

Java (VI)
Technologies de e-commerce :
manipulation des données et frameworks

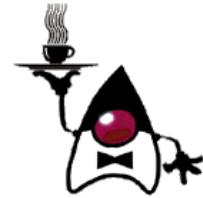
Claude VILVENS

claude.vilvens@hepl.be

<http://haute-ecole.provincedeliege.be/>

- I336 -
(2011-2012)

Sommaire



Introduction

Manipulations des données

XXXI. Le Java Beans Binding

1. La synchronisation de deux beans	3
2. Un Beans Binding classique	
2.1 Bindings, Binding et BindingGroup	3
2.2 La génération du Beans Binding dans NetBeans	6
3. Le Beans Binding avec données persistantes	
3.1 L'objectif de data binding	9
3.2 L'unité de persistance et la classe Entity	10
3.3 La réalisation interactive du Data Binding	15
3.4 Etude du code généré pour le Data Binding	18
4. Les mises à jour dynamiques	
4.1 Une liste d'observables	22
4.2 La modification d'une ligne de JTable	23
4.3 L'ajout d'une ligne dans la JTable	24
4.4 La suppression de lignes dans la JTable	26

XXXII. Les rowsets : un autre accès aux bases de données

1. Des JavaBeans en forme de ResultSets	29
2. L'interface RowSet	29
3. Des modèles d'implémentations	30
4. Un package d'implémentation des rowsets : utilisation basique	31
5. Les rowsets déconnectés	
5.1 Le principe d'utilisation	34
5.2 Le serveur de rowsets	35
5.3 Le client des rowsets	37
5.4 Les acteurs sous-jacents	43
6. Les événements liés aux rowsets	45

XXXIII. La librairie JFreeChart

1. Une bibliothèque pour graphiques statistiques	51
2. La représentation des données	52

3. L'exemple classique : le diagramme sectoriel	
3.1 Un Dataset particulier	53
3.2 Le JFreeChart correspondant	54
3.3 La classe factory	54
3.4 Les Plots	55
3.5 Le composant container visuel	55
3.6 L'application complète	56
3.7 La cascade des événements	57
3.8 Le diagramme de classes UML	59
4. L'utilisation d'une base de données	
4.1 Une classe facilitatrice	60
4.2 Les données dans une base de données	60
4.3 Le diagramme sectoriel à partir d'une base de données	61
5. D'autres graphiques classiques	
5.1 Les histogrammes comparés	25
5.2 Les graphiques linéaires d'évolution	29
6. Les graphiques de statistique à deux dimensions	
6.1 Le nuage de points	71
6.2 Les paramètres de régression et de corrélation	73
7. Les séries chronologiques	
7.1 Un axe des X avec des temps	76
7.2 Une graphique temporel à partir d'un base de données	79
8. Les graphiques dynamiques	82
9. Les graphiques statistiques dans les applications Web	
9.1 Une applet d'affichage	85
9.2 Une servlet d'affichage	85
9.3 Un servlet fournisseur d'image	88
9.4 Une servlet fournisseur d'étude statistique	89

XXXIV. Une petite introduction à Hibernate

1. Une bibliothèque pour graphiques statistiques	93
2. L'architecture d'Hibernate	
2.1 Les composants de la couche Hibernate	94
2.2 Les états des instances	95
3. En pratique avec Netbeans	95
4. Premier pas : la création d'un fichier de configuration	96
5. Tables et POJOs	
5.1 Création d'une classe à partir d'une table.	97
5.2 La génération des objets mappés	98
5.3 Utilisation directe du mapping	100
5.4 L'objet session et les transactions	101
5.5 Utilisation de l'objet mappé	103
6. La persistance : création d'une table à partir d'une classe	105
7. Les jointures	109
8. Deux optimisations d'Hibernate	
8.1 Les requêtes paramétrées	115
8.2 Le lazy loading	116

Authentifications et contrôle d'accès

XXXV. Les contrôles d'accès pour Tomcat et GlassFish

1. Un Tomcat intégré	117
2. Les utilisateurs de Tomcat	121
3. La définition de ressources protégées	
3.1 Une application Web banale	122
3.2 La définition des restrictions	122
3.3 L'utilisation de l'application Web	125
4. La definition de resources protégées : bis	126
5. L'utilitaire de digest de Tomcat	129
6. Les realms	
6.1 Un gestionnaire d'authentification	130
6.2 Le JDBC Realm	131
6.3 La définition du realm	132
6.4 L'utilisation de l'application Web	134
7. GlassFish	
7.1 Un serveur d'application	137
7.2 La console d'administration	138
7.3 Les domaines	140
7.4 Le déploiement d'une application Web sous GlassFish	140
8. Les realms JDBC sous GlassFish	
8.1 Divers gestionnaire d'authentification	143
8.2 Groupes et rôles	143
8.3 La base de données du JDBC Realm	144
8.4 La source de données d'authentification	145
8.5 La création du realm	149
8.6 La configuration de l'application Web	150
8.7 Le test de l'application	155

XXXVI. Java Authentication and Authorization Service

1. Le Java Authentication and Authorization Service	157
2. Les configurations d'authentification	157
3. Les entités d'authentification et d'autorisation	
3.1 Les subjects	159
3.2 Les principals	159
3.3 Les credentials	159
3.4 Utilisation des Subjects, Principals et Credentials	159
4. Le processus d'authentification selon JAAS	
4.1 La démarche globale	160
4.2 Le point d'entrée : le LoginContext	160
4.3 Utilisation du LoginContext dans une application	163
4.4 Le CallbackHandler	164
4.5 Le LoginModule	167
4.6 Un Principal	172

4.7 Exécuter un code en étant authentifié	173
4.8 Le diagramme de classes de JAAS	175
5. Le contexte Kerberos	
5.1 Objectifs et acteurs	177
5.2 Fonctionnement de Kerberos	177
5.3 Une implémentation Java	179

Technologies et frameworks de e-commerce

XXXVII. Une introduction au framework Struts

1. Un framework MVC	181
2. Une application Web élémentaire	182
3. Le JSP de login	185
4. Les librairies de tags de Struts	185
5. Le bean ActionForm associé au login	187
6. La classe métier Action	190
7. La servlet de contrôle	191
8. Le descripteur de déploiement de Struts	
8.1 La forme générale	192
8.2 L'action-mapping : les actions	193
8.3 L'action-mapping : les forwards	193
8.4 Les forwards globaux	193
8.5 Les form-beans	193
9. Le JSP de réponse	194
10. L'exécution de l'application Web	194
11. Un JSP de réponse négative	196
12. Le diagramme pseudo-UML récapitulatif	199
12. Les validations dans les formulaires	201

XXXVIII. Une introduction aux portails et aux portlets

1. La notion de portail	205
2. La JSR 168	207
3. Portail et portlets	208
4. Les caractéristiques d'une portlet	
4.1 Le cycle de vie	209
4.2 Portlets et servlets	210
4.3 Les modes des portlets	210
4.4 Les modes des fenêtres	211
5. Installation et intégration de JBoss Portal	211
6. Intégration des modules Portlets dans NetBeans	214
7. La création d'une portlet avec NetBeans	
7.1 Une application Web particulière	216
7.2 Le code de la portlet générée	218
7.3 Les fichiers de déploiements	220
7.4 Le déploiement de la portlet sur le serveur JBoss Portal	221

7.5 L'instanciation de la portlet	223
-----------------------------------	-----

XXXIX. Java Web Start

1. Une application Java téléchargée par HTTP	229
2. Le lancement d'une application JWS	230
3. Le gestionnaire d'application	233
4. Le fichier de déploiement d'une application JWS	
4.1 Les éléments de base	234
4.2 Un fichier de déploiement exemple	236
5. Analyse du trafic réseau	237
6. Les questions de sécurité	246
7. Les APIs JNLP	
7.1 Des services sécurisés	248
7.2 Le BasicService	248
7.3 Les services de fichiers	250
7.4 Le service de persistance et les muffins	253
8. Le développement d'une application pour Java Web Start sous Netbeans	255

Ouvrages consultés

Introduction



"La société de l'information" : c'est sous ce nom que s'est terminé le XXème siècle et qu'a débuté le millénaire suivant. Rien d'étonnant à ce que le monde du développement ait fait une place de plus en plus grande au traitement des données, nouant des relations de plus en plus étroites avec le monde des bases de données, le tout sur un arrière-plan de e-commerce.

En termes de développement orienté données, on peut parler de beaucoup de choses. Nous avons choisi ici un petit florilège de ces sujets amplement documentés sur Internet :

- ◆ le "**beans binding**" : bien connu des développeurs C#/.NET, il s'agit donc ici du "data binding" version Java;
- ◆ une librairie de traitement statistiques, nommée **JFreeChart** : maîtriser les chiffres, c'est tout simplement régner ...
- ◆ une introduction à **Hibernate** (implémentation évoluée de JPA).

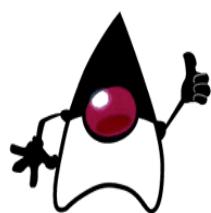
Ainsi armés, nous pouvons en venir à quelques questions de e-commerce proprement dit. Les applications **XML** et les **JSP** ont déjà été abordées dans le volume III. Mais bien d'autres sujets peuvent être abordés, comme l'évocation de certaines potentialités des serveurs d'applications **Tomcat** ou **Glassfish** (sujet développé plus largement dans les cours de Systèmes distribués), le retour des questions d'autorisation et d'authentification avec **JAAS**, ou encore la plate-forme **Struts**, qui facilite et systématisé la mise en place d'un modèle MVC. Et comment ignorer les successeurs des sites Web 1.0, ces fameux **portails** qui relèvent du Web sémantique dit "Web 2.0" ? Nous en dirons quelques mots.

Enfin, en termes de déploiement d'applications, on peut aussi considérer la solution **Java Web Start** qui semble vouloir allier les avantages des applets et des applications sans reprendre leurs inconvénients respectifs.

On le voit, on a le sentiment de partir dans tous les sens, parfois de manière un peu folle ... Mais c'est évidemment là l'intérêt de Java : ce langage permet d'aborder de très nombreuses questions informatiques avec élégance et rigueur, mais aussi "avec toute la simplicité et la générativité possibles" ;-)

Mais que ferait-on sans Java, n'est-il pas Mr Oracle ? Ainsi parlait le Sage ;-) ...

Claude Vilvens



Reprenons à nouveau le fil de nos idées ! Les chapitres sont numérotés en poursuivant la séquence du volume V. Nous commençons donc par le chapitre XXXI qui va nous ramener immédiatement vers le monde des données ...

XXXI. Le Java Beans Binding



Il y a des femmes dont l'infidélité est le seul lien qui les attache encore à leur mari.

(Sacha Guitry, Elles et toi)

1. La synchronisation de deux beans

La JSR 295 définit les spécifications du "Beans Binding". Bien clairement, il s'agit ici de lier une propriété d'un Java Bean à toute modification de la propriété d'un autre Java Bean (typiquement, un composant Swing) : on peut encore parler de "**synchronisation de deux beans**". Certes, ceci est déjà implantable au moyen des PropertyChangeSupport, mais une quantité non négligeable de code doit être rédigée. La librairie qui a développée (probablement, notamment, en réponse au célèbre "Data Binding" du monde C#/NET) vise,

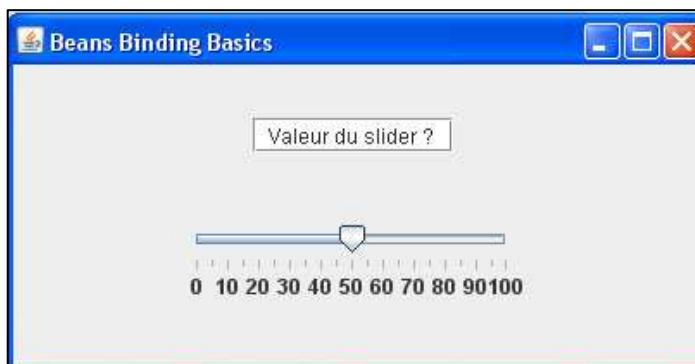
- ◆ d'une part, à systématiser le procédé, notamment en fournissant des outils adaptés à des environnements comme Netbeans,
- ◆ d'autre part à fournir d'autres outils qui permettront de lier un "Bean classique" à un "**Bean persistant**", c'est-à-dire un bean qui est en fait connecté sur des données se trouvant dans une base de données.

La librairie a été créée avec le JDK 1.5 comme support. Elle est distribuée sous licence LGPL, dans des packages org.jdesktop.* . Il est probable que certains de ses éléments finiront par être intégrés au JDK. Mais, de toute manière, l'outil n'est pas figé et le groupe de recherche-développement JSR 295 poursuit ses travaux.

2. Un Beans Binding classique

2.1 Bindings, Binding et BindingGroup

Considérons l'exemple classique (existant à n ($n \rightarrow \infty$) sur Internet) d'un slider dont la valeur encapsulée est affichée dans un textfield. Toute modification du slider doit donc modifier automatiquement cette valeur et, inversement, toute modification de la valeur doit déplacer le curseur du slider à la bonne position :



L'outil principal à utiliser pour implémenter ce comportement de "synchronisation de beans" est une implémentation de la classe abstraite **Binding<SS,SV,TS,TV>** (package org.jdesktop.beansbinding), dont les paramètres templates sont respectivement (et assez naturellement) :

- ◆ **SS**: la classe instanciée par l'objet source [**Source Synchronized object**] : initialement, pour nous, **JSlider**;
- ◆ **SV**: le type de la valeur de la propriété de la source [**Source property Value**] à laquelle on s'intéresse : ici, un nombre;
- ◆ **TS**: la classe instanciée par l'objet cible [**Target Synchronized object**] : initialement, pour nous, **JTextField**;
- ◆ **TV**: le type de la valeur de la propriété de la cible [**Target property Value**] qui doit être liée : ici, du texte.

La classe dérivée a pour tâche d'implémenter *la politique de synchronisation* mise en place. En pratique, on obtient une instance d'une telle classe en utilisant la classe factory **Bindings** (package org.jdesktop.beansbinding) qui contient des méthodes du type suivant :

```
public static <SS,SV,TS,TV> AutoBinding<SS,SV,TS,TV> createAutoBinding  
(AutoBinding.UpdateStrategy strategy, SS sourceObject, Property<SS,SV> sourceProperty,  
TS targetObject, Property<TS,TV> targetProperty)
```

C'est une méthode statique dont le prototype rendrait jaloux un concepteur C/C++ ;-) Donc :

- ◆ l'objet construit est une instance de la classe template **AutoBinding**, qui réalise donc une synchronisation automatique entre la source et la cible, en fonction d'une politique définie au moyen de l'une des trois constantes de la classe imbriquée **UpdateStrategy** :

- **READ_ONCE** : la synchronisation ne sera effectuée que lors de la liaison; il n'y aura donc plus de remises à jour des valeurs liées (version **snapshot**);
- **READ** : la cible restera synchronisée avec la source (version **unidirectionnelle**);
- **READ_WRITE** : la cible et la source sont synchronisées l'une par rapport à l'autre (version **bidirectionnelle**).

- ◆ on reconnaît bien sûr dans les 2^{ème} et 4^{ème} paramètres les objets sources et cibles;
- ◆ les 3^{ème} et 5^{ème} paramètres sont des instances de la classe **Property** dont le rôle est d'encapsuler l'accès à une propriété d'un Java Bean, accès dont l'apparence sera ainsi rendue uniforme de manière inaltérable; les deux paramètres templates représentent le bean considéré et le type de la propriété représentée.

Si donc les deux objets graphiques à lier sont **jSlider1** et **jTextField1** (c'est ce que NetBeans générera pour nous), nous allons donc utiliser cette méthode **createAutoBinding()**. Pour les deux paramètres **Property**, nous allons utiliser :

- 1) la classe **ELProperty**, qui est une petite-fille de **Property**; son intérêt est que l'on peut créer une instance de **Property** au moyen de la méthode

```
public static final <S,V> ELProperty<S,V> create(String expression)
```

qui attend comme paramètre une "**Expression Language**" dont la syntaxe générale est

```
 ${ expression }
```

- l'idée est que l'on demande ainsi d'insérer au point d'écriture l'évaluation de l'expression considérée (c'est fort pratique dans un JSP, par exemple). Dans notre cas, l'utilisation classique est

```
 ${ object.property }
```

Il est même possible

* de concaténer des propriétés pour créer un objet Property :

```
ELProperty.create("${prenom} ${nom}");
```

* de créer une Property booléenne testant une valeur :

```
BeanProperty.create("${enfant.age < 5}");
```

- mais on a changé de classe ? oui, car ...

2) la classe BeanProperty, qui est aussi une petite-fille de Property; c'est une simplification de la précédente pour le cas où il s'agit simplement de créer un objet Property à partir d'un nom de propriété de l'objet courant ou encore d'un nom de propriété d'une variable membre de cet objet :

```
BeanProperty.create("${nom}");  
BeanProperty.create("${enfant.prenom }");
```

Donc finalement, nous obtiendrons notre objet AutoBinding liant le slider au textfield par l'instruction :

Binding binding = Bindings.createAutoBinding

```
(AutoBinding.UpdateStrategy.READ_WRITE, jSlider1, ELProperty.create("${value}"),  
jTextField1, BeanProperty.create("text"));
```

(un objet JSlidet possède effectivement une propriété "value" puisque 'il dispose de la méthode getValue(), tandis qu'un TextField possède bien une propriété "text" vu sa méthode getText()).

Maintenant que nous disposons d'un objet de synchronisation (binding), il nous faut encore l'intégrer dans un objet **BindingGroup** (toujours du même package), au moyen de la méthode :

```
public final void addBinding(Binding binding)
```

Mais quel est l'intérêt ? Que cet objet groupe est celui qui initialise la liaison : l'objet binding est **le dépositaire des informations de cette liaison** et de l'implémentation, mais ne "démarre" pas dès son instanciation, ce qu'il réalise avec la méthode

```
public void bind()
```

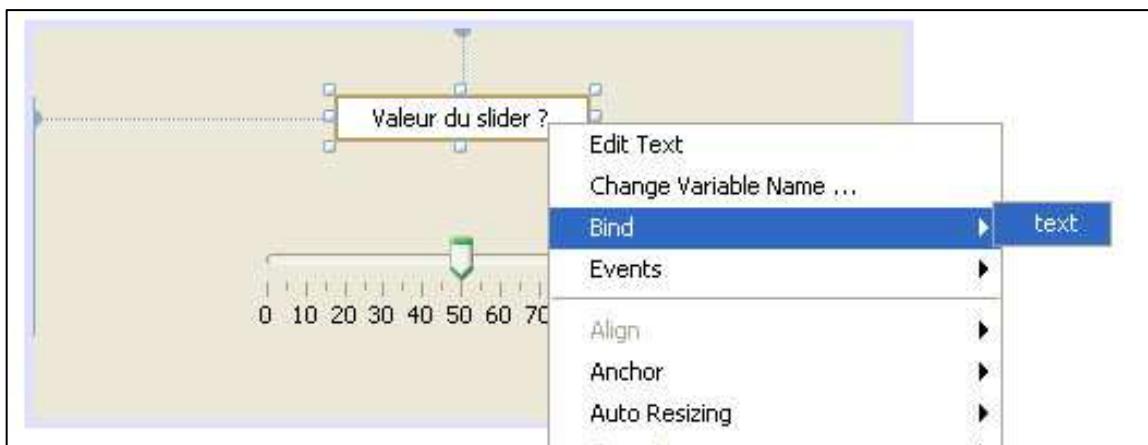
qui lie tous les objets qui doivent l'être. Donc, pour nous :

```
BindingGroup bindingGroup = new BindingGroup();
bindingGroup.addBinding(binding);
...
// en fin d'initialisation de l'application :
bindingGroup.bind();
```

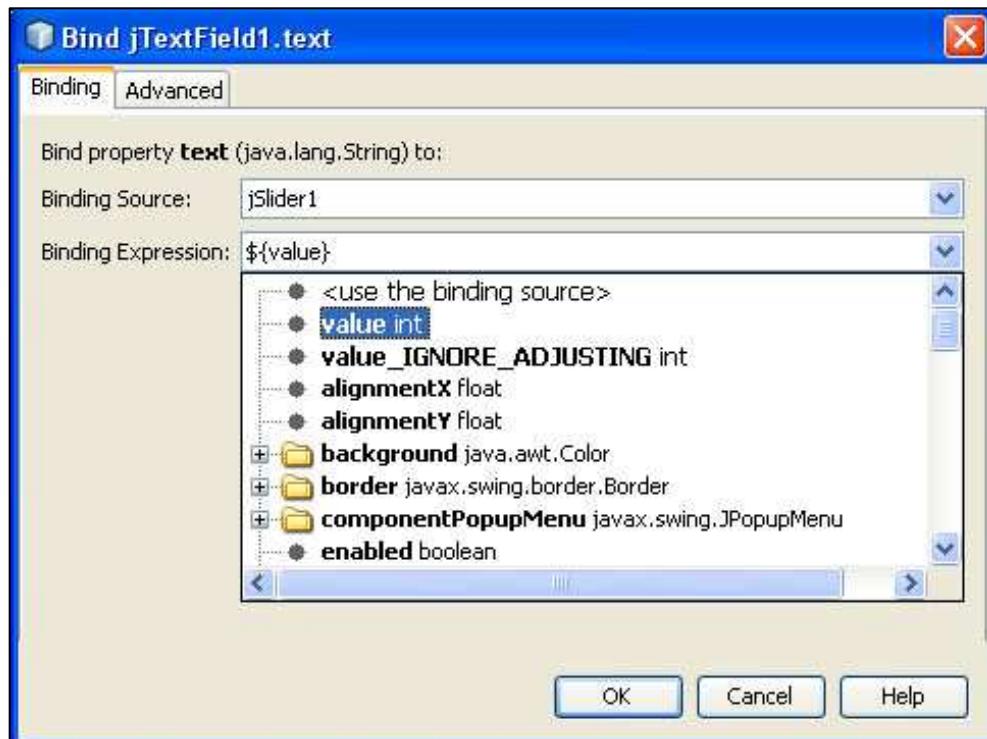
2.2 La génération du Beans Binding dans NetBeans

Le Beans Binding a été intégré dans Netbeans 6.* si bien que le code ci-dessus peut être généré par simple manipulation dans l'IDE. En effet, une fois les deux composants graphiques mis en place, il suffit de

- ◆ cliquer droit sur un des deux composants (ici, c'est le textfield, mais c'eût pu être le slider) :



- ◆ dans la boîte de dialogue ainsi obtenu, choisir l'objet slider dans la boîte combo et la propriété value :



- ♦ éventuellement vérifier par un nouveau clic droit sur un des deux composants :



- ♦ compiler et exécuter (on peut remarquer que la librairie Beans Binding – beansbinding-1.2.1.jar a été ajouté automatiquement au projet) :



Cela fonctionne :-) ! Le code généré est bien conforme à nos explications :

FenApp.java

```
package beansbindingbasique;

import java.awt.Desktop.Action;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

/**
 *
 * @author Vilvens
 */
public class FenApp extends javax.swing.JFrame
{
    public FenApp() { initComponents(); }

    private void initComponents()
    {
        bindingGroup = new org.jdesktop.beansbinding.BindingGroup();
```

```
jTextField1 = new javax.swing.JTextField();
jSlider1 = new javax.swing.JSlider();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Beans Binding Basics");

jTextField1.setHorizontalAlignment(javax.swing.JTextField.CENTER);

org.jdesktop.beansbinding.Binding binding =
    org.jdesktop.beansbinding.Bindings.createAutoBinding(
        org.jdesktop.beansbinding.AutoBinding.UpdateStrategy.READ_WRITE, jSlider1,
        org.jdesktop.beansbinding.ELProperty.create("${value}"), jTextField1,
        org.jdesktop.beansbinding.BeanProperty.create("text"));

...
bindingGroup.addBinding(binding);

jSlider1.setMajorTickSpacing(10); jSlider1.setMinorTickSpacing(5);
jSlider1.setPaintLabels(true); jSlider1.setPaintTicks(true);
...

bindingGroup.bind();

    pack();
}

public static void main(String args[])
{
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new FenApp().setVisible(true);
        }
    });
}

private javax.swing.JSlider jSlider1;
private javax.swing.JTextField jTextField1;
private org.jdesktop.beansbinding.BindingGroup bindingGroup;
}
```

3. Le Beans Binding avec données persistantes

3.1 L'objectif de data binding

Maintenant que le mécanisme de liaison (binding) de Java Beans a été assimilé, nous pouvons envisager de l'extrapoler à des Java Beans qui "se nourrissent" de données se trouvent dans des bases de données. En ce sens, nous effectuons ainsi un pas vers la persistance des Java Beans : car si ces beans trouvent les valeurs de leurs propriétés au moyen de bases de données (getXXX()), on peut aussi espérer que des modifications de ces propriétés se répercutent dans ces mêmes bases (setXXX()).

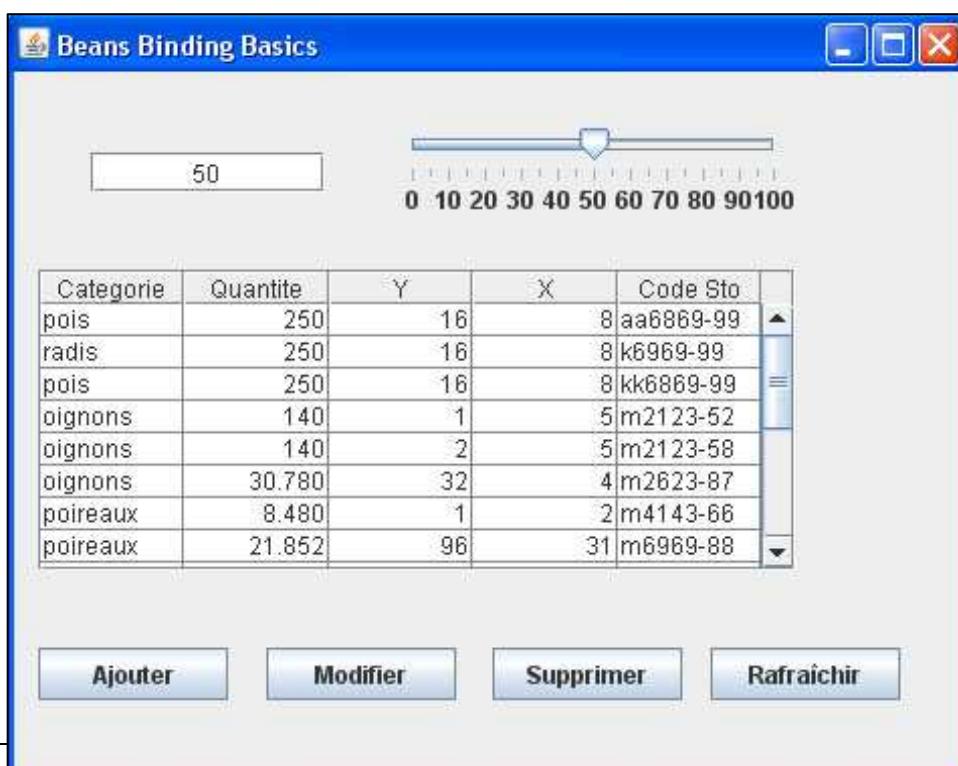
Considérons donc que nous disposons d'une base de données MySQL nommée mymarieinpres (comme d'habitude ;-)) qui comporte une table stocks :

Field...	Type *	codeSto *	x	y	quantite	categorie
codeSto	varchar(10)	aa6869-99	8	16	250	pois
x	int(11)	k6969-99	8	16	250	radis
y	int(11)	kk6869-99	8	16	250	pois
quantite	double	m2123-52	5	1	140	oignons
categorie	varchar(20)	m2123-58	5	2	140	oignons
		m2623-87	4	32	30780	oignons
		m4143-66	2	1	8480	poireaux

Nous aimeraisons que le contenu de cette table soit visible dans une JTable d'une application GUI classique, avec l'idée que ce composant graphique

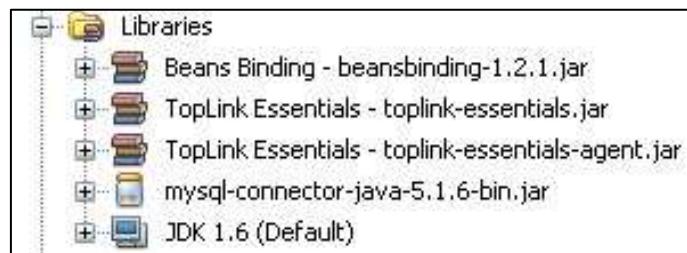
- ◆ pourra être mise à jour si des données de la table sont modifiées;
- ◆ permettra de réaliser des mises à jour avec répercussion sur la table.

Le GUI aurait donc l'aspect suivant :



En réalité, NetBeans fournit la possibilité de construire une telle "Java Desktop Application" avec la quasi-totalité du code généré automatiquement – mais de là à comprendre ce code généré, il y a un pas que seul peut franchir celui qui a au préalable développé lui-même une telle application ...

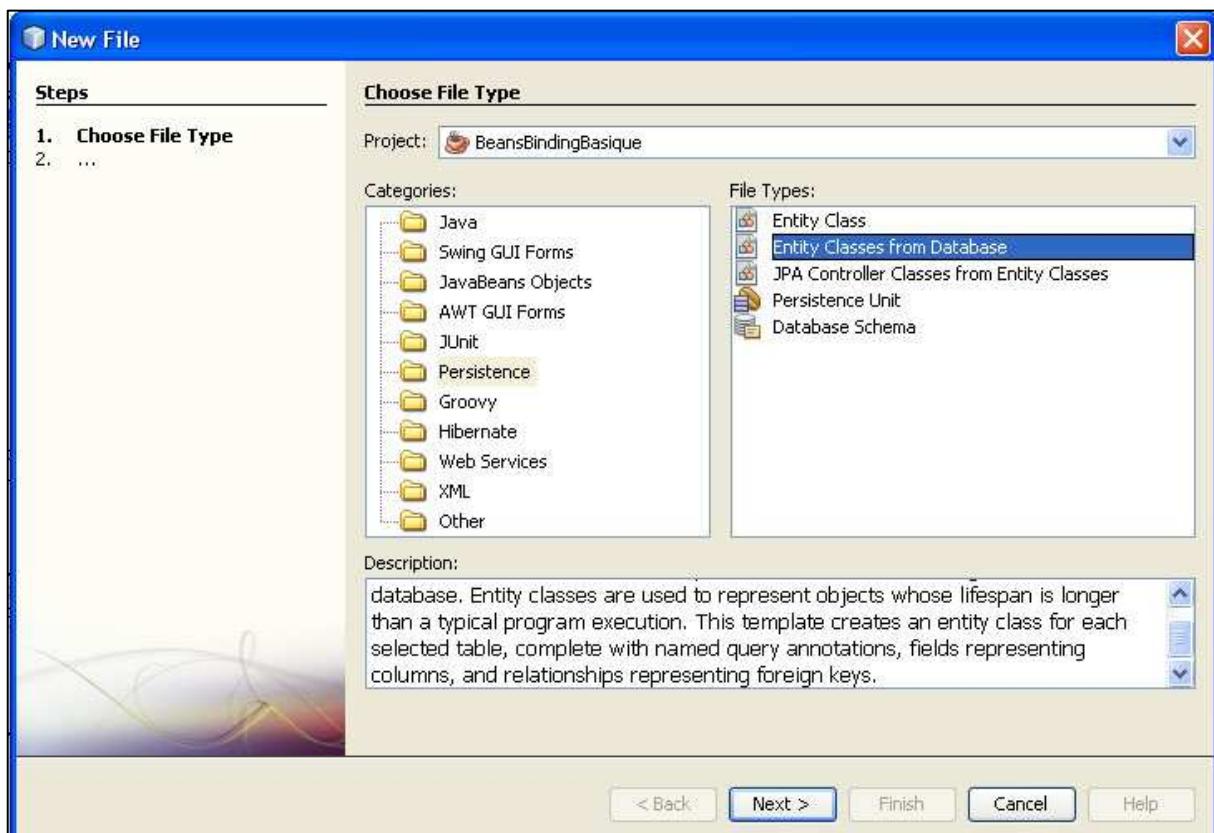
Nous créons donc bravement une application Java classique (projet et package beansbindingbasique), que nous munissons du GUI indiqué ci-dessus (classe FenApp). Mais, comme nous aurons besoin des mécanismes de Beans Binding et de Persistance, nous allons charger les librairies nécessaires :



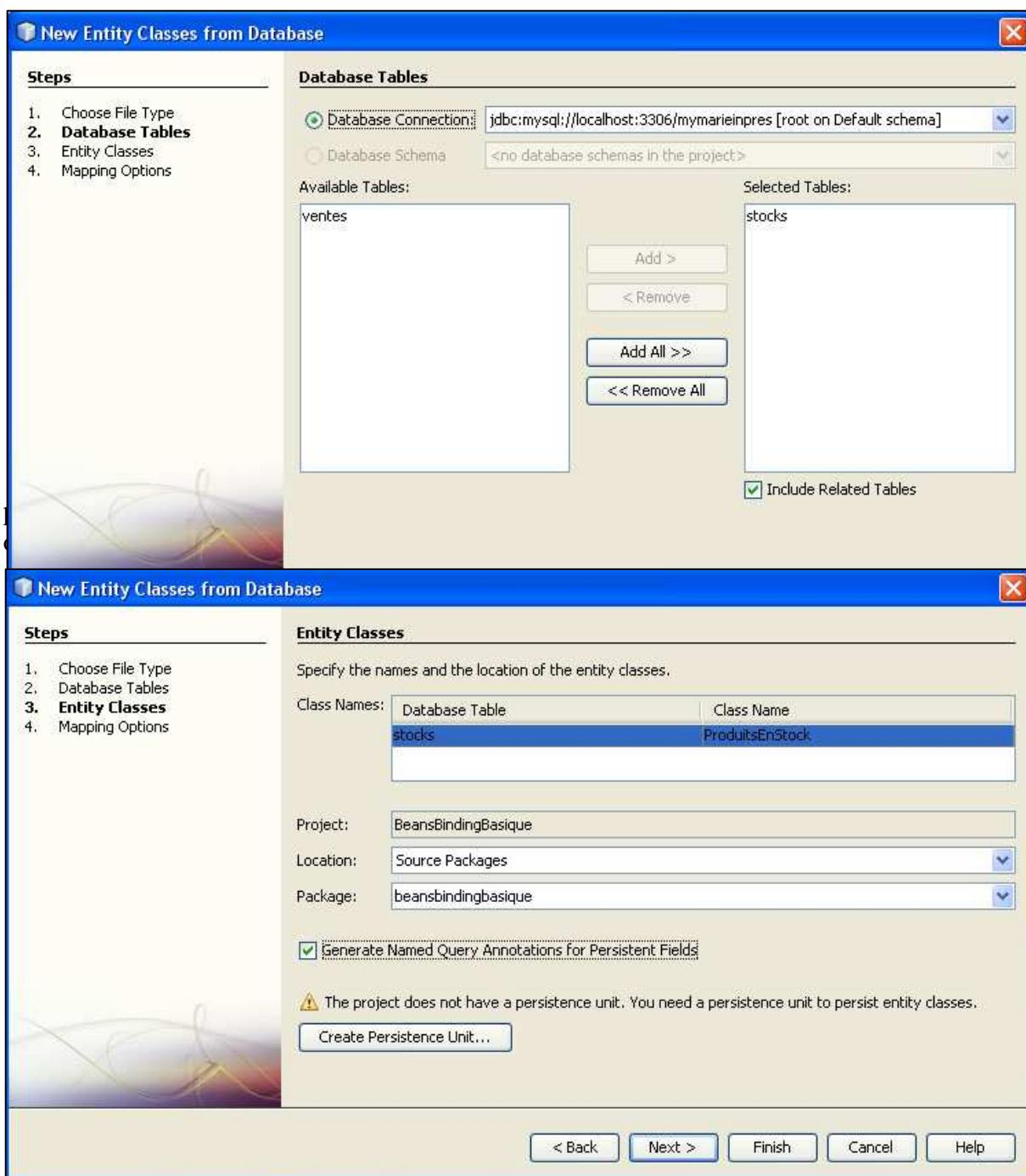
La seule librairie qui ne nous est pas familière (peut-être ...) est la "TopLink library" : il s'agit en fait de l'implémentation de référence du **JPA** (Java Persistence API) 2.0 (**JSR 317**) pour les EJB 3.0. En fait, c'est la version open source du produit Oracle correspondant, un ORM (Object/Relational Mapping) dont l'origine remonte aux années '90 et à Smalltalk.

3.2 L'unité de persistance et la classe Entity

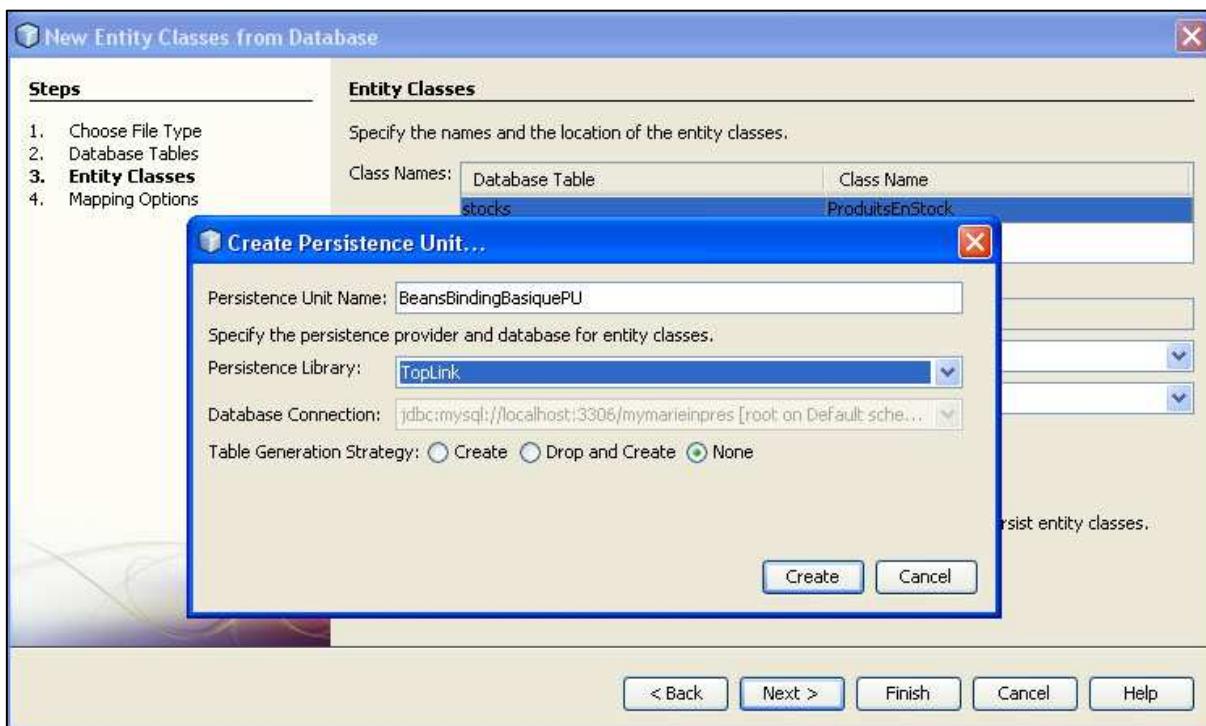
Nous allons à présent ajouter à notre projet une classe entité (Entity) chargée de réaliser le mapping avec la table stocks de la base mymarieinpres. Un clic droit sur le projet suivi de New nous permet de construire cette classe :



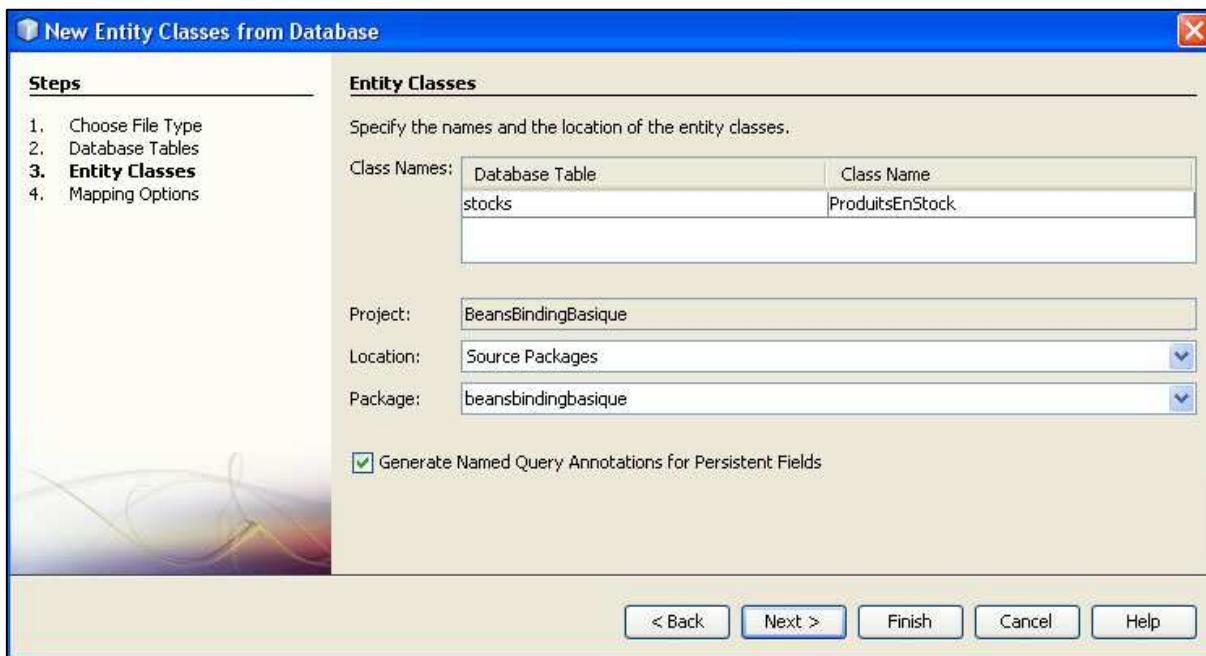
Nous choisissons la base de données :



A remarquer que nous sommes invités à créer une "Persistence Unit" : il s'agit en fait de métadonnées sur la liaison classe-table que nous allons construire. Ce genre d'unité de persistance (il peut donc y en avoir plusieurs par application) est mémorisé au sein d'un fichier XML nommé persistence.xml qui sera placé dans le répertoire META-INF du futur jar de l'application. On trouvera dans ce fichier, par exemple, les informations nécessaires à la connexion à la base de données ou le nom de la classe qui va être construite. Précisément, terminons d'abord la construction de celle-ci avant de voir le contenu de persistence.xml : nous demandons la construction de l'unité :



puis nous créons la classe qui sera mappée sur la table stocks : elle se nommera ProduitsEnStock et nous utiliserons, pour décrire les correspondances variables membres/champ de tuple, les annotations dans le code Java plutôt que d'utiliser une description par xml :



Après avoir accepté les options par défaut (notamment le type de Collection pour contenir les tuples – typiquement, une implémentation de l'interface `java.util.List`, qui contendra donc des objets **ProduitsEnStocks**), nous terminons et obtenons le résultat suivant :

a) comme annoncé, un fichier persistence.xml :

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="BeansBindingBasiquePU" transaction-type="RESOURCE_LOCAL">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <class>beansbindingbasique.ProduitsEnStock</class>
    <properties>
      <property name="toplink.jdbc.user" value="root"/>
      <property name="toplink.jdbc.password" value="xxxxxxxxxxxx" />
      <property name="toplink.jdbc.url" value="jdbc:mysql://localhost:3306/mymarieinpres" />
      <property name="toplink.jdbc.driver" value="com.mysql.jdbc.Driver" />
    </properties>
  </persistence-unit>
</persistence>
```

b) une classe ProduitsEnStock annotée comme entité :

ProduitsEnStock.java

```
package beansbindingbasique;

import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Transient;

@Entity
@Table(name = "stocks")
@NamedQueries({ @NamedQuery(name = "ProduitsEnStock.findAll", query = "SELECT p FROM ProduitsEnStock p"), @NamedQuery(name = "ProduitsEnStock.findByCodeSto", query = "SELECT p FROM ProduitsEnStock p WHERE p.codeSto = :codeSto"),
  @NamedQuery(name = "ProduitsEnStock.findByX", query = "SELECT p FROM ProduitsEnStock p WHERE p.x = :x"), @NamedQuery(name =
  "ProduitsEnStock.findByY", query = "SELECT p FROM ProduitsEnStock p WHERE p.y = :y"), @NamedQuery(name = "ProduitsEnStock.findByQuantite", query = "SELECT p FROM ProduitsEnStock p WHERE p.quantite = :quantite"), @NamedQuery(name =
  "ProduitsEnStock.findByCategorie", query = "SELECT p FROM ProduitsEnStock p WHERE p.categorie = :categorie") })
```

```

public class ProduitsEnStock implements Serializable {
    @Transient
    private PropertyChangeSupport changeSupport = new PropertyChangeSupport(this);
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "codeSto")
    private String codeSto;
    @Column(name = "x")
    private Integer x;
    @Column(name = "y")
    private Integer y;
    @Column(name = "quantite")
    private Double quantite;
    @Column(name = "categorie")
    private String categorie;

    public ProduitsEnStock() { }
    public ProduitsEnStock(String codeSto) { this.codeSto = codeSto; }

    public String getCodeSto() { return codeSto; }
    public void setCodeSto(String codeSto)
    {
        String oldCodeSto = this.codeSto;
        this.codeSto = codeSto;
        changeSupport.firePropertyChange("codeSto", oldCodeSto, codeSto);
    }

    public Integer getX() { return x; }
    public void setX(Integer x)
    {
        Integer oldX = this.x;
        this.x = x;
        changeSupport.firePropertyChange("x", oldX, x);
    }
    // etc pour Y, quantite et categorie ...

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (codeSto != null ? codeSto.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof ProduitsEnStock)) {
            return false;
        }
    }
}

```

```
ProduitsEnStock other = (ProduitsEnStock) object;
if ((this.codeSto == null && other.codeSto != null) || (this.codeSto != null &&
!this.codeSto.equals(other.codeSto))) {
    return false;
}
return true;
}

@Override
public String toString() {
    return "beansbindingbasique.ProduitsEnStock[codeSto=" + codeSto + "]";
}

public void addPropertyChangeListener(PropertyChangeListener listener)
{ changeSupport.addPropertyChangeListener(listener); }

public void removePropertyChangeListener(PropertyChangeListener listener)
{ changeSupport.removePropertyChangeListener(listener); }
```

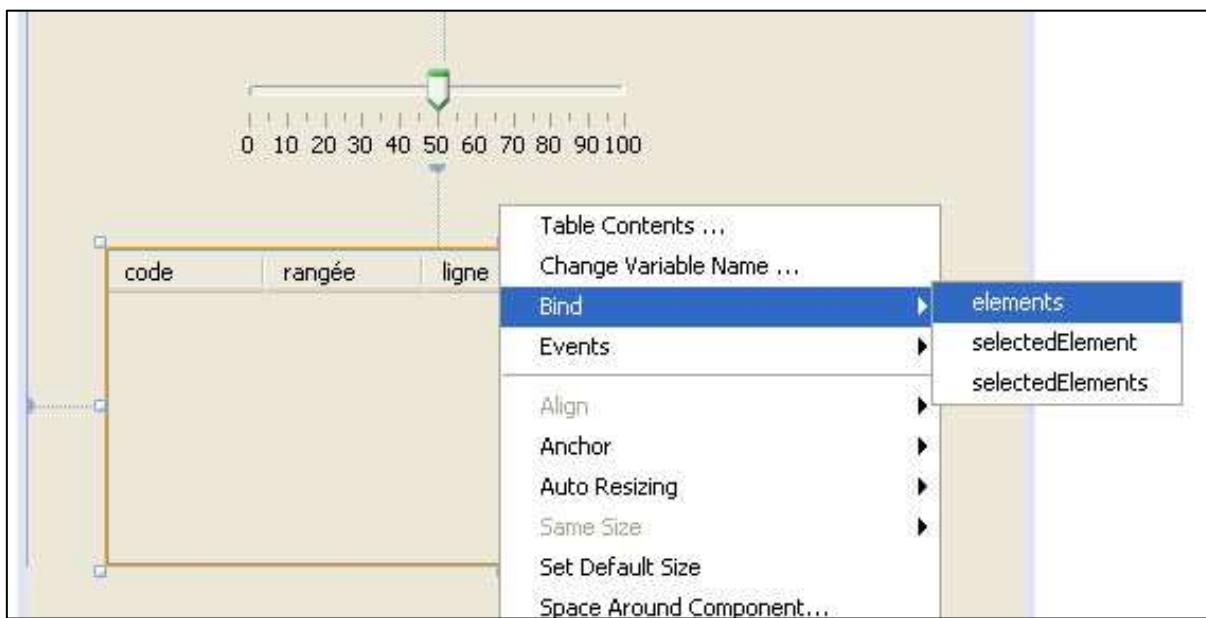
On remarquera :

- ◆ le fait que la classe est annotée comme étant une Entity, donc persistante et en correspondance avec une table stocks;
- ◆ les requêtes nommées (NamedQueries) qui sont les SELECT basiques;
- ◆ la variable membre instance de **PropertyChangeSupport** (nous sommes bien dans une logique de Java Beans), dont les services sont utilisés dans les méthodes setXXX() avec un appel à firePropertyChange(); on devine donc qu'il ne restera plus qu'à désigner le(s) listener(s) intéressé(s);
- ◆ le fait que la colonne codeSto, clé primaire, sera à la base de l'identifiant unique de chaque objet;
- ◆ la relation one-to-one entre les champs de la table et les variables membres de la classe;
- ◆ la redéfinition des méthodes héritées d'Object equals(), hashCode() et **toString()**.

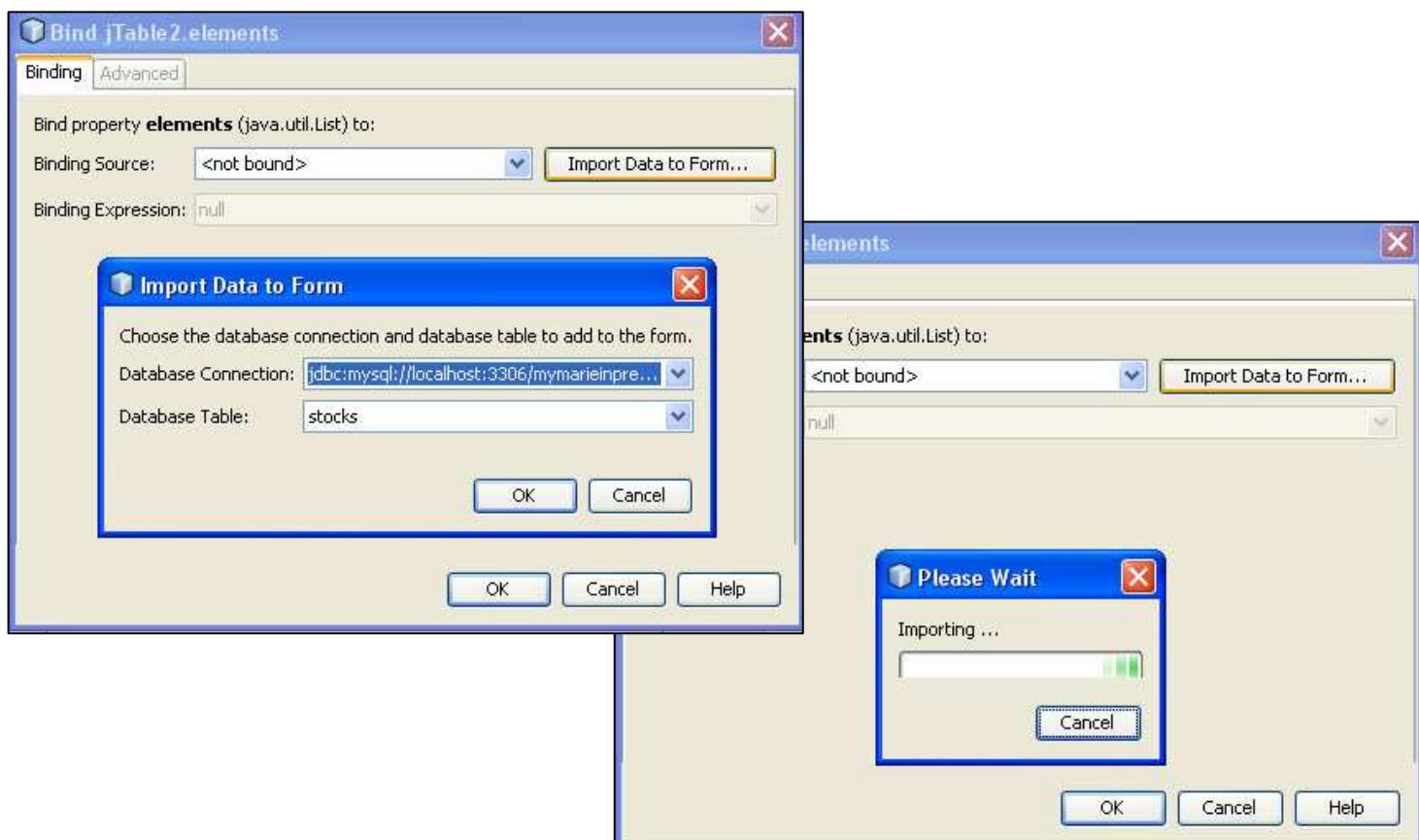
3.3 La réalisation interactive du Data Binding

Nous disposons donc à présent d'un Java Bean un peu particulier qui est cette classe ProduitsEnStock, reliée à la table stocks de la base. C'est en fait par son intermédiaire que nous allons pouvoir lier la JTable de notre interface graphique à la table stocks en question.

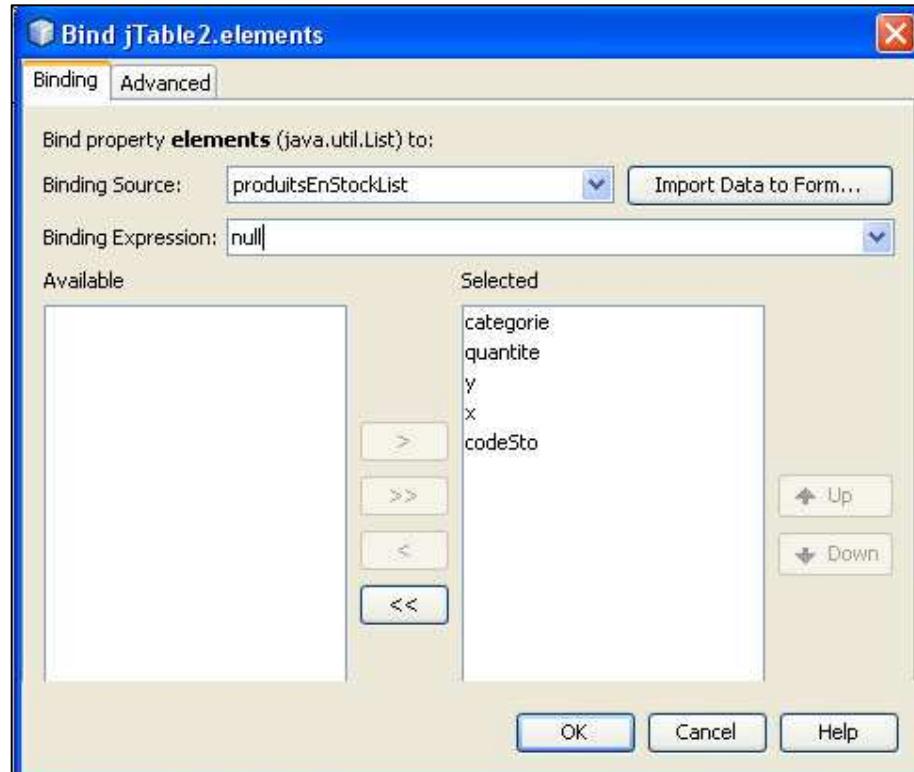
Voyons comment procéder. Dans l'outil de design graphique, nous choisissons Bind dans le menu contextuel obtenu par un clic droit sur la JTable :



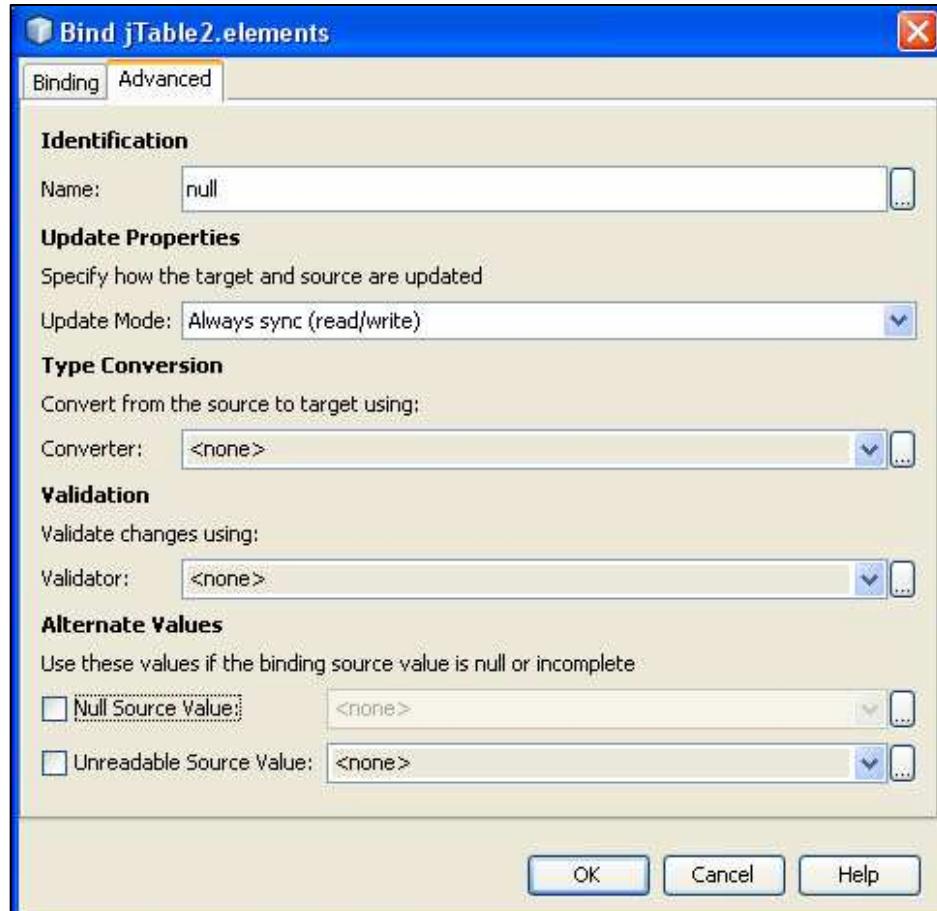
Dans la boîte de dialogue obtenue, nous allons provoquer la création de nos objets mappés en sollicitant le bouton "Import Data to Form" qui va nous permettre de désigner notre source de données :



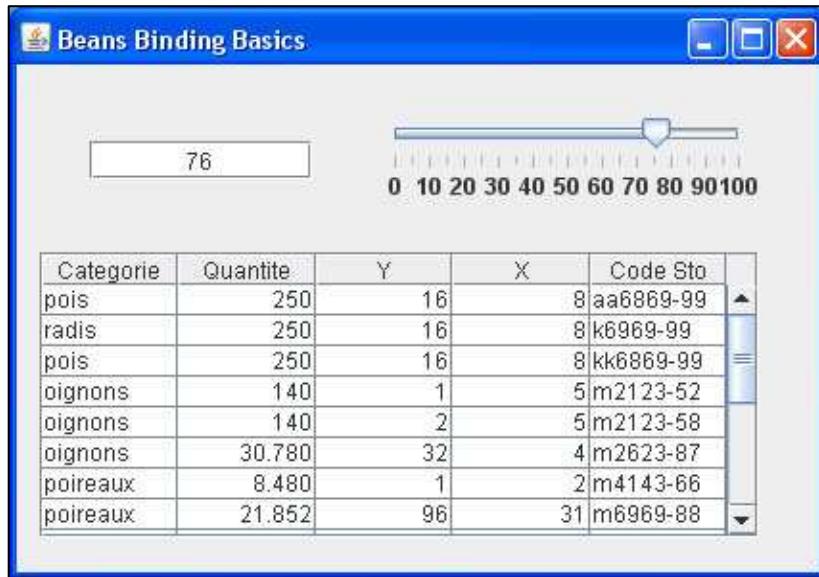
On nous propose alors de construire un objet produitsEnStockList (d'après bien sûr le nom de notre classe ProduitsEnStock qui assure le mapping) – à remarquer que l'on peut éviter certaines colonnes en les rejetant dans la zone "Available" de gauche :



L'onglet advanced permet de spécifier des caractéristiques plus pointues du binding, comme des validations ou des conversions – surtout, on reconnaît la stratégie d'update (ici, read-write) :



A l'exécution, pas de problème : la JTable est bien remplie avec les données de la table stock ☺ :



Mais que s'est-il réellement passé ?

3.4 Etude du code généré pour le Data Binding

Penchons-nous finalement sur le code généré dans notre classe FenApp par ces diverses manipulations :

FenApp.java

```
package beansbindingbasique;

import java.awt/Desktop.Action;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

/**
 *
 * @author Vilvens
 */

public class FenApp extends javax.swing.JFrame
{
    public FenApp() { initComponents(); }

    @SuppressWarnings("unchecked")
    private void initComponents()
    {
        bindingGroup = new org.jdesktop.beansbinding.BindingGroup();
```

```

// a)
BeansBindingBasiquePUEntityManager = java.beans.Beans.isDesignTime() ? null :
javax.persistence.Persistence.createEntityManagerFactory("BeansBindingBasiquePU").
createEntityManager();

// b)
produitsEnStockQuery = java.beans.Beans.isDesignTime() ? null :
BeansBindingBasiquePUEntityManager.createQuery("SELECT p FROM
ProduitsEnStock p");

// c)
produitsEnStockList = java.beans.Beans.isDesignTime() ?
java.util.Collections.emptyList() :
org.jdesktop.observablecollections.ObservableCollections.
observableList(produitsEnStockQuery.getResultList());

jTextField1 = new javax.swing.JTextField();
jSlider1 = new javax.swing.JSlider();
jScrollPane2 = new javax.swing.JScrollPane();
jTable2 = new javax.swing.JTable();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Beans Binding Basics");

jTextField1.setHorizontalAlignment(javax.swing.JTextField.CENTER);

// le binding précédent du Slider et du TextField
org.jdesktop.beansbinding.Binding binding =
    org.jdesktop.beansbinding.Bindings.createAutoBinding
        (org.jdesktop.beansbinding.AutoBinding.UpdateStrategy.READ_WRITE, jSlider1,
         org.jdesktop.beansbinding.ELProperty.create("${value}"), jTextField1,
         org.jdesktop.beansbinding.BeanProperty.create("text"));
bindingGroup.addBinding(binding);

jSlider1.setMajorTickSpacing(10);
jSlider1.setMinorTickSpacing(5);
jSlider1.setPaintLabels(true);
jSlider1.setPaintTicks(true);

// d) le binding avec persistance
org.jdesktop.swingbinding.JTableBinding jTableBinding =
    org.jdesktop.swingbinding.SwingBindings.createJTableBinding
        (org.jdesktop.beansbinding.AutoBinding.UpdateStrategy.READ_WRITE,
         produitsEnStockList, jTable2);
org.jdesktop.swingbinding.JTableBinding.ColumnBinding columnBinding =
    jTableBinding.addColumnBinding(org.jdesktop.beansbinding.ELProperty.
        create("${categorie}"));
columnBinding.setColumnName("Categorie");
columnBinding.setColumnClass(String.class);

```

```

columnBinding =
    jTableBinding.addColumnBinding(org.jdesktop.beansbinding.ELProperty.
        create("${quantite}"));
    columnBinding.setColumnName("Quantite");
    columnBinding.setColumnClass(Double.class);

columnBinding =
    jTableBinding.addColumnBinding(org.jdesktop.beansbinding.ELProperty.
        create("${y}"));
    columnBinding.setColumnName("Y");
    columnBinding.setColumnClass(Integer.class);

columnBinding =
    jTableBinding.addColumnBinding(org.jdesktop.beansbinding.ELProperty.
        create("${x}"));
    columnBinding.setColumnName("X");
    columnBinding.setColumnClass(Integer.class);

columnBinding =
    jTableBinding.addColumnBinding(org.jdesktop.beansbinding.ELProperty.
        create("${codeSto}"));
    columnBinding.setColumnName("Code Sto");
    columnBinding.setColumnClass(String.class);

bindingGroup.addBinding(jTableBinding);
jTableBinding.bind();
jScrollPane2.setViewportView(jTable2);
bindingGroup.bind();
pack();
}

public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new FenApp().setVisible(true);
        }
    });
}

private javax.persistence.EntityManager BeansBindingBasiquePUEntityManager;

private javax.swing.JScrollPane jScrollPane2;
private javax.swing.JSlider jSlider1;
private javax.swing.JTable jTable2;
private javax.swing.JTextField jTextField1;

private java.util.List<beansbindingbasique.ProduitsEnStock> produitsEnStockList;
private javax.persistence.Query produitsEnStockQuery;
private org.jdesktop.beansbinding.BindingGroup bindingGroup;
}

```

On constate donc

a) la présence d'un **EntityManager** : il s'agit bien évidemment d'un objet dont le rôle est de permettre l'interaction avec le contexte de persistance : création ou suppression d'une entité persistante (on parle encore d"*"objet managé"*), gestion des requêtes, gestion des transactions, pose de verrous, etc. Il s'agit en fait d'un interface. On obtient un objet l'implémentant en utilisant

- ◆ une classe factory qui implémente l'interface **EntityManagerFactory** : on l'obtient avec la factory de la classe Persistence :

```
public static EntityManagerFactory  
createEntityManagerFactory(String persistenceUnitName)
```

- bien clairement, cette factory est donc mise ne relation avec l'unité de persistance définie dans le contexte;

- ◆ la méthode de l'objet ainsi obtenu :

```
public EntityManager createEntityManager()
```

b) la requête générée par l'EntityManager au moyen de sa méthode

```
public Query createQuery(java.lang.String qlString)
```

- l'interface **Query** est assez proche de son homologue JDBC; ainsi, il possède les méthodes

```
int executeUpdate()  
public java.util.List getResultList()  
etc
```

c) la manière de récupérer la liste de tuples qui va nourrir la JTable : tout se fait par la méthode

```
public static <E> ObservableList<E> observableList(java.util.List<E> list)
```

de la classe **ObservableCollections** (package org.jdesktop.observablecollections) qui, à partir d'une liste, en fait une **ObservableList** (interface dérivé de List<E>) : une telle liste a ceci de plus qu'en cas de modification de ses éléments, elle notifie tous les listeners qu'elle a enregistré avec :

```
void addObservableListListener(ObservableListListener listener)  
void removeObservableListListener(ObservableListListener listener)
```

Ceci est un point important : nous devrons nous en souvenir pour l'implémentation des mises à jour ... On remarquera en passant la méthode

```
boolean supportsElementPropertyChanged()
```

qui permet de savoir si il est nécessaire d'installer un listener pour chaque élément de la liste.
Pour notre culture, on peut encore savoir qu'un doit posséder 4 méthodes :

```
void listElementsAdded(ObservableList list, int index, int length)
void listElementsRemoved(ObservableList list, int index, List oldElements)
void listElementReplaced(ObservableList list, int index, Object oldElement)
void listElementPropertyChanged(ObservableList list, int index)
```

d) le binding de la liste avec la JTable réalisé au moyen d'un **JTableBinding** (package org.jdesktop.swingbinding.JTableBinding), cas particulier de l'AutoBinding que nous connaissons déjà :

```
public final class JTableBinding<E,SS,TS>
extends AutoBinding<SS,java.util.List<E>,TS,java.util.List>
```

Dédicacée aux JTable (chaque objet de la liste va correspondre à une ligne de la JTable), cette classe possède la méthode :

```
public JTableBinding.ColumnBinding addColumnBinding(Property<E,?> columnProperty)
```

qui construit un objet instance de la classe imbriquée ColumnBinding pour chaque colonne (bien sûr, il lie une propriété d'un élément de la liste à une colonne de la JTable).

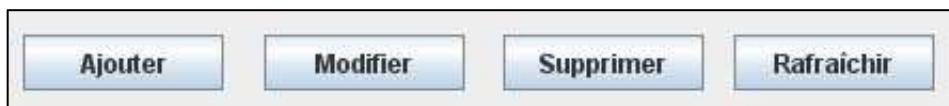
4. Les mises à jour dynamiques

4.1 Une liste d'observables

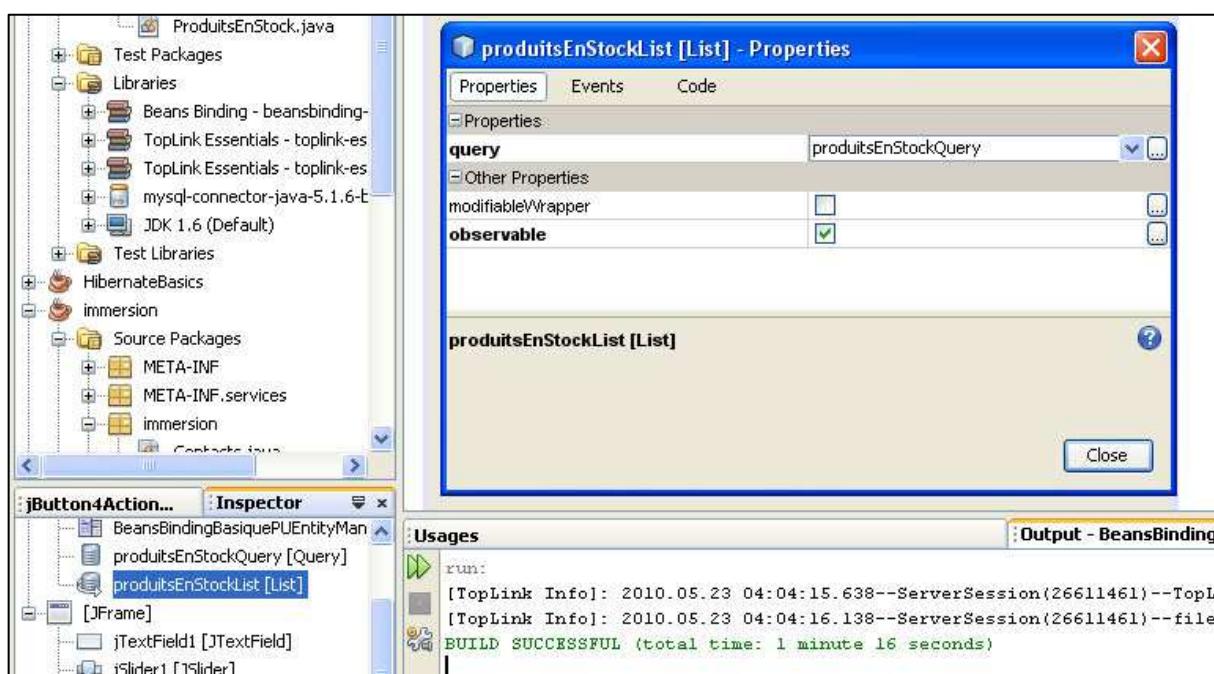
Comment notre liaison va-t-elle se comporter lorsqu'on effectue des mises à jour

- ◆ depuis la JTable – le SGBD va-t-il en être avisé pour rafraîchir la table ?
- ◆ directement dans la base de données – la JTable en prendre-t-elle conscience ?

Pour parvenir à cet objectif, ajoutons à notre GUI 4 boutons au rôle bien clair :



Mais avant tout chose, **une précaution indispensable** : il nous préciser que notre liste (qui est en fait une instance de ObservableCollections\$ObservableListImpl), chaînon entre la JTable et la table de la base de données, a sa propriété "observable" à true (une List de java.util n'a pas ce genre de propriété) :



car ainsi les événements de modifications provoqueront bien les notifications aux listeners mis en place par la librairie Toplink (on aurait pu s'attendre à ce comportement par défaut).

4.2 La modification d'une ligne de JTable

Après avoir modifié une (ou plusieurs) lignes dans la JTable, il suffit d'ouvrir une transaction (si il n'y en a pas une en cours) en utilisant notre EntityManager pour appeler sa méthode :

```
public EntityManager getEntityManager()
```

L'objet récupéré possède, notamment, les méthodes :

- ◆ public boolean **isActive()** : permet de savoir si une transaction est en cours;
- ◆ public void **begin()** : pour démarrer une transaction – bien sûr, une exception IllegalStateException sera lancée si isActive() est à true;
- ◆ public void **commit()** : termine la transaction - une exception IllegalStateException sera lancée si isActive() est à false;
- ◆ public void **rollback()** : pour annuler une transaction - une exception IllegalStateException sera lancée si isActive() est à false;

Le traitement du bouton "Modifier" est alors fort simple :

dans FenApp.java

```
...
private void ButtonModifierActionPerformed(java.awt.event.ActionEvent evt)
{
    EntityTransaction tr = BeansBindingBasiquePUEntityManager.getTransaction();
    if (! tr.isActive()) tr.begin();
    tr.commit();
}
```

et toute modification dans la JTable suivie d'un appui sur le bouton se fera sentir dans la base de données, comme on peut le vérifier par un select * sur la table visée.

4.3 L'ajout d'une ligne dans la JTable

Nous commençons par créer une boîte de dialogue des plus simples : 5 zones d'entrée pour confectionner un nouveau produit (une instance de ProduitsEnStock) lorsque l'on appuie sur son bouton Ok :

DialogNouveauProduit.java

```
package beansbindingbasique;

public class DialogNouveauProduit extends javax.swing.JDialog
{
    private ProduitsEnStock nouveauProduit;
    public DialogNouveauProduit(java.awt.Frame parent, boolean modal)
    {
        super(parent, modal);
        initComponents();
        nouveauProduit=null;
    }

    private void initComponents() { ... pack(); ... }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
    {
        setNouveauProduit(new ProduitsEnStock());
        getNouveauProduit().setCategorie(this.jTextField1.getText());
        getNouveauProduit().setQuantite(Double.valueOf(this.jTextField2.getText()));
        getNouveauProduit().setX(Integer.valueOf(this.jTextField3.getText()));
        getNouveauProduit().setY(Integer.valueOf(this.jTextField4.getText()));
        getNouveauProduit().setCodeSto(this.jTextField5.getText());
        this.setVisible(false);
    }

    private javax.swing.JButton jButton1; private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1; private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3; private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JTextField jTextField1; private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3; private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;

    public ProduitsEnStock getNouveauProduit() { return nouveauProduit; }
    public void setNouveauProduit (ProduitsEnStock nouveauProduit)
    {
        this.nouveauProduit = nouveauProduit;
    }
}
```

Bien sûr, cette boîte apparaît lorsque l'on appuie sur le bouton Ajouter de la fenêtre principale. Après avoir créé un objet ProduitsEnStock à partir des données entrées dans la boîte de dialogue, nous allons :

a) **enregistrer ces nouvelles données dans la table** en usant du mécanisme de persistance; pour cela, nous allons :

- ◆ débutter une transaction (si nécessaire);
- ◆ appeler la méthode

void **persist**(java.lang.Object entity)

de l'EntityManager en lui passant l'objet ProduitsEnStock que l'on vient de créer : on se souviendra en effet que cet EntityManager a pour mission de permettre l'interaction avec le contexte de persistance, notamment en créant ou détruisant les entités persistantes

- ◆ refermer la transaction pour propager l'objet managé à la base de données;

b) **mettre à jour notre JTable**, cela en tenant compte du fait que ce n'est pas une JTable "normale" (il suffit pour s'en convaincre de vérifier la nature du modèle de cette JTable : il s'agit d'un JTableBinding\$BindingTableModel !) : elle se remplit par le truchement de la liste intermédiaire produitsEnStockList; donc, il faut :

- ◆ ajouter le nouveau produit à cette liste;
- ◆ mettre l'affichage à jour grâce aux méthodes de JTable :

* public void **setRowSelectionInterval**(int index0, int index1)

sélectionne une zone de la table, dans le but de déterminer le rectangle au moyen de la méthode suivante :

* public Rectangle **getCellRect**(int row, int column, boolean includeSpacing)

fournit un rectangle correspondant à la ligne et la colonne spécifiées, contour compris si le 3ème paramètre est à true; ce rectangle étant déterminé, il reste à le faire afficher correctement par

* public void **scrollRectToVisible**(Rectangle aRect)

provoque le déroulement sur le rectangle passé en paramètre

En résumé :

dans FenApp.java

```
...
private void ButtonAjouterActionPerformed(java.awt.event.ActionEvent evt)
{
    DialogNouveauProduit dnp = new DialogNouveauProduit(this, true);
    dnp.setVisible(true);
    ProduitsEnStock t = dnp.getNouveauProduit();
```

```
// a)
EntityTransaction tr = BeansBindingBasiquePUEntityManager.getTransaction();
if (! tr.isActive()) tr.begin();
BeansBindingBasiquePUEntityManager.persist(t);
tr.commit();

// b)
produitsEnStockList.add(t);
int row = produitsEnStockList.size()-1;
this.jTable2.setRowSelectionInterval(row, row);
jTable2.scrollRectToVisible(jTable2.getCellRect(row, 0, true));
}
```

...

4.4 La suppression de lignes dans la JTable

Après avoir sélectionné une ou plusieurs lignes dans la JTable, l'appui sur le bouton Supprimer doit avoir pour effet de faire disparaître ces lignes et de la JTable et de la table correspondante de la base données. Pour cela :

- ◆ on prépare une liste (une ArrayList) dimensionnée sur le nombre de lignes sélectionnées; on aura récupérer les numéros de ces lignes avec la méthode de JTable

public int[] getSelectedRows()

- ◆ on débute une transaction (avec les précautions d'usage) en utilisant notre EntityManager;
- ◆ on récupère les ProduitsEnStock correspondant aux lignes sélectionnées en allant les rechercher dans la liste intermédiaire produitsEnStockList; pour cela, on passe d'abord par le modèle au moyen de la méthode

public int convertRowIndexToModel(int viewRowIndex)

puis par la méthode

E get(int index)

de la liste;

- ◆ pour chacun de ces produits, on demande à l'EntityManager de propager la suppression, ceci au moyen de sa méthode :

void remove(java.lang.Object entity)

- ◆ on n'oubliera pas, aussi, de détruire les produits sélectionnés de produitsEnStockList (sans quoi on aurait évidemment une incohérence);
- ◆ après quoi il reste à refermer la transaction.

En résumé :

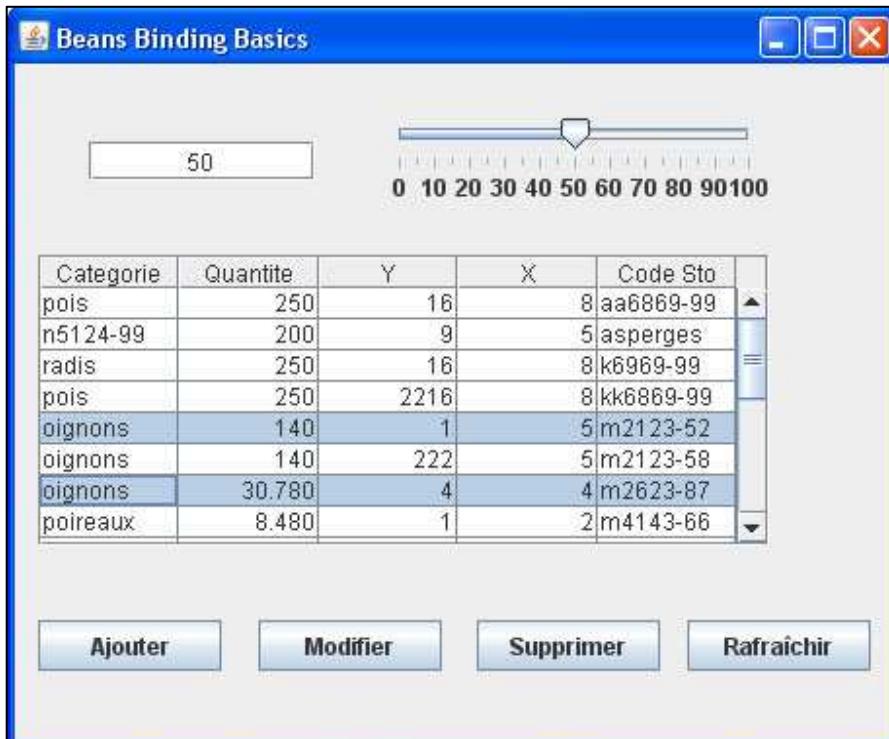
dans FenApp.java

