Traitement de documents XML
Les API DOM et Sax

Sommaire

Les deux principales interfaces de programmation XML :

- DOM (*Document Object Model*), basé sur une représentation hiérarchique
- SaX (*Simple API for XML*), basé sur des déclencheurs (événements/action)

Les API XML

L'API SaX :

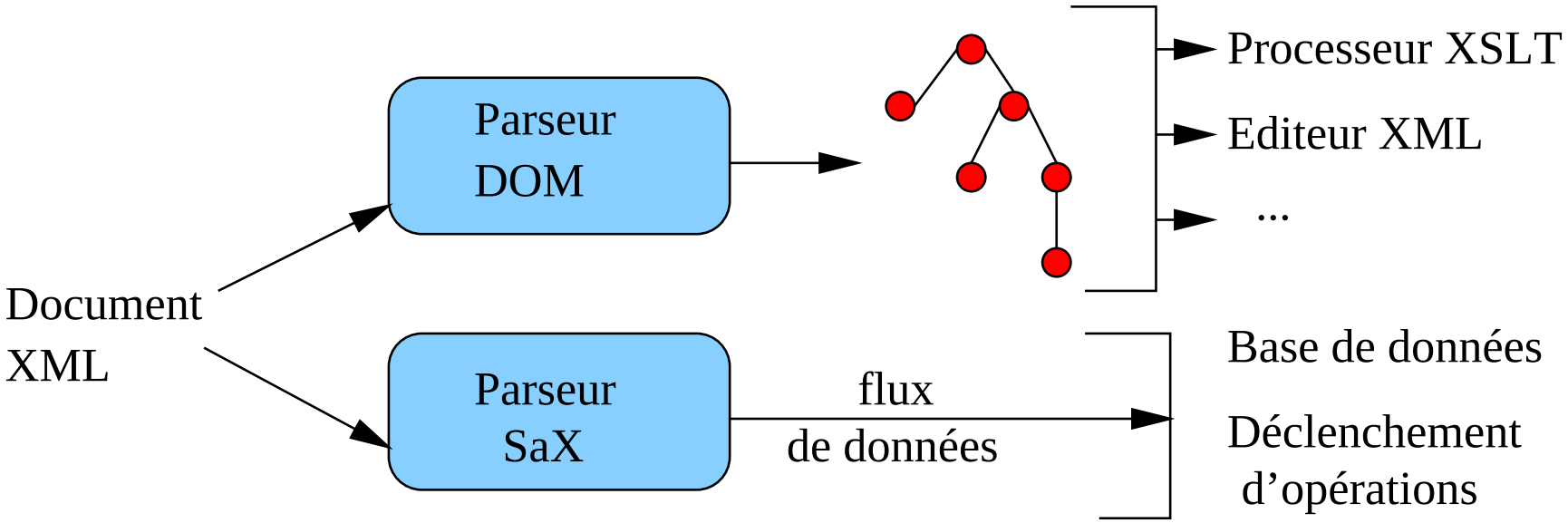
- Définit des *triggers* qui se déclenchent sur certaines balises.
- Adaptée aux applications qui extraient de l'information d'un document

L'API DOM :

- Construit une représentation du document en mémoire sous forme d'arbre
- Adaptée aux applications qui modifient ou traitent dans leur globalité un document.

