XML et Java

Xavier Crégut
<Prénom.Nom@enseeiht.fr>

ENSEEIHT
Télécommunications & Réseaux

Sommaire

Largement inspiré de http://jfod.cnam.fr/NFP121/supports/NFP121_cours_9_XML.pdf

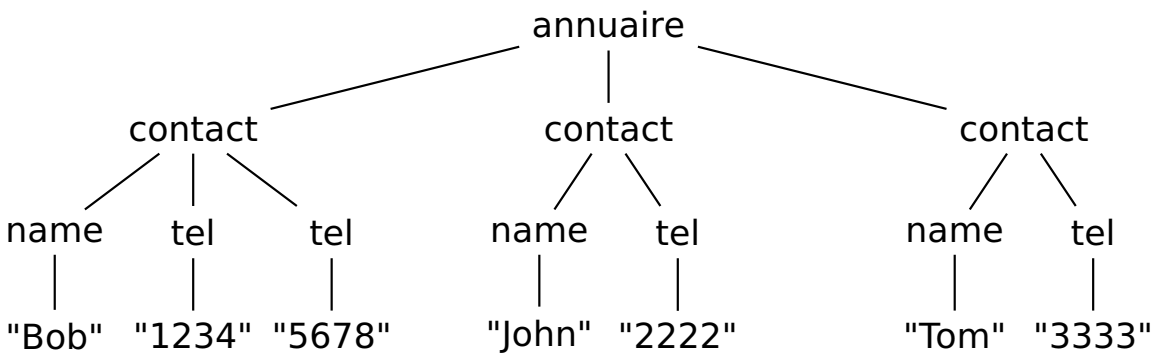
Sommaire

Structuration des données

Carnet d'adresses :

Bob	1234 (bureau)
Bob	5678 (maison)
John	2222 (bureau)
Tom	3333

```
1  <?xml version="1.0" encoding="ISO-8859-15" ?>
2
3  <annuaire>
4      <contact>
5          <name>Bob</name>
6          <tel type="bureau">1234</tel>
7          <tel type="maison">5678</tel>
8      </contact>
9      <contact>
10         <name>John</name>
11         <tel type="bureau">2222</tel>
12     </contact>
13     <contact>
14         <name>Tom</name>
15         <tel>3333</tel>
16     </contact>
17 </annuaire>
```



Document XHTML

```
1   <?xml version="1.0" encoding="iso-8859-1"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD␣XHTML␣1.0␣Transitional//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5   <html xmlns="http://www.w3.org/1999/xhtml">
6   <head>
7     <title>Une page XHTML</title>
8   </head>
9   <body>
10    <h1>Une page XHTML</h1>
11    <h2>Introduction</h2>
12    <p>
13    Une page XHTML est une page HTML
14    qui est bien formée au sens XML
15    et valide par rapport à une DTD.
16    </p>
17
18    <h2>Divers</h2>
19    <p>
20    Les blancs
21    ne sont pas significatifs.
22
23    Ni les lignes blanches !
24    </p>
25    <br />
26    <p>
27    Il faut explicitement mettre <![CDATA[<br />]]>
28    pour avoir un retour à la ligne.
29    </p>
30  </body>
31  </html>
```

Document à prédominance texte : voir élément contenu

```
1   <?xml version="1.0" encoding="ISO-8859-1"?>
2
3   <sujet nature="TP" title="Réalisation d'un ensemble d'entiers" duree="85" id="tp:ensemble">
4       <require sujet="td:ensemble"/>
5       <require sujet="tp:jml"/>
6       <contenu>   <!-- contient essentiellement du texte -->
7       Le <refsujet/> illustre la programmation par contrat sur la réalisation d'un ensemble qui est
8       ensuite utilisé pour trouver des nombres premiers par l'algorithme du crible d'Ératosthène.
9
10      Ce sujet montre l'intérêt de la notion d'interface.  Deux réalisations de l'ensemble sont
11      demandées.   La classe Crible s'appuyant sur l'interface de l'ensemble, elle peut être
12      utilisée avec les deux réalisations sans modification.
13
14      <remarque>
15      Les assertions de la programmation par contrat sont exprimées en
16      \footahref{http://www.jmlspecs.org/}{JML (Java Modeling Language)}.
17      </remarque>
18
19      Voici les \ENSREF{<numsujet />/}{classes fournies}.
20
21      <solution>
22      Voici une \ENSREF{<numsujet />solution/}{solution possible}.
23      </solution>
24
25      <makefile>
26      <pdf2up goal="repro">
27          <target><numsujet />-repro1</target>
28          <dependency><fichiersujet />.pdf</dependency>
29      </pdf2up>
30      </makefile>
31      </contenu>
32  </sujet>
```

Principaux intérêts de XML

Structuration des données

- Se concentrer sur la structure des données
- Et non sur la manière de les présenter... ou représenter !

Aspects lexical et syntaxique fixés

- inutile de réécrire ces analyseurs (réutilisation)
- pas besoin de générateurs d'analyseurs
- plusieurs approches existent pour :
  - lire un contenu XML
  - écrire un contenu XML
- $\Longrightarrow$ On peut **se concentrer sur l'analyse sémantique** !

Document texte

- (relativement) facile à lire (et modifer (!)) par un humain
  - mais les fichiers XML sont souvent et de plus en plus cachés !
- facile à transmettre sur un réseau

Sommaire

- Historique
- Document XML bien formé
- Document XML valide
- Feuille de style
- Espaces de nom

Historique

- Développé par le W3C à partir de 1996
- Normalisation : 10/02/1998 pour la version 1.0
- Ancêtre : SGML (1969)
  - langage à balise utilisé dans l'industrie de la documentation et de l'édition
  - + séparation structure (DTD), présentation (feuille de style) et contenu (instance)
  - syntaxe complexe
- Ancêtre (dans une moindre mesure) : HTML
  - documents facilement transmis, reçus et traités sur le Web
  - mais limité
- Caractéristiques d'XML :
  - langage à balise
  - extensible (les balises ne sont pas prédéfinies)
  - séparation entre structure (DTD), forme (feuille de style) et contenu (document XML)
  - facile à mettre en œuvre (objectif principal dans sa conception)
- Descendance :
  - XHTML, MathML, DocBook (O'Reilly), OpenDocument (LibreOffice), Ant...

Exemple de document XML

```
1  <?xml version="1.0" encoding="ISO-8859-15" standalone="yes" ?>
2
3  <book id="exemple"> <!-- largement inspiré de wikipedia -->
4     <title>Livre très simple</title>
5     <available />
6     <chapter id="premier">
7        <title>Chapitre très court</title>
8        <para>Bonjour tout le monde !</para>
9        <para>Ceci est un autre paragraphe...</para>
10    </chapter>
11    <chapter id='dernier'>
12       <title>Chapitre encore plus court</title>
13       <para>Au revoir !</para>
14    </chapter>
15 </book>
```

Principaux constituants d'un document XML

- Une déclaration en début de document (prologue) :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

- des éléments délimités par :
  - une balise ouvrante (ex : `<livre>`)
  - une balise fermante (ex : `</livre>`)
- un élément contient :
  - des éléments (ex : book)
  - du texte (ex : title)
  - ou mixte (pas conseillé pour structurer des données)
- si un élément est vide, on peut le noter : `<available />`
  - balises ouvrante et fermante sont contractées (balise auto-fermante)
- des attributs toujours définis sur la balise ouvrante
  - un nom (une clé, donc des noms différents pour un même élément !)
  - une valeur toujours entre guillemets (") ou apostrophes (')

```
<propriete nom="taille" valeur='10'/>
```

  - équivalent d'une entrée dans un tableau associatif (Map)

Règles syntaxiques

- Commencer par une déclaration :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

- Balisage sensible à la casse (<a> différent de <A>)
- Les balises doivent être appairées : <h1>...</h1>
- Les éléments ne doivent pas se chevaucher : <a> est faux !
- Il doit y avoir un et un seul élément racine qui encapsule tous les autres
  - structure d'arbre dont les nœuds sont les éléments
- Ne pas utiliser < (début de balise) et & (entité) seuls :
  - Les remplacer par `<` (<) et `&` (&) (ce sont des entités)

Un document qui respectent ces règles est un document bien formé.

