XML

M.Lahmer

Plan

- XML (eXtended Markup Language)
- Documents de validation
  - DTD (Document Type Definition)
  - Schéma XSD (XML Shema Document)
- Langage de requête
  - XPATH
- Langage de mise en forme
  - XSLT (eXtensible StyleSheet Language Tranformation)
- Les APIs de Traitement
  - SAX et DOM

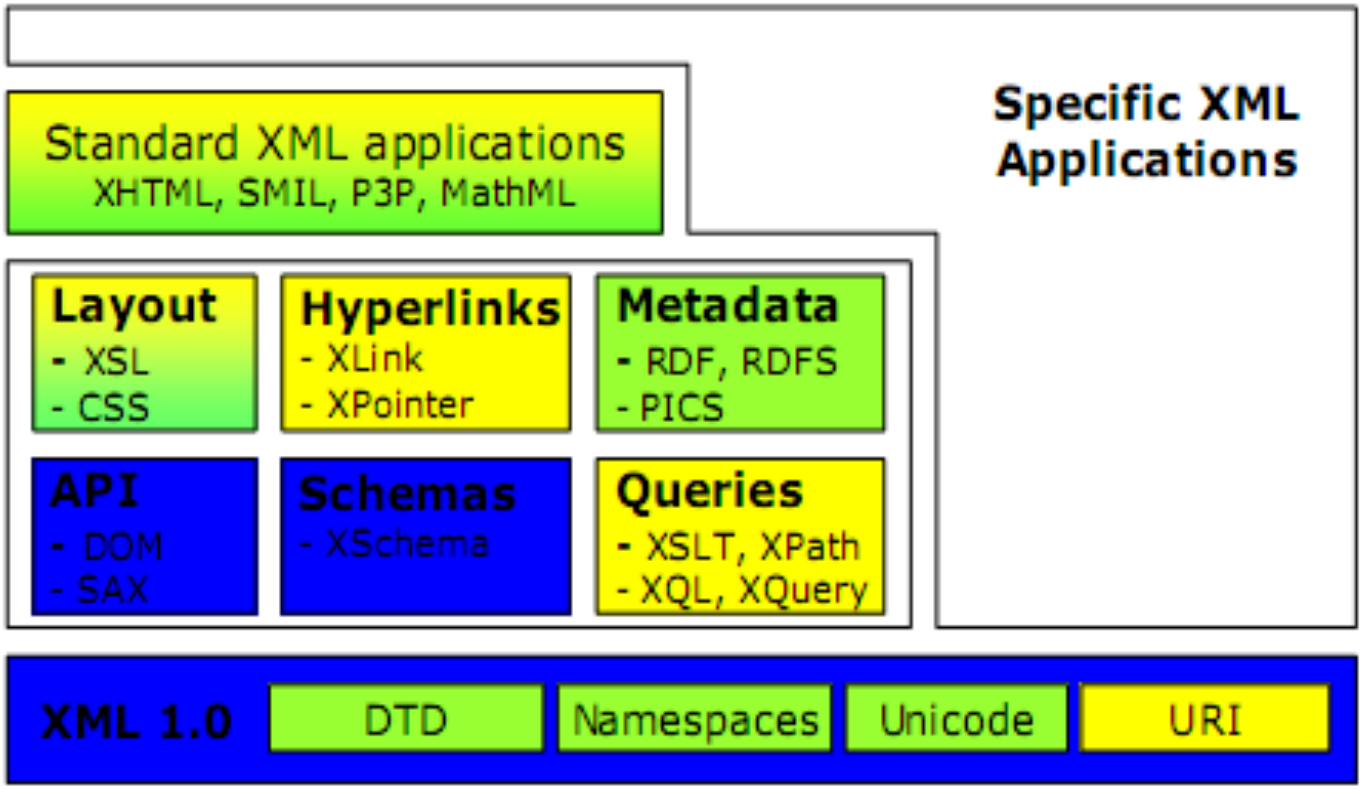
World Wide Web Consortium

- XML est normalisé par le **W3C** - Fondé en 1994
- Consortium industriel international qui regroupe différents sites
  - MIT/LCS aux Etats-Unis
  - INRIA en Europe
  - Keio University au Japon
- Plus de 448 membres industriels
- Objectifs :
  - Accroître le potentiel du Web
  - Développer des protocoles communs
  - Assurer l'inter-opérabilité sur le Web entre les différents systèmes

XML

- XML : eXtensible Markup Language
  - "XML will be the ASCII of the Web – basic, essential,"
- Un méta-langage universel pour les données sur le Web
  - Permet au développeur de délivrer du contenu depuis les applications à d'autres applications ou aux browsers
- XML permet de formaliser la manière dont l'information est :
  - Échangée (XML)
  - Personnalisée (XSLT)
  - Retrouvée (XPATH et XQuery)
  - Liée (XLink, Xpointer)
  - Sécurisée (Encryption, Signature)
  - ...

Architecture XML



RDF :Resource Description Framework

P3P: Platform for Privacy Preferences Project

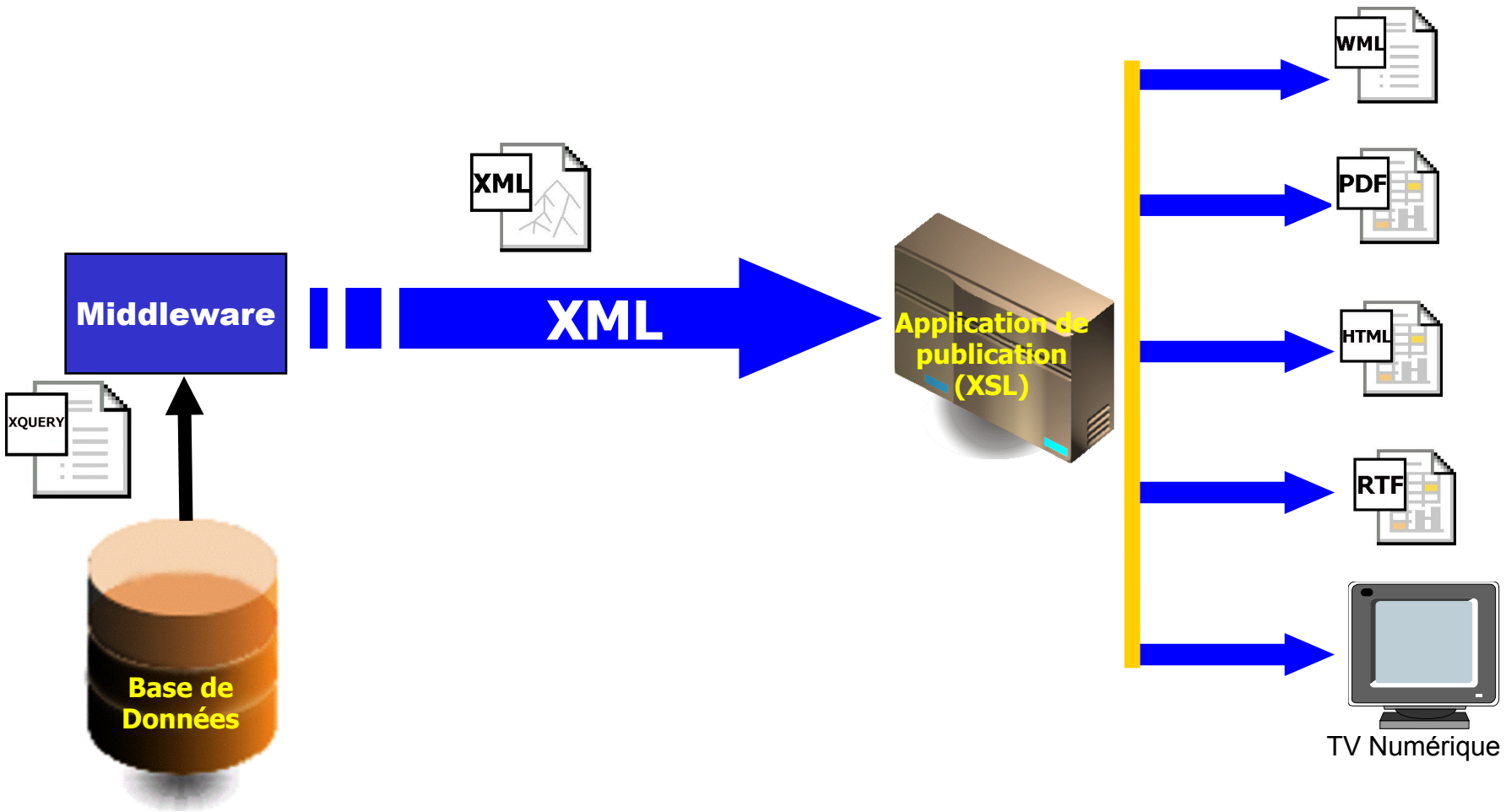
PICS : Platform for Internet Content Selection

SMIL: Synchronized Multimedia Integration Language

Pourquoi XML

- Définir vos propres langages d'échange
  - Commande, facture, bon de liv.raison, PDU (protocole) etc.
- Modéliser des données et des messages
  - Document Type Definitions (DTD)
  - Types et éléments agrégés (XML Schema)
  - Passerelle avec Unified Modelling Language (UML)
- Publier des informations
  - Sous plusieurs format
  - Mise en forme avec CSS et XSL
- Archiver des données
  - Auto-description des archives (audio, vidéo,..)

