INTRODUZIONE A XML

Ing. Marco Bertini - Università degli Studi di Firenze
Via S. Marta 3 - 50139 - Firenze - Italy
Tel. +39-055-4796262
Fax +39-055-4796363
E-mail bertini@dsi.unifi.it
Web: http://viplab.dsi.unifi.it/~bertini

Introduzione

- XML è l' eXtensible Markup Language
  - è *extensible* a differenza dell'HTML che è fisso.
- E' un metalinguaggio:
  - può definire nuovi linguaggi di markup

Introduzione

Def.: markup:

- sequenza di caratteri o altri simboli che si inseriscono all'interno di un documento per indicare come il contenuto deve apparire o per descrivere la struttura logica del documento. Spesso gli indicatori di markup sono chiamati *tag.*

Introduzione

- E’ una specifica ufficiale del World Wide Web Consortium (W3C).
- W3C lo definisce come:
  - "a common syntax for expressing *structure* in data."

Introduzione

- Lo scopo di XML è quello di separare la definizione dei dati dalla loro rappresentazione, per consentire lo scambio di documenti strutturati sul web.

- HTML specifica come un documento deve essere mostrato, non descrive che tipo di informazione è contenuta e come è strutturata.

Introduzione

- XML consente agli autori di un documento di organizzare l'informazione in un modo standard.
- Uno degli scopi principali di XML è di consentire lo scambio di dati tra sistemi potenzialmente incompatibili.

Introduzione

- XML è basato su file di testo, può essere usato su qualsiasi piattaforma.
- E’ nato per essere usato su Internet, ma va bene ovunque.
- E’ un sub-set di SGML (Standard Generalized Markup Language)

Introduzione

Def.: SGML:

- Metalinguaggio per la definizione di linguaggi di markup. Standard ISO

- SGML si basa sull'idea che i documenti hanno elementi strutturali e semantici che possono essere descritti senza tenere conto di come devono essere visualizzati.

Introduzione

- L’ Hypertext Markup Language (HTML), è un esempio di linguaggio basato su SGML.

- C’è un document type definition (DTD) per l’HTML.
- Un linguaggio definito in termini di SGML si chiama *applicazione* di SGML.

Introduzione

- SGML è usato per lo scambio di dati e documenti dal Dipartimento della Difesa degli Stati Uniti, dalle industrie aerospaziali e delle telecomunicazioni, etc.
- XML è nato per portare SGML sul web

HTML

- E’ un linguaggio per la descrizione di come appaiono documenti all’interno di un browser.
- Fornisce tag per titoli, paragrafi, font, link, immagini.

HTML

<P>
<FONT SIZE="+1"><STRONG>HTML: All form and no substance</STRONG></FONT>
HTML is a language designed to "talk about" documents: headings, titles, captions, fonts, and so on. It's heavily document structure- and presentation-oriented.

HTML: All form and no substance

HTML is a language designed to "talk about" documents: he

HTML

- Non è estendibile: a meno di non essere Microsoft o IBM…
  Lo standard è definito dal W3C.

- E’ orientato alla rappresentazione del contenuto:
  - mischia elementi strutturali (es. titoli) con elementi rappresentativi (es. *bold*)

HTML

- Non separa il contenuto dalla presentazione
- Fornisce una sola “vista”: è estremamente difficile scrivere pagine dinamiche che si riadattino al sistema usato dall’utente

- Non ha praticamente struttura semantica: non c’è indicazione sul significato dei dati di una pagina HTML

HTML

- Originariamente l'HTML prevedeva di contrassegnare l'informazione secondo il suo significato, indipendentemente da come il browser l'avrebbe resa:
  - <TITLE>Questo è un titolo di pagina</TITLE>
  - <STRONG>Summary</STRONG>
  - <H2>Header<H2>

HTML

- La tendenza è invece quella di specificare precisamente l'apparenza dei dati:

- <font size="-1" face="Arial, Helvetica, Sans-serif">

HTML

- Un metodo per separare la resa grafica dai contenuti è usare i CSS (Cascading Style Sheets), ma non sono ancora standardizzati nei browser di ultima generazione.

HTML

M1jgenfj.css

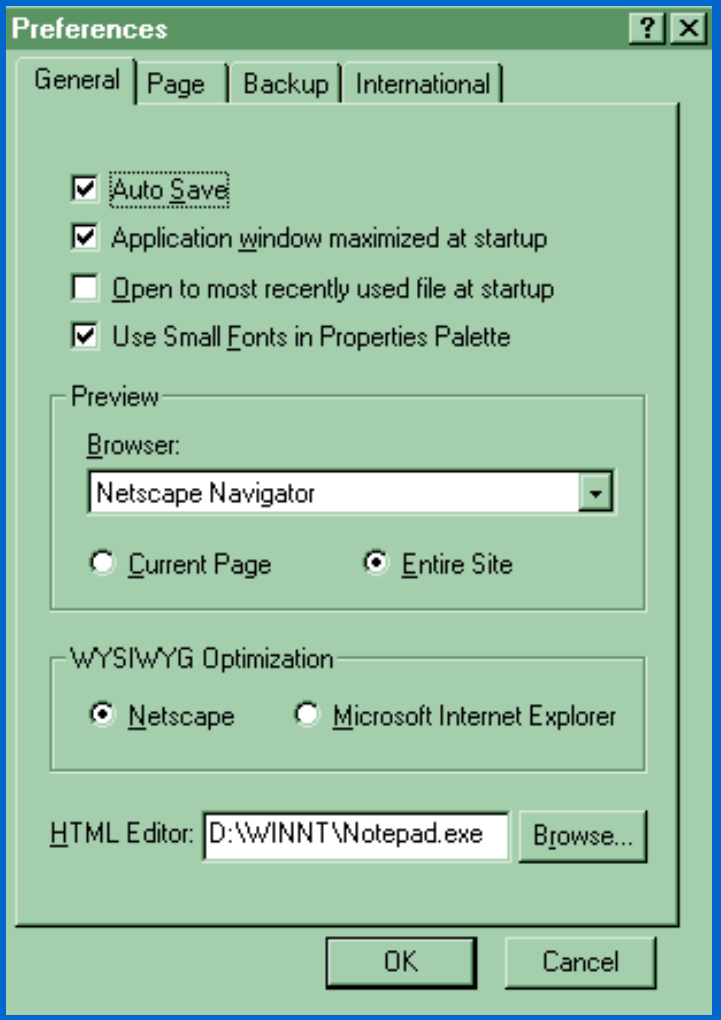
```
/* --- COMMON TO ALL ARTICLES --- */

BODY { margin-left: 15%; margin-right: 15%; margin-top: 5%;
       text-align: justify; background: white none; color: black }

DFN  { font-style: italic }

CODE { color: #0000AA; background: white none; }

H1,H2,H3,H4,H5,H6 { font-family: Arial, Helvetica, Sans-Serif;
                    text-align: left }

H1   { text-align: center } /* Overrides above rule as it appears k

.subtitle { text-align: center; font-size: 60% }

H2   { padding-top: 2em }

DT   { font-weight: bold }

.uferdig   { color: #AA0000     } /* Used for parts not yet finishe

P.center   { text-align: center }
P.author   { text-align: center }
P.contents { text-align: center; padding-bottom: 2em }
P.note     { background: white none; color: #AA0000; font-weight: k

DL    { margin-left: 5% }
```

HTML

- Gli editor HTML spesso usano tag per in base alla resa grafica che forniscono anziché al loro significato semantico:
  - es. <UL> per l'indentazione anziché per le liste non numerate

HTML

- Le pagine web sono spesso disegnate tenendo conto delle differenti rese grafiche delle varie versioni di browser



HTML

- La struttura interna di un documento HTML è molto ridotta, per cui è possibile scrivere documenti corretti ma senza senso semantico:
  - es.: i titoli di un “paragrafo” possono essere sopra ai titoli di un “capitolo”

HTML

```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
```

HTML

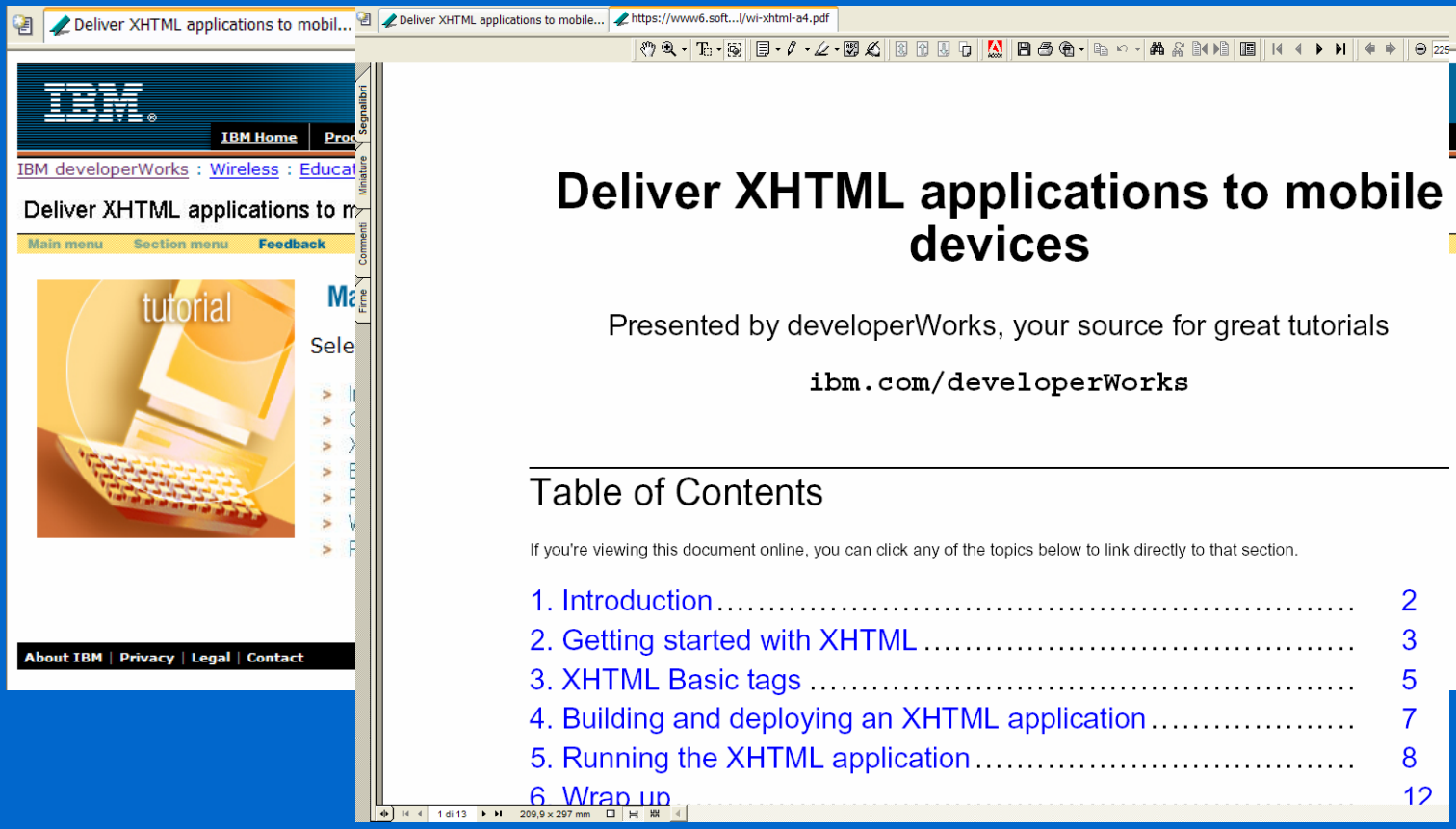
- Es.: dentro il body non devo rispettare l'ordine degli header H1, H2, H3.
- Per motivi pratici i browser "perdonano" molti errori nel codice HTML

HTML-XML: differenze

- XML consente di crearsi i propri tipi di documenti
- E’ facile scambiarsi dati: molti database accettano e producono file XML
- E’ possibile validare una struttura dati
- I tag proprietari per avere una resa grafica più precisa diventano inutili
- Le proprietà dei link sono molto più avanzate rispetto a quelle di HTML

HTML-XML: differenze

- È più facile creare viste diverse degli stessi dati



XML - standard associati

- Modeling Rules (DTD, Schema)
- Stylesheets (XSL)
- Linking (XLink, XPointer)

HTML - esempio di pagina

```
<HTML>
<HEAD>
<TITLE>Lime Jello Marshmallow Cottage Cheese Surprise</TITLE>
</HEAD>
<BODY>
<H3>Lime Jello Marshmallow Cottage Cheese Surprise</H3>
My grandma's favorite (may she rest in peace).
<H4>Ingredients</H4>
<TABLE BORDER="1">
<TR BGCOLOR="#308030"><TH>Qty</TH><TH>Units</TH><TH>Item</TH></TR>
<TR><TD>1</TD><TD>box</TD><TD>lime gelatin</TD></TR>
<TR><TD>500</TD><TD>g</TD><TD>multicolored tiny marshmallows</TD></TR>
<TR><TD>500</TD><TD>ml</TD><TD>cottage cheese</TD></TR>
<TR><TD></TD><TD>dash</TD><TD>Tabasco sauce (optional)</TD></TR>
</TABLE>
<P>
<H4>Instructions</H4>
<OL>
<LI>Prepare lime gelatin according to package instructions...</LI>
<!-- and so on -->
</BODY>
</HTML>
```

HTML - esempio di pagina

Lime Jello Marshmallow Cottage Cheese Surprise

My grandma's favorite (may she rest in peace).

Ingredients

Qty	Units	Item
1	box	lime gelatin
500	g	multicolored tiny marshmallows
500	ml	Cottage cheese
	dash	Tabasco sauce (optional)

Instructions

1. Prepare lime gelatin according to package instructions...

HTML - esempio di pagina

- Il significato degli elementi della pagina precedente non è reso dall'HTML:
  - gli elementi della prima colonna NON sono quantità, sono solo testo libero
  - è molto difficile fare un parsing automatico dei contenuti della pagina

XML - esempio di file

```
<?xml version="1.0" ?>
<Recipe>
  <Name>Lime Jello Marshmallow Cottage Cheese
    Surprise</Name>
  <Description>
   My grandma's favorite (may she rest in peace).
  </Description>
  <Ingredients>
    <Ingredient>
      <Qty unit="box">1</Qty>
      <Item>lime gelatin</Item>
    </Ingredient>
    <Ingredient>
      <Qty unit="g">500</Qty>
      <Item>multicolored tiny marshmallows</Item>
    </Ingredient>
    <Ingredient>
      <Qty unit="ml">500</Qty>
      <Item>Cottage cheese</Item>
    </Ingredient>
    <Ingredient>
      <Qty unit="dash"/>
      <Item optional="1">Tabasco
sauce</Item>
    </Ingredient>
  </Ingredients>
  <Instructions>
    <Step>
Prepare lime gelatin according to
package instructions
    </Step>
    <!-- And so on... -->
  </Instructions>
</Recipe>
```

XML - analisi

- <?xml version="1.0"?>

  Header

- <Ingredient>
  <Qty unit="box">1</Qty>
  <Item>lime gelatin</Item>
  </Ingredient>

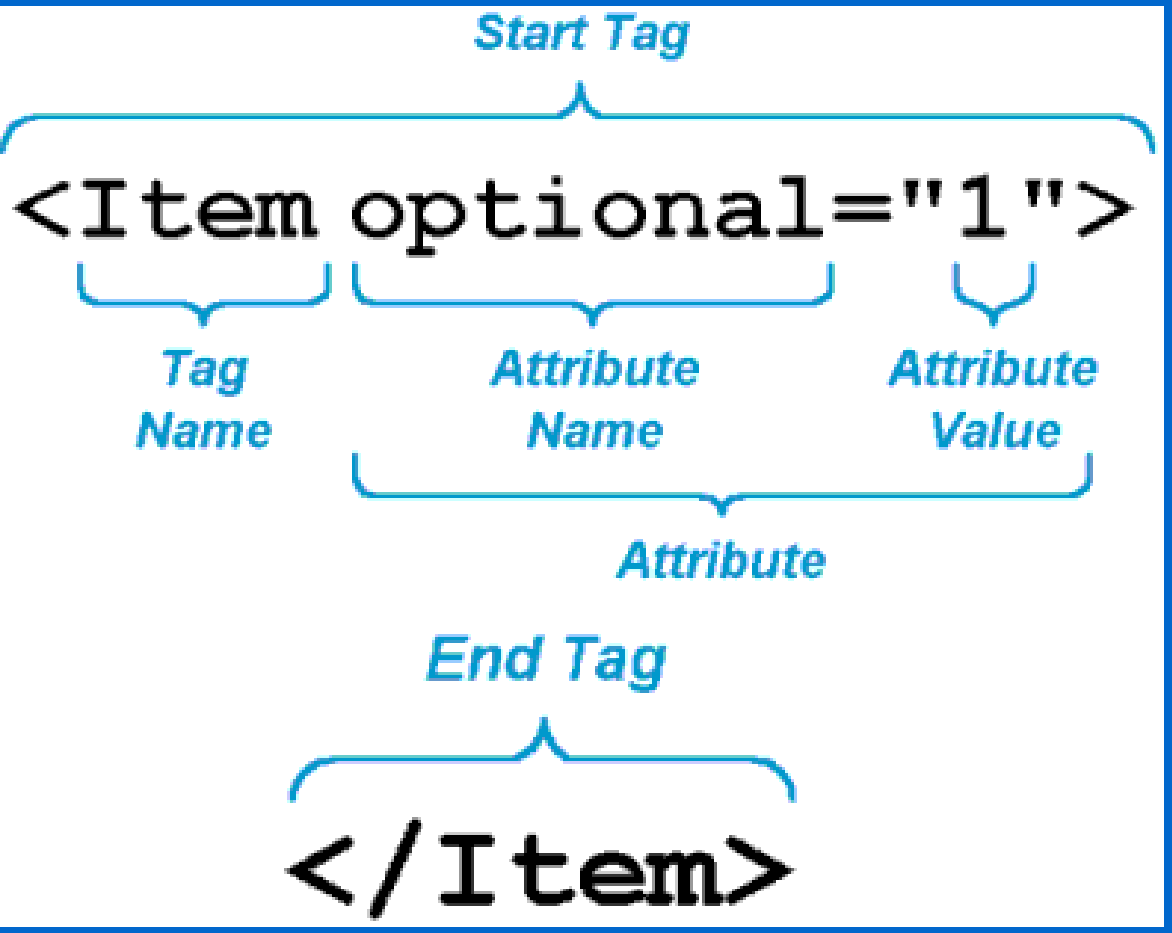
XML - analisi

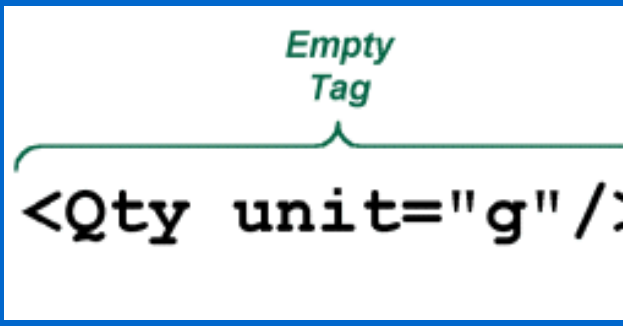
- Non c'è descritto come deve essere mostrata la ricetta.
- I tipi dei dati devono essere definiti in un DTD: document type definition

