Espaces de noms, DOM, SAX

Dan VODISLAV

CY Cergy Paris Université

Licence Informatique L3

Plan

- Espaces de noms
- Interfaces de programmation
  - DOM
  - SAX

Espaces de noms

- Espace de noms (« namespace »)
  - Collection de noms d'éléments ou noms d'attributs, identifiée par un URI
  - Idée: rajouter un préfixe afin de rendre "uniques" et identifiables les noms de ses propres balises
    - Unicité du préfixe: à travers l'URI associée
  - Partage: on peut utiliser des espaces de noms « standard » (Dublin Core, FOAF, etc.)

```
<document xmlns:dc='http://purl.org/dc/elements/1.1/'>
  <art:film xmlns:art='http://www.pariscope.fr/`
            dc:title='Décalage horaire'
            dc:creator='Danièle Thompson'>
    <com:acteur xmlns:com='http://www.comedie.fr/'
                com:nom='Juliette Binoche' />
  </art:film>
</document>
```


Nom qualifié et nom universel

- Nom qualifié = nom "visible" dans le document
  - Composé d'un préfixe (optionnel) et du nom de l'élément (nom local)
- Nom universel = vrai nom, vraie "signification"
  - Obtenu à partir du nom qualifié en remplaçant le préfixe par l'URI
  - Composé d'une URI (optionnelle) et du nom de l'élément (nom local)

```
<document xmlns:dc='http://purl.org/dc/elements/1.1/'>
  <art:film xmlns:art='http://www.pariscope.fr/`
            dc:title='Décalage horaire'
            dc:creator='Danièle Thompson'>
    <com:acteur xmlns:com='http://www.comedie.fr/'
                com:nom='Juliette Binoche' />
  </art:film>
</document>
```

Nom qualifié	Nom universel
art:film	http://www.pariscope.fr/:film
com:acteur	http://www.comedie.fr/:acteur
dc:title	http://purl.org/dc/elements/1.1/:title
dc:creator	http://purl.org/dc/elements/1.1/:creator

Espaces de nom sans préfixe

- Dans l'utilisation d'un espace de noms le préfixe n'est pas obligatoire
  - Espaces de noms définis par `xmlns=URI` au lieu de `xmlns:pre=URI`
- Signification
  - Si un espace de noms sans préfixe est défini dans un élément A, tous les éléments inclus dans A font partie de l'espace de noms (pas les attributs!)

```
<document xmlns='http://www.pariscope.fr/'>
  <film titre='Décalage horaire'>
    <acteur>Juliette Binoche</acteur>
  </film>
</document>
```

Nom qualifié	Nom universel
`document`	`http://www.pariscope.fr/:document`
`film`	`http://www.pariscope.fr/:film`
`acteur`	`http://www.pariscope.fr/:acteur`
`titre`	`titre`

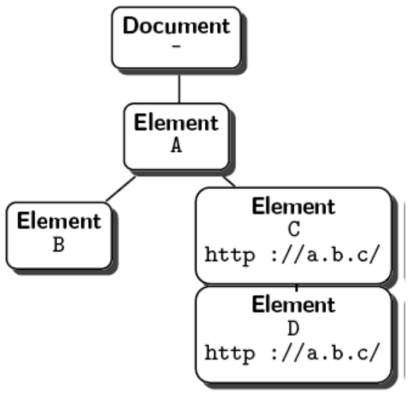
Attribution d'un nom universel pour un nom qualifié

- Problème: étant donné un nom d'élément avec ou sans préfixe, quel est son nom universel?
- Espaces de noms sans préfixe
  - Un élément sans préfixe appartient à l'espace de noms défini par *l'attribut `xmlns` de l'ancêtre le plus proche*. Si aucun ancêtre n'a d'attribut *`xmlns`*, l'élément n'appartient à aucun espace de noms.
  - Un attribut sans préfixe n'appartient à aucun espace de noms
- Espaces de noms avec un préfixe `pre`
  - L'élément (ou l'attribut) avec le préfixe `pre` appartient à l'espace de noms défini par *l'attribut `xmlns:pre` de l'ancêtre le plus proche*. Il y a erreur si aucun ancêtre n'a d'attribut *`xmlns:pre`*.

