

Traitements des données

JAXP : Java et XML



JAXP : Java API for XML Processing

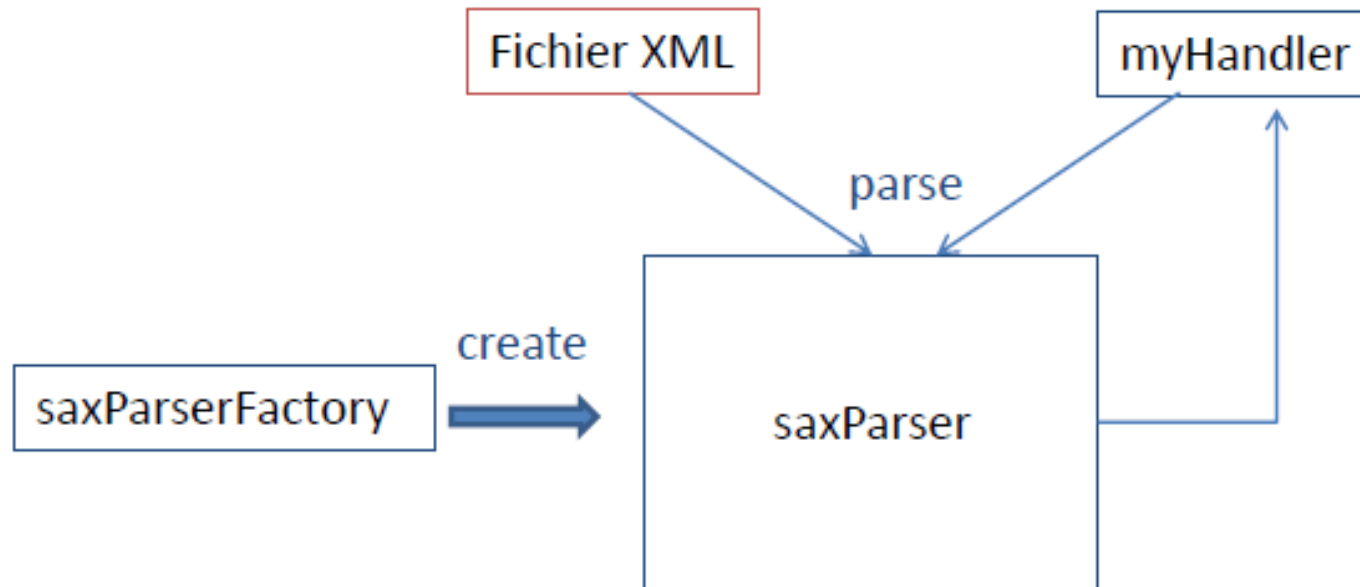
- Interface de programmation Java de Sun permettant la création, la manipulation et le traitement de fichiers XML à bas niveau.
- Cette interface contient trois parties
 1. SAX : programmation événementielle sur le parcours d'un arbre XML
 2. DOM : Création , modification et sauvegarde d'arbres XML
 3. XSLT : Transformation de fichiers XML à l'aide de feuilles XSL



Présentation de SAX

- SAX : Simple API for XML
- Site projet : <http://www.saxproject.org>
- Inclus JDK de Sun :
 - org.xml.sax
 - javax.xml.parsers
- Programmation événementielle sur le parcours du document XML suivant le type de fragment rencontré (élément, texte, commentaire, ...) => parser+ handlerSAX

