

Java Entreprise Edition et Industrialisation du génie logiciel

- Architecture JEE
- Maven et Junit
- JPA, Hibernate
- Spring IOC et Spring MVC
- Struts 2



Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

Email : med@youssfi.net

Chaîne vidéo : <http://youtube.com/mohamedYoussfi>



Industrialisation du génie logiciel

- Le processus du développement logiciel est, aujourd'hui complètement industrialisé.
- Un logiciel est construit à base de composants
 - Réutilisables
 - Interchangeable
 - Évolutifs
 - Reconfigurables
 - Mobiles
 - Surveillables à chaud



Avènement des technologies Open Source

- En dix ans le développement logiciel a évolué en grande partie grâce aux technologies Open Sources.
- Celles ci permettent aux développeurs de ne pas réinventer perpétuellement la roue et de se concentrer sur les aspects métiers, sans pour autant tomber sous l'emprise technologique d'un éditeur.
- Les technologies Open Source rivalisent avec des standards officiels au point de devenir des standards.
 - Spring
 - Struts
 - Hibernate



Avènement des méthodologies Agile

- Les méthodes Agile de gestion projet et de développement comme Scrum ou l'eXtreme Programming (XP) partent du constat que le cycle de développement en cascade est un échec.
- Développement en cascade:
 - spécifier pendant X mois,
 - puis ensuite coder Y mois,
 - tester Z mois,
 - pour livrer au client un projet qui ne correspond plus tout à fait à ses attentes.



Avènement des méthodologies Agile

- Les méthodes Agile prônent
 - les tests avant le développement,
 - des cycles de développement itératifs,
 - l'implication du client final tout au long du projet,
 - des spécifications réduites en début de projet,
 - etc.
- Les méthodologies Agile sont pragmatiques,
- Ont fait leurs preuves et sont prisées par de grands industriels de logiciels.



Industrialiser le cycle de vie : améliorations, maintenance, corrections

- Pour industrialiser les composants logiciels, il est nécessaire d'utiliser des outils qui permettent d'automatiser le processus de fabrication des logiciels
 - **Frameworks de tests** : JUnit
 - Ils permettent de créer des tests sur des fonctions métiers.
 - Faisant partie intégrante du projet, les tests peuvent être rejoués par tous les développeurs afin de vérifier que les dernières modifications du code ne génèrent aucune régression.
 - **Outils d'intégration continue** : Maven
 - Ils jouent le rôle de chef d'orchestre en lançant automatiquement
 - les tests,
 - le contrôle de qualité du code,
 - Génèrent les builds et la documentation technique des packages ou bibliothèques de classes.
 - L'intégration continue est un élément central de l'industrialisation du cycle de vie.

Exigences d'un projet informatique

- Exigences fonctionnelles:
 - Une application est créée pour répondre , tout d'abord, aux besoins fonctionnels des entreprises.
- Exigences Techniques :
 - Les performances:
 - La maintenance:
 - Sécurité
 - Portabilité
 - Distribution
 - Capacité de communiquer avec d'autres applications distantes.
 - Capacité de fournir le service à différents type de clients (Desk TOP, Mobile, SMS, http...)
 -
- Exigence financières

Constat

- Il est très difficile de développer un système logiciel qui respecte ces exigences sans **utiliser l'expérience des autres** :
 - Serveur d'application JEE:
 - JBOSS, Web Sphere,
 - GlassFish, Tomcat,
 - ...
 - Framework pour l'Inversion de contrôle:
 - Spring (Conteneur léger)
 - EJB (Conteneur lourd)
 - Frameworks :
 - Mapping objet relationnel (ORM) : JPA, Hibernate, Toplink, ...
 - Applications Web : Struts, JSF, SpringMVC
 -
 - Middlewares :
 - RMI, CORBA : Applications distribuées
 - JAXWS, JAXRS our Web services
 - JMS : Communication asynchrone entre les application
 - JMX : Supervision des composants
 - ...

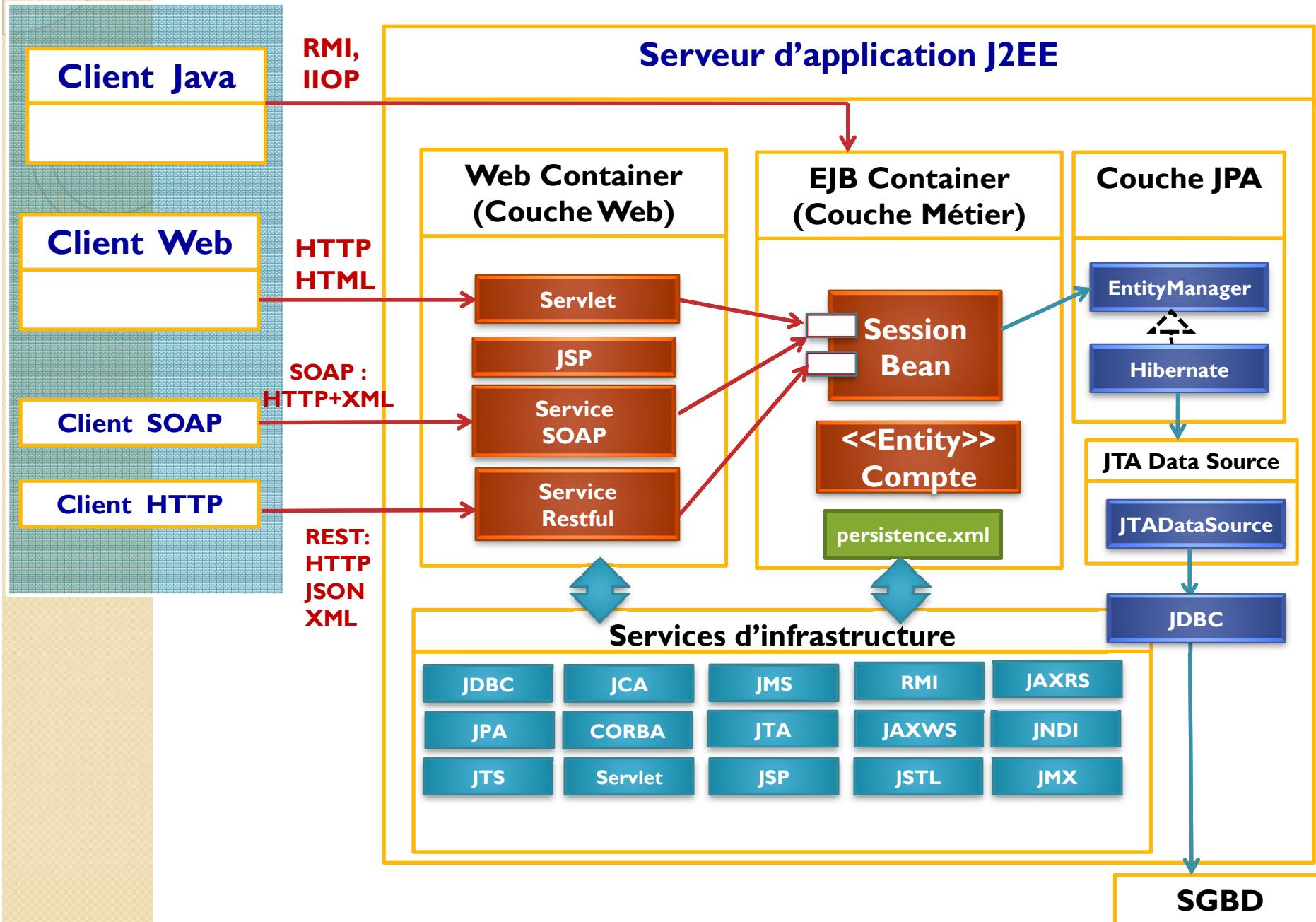


ARCHITECTURE JAVA ENTREPRISE EDITION : JEE

med@youssf.net

Couche Présentation

Architecture JEE





Couche Présentation

- Elle implémente la logique présentation de l'application
- La couche présentation est liée au type de client utilisé :
 - Client Lourd java Desktop:
 - Interfaces graphiques java SWING, AWT, SWT.
 - Ce genre de client peut communiquer directement avec les composants métiers déployés dans le conteneur EJB en utilisant le middleware RMI (Remote Method Invocation)
 - Client Leger Web
 - HTML, Java Script, CSS.
 - Un client web communique avec les composants web Servlet déployés dans le conteneur web du serveur d'application en utilisant le protocole HTTP.
 - Un client .Net, PHP, C++, ...
 - Ce genre de clients développés avec un autre langage de programmation autre que java, communiquent généralement avec les composants Web Services déployés dans le conteneur Web du serveur d'application en utilisant le protocole SOAP (HTTP+XML)
 - Client Mobile
 - Androide, iPhone, Tablette etc..
 - Généralement ce genre de clients communique avec les composants Web Services en utilisant le protocole HTTP ou SOAP



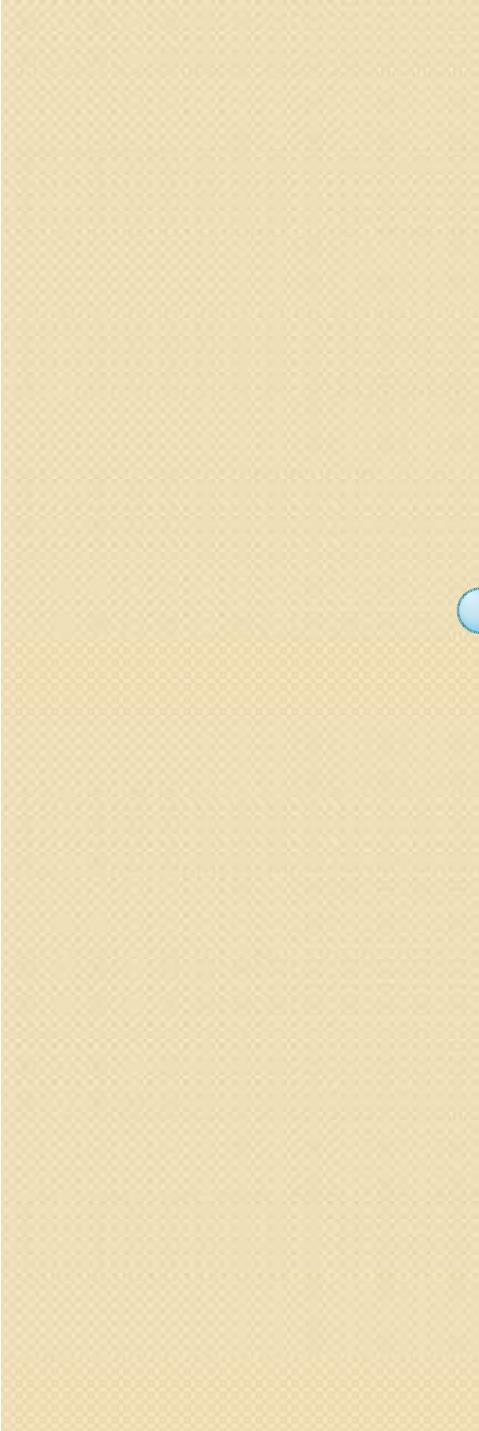
Couche Application ou Web

- Appelée également couche web.
- La couche application sert de médiateur entre la couche présentation et la couche métier.
- Elle contrôle l'enchaînement des tâches offertes par l'application
 - Elle reçoit les requêtes http clientes
 - Assure le suivi des sessions
 - Vérifier les autorisations d'accès de chaque session
 - Assure la validation des données envoyées par le client
 - Fait appel au composants métier pour assurer les traitements nécessaires
 - Génère une vue qui sera envoyée à la couche présentation.
- Elle utilise les composants web Servlet et JSP
- Elle respecte le modèle MVC (Modèle Vue Contrôleur)
- Des framework comme JSF, SpringMVC ou Struts sont généralement utilisés dans cette couche.



Couche Métier ou service

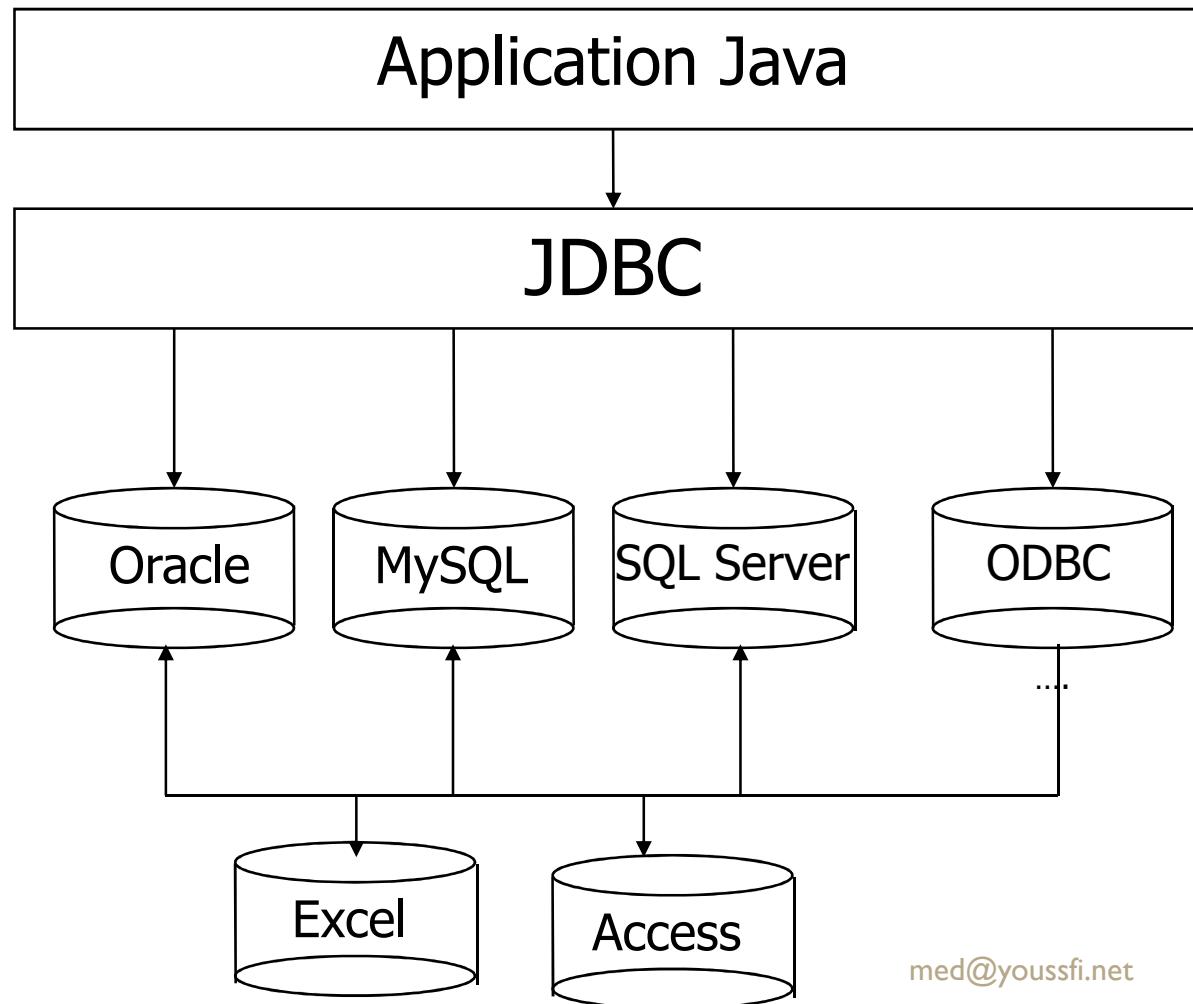
- La couche métier est la couche principale de toute application
 - Elle implémente la logique métier d'une entreprise
 - Elle se charge de récupérer, à partir des différentes sources de données, les données nécessaires pour assurer les traitements métiers déclenchés par la couche application.
 - Elle assure la gestion du WorkFlow (Processus de traitement métier en plusieurs étapes)
- Il est cependant important de séparer la partie accès aux données (Couche DAO) de la partie traitement de la logique métier (Couche Métier) pour les raisons suivantes :
 - Ne pas se perdre entre le code métier, qui est parfois complexe, et le code d'accès aux données qui est élémentaire mais conséquent.
 - Ajouter un niveau d'abstraction sur l'accès aux données pour être plus modulable et par conséquent indépendant de la nature des unités de stockage de données.
 - La couche métier est souvent stable. Il est rare qu'on change les processus métier. Alors que la couche DAO n'est pas stable. Il arrive souvent qu'on est contraint de changer de SGBD ou de répartir et distribuer les bases de données.
 - Faciliter la répartition des tâches entre les équipes de développement.
 - Déléguer la couche DAO à frameworks spécialisés dans l'accès aux données (Hibernate, Toplink, etc...)



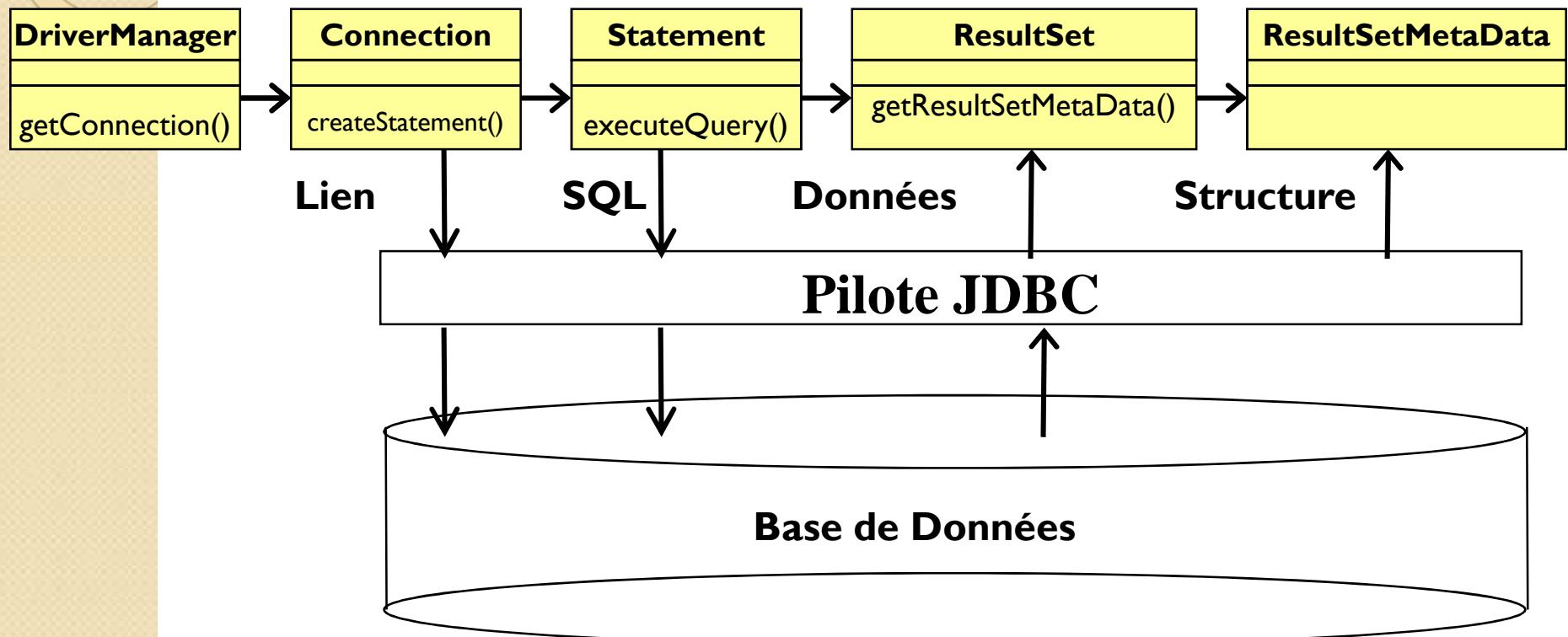
- **SERVICES
D'INFRASTRUCTURES
D'UN SERVEUR
D'APPLICATION JEE**

JDBC

- **JDBC** (Java Data Base Connectivity) : Pilotes java pour permettre l'accès aux bases de données.
- Chaque SGBD possède ses propres pilotes JDBC.



JDBC





JPA

- **JPA** (Java Persistence API).
- Interface qui permet de gérer la persistance des Entity d'une application
- JPA définit un ensemble d'annotations qui permettent d'effectuer le Mapping objet relationnel (ORM).
- JPA définit également une interface de gestion de la persistance des entités, dans un context transactionnel, appelée **EntityManager**.
- Le serveur d'application fournie un Framework implémentant la spécification JPA pour assurer le mapping objet relationnel (Pour JBOSS c'est Hibernate qui est utilisé)

JPA

- Exemple de classe persistante en utilisant les annotations JPA

```
package metier.entities;
import java.io.Serializable;
import java.util.Date;
import javax.persistence.*;
@Entity
@Table(name="COMPTES")
public class Compte implements Serializable {
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
@Column(name="CODE")
private Long code;
@Column(name="SOLDE")
private double solde;
@Temporal(TemporalType.TIMESTAMP)
@Column(name="DATE_CREATION")
private Date dateCreation;
// Getters et Setters
// Constructeur sans paramètre
// Constructeur avec paramètres
}
```

JPA

- Exemple de Mapping Objet Relationnel avec EntityManager

```
package metier.session;

import java.util.List; import javax.ejb.Stateless;
import javax.persistence.*; import metier.entities.Compte;
public class BanqueJpaImpl implements IBanqueDAO {

    @PersistenceContext(unitName="UP_BP")
    private EntityManager em;
    @Override
    public void addCompte(Compte c) {
        em.persist(c);
    }
    @Override
    public List<Compte> consulterComptes() {
        Query req=em.createQuery("select c from Compte c");
        return req.getResultList();
    }
}
```

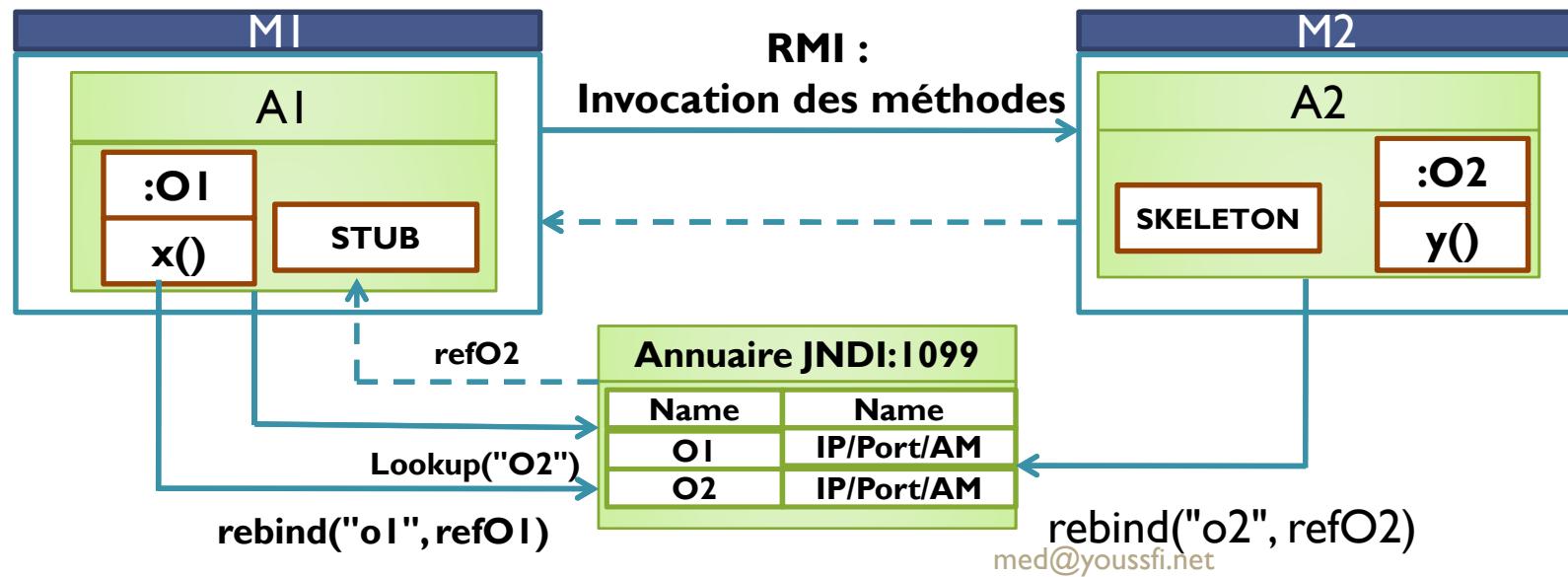
JPA

- Exemple de Mapping Objet Relationnel avec EntityManager

```
@Override  
public Compte consulterCompte(Long code) {  
    Compte cp=em.find(Compte.class,code);  
    if(cp==null) throw new RuntimeException("Ce compte n'existe pas");  
    return cp;  
}  
  
@Override  
public void verser(Long code, double montant) {  
    Compte cp=this.consulterCompte(code);  
    cp.setSolde(cp.getSolde()+montant);  
    em.persist(cp);  
}  
  
@Override  
public void retirer(Long code, double montant) {  
    Compte cp=this.consulterCompte(code);  
    if(cp.getSolde()<montant) throw new RuntimeException("Solde insuffisant");  
    cp.setSolde(cp.getSolde()-montant);  
}
```

RMI

- RMI : Remote Method Invocation
- Est un middleware java qui permet de créer des applications orientée objets distribués.
- Avec RMI un Objet java O1, déployé dans une application A1, dans une machine M1 peut faire appel aux méthode d'une autre objet O2, déployé dans une application A2, dans une machine M2 à travers des intermédiaires :
 - STUB : Proxy Coté Client
 - SKELETON : Proxy Coté Serveur
- Pour cela l'objet O1 a besoin de :
 - L'interface de l'objet O2 (Interface Remote)
 - La référence de l'objet O2 (IP+Port+Adresse mémoire)
- Cette référence est publié dans un annuaire JNDI en lui associant un nom fixe.



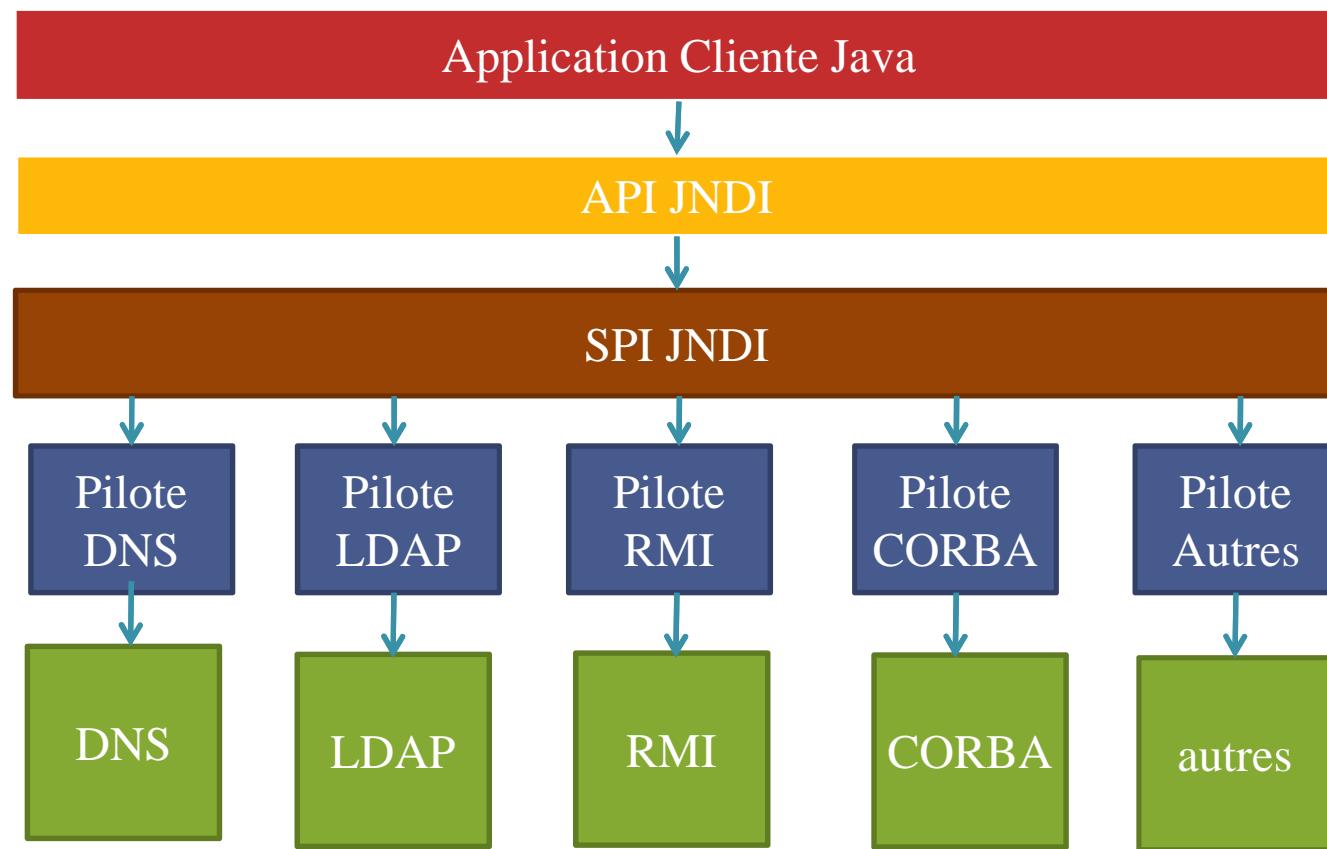


JNDI

- JNDI est l'acronyme de Java Naming and Directory Interface.
- Cette API fournit une interface unique pour utiliser différents services de nomenclature ou d'annuaires et définit une API standard pour permettre l'accès à ces services.
- Il existe plusieurs types de service de nommage parmi lesquels :
 - DNS (Domain Name System) : service de nommage utilisé sur internet pour permettre la correspondance entre un nom de domaine et une adresse IP
 - LDAP(Lightweight Directory Access Protocol) : annuaire
 - NIS (Network Information System) : service de nommage réseau développé par Sun Microsystems
 - COS Naming (Common Object Services) : service de nommage utilisé par Corba pour stocker et obtenir des références sur des objets Corba
 - RMI Registry : service de nommage utilisé par RMI
 - etc, ...

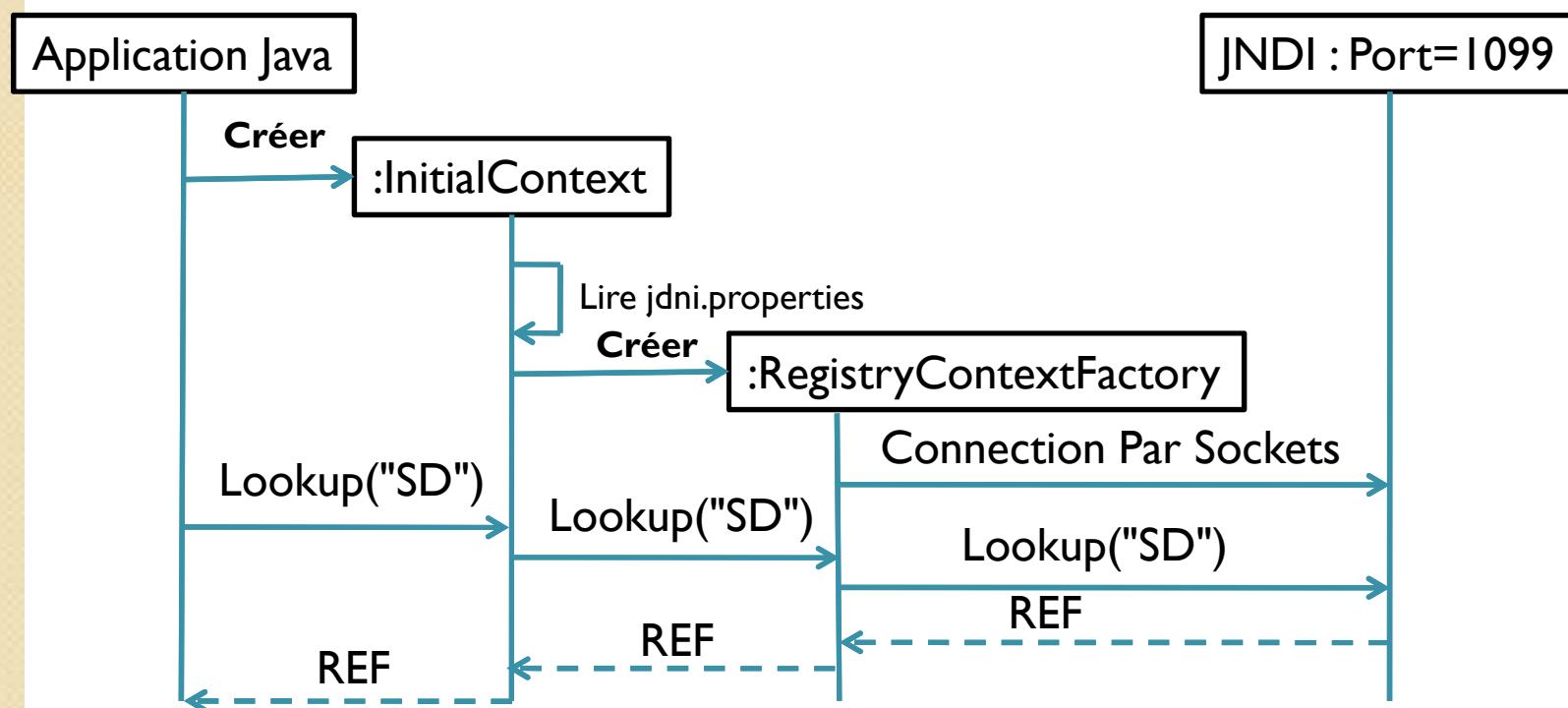
Architecture JNDI

- L'architecture de JNDI se compose d'une API et d'un Service Provider Interface (SPI).
- Les applications Java emploient l'API JNDI pour accéder à une variété de services de nommage et d'annuaire.
- Le SPI permet de relier, de manière transparente, une variété de services de nommage et d'annuaire ; permettant ainsi à l'application Java d'accéder à ces services en utilisant l'API JNDI



JNDI

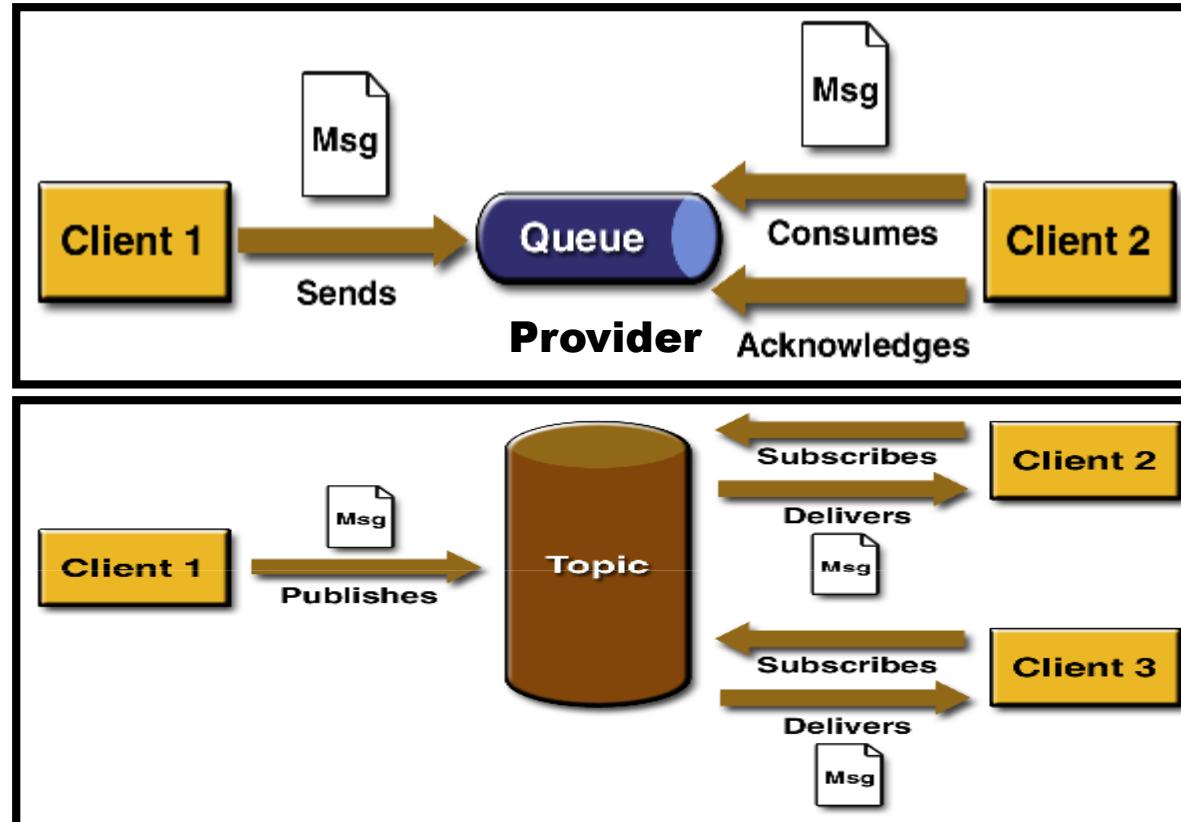
```
// Créer l'objet InitialContext JNDI en utilisant le fichier jndi.properties  
Context ctx=new InitialContext();  
// Publier la référence de l'objet distant avec le nom SD  
ctx.bind("SD", refObj);  
// Récupérer la référence d'un objet dont le nom est BK  
Object refObj=ctx.lookup("SD");
```



Fichier jndi.properties :

```
java.naming.factory.initial=com.sun.jndi.rmi.registry.RegistryContextFactory  
java.naming.provider.url=rmi://localhost:1099
```

JMS : Java Message Service



- **JMS**, ou **Java Message Service**, est une API d'échanges de messages pour permettre un dialogue entre applications via un fournisseur (**Provider**) de messages.
- L'application cliente envoie un message dans une **liste d'attente**, sans se soucier de la disponibilité de cette application.
- Le client a, de la part du fournisseur de messages, une garantie de **qualité de service** (certitude de remise au destinataire, délai de remise, etc.).



JMS

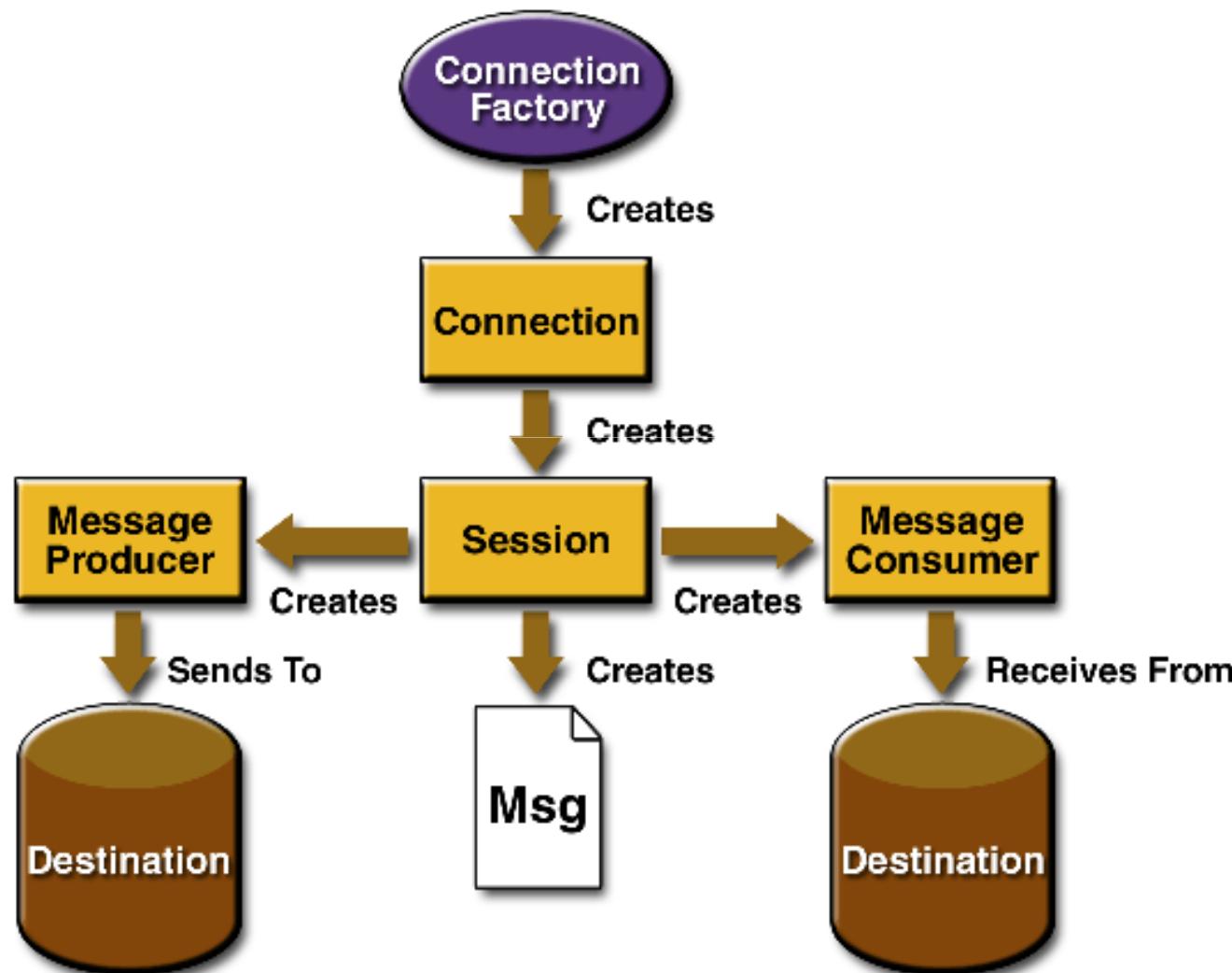
- JMS est une API, spécifiée en 1998, pour la création, l'envoie et la réception des messages de façon :
 - **Asynchrone** : un client reçoit les messages dès qu'ils se connectent ou lorsque le client est disponible et sans avoir à demander si un message est disponible.
 - **Fiable** : le message est délivré une fois et une seule.
- JMS peut accéder aux **Messaging-oriented middleware (MOM)**, tels que
 - MQSeries d'IBM,
 - JBoss Messaging,
 - One Message Queue de Sun Microsystem, ...



Les protocoles de communication

- Avant l'arrivée de JMS, les produits proposaient une gestion de messages selon deux types de protocoles :
 - un protocole **point-à-point (Queue)**,
 - un protocole **publier/souscrire (Topic)**
- Les JMS Provider compatible J2EE 1.3 doivent obligatoirement implémenter ces deux protocoles

Le modèle de programmation JMS





JMX : Java Management Extensions

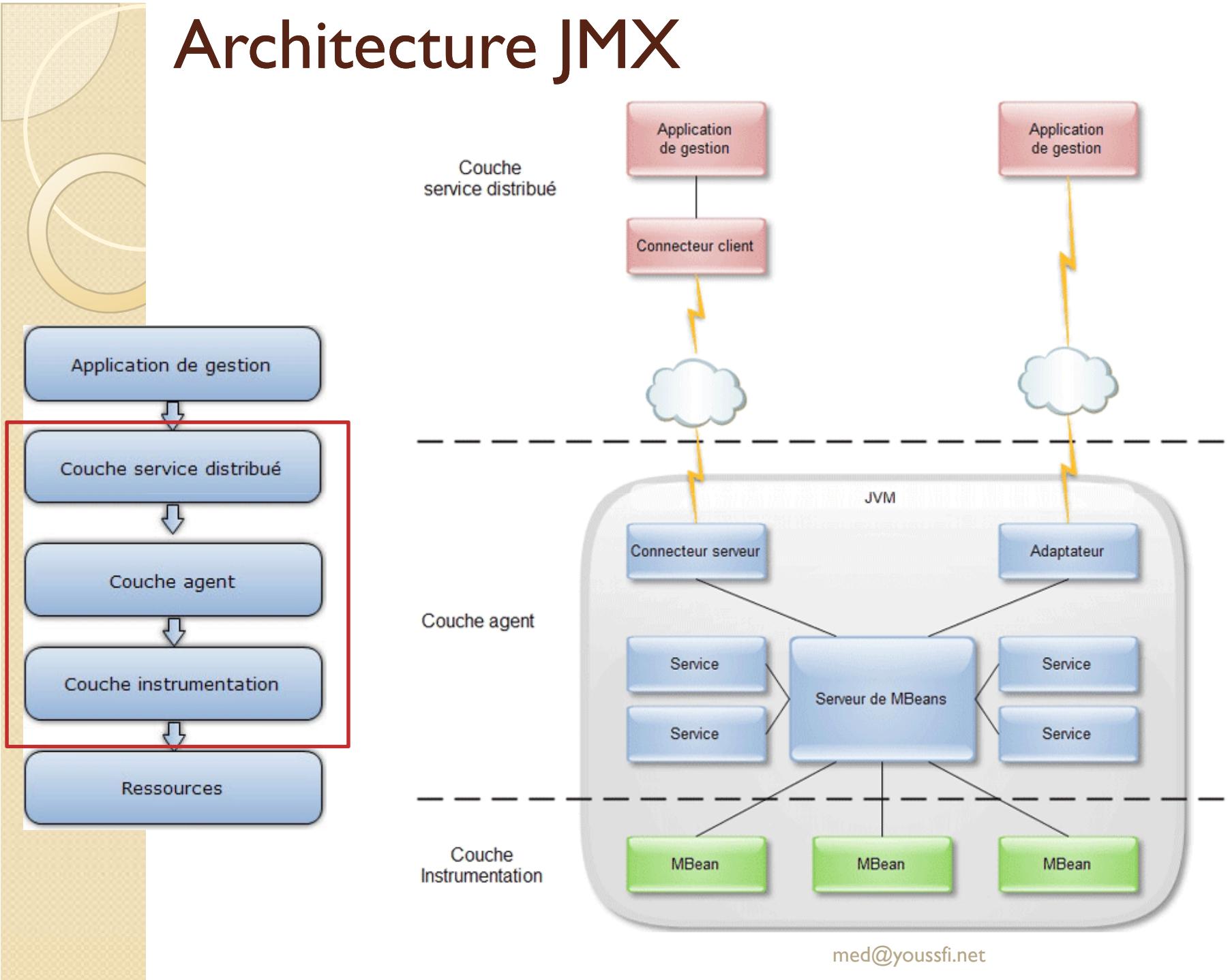
- JMX est l'acronyme de Java Management Extensions.
- JMX permet de configurer , surveiller et administrer les ressources d'une application à chaud.
- JMX est une spécification qui définit :
 - Une architecture,
 - Une API
 - et des services
- Son but est de proposer un standard pour faciliter le développement de systèmes de contrôle, d'administration et de supervision des applications et des ressources.
- JMX peut permettre de configurer, gérer et maintenir une application durant son exécution en fonction des fonctionnalités développées.
- Il peut aussi favoriser l'anticipation de certains problèmes par une information sur les événements critiques de l'application ou du système.



JMX : Java Management Extensions

- JMX repose sur une architecture à trois niveaux :
 - Services distribués : cette couche définit la partie IHM. C'est généralement une application qui permet de consulter les données relatives à l'application et d'interagir avec elles. Cette couche utilise des connecteurs et des adaptateurs de protocoles pour permettre à des outils de gestion de se connecter à un agent.
 - Agent : cette couche définit un serveur de MBeans qui gère les Managed Beans. Elle propose des fonctionnalités sous la forme d'un agent JMX et assure la communication avec la couche services distribués grâce à des Connectors et des Adapters
 - Instrumentation : cette couche définit des MBeans qui permettent l'instrumentation d'une ressource (application, service, composant, objet, appareil, ...) grâce à des attributs, des opérations et des événements.
- Ressources gérées (composants de l'application, services, périphériques, ...) : cette couche n'est pas directement concernée par l'API JMX

Architecture JMX



Exemple de MBean

Le nom de l'interface d'un MBean JMX doit se terminer par **MBean**

```
package jmx.beans;  
public interface PremierMBean {  
    public String getNom();  
    public int getValeur();  
    public void setValeur(int valeur);  
    public void rafraichir();  
}
```

Un exemple d'un MBean implémentant cette interface

```
package jmx.beans;  
public class Premier implements PremierMBean {  
    private static String nom="PremierMBean";  private int valeur=100;  
    public String getNom() { return nom; }  
    public int getValeur() { return valeur; }  
    public void setValeur(int valeur) { this.valeur=valeur; }  
    public void rafraichir() {  
        System.out.println("Raffrichier Les données");  
    }  
}
```

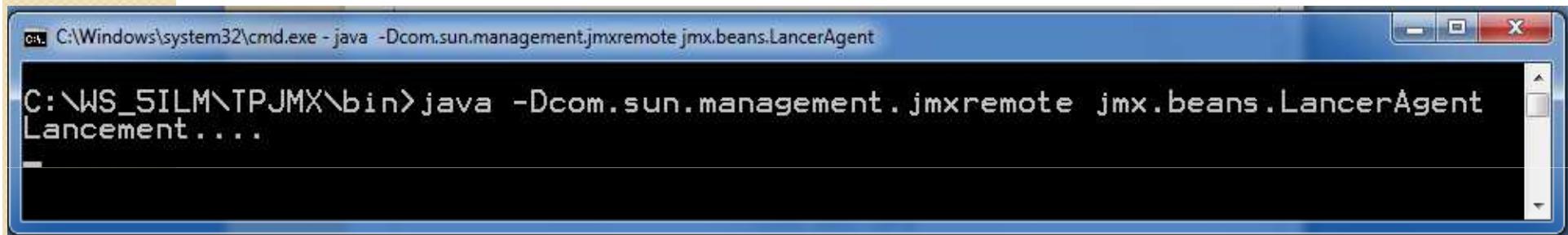
JMX : Démarrage d'un Agent JMX

- Il faut définir une application qui va créer un serveur de MBeans, instancier le MBean et l'enregistrer dans le serveur.

```
package jmx.beans;
import java.lang.management.ManagementFactory;
import javax.management.*;
public class LancerAgent {
public static void main(String[] args) {
    // Créer Un serveur de MBeans
    MBeanServer mbs=ManagementFactory.getPlatformMBeanServer();
    ObjectName name=null;
    try {
        // Créer Un nom JMX
        name=new ObjectName("jmx.beans:type=IPremierMBean");
        Premier mbean=new Premier(); // Créer le MBean
        mbs.registerMBean(mbean, name); // Enregistrer le MBean
        System.out.println("Lancement....");
        // Le Mbean en activité
        while(true){
            Thread.sleep(1000);
            mbean.setValeur(mbean.getValeur()+1);
        }
    } catch (Exception e) { e.printStackTrace(); }
}
```

JMX : Démarrage de l'Agent JMX

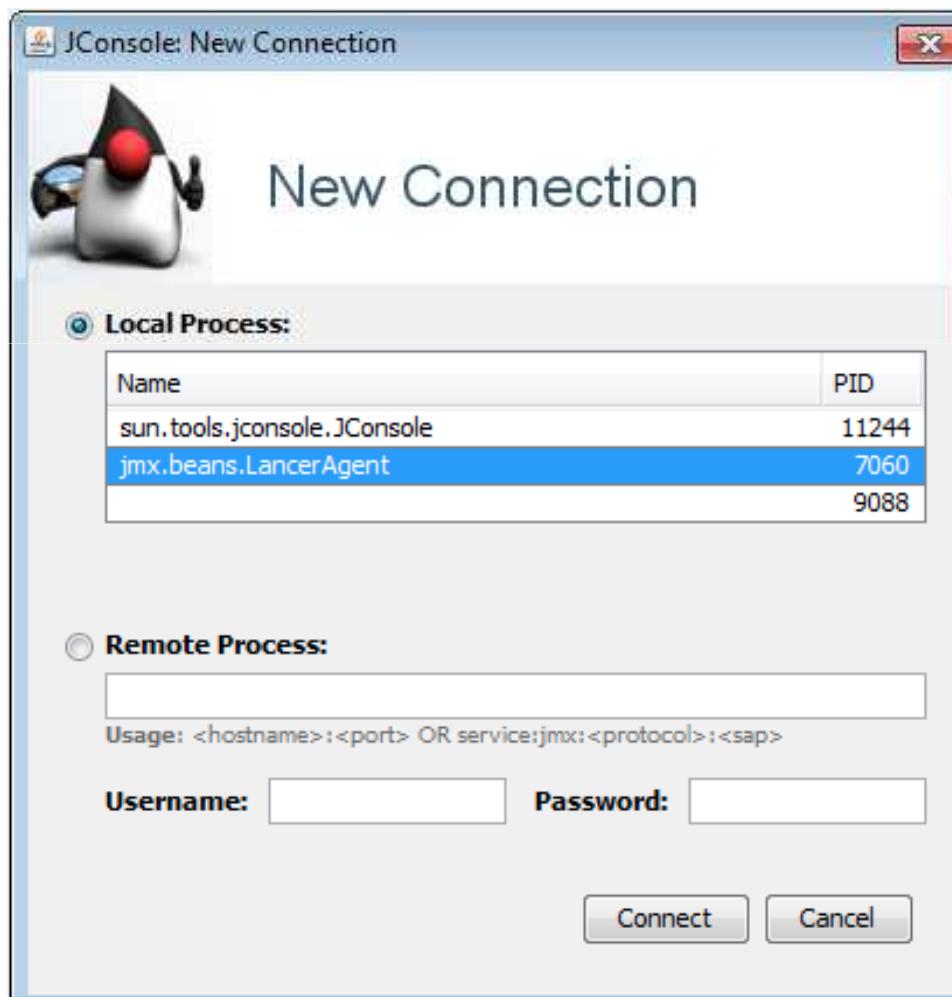
- Il faut compiler la classe et l'exécuter en demandant l'activation de l'accès distant aux fonctionnalités de JMX
- Exécuter le conteneur JMX sur ligne de commande :
 - `java -Dcom.sun.management.jmxremote jmx.beans.LancerAgent`



A screenshot of a Windows command prompt window titled "cmd C:\Windows\system32\cmd.exe". The window shows the command: `java -Dcom.sun.management.jmxremote jmx.beans.LancerAgent`. Below the command, the text "Lancement...." is displayed, indicating the agent is starting up.

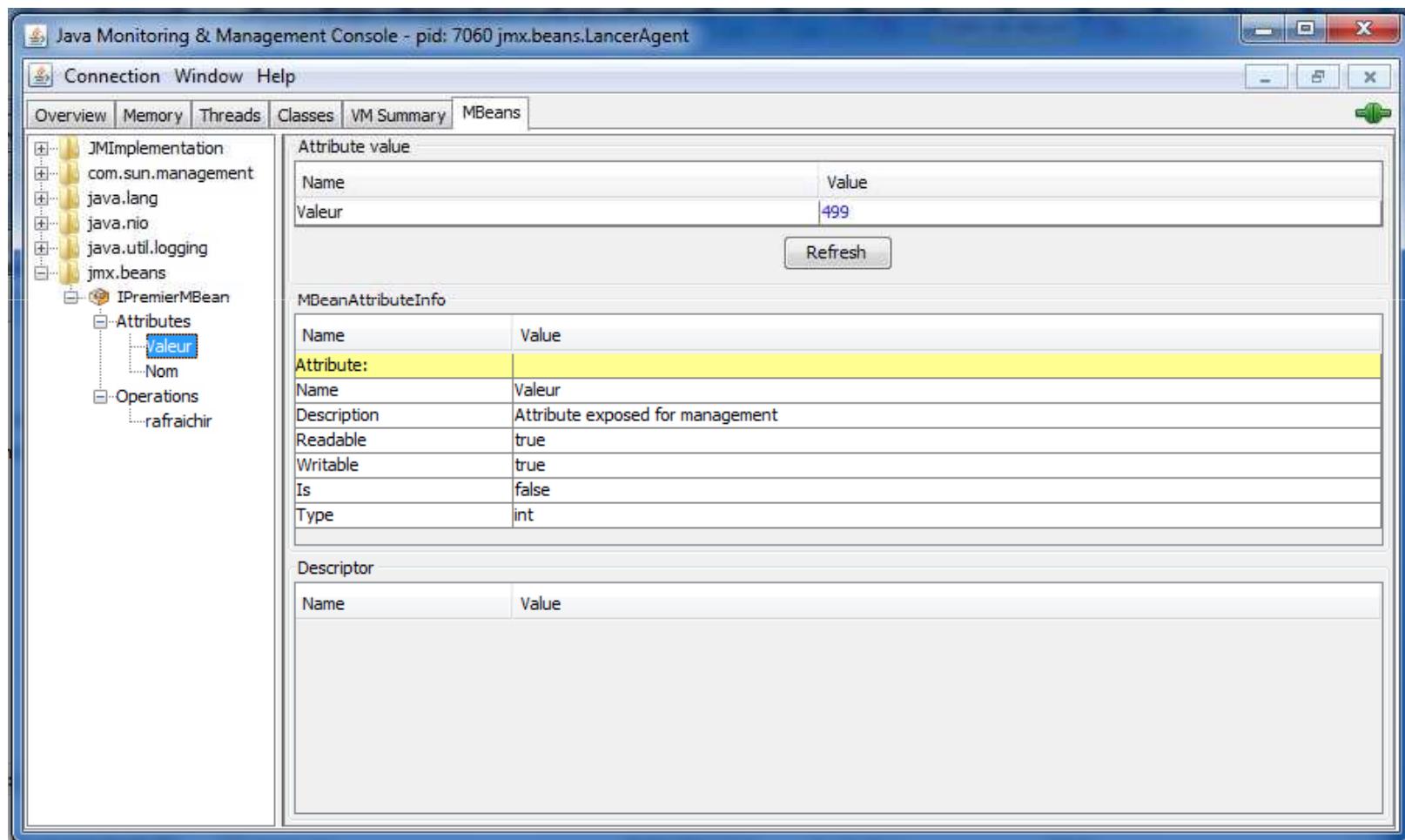
JMX : Surveiller le MBean avec jConsole

- Exécuter l'outil jConsole de Java
- Sélectionner L'application à surveiller LancerAgent
- Cliquer sur le bouton connecter



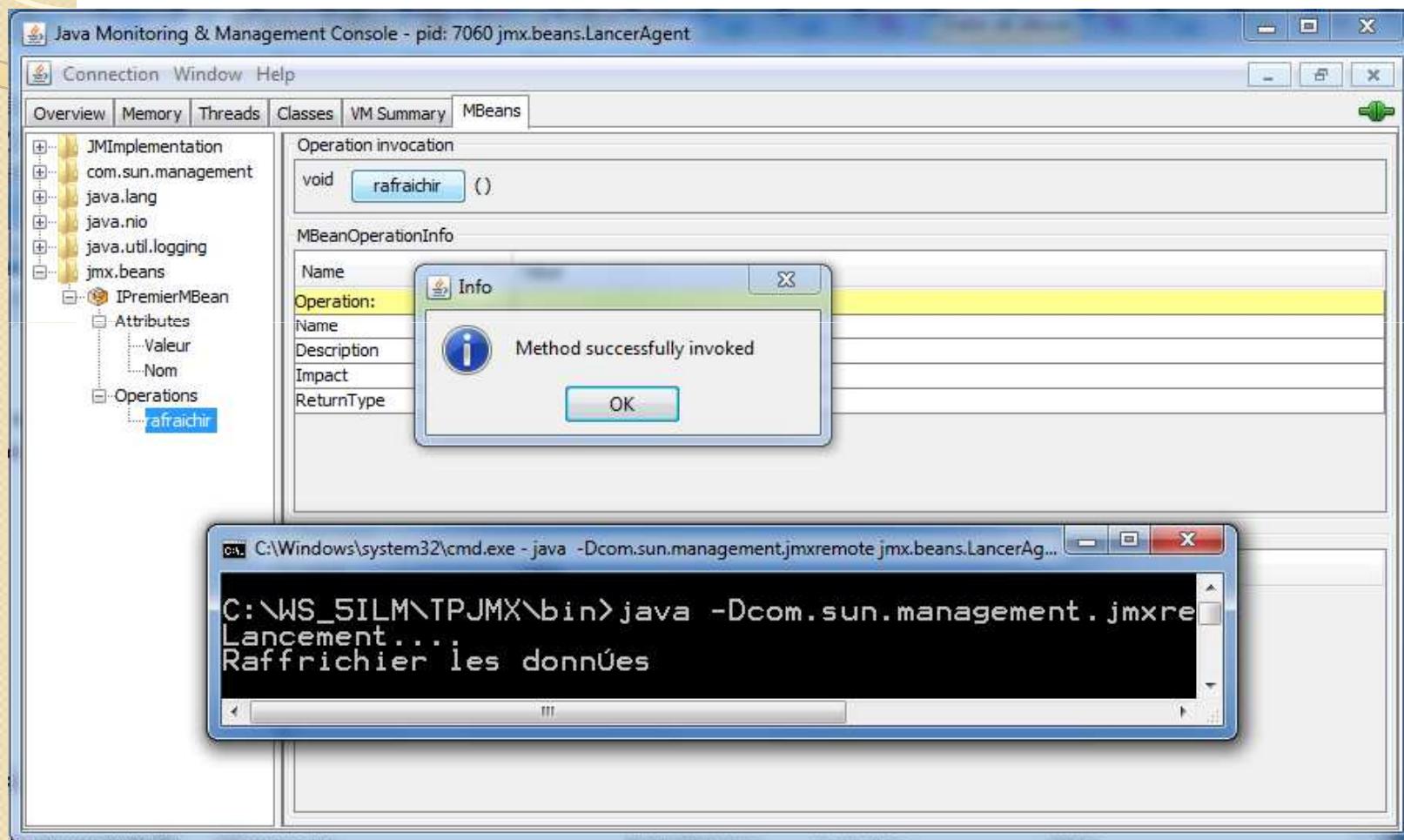
JMX : Surveiller le MBean avec jConsole

- Onglet Mbeans
- Selectionner PremierMBean
- Consulter la valeur des attributs du Mbean à chaud

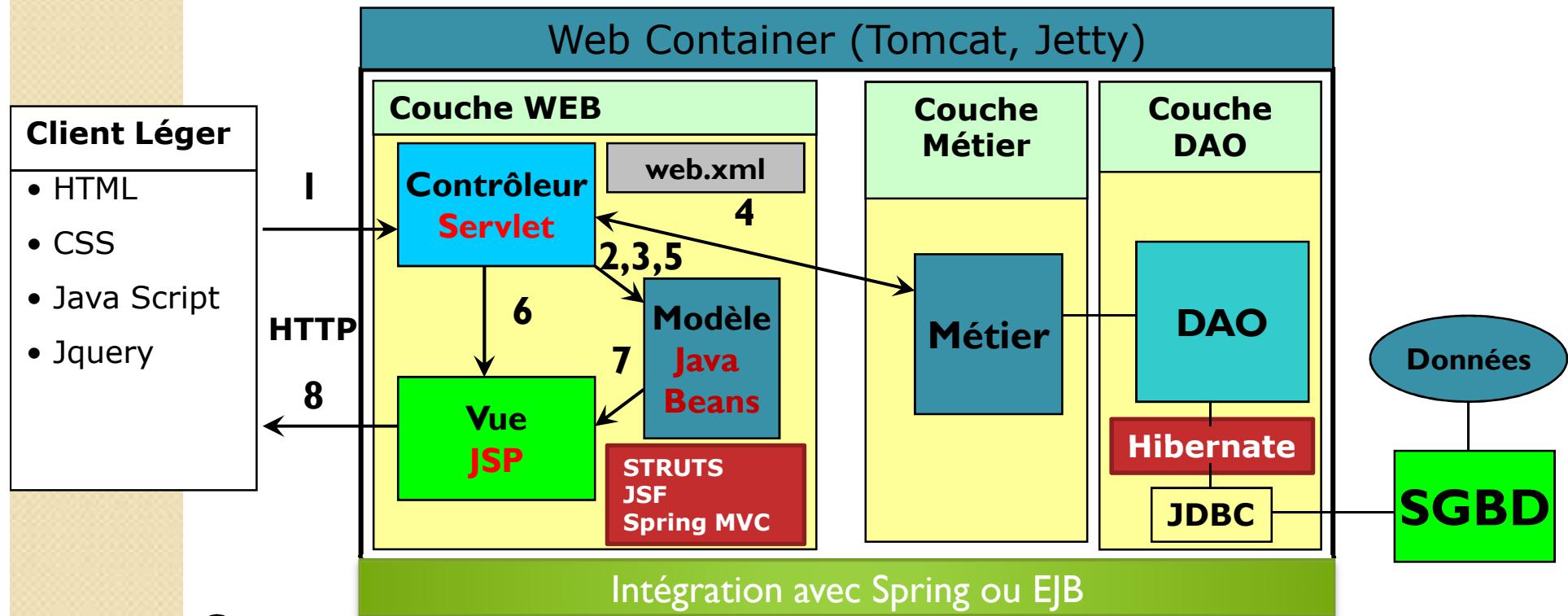


JMX : Surveiller le MBean avec jConsole

- Exécuter l'opération rafraîchir via la console JMX.