Exemples

- L'absence de préfixe dans un nom ne signifie pas que le nom universel n'a pas d'URI

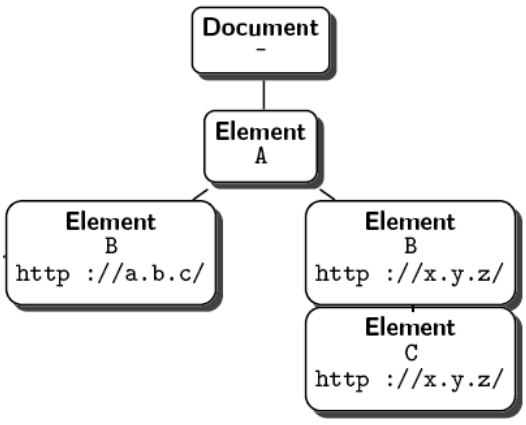
```
<?xml version="1.0"?>
<A>
  <B/>
  <C xmlns="http://a.b.c/">
    <D/>
  </C>
</A>
```



Exemples (suite)

- Deux éléments avec le même préfixe peuvent appartenir à des espaces de noms différents

```
<?xml version="1.0"?>
<A xmlns:p="http://a.b.c/">
  <p:B/>
  <p:B xmlns:p="http://x.y.z/">
    <p:C/>
  </p:B>
</A>
```



Interfaces de programmation pour XML

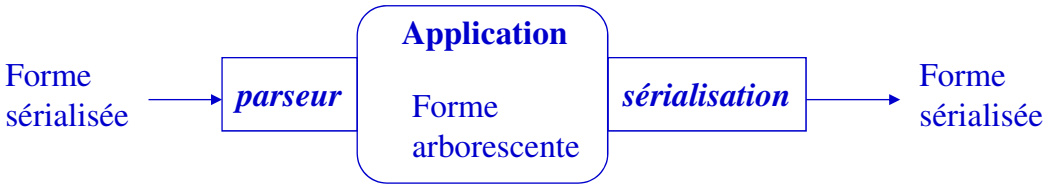
- Interfaces de programmation (API)
  - Utilisation de données XML dans les programmes
  - Opérations: recherche, transformation, validation, sérialisation, etc.
- Deux approches:
  - Utilisation de la forme arborescente XML: DOM
    - Avantage: utilisation très générale et simple, la plus utilisée
    - Inconvénient: stockage en mémoire de la structure d'arbre
  - Utilisation de la forme sérialisée XML (chaîne de caractères): SAX
    - Avantage: pas besoin de stocker le document en mémoire, rapidité
    - Inconvénient: programmation complexe, parfois le stockage est nécessaire

DOM et SAX

- Modèle + API basée sur ce modèle
- Les deux produisent une forme arborescente
  - DOM: arbre stocké en mémoire
  - SAX: arbre « virtuel », parcouru à la volée
- Niveau d'abstraction
  - SAX: niveau plus bas (événements de début/fin de balise)
    - Fonction principale: ***parseur XML*** paramétrable avec des actions
  - DOM: niveau plus haut (nœuds d'arbre)
    - Parseur, mais aussi parcours de l'arbre, modification, etc.
    - Un parseur DOM peut être écrit en SAX !
- Disponibles dans plusieurs langages de programmation
  - Java, C++, Perl, etc.