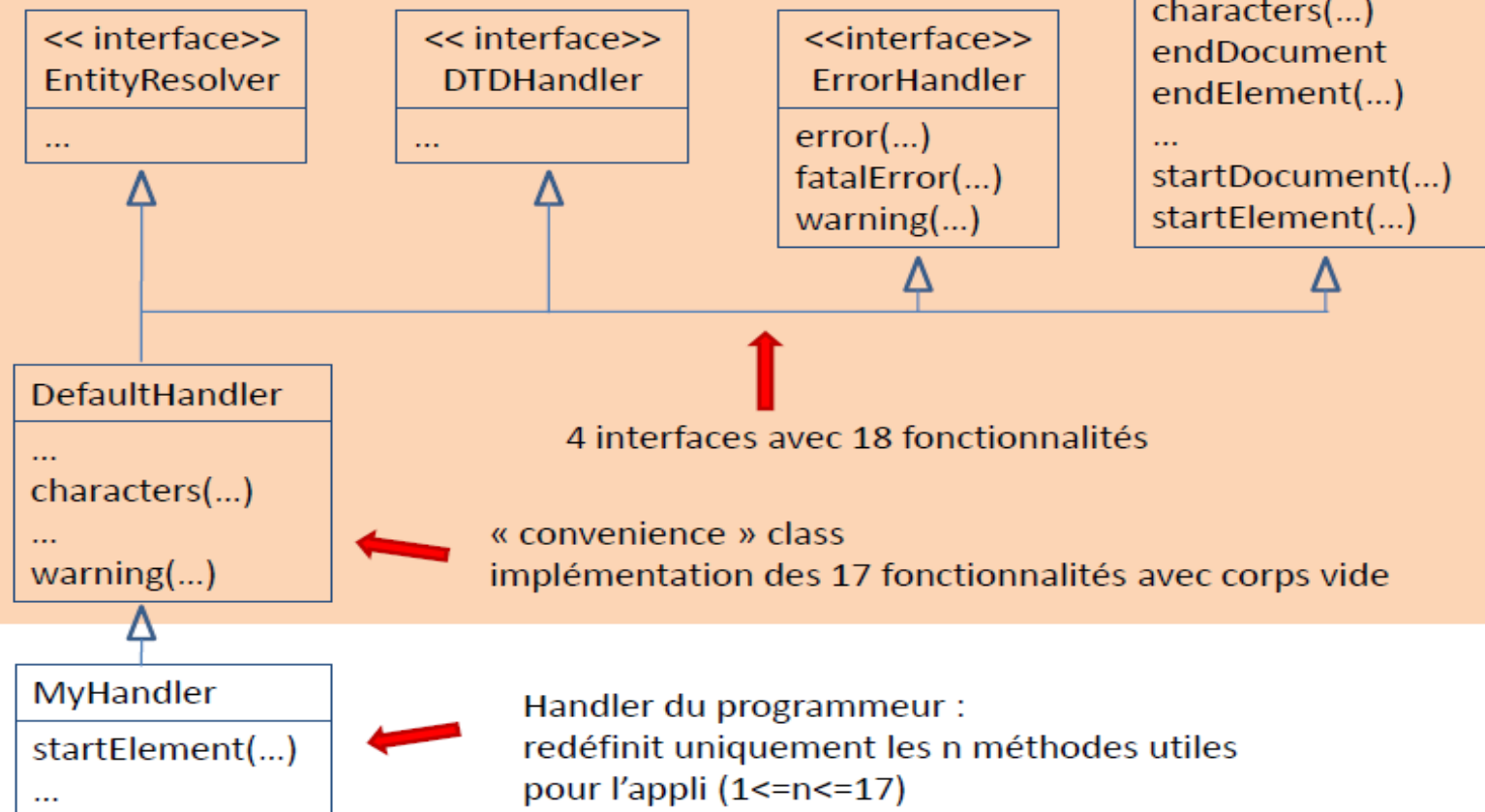
SAX : Principe général



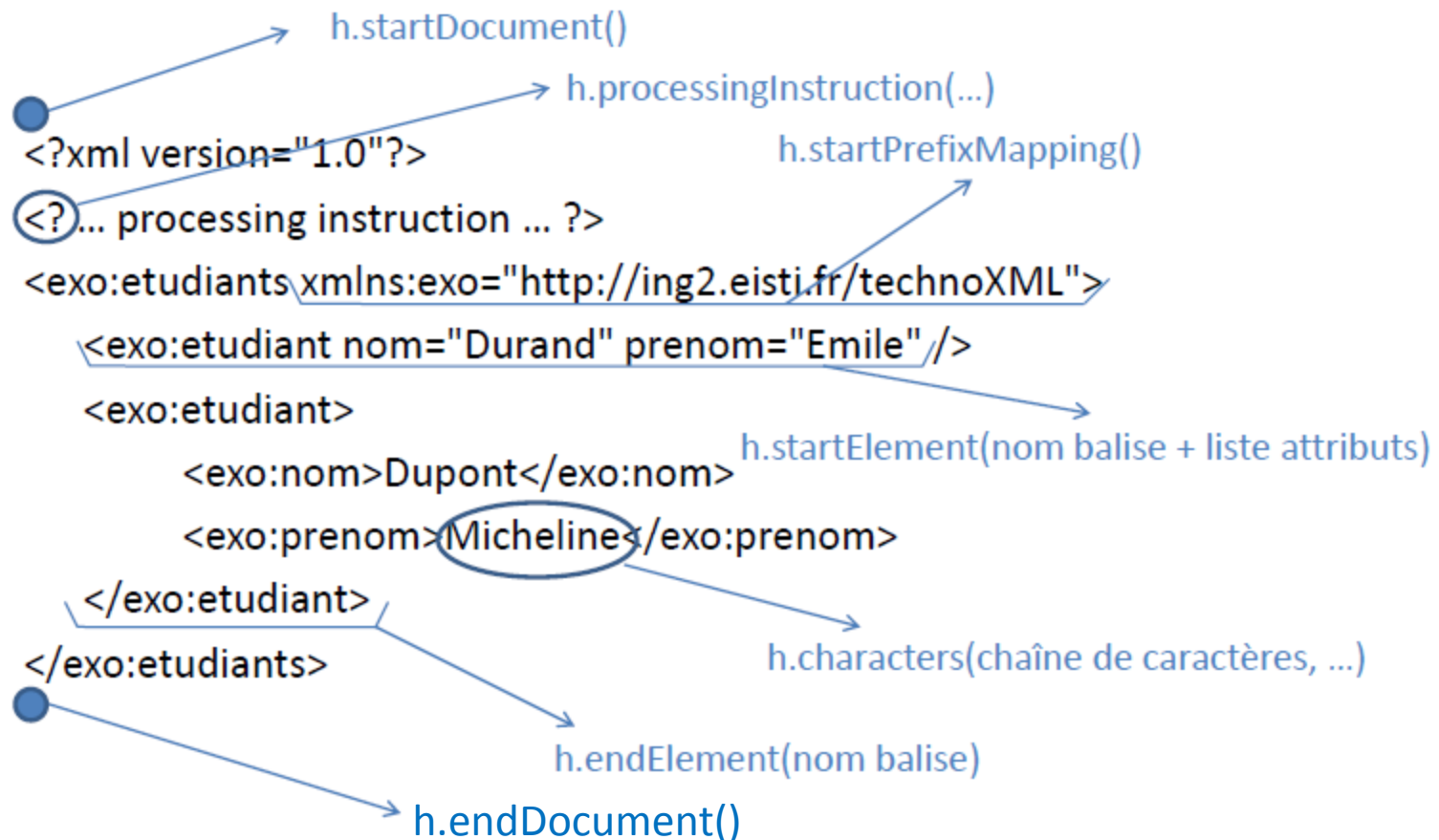
- classes SAXParser et SAXParserFactory (package javax.xml.parsers)
- fichier XML + handler fournis à la méthode parse
- Le parser appelle les méthodes appropriées du handler

SAX : Interfaces du Handler

Interfaces pour Handler SAX



Exemple





Validation XSD avec DOM et SAX

- XSD plus récent que DOM/SAX
- XSD pris en compte dans JAXP
- Combiner validation avec DOM/SAX :
 - attributs pour parserDOM : `setAttribute`
 - propriétés pour parserSAX : `setProperty`

nom	Valeur
JAXP_SCHEMA_LANGUAGE	W3C_XML_SCHEMA
JAXP_SCHEMA_SOURCE	Fichier XSD (objet File)

- Ajout Handler pour traiter les erreurs détectées :
 - implémenter obligatoirement `ErrorHandler`
 - prévu dans SAX
 - ajout pour DOM : `setErrorHandler`



Validation XSD avec SAX : Code Java

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

public MaClasse {
    static final String JAXP_SCHEMA_LANGUAGE = "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
    static final String W3C_XML_SCHEMA = "http://www.w3.org/2001/XMLSchema";
    static final String JAXP_SCHEMA_SOURCE = "http://java.sun.com/xml/jaxp/properties/schemaSource";