- **Tout document XML doit être bien formé !**

xmllint : un vérificateur de document XML

- xmllint est un exemple d'outil qui vérifie un document XML

```
1   <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3   <main a1="<" a2="&lt;" a3='>' a4="'" a5='"' a6='&quot;&apos;' >
4       '">
5   </main>
```

donne par xmllint :

```
1   exemple-lt-faux.xml:3: parser error : Unescaped '<' not allowed in attributes values
2   <main a1="<" a2="&lt;" a3='>' a4="'" a5='"' a6='&quot;&apos;' >
3             ^
4   exemple-lt-faux.xml:3: parser error : attributes construct error
5   <main a1="<" a2="&lt;" a3='>' a4="'" a5='"' a6='&quot;&apos;' >
6             ^
7   exemple-lt-faux.xml:3: parser error : Couldn't find end of Start Tag main line 3
8   <main a1="<" a2="&lt;" a3='>' a4="'" a5='"' a6='&quot;&apos;' >
9             ^
10  exemple-lt-faux.xml:3: parser error : Extra content at the end of the document
11  <main a1="<" a2="&lt;" a3='>' a4="'" a5='"' a6='&quot;&apos;' >
12            ^
```

xmllint : un vérificateur de document XML (2)

```
1   <?xml version="1.0" encoding="ISO-8859-1" ?>
2
3   <main a1="&amp;" a2="&lt;" a3='>' a4="'" a5='"' a6='&quot;&apos;' >
4       '">
5   </main>
```

donne par xmllint :

```
1   <?xml version="1.0" encoding="ISO-8859-1"?>
2   <main a1="&amp;" a2="&lt;" a3="&gt;" a4="'" a5="&quot;" a6="&quot;'">
3       '"&gt;
4   </main>
```

- Utiliser > plutôt que >

Les données

- Les données apparaissent comme contenu texte d'un élément
- Il s'agit d'un flot de caractères
  - tous les caractères sauf & et <
  - à remplacer par `&` et `<`
- Pour insérer des caractères « spéciaux », on peut utiliser un section littérale appelée CDATA (Character DATA)

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <root>
3      if (<![CDATA[a < b && b < c]]>) ...
4  </root>
```

Le texte associé à l'élément `root` est : `if (a < b && b < c) ...`

DTD : Document Type Declaration

- **Objectif :** Définir la structure d'un document :
  - ordre des éléments
  - les attributs définis sur chaque élément
  - avec quelques contraintes
- **Moyens :**
  - DTD : Document Type Declaration
  - XML Schema, Relax NG, etc.
- Un document est valide par rapport à une DTD s'il respecte cette DTD
  - Le document doit bien sûr être bien formé !
- La DTD peut être définie dans le document (standalone) où à l'extérieur
- Utilisation de l'option `--valid` de xmllint

Exemple de DTD externe

```
1   <?xml version="1.0" encoding="ISO-8859-1"?> <!-- book.dtd -->
2
3   <!ELEMENT book (title,available?,chapter+)>
4   <!ELEMENT chapter (title,para*)>
5   <!ELEMENT available EMPTY>
6   <!ELEMENT title (#PCDATA)>
7   <!ELEMENT para (#PCDATA)>
8
9   <!ATTLIST book
10      id ID #REQUIRED
11      page-count CDATA #IMPLIED
12      version CDATA #FIXED "1.0">
13  <!ATTLIST chapter id ID #REQUIRED>
```

```
1   <?xml version="1.0" encoding="ISO-8859-15" standalone="no" ?>
2
3   <!DOCTYPE book SYSTEM "book.dtd">
4
5   <book id="exemple"> <!-- largement inspiré de wikipedia -->
6      <title>Livre très simple</title>
7      <available />
8      <chapter id="premier">
9         <title>Chapitre très court</title>
10        <para>Bonjour tout le monde !</para>
11        <para>Ceci est un autre paragraphe...</para>
12     </chapter>
13  </book>
```

Syntaxe d'une DTD : déclaration d'un attribut

```
<!ELEMENT nom modele>
```

- nom : le nom de l'élément
- modèle : ce que peut contenir l'élément
  - ANY : Tout type de données
  - EMPTY : Élément vide
  - (contenu) : contenu décrit la structure de ce que contient l'élément
- Le contenu peut être :
  - #PCDATA : une donnée texte (Parsed CDATA)
  - une expression régulière :
    - E : 1 fois l'élément E
    - E1 , E2 : Séquence (E1 puis E2)
    - E1 | E2 : Choix (E1 ou E2)
    - E? : Optionnel (0 ou 1 fois E)
    - E+ : Au moins 1 fois E
    - E∗ : 0 ou plusieurs fois E
    - ( ) : grouper des éléments

Déclaration d'un attribut

Forme générale :

```
<!ATTLIST nomElement
    nomAttribut_1 type_1 mode_1
    ...
    nomAttribut_n type_n mode_1>
```

- **Type :** valeurs possibles pour l'attribut :
  - CDATA : chaîne de caractères
  - ID : identifiant (doit donc être unique dans le document)
  - IDREF : référence à un identifiant (qui doit exister dans le document)
  - NMTOKEN et NMTOKENS : un ou plusieurs identifiants XML séparés par des blancs
  - `(ValeurA | ValeurB | ...)` : une valeur prise dans la liste
- **Mode :** contrainte sur la présence de l'attribut ou sa valeur
  - "valeur" : valeur par défaut si attribut absent.
  - `#REQUIRED` : attribut obligatoire
  - `#IMPLIED` : attribut facultatif
  - `#FIXED` "valeur" : attribut avec valeur imposée

Déclaration d'une DTD interne

```
1   <?xml version="1.0"?>
2
3   <!DOCTYPE purchase-order [
4   <!ELEMENT purchase-order (customer)>
5   <!ELEMENT customer (account-id, name)>
6   <!ELEMENT account-id (#PCDATA)>
7   <!ELEMENT name (first, mi, last)>
8   <!ELEMENT first (#PCDATA)>
9   <!ELEMENT mi (#PCDATA)>
10  <!ELEMENT last (#PCDATA)>
11  ]>
12
13  <purchase-order>
14      <customer>
15      <account-id>10-487</account-id>
16      <name>
17          <first>William</first>
18          <mi>R</mi>
19          <last>Stanek</last>
20      </name>
21      </customer>
22  </purchase-order>
```

Les entités

Définition : Une entité est un alias pour une séquence d'information.
Les entités doivent être définies dans la DTD.

Entités caractères : Forme : `<!ENTITY nom "&#code;">`

```
1  <!ENTITY exist "&#8707;"> <!-- there exists, U+2203 ISOtech -->
2  <!ENTITY delta "&#948;"> <!-- greek small letter delta, U+03B4 ISOgrk3 -->
```

Entités générales : Forme : `<!ENTITY nom "texte">`

```
1  <!ENTITY wysiwyg "What␣You␣See␣Is␣What␣You␣Get">
2  ...
3  Les éditeurs de type &wysiwyg;
```

Entités externes

```
1  <?xml version="1.0" encoding="ISO-8859-15" ?>
2  <!DOCTYPE doc SYSTEM "book.dtd" [
3      <!ENTITY chap1 SYSTEM "chapitres/c1.xml">
4      <!ENTITY chap2 SYSTEM "chapitres/c2.xml">
5  ]>
6  <book>
7      <title>Libre très simple</title>
8      &chap1;
9      &chap2;
10 </book>
```

Les entités paramètres

Ne peuvent être utilisées que dans une DTD !

```
1   <?xml version="1.0" encoding="ISO-8859-1"?>
2
3   <!ELEMENT envoi (livraison,facture?)>
4   <!ENTITY % adresse "numero?,rue,code,ville?">
5   <!ELEMENT livraison (%adresse;)>
6   <!ELEMENT facture (%adresse;)>
7   <!ELEMENT numero (#PCDATA)>
8   <!ELEMENT rue (#PCDATA)>
9   <!ELEMENT code (#PCDATA)>
10  <!ELEMENT ville (#PCDATA)>
```

```
1   <?xml version="1.0" encoding="ISO-8859-15" standalone="no" ?>
2
3   <!DOCTYPE envoi SYSTEM "envoi.dtd">
4
5   <envoi>
6       <livraison>
7           <rue>Camichel</rue>
8           <code>31071</code>
9       </livraison>
10      <facture>
11          <rue>Monso</rue>
12          <code>31029</code>
13      </facture>
14  </envoi>
```

Quel est l'intérêt dans cet exemple ?

Visualisation d'un fichier XML dans un navigateur

```xml
<?xml version="1.0" encoding="ISO-8859-15" standalone="yes" ?>

<book id="exemple"> <!-- largement inspiré de wikipedia -->
   <title>Livre très simple</title>
   <available />
   <chapter id="premier">
      <title>Chapitre très court</title>
      <para>Bonjour tout le monde !</para>
      <para>Ceci est un autre paragraphe...</para>
   </chapter>
   <chapter id='dernier'>
      <title>Chapitre encore plus court</title>
      <para>Au revoir !</para>
   </chapter>
</book>
```

Aucune information de style ne semble associée à ce fichier XML.
L'arbre du document est affiché ci-dessous.

```
−<book id="exemple">
   <!-- largement inspiré de wikipedia -->
   <title>Livre très simple</title>
   <available/>
 −<chapter id="premier">
      <title>Chapitre très court</title>
      <para>Bonjour tout le monde !</para>
      <para>Ceci est un autre paragraphe...</para>
   </chapter>
 −<chapter id="dernier">
      <title>Chapitre encore plus court</title>
      <para>Au revoir !</para>
   </chapter>
</book>
```

- Résultat de la visualisation du document dans un navigateur.

Utilisation de feuille de style CSS

Ajout dans la déclaration.

