Programmation avancée

NFP 121

XML et Java 2007

XML pourquoi faire ?

Structuration des données

Titre	XML: Des BD aux Services Web
Auteur	Georges Gardarin
Section	1. Introduction
Paragraphe	Ces dernières années ont vu l'ouverture des systèmes d'information à l'Internet. Alors que depuis les années 1970, ces systèmes se développaient souvent par applications plus ou moins autonomes, le choc Internet ...
Paragraphe	Ainsi, on a vu apparaître une myriade de technologies nouvelles attrayantes mais peu structurantes voir perturbantes. Certaines n'ont guère survécues. D'autres ont laissé des systèmes peu fiables et peu sécurisés. ...
Paragraphe	L'urbanisation passe avant tout par la standardisation des échanges : il faut s'appuyer sur des standards ouverts, solides, lisibles, sécurisés, capable d'assurer l'interopérabilité avec l'Internet et les systèmes d'information. XML, "langua franca" ...

Origines

Vue Balisée en XML

```
<Livre>
<Titre> XML : Des BD aux Services Web </Titre>
<Auteur>Georges Gardarin</Auteur>
<Section titre = "Introduction">
<Paragraphe>Ces dernières années ont vu l'ouverture des systèmes d'information
    à l'Internet. Alors que depuis les années 1970, ces systèmes se développaient
    souvent par applications plus ou moins autonomes, le choc Internet ...
    </Paragraphe>
<Paragraphe>Ainsi, on a vu apparaître une myriade de technologies nouvelles
    attrayantes mais peu structurantes voir perturbantes. Certaines n'ont guère
    survécues. D'autres ont laissé des systèmes peu fiables et peu sécurisés.
    ...</Paragraphe>
<Paragraphe>L'urbanisation passe avant tout par la standardisation des échanges :
    il faut s'appuyer sur des standards ouverts, solides, lisibles, sécurisés, capable
    d'assurer l'interopérabilité avec l'Internet et les systèmes d'information. XML,
    "langua franca" ... </Paragraphe>
</Section>
</Livre>
```

Origines

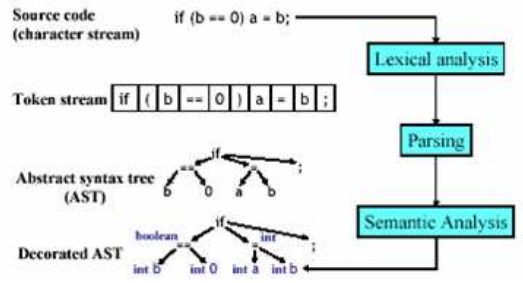
XML pourquoi faire ?

Définir un langage

```
<?xml version="1.0"?>
<!– ANT build file -->
<project name="tutorial" default="build" basedir=".">
    <target name="build">
        <javac srcdir="." />
    </target>
</project>
```


XML pourquoi faire ?

Disposer des parseurs XML



Exemple:

Le document ANT est 'parsé' et interprété

Lets run the file:

>ant

Buildfile: build.xml

build: [javac] Compiling 1 source file

BUILD SUCCESSFUL Total time: 3 seconds

XML : la famille

- Né : fin 96
- Père : W3C
- Petit-fils de SGML (ISO-1986)
- Cousin d'HTML
- Reconnu le : 10/02/98 – version 1.0
- Descendance – XHMTL, MathML, ANT…

-7-

X comme eXtensible

- HTML : nombre fini de balises
- XML : possibilité de définir les balises

- HTLM : balises pour formater
- XML : balises pour structurer

- DTD ou Schéma pour définir les balises

-8-

Règles syntaxiques

1. Commencer par une déclaration XML
2. Balisage sensible à la casse
3. La valeur des attributs doit être quotée
4. Balises non vides appariées
</br>
5. Balises vides fermées
6. Les élèments ne doivent pas se chevaucher
   <jour> <mois> </jour> </mois> interdit
7. Un élément doit encapsuler tous les autres
8. Ne pas utiliser les caractères < et & seuls

