جامعة الحسن الأول
UNIVERSITÉ HASSAN 1ER

UNIVERSITE HASSAN 1ER

ÉCOLE NATIONALE DES SCIENCES APPLIQUEES – BERRECHID

ENSA Ecole Nationale des Sciences Appliquées BERRECHID

DIAPORAMAS

(SUPPORT ELECTRONIQUE DE COURS)

PROGRAMMATION JEE

ROFESSEUR : LAHCEN MOUMOUN

DEPARTEMENT GENIE INFORMATIQUE & MATHEMATIQUES

Module : Programmation JEE

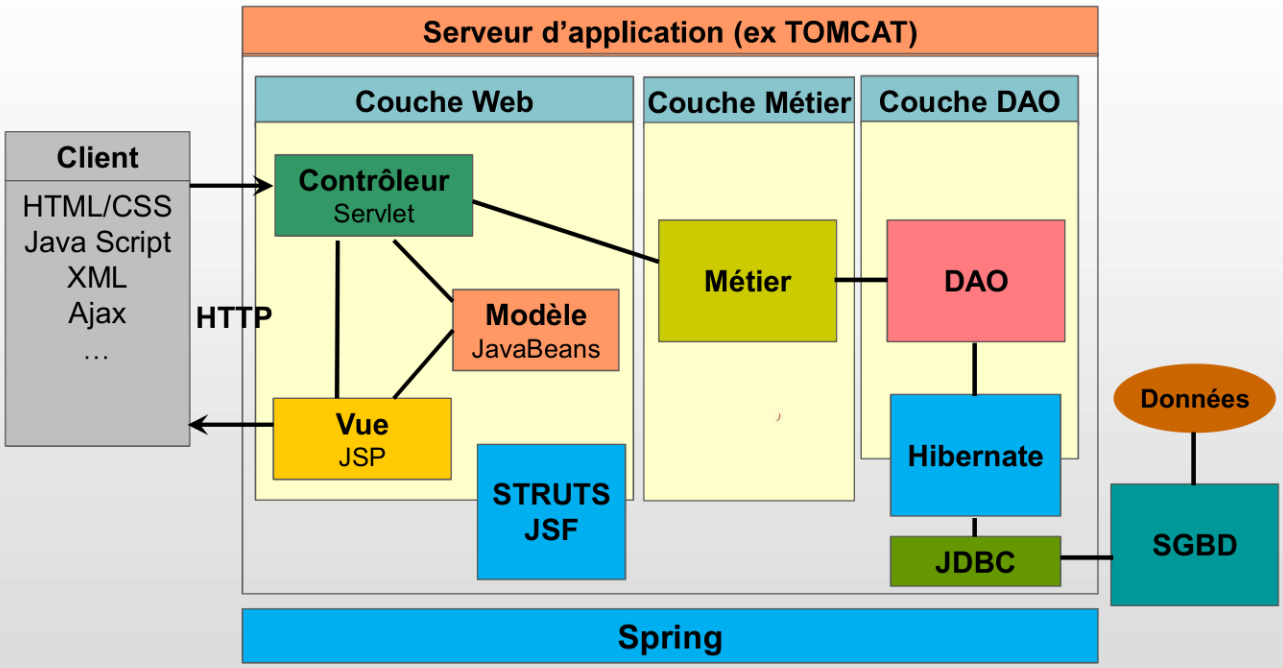
Chapitre 1: Les éléments de base du développement JEE

Pr LAHCEN MOUMOUN
estblearn@gmail.com

JEE

- JEE : Java Enterprise Edition(développée par Sun puis Oracle)
- JEE = JSE + plusieurs autres API
  - JSE : Java Standard Edition
  - API : Application Programming Interface
- Une technologie qui facilite le développement d'applications d'entreprise distribuées (développement d'applications web déployées et exécutées sur un serveur d'applications)

Architecture application JEE



Concept de Servlet

- Une servlet est une classe Java située et exécutée coté serveur en utilisant ,ainsi, ses ressources(puissance de calcul ou d'accès aux bases de données).
- Elle permet d'étendre les fonctionnalités du serveur Web avec la possibilité de générer du contenu dynamique en réponse à des requêtes HTTP clients.
- Une Servlet est compilée (indépendant de la plate-forme/du serveur) et exécute dans un conteneur de servlet

Serveur application

Serveurs aussi qualifies de conteneurs de servlets : gère le cycle de vie des servlets (création, initialisation, gestion, destruction) :

Tomcat (Apache)

http://tomcat.apache.org/

http://jakarta.apache.org/

GlassFish (oracle)

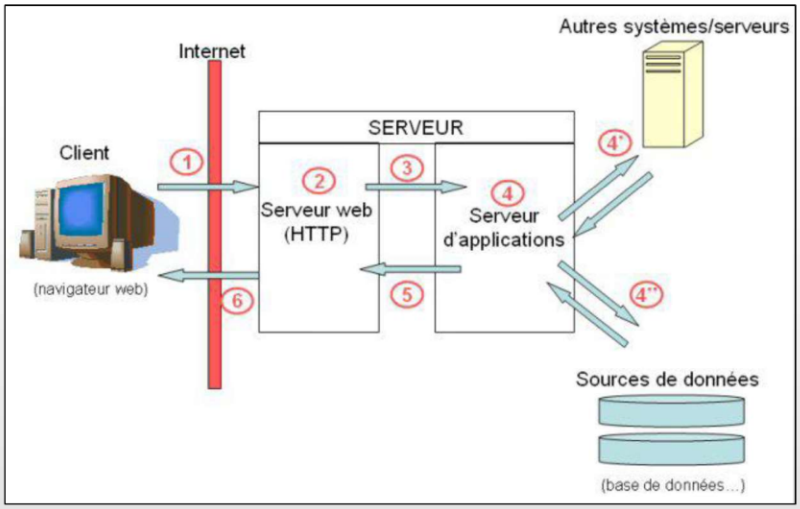
https://glassfish.java.net/

JBoss (Red Hat)

www.**jboss**.org

Websphere (IBM)

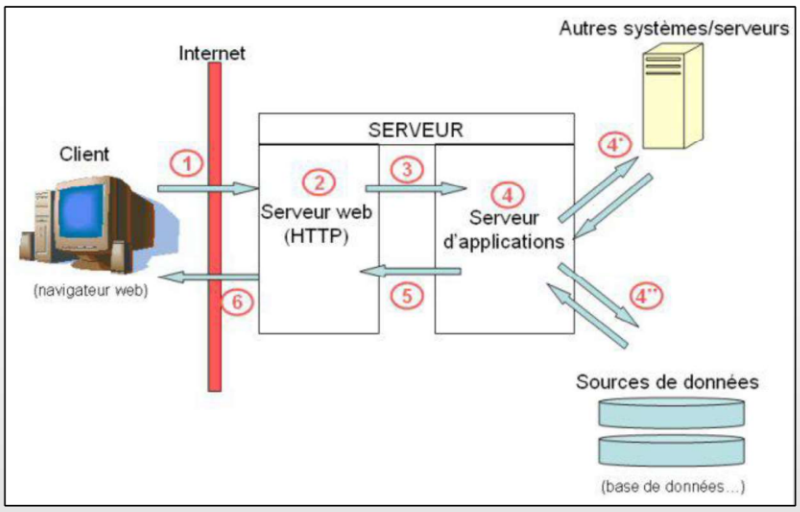
Fonctionnement d'une servlet



(1) Le client émet une requête pour demander une ressource au serveur (la réponse peut être statique par envoi de page HTML ou dynamique générée par une application web).

(2) Le serveur web (exemple : Apache) traite les requêtes HTTP entrantes (demande de ressource statique ou dynamique). Un serveur HTTP ne sait répondre qu'aux requêtes visant des ressources statiques (pages HTML, des images et données existantes).

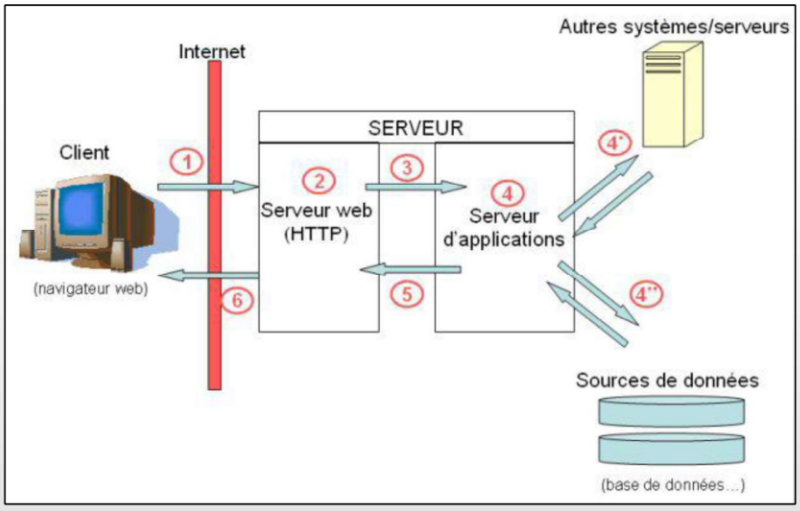
Fonctionnement d'une servlet



(3) si le serveur HTTP s'aperçoit que la requête reçue est destinée au serveur d'applications, il la lui transmet. Les deux serveurs sont reliés par un canal, nommé "connecteur".

(4) Le serveur d'applications (exemple : Tomcat) reçoit la requête à son tour pour la traiter. Une servlet est donc invoquée, et le serveur lui fournit deux objets Java : la requête et la réponse. Une consultation de sources de données est possible (4'') et l'interrogation d'autres serveurs ou systèmes est possible(4').

Fonctionnement d'une servlet



(5) Le serveur d'applications renvoie la réponse générée au serveur web. Celui-ci la récupère comme s'il était lui-même allé chercher une ressource statique (il a simplement délégué la récupération de la réponse).

(6) La réponse est dorénavant du simple code HTML, compréhensible par un navigateur. Le serveur HTTP peut donc retourner la réponse au client.

Installation Tomcat

- Tomcat est un conteneur (ou moteur d'exécution) utilisé pour l'implémentation de référence des servlets.
- Pour l'installer Tomcat , téléchargez une version récente de Tomcat au format ZIP disponible à « http://tomcat.apache.org/ » puis décompressez l'archive Zip.

Tomcat (Apache)

Intégration Tomcat dans Eclipse

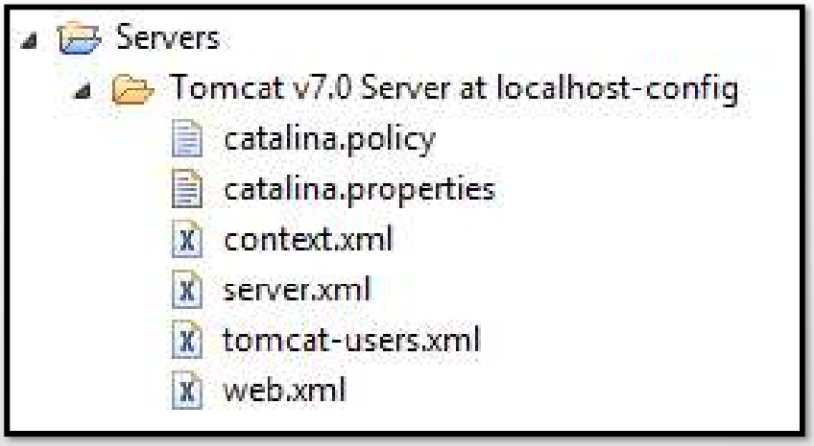
Dans l'environnement Eclipse:

- Menu New → Other ensuite choisir Server/Server
- Choisissez Apache/Tomcat vX.X Server
- Cliquez sur 'Browse' puis naviguez jusqu'au répertoire d'installation de tomcat.
- Une nouvelle vue 'Servers' est apparue, vous informant que Tomcat est bien intégré à Eclipse, avec un statut Arrêté (Stopped).

  - Double click sur le serveur et choix de l'option « Server locations → Use Tomcat installation ».
  - Propriété du serveur cliquer sur « Switch location ».
- Démarrer (ou arrêter) Tomcat. On peut vérifier que le serveur est bien lancé en accédant , depuis un navigateur, à l'adresse « http://127.0.0.1:8080/ ».

Intégration Tomcat dans Eclipse

Lors de l'intégration de Tomcat, Eclipse fait **une copie** des fichiers de configuration du Tomcat installé dans le workspace.



Le fichier « web.xml » définit les mappages entre les chemins d'URL et les servlets qui traitent les requêtes avec ces chemins.

Création projet Web dans Eclipse

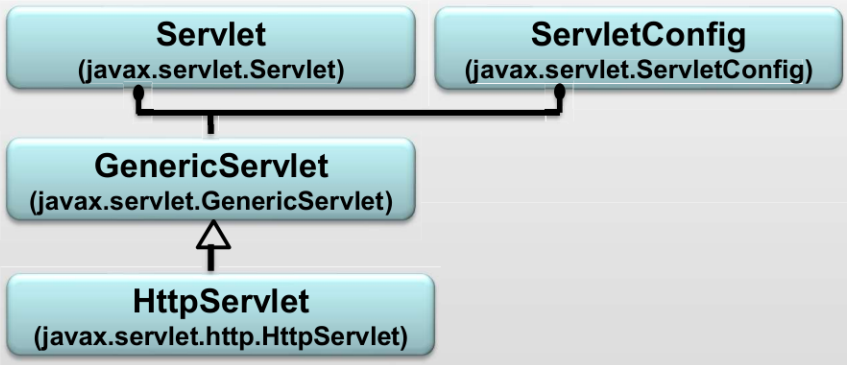
- A partir d'Eclipse, Choisir menu File → New → Dynamic Web Project
- Attribuer un nom au projet et spécifier le serveur d'application (Tomcat vX.X).
- Cocher la case « Generate web.xml deployment descriptor ». le fichier web.xml est un descripteur de déploiement qui assure la liaison entre une URL et la servlet (.class).

API Servlet

- L'API Servlet fournit un certain nombre de classes et d'interfaces permettant :
  - Le développement des Servlets,
  - Le déploiement qui consiste en l'installation des fichiers (servlets, html …) sur le conteneur de servlets
  - La mise en œuvre des Servlets au sein du conteneur Web.
- L'API Servlet est définie principalement par deux packages (fait partie des spécifications JEE) :
  - javax.servlet qui contient les classes pour un support des Servlets génériques (indépendant du protocole).
  - javax.servlet.http qui contient des extensions des classes du package javax.servlet pour prendre en considération des fonctionnalités spécifiques au protocole HTTP.

API Servlet

Une Servlet doit implémenter l'interface Servlet soit directement soit par l'extension des classes abstraites GenericServlet (Servlet utilisateur indépendante du protocole) ou HttpServlet (Servlet http).



Canevas d'une Servlet HTTP

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class …… extends HttpServlet {
….
}
```

- Les classes HttpServlet est définie dans le pakcage javax.servlet.http
- IOException, définie dans le package java.io,
- ServletException, définie dans le package javax.servlet.

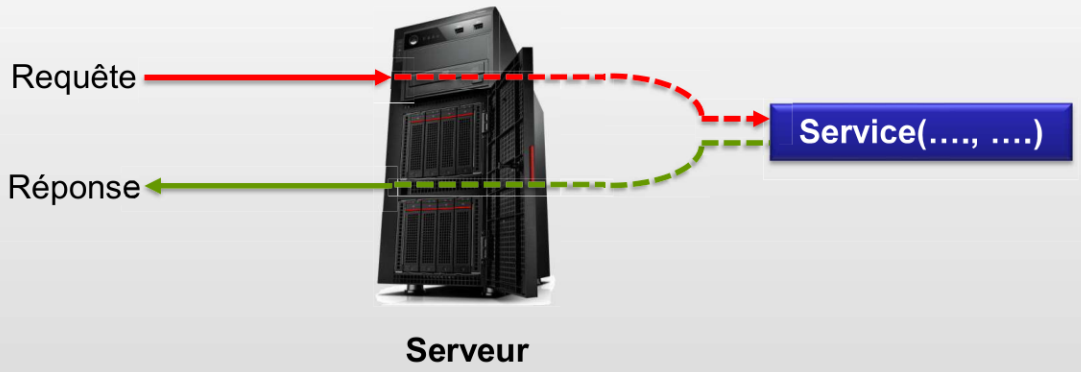
Cycle de vie d'une Servlet

Une Servlet suit un modèle de programmation requête/réponse. Pour gérer le cycle de vie d'une Servlet, l'interface javax.servlet.Servlet possède les méthodes :

- **void init(ServletConfig config) throws ServletException**
  Permet d'initialiser la Servlet. On peut , ainsi, se connecter à une BD , récupérer des informations générales sur le contexte d'exécution…
- **void service(ServletRequest req, ServletResponse res) throws ServletException, IOException**
  Elle sera invoquée automatiquement à chaque fois que la Servlet reçoit une requête client (paramètre **ServletRequest**). Les informations nécessaires pour une communication vers le client (réponse) sont définies par le paramètre **ServletResponse.**
- **Void destroy()**
  Utiliser pour détruire la servlet et ses ressources.

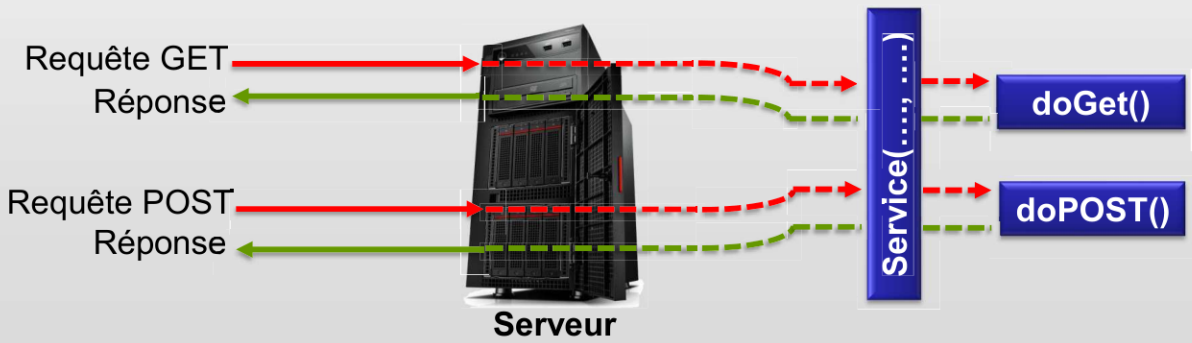
Cycle de vie d'une Servlet

Une Servlet générique (dérivé de la classe **GenericServlet**) doit redéfinir la méthode Service alors qu'une Servlet http (dérivé de la classe **HttpServlet**) ne doit pas le faire.



Servlet HTTP

- La méthode service() de HttpServlet (n'est pas à redéfinir) prend en charge l'appel automatique de la bonne méthode doGet / doPost (ou autre) en fonction du type de requêtes.
  - Requête GET : demander les informations et les données de l'URI (Uniform Resource Identifier. la forme d'URI la plus utilisée est l'URL: Uniform Resource Locator).
  - Requête POST : envoie de données (ex : formulaire) traitées par l'URI



La classe HttpServlet

- Une Servlet Http doit redéfinir l'une des méthodes doGet et doPost (ou autre).
  - void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
  - void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
- Le paramètre HttpServletRequest est utilisé pour la réception de données de la requête alors que le paramètre HttpServletResponse sert à l'envoi de données de réponse .
- En cas de non traitement des exceptions ServletException et IOException (si une telle exception survient), le serveur de servlet est programmé par défaut pour renvoyer un message d'erreur au client de la servlet.

L'interface HttpServletRequest

Un objet encapsule des données de la requête, ses principales méthodes sont :

- String getParameter(String parametre) : valeur d'un paramètre
- String [ ] getParameterValues(String parametre) : valeurs des paramètres (cas d'une liste)
- void setAttribute(String nom, Object obj)
- Object getAttribute(String nom)
- Cookie[ ] getCookies()
- HttpSession getSession()

L'interface HttpServletResponse

Un objet HttpServletResponse permet d'écrire des données texte au sein du navigateur, ses principales méthodes sont :

- La méthode ***setContentType*** *permet de faire* prévenir le navigateur du client du format d'information (une chaîne type / sous-type) qu'on va lui transmettre On trouve par exemple les types *text/html*, *text/xml*, *image/jpeg*, *image/gif*, *image/pjpeg*, *image.pgn*.
- La méthode ***getWriter*** fournit un flux de type *PrintWriter* qui sera automatiquement connecté au client correspondant (plus précisément à son navigateur). Il suffit d'afficher classiquement les informations voulues, en recourant aux méthodes de la classe *PrintWriter*, par exemple *print* ou *println*.

1ier exemple de Servlet

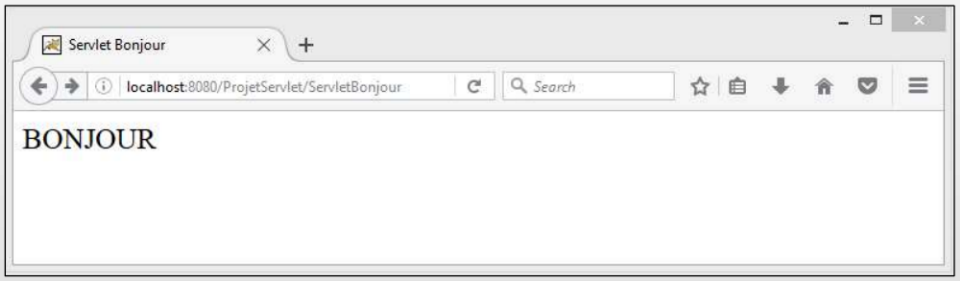
```
import java.io.* ;
import javax.servlet.* ;
import javax.servlet.http.* ;

public class ServletBonjour extends HttpServlet {
   public void doGet (HttpServletRequest req, HttpServletResponse rep)
                              throws IOException, ServletException {
        rep.setContentType ("text/html") ; PrintWriter page = rep.getWriter() ;
        page.println ("<html>") ; page.println ("<head>") ;
        page.println ("<title> Servlet Bonjour </title>") ;
        page.println ("</head>") ;
        page.println ("<body>") ;
        page.println ("<font size=+2>") ;
        page.println ("BONJOUR") ;
        page.println ("</body>") ; page.println ("</html>") ;
  }
}
```

1ier exemple de Servlet

1ère Invocation de la méthode doGet(…) : Saisie de l'URL de la Servlet dans la barre d'adresse du navigateur.

http://localhost:8080/ProjetServlet/ ServletBonjour



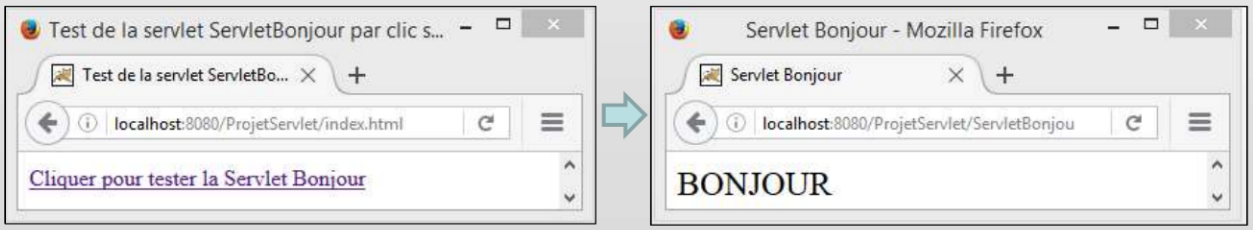
1ier exemple de Servlet

2ième Invocation de la méthode doGet(…): Clique sur un lien hypertexte qui pointe sur l'URL de la Servlet.

http://localhost:8080/ProjetServlet/index.html

Index.html