DOM

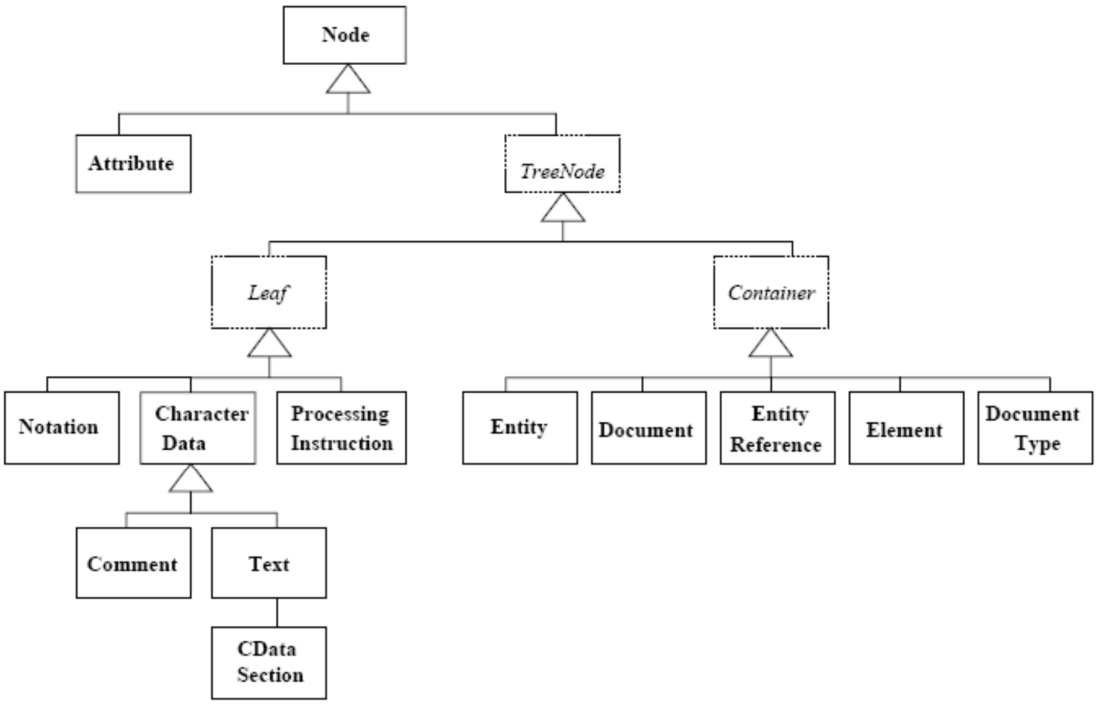
- DOM = *Document Object Model*
  - Modèle de type arbre pour les documents XML
  - API de programmation XML basée sur ce modèle
    - Construit et manipule des arbres XML en mémoire



Arbres DOM

- Un parseur DOM: document XML → **arbre** formé de **nœuds**
  - chaque nœud est un objet implémentant l'interface *Node*
  - des opérations sur ces objets permettent de **créer, modifier, détruire** des nœuds, ou de **naviguer** dans le document
- Nœuds de plusieurs types (sous-interfaces de *Node*)
  - Racine: type *Document*
  - Les autres types de nœuds: *Element*, *Attribute*, *Text*, *Comment*, etc.
  - Structure d'arbre
    - Nœuds feuilles (« Leaf »)
    - Nœuds containeurs (« Container »): nœuds pouvant avoir des descendants
    - Nœuds attribut (« Attribute »): statut particulier – similaires aux feuilles, mais avec quelques différences

