Università degli Studi di Modena e Reggio Emilia

Facoltà di Ingegneria – Reggio Emilia

CORSO DI TECNOLOGIE E APPLICAZIONI WEB

Le librerie Java per manipolare XML

Ing. Marco Mamei

Anno Accademico 2004-2005

JAVA e XML

Il linguaggio di programmazione Java e il metalinguaggio per la descrizione dei dati XML sono due tecnologie dalle eccezionali doti tecniche, ma che prese singolarmente presentano delle limitazioni.

Java non fornisce alcuna specifica per la rappresentazione delle informazioni, per cui porta gli sviluppatori a creare ogni volta nuovi formati proprietari per la descrizione e la presentazione delle stesse. Inoltre tecnologie come le Servlet o le JSP non forniscono da sole una vera separazione tra le attività di creazione e presentazione dei contenuti, in quanto devono produrre contemporaneamente (nello stesso documento) sia il contenuto del documento sia i tag HTML per la visualizzazione dello stesso.

XML d'altro canto è semplicemente un metalinguaggio. In assenza di un reale supporto da parte di programmi (parser, XSL processors, etc.) è solamente una specifica che non trova alcun riscontro nella pratica.

Java e XML sono due tecnologie che si completano vicendevolmente.

- Da un lato XML fornisce una sintassi standard e uniforme per la rappresentazione dei dati. Questo consente agli sviluppatori Java di produrre un codice portabile che si

interfaccia con dati rappresentati in una maniera altrettanto portabile.
- D'altro lato Java fornisce, grazie al supporto nativo per la manipolazione di stringhe, il supporto delle tabelle hash e degli URL, l'ambiente ideale per sviluppare delle API che consentano di manipolare l'XML.

Le **JAXP (Java API for XML Processing)** possono essere definite - un po' impropriamente - le librerie standard Java per trattare l'XML.

Più correttamente le JAXP sono un insieme di interfacce standard. Diverse librerie per la gestione del XML possono implementare le JAXP.

In questo modo le JAXP offrono un metodo standard per accedere al XML indipendente dalla librerie usata per fare il lavoro vero e proprio.

Le JAXP si compongono di tre "librerie":
- **SAX (Simple API for XML parsing)**
- **DOM (Document Object Model)**
- **XSLT (XML Style Sheet Translation)**.

SAX è una libreria di basso livello che consente di effettuare il parsing di un documento XML attraverso l'invocazione di metodi qualora, nella scansione del documento, si riscontri l'apertura o la chiusura di un tag, l'inizio o la fine del documento stesso, o altri eventi significativi.

DOM è una libreria di alto livello che consente di trasformare un documento XML in un albero di oggetti che riflette la struttura del

documento. L'albero può poi essere manipolato e riconvertito in un nuovo documento XML.

XSLT è una libreria che consente di applicare un foglio di stile XSL a un documento XML.

In questa guida vedremo attraverso alcuni esempi di codice come usare le **JAXP**. In particolare, useremo l'implementazione offerta dalla Sun e scaricabile gratuitamente all'indirizzo:

http://java.sun.com/xml/downloads/javaxmlpack.html

Le API sono contenute all'interno del **Java XML Pack** un file ZIP di circa 10MB che contiene tutto quello che la Sun propone per la gestione dell'XML. Una volta estratto il file all'interno della directory
\java_xml_pack-summer-02_01\jaxp-1.2_01
Si trovano i seguenti sei file JAR
dom.jar, sax.jar, xercesImpl.jar, jaxp-api.jar, xalan, xsltc.jar
Questi vanno inseriti nell'CLASSPATH del ambiente Java che utilizzate.

In particolare, i package più significativi sono:

- **javax.xml.parsers:** qui ci sono le interfacce JAXP vere e proprie. i.e. solo le interfacce non le implementazioni.
- **org.xml.sax:** qui ci sono le classi che implementano SAX
- **org.w3c.dom:** qui ci sono le classi che implementano DOM
- **javax.xml.transform:** qui ci sono le classi che implementano XSLT

Siamo a questo punto pronti per iniziare a scrivere del codice!
Come unica nota gli esempi che seguono solo volutamente molto semplici allo scopo di dimostrare solo l'uso delle JAXP e poco

altro. E' chiaro che queste tecniche trovano la loro piu' completa espressione quando si combinano con Servlet, JSP e quant' altro Java offre per la gesione del Web.

Negli esempi che seguono utilizzeremo i seguenti file **prova.xml** e **prova.xsl** come documenti su cui provare le manipolazioni. Vale la pena di ricordare comunque che queste librerie sono generiche per trattare documenti di markup e quindi si possono applicare anche a documenti HTML.

prova.xml

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

<!DOCTYPE PAPERS [
  <!ELEMENT PAPERS (PAPER+)>
  <!ELEMENT PAPER (TITLE,AUTHOR+,PUB,FILENAME)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT AUTHOR (#PCDATA)>
  <!ELEMENT PUB (#PCDATA)>
  <!ELEMENT FILENAME (#PCDATA)>
]>

<PAPERS>

<PAPER>
<TITLE>
Programming Context-Aware Pervasive Computing Applications in TOTA
</TITLE>
<AUTHOR>Marco Mamei</AUTHOR>
<AUTHOR>Franco Zambonelli</AUTHOR>
<AUTHOR>Letizia Leonardi</AUTHOR>
<PUB>ICSE 2003</PUB>
<FILENAME>tota_icse.pdf</FILENAME>
</PAPER>