-9-

Le Prologue

- Une déclaration XML

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

- Instructions de traitement (PI Processing Instruction)
  - Une indication de traitement est destinée aux applications (ex. XSL) qui manipulent les documents XML
- Une déclaration de type de document
  - indique le type de document auquel se conforme le document en question (ex. DTD)

```
<!DOCTYPE rapport SYSTEM "rapport.dtd">
```

-10-

Élément

- Composant de base
- Identifié par un nom
- Délimité par une balise ouvrante et une balise fermante

  <AUTEUR> Victor Hugo </AUTEUR>
- Ou élément vide

  <PHOTO Source= "victor.gif" />
- Contenu textuel, éléments ou mixte

-11-

Les attributs

- Inclus dans la balise ouvrante d'un élément
- Composé d'un nom et d'une valeur

```
<AUTEUR NE="1802" MORT="1885" >
   Victor Hugo
</AUTEUR>
```

-12-

Données

- Constituées par un flot de caractères
  - tous les caractères sont acceptés sauf le caractère « & » et le caractère « < »
  - Exemple : `<auteurs>Victor Hugo<auteurs>`
- Si l 'on souhaite insérer des caractères « spéciaux », il est préférable d'utiliser une section littérale ou CDATA

```
<![CDATA[<auteurs>S. Fleury &amp; al.</auteurs>]]>
```

qui se traduit en :

```
<auteurs>S. Fleury &amp; al.</auteurs>
```

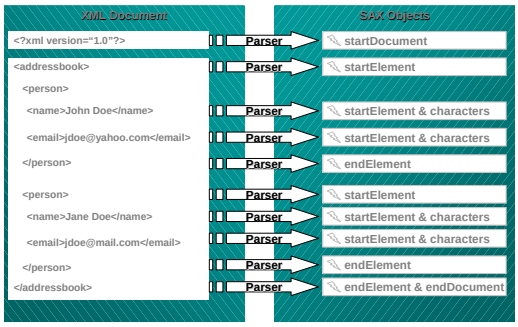
-13-

SAX
Simple Api for Xml

- Début des travaux Dec, 1997
- SAX 1.0 Mai, 1998
  - Tim Bray
  - David Megginson
  - …
- SAX 2.0 Mai, 2000
- SAX 2.0.2 *27-April 2004:*

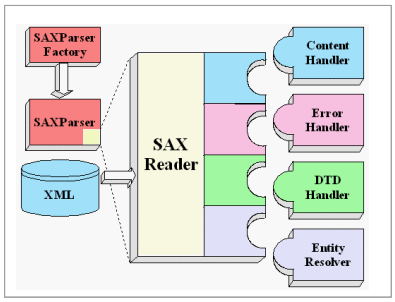
-14-

SAX Comment ça marche ?



-15-

Implémenter les Handlers d'évènements du parseur



DefaultHandler

Il implémente ces différents Handler avec des méthodes vides, de sorte que l'on peut surcharger seulement celles qui nous intéressent.

-16-

Structure du main

```
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.XMLReader;
public class SurveyReader extends DefaultHandler{
    public static void main (String args[]) {
        XMLReader xmlReader = null;
        try { SAXParserFactory spfactory = SAXParserFactory.newInstance();
            SAXParser saxParser = spfactory.newSAXParser();
            xmlReader = saxParser.getXMLReader();
            xmlReader.setContentHandler(new SurveyReader());
            InputSource source = new InputSource("surveys.xml");
            xmlReader.parse(source);
        } catch (Exception e) { System.err.println(e); System.exit(1);
        }
    }
}
```

-17-

org.xml.sax.ContentHandler

- Toutes les applications SAX doivent implanter un ContentHandler
- Méthodes :
  - public void **startDocument**() throws SAXException
  - public void **endDocument**() throws SAXException
  - public void **startElement**(String nspURI, String localName, String qName, Attributes atts) throws SAXException
  - public void **characters**(char[] ch, int start, int length) throws SAXException
  - …

