TP n°7

Analyses de fichiers XML
OpenStreetMap

1 Introduction à OpenStreetMap

> *OpenStreetMap est un projet qui a pour but de créer des cartes libres du monde sous licence libre, en utilisant le système GPS et d'autres données libres.*
>
> — `http://fr.wikipedia.org/wiki/OpenStreetMap`

Dans l'esprit des wikis, les utilisateurs d'OpenStreetMap souhaitant contribuer peuvent fournir des données afin de former progressivement une carte numérique mondiale de plus en plus précise. La façon la plus simple de contribuer est d'enregistrer des itinéraires parcourus dans le monde réel avec un récepteur GPS et, ensuite, de restituer les *traces GPS* correspondantes sur le serveur de données d'OpenStreetMap.

Les données de la carte OpenStreetMap sont encodés dans un dialecte XML. Les informations complètes sur la syntaxe du format XML OpenStreetMap sont disponible sur le site [**?**]. La DTD complète du format XML est disponible en [**?**]. Vous trouverez ci-dessous une introduction aux concepts de base et aux éléments fondamentaux du format, nécessaires pour ce sujet de TP.

1.1 Map

Un fragment de carte OpenStreetMap est un document XML ayant comme racine un élément `<osm>`, puis un certain nombre de sous-éléments tels que `<node>`, `<way>` ou encore `<relation>`.

Example 1 (squelette de carte en format XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="...">
  <!-- nodes -->
  <node ...> ... </node>
  ...
  <!-- ways -->
  <way ...> ... </way>
  ...
  <!-- relations -->
  <relation ...> ... </relation>
  ...
</osm>
```

1.2 Propriétés

Il existe deux manières d'associer des propriétés aux différents éléments d'un document XML OpenStreetMap.

Attributs. Un certain nombre de propriétés sont directement des attributs XML de l'élément. Pour chaque élément, la DTD précise les attributs possibles et ceux obligatoires. Vous rencontrerez fréquement des attributs communs tels que `id`, `user`, `uid`, `timestamp`, `visible`, `version`, `changeset`. Parmi ceux-ci, l'attribut `id`associe à l'élément un identifiant unique utilisé pour référencer cet élément ailleurs dans la carte.

Tags. Par ailleurs, certains éléments XML d'une carte peuvent aussi avoir un certain nombre d'éléments fils `<tag>`. Les attributs de ces éléments fils vont permettre d'associer des propriétés supplémentaires *clé=valeur* à l'élément XML père. Pour cela, un élément XML `<tag>` doit forcément posséder un attribut `k="clé"` et un attribut `v="valeur"`.

Pour ce TP on s'intéresse essentiellement aux tags suivants :

```
  <tag k="amenity" v="restaurant"/>
  <tag k="cuisine" v="turkish"/>
  <tag k="name" v="Welcome To Istanbul"/>
```

1.3 Nodes

En général, une carte contient plusieurs noeuds (*nodes*). Chaque noeud (élément XML `<node>`) correspond à un point géographique et est associé à une valeur de latitude (attribut `lat`) et à une valeur de longitude (attribut `lon`). Un noeud possède aussi obligatoirement un attribut `id`.

Example 2 (noeud en format XML)

```
<node id="1109569673" visible="true" version="2" changeset="28845481"
 timestamp="2015-02-14T16:25:06Z" user="Coucouf" uid="62548"
 lat="48.8526210" lon="2.3463757">
  <tag k="amenity" v="restaurant"/>
  <tag k="cuisine" v="turkish"/>
  <tag k="name" v="Welcome To Istanbul"/>
</node>
```

1.4 Ways, Areas, Relations

En général, une carte contient aussi plusieurs chemins, aires et relations . Un chemin (élément XML `<way>`) est une séquence ordonnée de noeuds (au minimum deux). Chaque chemin donne une description d'une caractéristique linéaire du paysage, comme par exemple une rue, une fleuve, un chemin de fer, etc.

Example 3 (chemin en format XML)

