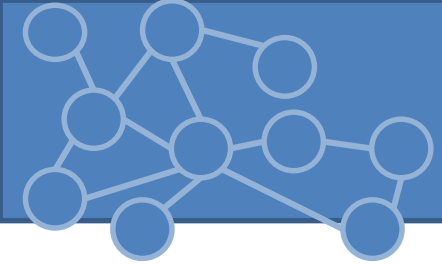


Laboratorio Reti di Calcolatori

Laurea Triennale in Comunicazione Digitale

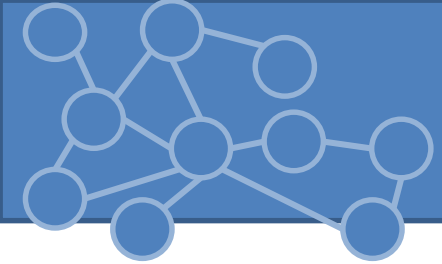
Anno Accademico 2013/2014

XML

- XML = eXtensible Markup Language
  - Progettato per transporto e memorizzazione dati, non per visualizzazione dato. Non esegue alcuna operazione ma creato per strutturare e memorizzare le informazioni all'interno di tag. Uno degli strumenti più usati per il trasporto dell'informazione.

```
<?xml version="1.0"?>
<note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

  - Linguaggio di markup con tag non predefiniti ma definibili dal programmatore. Autore del documento può inventare i propri tag. Solitamente nomi tag devono essere autoesplicativi del contenuto.
  - Raccomandato da W3C
  - Autodescrittivo e con regole sintattiche stringenti
  - Indipendente da hardware e software (text-based)
- Non rimpiazza HTML perchè obiettivi diversi:
  - XML si focalizza sul dato
  - HTML progettato per visualizzare il dato



Uso di XML

- Separa i dati da HTML
  - Se il dato cambia (pagina HTML dinamica) non devo cambiare il documento HTML ma il file di dati XML associato. HTML/CSS per visualizzazione e layout e XML per contenuto da visualizzare. Necessito di un linguaggio lato client per estrarre il dato da XML (Javascript).
- Semplifica condivisione di dati
  - Formato dei dati in database non sempre compatibili. Dati XML memorizzati come file di testo. Quasi totalmente indipendente da software.
- Semplifica trasporto dati e il cambio di piattaforma
- Rende il dato più disponibile
  - Applicazioni diverse possono accedere allo stesso dato, basta che io legga un file testuale.
- Utilizzato per definire linguaggi di markup specializzati

Albero XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

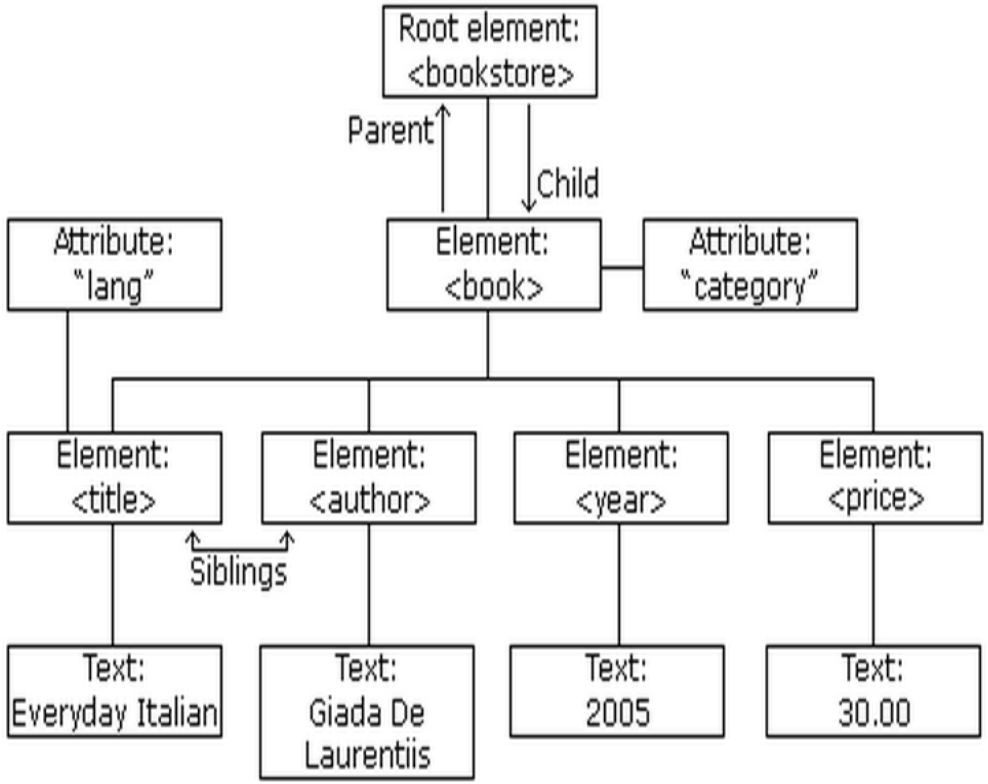
- Prima linea: dichiarazione XML con versione e codifica utilizzata.
- <note> è la radice
- <note> ha 4 figli
- </note> chiude la radice
- Elementi in un file XML formano un albero del documento. L'albero inizia dalla radice e si ramifica nei sottolivelli. Ogni elemento può avere dei sotto elementi