<PAPER>
<TITLE>
Tuples on The Air: a Middleware for Context-Aware Computing
</TITLE>
<AUTHOR>Marco Mamei</AUTHOR>
<AUTHOR>Franco Zambonelli</AUTHOR>
<AUTHOR>Letizia Leonardi</AUTHOR>
<PUB>ICDCS 2003</PUB>
<FILENAME>tota_icdcs.pdf</FILENAME>
</PAPER>

</PAPERS>
```

prova.xsl

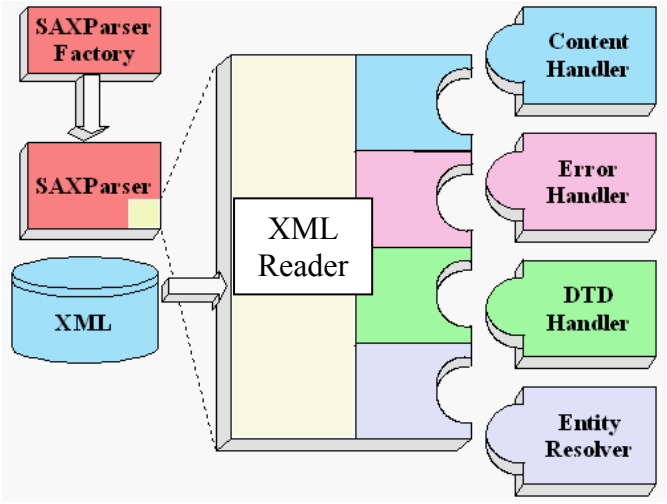
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0" >

<xsl:template match="/">
<HTML>
<BODY>
<xsl:apply-templates/>
</BODY>
</HTML>
</xsl:template>

<xsl:template match="TITLE">
<B>
<xsl:value-of select="."/>
</B><BR/>
</xsl:template>

<xsl:template match="AUTHOR">
<xsl:value-of select="."/>
</xsl:template>

<xsl:template match="PUB">
<BR/>
<xsl:value-of select="."/>
<BR/>
</xsl:template>

<xsl:template match="FILENAME">
<xsl:value-of select="."/>
<BR/>
</xsl:template>


</xsl:stylesheet>
```

SAX (Simple API for XML parsing)

L'idea che sta alla base di queste API e' molto semplice: si legge un file XML e ogni volta che si trovano degli elementi interessanti (tag, commenti, etc.) vengono invocati dei metodi (cosiddetti metodi di callback) per gestire l'evento che si e' verificato.
Es. Quando ricevo il tag <AUTHOR> invoca un metodo che esegue la stampa del contenuto.

Questa e' un API molto veloce e che non richiede di avere tutto il documento XML in memoria: un flusso dati XML può essere parsificato al volo man mano che viene ricevuto.

Vediamo le componenti di questa API più in dettaglio.



SAXParserFactory è usata per creare un'istanza del parser SAX. (perché non fare la **new** del parser SAX direttamente? Che bisogno c'è di una classe in più? ...vedi dopo…)

SAXParser è l'oggetto fondamentale. Incapsula un oggetto **XMLReader** che legge il repository XML, e implementa il metodo **parse()**.

Una volta invocato il metodo **parse()** l'**XMLReader** inizia a scandire il repository XML e invoca i metodi di callback implementati dalla nostra applicazione.

In particolare la nostra applicazione deve implementare le seguenti interfacce:

- **ContentHandler:** gestisce gli eventi di inizio e fine dei tag, attributi, etc.
- **ErrorHandler:** gestisce gli errori, dunque specifica il comportamento del parser di fronte ad anomalie
- **DTDHandler:** gestisce l'analisi della DTD
- **EntityResolver:** gestisce entità che si riferiscono a URL

In pratica però è già presente una classe **DefaultHandler** che implementa le interfacce con metodi vuoti (l'evento di callback viene ignorato). Basta ereditare da questa classe e fare l'overload dei metodi che interessano.

Vediamo un esempio, il programma seguente riceve come argomento il file XML (**prova.xml**) e stampa il suo contenuto man mano che avviene il parsing.

Esempio: MySAXPArser.java

```
import java.io.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class MySAXParser
{
 static private Writer  out;
 static StringBuffer textBuffer;

 public static void main(String argv[])
 {
  if (argv.length != 1)
  {
   System.err.println("Usage: cmd filename");
   System.exit(1);
  }

/* Qui si crea una factory che e' un oggetto atto a creare dei
parser SAX. Factory in inglese significa: fabbrica */

  SAXParserFactory factory = SAXParserFactory.newInstance();
  try
  {// Set up output stream
   out = new OutputStreamWriter(System.out, "UTF8");

/* Qui si crea un oggetto DefaultHandler che gestisce gli eventi
di call back generati dal parser. */

   DefaultHandler handler = new CallBack(out,textBuffer);

/* La factory fabbrica un parser */
   SAXParser saxParser = factory.newSAXParser();
/* collegiamo il parser all'handler degli eventi di call-back e
parsifichiamo il file */
   saxParser.parse( new File(argv[0]), handler);
  }
  catch (Throwable t)
  {
   t.printStackTrace();
  }
  System.exit(0);
 }
}
```

