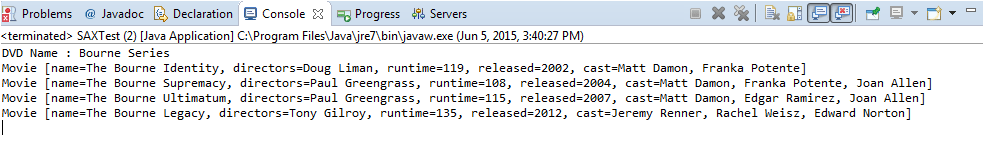
System.out.println(movie);

}

}

}

On executing the SAXTest class you will get the below output:

[](https://www.edureka.co/blog/wp-content/uploads/2015/06/output.png)

The general advantages of SAX include:

* The nature of a streaming model means that you need far less memory to process large XML documents.

* You do not have to process the entire document. Use call back procedures to identify and respond to only the XML elements you are interested in.

* You can halt the parse at any time.

* You can parse an XML document that is not well formed.

* SAX provides call back procedures that let you to provide more error handling.