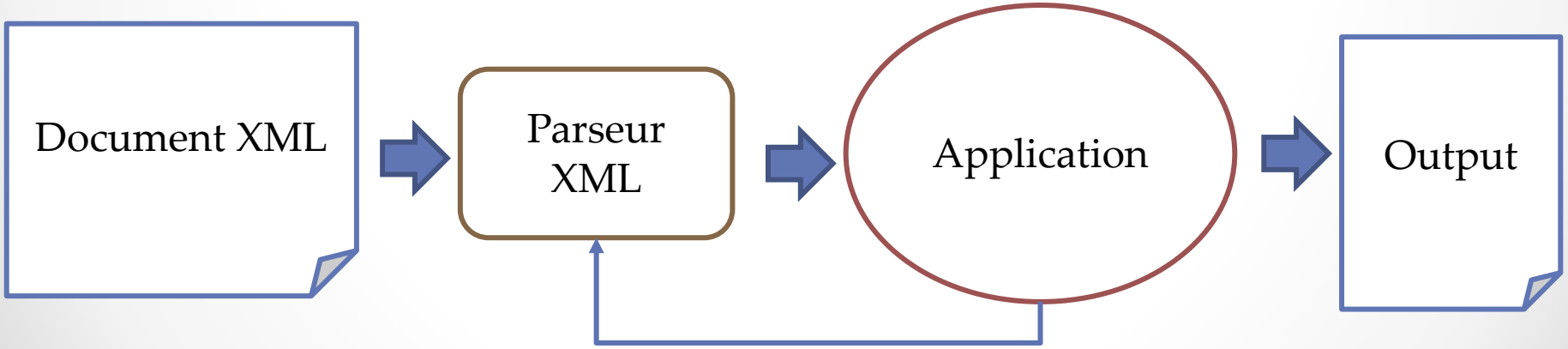
XML: DOM & SAX

Parsing des documents XML

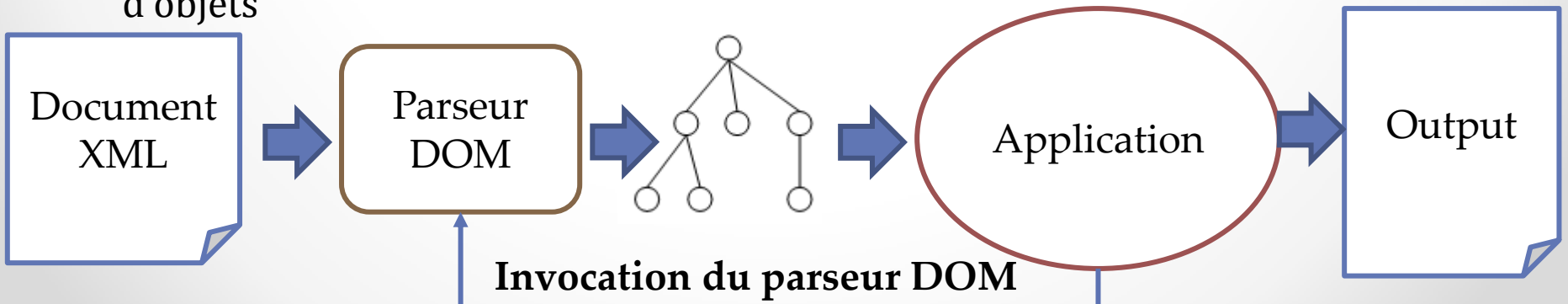
- **Objectif:** Analyse du document XML.
- **Rôle:** Vérifier la cohérence du document XML et de transmettre à l'application les informations utiles au traitement du document



APIs pour le Parsing des documents XML

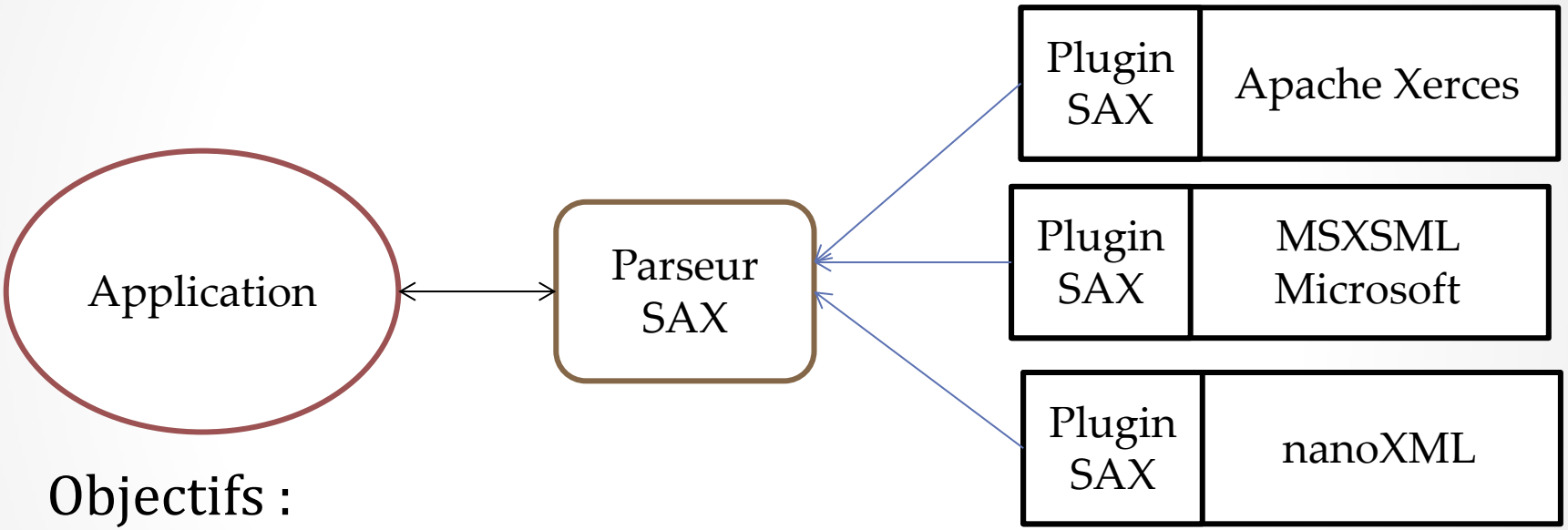
- **SAX (Simple API for XML) :** permettant la manipulation d'un document XML au fur et à mesure de la lecture de celui-ci. L'extraction des données est basée sur une gestion d'événements (création d'un document, élément, etc,).