DOM et SaX

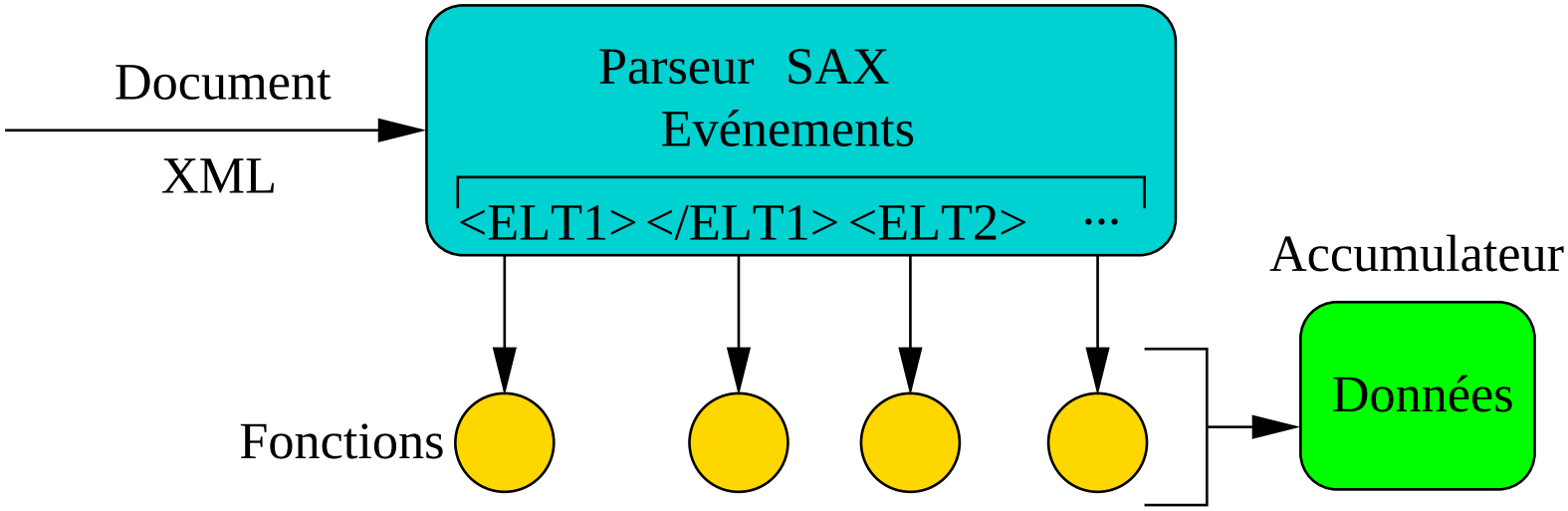
Toutes les applications XML passent par une phase préalable d'analyse



L'API SAX

Architecture SAX

Associer des événements aux balises :



Les événements SAX

- Début et fin de document
- Début et fin d'élément
- Instructions de traitements
- Commentaires
- ...

Les fonctions s'exécutent indépendamment => il faut leur faire partager une zone mémoire (« accumulateur »)

Exemple type : insertion dans une BD

À partir d'un flux XML contenant des films :

```
<FILMS>
  ...
  <FILM>
     <TITRE>Alien</TITRE>
     <ANNEE>1979</ANNEE>
     <AUTEUR>Ridley Scott</AUTEUR>
     <GENRE>Science-fiction</GENRE
     <PAYS>USA</PAYS>
  </FILM>
  ...
</FILMS>
```

Les événements

- Début de document : connexion à la base
- Balise <FILM> : création d'un enregistrement *Film*
- Balise <TITRE> : on affecte *Film.titre*
- Balise <ANNEE> : on affecte *Film.annee*
- etc..
- Balise </FILM> : insertion de l'enregistrement dans la base

Forme arborescente : le modèle DOM

Quelques mots sur DOM

DOM est une recommandation du W3C. Elle tient lieu de norme, mais

- C'est une spécification très abstraite : il peut y avoir des variantes selon les implantations
- DOM est issu d'une pratique de programmation pas toujours très propre du point de vue OO

$\Rightarrow$ les exemples sont basés sur l'implantation de la fondation XML/Apache.

L'API orientée-objet DOM

Un parseur DOM prend en entrée un document XML et construit :

- un **arbre** formé **d'objets**
- chaque objet appartient à une sous-classe de `Node`
- des opérations sur ces objets permettent de **créer** de nouveaux nœuds, ou de **naviguer** dans le document