```
<?xml-stylesheet href="book.css" type="text/css"?>
```

Et book.css :

```
<style type="text/css">
title {
    display: block;
    color: blue;
    font-size: 24px;
    font-weight: bold;
    margin: 20px 0px
}
book > title {
    text-align: center;
    font-size: 36px;
    font-weight: bold;
    background-color: blue;
    margin: 30px 5px;
    color: white;
}
para {
    color: green;
    display: block;
}
```

Sommaire

Structuration des données

Carnet d'adresses :

Bob	1234 (bureau)
Bob	5678 (maison)
John	2222 (bureau)
Tom	3333

```
1  <?xml version="1.0" encoding="ISO-8859-15" ?>
2
3  <annuaire>
4      <contact>
5          <name>Bob</name>
6          <tel type="bureau">1234</tel>
7          <tel type="maison">5678</tel>
8      </contact>
9      <contact>
10         <name>John</name>
11         <tel type="bureau">2222</tel>
12     </contact>
13     <contact>
14         <name>Tom</name>
15         <tel>3333</tel>
16     </contact>
17 </annuaire>
```

La commande Ant

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <project name="TP" default="run" basedir=".">
3      <property name="src" location="src"/>
4      <property name="build" location="build"/>
5
6      <target name="init" description="Initialiser␣le␣projet">
7          <tstamp />  <!-- définir des propriétés : DSTAMP, TSTAMP, TODAY -->
8          <mkdir dir="${build}"/>
9      </target>
10
11     <target name="compile" depends="init"
12             description="Compiler␣le␣programme␣principal">
13         <javac srcdir="${src}" destdir="${build}">
14             <include name="editeur/ExempleSchema.java"/>
15         </javac>
16     </target>
17
18     <target name="run" depends="compile"
19             description="Lancer␣le␣programme␣principal">
20         <java classname="editeur.ExempleSchema" classpath="${build}" fork="true">
21             <arg value="-2" />
22         </java>
23     </target>
24
25     <target name="clean" description="Nettoyer␣le␣projet">
26         <delete dir="${build}"/>
27     </target>
28 </project>
```

Les propriétés de Java

```java
import java.util.Properties;
import java.io.*;
public class SaveProperties {
    public static void main(String[] args) throws Exception {
        Properties props = System.getProperties();
        OutputStream out = new FileOutputStream("/tmp/prop.xml");
        props.storeToXML(out, "Propriétés système", "ISO-8859-15");
        // ...
        Properties chargees = new Properties();
        chargees.loadFromXML(new FileInputStream("/tmp/prop.xml"));
        assert System.getProperties().equals(chargees);
    }
}
```

```xml
<?xml version="1.0" encoding="ISO-8859-15" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Propriétés système</comment>
<entry key="java.runtime.name">Java(TM) SE Runtime Environment</entry>
<entry key="sun.boot.library.path">/usr/lib/jvm/java-6-sun-1.6.0.26/jre/lib/i386</entry>
<entry key="java.vm.version">20.1-b02</entry>
<entry key="java.vm.vendor">Sun Microsystems Inc.</entry>
<entry key="java.vendor.url">http://java.sun.com/</entry>
<entry key="path.separator">:</entry>
<entry key="java.vm.name">Java HotSpot(TM) Server VM</entry>
<entry key="file.encoding.pkg">sun.io</entry>
<entry key="sun.java.launcher">SUN_STANDARD</entry>
<entry key="user.country">FR</entry>
    <!-- ... -->
</properties>
```

Android

```xml
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/helpView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/helpText"
    />
</ScrollView>
```

```java
package fr.n7.android;
import android.app.Activity;
import android.os.Bundle;

public class Help extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.help);
      TextView text = findViewById(R.id.helpView);
   }
}
```

Autres utilisations

Persistance d'information

- OpenOffice / LibreOffice : le format OpenDocument Format (ODF)
- ...

Échange d'informations / Sérialisation

- Services Web (SOAP...)

Sommaire

JAXP : Java API for XML Processing

JAXP : interface de programmation Java pour manipuler et traiter des fichiers XML

JAXP inclut en particulier :

- SAX : Simple API for XML, voir T. 35
- DOM : Document Object Model (recommandation du W3C), voir T. 49
- StAX : Streaming API for XML (depuis JDK 6), voir T. 67
- XSLT : XML Stylesheet Language for Transformation
  - convertir un document XML en une autre forme de données
  - non présenté dans ce cours

Chaque API est définie sous la forme d'interfaces :

- permettre à chacun de proposer son implantation (plusieurs fournisseurs possibles)
- de manière transparente pour les programmes utilisateurs
- utilisation de fabriques (et fabriques abstraites) : pour construire les classes concrètes sans avoir à connaître explicitement le fournisseur.

Sommaire

Principe de SAX : Simple API for XML

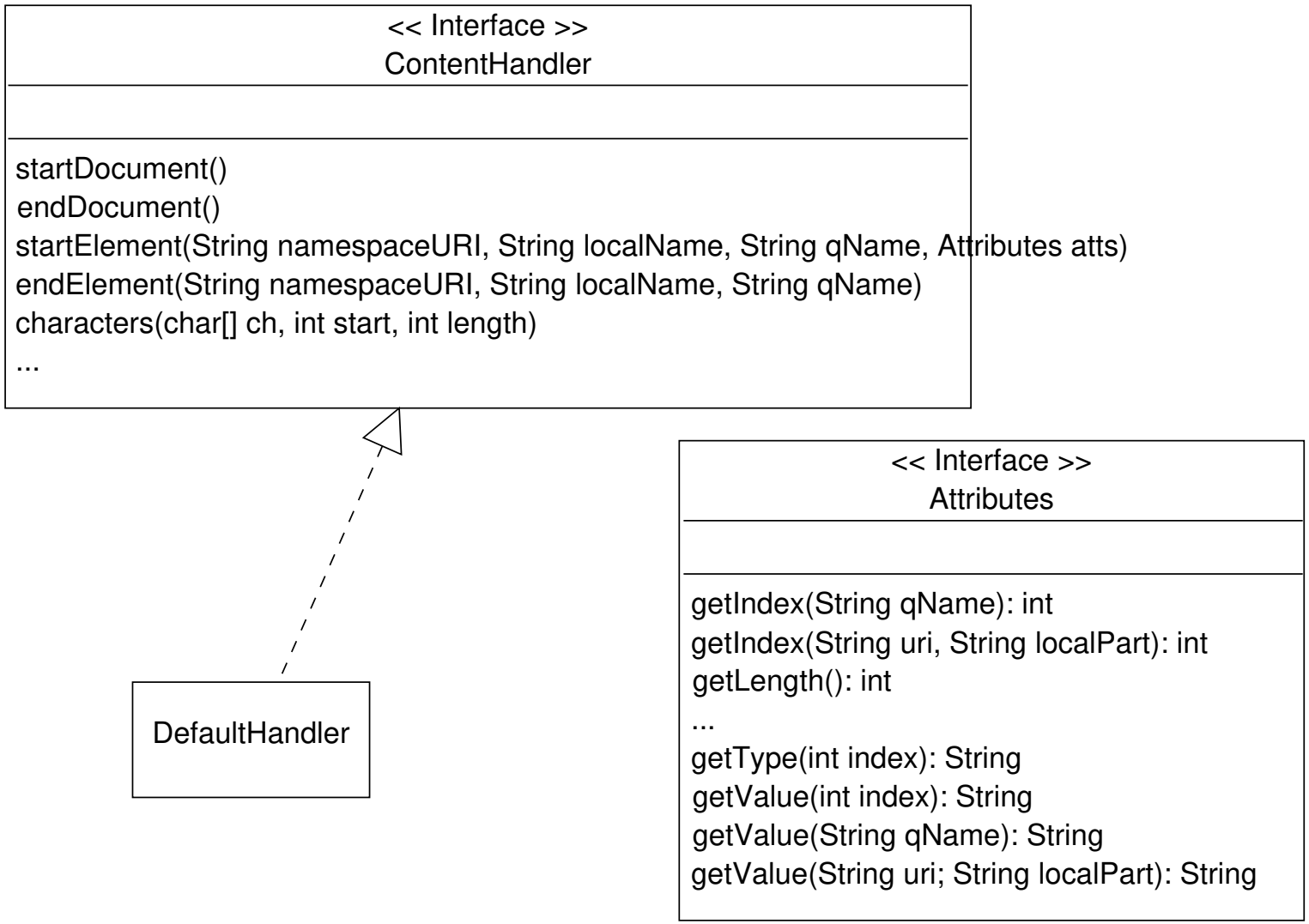
Approche événementielle :

- SAX analyse le fichier d'entrée (flot de caractères)
- reconnaît les constituants XML :
  - début d'un élément (balise ouvrante)
  - fin d'un élément (balise fermante)
  - texte
- appelle la méthode correspondant au constituant reconnu
  - startElement
  - endElement
  - characters
- ces méthodes sont abstraites