- **DOM XML (Document Object Model):** permettant la manipulation d'un document XML après l'avoir représenté en mémoire sous la forme d'un arbre d'objets



1. API SAX (Simple API for XML)

- API événementielle pour l'analyse d'un document XML

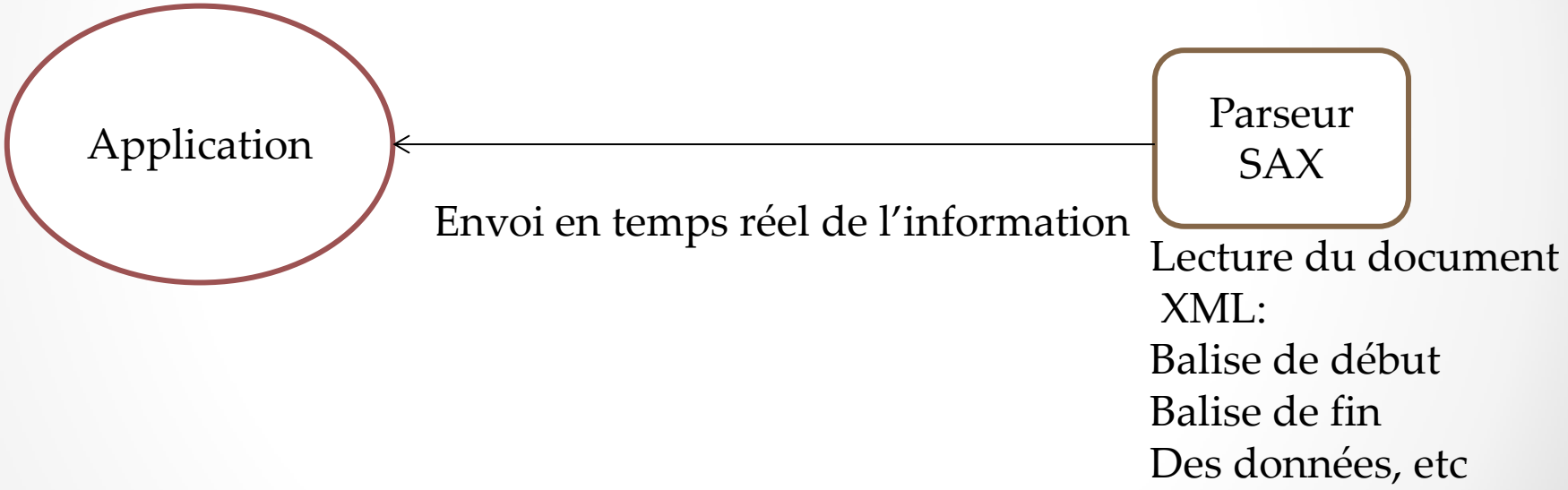


- Objectifs :
  - Simplifier l'accès aux analyseurs,
  - Rendre l'application indépendante des analyseurs,

- L'analyse génère des évènements (début et fin de document ; début et fin d'éléments, attributs, texte, ...) récupérés par SAX et passés à l'application.

Référence: http://www.saxproject.org

- Modèle événementiel et incrémental
- Parcours du fichier caractère par caractère



- Déclenche un événement et la procédure appelée est qualifiée de « callback »

- **Exemple**

Voici comment SAX lirait notre document XML « livre.xml »

1 - événement levé lors du début de lecture du document ;

2 - événement lors de la rencontre avec la balise <livre> ;

3 - événement lors de la rencontre avec la balise <auteurs> ;

4 - événement lors de la rencontre avec la balise <auteur> ;

5. événement lors de la rencontre avec la balise </auteur> ;

6 - ...

7 - événement lors de la rencontre avec la balise <paragraphe> ;

8. événement lors de la rencontre avec le contenu textuel de la balise en cours de lecture ;

9. événement lors de la rencontre avec la balise </paragraphe> ;

...

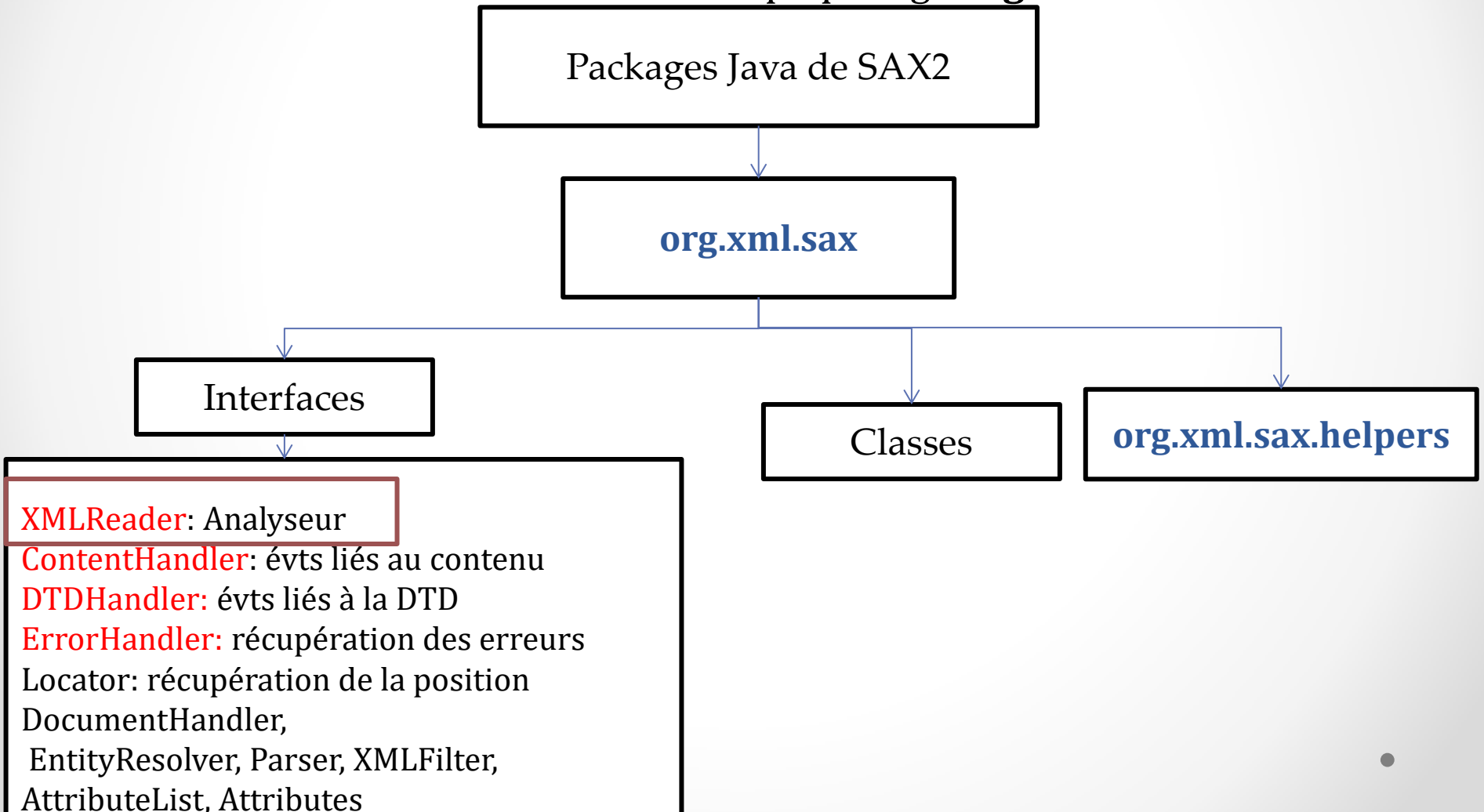
10 . événement levé lors de la fin du document.

Comment?

Un gestionnaire d'événements pour les intercepter

API SAX (Simple API for XML)

- SAX : une collection d'interfaces du paquetage **org.xml.sax**



Comment créer un analyseur?

1. Création d'un objet parseur

```
XMLReader parser = XMLReaderFactory.createXMLReader();
```

2. Un **gestionnaire d'événements** qui doit implémenter plusieurs interfaces (une extension de la classe **DefaultHandler**).

