LETTURA E SCRITTURA DI XML

PERCHÉ XML? FORMATI DI RAPPRESENTAZIONE MUSICALE

- IEEE 1599
  - http://emipiu.di.unimi.it
- MusicXML
  - http://www.musicxml.com
- Music Encoding Initiative (MEI)
  - http://music-encoding.org/home
- Altri formati meno noti: ChordML, Music Markup Language (MML), ecc.

SUPPORTO XML IN JAVA

- Java mette a disposizione diverse strategie per la gestione in lettura e in scrittura dell'XML

    1. DOM Parser/Builder
    2. SAX Parser
    3. StAx Reader/Writer
    4. JAXB

- Ciascun approccio ha le proprie classi e presenta vantaggi e svantaggi in merito a occupazione di spazio in memoria, velocità di accesso, possibilità di effettuare anche scritture, ecc.

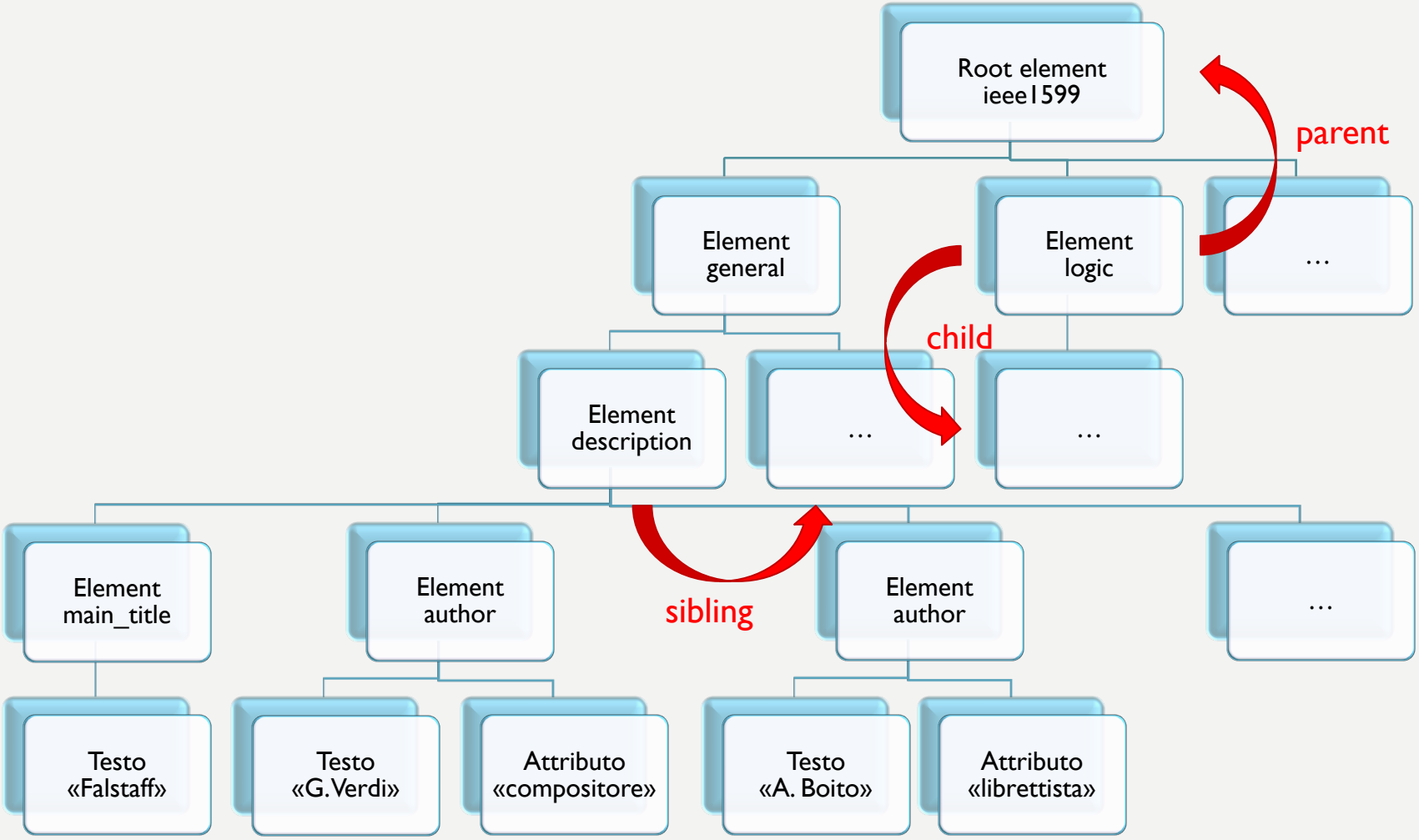
DOM PARSER/BUILDER

- Questa metodologia carica l'intero documento XML in memoria. E' poi possibile utilizzare i metodi propri dell'XML Document Object Model (DOM) per accedervi

- XML DOM definisce un modo standard per accedere ai documenti XML e manipolarli. In DOM il documento viene rappresentato con una struttura ad albero gerarchico (vedi slide successiva)

- Il caricamento delle strutture dati in memoria può essere piuttosto lento

ESEMPIO: STRUTTURA AD ALBERO DI UN FILE IEEE1599



CLASSE DOCUMENTBUILDERFACTORY

- La classe astratta DocumentBuilderFactory consente all'applicazione di istanziare un parser XML che produce alberi DOM a partire da documenti XML.
  Nota: è dichiarata astratta perché poi con il metodo newInstance viene scelto e restituito automaticamente il tipo di reale implementazione

- Metodi principali:
  - static DocumentBuilderFactory newInstance()
    crea una nuova istanza di DocumentBuilderFactory che potrà essere usata per processare file XML. Il costruttore vero e proprio è dichiarato protected
  - abstract DocumentBuilder newDocumentBuilder()
    crea una nuova istanza di DocumentBuilder utilizzando i parametri attualmente configurati
  - void setValidating (boolean validating)
    specifica se il parser effettuerà o meno la validazione dei documenti mentre questi vengono caricati