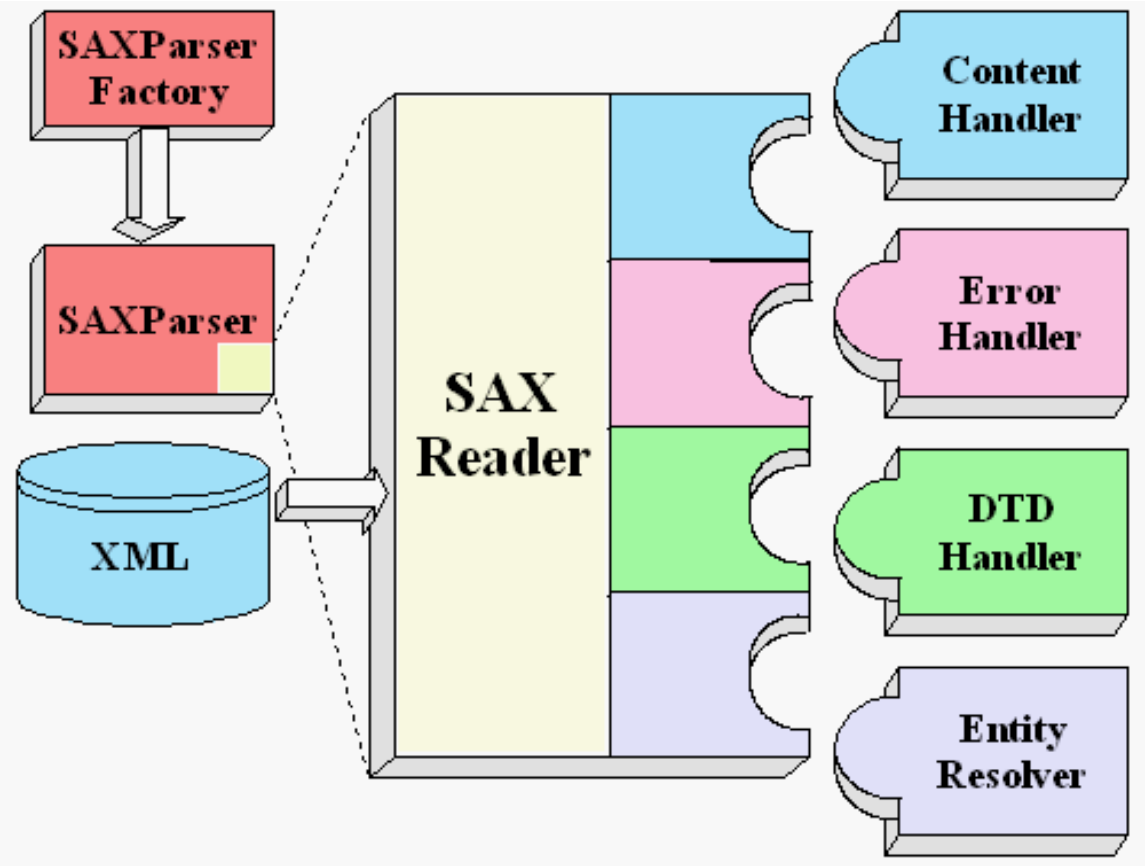
Principe sur un exemple

```
1  <?xml version="1.0" encoding="ISO-8859-15" standalone="yes" ?>
2                                  ⟶ startDocument()
3  <book id="exemple">             ⟶ startElement("book", ["id" = "exemple"])
4      <title>                     ⟶ startElement("title", [])
5          Livre très simple       ⟶ characters("Livre très simple")
6      </title>                    ⟶ endElement("title")
7      <available />               ⟶ startElement("available", []);
8                                  ⟶ endElement("available")
9      <chapter id="premier">      ⟶ startElement("chapter", ["id", "premier"])
10         <title>                 ⟶ startElement("title", [])
11             Chapitre très court ⟶ characters("Chapitre très court")
12         </title>                ⟶ endElement("title")
13         <para>                  ⟶ startElement("para", [])
14             Bonjour tout le monde ! ⟶ characters("Bonjour...")
15         </para>                 ⟶ endElement("para")
16         <para>                  ⟶ startElement("para", [])
17             Ceci est un autre paragraphe... ⟶ characters("Ceci...")
18         </para>                 ⟶ endElement("para")
19     </chapter>                  ⟶ endElement("chapter")
20     <chapter id='dernier'>      ⟶ startElement("chapter", ["id", "dernier"])
21     ...                         ⟶ ...
22     </chapter>                  ⟶ endElement("chapter")
23 </book>                         ⟶ endElement("book")
24                                 ⟶ endDocument()
```

Principales classes de SAX



Architecture de SAX



Exemple : AfficheurSaX

But : Afficher les éléments trouvés dans le fichier

```
1   import org.xml.sax.helpers.DefaultHandler;
2   import org.xml.sax.Attributes;
3   import org.xml.sax.SAXException;
4
5   public class AfficheurHandler extends DefaultHandler {
6       private Indenteur sortie = new Indenteur();
7
8       @Override public void startDocument() {
9           sortie.afficher("Document␣[");
10          sortie.indenter("\t");
11      }
12
13      @Override public void endDocument() {
14          sortie.annuler();
15          sortie.afficher("]");
16      }
17
18      @Override public void startElement(String uri, String localName,
19                          String qName, Attributes attributs) throws SAXException
20      {
21          sortie.afficher(qName + '/' + localName + '/' + uri + "␣{");
22          sortie.indenter("\t");
23          for (int indice = 0; indice < attributs.getLength(); indice++) {
24              sortie.afficher("[" + attributs.getQName(indice) + "␣=␣"
25                      + attributs.getValue(indice) + "]");
```

Exemple : AfficheurSaX (2)

```
26          }
27      }
28
29      @Override public void endElement(String uri, String localName,
30                          String qName) throws SAXException
31      {
32          sortie.annuler();
33          sortie.afficher("}␣--␣" + qName);
34      }
35
36      @Override public void characters(char[] contenu, int début, int taille) throws SAXException {
37          String texte = new String(contenu, début, taille);
38          sortie.afficher("\"" + texte.trim() + "\"");
39      }
40  }
```

Traiter un document XML avec SAX

But : Afficher les éléments trouvés dans le fichier

```
1   import javax.xml.parsers.*;     // SAXParserFactory, SAXParser
2   import org.xml.sax.*;           // XMLReader, SAXException
3
4   public class AfficheurSAX {
5
6       public static XMLReader construireLecteurXML()
7           throws SAXException, ParserConfigurationException, java.io.IOException
8       {
9           SAXParserFactory usine = SAXParserFactory.newInstance();
10          XMLReader lecteur = usine.newSAXParser().getXMLReader();
11          lecteur.setFeature("http://xml.org/sax/features/validation", true);
12          lecteur.setFeature("http://xml.org/sax/features/namespaces", true);
13          return lecteur;
14      }
15
16      public static void analyserFichierXML(String fichier, ContentHandler analyseur)
17          throws SAXException, ParserConfigurationException, java.io.IOException
18      {
19          XMLReader lecteur = construireLecteurXML();
20          lecteur.setContentHandler(analyseur);
21          lecteur.parse(fichier);
22      }
23
24
25      public static void main(String[] args) {
```

Traiter un document XML avec SAX (2)

```
26          if (args.length == 0) {
27              System.out.println("Utilisation : AfficheurSAX <nom fichier>");
28          } else {
29              try {
30                  String fichier = args[0];
31                  AfficheurHandler afficheur = new AfficheurHandler();
32                  analyserFichierXML(fichier, afficheur);
33              }
34              catch (Exception e) {
35                  e.printStackTrace();
36              }
37          }
38      }
39  }
```

Exemple d'utilisation : avec DTD

```
1   Document [
2       book/book/ {
3           [id = exemple]
4           [version = 1.0]
5           title/title/ {
6               "Livre␣très␣simple"
7           } -- title
8           available/available/ {
9           } -- available
10          chapter/chapter/ {
11              [id = premier]
12              title/title/ {
13                  "Chapitre␣très␣court"
14              } -- title
15              para/para/ {
16                  "Bonjour␣tout␣le␣monde␣!"
17              } -- para
18              para/para/ {
19                  "Ceci␣est␣un␣autre␣paragraphe..."
20              } -- para
21          } -- chapter
22          chapter/chapter/ {
23              [id = dernier]
24              title/title/ {
25                  "Chapitre␣encore␣plus␣court"
26              } -- title
27              para/para/ {
28                  "Au␣revoir␣!"
29              } -- para
30          } -- chapter
31      } -- book
32  ]
```

Exemple d'utilisation : sans DTD

Noter le traitement des PCDATA

```
Document [
[Error] docbook-wikipedia.xml:3:6: Le document nest pas valide : aucune grammaire détectée.
[Error] docbook-wikipedia.xml:3:6: L'élément racine de document "book" doit correspondre à la racine DO
    book/book/ {
        [id = exemple]
        ""
        ""
        title/title/ {
            "Livre très simple"
        } -- title
        ""
        available/available/ {
        } -- available
        ""
        chapter/chapter/ {
            [id = premier]
            ""
            title/title/ {
                "Chapitre très court"
            } -- title
            ""
            para/para/ {
                "Bonjour tout le monde !"
            } -- para
            ""
            para/para/ {
                "Ceci est un autre paragraphe..."
            } -- para
            ""
        } -- chapter
        ""
        chapter/chapter/ {
            [id = dernier]
```