-18-

Récupérer le début d'analyse de chaque élément

```
...
import org.xml.sax.Attributes;
public class SurveyReader extends DefaultHandler {
    ...
    public void startDocument() throws SAXException {
        System.out.println("Tallying survey results...");
    }
    public void startElement( String namespaceURI, String localName,
                              String qName, Attributes atts) throws SAXException {
        System.out.print("Start element: ");
        System.out.println(qName);
    }
...
}
```


Exemple de traitement SAX: startElement(…)

```
<?xml version="1.0"?>
<surveys>
<response username="bob">
<question subject="appearance">A</question>
<question subject="communication">B</question>
<question subject="ship">A</question>
<question subject="inside">D</question>
<question subject="implant">B</question>
</response>
<response username="sue">
<question subject="appearance">C</question>
<question subject="communication">A</question>
<question subject="ship">A</question>
<question subject="inside">D</question>
<question subject="implant">A</question>
</response>
…. </surveys>
```

```
Tallying survey results...
Start element: surveys
Start element: response
Start element: question
Start element: question
Start element: question
Start element: question
Start element: question
Start element: response
Start element: question
Start element: question
Start element: question
Start element: question
Start element: question
Start element: response
Start element: question
Start element: question
Start element: question
Start element: question
Start element: question
```


Récupérer les données

```
public void characters(char[] ch, int start, int length)
throws SAXException {

    if (thisElement == "question") {
    printIndent(4);
    System.out.print(thisQuestion + ": ");
    System.out.println(new String(ch, start, length));
}
}
...
```


exemple

```
Tallying survey results...
User: bob
    appearance: A
    communication: B
    ship: A
    inside: D
    implant: B
User: sue
    appearance: C
    communication: A
    ship: A
    inside: D
    implant: A
User: carol
    appearance: A
    communication: C
    ship: A
    inside: D
    implant: C
```


Valider un fichier XML

```
...
public static void main (String args[]) {
    XMLReader xmlReader = null;
    try {
        SAXParserFactory spfactory =
        SAXParserFactory.newInstance();
        spfactory.setValidating(true);
        SAXParser saxParser =
        spfactory.newSAXParser();
        xmlReader = saxParser.getXMLReader();
    } catch (Exception e) {
    System.err.println(e); System.exit(1);
    }
}
```


Qu'est qu'une DTD ?

D'après *F. Nolot*

- **Elle permet de vérifier qu'un document XML est conforme à une syntaxe donnée (à une grammaire)**
- **On distingue 2 types de conformité**
  - **Les documents valides : les documents XML avec une DTD**
  - **Les documents bien formés : les documents XML ne comportant pas de DTD mais répondant aux règles de base du XML**
- **Une DTD peut être définie de 2 façons**
  - **Sous forme interne, incluant la grammaire dans le document**
  - **Sous forme externe, soit en appelant un fichier contenant la grammaire à partir d'un fichier local ou bien en y accédant par son URL**

Décalaration d'une DTD interne

```
<?xml version="1.0" ?>
<!DOCTYPE purchase_order [
<!ELEMENT purchase_order (customer)>
<!ELEMENT customer (account_id, name)>
<!ELEMENT account_id (#PCDATA)>
<!ELEMENT name (first, mi, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT mi (#PCDATA)>
<!ELEMENT last (#PCDATA)>
]>
<purchase_order>
<customer>
<account_id>10-487</account_id>
<name>
<first> William </first>
<mi> R </mi>
<last> Stanek </last>
</name>
</customer>
</purchase_order>
```


Déclaration d'une DTD externe

- Dans un document XML, dans le cas de l'utilisation d'une DTD externe, on doit alors avoir :standalone="no"
- Puis l'élément <!DOCTYPE elt_racine SYSTEM "filename.dtd">

