Plan I

Plan II

JAXP

- Java API for XML Processing(JAXP)
- Une collection de packages JAVA, intégrés à la JDK depuis sa quatrième version.
- Intérêt : Traitement en Java de documents XML sans recourir à des parseurs externes.

JAXP

JAXP englobe les quatres standards suivants

- SAX, Simple API for XML
- DOM, Document Object Model
- XSLT, Transformations XSLT
- XPath, langage de navigation dans une arbre XML

Manipulations XML en JAVA

- Importer des documents XML (à partir de fichiers)
- Valider des documents à l'encontre de DTD et/ou XMLSchema
- Construction par code de documents XML (en mémoire) puis leur sauvegarde sur des fichiers.
- Parcours de l'arbre XML
- Modification de l'arbre XML.
- Extraction de l'information à l'aide d'XPath, Xquery...
- Transformation de socuments XML via XSLT
- Export et sérialisation dans des fichiers

Import/compilation, export/sérialisation

- Utiliser Document Object Model(DOM) :
  - Norme conçue par le W3C Solution
  - Indépendant du langage de programmation : Java, Javascript, C, Perl, Latex ...
  - Construction rapide d'un arbre de noeuds à partir d'un document textuel et inversement
- Fabriquer son propre compilateur et sérialiseur :
  - SAX Simple API for XML (pas de sérialiseur)
    - Norme conçue par un consortium de constructeur
    - Outils pour construire des compilateurs poussés par des évènements

Import/compilation, export/sérialisation

- Compilation classique (pas de sérialiseur)
  - tirée par un analyseur lexical
  - XMLPull, CyberNeko, StAX
- Génération automatiquement de compilateur et sérialiseur
  - JaxMe (JAXB) (à partir de dtd, schéma, Relax NG)
  - Zeus
  - Castor (à partir de dtd, schéma, classe Java)
  - Eclipse EMF (à partir de modèle de classes, schéma, UML)