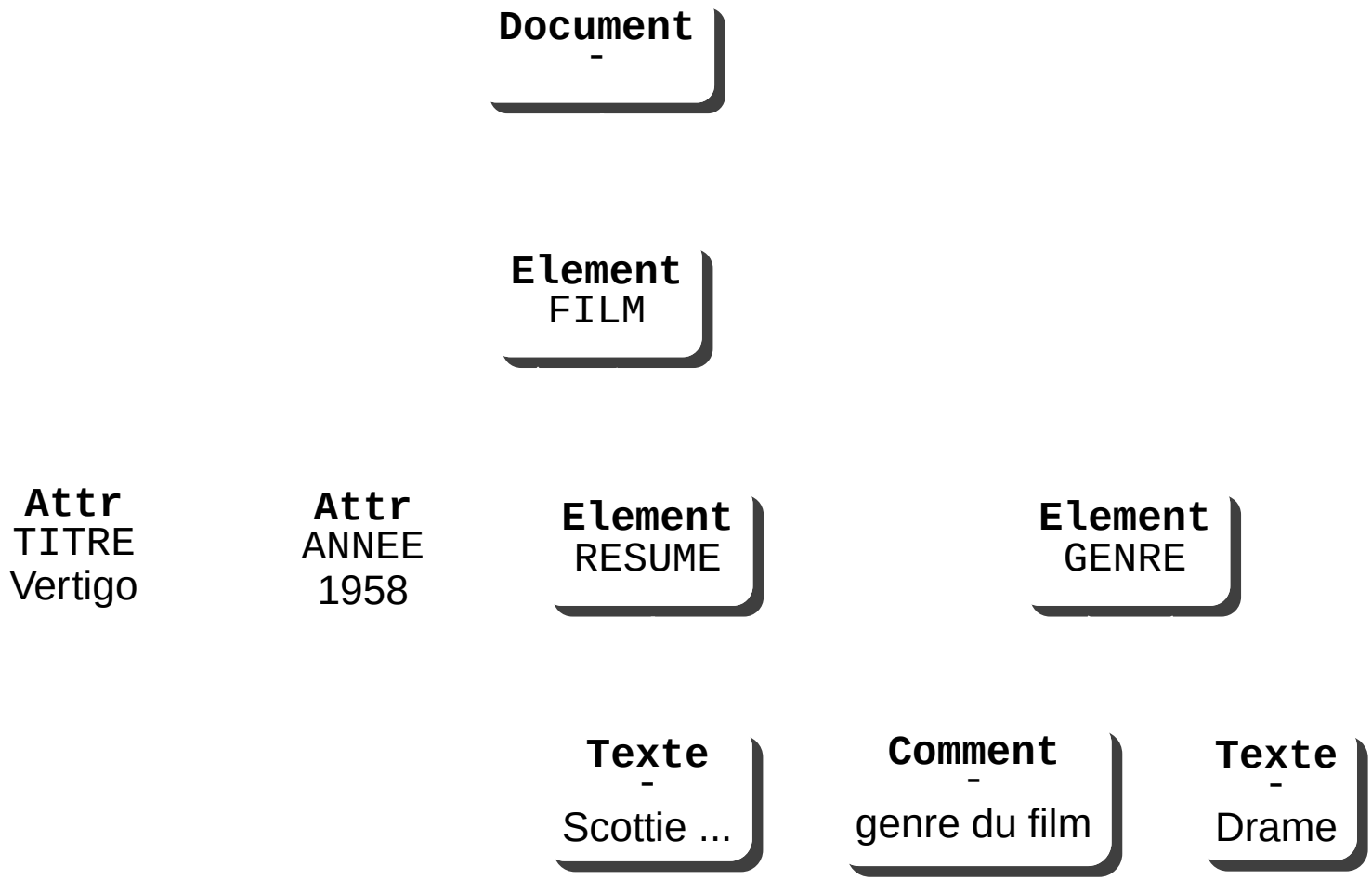
=> éditeurs XML, processeurs XSLT

DOM : représentation objet des arbres XML

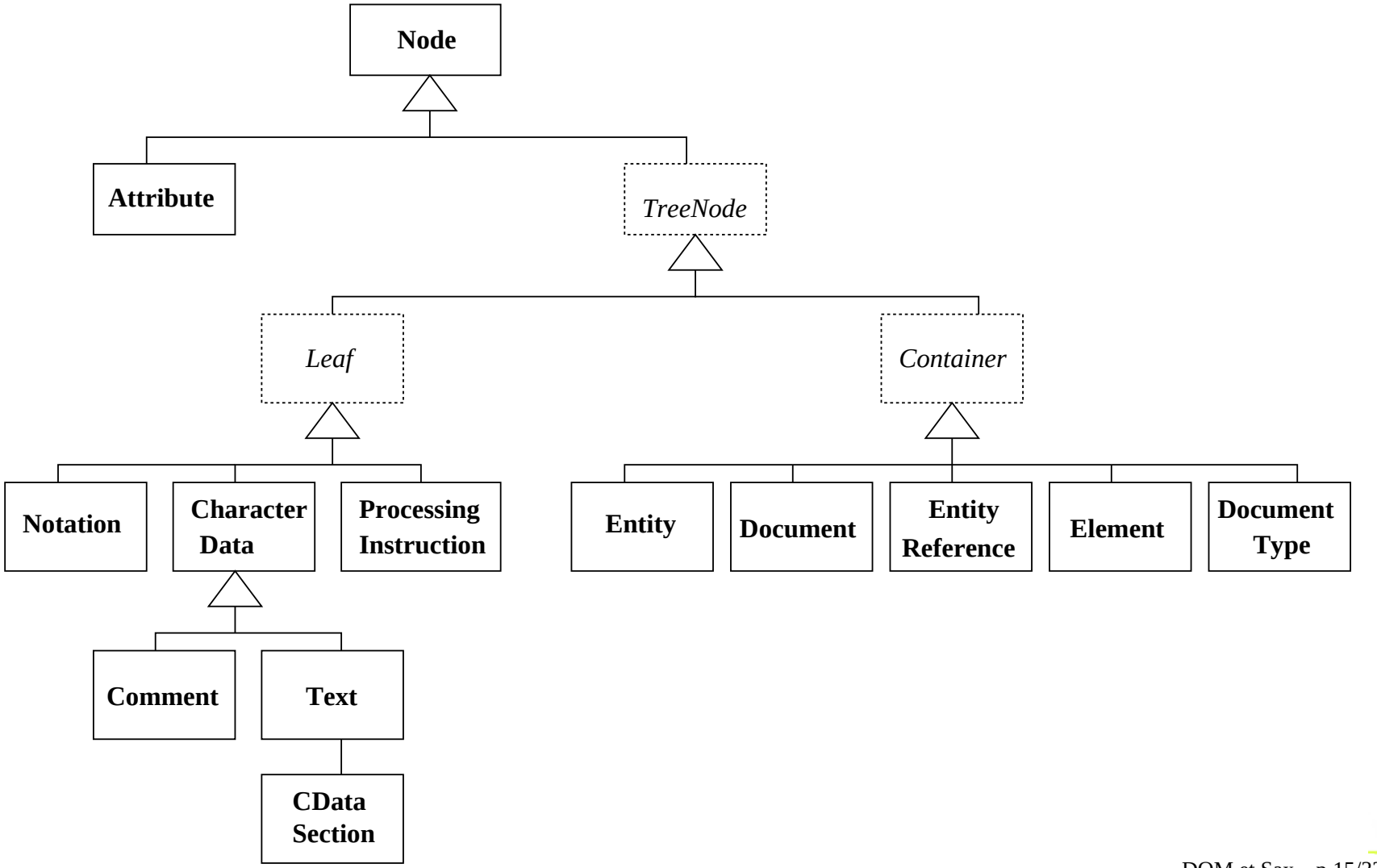
Un parseur DOM construit en mémoire une représentation du document où :

- chaque nœud est un objet de type **Node** ;
- chaque catégorie syntaxique est représentée par un sous-type de **Node** ;
- la racine du document est un nœud spécial, de type **Document**.

Exemple d'une représentation DOM



Principales classes DOM



La super-classe Node

Une approche faiblement objet

- Tout nœud DOM est un **Node**
- **En principe** les propriétés spécifiques à un sous-type devraient apparaître au niveau de ce sous-type
  Exemple : le nom, valable pour un nœud **Element**, pas pour un nœud **Text**
- **En pratique Node** rassemble toutes les propriétés de tous les types de nœuds

Propriétés du type Node

Propriété	Type	Propriété	Type
nodeType	**short**	*nodeName*	**String**
nodeValue	**String**	*parentNode*	**Node**
firstChild	**Node**	*lastChild*	**Node**
childNodes	**NodeList**	*previousSibling*	**Node**
nextSibling	**Node**	*attributes*	**NamedNodeM**

Propriétés nodeName et nodeValue

Type de nœud	nodeName	nodeValue
CDATASection	`#cdata-section`	contenu de la section CDATA
Comment	`#comment`	contenu du commentaire
Document	`#document`	`NULL`
DocumentType	nom de la DTD	`NULL`
Element	nom de l'élément	`NULL`
ProcessingInstruction	nom de la cible	le contenu (moins la cible)
Text	`#text`	contenu du nœud **Text**
Entity	nom de l'entité	`NULL`
EntityReference	nom de l'entité référencée	`NULL`
Attr	nom de l'attribut	valeur de l'attribut

TAB. 1 –

Opérations du type Node

Quelques exemples :

- *insertBefore* (**Node** `nouv`, **Node** `fils`)
- *replaceChild* (**Node** `nouv`, **Node** `ancien`)
- *removeChild* (**Node** `fils`)
- *appendChild* (**Node** `fils`)
- **boolean** *hasChildNodes*()

La classe Document

Représente la **racine du document**. C'est toujours le premier nœud créé.