```
...
private void ButtonSupprimerActionPerformed(java.awt.event.ActionEvent evt)
{
    int[] selected = jTable2.getSelectedRows();
    List<ProduitsEnStock> toRemove = new ArrayList<ProduitsEnStock>(selected.length);

    EntityTransaction et = BeansBindingBasiquePUEntityManager.getTransaction();
    if (et.isActive()) et.rollback();
    et.begin();

    for (int idx=0; idx<selected.length; idx++)
    {
        ProduitsEnStock c =
            produitsEnStockList.get(jTable2.convertRowIndexToModel(selected[idx]));
        toRemove.add(c);
        BeansBindingBasiquePUEntityManager.remove(c);
    }
    produitsEnStockList.removeAll(toRemove);
    et.commit();
}
...
...
```

A l'exécution :



Après appui sur le bouton Supprimer :

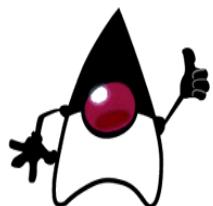
Beans Binding Basics

Categorie	Quantite	Y	X	Code Sto
pois	250	16	8	aa6869-99
n5124-99	200	9	5	asperges
radis	250	16	8	k6969-99
pois	250	2216	8	kk6869-99
oignons	140	222	5	m2123-58
poireaux	8.480	1	2	m4143-66
poireaux	21.852	96	31	m6969-88
pois	2.500	16	8	n6969-69

Ajouter Modifier Supprimer Rafraîchir

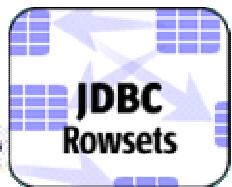
codeSto *	x	y	quantite	categorie
aa6869-99	8	16	250	pois
asperges	5	9	200	n5124-99
k6969-99	8	16	250	radis
kk6869-99	8	2216	250	pois
m2123-58	5	222	140	oignons
m4143-66	2	1	8480	poireaux
m6969-88	31	96	21852	poireaux
n6969-69	8	16	2500	pois
n6969-84	8	16	250	navets
nn6969-75	8	16	250	courgettes
p5421-69	7	9	520	choux fleurs
p6969-99	8	16	250	pois
u6869-99	8	16	250	pommes de terre
v6869-99	8	16	250	pois
wx6869-99	8	16	250	pois
xx6869-99	8	16	8900	haricots
y6869-99	8	16	15250	pois

Ouf ;-) !



Nous étions dans les Java Beans et les tables des bases de données :
restons-y !

XXXII. Les rowsets : un autre accès aux bases de données



Qui se sert de la lampe, au moins de l'huile y met.

(E. Jodelle, Au Roi)

1. Des JavaBeans en forme de ResultSets

On a coutume de présenter les rowsets comme "une couche d'abstraction au-dessus des resultsets". Waow ! Cela veut dire quoi ? En gros, que l'on va tenter de dissimuler les détails de la connexion aux bases de données utilisées, apparents avec les objets Connection et Statement, aux yeux du développeur peu intéressé par ces éléments techniques parce qu'ils sont répétitifs et toujours semblables. En clair, cela veut dire que la technique à base de resultsets du JDBC primitif sent un peu trop la plomberie, ce qui ne facilite pas leur intégration dans des architectures J2EE basées sur la technologie des JavaBeans.

Extension des ResultSet,

un objet **RowSet** est donc un objet qui est un **JavaBean** et qui contient des **données** sous forme de table qui ont été extraites d'une source de données

la source de données pouvant être bien sûr une base de données, mais aussi une feuille de calcul de tableur ou un fichier.

Un rowset est donc capable, grâce à ses propriétés, de se connecter à une base de données et d'y exécuter une requête pour acquérir les données qu'il souhaite contenir ou au contraire y écrire des données modifiées. Il peut ensuite choisir de se déconnecter de la base (on parle de "*rowset déconnecté*") ou, au contraire, de maintenir une connexion permanente avec cette base ("*rowset connecté*"). Etant un JavaBean, il connaît le mécanisme des événements et des listeners associés. De plus, il existe des objets Reader et Writer dont le rôle est de lire et écrire dans un rowset donné – celui pour lequel ils se seront enregistrées (comme les listeners).

Un RowSet déconnecté peut aussi être sérialisé, si bien que l'on peut envisager de l'envoyer sur le réseau à un client léger, avec retour en cas de mises à jour pour répercussion dans la base ... rien ne nous arrête !

Le rowset se présente donc effectivement comme une couche abstraite placée au-dessus des drivers JDBC : à la différence des ResultSet intimement liés à ceux-ci, les RowSet s'appuient certes sur ces drivers, mais peuvent être manipulés plus facilement.

2. L'interface RowSet

L'interface **RowSet**, défini dans le package javax.sql à partir du JDK 1.4, est un interface dérive de ResultSet. Ses méthodes getXXX() et setXXX() définissent ses propriétés, parmi lesquelles on peut déjà noter :

get/setCommand
get/setQueryTimeOut
get/setMaxFieldSize
get/setUserName
get/setMaxRows
get/setTransactionIsolation
etc

Deux méthodes présentent un intérêt immédiat :

1) public void **setCommand** (String cmd) throws SQLException

qui permet de définir la valeur de l'objet commande du rowset, c'est-à-dire l'opération à réaliser sur la source de données (par exemple, "select * from stock");

2) public void **execute()** throws SQLException

qui permet de charger le rowset avec des données en se basant sur les propriétés courantes du bean (command bien sûr, mais aussi read only, maximum field size, maximum rows, escape processing et query timeout).

Un rowset sera donc construit sur base de l'identification de la source de données et rempli au moyen de ces deux méthodes. Mais RowSet n'est qu'un interface ...

3. Des modèles d'implémentations

On s'attend donc à ce que divers providers proposent des implémentations de l'interface RowSet sur base de leurs drivers JDBC, implémentations qui seront effectivement utilisées dans les applications. Le package javax.sql ne fournit aucune implémentation, seulement, selon le principe même de l'interface, le cadre de référence de l'utilisation des RowSet. Cependant, l'API JDBC 2.0 package définit ce que devraient être les implémentations selon que le RowSet est destiné à être connecté ou déconnecté ou à usage selon HTTP. Ceci justifie la définition des trois interfaces, définis dans le package javax.sql.rowset :

♦ **CachedRowSet** : définit ce que devrait posséder un RowSet déconnecté, sérialisable et parcourable en tous sens [*disconnected, serializable and scrollable*]. Un tel RowSet conserve ses données (et d'ailleurs aussi ses métadonnées) en mémoire (ce qui le rend inapproprié pour les grands ensembles de données) et est donc utilisable sans nécessiter la présence d'un driver. Il n'a pas besoin de maintenir une connexion permanente avec la source données : il se contente de l'établir pour lire ses données ou propager les mises à jour qu'il a enregistrées. Dans le contexte des PDA (Personal Digital Assistant), le thin client utilise fréquemment ce principe. Dans le contexte 3-tiers, il utilise les technologies RMI/IIOP pour communiquer avec le tier intermédiaire qui accède effectivement aux données.

♦ **JdbcRowSet** : définit ce que devrait posséder un RowSet connecté (éventuellement parcourable en tous sens et éventuellement modifiable), qui conserve donc une connexion avec la source de données durant toute son utilisation. Ce genre de RowSet répond donc essentiellement au souhait de disposer d'un JavaBean qui encapsule l'utilisation d'un ResultSet et d'un driver JDBC; ce dernier donc bien être présent puisque, en définitive, tout passe par lui.

- ◆ **WebRowsSet** : définit l'allure d'un RowSet destiné aux applications distribuées, visant à délivrer des données à un client léger dans un contexte 3-tiers. Il utilise le protocole HTTP et les technologies basées sur XML pour communiquer avec le tier intermédiaire, qui est typiquement une servlet ou JSP.

4. Un package d'implémentation des rowsets : utilisation basique

Sun fournit dans un package com.sun.rowset des classes qui implémentent ces principaux types de rowsets. Ce package est proposé dans un fichier rowset.jar, qui contient également le package javax.sql.rowset évoqué ci-dessus.

A titre d'exemple, voyons comment procéder pour accéder aux données d'une table (ici, stock) appartenant à une base de données relationnelle (ici, impres-metal). Nous allons atteindre cet objectif selon un cheminement très similaire à celui d'un accès JDBC classique par ResultSet.

1) Il faudra tout d'abord charger le driver JDBC convenable. Nous utiliserons ici le pont jdbc-odbc pour une base MS-ACCESS, même si ce driver n'est pas idéal pour le contexte des rowsets, puisqu'il n'implémente pas toutes les fonctionnalités.

2) Notre rowset conteneur de données sera une implémentation de JdbcRowSet, qui est donc un rowset connecté et qui rappelle donc le plus nos manières de faire JDBC classique. La classe à utiliser, provenant du package com.sun.rowset, est :

```
public class JdbcRowSetImpl
extends javax.sql.rowset.BaseRowSet
implements javax.sql.rowset.JdbcRowSet
```

Cette classe admet plusieurs constructeurs, dont le plus utile est :

```
public JdbcRowSetImpl(java.lang.String url, java.lang.String user,
                     java.lang.String password) throws java.sql.SQLException
```

le premier paramètre désignant la source de données que nous sommes habitués à trouver dans une méthode getConnection(). Les deux autres constructeurs sont beaucoup plus orientés "JDBC classique" :

```
public JdbcRowSetImpl(java.sql.ResultSet res) throws java.sql.SQLException
public JdbcRowSetImpl(java.sql.Connection con) throws java.sql.SQLException
```

Le rowset ainsi créé possède toute une série de caractéristiques par défaut :

- ◆ il n'a pas de time-out pour ses commandes;
- ◆ il ne fixe pas de limite pour le nombre de tuples qu'il va contenir pas plus qu'il n'en a pour la taille des colonnes;
- ◆ il ne montre pas les données supprimées;
- ◆ il correspond à un curseur "scrollable" et "snapshot" (les modifications extérieures ne sont pas visibles);
- ◆ il ne voit pas les changements non validés (c'est-à-dire n'ayant pas fait l'objet d'un commit – on évite donc les "lectures impropre" au sens SGBD du terme).

3) Il nous faut à présent définir la propriété command de notre bean au moyen de :

public void **setCommand**(String cmd) throws SQLException

qui a été implémentée dans la classe de base BaseRowSet de toutes les classes implémentant RowSet.

4) Il reste à appeler la méthode

public void **execute()** throws java.sql.SQLException

qui, quant à elle, est bien implémentée dans la classe JdbcRowSetImpl, puisque son action est ici spécifique à un rowset connecté : ici, elle remplit le ResultSet encapsulé dans l'objet rowset !

5) Le parcours s'effectue comme on pratique avec un ResultSet (et pour cause).

Un programme élémentaire d'accès à une table stock dans une source de données ODBC fabricametal (qui cache en fait une base de données MS-Access nommé impres-metal) s'écrira comme suit :

RowsetBasics.java

```
/*
 * RowsetBasics.java
 * Created on 28 septembre 2004, 8:19
 */
/** 
 * @author vilvens
 */

import java.sql.*;
import javax.sql.*;
import com.sun.rowset.*;

public class RowsetBasics
{
    private static RowSet rs = null;
    public RowsetBasics() {}

    public static void main(String[] args)
    {
        try
        {
            Class leDriver = Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println("Driver JDBC-OBDC chargé");
        }
        catch (ClassNotFoundException e)
        { System.out.println("Driver adéquat non trouvable : " + e.getMessage()); }
    }
}
```

```
try
{
    rs = new JdbcRowSetImpl("jdbc:odbc:fabricametal","","","");
    System.out.println("Rowset instancié");

    rs.setCommand("select * from stock");
    rs.execute();
    System.out.println("Commande créée sur le rowset");

    int cpt = 0;
    while (rs.next())
    {
        if (cpt==0) System.out.println("Parcours du curseur");
        cpt++;
        String ns = rs.getString("num_serie");
        String cp = rs.getString("code_produit");
        float m1 = rs.getFloat("mesure_1");
        float m2 = rs.getFloat("mesure_2");
        System.out.println(cpt + ". " + ns + " - " + cp + " : " + m1 + "/" + m2);
    }
}
catch (SQLException e)
{
    System.out.println("Erreur SQL : " + e.getMessage());
}
catch (Exception e)
{
    System.out.println("Erreur générale : " + e.getMessage());
}
```

Résultat :

```
Driver JDBC-ODBC chargé
Rowset instancié
Commande créée sur le rowset
Parcours du curseur
1. 50002001 - 50001 : 49.0/199.0
2. 50002002 - 50001 : 49.6/201.0
3. 50002003 - 50001 : 50.2/200.5
4. 50002004 - 50001 : 49.2/201.5
5. 50002005 - 50001 : 49.9/199.5
6. 50002006 - 50001 : 50.2/200.3
7. 50002007 - 50001 : 50.01/200.3
8. 50002008 - 50001 : 544.5/200.1
JdbcRowSet (setTypeMap): null
JdbcRowSet (setMaxFieldSize): [Microsoft][Pilote ODBC Microsoft Access]Fonction
optionnelle non installée
JdbcRowSet (setQueryTimeout): [Microsoft][Pilote ODBC Microsoft Access]Fonction
optionnelle non installée
```

Mouais ... Cela fonctionne – mais il y a des plaintes quant aux propriétés du rowset qui ne peuvent être initialisées – quand on nous disait que jdbc-odbc était limité ...

Remarques

1) Nous eussions pu écrire :

```
con = DriverManager.getConnection("jdbc:odbc:fabricametal","","");
System.out.println("Connexion à la BDD inpres-metal réalisée");
rs = new JdbcRowSetImpl(con);
```

mais c'eût été moins pur ...

2) La source de données peut être une source sur une machine du réseau autre que celle où l'application est exécutée :

- ◆ pour jdbc:odbc : la source de données doit être une **source réseau**, c'est-à-dire que l'on a associé, lors de la création de cette source de données, un lecteur réseau au répertoire qui contient la base (répertoire préalablement partagé) et que la source de données a donc été définie par l'intermédiaire de ce lecteur réseau; donc, si le répertoire partagé "notes" de la machine "claude" contient la base inpres-metal, on associe un lecteur réseau, par exemple Y:, à ce répertoire et la source de données est alors associée à "Y:\|Claude\notes".
- ◆ pour les autres types de bases, comme MySQL, les choses sont plus directes puisque l'on peut spécifier l'**adresse réseau** directement dans l'url de la base de données, par exemple:

"jdbc:mysql://10.59.5.18/inpresMetal"

5. Les rowsets déconnectés

5.1 Le principe d'utilisation

Si l'exemple précédent montre certainement que l'on travaille avec les RowSet à un niveau d'abstraction plus élevé que celui des ResultSet, il n'en reste pas moins seulement moyennement convaincant : après tout, un développeur peut se créer ses propres bibliothèques JDBC sur un principe similaire, cela sans trop de problèmes.

Mais il y a lieu : il s'agit de l'utilisation de rowsets déconnectés, déjà évoqués. Pour rappel donc, un tel RowSet conserve ses données et ses métadonnées en mémoire et est donc utilisable sans nécessiter la présence d'un driver, du moins tant que le rowset est utilisé en standalone. Le schéma classique d'utilisation est donc le suivant :

1) deux acteurs interviennent : un serveur qui accède à la base de données visée et un client sous forme d'application distante qui souhaite manipuler les données contenues dans la base mais qui ne sait pas y accéder;

2) le client se connecte au serveur de manière classique (par exemple, par une connexion TCP/IP); éventuellement, il lui envoie la requête à exécuter, mais le plus souvent, le serveur connaît cette requête à l'avance (du genre "liste de tous les articles disponibles");

3) le serveur réalise cette requête et charge un rowset avec le résultat; ce rowset contient aussi l'url jdbc de la base de données, ainsi que le nom de user et le mot de passe associé; il sérialise ce rowset sur la connexion TCP/IP;

4) le client reçoit le rowset sérialisé : il peut à présent manipuler ses données en local; pour cela, il n'a pas besoin d'un driver;

5) si le client souhaite répercuter les modifications qu'il a apportées aux données dans la base de données, il devra alors utiliser les services du driver qui se chargera de ce travail en tâche de fond : il trouvera dans le rowset l'url jdbc qui permettra d'atteindre la base pour y enregistrer les mises à jour.

5.2 Le serveur de rowsets

Après avoir chargé le driver JDBC nécessaire à l'accès à la base données visée (dans l'exemple ci-dessous, il s'agit toujours d'une base inpres-metal accédé par un pont jdbc-odbc sur une source de données "fabricametal"), le serveur se comporte comme tout serveur TCP/IP : il instancie un objet ServerSocket et se met en attente sur un apple de la méthode accept().

Lorsqu'un client le contacte :

1) il instancie une implémentation de **CachedRowSet**, qui est donc un rowset déconnecté - la classe à utiliser, provenant à nouveau du package com.sun.rowset, est :

```
public class CachedRowSetImpl  
extends javax.sql.rowset.BaseRowSet  
implements javax.sql.RowSet, javax.sql.RowSetInternal, java.io.Serializable,  
java.lang.Cloneable, javax.sql.rowset.CachedRowSet
```

Cette classe admet un seul constructeur vraiment pratique, le constructeur par défaut :

```
public CachedRowSetImpl() throws java.sql.SQLException
```

qui crée un rowset déconnecté avec une configuration par défaut :

onInsertRow = false insertRow = null cursorPos = 0 numRows = 0 showDeleted = false queryTimeout = 0 maxRows = 0 maxFieldSize = 0 escapeProcessing = true	rowsetType = ResultSet.TYPE_SCROLL_INSENSITIVE concurrency = ResultSet.CONCUR_UPDATABLE readOnly = false isolation = Connection.TRANSACTION_READ_COMMITTED onInsertRow = false insertRow = null cursorPos = 0 absolutePos = 0 numRows = 0
--	---

Ce rowset va être voué à une base de données bien précise au moyen des méthodes :

```
public void setUrl(String url) throws SQLException  
public void setUsername(String name) throws SQLException  
public void setPassword(String password) throws SQLException
```

2) la requête SQL est préparée et exécutée par les habituelles méthodes prototypées dans RowSet :

```
public void setCommand (String cmd) throws SQLException  
public void execute() throws SQLException
```

3) le rowset résultat est finalement sérialisé sur le réseau.

En définitive, notre serveur se résume donc à :

RowsetDeconnecteServeur.java

```
/*  
 * RowsetDeconnecteServeur.java  
 * Created on 29 septembre 2004, 17:46  
 */  
  
/**  
 * @author Vilvens  
 */  
  
import java.sql.*;  
import java.net.*;  
import java.io.*;  
  
import javax.sql.*;  
import com.sun.rowset.*;  
  
public class RowsetDeconnecteServeur  
{  
    private static RowSet rs = null;  
    private static ServerSocket ss = null;  
    private static Socket s = null;  
  
    public RowsetDeconnecteServeur() {}  
  
    public static void main(String[] args)  
    {  
        try  
        {  
            Class leDriver = Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
            System.out.println("Driver JDBC-OBDC chargé");  
        }  
        catch (ClassNotFoundException e)  
        {  
            System.out.println("Driver adéquat non trouvable : " + e.getMessage());  
        }  
  
        ObjectOutputStream oos = null;
```

```

try
{
    ss = new ServerSocket(14000);
    System.out.println("Serveur en attente");
    s = ss.accept();
    oos = new ObjectOutputStream(s.getOutputStream());
}
catch (IOException e)
{
    System.out.println("Erreur réseau : " + e.getMessage());
    System.exit(0);
}

try
{
    rs = new CachedRowSetImpl();
    System.out.println("Rowset instancié");
    rs.setUrl ("jdbc:odbc:fabricametal");
    rs.setUsername("");
    rs.setPassword("");
    rs.setCommand("select * from stock");
    System.out.println("Commande créée sur le rowset");
    rs.execute();

    System.out.println("Envoi du rowset au client");
    oos.writeObject(rs);
    oos.close();s.close(); ss.close();
}
catch (SQLException e)
{
    System.out.println("Erreur SQL : " + e.getMessage());
}
catch (Exception e)
{
    System.out.println("Erreur générale : " + e.getMessage());
}
}
}

```

5.3 Le client des rowsets

A l'image du client FenCurseur du chapitre consacré à JDBC¹, nous allons imaginer que notre "consommateur de rowset" présente un interface du type suivant :

¹ voir chapitre XII de "Langage Java (II) : Programmation avancée des applications classiques"



Concrètement, notre client, après avoir créé son interface graphique,

1) acquiert, en réponse à un appui sur le bouton "Démarrer", le rowset associé à la requête prévue par un `readObject()` sur le flux réseau préalablement créé au moyen d'une classique connexion TCP/IP;

2) réalise toutes les opérations prévues de déplacement au sein du curseur et de mises à jour diverses, au moyen notamment des méthodes de `ResultSet` :

```
public void updateString(String columnName, String x) throws SQLException
public void updateFloat(String columnName, float x) throws SQLException
```

```
...
public void updateRow() throws SQLException
```

```
public void moveToInsertRow() throws SQLException
public void insertRow() throws SQLException
```

```
public void deleteRow() throws SQLException
```

3) répercute ces diverses modifications dans la base de données originelle au moyen de la méthode :

```
public void acceptChanges()
throws java.sql.SQLException, javax.sql.rowset.spi.SyncProviderException
```

Le client a donc l'aspect suivant :

RowsetDeconnecteClient.java

```
/*
 * RowsetDeconnecteClient.java
 * Created on 29 septembre 2004, 22:41
 */

/**
 * @author Vilvens
 */
import java.net.*;
import java.io.*;
import java.sql.*;

import com.sun.rowset.*;

public class RowsetDeconnecteClient extends java.awt.Frame
{
    private CachedRowSetImpl crs = null;
    private int nbreTuples = 0;

    public RowsetDeconnecteClient()
    {
        initComponents();
        setTitle("Contrôle par rowset des fabrications métalliques");
    }

    private void initComponents()
    {
        panel1 = new java.awt.Panel();
        // instructions pour le GUI - passons
        ...
        pack();
    }

    private void B_CommitActionPerformed(java.awt.event.ActionEvent evt)
    {
        try
        {
            crs.acceptChanges();
        }
        catch (SQLException e) { System.out.println("Erreur SQL : " + e.getMessage()); }
    }

    private void B_ArreterActionPerformed(java.awt.event.ActionEvent evt)
    { System.exit(0); }
}
```

```
private void B_FinActionPerformed(java.awt.event.ActionEvent evt)
{
    try
    {
        crs.last(); getTuple();
    }
    catch (SQLException e) { System.out.println("Erreur SQL : " + e.getMessage()); }
}

private void B_DebutActionPerformed(java.awt.event.ActionEvent evt)
{
    try
    {
        crs.first(); getTuple();
    }
    catch (SQLException e) { System.out.println("Erreur SQL : " + e.getMessage()); }
}

private void B_ReculerActionPerformed (java.awt.event.ActionEvent evt)
{
    try
    {
        crs.previous();
        if (crs.isLast())
        {
            System.out.println("Fin de curseur"); crs.previous();
        }
        getTuple();
    }
    catch (SQLException e) { System.out.println("Erreur SQL : " + e.getMessage()); }
}

private void B_AvancerActionPerformed (java.awt.event.ActionEvent evt)
{
    try
    {
        crs.next();
        if (crs.isFirst())
        {
            System.out.println("Début de curseur"); crs.next();
        }
        getTuple();
    }
    catch (SQLException e) { System.out.println("Erreur SQL : " + e.getMessage()); }
}
```

```

private void B_SupprimerActionPerformed (java.awt.event.ActionEvent evt)
{
    try
    {
        crs.deleteRow();
        //crs.acceptChanges(); on pourrait ...
    }
    catch (SQLException e) { System.out.println("Erreur SQL : " + e.getMessage()); }
}

private void B_AjouterActionPerformed (java.awt.event.ActionEvent evt) {
    try
    {
        crs.moveToInsertRow();
        crs.updateString("num_serie", ZE_num_serie.getText());
        crs.updateString("code_produit", ZE_code_produit.getText());
        crs.updateFloat("mesure_1", Float.parseFloat(ZE_mesure_1.getText()));
        crs.updateFloat("mesure_2", Float.parseFloat(ZE_mesure_2.getText()));
        crs.insertRow();
        crs.moveToCurrentRow();
        //crs.acceptChanges(); on pourrait ...
        String affiche = ++nbreTuples + ". " + ZE_num_serie.getText() + " - " +
            ZE_code_produit.getText() + ":" +
            Float.parseFloat(ZE_mesure_1.getText()) + "/" +
            Float.parseFloat(ZE_mesure_2.getText());
        BL_Tuples.add(affiche);
        System.out.println(affiche);
    }
    catch (SQLException e) { System.out.println("Erreur SQL : " + e.getMessage()); }
}

private void B_ModifierActionPerformed (java.awt.event.ActionEvent evt)
{
    try
    {
        crs.updateString("num_serie", ZE_num_serie.getText());
        crs.updateString("code_produit", ZE_code_produit.getText());
        crs.updateFloat("mesure_1", Float.parseFloat(ZE_mesure_1.getText()));
        crs.updateFloat("mesure_2", Float.parseFloat(ZE_mesure_2.getText()));

        crs.updateRow();
        //crs.acceptChanges();on pourrait ...
    }
    catch (SQLException e) { System.out.println("Erreur SQL : " + e.getMessage()); }
}

```

```

private void B_DémarrerActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {
        Socket s = new Socket ("192.168.2.2", 14000);
        ObjectInputStream ois = new ObjectInputStream (s.getInputStream());
        crs = (CachedRowSetImpl) ois.readObject();

        int cpt = 0;
        while (crs.next())
        {
            if (cpt==0) System.out.println("Parcours du curseur");
            cpt++;
            String ns = crs.getString("num_serie");String cp = crs.getString("code_produit");
            float m1 = crs.getFloat("mesure_1"); float m2 = crs.getFloat("mesure_2");
            String affiche = cpt + ". " + ns + " - " + cp + ":" + m1 + "/" + m2;
            BL_Tuples.add(affiche);
            System.out.println(affiche);
            getTuple();
        }

        nbreTuples = cpt;
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        // nécessaire pour les futures mises à jour
    }
    catch (Exception e)
    {
        System.out.println("Erreur SQL : " + e.getMessage());
    }
}

private void exitForm(java.awt.event.WindowEvent evt) { System.exit(0); }

public static void main(String args[])
{
    new RowsetDeconnecteClient().show();
}

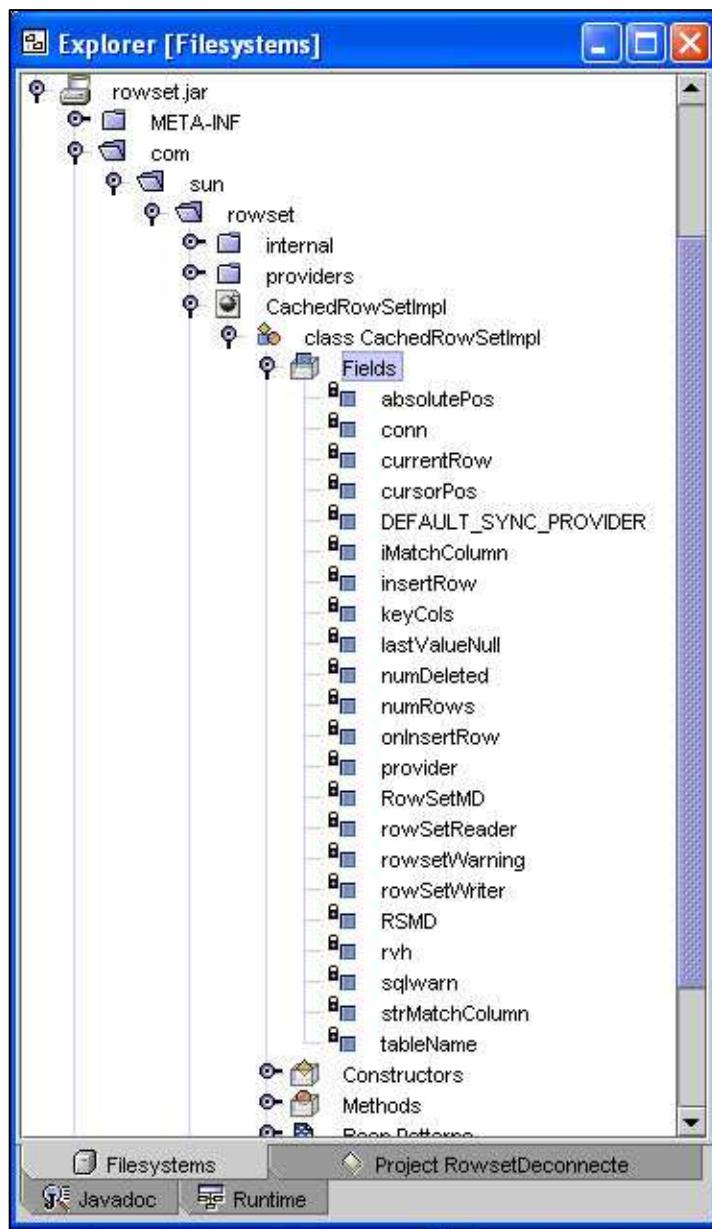
private java.awt.Button B_Supprimer;  private java.awt.Button B_Avancer;
private java.awt.Button B_Debut;  private java.awt.TextField ZE_mesure_1;
private java.awt.Button B_Modifier;  private java.awt.Panel panel2;
private java.awt.TextField ZE_mesure_2;  private java.awt.Button B_Fin;
private java.awt.Panel panel1;  private java.awt.Button B_Démarrer;
private java.awt.Button B_Commit;  private java.awt.Panel panel3;
private java.awt.Button B_Arreter;  private java.awt.Button B_Ajouter;
private java.awt.Button B_Reculer;  private java.awt.TextField ZE_num_serie;
private java.awt.Choice BL_Tuples;  private java.awt.TextField ZE_code_produit;
private java.awt.Panel panel4;

```

```
private void getTuple () throws SQLException
{
    String ns = crs.getString("num_serie"); ZE_num_serie.setText(ns);
    String cp = crs.getString("code_produit"); ZE_code_produit.setText(cp);
    float m1 = crs.getFloat("mesure_1"); ZE_mesure_1.setText(String.valueOf(m1));
    float m2 = crs.getFloat("mesure_2"); ZE_mesure_2.setText(String.valueOf(m2));
}
```

5.4 Les acteurs sous-jacents

La méthode acceptChanges() de répercussion dans la base de données des modifications apportées par le client semble "magique", mais fait en fait intervenir plusieurs objets voués chacun à une phase de ce travail. On peut vérifier leur présence en introspectant le jar d'implémentation des rowsets :



On distingue ainsi :

- ♦ un objet

private RowSetReader rowSetReader

implémentant l'interface **RowSetReader** du package javax.sql dont le rôle est bien entendu de lire les tuples qui vont constituer la version courante du RowSet – c'est le rôle de sa méthode

public boolean **readData**(RowSetInternal caller) throws SQLException

Le paramètre implémente l'interface **RowSetInternal** du même package qui représente l'interface du RowSet pour qui compte l'utiliser, avec des méthodes bien claires :

```
public Object[] getParams() throws SQLException  
public Connection getConnection() throws SQLException  
public void setMetaData(RowSetMetaData md) throws SQLException  
public ResultSet getOriginal() throws SQLException  
public ResultSet getOriginalRow() throws SQLException
```

- ♦ un objet

private RowSetWriter rowSetWriter

implémentant l'interface **RowSetWriter** du package javax.sql dont le rôle est bien entendu d'écrire les données modifiées – c'est le rôle de sa méthode

public boolean **writeData**(RowSetInternal caller) throws SQLException

pour ce faire, il utilise un objet Connection JDBC pour retrouver la base de données (objet conn – lui-même utilise une socket si nécessaire pour une base distante) et aussi ...

♦ un objet implémentant la classe abstraite **SyncProvider** du package javax.sql.rowset.spi : son rôle est clairement de prendre en charge la synchronisation de possibles mises à jour simultanées et donc concurrentielles : cet objet gère donc les transactions au sens SGBD du terme, ainsi que l'existence de méthodes comme

```
public abstract int getDataSourceLock() throws javax.sql.rowset.spi.SyncProviderException  
et  
public abstract void setDataSourceLock(int ) throws javax.sql.rowset.spi.SyncProviderException
```

ou encore de variables membres statiques comme

```
public static int DATASOURCE_ROW_LOCK  
ou  
public static int DATASOURCE_TABLE_LOCK
```

le laisse deviner. Dans le cas du package d'implémentation de Sun, ce fournisseur de mécanisme transactionnel est un objet instance de

```
public final class RIOptimisticProvider
extends javax.sql.rowset.spi.SyncProvider
implements java.io.Serializable
```

qui permet d'initialiser la variable membre du rowset :

```
private javax.sql.rowset.spi.SyncProvider provider
```

En pratique, fort heureusement, les implémentations proposées nous dispensent, dans la plupart des cas, de devoir créer ces objets qui sont les acteurs cachés et incontournables du mécanisme des rowset déconnectés.

6. Les événements liés aux rowsets

Les rowsets sont des JavaBeans et donc, à ce titre, ils sont susceptibles de générer des événements (des instances de **RowSetEvent**) correspondant à des changements de certaines de leurs propriétés, événements notifiés à tous les beans qui se sont enregistrés comme intéressés par ces événements (objets qui sont donc des **RowSetListener**). Les trois événements qui donnent lieu à une notification sont :

- ◆ la modification d'un tuple (ajout, suppression ou mise à jour)
- ◆ la modification de tout le rowset (remplissage initial, propagation des mises à jour)
- ◆ un mouvement du curseur.

Tout composant qui souhaite être averti de tels événements doit, comme on s'en doute dans le contexte des JavaBeans,

- ◆ implémenter l'interface

```
public interface RowSetListener extends EventListener xx
```

qui comporte les méthodes :

méthode	sémantique associée
public void rowChanged (RowSetEvent event)	modification d'un tuple
public void rowSetChanged (RowSetEvent event)	modification de la globalité du rowset
public void cursorMoved (RowSetEvent event)	mouvement du curseur

- ◆ s'enregistrer comme listener au moyen de la méthode

```
public void addRowSetListener(RowSetListener listener)
```

L'objet événement est une instance de

```
public class RowSetEvent extends EventObject
```

dont la méthode habituelle

```
public Object getSource()
```

permet de récupérer le RowSet qui a créé l'événement. Parfait donc, semble-t-il ...

Mouais ... si ce n'est que les événements sont notifiés dès que l'une des commandes updateRow(), insertRow() ou deleteRow() est exécutée, si bien que dans deux cas sur trois, le curseur est dans un état invalide (pour l'ajout) ou non rafraîchi (pour la suppression). Il est donc assez compliqué pour un RowSetListener de travailler valablement sur le rowset surveillé.

Néanmoins, nous allons, pour illustrer cette communication événementielle caractéristique des beans en utilisant une classe FichierJournal qui sera un auditeur attentif aux changements apportés au rowset. Cette classe s'écrit de la manière donnée ci-dessous (en ayant ajouté des instructions d'illustration et de traçage divers qui devraient disparaître dans une version finale). On pourra cependant remarquer que l'objet fichier journal est construit avec un nom de fichier et l'objet rowset qu'il est chargé de surveiller (on aurait pu utiliser la source de l'objet RowSetEvent passé aux méthodes du listener). Il écrit une ligne à chaque événement.

FichierJournal.java

```
/*
 * FichierJournal.java
 * Created on 5 octobre 2004, 9:44
 */

/**
 * @author Vilvens
 */

import java.io.*;
import java.util.*;

import java.sql.*;
import javax.sql.*;
import com.sun.rowset.*;

class FichierJournal implements RowSetListener
{
    private CachedRowSetImpl rowSetSurveille;
    private String nomFichier;

    public FichierJournal (CachedRowSetImpl crs, String n)
    {
        if (crs == null) System.out.println("Rowset vide");
        else
            { rowSetSurveille = crs; nomFichier = n; }
    }
}
```

```

public synchronized void écritLigne (String ligne)
{
    try
    {
        FileWriter f = new FileWriter(nomFichier, true);
        BufferedWriter bf = new BufferedWriter(f);
        bf.write("[" +new java.util.Date() + "] : " + ligne);
        bf.newLine();
        bf.close();
    }
    catch (IOException e)
    { System.out.println(e.getMessage()); }
}

public synchronized void écritLigne (String entete, String info)
{
    écritLigne(entete + "> " + info);
}

public void rowSetChanged (RowSetEvent e)
{
    Object obj = e.getSource();
    System.out.println("rowSetChanged : " + obj);
    Class cla = obj.getClass();
    System.out.println("rowSetChanged : / classe de la source : " + cla.getName());
    try
    {
        // CachedRowSetImpl rowSetSurveille =
        //     (CachedRowSetImpl)((CachedRowSetImpl)obj).createShared();

        if (rowSetSurveille == null)
            System.out.println("Curseur vide !!!");
        System.out.println("Position du tuple : " + rowSetSurveille.getRow());
        System.out.println ("commande : " + rowSetSurveille.getCommand());
        System.out.println ("url : " + rowSetSurveille.getUrl());
        rowSetSurveille.beforeFirst();
        while (rowSetSurveille.next())
        {
            String ns = rowSetSurveille.getString("num_serie");
            String cp = rowSetSurveille.getString("code_produit");
            float m1 = rowSetSurveille.getFloat("mesure_1");
            float m2 = rowSetSurveille.getFloat("mesure_2");
            String affiche = "rowSetChanged : " + ns + " - " + cp + " : " + m1 + "/" + m2;
            System.out.println(affiche);
            écritLigne("propagation vers la base", affiche);
        }
    }
    catch (SQLException ex)
    {System.out.println("Exception SQL : "+ ex.getMessage() + " -> " + ex.getSQLState());}
}

```

```

public void rowChanged (RowSetEvent e)
{
    Object obj = e.getSource();
    System.out.println("rowChanged : " + obj);
    Class cla = obj.getClass();
    System.out.println("rowChanged : / classe de la source : " + cla.getName());

    String affiche = "???";
    try
    {
        // CachedRowSetImpl crs =
        //     (CachedRowSetImpl)((CachedRowSetImpl)obj).createShared();

        if (rowSetSurveille == null)
            System.out.println("Curseur vide !!!");
        System.out.println("Position du tuple : " + rowSetSurveille.getRow());
        System.out.println ("commande : " + rowSetSurveille.getCommand());
        System.out.println ("url : " + rowSetSurveille.getUrl());

        affiche = "rowChanged - curseur non utilisable";
        rowSetSurveille.previous();
        if (rowSetSurveille.next())
        {
            String ns = rowSetSurveille.getString("num_serie");
            String cp = rowSetSurveille.getString("code_produit");
            float m1 = rowSetSurveille.getFloat("mesure_1");
            float m2 = rowSetSurveille.getFloat("mesure_2");
            affiche = "rowChanged : " + ns + " - " + cp + " : " + m1 + "/" + m2;
            System.out.println(affiche);
        }
    }
    catch (SQLException ex)
    {System.out.println("Exception SQL : " + ex.getMessage() + " -> " + ex.getSQLState());}
    écritLigne("modification locale", affiche);
}

public void cursorMoved (RowSetEvent e) {}
}

```

Nous allons reprendre notre serveur et notre client du paragraphe précédent. Le client va instancier un fichier journal, lequel va s'enregistrer dans la liste des listeners d'événements concernant le rowset déconnecté manipulé par le client :

RowsetDeconnecteClient.java (avec fichier journal)

```

...
public class RowsetDeconnecteClient extends java.awt.Frame
{
    private CachedRowSetImpl crs = null;
    private FichierJournal fichJour = null;

```

```
...
private void B_DémarrerActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {
        Socket s = new Socket ("192.168.2.2", 14000);
        ObjectInputStream ois = new ObjectInputStream (s.getInputStream());
        crs = (CachedRowSetImpl) ois.readObject();
        ...
    }
    catch (Exception e)
    {
        System.out.println("Erreur SQL : " + e.getMessage());
    }
    fichJour = new FichierJournal (crs, "c:\\java-sun-application\\metallica.jnl");
    crs.addRowSetListener(fichJour);
}
}
```

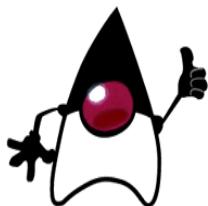
En exécutant les serveur et client, on peut voir sur la console, pour successivement une modification, un ajout et une suppression, le tout étant ensuite répercute dans la base originale :

```
Parcours du curseur
1. 50002001 - 50001 : 49.0/199.0
...
12. 5000201011111 - 50001 : 3544.5/3600.23
Fin de curseur
rowChanged : com.sun.rowset.CachedRowSetImpl@23e5d1
rowChanged : / classe de la source : com.sun.rowset.CachedRowSetImpl
Position du tuple : 7
commande : select * from stock
url : jdbc:odbc:fabricametal
rowChanged : 500020114141 - 50001 : 887766.7/8877202.0
rowChanged : com.sun.rowset.CachedRowSetImpl@23e5d1
rowChanged : / classe de la source : com.sun.rowset.CachedRowSetImpl
Position du tuple : 0
commande : select * from stock
url : jdbc:odbc:fabricametal
Exceptio SQL : Invalid Cursor position -> null
13. 5000200373839 - 50001 : 73.37/36.37
rowChanged : com.sun.rowset.CachedRowSetImpl@23e5d1
rowChanged : / classe de la source : com.sun.rowset.CachedRowSetImpl
Position du tuple : 0
commande : select * from stock
url : jdbc:odbc:fabricametal
rowChanged : 50002009 - 50001 : 1544.5/1600.23
rowSetChanged : com.sun.rowset.CachedRowSetImpl@23e5d1
rowSetChanged : / classe de la source : com.sun.rowset.CachedRowSetImpl
Position du tuple : 0
commande : select * from stock
```

```
url : jdbc:odbc:fabricametal
rowSetChanged : 50002001 - 50001 : 49.0/199.0
rowSetChanged : 50002004 - 50001 : 7766.66/77201.5
rowSetChanged : 500020055555 - 50001 : 5656.56/5656.56
rowSetChanged : 50002009 - 50001 : 1544.5/1600.23
rowSetChanged : 50002010 - 50001 : 3544.5/3600.23
rowSetChanged : 500020114141 - 50001 : 887766.7/8877202.0
rowSetChanged : 500020017 - 50001 : 7777.7/777.77
rowSetChanged : 50002003 - 50001 : 73.63/36.37
rowSetChanged : 500020037 - 50001 : 73.37/36.37
rowSetChanged : 5000200373839 - 50001 : 73.37/36.37
rowSetChanged : 500020101010 - 50001 : 3544.5/3600.23
rowSetChanged : 5000201011111 - 50001 : 3544.5/3600.23
```

tandis que le fichier journal contient :

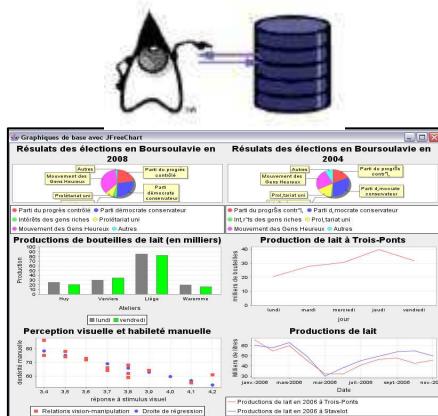
```
[Sun Oct 10 11:01:37 CEST 2004] : modification locale> rowChanged : 500020114141 - 50001 :
887766.7/8877202.0
[Sun Oct 10 11:01:53 CEST 2004] : modification locale> rowChanged - curseur non utilisable
[Sun Oct 10 11:02:20 CEST 2004] : modification locale> rowChanged : 50002009 - 50001 : 1544.5/1600.23
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 50002001 - 50001 : 49.0/199.0
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 50002004 - 50001 :
7766.66/77201.5
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 500020055555 - 50001 :
5656.56/5656.56
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 50002009 - 50001 :
1544.5/1600.23
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 50002010 - 50001 :
3544.5/3600.23
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 500020114141 - 50001 :
887766.7/8877202.0
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 500020017 - 50001 :
7777.7/777.77
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 50002003 - 50001 : 73.63/36.37
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 500020037 - 50001 :
73.37/36.37
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 5000200373839 - 50001 :
73.37/36.37
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 500020101010 - 50001 :
3544.5/3600.23
[Sun Oct 10 11:02:33 CEST 2004] : propagation vers la base> rowSetChanged : 5000201011111 - 50001 :
3544.5/3600.23
```



Il y aurait encore beaucoup à dire sur JDBC, qui est passé de la version 2.1 à la prochaine 4.0 : la manipulation des types SQL3 (BLOB, CLOB, ARRAY, REF, types structurés), les batch de mise à jour, les pools de connexions aux bases, les sources de données et la "gestion générique de données", etc.

Mais sans doute est-il temps de nous tourner vers les chiffres ☺ ...

XXXIII. La librairie JFreeChart



Je m'aime trop moi-même pour pouvoir haïr qui que ce soit.

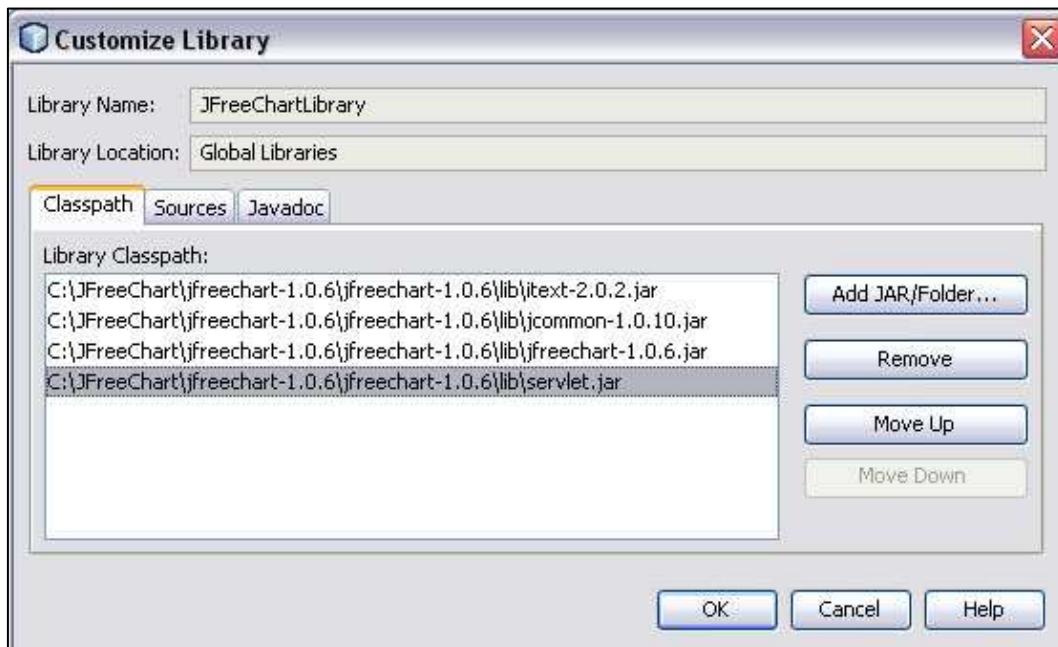
(J.-J. Rousseau, Les rêveries du promeneur solitaire)

[Un grand merci à J-M Wagner qui a bien débroussaillé le terrain à l'occasion d'une micro-formation de la Java Team ☺ !]

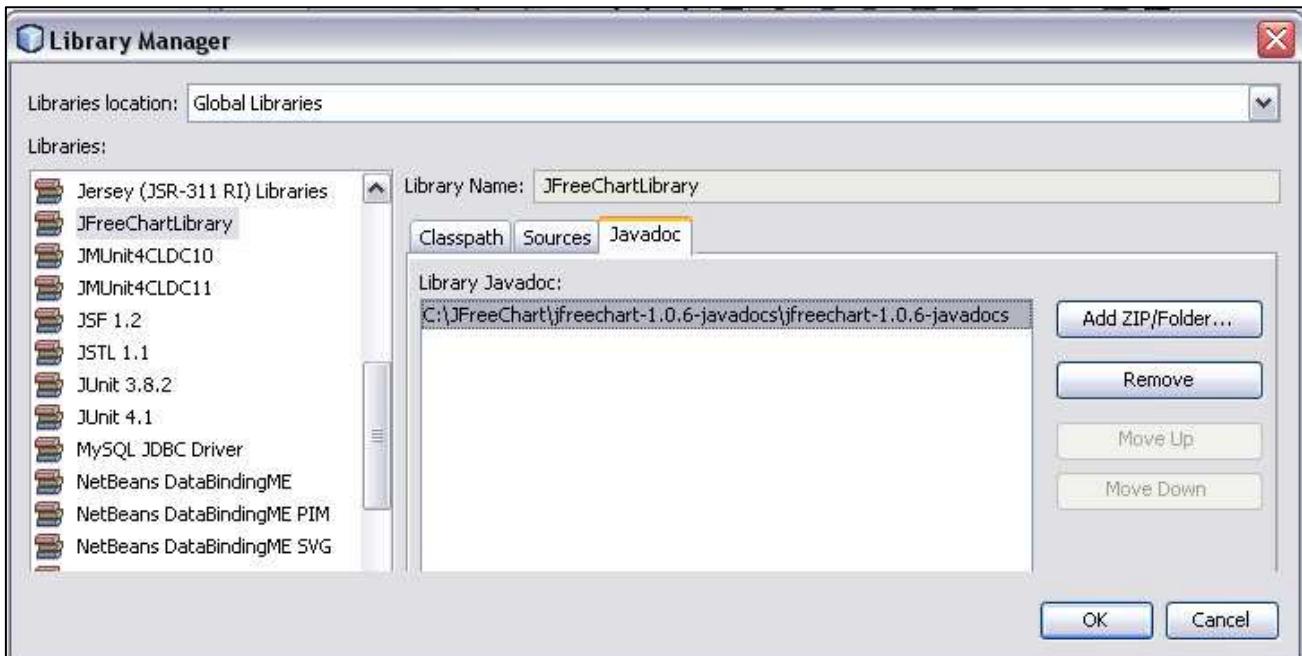
1. Une bibliothèque pour graphiques statistiques

JFreeChart est une librairie de classes Java offrant un large éventail de fonctionnalités de graphiques statistiques (dans un sens très large) utilisables au sein d'une application, d'une applet ou d'une servlet. Développée en 2000, elle est distribuée gratuitement (par sourceforge.net), avec son code source, sous licence GNU-LGPL, c'est-à-dire qu'elle peut être utilisée dans une application quelconque, libre ou propriétaire. On peut trouver les informations sur le software à <http://www.jfree.org/jfreechart/>, où l'on peut également le télécharger sous forme d'un fichier zip (jfreechart-1.0.6.zip) ainsi que la documentation correspondante.

On s'en doute, un certain nombre de jars matérialisent la librairie. Plutôt que de les incorporer un par un dans NetBeans 6.*, nous créons une librairie JFreeChartLibrary qui contient les jars utiles pour un usage courant. Classiquement, un clic droit sur le nœud Libraries du projet, "Add library ..." puis "Create ..." puis "Create new library" conduit à :



Un dernier "Add library" permet d'effectuer l'ajout de cette librairie au projet. On peut vérifier l'existence de notre nouvelle librairie par Tools→Libraries dans le menu principal et aussi en profiter pour y adjoindre les javadocs correspondants :



Nous voilà donc fin prêts à travailler ...

2. La représentation des données

L'idée de base est que les données destinées à produire un graphique statistique sont placées dans un container (un "**dataset**") implémentant l'interface **Dataset** : celui-ci se borne à assurer qu'un mécanisme d'événement (au sens des Java Beans) existe pour notifier tout modification de données au sein du dataset. On trouve donc des méthodes comme :

```
void addChangeListener (DatasetChangeListener listener)
```

où, dans la logique habituelle des Java Beans, le listener est un objet implémentant un interface **DatasetChangeListener** dont la seule méthode est :

```
void datasetChanged (DatasetChangeEvent event)
```

tandis que l'objet événement est capable de fournir le dataset concerné grâce à la méthode :

```
public Dataset getDataset()
```

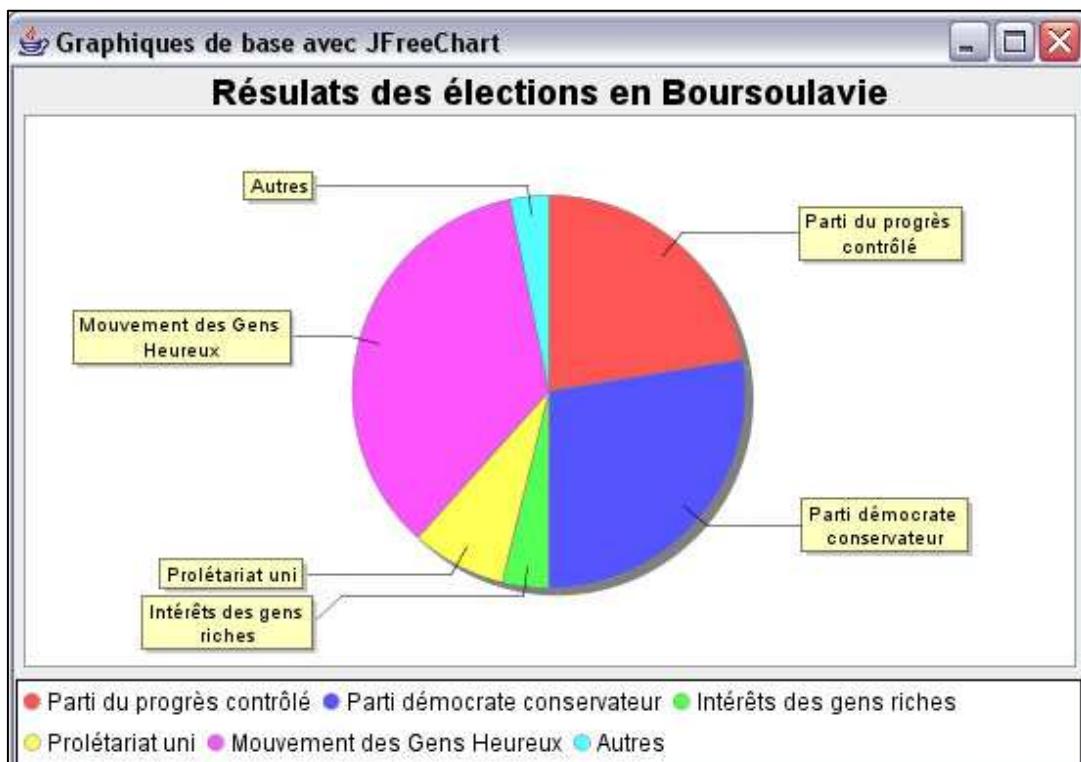
La manière de définir les données contenues dans le dataset dépend bien sûr du type de graphique statistique auquel il est destiné à servir : sectoriel, histogramme, nuage de points, etc. La classe abstraite **AbstractDataset**, qui implémente **l'interface**, ne pipe donc mot de méthodes d'insertion ou de retrait, mais focalise toujours sur la logique événementielle, avec des méthodes **protected** comme :

```
protected void fireDatasetChanged()  
protected void notifyListeners(DatasetChangeEvent event)
```

Ne se croirait-on pas revenus dans Java I (à la fin ...) ;-)?

3. L'exemple classique : le diagramme sectoriel

Afin d'illustrer la suite de notre propos, nous allons réaliser un diagramme sectoriel du type suivant :



3.1 Un Dataset particulier

Ce type de graphique est encore familièrement appelé "diagramme en camembert" ou "tarte" – *pie* en anglais. A priori, une manière de fournir les données consisterait à fournir les libellés des données (ici, le nom des partis politiques) et le pourcentage associé : c'est le rôle de la classe **DefaultPieDataset** qui, outre le fait qu'elle implémente la classe abstraite **AbstractDataset**, implémente aussi un interface **PieDataset** qui décrit les méthodes d'ajout de données dans un dataset pour "camembert" :

```
public void setValue (Comparable key, double value)
public void insertValue (int position, Comparable key, double value)
```

Comme on l'aura compris, ce genre de dataset est un container associatif de type hast-table, puisque la notion de clé est explicitement visible. Dans le cas de la deuxième méthode, le fait d'utiliser une clé existante a pour effet de réaliser une mise à jour de la valeur associée avant déplacement à la position souhaitée (comptée à partir de 0). Dans les deux cas, un événement **DatasetChangeEvent** sera envoyé aux listeners intéressés.

Bien sûr, la méthode

```
public java.lang.Number getValue (java.lang.Comparable key)
```

est aussi prévue. Dans le cas de notre exemple, les choses sont donc aussi simples que ceci :

```
DefaultPieDataSet ds = new DefaultPieDataSet(); // contient les data  
ds.setValue("Parti du progrès contrôlé", 22.36);  
ds.setValue("Parti démocrate conservateur", 27.69);  
ds.setValue("Intérêts des gens riches", 3.78);  
ds.setValue("Prolétariat uni", 7.85);  
ds.setValue("Mouvement des Gens Heureux", 35.12);  
ds.setValue("Autres", 3.2);
```

Donc, "Prolétariat uni" (par exemple) est considéré comme une clé qui permet d'accéder à l'information "7.85". Les données étant définies, il nous faut un objet réalisant le dessin souhaité ...

3.2 Le JFreeChart correspondant

La classe **JFreeChart** représente en fait un graphique statistique au sens général (histogramme, linéaire, sectoriel, nuage de points, ...). Le graphique réalisé sera dessiné en utilisant l'API Java2D. Un tel objet réalise ce travail en utilisant trois variables membres :

- ◆ une instance de **Dataset**, qui contient évidemment les données;
- ◆ une instance d'une classe dérivée de la classe abstraite **Plot**, qui gère notamment la représentation des données et leur référentiel (comme des axes);
- ◆ une instance de **Title**, qui comporte en particulier la légende du graphique).

Il existe bien sûr des constructeurs prévisibles comme

```
public JFreeChart (Plot plot)  
public JFreeChart (java.lang.String title, Plot plot)
```

- mais nous passerons plutôt par une factory ...

Signalons encore que la classe implémente les listeners **TitleChangeListener** et **PlotChangeListener** et qu'elle peut utiliser les services d'un **ChartChangeListener** :

```
public void addChangeListener (ChartChangeListener listener)  
public void fireChartChanged ()
```

à la sémantique assez évidente.

3.3 La classe factory

La classe **ChartFactory** regroupe en fait une série de méthodes permettant de créer un JFreeChart de type "camembert", "histogramme", "nuage de points", etc avec une série d'attributs par défaut (mais reconfigurables) : le rôle de ces "factories" n'est donc pas tant une question de portabilité que de facilité ... Citons parmi elles celle qui nous intéressera pour notre exemple :

```
public static JFreeChart createPieChart (java.lang.String title, PieDataset dataset,  
boolean legend, boolean tooltips, boolean urls)
```

les trois derniers paramètres indiquant si l'on souhaite une légende, des "tooltips" et la possibilité d'hyperliens (mais il faudrait alors les définir). Donc, pour notre exemple :

```
JFreeChart jf = ChartFactory.createPieChart("Résultats des élections en Boursoulavie", ds,  
true, true, false);
```

3.4 Les Plots

En fait, un objet JFreeChart possède une variable membre qui est une implémentation de l'interface **Plot** : son rôle est de réaliser les dessins relativement aux axes et aux données. Le JFreeChart utilisé ci-dessus, parce qu'il a été créé avec la factory `createPieChart()`, s'est vu muni automatiquement d'un objet instance de la classe dérivée **PiePlot**. On peut bien entendu définir un objet Plot explicitement pour ensuite l'affecter au service de notre JFreeChart ou encore configurer le Plot existant, par exemple ainsi :

```
PiePlot pp = (PiePlot)jfc.getPlot();  
pp.setSectionPaint("Prolétariat uni", new Color(255,0,0)); // logique, non ?
```

En fait, l'objet Plot contient un **Renderer** qui permet de configurer les séries de couleurs, etc ...

Comme d'habitude, un objet Plot peut se faire escorter de listeners :

```
public void addChangeListener(PlotChangeListener listener)  
public void notifyListeners(PlotChangeEvent event)
```

et déclare des méthodes prévisibles :

```
public void setOutlineVisible(boolean visible)  
public void setBackgroundImage(java.awt.Image image)  
...
```

La classe dérivée PiePlot possède ses méthodes propres comme :

```
public void setSectionPaint(Comparable key, Paint paint)  
public void setStartAngle(double angle)
```

Pour cette dernière méthode, tout est horlogique par défaut : le point de départ est à 90° et dans le sens négatif d'un point de vue trigonométrique.

3.5 Le composant container visuel

L'objet JFreeChart ainsi construit peut maintenant être placé dans un composant graphique permettant son affichage, donc un objet de type JFrame ou JPanel. La librairie JFreeChart fournit des classes dérivées des classes de Swing qui réalisent le travail d'intégration du graphique : ChartFrame et ChartPanel. Les constructeurs attendus sont bien disponibles :

```
public ChartFrame (java.lang.String title, JFreeChart chart)  
public ChartFrame (java.lang.String title, JFreeChart chart, boolean scrollPane)
```

et l'on peut donc imaginer dans l'application support :

```
ChartFrame fr = new ChartFrame("Résultats", jfc);
```

```
fr.pack();
fr.setVisible(true)
```

Cependant, le plus souvent, ce sera un panel qui sera l'élément de visualisation :

```
public ChartPanel (JFreeChart chart)
```

en remarquant, en passant, des méthodes alléchantes comme

```
public void doSaveAs () throws java.io.IOException
public void createChartPrintJob ()
```

Mais bref :

```
ChartPanel cp = new ChartPanel(jfc);
setContentPane(cp);
```

3.6 L'application complète

En résumé :

FenAppChart.java

```
/*
 * FenAppChart.java
 */

package jfreechartconcepts;

import java.awt.Panel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;

/**
 * @author Vilvens
 */

public class FenAppChart extends javax.swing.JFrame
{
    public FenAppChart()
    {
        initComponents();
        showPieChart();
    }

    private void initComponents() { ... }
```

```

private void showPieChart()
{
    // 1. Définir un dataset qui contient les data
    DefaultPieDataset ds = new DefaultPieDataset();
    ds.setValue("Parti du progrès contrôlé", 22.36);
    ds.setValue("Parti démocrate conservateur", 27.69);
    ds.setValue("Intérêts des gens riches", 3.78);
    ds.setValue("Prolétariat uni", 7.85);
    ds.setValue("Mouvement des Gens Heureux", 35.12);
    ds.setValue("Autres", 3.2);

    // 2. Se fournir un JFreeChart
    JFreeChart jfc = ChartFactory.createPieChart(
        "Résultats des élections en Boursoulavie", ds, true, true, true);

    // 3. Fabriquer le Panel
    ChartPanel cp = new ChartPanel(jfc);
    setContentPane(cp);
}

public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new FenAppChart().setVisible(true);
        }
    });
}
}

```

Le résultat est bien celui annoncé en début de paragraphe ☺

3.7 La cascade des événements

Il importe de bien remarquer le fonctionnement interne des éléments de la librairie, peu manifeste ici parce que le résultat est affiché en une seule fois. En fait, à chaque ajout de donnée au Dataset, une succession d'événements "à la Java Beans" sont engendrés :

- ➔ **DatasetChangeEvent** : reçu par l'objet Plot qui est DatasetChangeListener; il émet ...
- ➔ **PlotChangeEvent** : reçu par l'objet Chart qui est PlotChangeListener; il émet ...
- ➔ **ChartChangeEvent** : reçu par l'objet ChartPanel qui est ChartChangeListener; il redessine l'entièreté du dessin (ce qui n'est pas très performant dans le cas d'animations, mais bon).

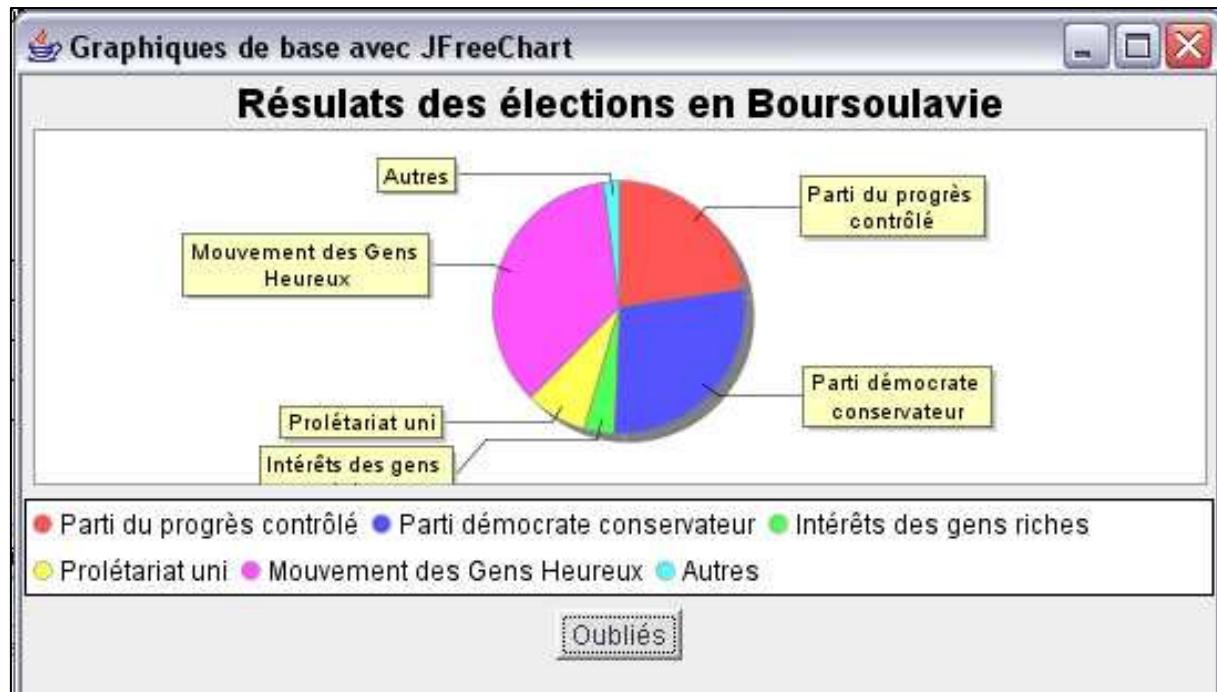
Donc, si on ajoute à la fenêtre de l'application un bouton "Oubliés" et que l'événement Action de ce bouton a pour effet d'ajouter une donnée au Dataset :

```

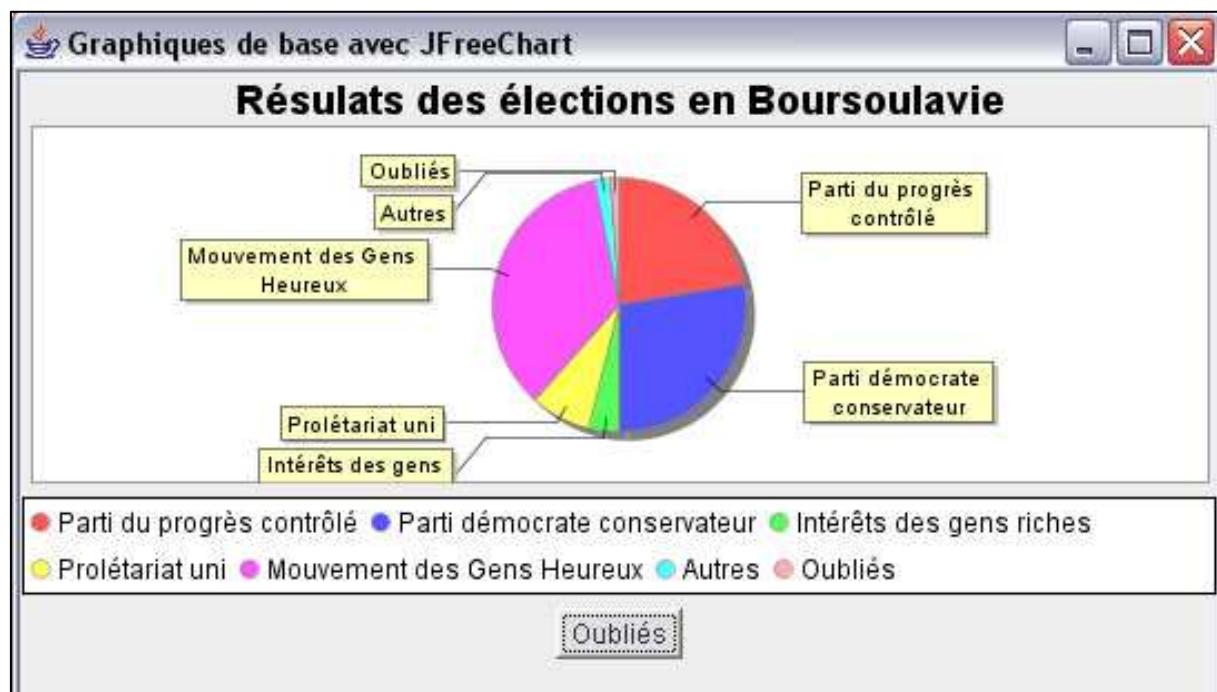
public void actionPerformed(ActionEvent e)
{
    ds.setValue("Oubliés", 1.2); // ds est devenu une variable membre
}

```

l'appui sur le bouton aura pour effet de mettre le graphique à jour. Ainsi, si d'abord :

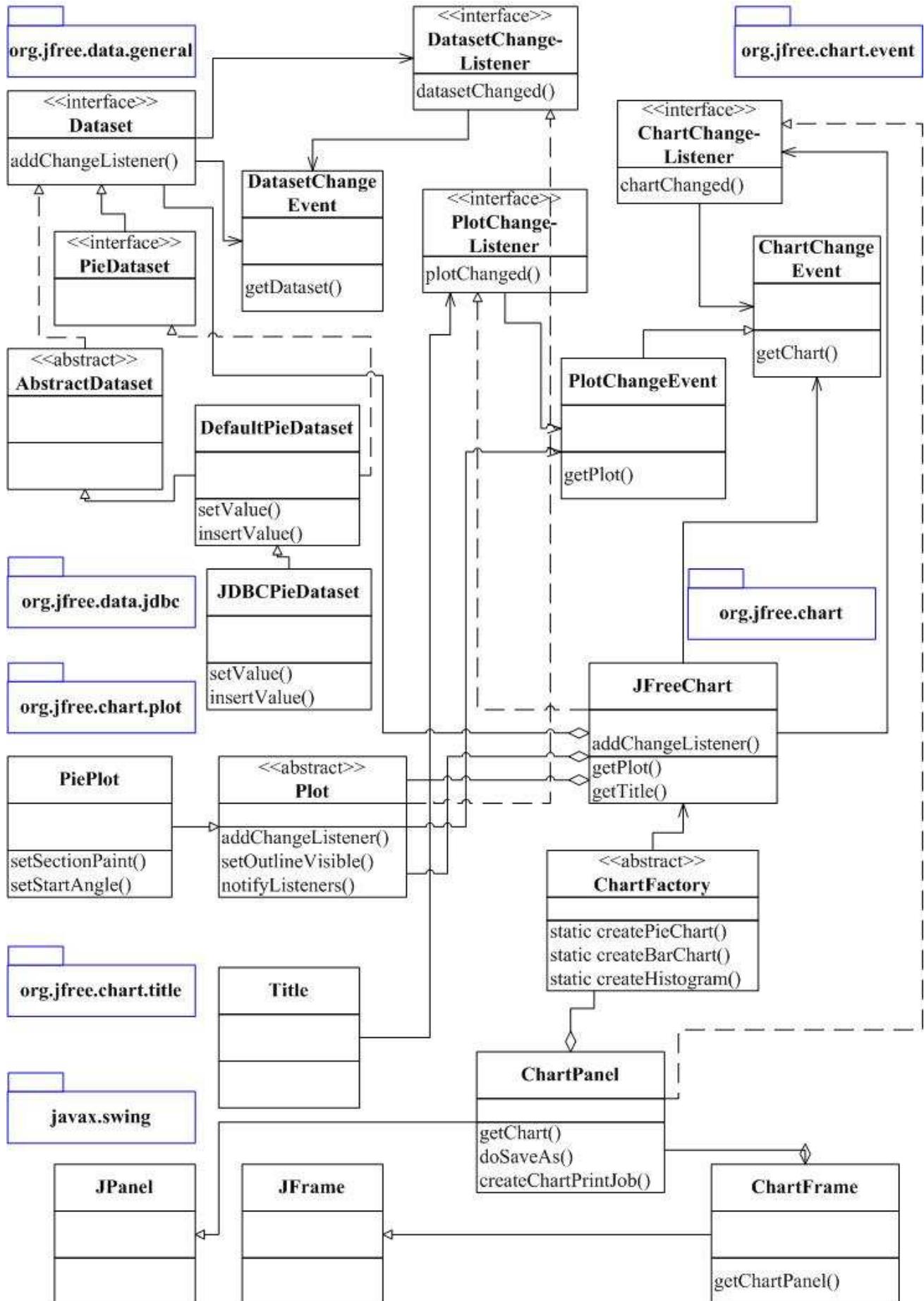


on aura, après appui :



3.8 Le diagramme de classes UML

Sans doute n'est-il pas superflu de visualiser les relations entre les différents intervenants :



4. L'utilisation d'une base de données

4.1 Une classe facilitatrice

Un lecteur attentif aura remarqué la présence d'une classe non encore évoquée :

JDBCPieDataset, dérivée de DefaultPieDataset. Comme on l'aura compris, il est possible de créer un graphique statistique à partir d'une base de données et la librairie fournit la classe qui facilite ce travail (nous pourrions nous en passer et accéder aux données par nos propres moyens, mais quel intérêt ?), en l'occurrence **JDBCPieDataset** qui matérialise un dataset à connecter sur une base de données.

Mais campons tout d'abord le décor ...

4.2 Les données dans une base de données

Avec derrière la tête l'idée de comparer les élections de 2008 à celles de 2004, exécutons les instructions SQL suivantes avec MySQL :

```
mysql> create database Elections;
```

```
mysql> use Elections;
```

```
mysql> grant all privileges on Elections to 'CFBUNAT'@'localhost' identified by  
'ViveLeParti' with grant option;
```

```
mysql> create table Boursoulavie2004 (parti VARCHAR(30), pourcents FLOAT);
```

```
mysql> grant all privileges on Boursoulavie2004 to 'CFBUNAT'@'localhost' identified by  
'ViveLeParti' with grant option;
```

```
mysql> insert into Boursoulavie2004 values ('Parti du progrès contrôlé', 18.25);  
mysql> insert into Boursoulavie2004 values ('Parti démocrate conservateur', 32.09);  
mysql> insert into Boursoulavie2004 values ('Intérêts des gens riches', 1.52);  
mysql> insert into Boursoulavie2004 values ('Prolétariat uni', 15.77);  
mysql> insert into Boursoulavie2004 values ('Mouvement des Gens Heureux', 24.43);  
mysql> insert into Boursoulavie2004 values ('Autres', 7.94);
```

```
mysql> select * from Boursoulavie2004;
```

parti	pourcents
Parti du progrès contrôlé	18.25
Parti démocrate conservateur	32.09
Intérêts des gens riches	1.52
Prolétariat uni	15.77
Mouvement des Gens Heureux	24.43
Autres	7.94

```
6 rows in set (0.02 sec)
```

L'utilisateur créé peut effectivement agir :

```
C:\>mysql -u CFBUNAT -pViveLeParti  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 9
```

Server version: 5.0.41-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

```
mysql> use Elections;
Database changed
mysql> select * from Boursoulavie2004;
```

parti	pourcents
Parti du progrès contrôlé	18.25
Parti démocrate conservateur	32.09
Intérêts des gens riches	1.52
Prolétariat uni	15.77
Mouvement des Gens Heureux	24.43
Autres	7.94

6 rows in set (0.03 sec)

```
mysql>
```

4.3 Le diagramme sectoriel à partir d'une base de données

L'utilisation de la classe **JDBCPieDataset** est très simple :

1) on crée une connexion (classe Connection de java.sql) vers la base de donnée souhaitée – le plus souvent, un Java Bean existe déjà pour réaliser ce travail;

2) on instancie un JDBCPieDataset sur cette connexion en utilisant le constructeur :

```
public JDBCPieDataset (Connection con)
```

Le Dataset ainsi créé est vide (car il se pourrait que l'on puisse le remplir avec différentes données, à choisir dynamiquement).

3) on remplit [*populate*] le Dataset au moyen de sa méthode :

```
public void executeQuery (String query) throws java.sql.SQLException
```

En adaptant notre programme précédent, cela donne donc :

FenAppChart.java (2)

```
/*
 * FenAppChart.java
 */

package jfreechartconcepts;

import com.mysql.jdbc.Connection;
import java.awt.GridLayout;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.data.general.DefaultPieDataset;
import org.jfree.data.jdbc.JDBCPieDataset;

<太后
 * @author Vilvens
 */

public class FenAppChart extends javax.swing.JFrame
{
    public FenAppChart()
    {
        initComponents();
        showPieChart();
        showPieChartJdbc();
    }

    private void initComponents() { ... }

    public static void main(String args[])
    {
        java.awt.EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                new FenAppChart().setVisible(true);
            }
        });
    }

    private void showPieChart()
    {
        // 1. Définir un dataset qui contient les data
        //DefaultPieDataset
        DefaultPieDataset ds = new DefaultPieDataset();
        ds.setValue("Parti du progrès contrôlé", 22.36);
        ...

        // 2. Se fournir un JFreeChart
        JFreeChart jfc = ChartFactory.createPieChart(
            "Résultats des élections en Boursoulavie en 2008", ds, true, true, true);

        // 3. Fabriquer le Panel
        ChartPanel cp = new ChartPanel(jfc);

        this.getContentPane().setLayout(new GridLayout(2,1));
        this.getContentPane().add(cp);
    }
}
```

```

private void showPieChartJdbc()
{
    JDBCPIEDataset jds = null;

    String url = "jdbc:mysql://localhost:3306/Elections";
    try
    {
        Class.forName("com.mysql.jdbc.Driver");
    }
    catch (ClassNotFoundException e)
    {
        System.err.println(e.getMessage());
    }
    System.out.println("Driver MySQL (com) chargé");

    Connection con = null;
    try
    {
        con = (Connection) DriverManager.getConnection(
            url, "CFBUNAT", "ViveLeParti");
        System.out.println("Connexion à la BDD Elections réalisée");
        jds = new JDBCPIEDataset(con);
        String req = "SELECT * FROM Boursoulavie2004;";
        jds.executeQuery(req);
        System.out.println("Dataset chargé");
        con.close();
    }
    catch (SQLException e)
    {
        System.err.println(e.getMessage());
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }

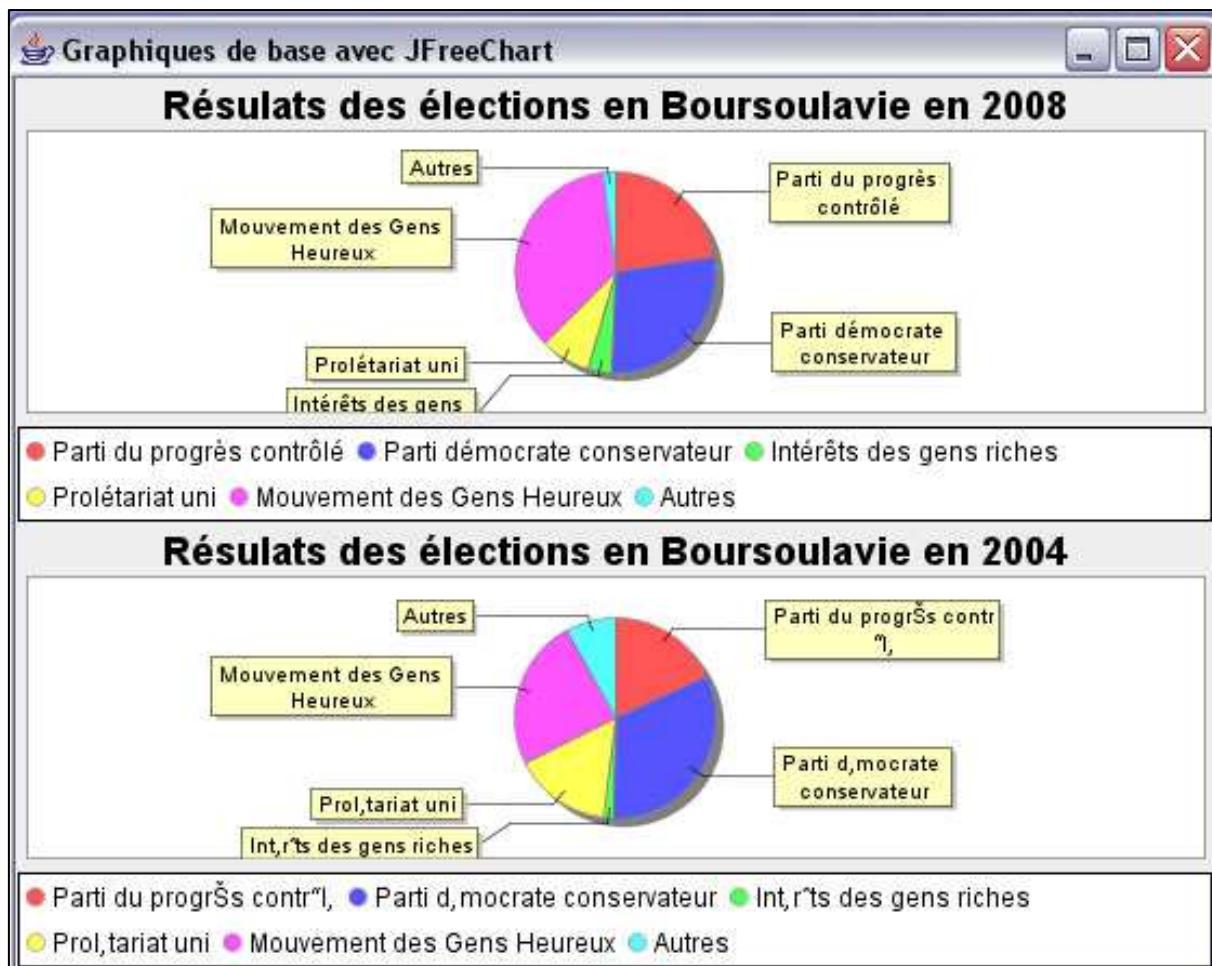
    JFreeChart jfcbd = ChartFactory.createPieChart(
        "Résultats des élections en Boursoulavie en 2004",
        jds,
        true, // générer des légendes ?
        true, // générer des tooltips ?
        true); // générer des URLs?