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

- Ogni elemento può avere degli attributi e del contenuto testuale

Esempio

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

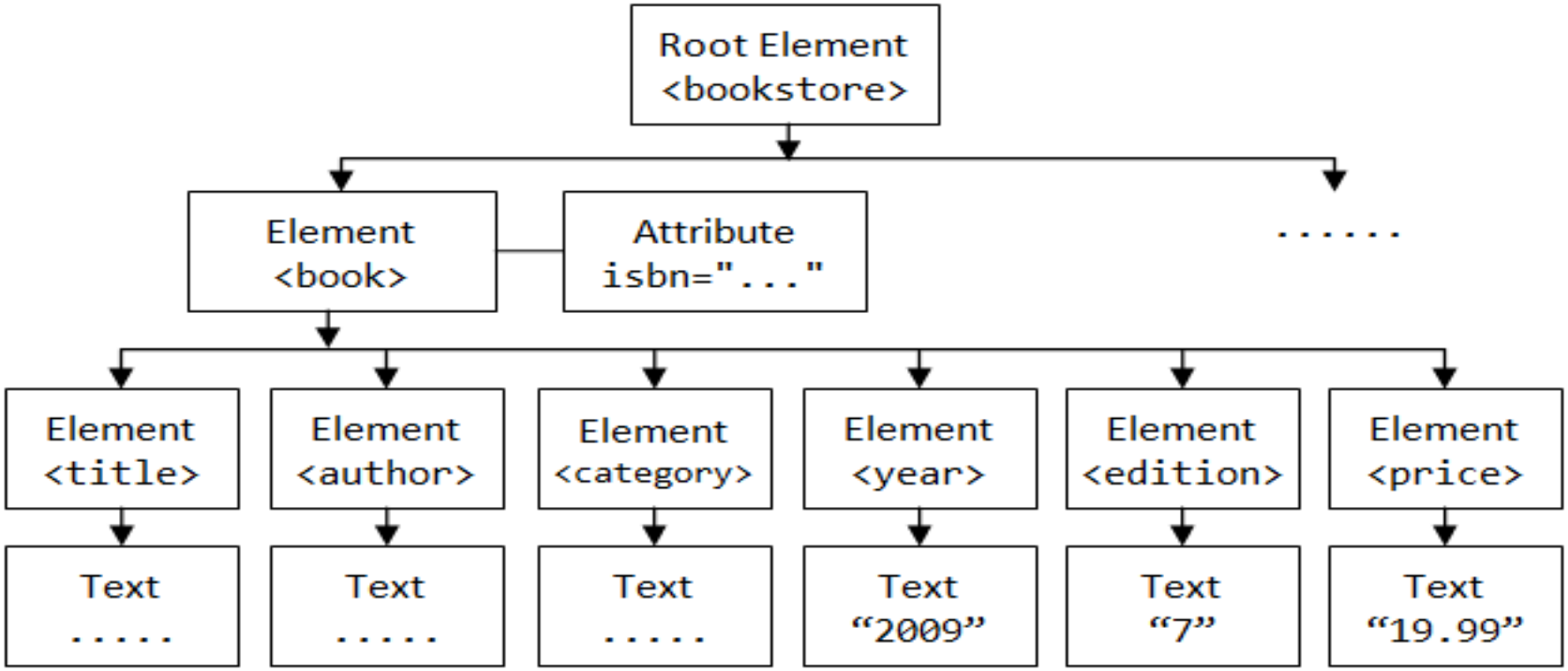


Sintassi XML

1. Elementi devono avere una start tag ed un end tag
2. Un elemento è costituito da start ed end tag + i caratteri inclusi tra i tag (dati e figli)
3. Il nome di un elemento può contenere qualsiasi carattere Unicode ma non spazi vuoti. Deve iniziare con una lettera o '_' o ':'. Non può iniziare con alcune parole riservate come 'xml'
4. Ogni documento XML deve avere una sola radice
5. Gli elementi devono essere propriamente innestati
6. XML è case sensitive. Start e end tags devono essere uguali
7. Uno start tag può contenere attributi nella forma nome-attributo=''valore-attributo''. Attributi forniscono ulteriori informazioni sull'elemento. Valori devono essere quotati(' o '')
8. I caratteri < e > devono essere rimpiazzati da entity reference nella forrma &nome. Ci sono 5 entity reference predefinite: < (<), > (>), & (&), " ("), and ' (')
9. Commenti nella forma <!– commento -->,come HTML
10. Gli spazi sono considerati e il carattere di nuova linea è \n

XML ben formato

- Un documento XML è ben formato se la sua struttura rispetta le specifiche XML (sintatticamente corretto).



Elemento XML

- Documento XML contiene elementi. Un elemento può contenere:
  - Altri elementi
  - Testo
  - Atttributi

```
<bookstore>
  <book category="CHILDREN">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

- Elementi devono seguire le regole di naming:
  - Nomi possono contenere lettere, numeri e altri caratteri
  - Non possono iniziare con un punto o con una cifra
  - Non possono iniziare con xml
  - Non possono contenere spazi
  - Devono essere descrittivi, corti e semplici
  - Evitare ‘-’ , ‘.’ , ‘:’ nel nome
- Elemento può essere esteso per raggruppare più informazioni

Attributo XML

- Attributo aggiungono ulteriori informazioni all'elemento che non sono parte del dato stesso ma possono aiutare il software che utilizza il documento XML

