Java et XML avec JDOM

1 Introduction à XML

XML (eXtented Markup Language) est un langage à *balises* employé pour le stockage et l'échange de données, contrairement au langage HTML qui est destiné à l'affichage via un navigateur web.
L'apprentissage des principales notions se fera à l'aide de l'exemple `jeux.xml` disponible sur le portail (archive `exemples_xml.tgz`). Il s'agit d'un document représentant une base de données de jeux vidéos.

Exercice 1 ***Analyse d'un fichier XML.*** *Importez les documents dans votre espace de travail. Ouvrez le fichier* `jeux.xml` *avec un navigateur web (Mozilla, Konqueror,...). ouvrez également le fichier avec un éditeur de textes. L'affichage par défaut d'un document XML dépend de la version de votre navigateur. Si le contenu du fichier n'apparaît pas, fermez votre fenêtre de navigation et ouvrez le fichier avec un éditeur de textes.*

La première ligne du document `jeu.xml` permet de définir la version de XML ainsi que l'encodage des caractères. Dans l'exemple ci-dessus, on spécifie que l'on utilise la version 1.0 de XML et le jeu de caractère "Latin-1". C'est cette entête que vous rencontrerez le plus souvent. La représentation sous-jacente d'un document XML est un *arbre ordonné* dont les noeuds internes sont les balises et les feuilles le contenu de ces balises (éventuellement vide). Il y a donc une balise particulière, la *racine* du document, qui correspond ici à la balise `<jeux>`.

Remarque 1. Règles à connaitre. *En* XML*, chaque balise doit être fermée (à une balise* `<a>` *doit correspondre une balise* `</a>`*). Cela n'est pas nécessaire en* HTML *(sauf dans sa version étendue). Une balise fermante clot toujours la dernière balise ouverte. Une balise qui ne contient pas de données peut être écrite sous cette forme :*

```
<balise/>
```

Ceci est équivalent à : `<balise></balise>`*. On fait de plus la distinction entre lettres majuscules et minuscules. Par exemple, les balises* `<jeu>` *et* `<Jeu>` *sont différentes.*

Remarque 2. Attributs. *Par souci de concision et de simplicité, nous ne parlerons pas ici des* attributs*. Ils existent cependant, leur usage étant similaire à celui des attributs en* HTML.

2 Manipulation de fichiers XML avec JDOM

Afin de manipuler les documents XML, il est nécessaire d'avoir à sa disposition des APIs efficaces. Il en existe deux principalement : DOM et SAX. Nous ne les décrirons pas ici, car ce n'est pas le but du TP. Il vous suffit de savoir que JDOM, que nous vous présentons ici, est une API écrite en java et pour java, qui passe outre les limitations inhérentes à DOM dont elle s'inspire. L'intérêt principal de JDOM est son extrême simplicité. SAX, quant à elle, s'avère trop limitée quand il s'agit d'effectuer des modifications au sein des documents.

2.1 Installer JDOM

Pour utiliser l'API JDOM, nous vous conseillons de récupérer le fichier `jdom.jar` disponible sur le portail. Copiez-le dans un répertoire quelconque auquel vous avez accès (par exemple : `/home/licence/toto/lib`, en supposant que vous ayez tous les droits sur ce répertoire). Lors de la compilation d'un fichier source JAVA utilisant JDOM (par exemple `Fichier.java`), vous devrez alors spécifier où trouver les classes de l'API JDOM de la façon suivante :

```
javac -classpath .:/home/licence/toto/lib/jdom.jar:$CLASSPATH Fichier.java
```

Nous vous incitons vivement à consulter la documentation de l'API JDOM afin de vous familiariser avec les différentes classes et méthodes. Celle-ci est également sur le portail. Enfin ppour les besoins du TP téléchargez également les fichiers exemples associés.

