

Programmation XML en Java

Yves Bekkers

XML et Java

- Java 1.5 : JAXP
 - Java Api for Xml Parsing (*Processing*)

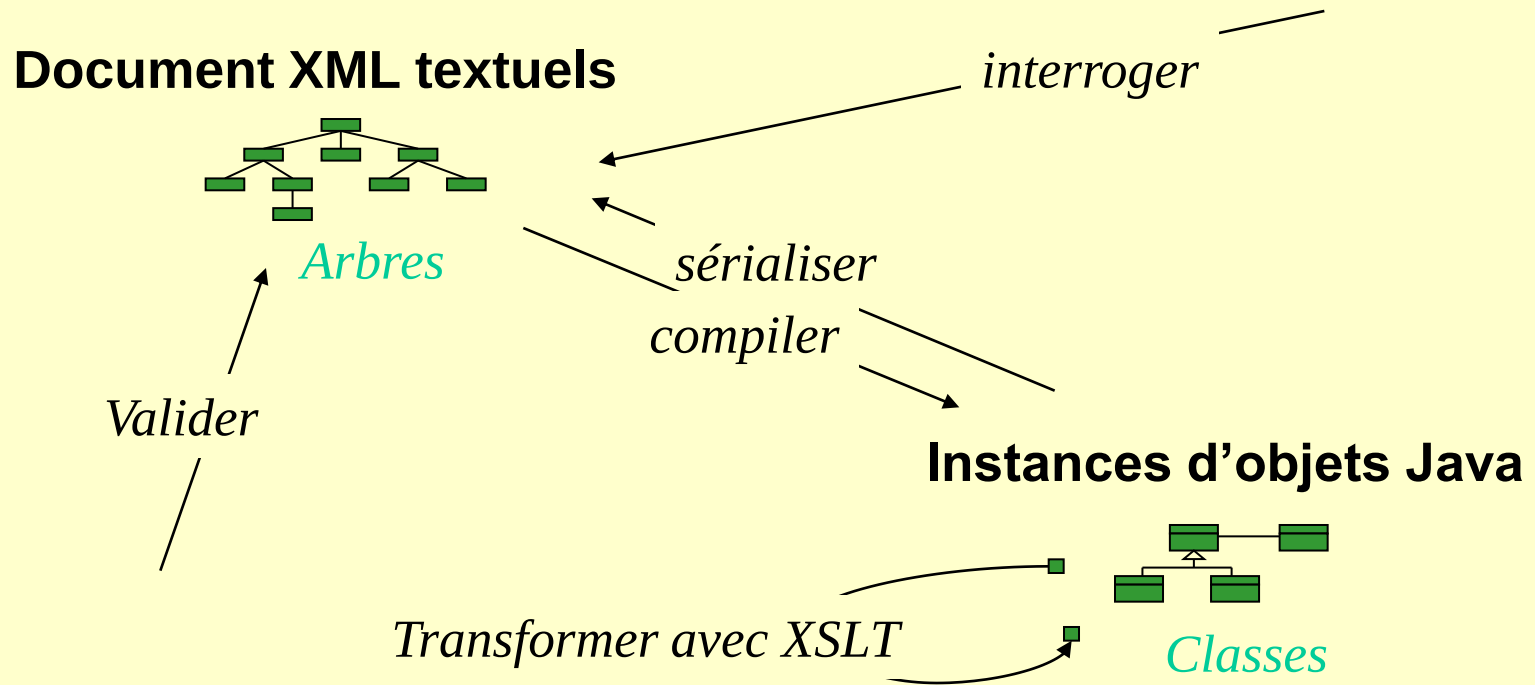
JAXP

- Un ensemble de paquetages **Java**
 - Avant Java 1.4 : librairies externes à Java
 - Depuis la version 1.4 : Intégrées à **JDK**,
- Objectif
 - Traitement en Java de documents **XML**.
- Nom de code (donné par Sun)
 - **JAXP** Java API for XML Processing (*Parsing*)

JAXP

- Quatre standards intégrés aux bibliothèques officielles de Java
 - SAX, Simple API for XML
 - DOM, Document Object Model
 - XSLT, Transformations XSLT
 - XPath, langage de repérage dans un arbre

Manipuler des documents XML par programme

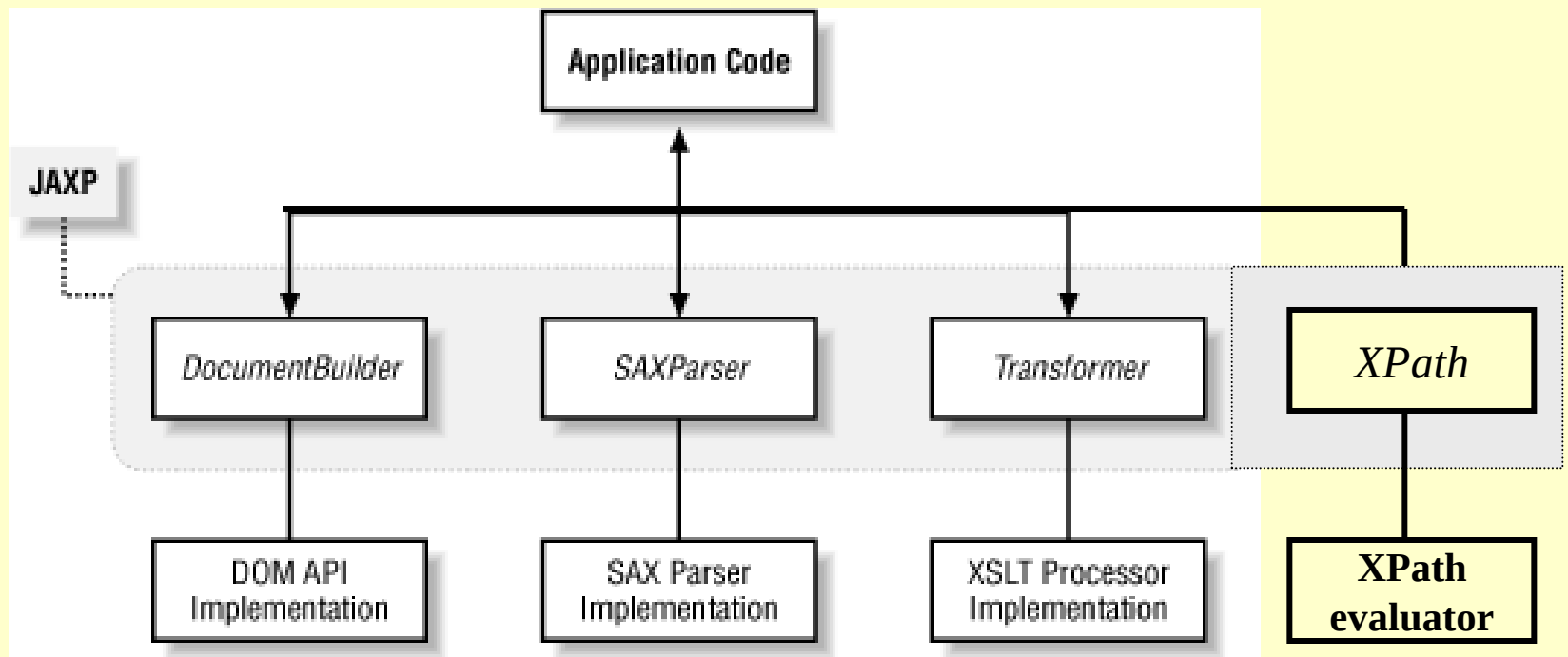


Objectifs des librairies XML

- Manipuler des documents XML en java
 - *Importer* des documents XML (à partir de fichiers)
 - *Valider* des documents (DTD, XMLSchema, ...)
 - *Construire* des documents à partir de rien en mémoire
 - *Parcourir*
 - *Modifier* ponctuellement
 - *Extraire* de l'information à l'aide d'XPath, Xquery ...
 - *Transformer* à l'aide de feuilles de style XSLT
 - *Exporter* (sérialiser dans des fichiers)

JAXP

- Des Interfaces d'utilisation des outils java externes
 - DOM, SAX, XSLT, XPATH
- Des outils internes

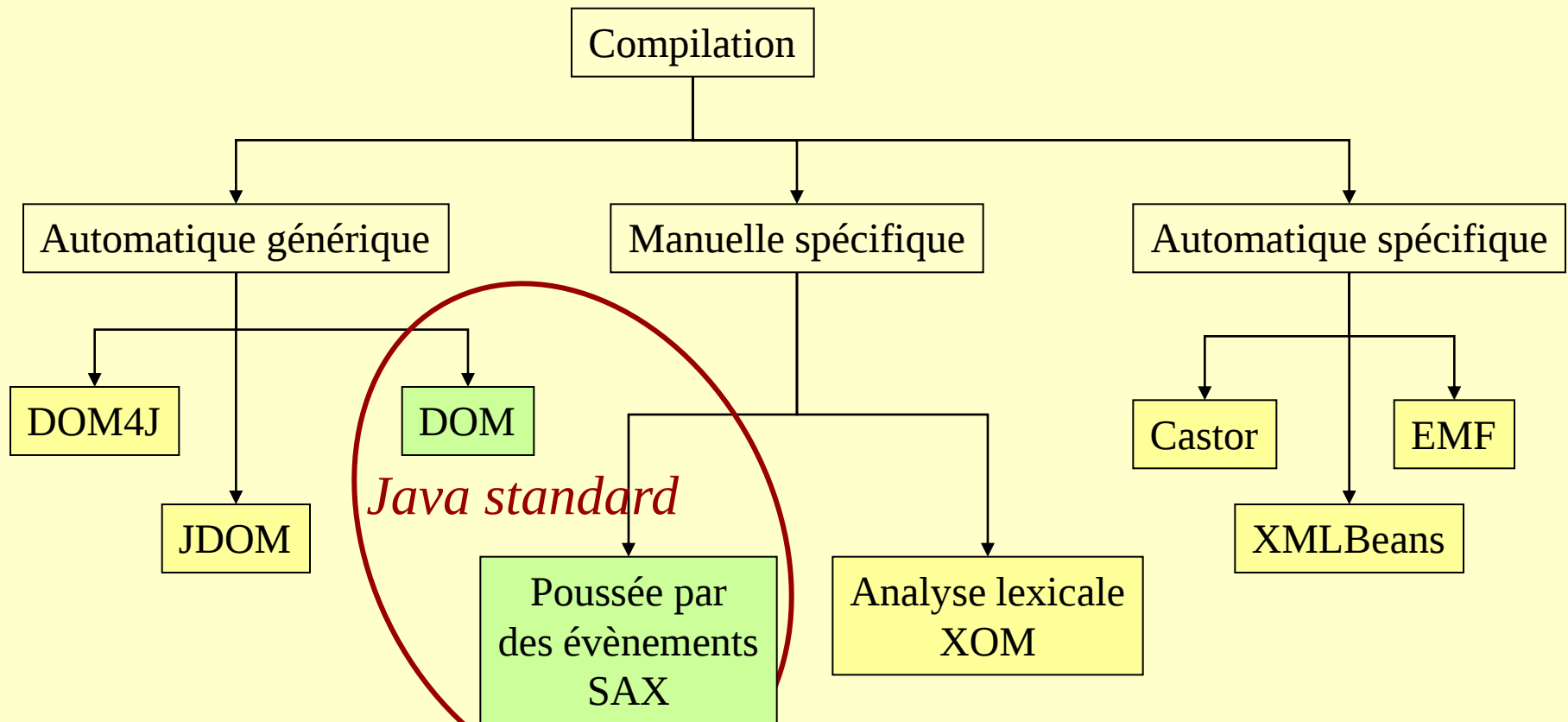


JAXP – son évolution

JAXP	
1.0	compilation seulement DOM1, SAX1, espace de nom
1.1	Transformation XSLT (Trax) DOM2, SAX2
1.2	Validation par XML schéma
1.3	Intégré à Java 1.5 Sélection Xpath Prise en compte d'Xinclude
1.4	en cours de développement Sérialisation binaire