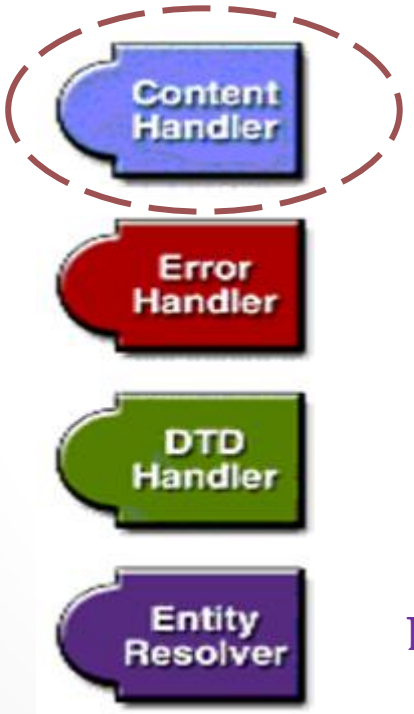
```
// // fixer le(s) gestionnaire(s) d'évènements
parser.setContentHandler(this); //affecter le handler au parseur grâce à setContentHandler
 parser.setErrorHandler(this);
// fixer les comportements pour être validé par SAX
 parser.setFeature("http://xml.org/sax/features/validation", true);
```

3.Lancement du processus

```
// analyser la source XML
 parser.parse(new InputSource(fileName));
```

Composants de SAX

Interfaces à implémenter



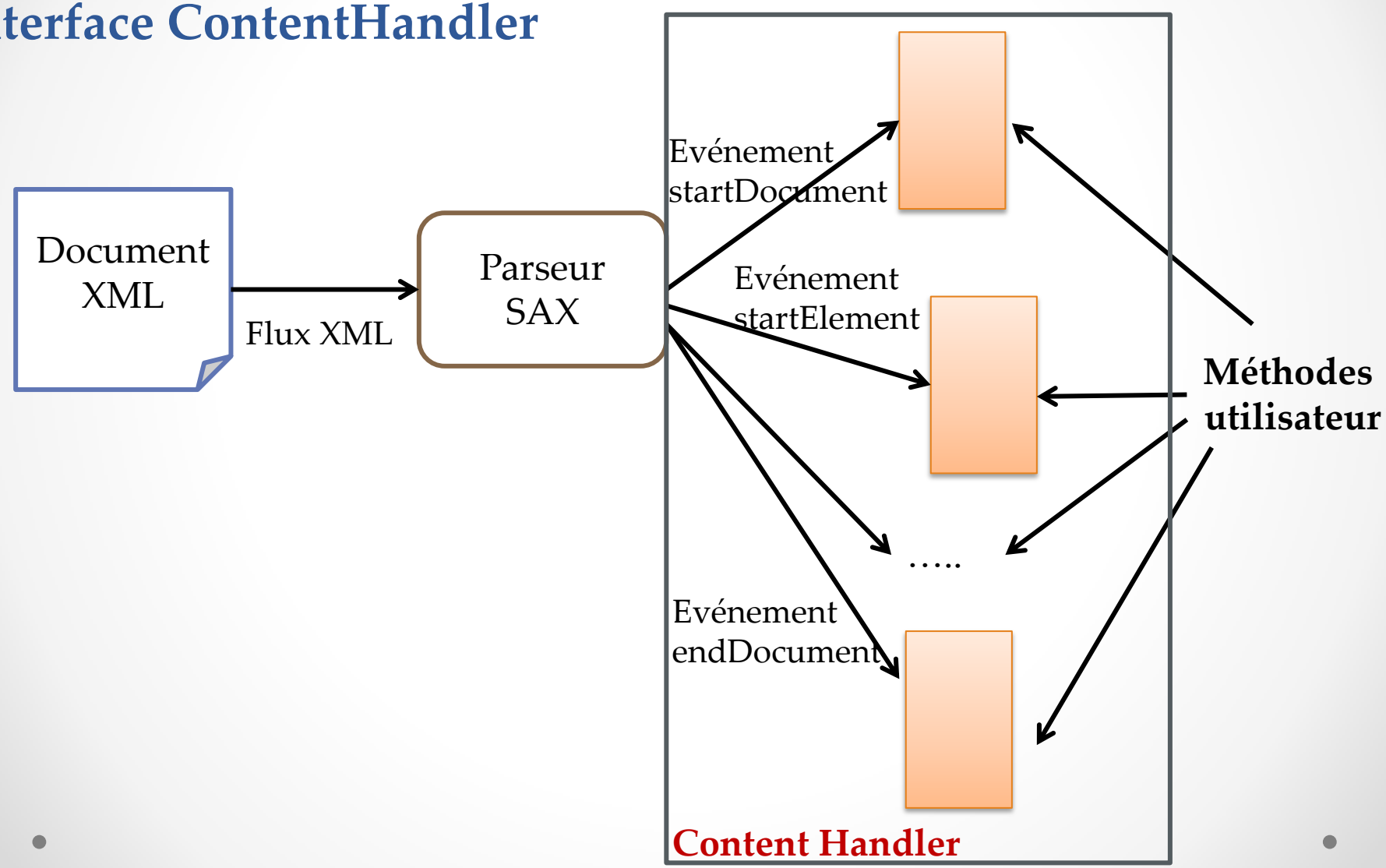
Les événements liés au contenu

Récupération des erreurs

Evénements liés à la DTD

Evénements liés aux entités externes

Interface ContentHandler



Interface ContentHandler

C'est l'interface principale qui est implémentée par la plupart des applications utilisant SAX.

- Elle comporte plusieurs méthodes permettant de notifier:

Description	Méthode
Début d'un document	void startDocument()
Fin d'un document	void endDocument()
Début d'un élément	void **startElement** (String uri, String localName, String qName, Attributes atts)
Fin d'un élément	**void endElement(String uri, String localName, String qName)**
Les données	void **characters**(char[] ch, int start, int length)

- etc,.,

Exemple d'utilisation de SAX

- Créer un nouveau Projet Java sous Eclipse que vous nommerez SAXProject

- Importer dans SAXProject le fichier XML carnet.xml et la classe ExempleSAX.java
- Exécuter et visualiser le résultat

- Analyser le programme
- Alors? Que fait ce programme?

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
   <personne id="p1">
            <nom>Fog</nom>
            <prenom>Phileas</prenom>
   </personne>

   <personne id="p2">
            <nom>Partout</nom>
            <prenom>Passe</prenom>
</personne>

</carnet>
```

carnet.xml

- **Evénements avec SAX:**

Document XML

```
<carnet>
   <personne id="p1">
            <nom>Fog</nom>
            <prenom>Phileas</prenom>
  </personne>

   <personne id="p2">
            <nom>Partout</nom>
            <prenom>Passe</prenom>
</personne>

</carnet>
```

- **Les événements :**

start document
start element: carnet
characters:
start element: personne id p1
characters:
start element: nom
characters: “Fog “
end element: nom
etc……..

end element: personne
end element: carnet

end document

Programme principal du parser main()

L'interface XMLReader représente le parseur XML

```
public static void main(String[] args) throws Exception{
XMLReader parser = XMLReaderFactory.createXMLReader();
ExempleSax handler=new ExempleSax();
 parser.setContentHandler(handler);
 parser.setErrorHandler(handler);
 parser.setFeature("http://xml.org/sax/features/validation", true);
 parser.parse("carnet.xml");
}
```

Création de parser

Association d'un gestionnaire de contenu et d'erreurs.

Parsing du document

- **Gestionnaire de contenu**

```
public void startDocument () {
System.out.println("Start document");
}
public void endDocument () {
System.out.println("End document");

}
public void startElement ( String uri,String name,String qName,Attributes
atts) {
System.out.println("Start element:" + name);
Etc…..
}

public void endElement (String uri, String name, String qName) {
System.out.println("End element: " + name);
}
```

- **Exemple: Traitement des caractères**

```
public void characters (char ch[], int start, int length) {
System.out.print("Characters: \"");
for( int i = 0; i < length; i++ ){
        System.out.print(ch[start+i]);
    }
    System.out.println();}
```

void **characters**(char[] ch, int start, int length) appelée à la lecture d'un contenu textuel, qui se trouve dans le tableau ch entre les indices start et start+length.

Interface ErrorHandler

- **Exemple: Traitement des erreurs**

