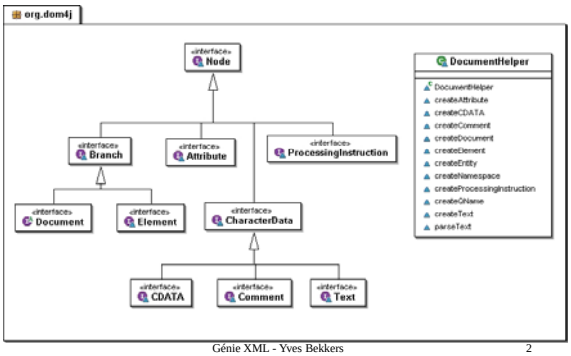
DOM4J

Diagramme des classes



Compiler un document XML à partir d'un fichier

- Compilateur
  `org.dom4j.io.SAXReader`
- Code

```
java.io.File myFile =
      new java.io.File("monFichier.xml");
org.dom4j.io.SAXReader reader =
      new SAXReader();
org.dom4j.Document document =
      reader.read(myFile);
```


Compiler un document XML a partir d'une chaîne

- Utiliser la méthode parseText() de la classe
  `org.dom4j.DocumentHelper`
- Code

```
String s = "<root id=\"a1\"/>";
org.dom4j.Document document =
      DocumentHelper.parseText(s);
```


Ecrire un document DOM4J dans un fichier (présentation brute)

- Utiliser la mèthode `write()` de
  `org.dom4j.Document`
- Code

```
java.io.FileWriter out =
    new java.io.FileWriter(
        new java.io.File("monfichier");
org.dom4j.Document doc = ...
doc.write( out );
out.close();
```


Ecrire un document DOM4J dans un fichier (avec indentation)

- Utiliser les formats
  `org.dom4j.io.OutputFormat`
- Code

```
OutputFormat format =
   OutputFormat.createPrettyPrint();
writer = new XMLWriter(System.out,format);
writer.write( document );
```

Ecrire un document DOM4J dans un fichier (présentation compacte)

- Utiliser les formats
  `org.dom4j.io.OutputFormat`
- Code

```
Document document = ...;
OutputFormat format =
   OutputFormat. createCompactFormat();
writer = new XMLWriter(System.out,format);
writer.write( document );
```


Ecrire un document DOM4J dans un fichier (codage « ISO-8859-1 »)

- Utiliser les formats
  `org.dom4j.io.OutputFormat`
- Code

```
Document document = ...;
OutputFormat format =
   OutputFormat. createCompactFormat();
format.setEncoding("ISO-8859-1");
Writer fileWriter = new OutputStreamWriter(
    new FileOutputStream(new File(docFileName)),
    "ISO-8859-1");
XMLWriter writer = new XMLWriter(
   new BufferedWriter(fileWriter), format);

writer.write(DOC);
writer.close();
```


Ecrire un document DOM4J dans une chaîne

- Utiliser la méthode `asXML()` de la classe
  `org.dom4j.Node`
- Code

```
Document document = ...;
String text = document.asXML();
```


Créer un document DOM4J

- Utiliser la classe de création
  `org.dom4j.DocumentHelper`
- Code

```
org.dom4j.Document document =
         DocumentHelper.createDocument();
org.dom4j.Element root =
         document.addElement( "root" );
```


Créer un élément DOM4J

- Utiliser la classe de création
  `org.dom4j.DocumentHelper`
- Code

```
Element elem =
  DocumentHelper.createElement("test");
```


Ajouter des éléments

- Composer le résultat des méthodes
  `addElement(), addAttribute(), addText()`
- Code

```
Element root = ...;
Element author1 =
       root.addElement( "author" )
  .addAttribute( "name", "James" )
  .addAttribute( "location", "UK" )
  .addText( "James Strachan" );
```

Naviguer avec XPath

- Utiliser les méthodes (classe `Node`)
  `selecNodes(), selecSingleNode()`
- Code

```
Document document = ...;
List list =
   document.selectNodes( "//foo/bar" );

Node node = document.selectSingleNode(
  "//foo/bar/author" );
```


Iterations Java

- Utiliser les iterations de Java
  `java.util.List, java.util.Iterator`
- Code

```
Document document = ...;
List list = document.selectNodes( "//a/@href" );
for (Iterator iter = list.iterator();
                          iter.hasNext(); ) {
   Attribute attribute = (Attribute) iter.next();
   String url = attribute.getValue();
}
```


Iterations rapides (DOM4J)

- Utiliser les méthodes suivantes de la classe `Node`
  `elem.nodeCount(), elem.node(i)`
- Code

```
public void treeWalk(Element element) {
   for (int i=0, size=element.nodeCount(); i<size;
  i++) {
      Node node = element.node(i);
      if ( node instanceof Element ) {
         treeWalk( (Element) node );
      } else {
         // do something....
      }
   }
}
```