    // 3. Fabriquer le Panel
    ChartPanel cpbd = new ChartPanel(jfcbd);

    this.getContentPane().add(cpbd);
}
}

```

avec pour résultat :



Tiens, tiens ... Un petit problème de charset ;-);-)?

On peut encore savoir qu'il existe des versions polymorphes du constructeur :

```
public JDBCPieDataset (String url, String driverName, String user, String password)
    throws java.sql.SQLException, java.lang.ClassNotFoundException
public JDBCPieDataset (Connection con, String query)
    throws java.sql.SQLException
```

ainsi que de la méthode d'exécution de la requête :

```
public void executeQuery (Connection con, String query)
    throws java.sql.SQLException
```

Evidemment, la première forme du constructeur dissimule l'objet Connection et, dans ce cas, il faut utiliser la méthode

```
public void close()
```

qui permet de refermer celle-ci.

5. D'autres graphiques classiques

5.1 Les histogrammes comparés

Après le diagramme sectoriel, les histogrammes sont les graphiques les plus populaires, en même temps que les graphiques d'évolution ("lignes brisées"). On s'en doute, la classe **ChartFactory** est capable de les générer avec par exemple :

```
public static JFreeChart createHistogram(String title,  
                                         String xTextLabel,  
                                         String yTextLabel,  
                                         IntervalXYDataset dataset,  
                                         PlotOrientation orientation,  
                                         boolean legend,  
                                         boolean tooltips,  
                                         boolean urls)
```

Mais le plus souvent, il s'agit de comparer les histogrammes (les "bar charts") et nous utiliserons d'emblée :

```
public static JFreeChart createBarChart(String title,  
                                         String categoryTextLabel,  
                                         String valueTextLabel,  
                                         CategoryDataset dataset,  
                                         PlotOrientation orientation,  
                                         boolean legend,  
                                         boolean tooltips,  
                                         boolean urls)
```

Le dataset à utiliser reste ici un **DefaultCategoryDataset**, dont la filiation est calquée sur celle des **DefaultPieDataset** : elle implémente la classe abstraite **AbstractDataset** qui elle-même implémente l'interface **Dataset**. Il est caractérisé par le fait que les valeurs qu'il contient (qui fixeront les valeurs en Y) dépendent de deux paramètres (on peut encore parler de *clés*) : celui qui détermine les groupes de rectangles de l'histogramme comparé et celui qui est à porter en X. Ainsi, par exemple, si nous nous intéressons aux productions de bouteilles de lait par jour dans un groupe de laiteries (disons que divers ateliers répartis géographiquement font partie de la même entreprise), ce type de graphique est idéal puisque les deux clés sont le jour et l'atelier. Un tel Dataset est instancié vide, puis se construit au moyen de la méthode :

```
public void addValue(double value, Comparable rowKey, Comparable columnKey)
```

Les deux derniers paramètres correspondent bien sûr aux deux paramètres évoqués plus haut : on peut considérer que l'on peut représenter les données dans une matrice, ce qui explique les noms données à ces paramètres dans la documentation ... Ce ne sont pas forcément des chaînes de caractères : on leur demande juste d'être Comparable (ce qui n'exige que la méthode compareTo()).

Sans finasser, supposons que nous créons le Dataset suivant :

```
String jour1 = "lundi", jour2 = "vendredi";  
String atelier1 = "Huy", atelier2 = "Verviers", atelier3 = "Liège", atelier4 = "Waremme";
```

```
DefaultCategoryDataset ds = new DefaultCategoryDataset();  
  
ds.addValue(25.3, jour1, atelier1);  
ds.addValue(20.7, jour2, atelier1);  
  
ds.addValue(30.1, jour1, atelier2);  
ds.addValue(34.2, jour2, atelier2);  
  
ds.addValue(85.3, jour1, atelier3);  
ds.addValue(82.1, jour2, atelier3);  
  
ds.addValue(19.6, jour1, atelier4);  
ds.addValue(15.9, jour2, atelier4);
```

Comme pour un PieChart, il nous suffit à présent de créer un BarChart par :

```
JFreeChart jfc = ChartFactory.createBarChart(  
    "Productions de bouteilles de lait (en milliers)",  
    "Ateliers",  
    "Production",  
    ds,  
    PlotOrientation.VERTICAL,  
    true, true, false  
)
```

On remarquera, outre l'apparition de titres pour les axes, l'utilisation de la constante indiquant l'orientation du graphique, membre statique de la classe **PlotOrientation** qui se limite à cela :

```
public static final PlotOrientation HORIZONTAL  
public static final PlotOrientation VERTICAL
```

L'application minimale suivante :

FenAppChart.java (3)

```
/*  
 * FenAppChart.java  
 */  
  
package jfreechartconcepts;  
  
import com.mysql.jdbc.Connection;  
import java.awt.GridLayout;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import org.jfree.chart.ChartFactory;  
import org.jfree.chart.ChartPanel;  
import org.jfree.chart.JFreeChart;  
import org.jfree.chart.plot.PlotOrientation;  
import org.jfree.data.category.DefaultCategoryDataset;
```

```

/***
 * @author Vilvens
 */

public class FenAppChart extends javax.swing.JFrame
{
    public FenAppChart()
    {
        initComponents();
        getContentPane().setLayout(new GridLayout(1,1));
        showHistogram();
    }

    private void initComponents() { ... }

    private void showHistogram()
    {
        String[] jours = {"lundi", "vendredi"};
        String[] ateliers = {"Huy", "Verviers", "Liège", "Waremme"};

        double[][] valProduction = { {25.3, 20.7}, {30.1, 34.2}, {85.3, 82.1}, {19.6, 15.9} };

        DefaultCategoryDataset ds = new DefaultCategoryDataset();

        for (int i=0; i<ateliers.length; i++)
            for (int j=0; j<jours.length; j++)
                ds.addValue(valProduction[i][j], jours[j], ateliers[i]);

        JFreeChart jfc = ChartFactory.createBarChart(
            "Productions de bouteilles de lait (en milliers)",
            "Ateliers",
            "Production",
            ds,
            PlotOrientation.VERTICAL,
            true, true, false
        );

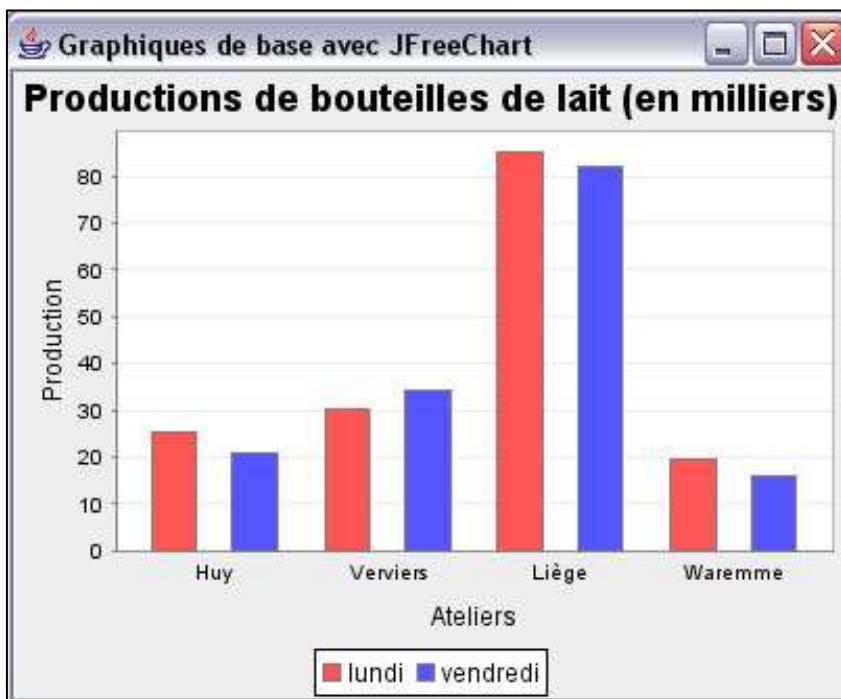
        ChartPanel cp = new ChartPanel(jfc);

        this.getContentPane().add(cp);
    }

    public static void main(String args[])
    {
        java.awt.EventQueue.invokeLater(new Runnable()
        {
            public void run()
            {
                new FenAppChart().setVisible(true);
            }
        });
    }
}

```

Ce qui donne :



A nouveau, l'objet chart utilise une instance d'une classe dérivée de la classe abstraite **Plot**, ici **CategoryPlot**, qui gère la représentation des données et leur référentiel avec des objets instances de **CategoryAxis** et **NumberAxis**, sans oublier un **BarRenderer**. Leur personnalisation approfondie relève de l'examen approfondi de la documentation, ce qui est de peu d'intérêt ici. Contentons-nous de

- ◆ changer la couleur des rectangles : il s'agit bien clairement d'un problème de rendu (comme dans les composants Swing) et c'est donc l'objet Renderer associé au Plot qui peut se charger des changements de couleur : on l'obtient avec

```
public CategoryItemRenderer getRenderer()
```

et l'objet obtenu (pour nous, ce sera plus précisément un BarRenderer) possède la méthode :

```
void setSeriesPaint (int series, java.awt.Paint paint)
```

Pour rappel, l'interface **Paint** possède notamment comme classe d'implémentation, la classe **Color** - cela donne donc ici, après obtention du JFreeChart jfc :

```
...
    CategoryPlot plot = jfc.getCategoryPlot();
    plot.getRenderer().setSeriesPaint(0, new Color(128, 128, 128));
    plot.getRenderer().setSeriesPaint(1, new Color(0, 255, 0));
...
}
```

changer les repères de l'axe vertical : à nouveau, l'objet Plot (ou plutôt CategoryPlot pour être exact) peut nous fournir un objet matérialisant l'axe :

```
public ValueAxis getRangeAxis()
```

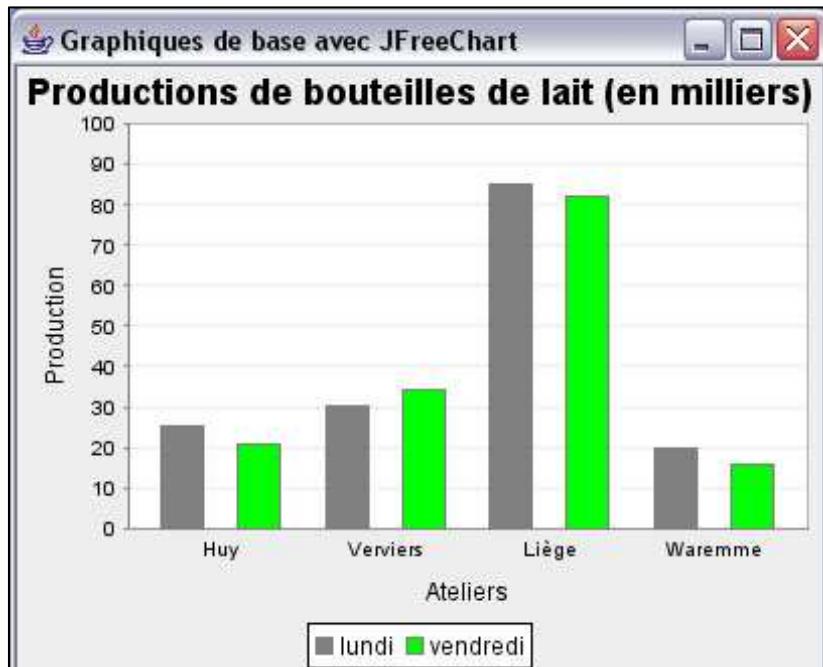
Pour nous, il s'agira plus précisément d'un NumberAxis, dont il n'y a plus qu'à changer le domaine et l'unité utilisée avec :

```
public void setRange (double lower, double upper)  
public void setTickUnit (NumberTickUnit unit)
```

un NumberTickUnit n'étant jamais qu'une encapsulation d'un double - cela donne donc ici, à la suite des instructions précédentes :

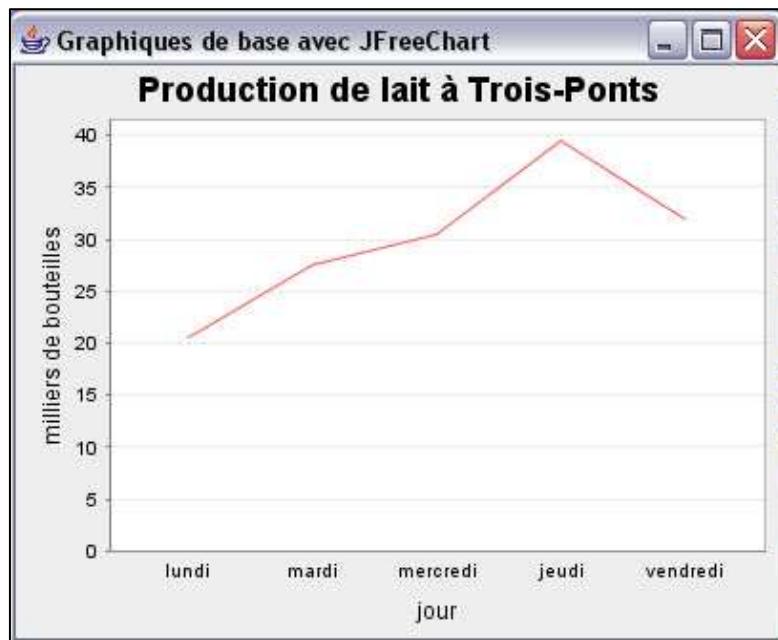
```
...  
    NumberAxis rangeAxis = (NumberAxis) plot.getRangeAxis();  
    rangeAxis.setRange(0.0, 100.0);  
    rangeAxis.setTickUnit(new NumberTickUnit(10.0));  
....
```

L'exécution donnera :



5.2 Les graphiques linéaires d'évolution

Sans entrer dans les détails, signalons l'existence des graphiques linéaires classiques dont le but est de matérialiser une évolution (les "lignes brisées" classiques). L'idée est pas exemple de réaliser le graphique suivant :



Le dataset est analogue à celui des histogrammes. Mais on utilise cette fois la factory :

```
public static JFreeChart createLineChart(
    String title,
    String categoryAxisLabel,
    String valueAxisLabel,
    CategoryDataset dataset,
    PlotOrientation orientation,
    boolean legend,
    boolean tooltips,
    boolean urls)
```

Point n'est sans doute trop besoin d'insister :

FenAppChart.java (4)

```
...
private void showEvolution()
{
    String[] jours = {"lundi", "mardi", "mercredi", "jeudi", "vendredi"};
    double[] valProduction = { 20.5, 27.6, 30.4, 39.5, 31.9 };

    DefaultCategoryDataset ds = new DefaultCategoryDataset();
    for (int i=0; i< jours.length; i++)
        ds.addValue(valProduction[i], "Bouteilles", jours[i]);

    JFreeChart jfc = ChartFactory.createLineChart
        ("Production de lait à Trois-Ponts", "jour", "milliers de bouteilles",
         ds, PlotOrientation.VERTICAL, false, true, false);
    ChartPanel cp = new ChartPanel(jfc);
    this.getContentPane().add(cp);
}
...
```

6. Les graphiques de statistique à deux dimensions

6.1 Le nuage de points

Pour retrouver le monde de la statistique à deux dimensions, considérons par exemple le problème suivant (voir "Inférence statistique", chapitre VI – du même auteur). Le psychologue attaché à une entreprise soupçonne qu'il existe une corrélation entre la perception visuelle et la dextérité manuelle des ouvriers travaillant sur une chaîne d'assemblage. Il va donc vérifier cette assertion en faisant passer (durant les heures de travail) un test de réponse à un stimulus visuel (soit **X** la VA correspondante) et un autre test de dextérité manuelle (soit **Y** la VA correspondante) à un échantillon de 15 ouvriers choisis au hasard (on peut donc même y trouver un délégué syndical ...). Les résultats donnent 15 couples de valeurs (résultat perception visuelle, résultat dextérité manuelle) :

xi	yi		
3,8	68	3,5	74
3,6	72	3,8	59
3,4	86	4,1	55
3,5	78	3,7	64
3,9	64	3,6	73
4,2	61	3,8	62
3,7	66	3,4	75
		3,5	78

La construction du nuage de points correspondant au problème à visualiser est très similaire aux précédents :

- ♦ on crée tout d'abord un Dataset approprié à la situation soit un **XYDataset**. Cet interface dérivé de Dataset est implémenté notamment par la classe **XYSeriesCollection** (qui en fait dérive de la classe abstraite **AbstractIntervalXYDataset** qui implémente l'interface **IntervalXYDataset** dérivé de **XYDataset** – ouf !). Cette classe est instanciée le plus souvent avec un constructeur par défaut et enregistre des séries statistiques à deux dimensions (donc des listes de couples (x,y)) au moyen de sa méthode :

```
public void addSeries(XYSeries series)
```

- ♦ il faut donc créer le (ou les) série(s) qui sont des instances de la classe **XYSeries**. Le constructeur usuel est

```
public XYSeries (Comparable key)
```

- l'idée est que l'on peut ainsi retrouver chaque série par une clé – usuellement, un simple nom suffit. Les valeurs peuvent être dupliquées (un constructeur polymorphe permet d'interdire cela). On place des couples dans la série par la méthode :

```
public void add (double x, double y)
```

On s'en doute un peu, l'ajout d'un couple provoque l'envoi d'un **SeriesChangeEvent** à tous les listeners intéressés (ce qui est le cas du Dataset par l'intermédiaire de sa classe mère **AbstractIntervalXYDataset**).

- ♦ on peut enfin appeler la méthode factory de **ChartFactory** qui créera le nuage de points à partir du Dataset :

```
public static JFreeChart createScatterPlot (
        java.lang.String title,
        java.lang.String xAxisLabel,
        java.lang.String yAxisLabel,
        XYDataset dataset,
        PlotOrientation orientation,
        boolean legend,
        boolean tooltips,
        boolean urls)
```

et, comme d'habitude, il n'y a plus qu'à créer le ChartPanel et à le placer sur le ContentPane.

En résumé, cela donnera :

FenAppChart.java (5)

```
...
private void showScatterPlot()
{
    double couples[][] = { {3.8,68.0}, {3.6,72.0}, {3.4,86.0}, {3.5,78.0}, {3.9,64.0},
        {4.2,61.0}, {3.7,66.0}, {3.5,74.0}, {3.8,59.0}, {4.1,55.0},
        {3.7,64.0}, {3.6,73.0}, {3.8,62.0}, {3.4,75.0}, {3.5,78.0} };

    XYSeries serieObs = new XYSeries("Relations vision-manipulation");

    for (int i=0; i<15; i++)
        serieObs.add(couples[i][0],couples[i][1]);

    XYSeriesCollection ds = new XYSeriesCollection();
    ds.addSeries(serieObs);

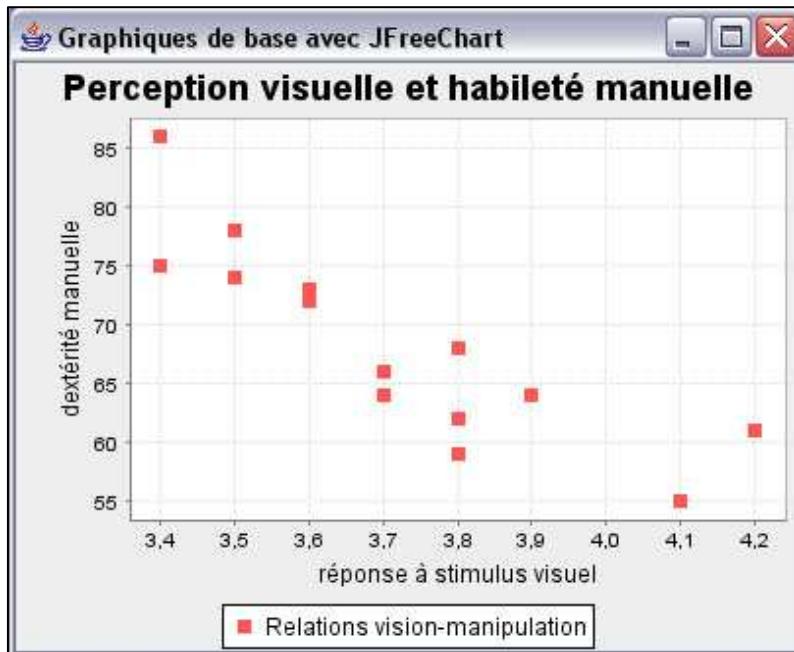
    JFreeChart jfc = ChartFactory.createScatterPlot(
        "Perception visuelle et habileté manuelle",
        "réponse à stimulus visuel", "dextérité manuelle",
        ds,
        PlotOrientation.VERTICAL,
        true, true, false );

    ChartPanel cp = new ChartPanel(jfc);

    this.getContentPane().add(cp);

}
```

Résultat :



Il est à soupçonner que nous soyons devant un phénomène de corrélation négative ...

6.2 Les paramètres de régression et de corrélation

La librairie fournit une classe utilitaire nommée **Statistics** qui possède un certain nombre de méthodes statiques permettant les calculs de moyenne et écart type pour une série statistique à une dimension, et des paramètres de régression et corrélation pour les séries statistiques à deux dimensions. Les séries considérées sont représentées par des tableaux d'objets **Number**, classe abstraite (du package `java.lang`) dont l'un des descendants est **Double** qui encapsule un réel en double précision. Les méthodes en question sont :

- ◆ public static double **calculateMean**(`java.lang.Number[] values`)
- ◆ public static double **getStdDev**(`java.lang.Number[] data`)

calculent respectivement la moyenne (m) et l'écart-type (s);

- ◆ public static double[] **getLinearFit**(`java.lang.Number[] xData, java.lang.Number[] yData`)

fournit dans le tableau résultat respectivement l'ordonnée à l'origine (b) et la pente (a) de la droite de régression ($y=ax+b$) calculée par une règle des moindres carrés;

- ◆ public static double **getCorrelation**(`java.lang.Number[] data1, java.lang.Number[] data2`)

fournit le coefficient de corrélation linéaire (r).

Nous pouvons donc ajouter à notre méthode dédiée aux nuages de points le code de calcul de ces paramètres. Une fois muni des valeurs de a et b , nous pouvons calculer les valeurs estimées (y_e) et en faire une deuxième série que nous ajouterons à notre Dataset – la magie des events-listeners fera le reste ...

FenAppChart.java (6)

```

...
    private void showScatterPlot()
    {
        ... // nuage de points

        Number[] x = new Double[15];
        Number[] y = new Double[15];
        for (int i=0; i<15; i++)
        {
            x[i] = couples[i][0];
            y[i] = couples[i][1];
        }
        double[] droiteRegression = Statistics.getLinearFit(x, y);
        double a = droiteRegression[1]; // et pas [0] !
        double b = droiteRegression[0];

        System.out.println("Droite de régression : y = " + a + "x + " + b);
        System.out.println("Droite de régression : y = " + viewDec(a,2)
                           + "x " + (b<0?"":"+" ) + viewDec(b,2));

        double r = Statistics.getCorrelation(x,y);
        System.out.println("Coefficient de corrélation : r = " + r);
        System.out.println("Coefficient de corrélation : r = " + viewDec(r,2));

        double mx = Statistics.calculateMean(x);
        double sx = Statistics.getStdDev(x);
        System.out.println("VA X : m = " + mx + " et s = " + sx);
        System.out.println("VA X : m = " + viewDec(mx,2) + " et s = " + viewDec(sx,2));

        double my = Statistics.calculateMean(y);
        double sy = Statistics.getStdDev(y);
        System.out.println("VA Y : m = " + my + " et s = " + sy);
        System.out.println("VA Y : m = " + viewDec(my,2) + " et s = " + viewDec(sy,2));

        // Droite de régression
        XYSeries serieReg = new XYSeries("Droite de régression");
        for (double xe = 3.4; xe<= 4.2; xe+=0.1)
        {
            double ye = a*xe + b;
            System.out.println(xe + "," + ye );
            serieReg.add(xe,ye);
        }
        ds.addSeries(serieReg);
    }

    public String viewDec (double x, int nd)
    {
        NumberFormat nf = NumberFormat.getInstance();

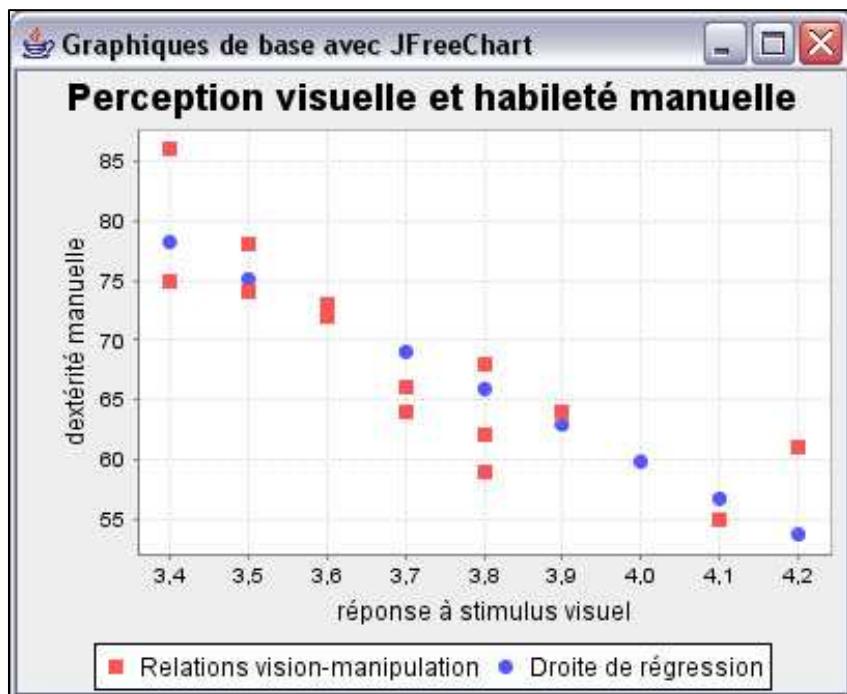
```

```
//System.out.println("Classe instanciée = " + nf.getClass().getName());
nf.setMinimumFractionDigits(nd);
nf.setMaximumFractionDigits(nd);
String str = nf.format(x);
return str;
}
...
```

Sur la console, cela donne :

Droite de régression : $y = -30.75000000000544x + 182.7750000000202$
 Droite de régression : **y = -30,75x +182,78**
 Coefficient de corrélation : $r = -0.8671437999440119$
 Coefficient de corrélation : **r = -0,87**
 VA X : m = 3.7 et s = 0.2390457218668787
 VA X : m = 3,70 et s = 0,24
 VA Y : m = 69.0 et s = 8.476859256655313
 VA Y : m = 69,00 et s = 8,48

et du point de vue graphique :



On remarquera l'opération cosmétique visant à limiter l'affichage des réels à deux décimales. Pour cela, nous utilisons la classe **NumberFormat** (du package `java.text`) ou plutôt, puisqu'elle est abstraite, sa classe dérivée **DecimalFormat**. En fait, nous ignorons cela car nous nous procurons l'objet au moyen de la factory :

`public static final NumberFormat getInstance()`

L'objet ainsi obtenu est capable de nous renvoyer une chaîne de caractères représentant le nombre réel formaté selon nos désirs, ceci au moyen de la méthode :

`public final String format(double number)`

après avoir, au préalable, configurer l'objet de formatage, par exemple avec les méthodes :

```
public void setMinimumFractionDigits (int newValue)  
public void setMaximumFractionDigits (int newValue)
```

7. Les séries chronologiques

7.1 Un axe des X avec des temps

Nous avons déjà réalisé un graphique d'évolution, c'est-à-dire fournissant un diagramme linéaire dont les valeurs sur X étaient fait des données temporelles (des mois, des jours, des temps). En fait, il existe pour ce cas particulier de diagramme en XY

- ♦ une classe dédiacée pour le Dataset associé : la classe **TimeSeriesCollection** (qui dérive de la classe abstraite **AbstractIntervalXYDataset** et est donc une sœur de **XYSeriesCollections**) : si les valeurs x de ce genre de séries sont bien des double, elles représentent en réalité le nombre de millisecondes écoulées depuis le 1^{er} janvier 1970 et sont donc susceptibles d'être transformées en objet Date. C'est le rôle d'un objet **DateAxis** de réaliser cette conversion (mais ceci est transparent dans une utilisation standard).
- ♦ une classe **TimeSeries** pour peupler l'instance de TimeSeriesCollection de séries de données chronologiques : son constructeur le plus utile

```
public TimeSeries(String name, java.lang.Class timePeriodClass)
```

réclame essentiellement le nom de la classe qui fournira les instances successives représentant les temps écoulés régulièrement selon la période fixée par la nature de cette classe (jour, mois, minute, ...). Toutes les classes prévues par la librairie sont sensées implémenter l'interface **TimePeriod** qui déclare les méthodes :

```
java.util.Date getStart()  
java.util.Date getEnd()
```

- en fait, elles dérivent de la classe abstraite **RegularTimePeriod** aux méthodes additionnelles de manipulations des millisecondes encapsulées et aussi pourvue de la déclaration de deux méthodes :

```
public abstract RegularTimePeriod previous()  
public abstract RegularTimePeriod next()
```

Les classes possibles sont bien normalement : Day, FixedMillisecond, Hour, Millisecond, Minute, Month, Quarter, Second, Week et Year.

- ♦ une factory dédiacée :

```
public static JFreeChart createTimeSeriesChart (  
                java.lang.String title,  
                java.lang.String timeAxisLabel,  
                java.lang.String valueAxisLabel,  
                XYDataset dataset,
```

```
boolean legend,
boolean tooltips,
boolean urls)
```

En pratique, supposons souhaiter comparer les données de production de milliers de litres de lait de deux exploitations agricoles en 2006. On peut tout d'abord imaginer, dans un objectif de généricté future, une classe ***Productions*** destinée à contenir des données de production (un tableau de double) pour un type de période donnée (par exemple, par mois) - ce type de période est désigné par une variable membre statique de la classe **Calendar**. De plus, la période supérieure (donc l'année dans le cas de mois) sera également encapsulée. Cela donne :

Productions.java

```
package jfreechartconcepts.data;

import java.util.Calendar;
import jfreechartconcepts.data.InvalidDataProduction;

/**
 * @author Vilvens
 */

public class Productions
{
    private int TypePeriode;
    private double data[];
    private int PeriodeSuperieure;

    public Productions (int t, double d[], int ps) throws InvalidDataProduction
    {
        Calendar c = Calendar.getInstance();
        TypePeriode = t; data = d; PeriodeSuperieure = ps;
        if (data.length != (c.getActualMaximum(TypePeriode)-
            c.getActualMinimum(TypePeriode)+1)) throw new InvalidDataProduction
            ("Nombre de données incohérent avec le type de période");
    }

    public int getTypePeriode() { return TypePeriode; }
    public void setTypePeriode(int TypePeriode) { this.TypePeriode = TypePeriode; }
    public double[] getData() { return data; }
    public void setData(double[] data) { this.data = data; }
    public int getPeriodeSuperieure() { return PeriodeSuperieure; }
    public void setPeriodeSuperieure(int PeriodeSuperieure)
        { this.PeriodeSuperieure = PeriodeSuperieure; }
}
```

avec une classe d'exception élémentaire utilisée si le nombre de données ne correspond pas au type de période choisi (par exemple, 11 données alors que l'on parle en mois d'une année) :

InvalidDataProduction.java

```
package jfreechartconcepts.data;

/**
 * @author Vilvens
 */

public class InvalidDataProduction extends Exception
{
    public InvalidDataProduction() { }

    public InvalidDataProduction(String msg) { super(msg); }
}
```

La réalisation du graphique statistique ne pose alors plus de problèmes :

FenAppChart.java (7)

```
...
private void showTimeSeries()
{
    double [][] ProductionsLait = {
        {65.23, 54.54, 59.71, 45.12,
         32.14, 32.19, 40.84, 46.21,
         47.67, 42.36, 45.65, 55.81 },

        {60.31, 57.54, 62.71, 49.12,
         30.14, 38.19, 44.84, 49.21,
         53.67, 54.36, 49.65, 56.81 }
    };

    String[] lieuxProductions = { "Trois-Ponts", "Stavelot"};
    int annee = 2006;

    Productions[] p = new Productions[2];
    try
    {
        for (int j=0; j<lieuxProductions.length; j++)
            p[j] = new Productions(Calendar.MONTH, ProductionsLait[j], annee);
    }
    catch (InvalidDataProduction e)
    {
        System.out.println("Oh oh ... " + e.getMessage());
    }

    Calendar c = Calendar.getInstance();

    TimeSeriesCollection ds = new TimeSeriesCollection();
    TimeSeries[] s = new TimeSeries[2];
```

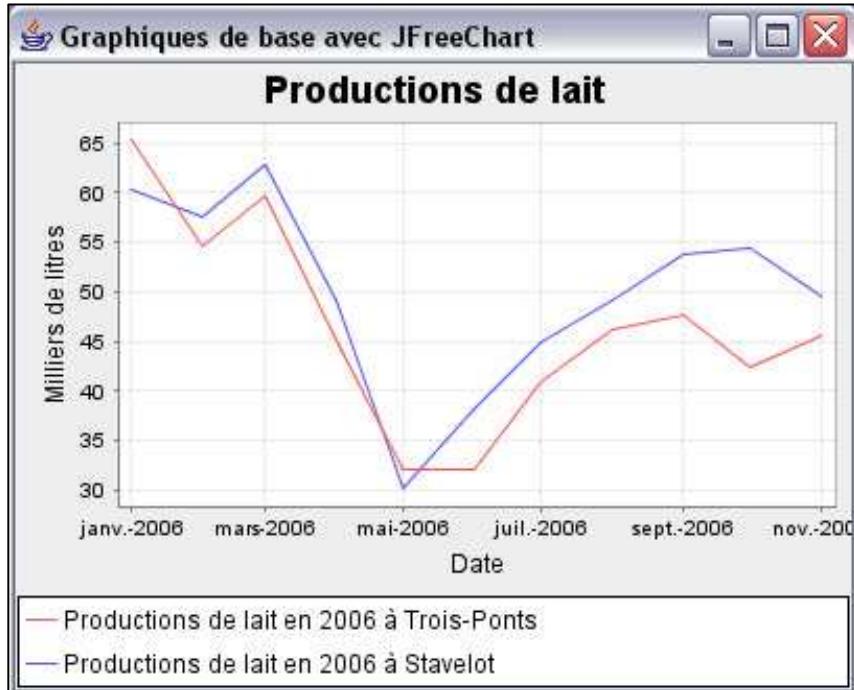
```

for (int j=0; j<lieuxProductions.length; j++)
{
    s[j] = new TimeSeries("Productions de lait en " + p[j].getPeriodeSuperieure() + " à "
        + lieuxProductions[j], Month.class);
    for (int i = 1; i<= c.getActualMaximum(p[j].getTypePeriode()); i++ )
    {
        s[j].add(new Month(i, p[j].getPeriodeSuperieure()), p[j].getData()[i-1]);
    }
    ds.addSeries(s[j]);
}

JFreeChart jfc = ChartFactory.createTimeSeriesChart(
    "Productions de lait",
    "Date", // x
    "Milliers de litres", // y
    ds,
    true, true, false
);
ChartPanel cp = new ChartPanel(jfc);
this.getContentPane().add(cp);
}
...

```

Ce qui nous donne :



7.2 Une graphique temporel à partir d'un base de données

Tout comme pour PieDataSet, il existe pour le XYDataSet un JDBCXYDataSet tout à fait analogue au JDBCPieDataSet. Remarquons cependant sa méthode :

public void setTimeSeries(boolean timeSeries)

permettant d'indiquer que les données considérées sont bien celles d'une série chronologique.

Pour reprendre un exemple similaire au précédent, nous pouvons créer une table contenant les données selon la règle que la 1^{ère} colonne correspond aux données en X (donc, ce sont les temps) tandis que les autres colonnes correspondent aux diverses séries. Cela pourrait donner :

```
mysql> create database GestionProduction;  
Query OK, 1 row affected (0.11 sec)
```

```
mysql> use GestionProduction;  
Database changed
```

```
mysql> grant all privileges on GestionProduction to 'CFSERVGESTION'@'localhost'  
identified by 'ViveLeProfit' with grant option;  
Query OK, 0 rows affected (0.27 sec)
```

```
mysql> CREATE TABLE productions (dateReleve DATE, dataTroisPonts DOUBLE,  
dataStavelot DOUBLE);  
Query OK, 0 rows affected (0.33 sec)
```

```
mysql> grant all privileges on productions to 'CFSERVGESTION'@'localhost' identi  
fied by 'ViveLeProfit' with grant option;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> INSERT INTO productions VALUES ('2006-1-31', 65.20, 60.31);
```

```
Query OK, 1 row affected (0.06 sec)
```

```
mysql> INSERT INTO productions VALUES ('2006-2-28', 65.23, 60.28);  
Query OK, 1 row affected (0.01 sec)
```

```
mysql> INSERT INTO productions VALUES ('2006-3-31', 54.54, 57.54);  
Query OK, 1 row affected (0.03 sec)
```

```
mysql> INSERT INTO productions VALUES ('2006-4-30', 59.71, 62.71);  
Query OK, 1 row affected (0.03 sec)
```

```
mysql> commit;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from productions;
```

dateReleve	dataTroisPonts	dataStavelot
2006-01-31	65.2	60.31
2006-02-28	65.23	60.28
2006-03-31	54.54	57.54
2006-04-30	59.71	62.71

```
4 rows in set (0.00 sec)
```

La méthode de construction du graphique est évidemment un héritage multiple ;-) de celle du graphique chronologique et de celle du graphique en camembert à partir d'une base de données :

FenAppChart.java (8)

```
...
private void showTimeSeriesJdbc()
{
    JDBCXYDataset jds = null;

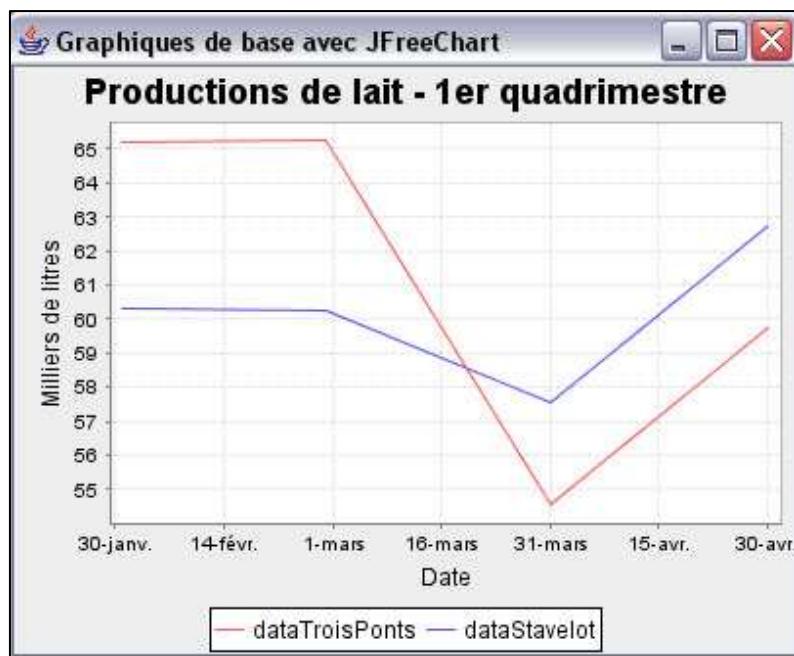
    String url = "jdbc:mysql://localhost:3306/GestionProduction";
    try
    {
        Class.forName("com.mysql.jdbc.Driver");
    }
    catch (ClassNotFoundException e)
    {
        System.err.println(e.getMessage());
    }
    System.out.println("Driver MySQL (com) chargé");

    Connection con = null;
    try
    {
        con = (Connection) DriverManager.getConnection(url, "CFSERVGESTION",
                                                       "ViveLeProfit");
        System.out.println("Connexion à la BDD GestionProduction réalisée");
        jds = new JDBCXYDataset(con);
        jds.setTimeSeries(true);
        String req = "SELECT * FROM productions;";
        jds.executeQuery(req);
        System.out.println("Dataset chargé");
        con.close();
    }
    catch (SQLException e)
    {
        System.err.println(e.getMessage());
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }

    if (jds.isTimeSeries()) System.out.println("Série temporelle");
    else System.out.println("Série PAS temporelle");
    JFreeChart jfc = ChartFactory.createTimeSeriesChart(
        "Productions de lait - 1er quadrimestre",
        "Date", // x
        "Milliers de litres", // y
        jds,
        true, true, false
    );
    ChartPanel cp = new ChartPanel(jfc);
    this.getContentPane().add(cp);

}
...
```

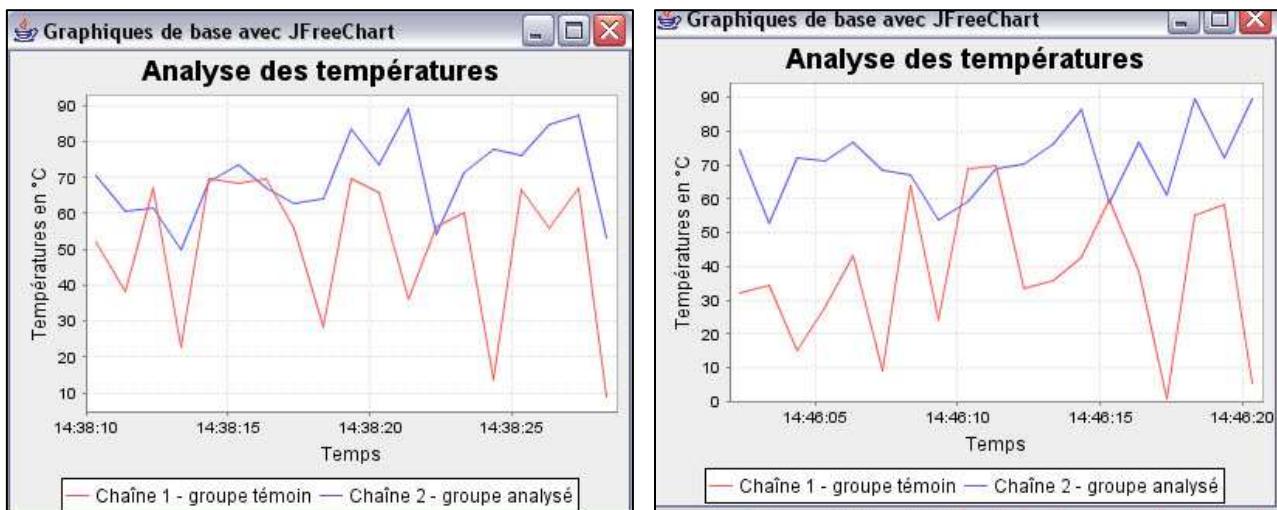
Le résultat cache cependant une petite surprise :



Vous avez vu laquelle ;-) ?

8. Les graphiques dynamiques

Une application pratique simple à réaliser consiste à visualiser un phénomène sur un graphique qui tient compte de nouvelles données au fur et à mesure qu'elles sont disponibles. Typiquement il s'agit par exemple de la surveillance de chaînes de fabrications. Non seulement de nouvelles données sont ajoutées, mais les plus anciennes sont éliminées afin de ne pas surcharger le graphique. On vise donc un résultat du type suivant, qui compare des mesures de températures relevées sur deux chaînes similaires, la première utilisant un groupe témoin :



Nous utiliserons pour cela un objet **Timer** (du package javax.swing) qui émet à intervalles réguliers un événement ActionEvent. Ici, il va modifier périodiquement les deux datasets utilisés. Son constructeur

```
public Timer(int delay, ActionListener listener)
```

réclame simplement sa période (en millisecondes) et, éventuellement, l'objet listener qui va réagir à l'émission de l'événement. On aura bien compris qu'il s'agit d'un thread dédié, que l'on fait démarrer avec sa méthode :

```
public void start()
```

L'addActionListener sera ici l'application elle-même et sa méthode actionPerformed() ajoutera une nouvelle donnée à chaque série de données – ces nouvelles données devraient provenir, par exemple, d'une socket ou d'une ligne série, mais nous les simulerons ici à coup de générateur de nombres aléatoires. Petite subtilité : un compteur de mesures observées est nécessaire afin que, parvenu à une certaine quantité de données, les premières soient éliminées des datasets au moyen de la méthode des TimeSeries ::

```
public void delete(int start, int end)
```

Sans trop insister, on peut donc écrire :

FenAppChart.java (9)

```
/*
 * FenAppChart.java
 */

package jfreechartconcepts;

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Timer;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
...
/**

 * @author Vilvens
 */

public class FenAppChart extends javax.swing.JFrame implements ActionListener
{
    private TimeSeries chaine1, chaine2;
    private int cptObs;
    //private DefaultPieDataset ds;
```

```

public FenAppChart()
{
    initComponents();
    getContentPane().setLayout(new GridLayout(1,1));
    cptObs = 0;

/* showPieChart(); showPieChartJdbc(); showHistogram(); showEvolution();
   showScatterPlot(); showTimeSeries(); showTimeSeriesJdbc();
*/
    showTimeSeriesEvolution();
}

private void initComponents() { ... }

private void showTimeSeriesEvolution()
{
    chaine1 = new TimeSeries("Chaîne 1 - groupe témoin", Millisecond.class);
    chaine2 = new TimeSeries("Chaîne 2 - groupe analysé", Millisecond.class);
    TimeSeriesCollection ds = new TimeSeriesCollection();
    ds.addSeries(chaine1);
    ds.addSeries(chaine2);

JFreeChart jfc = ChartFactory.createTimeSeriesChart(
    "Analyse des températures",
    "Temps", // x
    "Températures en °C", // y
    ds,
    true, true, false
);

ChartPanel cp = new ChartPanel(jfc);
this.getContentPane().add(cp);

Timer t =new Timer (1000, this);
t.start();
}

public void actionPerformed(ActionEvent e)
{
    chaine1.add(new Millisecond(), Math.abs(70*Math.sin(aleat(60.0,80.0))));
    chaine2.add(new Millisecond(), aleat(50.0,90.0));
    cptObs++;
    if (cptObs >= 20) { chaine1.delete(0,0) ; chaine2.delete(0,0) ; }
}

private double aleat (double bi, double bs)
{
    return bi + Math.random()*(bs-bi);
}
...
}

```

9. Les graphiques statistiques dans les applications Web

9.1 Une applet d'affichage

On pourrait évidemment imaginer d'afficher des graphiques statistique au sein d'une applet : il suffirait de placer les instructions correspondantes dans la méthode init() de l'applet. Cela fonctionne (à condition que le browser utilisé possède la plugin Java nécessaire) mais avec le terrible handicap du transfert des jars de librairie nécessaires en plus du jar de l'applet elle-même. Il faut en effet utiliser un tag <APPLET> du genre lourd :

```
<APPLET ARCHIVE="appletchart.jar, jfreechart-1.0.6.jar, jcommon-1.0.10.jar"
CODE="charts.appletNuagePoints" width=640 height=260
ALT="Si vous voyez ce texte, il y a un problème avec votre browser ...">
</APPLET>
```

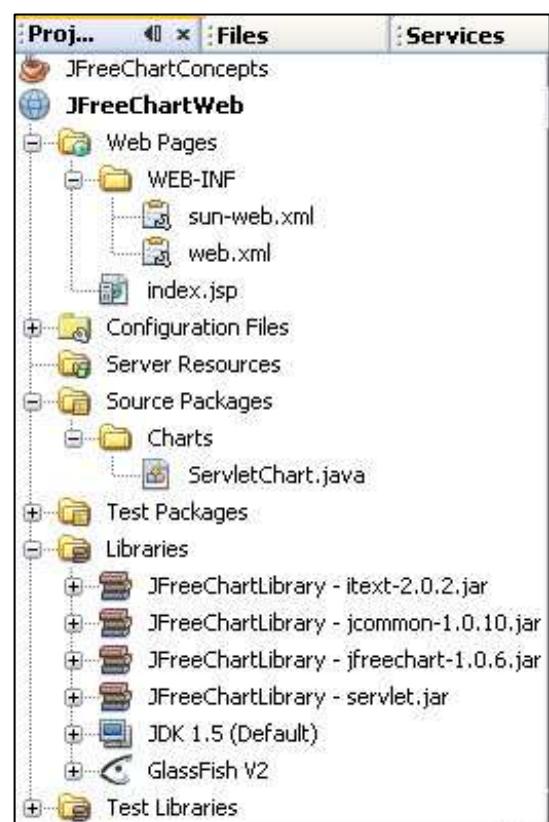
De toute manière, les restrictions de sécurité d'une applet font que, dans un contexte où l'accès aux données est toujours nécessaire, la solution d'une ou plusieurs servlets doit être préférée.

9.2 Une servlet d'affichage

Ce qui vient immédiatement à l'esprit est évidemment de faire générer le graphique par une servlet qui, du côté serveur, utilisera la librairie JFreeChart. De fait, il n'y a pas de difficulté réelle à travailler ainsi puisqu'il existe une classe **ChartUtilities** (package org.jfree.chart) qui possède les méthodes :

```
public static void writeChartAsJPEG(java.io.OutputStream out, JFreeChart chart, int width,
int height) throws java.io.IOException
public static void writeChartAsPNG(java.io.OutputStream out, JFreeChart chart, int width,
int height) throws java.io.IOException
```

à la fonction bien claire. On conçoit sans peine qu'il suffit de passer comme premier paramètre le flux de sortie de la servlet pour que le protocole HTTP apporte au client Web le graphique souhaité. Et de fait, si nous créons une application Web destinée à tourner sur le serveur GlassFish intégré et à laquelle nous avons attaché la librairie JFreeChart :



nous pouvons écrire une servlet élémentaire :

ServletChart

```
package Charts;

import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

/**
 * @author Vilvens
 */

public class ServletChart extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        response.setContentType("image/jpg");
        OutputStream out = response.getOutputStream();
        try
        {
            showScatterPlot (out);
        }
        finally { out.close(); }
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException { processRequest(request, response); }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException { processRequest(request, response); }

    public String getServletInfo()
    { return "Servlet fournissant un graphique en nuage de points"; }

    synchronized private void showScatterPlot (OutputStream os) throws IOException
    {
        double couples[][] = { {3.8,68.0}, {3.6,72.0}, {3.4,86.0}, {3.5,78.0}, {3.9,64.0},
            {4.2,61.0}, {3.7,66.0}, {3.5,74.0}, {3.8,59.0}, {4.1,55.0},
            {3.7,64.0}, {3.6,73.0}, {3.8,62.0}, {3.4,75.0}, {3.5,78.0} };
    }
}
```

```
XYSeries serieObs = new XYSeries("Relations vision-manipulation");

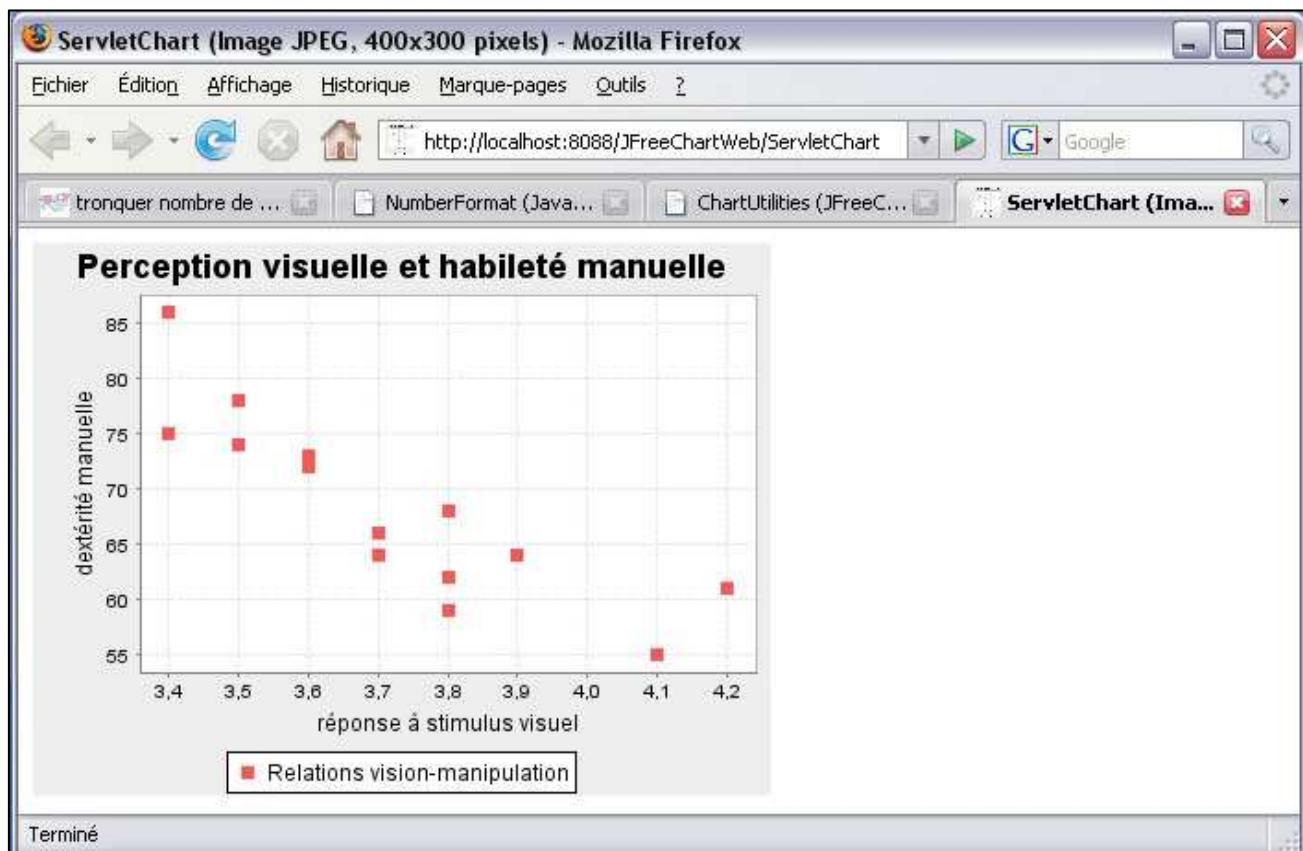
for (int i=0; i<15; i++)
    serieObs.add(couples[i][0],couples[i][1]);

XYSeriesCollection ds = new XYSeriesCollection();
ds.addSeries(serieObs);

JFreeChart jfc = ChartFactory.createScatterPlot(
    "Perception visuelle et habileté manuelle",
    "réponse à stimulus visuel", "dextérité manuelle",
    ds,
    PlotOrientation.VERTICAL,
    true, true, false );

ChartUtilities.writeChartAsJPEG (os, jfc, 400, 300);
}
```

Le résultat est simple mais foudroyant :



9.3 Un servlet fournit une image

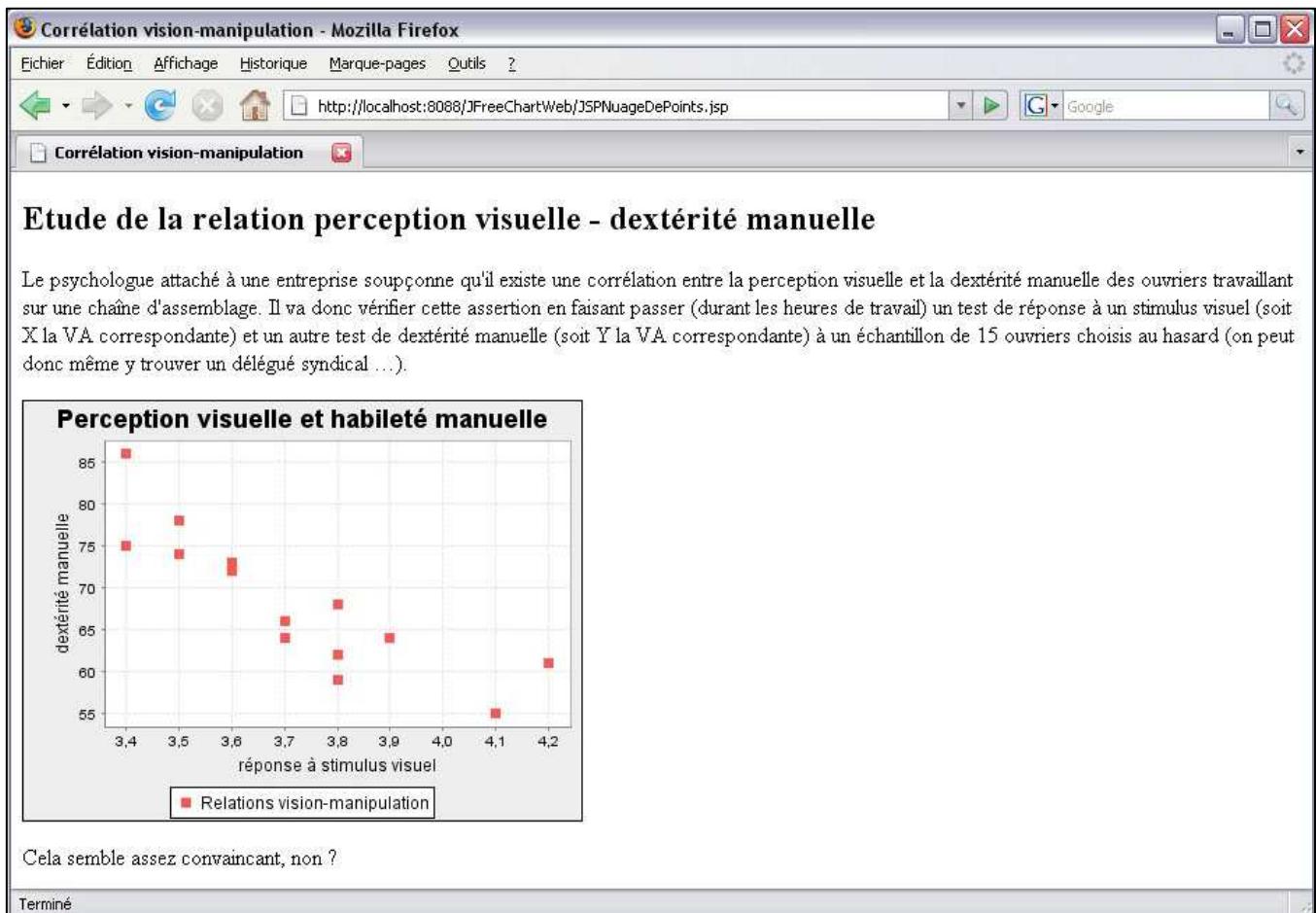
Evidemment, ce n'est pas très "décoré" comme résultat : on va donc plutôt considérer que le client charge un JSP dans lequel un simple tag demandera le graphique à la servlet :

JSPNuageDePoints.jsp

```
<%--  
 Document : JSPNuageDePoints  
 Author   : Vilvens  
--%>  
  
<%@page contentType="text/html" pageEncoding="UTF-8"%>  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
 "http://www.w3.org/TR/html4/loose.dtd">  
  
<html>  
  <head>  
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
    <title>Corrélation vision-manipulation</title>  
  </head>  
  <body>  
    <h2>Etude de la relation perception visuelle - dextérité manuelle</h2>  
    Le psychologue attaché à une entreprise soupçonne qu'il existe une corrélation entre la  
perception visuelle et la dextérité manuelle des ouvriers travaillant sur une chaîne d'assemblage. Il  
va donc vérifier cette assertion en faisant passer (durant les heures de travail) un test de réponse à  
un stimulus visuel (soit X la VA correspondante) et un autre test de dextérité manuelle (soit Y la  
VA correspondante) à un échantillon de 15 ouvriers choisis au hasard (on peut donc même y  
trouver un délégué syndical ...).  
    <p></p> <IMG SRC="ServletChart" BORDER=1 WIDTH=400 HEIGHT=300 />  
    <p><p></p>Cela semble assez convaincant, non ?  
  </body>  
</html>
```

Si la page index de l'application Web point sur le JSP, cela donnera, sans la moindre modification de la servlet :





9.4 Une servlet fournisseur d'étude statistique

Bien sûr, disposer du nuage de points est un peu limitatif pour le client : il souhaitera probablement connaître la valeurs des coefficients a, b et r de la statistique à deux dimensions des données considérées. Pour que le JSP invoqué par le client puisse afficher ces paramètres, nous allons utiliser un bean assez prévisible, puisque ses propriétés utilisables dans le JSP par les tags <jsp:getProperty>, seront précisément les paramètres demandés sous formes de chaînes de caractères – les autres propriétés sont les séries en x et y, le nombre d'observations et les valeurs numériques de ces paramètres :

LinearRegCorrBean.java

```
package Charts;

import java.beans.*;
import java.io.Serializable;
import java.text.NumberFormat;
import org.jfree.data.statistics.Statistics;

/**
 * @author Vilvens
 */
```

```

public class LinearRegCorrBean extends Object implements Serializable
{
    private Number[] x,y;
    private double[] ParamReg;
    private double CorrCoeff;
    private int nbObs;

    private PropertyChangeSupport propertySupport;

    public LinearRegCorrBean()
    {
        nbObs = 15;
        double couples[][] = { {3.8,68.0}, {3.6,72.0}, {3.4,86.0}, {3.5,78.0}, {3.9,64.0},
            {4.2,61.0}, {3.7,66.0}, {3.5,74.0}, {3.8,59.0}, {4.1,55.0},
            {3.7,64.0}, {3.6,73.0}, {3.8,62.0}, {3.4,75.0}, {3.5,78.0} };

        propertySupport = new PropertyChangeSupport(this);
        x = new Double[nbObs]; y = new Double[nbObs];
        for (int i=0; i<nbObs; i++)
        {
            x[i] = couples[i][0]; y[i] = couples[i][1];
        }
        ParamReg = this.getParamReg();
        CorrCoeff = this.getCorrCoeff();
    }

    public Number[] getX() { return x; }
    public void setX(Number[] x) { this.x = x; }
    public Number[] getY() { return y; }
    public void setY(Number[] y) { this.y = y; }

    public double[] getParamReg() { return Statistics.getLinearFit(x, y); }
    public String getAParamReg() { return viewDec(ParamReg[1],2); }
    public String getBParamReg() { return viewDec(ParamReg[0],2); }

    public double getCorrCoeff() { return Statistics.getCorrelation(x,y); }
    public String getRCorrCoeff() { return viewDec(CorrCoeff,2); }

    public int getNbObs() { return nbObs; }
    public void setNbObs(int nbObs) { this.nbObs = nbObs; }

    public String viewDec (double x, int nd)
    {
        NumberFormat nf = NumberFormat.getInstance();
        System.out.println("Classe instanciée = " + nf.getClass().getName());
        nf.setMinimumFractionDigits(nd); nf.setMaximumFractionDigits(nd);
        String str = nf.format(x);
        return str;
    }
}

```

Bien sûr, un tel bean n'est pas un chef d'œuvre ;-) : il faudrait pouvoir le configurer avec des données dynamiques fournies par une base de données – autrement dit, il faudrait un MVC orthodoxe. To do ! Pour l'heure, contentons-nous d'exploiter ce bean dans notre JSP :

JSPNusageDePoints.jsp

```
<%--  
Document : JSPNusageDePoints  
Created on : 04-sept.-2008, 8:48:41  
Author : Vilvens  
--%>  
  
<%@page contentType="text/html" pageEncoding="UTF-8"%>  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<jsp:useBean id="Statistique2D" scope="page" class="Charts.LinearRegCorrBean" />  
<html>  
  <head>  
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
    <title>Corrélation vision-manipulation</title>  
  </head>  
  <body>  
    <h2>Etude de la relation perception visuelle - dextérité manuelle</h2>  
    Le psychologue attaché à une entreprise soupçonne qu'il existe une corrélation entre la  
perception visuelle et la dextérité manuelle des ouvriers travaillant sur une chaîne  
d'assemblage. Il va donc vérifier cette assertion en faisant passer (durant les heures de travail)  
un test de réponse à un stimulus visuel (soit X la VA correspondante) et un autre test de  
dextérité manuelle (soit Y la VA correspondante) à un échantillon de 15 ouvriers choisis au  
hasard (on peut donc même y trouver un délégué syndical ...).  
    <p></p> <IMG SRC="ServletChart" BORDER=1 WIDTH=320 HEIGHT=240 />  
    <p><p></p>Cela semble assez convaincant, non ?  
    Paramètre statistiques :  
    <p><h3>Droite de régression linéaire :</h3></p>  
    a = <jsp:getProperty name="Statistique2D" property="AParamReg" />  
    <p></p>b = <jsp:getProperty name="Statistique2D" property="BParamReg" />  
    <p><h3>Coefficient de corrélation :</h3></p>  
    r = <jsp:getProperty name="Statistique2D" property="RCorrCoeff" />  
  </body>  
</html>
```

Le résultat est sans surprise :

Corrélation vision-manipulation - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

AbstractIntervalXYDataset (JFreeCha... Corrélation vision-manipulation

Etude de la relation perception visuelle - dextérité manuelle

Le psychologue attaché à une entreprise soupçonne qu'il existe une corrélation entre la perception visuelle et la dextérité manuelle des ouvriers travaillant sur une chaîne d'assemblage. Il va donc vérifier cette assertion en faisant passer (durant les heures de travail) un test de réponse à un stimulus visuel (soit X la VA correspondante) et un autre test de dextérité manuelle (soit Y la VA correspondante) à un échantillon de 15 ouvriers choisis au hasard (on peut donc même y trouver un délégué syndical ...).

Perception visuelle et habileté manuelle

Réponse à stimulus visuel (X)	Dextérité manuelle (Y)
3.4	82
3.5	75
3.5	78
3.6	72
3.7	68
3.7	65
3.8	62
3.8	60
3.9	65
4.0	55
4.1	58

■ Relations vision-manipulation

Cela semble assez convaincant, non ? Paramètre statistiques :

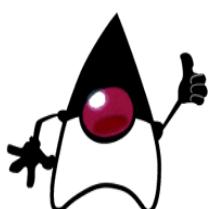
Droite de régression linéaire :

$a = -30,75$

$b = 182,78$

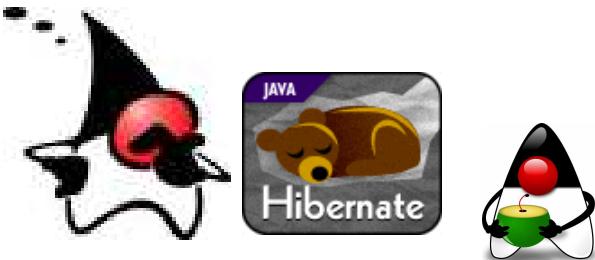
Coefficient de corrélation :

$r = -0,87$



Pour clôturer cette première partie sur les données, disons quelques mots d'une implémentation de JPA : serait-elle destinée aux ours pendant les mois d'hiver ;-)

XXXIV. Une petite introduction à Hibernate



La vie persiste au sein même de la destruction.

(Gandhi, La jeune Inde)

1. Le mapping ORM

Ce qui a conduit à construire un framework comme Hibernate (créé par JBoss Entreprise Middleware Systems – JEMS) est le constat qu'il n'est pas si aisés de développer une application basée à la fois sur un langage orienté objets et des bases des données relationnelles. On considère même qu'une fraction non négligeable du développement est perdue en termes de résolution de problèmes de transformation d'un paradigme OO en un paradigme SQL ou vice versa. Il est dès lors compréhensible que des plates-formes visant à harmoniser ce mariage de technologies presque "contre-nature" aient vu le jour. On parle encore de framework **ORM** (Object/Relational Mapping).



D'autre part, la persistance des données est l'un des problèmes récurrents de la Programmation Orientée Objets : comment disposer d'un mécanisme souple et simple permettant de conserver ses objets sans devoir parler explicitement de fichiers (comme dans le processus de sérialisation classique) ? Là aussi, des frameworks se sont proposés de prendre cette question en charge.

Hibernate est donc l'un de ces frameworks. *D'autres cours ont déjà permis aux lecteurs d'aborder l'ORM*. Mais ce qui fait l'intérêt de ce framework particulier, implémentation améliorée de JPA (JSR 220 – voir Java II), est qu'il s'intègre dans les plates-formes Java classiques (J2SE et J2EE) en n'imposant aucune implémentation d'interface ou dérivation de classe de base. Le mapping se fait au moyen d'APIs qui assurent une "persistance transparente" aux objets que l'application manipule : il leur est juste demandé de disposer d'un constructeur par défaut. À l'inverse, le mapping ORM à partir des bases de données se fait avec des POJOs (Plain Old Java Objects) (il leur est juste demandé de posséder un constructeur par défaut) ou encore des Java Beans.

L'accès aux données peut se faire par du SQL classique, mais aussi avec du **HQL** (**Hibernate Query Language**) ou même des critères de sélection exprimés plus généralement : c'est Hibernate qui générera les commandes SQL correctes pour le SGBD visé (celui-ci étant sensé comprendre le SQL).

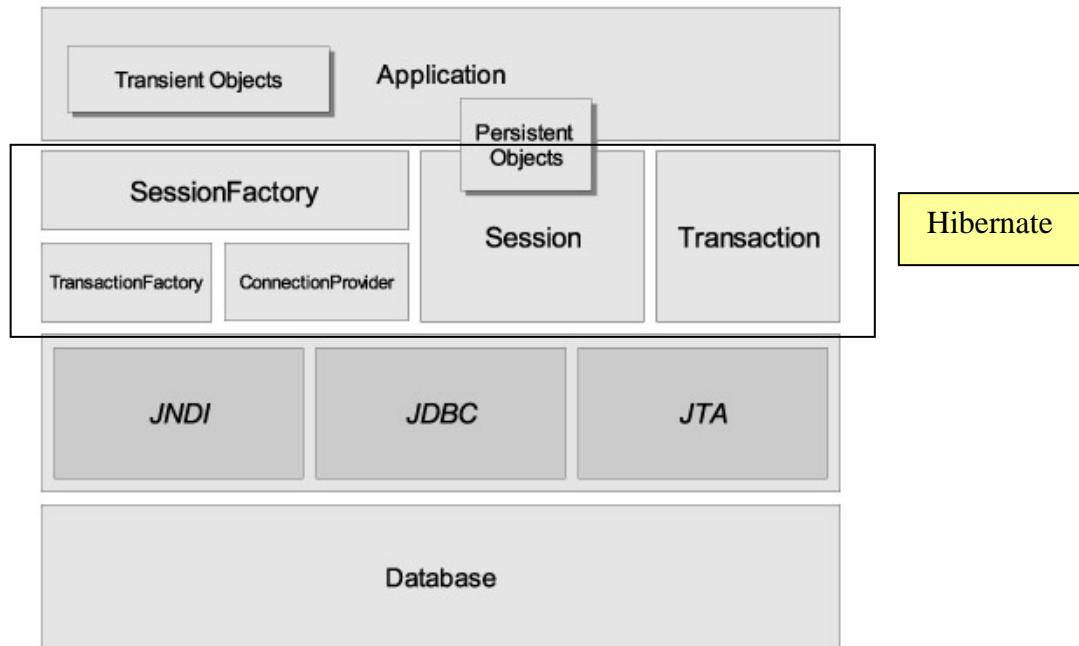
Mais l'intérêt d'Hibernate est aussi une optimisation des transferts effectifs des données, avec un système sophistiqué d'accès et de mise en cache de celles-ci.

En pratique, bien que les JDK 1.2 ou 1.3 puissent être suffisants pour travailler avec Hibernate, il est recommandé de travailler avec le JDK 1.5 au minimum. Il peut accéder aux données gérées par la plupart des SGBDs relationnels : Oracle, DB2, Sybase, MS SQL Server, PostgreSQL, MySQL, HypersonicSQL, Mckoi SQL, SAP DB, Interbase, Pointbase, Progress, FrontBase, Ingres, Informix, Firebird. Quant aux environnements et outils Java, il peut supporter JDBC 3.0, JNDI 1.21. et EJB 3.0

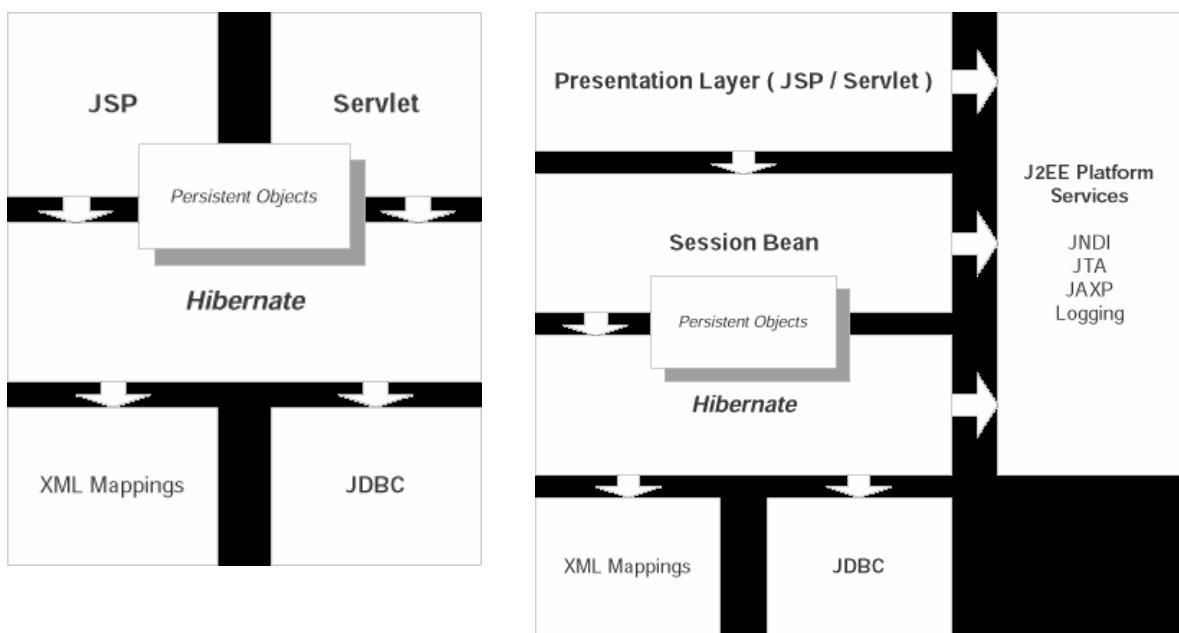
2. L'architecture d'Hibernate

2.1 Les composants de la couche Hibernate

Hibernate se présente comme une couche additionnelle entre l'application et les outils d'accès aux bases de données comme JDBC ou JNDI. Le contexte peut être celui de J2SE :



ou celui de J2EE :



Les trois éléments essentiels de Hibernate sont :

- ♦ l'interface **SessionFactory** (du package org.hibernate) : un objet l'implémentant devra fournir une session Hibernate (instance de **Session**) qui est, en fait, la matérialisation d'un cache contenant les mappings ORM vers la base de données visée;

- ◆ l'interface **Session** : un tel objet est attaché au thread en cours d'exécution et représente en fait l'intermédiaire entre l'application J2SE ou J2EE et les objets persistants associés à la base par l'intermédiaire d'Hibernate. Ces objets sont maintenus dans un cache. Bien clairement, cet objet session permet de gérer les transactions.
- ◆ l'interface **Transaction** : un tel objet matérialise bien sûr la transaction au sens SGBD du terme, mais il l'abstrait également car elle peut implémentée comme une transaction JDBC, JTA ou même CORBA.

2.2 Les états des instances

Grâce à des opérations que nous allons décrire plus précisément plus bas dans un exemple pratique sous Netbeans, nous allons donc construire des objets instances de classes "persistantes". On peut d'ores et déjà savoir que ces objets peuvent se trouver dans trois états distincts :

- ◆ transient ("éphémère" en français) : objet non associé à un contexte de persistance (c'est-à-dire à une session Hibernate) et qui n'est pas associé à une clé primaire;
- ◆ persistant : il est associé à une session et possède une valeur de clé primaire qui l'identifie de manière équivalente à son adresse mémoire (parler du tuple dans la base ou de l'objet en mémoire est donc strictement équivalent);
- ◆ détaché : la session initialement associée a été fermée mais l'objet possède toujours une valeur de clé primaire; cependant, l'objet mémoire et le tuple peuvent être à présent différents.

3. En pratique avec Netbeans

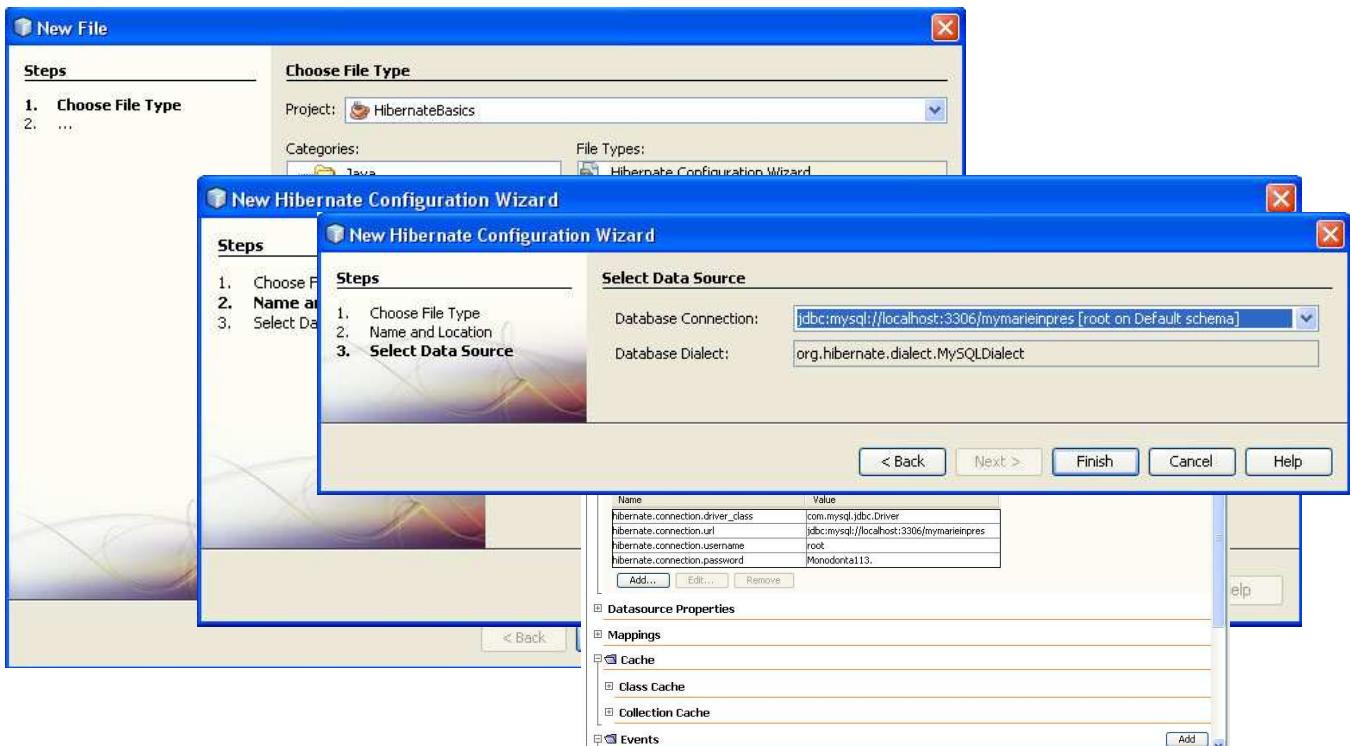
Pour développer avec Hibernate, il suffit de monter ses jars (téléchargés depuis <https://www.hibernate.org/6.html>) :

Pour disposer intégralement des outils complets de développement conjoint NetBeans-Hibernate, il convient cependant d'utiliser NetBeans 6.8 minimum.



4. Premier pas : la création d'un fichier de configuration

Supposons disposer d'une base de données MySQL (*mymarieinpres*) dont nous voulons mapper les données (il s'agira de la table *stocks* mais c'est sans importance pour l'instant) dans un schéma ORM. Nous aurons tout d'abord besoin d'un fichier de configuration (*hibernate.cfg.xml*) qui contiendra les informations nécessaires pour référencer cette base de données (nom de la base, type de SGBD) et y accéder (driver JDBC sous-jacent, utilisateur et mot de passe). Sous Netbeans, ce fichier peut être créé facilement puisqu'un simple clic droit sur le projet suivi de New → Hibernate → Hibernate Configuration Wizard donne

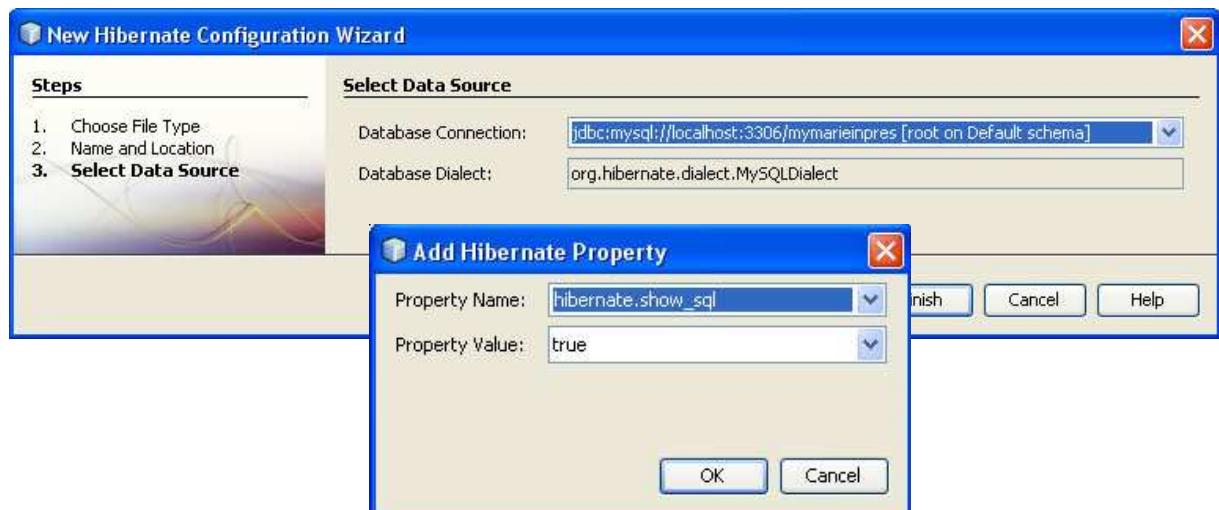


On a choisi une base de données à laquelle on s'était déjà connecté par Netbeans :

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://.hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/mymarieinpres</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">#####</property>
    </session-factory>
</hibernate-configuration>
```

On peut ajouter une property dans les "optional properties" – "configuration properties" – par exemple, si l'objectif est de sauver les commandes SQL générées :



Après l'ajout, on a en plus :

```
<property name="hibernate.show_sql">true</property>
```

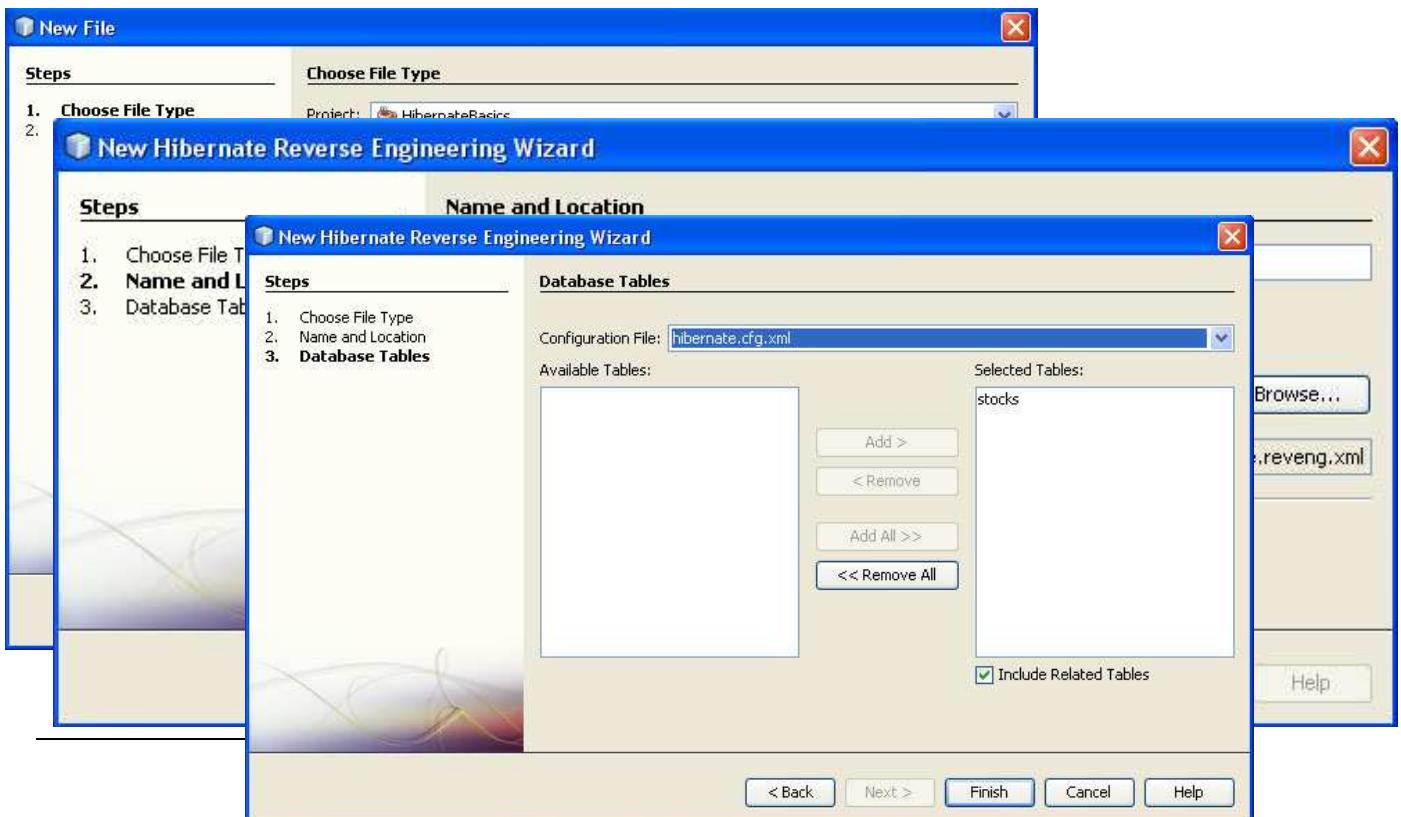
5. Tables et POJOs

Tout objet mappé à partir de la base de données (les données originales sont les tables) ou vers une base de données (les données originales sont dans des objets mémoires) doit être décrit

- ◆ soit au moyen d'un fichier xml portant le même nom que la classe;
- ◆ soit au moyen d'annotations.

