



Interazione Uomo-Macchina e tecnologie web

Prof.ssa Liliana Ardissono
Dipartimento di Informatica
Università di Torino

XML – basi del linguaggio



Questa presentazione è distribuita sotto licenza Creative Commons CC BY ND



HTML

HTML è un markup language: HTML usa i tag per aggiungere informazioni al contenuto di un documento

- MA: **non si possono aggiungere nuovi tag** (es: per descrivere contenuti di tipo specifico)
- ⇒ serve un linguaggio di markup estensibile per definire il proprio linguaggio di descrizione dei dati



XML - Extensible Markup Language - I

- **linguaggio di markup testuale**
- **standard W3C per rappresentare documenti e dati, e per lo scambio di dati nel Web**
- alcuni **obiettivi di design** di XML:
 - Essere direttamente utilizzabile su internet, supportare un'ampia gamma di applicazioni, essere compatibile con SGML, essere human-readable, essere facilmente processabile da programmi,

XML - Extensible Markup Language - II



- linguaggio mediante cui si possono **definire altri linguaggi**
 - es: XHTML, SMIL (Synchronized Multimedia Integration Language), WSDL (Web Services Definition Language), ...
- **estensibile**
 - Permette la **creazione di nuovi tag**
- **strutturato**
 - Mediante i tag si può rappresentare la struttura delle informazioni
- **Validante (lo vedremo poi)**
 - permette di definire **regole grammaticali** per verificare la correttezza sintattica delle informazioni rappresentate in XML (**DTD e XML schema**)

XML - Extensible Markup Language - III



XML si è imposto come standard per la condivisione di dati su internet

XML è basato sulla rappresentazione dei dati come stringhe di caratteri ⇒ facilmente trasferibili via HTTP

Essendo uno standard *de facto*, esistono strumenti per scrittura, interpretazione, gestione di documenti XML

→ Noi non dobbiamo scrivere gli analizzatori sintattici, i traduttori e gli strumenti di processing, per verificare la correttezza dei documenti XML

Documentazione su XML:

<http://www.w3schools.com/xml/>

(Specifiche di XML: www.w3.org/TR/xml/)



Esempio - I

Consideriamo la descrizione di una pizza, che contiene dati di vario genere:

peperoni
8.00
large

La descrizione non è strutturata: per estrarre i dati bisogna assumere che essi rispettino un pattern implicito nei dati stessi (nome pizza, seguito da prezzo, seguito da dimensione)

- *E, in un'applicazione java, leggere ciclicamente, per es. con Scanner, i dati, interpretandoli secondo il tipo atteso*

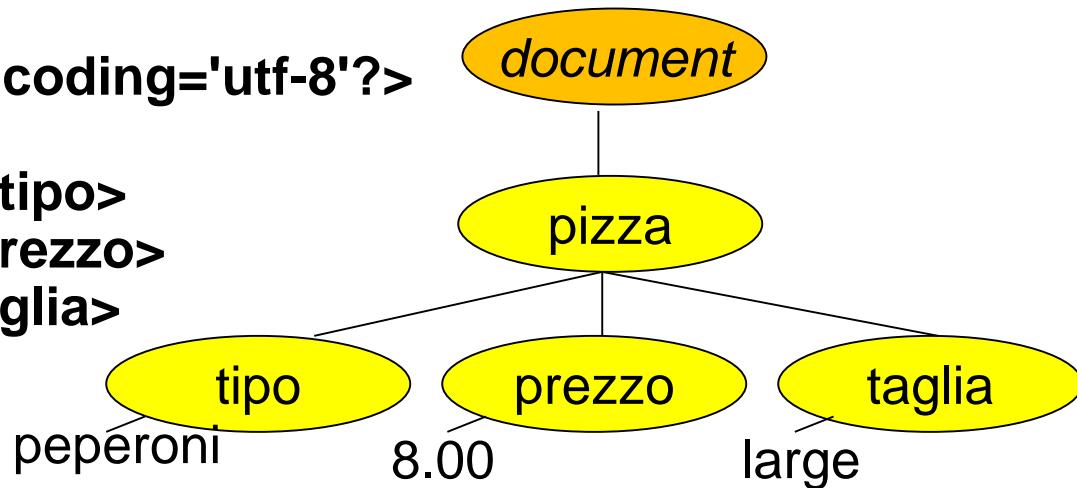


Esempio - II

Noi possiamo **strutturare** la descrizione **introducendo dei tag specifici** per descrivere il contenuto di una pizza **<pizza>** e gli elementi descrittivi **<tipo>**, **<prezzo>**, **<taglia>**