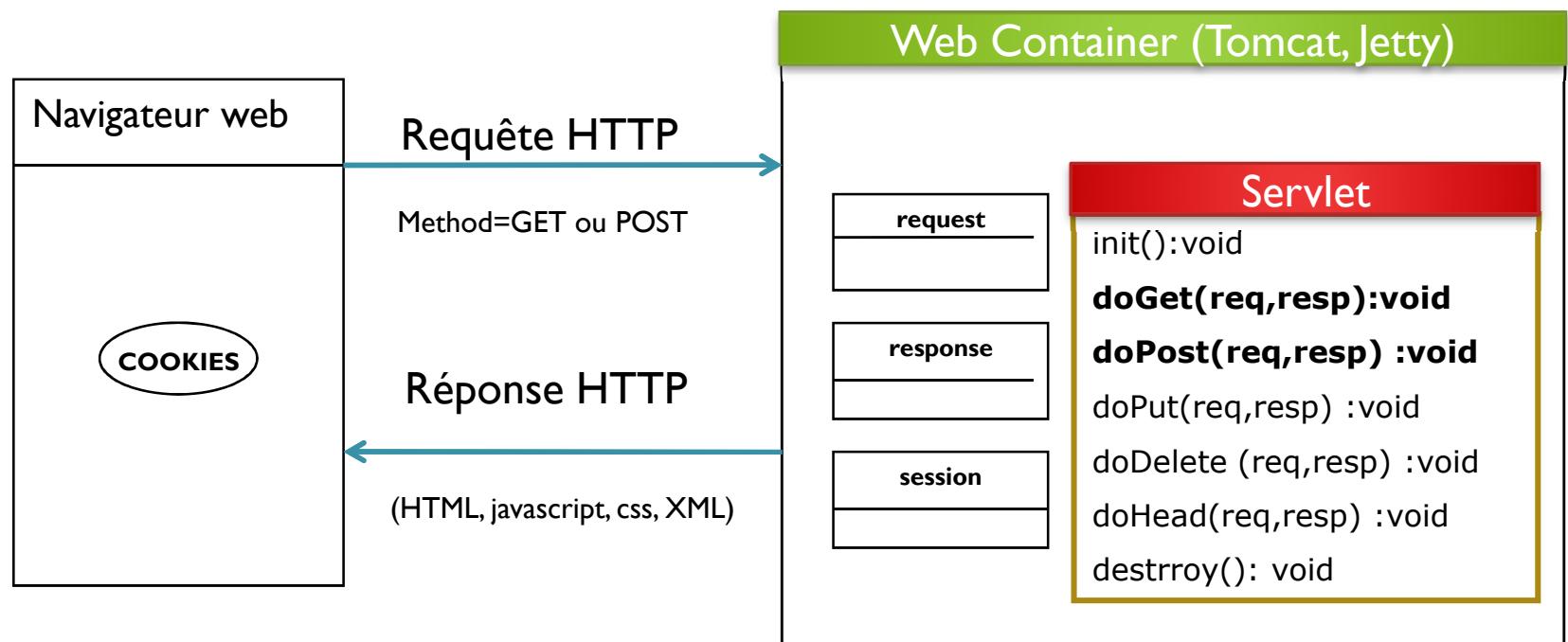
Architecture Web J2EE



- 1 Le client envoie la requête au contrôleur
- 2 Le contrôleur instancie le modèle
- 3 Le contrôleur Stocke les données de la requête dans le modèle puis vérifie la validité des données
- 4 Le contrôleur Fait appel à la couche métier pour faire les traitement
- 5 Le contrôleur Stocke les résultats de traitement dans le modèle
- 6 Le contrôleur fait un forward vers la vue JSP
- 7 La vue récupère les résultats du modèle
- 8 La vue envoie le résultat au Client Léger

Servlet

- Les Servlets sont des composants web JEE
- Permettent de gérer des requêtes HTTP et de fournir au client une réponse HTTP
- Une Servlet s'exécute dans un **moteur de Servlet** ou **conteneur de Servlet** qui gère son cycle de vie.



Exemple de servlet

```
package web; import java.io.*; import javax.servlet.*; import javax.servlet.http.*; import metier.*;
public class ControleurServlet extends HttpServlet {
    private ICatalogueMetier metier;
    @Override
    public void init() throws ServletException { metier=new CatalogueMetierImpl(); }
    @Override
    protected void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException {         doPost(request, response); }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        String action=request.getParameter("action");
        if(action!=null){
            if(action.equals("Save")){
                String des=request.getParameter("designation");
                double prix=Double.parseDouble(request.getParameter("prix"));
                int qte=Integer.parseInt(request.getParameter("quantite"));
                metier.addProduit(new Produit(des, prix, qte));  }}
            request.setAttribute("produits", metier.listProduits());
            request.getRequestDispatcher("vues/Produits.jsp").forward(request, response);
        } }
```

Déploiement d'une servlet

- Première solution : Descripteur de déploiement de servlet : **web.xml**

```
<servlet>
    < servlet-name>cs</servlet-name>
    < servlet-class>web.ControleurServlet</servlet-class>
</servlet>
<servlet-mapping>
    < servlet-name>cs</servlet-name>
    < url-pattern>*.do</url-pattern>
</servlet-mapping>
```

- Deuxième solution (Servlet 3.x): **Annotations**

```
@WebServlet(name="cs",urlPatterns={"/controleur","*.do"})
public class FirstServlet extends HttpServlet {

}
```



JSP : Java Server Pages

- Les Servlets sont très pratiques pour déclencher des traitements coté serveur suite à une requête HTTP
- Dans une Servlet, on peut également générer une réponse HTTP en utilisant l'objet response.
- Mais, quant il s'agit de générer une page web complète, les Servlets ne sont pas pratiques.
- Il existe un autre moyen pour créer les servlet, c'est les page JSP.
- Les page JSP sont très pratiques pour générer des pages web dynamiques.
- Une page JSP est une sorte de page HTML dans laquelle, on peut écrire du code java.
- Dans une application web JEE,
 - Les Servlets jouent le rôle du contrôleur de l'application
 - Les JSP joue le rôle des vues.
- Quand une page jsp est appelée pour la première fois, le web container la convertie en servlet.

Exemple de page JSP

```
<%@page import="metier.Client"%><%@page import="java.util.List"%>
<%@page import="web.ModeleClient"%>
```

→ Directives

```
<%
ModeleClient mod;
if(request.getAttribute("modele")!=null){
    mod=(ModeleClient)request.getAttribute("modele");
}
else{ mod=new ModeleClient(); }
%>
```

→ Scriptlet

```
<html> <body>
    <form action="contrroleur" method="post">
        Mot Clé:<input type="text" name="motCle" value='<%=mod.getMotCle() %>'>
        <input type="submit" value="Chercher">
    </form>
    <table border="1" width="80%">
        <tr> <th>ID</th><th>NOM</th><th>EMAIL</th><th>VILLE</th> </tr>
```

→ Expression

```
<%
List<Client> clts=mod.getClients();
for(Client c:clts){ %>
```

→ Scriptlet

```
<tr>
    <td><%=c.getIdClient() %></td> <td> <%=c.getNom() %></td>
    <td><%=c.getEmail() %></td> <td><%=c.getVille() %></td>
</tr>
<% } %>
</table></body></html>
```

→ Expression



JSTL : Java Standard Tag Library

- JSTL est l'acronyme de Java server page Standard Tag Library.
- C'est un ensemble de tags personnalisés développé sous la JSR 052 qui propose des fonctionnalités souvent rencontrées dans les JSP :
 - Tag de structure (itération, conditionnement ...)
 - Internationalisation
 - Exécution de requête SQL
 - Utilisation de document XML

Même exemple JSP en utilisant JSTL

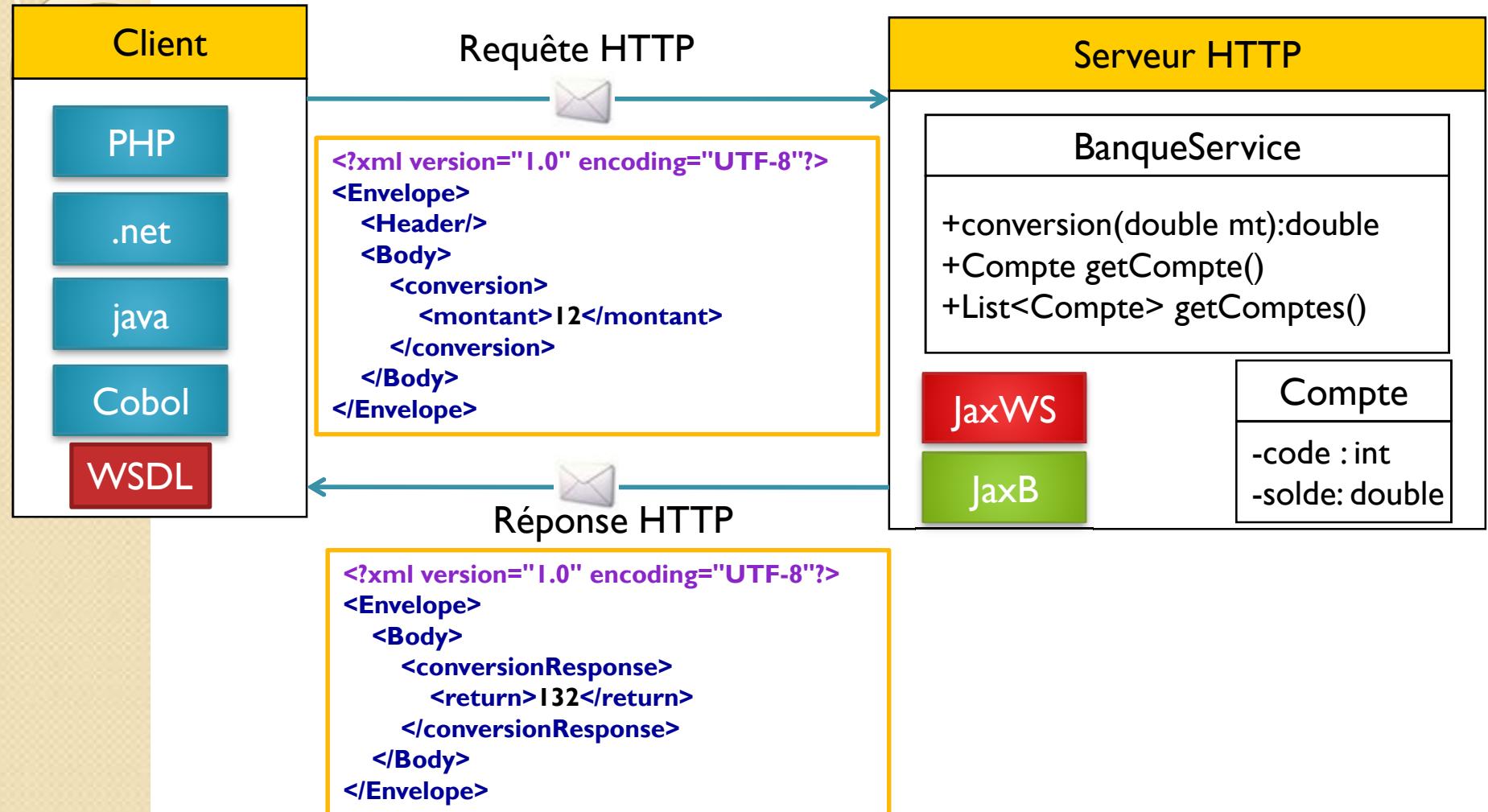
```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="controleur" method="post">
    Mot Clé:<input type="text" name="motCle" value="${modele.motCle}">
    <input type="submit" value="Chercher">
</form>
<table border="1" width="80%">
    <tr>
        <th>ID</th><th>NOM</th><th>EMAIL</th><th>VILLE</th>
    </tr>
    <c:forEach items="${modele.clients}" var="c">
        <tr>
            <td>${c.idClient}</td>
            <td>${c.nom}</td>
            <td>${c.email}</td>
            <td>${c.ville}</td>
        </tr>
    </c:forEach>
</table>
</body>
</html>
```



Web Services avec JAXWS

med@youssf.net

L'idée des Web Services



JAX-WS

- JAX-WS est la nouvelle appellation de JAX-RPC (Java API for XML Based RPC) qui permet de développer très simplement des services web en Java.
- JAX-WS fournit un ensemble d'annotations pour mapper la correspondance Java-WSDL. Il suffit pour cela d'annoter directement les classes Java qui vont représenter le service web.
- Dans l'exemple ci-dessous, une classe Java utilise des annotations JAX-WS qui vont permettre par la suite de générer le document WSDL. Le document WSDL est auto-généré par le serveur d'application au moment du déploiement :

```
@WebService(serviceName="BanqueWS")
public class BanqueService {
    @WebMethod(operationName="ConversionEuroToDh")
    public double conversion(@WebParam(name="montant")double mt){
        return mt*11;
    }
    @WebMethod
    public String test(){    return "Test";    }
    @WebMethod
    public Compte getCompte(){    return new Compte (1,7000);    }
    @WebMethod
    public List<Compte> getComptes(){
        List<Compte> cptes=new ArrayList<Compte>();
        cptes.add (new Compte (1,7000)); cptes.add (new Compte (2,9000));
        return cptes;
    }
}
```



EJB : Enterprise Java Beans

Entreprise Java Beans (EJB)

- Les Entreprise Java Bean ou EJB sont des composants qui permettent de d'implémenter la logique **métier**.
- Ce sont des composant **distribués** qui s'exécutent au sein d'un conteneur EJB qui fait partie du serveur d'application J2EE
- Tous les EJB peuvent évoluer dans un contexte transactionnel.
- Le but des EJB est de faciliter la création d'applications distribuées pour les entreprises.
- Autrement dit, EJB permet de **séparer** le code **métier** qui est propre aux spécifications fonctionnelles du code **technique** (spécification non fonctionnel)



Entreprise Java Beans (EJB)

- Une des principales caractéristiques des EJB est de permettre aux développeurs de se concentrer sur les traitements orientés métiers car les EJB et l'environnement dans lequel ils s'exécutent prennent en charge un certain nombre de traitements techniques en utilisant les services de l'infrastructure offert par le serveur d'application tel que :
 - La distribution
 - La gestion des transactions,
 - La persistance des données,
 - Le cycle de vie des objets
 - La montée en charge
 - La concurrence
 - La sécurité
 - La sérialisation
 - Journalisation
 - Etc....

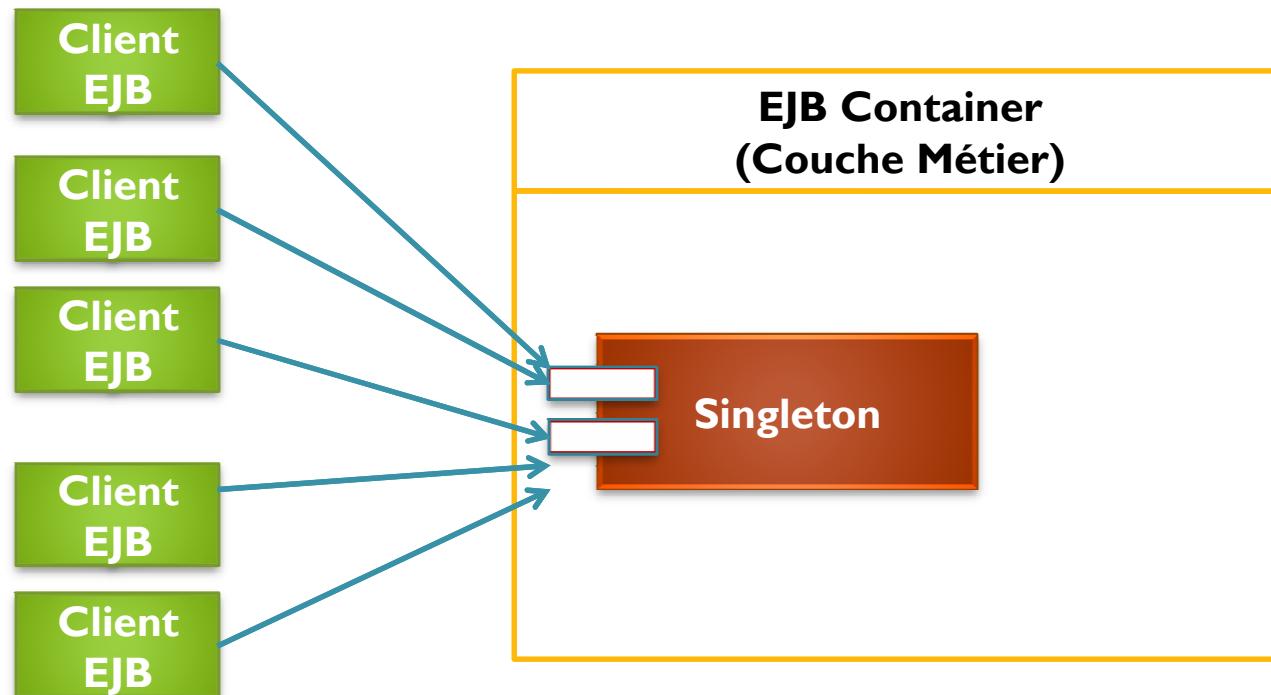


Différents types d'EJB

- Il existe trois types d'EJB :
 - **Entity Beans** : Les EJB entités
 - Représentent les données manipulées par l'application (Composants persistants)
 - Chaque EJB Entity est associé à une table au niveau de la base de données
 - **Session Beans** : Les EJB session
 - Composants distribués qui implémentent les traitement de la logique métier.
 - Ces composants sont accessibles à distance via les protocole RMI et IIOP.
 - Il existe deux types d'EJB Session.
 - **Stateless** : sans état
 - Une instance est créée par le serveur pour plusieurs connexions clientes.
 - Ce type de bean ne conserve aucune donnée dans son état.
 - **Statefull** : avec état
 - Création d'une instance pour chaque connexion cliente.
 - Ce type de bean peut conserver des données entre les échanges avec le client.
 - **Singleton**: Instance Unique
 - Création d'une instance unique quelque soit le nombre de connexion.
 - **Message Driven Beans** : Beans de messages
 - Un listener qui permet de déclencher des traitements au niveau de l'application suite à la réception d'un message asynchrone JMS

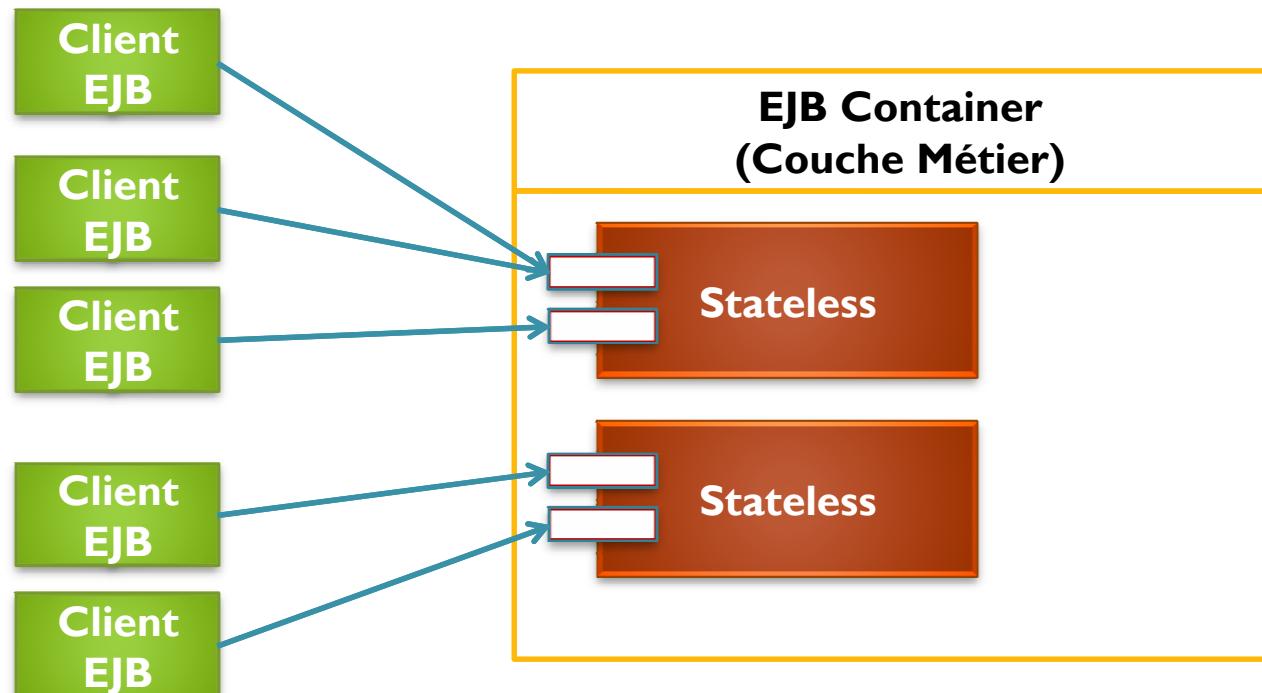
EJB Session Singleton

- Crédation d'une seule instance



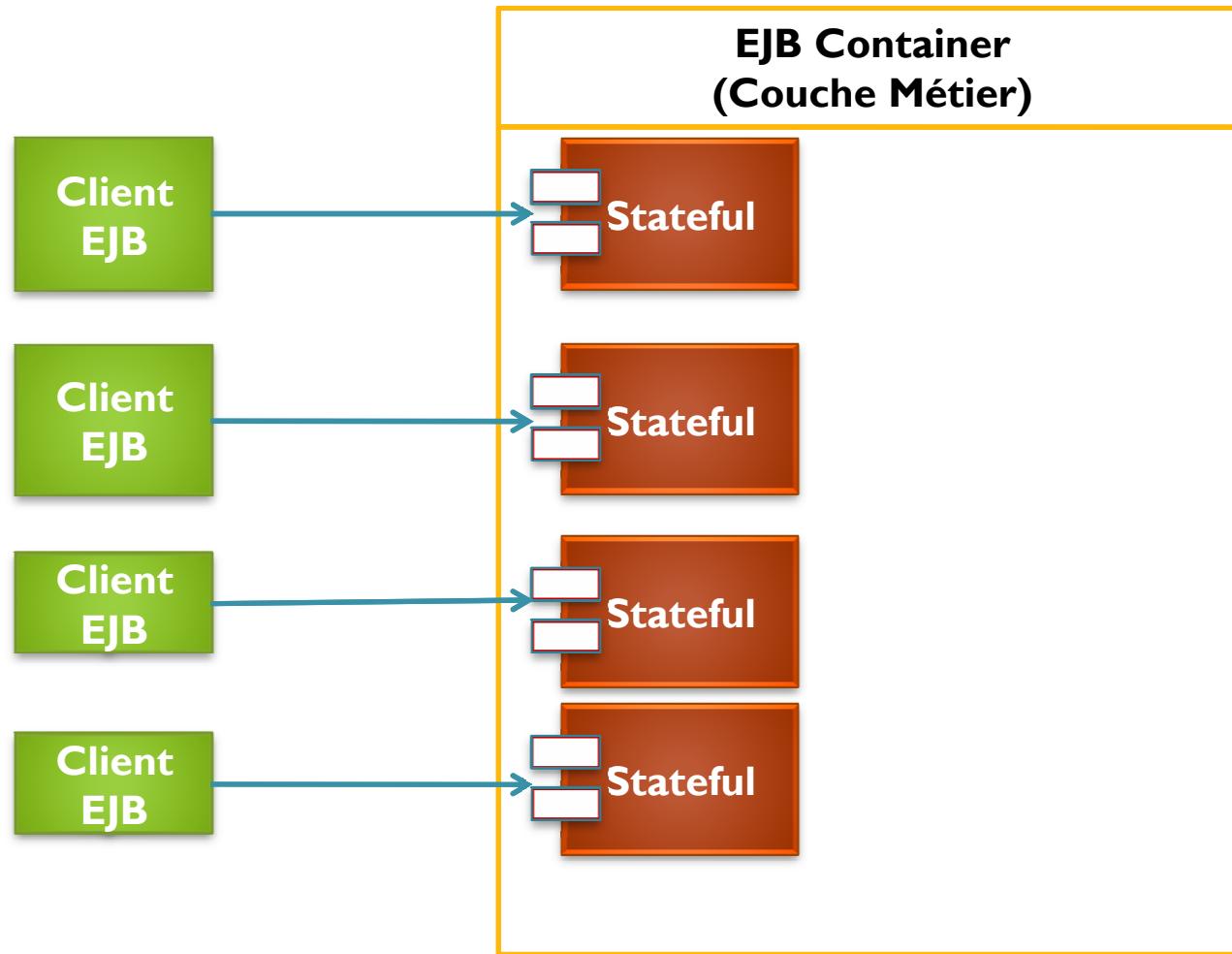
EJB Session Stateless

- Crédation d'un pool d'instances

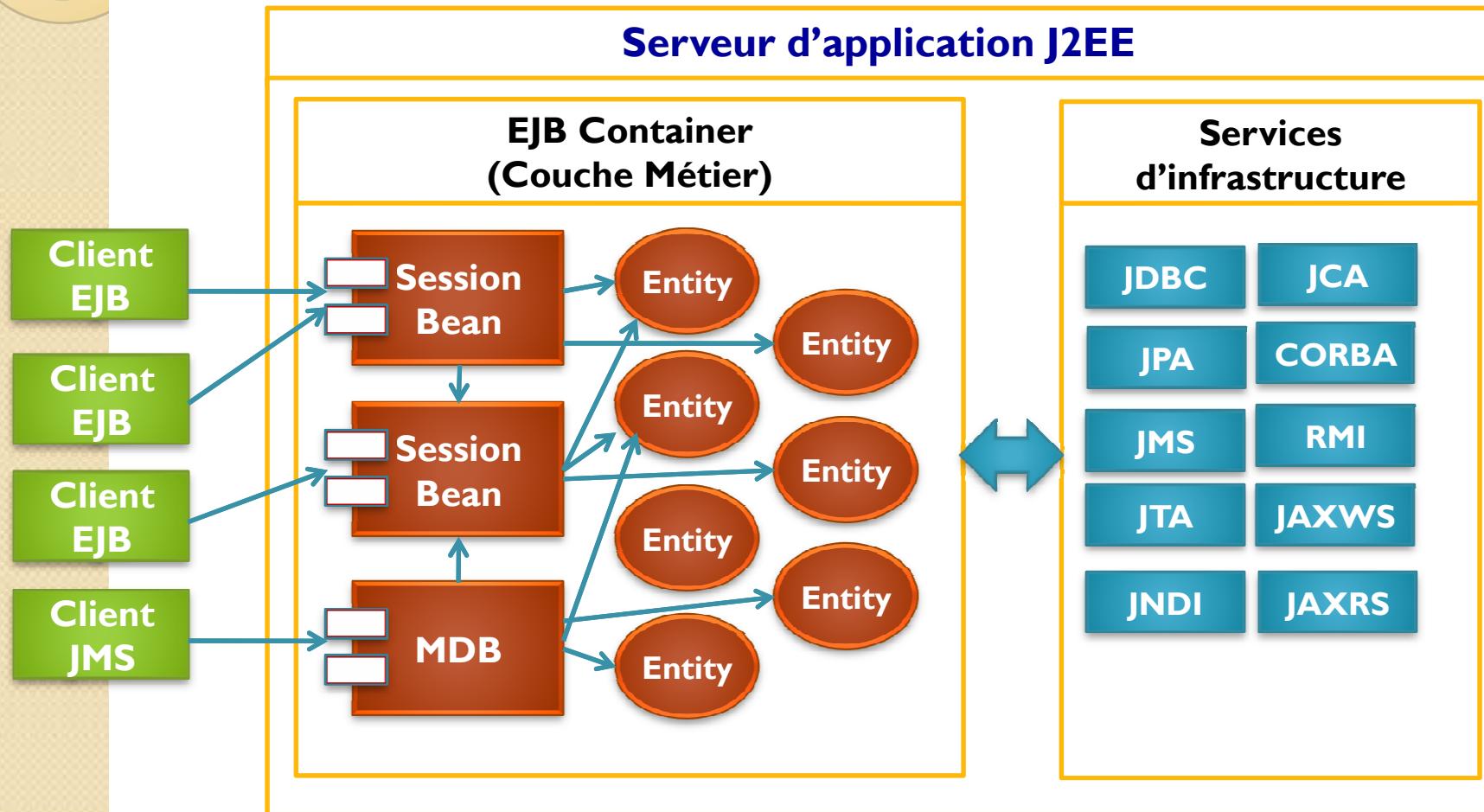


EJB Session Stateful

- Création d'une instance pour chaque connexion

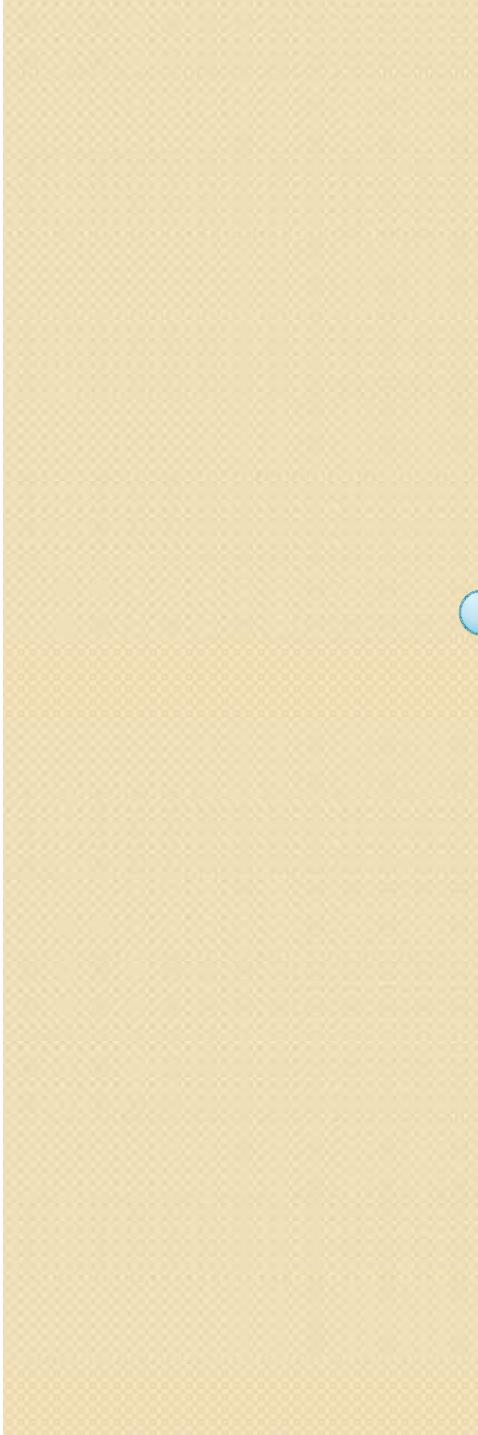


Conteneur EJB et Services d'infrastructures



Exemple d'EJB Entity : Compte.java

```
package metier.entities; import java.io.Serializable;  
import java.util.*; import javax.persistence.*;  
@Entity  
@Table(name="COMPTES")  
public class Compte implements Serializable {  
    @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    @Column(name="CODE")  
    private Long code;  
    @Column(name="SOLDE")  
    private double solde;  
    @Temporal(TemporalType.TIMESTAMP)  
    private Date dateCreation;  
    private boolean active;  
  
    // Getters et Setters  
    // Constructeurs  
}
```



EJB SESSION

med@youssf.net



EJB Session

- Un EJB Session est un composant qui possède
 - Une interface **remote** : qui permet de déclarer les méthodes qui sont accessibles à distance. C'est-à-dire accessible aux composants qui sont déployés dans d'autres machines.
 - Une interface **Local** : qui permet de déclarer les méthodes qui sont accessible en local. C'est-à-dire les méthodes accessible par les composants déployés dans le même serveur d'application.
 - La **classe du bean** qui implémente les deux interfaces remote et local. l'implémentation des méthodes de cette classe représentent les traitements métier de l'application

Interface Remote

- Une interface Remote d'un EJB Session doit être annotée **@Remote**

```
package metier.session;
import java.util.List;
import javax.ejb.Local;
import metier.entities.Compte;
@Remote
public interface IBanqueLocal {
    public void addCompte(Compte c);
    public List<Compte> getAllComptes();
    public Compte getCompte(Long code);
    public void verser(double mt, Long code);
    public void retirer(double mt, Long code);
    public void virement(double mt, Long cpte1, Long cpte2);
    public void updateCompte(Compte c);
    public void supprimerCompte(Long code);
}
```

L'interface Local

- Une interface Locale d'un EJB Session doit être annotée **@Local**

```
package metier.session;
import java.util.List;
import javax.ejb.Local;
import metier.entities.Compte;
@Local
public interface IBanqueLocal {
    public void addCompte(Compte c);
    public List<Compte> getAllComptes();
    public Compte getCompte(Long code);
    public void verser(double mt,Long code);
    public void retirer(double mt,Long code);
    public void virement(double mt,Long cpte1,Long cpte2);
    public void updateCompte(Compte c);
    public void supprimerCompte(Long code);
}
```

Implémentation d'un EJB Session Stateless

- Un EJB Session est une classe qui implémente ses deux interfaces Local et Remote.
- Cette classe doit être annoté par l'une des annotations suivantes:
 - **@Statless** : Un pool d'instances de cet EJB sera créé par le serveur
 - **@Statfull** : Pour chaque connexion, le serveur crée une instance
 - **@Singleton** : Un instance unique sera créée quelque soit le nombre de connexions
- Après instantiation d'un EJB Session, ses références Remote (IP, Port, Adresse Mémoire) et Locale (Adresse Mémoire) seront publiée dans l'annuaire JNDI.
- L'attribut **name** de ces trois annotations, permet de spécifier le nom JNDI qui sera associé aux références de l'EJB dans l'annuaire JNDI
- Par défaut, c'est le nom de la classe qui sera utilisé.
- Ce nom sera combiné à d'autres informations techniques pour garantir l'unicité de ce nom.
- Avec Jboss 7, le nom complet JNDI d'un EJB Session est de la forme suivante :
 - Pour un statless et singleton
 - `Nom_Projet_EAR/Nom_Projet_EJB/Name!Package.NomInterface`
 - Pour un statful
 - `Nom_Projet_EAR/Nom_Projet_EJB/Name!Package.NomInterface?statful`

Exemple d'EJB Session utilisant JPA

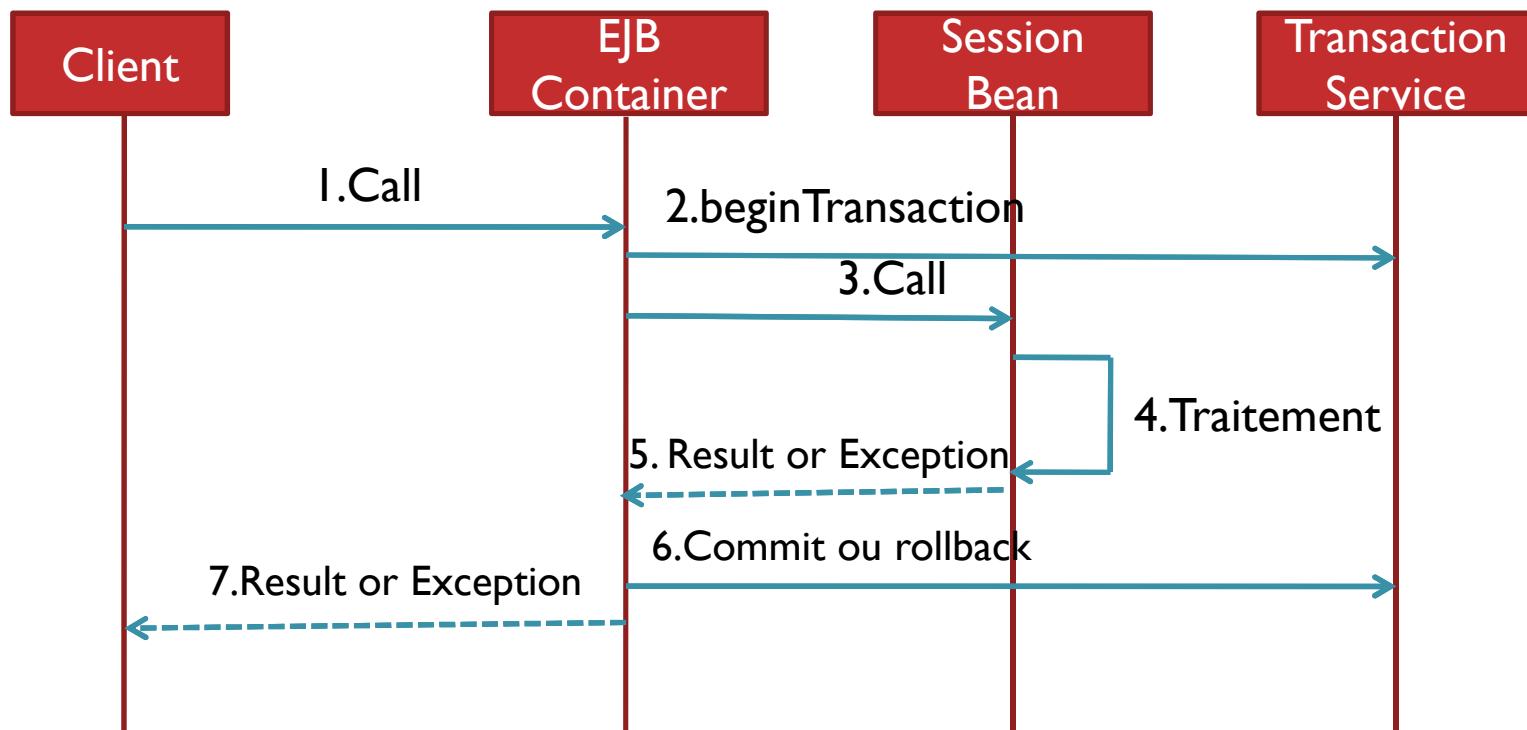
```
package metier.session; import java.util.*;import javax.ejb.*;import  
javax.persistence.*; import metier.entities.Compte;  
  
@Stateless(name="BK")  
  
public class BanqueEJBImpl implements IBanqueLocal,IBanqueRemote {  
    @PersistenceContext(unitName="UP_BANQUE")  
    private EntityManager em;  
  
    @Override  
    public void addCompte(Compte c) {  
        em.persist(c);  
    }  
  
    @Override  
    public List<Compte> getAllComptes() {  
        Query req=em.createQuery("select c from Compte c where c.active=true");  
        return req.getResultList();  
    }  
  
    @Override  
    public Compte getCompte(Long code) {  
        Compte cp=em.find(Compte.class,code);  
        if(cp==null) throw new RuntimeException("Compte introuvable");  
        return cp;  
    }  
}
```

Exemple d'EJB Session utilisant JPA

```
@Override  
public void verser(double mt, Long code) {  
    Compte cp=getCompte(code);cp.setSolde(cp.getSolde()+mt);  
    em.persist(cp);  
}  
  
@Override  
public void retirer(double mt, Long code) {  
    Compte cp=getCompte(code);  cp.setSolde(cp.getSolde()-mt);  
}  
  
@Override  
public void virement(double mt, Long cpte1, Long cpte2) {  
    retirer(mt, cpte1); verser(mt, cpte2);  
}  
  
@Override  
public void updateCompte(Compte c) {  
    em.merge(c);  
}  
  
@Override  
public void supprimerCompte(Long code) {  
    Compte cp=getCompte(code); em.remove(cp);  
}  
}
```

Gestion implicite des transactions

- C'est le mode par défaut
- Le bean est automatiquement enrôlé (*enrolled*) dans une transaction...
- Le container fait le travail...

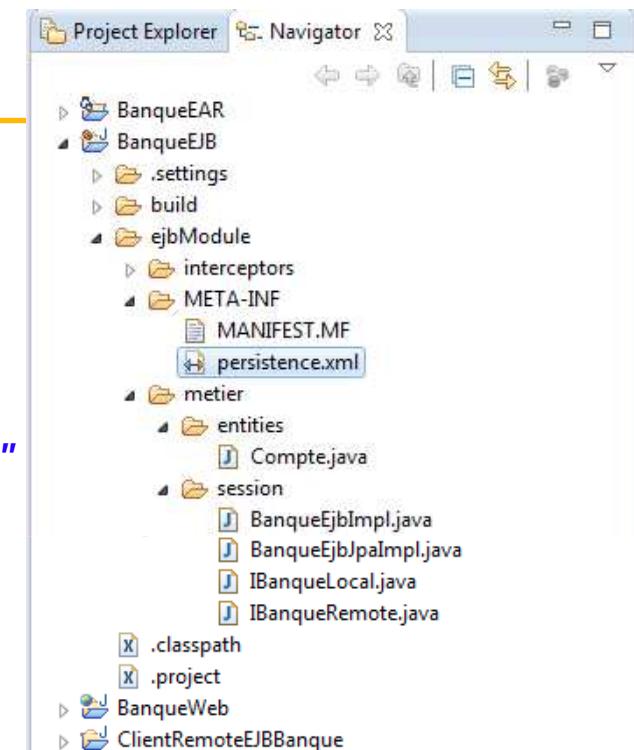


Exemple de Web Service lié à l'EJB Session

```
package metier.services; import java.util.*; import javax.ejb.*; import  
javax.jws.*;  
import metier.entities.Compte; import metier.session.IBanqueLocal;  
@Stateless  
@WebService  
public class BanqueService {  
    @EJB  
    private IBanqueLocal metier;  
    @WebMethod  
    public void addCompte(@WebParam(name="solde")double soldeInitial){  
        Compte cp=new Compte(soldeInitial, new Date(), true);  
        metier.addCompte(cp);  
    }  
    @WebMethod  
    public List<Compte> listComptes(){  
        return metier.getAllComptes();  
    }  
}
```

Configurer l'unité de persistance : persistence.xml

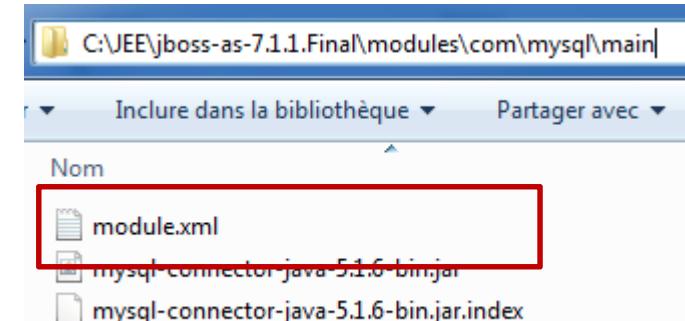
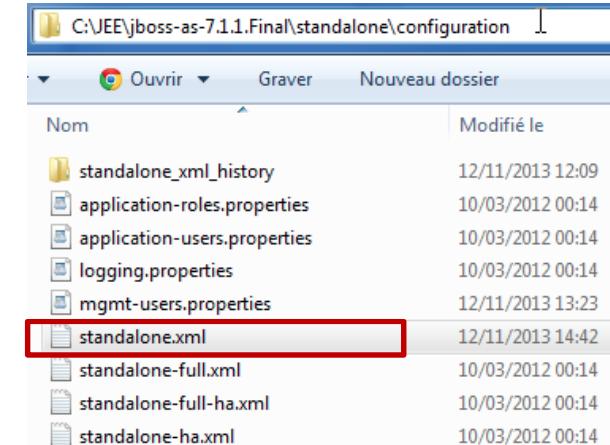
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence
    xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
    version="1.0">
    <persistence-unit name="UP_BP">
        <jta-data-source>java:/dsBanque</jta-data-source>
        <properties>
            <property name="hibernate.hbm2ddl.auto"
value="update"/>
        </properties>
    </persistence-unit>
</persistence>
```



Déployer le datasource pour jboss 7

Standalone.xml

```
<datasources>
    <datasource jndi-name="java:/dsbanque" pool-name="dsBanque" enabled="true">
        <connection-url>jdbc:mysql://localhost:3306/db_banque</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <driver>mysql</driver>
        <security>
            <user-name>root</user-name>
            <password></password>
        </security>
        <validation>
    </datasource>
    <drivers>
        <driver name="h2" module="com.h2database.h2">
            <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
        </driver>
        <driver name="mysql" module="com.mysql"/>
    </drivers>
</datasources>
```





Exemple de Message Driven Bean

- Démarrer un Listener JMS qui
 - Attende la réception d'un message JMS qui arrive dans une file d'attente de type Queue dont le nom est test/queue
 - Une fois le message arrive, on le récupère en supposant que c'est un message de type ObjectMessage.
 - Ce message contient un objet de type compte
 - Cet objet compte est envoyé vers la base de données en faisant appel à l'EJB Session.

Exemple d'EJB MDB

```
package metier.session; import javax.ejb.*; import javax.jms.*;  
import javax.jms.MessageListener; import metier.entities.Compte;  
@MessageDriven( activationConfig={  
    @ActivationConfigProperty(  
        propertyName="destinationType", propertyValue="javax.jms.Topic"),  
    @ActivationConfigProperty(  
        propertyName="destination", propertyValue="topic/test"  
)})  
public class BanqueMDB implements MessageListener {  
    @EJB  
    private IBanqueLocal metier;  
    @Override  
    public void onMessage(Message m) {  
        try {  
            ObjectMessage om=(ObjectMessage) m;  
            Compte cp=(Compte) om.getObject(); metier.addCompte(cp);  
        } catch (Exception e) { e.printStackTrace(); }  
    }  
}
```



MAVEN

med@youssf.net



Maven

- **Maven**, géré par l'organisation *Apache Software Foundation*. (*Jakarta Project*), est un **outil pour la gestion et l'automatisation de production des projets logiciels Java en général et Java EE en particulier**.
- L'objectif recherché est de
 - produire un logiciel à partir de ses sources,
 - en optimisant les tâches réalisées à cette fin
 - et en garantissant le bon ordre de fabrication.
 - **Compiler, Tester, Contrôler, produire les packages livrables**
 - **Publier la documentation et les rapports sur la qualité**
- **Apports :**
 - Simplification du processus de construction d'une application
 - Fournit les bonnes pratiques de développement
 - Tend à uniformiser le processus de construction logiciel
 - Vérifier la qualité du code
 - Faciliter la maintenance d'un projet

Historique

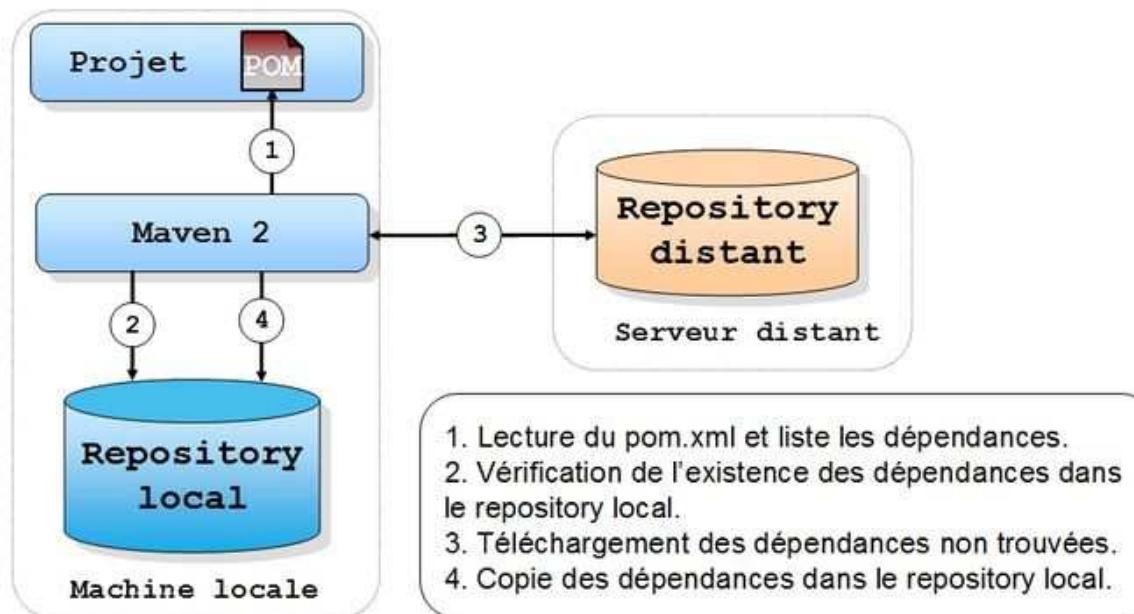


- **Job Control Language** (Langage de Contrôle des Tâches), couramment appelé **JCL**, désigne certains langages de scripts, en particulier sur les systèmes d'exploitation mainframe d'IBM, dont le rôle est d'exécuter un batch.
- **Make** est un logiciel qui construit automatiquement des fichiers, souvent exécutables, ou des bibliothèques à partir d'éléments de base tels que du code source.
- **Ant** est un logiciel créé par la fondation Apache qui vise à automatiser les opérations répétitives du développement de logiciel telles que la compilation, la génération de documents (Javadoc) ou l'archivage au format JAR.
- **Maven** ?

Maven : POM

- Maven utilise un paradigme connu sous le nom de **Project Object Model (POM)** afin de :
 - Décrire un projet logiciel,
 - Ses dépendances avec des modules externes
 - et l'ordre à suivre pour sa production.
- Il est livré avec un grand nombre de tâches (**GOLS**) prédéfinies, comme la compilation du code Java ou encore sa modularisation.

Gestion des dépendances par Maven 2



Remèdes apportés par Maven

Problématique	Réponses de Maven
Gestion des librairies du projet (Versions, Partage, ...)	Dépendances déclaratives Dépendances Transitives Référentiel de librairies
Multiplication des scripts de build	POM Plugins
Standardisation des projets JEE	Standardisation du Build
Travail collaboratif (Multi sites)	Intégration aux différents outils
Mauvaise qualité des livrables	Contrôle et Reporting



Maven : Les concepts

- Descripteurs de Projets
- Cycle de vie et plugins
- Référentiels de laibririe



Descripteurs de Projets

Project Object Model : POM

- Base de travail de Maven :
 - Un projet Maven est un **module d'une application**
 - Equivalent à un projet Eclipse
- Fichier XML (pom.xml) décrivant le projet Maven
 - Versions du projet
 - Description du projet
 - Liste des développeurs
 - Les dépendances
 - ...
- Ce fichier est utilisé par maven pour construire l'application:
 - Dépendances de l'application (Librairies .jar)
 - Tâches (Goals) à exécuter
- Fournie des valeurs par défaut (Bonne pratique):
 - Exemple : Répertoire source (src/main/java)
- Un seul POM est nécessaire pour un projet
 - Le commandes de maven sont à exécuter à la racine du projet : l'emplacement du fichier pom.xml

Le POM minimal

- La racine du projet : **<project>**
- La version du modèle de pom (**<modelVersion>**) : 4.0.0 pour Maven 2.x
- L'identifiant du groupe auquel appartient le projet : **<groupId>**
 - Généralement commun à tous les modules d'un projet
- L'identifiant de l'artifact à construire: **<artifactId>**
 - Généralement le nom du module du projet sans espace en minuscules.
- La version de l'artifact à construire **<version>** : Souvent SNAPSHOT sauf lors de la release
- Le type d'artifact à construire: **<packaging>** : pom, jar, war, ear

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.bp</groupId>
  <artifactId>reclamations</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
</project>
```



Caractéristiques du projet

- Description du projet
 - Informations diverses
- Dépendances du projet:
 - Liste des librairies utilisées
 - Précision du scope des librairies
 - Exclusion des dépendances transitives



Phase de la construction du projet

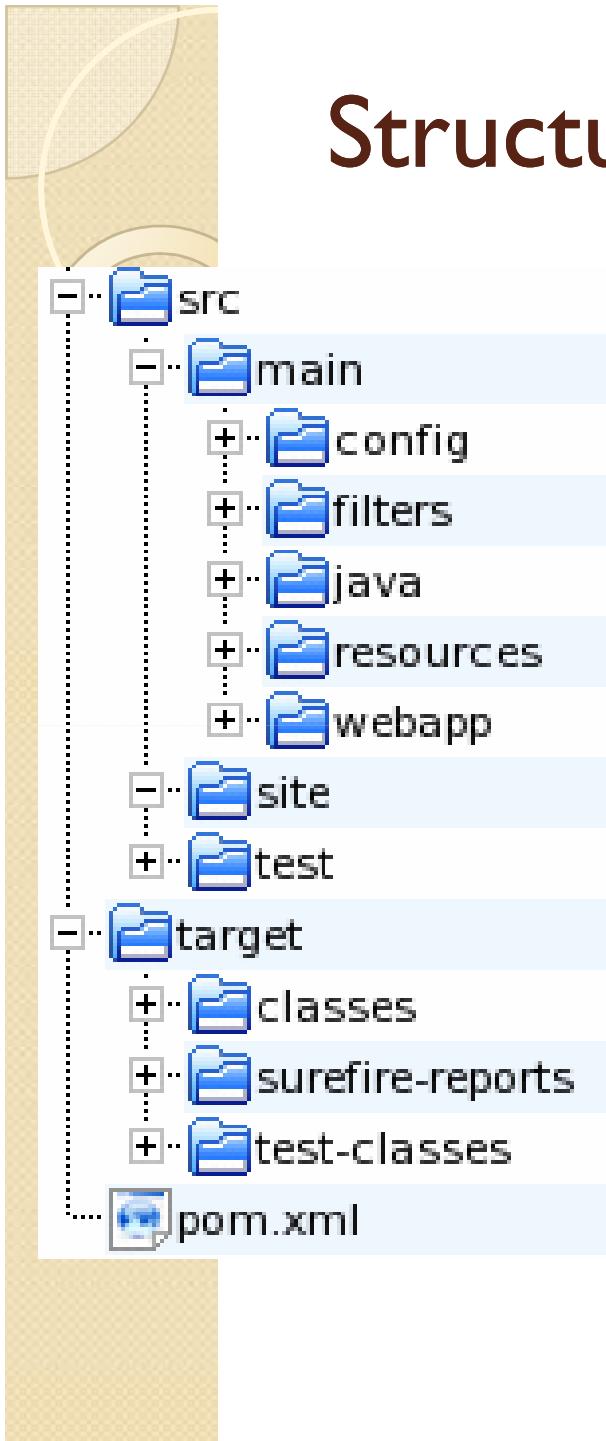
- Phase de la construction :
 - Agencement des répertoires : Structure du projet
 - Tâches (Gols)
 - Gestion des ressources du projet
 - En grande partie configurée par défaut
- Gestion des plugins (Optionnel)
 - Utilisation des plugins existants
 - Tâches personnalisés (Possibilité de créer de nouveau plugin)
- Gestion des rapports (Optionnelle)
 - Créer des rapports à générer
 - Utilisation des plugins dédiés



Organisation des répertoires

- Maven propose une structure de fichier complète. Il s'agit de la configuration par défaut mais elle est surchargeable.
- Le principe général est de limiter le répertoire racine du projet à trois éléments:
 - Le fichier de description du projet **pom.xml** ,
 - Le répertoire **src** qui contient uniquement les sources du projet
 - et le répertoire **target** qui contient tous les éléments créé par Maven.

Structure d'un projet maven



- **src/main/java** :
 - Contient les sources Java de l'application
- **src/main/resources**
 - Contient les ressources de l'application
- **src/main/webapp**
 - Contient les fichiers de l'application Web
- **src/test/java**
 - Contient les sources Java pour les tests unitaires
- **src/test/resources**
 - Contient les ressources pour les tests unitaires
- **src/site**
 - Contient les fichiers pour le site
- **target**
 - Répertoire de destination de tous les traitements Maven

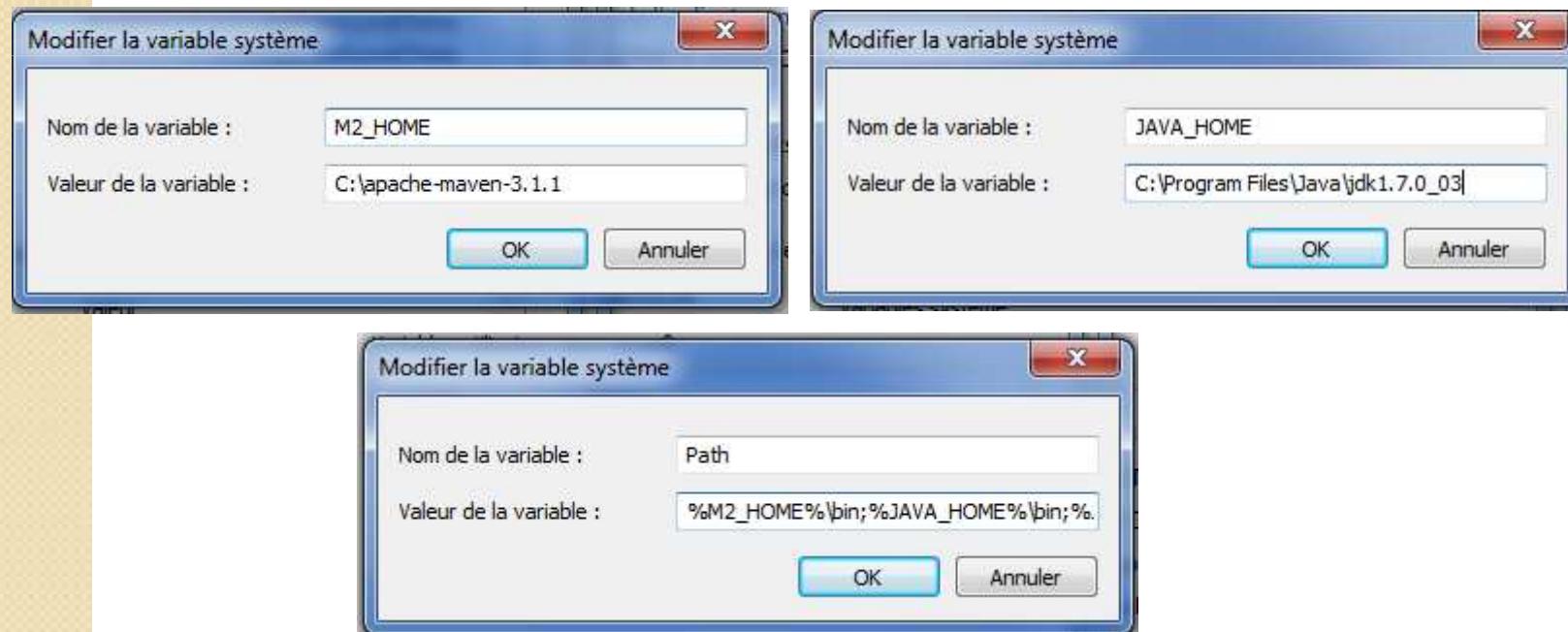


MISE EN ŒUVRE D'UN PROJET MAVEN

med@youssf.net

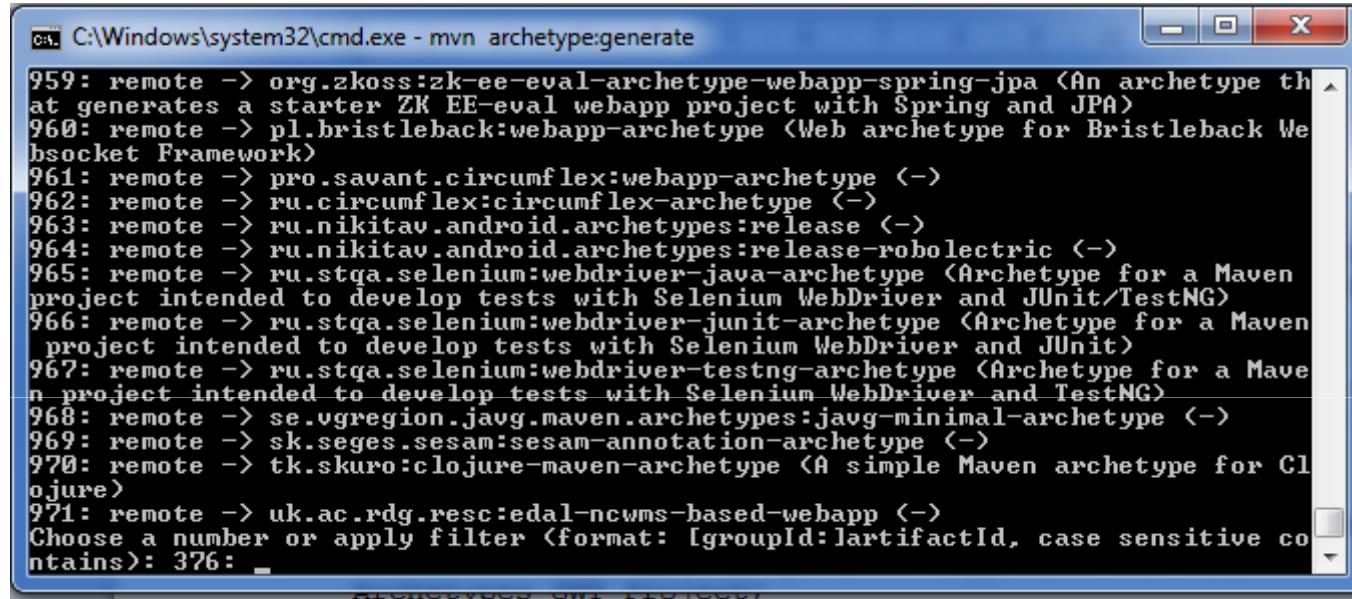
Installation et configuration

- Après avoir installé Maven2
- Définir dans les variable d'environnement :
 - JAVA_HOME= C:\Program Files\Java\jdk1.7.0_03
 - M2_HOME= C:\apache-maven-3.1.1
 - path=%JAVA_HOME%\bin;%M2_HOME%\bin;



Générer la structure d'un projet

- Dans un répertoire vide c :\TP_MVN, lancez la commande :
 - **mvn archetype:generate**
- Vous obtenez un résultat similaire à ceci :



The screenshot shows a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe - mvn archetype:generate'. The output of the command is displayed, listing various Maven archetypes from different groups and artifacts. The text is too long to reproduce here but includes entries like 'org.zkoss:zk-ee-eval-archetype-webapp-spring-jpa', 'pl.bristleback:webapp-archetype', and 'ru.circumflex:circumflex-archetype'. At the bottom of the list, it says 'Choose a number or apply filter <format: [groupId:artifactId, case sensitive contains]: 376: -'.

- Maven vous demande d'**entrer le numéro du type de projet** pour le que vous lui demandez de générer un squelette.
- Afin de vous repérer vous avez besoin de mettre dans un fichier tous les numéros d'archetype.
 - Pour cela faire : **mvn archetype:generate > arch_nums.txt**
 - Puis patientez 10 secondes et puis appuyez 3 fois sur [Crtl]-C
 - Vous pourrez ensuite faire des recherche dans le fichier arch_nums.txt.