Produire la table des matières d'un book

```
1   import org.xml.sax.helpers.DefaultHandler;
2   import org.xml.sax.Attributes;
3   import org.xml.sax.SAXException;
4
5   /** Afficher la table des matière d'un document XML valide par rapport à book.dtd. */
6   public class TDMHandler extends DefaultHandler {
7       private boolean inChapter;
8       private boolean inChapterTitle;
9       private String lastTitle;   // updated if inChapterTitle
10      private int chapterNumber;          // number of current chapter
11
12      @Override public void startDocument() {
13          this.inChapterTitle = this.inChapter = false;
14          this.chapterNumber = 0;
15      }
16
17      @Override public void startElement(String uri,
18                          String localName,
19                          String qName, Attributes attributs) throws SAXException
20      {
21          if (qName.equals("chapter")) {
22              this.inChapter = true;
23              this.chapterNumber++;
24          } else if (qName.equals("title")) {
25              if (this.inChapter) {
26                  this.inChapterTitle = true;
```

Produire la table des matières d'un book (2)

```
27        }   }   }
28
29        @Override public void endElement(String uri,
30                            String localName,
31                            String qName) throws SAXException
32        {
33            if (qName.equals("chapter")) {
34                this.inChapter = false;
35            } else if (qName.equals("title")) {
36                if (this.inChapterTitle) {
37                    System.out.println(this.chapterNumber + "\t" + lastTitle);
38                    this.inChapterTitle = false;
39        }   }   }
40
41        @Override
42        public void characters(char[] contenu, int début, int taille) throws SAXException {
43            if (this.inChapterTitle) {
44                this.lastTitle = new String(contenu, début, taille);
45        }   }
46    }
```

Résultat :

```
1   1   Chapitre très court
2   2   Chapitre encore plus court
```

Bilan sur SAX

- API SAX, version 2.0 (Mai 2000)
- Traitement à la volée d'un flot XML
  - + Simple (sur le principe, pas forcément pour l'écriture des Handler !)
  - + Rapide
  - + Faible consommation mémoire
  - − Impossible de lire en avant
  - − Mémoriser les informations qui seront utiles ensuite
- Approche événementielle :
  - C'est à l'utilisateur de gérer l'état de l'analyse
  - à faire évoluer en fonction des événements (tokens, méthodes) reçus
  - $\Longrightarrow$ s'appuyer sur les automates
- Ne permet pas de créer un fichier XML (pas le but !)

Sommaire

- DOM
- JDOM : DOM adapté à Java

DOM : Document Object Model

Principe :

- Construire en mémoire une représentation du document XML
- d'où le nom : Document Object Model (DOM)
  - recommandation du W3C
  - API indépendante de tout langage de programmation et plate-forme
  - pour accéder et mettre à jour le contenu et la structure d'un document XML

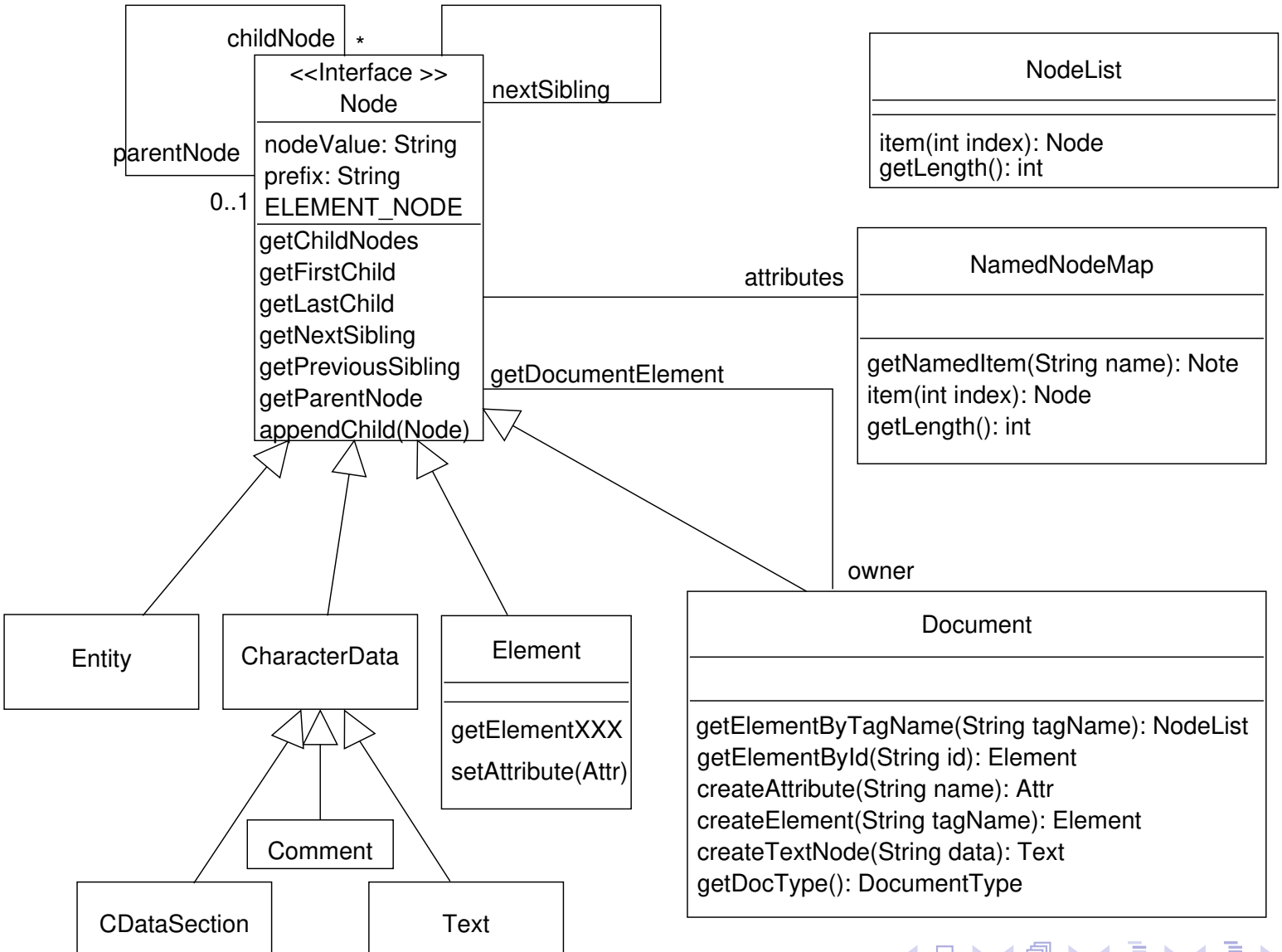
Conséquences :

- nécessite une lecture complète du document
- tout le document est représenté en mémoire
- pas de traitement à la volée
- toutes les informations du document sont disponibles

JDOM

- Adaptation de DOM pour le monde Java
- Donc, plus facile à utiliser pour un programmeur Java

Architecture de DOM



Traiter un document XML avec DOM

```
1   import javax.xml.parsers.*;
2   import org.xml.sax.*;
3   import org.w3c.dom.*;
4
5   /** Afficher les constituant d'un fichier XML */
6   public class AfficheurDOM {
7       private Indenteur sortie = new Indenteur();
8
9       /** Afficher le noeud n avec la tabulation précisée ainsi que tous ses fils
10       *  avec une tabulation supérieure.  Ici, on affiche seulement, les balises
11       *  (ouvrantes et fermantes), les attributs et les TEXT (PCDATA).  */
12      public void afficherNoeud(Node n) {
13          if (n.getNodeType() == Node.ELEMENT_NODE) { // un element
14              sortie.afficher(n.getNodeName() + " {");
15              sortie.indenter("\t");
16              // afficher les attributs
17              NamedNodeMap attributs = n.getAttributes();
18              for (int indice = 0; indice < attributs.getLength(); indice++) {
19                  Attr attribut = (Attr) attributs.item(indice);
20                  sortie.afficher("[" + attribut.getName() + " = " + attribut.getValue() + "]");
21              }
22          } else if (n.getNodeType() == Node.TEXT_NODE) {
23              sortie.afficher("\"" + n.getNodeValue().trim() + "\"");
24          }
25
26          // afficher les fils
```

Traiter un document XML avec DOM (2)

```
27            Node fils = n.getFirstChild();
28            while (fils != null) {
29                afficherNoeud(fils);
30                fils = fils.getNextSibling();
31            }
32
33            // Terminer l'affichage du noeud (après l'affichage de ses fils)
34            if (n.getNodeType() == Node.ELEMENT_NODE) {
35                sortie.annuler();
36                sortie.afficher("} -- " + n.getNodeName());
37        }   }
38
39        /** Analyser et afficher le fichier args[0] */
40        public static void main(String[] args) {
41            if (args.length == 0) {
42                System.out.println("Utilisation : AfficheurDOM <nomFichier>");
43            } else {
44                try {
45                    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
46                    factory.setValidating(true);
47                    DocumentBuilder builder = factory.newDocumentBuilder();
48                    Document document = builder.parse(args[0]);
49                    new AfficheurDOM().afficherNoeud(document);
50                } catch (Exception e) {
51                    e.printStackTrace();
52    }   }   }   }
```

Exemple d'utilisation : avec DTD

```
book {
    [id = exemple]
    [version = 1.0]
    ""
    ""
    title {
        "Livre très simple"
    } -- title
    ""
    available {
    } -- available
    ""
    chapter {
        [id = premier]
        ""
        title {
            "Chapitre très court"
        } -- title
        ""
        para {
            "Bonjour tout le monde !"
        } -- para
        ""
        para {
            "Ceci est un autre paragraphe..."
        } -- para
        ""
    } -- chapter
    ""
    chapter {
        [id = dernier]
        ""
        title {
            "Chapitre encore plus court"
        } -- title
        ""
        para {
            "Au revoir !"
        } -- para
        ""
    } -- chapter
    ""
} -- book
```

Noter les textes vides en trop (alors qu'il ne font pas partie de la DTD !)