Compiler un document XML

3 méthodes pour compiler un document XML



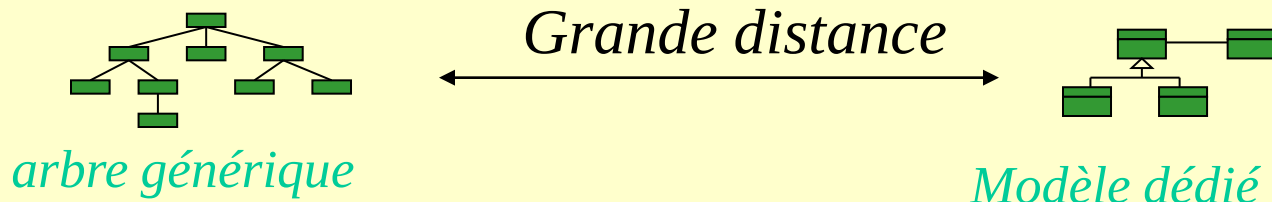
Importer/compiler exporter/sérialiser

- Utiliser un outil clé en main
 - DOM Document Object Model, Norme conçue par le W3C Solution
 - Générique, indépendante des langages : Javascript, C, Java, Perl, Latex ...
 - Construire rapidement un arbre de nœuds à partir d'un document textuel et vis-versa
- Fabriquer compilateur et sérialiseur
 - SAX Simple API for XML (pas de sérialiseur)
 - Norme conçue par un consortium de constructeur
 - Outils pour construire des compilateurs poussés par des évènements
 - Compilation classique (pas de sérialiseur)
 - tirée par un analyseur lexical
 - XMLPull, CyberNeko, StAX
 - Génération automatiquement de compilateur et sérialiseur
 - JaxMe (JAXB) (à partir de dtd, schéma, Relax NG)
 - Zeus
 - Castor (à partir de dtd, schéma, classe Java)
 - Eclipse EMF (à partir de modèle de classes, schéma, UML)

DOM

DOM, une solution générique

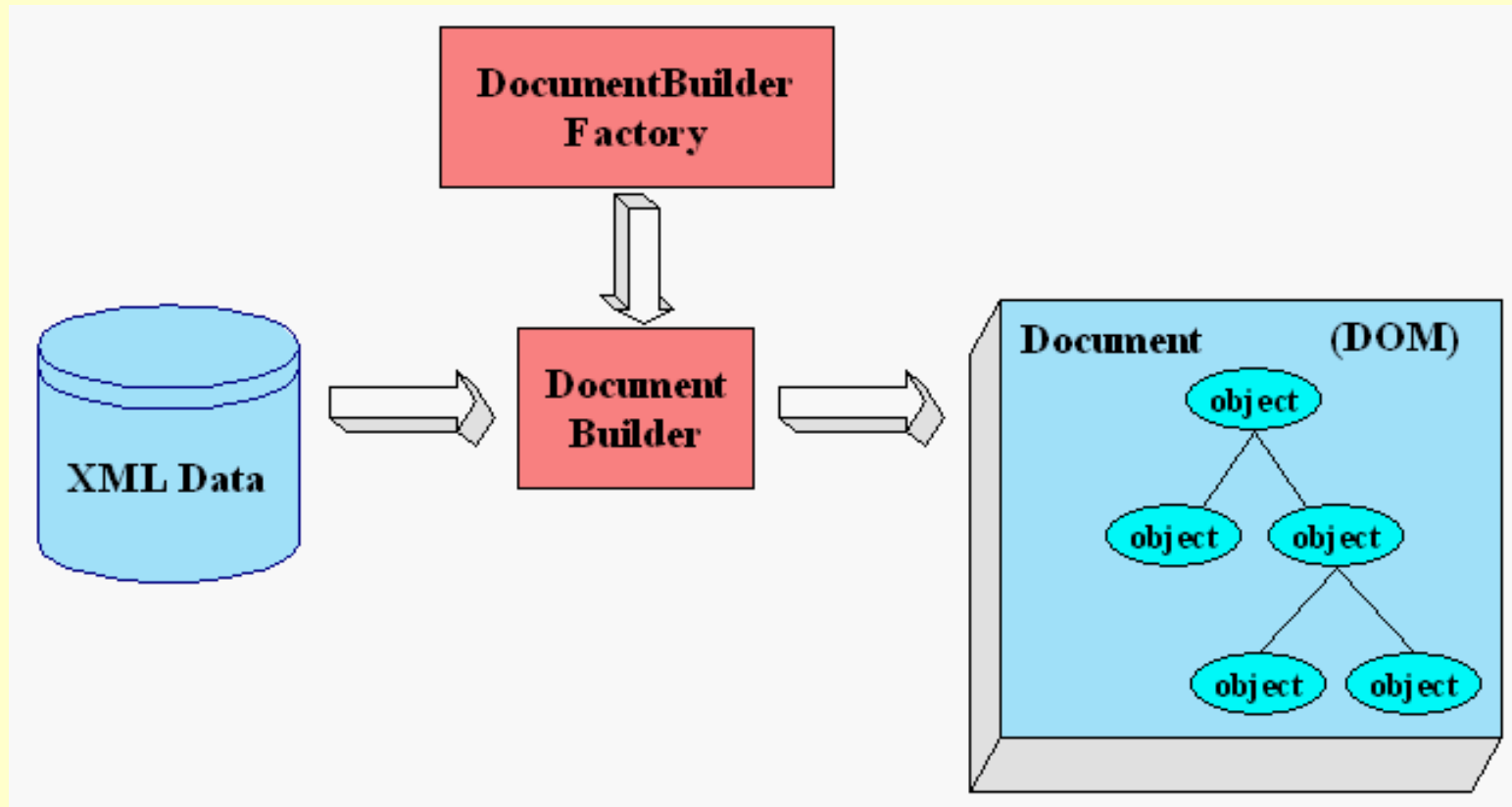
- Intérêt :
 - Compilation facile à mettre en œuvre (sans investissement)
 - Permettent le style *programmation extrême* ...
- Mais **pas forcément** la méthode la plus
 - *Rapide* pour arriver au résultat
 - *Extensible* (permettant de revenir sur la spécification)
 - *Efficace* (en temps et mémoire)
- Son défaut **trop souvent oublié**
 - *on construit un arbre générique*
 - *Parcours laborieux*
 - *Modifications laborieuses*



DOM

- Document Object Model
 - Modèle d'arbre générique permettant de représenter dans un programme un document html ou xml par des instances objets
 - Modèle indépendant des langages de programmation, s'applique à tout langage
 - C++, java, JavaScript, perl, Latex ...
- Pourquoi
 - parcourir l'arbre du document
 - ajouter, effacer, modifier un sous-arbre
 - sélectionner un sous-arbre par son contenu
- Par qui :
 - Conçu par le W3C
 - intégrée par Sun dans Java 1.4 sous le nom de Jaxp 1.0

Deux étages d'objets pour la compilation DOM



Compiler un document en un DOM

3 objets, 3 étapes

```
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import org.w3c.dom.Document;
```

```
DocumentBuilderFactory dbfactory =  
    DocumentBuilderFactory.newInstance();
```

```
DocumentBuilder domparser =  
    dbfactory.newDocumentBuilder();
```

```
Document doc = domparser.parse(  
    new File("data.xml"));
```

Exceptions lors de la construction du compilateur

```
try {  
    builder = factory.newDocumentBuilder();  
} catch (ParserConfigurationException e) {  
    e.printStackTrace();  
}
```

