

# FORMATION XML



# CONCEPTS DE BASE XML



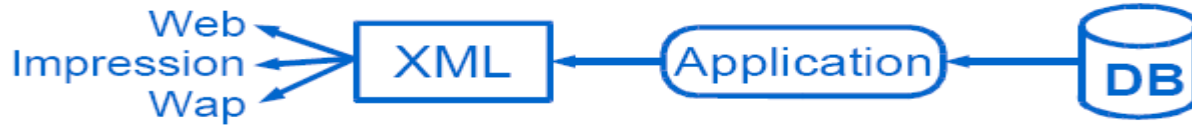
# QU'EST-CE QUE XML ?

- XML est une norme de structuration des données
- la structuration peut servir au stockage ou à la communication
- XML permet de définir de nouveaux types de documents
  - Wml
  - XHTML
  - RDF
  - MathML
  - SVG
  - ...
- ~~○ à terme, on voudrait que toutes les informations stockées ou échangées soient structurées en XML.~~



# CONTEXTES D' UTILISATION DE XML ?

- Découplage données/présentation (archi 4 tiers)



- Format de stockage
  - Applications spécifiques (ex:SVG)
  - Fichiers de configuration(ex:Tomcat, EJB/JEE)
- Format d' échange
  - Evolution de l' EDI, application B-to-B
  - Service web(SOAP, UDDI,...)



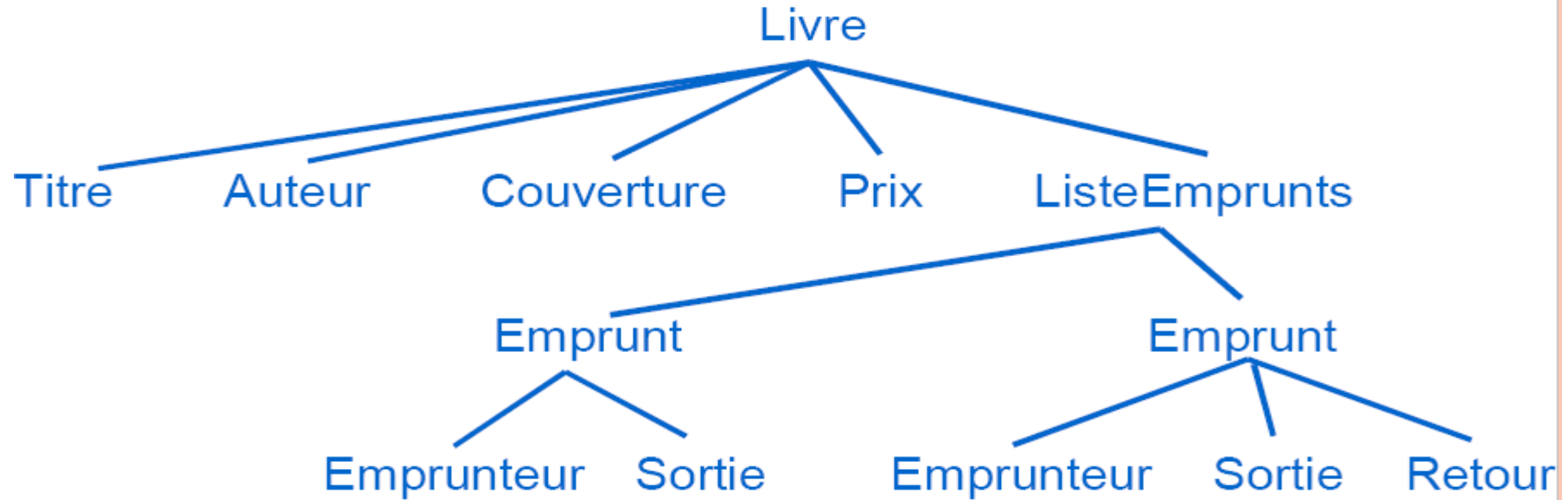
# EXEMPLE DE DOCUMENT XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet href="bib.css" ?>
<?cocoon-process type="xslt" ?>
<!DOCTYPE Livre SYSTEM "Livre.dtd">

<Livre>
  <Titre>Les réseaux</Titre>
  <Auteur>A. Tanenbaum</Auteur>
  <Couverture imgsrc="/imgs/res-tan.jpg"/>
  <Prix devise="EUR">25.42</Prix>
  <ListeEmprunts>
    <Emprunt>
      <Emprunteur>François Duchemin</Emprunteur>
      <Sortie>25/09/2000</Sortie>
      <Retour>02/10/2000</Retour>
    </Emprunt>
    <Emprunt>
      <Emprunteur>Hervé Delarue</Emprunteur>
      <Sortie>05/10/2000</Sortie>
    </Emprunt>
  </ListeEmprunts>
</Livre>
```



# STRUCTURE ARBORESCENTE D' UN DOCUMENT XML

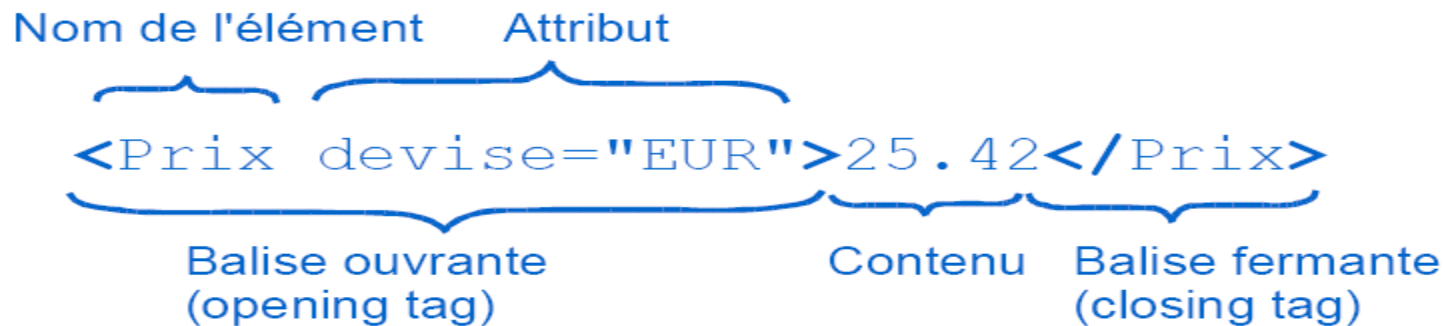


# TERMINOLOGIE XML: ÉLÉMENTS

Nom de l'élément      Attribut

`<Prix devise="EUR">25.42</Prix>`

Balise ouvrante  
(opening tag)      Contenu      Balise fermante  
(closing tag)

The diagram illustrates the components of an XML element using the example `<Prix devise="EUR">25.42</Prix>`. Above the code, two blue curly braces group the opening tag. The first brace, under `<Prix`, is labeled "Nom de l'élément". The second brace, under `devise="EUR"`, is labeled "Attribut". Below the code, three blue curly braces group the opening tag, the content, and the closing tag. The first brace, under `<Prix devise="EUR">`, is labeled "Balise ouvrante (opening tag)". The second brace, under `25.42`, is labeled "Contenu". The third brace, under `</Prix>`, is labeled "Balise fermante (closing tag)".

# CORRECTION XML

- Document **bien formé**(well-formed): correction syntaxique
  - Les balises fermantes sont toujours obligatoire(sauf éléments vides)
  - Le document comporte un seul élément racine
  - Les étiquettes sont correctement imbriquées (pas de croisement)
  - Guillemets(ou apostrophes ) obligatoire autour des valeurs
  - ...
- Document **valide** (valid): conformité de type de document
  - Respecte la DTD ou le schéma qui lui est associé





# CORRECTION SYNTAXIQUE

## Règles à respecter 1

Un document XML doit comporter un ou plusieurs éléments.

### Bien formé

```
<text>Ceci est un document XML</text>
```

Document XML bien formé comportant un élément

### Bien formé

```
<text>Ceci est un  
<doctype>document XML</doctype>  
</text>
```

Document XML bien formé comportant plusieurs éléments

### Mal formé

```
??? Ceci est un document XML ???
```

Un document XML doit comporter au moins un élément



# CORRECTION SYNTAXIQUE

## Règles à respecter 2

Il y a exactement un élément appelé élément racine ou élément document.

### Bien formé

```
<book>Ceci est un livre</book>
```

<book> est l'élément racine

### Bien formé

```
<list>  
<item>Item 1</item>  
<item>Item 2</item>  
<item>Item 3</item>  
</list>
```

<list> est l'élément racine

### Mal formé

```
???  
<item>Item 1</item>  
<item>Item 2</item>  
<item>Item 3</item>  
???
```

Seul un élément racine est autorisé



# CORRECTION SYNTAXIQUE

## Règles à respecter 3

Le nom de la balise de fin d'un élément doit correspondre à celui de la balise de début.

Les noms tiennent compte des majuscules et des minuscules

### Mal formé

```
<list>  
<item>Voiture</itm>  
<item>Avion</ITEM>  
<item>Train</item>  
</list>
```

<item> - </itm> et <item> - </ITEM> ne correspondent pas

# CORRECTION SYNTAXIQUE

## Règles à respecter 4

Les éléments délimités par les balises de début et de fin doivent s'imbriquer correctement les uns dans les autres.

### Mal formé

```
<text>  
    <bold><italic>XML</bold></italic>  
</text>
```



# CORRECTION SYNTAXIQUE

## Règles à respecter 4

Chaque élément comporte une balise de fin ou adopte la forme spéciale.

Il n'y a aucune différence entre `<AAA></AAA>` et `<AAA/>` en XML.

### Mal formé

```
<description>  
Il y a des pommes <color>jaunes<color> et <color>rouges</color>.  
</description>
```



# CORRECTION SYNTAXIQUE

## Règles à respecter 6

- Les noms d'éléments peuvent comporter des lettres, des chiffres, des tirets, des traits de soulignement, des deux-points ou des points.
- Le caractère deux-points (:) ne peut être utilisé que dans le cas particulier où il sert à séparer des espaces de noms.
- Les noms d'éléments commençant par xml, XML ou une autre combinaison de la casse de ces lettres sont réservés à la norme XML.



# CORRECTION SYNTAXIQUE

Document comportant des caractères autorisés

Bien formé

```
<permittedNames>
  <name/>
  <xsl:copy-of/>
  <A_long_element_name/>
  <A.name.separated.with.full.stops/>
  <a123323123-231-231/>
  <_12/>
</permittedNames>
```

Ce document comporte plusieurs erreurs.

Mal formé

```
<forbiddenNames>
  <A;name/>
  <last@name>
  <@###%^( )%+?= />
  <A*2/>
  <lex/>
</forbiddenNames>
```

Les noms ne peuvent pas commencer par xml

Mal formé

```
<forbiddenNames>
  <xmlTag/>
  <XMLTag/>
  <XmLTag/>
  <xMlTag/>
  <xmLTag/>
</forbiddenNames>
```



# CONFORMITÉ DE TYPE DE DOCUMENT

## Exemple de DTD

Fichier XML

Déclaration dans la DTD

```
<?xml version="1.0"?>
```

```
<S>
```

```
  <NP>
```

```
    <DET>le</DET>
```

```
    <NP1>
```

```
      <ADJ>petit</ADJ>
```

```
      <N>garçon</N>
```

```
    </NP1>
```

```
  </NP>
```

```
  <VP>
```

```
    <V>lit</V>
```

```
    <NP>
```

```
      <DET>un</DET>
```

```
      <N>livre</N>
```

```
    </NP>
```

```
  </VP>
```

```
</S>
```

<ELEMENT NP (DET, NP1)>

<ELEMENT VP (V, NP)>



# CONFORMITÉ DE TYPE DE DOCUMENT

## Exemple de schéma XML

### XML Schema Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```



# TP : CRÉATION DE PROJET XML(ECLIPSE)



# STRUCTURE DES DOCUMENTS XML



# STRUCTURE DES DOCUMENTS XML

- L' en-tête : le prologue
  - Les instructions de traitement
  - Les commentaires
  - La déclaration du type de document
  - Les noeuds élément
  - Les attributs d' un élément
  - Choix entre éléments et attributs
  - Les noeuds textes
  - Les entités du document
  - Quelques règles de syntaxe
- *Application des espaces de noms dans un document XML*
  - *Utilisation des espaces de noms dans un document XML*
  - L' espace de noms explicite
  - La suppression d' un espace de noms



# STRUCTURE GLOBALE DE DOCUMENT XML

```
<?xml version="1.0" encoding="utf-8"?>
```

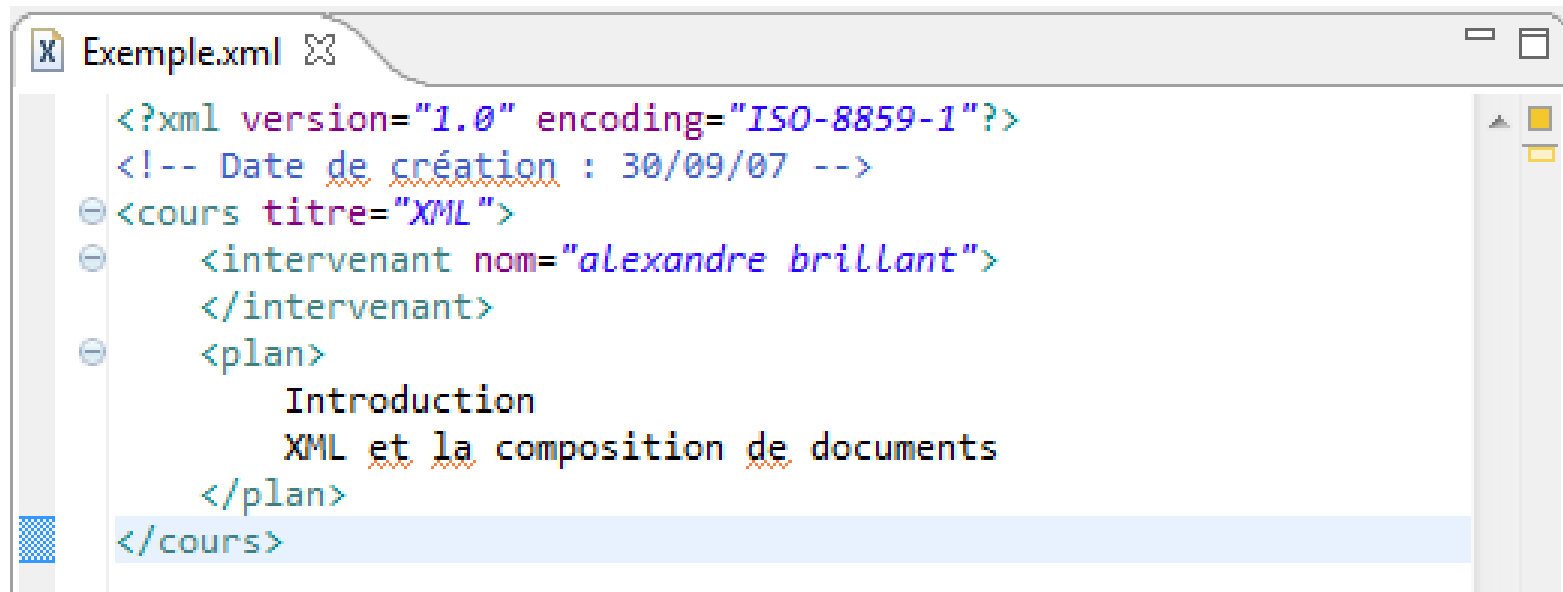
```
<!DOCTYPE nom SYSTEM "fichier.dtd ou URL" [  
  déclarations  
>
```

```
... corps du document ...
```



# STRUCTURE D'UN DOCUMENT XML

Commençons par prendre un exemple simple de document XML :



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Date de création : 30/09/07 -->
<cours titre="XML">
  <intervenant nom="alexandre brillant">
  </intervenant>
  <plan>
    Introduction
    XML et la composition de documents
  </plan>
</cours>
```

Nous allons maintenant décortiquer la syntaxe et en comprendre les tenants et les aboutissants.