Publication multi-supports



Parser XML

- Composants logiciels permettant d'analyser | accéder aux données d'un document XML
  - APIs DOM (Document Object Model)
    - Dirigée par les éléments d'un document XML
  - APIs SAX (Simple API for XML)
    - Dirigée par les événements
- Peuvent être utilisée en:
  - Java, C++
  - C#, C
  - Python, PHP
- Il existe d'autres APIs J2EE pour XML
  - JAXP : Interface commune pour SAX, DOM, XSL
  - JAXB : Mapping des objets java en XML

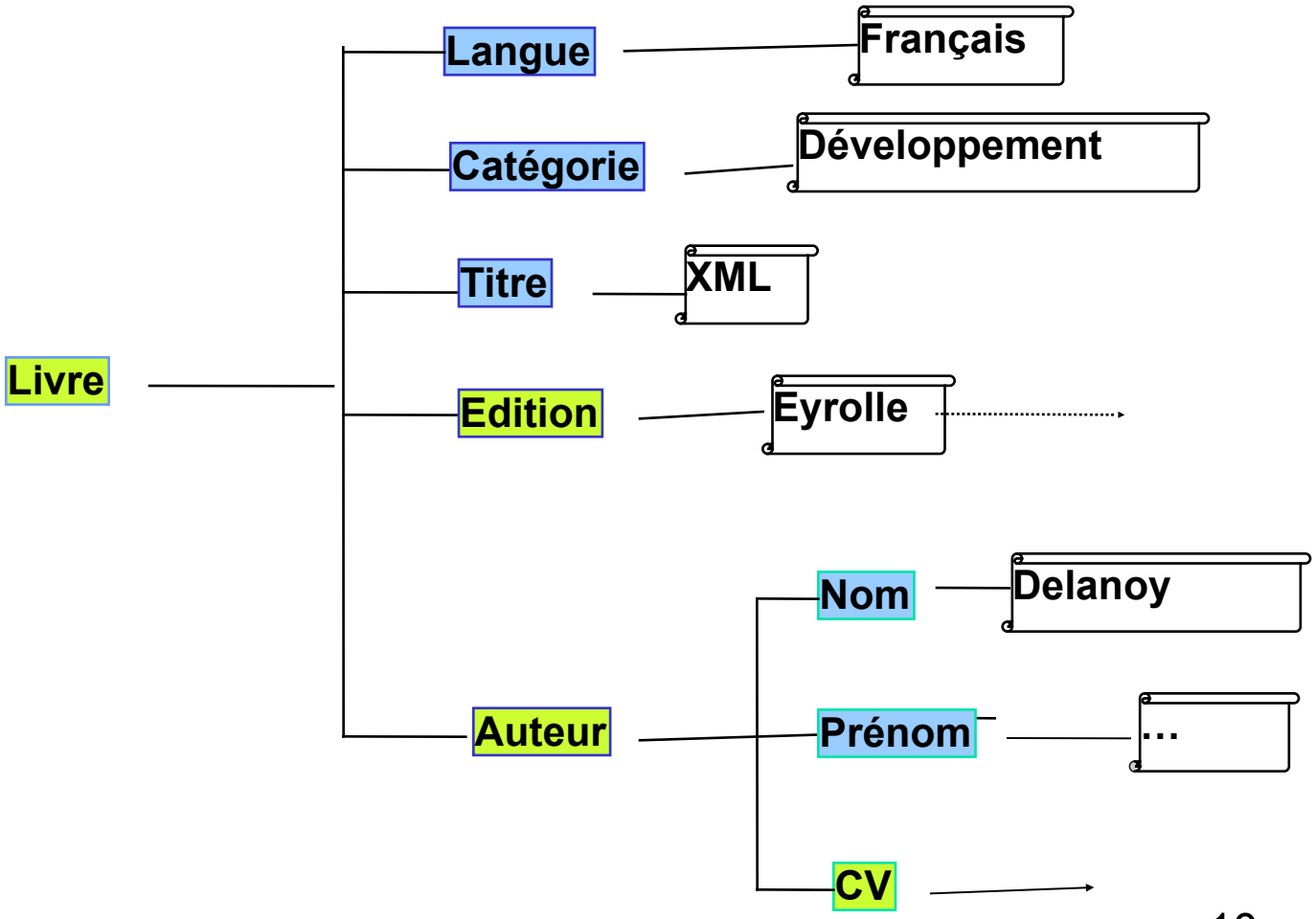
Balise ou Tag

- Marque de début et fin permettant d'identifier un élément d'un texte
- Les balises sont de la forme classique
  - **<balise>** valeur ou bloc **</balise>**
- Les balises peuvent être imbriquées
  - **<?xml version="1.0" ?> Prolog**
    - **<personne> Document root**
    - <adresse>
    - <rue> P. Lamamba </rue>
    - <ville> Rabat </ville>
    - </adresse>
    - **</personne>**

Eléments de données

- Un objet balisé s'auto-décrit

Structure d'un document XML



Prologue

- Un document XML doit toujours commencer par un prologue

```
<?xml version="1.0" encoding ="ISO-8859-1" standalone="no"?>
```

  - **La version XML est obligatoire**
  - **L'encoding identifie le type de codage caractère**
    - **UTF-8 est la valeur par défaut**
  - **standalone permet d'identifier la présence d'une DTD**
    - **no est la valeur par défaut**
- Prologue peut contenir d'autres directives
  - eXtensible Stylesheet Language XSL
  - <?xml-stylesheet type="text/css" href="ss.css"?>

Balise bien formée

- Document bien formé (Well Formed document)
  - Balises correctement imbriquées
    - **<A></A> | <A/>**
  - Sensibilité à la case <A>… n'est pas autorisé
  - Seul un élément racine (document) est autorisé
  - Parsable et manipulable
    - Nom d'un élément ne doit pas commencer par xml
    - Peut contenir des lettres, chiffres,(_,-,.,:)
    - Les caractères **<** et **&** ne peuvent pas être utilisés dans le texte
    - Utilisez **<** à la place de **<** et **&** à la place de **&**
    - Les caractères **>**, **"** , et **'** peuvent également être remplacés par **>** , **&quot**; et **'** respectivement
  - Pas nécessairement valide par rapport à la DTD

IDEs XML

- Environnements didiés
  - XMLspy de Altova
  - Alchemist XML IDE (free)
  - Oxygen de SyncRO
  - Butterfly XML (sourceforge)
  - XMLNotePade
  - CookTop
  - XMLPro
  - Xerces sous Linux
- Plugins intégérés
  - Eclipse
  - NetBeans

DTD

- Permet de définir le «vocabulaire» et la structure qui seront utilisés dans le document XML
- Grammaire du langage dont les phrases sont des documents XML (instances)
- Une DTD indique les noms des éléments et leur contenu
- Le contenu est spécifié en indiquant le nom, l'ordre et le nombre d'occurrences autorisées des sous-éléments
- La définition de type de document doit apparaître avant le premier élément du document.
  - DTD interne
- Comme elle peut être mise dans un fichier et être appelé dans le document XML
  - DTD externe