- **<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>**
- **<!DOCTYPE carnet SYSTEM "../Cours5-Solution/Exo1.dtd">**
- **<carnet>**
- **<NomPersonne>**
- **<M/>**
- **<Prenom>George</Prenom>**
- **<Nom>FILOCHE</Nom>**
- **</NomPersonne>**
- **<NomPersonne>**
- **<Mlle/>**
- **<Prenom>Martine</Prenom>**
- **<Prenom2>Yvonne</Prenom2>**
- **<Nom>GETUDAVE</Nom>**
- **</NomPersonne>**
- **</carnet>**

Déclaration d'une DTD

- <!DOCTYPE elt_racine SYSTEM|PUBLIC emplacement1 emplacement2>

- SYSTEM s'utilise avec une DTD externe dont l'emplacement est
  - Soit une URL http://www.....
  - Soit une URI file:///home/user/DTD/.....

- PUBLIC permet l'utilisation d'une référence générique à la DTD par l'intermédiaire d'un URI, voire un second URI
- Plutôt utiliser avec des DTD normalisées disponibles au public
- Exemple : validité d'un document HTML4
- <!doctype html public '-//W3C//DTD HTML 4.0//EN' 'http://www.w3.org/TR/REC-html40/strict.dtd'>

Exemple de DTD

- Définissons la conformité d'un fichier XML qui contient des informations concernant une personne

- les informations suivantes sont supposées nécessaires
  - personne:
  - Nom
  - Prénom
  - Téléphone

  - L'élément email est optionnel

- **<!ELEMENT personne (nom,prenom,telephone,email?) >**
- **<!ELEMENT nom (#PCDATA) >**
- **<!ELEMENT prenom (#PCDATA) >**
- **<!ELEMENT telephone (#PCDATA) >**
- **<!ELEMENT email (#PCDATA) >**

Explication de la syntaxe

- Déclaration d'un élément
  - **<! ELEMENT nom modèle>**
- Le paramètre modèle représente soit un type de données prédéfinies, soit une règle d'utilisation de l'élément

- Les types prédéfinis utilisables sont les suivants :
  - **ANY** : l'élément peut contenir tout type de données. A utiliser avec précaution car il supprime quasiment tout contrôle de validité
  - **EMPTY** : l'élément ne contient pas de données spécifiques
  - **#PCDATA** : l'élément doit contenir une chaîne de caractère

- Un élément (produit) qui ne doit contenir que dew caractères (#PCDATA), sera définit de la façon suivante :
  - **<!ELEMENT produit (#PCDATA)>**

Spécifications éléments

(#PCDATA)	Parsed Character DATA
(ELT)	1 fois ELT
(ELT1,ELT2)	Séquence
(ELT1|ELT2|…)	Choix
ELT?	0 ou 1 fois ELT
ELT+	au moins 1 fois ELT
ELT*	0 ou plusieurs fois ELT
()	groupe de sous éléments
ANY	n'importe quoi
EMPTY	rien

Contenu mixte

- Si un élément peut contenir à la fois des caractères et d'autres éléments Le mot-clé #PCDATA doit être en début de la liste des éléments
- <!ELEMENT personne (#PCDATA,nom,prenom,telephone)>
- Les modèles de contenu mixte n'accepte ni de suites d'éléments enfants, ni de choix d'opérateurs de cardinalité. L'ordre utilisé n'a donc pas d'importance sur les éléments suivants #PCDATA

Exemple

- **Un élément NomPersonne est composé**
  - **Soit d'un sigle M, Mme, Mlle**
  - **D'un prénom**
  - **D'un 2ième prénom**
  - **Et d'un nom de famille**
- **Ce qui donne**
- **<!ELEMENT NomPersonne ((M | Mme | Mlle), Prenom, Prenom2, Nom) >**
- **<!ELEMENT M EMPTY>**
- **<!ELEMENT Mme EMPTY>**
- **<!ELEMENT Mlle EMPTY>**
- **<!ELEMENT prenom (#PCDATA) >**
- **<!ELEMENT prenom2 (#PCDATA) >**
- **<!ELEMENT nom (#PCDATA) >**
- **Le document suivant est donc conforme**
- **<NomPersonne>**
- **<M/>**
- **<Prenom>John</Prenom>**
- **<Prenom2>Edouard</Prenom2>**
- **<Nom>Martin</Nom>**
- **</NomPersonne>**