- Esistono già vari DTD standard:

• HL7 SGML/XML	industria ospedaliera
• MathML	matematica
• XML/EDI	elettronica
• FDX	scarpe
• WML 1.x	wireless markup language

Nomenclatura XML





XML - tag

- In XML i tag seguono regole molto più rigide che in HTML
- I documenti XML devono essere “well-formed”:
  - i tag devono essere chiusi, in HTML il seguente codice è accettabile:

```
<P>
blah blah
<P>
blah2 blah2
```

XML - tag

- I tag non possono sovrapporsi:
  <primotag> aaaaa **<secondotag>** bbbbbb **</primotag> </secondotag>**

  la forma accettabile è:

  <primotag> aaaaa **<secondotag>**
  bbbbbb
  </secondotag> </primotag>

XML - tag

- I valori degli attributi sono racchiusi da: “”
- in HTML possono anche essere “nudi”:

  <TABLE BORDER=1>

- Well-formed: il documento segue le regole dell'XML
- valido: il documento segue anche le regole del DTD

- <Ingredient>
  <Qty unit="box">1</Qty>
  <Qty unit=”g">5</Qty>
  <Item>lime gelatin</Item>
  </Ingredient>

Quale è la quantità giusta ??

DTD

- Il DTD è la grammatica del linguaggio di markup definita dal disegnatore del linguaggio.

- Nell'esempio precedente dobbiamo definire un DTD che specifica quali elementi esistono, con quali attributi, con quali relazioni reciproche ed in quale ordine si trovano.

- Un parser XML che valida un documento legge il documento ed il DTD, e controlla la corrispondenza nei confronti del DTD.

- I validatori per HTML fanno un lavoro simile.
- I browser HTML sono molto liberali...

DTD

- I DTD danno l'estendibilità dell'XML
- Usano una sintassi diversa dai documenti XML
  - gli "*schema*" dovrebbero essere documenti XML con funzioni di DTD. XML Schema è adesso al livello Recommended di W3C

DTD - esempio

```
<!-- DTD per le ricette -->
<!ELEMENT Recipe (Name, Description?, Ingredients?,
    Instructions?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Ingredients (Ingredient)*>
<!ELEMENT Ingredient (Qty, Item)>
<!ELEMENT Qty (#PCDATA)>
<!ATTLIST Qty unit CDATA #REQUIRED>
<!ELEMENT Item (#PCDATA)>
<!ATTLIST Item optional CDATA "0"
              isVegetarian CDATA "true">
<!ELEMENT Instructions (Step)+>
```

- <!ELEMENT Recipe (Name, Description?, Ingredients?, Instructions?)>

  <!ELEMENT...> definisce un tag di nome Recipe che contiene gli elementi tra parentesi.

- “?” significa che l’elemento è opzionale e può apparire 1 o 0 volte
- “+” significa 1 o più
- “*” significa 1,0 o più volte

- <!ELEMENT Name (#PCDATA)>

  L’elemento può contenere solo caratteri e nient’altro (parsed character data). Non può avere figli.

- <!ATTLIST Item optional CDATA "0" isVegetarian CDATA "true">

  Il tag ha due possibili attributi con relativo valore di default

- Un attributo può essere reso obbligatorio con #REQUIRED
- Se un attributo non è richiesto allora si può indicare con #IMPLIED

- I tipi di attributi possono essere:
  - stringhe (CDATA o character data)
  - token (ID, IDREF, ENTITY, NMTOKEN)
  - attributi enumerati

- Attributi tipo stringa (CDATA):
  - Possono assumere qualsiasi valore tranne:
    - <
    - >
    - &
    - ‘
    - “
  - Per rappresentare questi simboli si devono usare entità

- Attributi tipo token
  - Impongono dei limiti ai valori degli attributi:
    - Tutti gli ID devono essere diversi
    - IDREF deve riferirsi ad un ID esistente:
      - Un parser validante controlla la corrispondenza

- Errori:
  - Più di un ID per attributo
  - Il valore dell’ID non inizia con una lettera, _, o :

- Per usare un DTD ci deve essere un riferimento nel documento XML, dopo lo header:
  - <!DOCTYPE Recipe SYSTEM ”http://www.dsi.unifi.it/xml/example.dtd">

    Recipe è il tag di primo livello, di seguito c’è l’URL dove si trova il DTD

- Per usare un DTD esterno si deve inserire standalone="no" nello header.
- Si può anche definire un DTD interno al documento con:

<!DOCTYPE *root_element* [

Document Type Definition (DTD):
elements/attributes/entities/notations/processing instructions/comments/PE references

]>

- Dentro <!ELEMENT…> si possono creare espressioni complesse:
  - se si definisce <!ELEMENT x (a,(b | c | d)*, e)*> allora sono validi tutti i seguenti elementi:
    <x> <a /> <b /> <e /></x>
    <x></x>
    <x> <a /> <e /> <a /><c /><d /><e /></x>

- Un tag può essere vuoto se per esempio al posto di #PCDATA si usa EMPTY
- Può contenere sia testo che altri tag o qualsiasi altra cosa se è definito come ANY
- NON vanno messi tra parentesi o il parser pensa siano tag da definire nel DTD !

- Il DTD completo del caso precedente potrebbe essere:

```
<!ELEMENT x (a,(b | c | d)*, e)*>
<!ELEMENT a EMPTY>
<!ELEMENT b EMPTY>
<!ELEMENT c EMPTY>
<!ELEMENT d EMPTY>
<!ELEMENT e EMPTY>
```

- Il “|” indica una scelta. Può essere fatta tra altri ELEMENT o anche dati #PCDATA:

```
<?xml version="1.0"?>
<!DOCTYPE student [
    <!ELEMENT student (#PCDATA | id )*>
    <!ELEMENT id (#PCDATA)>
]>
<student>
    Ecco del testo
    <id>9216735</id>
    Il testo puo’ essere ovunque prima e dopo l’elemento.
    Puo’ anche non esserci l’ id.
</student>
```

- <!ELEMENT orario (#PCDATA)>
- <orario> 24 Maggio 2000 20:30</orario>

In alternativa:

- <!ELEMENT orario (EMPTY)>
- <!ATTLIST orario giorno CDATA #REQUIRED
  mese CDATA #REQUIRED
  anno CDATA #REQUIRED
  …..

- Un attributo può assumere solo certi valori predefiniti:
  - <!ATTLIST corso_laurea (Elettronica | Informatica | Meccanica | Civile | Ambientale) “Informatica”>

- Si possono definire entità (standard: < > ") aggiuntive:
  - <!ENTITY DSI “Dip. Sistemi e Informatica”>

    &DSI; verrà sostituito dalla stringa associata

video.dtd

```
<!-- DTD per video -->
<!ELEMENT video (Titolo, Riassunto?,Stream?)>
<!ELEMENT Titolo (#PCDATA)>
<!ELEMENT Riassunto (#PCDATA)>
<!ELEMENT Stream (Episodio)*>
<!ELEMENT Episodio (Shot*,Avvenimento)>
<!ELEMENT Avvenimento (#PCDATA)>
<!ELEMENT Shot (Inizio,Fine,Keyframe,Audio?,Mosaic?)+>
<!ELEMENT Inizio (#PCDATA)>
<!ELEMENT Fine (#PCDATA)>
<!ELEMENT Keyframe (#PCDATA)>
<!ELEMENT Audio (#PCDATA)>
<!ELEMENT Mosaic (#PCDATA)>
```

prova.xml

```
<!DOCTYPE video SYSTEM "video.dtd">

<video>
<Titolo>Ronin</Titolo>
<Riassunto>Film d'azione con Robert De Niro</Riassunto>
<Stream>
<Episodio>
<Shot>
<Inizio>1</Inizio>
<Fine>2345</Fine>
<Keyframe>http://www.dsi.unifi.it/keyframe/ronin1.jpg
</Keyframe>
</Shot>
<Avvenimento>
Vengono presentati i personaggi
</Avvenimento>
</Episodio>
</Stream>
</video>
```

Schema

- I DTD **non** sono documenti XML
  - quindi non sono manipolabili in modo automatico
  - hanno limitazioni, es.:
    - <altezza>1.85</altezza>
      <altezza>biondo</altezza>
      sono entrambi validi se definiti con
      <!ELEMENT altezza (#PCDATA)>

- Problema:
  - diversi standard: oltre a W3C anche Microsoft

- Uno Schema XML è un documento XML conforme ad un DTD che definisce la struttura di uno schema… questi DTD sono associati al parser… è nato prima l'uovo!

- Con gli Schema XML diventa possibile differenziare

  <altezza>1.85</altezza>
  <altezza>biondo</altezza>

  - il primo potrebbe essere valido, il secondo no

- Gli Schema di Microsoft normalmente sono contenuti in file .xml
- W3C invece usa .xsd
  - xsd è usato anche come prefisso del namespace
    - Attualmente si usa **xs** come prefisso
  - http://www.w3.org/XML/Schema.html tools che convertono DTD in Schema

- In IE >= 5.x c’è un parser XML che gestisce MS Schema
- Xerces valida documenti W3C Schema
  - si può usare anche XMLSpy

- Esempio Schema W3C:

```
<xsd:element name="bookID"
  type="catalogID" />
<xsd:simpleType name="catalogID"
  base="xsd:string">
    <xsd:pattern value="\ d{3} - \ d {4} - d {3} "/>
</xsd:simpleType>
```

- Nell’esempio precedente viene definito un tag bookID di tipo catalogID
- catalogID viene definito come tipo semplice (ovvero senza sottolementi) deve seguire il pattern 3 cifre, trattino, 4 cifre, trattino, 3 cifre
  - I pattern seguono le regole del Perl!

Schema: tipi dati predefiniti

xs:string	Stringa di caratteri
xs:integer	Numero intero
xs:decimal	Numero decimale
xs:boolean	Valore booleano
xs:date	Data
xs:time	Ora
xs:uriReference	URL

Tipi dati semplici

- Si definiscono tipi di dato semplici quelli relativi a elementi che non possono contenere altri elementi e non prevedono attributi
- Es:

<xs:element name=”capitolo” type=”xs:integer />

- In accordo a questa def, e' valida l'istruzione:

<capitolo>12</capitolo>

- Un attributo viene definito con <xsd:attribute… />

- Un tipo complesso:
  - <xsd:complexType name="indirizzo">
    - <xsd:element name="Via" type="xsd:string"/>
    - <xsd:element name="Num." type="xsd:int"/>
    - …
  - </xsd:complexType>

Una dichiarazione più complessa

```
<xs:element name=”capitolo”>
<xs:simpleType>
<xs:restriction base=”xs:integer”>
<xs:minInclusive value=”1”/>
<xs:maxInclusive value=”20”/>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

- Un xsd:complexType può contenere diversi xsd:element e xsd:complexType
- Il numero di volte per cui si può/deve ripetere un tag viene definito usando gli attributi min/maxOccurs

- Le scelte (| del DTD) e le sequenze (“(“ e “)” dei DTD) vengono definite usando:
  - xsd:choice ora xs:choice
  - xsd:sequence ora xs:sequence

Inclusione di Schema

- Si utilizza la seguente sintassi (caso della biblioteca)

<biblioteca xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance"
xsi:noNameSpaceSchemaLocation="biblioteca.xsd"
titolo="Esempio di schema per il corso xml">

Namespace

- I namespace sono un’aggiunta recente a XML, evitano la collisione tra definizioni di tag uguali.
- Si dichiarano usando l’attributo **xmlns:**
- NOTA: si può continuare ad usare un solo DTD !

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE students SYSTEM "unifi.dtd">
<?xml-stylesheet href="studente.css" type="text/css" ?>
<students xmlns:unifi="http://www.unifi.it/unifi">
<unifi:student>
    Ecco del testo
    <unifi:id>9216735</unifi:id>
    Il testo puo' essere ovunque prima e dopo l'elemento.
    Puo' anche non esserci l' &lt;id&gt;.
</unifi:student>
<unifi:student>
    Secondo studente testo
    <unifi:id>73633678</unifi:id>
</unifi:student>
</students>
```

- Il namespace punta ad una URI (Uniform Resource Identifier)

Ecco del testo **9216735** Il testo puo' essere ovunque prima e dopo l'elemento. Puo' anche non esserci l' <id>.
Secondo studente testo **73633678**

```
<!ELEMENT students (unifi:student)*>
<!ATTLIST students xmlns:unifi CDATA #FIXED "http://www.unifi.it/unifi">
<!ELEMENT unifi:student (#PCDATA | unifi:id )*>
<!ELEMENT unifi:id (#PCDATA)>
```

- Il namespace può essere usato anche per gli attributi
- NON c'è validazione con i namespace e DTD

- Possono essere usati più namespace all'interno di un documento, uno può essere di default

  - Si può anche richiedere che non ci sia un namespace di default:

    <unifi:student
    xmlns=''
    xmlns:unifi="http://www.unifi.it/unifi" >

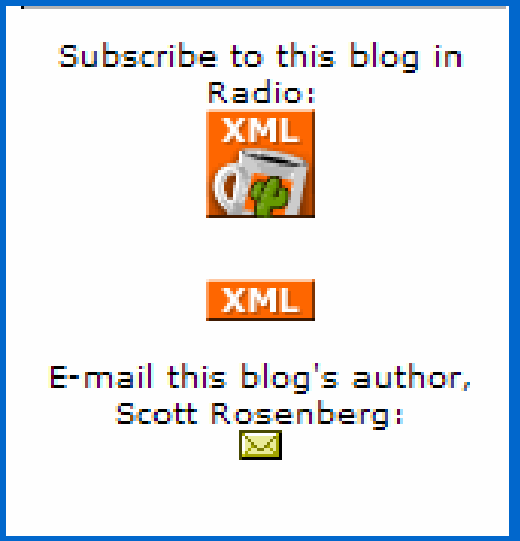
```
<furn:furniture xmlns:furn="http://myfurn/namespace"
    xmlns="http://www.w3.org/1999/xhtml">
    <table>
    <tr>
    <td><furn:table material="mahogany" type="dining"/></td>
    <td><furn:chair material="mahogany" type="dining"/></td>
    <td><furn:chair material="mahogany" type="dining"/></td>
    <td><furn:lamp material="brass" type="chandelier"/></td>
    </tr>
    </table>
    </furn:furniture>
```

- Es.: distinzione tra “table” di XHTML e “table” inteso come mobile

- La URI “assomiglia” ad una URL ma in realtà viene usata SOLO come identificatore univoco
  - è per questo motivo per cui è consigliabile usare una URL come URI...