```
public void error(SAXParseException e) {
System.err.println("Erreur non fatale (ligne " +
e.getLineNumber() + ", col " +
e.getColumnNumber() + ") : " + e.getMessage());
}
public void fatalError(SAXParseException e) {
System.err.println("Erreur fatale : " + e.getMessage());
}
public void warning(SAXParseException e) {
System.err.println("warning : " + e.getMessage());
}
```

- **Exercice**

On reprend l'exemple du livre

```
<?xml version="1.0" encoding="UTF-8"?>
<livre titre=“ mon livre“>
    <auteurs>
       <auteur nom=“Martin“  prenom=“Bill“ />
   </auteurs>
  <sections>
    <section titre=“ une section“ >
      <chapitre titre=“un chapitre “ >
              <paragraphe>paragraphe 1 </paragraphe>
              <paragraphe>paragraphe 2 </paragraphe>
      </chapitre>
      </section>
</sections>
</livre>
```

Vous allez écrire un programme utilisant SAX qui affichera le plan du livre avec la liste des auteurs.

Le document pourra contenir plusieurs auteurs, sections, chapitres

mon livre
Auteurs
Auteur
nom: Martin
Prenom: Bill
Sections
1.Section
1.1. Un chapitre
2.Section
2.1 Second chapitre

API SAX (Simple API for XML)

- Particularité du parser SAX:
  - Event-driven: on a besoin de fournir les fonctions répondant à l'envoi d'évènements
  - Le parcours des documents est séquentiel (sequential read-only access)
  - On accède une seule fois à chaque partie d'un document (one-time access)

- C'est une approche rapide et simple:
  - On n'a pas besoin de charger un document entièrement en mémoire pour travailler
  - Par contre on ne peut pas modifier un document dynamiquement

API SAX (Simple API for XML)

- **Avantages de SAX**
  - Vitesse de traitement
  - Faible consommation mémoire côté parseur
  - Outil pratique lors du traitement de documents de taille importante

- **Inconvénients de SAX**
  - Complexité de la programmation
  - Impossibilité de pratiquer des requêtes par XPath/Xquery nécessitant une représentation globale en mémoire

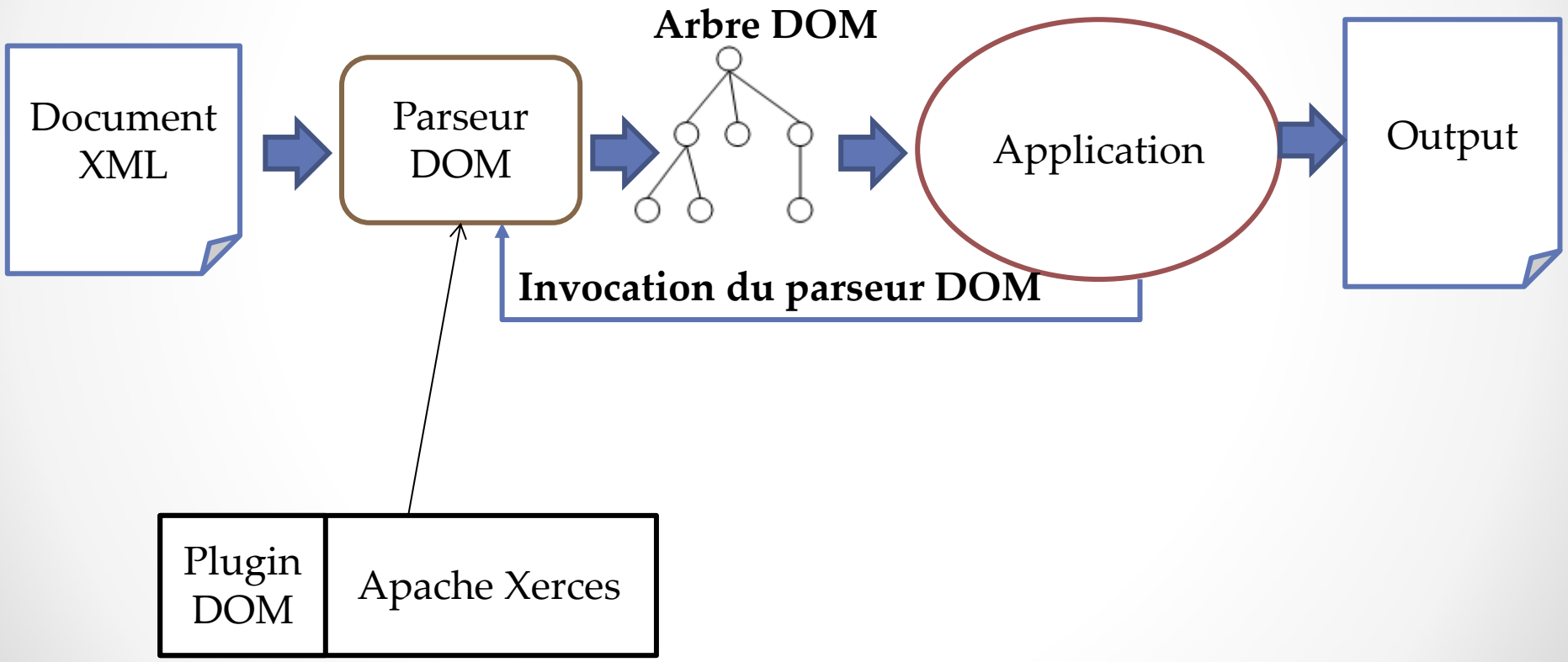
2. API DOM (Document Object Model)

(http://www.w3.org/DOM/)

DOM (Document Object Model)

- Est une représentation objet d'un document XML
- Chaque classe (ou type) d'objets provient des types de noeuds dans un document XML (élément, texte…).
- Est une API disponible dans la plupart des langages et une recommandation du W3C (*http://www.w3.org/DOM/*) depuis 1998.
- DOM 3 publié en 2004 et DOM4 en 2015

DOM (Document Object Model)



Exemple d'utilisation de DOM

- Créer un nouveau Projet Java sous Eclipse que vous nommerez DomProject
- Importer dans DomProject le fichier XML carnet.xml et la classe ExempleDom
- Exécuter et visualiser le résultat
- Alors? Que fait ce programme?
- Analyser le programme

```
<?xml version="1.0" encoding="UTF-8"?>
<carnet>
   <personne id="p1">
            <nom>Fog</nom>
            <prenom>Phileas</prenom>
   </personne>

   <personne id="p2">
            <nom>Partout</nom>
            <prenom>Passe</prenom>
</personne>

</carnet>
```

carnet.xml

- Déclaration de l'analyseur XML et création du document

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

public static void main(String[]args) throws Throwable{

  Document doc;
```

1. Création d'un objet pour la lecture du fichier XML

```
1. DocumentBuilderFactory   dbf=DocumentBuilderFactory.newInstance();
2.   DocumentBuilder db=dbf.newDocumentBuilder();
```

2. db est l'objet qui va produire un document DOM(parseur)

```
3. doc=db.parse("carnet.xml");
```

