# « Démarrer avec maven7 »

# Table des matières

I- **Introduction:**

Maven est un programme (utilitaire) qui connait tout sur l'environnement de la production des logiciels. Il permet principalement d'automatiser le cycle de vie des projets indépendamment de l'EDI (Eclipse, Netbean, IntelJ).

On peut souligner les trois points suivants :

- La gestion du cycle de vie d'un projet.
- La gestion des dépendances
- L'automatisation des taches de management des projets.

Les composants essentiels de cet environnement sont comme suit:

1- Le fichier POM.XML

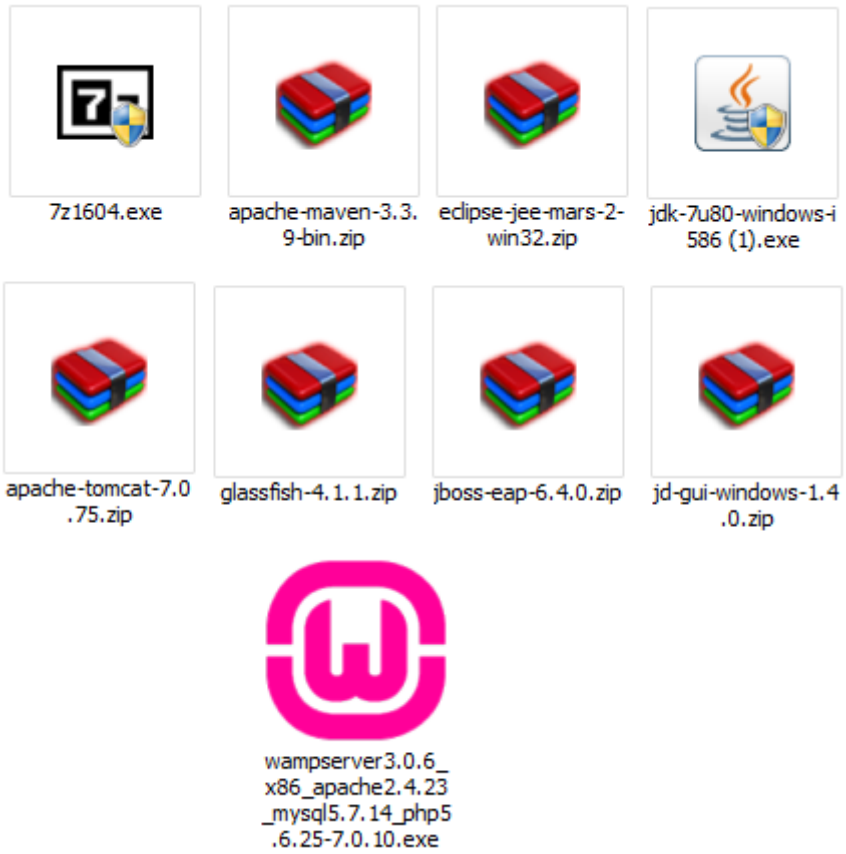
2- Un ensemble des conventions (normes et spécifications)

3- Un ensemble des commandes pour exécuter les différentes phases du cycle de vie d'un projet (Au lieu d'une seule commande : JAVAC sous windowsX86)

Ce document sera consacré à la version 7 de maven.

**II-Outils :**

Pour pouvoir travailler avec maven, il est indispensable de procurer la liste des produits en bas avec les versions indiquées :

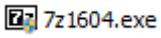



A l'aide de Google on peut facilement trouver ces produits en téléchargement gratuit.

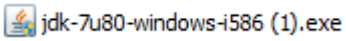
III- **Préparation de l'environnement**

a- Installation de JDK (Machine virtuel) :

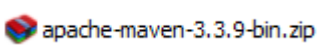
Dans un premier temps installer l'utilitaire ZIP :




Et puis Installer le JDK dans le répertoire suivant :
C:\Program Files (x86)\Java\jdk1.7.0_80
A l'aider du fichier .exe :

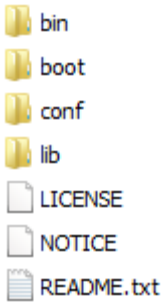



b- Créer le dossier :
C:\FormationJavaFramworks\env
Et décompresser le fichier :

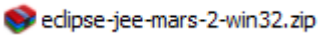



Vous obtiendrai ainsi le dossier:

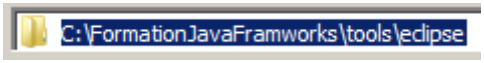

Avec les fichiers comme suit :



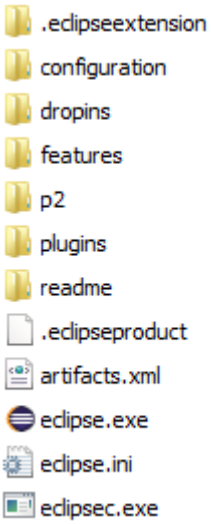

c- Créer le dossier :
C:\FormationJavaFramworks\tools\eclipse
Et décompresser le fichier :



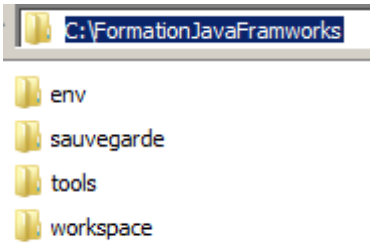

Vous obtiendrai ainsi le dossier:




Avec les fichiers comme suit :




d- Créer les dossiers :
C:\FormationJavaFramworks\workspace
Et
C:\FormationJavaFramworks\sauvegarde
On obtiendra ainsi l'arborescence suivant :
Dans le dossier :




e- Variables d'environnement :

Il est nécessaire de configurer cet environnement comme suit :

1- Click droit sur Ordinateur et choisir Propriété, puis Paramètres système avancés, puis variables d'environnement :
2- Créer les deux variables :
- JAVA_HOME : C:\Program Files (x86)\Java\jdk1.7.0_80
- MAVEN_HOME : C:\FormationJavaFramworks\env\apache-maven-3.3.9
3- Et modifier le variable :
Path: ;%JAVA_HOME%\bin;%MAVEN_HOME%\bin
%NON_VARIABLE%\bin

f- TESTER :

Avec les commandes dos : java –version et Mvn -version

**IV-Ateliers-01 :**

1- Gestion des dépendances :

Maven permet de gérer les dépendances d'autres projets qu'on utilisera dans notre projet principal, grâce à la notion du plugin voir Dependency.
Le but des pluging  est d'enrichir le cycle de vie du projet :
La bonne pratique :
<Source> V1 </Source>
<Target> V2 </Target>
V1=V2 : la même version Source et Target

Cette Résolution de dépendance s'avère transitive : elle doit être définie au niveau du POM :
Dans la situation ou on a plusieurs projets :
Un projet principal P et deux projets X et Y.
On peut admettre que si P→X Et X→Y alors P→Y

Par exemple la dépendance de X a Y doit être ajoutée dans le POM de X comme suit :

POM de Y :

```
<project>

<modelVersion>4.0.0</modelVersion>
<groupId>com.regrag.formationframworks</groupId>

<artifactId>Y</artifactId>

<version>0.0.1</version>

<packaging>pom</packaging>
</project>
```

POM de X :

```
<project>

<modelVersion>4.0.0</modelVersion>
<groupId>com.regrag.formationframworks</groupId>

<artifactId>X</artifactId>

<version>0.0.1</version>

<dependencies>

<dependency>

<groupId>com.regrag.formationframworks</groupId>

<artifactId>Y</artifactId>

<version>0.0.1</version>
<scope>provided</scope>

</dependency>


</dependencies>


</project>
```

POM de P :

```
<project>

<modelVersion>4.0.0</modelVersion>
<groupId>com.regrag.formationframworks</groupId>

<artifactId>P</artifactId>

<version>0.0.1</version>

<dependencies>

<dependency>

<groupId>com.regrag.formationframworks</groupId>

<artifactId>Y</artifactId>

<version>0.0.1</version>


</dependency>


</dependencies>


</project>
```

**REMARQUE** :

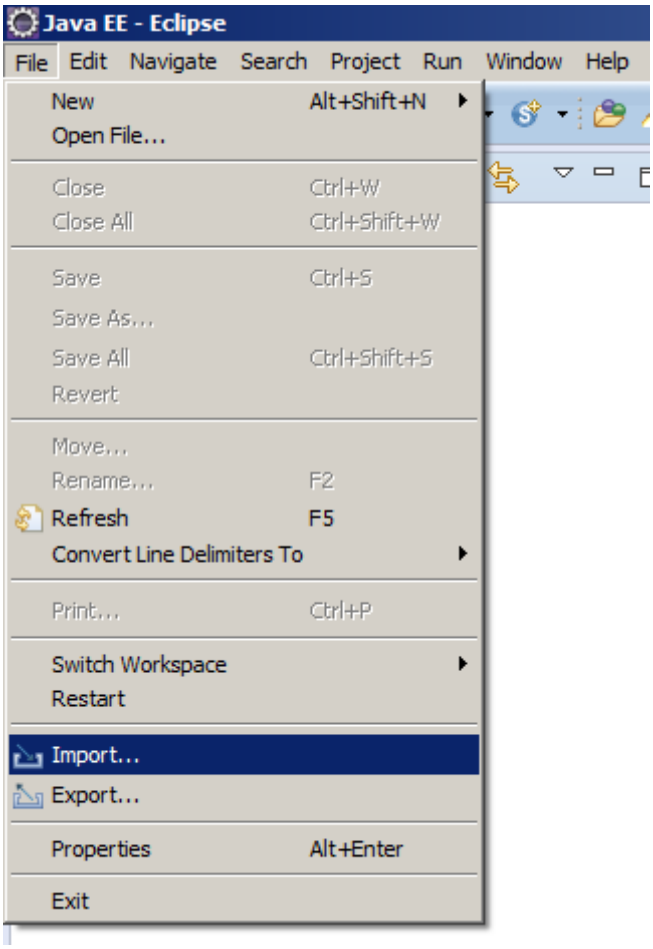
- La balise SCOPE : elle est facultative, elle permet de définir la portée des dépendances cad : Définir quel phase du cycle du projet on utilisera la dépendance.
- A l'exécution d'une commande compilation, Maven cherche X/X/1.0.0/JAR dans m2/rep s'il ne le trouve pas il va chercher sur INTERNET (sur les serveurs dédiés).

**TP0 :**

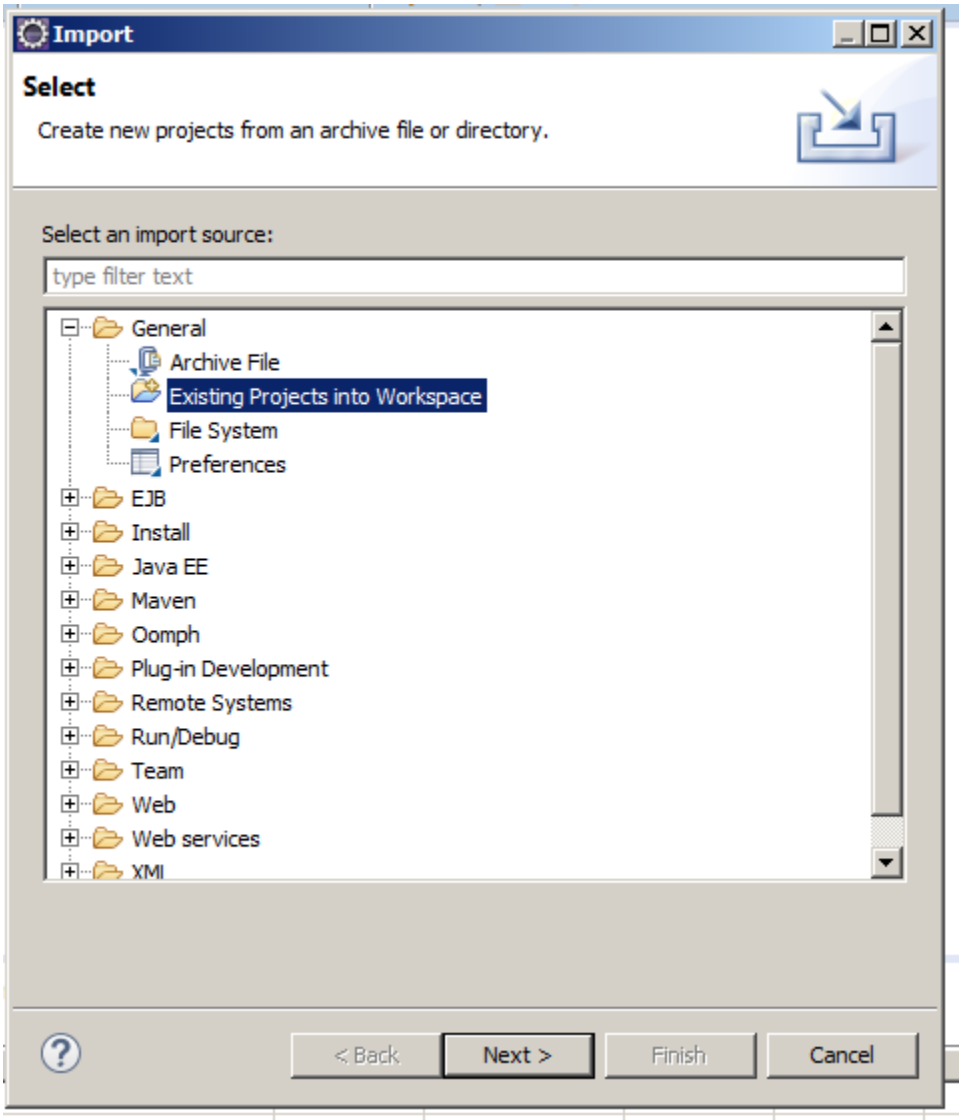
Comment importer les projets exemples de nos ateliers :