Types de nœuds DOM



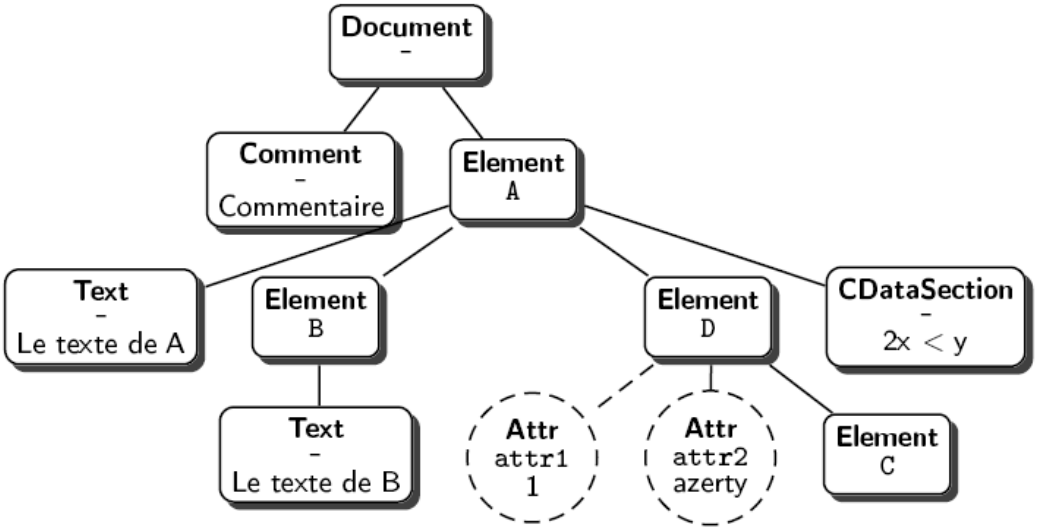
Représentation textuelle

- Exemple

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

<!-- Commentaire -->
<A>Le texte de A
  <B>Le texte de B</B>
  <D attr1="1" attr2="azerty">
   <C/>
  </D>
  <![CDATA[2x < y]]>
</A>
```

Représentation DOM



Types de programmation DOM

- *Node*: interface commune pour tous les nœuds DOM
  + interfaces spécifiques pour chaque type de nœud (élément, texte, ...)

→ Deux façons de programmer le parcours d'un arbre DOM

- Voir tous les nœuds comme des objets de type *Node*
  - Avantage: arbre homogène, plus simple pour un parcours complet
  - Inconvénient: méthodes simples (communes) ou comportement variable
- Voir chaque nœud avec son type spécifique
  - Avantage: méthodes spécifiques pour chaque type, plus puissantes
  - Inconvénient: arbres hétérogènes, plus difficile à programmer

- En pratique: deux méthodes principales
  – Parcours de l'arbre DOM comme un arbre d'objets *Node* + actions en fonction du type du nœud courant
  – Parcours des nœuds *Element* (balises) guidé par leur nom et accès à leurs attributs, contenus textuels, etc.

Principales propriétés de *Node*

Propriété	Type
nodeType	short (int)
nodeValue	String
firstChild	Node
childNodes	NodeList
nextSibling	Node

Propriété	Type
nodeName	String
parentNode	Node
lastChild	Node
previousSibling	Node
attributes	NamedNodeMap

NodeList: liste (tableau) de *Node*

- propriété ***length*** → *int*
- méthode ***item***(*int*) → *Node*

NamedNodeMap: table d'association par nom

- pareil que *NodeList* + méthode ***getNamedItem***(*String*) → *Node*
- aussi ***setNamedItem***, ***removeNamedItem***

Méthodes de *Node* par rapport à ses enfants

- ***insertBefore*** (*Node* nouveau, *Node* enfant)
  - Insère le nouveau nœud en tant que nouvel enfant, juste avant *enfant*
- ***appendChild*** (*Node* nouveau)
  - Insère le nouveau nœud en tant que nouvel enfant, en dernière position
- ***replaceChild*** (*Node* nouveau, *Node* ancien)
  - Remplace l'ancien enfant avec le nouveau
- ***removeChild*** (*Node* enfant)
  - Supprime le nœud enfant en question
- *boolean* ***hasChildNodes*** ()
  - Vérifie si c'est un nœud feuille

Méthodes de *Document* pour la création de nœuds

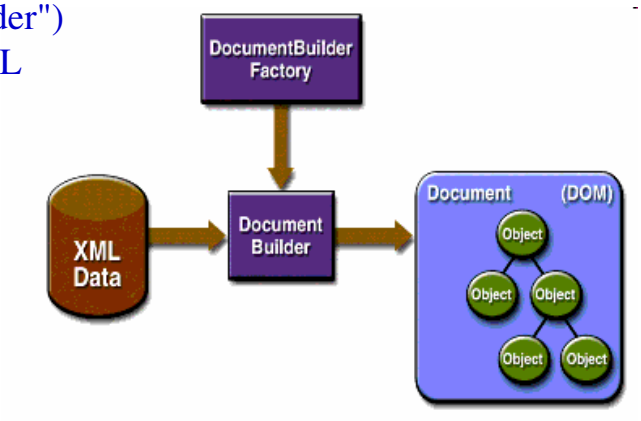
- Le nœud *Document*: le premier créé dans un arbre DOM
  - Joue le rôle de *fabrique* de nœuds pour l'arbre dont il est la racine
- Méthodes
  - ***createElement()*** : crée et retourne un nœud *Element*
  - ***createTextNode()*** : crée et retourne un nœud *Text*
  - ***createCommentNode()*** : crée et retourne un nœud *Comment*
  - …

Programmation DOM en Java

- JAPX: Java API for XML Processing
  - Incluse dans Java 1.6 (version JAPX 1.4)

