

# TP 3 - Java EE

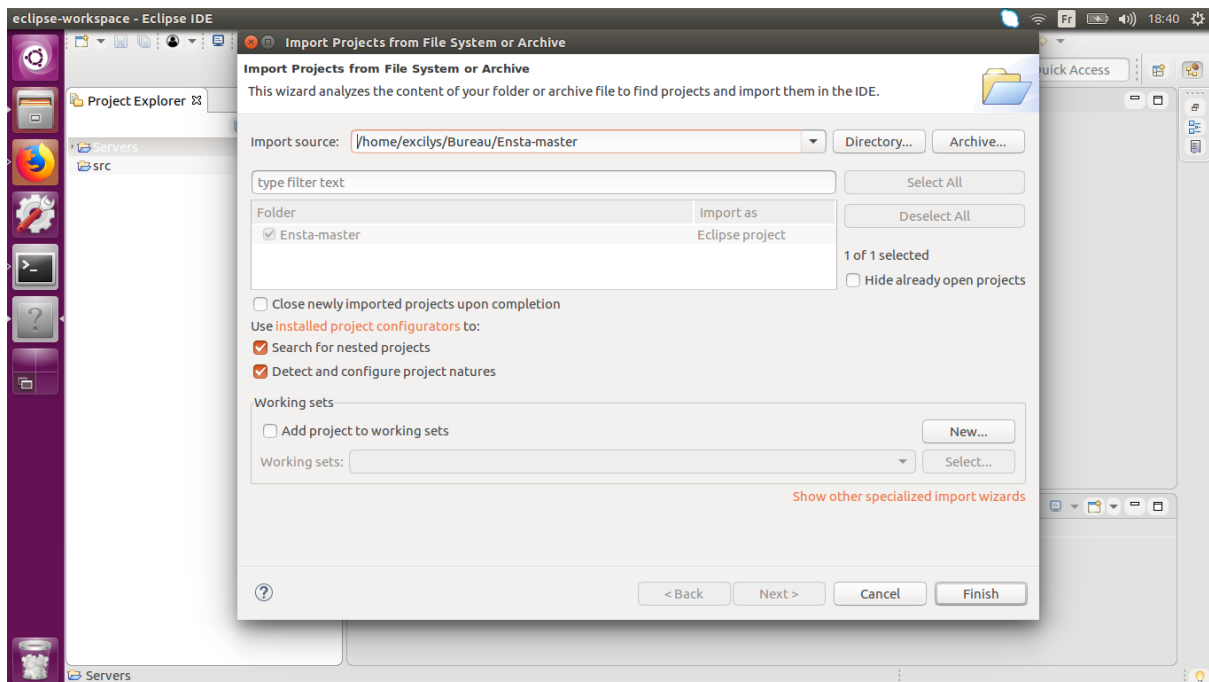
## 1. Configuration et installation

Pour réaliser ce TP, vous aurez besoin d'avoir sur votre ordinateur les éléments suivants :