On prendra comme exemple le premier module Atelier-01

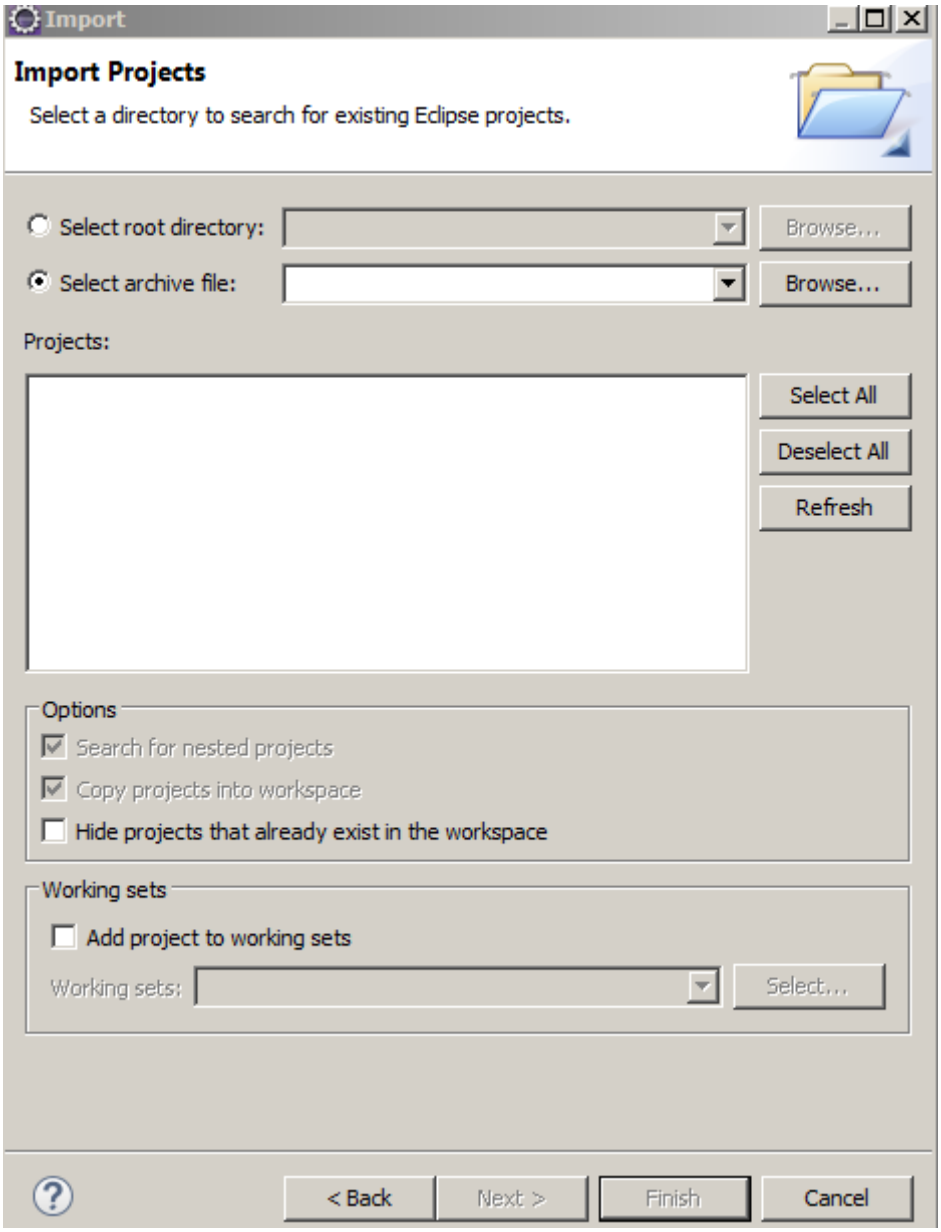
1- Dans la barre de menu choisir :
File→Import…



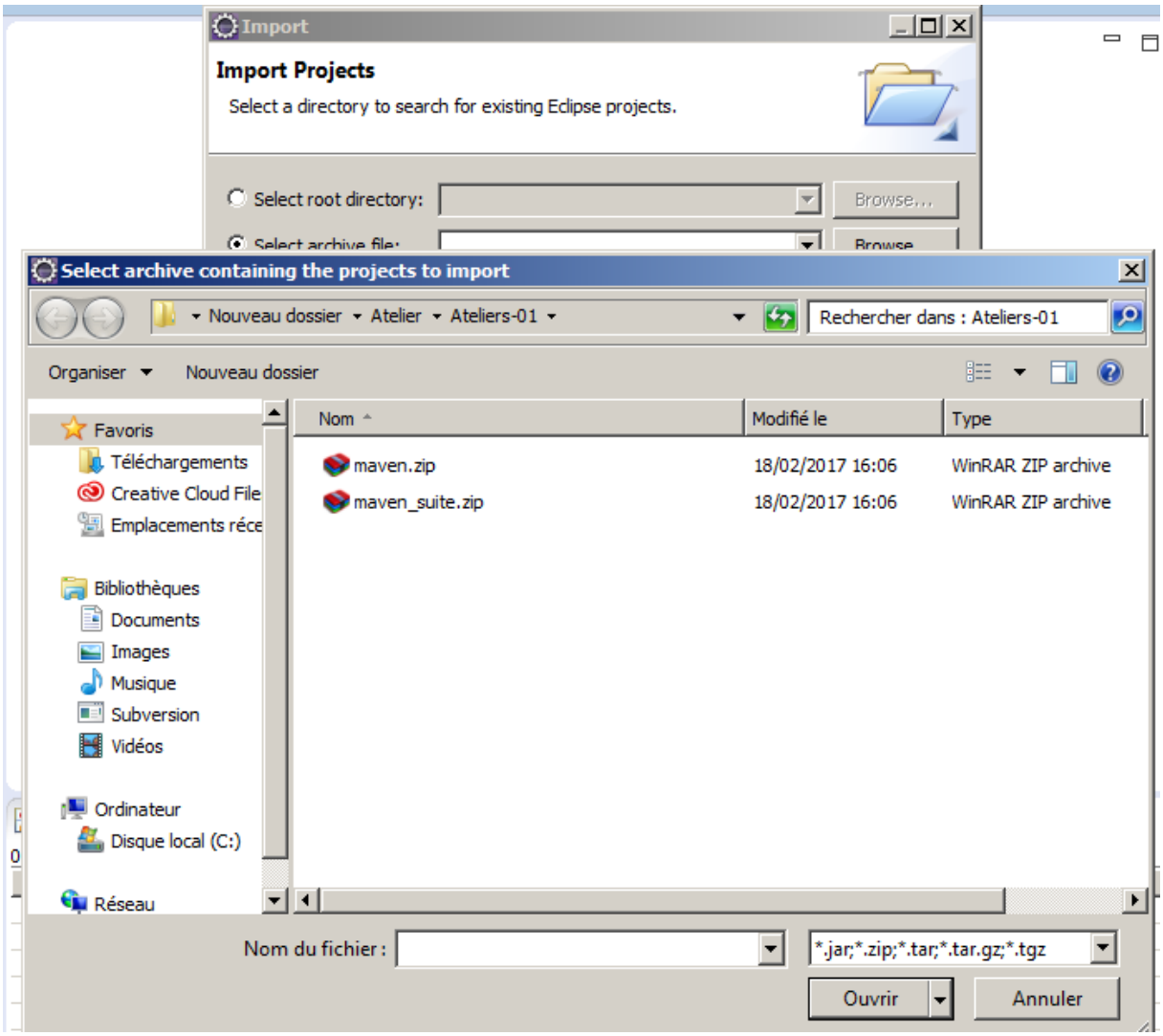
2- Selectionnez : General→Existing Projects into Workspace



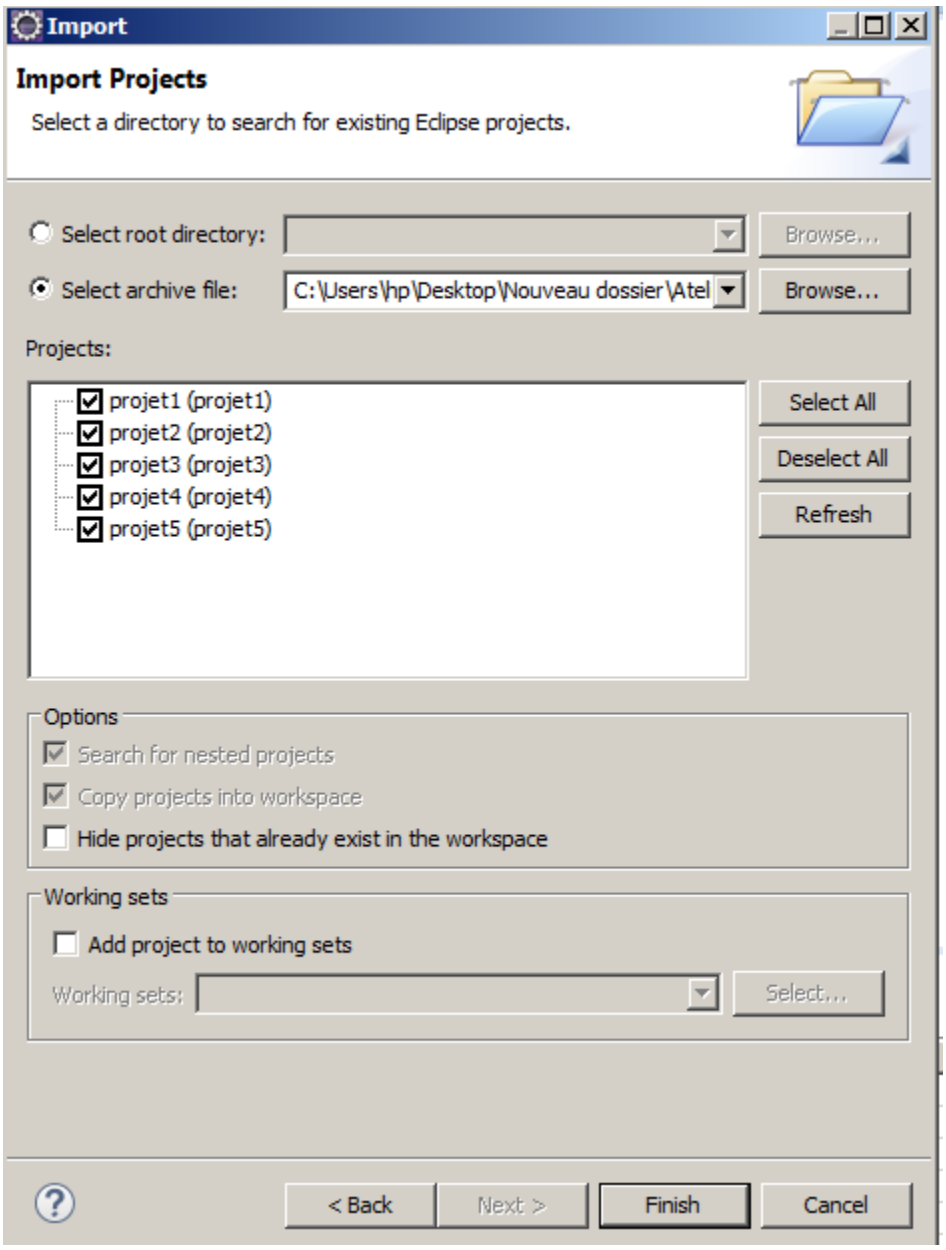
3- Cocher l'option : Select archive file et appuyez sur le bouton Browse :