Exemple de DTD

- **<!ELEMENT tag (contenu)>**
  - Décrit une balise qui fera partie du vocabulaire.
  - Le contenu peut être composé (d'autres balises) ou texte (#PCDATA)
  - PCDATA Elément de texte contenant des caractères à parser
  - <!ELEMENT Personne (Nom, Prenom)>
  - <!ELEMENT Nom (#PCDATA)>
  - <!ELEMENT Prenom (#PCDATA)>
- **<!ATTLIST tag [attribut type #mode [valeur]]***
  - Définit la liste d'attributs pour une balise
  - ex : <!ATTLIST auteur
    Nom CDATA #REQUIRED
    Profession CDATA #IMPLIED>
    <!ATTLIST editeur
    ville CDATA #FIXED "Casa">

DTD : Contenu d'un élément

Notations	Exemples
(a, b) séquence	(nom, prenom, rue, ville)
(a|b) liste de choix	(oui|non)
a? élément optionnel [0,1]	(nom, prenom?, rue, ville)
a* élément répétitif [0,N]	(produit*, client)
a+ élément répétitif [1,N]	(produit*, vendeur+)

Types des attributs

Type	Description
CDATA	Données texte [qui ne seront pas analysées (parsées)] <![CDATA["1234&12345"]]>
ANY	Tout texte possible
EMPTY	Vide
(en1|en2|..)	liste énumérée de valeurs
ID	id unique (Type NMTOKEN) commence par une lettre
IDREF	id d'un autre élément
IDREFS	Liste d'ids d'un autre élément
NMTOKEN	valeur contenant (Lettre, chiffre,.,-,_,:)
NMTOKENS	NMTOKEN plus espace, CR ou LF
ENTITY	La valeur est une entité
NOTATION	La valeur est une notation

Exemple de DTD

```
<!ELEMENT doc (livre* | article+)>
<!ELEMENT livre (titre, auteur+)>
<!ELEMENT article (titre, auteur*)>
<!ELEMENT titre(#PCDATA)>
<!ELEMENT auteur(nom, adresse)>
<!ATTLIST auteur
   id ID #REQUIRED>
<!ELEMENT nom(prenom?, nomfamille)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT nomfamille (#PCDATA)>
<!ELEMENT adresse ANY>
```

Valeurs par défaut

Value	Explication
valeur	La valeur par défaut
#REQUIRED	L'attribut est obligatoire
#IMPLIED	L'attribut est optionnel
#FIXED valeur	La valeur de l'attribut est fixé à valeur

DTD:

```
<!ELEMENT Personne EMPTY>
<!ATTLIST Personne
      marié  (oui|non) "non" >
```

DTD:

```
<!ATTLIST personne
      id ID #REQUIRED>
```

XML: **<personne id="A111" />**

DTD:

```
<!ELEMENT Personne EMPTY>
<!ATTLIST  Personne
      ville CDATA #FIXED "Rabat" >
```

Localisation de la DTD

- DTD interne à un document XML
  - **<!DOCTYPE root []> (Juste après Prologue)**
- DTD externe : Modèle pour plusieurs documents
  - partage des balises et structures
  - Définition locale ou externe
    - DTD privée localisée dans le SF ou le serveur HTTP
    - **<!DOCTYPE root SYSTEM URL>**
    - DTD à usage public
    - **<!DOCTYPE root PUBLIC FPI URL>**
- Exemple de document

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Test SYSTEM "Test.dtd">
```

DTD public

- Le FPI(Formal Public Identifier) est composé de quatre parties
  1. La DTD est liée à un standard
     - - personnelle
     - + détenue par un organisme reconnu par l'ISO
     - ISO, DTD approuvée par un comité de standardisation
  2. Groupe responsable de la DTD
  3. Description et type de document
  4. Langage utilisé dans la DTD

Exemple :

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
 Application 2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

Cette DTD est stockée sous lib\dtds de l'installation JEE SDK

DTD : Entités

- Permet la réutilisation dans une DTD
  - Générale : référence un texte
  - Paramètre : référence un élément dans la DTD
- Entité générale
  - **INTERNAL (PARSED)**
    - <?xml version="1.0" standalone="yes" ?>
    - <!DOCTYPE Personne [**<!ENTITY ma "maroc">]>**
    - <pays>**&ma;**<pays>
  - **EXTERNAL (PARSED)** : référence commune qui peut être partagée par plusieurs documents
    - <!DOCTYPE Livre [**<!ENTITY legal SYSTEM "legal.xml">**]>
    - <Livre> **&legal;**</Livre>

ENTITE paramètre

- Est utilisée uniquement à l'intérieur d'une DTD (Jamais dans un document XML)
  - Une référence à une entité paramètre commence par %
  - **<!ENTITY % nom "entity_value">**
- Entité paramètre interne:
  - **<!ENTITY % Texte "(#PCDATA)">**
    <!ELEMENT Nom %Texte;>
  - **<!ENTITY %mode "(cheque | espece|carte) 'espece'">**
    <!ATTLIST Achat payement %mode;>
- Peuvent être externes :

  <!ENTITY %nom SYSTEM "URI">

  %nom;

DTD : ENTITY paramètre externe

- Fichier pays.ent

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY ma "Maroc">
 <!ENTITY fr "France">
<!ENTITY jp "Japon">
<!ENTITY tn "Tunisie">
<!ENTITY tr "Turkie">
```

- Fichier XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Personne SYSTEM "Personne.dtd" [
<!ENTITY % pays SYSTEM "pays.ent" >
%pays;
]>
<pays>&ma;</pays>
```

Espace de Nom

- Un document XML peut faire référence à plusieurs DTDs
- Naming space est une manière pour dire que tel tag appartient à tel DTD
  - Résoudre les conflits de nom
- Comme Java, XML utilise un nom qualifié
  - **Java: myPack.myObject.myVariable**
  - **XML: myDTD:myTag**
- Un espace de nom est défini comme un string unique
  - On utilise **URI** (Uniform Resource Indicator) pas nécessairement réelle

```
<table xmlns="http://www.est.ac.ma/h">
    <nom>Etudiants</nom>
    <Lignes>1000</Lignes>
    <colonnes>10</colonnes>
  </table>
```

<table> <tr>
<td>TOTO</td>
<td>AOAO</td>

</tr> </table>

Espace de Nom

```
<h:table xmlns:h="http://www.w3.org/TR/
    html4/">
<h:tr>
<h:td>Java</h:td>
<h:td>Linux</h:td>
</h:tr>
</h:table>
```

```
<f:table xmlns:f="http://www.est.ac.ma/f">
<f:name>Table Etudiants</f:name>
<f:width>80</f:width>
<f:length>120</f:length>
</f:table>
```

```
<root xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.est.ac.ma/f">
```

Insuffisance des DTD

- Pas de types de données
  - Pas de possibilité pour distinguer les types
  - Difficile à traduire en schéma objets
  - Pas de notion d'espace de nom
- Très faibles en contraintes aussi bien sur les éléments que sur les attributs
- Pas en XML : langage spécifique (il faut un interpréteur indépendant de celui de XML)
- Propositions de compléments
  - XML-data de Microsoft (BizTalk)
  - XML-schema du W3C (version 1.1)
  - **Relax NG** (**REgular LAnguage for XML Next generation**) **ISO/IEC 19757 - DSDL**

XML Schéma

- XSD (XML Shema Defintion) est document XML
- Permettre de typer les données
  - Eléments simples et complexes
  - Attributs simples
  - Redéfinition des types très souples
- Permettre de définir des contraintes
  - Existence, obligatoire, optionnel
  - Domaines, cardinalités, références
  - Patterns, ...
- Présente de nombreux avantages
  - extensibilité par héritage et ouverture
  - analysable par un parseur XML standard

Types de données

- Simple : type primitif (chaîne, numérique et date)
  - **<xsd:element name="immat" type="xsd:string"/>**
- Dérivé : construit en ajoutant des contraintes (de toute sorte,
  - expressions régulières permises
  - Restriction (sous-type)
  - Union (alternative entre plusieurs types)
  - Liste (ensemble de types semblables)
- Complexes (contenu d'un élément) : construit par composition d'éléments
  - <sequence> : collection ordonnée d'éléments typés
  - <all> : collection non ordonnée d'éléments typés
  - <choice>: choix entre éléments typés

Le document XSD

- L'extension du fichier est .xsd
- L'élément racine est <schema>
- Le *namespace* suivant est obligatoire (préfixe xsd ou xs)
  - **xmlns:xsd="**http://www.w3.org/2001/XMLSchema**"**
  - Source des éléments et des types de données
- Les attributs possibles sont
  - **targetNamespace="URI"**
  - La source des éléments qui seront définis dans le schéma
  - **xmlns="URI"**
  - L'espace de nom par défaut
  - **elementFormDefault="qualified"**
  - Contenu du document xml doit utilisé un espace de nom

Référencer un xsd

- Un document xml fait référence à un schémas par
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
- Dans le cas d'un schéma local (cas d'une DTD SYSTEM) on utilise l'attribut
  xsi:noNamespaceSchemaLocation
- Si non
  xsi:schemaLocation="http://www.estm.ma fichier.xsd"
- Exemple
  - <?xml version="1.0"?>
  - <rootElement
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="fichier.xsd">
    ...
    </rootElement>

XML, Schéma et DTD