Esempio: L'Handler per i metodi di Callback

```
/* Questa e' la classe che gestisce gli eventi di call backs */

class CallBack extends DefaultHandler
{
 Writer  out;
 StringBuffer textBuffer;

 public CallBack(Writer  out, StringBuffer textBuffer)
 {
  this.out=out;
  this.textBuffer=textBuffer;
 }
 //============================================================
 // SAX DocumentHandler methods
 //============================================================

/* The parser call this method at the beginning of the document */
 public void startDocument() throws SAXException
 {
  nl();
  nl();
  emit("START DOCUMENT");
  nl();
  emit("<?xml version='1.0' encoding='UTF-8'?>");
 }

/* The parser call this method at the end of the document */
 public void endDocument() throws SAXException
 {
  nl();
  emit("END DOCUMENT");
  try
  {
   nl();
   out.flush();
  }
  catch (IOException e)
  {
   throw new SAXException("I/O error", e);
  }
 }

/* The parser call this method at the beginning of a tag */
 public void startElement(String namespaceURI,
                          String sName, // simple name
```

```
                            String qName, // qualified name
                            Attributes attrs) throws SAXException
 {
  echoText();
  nl();
  emit("ELEMENT: ");
  String eName = sName; // element name
  if ("".equals(eName)) eName = qName; // not namespaceAware
   emit("<"+eName);
  if (attrs != null)
  {
   for (int i = 0; i < attrs.getLength(); i++)
   {
    String aName = attrs.getLocalName(i); // Attr name
    if ("".equals(aName)) aName = attrs.getQName(i);
     nl();
    emit("   ATTR: ");
    emit(aName);
    emit("\t\"");
    emit(attrs.getValue(i));
    emit("\"");
   }
  }
  if (attrs.getLength() > 0) nl();
   emit(">");
 }

/* The parser call this method at the beginning of a tag */
 public void endElement(String namespaceURI,
                        String sName, // simple name
                        String qName  // qualified name
                        ) throws SAXException
 {
  echoText();
  nl();
  emit("END_ELM: ");
  String eName = sName; // element name
  if ("".equals(eName)) eName = qName; // not namespaceAware
   emit("</"+eName+">");
  }

/* The parser call this method to handle characters within tags */

  public void characters(char buf[], int offset, int len) throws
SAXException
  {
   String s = new String(buf, offset, len);
   if (textBuffer == null)
    textBuffer = new StringBuffer(s);

   else
```

```
    textBuffer.append(s);
   }

  //=========================================================
  // Utility Methods ...
  //=========================================================

  // Display text accumulated in the character buffer
  private void echoText() throws SAXException
  {
   if (textBuffer == null)
    return;
   nl();
   emit("CHARS: |");
   String s = ""+textBuffer;
   emit(s);
   emit("|");
   textBuffer = null;
  }

  // Wrap I/O exceptions in SAX exceptions, to
  // suit handler signature requirements
  private void emit(String s) throws SAXException
  {
   try
   {
    out.write(s);
    out.flush();
   }
   catch (IOException e)
   {
    throw new SAXException("I/O error", e);
   }
  }

  // Start a new line
  private void nl() throws SAXException
  {
   String lineEnd =  System.getProperty("line.separator");
   try
   {
    out.write(lineEnd);
   }
   catch (IOException e)
   {
    throw new SAXException("I/O error", e);
   }
  }
}
```

Una Piccola Digressione: Perché La Factory?

Nel codice precedente, abbiamo visto che in sintesi un parser SAX si crea in questo modo:

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

Perché non fare più semplicemente

```
SAXParser saxParser = new SAXParser();
```

L'idea che sta alla base delle JAXP è quella di creare delle interfacce vendor indipendent e che dunque diversi programmatori possano implementare come meglio credono. In questo modo un programma basato sulle JAXP è schermato dal parser. Il parser si può cambiare senza cambiare l'applicazione (purchè anche il nuovo parser implementi le JAXP).

C'è un problema! Il costruttore non può essere definito a livello di interfaccia. Quindi un vero disaccoppiamento è impossibile: se cambio parser devo cambiare il mio programma e riscrivere tutti i costruttori dei parser.

La factory serve proprio ad eliminare questo problema. Tutte le creazioni di parser sono mascherati dietro un metodo della factory che ritorna il parser (**newSAXParser**).

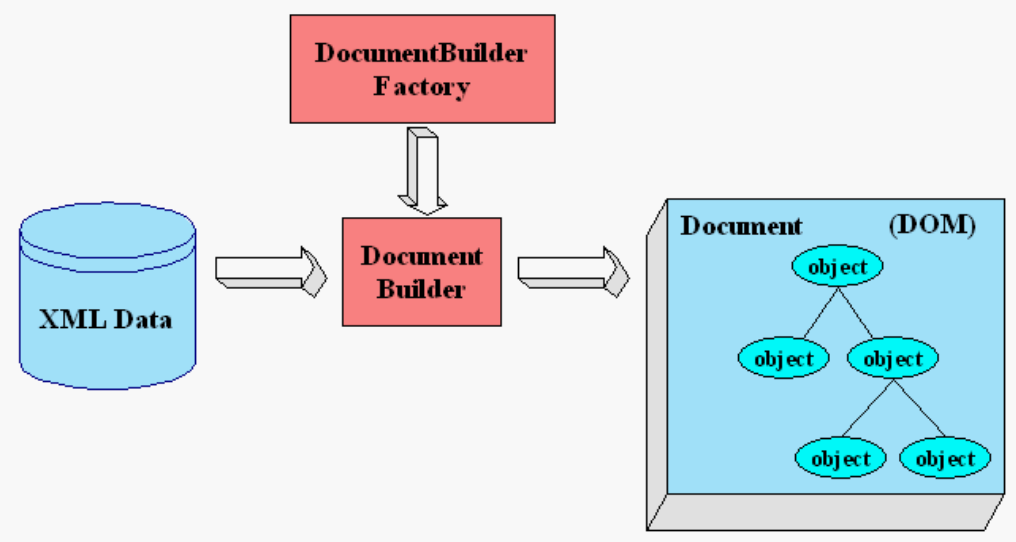
Diversi produttori di software possono implementare il metodo **newSAXParser** mettendoci dentro il loro costruttore.

