XML e Java

Dario Bottazzi
Tel. 051 2093541,
E-Mail: dario.bottazzi@unibo.it,
SkypeID: dariobottazzi

Outline

- Parsing di documenti XML
- Simple API for XML processing (SAX)
- Document Object Model (DOM)
- Java API for XML Processing (JAXP)

Come Possiamo elaborare documenti XML?

- Un elemento importante che ha consentito la diffusione dei linguaggi e delle tecnologie XML è il supporto fornito dagli strumenti per il parsing
- Esistono vari strumenti disponibili
  - Apache Crimson
  - Apache Xerces
  - …
- Gli strumenti supportano i diversi linguaggi
  - C++
  - Java
  - C#
  - JavaScript
  - …

Parser

- Ogni applicazione che vuole fruire di XML deve includere un supporto per il parsing
  - editors, browsers
  - transformation/style engines, DB loaders, ...
- I parser XML sono diventati strumenti standard nello sviluppo delle applicazioni
  - Per esempio JDK1.4 integra un supporto al parsing (specifiche JAXP), e a default dornisce un parser (Apache Crimson)

(Leventhal, Lewis & Fuchs: Designing XML Internet Applications, Chapter 10, and D. Megginson: Events vs. Trees)

I Compiti del Parser

- Decomporre i documenti XML (istanza) nei loro elementi costitutivi
  - elementi, attributi, testo, processing instructions, entità, ...
- Eseguire controlli sul documento
  - Controllare che il documento sia ben formato
    - Correttezza sintattica del documento XML
  - Controllare (eventualmente) che il documento sia valido (DTD, XML Schema)
- Non tutti i parser consentono la validazione dei documenti
  - La validazione è una operazione computazionalmente costosa ed è compito dello sviluppatore scegliere se avvalersene

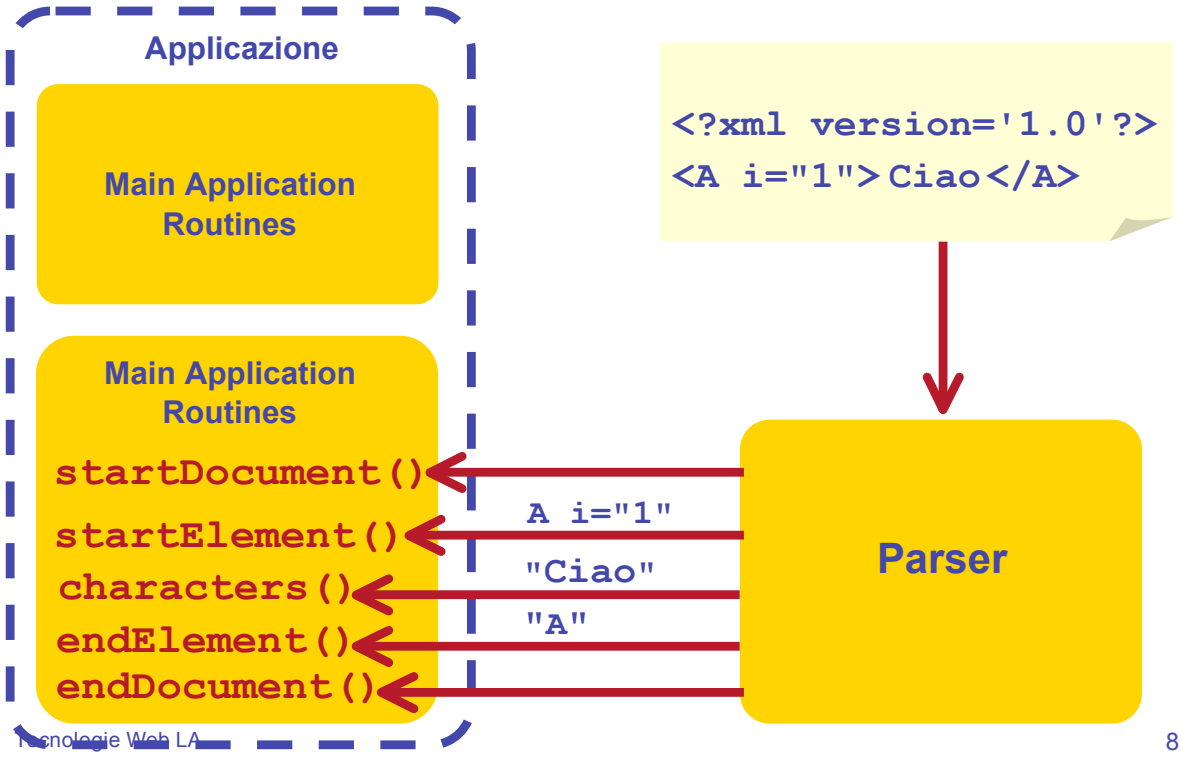
Modelli di Parsing per Documenti XML

- Interfacce basate su eventi
  - Interfaccie ESIS
    - Element Structure Information Set, interfaccia per il parsing di documenti SGML
  - Interfacce che sfruttano un modello a call-back: SAX
- Interfacce Object Model
  - W3C DOM Recommendation

Parsing Tramite Interfacce Basate su Eventi

- L'applicazione deve implementare un insieme di metodi di **callback** che verranno invocati dal parser mentre questi processa il documento
  - Le callback sono invocate al verificarsi di specifici eventi es. start-tag, end-tag,…
  - I parametri passati al metodo di callback forniscono ulteriori informazioni sull'evento
    - Nome dell'elemento
    - Nome e valore assunto dagli attributi
    - Valore delle stringhe contenute, …
- Modalità semplice ed efficiente per processare un documento XML
- Modello adottato da SAX (Simple API for XML)
  - SAX è uno standard de facto per il parsing di documenti XML
  - Potremmo pensare a SAX come "Serial Access XML"

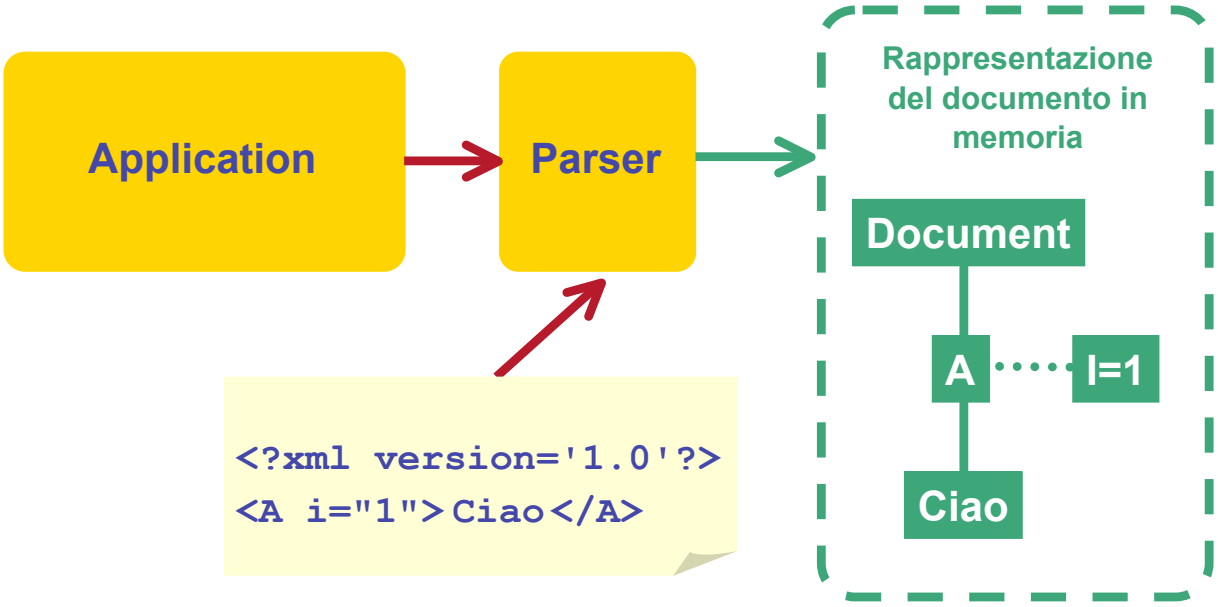
Struttura di una Applicazione con Modello a Callback per gli Eventi