Des compilateurs DOM qui lancent des exceptions SAX

```
try {  
    doc = builder.parse(  
                                new File(entree));  
} catch (SAXException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

- Ils sont construits sur un compilateur SAX ...

Préférences de compilation

- `setCoalescing(b)` : Si le paramètre `b` est vrai les nœuds `CDATA` sont effacés. Valeur par défaut `false`.
- `setExpandEntityReferences(b)` : Si le paramètre `b` est vrai les entités références sont étendues. Valeur par défaut `true`.
- `setIgnoringComments(b)` : Si le paramètre `b` est vrai les commentaires sont ignorés. Valeur par défaut `false`.
- `setIgnoringElementContentWhitespace(b)` : Si le paramètre `b` est vrai les espaces non significatifs sont ignorés. *Il faut une DTD ou un schéma XML pour décider.* Valeur par défaut `false`.
- `setNamespaceAware(b)` : Détermine si le compilateur est averti des espaces de noms. Valeur par défaut `false`.
- `setValidating(b)` : Par défaut le compilateur ne fait pas de validation. Mettre à `true` pour effectuer la validation

Sérialiser un DOM en un document XML

- Sortir un document à l'identique en utilisant la méthode `transform(in, out)` de la classe `javax.xml.transform.Transformer`
- Programme Java

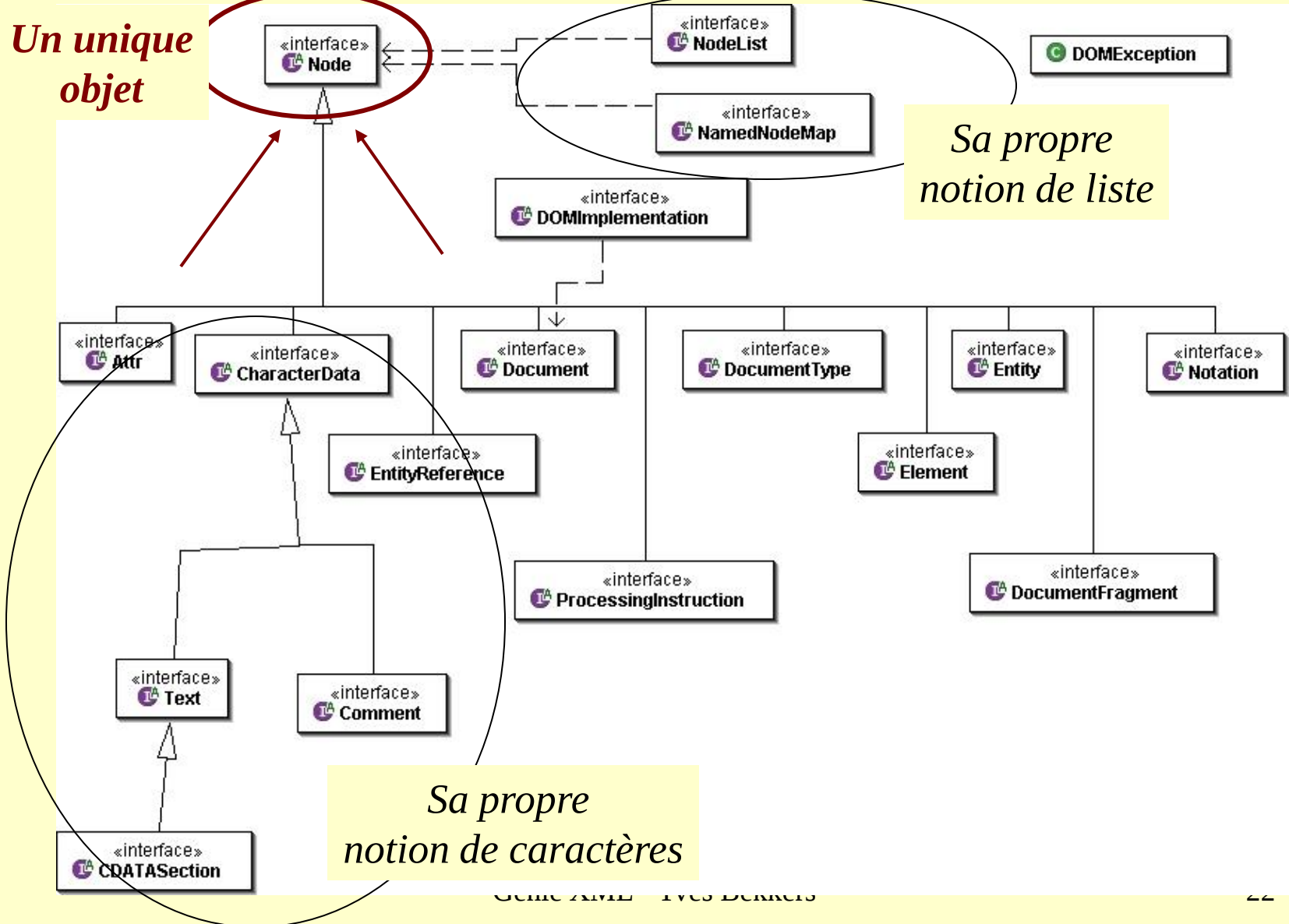
```
Document doc = ...;
Source source = new DOMSource(doc);
// préparation de la sortie
Result result = new StreamResult(
    new File(nomFichier));
// écriture du document DOM sur le fichier
Transformer trans =
    TransformerFactory.newInstance().newTransformer();
trans.transform(source, result);
```

Arbre DOM

« *Tout nœud est un objet **Node*** »

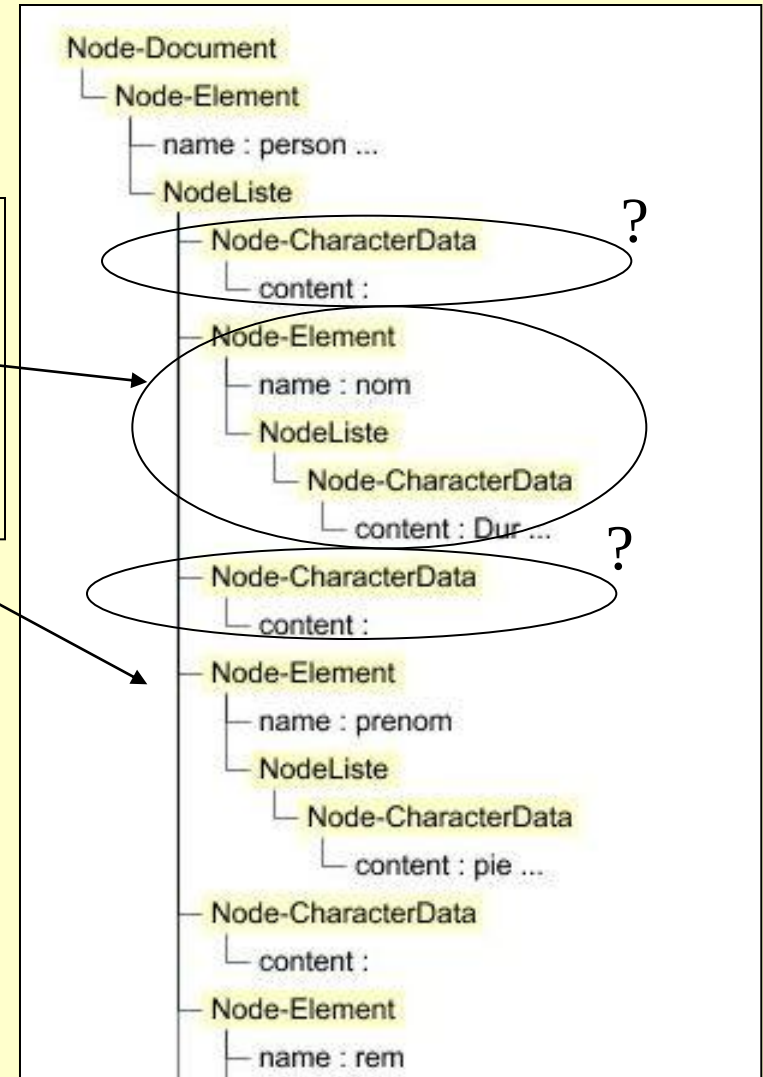
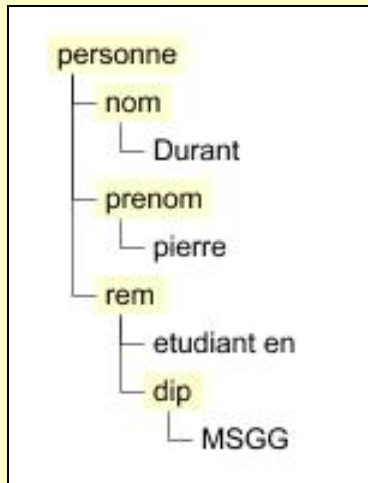
The diagram shows two UML interface boxes. The box on the left is labeled «interface» Node and is circled in red. The box on the right is labeled «interface» NodeList. A dashed dependency arrow points from the NodeList interface to the Node interface.

Un unique objet

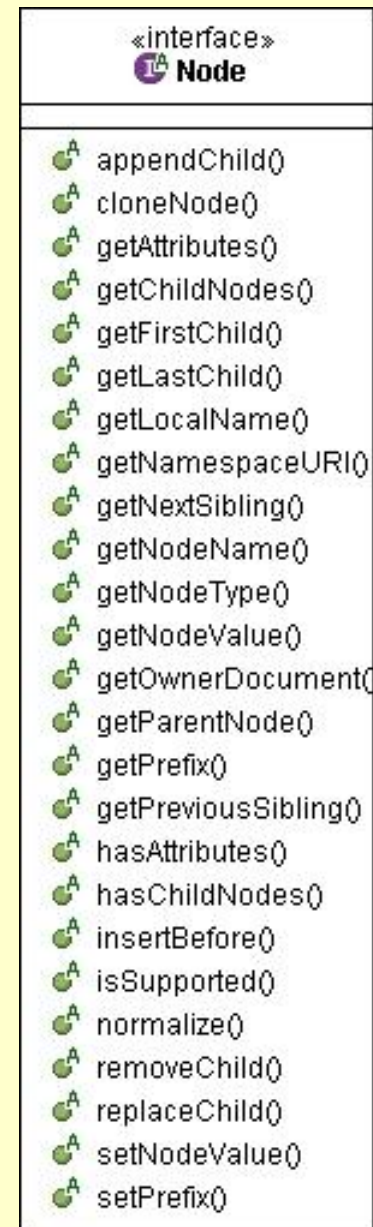
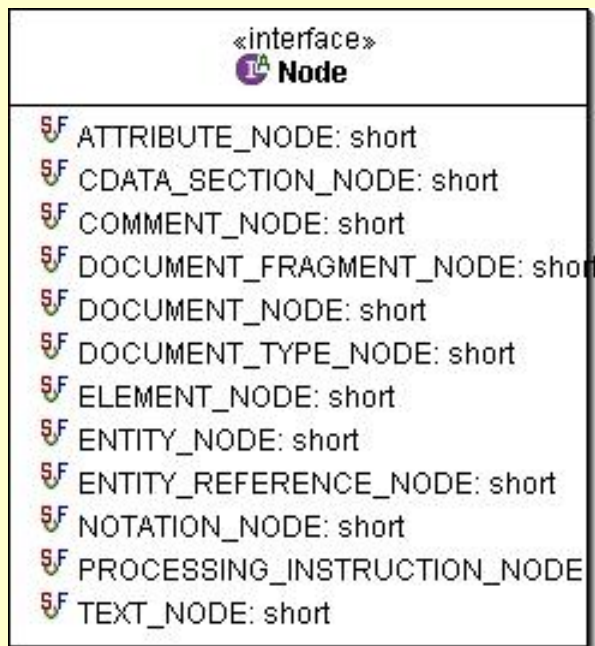


Exemple d'arbre DOM

```
<personne>
  <nom>Durant</nom>
  <prenom>pierre</prenom>
  <rem>etudiant en <dip>MSGG</dip></rem>
</personne>
```



Méthodes de l'interface Node



Méthodes `getNodeName()` `getNodeValue()`

Type de nœud	Role	<code>getNodeName()</code>	<code>getNodeValue()</code>
Attr	Attribut	nom	valeur
CDATASection	contenu de C.S.	"#cdata-section"	contenu
Comment	contenu de C.	"#comment"	contenu
DocumentType	nom de DTD	nom de la DTD	null
Document	document XML avec son prologue	"#document"	null
DocumentFragment	fragment de document XML	"#document-fragment"	null
Element	nom d'élément	nom de l'élément	null
EntityReference	nom d'entité	nom de l'entité	null
ProcessingInstruction	contenu de P.I.	nom du processeur	arguments
Text	contenu de nœud textuel	"#text"	contenu

Traverser un arbre DOM exemple

<adresse>

<ligne>IFSIC</ligne>

<ligne>Univ Rennes1</ligne>

<voie numéro="1">rue Le Bastard</voie>

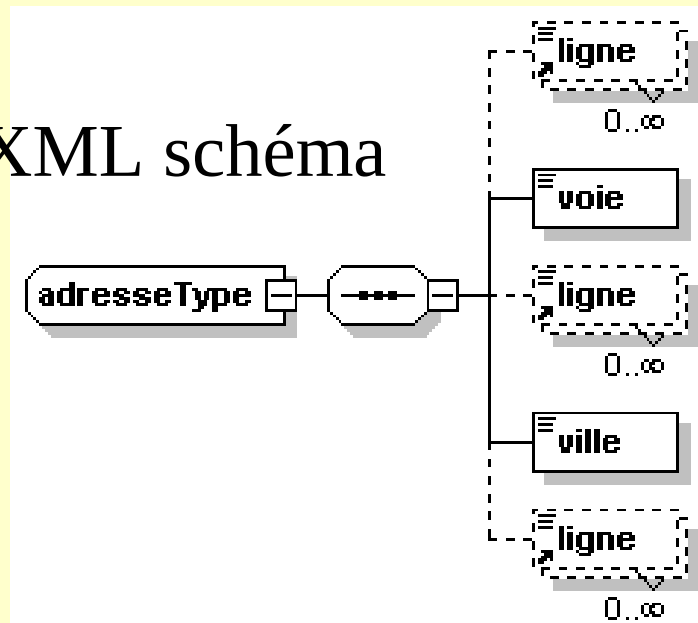
<ligne>CS 74205</ligne>

<ville codePostal="35042">Rennes</ville>

<ligne>CEDEX</ligne>

</adresse>

- Voici son type XML schéma



Descendre dans les fils

- Recherche des enfants de l'élément racine

```
Element racine = doc.getDocumentElement();  
NodeList enfants = racine.getChildNodes();
```

- Nombres de fils de l'élément racine

```
enfants.getLength()
```

- Résultat : 13 sans DTD, 6 avec DTD plus

```
setIgnoringElementContentWhitespace(true)
```

- Contenu textuel de l'élément `voie` 2 méthodes

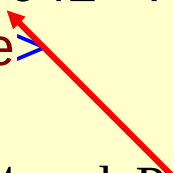
```
1) enfants.item(2).getFirstChild().getNodeValue()
```

```
2) racine.getElementsByTagName("voie").item(0).  
   getFirstChild().getNodeValue()
```

- Résultat : rue Le Bastard

Descendre dans un attributs

```
<adresse>
  <ligne>IFSIC</ligne>
  <ligne>Univ Rennes1</ligne>
  <voie numéro="1">rue Le Bastard</voie>
  <ligne>CS 74205</ligne>
  <ville codePostal="35042">Rennes</ville>
  <ligne>CEDEX</ligne>
</adresse>
```



- Valeur de l'attribut codePostal de l'élément ville