```
<?xml version="1.0"?>
<mail> <to>Ahmed</to>
<from>Mohamed</from>
<subject>Rendez-vous</subject>
<body>N'oublie pas le week-end!</body>
</mail>
```

```
<!ELEMENT mail (to, from, subject, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
<xsd:element name="mail">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="to" type="xsd:string"/>
<xsd:element name="from" type="xsd:string"/> <xsd:element name="subject" type="xsd:string"/> <xsd:element name="body" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Les types simples (1)

- Chaînes de caractères
  - string
  - normalizedString
  - token
- Binaires
  - byte
  - unsignedByte
  - base64Binary
  - hexBinary

- integer
  - -126789, -1, 0, 1, 126789
- positiveInteger
  - 1, 126789
- negativeInteger
  - -126789, -1
- nonNegativeInteger
  - 0, 1, 126789
- nonPositiveInteger
  - -126789, -1, 0
- int
  - -1, 126789675
- unsignedInt
  - 0, 1267896754

Les types simples (2)

- long
  - -1, 12678967543233
- unsignedLong
  - 0, 12678967543233
- short
  - -1, 12678
- unsignedShort
  - 0, 12678
- decimal
  - -1.23, 0, 123.4, 1000.00
- float
- double

- boolean
  - true, false 1, 0
- time
  - 13:20:00.000, 13:20:00.000-05:00
- dateTime
  - 1999-05-31T13:20:00.000-05:00
- duration
  - 10H30M12.3S
- date
  - 1999-05-31
- gMonth
  - --05--
- gYear
  - 2009

Les types simples (3)

- gYearMonth
  - 1999-02
- gDay
  - ---31
- gMonthDay
  - --05-31
- Name
  - shipTo
- QName
  - po:Address
- NCName
  - Address
- anyURI
- language
  - en-GB, en-US, fr
- ID
  - "A212"
- IDREF
  - "A212"
- IDREFS
  - "A212 B213"
- ENTITY
- ENTITIES
- NOTATION
- NMTOKEN, NMTOKENS

Elément simple

- Il ne peut contenir que du texte
  - Ne peut avoir d'attributs
  - Ne peut être vide
  - Extensible par des contraintes

<xs:element name="name" type="type" />

  - **name** est le nom de l'élément
  - **<xsd:element name="contact" type="xsd:string" />**
  - Les Types les plus utlisés sont : xs:boolean xs:integer xs:date xs:string xs:decimal xs:time
  - D'autres attributs sont possibles
    - **default="default value"**
    - **fixed="value"**
  - Le nombre minimum et maximum d'occurrences est défini par
    - **minOccurs**
    - **maxOccurs** : valeur maximale **unbounded**

Elément complexe

- Un élément complexe est défini par
  - **<xs:element name="*name*">**
    <xs:complexType>
    … informations
    </xs:complexType>
    </xs:element>
- Exemple
  - **<xs:element name="personne">**
    <xs:complexType>
    <xs:sequence>
    <xs:choice>
    <xs:element name="firstName" type="xs:string" />
    <xs:element name="lastName" type="xs:string" />
    </xs:choice>
    </xs:sequence>
    </xs:complexType>
    </xs:element
- <xs:sequence> : les éléments doivent apparaître dans l'ordre
- <xs:choice> : c'est le (ou |) d'une DTD
- <xs:all> : les éléments peuvent apparaître dans n'importe quel ordre (maxOccurs 1)

Les attributs

- Ils sont généralement déclarés comme les éléments simples
  - **<xs:attribute name="*name*" type="*type*" />**
- Un attribut peut avoir trois attributs optionnels : use, default et fixed
- Contraintes d'occurrence

```
<xsd:attribute name="maj" type="xsd:date" use="optional"
   default="2009-1-1"/>
```

  - **use**="optional" | ="required" | ="prohibited"
- Utilisations des attributs (élément vide avec attribut)
  - **<xsd:element name="parution">**

```
<xsd:complexType>

  <xsd:attribute name="date" type="xsd:date" />

</xsd:complexType>
</xsd:element>
```

Référencer un élément

- Une fois on défini un élément avec name="..." on peut le référencer par ref="..."

```
<xsd:element name="auteur" type="xsd:string" />
<xsd:attribut name="ISBN" type="xsd:string" />
  <xsd:element name="livre">
    <xsd:complexType>
      <xsd:attribut ref="ISBN" />
  <xsd:sequence>
       <xsd:element ref="auteur" />
   <xsd:element name="titre" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Restriction sur les types

- La syntaxe générale de mise en œuvre

```
<xs:simpleType  name="name">  (ou xs:complexType)
      <xs:restriction base="type">
           ... Les restrictions ...
      </xs:restriction>
  </xs:element>
```

- Exemple

```
<xs:simpleType  name="age">
      <xs:restriction base="xs:integer">
           <xs:minInclusive value="0">
           <xs:maxInclusive value="140">
      </xs:restriction>
  </xs:element>
```

Types de restrictions sur les nombres

- Restrictions sur les chiffres
  - minInclusive -- nombre ≥ à la valeur donnée
  - minExclusive -- nombre > à la valeur donnée
  - maxInclusive – nombre ≤ à la valeur donnée
  - maxExclusive – nombre < à la valeur donnée
  - totalDigits – nb de chiffres
  - fractionDigits – nb chiffres après virgule

Types de restrictions sur les châines

- Restrictions sur les Strings
  - length – le nombre de caractères construisant la chîne
  - minLength -- le nombre de caractères minimum
  - maxLength -- le nombre de caractères maximum
  - pattern – Expression régulière à vérifier par la chaîne
  - whiteSpace – conditions sur l'espace
    - value="preserve" préserver les espaces
    - value="collapse" remplacer tous les espaces par un simple espace

Les patterns

- Contraintes sur type simple prédéfini
- Utilisation d'expression régulières
  - Similaires à celles de Perl
- Exemples
  - ```
    <xsd:simpleType name="CIN">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[A-Z]{2}d{8}"/>
        </xsd:restriction>
    </xsd:simpleType>
    ```
  - ```
    <xsd:simpleType name="password">
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="[a-zA-Z0-9]{8}" />
      </xsd:restriction>
    </xsd:simpleType>
    ```

Enumération

- Une énumeration fixe l'ensemble des valeurs prise par l'attribut value
- Exemple:
  - <xs:simpleType name="Saison">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Printemps"/>
            <xs:enumeration value="Eté"/>
            <xs:enumeration value="Automne"/>
            <xs:enumeration value="Hiver"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>

Autres types d'élément

- Élément texte avec attributs
  - <xsd:element name="parution">
    <xsd:complexType>
    <xsd:simpleContent>
    <xsd:extension base="xsd:string">
    <xsd:attribute name="date" type="xsd:date" />
    </xsd:extension>
    </xsd:simpleContent>
    </xsd:complexType>
    </xsd:element>
- Vous avez aussi la possibilité d'utiliser une restriction

Élément a contenu mixte

- Il peut contenir des éléments des attributs et du texte
  - Il est déclaré par l'attribut **mixed="true"**

```
<xs:element name="auteur">
<xs:complexType mixed="true">
<xs:all>
<xs:element name="nom" type="xs:string"/>
<xs:element name="né" type="xs:date"/>
</xs:all>
</xs:complexType>
</xs:element>
```

Héritage ou extension

- On peut étendre un type complexe de base
-
```
<xs:complexType  name="newType">
   <xs:complexContent>
      <xs:extension  base="autreType">
         ……
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```
- Exemple :

```
<xs:complexType name="AdressePays">
  <xs:complexContent>
    <xs:extension base="Adresse">
      <xs:sequence>
        <xs:element name="pays" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Portée d'un élément

- Un élément déclaré au début d'un schémas <schema> a une portée globale
- Les éléments déclarés à l'intérieur d'un type complexe sont locales
- ```
  <xs:element name="personne">
      <xs:complexType>
          <xs:sequence>
              <xs:element name="firstName" type="xs:string" />
              <xs:element name="lastName" type="xs:string" />
          </xs:sequence>
      </xs:complexType>
  </xs:element>
  ```
  - firstName et lastName sont déclarés localement
- Pour utiliser le type personne
  - <xs:element name="élève" type="personne"/>
  - <xs:element name="professeur" type="personne"/>
  ```