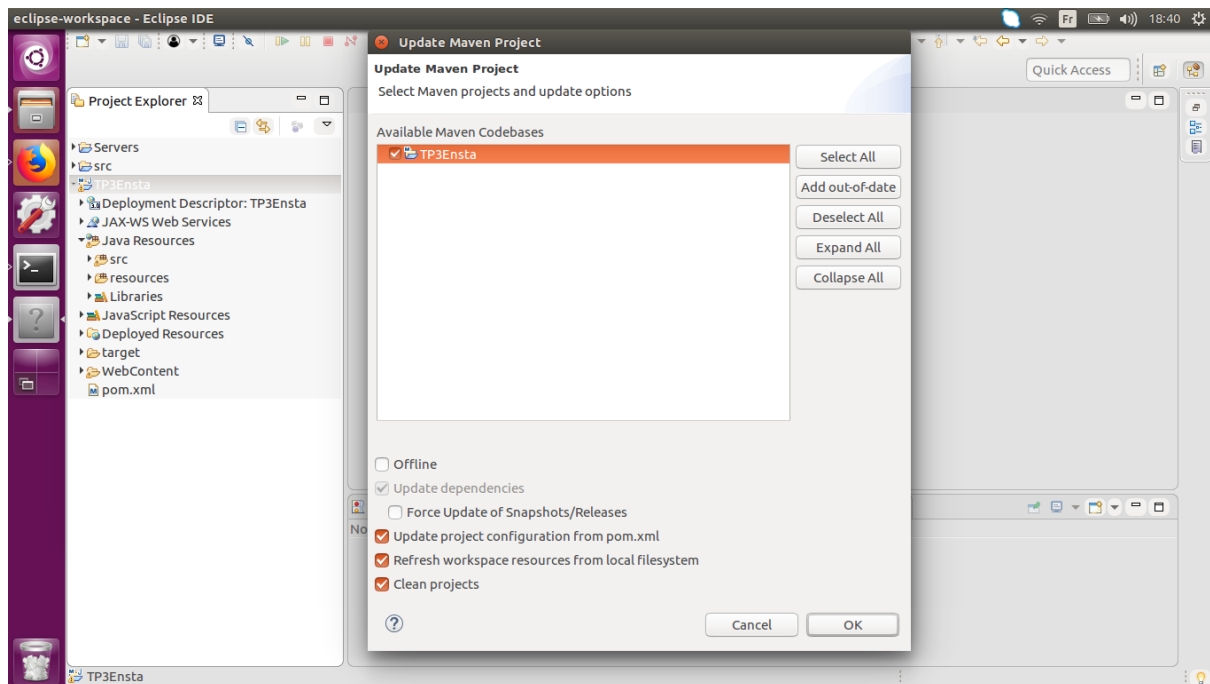
- **Une version de Java supérieure ou égale à 6**
- *Maven 3 au moins*
- [Eclipse IDE for Enterprise Developers](#) (contient déjà Maven)

Après avoir réalisé ces installations, vous pouvez importer l'archive **TP3\_fichiers.tar** (présente dans le drive) avec Eclipse en faisant la manipulation suivante :

- Sélectionner « fichier/OpenProjectFromFileSystem... » dans la barre d'Eclipse.
- Sélectionner le dossier TP3 dans « import source » puis « Finish » :

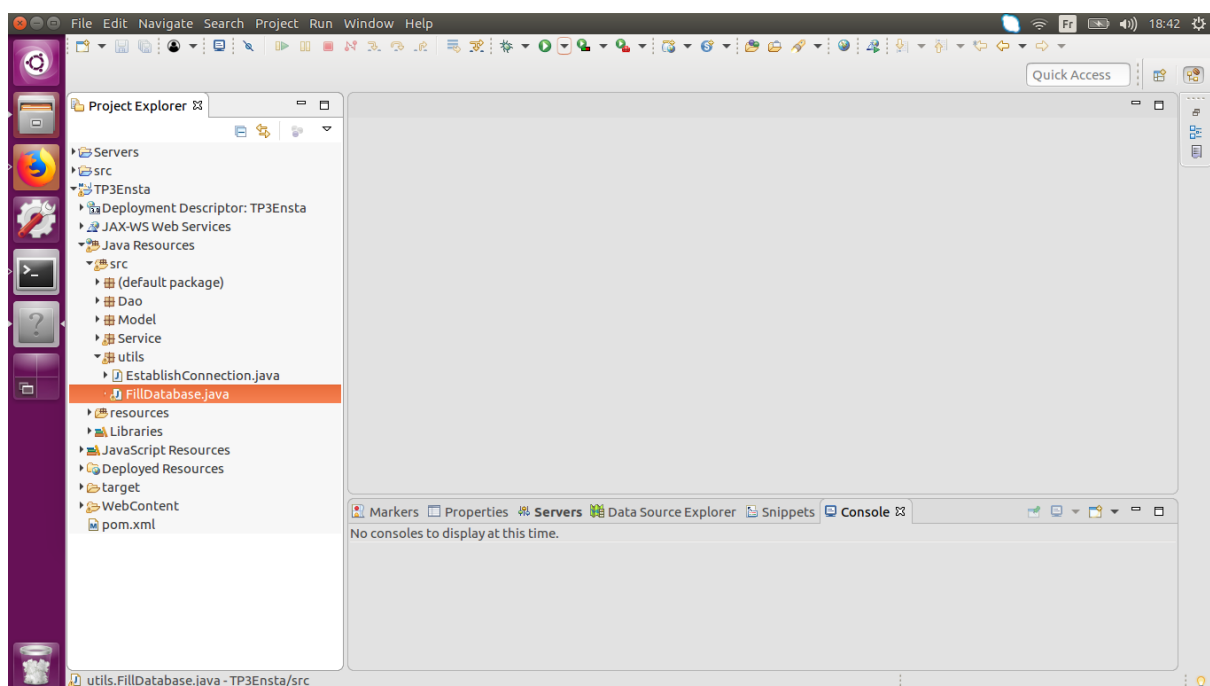


- Votre projet devrait apparaître dans la barre de navigation d'Eclipse. Au cours de votre TP, des classes seront sûrement soulignées en rouge. Pour régler cela, il faudra faire un clic-droit sur le projet TP3Ensta qui est apparu, sélectionner « maven » et faire « update project... » :