Déclaration de Listes d'attributs

```
<!ATTLIST NomElement
       NomAttribut1 TypeAttribut1 Mode1
       ...
       NomAttributN TypeAttributN ModeN>
```

- Type
  - **CDATA** : chaîne de caractères prise telle quelle
  - **ID** ou **IDREF** : clé ou référence à clé
  - **NMTOKEN** ou **NMTOKENS** : noms symboliques formés de caractère alphanumériques (1 ou plusieurs)
  - **(valeurA |valeurB | ...)** : valeurs de l 'attribut au choix dans la liste
- Mode (précise la caractère obligatoire ou non de l 'attribut)
  - **"valeur"** : valeur par défaut si non spécifié
  - **#REQUIRED** : attribut obligatoirement présent
  - **#IMPLIED** : présence de l'attribut facultative
  - **#FIXED "valeur"** : l'attribut prend toujours cette valeur

Exemple

```
<!ELEMENT personne (nom, prenom+, tel?, adresse) >
<!ATTLIST personne
                    num ID,
                    age CDATA,
                    genre (Masculin|Feminin)
   "Masculin">

<!ELEMENT auteur (#PCDATA) >
<!ELEMENT editeur (#PCDATA) >
<!ATTLIST auteur
                   genre (Masculin|Feminin ) #REQUIRED
                   ville CDATA #IMPLIED>
<!ATTLIST editeur
                    ville CDATA #FIXED "Paris">
```


Entités Paramètres

- Permet la définition d'un groupe d'éléments sous un nom (macro)
  - <!ENTITY % nom "definition">
- Réutilisable dans une DTD par simple appel :
  - %nom;
- Exemple :

  <!ENTITY % genres "(homme | femme)">
  <!ATTLIST auteur genre %genres; #REQUIRED>
- Peuvent être externes :

  <!ENTITY %mpeg SYSTEM "http://www.myweb.fr">

Entités Générales

- Permet la définition d'un texte sous un nom

  <!ENTITY nom "texte">
- Réutilisable dans un document par simple appel :

  %nom;
- Exemple

```
<?xml version="1.0"?>
<!DOCTYPE exemple [
   <!ELEMENT exemple (#PCDATA, important)>
   <!ELEMENT important (#PCDATA)>
   <!ENTITY cie "Les Vignerons Réunis">
   <!ENTITY imp "<important>Attention!</important>"> ]>

<exemple> &cie; &imp; </exemple>
```

Exemple d'Entités

- Définition d 'entité dans le DTD

```
<!ENTITY chap1 SYSTEM "chap1.xml">
<!ENTITY chap2 SYSTEM "chap2.xml">
<!ENTITY auteur "SF">
```

- Référence dans le XML

```
<livre>
   <titre>Mon livre</livre>
   <auteur>&SF;</auteur>
   &chap1;
   &chap2;
</livre>
```

-37-

Namespaces XML

espace de noms
modularité

-38-

Combiner deux documents xml ?

```
<?xml version="1.0" ?>
 <Address>
    <Street>10st.</Street>
    <City>Orange_County</City>
    <State>CA</State>
    <Country>States</Country>
    <PostalCode> CA92657 </PostalCode>
</Address>

 et:

<?xml version="1.0" ?>
<Server>
<Name>OurWebServer</Name>
 <Address>123.45.67.8</Address>
</Server>
```

-39-

Solution au conflit

- Renomer les deux Address dans des noms différents
  - Pas général
  - Peut poser des problèmes au code déjà écrit
- Assigner au deux documents des espaces de nommage différents (modules)