XPATH

- Langage d'interrogation (requêtes) et de parcours d'un document xml
- Langage de programmation compact (testes, fonctions et expressions)
- Recommandation w3C en 1999
  - Utilisé avec d'autres langages (XSLT et XPointer)
- Un document xml est vue comme un arbre (nœud, père, fils)
- Le chemin vers un nœud est décris de la même manière qu'un chemin vers un fichier sous Unix

Terminologie

```
<library>
  <book>

    <chapter>
    </chapter>

    <chapter>
      <section>
        <paragraph/>
        <paragraph/>
      </section>
    </chapter>

  </book>
</library>
```

- library est parent de book; book est parent de 2 chapters
- Les 2 chapters sont children de book et section est children de chapter
- Les 2 chapters ont le même parent book, ils sont siblings
- Library, book et le chapter 2 sont ancetors de section
- Les 2 chapters, la section et les 2 paragraph sont descendents de book

Syntaxe et sémantique

- Cheminement élémentaire
  - axes::sélecteur[predicat]
- Axes
  - parent, ancestor, ancestor-or-self
  - child, descendant, descendant-or-self
  - preceding, preceding-sibling
  - following, following-sibling
  - self, attribute, namespace
- Sélecteur
  - nom de nœud sélectionné (élément ou @attribut)
- Prédicat
  - [Fonction(nœud) = valeur]

Arbre XML

- Les éléments d'un arbre XML
  - La racine
  - Les éléments
  - Les attributs
  - Les instructions de traitement (PI)
  - Du texte
  - Les namespace
  - Les commentaires

Path ou chemin

- Un chemin qui commence par / est un chemin absolu débutant de la racine du document
  - Exemple : **/library/book/chapter/section/paragraph**
  - / signifie tout le document
- Un chemin qui ne commence pas par / représente un chemin débutant par le nœud courant
  - Exemple : **book/chapter**
- Un chemin qui commence par // peut débuter de n'importe quel nœud dans le document
  - **//book/chapter** : sélectionne chaque chapter fils de book
- Un nombre entre crochet représente l'indexe d'un élément (l'indexation débute par 1)
  - **//chapter/section[2]** sélectionne la deuxième section de chaque chapter dans le document XML

Les attributs

- Pour choisir un nœud avec un certain attribut
  - //nœud[@att]
  - //chapter[@num] : sélectionne les chapter avec l'attribut num
  - //@* : sélectionne tous les attributs de n'importe ou dans le document XML
  - //chapter[@num='3']
  - //chapter[not(@num)]
  - //chapter[normalize-space(@num)="3"] : supprimer l'espace avant et après la valeur de num avant comparaison

Les opérateurs

- **arihtmétiques: + - * div mod**
- **Egalité: = !=**
- **Relationnels: < > <= >=**
- **Boolean: or and**

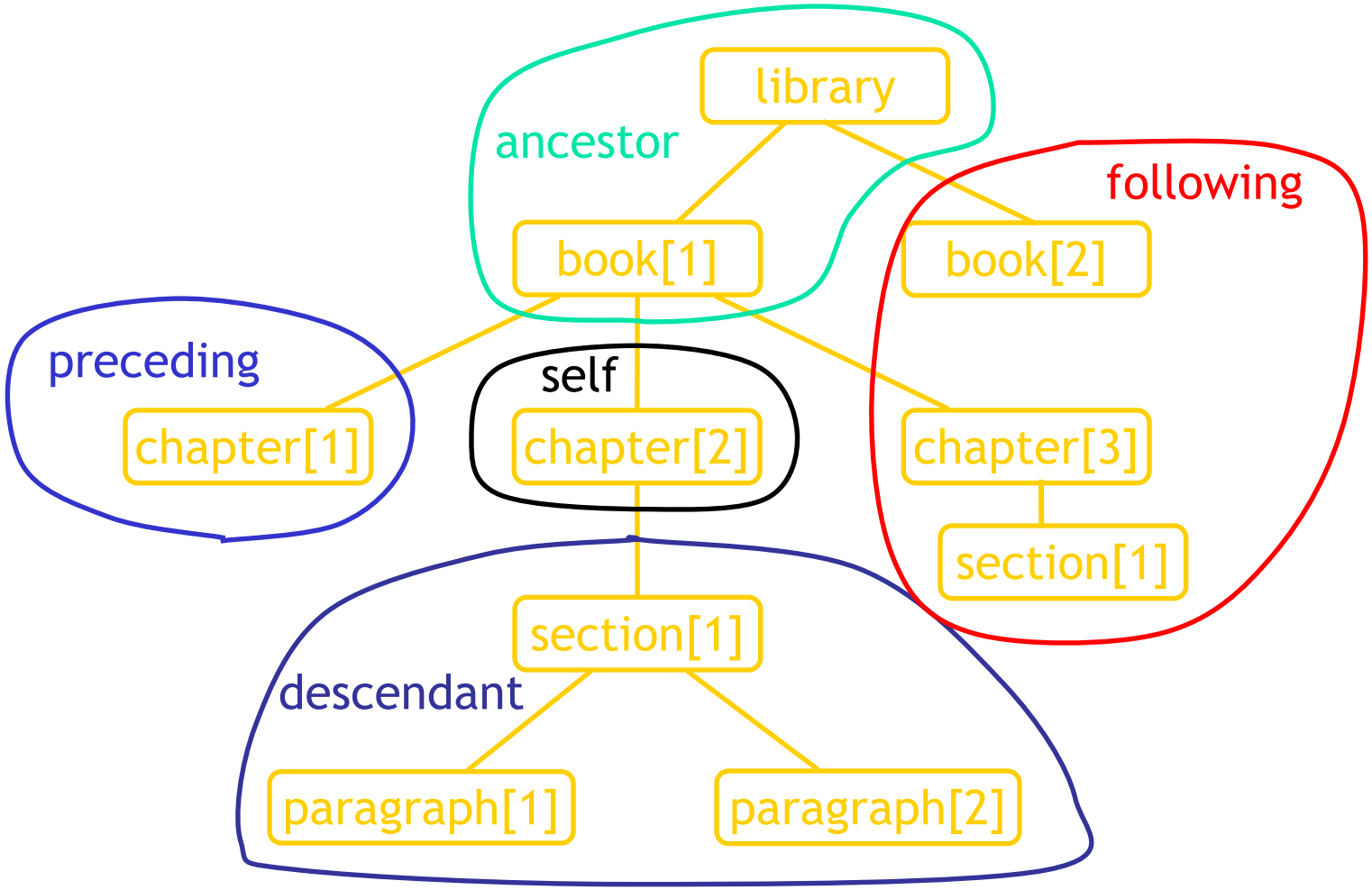
Exemples de fonctions XPATH

- last() : retourne un nombre égal à l'index du dernier nœud dans le contexte courant.
  - Exemple: /library/book/chapter[last()]
- position() : retourne un nombre égal à la position du nœud dans le contexte courant.
  - //chapter[position()=2]
- count(*elem*) counts the number of selected elements
  - Example: //chapter[count(section)=1] selectionne les chapters avec exactement une section comme children
- name() returns le nom de l'élément
  - Example: //*[name()='section'] est identique à //section
- starts-with(*arg1*, *arg2*) teste si *arg1* commence avec *arg2*
  - Example: //*[starts-with(name(), 'sec']
- contains(*arg1*, *arg2*) teste si *arg1* contient *arg2*
  - Example: //*[contains(name(), 'ect']
- concat (str1,str2) concatène deux chaînes de caractères

Axis

- **child** : contient les enfants directs du nœud contextuel.
- **descendant** : contient les descendants du nœud contextuel.
- **parent** : contient le parent du nœud contextuel, s'il y en a un.
- **ancestor** : contient les ancêtres du nœud contextuel.
- **following** : contient tous les nœuds situés après le nœud contextuel dans l'ordre du document
- **preceding** : contient tous les nœuds situés avant le nœud contextuel dans l'ordre du document,
  - à l'exclusion de tout descendant, des attributs et des espaces de noms.
- **self** : contient seulement le nœud contextuel.
- **following-sibling** :idem que following mais nœuds de même nom
- **preceding-sibling** :idem que preceding mais nœuds de même nom

Axes (vue en arbre)



Exemples Axis

- //book/descendant::* tous les descendants de chaque book
- //book/descendant::section toutes les section descendantes de chaque book
- //parent::* chaque élement parent, i.e., qui n'est pas terminal
- //section/parent::* tous les parents de l'élément section
- //parent::chapter chaque chapter parent, i.e., a des fils
- /library/book[3]/following::* tout élément après le 3 ième book
- attribute::* : sélectionne tous les attributs du nœud contextuel.