  <file type="gif">computer.gif</file>

- I valori dell'attributo devono essere racchiusi tra doppi apici (come le stringhe in Java)
- Nessuna regola circa l'uso degli elementi o degli attribute

```
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

```
<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

- Attributi non possono contenere valori multipli, strutture ad albero e non sono facilmente espandibili.
- Metadata memorizzati come attributo

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
```

DTD e Schema

- *DTD* e *Schema* sono tecniche per definire la struttura di un documento XML. Descrivono gli oggetti e le relazioni tra loro. Specificano un insieme di vincoli che stabiliscono se un albero XML (ben formato) è VALIDO.

DTD

- Dichiarabile all'interno (inline) di un file XML o in un file esterno

  Un *inline DTD* è racchiuso all'interno di una dichiarazione DOCTYPE

  <!DOCTYPE root-element [

  dichiarazione

  ]>

```
<!DOCTYPE bookstore [
  <!ELEMENT bookstore (book+)>
  <!ELEMENT book (title, author+, category*, language?, year?, edition?, price)>
        <!ATTLIST book ISBN CDATA #REQUIRED>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT author (#PCDATA)>
 <!ELEMENT category (#PCDATA)>
 <!ELEMENT language (#PCDATA)>
 <!ELEMENT year (#PCDATA)>
 <!ELEMENT edition (#PCDATA)>
 <!ELEMENT price (#PCDATA)>
]>
```

Faccio riferimento ad una definizione esterna con

<!DOCTYPE *root-element* **SYSTEM** "*DTD-filename*"**>**

```
  <!ELEMENT bookstore (book+)>
  <!ELEMENT book (title, author+, category*, language?, year?, edition?, price)>
        <!ATTLIST book ISBN CDATA #REQUIRED>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT author (#PCDATA)>
 <!ELEMENT category (#PCDATA)>
 <!ELEMENT language (#PCDATA)>
 <!ELEMENT year (#PCDATA)>
 <!ELEMENT edition (#PCDATA)>
 <!ELEMENT price (#PCDATA)>
```

Sintassi DTD

- **Dichiarazione di un elemento**

  <!ELEMENT nome-elemento (contenuto-elemento)>

  <!ELEMENT nome-elemento categoria>

  Categoria o contenuto:

  – #PCDATA
  – EMPTY
  – ANY

  Indicatori di occorrenza

  – +: 1 o più occorrenze
  – *: 0 o più occorrenze
  – ?: 0 o solo 1 occorrenza
  – Nessun indicatore: esattamente 1

  Connettori

  – , : sequenza degli elementi figli
  – | : scelta tra 1 degli elementi

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT name (first_name,
last_name)>
<!ELEMENT person (name, address?,
email+, hobby*)>
<!ELEMENT message
(#PCDATA|(head, body))>
<!ELEMENT out_of_print EMPTY
<!ELEMENT message ANY>
```

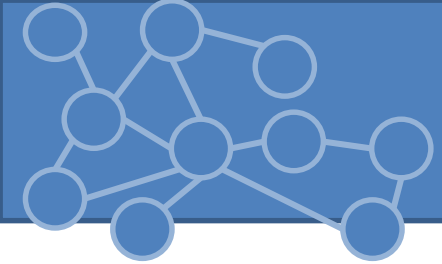
- **Dichiarazione di una lista di attributi**

  <!ATTLIST nome-elemento
  attribute-1-name attribute-1-type default
  >
  default-value|#REQUIRED|#IMPLIED|#FIXED value

  Tipi di attributo:
  - CDATA
  - ID: identificatore univoco
  - IDREF, IDREFS: riferimento all'ID di altro elemento
  - ENTITY, ENTITIES: entity esterna
  - NMTOKEN, NMTOKENS: parole che non contengono spazi
  - Lista di NMTOKEN separati da |

  Default:
  - #REQUIRED
  - #IMPLIED: usa default dell'applicazione
  - #FIXED valore : deve usare valore specificato
  - Valore di default

```
<!ATTLIST payment mode CDATA
"cash">

<!ATTLIST trade action (buy|sell)
#REQUIRED>

<!ATTLIST person
email CDATA #REQUIRED
handphone CDATA #REQUIRED >
```



- **Dichiarazione entity**

  Entity è una variabile per definire della sostituzioni testuali o caratteri speciali. Una volta definita posso invocarla usando *&nome-entity*

  <!ENTITY entity-name "entity-value">

  <!ENTITY entity-name SYSTEM "url">

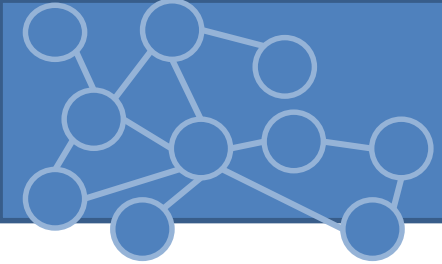
XML Namespace

- *Namespace* nasce da necessità di evitare conflitti tra nomi degli elementi.
- Associa un prefisso ad un URL univoco

  xmlns:abc="http://www.abc.com/XSL/1.0"

  Il prefisso usato come abbreviazione per l'URL e assicura unicità dei nomi (evita naming conflict)
- Valido all'interno dell'elemento in cui è stato dichiarato