Interfacce Object Model

- L'applicazione interagisce con una rappresentazione object-oriented di:
  - parser
  - documento
- Il documento è rappresentato da un **parse tree** costituito da vari oggetti quali *document*, *element, attribute, text*, …
- Il livello di astrazione che le interfacce Obiect Model forniscono al programmatore è maggiore rispetto a quello fornito dalle interfacce basate su modelli basati su eventi
  - Facile accedere ai figli di un elemento, ai fratelli, …
- Problema: Richiede la disponibilità di maggiori risorse computazionali (particolare memoria)

Struttura di una Applicazione con Modello DOM



Simple API for XML

SAX Event Callback API

- SAX stabilisce un insieme di API di utilizzo molto diffuso
- È uno standard de-facto
  - SAX 1.0 Maggio 1998, SAX 2.0 Maggio 2000
- SAX non è un parser, ma è una interfaccia
- L'interfaccia SAX è supportata da molti parser
  - Sun JAXP
  - IBM XML4J,
  - Oracle XML Parser for Java
  - Apache Xerces
  - Microsoft MSXML (in IE 5)
  - …

Interfacce SAX

- Le applicazioni interagiscono con parser conformi a SAX utlizzando **interfacce** stabilite
- Le interfacce di SAX possono essere divise in:
  - Interfacce **Parser-to-Application** (or **call-back**)
    - Consentono di definire funzioni di call-back e di associarle agli eventi generati dal parser mentre questi processa un documento XML
  - Interfacce **Application-to-parser**
    - Consentono di gestire il parser
  - Interfacce **Ausiliarie**
    - Facilitano la manipolazione delle informazioni fornite dal parser

Interfacce Parser to Application (call-back)

- Vengono implementate dall'applicazione per imporre un preciso comportamento a seguito del verificarsi di un evento. Qualora non implementate il comportamento di default è ignorare l'evento.
  - **`ContentHandler`**
    - Metodi per eleborare gli eventi generati dal parser
  - **`DTDHandler`**
    - Metodi per ricevere notifiche su entità esterne al documento e loro notazione dichiarata in DTD
  - **`ErrorHandler`**
    - Metodi per gestire gli errori ed i warning nel parsing di un documento
  - **`EntityResolver`**
    - Metodi per customizzare il processing di riferimenti ad entità esterne

Interfacce Application-to-Parser

- Sono implementate dal parser
  - **`XMLReader`**
    - Metodi per consentire all'applicazione di invocare il parser e di registrare gli oggetti che implementano le interfacce di call-back
  - **`XMLFilter`**
    - Interfaccia che consente di porre sequenza vari XMLReaders come una serie di filtri

Interfacce Ausiliarie

- **`Attributes`**
  - Metodi per accedere ad una lista di attributi
- **`Locator`**
  - Metodi per individuare l'origine degli eventi nel parsing dei documenti (es. systemID, numeri di lina e di colonna…)

ContentHandler

Metodi per ricevere informazioni sugli eventi generali che si manifestano nel parsing di un documento XML. (Riferirsi alle documentazione sulle API per la lista completa):

- `setDocumentLocator(Locator locator)`
  - Riceve un oggetto che determina l'origine di un evento SAX.
  - Per esempio potremmo usare il metodo per gestire errori semantici sul documento a livello applicativo o semplicemente per fornire informazioni agli utenti.

- `startDocument(); endDocument()`
  - Notifica dell'inizio o della conclusione del parsing del documento.

- `startElement(String namespaceURI, String localname, String qName, Attributes atts);`
  - Si noti che i questo metodo supporta i **namespaces**

- `endElement(String namespaceURI, String localname, String qName)`
  - Si noti che i questo metodo supporta i **namespaces**
  - Rispetto a startElement mancanoo gli attributi. La cosa non sorprende visto che gli attributi si possono trovare solo negli start-tag

SAX ed i Namespace: Esempio

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">
  <xsl:template match="/">
    <html>
      <xsl:value-of select="//total"/>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

- Una invocazione del metodo di SAX **`startElement`** per l'elemento **`xsl:template`** passerebbe i seguenti parametri attuali alla callback
  - **namespaceURI=** `http://www.w3.org/1999/XSL/Transform`
  - **localname** = `template`, **qName** = `xsl:template`

Namespaces: Esempio (2)

```
<xsl:stylesheet version="1.0" ...
   xmlns="http://www.w3.org/TR/xhtml1/strict">
   <xsl:template match="/">
     <html> ... </html>
   </xsl:template>
</xsl:stylesheet>
```

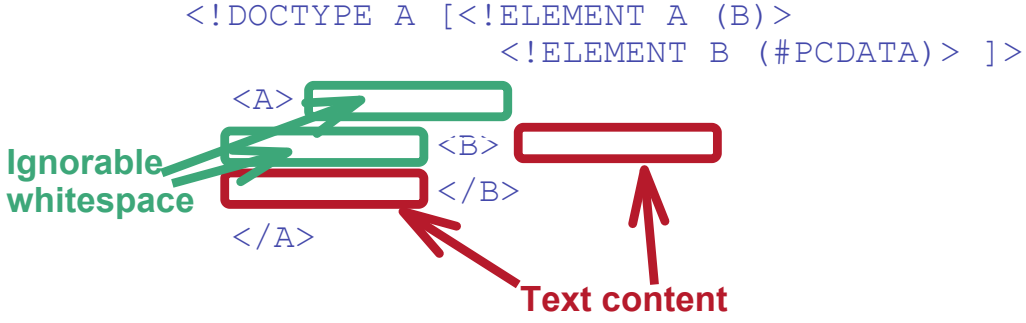
- Una invocazione SAX di **`startElement`** per l'elemento **`html`** passerebbe i parametri attuali alla calback

  namespaceURI= **`http://www.w3.org/TR/xhtml1/strict`**
  (come defalut namespace per gli elementi senza prefisso),

  localname = **`html`**, **qName =** **`html`**

Interfaccia ContentHandler

- **`characters(char ch[],`**
  `int start, int length)`
  - Notifica la presenza di *character data*.
- **`ignorableWhitespace(char ch[],`**
  `int start, int length)`
  - Notifica della presenza di whitespace ignorabili nell'element content.