    public static void main (String args ) {
        DefaultHandler handler = new MyContentHandler(); // Classe d'évts définies par ailleurs
        SAXParserFactory factory = SAXParserFactory.newInstance();
        factory.setNamespaceAware(true); // les parseurs produits supportent les espaces de nommage
        factory.setValidating(true); // true : validation du XML par une DTD ou un XSD
        try
        {
            SAXParser saxParser = factory.newSAXParser();
            saxParser.setProperty(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
            saxParser.setProperty(JAXP_SCHEMA_SOURCE, "???"); // ??? Le chemin d'accès au fichier xsd
            saxParser.parse( new File(???), handler); // ??? Le chemin d'accès au fichier xml
        }
    }
}
```



L'interface ErrorHandler

- Parsage d'un document XML
 - Document XML mal formé
 - Document XML ne respectant pas une DTD ou un XSD
- Parsage d'un document XML avec SAX
 - **Erreur fatale** (*fatalError(SAXParseException exception) throws SAXException*) : Le parseur s'arrête (Ex : XML mal formé)
 - **Erreur** (*error(SAXParseException exception) throws SAXException*) : Le parseur peut continuer son travail (Ex : non respect DTD ou XSD)
 - **Warning** (*warning(SAXParseException exception) throws SAXException*) : Le parseur SAX continue le parsing du document avec un simple avertissement.