# STRUCTURE D' UN DOCUMENT XML

## *L' en-tête : le prologue*

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

- Première ligne d' un document XML (pas d' espaces entre le début du document et cet élément) servant à donner les caractéristiques globales du document, c' est-à-dire :
- La version XML, soit 1.0 ou 1.1
- Le jeu de caractères employé (*encoding*).  
Indique à l'interpréteur XML [Parser] le jeu de caractères à utiliser.  
Lorsque l' encodage n' est pas précisé, c' est le standard UTF-8 qui est employé.

Rem: le prologue n' est pas obligatoire , défaut (version 1.0 et encoding UTF-8)



# STRUCTURE D' UN DOCUMENT XML

## *Les instructions de traitement*

- Elles servent à donner à l' application qui utilise le document XML des informations.
- Un cas typique est l' utilisation avec les navigateurs Mozilla Firefox ou Internet Explorer pour effectuer la transformation d' un document XML en document XHTML affichable avec l' instruction :

```
<?xml-stylesheet type="text/xsl" href="affichage.xsl"?>
```





# STRUCTURE D' UN DOCUMENT XML

## *Les commentaires*

- Ils se positionnent n'importe où après le prologue et peuvent figurer sur plusieurs lignes.

```
<!-- Date de création : 30/09/07 -->
```

Rem: les caractères -- sont interdits comme commentaires



# STRUCTURE D'UN DOCUMENT XML

## *La déclaration du type de document*

```
<!DOCTYPE racine SYSTEM "URI vers la DTD">
```

- Cette déclaration optionnelle sert à attacher une grammaire de type DTD (*Document Type Definition*) à votre document XML.
- **racine** est le premier élément (la première balise).
- L'URI peut être absolue(Ex: URL) ou relative au document.
- Exemple: 

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



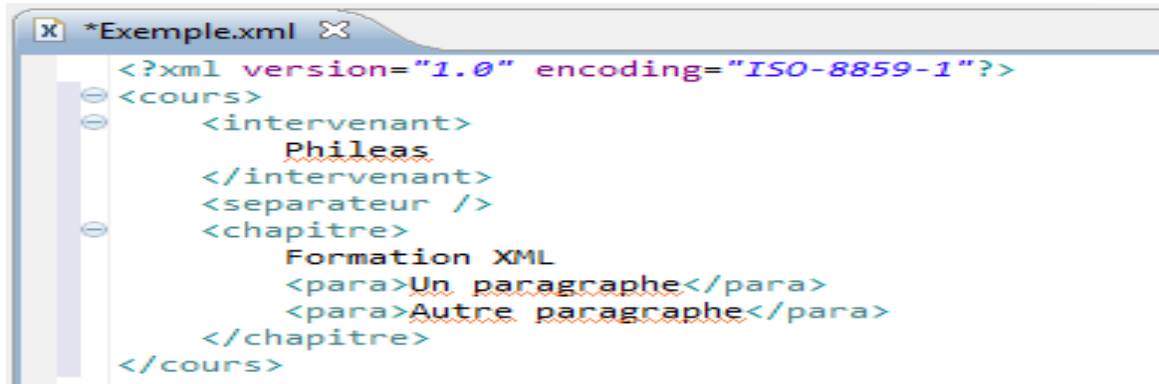
# STRUCTURE D' UN DOCUMENT XML

## *Les nœuds élément*

- Les éléments gèrent la structuration des données d' un document XML, un peu à la manière des répertoires qui servent à l' organisation des fichiers.
- *Pour décrire ce que contiennent les éléments, on parle de modèle de contenu. On trouve :*
  - Rien : il n' y pas de contenu, l' élément est vide.
  - Du texte : nous détaillerons par la suite cette notion.
  - Un ou plusieurs éléments
  - Un mélange de textes et d' éléments (forme rare)



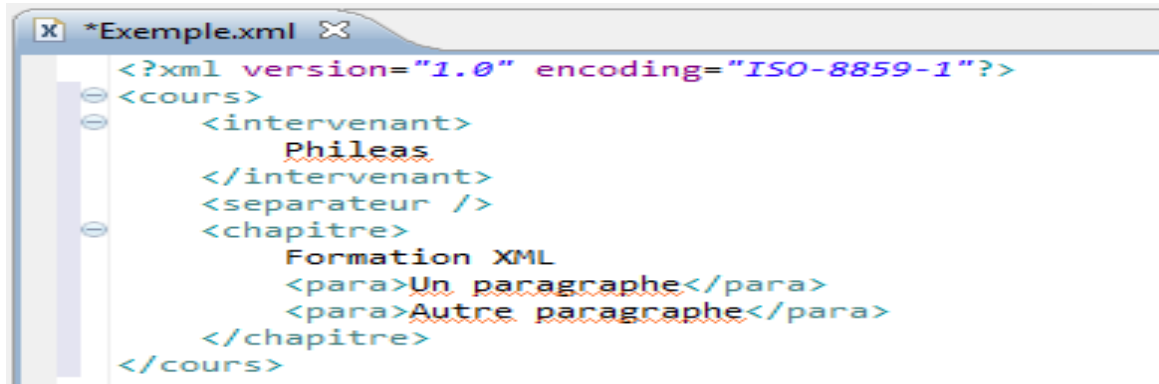
# STRUCTURE D'UN DOCUMENT XML



- **cours** : élément racine contenant trois éléments fils : intervenant, separateur et chapitre ;
- **intervenant** : élément contenant du texte ;
- **separateur** : élément sans contenu ;
- **chapitre** : élément contenant du texte et des éléments fils para ;
- **para** : élément contenant du texte.



# STRUCTURE D' UN DOCUMENT XML

A screenshot of a text editor window titled '\*Exemple.xml'. The editor displays an XML document with the following content:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cours>
  <intervenant>
    Phileas
  </intervenant>
  <separateur />
  <chapitre>
    Formation XML
    <para>Un paragraphe</para>
    <para>Autre paragraphe</para>
  </chapitre>
</cours>
```

The XML code is color-coded: opening and closing tags are in blue, the XML declaration is in purple, and text content is in black. The text 'Phileas' and the paragraph contents are underlined with red dashed lines. The editor has a sidebar on the left with expand/collapse icons.

Si maintenant nous nous penchons sur la syntaxe, nous avons donc :

- **<element>** : balise ouvrante.
- **</element>** : balise fermante.
- **<element/>** : balise ouverte et fermée.

C'est l'équivalent de **<element></element>**. Elle désigne donc un élément vide.



# STRUCTURE D' UN DOCUMENT XML

## *Les attributs d' un élément*

- Un attribut est un couple (clé, valeur) associé à la définition d' un élément.
- Il est localisé dans la balise ouvrante de l' élément.
- Un élément peut donc avoir de 0 à n attributs **uniques**.

Voici un exemple de document XML avec des attributs :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cours>
  <intervenant nom="fog" prenom="phileas" />
  <introduction />
  <chapitre numero="1">
    Formation XML
    <paragraphe>Détails du format</paragraphe>
  </chapitre>
</cours>
```



# STRUCTURE D' UN DOCUMENT XML

## *Choix entre éléments et attributs*

- L'attribut peut sembler superflu. En effet, ce qui s'écrit avec des attributs peut également l'être en s'appuyant uniquement sur des éléments.

Exemple :

- Cas avec attributs :

```
<personne nom="brillant" prenom="alexandre" />
```

- Cas sans attribut :

```
<personne>
```

```
  <nom>brillant</nom>
```

```
  <prenom>alexandre</prenom>
```

```
</personne>
```



# STRUCTURE D' UN DOCUMENT XML

## *Choix entre éléments et attributs (suite...)*

Règles simples pour déterminer s' il est préférable d' utiliser un attribut ou un élément.

- Lorsqu' une valeur est de taille modeste, a peu de chance d' évoluer vers une structure plus complexe, et n' est pas répétée, alors l' attribut peut tout à fait convenir.
- Dans tous les autres cas, **l' élément** reste incontournable.





# STRUCTURE D'UN DOCUMENT XML

## Les nœuds textes

- Dans un document XML, **ce qui est appelé donnée** est le texte qui est associé à **l'attribut**(=valeur), ou à **l'élément**(=contenu).
- Dans le vocabulaire propre à DOM (Document Object Model), la donnée apparaît comme un nœud texte.
- Certains caractères sont réservés dans le langage XML, il faut être vigilant lors de l'écriture des données.

```
<calcul>  
  if ( a<b et b>c) ...  
</calcul>
```

Exemple:

<b et b> n'est pas une balise mais fait partie des données liées à l'élément calcul.



# STRUCTURE D' UN DOCUMENT XML

## Les nœuds textes (suite...)

Pour résoudre ce problème, nous disposons d' entités prédéfinies.

Voici la liste des entités prédéfinies :

- &lt; équivalent de < (less than) ;
- &gt; équivalent de > (greater than) ;
- &amp; équivalent de & (ampersand) ;
- &quot; équivalent de " (quote) ;
- &apos; équivalent de ' (apostrophe).

L' exemple précédent peut donc être co

```
If (a&lt;b et b&gt;c)
```



# STRUCTURE D' UN DOCUMENT XML

## Les nœuds textes (suite...)

- Les entités prédéfinies présentes en trop grand nombre dans un même bloc peuvent alourdir inutilement le document.
- Dans le cas du contenu textuel d' un élément (et uniquement dans ce cas), nous disposons des sections CDATA (Character Data).
- Cette section doit être considérée comme un bloc de texte dont les caractères seront pris tel quel par le parseur jusqu' à la séquence de fin `]]>`.

### Exemple:

```
<![CDATA[  
  <element>C'est un document XML  
  </element>  
]]>
```

- Dans cet exemple, `<element>` n' est pas considéré comme une balise de structuration, mais comme du texte.

# STRUCTURE D' UN DOCUMENT XML

## *Quelques règles de syntaxe*

Ces règles de syntaxe sont à respecter impérativement pour le nommage d' un éléments xml.

- Le nom d' un élément ne peut commencer par un chiffre.
- Si le nom d' un élément est composé d' un seul caractère il doit être dans la plage [a-zA-Z] ou \_ ou :.
- Avec au moins 2 caractères, le nom d' un élément peut contenir \_, -, . et : plus les caractères alphanumériques (attention, le caractère : est réservé à un usage avec les espaces de nom que nous aborderons par la suite).

○ Ref: <http://www.w3.org/TR/2006/REC-xml-20060816/#sec-well-formed>



# STRUCTURE D' UN DOCUMENT XML

## Quelques conventions de nommage

Voici quelques conventions souvent employées dans les documents XML :

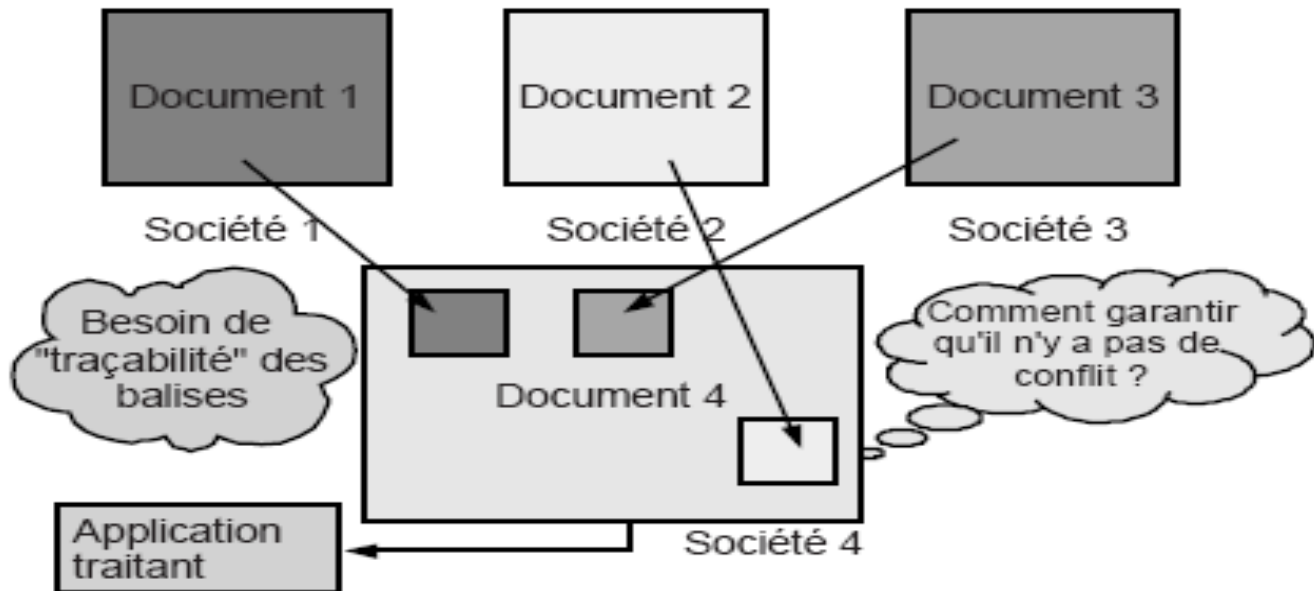
- Employer des minuscules pour les attributs et les éléments.
- Éviter les accents dans les noms d'attributs et d'éléments pour des raisons de compatibilité avec les outils du marché qui proviennent souvent d'un univers anglo-saxon.
- Préférer les guillemets délimitant les valeurs d'attribut.
- Séparer les noms composés de plusieurs mots par les caractères -, \_, . ou une majuscule.

# LES ESPACES DE NOMS

Les espaces de noms sont un concept très commun en informatique.

- Par exemple, dans le langage de programmation Java, les packages **servent** à **délimiter** la portée d'une classe.

## Application des espaces de noms dans un document XML




# LES ESPACES DE NOMS

## Application des espaces de noms dans un document XML(suite...)

- Pour délimiter la portée d'une balise, d'un attribut ou d'une valeur d'attribut, nous disposons d'espaces de noms (namespace).
- L'utilisation des espaces de noms garantit une forme de traçabilité de la balise et évite les ambiguïtés d'usage.
- Pour que les espaces de noms aient un sens, il faut pour chacun d'eux un identifiant unique.
- Cet identifiant unique peut être simplement l'URL, puisqu'il ne peut y avoir qu'un propriétaire pour une URL donnée.

**Attention:** L'URL ne signifie pas qu'il doit y avoir un document sur votre serveur HTTP



# LES ESPACES DE NOMS

## Vocabulaire : qualification des éléments

- Un élément qui est connu dans un espace de noms est dit **qualifié** ; dans le cas contraire, il est dit **non qualifié**.





# LES ESPACES DE NOMS

## L'espace de noms par défaut

Un premier usage consiste à utiliser simplement l'espace de noms par défaut.

- Ce dernier est précisé par un pseudo-attribut **xmlns**.
- La valeur associée sera une URL garantissant l'unicité de l'espace de noms.
- L'espace de noms par défaut s'applique à l'élément où se situe sa déclaration et à tout son contenu.

## Exemple:

```
<chapitre xmlns="http://www.masociete.com">  
  <paragraphe>  
    ...  
  </paragraphe>  
</chapitre>
```

Ici l'élément chapitre est dans l'espace de noms `http://www.masociete.com`.

C'est également le cas de l'élément paragraphe, puisqu'il est dans l'élément chapitre.



