# NODE: determining the type

DOM SAX

```
switch (node.getNodeType()) {
        case Node.ELEMENT_NODE;  ...; break;
        case Node.ATTRIBUTE_NODE;  ...; break;
        case Node.TEXT_NODE;  ...; break;
        case Node.CDATA_SECTION_NODE;  ...; break;
        case Node.ENTITY_REFERENCE_NODE;  ...; break;
        case Node.PROCESSING_INSTRUCTION;  ...; break;
        case Node.COMMENT_NODE;  ...; break;
        case Node.DOCUMENT_NODE;  ...; break;
        case Node.DOCUMENT_TYPE_NODE;  ...; break;
        case Node.DOCUMENT_FRAGMENT_NODE;  ...; break;
        case Node.NOTATION_NODE;  ...; break;
        default: throw (new Exception());
}
```

# DOM example

DOM SAX

```
import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;   // parser uses SAX methods to build DOM object
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;

public class CountDom {
  public static void main(String[ ] arg) throws Exception {
     if (arg.length != 1) {
        System.err.println("Usage: cmd filename (file must exist)");
        System.exit(1);
    }
      Node node = readFile(new File(arg[0]));
      System.out.println(arg + " elementCount: " +
getElementCount(node));
    }
  }
```

# DOM example

DOM SAX

```
public static Document readFile(File file) throws Exception {
    Document doc;
    try {
      DocumentBuilderFactory dbf =
 DocumentBuilderFactory.newInstance();
      dbf.setValidating(false);
      DocumentBuilder db = dbf.newDocumentBuilder();
      doc = db.parse(file);
      return doc;
    } catch (SAXParseException ex) {
      throw (ex);
    } catch (SAXException ex) {
      Exception x = ex.getException();   // get underlying Exception
      throw ((x == null) ? ex : x);
    }
  }
```

Parse File,
Return Document

# DOM example

DOM SAX

```
public static int getElementCount(Node node) {
  if (null == node)      return 0;
  int sum = 0;
  boolean isElement = (node.getNodeType() == Node.ELEMENT_NODE);
  if (isElement)             sum = 1;
  NodeList children = node.getChildNodes();
  if (null == children)      return sum;

  for (int i = 0; i < children.getLength(); i++) {
    sum += getElementCount(children.item(i));   // recursive call
  }
  return sum;
 }
}
```

use DOM methods to count elements:
for each subtree  if the root is an Element,
 set sum to 1, else to 0;
 add element count of all children of the root to sum

# Alternatives to DOM


*"Build a better mousetrap, and the world will beat a path to your door."*
*--Emerson*

# Alternatives to DOM


JDOM: Java DOM (see http://www.jdom.org).

The standard DOM is a very simple data structure that intermixes text nodes, element nodes, processing instruction nodes, CDATA nodes, entity references, and several other kinds of nodes. That makes it difficult to work with in practice, because you are always sifting through collections of nodes, discarding the ones you don't need into order to process the ones you are interested in. JDOM, on the other hand, creates a tree of *objects* from an XML structure. The resulting tree is much easier to use, and it can be created from an XML structure without a compilation step.

DOM4J: DOM for Java (see http://www.dom4j.org/)