Per un'applicazione che usa la factory però non cambia niente.

DOM (Document Object Model)

L'idea che sta alla base di queste API e' molto semplice: un documento XML viene convertito in un albero di oggetti che possono poi essere acceduti.
Questo programma riceve il nome del file da esaminare come argomento e lo visualizza in modo elaborato su standard output.

Vediamo le componenti di questa API più in dettaglio.



DocumentBuilderFactory è la factory usata per creare degli oggetti **DocumentBuilder**. Tramite questi ultimi un documento XML viene trasformato nell'albero **DOM** corrispettivo.

Vediamo direttamente il codice per vedere come analizzare il DOM.

Esempio: DomParser.java

```
/* imported JAXP API */
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
/* imported DOM */
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
/* imported java io */
import java.io.File;
/* imported java util */
import java.util.ArrayList;
import java.util.Iterator;

public class DomParser
{
 static Document document;

 public static void main (String args[])
 {
  if(args.length!=1)
  {
   System.err.println("Usage: java DomParser filename.xml");
   System.exit(-1);
  }

/* si crea una factory per parser DOM (costruttori di un DOM a
partire da un documento XML */
  DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
/* Questa istruzione commentata attiva il controllo di validazione
del documento nei confronti di una DTD */
  //factory.setValidating(true);
  try
  {
/* la factory crea il parser che viene usato per convertire il
file XML in un DOM */
   DocumentBuilder builder = factory.newDocumentBuilder();
   document = builder.parse(new File(args[0]));
  }
  catch(Exception e)
  {
   e.printStackTrace();
  }

  //Now document contains a DOM representation of my XML document
  //Let's get the root of the document
```

```
 Node root = getRoot(document);

 ArrayList papers = getElemChild(root);
 Iterator iter = papers.iterator();

 while(iter.hasNext())
  processPaper((Node)iter.next());
}

/* This method prints all the information about a paper */
private static void processPaper(Node paper)
{
 // This is the DTD of a paper
 // <!ELEMENT PAPER (TITLE,AUTHOR+,PUB,FILENAME)>

 String title = new String();
 String authors = new String();
 String pub = new String();
 String fileName = new String();

 ArrayList elem = getElemChild(paper);
 Iterator iter = elem.iterator();

 while(iter.hasNext())
 {
  Node n = (Node)iter.next();
  String type = n.getNodeName();
  String content = getContent(n);

  if(type.equals("TITLE"))
   title = content;
  else if(type.equals("AUTHOR"))
  {
   if(authors.equals(""))
    authors = content;
   else
    authors = new String(authors+", "+content);
  }
  else if(type.equals("PUB"))
   pub = content;
  else if(type.equals("FILENAME"))
   fileName = content;
 }

 /* let's print the paper */
 System.out.println(title);
 System.out.println(authors);
 System.out.println(pub);
 System.out.println("download at: "+fileName);
 System.out.println("*******************************");
}
```

```
 /* Il problema del DOM e' che per essere molto generale considera
nodi del DOM tutti gli elementi di un XML, quindi anche commenti,
processing instruction oltre che tag. Perciò abbiamo bisogno di
metodi a parte che selezionino solo i nodi tag */


 /* This method returns only the ELEMENT root of a document */
 private static Node getRoot(Document document)
 {
  NodeList stuff = document.getChildNodes();
  for(int i=0;i<stuff.getLength();i++)
  {
   Node node = stuff.item(i);
   if(node.getNodeType() == Node.ELEMENT_NODE)
    return node;
  }
  System.out.println("There has been some problems!");
  return null;
 }

 /* This method returns only the ELEMENT children of a node */
 private static ArrayList getElemChild(Node node)
 {
  ArrayList result = new ArrayList();
  NodeList stuff = node.getChildNodes();

  for(int i=0;i<stuff.getLength();i++)
  {
   Node n = stuff.item(i);
   if(n.getNodeType() == Node.ELEMENT_NODE)
    result.add(n);
  }
  return result;
 }

 /* This method returns the string corresponing to the only text
child of the node */
 private static String getContent(Node node)
 {
  NodeList stuff = node.getChildNodes();
  for(int i=0;i<stuff.getLength();i++)
  {
   Node n = stuff.item(i);
   if(n.getNodeType() == Node.TEXT_NODE)
    return n.getNodeValue();
  }
  return null;
 }
}
```