# LES ESPACES DE NOMS

## L' espace de noms par défaut(suite...)

- Nous pouvons changer l' espace de noms par défaut même dans les éléments enfants : dans ce cas, une règle de priorité est appliquée.

Attention, les espaces de noms ne sont pas imbriqués ; on ne peut appliquer Qu'un seul espace de noms à la fois.

### Exemple:

```
<chapitre xmlns="http://www.masociete.com">  
  <paragraphe xmlns="http://www.autresociete.com">  
    ...  
  </paragraphe>  
</chapitre>
```

L' élément paragraphe n' appartient pas à l' espace de noms `http://www.masociete.com` mais uniquement à l' espace de noms `http://www.autresociete.com`.



# LES ESPACES DE NOMS

## L'espace de noms explicite

- Pour disposer de davantage de souplesse dans ces espaces et pouvoir également les appliquer aux attributs et valeurs d'attributs, la syntaxe introduit la notion de préfixe.
- On déclare un préfixe comme un pseudo-attribut commençant par **xmlns:prefixe**.
- Une fois déclaré, il est employable uniquement dans l'élément le déclarant et dans son contenu.

### Exemple:

```
<p:resultat xmlns:p="http://www.masociete.com">  
</p:resultat>
```

L'élément résultat est dans l'espace de noms `http://www.masociete.com` grâce au préfixe `p`.

# LES ESPACES DE NOMS

## L'espace de noms explicite(suite...)

On peut déclarer et utiliser plusieurs espaces de noms grâce aux préfixes.

*Exemple:*

```
<p:res xmlns:p="http://www.masociete.com" xmlns:p2="http://www.autresociete.com">  
  <p2:res>  
  </p2:res>  
</p:res>
```

Le premier élément **res** est dans l'espace de noms `http://www.masociete.com` alors que l'élément `res` à l'intérieur est dans l'espace de noms `http://www.autresociete.com`.



# LES ESPACES DE NOMS

## La suppression d'un espace de noms

- Aucun espace de noms n'est utilisé lorsqu'il n'y a pas d'espace de noms par défaut ni de préfixe.

### Exemple:

```
<p:element xmlns:p="http://www.masociete.com">  
  <autrelement />  
</p:element>
```

L'élément **element** est dans l'espace de noms `http://www.masociete.com` alors que l'élément **autrelement**, qui n'est pas préfixé, n'a pas d'espace de noms.



# LES ESPACES DE NOMS

## La suppression d'un espace de noms (suite...)

- Pour supprimer l'action d'un espace de noms il suffit d'utiliser la valeur vide "", ce qui revient à ne pas avoir d'espace de noms.

### Exemple:

```
<element xmlns="http://www.masociete.com">
  <autreelement xmlns="">
    .. Aucun d'espace de noms
  </autreelement>
  <encoreunelement>
    ... Espace de nom par défaut
  </encoreunelement>
</element>
```

L'élément **element** est dans l'espace de noms `http://www.masociete.com` alors que l'élément **autreelement** n'est plus dans un espace de noms.

L'élément **encoreunelement** se trouve également dans l'espace de noms `http://www.masociete.com`, de par l'espace de noms de son parent.



# LES ESPACES DE NOMS

## Application d'un espace de noms sur un attribut

- Les espaces de nom peuvent s'appliquer via un préfixe sur un attribut ou une valeur d'attribut.
- Cet emploi peut servir à :
  - contourner la règle qui veut que l'on ne puisse pas avoir plusieurs fois un attribut de même nom sur une déclaration d'élément.
  - lever l'ambiguïté sur une valeur d'attribut

### Exemple:

```
<livre xmlns:p="http://www.imprimeur.com" p:quantite="p:50lots">  
  <papier type="p:A4" />  
</livre>
```

Dans cet exemple, nous avons qualifié l'attribut quantité ainsi que les valeurs d'attribut 50lots et A4.

# TP : STRUCTURE DES DOCUMENTS XML





# VALIDATION DES DOCUMENTS XML



# VALIDATION DES DOCUMENTS XML

## Rôle de la validation dans l'entreprise

### La première forme de validation par DTD

- *La définition d'un élément*
- *La définition d'un attribut*
- *La définition d'une entité*

### La validation par un schéma W3C

- Les différentes formes de type
- Les définitions globales et locales
- L'assignation d'un schéma à un document XML
- Les catégories de type simple
- L'utilisation des types complexes
- Les définitions d'éléments
- Réutilisation des définitions
- L'utilisation des clés et références de clés
- Relations entre schémas



# RÔLE DE LA VALIDATION DANS L' ENTREPRISE

- La validation va renforcer la qualité des échanges entre l' émetteur de données et le consommateur de données XML.
- Par cohérence, il faut entendre :
  - à la fois le vocabulaire (éléments, attributs et espaces de noms),
  - l' ordre,
  - et les quantités.
- La plupart des outils, et notamment les parseurs XML, proposent des outils de validation.
- Les parseurs courants supportent une ou plusieurs formes de grammaires.
  - **Les DTD** (*Document Type Definition*), sont présentes dans la plupart des outils.
  - **Les schémas** W3C, une forme de grammaire plus moderne et plus complexe.



# RELATIONS DE CONFORMITÉ

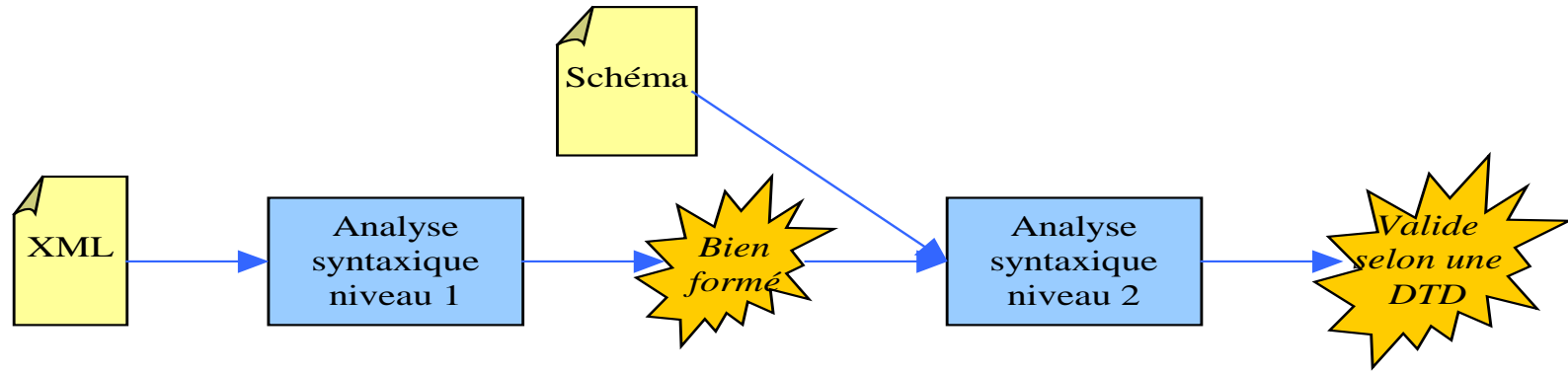
« *Est conforme à* »



<b>Programmation</b>	Définition de classe	Instance d'objet
<b>SGBD</b>	Définition de table	Intance de table
<b>XML</b>	Schéma de données	Instance de document



# VALIDATION DE DOCUMENT



# PRINCIPAUX SCHÉMAS DE DONNÉES

- Les DTDs (de SGML)
- Les schémas XML (W3C) :
  - <http://www.w3.org/TR/xmlschema-0/>
- RELAX :
  - <http://www.xml.gr.jp/relax/>
- Tree Regular Expression (TREG) :
  - <http://www.thaiopensource.com/trex/>
- Relax-NG :
  - <http://www.oasis-open.org/committees/relax-ng/>
- Schematron :
  - <http://www.ascc.net/xml/resource/schematron/schematron.html>



# VALIDATION PAR DTD



# LA PREMIÈRE FORME DE VALIDATION PAR DTD

- Une DTD (*Document Type Definition*)
  - Avantage : rapide à écrire
  - Inconvénient: pauvre en possibilités de contrôle (typage de données, par exemple).
- Une DTD peut être interne ou externe au document XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ARTICLES [
  <ELEMENT ARTICLES (ARTICLE)*>
  <ELEMENT ARTICLE (ARTICLEDATA)*>
  <ELEMENT ARTICLEDATA (TITLE, AUTHOR)>
  <ELEMENT TITLE (#PCDATA)>
  <ELEMENT AUTHOR (#PCDATA)>
]>
<ARTICLES>
  <ARTICLE>
    <ARTICLEDATA>
      <TITLE>XML Demystified</TITLE>
      <AUTHOR>Jaidev</AUTHOR>
    </ARTICLEDATA>
  </ARTICLE>
</ARTICLES>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ELEMENT ARTICLES (ARTICLE)*>
<ELEMENT ARTICLE (ARTICLEDATA)*>
<ELEMENT ARTICLEDATA (TITLE,
AUTHOR)>
<ELEMENT TITLE (#PCDATA)>
<ELEMENT AUTHOR (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ARTICLES SYSTEM
"D:\articles.dtd">
<ARTICLES>
  <ARTICLE>
    <ARTICLEDATA>
      <TITLE>XML Demystified</TITLE>
      <AUTHOR>Jaidev</AUTHOR>
    </ARTICLEDATA>
  </ARTICLE>
</ARTICLES>
```



# DÉCLARATION D'ÉLÉMENT

## ○ Syntaxe

`<!ELEMENT nom modèleDeContenu >`

- *nom* : nom de l'élément
- *modèleDeContenu* : expression définissant le contenu autorisé dans l'élément



# DÉCLARATION D'ÉLÉMENT

Quelques exemples :

```
<!ELEMENT personne (nom_prenom | nom)>  
<!ELEMENT nom_prenom (#PCDATA)>  
<!ELEMENT nom (#PCDATA)>
```

Cela nous autorise deux documents XML, soit :

```
<personne>  
  <nom_prenom>Brillant Alexandre</nom_prenom>  
</personne>
```

ou bien

```
<personne>  
  <nom>Brillant</nom>  
</personne>
```

L'opérateur de choix, | , indique que l'un ou l'autre de deux éléments doit être présent.

L'opérateur de suite (ou séquence), indique que les deux éléments doivent être présents.

# La première forme de validation par DTD

## 5 TYPES D'ÉLÉMENTS

- élément vide  
    <!ELEMENT nom EMPTY>
- élément avec contenu indifférent  
    <!ELEMENT nom ANY>
- élément avec du texte seulement comme contenu  
    <!ELEMENT nom (#PCDATA)>
- élément avec des éléments seuls comme contenu  
    <!ELEMENT nom (nom1 | nom2?)>  
    <!ELEMENT nom (nom1 , (nom2 | nom3)\*)>
- élément mixte  
    <!ELEMENT nom (#PCDATA | nom1 | nom2)\*>



# OPÉRATEURS D'EXPRESSIONS RÉGULIÈRES

<i>Sémantique</i>	<i>Opérateur</i>
Enchaînement	$\dots , \dots$
Choix	$\dots \mid \dots$
Zéro ou 1	$\dots ?$
Zéro ou plus	$\dots ^*$
Un ou plus	$\dots ^+$
Groupe	$(\dots)$
un et un seul	nom de l'élément



## Quelques exemples :

**<!ELEMENT plan (introduction?, chapitre+, conclusion?)>**

L'élément plan contient un élément introduction optionnel, suivi d'au moins un élément chapitre et suivi par un élément conclusion optionnel.

**<!ELEMENT chapitre (auteur\*, paragraphe+)>**

L'élément chapitre contient de 0 à n éléments auteur suivi d'au moins un élément paragraphe.

**<!ELEMENT livre (auteur?, chapitre)+>**

L'élément livre contient au moins un élément, chaque élément, étant un groupe d'éléments où l'élément auteur, est optionnel et l'élément chapitre est présent en un seul exemplaire.



# DÉCLARATION D'ATTRIBUT

## *La définition d'un attribut*

- Les attributs sont précisés dans l'instruction **ATTLIST**.
- Cette dernière, étant indépendante de l'instruction **ELEMENT**, on précise à nouveau le nom de l'élément sur lequel s'applique le ou les attributs.
- On peut considérer qu'il existe cette forme syntaxique :

nom TYPE OBLIGATION VALEUR\_PAR\_DEFAULT



# LA DÉFINITION D' UN ATTRIBUT(SUITE...)

Le **TYPE** peut être principalement :

- CDATA : du texte (*Character Data*) ;
- ID : un identifiant unique (combinaison de chiffres et de lettres) ;
- IDREF : une référence vers un ID ;
- IDREFS : une liste de références vers des ID (séparation par un blanc) ;
- NMTOKEN : un mot (donc pas de blanc) ;
- NMTOKENS : une liste de mots (séparation par un blanc) ;
- Une énumération de valeurs : chaque valeur est séparée par le caractère |.



# VALEURS ID, IDREF, ENTITY, NMTOKEN

## ○ Syntaxe

- doivent respecter la syntaxe des noms d'éléments

## ○ Contraintes

- Un attribut ID doit identifier de manière unique un élément au sein d'un document considéré (*contrainte d'unicité*)
- un attribut IDREF est contraint à prendre la valeur d'un attribut ID existant dans le document (*contrainte de référence*)





# TYPE ÉNUMÉRÉ

```
<!ELEMENT voie          (#PCDATA) >
```

```
<!ATTLIST voie  
  type (rue | avenue | impasse | cours  
        | square | boulevard | chemin | allée  
        | quai | route | passage | place  
        | rondPoint ) 'rue' >
```



# CONTRAINTES D'OCCURRENCE

Type de contrainte	Expression de la contrainte
Val par défaut d'un type énuméré	'val '
Val par défaut	#DEFAULT val
Obligatoire	#REQUIRED
Non obligatoire	#IMPLIED
Valeur constante	#FIXED val



## Quelques exemples

**<!ATTLIST chapitre**

**titre CDATA #REQUIRED**

**auteur CDATA #IMPLIED>**

L'élément chapitre possède ici un attribut titre obligatoire et un attribut auteur optionnel.

**<!ATTLIST crayon**

**couleur (rouge | vert | bleu) "bleu">**

L'élément crayon possède un attribut couleur dont les valeurs font partie de l'ensemble rouge, vert, bleu.



# RAPPEL - ENTITÉS CARACTÈRES

- Notation qui permet de désigner un caractère unicode par son code

*En hexadécimal* →

*En décimal* →

Référence	Caractère
&#238 ;	î
&#x2200 ;	∀
&#x0152 ;	Œ
&#xA9 ;	©

- Utilisation "le gîte et le couvert"

<titre>le g&#238;te et le couvert<titre>



# ENTITÉ – QU'EST-CE QUE C'EST ?

- Sorte de *d'abréviations* (ou de macro) qui associe
  - un nom d'entité
  - à un contenu d'entité qui est
    - Un simple texte ou un fragment de document XML
- Définition
  - <!ENTITY dtd "Document Type Definition">
  - <!ENTITY chap1 SYSTEM "chapitre1.xml">
- Utilisation on pose une référence
  - dans les contenus d'élément ou dans les valeurs d'attributs  
&dtd; ou &chap1;
  - la référence est remplacée par le contenu de l'entité



# ENTITÉS -SYNTAXE

- Schéma de la définition

```
<!ENTITY nom [SYSTEM] "valeur">
```

- Syntaxe des références

```
&nom;
```

- La valeur associée peut contenir des balises :

```
<!ENTITY afcepf "Institut de Formation Supérieur ...">
```

```
<!ENTITY piedDePage '<hr size="1"/>'
```



# TROIS USAGES

- Créer une abréviation

```
<!ENTITY dtd "Document Type Definition">
```

- Créer un lien vers une source de données externe (construction modulaire)

```
<!ENTITY chap1 SYSTEM "chapitre1.xml">
```

- Exprimer la transcriptions de signes spéciaux.