```
<html>
<head> <title>Test de la servlet ServletBonjour par clic sur lien </title> </head>
<body>
    <P> <A href="/ProjetServlet/ServletBonjour "> Cliquer pour tester la Servlet Bonjour </A> </P>
</body>
</html>
```



Utilisation de la méthode POST

La méthode doPost() est invoquée principalement lors de l'envoie des données saisies dans un formulaire HTML (par un clic sur un bouton de type submit).

Index.html

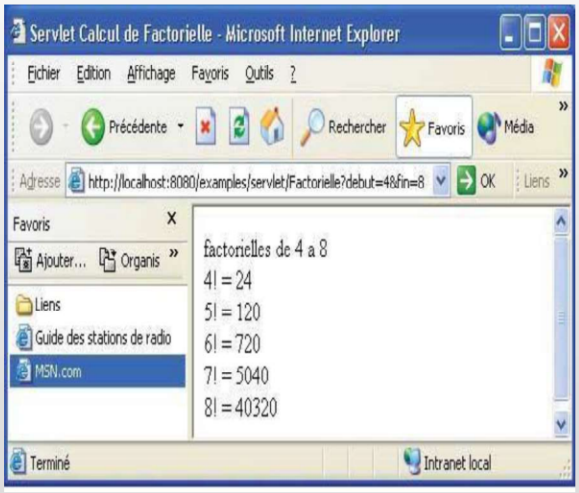
```
<html> <head> <title>Test de la servlet ServletBonjour </title> </head>
<body> <FORM action = "/ProjetServlet2/ServletBonjour2" method ="post">
<P> Prenom : <INPUT type ="text" name="prenom"> <BR>
      Nom : <INPUT type ="text" name="nom"> <BR> <INPUT type ="submit" value="Valider"> </P>
</FORM> </body> </html>
```

ServletBonjour2.Java

```
import java.io.* ; import javax.servlet.* ; import javax.servlet.http.* ;
public class ServletBonjour2 extends HttpServlet {
  protected void doPost(HttpServletRequest req,
            HttpServletResponse res) throws ServletException,
            IOException {
     String prenom = req.getParameter("prenom");
     String nom = req.getParameter("nom");
     res.setContentType("text/html");
     PrintWriter out = res.getWriter();
     out.println("<HTML><BODY>");
     out.println("<H1>Bonjour " +prenom+" "+ nom+"."+ "</H1>");
     out.println("</BODY> </HTML>");  }
}
```

Exemple 2

Écrire une servlet nommée *Factorielle*, permettant au client d'afficher les valeurs de factorielles de nombres entiers situés dans un intervalle dont il choisira les bornes à l'aide du formulaire HTML.



Exemple 2

Facto.html

```
<html> <head> <title>Servlet et calcul du Factoriel </title></head><body>
  <h1> Calcul des factoriels des nombres</h1>
  <FORM action = "/PrjFact/Fact" method ="post">
   <P> de: <INPUT type ="number" name="valDebut"> a : <INPUT type ="number" name="valFin">
        <BR><BR> <INPUT type ="submit" value="Ok">  </P>
  </FORM>
</body></html>
```

Fact.Java

```
protected void doPost(HttpServletRequest req, HttpServletResponse rep) throws ServletException,
IOException {
    int valDebut=Integer.parseInt(req.getParameter("valDebut"));
    int valfin=Integer.parseInt(req.getParameter("valFin"));
    rep.setContentType("text/html"); PrintWriter out=rep.getWriter();
    out.println("<head> <title> Servlet et calcul du Factoriel </title></head><body>");
    out.println("<h4> Factoriels de " + valDebut+ " à " + valfin+ "</h4>");
    long f=1L;
    for(int i=2; i<valDebut; i++)f*=i;
    for(int i=valDebut; i<=valfin; i++) {
       f*=i; out.println(i+" ! = "+ f+"<BR>"); }
    out.println("</body></html>");
}
```


L'interface ServletContext

- Les informations de configuration d'une application Web sont représentées par un objet de type javax.servlet.ServletContext (chaque Servlet d'une même application Web a accès à ces informations).
- Les Méthodes de l'interface ServletContext dédiées à la récupération des paramètres globaux d'initialisation sont:
  - public String getInitParameter(String nom) : Récupérer une chaîne de caractères contenant la valeur d'un paramètre nommé nom ou la valeur null si le paramètre n'existe pas.
  - public java.util.Enumeration getInitParameterNames(): Récupérer sous la forme d'un objet de type java.util.Enumeration l'ensemble des noms des paramètres déclarés pour la Servlet.

L'interface ServletContext

Les méthodes de l'interface ServletContext dédiées à la gestion des attributs du contexte d'application sont :

- public String setAttribute(String nom, Object objet) : Créer un attribut dans le contexte de l'application Web. Si le nom de l'attribut existe déjà, la valeur existante est remplacée par la nouvelle.
- public Object getAttribute(String nom) : Récupérer la valeur d'un attribut dont le nom est passé en paramètre, ou la valeur null si l'attribut n'existe pas.
- public java.util.Enumeration getAttributeNames(): Récupérer sous la forme d'un objet de type java.util.Enumeration le nom de tous les attributs stockés dans l'application Web.
- public void removeAttribute(String nom): Supprimer un attribut du contexte de l'application Web, dont le nom est passé en paramètre.

Les informations contenues dans la requête

De nombreuses informations en provenance du client peuvent être extraites de l'objet ServletRequest (ou de HttpServletRequest).

- public String getScheme( ) : Retourne le nom du protocole utilisé par le client pour émettre sa requête. Par exemple : http, ftp, etc.
- public String getContextPath( ) : Retourne sous la forme d'une chaîne de caractères commençant par un /, la portion de l'URL de la requête correspondant au nom du contexte de l'application Web.
- public String getMethod(): Retourne le nom de la méthode HTTP(GET, POST, etc) utilisée par le client pour émettre sa requête.
- public String getRequestURL( ) : Retourne l'URL que le client a utilisée pour émettre sa requête.
- public String getServletPath( ) : Retourne la partie de l'URL qui invoque la Servlet , composée du chemin et du nom ou de l'alias de la Servlet Par exemple : /ProjetServlet/ServletBonjour.

Le fichier web.xml

- Le tag <servlet> permet de définir une servlet.
  - Le tag <servlet-name> permet de donner un nom à la servlet qui sera utilisé pour le mapping avec l'URL par défaut de la servlet.
  - Le tag <servlet-class> permet de préciser le nom complètement qualifié de la classe Java dont la servlet sera une instance.
- Le tag <servlet-mapping> permet d'associer la servlet à une URL. Ce tag possède les tags fils <servlet-name> et <servlet-mapping>.

```
<servlet>
  <servlet-name>MaServlet</servlet-name>
  <servlet-class>Package.MaServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MaServlet</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
```


Exercice

Réaliser une servlet permettant de répondre à la page suivante:

Lorsque l'utilisateur soumet ces données, la servlet effectuera des contrôles élémentaires sur leur intégrité:

- Nom, prénom que des caractères alphabétiques
- Code postal 5 chiffres
- Adresse 2 chaînes arbitraires délimitées par "@"

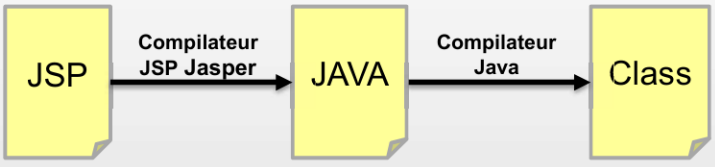
Une page montrera alors le résultat de ces vérifications à l'utilisateur.

Notion de JSP

- JSP (Java Server Pages) est un langage de scripts composé à la fois de balises HTML (ou XML) et d'instructions provenant du langage Java. Il sert d'extension à un serveur Web et permet la création et le renvoi de pages HTML comportant du contenu dynamique.
  - Servlet = du code Java contenant de l'HTML
  - JSP = une page HTML contenant du code Java
- Les scripts JSP sont des instructions Java embarquées dans du code HTML entre les balises <% %>
- La technologie JSP mélange la puissance de Java côté serveur et la facilité de mise en page d'HTML côté client (les pages JSP possèdent par convention l'extension .jsp).

JSP et Servlet associée

- Au premier appel de la page JSP, le moteur de JSP génère et compile automatiquement une servlet (classe étendant HttpJspBase) qui permet la génération de la page web (les JSP sont un moyen d´écrire des servlets).



- La servlet générée est compilée et sauvegardée puis elle est exécutée. Elle est invoquée à chaque requête
- Les appels suivants de la JSP sont beaucoup plus rapides car la servlet, conservée par le serveur, est directement exécutée.

Scripts d'une page JSP

- Commentaires JSP(ne sont pas produits en sortie): **<%-- commentaire --%>**
  Les commentaires HTML(sont transmis à la réponse): **<!-- commentaire -->**
- Expressions : **<%= expression java %>**
  Exemple : <%= java.util.Date() %>
- Scriplets : **<% instructions java %>**
- Déclarations (attributs et méthodes, utilisables dans toute la JSP) :
  <%! Déclarations de variables et de méthodes %>

  Exemple :

```
<% String resume = "Du texte, du texte, du texte, du texte."; %>
<body>
   Le résumé:  <%= resume %>
</body>
```

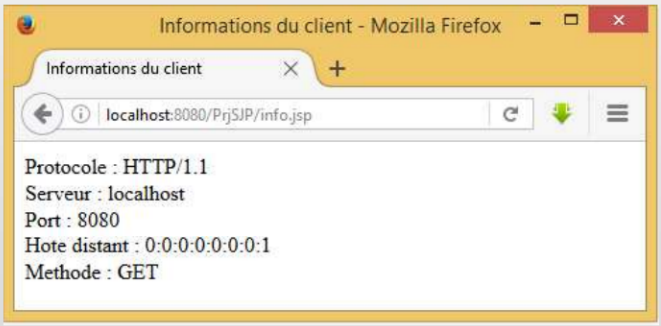

Les objets implicits

Représentent un certain nombre d'objets prédéfinies qui peuvent être immédiatement utilisés dans une expression ou une scriptlet JSP :

- out (instance de la classe javax.servlet.jsp.JspWriter): cet objet représente le canal de sortie utilisé pour communiquer la réponse au client (navigateur).
- request (instance de javax.servlet.http.HttpServletRequest) : cet objet possède les méthodes qui permettent d'accéder aux informations de la requête en cours.
- response (instance de javax.servlet.http.HttpServletResponse): cet objet contient la réponse de la page JSP

Exemple d'utilisation des objets implicits

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page language="java" %>
<html><head><title>Informations du client</title></head>
<body bgcolor="white">
    Protocole : <%= request.getProtocol() %><br>
    Serveur : <%= request.getServerName() %><br>
    Port : <% out.println(request.getServerPort()); %><br>
    Hote distant : <% out.println(request.getRemoteHost()); %><br>
    Methode : <%= request.getMethod() %><br>
</body></html>
```



La directive « <%@page ... » définit les options de configuration de la page

Exemple de page JSP

```
<%@ page contentType =" text / html ; charset =utf -8" %>
<! DOCTYPE html >
<html >
<head > <title >Ma première page JSP </ title > </head>
<body>
      <% String prenom = request.getParameter ("prenom"); %>
      <h1 > Bonjour
      <%=( prenom != null && prenom.length () !=0) ? prenom :"inconnu(e)"%> </h1 >
      <% if ( prenom != null && prenom.equals ("ENSAB")) { %>
          <h2 > Bien joué !!!! </h2 >
      <% } else { %>
        <form action ="TestJSP.jsp" method="post">
          <label> Prénom : </label >
          <input type ="text" name="prenom" size ="30">
          <input type ="submit" value =" envoyer ">
        </form >
      <% }  %>
</body >
</html >
```

Principe de traduction d'une page JSP en Servlet

- Les variables et les méthodes déclarées dans une balise <%! … %> sont traduites en variables et méthodes d'instance.
- Les variables déclarées à l'intérieur d'une scriptlet sont transformées en variables locales d'une méthode appelé _jspService().
- Les expressions JSP de type <%=… %> sont placées dans un flux de sortie de type out.println(…);
- Les commentaires JSP sont ignorés.
- Les balises HTML sont disposées à l'intérieur de la méthode _jspService() dans leur ordre d'arrivée et sont placées à l'intérieur d'un flux d'écriture de type out.println( ).
- Le code Java défini à l'intérieur des scriptlets est placé tel quel à l'intérieur de la méthode _jspService().

Cycle de vie d'une page JSP

- Le fichier source d'une servlet créée à partir d'une JSP appelée pageName se trouve dans un fichier appelé : pageName_jsp.java (la localisation dépend du serveur JEE utilisé)

  Par exemple, sous Eclipse avec Tomcat, dans :

  …/work\Catalina\localhost\nom_projet\org\apache\jsp
- Une fois la JSP traduite et compilée : le cycle de vie suit le même principe que pour une servlet normale :
  - Si la servlet n'existe pas encore, le conteneur charge la classe de la servlet, l'instancie puis initialise l'instance avec la méthode **jspInit**
  - Le conteneur invoque la méthode _jspService avec passage des objets request et response
  - Si le conteneur a besoin d'enlever la servlet, il invoque la méthode jspDestroy
- En cas d'erreur en phases de traduction ou de compilation le serveur renvoie (lors du premier chargement de la JSP) une JasperException et un message qui inclue le nom de la JSP et la ligne d'erreur.

Exemple de conversion JSP en Servlet

Testjsp.jsp

```
…
<title >Ma première page JSP </ title >
</head >  <body >
<% String prenom = request.getParameter ("prenom"); %> <h1 > Bonjour
…
```

```
public final class Testjsp_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
      …
      public void _jspInit() {   }
      public void _jspDestroy() {    }
      public void _jspService(final javax.servlet.http.HttpServletRequest request, final
            javax.servlet.http.HttpServletResponse response)  throws
            java.io.IOException, javax.servlet.ServletException {  …
        javax.servlet.jsp.JspWriter out; …
        out.write("<title>Ma première page JSP </title>\r\n");  …
        String prenom = request.getParameter ("prenom");  out.write("\r\n");
        out.write("\t<h1 > Bonjour \r\n"); …
      }
}
```


Directives de page JSP

- Les directives vont agir sur l'étape de compilation (JSP --> Java). Elles permettent de spécifier des informations globales sur la page

  <%@ directive attribut="valeur" %>

- 3 types de directives sont disponibles :
  - La directive « **Page** » définit les options de configuration de la page
  - La directive « **Include** » assure l'inclusions de fichiers statiques. Le fichier tout entier (variables, entrée/sortie etc..) est incorporé dans la JSP au moment de la transformation en Servlet.
  - La directive « Taglib » permet de définir des tags d'actions personnalisées.

Principaux attributs de la directive page

- <%@page import="java.util.*, java.sql.Connection" %>
- <%@page contentType="text/html; charset=ISO-8859-1" %>
- <%@page session="true|false" %>

  Indique si la page est incluse ou non dans une session. Par défaut true, ce qui permet d'utiliser un objet HttpSession pour gérer des données de session.
- <%@page errorPage="relativeURL" %>

  Précise la JSP appelée au cas où une exception est levée (URL relative par rapport au répertoire qui contient la page JSP ou relative par rapport au contexte de l'application Web si elle débute par /)
- <%@page isErrorPage=" true|false" %>

  Précise si la page JSP est une page de gestion d'erreur (dans ce cas l'objet Exception peut être utilisée dans la page), false par défaut.
- …

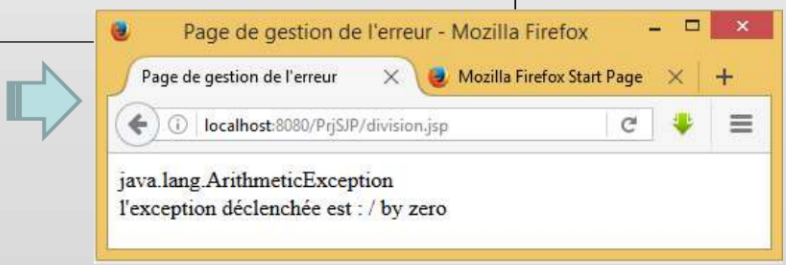
Exemple de la directive page

division.jsp

```
<%@ page language="java" contentType="text/html" %>
<%@ page errorPage="/erreur.jsp" %>
<html><head><title>Page avec une erreur</title></head>
<body> <% int var=90; %> Division par <% var = var/0; %> <%= var %>
</body></html>
```

erreur.jsp

```
<%@ page language="java" contentType="text/html" %>
<%@ page isErrorPage="true" %>
<html><head><title>Page de gestion de l'erreur</title></head>
<body><%=exception.getClass().getName()%><br>
l'exception déclenchée est : <%= exception.getMessage()%>
</body></html>
```



Gestion de session

- Cette fonctionnalité consiste à suivre l'activité du client sur plusieurs pages. Elle est supportée par les servlets à l'aide de l'objet HttpSession obtenu à l'aide de request.getSession().
  - HttpSession maSession = request.getSession(true|false) : retourne la session courante pour cet utilisateur (la valeur true en paramètre) ou une nouvelle session.
  - Les méthodes void putValue(String name, Object value) et Object getValue(String name) permettent respectivement d'attribuer et de consulter une valeur de session identifiée par son nom.
- L'objet implicite session (instance de la classe javax.servlet.http.HttpSession) est par défaut disponible à toutes les pages JSP qui participe à une session sauf si l'attribut session de la directive page est positionné à faux.

Méthodes d'un objet de type HttpSession

- void setAttribute(String name, Object value) : ajoute un couple (name, value) pour cette session
- Object getAttribute(String name) : retourne l'objet associé à la clé name ou null
- void removeAttribute(String name) : enlève le couple de clé name
- java.util.Enumeration getAttributeNames() : retourne tous les noms d'attributs associés à la session
- void setMaxIntervalTime(int seconds) : spécifie la durée de vie maximum d'une session
- long getCreationTime() / long getLastAccessedTime() : retourne la date de création / de dernier accès de la session en ms depuis le 1/1/1970, 00h00 GMT new Date(long)

Portée des objets

Scope objets	Objet implicite / Classe	Accessibilité depuis
Web context	**application /** javax.servlet.ServletContext	Les composants web d'une application
session	**session /** javax.servlet.http.HttpSession	Les composants web en rapport avec les requêtes d'une session (maintient de l'état client)
request	**request /** javax.servlet.ServletRequest	Les composants web en rapport avec une requête
page	**context /** javax.servlet.jsp.PageContext	La page JSP qui a créé l'objet

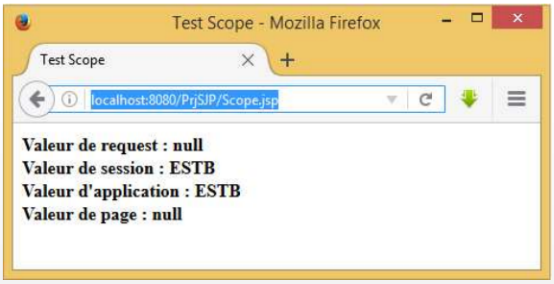
Transmission de paramètres à une page JSP

- Dans une servlet, il était nécessaire de savoir comment les paramètres étaient transmis de façon à redéfinir, soit la méthode doGet, soit la méthode doPost. Dans le cas du JSP, cette distinction n'existe plus (la servlet générée est formée essentiellement d'une méthode nommée _jspService).
- Pour transmettre des paramètres à une JSP nous pouvons soit :
  - Saisir l'adresse du JSP dans le navigateur, en l'accompagnant des valeurs (tout caractère non alphanumérique doit être codé en hexadécimal sous la forme %codeHexa) de ses paramètres. l'appel de JSP se présentera ainsi :

    http://localhost:8080/ProjetJSP/FichierJSP.jsp?parametre1=valeurPar1 & parametre2=valeurPar2&…
  - Utiliser un formulaire HTML. À notez que, bien que la méthode de transmission des paramètres (GET ou POST) n'intervienne pas dans l'écriture du JSP, celle-ci doit être précisée dans le formulaire HTML.

Transmission de paramètres à une page JSP

```
<html> <head>
<title>Test Scope </title> </head>
<body>
<%
    String nom =request.getParameter("nom");
    if (nom!="" && nom!=null){
    session.setAttribute("session_nom", nom);
    application.setAttribute("application_nom", nom);
    pageContext.setAttribute("pageContext_nom",
       nom, pageContext.SESSION_SCOPE);
    }
%>
<b> Valeur de request : <%= nom %></b> <br>
<b> Valeur de session : <%= (String) session.getAttribute("session_nom") %></b><br>
<b> Valeur d'application : <%= (String) application.getAttribute("application_nom") %>
</b> <br>
<b> Valeur de page : <%= (String) pageContext.getAttribute("pageContext_nom") %>
</b> <br>
</body>  </html>
```



Inclusion de JSP

- La directive « **Include** » sert à inclure d'autres fichier dans la JSP. Avec le tag <%@, le fichier tout entier (variables, entrée/sortie etc..) est incorporé dans la JSP au moment de la **transformation en Servlet**..

