JAVA ET XML :

PARSING AVEC SAX

SEMAINE 2

UNIVERSITÉ CAEN NORMANDIE

Complément de cours et TP

Notions abordées :
- révisions sur la gestion des exceptions (vues en L2)
- révisions sur l'implémentation d'interfaces ou l'héritage de classes (abstraites). - première utilisation du pattern Factory
- première utilisation du pattern Strategy

Ces 2 patterns seront intégralement mis en pratique dans quelques semaines. Nous nous contentons aujourd'hui de les utiliser en tant que clients, pas de les mettre en place.

1. Présentation

Java dispose de packages permettant de gérer XML. Les classes et interfaces proposées permettent de parser facilement des données au format XML, avec l'alternative suivante :
SAX = Simple Api for Xml
DOM = Document Object Model
Ces deux API ne sont pas redondantes, même si l'une ou l'autre peut être employée pour faire quelque traitement XML que ce soit, car elles impliquent deux démarches très différentes :

- retenons simplement que l'approche DOM consiste à créer en mémoire un équivalent de toute la structure portée par le document XML, sous forme d'arbre. La totalité des données est donc présente en mémoire quel que soit le traitement envisagé.
- SAX, pour sa part, propose un traitement sous forme de flux où rien n'est retenu, a priori (mais libre à vous de faire retenir par votre programme ce que bon vous semble), des données du document. Simplement, cette API met en jeu un certain nombre de "réflexes" lorsque des événements se produisent, ces événements étant notamment le fait que l'on trouve une balise d'ouverture ou de fermeture d'un élément XML. Ici les données sont traitées sous forme de flux, avec toutes les conséquences que cela implique (avantages et inconvénients).

Résumé :
DOM permet d'avoir en mémoire la totalité de l'information XML, sous forme d'arbre calculé automatiquement par le parseur dédié. Votre application travaille ensuite à sa guise sur cet arbre.
SAX permet de considérer le document XML comme un flux que l'on va parser progressivement. Le parseur repère automatiquement les différents ingrédients XML, et invoque

des méthodes (sortes de "reflexes") lorsqu'il trouve une balise d'ouverture, une balise de fermeture, etc.

Aujourd'hui, nous allons travailler avec SAX uniquement.

2. SAX, ou XML vu comme un flux.

1. Quand choisir Sax ?

Imaginons que l'on veuille faire un traitement très simple sur un document volumineux. À l'extrême, imaginons que ce traitement soit aussi simple que récupérer les noms d'un certain document xml, c'est-à-dire "Robert Durand" dans "<nom>Robert Durand</nom>" ; imaginons par ailleurs que le fichier XML en question fasse plusieurs MO, il serait alors inefficace de mettre la totalité du contenu en mémoire vive. SAX n'impose rien de tel : les éléments du document XML vont être gérés, puis oubliés, les uns après les autres. C'est le traitement en flux.
Nous venons de mentionner un exemple type où SAX est la bonne solution[1]. Souvent, ce sera le contraire, et enfin d'autres fois les choses seront moins tranchées, et le choix entre SAX et DOM sera alors moins immédiat.

2. SAX en Java, qu'est-ce ?

L'implémentation de SAX en Java consiste en un ensemble d'interfaces et de classes, le type essentiel étant celui qui spécifie les méthodes invoquées lors de l'analyse du document (ContentHandler).

2.1 Point de départ : la classe abstraite SAXParser.

Elle permet de lancer l'analyse d'un document. Si l'on dispose d'une instance de cette classe, par exemple saxParser, on peut parser le fichier "monFichier.xml" en utilisant le ContentHandler handler via la méthode suivante :

```
saxParser.parse("monFichier.xml", handler);
```

Mais cette classe étant abstraite, comment en obtenir une instance ?

2.2 SAXParserFactory

Pour cela, on utilise une instance de la classe SAXParserFactory, littéralement une usine à parseurs, de la façon suivante :

---

[1] Cet exemple est volontairement tendancieux. Nous aurions pu choisir l'exemple d'un document XML de taille raisonnable, et dont on voudrait en mémoire un équivalent exhaustif. DOM s'imposerait alors, bien que, dans ce cas encore, on puisse le faire nous même via SAX. En fait, il n'y a pas lieu de trancher entre DOM et SAX dans le cas général, mais de considérer avantages et inconvénients de chacun d'entre eux compte tenu de l'application envisagée, et de ses propres habitudes.

```
SAXParser saxParser = factory.newSAXParser();
```

Mais comment obtenir une instance telle que celle référencée par factory, vu que SAXParserFactory est elle-même abstraite (abstract factory) ? Cette dernière, certes abstraite, possède néanmoins une méthode statique (factory method), donc accessible à partir de la classe, sans qu'il y ait besoin de disposer d'une instance. Voici comment procéder :

```
SAXParserFactory factory = SAXParserFactory.newInstance();
```

2.3 Récapitulation : lancement d'un parseur SAX

```
import java.io.*;
import java.util.*;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.SAXParser;


(…)

// On demande la création d'une instance d'usine à parseurs SAX :
SAXParserFactory factory = SAXParserFactory.newInstance();
    try
        {
        // puis on obtient une instance de parseur à partir de cette usine.
        SAXParser saxParser = factory.newSAXParser();
        // Enfin, on lance le parsing :
        saxParser.parse("nomDuFichier.xml", handler);
         } catch …
```

La présence du bloc try est nécessaire, car, vous le constaterez le cas échéant à la compilation, chacune des deux méthodes peut lever des exceptions (cf. API).