5.1 Crédation d'une classe à partir d'une table.

Nous pouvons, sous Netbeans, utiliser le reverse engeneering. Pour cela, un clic droit sur le projet :



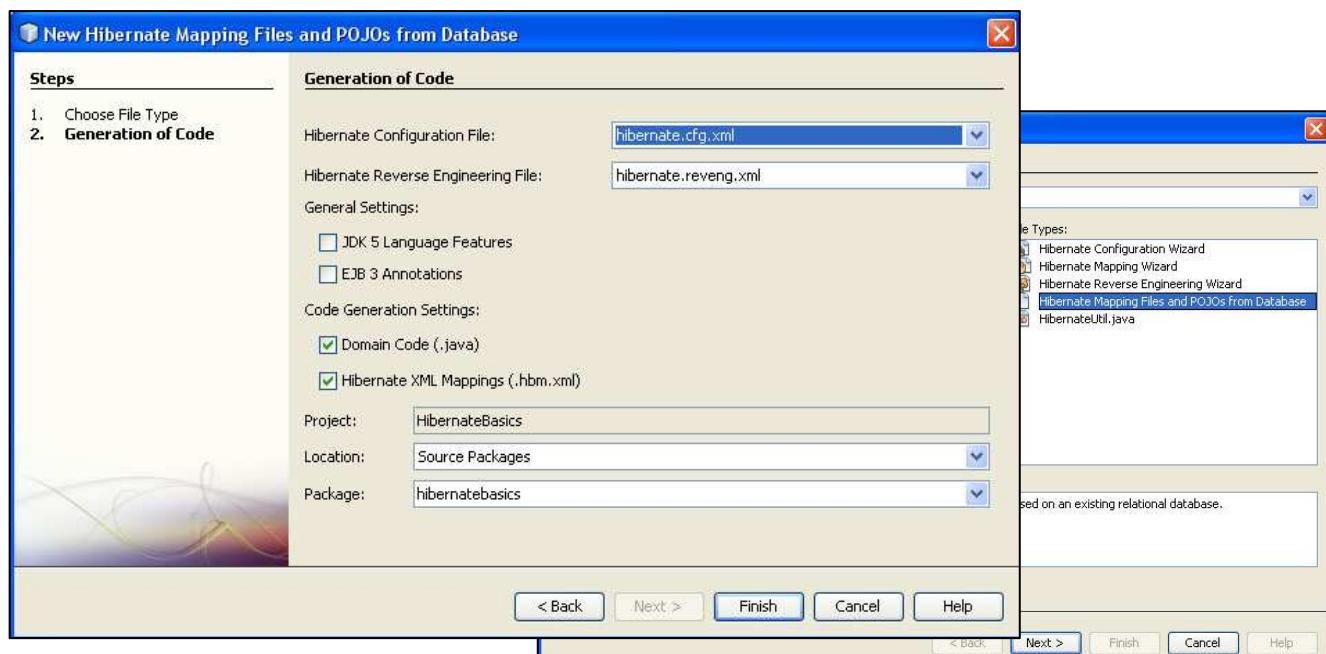
ce qui produira :

hibernate.reveng.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate Reverse
Engineering DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-reverse-engineering-
3.0.dtd">
<hibernate-reverse-engineering>
  <schema-selection match-catalog="mymarieinpres"/>
  <table-filter match-name="stocks"/>
</hibernate-reverse-engineering>
```

5.2 La génération des objets mappés

Ensuite, on peut demander la génération des POJOs sur base de des deux fichiers config et revenge :



Le résultat :

Stocks.java

```
package hibernatebasics;
// Generated 09-mars-2010 9:32:33 by Hibernate Tools 3.2.1.GA

/**
 * Stocks generated by hbm2java
 */
public class Stocks implements java.io.Serializable
{
    private String codeSto;
    private Integer x;
    private Integer y;
    private Double quantite;
```

```

private String categorie;

public Stocks() { }

public Stocks(String codeSto)
{
    this.codeSto = codeSto;
}

public Stocks(String codeSto, Integer x, Integer y, Double quantite, String categorie)
{
    this.codeSto = codeSto;
    this.x = x; this.y = y; this.quantite = quantite; this.categorie = categorie;
}

public String getCodeSto() { return this.codeSto; }
public void setCodeSto(String codeSto) { this.codeSto = codeSto; }
public Integer getX() { return this.x; }
public void setX(Integer x) { this.x = x; }
public Integer getY() { return this.y; }
public void setY(Integer y) { this.y = y; }
public Double getQuantite() { return this.quantite; }
public void setQuantite(Double quantite) { this.quantite = quantite; }
public String getCategorie() { return this.categorie; }
public void setCategorie(String categorie) { this.categorie = categorie; }
}

```

et un fichier rappelant ceux destinés à la technique de sérialisation/désérialisation XML : (voir Java III, chapitre XVIII, paragraphe 12 : "La sérialisation XML") :

Stocks.hbm.xml

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 09-mars-2010 9:32:35 by Hibernate Tools 3.2.1.GA -->
<hibernate-mapping>
    <class name="hibernatebasics.Stocks" table="stocks" catalog="mymarieinpres">
        <id name="codeSto" type="string">
            <column name="codeSto" length="10" />
            <generator class="assigned" />
        </id>
        <property name="x" type="java.lang.Integer">
            <column name="x" />
        </property>
        <property name="y" type="java.lang.Integer">
            <column name="y" />
        </property>
        <property name="quantite" type="java.lang.Double">
            <column name="quantite" precision="22" scale="0" />
        </property>
    </class>
</hibernate-mapping>

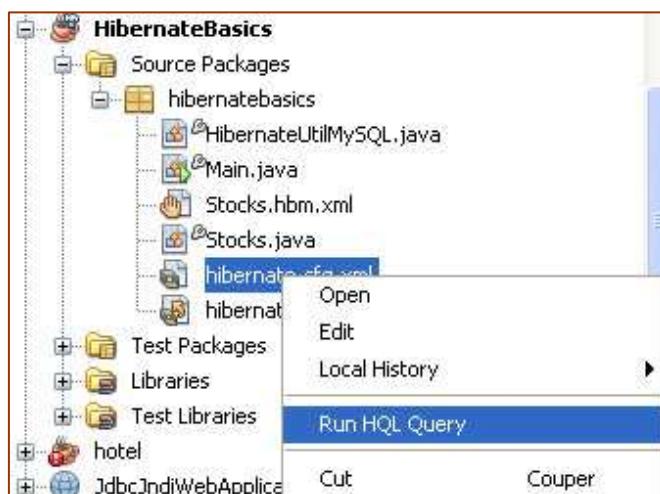
```

```
<property name="categorie" type="string">
    <column name="categorie" length="20" />
</property>
</class>
</hibernate-mapping>
```

En fait, la DTD de ce type de fichier XML est très étendue. Ce fichier DTD est inclus dans **hibernate-core.jar** ainsi que dans le répertoire src de la distribution Hibernate. Bien sûr, l'interprétation de ce fichier est assez simple : chaque colonne de la table correspond manifestement à une property (qui coïncidera manifestement avec une variable membre ici). Il convient cependant de remarquer les balises `<id>` et `<generator>` qui sont associés à la colonne "codeSto" servant de clé primaire à la table stocks : l'objectif est bien clairement pour Hibernate de posséder, d'une manière ou d'une autre, **un identifiant unique pour chaque objet persistant**. Ici, cet id est la clé primaire, ce que précise le tag `<generator>` avec son attribut class prenant la valeur "assigned"; ce tag définit donc la stratégie de génération de l'identifiant – ce pourrait être par exemple "native" pour que le framework génère lui-même un identifiant unique (en utilisant l'une des stratégies "identity", "sequence" ou "hilo" que l'on trouve reprise dans certains SGBDs).

5.3 Utilisation directe du mapping

Il est à présent possible d'exploiter la structure mise en place de manière interactive : un clic : un droit sur le fichier cfg :



fait apparaître l'éditeur HQL : on peut y taper "from Stocks" et solliciter l'icône "Run HQL Query" :

X	Y	CodeSto	Quantite	Categorie
5	1	m2123-52	140.0	oignons
5	2	m2123-58	140.0	oignons
3	1	m2623-87	780.0	oignons
2	1	m4143-66	65.0	poireaux
7	1	m6969-88	1852.0	poireaux

La sélection de l'onglet SQL donne le texte de la requête SQL :

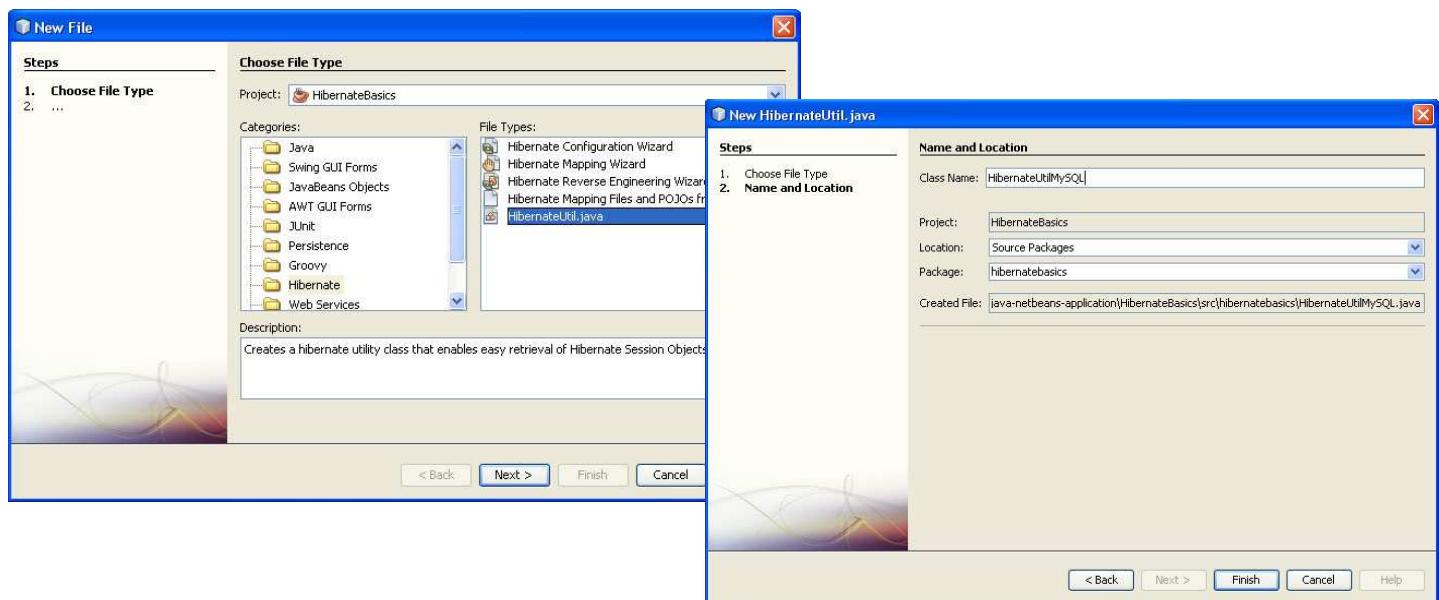
```
...ava hibernate.reveng.xml Stocks.java Stocks.hbm.xml HQL Query0
Session: hibernate.cfg
from Stocks

Result SQL Set Max. Row Count: 100
select stocks0_.codeSto as col_0_0_ from mymarieinpres.stocks stocks0_
```

Bien sûr, nous souhaiterons plutôt utiliser le mapping tuples-objets dans un applicatif programmé. Pour cela, nous aurons besoin d'un objet session.

5.4 L'objet session et les transactions

Netbeans permet de créer une classe utilitaire qui va mettre en place le "décor" :



Netbeans a généré une classe, nommée (sur nos indications) `HibernateUtilMySQL`. Son rôle est d'initialiser (dans un bloc statique, selon la technique du singleton) une factory (instance de `SessionFactory`) qui devra fournir en temps utiles une session Hibernate (instance de `Session`) attachée au thread en cours d'exécution. Cette session s'obtient au moyen de la méthode

```
public Session getCurrentSession() throws HibernateException
```

Bien clairement, cet objet session permet de gérer les transactions, celles-ci étant démarrées par

public Transaction **beginTransaction()** throws HibernateException

Un commit ou un rollback sont programmés par récupération de la transaction par

public Transaction **getTransaction()** throws HibernateException

suivi de l'appel de l'une des méthodes de Transaction :

public void **commit()** throws HibernateException
public void **rollback()** throws HibernateException

qui termine la transaction. Bien sûr, fermer la session clôture également la transaction et détache la session du thread. Un nouvel appel à la méthode `getCurrentSession()` fournit un nouvel objet session et débute une nouvelle transaction.

HibernateUtilMySQL.java

```
package hibernatebasics;

import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;

/**
 * Hibernate Utility class with a convenient method to get Session Factory object.
 *
 * @author Utilisateur
 */
public class HibernateUtilMySQL
{
    private static final SessionFactory sessionFactory;

    static
    {
        try
        {
            // Create the SessionFactory from standard (hibernate.cfg.xml) config file.
            sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();
        }
        catch (Throwable ex)
        {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() { return sessionFactory; }
}
```

A remarquer qu'une SessionFactory correspond à une base de données bien précise. Par conséquent, une application qui accède à plusieurs bases a besoin d'un nombre équivalent de factories distinctes.

C'est évidemment au sein de l'objet session que les requêtes d'accès aux données sont invoquées ...

5.5 Utilisation de l'objet mappé

Nous pouvons à présent créer une classe de test :

Application.java

```
package hibernatebasics;

import java.util.List;
import java.util.Vector;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;

/**
 *
 * @author Claude
 */
public class Application
{
    public static void main(String[] args)
    {
        try
        {
            Session session = HibernateUtilMySql.getSessionFactory().openSession();
            session.beginTransaction();
            Query q = (Query)session.createQuery("from Stocks");
            List<Stocks> resultList = (List<Stocks>) q.list();
            displayResult(session,resultList);
            session.getTransaction().commit();
        }
        catch (HibernateException he)
        {
            he.printStackTrace();
        }
    }

    private static void displayResult (Session s,List<Stocks> resultList)
    {
        Vector tableData = new Vector();

        for(Stocks o : resultList)
        {
            Stocks article = (Stocks)o;
            Vector<Object> oneRow = new Vector<Object>();
            oneRow.add(article.getId());
            oneRow.add(article.getName());
            oneRow.add(article.getQuantity());
            oneRow.add(article.getPrice());
            tableData.add(oneRow);
        }
    }
}
```

```
    oneRow.add(article.getCodeSto());
    oneRow.add(article.getCategorie());
    oneRow.add(article.getQuantite());
    oneRow.add(article.getX());
    oneRow.add(article.getY());
    tableData.add(oneRow);

}
System.out.println(tableData.toString());
}

}
```

On peut apprécier le niveau d'abstraction atteint puisque :

- ♦ la requête est formulée en HQL et est envoyée au SGBD au moyen de la méthode de l'objet Session :

```
public Query createQuery(String queryString)
```

- ♦ l'objet implémentant l'interface **Query** ainsi récupéré possède la méthode

```
public List list()throws HibernateException
```

Résultat :

```
run:
24-mars-2010 11:08:46 org.hibernate.cfg.annotations.Version <clinit>
INFO: Hibernate Annotations 3.3.1.GA
...
24-mars-2010 11:08:46 org.hibernate.cfg.Configuration addResource
INFO: Reading mappings from resource : hibernatebasics/Stocks.hbm.xml
...
24-mars-2010 11:08:46 org.hibernate.connection.DriverManagerConnectionProvider
configure
INFO: Using Hibernate built-in connection pool (not for production use!)
24-mars-2010 11:08:46 org.hibernate.connection.DriverManagerConnectionProvider
configure
INFO: Hibernate connection pool size: 20
24-mars-2010 11:08:46 org.hibernate.connection.DriverManagerConnectionProvider
configure
INFO: autocommit mode: false
24-mars-2010 11:08:46 org.hibernate.connection.DriverManagerConnectionProvider
configure
INFO: using driver: com.mysql.jdbc.Driver at URL:
jdbc:mysql://localhost:3306/mymarieinpres
24-mars-2010 11:08:46 org.hibernate.connection.DriverManagerConnectionProvider
configure
INFO: connection properties: {user=root, password=****}
```

```
....  
24-mars-2010 11:08:46 org.hibernate.cfg.SettingsFactory buildSettings  
INFO: JDBC driver: MySQL-AB JDBC Driver, version: mysql-connector-java-5.1.6 ( Revision: ${svn.Revision} )  
...  
24-mars-2010 11:08:46 org.hibernate.transaction.TransactionManagerLookupFactory  
getTransactionManagerLookup  
INFO: No TransactionManagerLookup configured (in JTA environment, use of read-write or  
...  
Hibernate: select stocks0_.codeSto as codeSto0_, stocks0_.x as x0_, stocks0_.y as y0_,  
stocks0_.quantite as quantite0_, stocks0_.categorie as categorie0_ from mymarieinpres.stocks  
stocks0_  
[[m2123-52 , oignons, 140.0, 5, 1], [m2123-58 , oignons, 140.0, 5, 2], [m2623-87 , oignons,  
780.0, 3, 1], [m4143-66 , poireaux, 65.0, 2, 1], [m6969-88 , poireaux, 1852.0, 7, 1]]
```

Bien sûr, il est possible de faire mieux, mais considérons à présent le problème inverse.

6. La persistance : création d'une table à partir d'une classe

Considérons que nous allons

- ◆ créer une classe vente : elle sera sensée matérialiser la vente de produits de notre stock;
- ◆ instancier cette classe et configurer l'objet résultant;
- ◆ **rendre cet objet persistant** dans la base de données au moyen d'un mapping Hibernate.

La classe Vente est un POJO à qui il est simplement demandé de posséder un constructeur par défaut :

Vente.java

```
package hibernatebasics;  
  
import java.util.Date;  
  
public class Vente  
{  
    private String id;  
    private String acheteur;  
    private Date date;  
    private Double montant;  
  
    public Vente() {}  
    public Vente(String i, String a, Date d, double m)  
    {  
        id = i; acheteur=a; date=d; montant=m;  
    }  
  
    public String getId() { return id; }  
    public void setId(String id) { this.id = id; }
```

```

public String getAcheteur() { return acheteur; }
public void setAcheteur(String acheteur) { this.acheteur = acheteur; }
public Date getDate() { return date; }
public void setDate(Date date) { this.date = date; }
public Double getMontant() { return montant; }
public void setMontant(Double montant) { this.montant = montant; }
}

```

Nous définissons aussi un fichier de mapping :

Vente.hbm.xml

```

<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://.hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 24-mars-2010 10:56:01 by Hibernate Tools 3.2.1.GA -->
<hibernate-mapping package="hibernatebasics">
    <class name="Vente" table="ventes" catalog="mymarieimpres">
        <id name="id" >
            <column name="numVente" />
            <generator class="assigned" />
        </id>
        <property name="acheteur"/>
        <property name="date" type="timestamp">
            <column name="dateVente" />
        </property>
        <property name="montant"/>
    </class>
</hibernate-mapping>

```

Par défaut, le nom de la propriété devient celui de la colonne de la table créée, ce qui nous arrange bien pour la propriété "acheteur", mais beaucoup moins pour la propriété "date" (d'où la précision du nom de colonne).

Concernant les types, Hibernate tentera de déterminer la bonne conversion et le type de mappage lui-même si l'attribut type n'est pas présent dans le mappage. Dans certains cas, cette détection automatique (utilisant la réflexion sur la classe Java) pourrait ne pas donner le type attendu. C'est bien sûr le cas avec la propriété date. Hibernate ne peut pas savoir si la propriété "mapera" une colonne SQL de type date, timestamp ou time. Nous déclarons que nous voulons conserver des informations avec une date complète et l'heure en mappant la propriété avec un convertisseur timestamp.

Nous rectifions le fichier de configuration :

```
<mapping resource="hibernatebasics/Vente.hbm.xml"/>
```

et dans l'application, nous ajoutons :

Application.java (ajout)

...

```

        session = HibernateUtilMySql.getSessionFactory().getCurrentSession();
        session.beginTransaction();

```

```
Vente v1 = new Vente ("v20100430-001", "Lapoutre S.A.", new Date(), 1523.41);
session.save(v1);
Vente v2 = new Vente ("v20100430-002", "Les Deux veaux SPRL", new Date(),
    745.41);
session.save(v2);
session.getTransaction().commit();
...
```

Mais l'exécution de ce code pose problème :

```
GRAVE: Table 'mymarieinpres.ventes' doesn't exist
25-avr.-2010 17:55:06 org.hibernate.event.def.AbstractFlushingEventListener
performExecutions
GRAVE: Could not synchronize database state with session
```

Faudrait-il donc que la table ventes existe déjà ? Le fait est que, si on crée la table en question (ici, avec Toad) :

Field *	Type *	Collation	Null *	Key *	Default
numVente	varchar(20)	latin1_swedish_ci	NO	PRI	
acheteur	varchar(40)	latin1_swedish_ci	YES		{null}
dateVente	timestamp	{null}	NO		CURRENT_TIMESTAMP
montant	double	{null}	YES		{null}

alors on obtient bien :

```
Hibernate: insert into mymarieinpres.ventes (acheteur, dateVente, montant, numVente) values
(?, ?, ?, ?)
Hibernate: insert into mymarieinpres.ventes (acheteur, dateVente, montant, numVente) values
(?, ?, ?, ?)
```

et effectivement :

numVente *	acheteur	dateVente *	montant
v20100430-001	Lapoutre S.A.	25/04/2010 18:15:54	1523,41
v20100430-002	Les Deux veaux SPRL	25/04/2010 18:15:54	745,41

Cependant, il serait tout de même souhaitable que la table soit créée automatiquement. Il faut pour cela lier la session au thread en écrivant dans le fichier de configuration :

```
...  
<property name="hibernate.current_session_context_class">thread</property>
```

en sachant que la propriété

hibernate.transaction.factory_class

doit avoir pour valeur

`org.hibernate.transaction.JDBCTransactionFactory`

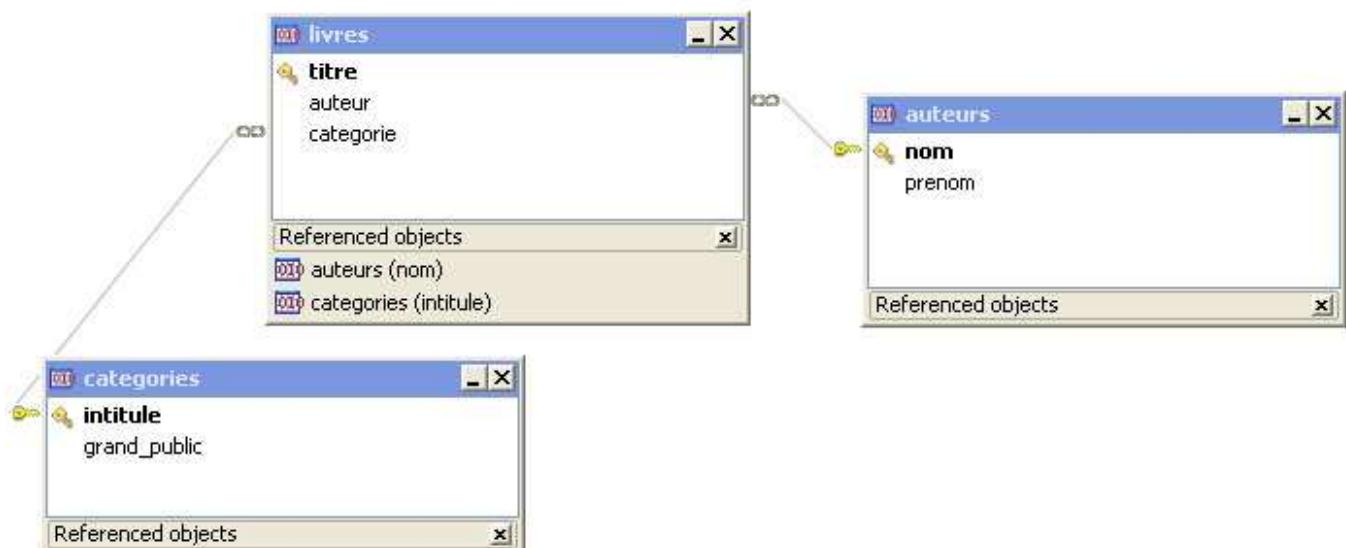
en l'absence de l'utilisation d'un environnement JTA. On peut alors constater que tout se passe bien.

Bien sûr, les contraintes sont vérifiées – si on exécute deux fois l'application ci-dessus, on obtiendra :

```
Hibernate: insert into mymarieinpres.ventes (acheteur, dateVente, montant, numVente) values  
        (?, ?, ?, ?)  
Hibernate: insert into mymarieinpres.ventes (acheteur, dateVente, montant, numVente) values  
        (?, ?, ?, ?)  
11-mai-2010 8:21:36 org.hibernate.util.JDBCExceptionReporter logExceptions  
ATTENTION: SQL Error: 1062, SQLState: 23000  
11-mai-2010 8:21:36 org.hibernate.util.JDBCExceptionReporter logExceptions  
GRAVE: Duplicate entry 'v20100430-002' for key 1  
11-mai-2010 8:21:36 org.hibernate.event.def.AbstractFlushingEventListener  
performExecutions  
GRAVE: Could not synchronize database state with session  
org.hibernate.exception.ConstraintViolationException: Could not execute JDBC batch update  
    at org.hibernate.exception.SQLStateConverter.convert(SQLStateConverter.java:71)  
...  
Caused by: java.sql.BatchUpdateException: Duplicate entry 'v20100430-002' for key 1  
...
```

7. Les jointures

Les fichiers de mapping d'Hibernate sont capables de prendre en charge les clés étrangères, les associations ainsi définies se matérialisant dans les objets mappés par des relations d'agrégation. Pour faire court, considérons l'exemple simple de livres écrit par un auteur et appartenant à une catégorie :



Le fichier de configuration d'Hibernate est simplement :

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://.hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/librairie</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">XXX007696969</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.current_session_context_class">thread</property>
    <mapping resource="hibernatejointure/Livres.hbm.xml"/>
    <mapping resource="hibernatejointure/Auteurs.hbm.xml"/>
    <mapping resource="hibernatejointure/Categories.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Les trois classes mappées et les fichiers de mapping correspondant sont :

a) les auteurs

Auteurs.java

```
package hibernatejointure;
// Generated 07-sept.-2010 10:03:44 by Hibernate Tools 3.2.1.GA

import java.util.HashSet;
import java.util.Set;

/**
 * Auteurs generated by hbm2java
 */
public class Auteurs implements java.io.Serializable
{
    private String nom;
    private String prenom;
    private Set livres = new HashSet(0);

    public Auteurs() { }

    public Auteurs(String nom) { this.nom = nom; }
    public Auteurs(String nom, String prenom, Set livreses)
    {
        this.nom = nom;
        this.prenom = prenom;
        this.livres = livreses;
    }

    public String getNom() { return this.nom; }
    public void setNom(String nom) { this.nom = nom; }
    public String getPrenom() { return this.prenom; }
    public void setPrenom(String prenom) { this.prenom = prenom; }
    public Set getLivres() { return this.livres; }
    public void setLivres(Set livres) { this.livres = livres; }
}
```

Auteurs.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 07-sept.-2010 10:03:46 by Hibernate Tools 3.2.1.GA --&gt;
&lt;hibernate-mapping&gt;
&lt;class catalog="<b>librairie" lazy="true" name="hibernatejointure.Auteurs" table="auteurs">
    <id name="nom" type="string">
        <column length="40" name="nom" />
        <generator class="assigned" />
    </id>
    <property name="prenom" type="string">
        <column length="20" name="prenom" />
```

```
</property>
<set inverse="true" lazy="true" name="livres">
  <key>
    <column length="40" name="auteur"/>
  </key>
  <one-to-many class="hibernatejointure.Livres"/>
</set>
</class>
</hibernate-mapping>
```

b) les catégories

Categories.java

```
package hibernatejointure;
// Generated 07-sept.-2010 10:03:44 by Hibernate Tools 3.2.1.GA

import java.util.HashSet;
import java.util.Set;

/**
 * Categories generated by hbm2java
 */
public class Categories implements java.io.Serializable
{
  private String intitule;
  private Byte grandPublic;
  private Set livreses = new HashSet(0);

  public Categories() { }
  public Categories(String intitule) { this.intitule = intitule; }
  public Categories(String intitule, Byte grandPublic, Set livreses)
  {
    this.intitule = intitule;
    this.grandPublic = grandPublic;
    this.livreses = livreses;
  }

  public String getIntitule() { return this.intitule; }
  public void setIntitule(String intitule) { this.intitule = intitule; }
  public Byte getGrandPublic() { return this.grandPublic; }
  public void setGrandPublic(Byte grandPublic) { this.grandPublic = grandPublic; }
  public Set getLivreses() { return this.livreses; }
  public void setLivreses(Set livreses) { this.livreses = livreses; }
}
```

Categories.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

```
<!-- Generated 07-sept.-2010 10:03:46 by Hibernate Tools 3.2.1.GA -->
<hibernate-mapping>
  <class catalog="librairie" lazy="true" name="hibernatejointure.Categories"
        table="categories">
    <id name="intitule" type="string">
      <column length="30" name="intitule" />
      <generator class="assigned" />
    </id>
    <property name="grandPublic" type="java.lang.Byte">
      <column name="grand_public" />
    </property>
    <set inverse="true" lazy="true" name="livreses">
      <key>
        <column length="30" name="categorie" />
      </key>
      <one-to-many class="hibernatejointure.Livres" />
    </set>
  </class>
</hibernate-mapping>
```

c) les livres

Livres.java

```
package hibernatejointure;
// Generated 07-sept.-2010 10:03:44 by Hibernate Tools 3.2.1.GA

/**
 * Livres generated by hbm2java
 */
public class Livres implements java.io.Serializable
{
    private String titre;
    private Categories categories;
    private Auteurs auteurs;

    public Livres() { }
    public Livres(String titre) { this.titre = titre; }
    public Livres(String titre, Categories categories, Auteurs auteurs)
    {
        this.titre = titre;
        this.categories = categories;
        this.auteurs = auteurs;
    }

    public String getTitre() { return this.titre; }
    public void setTitre(String titre) { this.titre = titre; }
    public Categories getCategories() { return this.categories; }
    public void setCategories(Categories categories) { this.categories = categories; }
    public Auteurs getAuteurs() { return this.auteurs; }
```

```
public void setAuteurs(Auteurs auteurs) { this.auteurs = auteurs; } }
```

Livres.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 07-sept.-2010 10:03:46 by Hibernate Tools 3.2.1.GA --&gt;
&lt;hibernate-mapping&gt;
&lt;class catalog="librairie" lazy="true" name="hibernatejointure.Livres" table="<u>livres">
  <id name="titre" type="string">
    <column length="50" name="titre"/>
    <generator class="assigned"/>
  </id>
  <many-to-one class="hibernatejointure.Categories" name="categories">
    <!--fetch="select"-->
    <column length="30" name="categorie"/>
  </many-to-one>
  <many-to-one class="hibernatejointure.Auteurs" name="auteurs">
    <!--fetch="select"-->
    <column length="40" name="auteur"/>
  </many-to-one>
</class>
</hibernate-mapping>
```

Il devient alors assez simple de rédiger un petit programme affichant les données :

Jointure.java

```
package hibernatejointure;

import java.util.List;
import java.util.Vector;
import org.hibernate.Query;
import org.hibernate.HibernateException;
import org.hibernate.Session;

/**
 *
 * @author Vilvens le Sage
 */
public class Jointure
{
  public static void main(String[] args)
  {
    try
    {
      Session session = HibernateUtilMySql.getSessionFactory().openSession();
      session.beginTransaction();
      Query q = (Query)session.createQuery("from Livres");
    }
  }
}
```

```

List<Livres> resultList = (List<Livres>) q.list();
displayResult(session,resultList);
session.getTransaction().commit();
}
catch (HibernateException he)
{
    he.printStackTrace();
}
}

private static void displayResult(Session s, List<Livres> resultList)
{
    Vector tableData = new Vector();

    for(Livres o : resultList)
    {
        Livres article = (Livres)o;
        Vector<Object> oneRow = new Vector<Object>();
        oneRow.add(article.getTitre());
        oneRow.add(article.getAuteurs().getNom());
        oneRow.add(article.getAuteurs().getPrenom());
        oneRow.add(article.getCategories().getIntitule());
        oneRow.add(article.getCategories().getGrandPublic());
        tableData.add(oneRow);
        System.out.println(tableData.toString());
    }
}
}

```

Résultat :

```

...
06-nov.-2010 14:42:26 org.hibernate.cfg.Configuration getConfigurationInputStream
INFO: Configuration resource: /hibernate.cfg.xml
06-nov.-2010 14:42:26 org.hibernate.cfg.Configuration addResource
INFO: Reading mappings from resource : hibernatejointure/Livres.hbm.xml
06-nov.-2010 14:42:26 org.hibernate.cfg.Configuration addResource
INFO: Reading mappings from resource : hibernatejointure/Auteurs.hbm.xml
06-nov.-2010 14:42:26 org.hibernate.cfg.Configuration addResource
INFO: Reading mappings from resource : hibernatejointure/Categories.hbm.xml
...
06-nov.-2010 14:42:26 org.hibernate.connection.DriverManagerConnectionProvider
configure
INFO: using driver: com.mysql.jdbc.Driver at URL: jdbc:mysql://localhost:3306/librairie
06-nov.-2010 14:42:26 org.hibernate.connection.DriverManagerConnectionProvider
configure
INFO: connection properties: {user=root, password=****}
...
06-nov.-2010 14:42:27 org.hibernate.cfg.SettingsFactory buildSettings

```

INFO: JDBC driver: MySQL-AB JDBC Driver, version: mysql-connector-java-5.1.6 (Revision: \${svn.Revision})

...

[**EJB pour les nuls, Gide, André, didactique, 0**], [**Java V - Systèmes logiciels embarqués, vilvens, Claude, didactique, 0**], [**La porte étroite de l'ISIL, Gide, André, roman, 1**]]

On se retrouve bien avec l'équivalent de :

```
SELECT DISTINCT
    livres.titre, livres.auteur, livres.categorie, categories.intitule, categories.grand_public,
    auteurs.nom, auteurs.prenom
FROM
    (
        librairie.livres livres
        INNER JOIN
        librairie.auteurs auteurs
        ON (livres.auteur = auteurs.nom)
    )
        INNER JOIN
        librairie.categories categories
        ON (livres.categorie = categories.intitule)
```

titre *	auteur	categorie	intitule *	grand_public	nom *	prenom
EJB pour les nuls	Gide	didactique	didactique	0	Gide	André
Java V - Systèmes logiciels embarqués	vilvens	didactique	didactique	0	Vilvens	Claude
La porte étroite de l'ISIL	Gide	roman	roman	1	Gide	André

Mais, au fait ... que signifient ces attributs "lazy" et "fetch" dans les fichiers hbm.xml ?

8. Deux optimisations d'Hibernate

8.1 Les requêtes paramétrées

Si on reprend la requête de sélection dans la table de stocks, on la complétant d'une clause "where", il est préférable d'utiliser une requête paramétrée (un "PreparedStatement") plutôt qu'une requête statique (un "Statement"). Le HQL s'accorde de cette manière de faire au moyen des méthodes

```
public Query setString(int position, String val)
public Query setCharacter(int position, char val)
public Query setBoolean(int position, boolean val)
public Query setByte(int position, byte val)
public Query setShort(int position, short val)
public Query setInteger(int position, int val)
public Query setLong(int position, long val)
public Query setFloat(int position, float val)
public Query setDouble(int position, double val)
...
```

Par exemple :

```
Query q = (Query)session.createQuery("from Stocks where codeSto=?");  
q.setString(0, "m2123-58");
```

De cette manière, la requête est traitée en deux temps :

- ◆ envoi et précompilation de la requête "select * from Stocks where codeSto=?";
- ◆ envoi du paramètre et traitement effectif.

La même requête peut évidemment être réutilisée avec une simple modification de paramètre. Le gain de temps (du à la précompilation de la requête de base) est de l'ordre d'un facteur 100.

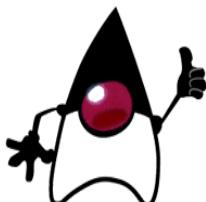
8.2 Le lazy loading

Les exemples simplissimes proposés ci-dessus pourraient nous faire perdre de vue une réalité pourtant évidente : les tables d'une base de données relationnelle sont liées par des relations de type clé étrangère ou table de relations. Dès lors, les objets associés sont également liés. On conçoit donc sans peine que Hibernate doit tenir compte de ces liaisons, ce qu'il fait en gérant un **graphe d'objets**. A priori, celui-ci doit être chargé intégralement en mémoire lors de l'exécution de l'application utilisatrice. Mais la lourdeur du procédé fait craindre, avec raison, des baisses de performances non négligeables.

Pour cette raison, Hibernate pratique par défaut le "**lazy loading**" (chargement à la demande) : lorsqu'une application accède à un objet, on ne charge en mémoire que les données strictement nécessaires à l'instruction en cours (autrement dit les données de la table principale, seules les clés étrangères des autres table étant effectivement chargées), quitte à générer une nouvelle requête SQL pour récupérer les données supplémentaires quand elles sont nécessaires (en utilisant les clés étrangères : on parle encore d'accès par "proxy").

Le principe semble séduisant, mais peut cependant engendrer une inflation (comme dirait Rachida ...) de requêtes : il suffit de charger une série de tuples dans un container puis d'y rechercher de manière itérative une information pointée par une clé étrangère : alors qu'on aurait pu tout avoir par une seule (grosse, certes, requête), on engendre n+1 requêtes pour obtenir le même résultat. Dans ce cas de figure, le "eager-fetching" est préférable : il consiste évidemment à charger en mémoire l'intégralité du graphe d'objets. On peut le demander en utilisant l'attribut "lazy" mis à "false" (et "fetch" à "join") dans les associations définies dans les fichiers hbm.xml.

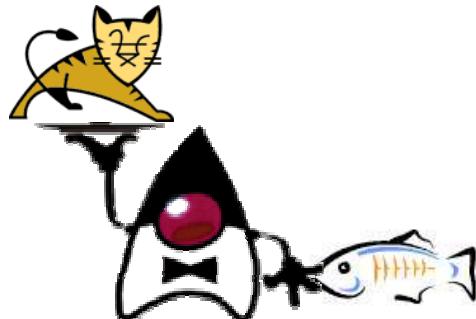
Mais l'idéal est évidemment d'optimiser soi-même le transfert d'informations en définissant explicitement soi-même les jointures nécessaires au déroulement de l'application au lieu de laisser une mécanisme automatisé décider seul ...



Il y a bien entendu beaucoup d'autres choses à explorer dans Hibernate, comme par exemple les questions traitant de l'héritage ou de la gestion des caches. Etrange monde vraiment que celui-là qui mélange objets et bases de données ...

Mais le monde du développement actuel dispose de tant de plate-formes, de serveurs d'applications et de frameworks qu'il serait dommage ne pas aller butiner dans ce pré si fleuri ... cela tout de même avec un minimum de sécurité ;-)

XXXV. Les contrôles d'accès sous Tomcat et GlassFish



Avant donc que d'écrire, apprenez à penser.

(N. Boileau, L'Art poétique)

Bien sûr, nous connaissons les mécanismes de sécurité de la plate-forme Java. Et nous savons aussi comment contrôler les accès aux ressources d'une application Web par des moyens "maison" au parfun d'objet session. Mais le serveur Tomcat, et le serveur GlassFish dans sa foulée, ont ajouté à ces mécanismes bien connus des possibilités complémentaires de contrôle des accès au sein même du serveur : on parle encore de "*realms*" – mais de quels "royaumes" s'agit-il ?

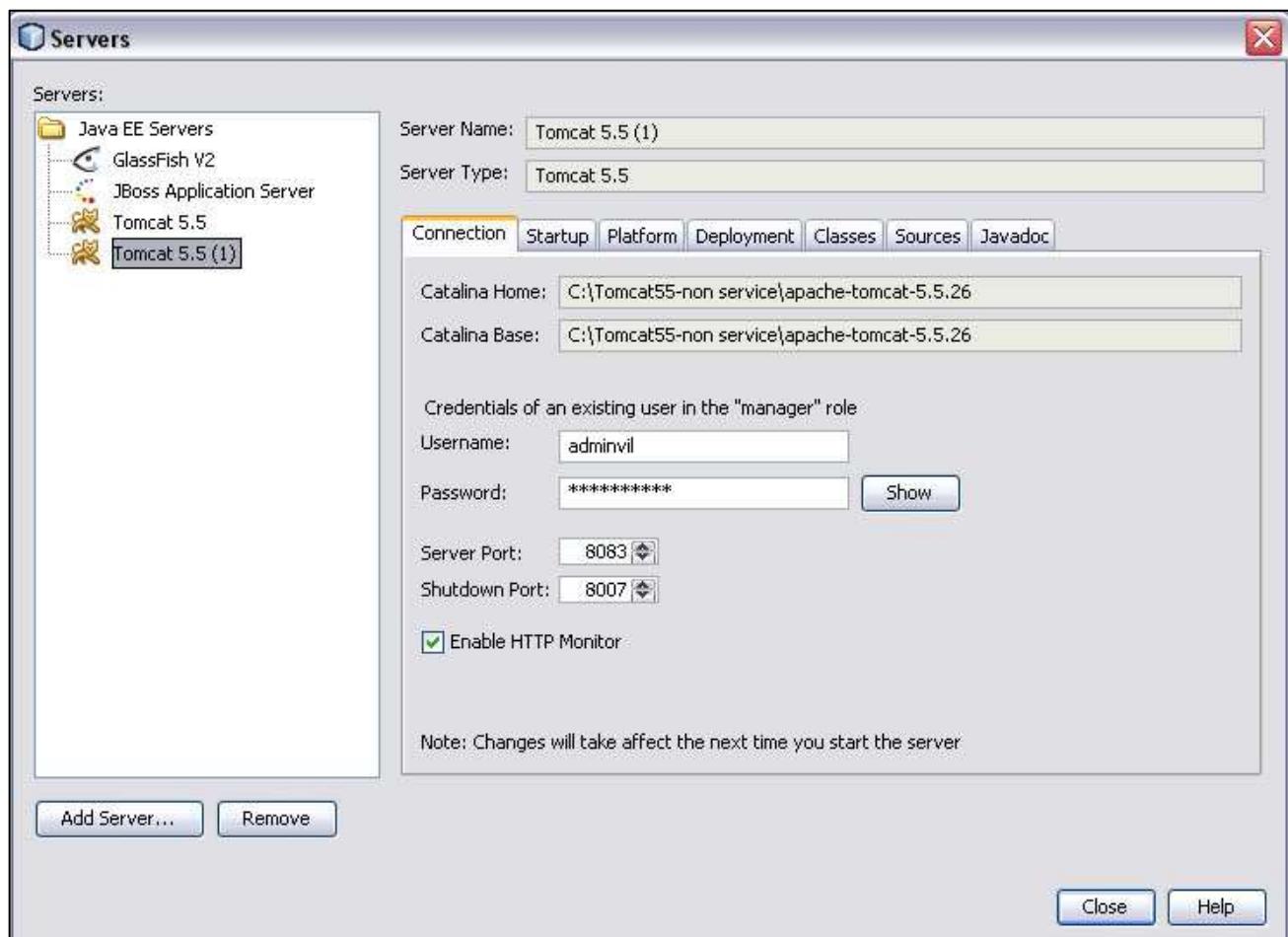
1. Un Tomcat intégré

Tomcat nous est bien connu de par nos exploits en programmation Web (voir Java III ;-)). Si ce serveur était intégré aux versions 4 et 5 de NetBeans, il n'en est plus ainsi dans les versions 6 (GlassFish l'a remplacé). Nous allons donc en ajouter un à NetBeans dans une version qui ne sera pas montée en service Windows. Pour cela, nous allons télécharger un Tomcat sous forme de fichier zip (donc pas sous la forme "Windows Service Installer"). On obtient un fichier :

apache-tomcat-5.5.26.zip

que l'on peut décompresser, par exemple et dans notre cas, dans un répertoire Tomcat55-non service. Nous pouvons ensuite intégrer ce serveur par l'onglet Services :





On a choisi pour l'administration "adminvil" et "adminadmin". On peut faire démarrer le serveur depuis NetBeans et vérifier que tout est en ordre :

```
C:\>netstat -an | find "8083"
TCP 0.0.0.0:8083 0.0.0.0:0 LISTENING
```

Remarque : on peut aussi faire démarrer Tomcat indépendamment de NetBeans (en faisant glisser l'icône de startup.bat dans une fenêtre DOS) :

```
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>"C:\Tomcat55-non service\apache-tomcat-5.5.26\bin\startup.bat"
Neither the JAVA_HOME nor the JRE_HOME environment variable is defined
At least one of these environment variable is needed to run this program
```

```
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_06
```

```
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>"C:\Tomcat55-non service\apache-tomcat-5.5.26\bin\startup.bat"
```

Using CATALINA_BASE: C:\Tomcat55-non service\apache-tomcat-5.5.26

Using CATALINA_HOME: C:\Tomcat55-non service\apache-tomcat-5.5.26

Using CATALINA_TMPDIR: C:\Tomcat55-non service\apache-tomcat-5.5.26\temp

Using JRE_HOME: C:\Program Files\Java\jdk1.5.0_06

```
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>netstat -an | find "8083"  
TCP 0.0.0.0:8083 0.0.0.0:0 LISTENING
```

```
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>
```

Bien sûr, si l'on démarre depuis un répertoire quelconque, il faut tout d'abord initialiser les variables d'environnement citées plus haut – un fichier bat peut être créé à cet effet dans le répertoire bin :

tomcat-env.bat

```
set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_06  
set CATALINA_BASE=C:\Tomcat55-non service\apache-tomcat-5.5.26  
set CATALINA_HOME=C:\Tomcat55-non service\apache-tomcat-5.5.26  
set CATALINA_TMPDIR=C:\Tomcat55-non service\apache-tomcat-5.5.26\temp  
set JRE_HOME=C:\Program Files\Java\jdk1.5.0_06
```

puis exécuté par simple glissement dans la fenêtre DOS :

```
C:\>"C:\Tomcat55-non service\apache-tomcat-5.5.26\bin\tomcat-env.bat"  
C:\>set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_06  
C:\>set CATALINA_BASE=C:\Tomcat55-non service\apache-tomcat-5.5.26  
C:\>set CATALINA_HOME=C:\Tomcat55-non service\apache-tomcat-5.5.26  
C:\>set CATALINA_TMPDIR=C:\Tomcat55-non service\apache-tomcat-5.5.26\temp  
C:\>set JRE_HOME=C:\Program Files\Java\jdk1.5.0_06  
C:\>"C:\Tomcat55-non service\apache-tomcat-5.5.26\bin\startup.bat"  
Using CATALINA_BASE: C:\Tomcat55-non service\apache-tomcat-5.5.26  
Using CATALINA_HOME: C:\Tomcat55-non service\apache-tomcat-5.5.26  
Using CATALINA_TMPDIR: C:\Tomcat55-non service\apache-tomcat-5.5.26\temp  
Using JRE_HOME: C:\Program Files\Java\jdk1.5.0_06  
C:\>
```

Dans les deux cas, une deuxième fenêtre indique que tout s'est bien passé :

```
16-mars-2008 15:46:04 org.apache.catalina.core.AprLifecycleListener lifecycleEvent
```

```
INFO: The Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: C:\Program Files\Java\jdk1.5.0_06\bin;;.;C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Reflection;C:\Program Files\Reflection;C:\Program Files\Reflection;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\Executive Software\Diskeeper\;C:\Gemplus\GemXpresso.rad3\bin;C:\Gemplus\GEMXPR~1.RAD\bin;C:\Program Files\MySQL\MySQL Server 5.0\bin;C:\Sun\AppServer\bin;C:\Gemplus\GemXpresso.rad3\bin;C:\Gemplus\GemXpresso.rad3\bin;C:\Gemplus\GemXpresso.rad3\bin;C:\Gemplus\GemXpresso.rad3\bin;C:\Gemplus\GemXpresso.rad3\bin;C:\Gemplus\GEMXPR~1.RAD\bin;C:\Program Files\Nmap;C:\cygwin\usr\bin;C:\cygwin\bin
```

```
16-mars-2008 15:46:04 org.apache.coyote.http11.Http11BaseProtocol init
```

```
INFO: Initialisation de Coyote HTTP/1.1 sur http-8083
```

```
16-mars-2008 15:46:04 org.apache.catalina.startup.Catalina load
```

```
INFO: Initialization processed in 1094 ms
16-mars-2008 15:46:05 org.apache.catalina.core.StandardService start
INFO: Démarrage du service Catalina
16-mars-2008 15:46:05 org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/5.5.26
16-mars-2008 15:46:05 org.apache.catalina.core.StandardHost start
INFO: XML validation disabled
16-mars-2008 15:46:07 org.apache.coyote.http11.Http11BaseProtocol start
INFO: Démarrage de Coyote HTTP/1.1 sur http-8083
16-mars-2008 15:46:07 org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
16-mars-2008 15:46:07 org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/16 config=null
16-mars-2008 15:46:08 org.apache.catalina.storeconfig.StoreLoader load
INFO: Find registry server-registry.xml at classpath resource
16-mars-2008 15:46:08 org.apache.catalina.startup.Catalina start
INFO: Server startup in 3187 ms
```

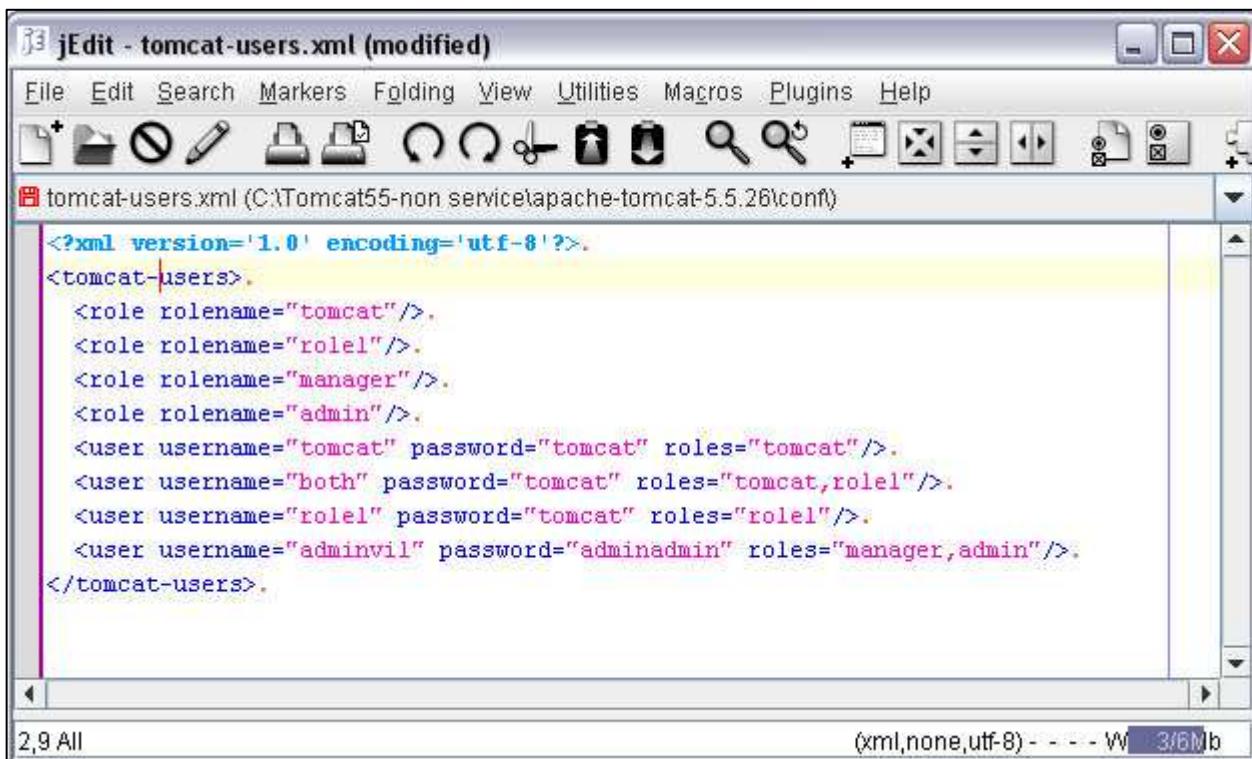
On peut placer un CaddieVirtuel.war dans le répertoire wabapps – pas de problème :



Le décor étant ainsi (re)planté, voyons comment procéder au sein de Tomcat pour ne permettre l'accès à certaines ressources qu'à certains utilisateurs.

2. Les utilisateurs de Tomcat

Les comptes utilisateurs de Tomcat (autre que les anonymes, évidemment) sont définis dans le fichier **tomcat-users.xml** (répertoire conf) :



```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
    <role rolename="tomcat"/>
    <role rolename="role1"/>
    <role rolename="manager"/>
    <role rolename="admin"/>
    <user username="tomcat" password="tomcat" roles="tomcat"/>
    <user username="both" password="tomcat" roles="tomcat,role1"/>
    <user username="role1" password="tomcat" roles="role1"/>
    <user username="adminvil" password="adminadmin" roles="manager,admin"/>
</tomcat-users>
```

Nous pouvons constater la présence de deux types de tags :

- ♦ les "**role**" : il s'agit simplement d'un groupe d'utilisateurs :

```
<role rolename="..."/>
```

- ♦ les "**user**" : pour définir un utilisateur appartenant à un (ou même plusieurs) role donné :

```
<user username="..." password="..." roles="..."/>
```

Nous pouvons donc ajouter un nouveau role avec de nouveaux users :

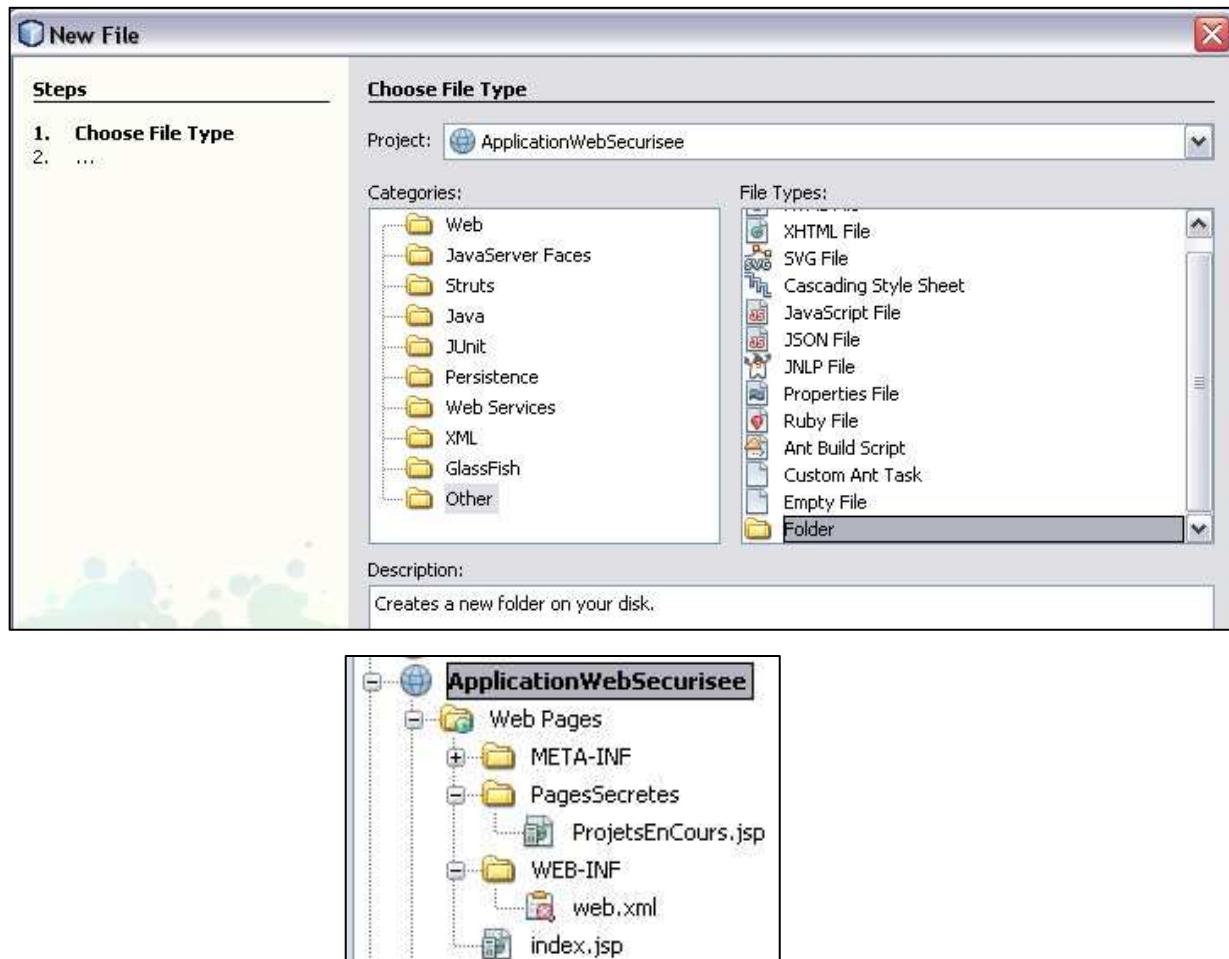
tomcat-users.xml
<?xml version='1.0' encoding='utf-8'?> <tomcat-users> <role rolename="tomcat"/> ... <user username="adminvil" password="adminadmin" roles="manager,admin"/> <role rolename="java-team"/> <user username="vilvens" password="genius" roles="java-team"/> <user username="wagner" password="groscerveau" roles="java-team"/> </tomcat-users>

Fort bien. Mais comment tenir compte de ces roles ?

3. La définition de ressources protégées

3.1 Une application Web banale

Nous commençons par créer une modeste application Web ("ApplicationWebSecurisee") de la manière habituelle. Les JSP de cette application seront cependant placés dans un répertoire (disons "PagesSecretes") qui fera l'objet de nos contrôles :



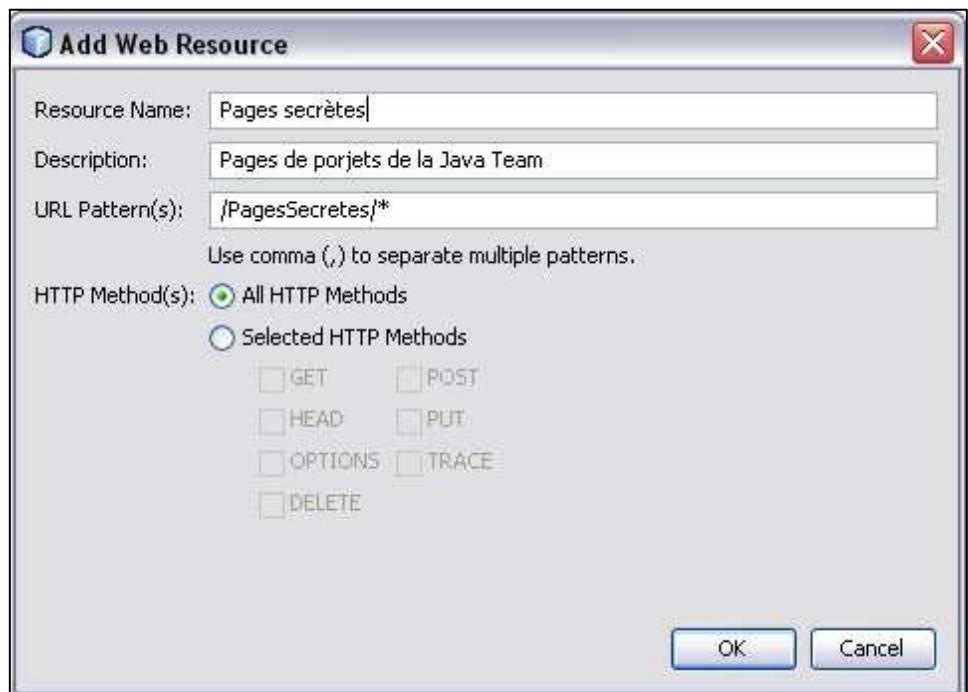
3.2 La définition des restrictions

La définition des contraintes de sécurité s'effectue dans le fichier web.xml de l'application. La version "assistée" de cette édition laisse apparaître un onglet "Security" dans lequel nous pouvons définir :

- ♦ les roles qui vont intervenir :



- ♦ les ressources visées :



- ♦ les rôles associés à cette contrainte et une procédure d'authentification de login qualifiée de **BASIC** :

The image shows two 'Edit Role Names' dialog boxes. In the first, the role 'java-team' is selected in the left list and is being moved to the right list using the 'Add >' button. In the second, 'java-team' is already listed in the right list. Below these dialogs is a 'Login Configuration' section. It includes radio buttons for 'None', 'Digest', 'Client Certificate', and 'Basic' (which is selected), and a 'Realm Name:' input field. It also includes sections for 'Form' authentication with 'Form Login Page:' and 'Form Error Page:' fields and browse buttons.

Au total :

The screenshot shows the NetBeans IDE interface with the 'Security' tab selected in the top navigation bar. Below, the 'Security Roles' panel is open, displaying a table with one role entry:

Role Name	Description
java-team	L'équipe Java de l'Inpres-Isil

Buttons for 'Add...', 'Edit...', and 'Remove...' are visible at the bottom of this panel.

Below the roles, the 'Security Constraints' panel is open, showing a constraint named 'AccesPagesSecretes'. It includes fields for 'Display Name' (set to 'AccesPagesSecretes'), 'Web Resource Collection' (with a table listing a single resource), and 'Description' (set to 'Les profs de Java et associés'). The 'Role Name(s)' field is set to 'java-team'. A checked checkbox for 'Enable Authentication Constraint' is present, along with an unchecked checkbox for 'Enable User Data Constraint'.

c'est-à-dire en clair :

```
web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
  app_2_4.xsd">
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <b><security-constraint></b>
    <display-name>AccesPagesSecretes</display-name>
    <b><web-resource-collection></b>
      <web-resource-name>Pages secrètes</web-resource-name>
      <description>Pages de projets de la Java Team</description>
      <b><url-pattern>/PagesSecretes/*</url-pattern></b>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
      <http-method>HEAD</http-method>
      <http-method>PUT</http-method>
```

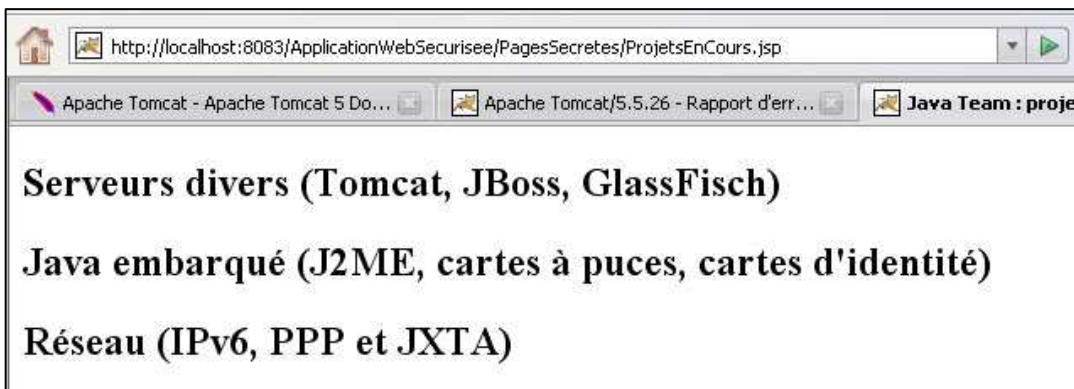
```
<http-method>OPTIONS</http-method>
<http-method>TRACE</http-method>
<http-method>DELETE</http-method>
</web-resource-collection>
<auth-constraint>
  <description>Les profs de Java et associés</description>
  <role-name>java-team</role-name>
</auth-constraint>
</security-constraint>
<security-constraint>
  <display-name>Constraint1</display-name>
</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name/>
</login-config>
<security-role>
  <description>L'équipe Java de l'Inpres-Isil</description>
  <role-name>java-team</role-name>
</security-role>
</web-app>
```

3.3 L'utilisation de l'application Web

Si nous tentons d'exécuter la page JSP, nous obtenons une fenêtre de login :



puis finalement :



En cas d'erreur de login, on boucle sur la boîte dialogue d'entrée et si on la ferme de force, on obtient :



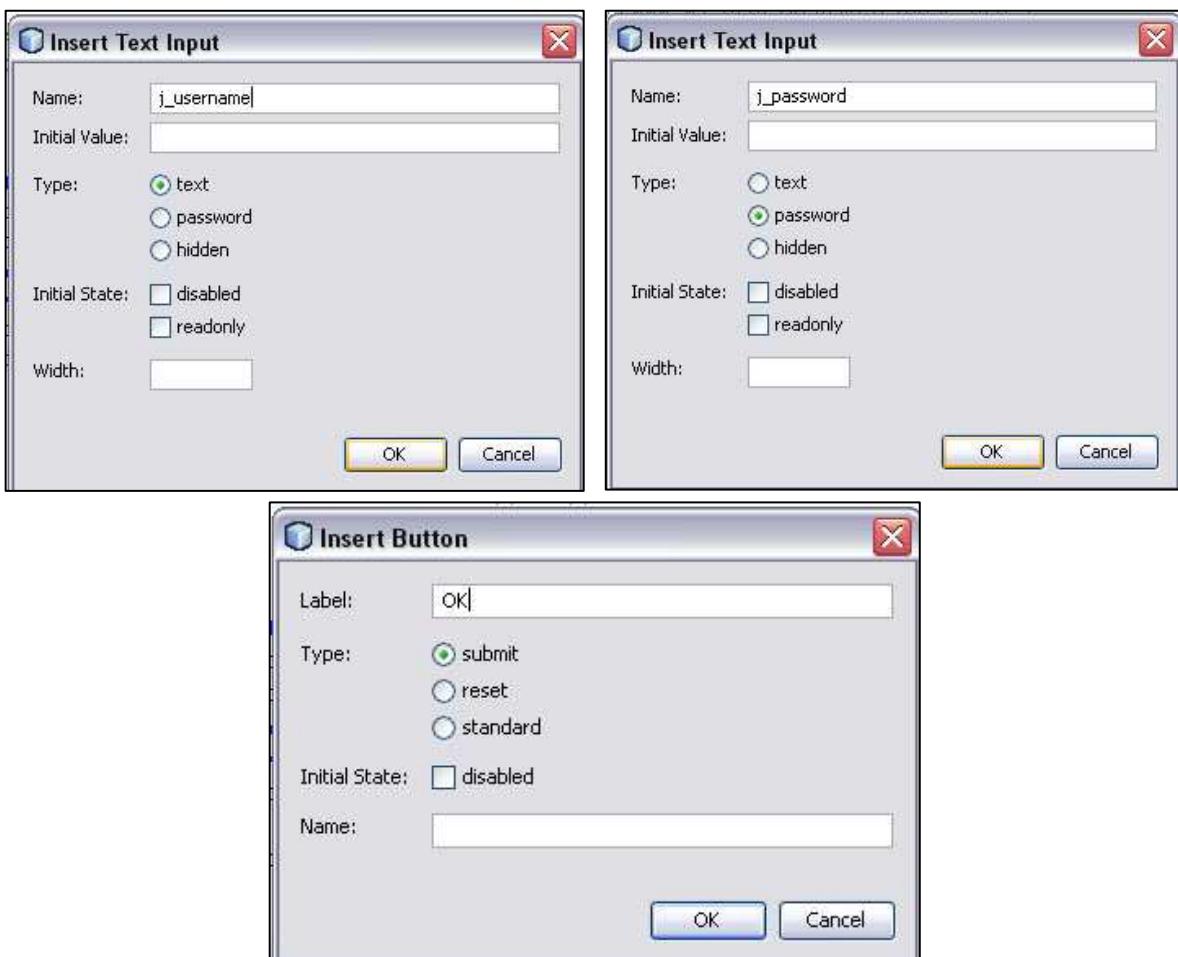
Evidemment c'est peu élégant : changeons donc la procédure de login ...

4. La definition de resources protégées : bis

Nous allons modifier notre procédé de manière à ce que les JSPs de PagesSecretes soient toujours protégées, mais aussi qu'une *page d'erreur plus personnalisée* soit envoyée au client en cas de problème plutôt que la page d'erreur (trop) classique du serveur Web. Pour ce faire, nous changeons de mode d'authentification du login utilisant le mode **FORM**. au lieu du mode **BASIC**. Ceci sous-entend que nous définissons au préalable deux JSPs :

a) un JSP de login (disons **Login.jsp**) qui permet d'introduire les informations d'authentification comme le formulaire par défaut; en fait, les données envoyées par ce JSP seront traitées par une servlet de Tomcat nommée **j_security_check** qui attend deux paramètres aux noms prédefinis : **j_username** et **j_password** et force nous est donc de créer le formulaire de cette manière au sein du JSP :





Ce qui donne :

Login.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Java Team : accès aux projets en cours</title>
</head>
<body>
    <h2>Bienvenue ! Veuillez vous identifier ....</h2>

    <form name="formulaire-login" action="j_security_check" method="POST">
        Votre nom d'utilisateur : <input type="text" name="j_username" value="" /><p></p>
        Votre mot de passe : <input type="password" name="j_password" value="" /><p></p>
        <input type="submit" value="OK" />
    </form>
</body>
</html>
```

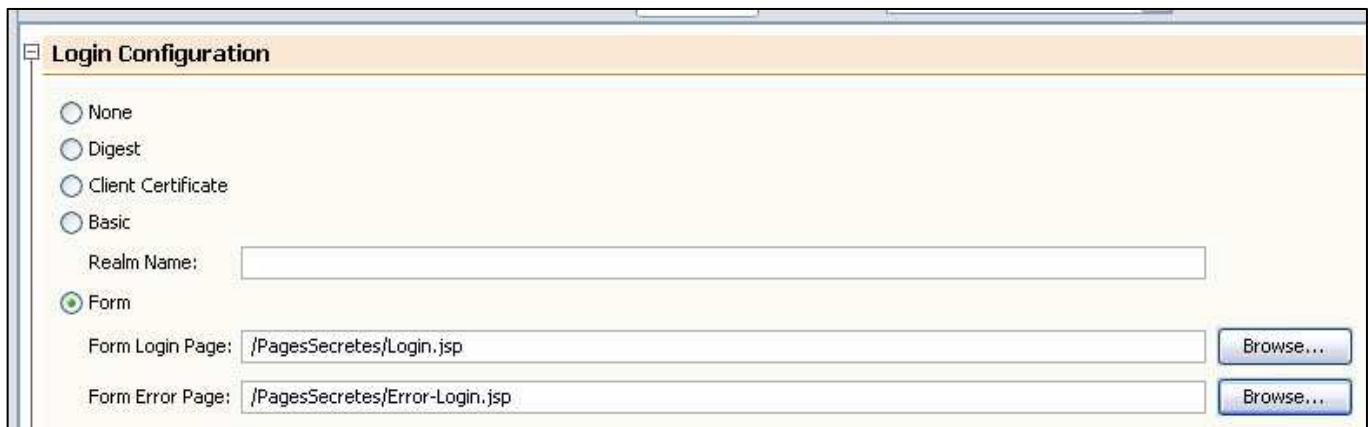
b) un JSP d'erreur des plus simples :

Error-Login.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Java Team : refus d'accès aux projets en cours</title>
  </head>
  <body>
    <h2>Désolé : il semblerait que vous ne disposiez pas des accès aux ressources
 demandées :-(</h2>
  </body>
</html>
```

Nous pouvons alors modifier web.xml pour que soit tenu compte de notre mode FORM :

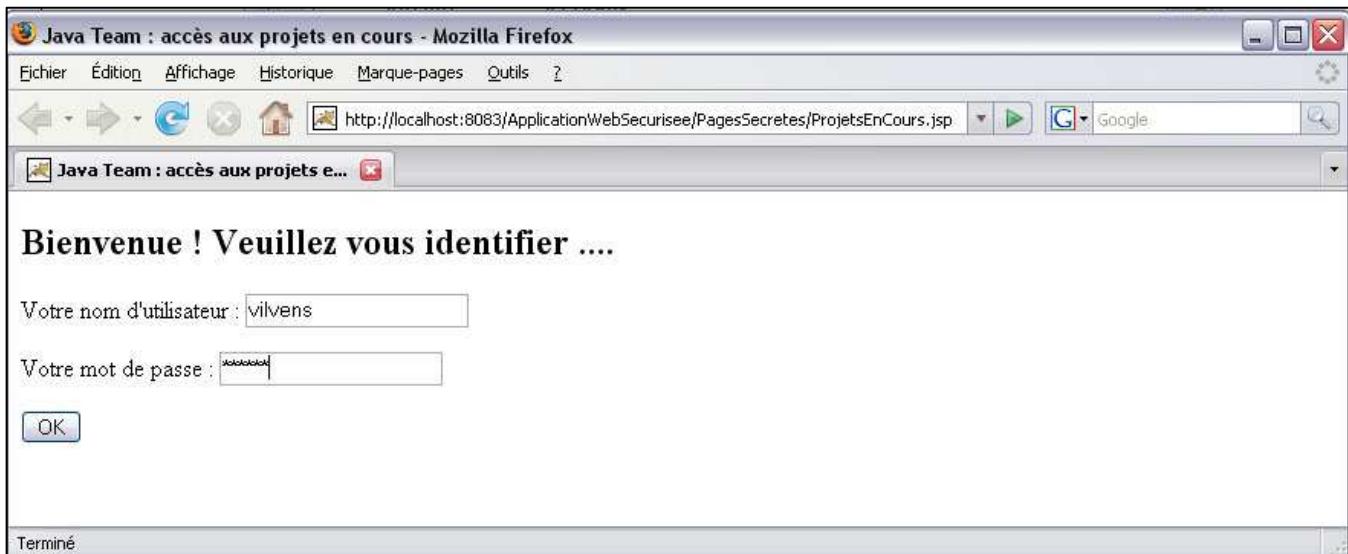


ce qui comme effet dans le code :

web.xml (mode FORM)

```
...
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name/>
  <form-login-config>
    <form-login-page>/PagesSecretes/Login.jsp</form-login-page>
    <form-error-page>/PagesSecretes/Error-Login.jsp</form-error-page>
  </form-login-config>
</login-config>...
```

Un essai d'accès à la page ProjetsEnCours.jsp donne :



et un accès autorisé tandis que des entrées erronées donnent :



Parfait – que vouloir de plus ?