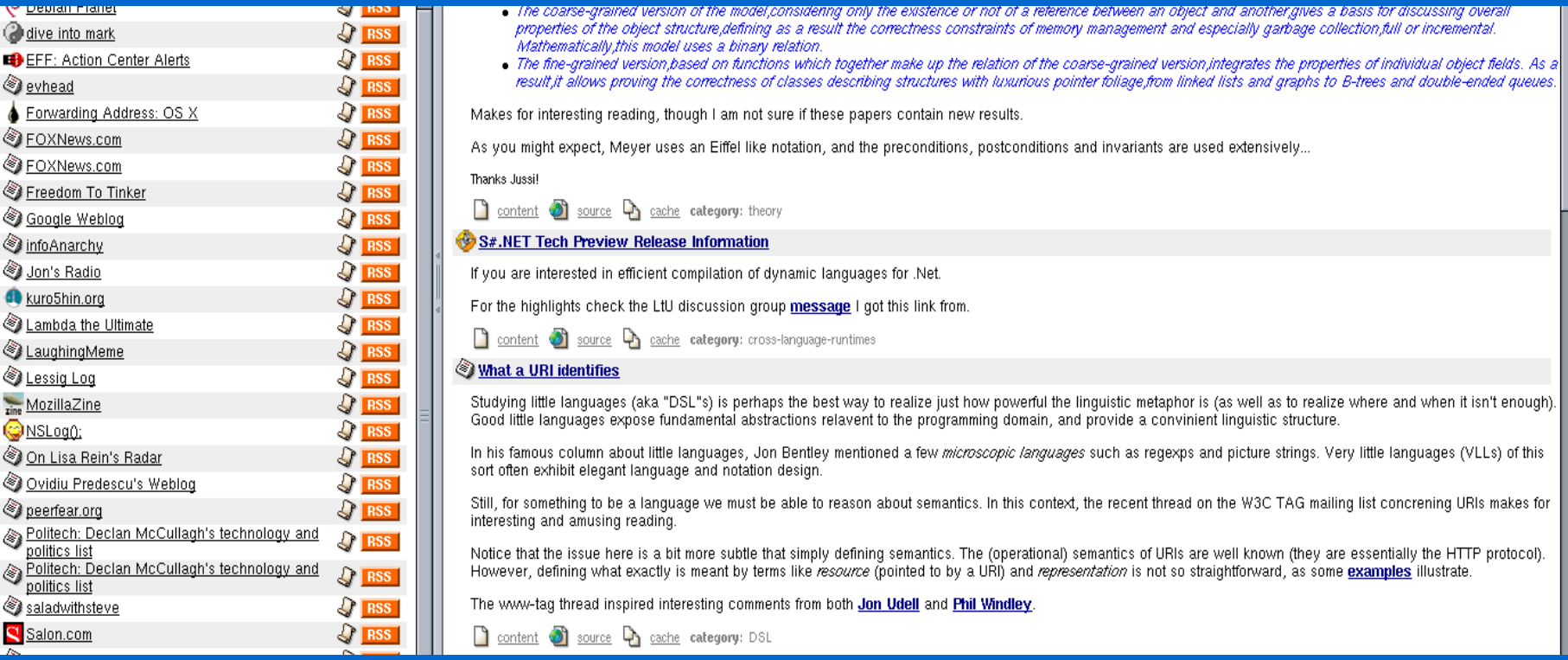
Namespace: un esempio

- RSS: RDF Site Summary (RDF: Resource Description Framework)
  - Indice dei contenuti di un sito, contiene link alle pagine del sito
  - Es.: molto usato nei blog



```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://purl.org/rss/1.0/"
>

  <channel rdf:about="http:///www.pie-r-squared.com/rss.rdf">
    <title>Pie-R-Squared</title>
    <description>
      Download a delicious pie from Pie-R-Squared!
    </description>
    <link>http://www.pie-r-squared.com</link>

    <image rdf:resource="http:///www.pie-r-squared.com/images/logo88x33.gif" />
    <textinput rdf:resource="http://www.pie-r-squared.com/search.pl" />

    <items>
      <rdf:Seq>
        <li rdf:resource="http://www.pie-r-squared.com/pies/pecan.html" />
        <li rdf:resource="http://www.pie-r-squared.com/pies/key_lime.html" />
      </rdf:Seq>
    </items>

  </channel>

  <image rdf:about="http:///www.pie-r-squared.com/images/logo88x33.gif">
    <title>Pie-R-Squared du Jour</title>
    <url>http://www.pie-r-squared.com/images/logo88x33.gif</url>
    <link>http://www.pie-r-squared.com</link>
  </image>

  <item rdf:about="http://www.pie-r-squared.com/pies/pecan.html">
    <title>Pecan Plenty</title>
    <link>http://www.pie-r-squared.com/pies/pecan.html</link>
  </item>
```

- Un programma può aggregare informazione da diversi siti usando RSS



- Per validare un documento con più namespace posso includere i vari Schema dei namespace in uno Schema globale

- <xs:import namespace="miaURI" schemaLocation="percorso_Schema_del_ namespace" />

- L’informazione su come rappresentare i dati è memorizzata da un’altra parte ed è scritta con un suo linguaggio di stile:
  - CSS e XSL - Cascading Style Sheets ed eXtensible Stylesheets Language

- I CSS sono già usati per l'HTML, XSL solo per XML (MS IE >= 5.0 e Mozilla)

  CSS per HTML si basa sull'idea di usare la struttura dell'HTML per indicare dove avvengono cambiamenti negli stili del testo

<HTML><HEAD></HEAD>
<BODY>
<H1>Parser Java per XML</H1>
Andando sul sito Alphaworks di IBM è possibile...
</BODY>
</HTML>

```
<STYLE TYPE="text/css">
<!--
H1 { color: red; font-size: 16pt; text-
   decoration: underline }
-->
</STYLE>
```

- Mettendolo in cima al documento HTML (o in un file separato) viene usato per cambiare tutte le occorrenze di <H1>

- <LINK REL="stylesheet" HREF="/gui/styles/master990806.css" TYPE="text/css"><STYLE TYPE="text/css"><!-- --> </STYLE>

```
B {font-weight: 700; font-size: 12pt;}
P {padding: 5px 0px; margin: 0px; font-size: 10pt; font-family: Arial,
    Helvetica, sans-serif;}
A   {text-decoration: none}
TD {font-size: 10pt; font-family: Arial, Helvetica, sans-serif;}
TD B{font-weight: 700; font-size: 10pt;}
```

- “Cascading” deriva dal fatto che le definizioni ricadono sui contenuti del tag a cui sono applicate, almeno fino a quando non viene trovata una nuova definizione.

CSS e XML

- I CSS si possono applicare direttamente ai documenti XML:
  - basta ridefinire i tag XML con la rappresentazione relativa
- La limitazione è che non possono trasformare i dati

CSS e XML

- La sintassi delle regole di applicazione dello stile segue lo schema:

```
selettore { proprieta':valore;
            proprieta': valore
            }
```

CSS e XML

- Esempio:

```
titolo { font-family: Arial;
font-size: 18pt;
color: #0000FF;
text-align: center;
}
```

Formatta l'elemento titolo

CSS e XML

- Selezione piu' precisa:

argomento titolo { font-family: Arial }

Formatta l'elemento <titolo> che sta dentro l'elemento <argomento>

immagine[file=”im1.jpg”] {font-family: Arial }

Formatta l'elemento <immagine> con attributo [file=”im1.jpg”]

CSS e XML

- La proprieta' DISPLAY
- Valori:

  block -> Il testo contenuto nell'elemento viene visualizzato su una nuova riga rispetto all'elemento precedente

  inline -> Il testo contenuto nell'elemento viene visualizzato sulla stessa riga dell'elemento precedente

  none -> Nessuna impostazione

CSS e XML

Proprietà	Valori	Descrizione
background-color	*red,green, blue, #666633,..*	*indica il colore dello sfondo di un elemento*
background-image	*url*	*specifica l'immagine da visualizzare come sfondo di un elemento*
border-style	*none, dotted, double,..*	*specifica lo stile del bordo da applicare a un elemento*
color	*red,green, blue, #666633,...*	*specifica il colore di un elemento*
display	*block, inline, none*	*specifica le modalità di visualizzazione di un elemento*
font-family	*times, helvetica, arial*	*indica il tipo di carattere da utilizzare*
font-size	*small, large, 10pt, 12pt,..*	*indica le dimensioni dei caratteri*
font-weight	*normal, bold, 100, 200, ..*	*indica lo spessore dei caratteri*
height, width	*30px, 40px, 75%, ..*	*indicano l'ampiezza e la larghezza di un elemento*
position	*absolute, relative, static*	*indica il tipo di posizionamento da applicare*
text-decoration	*none, underline, blink, ..*	*indica alcune decorazioni da applicare al testo, come ad esempio la sottolineatura*

CSS e XML

Proprietà	Valori	Descrizione
top, left	*30px, 40px, 50px*	*indicano le coordinate dell'angolo superiore sinistro di un elemento*
z-index	*1,2,3,..*	*specifica il posizionamento degli elementi nello spazio (sovrapposizione)*

CSS e XML

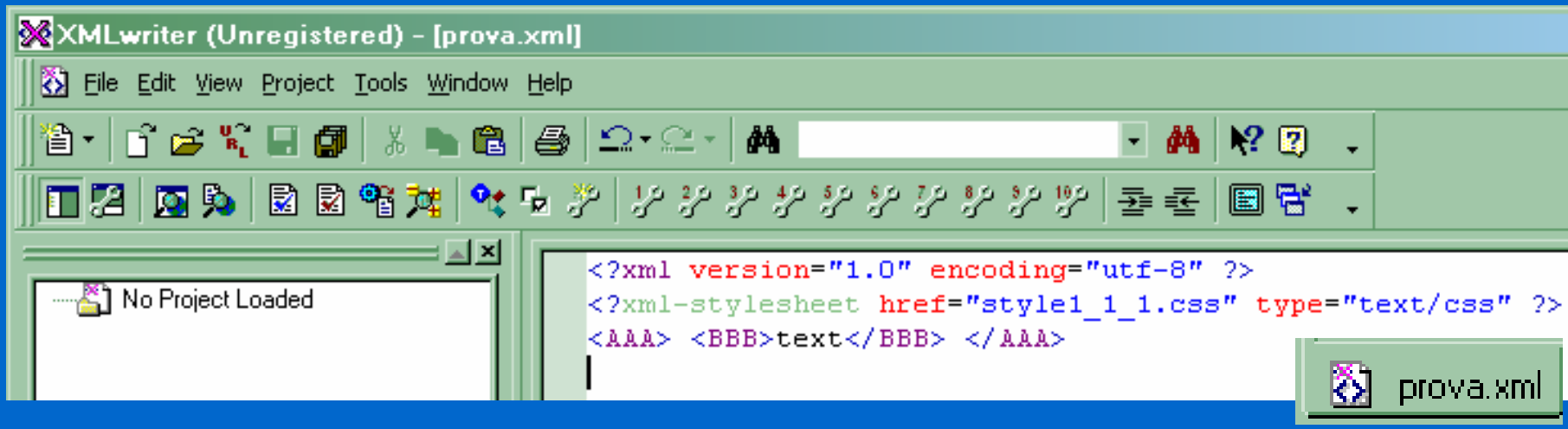
- Inclusione di un css in un documento XML
- Il css puo' essere dichiarato internamente o esternamente
- Di solito si conserva in un file esterno e si include con l'istruzione:

<?xml-stylesheet type=”text/css” href=”stile.css”?>

CSS e XML

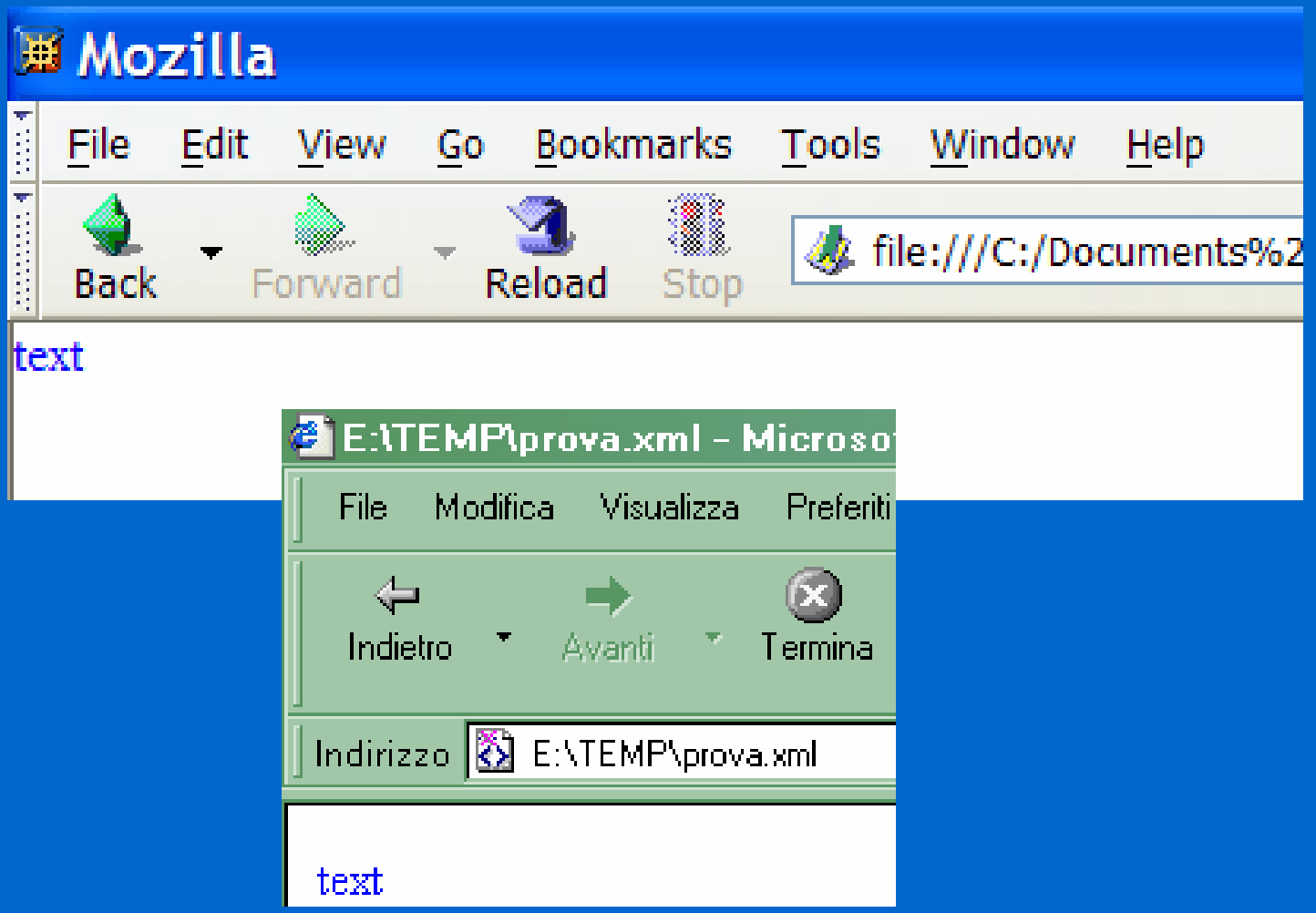
- Un CSS non può copiare un <TITLE> come un <Hx> della pagina, o comunque fare altre operazioni di ristrutturazione della pagina.
- XSL permette operazioni più complesse: trasformazione dei documenti e formattazione

CSS e XML


XMLwriter (Unregistered) - [prova.xml]
File Edit View Project Tools Window Help
No Project Loaded
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet href="style1_1_1.css" type="text/css" ?>
<AAA> <BBB>text</BBB> </AAA>
prova.xml


XMLwriter (Unregistered) - [style1_1_1.css]
File Edit View Project Tools Window Help
No Project Loaded
BBB {color:blue}
style1_1_1.css

CSS e XML


Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
file:///C:/Documents%2
text
E:\TEMP\prova.xml - Microso
File Modifica Visualizza Preferiti
Indietro Avanti Termina
Indirizzo E:\TEMP\prova.xml
text

XPATH

- XML Path Language (XPath) è una specifica di XSLT. Fornisce una notazione per accedere ai dati di un documento XML.
- Un documento XML può essere visto come un albero. I nodi dell'albero sono elementi, attributi e testo. Un'espressione XPath è usata durante la trasformazione XSLT per accedere ai nodi.

XPATH

- Nelle slide relative a XSL si vedranno espressioni XPath, usate per selezionare elementi di documenti XML
  - <xsl:value-of select=”Espressione_XPATH"/>
  - <xsl:template match=" Espressione_XPATH ">

XPATH

- Per accedere agli attributi di un’entità:
  - <xsl:value-of select=“entità/@attributo"/>
- E’ possibile anche accedere a vettori:
  - <xsl:value-of select=“entità[index]/@attributo"/>

- <books>
  - <book>
    - <title>Code complete</title>
    - <translation edition = “1”>French</translation>
    - <translation edition = “2”>French</translation>
  - </book>
  - <book>
    - <title>Rapid development</title>
    - <translation edition =“1”>French</translation>
  - <book> </books>

- L’espressione Xpath

  /books/book/translation[. = ‘French’]/../title

- può essere usata per estrarre il titolo dei libri che hanno il tag “translation” che contiene la parola “French”

XPATH

- Si possono usare diverse espressioni matematiche con XPath:
  - <xsl:template match="product[position() mod 2 = 1]">

    c’è match con i tag “product” dispari del documento XML

XPATH - esempio

```
xsl:template match="product[position() mod 2 = 1]">
  <tr class="odd">
     <td><xsl:value-of select="name"/></td>
     <td><xsl:value-of select="price"/></td>
     <td><xsl:value-of select="description"/></td>
  </tr>
/xsl:template>
xsl:template match="product">
  <tr class="even">
     <td><xsl:value-of select="name"/></td>
     <td><xsl:value-of select="price"/></td>
     <td><xsl:value-of select="description"/></td>
  </tr>
/xsl:template>
```

Name	Price	Description
Playfield Text	299	Faster than the competition.
Playfield Virus	199	Protect yourself against malicious coo
Playfield Calc	299	Clear picture on your data.
Playfield DB	599	Organize your data.

```
header { background-color: #999999; font-weight: bold; }
odd { background-color: normal; }
even { background-color: #dfdfdf; }
```

- Altri esempi di XPath:
  - NONNO/*/NIPOTE
  - PROGENITORE//DISCENDENTE equivalente a:
    - PROGENITORE/DISCENDENTE
    - PROGENITORE/*/DISCENDENTE
    - PROGENITORE/*/*/DISCENDENTE
    - ...

- Id('IDENTIFICATIVO')

- corrisponde ad un attributo che è stato definito come ID nel DTD
- es.:
  - <!ATTLIST NOME_TAG NOME_ATTRIBUTO ID>
    <NOME_TAG NOME_ATTRIBUTO="mioid">…</NOME_ATTRIBUTO>
    l'ID deve essere univoco !