Esempio SAX (1)

- Consideriamo il seguente file XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<db>
   <person idnum="1234">
   <last>Rossi</last><first>Mario</first></person>
   <person idnum="5678">
   <last>Bianchi</last><first>Elena</first></person>
   <person idnum="9012">
   <last>Verdi</last><first>Giuseppe</first></person>
   <person idnum="3456">
   <last>Rossi</last><first>Anna</first></person>
</db>
```


Esempio SAX (2)

- Stampare a video nome, cognome e identificativo di ogni persona

```
Mario Rossi (1234)
Elena Bianchi (5678)
Giuseppe Verdi (9012)
Anna Rossi (3456)
```

- Strategia di soluzione utilizzando un approccio event-based :
  - All'inizio di `person`, si registra `idnum` (e.g., 1234)
  - Si tiene traccia dell'inizio degli elementi `last` e `first`, per capire quando registrare nome e cognome (e.g., "Rossi" and "Mario")
  - Alla fine di `person`, si stampano i dati memorizzati

Esempio SAX (3)

- **Iniziamo ad importare le classi che ci servono**:

```
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;

//Default (no-op) implementation of
//interface ContentHandler:
import org.xml.sax.helpers.DefaultHandler;

// SUN JAXP used to obtain a SAX parser:
import javax.xml.parsers.*;
```


Esempio SAX (4)

- Definiamo una classe che `DefaultHandler` e ridefinisce i metodi di callback che sono rilevanti nella nostra applicazione:

```
public class SAXDBApp extends DefaultHandler{
// Flags to remember element context:
private boolean InFirst = false,
          InLast = false;
// Storage for element contents and
// attribute values:
   private String FirstName, LastName, IdNum;
```

Esempio SAX (5)

- Implementiamo i metodi di callback che ci servono
  - Start element registra l'inizio degli elementi `first` e `last`, ed il valore dell'attributo `idnum` dell'elemento `person`:

```
public void startElement (
   String namespaceURI, String localName,
   String rawName, Attributes atts) {
   if (localName.equals("first"))
       InFirst = true;
   if (localName.equals("last"))
       InLast = true;
   if (localName.equals("person"))
       IdNum = atts.getValue("idnum");
 } // startElement
```


Esempio SAX (6)

- Registriamo in modo opportuno il valore del testo a seconda che siamo dentro `first` o `last`:

```
public void characters (
     char ch[], int start, int length) {
 if (InFirst) FirstName =
          new String(ch, start, length);
 if (InLast) LastName =
     new String(ch, start, length);
} // characters
```

Esempio SAX (7)

Quanto abbiamo trovato la fine dell'elemento `person`, scriviamo i dati. Quando troviamo la fine dell'elemento `first` o dell'elemento `last` aggiorniamo i flag in modo opportuno

```
public void endElement(String namespaceURI, String
  localName, String qName) {
   if (localName.equals("person"))
     System.out.println(FirstName + " " +
  LastName + " (" + IdNum + ")" );
//Update the context flags:
  if (localName.equals("first"))
      InFirst = false;
  if (localName.equals("last"))
      InLast = false; }
```


Esempio SAX (8)

Il `main` dell'applicazione:

```
public static void main (String args[])
  throws Exception {
// Instantiate an XMLReader (from JAXP
// SAXParserFactory):
  SAXParserFactory spf =
     SAXParserFactory.newInstance();
  try {
    SAXParser saxParser = spf.newSAXParser();
    XMLReader xmlReader =
              saxParser.getXMLReader();
```

Esempio SAX (9)

```
// Instantiate and pass a new
// ContentHandler to xmlReader:
      ContentHandler handler = new SAXDBApp();
      xmlReader.setContentHandler(handler);
      for (int i = 0; i < args.length; i++) {
            xmlReader.parse(args[i]);
      }
   } catch (Exception e) {
       System.err.println(e.getMessage());
       System.exit(1);
   };
} // main
```


Document Object Model

Cosa è DOM?

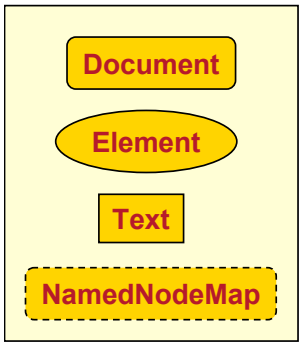
- DOM (Document Object Model) è una API per accedere e manipolare documenti XML ed HTML
  - Consente alle applicazioni (anche agli script) di costruire documenti, di navigare la loro struttura, di aggiungere, modificare o cancellare elementi e contenuto
- DOM è object-based
- DOM è uno standard W3C supportato da numerosi linguaggi che utilizzano le stesse API
- Mentre SAX fornisce un meccanismo sequenziale per il parsing dei documenti (si poteva dire SAX "Serial Access XML"), possiamo pensare DOM come acronimo di "Directly Obtainable in Memory"

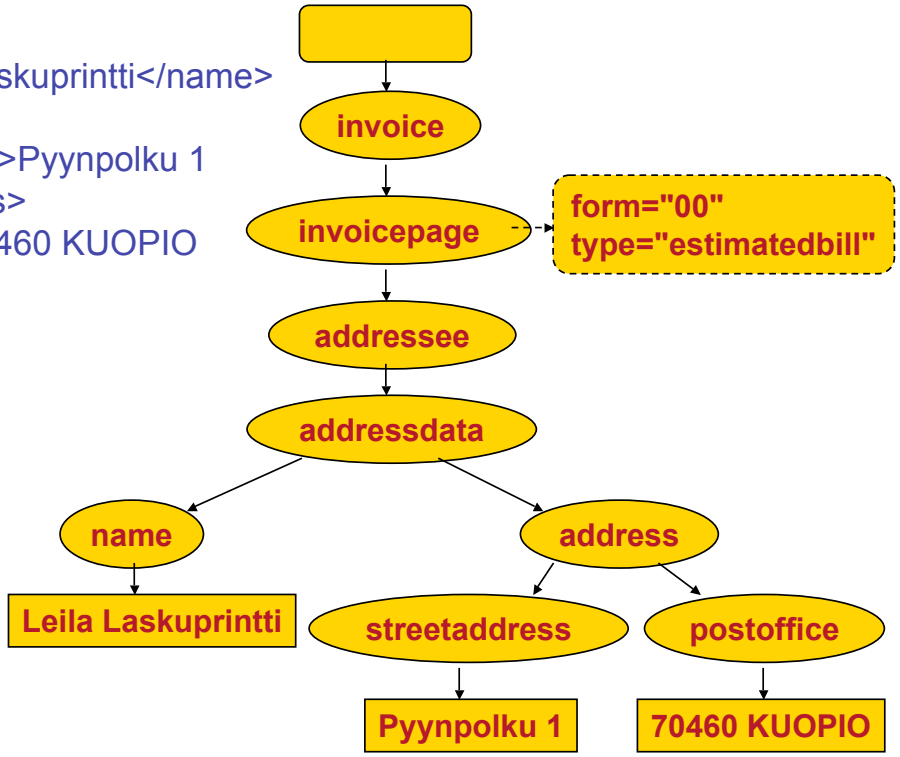
Modello della Struttura di DOM

- DOM è basata sui concetti tradizionali della programmazione ad oggetti:
  - *Metodi*: regolano l'accesso e consentono di alterare lo stato degli oggetti
  - *Interfacce*: consentono di dichiarare un insieme di metodi
  - *Oggetti:* incapsulano dati e metodi
- Il modello rappresenta i documenti XML come alberi (parse tree)
  - La struttura ad albero è determinata dalle intefacce stabilite dalla specifica di DOM. Tipicamente la struttura ad albero viene mantenuta nell'implementazione dei parser DOM.