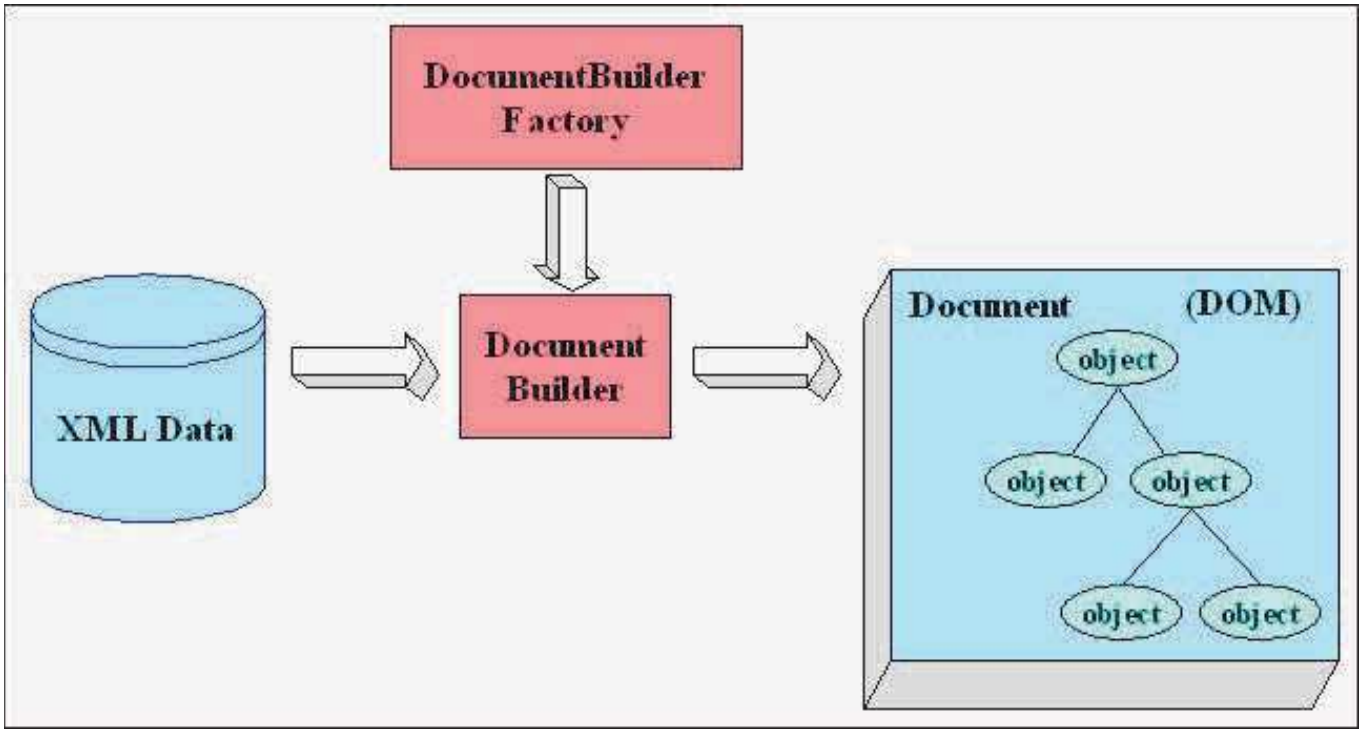
DOM

- Objectif : Compilation facile à mettre en place (approche générique)
- Généricité $\Longrightarrow \searrow$ Rapidité $\nearrow$ empreinte mémoire.
- Construction d'un arbre générique :
  - Parcours laborieux
  - Modifications laborieuses

Document Object Model

- Standard W3C qui fournit un modèle d'arbre générique représentant un document html ou xml par des instances objets
- Modèle indépendant des langages de programmation
- Permet de :
  - parcourir l'arbre du document
  - ajouter, effacer, modifier un sous-arbre
  - sélectionner par un sous-arbre par son contenu

Document Object Model



Compilation

```
1  import javax.xml.parsers.DocumentBuilder;
2  import javax.xml.parsers.DocumentBuilderFactory;
3  import org.w3c.dom.Document;
4
5  try {
6      DocumentBuilderFactory dbfactory = ↙
           ↳ DocumentBuilderFactory.newInstance();
7      DocumentBuilder domparser = dbfactory.newDocumentBuilder();
8      Document doc = domparser.parse(new File("fich.xml"));
9  } catch (ParserConfigurationException e) {
10     e.printStackTrace();
11 }
```

Compilation

Préférences

```
1  setCoalescing(true|false)
2  //Si le parametre est vrai les noeuds CDATA sont effaces.
3
4  setExpandEntityReferences(true|false)
5  //Si le parametre est vrai les entites references sont etendues
6
7  setIgnoringComments(true|false)
8  //Si le parametre est vrai les commentaires sont ignores
9
10 setIgnoringElementContentWhitespace(true|false):
11 //Si le parametre est vrai les espaces non significatifs sont ignores
12 //Decision basee sur un DTD ou un schema XML.
13
14 setNamespaceAware(true|false)
15 //Determine si le compilateur est averti des espaces de noms
16
17 setValidating(true|false)
18 //Par defaut le compilateur ne fait pas de validation
```

Compilation

Sérialisation DOM

```
1  Document doc = ...;
2  Source source = new DOMSource(doc);
3
4  // preparation de la sortie
5  Result result = new StreamResult(new File(nomFichier));
6
7  // ecriture du document DOM sur le fichier
8  Transformer trans = TransformerFactory.newInstance().newTransformer();
9  trans.transform(source, result);
```

Arbre DOM

Hiérarchie des Interfaces

- org.w3c.dom.Node
  - org.w3c.dom.Attr
  - org.w3c.dom.CharacterData
    - org.w3c.dom.Comment
    - org.w3c.dom.Text
  - org.w3c.dom.Document
  - org.w3c.dom.DocumentFragment
  - org.w3c.dom.DocumentType
  - org.w3c.dom.Element
  - org.w3c.dom.Entity
  - org.w3c.dom.EntityReference
  - org.w3c.dom.Notation
  - org.w3c.dom.ProcessingInstruction