- Dal nodo in analisi si possono estendere le ricerche usando gli assi, vale a dire quali nodi devono essere esaminati partendo dal nodo di partenza:

- // oppure descendant-or-self
- ancestor i nodi antenati
- following-sibling fratelli che seguono il nodo
- …

- //* tutti gli elementi che discendono dalla radice
- /* elementi figli della radice
- //*[count(padre)=2] tutti i “padre” che hanno due figli
- //*[count(*)=1] tutti gli elementi che hanno un solo figlio
- //*[name()=“pippo”] tutti gli elementi di nome pippo

- //*[string-length(name())>6] tutti gli elementi il cui nome è più lungo di 6 caratteri

- Altre funzioni:
  - starts-with()
  - contains()

- Posso selezionare nodi anche secondo il tipo, es.:
- text() testo
- comment() commenti
- node() qualsiasi nodo

XSL e XML

- La notazione del XSL è l'XML
- In pratica è un documento XML che dice come trasformare un altro documento XML
- Usando XSL diversi si ottengono più rappresentazioni dallo stesso XML

XSL e XML

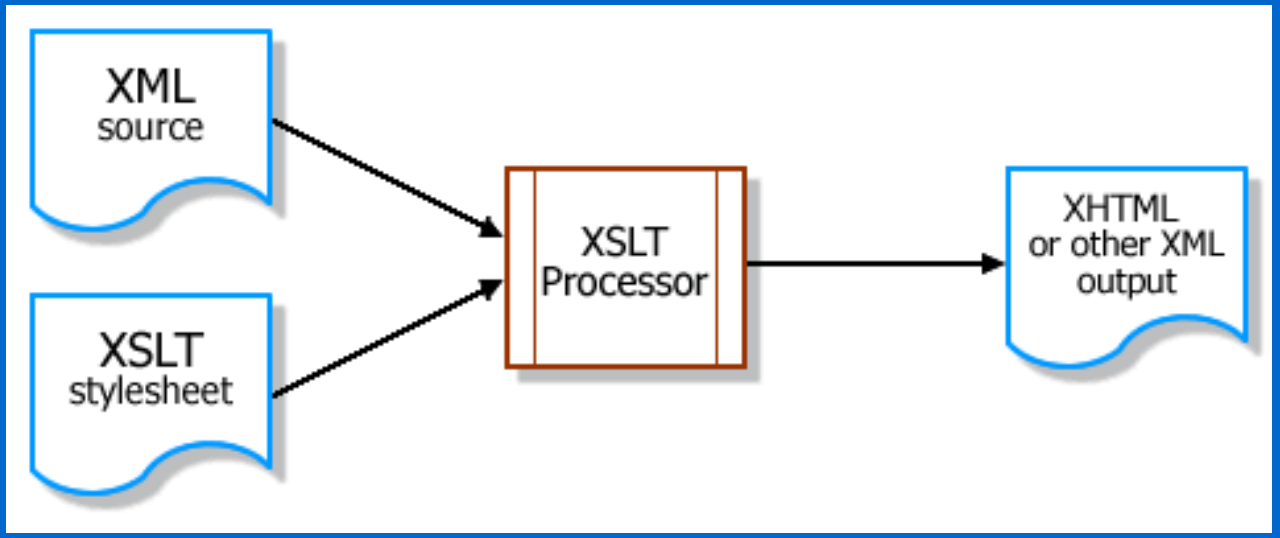
- Un file XSL è composto da una serie di regole (*template*) che sono applicate ad un file XML da un parser XSL.

- In realtà XSL deve ancora essere approvato (xsl 1.0 candidate recommendation). La parte di trasformazione (XSLT) è già stata approvata

- XSL comprende:
  - XSLT: linguaggio per effettuare la trasformazione di documenti XML
  - XSL-FO (Formatting Objects): vocabolario per specificare la semantica di formattazione

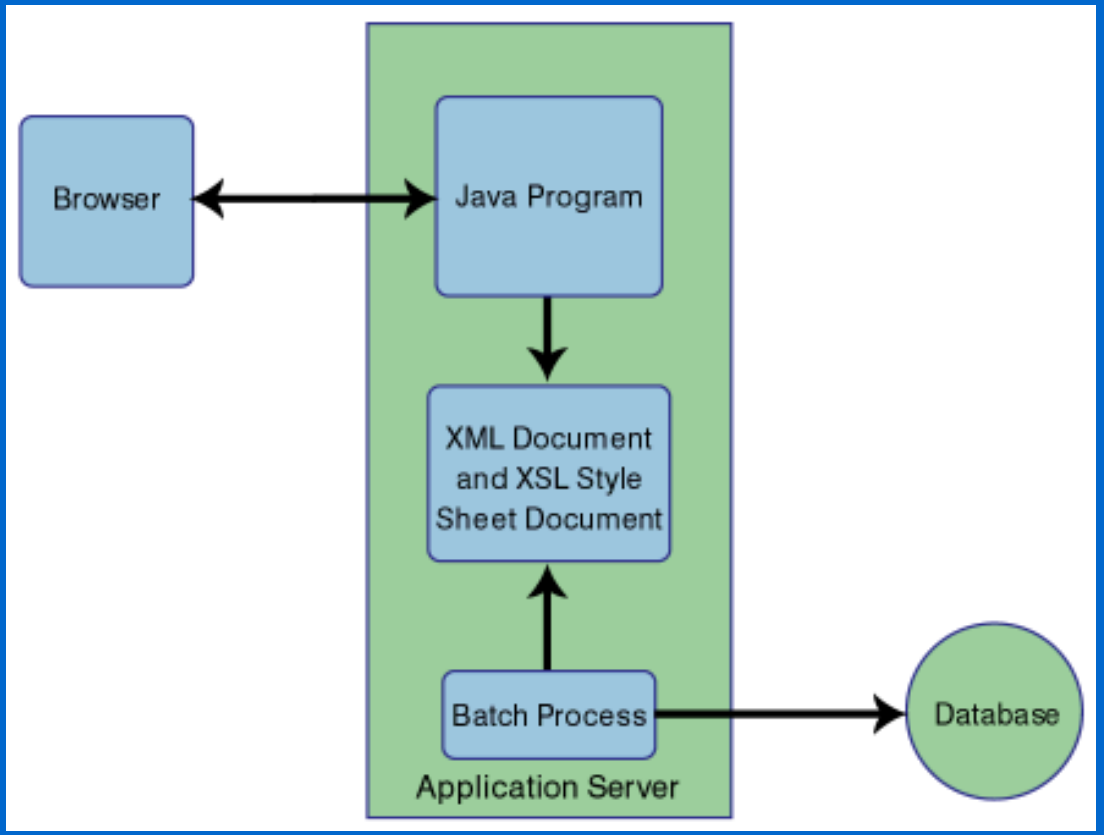
    - I FO sono il risultato di una trasformazione di un documento XML
      Praticamente sono un altro documento XML

- XSL-FO non è altro che un ulteriore vocabolario XML che specifica la trasformazione di un albero XML in un documento finale
- Apache FOP è un’appplicazione che trasforma un file di FO e genera PDF e altri formati testuali



- Diagramma generale di una trasformazione XSL

XSL e XML



- Creazione di pagine web con servlet

XSLT

- Apache ha già un modulo XSLT (Xalan) per generare per es. pagine HTML a partire dall'XML (http://xml.apache.org)
  - Xalan può essere eseguito anche a linea di comando
    - necessita del parser Xerces sempre fornito da xml.apache.org

- Xalan per Java viene eseguito come servlet dall'estensione JServ di Apache
  - Def.: una servlet può essere pensata come una applet eseguita sul lato server

- XSLT descrive regole per trasformare un albero XML sorgente in uno risultato

- Si scrivono regole modello che vengono attivate appena viene identificata una corrispondenza
- Il motore XSLT esmina l'albero sorgente un nodo alla volta ed esegue le regole corrispondenti
  - Quando viene identificata una corrispondenza le istruzioni contenute in una regola sono eseguite

- L’ordine delle regole nel documento XSL non è importante sono eseguite quando un’espressione XPath le raggiunge
- Il contenuto di un elemento in elaborazione viene esposto a tutte le altre regole usando il tag xsl:apply-templates

Cosa si puo' fare con XSL

- Trasformare elementi (p.es, per applicare informazioni di stile)
- Aggiungere/togliere elementi
- Non prendere in considerazione certe parti del documento, in base al risultato di certe elaborazioni
- Riordinare gli elementi

- Un foglio di stile XSLT ha la seguente struttura:

<?xml version=”1.0”?>

<xsl:stylesheet version=”1.0”
xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”>

....template di trasformazione ...

</xsl:stylesheet>

- Un generico template ha la seguente struttura:

<xsl:template match=”espressione XPath”>
....definizione dell'output...
</xsl:template>

- Esempio di template:

<xsl:template match=”/”>
Un esempio molto semplice (e inutile!)
</xsl:template>

- Esempio piu' significativo, che utilizza il contenuto dell'elemento:

```
<xsl:template match=”/”>
    Contenuto dell'attributo titolo:
    <xsl:value-of select=”corso/@titolo”/>
</xsl:template>
```

- L'applicazione dei template comincia a partire da quello che identifica la radice del documento
- Se non specifichiamo nient'altro, fatto questo il processore XSLT si ferma
- Per dirgli di applicare eventuali altri template dobbiamo specificare l'opzione

  <xsl:apply-templates/>
- Tipicamente, questo viene usato in tutti gli elementi tranne le foglie

XSL - esempio

```
<?xml version="1.0"?>
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/TR
    /WD-xsl">
<xsl:template match="Recipe”/>
      <HTML>
      <HEAD>
      <TITLE>
      <xsl:value-of select="Name"/>
      </TITLE>
      </HEAD>
      <BODY>
      <H3>
      <xsl:value-of select="Name"/>
      </H3>
      <STRONG>
      <xsl:value-of
select="Description"/>
      </STRONG>
      <xsl:apply-templates/>
      </BODY>
      </HTML>
</xsl:template>
```

XSL - esempio

```
<!-- Format ingredients -->
<xsl:template match="Ingredients">
    <H4>Ingredients</H4>
    <TABLE BORDER="1">
    <TR BGCOLOR="#308030"><TH>Qty</TH><TH>Units</TH><TH>Item</TH></TR>
    <xsl:for-each select="Ingredient">
    <TR>

    <!-- handle empty Qty elements separately -->
    <xsl:if test='Qty[not(.="")]' >
        <TD><xsl:value-of select="Qty"/></TD>
    </xsl:if>
    <xsl:if test='Qty[.=""]' >
        <TD BGCOLOR="#404040"> </TD>
    </xsl:if>
    <TD><xsl:value-of select="Qty/@unit"/></TD>
    <TD><xsl:value-of select="Item"/>
    <xsl:if test='Item/@optional="1"'>
        <SPAN> -- <em><STRONG>optional</STRONG></em></SPAN>
    </xsl:if>
    </TD>
    </TR>
    </xsl:for-each>
    </TABLE>
</xsl:template>
```

XSL - esempio

- Si nota che è presente l'header XML.
- I template sono racchiusi tra i tag <xsl:template ...> e </xsl:template ...>

- Tutti i tag che cominciano con <xsl:…> sono comandi XSL

- Il namespace definito nel tag xsl:stylesheet è un'etichetta speciale: l'XSLT la usa per selezionare certe trasformazioni piuttosto che altre:

  xmlns:xsl="http://www.w3.org/TR/WD-xsl“ - MSIE
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform“ - W3C

- IE 6.x e Mozilla rispettano le trasformazioni W3C
- IE 5.x e' “male”, NS 4.x e' “malissimo”

Applicazione della trasformazione

- Internamente (lato client), aggiungendo:
  <?xml-stylesheet type="text/xsl" href="corso.xsl"?>
- Esternamente (sul server), avviando il processore XSLT e dandogli il documento XML da trasformare
- NB: nel primo caso (analogamente ai css), la trasformazione e' solo interna (l'html non si vede)

- Esaminiamo alcune trasformazioni che si possono applicare a documenti XML tramite XSL:
- Muovere del testo con XSL
  - Per esempio si può copiare un titolo di un documento in uno header HTML
  - <xsl:value-of> inserisce il testo di un elemento dove vogliamo

- <!-- Esempio 1: last name first -->
  <xsl:text>Name: </xsl:text>
  <xsl:value-of select="last-name"/>
  <xsl:text>, </xsl:text>
  <xsl:value-of select="first-name"/>
- <!-- Esempio 2: first name first -->
  <xsl:text>Name: </xsl:text>
  <xsl:value-of select="first-name"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="last-name"/>

- Gli elementi degli esempi possono essere definiti da un DTD come:

- <!ELEMENT first-name (#PCDATA) >
  <!ELEMENT last-name (#PCDATA) >

- I nodi selezionati dall’attributo select di xsl:value-of sono definiti in termini di Xpath
- L’attributo select seleziona solo il primo nodo che soddisfa la condizione

- Per gestire selezioni multiple si usa:

- <xsl:for-each select="percorso_Xpath">
  …. miei tag ….
  </xsl:for-each>

- Ordinamento con XSL
  - Può essere utile generare viste differenti di un documento ordinando in modo diverso gli elementi
    - Es. ordinare gli ZIP (CAP)

- <xsl:for-each select="//zip">
  <xsl:sort order="ascending" select="."/>
  <xsl:value-of select="."/>
  </xsl:for-each>

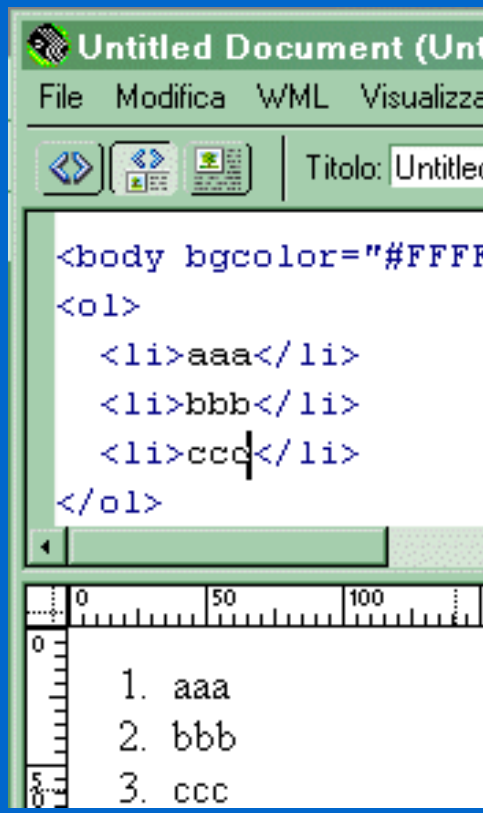
- L’elemento zip è definito da:
  <!ELEMENT zip (#PCDATA) >

- Inserire testo con XSL
  - Può essere necessario inserire del teso che non è presente nel documento XML originale, per cui non si può usare l’ operazione di spostamento vista prima

- Es. inserire la scritta “Totale” prima di una somma

```
<xsl:text>Totale: </xsl:text>
```

- Numerare liste
  - La numerazione di una lista può andare dalla creazione di semplici liste equivalenti ai codici HTML <LI> a numerazione di titoli e sottotitoli es.:
    1 Titolo grande
    1.1 Sezione
    1.1.1 Paragrafo



```
<xsl:for-each select="//section">
<h2>
<xsl:number level="multiple"
count="section" format="1.1.1. " />
<xsl:value-of select="./title"/>
</h2>
<xsl:apply-templates
select="./para"/>
</xsl:for-each>
```

- L’attributo multiple specifica che i numeri devono essere calcolati su più livelli
- Ogni aggiunta/eliminazione di sezioni nel documento XML verrà gestita dal file XSL automaticamente

- Il DTD di un documento a cui applicare la precedente trasformazione è:

```
<!ELEMENT section (title,para+,section*) >
<!ELEMENT title (#PCDATA) >
<!ELEMENT para (#PCDATA | note?)* >
<!ELEMENT note (#PCDATA) >
```

- Operazioni di calcolo
  - E’ possibile fare alcune semplici operazioni definite dallo standard Xpath, es. una somma:

```
<xsl:text>Totale: </xsl:text>
<xsl:value-of
select="sum(product)"/>
```

- L'elemento product dell’esempio può essere definito come

  <!ELEMENT product (#PCDATA) >

- E’ anche possibile aggiungere una formattazione ai numeri:

  <xsl:value-of select="format-number(sum(//product), '$#,##0.0')"/>

- Altre operazioni consentite da Xpath:
  - +, -, *, div, mod, ceiling, floor, round
- Operatori logici:
  - !=, <, <=, >, >=, |

- Es.:
  - <xsl:template match="GIOCATORI[position() > 11]">
    <xsl:value-of select="." />
    </xsl:template>

    potrebbe selezionare le riserve di una squadra di calcio

Da XML a HTML

- Per effettuare correttamente la trasformazione dobbiamo considerare sia il documento XML in ingresso che il formato dell'output che desideriamo

- Il primo template che deve essere creato è la radice del documento XML

- Internet Explorer 5 e superiori (e Mozilla) gestiscono molte delle trasformazioni di XSL, basta aggiungere un riferimento al file XSL che si vuole usare dentro il file XML:
  - <?xml version="1.0" encoding="ISO8859-1" ?>
    <?xml-stylesheet type="text/xsl"
    href=”miofile.xsl"?>

- Per migliorare la qualità della conversione da XML ad HTML si aggiunge:
  - <xsl:output method="html" indent="yes" version="4.0"/>

XSL - elementi

apply-imports
apply-templates
attribute
attribute-set
call-template
choose
comment
copy
copy-of
decimal-format
element
fallback
for-each
if
import
include
key
message
number
otherwise
output
param
preserve-space
processing-instruction
sort
strip-space
stylesheet
template
text
value-of
variable
when
with-param