-40-

Solution

```
<addr:Address
   xmlns:addr="http://www.server.Addresses.com">
   <addr:Street>10st</addr:Street>
   <addr:City>OrangeCounty</addr:City>
   <addr:State>CA</addr:State>
   <addr:Country>States</addr:Country>
    <addr:PostalCode>CA92657</addr:PostalCode>
</addr:Address>

<serv:Server xmlns:serv="http://www.server.com">
   <serv:Name>OurWebServer</serv:Name>
   <serv:Address> 123.45.67.8 </serv:Address>
 </serv:Server>
```

-41-

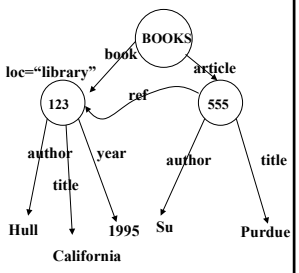
Declarer les Namespaces

```
<element xmlns:prefix="Namespace URI">
                    Optional
Element  Attribute    Prefix     Namespace name

<Micro:A
   xmlns:Micro="http://www.Microsoft.com/"
   >
   <Micro:B>abcd</Micro:B>
 </Micro:A>
```

-42-

JDOM
Représentation Arborescente des documents xml

```
<BOOKS>
<book id="123" loc="library">
  <author>Hull</author>
  <title>California</title>
  <year> 1995 </year>
</book>
<article id="555" ref="123">
  <author>Su</author>
  <title> Purdue</title>
</article>
</BOOKS>
```



De DOM à JDOM

DOM est l'acronyme de ***Document Object Model***. C'est une spécification du W3C pour proposer une API qui permet de modéliser, de parcourir et de manipuler un document XML. Le principal rôle de DOM est de fournir une représentation mémoire d'un document XML sous la forme d'un arbre d'objets et d'en permettre la manipulation (parcours, recherche et mise à jour).
DOM est défini pour être indépendant du langage dans lequel il sera implémenté. DOM n'est donc pas spécifique à Java.

JDOM manipule les éléments d'un Document Object Model spécifique (créé grâce à un constructeur basé sur SAX).

JDOM permet donc de construire des documents, de naviguer dans leur structure, d'ajouter, de modifier, ou de supprimer leur contenu.

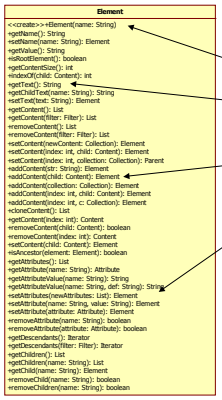
Créer un fichier XML avec JDOM

```
<personnes>
   <etudiant classe="P2">
      <nom>CynO</nom>
   <etudiant>
<personnes>
```


JDOM1.java

```
import java.io.*;
import org.jdom.*;
import org.jdom.output.*;
public class JDOM1{
    static Element racine = new Element("personnes");
    static org.jdom.Document document = new Document(racine);
    public static void main(String[] args){
        Element etudiant = new Element("etudiant");
        racine.addContent(etudiant);
        Attribute classe = new Attribute("classe","P2");
        etudiant.setAttribute(classe);
        Element nom = new Element("nom");
        nom.setText("CynO");
        etudiant.addContent(nom);
        affiche();
        enregistre("Exercice1.xml");
    }
}
```


Element



Créer un élément

Lui affecter un texte

Ajouter un fils

Ajouter un attribut

```
<personnes>
        <etudiant classe="P2">
            <nom>CynO</nom>
        <etudiant>
<personnes>
```


Document

Attributs

Attribute

<<create>>+Attribute(name: String, value: String, namespace: Namespace)
<<create>>+Attribute(name: String, value: String, type: int, namespace: Namespace)
<<create>>+Attribute(name: String, value: String)
<<create>>+Attribute(name: String, value: String, type: int)
+getParent(): Element
+getDocument(): Document
+detach(): Attribute
+getName(): String
+setName(name: String): Attribute
+getQualifiedName(): String
+getNamespacePrefix(): String
+getNamespaceURI(): String
+getNamespace(): Namespace
+setNamespace(namespace: Namespace): Attribute
+getValue(): String
+setValue(value: String): Attribute
+getAttributeType(): int
+setAttributeType(type: int): Attribute
+toString(): String
+equals(ob: Object): boolean
+hashCode(): int
+clone(): Object
+getIntValue(): int
+getLongValue(): long
+getFloatValue(): float
+getDoubleValue(): double
+getBooleanValue(): boolean