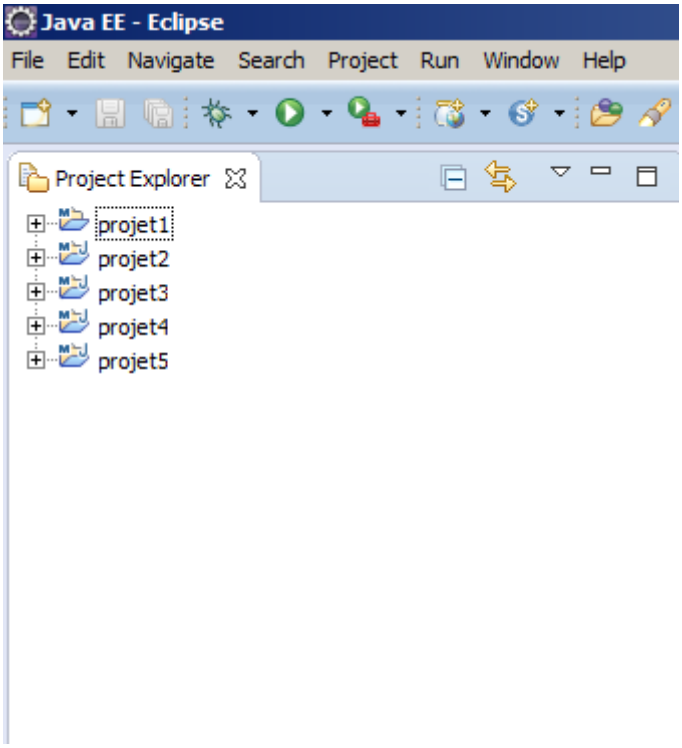
4- Dans la fenêtre de l'explorateur Windows sélectionnez le fichier .ZIP à importer :



5- Appuyez sur le bouton Finish pour lancer l'importation du projet



6- Après l'importation dans le volet Explorateur vous trouverez les dossiers des projets que contient ce fichier comme suit :



**TP1 :**

1- Allez en projet2 et supprimer Target.
2- Exécutez la commande *mvn clean* : le résultat est rien ne change
3- Et puis la commande *mvn validate* : rien ne change
4- Aussi la commande *mvn compile* : cette fois le programme crée Target
5- Supprimer Target et Exécutez la commande *mvn test* : le programme crée Target sans package (.jar)
6- La commande *mvn package* : crée le package .jar dans Target
7- Et la commande *mvn Déploy* :
   - crée dans .m2 projet2
   - et dans Target crée tout

- Dans le déploiement il doit y avoir un serveur.
- Les commandes maven les plus utilisées sont :
*Clean Install, Clean Deploy, Clean Test*.
- La commande *Site: Site* sert à générer la documentation maven de votre site.

**TP2 :**

a- Dans workspaces/maven/projet2/src/man/java
Créer le dossier :
Com/regrag/frameworks/maven
Et
Créer le fichier :
Maclasse et écrire la classe :

```
package com.regrag.frameworks.maven;

public class MaClasse {

private int a;

public void maMethode()
{

System.out.print("Hello java");

}

}
```

b- Dans le POM du projet2 tapez la commande :
*mvn verify*
Ou
Dans la console au niveau de dossier c:/formationjavaframework/workspace/maven/projet2
Tapez *mvn clean* et puis *mvn verify*

c- Manipulation du pluging dependency :

**Le pluging dépendancy :**

Soit le schéma si dessous :

Projet2→projet3 et projet3→projet1

a- Allez en projet3 et tapez la commande *mvn install*
b- Allez au projet1 (POM/xml) :
   1-ajoutez <Scope>provided</Scope>
   2-ajoutez <Packaging>POM</Packaging>

**REMARQUE** :

Dans le POM du projet1 il doit y avoir
<Packaging>POM</Packaging>
Et non
<Packaging>JAR</Packaging>
Pour que la compilation passe.

2- Gestion des Profils :

Avec maven on peut gérer un seul dossier et deux versions différentes.

Exemples :

Projet5→Projet4 V1.0.0

Et

Projet5→Projet4 V1.0.1

Dans C:\FormationJavaFramworks\workspace\maven allez dans projet5 :

- click droit
- choisir Maven Build…
- exécutez : Clean Install

a- pour compiler avec un profil

Tapez la commande :

*clean verify –X –Pprofil1*

b- pour activer un profil par default :

Écrire dans la balise profil :

<activation>

<activeByDefault>True< /activeByDefault >

< /activation >

3- Génération du site du projet :

Allez dans le projet2 et exécutez la commande :

Site : Site

**IV-Ateliers-02 :**

**TP0 :**

De la même manière que précédemment exécutez :

1- import→General (pour importer des projets qui existent)
2- Existing projects into workspace
3- Cherchez le fichier 'Ateliers-02.zip'
4- Appuyez sur « FINISH»

1-JUNIT :

**TP1 :**

Allez dans Projet6 et POM.XML
Faire commentez la balise <repositories> (par strg + flechehaut + C)

```
<!--   <repositories> -->
<!--         <repository> -->
<!--               <id>redhatrepo</id> -->
<!--               <name>redhat</name> -->
<!--               <url>https://maven.repository.redhat.com/ga/</url> -->
<!--         </repository> -->
<!--   </repositories> -->
```

Exécutez la commande *Clean install* :
Et lisez l'erreur.
Pour éliminer l'erreur décommentez la balise <repositories> (par strg + flechehaut + C)

```
<repositories>
      <repository>
            <id>redhatrepo</id>
            <name>redhat</name>
            <url>https://maven.repository.redhat.com/ga/</url>
      </repository>
</repositories>
```

Et
Exécutez la commande Clean install :
Il n'y a plus d'erreur.

**Prérequis sur le code :**

Les classes Test sont standardisées, et étend  la classe mère junit.framework.TestCase :

- Ne contient pas de main
- Contient des méthodes testXXXXXX()
- Chargées automatiquement
- Les méthodes doivent contenir des assertions :

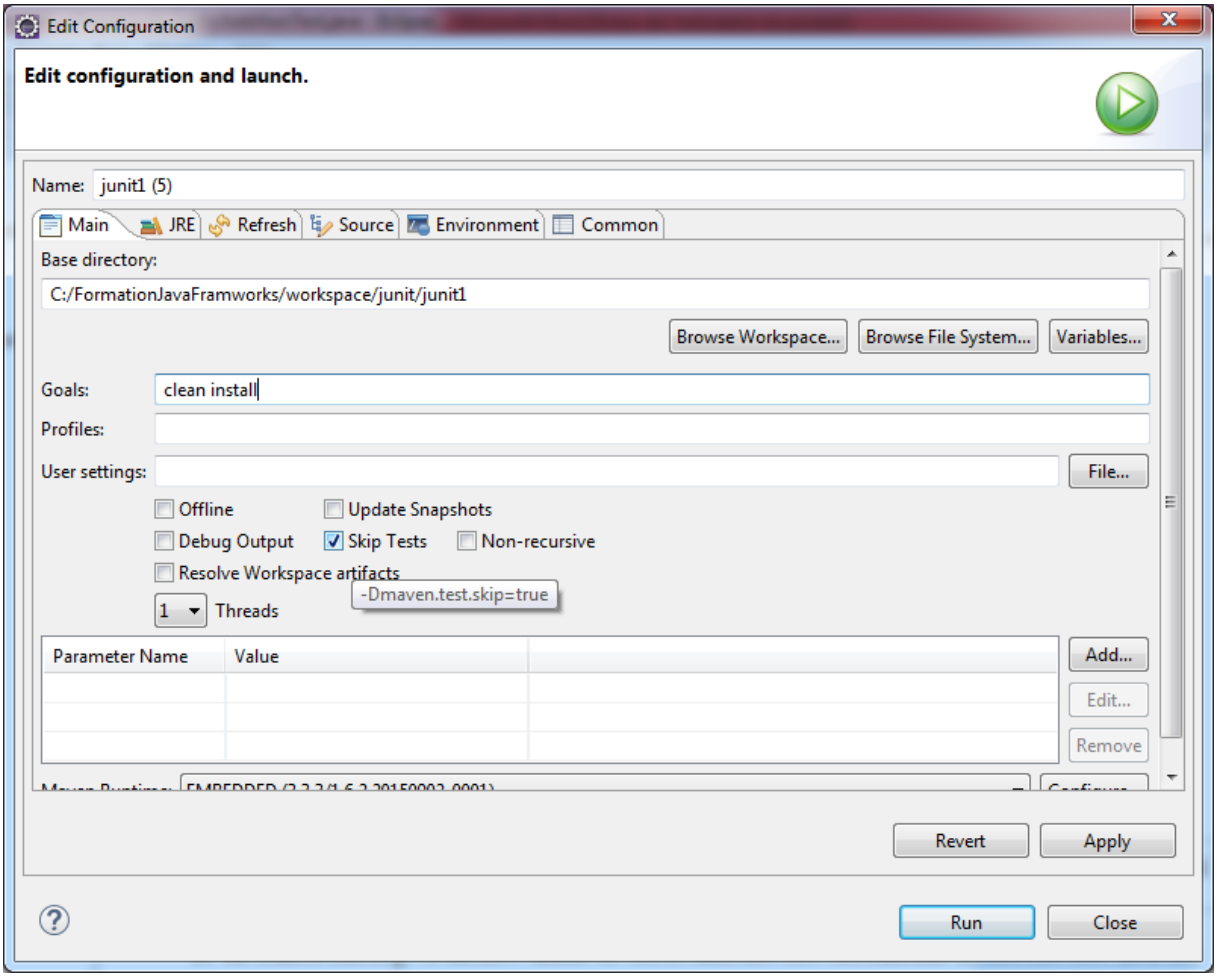
Mot clé Assert : mot réservé en java, qui permet de vérifier si une condition boolean est vraie :

- il faut activer les assertions avec assertTrue(bool condition) et assertEquals(int a, int b)
- fail() : test ne passe pas.

**TP2 :**

Allez dans le projet  Junit1 :

- J'ai créé la classe Addition.java dans src/main/java :
  Et j'ai écrit la méthode calculer()
- J'ai créé la classe AdditionTest.java dans src/test/java :
  Et j'ai écrit trois méthodes :
  1- testCalculer()
  2- testCalculer3()
  3- testCalculer2()
- Exécutez Clean install avec « Skip Tests » coché.




- Si on laisse « Skip Tests » décoché
  ET on exécutez *clean install*: les tests seront effectués.

Et on aura le message d'erreur : Mon message d'erreur qu'on a ajouté à la fonction de

test testCalculer2() :

```
public void testCalculer2() {
            Addition addition = new Addition();

            int resultat = addition.calculer(3, 5);
            assertTrue("Mon message d'erreur",resultat == 10);

      }
```

En sortie :

```
Failed tests:   testCalculer2(ma.regrag.formationframeworks.tests.AdditionTest):
Mon message d'erreur

Tests run: 3, Failures: 1, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
```

**REMARQUE :**

a-Pour voir le code de la fonction assertTrue() :

Sélectionnez assertTrue  et cliquez sur F3

```
    public void testCalculer3() {
        Addition addition = new Addition();

        int resultat = addition.calculer(5, 5);
        assertTrue(resultat == 10);

    }
```

+ F3

b-Utiliser Google pour voir le site de Junit :

Milestone : Version test produite pour les développeurs.