  Exemple :

  <%@ include file = "unfichier.html" %>

- L'inclusion d'une page (statique ou dynamique) dans une JSP peut se faire à **l'exécution** (= inclusion du résultat) à l'aide du tag **<jsp:include**

  <jsp:include page="fichier " />

  ou <jsp:include page="fichier ">
  <jsp:param name="nom " value="valeur "/> ...
  </jsp:include>

- Ces tags sont particulièrement utiles pour insérer un élément commun à plusieurs pages tel qu'un en-tête ou un bas de page. Si le chemin du fichier commence par un '/', alors le chemin est relatif au contexte de l'application, sinon il est relatif au fichier JSP.

Exemple de la directive include

titre.txt

```
<h1> Titre de la JSP , inclus a partir d'un fichier texte </h1>
```

texte.jsp

```
<% out. println (" Texte genere par la JSP texte .jsp "); %>
```

parametre.jsp

```
<% String nom = request.getParameter ("nom"); out.println (" Salut " + nom );  %>
```

inclusion.jsp

```
<%@ page contentType =" text / html ;
   charset =UTF-8" %>  <! DOCTYPE html >
<html> <head ><title> JSP avec inclusion
 </title ></head>
<body ><%@ include file ="titre.txt" %> <p>
    <jsp:include page ="texte.jsp" /> </p> <p>
    <jsp:include page ="parametre.jsp">
        <jsp:param name ="nom" value ="ESTB" />
    </jsp:include > </p>
</body > </html >
```

Délégation de JSP

- Une JSP peut déléguer le traitement d'une requête à une autre JSP. Dans ce cas une prise en compte complète de la requête est réalisée par la JSP déléguée.

  Syntaxe

```
<jsp:forward page="fichier" />
```

  Ou

```
<jsp:forward page="fichier " >
    <jsp:param name="parametre1" value="valeur1"/>
    <jsp:param name="parametre2" value="valeur2"/>
</jsp:forward>
```

- Ce qui suit l'action forward est ignoré, et tout ce qui a été généré dans cette page JSP est perdu.

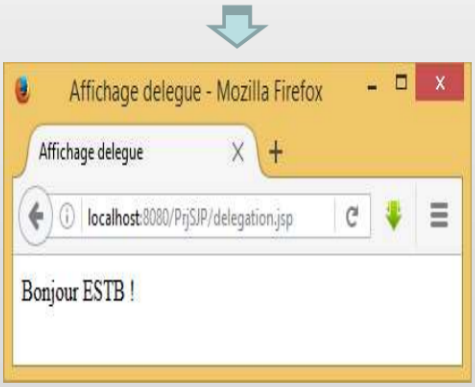
Délégation de JSP

delegation.jsp

```
<html >  <head > <title> JSP avec délégation </title></head >
<body >
<jsp:forward page ="parametre.jsp">
   <jsp:param name =" nom" value ="ESTB" />
</jsp:forward >
<p> Ceci ne peut pas être affiché vu qu 'on a délégué l'affichage !</p>
</body > </html >
```

parametre.jsp

```
<html > <head >
<title > Affichage delegue </ title ></head >
<body >
  <%
  String nom = request . getParameter ("nom");
  out.println ("<p> Bonjour " + nom + " !</p>");
  %>
</body > </html >
```



Partage de contrôle entre servlet et JSP

- Les Servlets et les JSP peuvent partager ou distribuer le contrôle de la requête. on utilise pour cela un objet RequestDispatcher.
- Dans la méthode de traitement de requête de Servlet :

  RequestDispatcher rd;

  rd= getServeletContext().getRequestDispatcher(String file);

  file indique un chemin relatif ou absolu ne pouvant pas sortir du contexte
- Redirection d'une requête: Dans la méthode de traitement de requête, Délégation du traitement à un JSP.

  rd.forward(request, response);
- Inclusion du résultat d'un JSP

  rd.include(request, response);

Partage de contrôle entre servlet et JSP

login.jsp

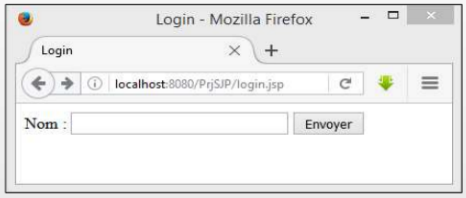
```
<html> <head> <title>Login</title> </head> <body>
<form action ="/PrjJSP/Navigation" method="post">
<label> Nom : </label > <input type ="text" name="nom" >
<input type ="submit" value =" Envoyer "> </form ><h3>
 <% String msg=(String)request.getAttribute("message");
  if (msg != null)out.println(msg); %> </h3> </body></html>
```

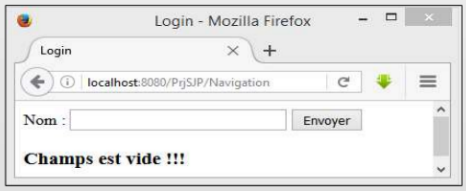
Navigation.java

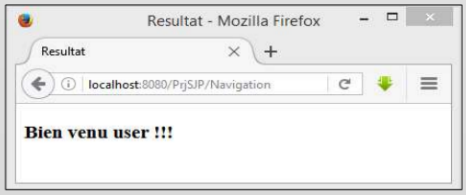
```
public class Navigation extends HttpServlet {
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
 String nom= (String)request.getParameter("nom"); String
 msg="";
 if(nom!="" && nom!=null){ msg ="Bien venu user !!!";
 request.setAttribute("message", msg);
 this.getServletContext().getRequestDispatcher("/WEB-
    INF/ result.jsp").forward(request,  response);  }
 else {  msg ="Champs est vide !!!!";
 request.setAttribute("message", msg);
 this.getServletContext().getRequestDispatcher("/login.jsp
    ").forward(request,  response);    }  }  }
```

WEB-INF/ result.jsp

```
<html> <head>
<title>Resultat</title>  </head>
<body> <h3>  <%=
  request.getAttribute("message")
%> </h3> </body> </html>
```







Flux et fichier

- En java, les Entrées et les Sorties (E/S) sont gérées par les objets de flux (la notion de flux désigne simplement un "canal" qui peut être connecté à différentes sources ou à différentes cibles : fichier, périphérique de communication…). On utilise des objets associés aux flux d'entrée et d'autres pour le flux de sortie.
- Java offre un nombre important de classes pour la manipulation des flux regroupées dans le package java.io. Java distingue les flux binaires des flux texte. Il permet, ainsi, d'accéder à un fichier binaire, soit de façon séquentielle, soit de façon directe.
  - **FileOutputStream, DataOutputStream, BufferedOutputStream, FileInputStream, DataInputStream, BufferedInputStream**
  - **RandomAccessFile**
  - **FileWriter PrintWriter, FileReader BufferedReader , File**

Exemple

UnMessage.java

```
package libMessage ;
import java.io .*; import java.text.*; import java.util.*;
public class UnMessage implements Serializable {
    private static SimpleDateFormat formatter =
                new SimpleDateFormat ("E d MMM yyyy , H:m:s", Locale.FRANCE);
    private Date date = new Date (); private String email ; private String texte ;

    public UnMessage () { date.setTime ( System.currentTimeMillis ()); }
    public String getDate () { return formatter.format ( date ); }
    public void setEmail ( String email ) { this . email = email ; }
    public String getEmail () { return email ; }
    public void setTexte ( String texte ) { this.texte = texte ; }
    public String getTexte () { return texte ; }
}
```


Exemple

LivreOr.java

La classe *ObjectInputStream* désérialise des objets précédemment écrits en utilisant un objet *ObjectOutputStream*.

```
package libMessage ;
import java . util .*; import java .io .*;
public class LivreOr implements Serializable {
  private ArrayList < UnMessage> messages ;
  private String nomFichier ;
  public LivreOr () { }
  public String getFichier () {  return this . nomFichier ;  }
  public void setFichier ( String fichier ) throws FileNotFoundException , IOException ,
          ClassNotFoundException {
    this . nomFichier = fichier ; FileInputStream fis ;  ObjectInputStream ois = null ;
    try { fis = new FileInputStream ( nomFichier );  ois = new ObjectInputStream ( fis);
          messages = ( ArrayList <UnMessage> ) ois.readObject (); ois.close(); }
    catch ( Exception e) { messages = new ArrayList <UnMessage> ();  }
  }
  public ArrayList<UnMessage> getMessages () { return messages ;}
  public void setMessages (ArrayList<UnMessage> messages){this.messages=messages;}
  public void addMessage ( UnMessage msg ) { messages .add (msg ); }
  public void enregistrer () throws FileNotFoundException , IOException {
    FileOutputStream fos = new FileOutputStream ( nomFichier );
    ObjectOutputStream oos = new ObjectOutputStream ( fos);
    oos . writeObject ( messages ); oos.close(); }
}
```

Exemple

LivreOr.jsp

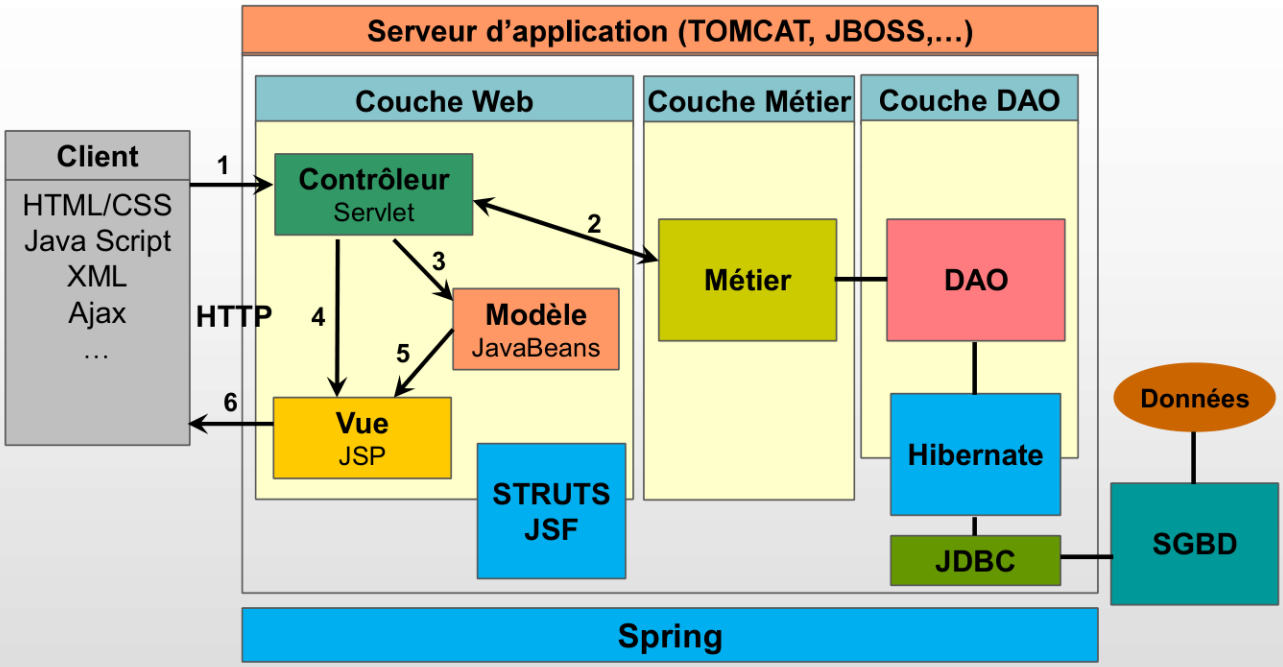
```
<body> <form action ='livredOr.jsp' method ='POST'>
   <label> email :</label> <input type ='text' name ='email' size ='50'/><br/>
   <textarea name=texte' rows ='10' cols='80'> message ici </textarea> <br/>
   <input type ='submit' name ='submit' value ='Envoyer'/> </form>
<% LivreOr livreor=(LivreOr) session.getAttribute("livreor");
 if(livreor==null){livreor=new LivreOr(); livreor.setFichier("messages.txt");}
 UnMessage msg= new UnMessage();msg.setEmail(request.getParameter("email"));
 msg.setTexte(request.getParameter("texte") );
 if(msg.getEmail()!=null && !msg.getEmail().equals("")){
 livreor.addMessage (msg ); livreor.enregistrer (); }
 if(livreor!=null&&livreor.getMessages()!=null&&
                          !livreor.getMessages().isEmpty()){
 for ( Object lemsg : livreor.getMessages ()) {  %>
<table border ="1" ><tr><td> <%= (( UnMessage ) lemsg).getEmail() %></td>
<td> <%= (( UnMessage ) lemsg ).getDate () %></td> </tr>  <tr>
<td colspan="2"> <pre> <%= ((UnMessage) lemsg ).getTexte() %> </pre>
</td >  </tr > </table>
<% }} %>
</body>
```

Module : Programmation JEE

Chapitre 2 : Framwork Hibernate

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

Le Framework Hibernate



Hibernate est une solution open source de type ORM (Object Relational Mapping) qui permet de faciliter le développement de la couche persistance d'une application.

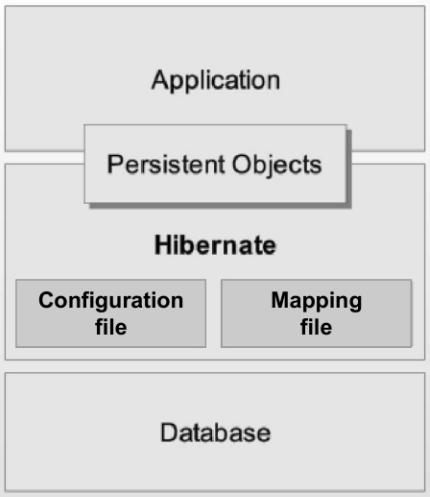
Hibernate et le mapping Objet / Relationnel

- Hibernate permet d'assure les fonctionnalités suivantes:
  - Assure le mapping des tables avec les classes, des champs avec les attributs, des relations et des cardinalités.
  - Propose une interface qui permette de mettre en œuvre des actions de type CRUD(create - read - update – delete).
  - Propose un langage de requêtes indépendant de la BD cible et assurer une traduction en SQL natif selon la BD utilisée.
  - Supporte les transactions et gère les accès concurrents.
  - …
- Hibernate permet de réduire (jusqu'à 90%) le temps de développement du code d'accès à la base de données.

Hibernate et le mapping Objet / Relationnel

Hibernate a besoin de plusieurs éléments pour fonctionner :

- Des classes de type JavaBean (appelée classe POJO) dont chacune encapsule les données d'une occurrence de table.
- Des fichiers de mapping dont chacun assure la correspondance entre une classe et une table (mapping XML ou annotation).
- Des propriétés de configuration notamment des informations concernant la connexion à la base de données.



Déploiement Hibernate

Pour déployer le Framework, Il faut ajouter au projet les librairies nécessaires(site officiel http://www.hibernate.org) .

version 3
(min lib)

- antlr-2.7.6.jar (outil de traitement de texte structuré)
- asm-attrs.jar (générateur de byte code , requis avec cglib)
- asm.jar (générateur de byte code, requis avec cglib)
- cglib-2.1.3.jar (génération de code pour étendre les classes à l'exécution)
- commons-collections-3.1.jar (collections Commons du projet Jakarta)
- dom4j.jar (parseur de configuration XML et de mapping)
- ehcache-1.2.3.jar (cache de niveau JVM)
- hibernate3.jar
- hibernate-annotations.jar
- hibernate-core.jar
- javassist-3.9.0.GA.jar (générateur de byte code Javaassist)
- jta.jar (Java Transaction API)
- jta-1.1.jar (implémentation de référence de l'API Sun JTA)
- Log4j.jar (outil de journalisation)
- slf4j.api-1.6.1.jar (couche d'abstraction pour les API de journalisation)
- slf4j-simple-1.6.1.jar (complémentaire à slf4j.api)

Fichier XML de mapping

- Différents éléments sont précisés dans le fichier de mapping XML :
  - La classe qui va encapsuler les données
  - L'identifiant dans la base de données et son mode de génération
  - Le mapping entre les propriétés (attributs) de classe et les champs de la table de la base de données
  - Les relations
  - ...
- Le tag racine du document XML est le tag <hibernate-mapping>.
- Le nom du fichier de mapping doit être formé par le nom de la classe suivi par ".hbm.xml". (ce fichier doit être situé dans le même répertoire que la classe correspondante).

Exemple de projet

Saisir_Developpeur.JSP

Lister_Developpeur.JSP

Exemple de mapping

Modèle Objet

Developpeur
- Identifiant :int - pseudo, mail: String - codePostal: int - dateInscription : Date;
+ Developpeur () + getXxx(): …; + setXxx(…) : void

```
import java.io.Serializable;
public class Developpeur
            implements Serializable {
    private int identifiant;
    private String pseudo;
    private String mail;
    private int codePostal;
    private java.util.Date dateInscription;
    public Developpeur (){ }
    public … getXxx() …
    public void setXxx( … )  …
}
```

Modèle relationnel (MySQL)

```
CREATE TABLE `devloppeurs` (
    `identifiant` int(11)  AUTO_INCREMENT,
    `pseudo` varchar(20) NOT NULL,
    `mail` varchar(30) NOT NULL,
    `codePostal` int(11) ,
     `dateInscription` date,
    PRIMARY KEY (`identifiant`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
                    AUTO_INCREMENT=1 ;
```

identifiant	pseudo	mail	codePostal	dateInscription
1	ENSAM	Ensam@uh2.ac.ma	21000	2017-09-26
2	ENSAB	ensab@uh1.ac.ma	26000	2018-09-01
3	FPK	fpk@uh1.ac.ma	26000	2019-01-01

Fichier de mapping « Developpeur.hbm.xml »

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="Developpeur" table="developpeurs">
       <id name="identifiant" type="int" column="identifiant">
            <generator class="assigned"/>   </id>
       <property name="pseudo" type="string" not-null="true" />
       <property name="mail" type="string" not-null="true" />
       <property name="codePostal" type="int" not-null="true" />
       <property name="dateInscription" type="date">
            <meta attribute="field-description">date d' inscription</meta>
       </property>
    </class>
</hibernate-mapping>
```

Nom des classes de génération fournies en standard par Hibernate:

increment	incrémentation d'une valeur dans la JVM
identity	utilisation d'un identifiant auto-incrémenté (supporté par DB2, MySQL,SQL Server, ...)
sequence	utilisation d'une séquence (supporté par Oracle, DB2, PostgreSQL, ...)
native	utilise la meilleure solution proposée par la base de données
assigned	la valeur est fournie par l'application

Les propriétés de configuration

- L'exécution d'Hibernate, nécessite la définition d'un certain nombre de propriétés de configuration relatives à la connexion de la BD.
- Ces propriétés peuvent être fournies sous plusieurs formes :
  - un fichier de configuration nommé hibernate.properties (stocké dans un répertoire inclus dans le classpath: scr par exemple)
  - un fichier de configuration au format XML nommé hibernate.cfg.xml
  - utiliser la méthode setProperties() de la classe Configuration

Fichier de configuration « hibernate.cfg.xml »

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
 <!-- Paramètres de connexion à la base de données -->
 <property
  name="connection.driver_class">com.mysql.jdbc.Driver</property>
 <property
  name="connection.url">jdbc:mysql://localhost:3306/DBName</property>
 <property name="connection.username">root</property>
 <property name="connection.password"></property>
 <!-- Pool de connexion JDBC (nb max de connexions dans le pool)-->
 <property name="connection.pool_size">1</property>
 <!-- le dialect SQL : dialogue BD  -->
 <property
  name="dialect">org.hibernate.dialect.MySQLDialect</property>
```


Fichier de configuration « hibernate.cfg.xml »

```
 <!-- Activer la gestion automatique du contexte de session -->
 <property name="current_session_context_class">thread</property>
 <!-- Désactiver le cache de second niveau -->
 <property
   name="cache.provider_class">org.hibernate.cache.NoCacheProvider
 </property>
 <!-- Écrit les ordres SQL dans la console  -->
 <property name="show_sql">true</property>
 <!-- Export automatiquement du schéma DDL vers la BD ( option :
     Validate, update, create, create-drop          -->
 <property name="hbm2ddl.auto"> update </property>
 <!-- resources de mapping -->
 <mapping resource="package/PojoClass.hbm.xml"/>
 </session-factory>
</hibernate-configuration>
```

Utilisation des services Hibernate

- Pour utiliser les services d'Hibernate, il est nécessaire de créer une instance de la classe Session.
- Une instance Session est obtenue à partir d'une fabrique de type SessionFactory (entrepôt de données unique: singleton), elle-même obtenue à partir de l'instance de type Configuration en utilisant la méthode « buildSessionFactory() »

```
SessionFactory sessionFactory = new Configuration().configure
                        ("Hibernate.cfg.xml").buildSessionFactory();
Session session = sessionFactory.openSession();
```

- En fin d'utilisation des services d'Hibernate, il faut clôturer l'objet Session avec la méthode close().

Classe Utilitaire « HibernateUtil.java »

```
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
  public static final SessionFactory sessionFactory;
    static {
        try {
            // Création de la SessionFactory à partir de hibernate.cfg.xml
            sessionFactory = new Configuration().configure
                                        ("Hibernate.cfg.xml").buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {  return sessionFactory;    }
}
```

La persistance d'une nouvelle occurrence

Pour créer une nouvelle occurrence dans la source de données, il faut:

- Créer une nouvelle instance de la classe encapsulant les données et valoriser ses propriétés
- Appeler la méthode « **save** » (ou l'une des méthodes « **update** » et « **saveOrUpdate** » utilisée en fonction de la valeur de l'identifiant dans la classe) de la session en lui passant en paramètre instance crée.
- Réaliser un **commit** sur la connexion ou la transaction ou faire appel de la méthode « **flush** » de la classe Session pour enregistrer les données dans la base de données.