3. Création de la structure en mémoire

```
4.  Element root=doc.getDocumentElement();
```

4. Récupération de la racine du document

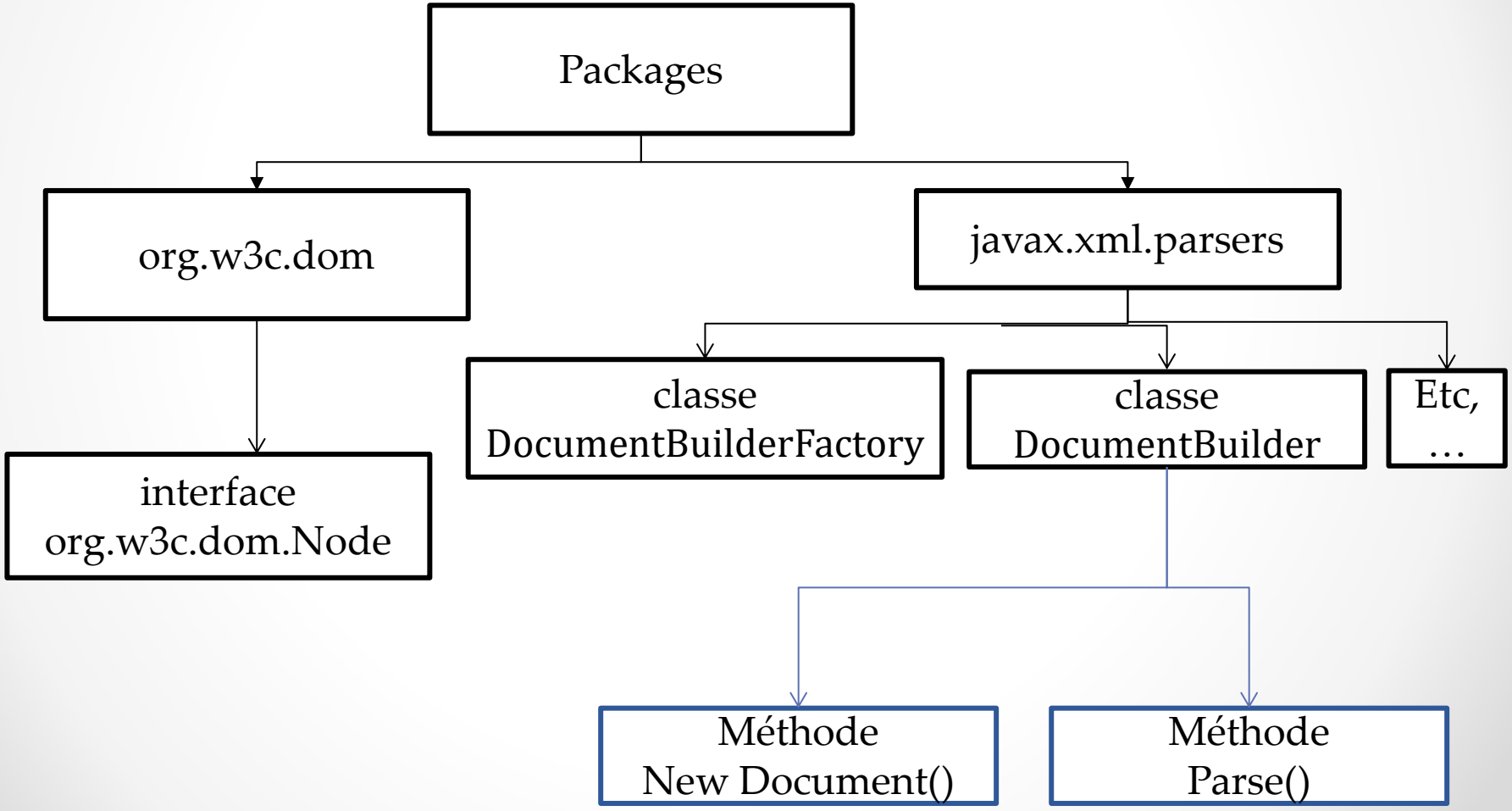
```
5.  parcours (root);
}
```

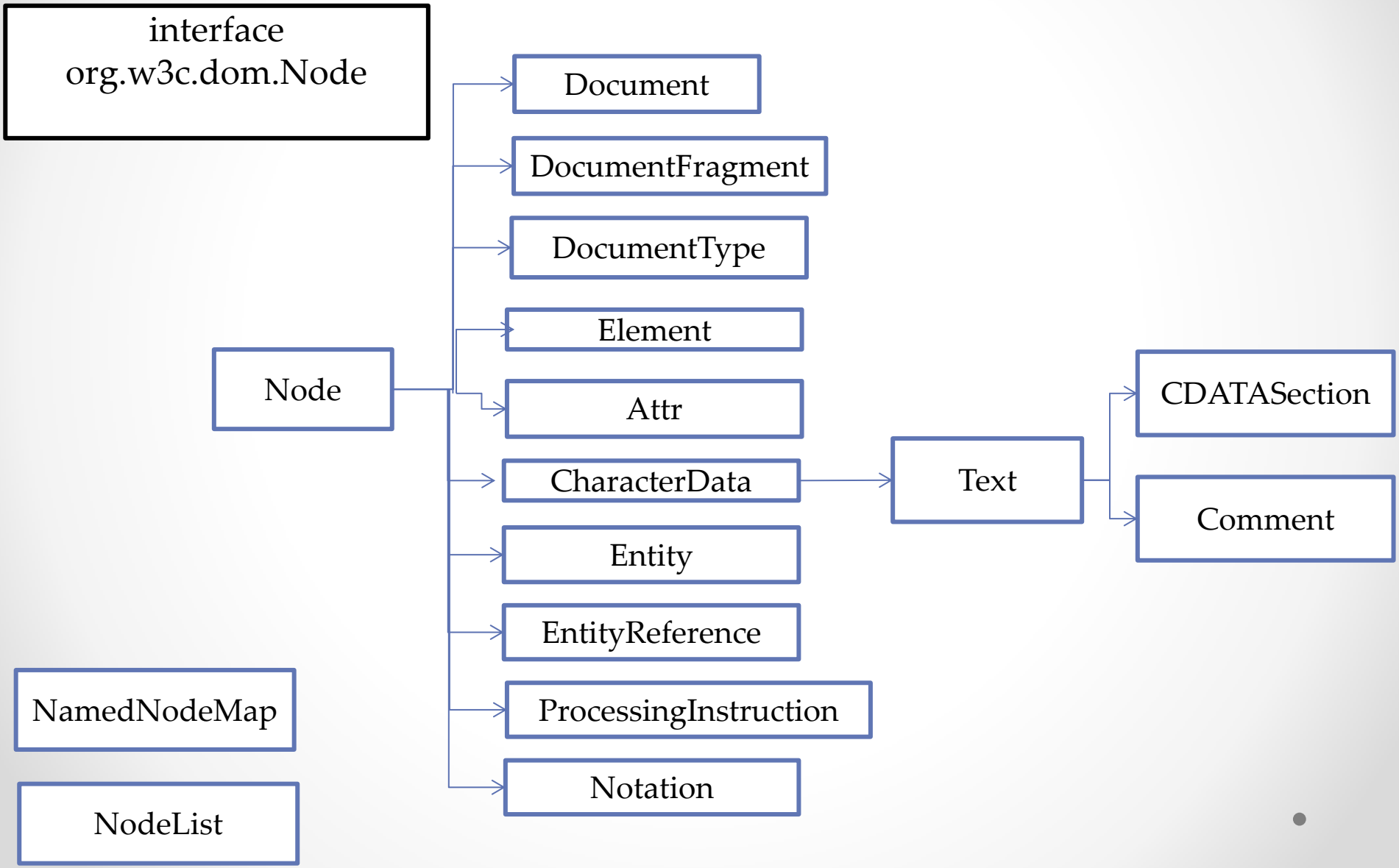
5. Méthode pour parcourir l'arbre

- Parcours d'un arbre DOM

```
static void parcours (Element e){

     System.out.println("-"+e.getNodeName());

//parcours de tous les attributs
    NamedNodeMap nnm=e.getAttributes();
     for (int i=0;i<nnm.getLength();i++){
        Node n=nnm.item(i);
         System.out.println("*"+n.getNodeName()+"="+n.getNodeValue());
              }

 //parcours des fils

      NodeList nl=e.getChildNodes();
       for (int i=0;i<nl.getLength();i++){
          Node n=nl.item(i);
          if(n.getNodeType()==Node.ELEMENT_NODE){
           parcours((Element)n);
              }else
          if(n.getNodeType()==Node.TEXT_NODE){
          System.out.println(n.getNodeValue());}}}
```