XSL - funzioni

boolean
ceiling
comment
concat
contains
count
current
document
element-available
false
floor
format-number
function-available
generate-id
id
key
lang
last
local-name
name
namespace-uri
node
normalize-space
not
number
position
processing-instruction
round
starts-with
string
string-length
substring
substring-after
substring-before
sum
system-property
text
translate
true
unparsed-entity-uri

XSL - operatori

!=
"" (literal)
" (literal)
() (grouping)
* (all nodes)
* (multiplication)
+
-
- (unary minus)
. (self axis short form)
.. (parent axis short form)
/ (step separator)

// (descendant-or-self axis short form)
:: (axis specifier)
<
<=
=
>
>=
@ (attribute axis short form)
@* (all attributes)
[] (predicate)
and
axis nodetest predicate (step)

div
func() (function call)
mod
name (node test)
or
| (union) ing-instruction
round

- In alternativa si può usare un Javascript per rendere indipendente il file XML dal XSL associato
  - anche in questo caso la trasformazione avviene sul lato client > IE funziona, altri browser non supportano XSL

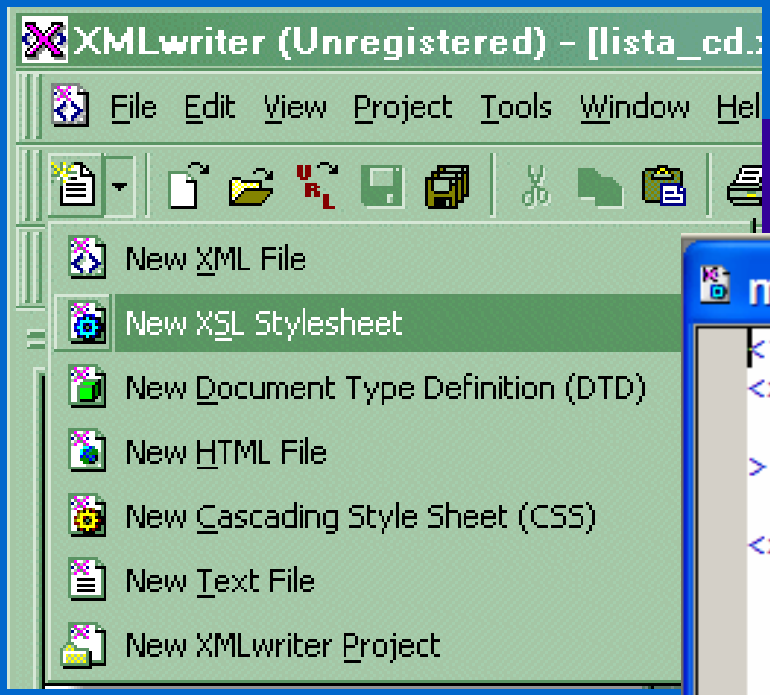
```
<html>
<body>
<script language="javascript">
// Load XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load(”miofile.xml")
// Load the XSL
var xsl = new ActiveXObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load(”miofile.xsl")
// Transform
document.write(xml.transformNode(xsl))
</script>

</body>
</html>
```

- Lo script precedente usa un controllo ActiveX Microsoft per creare un istanza di un parser per il file XML ed una seconda istanza per l'XSL

- L'unico metodo per avere una soluzione indipendente da browser e piattaforma rimane comunque quella della trasformazione sul lato server

- Gli esempi nel seguito possono essere eseguiti con IE e con Mozilla
- Generalmente sono mostrati due esempi, uno che funziona con IE 5.x ed uno che funziona con IE 6 e Mozilla

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="miotransform.xsl"?>

<CATALOGO>
<CD>
<TITOLO>Clandestino</TITOLO>
<AUTORE>Manu Chao</AUTORE>
<PREZZO>32.000</PREZZO>
</CD>
<CD>
<TITOLO>Esperanza</TITOLO>
<AUTORE>Manu Chao</AUTORE>
<PREZZO>32.000</PREZZO>
</CD>
<CD>
<TITOLO>Radio Chaos</TITOLO>
<AUTORE>Roger Waters</AUTORE>
<PREZZO>28.000</PREZZO>
</CD>
<CD>
<TITOLO>The Wall</TITOLO>
<AUTORE>Pink Floyd</AUTORE>
<PREZZO>32.000</PREZZO>
</CD>
</CATALOGO>
```

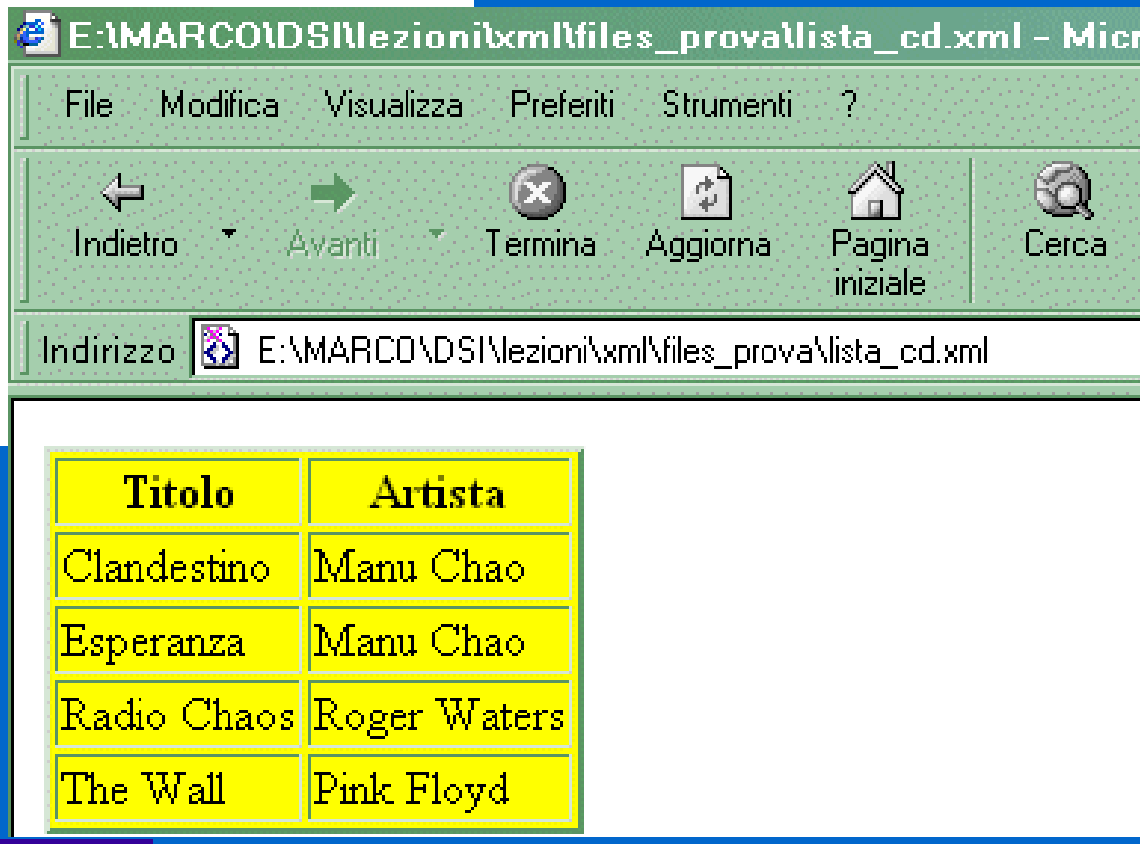


- XSL per il file XML precedente

miotransform.xsl

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform
>

<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Titolo</th>
        <th>Artista</th>
      </tr>
      <xsl:for-each select="CATALOGO/CD">
      <tr>
        <td><xsl:value-of select="TITOLO"/></td>
        <td><xsl:value-of select="ARTISTA"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>


</xsl:stylesheet>
```

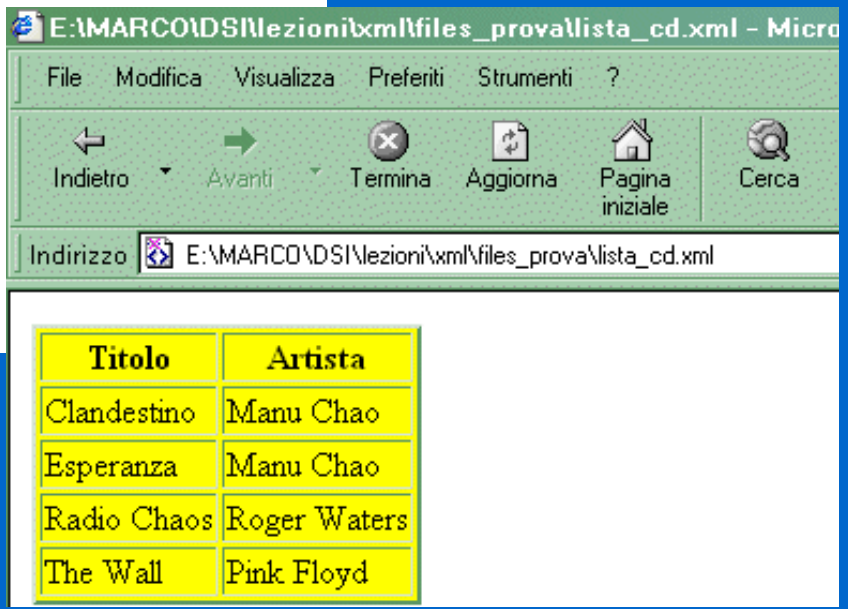
```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="miotransform.xsl"?>

<CATALOGO>
<CD>
<TITOLO>Clandestino</TITOLO>
<ARTISTA>Manu Chao</ARTISTA>
<PREZZO>32.000</PREZZO>
</CD>
<CD>
<TITOLO>Esperanza</TITOLO>
<ARTISTA>Manu Chao</ARTISTA>
<PREZZO>32.000</PREZZO>
</CD>
<CD>
<TITOLO>Radio Chaos</TITOLO>
<ARTISTA>Roger Waters</ARTISTA>
<PREZZO>28.000</PREZZO>
</CD>
<CD>
<TITOLO>The Wall</TITOLO>
<ARTISTA>Pink Floyd</ARTISTA>
<PREZZO>32.000</PREZZO>
</CD>
```



Titolo	Artista
Clandestino	Manu Chao
Esperanza	Manu Chao
Radio Chaos	Roger Waters
The Wall	Pink Floyd

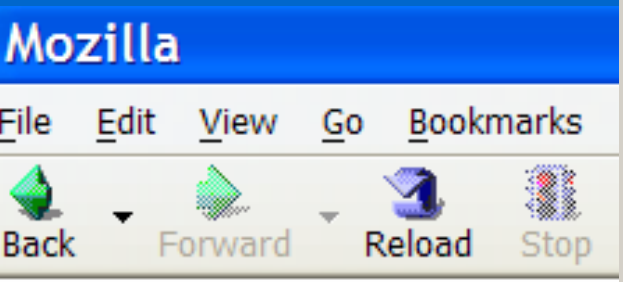
```
<?xml version='1.0'?>
<xsl:stylesheet
      xmlns:xsl="http://www.w3.org/TR/WD-xsl"
      xmlns="http://www.w3.org/TR/REC-html40"
      result-ns="">

<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Titolo</th>
        <th>Artista</th>
      </tr>
      <xsl:for-each select="CATALOGO/CD" order-by="+ ARTISTA">
      <tr>
        <td><xsl:value-of select="TITOLO"/></td>
        <td><xsl:value-of select="ARTISTA"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>


</xsl:stylesheet>
```

- Vecchia versione con ordinamento



- Nuova versione con ordinamento

```
<xsl:stylesheet version="1.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Titolo</th>
        <th>Artista</th>
      </tr>
      <xsl:for-each select="CATALOGO/CD">
        <xsl:sort select="ARTISTA" order="ascending" />
            <tr>
                <td><xsl:value-of select="TITOLO"/></td>
                <td><xsl:value-of select="ARTISTA"/></td>
            </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>


</xsl:stylesheet>
```

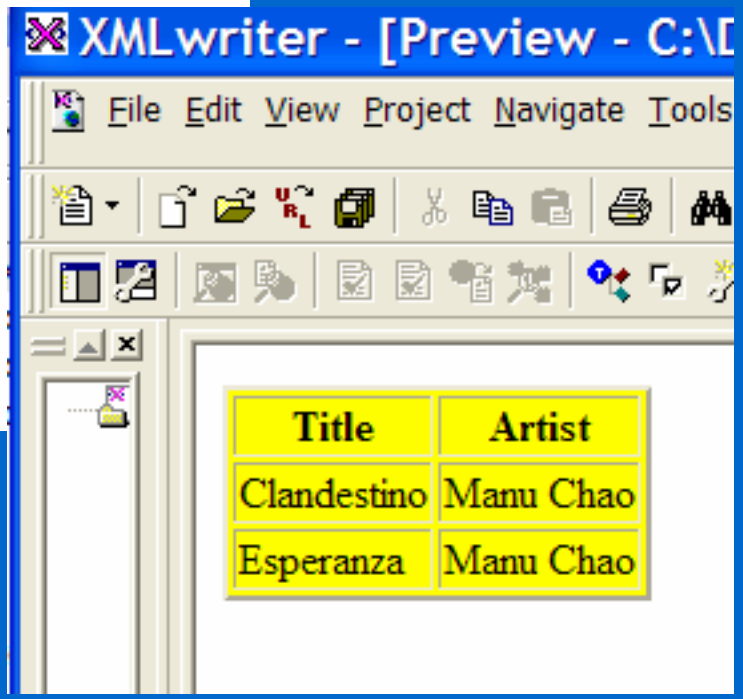


Titolo	Artista
Clandestino	Manu Chao
Esperanza	Manu Chao
The Wall	Pink Floyd
Radio Chaos	Roger Waters

```
<?xml version='1.0'?>
<xsl:stylesheet
       xmlns:xsl="http://www.w3.org/TR/WD-xsl"
       xmlns="http://www.w3.org/TR/REC-html40"
       result-ns="">

<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="CATALOGO/CD[ARTISTA='Manu Chao']">
      <tr>
        <td><xsl:value-of select="TITOLO"/></td>
        <td><xsl:value-of select="ARTISTA"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

- Vecchia versione

```
<xsl:stylesheet version="1.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Title</th>
        <th>Artist</th>
      </tr>
      <xsl:for-each select="CATALOGO/CD[ARTISTA='Manu Chao']">
      <tr>
        <td><xsl:value-of select="TITOLO"/></td>
        <td><xsl:value-of select="ARTISTA"/></td>
      </tr>
      </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>



</xsl:stylesheet>
```



- Nuova versione

- Operazioni condizionate
  - **<xsl:if test="ARTISTA=’Manu Chao'">**
    ... Accade qualcosa ...
    </xsl:if>

- Operazioni sugli attributi

```
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="miotransform5.xsl"?>

<CATALOGO>
<CD>
<TITOLO>Clandestino</TITOLO>
<ARTISTA>Manu Chao</ARTISTA>
<PREZZO tipo="pieno">32.000</PREZZO>
</CD>
<CD>
<TITOLO>Esperanza</TITOLO>
<ARTISTA>Manu Chao</ARTISTA>
<PREZZO tipo="scontato">30.000</PREZZO>
</CD>
<CD>
<TITOLO>Radio Chaos</TITOLO>
<ARTISTA>Roger Waters</ARTISTA>
<PREZZO tipo="pieno">28.000</PREZZO>
</CD>
<CD>
<TITOLO>The Wall</TITOLO>
<ARTISTA>Pink Floyd</ARTISTA>
<PREZZO tipo="scontato">32.000</PREZZO>
</CD>
</CATALOGO>
```

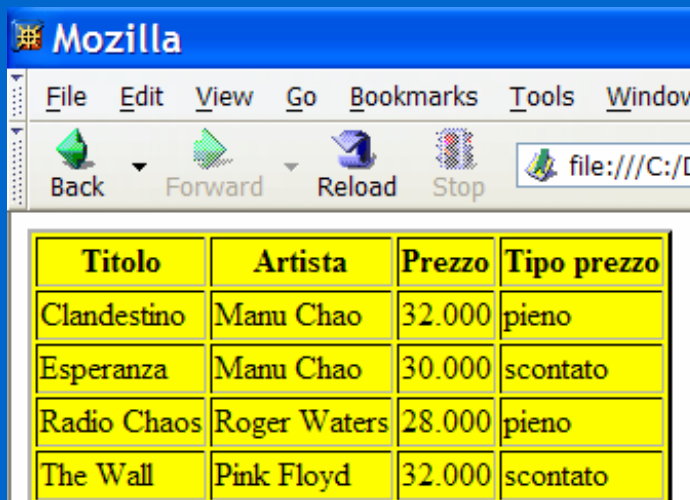
```xml
<?xml version='1.0'?>
<xsl:stylesheet
       xmlns:xsl="http://www.w3.org/TR/WD-xsl"
       xmlns="http://www.w3.org/TR/REC-html40"
       result-ns="">

<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Titolo</th>
        <th>Artista</th>
        <th>Prezzo</th>
        <th>Tipo prezzo</th>
      </tr>
      <xsl:for-each select="CATALOGO/CD">
          <tr>
          <td><xsl:value-of select="TITOLO"/></td>
          <td ><xsl:value-of select="ARTISTA"/></td>
          <td ><xsl:value-of select="PREZZO"/></td>
          <xsl:for-each select="PREZZO">
          <td ><xsl:value-of select="@tipo"/></td>
          </xsl:for-each>
          </tr>
       </xsl:for-each>
    </table>
  </body>
```

Titolo	Artista	Prezzo	Tipo prezzo
Clandestino	Manu Chao	32.000	pieno
Esperanza	Manu Chao	30.000	scontato
Radio Chaos	Roger Waters	28.000	pieno
The Wall	Pink Floyd	32.000	scontato

- Vecchia versione