Valuer des attributs nodeName & nodeValue

Interface	nodeName	nodeValue
Attr	same as Attr.name	same as Attr.value
CDATASection	"#cdata-section"	same as CharacterData.data, the content of the CDATA Section
Comment	"#comment"	same as CharacterData.data, the content of the comment
Document	"#document"	null
DocumentFragment	"#document-fragment"	null
DocumentType	same as DocumentType.name	null
Element	same as Element.tagName	null
Entity	entity name	null
EntityReference	name of entity referenced	null
Notation	notation name	null
ProcessingInstruction	same as ProcessingInstruction.target	same as ProcessingInstruction.data
Text	"#text"	same as CharacterData.data, the content of the text node

Interface Node

Attributs membres

```
1  static short  ATTRIBUTE_NODE
2  //The node is an Attr.
3  static short  CDATA_SECTION_NODE
4  //The node is a CDATASection.
5  static short  COMMENT_NODE
6  //The node is a Comment.
7  static short  DOCUMENT_FRAGMENT_NODE
8  //The node is a DocumentFragment.
9  static short  DOCUMENT_NODE
10 //The node is a Document.
11 static short  DOCUMENT_POSITION_CONTAINED_BY
12 //The node is contained by the reference node.
13 static short  DOCUMENT_POSITION_CONTAINS
14 //The node contains the reference node.
15 static short  DOCUMENT_POSITION_DISCONNECTED
16 //The two nodes are disconnected.
17 static short  DOCUMENT_POSITION_FOLLOWING
18 //The node follows the reference node.
```

Interface Node

Attributs membres

```
1  static short  DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC
2  //The determination of preceding versus following is ↙
       ↳ implementation-specific.
3  static short  DOCUMENT_POSITION_PRECEDING
4  //The second node precedes the reference node.
5  static short  DOCUMENT_TYPE_NODE
6  //The node is a DocumentType.
7  static short  ELEMENT_NODE
8  //The node is an Element.
9  static short  ENTITY_NODE
10 //The node is an Entity.
11 static short  ENTITY_REFERENCE_NODE
12 //The node is an EntityReference.
13 static short  NOTATION_NODE
14 //The node is a Notation.
15 static short  PROCESSING_INSTRUCTION_NODE
16 //The node is a ProcessingInstruction.
17 static short  TEXT_NODE
18 //The node is a Text node.
```

Interface Node

Méthodes membres

```
1  Node   appendChild(Node newChild)
2  //Adds the node newChild to the end of the list of children of this node.
3  Node   cloneNode(boolean deep)
4  //Returns a duplicate of this node, i.e., serves as a generic copy ↙
       ↳ constructor for nodes.
5  short  compareDocumentPosition(Node other)
6  //Compares the reference node, i.e.
7  NamedNodeMap  getAttributes()
8  //A NamedNodeMap containing the attributes of this node (if it is an ↙
       ↳ Element) or null otherwise.
9  String  getBaseURI()
10 //The absolute base URI of this node or null if the implementation ↙
       ↳ wasn't able to obtain an absolute URI.
11 NodeList     getChildNodes()
12 //A NodeList that contains all children of this node.
13 Object  getFeature(String feature, String version)
14 /*This method returns a specialized object which implements the ↙
       ↳ specialized APIs of the specified feature and version, as ↙
       ↳ specified in .*/
15 Node   getFirstChild()
16 //The first child of this node.
```

Interface Node

Méthodes membres

```
1  Node   getLastChild()
2  //The last child of this node.
3  String  getLocalName()
4  //Returns the local part of the qualified name of this node.
5  String  getNamespaceURI()
6  //The namespace URI of this node, or null if it is unspecified (see ).
7  Node   getNextSibling()
8  //The node immediately following this node.
9  String  getNodeName()
10 //The name of this node, depending on its type; see the table above.
11 short  getNodeType()
12 //A code representing the type of the underlying object, as defined ↙
       ↳ above.
13 String  getNodeValue()
14 //The value of this node, depending on its type; see the table above.
15 Document     getOwnerDocument()
16 //The Document object associated with this node.
17 Node   getParentNode()
18 //The parent of this node.
19 String  getPrefix()
```