5. L'utilitaire de digest de Tomcat

L'histoire est bien connue, évidemment : le mot de passe est codé en clair dans le fichier de configuration tomcat-users.xml ! Il serait évidemment plus raisonnable de n'y placer que le digest de celui-ci. C'est possible en mode **DIGEST** : on stocke dans tomcat-users.xml un digest du mot de passe obtenu avec un utilitaire "digest" se trouvant dans le répertoire bin de Tomcat :

```
C:\Program Files\Java\jdk1.5.0_06>cd C:\Tomcat55-non service\apache-tomcat-5.5.26\bin
```

```
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>digest
```

Neither the JAVA_HOME nor the JRE_HOME environment variable is defined
At least one of these environment variable is needed to run this program
'-Djava.endorsed.dirs' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.

```
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_06
```

```
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>digest  
Usage: RealmBase -a <algorithm> [-e <encoding>] <credentials>  
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>digest -a SHA1 genius  
genius:3dcad53b7bcd2d77a9c8abf601016b7adbdbfa  
C:\Tomcat55-non service\apache-tomcat-5.5.26\bin>digest -a SHA genius  
genius:3dcad53b7bcd2d77a9c8abf601016b7adbdbfa
```

Dans tomcat-users.xml, on trouvera

```
<user username="vilvens" password="3dcad53b7bcd2d77a9c8abf601016b7adbdbfa"  
roles="java-team"/>
```

et dans web.xml :



c'est-à-dire

web.xml (mode DIGEST)

```
...  
<login-config>  
    <auth-method>DIGEST</auth-method>  
</login-config>  
...
```

Lors de l'exécution de l'application Web, tout se passe comme en mode BASIC, mais c'est le digest qui est calculé par Tomcat sur base du password introduit, le résultat étant ensuite comparé avec le digest memorisé dans le fichier de configuration des utilisateurs. Mais que vouloir de plus ?

6. Les realms

6.1 Un gestionnaire d'authentification

De manière générale, les outils d'authentification d'un utilisateur peuvent revêtir diverses formes : des fichiers XML comme ceux évoqués ci-dessus, un serveur LDAP, ou encore bien sûr une base de données. On qualifie de tels moyens d'authentification de "**registre utilisateurs**". Ainsi, le fichier tomcat-users.xml (répertoire conf), chargé en mémoire au démarrage du serveur, nous a servi de registre utilisateurs dans les exemples ci-dessus.

La manière d'accéder et d'utiliser un tel registre sera la tâche d'un **gestionnaire d'authentification**, encore appelé un "realm" ("royaume" en anglais) que le serveur considéré utilisera lors de chaque opération de reconnaissance d'un utilisateur autorisé.

Un realm, dans le cas où il est dévolu à Tomcat, peut être défini à différents niveaux du server.xml (Engine, Host ou Context) ce qui définit évidemment aussi sa portée : respectivement, tous les hôtes virtuels avec toutes leurs applications Web, toutes les applications Web de l'hôte considéré ou la seule application Web associée au contexte.

6.2 Le JDBC Realm

Le registre utilisateurs le plus naturel, à partir du moment où notre fichier tomcat-users.xml se révèle inopérant (par exemple parce que le nombre d'utilisateurs devient trop grand ou trop dynamique), est bien sûr une base de données. Le realm correspondant aura donc pour tâche de gérer l'authentification sur base des informations qui seront stockés dans cette base.

Ici, nous utiliserons une base MySQL et elle sera créée au plus simple : une table pour les utilisateurs, une autre pour mémoriser le relation utilisateur-role. La gestion des clés étrangères étant parfois sujette à caution en MySQL, la structure suivante a été utilisée :

utilisateurs	
PK	unom
	umotdepasse
	nom
	prénom

roles	
PK	unom
	rnom

En pratique, cela donne :

```
mysql> create database rolestomcat;
Query OK, 1 row affected (0.03 sec)
```

```
mysql> use rolestomcat;
Database changed
mysql> show tables;
Empty set (0.02 sec)
mysql>
```

```
mysql> create table utilisateurs ( unom varchar(15) not null primary key, umotde passe
varchar(15) not null, nom varchar(30), prenom varchar(30));
Query OK, 0 rows affected (0.16 sec)
mysql> create table roles ( unom varchar(15) not null, rnom varchar(15) not null, primary
key(unom, rnom) );
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> insert into utilisateurs values ("vilvens", "genius", "Vilvens", "Claude");
Query OK, 1 row affected (0.06 sec)
mysql> insert into utilisateurs values ("wagner", "groscerveau", "Wagner", "Jean-Marc");
Query OK, 1 row affected (0.03 sec)
```

```
mysql> select * from utilisateurs;
+-----+-----+-----+-----+
| unom | umotdepasse | nom | prenom |
+-----+-----+-----+
| vilvens | genius | Vilvens | Claude |
| wagner | groscerveau | Wagner | Jean-Marc |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> insert into roles values ("vilvens", "java-team");
Query OK, 1 row affected (0.03 sec)
mysql> insert into roles values ("wagner ", "java-team");
Query OK, 1 row affected (0.03 sec)
mysql> select * from roles;
+-----+-----+
| unom | rnom |
+-----+-----+
| vilvens | java-team |
| wagner | java-team |
+-----+
2 rows in set (0.00 sec)
```

C'est donc sur base de ces tables que les accès vont être testés.

6.3 La définition du realm

Le realm qui va utiliser notre base de données comme registre utilisateurs est en fait une instance de la classe **JDBCRealm** (du package org.apache.catalina.realm). Cette instance sera paramétrée au moyen des attributs d'un tag <**Realm**> qui sera placé dans le tag <Context> associé à l'application, habituellement dans context.xml (mais cela pourrait être dans server.xml, au niveau des tags <Engine> ou <Host> si les accès sont réglementés de la même manière pour toutes les applications Web d'un même hôte virtuel, voire même pour tous les hôtes virtuels de la machine). Ces attributs sont essentiellement, outre le className :

- ◆ connectionURL : l'URL de connexion à la base de données, selon la syntaxe JDBC;
- ◆ connectionName et connectionPassword : le nom d'utilisateur et le mot de passe associé;
- ◆ driverName : la classe du driver à utiliser;

- ◆ userTable : le nom de la table qui contient les noms et mots de passe des utilisateurs;
- ◆ userNameCol et userCredCol : le nom des colonnes de cette table contentant ces noms et mots de passe;
- ◆ userRoleTable : le nom de la table qui contient les rôles et les noms associés;
- ◆ roleNameCol : le nom de la colonne de cette table contenant le nom des rôles;

- ◆ digest : facultatif – il s'agit du nom de l'algorithme de digest à utiliser si le mot de passe est en fait utilisé par digest interposé.

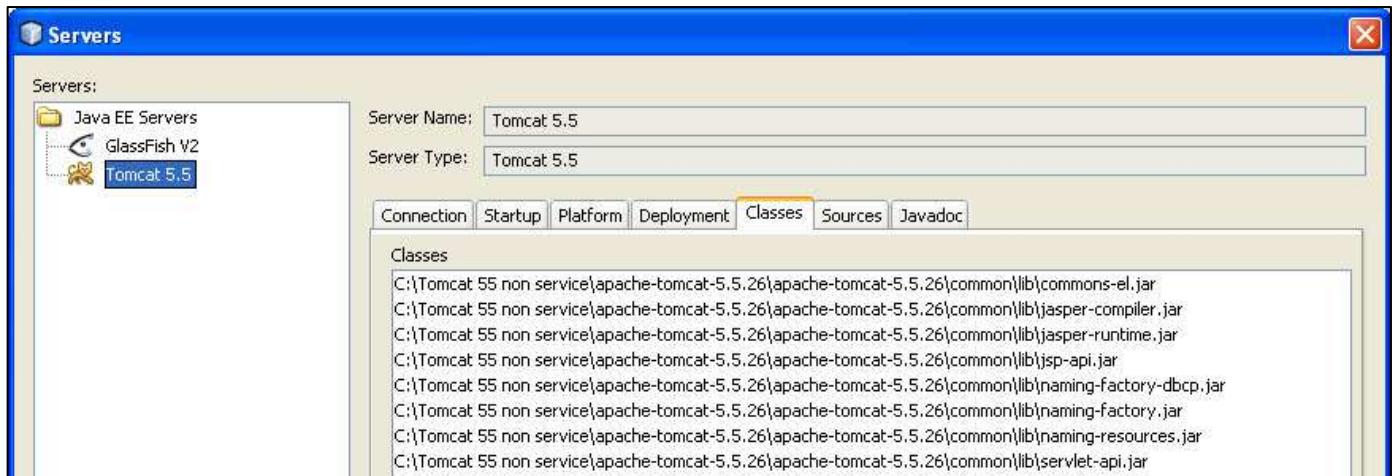
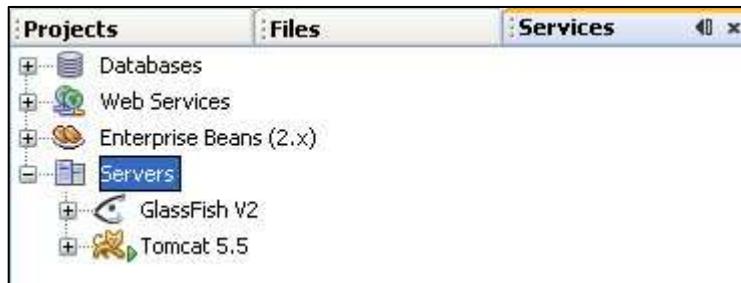
Dans notre cas, cela donne donc :



context.xml

```
<Context path="/ApplicationWebSecuriseeRealms">
    <Realm className="org.apache.catalina.realm.JDBCRealm"
        connectionName="root"
        connectionPassword="etpuisqueonencore"
        connectionURL="jdbc:mysql://localhost:3306/rolestomcat"
        driverName="com.mysql.jdbc.Driver"
        roleNameCol="rnom"
        userCredCol="umotdepasse"
        userNameCol="unom"
        userRoleTable="roles"
        userTable="utilisateurs"/>
</Context>
```

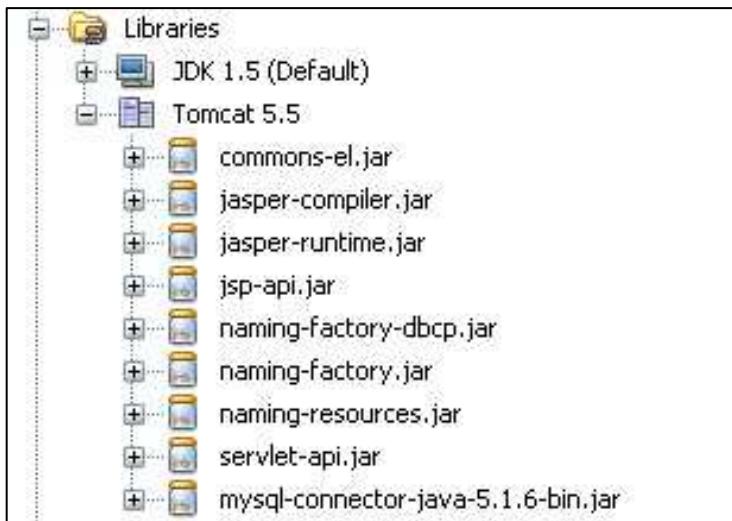
Il reste cependant une opération à réaliser. En effet, Tomcat va donc devoir accéder à la base de données MySQL : étant écrit en Java, il aura besoin des drivers JDBC correspondants. Pour cela, nous allons placer le fichier jar contenant ces drivers dans le répertoire où le serveur Tomcat utilise place ses ressources : dans le cas du Tomcat intégré à NetBeans, il suffit de consulter les propriétés du serveur pour découvrir ce répertoire :



Nous plaçons donc mysql-connector-java-5.1.6-bin.jar dans le répertoire indiqué :



Après redémarrage, on peut vérifier dans le noeud Libraries du projet que Tomcat dispose à présent du jar nécessaire :



6.4 L'utilisation de l'application Web

Reprendons une application Web similaire à celle utilisée ci-dessus :



- le fichier web.xml définit toujours les mêmes contraintes de sécurité en termes de "FORM" comme "auth-method" :

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  ...
    <security-constraint>
      <display-name>AccesPagesSecretes</display-name>
      <web-resource-collection>
        <web-resource-name>Pages secrètes</web-resource-name>
        <description>Pages de projets de la Java Team</description>
        <url-pattern>/PagesSecretes/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
      ...
      </web-resource-collection>
      <auth-constraint>
        <description>Les profs de Java et associés</description>
        <role-name>java-team</role-name>
      </auth-constraint>
    </security-constraint>

    <login-config>
      <auth-method>FORM</auth-method>
      <realm-name/>
      <form-login-config>
        <form-login-page>/PagesSecretes/Login.jsp</form-login-page>
        <form-error-page>/PagesSecretes/Error-Login.jsp</form-error-page>
      </form-login-config>
    </login-config>...

    <security-role>
      <description>L'équipe Java de l'Inpres-Isil</description>
      <role-name>java-team</role-name>
    </security-role>
  </web-app>
```

- la page de démarrage de l'application branche sur la page désirée :

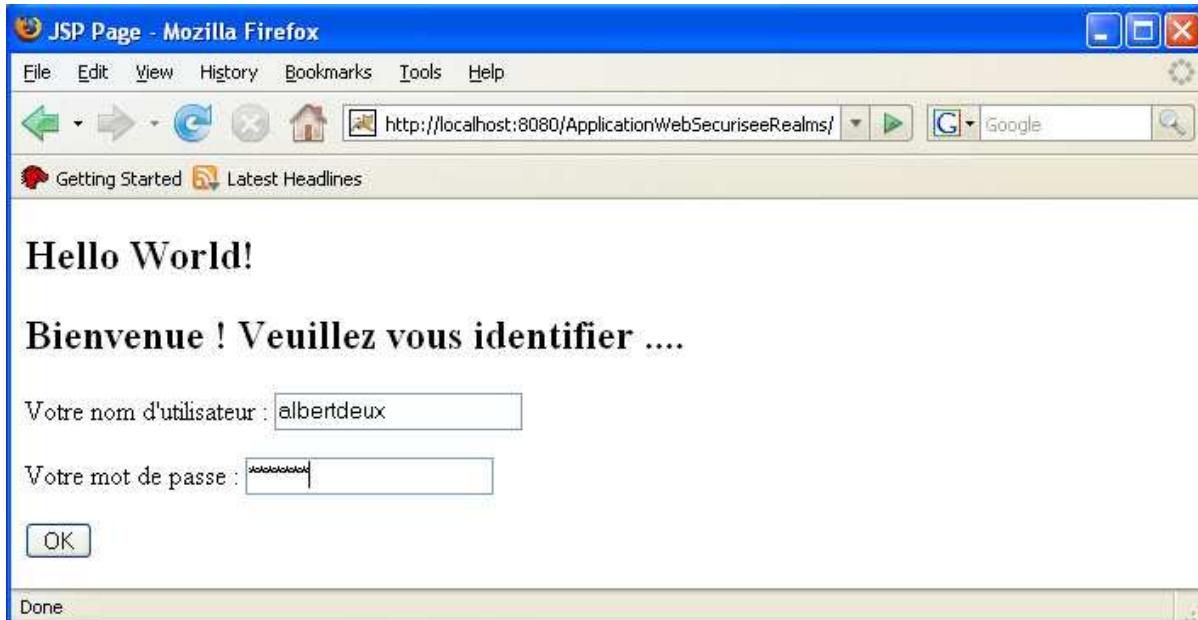
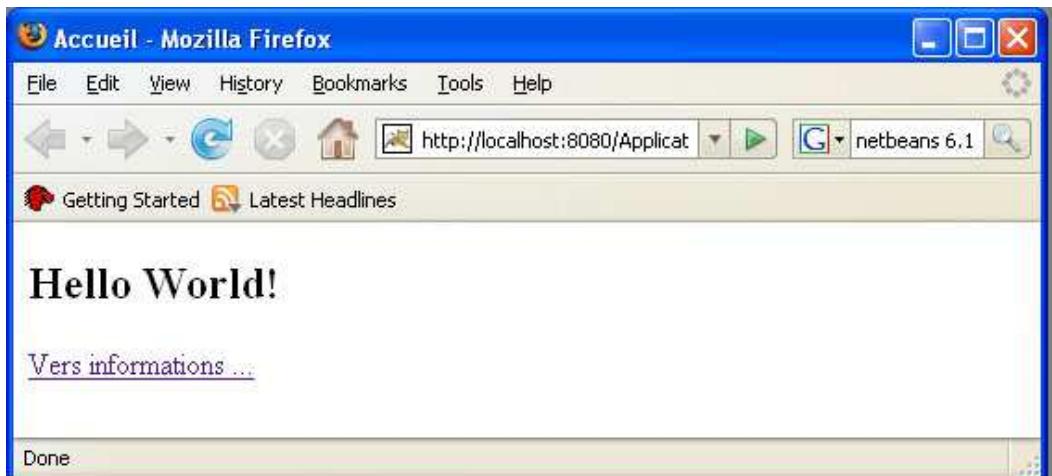
index.jsp

```
<%--
  Document : index
  Created on : 06-mai-2008, 17:07:42
  Author   : Vilvens
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Accueil</title>
  </head>
  <body>
    <h2>Hello World!</h2>
    <a href="PagesSecretes/Info.jsp">Vers informations ... </a>
  </body>
</html>
```

Deux essais sucessifs donnent :

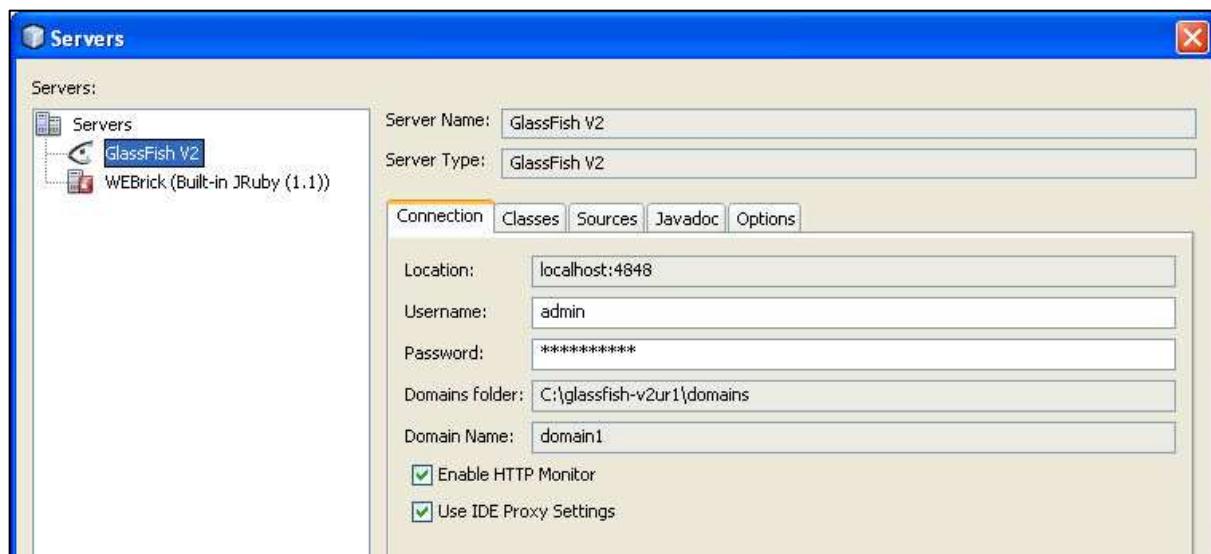


Cette fois, vraiment, que vouloir de plus ?

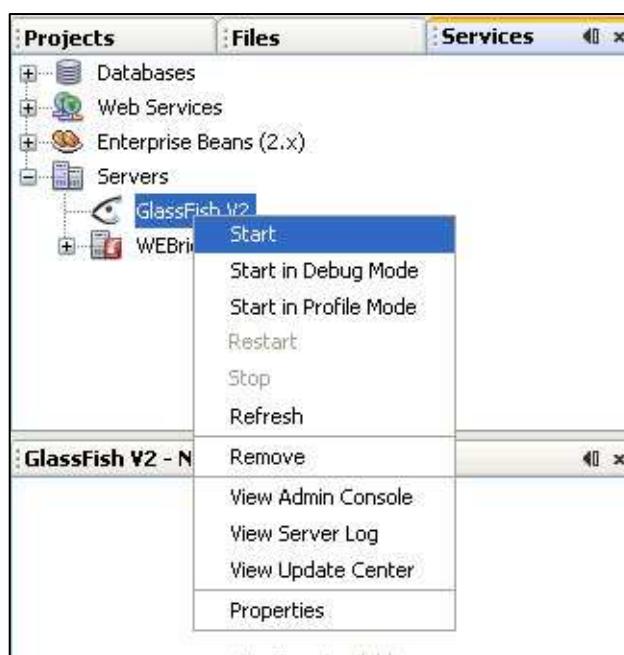
7. GlassFish

7.1 Un serveur d'application

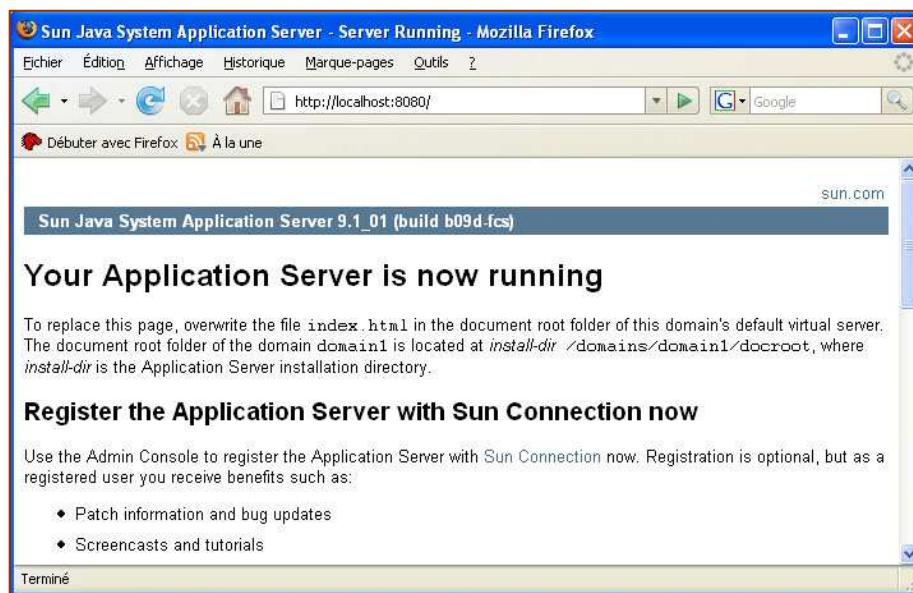
GlassFish (<https://GlassFish.dev.java.net/>) est un serveur d'application issu d'un projet open source et distribué sous licence GPL et CDDL. Il implémente la norme JEE 5, avec notamment les JSP 2.1, les JSF (JavaServer Faces) 1.2, les servlets 2.5, les EJB 3.0, le JAX-WS (Java API for Web Services) 2.0, le JAXB (Java Architecture for XML Binding) 2.0, etc (voir <http://java.sun.com/javaee/technologies/index.jsp> pour une présentation plus complète de la norme). On peut se procurer ce serveur par téléchargement et l'installer en suivant la documentation (un script Ant est à lancer). Cependant, les utilisateurs de NetBeans trouveront un GlassFish v2 intégré à la version 6.* du logiciel :



Ce serveur attend les requêtes HTTP et HTTPS respectivement sur les ports 8080 et 8181 tandis que le port 4848 est le port d'administration. On peut lancer le serveur par la fenêtre des Services :

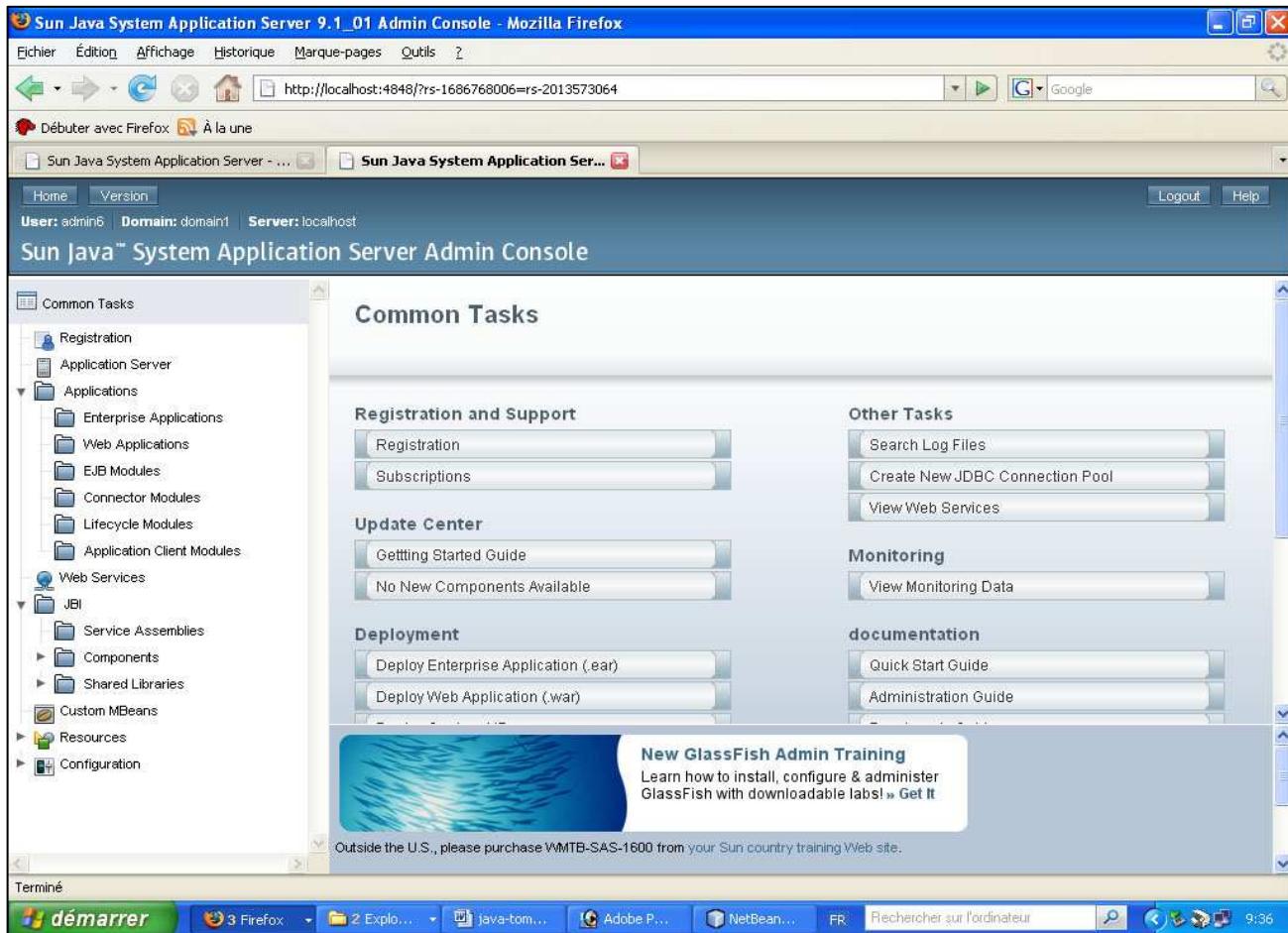


et vérifier qu'il attend bien sur le port 8080 :



7.2 La console d'administration

L'administration du serveur se fait par l'intermédiaire de la console d'administration (View Admin Console) – cet outil, introduit avec le JDK 5.0, permet aussi le JMX Monitoring (Java Management eXtension), mais ce n'est pas notre propos ici. Après introduction des login et mot de passe administrateur, on obtient :



avec, par exemple :

The screenshot shows the Sun Java System Application Server 9.1_01 Admin Console interface. The left sidebar contains a tree view of application components under 'Application Server'. The right panel displays 'General Information' with the following details:

Name:	localhost
HTTP Port(s):	4848,8181,8080
IIOP Port(s):	3700,3820,3920
JVM:	JVM Report
Configuration Directory:	C:\glassfish-v2ur1\domains\domain1\config
Installed Version:	Sun Java System Application Server 9.1_01 (build b09d-fcs)
Debug:	Not Enabled

Les répertoires de GlassFish se trouvent dans C:\GlassFish-v2ur1 :

The screenshot shows a Windows File Explorer window displaying the directory structure of GlassFish-v2ur1. The tree view on the left shows folders like 'glassfish-v2ur1', 'addons', 'bin', 'config', 'docs', 'domains', 'imq', 'javadb', 'jbi', 'lib', 'samples', and 'updatecenter'. The contents of the 'config' folder are listed in a table on the right:

Nom	Taille	Type	Date de modification
asadminenv	1 Ko	Fichier CONF	25/04/2008 13:27
asenv	2 Ko	Fichier de command...	25/04/2008 13:27

7.3 Les domaines

On aura remarqué dans les écrans précédents l'utilisation du mot "**domaine**". En fait, pour faire court, on peut dire qu'un domaine est un ensemble d'instances du serveur (c'est-à-dire des processus serveur J2EE utilisant une JVM et qui permettent le fonctionnement des entités J2EE) qui seront administrées en groupe. Le domaine possède sa propre configuration et un profil utilisateur, qui peut être :

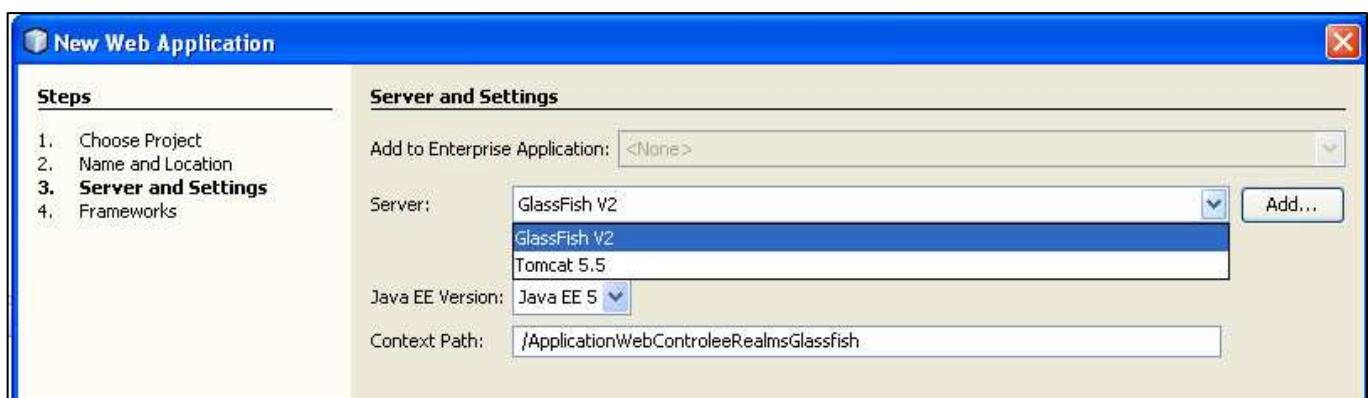
- ◆ **développer** : par défaut, à une seule instance, sans clustering ni load balancing;
- ◆ **cluster** : à plusieurs instances qui partagent leur configuration, leurs applications et leurs ressources; l'intérêt est que, par exemple, en cas de panne de l'une des instances, les clients des applications de ces instances sont automatiquement repris en charge par une autre instance du cluster, avec réplication de session (donc, concrètement, un article mis dans un caddie virtuel d'une application Web tournant sur une instance sera toujours dans le caddie lorsque l'application Web tournera sur une autre instance); évidemment, ceci implique que les différentes instances d'un cluster doivent constamment se synchroniser;
- ◆ **enterprise** : pour permettre un accès au NSS keystore (au lieu d'un classique keystore de type JKS).

L'idée est que les différents administrateurs de ces instances vont pouvoir ainsi partager la même configuration et les mêmes répertoires de déploiement. Chaque domaine possède son propre **Domain Administration Server (DAS)** qui attend sur un port bien précis : ainsi, dans NetBeans, l'instance présente par défaut appartient au domaine "domain1" et son DAS attend sur le port 4848. En fait, NetBeans donne accès à l'"**agent de nœud**" (*node agent*) qui gère l'instance, la (re)démarre ou l'arrête.

L'utilitaire asadmin (dans C:\GlassFish-v2\bin) permet, en ligne de commande, de créer ou faire démarrer un domaine avec des commandes create-domain, start-domain, stop-domain, etc.

7.4 Le déploiement d'une application Web sous GlassFish

Sous NetBeans, nous créons une application Web, disons ApplicationWebControleeRealmsGlassFish (pour faire court ?), en ayant bien soin de travailler avec GlassFish et JEE 5 :



Nous créons un JSP nomme Info.jsp dans un package PagesSecretes et nous faisons pointer index.jsp vers cet Info.jsp avec un simple lien HTML (intellectuel, non ? mais patience ...). Résultat à l'exécution de l'application :

1) le démarrage de GlassFish : la fenêtre de NetBeans "GlassFish v2" montre ce que l'on obtiendrait sur la ligne de commande avec un GlassFish indépendant (c'est pas triste ;- ...) :

```
01-sept.-2008 11:23:06 com.sun.enterprise.admin.servermgmt.launch.ASLauncher
buildCommand
INFO:
C:/Program Files/Java/jdk1.5.0_06/bin/java
-Dcom.sun.aas.instanceRoot=C:/GlassFish-v2ur1/domains/domain1
...
-Dcom.sun.aas.classloader.appserverChainJars=admin-cli.jar,admin-cli-ee.jar,j2ee-svc.jar
...
-Dcom.sun.aas.configName=server-config
-Dcom.sun.aas.configRoot=C:/GlassFish-v2ur1/config
-Dcom.sun.aas.defaultLogFile=C:/GlassFish-v2ur1/domains/domain1/logs/server.log
-Dcom.sun.aas.domainName=domain1
-Dcom.sun.aas.installRoot=C:/GlassFish-v2ur1
-Dcom.sun.aas.instanceName=server
...
-Dcom.sun.enterprise.overrideablejavaxpackages=javax.help,javax.portlet
-Dcom.sun.enterprise.taglibs=appserv-jstl.jar,jsf-impl.jar
-Dcom.sun.enterprise.taglisteners=jsf-impl.jar
...
-Ddomain.name=domain1
...
-Djava.ext.dirs=C:/Program Files/Java/jdk1.5.0_06/lib/ext;C:/Program
Files/Java/jdk1.5.0_06/jre/lib/ext;C:/GlassFish-
v2ur1/domains/domain1/lib/ext;C:/GlassFish-v2ur1/javadb/lib
-Djava.library.path=C:\GlassFish-v2ur1\lib;C:\GlassFish-v2ur1\lib;C:\GlassFish-
v2ur1\bin;C:\GlassFish-v2ur1\lib
-Djava.security.auth.login.config=C:/GlassFish-v2ur1/domains/domain1/config/login.conf
-Djava.security.policy=C:/GlassFish-v2ur1/domains/domain1/config/server.policy
...
-Djavax.net.ssl.keyStore=C:/GlassFish-v2ur1/domains/domain1/config/keystore.jks
-Djavax.net.ssl.trustStore=C:/GlassFish-v2ur1/domains/domain1/config/cacerts.jks
-Djdbc.drivers=org.apache.derby.jdbc.ClientDriver
...
...
Starting Sun Java System Application Server 9.1_01 (build b09d-fcs) ...
MBeanServer started: com.sun.enterprise.interceptor.DynamicInterceptor
CORE5098: AS Socket Service Initialization has been completed.
CORE5076: Using [Java HotSpot(TM) Client VM, Version 1.5.0_06] from [Sun
Microsystems Inc.]
SEC1002: Security Manager is OFF.
...
ADM1506: Status of Standard JMX Connector: Active = [true]
WEB0302: Starting Sun-Java-System/Application-Server.
JBIFW0010: JBI framework ready to accept requests.
WEB0712: Starting Sun-Java-System/Application-Server HTTP/1.1 on 8080
WEB0712: Starting Sun-Java-System/Application-Server HTTP/1.1 on 8181
WEB0712: Starting Sun-Java-System/Application-Server HTTP/1.1 on 4848
```

...

Application server startup complete.

deployed with moduleid = ***ApplicationWebControleeRealmsGlassFish***

Initializing Sun's JavaServer Faces implementation (1.2_04-b20-p03) for context "

2) l'exécution de l'application Web :

The image contains two screenshots of Mozilla Firefox browser windows. The top window, titled 'Accueil - Mozilla Firefox', shows the URL 'http://localhost:8080/ApplicationWebControleeRealmsGlassfish/'. The page content includes a heading 'Bienvenue !' and a link 'Accès aux informations ? Par ici ...'. The bottom window, titled 'Informations - Mozilla Firefox', shows the URL 'http://localhost:8080/ApplicationWebControleeRealmsGlassfish/Page'. The page content includes a heading 'Informations ?' and text 'Bonjouur !' and 'Dernière nouvelle people : le Prince Ph. a une relation avec Madonna :-o'. Both windows have a toolbar at the top and a status bar at the bottom indicating 'Terminé'.

3) le déploiement préalable de cette application que l'on peut vérifier dans la console d'administration :

The image shows the Sun Java System Application Server Admin Console interface. The top navigation bar includes links for 'Home', 'Version', 'Logout', and 'Help'. The central area displays the 'Sun Java™ System Application Server Admin Console'. On the left, a sidebar menu under 'Common Tasks' shows 'Registration', 'Application Server', 'Applications' (with sub-options: 'Enterprise Applications', 'Web Applications', 'EJB Modules', 'Connector Modules', 'Lifecycle Modules', 'Application Client Modules'), and 'Web Services'. The main content area shows 'Applications > Web Applications' with a 'Web Applications' section. It describes a Web application module as a collection of Web resources like JSPs, servlets, and HTML pages. Below this is a table titled 'Deployed Web Applications (1)'. The table has columns for 'Name', 'Enabled', 'Context Root', and 'Action'. A single entry is listed: 'ApplicationWebControleeRealmsGlassfish' with 'true' in the 'Enabled' column, '/ApplicationWebControleeRealmsGlassfish' in the 'Context Root' column, and 'Launch' and 'Redeploy' buttons in the 'Action' column.

Bien sûr, n'importe qui peut accéder aux précieuses informations 😊 - donc contrôlons l'accès à celles-ci ☺ ...

8. Les realms JDBC sous GlassFish

8.1 Divers gestionnaire d'authentification

On se souviendra qu'un "**realm**" ("royaume" en anglais) est un **gestionnaire d'authentification** que le serveur utilise lors de chaque opération de reconnaissance d'un utilisateur autorisé. GlassFish connaît 5 types de realms :

- ◆ le FileRealm : basique, l'idée est que les informations de connexion sont stockées dans un simple fichier;
- ◆ le CertificateRealm : classique, l'idée est qu'un utilisateur fournit un certificat pour s'authentifier;
- ◆ le JDBCRealm : classique aussi, les informations de connexion sont stockées dans une base de données – nous allons développer cette manière de faire ci-dessous;
- ◆ le LDAPRealm : pratique, on utilise un annuaire LDAP pour retrouver les informations de connexion;
- ◆ le SolarisRealm : particulier puisque ce sont les comptes utilisateurs Solaris qui tiennent lieu d'informations de connexion.

8.2 Groupes et rôles

On se souviendra qu'un "**role**" est un groupe d'utilisateurs. GlassFish va utiliser une notion supplémentaire : le **groupe**. L'objectif est d'éviter les répétitions de noms d'utilisateurs. En effet, on classe d'abord les utilisateurs en groupes, par exemple :

- ◆ groupe Enseignants = { Vilvens, Wagner, Mercenier, Charlet, Madani, Kuty }
- ◆ groupe Administratifs = { Paggen, Nicolas, Lambrechts, Gobin }

Ensuite, on détermine à quelles tâches ces groupes peuvent participer et on définit les roles :

- ◆ role Cotes = { Enseignants, Administratifs }
- ◆ role DocumentsEtudiants = { Administratifs }
- ◆ role Cours = { Enseignants }

Sans les groupes, il faudrait répéter les noms concernés plusieurs fois.

8.3 La base de données du JDBC Realm

Tout comme pour Tomcat, nous utiliserons une base MySQL et elle sera créée au plus simple : une table pour les utilisateurs, une autre pour mémoriser la relation utilisateur-groupe. La structure suivante a été utilisée, parfaitement analogue à celle utilisée pour Tomcat si ce n'est que le groupe remplace le rôle :

gutilisateurs	
PK	unom
	umotdepasse
	nom
	prénom

ggroupes	
PK	unom
	gnom

En pratique, cela donne :

```
mysql> create database groupesGlassFish;
Query OK, 1 row affected (0.03 sec)
mysql> use groupesGlassFish;
Database changed
mysql> show tables;
Empty set (0.02 sec)
mysql>
mysql> create table gutilisateurs (unom varchar(15) not null primary key, umotdepasse
varchar(15) not null, nom varchar(30), prenom varchar(30));
Query OK, 0 rows affected (0.16 sec)
mysql> create table ggroupes ( unom varchar(15) not null, gnom varchar(15) not null,
primary key(unom, gnom) );
Query OK, 0 rows affected (0.08 sec)

mysql> insert into gutilisateurs values ("vilvens", "genius", "Vilvens", "Claude");
Query OK, 1 row affected (0.06 sec)
mysql> insert into gutilisateurs values ("wagner", "groscerveau", "Wagner", "Jean-Marc");
Query OK, 1 row affected (0.03 sec)
mysql> select * from gutilisateurs;
+-----+-----+-----+-----+
| unom | umotdepasse | nom | prenom |
+-----+-----+-----+-----+
| vilvens | genius | Vilvens | Claude |
| wagner | groscerveau | Wagner | Jean-Marc |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> insert into ggroupes values ("vilvens", "Enseignants");
Query OK, 1 row affected (0.03 sec)
mysql> insert into ggroupes values ("wagner ", "Enseignants");
Query OK, 1 row affected (0.03 sec)
mysql> select * from ggroupes;
+-----+-----+
| unom | gnom |
+-----+-----+
| vilvens | Enseignants |
| wagner | Enseignants |
+-----+-----+
2 rows in set (0.00 sec)
```

C'est donc sur base de ces tables que les accès vont être testés. Mais cette fois, au lieu d'aller chercher les données avec un **DriverManager**, nous allons les définir comme formant une source de données au sens des **DataSource** (le niveau d'abstraction le plus élevé des données, avec utilisation du Java Naming and Directory Interface - JNDI), source de données qui utilisera une **PooledConnection** afin de privilégier les performances..

8.4 La source de données d'authentification

Nous allons donc sur la console admin pour créer un connection pool dont le nom sera associé à un nom JNDI que le realm accèdera :

JNDI Name	Resource Type	Datasource Classname	Description
DerbyPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
SamplePool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
CallFlowPool	javax.sql.XADatasource	org.apache.derby.jdbc.EmbeddedXADataSource	
TimerPool	javax.sql.XADatasource	org.apache.derby.jdbc.EmbeddedXADataSource	

Nous créons donc une DataSource nommée MySqlPool :

The screenshot shows the "New JDBC Connection Pool (Step 1 of 2)" configuration page. The left sidebar lists "Common Tasks" and "Resources > JDBC > Connection Pools". The main panel displays the following settings:

- Name:** MySqlPool
- Resource Type:** javax.sql.DataSource
- Database Vendor:** MySQL

Buttons at the top right include "Next" and "Cancel".

The screenshot shows the "New JDBC Connection Pool (Step 2 of 2)" configuration page. The left sidebar lists "Common Tasks" and "Resources > JDBC > Connection Pools". The main panel displays the following settings:

- Name:** MySqlPool
- Resource Type:** javax.sql.DataSource
- Database Vendor:** MySQL
- Datasource Classname:** com.mysql.jdbc.jdbc2.optional.MysqlDataSource
- Description:** (empty)

Below these, the "Pool Settings" section includes:

- Initial and Minimum Pool Size:** 8 Connections
- Maximum Pool Size:** 32 Connections
- Pool Resize Quantity:** 2 Connections
- Idle Timeout:** 300 Seconds
- Max Wait Time:** 60000 Milliseconds

Buttons at the top right include "Previous", "Finish", and "Cancel".

On peut ensuite créer les différentes propriétés logiquement nécessaires, comme le nom de la base de données, le nom d'utilisateur, son mot de passe, etc. On veillera à ne pas oublier la propriété "networkProtocol = TCP" (ou la créer si elle n'existe pas).

The screenshot shows the 'Transaction' configuration page in the Admin Console. On the left, a tree view lists 'Applications', 'Web Services', 'JBI', and 'Resources'. Under 'Resources', 'JDBC' is expanded, showing 'JDBC Resources' and 'Connection Pools'. 'Connection Pools' is selected. A large table titled 'Additional Properties (8)' displays properties for the selected pool, including 'databaseName' (groupesglassfish), 'networkProtocol' (TCP), 'password' (Monodonta113.), 'portNumber' (3306), 'serverName' (root), and 'user' (root). Buttons at the bottom right of the table area include 'Previous', 'Finish', and 'Cancel'.

Un appui sur Finish achève la création de la source de données :

The screenshot shows the 'Pools (5)' list. The table has columns: JNDI Name, Resource Type, Datasource Classname, and Description. The data is as follows:

JNDI Name	Resource Type	Datasource Classname	Description
DerbyPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
SamplePool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
_CallFlowPool	javax.sql.XADatasource	org.apache.derby.jdbc.EmbeddedXADataSource	
MySQLPool	javax.sql.DataSource	com.mysql.jdbc.jdbc2.optional.MysqlDataSource	
_TimerPool	javax.sql.XADatasource	org.apache.derby.jdbc.EmbeddedXADataSource	

Pour que le serveur GlassFish soit capable d'atteindre notre base de données MySQL, il aura besoin des drivers JDBC de MySQL : nous allons donc copier le jar de ces drivers MySQL dans C:\GlassFish-v2ur1\domains\domain1\lib\ext. Evidemment, il faudra redémarrer le serveur pour que les modifications apportées soient prises en compte. On peut alors retourner dans la console admin et tester la source de données avec le bouton **Ping** :

The screenshot shows the 'Edit Connection Pool' page for a MySQL connection pool named 'MySQLPool'. The 'General' tab is selected. A yellow button at the top right says 'Ping Succeeded'. The 'Datasource Classname' field contains 'com.mysql.jdbc.jdbc2.optional.MysqlDataSource'. The 'Resource Type' field is set to 'javax.sql.DataSource'. The 'Initial and Minimum Pool Size' is set to 8 connections. The left sidebar shows the navigation tree with 'JDBC Resources' selected under 'Resources'.

Cette source de données est connue du serveur GlassFish, mais pas encore des applications hébergées par le serveur d'application (et de notre futur realm) : pour cela, il faut associer à la source de données un nom JNDI. Nous retournons donc dans la console d'administration, sur le nœud "JDBC Ressources" où nous allons créer une nouvelle ressource :

The screenshot shows the 'JDBC Resources' page with four resources listed in a table:

JNDI Name	Enabled	Connection Pool	Description
jdbc/_CallFlowPool	true	_CallFlowPool	
jdbc/_TimerPool	true	_TimerPool	
jdbc/sample	true	SamplePool	
jdbc/_default	true	DerbyPool	

The left sidebar shows the navigation tree with 'JDBC Resources' selected under 'JDBC'.

L'appui sur New donne :

The screenshot shows the 'New JDBC Resource' dialog box. The left sidebar navigation tree is expanded to show 'JDBC Resources' under 'Resources > JDBC'. The main panel displays fields for creating a new JDBC resource:

- JNDI Name:** * `jdbc/mysql`
- Pool Name:** * `MySqlPool`
- Description:** `DataSource sur base de donnees MySql groupesglassfish`
- Status:** Enabled

Buttons at the top right include 'OK' and 'Cancel'.

Le nom JNDI qui sera employé par le realm sera, par convention, "jdbc/mysql". L'appui sur Ok donne :

Resources (5)				
	JNDI Name	Enabled	Connection Pool	Description
<input type="checkbox"/>	jdbc/mysql	true	MySqlPool	DataSource sur base de donnees MySql groupesglassfish
<input type="checkbox"/>	jdbc/_CallFlowPool	true	_CallFlowPool	
<input type="checkbox"/>	jdbc/_TimerPool	true	_TimerPool	
<input type="checkbox"/>	jdbc/sample	true	SamplePool	
<input type="checkbox"/>	jdbc/_default	true	DerbyPool	

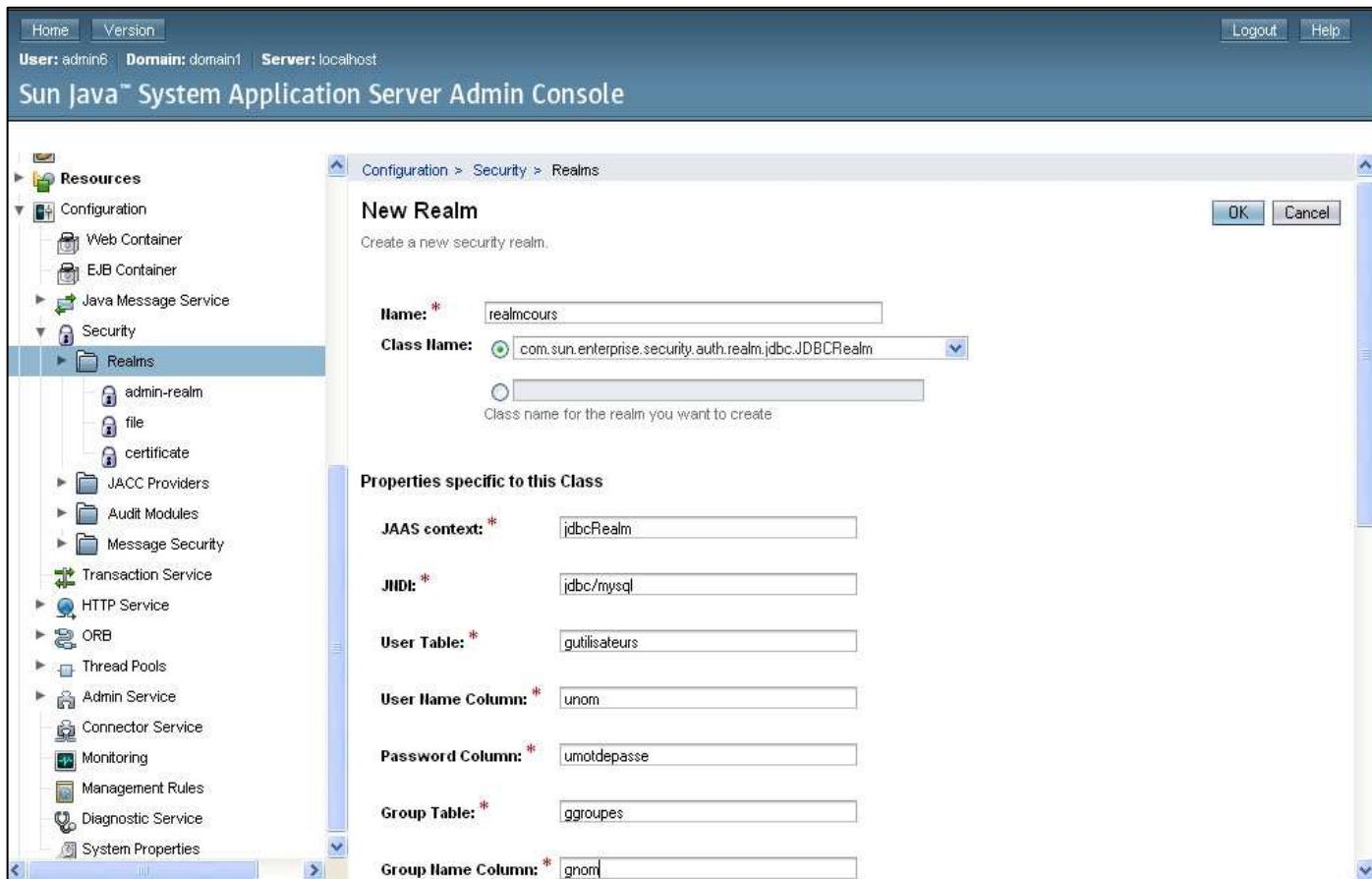
8.5 La création du realm

Un realm se définit sur la console d'administration sur le nœud

Config→Security→Realm :

The screenshot shows the 'Realms' configuration page. The left sidebar navigation tree is expanded to show 'Realms' under 'Configuration > Security'. The main panel displays the 'Realms (3)' table:

Name	Classname
file	com.sun.enterprise.security.auth.realm.file.FileRealm
admin-realm	com.sun.enterprise.security.auth.realm.file.FileRealm
certificate	com.sun.enterprise.security.auth.realm.certificate.CertificateRealm



Il faut être attentif et positionner le champ "digest-algorithm" à none, sous peine d'une future erreur d'accès à la base de données :

Digest Algorithm:

Un appui sur Ok et voilà :

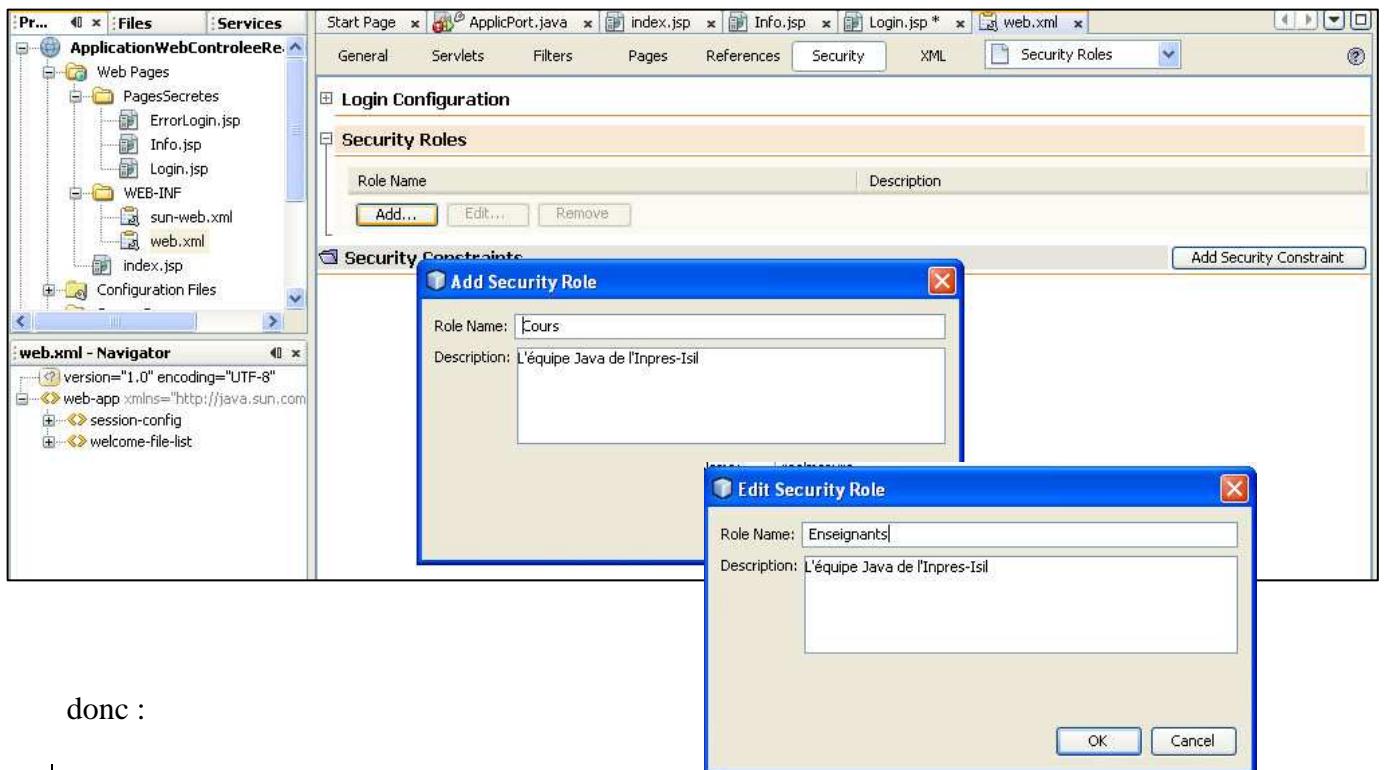
Realms (4)	
	New... Delete
<input type="checkbox"/>	Name
<input type="checkbox"/>	Classname
<input type="checkbox"/>	file com.sun.enterprise.security.auth.realm.file.FileRealm
<input type="checkbox"/>	realmcours com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm
<input type="checkbox"/>	admin-realm com.sun.enterprise.security.auth.realm.file.FileRealm
<input type="checkbox"/>	certificate com.sun.enterprise.security.auth.realm.certificate.CertificateRealm

8.6 La configuration de l'application Web

Notre application Web va maintenant devoir prendre conscience de l'existence du realm. Comme pour Tomcat, les données envoyées par le JSP de départ seront traitées par une servlet du serveur nommée **j_security_check** et qui attend deux paramètres aux noms prédefinis : **j_username** et **j_password**. Ceci nous force à créer le formulaire de cette manière au sein du JSP – en fait, nous pouvons reprendre les deux Login.jsp et Error-Login.jsp qui nous ont servi pour Tomcat. Reste alors à configurer les fichiers xml habituels :

1) web.xml

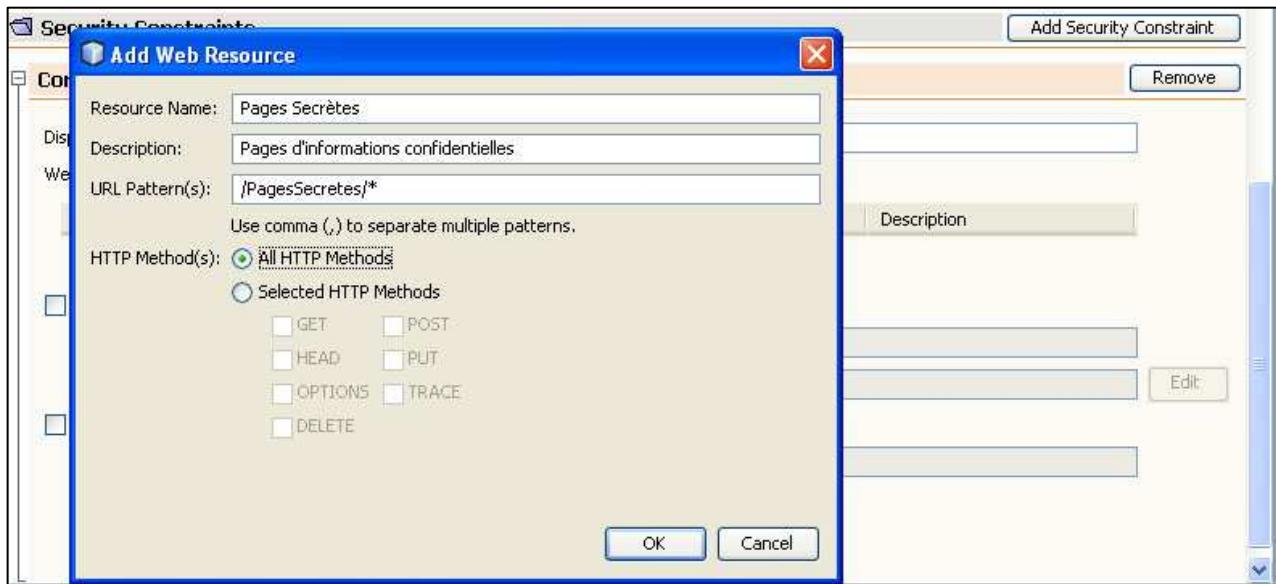
a) définir le rôle :



donc :

```
<security-role>
    <description>L'équipe Java de l'Inpres-Isil</description>
    <role-name>Cours</role-name>
</security-role>
```

et définir une contrainte de sécurité :



The screenshot shows the GlassFish Admin Console interface for managing security constraints. A modal dialog titled "Edit Role Names" is open, allowing the selection of roles to be associated with the constraint. The role "Enseignants" is currently selected and highlighted.

Name	URL Pattern	HTTP Method	Description
Pages Secrètes	/PagesSecretes/*	GET, POST, HEAD, PUT, OPTIO...	Pages d'informations confidentielles

Enable Authentication Constraint

Description:

Role Name(s): Edit

donc finalement :

The screenshot shows the final configuration of the security constraint "ContrainteCours". The role "Enseignants" has been successfully added to the constraint.

Name	URL Pattern	HTTP Method	Description
Pages Secrètes	/PagesSecretes/*	GET, POST, HEAD, PUT, OPTIO...	Pages d'informations confidentielles

Enable Authentication Constraint

Description:

Role Name(s): Enseignants Edit

Enable User Data Constraint

Description:

Transport Guarantee:

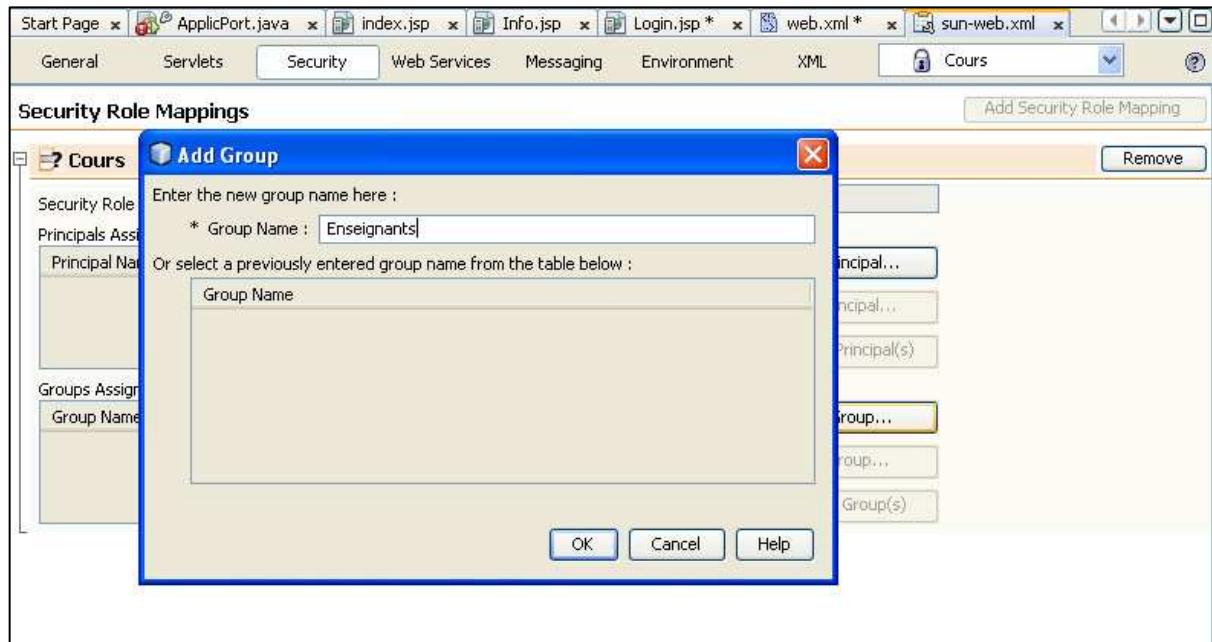
Le fichier web.xml aura donc finalement l'aspect suivant :

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<security-constraint>
  <display-name>ContrainteCours</display-name>
  <web-resource-collection>
    <web-resource-name>Pages Secrètes</web-resource-name>
    <description>Pages d'informations confidentielles</description>
    <url-pattern>/PagesSecretes/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>HEAD</http-method>
    <http-method>PUT</http-method>
    <http-method>OPTIONS</http-method>
    <http-method>TRACE</http-method>
    <http-method>DELETE</http-method>
  </web-resource-collection>
  <auth-constraint>
    <description/>
    <role-name>Enseignants</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>realmcours</realm-name>
  <form-login-config>
    <form-login-page>/PagesSecretes/Login.jsp</form-login-page>
    <form-error-page>/PagesSecretes/ErrorLogin.jsp</form-error-page>
  </form-login-config>
</login-config>
<security-role>
  <description>L'équipe Java de l'Inpres-Isil</description>
  <role-name>Enseignants</role-name>
</security-role>
</web-app>
```

2) sun-web.xml

Il reste à établir la relation entre le rôle et le groupe :

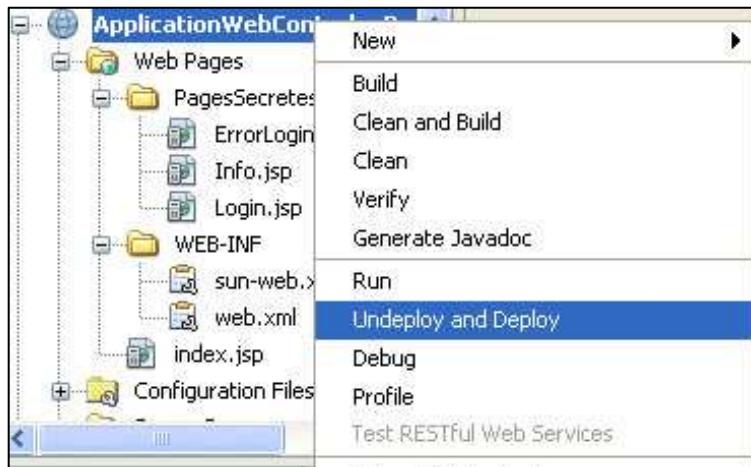


et le fichier sun-web.xml aura l'aspect suivant :

sun-web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server
9.0 Servlet 2.5//EN" "http://www.sun.com/software/appserver/dtds/sun-web-app_2_5-
0.dtd">
<sun-web-app error-url="">
<context-root>/ApplicationWebControleeRealmsGlassFish</context-root>
<security-role-mapping>
<role-name>Cours</role-name>
<group-name>Enseignants</group-name>
</security-role-mapping
```

Il n'y a plus qu'à redéployer l'application :



| >>> BUILD SUCCESSFUL (total time: 1 second)

8.7 Le test de l'application

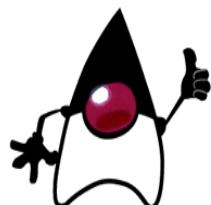
Notre page index, bien qu'elle veuille conduire aux informations, devra passer par la page de login :

The image contains two screenshots of Mozilla Firefox windows. The top window is titled 'Accueil - Mozilla Firefox' and displays a simple web page with the text 'Bienvenue !' and a link 'Accès aux informations ? Par ici ...'. The bottom window is titled 'Login préalable - qui es-tu ? - Mozilla Firefox' and displays a page with the text 'Hello World!' followed by the message 'Bienvenue ! Veuillez vous identifier'. Below this, there is a login form with fields for 'Votre nom d'utilisateur' (containing 'wagner') and 'Votre mot de passe' (containing redacted text). A large 'OK' button is at the bottom of the form. Both windows show the URL 'http://localhost:8080/ApplicationWebControleeRealmsGlassfish/' in the address bar.

Si les informations entrées sont correctes :

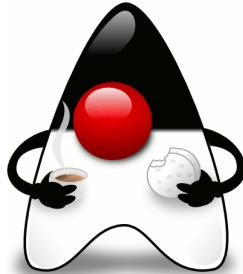


sinon :



Il y a des milliers d'autres choses à raconter sur Glasfish, qui est serveur utilisant JMX, les MBeans et JAAS (par exemple). Tiens, justement, JAAS ? Qu'est-ce donc ?

XXXVI. Java Authentication and Authorization Service (JAAS)



Partager ses connaissances est une manière d'obtenir l'immortalité.

(Tantra indien)

1. Le Java Authentication and Authorization Service

Nous nous intéressons ici à une librairie Java qui est en fait une implémentation du framework **PAM** (**Pluggable Authentication Module**) : il s'agit d'un mécanisme créé par l'ancien Sun Microsystems permettant d'intégrer différents schémas d'authentification de bas niveau dans une API de haut niveau, permettant de ce fait de rendre indépendants du schéma d'authentification/permissions les logiciels réclamant une authentification.

JAAS (**Java Authentication and Authorization Service**) vise donc deux objectifs que l'on peut résumer par "*authentication and authorization*" :

- ◆ permettre l'authentification des utilisateurs afin de savoir très précisément qui exécute un code Java donné (qu'il s'agisse d'une application, d'un applet ou d'une servlet);
- ◆ vérifier les permissions et droits d'accès dont ces utilisateurs ont besoin pour exécuter ce code.

Bien sûr, nous savons déjà que Java dispose d'un mécanisme de sécurité contrôlant les permissions d'un code en fonction de son origine et de sa signature (l'**AccessController** utilisant le **ProtectionDomain** du code – voir Java IV – "Le modèle de sécurité Java") : c'est l'aspect "**authorization**" de JAAS et son point d'entrée est constitué par la paire **SecurityManager** et **AccessController**. Mais ce mécanisme ne se préoccupe pas d'authentification, autrement dit de savoir "qui" exécute le code : c'est l'aspect "**authentication**" que nous allons à présent investiguer.

2. Les configurations d'authentification

Puisqu'il s'agit d'une implémentation du PAM, JAAS est construit de manière suffisamment modulaire pour rester indépendant du mécanisme d'authentification : on peut donc remplacer une authentification basée, par exemple, sur le binôme user-password par la reconnaissance d'une empreinte digitale ou l'usage d'un certificat. Ceci est possible parce que les applications utilisant JAAS utilisent en fait un **LoginContext** qui référence un objet **Configuration**, qui définit le type d'authentification en désignant le(s) objet(s) **LoginModule** qui réalise(nt) cette authentification (ils peuvent donc être plusieurs, qui agissent en cascade). Typiquement, le **LoginContext** se procure la configuration et y découvre les **LoginModule** au moyen des instructions :

```
Configuration config = Configuration.getConfiguration();
AppConfigurationEntry entries = config.getAppConfigurationEntry(appName);
```

Ce que contient l'objet configuration (issu d'un fichier par exemple) respecte la syntaxe :

```
Application1 {
    <nom complet de la classe du module> <flag> <paramètres divers du module>;
    <nom complet de la classe du module> <flag> <paramètres divers du module>;
    <nom complet de la classe du module> <flag> <paramètres divers du module>;
    ...
};

Application2 {
    <nom complet de la classe du module> <flag> <paramètres divers du module>;
    <nom complet de la classe du module> <flag> <paramètres divers du module>;
    ...
};

other {
    <nom complet de la classe du module> <flag> <paramètres divers du module>;
    <nom complet de la classe du module> <flag> <paramètres divers du module>;
    ...
};
```

Par exemple, on pourra trouver dans un fichier de configuration :

JAASNom.config

```
JAASNom {
    jaasbasics.module.JAASNomLoginModule required debug=true;
};
```

ou encore (mais nous y reviendrons)

JAASKerberos.config

```
Login {
    com.sun.security.auth.module.UnixLoginModule required;
    com.sun.security.auth.module.Krb5LoginModule optional
        useTicketCache="true"
        ticketCache="${user.home}${{/}tickets}";
};
```

Les flags utilisables sont :

- ◆ Required : Le LoginModule est obligatoire pour permettre l'authentification, mais dans tous les cas (succès ou échec), le processus d'authentification continue en suivant la liste des autres LoginModule.
- ◆ Requisite : Comme Required, mais un échec arrête le processus.
- ◆ Optional : Le LoginModule est facultatif pour permettre l'authentification, mais dans tous les cas (succès ou échec), le processus d'authentification continue en suivant la liste des autres LoginModule.
- ◆ Sufficient : Comme Optional, mais un échec arrête le processus.

3. Les entités d'authentification et d'autorisation

3.1 Les subjects

On désigne par le vocable "***subject***" l'entité qui souhaite obtenir un accès à une ressource. Il peut donc s'agir d'une personne ou d'un service, plus généralement d'un groupe d'informations reliées entre elles. C'est le rôle de la classe **Subject** (du package javax.security.auth) de matérialiser un tel demandeur.

3.2 Les principals

Un subject peut être identifié de plusieurs manières : ainsi, une personne peut être identifiée par

- ◆ ses nom et prénom
- ◆ son numéro de carte d'identité
- ◆ son numéro de sécurité sociale
- ◆ son acte de naissance

Chacune de ces différentes identités est désignée par le vocable de "***principal***" et un interface **Principal** (du package java.security) devra être implémenté pour matérialiser une telle identité.

3.3 Les credentials

Chaque subject peut aussi avoir divers attributs de sécurité, comme par exemple :

- ◆ une clé privée;
- ◆ un certificat;
- ◆ un ticket Kerberos pour un serveur donné.

Ces attributs sont désignés par le terme "***credential***". Certains sont clairement public, d'autres privés et réclament des permissions pour être accédés. *Il n'existe pas de classe Credential, ni même d'interface* : la nature éminemment variable de ces attributs de sécurité peut sans doute le justifier. Ils sont simplement comptabilisés dans un Set<Object>.

La classe Subject implémente Serializable : quand on sérialise un tel objet, les Principal associés le sont aussi (encore que l'interface Principal n'hérite pas de Serializable et que ce sont les classes qui l'implémentent qui doivent le faire).

3.4 Utilisation des Subjects, Principals et Credentials

On sait que l'AccessController a pour mission d'autoriser ou non l'exécution d'un code sur base de permissions soumises, le plus souvent, à la vérification d'une signature. Implicitement, le subject est alors l'utilisateur courant et le seul credential utilisé est la clé publique du signataire. Mais il est possible aussi d'***associer dynamiquement un subject authentifié à un contexte de contrôle*** : JAAS fournit en effet deux méthodes doAs() and doAsPrivileged() qui relient le subject passé en argument soit au contexte du thread courant (doAs) ou à un contexte précis (doAsPrivileged). Plus précisément, les deux méthodes suivantes permettent de soumettre l'exécution d'un code au fait que c'est un subject authentifié qui le demande :

- ◆ public static Object doAs(final Subject subject, final PrivilegedAction action)

- à remarquer pour pour appeler cette méthode, il faut disposer de la permission AuthPermission("doAs").

♦ public static Object **doAsPrivileged**(final Subject subject, final PrivilegedAction action, final AccessControlContext acc)

ou **PrivilegedAction** est un interface permettant de definir le code à exécuter, puisqu'il déclare la seule méthode run() :

PrivilegedAction.java

```
package java.security;

public interface PrivilegedAction
{
    public abstract Object run();
}
```

4. Le processus d'authentification selon JAAS

4.1 La démarche globale

La démarche d'authentification d'un sujet selon JAAS est la suivante :

1. on instancie un LoginContext;
2. ce LoginContext consulte un objet Configuration afin de trouver des LoginModule et les charge en mémoire;
3. on appelle la méthode login() du LoginContext;
4. celle-ci se réfère aux LoginModule en mémoire et cherche des principals en rapport avec le subject courant et, en cas de succès, ajoute les credentials associés au subject;
5. le LoginContext évalue ces credentials et retourne sa conclusion à l'application;
6. en cas de succès, l'application travaille comme étant le subject.

4.2 Le point d'entrée : le LoginContext

La classe **LoginContext** (du package javax.security.auth.login) est, on l'a dit, à la partie Authentication de JAAS ce que SecurityManager est à la partie Authorization. En substance, son code est le suivant :

LoginContext.java

```
package javax.security.auth.login;
...

public class LoginContext
{
    private static final String INIT_METHOD = "initialize";
    private static final String LOGIN_METHOD = "login";
    private static final String COMMIT_METHOD = "commit";
    private static final String ABORT_METHOD = "abort";
    private static final String LOGOUT_METHOD = "logout";
    private static final String OTHER = "other";
    private static final String DEFAULT_HANDLER = "auth.login.defaultCallbackHandler";
```

```
private Subject subject;
private CallbackHandler callbackHandler;
private Map state;
private Configuration config;
...
private void init(String s) throws LoginException
{
    SecurityManager securitymanager = System.getSecurityManager();
    if(securitymanager != null && !configProvided)
    {
        securitymanager.checkPermission( new AuthPermission
            ((new StringBuilder()).append("createLoginContext.").append(s).toString()) );
        ...
    }
}

public LoginContext(String s) throws LoginException { ... }
public LoginContext(String s, Subject subject1) throws LoginException { ... }
public LoginContext(String s, CallbackHandler callbackhandler)
    throws LoginException
{
    state = new HashMap();
    if(callbackhandler == null)
    { throw new LoginException(ResourcesMgr.getString("invalid null CallbackHandler
        provided")); }
    else
    { callbackHandler = new SecureCallbackHandler(AccessController.getContext(),
        callbackhandler);
        return;
    }
}
public LoginContext(String s, Subject subject1, CallbackHandler callbackhandler)
    throws LoginException
{ .. }
public LoginContext(String s, Subject subject1, CallbackHandler callbackhandler,
    Configuration configuration) throws LoginException
{ .... }

public void login() throws LoginException
{
    loginSucceeded = false;
    if(subject == null)
    {
        subject = new Subject();
    }
...
}
```

La classe `LoginContext` possède donc plusieurs constructeurs qui réclament tous au minimum un nom qui représente l'entrée dans l'objet configuration. Parmi ces constructeurs, le plus naturel est :

```
public LoginContext(String name, Subject subject) throws LoginException
```

le deuxième paramètre étant évidemment le subject à authentifier. Mais si l'authentification nécessite une interaction avec l'utilisateur (entrée d'un mot de passe ou d'un code PIN, par exemple), on utilisera plutôt :

```
public LoginContext(String name, CallbackHandler callbackHandler) throws LoginException
```

où l'interface **CallbackHandler** (du package `javax.security.auth.callback`) déclare la seule méthode :

```
void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException
```

L'interface **Callbak** (même package) caractérise un objet dont le rôle est de transmettre à l'application une requête visant à l'authentification (par exemple, l'entrée du mot de passe) et de récupérer la réponse. On peut remarquer que

- ◆ l'utilisation de ces `CallbackHandler` permet d'ajuster la forme de dialogue au type d'application : ainsi, si l'application est fenêtrée, le `CallbackHandler` fera sortir une boîte de dialogue pour demander le mot de passe tandis que si elle est en ligne de commande, il se contentera d'une entrée clavier;
- ◆ un tableau d'objets `Callback` est prévu : ainsi, on peut utiliser des classes de la librairie comme **NameCallback** ou **PasswordCallback** (package `javax.security.auth.callback`), le `Callbackhandler` orchestrant leur utilisation.

Les objets `Callback` seront construits dans les `LoginModule`, puisqu'ils dépendent clairement de la nature et de la procédure d'authentification. Le `LoginContext` découvre dans le fichier de configuration la nature de ce `LoginModule` à instancier – on travaille donc avec les outils d'introspection :

LoginContext.java (complément)