```
Element racine = doc.getDocumentElement();
NodeList enfants = racine.getChildNodes();
enfants.item(4).getAttributes().
    getNamedItem("codePostal").getNodeValue();
```

- Résultat

35042

Créer un document à partir de rien

- **Créer un document vide**

```
try {  
    DocumentBuilder builder =  
        DocumentBuilderFactory.newInstance().  
        newDocumentBuilder();  
    Document doc = builder.newDocument();  
}  
catch (ParserConfigurationException e) {  
    ...  
}
```

Créer un contenu de document

- Les nœuds `Element`, `Text`, `Comment`, etc. ne peuvent exister en dehors d'un document
- L'interface `Document` contient les méthodes de construction des nœuds
- Tout nœud `Node` possède un attribut `ownerDocument` qui l'associe au document pour lequel il a été créé.

Créer un contenu de document (suite)

```
Document doc = builder.newDocument();  
    // créer l'élément racine  
Element root = doc.createElement("racine");  
    // créer un noeud texte  
Text text = doc.createTextNode("ceci est le  
    contenu");  
    // mettre les noeuds dans l'arbre DOM du document  
root.appendChild(text);  
doc.appendChild(root);
```

Résultat

```
<racine>ceci est le contenu</racine>
```

Ajouter un élément à un contenu

// recherche de l'élément nom

```
Element racine = doc.getDocumentElement();  
Node nom = racine.getElementsByTagName("nom").item(0);
```

// création d'un élément prenom

```
Element prenom = doc.createElement("prenom");  
prenom.appendChild(doc.createTextNode("yves"));
```

// rattachement de l'élément prenom devant l'élément nom

```
racine.insertBefore(prenom, nom);
```

Résultat

```
<personne>  
  <nom>Durant</nom>  
  <age>28</age>  
  <taille>1.80</taille>  
</personne>
```



```
<personne>  
  <prenom>yves</prenom>  
  <nom>Durant</nom>  
  <age>28</age>  
  <taille>1.80</taille>  
</personne>
```

Autres opérations

- Traverser les nœuds d'un arbre DOM
 - L'interface `org.w3c.dom.Node` définit des méthodes pour traverser les nœuds et se déplacer n'importe où dans l'arbre
 - `getFirstChild()`, `getLastChild()`,
`getNextSibling()`, `getPreviousSibling()`,
`getParentNode()`...
- Créer des attributs
 - L'interface `org.w3c.dom.Element` qui étend l'interface `Node` définit un ensemble de méthodes pour manipuler les attributs d'un élément
 - `setAttribute()`, `getAttributeNode()`,
`hasAttribute()`...

Autres opérations (suite)

- Modifier un nœud d'arbre DOM
 - L'interface `org.w3c.dom.Node` fournit des méthodes pour enlever, remplacer, modifier un nœud
 - `removeChild()`, `replaceChild()`, `setNodeValue()`

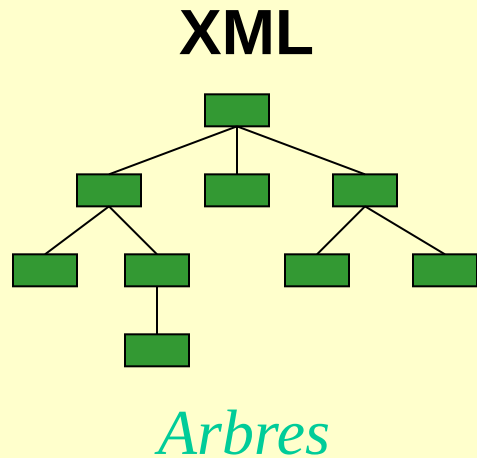
Dom, et les autres

- DOM est indépendant des langages
 - Contenus, chaînes, collections sont modélisés par des constructions spécifiques (n'utilisent pas directement les constructions du langage)
- JDOM, DOM4J, XOM adaptés à Java
 - Plus simple d'emploi
 - moins coûteux
 - Contenus, chaînes, collections sont directement des constructions standards du langage
- Construction

JDom, XOM sont basés sur la notion de *classes*
Dom, Dom4J sont basés sur la notion d'*interface*

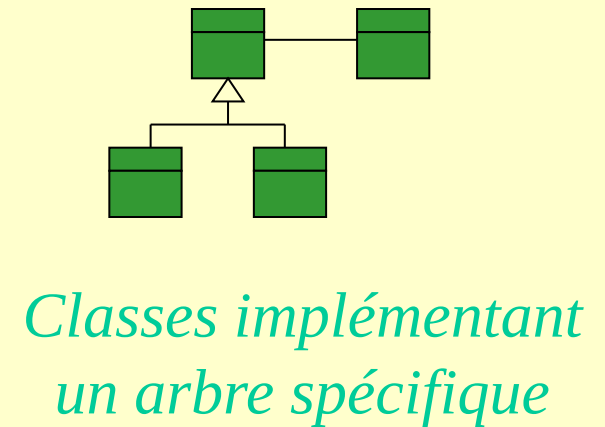
Compilation spécifique (adaptée à un dialecte) SAX, ...

Représentation externe/interne



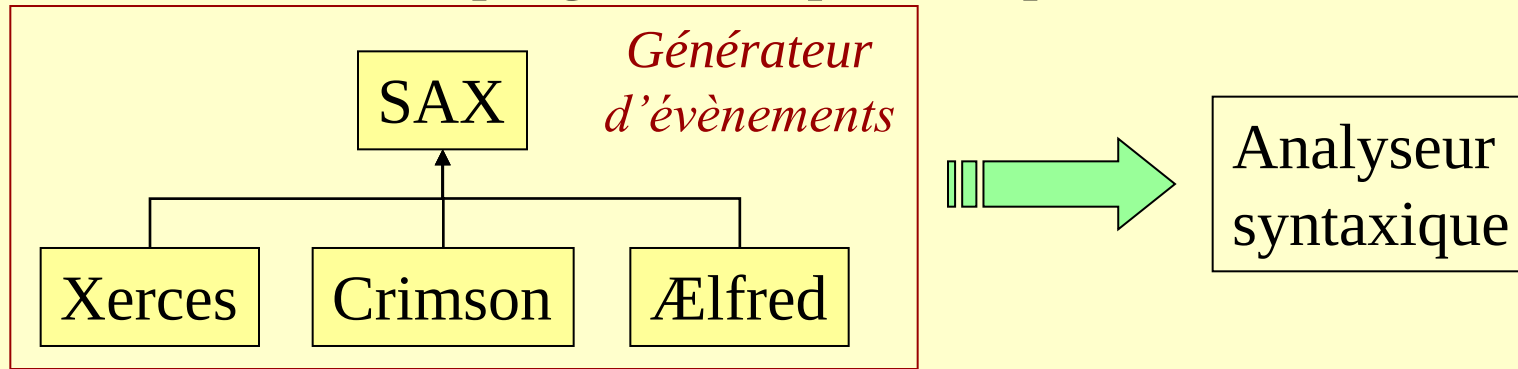
Compilateur
→
Sérialiser
←

Programmation Objet

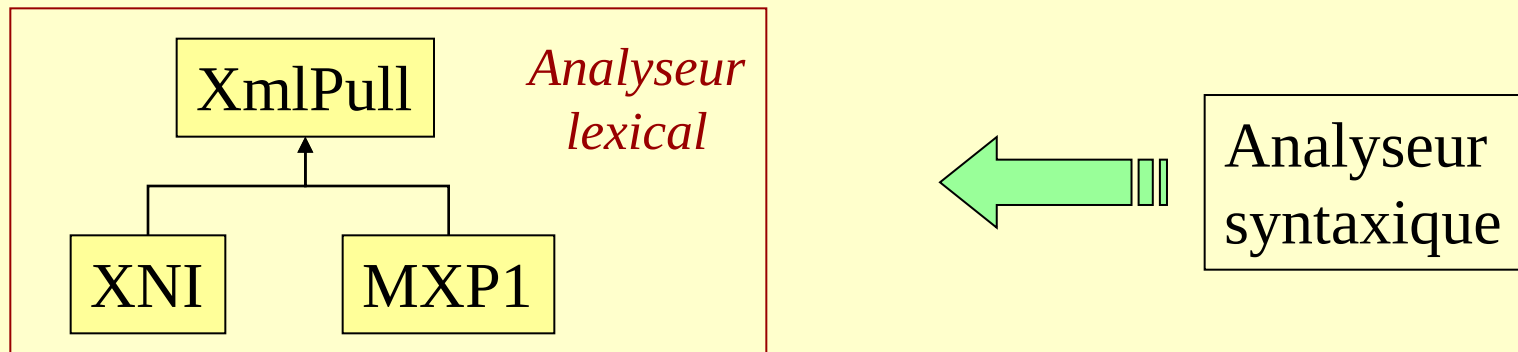


Deux formes de compilation manuelle

Librairies SAX : programmes poussés par des évènements



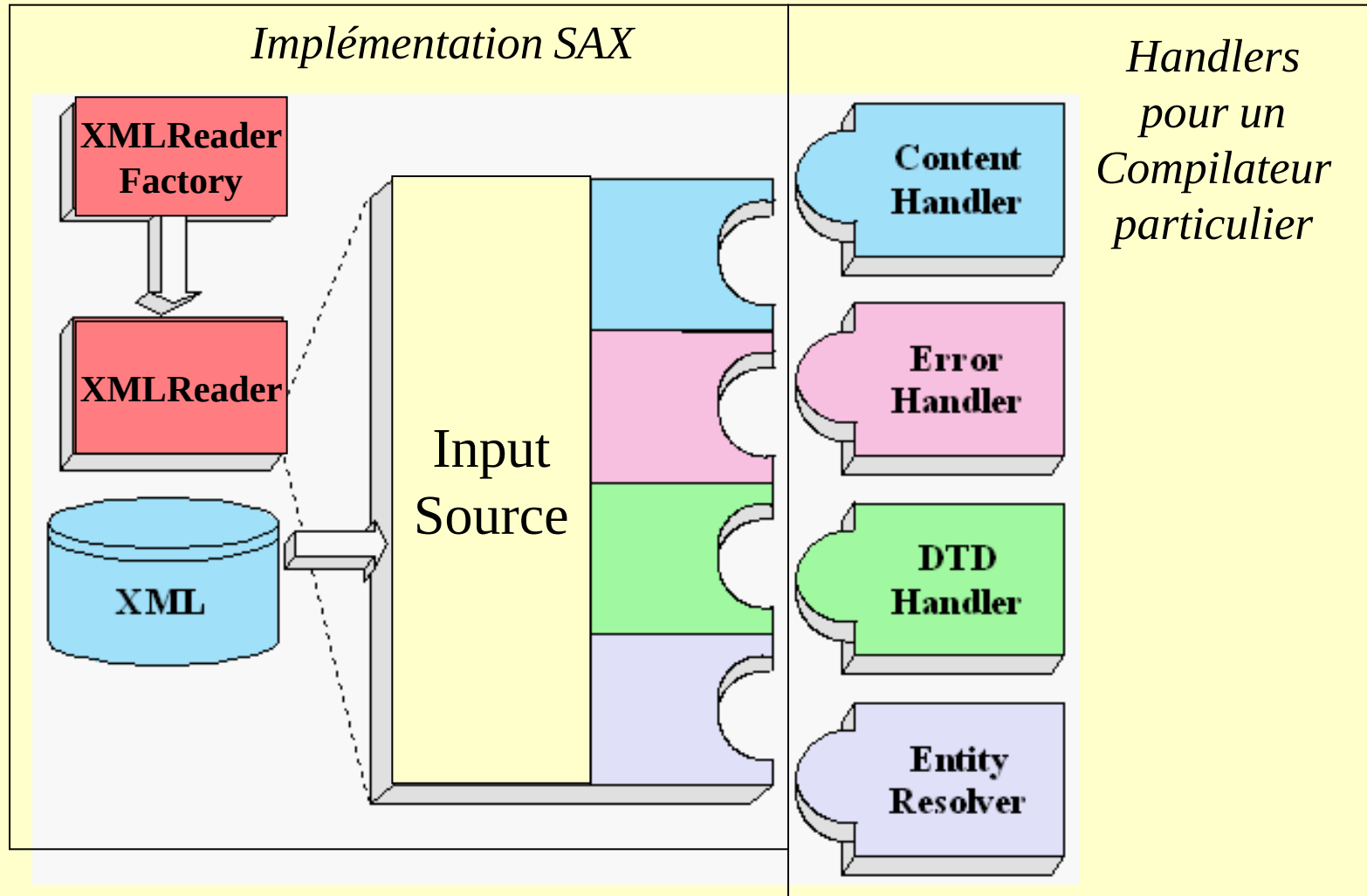
Librairies XmlPull : programmes tirant un analyseur lexical



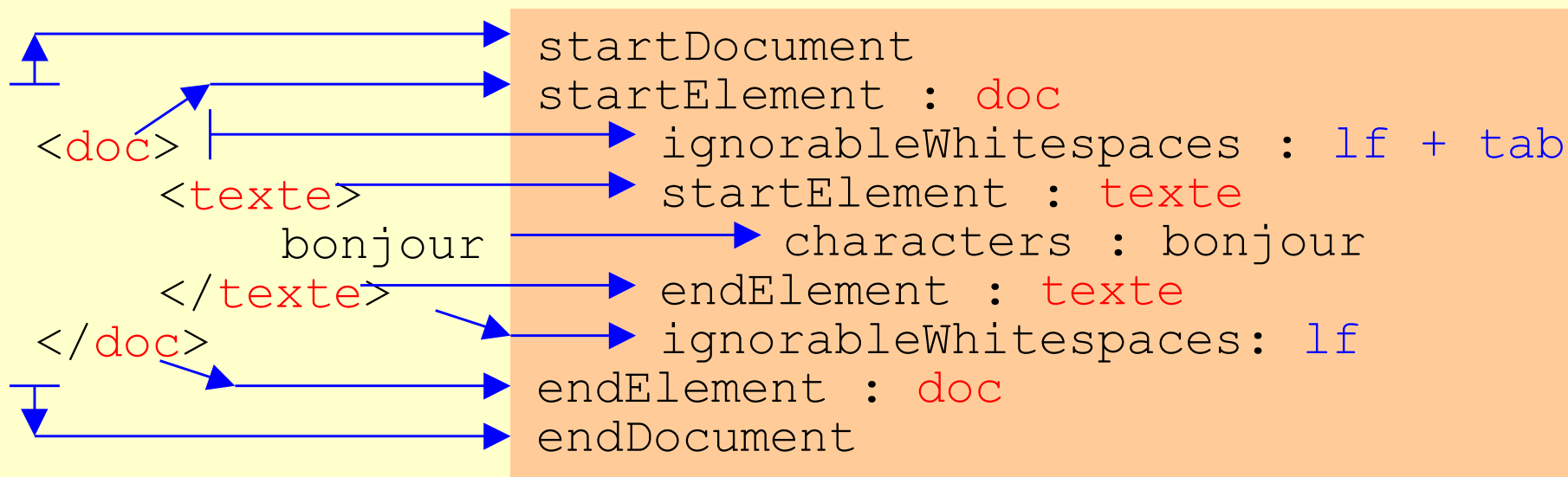
SAX - Compilation poussée par des évènements

- SAX : Simple API for XML
 - Modèle évènementiel
- Pourquoi
 - pour concevoir des compilateurs XML spécifiques
- Par qui
 - Un consortium de fabricants
 - Adopté par Sun dans Java 1.4 sous le nom Jaxp 1.0
- Comment
 - pas d'image mémoire du document lu
 - séquence d'événements à interpréter

Compilation SAX - principe



Séquence d'évènements



- Six types d'évènements
 - début et fin de document
 - début et fin d'éléments
 - caractères
 - caractères ignorables

Compiler un document avec SAX

les objets