CLASSE DOCUMENTBUILDER

- La classe DocumentBuilder consente di ottenere istanze di DOM Document a partire da un documento XML. In pratica, questa classe consente di trasformare i file XML in istanze di Document (classe usata per documenti XML e HTML)

- Uno dei metodi principali è Document parse(…), che restituisce oggetti document a partire da:
  - Document parse(File f) → file
  - Document parse(InputStream is) → stream
  - Document parse(String uri) → URI
  - …

CARICAMENTO DI FILE CON DOM

Il costruttore è dichiarato protected, quindi non è richiamabile direttamente. Per istanziare è necessario invocare il metodo newInstance()

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setValidating(true); // opzionale
try {
```

Il costruttore è dichiarato protected, quindi non è richiamabile direttamente. Per istanziare è necessario invocare il metodo newDocumentBuilder()

```
        DocumentBuilder builder = factory.newDocumentBuilder();
        File file = new File("test.xml");
        Document doc = builder.parse(file);
        // Operazioni sul documento
} catch (ParserConfigurationException | SAXException | IOException e) {
        // Gestione eccezioni (multi-catch)
}
```

NODI DOM

- Secondo l'approccio del DOM, qualsiasi oggetto in un documento XML è un nodo. Pertanto:
  - L'intero documento è un nodo di tipo document
  - Ogni elemento XML è un nodo di tipo element
  - Il testo contenuto negli elementi XML è un nodo text
  - Ogni attributo è un nodo di tipo attribute
  - Ogni commento è un nodo di tipo comment

- Attenzione: un errore comune è ritenere che un nodo elemento abbia come valore il testo associato.
  - Ad esempio, nel caso <title>Aida</title> il nodo elemento <title> ha un nodo testo il cui valore è "Aida". Quindi "Aida" NON è il valore dell'elemento <title>

INTERFACCIA DOCUMENT

public interface Document extends Node

- L'interfaccia Document rappresenta un intero documento HTML o XML. Concettualmente, si tratta della radice dell'albero del documento

- L'interfaccia mette a disposizione metodi per accedere in lettura e per creare nuovi nodi nell'albero

METODI PRINCIPALI

- DocumentType getDoctype()
  restituisce il DTD associato a questo documento

- Element getDocumentElement()
  restituisce il nodo radice del documento

- NodeList getElementsByTagName(String tagname)
  restituisce un oggetto NodeList (lista di nodi) di tutti gli elementi, ordinati come nel documento, aventi un certo nome

INTERFACCIA NODE

- Node è il tipo di dato principale nel Document Object Model (DOM), e rappresenta il singolo nodo dell'albero

- Sebbene le classi che implementano l'interfaccia mettano a disposizione gestori per i figli, non tutti i tipi di nodo possono avere figli. Ad esempio, i nodi di tipo Text sono foglie

- In generale, i comportamenti dei metodi sui generici Node variano a seconda della specializzazione del nodo

PRINCIPALI METODI IN LETTURA

- NamedNodeMap getAttributes()
  restituisce un oggetto NamedNodeMap che contiene tutti gli attributi del nodo, ammesso che sia un Elemento, altrimenti null

- NodeList getChildNodes()
  restituisce una NodeList contenente tutti i figli

- Node getFirstChild() / Node getLastChild()
  restituiscono il primo / l'ultimo figlio di questo nodo

- Node getNextSibling() / Node getPreviousSibling()
  restituiscono il nodo immediatamente seguente / precedente quello attuale

PRINCIPALI METODI IN LETTURA

- String getNodeName() / String getNodeValue()
  restituiscono il nome / il valore del nodo, e si comporta in modo diverso a seconda del tipo di nodo

- Document getOwnerDocument()
  restituisce l'oggetto Document associato al nodo

- Node getParentNode()
  è il nodo genitore di quello corrente

- String getTextContent()
  restituisce il contenuto testuale di questo nodo e dei suoi discendenti

INTERFACCE ELEMENT E ATTR

- Numerose interfacce specializzano l'interfaccia base Node: Element e Attr sono due di queste

Metodi della classe Element:

- String getAttribute(String name)
  restituisce il valore dell'attributo specificato

- Attr getAttributeNode(String name)
  restituisce l'attributo con nome specificato

- boolean hasAttribute(String name)
  controlla l'esistenza dell'attributo specificato

- NodeList getElementsByTagName(String tagname)
  restituisce un oggetto NodeList (lista di nodi) di tutti i sotto-elementi, ordinati come nel documento, aventi un certo nome

Metodi della classe Attr:

- String getValue()
  restituisce il valore dell'attributo

ESEMPI

(FirstDOMParsing.java)

Il software carica in un oggetto di classe Document un file XML e manda in output alcuni contenuti (nodo radice, elementi, attributi, nodi di testo)

(DOMParentChildrenSiblings.java)

Il software esemplifica la navigazione dell'albero DOM tramite i concetti di padre (parent), figli (child) e fratelli (sibling)

ESERCIZIO

(XMLStats.java)

Scrivere un software che crea delle statistiche sui contenuti musicali di un file XML in formato IEEE 1599. In particolare, si contino le occorrenze di ogni altezza (trascurando stato di alterazione e ottava) e le occorrenze di ogni valore ritmico (basandosi unicamente sui denominatori, tralasciando numeratori diversi da 1, punti di valore e gruppi irregolari)

In un file IEEE 1599 le note/pause si trovano nel DOM sul ramo:
/ieee1599/logic/los/part/measure/voice

Es. di contenuto di voice:

```
<chord event_ref="Piano_2_3_voice0_measure41_ev0">
   <duration num="1" den="4" />
   <notehead>
      <pitch octave="2" step="G" actual_accidental="natural" />
   </notehead>
   <notehead>
      <pitch octave="3" step="D" actual_accidental="natural" />
   </notehead>
</chord>
<rest event_ref="Piano_2_3_voice0_measure41_ev1">
   <duration num="1" den="4" />
</rest>
<rest event_ref="Piano_2_3_voice0_measure41_ev2">
   <duration num="1" den="2" />
</rest>
```

SCRITTURA DI XML

- Finora sono state usate le classi che permettono di leggere informazioni da file XML, la parte seguente sarà dedicata alla creazione/modifica

- L'approccio è quello di XML DOM, in cui le gerarchie di un documento XML vengono rappresentate come un albero di nodi, e ogni aspetto dell'XML (compresi i commenti) è l'istanza di un nodo

MODIFICARE UN DOCUMENTO XML

- Primo obiettivo: aggiungere nodi a XML esistente
- Esattamente come per la lettura di XML, è necessario innanzitutto istanziare un oggetto dell'interfaccia Document
- I passaggi sono quelli già visti:
  1. istanziare un oggetto DocumentBuilderFactory
  2. istanziare un oggetto DocumentBuilder
  3. ottenere un oggetto Document dal parsing di un file in formato XML

METODI DI DOCUMENT PER LA CREAZIONE DI NODI

- Metodi principali (della classe Document):
  - Attr createAttribute(String name)
    crea un Attr del nome specificato.
  - CDATASection createCDATASection(String data)
    crea un nodo CDATASection il cui valore è quello della stringa.
  - Comment createComment(String data)
    crea un nodo Comment il cui valore è la stringa in ingresso.
  - DocumentFragment createDocumentFragment()
    crea un oggetto DocumentFragment vuoto.
  - Element createElement(String tagName)
    crea un nodo Element.
  - Text createTextNode(String data)
    crea un nodo Text contenente la stringa specificata.

AGGIUNGERE NODI A DOCUMENT

- Una volta creato un nodo, è necessario inserirlo nella gerarchia XML nel punto corretto
- Il metodo principale per farlo è invocare appendChild(Node n) sul nodo cui deve essere agganciato il nuovo nodo figlio
- Per quanto riguarda gli attributi, è possibile aggiungerli all'elemento invocando sul corrispondente nodo Element il metodo void setAttribute(String name, String value)

SALVARE XML SU FILE

```
File file = new File("nuova_prova.xml")
Result output = new StreamResult(file);
Source input = new DOMSource(document);
```

Istanza dell'interfaccia
Document da salvare

```
TransformerFactory tf = TransformerFactory.newInstance();
Transformer t = tf.newTransformer();
// info di indentazione, opzionale
t.setOutputProperty(OutputKeys.INDENT, "yes");
t.setOutputProperty("{http://xml.apache.org/xslt}indent-
amount", "3");
t.transform(input, output);
```

CREARE UN NUOVO DOCUMENTO DA ZERO

- Primo problema: creare il nodo radice

```
DocumentBuilderFactory docFactory =
                    DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder =
                          docFactory.newDocumentBuilder();
Document doc = docBuilder.newDocument(); // radice
```

- Successivamente, è possibile creare i singoli nodi e aggiungerli alla gerarchia come visto precedentemente.

ESEMPIO

(CreateNodes.java)

Il software effettua operazioni basilari di creazione di elementi, attributi e commenti ed esemplifica l'aggiunta di nodi a una gerarchia pre-esistente

COME STRUTTURARE L'INFORMAZIONE IN XML

- Il problema della strutturazione dell'informazione in XML (ad es. come chiamare gli elementi e come annidarli e con quale molteplicità, quali e quanti attributi prevedere per ciascun elemento, ecc.) non sussiste quando si adotta un formato già esistente (ad es. IEEE1599 o MusicXML)

- In generale, esistono approcci molto eterogenei (basati solo su elementi, con al più contenuto testuale, o con ampio uso di attributi) che però risultano idempotenti nel rappresentare l'informazione. Un esempio è mostrato alla slide successiva

- Si devono però ricordare alcune caratteristiche distintive: ad esempio, gli attributi non possono essere annidati; gli attributi possono essere di tipo enumerativo, o ID o IDREF, mentre gli elementi no, ecc.

ESEMPIO

```
<melodia>
  <nota>
    <nome>Do</nome>
    <alterazione>diesis</alterazione>
    <durata>
      <num>1</num>
      <den>4</den>
    </durata>
  </nota>
  <nota>
    <nome>Re</nome>
    <durata>
      <num>1</num>
      <den>8</den>
    </durata>
  </nota>
  …
</melodia>
```

```
<melodia>
  <nota nome="Do" alt="diesis" num="1" den="4" />
  <nota nome="Re" num="1" den="8" />
</melodia>
```

ESERCIZIO

(IEEE1599Stub.java)

- Si crei via codice lo scheletro di un file IEEE 1599 valido, ossia:

```
<ieee1599 version="1.0">
  <general>
    <description>
      <main_title/>
      <author/>
    </description>
  </general>
  <logic>
    <spine>
      <event id="event_0" timing="null" hpos="null"/>
    </spine>
  </logic>
</ieee1599>
```

ESEMPIO

(SaveMelody.java)

- Questo esempio crea un documento XML ex-novo, effettuando il parsing di una struttura dati interna che descrive una melodia come successione di note. La scelta del formato XML è del tutto arbitraria e non aderisce ad alcun formato standardizzato

- Una funzione appositamente aggiunta mostra come – in un caso semplice – sia possibile (ma sintatticamente più rischioso) produrre l'XML in formato stringa

- Esempio formato creato:

```
<melodia>
  <nota>
    <altezza alterazione="natural" nome="E" ottava="1"/>
    <durata den="4" num="1"/>
  </nota>
  <nota>
    <altezza alterazione="flat" nome="A" ottava="0"/>
    <durata den="8" num="1"/>
  </nota>
  <pausa>
    <durata den="2" num="1"/>
  </pausa>
</melodia>
```

ESERCIZIO

(IEEE1599Convert.java)

- A partire da un file IEEE1599 (ad esempio gounod_ave_maria.xml) si crei una conversione nel formato creato nella slide precedente di note e pause presenti, senza tenere conto di accordi, parti, voci, misure, ecc.

INTRODUZIONE A XPATH

- XPath è un linguaggio per trovare informazioni in un file XML. Viene utilizzato per navigare attraverso elementi e attributi di un documento XML attraverso l'utilizzo di espressioni di interrogazione

- E' uno standard W3C. Versione attuale: 3.0 (10 Aprile 2014)
  Documentazione ufficiale: http://www.w3.org/TR/xpath/

- XPath è un linguaggio indipendente, ed è supportato dai maggiori linguaggi di programmazione moderni: C/C++, C# (piattaforma .NET), Delphi, Java, JavaScript, Perl, PHP, Python, Ruby, Scheme, …

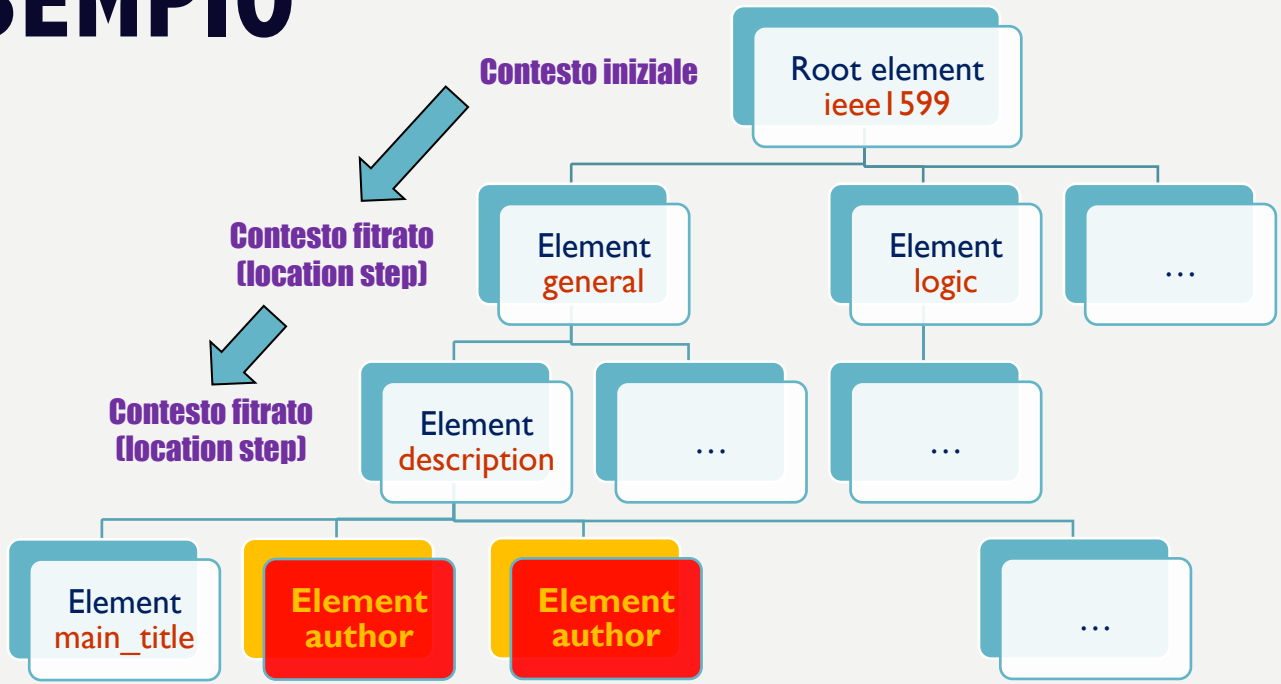
APPROCCIO

- XPath si compone di espressioni (o query) che vengono di volta in volta valutate in un determinato contesto restituendo uno o più nodi del documento XML. Un'espressione è costituita da combinazioni di uno o più chunk di testo separati da caratteri speciali

- Le espressioni XPath individuano contenuti del documento XML o parte di questi: tale processo avviene in uno o più passi, detti location step, che messi insieme danno vita ad un percorso completo di localizzazione, ossia un location path. Ad ogni passo di valutazione delle location step (ossia procedendo da sinistra verso destra della location path) vengono esclusi alcuni nodi dalla selezione e vengono considerati i rimanenti

- Per contesto si intende l'insieme dei nodi che ad un dato punto della location path sono selezionati

LOCATION STEP E LOCATION PATH: ESEMPIO



L'immagine mostra il risultato della location path

/ieee1599/general/description/author

In questo modo vengono selezionati soltanto i nodi-elemento *author* figli dei nodi-elemento *description* figli a loro volta di *general*, mentre tutti gli altri nodi presenti all'interno del documento vengono trascurati

TOKEN

- Le espressioni XPath sono costituite da token e delimitatori

- I token sono insiemi di caratteri Unicode che assumono il significato di stringa se delimitati da virgolette ed il significato di nodi-elemento se privi di virgolette

- Le specifiche XPath prevedono l'utilizzo del carattere * come wildcard: il suo utilizzo permette di selezionare tutti i nodi del contesto indipendentemente dal loro nome
  - Ad esempio l'espressione /ieee1599/general/* permette di selezionare indipendentemente tutti i nodi figlio degli elementi general (in questo caso dell'unico elemento general) presenti sotto l'elemento radice del documento XML. Come si vedrà nel seguito, il primo carattere / identifica il nodo radice, e un percorso aperto da / viene considerato assoluto

DELIMITATORI

- I token di un'espressione XPath sono separati da opportuni delimitatori:
  - / separa un passo di localizzazione da un altro
  - [ ] permettono di definire un predicato (vedi più avanti)
  - =, !=, <, >, <=, >= vengono usati all'interno di un predicato per verificare il valore true o false del test applicato sul contesto
  - :: separa l'asse di un'espressione XPath dal nome di uno specifico nodo
  - //, @, ., .. costituiscono delle forme abbreviate per indicare rispettivamente i concetti di "se stesso o discendente", "attributo", "se stesso", "genitore"
  - | permette di definire un'espressione XPath composta dall'unione di più espressioni XPath
  - ( ) permettono di raggruppare sottoespressioni XPath o delimitare gli argomenti di una funzione XPath
  - +, -, *, div, mod agiscono come operatori numerici col significato di somma, sottrazione, moltiplicazione, divisione e modulo

TIPI DI DATI

XPath è in grado di trattare quattro tipi differenti di dati: stringhe, numeri, booleani e nodi

- Le stringhe sono insiemi di caratteri delimitati dalle virgolette
- Un tipo numerico è un numero considerato in floating point sul quale è possibile applicare gli operatori numerici visti in precedenza ottenendo un risultato sempre di tipo numerico.
  XPath definisce anche il valore speciale NaN (Not a Number)
- Un tipo booleano può assumere valore true o false. L'uso principale all'interno di un'espressione XPath riguarda la composizione dei predicati (vedi più avanti)
- I nodi infine includono non solo elementi e attributi ma anche commenti, nodi testo, ecc. Per node-set si intende un insieme di nodi identificati da un location step (o location path)

VERIFICA DI ESPRESSIONI (O QUERY) XPATH

- Poichè XPath è un linguaggio a sè stante, esistono strumenti per verificare la correttezza di una espressione/query XPath, che in inglese prendono il nome di *tester* o *evaluator*

- Ad esempio, esistono siti Web che consentono di caricare e/o copiare e incollare documenti XML e di immettere una stringa XPath:
  - http://www.freeformatter.com/xpath-tester.html
  - http://xpath.online-toolz.com/tools/xpath-editor.php
  - http://www.xpathtester.com/xpath
  - …

SINTASSI DI UN LOCATION STEP

- La sintassi tipica di un location step prevede l'asse (opzionale), il nodetest (obbligatorio) ed il predicato (opzionale) organizzati secondo la seguente sintassi:

  asse::nodetest[predicato]

- Il nodetest indica il tipo di nodi che vogliamo selezionare. Esistono due possibili approcci:
  1. identificazione dei nodi in base al nome degli elementi
  2. identificazione dei nodi in base al tipo degli elementi

- Per selezionare gli elementi con un particolare nome è sufficiente indicare come nodetest il nome dell'elemento

- Asse e predicato verranno trattati più avanti

ESEMPI DI LOCATION STEP

- /ieee1599/logic
  selezionare tutti gli elementi figli dell'elemento radice che hanno nome pari a "logic" (sintassi con solo nodetest)

- Per selezionare tutti gli elementi a prescindere dal loro nome è possibile utilizzare il carattere wildcard *. Per selezionare tutti i figli dell'elemento radice: /*

- /ieee1599/general/description/author/text()
  per selezionare soltanto i nodi-testo. In tal modo si selezionano i nodi-testo figli degli elementi che hanno nome "author" (figli a loro volta dell'elemento radice). Analogamente si può utilizzare node() per selezionare tutti i nodi indipendentemente dal tipo, comment() per selezionare i nodi-commento, ecc.

ASSI

- Sintassi generica di un location step: asse::nodetest[predicato]

- L'asse di un location step permette di indicare la direzione verso la quale si esaminano i nodi precedentemente filtrati

- XPath definisce 13 tipi differenti di assi:
  - child:: individua il nodo immediatamente discendente dal nodo contesto. L'asse child è impostato di default, quindi le espressioni /child::ieee1599/child::logic e /ieee1599/logic sono equivalenti
  - parent:: individua il nodo genitore del nodo contesto