→ la struttura della pizza può essere rappresentata in modo gerarchico attraverso l'annidamento di tag: ciascun tag, all'interno del tag principale **<pizza>**, mantiene un particolare tipo di informazione relativa alla pizza:

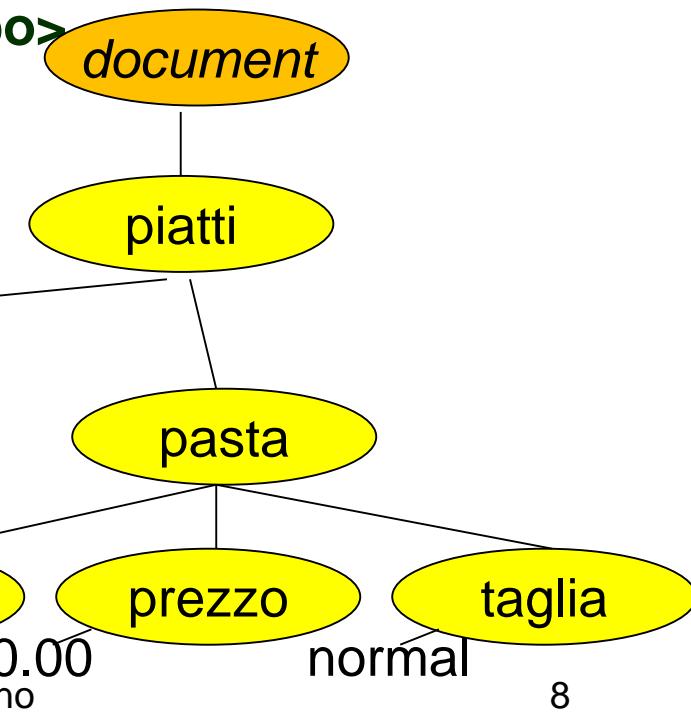
```
<?xml version='1.0' encoding='utf-8'?>
<pizza>
    <tipo> peperoni </tipo>
    <prezzo> 8.00 </prezzo>
    <taglia> large </taglia>
</pizza>
```



Esempio - III



```
<?xml version='1.0' encoding='utf-8'?>
<piatti>
    <pizza>
        <tipo> margherita </tipo>
        <prezzo> 8.00 </prezzo>
        <taglia> media </taglia>
    </pizza>
    <pasta>
        <tipo> spaghetti al pesto </tipo>
        <prezzo> 10.00 </prezzo>
        <taglia> normal </taglia>
    </pasta>
</piatti>
```



tipo
margherita 8.00

taglia
media

tipo
spaghetti al pesto 10.00

taglia
normal



Documenti XML

- Sono composti di entità (unità fisiche di memorizzazione) che contengono dati analizzabili sintatticamente o meno (codifica UNICODE dei caratteri)
- **PCDATA** (*Parsed Character Data*) sono i dati analizzabili sintatticamente. I PCDATA contengono caratteri
 - dati di tipo carattere
 - simboli di markup (tag di apertura e chiusura, commenti, ...; questi simboli descrivono la struttura e le proprietà dei documenti)
- **CDATA** (*unparsed data*) sono i dati non analizzabili sintatticamente. Essi possono contenere testo, codice di scripting (e.g., Javascript) o altri tipi di dati. Il contenuto di CDATA non è interpretato secondo la sintassi di XML



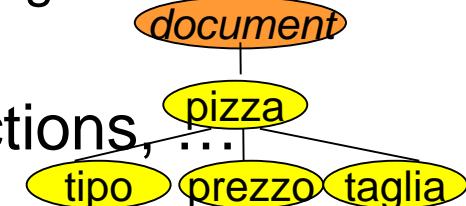
CDATA e PCDATA

```
<?xml version='1.0' encoding='utf-8'?>
<!-- commento in XML (come HTML) ... -->
<priceList>
    <coffee>
        <name> Mocha Java </name>
        <price> 11.95 </price>
    </coffee>
    <coffee>
        <name> Sumatra </name>
        <price> 12.50 </price>
    </coffee>
    <!CDATA[
        // qui io posso inserire per esempio del codice di
        programmazione
    ]]>
</priceList>
```

Documenti XML ben formati (cioè, sintatticamente corretti secondo XML)