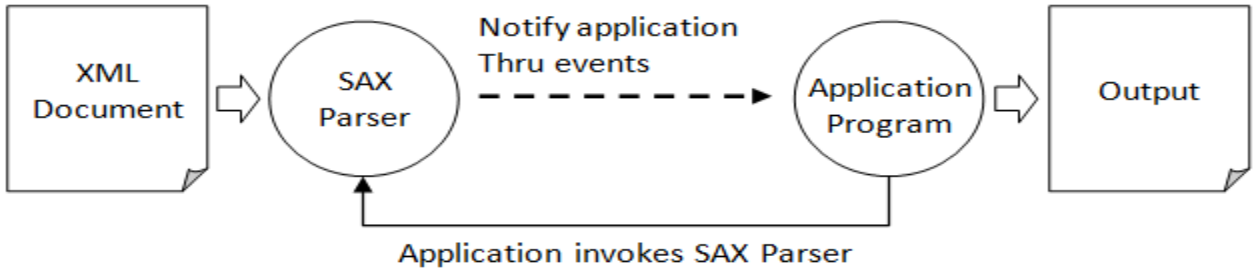
```
<?xml version="1.0"?>
<book_review
   xmlns:abc="http://abc.com/rating/v10"
   xmlns:xyz="http://xyz.com/book/rating"
   xmlns="http://mydotcom.com/rating/book"> //default ns
   <book title="XML for dummies">
      <abc:rating>5</abc:rating>
      <xyz:rating>Excellent</xyz:rating>
      <rating>0.7</rating>
    <book> ...
</book_review>
```



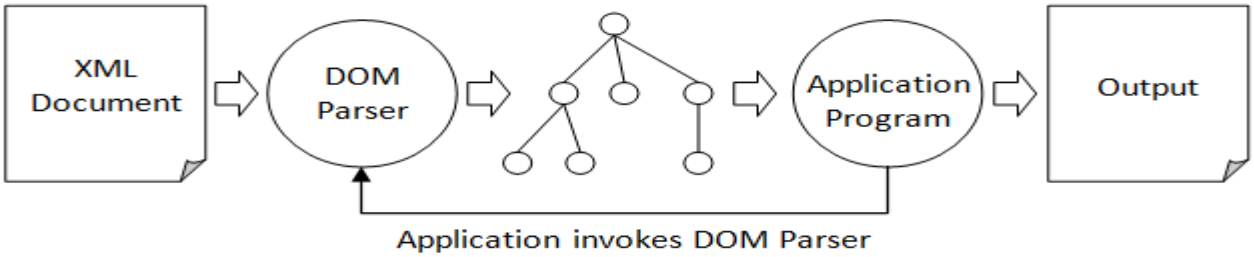
JAXP

- Per processare i dati contenuti in un file XML di deve utilizzare un parser che tookenizza e reperisce gli oggetti di un documento XML. Il parser si pone tra XML e applicazione assicurandosi che documento sia ben formato
- *Java API for XML Processing*: insieme di interfacce standard per la gestione dei file XML
  - Metodo standard di accesso e manipolazione XML indipendente da librerie utilizzate
  - Diverse librerie già implementate
- Si compongono di
  1. *SAX* (Simple API for XML parsing): libreria di basso livello per parsing. Metodi vengono invocati quando ho un evento di apertura/chiusura tag
  2. *DOM* (Document Object Model): libreria di alto livello che trasforma XML in un albero di oggetti con la stessa struttura del documento.
  3. *XSLT* (XML Style Sheet Transformation): applico foglio di stile a XML

- **SAX** è un *event-driven* API. SAX API definisce una serie di metodi di callback che vengono invocati quando occorre un evento durante il parsing Eventi per inizio/fine documento, inizio/fine tag, attributi, entity, commenti..

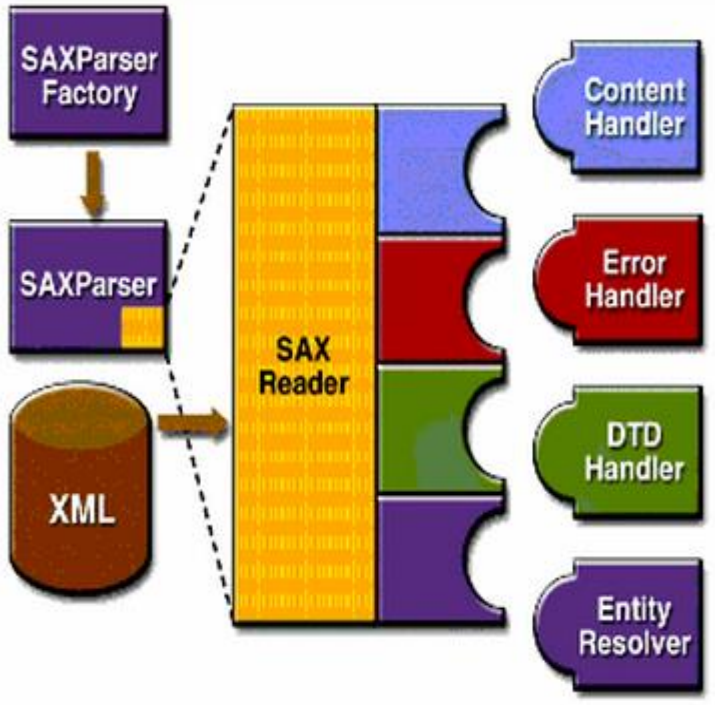


- **DOM** è un *object-oriented* API. Il parser DOM costruisce esplicitamente l'interno albero del documento XML. Manipolo i nodi dell'albero. Meccanismi di query, traversal e manipolazione del modello