Générer la structure d'un projet

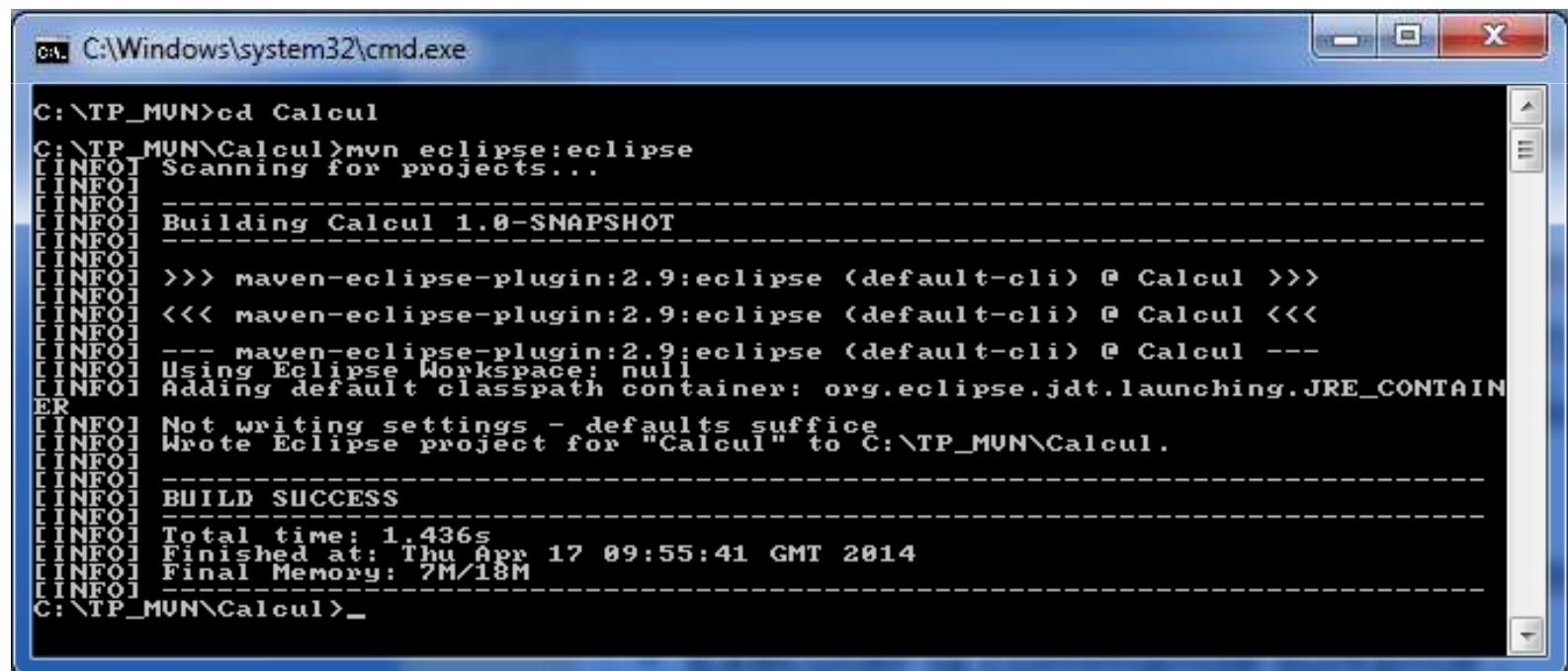
- si vous ne choisissez pas de numéro et que vous tapez ENTREE Maven va créer le type correspondant au projet [maven-archetype-quikstart](#) générant un squelette de projet Maven d'une application java simple. Maven y crée un fichier source Main.java dans src/main/java et un fichier test dans src/test.
- Les autres information à fournir sont :
 - groupId : ma.bp
 - artifactId: Calcul
 - version : par défaut (1.0-SNAPSHOT)
 - package : ma.bp.calcul
- Après confirmer les propriétés :Y

Générer la structure d'un projet

```
C:\Windows\system32\cmd.exe
Choose a number or apply filter <format: [groupId:]artifactId, case sensitive contains>: 376:
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
Choose a number: 6:
Define value for property 'groupId': : ma.bp
Define value for property 'artifactId': : Calcul
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': ma.bp: : ma_bp.calcul
Confirm properties configuration:
groupId: ma.bp
artifactId: Calcul
version: 1.0-SNAPSHOT
package: ma_bp.calcul
Y: :
[INFO] -----
[INFO] Using following parameters for creating project from Old <1.x> Archetype:
maven-archetype-quickstart:1.1
[INFO] -----
[INFO] Parameter: groupId, Value: ma.bp
[INFO] Parameter: packageName, Value: ma_bp.calcul
[INFO] Parameter: package, Value: ma_bp.calcul
[INFO] Parameter: artifactId, Value: Calcul
[INFO] Parameter: basedir, Value: C:\TP_MUN
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old <1.x> Archetype in dir: C:\TP_MUN\Calcul
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 52.015s
[INFO] Finished at: Thu Apr 17 09:53:01 GMT 2014
[INFO] Final Memory: 10M/26M
[INFO] -----
C:\TP_MUN>
```

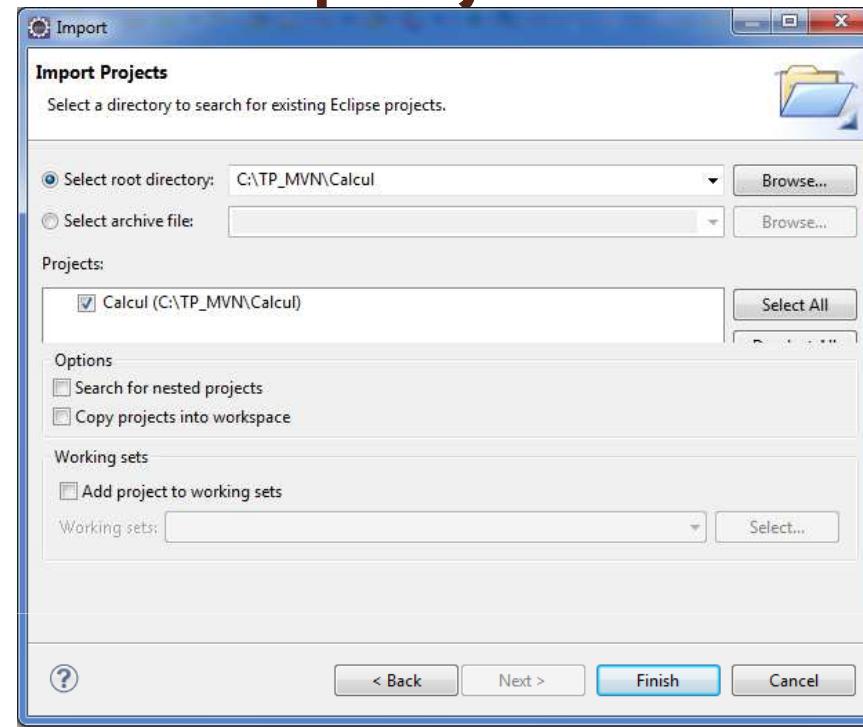
Editer le Projet Généré avec eclipse

- Pour éditer le projet généré avec eclipse, nous avons besoin de demander à maven de générer les fichiers .project et .classpath, nécessaires à un projet eclipse
- Nous utilisons pour cela le plugin eclipse
- Exécuter la commande suivante : [mvn eclipse:eclipse](#)

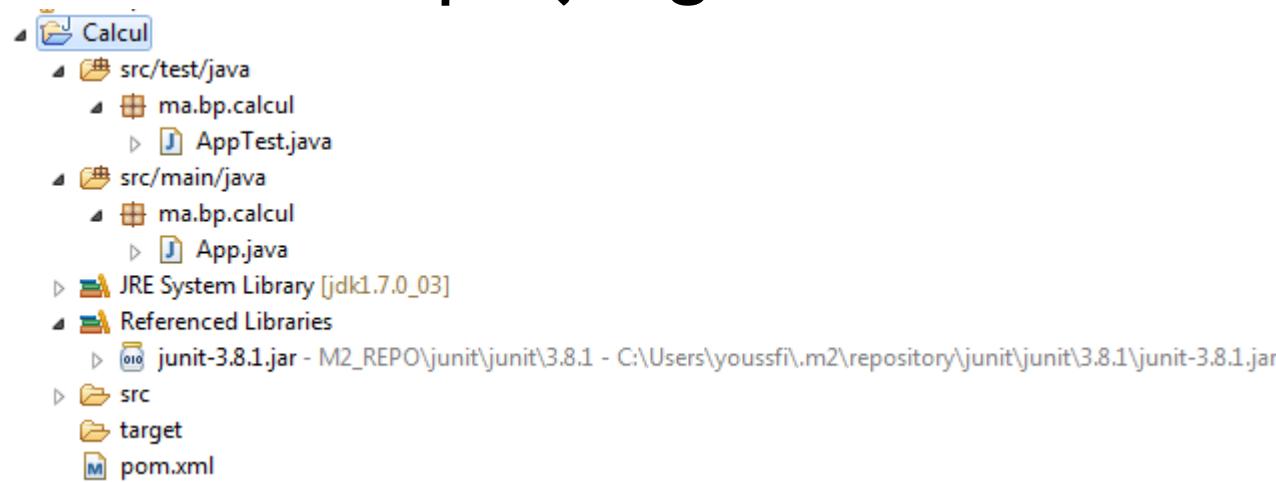


```
C:\Windows\system32\cmd.exe
C:\TP_MUN>cd Calcul
C:\TP_MUN\Calcul>mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] Building Calcul 1.0-SNAPSHOT
[INFO]
[INFO] >>> maven-eclipse-plugin:2.9:eclipse (default-cli) @ Calcul >>>
[INFO] <<< maven-eclipse-plugin:2.9:eclipse (default-cli) @ Calcul <<<
[INFO] --- maven-eclipse-plugin:2.9:eclipse (default-cli) @ Calcul ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Not writing settings - defaults suffice
[INFO] Wrote Eclipse project for "Calcul" to C:\TP_MUN\Calcul.
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 1.436s
[INFO] Finished at: Thu Apr 17 09:55:41 GMT 2014
[INFO] Final Memory: 7M/18M
[INFO]
C:\TP_MUN\Calcul>_
```

Importer le projet avec eclipse



- Structure du projet généré



pom.xml du projet généré

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>ma.bp</groupId>
  <artifactId>Calcul</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Calcul</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```



Création de la classe calcul

- Dans ce projet, nous allons faire quelque chose de très simple :
- Une classe Calcul qui contient deux méthodes :
 - Somme qui permet de retourner la somme de deux nombres:
 - Produit qui permet de retourner le produits de deux nombre
- Un Test unitaire qui permet de tester les deux méthodes
- Nous demanderons ensuite à maven de:
 - compiler toutes les classes
 - Exécuter tous les test unitaires
 - Installer le jar du projet dans le repository local de maven



Code source de la classe CalculMetier

```
package ma.bp.calcul;  
public class CalculMetier {  
    public double somme(double a,double b){  
        return (a+b);  
    }  
    public double produit(double a,double b){  
        return a*b;  
    }  
}
```

Test Unitaire de la classe CalculMetier

```
package ma.bp.calcul;  
import junit.framework.TestCase;  
public class CalculMetierTest extends TestCase {  
    private CalculMetier calcul;  
    protected void setUp() throws Exception {  
        super.setUp();  
        calcul=new CalculMetier();  
    }  
    public void testSomme() {  
        assertTrue(calcul.somme(6, 9)==15);  
    }  
    public void testProduit() {  
        assertTrue(calcul.produit(7, 4)==28);  
    }  
}
```



Gols : Compilation, Test, Installation

- Pour lancer la compilation de toutes les classes du projet , on exécute la commande :
 - **mvn compile**
- Pour lanacer tous les test unitaires du ptojet:
 - **mvn test** ou **mvn test -Dtest=*Test**
- Pour installer le jar du projet :
 - **mvn install**

Compilation des classes : mvn compile

```
C:\Windows\system32\cmd.exe

C:\TP_MUN\Calcul>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Calcul 1.0-SNAPSHOT
[INFO] -----
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Calcul ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\TP_MUN\Calcul\src\main\resources
[INFO] -----
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ Calcul ---
[INFO] Nothing to compile - all classes are up to date
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.997s
[INFO] Finished at: Thu Apr 17 10:29:00 GMT 2014
[INFO] Final Memory: 6M/15M
[INFO] -----
C:\TP_MUN\Calcul>_
```

Exécution des test unitaires : mvn test

```
C:\Windows\system32\cmd.exe
[INFO] [INFO] Scanning for projects...
[INFO] [INFO] Building Calcul 1.0-SNAPSHOT
[INFO] [INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Calcul ---
[INFO] [INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] [INFO] skip non existing resourceDirectory C:\TP_MUN\Calcul\src\main\resources
[INFO] [INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ Calcul ---
[INFO] [INFO] Nothing to compile - all classes are up to date
[INFO] [INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ Calcul ---
[INFO] [INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] [INFO] skip non existing resourceDirectory C:\TP_MUN\Calcul\src\test\resources
[INFO] [INFO] --- maven-compiler-plugin:2.5.1:testCompile (default-testCompile) @ Calcul ---
[INFO] [INFO] Nothing to compile - all classes are up to date
[INFO] [INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ Calcul ---
[INFO] [INFO] Surefire report directory: C:\TP_MUN\Calcul\target\surefire-reports

-----
T E S T S
-----
Running ma.bp.calcul.CalculMetierTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.012 sec
Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

[INFO] [INFO] BUILD SUCCESS
[INFO] [INFO] Total time: 2.175s
[INFO] [INFO] Finished at: Thu Apr 17 10:29:24 GMT 2014
[INFO] [INFO] Final Memory: 6M/16M
[INFO]
C:\TP_MUN\Calcul>
```

Installation du projet : mvn install

The screenshot shows a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The command 'mvn install' is being run in the directory 'C:\TP_MUN\Calcul'. The output of the command is displayed in the terminal window, showing Maven's internal log messages and the results of the build process.

```
C:\TP_MUN\Calcul>mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] Building Calcul 1.0-SNAPSHOT
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Calcul ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resource files
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Calcul ---
[INFO] Nothing to compile - all classes are up to date since previous compile or no classes defined
[INFO] --- maven-resources-plugin:2.6:resources (processResources) @ Calcul ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resource files
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Calcul ---
[INFO] Nothing to compile - all classes are up to date since previous compile or no classes defined
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ Calcul ---
[INFO] Surefire report directory: C:\TP_MUN\Calcul\target\surefire-reports

TESTS
-----
Running ma.bp.calcul.CalculMetierTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.01 sec
Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ Calcul ---
[INFO] Building jar: C:\TP_MUN\Calcul\target\Calcul-1.0-SNAPSHOT.jar
[INFO] --- maven-install-plugin:2.4:install (default-install) @ Calcul ---
[INFO] Installing C:\TP_MUN\Calcul\target\Calcul-1.0-SNAPSHOT.jar to C:\Users\youssfi\.m2\repository\ma\bp\Calcul\1.0-SNAPSHOT\Calcul-1.0-SNAPSHOT.jar
[INFO] Installing C:\TP_MUN\Calcul\pom.xml to C:\Users\youssfi\.m2\repository\ma\bp\Calcul\1.0-SNAPSHOT\Calcul-1.0-SNAPSHOT.pom
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 2.070s
[INFO] Finished at: Thu Apr 17 10:29:55 GMT 2014
[INFO] Final Memory: 6M/18M
C:\TP_MUN\Calcul>
```

A file explorer window is overlaid on the command prompt, showing the contents of the directory 'C:\Users\youssfi\.m2\repository\ma\bp\Calcul\1.0-SNAPSHOT'. It lists four files: '_remote.repositories' (Fichier REPOSITORY, 1 Ko), 'Calcul-1.0-SNAPSHOT.jar' (Executable Jar File, 3 Ko), 'Calcul-1.0-SNAPSHOT.pom' (Fichier POM, 1 Ko), and 'maven-metadata-local.xml' (Fichier XML, 1 Ko).



Utilisation du jar généré dans un autre projet web

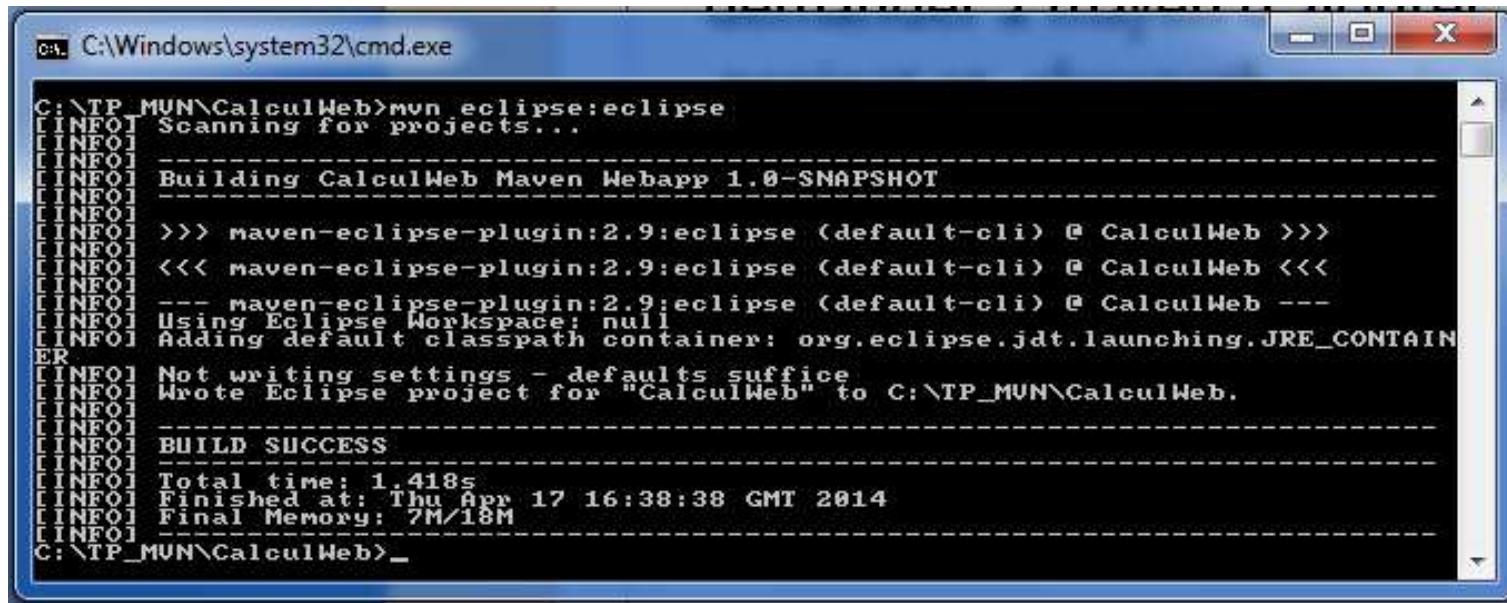
- Nous allons à nouveau générer un nouveau projet maven cette fois de type webapp.
- Dans le dossier TP_MVN, exécuter la commande :
 - mvn archetype:generate
- Cette fois ci, nous allons choisir le numéro 379 correspondant au modèle
org.apache.maven.archetypes:maven-archetype-webapp
- Les autres information à fournir:
 - La version du modèle :Valeur par défaut
 - groupId : ma.bp
 - artifactId : CalculWeb
 - Version : par défaut
 - package : ma.bp.web

Nouveau Projet Web Maven

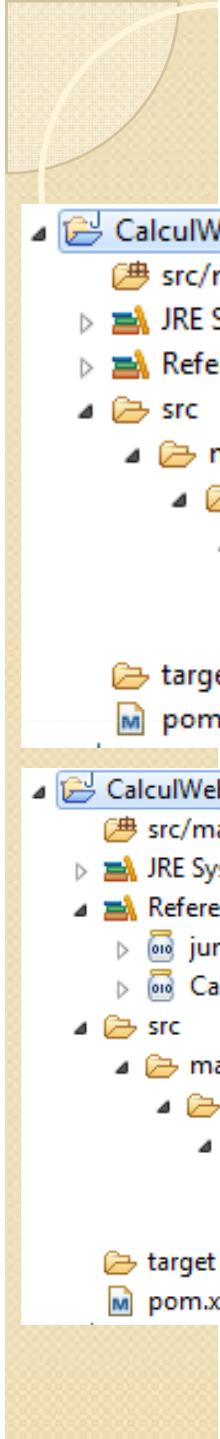
```
C:\Windows\system32\cmd.exe
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 376: 379
Choose org.apache.maven.archetypes:maven-archetype-webapp version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
Choose a number: 5:
Define value for property 'groupId': ma.bp
Define value for property 'artifactId': CalculWeb
Define value for property 'version': 1.0-SNAPSHOT
Define value for property 'package': ma.bp: ma.bp.web
Confirm properties configuration:
groupId: ma.bp
artifactId: CalculWeb
version: 1.0-SNAPSHOT
package: Ma.bp.web
Y:
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-webapp:1.0
[INFO] -----
[INFO] Parameter: groupId, Value: ma.bp
[INFO] Parameter: packageName, Value: Ma.bp.web
[INFO] Parameter: package, Value: ma.bp.web
[INFO] Parameter: artifactId, Value: CalculWeb
[INFO] Parameter: basedir, Value: C:\TP_MUN
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\TP_MUN\CalculWeb
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4:51.086s
[INFO] Finished at: Thu Apr 17 10:47:05 GMT 2014
[INFO] Final Memory: 10M/29M
[INFO] -----
C:\TP_MUN>
```

Edition du projet avec eclipse

- A nouveau, nous aurons besoin de demander à mayen d'ajouter les fichiers .project et .classpath requis par eclipse
- Exécuter à nouveau la commande :
 - mvn eclipse:eclipse



```
C:\TP_MVN\CalculWeb>mvn eclipse:eclipse
[INFO] Scanning for projects...
[INFO]
[INFO] Building CalculWeb Maven Webapp 1.0-SNAPSHOT
[INFO] -----
[INFO] >>> maven-eclipse-plugin:2.9:eclipse (default-cli) @ CalculWeb >>>
[INFO] <<< maven-eclipse-plugin:2.9:eclipse (default-cli) @ CalculWeb <<<
[INFO] --- maven-eclipse-plugin:2.9:eclipse (default-cli) @ CalculWeb ---
[INFO] Using Eclipse Workspace: null
[INFO] Adding default classpath container: org.eclipse.jdt.launching.JRE_CONTAINER
[INFO] Not writing settings - defaults suffice
[INFO] Wrote Eclipse project for "CalculWeb" to C:\TP_MVN\CalculWeb.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.418s
[INFO] Finished at: Thu Apr 17 16:38:38 GMT 2014
[INFO] Final Memory: 7M/18M
[INFO]
C:\TP_MVN\CalculWeb>_
```



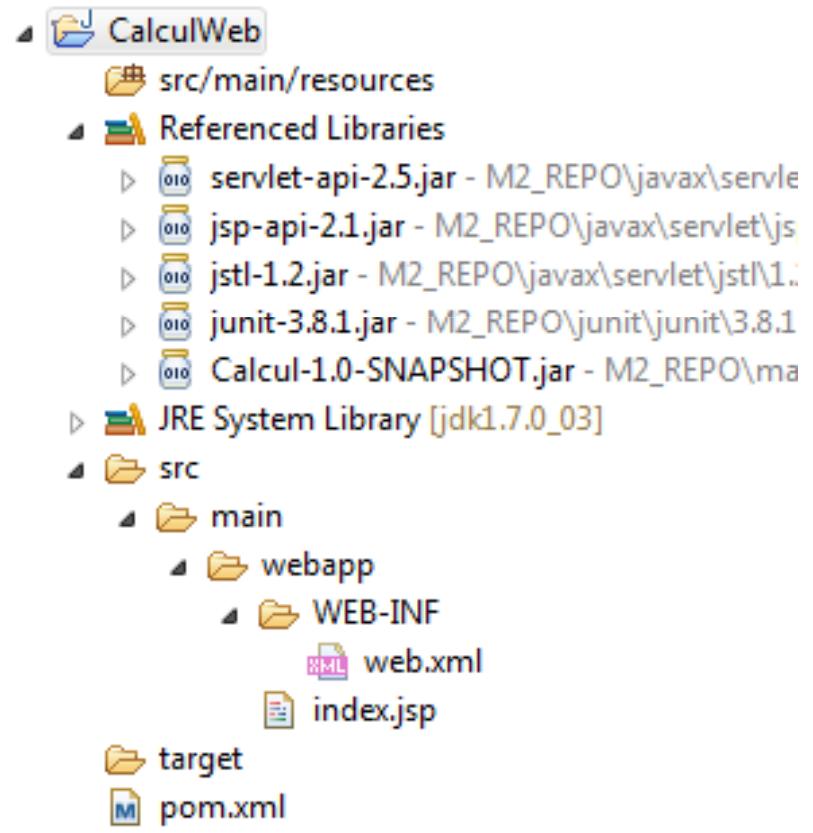
Structure du projet web généré

- Dans ce projet nous aurons besoin du jar du projet précédent.
- Il faut donc ajouter sa dépendance dans pom.xml
- Pour mettre à jour le classpath ecclipse, nous avons besoin de réuxécuter la commande :
 - mvn eclipse:eclipse
- Ensuite actualiser le projet

```
<dependency>
    <groupId>ma.bp</groupId>
    <artifactId>Calcul</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

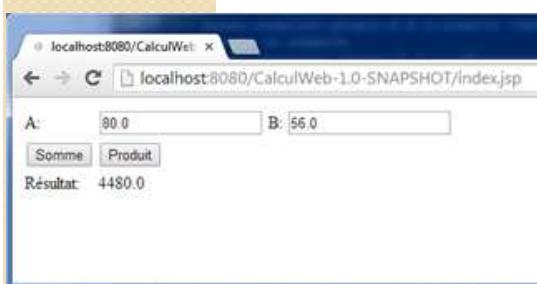
Dépendances JSP, Servlet, JSTL

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId> servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId> jsp-api</artifactId>
    <version>2.1</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId> jstl</artifactId>
    <version>1.2</version>
    <scope>compile</scope>
</dependency>
```



Page JSP : index.jsp

- Maintenant , nous allons créer une simple page JSP qui permet de
- saisir un deux nombre a et b
- et d'afficher la somme ou le produit de ces deux nombres.



```
<%@page import="ma.bp.calcul.CalculMetier"%>
<%
double a=0; double b=0; double res=0;
String action=request.getParameter("action");
if (action!=null){
    a=Double.parseDouble(request.getParameter("a"));
    b=Double.parseDouble(request.getParameter("b"));
    CalculMetier metier=new CalculMetier();
    if(action.equals("Somme")){
        res=metier.somme(a, b);
    }
    else{
        res=metier.produit(a, b);
    }
}
%>
<html>
<body>
    <form action="index.jsp" method="post">
        <table>
            <tr>
                <td>A:</td> <td><input type="text" name="a" value="<%a%>" /></td>
                <td>B:</td> <td><input type="text" name="b" value="<%b%>" /></td>
            </tr>
            <tr>
                <td><input type="submit" value="Somme" name="action"/></td>
                <td><input type="submit" value="Produit" name="action"/></td>
            </tr>
            <tr>
                <td>Résultat:</td> <td><%res%></td>
            </tr>
        </table>
    </form>
</body>
</html>
```

Génération du war : mvn install

- Pour générer l'application web,
- Executer la commande : mvn install

The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe" with the following Maven command and its output:

```
C:\TP_MUN\CalculWeb>mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] Building CalculWeb Maven Webapp 1.0-SNAPSHOT
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ CalculWeb
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ CalculWeb ---
[INFO] No sources to compile
[INFO] --- maven-resources-plugin:2.6:testResources
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\TP_MUN\CalculWeb\src\test\resources
[INFO] --- maven-compiler-plugin:2.5.1:testCompile
[INFO] No sources to compile
[INFO] --- maven-surefire-plugin:2.12.4:test
[INFO] No tests to run.
[INFO] --- maven-war-plugin:2.2:war (default-war)
[INFO] Packaging webapp [CalculWeb] in [C:\TP_MUN\CalculWeb\target\CalculWeb]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\TP_MUN\CalculWeb\src\main\webapp]
[INFO] Webapp assembled in [71 msecs]
[INFO] Building war: C:\TP_MUN\CalculWeb\target\CalculWeb.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO] --- maven-install-plugin:2.4:install (default-install) @ CalculWeb ---
[INFO] Installing C:\TP_MUN\CalculWeb\target\CalculWeb.war to C:\Users\youssfi\.m2\repository\ma\bp\CalculWeb\1.0-SNAPSHOT\CalculWeb-1.0-SNAPSHOT.war
[INFO] Installing C:\TP_MUN\CalculWeb\pom.xml to C:\Users\youssfi\.m2\repository\ma\bp\CalculWeb\1.0-SNAPSHOT\CalculWeb-1.0-SNAPSHOT.pom
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 5.349s
[INFO] Finished at: Fri Apr 18 08:18:25 GMT 2014
[INFO] Final Memory: 6M/17M
[INFO]
C:\TP_MUN\CalculWeb>
```

Below the command prompt, a file explorer window is open, showing the contents of the ".m2\repository\ma\bp\CalculWeb\1.0-SNAPSHOT" directory. The contents are:

Nom	Modifié le	Type	Taille
_remote.repositories	18/04/2014 08:18	Fichier RE...	1 Ko
CalculWeb-1.0-SNAPSHOT.pom	18/04/2014 07:59	Fichier POM	2 Ko
CalculWeb-1.0-SNAPSHOT.war	18/04/2014 08:18	Fichier WAR	364 Ko
maven-metadata-local.xml	18/04/2014 08:18	Fichier XML	1 Ko

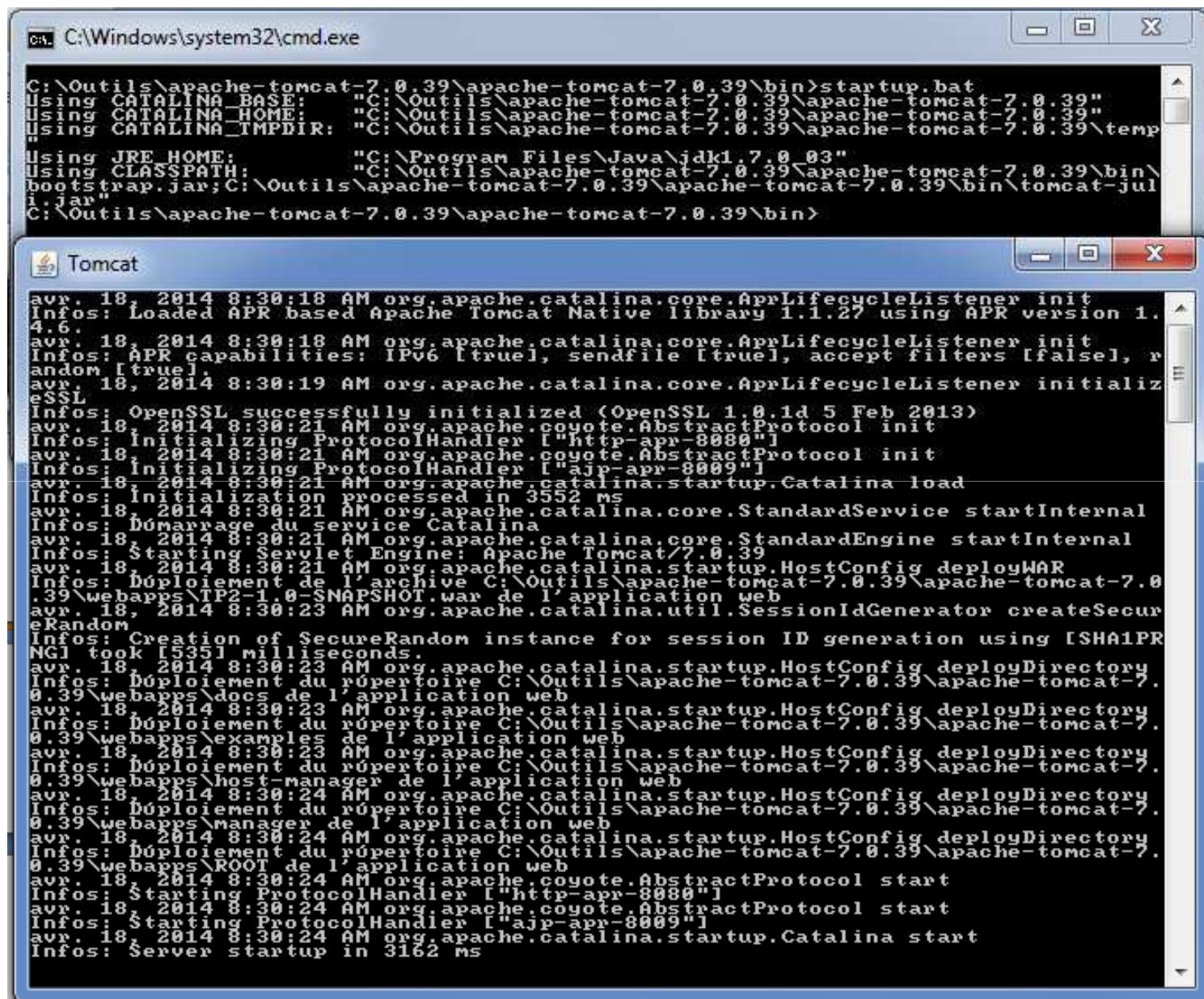
Déployer et tester le projet web

- Pour démarrer tomcat 7 sur ligne de commande , il faut s'assurer que les variables d'environnement JAVA_HOME est définie :



- Ensuite lancer tomcat en exécutant le script startup.bat qui se trouve dans le dossier bin de tomcat

Démarrage de tomcat



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe" and a separate window titled "Tomcat". The cmd window displays the output of the "startup.bat" script, which configures Tomcat with various environment variables and paths. The Tomcat window displays the detailed log output from the "catalina.bat" startup script, showing the initialization of APR, the StandardService, and the StandardEngine, followed by the deployment of the "TP2-1.0-SNAPSHOT.war" application, and finally the successful startup of the server.

```
C:\> C:\Windows\system32\cmd.exe
C:\> C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\bin>startup.bat
Using CATALINA_BASE: "C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39"
Using CATALINA_HOME: "C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39"
Using CATALINA_TMPDIR: "C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\temp"
Using JRE_HOME: "C:\Program Files\Java\jdk1.7.0_03"
Using CLASSPATH: "C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\bin\bootstrap.jar;C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\bin\tomcat-juli.jar"
C:\> C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\bin>

Tomcat
avr. 18, 2014 8:30:18 AM org.apache.catalina.core.AprLifecycleListener init
Infos: Loaded APR based Apache Tomcat Native library 1.1.27 using APR version 1.4.6.
avr. 18, 2014 8:30:18 AM org.apache.catalina.core.AprLifecycleListener init
Infos: APR capabilities: IPv6 [true], sendfile [true], accept filters [false], random [true]
avr. 18, 2014 8:30:19 AM org.apache.catalina.core.AprLifecycleListener initializeSSL
Infos: OpenSSL successfully initialized (OpenSSL 1.0.1d 5 Feb 2013)
avr. 18, 2014 8:30:21 AM org.apache.coyote.AbstractProtocol init
Infos: Initializing ProtocolHandler ["http-apr-8080"]
avr. 18, 2014 8:30:21 AM org.apache.coyote.AbstractProtocol init
Infos: Initializing ProtocolHandler ["ajp-apr-8009"]
avr. 18, 2014 8:30:21 AM org.apache.catalina.startup.Catalina load
Infos: Initialization processed in 3552 ms
avr. 18, 2014 8:30:21 AM org.apache.catalina.core.StandardService startInternal
Infos: Démarrage du service Catalina
avr. 18, 2014 8:30:21 AM org.apache.catalina.core.StandardEngine startInternal
Infos: Starting Servlet Engine: Apache Tomcat/7.0.39
avr. 18, 2014 8:30:21 AM org.apache.catalina.startup.HostConfig deployWAR
Infos: Déploiement de l'archive C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\webapps\TP2-1.0-SNAPSHOT.war de l'application web
avr. 18, 2014 8:30:23 AM org.apache.catalina.util.SessionIdGenerator createSecureRandom
Infos: Creation of SecureRandom instance for session ID generation using [SHA1PRNG] took [535] milliseconds.
avr. 18, 2014 8:30:23 AM org.apache.catalina.startup.HostConfig deployDirectory
Infos: Déploiement du répertoire C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\webapps\docs de l'application web
avr. 18, 2014 8:30:23 AM org.apache.catalina.startup.HostConfig deployDirectory
Infos: Déploiement du répertoire C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\webapps\examples de l'application web
avr. 18, 2014 8:30:23 AM org.apache.catalina.startup.HostConfig deployDirectory
Infos: Déploiement du répertoire C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\webapps\host-manager de l'application web
avr. 18, 2014 8:30:24 AM org.apache.catalina.startup.HostConfig deployDirectory
Infos: Déploiement du répertoire C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\webapps\manager de l'application web
avr. 18, 2014 8:30:24 AM org.apache.catalina.startup.HostConfig deployDirectory
Infos: Déploiement du répertoire C:\Outils\apache-tomcat-7.0.39\apache-tomcat-7.0.39\webapps\ROOT de l'application web
avr. 18, 2014 8:30:24 AM org.apache.coyote.AbstractProtocol start
Infos: Starting ProtocolHandler ["http-apr-8080"]
avr. 18, 2014 8:30:24 AM org.apache.coyote.AbstractProtocol start
Infos: Starting ProtocolHandler ["ajp-apr-8009"]
avr. 18, 2014 8:30:24 AM org.apache.catalina.startup.Catalina start
Infos: Server startup in 3162 ms
```

Interface d'administration de tomcat

The screenshot shows the Apache Tomcat Manager interface. At the top, there's a banner for "The Apache Software Foundation" and a cartoon cat logo. Below it, the title "Gestionnaire d'applications WEB Tomcat" is displayed. A message box shows "Message: OK". A navigation bar includes "Gestionnaire", "Lister les applications", "Aide HTML Gestionnaire", "Aide Gestionnaire", and "Etat du serveur".

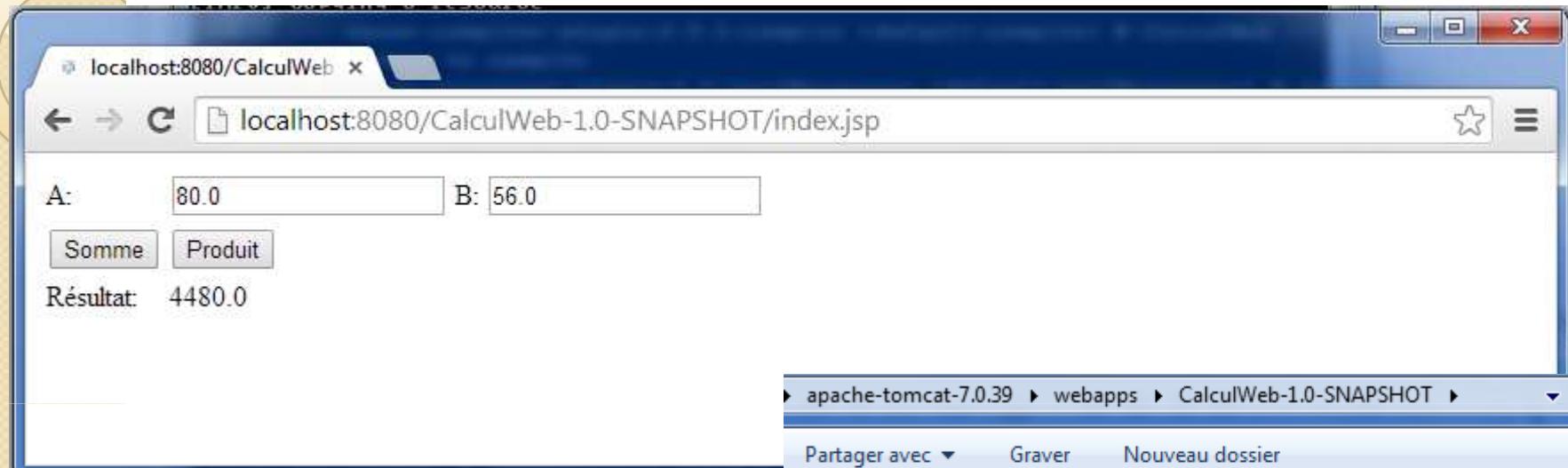
Fichier WAR à déployer

Choisir le fichier WAR à téléverser CalculW...HOT.war

Applications

Chemin	Version	Nom d'affichage	Fonctionnelle	Sessions	Commandes
/	None specified	Welcome to Tomcat	true	0	<input type="button" value="Démarrer"/> <input type="button" value="Arrêter"/> <input type="button" value="Recharger"/> <input type="button" value="Retirer"/> <input type="button" value="Expirer les sessions"/> inactives depuis ≥ 30 minutes
/CalculWeb-1.0-SNAPSHOT	None specified	Archetype Created Web Application	true	0	<input type="button" value="Démarrer"/> <input type="button" value="Arrêter"/> <input type="button" value="Recharger"/> <input type="button" value="Retirer"/> <input type="button" value="Expirer les sessions"/> inactives depuis ≥ 30 minutes

Tester la page JSP



- Structure du war :

Inclure dans la bibliothèque	Partager avec	Graver	Nouveau dossier
Nom	Modifié le	Type	
Calculus-1.0-SNAPSHOT.jar	17/04/2014 10:29	Executable Jar File	
jstl-1.2.jar	01/12/2013 15:16	Executable Jar File	



Droits d'administration de tomcat

- Pour accéder à l'interface d'administration, il faut s'assurer que vous avez défini dans le fichier tomcat/conf/tomcat-users.txt
 - Le rôle **manager-gui**
 - un utilisateur tomcat ayant ce rôle
- Fichier tomcat-users.txt
 - **<role rolename="manager-gui"/>**
 - **<role rolename="admin-gui"/>**
 - **<role rolename="manager-script"/>**
 - **<user username="admin" password="admin" roles="manager-gui, admin-gui, manager-script"/>**

Déploiement avec Maven

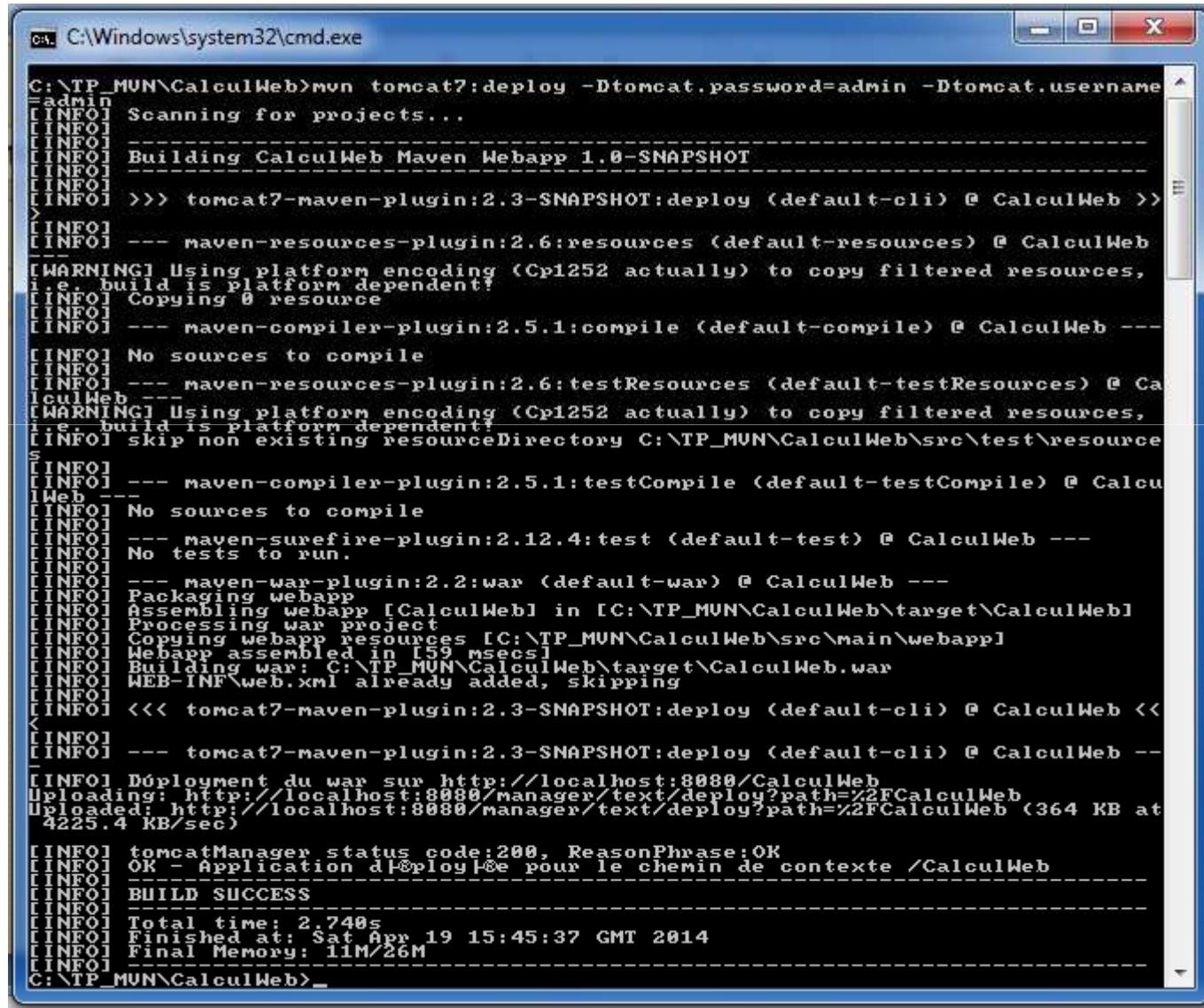
- Pour déployer une application web dans le serveur tomcat en utilisant maven, nous aurons besoin d'utiliser le plugin maven tomcat7.
- Déclaration du plugin dans pom.xml :

```
<build>
    <finalName>CalculWeb</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.3-SNAPSHOT</version>
            <configuration>
                <url>http://localhost:8080/manager/text</url>
            </configuration>
        </plugin>
    </plugins>
</build>

<pluginRepositories>
    <pluginRepository>
        <id>apache.snapshots</id>
        <name>Apache Snapshots</name>
        <url>http://repository.apache.org/content/groups/snapshots-group/</url>
        <releases>
            <enabled>false</enabled>
        </releases>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </pluginRepository>
</pluginRepositories>
```

Commande de déployer le war :

C:\TP_MVN\CalculWeb>mvn tomcat7:deploy -Dtomcat.password=admin -Dtomcat.username=admin



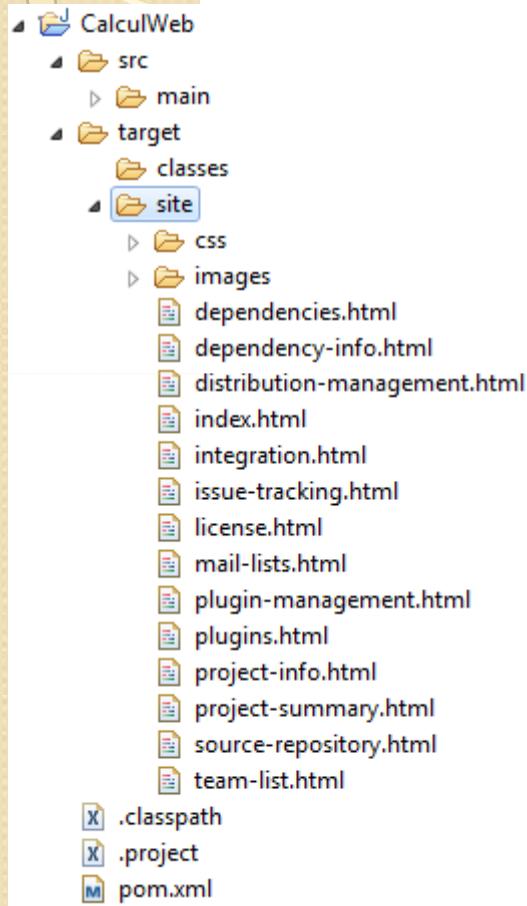
```
C:\Windows\system32\cmd.exe
C:\TP_MVN\CalculWeb>mvn tomcat7:deploy -Dtomcat.password=admin -Dtomcat.username=admin
[INFO] Scanning for projects...
[INFO]
[INFO] Building CalculWeb Maven Webapp 1.0-SNAPSHOT
[INFO]
[INFO] >>> tomcat7-maven-plugin:2.3-SNAPSHOT:deploy (default-cli) @ CalculWeb >>
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ CalculWeb
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile) @ CalculWeb ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ CalculWeb
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\TP_MVN\CalculWeb\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:testCompile (default-testCompile) @ CalculWeb
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ CalculWeb ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:2.2:war (default-war) @ CalculWeb ---
[INFO] Packaging webapp
[INFO] Assembling webapp [CalculWeb] in [C:\TP_MVN\CalculWeb\target\CalculWeb]
[INFO] Processing war project
[INFO] Copying webapp resources [C:\TP_MVN\CalculWeb\src\main\webapp]
[INFO] Webapp assembled in [59 msecs]
[INFO] Building war: C:\TP_MVN\CalculWeb\target\CalculWeb.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO] <<< tomcat7-maven-plugin:2.3-SNAPSHOT:deploy (default-cli) @ CalculWeb <<
[INFO]
[INFO] --- tomcat7-maven-plugin:2.3-SNAPSHOT:deploy (default-cli) @ CalculWeb ---
[INFO] Déploiement du war sur http://localhost:8080/CalculWeb
Uploading: http://localhost:8080/manager/text/deploy?path=%2FCalculWeb
Uploaded: http://localhost:8080/Manager/text/deploy?path=%2FCalculWeb (364 KB at
4225.4 KB/sec)
[INFO] tomcatManager status code:200, ReasonPhrase:OK
[INFO] OK - Application déployée pour le chemin de contexte /CalculWeb
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 2.740s
[INFO] Finished at: Sat Apr 19 15:45:37 GMT 2014
[INFO] Final Memory: 11M/26M
[INFO]
C:\TP_MVN\CalculWeb>_
```

Autres Gols du plugin tomcat7

- **tomcat7:deploy** : Deploy a WAR to Tomcat.
- **tomcat7:deploy-only** : Deploy a WAR to Tomcat without forking the package lifecycle.
- **tomcat7:exec-war** : Create a self executable jar file containing all necessary Apache Tomcat classes. This allows for using just java -jar mywebapp.jar to run your webapp without needing to install a Tomcat instance.
- **tomcat7:exec-war-only** : Same as exec-war goal without forking the package lifecycle.
- **tomcat7:help** : Display help information on tomcat7-maven-plugin.
- **tomcat7:redploy** : Redeploy a WAR in Tomcat.
- **tomcat7:redploy-only** : Redeploy a WAR in Tomcat without forking the package lifecycle.
- **tomcat7:run**: Runs the current project as a dynamic web application using an embedded Tomcat server.
- **tomcat7:run-war** : Runs the current project as a packaged web application using an embedded Tomcat server.
- **tomcat7:run-war-only**: Same as run-war goal without forking the package cycle.
- **tomcat7:shutdown** : Shuts down all possibly started embedded Tomcat servers.
- **tomcat7:standalone-war** : Will create an executable war file with embedded Tomcat that is also capable of being deployed elsewhere.
- **tomcat7:standalone-war-only** : Will create an executable war file with embedded Tomcat that is also capable of being deployed elsewhere.
- **tomcat7:undeploy** : Undeploy a WAR from Tomcat.

Générer le site du projet

- Exécuter la commande : **mvn site**



The screenshot shows a web browser window titled 'CalculWeb Maven Webapp'. The URL in the address bar is 'file:///C:/TP_MVN/CalculWeb/target/site/dependency-info.html'. The page header includes 'Last Published: 2014-04-18 | Version: 1.0-SNAPSHOT' and a 'CalculWeb Maven Webapp' link. The main content area has a title 'Dependency Information' and a section 'Apache Maven' containing the following XML code:

```
<dependency>
<groupId>ma_bp</groupId>
<artifactId>CalculWeb</artifactId>
<version>1.0-SNAPSHOT</version>
<type>war</type>
</dependency>
```

Below this is a section 'Apache Buildr' which is currently empty.



• GESTION DES PLUGINS

med@youssf.net



Gestion des plugins

- Quand on télécharge Maven, il ne comprend que le moteur qui sert à télécharger des plugins.
- Tous les goals Maven sont dans des plugins même les plus indispensables comme le plugin compiler.
- Ainsi, il faut s'attendre à voir Maven télécharger énormément de plugins lors de la première exécution d'un goal.



Cartographie des plugins

- **Core plugins :**
 - **clean** : nettoie le répertoire de travail du projet : suppression des fichiers générés, etc.
 - **compile** : compilation des sources du projet
 - **resources** : copie les ressources du projet dans le répertoire de build (classes ou test-classes)
 - **site** : génère le site web du projet
 - **surefire** : joue les tests unitaires
 - Et aussi : **deploy, install, verifier**
- **Packaging plugins :**
 - **jar** : construit un jar à partir du projet
 - **war** : construit un war à partir du projet
 - Et aussi : **ear, ejb, rar**



Cartographie des plugins

- **Tools plugins :**
 - archetype : génère une structure de projet vide à partir d'un modèle
 - assembly : génère une distribution de sources / fichiers binaires
 - dependency : manipulation et analyse des dépendances
 - help : donne des informations sur l'environnement de travail du projet
 - Et aussi : ant, antrun, enforcer, gpg, invoker, one, patch, release, remote-resources, repository, scm, source, stage, etc.
- **Reporting plugins :**
 - checkstyle : génère un rapport d'audit de code checkstyle
 - javadoc : génère la javadoc du projet
 - pmd : génère un rapport PMD (pour analyser le code source Java)
 - project-info-reports : génère un rapport standard du projet
 - surefire-reports : génère le rapport de tests unitaires
 - jdepend : génère un rapport de métriques de code
 - cobertura : génère un rapport de couverture de tests
 - Findbugs : génère un rapport d'audit de code findbugs
 - Et aussi : changelog, changes, clover, doap, docck, jxr, etc.
- **IDE plugins :**
 - eclipse : Génère un fichier .project pour intégration du projet dans Eclipse
 - Et aussi : idea

Configuration des plugins

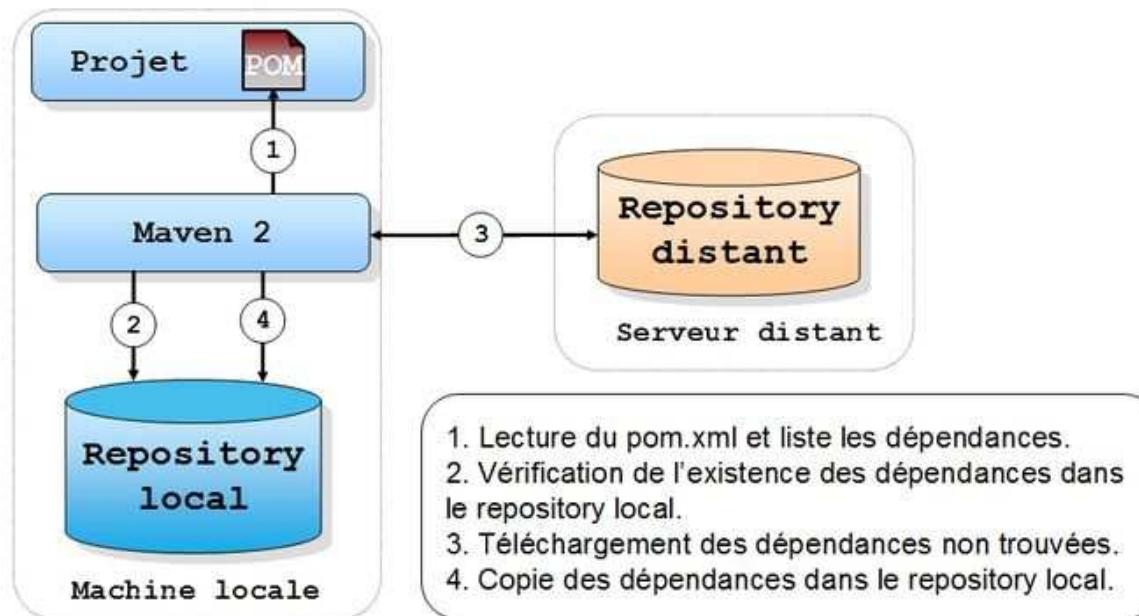
```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.0.2</version>
      <configuration>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- Un projet héritera d'un pom générique qui sélectionne au mieux les versions de chaque plugin.
- La liste des plugins disponibles sont sur le site suivant :
 - <http://maven.apache.org/plugins/>

Repositories

- Le repository représente un élément important de Maven.
- Afin de bien gérer les dépendances, Maven utilise un système qui s'appuie sur des repositories pour télécharger automatiquement les composants qu'il a besoin.
- Mais pour éviter que les fichiers se téléchargent à chaque reconstruction, Maven stocke automatiquement les dépendances nécessaires dans le repository local.
- Par exemple, à la première exécution de maven, maven télécharge plusieurs plugins requis. Il se peut que cela prenne un certain temps.
- Le local repository se trouve toujours par défaut dans le répertoire **.m2/repository**

Gestion des dépendances par Maven 2



Structure d'un repository

- Maven utilise, par défaut, un serveur central qui contient énormément de jar et pratiquement tous les plugins de base de Maven.
- Pour ajouter des repository il faut ajouter dans le pom.xml
- Comme on peut le voir, Maven différencie les repository qui contiennent les plugins de ceux qui contiennent les dépendances.

```
<repositories>
  <repository>
    <id>id</id>
    <name>name</name>
    <url>url</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>id</id>
    <name>name</name>
    <url>url</url>
  </pluginRepository>
</pluginRepositories>
```

Configuration du Repository central

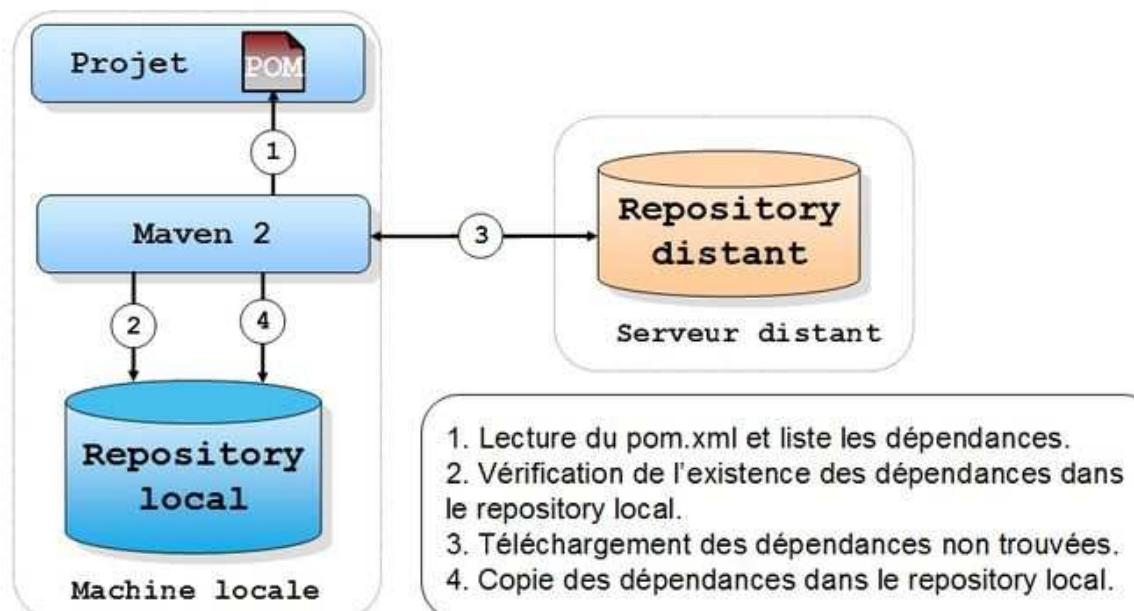
```
<repositories>
  <repository>
    <id>central</id>
    <name>
      Maven Repository Switchboard
    </name>
    <layout>default</layout>
    <url>
      http://repo1.maven.org/maven2
    </url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

```
<pluginRepositories>
  <pluginRepository>
    <id>central</id>
    <name>
      Maven Plugin Repository
    </name>
    <url>
      http://repo1.maven.org/maven2
    </url>
    <layout>
      default
    </layout>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <releases>
      <updatePolicy>
        never
      </updatePolicy>
    </releases>
  </pluginRepository>
</pluginRepositories>
```

Gestion des dépendances

- Avec Maven toutes les dépendances d'un projet sont déclarées dans le fichier pom.xml
- Le plugin Maven de gestion de dépendances se charge de télécharger sur les repositories distants les fichiers jar indiqués comme dépendances, s'ils ne se trouvent pas dans le repository local.

Gestion des dépendances par Maven 2



Déclaration des dépendances

```
<properties>
    <servlet.version>2.5</servlet.version>
    <spring-framework.version>3.2.3.RELEASE</spring-framework.version>
</properties>
<dependencies>
    <!-- Spring MVC -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring-framework.version}</version>
        <scope>compile</scope>
    </dependency>
    <!-- Other Web dependencies -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId> servlet-api</artifactId>
        <version>${servlet.version}</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

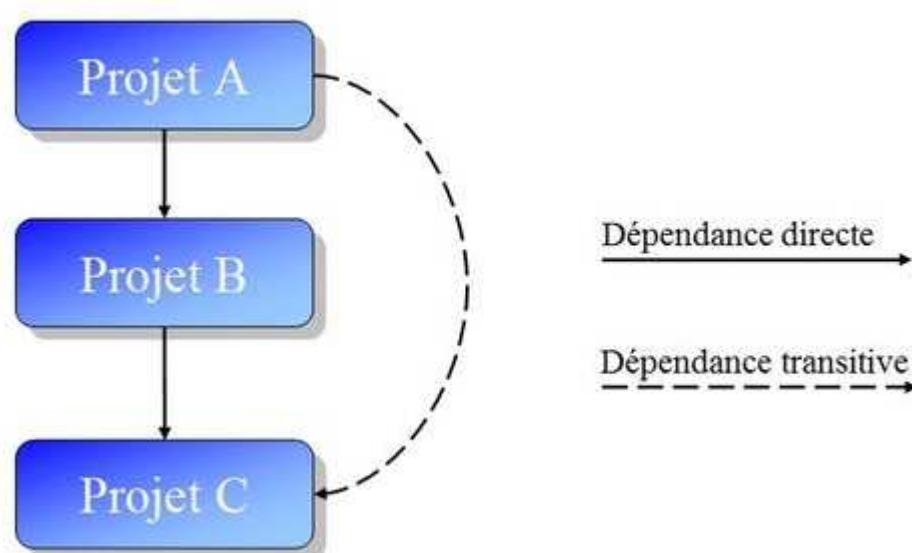


Déclaration des dépendances

- Les seuls paramètres obligatoires sont le **groupId** et l'**artifactId**.
- **Il est très vivement recommandé de toujours spécifier la version.** Sans cela, Maven utilise toujours la dernière version en date.
- Il est tout à fait possible que la mise à jour d'une dépendance publiée dans une version alpha soit automatiquement utilisée et empêche le projet de tourner alors qu'aucune modification n'y a été apportée.
- Le paramètre scope est parfois nécessaire. Les différentes valeurs à prendre en compte sont les suivantes :
 - **compile** : C'est la valeur par défaut, la dépendance sera toujours disponible dans le classpath.
 - **provided** : Indique que la dépendance est nécessaire pour la compilation mais sera fournie par le container ou le JDK et donc ne sera pas fournie dans le package.
 - **runtime** : Indique que la dépendance est nécessaire pour l'exécution mais pas pour la compilation.
 - **test** : Indique que la dépendance est nécessaire pour la compilation et l'exécution des tests unitaires.
- Le scope provided est très intéressant pour les servlet. Les jars sont fournis automatiquement par Tomcat (ou Jetty...) mais il est nécessaire de les avoir pour la compilation.

Dépendances transitives

- La gestion des dépendances de Maven permet des dépendances transitives.
- Si un artifact X dépend d'un artifact Y qui dépend d'un artifact Z, la résolution des dépendances de X trouvera Y et Z.
- Ce mécanisme implique souvent le téléchargement de beaucoup de librairies. Chaque artifact va dépendre de tous les autres dont il est susceptible d'avoir besoin.
- La réponse à la multiplication des dépendances est la division en modules des grands frameworks.
- Cela permet de n'utiliser que certains morceaux d'un framework et de s'abstraire des dépendances des modules qu'on n'utilisera pas.





Exclusion des dépendances

- En allant plus loin, il est possible de trouver des situations où les dépendances transitives posent problèmes.
- Par exemple, une dépendance transitive sur un framework dans une version trop vieille peut poser problème si votre application utilise une version récente.
- Suivant les versions de Maven et le plugin qui utilise la résolution de dépendance, **il n'est pas possible de savoir précisément quelle version de l'artifact sera utilisée**.
- Notamment dans les packagings war, il est possible que les deux fichiers jar avec les deux versions différentes soit présents dans le répertoire WEB-INF/lib.
- Pour gérer ce cas de figure, il faut utiliser les exclusions qui permettent d'interdire les dépendances transitives. La syntaxe sera la suivante :

Déclaration des exclusions

```
<!-- Spring -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${org.springframework-version}</version>
    <exclusions>
        <!-- Exclude Commons Logging in favor of SLF4j -->
        <exclusion>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

Installation du projet

- Tous les projets sont définis comme des paquets Maven.
- Il est donc possible de publier ces paquets.
- Tout d'abord pour publier dans le localrepository, il suffit d'utiliser le goal install :

mvn install

- Pour l'installer sur un repository externe, il faut lui configurer dans le pom.xml la gestion de la distribution :

```
<distributionManagement>
  <repository>
    <id>ganesh3-repo</id>
    <name>Ganesh Repository for Maven2</name>
    <url>file://${deploy.repository}</url>
  </repository>
</distributionManagement>
```

- L'URL peut être exprimée au moyen de beaucoup de protocoles, ici on voit file, mais cela peut être également scp, ftp, http (à condition qu'il y ait un webdav) etc...

Ajout d'un jar à un repository

- On finit toujours par utiliser un jar qui n'est sur aucun repository Maven.
- Pourtant, les principes de Maven nous interdisent d'ajouter un jar directement dans les sources du projet.
- Pour venir à bout de cette particularité, Maven propose la possibilité d'ajouter manuellement des artifacts dans les repository.
- Une fois installé, il est possible d'en dépendre de la façon habituelle.
- Pour installer dans le repository local un artifact à partir d'un fichier, il faut utiliser le goal **install:install-file** .
- Il faut renseigner en ligne de commande les informations nécessaires à définir l'artifact qui correspondra au fichier qu'on installe :

```
mvn install:install-file -Dfile=your-artifact-1.0.jar \
[-DpomFile=your-pom.xml] \
[-DgroupId=org.some.group] \
[-DartifactId=your-artifact] \
[-Dversion=1.0] \
[-Dpackaging=jar] \
[-Dclassifier=sources] \
[-DgeneratePom=true] \
[-DcreateChecksum=true]
```

Ajout d'un jar à un repository

- Il existe la même commande pour installer un artifact dans un repository distant. Il s'agira cette fois ci du goal
 - deploy:deploy-file** .

```
mvn deploy:deploy-file -Durl=file:///C:/m2-repo \
-DrepositoryId=some.id \
-Dfile=your-artifact-1.0.jar \
[-DpomFile=your-pom.xml] \
[-DgroupId=org.some.group] \
[-DartifactId=your-artifact] \
[-Dversion=1.0] \
[-Dpackaging=jar] \
[-Dclassifier=test] \
[-DgeneratePom=true] \
[-DgeneratePom.description="My Project Description"] \
[-DrepositoryLayout=legacy] \
[-DuniqueVersion=false]
```



Proxy d'entreprises

- Si on prend les sources d'un projet Maven, elles ne contiennent pas les dépendances.
- Pourtant, dès qu'on lancera une commande de compilation, les dépendances seront téléchargées sur le poste.
- Ce mécanisme est très puissant mais repose sur une supposition qui peut avoir ses limites :
 - toutes les librairies sont toujours disponibles sur Internet. Le corollaire est que si certains serveurs Web sont en panne au moment où l'on désire compiler notre projet, la compilation va échouer.
- Il est également souvent nécessaire dans une entreprise de posséder un repository interne qui permet de rendre accessible facilement les librairies de l'entreprise.
- Le principe du proxy d'entreprise répond à ces attentes.