- **Documento ::= prologo elemento Misc***
 - Il **prologo** specifica la versione di XML e la codifica dei caratteri. Es:
 - <?xml version='1.0' encoding='utf-8'?>
 - L'**elemento** corrisponde a una componente logica del documento XML e viene rappresentato mediante tag
 - **elemento: EmptyElementTag / StartTag content EndTag**
 - ogni documento ha **un solo elemento principale** (la sua radice), non incluso in alcun altro elemento XML
 - un elemento può essere composto da altri elementi (è un *contenitore*). Un elemento ha una struttura gerarchica (albero con radice *document*)
 - **Misc** sono i commenti, processing instructions,





Documenti XML ben formati

Un documento XML ben formato **deve** contenere **1 e 1 solo prologo** e **1 e 1 solo elemento radice**

Intuitivamente, un documento XML è ben formato sse

- contiene tutte le componenti necessarie (1 prologo, 1 solo elemento radice, ...)
- i suoi tag sono bilanciati. E.g.
 - Per ogni tag aperto esiste il tag chiuso (ad eccezione di tag vuoti, che possono essere chiusi direttamente:
< t > </ t > è equivalente a < / t >)
 - i tag sono annidati in modo che l'ultimo tag aperto sia il primo chiuso: *< t1 > < t2 > < t3 > < / t3 > < / t2 > < / t1 >*
 - ...

NB: XML è **case-sensitive** → $\langle t1 \rangle \neq \langle T1 \rangle$



Es: Documento XML ben formato e non

```
<?xml version='1.0' encoding='utf-8'?>
<!-- commento... -->
<priceList>
  <coffee>
    <name> Mocha Java </name>
    <price> 11.95 </price>
  </coffee>
  <coffee>
    <name> Sumatra </name>
    <price> 12.50 </price>
  </coffee>
</priceList>
```

```
<?xml version='1.0' encoding='utf-8'?>
<commento... --> tag sbagliato
<priceList>
  </coffee> mancano i tag aperto
  <name> Mocha Java <-- " " chiuso
  <price> 11.95 </price>
  </coffee>
  <coffee>
    <names> Sumatra </name> tag ≠
    <price> 12.50 </price>
  </coffee>
</priceList>
< priceList> ... </priceList> 2 radici
```



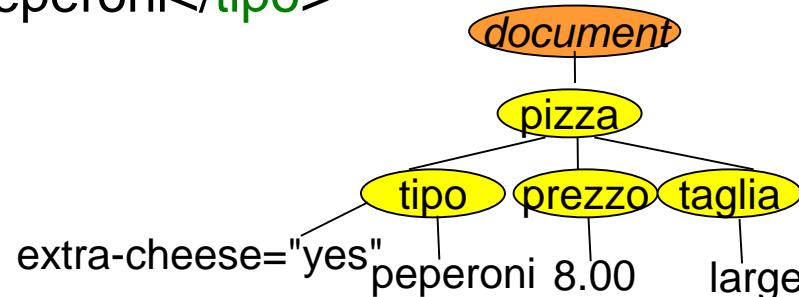
Elementi e attributi

Come già visto per HTML, un elemento può avere degli attributi, racchiusi nel tag di apertura:

- <element attr="value">

Es: specifichiamo un attributo di topping delle pizze (extracheese)

```
<pizza>
  <tipo extracheese="yes">peperoni</tipo>
  <prezzo> 8.00 </prezzo>
  <taglia> large </taglia>
</pizza>
```



XML – Esempi di uso del linguaggio



- Rappresentazione interna di dati e documenti
- Visualizzazione di dati in modalità diverse (usando CSS, come per HTML. NB: HTML è un dialetto di XML)
- Condivisione di dati tra applicazioni
 - Formato comune di rappresentazione dei dati
 - Eventuale trasformazione dei dati per tradurre dal formato di un'applicazione al formato di un'altra. Questa trasformazione è basata su tecnologie standard (XSLT – XML Transformations, che io non presento nel corso per limitazioni di tempo)



Fogli Stile – CSS

Come per HTML è possibile definire regole di presentazione del contenuto degli elementi XML

**Cascading Style Sheets (documentazione:
www.w3schools.com/css/default.asp)**

- I CSS specificano le *regole di visualizzazione* degli elementi di un documento XML
- A parità di contenuto, i dati possono essere visualizzati in modo diverso applicando fogli stile diversi
- Quando il browser utente carica un documento, esso applica le regole di visualizzazione al contenuto del documento in cascata