Remarque 3. Important. *JDOM est en "Java 1.2", ce qui implique que les méthodes manipulant des* `List` *par exemple fonctionnent sous le modèle de Java 1.2 et ne sont donc pas typées, il faudra donc faire de casts explicites pour obtenir les éléments de la liste.*

2.2 Ecrire des documents XML avec Java et JDOM

On désire stocker un nombre complexe dans un fichier `XML`. L'exemple 2.2 nous présente un format pour représenter le nombre $1 + 3i$. L'élément racine `<complexe>` est ainsi nommé pour donner une indication pertinente sur le contenu du fichier, mais gardons à l'esprit que le nommage des différentes balises est purement arbitraire. Les sous-éléments `<re>` et `<im>` sont respectivement associés aux parties réelles et imaginaires du nombre. **Exemple 1 . Représentation d'un complexe**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<complexe>
   <re>1.0</re>
   <im>3.0</im>
</complexe>
```

Nous allons commencer par écrire un programme qui crée le document `XML` de l'exemple 2.2 et affiche celui-ci sur la sortie standard. Le programme le réalisants'appelle `Complexe.java`.

Exercice 2 ***Analyse du programme.*** *Importez le fichier* `Complexe.java` *dans votre espace de travail et ouvrez-le avec votre éditeur favori. Ce programme est relativement court. Analysez-le ! Vous devriez facilement pouvoir le comprendre grâce aux commentaires...*

Exercice 3 ***A vous de jouer...*** *Inspirez-vous du fichier que vous venez d'analyser pour créer le document* `XML` *de l'exemple 2 en écrivant une classe* `Segment.java`*. Affichez-le sur la sortie standard pour vérifier que celui-ci est correct.*

Exemple 2 . Un segment de droite

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<segment>
   <point>
      <abscisse>0.0</abscisse>
      <ordonnee>0.0</ordonnee>
   </point><point>
      <abscisse>1.0</abscisse>
      <ordonnee>2.0</ordonnee>
   </point>
</segment>
```

Notons que pour l'instant nous n'avons fait que créer des documents `XML` sans les sauvegarder dans des fichiers. Le second argument de la méthode `output` pour une instance de la classe `XMLOutputter` doit être un objet de type `java.io.OutputStream`. En lui passant `System.out`, comme cela a été fait jusqu'à maintenant, nous envoyions le flux de données sur la sortie standard. Pour stocker le document créé dans un fichier, il nous suffit de créer un objet de type `java.io.FileOutputStream` et de passer celui en paramètre.

Exercice 4 *Modifiez le programme de l'exercice précédent afin de sauvegarder le document dans un fichier* `segment.xml`*. Ouvrez celui-ci afin de vérifier le bon déroulement des opérations.*

2.3 Lire des documents XML avec JDOM

Il est possible de lire des documents `XML` préexistants à partir de fichiers ou d'autres types de flux d'entrée. JDOM, cependant, n'intègre pas son propre analyseur de syntaxe (*parser*) `XML`. Nous utiliserons donc celui de SAX – sans avoir besoin d'être familier avec SAX – pour effectuer cette tâche. L'exemple suivant illustre la méthodologie à suivre. Il s'agit d'un programme qui lit un fichier XML dont le nom est passé en argument de la ligne de commande et instancie un objet de type `Document` représentant le fichier :

```
import org.jdom.* ;
import org.jdom.input.* ;
import java.io.IOException ;

public class Read
{
  public static void main (String [] args)
  {
    if (args.length == 0) {
      System.out.println ("Vous devez spécifier un fichier !") ;
      return ;
    }
    /* Création d'un parser SAX */
    SAXBuilder builder = new SAXBuilder () ;
    try {
      /* Instanciation du document correspondant */
      Document doc = builder.build (args[0]) ;
    } catch (JDOMException e) {
      /* Si le fichier est mal formé */
      System.err.println ("Fichier XML mal formé !") ;
    } catch (IOException ee) {
      System.err.println ("Erreur d'I/O...") ;
    }
  }
}
```

On doit tout d'abord créer un analyseur SAX, ce qui est fait grâce à l'appel au constructeur `SAXBuilder`. Ensuite, il suffit d'appeler la méthode `build` de ce parser pour générer le document correspondant (`doc`). S'il y a une erreur de syntaxe, par exemple si une balise n'est pas fermée, alors une exception de type `JDOMException` est levée. Le programme ci-dessus s'appelle `Read.java`.