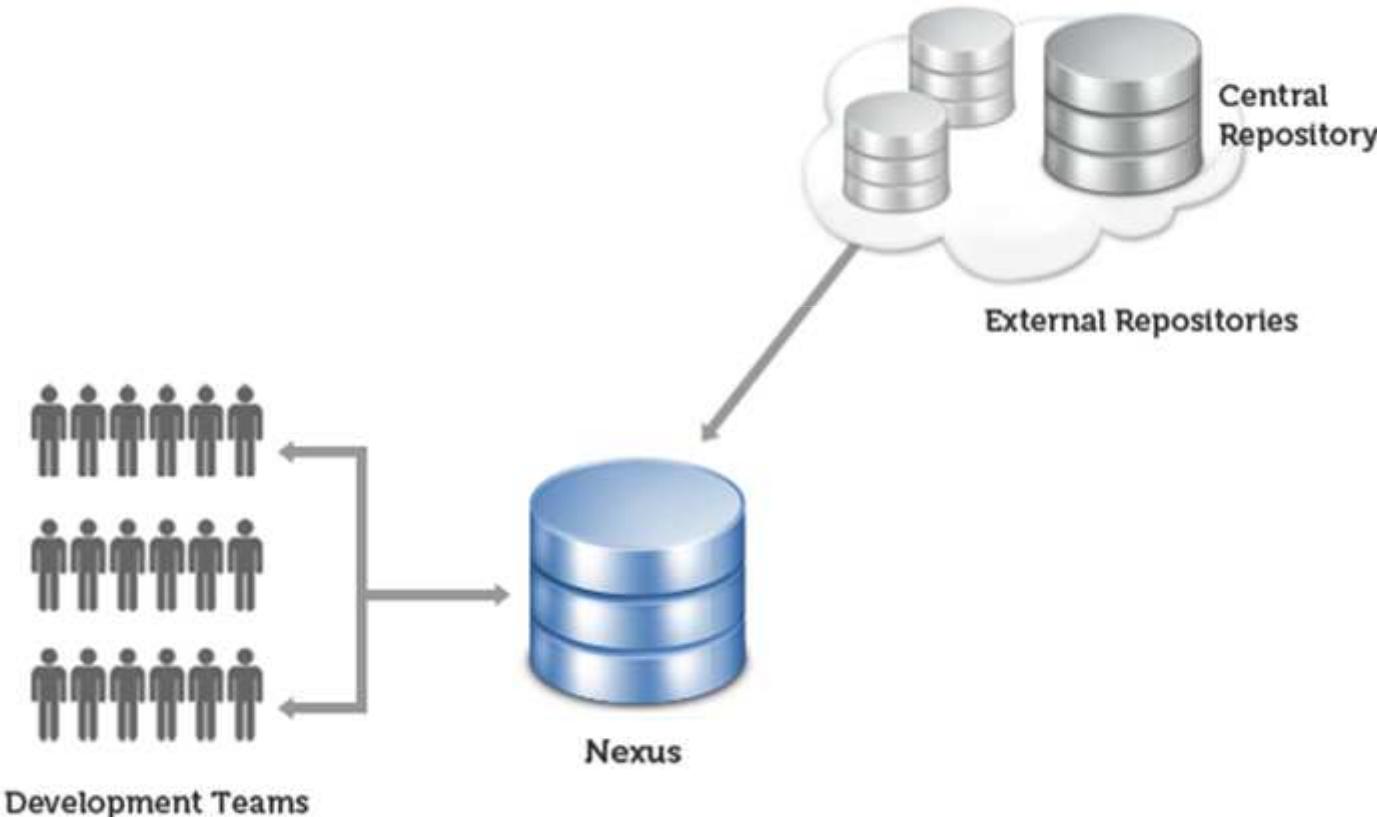


Proxy d'entreprises

- Son fonctionnement est le suivant : lorsqu'une instance de Maven sur un poste de développeur demande un artifact, il s'adresse au proxy (via la configuration dans le pom).
- Le proxy va alors chercher l'artifact sur Internet et lui rendre. Lors de la seconde demande, l'artifact sera immédiatement disponible sur le proxy.
- Le plus souvent, le proxy d'entreprise propose aussi la fonctionnalité de repository d'entreprise et propose des solutions simplifiées pour déployer des artifact dessus.
- Les solutions les plus courantes pour fournir ce service sont les suivantes :
 - Nexus : <http://www.sonatype.org/nexus/>
 - Archiva : <http://archiva.apache.org/index.cgi>

Proxy Maven : Nexus

<http://books.sonatype.com/nexus-book/reference/index.html>

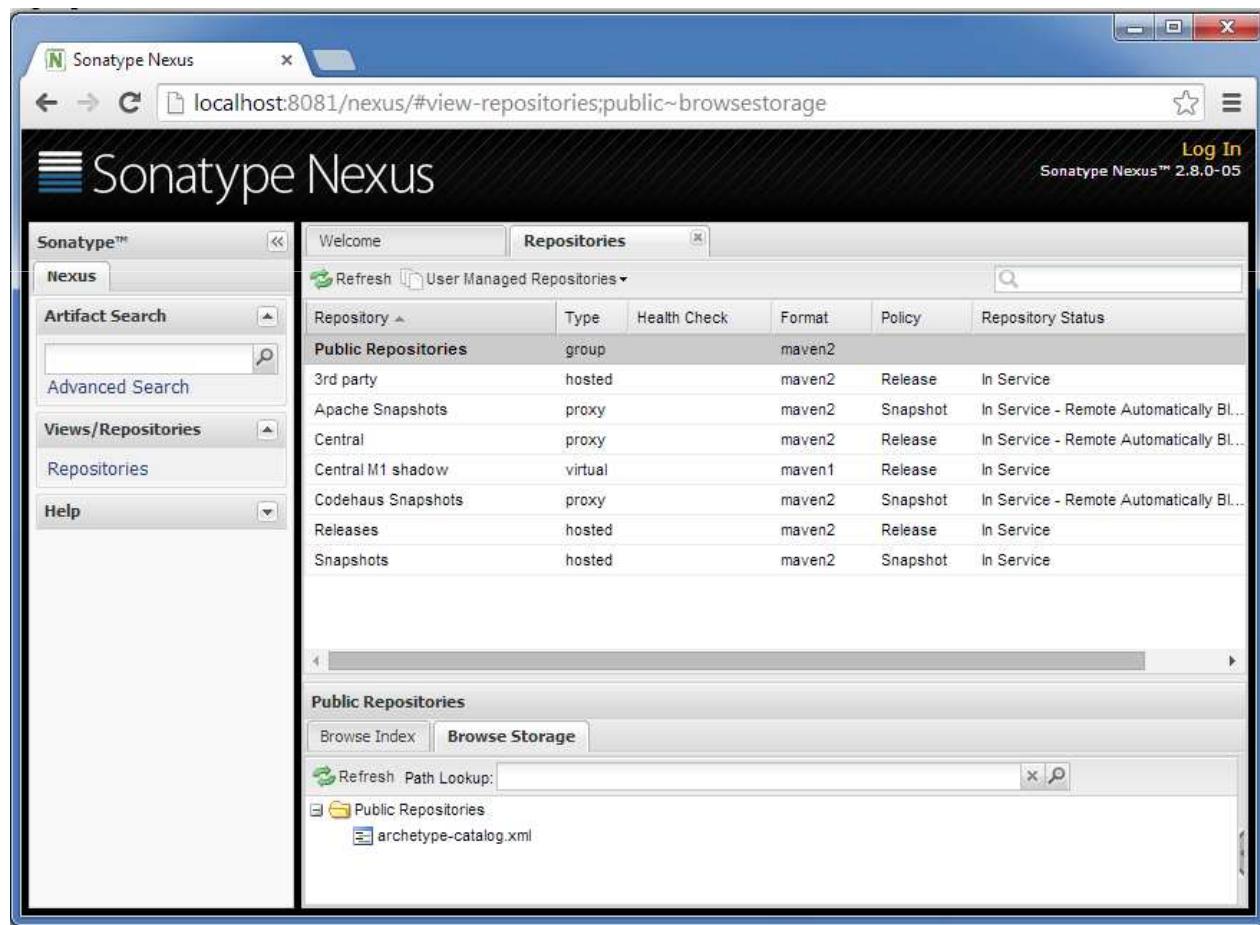


Démarrage de Nexus



```
C:\Windows\system32\cmd.exe
C:\Outils\nexus-2.8.0-05-bundle\nexus-2.8.0-05\bin\jsw\windows-x86-32>console-nexus.bat
```

<http://localhost:8081/nexus>



The screenshot shows the Sonatype Nexus web interface. The top navigation bar includes links for 'Log In' and 'Sonatype Nexus™ 2.8.0-05'. The main content area has tabs for 'Welcome' and 'Repositories'. The 'Repositories' tab is active, displaying a table of public repositories:

Repository	Type	Health Check	Format	Policy	Repository Status
Public Repositories	group		maven2		
3rd party	hosted		maven2	Release	In Service
Apache Snapshots	proxy		maven2	Snapshot	In Service - Remote Automatically Bl...
Central	proxy		maven2	Release	In Service - Remote Automatically Bl...
Central M1 shadow	virtual		maven1	Release	In Service
Codehaus Snapshots	proxy		maven2	Snapshot	In Service - Remote Automatically Bl...
Releases	hosted		maven2	Release	In Service
Snapshots	hosted		maven2	Snapshot	In Service

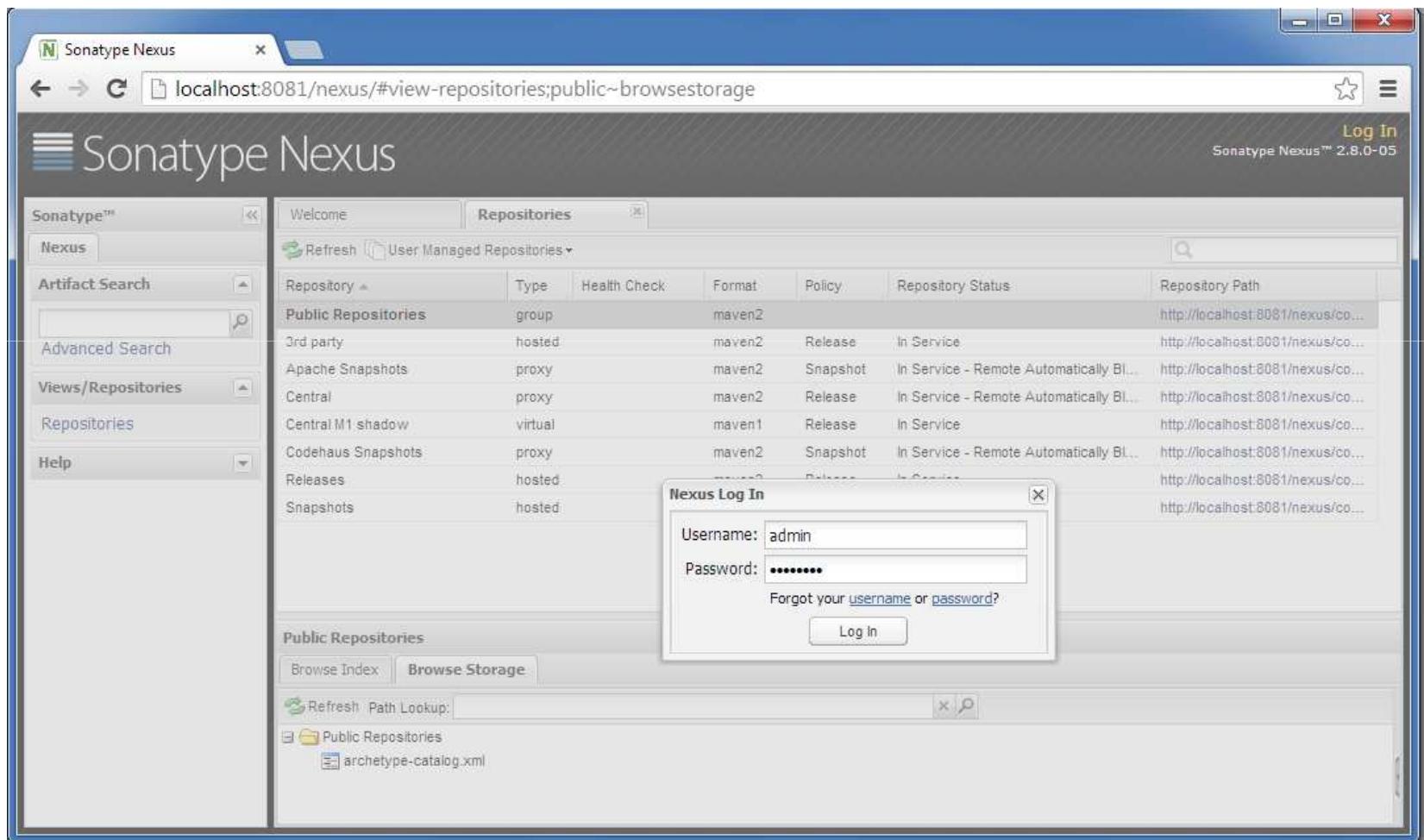
Below the table, there is a section titled 'Public Repositories' with tabs for 'Browse Index' and 'Browse Storage'. The 'Browse Storage' tab is selected, showing a tree view of repository contents:

- Public Repositories
 - archetype-catalog.xml

med@youssfi.net

Authentification

- Username : admin
- Password : admin123



Après Authentification

- L'administrateur de Nexus peut gérer les repositories

The screenshot shows the Sonatype Nexus web interface. The title bar reads "Sonatype Nexus" and the address bar shows "localhost:8081/nexus/#view-repositories;public~browsestorage". The top right corner indicates the user is "admin" and the version is "Sonatype Nexus™ 2.8.0-05". The main content area is titled "Sonatype Nexus" and displays the "Repositories" section. On the left, there is a sidebar with links for "Nexus", "Artifact Search", "Views/Repositories", "Security", "Administration", and "Help". The central part of the screen shows a table titled "Public Repositories" with the following data:

Repository	Type	Format	Policy	Repository Status	Repository Path
3rd party	hosted	maven2	Release	In Service	http://localhost:8081/nexus/co...
Apache Snapshots	proxy	maven2	Snapshot	In Service - Remote Automatically Bl...	http://localhost:8081/nexus/co...
Central	proxy	maven2	Release	In Service - Remote Automatically Bl...	http://localhost:8081/nexus/co...
Central M1 shadow	virtual	maven1	Release	In Service	http://localhost:8081/nexus/co...
Codehaus Snapshots	proxy	maven2	Snapshot	In Service - Remote Automatically Bl...	http://localhost:8081/nexus/co...

Below the table, there is a "Browse Storage" tab and a "Path Lookup" section showing "Public Repositories" and "archetype-catalog.xml".



Connecter votre instance de Maven au proxy Nexus

- Maintenant nous allons configurer l'instance Maven du développeur pour qu'elle puisse chercher les dépendances dans le proxy Nexus au lieu du serveur central
- Pour cela vous aurez besoin de modifier le fichier de configuration setting.xml de Maven:



Connecter votre instance de Maven au proxy Nexus

- Pour changer le dossier de repository local de maven ajouter la configuration suivante au fichier setting.xml de maven :
- <localRepository>\${user.home}/.m2/rep</localRepository>
- Déclarer l'adresse http de Nexus dans mirrors
- <mirrors>
 <mirror>
 <id>nexus</id>
 <mirrorOf>*</mirrorOf>
 <url>http://localhost:8081/nexus/content/groups/public/</url>
 </mirror>
</mirrors>

Connecter votre instance de Maven au proxy Nexus

- Déclarer Nexus comme profile dans l'élément `<profiles>`:

```
• <profile>
  <id>nexus</id>
  <!--Enable snapshots for the built in central repo to direct -->
  <!--all requests to nexus via the mirror -->
  <repositories>
    <repository>
      <id>central</id>
      <url>http://localhost:8081/nexus/content/groups/public/</url>
      <releases><enabled>true</enabled></releases>
      <snapshots><enabled>true</enabled></snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>central</id>
      <url>http://localhost:8081/nexus/content/groups/public/</url>
      <releases><enabled>true</enabled></releases>
      <snapshots><enabled>true</enabled></snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```



Connecter votre instance de Maven au proxy Nexus

- Activer le profile Nexus

- <activeProfiles>
 <!--make the profile active all the time -->
 <activeProfile>nexus</activeProfile>
 </activeProfiles>



MISE EN ŒUVRE DE MAVEN DANS LES APPLICATION JAVA/JEE

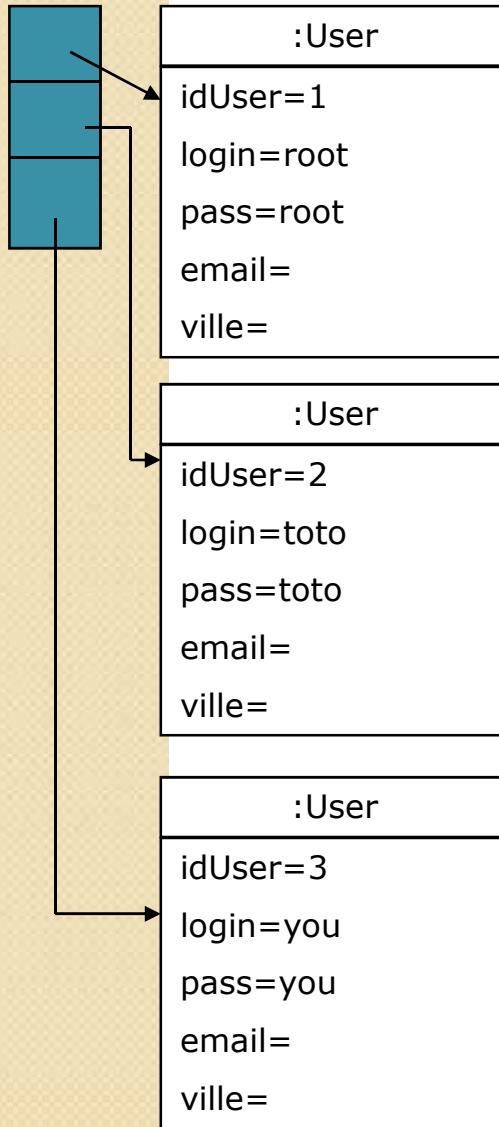


Mapping Objet Relationnel avec Hibernate

med@youssf.net

Application orientée objet

Users:Collection



Mapping Objet Relationnel

```
public List<User> getAllUsers() {  
    List<User> users=new ArrayList<User>();  
    Class.forName("com.mysql.jdbc.Driver");  
    Connection conn=DriverManager.getConnection  
    ("jdbc:mysql://localhost:3306/DB_USERS","root","");
    PreparedStatement ps=conn.prepareStatement  
    ("select * from users");
    ResultSet rs=ps.executeQuery();
    while(rs.next()){
        User u=new User();
        u.setIdUser(rs.getInt("ID_USER"));
        u.setLogin(rs.getString("LOGIN"));
        u.setPass(rs.getString("PASS"));
        users.add(u);
    }
    return(users);
}
```

Users : Table					
	idUser	login	pass	email	ville
1	1	root	root	root@yahoo.fr	casa
2	2	toto	toto	toto@yahoo.fr	rabat
3	3	you	you	med@yahoo.fr	casa
*	(NuméroAuto)				

Base de données relationnelle

Introduction

- Travailler dans les deux univers que sont l'orienté objet et la base de données relationnelle peut être lourd et consommateur en temps dans le monde de l'entreprise d'aujourd'hui.
- Hibernate est un outil de mapping objet/relationnel pour le monde Java.
- Le terme mapping objet/relationnel (ORM) décrit la technique consistant à faire le lien entre la représentation objet des données et sa représentation relationnelle basée sur un schéma SQL.

Introduction

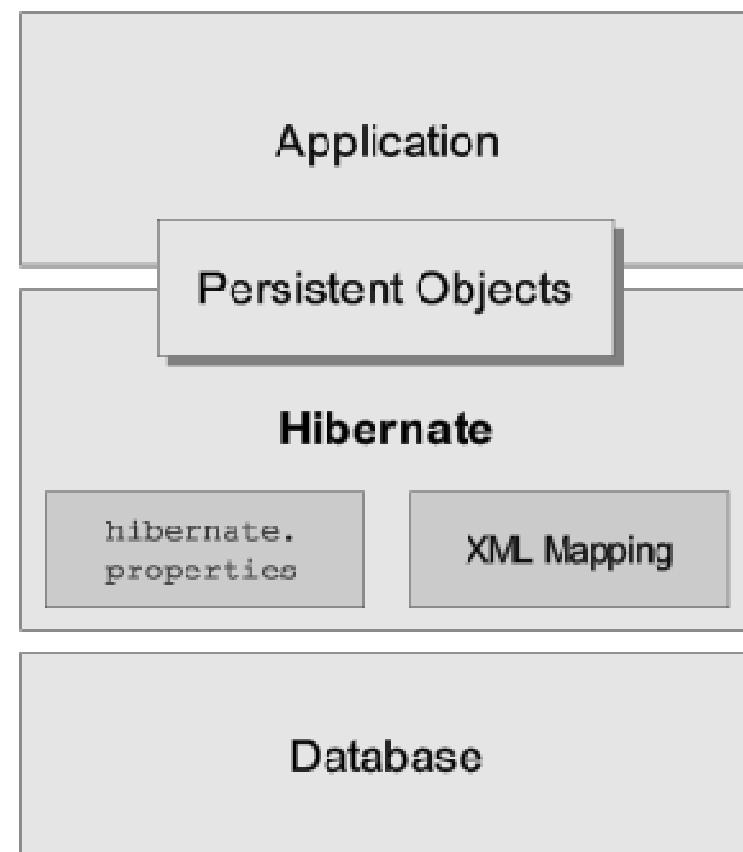
- Hibernate s'occupe du transfert des objets Java dans les tables de la base de données
- En plus, il permet de requêter les données et propose des moyens de les récupérer.
- Il peut donc réduire de manière significative le temps de développement qui aurait été autrement perdu dans une manipulation manuelle des données via SQL et JDBC

But de Hibernate

- Le but d'Hibernate est de libérer le développeur de 95 pourcent des tâches de programmation liées à la persistance des données communes.
- Hibernate assure la portabilité de votre application si vous changer de SGBD.
- Hibernate propose au développeur des méthodes d'accès aux bases de données plus efficace ce qui devrait rassurer les développeurs.
- Maven est utile pour les applications dont la couche métier est implémentée au niveau de l'application et non au niveau du SGBD en utilisant des procédures stockées.

Première approche de l'architecture d'Hibernate

- Hibernate permet d'assurer la persistance des objets de l'application dans un entrepôt de données.
- Cet entrepôt de données est dans la majorité des cas une base de données relationnelle, mais il peut être un fichier XML.
- Le mapping des objets est effectuée par Hibernate en se basant sur des fichiers de configuration en format texte ou souvent XML.

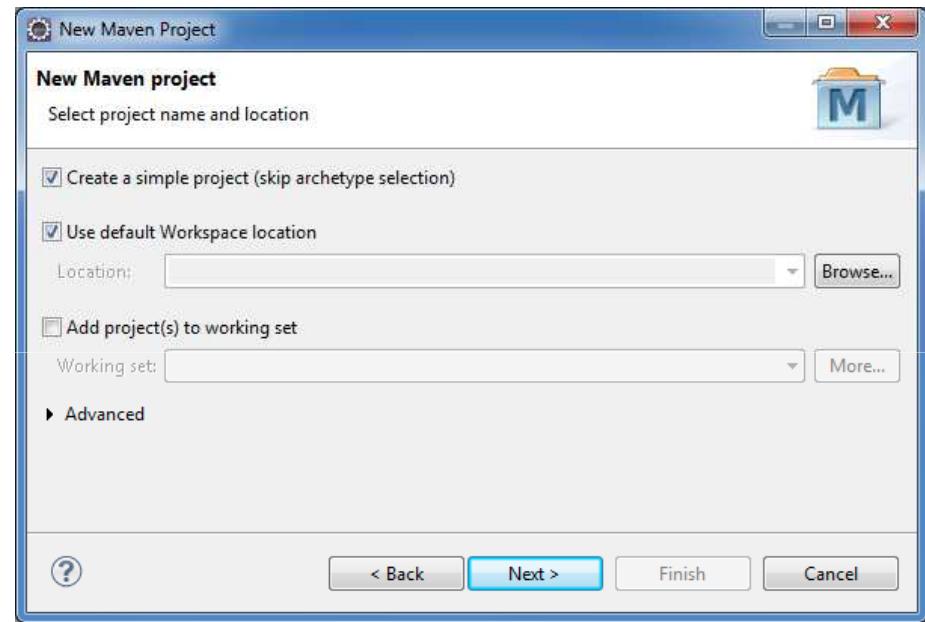
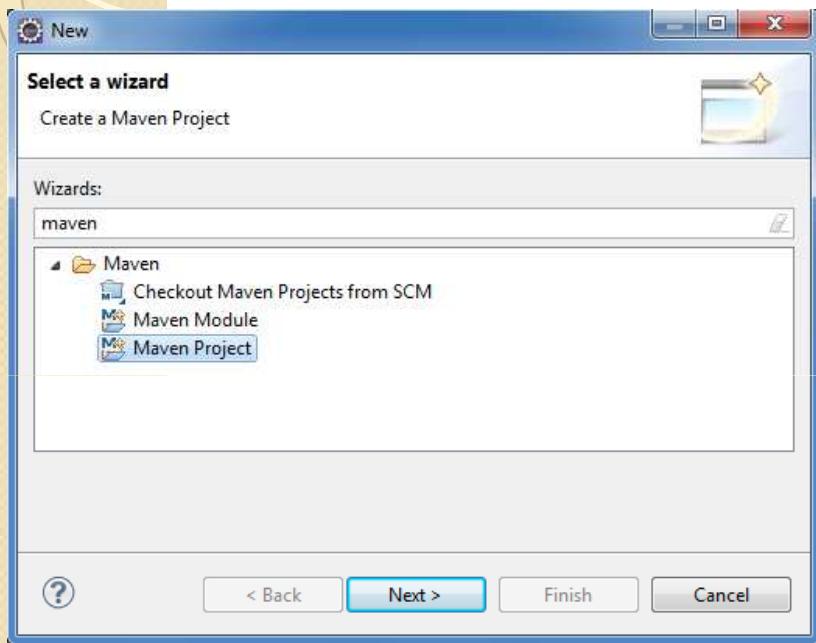


Exemple d'application

- Supposant que l'on souhaite créer une application qui permet de gérer le catalogue des produits appartenant à des catégories.
- Chaque produit est défini par :
 - Sa référence de type String
 - Sa désignation de type String
 - Son prix de type double
 - Sa quantité de type int
 - Sa disponibilité de type boolean
 - sa date création de type Date
- Une catégorie est définie par :
 - Son code de type Long (Auto Increment)
 - Son nom de type String
 - sa photo de type byte[]
- L'application doit permettre
 - D'ajouter une nouvelle catégorie
 - Ajouter un produit appartenant à une catégorie
 - Consulter toutes les catégories
 - Consulter les produits dont le nom contient un mot clé
 - Consulter les produits d'une catégorie
 - Consulter un produit
 - Mettre à jour un produit
 - Supprimer une catégorie

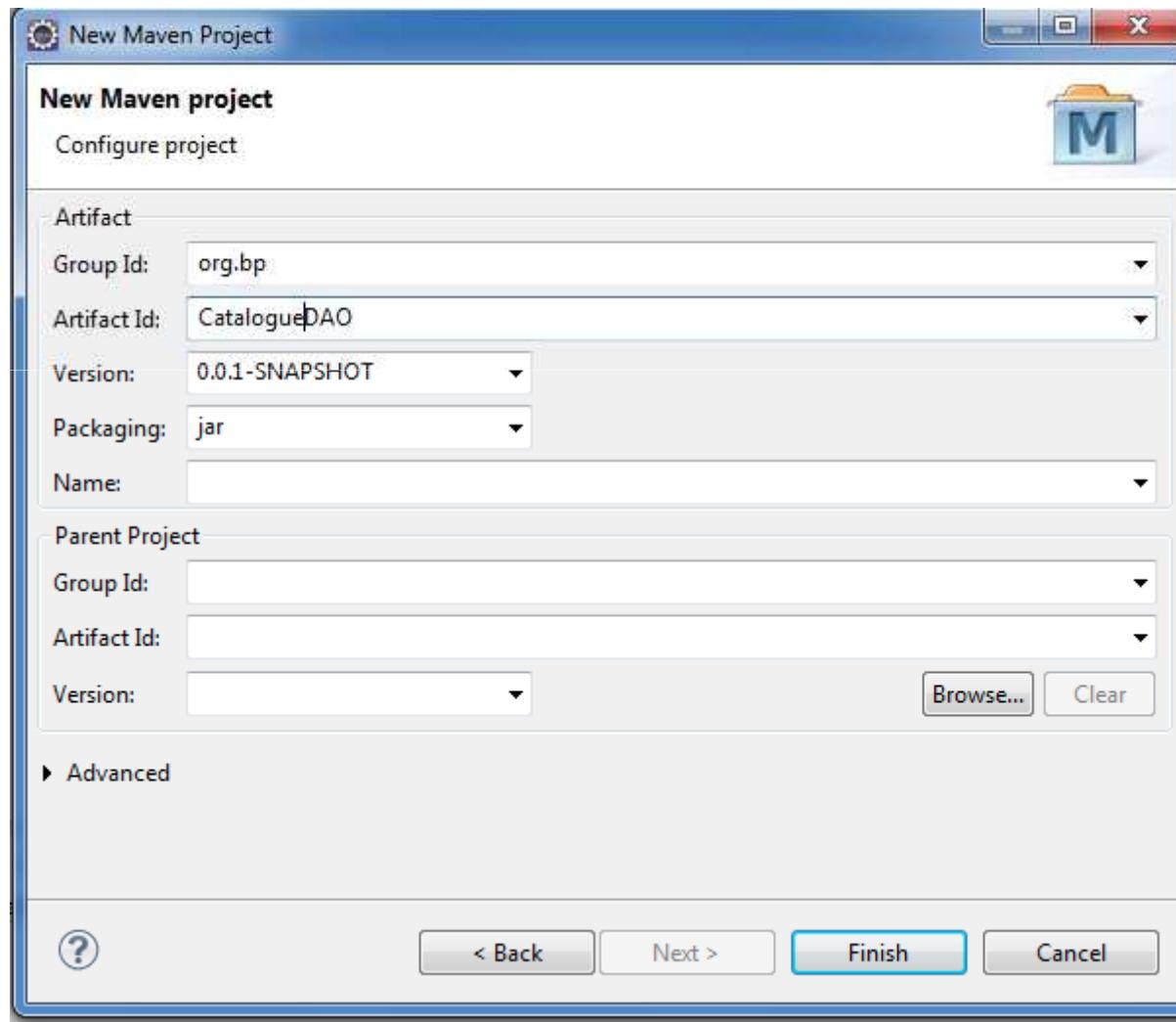
Projet Maven

- File>New>Maven>Maven Project



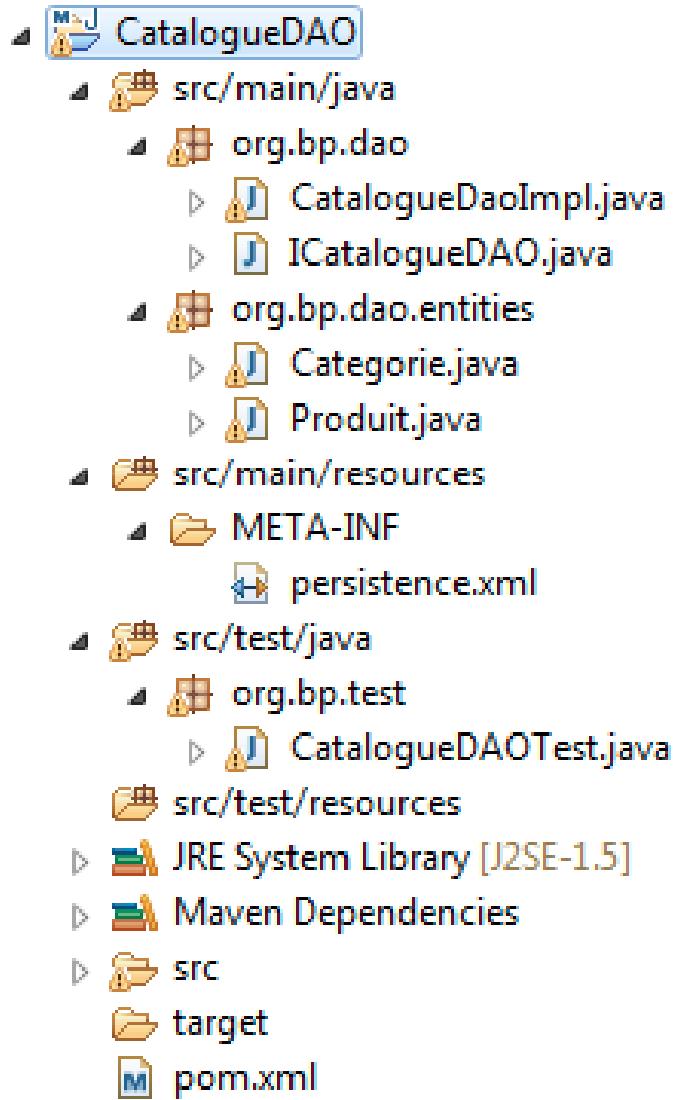
Paramètre du projet

- Group Id : org.bp
- Artifact Id : CatalogueDAO



Structure du projet

- Vue Packages



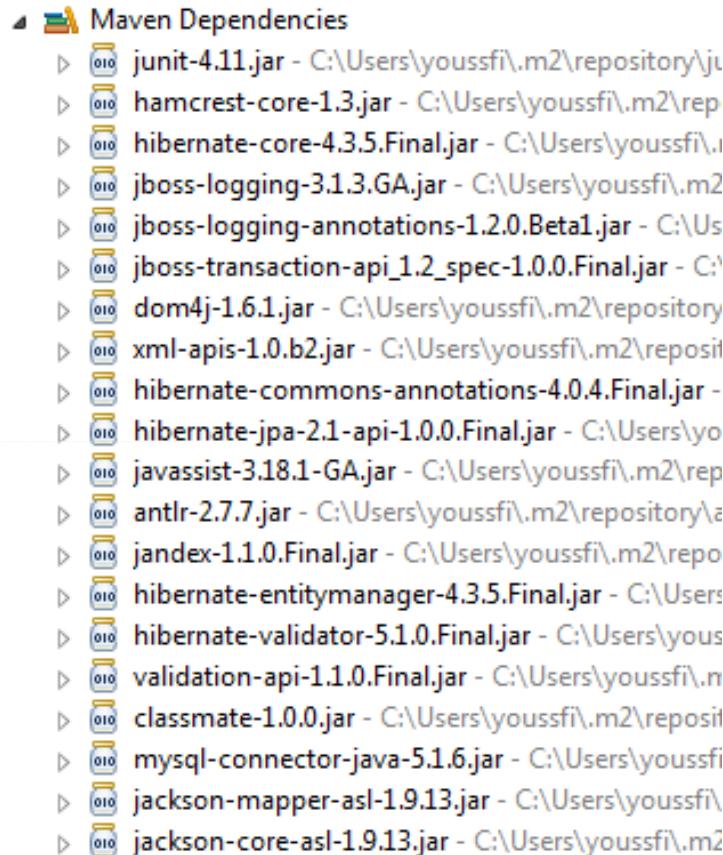


Dépendances Maven

- ◀ Maven Dependencies
 - ▷ junit-4.11.jar - C:\Users\youssf\m2\repository\ju
 - ▷ hamcrest-core-1.3.jar - C:\Users\youssf\m2\rep
 - ▷ hibernate-core-4.3.5.Final.jar - C:\Users\youssf\m2\re
 - ▷ jboss-logging-3.1.3.GA.jar - C:\Users\youssf\m2\re
 - ▷ jboss-logging-annotations-1.2.0.Beta1.jar - C:\Us
 - ▷ jboss-transaction-api_1.2_spec-1.0.0.Final.jar - C:\U
 - ▷ dom4j-1.6.1.jar - C:\Users\youssf\m2\repository\dom4j\1.6.1\dom4j-1.6.1.jar
 - ▷ xml-apis-1.0.b2.jar - C:\Users\youssf\m2\repository\xml-apis\1.0.b2\xml-apis-1.0.b2.jar
 - ▷ hibernate-commons-annotations-4.0.4.Final.jar - C:\Users\youssf\m2\repository\org\hibernate\hibernate-commons-annotations\4.0.4.Final\hibernate-commons-annotations-4.0.4.Final.jar
 - ▷ hibernate-jpa-2.1-api-1.0.0.Final.jar - C:\Users\youssf\m2\repository\org\hibernate\hibernate-jpa-2.1-api\1.0.0.Final\hibernate-jpa-2.1-api-1.0.0.Final.jar
 - ▷ javassist-3.18.1-GA.jar - C:\Users\youssf\m2\repository\javassist\3.18.1-GA\javassist-3.18.1-GA.jar
 - ▷ antlr-2.7.7.jar - C:\Users\youssf\m2\repository\antlr\2.7.7\antlr-2.7.7.jar
 - ▷ jandex-1.1.0.Final.jar - C:\Users\youssf\m2\repository\org\hibernate\jandex\1.1.0.Final\jandex-1.1.0.Final.jar
 - ▷ hibernate-entitymanager-4.3.5.Final.jar - C:\Users\youssf\m2\repository\org\hibernate\hibernate-entitymanager\4.3.5.Final\hibernate-entitymanager-4.3.5.Final.jar
 - ▷ hibernate-validator-5.1.0.Final.jar - C:\Users\youssf\m2\repository\org\hibernate\validator\5.1.0.Final\hibernate-validator-5.1.0.Final.jar
 - ▷ validation-api-1.1.0.Final.jar - C:\Users\youssf\m2\repository\org\hibernate\validation\1.1.0.Final\validation-api-1.1.0.Final.jar
 - ▷ classmate-1.0.0.jar - C:\Users\youssf\m2\repository\org\hibernate\classmate\1.0.0\classmate-1.0.0.jar
 - ▷ mysql-connector-java-5.1.6.jar - C:\Users\youssf\m2\repository\mysql\mysql-connector-java\5.1.6\mysql-connector-java-5.1.6.jar
 - ▷ jackson-mapper-asl-1.9.13.jar - C:\Users\youssf\m2\repository\com\fasterxml\jackson\mapper\jackson-mapper-asl\1.9.13\jackson-mapper-asl-1.9.13.jar
 - ▷ jackson-core-asl-1.9.13.jar - C:\Users\youssf\m2\repository\com\fasterxml\jackson\core\jackson-core-asl\1.9.13\jackson-core-asl-1.9.13.jar

```
<!-- JUNIT -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
</dependency>
<!-- Hibernate -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>4.3.5.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>4.3.5.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.0-api</artifactId>
    <version>1.0.1.Final</version>
</dependency>
```

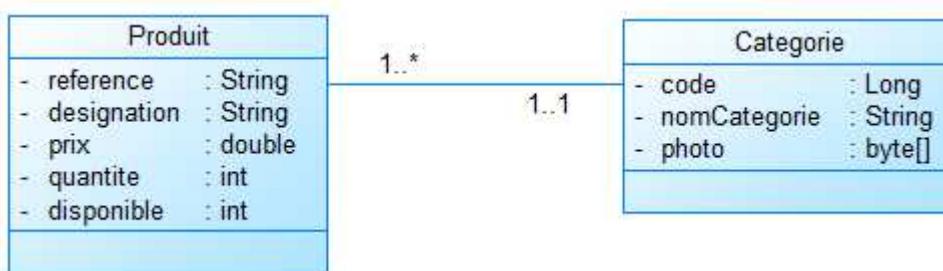
Dépendances Maven



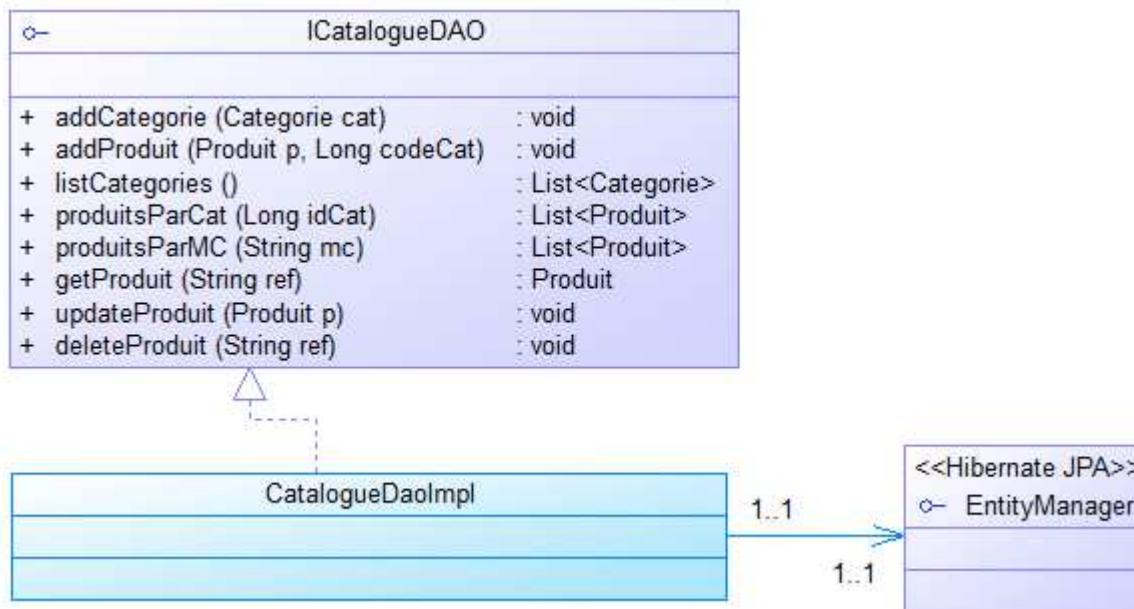
```
<!-- Hibernate Validator -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.1.0.Final</version>
</dependency>
<!-- Jackson JSON -->
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.13</version>
</dependency>
<!-- MySQL Driver -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
</dependency>
```

Diagramme de classes

- Entités :



- Traitements:



Categorie

|..|

Produit

|..*

Implémentation des entités

```
package org.bp.dao.entities;  
import java.io.Serializable;import java.util.Collection;  
public class Categorie implements Serializable {  
    private Long codeCategorie;  private String nomCategorie;  
    private byte[] photo;  
    private Collection<Produit> produits;  
    // Constructeurs  
    public Categorie() { }  
    public Categorie(String nomCategorie) { this.nomCategorie = nomCategorie; }  
    // Getters et Setters }
```

```
package org.bp.dao.entities;  
import java.io.Serializable;  
public class Produit implements Serializable {  
    private String reference;  private String designation;  
    private double prix;  private int quantite;  private boolean disponible;  
    private Categorie categorie;  
    public Produit() {}  
    public Produit(String ref, String des, double prix, int quantite) {  
this.reference = ref;this.designation = des; this.prix = prix;  
this.quantite = quantite; this.disponible=true;  
}  
    // Getters et Setters  
}
```



Mapping Objet Relationnel des entités

- Il existe deux moyens pour mapper les entités :
 - Créer des fichier XML de mapping
 - Utiliser les Annotations JPA
- L'utilisation des annotations JPA laisse votre code indépendant de Hibernate.
- La création des fichiers XML de mapping a l'avantage de séparer le code java du mapping objet relationnel.
- Dans cette formation, nous allons utiliser les annotations JPA



Quelques annotations JPA de Mapping des Entités

- **@Table**
 - Préciser le nom de la table concernée par le mapping. Par défaut c'est le nom de la classe qui sera considérée
- **@Column**
 - Associer un champ de la colonne à la propriété. Par défaut c'est le nom de la propriété qui sera considérée.
- **@Id**
 - Associer un champ de la table à la propriété en tant que clé primaire
- **@GeneratedValue**
 - Demander la génération automatique de la clé primaire au besoin
- **@Basic**
 - Représenter la forme de mapping la plus simple. Cette annotation est utilisée par défaut
- **@Transient**
 - Demander de ne pas tenir compte du champ lors du mapping
- **@OneToMany, @ManyToOne**
 - Pour décrire une association de type un à plusieurs et plusieurs à un
- **@JoinColumn**
 - Pour décrire une clé étrangère dans une table
- **@ManyToMany**
 - Pour décrire une association plusieurs à plusieurs
- **Etc...**

Mapping des entités en utilisant les annotations JPA

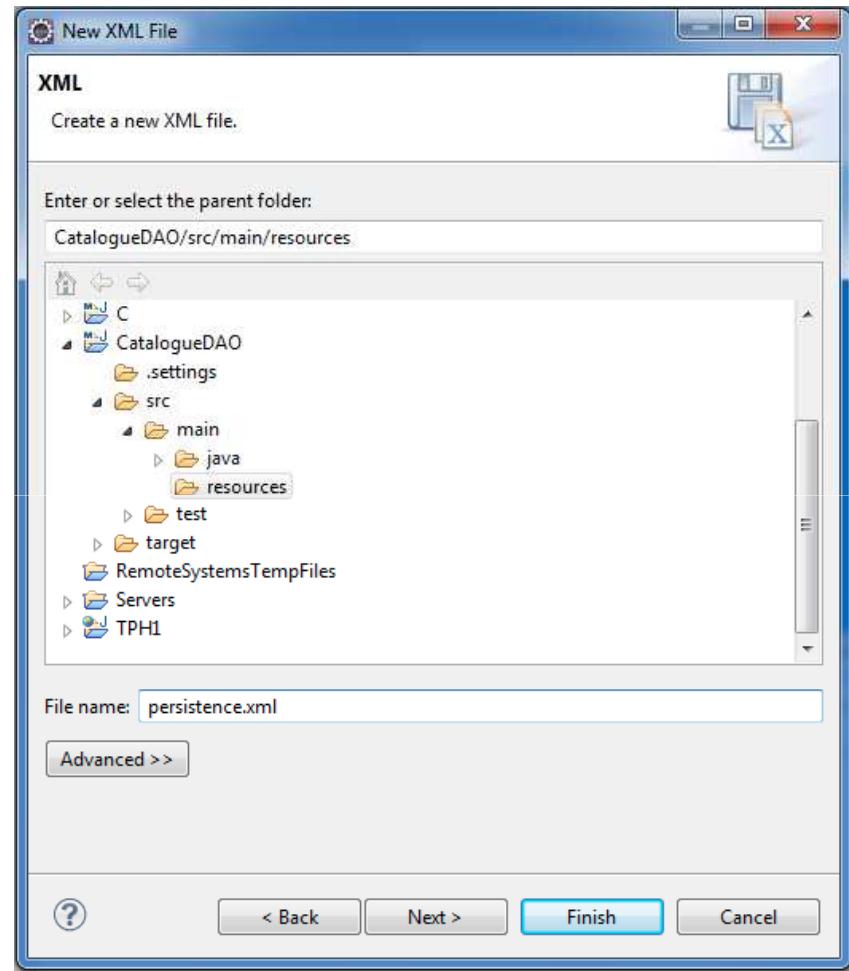
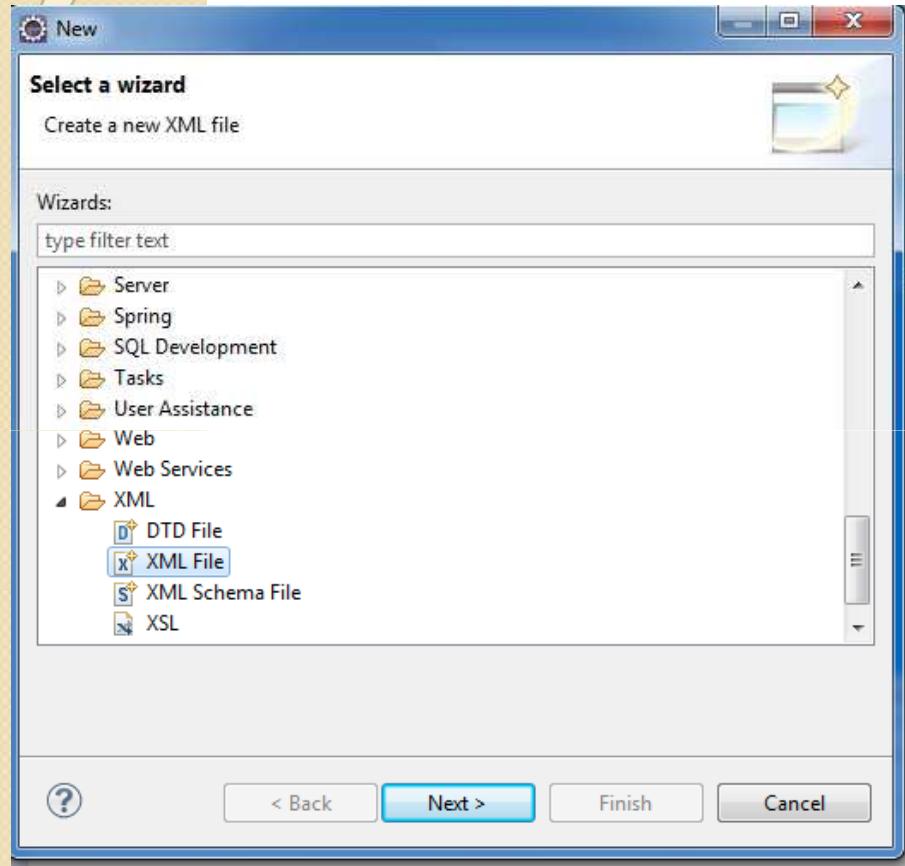
```
package org.bp.dao.entities;import java.io.Serializable;import java.util.Collection;
import javax.persistence.*; import org.hibernate.validator.constraints.NotEmpty;
@Entity
@Table(name="CATEGORIES")
public class Categorie implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="CODE_CAT")
    private Long codeCategorie;
    @NotEmpty
    private String nomCategorie;
    @Lob
    private byte[] photo;
    @OneToMany(mappedBy="categorie",fetch=FetchType.LAZY)
    private Collection<Produit> produits;
    // Constructeurs
    public Categorie() { }
    public Categorie(String nomCategorie) { this.nomCategorie = nomCategorie; }
    // Getters et Setters
}
```

Mapping des entités en utilisant les annotations JPA

```
package org.bp.dao.entities; import java.io.Serializable;
import javax.persistence.*;import javax.validation.constraints.*;
import org.hibernate.validator.constraints.NotEmpty;
@Entity
public class Produit implements Serializable {
    @Id
    @NotEmpty @Size(min=4,max=12)
    private String reference;
    @NotEmpty
    private String designation;
    @DecimalMin(value="10")
    private double prix;
    @Min(1)
    private int quantite;
    private boolean disponible;
    @ManyToOne @JoinColumn(name="CODE_CAT")
    private Categorie categorie;
    public Produit() {disponible=true;}
    public Produit(String ref, String des, double prix, int q) {
        this.reference = ref;this.designation = des;this.prix = prix;this.quantite =q;
        this.disponible=true;}
    // Getters et Setters }
```

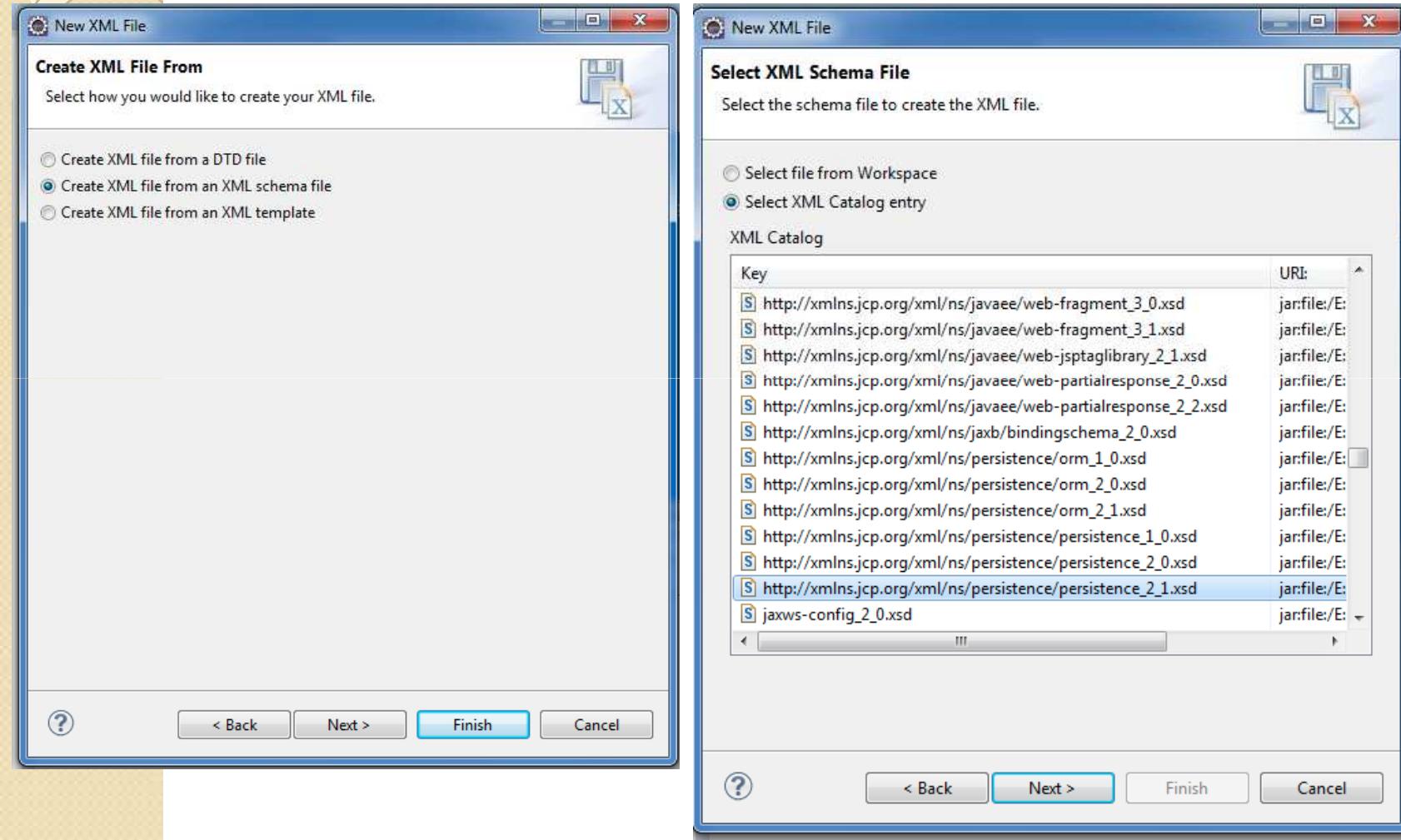
Unité de persistance : persistence.xml

- Création du fichier persistence.xml



Unité de persistance : persistence.xml

- Création du fichier persistence.xml



src/main/resources/META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd ">
    <persistence-unit name="UP_CAT" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
            <property name="hibernate.connection.username" value="root"/>
            <property name="hibernate.connection.password" value="" />
            <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/DB_CAT_BP"/>
        </properties>
    </persistence-unit>
</persistence>
```

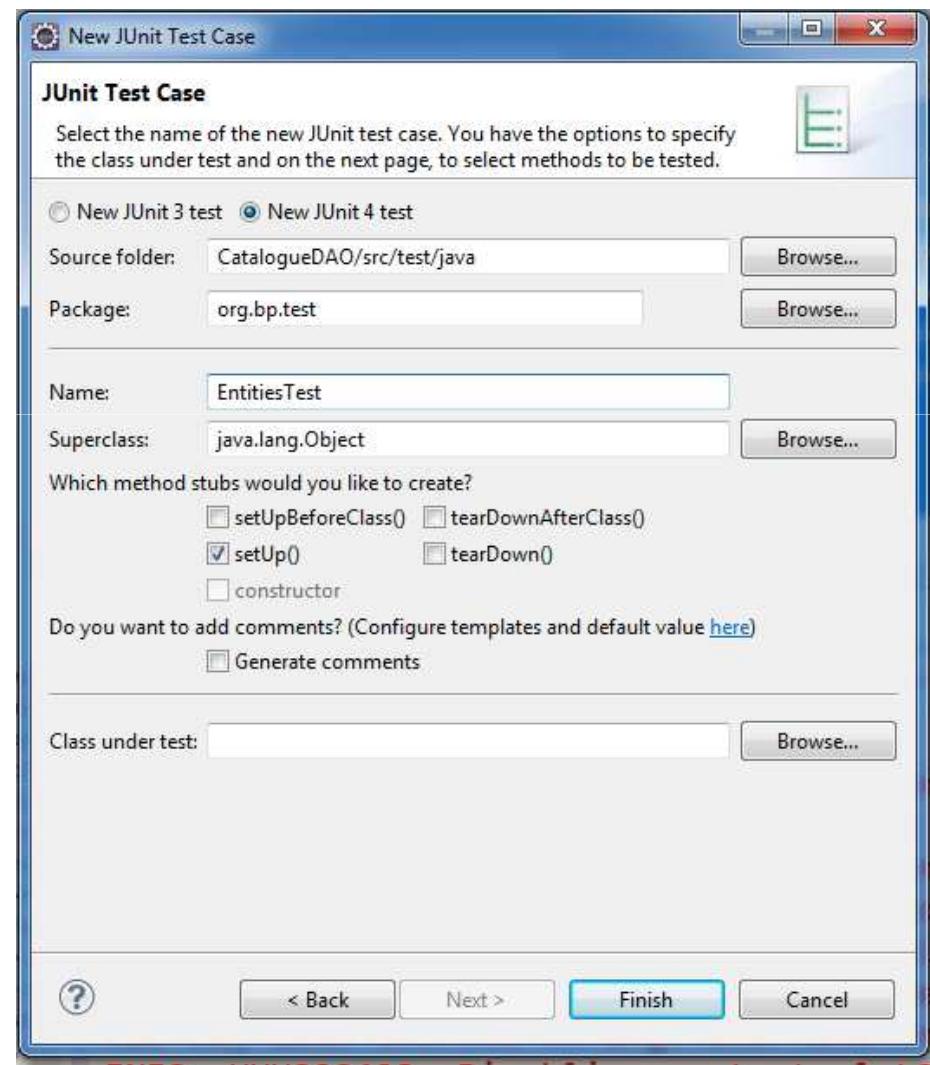
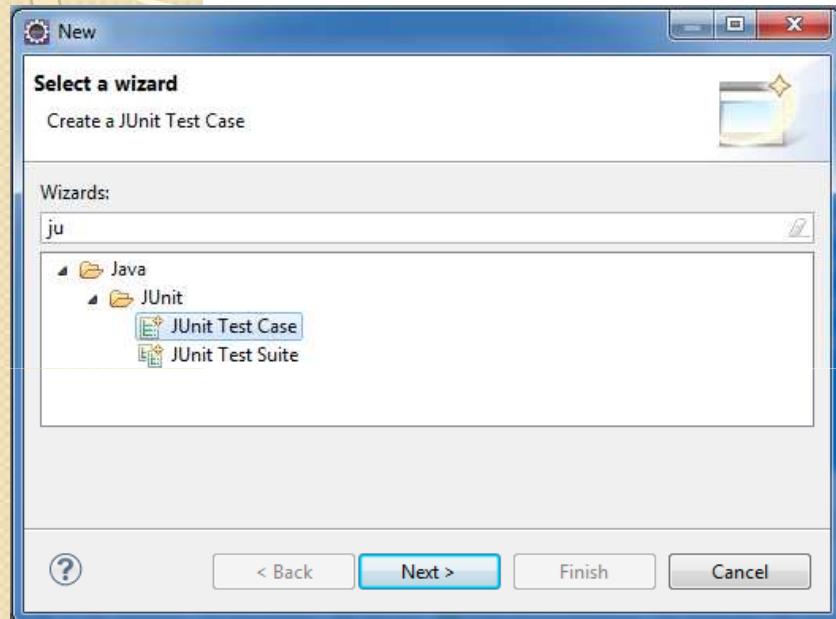


Tester les entités

- Avant d'implémenter les traitements, nous allons tester si les entités sont bien annotées et que l'unité de persistance est bien configurée.
- Nous allons créer un test unitaire qui permet de :
 - Créer un objet de type EntityManagerFactory qui va se charger de lire le fichier persistence.xml et de configurer l'unité de persistance.
 - Le succès de ce test devrait permettre de générer les tables produits et catégories dans la base de données.

Créer un Test JUNIT

- Créer un nouveau JUnit Test



EntitiesTest.java

```
package org.bp.test; import static org.junit.Assert.*;  
import javax.persistence.EntityManagerFactory; import javax.persistence.Persistence;  
import org.junit.Before; import org.junit.Test;  
public class EntitiesTest {  
    private EntityManagerFactory entityManagerFactory;  
    @Before  
    public void setUp() throws Exception {  
    }  
    @Test  
    public void testEntities() {  
        try {  
            EntityManagerFactory=Persistence.createEntityManagerFactory("UP_CAT");  
            EntityManager entityManager=entityManagerFactory.createEntityManager();  
            assertTrue(true);  
        } catch (Exception e) {  
            fail(e.getMessage());  
            e.printStackTrace();  
        }  
    }  
}
```

Après Exécution du Test

- Les messages suivants s'affichent dans la console :

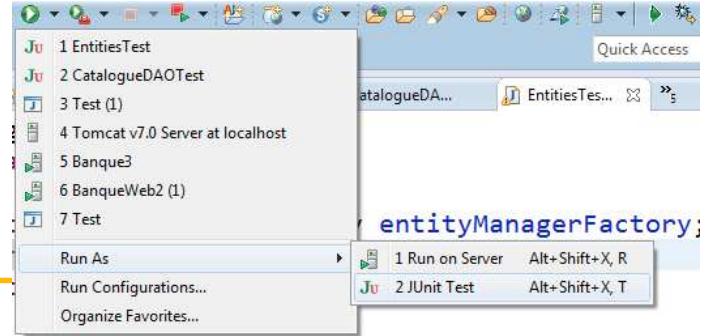
```
Hibernate: drop table if exists CATEGORIES
Hibernate: drop table if exists Produit
Hibernate: create table CATEGORIES (CODE_CAT bigint not null auto_increment, nomCategorie varchar(255), primary key (CODE_CAT))
Hibernate: create table Produit (reference varchar(255) not null, designation varchar(255), disponible bit not null, prix double precision not null, quantite integer not null, CODE_CAT bigint, primary key (reference))
Hibernate: alter table Produit add constraint FK_5pst292t2rsnfynx7cs418q foreign key (CODE_CAT) references CATEGORIES (CODE_CAT)
mai 01, 2014 2:03:40 PM org.hibernate.tool.hbm2ddl.SchemaExport execute
INFO: HHH000230: Schema export complete
```

- Les tables produits et categories sont générées

Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra	
<u>CODE_CAT</u>	bigint(20)			Non	Aucun	AUTO_INCREMENT	
nomCategorie	varchar(255)	latin1_swedish_ci		Oui	NULL		
photo	longblob		BINARY	Oui	NULL		

categories

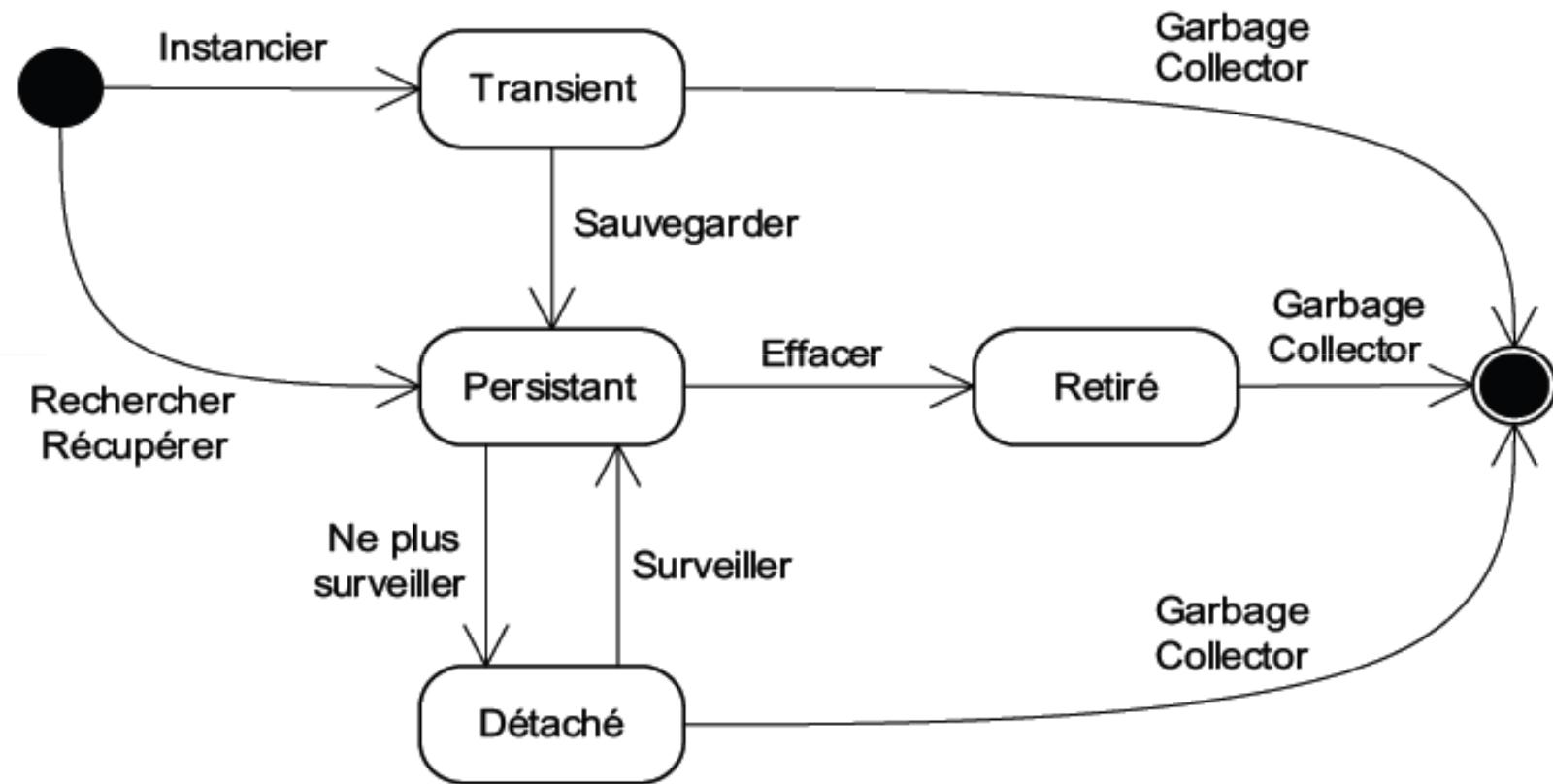
Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra
<u>reference</u>	varchar(255)	latin1_swedish_ci		Non	Aucun	
designation	varchar(255)	latin1_swedish_ci		Oui	NULL	
disponible	bit(1)			Non	Aucun	
prix	double			Non	Aucun	
quantite	int(11)			Non	Aucun	
CODE_CAT	bigint(20)			Oui	NULL	



Gestion des entités par EntityManager

- EntityManager est une interface définie dans JPA.
- Chaque framework ORM possède sa propre implémentation de cette interface.
- EntityManager définit les méthodes qui permettent de gérer le cycle de vie de la persistance des Entity.
 - La méthode persist() permet rendre une nouvelle instance d'un EJB Entity persistante. Ce qui permet de sauvegarder son état dans la base de données
 - La méthode find() permet de charger une entité sachant sa clé primaire.
 - La méthode createQuery() permet de créer une requête EJBQL qui permet de charger une liste d'entités selon des critères.
 - La méthode remove() permet de programmer une entité persistante pour la suppression.
 - La méthode merge() permet de rendre une entité détachée persistante.

Cycle de vie d'un EJB Entity



Objet Persistant

- Un objet *persistent* est un objet qui possède son image dans le datastore et dont la durée de vie est potentiellement infinie.
- Pour garantir que les modifications apportées à un objet sont rendues persistantes, c'est-à-dire sauvegardées, l'objet est surveillé par un « traqueur » d'instances persistantes.
- Ce rôle est joué par le gestionnaire d'entités.

Etat Transient

- Un objet *transient* est un objet qui n'a pas son image stockée dans le datastore.
- Il s'agit d'un objet « temporaire », qui meurt lorsqu'il n'est plus utilisé par personne. En Java, le garbage collector le ramasse lorsque aucun autre objet ne le référence.