```
<!ENTITY euro "&#x20AC;">
```



# CONSTRUCTION MODULAIRE

```
<?xml version="1.0"?>
```

```
<!DOCTYPE livre [
```

```
  <!ELEMENT livre (html*)>
```

```
  <!ENTITY chapitre1 SYSTEM "cours1/cours.xml">
```

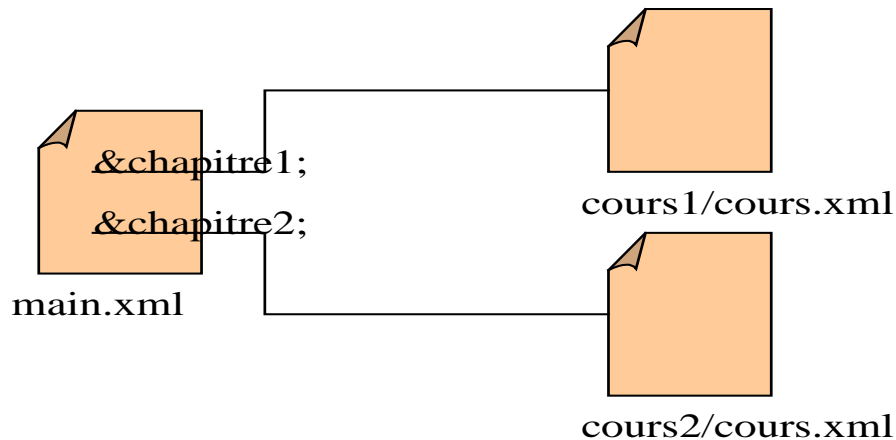
```
  <!ENTITY chapitre2 SYSTEM "cours2/cours.xml">
```

```
<livre>
```

```
&chapitre1;
```

```
&chapitre2;
```

```
</livre>
```





# CARACTÈRES SPÉCIAUX

- 5 entités prédéfinies

Référence	glyphe	Nom
<code>&amp;amp; ;</code>	<code>&amp;</code>	ampersand
<code>&amp;lt; ;</code>	<code>&lt;</code>	plus petit
<code>&amp;gt; ;</code>	<code>&gt;</code>	plus grand
<code>&amp;apos; ;</code>	<code>'</code>	apostrophe
<code>&amp;quot; ;</code>	<code>"</code>	double quote

- Utilisation : `"A >5"`  
`< if > A &gt; 5 < / if >`



# DEUX SORTES D' ENTITÉS

- *Entités générales* pour insérer du texte
  - dans la DTD
  - `<!ENTITY dtd "Definition type document">`
  - dans le document XML, en dehors de la DTD
  - Référence : `&dtd;`
- *Entités paramètres* pour insérer du texte
  - Dans la DTD seulement
  - `<!ENTITY % contenuAdresse "ville, rue">`
  - Référence : `%contenuAdresse;`



TP : LAB\_DTD



# LA VALIDATION PAR UN SCHÉMA W3C



# LA VALIDATION PAR UN SCHÉMA W3C

## Introduction

### Limitations des DTD

1. Premièrement, les DTD ne sont pas au format XML.
2. Deuxièmement, les DTD ne supportent pas les « espaces de nom »
3. Troisièmement, le « typage » des données est extrêmement limité.

### Apports des schémas

1. Le typage des données est introduit.
2. Le support des espaces de nom.
3. Les indicateurs d'occurrences des éléments peuvent être tout nombre non négatif
4. Les schémas sont très facilement concevables par modules.



# LA VALIDATION PAR UN SCHÉMA W3C

## Les premiers pas

- Le but d'un schéma est de définir une classe de documents XML.
- Il permet de décrire les autorisations d'imbrication et l'ordre d'apparition des éléments et de leurs attributs, tout comme une DTD.

## Structure de base

- Un premier point est qu'un fichier Schéma XML est un document XML.
- Comme tout document XML, un Schéma XML commence par un prologue, et a un élément racine.

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <!-- déclarations d'éléments, d'attributs et de types ici -->  
</xsd:schema>
```

- L'élément racine est l'élément xsd:schema.
- Tout élément d'un schéma doit commencer par le préfixe xs ou xsd.



# EXAMPLE

The screenshot shows a software window titled "XML Schema Validation". It has two main sections: "XML Schema Document:" and "XML Instance Document:". The schema section contains an XSD for a "contact" element with fields for givenName, familyName, birthdate, homeAddress, and workAddress. The instance section shows an XML document for a contact named John Doe, but it includes a "title" element which is not defined in the schema. This error is highlighted with a red background. At the bottom, the status bar says "Status: element title is not defined in this scope" and there is a "Validate" button.

XML Schema Validation

XML Schema Document: Contact Schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="contact">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="givenName" type="xsd:string"/>
        <xsd:element name="familyName" type="xsd:string"/>
        <xsd:element name="birthdate" type="xsd:date" minOccurs="0"/>
        <xsd:element name="homeAddress" type="address"/>
        <xsd:element name="workAddress" type="address" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

XML Instance Document: Invalid Contact Instance

```
<contact>
  <givenName>John</givenName>
  <familyName>Doe</familyName>
  <title>Prof.</title>
  <workAddress>
    <street>Sandakerveien 116</street>
    <zipCode>N-0550</zipCode>
    <city>Oslo</city>
    <country>Norway</country>
  </workAddress>
</contact>
```

Status: element title is not defined in this scope

Validate

# ÉLÉMENT <XS:SCHEMA>

- Élément racine d'un schéma XML

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema name="..." xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
  <!-- constructions de niveau 0 -->
```

```
</xs:schema>
```

- Constructions de niveau 0 (*composants globaux*)
  - Des déclarations d'élément
  - Des définitions de type complexes.
  - Des définitions de types simples
  - D'autres encore .../...





- *Les définitions globales et locales*

- Les composants globaux apparaissent au premier niveau au sein de l'élément `<xsd:schema>`

- Ils sont toujours nommés (`xsd:... name="..."`)
- Leur nom doit être unique au sein de leur type de composant

- Les composants locaux

- Leur nom a une portée locale au type complexe dans lequel ils sont définis
- Types simples et types complexes définis localement sont anonymes (ils ne peuvent être réutilisés)



# LES TYPES DE DONNÉES

- on distingue:

- **Types simples** ( ni attributs, ni éléments enfants).

```
<xs:element name="prénom" type="xs:string"/>
```

- Exemple

```
<prénom>Yves</prénom>
```

- **Types complexes** (pouvant contenir une suite de sous-éléments ou attributs).

```
<xs:element name="prénom">  
  <xs:complexType>  
    <xs:attribute name="val" type="xs:string" />  
  </xs:complexType>  
</xs:element>
```

- Exemple

```
<prénom val="yves"/>
```



# TYPES SIMPLES



# TYPE SIMPLE AVEC/SANS NOM

## ○ *Sans Nom*

```
<xs:element name="AGE">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1" />
      <xs:maxExclusive value="100" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

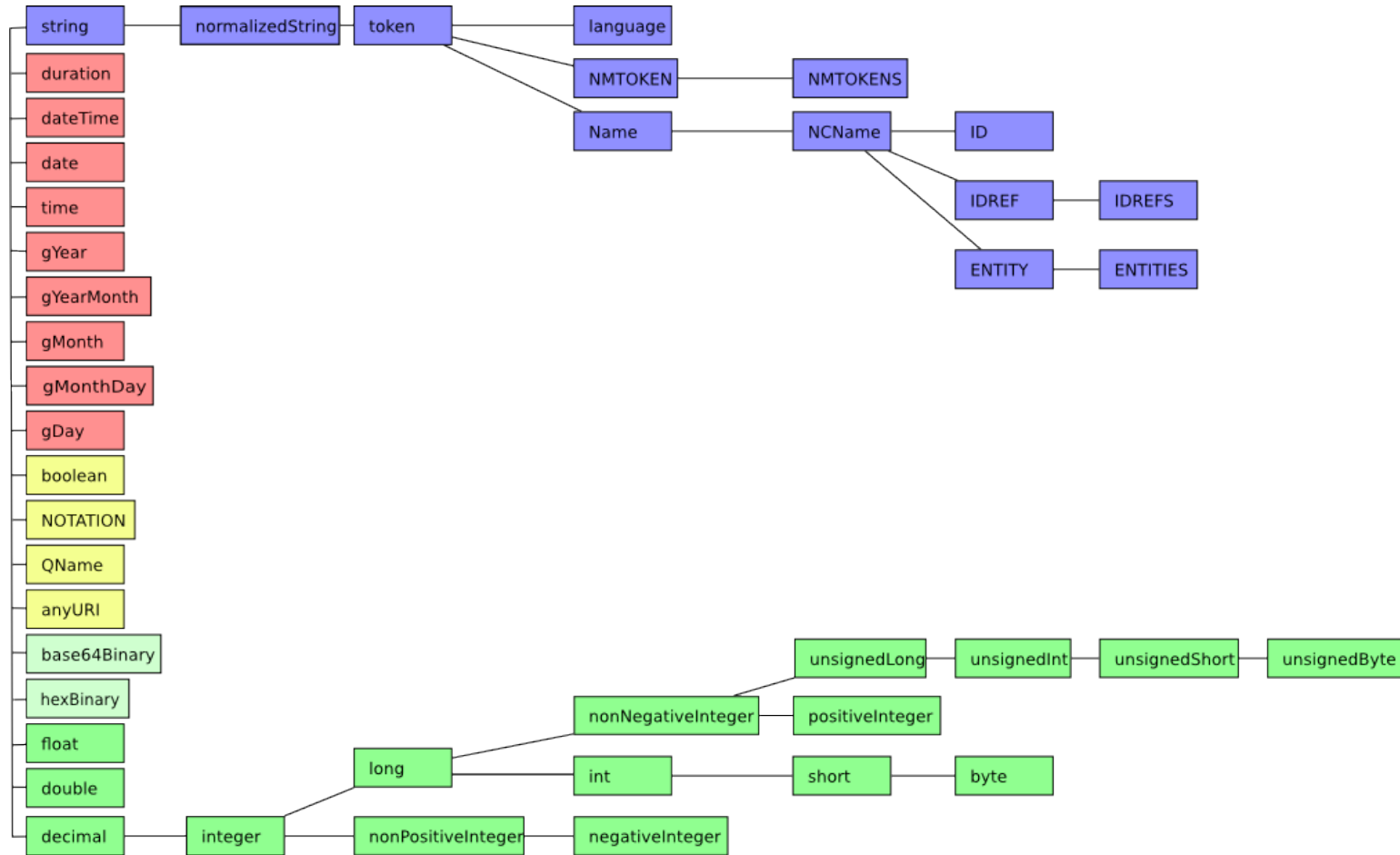
## ○ *Avec Nom*

```
<xs:simpleType name="ageType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="1" />
    <xs:maxExclusive value="100" />
  </xs:restriction>
</xs:simpleType>
```

<xs:element name="AGE" type="ageType">



# LES TYPES PRÉDÉFINIS



# DÉRIVATION DE TYPES EXISTANTS

## ○ Exemple 1:

```
<xs:element name="PHONE">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{3}-\d{7}" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

## ○ XML

- Valid Phone Number: 816-6724567
- Invalid Phone Number: 81-6724567



# DÉRIVATION DE TYPES EXISTANTS

## ○ Exemple 2:

```
<xs:element name="AGE">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1" />
      <xs:maxExclusive value="100" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

## ○ XML

- Valid Age: 67, 1, 99
- Invalid Age: 0 or 100 or -1



# DÉRIVATION DE TYPES EXISTANTS

## ○ Exemple 3:

```
<xs:element name="RGBCOLOR">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="RED" />
      <xs:enumeration value="GREEN" />
      <xs:enumeration value="BLUE" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

## ○ XML

- Valid RGB Color: RED, GREEN
- Invalid RGB Color: PURPLE, 4, PINK





# DÉRIVATION DE TYPES EXISTANTS

## ○ Exemple 4:

```
<xs:element name="COUNTRYCODE">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="2" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

## ○ XML

- Valid Country Code: US, UK, SG
- Invalid Country Code USA, GBR, SIN



# DÉRIVATION DE TYPES EXISTANTS

## ○ Exemple 5:

```
<xs:element name="PRICE">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:precision value="5" />
      <xs:scale value="2" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

## ○ XML

- Valid Price: 100.56, 9.9
- Invalid Price: 65.679, 54321.00



# DÉRIVATION DE TYPES EXISTANTS

## ○ Exemple 6:

```
<xs:element name="eng_degree" type="xs:string" default="BS">
```

- XML

- Valid Elements: <eng\_degree>BS</eng\_degree>

<eng\_degree>BE</eng\_degree>

## ○ Exemple 7:

```
<xs:element name="eng_degree" type="xs:string" fixed="BS">
```

- XML

Valid Element: <eng\_degree>BS</eng\_degree>

Invalid Element: <eng\_degree>BE</eng\_degree>



# TYPES COMPLEXES



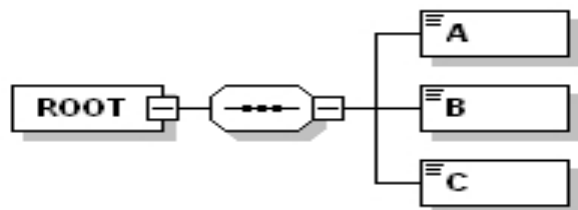
# TYPES COMPLEXES

- Un élément de type simple ne peut contenir pas de sous-élément.
- Il est nécessaire pour cela de le déclarer de type complexe.
- On peut alors déclarer des:
  - séquences **d'éléments**,
  - des types de **choix**
  - ou des **contraintes d'occurences**



# TYPES COMPLEXES

## ○ Séquences d'éléments



```
<xs:element name="ROOT">
  <xs:complexType mixed="false">
    <xs:sequence>
      <xs:element name="A" type="xs:string" />
      <xs:element name="B" type="xs:string" />
      <xs:element name="C" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

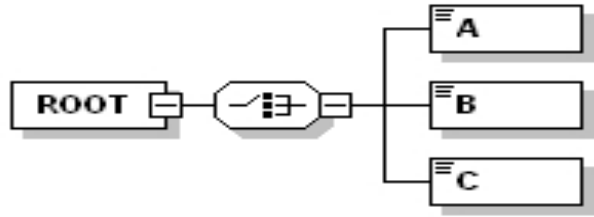
## ○ Exemple

```
<xs:element name="newspaper">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" />
      <xs:element name="publisher">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" />
            <xs:element name="address" />
            <xs:element name="registerDate" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="city" />
      <xs:element name="type" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<newspaper>
  <name>The XML Daily</name>
  <publisher>
    <name>XML Master</name>
    <address>SGML Lane, Museum Sq</address>
    <registerDate>12 July 2001</registerDate>
    <city>XCitee</city>
    <type>Daily</type>
  </publisher>
</newspaper>
```

# TYPES COMPLEXES

## Choix d'élément



## Exemple

```
<xs:element name="ROOT">
  <xs:complexType mixed="false">
    <xs:choice>
      <xs:element name="A" type="xs:string" />
      <xs:element name="B" type="xs:string" />
      <xs:element name="C" type="xs:string" />
    </xs:choice>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="identifier">
  <xs:complexType>
    <xs:choice>
      <xs:element name="name" />
      <xs:element name="identityNo" />
    </xs:choice>
  </xs:complexType>
</xs:element>
```

```
<identifier>
  <name>Jaidev</name>
</identifier>
<identifier>
  <identityNo>A12345</identityNo>
</identifier>
```

# TYPES COMPLEXES

## ○ L'élément all

```
<xs:element name="plan">
  <xs:complexType>
    <xs:all>
      <xs:element name="auteur" type="xs:string" />
      <xs:element name="chapitres" type="xs:string" />
    </xs:all>
  </xs:complexType>
</xs:element>
```

## ○ Exemple

```
<xs:element name="city">
  <xs:complexType>
    <xs:all>
      <xs:element name="name" />
      <xs:element name="state" />
      <xs:element name="elevation" />
      <xs:element name="mayor" />
    </xs:all>
  </xs:complexType>
</xs:element>
```

```
<city>
  <elevation>540m</elevation>
  <state>Kansas</state>
  <mayor>Alexey</mayor>
  <name>See Sharp City</name>
</city>
```





# DÉCLARATION D'ATTRIBUT

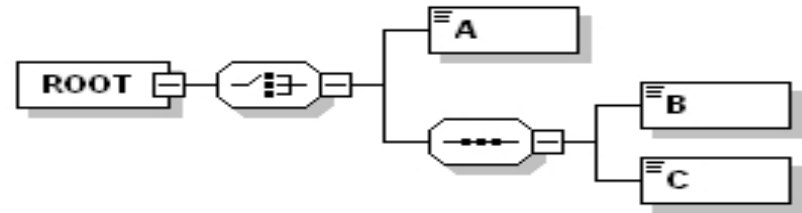
- **Si** un élément contient des attributs, **alors** les déclarations de ces derniers doivent se faire *juste avant* la balise fermante `</xsd:complexType>`.