Interface Node

Méthodes membres

```
1  Node   getPreviousSibling()
2  //The node immediately preceding this node.
3  String  getTextContent()
4  //This attribute returns the text content of this node and its ↙
       ↳ descendants.
5  Object  getUserData(String key)
6  //Retrieves the object associated to a key on a this node.
7  boolean       hasAttributes()
8  //Returns whether this node (if it is an element) has any attributes.
9  boolean       hasChildNodes()
10 //Returns whether this node has any children.
11 Node   insertBefore(Node newChild, Node refChild)
12 //Inserts the node newChild before the existing child node refChild.
13 boolean       isDefaultNamespace(String namespaceURI)
14 //This method checks if the specified namespaceURI is the default ↙
       ↳ namespace or not.
15 boolean       isEqualNode(Node arg)
16 //Tests whether two nodes are equal.
17 boolean       isSameNode(Node other)
18 //Returns whether this node is the same node as the given one.
```

Interface Node

Méthodes membres

```
1  boolean      isSupported(String feature, String version)
2  /*Tests whether the DOM implementation implements a specific feature ↙
       ↳ and that feature is supported by this node, as specified in .*/
3  String  lookupNamespaceURI(String prefix)
4  //Look up the namespace URI associated to the given prefix, starting ↙
       ↳ from this node.
5  String  lookupPrefix(String namespaceURI)
6  //Look up the prefix associated to the given namespace URI, starting ↙
       ↳ from this node.
7  void   normalize()
8  /*Puts all Text nodes in the full depth of the sub-tree underneath ↙
       ↳ this Node, including attribute nodes, into a "normal" form ↙
       ↳ where only structure (e.g., elements, comments, processing ↙
       ↳ instructions, CDATA sections, and entity references) separates ↙
       ↳ Text nodes, i.e., there are neither adjacent Text nodes nor ↙
       ↳ empty Text nodes.*/
9  Node   removeChild(Node oldChild)
10 //Removes the child node indicated by oldChild from the list of ↙
       ↳ children, and returns it.
11 Node   replaceChild(Node newChild, Node oldChild)
12 //Replaces the child node oldChild with newChild in the list of ↙
       ↳ children, and returns the oldChild node.
```

Interface Node

Méthodes membres

```
1  void   setNodeValue(String nodeValue)
2  //The value of this node, depending on its type; see the table above.
3  void   setPrefix(String prefix)
4  //The namespace prefix of this node, or null if it is unspecified.
5  void   setTextContent(String textContent)
6  //This attribute returns the text content of this node and its ↙
       ↳ descendants.
7  Object  setUserData(String key, Object data, UserDataHandler handler)
8  //Associate an object to a key on this node.
```

Exemple

fich.xml

```
1  <adresse>
2    <ligne>ENSAO</ligne>
3    <ligne>Univ Mohammed Premier</ligne>
4    <voie numero="1">Campus universitaire</voie>
5    <ligne>1225262</ligne>
6    <ville codePostal="6050">Oujda</ville>
7    <ligne>gh122522</ligne>
8  </adresse>
```

Exemple

Parcourir les fils

```
1  Element racine = doc.getDocumentElement();
2  NodeList enfants = racine.getChildNodes();
3  int x=enfants.getLength()
4  //Nombres de fils de l'element racine
5  //Resultat : 13
```

Contenu textuel de l'element voie :

```
1  String s1=enfants.item(2).getFirstChild().getNodeValue()
2  String ↙
       ↳ s2=racine.getElementsByTagName("voie").item(0).getFirstChild().getNodeValu
3  //Resultat : Campus universitaire
```

Exemple

Parcourir les fils

Valeur de l'attribut codePostal de l'élément ville

```
1  Element racine = doc.getDocumentElement();
2  NodeList enfants = racine.getChildNodes();
3  String ↙
       ↳ s=enfants.item(4).getAttributes().getNamedItem("codePostal").getNodeValue(
```

Créer un Arbre vide

```
1  try {
2     DocumentBuilder builder ↙
          ↳ =DocumentBuilderFactory.newInstance().newDocumentBuilder();
3     Document doc = builder.newDocument();
4  }
5  catch (ParserConfigurationException e) {
6     ...
7  }
```