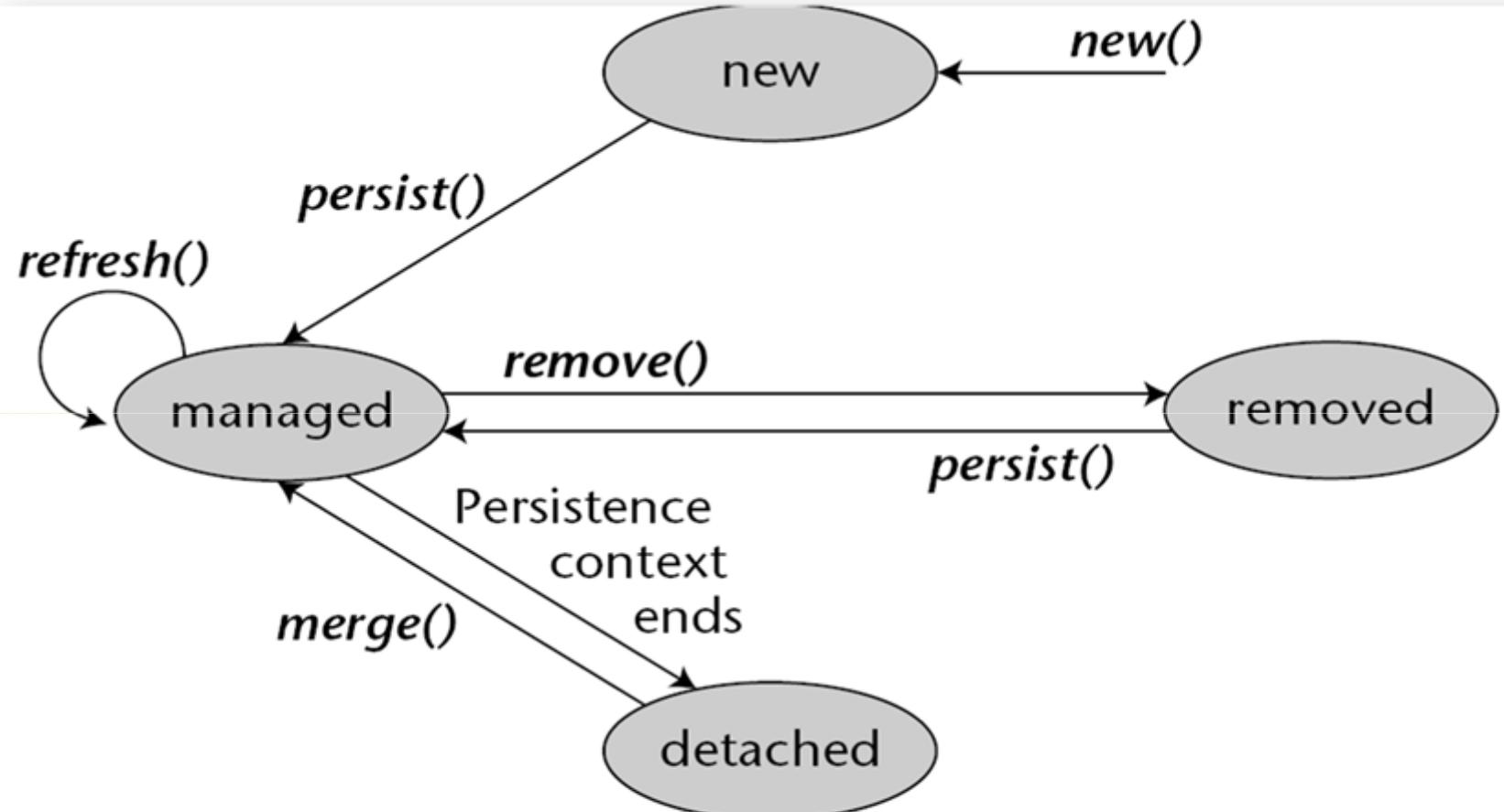
Etat Détaché

- Un objet détaché est un objet qui possède son image dans le datastore mais qui échappe temporairement à la surveillance opérée par le gestionnaire d'entités.
- Pour que les modifications potentiellement apportées pendant cette phase de détachement soient enregistrées, il faut effectuer une opération manuelle pour *merger* cette instance au gestionnaire d'entités.

Etat Retiré

- Un objet *retiré* est un objet actuellement géré par le gestionnaire d'entités mais programmé pour ne plus être persistant.
- À la validation de l'unité de travail, un ordre SQL delete sera exécuté pour retirer son image du datastore.

Cycle de vie d'un EJB Entity



Interface ICatalogueDAO

```
package org.bp.dao;
import java.util.List;
import org.bp.dao.entities.Categorie;
import org.bp.dao.entities.Produit;
public interface ICatalogueDAO {
    public void addCategorie(Categorie c);
    public void addProduit(Produit p, Long codeCat);
    public List<Categorie> listCategories();
    public List<Produit> produitsParCat(Long codeCat);
    public List<Produit> produitsParMC(String mc);
    public Produit getProduit(String ref);
    public void updateProduit(Produit p);
    public void deleteProduit(String ref);
}
```

Implémentation JPA de la couche DAO

```
package org.bp.dao;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import org.bp.dao.entities.Categorie;
import org.bp.dao.entities.Produit;
public class CatalogueDaoImpl implements ICatalogueDAO {
    @PersistenceContext(unitName="UP_CAT")
    private EntityManager em;
    public void addCategorie(Categorie c) {
        em.persist(c);
    }
    public void addProduit(Produit p, Long codeCat) {
        Categorie c=em.find(Categorie.class,codeCat);
        p.setCategorie(c);
        em.persist(p);
    }
}
```

Implémentation JPA de la couche DAO

```
public List<Categorie> listCategories() {  
    Query req=em.createQuery("select c from Categorie c");  
    return req.getResultList();  
}  
  
public List<Produit> produitsParCat(Long codeCat) {  
    Query req=em.createQuery("select p from Produit p where  
        p.categorie.codeCategorie=:x");  
    req.setParameter("x", codeCat);  
    return req.getResultList();  
}  
  
public List<Produit> produitsParMC(String mc) {  
    Query req=em.createQuery("select p from Produit p where p.designation  
        like:x");  
    req.setParameter("x", "%" +mc+ "%");  
    return req.getResultList();  
}
```

Implémentation JPA de la couche DAO

```
public Produit getProduit(String ref) {  
    Produit p=em.find(Produit.class, ref);  
    return p;  
}  
  
public void updateProduit(Produit p) {  
    em.merge(p);  
}  
  
public void deleteProduit(String ref) {  
    Produit p=getProduit(ref);  
    em.remove(p);  
}  
  
public EntityManager getEm() {  
    return em;  
}  
  
public void setEm(EntityManager em) {  
    this.em = em;  
}  
}
```

Tester l'implémentation JPA

```
package org.bp.test; import static org.junit.Assert.*; import java.util.List;
import javax.persistence.EntityManager; import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import org.bp.dao.CatalogueDaoImpl; import org.bp.dao.ICatalogueDAO; import org.bp.dao.entities.Categorie;
import org.junit.Before;import org.junit.Test;

public class CatalogueDAOTest {
    private EntityManagerFactory entityManagerFactory;
    private CatalogueDaoImpl dao;
    @Before
    public void setUp() throws Exception {
        entityManagerFactory=Persistence.createEntityManagerFactory("UP_CAT");
        EntityManager entityManager=entityManagerFactory.createEntityManager();
        dao=new CatalogueDaoImpl();
        dao.setEm(entityManager);
    }
}
```

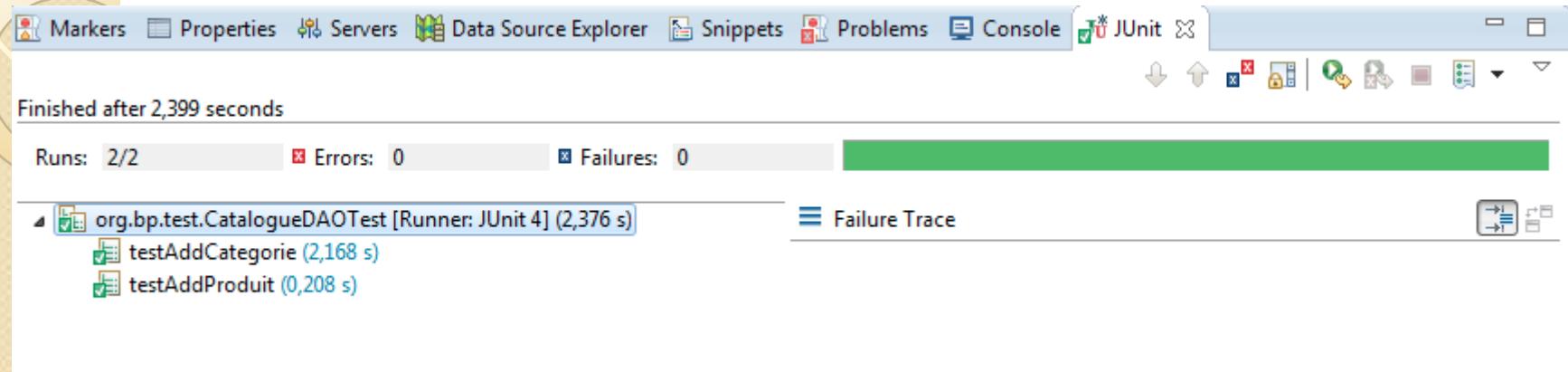
Tester L'ajout des catégories

```
@Test  
public void testAddCategorie() {  
    try{  
        dao.getEm().getTransaction().begin();  
        dao.addCategorie(new Categorie("Ordinateurs"));  
        dao.addCategorie(new Categorie("Imprimantes"));  
        List<Categorie> categories=dao.listCategories();  
        dao.getEm().getTransaction().commit();  
        assertTrue(categories.size()==2);  
    }  
    catch(Exception e){  
        dao.getEm().getTransaction().rollback();  
        fail(e.getMessage());  
        e.printStackTrace();  
    }  
}
```

Tester L'ajout des produits

```
@Test  
public void testAddProduit() {  
    try{  
        dao.getEm().getTransaction().begin();  
        List<Produit> prods1=dao.produitsParCat(1L);  
        dao.addProduit(new Produit("HP765", "Ordinateur HP 765", 9800, 34), 1L);  
        dao.addProduit(new Produit("HP860", "Ordinateur HP 860", 3200, 10), 1L);  
        dao.addProduit(new Produit("AT23", "IMprimante AT18", 1200, 11), 2L);  
        List<Produit> prods2=dao.produitsParCat(1L);  
        dao.getEm().getTransaction().commit();  
        assertTrue(prods2.size()==prods1.size()+2);  
    }  
    catch(Exception e){  
        dao.getEm().getTransaction().rollback();  
        fail(e.getMessage());  
        e.printStackTrace();  
    }  
}
```

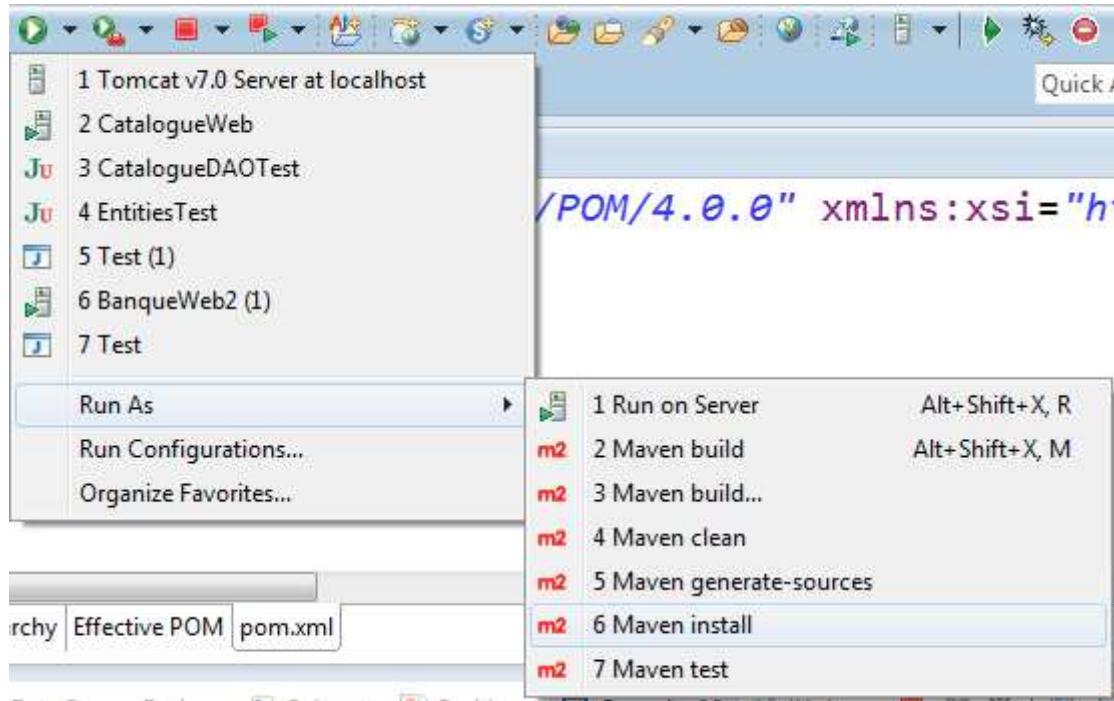
Exécution des Tests Unitaires



CODE_CAT	nomCategorie
1	Ordinateurs
2	Imprimantes

reference	designation	disponible	prix	quantite	CODE_CAT
AT23	IMprimante AT18	1	1200	11	2
HP765	Ordinateur HP 765	1	9800	34	1
HP860	Ordinateur HP 860	1	3200	10	1

Installation du projet dans le repository



- Le Fichier : CatalogueDAO-0.0.1-SNAPSHOT.jar est généré et placé dans le repository de maven

C:\Users\youssf\m2\repository\org\bp\CatalogueDAO\0.0.1-SNAPSHOT				
Inclure dans la bibliothèque		Partager avec	Graver	Nouveau dossier
Nom	Modifié le	Type	Taille	
_remote.repositories	02/05/2014 08:25	Fichier REPOSITORY	1 Ko	
CatalogueDAO-0.0.1-SNAPSHOT.jar	02/05/2014 08:25	Executable Jar File	8 Ko	
CatalogueDAO-0.0.1-SNAPSHOT.pom	02/05/2014 07:26	Fichier POM	2 Ko	
maven-metadata-local.xml	02/05/2014 08:25	Fichier XML	1 Ko	



Gérer les associations et l'héritage entre les entités

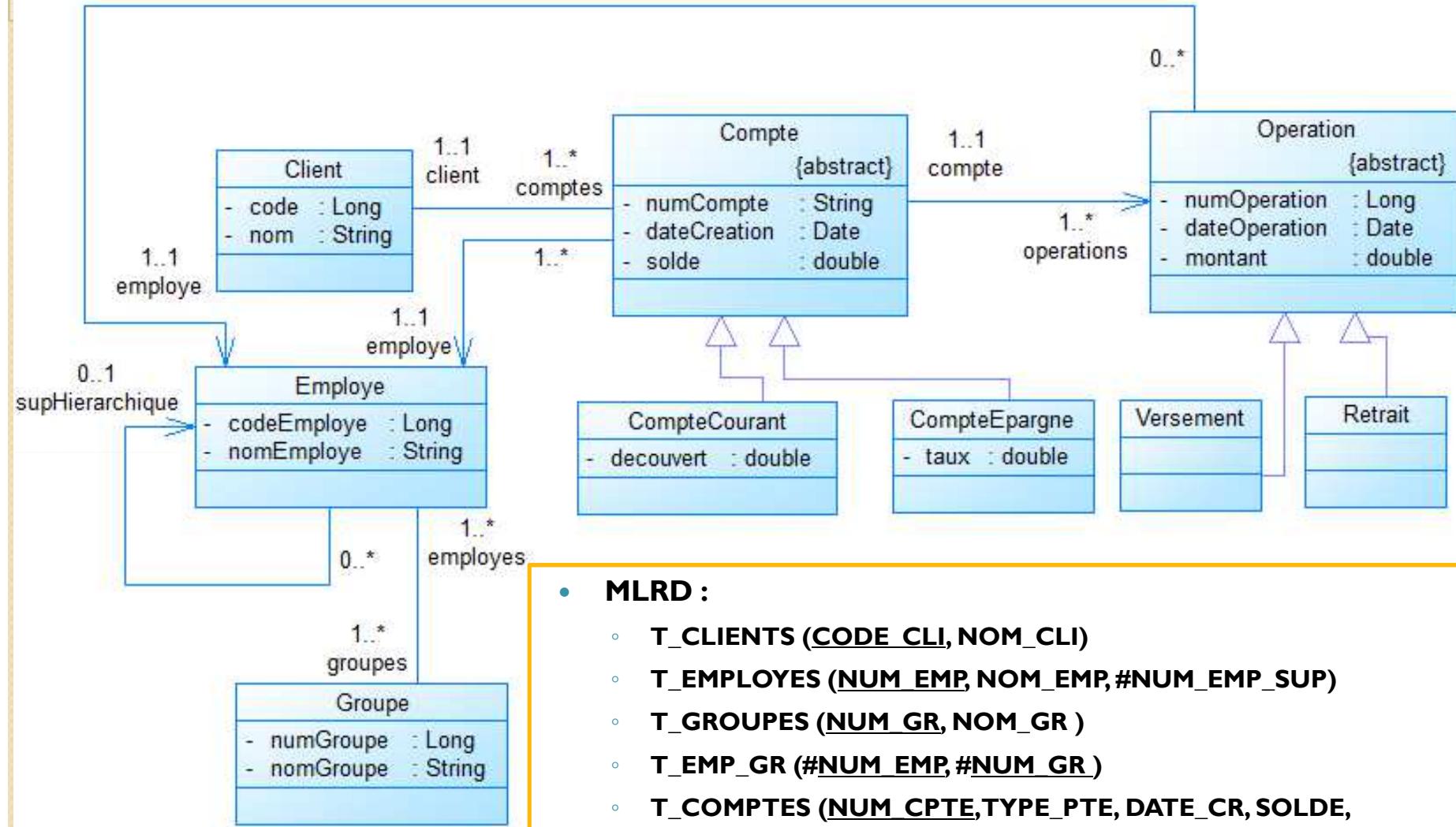
- Associations
 - **@OneToMany**
 - **@ManyToOne**
 - **@ManyToMany**
 - **@OneToOne**
- Héritage
 - Une table par hiérarchie
 - Une table pour chaque classe concrète
 - Une table pour la classe parente et une table pour chaque classe fille



Exemple de problème

- On souhaite créer une application qui permet de gérer des comptes bancaire.
 - Chaque compte est défini un numéro, un solde et une date de création
 - Un compte courant est un compte qui possède en plus un découvert
 - Un compte épargne est un compte qui possède en plus un taux d'intérêt.
 - Chaque compte appartient à un client et créé par un employé.
 - Chaque client est défini par son code et son nom
 - Un employé est défini par son code et sont solde.
 - Chaque employé possède un supérieur hiérarchique.
 - Chaque employé peut appartenir à plusieurs groupes
 - Chaque groupe, défini par un code est un nom, peut contenir plusieurs employés.
 - Chaque compte peut subir plusieurs opérations.
 - Il existe deux types d'opérations : Versement et Retrait
 - Chaque opération est effectuée par un employé.
 - Une opération est définie par un numéro, une date et un montant.

Diagramme de classes et MLDR



- **MLRD :**
- **T_CLIENTS (CODE_CLI, NOM_CLI)**
- **T_EMPLOYES (NUM_EMP, NOM_EMP, #NUM_EMP_SUP)**
- **T_GROUPES (NUM_GR, NOM_GR)**
- **T_EMP_GR (#NUM_EMP, #NUM_GR)**
- **T_COMPTES (NUM_CPT, TYPE_PTE, DATE_CR, SOLDE, #NUM_EMP, #CODE_CLI)**
- **T_OPERATIONS (NUM_OP, TYPE_OP, DATE_OP, MONTANT, #NUM_EMP, #NUM_CPT)**

Entity Client

```
package banque.metier;
import java.io.Serializable; import java.util.Collection;
import javax.persistence.*;
@Entity
@Table(name="CLIENTS")
public class Client implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="CODE_CLI")
    private Long codeClient;
    @Column(name="NOM_CLI")
    private String nomClient;
    @OneToMany(mappedBy="client",fetch=FetchType.LAZY)
    ,cascade=CascadeType.ALL
    private Collection<Compte> comptes;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity Employe

```
package banque.metier;
import java.io.Serializable; import java.util.Collection;
import javax.persistence.*;
@Entity
public class Employe implements Serializable{
    @Id
    @GeneratedValue
    private Long numEmploye;
    private String nomEmploye;
    private double salaire;
    @ManyToOne
    @JoinColumn(name="NUM_EMP_SUP")
    private Employe supHierarchique;
    @ManyToMany
    @JoinTable(name="EMP_GROUPES",joinColumns =
    @JoinColumn(name = "NUM_EMP"),
    inverseJoinColumns = @JoinColumn(name = "NUM_GROUPE"))
    private Collection<Groupe> groupes;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity Groupe

```
package banque.metier;  
import java.io.Serializable; import  
java.util.Collection;  
import javax.persistence.*;  
@Entity  
public class Groupe implements Serializable {  
    @Id  
    @GeneratedValue  
    private Long numGroupe;  
    private String nomGroupe;  
    @ManyToMany(mappedBy="groupes")  
    private Collection<Employe> employes;  
    // Getters et Setters  
    // Constructeur sans param et avec params  
}
```

Entity Compte

```
package banque.metier;
import java.io.Serializable; import java.util.Collection;
import javax.persistence.*;
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TYPE_CPTE",discriminatorType=DiscriminatorType.STRING,length=2)
public abstract class Compte implements Serializable {
    @Id
    private String numCompte;
    private Date dateCreation;
    private double solde;
    @ManyToOne
    @JoinColumn(name="CODE_CLI")
    private Client client;
    @ManyToOne
    @JoinColumn(name="NUM_EMP")
    private Employe employe;
    @OneToMany(mappedBy="compte")
    private Collection<Operation> operations;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity CompteCourant

```
package banque.metier;
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.*;
@Entity
@DiscriminatorValue("CC")
public class CompteCourant extends Compte{
    private double decouvert;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity CompteEpargne

```
package banque.metier;
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.*;
@Entity
@DiscriminatorValue("CE")
public class CompteEpargne extends Compte {
private double taux;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity Operation

```
package banque.metier;
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.*;
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TYPE_OP",discriminatorType=DiscriminatorType.
    STRING,length=2)
public abstract class Operation implements Serializable {
    @Id
    @GeneratedValue
    private Long numOperation;
    private Date dateOperation;
    private double montant;
    @ManyToOne
    @JoinColumn(name="NUM_CPTE")
    private Compte compte;
    @ManyToOne
    @JoinColumn(name="NUM_EMP")
    private Employe employe;
    // Getters et Setters
    // Constructeur sans param et avec params
}
```

Entity Versement

```
package banque.metier;  
import java.io.Serializable;  
import java.util.Collection;  
import javax.persistence.*;  
  
@Entity  
@DiscriminatorValue("V")  
public class Versement extends Operation{  
    // Constructeur sans param et avec params  
}
```

Entity Retrait

```
package banque.metier;  
import java.io.Serializable;  
import java.util.Collection;  
import javax.persistence.*;  
  
@Entity  
@DiscriminatorValue("R")  
public class Retrait extends Operation {  
    // Constructeur sans param et avec params  
}
```



Interface DAO

```
package banque.metier.session;
import java.util.List;
import javax.ejb.Remote;
import banque.metier.*;
public interface BanqueDAO {
    public void addClient(Client c);
    public void addEmploye(Employe e,Long numEmpSup);
    public void addGroupe(Groupe g);
    public void addEmployeToGroupe(Long idGroupe,Long idEmp);
    public void addCompte(Compte c,Long numCli,Long numEmp );
    public void addOperation(Operation op,String numCpte,Long numEmp);
    public Compte consulterCompte(String numCpte);
    public List<Client> consulterClientsParNom(String mc);
    public List<Client> consulterClients();
    public List<Groupe> consulterGroupes();
    public List<Employe> consulterEmployes();
    public List<Employe> consulterEmployesParGroupe(Long idG);
    public Employe consulterEmploye(Long idEmp);
}
```

Implémentation JPA

```
package banque.metier.session;
import java.util.List;import javax.ejb.Stateless;
import javax.persistence.*; import banque.metier.*;
public class BanqueDAOImpl implements BanqueDAO {
    private EntityManager em;
    @Override
    public void addClient(Client c) {
        em.persist(c);
    }
    @Override
    public void addEmploye(Employe e, Long numEmpSup) {
        Employe empSup;
        if(numEmpSup!=null){
            empSup=em.find(Employe.class, numEmpSup);
            e.setSupHierarchique(empSup);
        }
        em.persist(e);
    }
}
```

Implémentation JPA

```
@Override  
public void addGroupe(Groupe g) {  
    em.persist(g);  
}  
  
@Override  
public void addEmployeToGroupe(Long idGroupe, Long idEmp) {  
    Employe emp=em.find(Employe.class, idEmp);  
    Groupe g=em.find(Groupe.class, idGroupe);  
    emp.getGroupes().add(g);  
    g.getEmployes().add(emp);  
}  
  
@Override  
public void addCompte(Compte c, Long numCli, Long numEmp) {  
    Client cli=em.find(Client.class, numCli);  
    Employe e=em.find(Employe.class, numEmp);  
    c.setClient(cli);  
    c.setEmploye(e);  
    em.persist(c);  
}
```

Implémentation JPA

```
@Override  
public void addOperation(Operation op, String numCpte, Long  
    numEmp) {  
    Compte c=em.find(Compte.class, numCpte);  
    Employe emp=em.find(Employe.class, numEmp);  
    op.setEmploye(emp);  
    op.setCompte(c);  
    em.persist(op);  
}  
  
@Override  
public Compte consulterCompte(String numCpte) {  
    Compte cpte=em.find(Compte.class, numCpte);  
    if(cpte==null) throw new RuntimeException("Compte "+numCpte+"  
        n'existe pas");  
    cpte.getOperations().size();  
    return cpte;  
}
```

Implémentation JPA

```
@Override  
public List<Client> consulterClientsParNom(String mc) {  
    Query req=em.createQuery("select c from Client c where c.nom like  
        :mc");  
    req.setParameter("mc", "%" + mc + "%");  
    return req.getResultList();  
}  
  
@Override  
public List<Client> consulterClients() {  
    Query req=em.createQuery("select c from Client c");  
    return req.getResultList();  
}  
  
@Override  
public List<Groupe> consulterGroupes() {  
    Query req=em.createQuery("select g from Groupe g");  
    return req.getResultList();  
}
```

Implémentation JPA

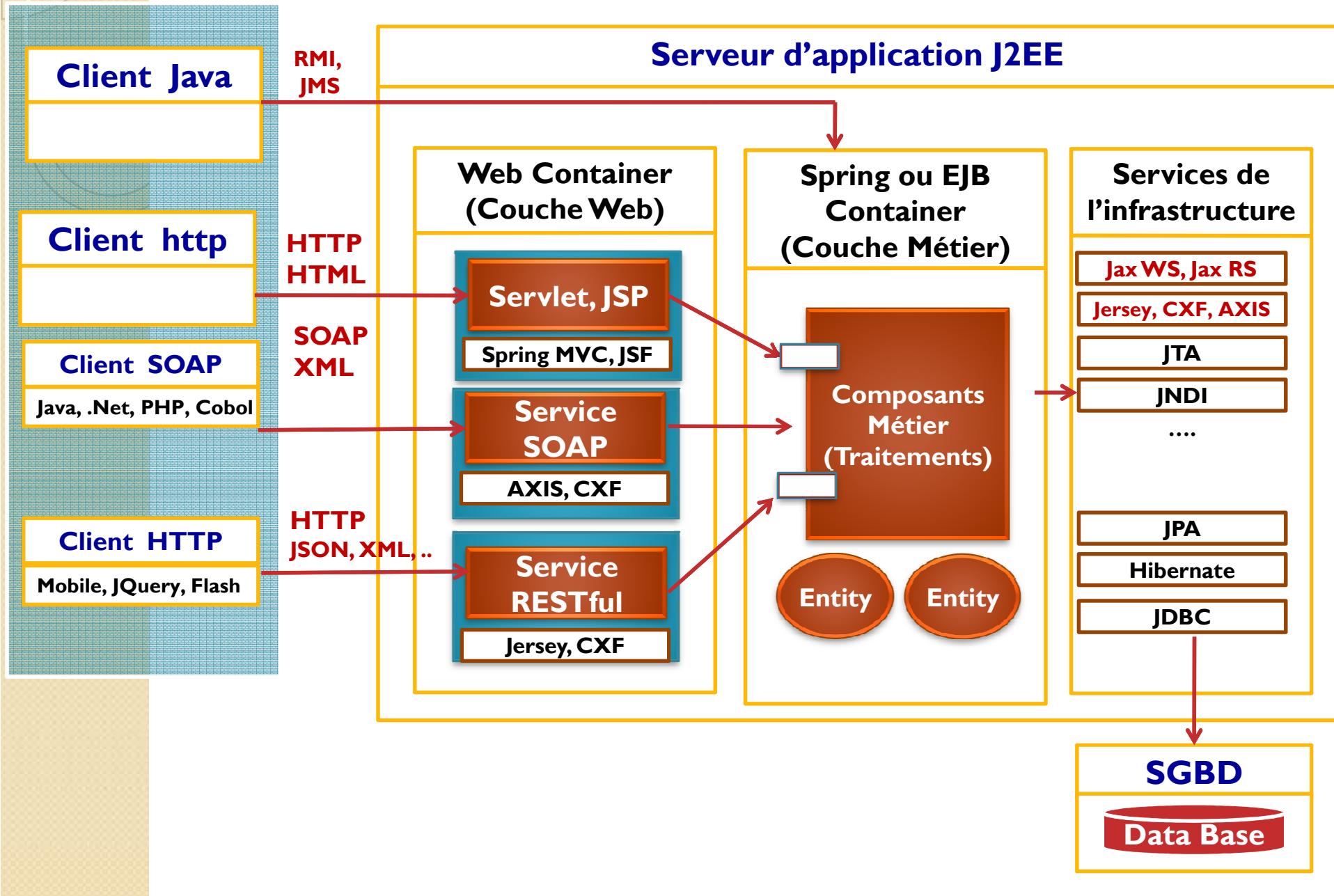
```
@Override  
public List<Employe> consulterEmployes() {  
    Query req=em.createQuery("select eg from Employe e");  
    return req.getResultList();  
}  
  
@Override  
public List<Employe> consulterEmployesParGroupe(Long idG) {  
    Query req=em.createQuery("select e from Employe e where  
        e.groupes.numGroupe=:x");  
    req.setParameter("x", idG);  
    return req.getResultList();  
}  
  
@Override  
public Employe consulterEmploye(Long idEmp) {  
    Employe e=em.find(Employe.class,idEmp);  
    if(e==null) throw new RuntimeException("Employe "+idEmp+"  
        n'existe pas");  
    return e;  
}  
}
```



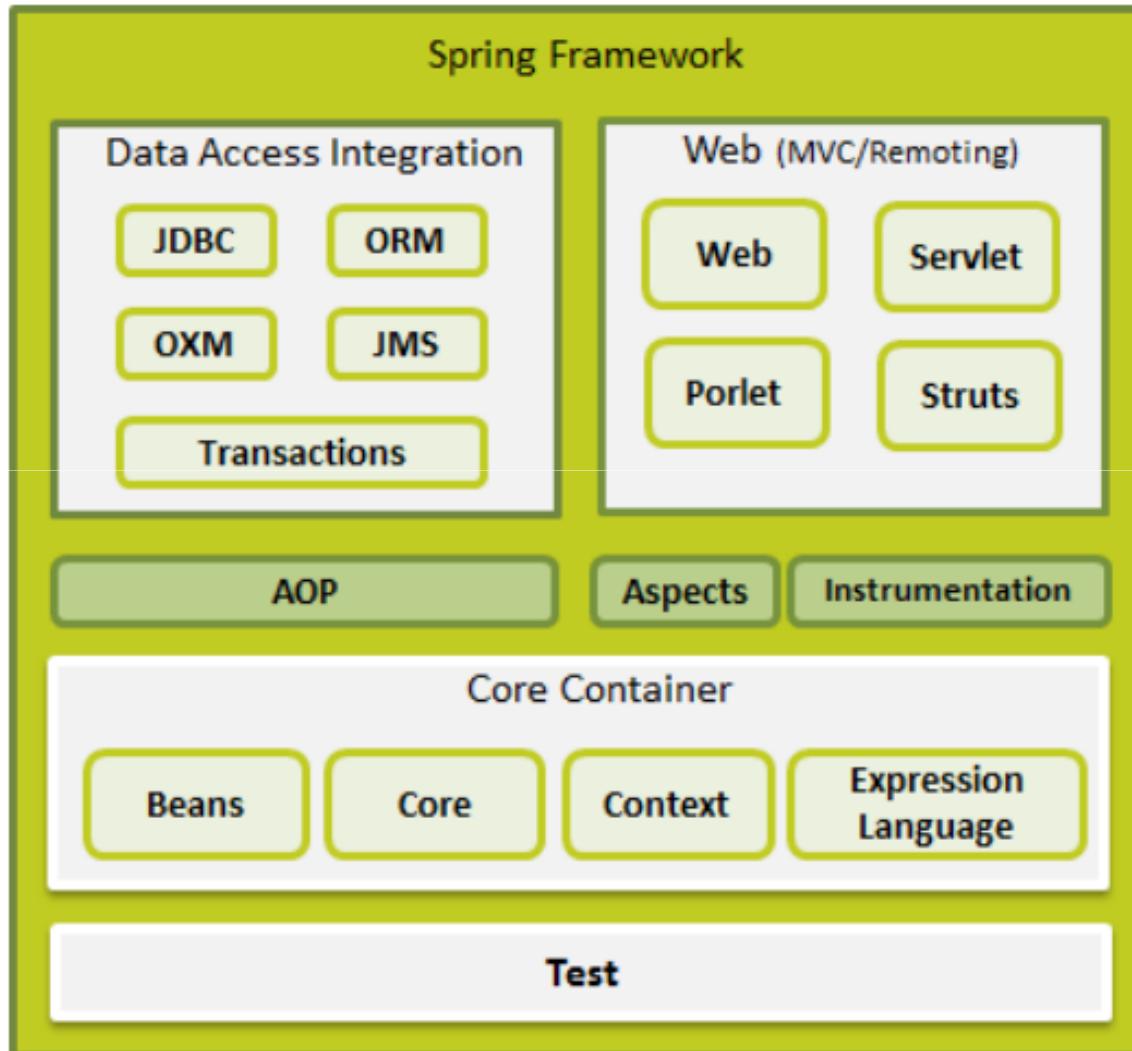
SPRING FRAMEWORK

med@youssf.net

Architecture J2EE



Spring Framework Architecture





Spring Framework Architecture

- Spring est modulaire , permettant de choisir les modules appropriés à votre application, sans être obligé d'utiliser le reste.
- Spring Framework fournit plus de 20 modules qui peuvent être utilisé dans els applications.



Core Container

- The Core Container consists of the Core, Beans, Context, and Expression Language modules whose detail is as follows:
 - The **Core** module provides the fundamental parts of the framework, including the **IoC** and Dependency Injection features.
 - The **Bean** module provides **BeanFactory** which is a sophisticated implementation of the factory pattern.
 - The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The **ApplicationContext** interface is the focal point of the Context module.
 - The **Expression Language** module provides a powerful expression language for querying and manipulating an object graph at runtime.



Data Access/Integration

- The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows:
 - The **JDBC** module provides a JDBC-abstraction layer that removes the need to do tedious JDBC related coding.
 - The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
 - The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
 - The Java Messaging Service **JMS** module contains features for producing and consuming messages.
 - The **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.



Web

- The Web layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules whose detail is as follows:
 - The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
 - The **Web-Servlet** module contains Spring's model-view-controller (MVC) implementation for web applications
 - The **Web-Struts** module contains the support classes for integrating a classic Struts web tier within a Spring application.
 - The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.



Autres

- There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules whose detail is as follows:
 - The **AOP** module provides aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
 - The **Aspects** module provides integration with AspectJ which is again a powerful and mature aspect oriented programming (AOP) framework.
 - The **Instrumentation** module provides class instrumentation support and class loader implementations to be used in certain application servers.
 - The **Test** module supports the testing of Spring components with JUnit or TestNG frameworks.



Inversion de contrôle ou Injection de dépendances

Rappels de quelque principes de conception

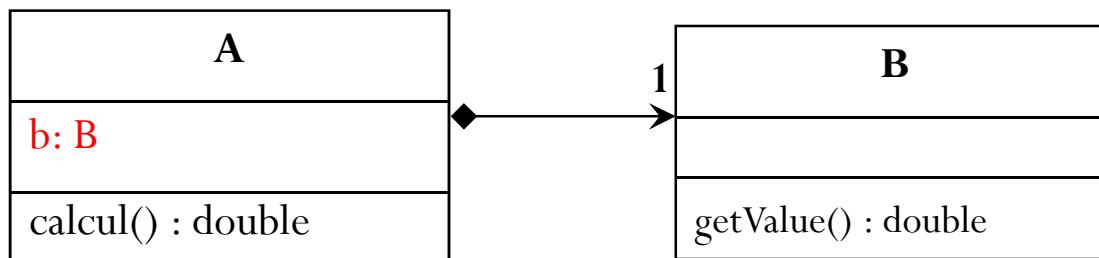
- Une application qui n'évolue pas meurt.
- Une application doit être fermée à la modification et ouverte à l'extension.
- Une application doit s'adapter aux changements
- Efforcez-vous à coupler faiblement vos classes.
- Programmer une interface et non une implémentation
- Etc..



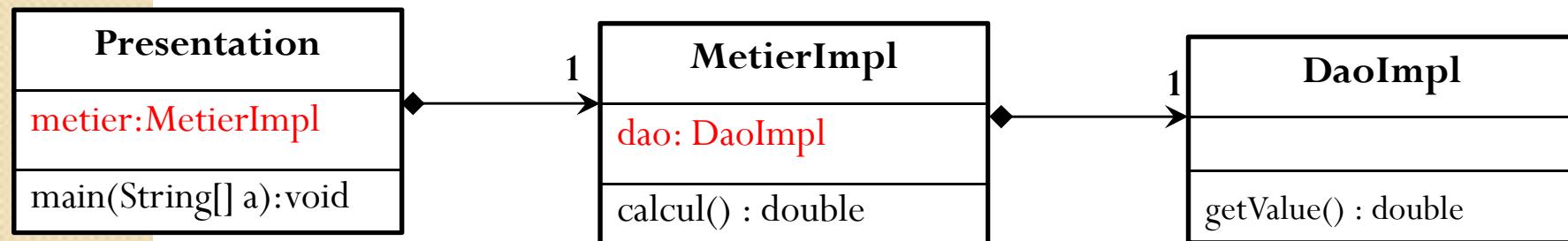
Couplage Fort et Couplage faible

Couplage fort

- Quand une classe A est lié à une classe B, on dit que la classe A est fortement couplée à la classe B.
- La classe A ne peut fonctionner qu'en présence de la classe B.
- Si une nouvelle version de la classe B (soit B2), est créée, on est obligé de modifier dans la classe A.
- Modifier une classe implique:
 - Il faut disposer du code source.
 - Il faut recompiler, déployer et distribuer la nouvelle application aux clients.
 - Ce qui engendre un cauchemar au niveau de la maintenance de l'application



Exemple de couplage fort



```
package metier;
import dao.DaoImpl;
public class MetierImpl {
    private DaoImpl dao;
    public MetierImpl() {
        dao=new DaoImpl();
    }
    public double calcul(){
        double nb=dao.getValue();
        return 2*nb;
    }
}
```

```
package dao;
public class DaoImpl {
    public double getValue(){
        return(5);
    }
}
```

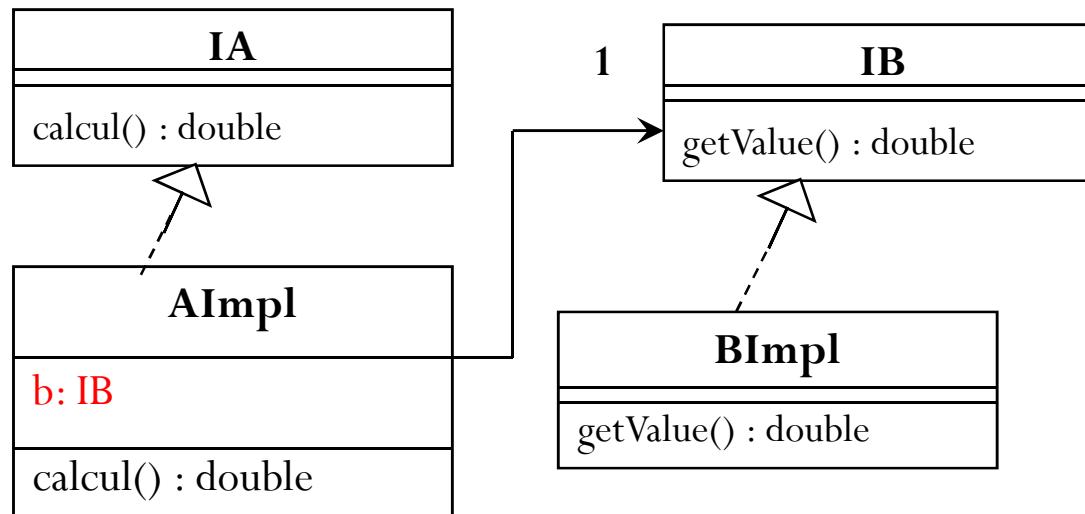
```
package pres;
import metier.MetierImpl;
public class Presentation {
    private static MetierImpl metier;
    public static void main(String[] args) {
        metier=new MetierImpl();
        System.out.println(metier.calcul());
    }
}
```

Problèmes du couplage fort

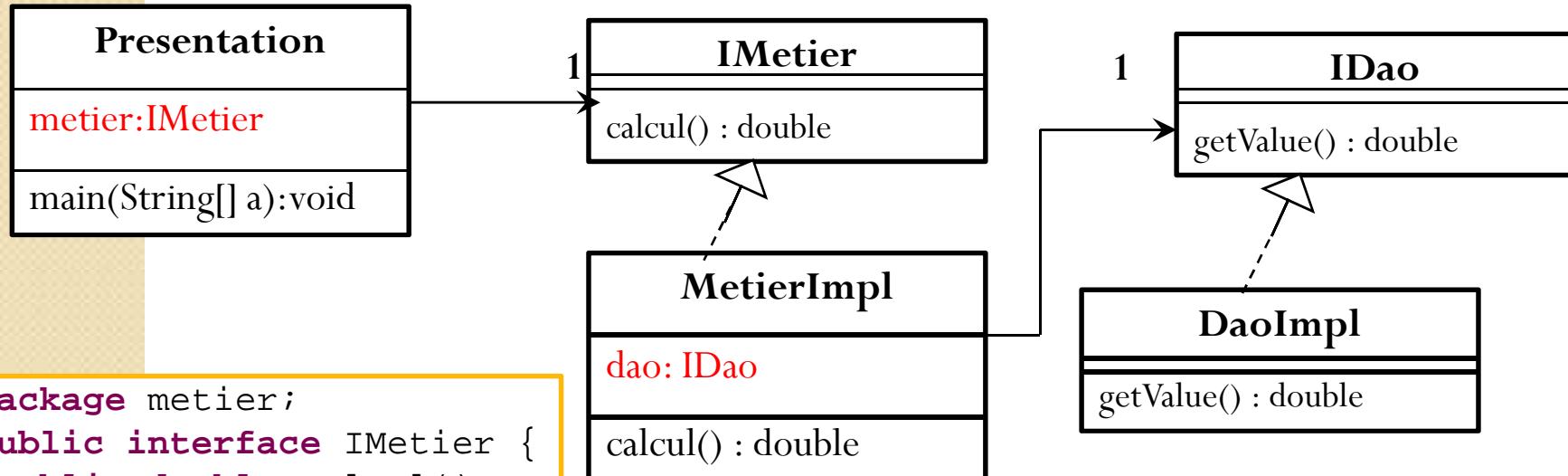
- Dans l'exemple précédent, les classes MetierImpl et Daolmpl sont liées par un couplage fort. De même pour les classe Presentation et MetierImpl
- Ce couplage fort n'a pas empêché de résoudre le problème au niveau fonctionnel.
- Mais cette conception nous ne a pas permis de créer une application fermée à la modification et ouverte à l'extension.
- En effet, la création d'une nouvelle version de la méthode getValue() de la classe Daolmpl, va nous obliger d'éditer le code source de l'application aussi bien au niveau de Daolmpl et aussi MetierImpl.
- De ce fait nous avons violé le principe « une application doit être fermée à la modification et ouverte à l'extension»
- Nous allons voir que nous pourrons faire mieux en utilisant le couplage faible.

Couplage Faible.

- Pour utiliser le couplage faible, nous devons utiliser les interfaces.
- Considérons une classe A qui implémente une interface IA, et une classe B qui implémente une interface IB.
- Si la classe A est liée à l'interface IB par une association, on dit que la classe A et la classe B sont liées par un couplage faible.
- Cela signifie que la classe B peut fonctionner avec n'importe quelle classe qui implémente l'interface IA.
- En effet la classe B ne connaît que l'interface IA. De ce fait n'importe quelle classe implementant cette interface peut être associée à la classe B, sans qu'il soit nécessaire de modifier quoi que se soit dans la classe B.
- Avec le couplage faible, nous pourrons créer des applications fermées à la modification et ouvertes à l'extension.



Exemple de coupe faible



```
package metier;
public interface IMetier {
    public double calcul();
}
```

```
package metier;
import dao.IDao;
public class MetierImpl implements IMetier {
    private IDao dao;
    public double calcul() {
        double nb=dao.getValue();
        return 2*nb;
    }
    // Getters et Setters
}
```

```
package dao;
public interface IDao {
    public double getValue();
}
```

```
package dao;
public class DaoImpl implements IDao {
    public double getValue() {
        return 5;
    }
}
```

Injection des dépendances

- Injection par instantiation statique :

```
import metier.MetierImpl;
import dao.DaoImpl;
public class Presentation {
public static void main(String[] args) {
    DaoImpl dao=new DaoImpl();
    MetierImpl metier=new MetierImpl();
    metier.setDao(dao);
    System.out.println(metier.calcul());
}
}
```



Injection des dépendances

- Injection par instantiation dynamique par réflexion :
 - Fichier texte de configuration : config.txt
- ext.DaoImp
metier.MetierImpl

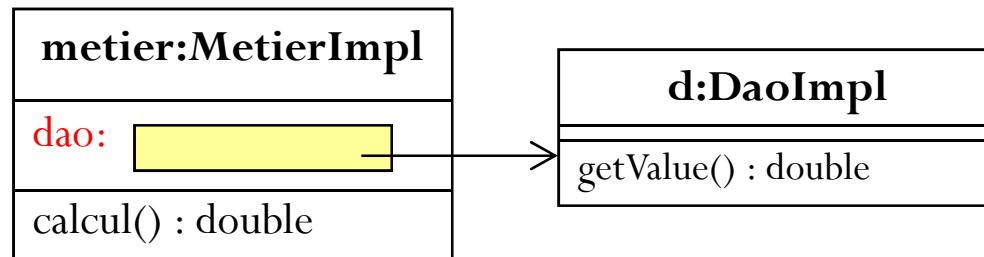
```
import java.io.*;import java.lang.reflect.*;
import java.util.Scanner; import metier.IMetier;
import dao.IDao;
public class Presentation {
public static void main(String[] args) {
try {
Scanner scanner=new Scanner(new File("config.text"));
String daoClassname=scanner.next();
String metierClassName=scanner.next();
Class cdao=Class.forName(daoClassname);
IDao dao= (IDao) cdao.newInstance();
Class cmetier=Class.forName(metierClassName);
IMetier metier=(IMetier) cmetier.newInstance();
Method meth=cmetier.getMethod("setDao",new Class[]{IDao.class});
meth.invoke(metier, new Object[]{dao});
System.out.println(metier.calcul());
} catch (Exception e) { e.printStackTrace(); }
}
}
```

Injection des dépendances avec Spring.

- L'injection des dépendance, ou l'inversion de contrôle est un concept qui intervient généralement au début de l'exécution de l'application.
- Spring IOC commence par lire un fichier XML qui déclare quelles sont différentes classes à instancier et d'assurer les dépendances entre les différentes instances.
- Quand on a besoin d'intégrer une nouvelle implémentation à une application, il suffirait de la déclarer dans le fichier xml de beans spring.

Injection des dépendances dans une application java standard

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-
2.0.dtd" >
<beans>
    <bean id="d" class="dao.DaoImpl2"></bean>
    <bean id="metier" class="metier.MetierImpl">
        <property name="dao" ref="d"></property>
    </bean>
</beans>
```



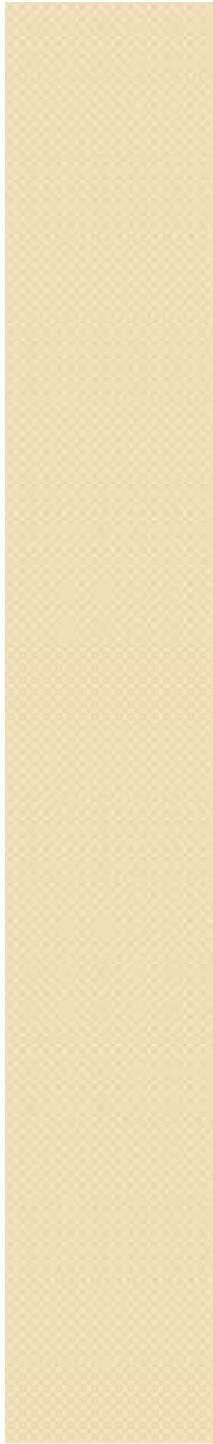
Injection des dépendances dans une application java standard

```
package pres;

import metier.IMetier;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Presentation {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context=new
        ClassPathXmlApplicationContext(new String[]{"spring-ioc.xml"});
        IMetier metier=(IMetier) context.getBean("metier");
        System.out.println(metier.calcul());
    }
}
```

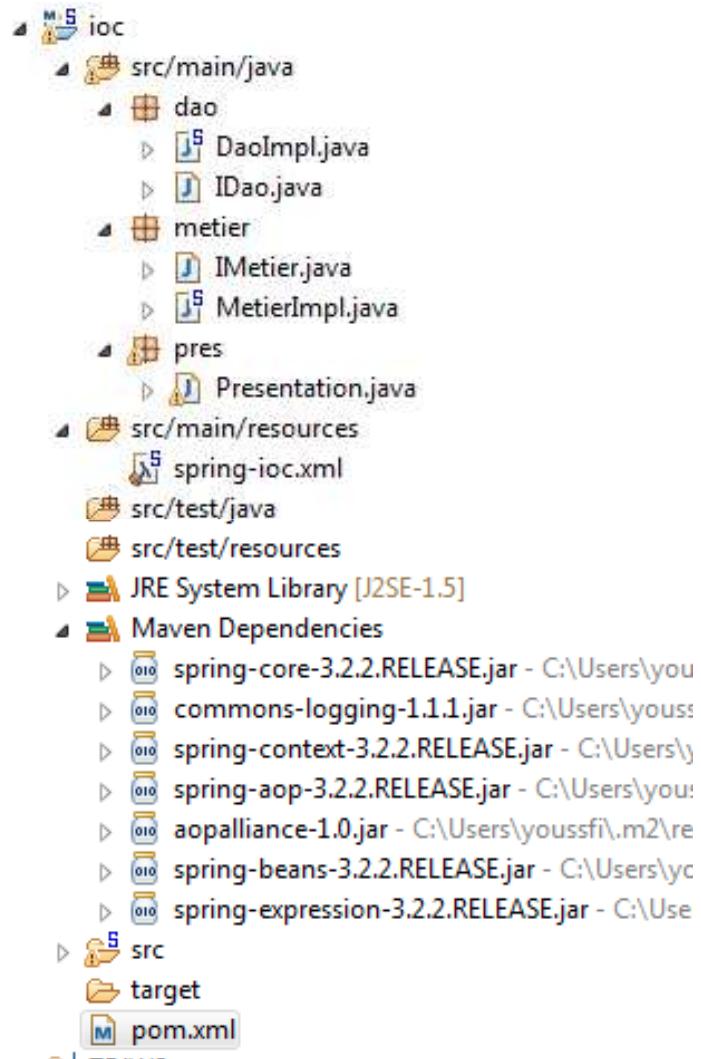


med@youssf.net

Maven dependencies

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>3.2.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>3.2.2.RELEASE</version>
    </dependency>
</dependencies>
```

Structure du projet



Injection des dépendances dans une application web

- Dans une application web, SpringIOC est appelé au démarrage du serveur en déclarant le listener ContextLoaderListener dans le fichier **web.xml**

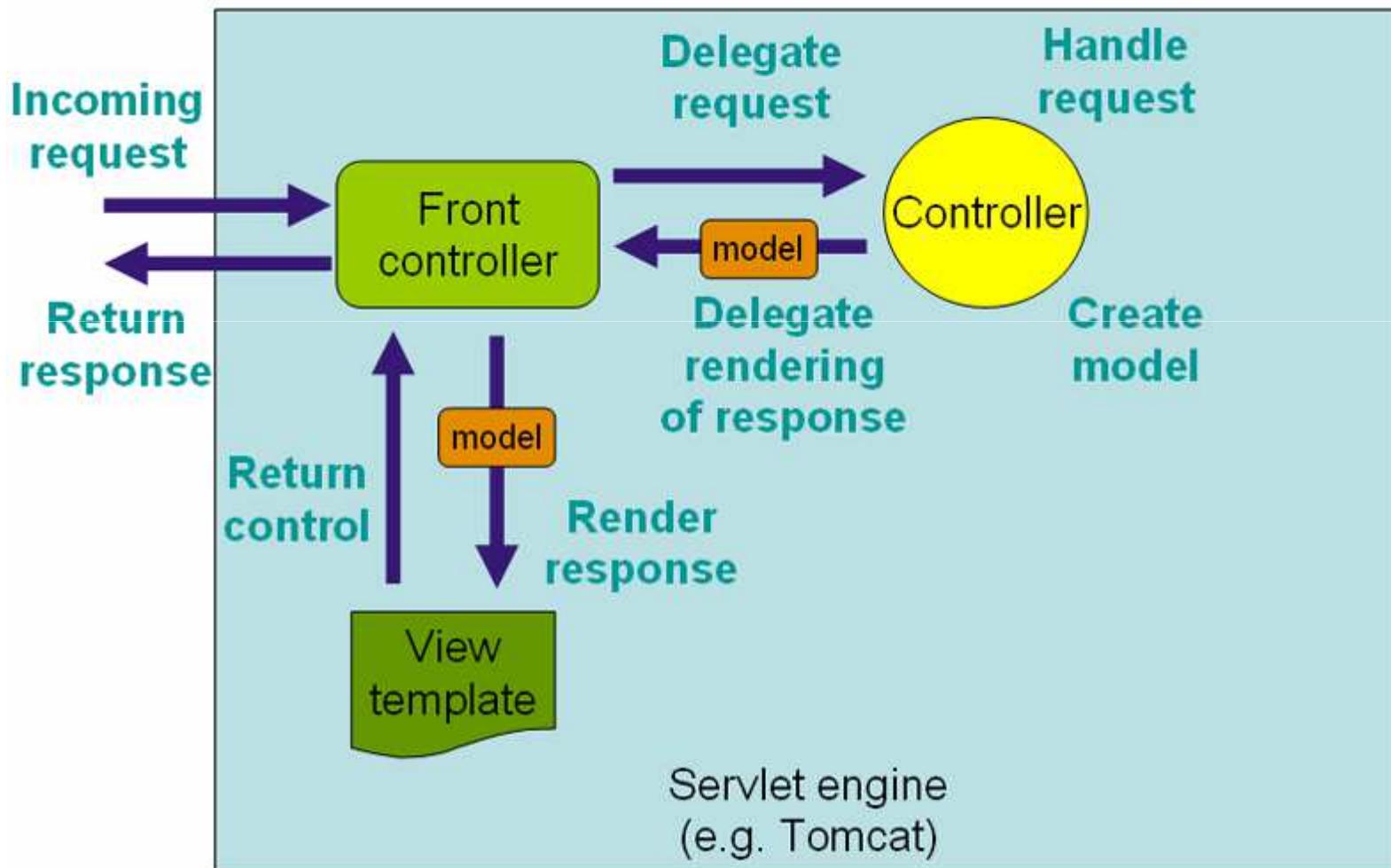
```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-beans.xml</param-value>
</context-param>
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

- Dans cette déclaration, CotextLoaderListener est appelé par Tomcat au moment du démarrage de l'application. Ce listener cherchera le fichier de beans spring « spring-beans.xml » stocké dans le dossier WEB-INF ce qui permet de faire l'injection des dépendances entre MetierImpl et Daolmpl

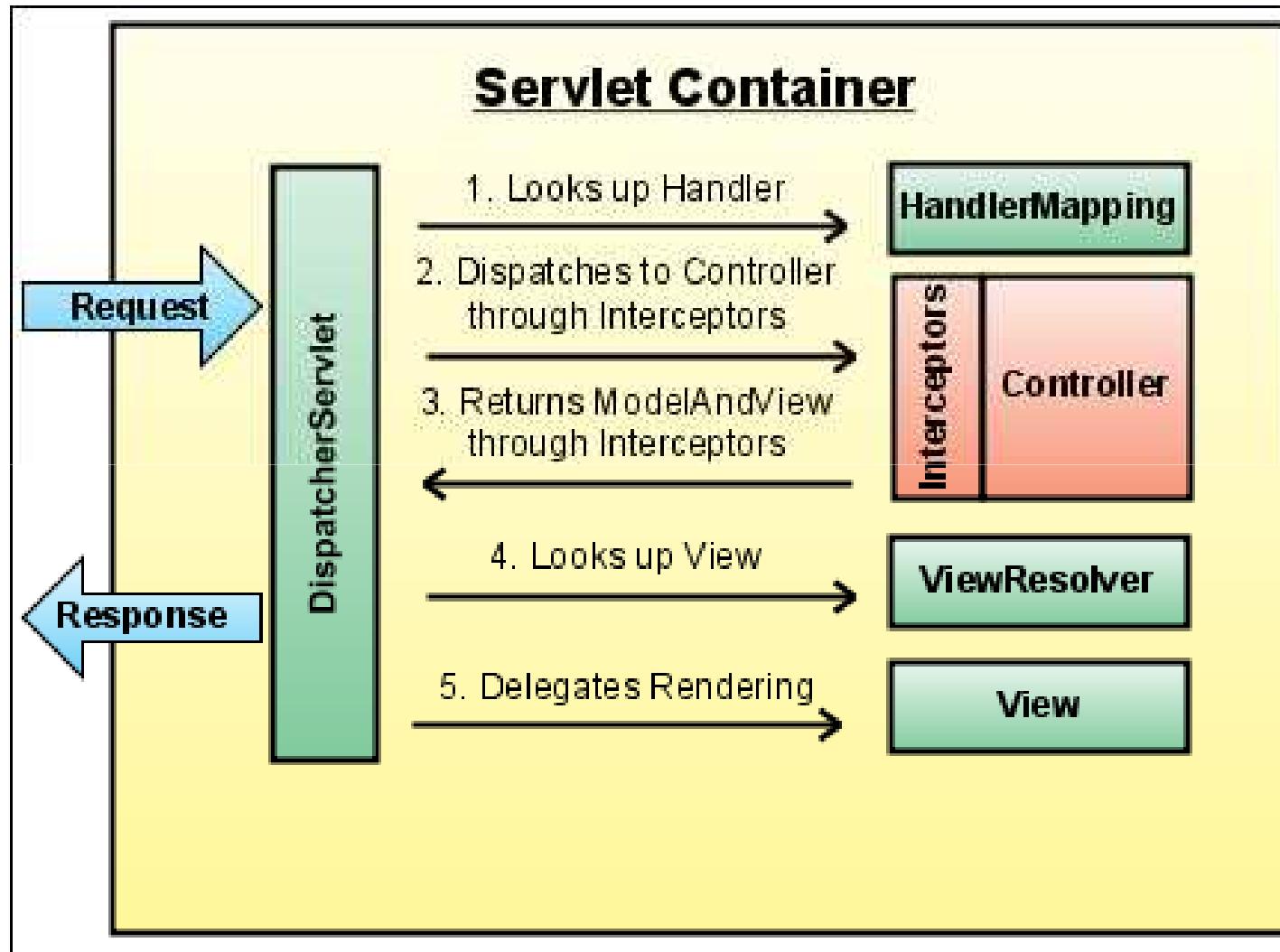
Spring MVC

med@youssf.net

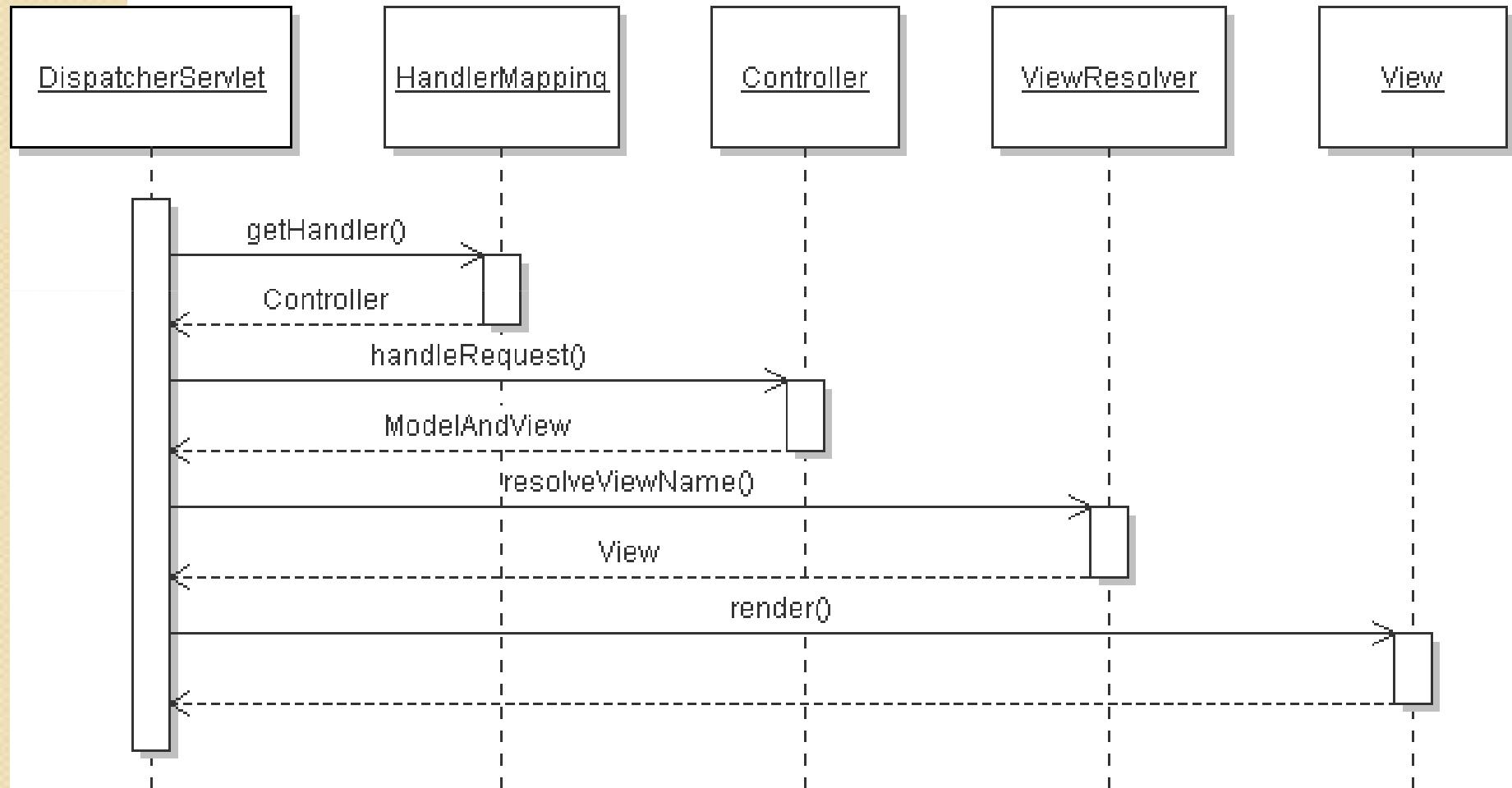
Spring MVC



Spring MVC Architecture



Spring MVC



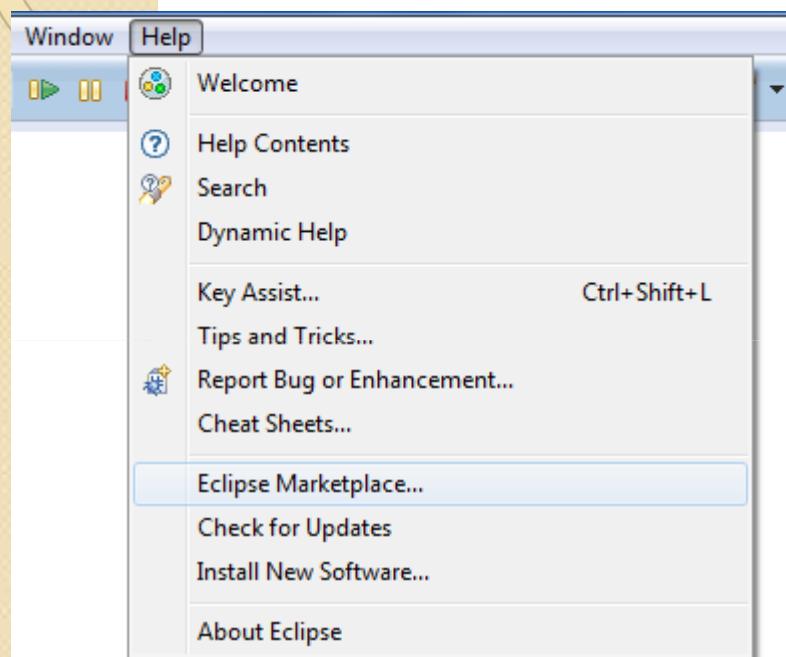
Spring MVC

- Le client fait une demande au contrôleur. Celui-ci voit passer toutes les demandes des clients. C'est la porte d'entrée de l'application. C'est le **C** de MVC. Ici le contrôleur est assuré par une servlet générique :
org.springframework.web.servlet.DispatcherServlet
Le contrôleur principal [**DispatcherServlet**] fait exécuter l'action demandée par l'utilisateur par une classe implémentant l'interface :
org.springframework.web.servlet.mvc.Controller
 - A cause du nom de l'interface, nous appellerons une telle classe un contrôleur secondaire pour le distinguer du contrôleur principal [**DispatcherServlet**] ou simplement contrôleur lorsqu'il n'y a pas d'ambiguïté.
- Le contrôleur [**Controller**] traite une demande particulière de l'utilisateur. Pour ce faire, il peut avoir besoin de l'aide de la couche métier. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
 - une page d'erreurs si la demande n'a pu être traitée correctement
 - une page de confirmation sinon

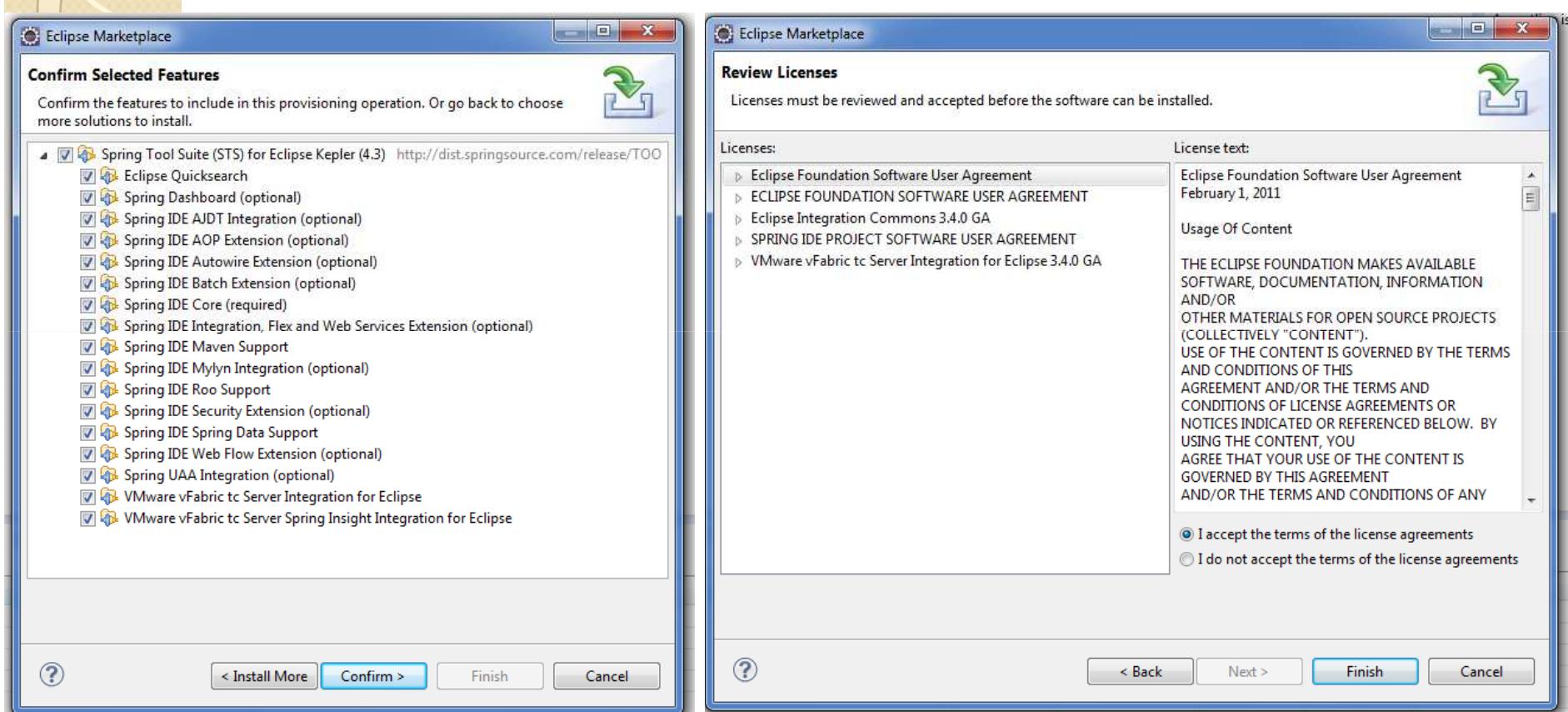
Spring MVC

- 4- Le contrôleur choisit la réponse (= vue) à envoyer au client. Choisir la réponse à envoyer au client nécessite plusieurs étapes :
 - choisir l'objet qui va générer la réponse. C'est ce qu'on appelle la vue **V**, le **V** de MVC. Ce choix dépend en général du résultat de l'exécution de l'action demandée par l'utilisateur.
 - lui fournir les données dont il a besoin pour générer cette réponse. En effet, celle-ci contient le plus souvent des informations calculées par la couche métier ou le contrôleur lui-même. Ces informations forment ce qu'on appelle le modèle **M** de la vue, le **M** de MVC. Spring MVC fournit ce modèle sous la forme d'un dictionnaire de type **java.util.Map**.
 - Cette étape consiste donc en le choix d'une vue **V** et la construction du modèle **M** nécessaire à celle-ci.
- 5- Le contrôleur **DispatcherServlet** demande à la vue choisie de s'afficher. Il s'agit d'une classe implémentant l'interface
org.springframework.web.servlet.View
 - Spring MVC propose différentes implémentations de cette interface pour générer des flux HTML, Excel, PDF, ...
- 6. le générateur de vue **View** utilise le modèle **Map** préparé par le contrôleur **Controller** pour initialiser les parties dynamiques de la réponse qu'il doit envoyer au client.
- 7. la réponse est envoyée au client. La forme exacte de celle-ci dépend du générateur de vue. Ce peut être un flux HTML, XML, PDF, Excel, ...