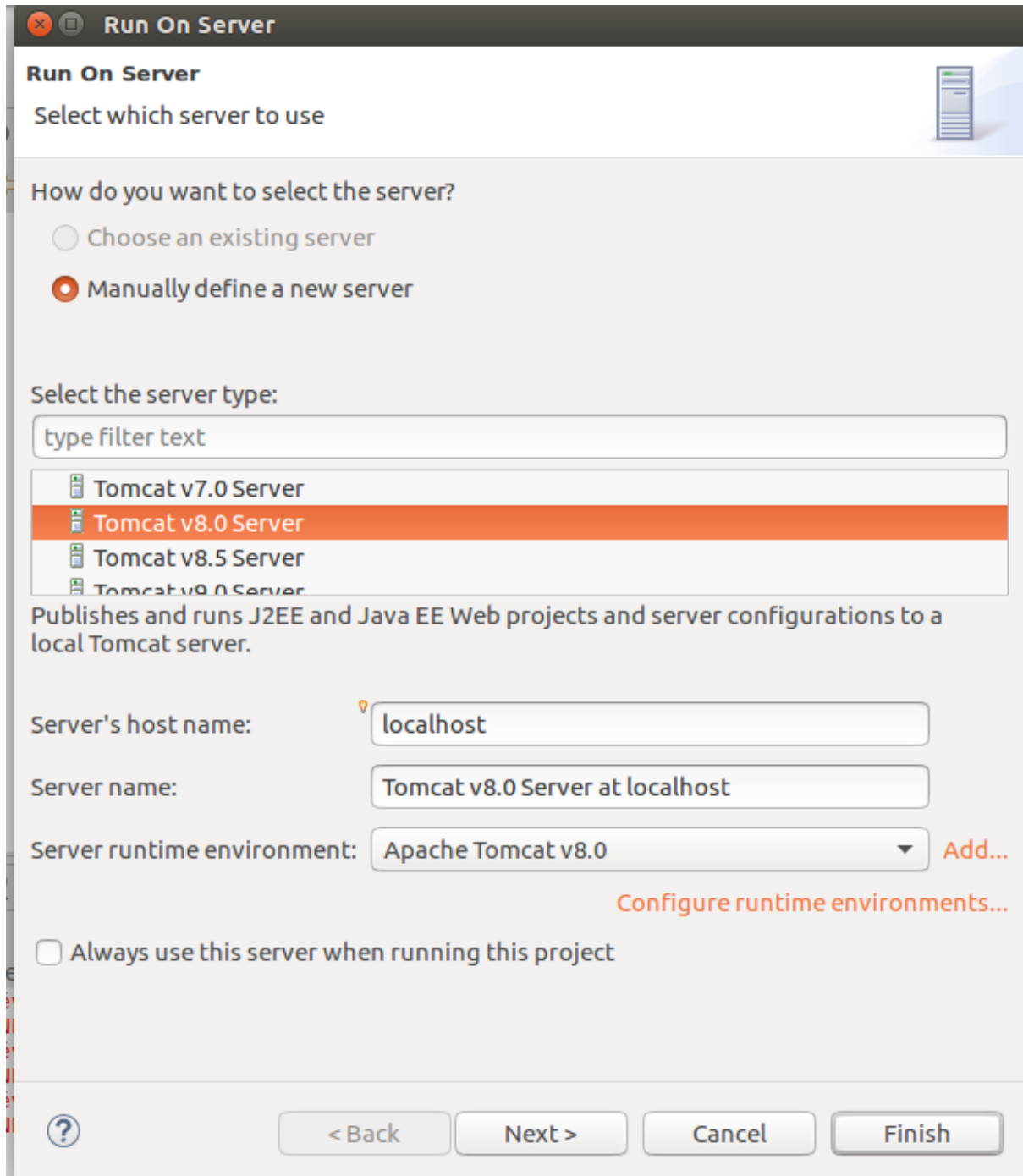
Sélectionner TP3Ensta puis « Ok »