- Il contient éventuellement une référence vers la DTD
- Il sert « d'usine » à créer de nouveaux nœuds avec les méthodes `createElement()`, `createTextNode()`, `createComment()`, etc

Un exemple de programme DOM

Le programme *Preordre.java* prend en argument un nom de fichier contenant un document XML.

- Il analyse ce document avec un parseur DOM
- Il parcourt ensuite les nœuds du document et numérote les nœuds de type **Text** par leur position dans l'arbre
- Enfin le document modifié est sérialisé.

$\Rightarrow$ implantation avec le parseur Xerces

Structure du programme

```
// Import des classes Java
import java.io.*, javax.xml.parsers.*,
     org.w3c.dom.*, Serialiseur;

class Preordre
{
 public static void main (String args [])
  {
    // Instanciation du parseur, analyse
    // du document, parcours
 }
private static int Parcours (Node noeud, int
{
    // Fonction de parcours et numérotation
}
}
```

Phase d'analyse

```
// Récupération du document
File fdom = new File (args[0]);

// Instanciation du parseur
DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
DocumentBuilder builder =
      factory.newDocumentBuilder();

// Analyse du document
Document dom = builder.parse(fdom);

// Début du parcours avec le numéro 1
Node elementRacine =
     dom.getDocumentElement();
Parcours (elementRacine, 1);
```

Parcours : numérotation

```
private static int Parcours (Node noeud,
                             int numero)
{
  String str = new String();
  numero++;
  // Numérotation du noeud s'il est de
  // type texte
  if (noeud.getNodeType() == Node.TEXT_NODE)
  {
     str = "(" + numero + ") "
               + noeud.getNodeValue();
     noeud.setNodeValue (str);
  }
  ...
```

Parcours : récursion

```
...
// Parcours récursif si le noeud a des fils
if (noeud.hasChildNodes())
{
  // Récupère la liste des fils du
  // noeud courant (liste de type NodeList)
  NodeList fils = noeud.getChildNodes();

  // Parcours de la liste et appel récursif
 for (int i=0; i < fils.getLength(); i++)
   numero = Parcours (fils.item(i), numero);
 }
 return numero;
}
```