```
import org.xml.sax.Attributes;  
import org.xml.sax.InputSource;  
import org.xml.sax.SAXException;  
import org.xml.sax.XMLReader;  
import org.xml.sax.helpers.DefaultHandler;  
import org.xml.sax.helpers.XMLReaderFactory;
```

Evolution des librairies SAX

- SAX 1 : ne plus utiliser l'interface

`org.xml.sax.Parser`

- Remplacée en SAX 2 par

`org.xml.sax.XMLReader`

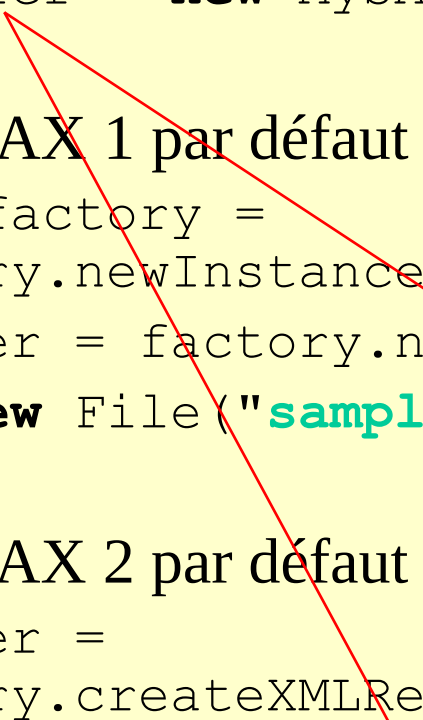
- Définit un standard pour valoriser et tester des propriétés
- Ajoute la prise en compte des espace de noms

Créer un compilateur SAX 1- SAX 2

```
MySAXHandler handler = new MySAXHandler();

// Utiliser le parseur SAX 1 par défaut (non validant)
SAXParserFactory factory =
    SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();
saxParser.parse(new File("sample"), handler);

// Utiliser le parseur SAX 2 par défaut (non validant)
XMLReader saxParser =
    XMLReaderFactory.createXMLReader();
saxParser.setContentHandler(handler);
saxParser.parse(new InputSource(new
    FileReader("sample")));
```



Flux d'entrée pour la compilation SAX

Trois sources possibles, *fonction*

`XMLReader.parse (InputSource)`

1. // À partir d'un flux d'octets avec détection, plus ou moins assistée, du codage d'entrée

`InputSource (InputStream byteStream)`

2. // À partir d'un flux de caractères Unicode

`InputSource (Reader characterStream)`

3. // À partir d'une identification unique Système URI

`InputSource (String systemId)`

Utiliser SAX en deux étapes

1) Spécifier un récepteur d'évènements

- Soit définir une classe qui étend la classe

`org.xml.sax.helpers.DefaultHandler`

Exemple `TestHandler extends DefaultHandler`

- Soit implémenter les interfaces

`DocumentHandler, DTDHandler,
EntityResolver, ErrorHandler`

2) Lancer la compilation

```
XMLReader xmlReader = XMLReaderFactory.createXMLReader();  
TestHandler handlerTest = new TestHandler();  
xmlReader.setContentHandler(handlerTest);  
FileReader reader = new FileReader("myFile");  
xmlReader.parse(new org.xml.sax.InputSource("myFile"));
```

Attention : base d'adresse des dtd

- Lorsqu'une DTD est présente, le nom du fichier DTD peut être relatif au répertoire du document XML, exemple :

```
<!DOCTYPE root SYSTEM "schemas/file.dtd">
```

- NE PAS UTILISER de `FileReader`

Adresse de DTD relative au répertoire de lancement du programme

```
FileReader lecteur = new FileReader("myFile");  
xmlReader.parse(new org.xml.sax.InputSource(lecteur));
```

- MAIS UTILISER DIRECTEMENT

Adresse de DTD relative au répertoire du document source

```
xmlReader.parse(new org.xml.sax.InputSource("myFile"));
```

Configurer un analyser

- Créer une fabrique de parseurs

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

- Configurer une fabrique de parseurs

```
spf.setNamespaceAware(boolean awareness)
```

Pour construire des analyseurs acceptant les espaces de noms

```
spf.setValidating(boolean validating)
```

Pour construire des analyseurs validant (DTD)

```
spf.setFeature(String name, boolean value)
```

Pour les autres propriétés spécifiques à un vendeur

Un exemple simple

- Tabuler un document

TestHandler exemple simple

/ Compter les éléments rencontrés**

Imprimer les contenus avec une présentation claire (indentée)

***/**

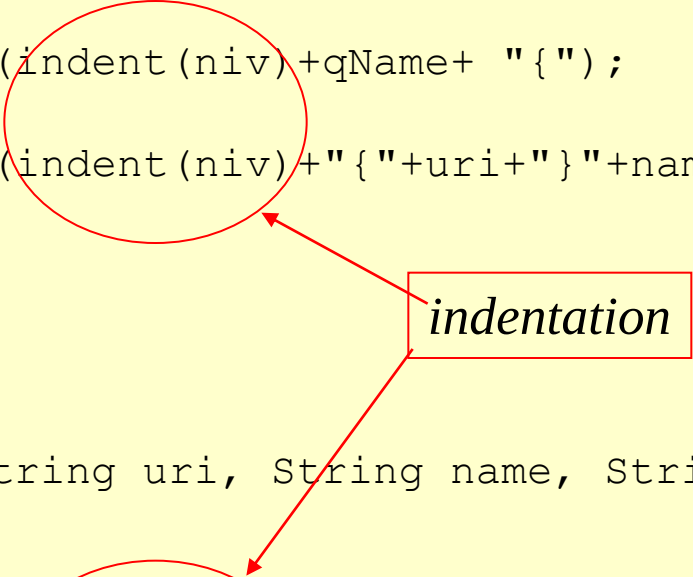
```
public class TestHandler extends DefaultHandler {  
    int nbElem = 0; // nombre d'éléments rencontrés  
    int niv = 0; // niveau de profondeur courant  
  
    public TestHandler() {  
        super();  
    }  
    ...  
}
```

Suite : Début et fin de document

```
public void startDocument() {  
    System.out.println("début de document");  
}  
  
public void endDocument() {  
    System.out.println("fin de document : " +  
        nbElem + " éléments rencontrés");  
}
```

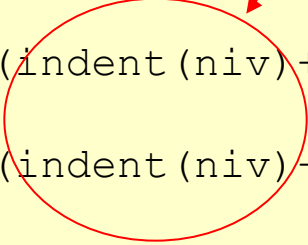
Suite : Début et fin d'éléments

```
public void startElement(String uri, String name,  
    String qName, Attributes atts) {  
    if("".equals(uri)) {  
        System.out.println(indent(niv)+qName+ "{");  
    } else {  
        System.out.println(indent(niv)+"{" +uri+"}" +name+ "{");  
    }  
    nbElem++;  
    niv++;  
}
```



The diagram illustrates the concept of indentation in the provided code. A red box labeled "indentation" has two red arrows pointing to the `indent(niv)` method calls in the `startElement` method. One arrow points to the call in the `if` branch, and the other points to the call in the `else` branch. Both calls are circled in red.