Produire la table des matières d'un books

```java
import javax.xml.parsers.*;
import org.w3c.dom.*;

/** Afficher la table des matière d'un document XML valide par rapport à book.dtd. */
public class TDMDOM {

    public static void main(String[] args) throws Exception {
        if (args.length == 0) {
            System.out.println("Utilisation : java TDMDOM <nomFichier>");
        } else {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document document = builder.parse(args[0]);
            NodeList chapters = document.getElementsByTagName("chapter");
            for (int i = 0; i < chapters.getLength(); i++) {
                Element chapter = (Element) chapters.item(i);
                Node title = chapter.getElementsByTagName("title").item(0);
                System.out.println((i + 1) + "\t" + title.getTextContent());
}   }   }   }
```

```
1   Chapitre très court
2   Chapitre encore plus court
```

Et avec XPath...

https://www.w3.org/TR/xpath/

XPath : Langage (non XML) de requête simple sur un document XML.

```
1   import javax.xml.parsers.*;
2   import org.w3c.dom.*;
3   import javax.xml.xpath.*;
4
5   /** Afficher la table des matière d'un document XML valide par rapport à book.dtd. */
6   public class TDMXPath {
7       public static void main(String[] args) throws Exception {
8           assert args.length > 0;
9           DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
10          DocumentBuilder builder = factory.newDocumentBuilder();
11          Document document = builder.parse(args[0]);
12          Element racine = document.getDocumentElement();
13          XPathFactory xpf = XPathFactory.newInstance();
14          XPath path = xpf.newXPath();
15          // On pourrait aussi utilise le chemin "/book/chapter/title"
16          NodeList titles = (NodeList) path.evaluate("//chapter/title",
17                  racine, XPathConstants.NODESET);
18          for (int i = 0; i < titles.getLength(); i++) {
19              System.out.println((i + 1) + "\t" + titles.item(i).getTextContent());
20  }   }   }
```

```
1   1   Chapitre très court
2   2   Chapitre encore plus court
```

Créer un document XML

```
1   import javax.xml.parsers.*; // DocumentBuilderFactory, DocumentBuilder
2   import org.w3c.dom.*;       // Document
3   import javax.xml.transform.*;   // Transformer, TransformerFactory
4   import javax.xml.transform.dom.DOMSource;
5   import javax.xml.transform.stream.StreamResult;
6   import java.io.*;
7
8   public class ConstruireDOM {
9
10      public static void main(String[] args) throws Exception {
11          // Création d'un nouveau document DOM
12          DocumentBuilderFactory fabrique = DocumentBuilderFactory.newInstance();
13          DocumentBuilder constructeur = fabrique.newDocumentBuilder();
14          Document document = constructeur.newDocument();
15
16          document.setXmlStandalone(true);
17
18          // Construire quelques éléments
19          Element racine = document.createElement("racine");
20          racine.appendChild(document.createComment("un exemple"));
21          Element e1 = document.createElement("fils");
22          racine.appendChild(e1);
23          Text texte = document.createTextNode("le contenu");
24          e1.appendChild(texte);
25
26
27          // Définir des attributs
28          NamedNodeMap attributes = e1.getAttributes();
29          Attr a = document.createAttribute("taille");
30          a.setValue("10");
31          attributes.setNamedItem(a);
```

Créer un document XML (2)

```
32            Attr x = document.createAttribute("x");
33            x.setValue("A␣<␣B");    // < est spécial en XML !
34            racine.getAttributes().setNamedItem(x);
35
36            // Définir la racine de document
37            document.appendChild(racine);
38
39            // Obtenir le document dans une String
40            DOMSource domSource = new DOMSource(document);
41            StringWriter writer = new StringWriter();
42            StreamResult result = new StreamResult(writer);
43            TransformerFactory tf = TransformerFactory.newInstance();
44            Transformer transformer = tf.newTransformer();
45            transformer.setOutputProperty(OutputKeys.INDENT, "yes");
46            transformer.transform(domSource, result);
47            System.out.println(writer.toString());
48        }
49
50    }
```

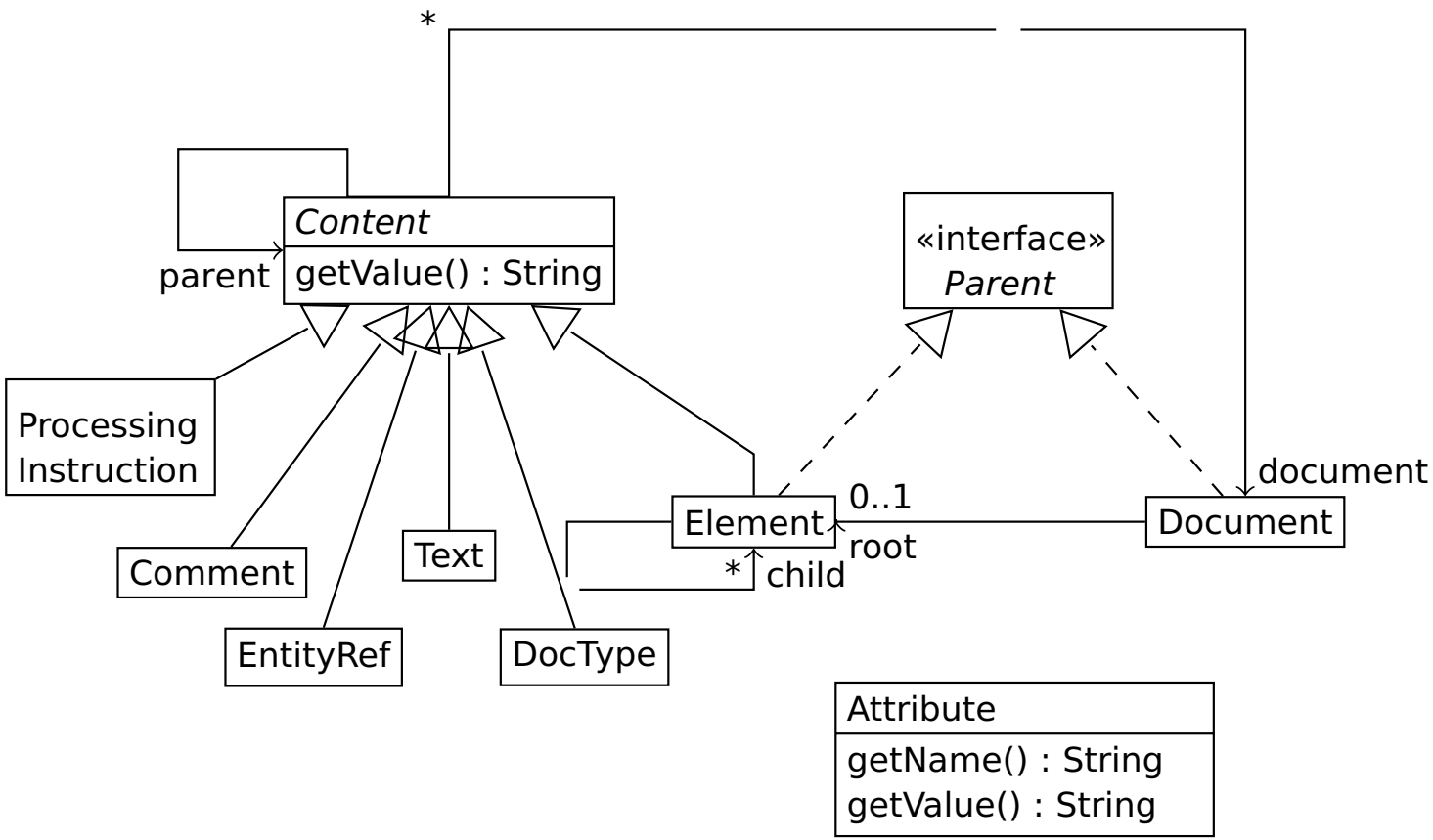
Résultat :

```
1   <?xml version="1.0" encoding="UTF-8"?><racine x="A␣&lt;␣B">
2   <!--un exemple-->
3   <fils taille="10">le contenu</fils>
4   </racine>
```

JDOM

- `http://www.jdom.org`
- Pourquoi JDOM plutôt que DOM
  - Plus facile à écrire
    - Plusieurs méthodes possibles pour faire la même chose
  - Meilleure intégration à Java
    - Utilisation de l'API des collections
    - Y compris la généricité depuis la version 2 (2012)
  - Interfaçage possible avec DOM (et SAX)