Créer un contenu de document

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

Créer un contenu de document (suite)

```
1  Document doc = builder.newDocument();
2    // creer l'element racine
3  Element root = doc.createElement("racine");
4    // creer un noeud texte
5  Text text = doc.createTextNode("ceci_est_le_contenu");
6    // mettre les noeuds dans l'arbre DOM du document
7  root.appendChild(text);
8  doc.appendChild(root);
```

Résultat : <racine>ceci est le contenu</racine>

```
1      // recherche de l'element nom
2  Element racine=doc.getDocumentElement();
3  Node nom=racine.getElementsByTagName("nom").item(0);
4      // creation d'un element prenom
5  Element prenom=doc.createElement("prenom");
6  prenom.appendChild(doc.createTextNode("ahmed"));
7     // rattachement de l'element prenom devant l'element nom
8  racine.insertBefore(prenom,nom);
```

```
1  <personne>
2    <nom>mohammed</nom>
3    <age>28</age>
4    <taille>1.80</taille>
5  </personne>
```

```
1  <personne>
2    <prenom>ahmed</prenom>
3    <nom>mohammed</nom>
4    <age>28</age>
5    <taille>1.80</taille>
6  </personne>
```

Autres opérations

- Traverser les nœuds d'un arbre DOM
  - L'interface org.w3c.dom.Node définit des méthodes pour traverser les nœuds et se déplacer n'importe où dans l'arbre

```
1    getFirstChild()
2    getLastChild()
3    getNextSibling()
4    getPreviousSibling()
5    getParentNode()
```

- Créer des attributs
  - L'interface org.w3c.dom.Element qui étend l'interface Node définit un ensemble de méthodes pour manipuler les attributs d'un élément

```
1    setAttribute()
2    getAttributeNode()
3    hasAttribute()
```

Autres opérations (Suite)

- Modifier un nœud d'arbre DOM
  - L'interface org.w3c.dom.Node fournit des méthodes pour enlever, remplacer, modifier un nœud

```
1 removeChild()
2 replaceChild()
3 setNodeValue()
```

Dom, et les autres

- DOM est indépendant des langages
  - Contenus, chaînes, collections sont modélisés par des constructions spécifiques (n'utilisent pas directement les constructions du langage)
- JDOM, DOM4J, XOM adaptés à Java
  - Plus simple d'emploi
  - moins couteux
- Contenus, chaînes, collections sont directement des constructions standards du langage
- Construction :JDom, XOM sont basés sur la notion de classes Dom, Dom4J sont basés sur la notion d'interface

Simple API for XML (SAX)

- Modèle évènementiel, pour concevoir des compilateurs XML spécifiques
- Un consortium de fabriquants : Adopté par Sun dans Java 1.4 sous le nom Jaxp 1.0
- Ne produit pas d'image mémoire du document lu séquence d'événements à interpréter

Séquence d'évènements

Six types d'évènements :

- début/fin de document
- début/fin d'éléments
- caractères
- caractères ignorables

Compiler un document avec SAX

les objets

```
1  import org.xml.sax.Attributes;
2  import org.xml.sax.InputSource;
3  import org.xml.sax.SAXException;
4  import org.xml.sax.XMLReader;
5  import org.xml.sax.helpers.DefaultHandler;
6  import org.xml.sax.helpers.XMLReaderFactory;
```

Créer un compilateur SAX

```
1  MySAXHandler handler = new MySAXHandler();
2  XMLReader saxParser = XMLReaderFactory.createXMLReader();
3  saxParser.setContentHandler(handler);
4  saxParser.parse(new InputSource(new FileReader("sample")));
```

Utiliser SAX en deux étapes

- Deux approches
  - Sous-classer :

```
1  org.xml.sax.helpers.DefaultHandler
```

  - Implémenter les interfaces

```
1  DocumentHandler
2  DTDHandler
3  EntityResolver
4  ErrorHandler
```

- Lancer la compilation

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

Configuration

- Créer une fabrique de parseurs :