```
<invoice>
 <invoicepage form="00"
                  type="estimatedbill">
  <addressee>
   <addressdata>
    <name>Leila Laskuprintti</name>
    <address>
     <streetaddress>Pyynpolku 1
     </streetaddress>
     <postoffice>70460 KUOPIO
     </postoffice>
    </address>
   </addressdata>
  </addressee> ...
```

DOM structure model





Struttura di DOM Level 1

- DOM **Core Interface**
  - **Fornisce le interfacce di base per accedere ai documenti strutturati**
  - Fornisce inoltre Extended interface
    - Sono specifiche di XML: CDATASection, DocumentType, Notation, Entity, EntityReference, ProcessingInstruction
- DOM **HTML Interfaces**
  - Forniscono supporto per accedere ai documenti HTML
  - (vedremo qualcosa più avanti nel corso)

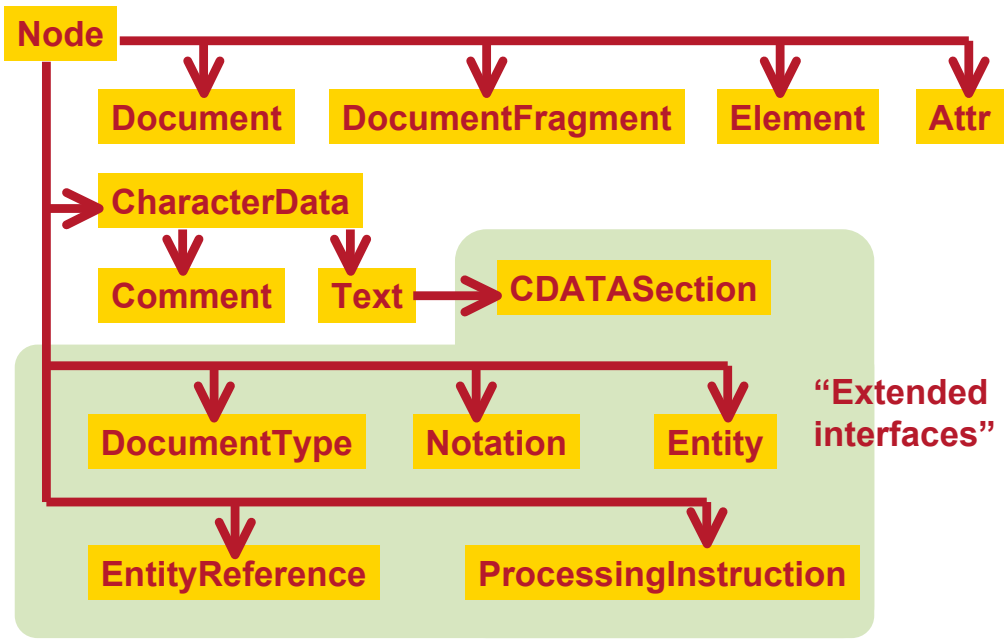
DOM Level 2

- DOM Level 1: fornisce il modello ed meccanismi di base per rappresentare e manipolare struttura e contenuto di un documento
  - Non fornisce però accesso ai contenuti del DTD
- DOM Level 2 aggiunge
  - supporto per i namespaces
  - Accesso agli elementi tramite il valore degli attributi ID
  - Funzionalità opzionali
    - Interfacce per document views e style sheets
    - Modello ad eventi (per gestire ad esempio le azioni degli utenti sugli elmenti)
    - Metodi per attraversare il documento e per manipolarne intere regioni (es. regioni selezionate dall'utente tramite un editor)
  - Caricamento e scrittura dei documenti, benchè spesso supportata non è specificata in DOM Level 2 ma sarà inserita in DOM Level 3

DOM Language Bindings

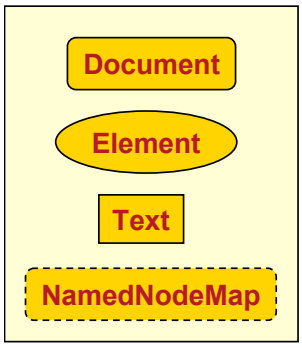
- DOM non dipende da un linguaggio specifico (Language-independence):
  - Le interfacce DOM sono definite sulla base dell'Interface Definition Language (OMG IDL; definito nelle specifiche di Corba)
- Language bindings (implementazioni delle interfacce di DOM) per tutti i principali linguaggi
  - Java
  - C++
  - JavaScript
  - C#
  - Python
  - ...
- Definite nelle specifiche di DOM alcuni Language Bindings:
  - Java
  - JavaScript

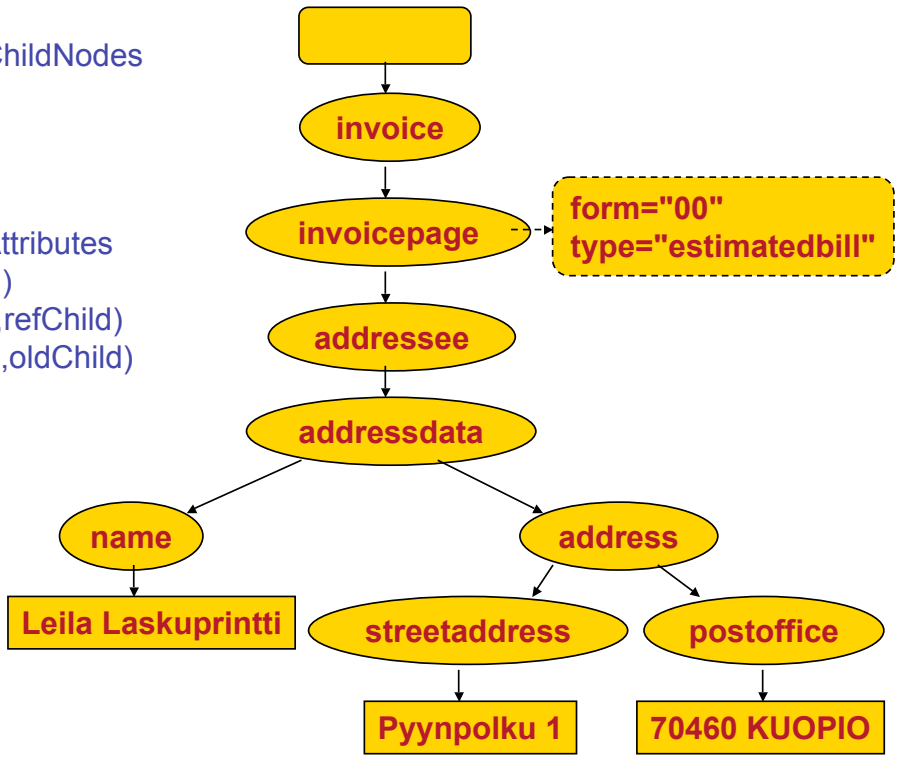
Core Interface: la Tassonomia di Node



Node
getNodeType
getNodeValue
getOwnerDocument
getParentNode
hasChildNodes getChildNodes
getFirstChild
getLastChild
getPreviousSibling
getNextSibling
hasAttributes getAttributes
appendChild(newChild)
insertBefore(newChild,refChild)
replaceChild(newChild,oldChild)
removeChild(oldChild)

DOM interfaces: Node





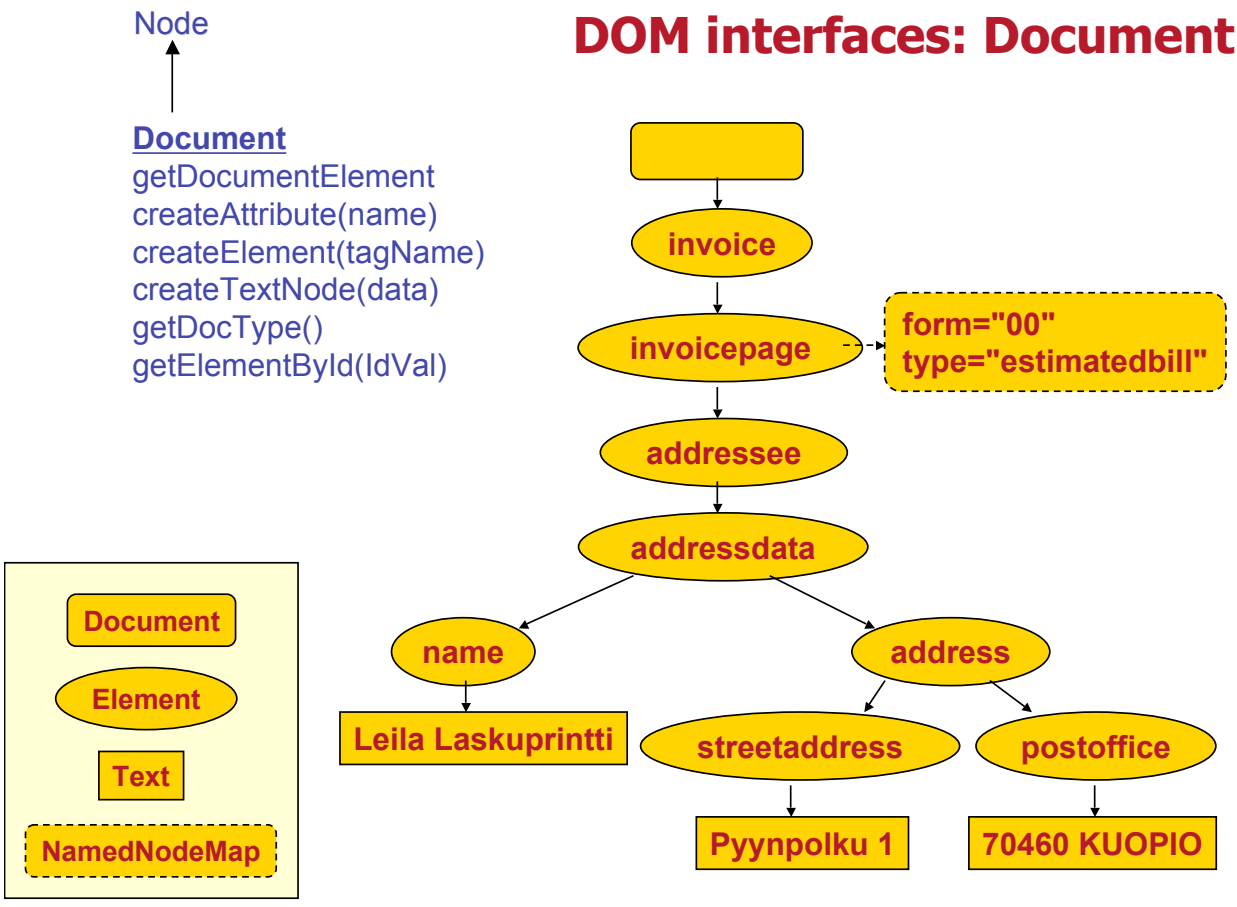
Object Creation in DOM

- DOM specifica diverse classi
- L'oggetto *X* di DOM è inserito nel contesto di un `Document`: *X*`.getOwnerDocument()`
- Gli oggetti che implementano l'interfaccia *X* sono creati da una factory
  D`.create`*X*`(…)` ,
  dove D è l'oggetto `Document`. Esempi:

```
createElement("A"),
createAttribute("href"),
createTextNode("Hello!")
```

- Creazione e salvataggio di `Document` sono demandati alle specifiche implementazioni

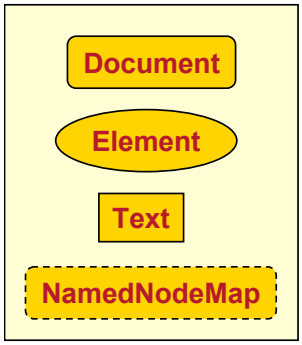
DOM interfaces: Document

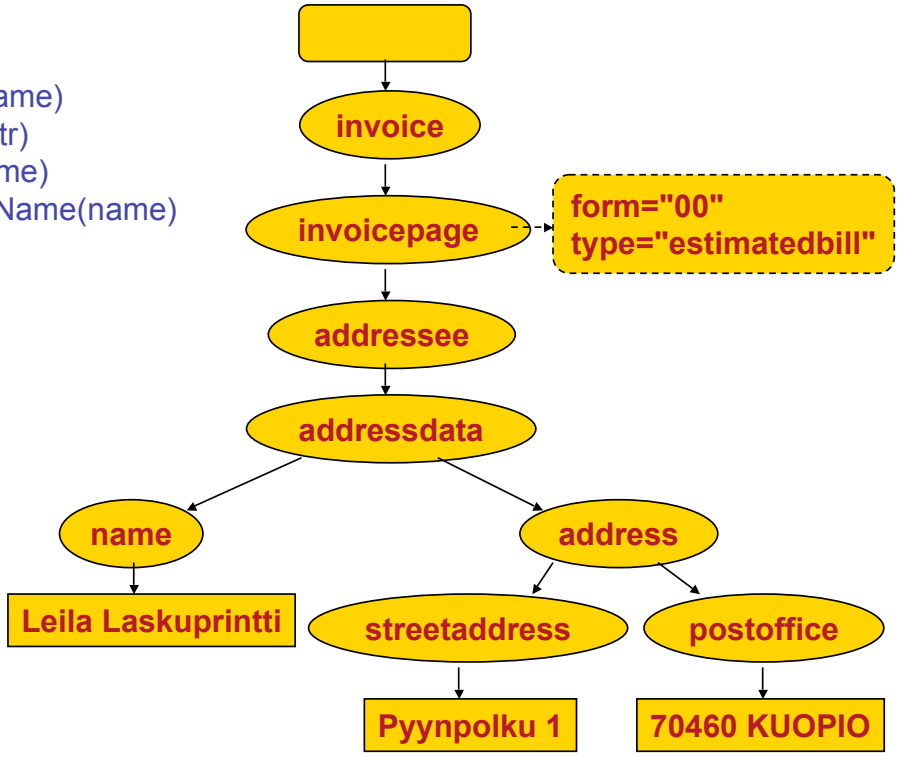


DOM interfaces: Element

Node

Element
getTagName
getAttributeNode(name)
setAttributeNode(attr)
removeAttribute(name)
getElementsByTagName(name)
hasAttribute(name)





Accedere alle proprioetà di Node

- **`Node.getNodeName()`**
  - Se applicato ad `Element` equivale a `getTagName()`
  - Se applicato ad `Attr` restituisce il nome dell'attributo
  - …
- **`Node.getNodeValue()`**
  - Restituisce il contenuto di un text node, il valore di un attributo…;
  - Se applicato ad `Element` restituisce **null**
- **`Node.getNodeType()`**:
  - Restituisce un valore numerico constante (es. 1, 2, 3, …, 12) che corrisponde a `ELEMENT_NODE, ATTRIBUTE_NODE, TEXT_NODE, …, NOTATION_NODE`

Manipolazione di Content ed element

- Manipolare **`CharacterData`** D:
  - D.**`substringData`**(offset, count)
  - D.**`appendData`**(string)
  - D.**`insertData`**(offset, string)
  - D.**`deleteData`**(offset, count)
  - D.**`replaceData`**(offset, count, string)
    `(= delete + insert)`

- Accedere agli attributi dell'**`Element`** E:
  - E.**`getAttribute`**(name)
  - E.**`setAttribute`**(name, value)
  - E.**`removeAttribute`**(name)

Ulteriori Core Interface (1)

- **`NodeList`** per liste ordinate di nodi
  - Per esempio può essere ottenuto da `Node.getChildNodes()` oppure da `Element.getElementsByTagName("name")`
    - Abbiamo chiesto tutti i discendenti di tipo `"name"` nell'ordine in cui appaiono nel documento.
    - Potevamo usare la wild-card `"*"` per ottenere tutti gli elementi

- Accedere ad un nodo specifico e iterare tutti i nodi di una **`NodeList`**:
  - E.g. Java code to process all children:

```
for (i=0;i<node.getChildNodes().getLength(); i++)
   process(node.getChildNodes().item(i));
```

Ulteriori Core Interface (2)

- **`NamedNodeMap`** per insiemi non ordinati di nodi acceduti tramite il nome
  - esempio `Node.getAttributes()`

- `NodeList` e `NamedNodeMap` riflettono eventuali cambiamenti nella struttura dei documenti (sono “live”)

Implementazioni di DOM

- Parser Java
  - **IBM XML4J**
  - **Apache Xerces**
  - **Apache Crimson**
  - **SUN JAXP**
  - **…**
- MS IE5 browser: interfacce per la programmazione COM per C/C++ ed MS Visual Basic, oggeti ActiveX per i linguaggi di script
- XML::DOM (implementazione Perl di DOM Level 1)

Un Esempio di Java-DOM

- Vogliamo acrivere una applicazione `BuildXml`
  - Crea un nuovo documento `db` con due elementi `person` se non esiste, o li aggiunge se esiste
- Base tecnica
  - Supporto al DOM fornito da Sun JAXP (in particolare fornito da Apache Crimson)
  - Apache Crimson fornisce la possibilità di inizializzare e di memorizzare il documento
  - Apache Crimson è il parser di default di Java a partire dalla JDK 1.4

Un Esempio di Java-DOM (1)

Importiamo i package che ci servono:

```
import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
// Native (parse and write) methods of the
// JAXP 1.1 default parser (Apache Crimson):
import org.apache.crimson.tree.XmlDocument;
```

Un Esempio di Java-DOM (2)

Classi per modificare il documento nel file `fileName` specificato:

```
public class BuildXml {
  private Document document;

public BuildXml(String fileName) {
  File docFile = new File(fileName);
  Element root = null; // doc root element
  // Obtain a SAX-based parser:
  DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
```


Un Esempio di Java-DOM (3)

```
try { // to get a new DocumentBuilder:
   documentBuilder builder =
       factory.newInstance();

  if (!docFile.exists()) { //create new doc
     document = builder.newDocument();
     // add a comment:
     Comment comment =
            document.createComment(
                  "A simple personnel list");
     document.appendChild(comment);
     // Create the root element:
     root = document.createElement("db");
     document.appendChild(root);
```

Un Esempio di Java-DOM (4)

… oppure se il file `docFile` esiste:

```
} else   { // access an existing doc
  try { // to parse docFile
    document = builder.parse(docFile);
    root = document.getDocumentElement();
  } catch (SAXException se) {
    System.err.println("Error: " +
          se.getMessage() );
    System.exit(1);
  }
  /* A similar catch for a possible IOException */
```


Un Esempio di Java-DOM (5)

Crea due elementi e collegali a `root`:

```
Node personNode =
    createPersonNode(document, "1234",
                "Pekka", "Kilpeläinen");
root.appendChild(personNode);
personNode =
    createPersonNode(document, "5678",
                     "Irma", "Könönen");
root.appendChild(personNode);
```

Un Esempio di Java-DOM (6)

- Salva il documento risultante:

```
try { // to write the
      // XML document to file fileName
  ((XmlDocument) document).write(
     new FileOutputStream(fileName));
} catch ( IOException ioe ) {
      ioe.printStackTrace();
}
```


Un Esempio di Java-DOM

```
public Node createPersonNode(Document document,
      String idNum, String fName, String lName) {
  Element person = document.createElement("person");
  person.setAttribute("idnum", idNum);
  Element firstName = document. createElement("first");
  person.appendChild(firstName);
  firstName.appendChild(document.createTextNode(fName));
  ….
/* in modo analogo per lastName */
  return person;
}
```

Un Esempio di Java-DOM