SAX

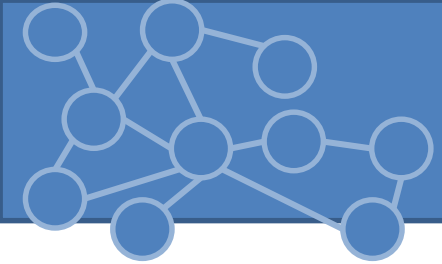
- Si legge XML e ad ogni elemento interessante viene invocato un metodo (metodi di callback) per gestire evento sollevato.
- Molto veloce e non richiede tutto il documento in memoria



SAXParserFactory: crea un'istanza del parser SAX
SAXParser è l'istanza fondamentale, contiene oggetto XMLReader che legge XML e implementa il metodo parse(). Scansiona il documento e invoca i metodi di callback implementati
Interfacce per metodi di callback:

- *Content Handler*
- *ErrorHandler*
- *DTDHandler*
- *EntityResolver*

Basta estendere la classe **DefaultHandler**

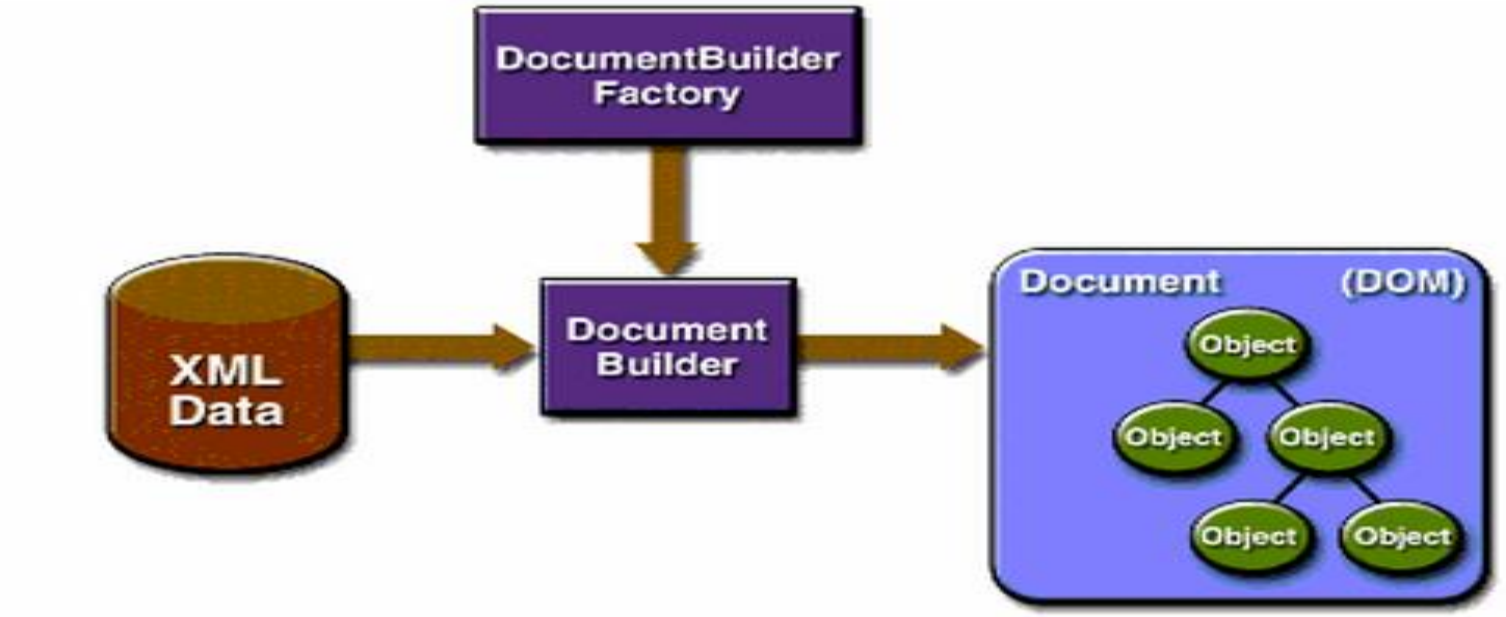


Metodi di callback più comuni:

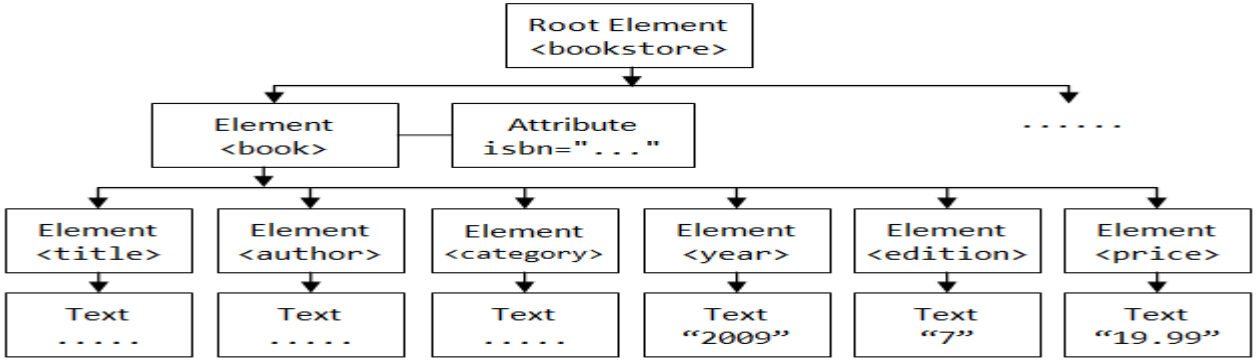
- Inizio e fine documento
  - ***void startDocument() void endDocument()***
- Inizio e fine tag
  - ***void startElement(String uri, String localName, String qName, Attributes attributes)***
  - ***void endElement(String uri, String localName, String qName)***
- Gestione dati all'interno del documento
  - ***void characters(char[] chars, int start, int length)***
- In startElement accedo agli attributi attraverso l'argomento Attributes che dispone dei metodi:
  - ***int getLength()***
  - ***String getValue(int qName)***
  - ***String getValue(int index)***
  - ***String getQName(int index)***