```
private void invoke(String s) throws LoginException
{
    ...
    Class class1 = Class.forName(moduleStack[i].entry.getLoginModuleName(), true,
        contextClassLoader);
    Constructor constructor = class1.getConstructor(PARAMS);
    Object aobj2[] = new Object[0];
    moduleStack[i].module = constructor.newInstance(aobj2);
    amethod = moduleStack[i].module.getClass().getMethods();
    for(j = 0; j < amethod.length && !amethod[j].getName().equals("initialize"); j++) { }
    Object aobj3[] = {
        subject, callbackHandler, state, moduleStack[i].entry.getOptions() }
    ...
}
```

Dans notre cas, il recherché donc un JAASNom.config qui, pour rappel, contient :

```
JAASNom {  
    jaasbasics.module.JAASNomLoginModule required debug=true;  
};
```

Cependant, pour retrouver ce fichier, la JVM se sert du (des) chemin(s) indiqués dans le fichier java.security du JRE au moyen de la clause login.config.url, qu'il convient donc d'adapter :

java.security
<pre>... # # Default login configuration file # login.config.url.1=file:\${user.home}/JAASNom.config #login.config.url.1=file:///C:/java-netbeans-application/JAASBasics/JAASNom.config ...</pre>

Avant d'entrer dans ce LoginModule, mettons en place l'application qui va nous servir d'exemple.

4.3 Utilisation du LoginContext dans une application

Nous allons donc ici imaginer une simple application qui, dans un premier temps, va provoquer la mise en route du processus d'authentification – ici, ce sera sur la base classique d'un nom et d'un mot de passe. Elle peut s'écrire de la manière suivante :

JAASNom.java
<pre>package jaasbasics; import java.io.*; import java.util.*; import javax.security.auth.login.*; import javax.security.auth.callback.*; public class JAASNom { private static int NOMBRE_MAXIMUM_ESSAIS = 3; public static void main(String[] args) { LoginContext lc = null; try { lc = new LoginContext("JAASNom", new <i>MyCallbackHandler</i>()); // voir plus loin } catch (LoginException le) { System.err.println("LoginContext non créé. LoginException. "+ le.getMessage()); } } }</pre>

```

        System.exit(-1);
    }
    catch (SecurityException se)
    {
        System.err.println("LoginContext non créé. SecurityException." + se.getMessage());
        System.exit(-2);
    }

    // Essais successifs d'authentification
    int i;
    for (i = 0; i < NOMBRE_MAXIMUM_ESSAIS; i++)
    {
        try
        {
            lc.login();
            break;
        }
        catch (LoginException le)
        {
            System.err.println("Authentification ratée :-(");
            System.err.println(" " + le.getMessage());
            try { Thread.currentThread().sleep(3000); }
            catch (Exception e) { System.err.println(" " + e.getMessage()); }
        }
    }

    if (i == 3)
    {
        System.out.println("Désolé : on en reste là ...");
        System.exit(-3);
    }
    System.err.println("Authentification réussie :-)");
}
}

```

4.4 Le CallbackHandler

On l'a dit aussi, *le CallbackHandler est à implémenter pour une application donnée* : son rôle est de retrouver les données d'authentification et d'afficher les messages résultant des tentatives d'authentification de l'utilisateur de l'application. En fait, ce que le CallbackHandler doit faire par rapport à la politique d'authentification choisie lui est signifié en lui passant des objets Callback (par exemple, dans notre cas, des objets NameCallback and PasswordCallback). Il prendra connaissance de sa liste de tâches dans sa seule méthode

`void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException`

en scannant le tableau passé en paramètre : il reconnaît le type des différents Callbacks et exécute les tâches déterminées par les méthodes de ceux-ci. Les Callbacks possibles sont :

AuthorizeCallback, ChoiceCallback, ConfirmationCallback, LanguageCallback,
NameCallback, PasswordCallback, RealmCallback, RealmChoiceCallback,
TextInputCallback, TextOutputCallback

Par exemple, un **NameCallback** doit posséder les méthodes

```
String getDefaultName()  
String getName()  
String getPrompt()  
void setName(String name)
```

- son implémentation étant :

NameCallback.java

```
package javax.security.auth.callback;  
  
import java.io.Serializable;  
  
public class NameCallback implements Callback, Serializable  
{  
    private static final long serialVersionUID = 0x345510db07eb277dL;  
    private String prompt;  
    private String defaultValue;  
    private String inputName;  
  
    public NameCallback(String s)  
    {  
        if(s == null || s.length() == 0) { throw new IllegalArgumentException(); }  
        else  
        { prompt = s; return; }  
    }  
  
    public NameCallback(String s, String s1)  
    {  
        if(s == null || s.length() == 0 || s1 == null || s1.length() == 0)  
        { throw new IllegalArgumentException(); }  
        else  
        { prompt = s; defaultValue = s1; return; }  
    }  
  
    public String getPrompt() { return prompt; }  
    public String getDefaultName() { return defaultValue; }  
    public void setName(String s) { inputName = s; }  
    public String getName() { return inputName; }  
}
```

Autre exemple : un TextOutputCallback dispose des méthodes

```
String getMessage()
```

```
int getMessageType()
```

cette dernière faisant allusion aux constantes :

```
static int      ERROR
static int      INFORMATION
static int      WARNING
```

Dans notre cas, puisque le but est d'introduire un nom et un mot de passe :

MyCallbackHandler.java

```
package jaasbasics;

...
import javax.security.auth.callback.*;

public class JAASNom
{ ... }

class MyCallbackHandler implements CallbackHandler
{
    public void handle(Callback[] callbacks) throws IOException,
                                               UnsupportedCallbackException
    {
        for (int i = 0; i < callbacks.length; i++)
        {
            if (callbacks[i] instanceof TextOutputCallback)
            {
                TextOutputCallback toc = (TextOutputCallback)callbacks[i];
                switch(toc.get MessageType())
                {
                    case TextOutputCallback.INFORMATION:
                        System.out.println(toc.getMessage());
                        break;
                    case TextOutputCallback.ERROR:
                        System.out.println("ERROR: " + toc.getMessage());
                        break;
                    case TextOutputCallback.WARNING:
                        System.out.println("WARNING: " + toc.getMessage());
                        break;
                    default:
                        throw new IOException("Type de message non supporté: " +
                                              toc.getMessageType());
                }
            }
            else if (callbacks[i] instanceof NameCallback)
            {
                NameCallback nc = (NameCallback)callbacks[i];
                System.out.print(nc.getPrompt());
            }
        }
    }
}
```

```
        System.err.flush();
        nc.setName((new BufferedReader (
            new InputStreamReader(System.in))).readLine()));

    }
    else if (callbacks[i] instanceof PasswordCallback)
    {
        PasswordCallback pc = (PasswordCallback)callbacks[i];
        System.err.print(pc.getPrompt());
        System.err.flush();
        String pwd = (new BufferedReader (
            new InputStreamReader(System.in))).readLine();
        char[] pwdChar = new char[pwd.length()];
        pwd.getChars(0,pwd.length(), pwdChar,0);
        pc.setPassword(pwdChar);

    }
    else
    {
        throw new UnsupportedCallbackException (callbacks[i], "Callback inconnu");
    }
}
}
```

Mais qui va créer les Callbacks ? Le LoginModule, dans sa méthode login(), laquelle sera appelée par le login() du LoginContext, qui lui transmettra le CallbackHandler pour qu'il reçoive ces Callbacks. Dans la foulée, en cas de succès, la méthode login() du LoginContext appelle aussi la méthode commit() du LoginModule, dont le rôle est classiquement d'ajouter au Subject le Principal créé à partir du nom de l'utilisateur.

4.5 Le LoginModule

L'interface LoginModule se résume à ceci :

LoginModule.java

```
package javax.security.auth.spi;

import java.util.Map;
import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginException;

public interface LoginModule
{
    public abstract void initialize(Subject subject, CallbackHandler callbackhandler,
        Map map, Map map1);
    /* subject - the Subject to be authenticated.
       callbackHandler - a CallbackHandler for communicating with the end user
       (prompting for usernames and passwords, for example).
}
```

```

sharedState - state shared with other configured LoginModules.
options - options specified in the login Configuration for this particular LoginModule.
*/
public abstract boolean login() throws LoginException;
    /* Authenticate the user by prompting for a user name and password */
public abstract boolean commit() throws LoginException;
    /* This method is called if the LoginContext's overall authentication succeeded
     (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL
     LoginModules succeeded)
*/
public abstract boolean abort() throws LoginException;
    /* This method is called if the LoginContext's overall authentication failed
     (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL
     LoginModules did not succeed).
*/
public abstract boolean logout() throws LoginException;
    /* This method removes the <code>SamplePrincipal</code> that was added by the commit
*/
}

```

et une implémentation de LoginModule (la classique que l'on trouve partout) est du style :

JAASNomLoginModule.java

```

package jaasbasics.module;

import java.util.*;
import java.io.IOException;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;
import jaasbasics.principal.JAASNomPrincipal;

public class JAASNomLoginModule implements LoginModule
{
    // initial state
    private Subject subject;
    private CallbackHandler callbackHandler;
    private Map sharedState;
    private Map options;

    // configurable option
    private boolean debug = false;

    // the authentication status
    private boolean succeeded = false;
    private boolean commitSucceeded = false;

    // username and password

```

```
private String username;
private char[] password;

// testUser's SamplePrincipal
private JAASNomPrincipal userPrincipal;

public void initialize(Subject subject, CallbackHandler callbackHandler,
                      Map sharedState, Map options)
{
    this.subject = subject;
    this.callbackHandler = callbackHandler;
    this.sharedState = sharedState;
    this.options = options;

    // initialize any configured options
    debug = "true".equalsIgnoreCase((String)options.get("debug"));
}

public boolean login() throws LoginException
{
    // prompt for a user name and password
    if (callbackHandler == null)
        throw new LoginException("Error: no CallbackHandler available " +
                               "to garner authentication information from the user");

    Callback[] callbacks = new Callback[2];
    callbacks[0] = new NameCallback("user name: ");
    callbacks[1] = new PasswordCallback("password: ", false);

    try
    {
        callbackHandler.handle(callbacks);
        username = ((NameCallback)callbacks[0]).getName();
        char[] tmpPassword = ((PasswordCallback)callbacks[1]).getPassword();
        if (tmpPassword == null)
        {
            // treat a NULL password as an empty password
            tmpPassword = new char[0];
        }
        password = new char[tmpPassword.length];
        System.arraycopy(tmpPassword, 0,
                         password, 0, tmpPassword.length);
        ((PasswordCallback)callbacks[1]).clearPassword();
    }
    catch (java.io.IOException ioe)
    {
        throw new LoginException(ioe.toString());
    }
    catch (UnsupportedCallbackException uce)
    {
```

```
throw new LoginException("Error: " + uce.getCallback().toString() +
    " not available to garner authentication information " +
    "from the user");
}

// print debugging information
if (debug)
{
    System.out.println("\t\t[LoginModule] " +"user entered user name: " +
        username);
    System.out.print("\t\t[LoginModule] " + "user entered password: ");
    for (int i = 0; i < password.length; i++)
        System.out.print(password[i]);
    System.out.println();
}

// verify the username/password
boolean usernameCorrect = false;
boolean passwordCorrect = false;
if (username.equals("Vilvens the Great"))
    usernameCorrect = true;
if (usernameCorrect &&
    password.length == 12 &&
    password[0] == 'W' &&
    password[1] == 'e' &&
    password[2] == 's' &&
    password[3] == 's' &&
    password[4] == 'e' &&
    password[5] == 'x' &&
    password[6] == '.' &&
    password[7] == '.' &&
    password[8] == '.' &&
    password[9] == '.' &&
    password[10] == '.' &&
    password[11] == '.') {

    // authentication succeeded!!!
    passwordCorrect = true;
    if (debug)
        System.out.println("\t\t[LoginModule] " + "authentication succeeded");
    succeeded = true;
    return true;
}
else
{
    // authentication failed -- clean out state
    if (debug)
        System.out.println("\t\t[LoginModule] " + "authentication failed");
    succeeded = false;
    username = null;
```

```
for (int i = 0; i < password.length; i++) password[i] = ' ';
password = null;
if (!usernameCorrect)
{
    throw new FailedLoginException("User Name Incorrect");
}
else
{
    throw new FailedLoginException("Password Incorrect");
}
}

public boolean commit() throws LoginException
{
    if (succeeded == false) {return false;}
    else
    {
        // add a Principal (authenticated identity) to the Subject
        // assume the user we authenticated is the Principal
        userPrincipal = new JAASNomPrincipal(username);
        if (!subject.getPrincipals().contains(userPrincipal))
            subject.getPrincipals().add(userPrincipal);

        if (debug)
        {
            System.out.println("\t\t[LoginModule] " +"added SamplePrincipal to Subject");
        }

        // in any case, clean out state
        username = null;
        for (int i = 0; i < password.length; i++) password[i] = ' ';
        password = null;

        commitSucceeded = true;
        return true;
    }
}

public boolean abort() throws LoginException
{
    if (succeeded == false)
    {
        return false;
    }
    else if (succeeded == true && commitSucceeded == false)
    {
        // login succeeded but overall authentication failed
        succeeded = false;
        username = null;
    }
}
```

```

if (password != null)
{
    for (int i = 0; i < password.length; i++) password[i] = ' ';
    password = null;
}
userPrincipal = null;
}
else
{
    // overall authentication succeeded and commit succeeded,
    // but someone else's commit failed
    logout();
}
return true;
}

public boolean logout() throws LoginException
{
    subject.getPrincipals().remove(userPrincipal);
    succeeded = false;
    succeeded = commitSucceeded;
    username = null;
    if (password != null)
    {
        for (int i = 0; i < password.length; i++)
            password[i] = ' ';
        password = null;
    }
    userPrincipal = null;
    return true;
}
}

```

4.6 Un Principal

Pour notre modeste exemple, le Principal sera simplement le nom entré dans la procédure d'identification :

JAASNomPrincipal.java
package jaasbasics.principal;
import java.security.Principal;
public class JAASNomPrincipal implements Principal , java.io.Serializable
{
private String name;
public JAASNomPrincipal(String name)
{
if (name == null) throw new NullPointerException("illegal null input");

```
        this.name = name;
    }

    public String getName() { return name; }

    public String toString() { return("Principal: " + name); }

    public boolean equals(Object o)
    {
        if (o == null) return false;
        if (this == o) return true;
        if (!(o instanceof JAASNomPrincipal)) return false;
        JAASNomPrincipal that = (JAASNomPrincipal)o;
        if (this.getName().equals(that.getName())) return true;
        return false;
    }

    public int hashCode() {return name.hashCode(); }
}
```

4.7 Exécuter un code en étant authentifié

Pour tester tout cela, il nous reste à définir un code (basique) à exécuter :

MyAction.java

```
package jaasbasics;

import java.security.PrivilegedAction;

public class MyAction implements PrivilegedAction
{
    private String message;
    public MyAction(String m)
    {
        message=m;
    }
    public Object run()
    {
        System.out.println(message);
        return message;
    }
}
```

puis à demander son exécution :

JAASNom.java

```
package jaasbasics;
...
public class JAASNom
{
```

```
private static int NOMBRE_MAXIMUM_ESSAIS = 3;
public static void main(String[] args)
{
    ...
    int i;
    for (i = 0; i < NOMBRE_MAXIMUM_ESSAIS; i++)
    {
        try
        {
            lc.login();
            break;
        }
        ...
    }

    Subject sub = lc.getSubject();
    Iterator principalIterator = sub.getPrincipals().iterator();
    System.out.println("Authenticated user has the following Principals:");
    while (principalIterator.hasNext())
    {
        Principal p = (Principal)principalIterator.next();
        System.out.println("\t" + p.toString());
    }

    System.out.println("User has " + sub.getPublicCredentials().size() +
                       " Public Credential(s)");

    PrivilegedAction action = new MyAction("Ben dis donc !");
    Subject.doAs(sub, action);

    System.exit(0);
}
```

On obtient dans la console :

```
Répertoire courant = C:\Documents and Settings\Utilisateur
user name: Vilvens the Great
password: Wessex.....
[LoginModule] user entered user name: Vilvens the Great
[LoginModule] user entered password: Wessex.....
Authentification réussie :-
[LoginModule] authentication succeeded
[LoginModule] added SamplePrincipal to Subject
Authenticated user has the following Principals:
    Principal: Vilvens the Great
User has 0 Public Credential(s)
Ben dis donc !
```

Si on a oublié de modifier la ligne de l'url du fichier de configuration dans java.security :

Cannot create LoginContext. SecurityException.Impossible de trouver une configuration de connexion
Java Result: -2

En ligne de commande, cela donnerait

```
C:\java-netbeans-application\JAASBasics>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 6CF8-0DE7

Répertoire de C:\java-netbeans-application\JAASBasics

10/05/2011 10:59  <REP>      .
10/05/2011 10:59  <REP>      ..
10/05/2011 10:59  <REP>      build
08/05/2011 10:44      3.725 build.xml
10/05/2011 10:59  <REP>      dist
10/05/2011 10:58      140 JAASNom.config
08/05/2011 10:44      85 manifest.mf
08/05/2011 10:44  <REP>      nbproject
10/05/2011 10:42  <REP>      src
08/05/2011 10:44  <REP>      test
            3 fichier(s)    7.072 octets
            7 Rép(s)  62.779.920.384 octets libres
```

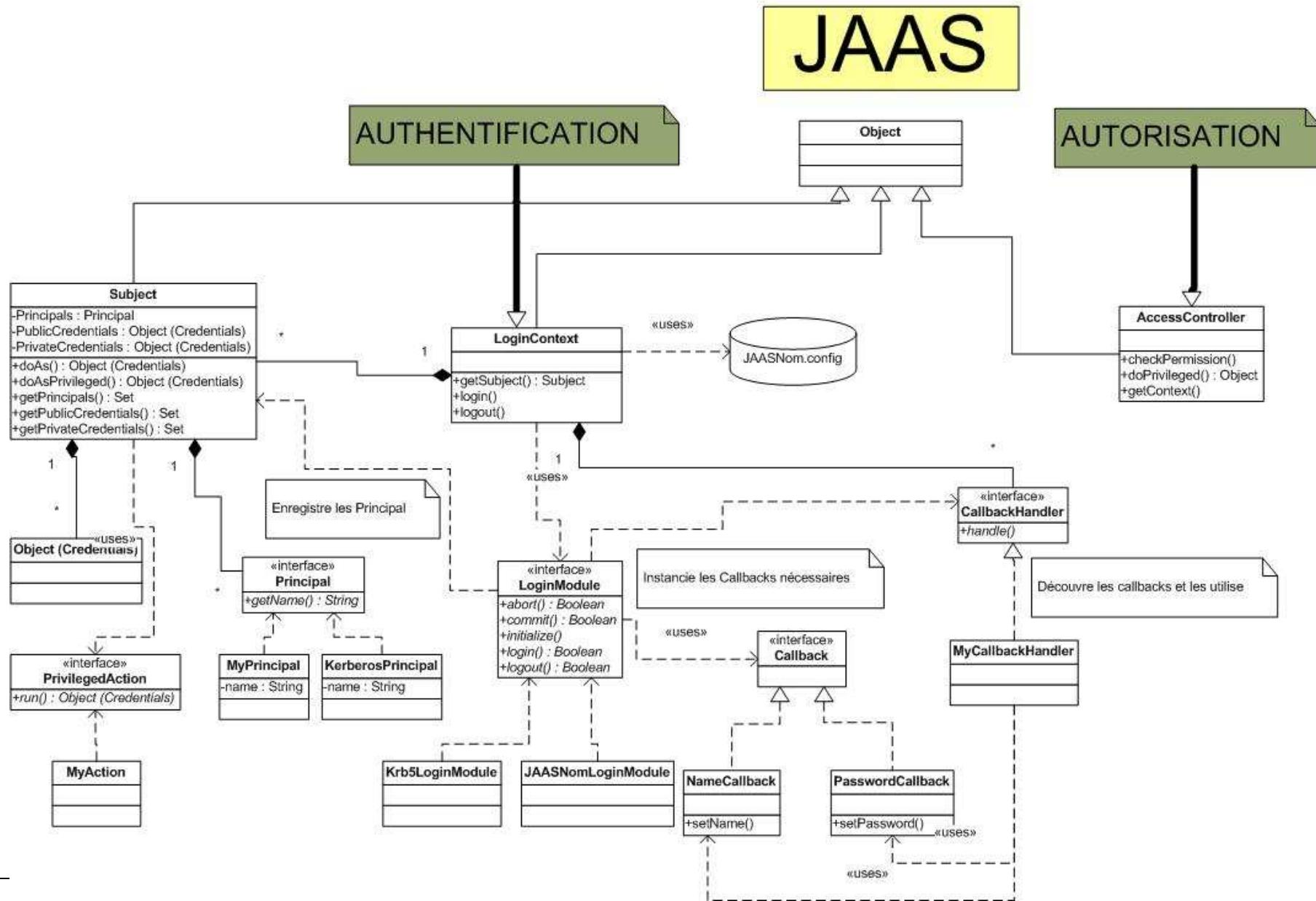
```
C:\java-netbeans-application\JAASBasics>javac -d . src/jaasbasics/*.java src/jaa
sbasics/module/*.java src/jaasbasics/principal/*.java
```

```
C:\java-netbeans-application\JAASBasics>java -Djava.security.auth.login.config=
JAASNom.config jaasbasics.JAASNom
```

On obtient dès lors la même exécution que sous Netbeans. Ouf !

4.8 Le diagramme de classes de JAAS

En résumé :



5. Le contexte Kerberos

5.1 Objectifs et acteurs

Pour rappel (voir Java II, chapitre XIV. La cryptographie et Java), le protocole Kerberos est **un protocole d'authentification de clients vis-à-vis de serveurs**

- ◆ qui utilise comme éléments d'authentification des mots de passe et des clés symétriques partagées;
- ◆ qui fonctionne sur base de "tickets" (ayant une validité limitée à un certain temps) et d'"authentificateurs" (variables au cours du temps) cryptés avec des clés symétriques.

Le protocole Kerberos est issu du projet "Athena" du MIT, mené par Miller et Neuman. La version 5 du protocole Kerberos a été normalisée par l'IETF dans les **RFC 1510** (1993) et **1964** (1996).

L'architecture de Kerberos consiste donc en

- ◆ un certain nombre de **serveurs** regroupés dans un domaine;
- ◆ des **clients** potentiels de ces serveurs, utilisateurs distants désirant utiliser les services disponibles,
- ◆ un (voire plusieurs) **serveur d'authentification** (AS - Authentication Server) : il authentifie les utilisateurs distants et leur permet d'entrer dans le domaine Kerberos
- ◆ un (voire plusieurs) **serveur de distribution de tickets de service** (TGS - Ticket Granting Service) : il permet à ces utilisateurs d'accéder à des services disponibles sur les serveurs du domaine.

Quelques précisions complémentaires :

- ◆ les clients peuvent aussi bien être des utilisateurs que des machines;
- ◆ la plupart du temps, les deux types de services sont regroupés sur un même serveur, le Centre de Distribution des Clés (KDC - Key Distribution Center);
- ◆ à priori, les messages d'une taille inférieure à 2000 bytes sont transportés par UDP, les messages de taille supérieure par TCP, dans les deux cas sur le port **88** (anciennement, le port 750 était également utilisé) – mais tout ceci peut être modifié dans les diverses implémentations.

5.2 Fonctionnement de Kerberos

Le protocole Kerberos repose sur un système de cryptographie à base de clés secrètes (clés symétriques ou clés privées), à priori avec l'algorithme DES (mais AES est disponible dans les versions récentes). Kerberos partage avec chaque client du réseau une clé secrète faisant office de preuve d'identité.

Représons brièvement le principe de fonctionnement de Kerberos qui repose sur la notion de tickets :

- ◆ Afin d'obtenir l'autorisation d'accès à un service, un utilisateur distant doit envoyer son identifiant au serveur d'authentification.
- ◆ Le serveur d'authentification vérifie que l'identifiant existe et envoie un ticket initial au client distant, chiffré avec la clé associée au client. Le ticket initial contient :
 - o un clé de session, faisant office de mot de passe temporaire pour chiffrer les communications suivantes ;

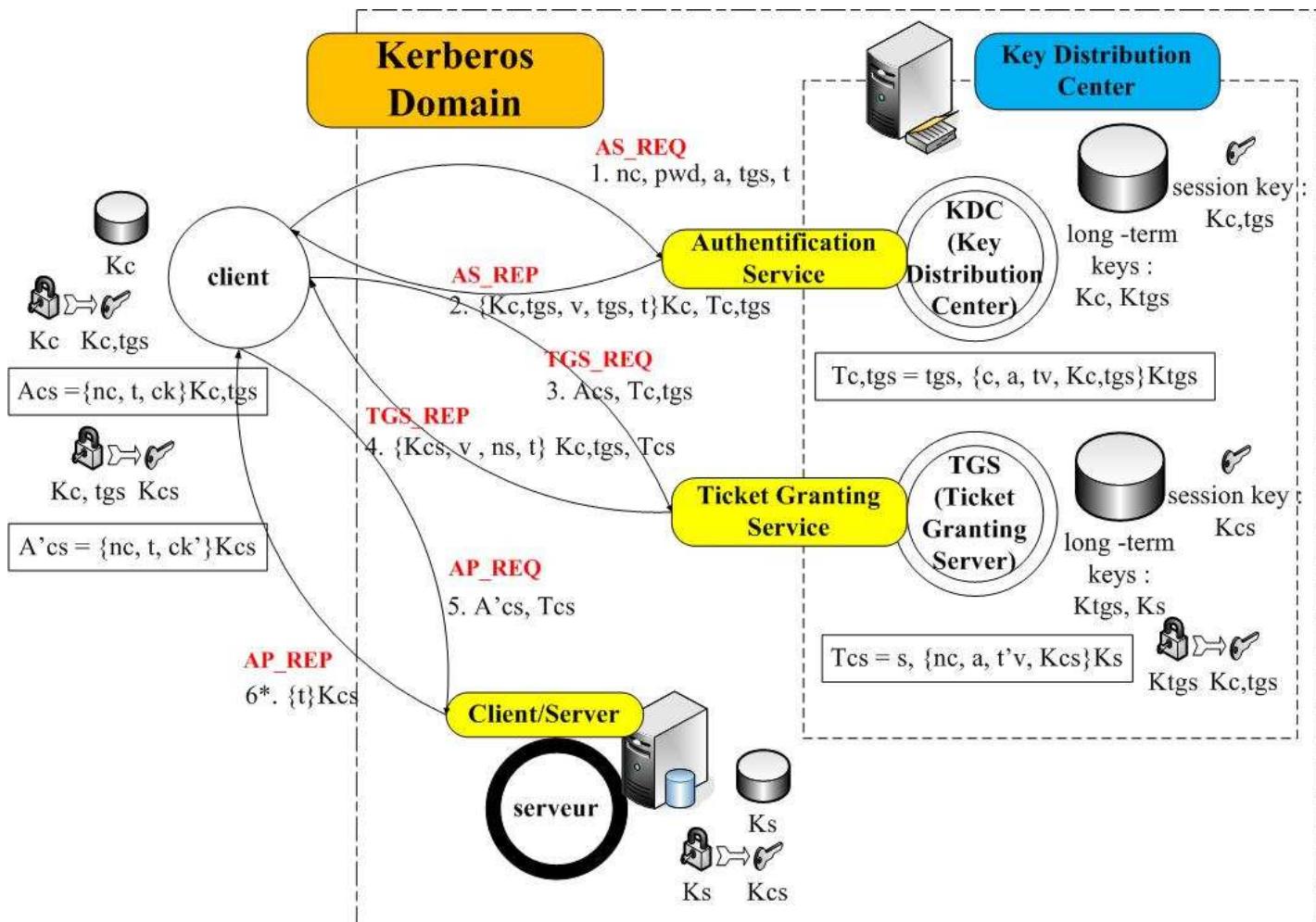
o un ticket d'accès au service de délivrement de ticket.

◆ Le client distant déchiffre le ticket initial avec sa clé et obtient ainsi un ticket et une clé de session.

◆ Grâce à son ticket et sa clé de session, le client distant peut envoyer une requête chiffrée au service de délivrement de ticket, afin de demander l'accès à un service.

L'authentification proposée par le serveur Kerberos a une durée limitée dans le temps, ce qui permet d'éviter à un pirate de continuer d'avoir accès aux ressources : on parle ainsi d'anti re-jeu.

Schématiquement :



Les serveurs, les clients constituent ce que l'on désigne souvent sous le nom de "realm" tandis que les serveurs KDC et TGS (éventuellement fusionnés) constituent le "trusted third party" (la tierce partie de confiance). Kerberos est bien clairement un service SingleSign-On (SSO).

En pratique, les implémentations de systèmes Kerberos utilisent les règles de bonne pratique suivantes :

- ◆ un ticket peut être "renouvelé", mais pour une durée totale n'excédant pas 7 jours;
- ◆ un ticket d'accès à un serveur n'est valable qu'une heure;
- ◆ un ticket d'accès à TGS reste valable de 8 à 10 heures;
- ◆ la différence tolérée de synchronisation des horloges des participants est de 5 minutes.

5.3 Une implémentation Java

Il existe donc, sans réelle surprise, une classe `com.sun.security.auth.module.Krb5LoginModule` qui est un LoginModule au sens de JAAS et qui utilise le mécanisme du Ticket Granting Ticket (TGT) avec un ticket conservé dans l'ensemble privé de credentials du subject considéré. Ce module peut aussi utiliser un cache existant de credentials, comme celui de Windows 2000 ou celui d'un Solaris, ce qui est évidemment indispensable dans le contexte d'une application Java portable !

L'ensemble de credentials d'un client peut être le cache natif et pourrait être désigné par une configuration comme :

JAASKClient.config

```
ClientKerSSO {  
    com.sun.security.auth.module.Krb5LoginModule required  
    useTicketCache=true  
};
```

Le serveur pourrait utiliser une configuration où une clé secrète en mémoire est utilisée pour authentifier un principal placé dans le fichier `nfs/trusted.clients.com` – le ticket et la clé secrète sont stockées dans le credentials set du subject :

JAASKServeur.config

```
ServeurKerSSO {  
    com.sun.security.auth.module.Krb5LoginModule  
    required useKeyTab=true storeKey=true  
    principal="nfs/trusted.clients.com"  
};
```

Du point de vue programmation, cela donne simplement côté client :

KerbClient.java

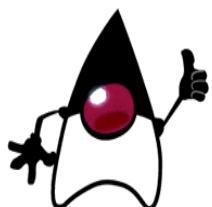
```
...  
LoginContext lc = null;  
try  
{  
    lc = new LoginContext("ClientKerSSO", newTextCallbackHandler());  
    // attempt authentication  
    lc.login();  
}  
catch (LoginException le)  
{  
    ...  
}  
  
// Now try to execute ClientAction as the authenticated Subject  
Subject mySubject = lc.getSubject();  
PrivilegedAction action = new ClientAction();  
Subject.doAs(mySubject, action);  
...
```

et côté serveur :

KerbServeur.java

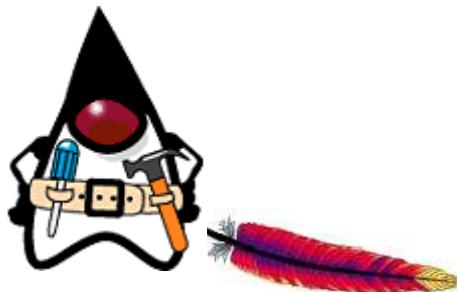
```
...
LoginContext lc = null;
try
{
    lc = new LoginContext("ServeurKerSSO", newTextCallbackHandler());
    // attempt authentication
    lc.login();
}
catch (LoginException le)
{
    ...
}

// Now try to execute ServerAction as the authenticated Subject
Subject mySubject = lc.getSubject();
PrivilegedAction action = new ServerAction();
Subject.doAs(mySubject, action);
...
```



Nous pourrions encore étudier plus profondément les implémentations Kerberos et surtout des APIs plus générales comme GSS. Mais quittons ce monde sévère des authentifications et des serveurs d'application et retrouvons les applications Web et/ou classiques ... mais "automatisées" : vous avez dit "framework" ?

XXXVII. Une introduction au framework Struts



Love your neighbour as yourself, but choose your neighbouring.

(L. Beale)

1. Un framework MVC

Le modèle MVC est à présent devenu un classique de la programmation des applications Web : c'est même devenu un *design pattern*. Il est implémenté dans diverses plates-formes de développement en anglais, des "frameworks"), dont l'une des plus célèbres est **Struts**. Struts¹⁻² est un "framework MVC" : autrement dit, il fournit des bibliothèques et des fichiers de configuration facilitant le développement d'applications Web en séparant la logique (servlets et Java Beans) de la présentation (Java Server Pages). C'est un framework opensource (donc extensible) géré par l'Apache Software Foundation.



Struts

Très clairement, Struts concerne donc les développements basés sur les technologies servlets/JSP. Il en utilise aussi les technologies dérivées, comme :

- ◆ Expression Language (EL) : utilisation de Java Beans dans des JSP sans connaissance de Java;
- ◆ les tags personnalisés (*custom tag*) : déjà évoqués dans le chapitre consacré aux JSPs;
- ◆ Java Standard Tag Library (JSTL) : bibliothèque de tags personnalisés standards, prêts à l'emploi – mais Struts fournit aussi les siens.

En fait, Struts poursuit les mêmes buts que **JavaServer Faces (JSF)**, technologie de conception d'interfaces utilisateur en séparant logique et présentation, et est donc dans une certaine concurrence avec elle.

Le gros avantage par rapport à un développement MVC créé à partir de rien (comme exposé dans les chapitres de Java III) est non seulement que l'on y trouve des outils prêts à l'emploi (comme par exemple des tags personnalisés placés dans des bibliothèques), mais aussi que le développeur est guidé dans ses travaux afin de conserver une véritable architecture MVC. Le **contrôleur MVC** de l'implémentation de cette architecture est un

¹ <http://struts.apache.org>

² le mot anglais "strut" est un terme de construction et d'architecture; il signifie en fait "étai", "traverse", "arc boutant" – allusion au fait que l'on procure ici des éléments de structure dissimulés qui constituent la charpente d'une application.

composant propre à Struts. Par contre, le **modèle MVC** utilise notamment les Java Beans, la technologie JDBC et les Enterprise Java Beans, tandis que la **vue MVC** utilise entre autres les Java Server Pages et les techniques XSLT.

Si Struts ne fait pas partie de J2EE, il n'en reste donc pas moins étroitement associé à cette plate-forme. En fait, Struts apporte une série d'outils et de méthodes de travail en accord avec la philosophie de J2EE.

Inutile de dire que l'utilisation de Struts ne se justifie que pour des applications Web d'une certaine ampleur et à gros besoins de maintenance.

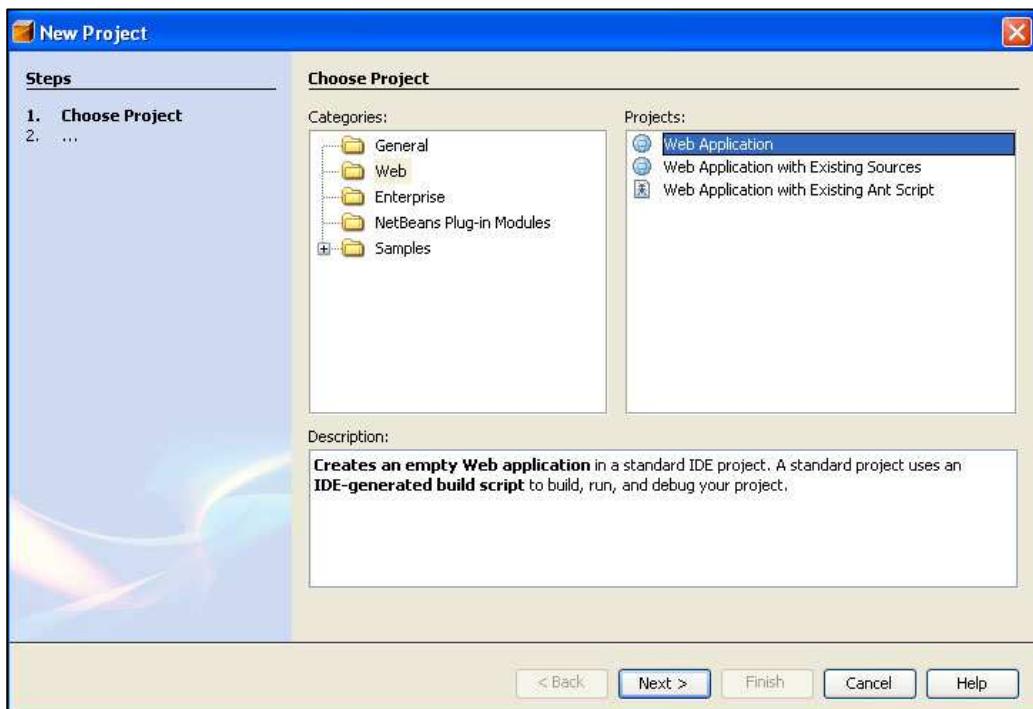
Remarque

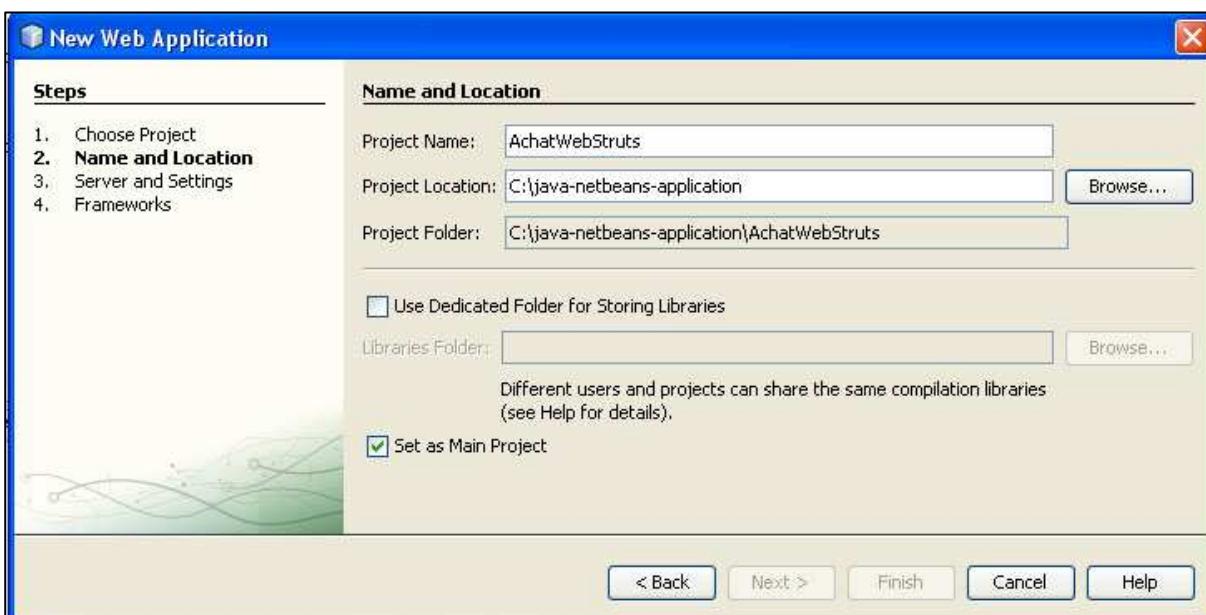
A remarquer que Struts n'est pas un portail, c'est-à-dire une application Web disposant en fait d'un ensemble de petites applications Web indépendantes les unes des autres et affichées sur la même page (une application s'intégrant sur un portail est encore appelée une portlet – voir plus loin dans cet ouvrage).

2. Une application Web élémentaire

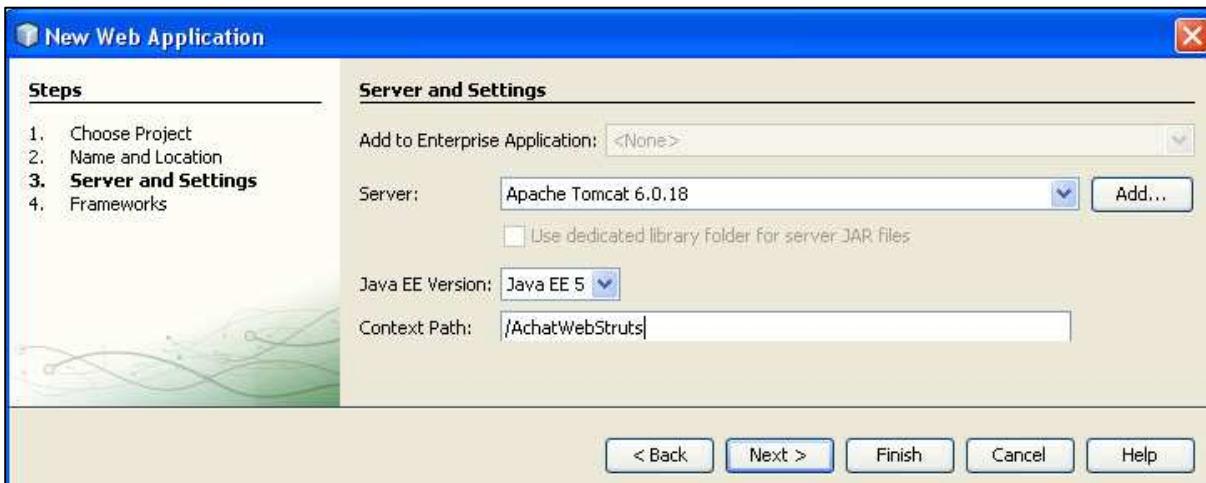
Pour être concret, imaginons vouloir développer une application Web d'achat élémentaire sur le Web. Dans un premier temps, notre objectif sera très limité : le client contacte le magasin virtuel au moyen d'un JSP, introduit son nom et son mot de passe (supposons qu'il en possède un pour simplifier) et reçoit en réponse une page d'acceptation. Pour être élémentaire, ce l'est ... mais avec Struts, cela va déjà mettre en branle une série impressionnante de concepts.

Nous allons travailler concrètement avec NetBeans 5.5 ou 6.*, qui présente l'avantage d'intégrer le framework Struts (donc, sans nécessiter une phase configuration). Il convient tout d'abord de créer une application Web :

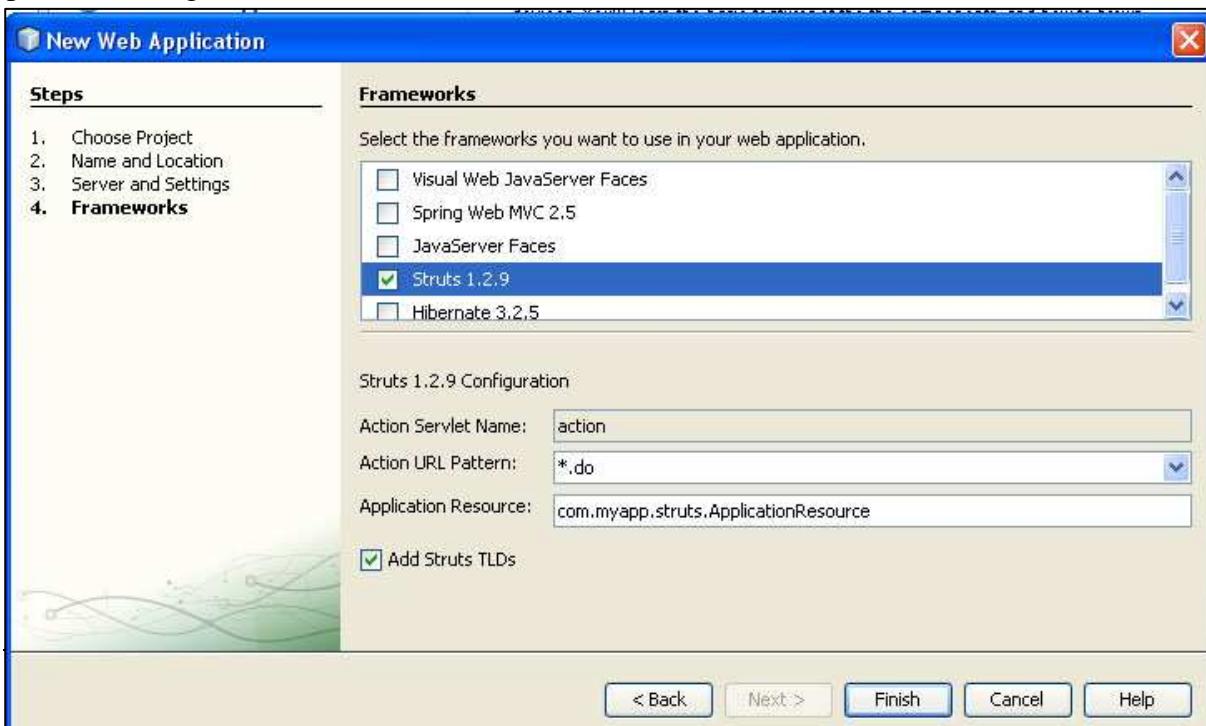




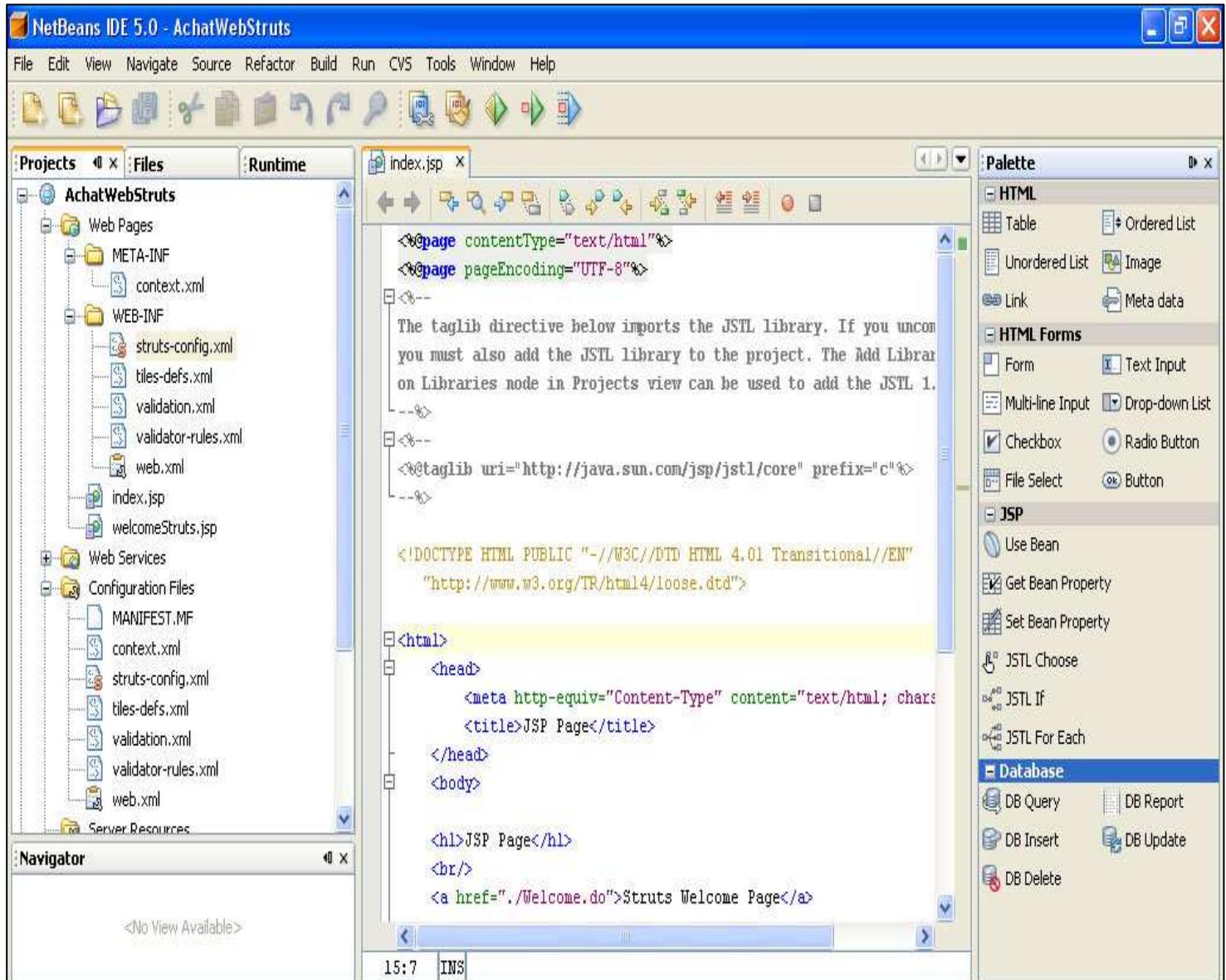
A remarquer que l'on se base sur J2EE 1.4 ou J EE 5 (NetBeans 6) et que le serveur Web utilisé pour le développement est un Tomcat ou un GlassFish (NetBeans 6) intégré :



puis nous intégrons Struts :



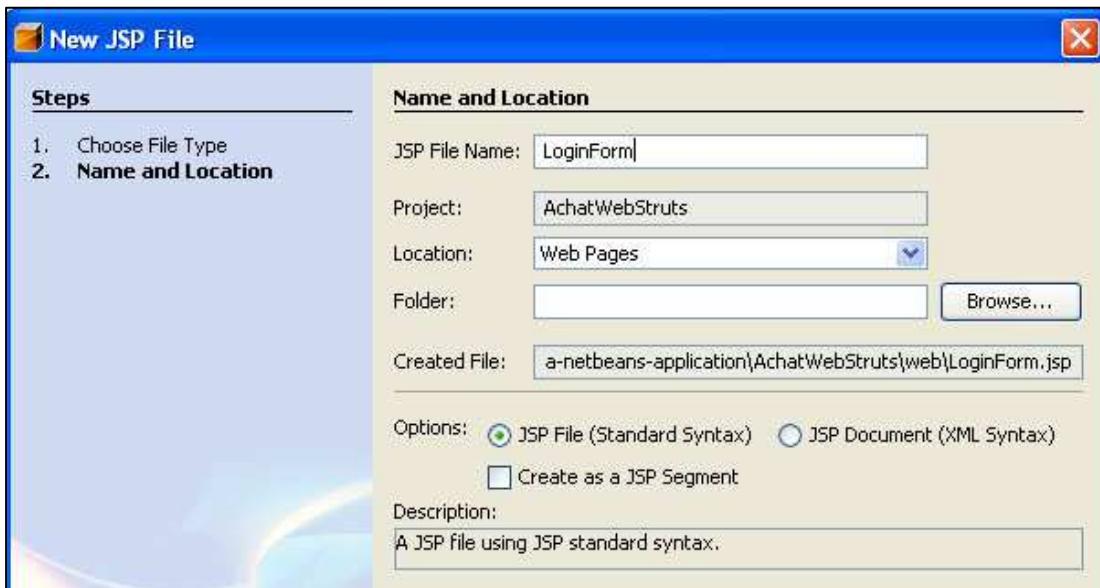
Le résultat est un projet déjà bien fourni :



On peut remarquer notamment, outre la présence de deux JSPs de base (index.jsp et welcomeStruts.jsp), l'existence d'un fichier **web.xml** (logique pour une application Web) et de plusieurs autres fichiers xml, à la fois dans les fichiers de configurations de l'application développée sous NetBeans et dans le répertoire WEB-INF (ceci en vue du déploiement de l'application sur un serveur distant et non plus local intégré). L'un de ces fichiers xml présente une importance particulière : il s'agit du descripteur de déploiement **struts-config.xml**, dont nous allons reparler.

3. Le JSP de login

Créons un premier JSP pour l'entrée sur le site :



A priori, nous pourrions compléter le JSP généré par quelque chose du genre :

```
<h1>Bienvenue ! Veuillez vous identifier</h1>
<form action="login">
    Votre nom : <input type="text" name="nom" value="" /> <p>
    Votre mot de passe : <input type="password" name="motDePasse" value="" /> <p>
        <input type="submit" value="soumettre" name="submit" />
</form>
```

avec des tags HTML l'on sélectionne dans la palette de droite et que l'on "drag and drop" dans le JSP. Cependant, en fait, nous allons utiliser les tags des bibliothèques de Struts.

4. Les librairies de tags de Struts

Struts fournit également des librairies de tags personnalisés qui facilitent la création des JSP qui seront envoyés au client. En fait, certains tags de la JSTL sont utilisables et, dans ce cas, Struts conseille de préférer ces tags standards aux tags Struts. Les librairies de tags doivent évidemment être définies dans web.xml au moyen de directives **taglib** :

```
<taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
<taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
...
...
```

La librairie que nous utiliserons immédiatement est **struts-html**. On s'en doute, les tags de cette librairie contribuent à la création de formulaires au sens HTML du terme, soit un GUI permettant l'introduction de données. Pour cela, il faut au préalable insérer les directives concernant les bibliothèques de tags :

```
<%@taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>
<%@taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>
<%@taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic" %>
```

On peut à présent insérer dans la JSP des tags de struts tels que **<html:form>** qui va en fait générer la classique balise **<form>** qui lancera la servlet de contrôle. L'attribut indispensable est donc le nom de l'action que cette servlet doit reconnaître : il doit donc s'agir d'une action définie dans struts-config.xml - ici, ce sera "login".

A l'intérieur de ce tag, on va pouvoir placer les tags habituels d'un formulaire, comme par exemple :

- ◆ **<html:text>** qui génère en fait une balise html **<input type="text">**; l'attribut **property** permet de donner le nom qui sera associé à cette zone de saisie et qui représentera cette donnée en tant que propriété dans le bean formulaire;
- ◆ **<html:password>** analogue au précédent mais version "mot de passe", donc caché;
- ◆ **<html:submit>** qui génère bien sûr une balise html **<input type="submit">**

Cela peut donner :

LoginForm.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Entrée dans le magasin virtuel</title>
  </head>
  <body>

    <h1>Bonjour et bienvenue ! <br> Veuillez vous identifier ...</h1>
    <%-- un tag <Form> normal ne fonctionne pas !!!! --%>
    <html:form action="login">
      Votre nom : <html:text property="name" /> <p>
      et votre mot de passe : <html:password property="password" /> <p>
      <html:submit value="Ok" />
    </html:form>
    ...
  </body>
</html>
```

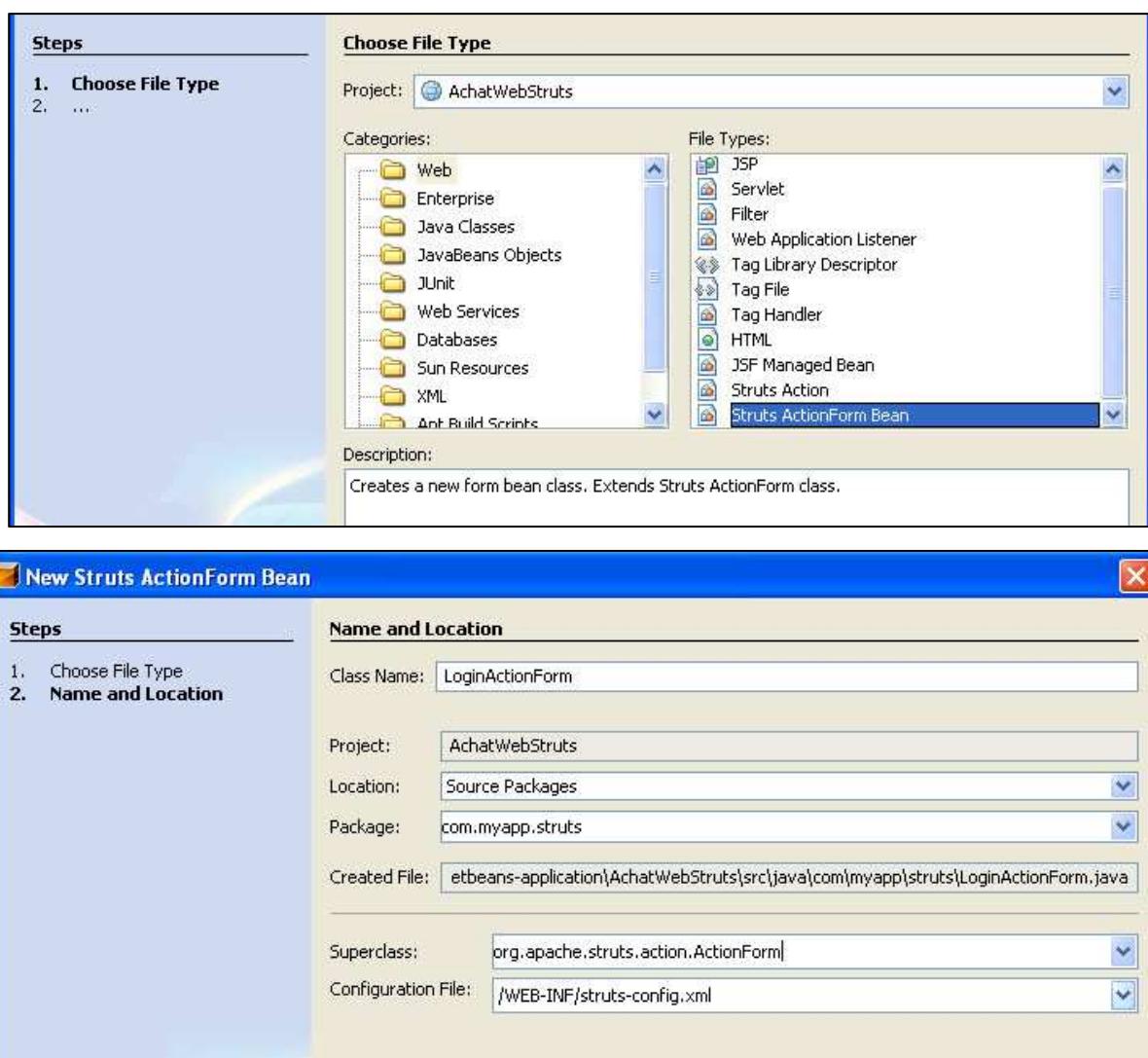
Bien sûr, ce formulaire va activer une servlet située du côté du serveur. Avant d'en parler, il conviendrait de fabriquer d'abord *le Java Bean qui, selon le modèle Struts, va contenir les données introduites dans ce formulaire.*

5. Le bean ActionForm associé au login

Le bean associé à cette opération de login, et qui est chargé de contenir les données saisies au moyen du formulaire, doit s'appeler LoginActionForm parce qu'il correspond à une action nommée "login", et doit posséder les propriétés name et password, d'après les propriétés que nous avons imaginé dans le JSP. Ce bean sert donc clairement de container pour les informations saisies. Il doit être une instance d'une classe dérivée de la classe **ActionForm** du framework.

En pratique :

New → File/Folder puis Web et Struts ActionForm :



Après quelques modifications, on parvient à :

LoginActionForm.java

```
/*
 * LoginActionForm.java
 *
 */

package com.myapp.struts;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;

/***
 *
 * @author Vilvens
 */

public class LoginActionForm extends org.apache.struts.action.ActionForm
{
    private String name;
    private String password;

    public String getName() { return name; }
    public void setName(String string) { name = string; }

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }

    public LoginActionForm() {
        super();
        // TODO Auto-generated constructor stub
    }

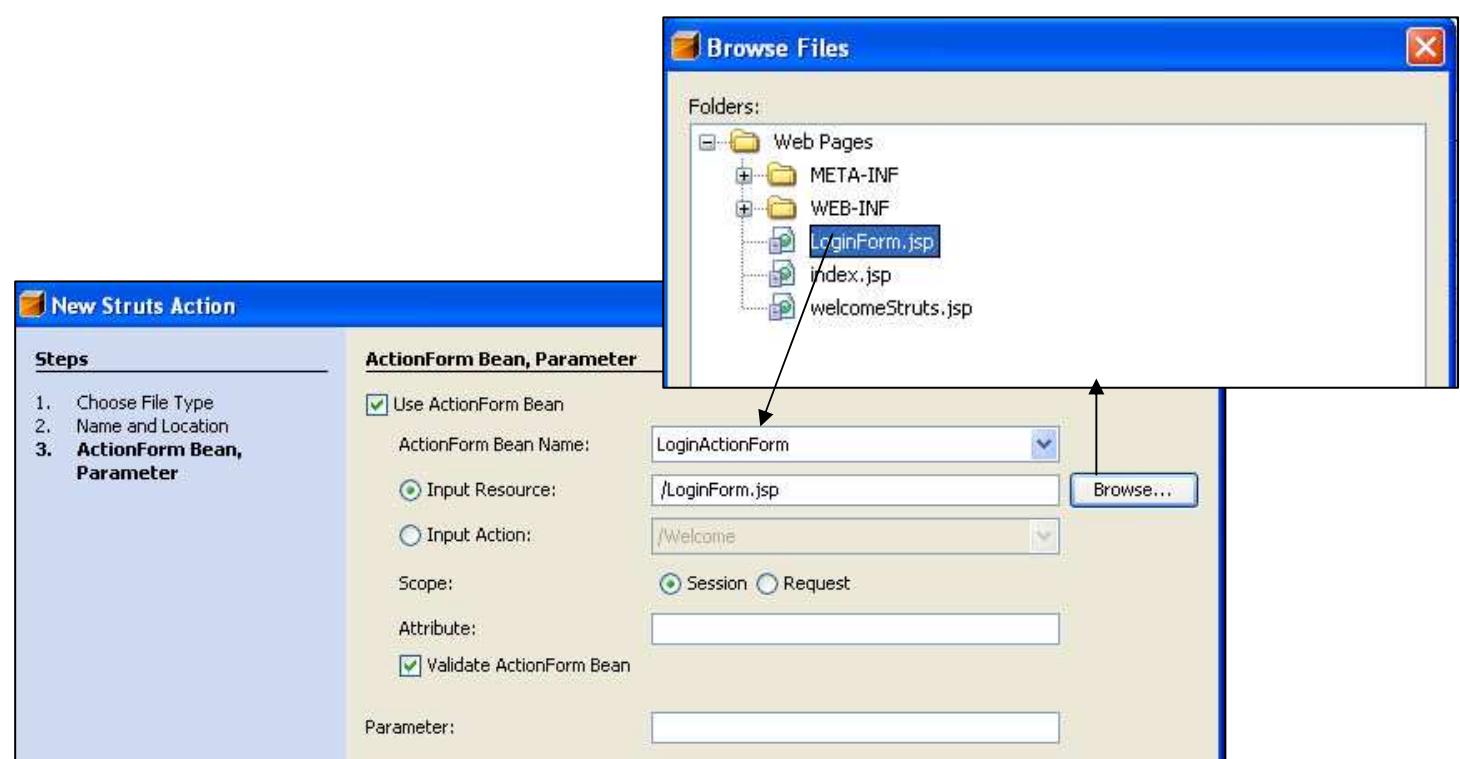
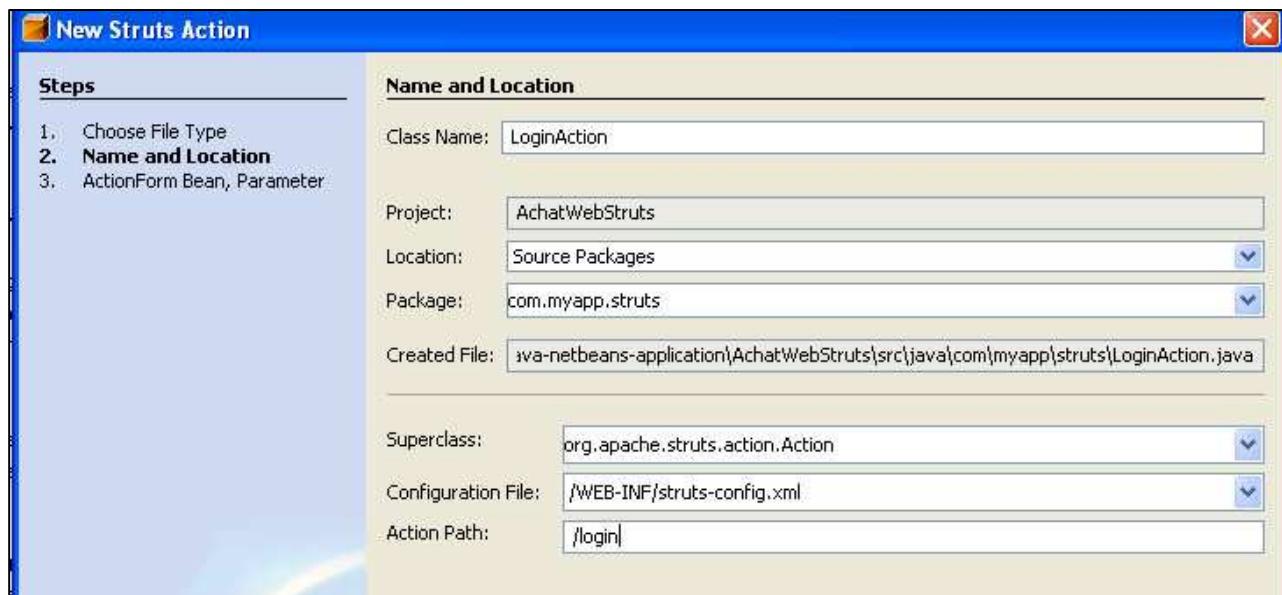
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
        ActionErrors errors = new ActionErrors();
        if (getName() == null || getName().length() < 1) {
            errors.add("name", new ActionMessage("error.name.required"));
            // TODO: add 'error.name.required' key to your resources
        }
        return errors;
    }
}
```

6. La classe métier Action

Il nous faut à présent associer une réaction à l'introduction de nos données dans le formulaire. Cette réaction va, indirectement et en définitive, être assurée par une classe dérivée de la classe **Action** du package org.apache.struts.action. et qui s'appellera logiquement *LoginAction* (mais elle pourrait s'appeler autrement si on le voulait : voir plus loin dans struts-config.xml). L'idée est que ***la servlet de contrôle*** (dont nous allons parler au paragraphe suivant) ***délègue le travail à un tel objet Action***, qui implémente donc la logique métier de réponse à une requête – on parle encore pour cette raison de "*sous-contrôleur*".

Donc, concrètement :

New → File/Folder puis Web et Struts Action :



On obtient ainsi :

LoginAction.java

```
/*
 * LoginAction.java
 *
 */

package com.myapp.struts;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForward;

/**
 *
 * @author vilvens
 */

public class LoginAction extends Action
{
    private final static String SUCCESS = "success";

    /**
     * This is the action called from the Struts framework.
     * @param mapping The ActionMapping used to select this instance.
     * @param form The optional ActionForm bean for this request.
     * @param request The HTTP Request we are processing.
     * @param response The HTTP Response we are processing.
     * @throws java.lang.Exception
     * @return
     */
    public ActionForward execute (ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception
    {
        return mapping.findForward(SUCCESS);
    }
}
```

On comprend sans peine que les actions à entreprendre sont à programmer dans la méthode

```
public ActionForward execute (ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response)
```

- ◆ le 1^{er} paramètre est un **ActionMapping**, c'est-à-dire qu'il contient les informations reliant les JSP, les Java Beans et les actions; ces informations se trouvent en fait dans le descripteur de déploiement (le fichier **struts-config.xml** – voir paragraphes suivants) et l'objet ActionMapping ne fait que rendre ces informations plus rapidement accessibles;
- ◆ le 2^{ème} paramètre représente évidemment le Java bean (un ActionForm) qui contient les informations introduites dans le formulaire qu'est le JSP;
- ◆ les deux derniers paramètres n'ont sans doute pas besoin d'être commentés ;-).

Par défaut, le générateur de NetBeans préconise que l'on soit redirigé vers une vue associée au nom "success" – nous allons y revenir. Mais auparavant, comme la servlet de contrôle du modèle MVC va-t-elle savoir qu'elle doit utiliser cette classe Action ?

7. La servlet de contrôle

La servlet de contrôle du modèle MVC est, dans le framework, une instance de la classe **ActionServlet** du package org.apache.struts.action (qui se trouve dans le fichier struts.jar). En réalité, on ne peut se rendre compte de son existence que si on pousse la porte du descripteur de l'application Web :

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
    app_2_4.xsd">
    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>2</param-value>
        </init-param>
        <init-param>
            <param-name>detail</param-name>
            <param-value>2</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
    ...
</web-app>
```

On peut voir que la seule servlet désignée est une instance d'**ActionServlet** du package org.apache.struts.action et est nommée "**action**" en interne. De plus, on peut constater que :

- ◆ toutes les requêtes dont l'URL se termine par .do seront prises en charge par cette servlet, qui aiguillera vers le JSP correspondant, après utilisation éventuelle de Java Beans ou d'Enterprise Java Beans;
- ◆ la servlet action doit être démarrée dès le lancement de l'application Web (balise <load-on-startup>);
- ◆ un paramètre d'initialisation de cette méga-servlet est un fichier struts-config.xml, qui permet de "configurer Struts" pour l'application. Mais encore ? En fait, au lieu de présenter un code statique qui envisage les diverses possibilités d'état (comme la servlet de contrôle du modèle MVC du chapitre consacré aux JSP et qui testait la valeur du paramètre action à coup de "if" écrits "à la dure"), cette servlet va découvrir dynamiquement ces possibilités et les noms des servlets et JSP à utiliser en lisant ce fichier struts-config.xml ...

8. Le descripteur de déploiement de Struts

8.1 La forme générale

En fait, une application développée avec Struts est complémentairement décrite au moyen d'un descripteur similaire à web.xml : **struts-config.xml**. Celui-ci a pour rôle de décrire les JSPs, les Java Beans et les actions qui vont être utilisés. Il peut ressembler à ceci pour notre exemple :

struts-config.xml

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

<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
  "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>
  <form-beans>
    <form-bean name="LoginActionForm" type="com.myapp.struts.LoginActionForm" />
  </form-beans>

  <global-exceptions></global-exceptions>

  <global-forwards></global-forwards>

  <action-mappings>
    <action input="/LoginForm.jsp" name="LoginActionForm" path="/login"
           scope="session" type="com.myapp.struts.LoginAction">
      <forward name="success" path="/LoginReussi.jsp"/>
    </action>
  </action-mappings>
  ...
</struts-config>
```

8.2 L'action-mapping : les actions

L'élément fondamental est la présence du tag <action-mappings>. Le rôle d'un tel tag est de permettre au contrôleur Struts d'associer à une requête (disons par exemple `http://myriam:8080/login.do`) définie par l'attribut **path** (toujours précédée par "/") la classe dont il faudra invoquer la méthode **execute()** pour effectuer le traitement adéquat (donc ce qui est "à faire" = "to do"). Comme déjà évoqué, cette classe est une classe dérivée de la classe **Action**, dont le nom sera supposé implicitement être celui de la requête .do complété de "Action" (donc : login.do → classe LoginAction).

Plus précisément, on peut décrire le tag <action-mappings> comme comportant des sous-tags <action ...></action> dont les principaux attributs sont :

- ◆ **input** : indique le JSP qui a généré le formulaire servant aux entrées (ici : "/LoginForm.jsp");
- ◆ **name** : spécifie le bean ActionForm à utiliser pour contenir les informations saisies (ici : "LoginActionForm");
- ◆ **path** : donne la requête (ici : "/login");
- ◆ **scope** : précise la durée de vie du bean ActionForm au sens HTTP des servlets et des JSPs; la valeur peut être "request" mais surtout "session" (c'est le cas ici);
- ◆ **type** : précise le nom de la classe Action utilisée (ici : "com.myapp.struts.LoginAction").

Il existe encore un attribut **validate**, mais nous en reparlerons plus loin.

8.3 L'action-mapping : les forwards

Le même tag <action> contient un ou plusieurs tags <forward> : ils permettent de *préciser le JSP qui constituera la réponse en fonction de la chaîne renournée* (typiquement "success" ou "failure") *par l'objet Action au contrôleur principal ActionServlet*. Ceci se fait au moyen des attributs "name" (la réponse de l'Action) et "path" (le JSP auquel il faut passer). De tels forwards sont encore qualifiés de "**forwards locaux**", car ils ne sont connus que dans le contexte d'une action précise.

8.4 Les forwards globaux

Enfin, le tag <global-forwards> permet, au moyen de ses tags <forward>, de définir globalement les instances d'ActionForward qui seront renvoyées par les méthodes execute() des classes ActionForm : l'attribut "name" en définit le nom logique tandis que "path" permet de retrouver la ressource, c'est-à-dire le JSP, correspondant. L'intérêt est que l'on peut ainsi changer le JSP en ce seul endroit en conservant le nom logique. Dans notre cas, on pourrait imaginer :

```
<global-forwards>
    <forward name="Connected" path="/Connected.jsp" />
</global-forwards>
```

8.5 Les form-beans

Le tag <form-beans> permet de définir les beans **containers**, c'est-à-dire les beans qui contiennent les informations encodées dans les formulaires. Dans chaque tag <form-bean>, le nom complet de la classe utilisée doit être précisé dans l'attribut "type" tandis que l'attribut "name" permet de définir un identifiant unique qui sera référencé dans le tag <action-mappings> expliqué ci-dessus.

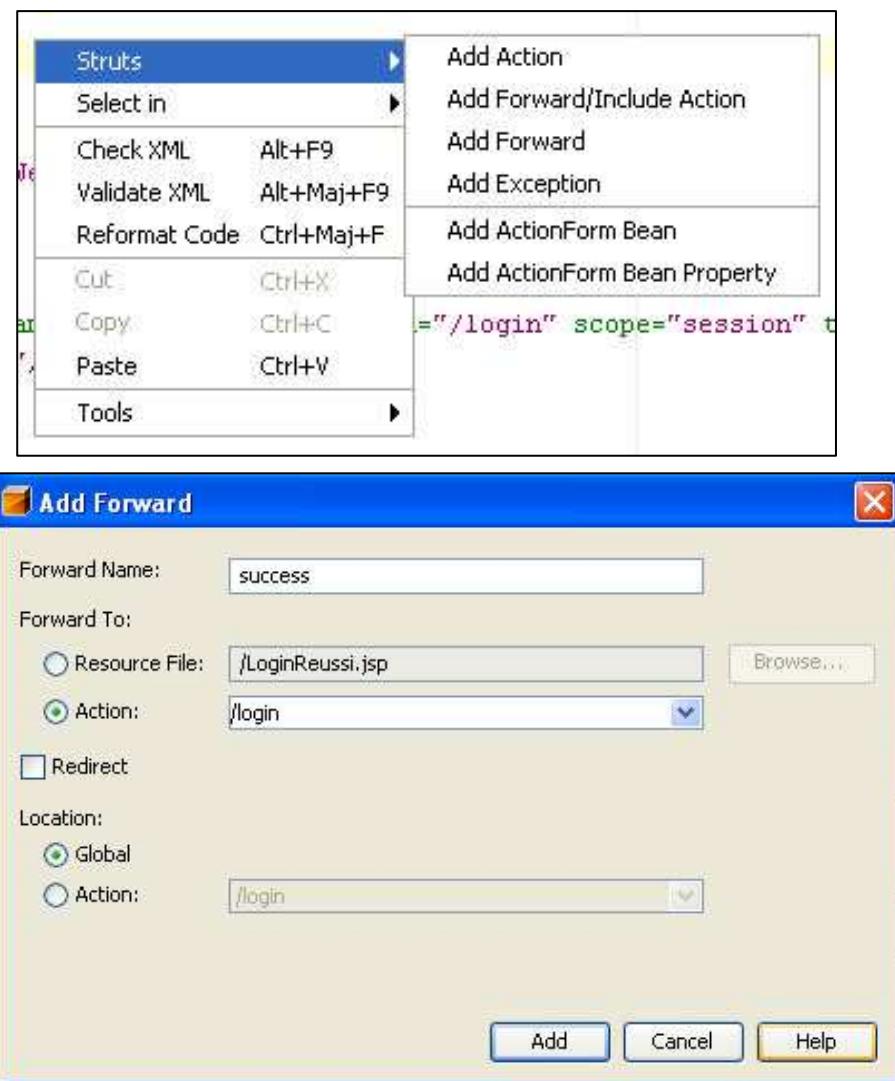
Lorsque sa servlet de contrôle se rend compte que l'on veut utiliser un ActionForm, le framework instancie celui-ci si ce n'est pas déjà fait, l'initialise avec sa méthode reset() (qui, par défaut, ne fait rien) et ses méthodes setXXX() puis, éventuellement (si l'attribut validate est présent), appelle sa méthode validate().