```
xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"


xsl:template match="/">
 <html>
 <body>
   <table border="2" bgcolor="yellow">
     <tr>
       <th>Titolo</th>
       <th>Artista</th>
       <th>Prezzo</th>
       <th>Tipo prezzo</th>
     </tr>
     <xsl:for-each select="CATALOGO/CD">
         <tr>
         <td><xsl:value-of select="TITOLO"/></td>
         <td ><xsl:value-of select="ARTISTA"/></td>
         <td ><xsl:value-of select="PREZZO"/></td>
         <xsl:for-each select="PREZZO">
         <td ><xsl:value-of select="@tipo"/></td>
         </xsl:for-each>
         </tr>
      </xsl:for-each>
   </table>
 </body>
 </html>
/xsl:template>


/xsl:stylesheet>
```

- Nuova versione



Titolo	Artista	Prezzo	Tipo prezzo
Clandestino	Manu Chao	32.000	pieno
Esperanza	Manu Chao	30.000	scontato
Radio Chaos	Roger Waters	28.000	pieno
The Wall	Pink Floyd	32.000	scontato

- Test condizionali

```
<xsl:choose>
    <xsl:when test="ARTISTA='Manu Chao'">
        ... qualcosa ...
    </xsl:when>
    <xsl:when test="ARTISTA='qualcosaltro'">
        … qualcosaltro…
    </xsl:when>
    <xsl:otherwise>
        ... qualcosa ....
    </xsl:otherwise>
</xsl:choose>
```

```
<?xml version='1.0'?>
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/TR/WD-xsl"
        xmlns="http://www.w3.org/TR/REC-html40"
        result-ns="">

<xsl:template match="/">
  <html>
  <body>
    <table border="2" bgcolor="yellow">
      <tr>
        <th>Titolo</th>
        <th>Artista</th>
      </tr>
      <xsl:for-each select="CATALOGO/CD">
      <tr>
      <td><xsl:value-of select="TITOLO"/></td>
        <xsl:choose>
          <xsl:when match=".[ARTISTA='Manu Chao']">
            <td bgcolor="#ff0000"><xsl:value-of select="ARTISTA"/></td>
          </xsl:when>
          <xsl:otherwise>
            <td><xsl:value-of select="ARTISTA"/></td>
          </xsl:otherwise>
        </xsl:choose>
        </tr>
      </xsl:for-each>
    </table>
  </body>
```

Titolo	Artista
Clandestino	Manu Chao
Esperanza	Manu Chao
Radio Chaos	Roger Waters
The Wall	Pink Floyd

- Vecchia versione

```
l:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"


l:template match="/">
html>
body>
 <table border="2" bgcolor="yellow">
   <tr>
     <th>Titolo</th>
     <th>Artista</th>
   </tr>
   <xsl:for-each select="CATALOGO/CD">
   <xsl:choose>
     <xsl:when test="ARTISTA='Manu Chao'">
       <tr>
       <td><xsl:value-of select="TITOLO"/></td>
       <td bgcolor="red"><xsl:value-of select="ARTISTA"/></td>
     </tr>
     </xsl:when>
     <xsl:otherwise>
         <tr>
         <td><xsl:value-of select="TITOLO"/></td>
         <td bgcolor="white"><xsl:value-of select="ARTISTA"/></td>
         </tr>
     </xsl:otherwise>
   </xsl:choose>
   </xsl:for-each>
 </table>
/body>
/html>
sl:template>
```

- Nuova versione

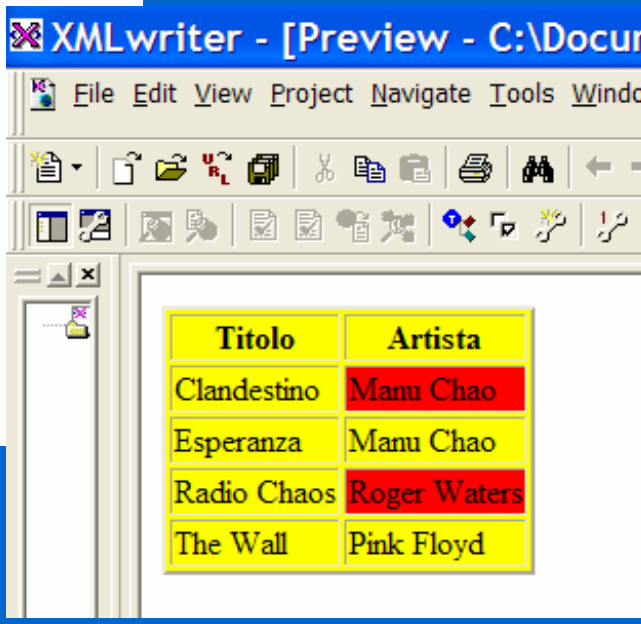
XMLwriter - [Preview - C:\Document

File Edit View Project Navigate Tools Window Hel

Titolo	Artista
Clandestino	Manu Chao
Esperanza	Manu Chao
Radio Chaos	Roger Waters
The Wall	Pink Floyd

```xml
<?xml version='1.0'?>
<?xml-stylesheet type="text/xsl" href="miotransform6.xsl"?>

<CATALOGO>
<CD>
<TITOLO>Clandestino</TITOLO>
<ARTISTA>Manu Chao</ARTISTA>
<PREZZO tipo="pieno">32.000</PREZZO>
</CD>
<CD>
<TITOLO>Esperanza</TITOLO>
<ARTISTA>Manu Chao</ARTISTA>
<PREZZO tipo="scontato">30.000</PREZZO>
</CD>
<CD>
<TITOLO>Radio Chaos</TITOLO>
<ARTISTA>Roger Waters</ARTISTA>
<PREZZO tipo="pieno">28.000</PREZZO>
</CD>
<CD>
<TITOLO>The Wall</TITOLO>
<ARTISTA>Pink Floyd</ARTISTA>
<PREZZO tipo="scontato">32.000</PREZZO>
</CD>
</CATALOGO>
```

```
xml version="1.0"?>
sl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"


sl:template match="/">
<html>
<body>
  <table border="2" bgcolor="yellow">
    <tr>
      <th>Titolo</th>
      <th>Artista</th>
    </tr>
    <xsl:for-each select="CATALOGO/CD">
    <tr>
    <td><xsl:value-of select="TITOLO"/></td>
      <xsl:choose>
        <xsl:when test="./PREZZO/@tipo='pieno'">
          <td bgcolor="#ff0000"><xsl:value-of select="ARTISTA"/></td>
        </xsl:when>
        <xsl:otherwise>
          <td><xsl:value-of select="ARTISTA"/></td>
        </xsl:otherwise>
      </xsl:choose>
      </tr>
    </xsl:for-each>
  </table>
</body>
</html>
xsl:template>


xsl:stylesheet>
```



- Per generare viste diverse di un documento XML si possono passare dei parametri dalla URL all'XSL
  - Per esempio uno script riconosce il browser collegato e genera viste ottimizzate, e.g. HTML o WML

- <xsl:param name="url" select="' '"/>
  <xsl:variable name=”isWML" select="substring-after($url, '?')" />
  <xsl:choose>
  <xsl:when test="**$isWML=’’true**">
  <xsl:template match="/">
  <?xml version="1.0"?>
  <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
  <wml><card>

- E’ possibile passare parametri anche tramite script:
  - IIS: mediante script ASP
  - Apache: tramite Xalan

- Es.: creazione di link

- Si deve creare l’attributo href del tag HTML <A> per creare il link, serve anche l’attributo name del tag A per creare l’anchor

- Per creare un link dobbiamo creare un tag

- Questa operazione può essere fatta in generale quando si trasforma un documento XML in un altro documento XML:
  - Trasformare un tag (o attributo) in un nuovo tag

- xsl:element e xsl:attribute sono gli elementi XSL che fanno al caso nostro
  - NOTA: xsl:text fa parte della stessa famiglia: il testo (es. contenuto in un tag) è un nodo foglia, come un tag vuoto.

- Per aggiungere un set di attributi:

- <xsl:attribute-set name="nomeset">
  <xsl:attribute name="nome1">xx</xsl:attribute>
  <xsl:attribute name="nome2">yy</xsl:attribute>
  ….
  </xsl:attribute-set>

- Per usare un set di attributi:

<QUALCHETAG xsl:use-attribute-sets=“nomeset”>
xxxx
</QUALCHETAG>

```
    <th>Artista</th>
    <th>Prezzo</th>
    <th>Link</th>
  </tr>

  <xsl:for-each select="CATALOGO/CD">
     <tr>
     <xsl:element name="a">
      <xsl:attribute name="name">
        <xsl:value-of select="ID" />
      </xsl:attribute>
    </xsl:element>

      <td><xsl:value-of select="TITOLO"/></td>
      <td ><xsl:value-of select="ARTISTA"/></td>
      <td ><xsl:value-of select="PREZZO"/></td>
      <td>
        <xsl:element name="a">
        <xsl:attribute name="href">
        <xsl:text>#</xsl:text>
        <xsl:value-of select="IDREF"/>
        </xsl:attribute>
        </xsl:element>
    </td>
    </tr>
   </xsl:for-each>
```

- Il risultato del frammento XSL visto prima e' la creazione di un tag (HTML):

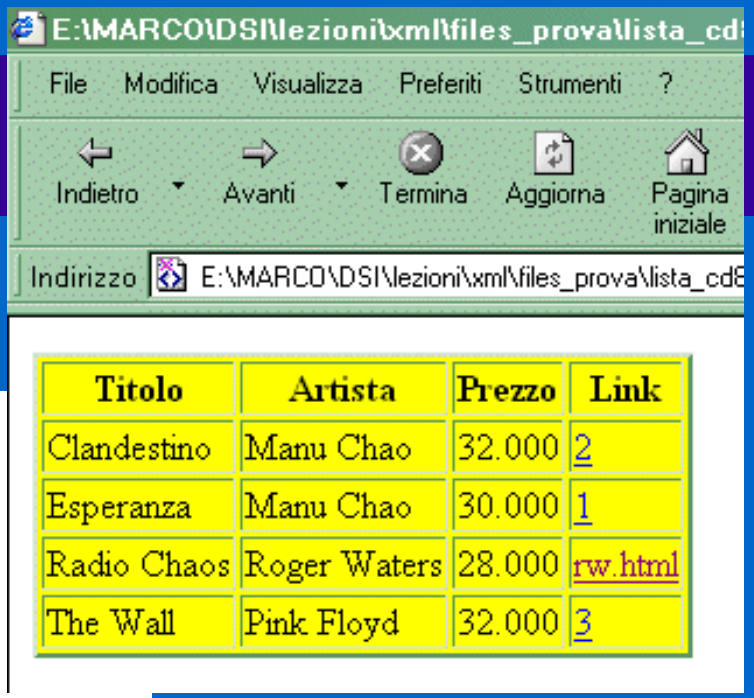
- <A HREF="#xxxxx"></A>

- Se vogliamo creare:
  - <A HREF="#xxxxx">xxxxx</A>
- Dobbiamo fare come nel frammento seguente

```xml
<CD>
<TITOLO>Radio Chaos</TITOLO>
<ARTISTA>Roger Waters</ARTISTA>
<PREZZO tipo="pieno">28.000</PREZZO>
<ID>3</ID>
<IDREF>rw.html</IDREF>
</CD>
```

```xml
<xsl:for-each select="CATALOGO/CD">
   <tr>
   <xsl:element name="a">
    <xsl:attribute name="name">
      <xsl:value-of select="ID" />
    </xsl:attribute>
  </xsl:element>

    <td><xsl:value-of select="TITOLO"/></td>
    <td ><xsl:value-of select="ARTISTA"/></td>
    <td ><xsl:value-of select="PREZZO"/></td>
    <td>
      <xsl:element name="a">
      <xsl:attribute name="href">
      <!-- xsl:text>#</xsl:text -->
      <xsl:value-of select="IDREF"/>
      </xsl:attribute>
      <xsl:value-of select="IDREF"/>
      </xsl:element>
  </td>
  </tr>
 </xsl:for-each>
```



- Accesso a elementi
  - <xsl:value-of select=”CD[1]/TITOLO"/>
  - <xsl:value-of select=”CD[last()]/TITOLO "/>

- E’ possibile anche creare indici (coppie key-value) a cui accedere con l’espressione key()
  - <xsl:key name="autref" match="/CD" use="ARTISTA"/>
  - <xsl:value-of select="key('autref', ARTISTA)"/>
  - È una specie di lookup table

- In generale:

- **<xsl:key**
  match="pattern” **// nodi a cui si applica**
  name=”identificatore_key"
  use="expression” **// espressione usata per determinare il valore della chiave**
  >
  </xsl:key>

- Es. in un glossario:

```
<glentry id="GLE-applet">
<term id="GLT-applet"
xreftext="applet">
applet
</term></glentry>
```

- <glentry id="GLE-servlet">
  <term id="GLT-servlet" xreftext="servlet">
  servlet
  </term>
  <defn id="GLD-servlet-001">
  ...
  Vedi anche <xref refid="GLT-applet" />.
  </defn>
  </glentry>

- Creo l’indice del glossario con la seguente key:

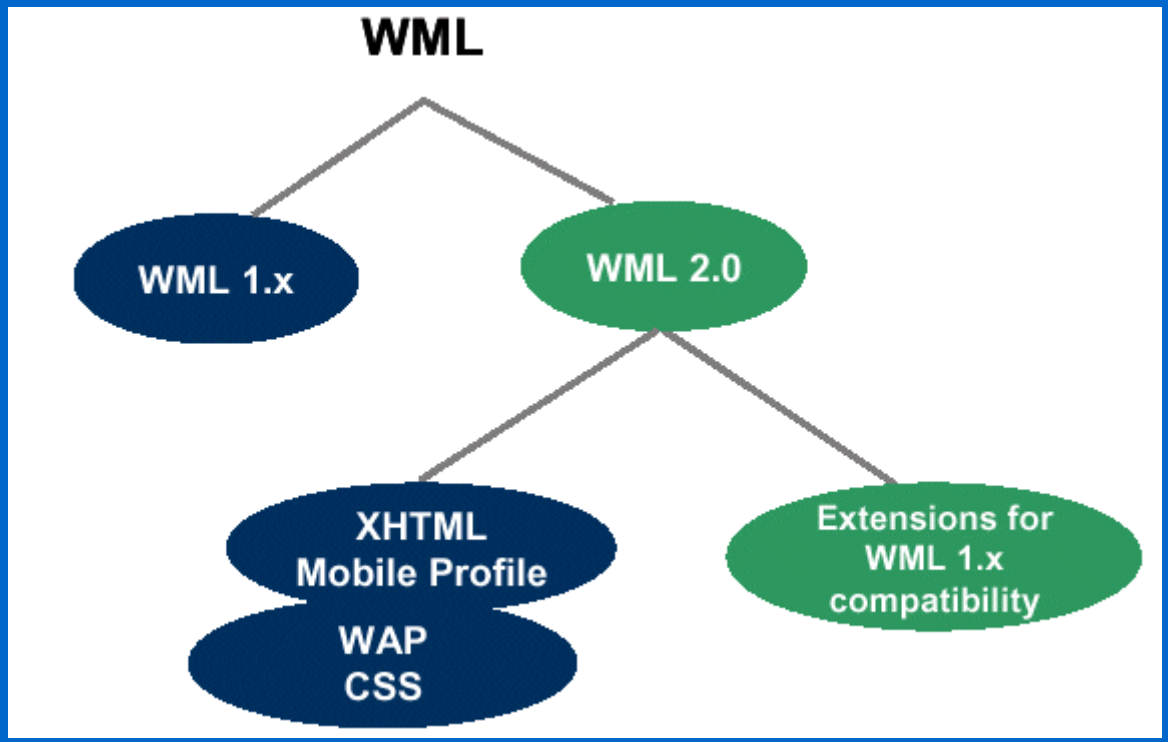
- ```
  <xsl:key name="termref"
  match="/glossary/glentry/term"
  use="@id"/>
  ```

- Con la key posso anche creare link come nell’esempio seguente, dove uso il testo definito in xreftext, usando il secondo “value-of select”
  ```