AttributeList

<<create>>~AttributeList(parent: Element)
~uncheckedAddAttribute(a: Attribute)
+add(obj: Object): boolean
+add(index: int, obj: Object)
~add(index: int, attribute: Attribute)
+addAll(collection: Collection): boolean
+addAll(index: int, collection: Collection): boolean
+clear()
~clearAndSet(collection: Collection)
+get(index: int): Object
~get(name: String, namespace: Namespace): Object
~indexOf(name: String, namespace: Namespace): int
+remove(index: int): Object
~remove(name: String, namespace: Namespace): boolean
+set(index: int, obj: Object): Object
~set(index: int, attribute: Attribute): Object
+size(): int
+toString(): String

Éditer le document

```
static void affiche(){
      try{
            XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
            sortie.output(document, System.out);
      }catch (java.io.IOException e){}
}

static void enregistre(String fichier){
      try{
            XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
            //Remarquez qu'il suffit simplement de créer une instance de
            // FileOutputStream avec en argument le nom du fichier
            // pour effectuer la sérialisation.
            sortie.output(document, new FileOutputStream(fichier));
      }catch (java.io.IOException e){}
}
```


XMLOutputter

<<create>>+XMLOutputter()
<<create>>+XMLOutputter(format: Format)
<<create>>+XMLOutputter(that: XMLOutputter)
+setFormat(newFormat: Format)
+getFormat(): Format
+output(doc: Document, out: OutputStream)
+output(doctype: DocType, out: OutputStream)
+output(element: Element, out: OutputStream)
+outputElementContent(element: Element, out: OutputStream)
+output(list: List, out: OutputStream)
+output(cdata: CDATA, out: OutputStream)
+output(text: Text, out: OutputStream)
+output(comment: Comment, out: OutputStream)
+output(pi: ProcessingInstruction, out: OutputStream)
+output(entity: EntityRef, out: OutputStream)
+output(doc: Document, out: Writer)
+output(doctype: DocType, out: Writer)
+output(element: Element, out: Writer)
+outputElementContent(element: Element, out: Writer)
+output(list: List, out: Writer)
+output(cdata: CDATA, out: Writer)
+output(text: Text, out: Writer)
+output(comment: Comment, out: Writer)
+output(pi: ProcessingInstruction, out: Writer)
+output(entity: EntityRef, out: Writer)
+outputString(doc: Document): String
+outputString(doctype: DocType): String
+outputString(element: Element): String
+outputString(list: List): String
+outputString(cdata: CDATA): String
+outputString(text: Text): String
+outputString(comment: Comment): String
+outputString(pi: ProcessingInstruction): String
+outputString(entity: EntityRef): String
+escapeAttributeEntities(str: String): String
+escapeElementEntities(str: String): String
+clone(): Object
+toString(): String

Parser et valider un fichier xml

```
import org.jdom.*;

public class  saxValidate{
  public static void main(String[] args){
    org.jdom.input.SAXBuilder builder =
       new org.jdom.input.SAXBuilder(true);
    try{  Document foo = builder.build("rootBeer.xml");
          System.out.println("XML loaded fine!");
    }
    catch (org.jdom.JDOMException e){
      System.out.println("Error loading XML: " + e.getMessage() );
    }
  }
}
```


exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rootBeer SYSTEM "rootBeer.dtd">
<rootBeer>
  <Foo date="20011021" city="Scottsdale">Look at me.</Foo>
  <Bar>Look at me.</Bar>
</rootBeer>
```