9. Le JSP de réponse

Mais on voit dans la méthode execute() que l'on est redirigé vers une vue associée au nom "success"; il nous faut donc définir un JSP qui sera la page de succès, ainsi que déclaré par un tag forward associé à l'action

```
<forward name="success" path="/LoginReussi.jsp"/>
```

On peut ajouter cette relation avec "success" dans struts-config.xml : après avoir édité celui-ci, un clic droit donne :



Il n'y a plus qu'à construire le JSP en question :

LoginReussi.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Login réussi !</title>
  </head>
  <body>

    <h1>Félicitations ! Vous voilà au sein du plus grand magasin virtuel de la galaxie !!!</h1>

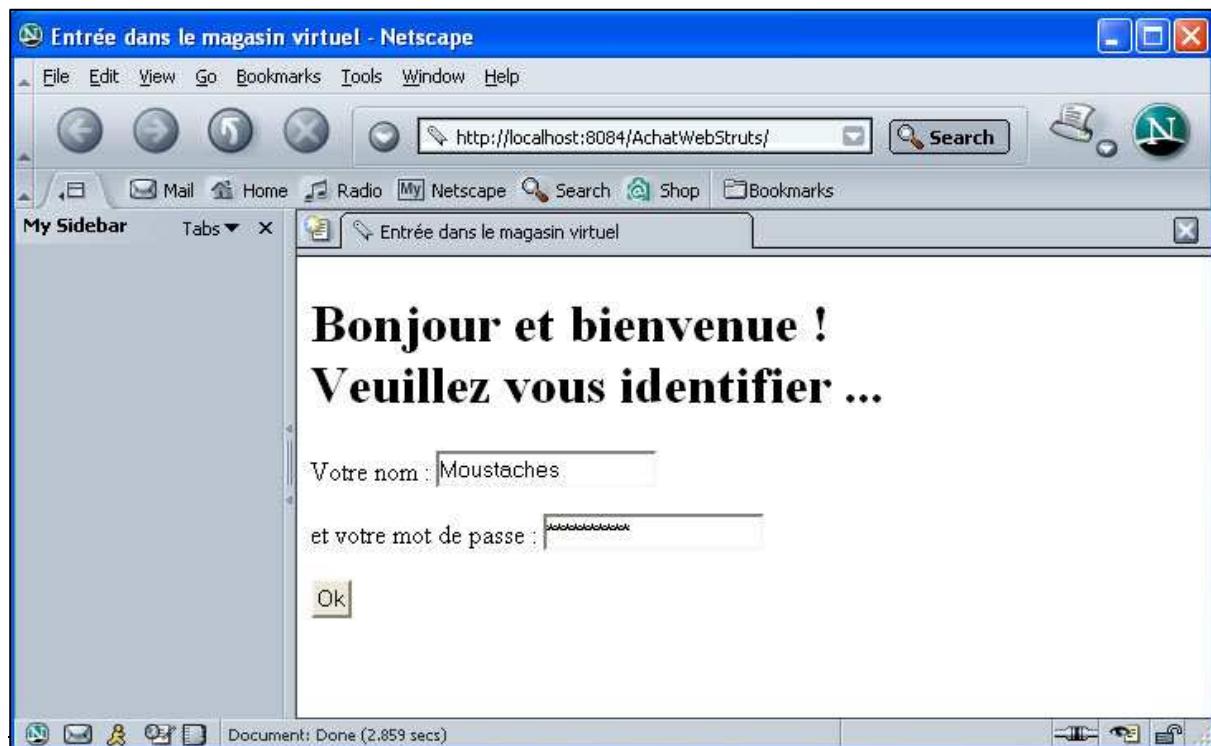
    Vos achats s'effectueront sous l'identité :
    <bean:write name="LoginActionForm" property="name" />

  </body>
</html>
```

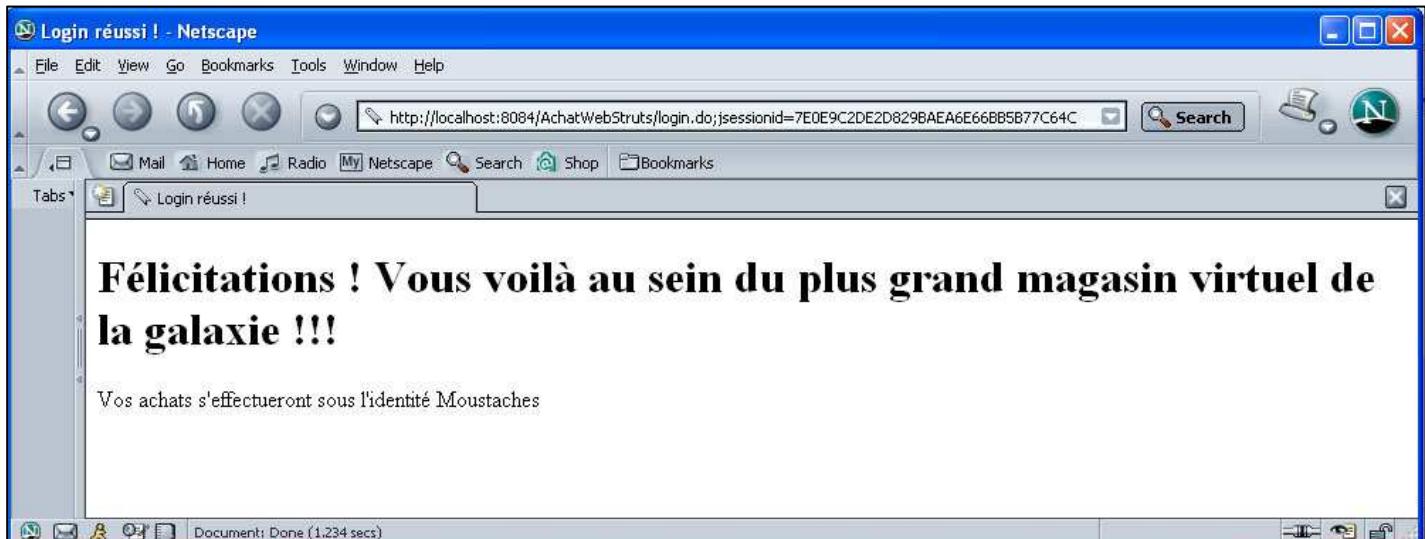
On remarquera l'utilisation de la bibliothèque de tags bean de Struts : on écrit dans le JSP, au moyen du tag <bean:write>, la propriété name du LoginActionForm.

10.L'exécution de l'application Web

Un simple appel de l'application Web :



Après appui sur le bouton "Ok" :



On remarquera :

- ◆ la requête "login.do" dans l'URL;
- ◆ l'id de session ajouté dans cette URL parce que l'utilisateur a bloqué les cookies (utilisés par défaut pour écrire l'id) – c'est de la réécriture d'URL ;-).

11. Un JSP de réponse négative

Affinons quelque peu notre application en ajoutant une véritable vérification du login de l'utilisateur.

1) Tout d'abord, il nous faut modifier notre action de manière à ce qu'elle vérifie effectivement la concordance entre nom d'utilisateur et mot de passe. Pour simplifier (c'est-à-dire pour ne pas effectuer d'accès à une base de données), nous allons supposer que les associations user/password sont mémorisées dans une Hashtable :

LoginAction.java (2)

```
/*
 * LoginAction.java
 */
package com.myapp.struts;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForward;

import java.util.*;
```

```
/***
 * @author vilvens
 */

public class LoginAction extends Action
{
    static Hashtable listeHachee = new Hashtable();

    static
    {
        listeHachee.put("Vilvens","008");
        listeHachee.put("Madani","0010");
        listeHachee.put("Kuty","009");
        listeHachee.put("Wagner","013");
        listeHachee.put("Charlet","0011");
    }

    private final static String SUCCESS = "success";
    private final static String FAILED = "raté";

    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception
    {
        LoginActionForm beanLogin = (LoginActionForm) form;

        if ( beanLogin.getPassword().equals( listeHachee.get(beanLogin.getName()) ) )
            return mapping.findForward(SUCCESS);
        else
            return mapping.findForward(FAILED);
    }
}
```

2) Il nous faut mettre à jour le descripteur de déploiement :

dans **struts-config.xml**

```
...
<action-mappings>
    <action input="/LoginForm.jsp" name="LoginActionForm" path="/login"
        scope="session" type="com.myapp.struts.LoginAction">
        <forward name="success" path="/LoginReussi.jsp"/>
        <forward name="raté" path="/LoginRate.jsp" />
    </action>
</action-mappings>
...
```

3) Il reste à écrire le JSP associé à la réponse d'échec :

LoginRate.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>
<%@taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>

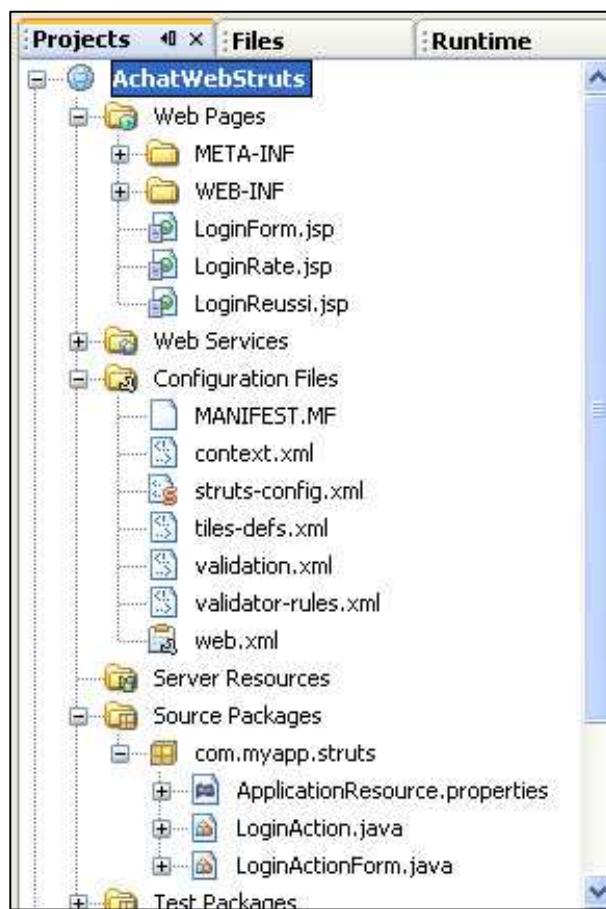
    <h1>Désolé ! Il y a un problème avec votre identification</h1>
    Nous ne trouvons pas le mot de passe que vous avez introduit pour l'identité :
    <bean:write name="LoginActionForm" property="name" /> <p>
    <html:link page="/LoginForm.jsp"> Nouvel essai d'identification </html:link>
    </body>
</html>
```

Un exemple d'exécution avec une première tentative erronée suivie d'une seconde réussie serait :



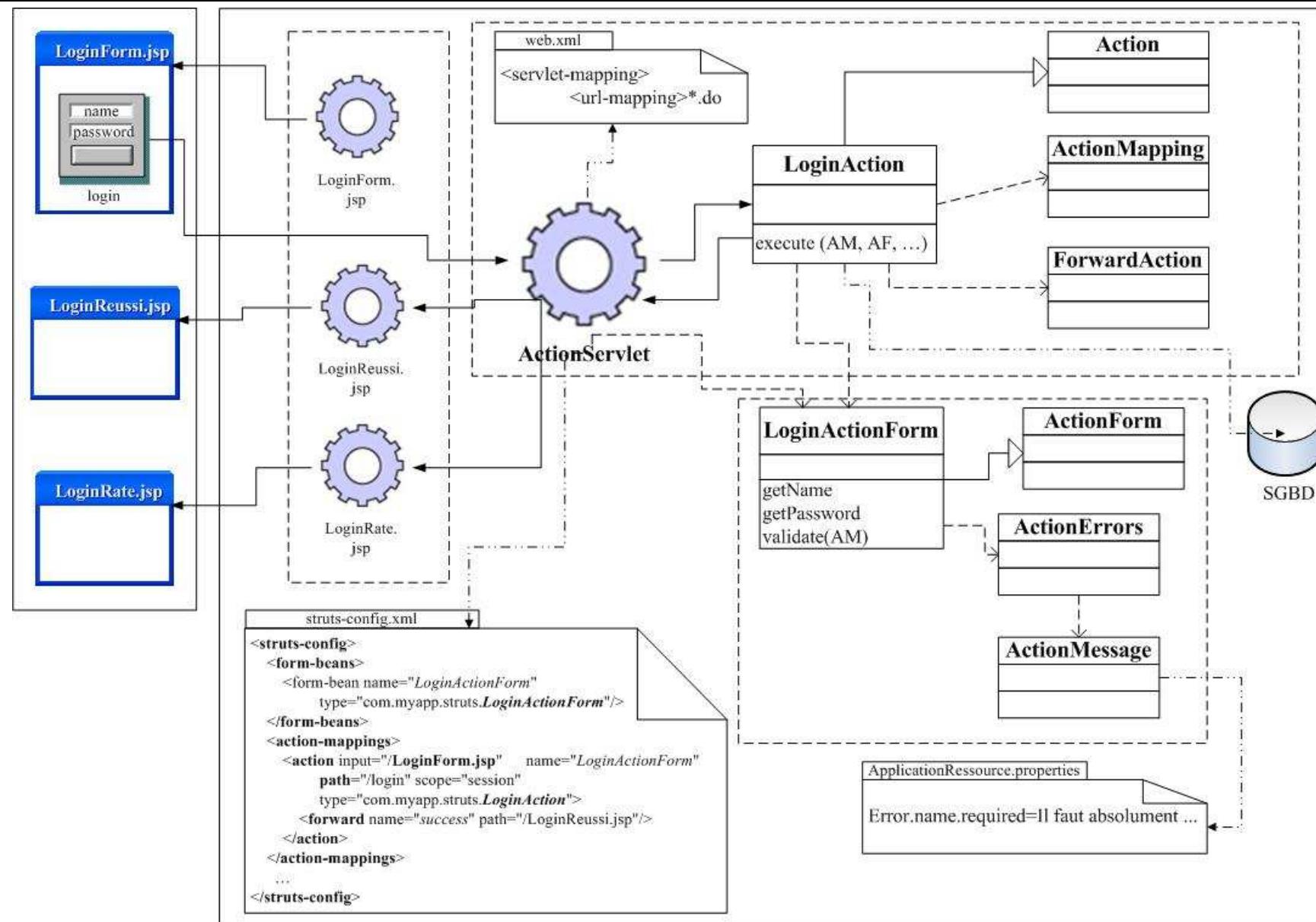


Le projet complet ressemble donc finalement à ceci :



12. Le diagramme pseudo-UML récapitulatif

Nous pouvons à présent rassembler nos idées dans le schéma suivant :



13.Les validations dans les formulaires

Struts offre la possibilité d'opérer un contrôle sur les données introduites dans les formulaires afin de ne pas lancer la méthode execute() de l'action associée si celle-ci n'a aucune chance de réussir. Ainsi, typiquement, on peut contrôler que certaines rubriques indispensables ont effectivement été remplies et renvoyer au formulaire dans la négative. Ceci est possible en utilisant effectivement la méthode invoquée automatiquement par la plate-forme pour la validation :

```
public ActionErrors validate (ActionMapping mapping, HttpServletRequest request)
```

Cette méthode reçoit, bien logiquement, la requête qu'elle doit analyser ainsi que l'habituel objet ActionMapping reflétant les liens décrits dans le descripteur de déploiement de struts. Dans notre cas, nous allons y signifier vers quel JSP il faut dérouter en cas d'entrées invalides :

struts-config.xml (validation)
<pre>... <action-mappings> <action name="LoginActionForm" path="/login" scope="session" type="com.myapp.struts.LoginAction" validate="true" input="/erreurs.do"> <forward name="success" path="/LoginReussi.jsp"/> <forward name="raté" path="/LoginRate.jsp" /> </action> <action path="/erreurs" parameter="/LoginInvalide.jsp" type="org.apache.struts.actions.ForwardAction" /> </action-mappings> ... <message-resources parameter="com/myapp/struts/ApplicationResource"/> ...</pre>

On remarquera l'utilisation, dans le tag action, de l'attribut input permettant de désigner la page à renvoyer en cas d'erreur (désignation par "/erreurs.do" ici).

Ce qu'il faut écrire dans la méthode validate() nous est déjà partiellement fourni par Struts : ce qui concerne la propriété name a en effet été généré automatiquement. Ce n'est pas le cas pour la propriété password parce que nous n'en avons fait une propriété que par après ! Mais l'adaptation n'est pas trop dure à fournir ;-) ...

Plus précisément, on peut constater que la méthode va construire un objet instance de **ActionErrors** qui est une espèce de container associatif reliant une propriété à un message d'erreur instance de la classe ActionMessage, par exemple :

```
"name" → new ActionMessage("error.name.required")
```

Le paramètre passé au constructeur de l'ActionMessage est en réalité à instancier à partir d'une clé à utiliser dans un fichier **ApplicationResource.properties** (son nom est spécifié dans struts-config.xml dans le tag <message-resources>), ce qui permet de trouver le véritable message (le contenu de celui-ci est donc dynamique) :

ApplicationResource.properties

```
errors.header=<UL>
errors.prefix=<LI>
errors.suffix=</LI>
errors.footer=</UL>
...
errors.general=The process did not complete. Details should follow.
errors.token=Request could not be completed. Operation is not in sequence.
welcome.title=Struts Application
welcome.heading=Struts Applications in Netbeans!
welcome.message=It's easy to create Struts applications with NetBeans.

error.name.required=Il faut imp\u00e9rativement indiquer votre nom
error.password.required=Il faut imp\u00e9rativement donner votre mot de passe
```

On devine sans peine que cette manière de faire tend les bras vers l'internationalisation au moyen de bundles. Notre bean sera donc complété comme suit :

LoginActionForm.java (2)

```
package com.myapp.struts;

import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;

public class LoginActionForm extends org.apache.struts.action.ActionForm
{
    ...
    public ActionErrors validate (ActionMapping mapping, HttpServletRequest request)
    {
        System.out.println("Validation des entrées en cours");
        ActionErrors errors = new ActionErrors();
        if (getName() == null || getName().length() < 1)
        {
            errors.add("name", new ActionMessage("error.name.required"));
            System.out.println("Erreur sur le nom");
        }
        if (getPassword()== null || getPassword().length() < 1)
        {
            errors.add("password", new ActionMessage("error.password.required"));
            System.out.println("Erreur sur le mot de passe");
        }
        return errors;
    }
}
```

Le JSP sur lequel on sera renvoyé en cas d'entrées vides dans les deux champs du formulaire de la page de login, et dont le nom est précisé dans struts-config.xml, sera :

LoginInvalide.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>
<%@taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Login : invalide</title>
  </head>
  <body>

    <h2>Des erreurs se sont produites durant votre tentative de login :</h2>
    <html:errors />
    <p>
      <html:link page="/LoginForm.jsp" >
        Retour à la page de login
      </html:link>

    </body>
  </html>
```

On constate que ce JSP récupère l'objet errors généré dans la méthode de validation du bean au moyen du tag de la librairie Struts :

```
<html:errors />
```

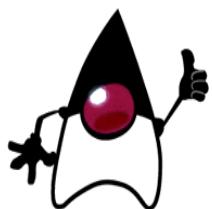
En pratique, en cas d'entrée d'un formulaire vide, on obtient bien:



Remarque

C'est idiot, mais il convient de ne pas oublier le "/" dans le tag html:link :

```
<html:link page="/LoginForm.jsp" >
```



La programmation Web n'a pas de secrets pour nous ;-) A moins de monter d'un niveau dans l'agrégation d'informations ... Vous avez dit "portail" ?

XXXVIII. Une introduction aux portails et aux portlets

The screenshot shows the java.net homepage from September 05, 2008. The main navigation bar includes 'User:', 'Password:', 'Login', 'Register', and 'Login help'. Below the bar, there are links for 'My pages', 'Projects', 'Communities', and 'java.net'. A sidebar on the left titled 'Get Involved' lists options like 'java.net Project', 'Request a Project', 'Project Help Wanted Ads', 'Publicize your Project', and 'Submit Content'. Another sidebar titled 'Get Informed' lists 'About java.net', 'Articles', 'Books', 'Events', 'Also in Java Today', 'java.net Online Books', and 'java.net Archives'. The central content area features a 'Featured Articles' section with links to 'Java Today' and 'Community Corner Podcast'. Below that is a 'Java Mobility' section with a link to 'Java Mobility Podcast 55: Back to School Special'. At the bottom of the page, there's a 'Downloads' section.

J'ai fini par acquérir durablement le sentiment de l'éphémère.

(J. Rostand, Carnet d'un biologiste)

1. La notion de portail

A priori, tout le monde voit ce qu'est un portail [*portal* en anglais]. Et pourtant, en donner une définition précise n'est pas si simple. Donc, **un portail est une application Web qui donne accès à un ensemble de services concernant un thème ou un domaine précis.**

Par exemple, le "portail du monde du logiciel libre" ressemble à ceci :

The screenshot shows the UnixTech portal homepage. The top navigation bar includes 'Accueil', 'Forum', and 'Dogmanet'. A banner at the top says 'Portail du monde du logiciel libre'. The main content area features a large image of penguins. Below the image, the title 'UnixTech' is displayed. A search bar with 'Recherche' and 'Go !' buttons is present. The first article listed is 'Desktop Publishing with OpenOffice.org' dated '25 mai 2004'. It includes a snippet of text: 'Etonnez vos clients, patron ou amis avec cet outil gratuitement disponible que vous avez probablement installé avec votre dernière distribution linux'. Below the article are links for 'Url : Article complet' and 'Site : linuxjournal.com'. A 'tags:aucun' link is also shown. To the right of the article, there is a calendar for March 2008 and a sidebar with a 'Menu' containing links for 'Chattez avec nous!', 'Curriculum Vitae's', 'Entreprises orientées Logiciel Libre', 'Introduction', 'Linux User Groups', 'Mailing List', and 'Soumettre un article'.

- on y voit clairement voisiner des références d'article, un moteur de recherche orienté, un calendrier avec références vers les événements intéressants, la possibilité de poster des questions ou des commentaires, etc.

En fait, les principales fonctionnalités que l'on attend d'un portail sont :

- ◆ **l'agrégation de contenu** : on rassemble au sein d'une même page plusieurs applications Web, qui peuvent partager des ressources ou communiquer entre elles;
- ◆ **la catégorisation des données** : des données de nature différentes sont clairement identifiées, mais disponibles simultanément;
- ◆ **la gestion de contenu** : la plupart des portails propose désormais un CMS (Content Management System) plus ou moins élaboré gérant à la fois les divers types d'informations et d'autre part la chaîne de validation les concernant; l'accès, la présentation et diffusion de ce contenu sont donc pris en charge globalement;
- ◆ **la recherche d'informations** : l'idée est toujours de faciliter le partage d'informations, par exemple en associant aux informations des mots-clés utilisés lors des recherches; on s'oriente donc vers une navigation simplifiée aux informations;
- ◆ **la personnalisation par profil**, par préférence (navigation, couleur, taille de caractère, filtre..) : l'idée est que l'utilisateur retrouve sa page comme il l'avait configurée lors de sa dernière visite;
- ◆ **le travail collaboratif** ; il s'agit d'un agenda, de forums, wikis, blogs; de manière plus générale, une bonne communication : e-mails, chats;
- ◆ **un mécanisme d'authentification unique** (un Single Sign On – SSO) qui permet à l'utilisateur de s'authentifier une seule fois pour plusieurs applications;
- ◆ **l'intégration de services, d'application et de business intelligence** : fondamentalement, il s'agit de collecter, modéliser et finalement présenter de manière synthétique et compréhensible un gros volume de données afin que les décideurs puissent prendre les bonnes décisions en toute connaissance de cause.

Le site didactique <http://portailweb20.wordpress.com/2007/09/01/les-principales-fonctionnalites-d'un-portail/> donne une idée des multiples possibilités :

The screenshot shows a WordPress blog post titled "Les principales fonctionnalités d'un portail". The header features a large image of a coastal landscape with green hills and a blue sky. Below the header, there's a sidebar with a "À propos" button, a list of post categories, and a sidebar with a calendar for September 2007. The main content area contains the post text and a footer with page numbers.

Portails Web 2.0

À propos

Les principales fonctionnalités d'un portail

Posted by: portailweb20 | septembre 1, 2007

Recherche

Agrégation de contenu
Catégorisation des données
Accès, présentation et diffusion de contenu
Gestion de contenu (la plupart des portails propose désormais un CMS plus ou moins élaboré)
Recherche d'informations
Navigation simplifiée aux informations
Personnalisation par profil, par préférence (navigation, couleur, taille de caractère, filtre..)
Notification personnalisée ou non
Travail collaboratif (agenda, forum, wiki, blog..)
Communication (e-mail, chat..)
Intégration de services, d'application et de business intelligence
Administration (et notamment le Single Sign On) et la sécurité
Infrastructure

septembre 2007

L	Ma	Me	J	V	S	D
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
<< août >>			oct >>			

CATÉGORIES

Page 200

On voit donc apparaître ici le **Web 2.0**, concept bien flou mais qui veut marquer un changement radical dans les applications Web en s'appuyant sur

- ◆ des technologies bien connues mais, jusqu'à ces derniers temps, pas vraiment utilisées conjointement : HTTP et (X)HTML, URIs, CSS, JavaScript, XML (avec DOM), RSS, services Web, ...
- ◆ des méthodes d'utilisation nouvelles basées sur un caractère interactif et collaboratif : blogs, wikis, systèmes de "tagging" pour catégoriser les ressources Web, réseaux sociaux, ...

Une vision des choses est "utilisatrice" résume les (gros gros) souhaits des utilisateurs :



2. La JSR 168

La **JSR 168** (Java Specification Request) est la spécification des portlets définissant le fonctionnement des interactions entre les conteneurs de portlets et les portlets elles-mêmes. Cette JSR fait partie du **JCP** (le Java Community Process). L'objectif est d'assurer l'interopérabilité entre les portlets et les portails qui les utilisent : on veut certifier que toute portlet développée en Java puisse s'exécuter sur tout serveur d'application compatible J2EE. Outre la définition d'une API standard, ceci implique aussi qu'il faut supporter des concepts comme la personnalisation (et donc l'internationalisation), l'agrégation, la présentation, le multi-terminal.

On trouve à l'origine de cette spécification Sun et IBM. Mais ils ont été rejoints par de nombreux autres grands noms comme Apache Software Foundation, Oracle, Novell, HP, Macromedia, Sybase, etc. La version 1.0 date de fin 2003.

Des exemples de portails JSR 168 sont Apache Jetspeed, BEA Weblogic Portal, IBM Websphere Portal et surtout Liferay Portal et **JBoss Portal**.

The Java Community Process(S...)

Community Development of Java Technology Specifications

List of all JSRs **JCP2** **Spec Lead Guide** **Expert Group Login** **Calendar** **FAQ**

JSRs: Java Specification Requests
JSR 168: Portlet Specification

To enable interoperability between Portlets and Portals, this specification will define a set of APIs for Portal computing addressing the areas of aggregation, personalization, presentation and security.

Status: Final	Stage	Start	Finish
Final Release	Download page	27 Oct, 2003	
Final Approval Ballot	View results	23 Sep, 2003	06 Oct, 2003
Proposed Final Draft 2	Download page	08 Sep, 2003	
Proposed Final Draft	Download page	27 Aug, 2003	
Public Review	Download page	17 Jul, 2003	16 Aug, 2003
Community Draft Ballot	View results	17 Jun, 2003	23 Jun, 2003
Community Review	Login page	16 Apr, 2003	23 Jun, 2003
Expert Group Formation		12 Feb, 2002	21 Aug, 2002
JSR Review Ballot	View results	29 Jan, 2002	11 Feb, 2002

JCP version in use: [2.1](#)
Java Specification Participation Agreement version in use: 1.0
Please direct comments on this JSR to: jsr-168-comments@jcp.org

Specification Lead
Stefan Hepper IBM

Expert Group
Apache Software Foundation
Boeing Art Technology Group Inc.(ATG)
Borland Software Corporation BEA Systems
Broadvision Inc.

Une autre JSR, la **JSR 286**, termine sa gestation : elle fera évoluer la spécification des portlets au niveau 2.0 avec

- ◆ le support de la gestion des événements entre portlets, leur permettant ainsi de communiquer entre elles;
- ◆ l'utilisation des filtres que les servlets récentes connaissent déjà;
- ◆ un cache amélioré;
- ◆ etc.

3. Portail et portlets

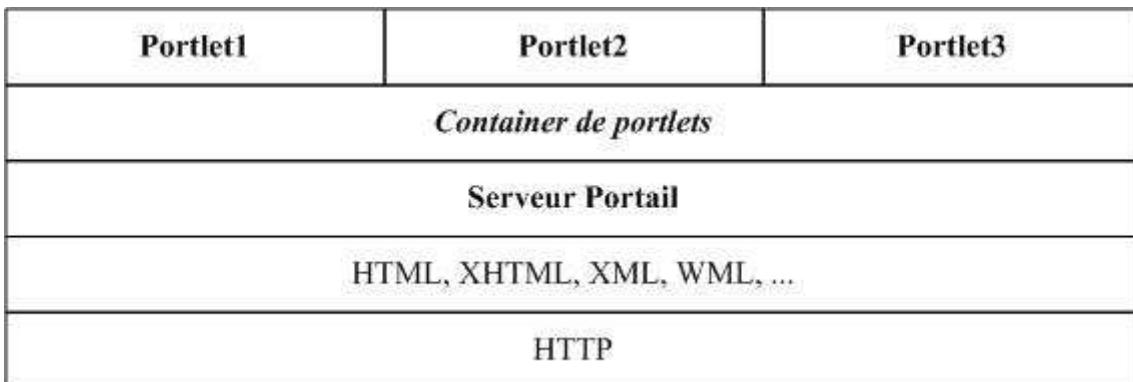
Un portail est donc clairement formé d'une certain nombre de composantes apparaissant chacune dans une fenêtre. Sur une plate-forme Java JSR 168 compatible, chacune de ces composantes est générée, statiquement ou dynamiquement, par un composant Web appelé **portlet** dont le rôle est de fournir un fragment de la page HTML complète :

une **portlet** est donc un composant Web qui génère statiquement ou dynamiquement un morceau de code HTML qui, réuni avec d'autres codes similaires, formera la page complète du portail – celui-ci utilise donc un **container** qui rassemble les portlets qui définissent le portail.

Donc, quand un client effectue une requête sur un portail

- ◆ une requête HTTP parvient au portail ;
- ◆ celui-ci vérifie si elle concerne l'une des portlets du portail;
- ◆ si c'est le cas, il passe la main au container de portlets qui transmettra la requête à la portlet concernée;
- ◆ la portlet génère les fragments demandés et les transmet au container;
- ◆ le portail rassemble les composantes fournies par les autres portlets;
- ◆ la réponse HTTP qu'il va ensuite former est envoyée au client.

Logique ... Nous sommes donc en face d'une modèle en couches assez simple :



4. Les caractéristiques d'une portlet

4.1 Le cycle de vie

En programmation Java, une portlet est l'instance d'une classe qui implémente l'interface **Portlet** (du package javax.portlet) :

```
public interface Portlet
{
    public void init(PortletConfig config) throws PortletException;
    public void processAction(ActionRequest request, ActionResponse response)
        throws PortletException, java.io.IOException;
    public void render(RenderRequest request, RenderResponse response)
        throws PortletException, java.io.IOException;
    public void destroy();
}
```

La classe **GenericPortlet**, implémentant l'interface, est également fournie : il suffira donc en pratique d'hériter de cette classe et de redéfinir les méthodes pour lesquelles c'est nécessaire.

Les méthodes qui y sont déclarées reflètent évidemment l'interaction avec l'utilisateur :

- ◆ traitement de la requête utilisateur avec **processAction()** : les paramètres de requête et de réponse implémentent les interfaces ActionRequest et ActionResponse (qui jouent un rôle analogue aux objets request et response bien connus des programmeurs de servlets);
- ◆ génération des fragments avec **render()** : à nouveau avec des paramètres RenderRequest et RenderResponse qui sont en fait des frères des précédents (des interfaces

PortletRequest et PortletResponse chapeautent le tout); son action dépend du mode de la portlet (voir plus loin);

- ◆ réaction en cas d'initialisation ou de destruction de la portlet (**init()** et **destroy()**).

Plus précisément, si un portail est matérialisé par un container à n portlets, l'accès d'un client :

- ◆ provoque l'appel de la méthode init() des n portlets;
- ◆ si le client agit sur une portlet i, la méthode processAction() de celle-ci est appelée;
- ◆ la méthode render() de chaque portlet est appelée, renvoyant chacun un fragment HTML que le container rassemble.

Le portail n'a plus qu'à renvoyer le résultat au client.

4.2 Portlets et servlets

Bien sûr, le fait de voir une portlet renvoyer du code HTML au client fait penser au mécanisme des servlets. De fait, une portlet et une servlet sont toutes deux écrites en Java, sont basées sur un mécanisme requête/réponse (le plus souvent avec http sous-jacent), s'exécutent dans un container (mais qui n'est pas de même nature) et fournissent toutes les deux du contenu statique ou dynamique.

Mais des différences apparaissent tout aussi clairement :

- ◆ une servlet retourne une page entière, une portlet ne renvoie que des fragments de pages;
- ◆ une portlet ne peut être appelée directement à partir d'un navigateur;
- ◆ une portlet peut être instanciée plusieurs fois.

Aussi, une portlet a un "mode" ...

4.3 Les modes des portlets

En pratique, une portlet possède dans ses caractéristiques un "mode" qui détermine un contenu généré différent :

- ◆ en mode **VIEW** : le fonctionnement est le même que celui d'une page Web classique, soit afficher des informations et interagir avec l'utilisateur; ce mode doit toujours être implémenté;
- ◆ en mode **EDIT** : le fonctionnement est alors plutôt celui d'un formulaire, en ce sens que l'utilisateur peut modifier le comportement de la portlet en modifiant certains paramètres (par exemple, sur un site touristique, le lieu dont on montre une photo); l'implémentation de ce mode est facultative;
- ◆ en mode **HELP** : il s'agit de fournir de l'aide à l'utilisateur sur la manière d'utiliser la portlet; l'implémentation de ce mode est également facultative.

La méthode render() de GenericServlet délègue en fait son travail d'affichage à l'une des trois méthodes **doView()**, **doEdit()** et **doHelp()**, selon le mode de la portlet (nous y reviendrons plus loin).

Mais les fragments HTML fournis par les portlets en sont pas rassemblés de manière anarchique ...

4.4 Les modes des fenêtres

Le portail rassemble les différents fragments générés par chacune de ses portlets dans des **fenêtres** distinctes comportant un titre et des boutons de contrôle permettant de modifier le "mode" de la fenêtre; celui-ci peut être

- ◆ **NORMAL** : la fenêtre de la portlet partage avec les fenêtres des autres portlets l'espace disponible de la page du portail;
- ◆ **MAXIMIZED** : seule la fenêtre considérée est affichée sur le site du portail;
- ◆ **MINIMIZED** : la fenêtre prend "le minimum" de place, ce qui peut pouvoir dire à la limite qu'elle n'est plus visible.

Il nous faut à présent un serveur d'applications pour héberger ce petit monde ...

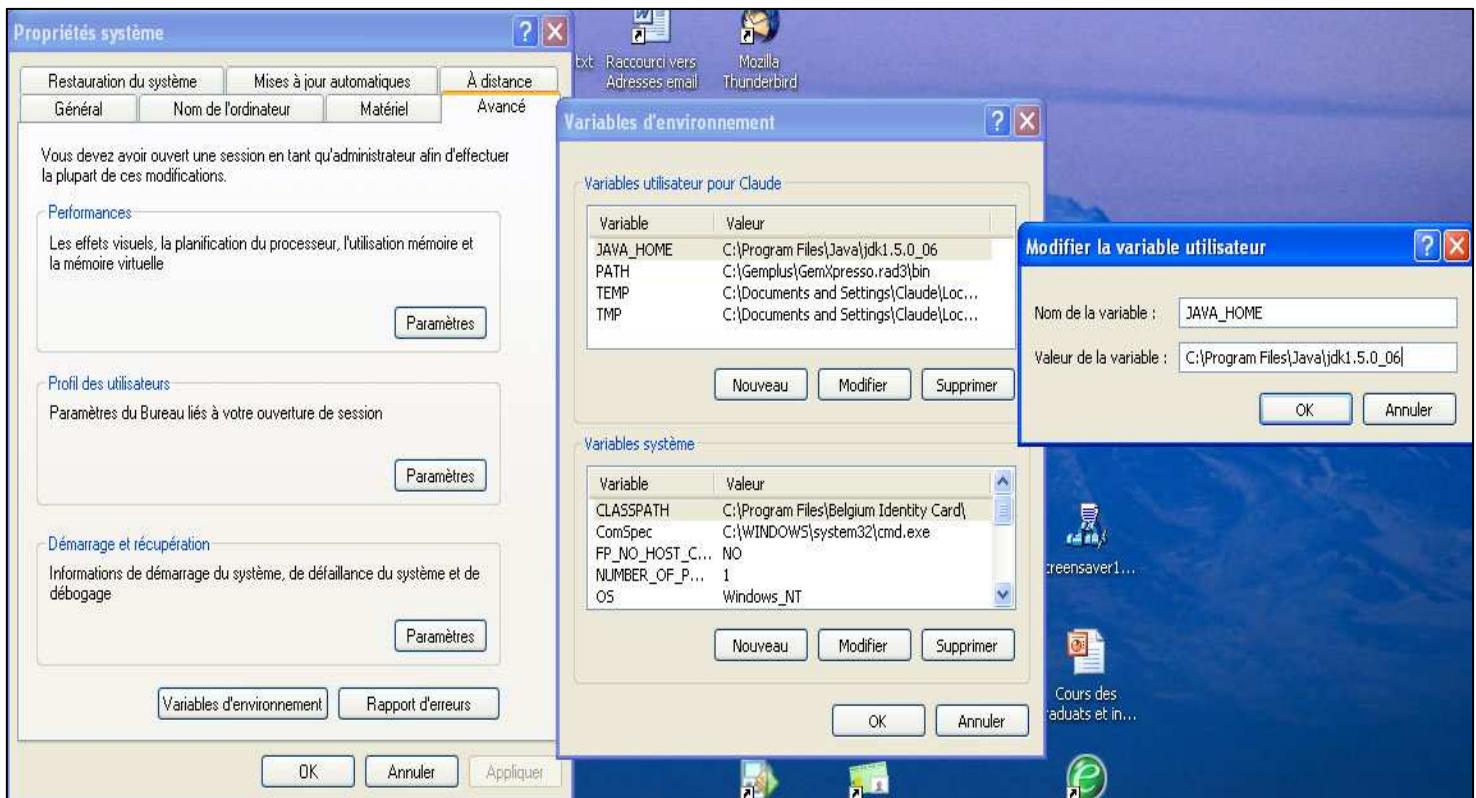
5. Installation et intégration de JBoss Portal

JBoss Portal est sans doute l'un des serveurs d'application pour portails les plus répandus. Nous allons l'intégrer à notre environnement de travail habituel NetBeans pour ensuite développer une portlet basique.

Pour downloader JBoss Portal, il suffit d'aller sur <http://labs.jboss.com/index.html> : on récupère ainsi un fichier jboss-portal-2.6.4-bundled.zip. On décomprime celui-ci puis on lance le run.bat du répertoire bin :

```
JAVA_HOME is not set. Unexpected results may occur.  
Set JAVA_HOME to the directory of your local JDK to avoid this message.  
=====  
JBoss Bootstrap Environment  
JBoss_HOME: C:\JBoss portal\jboss-portal-2.6.4-bundled\jboss-portal-2.6.4  
JAVA: java  
JAVA_OPTS: -Dprogram.name=run.bat -server -Xms128m -Xmx512m -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000  
CLASSPATH: C:\JBoss portal\jboss-portal-2.6.4-bundled\jboss-portal-2.6.4\bin\run.jar  
=====  
Error: no 'server' JVM at 'C:\Program Files\Java\jre1.5.0_06\bin\server\jvm.dll'  
Appuyez sur une touche pour continuer... _
```

Problème ⊖ Il manque le répertoire server avec une JVM dans le JRE. On pourrait recopier le répertoire server qui est dans le jdk/jre/bin mais, plus élégamment, nous allons définir la variable d'environnement JAVA_HOME :



On recommence :

```
14:37:18,718 INFO [PortletAppDeployment] Parsing /portal-news-samples/jboss-portlet.xml
14:37:19,250 INFO [TomcatDeployer] deploy, ctxPath=/jmx-console, warUrl=.../deploy/jmx-console.war/
14:37:19,593 INFO [Http11Protocol] Démarrage de Coyote HTTP/1.1 sur http-127.0.1-8080
14:37:19,718 INFO [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127.0.0.1-8009
14:37:19,765 INFO [Server] JBoss (MX MicroKernel) [4.2.2.GA <build: SVNTag=JBoss_4_2_2_GA date=200710221139>] Started in 2m:15s:140ms
```

et JBoss est donc démarré. On peut le tester par <http://localhost:8080/portal> :

Greetings!

This is a basic installation of **JBoss Portal 2.6.4-GA**. You may log in at any time, using the [Login](#) link at the top-right of this page, with the following credentials:
user/user or admin/admin

If you are in need of guidance with regards to navigating, configuring, or operating the portal, please view our online documentation.

User portlet

Bienvenue S'enregistrer

You n'êtes pas connecté.

Support Services

JBoss Inc. offers various support services tailored to fit your needs. Explore support and service options for JBoss Portal.

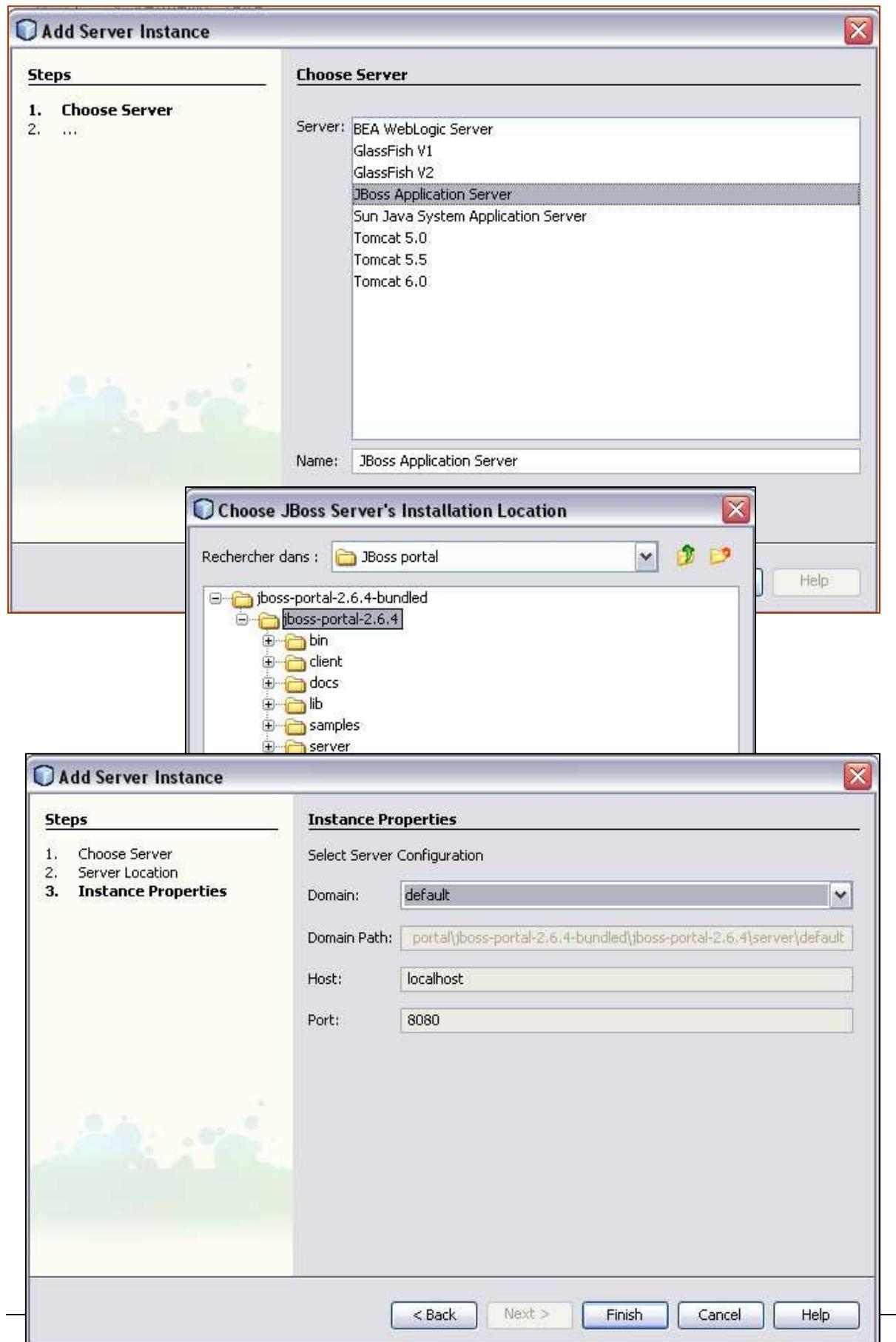
PortletSwap

Portletswap.com is an open community sponsored by JBoss, Inc. to facilitate the exchange of portlets and layouts for use in JBoss Portal.

Project Information

Learn more about the JBoss Portal project, on-going development, open issues, and our user and developer communities.

Nous pouvons à présent intégrer JBoss Portal dans Netbeans : dans l'onglet Services, un clic droit sur le nœud Servers permet de choisir l'habituel "Add server" :



Voilà donc le serveur intégré – mais cela ne suffit pas ...

6. Intégration des modules Portlets dans NetBeans

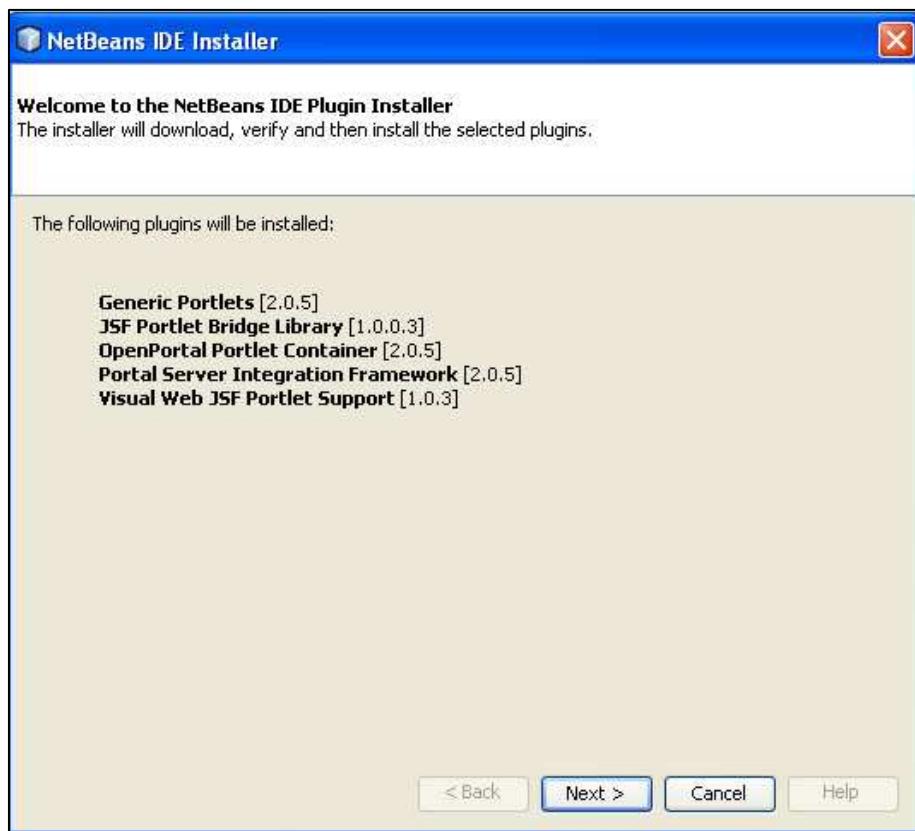
Car si NetBeans connaît à présent le serveur, il n'en est pas devenu pour autant capable de prendre en charge la création de portlets. Pour cela, il nous faut y ajouter un plugin que nous trouverons en <http://portalpack.netbeans.org/>. On récupère ainsi un portal-pack-plugin-2_0.zip



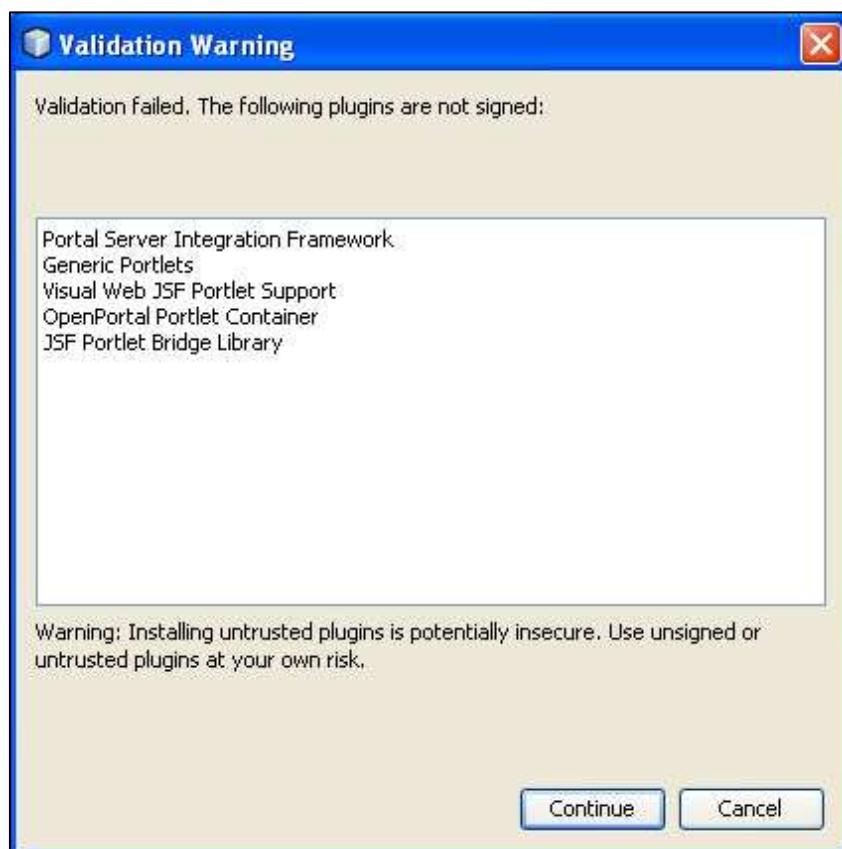
On décomprime. Puis, dans NetBeans, on choisit Tools → Plugins. Dans l'onglet Downloaded, on commence par sélectionner les 5 plugins :

The screenshot shows two instances of the NetBeans Plugins dialog. The top instance is the 'Add Plugins...' dialog, which is open over the main 'Plugins' window. It displays a list of five .nbm files from a folder named 'portalpack20'. The bottom instance is the main 'Plugins' window, showing the 'Downloaded' tab selected. Five plugins are listed with checkboxes checked: 'OpenPortal Portlet Container', 'JSF Portlet Bridge Library', 'Generic Portlets', 'Portal Server Integration Framework', and 'Visual Web JSF Portlet Support'. To the right of the list, detailed information is shown for the 'OpenPortal Portlet Container' plugin, including its version (2.0.5), date (17/04/08), source (org-netbeans-modules-portalpack-servers-opensourcepc.nbm), and homepage (<http://portalpack.netbeans.org>). A 'Plugin Description' section provides a brief overview of the plugin's purpose. At the bottom left of the main window, a message says '5 plugins selected'.

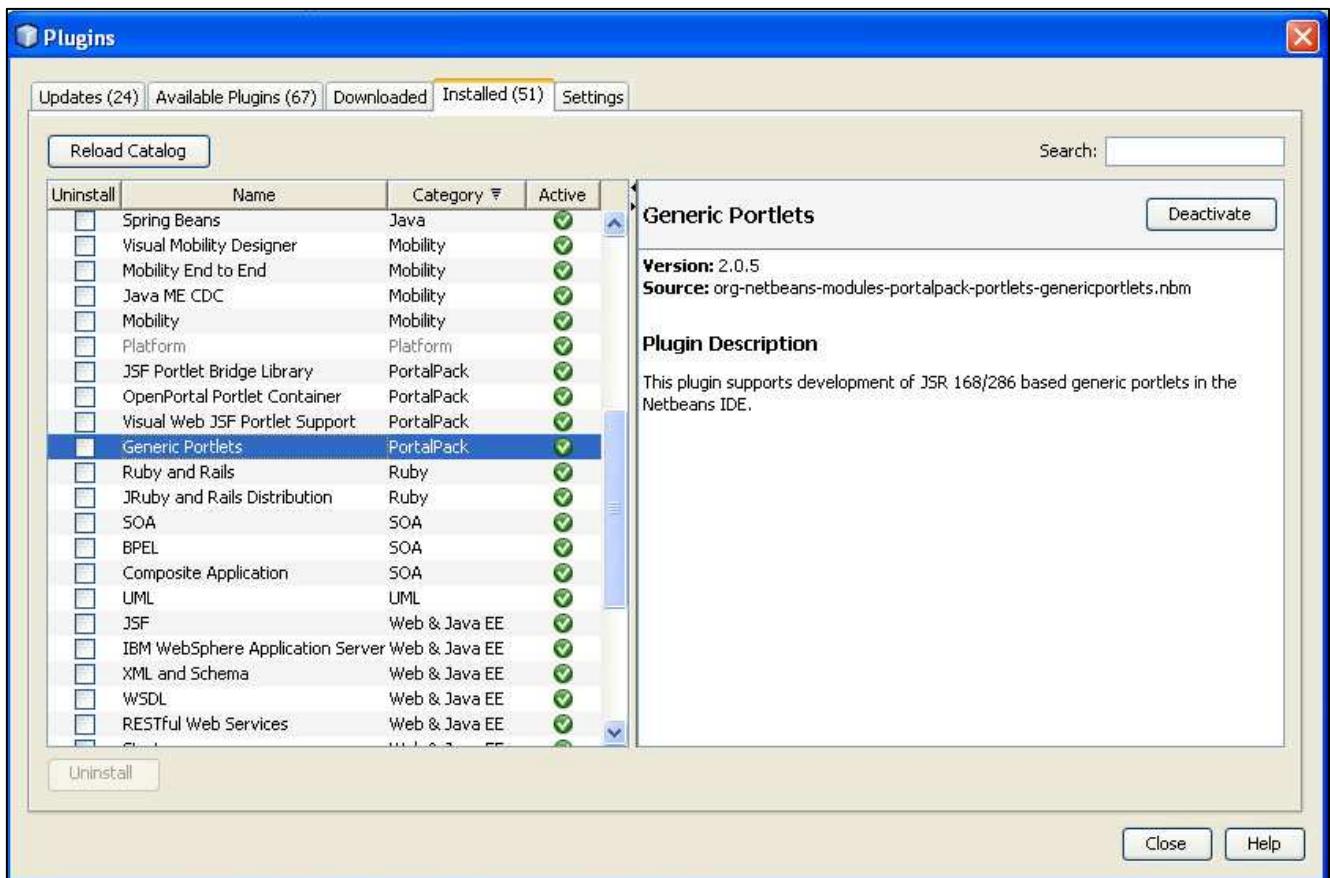
On appuie sur Install :



On ne tient pas compte de l'avertissement :



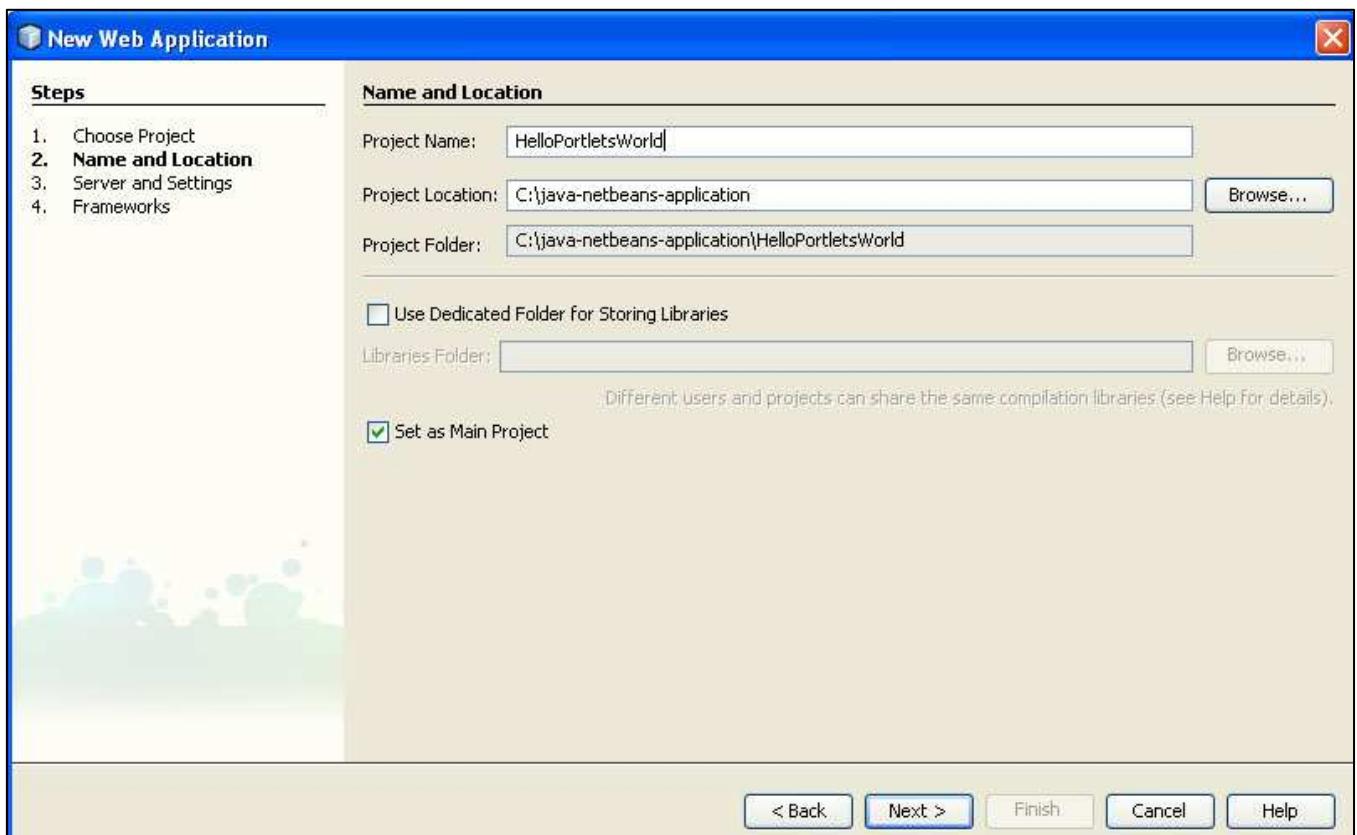
C'est fait, comme on peut le vérifier :

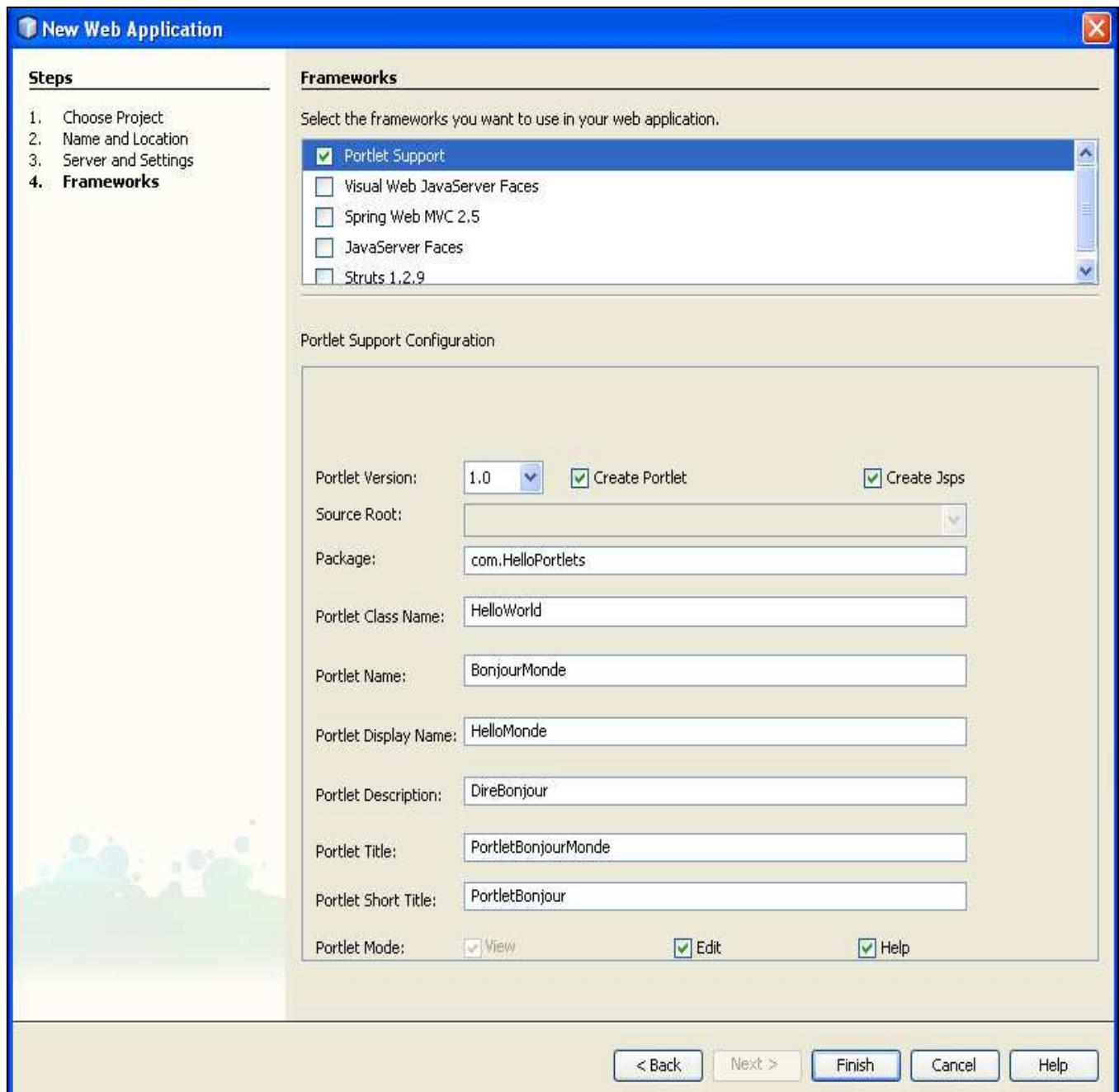


7. La création d'une portlet avec NetBeans

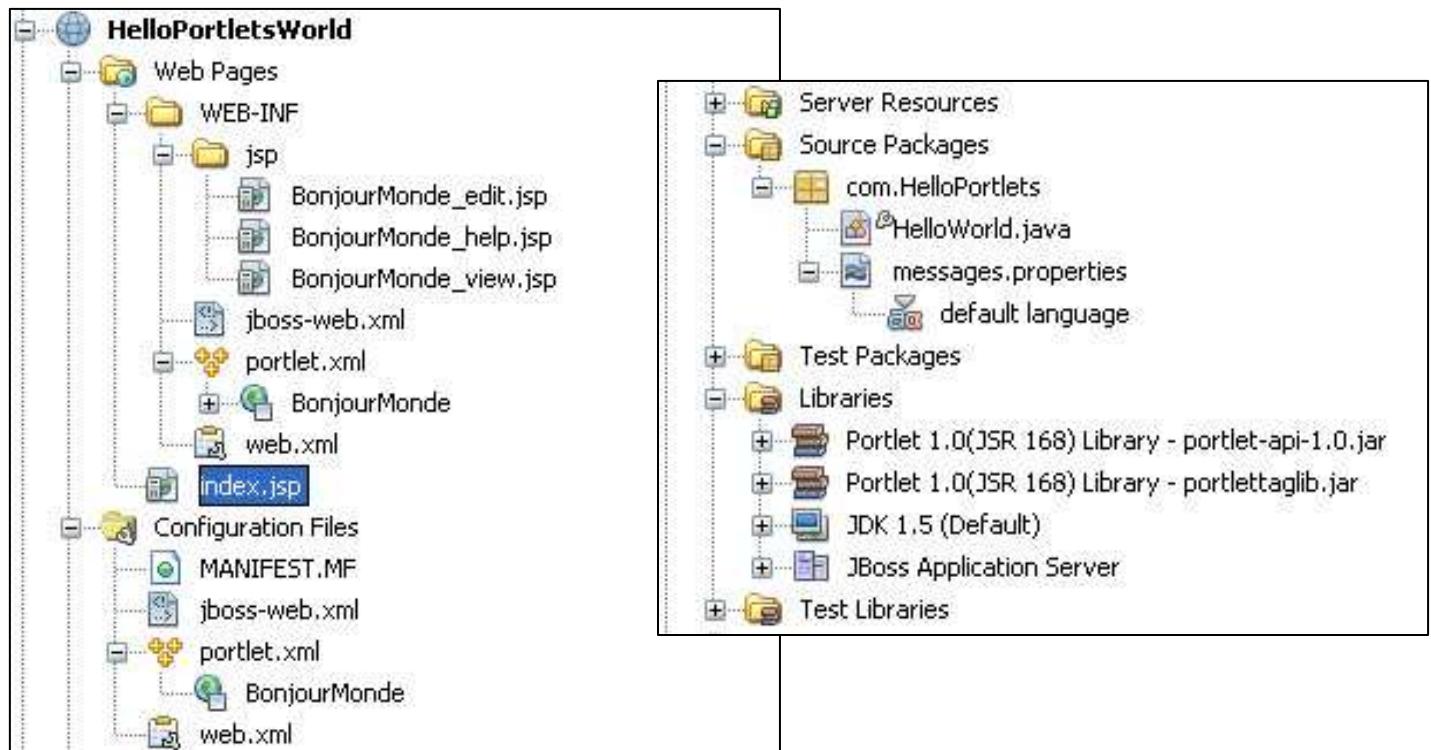
7.1 Une application Web particulière

Dans NetBeans 6, une application Web structurée en portail (une "Portlets Application") est créée comme une application Web classique, mais en apportant les précisions particulières aux portlets :





On se retrouve avec un projet bien peuplé :



7.2 Le code de la portlet générée

La portlet générée par NetBeans a l'aspect suivant :

HelloWorld.java

```
package HelloPortlets;  
  
import javax.portlet.GenericPortlet;  
import javax.portlet.ActionRequest;  
import javax.portlet.RenderRequest;  
import javax.portlet.ActionResponse;  
import javax.portlet.RenderResponse;  
import javax.portlet.PortletException;  
import java.io.IOException;  
import javax.portlet.PortletRequestDispatcher;  
  
public class HelloWorld extends GenericPortlet  
{  
    public void processAction(ActionRequest request, ActionResponse response)  
        throws PortletException, IOException { }  
  
    public void doView(RenderRequest request, RenderResponse response)  
        throws PortletException, IOException  
    {  
        response.setContentType("text/html");  
        PortletRequestDispatcher dispatcher =  
            getPortletContext().getRequestDispatcher("/WEB-INF/jsp/HelloWorld_view.jsp");  
        dispatcher.include(request, response);  
    }  
}
```

```
public void doEdit(RenderRequest request, RenderResponse response)
    throws PortletException, IOException
{
    response.setContentType("text/html");
    PortletRequestDispatcher dispatcher =
        getPortletContext().getRequestDispatcher("/WEB-INF/jsp/HelloWorld_edit.jsp");
    dispatcher.include(request, response);
}

public void doHelp(RenderRequest request, RenderResponse response)
    throws PortletException, IOException
{
    response.setContentType("text/html");
    PortletRequestDispatcher dispatcher =
        getPortletContext().getRequestDispatcher("/WEB-INF/jsp/HelloWorld_help.jsp");
    dispatcher.include(request, response);
}
```

Sans surprise, notre portlet implémente la classe **GenericPortlet** qui apporte avec elle, outre les méthodes de l'interface portlet et les méthodes doView(), doEdit() et doHelp() (qui, pour rappel, sont appelées par render() selon le mode), diverses méthodes additionnelles dont

public PortletContext **getPortletContext()**

qui fournit un objet implementant l'interface PortletContext. Celui-ci donne accès au container des portlets et va donc lui permettre

- ◆ de récupérer ou d'enregistrer des données partagées entre les diverses portlets (et aussi servlets) de l'application :

Object **getAttribute** (String name)
void **setAttribute** (String name, Object object)

- ◆ de récupérer des ressources (ou plutôt leur URL) qui ont été mappées dans l'application avec un nom propre au contexte (débutant toujours par "/") :

URL **getResource** (String path) throws java.net.MalformedURLException

- la méthode

InputStream **getResourceAsStream**(String path)

permettant même d'obtenir le flux associé;

- ◆ de récupérer des paramètres d'initialisation définis dans l'application Web :

java.lang.String **getInitParameter** (java.lang.String name)

Tout ceci a évidemment un petit air d'API de servlet, non ? De plus, et à propos de servlets, la méthode :

PortletRequestDispatcher getRequestDispatcher (String path)

va permettre à la portlet de passer la main (totalement ou partiellement) à un JSP ou à une servlet pour générer le fragment HTML. En effet, le paramètre de cette méthode permet de les désigner pour en retirer un contenu que le dispatcher va inclure dans la réponse HTTP au moyen de son unique méthode :

```
void include(RenderRequest request, RenderResponse response)
            throws PortletException, IOException
```

C'est ce que font les trois méthodes doXXX() qui se servent de trois JSP dont le nom est celui de la portlet complété de "_view", "_edit" ou "_help" – par exemple :

BonjourMonde_view.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%-- Uncomment below lines to add portlet taglibs to jsp
<%@ page import="javax.portlet.*"%>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>

<portlet:defineObjects />
<%PortletPreferences prefs = renderRequest.getPreferences();%>
--%>

<b>
    BonjourMonde - VIEW MODE
</b>
```

Pas de <body> ou de <html> : il s'agit bien d'un fragment !

7.3 Les fichiers de déploiements

Comme pour les applications Web classiques, une Portlet application utilise un certain nombre de fichiers de déploiements, dont **web.xml** et **portlet.xml**. En fait, le fichier web.xml ne présente aucune particularité : il sert toujours à mapper les servlets de l'application et à définir les paramètres d'initialisation. Le deuxième fichier est plus spécifique et son nom dit bien à quoi il sert : il définit les paramètres de chaque portlet de l'application, précisant par exemple quel(s) mode(s) le portail doit prévoir pour elles. Pour notre exemple :

portlet.xml

```
<?xml version='1.0' encoding='UTF-8' ?>

<portlet-app xmlns='http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
        http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd' version='1.0'>
```

```

<portlet>
    <description>DireBonjour</description>
    <portlet-name>BonjourMonde</portlet-name>
    <display-name>HelloMonde</display-name>
    <portlet-class>com.HelloPortlets.HelloWorld</portlet-class>
    <expiration-cache>0</expiration-cache>
    <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
        <portlet-mode>EDIT</portlet-mode>
        <portlet-mode>HELP</portlet-mode>
    </supports>
    <resource-bundle>com.HelloPortlets.messages</resource-bundle>
    <portlet-info>
        <title>PortletBonjourMonde</title>
        <short-title>PortletBonjour</short-title>
    </portlet-info>
</portlet>
</portlet-app>

```

La compréhension de ce fichier est immédiate. On remarquera simplement que les tags `<portlet-mode>` qui vont permettre au serveur portail de savoir quels boutons il doit faire apparaître sur la fenêtre gérant la portlet considérée. Signalons que le tag `<portlet>` connaît aussi le sous-tag

```

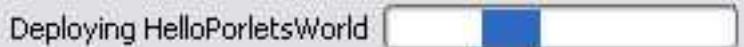
<init-param>
    <name>....</name>
    <value>....</value>
</init-param>

```

qui définit des paramètres d'initialisation récupérables dans la méthode init() de la portlet.

7.4 Le déploiement de la portlet sur le serveur JBoss Portal

Un build de notre application génère, comme d'habitude, un fichier war dans le répertoire dist. Mais il faut encore, pour voir le résultat final, effectuer deux tâches ... Tout d'abord, déployer la portlet sur le serveur JBoss Portal : pour cela, il suffit d'un clic droit sur le nœud du projet suivi du choix "Undeploy and Deploy" :



```

=====
JBoss Bootstrap Environment
JBOSS_HOME: C:\JBoss portal\jboss-portal-2.6.4-bundled\jboss-portal-2.6.4
JAVA: C:\Program Files\Java\jdk1.5.0_06\bin\java
JAVA_OPTS: -Dhttp.nonProxyHosts="localhost|127.0.0.1|private-v4zrgl4l" -
Dprogram.name=run.bat -server -Xms128m -Xmx512m -
Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000

```

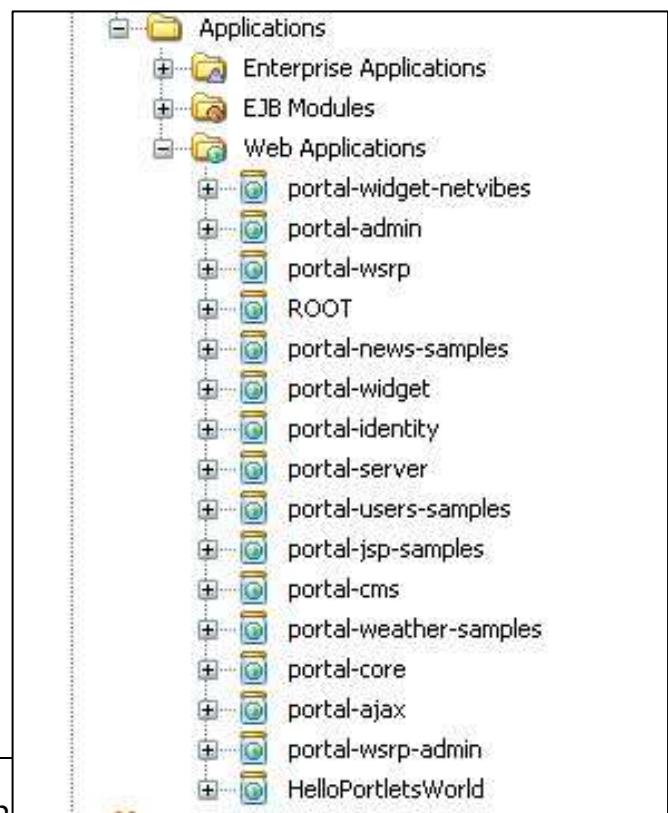
```
CLASSPATH: C:\Program Files\Java\jdk1.5.0_06\lib\tools.jar;C:\JBoss portal\jboss-portal-2.6.4-bundled\jboss-portal-2.6.4\bin\run.jar
=====
17:44:02,656 INFO [Server] Starting JBoss (MX MicroKernel)...
...
...
17:47:55,187 INFO [TomcatDeployer] deploy, ctxPath=/jmx-console,
warUrl=.../deploy/jmx-console.war/
17:47:56,656 INFO [Http11Protocol] Démarrage de Coyote HTTP/1.1 sur http-127.0.0.1-
8080
17:47:56,734 INFO [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127.0.0.1-8009
17:47:56,765 INFO [Server] JBoss (MX MicroKernel) [4.2.2.GA (build:
SVNTag=JBoss_4_2_2_GA date=200710221139)] Started in 3m:53s:656ms
17:48:16,625 INFO [TomcatDeployer] deploy, ctxPath=/HelloPorletsWorld,
warUrl=.../tmp/deploy/tmp46947HelloPorletsWorld-exp.war/
```

Après un bon bout de temps ;-), on parvient enfin à :

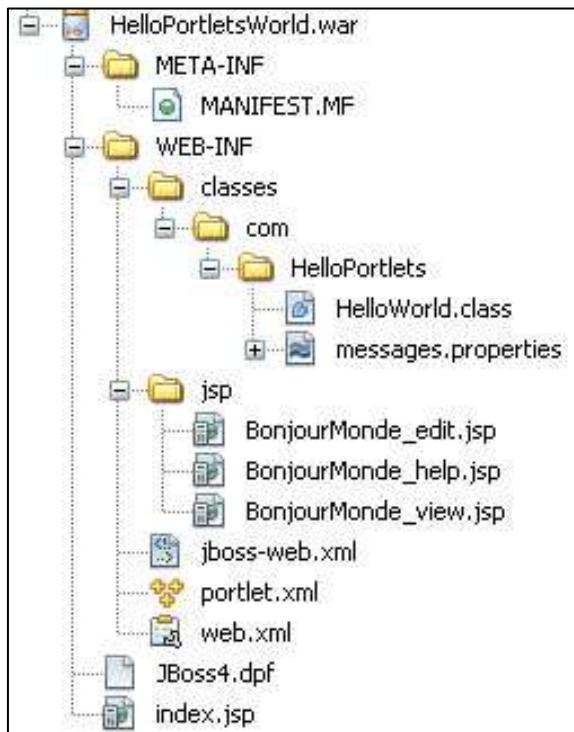
HelloPorletsWorld (run-deploy)

```
Starting JBoss Application Server
JBoss Application Server Started
Distributing C:\java-netbeans-application\HelloPortletsWorld\dist\HelloPortletsWorld.war
to [org.jboss.deployment.spi.LocalhostTarget@571e57]
Deploying C:\java-netbeans-application\HelloPortletsWorld\dist\HelloPortletsWorld.war
Waiting for server to start the module http://localhost:8080/HelloPortletsWorld
Application Deployed
Operation start completed
run-deploy:
BUILD SUCCESSFUL (total time: 5 minutes 33 seconds)
```

On peut vérifier les résultats de ce déploiement dans l'onglet Services sur le nœud de JBoss Application Server :



En fait, un fichier war a été créé dans le répertoire dist habituel :



Le déploiement effectué, il reste une dernière chose à faire.

7.5 L'instanciation de la portlet

Pour instancier la portlet dans une page de portail, il nous faut aller dans l'administration du serveur :

<http://localhost:8080/portal/auth/portal/admin>

ce qui donne :



Par défaut, les login et mot de passe sont admin-admin. On obtient :

The screenshot shows the JBoss Portal administration interface in Mozilla Firefox. The title bar reads "JBoss Portal 2.6.4-GA - Mozilla Firefox". The menu bar includes "Fichier", "Édition", "Affichage", "Historique", "Marque-pages", "Outils", and "?". The address bar shows the URL "http://localhost:8080/portal/auth/portal/admin". The page header "JBoss Portal 2.6.4-GA" and "JBossPortal" are visible. The top right corner shows "Logged in as: admin", "Dashboard", "Portal", and "Logout". The main navigation tabs are "CMS", "Members", "WSRP", and "Administration", with "Administration" being the active tab. Below the tabs are sub-tabs: "Portal Objects", "Portlet Instances", "Portlet Definitions", and "Dashboards", with "Portlet Definitions" being the active tab. The left sidebar has sections for "Manage portals" (Properties) and "Manage sub-portals". A form to "Create a portal named:" is present with a "Create portal" button. The main content area displays a table of existing portals:

Portal	Actions
admin	Security Properties Theme Make Default
default	Security Properties Theme Rename Default
template	Security Properties Theme Make Default

At the bottom right, there is a link "Switch to wizard mode". The footer says "Powered by JBoss Portal".

On choisit le portail par défaut pour instancier notre portlet puis, dans default, on choisit le page layout dans "Manage default page" :

The screenshot shows the JBoss Portal administration interface in Mozilla Firefox. The title bar reads "JBossPortal". The menu bar includes "Fichier", "Édition", "Affichage", "Historique", "Marque-pages", "Outils", and "?". The address bar shows the URL "http://localhost:8080/portal/auth/portal/admin". The page header "JBoss Portal 2.6.4-GA" and "JBossPortal" are visible. The top right corner shows "Logged in as: admin", "Dashboard", "Portal", and "Logout". The main navigation tabs are "CMS", "Members", "WSRP", and "Admin", with "Admin" being the active tab. Below the tabs are sub-tabs: "Portal Objects", "Portlet Instances", "Portlet Definitions", and "Dashboards", with "Portlet Definitions" being the active tab. The left sidebar has sections for "Portals" and "default portal". A form to "Create a page named:" is present with a "Create page" button. The main content area displays a table of pages under the "default portal":

Page	Actions
News	Page layout Security Properties Theme Rename Display Names Delete Make Default
Weather	Page layout Security Properties Theme Rename Display Names Delete Make Default
default	Page layout Security Properties Theme Rename Display Names Delete Make Default

At the bottom right, there is a link "Switch to wizard mode". The footer says "Powered by JBoss Portal".

Dans l'onglet "Portlet Definitions" :

JBossPortal

Logged in as: admin
Dashboard | Portal | Logout

CMS Members WSRP Admin

Portal Objects Portlet Instances Portlet Definitions Dashboards

View portlets provided by the portlet provider named: local View portlets

Portlet name	Description	Remote	Remotable	Actions
Administration Portlet	Administration Portlet	<input type="checkbox"/>	<input type="checkbox"/>	 Create instance
Content Management System Administration Portlet	Administration Portlet for CMS	<input type="checkbox"/>	<input type="checkbox"/>	 Create instance
Current Users Portlet	Current users portlet	<input type="checkbox"/>	<input checked="" type="checkbox"/>	 Create instance
Dashboard Configurator Portlet	Dashboard Configurator Portlet	<input type="checkbox"/>	<input type="checkbox"/>	 Create instance
HelloMonde	DireBonjour	<input type="checkbox"/>	<input type="checkbox"/>	 Create instance
Identity admin portlet	Identity admin portlet	<input type="checkbox"/>	<input type="checkbox"/>	 Preferences  Create instance
Identity user portlet	Identity user portlet	<input type="checkbox"/>	<input type="checkbox"/>	 Preferences  Create instance
JSP Portlet	Simple JSP portlet	<input type="checkbox"/>	<input type="checkbox"/>	 Create instance
News Portlet	Portlet aggregating news from different feeds	<input type="checkbox"/>	<input type="checkbox"/>	 Preferences  Create instance
Portal Pages Catalog Portlet	Portlet providing navigable list of portal pages	<input type="checkbox"/>	<input type="checkbox"/>	 Create instance
User Portlet	Portlet providing user login/logout and profile management	<input type="checkbox"/>	<input type="checkbox"/>	 Create instance
User Roles Portlet	Portlet for managing user roles	<input type="checkbox"/>	<input type="checkbox"/>	 Create instance
Weather Portlet	Portlet providing weather forecast	<input type="checkbox"/>	<input type="checkbox"/>	 Preferences  Create instance
WSRP Configuration	Configuration portlet for WSRP	<input type="checkbox"/>	<input type="checkbox"/>	 Create instance

JBossPortal

Logged in as: admin
Dashboard | Portal | Logout

CMS Members WSRP Admin

Portal Objects Portlet Instances Portlet Definitions Dashboards

Portlet Definitions HelloMonde instance creation

Create an instance named: HelloMondeInstance
Create instance

Powered by JBoss Portal

et l'on confirme avec Create instance :

JBossPortal

Logged in as: admin
Dashboard | Portal | Logout

CMS Members WSRP Admin

Portal Objects Portlet Instances Portlet Definitions Dashboards

Portlet Instances HelloMondeInstance portlet details

Portlet Instance Information

Portlet name: HelloMonde	Portlet description: DireBonjour	Portlet title: PortletBonjourMonde
Portlet keywords:	Portlet locales: en	

Portlet Instance Display Names

Add Instance Display Names		Current Instance Display Names		
Locale	Display Name	Delete	Rename	
Albanian	Add Name			

Powered by JBoss Portal

The screenshot shows the JBoss Portal Administration interface. The top navigation bar includes links for CMS, Members, WSRP, Administration (selected), Dashboard, Portal, and Logout. The main content area is titled "Portlet Instances" and "PortletTest portlet details". Under "Portlet Instance Information", it displays the Portlet name (HelloWorldPortlet), Portlet description (HelloWorldPortlet), Portlet title (HelloWorldPortlet), Portlet keywords (empty), and Portlet locales (en). Below this is a section for "Portlet Instance Display Names" with a "Current Instance Display Names" table. The table has columns for Locale, Display Name, Delete, and Rename. It contains one entry: Français, PortletEssai, with a delete icon and a rename icon. At the bottom of the page, it says "Powered by JBoss Portal".

On passe enfin sur "portlets objects" :

The screenshot shows the JBoss Portal Admin interface. The top navigation bar includes links for CMS, Members, WSRP, Admin (selected), Dashboard, Portal, and Logout. The main content area is titled "Portals > default portal > default page Layout". It is divided into two main sections: "Content Definition" and "Page Layout".