- Pour finir, exécuter la classe « FillDatabase » pour remplir la base de données (à ne faire qu'une fois sous peine de voir vos nouvelles données supprimées) :




Pour lancer l'application avec un serveur Tomcat:

- click droit sur TP3Ensta puis « Run as... »


The image shows a 'Run On Server' dialog box from an IDE. The title bar says 'Run On Server'. Inside, the main heading is 'Run On Server' with a subtitle 'Select which server to use'. There are two radio buttons: 'Choose an existing server' (unselected) and 'Manually define a new server' (selected). Below this is a section 'Select the server type:' with a text input field containing 'type filter text'. A list of server types is shown below the input: 'Tomcat v7.0 Server', 'Tomcat v8.0 Server' (highlighted in orange), 'Tomcat v8.5 Server', and 'Tomcat v9.0 Server'. Below the list is a descriptive text: 'Publishes and runs J2EE and Java EE Web projects and server configurations to a local Tomcat server.' There are three input fields: 'Server's host name:' with 'localhost', 'Server name:' with 'Tomcat v8.0 Server at localhost', and 'Server runtime environment:' with a dropdown menu showing 'Apache Tomcat v8.0'. To the right of the dropdown is an 'Add...' button. Below these fields is a link 'Configure runtime environments...'. At the bottom, there is a checkbox 'Always use this server when running this project' which is unchecked. The bottom of the dialog has a question mark icon and four buttons: '< Back', 'Next >', 'Cancel', and 'Finish'.

Sélectionner tomcat v8.0 puis « Next »

Vous allez alors télécharger le serveur :

 New Server Runtime Environment

### Tomcat Server

 The name is already in use. Specify a different name.



Name:

Apache Tomcat v8.0

Tomcat installation directory:

C:\Users\Bensa\Downloads

Browse...

apache-tomcat-8.0.36

Download and Install...

JRE:

Workbench default JRE

Installed JREs...



< Back

Next >

Finish

Cancel

Sélectionner un dossier ou le télécharger avec « Browse... » puis « Download and Install ».

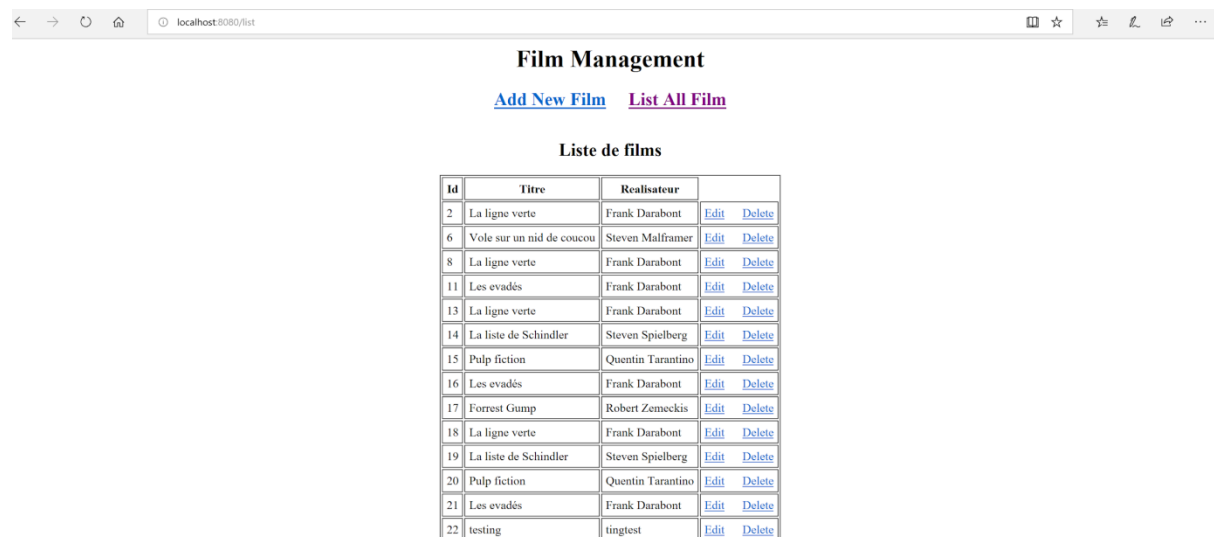
Quand l'installation est fini, cliquer sur « Finish ».