```
public void endElement(String uri, String name, String qName) {  
    niv--;  
    if("".equals(uri)) {  
        System.out.println(indent(niv)+"}" +qName);  
    } else {  
        System.out.println(indent(niv)+"}" + "{" +uri+"}" +name);  
    }  
}
```



The diagram illustrates the concept of indentation in the provided code. A red box labeled "indentation" has two red arrows pointing to the `indent(niv)` method calls in the `endElement` method. One arrow points to the call in the `if` branch, and the other points to the call in the `else` branch. Both calls are circled in red.

Suite : Gestion de l'indentation

```
public static String indent (int i) {  
    StringBuffer s = new StringBuffer("");  
    for ( int j=0; j<i; j++) {  
        s.append("    ");  
    }  
    return s.toString();  
}
```

Suite : Textes contenus

```
public void characters(char[] ch, int start, int lenght)
{
    System.out.print(indent(niv));
    for (int i = start; i<start+lenght; i++) {
        switch (ch[i]) {
            case '\\':
                System.out.print("\\\\");
                break;
            case '"':
                System.out.print("\\\"");
                break;
            ...
            default:
                System.out.print(ch[i]);
                break;
        }
    }
    System.out.println("");
}
```

Fin : Exemple d'exécution

début de document

```
personne{
```

```
  \n
```

```
  nom{
```

```
    Durant
```

```
  }nom
```

```
  \n
```

```
  age{
```

```
    28
```

```
  }age
```

```
  \n
```

```
  taille{
```

```
    1.80
```

```
  }taille
```

```
  \n
```

```
}personne
```

fin de document : 4 éléments rencontrés

<personne>

<nom>Durant</nom>

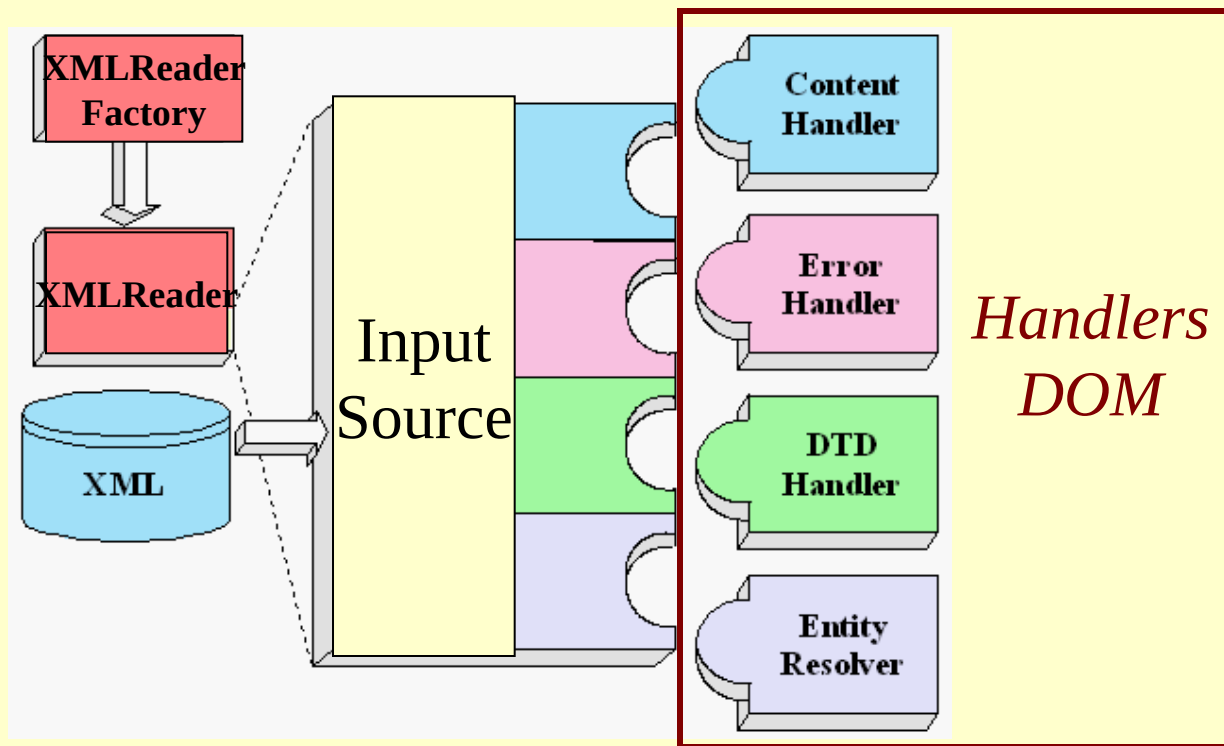
<age>28</age>

<taille>1.80</taille>

</personne>

DOM versus SAX

- Implémentation des compilateurs DOM
 - En Java tous les compilateurs DOM sont construits sur SAX

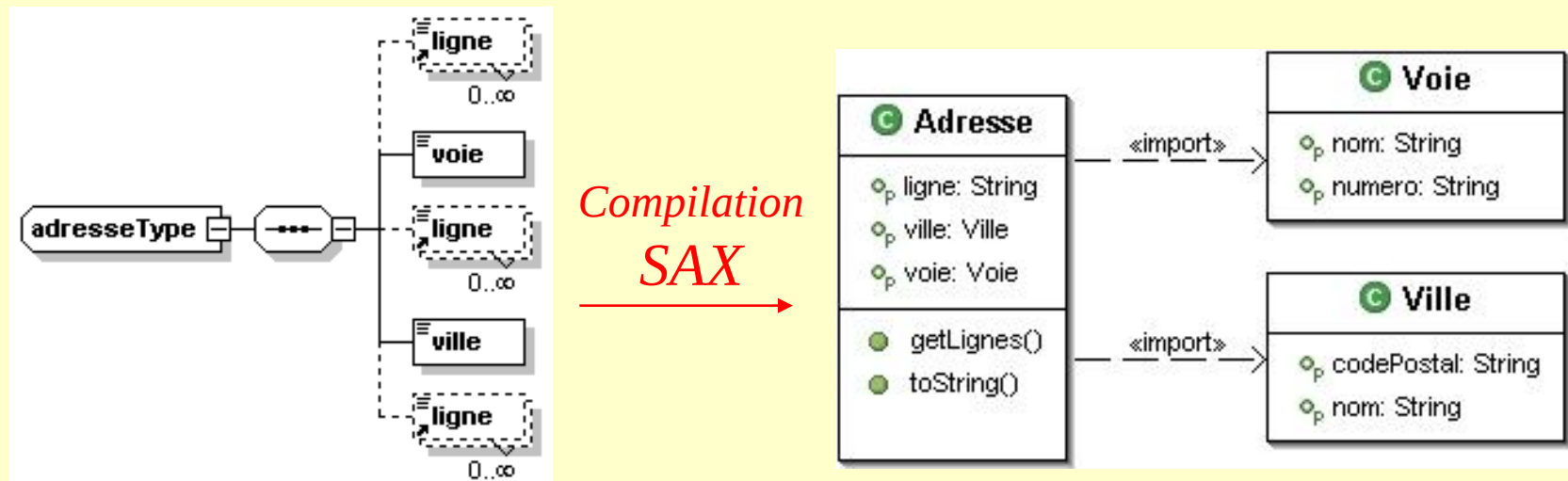


Second exemple

créer des classes propriétaires

```
<adresse>  
  <ligne>IFSIC</ligne>  
  <ligne>Univ Rennes1</ligne>  
  <voie numéro="1">rue Le Bastard</voie>  
  <ligne>CS 74205</ligne>  
  <ville codePostal="35042">Rennes</ville>  
  <ligne>CEDEX</ligne>  
</adresse>
```

Classes propriétaires



Choix de "projection"

- Éléments multiples à contenu textuel sans attribut
`<ligne>CEDEX</ligne>`
projection : `String[]` attribut d'objet
- Élément à contenu textuel avec attributs
`<ville codePostal="25042">Rennes</ville>`
projection : une classe avec 1 attribut par attribut d'élément plus un attribut pour le contenu
- Élément à contenu composé
`<adresse>...</adresse>`
projection : une classe avec 1 attribut par élément contenu

Adresse

```
public class Adresse {
    private Voie voie = new Voie();
    private Ville ville = new Ville();
    private String[][] lignes =
        {new String[0], new String[0], new String[0]};

    public String[] getLignes(int i) {
        assert (0 <= i && i < 3);
        return lignes[i];
    }

    public void setLignes(int i, String[] mesLignes){
        assert (0 <= i && i < 3);
        lignes[i] = mesLignes;
    }

    ...
}
```

AdresseHandler

```
public class AdresseHandler extends DefaultHandler {  
    // numéro du groupe courant de lignes  
    private int groupeLigne = 0;  
    // dernier texte lu  
    private StringBuffer textCourant;  
    // groupe de lignes en cours de construction  
    private List<String> listeStrings;  
    // adresse en construction  
    private Adresse adresse;  
  
    public AdresseHandler() {  
        super();  
    }  
    ...  
}
```

Sauver les lignes courantes

```
private void saveLines() {  
    adresse.setLignes(groupeLigne,  
        listeStrings.toArray(adresse.getLignes(groupeLigne)));  
    listeStrings = new ArrayList<String>();  
    groupeLigne++;  
}  
  
public void startDocument() {  
    adresse = new Adresse();  
    listeStrings = new ArrayList<String>();  
}  
  
public void endDocument() {  
    saveLines();  
    System.out.println(adresse.toString());  
}
```

Début d'éléments

```
public void startElement(String uri, String name,
    String qName, Attributes atts) {
    if (qName.equals("voie")) {
        adresse.getVoie().
            setNumero(atts.getValue("numéro"));
        saveLines();
    } else if (qName.equals("ville")) {
        adresse.getVille().
            setCodePostal(atts.getValue("codePostal"));
        saveLines();
    }
    textCourant = new StringBuffer("");
}
```

Fin d'éléments

```
public void endElement(String uri, String name,  
    String qName) {  
    if (qName.equals("voie")) {  
        adresse.getVoie().setNom(textCourant);  
    } else if (qName.equals("ville")) {  
        adresse.getVille().setNom(textCourant);  
    } else if (qName.equals("ligne")) {  
        listeStrings.add(textCourant + "\n");  
    }  
}
```

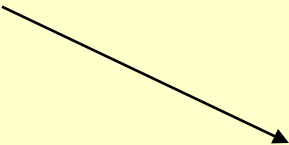
Textes

```
public void characters(char[] ch, int start,  
    int lenght) {  
    StringBuffer s = new StringBuffer("");  
    for (int i = start; i < start + lenght; i++) {  
        s.append(ch[i]);  
    }  
    textCourant = textCourant.append(s.toString());  
}
```

Lancement de la compilation

```
XMLReader xr =  
    XMLReaderFactory.createXMLReader();  
MyHandler handler = new MyHandler();  
xr.setContentHandler(handler);  
FileReader r = new FileReader("file.xml");  
xr.parse(new InputSource(r));
```

```
<adresse>
  <ligne>IFSIC</ligne>
  <ligne>Univ Rennes1</ligne>
  <voie numéro="1">rue Le Bastard</voie>
  <ligne>CS 74205</ligne>
  <ville codePostal="35042">Rennes</ville>
  <ligne>CEDEX</ligne>
</adresse>
```



Exemple d'exécution

IFSIC
Univ Rennes1
1 rue Le Bastard
CS 74205
Rennes 35042
CEDEX

Avantages et inconvénients de SAX

- Avantage
 - Construction d'un arbre spécifique au document en un seul parcours
- Inconvénient
 - Ce n'est pas un compilateur
 - Il faut programmer un récepteur d'évènements

Implémentations externes de compilateur SAX

- Xerces d'Apache
<http://xml.apache.org/xerces-j/index.html>
- Crimson d'Apache
<http://xml.apache.org/crimson/index.html>
- Ælfred utilisé par Saxon de Michael Kay
<http://saxon.sourceforge.net/>
- XDK d'Oracle
http://technet.oracle.com/tech/xml/xdk_java/content.html

Validation

Validation par une DTD

Compilation DOM

- Document XML : déclarer une DTD

```
<!DOCTYPE root SYSTEM "file.dtd">
```

- Programme Java : analyseur DOM

```
final String source = "monFichier.xml";
```

```
DocumentBuilderFactory factory =
```

```
    DocumentBuilderFactory.newInstance();
```

```
factory.setValidating(true);
```

```
DocumentBuilder parser = factory.newDocumentBuilder();
```

```
Document document = parser.parse(new File(source));
```

Validation par un schéma xml

Compilation DOM