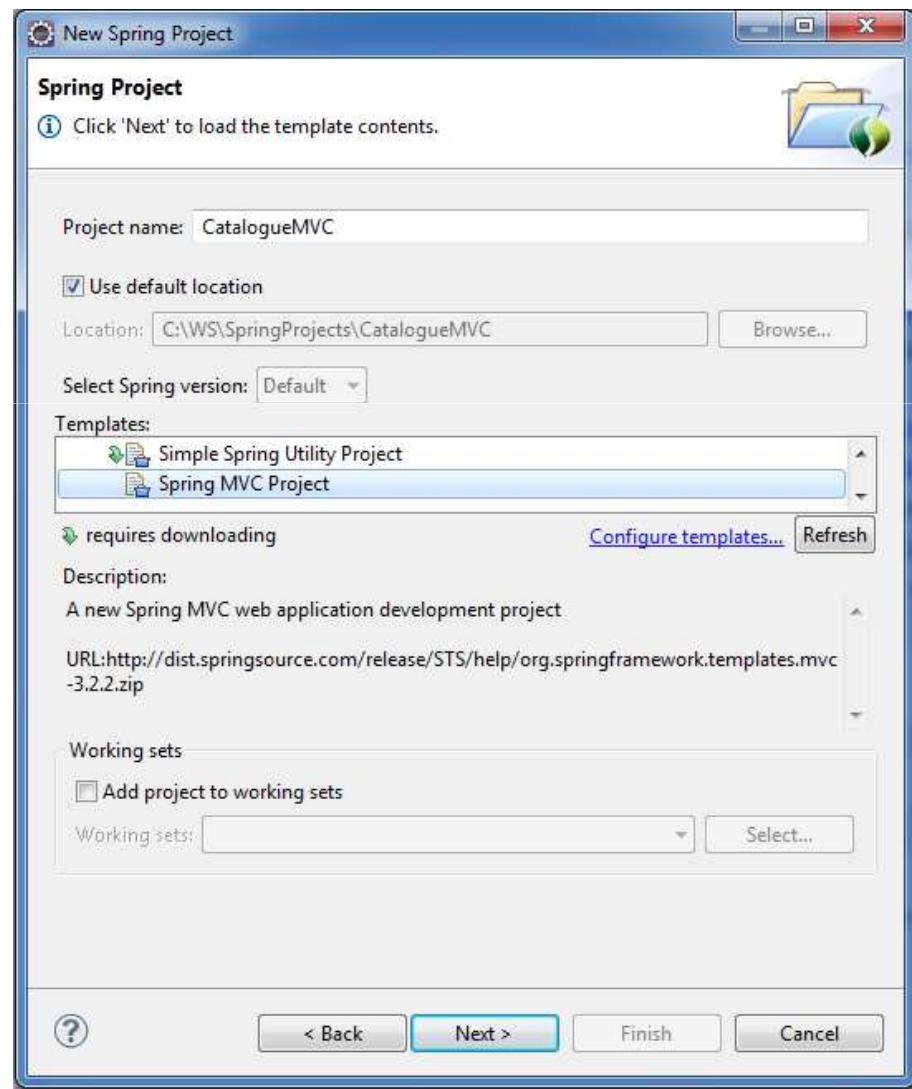
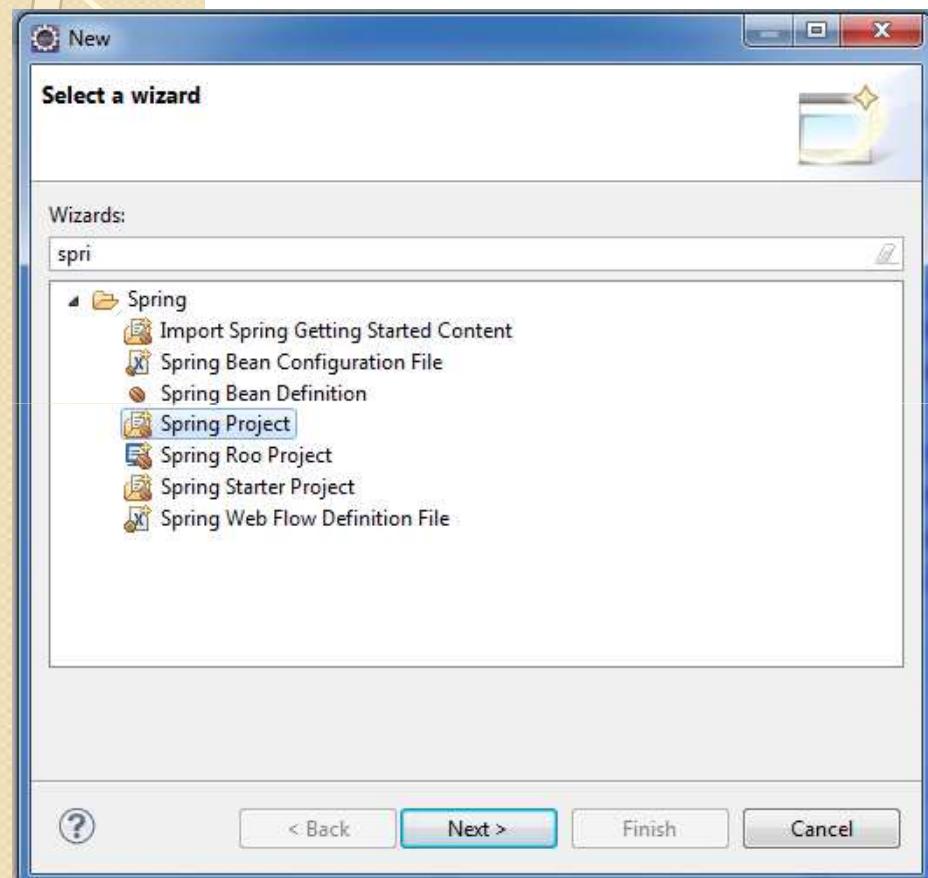
Installation du plugin : spring tools pour eclipse



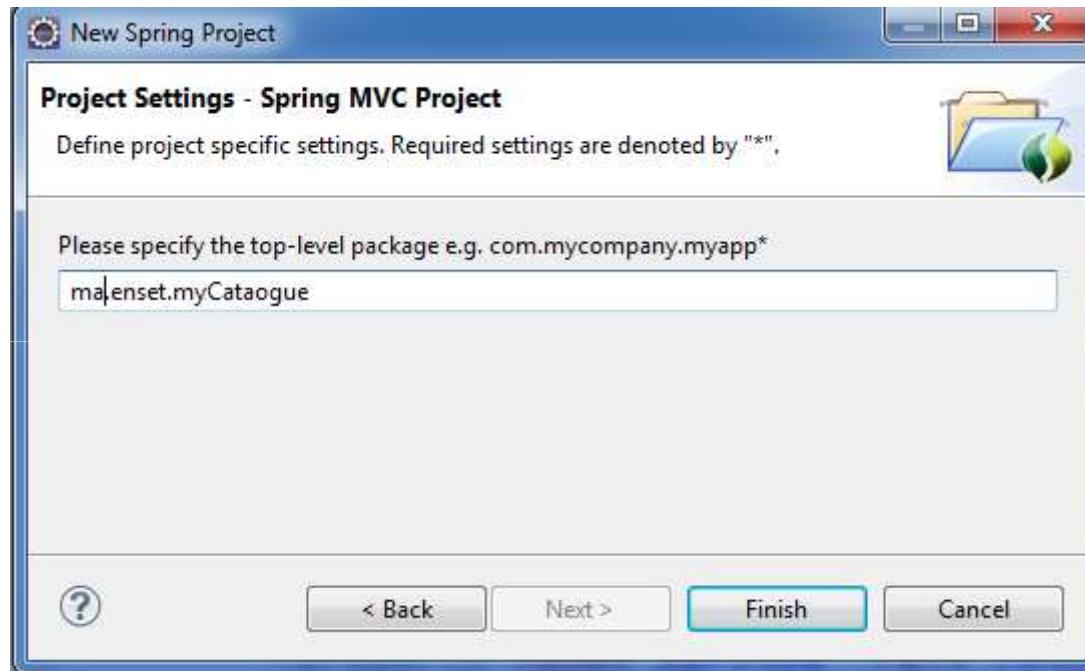
Installation du plugin : spring tools pour eclipse



Création d'un projet Spring

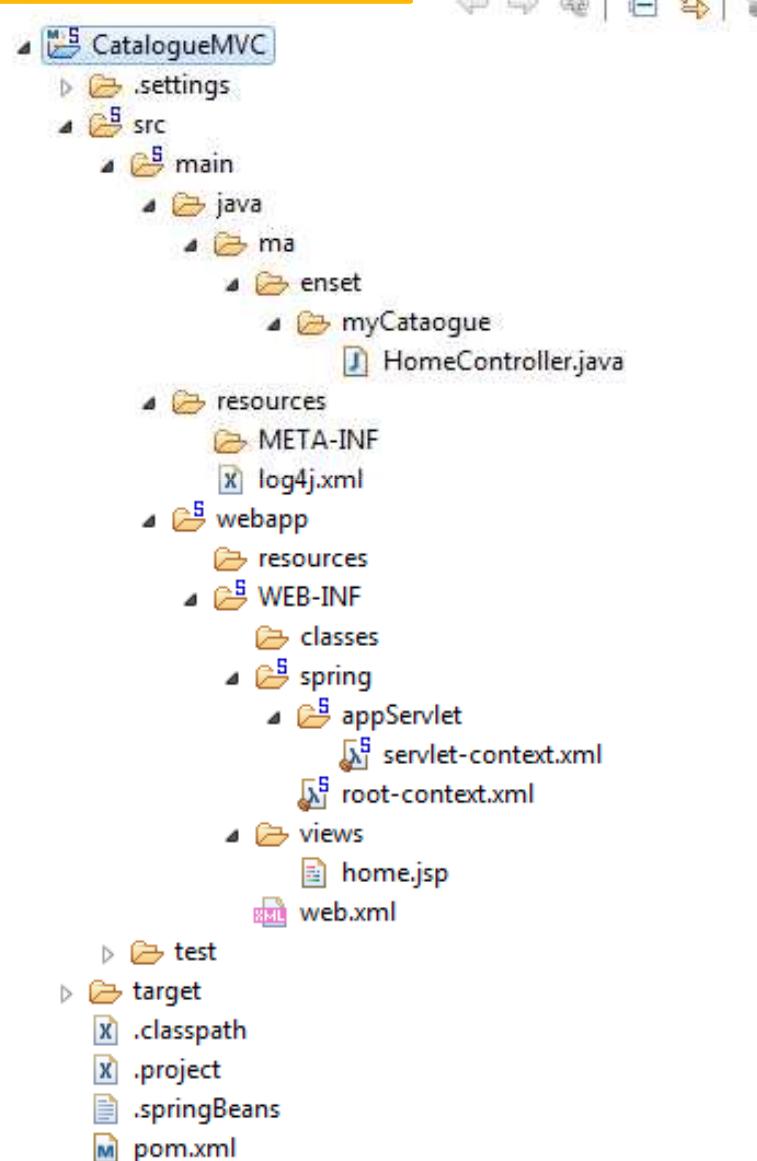


Création d'un projet Spring



Structure du projet

Navigator Explorer



med@youssfi.net

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">
<!-- The definition of the Root Spring Container shared by all Servlets and
Filters --&gt;
&lt;context-param&gt;
    &lt;param-name&gt;contextConfigLocation&lt;/param-name&gt;
    &lt;param-value&gt;/WEB-INF/spring/root-context.xml&lt;/param-value&gt;
&lt;/context-param&gt;
<!-- Creates the Spring Container shared by all Servlets and Filters --&gt;
&lt;listener&gt;
&lt;listener-
    class&gt;org.springframework.web.context.ContextLoaderListener&lt;/listener-
    class&gt;
&lt;/listener&gt;</pre>
```

web.xml

```
<!-- Processes application requests -->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```



>/WEB-INF/spring/root-context.xml

- Ce fichier est lu par ContextLoaderListener, au démarrage du serveur .
- C'est un fichier dans lequel contexte de l'application sera construit
- ContextLoaderListener représente Spring IOC
- c'est donc un fichier pour l'injection des dépendances
- Pour le moment, il est vide

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-
    beans.xsd">

    <!-- Root Context: defines shared resources visible
        to all other web components -->

</beans>
```

>/WEB-INF/spring/appServlet/servlet-context.xml

- Ce fichier est lu par DispatcherServlet qui représente le contrôleur web de l'application

```
<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static
        resources in the ${webappRoot}/resources directory -->
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the
        /WEB-INF/views directory -->
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <context:component-scan base-package="ma.enset.myCataogue" />
</beans:beans>
```

Un exemple de contrôleur Spring MVC

```
package ma.enset.myCataogue;

import java.text.*;import java.util.*;import org.slf4j.*;import
    org.springframework.stereotype.Controller;

import org.springframework.ui.Model; import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
/** Handles requests for the application home page. */
@Controller
public class HomeController {

    private static final Logger Logger = LoggerFactory.getLogger(HomeController.class);
    /** Simply selects the home view to render by returning its name. */
    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        Logger.info("Welcome home! The client Locale is {}.", locale);
        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG, Locale);
        String formattedDate = dateFormat.format(date);
        model.addAttribute("serverTime", formattedDate );
        return "home";
    }
}
```

Un exemple de vue JSP

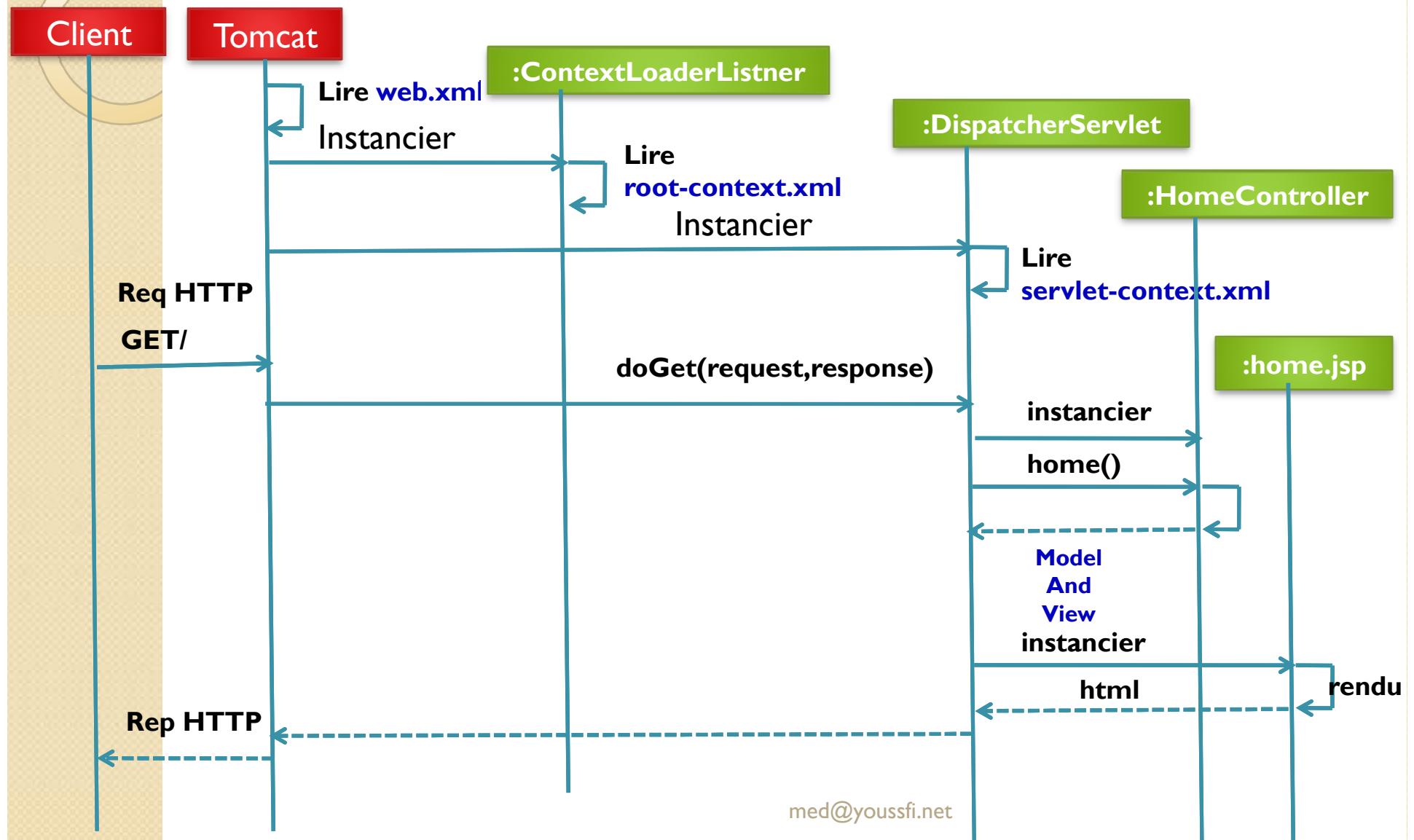
```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
    <head>
        <title>Home</title>
    </head>
    <body>
        <h1> Hello world! </h1>
        <p> The time on the server is ${serverTime}. </p>
    </body>
</html>
```

http://localhost:8080/myCataogue/

Hello world!

The time on the server is 23 décembre 2013 13:47:12 WET.

Fonctionnement





Application

- Créer une application qui permet de gérer le catalogue de produits classés par catégories.
- L'application doit permettre de :
 - Saisir, ajouter, éditer, Supprimer et consulter les catégories
 - Saisir, ajouter, éditer, supprimer et consulter les produits
- L'application peut être consultée via :
 - Un Client Web
 - Un client SOAP
 - Un client RMI
 - Un Client Mobile Androide

Aperçu des écran des : application Web

Catalogue

localhost:8080/web/categories/save

CODE CAT:

NOM CAT:

Photo: Choisissez un fichier Aucun fichier choisi

CODE	NOM CAT	PHOTO		
1	Ordinateurs		Supprimer	Editer
5	Téléphones		Supprimer	Editer

Produits

localhost:8080/web/produits/chercher?motCle=

Mot Clé :

REF:

Désignation:

Catégorie: Ordinateurs ▾

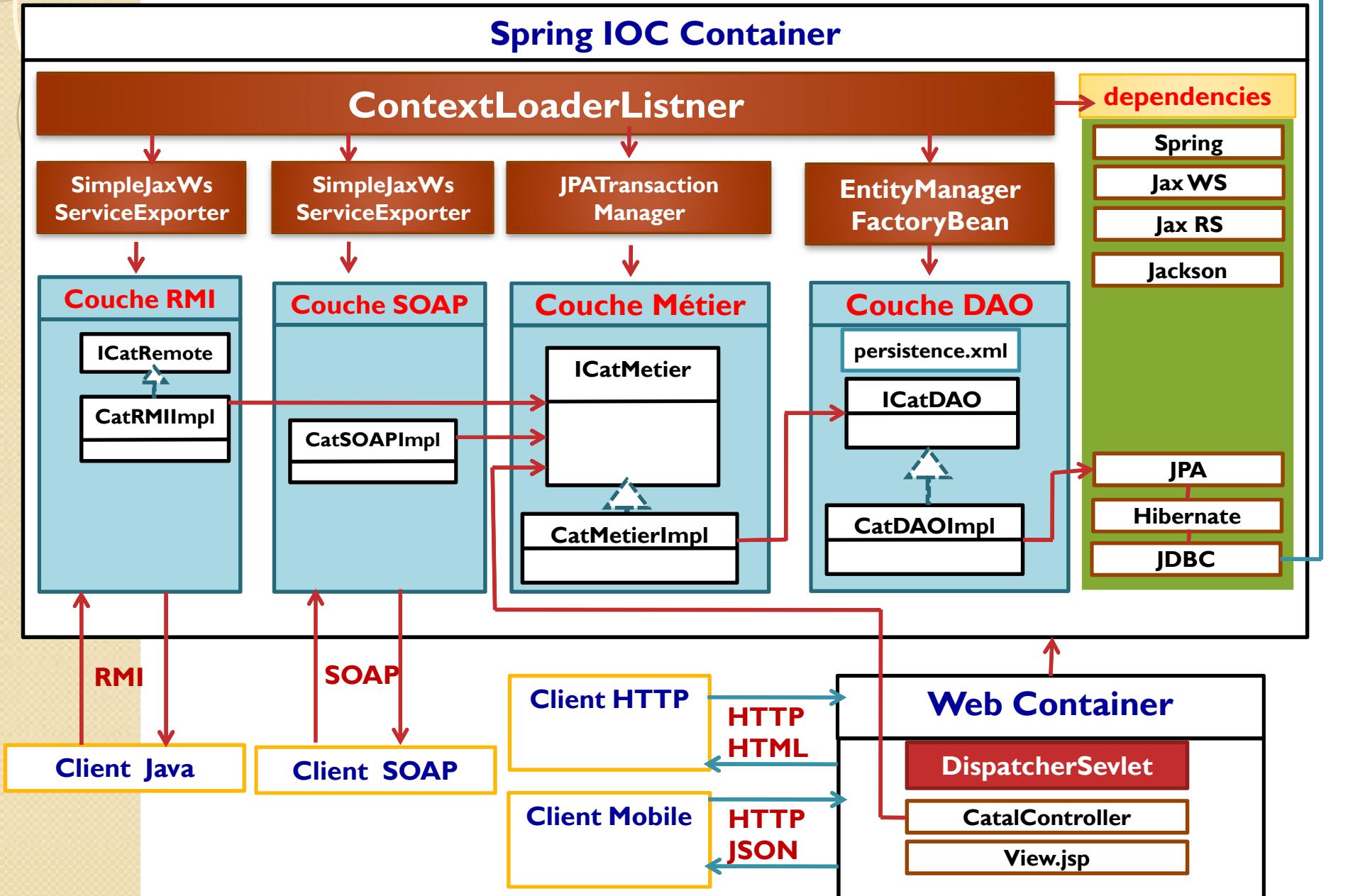
Prix:

Quantité:

Disponible:

REF	DES	PRIX	QUANTITE	DISPO	
C	D	0.0	0	Supprimer	Editer
MMM78	12345	89.0	3	Supprimer	Editer
PK34	DES34	900.0	5	Supprimer	Editer

Architecture



Les propriétés du projet maven

```
<modelVersion>4.0.0</modelVersion>
<groupId>org.bp</groupId>
<artifactId>web</artifactId>
<name>CatalogueWeb</name>
<packaging>war</packaging>
<version>1.0.0-BUILD-SNAPSHOT</version>
<properties>
<java-version>1.6</java-version>
<org.springframework-version>3.2.8.RELEASE</org.springframework-
version>
<org.aspectj-version>1.6.10</org.aspectj-version>
<org.slf4j-version>1.6.6</org.slf4j-version>
</properties>
```

Les dépendances Maven : Module CatalogueDAO

```
<dependency>
    <groupId>org.bp</groupId>
    <artifactId>CatalogueDAO</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

Les dépendances Maven : Spring

```
<!-- Spring -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${org.springframework-version}</version>
    <exclusions>
        <!-- Exclude Commons Logging in favor of SLF4j -->
        <exclusion>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
```

Les dépendances Maven : Spring

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${org.springframework-version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${org.springframework-version}</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>3.2.2.RELEASE</version>
</dependency>
```

Les dépendances Maven :Apache FileUpload

```
<!-- Apache Commons Upload -->
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.2.2</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-io</artifactId>
    <version>1.3.2</version>
</dependency>
```



Les dépendances Maven : Servlet, JSP, JSTL

```
<!-- Servlet -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.1</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

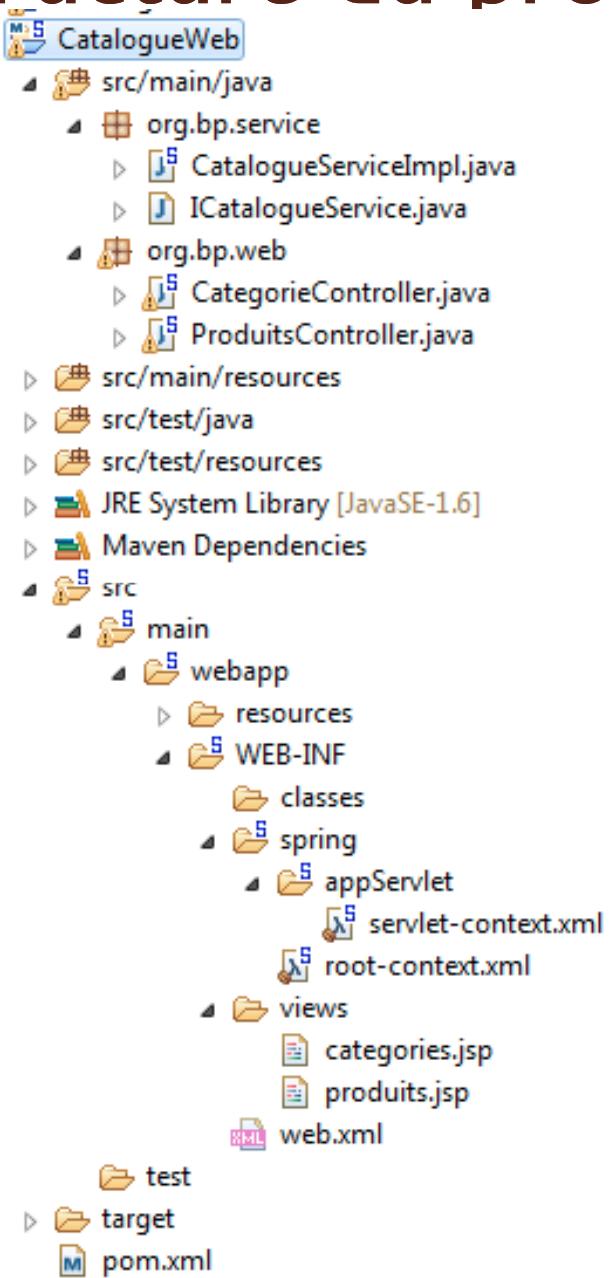
Les dépendances Maven : Logging (SLF4J)

```
<!-- Logging -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${org.slf4j-version}</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>${org.slf4j-version}</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>${org.slf4j-version}</version>
    <scope>runtime</scope>
</dependency>
```

Les dépendances Maven : Logging (Log4j)

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.15</version>
  <exclusions>
    <exclusion>
      <groupId>javax.mail</groupId>
      <artifactId>mail</artifactId>
    </exclusion>
    <exclusion>
      <groupId>javax.jms</groupId>
      <artifactId>jms</artifactId>
    </exclusion>
    <exclusion>
      <groupId>com.sun.jdmk</groupId>
      <artifactId>jmxtools</artifactId>
    </exclusion>
    <exclusion>
      <groupId>com.sun.jmx</groupId>
      <artifactId>jmxri</artifactId>
    </exclusion>
  </exclusions>
  <scope>runtime</scope>
</dependency>
```

Structure du projet à développer



med@youssfi.net



Couche Service

- La couche métier ou service qui la couche qui s'occupe des traitements l'application.
- Cette couche fait appel à la couche DAO
- La couche web interagit avec la couche service et avec la couche DAO
- C'est dans la couche service ou nous allons gérer les transactions
- Spring possède un module qui permet de gérer les transaction via des annotations.

L'interface ICatalogueService

```
package org.bp.service;
import java.util.List;
import org.bp.dao.entities.Categorie;
import org.bp.dao.entities.Produit;
public interface ICatalogueService {
    public void addCategorie(Categorie c);
    public void addProduit(Produit p, Long codeCat);
    public List<Categorie> listCategories();
    public List<Produit> produitsParCat(Long codeCat);
    public List<Produit> produitsParMC(String mc);
    public Produit getProduit(String ref);
    public void updateProduit(Produit p);
    public void deleteProduit(String ref);
    public Categorie getCategorie(Long codecat);
    public void deleteCategorie(Long codeCat);
    public void updateCategorie(Categorie c);
}
```

Implémentation : CatalogueServiceImpl

```
package org.bp.service;

import java.util.List;import org.bp.dao.ICatalogueDAO;import
org.bp.dao.entities.Categorie;
import org.bp.dao.entities.Produit;
import org.springframework.transaction.annotation.Transactional;
@Transactional

public class CatalogueServiceImpl implements ICatalogueService {
    private ICatalogueDAO dao;
    public void setDao(ICatalogueDAO dao) { this.dao = dao; }

    @Override
    public void addCategorie(Categorie c) { dao.addCategorie(c); }

    @Override
    public void addProduit(Produit p, Long codeCat) { dao.addProduit(p, codeCat); }

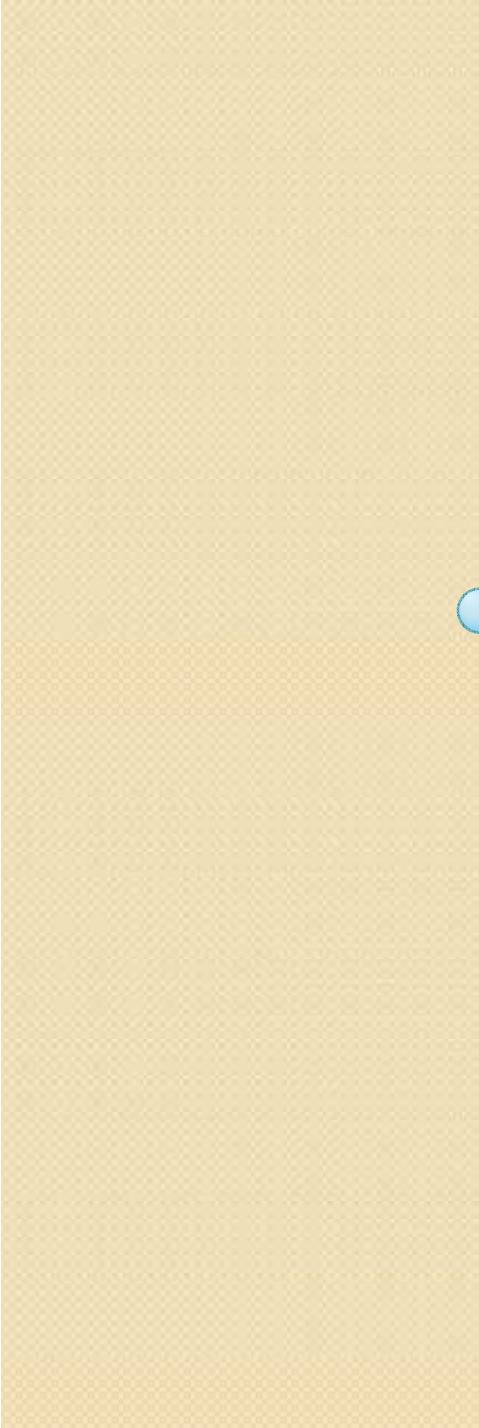
    @Override
    public List<Categorie> listCategories() { return dao.listCategories(); }

    @Override
    public List<Produit> produitsParCat(Long codeCat) { return
        dao.produitsParCat(codeCat);}

    @Override
    public List<Produit> produitsParMC(String mc) { return dao.produitsParMC(mc); }
```

Implémentation : CatalogueServiceImpl

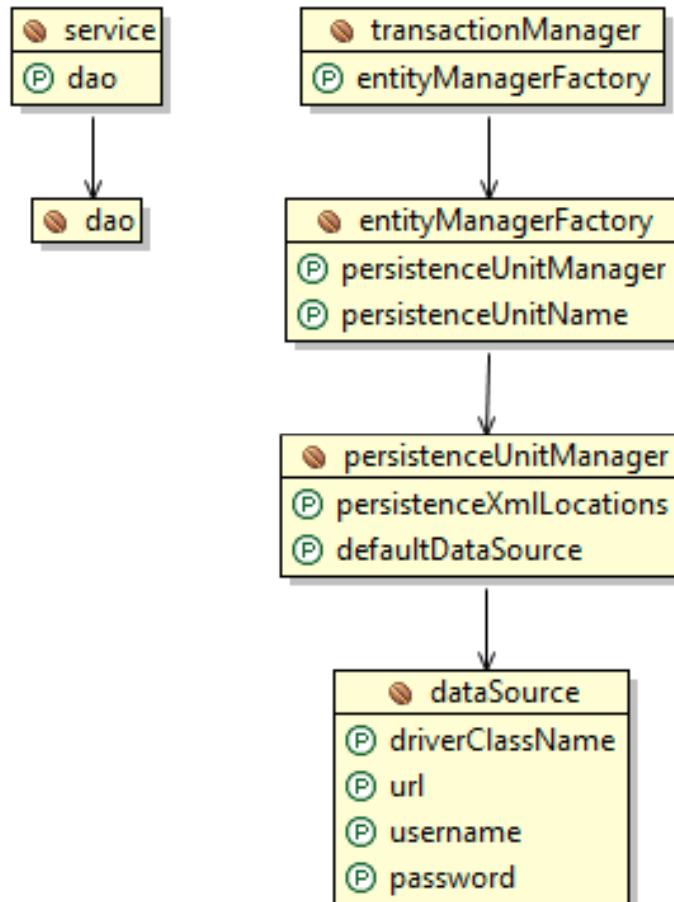
```
@Override  
public Produit getProduit(String ref) { return dao.getProduit(ref); }  
  
@Override  
public void updateProduit(Produit p) { dao.updateProduit(p); }  
  
@Override  
public void deleteProduit(String ref) { dao.deleteProduit(ref); }  
  
@Override  
public Categorie getCategorie(Long codecat) {  
    return dao.getCategorie(codecat);  
}  
  
@Override  
public void deleteCategorie(Long codeCat) { dao.deleteCategorie(codeCat); }  
  
@Override  
public void updateCategorie(Categorie c) { dao.updateCategorie(c); }  
}
```



- **INJECTION DES
DEPENDANCES**

Injection des dépendances

Spring Beans



[Source](#) [Namespaces](#) [Overview](#) [beans](#) [context](#) [tx](#) [Beans Graph](#)

Injection dé dépendances

- Au démarrage du projet, Spring ContextLoaderLisener va démarrer en premier lieu.
- Il va chercher son fichier de configurations root-context.xml dans lequel il va trouver les différents objet à instancier et gérer les dépendances entre ces objet de façon à ce que le contexte de l'application soit configuré.
- Il devrait commencer par instancier l'objet DAO
- Ensuite, il va instancier l'objet service en injectant les dépendances entre l'objet servit l'objet DAO
- Il doit également configurer le Data Source
- Ensuite il doit configurer l'unité de persistance en faisant appel au fichier de la couche DAO persietnce.xml et en lui associant le datasource.
- Par la suite , il doit Créer une fabrique EntityManager qui sera injecté dans la couche DAO via l'annotation @PersistanceContext.
- Instancier un gestionnaire de transaction
- Les deux dernières lignes de ce fichier permettent de forcer Spring à prendre en considération les deux annotations:
 - @Trasaction , utilisée dans la couche service pour associer aux méthodes un aspect qui permet de faire évoluer les méthodes métier dans un context transactionnel.
 - @PersistanceContext , utilisée da la couche DAO pour injecter un EntityManager qui gère la persistance.



Injection des dépendances

/WEB-INF/spring/root-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
                           http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd">

    <bean id="dao" class="org.bp.dao.CatalogueDaoImpl" init-method="init"></bean>
    <bean id="service" class="org.bp.service.CatalogueServiceImpl">
        <property name="dao" ref="dao"></property>
    </bean>
    <bean id="dataSource"
          class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/DB_CAT_BP2"/>
        <property name="username" value="root"></property>
        <property name="password" value=""></property>
    </bean>
```



Injection des dépendances

/WEB-INF/spring/root-context.xml

```
<bean id="persistenceUnitManager"
      class="org.springframework.orm.jpa.persistenceunit.DefaultPersistenceUnitManager">
    <property name="persistenceXmlLocations">
      <list>
        <value>classpath*:META-INF/persistence.xml</value>
      </list>
    </property>
    <property name="defaultDataSource" ref="dataSource"></property>
  </bean>
  <bean id="entityManagerFactory"
        class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="persistenceUnitManager" ref="persistenceUnitManager"></property>
    <property name="persistenceUnitName" value="UP_CAT"></property>
  </bean>
  <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"></property>
  </bean>
  <tx:annotation-driven transaction-manager="transactionManager"/>
  <context:annotation-config></context:annotation-config>
</beans>
```



PARTIE SPRING MVC

med@youssf.net

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">
    <!-- The definition of the Root Spring Container shared by all Servlets and Filters --
    >
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>
    <!-- Creates the Spring Container shared by all Servlets and Filters -->
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
```

web.xml

```
<!-- Processes application requests -->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

WEB-INF/spring/appServlet/servlet-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- DispatcherServlet: defines this servlet's request-processing infrastructure -->
    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />
    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static
        resources in the ${webappRoot}/resources directory -->
    <resources mapping="/resources/**" location="/resources/" />
    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the
        /WEB-INF/views directory -->
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>
    <context:component-scan base-package="org.bp.web" />
```

WEB-INF/spring/appServlet/servlet-context.xml

Configuration de l'opération Upload:

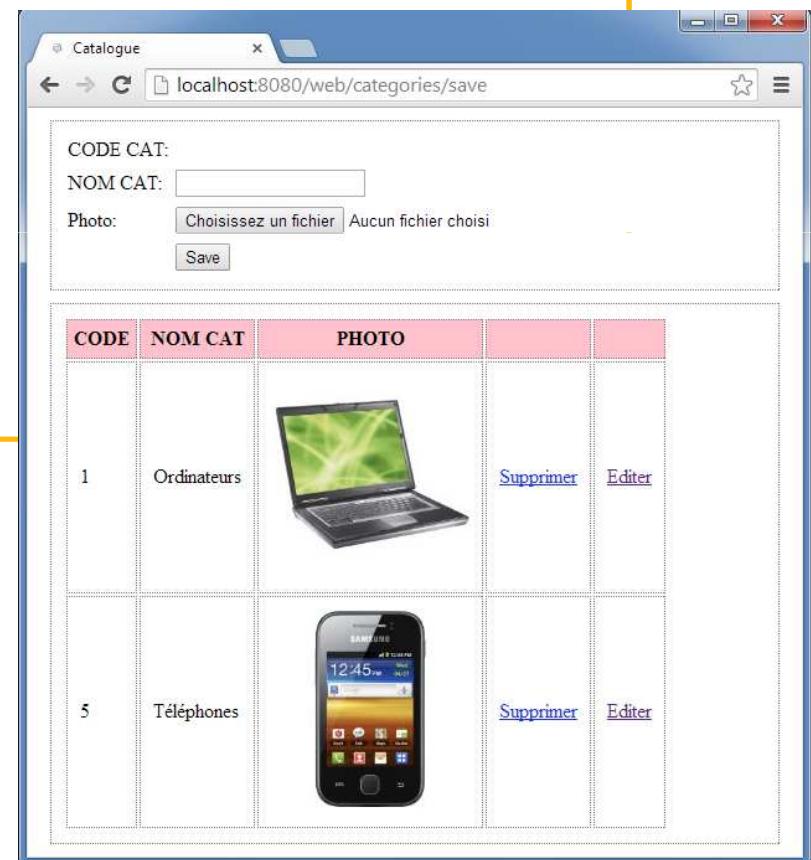
- La taille des fichiers Uploadés ne doit pas dépasser 100000 octets

```
<!-- Configuration Upload-->
<beans:bean name="multipartResolver"
  class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
  <beans:property name="maxUploadSize" value="100000"></beans:property>
</beans:bean>

</beans:beans>
```

Gestion des catégories

- Nous allons définir un contrôleur et une vue JSP qui permet de gérer les catégories :
 - Formulaire de saisie
 - Ajout d'une catégorie avec la validation des données saisies dans le formulaire.
 - Editer une catégorie
 - Modifier une catégorie
 - Supprimer une catégorie
 - Récupérer la photo d'une catégorie.



CategorieController

```
package org.bp.web;  
import java.io.*; import javax.validation.Valid;  
import org.apache.commons.io.IOUtils;  
import org.bp.dao.entities.Categorie; import org.bp.service.ICatalogueService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.MediaType;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.ResponseBody;  
import org.springframework.web.multipart.MultipartFile;  
  
@Controller  
@RequestMapping("/categories")  
public class CategorieController {  
    @Autowired  
    private ICatalogueService service;
```

Action : GET categories/index

```
@RequestMapping(value="/index")
public String index(Model model){
    model.addAttribute("categorie", new Categorie());
    model.addAttribute("action", "save");
    model.addAttribute("categories", service.listCategories());
    return "categories";
}
```

Quand le client envoie la requête : <http://localhost:8080/categories/index>,

- DispatcherServlet reçoit cette requête, ensuite il fait appel à la méthode index du contrôleur
- La méthode index créer et stocker un objet de type Categorie dans le modèle. Cet objet categorie, sera utilisé dans la vue comme modèle du formulaire.
- la vue contiendra un champ caché qui indique si le formulaire dans dans le mode édition ou ajout. Ici on suppose que le formulaire est en mode save.
- dans le modèles nous stockons aussi toutes les catégories qui seront affichées dans la vue.
- A la fin, Cette méthode retourne à DispatcherServlet le nom de la vue à affichée.
- Le résolveur de vue est configuré pour chercher la vue views/categories.jsp

Vue : views/categories.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="f"%>
<!DOCTYPE html >
<html> <head> <meta charset="UTF-8"> <title>Catalogue</title>
<link rel="stylesheet" type="text/css"
      href="<%request.getContextPath()%>/resources/css/style.css"/>
</head>
<body>
  <div>
    <f:form modelAttribute="categorie" action="save" enctype="multipart/form-data">
      <table>
        <tr>
          <td>CODE CAT:</td> <td>${categorie.codeCategorie}
          <f:input type="hidden" path="codeCategorie" id="codeCategorie"/></td>
          <td><f:errors path="codeCategorie" cssClass="errors"></f:errors></td>
        </tr>
        <tr>
          <td>NOM CAT:</td><td><f:input path="nomCategorie" id="nomCategorie"/></td>
          <td><f:errors path="nomCategorie" cssClass="errors"></f:errors></td>
        </tr>
```



Vue : views/categories.jsp

```
<tr>
    <td>Photo:</td>
    <c:if test="#{categorie.codeCategorie!=null}">
        <td>
            
        </td>
    </c:if>
    <td><input type="file" name="fileCat"/></td>
</tr>
<tr>
    <td><input type="hidden" name="action" value="${action}"/></td>
    <td><input type="submit" value="Save"></td>
</tr>
</table>
</f:form>
</div>
```

Vue : views/categories.jsp

```
<div>
    <table class="table1">
        <tr>
            <th>CODE</th><th>NOM CAT</th><th>PHOTO</th><th></th><th></th></tr>
        </tr>
        <c:forEach items="${categories}" var="cat">
            <tr>
                <td>${cat.codeCategorie }</td><td>${cat.nomCategorie }</td>
                <td></td>
                <td><a
href="<%request.getContextPath()%>/categories/supprimer?codeCat=${cat.codeCategorie }">Supprimer</a></td>
                <td><a
href="<%request.getContextPath()%>/categories/editer?codeCat=${cat.codeCategorie }">Editer</a></td>
            </tr>
        </c:forEach>
    </table>
</div>
</body>
</html>
```

Action : GET categories/photo?codeCat=xx

```
@RequestMapping(value="/photo", produces=MediaType.IMAGE_JPEG_VALUE)
@ResponseBody
public byte[] getPhoto(Long codeCat) throws IOException{
    Categorie c=service.getCategorie(codeCat);
    if(c.getPhoto()==null) return new byte[0];
    return IOUtils.toByteArray(new ByteArrayInputStream(c.getPhoto()));
}
```

- La vue categories JSP a besoin d'afficher la photo de chaque catégorie.
- L'action /photo? codeCat=xxx, est associée à l'exécution de la méthode getPhoto.
- Cette méthode récupère le code de la catégorie
- Récupère l'objet Categorie de la base de données
- Envoie les données de la photo dans la réponse http.



Action : POST categories/save

Action pour ajouter ou mettre à jour une catégorie

```
@RequestMapping(value="/save")  
  
public String saveCategorie(String action, @Valid Categorie c,BindingResult bindingResult,Model  
model,MultipartFile fileCat) throws IOException{  
  
    if(bindingResult.hasErrors()){  
        model.addAttribute("categories", service.listCategories());  
        model.addAttribute("action", "save");return "categories";  
    }  
    if(!fileCat.isEmpty()){ c.setPhoto(fileCat.getBytes()); }  
    if(action.equals("save")) service.addCategorie(c);  
    else if(action.equals("edit")){  
        if(fileCat.isEmpty()){  
            Categorie ac=service.getCategorie(c.getCodeCategorie()); c.setPhoto(ac.getPhoto());  
        }  
        service.updateCategorie(c);  
    }  
    model.addAttribute("categorie", new Categorie()); model.addAttribute("action", "save");  
    model.addAttribute("categories", service.listCategories()); return "categories";  
}
```

Action : POST categories/save

- L'action save est invoqué au moment du post du formulaire pour ajouter une catégorie ou pour mettre à jour la catégorie éditée.
- DispatcherServlet stocker les données du formulaire dans un objet de type Categorie.
- Effectue la validation des données grâce à l'annotation @Valid puis stockent les résultats de validation dans la collection d'erreurs de type BindingResult.

Action : GET categories/supprimer?codeCat=xx

L'action categories/supprimer?codeCat=xx est associée à :

- ✓ l'exécution de la méthode supprimer
- ✓ Cette méthode récupère le paramètre URL codeCat
- ✓ Supprimer la catégorie de la base de données
- ✓ Avant de revenir à la vue, on charge à nouveau dans le modèle :
 - Une nouvelle catégorie à saisir
 - le mode du formulaire à save
 - toutes les catégories

```
@RequestMapping(value="/supprimer")  
public String supprimer(Long codeCat,Model model){  
    service.deleteCategorie(codeCat);  
    model.addAttribute("categorie", new Categorie());  
    model.addAttribute("action", "save");  
    model.addAttribute("categories", service.listCategories());  
    return "categories";  
}
```

Action : GET categories/supprimer?codeCat=xx

L'action categories/supprimer?codeCat=xx est associée à :

- ✓ l'exécution de la méthode supprimer
- ✓ Cette méthode récupère le paramètre URL codeCat
- ✓ Supprimer la catégorie de la base de données
- ✓ Avant de revenir à la vue, on charge à nouveau dans le modèle :
 - Une nouvelle catégorie à saisir
 - le mode du formulaire à save
 - toutes les catégories

```
@RequestMapping(value="/supprimer")  
public String supprimer(Long codeCat,Model model){  
    service.deleteCategorie(codeCat);  
    model.addAttribute("categorie", new Categorie());  
    model.addAttribute("action", "save");  
    model.addAttribute("categories", service.listCategories());  
    return "categories";  
}
```

Action : GET categories/editer?codeCat=xx

L'action categories/editer?codeCat=xx est associée à :

- ✓ l'exécution de la méthode editer
- ✓ Cette méthode récupère le paramètre URL codeCat
- ✓ Charge la catégorie dans le modèle
- ✓ Avant de revenir à la vue, on charge à nouveau dans le modèle :
 - le mode du formulaire à edit
 - Toutes les catégories

```
@RequestMapping(value="/editer")  
public String editer(Long codeCat,Model model){  
    model.addAttribute("categorie",  
        service.getCategorie(codeCat));  
    model.addAttribute("action", "edit");  
    model.addAttribute("categories", service.listCategories());  
    return "categories";  
}  
}
```

Gestion des produits

- Nous allons définir un contrôleur et une vue JSP qui permet de gérer les produits:
 - Formulaire de saisie
 - Ajout d'un produit avec la validation des données saisies dans le formulaire.
 - Editer un produit
 - Modifier un produit
 - Supprimer un produit

The screenshot shows a Java Swing application window titled "Produits". The URL in the browser bar is "localhost:8080/web/produits/chercher?motCle=". The window contains a search bar with "Mot Clé : []" and a "chercher" button. Below the search bar is a form with fields: REF (text input), Désignation (text input), Catégorie (dropdown menu set to "Ordinateurs"), Prix (text input "0.0"), Quantité (text input "0"), and Disponible (checkbox checked). A "save" button is located below the form. At the bottom is a table with columns: REF, DES, PRIX, QUANTITE, and DISPO. The table contains three rows of data:

REF	DES	PRIX	QUANTITE	DISPO
C	D	0.0	0	Supprimer Editer
MMM78	12345	89.0	3	Supprimer Editer
PK34	DES34	900.0	5	Supprimer Editer

ProduitController

```
package org.bp.web;

import java.util.List;import javax.validation.Valid; import org.bp.dao.entities.*;
import org.bp.service.ICatalogueService; import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller; import org.springframework.ui.Model;
import org.springframework.validation.BindingResult; import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/produits")
public class ProduitsController {

    @Autowired
    private ICatalogueService service;

    @RequestMapping(value="/index")
    public String index(Model model){
        model.addAttribute("mode", "insert");
        return "produits";
    }

    @RequestMapping(value="/chercher")
    public String chercher(String motCle,Model model){
        model.addAttribute("motCle", motCle);
        model.addAttribute("produits", service.produitsParMC(motCle));
        model.addAttribute("mode", "insert");
        return "produits";
    }
}
```

ProduitController

```
@ModelAttribute("produit")
public Produit produit(){ return new Produit(); }

@ModelAttribute("categories")
public List<Categorie> categories(){ return service.listCategories();}

@RequestMapping(value="save")
public String save(@Valid Produit p,BindingResult bindingResult,String
mode,Model model){

if(bindingResult.hasErrors()){ return "produits"; }

if(mode.equals("insert"))
    service.addProduit(p, p.getCategorie().getCodeCategorie());
else service.updateProduit(p);

model.addAttribute("produits",
service.produitsParCat(p.getCategorie().getCodeCategorie()));

model.addAttribute("mode", "insert");
return "produits";
}
```

ProduitController

```
@RequestMapping("/supprimer")
public String supprimer(String ref,Model model){
    service.deleteProduit(ref);
    model.addAttribute("produits",service.produitsParMC(""));
    model.addAttribute("mode", "insert");
    return "produits";
}

@RequestMapping("/editer")
public String editer(String ref,Model model){
    model.addAttribute("produit",service.getProduit(ref));
    model.addAttribute("mode", "edit");
    return "produits";
}
// Consulter les produit au format JSON
@RequestMapping(value="/produitsParMC",produces={"application/json"})
@ResponseBody
public List<Produit> produits(@RequestParam(value="motCle")String motCle){
    return service.produitsParMC(motCle);
} }
```



Vue : views/produits.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="f"%>
<!DOCTYPE html >
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Produits</title>
<link rel="stylesheet" type="text/css"
      href="<%request.getContextPath()%>/resources/css/style.css"/>
</head>
<body>
<div>
<form action="chercher">
    Mot Clé :<input type="text" name="motCle" value="${motCle}" />
    <input type="submit" value="chercher">
</form>
</div>
<div>
```



Vue : views/produits.jsp

```
<div>
    <f:form modelAttribute="produit" action="save">
        <table>
            <tr>
                <td>REF:</td> <td><f:input path="reference"/></td>
                <td><f:errors path="reference" cssClass="errors"></f:errors></td>
            </tr>
            <tr>
                <td>Désignation:</td> <td><f:input path="designation"/></td>
                <td><f:errors path="designation" cssClass="errors"></f:errors></td>
            </tr>
            <tr>
                <td>Catégorie:</td>
                <td><f:select path="categorie.codeCategorie" items="${categories}"
itemLabel="nomCategorie" itemValue="codeCategorie">
                    </f:select></td>
                <td><f:errors path="categorie.codeCategorie"
cssClass="errors"></f:errors></td>
            </tr>
        </table>
    </f:form>
</div>
```

Vue : views/produits.jsp

```
<tr>
    <td>Prix:</td><td><f:input path="prix"/></td>
    <td><f:errors path="prix" cssClass="errors"></f:errors></td>
</tr>

<tr>
    <td>Quantité:</td> <td><f:input path="quantite"/></td>
<td><f:errors path="quantite" cssClass="errors"></f:errors></td>
</tr>

<tr>
    <td>Disponible:</td><td><f:checkbox path="disponible"/></td>
<td><f:errors path="disponible" cssClass="errors"></f:errors></td>
</tr>

<tr>
    <td><input type="submit" value="save"/></td>
    <td><input type="hidden" name="mode" value="${mode}"></td>
    <td></td>
</tr>  </table>
</f:form>
</div>
```

Vue : views/produits.jsp

```
<table class="table1">
    <tr>
        <th>REF</th><th>DES</th><th>PRIX</th><th>QUANTITE</th><th>DISPO</th>
    </tr>
    <c:forEach items="${produits}" var="p">
        <tr>
            <td>${p.reference }</td><td>${p.designation }</td>
            <td>${p.prix }</td> <td>${p.quantite }</td>
            <td><a href="<%request.getContextPath()%>/produits/supprimer?ref=${p.reference}">Supprimer</a></td>
            <td><a href="<%request.getContextPath()%>/produits/editer?ref=${p.reference }">Editer</a></td>
        </tr>
    </c:forEach>
</table>
</div>
</body>
</html>
```

Ecrans de l'application

Mot Clé : chercher

REF:

Désignation:

Catégorie:

Prix:

Quantité:

Disponible:

save

REF	DES	PRIX	QUANTITE	DISPO
C	D	0.0	0	Supprimer Editer
MMM78	12345	89.0	3	Supprimer Editer
PK34	DES34	900.0	5	Supprimer Editer

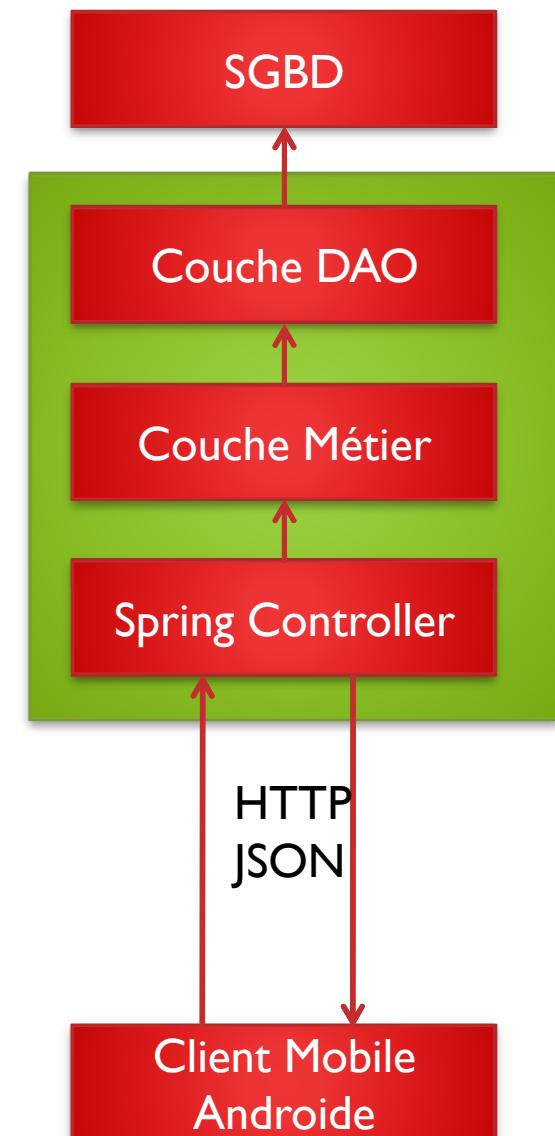
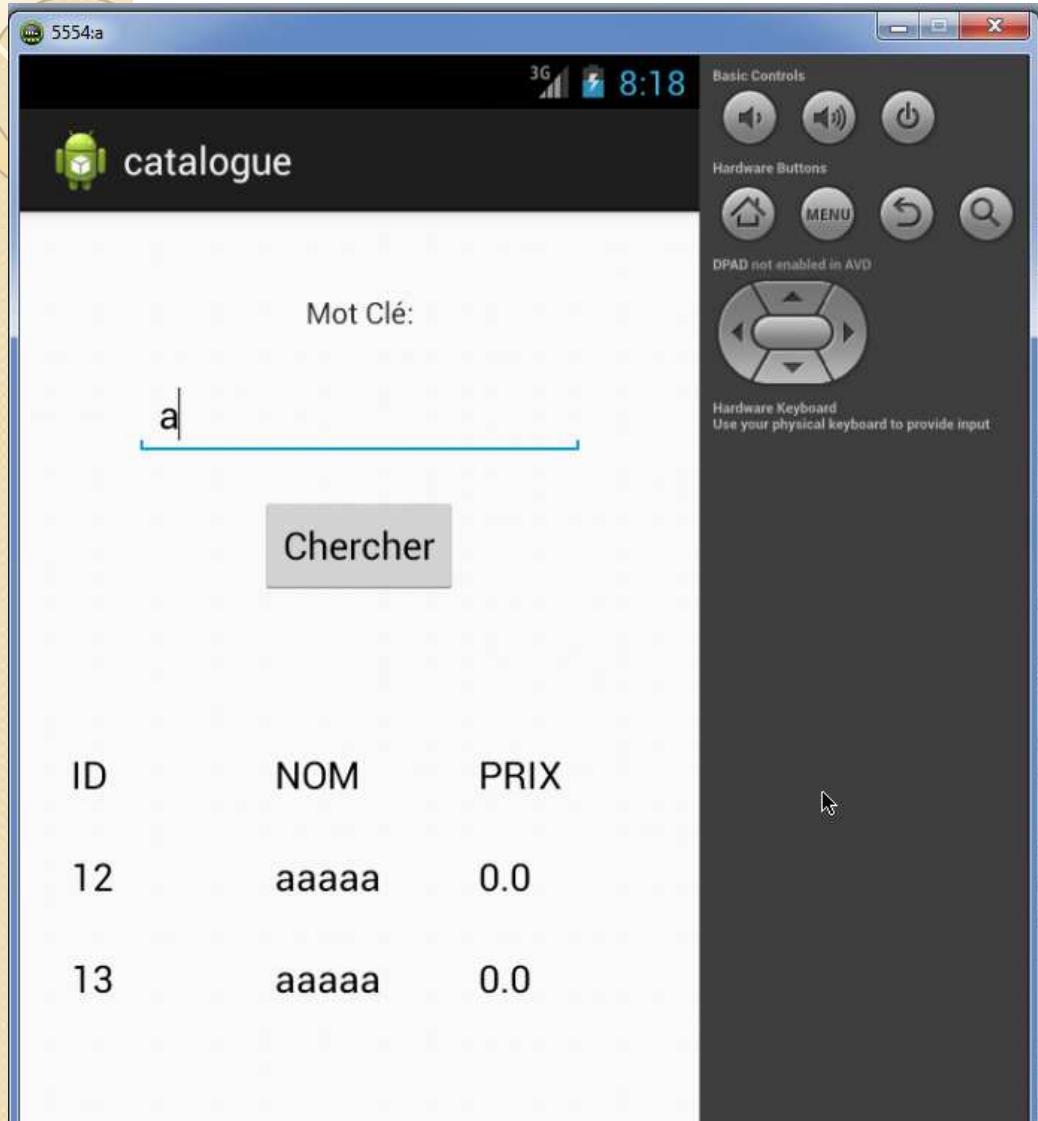
Dans l'entité Categorie.java
Pour ne pas sérialiser la photo et
Les produits d'une catégorie

```
@XmlTransient  
@JsonIgnore  
public Collection<Produit>  
getProduits() {  
    return produits;  
}  
  
@XmlTransient  
@JsonIgnore  
public byte[] getPhoto() {  
    return photo;  
}
```

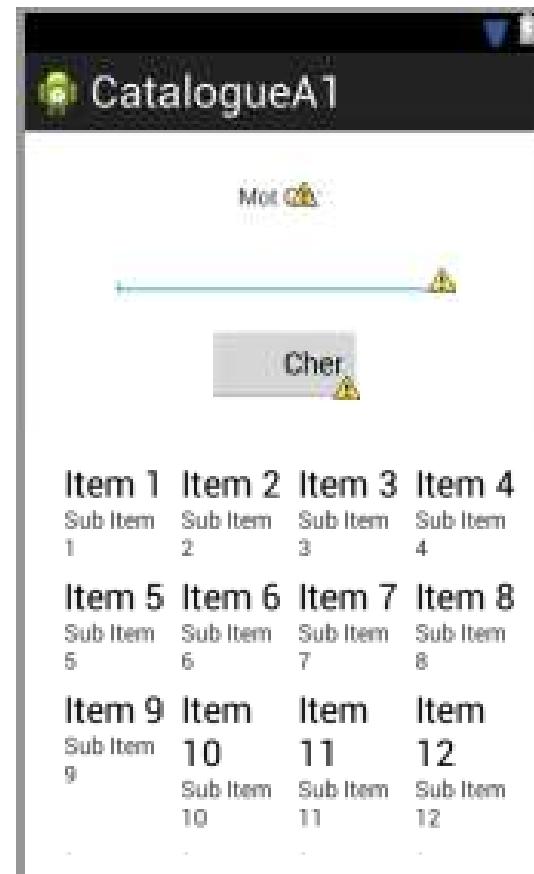
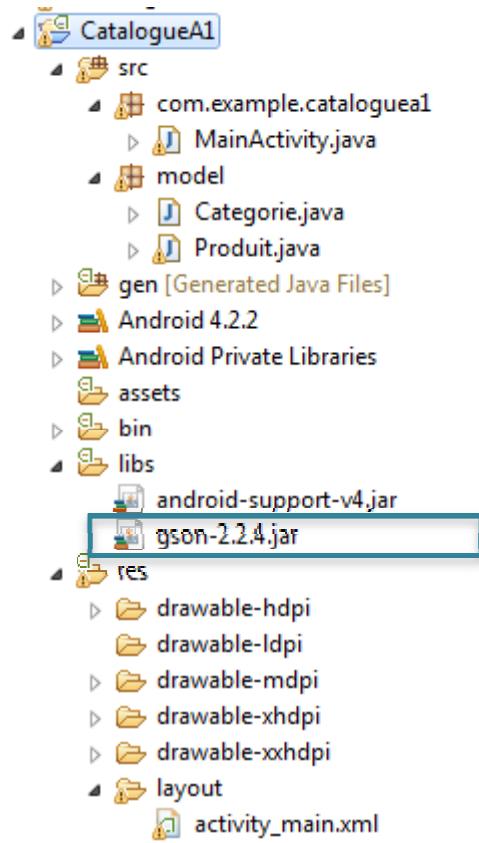
localhost:8080/web/produits/produitsParMC?motCle=D

```
[{"reference": "C", "designation": "D", "prix": 0.0, "quantite": 0, "disponible": false, "categorie": {"codeCategorie": 1, "nomCategorie": "Ordinateurs"}}, {"reference": "PK34", "designation": "DES34", "prix": 900.0, "quantite": 5, "disponible": true, "categorie": {"codeCategorie": 1, "nomCategorie": "Ordinateurs"}}]
```

Client Androïde



Structure du projet



Entités Produit : Mappée par GJSON

```
package model;  
import java.io.Serializable;  
import com.google.gson.annotations.SerializedName;  
public class Produit implements Serializable {  
    @SerializedName("reference")  
    public String reference;  
    @SerializedName("designation")  
    public String designation;  
    @SerializedName("prix")  
    public double prix;  
    @SerializedName("quantite")  
    public int quantite;  
    @SerializedName("disponible")  
    public boolean disponible;  
    public Categorie categorie;  
}
```

Entité Catégorie : Mappée par GJSON

```
package model;

import com.google.gson.annotations.SerializedName;

public class Catégorie {
    @SerializedName("codeCatégorie")
    public Long codeCatégorie;
    @SerializedName("nomCatégorie")
    public String nomCatégorie;
}
```

Activité

```
package com.example.cataloguea1;
import java.io.IOException; import java.io.InputStream;
import java.io.InputStreamReader;import java.io.Reader;
import java.util.List; import model.Produit;
import org.apache.http.HttpEntity; import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus; import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import com.google.gson.Gson;
import android.os.Bundle; import android.os.StrictMode;
import android.os.StrictMode.ThreadPolicy; import android.app.Activity;
import android.util.Log; import android.view.Menu;
import android.view.View; import android.view.View.OnClickListener;
import android.widget.ArrayAdapter; import android.widget.Button;
import android.widget.EditText;import android.widget.GridView;
import android.widget.Toast;
```

Activité

```
public class MainActivity extends Activity implements OnClickListener {  
    private Button buttonChercher;  
    private GridView gridViewProduits;  
    private EditText editTextMC;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        buttonChercher=(Button) findViewById(R.id.buttonChercher);  
        gridViewProduits=(GridView) findViewById(R.id.gridViewProduits);  
        editTextMC=(EditText) findViewById(R.id.editTextMC);  
        buttonChercher.setOnClickListener(this);  
        StrictMode.ThreadPolicy threadPolicy=new  
            StrictMode.ThreadPolicy.Builder().permitAll().build();  
        StrictMode.setThreadPolicy(threadPolicy);  
    }  
}
```

Activité

```
@Override
public void onClick(View v) {
try{
    String mc =editTextMC.getText().toString();
    InputStream is=
        getStream("http://192.168.1.53:8080/web/produits/produitsParMC?motCle="+mc);
    Gson gson=new Gson();
    Reader reader=new InputStreamReader(is);
    Produit[] produits=gson.fromJson(reader, Produit[].class);
    String[] data=new String[4*produits.length];
    int index=-1;
    for(Produit p:produits){
        data[++index]=p.reference; data[++index]=p.designation;
        data[++index]=String.valueOf(p.prix); data[++index]=String.valueOf(p.quantite);
    }
    ArrayAdapter<String> adapter=new ArrayAdapter<String>
(this,android.R.layout.simple_list_item_1,data);
    gridViewProduits.setAdapter(adapter);
} catch (Exception e){ e.printStackTrace(); }
}
```

Activité : la méthode getStream()

```
private InputStream getStream(String url) {  
    DefaultHttpClient client = new DefaultHttpClient();  
    HttpGet getRequest = new HttpGet(url);  
    try {  
        HttpResponse getResponse = client.execute(getRequest);  
        final int statusCode = getResponse.getStatusLine().getStatusCode();  
        if (statusCode != HttpStatus.SC_OK) {  
            Log.w(getClass().getSimpleName(),  
                  "Error " + statusCode + " for URL " + url);  
            return null;  
        }  
        HttpEntity getResponseEntity = getResponse.getEntity();  
        return getResponseEntity.getContent();  
    }  
    catch (IOException e) {  
        getRequest.abort();  
        Log.w(getClass().getSimpleName(), "Error for URL " + url, e);  
    }  
    return null;  
}
```



Integration Spring JaxWS

med@youssf.net

Web Service SOAP

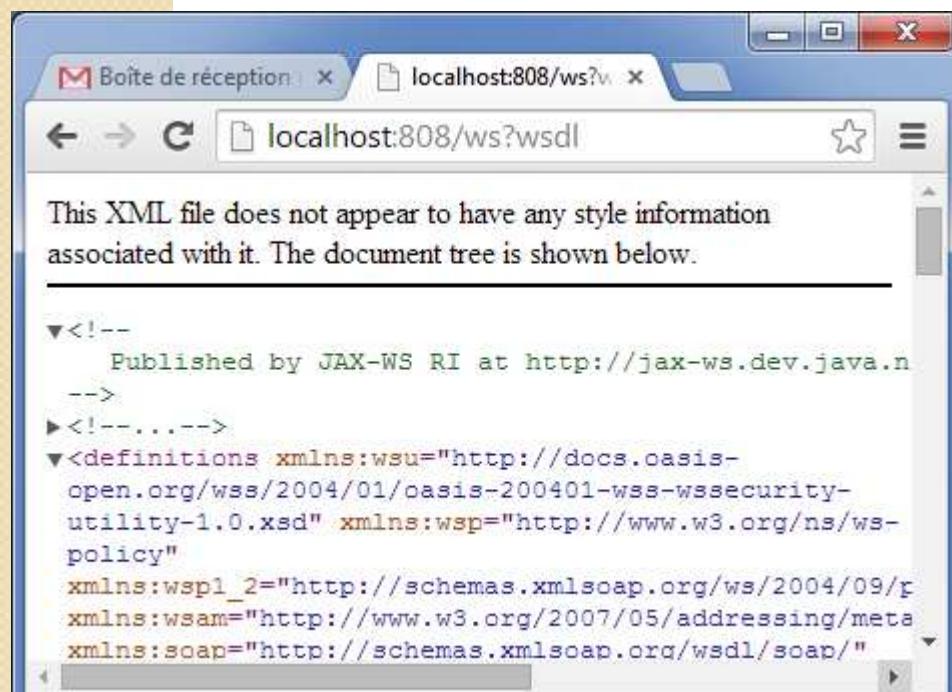
```
package org.bp.ws;
import java.util.List;
import javax.jws.WebParam;
import javax.jws.WebService;
import org.bp.dao.entities.Categorie;
import org.bp.dao.entities.Produit;
import org.bp.service.ICatalogueService;
import org.springframework.beans.factory.annotation.Autowired;
@WebService
public class CatalogueWS {
    @Autowired
    private ICatalogueService service;

    public List<Categorie> getAllCategories(){
        return service.listCategories();
    }
    public List<Produit> produitsParCat(@WebParam(name="codeCat")Long codeCat){
        return service.produitsParCat(codeCat);
    }
}
```

Déployer un web service avec Spring

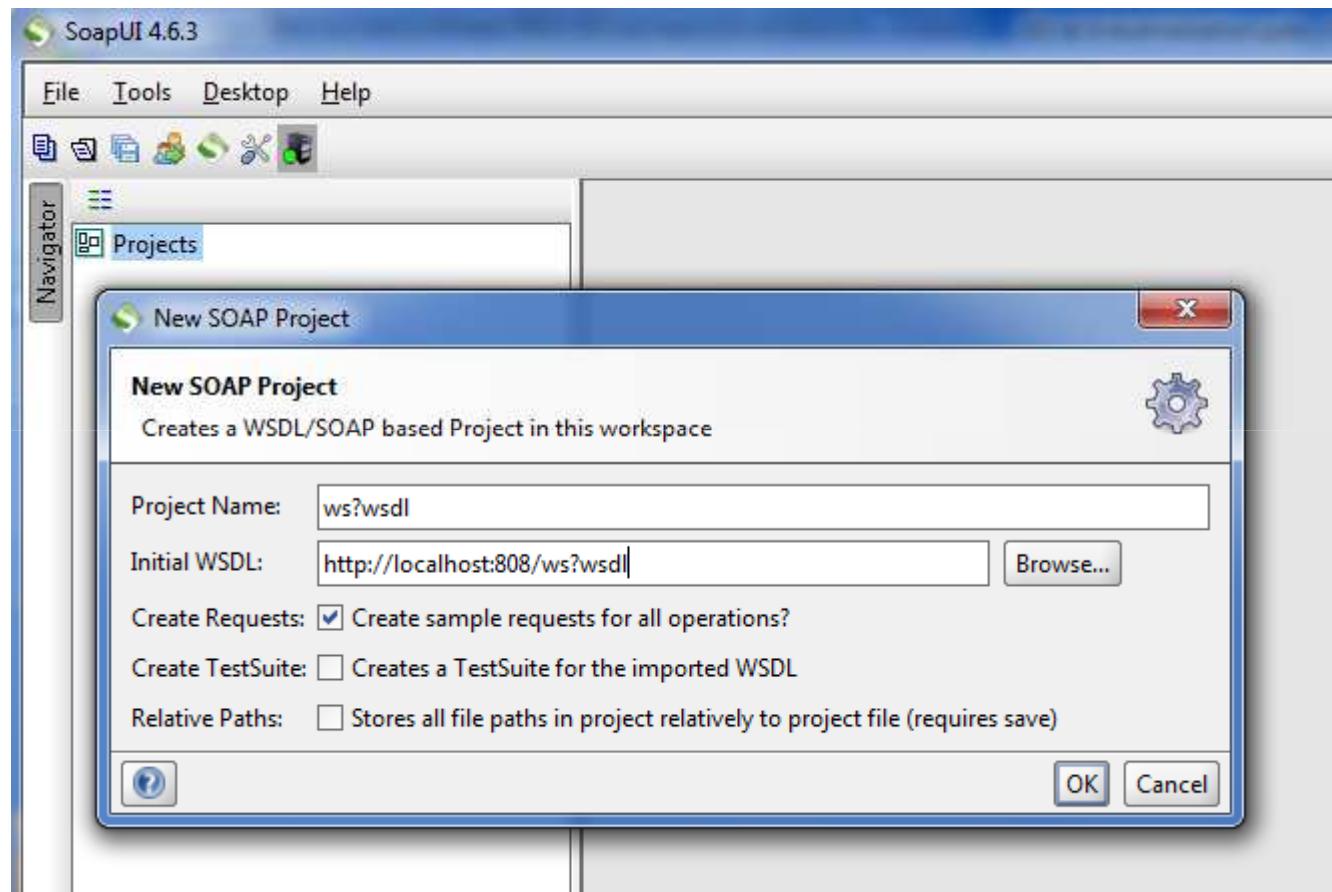
root-context.xml

```
<bean id="ws" class="org.bp.ws.CatalogueWS"></bean>
<bean id="jws"
      class="org.springframework.remoting.jaxws.SimpleJaxWsServiceExporter">
    <property name="baseAddress" value="http://localhost:808/ws"></property>
</bean>
```



med@youssfi.net

Test du Web service



Test du Web service

Screenshot of SoapUI 4.6.3 interface showing a successful test run of a Web service.

The interface includes:

- Toolbar:** File, Tools, Desktop, Help, and various icons for navigation and testing.
- Search Forum:** Search bar at the top right.
- Navigator:** Shows the project structure under "Projects". A "CatalogueWSPortBinding" port has two operations: "getAllCategories" and "produitsParCat". Each operation has a "Request 1" entry.
- Request 1:** The main workspace shows the request and response XML side-by-side.
 - Request (Left):** XML message sent to `http://localhost:808/ws`. It contains a `<soapenv:Envelope>` with a `<soapenv:Body>` containing a `<ws:getAllCategories/>`.
 - Response (Right):** XML message received from the server. It contains an `<S:Envelope>` with an `<S:Body>` containing a `<ns2:getAllCategoriesResponse>`. The response body includes two `<return>` elements, each containing a category:
 - `<codeCategorie>1</codeCategorie>` and `<nomCategorie>Ordinateurs</nomCategorie>`
 - `<codeCategorie>5</codeCategorie>` and `<nomCategorie>Téléphones</nomCategorie>`
- Request Properties:** Table showing properties for the current request.

Property	Value
Name	Request 1
Description	
Message Size	217
Encoding	UTF-8
- Log:** Buttons for SoapUI log, http log, jetty log, error log, wsrm log, and memory log.
- System Tray:** Icons for Windows, Internet Explorer, Google Chrome, Java, and other system utilities.
- Status Bar:** FR, 08:24, 06/05/2014.

Test du Web service

SoapUI 4.6.3

File Tools Desktop Help

Search Forum

Navigator

Projects ws?wsdl CatalogueWSPortBinding getAllCategories Request1 produitsParCat Request1

Request 1 http://localhost:808/ws

Raw XML

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope">
  <soapenv:Header>
  <soapenv:Body>
    <ws:produitsParCat>
      <!--Optional:-->
      <codeCat>1</codeCat>
    </ws:produitsParCat>
  </soapenv:Body>
</soapenv:Envelope>
```

Raw XML

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope">
  <S:Body>
    <ns2:produitsParCatResponse xmlns:ns2="http://ws.bp.<!--Optional:-->
      <return>
        <categorie>
          <codeCategorie>1</codeCategorie>
          <nomCategorie>Ordinateurs</nomCategorie>
        </categorie>
        <designation>D</designation>
        <disponible>false</disponible>
        <prix>0.0</prix>
        <quantite>0</quantite>
        <reference>C</reference>
      </return>
      <return>
        <categorie>
          <codeCategorie>1</codeCategorie>
          <nomCategorie>Ordinateurs</nomCategorie>
        </categorie>
        <designation>12345</designation>
        <disponible>true</disponible>
        <prix>89.0</prix>
        <quantite>3</quantite>
      </return>
    </ns2:produitsParCatResponse>
  </S:Body>
</S:Envelope>
```

Request Properties

Property	Value
Name	Request1
Description	
Message Size	300
Encoding	UTF-8

Properties

A... Head... Attachme... W... WS... JMS He... JMS Prop...

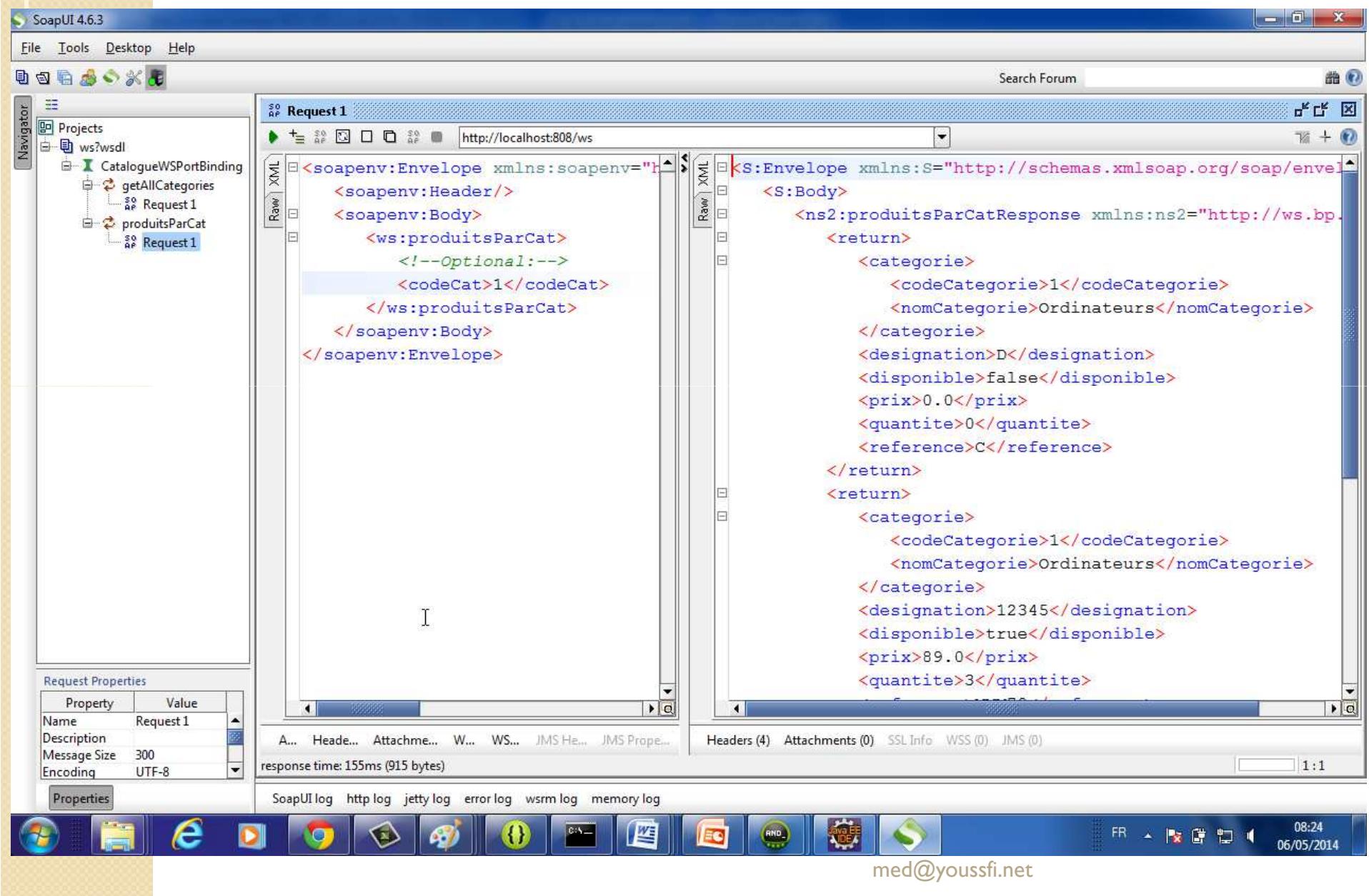
response time: 155ms (915 bytes)

Headers (4) Attachments (0) SSL Info WSS (0) JMS (0)

SoapUI log http log jetty log error log wsrm log memory log

FR 08:24 06/05/2014

med@youssfi.net





Intégration Spring avec RMI

Service RMI : Interface Remote

```
package org.bp.rmi; import java.rmi.Remote; import java.util.List;
import javassist.tools.rmi.RemoteException; import org.bp.dao.entities.Categorie;
import org.bp.dao.entities.Produit;
public interface ICatalogueRemote extends Remote {
    public void newCategorie(Categorie c) throws RemoteException;
    public void newProduit(Produit p,Long codeCat) throws RemoteException;
    public List<Categorie> getAllCategories() throws RemoteException;
    public List<Produit> getProduitsParCat(Long codecat) throws RemoteException;
}
```

Service RMI : Implémentation

```
package org.bp.rmi;

import java.util.List; import javassist.tools.rmi.RemoteException;
import org.bp.dao.entities.*; import org.bp.service.ICatalogueService;
import org.springframework.beans.factory.annotation.Autowired;

public class CatalogueRMIService implements ICatalogueRemote{

    @Autowired
    private ICatalogueService service;

    @Override
    public void newCategorie(Categorie c) throws RemoteException {
        service.addCategorie(c);
    }

    @Override
    public void newProduit(Produit p, Long codeCat) throws RemoteException {
        service.addProduit(p, codeCat);
    }

    @Override
    public List<Categorie> getAllCategories() throws RemoteException {
        return service.listCategories();
    }

    @Override
    public List<Produit> getProduitsParCat(Long codecat) throws RemoteException {
        return service.produitsParCat(codecat);
    }
}
```

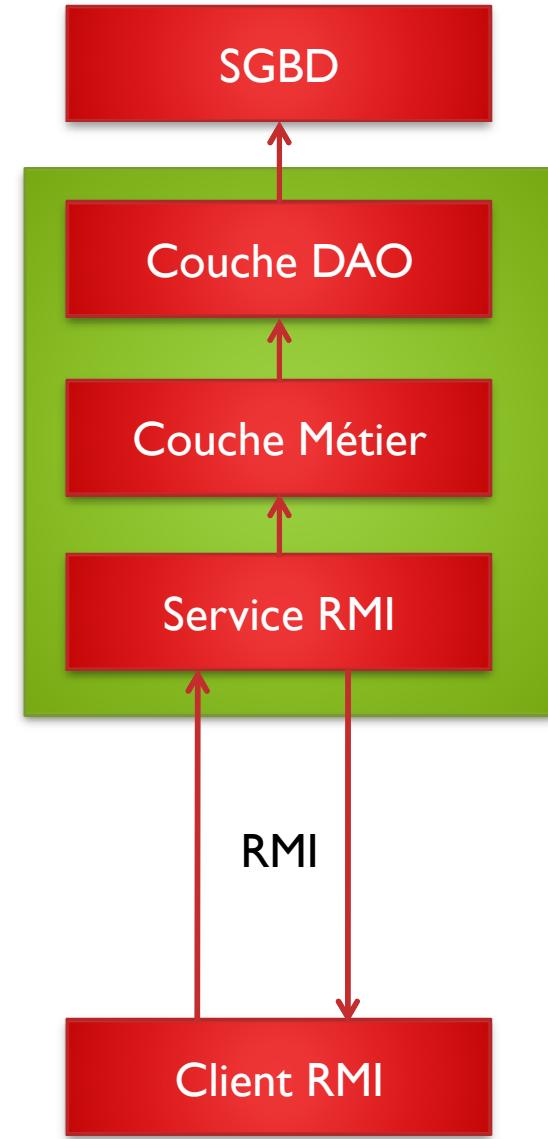
Déployer le service RMI avec Spring

```
<bean id="rmiCat" class="org.bp.rmi.CatalogueRMIService"></bean>
<bean class="org.springframework.remoting.rmi.RmiServiceExporter">
    <property name="serviceName" value="CATAL"></property>
    <property name="service" ref="rmiCat"></property>
    <property name="registryPort" value="1099"></property>
    <property name="serviceInterface"
        value="org.bp.rmi.ICatalogueRemote"></property>
</bean>
```

Client RMI

```
import java.rmi.Naming;
import java.util.List;
import org.bp.dao.entities.Categorie;
import org.bp.rmi.ICatalogueRemote;
public class ClientRMI {

    public static void main(String[] args) {
        try {
            ICatalogueRemote stub=(ICatalogueRemote)
                Naming.lookup("rmi://localhost:1099/CATAL");
            List<Categorie> cats=stub.getAllCategories();
            for(Categorie c:cats){
                System.out.println(c.getNomCategorie());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



Upload d'un fichier Format Excel qui contient les produits

http://localhost:8080/catalogue/saveUploadedFile

Fichier Excel (Format.xls): desktop\ex\Classeur1.xls | Parcourir...

Content

Nom	Prix
Aaaaa	600.0
Bbbbb	600.0
Ccccc	12.0
Dddddd	78.0
Eeeee	12.0

Classeur.xls

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						

Nom Prix

Aaaaa 600
Bbbbb 600
Ccccc 12
Dddddd 78
Eeeee 12

Dépendances

```
<!-- Apache Commons Upload -->
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.2.2</version>
</dependency>

<!-- Apache Commons Upload -->
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>1.3.2</version>
</dependency>
<!-- Apache Excel -->
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>3.7</version>
</dependency>
```

Classe UploadedFile

```
package ma.enset.catalogue.controllers;

import org.springframework.web.multipart.commons.CommonsMultipartFile;

public class UploadedFile {
    private CommonsMultipartFile file;

    public CommonsMultipartFile getFile() {
        return file;
    }

    public void setFile(CommonsMultipartFile file) {
        this.file = file;
    }
}
```

Controleur

```
package ma.enset.catalogue.controllers;

import java.io.File;import java.io.FileOutputStream;
import java.util.ArrayList;import java.util.HashMap;
import java.util.List;import java.util.Map;

import javax.servlet.http.HttpServletRequest;import javax.servlet.http.HttpServletResponse;

import org.apache.poi.hssf.usermodel.HSSFRow;import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.HandlerExceptionResolver;
import org.springframework.web.servlet.ModelAndView;

import ma.enset.catalogue.entities.Produit;
import ma.enset.catalogue.metier.ICatalogueMetier;
```

@Controller

```
public class UploadProduitsController implements HandlerExceptionResolver {

    @Autowired
    private ICatalogueMetier metier;
    @RequestMapping(value="/formUpload")
    public String formUpload(@ModelAttribute(value="form") UploadedFile form){
        return "formUpload";
    }
}
```

Contrôleur : Sauvegarde du fichier

```
@RequestMapping(value="/saveUploadedFile")
public String saveUploadedFile(
    @ModelAttribute(value="form") UploadedFile form,
    BindingResult bindingResult,Model model){
    if(!bindingResult.hasErrors()){
        try {
            // Enregistrement du fichier
String filePath=
    System.getProperty("java.io.tmpdir")+"/"+form.getFile().getOriginalFi
lename();
FileOutputStream outputStream=new FileOutputStream(new File(filePath));
outputStream.write(form.getFile().getFileItem().get());
outputStream.close();
System.out.println(form.getFile().getSize());
```

Contrôleur : Traitement du fichier Excel

```
// Interprétation du fichier Excel

HSSFWorkbook workbook=new HSSFWorkbook(form.getFile().getInputStream());
    HSSFSheet f1=workbook.getSheet("Feuill1");
    int l1=f1.getFirstRowNum();
    int l2=f1.getLastRowNum();
    List<Produit> produits=new ArrayList<Produit>();
    for(int i=l1+1;i<=l2;i++){
        Produit p=new Produit();
        HSSFRow r1=f1.getRow(i);
        int n1=r1.getFirstCellNum();
        p.setNomProduit(r1.getCell(n1).getStringCellValue());
        p.setPrix(r1.getCell(n1+1).getNumericCellValue());
        metier.addProduit(p);
        produits.add(p);
    }
    model.addAttribute("produits", produits);

} catch (Exception e) {
throw new RuntimeException(e);
}
return "formUpload";
}
```



Contrôleur : Gestion des Exceptions

```
@Override  
public ModelAndView resolveException(HttpServletRequest request,  
HttpServletResponse response, Object o, Exception e) {  
    Map<Object, Object> model=new HashMap<Object, Object>();  
    model.put("errors", e.getMessage());  
    model.put("form", new UploadedFile());  
    return new ModelAndView("formUpload", (Map)model);  
}  
}
```

Vue : formUpload.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="f" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <f:form commandName="form" action="saveUploadedFile" enctype="multipart/form-data"
    method="post">
        <table>
            <tr>
                <td>Fichier Excel (Format XLS):</td>
                <td><f:input type="file" path="file"/>
                <td><f:errors path="*"/></f:errors>${errors}</td>
            </tr>
            <tr> <td><input type="submit" value="Upload"/></td> </tr>
        </table>
    </f:form>
```

Vue : formUpload.jsp

```
<h3>Content</h3>

<table border="1">
  <tr>
    <th>Nom</th><th>Pri</th>
  </tr>
  <c:forEach items="${produits}" var="p">
    <tr>
      <td>${p.nomProduit}</td>
      <td>${p.prix}</td>
    </tr>
  </c:forEach>
</table>

</body>
</html>
```

Limitation de la taille des fichiers uploadé

Fichier : /WEB-INF/spring/appServlet/**servlet-context.xml**

```
<beans:bean  
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver"  
    id="multipartResolver">  
    <beans:property name="maxUploadSize" value="100000"></beans:property>  
</beans:bean>
```



SPRING SECURITY

med@youssf.net



Maven Dependencies : Spring Security

```
<!-- Spring Security -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>3.2.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>3.2.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>3.2.0.RELEASE</version>
</dependency>
</dependencies>
```

web.xml

- Déclarer le filtre DelegatingFilterProxy dans le fichier web.xml
- Toutes les requêtes HTTP passent par ce filtre.
- Le nom du filtre est : **springSecurityFilterChain**
- Ce nom devrait correspondre au nom d'un bean spring qui sera déployé par ContextLoaderListener et qui contient les règles de sécurité à exécuter.

```
<!-- Spring Security -->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-
        class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

Configuration Spring security

```
<s:http>
    <s:intercept-url pattern="/produits/**" access="ROLE_ADMIN_PROD"/>
    <s:intercept-url pattern="/categories/**" access="ROLE_ADMIN_CAT"/>
    <s:form-login login-page="/Login" default-target-url="/produits/index"
        authentication-failure-url="/Login" />
    <s:logout logout-success-url="/Login" />
</s:http>
<s:authentication-manager>
    <s:authentication-provider>
        <s:user-service>
            <s:user name="admin1" password="admin1"
authorities="ROLE_ADMIN_PROD"/>
            <s:user name="admin2" authorities="ROLE_ADMIN_CAT,ROLE_ADMIN_PROD"
password="admin2" />
        </s:user-service>
    </s:authentication-provider>
</s:authentication-manager>
```

LoginContrôleur

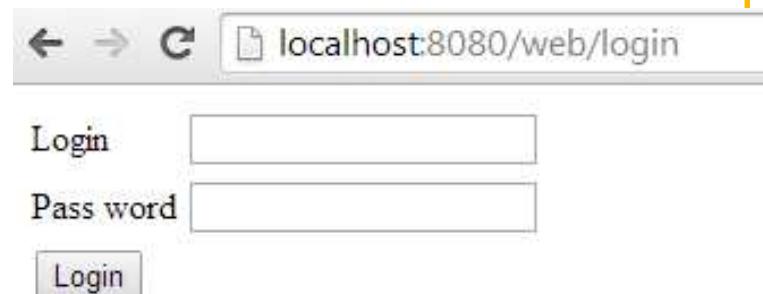
```
package org.bp.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class LoginController {
    @RequestMapping(value="/login")
    public String login(){
        return "index";
    }
}
```

login.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="j_spring_security_check" method="post">
<table>
<tr>
<td>Login</td>
<td><input type="text" name="j_username"></td>
</tr>
<tr>
<td>Pass word</td>
<td><input type="password" name="j_password"></td>
</tr>
<tr>
<td><input type="submit" value="Login"></td>
</tr>
</table>
</form>
</body>
</html>
```



Lien Logout

```
<div>
    <c:url value="/j_spring_security_logout" var="LogoutUrl" />
    <a href="#">Logout</a>
</div>
```

Utilisateurs dans la base de données

- Créer deux tables :
 - Users : qui contient les utilisateurs autorisés à accéder à l'application
 - Roles : qui contient les rôles de chaque utilisateur

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	<u>ID_USER</u>	int(11)			Non	Aucune	AUTO_INCREMENT
2	<u>username</u>	varchar(15)	latin1_swedish_ci		Non	Aucune	
3	<u>PASSWORD</u>	varchar(100)	latin1_swedish_ci		Non	Aucune	
4	<u>ACTIVED</u>	tinyint(1)			Non	Aucune	

Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
<u>ID_ROLE</u>	int(11)			Non	Aucune	AUTO_INCREMENT
<u>ID_USER</u>	int(11)			Non	Aucune	
<u>ROLE_NAME</u>	varchar(20)	latin1_swedish_ci		Non	Aucune	

<u>ID_USER</u>	<u>username</u>	<u>PASSWORD</u>	<u>ACTIVED</u>
1	admin1	e00cf25ad42683b3df678c61f42c6bda	1
2	admin2	c84258e9c39059a89ab77d846ddab909	1
3	user	ee11cbb19052e40b07aac0ca060c23ee	1

<u>ID_ROLE</u>	<u>ID_USER</u>	<u>ROLE_NAME</u>
1	1	ROLE_ADMIN_CAT
2	1	ROLE_ADMIN_PROD
3	2	ROLE_ADMIN_PROD
4	3	ROLE_USER

Base de données : Table Users

```
--  
-- Structure de la table `users`  
  
--  
CREATE TABLE IF NOT EXISTS `users` (  
    `ID_USER` int(11) NOT NULL AUTO_INCREMENT,  
    `username` varchar(15) NOT NULL,  
    `PASSWORD` varchar(100) NOT NULL,  
    `ACTIVED` tinyint(1) NOT NULL,  
    PRIMARY KEY (`ID_USER`),  
    UNIQUE KEY `LOGIN` (`username`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=4 ;  
  
--  
-- Contenu de la table `users`  
  
--  
INSERT INTO `users` (`ID_USER`, `username`, `PASSWORD`, `ACTIVED`) VALUES  
(1, 'admin1', 'e00cf25ad42683b3df678c61f42c6bda', 1),  
(2, 'admin2', 'c84258e9c39059a89ab77d846ddab909', 1),  
(3, 'user', 'ee11cbb19052e40b07aac0ca060c23ee', 1);
```

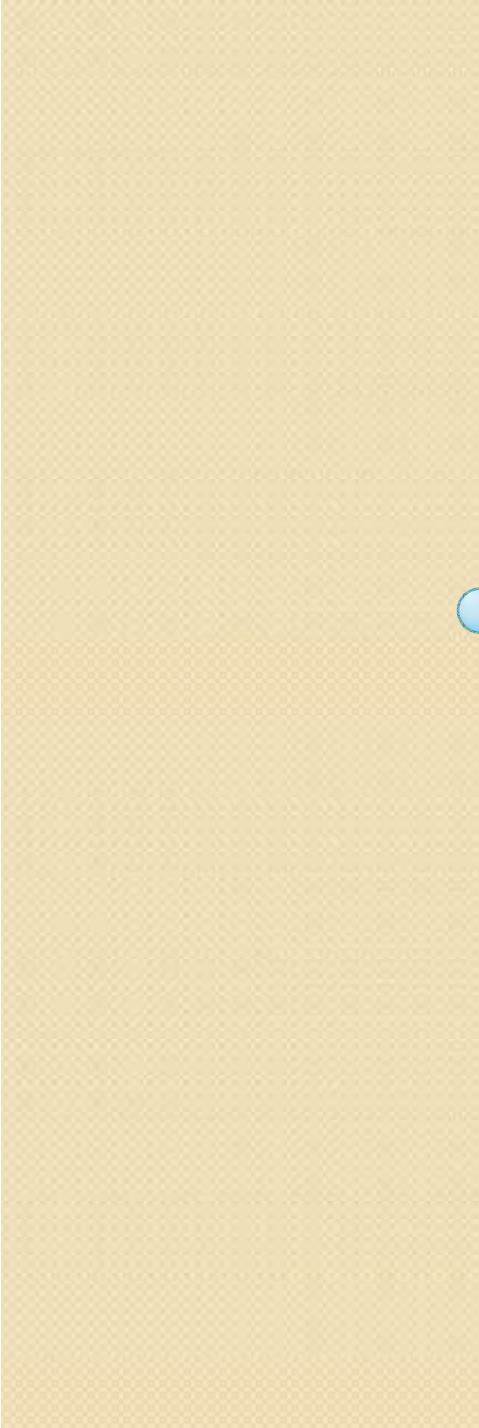
Base de données : Table roles

```
--  
-- Structure de la table `roles`  
  
--  
CREATE TABLE IF NOT EXISTS `roles` (  
    `ID_ROLE` int(11) NOT NULL AUTO_INCREMENT,  
    `ID_USER` int(11) NOT NULL,  
    `ROLE_NAME` varchar(20) NOT NULL,  
    PRIMARY KEY (`ID_ROLE`),  
    KEY `ID_USER` (`ID_USER`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=5 ;  
  
--  
-- Contenu de la table `roles`  
  
--  
INSERT INTO `roles` (`ID_ROLE`, `ID_USER`, `ROLE_NAME`) VALUES  
(1, 1, 'ROLE_ADMIN_CAT'),  
(2, 1, 'ROLE_ADMIN_PROD'),  
(3, 2, 'ROLE_ADMIN_PROD'),  
(4, 3, 'ROLE_USER');  
  
--  
-- Contraintes pour la table `roles`  
  
--  
ALTER TABLE `roles`  
    ADD CONSTRAINT `roles_ibfk_1` FOREIGN KEY (`ID_USER`) REFERENCES `users` (`ID_USER`);
```



Configuration Spring security

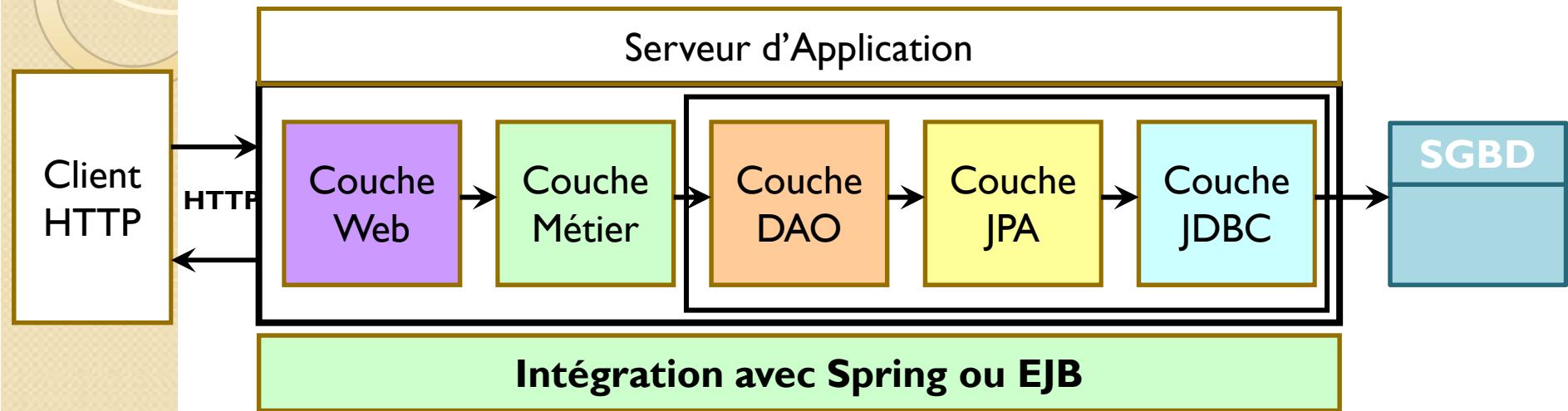
```
<s:authentication-manager>
    <s:authentication-provider>
        <s:password-encoder hash="md5"></s:password-encoder>
        <s:jdbc-user-service data-source-ref="dataSource"
            users-by-username-query="select user_name,password, activated
            from users where user_name=?"
            authorities-by-username-query="select u.user_name, r.roleName from users
            u, role r
            where u.user_id = r.user_id and u.user_name =? " />
    </s:authentication-provider>
</s:authentication-manager>
```



STRUTS FRAMEWORK

med@youssf.net

Architecture d'une application multi-couches J2EE



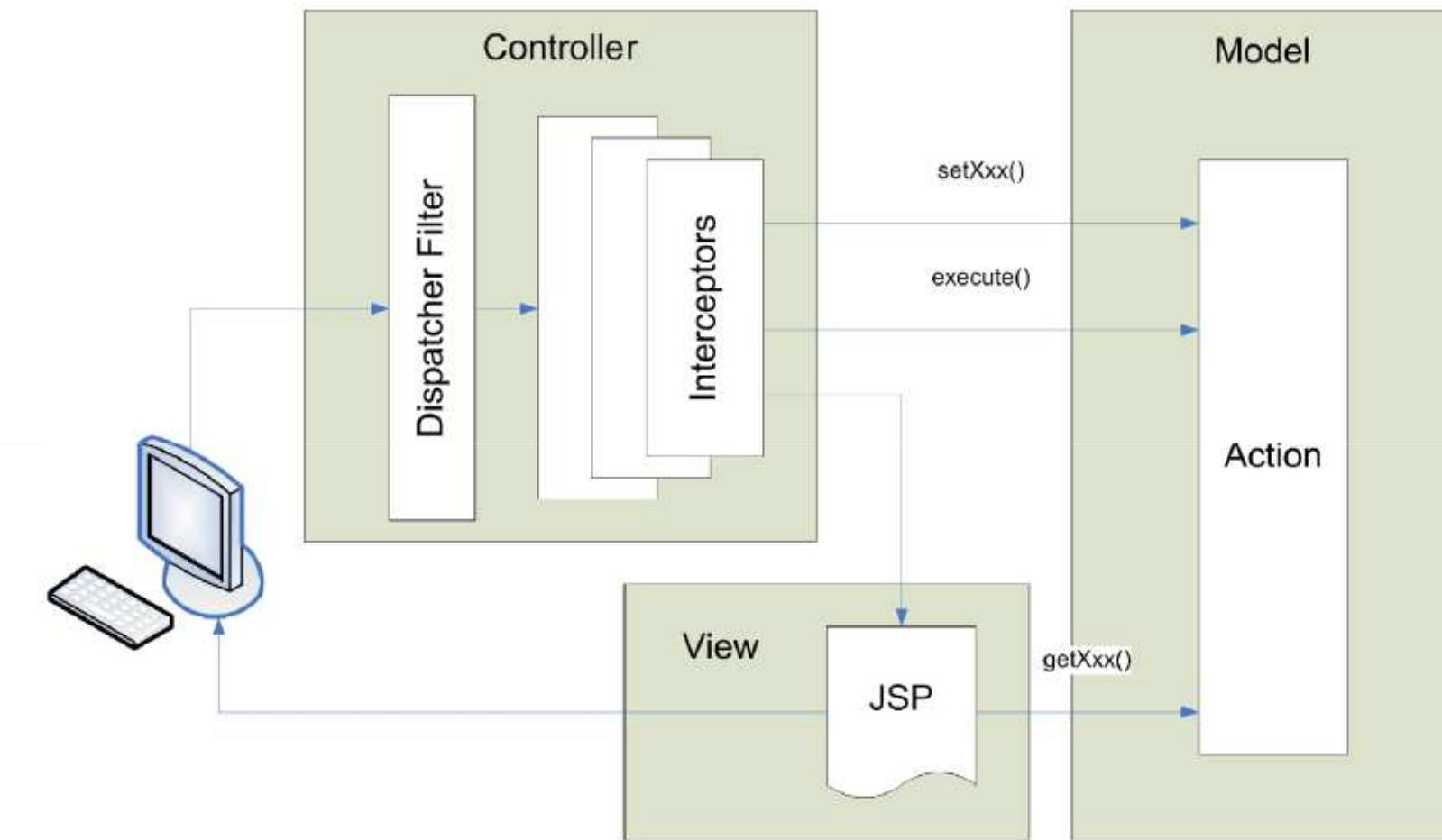
- la couche [**web**] est la couche en contact avec l'utilisateur en utilisant le protocole HTTP.
- la couche [**metier**] implémente les règles de gestion de l'application, tels que le calcul d'un salaire ou d'une facture. Cette couche utilise des données provenant de l'utilisateur via la couche [web] et du Sgbd via la couche [dao].
- la couche [**dao**] (Data Access Objects), la couche [**jpa**] (Java Persistence Api) et le pilote Jdbc gèrent l'accès aux données du Sgbd.
- l'intégration des couches peut être réalisée par un conteneur Spring ou Ejb3 (Enterprise Java Bean).



Struts2

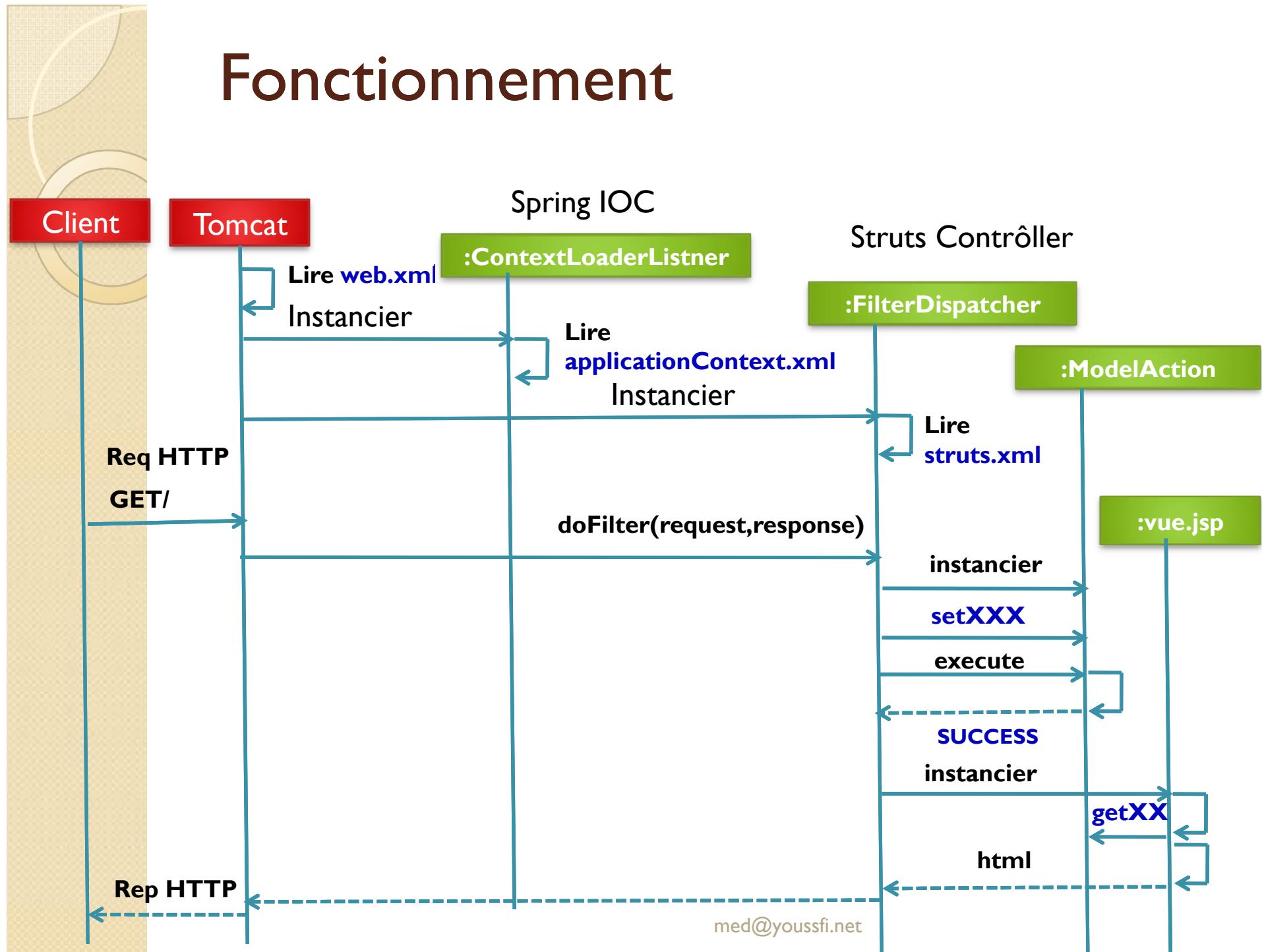
- **Apache Struts** est un framework libre
 - servant au développement d'applications web Java EE.
 - Il utilise et étend l'**API Servlet Java** afin d'encourager les développeurs à adopter l'architecture **Modèle-Vue-Contrôleur**.
- Struts 2 est un framework issu de Struts1 et de Webwork.
- En 2005 Webwork2.2 a été adopté comme moteur de Struts2.
- Struts 2 est donc un changement radical de Struts1.

Architecture MVC basée sur



- Couche Web basée sur Struts 2

Fonctionnement





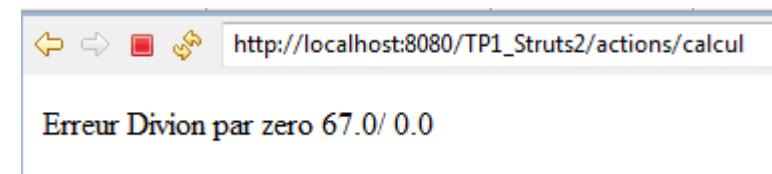
Fonctionnement

- Au démarrage du serveur d'application, le **Contrôleur** FilterDispatcher est instancié. FilterDispatcher devrait être déclaré dans le fichier web.xml.
- Ce dernier lit sa configuration à partir du fichier struts.xml et charge un proxy d'intercepteurs qui vont s'occuper de traiter les requêtes et les réponses avant et après l'exécution de l'action
- Toutes les requêtes HTTP sont destiné à FilterDispatcher.
- L'url de chaque requête contient le nom de l'action qu'il faut exécuter.
- La classe relative à cette action représente l' **Action**. C'est une classe qui peut hériter d'une classe **ActionSupport** fournie par struts. Elle déclare des attributs pour stocker les données de la requête et les résultats qui seront affichés (**Modèle**). Elle doit définir également une méthode execute (**public String execute() throws Exception**) qui contient le code à exécuter pour cette action. Le reste des méthodes sont les getters et setters Obligatoires.
- FilterDispatcher instancie la classe de L'action associée à l'URL, ensuite stocke les données de la requête dans cette instance , via les setters, puis fait appel à la méthode execute de cette action. La méthode execute retourne au contrôleur le nom de la vue qui sera appelée pour afficher les résultats.

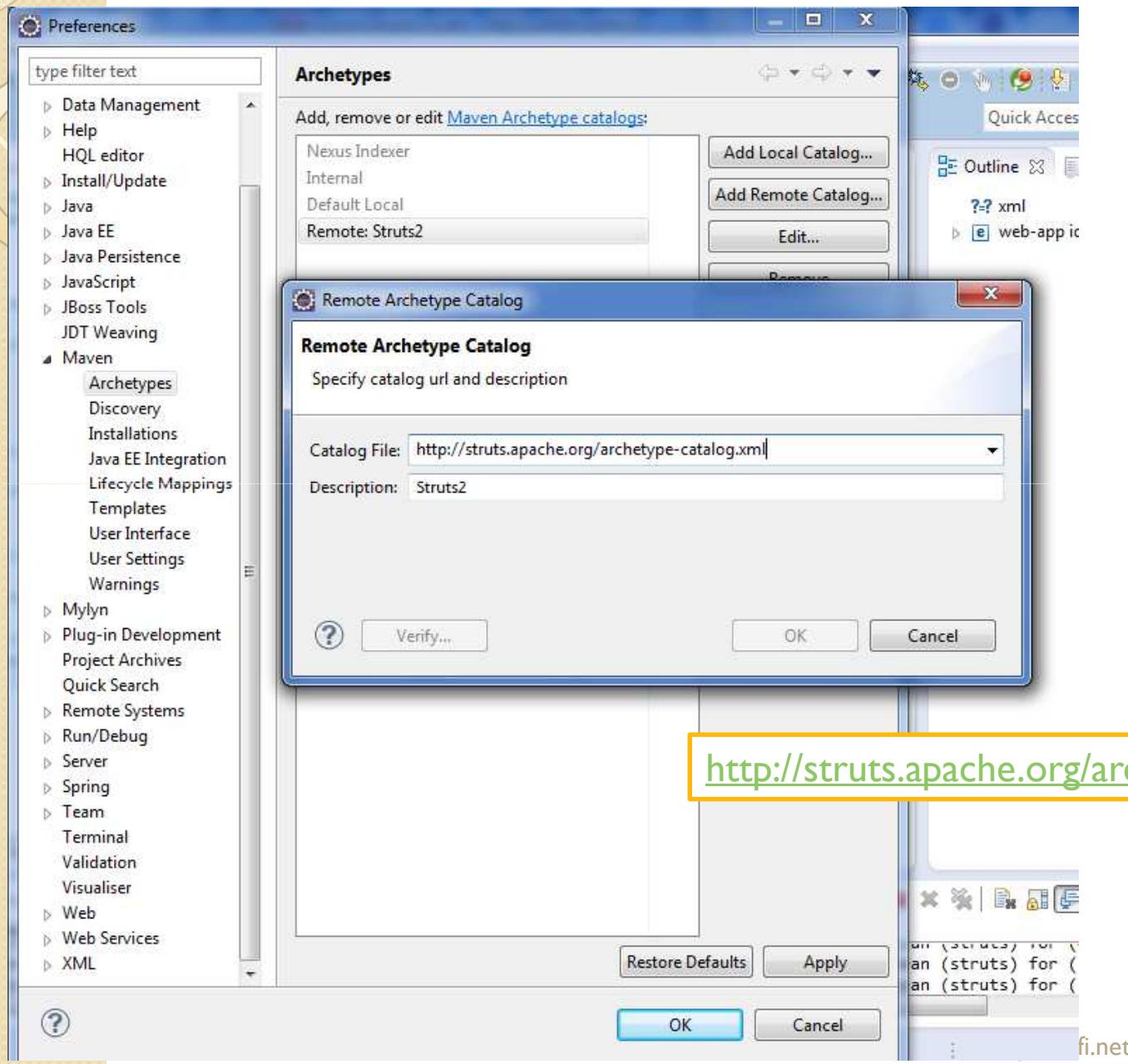
Premier Exemple

- Supposons que l'on souhaite créer une application web qui permet de saisir deux nombres v1 et v2 et d'afficher le rapport entre ces deux nombres $\text{res} = \text{v1}/\text{v2}$. Si V2 est nul une page des erreurs sera affichée.

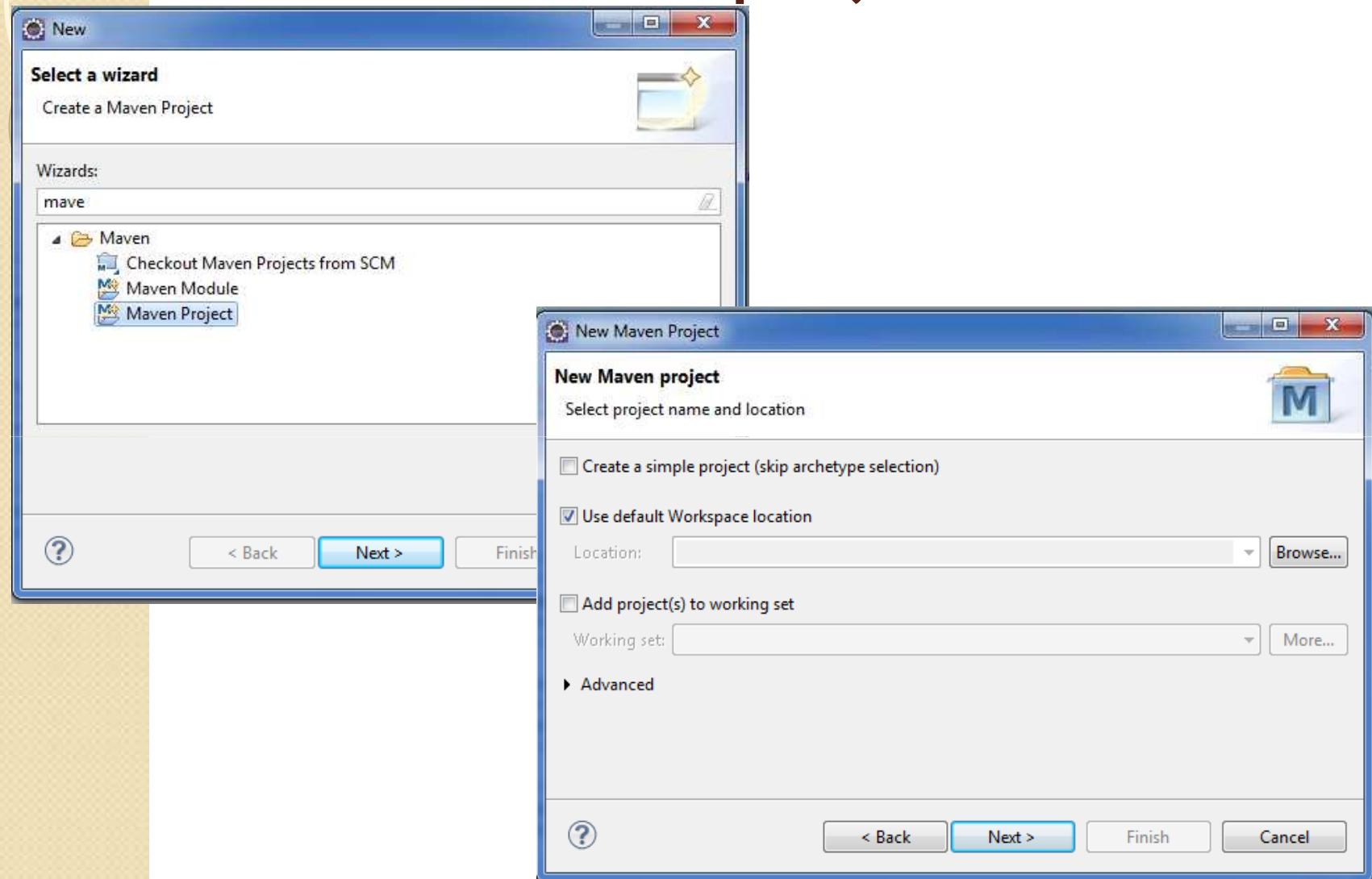
A screenshot of a web browser window. The address bar shows the URL `http://localhost:8080/TP1_Struts2/actions/calcul`. The page contains two input fields labeled "V1" and "V2" with values "67.0" and "6.0" respectively, and a button labeled "calcul". Below the inputs, the text "Résultat=11.16666666666666" is displayed.



L'ajout d'une nouvel catalogue Archetypes Maven relatif à struts

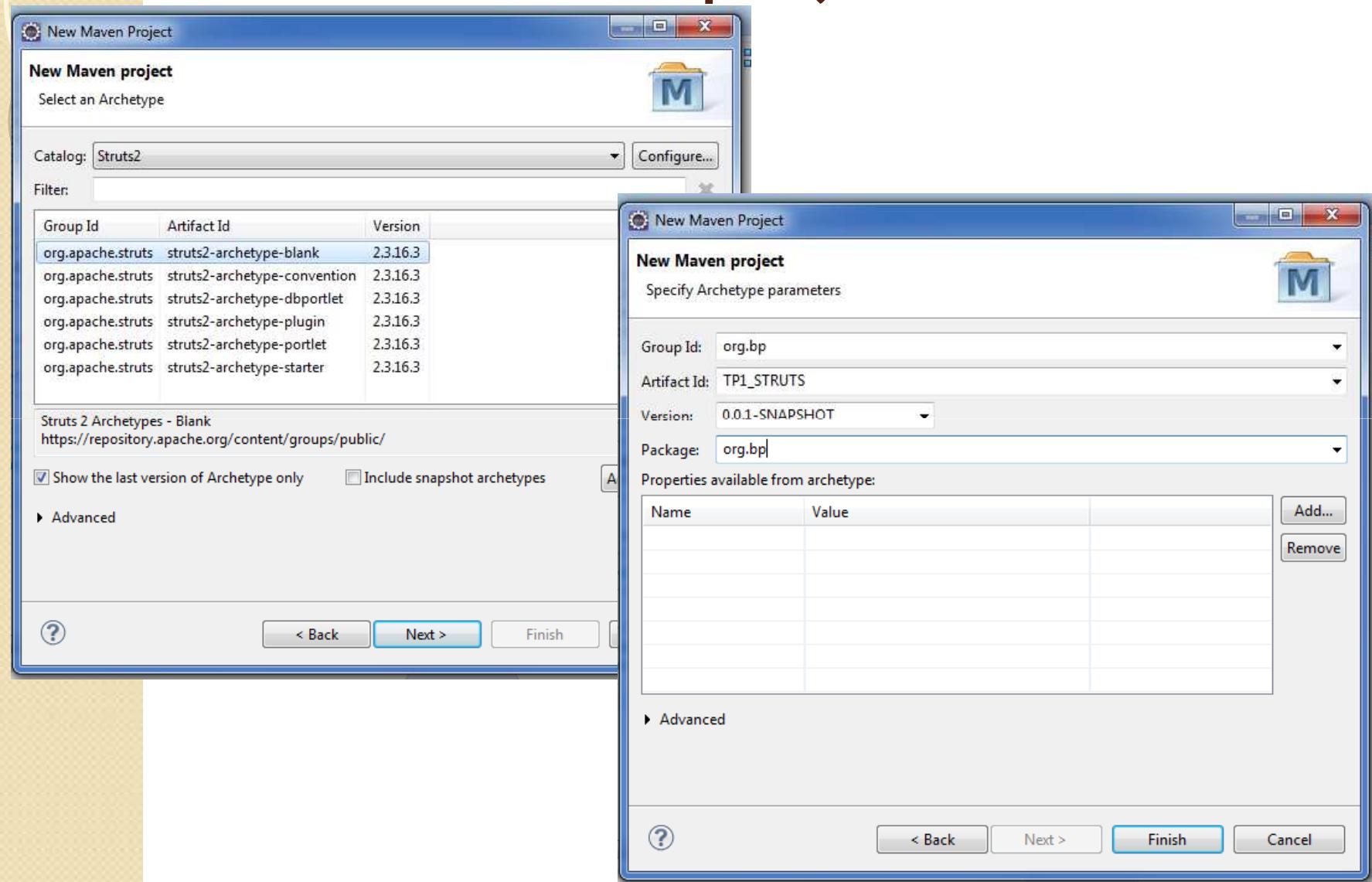


Création d'un projet Maven Struts

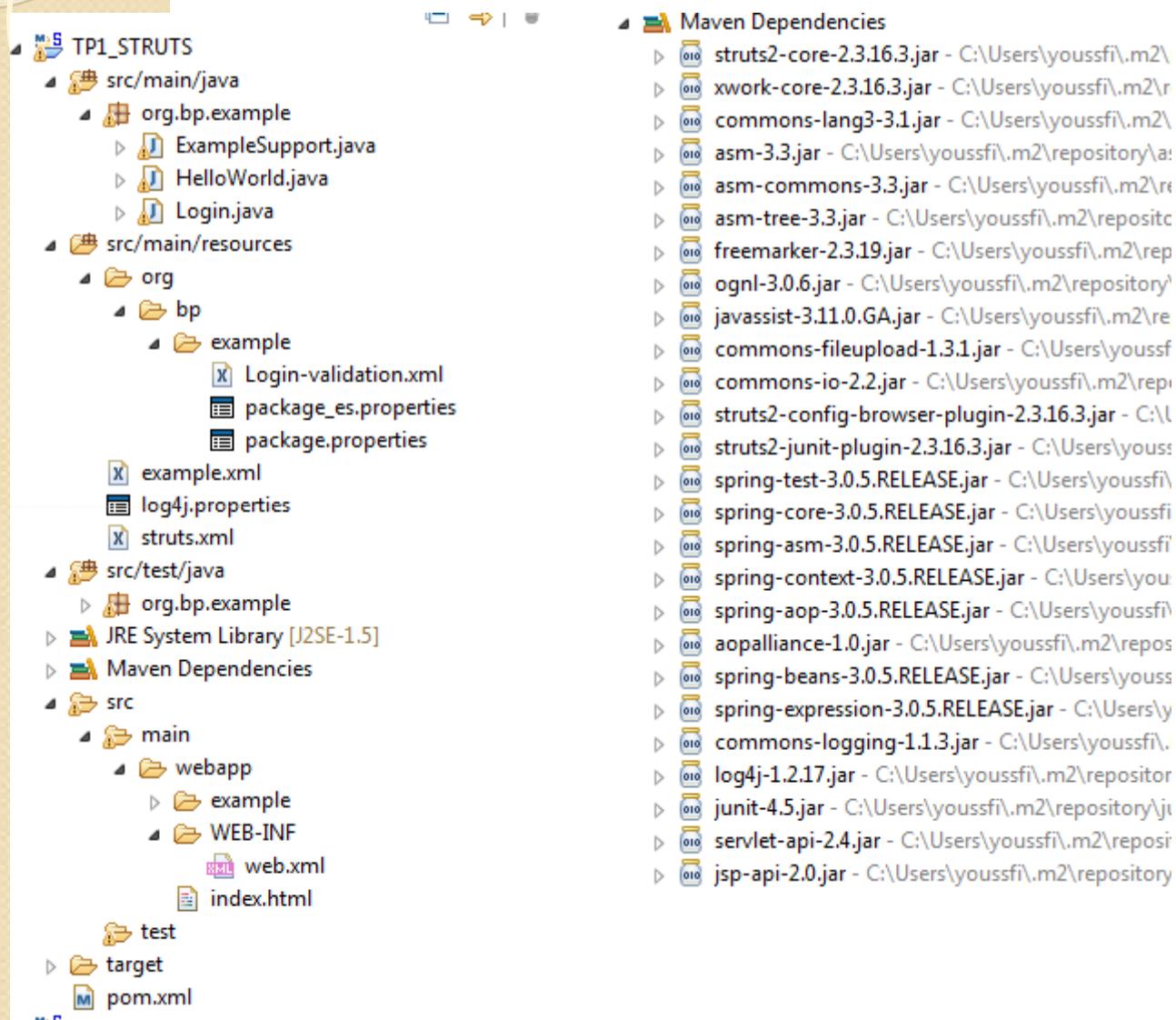


med@youssfi.net

Création d'un projet Maven Struts



Structure du projet : Projet Spring Maven Web



Pom.xml

```
<modelVersion>4.0.0</modelVersion>
<groupId>org.bp</groupId>
<artifactId>TP1_STRUTS</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>TP1_STRUTS</name>
<properties>
    <struts2.version>2.3.16.3</struts2.version>
    <project.build.sourceEncoding>UTF-8
    </project.build.sourceEncoding>
</properties>
```

Struts Maven dependencies