DOM

- Documento XML viene convertito in albero di oggetti a cui posso accedere



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



Per accedere alla root:

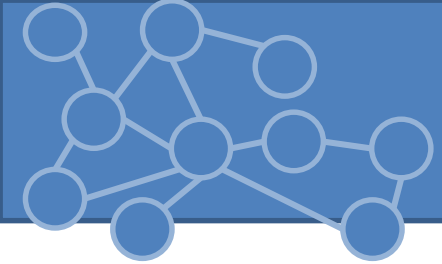
Element root = doc.getDocumentElement();

Ricerca di elementi per nome del tag

NodeList bookNodes = doc.getElementsByTagName("book");

NodeList allNodes = doc.getElementsByTagName("*");

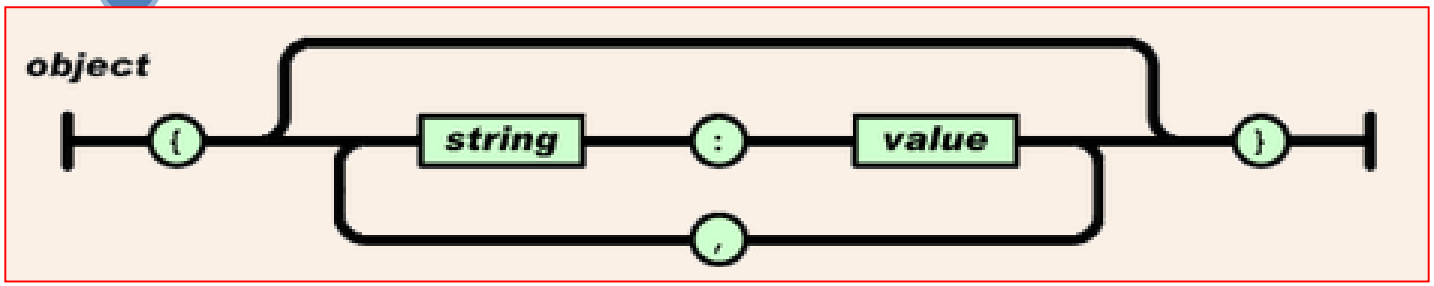
- ***org.w3c.dom.Node*** definisce delle costanti per i tipi di nodo:
  Node.ELEMENT_NODE, Node.ATTRIBUTE_NODE, Node.COMMENT_NODE ,Node.ENTITY_NODE, Node.ENTITY_REFERENCE_NODE,Node.PROCESSIN G_INSTRUCTION_NODE, Node.TEXT_NODE, ..