Ce code en gras sera notre canevas de base pour toute la séance de TP…

2.4 L'interface ContentHandler

La lecture d'un document xml via un parseur SAX génère l'appel d'une méthode spécifique d'une certaine instance (le Handler qu'on lui a passé en ce sens) lorsque les "événements" suivants se produisent :

- ouverture du document
- fermeture du document
- ouverture d'un élément (balise d'ouverture)
- fermeture d'un élément (balise de fermeture)
- accès au contenu de l'élément

Ce sont les appels à ces méthodes que nous avons précédemment appelés "réflexes". Le principe est proche de la gestion d'événements dans une interface graphique.
Le handler en question est une instance de DefaultHandler, qui implémente, entre autres, l'interface ContentHandler dont les méthodes nous intéressent ici (et qui implémente aussi, par ex., DTDHandler, que nous n'aborderons pas dans le cadre de cet enseignement). Par contre, par défaut, les méthodes de DefaultHandler ne font rien. Il va falloir en redéfinir les méthodes qui nous intéressent, et qui sont pour l'essentiel :

```
void startDocument()
void endDocument()
void startElement(String namespaceURI, String localName, String qName, Attributes atts)
```

- qName = le nom de la balise
- atts = l'ensemble des attributs et leur valeur associée, sous forme d'une instance de l'interface Attributes. cf. la doc de l'API pour manipuler ces attributs.

```
void endElement(String namespaceURI, String localName, String qName)
void Characters(char[] ch, int indiceDebut, int longueur)
```

- déclenché lors de l'accès au contenu de l'élément. ch est un tableau de caractères contenant l'ensemble du document. La partie qui nous concerne est celle comprise entre les deux indices fournis.

3. Comment créer un parsing avec SAX ?

Tout simplement en implémentant le morceau de code fourni en 2.3. Le parsing sera lancé sur le fichier passé en paramètre. Il nous manque juste le deuxième paramètre, c'est-à-dire le DefaultHandler. Alors il suffit de passer un new DefaultHandler() : on obtient un parsing de notre document, géré entièrement par Java, mais qui, par défaut, ne fait encore rien.

Ce qu'il faut bien comprendre ici, c'est que la matière grise d'un parsing SAX est contenue dans la définition des méthodes de ContentHandler, donc dans la redéfinition de certaines méthodes de DefaultHandler. Par défaut, ces méthodes ne font rien, puisque cette classe est juste une « coquille vide ».
Au contraire, le lancement d'un parsing utilisant DOM, même sans rien programmer d'autre que les quelques lignes nécessaires à son lancement, nous donne en mémoire un arbre complet. Ayez en tête ces deux philosophies très différentes.

Il nous faut donc, à partir du canevas de base, décliner différentes versions de DefaultHandler suivant l'application envisagée (on créera, à chaque fois, une sous-classe particulière, dédiée à un certain traitement).

3. Travail à réaliser

Nous allons travailler de façon incrémentale, c'est-à-dire que certaines choses écrites pour un exercice n pourront généralement être réutilisées pour l'exercice n+1.
Les tests seront faits au moins avec le document "juicers.txt" situé sur : *http://mathet.free.fr/javaxml/juicers.xml*
Vous pouvez bien sûr écrire vos propres fichiers xml pour affiner vos tests.

1. SaxEcho.java

Ecrire un programme qui permette de parser un document XML et d'en produire un "écho" sur la sortie standard (via System.out). Dans cette première mouture, on se contentera d'un écho réduit aux seules balises d'ouverture et de fermeture, en "oubliant" les attributs, et le contenu des éléments.

2. SaxEcho2.java

Idem mais en n'omettant pas, cette fois, ni les attributs, ni le contenu.

3. SaxMontreChemin.java

Il s'agit à présent d'afficher, à tout moment de l'analyse où l'on trouve une balise d'ouverture, le "chemin" menant de la racine de l'arbre jusqu'à cet élément.

4. SaxSelectionne.java

Il s'agit à présent d'afficher tous les éléments correspondant à un certain chemin spécifié par rapport à la racine. Vous ferez le test avec le chemin : /juicers/juicer/cost

5. SaxSelectionne2.java

Idem sauf que l'on cherche les éléments correspondant à une certaine **fin** de chemin. Vous ferez le test avec les fins de chemin : cost puis avec juicer/cost.

6. SaxSelectionne3.java

Il s'agit pour cette dernière application de calculer le total de tous les coûts (élément <cost>) des "juicer", et uniquement des "juicer" !
On partira de la base fournie par SaxSelectionne2 pour ne retenir que les cost qui concernent des juicer. Mais attention ! Parfois, le prix est donné dans deux devises, parfois uniquement dans une devise, et dans tous les cas au moins en USD (dollar US).
Il ne faut donc comptabiliser que les coûts exprimés en USD. Il vous faut donc jouer sur les attributs et leur valeur.