DOM : Document Object Model

- DOM est une normalisation du consortium W3C :
<http://www.w3.org>
- DOM est implémenté dans de nombreux langages informatiques dont Java et Javascript.
- Dans le langage Java, le package org.w3c.org est inclus dans le JDK (il existe également une autre version opensource JDom)
- DOM est un ensemble d'interfaces qui permettent de :
 - représenter sous la forme d'un arbre un fichier XML
 - naviguer dans l'arbre
 - faire une recherche dans un arbre
 - modifier un arbre
 - sauvegarder un arbre XML



package org.w3c.dom



Interface	Partie de l'arbre
interface Document	Arbre XML
interface Element	Élément XML
interface Attr	Attribut d'un élément XML
interface Text	Texte d'un élément XML
interface Comment	Commentaire XML



Correspondance XML et Interface DOM

<Bibliographie>

<Livre titre="Modeling XML Applications with UML "année="2001">

<Auteur>David Carlson</Auteur>

</Livre>

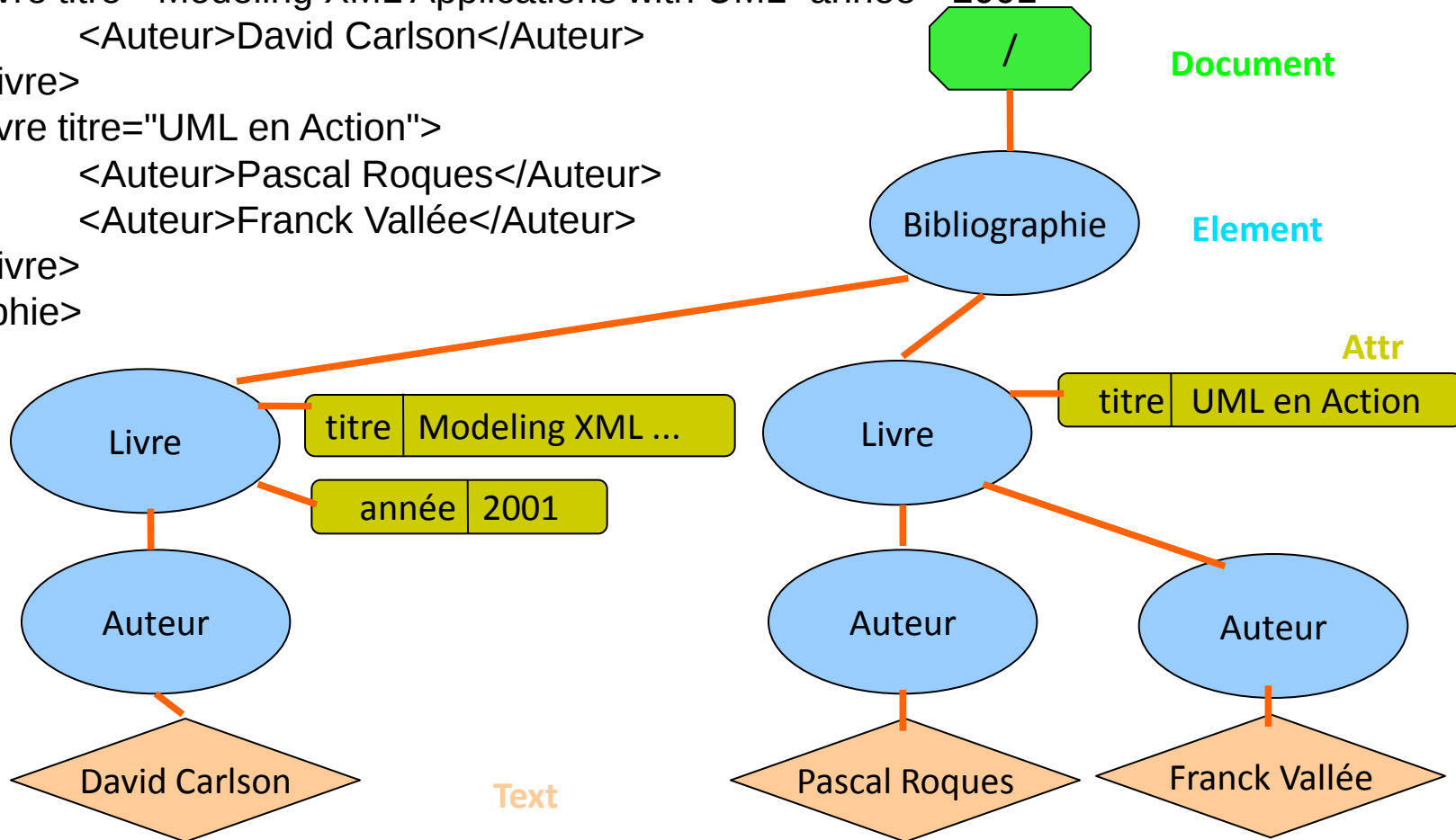
<Livre titre="UML en Action">

<Auteur>Pascal Roques</Auteur>

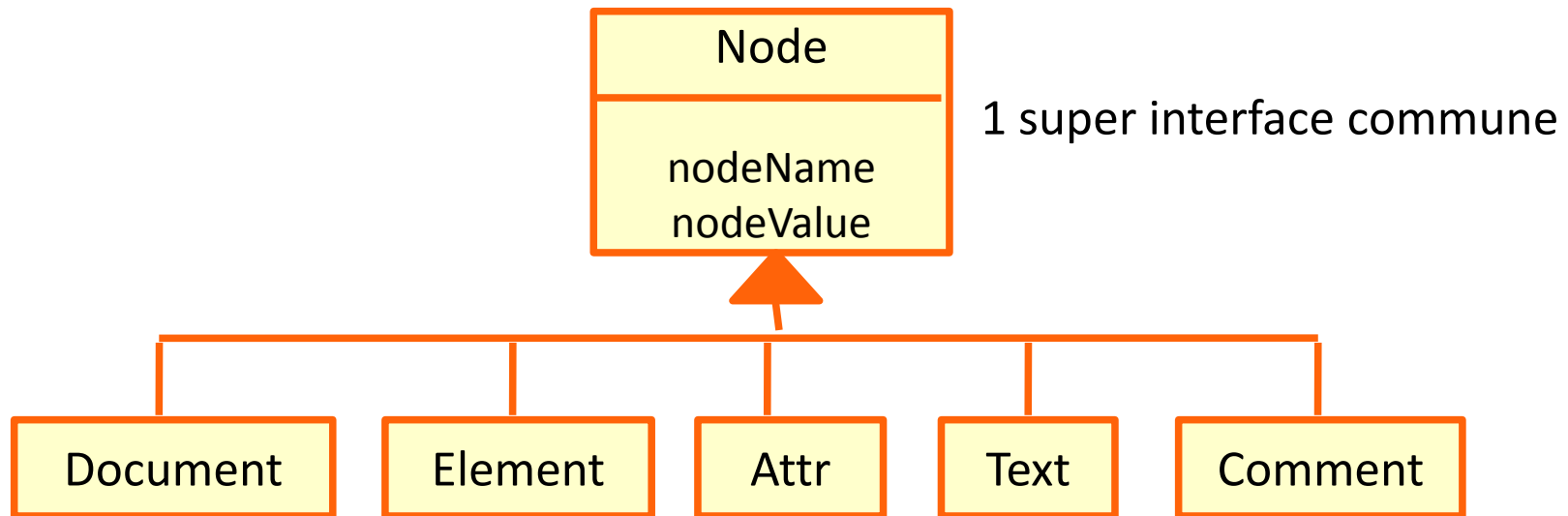
<Auteur>Franck Vallée</Auteur>

</Livre>

</Bibliographie>



DOM : Interfaces et attributs



	nodeName	nodeValue
Document	"#document"	null
Element	nom de balise	null
Attr	nom attribut	valeur attribut
Text	"#text"	texte
Comment	"#comment"	commentaire

DOM : Chargement d'un fichier XML

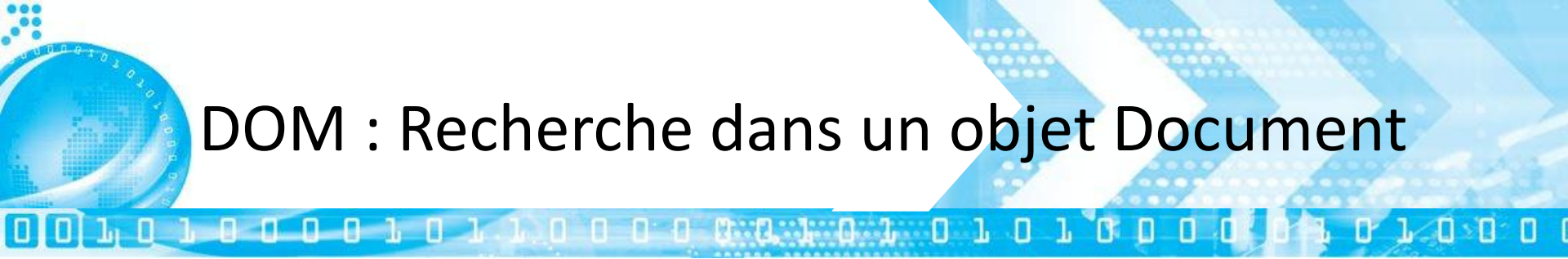