Architecture de JDOM



Construire un document avec JDOM

```
1   import org.jdom2.*;
2
3   public class ContactJDom {
4       public static void main(String[] args) {
5           Element racine = new Element("annuaire");
6           Element bob = new Element("contact");
7           racine.addContent(bob);
8           Element bobName = new Element("name");
9           bob.addContent(bobName);
10          bobName.setText("Bob");
11
12          // Définir <tel type="bureau">1234</tel>
13          Element telBureau = new Element("tel");
14          Attribute bureau = new Attribute("type", "bureau");
15          telBureau.setAttribute(bureau);
16          telBureau.setText("1234");
17          bob.addContent(telBureau);
18
19          // Définir <tel type="maison">5678</tel>
20          //  Plus concis !  Lisible ?
21          bob.addContent(new Element("tel")
22                  .setAttribute("type", "maison")
23                  .setText("5678"));
24
25          // Toujours peu lisible
26          racine.addContent(new Element("contact")
```

Construire un document avec JDOM (2)

```
27                  .addContent(new Element("name").setText("John"))
28                  .addContent(new Element("tel")
29                          .setAttribute("type", "bureau")
30                          .setText("2222")));
31
32          Document document = new Document(racine);
33          JDOMTools.ecrire(document, System.out);
34      }
35  }
```

Résultat :

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <annuaire>
3     <contact>
4       <name>Bob</name>
5       <tel type="bureau">1234</tel>
6       <tel type="maison">5678</tel>
7     </contact>
8     <contact>
9       <name>John</name>
10      <tel type="bureau">2222</tel>
11    </contact>
12  </annuaire>
```

Lire et écrire un document avec JDOM

```
1   import org.jdom2.*;
2   import org.jdom2.output.*;
3   import org.jdom2.input.*;
4   import java.io.*;
5
6   public class JDOMTools {
7
8       public static void ecrire(Document document, OutputStream out) {
9           try {
10              XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
11              sortie.output(document, out);
12          } catch (java.io.IOException e) {
13              throw new RuntimeException("Erreur␣sur␣écriture␣:␣", e);
14          }
15      }
16
17      public static Document getDocument(Reader reader) throws IOException {
18          SAXBuilder builder = new SAXBuilder(true);  // validant !
19          try {
20              return builder.build(reader);
21          } catch (JDOMException e) {
22              throw new RuntimeException("Erreur␣sur␣lecture␣XML␣:␣"
23                      + e.getMessage(), e);
24          }
25      }
26
27  }
```

Manipuler un document JDOM : Table des matières

```
1   import org.jdom2.*;
2   import org.jdom2.filter.*;
3   import org.jdom2.util.IteratorIterable;
4   import java.io.*;
5
6   public class TDMJDom {
7       public static void main(String[] args) throws Exception {
8           assert args.length > 0;
9           Document doc = JDOMTools.getDocument(new FileReader(args[0]));
10
11          int numero = 0;
12          // Un itérateur sur tous les éléments "chapter" du document
13          IteratorIterable<Element> itChapters
14              = doc.getDescendants( new ElementFilter("chapter"));
15          for (Element chapter : itChapters) {
16              Element title = chapter.getChild("title");
17              System.out.println(++numero + "\t" + title.getTextTrim());
18          }
19      }
20  }
```

Transformer book en article (simpliste)

```
1   import org.jdom2.*;
2   import org.jdom2.filter.*;
3   import java.io.*;
4   public class BookToArticleJDom {
5       public static void main(String[] args) throws Exception {
6           assert args.length > 0;
7           Document doc = JDOMTools.getDocument(new FileReader(args[0]));
8           renameIn(doc, "book", "article");
9           renameIn(doc, "chapter", "section");
10          removeElements(doc, "available");
11          JDOMTools.ecrire(doc, System.out);
12      }
13
14      public static void renameIn(Document doc, String oldName, String newName) {
15          for (Element current : doc.getDescendants(new ElementFilter(oldName))) {
16              current.setName(newName);
17      }   }
18
19      public static void removeElements(Document doc, String name) {
20          java.util.Iterator<Element> itElts = doc.getDescendants(new ElementFilter(name));
21          while (itElts.hasNext()) {
22              Element current = itElts.next();
23              // current.detach();  // --> ConcurrentModificationException
24              itElts.remove();
25  }   }   }
```

Résultat de la transformation

Entrée :

```
<?xml version="1.0" encoding="ISO-8859-15" standalone="no" ?>

<!DOCTYPE book SYSTEM "book.dtd">

<book id="exemple"> <!-- largement inspiré de wikipedia -->
   <title>Livre très simple</title>
   <available />
   <chapter id="premier">
      <title>Chapitre très court</title>
      <para>Bonjour tout le monde !</para>
      <para>Ceci est un autre paragraphe...</para>
   </chapter>
   <chapter id='dernier'>
      <title>Chapitre encore plus court</title>
      <para>Au revoir !</para>
   </chapter>
</book>
```

Résultat :

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<!DOCTYPE book SYSTEM "book.dtd">
<article id="exemple" version="1.0">
  <!-- largement inspiré de wikipedia -->
  <title>Livre très simple</title>
  <section id="premier">
    <title>Chapitre très court</title>
    <para>Bonjour tout le monde !</para>
    <para>Ceci est un autre paragraphe...</para>
  </section>
  <section id="dernier">
    <title>Chapitre encore plus court</title>
    <para>Au revoir !</para>
  </section>
</article>
```

Sommaire

StAX : Streaming API for XML

https://docs.oracle.com/javase/tutorial/jaxp/stax/index.html

- Document XML = flot de ses constituants, avec deux approches :
  - Stream API (ou Cursor API) : faire avancer un curseur sur les constituants et récupérer les informations du constituant courant

```
1  public interface XMLStreamReader {
2    public int next() throws XMLStreamException;
3    public boolean hasNext() throws XMLStreamException;
4    public String getLocalName();
5    public String getText();
6    public int getAttributeCount();
7    ...
8  }
```

  - Event API (Iterator API) : retourne les constituants imuables sous forme d'Event (`XMLEventReader`)
- Aussi léger que SAX : pas de construction explicite d'un modèle du document
- (Presque) aussi simple que DOM : le programmeur décide d'avancer dans le document
- Permet d'écrire un document XML (SAX ne permet que la lecture)
  - avec `XMLStreamWriter`

```
1  public interface XMLStreamWriter {
2    public void writeStartElement(String localName) throws XMLStreamException;
3    public void writeEndElement() throws XMLStreamException;
4    public void writeCharacters(String text) throws XMLStreamException;
5    ...
6  }
```

  - ou avec `XMLEventWriter`

Engendrer la table des matières

```
1   import javax.xml.stream.*;
2   import java.io.*;
3
4   public class TDMStAX {
5       public static void main(String[] args) throws Exception {
6           XMLInputFactory factory = XMLInputFactory.newInstance();
7           if (factory.isPropertySupported("javax.xml.stream.isValidating")) {
8               factory.setProperty("javax.xml.stream.isValidating", "true");
9           }
10          XMLStreamReader reader = factory.createXMLStreamReader(
11                  new FileReader("docbook-wikipedia-dtd.xml"));
12
13          // Réaliser le traitement !
14          int numero = 0;
15          while (reader.hasNext()) {
16              reader.next();
17              if (reader.isStartElement()) {
18                  if ("chapter".equals(reader.getLocalName())) {
19                      // <!ELEMENT chapter (title, para*)>
20                      // <!ELEMENT title (#PCDATA)>
21                      while (reader.next() != XMLStreamReader.START_ELEMENT); // <title>
22                      assert reader.getLocalName().equals("title");
23                      reader.next();  // character
24                      assert reader.isCharacters();
25                      String texte = reader.getText().trim();
26                      System.out.println(++numero + "\t" + texte);
27                  }
28  }   }   }   }
```

API Événementielle : Recopier un fichier XML

```
1   import javax.xml.stream.*;
2   import javax.xml.stream.events.*;
3   import java.io.*;
4
5   public class StAXCat {
6       public static void main(String[] args) throws Exception {
7           assert args.length > 0;
8           Reader in = new FileReader(args[0]);
9           XMLInputFactory factory = XMLInputFactory.newInstance();
10          XMLEventReader reader = factory.createXMLEventReader(in);
11          XMLOutputFactory outputFactory = XMLOutputFactory.newInstance();
12          XMLEventWriter writer = outputFactory.createXMLEventWriter(System.out);
13
14          while (reader.hasNext()) {
15              XMLEvent next = reader.nextEvent();
16              writer.add(next);
17          }
18
19
20      }
21  }
```

L'intérêt est surtout sur l'écriture : ce n'est pas le programme qui doit appeler explicitement la bonne méthode d'un StAXWriter.