## 2. Présentation du TP

Le but du TP est de faire la gestion d'une bibliothèque de films, c'est-à-dire les stocker, les modifier, les afficher, les supprimer sur une page web.

L'objectif est d'obtenir le fonctionnement suivant :

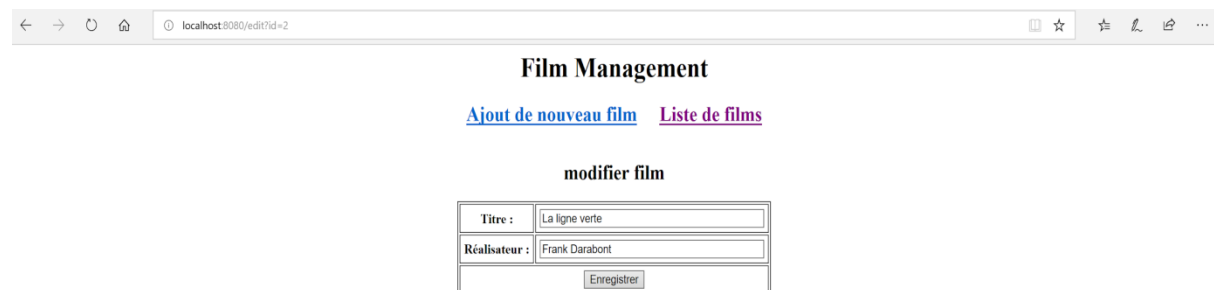
Lorsque l'on arrive sur l'URI **/list**, on obtient la liste de tous les films.



Film Management			
<a href="#">Add New Film</a> <a href="#">List All Film</a>			
Liste de films			
Id	Titre	Réalisateur	
2	La ligne verte	Frank Darabont	<a href="#">Edit</a> <a href="#">Delete</a>
6	Vole sur un nid de coucou	Steven Malframer	<a href="#">Edit</a> <a href="#">Delete</a>
8	La ligne verte	Frank Darabont	<a href="#">Edit</a> <a href="#">Delete</a>
11	Les évadés	Frank Darabont	<a href="#">Edit</a> <a href="#">Delete</a>
13	La ligne verte	Frank Darabont	<a href="#">Edit</a> <a href="#">Delete</a>
14	La liste de Schindler	Steven Spielberg	<a href="#">Edit</a> <a href="#">Delete</a>
15	Pulp fiction	Quentin Tarantino	<a href="#">Edit</a> <a href="#">Delete</a>
16	Les évadés	Frank Darabont	<a href="#">Edit</a> <a href="#">Delete</a>
17	Forrest Gump	Robert Zemeckis	<a href="#">Edit</a> <a href="#">Delete</a>
18	La ligne verte	Frank Darabont	<a href="#">Edit</a> <a href="#">Delete</a>
19	La liste de Schindler	Steven Spielberg	<a href="#">Edit</a> <a href="#">Delete</a>
20	Pulp fiction	Quentin Tarantino	<a href="#">Edit</a> <a href="#">Delete</a>
21	Les évadés	Frank Darabont	<a href="#">Edit</a> <a href="#">Delete</a>
22	testing	tingtest	<a href="#">Edit</a> <a href="#">Delete</a>

À côté de chaque film, faites en sorte d'avoir la possibilité d'éditer ou de supprimer ledit film.

L'édition peut se présenter ainsi :



Film Management	
<a href="#">Ajout de nouveau film</a> <a href="#">Liste de films</a>	
modifier film	
Titre :	<input type="text" value="La ligne verte"/>
Réalisateur :	<input type="text" value="Frank Darabont"/>
<input type="button" value="Enregistrer"/>	

Pour ce qui est de la page d'ajout de nouveaux films, elle est similaire à celle de l'édition.

Dans l'archive que vous avez téléchargée, vous avez déjà à votre disposition la **persistance**, dans le fichier **EstablishConnection.java** dans le **package utils**.

### *C'est quoi la persistance ?*

C'est la couche qui permet entre autres de communiquer avec la base de données.

### 3. Si vous travaillez sur Eclipse

#### a. Première partie : Modèle

Nous allons commencer par nous intéresser à la création de notre **modèle**.

##### *C'est quoi un modèle ?*

On crée une classe Java qui aura les mêmes attributs que la table de la BDD à laquelle elle correspond. Cette classe sera chargée de représenter les instances de la table dans notre programme Java.