```
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-core</artifactId>
    <version>${struts2.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-config-browser-plugin</artifactId>
    <version>${struts2.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-junit-plugin</artifactId>
    <version>${struts2.version}</version>
    <scope>test</scope>
</dependency>
```

Logging and JUnit Maven dependencies

```
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.1.3</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.5</version>
    <scope>test</scope>
</dependency>
```

Servlet, JSP Maven dependencies

```
<dependency>
<groupId>javax.servlet</groupId>
<artifactId> servlet-api</artifactId>
<version>2.4</version>
<scope>provided</scope>

</dependency>
<dependency>
<groupId>javax.servlet</groupId>
<artifactId> jsp-api</artifactId>
<version>2.0</version>
<scope>provided</scope>
</dependency>
```



web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="struts_blank" 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">
    <display-name>Struts Blank</display-name>

    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
        </filter-class>
    </filter>

    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>*</url-pattern>
    </filter-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

struts.xml : Pour la configuration du contrôleur

```
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<package name="default" namespace="/" extends="struts-default">
    <default-action-ref name="index" />
    <action name="index">
        <result type="redirectAction">
            <param name="actionName">Saisie</param>
            <param name="namespace">/actions</param>
        </result>
    </action>
</package>
<package name="actions" namespace="/actions" extends="struts-default">
    <action name="Saisie">
        <result>/Vues/Calcul.jsp</result>
    </action>
    <action name="calcul" class="org.bp.web.CalculAction">
        <result name="success">/Vues/Calcul.jsp</result>
        <result name="error">/Vues/Erreur.jsp</result>
    </action>
</package>
</struts>
```

struts.xml : Pour la configuration du contrôleur

- Ce fichier XML indique au contrôleur que :
 - L'action par défaut est index. <http://Machine:port/nomProjet/index>
 - Pour cette action une redirection sera effectuée vers l'action Saisie du namespace actions : <http://Machine:port/nomProjet/actions/Saisie>
 - Pour cette dernière requête, la vue : /Vues/Calcul.jsp est affichée
 - Pour une requête dont l'url est de type :
 - <http://Machine:port/nomProjet/actions/calcul>
 - Le contrôleur FilterDispatcher va instancier la classe d'action : web.CalculAction, charge les données de la requête dans cette instance, ensuite fait appel à sa méthode execute().
 - Si la méthode execute() retourne « success », un forward sera effectué vers la page jsp Calcul.jsp du dossier Vues.
 - Si la méthode execute() retourne « error », un forward sera effectué vers la page jsp Erreur.jsp du dossier Vues.
 -

Le modèle Action : CalculAction.java

```
package org.bp.web;
import com.opensymphony.xwork2.ActionSupport;
public class CalculAction extends ActionSupport {
    private double v1;private double v2;
    private double resultat;
    @Override
    public String execute() throws Exception {
        if(v2==0){
            return "error";
        }
        else{
            resultat=v1/v2;
            return "success";
        }
    }
    // Getters et Setters
}
```

La vue : Calcul.jsp

```
<%@taglib prefix="s" uri="/struts-tags" %>
<html>
<body>
<s:form action="calcul">
    <s:div>
        <s:textfield label="V1" name="v1"></s:textfield>
        <s:textfield label="V2" name="v2"></s:textfield>
    </s:div>
    <s:div>
        <s:submit value="calcul"></s:submit>
    </s:div>
</s:form>
<s:div>
    Résultat=<s:property value="resultat"/>
</s:div>
</body>
</html>
```

V1: 67.0

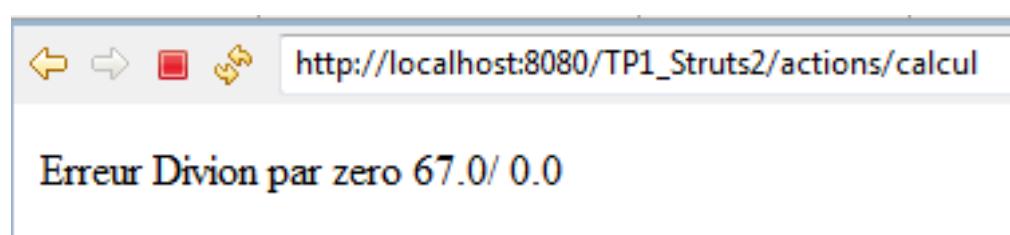
V2: 6.0

calcul

Résultat=11.16666666666666

La vue : Erreur.jsp

```
<%@taglib prefix="s" uri="/struts-tags" %>
<html>
<body>
    Erreur Division par zero <s:text
        name="v1"></s:text>/
    <s:text name="v2"></s:text>
</body>
</html>
```



Version Annotations

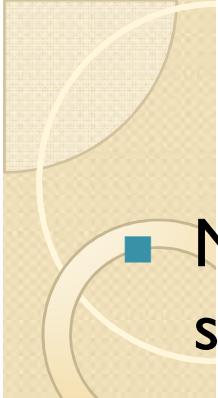
- Avec Struts2, il est plus simple de créer une application web en utilisant les annotations.
- L'utilisation des annotations permet:
 - De minimiser les configurations XML
 - Regrouper les traitements de plusieurs actions dans une même classe d'action.
 - Faciliter la validation des formulaires
 -

Dépendance Maven

```
<dependency>
<groupId>org.apache.struts</groupId>
<artifactId>struts2-convention-plugin</artifactId>
<version>${struts2.version}</version>
</dependency>
```

CalculAction.java : Version Annotations

```
package org.bp.web;
import org.apache.struts2.convention.annotation.*;
import com.opensymphony.xwork2.ActionSupport;
public class CalculAction extends ActionSupport {
    private double v1;private double v2; private double resultat;
    @Action(value="/index",
        results{@Result(name="success",location="/views/Calcul.jsp")})
    public String index(){
        return "success";
    }
    @Action(value="/calcul", results={
        @Result(name="success",location="/views/Calcul.jsp"),
        @Result(name="error",location="/views/Erreur.jsp")})
    public String calcul() throws Exception {
        if(v2==0){ return "error"; } else {
            resultat=v1/v2; return "success"; }
    }
    // Getters et Setters
}
```



struts.xml version Annotations:

- Même Contenu pour toutes les applications, quelques soit la taille de l'application

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>
    <constant name="struts.convention.action.packages" value="org.bp.web" />
    <constant name="struts.convention.action.suffix" value="Action" />
</struts>
```

- Dans ce fichier, on déclare:
 - Les action se trouvent dans le package org.bp.web
 - Les noms des classes d'actions se terminent par le mot Action



Application

- On souhaite créer une application qui permet de gérer des abonnements.
- Il existe deux type d'abonnement GSM et INTERNET
- Un abonnement est défini par son identifiant, sa date, son solde sont état (actif ou non)
- Un abonnement GSM est un abonnement qui possède en plus le nombre de points fidelio.
- Un abonnement internet est un abonnement qui possède en plus le débit.
- L'application doit permettre de :
 - Saisir et ajouter un abonnement
 - Consulter les abonnement actif ou non
 - Consulter les abonnements entre deux dates
 - Editer et modifier un abonnement
 - Supprimer un abonnement.
 - Consommer un montant d'un abonnement

Abonnements

localhost:8080/AbonnementWEB/index

ID:

Solde:

Date:

Actif:

Type Ab:

fidelio:

ID	Date	Solde	Type	Fidelio	debit		
2	08/05/14 00:00:00.000	200.0	AbonnementGSM	300	0	Supp	Edit
3	08/05/14 00:00:00.000	11111.0	AbonnementInternet	0	4	Supp	Edit
7	05/06/14 00:00:00.000	980.0	AbonnementInternet	0	8	Supp	Edit
31	05/07/14 00:00:00.000	300.0	AbonnementGSM	400	0	Supp	Edit
32	05/08/14 00:00:00.000	12345.0	AbonnementGSM	300	0	Supp	Edit
47	06/05/14 00:00:00.000	400.0	AbonnementGSM	100	0	Supp	Edit

Architecture

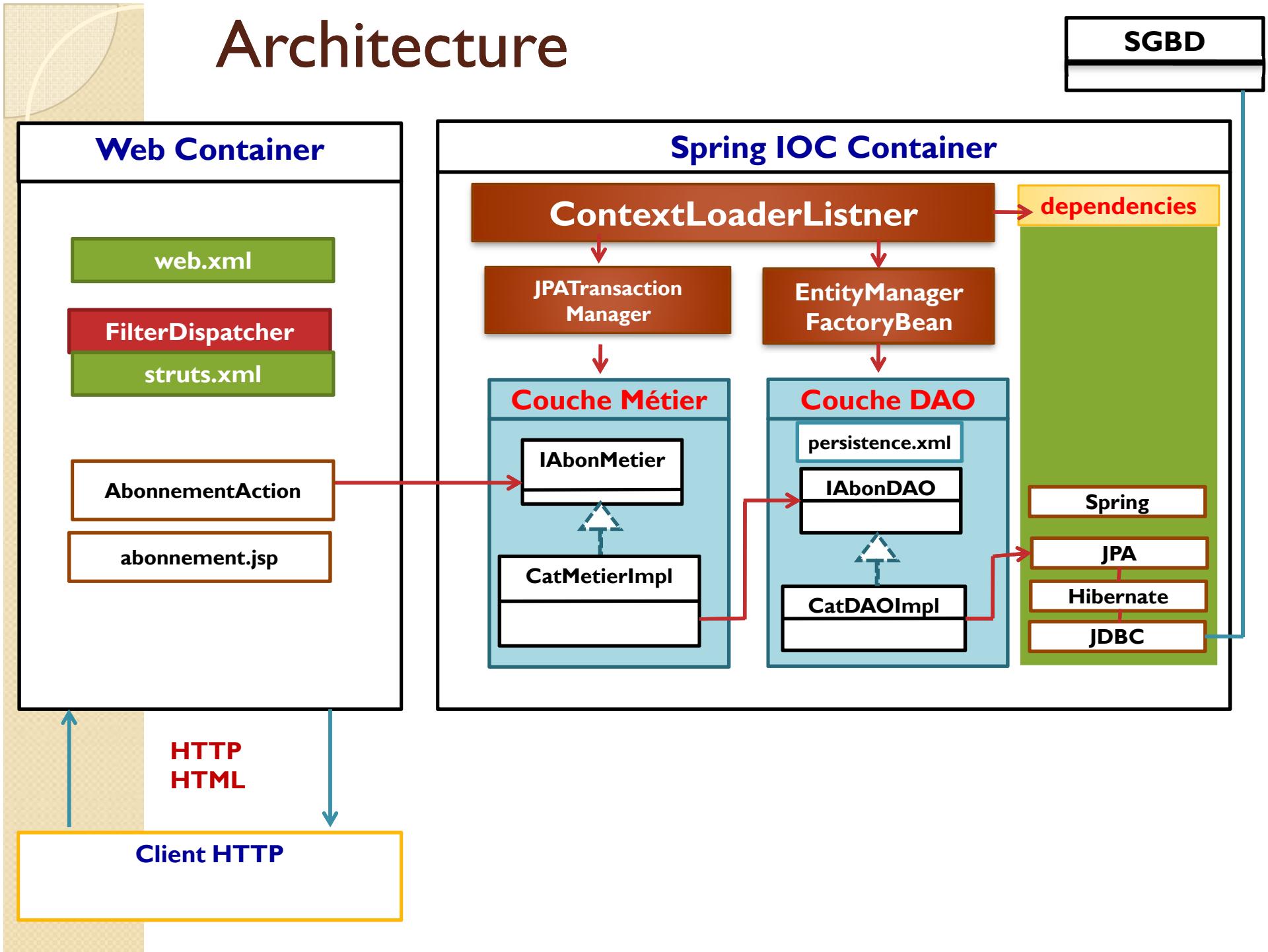
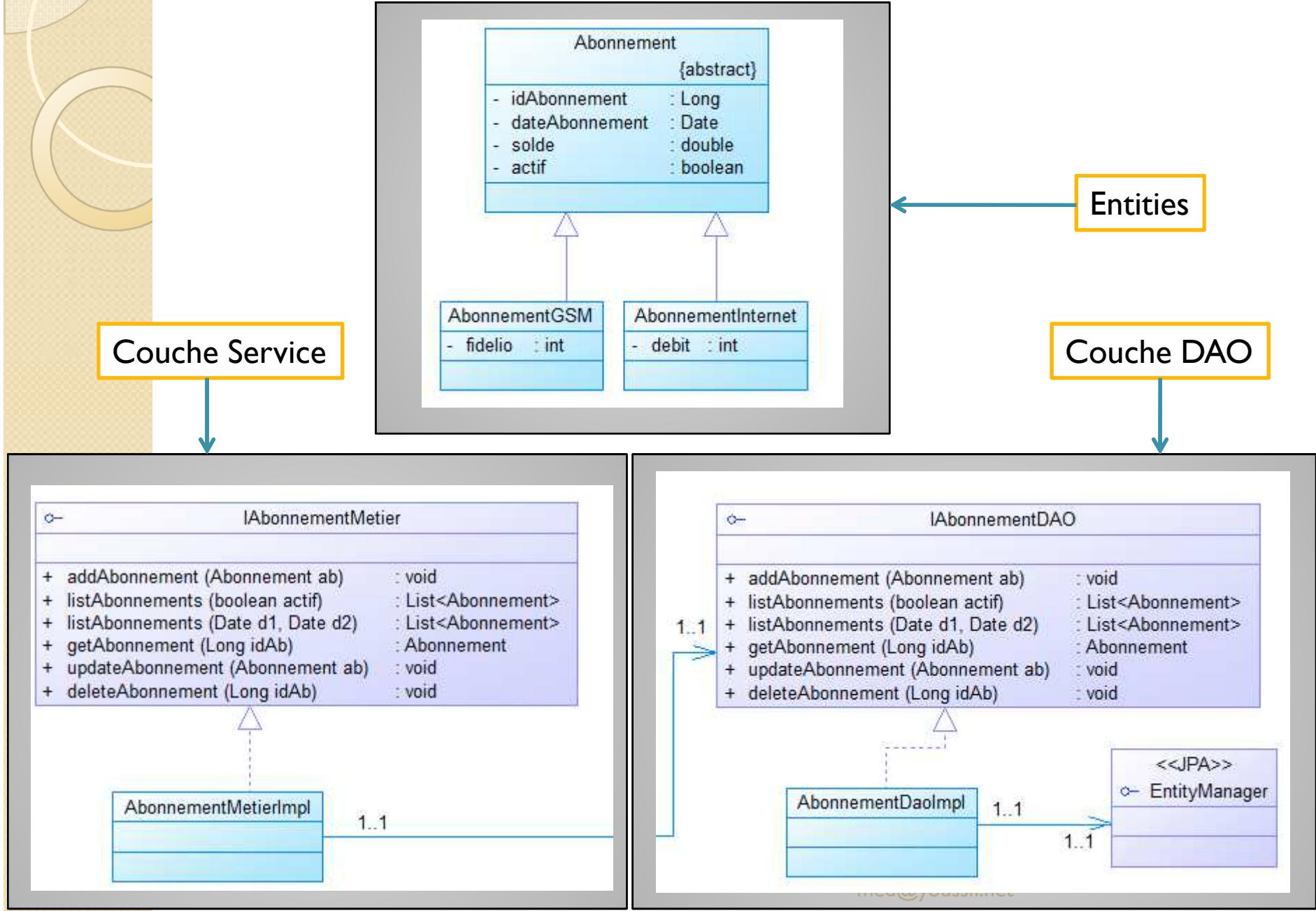
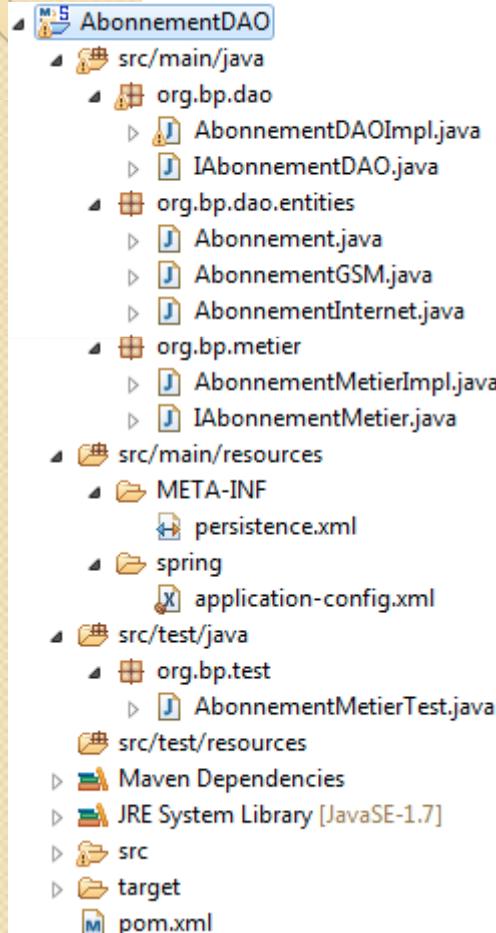


Diagramme de classes



Modules DAO et Métier

Structure du projet



Dépendances

Maven Dependencies

- spring-core-3.2.3.RELEASE.jar - C:\Users\you
- commons-logging-1.1.1.jar - C:\Users\youssf
- spring-context-3.2.3.RELEASE.jar - C:\Users\y
- spring-aop-3.2.3.RELEASE.jar - C:\Users\you
- spring-expression-3.2.3.RELEASE.jar - C:\User
- spring-beans-3.2.3.RELEASE.jar - C:\Users\yo
- spring-tx-3.2.3.RELEASE.jar - C:\Users\youssf
- aopalliance-1.0.jar - C:\Users\youssf\.m2\re
- spring-orm-3.2.3.RELEASE.jar - C:\Users\you
- spring-jdbc-3.2.3.RELEASE.jar - C:\Users\you
- spring-webmvc-3.2.3.RELEASE.jar - C:\Users\y
- spring-web-3.2.3.RELEASE.jar - C:\Users\you
- hibernate-entitymanager-4.2.1.Final.jar - C:\U
- jboss-logging-3.1.0.GA.jar - C:\Users\youssf\
- hibernate-core-4.2.1.Final.jar - C:\Users\you
- antlr-2.7.7.jar - C:\Users\youssf\.m2\repo3\
- dom4j-1.6.1.jar - C:\Users\youssf\.m2\repo3\
- jboss-transaction-api_1.1_spec-1.0.1.Final.ja
- hibernate-jpa-2.0-api-1.0.1.Final.jar - C:\Us
- javassist-3.15.0-GA.jar - C:\Users\youssf\.m2\
- hibernate-commons-annotations-4.0.1.Final.j
- mysql-connector-java-5.1.6.jar - C:\Users\yo
- junit-4.11.jar - C:\Users\youssf\.m2\repo3\ju
- hamcrest-core-1.3.jar - C:\Users\youssf\.m2\

Propriétés du projet

```
<modelVersion>4.0.0</modelVersion>
<groupId>org.bp</groupId>
<artifactId>AbonnementDAO</artifactId>
<version>0.0.1-SNAPSHOT</version>
<properties>
    <!-- Generic properties -->
    <java.version>1.7</java.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <!-- Spring -->
    <spring-framework.version>3.2.3.RELEASE</spring-framework.version>
    <!-- Hibernate / JPA -->
    <hibernate.version>4.2.1.Final</hibernate.version>
    <!-- Logging -->
    <logback.version>1.0.13</logback.version>
    <slf4j.version>1.7.5</slf4j.version>
    <!-- Test -->
    <junit.version>4.11</junit.version>
</properties>
```

Maven dependencies

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring-framework.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring-framework.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>${spring-framework.version}</version>
</dependency>
```

Maven dependencies

```
<!-- Spring and Transactions -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${spring-framework.version}</version>
</dependency>
<!-- Spring ORM -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring-framework.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring-framework.version}</version>
</dependency>
```

Maven dependencies

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>${hibernate.version}</version>
</dependency>
<!-- MYSQL -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.6</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>${junit.version}</version>
  <scope>test</scope>
</dependency>
```



Compiler plugin

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>2.5.1</version>
    <configuration>
        <source>1.7</source>
        <target>1.7</target>
        <compilerArgument>-Xlint:all</compilerArgument>
        <showWarnings>true</showWarnings>
        <showDeprecation>true</showDeprecation>
    </configuration>
</plugin>
```



COUCHE DAO

med@youssf.net

Entities :Abonnement

```
package org.bp.dao.entities;
import java.util.Date; import javax.persistence.*;
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TYPE_AB",length=4)
public abstract class Abonnement {
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
    private Long idAbonnement; private double solde;
    private Date dateAbonnement; private boolean actif;
public Abonnement(double solde, Date dateAbonnement, boolean actif) {
    this.solde = solde; this.dateAbonnement = dateAbonnement; this.actif = actif;
}
public Abonnement() {
}
// Getters et Setters
}
```

Entities :AbonnementGSM

```
package org.bp.dao.entities;
import java.util.Date;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
@Entity
@DiscriminatorValue(value="GSM")
public class AbonnementGSM extends Abonnement {
    private int fidelio;

    public AbonnementGSM(double solde, Date dateAbonnement, boolean actif,
    int fidelio) {
        super(solde, dateAbonnement, actif);
        this.fidelio = fidelio;
    }
    public AbonnementGSM() {
        super();
    }
    // Getters et Setters
}
```

Entities :AbonnementInternet

```
package org.bp.dao.entities;
import java.util.Date;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
@Entity
@DiscriminatorValue("INT")
public class AbonnementInternet extends Abonnement {
    private int debit;

    public AbonnementInternet(double solde, Date dateAbonnement, boolean actif,
    int debit) {
        super(solde, dateAbonnement, actif);
        this.debit = debit;
    }
    public AbonnementInternet() {
    }
    // Getters et Setters
}
```

Interface DAO : IAbonnementDAO

```
package org.bp.dao;  
import java.util.Date;  
import java.util.List;  
  
import org.bp.dao.entities.Abonnement;  
public interface IAbonnementDAO {  
    public void addAbonnement(Abonnement ab);  
    public List<Abonnement> listAbonnements(boolean actif);  
    public List<Abonnement> listAbonnements(Date d1, Date d2);  
    public Abonnement getAbonnement(Long idAb);  
    public void deleteAbonnement(Long idAb);  
    public void updateAbonnement(Abonnement ab);  
    public void consommer(Long idAb, double mt);  
}
```



Implémentation JPA: AbonnementDAOImpl

```
package org.bp.dao;  
import java.util.Date;import java.util.List;  
import javax.persistence.*;  
import org.bp.dao.entities.Abonnement;  
public class AbonnementDAOImpl implements IAbonnementDAO {  
    @PersistenceContext  
    private EntityManager em;  
    @Override  
    public void addAbonnement(Abonnement ab) {  
        em.persist(ab);  
    }  
    @Override  
    public List<Abonnement> listAbonnements(boolean actif) {  
        Query req=em.createQuery("select ab from Abonnement ab where ab.actif=:x");  
        req.setParameter("x", actif);  
        return req.getResultList();  
    }  
}
```



Implémentation JPA: AbonnementDaoImpl

```
@Override  
public List<Abonnement> listAbonnements(Date d1, Date d2) {  
    Query req=em.createQuery("select ab from Abonnement ab where  
        ab.dateAbonnement beetwen :x and :x");  
    req.setParameter("x", d1);  
    req.setParameter("y",d2);  
    return req.getResultList();  
}  
  
@Override  
public Abonnement getAbonnement(Long idAb) {  
    return em.find(Abonnement.class, idAb);  
}  
  
@Override  
public void deleteAbonnement(Long idAb) {  
    Abonnement ab=getAbonnement(idAb);  
    em.remove(ab);  
}
```



Implémentation JPA: AbonnementDaoImpl

```
@Override  
public void updateAbonnement(Abonnement ab) {  
    em.merge(ab);  
}  
  
@Override  
public void consommer(Long idAb, double mt) {  
    Abonnement ab=getAbonnement(idAb);  
    ab.setSolde(ab.getSolde()-mt);  
}  
}
```



COUCHE SERVICE

med@youssf.net

Interface Service : IAbonnementDAO

```
package org.bp.metier;

import java.util.Date;
import java.util.List;
import org.bp.dao.entities.Abonnement;
public interface IAbonnementMetier {
    public void addAbonnement(Abonnement ab);
    public List<Abonnement> listAbonnements(boolean actif);
    public List<Abonnement> listAbonnements(Date d1, Date d2);
    public Abonnement getAbonnement(Long idAb);
    public void deleteAbonnement(Long idAb);
    public void updateAbonnement(Abonnement ab);
    public void consommer(Long idAb, double mt);
}
```

Implémentation JPA: AbonnementDaoImpl

```
package org.bp.metier;
import java.util.*;import org.bp.dao.IAbonnementDAO;
import org.bp.dao.entities.Abonnement;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
@Service
@Transactional
public class AbonnementMetierImpl implements IAbonnementMetier {
    @Autowired
    private IAbonnementDAO dao;
    public void setDao(IAbonnementDAO dao) {
        this.dao = dao;
    }
}
```

Implémentation Service: AbonnementMetierimpl

```
@Override  
public void addAbonnement(Abonnement ab) { dao.addAbonnement(ab); }  
  
@Override  
public List<Abonnement> listAbonnements(boolean actif) {  
    return dao.listAbonnements(actif);  
}  
  
@Override  
public List<Abonnement> listAbonnements(Date d1, Date d2) {  
    return dao.listAbonnements(d1, d2);  
}  
  
@Override  
public Abonnement getAbonnement(Long idAb) { return dao.getAbonnement(idAb); }  
  
@Override  
public void deleteAbonnement(Long idAb) { dao.deleteAbonnement(idAb); }  
  
@Override  
public void updateAbonnement(Abonnement ab) { dao.updateAbonnement(ab); }  
  
@Override  
public void consommer(Long idAb, double mt) { dao.consommer(idAb, mt); }  
}
```

Unité de persistance JPA :

/AbonnementDAO/src/main/resources/META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" 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_2_0.xsd ">
  <persistence-unit name="UP_AB" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibetnate.show_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

Unité de persistance JPA :

/AbonnementDAO/src/main/resources/META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>

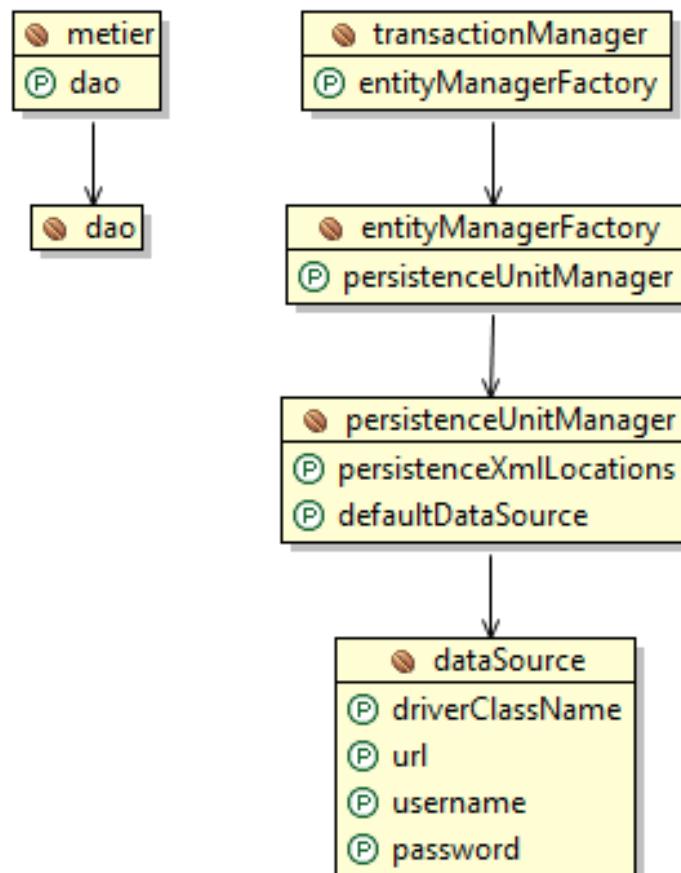
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd ">

  <persistence-unit name="UP_AB" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibetnate.show_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

Injection des dépendances Spring IOC :

/AbonnementDAO/src/main/resources/spring/**application-config.xml**

- Graphe des dépendances :



Injection des dépendance Spring IOC :

/AbonnementDAO/src/main/resources/spring/**application-config.xml**

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
                           http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="dao" class="org.bp.dao.AbonnementDAOImpl"></bean>
    <bean id="metier" class="org.bp.metier.AbonnementMetierImpl">
        <property name="dao" ref="dao"></property>
    </bean>
```

Injection des dépendance Spring IOC :

/AbonnementDAO/src/main/resources/spring/**application-config.xml**

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
    <property name="url" value="jdbc:mysql://localhost:3306/DB_AB"></property>
    <property name="username" value="root"></property>
    <property name="password" value=""></property>
</bean>
<bean id="persistenceUnitManager"
      class="org.springframework.orm.jpa.persistenceunit.DefaultPersistenceUnitManager">
    <property name="persistenceXmlLocations">
        <list>
            <value>classpath*:META-INF/persistence.xml</value>
        </list>
    </property>
    <property name="defaultDataSource" ref="dataSource"></property>
</bean>
```



Injection des dépendances Spring IOC :

/AbonnementDAO/src/main/resources/spring/**application-config.xml**

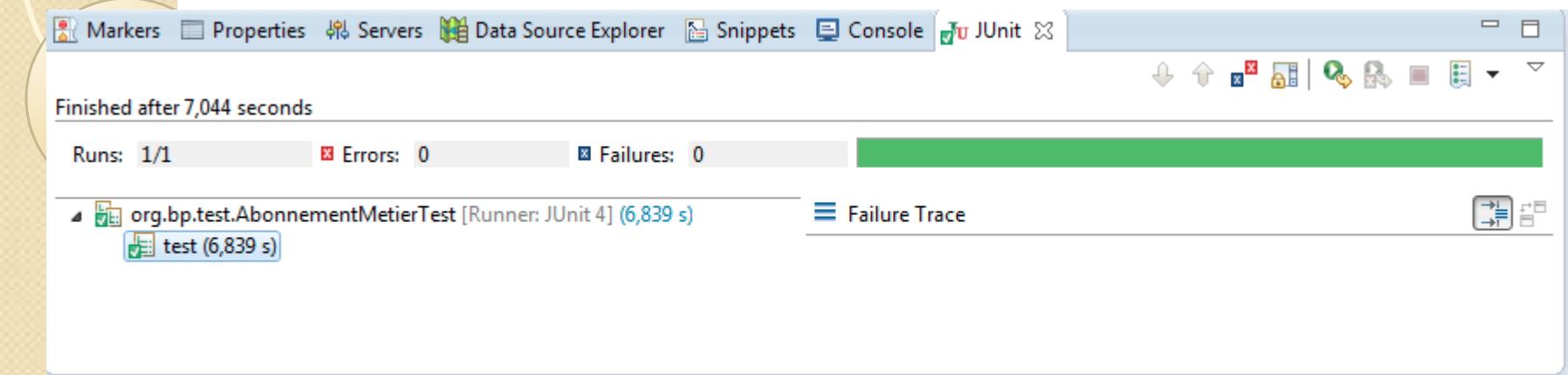
```
<bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="persistenceUnitManager" ref="persistenceUnitManager"></property>
</bean>
<bean id="transactionManager"  class="org.springframework.orm.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"></property>
</bean>
<tx:annotation-driven transaction-manager="transactionManager"/>
<context:annotation-config></context:annotation-config>
</beans>
```

JUnit Test de des couches service et dao

```
package org.bp.test;

import static org.junit.Assert.*;
import java.util.*; import org.bp.dao.entities.*;
import org.bp.metier.*; import org.junit.Before;
import org.junit.Test;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class AbonnementMetierTest {
    private ClassPathXmlApplicationContext context;
    @Before
    public void setUp() throws Exception {
        context=new ClassPathXmlApplicationContext(new String[]{"spring/application-
        config.xml"});
    }
    @Test
    public void test() {
        IAbonnementMetier metier=(IAbonnementMetier) context.getBean("metier");
        List<Abonnement> abs1=metier.listAbonnements(true);
        metier.addAbonnement(new AbonnementGSM(5000,new Date(),true,0));
        metier.addAbonnement(new AbonnementInternet(9000,new Date(),true,4));
        List<Abonnement> abs2=metier.listAbonnements(true);
        assertTrue(abs2.size()==abs1.size()+2);
    }
}
```

JUnit Test

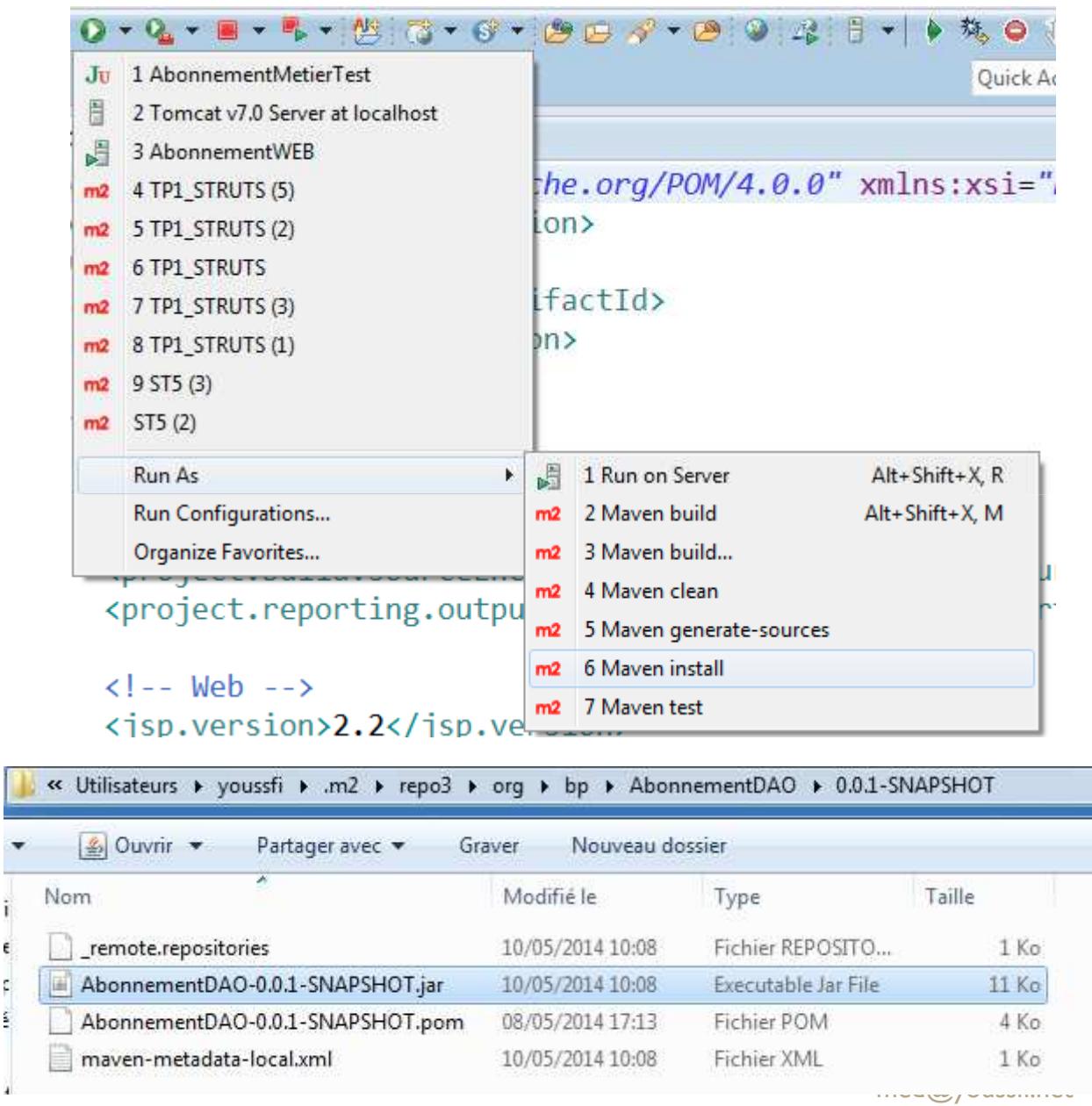


Base de données générée : La table abonnements

Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra
TYPE_AB	varchar(4)	latin1_swedish_ci		Non	Aucun	
<u>idAbonnement</u>	bigint(20)			Non	Aucun	AUTO_INCREMENT
actif	tinyint(1)			Non	Aucun	
dateAbonnement	datetime			Oui	NULL	
solde	double			Non	Aucun	
fidelio	int(11)			Oui	NULL	
debit	int(11)			Oui	NULL	

TYPE_AB	idAbonnement	actif	dateAbonnement	solde	fidelio	debit
GSM		1	1 2014-05-10 10:04:33	5000	0	NULL
INT		2	1 2014-05-10 10:04:33	9000	NULL	4

Installation du package dans le repository



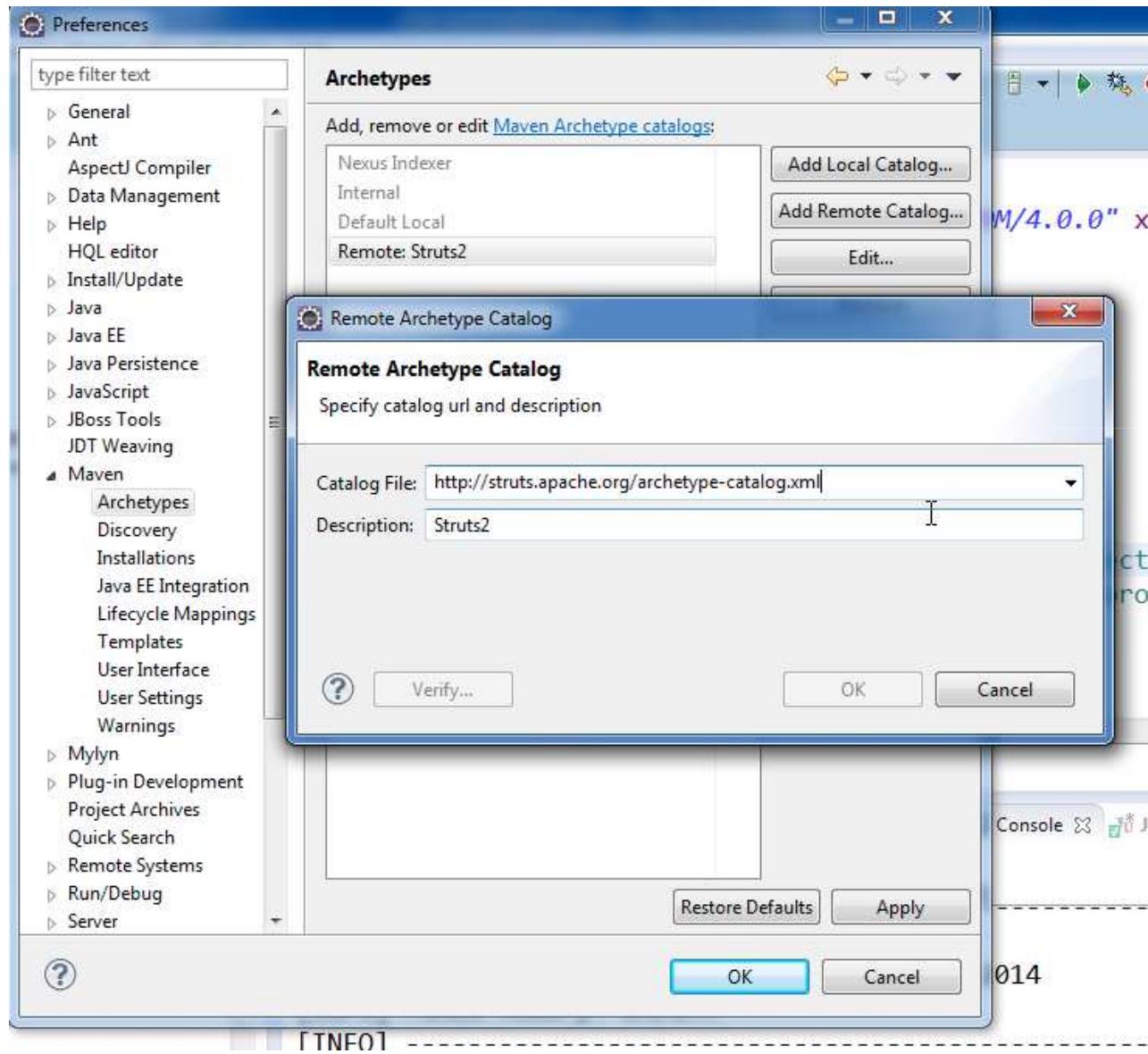


COUCHE WEB AVEC STRUTS

med@youssf.net

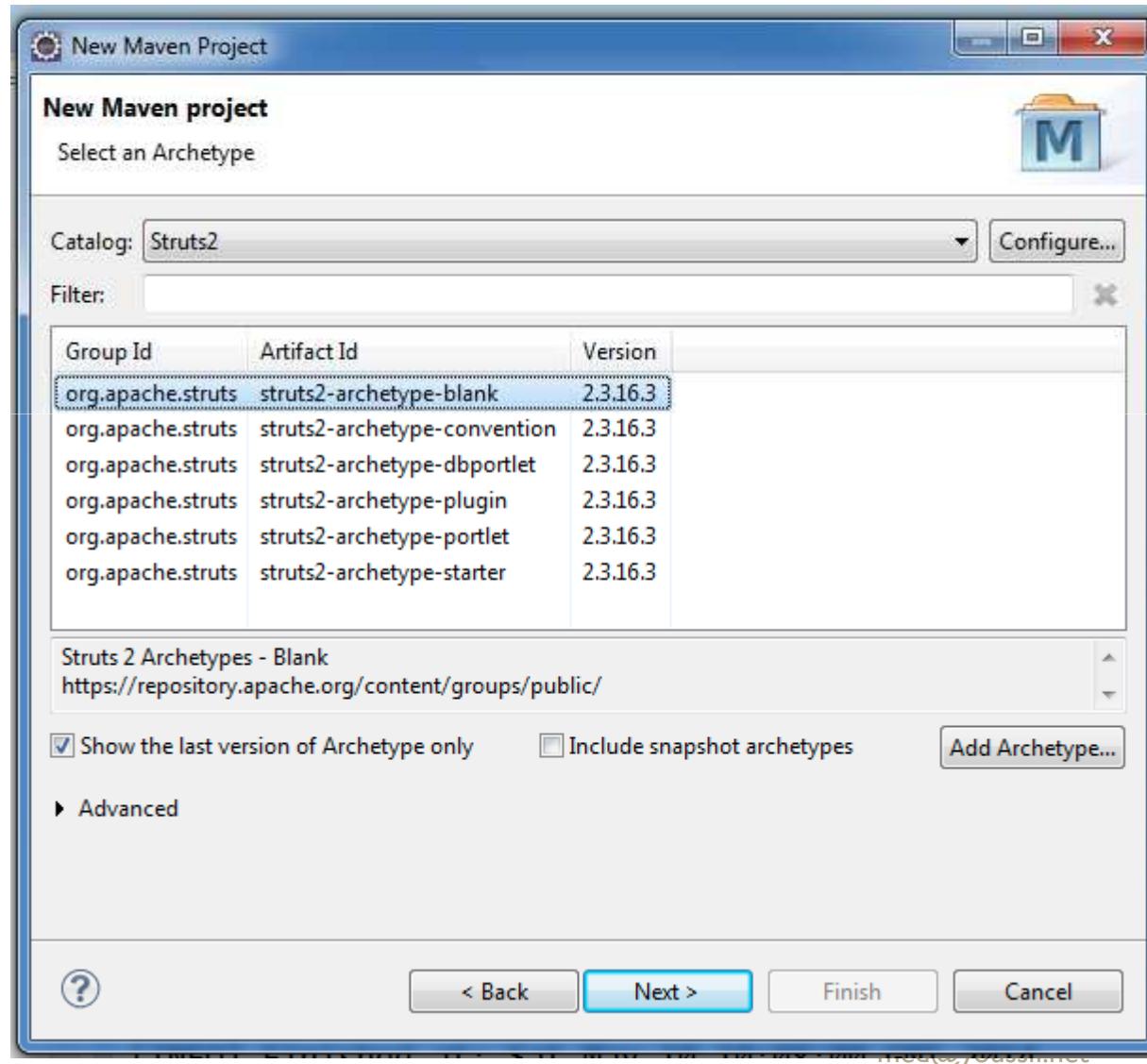
Ajouter un catalogue maven archetype pour struts

- <http://struts.apache.org/archetype-catalog.xml>

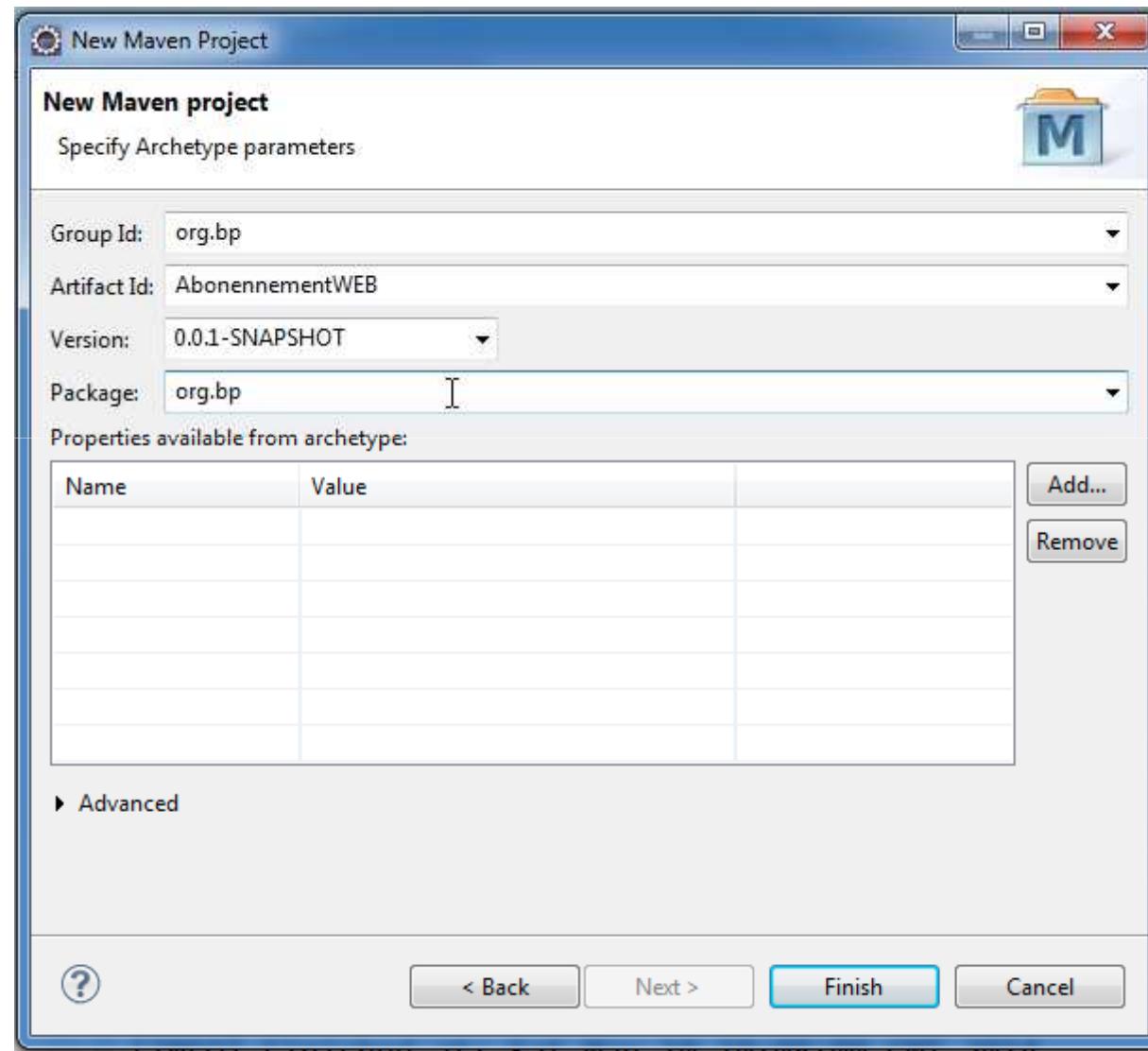


Créer un projet maven struts

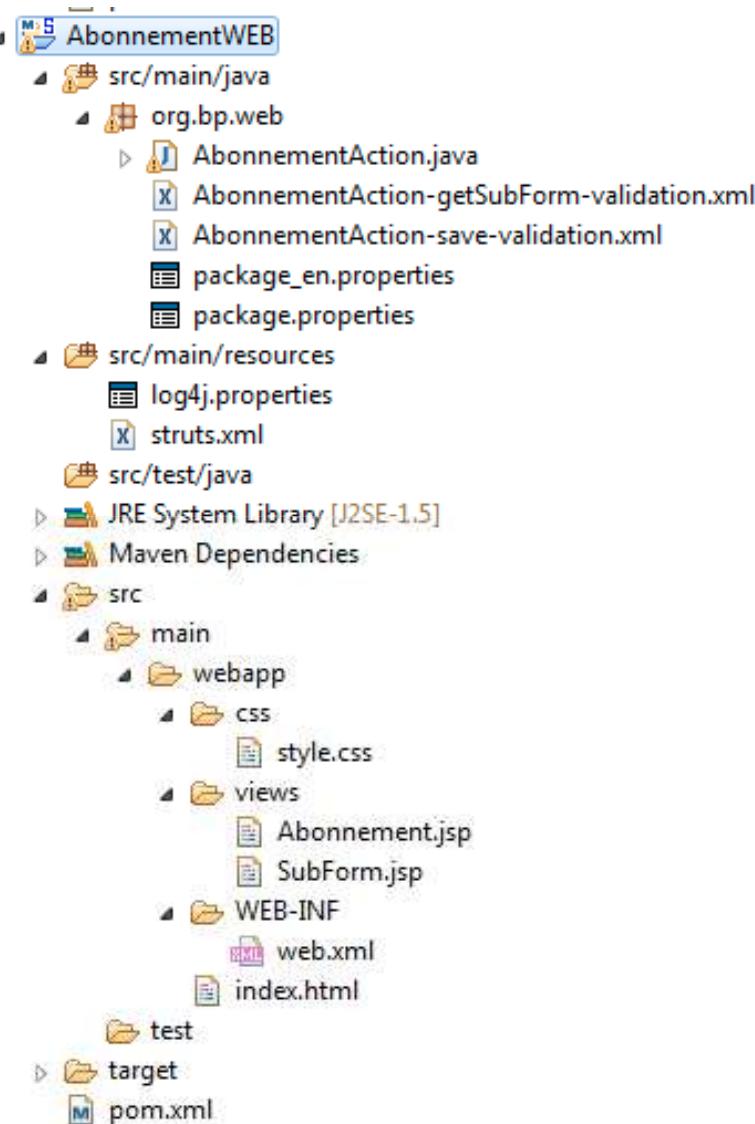
- File >New > Maven Project



Créer un projet maven struts



Structure du projet à créer



Maven project properties

```
<modelVersion>4.0.0</modelVersion>
<groupId>org.bp</groupId>
<artifactId>AbonnementWEB</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>AbonnementWEB</name>
<properties>
    <struts2.version>2.3.16.3</struts2.version>
        <project.build.sourceEncoding>UTF-
        8</project.build.sourceEncoding>
</properties>
```

Maven dependencies (Struts, Servlet, JSP)

```
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-core</artifactId>
    <version>${struts2.version}</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.4</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.0</version>
    <scope>provided</scope>
</dependency>
```

Maven dependencies (Struts2-Spring-plugin)

```
<dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-spring-plugin</artifactId>
    <version>${struts2.version}</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```



Maven dependencies : Struts2-Jquery-Plugin et AbonnementDAO

```
<dependency>
    <groupId>com.jgeppert.struts2.jquery</groupId>
    <artifactId>struts2-jquery-plugin</artifactId>
    <version>3.7.0</version>
</dependency>

<dependency>
    <groupId>org.bp</groupId>
    <artifactId>AbonnementDAO</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```



Maven dependencies : Logging et JUnit

```
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.1.3</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.5</version>
    <scope>test</scope>
</dependency>
```

Maven plugins : compiler plugin

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>2.0.2</version>
  <configuration>
    <source>1.7</source>
    <target>1.7</target>
  </configuration>
</plugin>
```

Maven plugins : jetty plugin

```
<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>jetty-maven-plugin</artifactId>
  <version>8.1.7.v20120910</version>
  <configuration>
    <stopKey>CTRL+C</stopKey>
    <stopPort>8999</stopPort>
    <systemProperties>
      <systemProperty>
        <name>log4j.configuration</name>
        <value>file:${basedir}/src/main/resources/log4j.properties</value>
      </systemProperty>
      <systemProperty>
        <name>slf4j</name>
        <value>false</value>
      </systemProperty>
    </systemProperties>
```

Maven plugins : jetty plugin

```
<scanIntervalSeconds>10</scanIntervalSeconds>
<webAppSourceDirectory>${basedir}/src/main/webapp/</webAppSourceDirectory>
<webAppConfig>
    <contextPath>/AbonnementWEB</contextPath>
    <descriptor>${basedir}/src/main/webapp/WEB-INF/web.xml</descriptor>
</webAppConfig>
</configuration>
<dependencies>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
</dependencies>
</plugin>
```

Déploiement du contrôleur Struts : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="struts_blank" 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">
    <display-name>Struts Blank</display-name>
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

Déploiement de Spring IOC: web.xml

```
<!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring/application-config.xml</param-value>
</context-param>
<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

Configuration de Struts : struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
        "http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

    <package name="default" namespace="/" extends="struts-default">

        <action name="index" class="org.bp.web.AbonnementAction" method="index">
            <result name="success">/views/Abonnement.jsp</result>
        </action>

        <action name="save" class="org.bp.web.AbonnementAction" method="save">
            <result type="redirectAction">
                <param name="actionName">index</param>
                <param name="namespace">/</param>
            </result>
            <result name="input">/views/Abonnement.jsp </result>
        </action>
```

Configuration de Struts : struts.xml

```
<action name="delete" class="org.bp.web.AbonnementAction" method="delete">
    <result type="redirectAction">
        <param name="actionName">index</param>
        <param name="namespace">/</param>
    </result>
</action>
<action name="edit" class="org.bp.web.AbonnementAction" method="edit">
    <result name="success">/views/Abonnement.jsp</result>
</action>
<action name="getSubForm" class="org.bp.web.AbonnementAction" method="getSubForm">
    <result name="success">/views/SubForm.jsp</result>
    <result name="input">/views/SubForm.jsp</result>
</action>
</package>
</struts>
```

ModèleAction :AbonnementAction

```
package org.bp.web;

import java.util.*; import org.bp.dao.entities.*;
import org.bp.metier.IAbonnementMetier;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import com.opensymphony.xwork2.ActionSupport;
@Component
public class AbonnementAction extends ActionSupport {
@Autowired
private IAbonnementMetier metier;
private Date dateAbonnement; private double solde;
private boolean actif; private String type;
private int fidelio; private int debit;
private Long idAb; private byte mode;
private String[] typesAb=new String[]{"","GSM","INTERNET"};
private List<Abonnement> listAbonnements;
```

ModèleAction :AbonnementAction

```
public String index(){  
    listAbonnements=metier.listAbonnements(true); mode=0;  
    return SUCCESS;  
}  
  
public String getSubForm(){ return SUCCESS; }  
  
public String save(){  
    Abonnement ab;  
    if(type.equals("GSM"))  
        ab=new AbonnementGSM(solde,dateAbonnement,actif,fidelio);  
    else  
        ab=new AbonnementInternet(solde,dateAbonnement,actif,debit);  
    if(mode==0)  
        metier.addAbonnement(ab);  
    else{  
        ab.setIdAbonnement(idAb);  
        metier.updateAbonnement(ab);  
    }  
    listAbonnements=metier.listAbonnements(true);  
    return SUCCESS;  
}
```

ModèleAction :AbonnementAction

```
public String delete(){
    metier.deleteAbonnement(idAb);
    listAbonnements=metier.listAbonnements(true);
    return SUCCESS;
}

public String edit(){
    Abonnement ab=metier.getAbonnement(idAb);
    String abClassName="Abonnement";
    String className=ab.getClass().getSimpleName();
    type=className.substring(abClassName.length(), className.length()).toUpperCase();
    idAb=ab.getIdAbonnement();
    dateAbonnement=ab.getDateAbonnement();
    solde=ab.getSolde();
    actif=ab.isActif();
    if(ab instanceof AbonnementGSM) fidelio=((AbonnementGSM)ab).getFidelio();
    if(ab instanceof AbonnementInternet) debit=((AbonnementInternet)ab).getDebit();
    listAbonnements=metier.listAbonnements(true);
    mode=1;
    return SUCCESS;
}
// Getters et Setters
}
```

Vue : abonnement.jsp

```
<%@taglib uri="/struts-tags" prefix="s" %>
<%@taglib uri="/struts-jquery-tags" prefix="j" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Abonnements</title>
<j:head/>
<link rel="stylesheet" type="text/css" href="css/style.css">
<script type="text/javascript">
$(function(){
    getSubForm($("#typesAb").val());
});
function getSubForm(type){
    $.get("getSubForm?fidelio=<s:property value='fidelio' />&debit=<s:property
value='debit' />&type="+type, function(rep){
        $("#divSubForm").html(rep);
    });
}
</script>
</head>
```

Vue : abonnement.jsp

```
<body>
  <div class="cadre">
    <s:form action="save" id="form1" method="get">
      <s:hidden name="idAb"></s:hidden>
      <s:textfield label="ID" name="idAb" disabled="true"></s:textfield>
      <s:textfield label="Solde" name="solde" Labelposition="Left"
key="solde"></s:textfield>
      <j:datepicker label="Date" name="dateAbonnement"
Labelposition="Left"></j:datepicker>
      <s:checkbox label="Actif" name="actif" Labelposition="Left"></s:checkbox>
      <s:select id="typesAb" list="typesAb" name="type" value="type" Label="Type Ab"
onchange="getSubForm(this.value)"></s:select>
      <tbody id="divSubForm"></tbody>
      <s:hidden name="mode"></s:hidden>
      <s:submit value="Save" method="save"></s:submit>
    </s:form>
  </div>
```

Vue : abonnement.jsp

```
<div class="cadre">
  <table class="table1">
    <tr>
      <th>ID</th><th>Date</th><th>Solde</th><th>Type</th><th>Fidelio</th><th>debit</th>
    </tr>
    <s:iterator value="listAbonnements">
      <tr>
        <td><s:property value="idAbonnement"/></td>
        <td><s:property value="dateAbonnement"/></td>
        <td><s:property value="solde"/></td>
        <td><s:property value="class.simpleName"/></td>
        <td><s:property value="fidelio"/></td> <td><s:property value="debit"/></td>
        <s:url action="delete" var="lien1">
          <s:param name="idAb">
            <s:property value="idAbonnement"/>
          </s:param>
        </s:url>
        <td><s:a href="#">Supp</s:a></td>
      </tr>
    </s:iterator>
  </table>
</div>
```

Vue : abonnement.jsp

```
<s:url action="edit" var="lien1">
    <s:param name="idAb">
        <s:property value="idAbonnement"/>
    </s:param>
</s:url>
<td><s:a href="#">Edit</s:a></td>
</tr>
</s:iterator>
</table>
</div>
</body>
</html>
```

Vue SubForm.jap

```
<%@taglib uri="/struts-tags" prefix="s" %>
<s:if test="%{type=='GSM'}">
    <s:textfield name="fidelio" label="fidelio"></s:textfield>
</s:if>
<s:elseif test="%{type=='INTERNET'}">
    <s:textfield name="debit" label="Débit"></s:textfield>
</s:elseif>
```

Style.css

```
div.cadre{  
    border: 1px dotted gray;  
    padding: 10px;  
    margin: 10px;  
}  
.table1 th{  
    border: 1px dotted gray;  
    padding: 10px;  
    background: pink;  
}  
.table1 td{  
    border: 1px dotted gray;  
    padding: 10px;  
    background: white;  
}
```

Validation des formulaires pour l'action save :

AbonnementAction-save-validation.xml

```
<?xml version="1.0"?>
<!DOCTYPE validators PUBLIC
    "-//Apache Struts//XWork Validator 1.0.2//EN"
    "http://struts.apache.org/dtds/xwork-validator-1.0.2.dtd">

<validators>
    <field name="solde">
        <field-validator type="double">
            <param name="minInclusive">100</param>
            <message key="solde.invalide"/>
        </field-validator>
    </field>
    <field name="type">
        <field-validator type="requiredstring">
            <message key="type.invalide"/>
        </field-validator>
    </field>
</validators>
```

Validation des formulaires pour l'action getSubForm :

AbonnementAction-getSubForm-validation.xml

```
<?xml version="1.0"?>
<!DOCTYPE validators PUBLIC
    "-//Apache Struts//XWork Validator 1.0.2//EN"
    "http://struts.apache.org/dtds/xwork-validator-1.0.2.dtd">

<validators>
    <field name="fidelio">
        <field-validator type="int">
            <param name="min">0</param>
            <param name="max">4000</param>
            <message>Entre 0 et 4000</message>
        </field-validator>
    </field>
    <field name="debit">
        <field-validator type="int">
            <param name="min">1</param>
            <param name="max">16</param>
            <message>Entre 1 et 16</message>
        </field-validator>
    </field>
</validators>
```

Fichier de propriétés

/AbonnementWEB/src/main/java/org/bp/web/package.properties

solde.invalid= Le solde doit être supérieur à 100

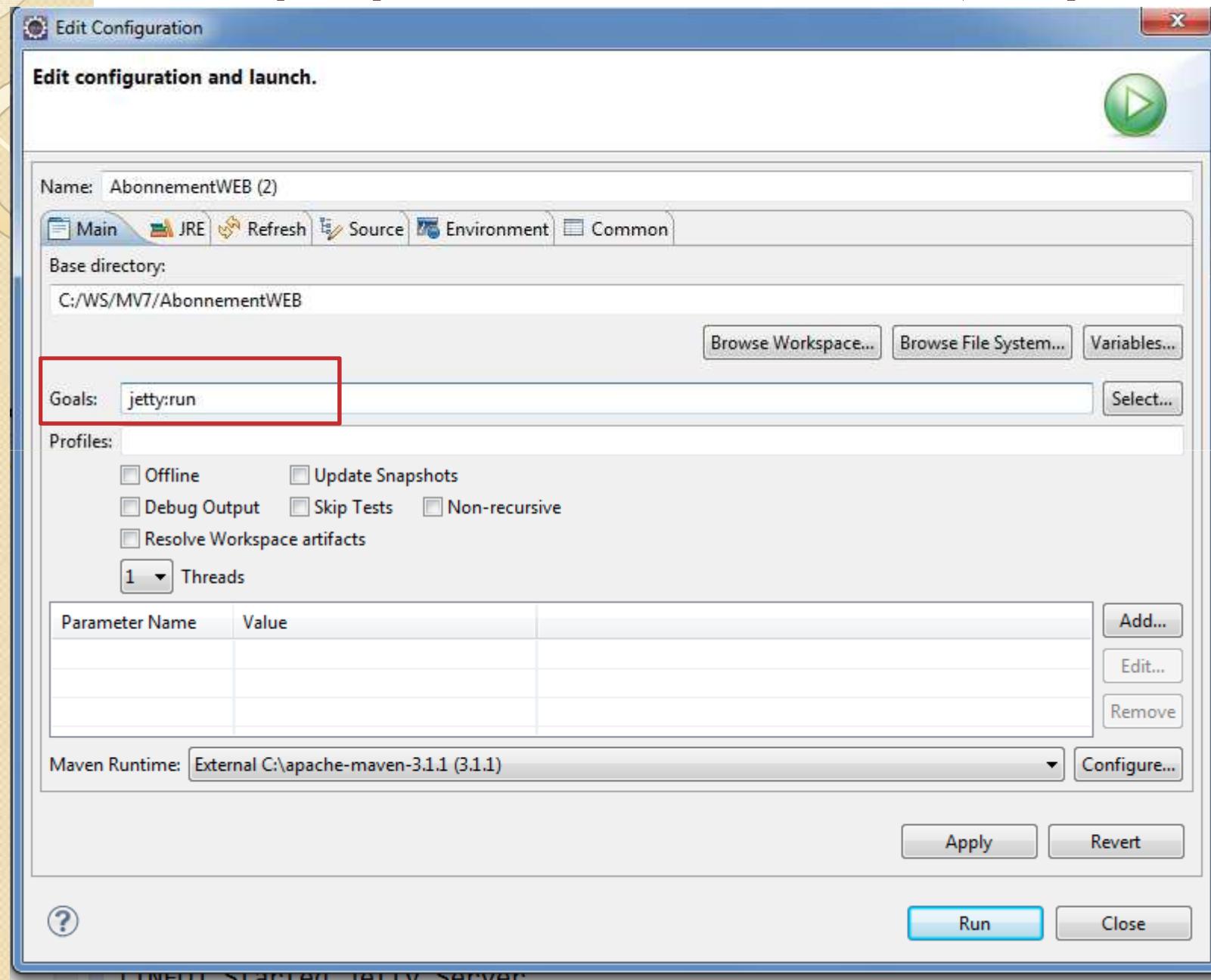
type.invalid=Type Invalid

/AbonnementWEB/src/main/java/org/bp/web/package_en.properties

solde.invalid= Anglais Le solde doit être supérieur à 100

type.invalid=Anglais Type Invalid

Déployer sur le serveur jetty



Test

Abonnements

localhost:8080/AbonnementWEB/edit.action;jsessionid=1i2g8ue4ahht61ouuutkuai

ID:	2
Solde:	9000.0
Date:	05/10/2014
Actif:	<input checked="" type="checkbox"/>
Type Ab:	INTERNET
Débit:	4
<input type="button" value="Save"/>	

ID	Date	Solde	Type	Fidelio	debit	Supp	Edit
1	05/10/14 00:00:00.000	5000.0	AbonnementGSM	0	4	Supp	Edit
2	10/05/14 10:04:33.000	9000.0	AbonnementInternet	0	4	Supp	Edit
3	05/10/14 00:00:00.000	50456.0	AbonnementGSM	0	4	Supp	Edit
4	10/05/14 10:07:59.000	9000.0	AbonnementInternet	0	4	Supp	Edit