```
import javax.xml.parsers.*
import org.w3c.dom.*
```

- Idée générale
  - *Parseur DOM* ("Document Builder") qui transforme un document XML en arbre DOM
  - Le parseur est produit par une *fabrique de parseurs*
  - L'arbre DOM est manipulé à l'aide des méthodes de parcours, modification, … de l'API



Obtenir un arbre DOM

- A mettre dans un `try … catch`

```
try{
  // création d'une fabrique de parseurs
  DocumentBuilderFactory fabrique =
       DocumentBuilderFactory.newInstance();
  fabrique.setValidating(true); //si l'on veut vérifier une DTD

  // création d'un parseur
  DocumentBuilder parseur = fabrique.newDocumentBuilder();

  // transformation d'un fichier XML en DOM
  File xml = new File("exemple.xml");
  Document document = parseur.parse(xml);
  ...
} catch(ParserConfigurationException pce){
     System.out.println("Erreur de configuration du parseur DOM");
} catch(SAXException se){
     System.out.println("Erreur lors du parsing du document");
} catch(IOException ioe){
     System.out.println("Erreur d'entrée/sortie");
}
```


Méthodes Java pour *Node*

- Pour chaque propriété de *Node* → une méthode *get*
  - *getNodeType(), getNodeValue(), getFirstChild(), …*
  - Type: *Node.ELEMENT_NODE, Node.TEXT_NODE, Node.DOCUMENT_NODE, Node.ATTRIBUTE_NODE*, …
  - Valeur*:* pour les nœuds de type attribut, texte, commentaire, …
  - Attributs: pour élément
  - Nom: pour attribut, élément, entité, …
    - En plus: *getLocalName(), getPrefix(), getNamespaceURI()*
- Texte
  - *getTextContent()*: concaténation du texte dans tout le sous-arbre
- Des méthodes de modification
  - *setNodeValue, setTextContent, …*
  - *insertBefore, appendChild, replaceChild, removeChild*

Méthodes Java pour *Element*

- Informations sur l'élément

  String ***getTagName*** () : nom de l'élément

  boolean ***hasAttribute*** (*String* anom) : existence d'un attribut appelé *anom*

- Accès aux sous-éléments

  NodeList ***getElementsByTagName***(*String* nom) : liste des sous-éléments appelés *nom* (à n'importe quel niveau de profondeur!)

- Accès aux attributs

  String ***getAttribute*** (*String* anom) : valeur de l'attribut appelé *anom*

  Attr ***getAttributeNode*** (*String* anom) : nœud attribut appelé *anom*

  void ***setAttribute*** (*String* anom, *String* val) : modifie/ajoute l'attribut *anom*

  void ***setAttributeNode*** (*Attr* node) : ajoute le nœud attribut *node*

  void ***removeAttribute*** (*String* anom) : supprime l'attribut *anom*

  Attr ***removeAttributeNode*** (*Attr* node) : supprime le nœud attribut *node*

Parcourir un arbre DOM de nœuds *Node*

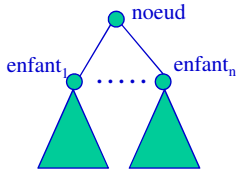
- Parcours récursif à partir de l'élément racine

```
Document document = parseur.parse(xml);
//obtenir l'élément racine
Node racine = document.getDocumentElement();
//explorer par appel récursif
TypeRésultat res = explorer(racine, ...);
```

- Le parcours peut:
  - Calculer un résultat: nombre de nœuds, d'éléments, représentations variées sous forme de chaîne de caractères, etc.
  - Modifier l'arbre DOM: ajout, suppression, modification de nœuds
  - Réaliser des actions: affichage, appels d'autres programmes

Exploration récursive

- Forme générale de la méthode d'exploration