Esempio 2: Modificare il DOM

Vediamo altri spezzoni di codice per modificare un DOM.

```
DocumentBuilder builder = factory.newDocumentBuilder();
document = builder.newDocument();
Element root = (Element) document.createElement("rootElement");
document.appendChild(root);
root.appendChild( document.createTextNode("Some") );
root.appendChild( document.createTextNode(" ")    );
root.appendChild( document.createTextNode("text") );
```

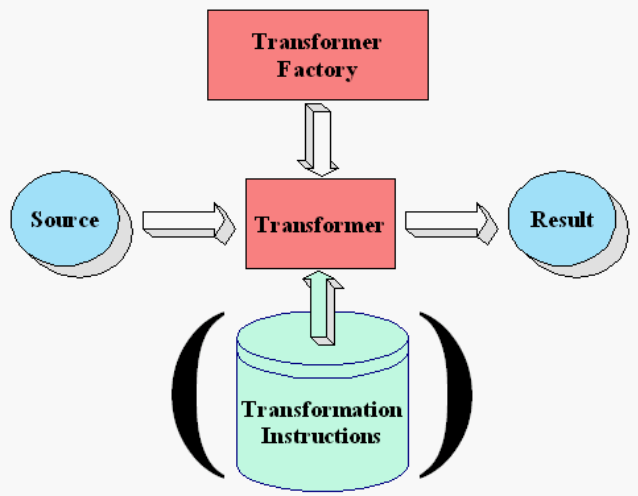
Esistono poi metodi per cancellare/modificare nodi, attributi (che sono nodi particolari), etc.

- **Node insertBefore(Node newChild, Node refChild):** inserts the node newChild before the existing child node refChild.
- **Node removeChild(Node oldChild):** removes the child node indicated by oldChild from the list of children, and returns it.
- **Node replaceChild(Node newChild, Node oldChild):** replaces the child node oldChild with newChild in the list of children, and returns the oldChild node.

XSL Transformer

Questo esempio mostra come applicare un foglio di stile XSL a un documento XML. E' chiaro che manipolazioni come quelle di un XSL possono essere fatte a livello di manipolazioni del DOM. Tuttavia in molte circostanze queste manipolazioni possono essere molto comode.

Vediamo le componenti di questa API più in dettaglio.



TransformerFactory crea un oggetto **Transformer**. Questo oggetto riceve le istruzioni per le trasformazioni (come ad esempio un documento XSLT) come parametro. Il Trasformer offre quindi un metodo **transform** che riceve in ingresso un oggetto sorgente (che può essere sia un SAX reader, un DOM o un input stream) e fornisce in uscita l'oggetto trasformato (che può essere nella forma di SAX event handler, DOM, output stream).

Curiosamente, il primo esempio di trasformazione non usa l'XSLT! Serve per passare da un formato DOM, SAX, stream all'altro, preservandone il contenuto.

Esempio 1: Format Transformer

```
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
…
try {
   File f = new File(argv[0]);
   DocumentBuilder builder = factory.newDocumentBuilder();
   document = builder.parse(f);

   // Use a Transformer for output
   TransformerFactory tFactory =
       TransformerFactory.newInstance();
   Transformer transformer = tFactory.newTransformer();

   DOMSource source = new DOMSource(document);
   StreamResult result = new StreamResult(System.out);
   transformer.transform(source, result);
```

Note:
Ovviamente, al posto del System.out come StreamResult avremmo potuto mettere altri stream. Ad esmpio, verso un file o verso una socket.
Più in generale, il metodo **transform** ha la seguente dichiarazione:

```
public void transform(Source xmlSource, Result outputTarget)
```

Source è un'interfaccia implementata da **DOMSource, SAXSource, StreamSource** che sono classi che si costruiscono rispettivamente: con un **Node** (di un DOM), un **XML Reader** (vedi dopo) e con un **File** o un **InputStream**.
Result è un'interfaccia implementata da **DOMResult, SAXResult, StreamResult** che sono classi che si costruiscono rispettivamente: con un **Node** (di un DOM), un **DefaultHandler** (di SAX) e con un **File** o un **Writer**.

Esempio2: XSLTTransformer.java

Vediamo adesso un esempio di vera trasformazione XSLT

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.xml.sax.SAXException;
import org.w3c.dom.Document;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;

import java.io.*;

public class XSLTTransformer
{
 static Document document;
 public static void main (String argv [])
 {
  if (argv.length != 2)
   System.exit (-1);
/* creiamo una factory DOM */
  DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
//factory.setValidating(true);
  try
  {
   File stylesheet = new File(argv[0]);
   File datafile   = new File(argv[1]);
/* creiamo il DOM corrispettivo del documento XML */
   DocumentBuilder builder = factory.newDocumentBuilder();
   document = builder.parse(datafile);
/* Attraverso una factory opportuna, creiamo un oggetto
Transformer atto ad applicare fogli di stile XSL a un DOM */
   TransformerFactory tFactory = TransformerFactory.newInstance();
   StreamSource stylesource = new StreamSource(stylesheet);
   Transformer transformer = tFactory.newTransformer(stylesource);
   DOMSource source = new DOMSource(document);
//StreamResult result = new StreamResult(System.out);
   File htmlFile = new File("out.html");
   StreamResult result = new StreamResult(htmlFile);
/* Effettuiamo la trasformazione del source, mettendo il risultato
in result */
   transformer.transform(source, result);
  } catch (Exception e)
    {e.printStackTrace();}
 } // main
}
```

Esempio 3: Legacy Data Sources

In questo esempio discuteremo come si possa convertire una sorgente dati non XML (come un file o un DB) in XML.

Note: si pensi a un applicazione servlet che sulla base delle informazioni di un db genera automaticamente le form html necessarie per consultare il db…

Costruiremo questo esempio per passi:

- Creiamo un parser che legge un file non-XML e lo stampa a video
- Modificheremo il programma precedente per generare eventi-SAX
- Utilizzeremo un oggetto Transformer trasformare il nostro generatore di eventi SAX in un altro formato.

• Creiamo il File di Esempio

Prendiamo come esempio un file LDAP (Lightweight Directory Access Protocol) in cui sono memorizzate informazioni relative a diversi utenti. Per ogni utente vengono archiviate le informazioni seguenti, in particolare noi ci focalizzeremo sulla parte in neretto. In ogni linea del file c'è il nome di una variabile, due punti (:), uno spazio e il valore di quella variabile.