I lucidi su CSS per HTML descrivono la sintassi di CSS



CSS – Esempio – documento XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="stile.css"?>
<studenti>
  <studente genere="M">
    <nome>Mario</nome>
    <cognome>Mario</cognome>
    <matricola>101111</matricola>
    <recapito>
      <via>via Risorgimento</via> <cap>33100</cap>
      <comune>Torino</comune> <provincia>TO</provincia>
    </recapito>
  </studente>
  <studente genere="F">
    <nome>Luisa</nome> <cognome>Bianchi</cognome>
    <matricola>2022222</matricola>
    <recapito>
      <via>via Roma</via> <cap>10100</cap>
      <comune>Torino</comune> <provincia>TO</provincia>
    </recapito>
  </studente>
</studenti>
```

Processing Instruction:
Il tag contiene il link al foglio stile
da applicare per visualizzare
il documento XML su browser

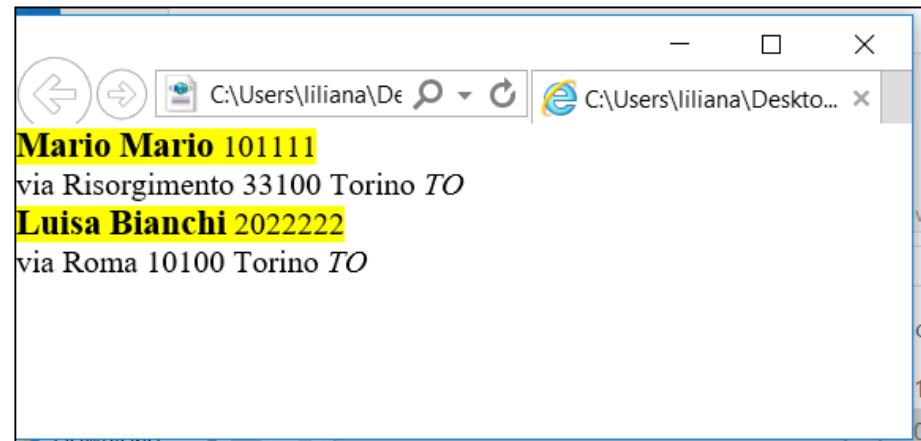
CSS – Esempio1 – foglio stile (stile.css)



```
studente {  
    background-color:  
    yellow;  
}  
  
nome {  
    font-size: larger;  
    font-weight: bold;  
}  
  
cognome {  
    font-size: larger;  
    font-weight: bold;  
}  
  
recapito {  
    display: block;  
}  
  
provincia {  
    font-style: italic;  
}
```

stile.css

Regola di presentazione
del tag “studente”
NB: si applicano gli stili
già studiati per HTML ma
i selettori delle regole fanno
riferimento ai tag XML



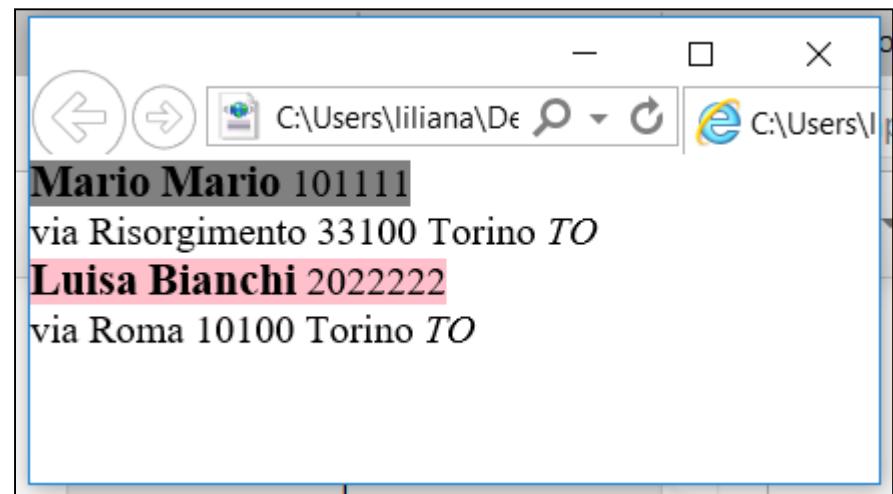


CSS – Esempio2

