

Commenti al Laboratorio 2. XML e DTD con DOM

Error handler

Nello scorso laboratorio ci siamo resi conto che alcuni errori nel documento XML non venivano gestiti nella maniera opportuna dal parser.

Questo accade perché, a parte alcuni errori non recuperabili, la politica del parser è molto lassista e lascia passare la maggior parte degli errori **senza fare niente** (neanche una stampa a video).

Tuttavia, al programmatore è permesso definire un *error handler* per gestire questi errori (vedi dispense lezione, *XML – Interfaccia di programmazione DOM, slides 21 – 23*). Attraverso l'uso dell'*error handler* si possono stampare a video i warning e gli errori per rendersi conto di cosa non va nel documento XML.

L'*error handler* è una classe che estende l'interfaccia `org.xml.sax.ErrorHandler` e che deve essere composta da tre metodi il cui nome è autoesplicativo (`error`, `fatalError` e `warning`).

L'*error handler* deve essere settato dal metodo `setErrorHandler` della classe `DocumentBuilder` **prima** del parsing.

Come ulteriore esercizio, si consiglia l'aggiunta di un semplice error handler al codice del Laboratorio 2. Si consiglia inoltre di effettuare alcuni tentativi, modificando il file XML, per controllare quali errori vengono effettivamente catturati dall'error handler.

Classi wrapper

In Java, per ogni tipo primitivo esiste una classe cosiddetta *wrapper* che discende da `Object`. Per esempio, per gli `int` esiste la classe `Integer`. Per convertire una stringa in un numero (es. il contenuto di un nodo/attributo) è possibile usare il metodo statico `parseXXX(String)` di ogni wrapper (es. `Integer.parseInt()` per gli interi).

L'unico requisito per il funzionamento è che la stringa da parserizzare contenga effettivamente una rappresentazione testuale del tipo primitivo che si vuole ottenere e non altri caratteri (ad esempio, `Integer.parseInt("3 km")` non dà un intero di valore 3 ma lancia semplicemente una eccezione, così come `Integer.parseInt("3.15")`).

Laboratorio 3. XML con SAX

Nel laboratorio precedente abbiamo visto un'applicazione dell'interfaccia di programmazione DOM per elaborare un documento XML. In questo laboratorio, invece, vedremo il funzionamento e l'utilizzo dell'interfaccia di programmazione SAX. In particolare partendo da un DTD e da un documento XML valido scriveremo un semplice programma Java per estrarne alcune informazioni usando SAX.

Il DTD ed il file XML di riferimento possono essere scaricati da `\\servlab\SARIdocs`. Il file XML può essere modificato a piacere nel rispetto della DTD.

Analizzando il documento DTD si giunge alle seguenti conclusioni:

- ◆ L'itinerario è suddiviso in uno o più giorni e ogni giorno è suddiviso in una o più tratte.
- ◆ Una tratta ha una partenza, un arrivo ed eventuali soste.
- ◆ All'itinerario è associato il prezzo del carburante (al litro) ed il consumo medio (in Km/l) validi solamente per le tratte in automobile.
- ◆ Ogni tratta è caratterizzata da un mezzo di locomozione, dalla durata e dalla lunghezza in Km.
- ◆ Ogni sosta è caratterizzata da una durata e da una descrizione.

L'obiettivo del nostro parser SAX è visualizzare partenza e arrivo di ogni tratta, le varie soste e la spesa giornaliera e totale per il carburante (solo per le tratte in automobile).

Per far ciò utilizzeremo l'implementazione dell'interfaccia di programmazione SAX fornita con la distribuzione di *Java 2 Standard Edition* e la otterremo, in maniera molto simile al laboratorio precedente (design pattern factory), sfruttando la *JAXP: Java API for XML Processing* (package `javax.xml.parsers`).

L'elaborazione di un documento XML sfruttando l'interfaccia SAX è di tipo *event-driven*. Questo significa che durante l'elaborazione del documento XML il parser effettua chiamate a delle funzioni (*callback*) al verificarsi di certe condizioni. Infatti il parser, oltre al documento da elaborare, richiede in ingresso alcune istanze di classe, costruite secondo le specifiche SAX, sulle quali richiamare i metodi. L'interfaccia di programmazione SAX comprende le interfacce `DocumentHandler`, `DTDHandler`, `EntityResolver`, `ErrorHandler` (tutte contenute nel package `org.xml.sax`) che definiscono i metodi invocati dal parser e che devono essere implementate dalle classi adibite al trattamento degli eventi. Le più importanti sono `DocumentHandler` e `ErrorHandler` che contengono, rispettivamente, i metodi relativi al contenuto del documento XML e i metodi richiamati in caso di errore.

La classe `DefaultHandler` (package `org.xml.sax.helpers`) fornisce un'implementazione standard di tutte le interfacce elencate sopra ed ha il compito di facilitare il lavoro dello sviluppatore. Infatti quest'ultimo, estendendo questa classe, può concentrarsi solamente nella implementazione (*overriding*) dei metodi strettamente necessari ai suoi scopi.

Suggerimenti

Riassumendo, per realizzare il nostro esercizio le operazioni da svolgere sono le seguenti:

1. Scrivete la classe `CompRoute` che estende `DefaultHandler`. I metodi sui cui fare particolare attenzione sono `startElement`, `endElement` e `character` (vedi la documentazione sul server o online [qui](#)).
2. Ottenete un'istanza della classe `SAXParseFactory` e impostare alcuni parametri.
3. Ottenete dalla classe factory un'istanza della classe `SAXParser`.
4. Richiamate il metodo `parse(String, DefaultHandler)` della classe `SAXParser` con il percorso del documento XML da elaborare e una istanza di `CompRoute` come parametri.
5. Notate che `DefaultHandler` implementa `ErrorHandler`. È quindi necessario andare a definire i tre metodi `error`, `fatalError` e `warning` descritti nei commenti al Laboratorio 2.
6. Analizzate bene il codice che avete prodotto e confrontatelo con quello realizzato per il DOM. Ci sono alcune somiglianze (il design pattern basato sulla factory, il meccanismo di gestione degli errori, ...) e molte differenze. Provate a fare alcuni esempi di applicazioni in cui un modello è migliore dell'altro.