C'est le modèle. (cf. le cours)

Pour ce faire, vous allez tout d'abord créer un **package** que vous nommerez **model**. Dans ce package, créez une classe que vous nommerez **Film**.

##### **Chose à savoir avant de vous lancer dans l'écriture de la classe Film**

Pour ce TP, nous vous fournissons la table film :

```
CREATE TABLE IF NOT EXISTS film (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    titre VARCHAR(100),  
    realisateur VARCHAR(50)  
);
```

**Tout est déjà géré à ce niveau là, ne vous en préoccupez pas ! Faites simplement en sorte de ne pas oublier d'exécuter FillDatabase.java une fois !**

La classe Film n'est qu'un reflet de ce qu'est la table film. Faites donc en sorte d'avoir dans cette classe les champs id, titre et realisateur ainsi que les mutateurs/accesseurs associés.

Ajouter à cette classe au moins un constructeur qui instancie tous les champs.

#### b. Deuxième partie : DAO (data access object) et Service

Nous allons maintenant nous attaquer au **DAO**.

##### *C'est quoi un DAO ?*

Il permet de gérer l'accès aux données. Les DAO...

- Encapsulent la logique liée à la base de données ;
- Respectent le CRUD (**C**reate, **R**ead, **U**ppdate, **D**eleter) ;
- Suivent généralement le design pattern « Singleton ».

(cf. le cours)

Tout d'abord commencé par créer un **package** que vous nommerez **dao**.

Dans ce package vous créerez une classe **FilmDao**.

Vous commencerez par écrire les requêtes qui seront utilisées par les méthodes de votre DAO.

Pour ce faire, prenons pour exemple ce qui à été fait dans la classe FillDatabase :

```
String InsertQuery = "INSERT INTO film (titre, realisateur) values (?,?)";
```

Dans la classe FillDatabase, la variable `InsertQuery` contient la requête sql qui est utilisée pour insérer un enregistrement dans la table film. Vous pouvez voir qu'à la fin, dans cette requête, nous avons `(?,?)`.

Ces deux `?` nous permettent d'appliquer une méthode sur la String pour les changer en utilisant un `PreparedStatement`, ainsi nous pouvons utiliser une seul String (`InsertQuery`) pour faire toutes les insertions dont on a besoin dans la table film. Voici la création du `PreparedStatement` :

```
Connection connection = EstablishConnection.getConnection();
PreparedStatement insertPreparedStatement = null;
```

puis le remplacement des `?` :

```
insertPreparedStatement = connection.prepareStatement(InsertQuery);
insertPreparedStatement.setString(1, "Forrest Gump"); // on change Le premier ?
insertPreparedStatement.setString(2, "Zemeckis"); // on change Le deuxième ?
```

Fort de cette explication, vous pouvez maintenant écrire les cinq requêtes (qui sont de simples chaîne de caractères à l'image de `InsertQuery`) dont vous aurez besoin dans les méthodes du DAO :

- **get** : Permet de récupérer un film depuis la base de données à partir d'un id donné en argument.
- **getAll** : Permet de récupérer tous les films.
- **create** : Permet de créer un film à partir d'un objet film donné en argument.
- **update** : Permet de mettre à jour un film à partir d'un objet film donné en argument.
- **delete** : Permet de supprimer un film à partir d'un id donné en argument.

Après avoir fini d'écrire le DAO, créez un nouveau package que vous nommerez **service**.

### **C'est quoi un Service ?**

Il gère la logique de l'application et les traitements à appliquer aux données. (*cf. le cours*)

Dans ce package, créez une classe `FilmService` qui ne fera qu'instancier un objet `FilmDao` et l'utiliser. En temps normal, un Service fournit une couche d'abstraction au DAO qui permet par exemple d'ajouter une étape de vérification avant de faire appel à la méthode `create` du DAO.

Dans notre cas, le Service ne fait qu'appeler les méthodes (CRUD) de notre DAO, mais il est tout de même préférable de prendre de bonnes habitudes.

### c. Troisième partie : Servlet et JSP

Maintenant créer un **package ui** dans lequel vous écrirez votre classe **Servlet**.

#### *C'est quoi une Servlet ?*

Code qui s'exécute côté serveur.

- Elle est le centre névralgique de l'application.
- Elle reçoit une requête du client, elle effectue des traitements et renvoie le résultat.
- Une servlet peut être invoquée plusieurs fois en même temps pour répondre à plusieurs requêtes simultanées.