  - descendant:: individua tutti i nodi discendenti del nodo contesto a qualsiasi profondità (i figli, i figli dei figli...)
  - ancestor:: individua tutti gli antenati del nodo contesto fino all'elemento radice (il padre, il padre del padre...)
  - descendant-or-self:: e ancestor-or-self:: individuano il nodo contesto stesso e tutti i suoi discendenti/antenati

ASSI

- XPath definisce 13 tipi differenti di assi [continua]:
  - following:: e preceding:: individuano tutti i nodi che all'interno del documento XML seguono/precedono il nodo contesto esclusi i discendenti/antenati del nodo contesto
  - following-sibling:: e preceding-sibling:: individuano tutti i nodi che seguono/precedono il nodo contesto e che condividono con questo lo stesso nodo parent
  - attribute:: individua gli attributi del nodo contesto
  - namespace:: individua i nodi namespace
  - self:: individua il nodo contesto stesso

- Per l'asse attribute:: esiste l'abbreviazione @

- Per l'asse descendant-or-self:: esiste l'abbreviazione //

PREDICATI

- Sintassi generica di una location step: asse::nodetest[predicato]

- L'utilità dei predicati sta nella possibilità di poter filtrare ulteriormente i nodi dal contesto corrente in base al verificarsi di opportune condizioni

- Generalmente un predicato è racchiuso fra parentesi quadre ed espresso in termini di espressioni booleane del tipo:

  valore1 operatore valore2

  dove valore1 e valore2 sono delle espressioni XPath e operatore è uno degli operatori booleani descritti in precedenza

PREDICATI

- Ogni location path che appare all'interno del predicato fa riferimento al contesto stabilito dai location step che precedono il predicato stesso

- Ad esempio, c'è differenza tra le seguenti espressioni:
  //description/author[@type="composer"]
  //description[author/@type="composer"]
  Il primo predicato seleziona i nodi *author* (con attributo *type* pari a "composer") mentre il secondo gli elementi *description* aventi almeno un elemento-figlio *author* (sempre con attributo *type* pari a "composer")

ESEMPI DI PREDICATI SU IEEE 1599

- //measure[@number="1"]
  seleziona tutti i nodi discendenti della root chiamati measure aventi attributo number il cui valore è «1»

- //chord[notehead/tie]
  seleziona tutti i nodi chord aventi un figlio notehead che presenta un figlio tie

- //measure[*/*/*/@den="16"]
  seleziona tutte le battute che contengono figure di sedicesimo

- //duration[ancestor::rest]
  seleziona la durata di tutte e sole le pause (non degli accordi)

PREDICATI INNESTATI

E' possibile definire all'interno di un predicato altri predicati innestati

- Esempi:
  //measure[voice/rest/duration[@den='4']]
  //measure[descendant::rest[duration[@den='4']]]
  seleziona tutte le battute che contengono pause di durata x quarti.

E' possibile comporre predicati mediante operatori logici and e or

- Esempio: //measure[voice/rest/duration[@num='3' and @den='4']]

Un utilizzo particolare dei predicati è quello di verificare o meno l'esistenza di un determinato nodo figlio del contesto attuale

- Esempio: //measure[voice/rest] verifica se esiste una measure con delle pause

TEMPI DI CALCOLO DI UNA QUERY XPATH

- In XPath normalmente esistono numerose varianti sintattiche per ottenere lo stesso risultato
  - Ad esempio, nel caso di documenti IEEE 1599 (in cui author si trova sempre e solo in un determinato punto della gerarchia) le seguenti scritture sono equivalenti:
    /ieee1599/general/description/author/text()
    /*/*/*/author/text()
    //author/text()

- Considerando come viene ricalcolato il contesto a ogni location step, ossia come viene potato l'albero, la prima espressione (per quanto più tediosa da formulare) è certamente la più efficiente

- In generale, ove possibile è consigliabile esplicitare ogni step. Alle volte però l'uso di descendant:: e * è necessario (in particolare se l'oggetto della ricerca sta su rami o a livelli gerarchici distinti)

ESERCIZI

- Si scarichi un file IEEE1599 di esempio, e si utilizzi un software per la valutazione di espressioni XPath al fine di determinare:

1. l'elemento titolo, ossia ieee1599 > general > description > main_title
2. l'elenco di tutte le pause, ossia gli elementi rest
3. il nome di tutti gli autori, ossia il text content dell'elemento author
4. l'elemento author il cui attributo type ha valore «composer»
5. la descrizione di tutti i file collegati, ossia l'attributo description di tutti gli elementi related_file
6. i riferimenti a tutti i file presenti, ossia l'elenco di tutti gli elementi che presentano un attributo file_name
7. tutti i Sib della 4a ottava, ossia gli elementi pitch aventi attributi step="B", octave="4" e actual_accidental="flat"
8. tutte le note con alterazione e legatura, ossia gli elementi notehead che hanno sia figli printed_accidentals sia tie
9. tutti gli elementi fratelli di augmentation_dots che li precedono
10. tutti gli accordi (elementi chord) di misura 36 (attributo number="36" di measure) contenenti punti di valore (presenza del sotto-elemento augmentation_dots)

ESERCIZI - SOLUZIONE

1. /ieee1599/general/description/main_title
2. /ieee1599/logic/los/part/measure/voice/rest
   Alternative: //rest oppure descendant::rest
3. //author/text(), in alternativa con path assoluto completo
4. //author[@type="composer"]
5. //related_file/@description
6. descendant::*[@file_name] o il suo alias //*[@file_name]
7. //pitch[@step="B" and @octave="4" and @actual_accidental="flat"]
8. //notehead[printed_accidentals and tie]
9. //augmentation_dots/preceding-sibling::*
10. //chord[ancestor::measure[@number="36"] and descendant::augmentation_dots]

- Si osservi che le soluzioni sono valide in generale per qualsiasi file IEEE 1599.

FUNZIONI XPATH

- Le funzioni XPath utilizzate generalmente all'interno dei predicati, assumono una struttura sintattica del tipo:

  nomefunzione(param1, param2,..., param n)

- Ogni funzione ha un nome distinto ed accetta uno o più argomenti in base ai quali modifica il suo comportamento

- Le funzioni disponibili in XPath possono essere classificate in base al tipo del valore che restituiscono o in base al tipo degli argomenti sui quali operano

FUNZIONI NODE-SET

- Una funzione node-set agisce su un node-set e restituisce un node-set

- Alcuni esempi:
  - last() restituisce la posizione dell'ultimo nodo del node-set corrispondente al contesto attualmente selezionato
  - position() restituisce la posizione di un determinato nodo all'interno del node-set corrispondente al contesto attualmente selezionato
  - count(nodeset) restituisce il numero di nodi appartenenti ad un node-set, agisce come last() ma a differenza di quest'ultimo accetta in ingresso un'espressione XPATH che individua il node-set sul quale la funzione deve operare
  - …

XML DI ESEMPIO

- Il seguente XML di esempio (che è ben formato ma non aderisce ad alcuno specifico formato) può essere copiato e incollato in un valutatore online per fare prove sugli argomenti della lezione, ad esempio http://www.freeformatter.com/xpath-tester.html

