Università degli Studi di Bologna

Facoltà di Ingegneria

Tecnologie Web T

A.A. 2019 – 2020

Esercitazione 2

XML, DTD, XSD, Parser SAX/DOM

Agenda

- Creazione di documenti XML
- Creazione di documenti XML Schema

- Parsing e validazione di file XML con Java
  - JAXP per parsificare e validare documenti XML tramite XML Schema
  - SAX vs. DOM
  - well-formed XML vs. valid XML

- Uso dei parser per navigare e modificare documenti XML

Specifica di “Lettera”

- Si progetti una grammatica XML Schema in grado di modellare il contenuto informativo di una “**Lettera**”
- Il contenuto della Lettera è soggetto alle seguenti specifiche:
  - ogni **lettera** è caratterizzata da un mittente, una data, un destinatario, un oggetto, una forma cortese di saluto, un corpo, una chiusura, una firma
  - il corpo della lettera è costituito da almeno un paragrafo
- Si scriva inoltre un documento XML valido per le grammatiche al punto precedente

Specifica di “Address List”

- Si progetti una grammatica XML Schema per la gestione di “**Address List**”
- Address List deve rispettare le seguenti specifiche:
  - ciascun **address list** contiene almeno una informazione
  - ogni informazione include: un nome, zero o più indirizzi, zero o più numeri di telefono, zero o più indirizzi email, eventualmente la nazionalità e zero o più note
  - il nome contiene: un nome proprio, zero o più secondi nomi e il cognome
  - un indirizzo è composto da: almeno una via, un indirizzo postale, eventualmente la provincia e sicuramente lo stato
- Si scriva inoltre un file XML valido per le grammatiche al punto precedente

RSS 0.92 (1)

- Si progetti una grammatica XML Schema per i feed Really Simple Syndication (RSS) versione 0.92
  - si veda come esempio di documento XML il file *RSS-0.92-example-gratefulDead.xml* presente sul sito del corso

- In questo esercizio omettiamo le specifiche dettagliate di RSS 0.92 (per altro disponibili online) e ci limitiamo a descrivere elementi ed attributi che compongono un feed

- L'esempio è più complesso dei precedenti e mira a mostrare XML Schema possano essere applicati in situazioni reali nella modellazione di documenti con una struttura complessa

RSS 0.92 (2)

- Ogni feed RSS ha un canale (**channel**)
- Inoltre ogni feed ha un attributo versione (**version**) il cui valore è (nel nostro esempio) 0.92

- Il canale può inoltre avere zero o più sotto-elementi scelti fra:
  - title
  - description
  - link
  - language (opzionale)
  - item (almeno un item)
  - rating (opzionale)
  - image (opzionale)
  - textInput (opzionale)
  - copyright (opzionale)
  - pubDate (opzionale)
  - lastBuildDate (opzionale)
  - docs (opzionale)
  - managingEditor (opzionale)
  - webMaster (opzionale)
  - skipHours (opzionale)
  - skipDays (opzionale)
  - cloud (opzionale)

RSS 0.92 (3)

- Un’immagine (**image**) può a sua volta avere zero o più sotto-elementi scelti fra:
  - title
  - url
  - link
  - width (opzionale)
  - height (opzionale)
  - description (opzionale)
- Una **item** può avere zero o più sotto-elementi scelti fra:
  - title (opzionale)
  - link (opzionale)
  - description (opzionale)
  - source (opzionale)
  - enclosure (opzionale)
  - category (opzionale)

RSS 0.92 (4)

- Sia l'elemento **source**, sia l'elemento **enclosure**, qualora siano presenti in un feed RSS, devono avere un attributo **url**
- Inoltre, l'elemento **enclosure** deve anche avere due ulteriori attributi: **lenght** e **type**
- L'elemento **category**, se presente in un feed RSS, ha un attributo opzionale **domain**
- L'elemento **textInput**, se presente in un feed RSS, può avere zero o più sotto-elementi scelti fra:
  - title
  - description
  - name
  - link