```
dn: cn=Fred Flinstone,mail=fred@barneys.house
modifytimestamp: 20010409210816Z
cn: Fred Flinstone
xmozillanickname: Fred
mail: Fred@barneys.house
xmozillausehtmlmail: TRUE
givenname: Fred
sn: Flinstone
telephonenumber: 999-Quarry
homephone: 999-BedrockLane
facsimiletelephonenumber: 888-Squawk
pagerphone: 777-pager
cellphone: 555-cell
xmozillaanyphone: 999-Quarry
objectclass: top
objectclass: person
```

Creare un semplice Parser non-XML

Creiamo un parser super-semplice non-XML che servirà come punto di partenza:

```
import java.io.*;
public class AddressBookReader01
{
/* The main method gets the name of the file from the
command line, creates an instance of the parser, and
sets it to work parsing the file. This method will be
going away when we convert the program into a SAX
parser. (That's one reason for putting the parsing
code into a separate method.) */

    public static void main(String argv[])
    {
        // Check the arguments
        if (argv.length != 1) {
            System.err.println ("Usage: java AddressBookReader filename");
            System.exit (1);
        }
        String filename = argv[0];
        File f = new File(filename);
        AddressBookReader01 reader = new AddressBookReader01();
        reader.parse(f);
    }

/* This method operates on the File object sent to it
by the main routine. As you can see, its about as
simple as it can get! The only nod to efficiency is
the use of a BufferedReader, which can become
important when you start operating on large files. */
    public void parse(File f)
    {
        try {
            // Get an efficient reader for the file
            FileReader r = new FileReader(f);
            BufferedReader br = new BufferedReader(r);

            // Read the file and display it's contents.
            String line = br.readLine();
            while (null != (line = br.readLine())) {
              if (line.startsWith("xmozillanickname: ")) break;
            }

            output("nickname", "xmozillanickname", line);
            line = br.readLine();
            output("email",    "mail",             line);
            line = br.readLine();
            output("html",     "xmozillausehtmlmail", line);
            line = br.readLine();
            output("firstname","givenname",        line);
```

```
            line = br.readLine();
            output("lastname", "sn",                line);
            line = br.readLine();
            output("work",     "telephonenumber",   line);
            line = br.readLine();
            output("home",     "homephone",         line);
            line = br.readLine();
            output("fax",      "facsimiletelephonenumber", line);
            line = br.readLine();
            output("pager",    "pagerphone",        line);
            line = br.readLine();
            output("cell",     "cellphone",         line);

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

/* The output method contains the smarts about the structure of a line. Starting from the right It takes 3 arguments. The first argument gives the method a name to display, so we can output "html" as a variable name, instead of "xmozillausehtmlmail". The second argument gives the variable name stored in the file (xmozillausehtmlmail). The third argument gives the line containing the data. The routine then strips off the variable name from the start of the line and outputs the desired name, plus the data. */

```
    void output(String name, String prefix, String line)
    {
        int startIndex = prefix.length() + 2;  // 2=length of ": "
        String text = line.substring(startIndex);
        System.out.println(name + ": " + text);
    }
}
```

Questo è il risultato dell'elaborazione

```
nickname: Fred
email: Fred@barneys.house
html: TRUE
firstname: Fred
lastname: Flintstone
work: 999-Quarry
home: 999-BedrockLane
fax: 888-Squawk
pager: 777-pager
cell: 555-cell
```

Modficare il parser per generare eventi SAX

In questo esempio trasformiamo il nostro parser in un XMLReader che possa dunque essere usato all'intrno di un SAXReader. Per questo, anziche stampare delle stringhe a video invocheremo i metodi di call-back che avevamo visto nel SAX.