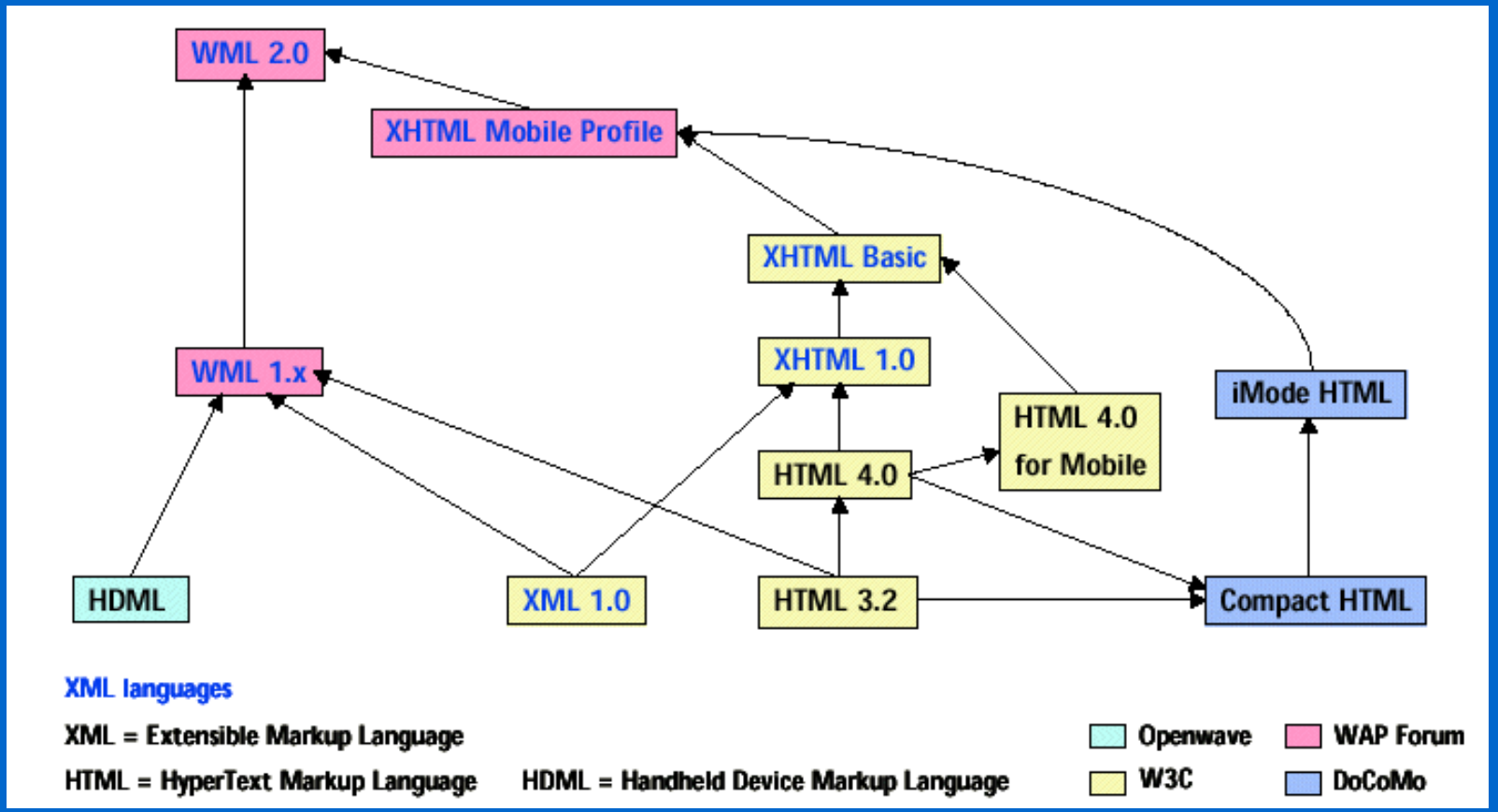
```
<xsl:template match="xref">
<xsl:element name="a">
<xsl:attribute name="href">
<xsl:text>#</xsl:text>
<xsl:value-of select="@refid"/>
</xsl:attribute>
<xsl:value-of
select="key('termref',
@refid)/@xreftext"/>
</xsl:element>
</xsl:template>
```

WML 2.0

- WML 2.0 si basa XHTML:



- XHTML Basic è un subset per dispositivi come PDA e cellulari.
- XHTML Mobile Profile parte da un set ristretto di XHTML Basic aggiungendo però elementi specifici per dispositivi mobili. E’ uno standard curato dal WAPForum
- WAP CSS è un subset di CSS

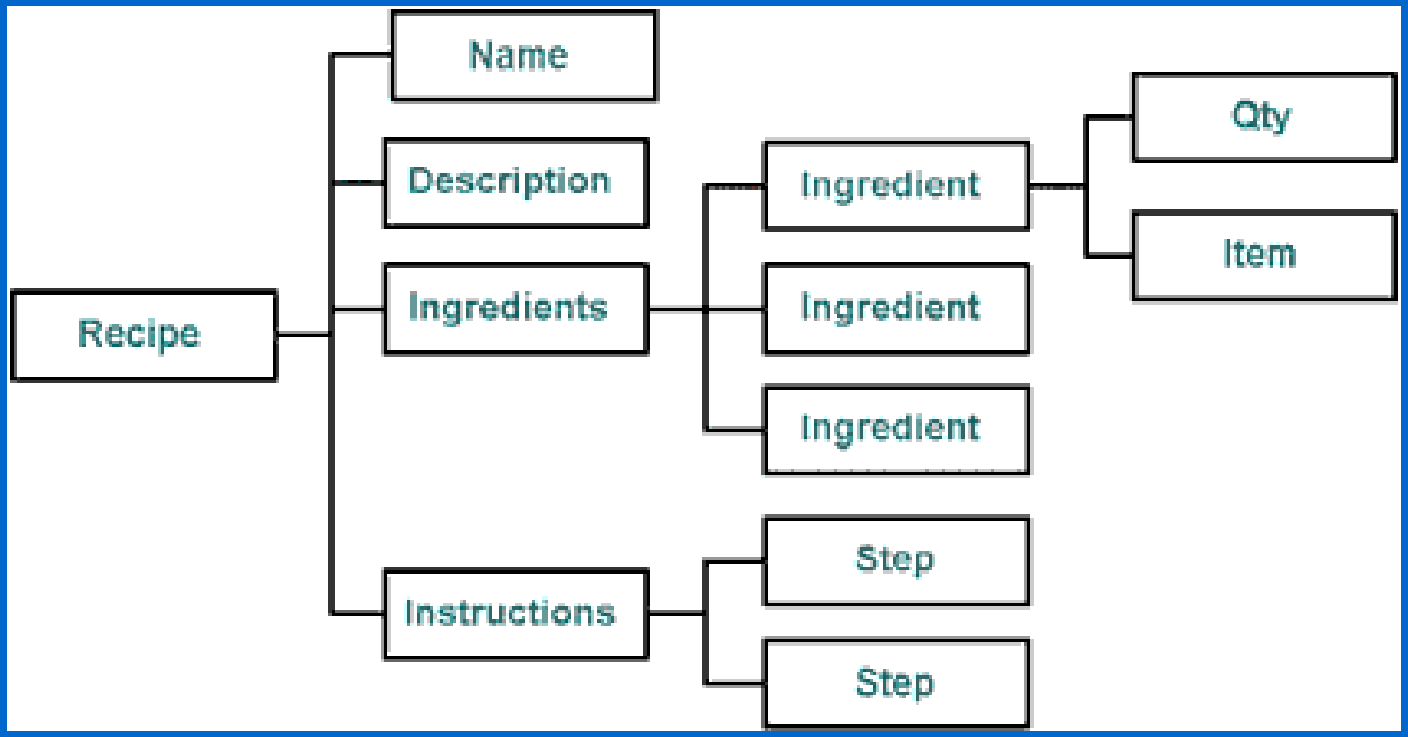

WML 2.0
XHTML Mobile Profile
XHTML Basic
XHTML 1.0
WML 1.x
HTML 4.0
for Mobile
iMode HTML
HTML 4.0
HDML
XML 1.0
HTML 3.2
Compact HTML
XML languages
XML = Extensible Markup Language
HTML = HyperText Markup Language
HDML = Handheld Device Markup Language
Openwave
WAP Forum
W3C
DoCoMo

- WML 2.0 separa contenuto e rappresentazione proprio come XML e XSL
  - WML 1.x invece li teneva insieme
- Non deve più essere trasformato in binario (usato su cellulari GPRS e >)

- I tag di WML 1.x si trovano dentro il namespace “wml:”
- I browser XHMTL di Nokia non hanno comunque bisogno del namespace “wml:” sono compatibili nativamente con WML 1.x

DOM

- Il documento XML con relativo DTD visto negli esempi ha una struttura ad albero:



DOM

- Il Document Object Model del W3C è un API astratta per l'accesso, la manipolazione e la costruzione di documenti XML e HTML.

- Consente di manipolare l'albero di oggetti del documento anziché lo stream di testo del documento XML

DOM

- DOM fornisce un metodo comune per accedere a strutture di dati da documenti strutturati.
- Qualsiasi linguaggio per il quale sia stato sviluppato DOM può manipolare, salvare, caricare strutture di oggetti usando XML

- L’API è standard:
  - es. se scrivo un programma Java che usa un certo parser DOM, il mio codice può funzionare ugualmente anche se cambio parser

- L’intero albero del documento è mantenuto in memoria: richiede molto spazio
  - Posso modificarlo facilmente, agendo sui nodi in memoria

Alcune classi usate in DOM

- Document — il nodo più alto del documento, si accede alla radice da questo
- Node — un nodo dell'albero
- NodeList — lista degli oggetti Node
- Element — Nodo elemento (un tag), deriva da Node
- Attr — Nodo attributo, deriva da Node
- CharacterData — caratteri (es. valore del tag)

- Le classi viste prima hanno diversi metodi, usati per creare elementi e attributi (Document), prendere il valore dei nodi, copiarli, rimpiazzarli e cancellarli (Node / Element)

- DOMParser parser = new DOMParser();
- parser.parse(uri);
- Document doc = parser.getDocument();

- Un **parser** analizza un **file XML**, crea un **documento DOM**, composto da **nodi**, che possono essere **elementi, testo, attributi**… sui quali agisco con **metodi** della **API DOM**
  - Il contenuto di un tag è un nodo testo

- Nell’esempio seguente un semplice file XML viene letto, il valore di un nodo è modificato, e viene aggiunto un nuovo nodo

```xml
<?xml version="1.0"?>
<!DOCTYPE myMessage [
<!ELEMENT myMessage (message*)>
<!ELEMENT message (#PCDATA)>
]>

<myMessage>
    <message>Benevenuti al DOM !</message>
</myMessage>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE myMessage>

<myMessage>

  <message>Ho cambiato il messaggio!</message>

  <message>un secondo messaggio !</message>
</myMessage>
```

- Per effettuare il parsing del file:

1. Si crea la **DocumentBuilderFactory**
2. Da cui si crea il **DocumentBuilder**: effettua il parsing vero e proprio
3. Si usa il metodo **parse** del **DocumentBuilder**, ottenendo un oggetto **Document**

- Per esaminare il documento devo prendere la radice con **getDocumentElement**

- Prendo i nodi figli mettendoli in una **NodeList** su cui posso iterare
  - Oltre ai nodi degli elementi ci sono tutti gli altri nodi (es. nodi testo)

- Con i metodi della classe Node posso muovermi su fratelli, figli e genitori:

- for (Node child = root.getFirstChild;
  child != null;
  child = child.getNextSibling())
  {
  ….

- Per esaminare tutto l'albero posso seguire un approccio ricorsivo:

```
…
stepThrough(root);
…

private void stepThrough (Node start)
 {
     // stampo tag e valore
     System.out.println(start.getNodeName()+" = "+start.getNodeValue());
     // itero su tutti i figli
     for (Node child = start.getFirstChild();
     child != null;
     child = child.getNextSibling())
     {
        stepThrough(child);
     }
 }
```

```java
// Legge il file intro.xml, cambia il testo di <message> e aggiunge un nuovo <message>

import java.io.*;
// con org.apache.crimson.tree.XmlDocument posso scrivere file XML
import org.apache.crimson.tree.XmlDocument;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import org.xml.sax.*;

public class ReplaceText {
   private Document document;

   public ReplaceText()
   {
      try {

         // crea il default parser
         DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

         // Imposta parser validante
         factory.setValidating( true );

         DocumentBuilder builder = factory.newDocumentBuilder();

         // Imposta error handler
         builder.setErrorHandler( new MyErrorHandler() );

         // Parsing del file XML
         document = builder.parse( new File( "intro.xml" ) );

         // Prende il nodo radice
         Node root = document.getDocumentElement();

         // prende il nodo radice
         if ( root.getNodeType() == Node.ELEMENT_NODE ) {
            Element myMessageNode = ( Element ) root;
            // crea lista di nodi <message>
            NodeList messageNodes = myMessageNode.getElementsByTagName( "message" );

            // se la lista non è vuota...
```

```
44              Node message = messageNodes.item( 0 );
45
46              // crea un text node
47              Text newText = document.createTextNode(
48                              "Ho cambiato il messaggio!" );
49
50              // prende il vecchio text node
51              Text oldText =
52                 ( Text ) message.getChildNodes().item( 0 );
53
54              // cambia il testo
55              message.replaceChild( newText, oldText );
56           }
57
58           // crea nuovo nodo
59           Node newMsg = createNewNode(document);
60           // appende il nodo come figlio del nodo Element marcato myMessage
61           myMessageNode.appendChild(newMsg);
62        }
63
64        // crea il nuovo file XML
65        ( (XmlDocument) document).write( new FileOutputStream( "intro1.xml" ) );
66     }
67     catch ( SAXParseException spe ) {
68        System.err.println( "Parse error: " +
69           spe.getMessage() );
70        System.exit( 1 );
71     }
72     catch ( SAXException se ) {
73        se.printStackTrace();
74     }
75     catch ( FileNotFoundException fne ) {
76        System.err.println( "File \'intro.xml\' mancante. " );
77        System.exit( 1 );
78     }
79     catch ( Exception e ) {
80        e.printStackTrace();
81     }
```

- Per aggiungere un nodo prima preparo il nodo (riga 59 e 86) poi lo aggiungo (riga 61)

```
85      // crea nodo Element con nome message
86      public Node createNewNode(Document document){
87
88          Element newMessage = document.createElement("message");
89          // crea text node e lo mette come figlio del nodo elemento <message>
90          newMessage.appendChild(document.createTextNode("un secondo messaggio !"));
91          return(newMessage);
92      }
93
94      public static void main( String args[] )
95      {
96           ReplaceText d = new ReplaceText();
97      }
98  }
```

- Per cancellare un nodo ne prendo il padre e poi su questo eseguo il metodo **removeChild**:

  Node cancellato = daCancellare.getParentNode().removeChild(daCancellare)

- Memorizzando il nodo in cancellato posso spostarlo, appendendolo da un’altra parte

SAX

- Simple API for XML è un secondo tipo di API per interpretare i documenti XML
  - Viene notificato quando avviene un certo evento durante il parsing del documento.
  - Si può decidere di elaborare solo alcuni dati
  - E' più semplice (e meno potente) di DOM

- La memorizzazione dei dati contenuti nel file XML è a carico del programmatore
  - Se dobbiamo modificare molti dati di un documento XML forse risulta più comodo DOM
- Se dobbiamo elaborare un grosso file XML, o dobbiamo fare elaborazioni veloci è meglio SAX

SAX - esempio

- Es. prendere dati da una tabella usando il SUN SAX parser (JAXP)

```
<table>
   <tr>
      <td>8</td><td>45</td>
   </tr>
   <tr>
      <td>21</td><td>56</td>
   </tr>
   <tr>
      <td>34</td><td>55</td>
   </tr>
   <tr>
      <td>47</td><td>73</td>
   </tr>
   <tr>
      <td>60</td><td>72</td>
   </tr>
   <tr>
      <td>73</td><td>34</td>
   </tr>
   <tr>
      <td>86</td><td>45</td>
   </tr>
   <tr>
      <td>99</td><td>43</td>
   </tr>
   <tr>
      <td>112</td><td>74</td>
   </tr>
</table>
```

table.xml

SAX - esempio

```
import java.util.*;
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
```

- Import per Java (JAXP) – SAX1

SAX - esempio

```
ublic class parser extends HandlerBase

   private String CurrentElement = "";
   private boolean First = false;
   private boolean Second = false;
   private ArrayList x_List = new ArrayList();
   private ArrayList y_List = new ArrayList();
   private String temp_x = "";
   private String temp_y = "";
   private WirelessBitmap wirelessbitmap;

   public void parser(){}

   public void parseFile()
   {
       First = false;
       Second = false;

       // Use the default (non-validating) parser
       SAXParserFactory factory = SAXParserFactory.newInstance();
       try {
           SAXParser saxParser = factory.newSAXParser();
           saxParser.parse( new File("..\\webapps\\examples\\xml_files\\table.xml"), this );

          } catch (Throwable t) {
            t.printStackTrace ();
          }
   }
```

SAX - esempio

- Il metodo *parse* di SAXParser definisce quale sorgente e handler usare.

- In questo caso *this* indica di usare i metodi definiti nella classe parser

```
//============================================================
// SAX DocumentHandler methods
//============================================================

public void startElement (String name, AttributeList attrs)
throws SAXException
{
    CurrentElement = name;
    if (name=="tr"){
       First = true;
       Second = false;
    }//if (name=="tr")
}// startElement

public void endElement (String name)
throws SAXException
{
    CurrentElement = "";
    if ("tr"==name){
       First=false;
       Second=false;

       x_List.add(temp_x);
       y_List.add(temp_y);
    }//if ("tr"==name)

    if (name=="td"){
       if ((First)&&(!Second)){
          First=false;
          Second=true;
          //MainMenu+="First=true and Second=False";
       }//if (First)&&(!Second)
    }//if (name=="td")
}
```

SAX - esempio

```
public void characters (char buf [], int offset, int len)
throws SAXException
{
    String s = new String(buf, offset, len);
    if (CurrentElement == "td")
       if (First && !Second)
          temp_x = s;
       else
          temp_y = s;
}//characters
```

- Nell'esempio vengono usati solo 3 eventi (startElement, endElement e characters)

SAX - esempio

All Classes

Packages

javax.xml.parsers
javax.xml.transform
javax.xml.transform.dom
javax.xml.transform.sax

javax.xml.parsers

Classes

DocumentBuilder
DocumentBuilderFactory
SAXParser
SAXParserFactory

Exceptions

ParserConfigurationExceptic

Errors

FactoryConfigurationError

characters

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

Deprecated.
Receive notification of character data inside an element.

By default, do nothing. Application writers may override this method to take specific actions for each chunk
character data (such as adding the data to a node or buffer, or printing it to a file).

Specified by:

characters in interface DocumentHandler

Parameters:

ch - The characters.
start - The start position in the character array.
length - The number of characters to use from the character array.