```
studente[genere="F"] {  
    background-color: pink  
}  
studente[genere="M"] {  
    background-color: grey  
}  
studente[genere="ND"] {  
    background-color: green  
}  
  
nome {  
    font-size: larger;  
    font-weight: bold;  
}  
  
cognome {  
    font-size: larger;  
    font-weight: bold;  
}  
  
recapito {  
    display: block;  
}  
  
provincia {  
    font-style: italic;  
}
```

Esempio di uso di selettore E[attr=val].

Il selettore filtra l'applicazione della regola sugli attributi



DOM: Rappresentazione a oggetti di documenti XML



W3C ha definito le specifiche per la rappresentazione strutturata di documenti XML: queste sono il modello DOM (**Document Object Model**)

- **DOM è lo standard W3C per la gestione di documenti XML**
- DOM rappresenta i documenti XML con un modello a oggetti (albero generico, simile a quelli disegnati per pizza e piatti nei lucidi precedenti)
- DOM facilita la manipolazione di documenti XML (estrazione di dati, modifica dei documenti). I programmi software manipolano i documenti XML trattandoli come alberi
- **Ogni documento XML (o HTML) ben formato ha una rappresentazione DOM** (quelli non ben formati non hanno il DOM)

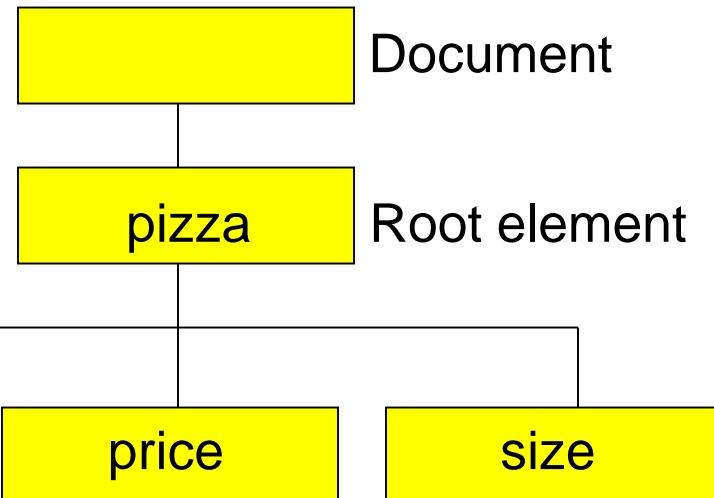


DOM - esempio

La rappresentazione DOM si basa su **struttura ad albero**:

- **elementi, attributi, etc.** sono rappresentati come **nodi di un albero**. L’albero ha **radice di tipo Document**.

```
<?xml version='1.0' encoding='utf-8'?>
<pizza>
  <topping> Peperoni </topping>
  <price> 7.00 </price>
  <size> large </size>
</pizza>
```



DOM - Document Object Model



DOM offre API standard per gestire i documenti XML. Questi API sono basati sulla definizione di interfacce

→ i produttori di software offrono implementazioni del modello DOM specifiche per i vari linguaggi (Java, PHP, etc.)

Documentazione su DOM: www.w3.org/DOM/

NB: modello DOM è indipendente dal linguaggio di programmazione che si usa per gestire i documenti XML → permette di trattare I documenti XML in modo uniforme, al di là dei dettagli sintattici del linguaggio di programmazione usato

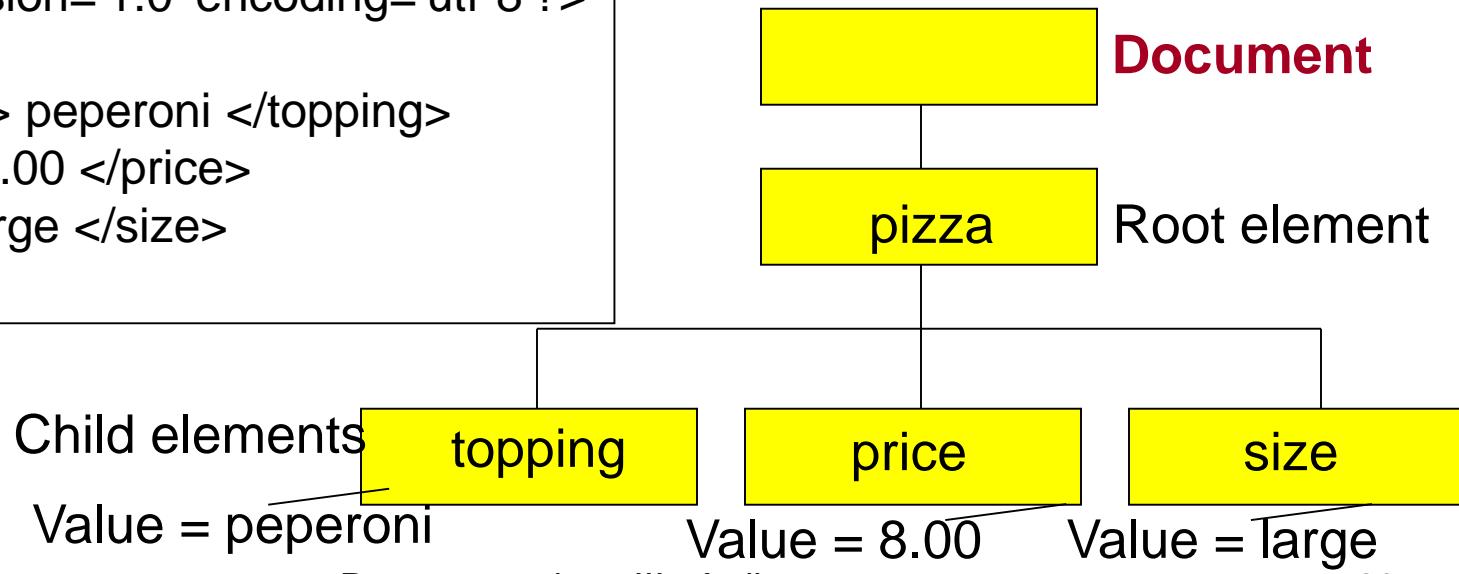


DOM - struttura

La rappresentazione DOM si basa su **struttura ad albero**:

- il **nodo principale** dell'albero ha tipo **Document** (Document rappresenta l'intero documento XML)
 - **elementi, attributi, etc.** del documento XML sono rappresentati come **nodi dell'albero**