```
final String source = "carnetDAdresse.xml";  
final String schema = "carnetDAdresse.xsd";  
SchemaFactory schemaFactory =  
    SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);  
Schema schemaXSD = schemaFactory.newSchema(new File(schema));  
// créer un compilateur  
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
// associer le schéma xml au compilateur  
factory.setSchema(schemaXSD);  
factory.setNamespaceAware(true);  
DocumentBuilder parser = factory.newDocumentBuilder();  
Document document = parser.parse(new File(source));
```

Validation par une DTD

Compilation SAX 2

- Document XML : déclarer une DTD

```
<!DOCTYPE myRoot SYSTEM "schemas/file.dtd">
```

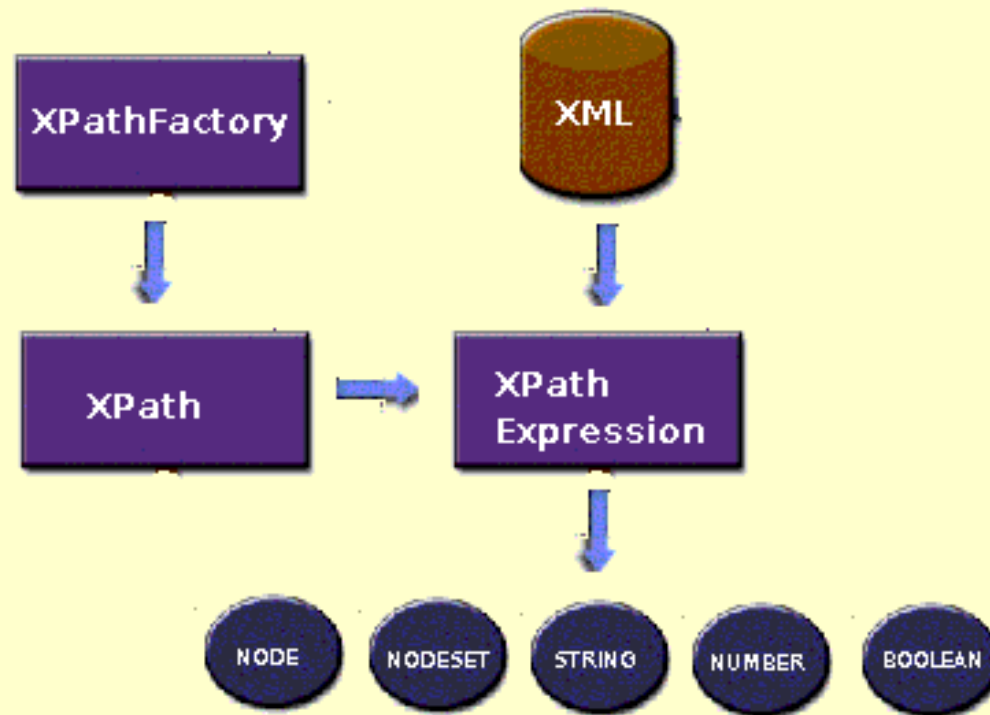
- Programme Java : compilation SAX 2

```
final String source = "monFichier.xml";  
XMLReader xmlReader =  
    XMLReaderFactory.createXMLReader();  
xmlReader.setFeature(  
    "http://xml.org/sax/features/validation", true);  
xmlReader.setContentHandler(new DefaultHandler());  
xmlReader.parse(new org.xml.sax.InputSource(source));
```

XPath

Se déplacer rapidement dans un document XML

XPath



Evaluer une expression XPath sur un document DOM

//Compiler le document XML

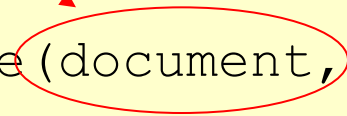
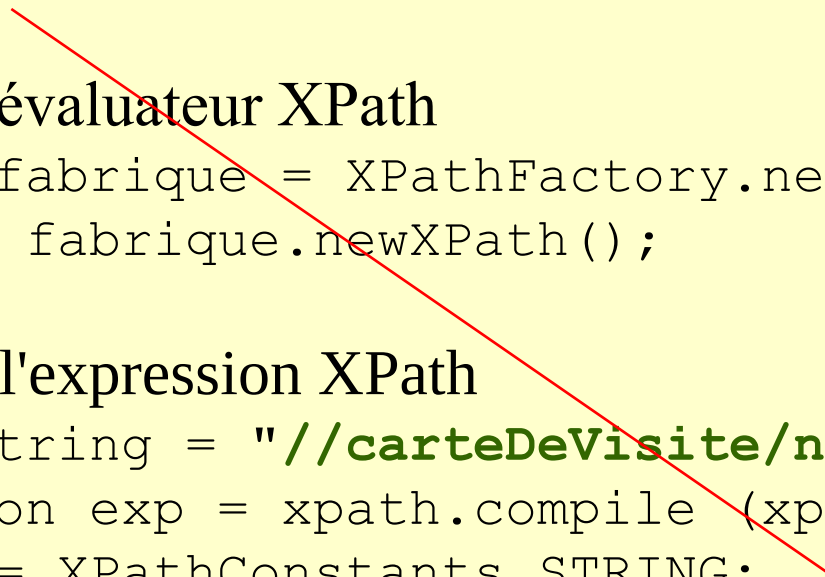
```
DocumentBuilder builder = DocumentBuilderFactory.  
    newInstance().newDocumentBuilder();  
Document document = builder.parse(new File("file.xml"));
```

//création d'un évaluateur XPath

```
XPathFactory fabrique = XPathFactory.newInstance();  
XPath xpath = fabrique.newXPath();
```

//évaluation de l'expression XPath

```
String xpathString = "//carteDeVisite/nom";  
XPathExpression exp = xpath.compile(xpathString);  
QName retour = XPathConstants.STRING;  
String resultat = (String)exp.evaluate(document, retour);
```



Evaluer une expression XPath sur un flux SAX

//création du flux SAX

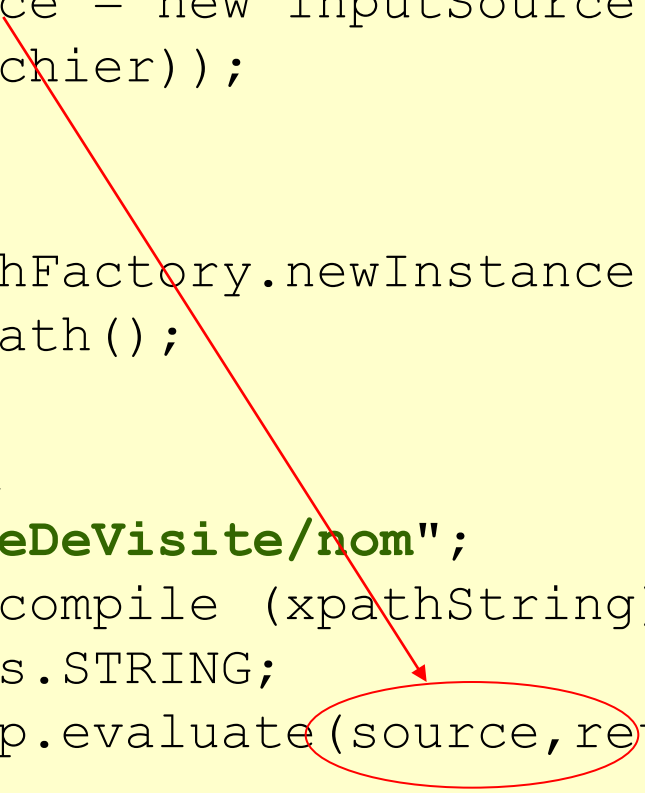
```
org.xml.sax.InputSource source = new InputSource(  
    new FileInputStream(fichier));
```

//création d'un évaluateur XPath

```
XPathFactory fabrique = XPathFactory.newInstance();  
XPath xpath = fabrique.newXPath();
```

//évaluation de l'expression XPath

```
String xpathString = "//carteDeVisite/nom";  
XPathExpression exp = xpath.compile(xpathString);  
QName retour = XPathConstants.STRING;  
String resultat = (String)exp.evaluate(source, retour);
```



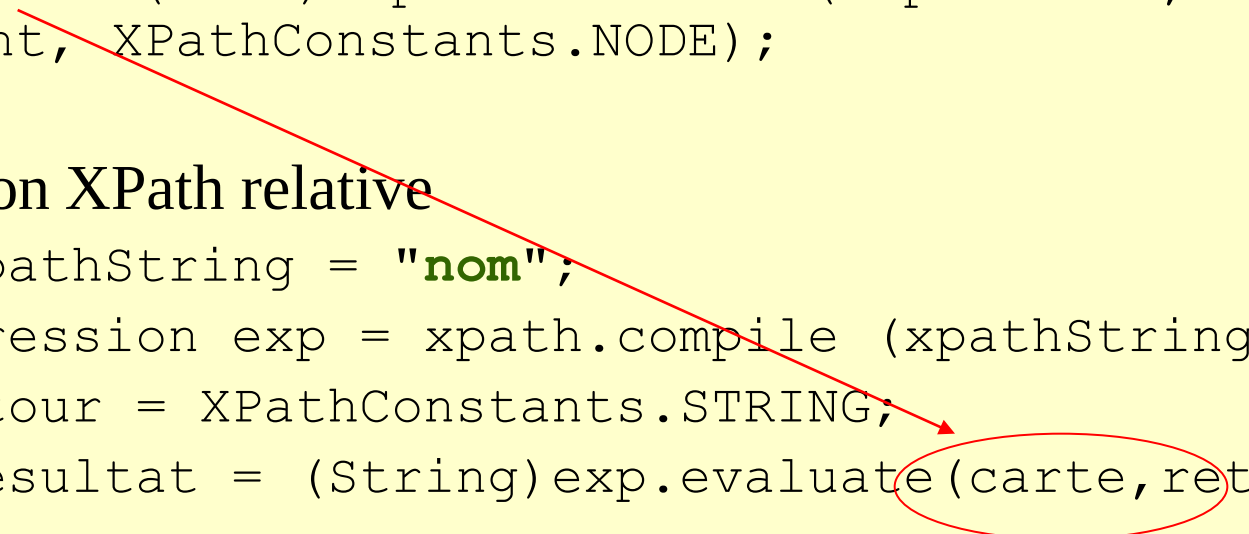
Expression XPath relative à un nœud

//recherche d'un nœud par une Première évaluation XPath

```
XPath xpath = XPathFactory.newInstance().newXPath();  
String expression = "//carteDeVisite";  
Node carte = (Node) xpath.evaluate(expression,  
    document, XPathConstants.NODE);
```

//évaluation XPath relative

```
String xpathString = "nom";  
XPathExpression exp = xpath.compile(xpathString);  
QName retour = XPathConstants.STRING;  
String resultat = (String)exp.evaluate(carte, retour);
```



Récupérer un ensemble de noeuds

```
XPath xpath =  
    XPathFactory.newInstance().newXPath();  
String expression =  
    "/carnetDAdresse/carteDeVisite";  
  
QName retour = XPathConstants.NODESET;  
  
NodeList carte = (NodeList)  
    expr.evaluate(document, retour);
```

Déclarer une variable dans le contexte d'évaluation

```
final QName f_var =  
    new QName("http://localhost/f", "pi");  
xpath.setXPathVariableResolver(  
    new XPathVariableResolver() {  
        public Object resolveVariable(QName qName) {  
            if (qName.equals(f_var)) {  
                return new Double(Math.PI);  
            } else {  
                return null;  
            }  
        }  
    }  
);
```

Déclarer un espace de noms