Xpath - Synthèse

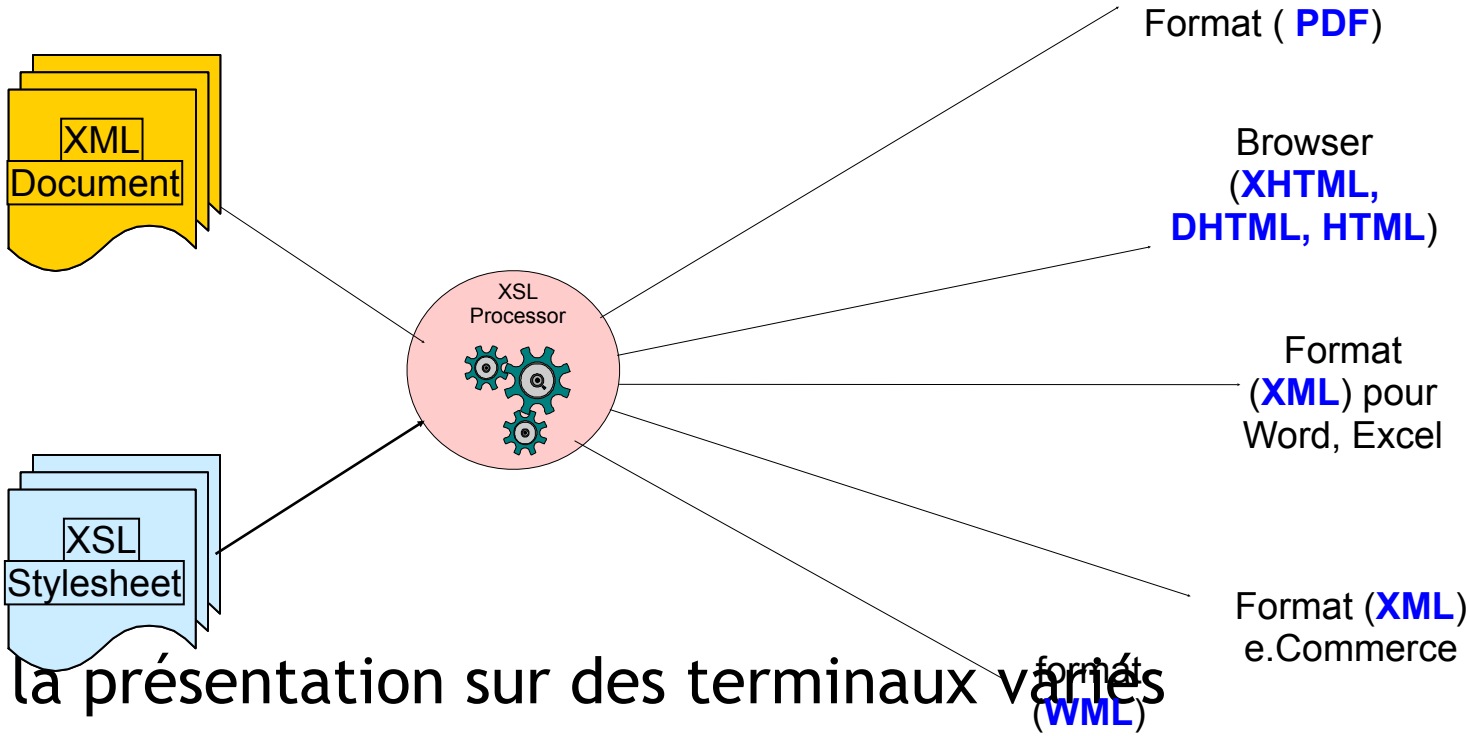
Sélecteur	Nœuds sélectionnés
/	Racine du document
//	Tout sous-chemin
*	Tout élément
nom	Elément de tag nom
.	Elément courant
@att	Attribut de nom att
text()	Tout nœud de type texte
processing-instruction(‘pro')	Processing instruction de nom pro
comment()	Tout nœud commentaire
node()	Tout nœud
id(‘val’)	Elément d'identifiant val

XSL (eXtended Style Langage)

- XSL est utilisé pour permettre de convertir les données d'un document XML en un autre document (HTML par exemple)
- Il est beaucoup plus puissant que CSS
  - Véritable langage de programmation par règles
- XSL est un langage XML constitué de deux parties:
  - Un langage pour transformer un document XML en un autre (XSLT: Transformations) - Nov. 1999
  - Un langage pour spécifier des instructions de formatage (XSL-FO: Formating Objects) – Oct. 2001

Publications avec XSL

- Plusieurs formats de publication d'un même jeu de données



- Il permet la présentation sur des terminaux variés

Document XSL

- Le fonctionnement du XSL est fondé sur les manipulations de modèles (templates)
- Les éléments du document XML d'origine sont remplacés (ou légèrement modifiés) par ces modèles
- Une feuille de style XSL est un document XML composé de règles de transformations
  - Une règle permet de décrire le style à appliqué à chaque nœud de l'arbre

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0 "
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!-- Les règles à appliquer -- >
</xsl:stylesheet>
```

XML et XSL

```
<?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="document">
<center>
<h1><xsl:value-of select="salutation"/></h1>
</center>
</xsl:template>
</xsl:stylesheet>
```

hello.xsl

```
<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="hello.xsl"?>
<document>
    <salutation>
     Salut Tout le Monde
    </salutation>
</document>
```

hello.xml

Règles de transformation

- Chaque règle (<xsl:template>) est définie par:
  - Une condition spécifiant le sous-arbre du document d'entrée auquel elle s'applique
  - Une production spécifiant le résultat de l'application de la règle

```
<xsl:template name="nommodele"   match="expression" >
</xsl:template>
```

  - name correspond au nom associé au modèle (optionnel)
  - match indique quel jeu de nœuds sera affecté par le modèle (chemin XPath)
- Exemple

```
<xsl: template match="/">
<html>
<head><title>Liste des Livres</title></head
<body>
      <h1>Liste des Livres</h1>
</body>
</html>
</xsl:template>
```

Sélectionner la racine du fichier XML

Instruction suplémentaires

- Parcourir l'arbre du document source
  - xsl:apply-templates
  - xsl:for-each
- Copier le contenu du document source dans le résultat
  - xsl:value-of select= …
- Teste
  - xsl:if
- Choix multiple
  - xsl:choose ... xsl:when ... xsl:otherwise
- Tri dans une boucle for-each
  - xsl:sort

xsl:value-of

- <xsl:value-of select="*XPath expression*"/> sélectionne le contenu d'un élément et le copie en sortie (html, xml, text)
  - L'attribut select est obligatoire
  - xsl:value-of doit terminer par slash
  - Appliquée sur le premier élément trouvé
- "." sélectionne le contenu texte d'un élément
- Exemples:

```
<h1> <xsl:value-of  select="produit[@code]"/> </h1>
```

```
<xsl:template match="salutation">
<h1><xsl:value-of select="."/></h1>
</xsl:template>
```

xsl:for-each

- xsl:for-each est une sorte de boucle pour parcourir l'arbre d'un document XML
- La syntaxe générale est

```
<xsl:for-each   select="XPath expression">
        Text to insert and rules to apply
 </xsl:for-each>
```

- Exemple: pour afficher le titre (title) de chaque (//book) dans une liste non ordonnée (<ul>) :

```
<ul>
  <xsl:for-each select="//book">
    <li>   <xsl:value-of select="title"/>   </li>
  </xsl:for-each>
 </ul>
```

xsl:apply-templates

- <xsl:apply-templates> effectue un appel récursive à partir du noeud courant
- Si on ajoute l'attribut select, le template est appliqué uniquement aux noeuds child satisfont la règle