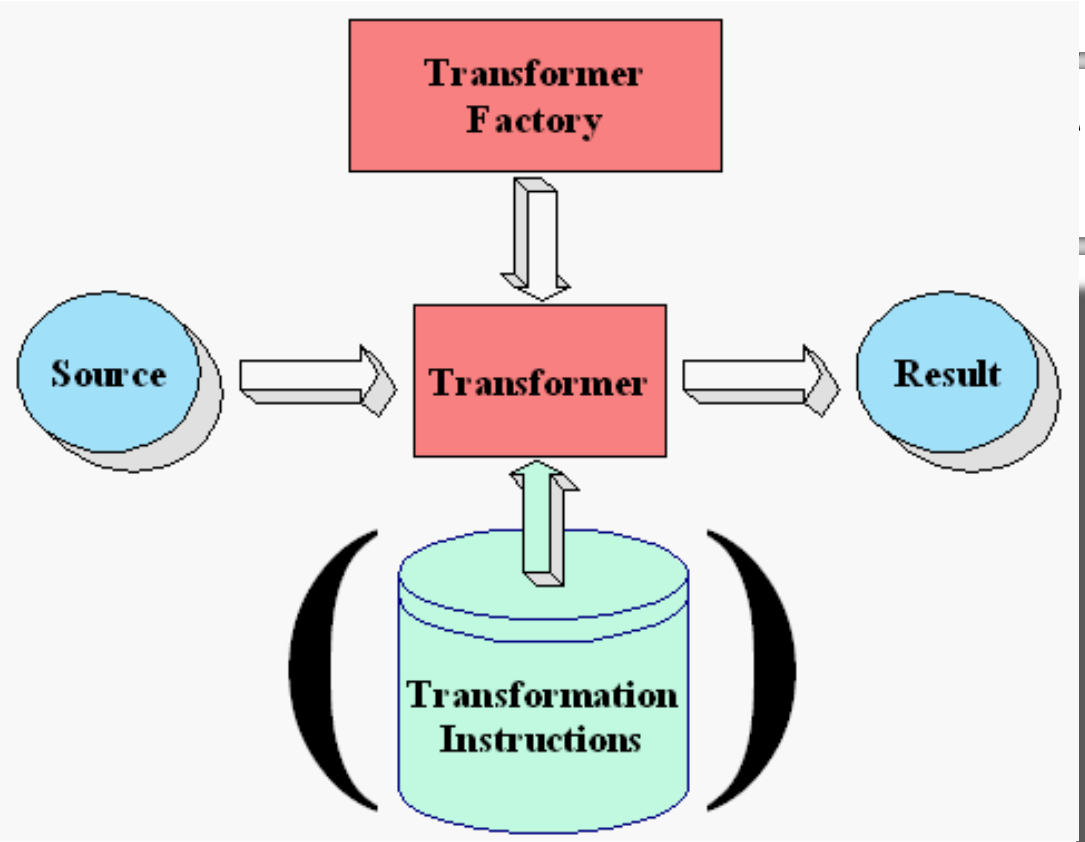
***dom4j*** **is an easy to use, open source library for working with XML, XPath and XSLT on the Java platform using the Java Collections Framework and with full support for DOM, SAX and JAXP. (last release 2010, Java5)**

# Transformations

DOM SAX

*Using XSLT from Java*

# TrAX




```
TransformerFactory tf = TransformerFactory .newInstance();
 StreamSource xslSS=new StreamSource("source.xsl");
 StreamSource xmlSS=new StreamSource("source.xml");
 Transformer t=tf.newTrasformer(xslSS);
 t.transform(xmlSS,new StreamResult(new
          FileOutputStream("out.html");
```

java –Djavax.xml.transform.TransformerFactory=
org.apache.xalan.processor.TrasformerFactoryImpl MyClass

# xml.transform packages

DOM SAX

| *Package* | *Description* |
|---|---|
| javax.xml.transform | Defines the TransformerFactory and Transformer classes, which you use to get a object capable of doing transformations. After creating a transformer object, you invoke its transform() method, providing it with an input (source) and output (result). |
| javax.xml.transform.dom | Classes to create input (source) and output (result) objects from a DOM. |
| javax.xml.transform.sax | Classes to create input (source) from a SAX parser and output (result) objects from a SAX event handler. |
| javax.xml.transform.stream | Classes to create input (source) and output (result) objects from an I/O stream. |

# TrAX main classes

DOM SAX

javax.xml.transform.Transformer

transform(Source xmls, Result output)

javax.xml.transform.sax.SAXResult implements Result

javax.xml.transform.sax.SAXSource implements Source

javax.xml.transform.stream.StreamResult implements Result

javax.xml.transform.stream.StreamSource implements Source

javax.xml.transform.dom.DOMResult implements Result

javax.xml.transform. dom.DOMSource implements Source

# Other Java-XML APIs

DOM SAX

**Java Architecture for XML Binding (JAXB) provides a convenient way to bind an XML schema to a representation in Java code.**

**See also:**

- **JAX-WS**
- **JAX-SWA**
- **JAX- RPC**
- **SAAJ**
- **XML –Digital Signatures**
- **ecc.**