## *Exemple:*

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="dateDeNaissance" type="xsd:date" />
    <xsd:element name="adresse" type="xsd:string" />
    <xsd:element name="adresseElectronique" type="xsd:string" />
    <xsd:element name="téléphone" type="numéroDeTéléphone" />
  </xsd:sequence>
  <xsd:attribute name="nom" />
  <xsd:attribute name="prénom" />
</xsd:complexType>
```



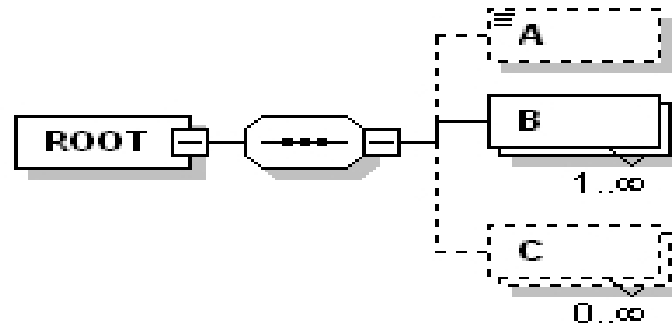
# EXEMPLE SUR LES TYPES COMPLEXES (1)



```
<xs:element name="ROOT">
  <xs:complexType mixed="false">
    <xs:choice>
      <xs:element name="A" type="xs:string" />
      <xs:sequence>
        <xs:element name="B" type="xs:string" />
        <xs:element name="C" type="xs:string" />
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

## EXEMPLE SUR LES TYPES COMPLEXES (2)

```
<xs:element name="ROOT">
  <xs:complexType mixed="false">
    <xs:sequence>
      <xs:element name="A" minOccurs="0" />
      <xs:element name="B" maxOccurs="unbounded" />
      <xs:element name="C" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



# INDICATEURS D'OCCURENCES

Dans une DTD	Valeur de <code>minOccurs</code>	Valeur de <code>maxOccurs</code>
*	0	unbounded
+	1 (pas nécessaire, valeur par défaut)	unbounded
?	0	1 (pas nécessaire, valeur par défaut)
rien	1 (pas nécessaire, valeur par défaut)	1 (pas nécessaire, valeur par défaut)
impossible	nombre entier n quelconque	nombre entier m quelconque supérieur ou égal à n



# CONSTRUCTION D'UN SCHÉMA



## PREMIER STYLE D'ÉCRITURE : "À PLAT"

- Utiliser une liste de définitions et déclarations *globales*
  - Toutes les définitions et déclarations sont directement au niveau 0 de l'élément `<xs:schema>`

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="ROOT" type="root"/>
<xs:element name="A" type="xs:string"/>
<xs:element name="B" type="xs:string"/>
<xs:complexType mixed="false" name="root">
  <xs:choice>
    <xs:element ref="A"/>
    <xs:element ref="B"/>
  </xs:choice>
</xs:complexType>
</xs:schema>
```

*Définition globale de type  
- peut être dérivé*



*Définitions globales d'éléments  
- peuvent être réutilisées partout*

## SECOND STYLE D'ÉCRITURE : "EN POUPÉES RUSSES"

- Définitions et déclarations sont mises en ligne

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ROOT">
    <xs:complexType mixed="false">
      <xs:choice>
        <xs:element name="A" type="xs:string"/>
        <xs:element name="B" type="xs:string"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

← *Définition locale de type*  
- ne peut pas être dérivée

↖ *déclaration locale d'élément*  
- ne peut pas être réutilisé ailleurs  
- y compris comme racine de document

# TP : LAB\_SCHÉMA





**XSL**



# XSL

- **XSL** se compose de deux parties:
  - XSLT: un langage pour transformer des documents XML
  - XPath : une langue pour la navigation dans des documents XML
- **Objectif** : référencer noeuds (éléments, attributs, commentaires, ...) dans un document XML.



# XPATH

## Présentation

- Le langage XPATH offre un moyen d'identifier un ensemble de nœuds dans un document XML.
- Toutes les applications ayant besoin de repérer un fragment de document XML peuvent utiliser ce langage.
- Les feuilles de style XSL, les pointeurs XPOINTER et les liens XLINK utilisent de manière intensive les expressions XPATH.
- XPATH est un premier pas vers un langage d'interrogation d'une base de données XML (XQuery).



# PATH

➤ Pour voir comment fonctionne XPath, nous allons voir:

1. comment XPATH utilise un modèle de représentation arborescente d'un document XML
2. Les expression Xpath ou comment cheminer dans un tel arbre, avec la notion de *chemin de localisation* (axes de localisation, les *filtres*, et les *prédicats*.)



# XPATH

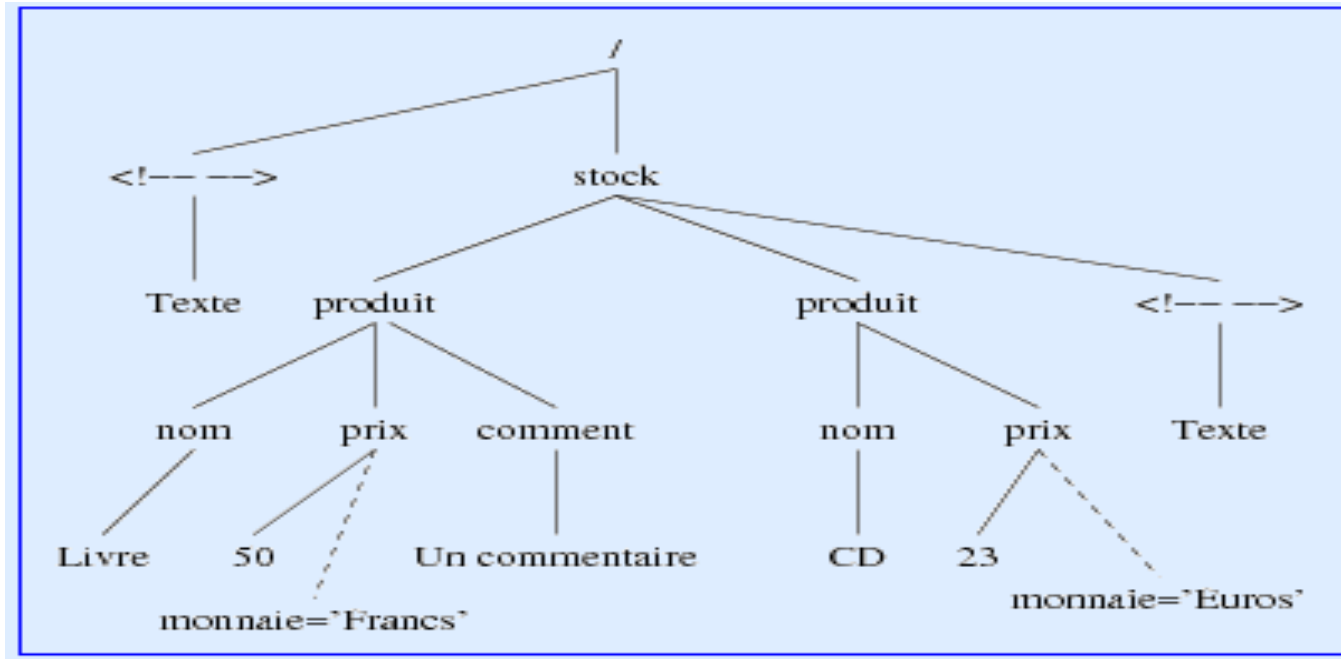
## Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Texte -->
<?play audio armide.avi?>
<stock>
  <produit>
    <nom> Livre </nom>
    <prix monnaie="Francs"> 50 </prix>
    <comment> Un commentaire </comment>
  </produit>
  <produit>
    <nom> CD </nom>
    <prix monnaie="Euros"> 23 </prix>
  </produit>
  <!-- Texte -->
</stock>
```



# XPATH

## Modèle arborescent d'un document XML vu par Xpath



# XPATH

## Les expressions

- La forme générale d'une expression XPATH est

sélecteur1/sélecteur2/...      (exp. relative)  
/sélecteur1/sélecteur2/...      (exp. absolue)

- Chaque **sélecteur** sélectionne un ensemble de nœuds en fonction du résultat du sélecteur précédent.
- L'ensemble **initial** est soit le nœud courant (forme relative) soit la racine (forme absolue).
- Exemple: /stock/produit/comment



# XPATH

## Les sélecteurs de nœuds

Une étape de positionnement est défini par un **axe** et un **test**:

1 **L'axe** sélectionne un ensemble de nœuds par rapport à leur position absolue ou relative à un autre nœud.

2 **le test** est évalué pour chaque nœud dans la sélection.

Formule: axe::test/axe::test/.../axe::test

Exemples: /child::film/descendant::acteurs/child::@nom





# XPATH

## Les axes de recherche

- Un axe de localisation représente une première approximation de ce que l'on veut.
- Un axe se base sur la notion de voisinage du nœud contexte :
  - les enfants,
  - les frères, les ascendants, etc. ..on choisit ce qui se rapproche le plus du node-set souhaité, quitte ensuite à filtrer les nœuds en trop.
- L'approximation doit toujours se faire par excès, puisqu'il est possible de filtrer, mais pas d'ajouter.

# XPATH

## Les axes: Père/Fils

- L'axe le plus utilisé est celui du fils `child::` suivi d'un test.

**`/child::*`** (le **root element**) raccourci **`/*`**

**`/child::DEBUT`** (récupère le premier élément DEBUT après la racine )

s'écrit

**`/DEBUT`**



# XPATH

Exemple: Pour la suite les éléments sélectionnés apparaîtront en rouge

XPath : **/ROOT/AA**

**<ROOT>**

**<AA>**

**<BB/>**

**</AA>**

**<AA>**

**<BB/>**

**<CC/>**

**</AA>**

**</ROOT>**



# XPATH

- L'axe parent s'écrit **parent::**. Raccourci « .. »

XPath : `/ROOT/*/BB/..`

`<ROOT>`

`<AA>`

`<BB/>`

`</AA>`

`<EE>`

`<BB/>`

`<CC/>`

`</EE>`

`</ROOT>`



# XPATH

- Se sélectionner soi-même s'écrit `self::node()` mais on lui préfère le raccourci « . »

XPath : `/ROOT/AA/.`

`<ROOT>`

`<AA>`

`<BB/>`

`</AA>`

`<EE>`

`<BB/>`

`<CC/>`

`</EE>`

`<ROOT>`



# XPATH

- L'axe descendant sélectionne tous les nœuds spécifiés contenus dans le nœud original. Il s'écrit **descendant::**

XPath : `/ROOT/ancestor::*`

`<ROOT>`

`<AA>`

`<BB>`

`<CC/>`

`</BB>`

`<DD/>`

`</AA>`

`<AA>`

`<BB/>`

`</AA>`

`</ROOT>`



# XPATH

- Il existe une variante permettant de sélectionner en plus le noeud à l'origine **descendant-or-self::**, le raccourci pour cela est **//**

XPath : `/ROOT/descendant-or-self::*` ou `/ROOT//*`

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<BB/>

</AA>

</ROOT>



# XPATH

- L'axe des ancêtres fonctionne dans le sens inverse. Il s'écrit **ancestor::** comme pour descendant, il existe une version **ancestor-or-self::**

XPath : `/ROOT/*/*/ ancestor::*`

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<BB/>

</AA>

</ROOT>





# XPATH

- Par l'axe **preceding::**, on sélectionnera tous les noeuds précédant - hors ancêtres - le noeud en lecture.

XPath : **//EE/preceding ::\***

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<EE/>

</AA>

</ROOT>



# XPATH

- Par l'axe **following::**, on sélectionnera tous les noeuds suivant - hors descendants et ancêtres - le noeud en lecture.

XPath : **//BB/following::\***

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<EE/>

</AA>

</ROOT>



# XPATH

Il existe aussi la notion de « frères » en Xpath:

- les « frères » précédents par l'axe **preceding-sibling::**
- les « frères » suivants par l'axe **following-sibling::**

XPath : `//BB/following-sibling ::*`

<ROOT>

<AA>

<BB>

<CC/>

</BB>

<DD/>

</AA>

<AA>

<EE/>

</AA>

</ROOT>



# XPATH

- Pour sélectionner des attributs par XPath, on utilisera l'axe **attribute::\*** dont le raccourci est **@\***.

XPath : **//@\***

**<ROOT>**

**<AA>**

**<BB test1='1' >**

**<CC/>**

**</BB>**

**<DD/>**

**</AA>**

**<AA test2='1' >**

**< BB/>**

**</AA>**

**</ROOT>**



# XPATH

- XPath permet de faire l'union entre deux XPath par l'opérateur « | ».

XPath : `//DD|//EE`

`<ROOT>`

`<AA >`

`<BB>`

`<CC/>`

`</BB>`

`<DD/>`

`</AA>`

`<AA>`

`<EE/>`

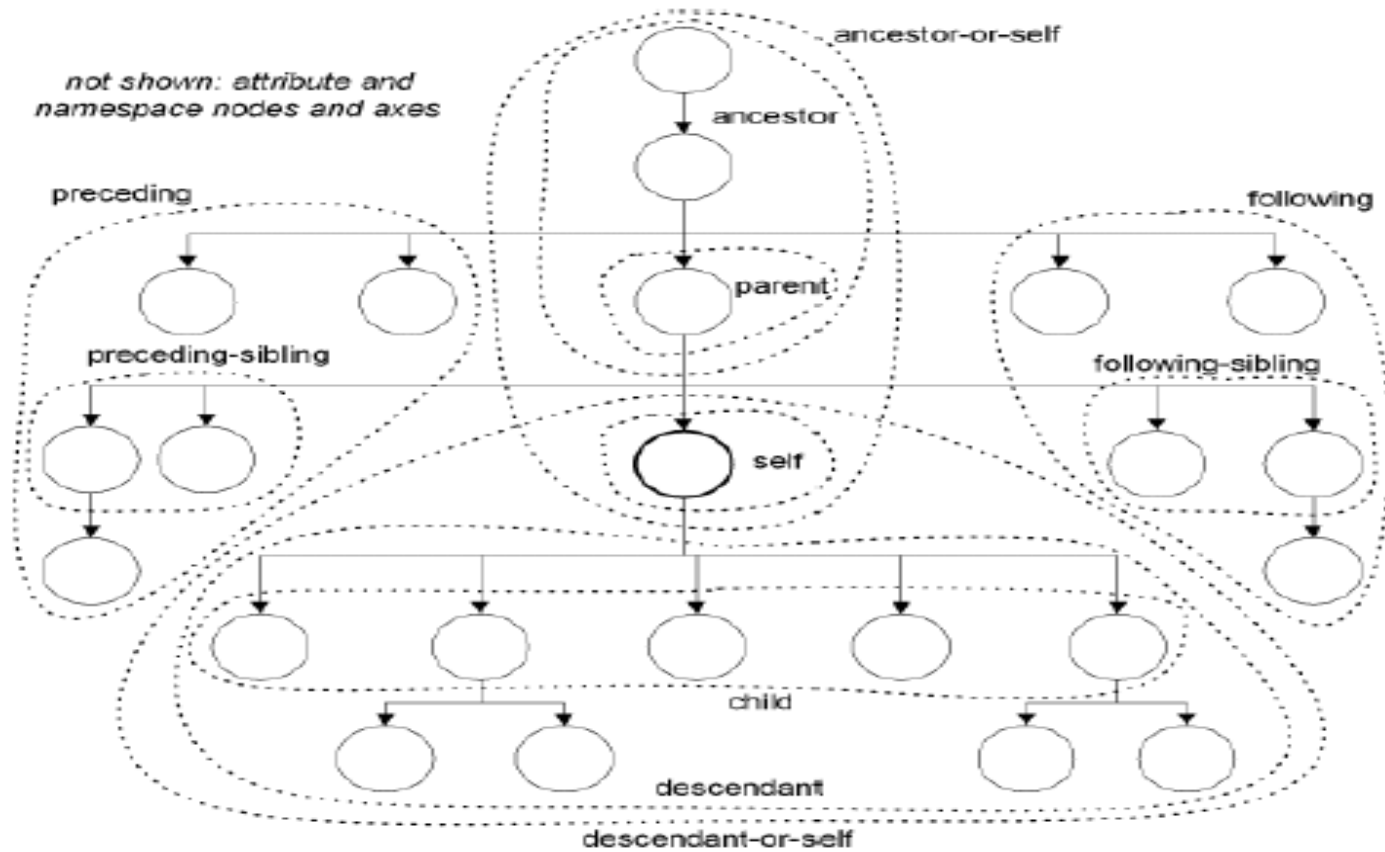
`</AA>`

`</ROOT>`



XP

*not shown: attribute and  
namespace nodes and axes*



# XPATH

## Les tests(prédicats)

- Un prédicat commence par [et se termine par].
- Un prédicat peut contenir des XPath et ainsi d'autres prédicats.
- Un prédicat est une condition, on peut le comparer à la clause **WHERE** en SQL.



# XPATH

## Les tests: Opérateurs

### Les opérateurs booléens :

Les booléens sont `true()` et `false()`. Le noeud vide, la chaîne vide et zéro sont convertis en `false()`.

- NON : c'est une fonction en XPath, **not(...)**, elle englobe la partie sur laquelle porte la négation ;
- OU : **or** ;
- ET : **and** ;
- EGAL et DIFFERENT : `=` et `!=` attention `!=` n'est pas la négation de `=`, comme cela sera détaillé plus tard.
- COMPARETEURS D'ORDRE : `<=`, `<`, `>=`, `>`.





# XPATH

## Les tests: Opérateurs

### Les opérateurs numériques:

Si une de ces opérations est effectuée sur une chaîne de caractères la valeur renvoyée est **NaN**(Not a Number).

- **ADDITION** : + ;
- **SOUSTRACTION** : -, attention sur l'opérateur de soustraction, il faut toujours le faire précéder et suivre d'un espace sinon l'expression peut être confondue avec un nom d'élément ;
- **MULTIPLICATION** : \* ;
- **DIVISION**: div ;
- **MODULO**: mod.



# XPATH

## Exemple sur les tests:

- Lors de test entraînant une comparaison tout nœud est converti en sa valeur textuelle.
- Celle-ci pourra être considérée comme un nombre (si sa forme le permet) ou une chaîne de caractères.
- Pour la suite les éléments sélectionnés apparaîtront en rouge.



# XPATH

## Test de valeur

- Il porte sur la valeur textuelle du noeud sélectionné. On ne peut comparer que des types simples.
- On notera ici l'importance du self::node() et de son raccourci «.».

Xpath : `//AA[.=1]`

`<ROOT>`

`<AA>1</AA>`

`<AA>2</AA>`

`</ROOT>`

Xpath : `//AA[. > 1]`

`<ROOT>`

`<AA>1</AA>`

`<AA>2</AA>`

`</ROOT>`



# XPATH

## Test de position

- La fonction position() renvoie la position du noeud en lecture dans son contexte parent.
- La numérotation des positions en XPath commence à 1.