```
<?xml version='1.0' encoding='utf-8'?>
<pizza>
  <topping> peperoni </topping>
  <price> 8.00 </price>
  <size> large </size>
</pizza>
```



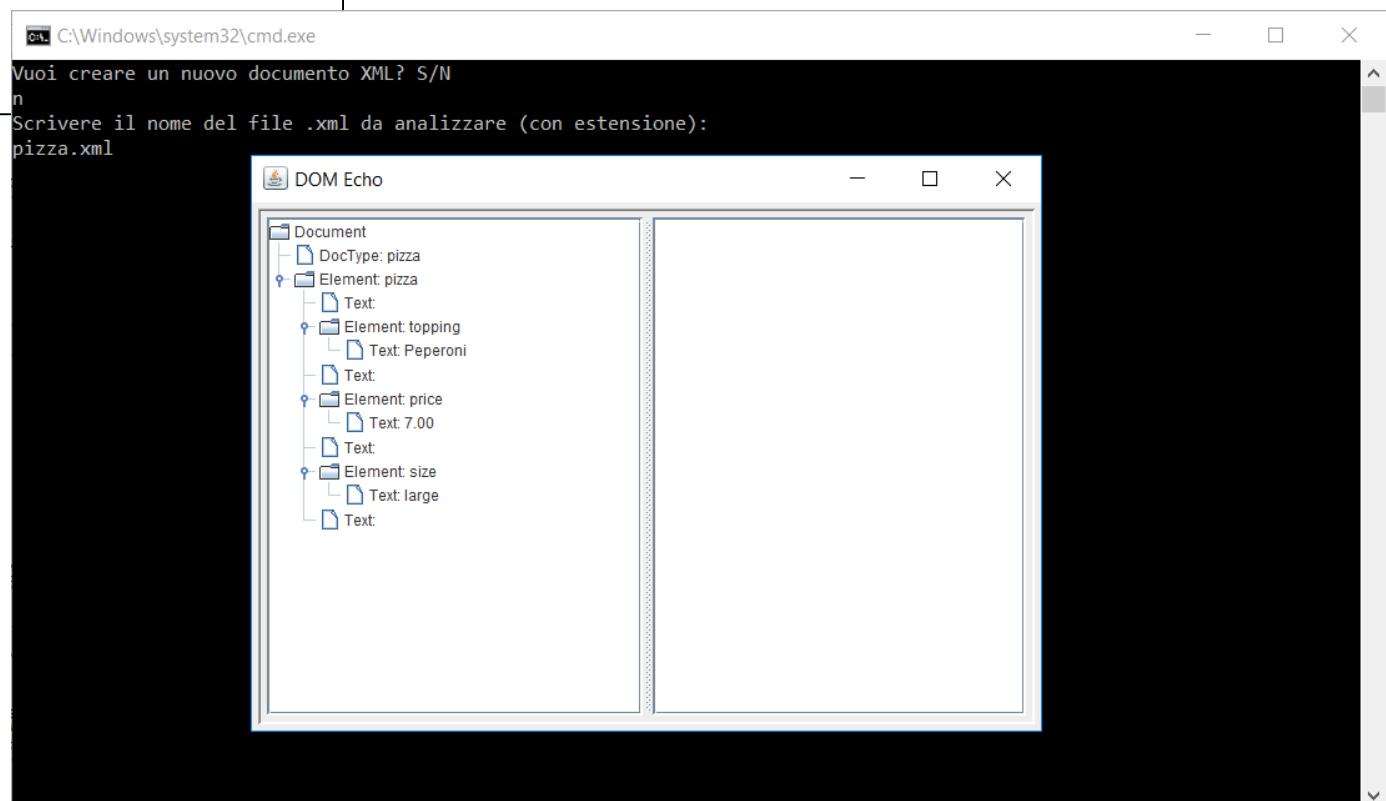


DOM - esempio

Visualizzazione di un pizza.xml utilizzando un'applicazione java