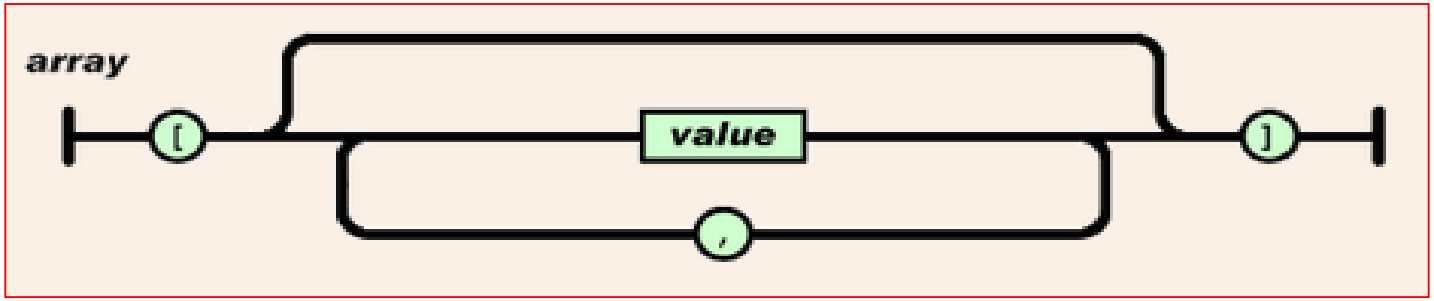


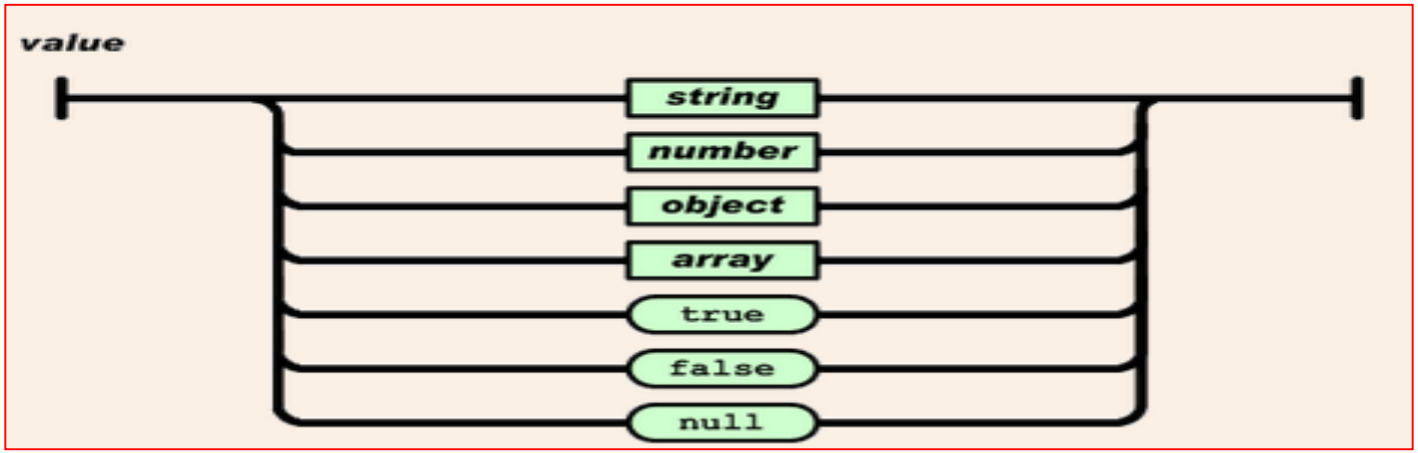
JSON

- *JSON* (JavaScript Object Notation) è un formato per lo scambio di dati. Si basa su un sottoinsieme di JavaScript
- JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza convenzioni conosciute dai programmatori di linguaggi della famiglia del C.
- JSON è basato su due strutture:
  - Un insieme di coppie nome/valore.
  - Un elenco ordinato di valori.
- Queste sono strutture di dati universali. Virtualmente tutti i linguaggi di programmazione moderni li supportano in entrambe le forme

Sintassi


object
{
string
:
value
}
,


array
[
value
]
,


value
string
number
object
array
true
false
null

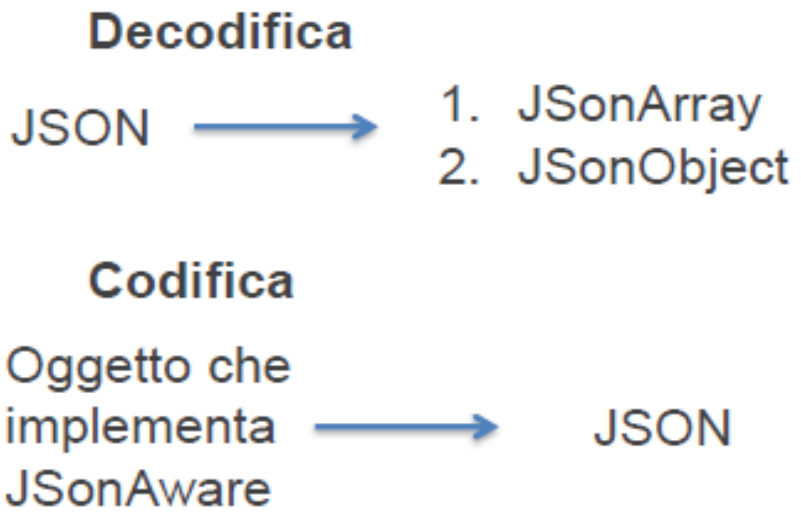
Esempio

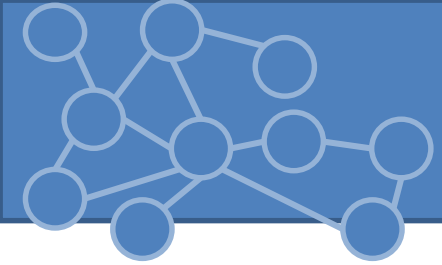

```
{
  "completed_in": 0.018,
  "page": 1,
  "query": "moratti",
  "results": [
    {
      "created_at": "Tue, 20 Nov 2012 11:30:38 +0000",
      "from_user_id": 366233317,
      "text": "RT @IlariaTriplete: #Moratti: \"Ho detto a Cellino che ha una squadra bellissima e gli ho augurato di tenersi i due tecnici che mi sembrano molto in gamba\"",
    },
    {
      "created_at": "Tue, 20 Nov 2012 11:30:11 +0000'',
      "from_user_id": 72009402,

      "text": "Il \"signore\" #Moratti #vandermeyde #cinquantamila #inter http://t.co/pVRozpqv",
    },
    {
      "created_at": "Tue, 20 Nov 2012 11:30:01 +0000",
      "from_user_id": 2800649,
      "text": "RT @IlariaTriplete: #Moratti: \"Il rigore? Sono cose che nel calcio capitano. Il brutto è che capitano tutte le domeniche\''
    }
  ],
  "results_per_page": 3,
}
```

JSON-Simple

- *JSON-simple*: una della tante librerie Java per trattare il formato JSON
  - Semplice e di facile utilizzo (Map e List)
  - Piena compatibilità con la RFC di JSON
  - Codifica e decodifica
  - Dove: http://code.google.com/p/json-simple/downloads/list
- Mapping