Packages


Packages
org.w3c.dom
javax.xml.parsers
interface
org.w3c.dom.Node
classe
DocumentBuilderFactory
classe
DocumentBuilder
Etc,
…
Méthode
New Document()
Méthode
Parse()


interface
org.w3c.dom.Node
Node
Document
DocumentFragment
DocumentType
Element
Attr
CharacterData
Text
CDATASection
Comment
Entity
EntityReference
ProcessingInstruction
Notation
NamedNodeMap
NodeList

interface
org.w3c.dom.Node

Méthodes de lecture	Description
String getNodeName()	Le nom du nœud.
String getNodeValue()	Si le nœud est un attribut cela correspondra à sa valeur
short getNodeType()	Renvoie le type de noeud
NamedNodeMap getAttributes()	Liste des attributs (pour l'élément). L'ensemble des nœuds attribut sera disponible via un objet de type NamedNodeMap.
Node getFirstChild()	Le premier nœud fils
Node getNextSibling()/getPreviousSibling()	Le nœud adjacent (après ou avant)
Node getLastChild()	Le dernier nœud fils
Node getParentNode()	Le nœud parent
NodeList getChildNodes()	Liste des nœuds fils
boolean hasChildNodes()	Renvoie un booléen qui indique si le noeud a au moins un fils
Document getOwnerDocument()	Renvoie le document dans lequel le noeud est inclus
String getNamespaceURI()	L'URI, si un espace de noms est utilisé

interface
org.w3c.dom.Node

Méthodes de modification	Description
Node appendChild(Node newchild)	Ajoute un fils
Node cloneNode(boolean deep)	Retourne un clone (complet ou non) du noeud
Node insertBefore(Node newchild, Node refchild)	Insertion d'un nouveau nœud avant le fils refchild
Node removeChild(Node oldChild)	Suppression d'un nœud fils
Node replaceChild(Node newChild, Node oldChild)	Replacement d'un nœud fils par un nouveau
void setNodeValue(String nodeValue)	Modification de la valeur d'un nœud; l 'opération dépendra du type de noeud
void setPrefix(String prefix)	Définir un préfixe pour ce noeud

- **Exercice**

On reprend l'exemple du livre

```
<?xml version="1.0" encoding="UTF-8"?>
<livre titre=“ mon livre“>
    <auteurs>
       <auteur nom=“Martin“  prenom=“Bill“ />
   </auteurs>
  <sections>
    <section titre=“ une section“ >
      <chapitre titre=“un chapitre “ >
              <paragraphe>paragraphe 1 </paragraphe>
              <paragraphe>paragraphe 2 </paragraphe>
      </chapitre>
      </section>
</sections>
</livre>
```

Vous allez écrire un programme utilisant DOM qui affichera le plan du livre avec la liste des auteurs.

Le document pourra contenir plusieurs auteurs, sections, chapitres

Livre:Mon livre
Auteur: nom1 prenom1
Auteur: nom2 prenom2

1. Section1
   1.1. Chapitre1
2. Section2
   2.1. Chapitre1
   2.2. Chapitre2

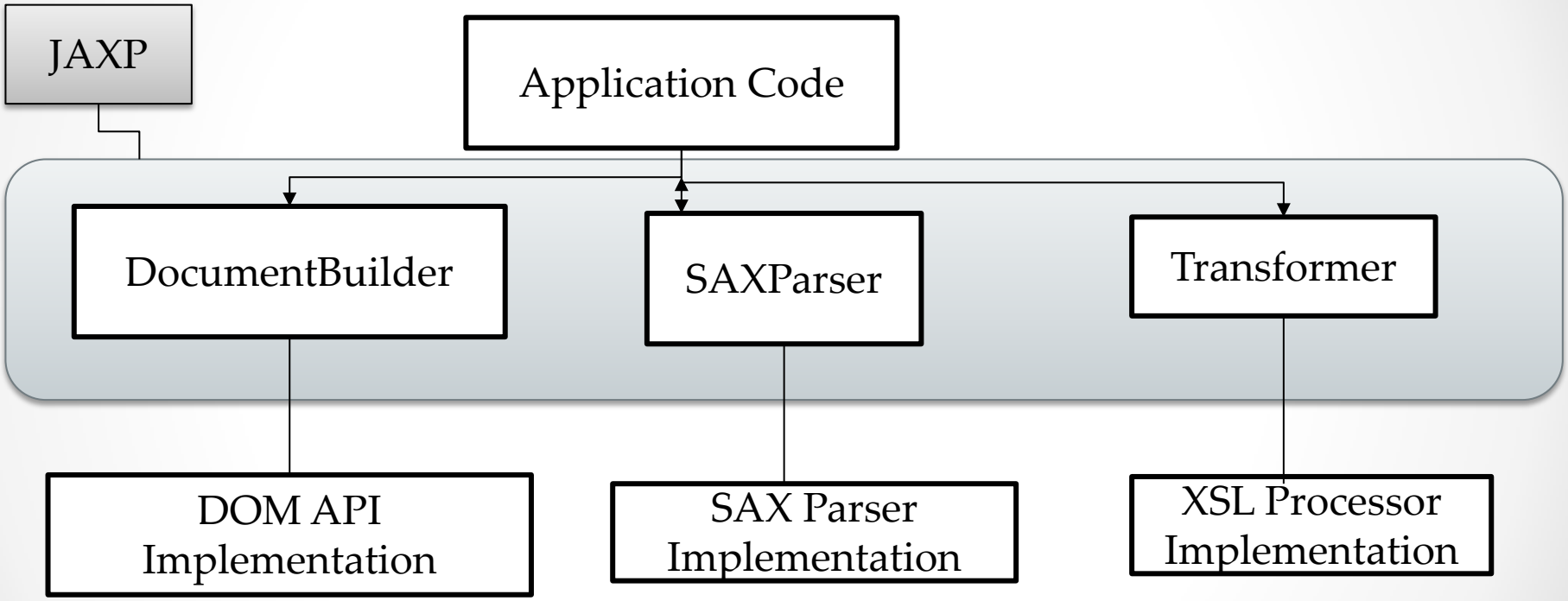
- **Avantages de DOM**
  - Permet un accès direct (random-access) à un document XML.
  - On peut créer un DOM à partir de rien ou encore modifier un document contenu dans un fichier déjà existant.

- **Inconvénients de DOM**
  - Cette approche n'est pas adaptée à la manipulation de grands documents car le document entier doit être chargé en mémoire

JAXP (Java API for XML Processing)

(https://jaxp.java.net/)

JAXP (Java API for XML Processing)



- Proposé par JavaSoft (http://java.sun.com/xml/jaxp/) et présent depuis java2 version 1.4.