```
<xsl:template match="/">
<center>
<h1><xsl:apply_templates/></h1>
</center>
</xsl:template>
<xsl:template match="chapitre">
    <xsl:value-of  select="."/>
</xsl:template>
</xsl:stylesheet>
```

Affichera le titre de tous les chapitres

xsl:if

- xsl:if permet d'ajouter une condition sur le contenu d'un élément
  - La condition est placer dans l'attribut test
  - Pas de if-else mais vous pouvez utiliser choose
- Exemple:

```
<xsl:for-each  select="//book">
  <xsl:if  test="author='M. Lahmer'">
    <li>
      <xsl:value-of  select="title"/>
    </li>
  </xsl:if>
</xsl:for-each>
```

- Les opérateurs de test sont
  - = != < >

xsl:choose

- xsl:choose ... xsl:when ... xsl:otherwise en XML est équivalente à switch ... case ... default en java
- syntaxe:

```
<xsl:choose>
    <xsl:when test="condition">
        ... code ...
    </xsl:when>
    <xsl:otherwise>
        ... code ...
    </xsl:otherwise>
</xsl:choose>
```

• xsl:choose est toujours placée dans la boucle xsl:for-each

xsl:sort

- On peut placer xsl:sort à l'intérieur d'un xsl:for-each
- L'attribut select de sort défini le champ sur lequel on doit effectuer le tri

- Exemple:

```
<ul>
    <xsl:for-each select="//book">
      <xsl:sort select="author"/>
      <li>   <xsl:value-of select="title"/> by
             <xsl:value-of select="author">  </li>
    </xsl:for-each>
  </ul>
```

  - Cet exemple crée une liste de titre et auteur triée par auteur

xsl:variable

- Permet de créer une variable pour une réutilisation ultérieure
  - Les attributs possibles :
    - name : spécifie le nom de la variable
    - select : expression XPath permettant de définir la valeur de la variable
  - Exemples