```
<way id="4227132" visible="true" version="4" changeset="4465931"
timestamp="2010-04-19T08:50:21Z" user="STA" uid="18744">
 <nd ref="25183373"/>
 <nd ref="701743404"/>
 <nd ref="25184251"/>
 <tag k="highway" v="pedestrian"/>
 <tag k="name" v="Rue de l'Hirondelle"/>
</way>
```

L'attribut `id` donne un identifiant unique à chaque chemin. Les fils `<nd>` donnent les noeuds qui forment le chemin : pour chaque `<nd ref="`v`">`, il doit exister dans la carte un noeud correspondant `<node id="`v`">`.

Une aire est un cas particulier de chemins, où le premier point et le dernier sont identiques. La plupart des objets "fermés" qu'on trouve habituellement dans des cartes géographiques (bâtiments, lacs, parcs, aéroports, etc.) sont représentés par des aires.

Enfin, une relation (élément XML `<relation>`) permette de grouper plusieurs chemins et/ou noeuds dans une même entité. Une relation permet par exemple de représenter une zone dont la frontière ne peut pas être décrite à l'aide d'un unique chemin. Un bâtiment peut ainsi avoir des cours intérieures.

2 Exercices

Sur Didel vous trouverez le fichier XML OpenStreetMap `map.osm.xml` décrivant un fragment de carte autour de la Place Saint-Michel à Paris. Le but de ce TP est de construire un arbre de syntaxe abstraite du document `map.osm.xml` afin d'extraire la liste des restaurants presents dans la carte.

Voici une grammaire d'une version simplifiée du langage XML pour se focaliser sur les exercices de ce TP. Les symboles terminals sont : `prologue` `<`, `>`, `/`, `mot`, `=`, `string`, \$.

$$
\begin{aligned}
S &\mapsto \texttt{prologue}\ \textit{Arbre}\ \$ \\
\textit{Arbre} &\mapsto \texttt{<}\ \texttt{mot}\ \textit{Attributs}\ \texttt{/>} \mid \texttt{<}\ \texttt{mot}\ \textit{Attributs}\ \texttt{/>}\ \textit{Elements}\ \texttt{</mot>} \\
\textit{Attributs} &\mapsto \epsilon \mid \texttt{mot = string}\ \textit{Attributs} \\
\textit{Elements} &\mapsto \epsilon \mid \textit{Arbre}\ \textit{Elements}
\end{aligned}
$$

Exercice 1 *Dans un premier temps, on se propose d'écrire, à l'aide de* `JFlex`*, un analyseur syntaxique qui transforme un fichier OpenStreetMap dans un flot des jetons associés aux symboles de la grammaire ci-dessus. Vous trouverez sur Didel et compléterez le fichier* osm.flex *en utilisant les fichiers* Sym.java *et* Token.java*. En particulier, il faut que les jetons associés à* `mot` *et* `string` *soient de type* `ValuedToken`*, mémorisant leur chaine de caractères.*

Exercice 2 *Dans le fichier* `AbstractSyntax.java` *vous trouverez une implémentation très simple des arbres de syntaxe abstraite associés au langage définit par la grammaire ci-dessous. Le fichier a deux seules classes (*Arbre *et* Attribute*) et utilise les listes de* `java.util`*.*

Compléter le fichier Parser.java *afin de définir un analyseur sémantique transformant un flot de jetons qui satisfait la grammaire ci-dessus, dans un arbre de syntaxe abstraite .*

Malheureusement, la grammaire donnée dessus n'est pas LL(1) (pourquoi ?). Une grammaire LL(1) équivalente est proposée ci-dessous :

$$S \mapsto \texttt{prologue} < A\ \$ \qquad A \mapsto \texttt{mot}\ Atr\ ASuite \qquad Atr \mapsto \epsilon \mid \texttt{mot = string}\ Atr$$
$$ASuite \mapsto \texttt{/>} \mid \texttt{>} <E \qquad E \mapsto \texttt{/ mot >} \mid A<E$$

Dans le fichier Parser.java *vous trouverez déjà l'en-tête des méthodes associées à ces règles, complétez le fichier.*

On vous rappelle qu'un constructeur des listes est `ArrayList()` *et pour ajouter un élément dans une liste on utilise* `boolean add(E e)`.

Exercice 3 *Écrivez une méthode dans la classe* Arbre *qui parcourt l'arbre de syntaxe abstraite construit à l'étape précédente et renvoie la liste des restaurants présents dans le document* `map.osm`

Vous pourrez vérifier que vous trouvez bien le bon nombre de restaurants en comparant votre résultat au nombre d'occurrences du mot restaurant dans le fichier (obtenu par la commande `grep " restaurant " map.osm.xm | wc -l`*).*

Exercice 4 *Écrivez une fonction qui parcourt l'arbre de syntaxe abstraite et prend en entrée un type de restaurant (par exemple "french") et affiche le nombre de restaurants de ce type.*

Exercice 5 *(Bonus) Modifiez vos fichiers afin d'écrire une fonction qui prend en entrée un nom de restaurant et qui parcourt l'arbre de syntaxe abstraite et indique si ce restaurant est présent dans la liste ainsi que sa longitude et sa latitude.*