**Dans Junit2: version 4**

-Dans la classe Addition.java , ajouter la méthode suivante :

```
public void methodeException() throws Exception
{
throw new Exception("mon excepttion");
}
```

-Dans la classe Testaddition.java , faire appel à cette exception comme suit:

```
@Test(expected=Exception.class)
       public void abc() throws Exception{
              Addition addition = new Addition();
         addition.methodeException();
       }
```

c- click droit sur junit2 et run as → Juint Test

2- Filtres :

**TP2 :**

La déclaration d'un filtre : fichier de traduction.
- Nom du fichier =Fichier de traduction
- Variable= Valeur

`Allez en projet7 :`

a- Dans POM .XML : on déclare la balise filter comme suit :

`<filter>src/main/filters/monFiltre2.properties</filter>` et à chaque fois on passe l'un des deux fichies : `src/main/filters/monFiltre2.properties` et `src/main/filters/monFiltre.properties dans son contenue.`

**POM.XML :**

```
<groupId>ma.regrag.formationframeworks</groupId>
<artifactId>projet7</artifactId>
<version>0.0.1-SNAPSHOT</version>

 <build>

	<filters>
		<filter>src/main/filters/monFiltre2.properties</filter>
	</filters>

	<resources>
		<resource>
			<directory>src/main/resources</directory>
			<filtering>true</filtering>
		</resource>
	</resources>

</build>
```

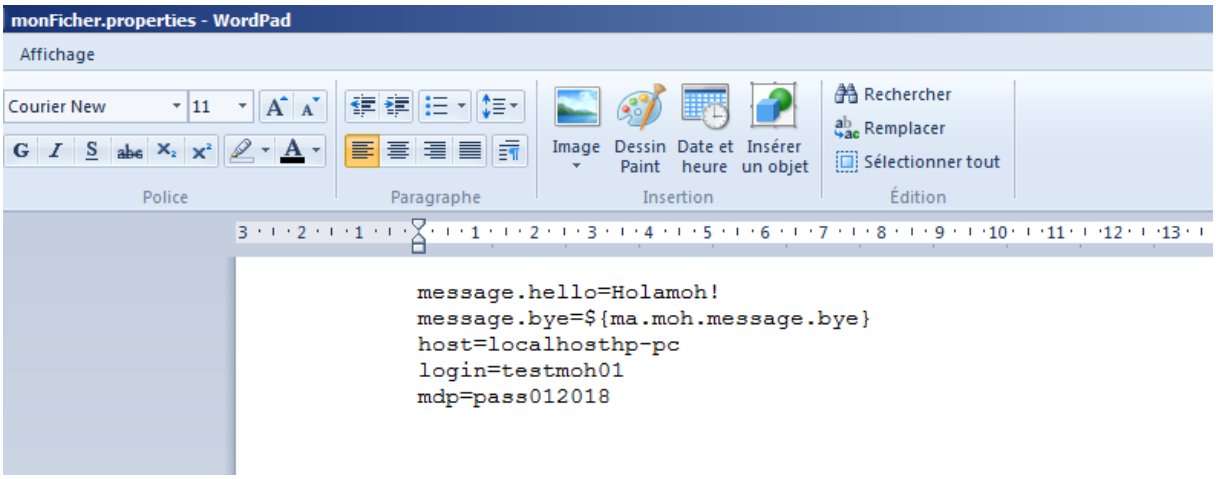
Et on exécute à chaque fois la commande : Clean install :

Le résultat du filtre se trouve dans le fichier :

..target\projet7-0.0.1-SNAPSHOT.jar→monFicher.properties

Comme suit :

```
message.hello=Hola!
message.bye=${ma.moh.message.bye}
host=localhost
login=test
mdp=pass
```




b- Etablir le filtre :

Trois fichiers à établir :

b1- Le fichier src\main\filters\monFiltre.properties

(*Variable= Valeur*)

```
ma.moh.message.hello=Bonjour!
ma. moh.bd.host=localhost2
ma. moh.bd.login=test2
ma. moh.bd.mdp=pass2
```

b2- Le fichier src\main\filters\monFiltre2.properties

(*Variable= Valeur*)

```
ma. moh.message.hello=Hola!
ma. moh.bd.host=localhost
ma. moh.bd.login=test
ma. moh.bd.mdp=pass
```

b3- src/main/resources/monFicher.properties (*Nom du fichier =Fichier de traduction*)

```
message.hello=${ma.moh.message.hello}
message.bye=${ma.moh.message.bye}
host=${ma.moh.bd.host}
login=${ma.moh.bd.login}
mdp=${ma.moh.bd.mdp}
```

**REMARQUE :**

Dans le fichier résultat on trouve :
message.bye=${ma.moh.message.bye}

Et cela parce que :

la variable « bye » n’a pas de valeur dans le fichier filtre « monFiltre.properties », même s’elle est déclaré dans le fichier « monFicher.properties ».

3- Héritage :

**TP3 : Héritage Parent**

a- Allez dans projet8

Dans le POM.XML :

Pour faire référence à la version Parent dans le projet8 :

- on ajoute la balise `<dependencyManagement>` ou on spécifie la version du projet Parent « Projet8 » dans la balise `<version>2.3</version>`.
- On spécifie la valeur de packaging 'pom' dans la balise `<packaging>.`
- On exécute : *clean install* sur POM projet8 :
  On remarque que le dossier target est vide car le packaging est pom et non jar.

**POM.XML :**

```
<groupId>ma.regrag.formationframeworks</groupId>

      <artifactId>projet8</artifactId>
      <packaging>pom</packaging>
      <version>0.0.1-SNAPSHOT</version>

      <name>Projet Parent</name>

      <dependencies>
            <dependency>

                  <groupId>junit</groupId>
                  <artifactId>junit</artifactId>
                  <version>3.8.1</version>
                  <scope>test</scope>

            </dependency>
      </dependencies>

      <dependencyManagement>

            <dependencies>

                  <dependency>

                        <groupId>commons-lang</groupId>
                        <artifactId>commons-lang</artifactId>
                        <version>2.3</version>

                  </dependency>

            </dependencies>

      </dependencyManagement>
```

Ce qu'on appelle les fiches bom , bills of matériels = POM + Dependency `Management.`

b- Allez dans projet8_Enfant

Dans le POM.XML

- Ajouter la balise 'parent' Pour faire référence au projet parent projet8.

```
<parent>
            <groupId>ma.regrag.formationframeworks</groupId>
            <artifactId>projet8</artifactId>
            <version>0.0.1-SNAPSHOT</version>
</parent>
```

- Cette fois On spécifie la valeur de packaging 'jar' dans la balise <packaging>.
- On ajoute la balise 'Dependency' comme suit :

```
<dependencies>
      <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
            <scope>compile</scope>
      </dependency>
</dependencies>
```

- On exécute : clean Install sur POM projet8_Enfant :
  On remarque que le dossier Target n'est pas vide car le packaging est 'jar' et non 'pom'.

c- Si le cas présent un projet parent projet9 composé de plusieurs modules (projets sous forme de module enfant) :

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:
  <modelVersion>4.0.0</modelVersion>
  <groupId>ma.regrag.formationframeworks</groupId>
  <artifactId>projet9</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Projet Parent Modulaire</name>
  <description>Projet Parent Modulaire</description>
  <packaging>pom</packaging>
  <modules>
    <module>projet9_module1</module>
    <module>projet9_module2</module>
    <module>projet9_module3</module>
  </modules>
</project>
```

Il faut ajouter dans le POM de chaque projet (module) la balise Parent comme suit :

```
<parent>
<groupId>ma.regrag.formationframeworks</groupId>
<artifactId>projet9</artifactId>
<version>0.0.1-SNAPSHOT</version>
</parent>
```

Exécutez *clean install* sur le POM du projet projet9

4- Plugin :

Allez dans le projet10_plugin :

Dans le dossier /projet10_plugin/src/main/java/ma/regrag/formationframeworks , vous trouverez la classe FirstPluginMojo.java comme suit :

```
/**
 *
 * @goal monGoal
 *
 * @phase test
 */
public class FirstPluginMojo extends AbstractMojo {

    public void execute() throws MojoExecutionException {
        System.out.println("blablas");

    }

}
```

a- Exécutez *clean install*

b- Et pour exécuter le plugin (Goal) exécutez la commande suivante :

*groupId :artifactId :monGoal*

Dans notre exemple la commande sera :

*ma.regrag.formationframeworks:projet10:monGoal*

**VI-Ateliers-03 (Hibernate)**

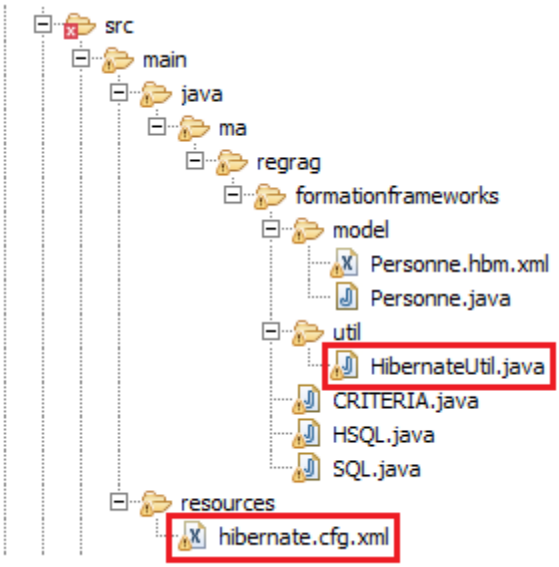
Charger les exercices : Ateliers_03.zip

**TP1 :**

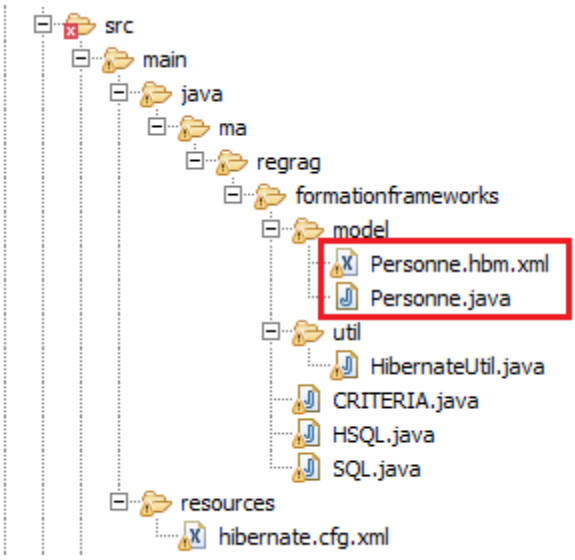
Allez dans le projet « hibernate0 »

A- Dans l'arborescence du projet on constate les fichiers suivants:

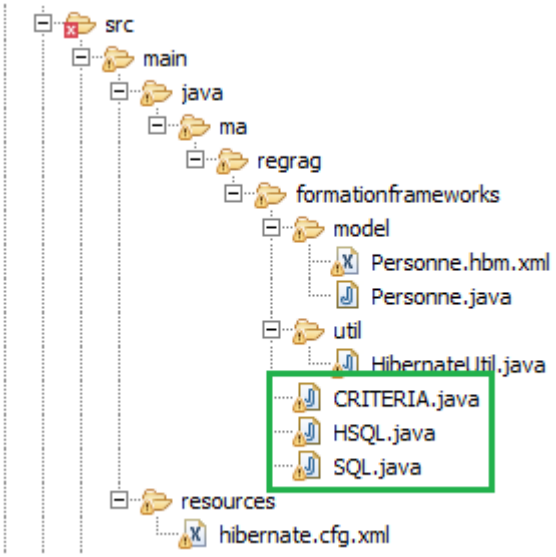
- Fichiers de configuration du Hibernate :




- Fichiers qui présentent le modèle des données :



- Fichiers qui présentent les trois manières d'exploiter Hibernate :




B- Dans POM.XML, on remarque trois balises `<dependency>` :
`org.hibernate, hsqldb : 100% base de données java, Junit.`

```
<dependencies>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core-->
	<dependency>
		<groupId>org.hibernate</groupId>
		<artifactId>hibernate-core</artifactId>
		<version>4.3.11.Final</version>
	</dependency>


<!-- https://mvnrepository.com/artifact/hsqldb/hsqldb -->
	<dependency>
		<groupId>hsqldb</groupId>
		<artifactId>hsqldb</artifactId>
		<version>1.8.0.10</version>
	</dependency>

	<dependency>
		<groupId>junit</groupId>
		<artifactId>junit</artifactId>
		<version>3.8.1</version>
		<scope>test</scope>
	</dependency>

</dependencies>
```

On remarque aussi dans la balise <build> il y a la balise <plugins> :

Qui contient le pluging « org.codehaus.mojo » (exec-maven-plugin)

```
<build>

	<plugins>
		<plugin>
			<groupId>org.codehaus.mojo</groupId>
			<artifactId>exec-maven-plugin</artifactId>
			<version>1.4.0</version>
			<configuration>