Transformer un document DOM4J (1) préparer la feuille de style

- Utiliser les classes (JAPX)
  `javax.xml.transform.Transformer`
  `javax.xml.transform.TransformerFactory`
  `javax.xml.transform.stream.StreamSource`
- Code

```
File stylesheet = new File("style.xslt");
TransformerFactory factory =
    TransformerFactory.newInstance();
Transformer myTrans =
    factory.newTransformer(
        new StreamSource( stylesheet );
```


Transformer un document DOM4J (2) Transformer

- Utiliser les classes
  `org.dom4j.io.DocumentSource`
  `org.dom4j.io.DocumentResult`
- Code

```
org.dom4j.io.DocumentSource source =
   new DocumentSource( document );
org.dom4j.io.DocumentResult result =
   new DocumentResult();
myTrans.transform( source, result );
```


Transformer un document DOM4J (3) Récupérer le résultat

- Utiliser la méthode `getDocument()` de la classe
  `org.dom4j.io.DocumentResult`
- Code

```
Document transformedDoc =
   result.getDocument();
```

Transformer un document DOM4J en un document DOM

```
import org.dom4j.io.DocumentSource;
import javax.xml.transform.dom.DOMResult;

public org.w3c.dom.Document getDOMDocument() {
   org.dom4j.Document doc;
   Transformer transformer = TransformerFactory
      .newInstance().newTransformer();
   DocumentSource source = new DocumentSource(doc);
   DOMResult result = new DOMResult();
   transformer.transform( source, result );

   return (org.w3c.dom.Document) result.getNode();
}
```


Conclusion Dom4J

- DOM4J
  - basé sur l'utilisation d'interfaces
  - Implémente Xpath en natif
    - permet d'explorer un document DOM4J directement

```
Node nd = doc.selectSingleNode("//fo/bar/auteur");
```

- Un article de comparaison par DOM4J
  - `http://dom4j.org/compare.html`
- Un article de comparaison par IBM
  - `ftp://www6.software.ibm.com/software/developer/library/x-injava.pdf`

Castor

Castor à quoi cela sert ?

- Génération automatique de compilateur à partir
  - D'une classe modèle
  - D'un schéma

Une Classe JavaBean

```
public class Personne implements
  java.io.Serializable {
  private String nom = null;
  private int age;

  public Personne() {super();}
  public Personne(String nom) {this.nom = nom;}
  public int getAge() {return age;}
  public String getNom() {return nom;}
  public void setAge(int age) {this.age = age;}
  public void setNom(String nom) {this.nom =
  nom;}
}
```


JavaBean

- Propriétés
  1. La classe est définie avec la protection `public`
  2. Elle possède un constructeur par défaut (constructeur sans paramètre)
  3. La classe implémente l'interface `java.io.Serializable`
  4. Chaque attribut `XXX` de type `TT` est accédé exclusivement à l'aide de méthodes
     - `public TT getXXX()`
     - `public void setXXX(TT val)`

Sérialiser une Classe JavaBean

- Initialisation d'une personne

```
Personne dutertre =
   new Personne("Du tertre");
dutertre.setAge(35);
```

- Sérialisation d'une personne

```
FileWriter file =
    new FileWriter(nomFichier);
Marshaller.marshal(dutertre, file);
file.close();
```

Méthode `static` => sérialisation par défaut

Résultant de la sérialisation par défaut

```
<?xml version="1.0" encoding="UTF-8"?>
<personne age="35">
  <nom>Du tertre</nom>
</personne>
```


Sérialisation par défaut

- Castor utilise les facilités d'introspection dynamique des classes offertes par Java
- Sérialisation par défaut
  - Le nom d'élément est dérivé de celui de la classe
  - Les attributs de la classe et leur type sont découverts par introspection des méthodes `getXXX` et `setXXX` (dans le cas d'un JavaBean).
  - Les attributs de la classe donnent lieu à deux types de sérialisation selon le type Java de l'attribut
    - Pour un type primitif on génère un attribut d'élément de même nom que l'attribut d'instance
    - Pour un type non primitif (`String` compris) on génère un sous-élément de même nom que l'attribut d'instance

Gérer les exceptions

```
try {
  writer = new FileWriter(nomFichier);
  Marshaller.marshal(dutertre, writer);
} catch (MarshalException e) {
  // TODO Bloc catch auto-généré
  e.printStackTrace();
} catch (ValidationException e) {
  // TODO Bloc catch auto-généré
  e.printStackTrace();
} catch (IOException e) {
  // TODO Bloc catch auto-généré
  e.printStackTrace();
}
```