```
<ROOT>
  <AA>
    <BB>
      <CC/>
    </BB>
  <DD/>
</AA>
<AA>
  <BB/>
</AA>
</ROOT>
```

Xpath : `/ROOT/AA[position()=2]`

ou `/ROOT/AA[2]` ( écriture raccourcie) sélectionne le deuxième fils AA de ROOT



# XPATH

## Test de position 2

- Attention un XPath du type `//*[2]` ne renvoie pas le deuxième noeud de la sélection mais tous les noeuds sélectionnés précédemment qui sont des deuxièmes fils.

```
<ROOT>
  <AA>
    <BB>
      <CC/>
    </BB>
    <DD/>
  </AA>
  <AA>
    <BB/>
  </AA>
</ROOT>
```

Xpath : `//*[2]`



# XPATH

## Exemple de test avec des fonctions

**last()** : Récupérer le dernier fils d'un nœud :

```
<ROOT>
  <AA>
    <BB/>
  </AA>
  <AA>
    <BB/>
    <BB/>
    <BB/>
  </AA>
  <AA>
    <BB/>
    <BB/>
  </AA>
</ROOT>
```

Xpath : `/ROOT/AA/BB[position()=last()]`



TP : LAB\_XPATH



# XSLT

## Présentation

- XSL est synonyme de e**X**tensible **S**tylesheet **L**anguage, et c' est un langage de feuille de style pour les documents XML.
- XSLT est synonyme de transformations XSL.
- XSLT est utilisé pour transformer un document XML dans un autre document XML, ou un autre type de document qui est reconnu par un navigateur, comme le HTML et XHTML.

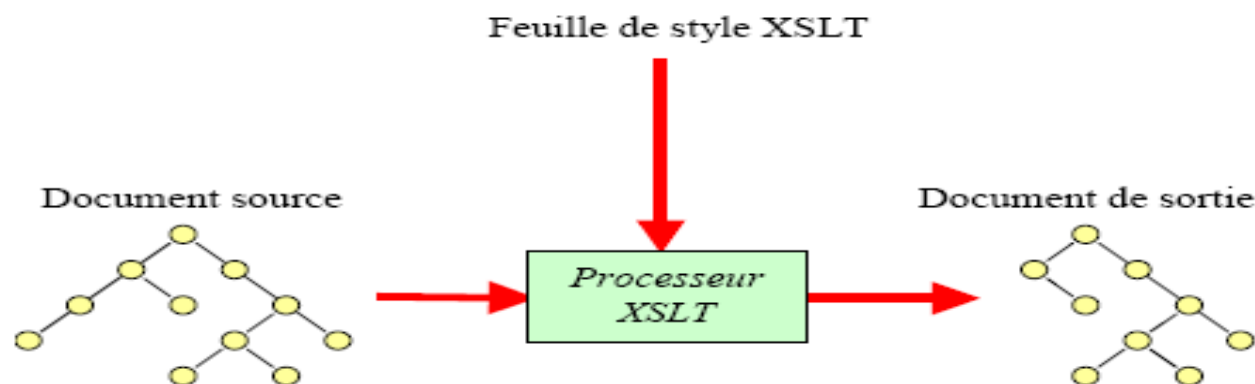




# XSLT

## Présentation

XSLT = Transformation d'arbre



# XSLT

## Présentation

- Avec XSLT, vous pouvez ajouter / supprimer des éléments et des attributs vers un fichier de sortie.
- Vous pouvez également :
  - réorganiser
  - Effectuer des tri
  - prendre des décisions sur les éléments à masquer et d'afficher,
  - et beaucoup plus...



# XSLT

## Présentation

- XSLT utilise XPath pour trouver des informations dans un document XML.
- XPath est utilisé pour naviguer à travers les éléments et attributs dans les documents XML.



# XSLT

## Présentation

Comment ça marche?

- XSLT utilise XPath pour définir les parties du document source qui correspondent à un ou plusieurs modèles prédéfinis.
- Lorsqu'une correspondance est trouvée, XSLT va transformer la partie correspondante du document source dans le document résultat.

XSLT est une recommandation du W3C



# XSLT

## XSLT Navigateurs

- Les principaux navigateurs sont compatibles avec XML et XSLT.
  - Mozilla Firefox (version 3)
  - Internet Explorer (version 6)
  - Google Chrome (version 1)
  - Opéra (version 9)
  - Apple Safari (version 3)



# XSLT

## XSLT – Transformation

- L'élément racine du document qui déclare être une feuille de style XSL est **<xsl:stylesheet>** ou **<xsl:transform>**.
- Exemple:

```
<xsl:stylesheet version="1.0"  
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Ou:

```
<xsl:transform version="1.0"  
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```



# XSLT

## L'élément `<xsl:stylesheet>`

- Élément racine d'un document XSLT

```
<xsl:stylesheet  
  version="1.0"  
  xmlns:xsl=  
    "http://www.w3.org/1999/XSL/Transform"  
>
```

- Attribut `version` : version de langage XSL (obligatoire)
- Attribut `xmlns:xsl` : espace de nom XSL



# XSLT

## L'élément `<xsl:output>`

- Format de sortie du document résultat

```
<xsl:output method="xml" version="1.0"  
            encoding="UTF 8" indent="yes"/>
```

- Attribut `method` : type du document en sortie
- Attribut `encoding` : codage du document
- Attribut `indent` : indentation en sortie





# XSLT

L'élément `<xsl:output>`

## Type de document en sortie

- Trois types de document en sortie
  - **xml** : vérifie que la sortie est bien formée
    - *(sortie par défaut)*
  - **html** : accepte les balises manquantes, génère les entités HTML (&acute; ...)
    - *(sortie par défaut si XSL reconnaît l'arbre de sortie HTML4)*
  - **text** : tout autre format textuel :
    - du code Java, format Microsoft RTF, LaTeX



# XSLT

## XSLT – Transformation

**Exemple:** Nous voulons transformer le document XML suivant ("cdcatalog.xml") en XHTML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
</catalog>
```

# XSLT

## Créer une feuille de style XSL

- Ensuite, vous créez une feuille de style XSL ("cdcatalog.xml") avec un modèle de transformation:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd">
      <tr>
        <td><xsl:value-of select="title"/></td>
        <td><xsl:value-of select="artist"/></td>
      </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

# XSLT

## Lier la feuille de style XSL au document XML

- Ajouter la référence feuille de style XSL à votre document XML ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
</catalog>
```

# XSLT

## L'élément `<xsl:template>`

- Une feuille de style XSL est composée d'une ou plusieurs ensemble de règles qui sont appelées modèles.
- L'élément `<xsl:template>` est utilisé pour construire des modèles.
- La valeur de l'attribut **match** est une expression XPath
- Exemple: **match** = `" / "` définit l'ensemble du document.



# XSLT

## L'élément `<xsl:value-of>`

- L'élément `<xsl:value-of>` est utilisé pour extraire la valeur d'un nœud sélectionné.



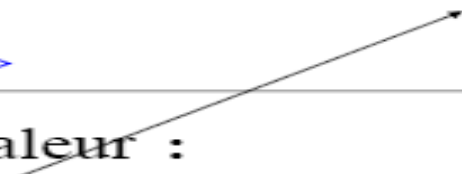
# XSLT

## L'élément `<xsl:value-of>`

### Élément `<xsl:value-of>`

- Générer le contenu d'un élément

```
<xsl:template match="carteDeVisite">  
  <p>Nom : <xsl:value-of select="nom"/>  
  </p>  
</xsl:template>
```



- Sélection de la valeur :
  - attribut `select` : expression xpath
  - ici : le texte contenu dans l'élément `nom` de l'élément `carteDeVisite`



# XSLT

## L'élément `<xsl:value-of>`

### Résultat de `<xsl:value-of>` et type nœud

- Le nœud sélectionné est un *élément*
  - Concaténation de tous les textes qui se trouvent comme contenu de cet élément et de ses descendants
- Le nœud est un nœud *text*
  - Texte du nœud lui même
- Le nœud est un *Attribut*
  - Valeur de l'attribut normalisée (pas d'espace de début et fin)
- Le nœud est une *Instruction de traitement*
  - Valeur de l'instruction de traitement (sans les marques `<?` et `?>` et sans le nom)
- Le nœud est un *Commentaire*
  - Le texte du commentaire (sans les marques `<!--` et `-->`)





# XSLT

## L'élément `<xsl:value-of>`

### Exemple 2

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Règle

```
<xsl:template match="note">  
  <xsl:value-of select="."/>  
</xsl:template>
```

- En sortie

```
enseigne XML au SEP
```



# XSLT

## L'élément `<xsl:value-of>`

### Exemple 3

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Règle

```
<xsl:template match="note">  
  <xsl:value-of select="text()"/>  
</xsl:template>
```

- En sortie

```
enseigne
```

Seul le premier élément sélectionné est produit



# XSLT

## L'élément `<xsl:for-each>`

- Itération sur une ensemble de nœuds

```
<xsl:template match="/carnetDAdresse">  
  <xsl:for-each select="carteDeVisite">  
    <p><xsl:value-of select="nom"/></p>  
  </xsl:for-each>  
</xsl:template>
```



# XSLT

## XSLT `<xsl:sort>` élément

- Permet de trier l'ensemble des nœuds sélectionnés par les instructions avant de les traiter.
- Tri sur plusieurs critères (possible)



# XSLT

## L'élément `<xsl:if>`

- Conditionnelle