```
import org.w3c.dom.*;
import javax.xml.parsers.*; // DocumentBuilderFactory, DocumentFactory
public void chargerXML() {
    try {
        // création d'une fabrique de documents, fabrique de constructeurs
        DocumentBuilderFactory fabrique = DocumentBuilderFactory.newInstance();
        // création d'un constructeur de documents
        DocumentBuilder parser = fabrique.newDocumentBuilder();
        // lecture du contenu d'un fichier XML avec DOM
        File fdom = new File("???"); // ??? chemin du fichier xml
        Document dom = parser.parse(fdom);
    }
    catch(Exception e) { }
```



DOM : Navigation dans un objet Document

- Pour un document :
 - `getDocumentElement()` : élément racine de l'arbre
- Pour un élément :
 - `getChildNodes()` : liste des fils d'un élément de type
 - Element
 - Text
 - Comment
 - `getAttributes()` : liste des attributs d'un élément
- Pour un noeud quelconque :
 - `getNodeName()` : son nom (cf tableau)
 - `getNodeValue()` : sa valeur (cf tableau)
 - `getNodeType()` : son type (ELEMENT_NODE, TEXT_NODE, etc ...)



DOM : Recherche dans un objet Document

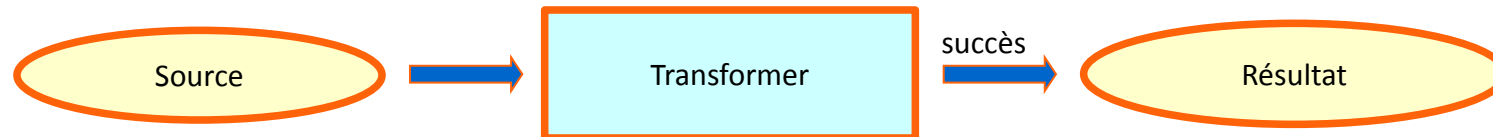
- A partir d'un document
 - `getElementById` : obtenir un élément par la valeur de son attribut ID unique
 - `getElementsByTagName` : obtenir une liste d'éléments par leur nom
 - `getElementsByTagNameNS` : obtenir une liste d'éléments par leur nom qualifié par un espace de nom (ex: `xs:stylesheet`)
- A partir d'un élément
 - `getElementsByTagName` / `getElementsByTagNameNS` : recherche uniquement à partir de cet élément
 - `getAttribute` / `getAttributeNS` : obtenir la valeur d'un attribut à partir de son nom
 - `getAttributeNode` / `getAttributeNodeNS` : obtenir un attribut à partir de son nom



DOM : Modification d'un objet Document

- Pour un noeud quelconque :
 - Remplacer la valeur du noeud : `setNodeValue(nouvelleValeur)`
- Pour un élément :
 - Supprimer un fils : `removeChild(fils)`
 - Remplacer un fils : `replaceChild(nouveauFils, ancienFils)`
 - Ajouter un attribut / Remplacer la valeur d'un attribut : `setAttribute(nomAttribut, valeurAttribut)`
 - Supprimer un attribut : `removeAttribute(nomAttribut)`
- Pour un document :
 - `createElement` / `createTextNode` / `createComment` => un noeud est toujours créé dans le contexte de son document, puis rattaché au bon endroit dans l'arbre avec les méthodes ci-dessus

JAXP : Transformation

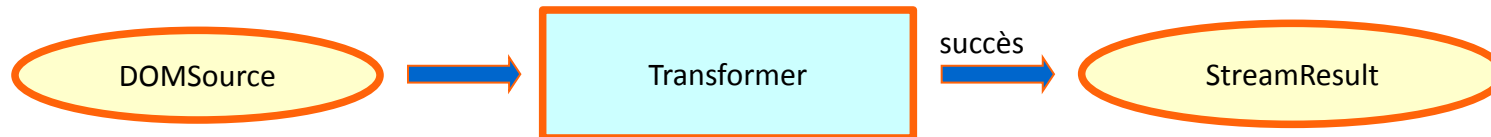


Type de source
DomSource
SAXSource
StAXSource
StreamSource

Type de résultat
DomResult
SAXResult
StAXResult
StreamResult

- Le choix de type de la source et celui du résultat sont indépendants
- On peut travailler sur la source et sur le résultat pour travailler en mémoire vive ou sur fichier
- Un transformer par défaut fait de la copie
- Dans ce qui suit, on montre deux transformations :
 - Une sauvegarde d'un objet DOM dans un fichier XML
 - Deux versions de transformation XSLT : v1.0 et v2.0

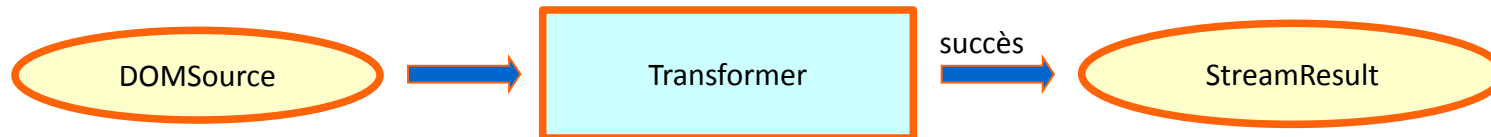
JAXP : Sauvegarde d'un objet DOM