```
<?xml version='1.0' encoding='utf-8'?>
<pizza>
    <topping>Peperoni </topping>
    <price> 7.00 </price>
    <size> large </size>
</pizza>
```

applicazione **DomManipulate**





coffee.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!-- comment... -->
<!DOCTYPE priceList [
    <!ELEMENT priceList (coffee)+>
    <!ELEMENT coffee (name, price)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT price (#PCDATA)>
]>
<priceList>
    <coffee>
        <name> Mocha Java </name>
        <price> 11.95 </price>
    </coffee>
    <coffee>
        <name> Sumatra </name>
        <price> 12.50 </price>
    </coffee>
</priceList>
```



DOM – API



- DOM usa **interface Node** (e tipi + specifici di nodo, come **Element**), per rappresentare i nodi dell'albero di un documento XML
- Un albero DOM è un **albero generico** (ogni nodo può avere un numero variabile di figli)
- Le API DOM offrono i metodi per
 - creare l'albero DOM che rappresenta il contenuto di un documento XML (o per creare un documento nuovo)
 - prelevare il nodo radice dell'albero
 - trovare nodi i figli di un nodo
 - aggiungere/rimuovere figli
 - leggere/modificare il valore contenuto in un nodo, ...
- Un programma software può ispezionare e modificare un documento XML navigando l'albero a partire dal nodo Document e visitando i nodi figli



Gestione di documenti XML - JAXP

Java offre la libreria **JAXP (JAvA XML Processing)** per gestire documenti XML (analisi sintattica, estrazione di dati, etc.)

JAXP implementa le specifiche DOM e offre i parser per DTD e per XMLSchema (queste sono grammatiche per XML)

Il parser JAXP opera come segue:

1. Il parser JAXP verifica che il documento in input (D) sia ben formato
 - Se D lo è, il parser costruisce la sua rappresentazione DOM
 - Altrimenti il parser segnala i problemi sintattici lanciando eccezioni
2. Se D è *ben formato ed è anche associato a una grammatica G , un parser validante verifica anche se D è valido o meno rispetto a G . Se il documento non è valido, il parser segnala i problemi lanciando eccezioni. Queste eccezioni riguardano solo la validità di D rispetto a G , non il fatto che D sia ben formato o meno*

Per verificare se documento XML è ben formato



1. Prendere riferimento a Factory per la generazione di parser XML:

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

2. Scegliere il tipo di parser da usare

```
f.setValidating(false);           // false: parser non validante – controlla  
                                // solo ben formatezza dei documenti XML
```

3. Tentare l'analisi sintattica del documento (rispetto a XML)

```
try {
```

```
    // creare il parser
```

```
    DocumentBuilder builder = f.newDocumentBuilder();
```

// invocare il metodo di analisi sintattica sul parser passando il documento come parametro. **Il parser crea il DOM del documento**

```
    Document document = builder.parse(fileName);
```

```
} catch (SAXParseException spe) {... Gestione eccezioni di parsing ...}
```

ES: Verifica se un documento XML è ben formato



```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
...
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;

public class DomEcho01{
    public static void main(String argv[]) {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setValidating(false);
        try {DocumentBuilder builder = factory.newDocumentBuilder();
            Document document = builder.parse( "coffee.xml" );
        } catch (SAXParseException spe) {System.out.println("\n** Parsing error");
                                         ..... Gestione di altre eccezioni... }
    }
}
```