```
<xsl:for-each select="carteDeVisite">  
  <xsl:value-of select="nom"/>  
  <xsl:if test="position() != last()">,  
  </xsl:if>  
</xsl:for-each>
```

- Génère une virgule après chaque nom sauf pour le dernier



TP : LAB\_XSLT



# LES PARSEURS XML



# Les parseurs XML

La récupération des données encapsulées dans le document nécessite un outil appelé *analyseur syntaxique* (en anglais *parser*), permettant de parcourir le document et d'en extraire les informations qu'il contient.

- Un parser est un outil logiciel permettant de parcourir un document et d'en extraire des informations.





# Les parseurs XML

- Les analyseurs XML sont également divisés selon l'approche qu'ils utilisent pour traiter le document.
- On distingue actuellement deux types d'approches :
  - Les API utilisant une approche hiérarchique : La principale API utilisant cette approche est DOM (*Document Object Model*)
  - Les API basés sur un mode événementiel permettent de réagir à des événements (comme le début d'un élément, la fin d'un élément) : SAX (*Simple API for XML*) est la principale interface utilisant l'aspect événementiel
- Ainsi, on tend à associer l'approche hiérarchique avec DOM et l'approche événementielle avec SAX.

# DOM (Document Object Model)

- DOM (*Document Object Model*) est une spécification du W3C définissant la structure d'un document sous forme d'une hiérarchie d'objets, afin de simplifier l'accès aux éléments constitutifs du document.
- Plus exactement DOM est un langage normalisé d'interface (API), indépendant de toute plateforme et de tout langage, permettant à une application de parcourir la structure du document et d'agir dynamiquement sur celui-ci.
- DOM n'est qu'une spécification qui, pour être utilisée, doit être implémentée par un éditeur tiers.
- DOM n'est donc pas spécifique à Java.



# DOM (Document Object Model)

- L' API DOM ressemble très étroitement à la structure des documents qu'elle modélise.

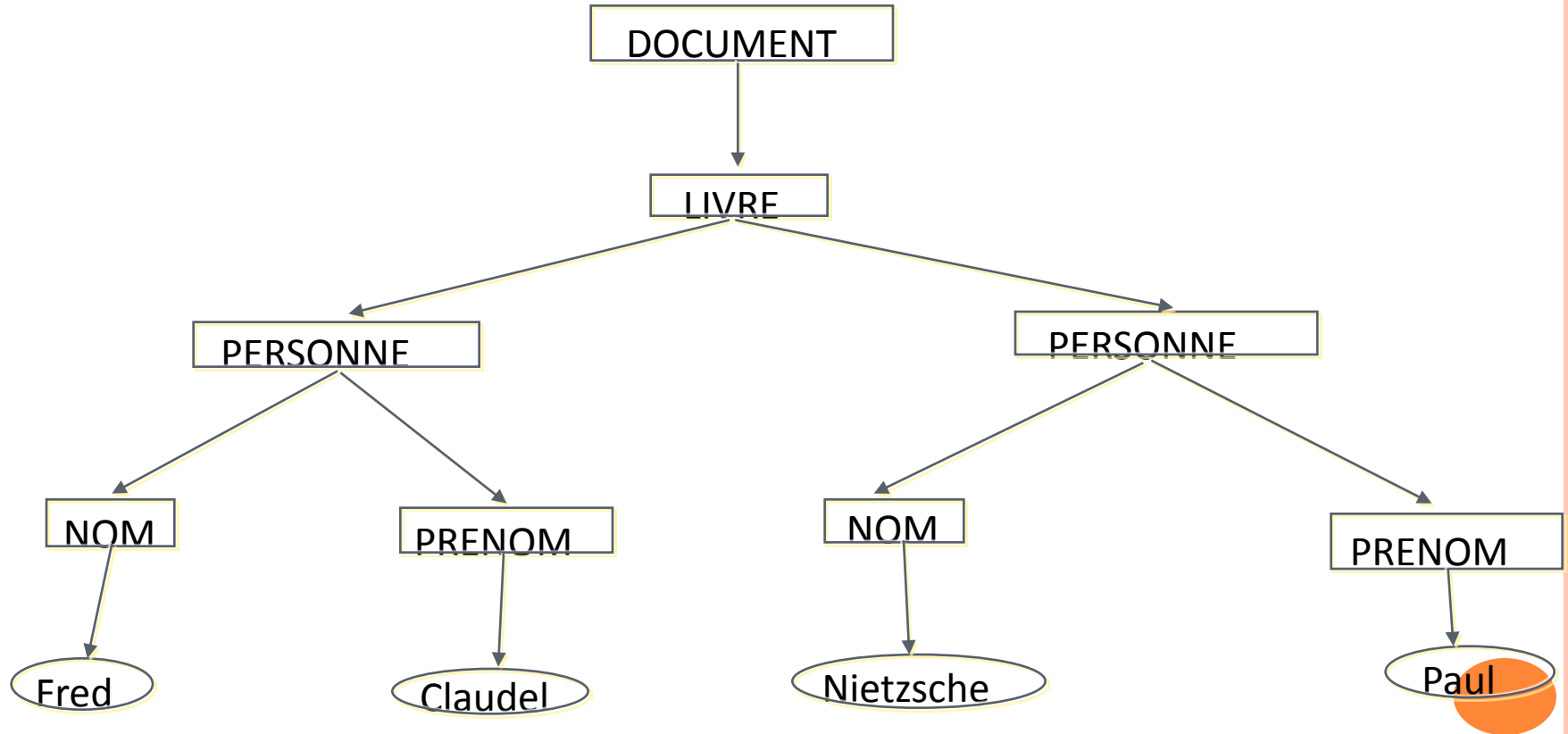
Par exemple, si l'on considère le document XML suivant:

```
<?xml version="1.0" encoding="iso-8859-1?">
<liste>
  <personne>
    <nom>Fred</nom>
    <prenom>Nietzsche</prenom>
  </personne>
  <personne>
    <nom>Paul</nom>
    <prenom>Caudel</prenom>
  </personne>
</liste>
```



# DOM (Document Object Model)

- DOM le représente ainsi:



# DOM (Document Object Model)

## Les principaux types de nœuds

- **Document** : cet objet représente l'ensemble du document.
- **Element** : l'élément est le noeud le plus employé et caractérise la balise.
- **Attr** : il s'agit d'un attribut d'élément avec un nom et une valeur.
- **ProcessingInstruction** : il s'agit d'une instruction de traitement.
- **Text** : c'est un noeud texte.
- **Entity** : il s'agit d'une entité issue d'une DTD.
- **Comment** : il s'agit d'un commentaire
- **DocumentFragment** : un fragment de document est une sorte de mini-document.



# DOM (Document Object Model)

## L'interface Node

- Tous les noeuds ont des primitives communes par héritage de l'interface Node.
- Pour distinguer néanmoins la nature du noeud manipulé, on dispose de la méthode `getNodeTypes` qui retourne une valeur entière associée aux constantes suivantes :
  - `Node.DOCUMENT_NODE`
  - `Node.ELEMENT_NODE`
  - `Node.TEXT_NODE`
  - `Node.ATTRIBUTE_NODE`



# DOM (Document Object Model)

## L'interface Node

- Les trois méthodes courantes vont donner un résultat différent en fonction du nœud :
  - **getNodeName** : le nom du nœud.
    - Élément = nom de la balise
    - Attribut = nom de l'attribut
  - **getNodeValue**:
    - Attribut = valeur de l'attribut.
  - **getAttributes** : seul l'élément est concerné.
    - L'ensemble des nœuds attribut sera disponible via un objet de type NamedNodeMap.



# DOM (Document Object Model)

## Exemple:

### Récupérer un élément root:

```
Element rootElement = document.getDocumentElement();  
String rootElementName = rootElement.tagName();
```

### Récupérer les attributs de root:

```
if (rootElement.hasAttributes()) {  
    NamedNodeMap attributes = rootElement.getAttributes();  
}
```

### Récupérer les valeurs des attributs :

```
for (int i = 0; i < attributes.getLength(); i++) {  
    Attr attribute = (Attr) (attributes.item(i));  
    System.out.println("Attribute:" + attribute.getName()+ " with value " + attribute.getValue());  
}
```





# DOM (Document Object Model)

## Exemple:

**Récupérer les fils de root:**

```
if (rootElement.hasChildNodes()) {  
    NodeList nodeList = rootElement.getChildNodes();  
}
```

**Récupérer un fils de root:**

```
for (int i = 0; i < nodeList.getLength(); i++) {  
    Node node = nodeList.item(i);  
}
```



# DOM (Document Object Model)

## LA CRÉATION D' UN PARSEUR POUR DOM

```
package com;

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

public class Dom {
    public static void main(String[] args) throws Throwable {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        // Partie DOM
        Document d = db.parse( "carnet.xml" );
        Element root = d.getDocumentElement();
    }
}
```



TP : LAB\_DOM

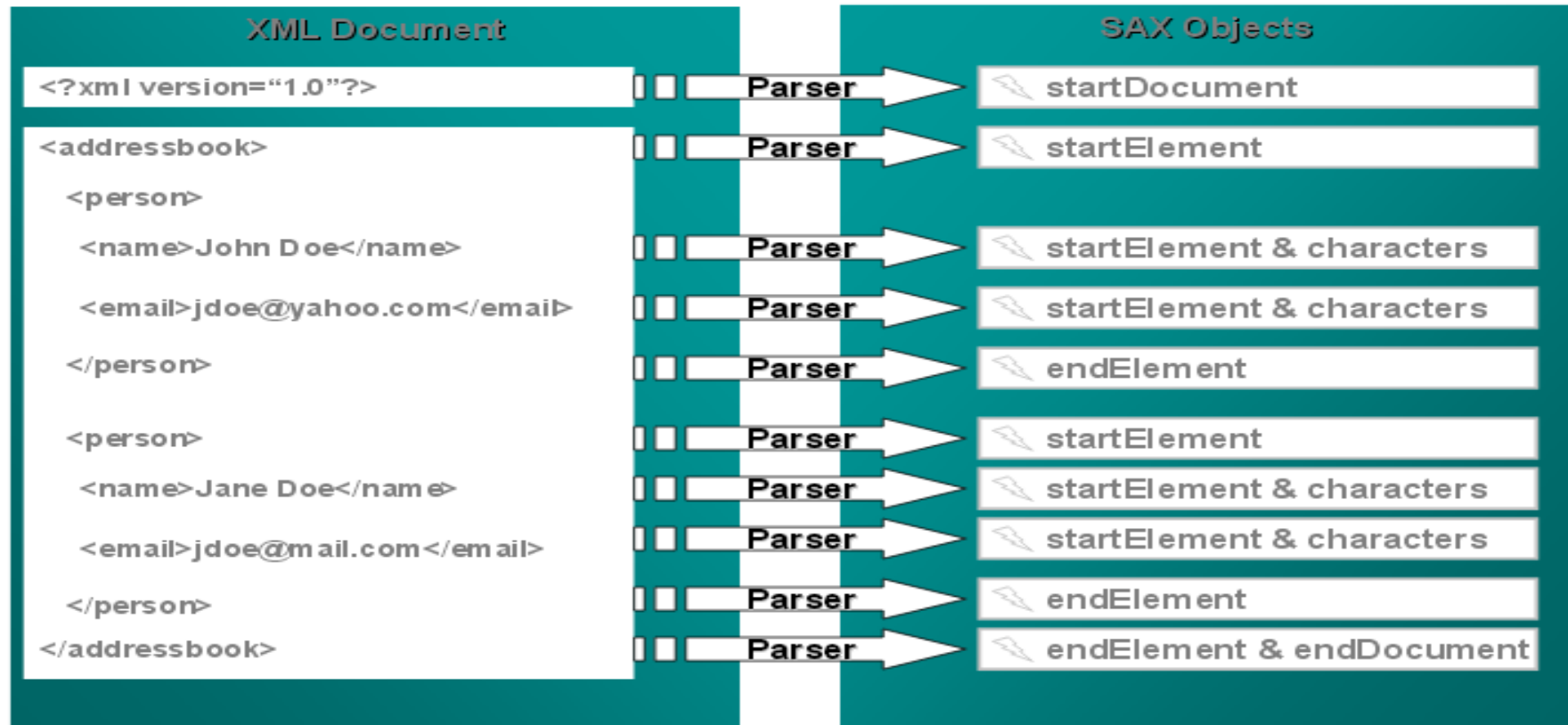


# SAX SIMPLE API FOR XML

- 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:
- ...

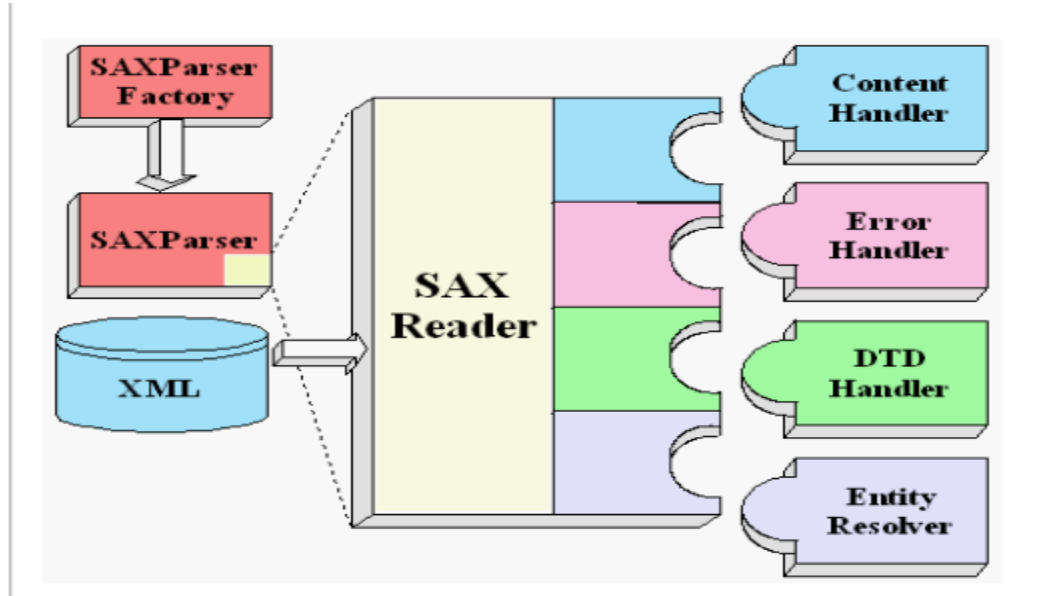


# SAX Simple Api for Xml



# SAX Simple Api for Xml

## IMPLÉMENTER LES HANDLERS D'ÉVÈNEMENTS DU PARSEUR



# SAX Simple Api for Xml

---

```
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);
        }
    }
}
```



# SAX Simple Api for Xml

- Toutes les applications SAX doivent implémenter 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
  - ...





# SAX Simple Api for Xml

RÉCUPÉRER LE DÉBUT D'ANALYSE DE CHAQUE ÉLÉMENT

...

```
import org.xml.sax.Attributes;
```

```
public class SurveyReader extends DefaultHandler {
```

```
    String thisElement = "", thisQuestion = "";
```

```
    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);
```

```
        thisElement = qName;
```

```
        if(qname.equals("question")){ thisQuestion = atts.getValue("subject"); }
```

```
    } ... }
```



# SAX Simple Api for Xml

## 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: question
Start element: response
Start element: question
Start element: question
Start element: question
Start element: question
Start element: question
Start element: question
Start element: question
Start element: question
Start element: question
```



# SAX Simple Api for Xml

## RÉCUPÉRER LES DONNÉES

```
public void characters(char[] ch, int start, int length)
throws SAXException {
    if (thisElement.equals("question")) {
        System.out.print(thisQuestion + ": ");
        System.out.println(new String(ch, start, length));
    }
}
...

```



# SAX Simple Api for Xml

## EXAMPLE

```
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
```



**JAX-B**



# JAX-B

- OBJECTIF
- VERSION
- MISE EN OEUVRE
- UTILISATION
  - Génération des classes à partir d'un schéma
  - Le mapping d'un document XML à des objets (unmarshal)
  - La création d'un document XML à partir d'objets (marshal)
  - La génération d'un schéma à partir de classes compilées



## OBJECTIF

- JAXB est une spécification qui permet de faire correspondre un document XML à un ensemble de classes et vice et versa (marshaling/unmarshaling).
- JAXB permet aux développeurs :
  - de manipuler un document XML sans à avoir connaître XML
  - sans à avoir connaître la façon dont un document XML est traitée comme cela est le cas avec SAX ou DOM.
- La manipulation du document XML se fait en utilisant des objets précédemment générés à partir d'un schéma XML du document à traiter.