```
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import javax.xml.transform.*;

public void SauverArbre(String fileName) {
    try {
        TransformerFactory tf = TransformerFactory.newInstance(); // fabrique de transformateur
        Transformer transformer = tf.newTransformer(); // creation d'un transformateur
        // Réglages du transformateur
        transformer.setOutputProperty(OutputKeys.METHOD, "xml");
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "no");
        // Sauvegarde de l'objet Document
        File f = new File(???); // ??? Chemin du fichier
        if (f.exists()) { f.delete(); }
        StreamResult nvFichier = new StreamResult(f);
        transformer.transform(new DOMSource(d), nvFichier);
    }
    catch(Exception e) { }
```

JAXP : Transformation XSLT



```
import javax.xml.transform.dom.*;  
import javax.xml.transform.stream.*;  
import javax.xml.transform.*;
```

```
Document dom; // source en mémoire  
String resultFilename; // résultat sur fichier  
String xslFilename; // feuille XSL sur fichier  
DOMSource source = new DOMSource(dom);  
StreamResult result = new StreamResult(new File(resultFilename));  
TransformerFactory tf = TransformerFactory.newInstance(); // XSLT de JAXP (v1.0) (*)  
Transformer transformer = tf.newTransformer(new StreamSource(new File(xslFilename)));  
transformer.transform(source, result);
```

(*) Pour faire une transformation XSLT 2.0, il faut :

- mettre saxon9he.jar dans votre projet (dans le classpath)
- TransformerFactory tf = new net.sf.saxon.TransformerFactoryImpl(); // XSLT de JAXP (v2.0)