```
<opere>
  <artista nome="Giuseppe Verdi">
    <opera anno="1871" atti="4">Aida</opera>
    <opera anno="1851" atti="3">Rigoletto</opera>
    <opera anno="1857" atti="3" prologo="1">Simon Boccanegra</opera>
  </artista>
  <artista nome="Giacomo Puccini">
    <opera anno="1896" atti="4">La bohème</opera>
    <opera anno="1887" atti="3">Tosca</opera>
    <opera anno="1926" atti="3">Turandot</opera>
  </artista>
</opere>
```

ESEMPI CON FUNZIONI NODE-SET

- Se si vuole contare la numerosità di un determinato elemento all'interno del documento XML:
  count(//artista)

- Un caso particolare è rappresentato dai predicati che filtrano in base alla posizione occupata dai nodi selezionati ad un determinato momento. Volendo selezionare il secondo autore di tutto il documento ad esempio è possibile utilizzare l'espressione:
  //artista[position()=2] oppure l'alias //artista[2]

  Nell'esempio XML corrente, è anche //artista[last()]

  L'espressione //artista[position()=4] invece non avrebbe restituito alcuna corrispondenza

FUNZIONI STRINGA

- Le funzioni stringa si occupano di manipolare stringhe

- string(anything)
  converte ogni argomento (opzionale) in una stringa di caratteri.
  - Se l'argomento è un nodeset la funzione restituisce la conversione in stringa del primo elemento (se è vuoto viene restituita una stringa vuota)
  - Se è un numero viene restituita la corrispondente rappresentazione stringa
  - Se è un booleano restituisce le stringhe "true" o "false" a seconda del valore assunto dal booleano stesso

- concat(stringa1, stringa2, ..., stringan)
  concatena due o più stringhe in un'unica stringa

- starts-with(stringa1, stringa2)
  restituisce true se la prima stringa inizia con il valore della seconda, altrimenti false

FUNZIONI STRINGA

- contains(stringa1, stringa2)
  restituisce true se la prima stringa contiene la seconda, false viceversa

- substring(stringa, numero1[, numero2])
  restituisce una sottostringa della stringa passata come parametro a partire dalla posizione indicata dall'argomento numero1.
  Opzionalmente è possibile indicare quanti caratteri devono essere estratti (numero2): se tale parametro non viene specificato vengono estratti tutti i caratteri a partire dalla posizione indicata da numero1.
  La prima posizione è rappresentata dal numero 1 e non dal numero 0 come avviene in molti linguaggi di programmazione.

FUNZIONI STRINGA

- substring-before(stringa1, stringa2)
  substring-after(stringa1, stringa2)
  restituiscono la porzione di stringa1 precedente/successiva alla prima occorrenza della stringa2 all'interno della stringa1. Se la stringa2 non appare la funzione restituisce la stringa vuota

- string-length(stringa) restituisce il numero di caratteri della stringa passata come parametro, se il parametro non viene indicato la funzione agisce sullo string-value del contesto

- normalized-space(stringa) elimina caratteri whitespace estranei (ovvero quei caratteri che visivamente sono spazi bianchi)

ESEMPI CON FUNZIONI STRINGA

- Ad esempio, se si desidera selezionare non l'attributo nome dell'elemento artista ma il solo contenuto testuale, ossia il valore:
  string(/opere/artista/@nome)

- Visualizzazione degli attributi tramite concatenazione di stringhe: titolo e anno di prima rappresentazione dell'ultima opera (nell'ordine dell'XML) del secondo artista:

```
concat("Prima esecuzione di ",
      //artista[2]/opera[last()]/text() ,
      ": ",
      string(//artista[2]/opera[last()]/@anno))
```

Risultato atteso: «Prima esecuzione di Turandot: 1926»

FUNZIONI BOOLEANE

- Le funzioni booleane restituiscono sempre valori booleani

- boolean(anytype) converte il parametro passato alla funzione in un booleano in base alle seguenti regole:
  - se il parametro è una stringa restituisce true se la stringa ha una lunghezza maggiore di 0, false altrimenti
  - se il parametro è un numero legittimo (e non NaN) restituisce false se il numero è pari a 0 e true altrimenti

- not(boolean) restituisce il negato del valore booleano passato come parametro
- true() restituisce il valore true, false() restituisce il valore false

FUNZIONI NUMERICHE

- number(anytype) converte il parametro passato come argomento in un tipo numerico secondo le seguenti regole:
  - se il parametro è una stringa questa viene convertita in un numero solo se costituita da uno spazio opzionale, un segno meno opzionale, un altro spazio opzionale, un valore numerico ed un altro spazio opzionale: negli altri casi viene convertita in NaN. Esempi:
    «3» OK « -3 » OK «- 3» OK «Pippo» NaN
  - se il parametro è un booleano viene convertito in 0 se pari a false e in 1 se pari a true
  - se il parametro è un nodeset questo viene dapprima convertito in stringa e poi in numero

FUNZIONI NUMERICHE

- sum(nodeset) effettua la somma di tutti i valori dei nodi presenti convertiti in tipi numerici
- floor(number) restituisce il numero intero immediatamente più piccolo del numero passato come argomento
- ceiling(number) restituisce il numero intero immediatamente più grande del numero passato come argomento
- round(number) restituisce il numero intero immediatamente più vicino a quello passato come argomento della funzione.

ESEMPI CON FUNZIONI BOOLEANE E NUMERICHE

- Selezione di tutte le opere dell'800 (basandosi sul fatto che il valore-stringa dell'attributo Anno inizi per «18»)