Compiler un document XML en une instance de Classe JavaBean

- Compiler un document Personne.xml

```
FileReader reader = new
  FileReader(nomFichier);
Personne unePersonne = (Personne)
  Unmarshaller.
    unmarshal(Personne.class, reader);
```

Résultat

Nom de la classe

Méthode `static` => traduction par défaut

Gérer les exceptions

```
try {
  FileReader reader = new
              FileReader(nomFichier);
  Personne unePersonne =
     (Personne) Unmarshaller.
      unmarshal(Personne.class, reader);
} catch (MarshalException e1) {
  e1.printStackTrace();
} catch (ValidationException e1) {
  e1.printStackTrace();
} catch (FileNotFoundException e) {
  e.printStackTrace();
}
```

Modifier un bean

- Modifier un bean

```
unePersonne.setNom("?!xml#");
unePersonne.setAge(2000);
```


Modification du bean

- En entrée

```
<?xml version="1.0" encoding="UTF-8"?>
<personne age="35">
  <nom>Du tertre</nom>
</personne>
```

- En sortie

```
<?xml version="1.0" encoding="UTF-8"?>
<personne age="2000">
  <nom>?!xml#</nom>
</personne>
```


Provoquer une erreur de compilation

- En entrée

```
<?xml version="1.0" encoding="UTF-8"?>
<personne age="deux">
  <nom>Du tertre</nom>
</personne>
```

- En sortie

```
java.lang.NumberFormatException: deux
at
  java.lang.Integer.parseInt(Integer.java:426
  )
```


Castor accepte n'importe quelle classe

- Si la classe ne possède aucune méthode `getXxx` et `setXxx` Castor recherche les attributs `public` d'instance
- Si elle possède un seul accesseur `get` ou `set` les attributs d'instance sont ignorés

Choix d'un mode de traduction

- Traduction par défaut

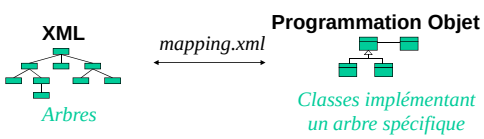
```
<personne age="35">
  <nom>Du tertre</nom>
</personne>
```

- Traduction désirée

```
<personne>
  <age>35</age>
  <nom>Du tertre</nom>
</personne>
```


Assister la traduction

- Lorsque la traduction par défaut ne convient pas, on peut spécifier sa propre traduction
  - Définir un fichier `mapping.xml`



Spécifier un mode de traduction

- Fichier mapping.xml

```
<mapping>
  <description>un essai</description>
  <class name="Personne" auto-complete="true">
    <field name="age">
      <bind-xml name="age" node="element"/>
    </field>
    <field name="nom" type="string"/>
  </class>
</mapping>
```

L'attribut age sera généré comme un élément XML

fichier `mapping.xml` en lecture

```
// Creation du objet de traduction
Mapping mapping = new Mapping();
mapping.loadMapping(mapFile);

// Creation du lecteur
FileReader reader = new FileReader(nomFichier);

// Creation d'un compilateur
Unmarshaller unmarshaller = new
  Unmarshaller(Personne.class);
unmarshaller.setMapping(mapping);

// Compilation
unePersonne = (Personne)
  unmarshaller.unmarshal(reader);
```

Configuration du compilateur

fichier `mapping.xml` en écriture

```
// Creation de l'objet
Personne laPersonne = …;
Mapping mapping = new Mapping();
mapping.loadMapping(mapFile);

// Creation du writer
FileWriter writer = new FileWriter(nomFichier);

// Creation d'un sérialisateur
Marshaller marshaller = new Marshaller(writer);
marshaller.setMapping(mapping);

// sérialisation
marshaller.marshal(laPersonne);
```


Collections d'éléments - 1

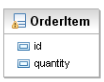
- Collection d'éléments à la racine d'un document XML

```
<order>
  <order-item id="1" quantity="15" />
  <order-item id="2" quantity="20" />
</order>
```

Collections d'éléments - 2

- Fichier de mapping

```
<mapping>
  <class name="fr.ifsic.bd.order.OrderItem">
    <field name="Id" type="java.lang.String">
      <bind-xml name="id" node="attribute" />
    </field>
    <field name="Quantity" type="java.lang.Integer">
      <bind-xml name="quantity" node="attribute" />
    </field>
  </class>
</mapping>
```



Lecture de la collections

- Utilisation d'un `ArrayList`

```
mapping.loadMapping("mapping.xml");
FileReader in = new FileReader("items.xml");

Unmarshaller unmarshaller =
          new Unmarshaller(ArrayList.class);
unmarshaller.setMapping(mapping);
ArrayList orders = (ArrayList)
          unmarshaller.unmarshal(in);
```