Exemple de persistance d'une nouvelle occurrence


```
Developpeur developpeur= new Developpeur();
developpeur.setIdentifiant(
   Integer.parseInt(request.getParameter("identifiant")) );
developpeur.setPseudo( request.getParameter("pseudo") );
developpeur.setMail( request.getParameter("mail") );
developpeur.setCodePostal(
   Integer.parseInt(request.getParameter("codePostal")) );
Session sessionHiber  =
   HibernateUtil.getSessionFactory().getCurrentSession();
sessionHiber.beginTransaction();
sessionHiber.save(developpeur);
sessionHiber.getTransaction().commit();
```

Configurer le système de logs

- Le système de logs permet au développeur d'afficher sur une sortie les messages fournis par Hibernate lors de son exécution (rendre transparente la couche logicielle de Hibernate)
- Pour configurer le système de logs, il faut copiez « log4j.properties » de la distribution d'Hibernate (voir le répertoire etc/) dans le répertoire src.
- Le fichier « log4j.properties» contient une configuration du système de logs qui peut être personnalisée par l'utilisateur. Hibernate utilise commons-logging et vous laisse le choix entre log4j et le système de logs du JDK 1.4

Obtention des données

Hibernate utilise plusieurs moyens pour obtenir des données de la base de données :

- Hibernate Query Language (HQL) qui possède une syntaxe similaire à SQL
- API Criteria (Query By Criteria "QBC") qui permet de définir une requête à partir de critères pour retrouver des données.
- Requêtes SQL natives

Le langage de requête HQL

- HQL est un langage de requêtes orienté objets :
  - La syntaxe de HQL est similaire à SQL.
  - Les entités utilisées sont des objets et des propriétés (aucune référence aux tables ou aux champs)
- Une requête HQL n'est pas sensible à la casse (à l'exception des noms de classes Java et des propriétés), elle peut être composée de:
  - Clauses
  - Fonctions d'agrégation
  - Sous requêtes
- HQL est indépendant de la base de données (Hibernate se charge de générer la requête SQL à partir de la requête HQL en tenant compte du contexte : paramètres défini dans le fichier de configuration et les fichiers du mapping).

Mise en œuvre de HQL

Une requête HQL peut être exécutée avec une instance de la classe Query en invoquant la méthode createQuery() de la session Hibernate.

```
Query query = session.createQuery("requête HQL");
List result = query.list();
```

Exemple de requête HQL :

- from Devloppeur as dev
- select dev.mail from Devloppeur as dev
- from Devloppeur as dev where dev. pseudo = "ENSA"
- select dev.mail, dev.pseudo from Devloppeur as dev order by dev.pseudo asc, dev.mail desc
- select count(*) from Devloppeur
- from Devloppeur as dev where dev.codePostal = (select min(dev.codePostal) from Devloppeur)

Exemple de mise en œuvre de HQL

Affichage des messages

Identifiant	Pseudo	Mail	CodePostal	DateInscription
1	ENSAB	ensab@uhp.ac.ma	21000	01/01/2020

Ajout messages...

```
public static List<Developpeur> getAllDeveloppeurs(){
    try {
        Session sessionHib=HibernateUtil.getSessionFactory().
                                getCurrentSession() ;
        sessionHib.beginTransaction();
        Query req=sessionHib.createQuery("from Developpeur");
        return  req.list();
    }catch (Exception e) {
        System.out.println(e.getMessage());
    }
    return null;
}
```


Mise en œuvre de HQL

Avec HQL, Il est possible de définir des requêtes paramétrées :

- Les paramètres sont précisés avec un caractère « : » suivi d'un nom unique).
- Des méthodes setXXX(nom ou index, valeur) permettent d'associer à chaque paramètre une valeur (XXX représente le type du paramètre).

Exemple

```
String ps="ENSA";
session.beginTransaction();
Query req=session.createQuery ("from Developpeur where pseudo like :p");
req.setString("p", "%"+ps+"%");
```

L'API Criteria

- L'API Criteria Query propose des objets pour définir les critères d'une requête(L'API Criteria utilise HQL en sous-jacent).
- L'API Criteria propose des classes et des interfaces qui encapsulent les fonctionnalités SQL dont les principales sont :

  Criteria, Criterion, Restrictions, Projection, Order

  - Le type Criterion encapsule un élément de la clause "where" de la requête SQL.
  - La classe Restrictions est une fabrique qui propose des méthodes statiques pour créer des instances de type Criterion
  - L'interface Projection encapsule un champ de la clause "select" de la requête SQL. La classe Projections est une fabrique pour les instances de type Projection.

L'API Criteria

- Les opérateurs de comparaison sont encapsulés dans des méthodes de la classe Restrictions : eq(), lt(), le(), gt(), ge().
- La classe Restrictions propose aussi des méthodes pour les opérateurs SQL : like, between, in, is null, is not null ...
- Chaque instance d'un critère (interface Criterion) doit être ajoutée aux critères de la requête en utilisant la méthodes add() de l'instance de type Criteria

Exemple:

```
Criteria criteria = session.createCriteria(Developpeur.class);
Criterion critere = Restrictions.like("pseudo", "ENSA%");  criteria.add(critere);
List< Developpeur>  developpeurs = criteria.list();
```

Mapper les relations avec Hibernate

- Hibernate propose de transcrire les relations du modèle relationnel dans le modèle objet. Il supporte plusieurs types de relations :
  - relation de type 1 - 1 (one-to-one)
  - relation de type 1 - n (one-to-many)
  - relation de type n - n (many-to-many)
- Il est ainsi, nécessaire de définir dans le fichier de mapping, les relations entre la table concernée et les tables avec lesquelles elle possède des relations (les relations peuvent aussi être définies avec des annotations).

La relation de type 1 - 1

Il y a plusieurs façons de traiter une relation « one – to - one » (selon l'utilisation d'une ou deux tables dans la base de données) :

- Une seule table avec un Component d'Hibernate
- Deux tables (clé primaire partagée ou une clé étrangère avec contraintes) et une relation One-to-One d'Hibernate

La relation un / un (une seule table)

Modèle Objet

Personne

- Id : long
- nom : String
- prenom: String
- dateNais :Date
- adresse : Adresse

+ Personne ()
+ getXxx(): … ; + setXxx(…) : …

Adresse

- ligne1 :String
- ligne2 : String
- Cp : String
- Ville : String
- ligne3 : String

+ Adresse()
+ getXxx(): … ; + setXxx(…) : …

Modèle relationnel

```
CREATE TABLE `personne` (
    `id` bigint(20) NOT NULL,
    `Nom` varchar(30) NOT NULL,
    `Prenom` varchar(20) ,
     `DateNais` datetime,
     `ligne1_adr` varchar(255),
     `ligne2_adr` varchar(255) DEFAULT NULL,
     `cp_adr` varchar(255),
     `ville_adr` varchar(255),
      `ligne3_adr` varchar(255),
    PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```


La relation un / un (une seule table)

Fichier de mapping de l'entité Personne (Personne.hbm.xml)

```
<hibernate-mapping>
 <class name="metier.Personne" table="personne">
    <id name="id" column="id"> <generator class="increment"/> </id>
    <property name="nom" column="Nom" /> <property name="prenom" column="Prenom"/>
    <property name="dateNais" column="DateNais"/>
    <component name="adresse" class="metier.Adresse">
       <property name="ligne1" /> <property name="ligne2" />
       <property name="cp" column="cp_adr" /> <property name="ville" column="ville_adr" />
       <property name="ligne3" column="ligne3_adr"/>
    </component>
 </class>
</hibernate-mapping>
```

Fichier de configuration d'Hibernate

```
<hibernate-configuration>  <session-factory>
   …
   <mapping resource="Personne.hbm.xml"></mapping>
</session-factory> </hibernate-configuration>
```

La relation un / un (une seule table)

Exemple de mise en œuvre

```
public static boolean addPersonne(Personne personne)  {
    Session session=null;
    try {
       session=HibernateUtil.getSessionFactory().openSession();
       session.beginTransaction();
       session.save(personne);
       session.getTransaction().commit();
       } catch (Exception e) { System.out.println(e.getMessage());
       } finally { if (session!=null) session.close();   }
       return true;
}
```

Table "personne"

id	Nom	Prenom	DateNais	ligne1_adr	ligne2_adr	cp_adr	ville_adr	ligne3_adr
1	Ensa	Berrechid	2018-09-01 18:44:58	ligne1_1	ligne2_1	cp_1	ville1	ligne3_1

La relations un / un (deux tables avec clé primaire partagée)

Modèle Objet

Personne2

- Id : long
- nom : String
- prenom: String
- dateNaid :Date
- adresse : Adresse2

+ Personne2 ()
+ getXxx(): … ; + setXxx(…) : …

Adresse2

- Id : long
- ligne1 :String
- ligne2 : String
- Cp : String
- Ville : String
- ligne3 : String
- personne : Personne2

+ Adresse2()
+ getXxx(): … ; + setXxx(…) : …

Modèle relationnel

```
CREATE TABLE `personne2` (
   `Id` bigint(20) NOT NULL auto_increment,
   `Nom` varchar(255) NOT NULL '',
   `Prenom` varchar(255) NOT NULL '',
   `DateNais` date default NULL, PRIMARY KEY (`Id`)
)  ENGINE=InnoDB DEFAULT CHARSET=latin1
          AUTO_INCREMENT=0 ;
CREATE TABLE `adresse2` (
   `id` bigint(20) NOT NULL default '0',
   `ligne1_adr` varchar(80) NOT NULL '',
   `ligne2_adr` varchar(80) default NULL,
   `cp_adr` varchar(5) default NULL,
   `ville_adr` varchar(80) default NULL,
   `ligne3_adr` varchar(80), PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

ALTER TABLE `adresse2` ADD CONSTRAINT `FK…`
FOREIGN KEY (`Id`) REFERENCES `personne2` (`id`);
```

La valeur des clés primaires est partagée entre les deux tables. L'identifiant de la table adresse correspond à la valeur de l'identifiant de la table personne.

La relations un / un (deux tables avec clé primaire partagée)

Fichier de mapping de l'entité Personne2 (Personne2.hbm.xml)

```
<hibernate-mapping>
  <class name="metier.Personne2" table="Personne2">
    <id name="id" column="id"> <generator class="increment" /> </id>
    <property name="nom" column="Nom" /> <property name="prenom" column="Prenom" />
    <property name="dateNais" column="DateNais" />
    <one-to-one name="adresse" class="metier.Adresse2“ cascade="save-update" />
  </class> </hibernate-mapping>
```

Fichier de mapping de l'entité Adresse2 (Adresse.hbm.xml)

```
<hibernate-mapping>
  <class name="metier. Adresse2" table="Adresse2">
   <id name="id" column="Id"> <generator class="foreign">  <param name="property">personne</param>
   </generator> </id>
   <property name="ligne1" column="ligne1_adr" /> <property name="ligne2" column="ligne2_adr" />
   <property name="cp" column="cp_adr" /> <property name="ville" column="ville_adr" />
   <property name="ligne3" column="ligne3_adr" />
   <one-to-one name="personne" class="métier.Personne2" constrained="true" />
  </class> </hibernate-mapping>
```

- Le champ identifiant id est défini avec un générateur de type foreign avec un paramètre qui précise que la valeur sera celle de l'identifiant de personne
- La relation inverse avec Personne est définie avec un tag <one-to-one> avec l'attribut constrained ayant la valeur true.

La relations un / un (deux tables avec clé primaire partagée)

Fichier de configuration d'Hibernate

```
<hibernate-configuration>
   …
  <mapping resource="Personne2.hbm.xml"></mapping>
  <mapping resource="Adresse2.hbm.xml"></mapping>
 </session-factory>
</hibernate-configuration>
```

Exemple de mise en œuvre

```
public static boolean addPersonne2(Personne2 personne)  {
  Session session=null;
  try {
    session=HibernateUtil.getSessionFactory().openSession(); session.beginTransaction();
    personne.getAdresse().setPersonne(personne);//non prise charge automatique des liens bidirectionnels
    session.save(personne); session.getTransaction().commit();
  } catch (Exception e) {    System.out.println(e.getMessage());
  } finally { if (session!=null) session.close(); }
  return true;
}
```

Table "personne2"

id	Nom	Prenom	DateNais
1	Ensa	Berrechid	2018-09-01

Table "adresse2"

Id	ligne1_adr	ligne2_adr	cp_adr	ville_adr	ligne3_adr
1	ligne1_1	ligne2_1	cp_1	ville1	ligne3_1

La relation un / un (2 tables avec clé étrangère)

Modèle Objet

Personne3

- Id : long
- nom : String
- prenom: String
- dateNais :Date
- adresse : Adresse3

+ Personne3 ()
+ getXxx(): … ; + setXxx(…) : …

Adresse3

- Id : long
- ligne1 :String
- ligne2 : String
- Cp : String
- Ville : String
- ligne3 : String

+ Adresse3()
+ getXxx(): … ; + setXxx(…) : …

Modèle relationnel

```
CREATE TABLE `personne3` (
   `Id` bigint(20) NOT NULL auto_increment,
   `Nom` varchar(255) NOT NULL '',
   `Prenom` varchar(255) NOT NULL '',
   `DateNais` date default NULL,
   `adresse_id` bigint(20) NOT NULL  Unique default '0',
   PRIMARY KEY (`Id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
            AUTO_INCREMENT=0 ;
CREATE TABLE `adresse3` (
   `adresse_id` bigint(20) NOT NULL auto_increment,
   `ligne1_adr` varchar(80) NOT NULL '',
   `ligne2_adr` varchar(80), `cp_adr` varchar(5) ,
   `ville_adr` varchar(80) , `ligne3_adr` varchar(80),
   PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
            AUTO_INCREMENT=0 ;
ALTER TABLE `personne3` ADD CONSTRAINT `FK…`
     FOREIGN KEY (`adresse_id`) REFERENCES
      `adresse3` (`adresse_id`);
```

La relation entre les 2 tables est assurée par une clé étrangère de la table personne vers la table adresse

La relation un / un (2 tables avec clé étrangère)

Fichier de mapping de l'entité Personne (Personne.hbm.xml)

```
<hibernate-mapping>
 <class name="metier.Personne3" table="Personne3">
   <id name="id" column="Id"> <generator class="increment" /></id>
   <property name="nom" column="Nom" /> <property name="prenom" column="Prenom" />
   <property name="dateNais" column="DateNais" />
   <many-to-one name="adresse" class="métier.Adresse3“ column="adresse_id" cascade="all"
                unique="true" />
</class> </hibernate-mapping>
```

L'unicité de la relation (tag <many-to-one>) est garantie par la valeur true de l'attribut unique. La propriété column précise la colonne associée à la clé étrangère vers la table adresse.

Fichier de mapping de l'entité Adresse (Adresse.hbm.xml)

```
<hibernate-mapping>
   <class name="metier.Adresse3"  table="Adresse3">
      <id name="id"  column="adresse_id" > <generator class="increment" /> </id>
      <property name="ligne1" column="ligne1_adr" /> <property name="ligne2" column="ligne2_adr" />
      <property name="cp" column="cp_adr" /> <property name="ville" column="ville_adr" />
      <property name="ligne3" column="ligne3_adr" />
   </class> </hibernate-mapping>
```

La relation un / un (2 tables avec clé étrangère)

Fichier de configuration d'Hibernate

```
<hibernate-configuration>
  <session-factory>
    …
    <mapping resource="metier/Personne3.hbm.xml"></mapping>
    <mapping resource="metier/Adresse3.hbm.xml"></mapping>
  </session-factory>
</hibernate-configuration>
```

Exemple de mise en œuvre

```
public static boolean addPersonne(Personne3 personne)  {
     Session session=null; Transaction = null; try {
     session=HibernateUtil.getSessionFactory().openSession();
     transaction = session.beginTransaction();   session.save(personne);   transaction .commit();
   } catch (Exception e) {
   System.out.println(e.getMessage());
   } finally { if (session!=null) session.close();    }
   return true;
   }
```

Table "personne3"

id	Nom	Prenom	DateNais	adresse_id
1	Ensa	Berrechid	2019-04-27	1

Table "auteur3"

adresse_id	ligne1_adr	ligne2_adr	cp_adr	ville_adr	ligne3_adr
1	ligne1_1	ligne2_1	cp_1	ville1	ligne3_1

La relation many to many bidirectionnelle

Modèle Objet

Personne

- Id : long
- nom : String
- prenom: String
- dateNaid :Date
- adresse : Adresse
- reunions : Set<Reunion>

+ Personne ()
+ getXxx(): … ; + setXxx(…) : …

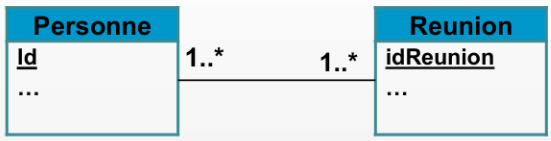
Reunion

- idReunion :long ;
- dateReunion : Date;
- Titre : String;
- personnes: Set<Personne>

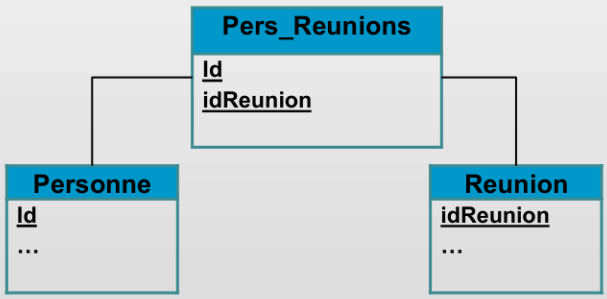
+ Reunion ()
+ getXxx(): … ; + setXxx(…) : …

- La collection Set «reunions» regroupe les réunions d'une personne.
- La collection Set «personnes» définit les personnes d'une reunion.

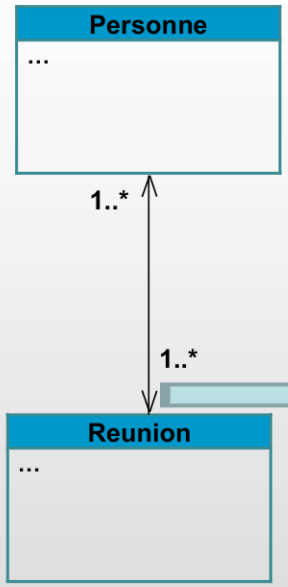
Modèle relationnel



Association plusieurs à plusieurs non porteuse de propriété.



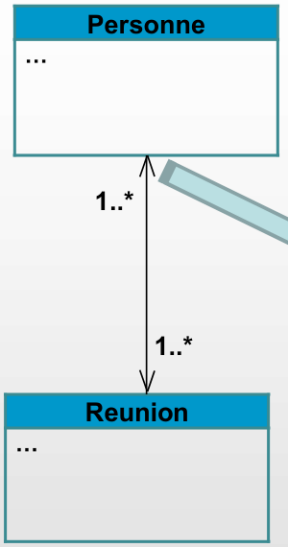
La relation many to many bidirectionnelle



Fichier de mapping de l'entité Personne (Personne.hbm.xml)

```
<hibernate-mapping>
  <class name="metier.Personne" table="personne">
    <id name="id" column="id">
        <generator class="increment"/> </id>
    <property name="nom" column="Nom" />
    <property name="prenom" column="Prenom"/>
    <property name="dateNais" column="DateNais"/>
    <set name="reunions"  table="Pers_Reunions" inverse="true">
        <key column="Id_Personne"/>
        <many-to-many column="Id_Reunion"
                      class="metier.Reunion"/>
    </set>
  </class>
<hibernate-mapping>
```


La relation many to many bidirectionnelle



Fichier de mapping de l'entité Reunion (Reunion .hbm.xml)

```
<hibernate-mapping>
  <class name="metier.Reunion" table="reunion">
     <id name="idReunion" column="idReunion">
         <generator class="increment"/> </id>
     <property  name="dateReunion"  column="dateReunion"  />
     <property name="titre" column="titre"/>
     <set name= "personnes" table="Pers_Reunions" >
         <key column="Id_Reunion"/>
         <many-to-many column="Id_Personne“
                       class="metier.Personne"/>
     </set>
</class> <hibernate-mapping>
```

Fichier de configuration d'Hibernate

```
<hibernate-configuration>  <session-factory>
  …
  <mapping resource="metier/Personne.hbm.xml"></mapping>
  <mapping resource="metier/Reunion.hbm.xml"></mapping>
</session-factory> </hibernate-configuration>
```

La relation many to many unidirectionnelle

Exemple de mise en œuvre

```
public static boolean addReunionToPersonne(long idPers, long
      idReunion){
 try {
  Session
  session=HibernateUtil.getSessionFactory().getCurrentSession();
  session.beginTransaction();
  // Charger la personne et la réunion
  personne = (Personne) session.load(Personne.class, idPers);
  reunion = (Reunion) session.load(Reunion.class, idReunion);
  reunion.getPersonnes().add(personne);
  session.getTransaction().commit();
 } catch (Exception e) {  System.out.println(e.getMessage());  }
 return true;
}
```

Nom	Prenom	DateNais
ENSA	FI_1	2019-04-28 15:08:32
ENSA	FI_2	2019-04-28 15:08:33
ENSA	FI_3	2019-04-28 15:08:33
ENSA	FI_4	2019-04-28 15:08:33
ENSA	FI_5	2019-04-28 15:08:33

idReunion	dateReunion	titre
1	2019-04-28 15:08:33	ENSA_Reunion1
2	2019-04-28 15:08:33	ENSA_Reunion2
3	2019-04-28 15:08:33	ENSA_Reunion3
4	2019-04-28 15:08:33	ENSA_Reunion4
5	2019-04-28 15:08:33	ENSA_Reunion5

Id_Personne	Id_Reunion
1	1
1	5
2	2
2	4
3	3
4	2
4	4
5	1
5	5

- Hibernate détecte automatiquement que la collection a été modifiée et a besoin d'être mise à jour: vérification sale automatique ("automatic dirty checking")
- Dans le cas « many-to-many » Une BDR est plus flexible qu'un langage de programmation orienté objet puisque qu' elle n'a pas besoin de direction de navigation

La relation bidirectionnelle de type one-to-many et many-to-one

Modèle Objet

Personne

- Id : long
- nom : String
- prenom: String
- dateNaid :Date
- adresse : Adresse
- dept : Departement

+ Personne ()
+ getXxx(): … ; + setXxx(…) : …

Departement

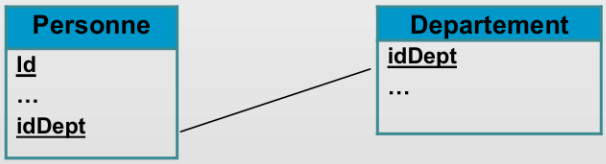
- idDept :long
- intitule : String
- personnes: Set<Personne>

+ Departement()
+ getXxx(): … ; + setXxx(…) : …

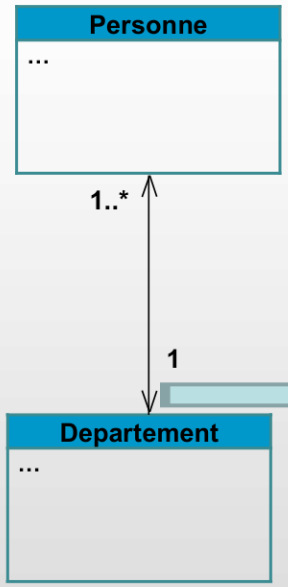
Collection d'objets Personne dans la classe Departement.
La création d'une référence à un objet Departement dans la classe Personne.

Modèle relationnel

Association un à plusieurs



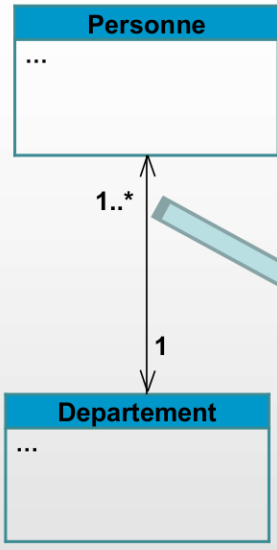
La relation bidirectionnelle de type one-to-many et many-to-one



Fichier de mapping de l'entité Personne (Personne.hbm.xml)