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.AttributesImpl;

/* now we subclass XMLReader */

public class AddressBookReader02 implements XMLReader
{

/* The SAX ContentHandler is the thing that is going to
get the SAX events the parser generates. Here we have to
fire the call-back methods */
    ContentHandler handler;

    // We're not doing namespaces, and we have no attributes on our elements.
    String nsu = "";  // NamespaceURI
    Attributes atts = new AttributesImpl();
    // XML Root
    String rootElement = "addressbook";
    // To indetnt the document for readability
    String indent = "\n    ";

/** Parse the input */
    public void parse(InputSource input) throws IOException, SAXException
    {
        try {
            // Get an efficient reader for the file

            java.io.Reader r = input.getCharacterStream();

            BufferedReader br = new BufferedReader(r);

            // Read the file and display it's contents.
            String line = br.readLine();
            while (null != (line = br.readLine())) {
              if (line.startsWith("xmozillanickname: ")) break;
            }

            if (handler==null) {
              throw new SAXException("No content handler");
            }
```

/* Here we invoke the callback methods on the ContentHandler. In particolar here we call start document, and start of the root element */

```
            handler.startDocument();
            handler.startElement(nsu, rootElement, rootElement, atts);

            output("nickname", "xmozillanickname", line);
            line = br.readLine();
            output("email",    "mail",             line);
            line = br.readLine();
            output("html",     "xmozillausehtmlmail", line);
            line = br.readLine();
            output("firstname","givenname",        line);
            line = br.readLine();
            output("lastname", "sn",               line);
            line = br.readLine();
            output("work",     "telephonenumber",  line);
            line = br.readLine();
            output("home",     "homephone",        line);
            line = br.readLine();
            output("fax",      "facsimiletelephonenumber", line);
            line = br.readLine();
            output("pager",    "pagerphone",       line);
            line = br.readLine();
            output("cell",     "cellphone",        line);

// This is only used to improve readability of the output.
        handler.ignorableWhitespace("\n".toCharArray(),
                                    0, // start index
                                    1  // length
                                    );
          handler.endElement(nsu, rootElement, rootElement);
          handler.endDocument();

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

/* The output method fires call back methods related to single tags */

```
    void output(String name, String prefix, String line)

    throws SAXException

    {
      int startIndex = prefix.length() + 2; // 2=length of ": " after the name
      String text = line.substring(startIndex);

      int textLength = line.length() - startIndex;
      handler.ignorableWhitespace(indent.toCharArray(),
                                  0, // start index
                                  indent.length()
                                  );
      handler.startElement(nsu, name, name /*"qName"*/, atts);
      handler.characters(line.toCharArray(),
                         startIndex,
                         textLength);
```

```
      handler.endElement(nsu, name, name);

    }
```

/* set and get of the content handler */

```
    /** Allow an application to register a content event handler. */
    public void setContentHandler(ContentHandler handler) {
      this.handler = handler;
    }

    /** Return the current content handler. */
    public ContentHandler getContentHandler() {
      return this.handler;
    }
```

/* Although we do not use these methods, they must be implemented – at least with null methods – since they belong to the XMLReader interface */

```
    /** Allow an application to register an error event handler. */
    public void setErrorHandler(ErrorHandler handler)
    { }
    /** Return the current error handler. */
    public ErrorHandler getErrorHandler()
    { return null; }
    /** Parse an XML document from a system identifier (URI). */
    public void parse(String systemId)
    throws IOException, SAXException
    { }
    /** Return the current DTD handler. */
    public DTDHandler getDTDHandler()
    { return null; }
    /** Return the current entity resolver. */
    public EntityResolver getEntityResolver()
    { return null; }
    /** Allow an application to register an entity resolver. */
    public void setEntityResolver(EntityResolver resolver)
    { }
    /** Allow an application to register a DTD event handler. */
    public void setDTDHandler(DTDHandler handler)
    { }
    /** Look up the value of a property. */
    public Object getProperty(java.lang.String name)
    { return null; }
    /** Set the value of a property. */
    public void setProperty(java.lang.String name, java.lang.Object value)
    { }
    /** Set the state of a feature. */
    public void setFeature(java.lang.String name, boolean value)
    { }
    /** Look up the value of a feature. */
    public boolean getFeature(java.lang.String name)
    { return false; }}
```

Trasformare il Parser in una SAXSource

Vediamo infine come incapsulare il nostro SAXSource in un Transformer per ottenere l'output:

```
// Use a Transformer for output
Transformer transformer = tFactory.newTransformer();
// Create the sax "parser".
AddressBookReader saxReader = new AddressBookReader();
// Use the parser as a SAX source for input
FileReader fr = new FileReader(f);
BufferedReader br = new BufferedReader(fr);
InputSource inputSource = new InputSource(br);
SAXSource source = new SAXSource(saxReader, inputSource);
StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);
```

Ed ecco il risultato:

```
<?xml version="1.0" encoding="UTF-8"?>
<addressbook>
    <nickname>Fred</nickname>
    <email>fred@barneys.house</email>
    <html>TRUE</html>
    <firstname>Fred</firstname>
    <lastname>Flintstone</lastname>
    <work>999-Quarry</work>
    <home>999-BedrockLane</home>
    <fax>888-Squawk</fax>
    <pager>777-pager</pager>
    <cell>555-cell</cell>
</addressbook>
```

Concatenare più trasformazioni XSL

Spesso è utile concatenare più trasformazioni XSL in modo che l'output di una diventi l'input per quella successiva.