Sommaire

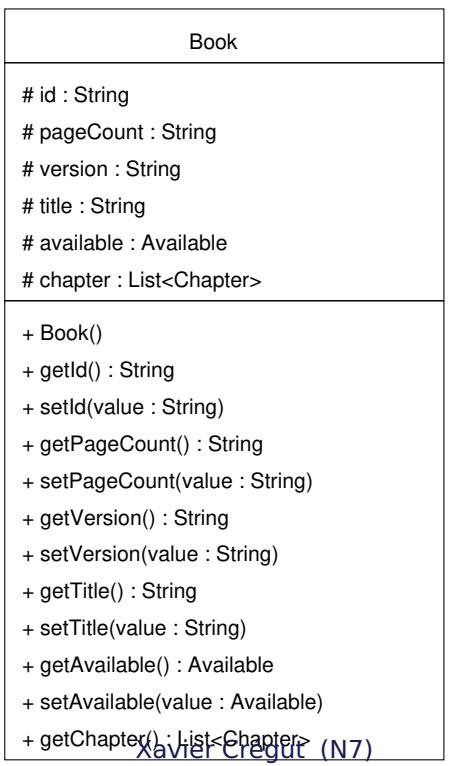
JAXB : Java Architecture for XML Binding

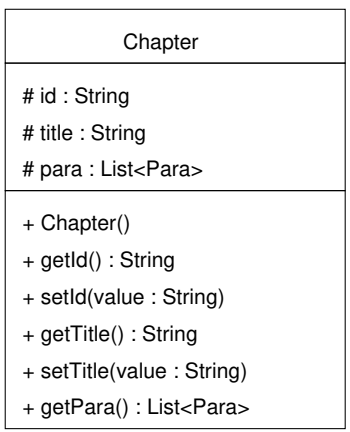
- Crée les classes Java qui correspondent à une DTD :
  - un élément XML devient une classe Java
  - un attribut XML devient un attribut Java
  - un fils XML devient un composant en Java (composition)
- Permet de créer un modèle objet à partir d'un fichier XML (et inversement)
  - marshal : passer d'un objet Java en mémoire à un document XML
  - unmarshal : passer d'un document XML à un objet en mémoire.
- Manipuler le document XML au travers d'un modèle objet typé (contrairement à DOM)
  - un élément XML = une classe Java
- Fonctionne plutôt avec des Schemas (DTD expérimental)

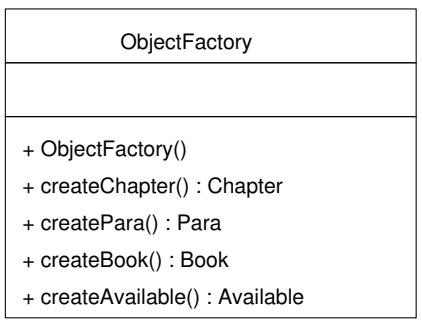
JAXB : Un exemple

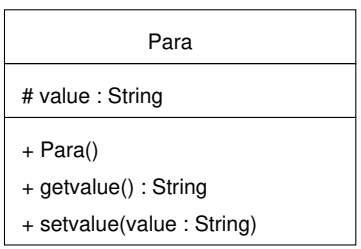
```
1   > xjc -p book -dtd book.dtd
2   parsing a schema...
3   compiling a schema...
4   book/Available.java
5   book/Book.java
6   book/Chapter.java
7   book/ObjectFactory.java
8   book/Para.java
```

```
1   <?xml version="1.0" encoding="ISO-8859-1"?> <!--
2
3   <!ELEMENT book (title,available?,chapter+)>
4   <!ELEMENT chapter (title,para*)>
5   <!ELEMENT available EMPTY>
6   <!ELEMENT title (#PCDATA)>
7   <!ELEMENT para (#PCDATA)>
8
9   <!ATTLIST book
10      id ID #REQUIRED
11      page-count CDATA #IMPLIED
12      version CDATA #FIXED "1.0">
13  <!ATTLIST chapter id ID #REQUIRED>
```









Exemple d'utilisation de JAXB

```
1   import javax.xml.bind.*;
2   import book.*;
3   public class Book1 {
4       public static void main(String[] args) throws Exception {
5           ObjectFactory fabrique = new ObjectFactory();
6           Book b1 = fabrique.createBook();
7           b1.setTitle("Livre␣très␣simple");
8           Chapter c1 = fabrique.createChapter();
9           c1.setTitle("Chapitre␣très␣court");
10          Para p1 = fabrique.createPara();
11          p1.setvalue("Bonjour␣tout␣le␣monde␣!");
12          c1.getPara().add(p1);
13          Para p2 = fabrique.createPara();
14          p2.setvalue("Ce␣est␣un␣autre␣paragraphe...");
15          c1.getPara().add(p2);
16          b1.getChapter().add(c1);
17          // Engendrer le fichier XML correspond
18          JAXBContext context = JAXBContext.newInstance("book");
19          Marshaller marshaller = context.createMarshaller();
20          marshaller.marshal(b1, System.out);
21          // Construire le modèle objet depuis un document XML
22          Unmarshaller unmarshaller = context.createUnmarshaller();
23          Book b2 = (Book) unmarshaller.unmarshal(new java.io.File("docbook-wikipedia-dtd.xml"));
24          System.out.println();
25          // Produire la table des matières
26          int numero = 0;
27          for (Chapter c : b2.getChapter()) {
28              System.out.println(++numero + "␣" + c.getTitle());
29  }   }   }
```

Sommaire

- Principales interfaces de l'API JDOM

abstract class org.jdom.Content

```
1  Parent getParent()
2  Content detach()
3  Document getDocument()
4
5  abstract String getValue()
6
7  final boolean equals(Object)
8  final int hashCode()
9  Object clone()
```

class org.jdom.Text extends Content

```
1  String getText()
2  Text setText(String)
3  String getTextTrim()
4  String getTextNormalize()   // .trim().replaceAll("\\s+", " ")
5  String toString()       // for debugging
6
7  void append(String str)
8  void append(Text text)
9
10 static static String normalizeString(String)
```

class org.jdom.Attribute extends Content

```
1  // Les essentielles !
2  String getName()
3  String getValue()
4  Attribute setName(String)
5  Attribute setValue(String)
6  // Le contexte
7  Element getParent()
8  Attribute detach()
9  Document getDocument()
10 // Les espaces de nom
11 String getQualifiedName()
12 String getNamespacePrefix()
13 String getNamespaceURI()
14 Namespace getNamespace()
15 Attribute setNamespace(Namespace)
16 // Exploitation de la valeur
17 int getAttributeType()
18 Attribute setAttributeType(int)
19 int getIntValue() throws DataConversionException
20 long getLongValue() throws DataConversionException
21 float getFloatValue() throws DataConversionException
22 double getDoubleValue() throws DataConversionException
23 boolean getBooleanValue() throws DataConversionException
```

interface org.jdom.Parent

```
1  // Le context
2  Parent getParent()
3  Document getDocument()
4
5  // Accès au contenu
6  int getContentSize()
7  int indexOf(Content)
8  Content getContent(int)
9  List getContent()
10 List getContent(filter.Filter)
11
12 Iterator getDescendants()
13 Iterator getDescendants(filter.Filter)
14
15 // Supprimer le contenu
16 List removeContent()
17 List removeContent(filter.Filter)
18 boolean removeContent(Content)
19 Content removeContent(int)
20
21 Object clone()      // redéfinition de Object.clone()
22 List cloneContent()
```

class org.jdom.Element extends Content implements Parent

```
1  String getName()        // Le nom de l'élément
2  Element setName(String)
3  boolean isAncestor(Element elt) // this est un ancêtre de elt
4  boolean isRootElement()
5
6  // Les attributs de cet élément
7  List getAttributes()
8  Attribute getAttribute(String)
9  String getAttributeValue(String name)
10 Element setAttributes(Collection)
11 Element setAttributes(List)
12 Element setAttribute(String name, String value)
13 Element setAttribute(Attribute)
14 boolean removeAttribute(String name)
15 boolean removeAttribute(Attribute)
16
17 // Pour le texte (#PCDATA)  (voir Text)
18 String getText()
19 Element setText(String)         String getChildText(String)
20 String getTextTrim()            String getChildTextTrim(String)
21 String getTextNormalize()       String getChildTextNormalize(String)
22
23 // Ajouter un contenu
24 Element addContent(String str)  // ajouter un contenu Text
25 Element addContent(Content)
26 Element addContent(Collection newContent)
27 Element addContent(int index, Content child)
```

class org.jdom.Element extends Content implements Parent (2)

```
28  Element addContent(int index, Collection newContent)
29
30  // Changer le contenu
31  Element setContent(Collection newContent)
32  Element setContent(int index, Content child)
33  Parent setContent(int index, Collection newContent)
34  Element setContent(Content)
35
36  // Accéder aux fils Element
37  List getChildren()
38  List getChildren(String)
39  List getChildren(String,Namespace)
40  Element getChild(String,Namespace)
41  Element getChild(String)
42
43  // Supprimer des fils
44  boolean removeChild(String)
45  boolean removeChild(String,Namespace)
46  boolean removeChildren(String)
47  boolean removeChildren(String,Namespace)
48
49  // Pour les espaces de nom
50  Namespace getNamespace()
51  Namespace getNamespace(String)
52  Element setNamespace(Namespace)
53  ...
```

class org.jdom.Document implements Parent

```
1  void setProperty(String,Object)
2  Object getProperty(String)
3
4  // L'élément racine
5  boolean hasRootElement()
6  Element getRootElement()
7  Document setRootElement(Element)
8  Element detachRootElement()
9
10 // Définir la DTD
11 DocType getDocType()
12 Document setDocType(DocType)
13
14 // Ajouter un contenu
15 Document addContent(Content)
16 Document addContent(Collection)
17 Document addContent(int,Content)
18 Document addContent(int,Collection)
19
20 // Changer le contenu
21 Document setContent(Collection)
22 Document setContent(int,Content)
23 Document setContent(int,Collection)
24 Document setContent(Content)
```