*(cf. le cours)*

Lors de la création de votre classe n'oubliez de la faire hériter de **HttpServlet**, de plus n'oubliez d'y ajouter les méthodes **doGet** et **doPost** pour traiter les requêtes HTTP.

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

#### *C'est quoi HttpServletRequest et HttpServletResponse ?*

Lorsqu'un serveur HTTP reçoit une requête, il crée deux objets qu'il transmet au conteneur de Servlets :

- **HttpServletRequest** : Contient toutes les informations relatives à la requête HTTP envoyée par le client.
- **HttpServletResponse** : Correspond à la réponse HTTP qui sera renvoyée au client par le serveur. L'objet est initialisé, mais pourra être rempli/complété lors de son passage dans la servlet avant d'être renvoyé par le serveur.

*(cf. le cours)*

N'oubliez pas d'y ajouter un constructeur qui initialise votre Service, c'est le conteneur web (Tomcat) qui se chargera d'y faire appel.

Avant de vous lancer dans l'écriture de votre Servlet, il va tout d'abord falloir configurer le fichier **WebContent/WEB-INF/web.xml** pour associer notre Servlet à une URL.



Je vous propose de supprimer le contenu du fichier **web.xml** et d'y coller ceci :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
</web-app>
```

Ajouter dans la balise `<web-app>` la définition de votre Servlet :

```
<servlet>
    <!-- Le nom de votre Servlet -->
    <servlet-name>MaServlet</servlet-name>
    <!-- Le chemin de votre Servlet dans votre application -->
    <servlet-class>ui.MaServlet</servlet-class>
</servlet>
```

On déclare l'URL à partir de laquelle notre Servlet sera joignable :

```
<servlet-mapping>
    <!-- Référence au nom de la Servlet -->
    <servlet-name>MaServlet</servlet-name>
    <!-- URL à partir de laquelle votre Servlet sera accessible -->
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

On va maintenant pouvoir nous occuper de notre Servlet, pour ce faire, je vous propose de séparer votre code dans des méthodes afin de ne pas avoir une méthode qui fait 100 lignes.

Commencez par coller ce code dans la méthode **doGet** :

```
String action = request.getServletPath();

switch (action) {
    case "/new":
        showAddForm(request, response);
        break;
    case "/insert":
        insert(request, response);
        break;
    case "/delete":
        delete(request, response);
        break;
```

```

    case "/edit":
        showEditForm(request, response);
        break;
    case "/update":
        update(request, response);
        break;
    case "/list":
        showAllFilm(request, response);
        break;
}

```

**doGet** récupère les requêtes envoyées par le client, ce code permet de faire appel à une méthode précise lorsque vous visiterez une URL :

- **insert** : Elle récupère des données écrites dans un formulaire (**FilmForm.jsp**) pour insérer un nouveau film dans la base de données.
- **update** : De la même manière que **insert** mais cette fois ci pour modifier un film.
- **delete** : Permet de supprimer un film en récupérant son identifiant.
- **showEditForm** : Affiche la JSP qui permet de modifier un film.
- **showAddForm** : Affiche la JSP qui permet d'ajouter un film.
- **showAllFilm** : Affiche la JSP qui liste tous les films.

Vous pouvez vous lancer dans l'écriture de votre Servlet ainsi que des JSP (**prenez pour exemple la JSP fournie dans le dossier WebContent/View/**). Dans la suite de ce TP, vous trouverez des éléments de code qui vous seront utiles.

- Si vous voulez mettre en place un `forEach` dans une JSP, n'oubliez pas d'ajouter la ligne suivante en haut (avant la balise `html`) de votre JSP :

```

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

```

- Récupération d'un paramètre donné dans un formulaire (par exemple dans la méthode **update**) :

```

request.getParameter("realisateur");

```

- Afficher une JSP tout en lui fournissant un attribut (un objet qu'elle peut vouloir afficher dans par exemple la méthode **showEditForm**) :

```

RequestDispatcher dispatcher = request.getRequestDispatcher("FilmForm.jsp");
request.setAttribute("film", film);
dispatcher.forward(request, response);

```

- Rediriger vers une JSP (par exemple pour rafraîchir une vue après une suppression dans la méthode **delete**)

```
response.sendRedirect("list");
```

- Rediriger vers une url dans un formulaire :

```
<form action="insert" method="post">  
...  
</form>
```

Les boucles dans les JSP :

```
<c:forEach var="film" items="${listFilm}">  
...  
</c:forEach>
```

## 4. Si vous travaillez sur VS Code ou autre

Si vous êtes ici c'est que vous ne voulez pas faire le TP sur Eclipse pour diverses raisons (besoin de comprendre le fonctionnement d'un conteneur web ou encore une préférence pour VS Code).