<mainClass>ma.regrag.framworks.hibernate.DemoHibernate</mainClass>
				<cleanupDaemonThreads>false</cleanupDaemonThreads>
			</configuration>
		</plugin>
	</plugins>

	<sourceDirectory>src/main/java</sourceDirectory>

	<resources>
		<resource>
			<directory>src/main/java</directory>
			<includes>
				<include>**/*.xml</include>
			</includes>
		</resource>
	</resources>

</build>
```

C- Problème de la compilation :

Le projet Hibernate0 manque quelque chose au début la balise <ressource> : ce qui affiche avec *clean install* la console suivante :

```
INFO: HHH000040: Configuration resource: /hibernate.cfg.xml
Echec création SessionFactoryorg.hibernate.HibernateException: /hibernate.cfg.xml
not found
Exception in thread "main" java.lang.ExceptionInInitializerError
	At
```

**SOLUTION :**

Ajouter dans le POM du projet : et dans la balise <build> dans la sous balise <resources>

```
<resource>
	<directory>src/main/resources</directory>
	<includes>
		<include>**/*.xml</include>
	</includes>
</resource>
```

1- Exécutez à nouveau la commande : Clean Install
2- Exécutez à nouveau les demos SQL, HSQL et CRITERIA

Le problème a disparu.

**TP2 :**

**Quelques Définition:**

- **Session** : Classe hibernate
- **SessionFactory** : Interface Hibernate
  Sa définition se trouve dans le fichier Hibernate.cfg.xml
- **Base De données** : hsqldb
- **DependencyHierachy :**
  Exporter le mapping dans la base Après commit→insert

**Description du projet Hibernate0 :**

a- Dans le POM on remarque dans la balise <build> :

```
<plugins>
<plugin>

<groupId>org.codehaus.mojo</groupId>
<artifactId>exec-maven-plugin</artifactId>
<version>1.4.0</version>

<configuration>
<mainClass>ma.regrag.formationframeworks.DemoHibernate</mainClass>
<cleanupDaemonThreads>false</cleanupDaemonThreads>
</configuration>

</plugin>
</plugins>
```

Le pluging : exec-maven-plugin :

1- Lancez une recherche google sur la chaine :

exec-maven-plugin

2- Sur POM executer Runas→Bluid…→Clean package exec :exec

Ou clean package exec :java

b- On utilisera une base de données qui ne n'exige pas un stockage physique

Mais seulement logique : temporaire (RAM)

```
<property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
```

c- Le nom de la base est spécifié dans : mem : nom_de_la_base

```
<property name="connection.url">jdbc:hsqldb:mem:firsthibernate</property>
```

d- La connexion à cette base de données 'firsthibernate' est défini dans le fichier Hibernate.cfg.xml.

e- Par défaut on aura le login : Sa avec le mot de passe vide :

```
<property name="connection.username">sa</property>
<property name="connection.password"></property>
```

f- Pool de connection :

```
<!-- Pool de connection (interne) -->
<property name="connection.pool_size">1</property>
```

g- Dans le POM du projet on doit spécifier la dépendance à la base :

```
<!-- https://mvnrepository.com/artifact/hsqldb/hsqldb -->
            <dependency>
                  <groupId>hsqldb</groupId>
                  <artifactId>hsqldb</artifactId>
                  <version>1.8.0.10</version>
            </dependency>
```

Dans la balise <dependencies>.

h- Le Dialecte :

```
<!-- SQL dialect -->
<property name="dialect">org.hibernate.dialect.HSQLDialect</property>
```

i- Créer la base de données au démarrage de l'application:

```
<!-- Supprimer et re-créée le schéma de base de données au démarrage -->
<property name="hbm2ddl.auto">create</property>
```

j- Spécification du fichier du mapping :

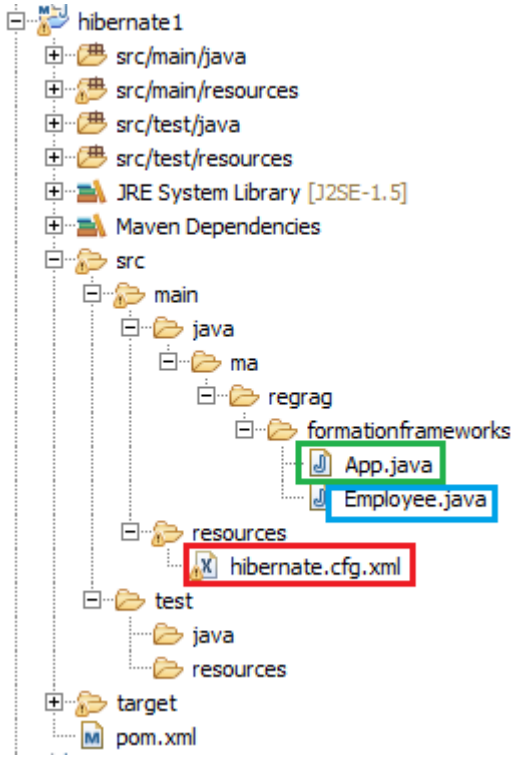
```
<mapping resource="ma/regrag/formationframeworks/model/Personne.hbm.xml" />
```

Qui correspond à la table Personnes.

**TP3:**

Dans l'arborescence du projet on distingue les fichiers suivants :

Employee.java : représente le modèle de donnée
Hibernate.cfg.xml : la configuration requise pour Hibernate
App.java : demo d'utilisation d'Hibernate




Le but et de basculer de la base hsqldb vers la base com.h2database :

1- Dans POM Hibernate.cfg.xml et POM.xml faire passer en commentaire

Le bloc relatif à la base hsqldb

```
<!--<property name="connection.driver_class">org.hsqldb.jdbcDriver</property> -->
<!--<property name="connection.url">jdbc:hsqldb:mem:firsthibernate1</property> -->
<!--<property name="dialect">org.hibernate.dialect.HSQLDialect</property> -->
```

Et

```
<!--         <dependency> -->
<!--               <groupId>hsqldb</groupId> -->
<!--               <artifactId>hsqldb</artifactId> -->
<!--               <version>1.8.0.10</version> -->
<!--         </dependency> -->
```

2- Et faire décommenter les blocs relatifs à la base com.h2database

```
<property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
<property name="hibernate.connection.driver_class">org.h2.Driver</property>
<property name="hibernate.connection.url">jdbc:h2:mem:test</property>
```

et

```
<!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
		<dependency>
		<groupId>com.h2database</groupId>
		<artifactId>h2</artifactId>
		<version>1.4.192</version>
		</dependency>
```

3- Clic droit et chosir run→ clean install
4- Clic droit →maven→Updateproject
5- Clic droit sur Hibernate1 runas→java application →App

**REMARQUES :**

a- pas de fichier hbm.xml
b- Pas de fichier HibernateUtil.java
c- Pas de balise <Build>
d- Pas de dépendance Junit
e- Fichier App.java et fichier Employee.java
f- Dans le fichier Employee.java on Remarque : existence des annotations

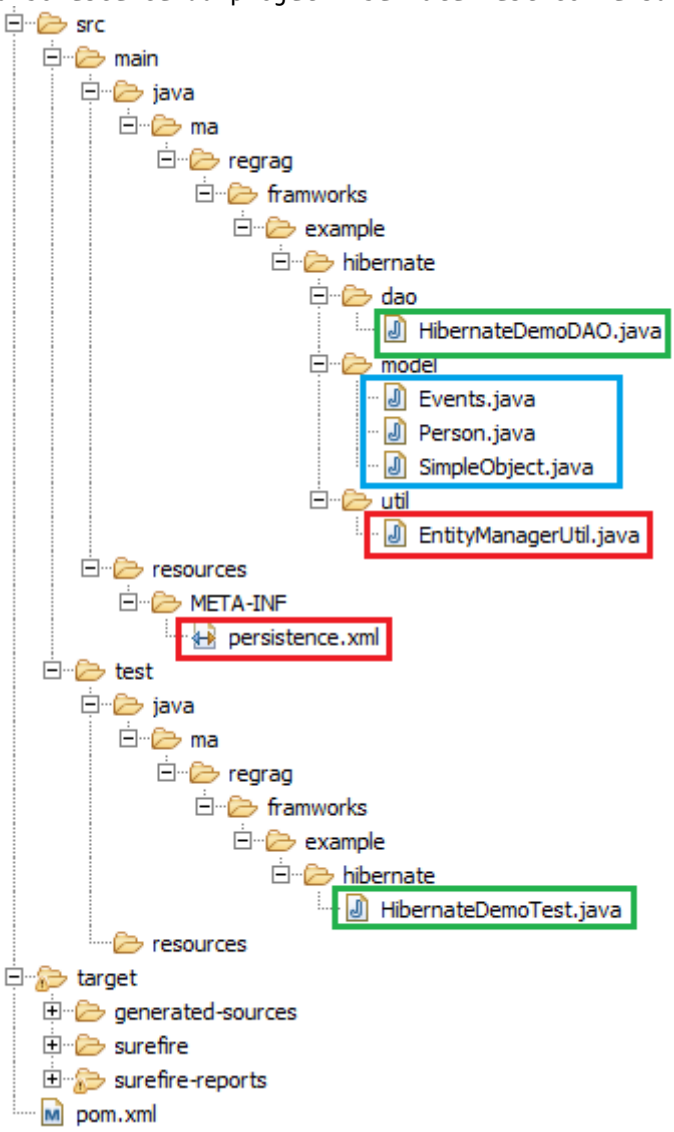
```
@Entity
@Table(name = "employee")
public class Employee {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column
	private Integer id;

	@Column(name = "name")
	private String name;

	@Column(name="age")
	private Integer age;

	public Employee() {
	}
….

	@Override
	public String toString() {
		return "Employee: " + this.id + ", " + this.name + ", " + this.age;
	}
```

g- Dans le fichier APP.java :

L'appel de la fonction Create() implique automatiquement l'action save() dans La base de données.

**TP4:**

L'arborescence du projet Hibernate2 est comme suit :




**a-** On peut noter le nouveau fichier persistence.xml et l'absence des fichiers de configurations usuels (.cfg.xml)

**b-** L'utilisation de 'persistence' est équivalent a la configuration cfg.xml dans JPA : une classe <-> mapping dans File.cfg.xml.

**c-** On peut déclarer plusieurs 'persistence' avec d'autres paramètres : name, transaction-type …

Le contenu de ce fichier est comme suit :

```
<persistence-unit name="hsqldb-ds" transaction-type="RESOURCE_LOCAL">

      <description>HSQLDB Persistence Unit</description>
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <class>ma.regrag.framworks.example.hibernate.model.SimpleObject</class>
      <properties>
         <property name="javax.persistence.jdbc.driver"
value="org.hsqldb.jdbcDriver" />
         <property name="javax.persistence.jdbc.url"
value="jdbc:hsqldb:mem:demodb" />
         <property name="javax.persistence.jdbc.user" value="sa" />
         <property name="javax.persistence.jdbc.password" value="" />
         <property name="hibernate.dialect"
value="org.hibernate.dialect.HSQLDialect" />
         <property name="hibernate.hbm2ddl.auto" value="create-drop" />
         <property name="hibernate.show_sql" value="true" />
         <property name="hibernate.format_sql" value="true" />
         <property name="hibernate.transaction.flush_before_completion"
value="true" />
      </properties>

   </persistence-unit>
```

**d-** Cette approche est orienté JAVA plus que Hibernate0, 1.

**e-** JPA : seulement les interfaces.

**f-** nouveaux paramètres : dans le fichier ‘persistence.xml’