```
static TypeRésultat explorer(Node noeud, ...){
   //initialisation résultat
   TypeRésultat resultat = ...;
  //traitement nœud courant
  if (noeud.getNodeType() == Node.TEXT_NODE){
       ... //action pour les nœuds texte
  } else if (noeud.getNodeType() == Node.ELEMENT_NODE){
        ... //action pour les nœuds élément
  } ...
  //parcours récursif
  if (noeud.hasChildNodes()){
       NodeList enfants = noeud.getChildNodes();
       for(int i=0; i<enfants.getLength(); i++){
          TypeRésultat resenf = explorer(enfants.item(i), ...);
          ... //combiner resenf avec resultat
       }
  }
  return resultat;
}
```


Exemple: modifier chaque nœud texte

- On rajoute la chaîne " (modifié)" à la fin de chaque texte

```
static void explorer(Node noeud){
  //traitement nœud courant seulement s'il est un texte
  if (noeud.getNodeType() == Node.TEXT_NODE){
       String modif = noeud.getNodeValue() + " (modifié)";
       noeud.setNodeValue(modif);
  }
  //parcours récursif
  if (noeud.hasChildNodes()){
       NodeList enfants = noeud.getChildrenNodes();
       for(int i=0; i<enfants.getLength(); i++){
          explorer(enfants.item(i));
       }
  }
}
```

Parcourir un arbre DOM d'éléments *Element*

- On part de l'élément racine et on descend jusqu'à l'élément souhaité à l'aide de *getElementsByTagName*
  - Avantage: on n'a pas à s'occuper du type des nœuds, on ne traite que des éléments
- Exemple: compter les éléments *A* dans le document, qui ont un attribut *a*

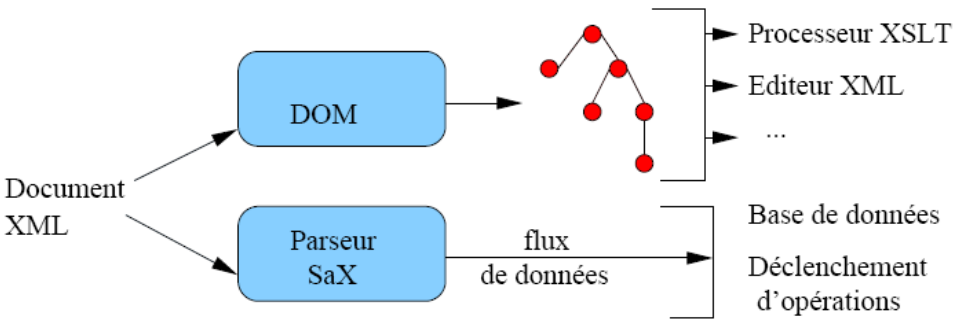
```
Document document = parseur.parse(xml);
Element racine = document.getDocumentElement();
int compteur = 0;

//liste des descendants qui s'appellent A
NodeList listeA = racine.getElementsByTagName("A");