```
<hibernate-mapping>
  <class name="metier.Personne" table="personne">
    <id name="id" column="id">
        <generator class="increment"/> </id>
    <property name="nom" column="Nom" />
    <property name="prenom" column="Prenom"/>
    <property name="dateNais" column="DateNais"/>
    <many-to-one name="dept" column="IdDept"/>
  </class>
<hibernate-mapping>
```


La relation bidirectionnelle de type one-to-many et many-to-one



Fichier de mapping de l'entité Departement(Departement.hbm.xml)

```
<hibernate-mapping>
  <class name="metier.Departement" table="Departement">
    <id name="idDept" column=" idDept ">
        <generator class="increment"/> </id>
    <property name="intitule" column=" intitule" />
    <set name="personnes">
        <key column="IdDept"/>
        <one-to-many class="metier.Personne"/>
    </set>
  </class>
<hibernate-mapping>
```

Fichier de configuration d'Hibernate

```
<hibernate-configuration>  <session-factory>
  …
  <mapping resource="metier/Personne.hbm.xml"></mapping>
  <mapping resource="metier/Departement.hbm.xml"></mapping>
</session-factory> </hibernate-configuration>
```

La relation bidirectionnelle de type one-to-many et many-to-one

Exemple de mise en œuvre

```
public static boolean addDepartement(Departement departement)  {
  try {
    Session
      session=HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    session.save(departement);
    session.getTransaction().commit();
  } catch (Exception e) {   System.out.println(e.getMessage());    }
  return true;
}
public static Departement getDepartement (long idDept) {
  try {
    Session
       session=HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
    return ((Departement) session.load(Departement.class, idDept));
  } catch (Exception e) {   System.out.println(e.getMessage());   }
  return null;
}
```

idDept	intitule
1	Dept_1
2	Dept_2
3	Dept_3

id	Nom	Prenom	DateNais	IdDept
1	ENSA	FI_1	2019-05-03	1
2	ENSA	FI_2	2019-05-03	2
3	ENSA	FI_3	2019-05-03	2
4	ENSA	FI_4	2019-05-03	3
5	ENSA	FI_5	2019-05-03	3

Le mapping de l'héritage de classes

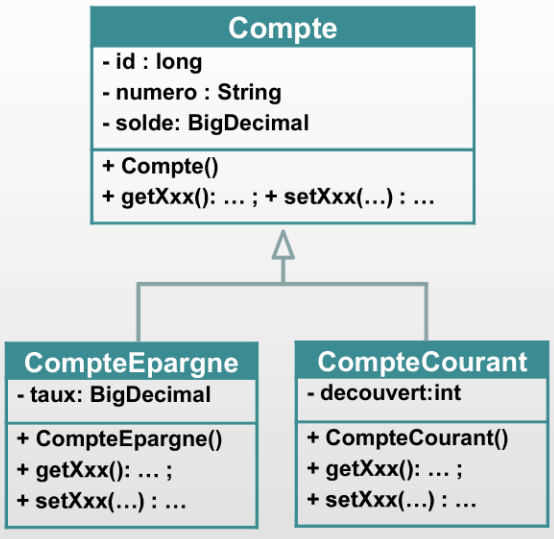
Hibernate propose trois stratégies de base pour le mapping de l'héritage des classes :

- une table par hiérarchie de classes (Table Per Hierarchy) qui possède une colonne supplémentaire qui sert de discriminant en précisant le type des données de la ligne
- une table par classe concrète (Table Per Concrete class). Les champs communs sont dupliqués dans chaque table fille.
- une table par sous-classe (Table Per Subclass) : à chaque classe correspond une table. Les relations entre ces tables se font en utilisant des relations par clés étrangères.

Le mapping de l'héritage (une table par hiérarchie de classe)

- Pour définir le mapping d'un héritage utilisant le modèle d'une table par hiérarchie de classe, il faut utiliser dans un même fichier de mapping hbm:
  - un tag <class> pour le mapping classe mère
  - un tag <subclass> pour le mapping de chaque classe fille
- Le tag <class> possède l'attribut polymorphism qui peut prendre deux valeurs (implicit ou explicit) et permet de préciser le type de requête polymorphique.
- L'attribut abstract permet de préciser si la classe mère de la hiérarchie est abstraite. Les valeurs possibles sont true et false
- Le tag <discriminator> permet de préciser la colonne qui servira de discriminant (cette colonne , utilisée par Hibernate et la base de données, ne doit pas être définie dans la classe Java correspondante).
- Les tags <class> et <subclass> possèdent un attribut discriminator-value qui permet de préciser la valeur de la colonne discriminante.

Le mapping de l'héritage de classes




```
package metier;
import java.math.BigDecimal;
import java.io.Serializable;
public class Compte  implements Serializable {
    protected int id;
    protected String numero;
    protected BigDecimal solde;
    Compte() { }
    // getter , setter toString
}
public  class  CompteCourant  extends  Compte
          implements Serializable {
    protected int decouvert;
    CompteCourant(){ Super(); }
     // getter , setter toString
}
public  class  CompteEpargne  extends  Compte
          implements Serializable {
    protected BigDecimal taux;
    CompteEpargne(){ Super(); }
    // getter , setter toString
}
```

Le mapping de l'héritage (une table par hiérarchie)

Fichier de configuration d'Hibernate

```
<hibernate-configuration>  <session-factory>  …
  <mapping resource="Compte.hbm.xml"/>  </session-factory> </hibernate-configuration>
```

Fichier de mapping « Compte.hbm.xml »

```
<hibernate-mapping>
 <class name="metier.Compte" table="compte"
          discriminator-value="Compte">
   <id name="id" column="id" type="int" > <generator class="native" /> </id>
   <discriminator column="DTYPE" type="string" />
   <property name="numero" column="numero" type="string" />
   <property name="solde" column="solde" type="big_decimal" />
   <subclass name="metier.CompteCourant" discriminator-value="CompteCourant">
        <property name="decouvert" column="decouvert" type="int"/>
   </subclass>
   <subclass name="metier.CompteEpargne" discriminator-value="CompteEpargne">
        <property name="taux" column="taux" type="big_decimal" />   </subclass>
</class> </hibernate-mapping>
```

Nom	Type
id	int(11)
DTYPE	varchar(255)
numero	varchar(255)
solde	decimal(19,2)
decouvert	int(11)
taux	decimal(19,2)

Exemple de code de mise en œuvre

```
Compte compte = new Compte(…); save (compte) ;
save (new CompteCourant (…) );
compte= new CompteEpargne (..); save(compte)
```

id	DTYPE	numero	solde	decouvert	taux
1	Compte	numero-1	1000.00	*NULL*	*NULL*
2	CompteCourant	numero-1	1000.00	100	*NULL*
3	CompteCourant	numero-1	1000.00	200	*NULL*
4	CompteEpargne	numero-1	1000.00	*NULL*	0.10

Le mapping de l'héritage (une table par classe concrète)

- Chaque classe concrète est mappée sur une table contenant toutes les propriétés de la classe mère.
- Une classe mère est définie grâce à un tag <class>.
- Une classe fille est définie grâce à un tag fils <union-subclass>. Il possède plusieurs attributs :
  - name : nom pleinement qualifié de la classe
  - table : nom de la table dans la base de données
- Chaque propriété propre à une classe fille doit être mappée dans le tag <union-subclass> correspondant.

Le mapping de l'héritage (une table par classe concrète)

Fichier de configuration d'Hibernate

```
<hibernate-configuration>  <session-factory>  …
  <mapping resource="Compte.hbm.xml"/>  </session-factory> </hibernate-configuration>
```

Fichier de mapping « Compte.hbm.xml »

```
<hibernate-mapping> <class name="metier.Compte" table="compte" >
    <id name="id" column="id" type="int"> <generator class="native" /> </id>
    <property name="numero" column="numero" type="string" />
    <property name="solde" column="solde" type="big_decimal" />
    <union-subclass name="metier.CompteCourant" table="compte_courant">
        <property name="decouvert" column="decouvert" type="int"/>
    </union-subclass>
    <union-subclass name="metier.CompteEpargne" table="compte_epargne">
        <property name="taux" column="taux" type="big_decimal"/>
    </union-subclass>
</class> </hibernate-mapping>
```

Exemple de code de mise en œuvre

```
Compte compte = new Compte(…);
save (compte) ;
save (new CompteCourant (…) );
compte= new CompteEpargne (..);
save(compte);
```

id	numero	solde
1	numero-1	1000.00

id	numero	solde	decouvert
2	numero-1	1000.00	100

id	numero	solde	taux
3	numero-1	1000.00	0.10

Le mapping de l'héritage (une table par sous-classe)

- Les données héritées sont placées sur la table de la classe mère alors que les données spécifiques sont enregistrées dans la table de la classe fille.
- La clé primaire des tables des classes filles est une clé étrangère vers la table de la superclasse. Hibernate utilise une relation de type 1-1 entre ces deux clés.

- La définition du mapping se fait dans un même fichier « .hbm » :
  - un tag <class> pour le mapping classe mère
  - un tag <joined-subclass> pour le mapping de chaque classe fille
- Le tag <id> du tag <class> permet de préciser la colonne qui est la clé primaire. Le tag <key> fils du tag <joined-subclass> permet de préciser la clé étrangère qui sera utilisée pour réaliser la jointure avec la table mère.

Le mapping de l'héritage (une table par sous-classe)

Fichier de configuration d'Hibernate

```
<hibernate-configuration>  <session-factory>  …
  <mapping resource="Compte.hbm.xml"/>  </session-factory> </hibernate-configuration>
```

Fichier de mapping « Compte.hbm.xml »

```
<hibernate-mapping> <class name="metier.Compte" table="compte" >
    <id name="id" column="id" type="int"> <generator class="native" /> </id>
    <property name="numero" column="numero" type="string" />
    <property name="solde" column="solde" type="big_decimal" />
    <joined-subclass name="metier.CompteCourant" table="compte_courant">
       <key column="id"/> <property name="decouvert"   column="decouvert" type="int"/>
    </joined-subclass>
    <joined-subclass name="metier.CompteEpargne" table="compte_epargne">
       <key column="id"/> <property name="taux" column="taux" type="big_decimal"/>
    </joined-subclass>
</class> </hibernate-mapping>
```

Exemple de code de mise en œuvre

```
Compte compte = new Compte(…);
save (compte) ;
save (new CompteCourant (…) );
compte= new CompteEpargne (..);
save(compte);
```

id	numero	solde
1	numero-1	1000.00
2	numero-1	1000.00
3	numero-1	1000.00

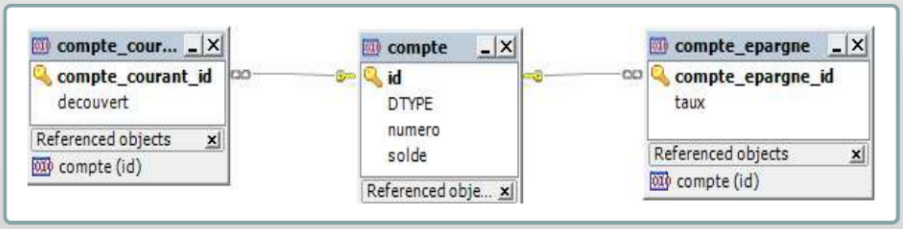
id	decouvert
2	100

id	taux
3	0.10

Pour chaque instance dérivée (fille), l'instance de la superclasse est placée sur la table mere correspondante

Le mapping de l'héritage (une table par sous-classe avec discriminant)

- Cette stratégie requiert la définition d'un champ discriminant dans la classe mère.
- Une classe fille est mappée en utilisant un tag <subclass> avec deux propriétés :
  - name qui précise le nom pleinement qualifié de la classe
  - discriminator_value qui précise la valeur de la colonne discriminator.
  - Le tag fils <key> qui précise le champ qui sera la clé de la table.
- Il faut ajouter au tag <subclass> un tag fils <join> pour définir le mapping de la table avec les propriétés :
  - table permet de préciser le nom de la table.
  - La déclaration optionnelle fetch="select" qui indique à Hibernate de ne pas récupérer les données de la classe fille par une jointure externe lors des requêtes sur la classe mère.



Le mapping de l'héritage (une table par sous-classe avec discriminant)

Fichier de configuration d'Hibernate

```
<hibernate-configuration>  <session-factory>  …
  <mapping resource="Compte.hbm.xml"/>  </session-factory> </hibernate-configuration>
```

Fichier de mapping « Compte.hbm.xml »

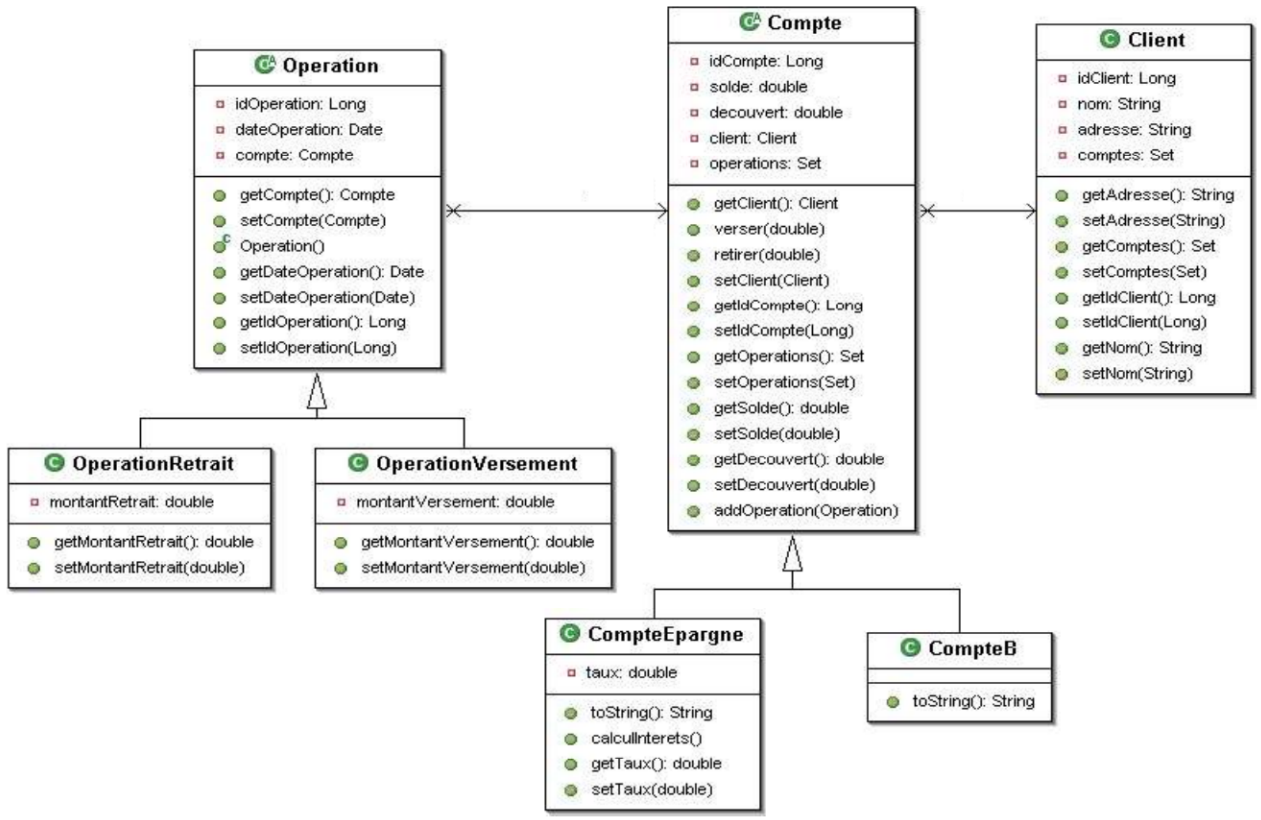
```
<hibernate-mapping>
  <class name="metier.Compte" table="compte" discriminator-value="Compte" >
    <id name="id" column="id" type="int" > <generator class="native" /> </id>
    <discriminator column="DTYPE" type="string" />
    <property name="numero" column="numero" type="string" />
    <property name="solde" column="solde" type="big_decimal" />
    <subclass name="metier.CompteCourant" discriminator-value="CompteCourant">
      <join table="compte_courant" fetch="select" > <key column="compte_courant_id" />
      <property name="decouvert" column="decouvert" type="int"/> </join> </subclass>
    <subclass name="metier.CompteEpargne" discriminator-value="CompteEpargne">
      <join table="compte_epargne"> <key column="compte_epargne_id" />
      <property name="taux" column="taux" type="big_decimal"/> </join> </subclass>
</class> </hibernate-mapping>
```

id	DTYPE	numero	solde
1	Compte	numero-1	1000.00
2	CompteCourant	numero-1	1000.00
3	CompteEpargne	numero-1	1000.00

compte_courant_id	decouvert
2	100

compte_epargne_id	taux
3	0.10

Exemple de mapping



Exemple de mapping

Client.hbm.xml

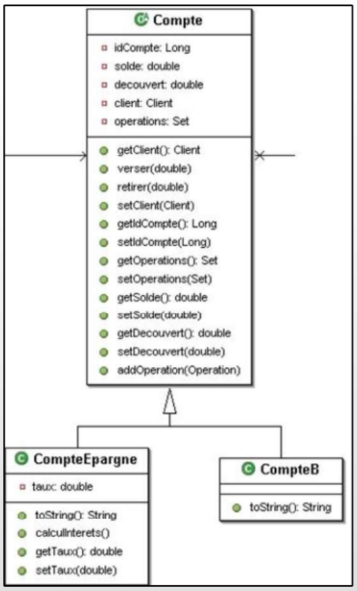
```
<hibernate-mapping>
    <class name="metier.Client" table="CLIENTS">
      <id name="idClient" column="ID_CLIENT">
         <generator class="increment"/> </id>
      <property name="nom" column="NOM"/>
      <property name="adresse" column="ADRESSE"/>
      <set name="comptes" inverse="true">
           <key column="ID_CLIENT"/>
           <one-to-many class="metier.Compte"/>
      </set>
    </class>
</hibernate-mapping>
```

ID_CLIENT	NOM	ADRESSE
1	ENSAB	Berrechid
2	FST	Settat

Exemple de mapping

Compte.hbm.xml

```
<hibernate-mapping>
  <class name="metier.Compte" table="COMPTES"
                 discriminator-value="Compte">
    <id name="idCompte" column="ID_COMPTE">
       <generator class="increment"/> </id>
    <discriminator column="COMPTE_TYPE" type="string"/>
    <property name="solde"/>  <property name="decouvert"/>
    <many-to-one name="client" column="ID_CLIENT"/>
    <set name="operations" inverse="true" table="OPERATIONS" >
       <key column="ID_COMPTE" update="true"/>
       <one-to-many class="metier.Operation"/>  </set>
    <subclass name="metier.CompteB"
            discriminator-value ="CompteB"> </subclass>
    <subclass name="metier.CompteEpargne"
            discriminator-value="CompteEpargne">
       <property name="taux" column="TAUX"/>
    </subclass>
</class> </hibernate-mapping>
```

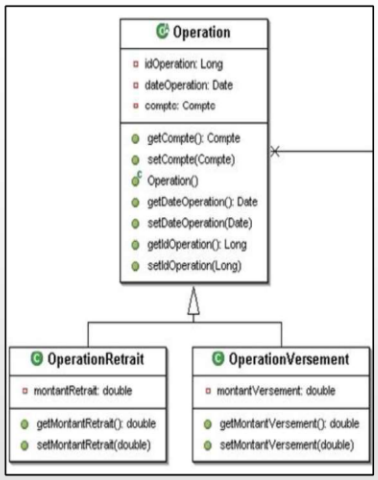


ID_COMPTE	COMPTE_TYPE	solde	decouvert	ID_CLIENT	TAUX
1	CompteB	15000	5000	1	*NULL*

Exemple de mapping

Operation.hbm.xml

```
<hibernate-mapping>
  <class name="metier.Operation" table="OPERATIONS">
      <id name="idOperation"> <generator class="increment"/>
      </id>
      <discriminator column="OPERATION_TYPE" type="string"/>
      <property name="dateOperation"/>
      <many-to-one name="compte" column="ID_COMPTE"/>
      <subclass name="metier.OperationVersement"
                    discriminator-value="Versement">
         <property name="montantVersement"/>
      </subclass>
      <subclass name="metier.OperationRetrait"
                    discriminator-value="Retrait">
         <property name="montantRetrait"/>
      </subclass>
</class> </hibernate-mapping>
```



idOperation	OPERATION_TYPE	dateOperation	ID_COMPTE	montantVersement	montantRetrait
1	Versement	2019-05-07 01:26:18	1	7000	*NULL*
2	Retrait	2019-05-07 01:26:18	1	*NULL*	2000

Module : JEE

Chapitre 3 : Struts

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

Département Génie Informatique & Mathématiques

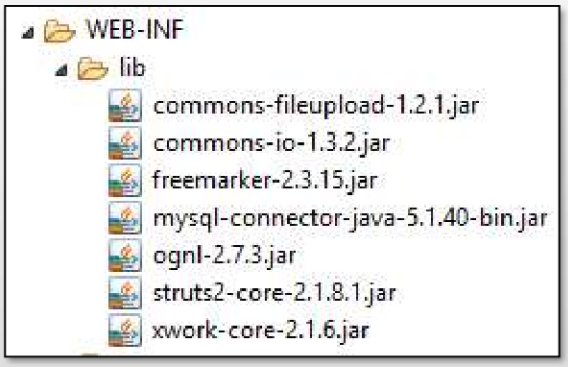
Framework Struts

Struts est un Framework open source (projet Jakarta de la fondation Apache) pour le développement d'applications Web Java respectant le modèle MVC:

- Architecture générique pour la partie contrôleur. Un contrôleur (une seule servlet) facilement configurable permettant d'associer des actions (méthode d'un objet Java) à des requêtes HTTP.
- Facilités pour la réalisation des vues (page JSP) avec des librairies de tags spécifiques pour créer facilement une vue.

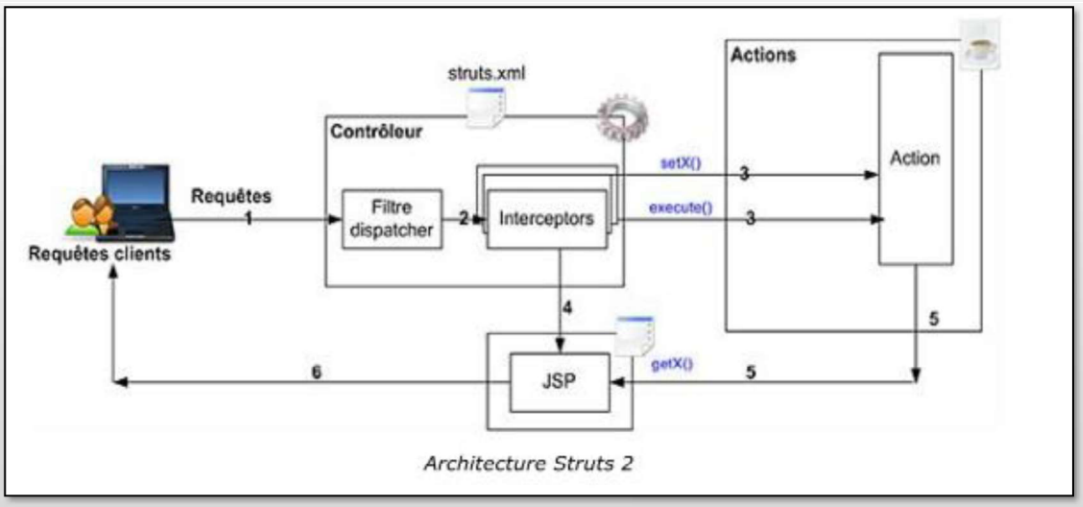
Installation du framework Struts 2

- Télécharger une version Struts (archives au format .zip) sur https://struts.apache.org/download.cgi.
- Placer les librairies au format .jar nécessaires à la mise en place de Struts 2 dans le répertoire WEB-INF/lib de l'application web.



Fonctionnement du framework Struts 2

1) Le client envoie des requêtes à partir d'URLs adaptées (avec paramètres).
2) À l'aide d'un filtre dispatcher, Struts consulte son fichier de configuration struts.xml afin de retrouver la configuration de l'action à exécuter
3) Les intercepteurs associés à l'action sont déclenchés et réalisent les services associés. L'intercepteur params assigne les valeurs de la requête à la classe d'action associée (accesseurs) et exécute la méthode de traitement (execute() par défaut).



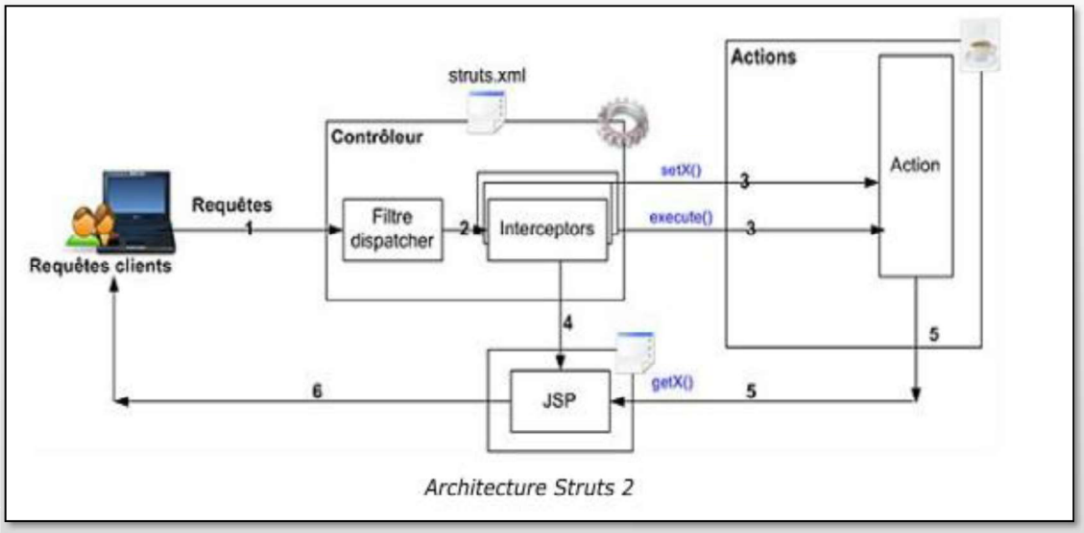
Architecture Struts 2

Fonctionnement du framework Struts 2

4) La vue à afficher est sélectionnée en accord avec le fichier struts.xml

5) La classe d'action transmet les données nécessaires à la vue.

6) La vue affiche au client les résultats traités.



Architecture Struts 2

Le fichier web.xml

- Dans le fichier web.xml, il faut configurer le filtre (FilterDispatcher) par lequel passeront toutes les requêtes (/*) associées à l'application.

```
<filter>
   <filter-name>struts2</filter-name>
     <filter-class> org.apache.struts2.dispatcher.FilterDispatcher
     </filter-class>
</filter>
<filter-mapping>
   <filter-name>struts2</filter-name>
   <url-pattern> /* </url-pattern>
 </filter-mapping
```

- Le filtre analyse systématiquement le fichier struts.xml ,(fichier de configuration de struts) pour lancer les intercepteurs associés à la requête .

Le fichier struts.xml

- Une application Struts possède un fichier de configuration **struts.xml** qui permet de configurer le mapping entre l'URL et l'implémentation de l'action à réaliser.
- Dans le fichier struts.xml il faut définir la configuration générale de struts :
  - Les paramètres de configuration de Struts.
  - Le mapping requêtes / actions / resultats (vue).
- Avec Struts, la méthode d'action de la classe est exécutée après que toutes les propriétés ont été traitées et affectées. De son côté, une méthode d'action retourne une chaîne de caractères de type String. Elle indique à Struts où le contrôleur doit se rediriger (par exemple, la chaîne "success" indique un traitement correct)

Le fichier struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
   "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
   "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
   <constant name="struts.enable.DynamicMethodInvocation" value="false" />
   <constant name="struts.devMode" value="false" />
<package name= "packageActions"  namespace="/" extends="struts-default">
      <default-action-ref name="NameDefaultAction"/>
      <action name="NameDefaultAction">
           <result>  JSPAction1  </result>   </action>
      ...
      <action name="NameActionX"  class= "NameClasseAction"
           method="NameMethod" >
        <result name="result1" > JSPActionXResult1</result>
        <result name="result2"> JSPActionXResult2 </result>
        …
      </action>
</package>  </struts>
```

- Il est usuel de mettre Struts.xml dans /src
- En absence de l'attribut method, la méthode execute() est utilisée.

Le fichier struts.xml

- Si une méthode d'une classe d'action retourne un résultat qui n'est pas présent dans une balise <result/> de l'action, Struts cherche alors un résultat correspondant dans le groupe <global-results/>.
- Cette balise permet de réaliser des groupements pour éviter les définitions répétitives (par exemple pour les redirections vers la page d'accueil du site, la page d'erreur ou d'authentification).

  Exemple

```
<global-results>
    <result name="accueil">/…/index.jsp</result>
</global-results>
```

- Si Struts ne trouve aucune correspondance de résultat, en portée locale ou dans la balise <global-results/>, une exception est levée.

Classes d'actions

Une classe d'action est une classe Java qui hérite de la classe ActionSupport. Elle doit respecter les règles suivantes:

- Comme pour les JavaBeans, une classe d'action doit avoir un constructeur par défaut et posséder des accesseurs (getter et setter)
- Une classe d'action doit avoir au moins une méthode pour réaliser une action (par défaut execute()). Chaque méthode porte un nom en association avec l'attribut method de la balise <action…/> et présents dans struts.xml.

Classes d'actions

Dans une classe d'action, plusieurs constantes peuvent être utilisés pour définir le résultat retourné par une méthode:

- SUCCESS : Exécution correcte de l'action, la page de succès adaptée doit être affichée.
- NONE : Exécution correcte de l'action mais aucun résultat ne doit être retourné.
- ERROR : Exécution incorrecte de l'action , l'utilisateur doit être redirigé vers une page d'erreur.
- INPUT : Erreur de validation des entrées ou saisies utilisateur. Elle précise généralement que la page de saisie doit être à nouveau affichée sans perte des données.
- LOGIN : L'action ne peut être exécutée car l'utilisateur n'est pas authentifié , forcer l'affichage de la page d'identification.

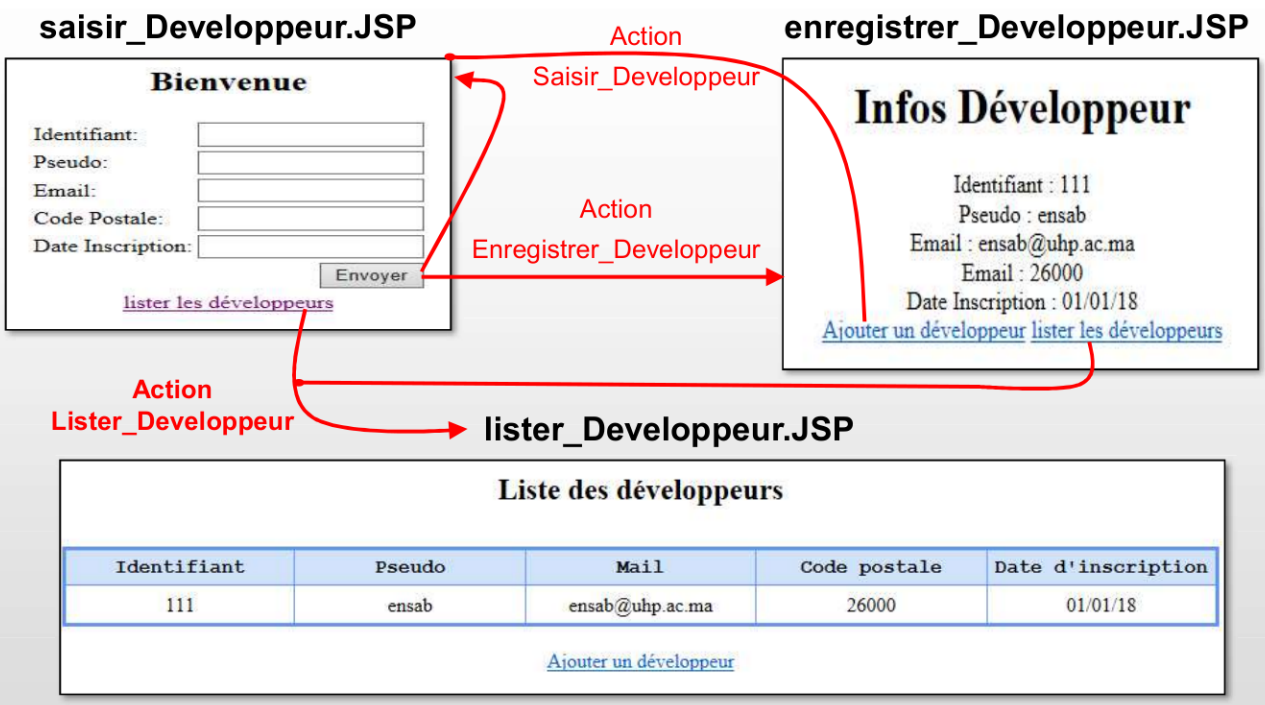
Exemple de projet

saisir_Developpeur.JSP

Affiche_Developpeur.JSP

lister_Developpeur.JSP

Exemple de projet



Exemple de projet : Fichier struts.xml

```
...
<package name="web.actions"  namespace="/" extends="struts-default">
      <default-action-ref name="saisir_Developpeur"/>
      <action name="saisir_Developpeur">
          <result>/jsp/saisir_Developpeur.jsp</result>  </action>
      <action name="enregistrer_Developpeur"
            class="web.actions.DeveloppeurAction"  method="enregistrer">
        <result name="success" >/jsp/afficher_Developpeur.jsp</result>
        <result name="input">/jsp/saisir_Developpeur.jsp</result>
      </action>
      <action name="lister_Developpeur"
            class="web.actions.DeveloppeurAction"  method="lister">
        <result name="success">/jsp/lister_Developpeur.jsp</result>
      </action>
</package>
 </struts>
```

Exemple de projet : Classe métier (Pojo) Developpeur

```
package ma.eclipse.metier;
import java.io.Serializable;
public class Developpeur implements Serializable{
    private int identifiant; private String pseudo;
    private String mail; private String codePostal;
    private java.util.Date dateInscription;
    public Developpeur (){ }
    public int getIdentifiant() { return identifiant;}
    public void setIdentifiant(int identifiant)  {
            this.identifiant = identifiant;}
    public String getPseudo() { return pseudo;}
    public void setPseudo(String pseudo) { this.pseudo = pseudo; }
    public String getMail() { return mail; }
    public void setMail(String mail) { this.mail = mail;}
    public String getCodePostal(){ return codePostal;}
    public void setCodePostal(String codePostal){ this.codePostal=codePostal; }
    public java.util.Date getDateInscription() { return dateInscription;}
    public void setDateInscription(java.util.Date dateInscription) {
            this.dateInscription = dateInscription; }
    }
```

Developpeur
- Identifiant :int - pseudo, mail, codePostal: String - dateInscription : Date;
+ Developpeur () + getXxx(): … + setXxx(…) : void

Exemple de projet : La classe Action « DeveloppeurAction »

```
package ma.eclipse.web.actions;
import ma.eclipse.metier.*; import java.util.ArrayList;
import com.opensymphony.xwork2.ActionSupport;
public class DeveloppeurAction extends ActionSupport {
  private int identifiant; private String pseudo;
  private String mail; private String codePostal;
  private java.util.Date dateInscription;
  private static ArrayList<Developpeur>
            listDeveloppeurs=new ArrayList<Developpeur>();
  public int getXxx() {return …;}     public void setXxx(…) {…}
  public String enregistrer() {
    if(this.pseudo.equals("")) {
            addFieldError("pseudo", "le pseudo ne doit pas etre vide"); return "input";  }
    Developpeur developpeur = new Developpeur ();
    developpeur.setIdentifiant(identifiant); developpeur.setPseudo(pseudo);
    developpeur.setMail(mail); developpeur.setCodePostal(codePostal);
    developpeur.setDateInscription(dateInscription);
    listDeveloppeurs.add(developpeur);
            return "success"; }
    public String lister() { return "success"; }
```

Tags de struts

- Le framework Struts 2 est livré en standard avec une librairie de tags pour les pages JSP
- La bibliothèque de balises Struts permet de programmer des services dynamiques côté client sans utiliser du code Java. Elle est composée de:
  - 1ière catégorie de tags pour la gestion des données
  - 2ème catégorie de tags pour les propriétés, paramètres et structures de contrôles (conditionnelles, boucles…).
- Les tags Struts possède des paramètres de mise en forme, des styles CSS à appliquer et des actions JavaScript associées.
- Pour utiliser les tag struts dans une page JSP, il faut ajouter la déclaration suivante en haut de chaque page et préfixer les balises par <s:nombalise/> :

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```


La balise <s:form/>

- La balise <s:form/> assure l'affichage d'un formulaire HTML. Les paramètres habituels sont disponibles comme l'action à exécuter, la méthode HTTP ou encore le type d'encodage utilisé.
- Struts utilise par défaut un thème de présentation permettant de générer un affichage sous forme de tableau XHTML (l'attribut theme permet de choisir un autre thème: simple, css_xhmtl, ajax,…)

Exemple projet : La page saisir_Developpeur.jsp

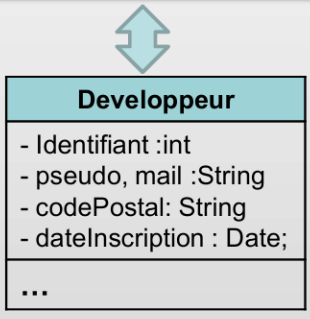
```
<%@ taglib prefix ="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html> <head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Saisir Developpeur</title>
</head> <body>
<center><h2>Bienvenue </h2>
<div id="formulaire">
    …
</div> </center> </body> </html>
```

saisir_Developpeur.jsp

Exemple de projet : La page saisir_Developpeur.jsp

```
<%@ taglib prefix ="s" uri="/struts-tags" %>
…
<div id="formulaire">
  <s:form method ="post" action="enregistrer_Developpeur">
     <s:textfield name="identifiant" id="identifiant"
          label="Identifiant" labelposition="left"> </s:textfield>
     <s:textfield name="pseudo" id="pseudo“ label="Pseudo"
          labelposition="left">  </s:textfield>
     <s:textfield name="mail" id="mail" label="Email"
          labelposition="left">  </s:textfield>
     <s:textfield name="codePostal" id="codePostal"
          label="Code Postale" labelposition="left"> </s:textfield>
     <s:textfield name="dateInscription" id="dateInceription"
          label="Date Inscription" labelposition="left"> </s:textfield>
     <s:submit value = "Envoyer"></s:submit>
  </s:form>
  <a href="lister_Developpeur.action">lister les développeurs
      </a><br/>
</div>
</center>
</body> </html>
```

saisir_Developpeur.jsp



La balise <s:property/>

- La balise <s:property/> permet d'afficher une information présente dans la classe d'action associée à ses accesseurs ou dans le contexte de l'application (application, session, request, parameters, attr).
- L'attribut value permet de donner le nom de la propriété (dans le cas d'une propriété de contexte , elle doit être précédée par : # suivi de sa portée)

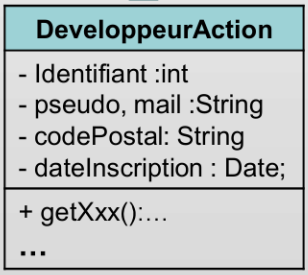
<s:property value="NameProperty"/>

Il est également possible d'utiliser les notations JSP EL (Expression Language) : ${NameProperty}.

Exemple de projet : La page Afficher_Developpeur.jsp

```
<%@ taglib prefix ="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html> <head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Afficher un développeur</title>
</head> <body>
 <center> <div>
    <h1>Infos Développeur</h1>
    Identifiant : <s:property value ="identifiant"/><br/>
    Pseudo : <s:property value = "pseudo"/><br/>
    Email : <s:property value ="mail"/><br/>
    Email : <s:property value ="codePostal"/><br/>
    Date Inscription : <s:property value="dateInscription"/>
    <br/>
    <a href="Saisir_Developpeur.action">
                        Ajouter un développeur</a>
    <a href="lister_Developpeur.action">
                    lister les développeurs</a><br/>
 </div> </center>
</body> </html>
```

Afficher_Developpeur.JSP



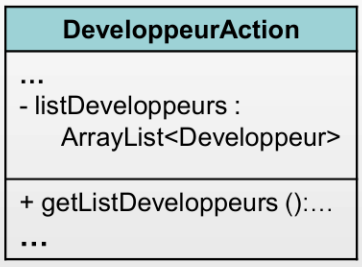
Les balises de test et d'iteration

- Les balises <s:if/>, <s:else/> et <s:elseif/> sont utilisées pour réaliser des tests conditionnels.
- La balise <s:iterator/> permet de parcourir un tableau ou une collection Java.

```
<s:iterator value="NameList">
    …
    <s:property value=" NameProperty"/>
    …
</s:iterator>
```


Exemple de projet: La page lister_Developpeur.jsp

```
<%@ taglib prefix ="s" uri="/struts-tags" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html> <head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Lister les developpeurs</title>
</head> <body>  <center>
   <h2>Liste des développeurs </h2>
   <s:if test="%{listDeveloppeurs.size()>0}">
      <s:iterator value="listDeveloppeurs"><br/>
          Identifiant : <s:property value="identifiant"/><br/>
          Pseudo : <s:property value="pseudo"/><br/>
          Mail : <s:property value="mail"/><br/>
          Code postale : <s:property value="codePostal"/><br/>
          Date d'inscription : <s:property value="dateInscription"/><br/>
</s:iterator>
</s:if>  <s:else >  Aucun développeur dans la liste  </s:else>
<br>
<a href="saisir_Developpeur.action">Ajouter un développeur</a><br/>
</center> </body> </html>
```



Module : JEE

Chapitre 4 : Spring MVC

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

Département Génie Informatique & Mathématiques

Historique de Spring

- Octobre 2002 Rod Johnson publie le code de base du framwork spring dans le livre «Expert One-on-One J2EE Design and Development»
- Mars 2004 Spring 1.0 sort sous licence Apache 2.0
- 2007 Sortie de Spring 2.5, avec support des annotations
- 2009 Achat de SpringSource par VMWare
- 2013 Création de Pivotal, une joint venture entre VMWare et EMC.