JSON	Java
String	java.lang.String
Number	java.lang.Number
True/false	java.lang.Boolean
null	null
Array	java.util.List
Object	java.util.Map





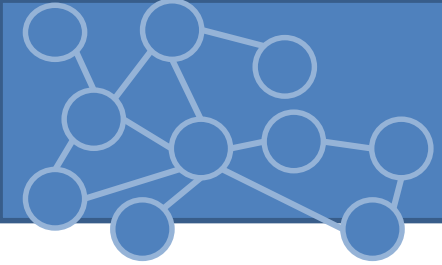
Codifica

Codifica solo con JSONObject

```
JSONObject obj=new JSONObject();
obj.put("name","foo");
obj.put("num",new Integer(100));
obj.put("balance",new Double(1000.21);
obj.put("is_vip",new Boolean(true));
obj.put("nickname",null);
System.out.print(obj);
```

Codifica con Map

```
Map obj=new LinkedHashMap();
obj.put("name","foo");
obj.put("num",new Integer(100));
obj.put("balance",new Double(1000.21));
obj.put("is_vip",new Boolean(true));
obj.put("nickname",null);
String jsonText =
JSONValue.toJSONString(obj);
System.out.print(jsonText);
```



Codifica

Codifica solo con JSONArray

```
JSONArray list = new JSONArray();
list.add("foo");
list.add(new Integer(100));
list.add(new Double(1000.21));
list.add(new Boolean(true));
list.add(null);
System.out.print(list);
```

Codifica con List

```
LinkedList list = new LinkedList();
list.add("foo");
list.add(new Integer(100));
list.add(new Double(1000.21));
list.add(new Boolean(true));
list.add(null);
String jsonText =
JSONValue.toJSONString(list);
System.out.print(jsonText);
```

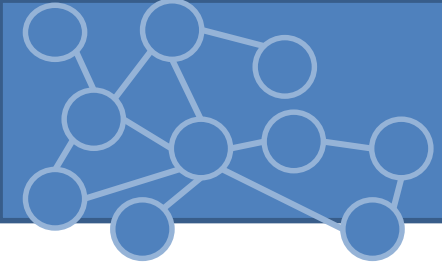
Codifica

Combinazione JSONArray/JSONObject

```
JSONArray list1 = new JSONArray();
list1.add("foo");
list1.add(new Integer(100));
list1.add(new Double(1000.21));

JSONArray list2 = new JSONArray();
list2.add(new Boolean(true));
list2.add(null);

JSONObject obj = new JSONObject();
obj.put("name","foo");
obj.put("num",new Integer(100));
obj.put("balance",new Double(1000.21));
obj.put("is_vip",new Boolean(true));
obj.put("nickname",null);

obj.put("list1", list1);
 obj.put("list2", list2);

System.out.println(obj);
```

Combinazione Map e List

```
Map m1 = new LinkedHashMap();
  Map m2 = new HashMap();
  List  l1 = new LinkedList();

  m1.put("k11","v11");
  m1.put("k12","v12");
  m1.put("k13", "v13");
  m2.put("k21","v21");
  m2.put("k22","v22");
  m2.put("k23","v23");
  l1.add(m1);
  l1.add(m2);

  String jsonString =
JSONValue.toJSONString(l1);

  System.out.println(jsonString);
```



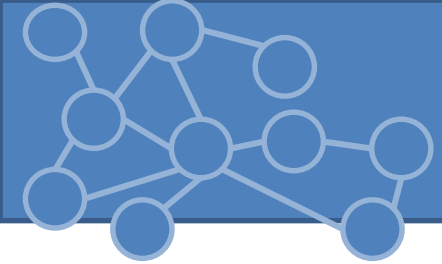
Decodifica

JSONValue

```
String s="[0,{\"1\":{\"2\":{\"3\":{\"4\":[5,{\"6\":7}]}}}}]";
Object obj=JSONValue.parse(s);
JSONArray array=(JSONArray)obj;
JSONObject obj2=(JSONObject)array.get(1);
s="{}";
obj=JSONValue.parse(s);
System.out.println(obj);
s="[5,]";
obj=JSONValue.parse(s);
System.out.println(obj);
s="[5,,2]";
obj=JSONValue.parse(s);
System.out.println(obj);
```

JSONParser

```
JSONParser parser=new JSONParser();
String s="[0,{\"1\":{\"2\":{\"3\":{\"4\":[5,{\"6\":7}]}}}}]";
Object obj=parser.parse(s);
JSONArray array=(JSONArray)obj;
JSONObject obj2=(JSONObject)array.get(1);
 s="{}";
 obj=parser.parse(s);
 System.out.println(obj);
 s="[5,]";
 obj=parser.parse(s);
 System.out.println(obj);
 s="[5,,2]";
 obj=parser.parse(s);
 System.out.println(obj);
```



Esercizi

1. Si codifichi una stringa in formato JSON utilizzando tutti i metodi presentati a lezione e infine si memorizzi la stringa sul file ‘encode.txt’. Il contenuto della stringa è quello presentato come esempio nelle slide.
2. Si legga il file precedentemente salvato e lo si decodifichi utilizzando i tutti i metodi visti a lezione
3. Partendo dal un file bookstore.xml definito come nelle slide, si popoli una libreria (da creare) con oggetti libro costruiti a partire dall’ XML. In una versione di utilizzi un parser SAX mentre nella seconda si utilizzi un parser DOM