```
C:\dev\topxml\jdom\testcode>java saxValidate
Error loading XML:
Error on line 2 of document file:/C:/dev/topxml/jdom/testcode/roo
tBeer.xml:
External entity not found: "file:/C:/dev/topxml/jdom/testcode/root
Beer.dtd".
```

Avec DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rootBeer SYSTEM "rootBeer.dtd">
<rootBeer>
    <Foo date="20011021" city="Scottsdale">Look at me.</Foo>
    <Bar>Look at me.</Bar>
</rootBeer>
```

```
<!ELEMENT rootBeer (Foo, Bar) >
<!ELEMENT Foo (#PCDATA) >
<!ELEMENT Bar (#PCDATA) >
<!ATTLIST Foo date CDATA #REQUIRED >
<!ATTLIST Foo city CDATA #IMPLIED >
```


Autres erreurs

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rootBeer SYSTEM "rootBeer.dtd">
<rootBeer>
  <Foo date="20011021" city="Scottsdale">Look at me.</Foo>
  <Bar>Look at me.</Bar>
  <Blug>This is wrong</Blug>
</rootBeer>
```

```
C:\dev\topxml\jdom\testcode>java saxValidate
Error loading XML:
Error on line 6 of document  file:/C:/dev/topxml/jdom/testcode/rootBeer.xml:
Element "rootBeer" allows no further input; "Blug" is not allowed.
```


Modifier un fichier xml

```
<?xml version="1.0" encoding="UTF-8"?>
<personnes>
    <etudiant classe="P2">
        <nom>CynO</nom>
        <prenoms>
            <prenom>Nicolas</prenom>
            <prenom>Laurent</prenom>
        </prenoms>
    </etudiant>
    <etudiant classe="P1">
        <nom>Superwoman</nom>
    </etudiant>
    <etudiant classe="P1">
        <nom>Don Corleone</nom>
    </etudiant>
</personnes>
```


JDom3.java

```
import java.io.*;
…
import java.util.Iterator;
public class JDom3{
static org.jdom.Document document;
static Element racine;
public static void main(String[] args){
    try{
        lireFichier("Exercice 2.xml");
        supprElement("prenoms");
        enregistreFichier("Exercice 2.xml");
    }catch(Exception e){}
}
//On parse le fichier et on initialise la racine de notre arborescence
static void lireFichier(String fichier) throws Exception{
    SAXBuilder sxb = new SAXBuilder();
    document = sxb.build(new File(fichier));
    racine = document.getRootElement();
}
```


suite

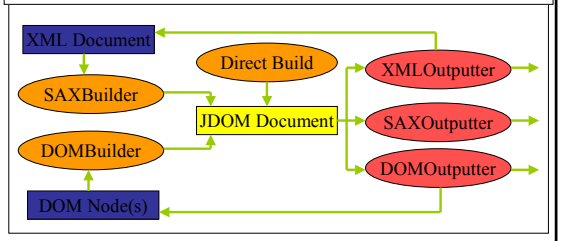
```
static void supprElement(String element){
    List listEtudiant = racine.getChildren("etudiant"); //lister les étudiants
    Iterator i = listEtudiant.iterator();
    while(i.hasNext()){ //On parcours la liste grâce à un iterator
        Element courant = (Element)i.next();
        if(courant.getChild(element)!=null){
            courant.removeChild(element);
            courant.setName("etudiant_modifie");
        }
    }
}

static void enregistreFichier(String fichier) throws Exception {
    XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat());
    sortie.output(document, new FileOutputStream(fichier));
}
}
```


résultat

```
<?xml version="1.0" encoding="UTF-8"?>
<personnes>
    <etudiant_modifie classe="P2">
        <nom>CynO</nom>
    </etudiant_modifie>
    <etudiant classe="P1">
        <nom>Superwoman</nom>
    </etudiant>
    <etudiant classe="P1">
        <nom>Don Corleone</nom>
    </etudiant>
</personnes>
```

Structure générale

- Normally XML Document -> SAXBuilder -> XMLOutputter



-61-