### a. Introduction

Avant de lancer VSCode, faites en sorte d'avoir Maven sur votre ordinateur et ajoutez-le au PATH (si ce n'est pas déjà le cas, n'ayant pas envie de me lancer dans une explication exhaustive, je vous invite à chercher par vous même ou encore mieux, à utiliser Eclipse ! :))

Rendez-vous dans le répertoire contenant le projet, de préférence via un terminal (par exemple celui de VS Code), et lancez-y la commande suivante :

```
mvn install
```

Si vous obtenez une erreur qui ressemble à cela :

```
No compiler is provided in this environment. Perhaps you are running on  
a JRE rather than a JDK?
```

Je vous invite à localiser le dossier **bin** de votre JDK (Java Development Kit), puis à copier le code suivant dans le fichier pom.xml du TP :

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.1</version>
  <configuration>
    <fork>true</fork>
    <executable>Emplacement_de_votre_dossier_bin/javac</executable>
  </configuration>
</plugin>
```

Remplacez `Emplacement_de_votre_dossier_bin` par le PATH de votre dossier bin (suivi de l'exécutable `javac` avec l'extension si vous êtes sur Windows par exemple).

Vous pouvez cette fois-ci lancer la commande `mvn install` et si tout se passe bien vous verrez un dossier **target** apparaître à la racine.

#### b. Mise en place de Tomcat et déploiement de votre App à la main

Installez Tomcat depuis le [site](#), puis faites en sorte d'avoir les deux variables d'environnements **JAVA\_HOME** et **JRE\_HOME** définies dans votre ordinateur. Si vous êtes sous un Linux qui ne sort pas trop de l'ordinaire, allez voir du côté du **.bashrc** dans votre homedir. Si vous êtes sous Windows, ouvrez un terminal (windows+r puis saisissez « cmd » et validez) et définissez les deux variables à coup de SET. Si vous êtes sur Mac, je ne sais pas mais un coup d'œil sur Google peut vous aider.

Exemple sur Windows :

```
C:\Users\Bensa\Downloads\apache-tomcat-8.5.38\apache-tomcat-8.5.38\bin>SET
JAVA_HOME=C:\Program Files\Java\jdk1.8.0_201\bin

C:\Users\Bensa\Downloads\apache-tomcat-8.5.38\apache-tomcat-8.5.38\bin>SET
JRE_HOME=C:\Program Files\Java\jdk1.8.0_201\jre
```

Vous pouvez maintenant lancer le conteneur web Tomcat via :

sur Windows :

```
startup.bat
```

sur Linux/Mac :

```
./startup.sh
```

Si vous n'avez pas défini d'utilisateur, faites-le dans `conf/tomcat-users.xml`. Le contenu de ce fichier devrait ressembler à ce qui suit :

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

Retournez dans le dossier **target**, dans lequel vous trouverez un fichier portant l'extension **.war**. Copiez-le dans le dossier **webapps** de Tomcat, puis rendez-vous à l'aide de votre navigateur préféré sur <http://localhost:8080/>, puis cliquez sur **Manager App** et vous devriez trouver votre application dans la liste.

### c. Mise en place de Tomcat et déploiement de votre App sur VS Code

Téléchargez l'extension **Tomcat For Java**. Vous verrez alors un onglet **Tomcat Servers** apparaître en bas de l'interface VS Code. Ajoutez le dossier du Tomcat que vous avez téléchargé (en appuyant sur le +).

Lancez la commande :

```
mvn install
```

Dans le dossier **target**, vous verrez apparaître un dossier portant le nom de votre application suivi d'un numéro de version. Faites clic-droit puis sélectionnez **Run on Tomcat Server**, rendez-vous sur <http://localhost:8080/> et vous verrez alors un dossier avec le nom de votre application.

---

Vous pouvez aller consulter ce site si vous voulez des informations complémentaires concernant l'installation/configuration d'Eclipse ou de Tomcat : <https://zestedesavoir.com/tutoriels/591/creez-votre-application-web-avec-java-ee/214-les-bases-du-java-ee/1304-outils-et-environnement-de-developpement/#2-4221-le-serveur-tomcat>