```
<property name="hibernate.format_sql" value="true" />
<property name="hibernate.transaction.flush_before_completion" value="true" />
```

*format_sql :* formater l’affichage SQL : pour qu’il soit lisible.

*flush_before_completion :* fait le travail si on n’a pas de commit : forcer le stockage

**g-** regardez le fichier : ’ EntityManagerUtil.java’

EntityManagerFactory emf = Persistence.createEntityManagerFactory("hsqldb-ds");

"hsqldb-ds" : est celui déclarer dans ‘persistence.xml’ avec :

<persistence-unit name="hsqldb-ds" transaction-type="RESOURCE_LOCAL">

Qui est celui dans POM.XML.

Au lieu d’utiliser les fichiers mappings comme avant, cette fois on utilise les annotations :

Dans cette exemple : @ManyToOne utilisé ici dans ‘Events.java’.

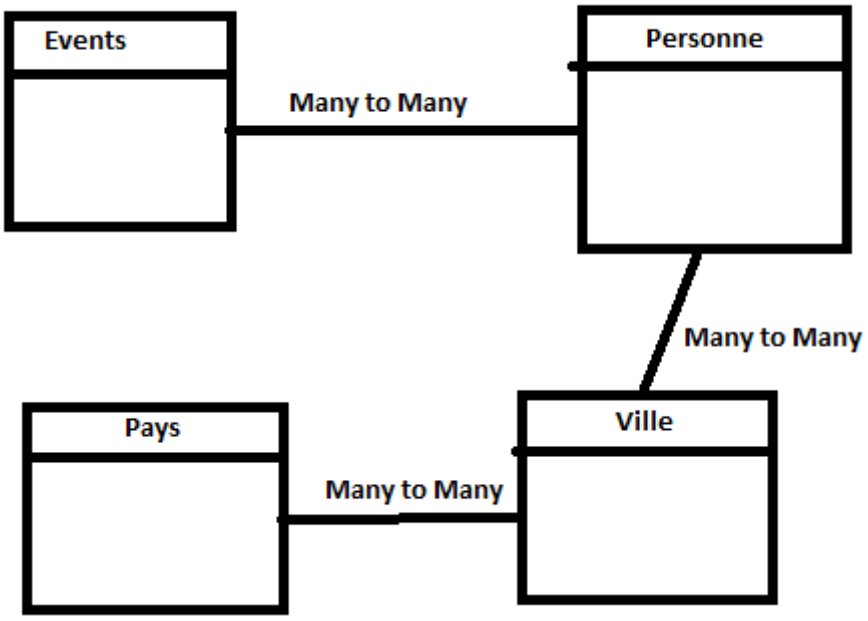
**h-** Exécution du demo Hibernate2 :

On n'a pas de classe qui contient main mais une classe test 'HibernateDemoTest.java'
Allez dans Hiberante2 et exécutez :
click droit sur Hibernate2→run as→Maven test.
Ou bien
click droit sur HibernateDemoTest.java →run as→JunitTest.

**Quelques rappels :**

- Traitement en cascade : Delete->Update->All :
  cascade type et persistence .

Le paramètre fetchtype  prend deux valeurs : EAGER /LAZY :
(EAGER : Gourmand et LAZY : Paresseux)
Généralement on fait : EAZY.
Pourquoi EAZY pas EAGER :
On suppose avoir le schéma suivant:




Si on fait appel a load (Events.class,1) :

- Avec EAGER c'est plus lent :
- Avec EAZY c'est plus rapide : à la demande :

**VII-Ateliers-04 (Spring)**

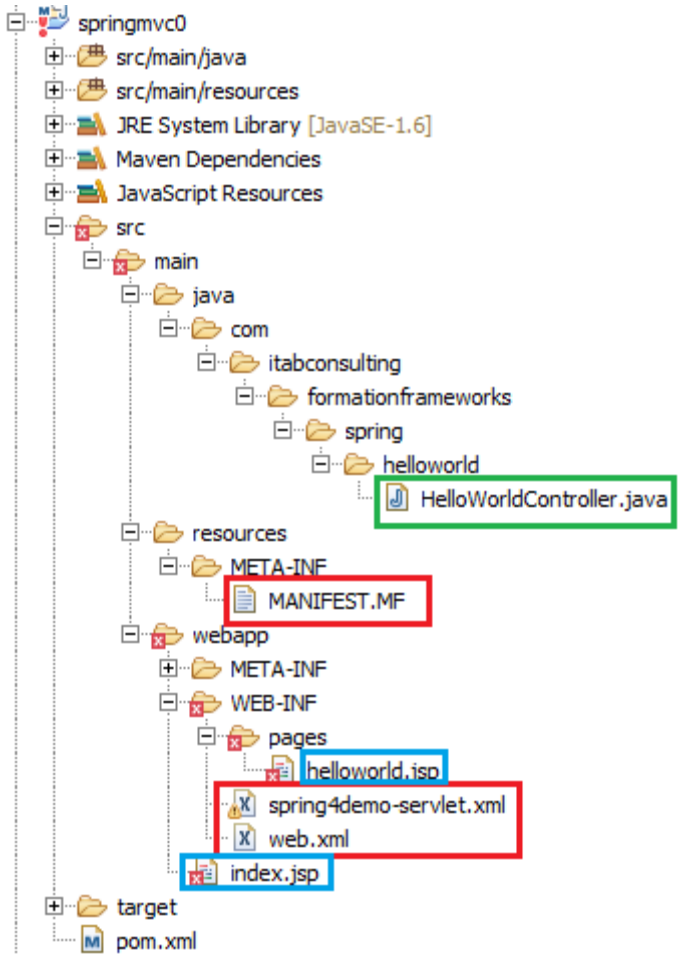
Importer dans votre workspace « Atelier-O4 » : le fichier Atelier-04.zip.

Dans le projet springmvc0 : voir dans le dossier META-INF : le fichier 'MANIFEST.MF' :

C'est le fichier descripteur de projet : là où il y a le main

**TP1 :**

Allez dans springmvc0 : c'est la version 4 du spring :




**a-** Dans POM.XML on remarque :

- la balise packaging : war <packaging>war</packaging>

Cad qu'on a besoin d'un conteneur web ou conteneur des servlets ou encore serveur web, pour l'implémentation.

Dans notre cas : on utilise Apache-Tomcat version 7.

- La dépendance : commons-loggins
- JSTL : JSP
- SpringFramework : aop,beans,context, core

**b-** On remarque qu'on a 3 dossiers : par convention

- Java
- Ressources
- Webapps : →web.xml : descripteur de JEE
  →spring4demoservlet.xml : fichier de configuration du spring