Exercice 5 *Testez le programme précédent avec un fichier* `XML` *mal formé (omettez par exemple de fermer une balise).*

Exercice 6 *Modifiez* `Read.java` *de manière à afficher le contenu de* `doc` *(en utilisant la méthode* `output` *d'un objet de type* `XMLOutputter` *sur le document obtenu). Testez cette nouvelle version avec le fichier généré lors de l'exercice précédent.*

2.4 Naviguer dans des documents XML avec JDOM

Vous aurez besoin, pour retrouver des informations contenues dans un document `XML`, d'explorer l'arbre sous-jacent. L'exemple suivant permet d'afficher la représentation arborescente d'un fichier `XML` passé en paramètre, sans l'information contenue dans chaque élément (on n'affiche que la structure de l'arbre).

```
import org.jdom.* ;
import org.jdom.input.* ;
import java.io.IOException ;
import java.util.* ;

public class Explore
{
    public static void main (String [] args)
    {
        SAXBuilder builder = new SAXBuilder () ;
        try {
            Document doc = builder.build (args[0]) ;
            Explore ex = new Explore () ;
            System.out.println (ex.XML2String (doc.getRootElement ())) ;
        } catch (JDOMException e) {
            /* Erreur de syntaxe */
        } catch (IOException e) {
            /* Erreur d'I/O */
        }
    }
```

```
    /* Transforme un fichier XML en son arbre sous-jacent
       étiqueté par les noms des éléments */
    public String XML2String (Element current)
    {
        String s1 = current.getName () ;
        List children = current.getChildren () ;
        Iterator it = children.iterator () ;
        if (!it.hasNext ()) {
            /* current est une feuille */
            return s1 ;
        } else {
            current = (Element) it.next () ;
            String s2 = XML2String (current) ;
            while (it.hasNext ()) {
                s2 += "," ;
                current = (Element) it.next () ;
                s2 += XML2String (current) ;
            }
            return s1 + "(" + s2 + ")" ;
        }
    }
}
```

Exercice 7 *Analysez ce programme en vous aidant des commentaires et des explications ci-après, puis testez-le ensuite sur un fichier* XML *syntaxiquement valide (il s'intitule.* `Explore.java`*).*

La méthode `getRootDocument` permet de récupérer l'élément racine du document. La méthode `getName` permet de récupérer l'étiquette d'un élément. Enfin, la méthode `getChildren` pour un élément permet de récupérer ses enfants dans un objet de type `List`.

Exercice 8 *En vous inspirant de cet exemple, écrire un programme qui explore l'arborescence d'un document* XML *et affiche l'étiquette des* feuilles *suivie de leur contenu informatif. Le contenu textuel d'un élément est obtenu par un appel à la méthode* `getText` *(voir documentation en ligne pour plus d'informations). En testant votre programme sur l'exemple 1, vous devrez obtenir un affichage proche de ce qui suit – pas forcément dans le même ordre, puisque vous êtes libre de choisir votre parcours dans l'arbre :*

```
re:1.0
im:3.0
```

Testez également sur `segment.xml` *et* `jeux.xml`*.*

Exercice 9 ***Synthèse.*** *Créez un programme qui prend en paramètre deux noms de fichiers* XML *stockant des nombres complexes et qui affiche le document* XML *(qu'il faudra donc avoir créé) correspondant à la somme des ces deux nombres.*

3 Introduction aux DTD

On peut associer à tout document XML une DTD (Définition de Type de Document) qui impose des contraintes sur celui-ci. Une DTD est en fait une grammaire qui définit un ensemble de documents XML. Une DTD peut être vue comme une *grammaire algébrique* dont les non-terminaux sont les éléments balises et les terminaux sont les différents types de données présents dans le feuillage de l'arborescence du document. Les règles sont similaires à celles d'une grammaire algébrique classique, à ceci près qu'en partie droite d'une règle on spécifie une *expression régulière* sur les éléments (terminaux ou non). L'ensemble des documents définis par une DTD est l'ensemble des *arbres de dérivation* engendrés par la grammaire. Une DTD valide pour le document `jeux.xml` est donnée en exemple 3.

Exemple 3 . Une DTD valide pour `jeux.xml`

```
<!ELEMENT jeux    (jeu*)>
<!ELEMENT jeu     (titre,support+,genre)>
<!ELEMENT titre   (#PCDATA)>
<!ELEMENT support (#PCDATA)>
<!ELEMENT genre   (#PCDATA)>
```

La première ligne dénote un élément `jeux` qui contient un nombre indéfini, potentiellement nul, d'éléments de type `jeu`. La deuxième ligne défini l'élément `jeux` susnommé comme contenant une séquence formée par un élément `titre`, un ou plusieurs éléments `support` et un élément `genre`. Le littéral `#PCDATA` désigne une donnée textuelle. Il y a d'autres éléments par défaut, notamment `ANY` et `EMPTY` qui désignent respectivement tout type de contenu et un contenu vide. La disjonction d'éléments se fait grâce au caractère '—'. Le caractère '?', utilisé comme suffixe d'élément, désigne au plus une occurence de l'élément qui le précède. Pour fixer les choses et faire un parallèle avec la syntaxe classique des expressions régulières, l'expresssion $x = (a+b).c* + d?$ s'écrirait en *syntaxe* DTD comme suit :

```
<!ELEMENT x ((a|b),c*)|d?>
```

Exercice 10 ***Une DTD pour les complexes*** *Ecrivez une DTD valide pour les documents* `XML` *des nombres complexes et sauvegardez-la dans un fichier* `complexe.dtd`.