```
public static void main(String args[]){ if
   (args.length > 0) {
       String fileName = args[0];
       BuildXml buildXml = new
                   BuildXml(fileName);
   } else {
       System.err.println(
             "Give filename as argument");
   };
} // main
```


Sommario su XML e Java

- I parser XML rendono disponibile alle applicazioni la struttura ed il contenuto dei documenti XML
- Specificano un insieme ricco di APIs
- Event-based API
  - Notificano alle applicazioni eventi generati nel parsing dei documenti
  - Es. API SAX basati su call-back
- Object-model based APIs
  - Forniscono accesso al parse tree che rappresenta il documento XML
  - es, DOM, W3C Recommendation
  - Molto comode ed eleganti richiedono però maggiori risorse computazionali es. memoria
- I parser maggiormente diffusi supportano si SAX sia DOM. In realtà spesso i parser DOM sono sviluppati su parser SAX…

JAXP

JAXP 1.1

- JAXP è una interfaccia che ci consente di istanziare ed utilizzare parser XML nelle applicazioni Java
  - Fornisce vari package
    - `org.xml.sax`: interfacce SAX 2.0
    - `org.w3c.dom`: interfacce DOM Level 2
    - `javax.xml.parsers`: inizializzazione ed uso dei parser
    - `javax.xml.transform`: inizializzazione ed uso dei transformers XML (forse vedremo qualcosa a fine corso)
- Inclusa nella JDK dalla vers. 1.4

JAXP: processor plugin (1)

- Metodo non proprietario per selezionare l'implementazione del parser XML che preferiamo a tempo di esecuzione
  - Tipicamente sfrutta le system properties
    ```
    javax.xml.parsers.SAXParserFactory
    javax.xml.parsers.DocumentBuilderFactory
    javax.xml.transform.TransformerFactory
    ```
  - esempio:
    ```
    System.setProperty(
    "javax.xml.parsers.DocumentBuilderFactory",
    "com.icl.saxon.om.DocumentBuilderFactoryImpl");
    ```


JAXP: processor plugin (2)

- A default, l'implementazione di riferimento sfrutta
  - Apache Crimson/Xerces per il parsing dei documenti XML
  - Apache Xalan per la traformazione di documenti
- Attualmente supportata da pochi (ma diffusi) strumenti
  - Parsers: Apache Crimson and Xerces, Aelfred
  - XSLT transformers: Apache Xalan, Saxon

JAXP: Funzionalità

- Fornisce il parsing sfruttando SAX 2.0 e DOM Level 2
- Supporta le traformazioni
  - TrAX transformation API
- Specifica elementi non trattati nelle specifiche di SAX 2.0 e DOM Level 2
  - Gestione della validazione e della gestione degli errori
  - Creazione e salvataggio di oggetti documento DOM

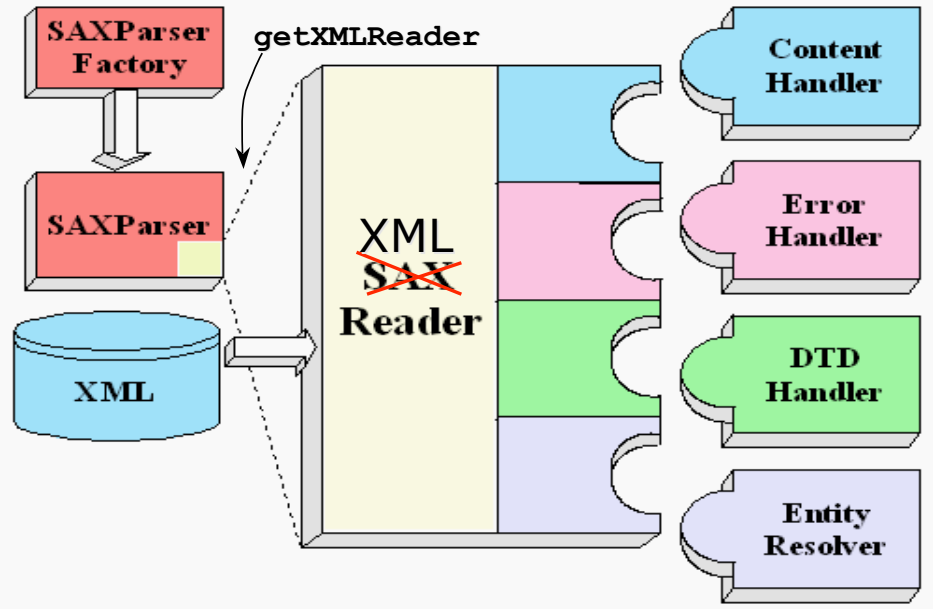
JAXP Parsing API

- Nei package di JAXP è incluso
  `javax.xml.parsers`
- Usato per invocare ed usare le implementazioni dei parser SAX e DOM:

```
SAXParserFactory spf =
    SAXParserFactory.newInstance();

DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();
```

JAXP: Usare un Parser SAX (1)

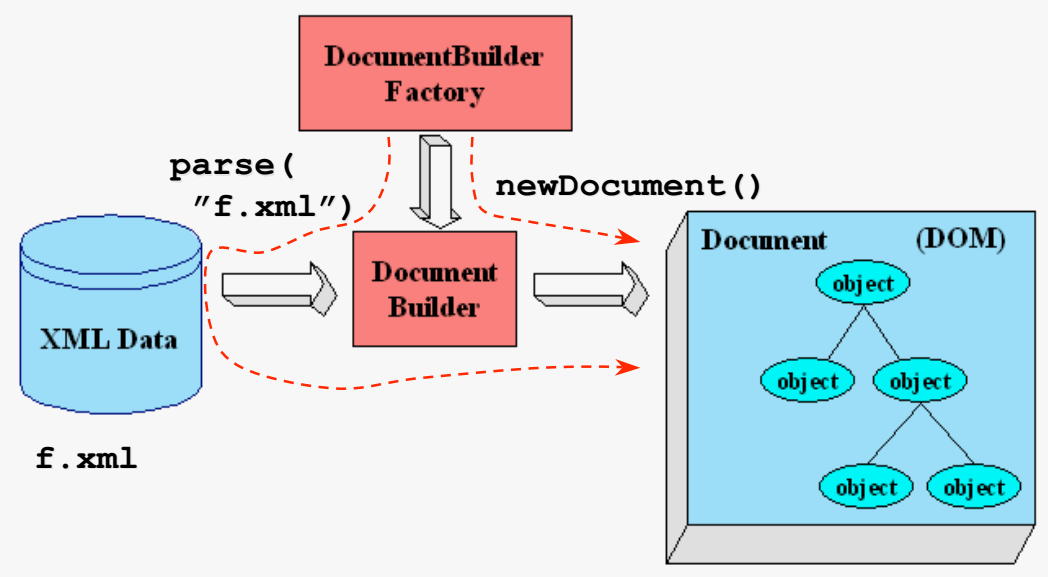


JAXP: Usare un Parser SAX (2)

- Abbiamo già usato questo codice:

```
SAXParserFactory spf =
       SAXParserFactory.newInstance();
  try {
        SAXParser saxParser = spf.newSAXParser();
        XMLReader xmlReader =
                  saxParser.getXMLReader();
      } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
      };
```

JAXP: Usare un Parser DOM (1)



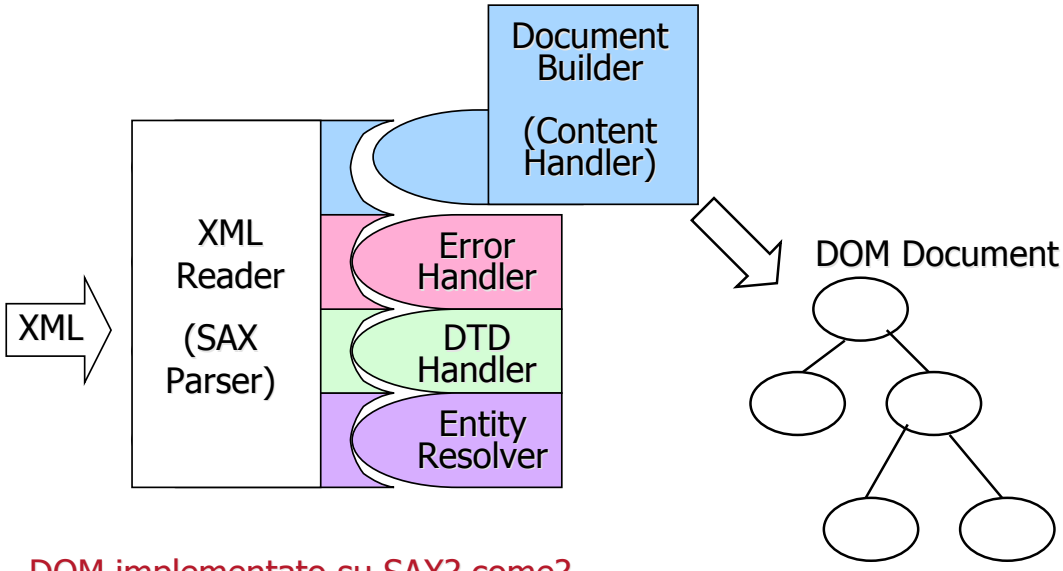
JAXP: Usare un Parser DOM (2)

- Abbiamo usato anche questo:

```
DocumentBuilderFactory dbf =
            DocumentBuilderFactory.newInstance();

try { // to get a new DocumentBuilder:
       documentBuilder builder =
            dbf.newDocumentBuilder();
      } catch (ParserConfigurationException e) {
            e.printStackTrace());
            System.exit(1);
      };
```

Costruzione di DOM in JAXP



DOM implementato su SAX? come?

JAXP: Gestione del Parsing

- Consente la gestione degli errori nel parsing di documenti DOM
  - Si creano degli **ErrorHandler** di SAX, che impementano i metodi **error**, **fatalError** e **warning**. Questi metodi vengono passati con **setErrorHandler** al **DocumentBuilder**
- La Validazione ed i namespace vengono gestiti sia per **SAXParserFactories** e per **DocumentBuilderFactories** con

```
setValidating(boolean)
setNamespaceAware(boolean)
```

Altre API Java per il Parsing di XML

- JDOM
  - variante di DOM; maggiormente allineato alla programmazione orientata algl oggetti
  - http://www.jdom.org/
- DOM4J
  - Simile a JDOM ma con maggiori funzionalità
  - http://www.dom4j.org/
- JAXB (Java Architecture for XML Binding)
  - Compila i DTD in classi Java DTD-specific che consentono di leggere, scrivere e manipolare documenti XML validi
  - http://java.sun.com/xml/jaxb/

Semplice Esempio DOM

Stampa gli elementi selezionati dal file XML specificato

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.*;

public class DOMCountElement{
   public static void main(String[] args) {
                        try {
                                BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
                                System.out.print("Enter File name: ");
                                String xmlFile = bf.readLine();
                                File file = new File(xmlFile);
                                if (file.exists()){
                                        DocumentBuilderFactory factory =
                                                DocumentBuilderFactory.newInstance();
                                        // Create the builder and parse the file
                                        Document doc = factory.newDocumentBuilder().parse(xmlFile);
                                        System.out.print("Enter element name: ");
                                        String element = bf.readLine();
                                        NodeList nodes = doc.getElementsByTagName(element);
                                        System.out.println("xml Document Contains " + nodes.getLength()
                                                                + " elements.");
                                }
                                else{
                                        System.out.print("File not found!");
                                }
                        }
                        catch (Exception ex) {
        System.out.println(ex);
    }
  }
}
```

Un documento è Ben Formato?

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;

public class DOMParserCheck {
static public void main(String[] arg){
        try{
                BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Enter File name: ");
                String xmlFile = bf.readLine();
                File file = new File(xmlFile);
                if(file.exists()){
                        try {
                                // Create a new factory to create parsers
                                DocumentBuilderFactory dBF = DocumentBuilderFactory.newInstance();
                                // Use the factory to create a parser (builder) and use it to parse the document.
                                DocumentBuilder builder = dBF.newDocumentBuilder();
                                // builder.setErrorHandler(new MyErrorHandler());
                                InputSource is = new InputSource(xmlFile);
                                Document doc = builder.parse(is);
                                System.out.println(xmlFile + " is well-formed!");
                        }
                       catch (Exception e) { System.out.println(xmlFile + " isn't well-formed!"); System.exit(1) }
                }
                else{ System.out.print("File not found!");}
        } catch(IOException io){io.printStackTrace();}
    }
}
```


Dove Sono gli Errori? (1)

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;

public class DOMLocateError{
        static public void main(String[] arg){
                try {
                        BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
                        System.out.print("Enter File name: ");
                        String xmlFile = bf.readLine();
                        File file = new File(xmlFile);
                        if(file.exists()){
                                // Create a new factory
                                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                                // Use the factory to create builder document.
                                DocumentBuilder builder = factory.newDocumentBuilder();
                                Document doc = builder.parse(xmlFile);
                                System.out.println(xmlFile + " is well-formed!");
                        }
                        else{
                                System.out.print("File not found!");
                        }
                }
```

Dove Sono gli Errori? (2)

```
        catch (SAXParseException e) {
                System.out.println("type" + ": " + e.getMessage()+"\n");
                System.out.println("Line " + e.getLineNumber() + " Column "+ e.getColumnNumber());
                }
        catch (SAXException e) {
                System.err.println(e);
                System.exit(1);
                }
        Catch (ParserConfigurationException e) {
                System.err.println(e);
                System.exit(1);
                }
        catch (IOException e) {
                System.err.println(e);
                System.exit(1);
                }
        }
}
```


Accesso tramite API DOM ad un semplice file XML (1)

```
<?xml version="1.0"?>
<student>
        <student-name>
                <firstname>Anusmita</firstname>
                <lastname>Singh</lastname>
        </student-name>

         <student-address>
                <address>Rohini</address>
                 <city>Delhi</city>
        </student-address>
</student>
```

Accesso tramite API DOM ad un semplice file XML (2)

```
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.*;

public class AccessingXmlFile {

 public static void main(String argv[]) {
  try {
  File file = new File(argv[0]);
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  DocumentBuilder db = dbf.newDocumentBuilder();
  Document document = db.parse(file);
  document.getDocumentElement().normalize();
  System.out.println("Root element " +
                          document.getDocumentElement().getNodeName());
  NodeList node = document.getElementsByTagName("student");
  System.out.println("Information of the students");
```


Accesso tramite API DOM ad un semplice file XML (3)

```
  for (int i = 0; i < node.getLength(); i++) {
    Node firstNode = node.item(i);

   if (firstNode.getNodeType() == Node.ELEMENT_NODE) {
       Element element = (Element) firstNode;
       NodeList firstNameElemntList = element.getElementsByTagName("firstname");
       Element firstNameElement = (Element) firstNameElemntList.item(0);
       NodeList firstName = firstNameElement.getChildNodes();
       System.out.println("First Name : "  + ((Node) firstName.item(0)).getNodeValue());
       NodeList lastNameElementList = element.getElementsByTagName("lastname");
       Element lastNameElement = (Element) lastNameElementList.item(0);
       NodeList lastName = lastNameElement.getChildNodes();
       System.out.println("Last Name : " + ((Node) lastName.item(0)).getNodeValue());
      NodeList addressList = element.getElementsByTagName("address");
       Element addressElement = (Element) addressList.item(0);
       NodeList address = addressElement.getChildNodes();
       System.out.println("Address : "  + ((Node) address.item(0)).getNodeValue());

      NodeList cityList = element.getElementsByTagName("city");
       Element cityElement = (Element) cityList.item(0);
       NodeList city = cityElement.getChildNodes();
       System.out.println("City : "  + ((Node) city.item(0)).getNodeValue());
  }} } catch (Exception e) {
     e.printStackTrace();}}}
```