- Constituée de packages pour: le parsing (SAX et DOM), création et transformation de fichiers XML)

Packages JAXP

La librairie JAXP utilise plusieurs packages

- **javax.xml.parsers:** contient les classes permettant de manipuler les documents XML. On peut les « connecter » à deux types possibles de parseurs :
  - SAX : SAXParser et SAXParserFactory
  - DOM : DocumentBuilder et DocumentBuilderFactory
- org.w3c.dom
  - Package DOM
- org.xml.sax et org.xml.sax.helpers
  - Packages SAX
- javax.xml.transform : ensemble des classes et interfaces pour transformer un document XML

Les transformations XSLT

- Créer un nouveau projet que vous nommerez JAXPProject
- Importer la feuille de style carnet.xslt et le document carnet.xml
- Vous allez tester le code de transformation suivant:

```
public static void main(String[] args) throws Throwable {
TransformerFactory factory = TransformerFactory.newInstance();
Transformer t = factory.newTransformer(
new StreamSource( "carnet.xslt" ) );
t.transform(
new StreamSource( "carnet.xml" ),
new StreamResult("c:/carnet.html" ) );
}
```

- Quel résultat obtenez vous et pourquoi?

Les transformations XSLT à partir d'un arbre DOM

- En s'appuyant sur un document à transformer, fourni sous la forme d'un arbre DOM, vous allez tester le code de transformation suivant:

```
// Création d'un document DOM
DocumentBuilderFactory factory1 = DocumentBuilderFactory.newInstance();
DocumentBuilder db = factory1.newDocumentBuilder();
Document doc = db.parse( "carnet.xml" );
...
TransformerFactory factory = TransformerFactory.newInstance();
Transformer t = factory.newTransformer( new StreamSource( "carnet.xslt" ) );

// Transformation de l'arbre DOM
t.transform( new DOMSource(doc ), new StreamResult( "c:/carnet.html" ) );
```

- Quel résultat obtenez vous et pourquoi?

4. Les alternatives à SAX/DOM/JAXP

4.1 La librairie JDOM (http://www.jdom.org/)

4.2 La librairie DOM4J (http://www.dom4j.org/)

JDOM (Java Document Object Model)

- **JDOM:** Une API du langage Java
- **Objectifs :** offrir une bibliothèque simple pour la représentation et la manipulation de documents XML (http://www.jdom.org).

- Propose différentes fonctionnalités:
  - Création de documents XML
  - Représentation DOM,
  - les évènements SAX,
  - Support de XSLT et XPath

JDOM

Tester l'exemple suivant et observer le résutlat.
N'oublier pas d'importer la librairie jdom.jar

```
public class JDOMExample {

public static void main(String[] args) throws FileNotFoundException, IOException
{
        Element livre = new Element("livre").addContent(new
        Element("auteur").setText("Jack")) .addContent(new
        Element("section").setText("MaSection"));
        Document root = new Document(livre);
        Format format = Format.getPrettyFormat();
        format.setEncoding("ISO-8859-1");

        XMLOutputter outputter = new XMLOutputter (format);
        outputter.output(root, new FileOutputStream("c:/ExempleJDOM.xml"));

        try { outputter.output(root, System.out); }
        catch (java.io.IOException e) { }
}
}
```

DOM vs JDOM

- **La gestion des éléments: Création d'un élément**

  - **JDOM**

La création d'un élément avec JDOM est directe, puisque chaque noeud est associé à une classe : Element element = new Element("test");

- **DOM**

Il faut obtenir un document pour pouvoir créer un noeud, ce qui donne :

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element element = doc.createElement("test");
```

DOM vs JDOM

- **La gestion des éléments: Ajout d'un texte**
  - **JDOM**

    element.setText("ok");
  - **DOM**

    Text t = doc.createTextNode("ok");

    element.appendChild(t);

- **La gestion des éléments: Affectation**

Identique entre DOM et JDOM: element.setAttribute("a", "1");

DOM4J

- Une autre librairie qui utilise XPATH (http://www.dom4j.org)

```
public class DOM4JExample {
        public static void main(String args[]) throws IOException {
  Document document = DocumentHelper.createDocument();
        Element root = document.addElement("livre");
        root.addElement("author").addAttribute("name", "Toby")
        .addAttribute("location", "Germany")
        .addText("Tobias Rademacher");
        root.addElement("author").addAttribute("name", "James")
        .addAttribute("location", "UK").addText("James Strachan");

        List<?> results = document.selectNodes("//author[@location = 'UK']");
        for (Iterator iter = results.iterator(); iter.hasNext();) {
        Element element = (Element) iter.next();
        System.out.println(element.valueOf("concat(@name,' : ', .)"));}

  OutputFormat outformat = OutputFormat.createPrettyPrint() ;
  outformat.setEncoding("ISO-8859-1");
   XMLWriter writer = new XMLWriter(outformat);
   writer.write (document);
   writer.close();}}
```

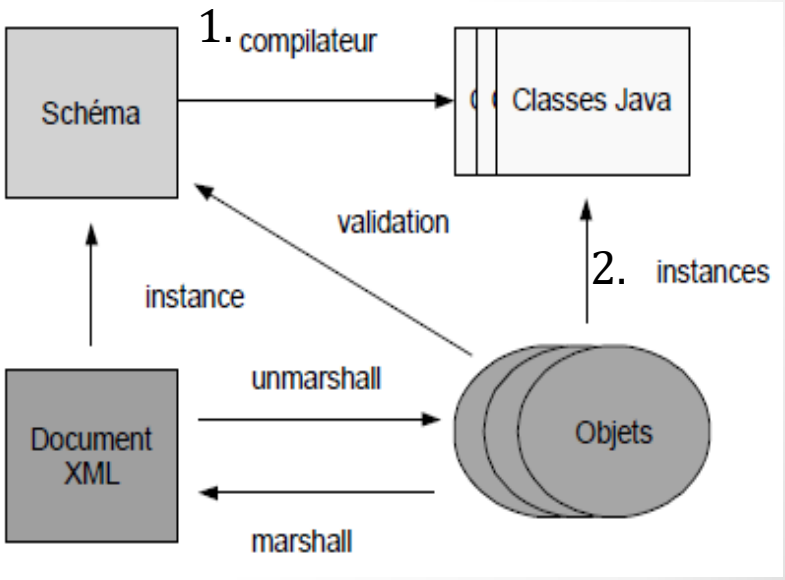
5. JAXB (Java API for XML Binding)

JAXB (Java API for XML Binding) : API permettant de faire correspondre un document XML à un ensemble de classes et vice versa au moyen d'opérations de sérialisation/désérialisation (marshalling/unmarshalling).

- 2 étapes pour utiliser JAXB :

  1. Génération des classes et interfaces à partir du schéma XML (grâce au compilateur xjc)

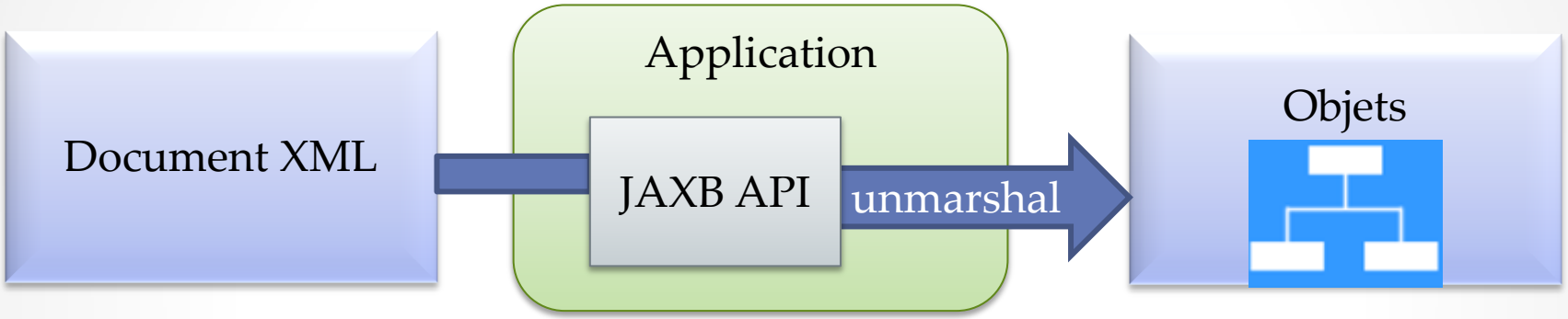
  2. Utilisation des classes générées et de l'API JAXB pour transformer un document XML en graphe d'objets et vice versa, pour manipuler les données dans le graphe d'objets et pour valider le document



- **Compilation d'un Schéma (DTD non disponible pour JAXB**

  - L'outil xjc permet de générer les classes à partir d'un schéma XML.
  - Les classes générées:
    - Chaque classe qui encapsule un type complexe du schéma possède des getters et setters sur les éléments du schéma.
    - La fabrique permet de créer des instances de chacun des types d'objets correspondant à un type complexe du schéma.

Lecture d'un document XML(opération Unmarshalling)



- La création des objets nécessite la création d'un objet de type JAXBContext (qui permet d'utiliser l'API) en utilisant la méthode statique newInstance().