```
<xsl:variable name="num_chapitre" select="book/chapter/position()" />

<xsl:variable name="titre" select=" 'Introduction XML' " />
```

  - L'appel d'une variable se fait par $var
  - <xsl:value-of select="$titre"/>

xsl:output

- Permet de contrôler le type de fichier en sortie
  - Les attributs possibles sont
  - methode="[text|xml|html]"
  - ident="yes|no"
  - version=xml 1.0, html 4.01
  - doctype-public="spesification"
  - doctype-system="spesification"
  - encoding="type"
  - standalone="yes|no"
- Exemple
  - <xsl:output method="html" version="html4.01" encoding="ISO-8859-1" doctype-system="http://www.w3.org/TR/html4/strict.dtd" />

XML en output

- Ajout d'éléments **<xsl:element>**
  - Insère un nouvel élément dans la transformation. Le nom de l'élément est donné par l'attribut name.
  - <xsl:element name="nomelement" use-attribute-sets="attr"> </xsl:element>
  - Exemple

    <xsl:element name="nom">

    <xsl:value-of select="auteur" /> </xsl:element>
- Ajout d'attributs **<xsl:attribute>**
  - définit un attribut et l'ajoute à l'élément résultat de la transformation
  - <xsl:attribute name="nom">valeur</xsl:attribute>
  - Exemple

<xsl:element name="img"><xsl:attribute name="src">

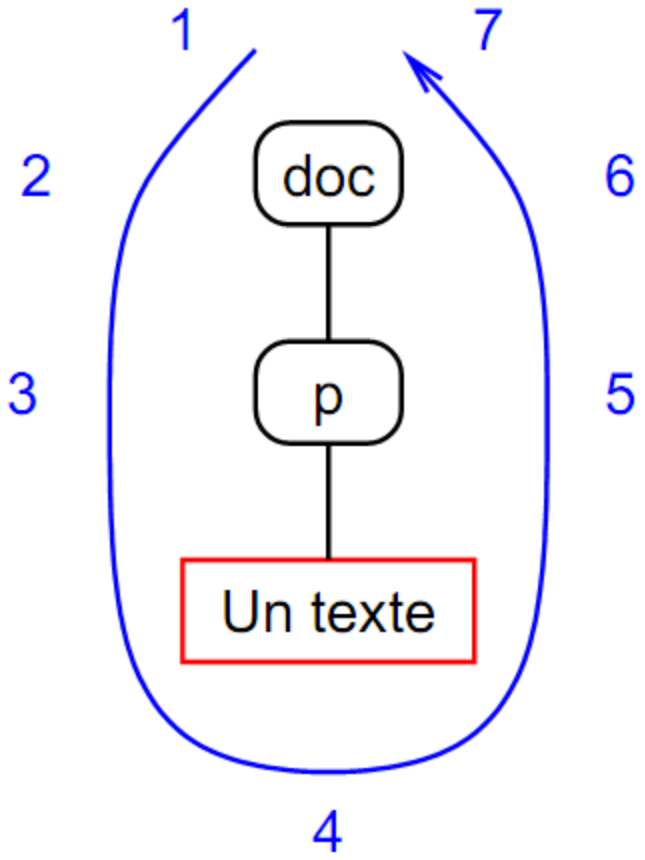
<xsl:value-of select="@source" /></xsl:attribute> </xsl:element>

L'API SAX

- SAX (Simple API for XML)
  - modèle simplifié d'événements
  - développé par un groupe indépendant :
  - http ://www.saxproject.org/
- Types d'événement :
  - début et fin de document ;
  - début et fin d'éléments ;
  - attributs, texte, … .
- Nombreux parseurs publics
  - MSXML3 de Microsoft
  - Xerces de Apache
  - JAXP de SUN (intégré depuis J2SDK 1.4)

Exemple

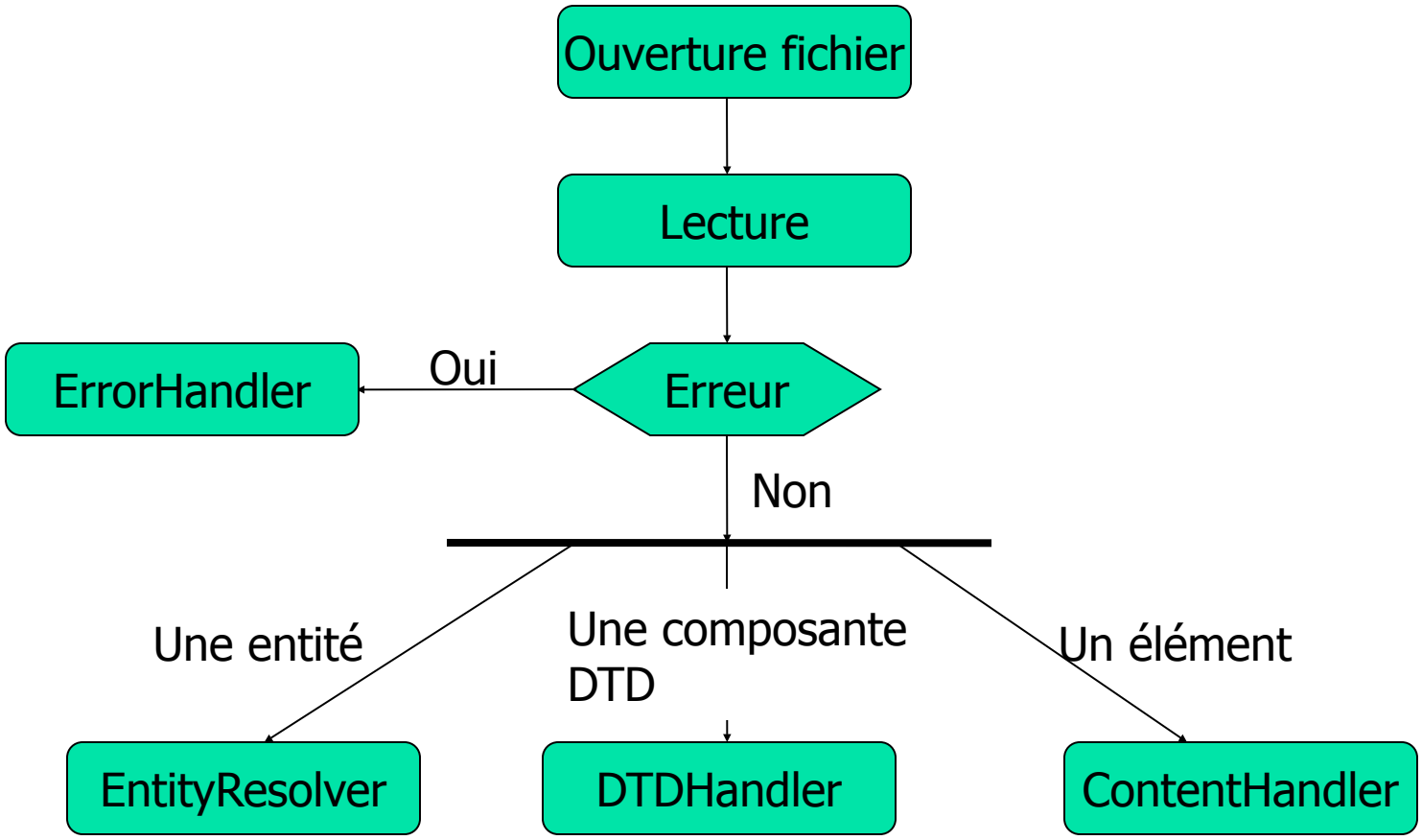
- Document XML
  - <?xml version="1.0"?>
  - <doc>
  - Un texte
  - </doc>
- Les événements générés
  - 1. start document
  - 2. start element : doc
  - 3. start element : p
  - 4. characters : Un texte
  - 5. end element : p
  - 6. end element : doc
  - 7. end document



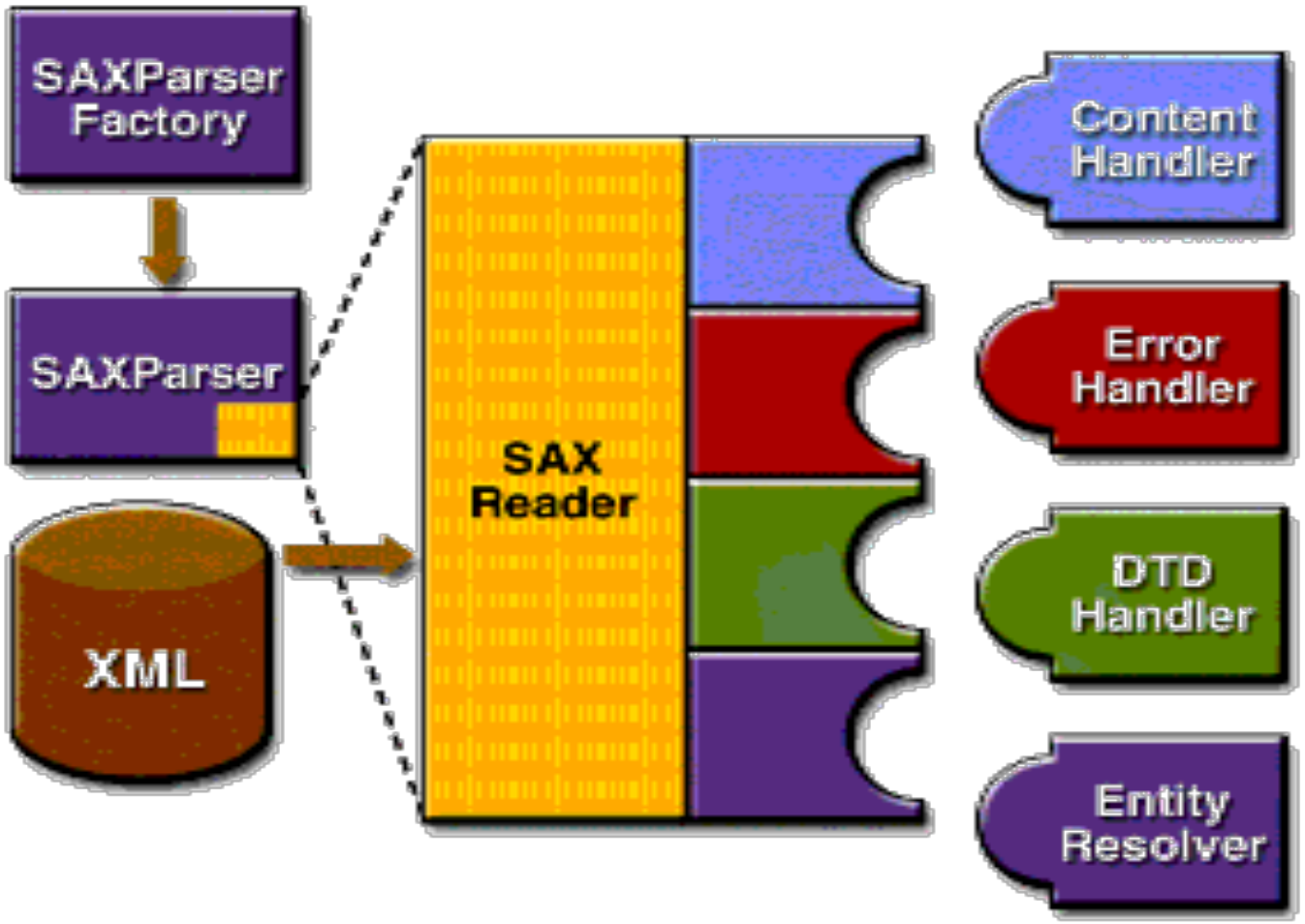
Principes

- Tout est basé sur quatre interfaces :
  - ContentHandler : la plus importante réagit aux évènements : une méthode par type d'évènement (exemple startDocument)
  - ErrorHandler : gestion des erreurs, en particulier les problèmes de validation
  - DTDHandler : gestion d'une petite partie des DTDs
  - EntityResolver : gestion du remplacement des entités externes
  - Les quatre interfaces sont implantées par DefaultHandler

Diagramme d'état



Les composants de l'API



Les paquetages

- Les classes et interfaces de SAX sont définies dans les packages suivants:
  - javax.xml.parsers : toutes les classes nécessaires pour supporter DOM et SAX
  - org.xml.sax
    - ContentHandler
    - DTDHandler
    - EntityResolver
    - ErrorHandler
    - XMLReader
  - org.xml.sax.helpers
    - XMLReaderFactory : permet de créer une classe XMLReader
- Les classes de l'API XSLT sont définis dans le package
  - javax.xml.transform
  - javax.xml.transform.sax
  - javax.xml.transform.stream

L’interface ContentHandler

- `void startDocument()`
- `void endDocument()`
- `void startElement(String namespaceURI, String localName, String qName, Attributes atts)`
- `void endElement(…idem…)`
- `void characters(char[] ch, int start, int length)`
- `endDocument(),`
- `processingInstruction(),`
- `setDocumentLocator(),`
- `skippedEntity(),`
- `ignorableWhitespace()`

Exemple

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

<?xml-stylesheet type='text/css'
      href='person.css'?>


<person >


<n:name xmlns:n="http://xml.ns.orgn">


</n:name>


</person>
```

startDocument( )

processingInstruction(String target, String data)

- Target: "xml-stylesheet"
- data: "type='text/css‘ href='person.css‘

startElement(..)

- URI: "http://xml.ns.org"
- localName: "n"
- qualifiedName: "n:name"
- atts: {}

Exemple

- Dans le cas de la méthode startElement
  - Les attributs sont stockés sous forme de liste
  - //pour chaque attribut

```
for (int i=0; i< attrs.getLength(); i++)
{
        // nom et valeur de l'attribut
        String nom = attrs.getQName(i);
        System.out.print(nom);
         String valeur = attrs.getValue(i);
        System.out.println("\t"+valeur);
}
```

  - D'autres méthodes sont disponibles
    - getLocalName(int index) : nom non qualifié de l'attribut par index.
    - getType(int index) : le type de l'attribut par index
    - getURI(int index) : l'espace de nom associé à l'attribut d'indexe index

Main()

```
// 1. Creation d'un SAXParserFactory et configuration
SAXParserFactory spf = SAXParserFactory.newInstance();
// 2. Creation d'un JAXP SAXParser
SAXParser saxParser = spf.newSAXParser();
// 3. Initialisation du fichier à interpréter
 InputStream xmlInput  = new FileInputStream("chemin vers
   fichier xml");
// 4. Création d'un handler
 parser handler=new parser();
// 5. On parse le document
saxParser.parse(xmlInput, handler);
```