- On obtient une **`ArrayList<AnyNode>`**
  - org.exolab.castor.types.AnyNode
- Le nom de l'élément racine est indifférent

Ecriture de la collection

```
List orders = …;
Mapping mapping = new Mapping();
mapping.loadMapping("mapping.xml");
PrintWriter writer = …;

Marshaller marshall = new Marchaller(writer);
marshall.setRootElement("orders");
marshall.setMapping(mapping);
marshall.marshal(orders);
```


Traduction à partir d'un schéma de données

Castor source generator

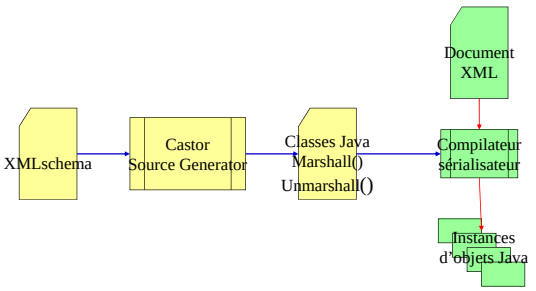
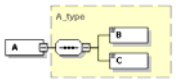


Schéma XML



- Un schéma XML

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="A_type">
    <xs:sequence>
      <xs:element name="B" type="xs:string"/>
      <xs:element name="C" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="A" type="A_type"/>
</xs:schema>
```

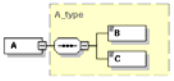

Générer des classes java à partir d'un Schema XML

- Classe **`org.exolab.castor.builder.SourceGenerator`**

```
SourceGenerator srcGen = new SourceGenerator();
srcGen.generateSource("Person.xsd","fr.ifsic.test");
```

- 4 Classes générées pour un type d'élément

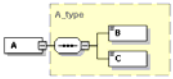
Classe abstraite générée pour le type complexe




```
public abstract class A_type implements
    java.io.Serializable {
  private java.lang.String _b;
  private java.lang.String _c;
  public A_type() {super();}
  public java.lang.String getB()
    {return this._b;}
  public java.lang.String getC()
    {return this._c;}
  public void setB(java.lang.String b)
    {this._b = b;}
  public void setC(java.lang.String c)
    {this._c = c;}
}
```

Classe générée pour l'élément <A>

```
public class A extends A_type
      implements java.io.Serializable {
  public A() {
    super();
  }
 public void marshal(java.io.Writer out) {…}
 public void marshal(org.xml.sax.ContentHandler
  handler) {…}
 public static A unmarshal(java.io.Reader reader)
   {…}
}
```

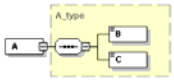


Classe de test

```
public class TestABC {
  public static void main(String[]
    args) {
    if (args.length!=1) {…erreur…}
    A aaa = new A();
    aaa.setB("bonjour");
    aaa.setC("monsieur");
    aaa.marshal(new OutputStreamWriter(
      new FileOutputStream(args[0])));
  }
}
```


Document xml produit

```
<?xml version="1.0" encoding="UTF-8"?>
<A>
  <B>bonjour</B>
  <C>monsieur</C>
</A>
```



Résumé des possibilités de Castor

Castor offre quatre possibilités de traduction

- À partir d'une classe
  - Traduction par défaut
  - Traduction assistée par un fichier `mapping.xml`
- À partir d'un Schéma XML
  - Traduction par défaut
  - Traduction assistée par un fichier `binding.xml`

Castor offre d'autres mécanismes

- Génération d'EJBs, échange avec un SGBDR
- échanges avec un annuaire LDAP …

D'autres convertions objets/XML

- XMLBeans
  - http://xmlbeans.apache.org/
- X-Stream
  - http://xstream.codehaus.org/
- EMF+Teneo
  - http://www.eclipse.org/modeling/emf/?project=teneo#teneo

Conclusion

- Génération de XML à partir d'un fichier texte non XML
  - Préférer DOM4J et Jdom à DOM si vous utilisez Java
- Génération de classe Java à partir d'un schéma de donnée (DTD ou Schéma XML)
  - Castor est un produit très abouti qui mérite le détour
- Lien avec sgbdr et j2EE
  - Castor semble être le bon candidat

Liens officiels

- DOM
  `http://www.w3.org/DOM/`
- JDOM
  `http://www.jdom.org/`
- DOM4J
  `http://dom4j.org/`
- Castor
  `http://www.castor.org`
- JAXB
  `http://java.sun.com/xml/jaxb/index.html`
- JaxMe
  `http://ws.apache.org/jaxme/`