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

- Configurer une fabrique de parseurs :

```
1  spf.setNamespaceAware(boolean awareness)
2  //Pour construire des analyseurs acceptant les espaces de noms
3
4  spf.setValidating(boolean validating)
5  //Pour construire des analyseurs validant (DTD)
6
7  spf.setFeature(String name, boolean value)
8  //Pour les autres proprietes specifiques a un vendeur
```

Exemple

Lines 1–21 / 68

```
1  import org.xml.sax.*;
2  import org.xml.sax.helpers.*;
3  import java.io.*;
4  public class TestHandler extends DefaultHandler {
5        int nbElem = 0; // nombre d'elements rencontres
6        int niv = 0; // niveau de profondeur courant
7
8        public TestHandler() {
9              super();
10       }
11
12       public void startDocument() {
13             System.out.println("debut de document");
14       }
15
16       public void endDocument() {
17         System.out.println("fin de document : " + nbElem + " elements ↙
                ↳ rencontres");
18       }
19
20       public void startElement(String uri, String name,
21     String qName, Attributes atts) {
```

Exemple

Lines 20–40 / 68

```
1          public void startElement(String uri, String name,
2        String qName, Attributes atts) {
3     if("".equals(uri)) {
4        System.out.println(indent(niv)+qName+ "{");
5     } else {
6        System.out.println(indent(niv)+"{"+uri+"}"+name+ "{");
7     }
8     nbElem++;
9     niv++;
10 }
11
12 public void endElement(String uri, String name, String qName) {
13    niv--;
14    if("".equals(uri)) {
15       System.out.println(indent(niv)+"}"+qName);
16    } else {
17       System.out.println(indent(niv)+"}"+"{"+uri+"}"+name);
18    }
19 }
20 public static String indent (int i) {
21    StringBuffer s = new StringBuffer("");
```

Exemple

Lines 39–59 / 68

```
1  public static String indent (int i) {
2     StringBuffer s = new StringBuffer("");
3     for ( int j=0; j<i; j++) {
4        s.append("␣␣␣");
5     }
6     return s.toString();
7  }
8
9  public void characters(char[] ch, int start, int lenght) {
10                System.out.print(indent(niv));
11                for (int i = start; i<start+lenght; i++) {
12                      switch (ch[i]) {
13                      case '\\':
14                            System.out.print("\\\\");
15                            break;
16                      case '"':
17                            System.out.print("\\\"");
18                            break;
19                      //...
20                      default:
21                            System.out.print(ch[i]);
```

Exemple

Lines 58–78 / 68

```
1                       default:
2                             System.out.print(ch[i]);
3                             break;
4                       }
5                 }
6                 System.out.println("");
7           }
8           public static void main(String[] args){
9           try{XMLReader xmlReader = XMLReaderFactory.createXMLReader();
10          TestHandler handlerTest = new TestHandler();
11          xmlReader.setContentHandler(handlerTest);
12
13          FileReader reader = new FileReader("myFile");
14          xmlReader.parse(new org.xml.sax.InputSource("myFile"));
15          }catch(Exception e){;}
16          }
17    }
```

JDOM et XSLT

Lines 1–21 / 68

```
1  import java.io.*;
2  import org.jdom.transform.*;
3  import org.jdom.output.*;
4  import javax.xml.transform.*;
5  import javax.xml.transform.stream.StreamSource;
6  void outputXSLT(org.jdom.Document documentJDOMEntree ,String fichierXSL){
7          JDOMResult documentJDOMSortie = new JDOMResult();
8          org.jdom.Document resultat = null;
9          try{
10                 TransformerFactory factory = ↙
                       ↳ TransformerFactory.newInstance();
11                 Transformer transformer = factory.newTransformer(new ↙
                       ↳ StreamSource(fichierXSL));
12                 transformer.transform(new ↙
                       ↳ org.jdom.transform.JDOMSource(documentJDOMEntree ↙
                       ↳ ), documentJDOMSortie);
13                 XMLOutputter outputter = new ↙
                       ↳ XMLOutputter(Format.getPrettyFormat());
14         outputter.output(resultat, new FileOutputStream("resultat.xml"));
15         }
16         catch(Exception e){}
17 }
```