DomManipulate: coffee.xml

coffee.xml è ben formato → no eccezioni, DOM

The screenshot shows a Windows command prompt window titled 'cmd.exe' with the path 'C:\Windows\system32\cmd.exe'. It displays the following text:

```
Vuoi creare un nuovo documento XML? S/N  
n  
Scrivere il nome del file .xml da analizzare (con estensione):  
coffee.xml
```

Below the command prompt is a window titled 'DOM Echo'. This window contains a tree view of the XML document structure. The structure is as follows:

- Document
 - Comment: comment...
 - DocType: priceList
 - Element: priceList
 - Text:
 - Element: coffee
 - Text:
 - Element: name
 - Text: Mocha Java
 - Text:
 - Element: price
 - Text: 11.95
 - Text:
 - Text:
 - Element: coffee
 - Text:
 - Element: name
 - Text:
 - Text:
 - Element: price
 - Text:
 - Text:



DomManipulate: pizzaWrong.xml

pizzaWrong.xml non è ben formato → eccezioni, niente DOM

```
C:\Windows\system32\cmd.exe
Impl.scanEndElement(XMLDocumentFragmentScannerImpl.java:1750)
    at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScanner
Impl$FragmentContentDriver.next(XMLDocumentFragmentScannerImpl.java:2970)
    at com.sun.org.apache.xerces.internal.impl.XMLDocumentScannerImpl.nex
t(XMLDocumentScannerImpl.java:606)
    at com.sun.org.apache.xerces.internal.impl.XMLDocumentFragmentScanner
Impl.scanDocument(XMLDocumentFragmentScannerImpl.java:510)
    at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.pars
e(XML11Configuration.java:848)
    at com.sun.org.apache.xerces.internal.parsers.XML11Configuration.pars
e(XML11Configuration.java:777)
    at com.sun.org.apache.xerces.internal.parsers.XMLParser.parse(XMLPars
er.java:141)
    at com.sun.org.apache.xerces.internal.parsers.DOMParser.parse(DOMPars
er.java:243)
    at com.sun.org.apache.xerces.internal.jaxp.DocumentBuilderImpl.parse(
DocumentBuilderImpl.java:339)
    at javax.xml.parsers.DocumentBuilder.parse(DocumentBuilder.java:177)
    at DomManipulate.main(DomManipulate.java:186)
Premere un tasto per continuare . . .
```



Creazione di oggetto DOM che rappresenta un nuovo documento XML

```
public static void buildDom() {  
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
    try {  
        DocumentBuilder builder = factory.newDocumentBuilder();  
        Document document = builder.newDocument(); // creo nuovo DOM  
        // creo il nodo principale del DOM  
        Element root = (Element) document.createElement("rootElement");  
        // lo assegno come figlio al nodo Document  
        document.appendChild(root);  
        // aggiungo 3 nodi foglia con valori  
        root.appendChild( document.createTextNode("Some") );  
        root.appendChild( document.createTextNode(" ") )  
        root.appendChild( document.createTextNode("text") );  
    } catch (ParserConfigurationException pce) {  
        // Parser with specified options can't be built  
        pce.printStackTrace();  
    }  
}
```



Visita di oggetto DOM - I

```
public static void visitDocument(Document document, String feature) {  
    if (document!=null) {  
        Element el = document.getDocumentElement(); // prende il nodo Document  
        visit(el, feature);  
    }  
}  
  
public static void visit(Node el, String feature) {  
    if (el!=null) {  
        NodeList children = el.getChildNodes();  
        for (int i=0; i<children.getLength(); i++) {  
            Node n=children.item(i);  
            getFeatureVal(n, feature);  
            visit(children.item(i), feature);  
        }  
    }  
}
```

Feature: nome del tag
(element) di cui si
vuole conoscere il valore



Visita di oggetto DOM - II

```
public static void getFeatureVal(Node domNode, String feature) {  
    String out = "";  
    if (domNode!=null) {  
        // leggo il nome del nodo (tag)  
        String name = domNode.getNodeName();  
        if (name.equals(feature)) {  
            // cast a nodo Element per poter prendere il text node  
            Element el = (Element)domNode;  
            // prendo il nodo Text che contiene il valore del tag  
            // e leggo il valore  
            out = el.getFirstChild().getNodeValue();  
            System.out.println("The value of feature "+feature+" is: " +out);}  
    }}  
}
```



HTML e DOM

HTML un dialetto di XML

→ ogni documento HTML ben formato ha un suo albero DOM con radice DOCUMENT. In questa figura, **la radice DOCUMENT non è mostrata**:

