Résumer Hibernate J2EE ORM Servlet JSP JSTL

- On charge les lib nécessaire à l'application
- On met les 2 fichiers utile/hibernateUtil.java et hibernate.cfg.xml dans le fichier src
- On définit la classe Produit

```
package metier.entities;

import java.io.Serializable;

@SuppressWarnings("serial")
public class Produit implements Serializable {

	private Long idProduit;
	private String designation;
	private double prix;
	private int quantite;

	public Long getIdProduit() {
		return idProduit;
	}

	public void setIdProduit(Long idProduit) {
		this.idProduit = idProduit;
	}

	public String getDesignation() {
		return designation;
	}

	public void setDesignation(String designation) {
		this.designation = designation;
	}

	public double getPrix() {
		return prix;
	}

	public void setPrix(double prix) {
		this.prix = prix;
	}

	public int getQuantite() {
		return quantite;
	}

	public void setQuantite(int quantite) {
		this.quantite = quantite;
	}

	public Produit(String designation, double prix, int quantite) {
		super();
		this.designation = designation;
		this.prix = prix;
		this.quantite = quantite;
	}

	public Produit() {
		super();
	}
```

```
}
```

Pour faire le mapping relationnel il y a 2 solution avec hibernate

1. Créer un fichier xml de mapping hibernate
2. On utilise les annotations JPA
    a. Pour une classe persistente on utilise @Entity
    b. Pour dire qui correspond à une table de la BD on utilise @Table(name= « PRODUITS »
    c. Pour la clé primaire on utilise @Id
    d. Pour dire auto-incrémentation @GenerateValue `@GeneratedValue(strategy=GenerationType.IDENTITY)` 1+1 qui doit générer
3. On utilise @Column(name= « ID_Produit »)
    a. `@Column(name = "DESIGNATION", length = 70)la longueur de chaine 70 caractère`

On crée l'interface de catalogue

```
package metier;

import java.util.List;

import metier.entities.Produit;

public interface ICatalogueMetier {

	public void ajouterProduit (Produit p);
	public void updateProduit (Produit p);
	public void deleteProduit (Long id);
	public List<Produit> produitsParMC (String mc);
	public Produit getProduit (Long id);
	public List<Produit> listeProduits();

}
```

On crée la classe de l'implémentation

Pour ajouter un objet on a besoin de créer un objet session hibernate

```
package metier;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;

import util.HibernateUtil;
import metier.entities.Produit;

public class CatalogueImpl implements ICatalogueMetier {

	@Override
	public void ajouterProduit(Produit p) {
		Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
		session.beginTransaction();
```

```
            try {
                  session.save(p);
            } catch (Exception e) {
                  session.getTransaction().rollback();
                  e.printStackTrace();
            }
            session.getTransaction().commit();
      }

      @Override
      public void updateProduit(Produit p) {
            Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
            session.beginTransaction();
            session.update(p);
            session.getTransaction().commit();
      }

      @Override
      public void deleteProduit(Long id) {
            Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
            session.beginTransaction();
            Object p = session.get(Produit.class, id);
            if (p == null)
                  throw new RuntimeException("Objet non trouvable");
            session.delete(p);
            session.getTransaction().commit();
      }

      @SuppressWarnings("unchecked")
      @Override
      public List<Produit> produitsParMC(String mc) {
            Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
            session.beginTransaction();
            Query requete = session
                        .createQuery("select p from produit p where
p.designation like :x");
            requete.setParameter("x", "%" + mc + "%");
            List<Produit> prods = requete.list();
            session.getTransaction().commit();
            return prods;
      }

      @Override
      public Produit getProduit(Long id) {
            Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
            session.beginTransaction();
            Object p = session.get(Produit.class, id);
            if (p == null)
                  throw new RuntimeException("Objet non trouvable");
           session.getTransaction().commit();
            return (Produit) p;
      }

      @SuppressWarnings("unchecked")
      @Override
      public List<Produit> listeProduits() {
            Session session =
```

```
HibernateUtil.getSessionFactory().getCurrentSession();
            session.beginTransaction();
            Query requete = session.createQuery("select p from produit p
");
            List<Produit> prods = requete.list();
            session.getTransaction().commit();
            return prods;
      }

}
```

La configuration de fichier hibernate.cfg.xml

Si on a installé le plugin hibernant on va new -> other ->hibernate-> hibernate Configuration File et finish

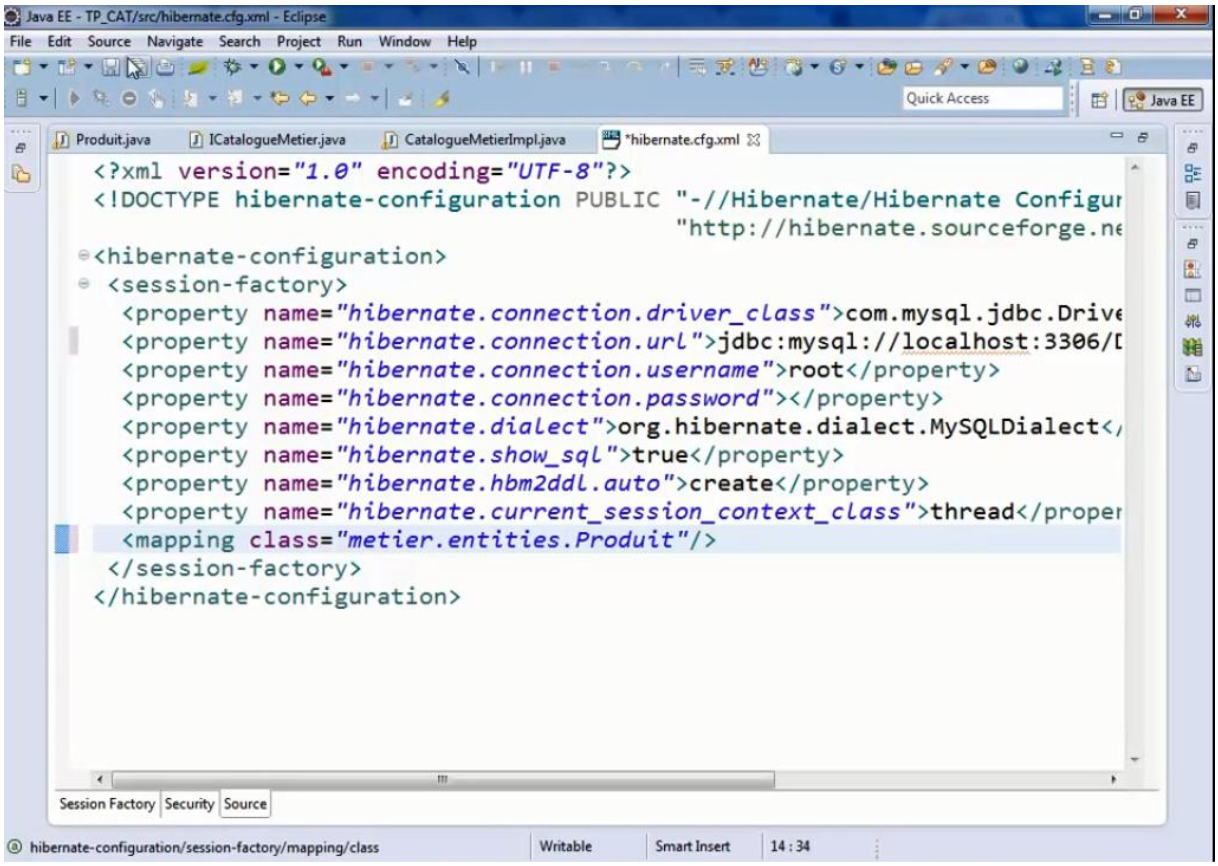
```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

      <session-factory>

            <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
            <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/bd_catalogue</p
roperty>
            <property name="hibernate.connection.username">root</property>
            <property name="hibernate.connection.password"></property>
            <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
            <property name="hibernate.show_sql">true</property>
            <property name="hibernate.hbm2ddl.auto">create</property>
            <property
name="hibernate.current_session_context_class">thread</property>
            <mapping class="metier.entities.Produit" />

      </session-factory>
</hibernate-configuration>
```



On crée une classe de test

```
package test;

import java.util.List;

import metier.CatalogueImpl;
import metier.ICatalogueMetier;
import metier.entities.Produit;

public class Test {

	public static void main(String[] args) {

		ICatalogueMetier metier = new CatalogueImpl();
		/*metier.ajouterProduit(new Produit("BSM 52", 1000, 10));
		metier.ajouterProduit(new Produit("HK 53", 1250, 60));
		metier.ajouterProduit(new Produit("LG 100", 800, 110));
		metier.ajouterProduit(new Produit("LKPM 95", 700, 90));*/
		List<Produit> produits = metier.listeProduits();
		for (Produit p : produits) {
	System.out.println("l'iD : " + p.getIdProduit() + " le designation : " + p.getDesignation());
		}
		Produit p = metier.getProduit((long) 5);
		System.out.println("le produit rechercher est : " + p.getDesignation());
		System.out.println("---------List par MC------");
		List<Produit> prods=metier.produitsParMC("M");
```

```
                for (Produit p3 : prods) {
                        System.out.println("l'iD : " + p3.getIdProduit() + " le designation : " +
p3.getDesignation());
                }
                System.out.println("------Modifier----------");
                p.setDesignation("SSSAG 20");
                p.setPrix(520);
                p.setQuantite(100);
                metier.updateProduit(p);
                System.out.println("la designation apres la MJR : "+p.getDesignation()+" le prix :
"+p.getPrix()+" qte : "+p.getQuantite());
                System.out.println("------------Supprission");

                metier.deleteProduit(3L);
        }

}
```

On tester les opérations à effectuer et on passe à la page web

Création de servlet

Initialisation de l'interface au de but de programme et en suite on initialise dans init

```
package web;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import metier.CatalogueImpl;
import metier.ICatalogueMetier;

@WebServlet("/ControlleurServlet")
public class ControlleurServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

        private ICatalogueMetier dao;

        public ControlleurServlet() {
                super();
        }

        public void init(ServletConfig config) throws ServletException {
                dao= new CatalogueImpl();
        }

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
```

```
	}

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

}
```

On commence de charger les données dans la page jsp

On va vers la méthode doGet et ajoute ce code sachant que nous avons ajouter la page index dans un dossier vues dans le répertoire Web-Content

```
request.setAttribute("produits", dao.listeProduits());
request.getRequestDispatcher("vues/index.jsp").forward(request, response);
```

Dans la page index.jsp on ajoute les fichiers de JSTL au début de la page et on déclare les composants nécessaires dans l'affichage

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


<table class="table table-striped">
			<tr>
				<th>ID PRODUIT</th>
				<th>DESIGNATION</th>
				<th>PRIX</th>
				<th>QUANTITé</th>
			</tr>
			<c:forEach items="${produits }" var="p">
				<tr>
					<td>${p.idProduit }</td>
					<td>${p.designation }</td>
					<td>${p.prix }</td>
					<td>${p.quantite }</td>
				</tr>
			</c:forEach>
</table>
```

Ensuite nous ajoutons le formulaire de l'ajout du produit dans la page index.jsp

```
<form action="controller.do" method="post" class="form-horizontal col-md-
12">
			<div class="form-group">
				<legend>Insertion PRODUIT</legend>
			</div>
			<div class="row">
				<div class="form-group">
				<label class="col-md-2">Désignation</label>
				<div class="col-md-10">
<input type="text" name="designation" class="form-control"
required="required" >
					</div>
				</div>
			</div>

			<div class="row">
				<div class="form-group">
```

```
                                    <label class="col-md-2">Prix</label>
                                    <div class="col-md-10">
<input type="text" name="prix" class="form-control" required="required" >
                                    </div>
                              </div>
                        </div>

                        <div class="row">
                              <div class="form-group">
                        <label class="col-md-2">Quantité</label>
                                    <div class="col-md-10">
<input type="text" name="quantite" class="form-control"
required="required" >
                                    </div>
                              </div>
                        </div>
                        <div class="row">
<button type="submit" name="action" value="save" class="btn btn-
primary">Save</button>
                        </div>
                  </form>
```

Et dans la méthode DoPost on ajoute les choses nécessaires pour ajouter un produit

```
            String action = request.getParameter("action");
            if (action.equals("save")) {
                  try {
                  String designation = request.getParameter("designation");
            double prix = Double.parseDouble(request.getParameter("prix"));
            int qte = Integer.parseInt(request.getParameter("quantite"));
                  dao.ajouterProduit(new Produit(designation, prix, qte));
                  } catch (Exception e) {
                        request.setAttribute("erreur", e.getMessage());
                  }

            }
            doGet(request, response);
```

On simplifier les choses on doit traiter toute le actions dans doPost pour cela on doit charger les deux expressions qu'on a déjà traité dans DoGet et on fait appelle à doPost

Mais on fait un test pour différencier entre le doPost et le doGet et on ajoute la suppression

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
            String action = request.getParameter("action");
            if (action != null) {
                  if (action.equals("save")) {
                        try {
                  String designation = request.getParameter("designation");
            double prix = Double.parseDouble(request.getParameter("prix"));
            int qte = Integer.parseInt(request.getParameter("quantite"));
            dao.ajouterProduit(new Produit(designation, prix, qte));
                        } catch (Exception e) {
                        request.setAttribute("erreur", e.getMessage());
                        }
                  } else if (action.equals("delete")){
                        Long id=Long.parseLong(request.getParameter("id"));
                        dao.deleteProduit(id);
                  }
```

```
            }
request.setAttribute ("produits", dao.listeProduits());
request.getRequestDispatcher("vues/index.jsp").forward(request, response);
}
```

Et on ajoute ce code dans le lien hypertexte dans le lien de supprimer

```
<td><a onclick="return confirm('Etes vous sûre?')"
href="controller.do?action=delete&id=${p.idProduit }">Supprimer</a> </td>
```

Ou bien avec un autre code de java scripte

```
<td><a href="javascript:confirmation('${p.idProduit }')">Supprimer</a>
</td>
```

Et on ajoute dans le haut une partie pour écrire le java scripte

```
<script type="text/javascript">
      function confirmation(id) {
            var conf = confirm("Êtes vous sûre de le supprimer?");
            if (conf == true) {
            document.location = "controller.do?action=delete&id=" + id;
            }
      }
</script>
```

On passe à la mise à jour

Dans le lien hypertexte de index.jsp

```
<td><a href="controller.do?action=update&id=${p.idProduit}">Modifier</a>
      </td>
```

Dans le doPost on doit recevoir le produit

```
else if (action.equals("update")) {
                  Long id = Long.parseLong(request.getParameter("id"));
                  Produit p = dao.getProduit(id);
                  request.setAttribute("produit", p);
}
```

Pour bien afficher les informations de Produit dans notre formulaire on ajoute les valeurs de chaque champ et on ajoute un champ pour l'id de notre produit

```
Id Produit : <input type="text" name="idProduit" value="${produit.idProduit }">
Designation : <input type="text" name="designation"
value="${produit.designation}">
Prix : <input type="text" name="prix" value="${produit.prix}">
Quantite : <input type="text" name="quantite" value="${produit.quantite}">
```

On ajoute un test pour bien connue si du save ou si de l'edit

```
<c:if test="${produit==null }">
    <button type="submit" name="action" value="save" >Save</button>
</c:if>

<c:if test="${produit!=null }">
    <button type="submit" name="action" value="edit">Editer</button>
</c:if>
```

Dans le ControlleurServlet on ajoute ce joli code

```
else if (action.equals("edit")) {
                  try {
            Long id = Long.parseLong(request.getParameter("idProduit"));
```

```
            String designation = request.getParameter("designation");
            double prix = Double.parseDouble(request.getParameter("prix"));
            int qte = Integer.parseInt(request.getParameter("quantite"));
            Produit p = new Produit(designation, prix, qte);
            p.setIdProduit(id);
            dao.updateProduit(p);
            } catch (Exception e) {
                  request.setAttribute("erreur", e.getMessage());
            }
}
```

Et pour terminer on va changer un peut le code Id dans la page index.jsp

```
<c:if test="${produit!=null }">
   <label>Id Produit : </label>     <b> ${produit.idProduit }</b>
   <input type="hidden" name="idProduit" value="${produit.idProduit }">
</c:if>
```

L'authentification

- L'authentification classique

  Dans le fichier web.xml on ajoute ce code :

```
      <security-constraint>
            <web-resource-collection>
                  <web-resource-name>ControlleurServlet</web-resource-name>
                  <url-pattern>*.do</url-pattern>
                  <http-method>GET</http-method>
                  <http-method>POST</http-method>
            </web-resource-collection>
            <auth-constraint>
                  <role-name>manager</role-name>
            </auth-constraint>
      </security-constraint>
      <login-config>
            <auth-method>BASIC</auth-method>
            <realm-name>Authentification</realm-name>
      </login-config>
```

Si on restart le serveur un formulaire d'authentification se déclenche

Si on veut ajouter des utilisateurs on doit aller au serveur ->tomcat-users-xml

```
<role rolename="manager"></role>
<user username="root" password="root" roles="manager"></user>
<user username="admin" password="123" roles="manager"/>
```

- L'authentification de notre formulaire

On va vers le fichier web.xml et on change BASIC par FORM

```
<auth-constraint>
            <role-name>manager</role-name>
            </auth-constraint>
      </security-constraint>
      <login-config>
            <auth-method>FORM</auth-method>
            <realm-name>Authentification</realm-name>
            <form-login-config>
                  <form-login-page>/login.html</form-login-page>
                  <form-error-page>/login.html</form-error-page>
            </form-login-config>
      </login-config>
```

On crée une page html dans web-content

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>L O G I N</title>
</head>
<body>
<form action="j_security_check" method="post">
      <label>Login</label>
      <input type="text" name="j_username">
      <label>Password</label>
      <input type="password" name="j_password">
      <button type="submit">Authentifier</button>
</form>
</body>
</html>
```

Et de même login et demot de passe qu'on a enregistré dans tomcat-users