**c-** Dans le dossier ‘Java’ on voit le fichier ‘HelloWordController.java’

On remarque l’annotation dans ce fichier source:

@RequestMapping : (value=’printHello’)

Cad : si on fait appel à la méthode ‘printHello’ → elle appelle le code qui suit : return helloworld

Avec msg=dynamicmessage

**d-** Scénario :

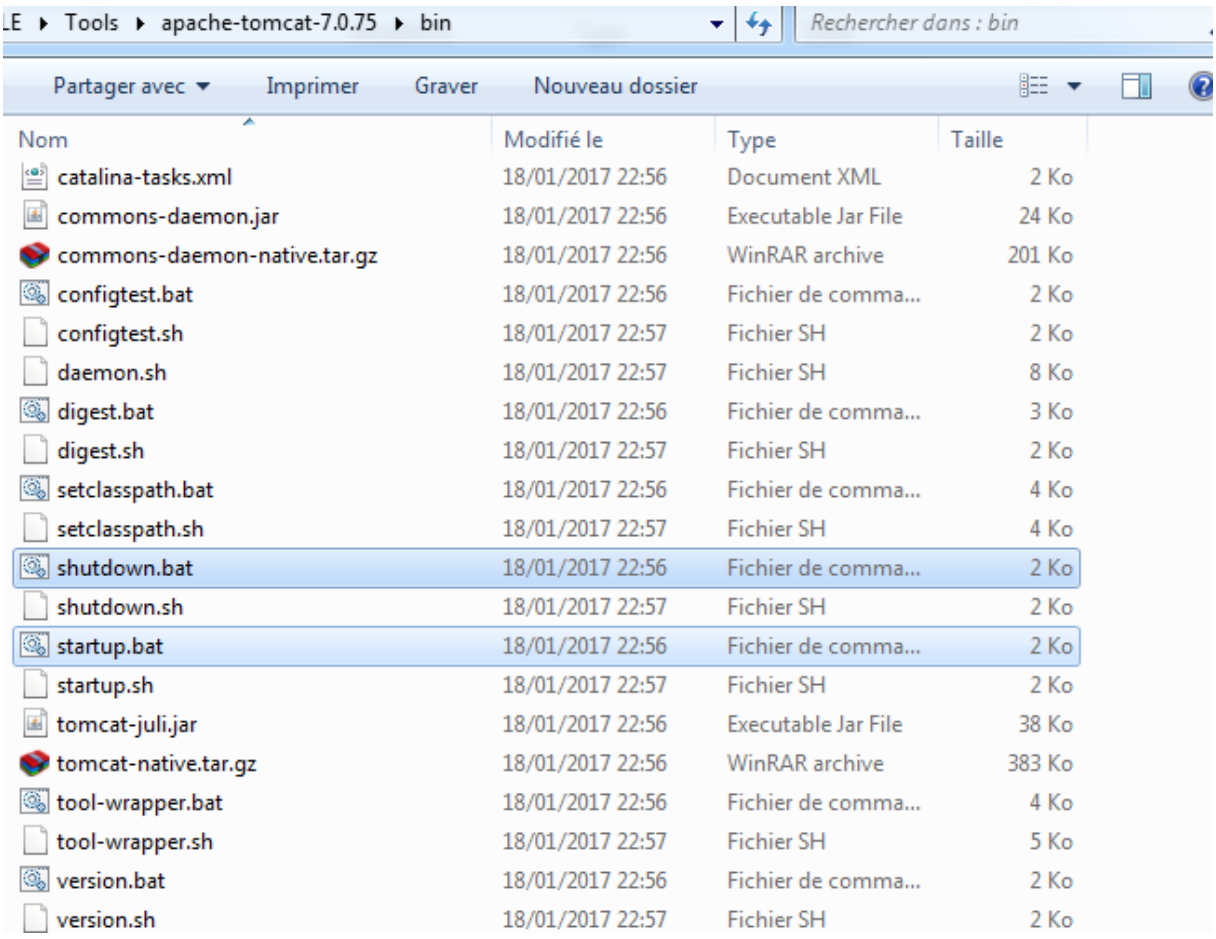
La page Index → printhello→controleur (qui prépare Model et Vue) →viewResolver→

Dossier pages +JSP+helloworld+varriable msg = dynamicmessage

**e-** Comment faire fonctionner le projet : springmvc0 ?

1- Faire extraire le fichier ZIP ‘apache-tomcat-7.0.75.zip’ dans un dossier ‘apache-tomcat-7.0.75’

2- Accéder au dossier ‘apache-tomcat-7.0.75’ sur le dossier ‘bin’ : Click droit →Ouvrir dans une nouvelle fenêtre.

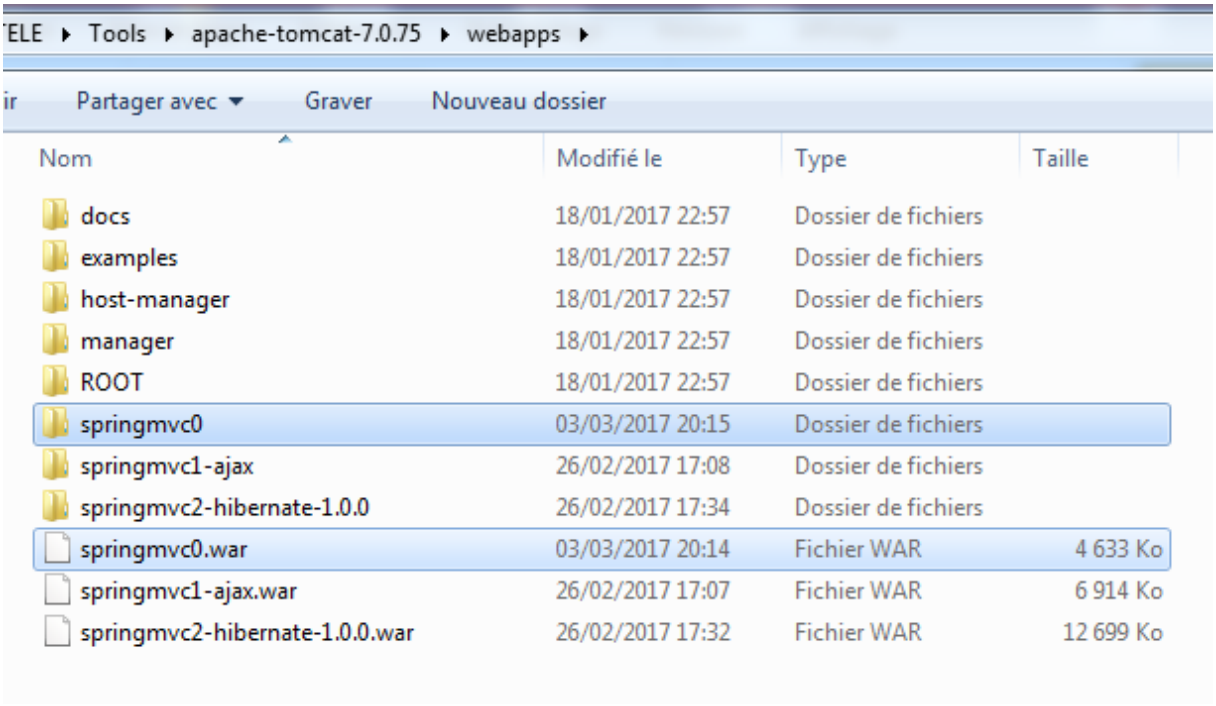



- Les autres dossiers : ‘conf’, ‘lib’ et ’ webapps’
- Exécuter le fichier .bat ‘Startup.bat’ : pour démarrer APPACHE

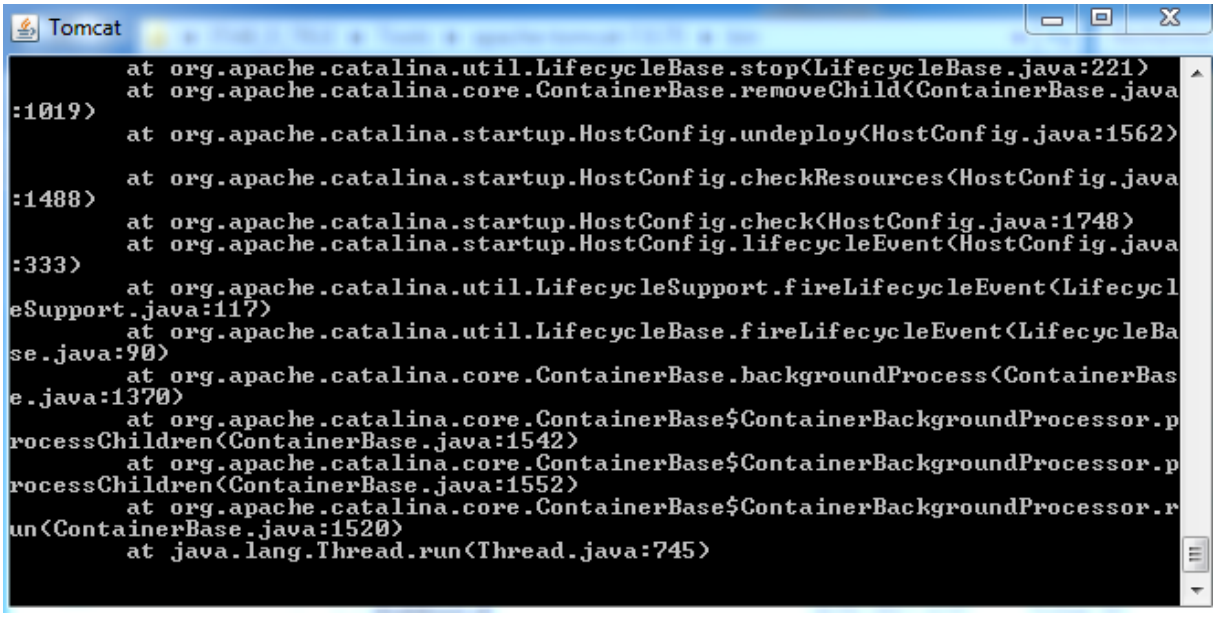
- Exécuter le fichier .bat 'Shutdown.bat' : pour arrêter APPACHE

3- Exécuter dans le POM du projet 'springmvc0' : clean install/verrify/package
4- Faire copier 'springmvc0.war ' dans le dossier '\apache-tomcat-7.0.75\webapps'

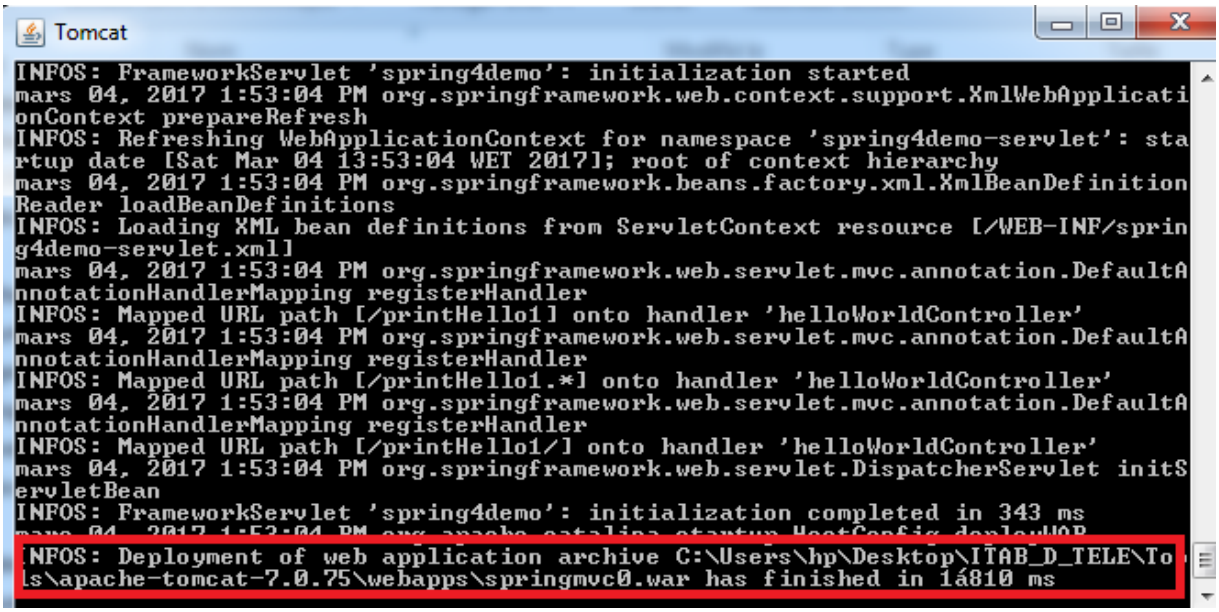
Pour la génération du dossier Dézipé 'springmvc0' :




Voir l'invite commande avant :




Et après l'opération :



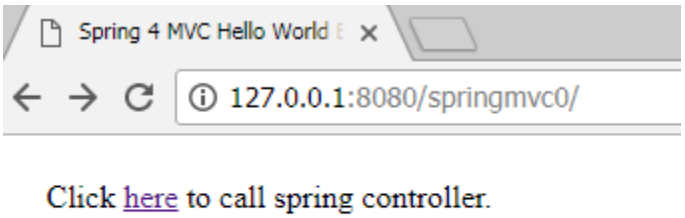

5- Pour administrer Tomcat lancez dans le browser :

127.0.0.1 :8080

6- pour regarder le résultat du projet :

Lancez dans le browser : http://127.0.0.1:8080/springmvc0/

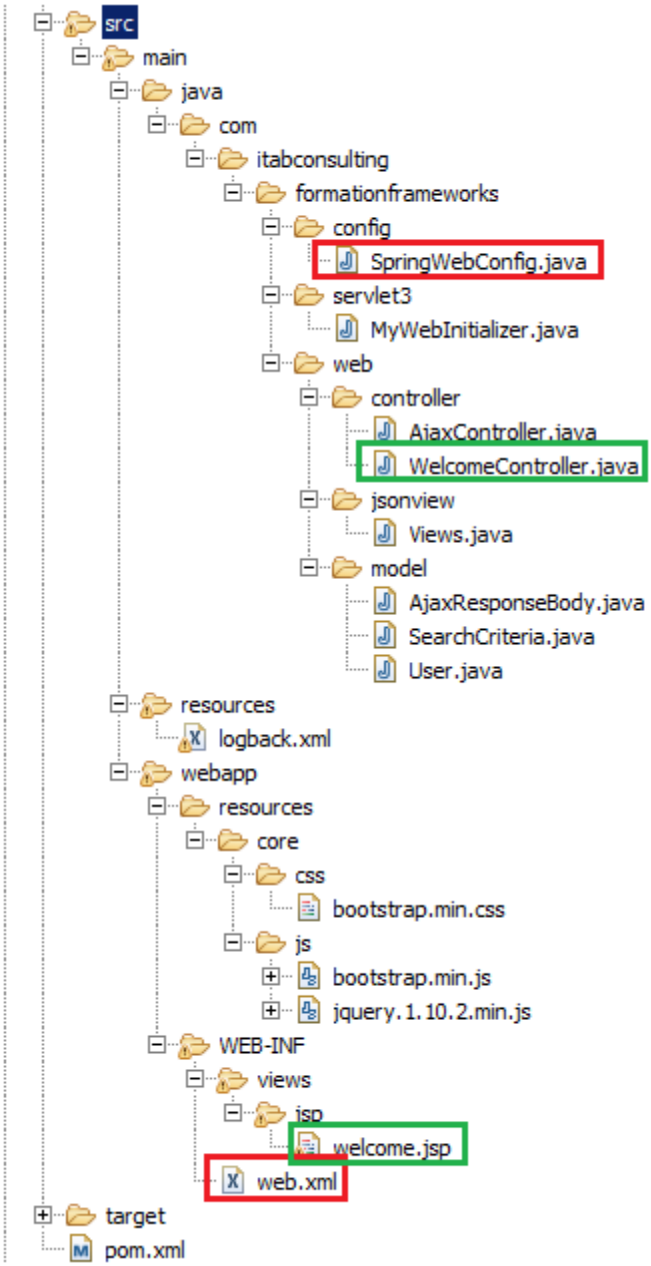
- Voir le lien hypertext : vers 'printhello.htm'
- Dans 'web.xml' : le nom de spring4demo-servlet.xml dans 'web.xml' : <servlet-name>



**TP2 :**

Allez dans le projet 'springmvc1-ajax ' :

L'arborescence du projet est comme suit :




On remarque principalement les fichiers suivants :

- Fichier de configuration : SpringWebConfig.java
- Fichier du modèle : User.java
- Fichier des contrôleurs : WelcomeContoller.java et AjaxController.java
- Fichier de vue : welcome.jsp
- Fichier de style : bootstrap.min.css
- Fichier Jquery: jquery.1.10.2.min.js

**Synthèse:**

Ce projet suit le modèle AJAX :

a- Dans POM il y a WAR donc on doit déployer dans le serveur
b- Web.xml est vide !
c- Remarquer le fichier 'welcome.jsp'