Spring Framework

spring

Informations

Développé par	VMware
Première version	25 juin 2003[1]
Dernière version	5.3.23 (15 septembre 2022)[2]
Dépôt	github.com/spring-projects/spring-framework
Écrit en	Java, Kotlin et Groovy
Système d'exploitation	Multiplateforme
Environnement	Multiplate-forme
Type	Framework web (en) Framework Bibliothèque Java (d)
Licence	Licence Apache
Site web	spring.io/projects/spring-framework

Framework Spring

- Spring n'est pas un serveur d'applications (Il peut fonctionner sans serveur d'applications : application «standalone»), il est utilisé généralement conjointement à un serveur d'applications léger (Tomcat par exemple)
- Le code de Spring est disponible gratuitement, à plusieurs endroits
  - Officiellement : le site de Spring (http://spring.io/)
  - GitHub
    - Spring Core (https://github.com/spring-projects/spring-framework)
    - Sous-projets Spring (https://github.com/spring-projects)

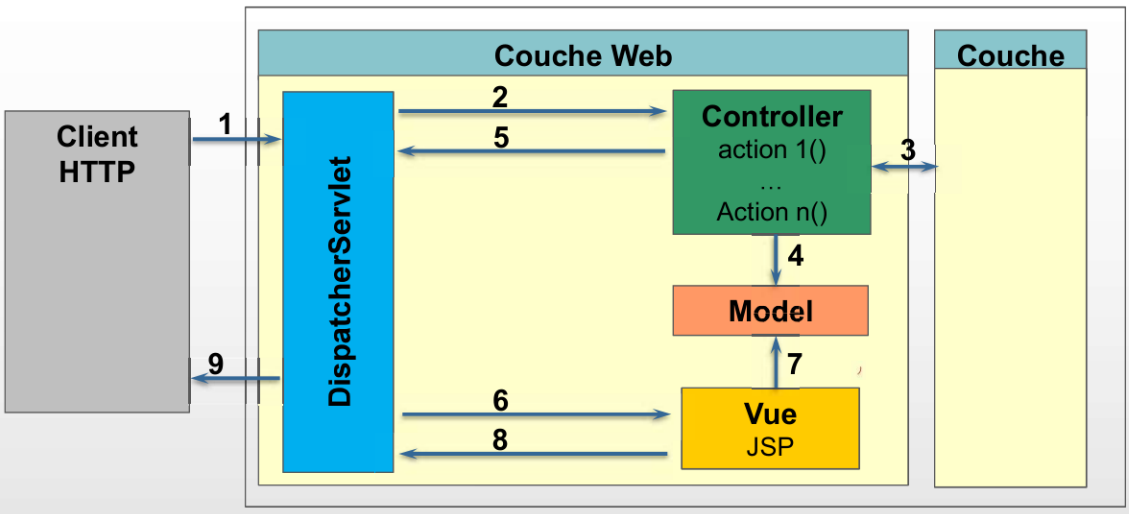
Le Framework SpringMVC

- SpringMVC est un framework de présentation, pour application WEB, suivant le modèle MVC, et fondé sur le conteneur léger de SPRING
- Le conteneur de SpringMVC permet de créer:
  - Le contexte de l'application Web
  - Les objets traitant les requêtes (Controller)
  - Les objets créant les pages HTML (View)
  - Les objets donnés des formulaires
  - Les liens avec les couches et BD
  - Le mapping des URL vers les contrôleurs
  - Le mapping des vues , etc.

SpringMVC et inversion du contrôle (Ioc)

- Le conteneur léger Spring sert à contenir un ensemble d'objets instanciés et initialisés, formant un contexte initial pour une application. Ce contexte est souvent construit à partir d'une description externe (xml) décrivant les objets à créer, les valeurs initiales et les **dépendances entre objets**.
- Les **dépendances** (liens) entre objets sont automatiquement créées à partir de la description (on parle d'injection de dépendances) et non par les objets eux-mêmes par programmation.
- **L'inversion du contrôle** permet ensuite de changer le comportement de l'application, en modifiant la description xml du conteneur, sans changer les éléments programmés.
- Avec **IoC**, la gestion des objets est confiée à un objet dédié. Celui-ci se charge de créer les instances requises et de les fournir par injection.

Architecture Spring MVC

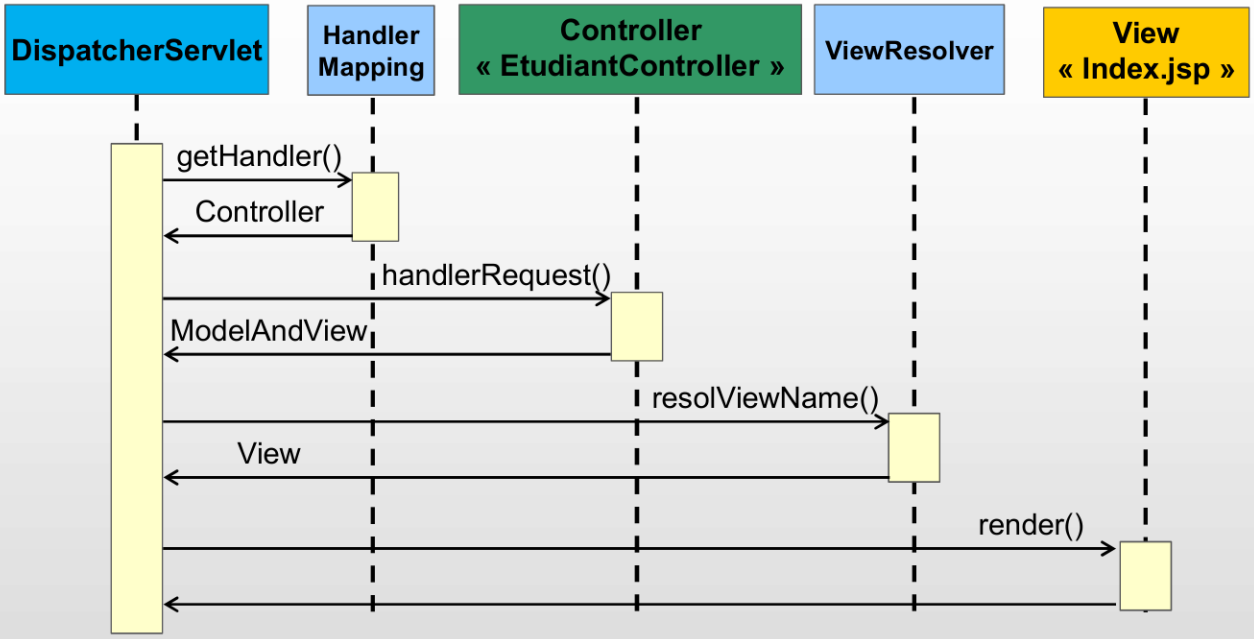


1–2) La requête est traitée par un contrôleur frontal «DispatcherServlet» qui assure le mapping URL/action. La méthode de l'action à choisir est définie dans la classe « Controller » (un contrôleur secondaire)

3–4–5) Appel à la couche métier et récupération du résultat qui sera stocker dans le modèle. Le «Controller» envoi le nom de la vue et le modèle au «DispatcherServlet»

6–7–8–9) Le « DispatcherServlet » appel la vue et lui transmet le modèle. La vue construite est retournée pour générer un rendu HTML

Architecture Spring MVC



- **Handler Mapping :** Assure le mapping entre requête et méthode d'action.
- **ViewResolver:** Fournit un mapping entre nom de vue et objet View

Exemple d'appliocation

Etudiant : entities

- Code : int
- Nom: String
- Ville : String

+ Etudiant()
+ getXxx():…
+ setXxx(…): void

Base de données MySql `dbetudiants`

```
CREATE TABLE `etudiant`(
  `code` int(11) NOT NULL,
  `nom` varchar(255),
  `ville` varchar(255),
  PRIMARY KEY (`code`) )
```

Hibernate

Etudiant.hbm.xml

Hibernate.cfg.xml

Index.jsp

Code	
NomPrenom	
Ville	
valider	

Code	
Rechercher	

Code	NomPrenom	Ville	
1	ENSA GI	B	Supprimer
2	FS	Casa	Supprimer

Insérer un nouvel étudiant

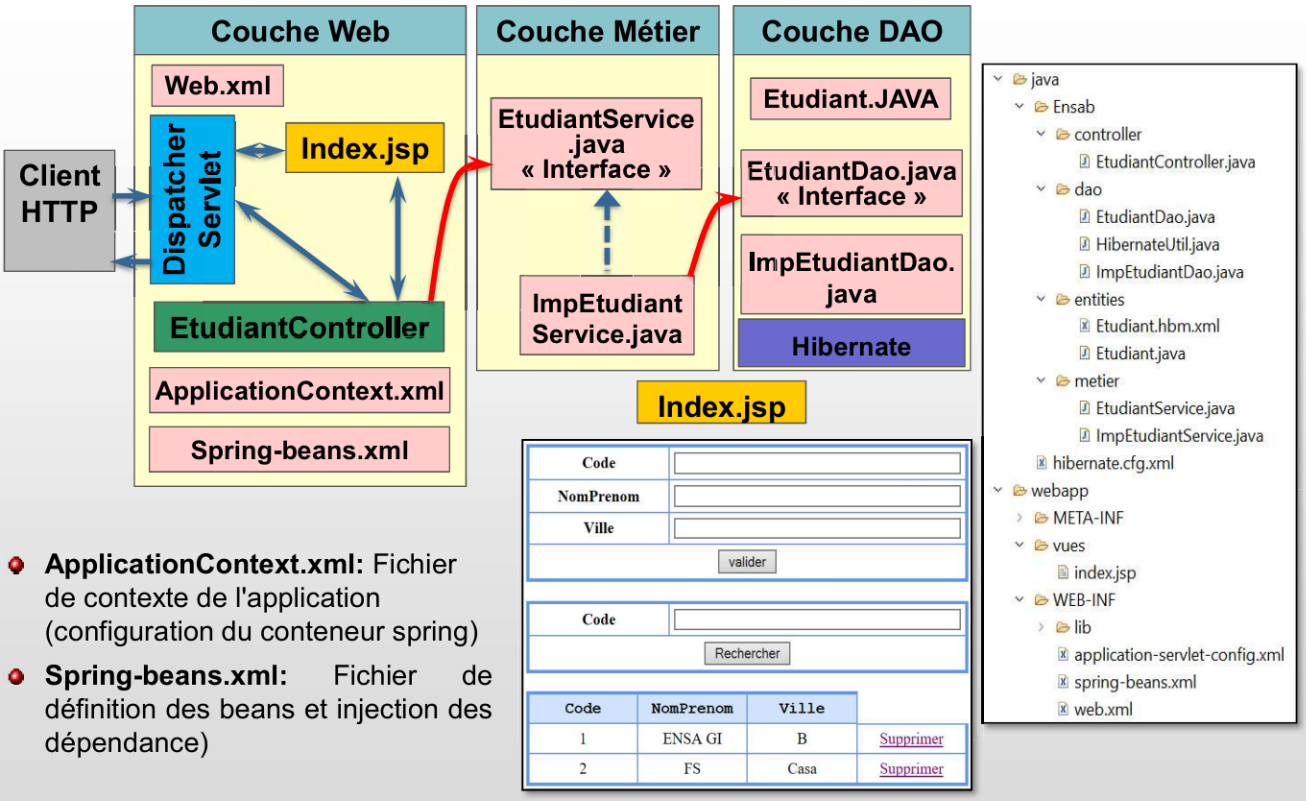
Recherche d'étudiant par code

Lister les étudiants (tous ou celui recherché)

Supprimer un étudiant

Lib Spring
Lib Hibernate
Connecteur MySql

Exemple d'appliocation



- **ApplicationContext.xml:** Fichier de contexte de l'application (configuration du conteneur spring)
- **Spring-beans.xml:** Fichier de définition des beans et injection des dépendance)

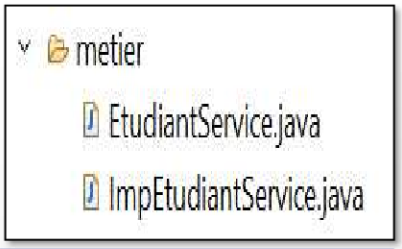
Couche dao: Interface et Implémentation

```
public interface EtudiantDao {
  public void inserer(Etudiant e);
  public void modifier(Etudiant e);
  public void supprimer(int code);
  public Etudiant rechercher(int code);
  public List<Etudiant> lister(); }
```


```
public class ImpEtudiantDao implements EtudiantDao{
  @Override
  public void inserer(Etudiant e) {
   Session session=HibernateUtil.getSessionFactory().openSession();
   session.beginTransaction();
   session.save(e); session.getTransaction().commit();  }
  @Override
  public void modifier(Etudiant e) {
   Session session=HibernateUtil.getSessionFactory().openSession();
   session.beginTransaction();
   session.merge(e); session.getTransaction().commit();}
  ...}
```

Couche metier: Interface et Implémentation

```
public interface EtudiantService {
   public void insererService(Etudiant e);
   public void modifierService(Etudiant e);
   public void supprimerService(int code);
   public Etudiant rechercherService(int code);
   public List<Etudiant> listerService();}
```




```
public class ImpEtudiantService implements EtudiantService {
   private EtudiantDao dao; //injection du code
   @Override
   public void insererService(Etudiant e) {dao.inserer(e);}
   @Override
   public void modifierService(Etudiant e) {dao.modifier(e);}
   @Override
   public void supprimerService(int code) {dao.supprimer(code);}
   @Override
   public Etudiant rechercherService(int code) {
     return dao.rechercher(code);}
   @Override
   public List<Etudiant> listerService() { return dao.lister();}
   public EtudiantDao getDao() { return dao;}
   public void setDao(EtudiantDao dao) {this.dao = dao;}    }
```


Gestion des Beans : Spring-beans.xml

Les beans Spring sont des POJOs instanciés par le conteneur(à partir de BeanDefinitions): Nom(id), classe(pleinement qualifiée), dépendances et Scope

```
<?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:tx="http://www.springframework.org/schema/tx"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
   http://www.springframework.org/schema/tx
   http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-2.5.xsd">
  <bean class="Ensab.dao.ImpEtudiantDao" id="daoEtudiant"> </bean>
  <bean class="Ensab.metier.ImpEtudiantService" id="serviceDao">
     <property name="dao" ref="daoEtudiant"></property>
  </bean>
</beans>
```

Injection de dépendance entre couche metier et dao

Ficher de contexte « application-servlet-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:sec="http://www.springframework.org/schema/security"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-2.5.xsd
   http://www.springframework.org/schema/security
   http://www.springframework.org/schema/security/spring-security-2.0.xsd">
  <context:component-scan base-package="Ensab.controller"/>
 <bean
 class="org.springframework.web.servlet.view.InternalResourceViewResolver">
   <property name="prefix" value="/vues/"/>
   <property name="suffix" value=".jsp"/>
 </bean>
</beans>
```


Le fichier web.xml

```
...
<servlet>
 <servlet-name>action  </servlet-name>
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <init-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>/WEB-INF/application-servlet-config.xml</param-value>
   </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>action</servlet-name>  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
<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>
<welcome-file-list> <welcome-file>index.htm</welcome-file> </welcome-file-list>
```

La classe Controller

- Un contrôleur est une classe JAVA annoté par @Controller
- En Spring MVC, une méthode de contrôleur est supposée renvoyer :
  - Le nom (une String) d'une vue qui servira à l'affichage (les données de la vue sont placées dans le « modèle » reçu comme argument)
  - L'objet ModelAndView
- Une méthodes action est annoté par @RequestMapping qui reçoit en paramètre l'URL de la requête.
- La méthode traite la requête et utilise la couche métier pour déterminer les données de modèle de la réponse (Spring copie les paramètres http dans les arguments de méthodes)

La classe Controller

```
package Ensab.controller;
@Controller
public class EtudiantController {
   @Autowired //injection de dépendance avec la couche Métier
   private EtudiantService service;
   @RequestMapping(value="/index")
   public String accueil(Model m) {
     m.addAttribute("listeE",service.listerService());
     return "index";}
   @RequestMapping (value="/saveE")
   public String save(Model m, int code,
      @RequestParam (value="nomPr") String nom,String ville) {
    Etudiant e= new Etudiant(); e.setCode(code);e.setNom(nom);
    e.setVille(ville); service.insererService(e);
    m.addAttribute("listeE",service.listerService());return"index";}
   ...
}
```

Vue index.jsp

```
<%@taglib prefix="ss" uri="http://java.sun.com/jsp/jstl/core" %>
...
<body>
<FORM action = "saveE.htm" method ="post">
 <TABLE > <TR> <TD> <b>Code</b>  </TD>
  <TD><input type="text“ name="code" style="width:300px"></TD></TR>
  <TR><TD> <b>NomPrenom</b></TD>
  <TD><input type="text" name="nomPr" style="width:300px"></TD></TR>
  <TR><TD> <b>Ville </b>  </TD>
  <TD><input type="text" name="ville" style="width:300px"></TD></TR>
  <TR><TD colspan =2 align=center >
    <INPUT type ="submit" name="Valider" value="valider"></TD></TR>
 </TABLE> </FORM> <br>
<ss:if test="${listeE.size()>0}" >
<TABLE>
  <TR><TH>Code</TH><TH>NomPrenom</TH><TH>Ville</TH></TR>
  <ss:forEach items="${listeE}" var='lst’>
    <TR><TD>${lst.code}</TD><TD>${lst.nom}</TD><TD>${lst.ville}</TD>
    </TR>
  </ss:forEach>
</TABLE> </ss:if>
</body> </html>
```

La notation ${…} permet d’accéder à des objets du modèle et leurs propriétés .

La classe Controller (suite)

```
@Controller
public class EtudiantController {
   ...
   @RequestMapping (value="/researchE")
   public String research(Model m, int code) {
    List<Etudiant> lst=new ArrayList<Etudiant>();
    Etudiant e=service.rechercherService(code);
    if (e!= null) lst.add(e); m.addAttribute("listeE",lst);
    return "index";}
   @RequestMapping (value="/deleteE")
   public String delete(Model m, int code) {
    service.supprimerService(code);
    m.addAttribute("listeE",service.listerService());return "index";}
}
```

Vue index.jsp (suite)

```
...
<body>
<FORM action = "saveE.htm" method ="post">
 ...
</FORM> <br>
<FORM action = "researchE.htm" method ="post">
  <TABLE > <TR> <TD> <b>Code</b>  </TD>
   <TD><input type="text" name="code" style="width:300px"></TD></TR>
   <TR> <TD colspan =2 align=center >
     <INPUT type="submit" name="rechercher" value="Rechercher"></TD>
   </TR>
  </TABLE> </FORM> <br>
 <TABLE>
  <TR><TH>Code</TH><TH >NomPrenom</TH><TH>Ville</TH></TR>
  <ss:forEach items="${listeE}" var='lst'>
    <TR><TD>${lst.code}</TD><TD>${lst.nom}</TD><TD>${lst.ville}</TD>
      <TD><a href="deleteE.htm?code=${lst.code}">Supprimer</a></TD>
    </TR>
  </ss:forEach>
 </TABLE>
</body></html>
```

Module : JEE

Chapitre 5 : Spring Boot

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

Département Génie Informatique & Mathématiques

Framework Spring Boot

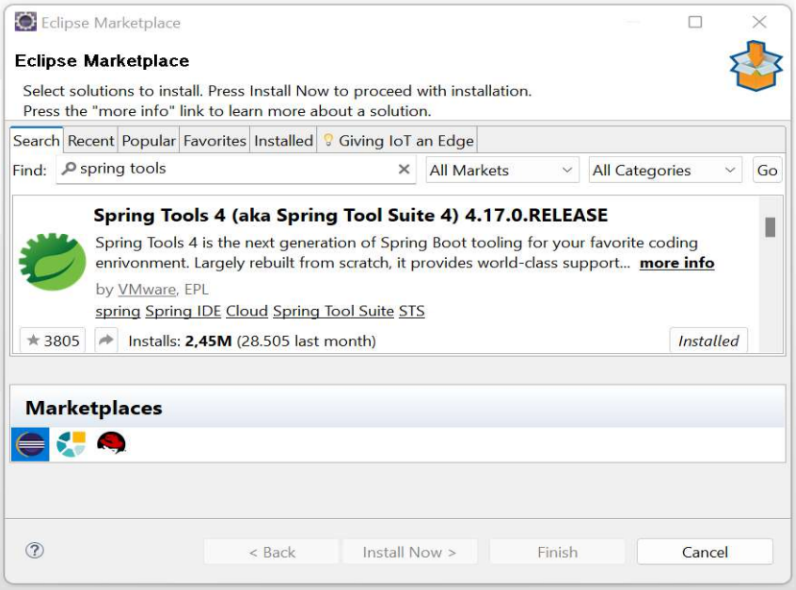
- Spring Boot est un sous projet de Spring visant la simplification des étapes de configuration.
- L'objectif de Spring Boot est de permettre aux développeurs de se concentrer sur des tâches techniques et non des tâches de configurations, de déploiements, etc.
  - Faciliter le développement d'applications et réduire l'utilisation des fichiers de configurations
  - Faciliter l'injection des dépendances (gestion des dépendances **Maven**).
- Récupération des dépendances depuis le site officiel Spring (http://spring.io/) – voir GitHub Spring Core (https://github.com/spring-projects/spring-framework) et GitHub Sous-projets Spring (https://github.com/spring-projects)

Spring MVC Vs Spring Boot

- Avec Spring MVC:
  - Trop de dépendance à gérer (parfois incompatibilité entre les versions)
  - Beaucoup de configuration (Contrôleur, vue,…)
- Avec Spring Boot:
  - Un démarreur (starter) qui comporte un paquet de dépendance nécessaires à la réalisation d'un projet donnée (fournir un conteneur Tomcat de servlet embarqué)
  - Assurer l'auto-configuration nécessaire à partir des dépendances spécifiées dans le démarreur (la plupart des beans sont créés si le ou les jar(s) adéquats sont dans le classpath).

Tools pour Eclipse

- Ajouter le plugin « Spring Tools » à l'IDE Eclips (Eclips Marketplace)

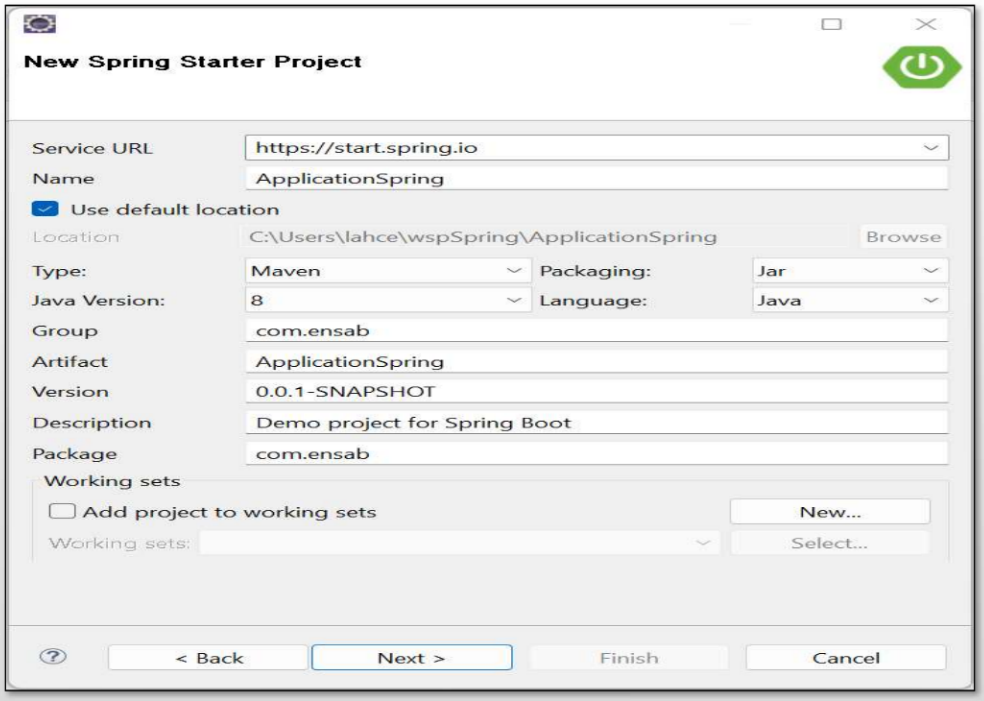


- Installer l'IDE "Spring Tools 4 for Eclipse" depuis https://spring.io/tools

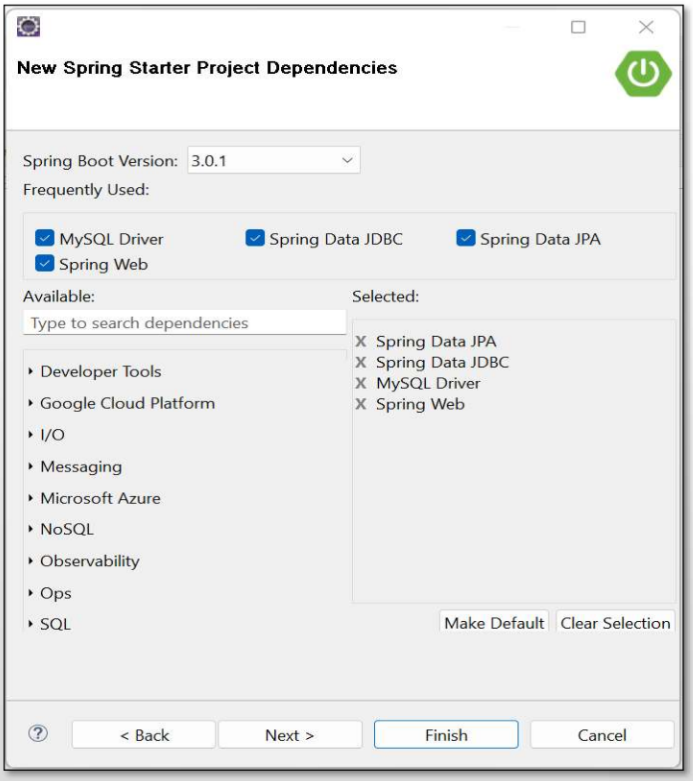
Spring Tools 4

Création d'un projet Spring Starter

Créer un nouveau projet « Spring Starter Project » (possibilité de générer le projet depuis «https://start.spring.io/»)

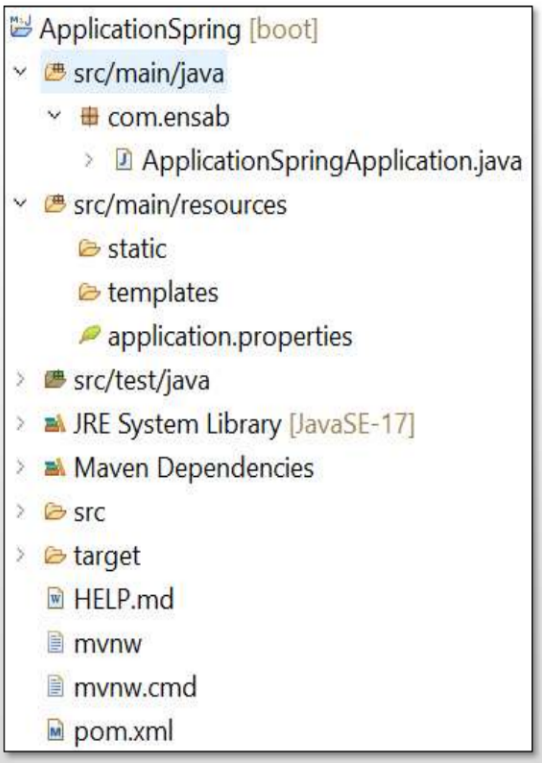


Spécification des dépendances



- **Spring Web** : Ajout du Tomcat embarqué , Spring Framwork et SpringMVC, ...
- **Spring Data JPA** : ajout des dépendances Hibernate, JPA et Spring Data qui faciliter à l'extrême l'accès aux données.
- **MySQL Driver** : Ajout du pilote JDBC MysQL

Structure du projet



- **Classe de démarrage de l'application**:

```
@SpringBootApplication
public class ApplicationSpringApplication{
public static void main(String[] args){
    SpringApplication.run(ApplicationSpring
      Application.class, args); } }
```

- **Ressources statiques « src/main/resoures »:**

  HTML, CSS, Java Script, images, etc…

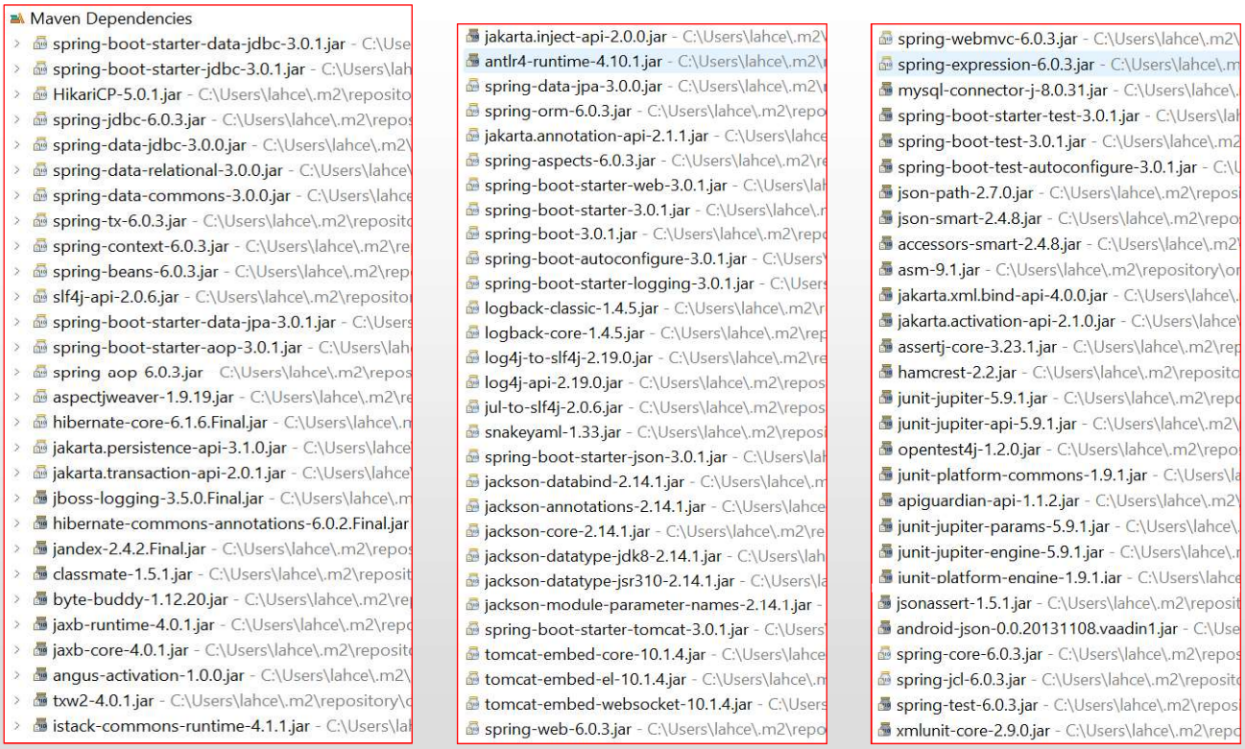
- **Vues de l'applications « package templates »**

  Vues interprétées coté serveur par le moteur de vue Tymeleaf

Fichier « pom.xml »

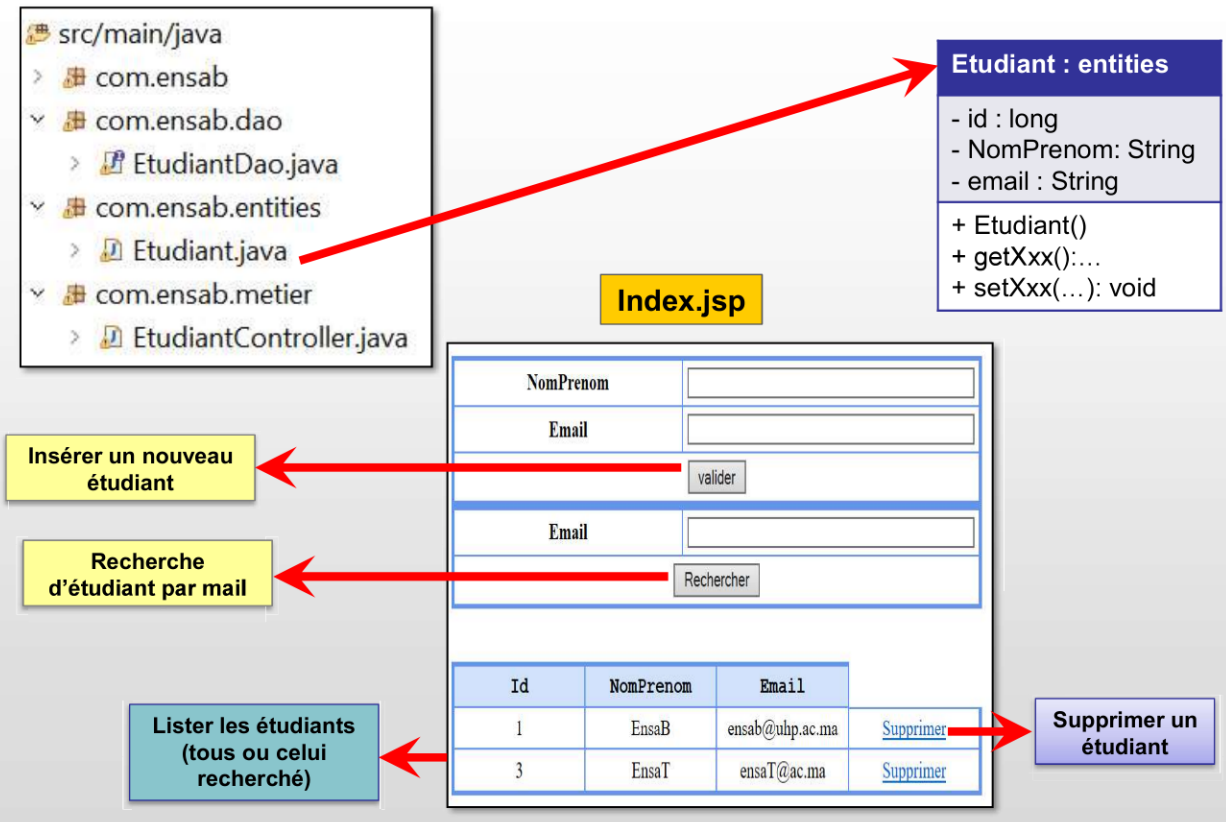
```
<project ...">
  ...
  <dependencies>
    <dependency> <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-data-jdbc</artifactId>
    </dependency>
    <dependency> <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency> <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency> <groupId>com.mysql</groupId>
       <artifactId>mysql-connector-j</artifactId>
       <scope>runtime</scope> </dependency>
  ...
</project>
```

Dépendances maven



Mise à jours des dépendances: Spring du menu contextuel du projet → addStarters
ou modification direct sur le fichier « pom.xml »

Exemple d'application

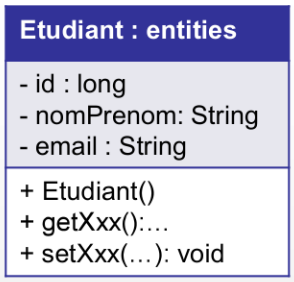


Fichier application.properties (src/main/resources)

```
spring.datasource.url=
   jdbc:mysql://localhost:3306/bdName?serverTimezone=UTC
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming.physical-
   strategy=org.hibernate.boot.model.naming.PhysicalNamingStr
   ategyStandardImpl
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect
   .MySQL8Dialect
spring.jpa.properties.hibernate.dialect.storage_engine=innodb
spring.jpa.properties.hibernate.format_sql=true
```


Classe entity Etudiant »

```
package com.ensab.entities;
import jakarta.annotation.Generated;
import jakarta.persistence.*;
@Entity
@Table(name="TEtudiant")
public class Etudiant {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  @Column (name="NomPrenom")
  private String nomPrenom;
  private String email;
  public Etudiant() { }
  public Etudiant(String nomPrenom, String email) {
     this.nomPrenom = nomPrenom;this.email = email; }
  public ... getXxx() {...}
  public void setXxx(...) {...}
}
```



JPA avec Spring Data

- L'objectif de Spring Data est de simplifier le travail des développeurs en prenant en charge l'implémentation des méthodes d'accès aux données.
- Spring Data s'organise autour de la notion de repository (référentiel de stockage) et fournit une interface marqueur générique **Repository<T,ID>**. Le type T correspond au type de l'objet géré par le repository. Le type ID correspond au type de la clé d'un objet.
- Spring Data JPA fournit l'interface **JpaRepository<T, ID>** assurant l'ensemble de méthodes plus spécifiquement adaptées pour interagir avec une base de données relationnelle (pour définir un repository, il suffit de créer une interface qui hérite d'une des interfaces **Repository**).

JpaRepository<T, ID> ⇢ **CrudRepository<T, ID>** ⇢ **Repository<T,ID>**

La couche dao

```
package com.ensab.dao;
import
 org.springframework.data.jpa.repository.JpaRepository;
import com.ensab.entities.Etudiant;
public interface EtudiantDao extends
        JpaRepository<Etudiant, Long> {
}
```

JPA avec Spring Data

L'interface JpaRepository<T, ID> comporte plusieurs méthodes pour manipuler une base de données:

- **count** (…) : Retourner le nombre d'occurrences des entités
- **existeById** (…) : Tester l'existence d'entité possédant un id donnée
- **save** (…) : Ajout de nouvelle instance (argument)
- **delete** (…) : Supprimer une instance (entité passée en argument)
- **findAll** ( ) : Retourner un itérable de toutes les instances de la BD
- **findById**(…) : Retourner l'instance correspondante à un id donné
- **deleteById** (…) : Supprimer une instance d'un id donné
- **deleteAll** (…) : Supprimer toutes entités (repository)
- …

JPA avec Spring Data (Test des méthodes)

```
public class ApplicationSpringApplication {
 public static void main(String[] args) {
  ApplicationContext context=
    SpringApplication.run(ApplicationSpringApplication.class,
    args);
  EtudiantDao dao= context.getBean(EtudiantDao.class);
  dao.save(new Etudiant("EnsaB", "ensab@uhp.ac.ma"));
  dao.save(new Etudiant("Ensa", "ensa@ac.ma"));
  List<Etudiant> lst= dao.findAll();
  for (Etudiant e: lst)
     System.out.println("Id: "+ e.getId()+ "NomPrenom: "+
       e.getNomPrenom()+ " Email: "+e.getEmail());
  dao.deleteById(2L);
  for (Etudiant e: dao.findAll())
     System.out.println("Id: "+ e.getId()+ "NomPrenom: "+
       e.getNomPrenom()+ " Email: "+e.getEmail());
 }
}
```

JPA avec Spring Data

- Pour définir une fonctionnalité spécifique Spring Data JPA utilise une convention de nommage suivante:
  - supprimer du début de la méthode les prefixes find, findAll, read, query, count et get et recherche la présence du mot By pour marquer le début des critères de filtre (le terme après By fait référence à un attribut de l'entité JPA du filtre).
  - Chaque critère doit correspondre à un paramètre de la méthode en respectant l'ordre.
- La requête est déduite de la signature de la méthode (query methods).
- L'annotation **@Query** peut être utilisée pour préciser directement la requête.
- L'utilisation de l'interface « **Page** » répartir le résultat d'une requête en plusieurs page comportant chacune un certain n'ombre d'instance (L'interface « Pageable » définit les informations telles que la taille et le numéro de la page).

JPA avec Spring Data (Test des méthodes)

```
public interface EtudiantDao extends JpaRepository<Etudiant,Long>{
 public Etudiant findByEmail(String email);
 long countByEmail(String email);
 List<Etudiant> findByNomPrenomOrEmail(String nomPrenom,String
    email);
 @Query("select e from Etudiant e where e.email like :p")
 public Page<Etudiant> chercherEtudiants(@Param("p")String
    partEmail,Pageable pageable);     }
```

ApplicationSpringApplication

```
for(Etudiant e:dao.findByNomPrenomOrEmail("EnsaB" ,"ensaT@ac.ma"))
  System.out.println("Id: "+ e.getId()+ " NomPrenom: "+
    e.getNomPrenom()+ " Email: "+e.getEmail());
Page<Etudiant> pageEtudiants=dao.chercherEtudiants("%",
  PageRequest.of(0, 5));
System.out.println("Page "+ pageEtudiants.getNumber()+ 1 + " / " +
  pageEtudiants.getTotalPages());
for(Etudiant e: pageEtudiants.getContent())
  System.out.println("Id: "+ e.getId()+ " NomPrenom: "+
    e.getNomPrenom()+ " Email: "+e.getEmail());   } }
```

La classe Controller

```
package com.ensab.metier;
...
@Controller
public class EtudiantController {
 @Autowired
 private EtudiantDao dao;
 @RequestMapping(value="/")
 public String accueil(Model m) {
  m.addAttribute("listeE",dao.findAll());
  return "index"; }
 @RequestMapping(value="/SEtudiant",method=RequestMethod.POST)
 public String save(Model m, @RequestParam (value="nomPr")
                    String nomPrenom,String email) {
  Etudiant e= new Etudiant(nomPrenom, email);
  dao.save(e);
  m.addAttribute("listeE",dao.findAll());
  return "index";}
 ...
 }
```


La classe Controller

```
@Controller
public class EtudiantController {
 ...
 @RequestMapping(value="/REtudiant",method=RequestMethod.POST)
 public String research(Model m, String email) {
  List<Etudiant> lst=new ArrayList<Etudiant>();
  Etudiant e=dao.findByEmail(email);
  if (e!= null) lst.add(e);
  m.addAttribute("listeE",lst);
  return "index";}
 @RequestMapping (value="/SupEtudiant{id}")
 public String delete(Model m, long id) {
   dao.deleteById(id);
   return accueil(m);
 }
```

Prise en charge JSP

application.properties

```
spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp
```

application.properties

```
<dependency>
 <groupId> org.apache.tomcat.embed</groupId>
 <artifactId> tomcat-embed-jasper</artifactId>
</dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```

Ou ajout direct des fichier jar d'une version compatible

Vue index.jsp

```
<%@taglib prefix="ss" uri="http://java.sun.com/jsp/jstl/core" %>
...
<body>
 <FORM action = "SEtudiant" method ="post">
  <TABLE >
   <TR> <TD> <b>NomPrenom</b> </TD>
     <TD><input type="text" name="nomPr" style="width:300px">
   </TD> </TR>
   <TR> <TD> <b>Email </b> </TD>
     <TD> <input type="text" name="email" style="width:300px">
   </TD> </TR>
   <TR> <TD colspan =2 align=center >
     <INPUT type ="submit" name="Valider" value="valider">
   </TD> </TR>
  </TABLE> </FORM>
 <FORM action = "REtudiant" method ="post">
  <TABLE >
   <TR> <TD> <b>Email</b> </TD>
     <TD> <input type="text" name="email" style="width:300px">
   </TD> </TR>
   <TR> <TD colspan =2 align=center >
   <INPUT type ="submit" name="rechercher" value="Rechercher">
   </TD> </TR> </TABLE>
 </FORM>
... </body> </html>
```

Vue index.jsp

```
<%@taglib prefix="ss" uri="http://java.sun.com/jsp/jstl/core" %>
...

<body>
 ...
 <br>
 <ss:if test="${listeE.size()>0}" >
  <TABLE >
   <TR> <TH >Id</TH> <TH >NomPrenom</TH> <TH >Email</TH> </TR>
   <ss:forEach items="${listeE}" var='lst'>
     <TR> <TD> ${lst.id}</TD> <TD> ${lst.nomPrenom} </TD>
     <TD> ${lst.email} </TD>
     <TD> <a href="SupEtudiant?id=${lst.id}" > Supprimer</a>
     </TD> </TR>
   </ss:forEach>
  </TABLE>
</ss:if>
</body> </html>
```

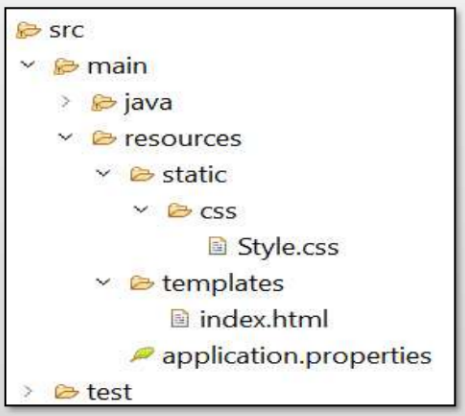

Utilisation des moteur de Template Tymeleaf

application.properties

```
spring.thymeleaf.cache=false
```

application.properties

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```



Utilisation des moteur de Template Tymeleaf (index.html)

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8"/>
<link rel="stylesheet" type="text/css"
                    th:href="@{css/Style.css}"/>
</head> <body>
 <FORM action = "SEtudiant" method ="post">
   <TABLE >
     <TR> <TD> <b>NomPrenom</b> </TD>
     <TD> <input type="text" name="nomPr" style="width:300px">
    </TD> </TR>
    <TR> <TD> <b>Email </b> </TD>
     <TD> <input type="text" name="email" style="width:300px">
    </TD> </TR>
    <TR> <TD colspan =2 align=center >
      <INPUT type ="submit" name="Valider" value="valider">
    </TD></TR>
  </TABLE>
 </FORM>
...
```


Utilisation des moteur de Template Tymeleaf (index.html)

```
...
 <FORM action = "REtudiant" method ="post">
  <TABLE >
   <TR> <TD> <b>Email</b> </TD>
    <TD> <input type="text" name="email" style="width:300px">
   </TD> </TR>
   <TR> <TD colspan =2 align=center >
    <INPUT type="submit" name="rechercher" value="Rechercher">
   </TD> </TR>
 </TABLE> </FORM> <br>
 <TABLE >
   <TR> <TH >Id</TH> <TH >NomPrenom</TH> <TH >Email</TH> </TR>
   <tr th:each ="lst:${listeE}">
     <TD th:text="${lst.id}"> </TD>
     <TD th:text= "${lst.nomPrenom}" > </TD>
     <TD th:text="${lst.email}"> </TD>
   </TR>
 </TABLE>
</body> </html>
```