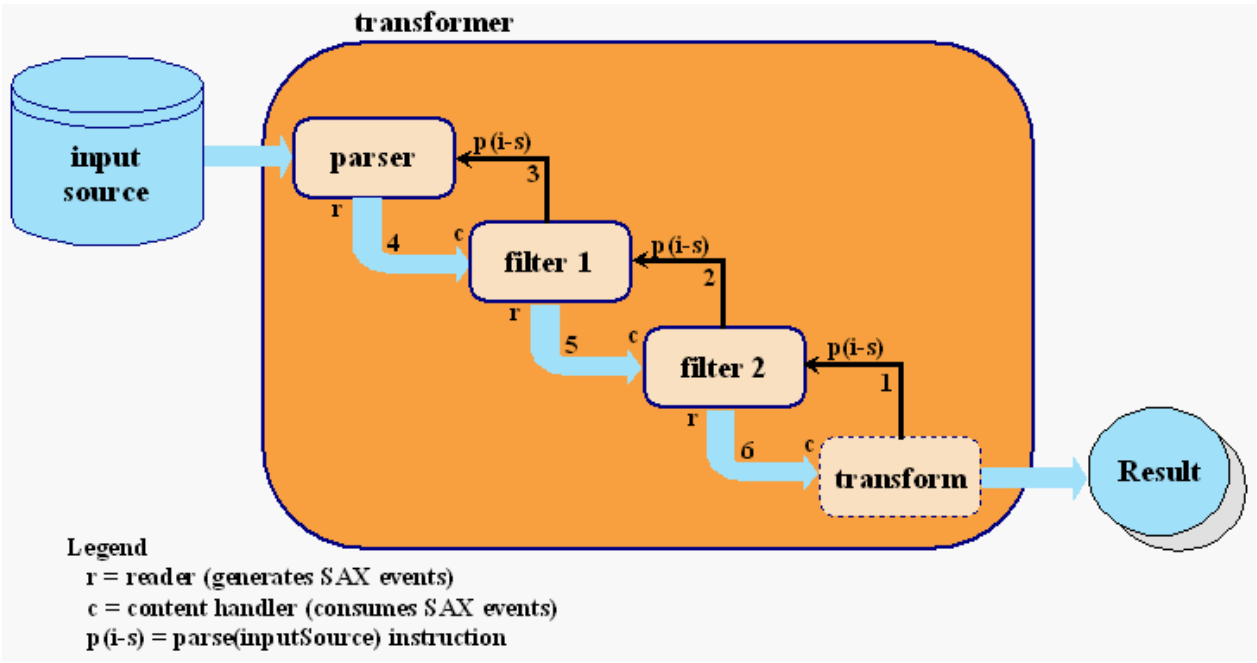
La classe chiave per effettuare quest'operazione è la classe **XMLFilter**. Un XMLFilter è sia un XMLReader che un SAXContentHandler. Come XMLReader genera eventi SAX verso chiunque gli sia registrato. Come SAXContentHandler consuma eventi SAX a cui si è sottoscritto. Questa dualità fa sì che gli oggetti XMLFilter possano connettersi in cascata.

Un XMLFilter può essere associato a un foglio XSL che effettua una trasformazione tra input e output.

Supponiamo di voler connettere in cascata sue fogli XSL (filter1 e filter2) :

- Dall'esempio precedente abbiamo visto che si crea un Transformer passandogli un XMLReader e un input stream.
- Creaimo un Transformer con un XMLFilter associato a filter2 e l'input stream che si vuole trasfromare.
- A questo punto il Transforment crea un SAXContentHandler associato a filter2 e richiede di eseguire il parsing
- XMLFilter associato a filter2 (attraverso il metodo setParent) si collega a un altro SAXContentHandler associato a filter1 e richiede di eseguire il parsing.
- A sua volta l' XMLFilter associato a filter1 chiede al SAX Parser di effettuare il parsing

- Inizia il parsing e gli eventi di call-back vengono girati dal SAXParser al XMLFilter1. Dal XMLFilter1 al XMLFilter2. Infine dal XMFilter2 al Transformer ContentHandler che li gira all'output sptream.




```
public static void main (String argv[])
{
 if (argv.length != 3) {
 System.err.println ("Usage: java FilterChain stylesheet1 stylesheet2 xmlfile");
 System.exit (1);
 }

 try {

 // Read the arguments
 File stylesheet1 = new File(argv[0]);
 File stylesheet2 = new File(argv[1]);
 File datafile    = new File(argv[2]);

 // Set up the input stream
 BufferedInputStream bis = new BufferedInputStream(
                           new FileInputStream(datafile));
 InputSource input = new InputSource(bis);

 // Set up to read the input file
 SAXParserFactory spf = SAXParserFactory.newInstance();
 SAXParser parser = spf.newSAXParser();
 XMLReader reader = parser.getXMLReader();
```

```
 // Create the filters
 SAXTransformerFactory stf = (SAXTransformerFactory)
                             TransformerFactory.newInstance();
 XMLFilter filter1 = stf.newXMLFilter(new StreamSource(stylesheet1));
 XMLFilter filter2 = stf.newXMLFilter(new StreamSource(stylesheet2));

 /* wire the output of the reader to filter1 and the output of filter1 to
filter2 */

 filter1.setParent(reader);
 filter2.setParent(filter1);

 // Set up the output stream
 StreamResult result = new StreamResult(System.out);

 // Set up the transformer to process the SAX events generated
 // by the last filter in the chain
 Transformer transformer = stf.newTransformer();
 SAXSource transformSource = new SAXSource(filter2, input);
 transformer.transform(transformSource, result);
  } catch (...) {
    ...
```

Per ulteriori informazioni

Consultare il tutorial ufficiale JAXP e la documentazione dell'API:

- **JAXP Home Page:** http://java.sun.com/xml/jaxp/
- **JAXP Tutorial:** http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/
- **JAXP API:** http://java.sun.com/xml/jaxp/dist/1.1/docs/api/overview-summary.html