```
JAXBContext context = JAXBContext.newInstance("monpkg");
```

- Instancier (méthode createUnmarshaller) un objet de type unmarshaller qui transformera le document XML en un ensemble d'objets.

```
// creer les instances à partir du fichier XML
Unmarshaller um = context.createUnmarshaller();
```

Génération de documents XML (opération marshalling)

Est l'étape de reconstruction du document XML à partir de sa représentation objet.

- La création des objets nécessite la création d'un objet de type JAXBContext (qui permet d'utiliser l'API) en utilisant la méthode statique newInstance().

```
JAXBContext jc = JAXBContext.newInstance( "monpkg" );
```

- Instancier un objet de type Marshaller qui va permettre de transformer un ensemble d'objets en un document XML

```
Marshaller ms = jc.createMarshaller();
ms.setProperty( "jaxb.formatted.output", Boolean.TRUE );
ms.marshal( element, new FileWriter( "c:/carnet.xml" ) );
```

Mise en oeuvre:

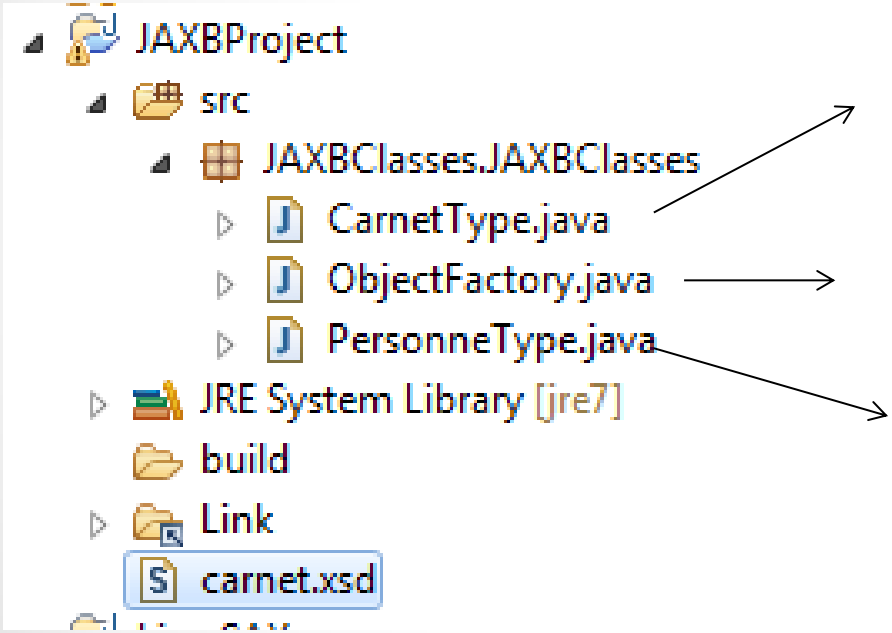
- Utilisation de l'outil xjc de JAXB pour la génération des classes à partir d'un schéma du document XML

- Ecriture de code utilisant les classes générées et l'API JAXB pour transformer un document XML en objets Java, modifier des données encapsulées dans le graphe d'objets ou transformer le graphe d'objets en un document XML avec une validation optionnelle du document

- Compilation du code généré et écrit puis l'exécution de l'application

- Télécharger le plugin Jaxb (ou bien récupérer sur ma page le .zip de jaxB) et ajouter le contenu du dossier plugin aux plugins de eclipse
- Démarrer Eclipse
- Créer un nouveau Projet Java que vous nommerez JAXBProject
- Ajouter à votre projet le schéma XML carnet.xsd et le document carnet.xml
- Faites un clic droit sur le schéma, sélectionner JAXB2.0→ run XJC
- Que s'est il passé?

- 3 fichiers ont été crées



définit un getter sur une collection qui contiendra toutes les personnes

fabrique qui permet d'instancier des objets utilisés lors du mapping.

définit des getters et des setters sur les Éléments nom et prénom

- Regarder leur contenu et expliquer ce qui a été généré

Un objet de type factory et des classes annotées pour chacun des éléments complexes qui composent le document sont définis (Personne et Carnet)

- À l'aide des classes obtenues grâce au compilateur, vous allez pouvoir manipuler le document XML;
- Créer une nouvelle classe que vous nommerez JAXBTest et ajouter dans votre main les lignes suivantes. Compiler et exécuter votre application.

permet de gérer la transformation d'objets Java en XML et vice versa

```
JAXBContext context = JAXBContext.newInstance("monpkg");
// créer les instances à partir du fichier XML
Unmarshaller um = context.createUnmarshaller();

JAXBElement element = ( JAXBElement )um.unmarshal( new File( "carnet.xml" ) );
CarnetType ct = ( CarnetType )element.getValue();
List<PersonneType> lp = ct.getPersonne();
for ( int i = 0; i < lp.size(); i++ ) {
        PersonneType tp = lp.get( i );
        System.out.println( tp.getNom() );
        System.out.println( tp.getPrenom() );}
```

- L’activité de développement "parsing" est réduite
  - Gain en coût de développement des interfaces
  - Les applications peuvent se concentrer sur le fonctionnel
  - Les développeurs n'ont pas à se soucier du mapping XML
- Les Systèmes Techniques sont simplifiés
  - Gain en termes de fiabilité (fin de l’utilisation de plusieurs parseurs, parfois écrits à la main)
  - Les performances de l’interface sont optimisées

Pour conclure

- SAX : parsing “à la volée” d’un document XML. Une seule lecture (économe en mémoire), mais pas de modifications possibles.
- DOM : construction d’un arbre représentant le document. Toutes les transformations sont ensuite envisageables...
- JAXB : traduction automatique d’une structure de Schéma XML en graphe de classes Java.
- Et allez voir la Java Doc

Références

- DOM JavaDoc du package org.w3c.dom : http://www.dil.univ-mrs.fr/docs/java/api/org/w3c/dom/package-summary.html
- SAX: http://www.dil.univ-mrs.fr/docs/java/api/index.html?javax/xml/transform/sax/package-summary.html
- JAXB: http://www.jmdoudoux.fr/java/dej/chap-jaxb.htm