```
xpath.setNamespaceContext(new NamespaceContext() {  
    public String getNamespaceURI(String s) {  
        if (s.equals("f")) {  
            return "http://localhost/f";  
        } else {  
            return null;  
        }  
    }  
}  
  
    public String getPrefix(String s) {  
        return null;  
    }  
  
    public Iterator getPrefixes(String s) {  
        return null;  
    }  
});
```

Utiliser la variable

- Référence à une variable définie par programme dans une expression xpath

```
$f:pi
```

Déclarer une fonction dans le contexte d'évaluation (1)

```
final XPathFunction sqrt = new XPathFunction() {  
    public Object evaluate(List list) throws  
        XPathFunctionException {  
        Object arg = list.get(0);  
        if (!(arg instanceof Double)) {  
            throw new XPathFunctionException(  
                "f:sqrt() expects an xs:double argument");  
        }  
        return new Double(Math.sqrt((  
            (Double) arg).doubleValue()));  
    }  
};
```

Déclarer une fonction dans le contexte d'évaluation (2)

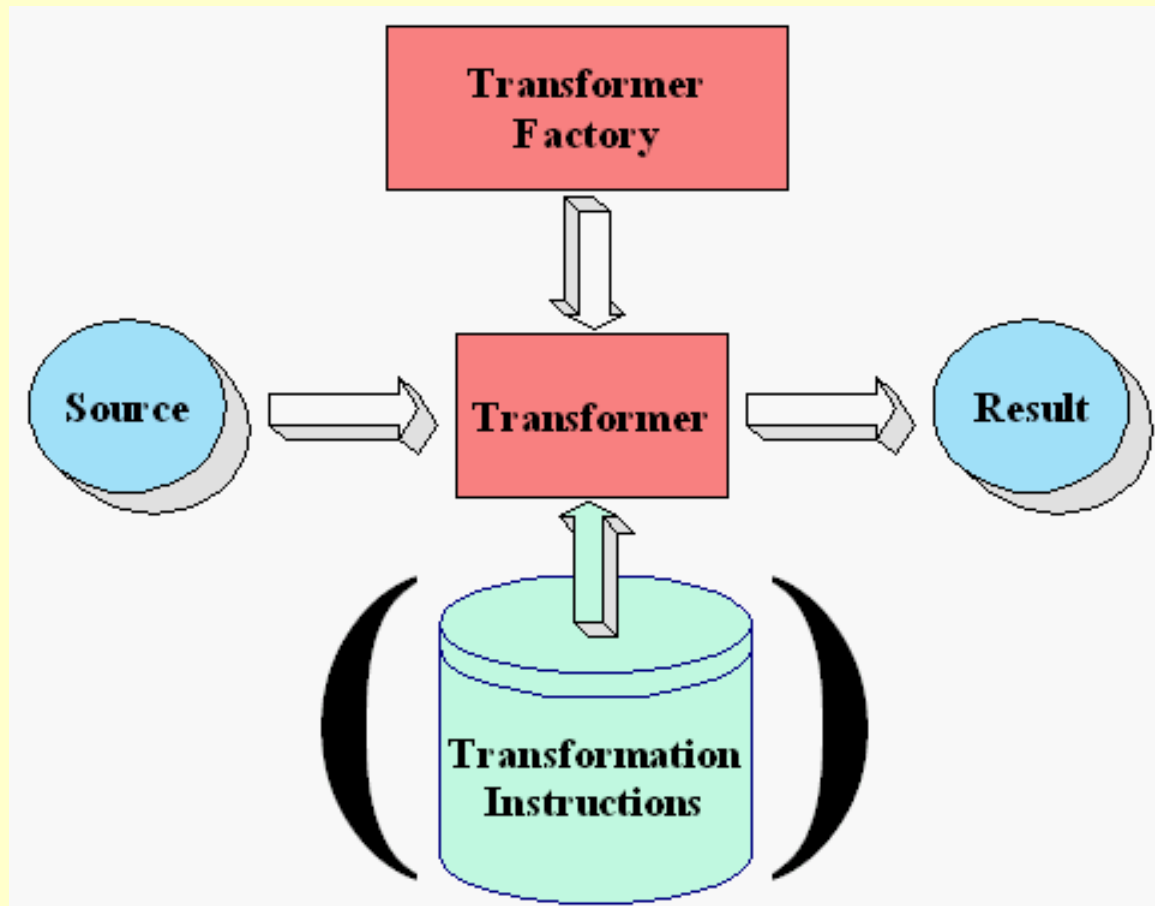
```
final QName f_sqrt =  
    new QName("http://localhost/f", "sqrt");  
xpath.setXPathFunctionResolver(  
    new XPathFunctionResolver() {  
        public XPathFunction resolveFunction(  
            QName qName, int arity) {  
            if (qName.equals(f_sqrt) && arity == 1) {  
                return sqrt;  
            } else {  
                return null;  
            }  
        }  
    }  
);
```

APIs Java pour Xpath

- Jaxent
 - Une machine universelle d'évaluation Xpath
 - Compatible JDOM et DOM4J

XSLT

Principe d'une transformation



TRaX

- Processeurs XSLT interchangeables
 - TRaX est une « façade » qui permet d'effectuer des transformations XSLT indépendamment du processeur utilisé
 - Les programmes utilisant TRaX sont portables sur n'importe quel processeur compatible TRaX
- TRaX a été intégré par Sun dans Java 1.4 sous le nom JAXP 1.1

TRaX des outils complémentaires

- 3 interfaces
 - `Source` : Le document en entrée
 - `Result` : Le document en sortie
 - `Templates` : Une vue interne d'une feuille de style
 - « Thread safe »
- 2 classes abstraites
 - `TransformerFactory` : Fabrique de transformateurs
 - `Transformer` (associé à un `Thread`)
 - Un objet qui applique une transformation à un document source et produit un résultat
 - Le résultat reste en interne, la sérialisation est celle de la machine de transformation, son comportement est paramétrable

Lancer une transformation en Java

- Procédez comme suit :
 - Créez une source ‘document d’entrée’
 - Créez une source ‘feuille de style’
 - Créez un objet résultat
 - Créez un instance de transformateur
 - Transformer

TRaX - créer une source xml

```
// create the XML content input source:  
// can be a DOM node, SAX stream, or any  
// Java input stream/reader  
String xmlInputFile = "myXMLinput.xml";  
Source xmlSource = new StreamSource(new  
FileInputStream(xmlInputFile));
```

TRaX - créer la source xslt

```
// create the XSLT Stylesheet input source
// can be a DOM node, SAX stream, or a
// java input stream/reader
String xsltInputFile =
"myXsltStylesheet.xsl";
Source xsltSource = new StreamSource(new
FileInputStream(xsltInputFile));
```

TRaX créer un objet résultat

```
// create the result target of the
// transformation
// can be a DOM node, SAX stream, or
// a java out
// stream/reader
String xmlOutputFile = "result.html";
Result transResult = new
StreamResult(new
FileOutputStream(xmlOutputFile));
```

TRaX - effectuer la transformation

```
// create the transformerfactory &  
transformer instance  
TransformerFactory tf =  
TransformerFactory.newInstance();  
Transformer t =  
tf.newTransformer(xsltSource);  
  
// execute transformation & fill  
result target object  
t.transform(xmlSource, transResult);
```

Mémoire cache des feuilles de styles

- Eviter de compiler et valider répétitivement une même feuille de style.
 - Créer une mémoire cache de feuilles précompilées.
 - Utiliser la classe

`javax.xml.transform.Templates`

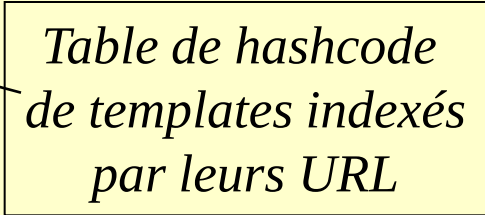
- **Code**

```
Templates pss = tryCache(styleUrl);  
Transformer trans = pss.newTransformer();
```

Gestion du cache

```
private synchronized Templates tryCache(String url)
    throws TransformerException, java.io.IOException {
    String path = getServletContext().getRealPath(url);
    if (path==null) {
        throw new TransformerException("Stylesheet " +
            url + " not found");
    }

    Templates template = (Templates)cache.get(path);
    if (template ==null) {
        TransformerFactory factory =
            TransformerFactory.newInstance();
        template = factory.newTemplates(new StreamSource(
            new File(path)));
        cache.put(path, template);
    }
    return x;
}
```



*Table de hashcode
de templates indexés
par leurs URL*

Transformer un document à l'aide d'un programme XSLT

```
Source source = new DOMSource(doc);
Source xslSource = new StreamSource(
    new File(fichierXSL));
Result result = new StreamResult(
    new File(nomFichier));
Transformer transformer = null;
try {
    transformer = TransformerFactory.newInstance().
        newTransformer(xslSource);
    transformer.transform(source, result);
} catch (TransformerConfigurationException e) {...}
```

Deux types de transformation

- Intanciation de la fabrique de transformer

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

- Transformation à l'identique

```
Transformer transformer =  
    factory.newTransformer();
```

- Transformation selon une feuille de style XSLT

```
Transformer transformer =  
    factory.newTransformer(  
        new StreamSource(  
            new File(fichierXSL)) );
```

Créer un transformateur XSLT

- Création directe

```
Transformer transformer =  
    factory.newTransformer(  
        new StreamSource(  
            new File(fichierXSL)) );
```

- Création d'une version compilée

```
Templates proc = factory.newTemplates(  
    new StreamSource(  
        new File(fichierXSL)) );  
Transformer transformer =  
    proc.newTransformer();
```

Xinclude - 1

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude">  
  <xi:include href="somewhere.xml"/>  
</foo>
```

Où le document `somewhere.xml` contient :

```
<bar>...</bar>
```

Est équivalent à :

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude">  
  <bar>...</bar>  
</foo>
```

Xinclude

- <http://www.w3.org/2001/XInclude> est le nom officiel de l'espace **XInclude**
- L'élément `xi:include` est une directive d'inclusion
- Il y a d'abord une étape de compilation puis juste après il y a une étape de remplacement (d'arbre)
- Les deux document doivent être considérés comme équivalents
- Il ne peut y avoir de boucle récursives d'inclusion
- Java 1.5 classe

`javax.xml.parsers.DocumentBuilderFactory`

Méthode

void `setXIncludeAware(boolean state)`

Intérêt du DOM

- Intérêt du DOM
 - Pas d'investissement
 - Interopérabilité : Norme acceptée universellement
 - Échange d'arbres XML entre applications
 - Compilateur indépendant du dialecte XML
 - Des mise en œuvres pour tous les langages
 - java, C++, Perl, LaTeX, ...
- Exemple de succès du DOM
 - html dynamique et javascript
 - Compilateurs DOM sur l'étagère
 - Transformateur XSLT produisant en sortie un arbre DOM

Inconvénient du DOM

- Spécification indépendante du langage
 - on ne normalise que des interfaces
 - les implémentations sont laissées en dehors de la norme
 - lourdeur dans l'utilisation (pas de `new` en direct, utilisation de « fabriques » d'objets)
- Spécification permettant de représenter tout arbre xml en général
 - taille des objets en mémoire ...
 - généricité pas adaptée au gros documents ou aux documents complexes

DOM références

- Document de définition du DOM niveau 1 (W3C)
 - <http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/level-one-core.html>
- Information sur la partie spécifique html
 - <http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/level-one-html.html>
- Un tutorial sur l'utilisation du DOM en Javascript
 - <http://www.pageresource.com/dhtml/ryan/part4-1.html>
- Un autre sur l'utilisation du DOM en VisualBasic
 - <http://www.thescarms.com/XML/DOMTutorial.asp#DOMDocument>