Throws:

SAXException - Any SAX exception, possibly wrapping another exception.

See Also:

DocumentHandler.characters(char[], int, int)

SAX - esempio

- Durante il parsing del documento XML verranno intercettati solo i tre eventi definiti prima:
  - startElement (table)
  - startElement (tr)
  - startElement (td)
  - characters
  - endElement (td)
  - …

SAX - esempio

- Nota: i dati dell'esempio precedente potevano anche essere scritti come:
  - <table>
    <entry x="8" y="45"/>
    <entry x="21" y="56"/>
    <entry x="34" y="55"/>
    <entry x="47" y="73"/>
    <entry x="60" y="72"/>
    <entry x="73" y="34"/>
    <entry x="86" y="45"/>
    <entry x="99" y="43"/>
    <entry x="112" y="74"/>
    </table>

SAX - esempio

- Nel caso precedente i dati sono i parametri dell'elemento entry.
- Il metodo getValue della classe Attributes ritorna il valore di un parametro:

- es.: *public void startElement(String uri,*
  String name,
  String qualifiedName,
  Attributes atts)

int temp_x=Integer.parseInt(atts.getValue("x"));

- Stampa dell’albero di un documento XML
  - Notare cosa cambia se si valida o meno il documento !
  - L’indentazione viene incrementata/diminuita dai metodi startElement ed endElement

```
    // main method
    public static void main( String args[] )
    {
        boolean validate = false;

        if ( args.length != 2 ) {
            System.err.println( "Usage: java SimpleSAXExample [validate] " +
                                "[filename]\n" );
            System.err.println( "Options:" );
            System.err.println( "   validate [yes|no] : " +
                                "DTD validation" );
            System.exit( 1 );
        }

        if ( args[ 0 ].equals( "yes" ) )
            validate = true;

        SAXParserFactory saxFactory =
            SAXParserFactory.newInstance();

        saxFactory.setValidating( validate );

        try {
            SAXParser saxParser = saxFactory.newSAXParser();
            saxParser.parse( new File( args[ 1 ] ), new SimpleSAXExample() );
        }
        catch ( SAXParseException spe ) {
            System.err.println( "Parse Error: " + spe.getMessage() );
        }
        catch ( SAXException se ) {
            se.printStackTrace();
        }
        catch ( ParserConfigurationException pce ) {
            pce.printStackTrace();
        }
        catch ( IOException ioe ) {
            ioe.printStackTrace();
        }

        System.exit( 0 );
    }
}
```

```
22 {
23
24     private int indent = 0;  // indentation counter
25
26     // returns the spaces needed for indenting
27     private String spacer( int count )
28     {
29         String temp = "";
30
31         for ( int i = 0; i < count; i++ )
32             temp += "  ";
33
34         return temp;
35     }
36
37     // method called before parsing
38     // it provides the document location
39     public void setDocumentLocator( Locator loc )
40     {
41         System.out.println( "URL: " + loc.getSystemId() );
42     }
43
44     // method called at the beginning of a document
45     public void startDocument() throws SAXException
46     {
47         System.out.println( "[ document root ]" );
48     }
49
50     // method called at the end of the document
51     public void endDocument() throws SAXException
52     {
53         System.out.println( "[ document end ]" );
54     }
55
56     // method called at the start tag of an element
57     public void startElement( String name,
58         AttributeList attributes ) throws SAXException
59     {
60         System.out.println( spacer( indent++ ) +
61                                         "+-[ element : " + name + " ]");
```

```
57    public void startElement( String name,
58         AttributeList attributes ) throws SAXException
59    {
60         System.out.println( spacer( indent++ ) +
61                                                  "+-[ element : " + name + " ]");
62
63         if ( attributes != null )
64
65               for ( int i = 0; i < attributes.getLength(); i++ )
66                    System.out.println( spacer( indent ) +
67                           "+-[ attribute : " + attributes.getName( i ) +
68                           " ] \"" + attributes.getValue( i ) + "\"" );
69    }
70
71    // method called at the end tag of an element
72    public void endElement( String name ) throws SAXException
73    {
74         indent--;
75    }
76
77    // method called when a processing instruction is found
78    public void processingInstruction( String target,
79         String value ) throws SAXException
80    {
81         System.out.println( spacer( indent ) +
82               "+-[ proc-inst : " + target + " ] \"" + value + "\"" );
83    }
84
85    // method called when characters are found
86    public void characters( char buffer[], int offset,
87         int length ) throws SAXException
88    {
89         if ( length > 0 ) {
90               String temp = new String( buffer, offset, length );
91
92               System.out.println( spacer( indent ) +
93                                                  "+-[ text ] \"" + temp + "\"" );
94         }
95    }
96
97    // method called when ignorable whitespace is found
```

- Finora i metodi sono più o meno quelli visti prima, nel seguito vediamo un metodo nuovo:

- Il metodo ignorableWhitespace può essere chiamato solo quando il documento XML viene validato:
  - Il parser SAX non può sapere se sono caratteri validi (es. tag misto)

```
 97     // method called when ignorable whitespace is found
 98     public void ignorableWhitespace( char buffer[],
 99         int offset, int length )
100     {
101         if ( length > 0 ) {
102                 System.out.println( spacer( indent ) + "+-[ ignorable ]" );
103         }
104     }
105
106     // method called on a non-fatal (validation) error
107     public void error( SAXParseException spe )
108         throws SAXParseException
109     {
110         // treat non-fatal errors as fatal errors
111         throw spe;
112     }
113
114     // method called on a parsing warning
115     public void warning( SAXParseException spe )
116         throws SAXParseException
117     {
118         System.err.println( "Warning: " + spe.getMessage() );
119     }
120
```

- I metodi error e warning si usano per gestire gli errori di validazione

- Nel prossimo esempio si vede come senza validazione gli spazi sono stampati sullo schermo, mentre nell'esempio successivo viene chiamato il metodo ignorableWhitespace

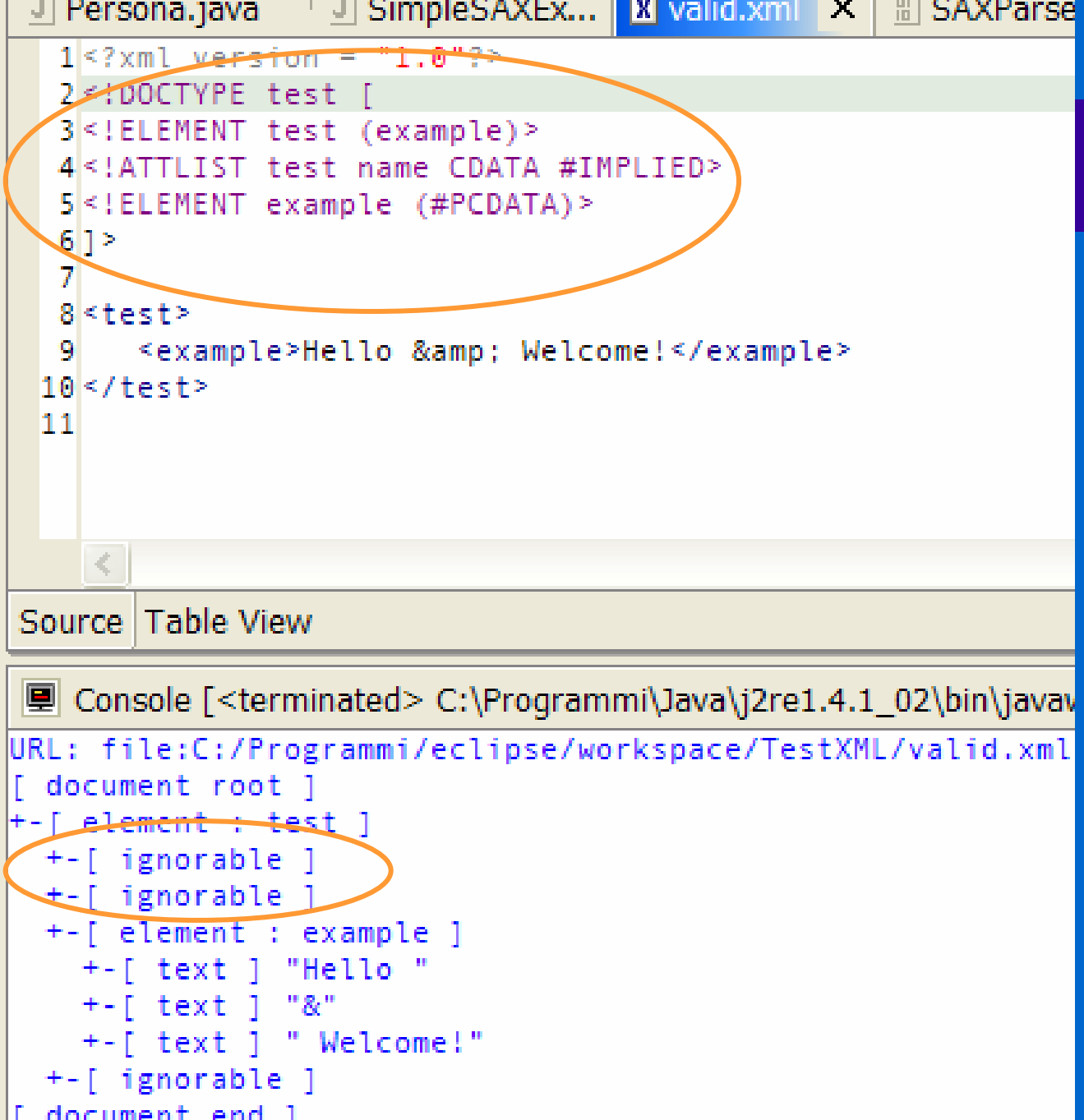
Persona.java | SimpleSAXEx... | valid.xml | SAXParser.cl

```
1 <?xml version = "1.0"?>
2 <!--DOCTYPE test [
3 <!ELEMENT test (example)>
4 <!ATTLIST test name CDATA #IMPLIED>
5 <!ELEMENT example (#PCDATA)>
6 ]-->
7
8 <test>
9    <example>Hello &amp; Welcome!</example>
10 </test>
11
```

Source | Table View

Console [<terminated> SimpleSAXExample at localhost:6474]

```
URL: file:C:/Programmi/eclipse/workspace/TestXML/valid.xml
[ document root ]
+-[ element : test ]
  +-[ text ] "
"
  +-[ text ] "    "
  +-[ element : example ]
    +-[ text ] "Hello "
    +-[ text ] "&"
    +-[ text ] " Welcome!"
  +-[ text ] "
"
[ document end ]
```


Persona.java
SimpleSAXEx...
valid.xml
SAXParse
1 <?xml version = "1.0"?>
2 <!DOCTYPE test [
3 <!ELEMENT test (example)>
4 <!ATTLIST test name CDATA #IMPLIED>
5 <!ELEMENT example (#PCDATA)>
6]>
7
8 <test>
9 <example>Hello & Welcome!</example>
10 </test>
11
Source
Table View
Console [<terminated> C:\Programmi\Java\j2re1.4.1_02\bin\javav
URL: file:C:/Programmi/eclipse/workspace/TestXML/valid.xml
[document root]
+-[element : test]
+-[ignorable]
+-[ignorable]
+-[element : example]
+-[text] "Hello "
+-[text] "&"
+-[text] " Welcome!"
+-[ignorable]
[document end]

- In generale la chiamata ad endElement è un buon momento per processare il contenuto di un elemento
  - Come nel primo esempio: i dati venivano memorizzati nel vettore dei dati
- NOTA: il metodo charachters NON sa a quale tag appartiene il buffer
  - Ce lo dobbiamo ricordare noi: vedi primo esempio

SAX 2.0

- In SAX 2.0 è stato introdotto il supporto per i namespace
  - I metodi principali visti prima rimangono li stessi, con le dovute modifiche per usare i namespace
  - La classe base è DefaultHandler

- Nel seguente esempio i dati di un file XML vengono letti e memorizzati in un oggetto della classe Properties

properties.xml

```
1  <?xml version="1.0"?>
2
3  <PropertySet>
4      <property name="height">5</property>
5      <property name="width">9</property>
6      <property name="color">blu</property>
7      <property name="depth">12</property>
8  </PropertySet>
9
```

```
24{
25    String thisProp = "";
26    String thisElement = "";
27    String thisPropValue = "";
28    Properties propSet;
29
30
31    public static void main(String[] args)
32    {
33
34        boolean validate = false;
35
36         if ( args.length != 2 ) {
37                System.err.println( "Uso: java SAXE2xample [validate] " +
38                                                        "[filename]\n" );
39                System.err.println( "Opzioni:" );
40                System.err.println( "   validate = [si|no] : " +
41                                                        "DTD validation" );
42                System.exit( 1 );
43         }
44
45         if ( args[ 0 ].equals( "si" ) )
46                validate = true;
47
48         SAXParserFactory saxFactory =
49                SAXParserFactory.newInstance();
50
51         saxFactory.setValidating( validate );
52
53        try
54        {
55            SAXParser saxParser = saxFactory.newSAXParser();
56            saxParser.parse( new File( args[ 1 ] ), new AnotherSAXExample() );
57        } catch (ParserConfigurationException e)
58        {
59            e.printStackTrace();
60            System.out.println(e.getLocalizedMessage());
61        } catch (SAXException e)
62        {
63            e.printStackTrace();
```

- Le stringhe:
  - thisElement
  - thisProp
  - thisPropValue

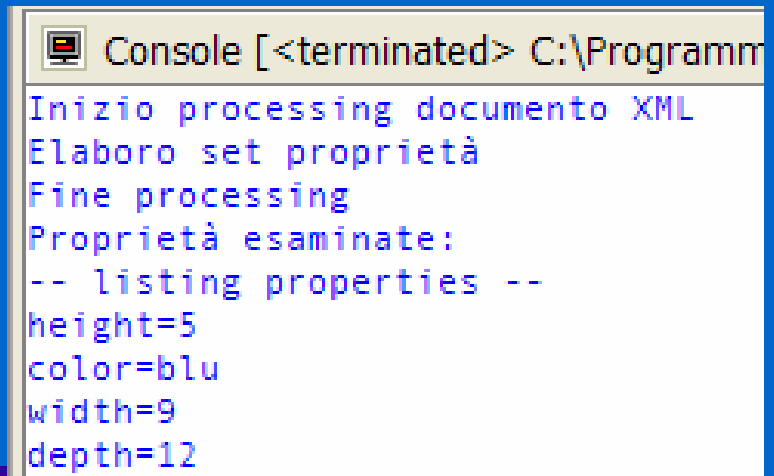
sono usate per leggere: il nome del tag, il valore dell’attributo ed il valore del tag

```java
public void characters(char[] ch, int offset, int length) throws SAXException
{
    if ( thisElement == "property")
    {
        thisPropValue = new String( ch, offset, length);
    }
}

/* (non-Javadoc)
 * @see org.xml.sax.ContentHandler#endDocument()
 */
public void endDocument() throws SAXException
{
    System.out.println("Fine processing");
    System.out.println("Proprietà esaminate: ");
    propSet.list( System.out );
}

/* (non-Javadoc)
 * @see org.xml.sax.ContentHandler#endElement(java.lang.String, java.lang.String, java.lang.String)
 */
public void endElement(String arg0, String arg1, String arg2)
    throws SAXException
{
    propSet.setProperty(thisProp, thisPropValue);
    thisProp = "";
    thisElement = "";
    thisPropValue = "";
}

/* (non-Javadoc)
 * @see org.xml.sax.ContentHandler#startDocument()
 */
public void startDocument() throws SAXException
{
    System.out.println("Inizio processing documento XML");
    propSet = new Properties();
}
```

- Nel metodo characters viene letto il valore del tag, nel caso questo sia del tipo voluto
- Il dato viene effettivamente memorizzato al momento della chiamata di endElement
  - È l’approccio migliore

```
116      * @see org.xml.sax.ContentHandler#startElement(java.lang.String, java.lang.String, jav
117      */
118     public void startElement(
119         String nsUri,
120         String localName,
121         String qName,
122         Attributes atts)
123         throws SAXException
124     {
125         if (qName == "PropertySet")
126             System.out.println("Elaboro set proprietà");
127         else if ( qName == "property")
128         {
129             thisProp = atts.getValue("name");
130         }
131 
132         thisElement = qName;
133 
134     }
135 
136 
137     /* (non-Javadoc)
138      * @see org.xml.sax.ErrorHandler#error(org.xml.sax.SAXParseException)
139      */
140     public void error(SAXParseException arg0) throws SAXException
141     {
142         System.out.println("SAX Error: " + arg0.getLocalizedMessage());
143     }
144 
145     /* (non-Javadoc)
146      * @see org.xml.sax.ErrorHandler#fatalError(org.xml.sax.SAXParseException)
147      */
148     public void fatalError(SAXParseException arg0) throws SAXException
149     {
150         System.out.println("SAX Fatal error: " + arg0.getLocalizedMessage());
151     }
152 
153 }
```

- In **startElement** leggo l'attributo e memorizzo il nome del tag letto, così da poter associare il valore del tag letto da **characters** al momento in cui viene chiamato **endElement**



Hyperlinks

- Anche i link ipertestuali sono estendibili
  - Il link HTML: <A HREF="qualcosa">xx</A> porta da una sola parte
- In XML è possibile aggiungere più punti di arrivo

Hyperlinks

- Un link si può aprire come un menu di link (link multiplo)
- I link possono essere bidirezionali (non serve più il pulsante Back)

- Gli anchor sono potenziati: è possibile far inserire del testo nella pagina visitata, oppure collegarsi ad una sezione intera di testo

- Ci si può collegare ad un’entità del documento XML, senza bisogno di specificare l’ancoraggio nel documento