RSS 0.92 (5)

- L'elemento **cloud**, qualora presente in un feed RSS, deve avere i seguenti attributi (tutti obbligatori):
  - domain
  - port
  - path
  - registerProcedure
  - protocol

- Infine gli elementi **skipDays** e **skipHours**, se presenti in un feed, devono avere rispettivamente almeno un sotto-elemento **day** ed un sotto-elemento **hour**

JAXP: Java API for XML Processing

- Per parsificare e validare un documento XML occorre disporre di:
  - un documento XML
  - un parser che ne navighi la struttura, e.g., SAX e DOM parser
  - una specifica grammatica DTD o un XML Schema da rispettare
  - un gestore di errori che sappia distinguere tra
    - **warning**: errori secondari, solitamente ignorati; ad esempio, esiste un elemento con nome XMLDocument, teoricamente vietato in quanto W3C vieta l'uso di XML come prefisso del nome degli elementi
    - **errors**: errori importanti ma che non pregiudicano la corretta parsificazione del documento XML; ad esempio, documento **well-formed ma non valido**
    - **fatal errors**: errori molto gravi che impediscono la corretta parsificazione del documento XML; ad esempio, documento **non well-formed**

      N.B. Per convenienza, si consiglia di estendere l'oggetto *DefaultHandler*, gestendo diversamente errori relativi a "well-formed" o "valid"

JAXP per validare documenti XML (1)

- Come specificare DTD o XML Schema da utilizzare per la validazione?
  - http://jaxp.java.net/
  - http://jaxp.java.net/docs/spec/html/
  - http://xerces.apache.org/xerces2-j/features.html
- È possibile specificare la grammatica da utilizzare per la validazione in due modi:
  - Tramite link diretto all'interno del documento XML
    - DTD: *<!DOCTYPE Nome_root SYSTEM "fileName.dtd">*
    - XSD: *<Nome_root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="fileName.xsd">*
  - Impostando una proprietà del parser (utilizzabile solo nel caso di XSD)
    - SAX: *saxParser.setProperty("http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation", "fileName.xsd");*
    - DOM: *documentBuilderFactory.setAttribute("http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation", "fileName.xsd");*

JAXP per validare documenti XML (2)

- Per specificare l'uso di XML Schema (invece che DTD)
  - SAX: *xmlReader.setFeature("http://apache.org/xml/features/validation/schema", true)*;
  - DOM: *dbf.setFeature("http://apache.org/xml/features/validation/schema",true)*;
- Attenzione, ricordarsi di abilitare il namespace
  - *parserFactory.setNamespaceAware(true)*;
- Inoltre, ricordarsi di abilitare la validazione del documento:
  - *parserFactory.setValidating(true);*
- DOM: ignorare whitespace tra un tag e l'altro, altrimenti nodi di testo "fittizi"
  - *dbf.setFeature("http://apache.org/xml/features/dom/include-ignorable-whitespace", false)*;

SAX: Simple API for XML

- **Event-based XML parser**: passi generali per registrare ed attivare opportuni gestori al parser SAX

- Creare un **SAXParserFactory**
  - è necessario creare un SAXParserFactory da cui ottenere una classe che estenda la classe astratta SAXParser
  - ricordarsi di invocare setValidating(true) affinché il parser ottenuto faccia anche la validazione
- Ottenere un oggetto che implementi l'interfaccia **XMLReader**
- Agganciare opportuni listener al lettore XML e poi parsificare
  - **ContentHandler**: gestore eventi di base generati dal parser
  - DTDHandler: gestore eventi legati al DTD
  - **ErrorHandler**: metodi per gestire gli errori ed i warning nell'elaborazione di un documento
  - EntityResolver: metodi per personalizzare l'elaborazione di riferimenti ad entità esterne

DOM: Document Object Model

- Passi generali da seguire per parsificare un documento XML ed **ottenere un documento DOM**