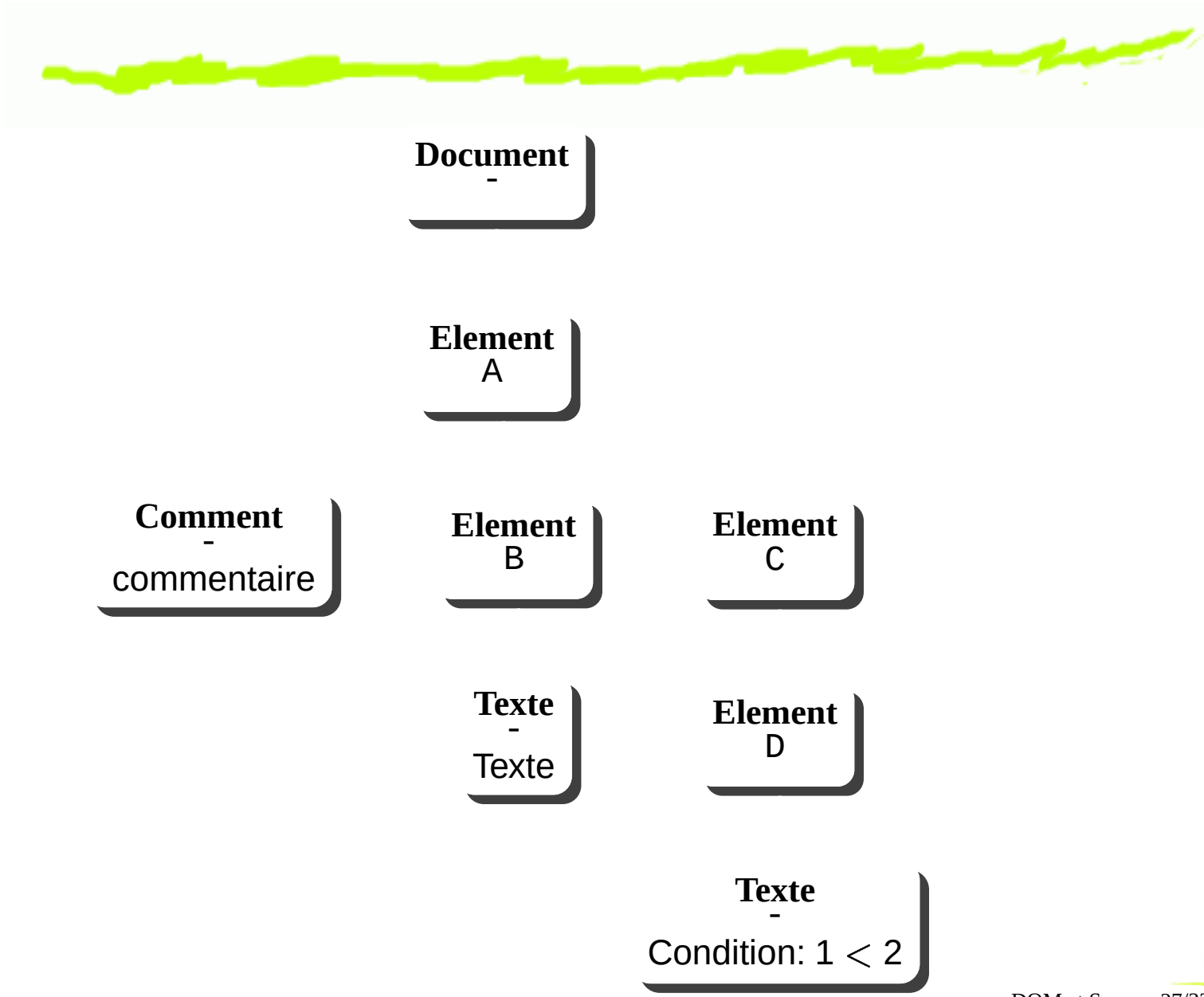
Sérialisation

Un traitement DOM s'effectue en trois phases :

- La version sérialisée est transformée en un arbre DOM en mémoire
- Des manipulations de ce document sont effectuées
- Le document modifié est **sérialisé** à nouveau (par exemple pour être stocké)

$\Rightarrow$ il existe plusieurs sérialisations possibles pour un même document.

Exemple : un document DOM



Une première version sérialisée

Tout est inclus dans le même fichier, avec une référence à une entité.

```
<?xml version=’1.0’?>
<A>
  <!-- commentaire -->
  <B>Texte</B>
  <C>
    <D>Condition: 1 &#60; 2</D>
  </C>
</A>
```

Seconde version

Tout est inclus dans le même fichier, avec une section CDATA

```
<?xml version='1.0'?>
<A>
  <!-- commentaire -->
  <B>Texte</B>
  <C>
    <D>Condition: <![CDATA[1 < 2]]></D>
  </C>
</A>
```

Troisième version

Une partie du document est inséré par référence à une entité externe.

```
<?xml version=’1.0’?>
<!DOCTYPE A [
<!ENTITY file
  SYSTEM "ExXMLArbreB.xml">
]>
<A>
  <!-- commentaire -->
  <B>Texte</B>
  &file;
</A>
```

Serialiseur : construction

```
public class Serialiseur
{
  Document doc;
  Node elementRacine;

  // Constructeur de la classe
  public Serialiseur (Document argdoc)
  {
    doc = argdoc;
    // On prend la racine du document
    elementRacine = doc.getDocumentElement()
  }
```

Serialiseur : sortie

```
// Méthode publique pour séraliser dans un f
public void sortie (String nomFichier)
        throws IOException
{
 FileWriter fichier = null;

 fichier = new FileWriter (nomFichier);
 fichier.write
  ("<?xml version=\"1.0\" encoding=\"ISO-885

 // Parcours récursif
 this.parcours (fichier, elementRacine);
 fichier.close();
}
```

Serialiseur : parcours

```
private void parcours (FileWriter fichier,
                       Node noeud) throws IOExcepti
{
 // Traitement des noeuds de type ELEMENT
 if (noeud.getNodeType() == Node.ELEMENT_NOD
 {
  // On crée la balise ouvrante
  fichier.write ("<" + noeud.getNodeName()
                 + this.serialiserAttrs(noeud)
                 + ">");
  // Parcours récursif si le noeud a des fil
  if (noeud.hasChildNodes()) { ... }

  // Et la balise fermante
  fichier.write ("</" + noeud.getNodeName()
 }
```