  ```
  //opera[starts-with(@anno,'18') = true()]
  //opera[starts-with(@anno,'18') = 1]
  ```

- Visualizzazione del titolo di tutte le opere successive al 1880 (basandosi sulla conversione in numero del valore dell'attributo anno)

  ```
  //opera[number(@anno) > 1880]/text()
  ```

- Somma degli anni di tutte le opere

  ```
  sum(//opera/@anno)
  ```

PASSAGGI PER INTEGRARE XPATH IN JAVA

- Per effettuare query Xpath in Java, sono richiesti diversi passaggi:

1. Creare un oggetto di classe DocumentBuilderFactory
2. Creare un parser XML in DOM, di classe DocumentBuilder
3. Effettuare il parsing dell'XML utilizzando tale parser, il che produce un Document
4. Creare un oggetto XPath (classi XPathFactory e XPath)
5. Usare l'XPath per navigare l'XML

- Si noti che i primi 3 passaggi sono quelli tipici del parsing XML utilizzando il DOM

CREARE UN OGGETTO XPATH

- Per creare un'espressione XPath, si deve dapprima utilizzare la classe XPathFactory per poi istanziare il vero e proprio oggetto XPath

- Una volta importate le classi

```
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathFactory;
```

- Sintassi completa:

```
XPathFactory myXPathFactory = XPathFactory.newInstance();
XPath myXPath = myXPathFactory.newXPath();
```

- Sintassi sintetica (in alternativa):

```
XPath myXPath =  XPathFactory.newInstance().newXPath();
```

COMPILAZIONE DELL'ESPRESSIONE E VALUTAZIONE

- Compilazione dell'espressione (formato stringa)

```
String myExpr = "/ieee1599/*/*";
```

- Valutazione dell'espressione: dipende dal tipo restituito
  - Stringhe

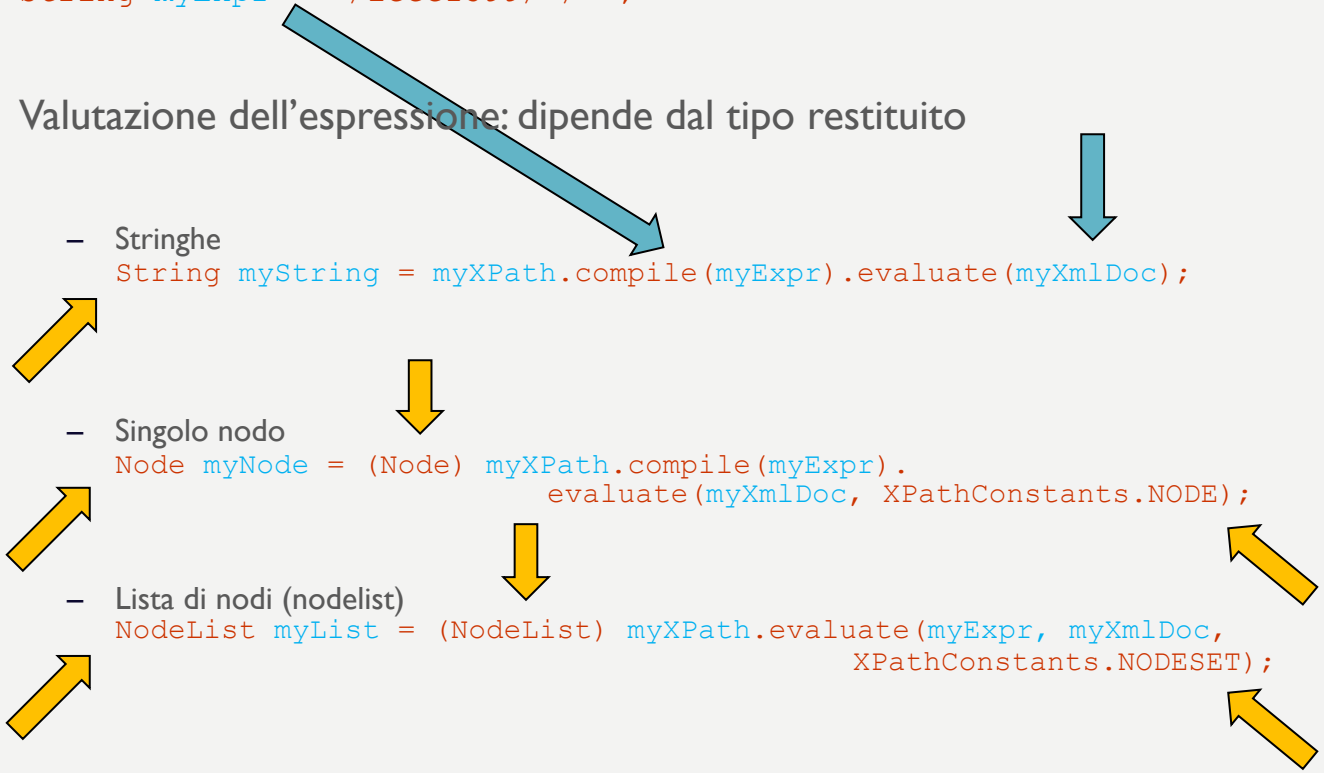
```
String myString = myXPath.compile(myExpr).evaluate(myXmlDoc);
```

  - Singolo nodo

```
Node myNode = (Node) myXPath.compile(myExpr).
                     evaluate(myXmlDoc, XPathConstants.NODE);
```

  - Lista di nodi (nodelist)

```
NodeList myList = (NodeList) myXPath.evaluate(myExpr, myXmlDoc,
                                              XPathConstants.NODESET);
```



COMPILAZIONE DELL'ESPRESSIONE E VALUTAZIONE

- `myXPath.compile(myExpr)` effettua una pre-compilazione dell'espressione, creando in uscita una XPathExpression

- `myXPath.evaluate(myExpr, myXmlDoc, QName)` effettua implicitamente la compilazione dell'espressione, e poi effettua la valutazione, restituendo un Object

- Quando occorre richiamare più volte la valutazione di un'espressione conviene prima compilarla (`compile`) e poi usare più volte il metodo `evaluate` sull'espressione compilata

- Il terzo parametro di evaluate deve essere uno dei seguenti: XPathConstants.{NUMBER | STRING | BOOLEAN | NODE | NODE_SET}

ESERCIZIO

(JavaXPath.java)

- Creare un software minimale che apre un file XML e ne seleziona alcuni nodi utilizzando la sintassi XPath:
  - Il titolo del brano (selezione del contenuto testuale del nodo /ieee1599/general/description/main_title)
  - Il nome del primo autore (selezione del primo nodo /ieee1599/general/description/author e lettura del contenuto testuale di tale nodo)
  - La lista di tutte le descrizioni dei file correlati (selezione di tutti i nodi related_file e lettura dei valori dell'attributo description di tali nodi)

ESERCIZIO

(XPathAnalysis.java)

- Si crei un software che estrapola dati di interesse musicale da un file in formato IEEE 1599:
  - Battuta per battuta il software calcola la nota di durata più lunga escludendo le pause, e restituendo il nome dello strumento (part) che la esegue