- Creare un **DocumentBuilderFactory**
  - è necessario creare un DocumentBuilderFactory da cui ottenere una classe che estenda la classe astratta DocumentBuilder
  - ricordarsi di invocare setValidating(true) affinché il parser ottenuto faccia anche la validazione
- Ottenere un oggetto che estenda la classe astratta **DocumentBuilder**
  - l'oggetto che effettua parsificazione e (opzionalmente) validazione
- Realizzare e registrare un **ErrorHandler**
- Parsificare il documento per ottenere un documento DOM
- Utilizzare opportunamente il documento DOM ottenuto

Creare un ErrorHandler (1)

- Esempio di classe che estende DefaultHandler

```
public class ErrorChecker extends DefaultHandler {
    public ErrorChecker (){ }
    public void error (SAXParseException e) {
        System.out.println("Parsing error: "+e.getMessage());
    }
    public void warning (SAXParseException e) {
        System.out.println("Parsing problem: "+e.getMessage());
    }
    public void fatalError (SAXParseException e) {
        System.out.println("Parsing error: "+e.getMessage());
        System.out.println("Cannot continue.");
        System.exit(1);
    }
}
```

Creare un ErrorHandler (2)

- Esempio di uso della classe ErrorChecker mediante parser DOM

```
...
try {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setValidating(true);
    DocumentBuilder db = dbf.newDocumentBuilder();

    ErrorChecker errors = new ErrorChecker();
    db.setErrorHandler(errors);

    Document doc = db.parse(docFile);
}
catch (Exception e) { System.out.print("Parsing problem."); }
...
```

E ora a voi

- **Realizzare un progetto Java** che parsifichi e validi i documenti XML realizzati precedentemente, sia con SAX che con DOM
  - per RSS 0.92, scaricare dal sito del corso (o da Internet) un file XML RSS 0.92
  - negli altri casi utilizzare i documenti XML e XSD realizzati
- Modificare i documenti XML/XSD creati in modo tale da **testare il gestore di errori**
  - come generare/gestire un warning/error/fatal error?
- **Utilizzare i parser SAX e DOM a piacere**, ad esempio per
  - contare il numero di "ignorableCharacters" in un documento XML
  - in un documento XML AddressList
    - contare le persone presenti (SAX vs. DOM)
    - contare le persone prima di Mickey Mouse (SAX vs. DOM)
    - il numero di telefono di tutte le persone il cui nome inizia per "Don" (SAX vs. DOM)
    - sostituire/inserire il numero telefonico di una data persona (DOM)

APPENDICE

(a integrazione, anche esempi basati su DTD…)

Da DTD a documento XML: “Ricette”

- Scrivere una descrizione testuale ed un documento XML valido per la grammatica DTD “**Ricette**”

```
<!ELEMENT recipes (recipe+, document_info)>
<!ELEMENT recipe (recipe_head, recipe_body, recipe_footer?)>
<!ELEMENT recipe_head (recipe_name, recipe_author?, meal_type)>
<!ELEMENT recipe_name (#PCDATA)>
<!ELEMENT recipe_author (#PCDATA)>
<!ELEMENT meal_type (#PCDATA)>
<!ELEMENT recipe_body (ingredients, directions)>
<!ELEMENT ingredients (ingredient+)>
<!ELEMENT ingredient (#PCDATA)>
<!ELEMENT directions (direction)+ >
<!ELEMENT direction (#PCDATA)>
<!ELEMENT recipe_footer (serving?, preparation_time?, cooking_time?)>
<!ELEMENT serving (#PCDATA)>
<!ELEMENT preparation_time (#PCDATA)>
<!ELEMENT cooking_time (#PCDATA)>
<!ELEMENT document_info (document_author, date_updated, source)>
<!ELEMENT document_author (#PCDATA)>
<!ELEMENT date_updated (#PCDATA)>
<!ELEMENT source (#PCDATA)>
```

Let’s put it all together…

- A completamento, forniamo la versione di ogni esercizio proposto nell’esercitazione di oggi
  - basato su grammatica XML Schema
- anche con uso di grammatica DTD