## OBJECTIF

- En plus de son utilité principale, JAXB 2.0 propose d'atteindre plusieurs objectifs :
  - Être facile à utiliser pour consulter et modifier un document XML sans connaissance ni de XML ni de techniques de traitement de documents XML
  - Être configurable : JAXB met en oeuvre des fonctionnalités par défaut qu'il est possible de modifier par configuration pour répondre à ces propres besoins
  - S'assurer que la création d'un document XML à partir d'objets et retransformer ce document en objets donne le même ensemble d'objets





## OBJECTIF

- Pouvoir valider un document XML ou les objets qui encapsulent un document sans avoir à écrire le document correspondant
- Être portable : chaque implémentation doit au minimum mettre en oeuvre les spécifications de JAXB



## VERSION

- Pour ce projet nous avons utilisé JAXB 2.1 qui est incorporée à Java EE 5 et à Java SE 6.
- Les fonctionnalités de JAXB 2.0 par rapport à JAXB 1.0 sont :
  - support uniquement des schémas XML (les DTD ne sont plus supportées)
  - mise en oeuvre des annotations
  - assure la correspondance bidirectionnelle entre un schéma XML et le bean correspondant.



## VERSION

l'utilisation de fonctionnalités proposées par Java 5 notamment les generics et les énumérations

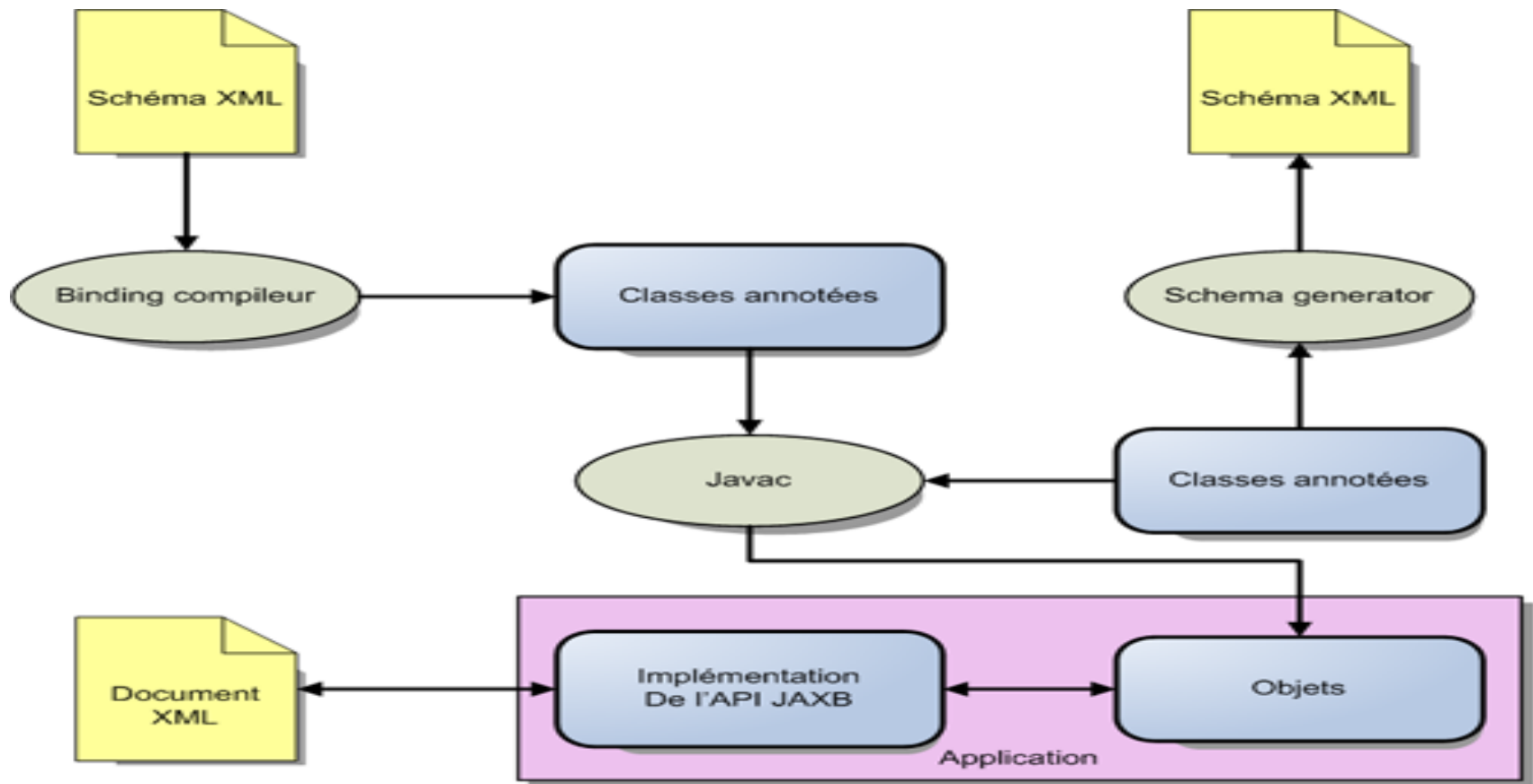
- le nombre d'entités générées est moins important : JAXB 2.0 génère une classe pour chaque complexType du schéma alors que JAXB 1.0 génère une interface et une classe qui implémente cette interface.

- Une méthode de la classe ObjectFactory est générée pour renvoyer une instance de cette classe.

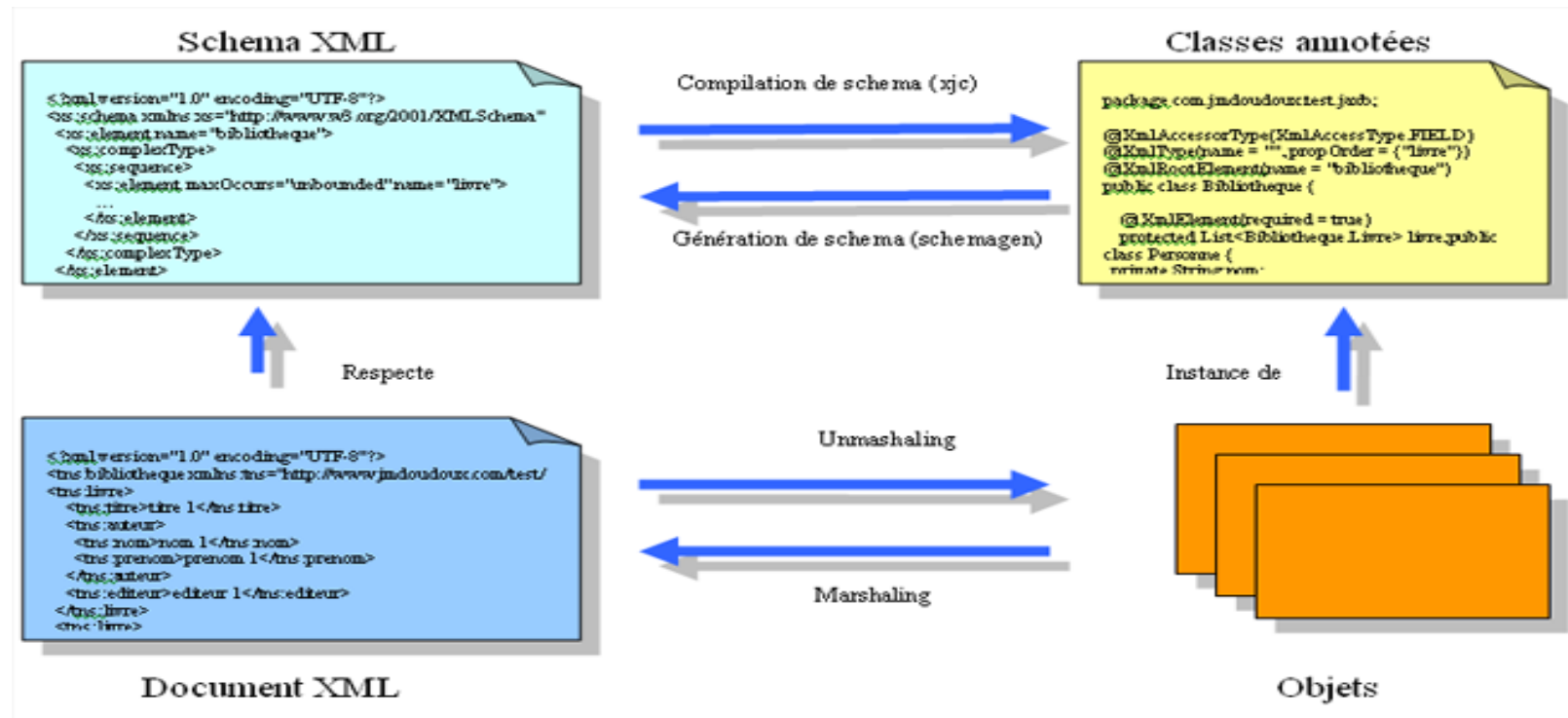


# JAX-B

## MISE EN OEUVRE



# MISE EN OEUVRE

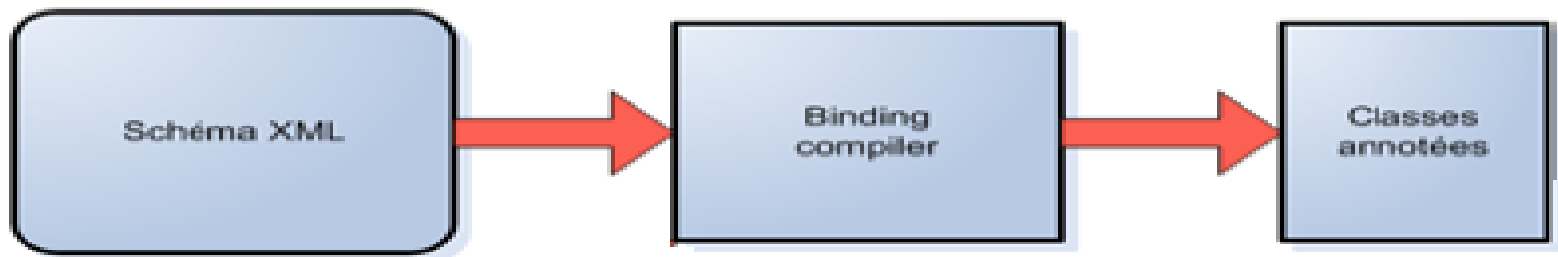


# UTILISATION

- Génération des classes à partir d'un schéma
- Le mapping d'un document XML à des objets (unmarshal)
- La création d'un document XML à partir d'objets (marshal)
- La génération d'un schéma à partir de classes compilées



## GÉNÉRATION DES CLASSES À PARTIR D' UN SCHÉMA



L'outil xjc permet de générer les classes à partir d'un schéma XML

```
xjc -d src personne.xsd
```

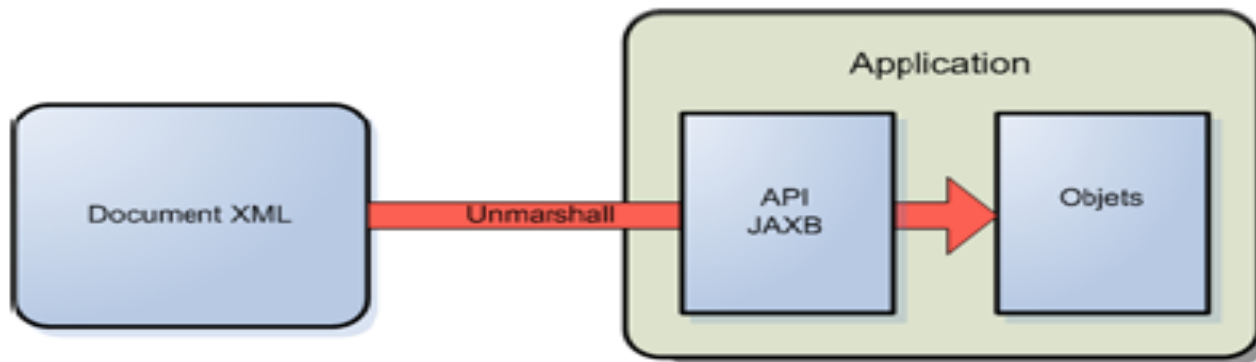
Les classes générées :

- Personnes.java : classes qui encapsulent le document XML
- ObjectFactory.java : fabrique qui permet d'instancier des objets utilisés lors du mapping



## LE MAPPING D'UN DOCUMENT XML À DES OBJETS (UNMARSHAL)

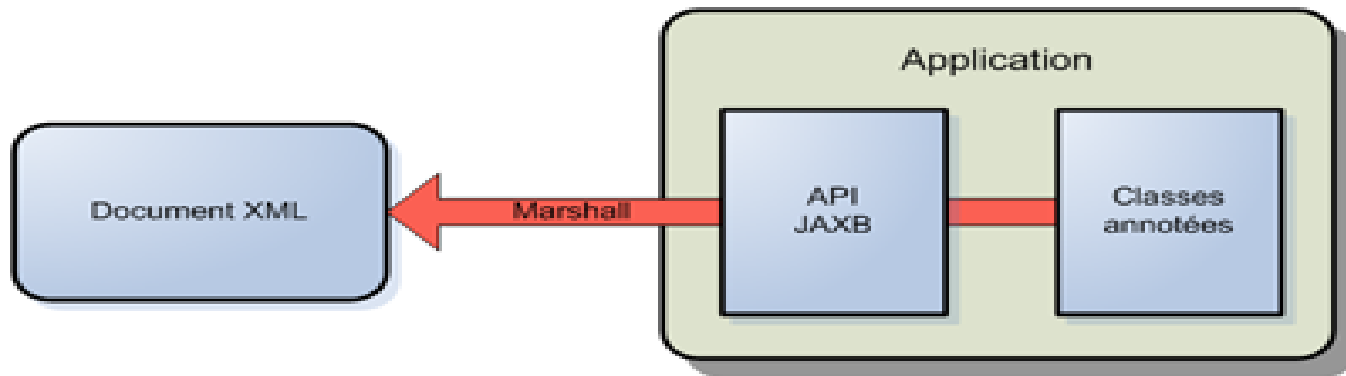
- L'API JAXB propose de transformer un document XML en un ensemble d'objets qui vont encapsuler les données et la hiérarchie du document.
- Ces objets sont des instances des classes générées à partir du schéma XML.





## LA CRÉATION D'UN DOCUMENT XML À PARTIR D'OBJETS (MARSHAL)

- JAXB permet de créer un document XML à partir d'un graphe d'objets : cette opération est nommée marshalling. Une opération de marshalling est l'opération inverse de l'opération d'unmarshalling.
- Ce graphe d'objets peut être issu d'une opération de type unmarshalling (construction à partir d'un document XML existant) ou issu d'une création de toutes pièces de l'ensemble des objets. Dans le premier cas cela correspond à une modification du document et dans le second cas à une création de document.



# JAX-B

## UNMARSHAL : INSTANCE EN XML

```
import javax.xml.bind.*;
import java.io.*;
import java.util.*;
import generated.*;

public class TestJAXB {
    public static void main(String[] args) throws Exception{
        JAXBContext jc = JAXBContext.newInstance("generated« );
        Unmarshaller unmarshaller = jc.createUnmarshaller();
        Addition expr = (Addition)unmarshaller.unmarshal(new File("add.xml »));
        for( Object o : expr.getAdditionOrEntier()){
            System.out.println("o : " + o);
        }
    }
}
```





```
public class TestJAXB {  
    public static void main(String[] args) throws Exception{  
        JAXBContext jc = JAXBContext.newInstance("generated« ");  
        Unmarshaller unmarshaller = jc.createUnmarshaller();  
        Addition expr =(Addition)unmarshaller.unmarshal(new File("add.xml »));  
        jc = JAXBContext.newInstance("generated");  
        Marshaller marshaller = jc.createMarshaller();  
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, new Boolean(true));  
        marshaller.marshal(expr,new FileOutputStream("jaxbOutput.xml"));  
    }  
}
```



TP : LAB\_JAXB