4 Validation de documents XML

Un document `XML` est *bien formé* s'il respecte la norme 1.0 (balises ouvertes toujours fermées, *etc*...). Il est *valide* s'il est conforme à une DTD donnée. Il existe deux moyens équivalents pour spécifier qu'un document `XML` doit se conformer à une DTD. La manière *interne* consiste à inclure la définition de type en tête du document. La manière *externe* consiste à insérer une référence sur un fichier contenant la DTD (nous nous intéresserons uniquement à celle-ci).

Exercice 11 ***Déclaration externe.*** *Observez les différences entre les fichiers* `jeux_dtd.xml` *et* `jeux.dtd`.

Cette nouvelle ligne, insérée après la spécification de la norme `XML` utilisée, impose que le document qui suit est valide pour la DTD stockée dans le fichier `jeux.dtd` situé dans le même répertoire. Il impose aussi que la racine du document est la balise `jeux` (choix de l'axiome, par comparaison avec les grammaires algébriques). D'une manière générale, la syntaxe d'une déclaration externe est la suivante :

<!DOCTYPE $racine$ SYSTEM "$nom\_de\_fichier\_complet$">

Remarque 4. `SYSTEM`. *Le nom de fichier qui suit le littéral* `SYSTEM` *peut naturellement être une URL valide. C'est d'ailleurs le plus souvent dans ce contexte qu'on l'emploie...*

5 JDOM et DTD

L'ajout d'une DTD à un document XML est très simple à réaliser avec JDOM.
Supposons que `complexe.xml` soit représenté par un objet `doc` de type `Document`. On peut spécifier que `doc` doit se conformer à la DTD `complexe.dtd` en choisissant la balise racine `<complexe>` grâce à la portion de code qui suit :

```
DocType type = new DocType ("complexe","complexe.dtd") ;
doc.setDocType (type) ;
```

Exercice 12 ***Ajout d'une DTD.*** *Utilisez vous connaissances pour créer un programme qui récupère le document* `XML` *contenu dans* `complexe.xml`, *ajoute la DTD* `complexe.dtd` *et stocke le document résultant dans un fichier* `complexe_dtd.xml`. *Testez ce programme et ouvrez le fichier obtenu pour observer la différence avec le fichier initial.*

Pour l'instant, aucune validation n'est effectuée par le parser SAX implémenté dans la classe `SAXBuilder` de JDOM. Pour effectuer une validation lors de l'appel à la méthode `build`, il faut positionner à `true` le tag de validation de l'analyseur, ce qui peut être fait soit lors de l'instanciation de l'objet de type `SAXBuilder`, soit par un appel à la méthode `setValidation` pour celui-ci.

Exercice 13 ***Des points et des droites...*** *Proposez une DTD pour la représentation des droites. Créez deux fichiers* `XML` *conforme à cette DTD (en incluant une référence externe sur celui-ci* ***via l'URL*** *dans les deux documents). Créez un programme qui prend en paramètre un nom de fichier* `XML` *et teste la validité de celui-ci pour la DTD qu'il référence. Testez ce programme avec vos fichiers.*

Exercice 14 ***Egalité de deux droites.*** *Ecrivez un programme qui prend en paramètre deux noms de fichiers* `XML`, *vérifient que ceux-ci valident la DTD pour les droites, et teste l'égalité des deux droites représentées par ces fichiers. Testez votre programme avec vos fichiers et des fichiers envoyés par vos voisins !*

Notez qu'on travaille maintenant à partir de la connaissance de la DTD, contrairement à l'exercice 9, dans lequel on supposait connaitre la structure des fichiers fournis sans aucune validation. La méthode de l'exercice précédent est bien sûr nettement préférable.

6 Références

- Les tutoriaux du W3C :
  `http://www.w3schools.com/xml/`
- Le site de JDOM :
  `http://www.jdom.org/`