Content Definition: A form to define a name for the window of content (optional). The "Window Name" field contains "Pour dire bonjour". Below it, a "Content Type" dropdown is set to "portlet".

Page Layout: A configuration for the "center Region" which contains a "CMSWindow". There are "Add", "Up", "Down", and "Delete" buttons for this window. Below the center region is the "left Region", which contains "JSPPortletWindow", "IdentityUserPortletWindow", and "CurrentUsersPortletWindow". There are also "Add", "Up", "Down", and "Delete" buttons for these windows.

Portlet instance associated to this window: A list of portlets available to be added to the current window. The "Portlet name:HelloMonde" and "Portlet description:DiréBonjour" for the "PortletBonjourMonde" portlet are highlighted. Other listed portlets include "Administration portlet", "Catalog portlet", "CMSAdminPortletInstance", "Who's online por", "DashboardConfig", "HelloMondeInstance", and "IdentityAdminPortletInstance".

On sélectionne notre portlet :

Content Definition

Define a name for the window of content (optional):
Window Name: Pour dire bonjour

Select the type of content that will be added to the page:
Content Type: portlet

Select content that will be added to the page:

Portlet instance associated to this window:

- HelloMondeInstance
Portlet name:HelloMonde
Portlet description:DireBonjour
- Administration portlet
- Catalog portlet
- CMSAdminPortletInstance
- Who's online portlet
- DashboardConfigPortletInstance

Page Layout

center Region

Add CMSWindow

Up
Down
Delete

left Region

Add JSPPortletWindow
IdentityUserPortletWindow
CurrentUsersPortletWindow

Up
Down
Delete

On choisit pour elle la région "left" ou "center" :

Content Definition

Define a name for the window of content (optional):
Window Name: Pour dire bonjour

Select the type of content that will be added to the page:
Content Type: portlet

Select content that will be added to the page:

Portlet instance associated to this window:

- HelloMondeInstance
Portlet name:HelloMonde
Portlet description:DireBonjour
- Administration portlet
- Catalog portlet
- CMSAdminPortletInstance
- Who's online portlet
- DashboardConfigPortletInstance
- Administration portlet

Page Layout

center Region

Add CMSWindow
Pour dire bonjour

Up
Down
Delete

left Region

Add JSPPortletWindow
IdentityUserPortletWindow
CurrentUsersPortletWindow

Up
Down
Delete

On peut évidemment la faire monter avec le bouton Up. Une fois ces opérations terminées, on peut enfin afficher la page :

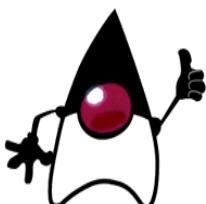
<http://localhost:8080/portal/auth/portal/default>

Si, si, regardez-bien - **notre portlet est en haut** ("PortletBonjourMonde") :

The screenshot shows the JBoss Portal 2.6.4-GA interface in Mozilla Firefox. The main content area features a portlet titled "PortletBonjourMonde" in "EDIT MODE". The portlet displays a photograph of a person standing on a rocky outcrop, looking through binoculars, with the text "Unbound Opportunity..." overlaid. To the left, there is a "Greetings!" portlet with three icons: "Demo.", "Download.", and "Accessorize.". Below these is a "User portlet" showing the user profile for "admin". The "User portlet" includes links for "Voir mon profil" and "Modifier mon profil", and displays the identifier "Identifiant: admin" and email "Courriel: admin@portal.com". At the bottom left, a "Current users" portlet shows "Il y a 1 utilisateur en ligne: [admin]". The browser toolbar at the top includes links for File, Edit, View, History, Bookmarks, Tools, and Help, along with a search bar and a Google button. The address bar shows the URL <http://localhost:8080/portal/auth/portal/default>. The status bar at the bottom indicates the system is running Windows and shows the time as 14:00.

Bien sûr, ce généreux sujet est à peine introduit : nous aurons certainement l'occasion de revisiter ce domaine passionnant des portails !

Mais sans doute est-il temps de nous tourner vers une technologie moderne dans le contexte du e-commerce : le goût des applets sans les pépins ;-) ...



XXXIX. Java Web Start



*Ne t'enorgueillis pas parce que tu apprends
Interroge l'ignorant comme le savant
Car on n'a jamais atteint les limites de l'art
Et il n'est pas d'artisan dont la maîtrise soit
parfaite.*

(Enseignement de Ptahhotep [Egypte, vers 1500 av. J.C.])

1. Une application Java téléchargée par HTTP

Historiquement, Java a d'abord été accueilli dans le cercle des développeurs parce qu'il permettait de doter les pages HTML d'applets apportant une certaine intelligence à celles-ci. Si, au départ, il n'était question que d'animations et de logique élémentaire appliquée aux entrées effectuées dans le GUI de l'applet, les possibilités de programmation se sont accrues pour les applets avec la technologie middleware des servlets, bientôt accompagnées des Java Server Pages. Le dialogue client-serveur basique selon http se transforma ainsi progressivement en une communication applet-servlet.

Cependant, on sait que les applets présentent bien des limites par rapport aux applications Java classiques. C'est ici qu'intervient Java Web Start (**JWS** en abrégé).



En effet, cette technologie permet le déploiement sur le client d'**applications**, au sens complet du terme, qui sont **téléchargées depuis un serveur Web et lancées sur ce client**, ceci à partir d'un browser classique. En quelque sorte, on dispose de la souplesse d'accessibilité des applets mais aussi du fait que l'on utilise une application, aux possibilités bien plus grandes. De plus, le fait que l'application provienne d'un **serveur assure que le client disposera toujours de la version la plus récente de celle-ci**, ce qui ne serait pas le cas avec une application installée classiquement sur le client. En effet, Java Web Start vérifie à chaque exécution si une version plus récente ne se trouve pas sur le serveur Web; si c'est le cas, cette version plus récente est téléchargée.

On peut encore pointer d'autres qualités à cette technologie :

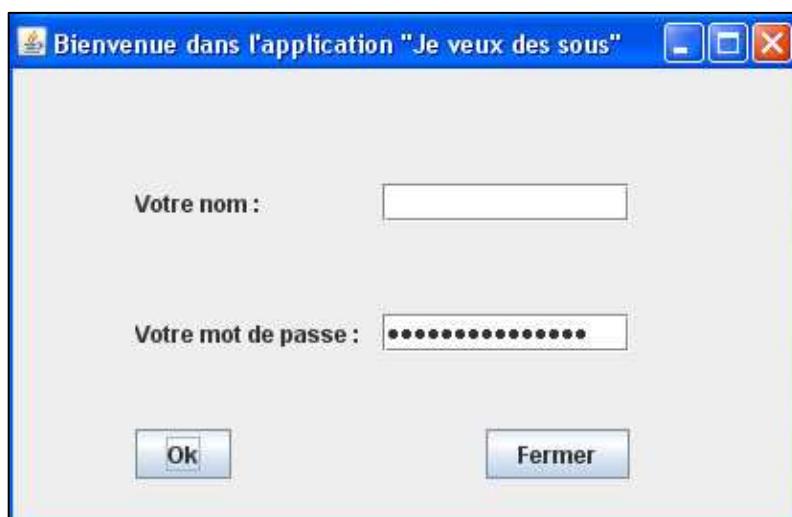
- ♦ une application JWS étant écrite en Java et étant disponible sur un simple serveur Web sous forme de bytecode (placé dans un jar pour être précis), elle peut être déployée sur toute une série de plates-formes (les divers Windows, Linux, Unix, etc) – on retrouve évidemment l'argument de portabilité Java;

- ◆ on retrouve également l'argument classique de la sécurité Java : une application JWS non authentifiée tournera dans une classique sandbox;
- ◆ une application JWS peut sans problème réclamer une plate-forme Java particulière : en effet, Java Web Start télécharge et installe automatiquement la révision nécessaire de la plate-forme si la version nécessaire ne se trouve pas sur le client;
- ◆ une application JWS est mise en cache à son démarrage; ceci explique, entre autres, que les applications JWS ont des exigences réduites en matière de bande passante, puisqu'elles ne communiquent pas systématiquement avec le serveur Web.

2. Le lancement d'une application JWS

La version actuelle de JWS (1.2 à 1.6) appartient d'office au J2SE (le jar se trouve dans C:\Program Files\Java\jdk1.6.0_13\jre\lib). Pour les versions antérieures, on installe Java Web Start sans difficultés : il suffit de le télécharger depuis le site de Sun et de l'installer (sur une machine Windows, il le sera dans le répertoire Java Web Start de Program Files). La seule précaution supplémentaire éventuelle est de configurer le browser utilisé pour qu'il associe les fichiers d'extensions **jnlp** (qui, comme nous allons le voir, décrivent une application JWS) à Java Web Start (Firefox posera les éventuelles questions nécessaires à la première utilisation et mettra sa liste d'extensions à jour – en principe, tout est configuré par défaut pour que on trouve dans Applications section : "jnlp", "application/x-java-jnlp-file" associé à "javaws.exe").

Supposons avoir créé et transformé en jar une application qui affiche un (très) modeste GUI (qui fait apparaître ensuite une boîte de dialogue de confirmation) :



dont le code est évidemment de peu d'intérêt :

JWSFenGui.java
package ApplicationSousCore;
import ApplicationSousCore.Dialogues.NomPwdConfirmation;
public class JWSFenGui extends javax.swing.JFrame

```
{  
    public FenetreApplication() { initComponents(); }  
    private void initComponents()  
    {  
        jLabel1 = new javax.swing.JLabel(); jLabel2 = new javax.swing.JLabel();  
        jTextField1 = new javax.swing.JTextField();  
        jButton1 = new javax.swing.JButton(); jButton2 = new javax.swing.JButton();  
        jPasswordField1 = new javax.swing.JPasswordField();  
        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);  
        setTitle("Bienvenue dans l'application \"Je veux des sous\"");  
        jLabel1.setText("Votre nom :"); jLabel2.setText("Votre mot de passe :");  
        jButton1.setText("Ok");  
        jButton1.addActionListener(new java.awt.event.ActionListener() {  
            public void actionPerformed(java.awt.event.ActionEvent evt) {  
                jButton1ActionPerformed(evt);  
            }  
        });  
        jButton2.setText("Fermer");  
        jButton2.addActionListener(new java.awt.event.ActionListener() {  
            public void actionPerformed(java.awt.event.ActionEvent evt) {  
                jButton2ActionPerformed(evt);  
            }  
        });  
        ...  
        pack();  
    } // </editor-fold>  
  
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
        NomPwdConfirmation dial = new NomPwdConfirmation(this, true);  
        dial.setTitle("Confirmation");  
        dial.setVisible(true);  
    }  
  
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
        System.exit(0);  
    }  
  
    public static void main(String args[]) {  
        java.awt.EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                new FenetreApplication().setVisible(true);  
            }  
        });  
    }  
  
    private javax.swing.JButton jButton1;  
    ...  
}
```

En pratique, une application JWS est lancée classiquement depuis une page HTML. Celle-ci comporte un lien désignant **un fichier de déploiement** (en fait, un fichier **JNLP** qui est un fichier XML – nous allons y revenir) qui se trouve sur le serveur Web utilisé (dans le cas de Tomcat, les deux fichiers se trouvent dans le répertoire ROOT) :

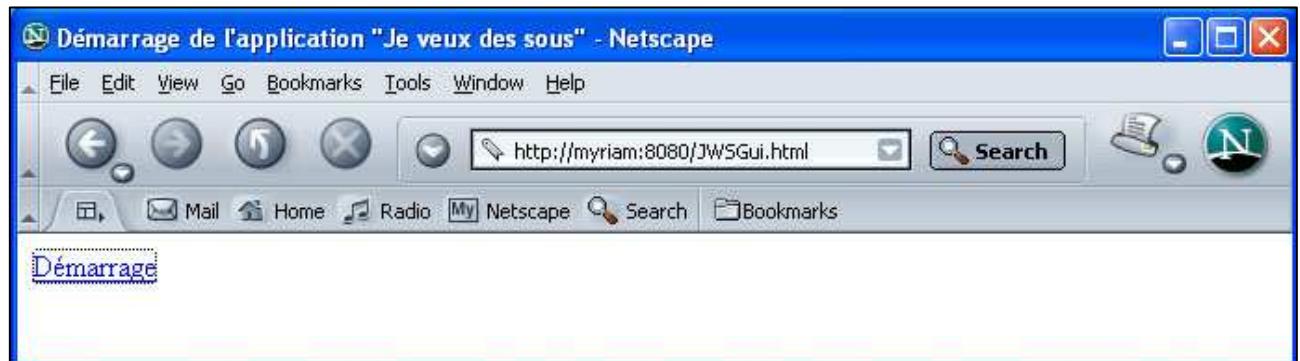
JWSGui.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Démarrage de l'application "Je veux des sous"</TITLE>
</HEAD>
<BODY>
<A HREF="JWSGui.jnlp">Démarrage</a>
</BODY>
</HTML>
```

La sollicitation de ce lien a pour effet de ***lancer sur la machine client l'application Java Web Start*** qui va

- ◆ télécharger l'application désignée dans ce fichier JNLP et qui se trouve sur le serveur (par exemple sur la machine "myriam") sous forme d'un fichier jar trouvent (dans le répertoire ROOT également);
- ◆ mettre cette application en cache sur le client;
- ◆ lancer l'exécution de cette application sur ce client.

Donc, en images, pour une modeste application Java nommée "Je veux des sous" :



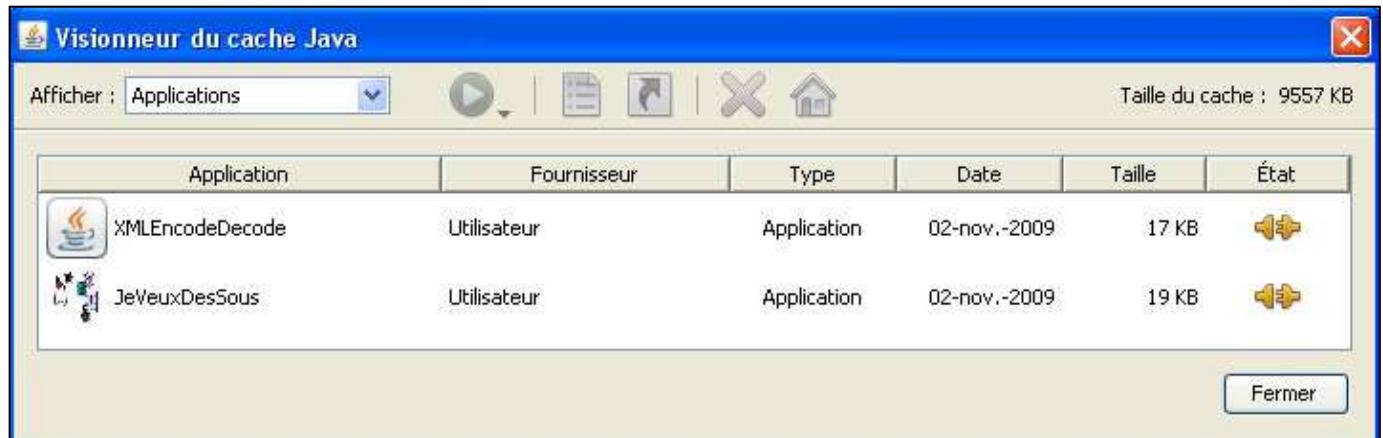
Le choix du browser n'a évidemment aucune importance : l'exécution est semblable sous Internet Explorer.

3. Le gestionnaire d'applications

Bien que l'application soit normalement lancée depuis un lien dans une page HTML, il est aussi possible de *lancer le mécanisme du Java Web Start manuellement* depuis la ligne de commande :

```
| C:\Program Files\Java\jdk1.6.0_13\jre\bin>javaws -viewer
```

ce qui a pour effet de faire apparaître le contenu du cache de Java Web Start :



et de rendre possible le lancement du processus :

The screenshot shows the 'Visionneur du cache Java' window with the same interface as before. However, the second row ('JeVeuxDesSous') now has a dashed selection border around its entire row. To the right of this window, a larger window titled 'Panneau de configuration Java' is visible, showing tabs for Général, Mise à jour, Java, Sécurité, and Avancé. The 'Général' tab is selected. In the bottom right corner of the main window, there is a 'Paramètres réseau...' button. On the far right of the screen, there is a vertical sidebar with sections for 'Fichiers Internet temporaires' and 'Paramètres... / Afficher... / OK / Annuler / Appliquer' buttons.

avec en filigrane le panneau de configuration Java :

Mais que contient donc le fameux fichier JNLP ?

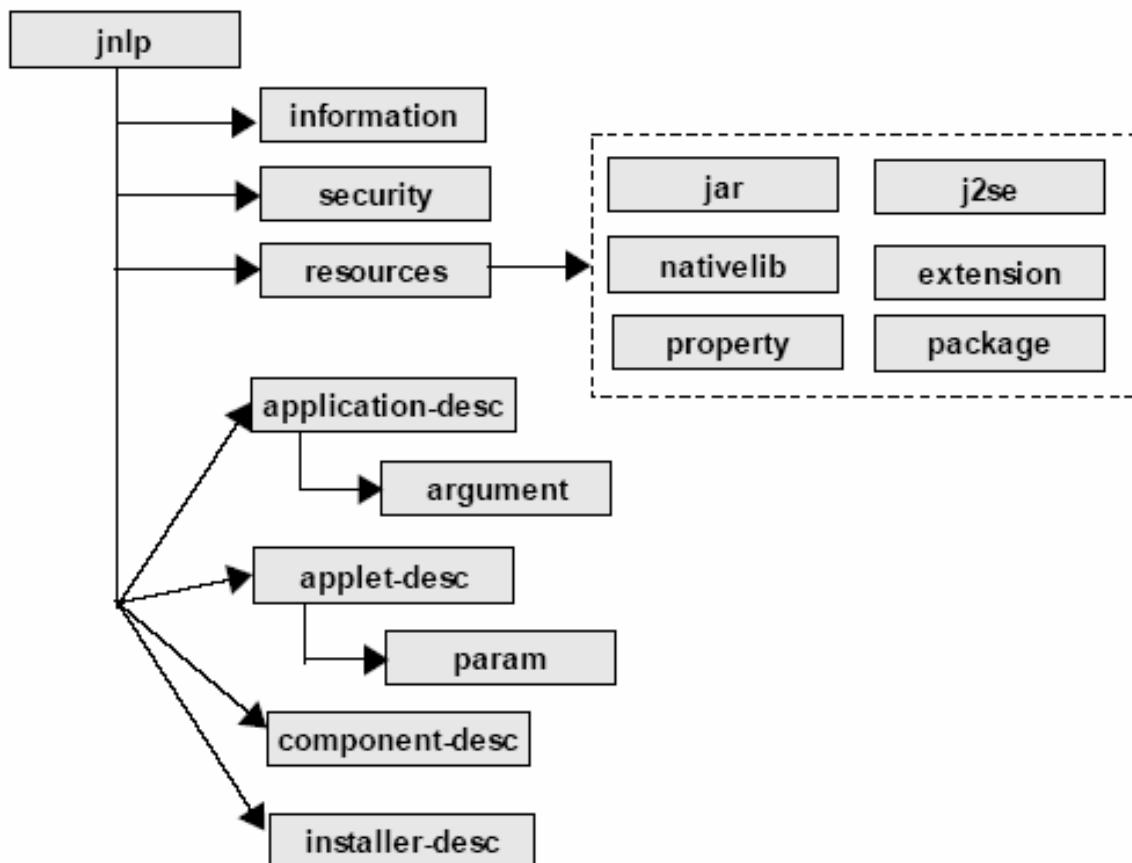
4. Le fichier de déploiement d'une application JWS

4.1 Les éléments de base

La cheville fondamentale de Java Web Start est le **Java Network Launching Protocol (JNLP)** en abrégé). La trame de ce protocole est placée dans un fichier d'extension jnlp. Globalement, ce fichier qui est en fait un fichier XML, permet de

- ◆ spécifier les fichiers jar contenant les fichiers class de l'application et qui seront téléchargés depuis le serveur Web;
- ◆ donner des informations sur les packages requis;
- ◆ décrire les ressources à utiliser;
- ◆ préciser le nom de la classe qui contient la méthode main();
- ◆ préciser la politique de sécurité;
- ◆ etc.

Ce fichier (et donc le protocole qu'il représente) possède la structure suivante :



Le tag de base `<jnlp>` est défini dans la DTD précisée dans la spécification officielle de Java Web Start. Il se décompose en 4 tags imbriqués et est paré de 4 attributs optionnels :

```

<!ELEMENT jnlp (information+, security?, resources*, (application-desc |
applet-desc | component-desc | installer-desc))>
<!ATTLIST jnlp spec CDATA #IMPLIED>
<!ATTLIST jnlp version CDATA #IMPLIED>
<!ATTLIST jnlp codebase CDATA #IMPLIED>
<!ATTLIST jnlp href CDATA #IMPLIED>

```

Fondamentalement, seuls les tags <information> et <application-desc> (si nous nous intéressons spécialement aux applications téléchargées) sont obligatoires – nous allons les décrire sommairement ci-dessous. Le tag <resources> est le plus souvent présent parce qu'il permet notamment de désigner le jar associé à l'application. Quant au tag <security>, il n'intervient évidemment que si l'on se préoccupe de permissions à donner à cette application.

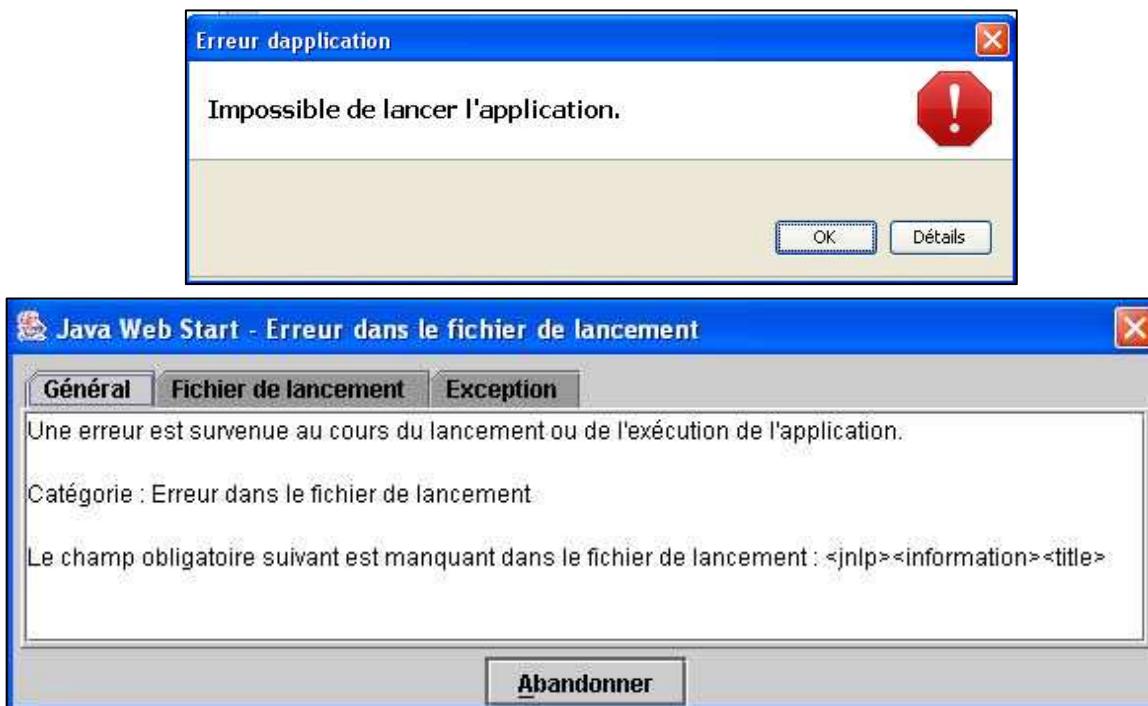
1) Les attributs du tag <jnlp>, qui sont le plus souvent présents bien qu'optionnels, ont la signification suivante :

- ◆ spec : il s'agit simplement de la version de la spécification utilisée par le fichier JNLP; par défaut, cette version est "1.0+";
- ◆ version : bien sûr, il s'agit de la version du fichier, et donc aussi de celle de l'application qui sera lancée;
- ◆ codebase : comme pour une applet classique, il s'agit de préciser ici la racine des références en forme d'URL qui seront utilisées dans le fichier par des clauses "href";
- ◆ href : cette URL du fichier JNLP lui-même est utilisée par le manager d'applications; elle lui permet notamment de rechercher une éventuelle nouvelle version.

2) Le tag <information> est défini par la DTD selon :

```
<!ELEMENT information (title?, vendor?, homepage?, description*, icon*, offline-allowed?)>
```

A remarquer que les éléments title et vendor sont en fait exigés par JWS. Ainsi, l'omission de title donne :



3) Le tag <ressources> défini par :

```
<!ELEMENT resources (j2se | jar | nativelib | extension | property | package)*>
```

permet évidemment de décrire les éléments qui constituent l'application. Nous serons essentiellement intéressés par les tags :

```
<!ELEMENT j2se (resources*)>
<!ATTLIST j2se version CDATA #REQUIRED>
    qui définit la version de J2SE nécessaire pour exécuter l'application

<!ELEMENT jar EMPTY>
<!ATTLIST jar href CDATA #REQUIRED>
    qui désigne bien sûr le fichier jar contenant la ressource en question, avec son URL.
```

4) Le tag de type "desc" existe en plusieurs versions, selon que l'on développe une application ou une applet (par exemple). Ce qui nous intéresse ici est :

```
<!ELEMENT application-desc (argument*)>
```

qui explique comment l'application doit être lancée, c'est-à-dire essentiellement où trouver la méthode main() :

```
<!ATTLIST application-desc main-class CDATA #IMPLIED>
```

et, éventuellement, la valeur des arguments qui sont passés en ligne de commande :

```
<!ELEMENT argument (#PCDATA)>
```

4.2 Un fichier de déploiement exemple

Pour notre application JWSFenGui, un fichier JNLP raisonnable serait :

JWSGui.jnlp
<?xml version="1.0" encoding="utf-8"?> <!-- JNLP File for SwingSet2 Demo Application --> <jnlp spec="1.0+" codebase="http://myriam:8080" href="JWSGui.jnlp"> <!--jnlp spec="1.0+" codebase="http://u2.wildness.loc/~vilvens" href=" JWSGui.jnlp "--> <information> <title>Application Je Veux Des Sous</title> <vendor>Vilvens</vendor> <!--homepage href="Aide.html"--> <description>Application GUI</description> <description kind="short">Utilisation d'un GUI classique.</description> <!--icon href="dollar.jpg"--> <offline-allowed/> </information>

```
<security>
</security>
<resources>
    <j2se version="1.4"/>
    <jar href="JWSGui.jar" main="true"/>
</resources>
<application-desc main-class="JWSFenGui"/>
</jnlp>
```

5. Analyse du trafic réseau

Concrètement, si on analyse le trafic réseau au moyen d'un sniffer, on obtient les résultats suivants :

1) téléchargement de la page HTML :

The screenshot shows two network connections in Wireshark. The first connection (highlighted in blue) is between 192.168.2.2:3084 and 192.168.2.1:8080, with 13 TCP packets. The second connection (highlighted in red) is between 192.168.2.2:3089 and 192.168.2.1:8080, with 20 TCP packets. The details pane on the right shows the first connection's packets. The highlighted packet is the GET request to /JWSgui.html, which includes the Host header 'myriam:8080'. The response (HTTP/1.1 200 OK) includes the ETag 'W/"221-1108733952000"'.

requête :

➔ **GET /JWSgui.html HTTP/1.1**

Host: myriam:8080

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.1) Gecko/20020823 Netscape/7.0

Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,text/css,*/*;q=0.1

Accept-Language: en-us, en;q=0.50

Accept-Encoding: gzip, deflate, compress;q=0.9

Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66

Keep-Alive: 300

Connection: keep-alive

réponse :

➔ **HTTP/1.1 200 OK**

ETag: W/"221-1108733952000"

Last-Modified: Fri, 18 Feb 2005 13:39:12 GMT

Content-Type: text/html;charset=ISO-8859-1

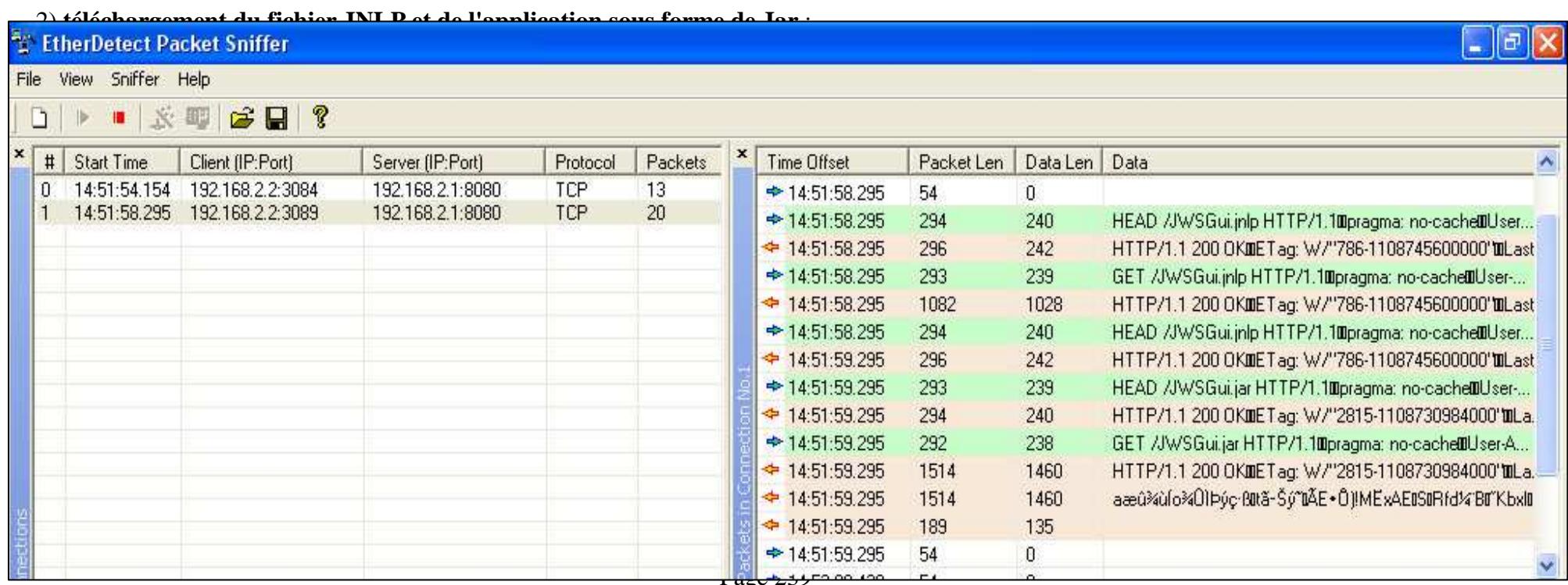
Content-Length: 221

Date: Thu, 03 Mar 2005 15:43:02 GMT

Server: Apache-Coyote/1.1

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

```
<HTML>
<HEAD>
  <TITLE>Démarrage de l'application "Je veux des sous"</TITLE>
</HEAD>
<BODY>
<a href="JWSGui.jnlp">Démarrage</a>
</BODY>
</HTML>
```



2.1) ➔ HEAD /JWSGui.jnlp HTTP/1.1

pragma: no-cache

User-Agent: JNLP/1.0.1 javaws/1.4.2_04 (b05) J2SE/1.4.2_04

UA-Java-Version: 1.4.2_04

Host: myriam:8080

Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2

Connection: keep-alive

◀ HTTP/1.1 200 OK

ETag: W/"786-1108745600000"

Last-Modified: Fri, 18 Feb 2005 16:53:20 GMT

Content-Type: application/x-java-jnlp-file; charset=ISO-8859-1

Content-Length: 786

Date: Fri, 04 Mar 2005 14:02:07 GMT

Server: Apache-Coyote/1.1

2.2) ➔ GET /JWSGui.jnlp HTTP/1.1

pragma: no-cache

User-Agent: JNLP/1.0.1 javaws/1.4.2_04 (b05) J2SE/1.4.2_04

UA-Java-Version: 1.4.2_04

Host: myriam:8080

Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2

Connection: keep-alive

◀ HTTP/1.1 200 OK

ETag: W/"786-1108745600000"

Last-Modified: Fri, 18 Feb 2005 16:53:20 GMT

Content-Type: application/x-java-jnlp-file; charset=ISO-8859-1

Content-Length: 786

Date: Fri, 04 Mar 2005 14:02:07 GMT

Server: Apache-Coyote/1.1

```
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for SwingSet2 Demo Application -->
<jnlp
    spec="1.0+"
    codebase="http://myriam:8080"
    href="JWSGui.jnlp">
<!--jnlp
    spec="1.0+"
    codebase="http://u2.wildness.loc/~vilvens"
    href="JWSGui.jnlp"-->
<information>
    <title>Application Je Veux Des Sous</title>
    <vendor>Vilvens</vendor>
    <!--homepage href="Aide.html"-->
    <description>Application GUI</description>
    <description kind="short">Utilisation d'un GUI classique.</description>
    <!--icon href="dollar.jpg"-->
    <offline-allowed/>
</information>
<security>
</security>
<resources>
    <j2se version="1.4"/>
    <jar href="JWSGui.jar" main="true"/>
</resources>
<application-desc main-class="JWSFenGui"/>
</jnlp>
```

2.3) ➔ HEAD /JWSGui.jnlp HTTP/1.1

pragma: no-cache

User-Agent: JNLP/1.0.1 javaws/1.4.2_04 (b05) J2SE/1.4.2_04

UA-Java-Version: 1.4.2_04

Host: myriam:8080

Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2

Connection: keep-alive

◀ HTTP/1.1 200 OK

ETag: W/"786-1108745600000"

Last-Modified: Fri, 18 Feb 2005 16:53:20 GMT

Content-Type: application/x-java-jnlp-file; charset=ISO-8859-1

Content-Length: 786

Date: Fri, 04 Mar 2005 14:02:07 GMT

Server: Apache-Coyote/1.1

2.4) ➔ HEAD /JWSGui.jar HTTP/1.1

pragma: no-cache

User-Agent: JNLP/1.0.1 javaws/1.4.2_04 (b05) J2SE/1.4.2_04

UA-Java-Version: 1.4.2_04

Host: myriam:8080

Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2

Connection: keep-alive

◀ HTTP/1.1 200 OK

ETag: W/"2815-1108730984000"

Last-Modified: Fri, 18 Feb 2005 12:49:44 GMT

Content-Type: application/java-archive; charset=ISO-8859-1

Content-Length: 2815

Date: Fri, 04 Mar 2005 14:02:07 GMT

Server: Apache-Coyote/1.1

2.5) ➔ GET /JWSGui.jar HTTP/1.1

pragma: no-cache

User-Agent: JNLP/1.0.1 javaws/1.4.2_04 (b05) J2SE/1.4.2_04

UA-Java-Version: 1.4.2_04

Host: myriam:8080

Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2

Connection: keep-alive

◀ HTTP/1.1 200 OK

ETag: W/"2815-1108730984000"

Last-Modified: Fri, 18 Feb 2005 12:49:44 GMT

Content-Type: application/java-archive; charset=ISO-8859-1

Content-Length: 2815

Date: Fri, 04 Mar 2005 14:02:07 GMT

Server: Apache-Coyote/1.1

<contenu du fichier jar : cryptique avec

META-INF/MANIFEST

...

JWSFenGui\$1.class

...

JWSFenGui\$2.class

...

JWSFenGui.class

>

A remarquer les 3 paquets nécessaires pour transférer le jar.

On peut remarquer que les requêtes HTTP GET sont toujours précédées d'une requête **HEAD**. Celle-ci, qui demande seulement l'en-tête (sans les données) de l'URI visé, permet à Java Web Start de vérifier si l'un ou l'autre fichier demandé a été modifié, ce qui se constate en comparant le header Last-Modified de la réponse avec le précédent.

3) **exécution ultérieures** : vérification de changements de version des fichiers JNLP et JAR :

The screenshot shows two panes of a network traffic analysis tool. The left pane, titled 'Connections', lists five entries. The right pane, titled '13 Packets in Connection No. 3', displays detailed information for each packet, including timestamp, offset, length, data, and protocol type. The data column shows the raw hex and ASCII representation of the captured packets.

#	Start Time	Client (IP:Port)	Server (IP:Port)	Protocol	Packets
0	15:29:27.171	192.168.2.2:3020	192.168.2.1:8080	TCP	10
1	15:29:49.499	192.168.2.1:138	192.168.2.255:138	UDP	4
2	15:29:49.514	192.168.2.2:3021	192.168.2.1:8080	TCP	10
3	15:29:53.561	192.168.2.2:3025	192.168.2.1:8080	TCP	13
4	15:32:18.858	192.168.2.1:137	192.168.2.255:137	UDP:ne...	6

13 Packets in Connection No. 3

Time Offset	Packet Len	Data Len	Data
15:29:53.561	62	0	
15:29:53.561	62	0	
15:29:53.561	54	0	
15:29:53.561	294	240	HEAD /JWSgui.jnlp HTTP/1.1 pragma: no-cache User-Agent: JNLP/1.0.1 javaws/1.4.2_04 (b05) J2SE/1.4.2_04 UA-Java-Version: 1.4.2_04 Host: myriam:8080 Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2 Connection: keep-alive
15:29:53.561	296	242	HTTP/1.1 200 OK ETag: W/"786-1108745600000"
15:29:53.561	293	239	HEAD /JWSgui.jar HTTP/1.1 pragma: no-cache User-Agent: JNLP/1.0.1 javaws/1.4.2_04 (b05) J2SE/1.4.2_04 UA-Java-Version: 1.4.2_04 Host: myriam:8080 Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2 Connection: keep-alive
15:29:54.561	60	0	
15:29:54.561	294	240	HTTP/1.1 200 OK ETag: W/"2815-1108730984000"
15:29:54.561	54	0	

➔ **HEAD /JWSgui.jnlp HTTP/1.1**

pragma: no-cache

User-Agent: JNLP/1.0.1 javaws/1.4.2_04 (b05) J2SE/1.4.2_04

UA-Java-Version: 1.4.2_04

Host: myriam:8080

Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2

Connection: keep-alive

◀ **HTTP/1.1 200 OK**

ETag: W/"786-1108745600000"

Last-Modified: Fri, 18 Feb 2005 16:53:20 GMT

Content-Type: application/x-java-jnlp-file; charset=ISO-8859-1

Content-Length: 786

Date: Thu, 03 Mar 2005 14:40:02 GMT

Server: Apache-Coyote/1.1

➔ **HEAD /JWSGui.jar HTTP/1.1**

pragma: no-cache

User-Agent: JNLP/1.0.1 javaws/1.4.2_04 (b05) J2SE/1.4.2_04

UA-Java-Version: 1.4.2_04

Host: myriam:8080

Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2

Connection: keep-alive

◀ **HTTP/1.1 200 OK**

ETag: W/"2815-1108730984000"

Last-Modified: Fri, 18 Feb 2005 12:49:44 GMT

Content-Type: application/java-archive; charset=ISO-8859-1

Content-Length: 2815

Date: Thu, 03 Mar 2005 14:40:02 GMT

Server: Apache-Coyote/1.1

Java Web Start gère en fait son propre cache dans C:\Program Files\Java Web Start\javaws\cache\http\. Ainsi pour notre application, on trouve dans C:\Program Files\Java Web Start\javaws\cache\http\myriam\P8080 les fichiers :

ALJWSGui.jnlp

AMJWSGui.jnlp

ATJWSGui.jnlp

RCJWSGui.jar

RMJWSGui.jar

RTJWSGui.jar

6. Les questions de sécurité

La DTD des fichiers JNLP prévoit un tag security qui définit le type de permissions dont l'application JWS va jouir :

```
<!ELEMENT security (all-permissions?, j2ee-application-clientpermissions?)>
```

Comme on peut le voir, deux possibilités se présentent :

```
<!ELEMENT all-permissions EMPTY>
```

indique que l'application a besoin d'un accès complet à la machine locale et au réseau

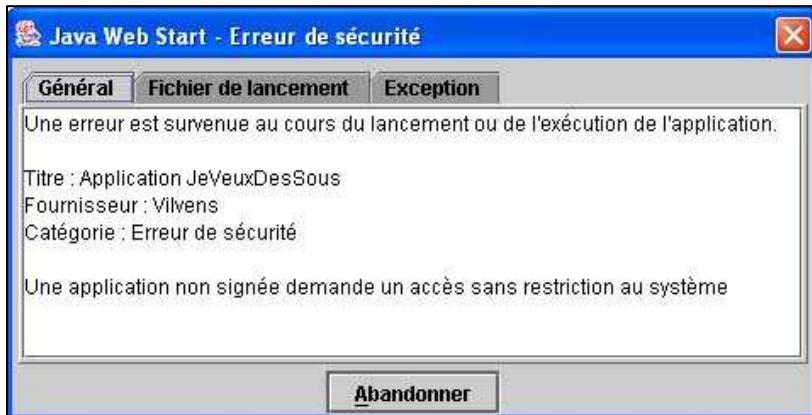
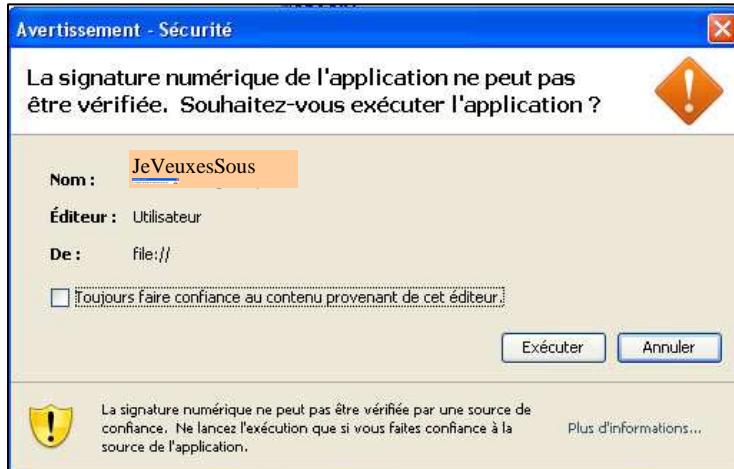
```
<!ELEMENT j2ee-application-client-permissions EMPTY>
```

indique au contraire que l'application ne nécessite que les permissions d'un client J2EE, autrement dit travaille dans une sand-box

Supposons vouloir faire travailler notre application avec un accès complet, soit avec la clause :

```
<security>
    <all-permissions/>
</security>
```

dans le fichier JNLP. Une tentative d'exécution donne quelque chose comme :



Voilà qui est clair. Nous allons donc utiliser l'outil jarsigner et un keystore contenant une clé privée permettant de signer notre fichier jar. Supposons que le keystore soit celui-ci :

```
C:\java-sun-application\JWS>keytool -list -keystore .keystore  
Tapez le mot de passe du Keystore : beaugosse
```

Type Keystore : jks
Fournisseur Keystore : SUN

Votre Keystore contient 1 entrée(s)

claudie, 15-avr.-2005, keyEntry,
Empreinte du certificat (MD5) : 01:4A:01:4E:98:C4:D8:BC:6C:53:4C:6B:A8:52:00:8C

Nous pouvons créer un fichier jar signé utilisant la keyEntry en commandant :

```
C:\java-sun-application\JWS>jarsigner -keystore .keystore -signedjar JWSGuiSigned.jar  
JWSGui.jar Claude  
Enter Passphrase for keystore: beaugosse  
Enter key password for Claude: genius
```

Bien sûr, nous modifierons le fichier jnlp pour que la ressource à utiliser soit le nouveau jar :

```
<resources>  
    <j2se version="1.4"/>  
    <jar href="JWSGuiSigned.jar" main="true"/>  
</resources>
```

Une fois l'application JWS lancée par l'intermédiaire de la page HTML, nous obtenons :



Les choix OK et Démarrer permettent de lancer l'application ☺.

7. Les APIs JNLP

7.1 Des services sécurisés

A priori, une application lancée au moyen de Java Web Start fonctionne au sein d'une "sandbox", ce qui restreint évidemment ses possibilités par rapport aux fichiers et peut se révéler ennuyeux pour toutes les questions d'accès aux données et de persistance de celles-ci (problème analogue à celui des cookies pour une applet classique). Pour cette raison, Java Web Start fournit un ensemble d'APIs, décrits par des interfaces définis dans le package javax.jnlp, qui permettent de réaliser de telles opérations de manière sécurisée, c'est-à-dire sous le contrôle de l'utilisateur.

En fait, c'est Java Web Start qui gère ces opérations délicates, et pas l'application elle-même. Celle-ci manipule en fait les interfaces et demande à Java Web Start un objet réalisant tel ou tel service, autrement dit implémentant l'interface correspondant, ceci en utilisant les méthodes statiques d'une classe nommée **ServiceManager** (toujours du package javax.jnlp). Toutes les classes se trouvent dans le package javax.jnlp, qui doit évidemment se trouver dans le classpath.

La classe **ServiceManager**, qui est final, fournit essentiellement une méthode

```
public static Object lookup (String name) throws UnavailableServiceException
```

qui recherche sur le client une classe qui implémente le service dont le nom est passé en paramètre (pour être tout à fait exact, c'est plutôt à un objet implémentant l'interface **ServiceManagerStub** sur le client que la requête est transmise). On peut d'ailleurs obtenir la liste des services disponibles sur le client par :

```
public static String[] getServiceNames()
```

7.2 Le BasicService

Ce service justifie son nom par le fait qu'il fournit des informations de base qui peuvent être utiles pour gérer le fonctionnement de l'application (ou de l'applet) téléchargée par Java Web Start. Principalement, on peut utiliser :

```
public URL getCodeBase()
```

qui fournit l'URL du jar utilisé

```
public boolean showDocument(java.net.URL url)
```

qui passe la main à la page d'url spécifiée (elle sera affichée par le browser actif ou, s'il n'y en a pas encore, le browser de la plateforme sera lancé)

```
public boolean isWebBrowserSupported()
```

qui permet de savoir si le client possède un browser.

Nous pouvons donc reprendre notre application GUI et lui demander d'afficher de tels renseignements :

JWSFenGui.java (2)
import javax.jnlp.*; import java.net.*;

```

public class JWSFenGui extends java.awt.Frame
{
    public JWSFenGui() { initComponents(); }
    private void initComponents() { ... }

    private void BOkActionPerformed(java.awt.event.ActionEvent evt)
    {
        System.out.println("A l'abordage !");
        String[] tabServices = ServiceManager.getServiceNames();
        System.out.println("Liste des services JNLP disponibles");
        for (int i=0; i<tabServices.length; i++)
        {
            System.out.println(i + ". " + tabServices[i]);
        }

        BasicService bs=null;

        try
        {
            bs = (BasicService)ServiceManager.lookup("javax.jnlp.BasicService");
        }
        catch (UnavailableServiceException e)
        {
            System.out.println("Service Basic non disponible :-( ");
        }
        URL url = bs.getCodeBase();
        System.out.println("URL d'origine = " + url);
        ZTUrl.setText(url.toString());
        if (bs.isWebBrowserSupported())
            System.out.println("Un browser Web est disponible");
        else System.out.println("Pas de browser Web disponible");
    }

    public static void main(String args[])
    {
        new JWSFenGui().show();
    }
    private java.awt.Label ZTUrl;
    ...
}

```

L'exécution de l'application Java Web Start donne :



tandis que la console Java Web Start affiche les résultats :

```
Java Web Start 1.4.2_04 Console, démarrée Tue Mar 22 18:18:06 CET 2005
Environnement d'exécution Java 2 : version 1.4.2_04 par Sun Microsystems Inc.
A l'abordage !
Liste des services JNLP disponibles
0. javax.jnlp.BasicService
1. javax.jnlp.FileOpenService
2. javax.jnlp.FileSaveService
3. javax.jnlp.DownloadService
4. javax.jnlp.ClipboardService
5. javax.jnlp.PersistenceService
6. javax.jnlp.PrintService
URL d'origine = http://myriam:8080/
Un browser Web est disponible
```

7.3 Les services de fichiers

Ces services permettent de lire et écrire dans des fichiers locaux (une vraie hérésie pour une sand-box !). Supposons ainsi vouloir sauvegarder le nom de client dans un fichier texte :

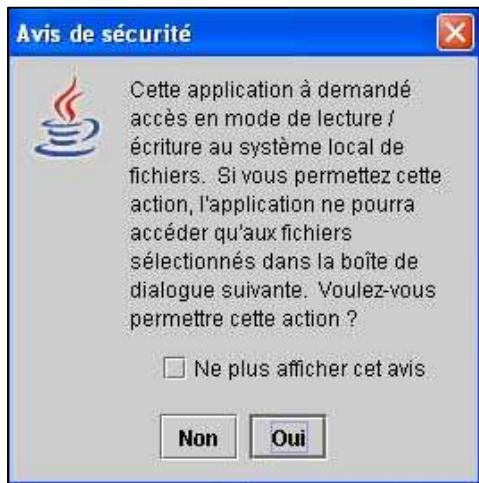


Il nous suffit d'ajouter dans l'application précédente le traitement pour le bouton "Sauver dans un fichier".

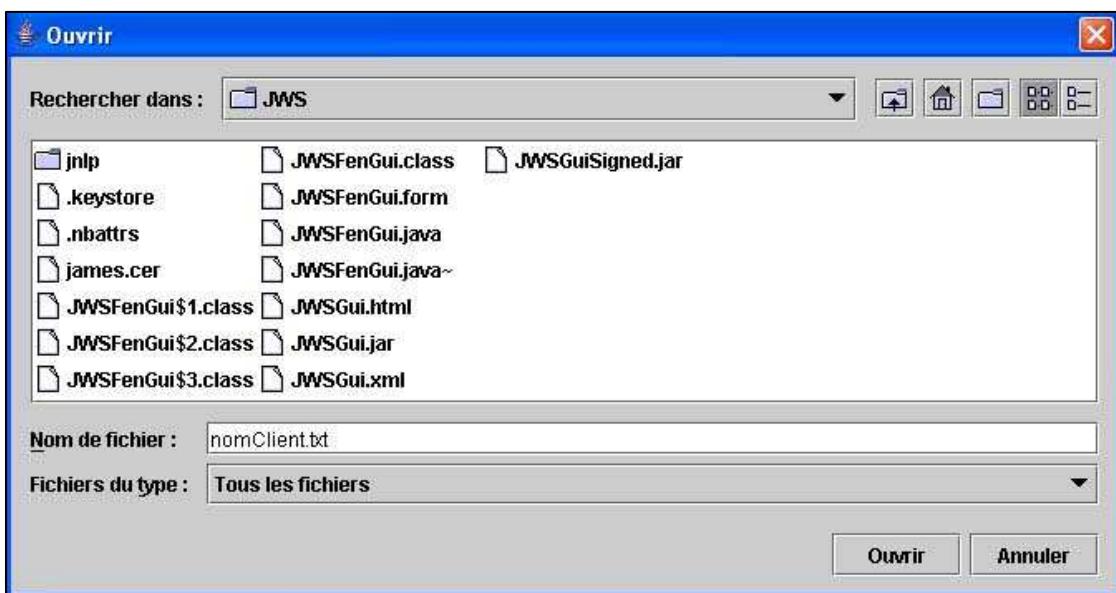
- ♦ Nous demandons la recherche du service **FileOpenService**;
- ♦ Si on l'a obtenu, nous utilisons sa méthode :

```
public FileContents openFileDialog(String chemin, String[] extensions)
throws java.io.IOException
```

qui provoque l'ouverture d'une boîte de dialogue permettant de choisir le fichier (éventuellement nouveau) dans lequel on écrira. Les deux paramètres, à la signification claire, peuvent en fait être ignorés par le client JNLP – nous n'en tiendrons donc pas compte (leur valeur sera prise à null). A l'exécution après l'avertissement :



donnera donc une boîte de dialogue du type suivant :



- ◆ Cette méthode fournit, à la fermeture de la boîte de dialogue, un objet instance d'une classe implémentant l'interface **FileContents** : il est sensé encapsuler le nom du fichier et son contenu. Dans le cas d'un nouveau fichier, la longueur, telle qu'elle a été définie par le créateur du fichier, est fournie par l'appel de la méthode de FileContents :

```
public long getLength()
    throws java.io.IOException
```

tandis que

```
public long getMaxLength()
    throws java.io.IOException
```

donne évidemment la longueur maximale.

Et cela donne pour la longueur actuelle 0 ! Il faut donc, au préalable, fixer cette longueur limite à une valeur plus acceptable au moyen de la méthode

```
public long setMaxLength(long maxlen)
    throws java.io.IOException
```

Pour écrire effectivement dans le fichier, il nous cependant un véritable flux. Nous pouvons nous le procurer avec la méthode de FileContents :

```
public JNLPRandomAccessFile getRandomAccessFile (String mode)
    throws java.io.IOException
```

le paramètre n'étant que le mode d'ouverture ("r" ou "rw").

- ♦ Le flux obtenu implémente donc l'interface **JNLPRandomAccessFile** qui dérive des célèbres DataInput et DataOutput. Ceci implique donc qu'il dispose de toutes les méthodes readXXX() et writeXXX().

Nous pouvons donc compléter notre application avec ceci :

JWSFenGui.java (3)

```
import javax.jnlp.*;
import java.net.*;
import java.io.*;

public class JWSFenGui extends java.awt.Frame
{
    ...
    private void BSauverActionPerformed (java.awt.event.ActionEvent evt)
    {
        FileOpenService fos = null;
        FileContents fc = null;
        JNLPRandomAccessFile jraf = null;
        try
        {
            fos = (FileOpenService)ServiceManager.lookup("javax.jnlp.FileOpenService");
        }
        catch (UnavailableServiceException e)
        {
            System.out.println("Service FileOpen non disponible :-( ");
        }

        if (fos == null)
        { System.out.println("Problème avec le service Fichier"); return; }

        try
        {
            fc = fos.openFileDialog(null, null);
            if (fc == null)
            { System.out.println("Problème avec le service Fichier"); return; }

            long longueurAutorisee= fc.getLength();
            System.out.println("Longueur actuelle : " + longueurAutorisee);
        }
    }
}
```

```

if(longueurAutorisee+ 1024 > fc.getMaxLength())
{
    longueurAutorisee=fc.setMaxLength(longueurAutorisee+ 1024);
}

jraf=fc.getRandomAccessFile("rw");
jraf.seek(0);
String ch = ZTNom.getText();
jraf.writeBytes(ch);
jraf.close();
}
catch (IOException e)
{
    System.out.println("Erreur dans l'utilisation du fichier:-( --> "
        + e.getMessage() + " -- exception = " + e.getClass().getName());
}
if (jraf == null) return;
}

...
private java.awt.Button BSauver;
private java.awt.Button BLire;
}

```

On pourra vérifier à l'exécution que le fichier est bien créé.

7.4 Le service de persistance et les muffins

Ce service PersistenceService a pour objectif de permettre à une application Java Web Start, fonctionnant dans un modèle client serveur, de mémoriser sur le client des données qui pourront être réutilisées par des applications provenant de la même URL que l'application qui les a mémorisées. De telles données, dont le rôle est donc assez similaire à celui des **cookies** du protocole http, sont appelées, par analogie pâtissière, des "**muffins**" ;-) ... On ne peut jamais les accéder que par le biais des urls. Par "modèle client-serveur", il faut comprendre une communication réseau impliquant

- ◆ une applet et une servlet connectées selon un tunnel http;
- ◆ deux applications connectées selon TCP au moyen de sockets classiques.

En pratique, le client, après avoir acquis le service **PersistenceService** en plus du BasicService, qui lui sera utile pour obtenir l'URL de provenance de l'application (méthode getCodeBase()) :

```

try
{
    bs = (BasicService)ServiceManager.lookup("javax.jnlp.BasicService");
    ps = (PersistenceService)ServiceManager.lookup("javax.jnlp.PersistenceService");
}
catch (UnavailableServiceException e)
{ System.out.println("Service non disponible :-( " + e.getMessage()); return; }

```

- ◆ crée un muffin au moyen de la méthode :

```
public long create (URL url, long maxsize)
    throws java.net.MalformedURLException, java.io.IOException
```

donc avec une séquence du type :

```
URL url = bs.getCodeBase();
URL urlMuffinIdentifiant = null;
try
{
    urlMuffinIdentifiant = new URL(url, "IdentifiantClient");
    System.out.println("url du muffin = " + urlMuffinIdentifiant);
    ps.create(urlMuffinIdentifiant, 1024);
}
catch (MalformedURLException e)
{
    System.out.println("URL du muffin mal formée :-( " + e.getMessage()); return; }
catch (IOException e)
{
    System.out.println("Problème d'E/S sur le muffin :-( " + e.getMessage()); }
```

- ◆ récupère un muffin au moyen de la méthode :

```
public FileContents get (URL url)
    throws java.net.MalformedURLException, java.io.IOException,
        java.io.FileNotFoundException
```

Pour rappel, **FileContents** est un interface de javax.jnlp et correspond à un objet qui encapsule un nom de fichier et son contenu – la méthode **get()** en fournit une implémentation. C'est prévisible, on en lit le contenu au moyen d'un flux obtenu au moyen de la méthode

```
public InputStream getInputStream() throws java.io.IOException
```

sur lequel on construit simplement un **DataInputStream** :

```
URL url = bs.getCodeBase();
URL urlMuffinIdentifiant = null;
FileContents fc = null;
try
{
    urlMuffinIdentifiant = new URL(url, "IdentifiantClient");
    System.out.println("url du muffin = " + urlMuffinIdentifiant);
    fc = ps.get(urlMuffinIdentifiant);
}
catch (MalformedURLException e)
{
    System.out.println("URL du muffin mal formée :-( " + e.getMessage()); return; }
catch (IOException e)
{
    System.out.println("Problème d'E/S sur le muffin :-( " + e.getMessage()); }
```

```

try
{
    DataInputStream dis = new DataInputStream(fc.getInputStream());
    String identifiant = dis.readUTF();
    System.out.println("Identifiant = " + identifiant);
}
catch (IOException e)
{ System.out.println("Problème d'E/S :-( " + e.getMessage()); return; }

```

La réécriture fonctionne bien sûr selon le mode symétrique en output.

Remarque

A l'image des rowsets, la notion de muffin correspond à une copie locale d'une donnée mémorisée sur le serveur. Elle apporte donc avec elle le problème de la mise en cache et donc de données éventuellement salies. Dans cet esprit, la méthode de PersistenceService

```

public int getTag (URL url)
    throws java.net.MalformedURLException, java.io.IOException

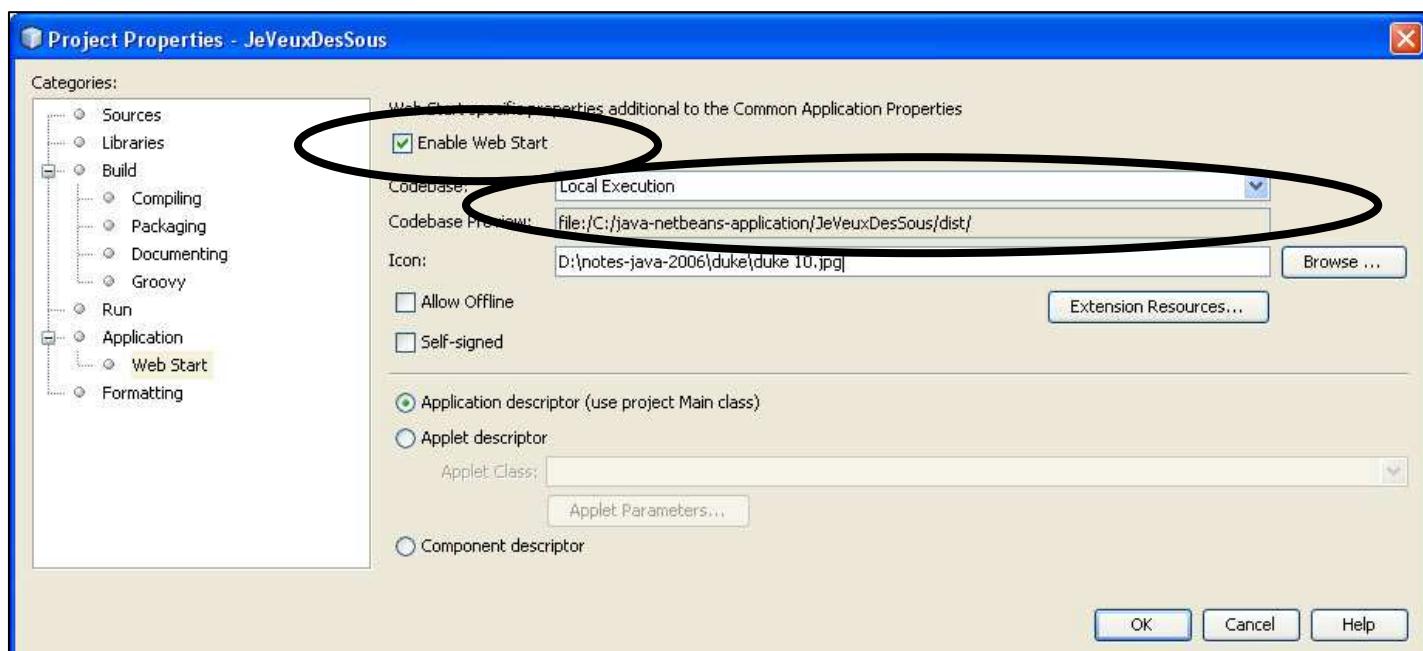
```

permet de connaître l'état d'un muffin; celui-ci peut être :

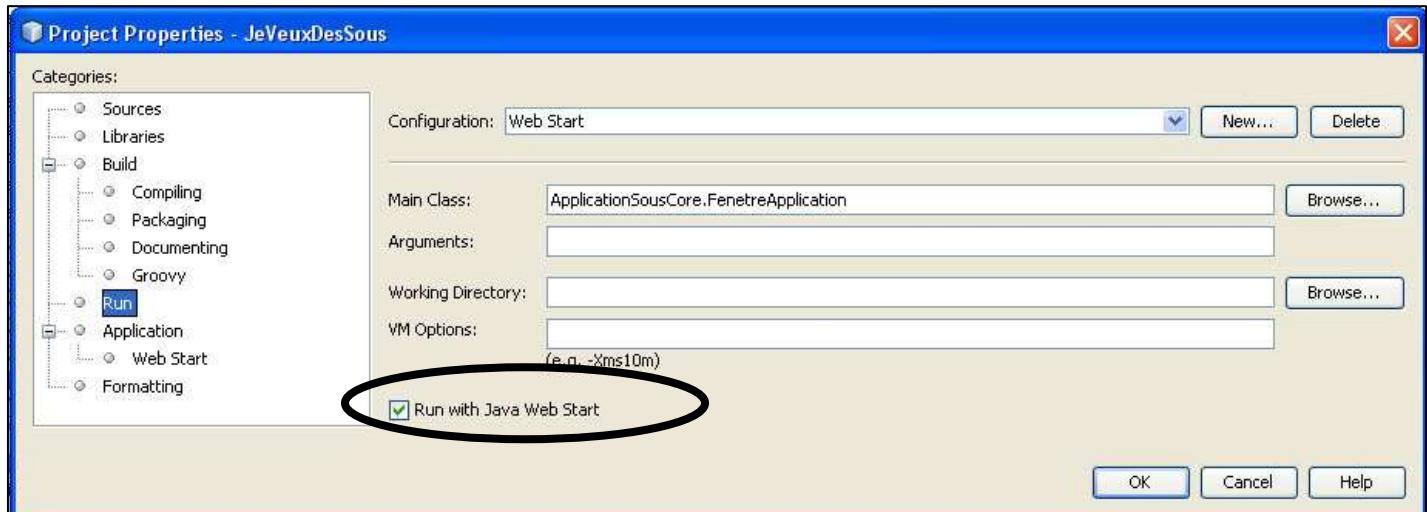
- ◆ CACHED : la version du muffin sur le serveur est la même que celle du client;
- ◆ DIRTY : une mise à jour du muffin sur le client n'a pas été répercutee sur le serveur;
- ◆ TEMPORARY : la donnée peut être recréée à tout moment sans conséquence.

8. Le développement d'une application pour Java Web Start sous Netbeans

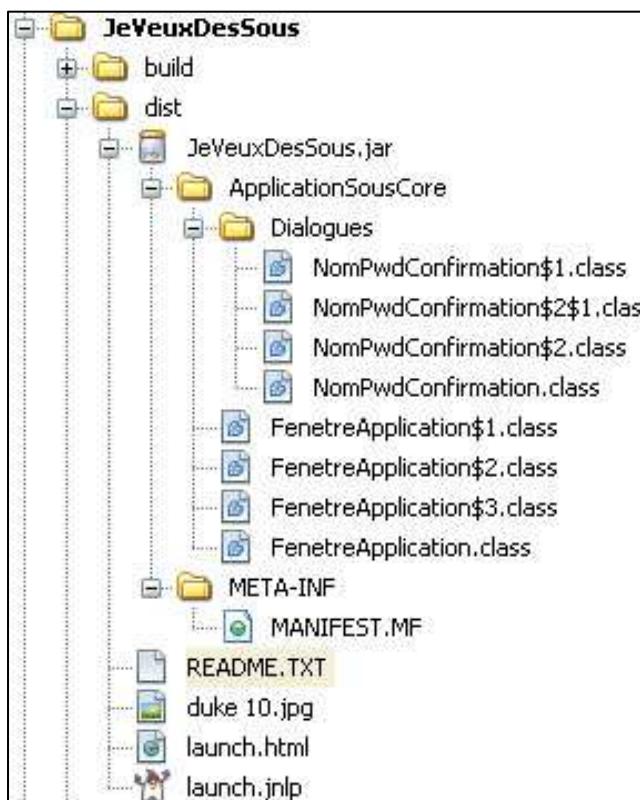
Après avoir développé un projet "Java Application" tout à fait normal, on peut préciser dans les propriétés du projet que le déploiement se fera par Java Web Start :



et même plus précisément en local (pour l'instant) – on peut vérifier que la propriété Run a été affectée :



Après compilation, on peut visualiser les fichiers générés dans l'onglet Files du projet :



Le fichier jnlp contient :

```
launch.jnlp
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<jnlp codebase="file:/C:/java-netbeans-application/JeVeuxDesSous/dist/" href="launch.jnlp" spec="1.0+"></pre>
```

```
<information>
<title>JeVeuxDesSous</title>
<vendor>Vilvens</vendor>
<homepage href="">
<description>JeVeuxDesSous</description>
<description kind="short">JeVeuxDesSous</description>
<icon href="duke 10.jpg" kind="default"/>
</information>

<resources>
<j2se version="1.5+"/>
<jar eager="true" href="JeVeuxDesSous.jar" main="true"/>
</resources>

<application-desc main-class="ApplicationSousCore.FenetreApplication">
</application-desc>

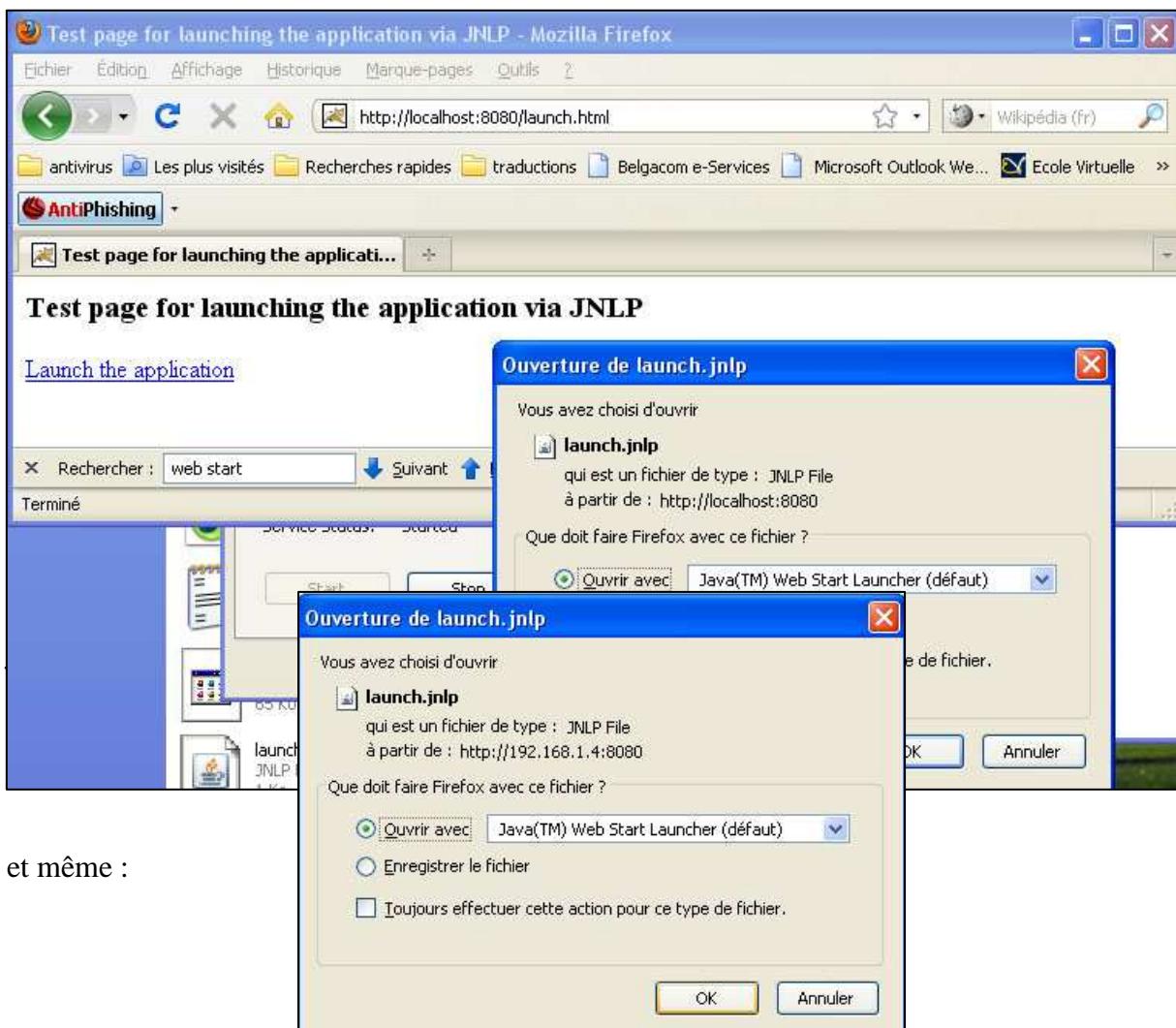
</jnlp>
```

tandis que la page html se limite à :

launch.html

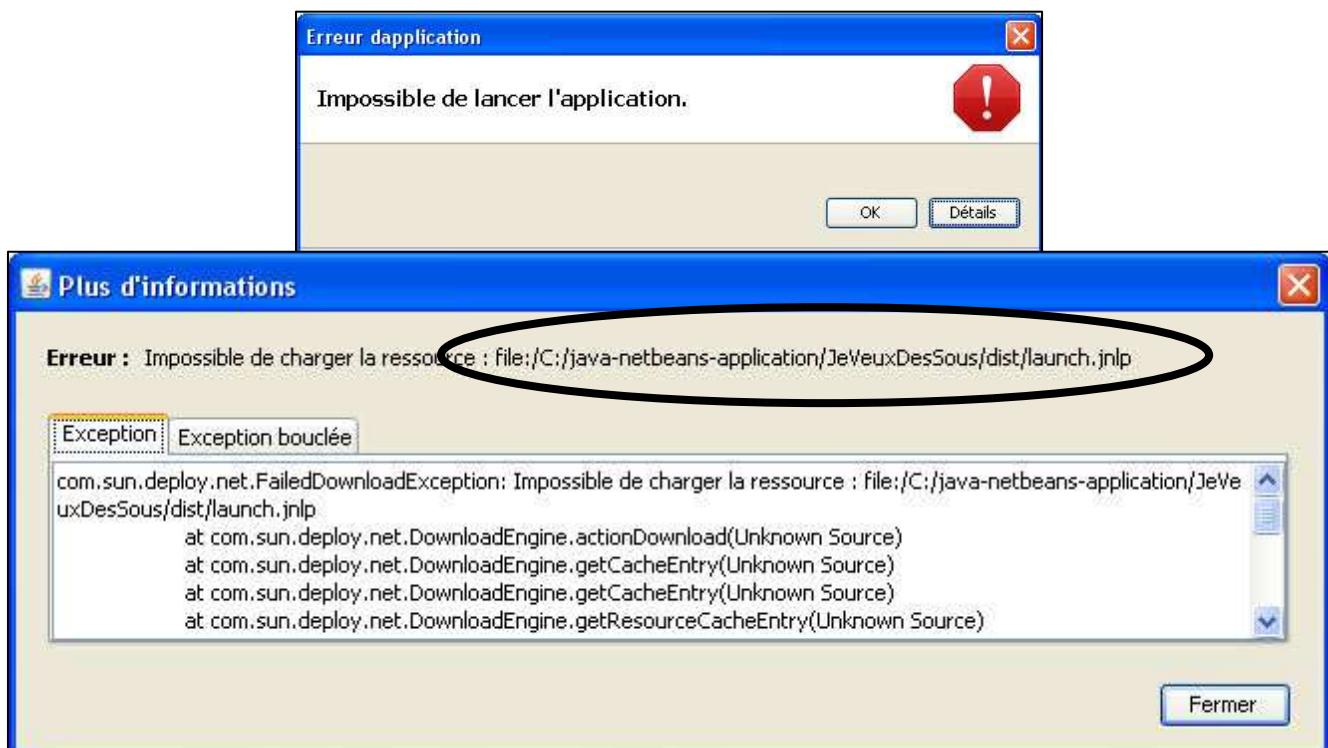
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Test page for launching the application via JNLP</title>
</head>
<body>
<h3>Test page for launching the application via JNLP</h3>
<a href="launch.jnlp">Launch the application</a>
<!-- Or use the following script element to launch with the Deployment Toolkit -->
<!-- Open the deployJava.js script to view its documentation -->
<!--
<script src="http://java.com/js/deployJava.js"></script>
<script>
  var url="http://[fill in your URL]/launch.jnlp"
  deployJava.createWebStartLaunchButton(url, "1.6")
</script>
-->
</body>
</html>
```

Une exécution de l'application donne :

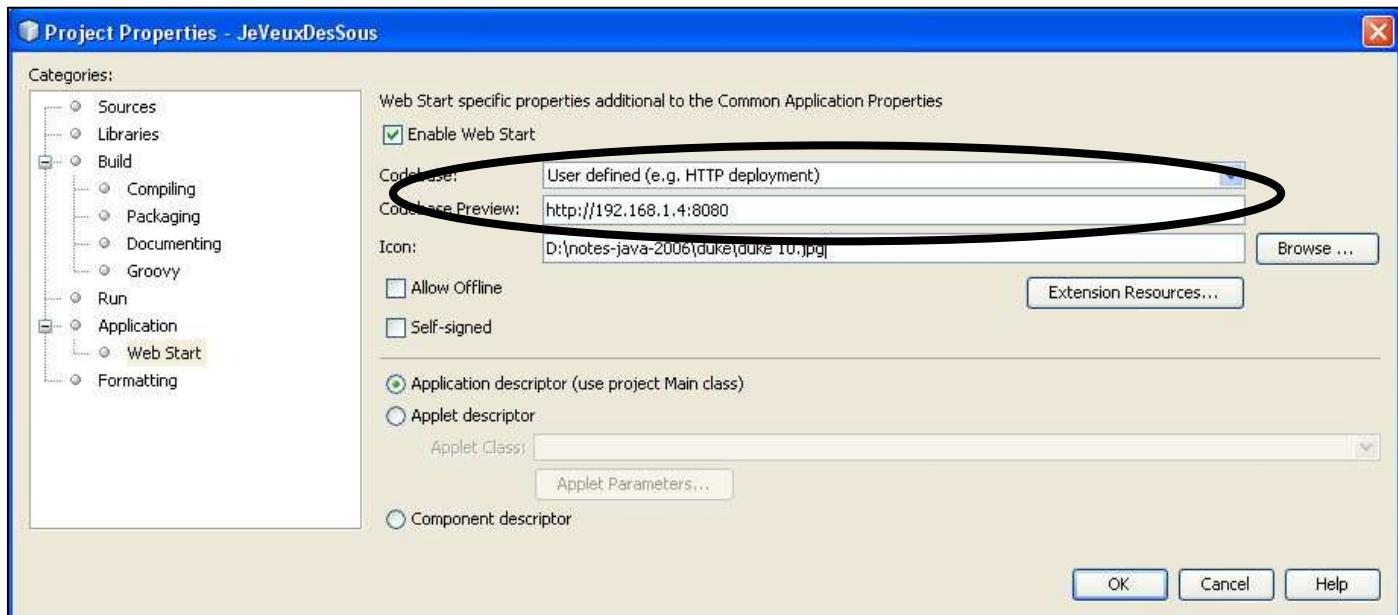


et même :

Si un client distant veut accéder à notre application, il aura évidemment un problème :



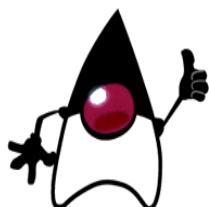
Il faut bien entendu modifier le paramétrage de notre projet :



avec dans le fichier jnlp :

```
...
<jnlp codebase="http://192.168.1.4:8080" href="launch.jnlp" spec="1.0+">>
...
...
```

Plus de problème : l'application est exécutée ☺ !



Quittons ce monde étrange qui mélange applets et applications. Et restons-en là pour le moment : mais il reste encore beaucoup de sujets à aborder ...

à suivre ...

Ouvrages consultés

Ouvrages imprimés et électroniques

Allamaraju, S, et al. Professional Java E-Commerce. Birmingham, United Kingdom. Wrox Press Ltd. 2001.

Flanagan, D., Farley, J., Crawford, W. & Magnusson, K. Java Enterprise in a nutshell. Sebastopol, California, U.S.A. O'Reilly and Associates, Inc. 1999.

Jeugmans, D. Evolution des technologies en serveurs d'application et plate-formes de développement Java. TFE Haute Ecole de la Province de Liège (In.Pr.E.S.). 2008.

Jeunesse, F. Etude synthétique de techniques de sécurité utilisées par les applications réseaux. Seraing, Belgique. TFE Haute Ecole de la Province de Liège (In.Pr.E.S.). 2009.

Puente Gonzalez, I. Etude du protocole Kerberos et déploiement d'une implémentation dans un environnement Unix-Java. TFE Erasmus Haute Ecole de la Province de Liège (I.S.I.L.). 2011.

Vanfrachem, J. Etude de la technologie Hibernate et utilisation dans une application de gestion de notes de cours. TFE Haute Ecole de la Province de Liège (In.Pr.E.S.). 2010.

Vilvens, C. Langage Java (I) : Programmation de base. Seraing, Belgique. A.S.B.L. DEFI. 2011.

Vilvens, C. Langage Java (II) : Programmation avancée des applications classiques. Seraing, Belgique. A.S.B.L. DEFI. 2011.

Vilvens, C. Langage Java (III) : Programmation des applications WEB. Seraing, Belgique. A.S.B.L. DEFI. 2011.

Vilvens, C. Langage Java (IV) : Programmation de protocoles applicatifs et de techniques de sécurité. Seraing, Belgique. A.S.B.L. DEFI. 2011.

White, S., Fischer, M., Catell, R., Hamilton, G. & Hapner, M. JDBC API Tutorial and Reference, Second Edition / The Java Series from the Source. Reading, Massachussettes, U.S.A. Addison-Wesley Publishing Company. 2001.



Sites Internet

<http://java.sun.com/>

avec en particulier :

<http://developer.java.sun.com/>

<http://developer.java.sun.com/developer/onlineTraining/>

<http://developers.sun.com/mobility/midp/articles/deploy/>

<http://www.bejug.org/>

Site du Belgian Java User Group



<http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/contents.html>

Un Java Web Start Guide

<https://beansbinding.dev.java.net/>

<http://netbeans.org/kb/docs/java/gui-binding.html>

<http://wiki.netbeans.org/JavaPersistenceApi>

<http://blogexpertise.alti.com/articles-pdf/TutorialBeansBinding.pdf>

A propos du Bean binding

<https://www.hibernate.org/>

<http://netbeans.org/kb/docs/java/hibernate-java-se.html#03a>

<http://www.jboss.com/pdf/HibernateBrochure-Jun2005.pdf>

A propos de Hibernate