//parcours liste à la recherche d'attributs a
for(int i=0; i<listeA.getLength(); i++){
   Element e = (Element) listeA.item(i);
   if (e.hasAttribute("a")) compteur++;
}
```

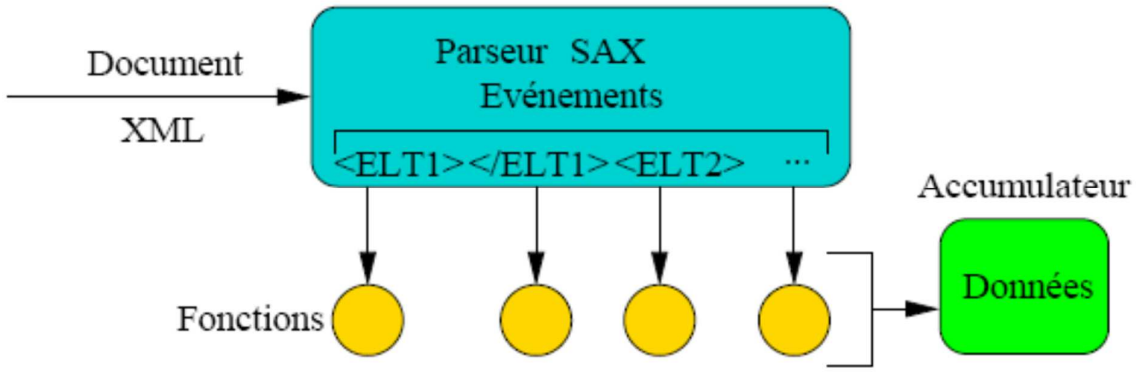

SAX

- SAX = *Simple API for XML*
  - Document XML parcouru sous forme sérialisée (chaîne de caractères, flux) → séquence d'événements
- Événements générés pour:
  - Début et fin du document
  - Début et fin d'un élément
  - Commentaire, instruction de traitement, …



Programmation SAX

- Déclencheurs spécifiques à chaque événement
  - Partagent un même espace mémoire (« accumulateur »)
- Programmation SAX: écrire des gestionnaires d'événements (« handlers »)



Utilisation

- Packages à importer dans JAPX

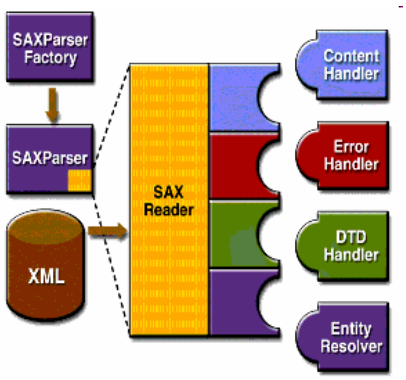
```
import javax.xml.parsers.*
import org.xml.sax.*
import org.xml.sax.helpers.*
```

- Création d'un parseur SAX

```
try{
  // création d'une fabrique de parseurs SAX
  SAXParserFactory fabrique = SAXParserFactory.newInstance();
  fabrique.setValidating(true); //si l'on veut vérifier une DTD
  // création d'un parseur SAX
  SAXParser parseur = fabrique.newSAXParser();
  // lecture d'un fichier XML avec un DefaultHandler
  File xml = new File("exemple.xml");
  DefaultHandler gestionnaire = new DefaultHandler();
  parseur.parse(xml, gestionnaire);
} catch(ParserConfigurationException pce){
    System.out.println("Erreur de configuration du parseur DOM");
} catch(SAXException se){
    System.out.println("Erreur lors du parsing du document");
} catch(IOException ioe){
    System.out.println("Erreur d'entrée/sortie");
}
```

Les gestionnaires (handlers)

- Le parseur gère plusieurs gestionnaires
  - De contenu: éléments, texte, attributs, …
  - De DTD
  - D'entités
  - D'erreurs
- *DefaultHandler*
  - Ne fait rien
  - Toutes les méthodes des gestionnaires sont *null*



Programmer un gestionnaire

```
public class MonHandler extends DefaultHandler{
  //variables locales ("accumulateur")
  private ...;

  //constructeur
  public MonHandler(){
    super();
    ... //initialisation des variables locales
  }

  //détection d'ouverture de balise
  public void startElement(String uri, String localName,
                           String qName, Attributes attributes)
                           throws SAXException{
    if(qName.equals("...")) ...
  }

  //détection de fin de balise
  public void endElement(String uri, String localName,
                         String qName) throws SAXException{
    if(qName.equals("...")) ...
  }


  //suite page suivante ...
```

Programmer un gestionnaire (suite)

```
   //... suite

   //détection de caractères
   public void characters(char[] ch, int start, int length)
                          throws SAXException{
      String texte = new String(ch,start,length);
      ...
   }

   //détection début document
   public void startDocument() throws SAXException {
      ...
   }

   //détection fin document
   public void endDocument() throws SAXException {
      ...
   }
}
```

- Utilisation

```
DefaultHandler mongestionnaire = new MonHandler();
parseur.parse(xml, mongestionnaire);
```


Exemple d'application

- Flux XML à insérer dans une BD

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

- Début de document : connexion à la base et début de transaction
- Balise <FILM> : initialisation en mémoire d'un enregistrement *Film*
- Balise <TITRE> : affectation *Film.titre*, …
- Balise </FILM> : insertion de l'enregistrement dans la base
- Fin de document : fin de transaction et déconnexion de la base