```
<spring:url value="/resources/core/js/jquery.1.10.2.min.js" var="jqueryJs" />
<script src="${jqueryJs}"></script>
```

La bibliothèque jquery = javascript + ajax

d- Remarquer la classe 'springwebconfig.java' : elle traduit ce qui était en XML dans l'ancien méthode.

Et donc à la place de XML on utilise un fichier Java

e- Remarquer le fichier 'MyWebInitializer.java'
f- Remarquer le fichier 'AjaxController.java' :

L'annotation '@RestController' :

L'annotation '@RequestMapping(value = "/search/api/getSearchResult")'

Dans le fichier 'welcome.jsp' : searchViaAjax →SearchCriteria

g- Sur POM executez clean install

Puis

Copier springmvc-ajax.WAR dans webapps de tomcat

**REMARQUES :**

Ajax Controller : il cherche USER et autres, puis affiche dans JSON.

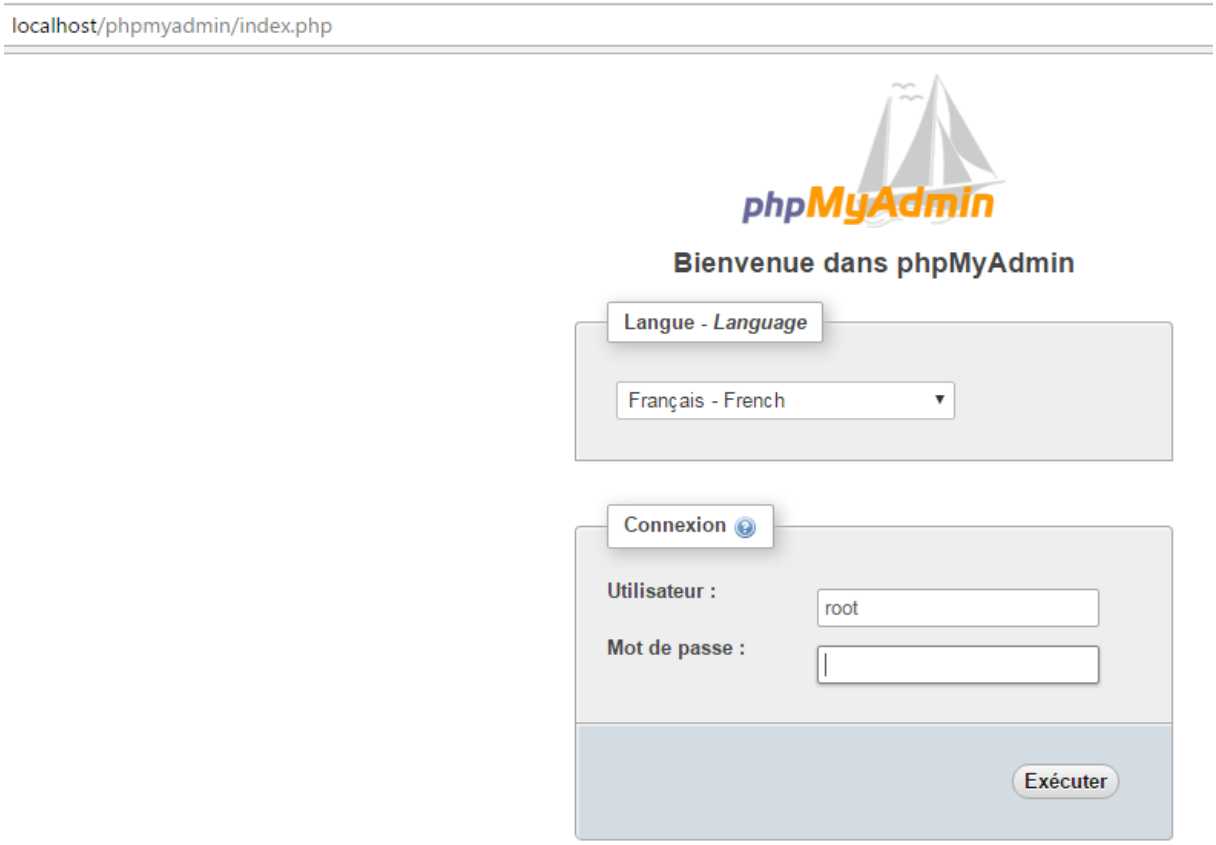
h- Lancer dans le browser :

127.0.0.1 :8080/springmvc1-ajax

**TP3 :**

Allez dans springmv2c-Hibernate :

1- Installer 'Wampserver32' : simulateur de serveur appache
2- Lancez : localhost/phpmyadmin dans le browser : login = root + mot de passe =vide




3- Dans phpmyadmin :
   a- Allez dans l'onglet SQL
   b- Dans le dossier 'springmvc2-hibernate' vous trouverez le fichier 'SQL'

      Copier le script SQL dans phpmyadmin→SQL

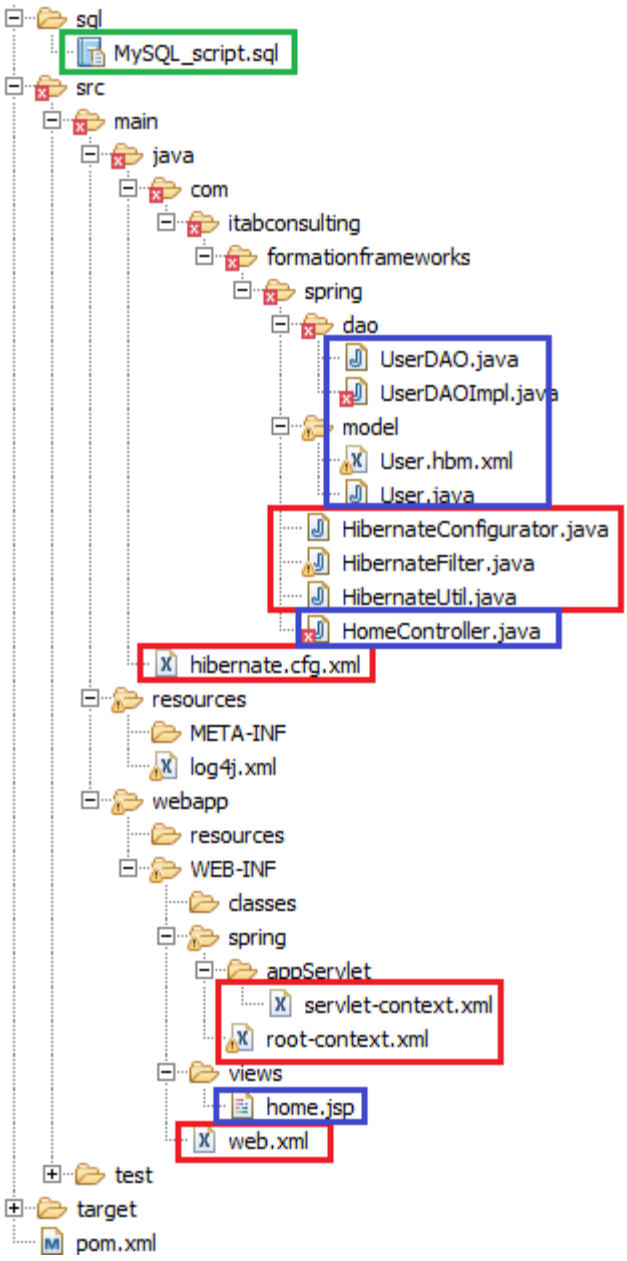
   c- Faire exécuter : par le bouton 'Exécuter' :

      Création de la base + création des tables + création des données

   d- Sur POM : clean install
   e- Copier 'springmvc2-hibernate.war' dans webapps
   f- Executez url=127.0.0.1:8080/springmvc2-hibernate

4- Description du project :

L'arborescence du projet est comme suit :




On remarque principalement les fichiers suivants :

- Fichier de configuration : hibernate.cfg.xml, web.xml, log4j.xml
- Fichier du modèle : User.java, User.hbm.xml
- Fichier des contrôleurs : HomeController.java
- Fichier de vue : home.jsp
- Fichier de spring : servlet-context.xml, root-context.xml
- Fichier du spring pour accès (Objet) aux données : UserDAO et UserDAOImpl

**Synthèse:**

On remarque plusieurs fichiers de la configuration du spring :

- root-context.xml, log4j.xml
- servlet-context.xml : pooling de connexion : La définition de datasource : la gestion des connections à la base

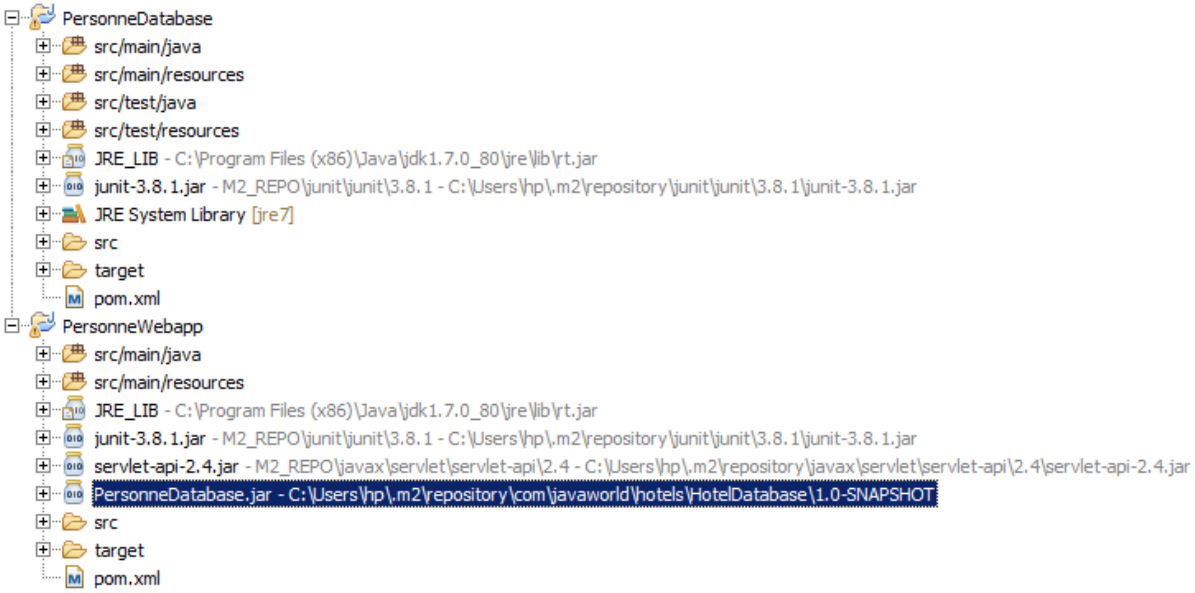
Aussi les fichiers du modèle et les fichiers dao (requête CRETIRIA):

- User.java, User.hbm.xml , UserDAO

## VIII- Examples:

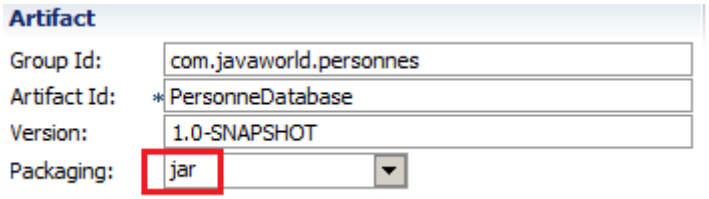
1- Arborescence du projet :

L'exemple suivant présente un workspace regroupement deux projets, le premier est de packaging JAR et le deuxième est de packaging WAR.

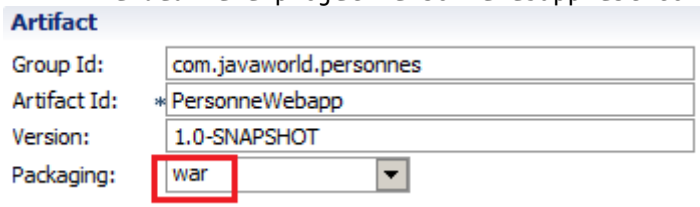



2- Gestion de la dépendance :

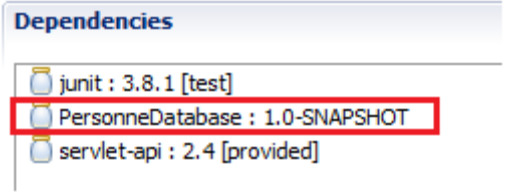
Le premier projet PersonneDatabase est comme suit :




Le deuxième projet PersonneWebapp est comme suit :

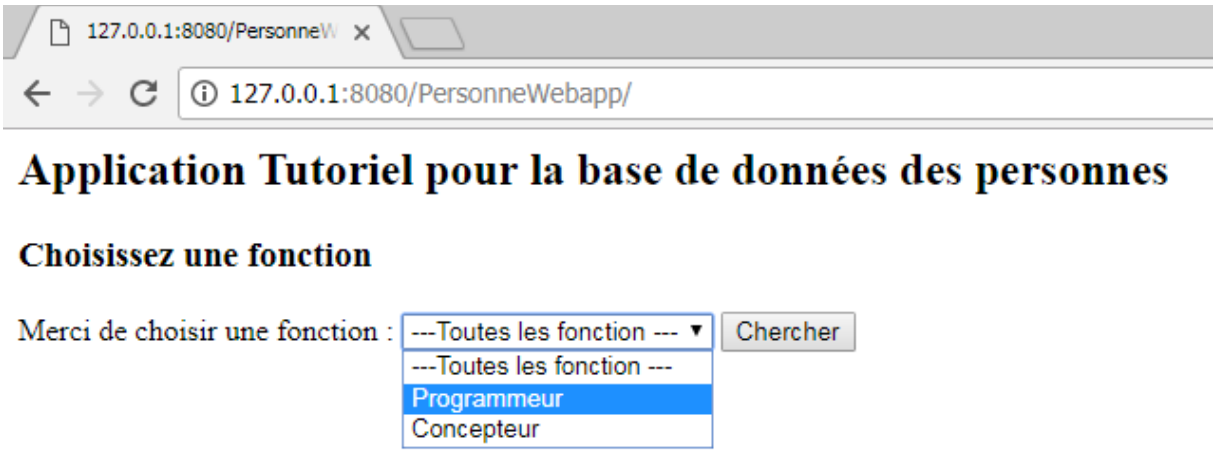



Le projet PersonneWebapp dépend du projet PersonneDatabase :



3- Exécution de l'exemple :

L'exemple illustre un filtre d'une liste des personnes par fonction.




Le choix de la fonction Programmeur affiche la liste des programmeurs parmi ceux disponibles.