Formation JAVA – Rappels sur les Servlets

Pour	Formation
Date	12/11/2024
Objet	Java rappels Servlets

I) Vocabulaire

- **API** : Signifie Application Programming Interface. Ce qui veut dire que c'est un ensemble de bibliothèques et librairies dédié pour implémenter une fonctionnalité donnée.

- **ORM** : Object-Relational Mapping (**MOR** : Mapping Objet-Relationnel en français) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.

- **JPA** : Java Persistence API (abrégée en JPA), est une interface de programmation Java permettant aux développeurs d'organiser des données relationnelles dans des applications utilisant la plateforme Java.

- **JPQL** : Le langage JPQL (**Java Persistence Query Language**) est un langage de requête orienté objet, similaire à SQL, mais au lieu d'opérer sur les tables et colonnes, JPQL travaille avec des objets persistants et de leurs propriétés. Il est très proche du langage SQL dont il s'inspire fortement mais offre une approche objet. La grammaire de ce langage est définie par la spécification J.P.A.

- **HQL** : **Hibernate Query Language** est aussi un langage de requête orienté objet au même titre que JPQL. La principale différence avec le langage JQL est que le en HQL le « **Select** » sur l'objet n'est pas nécessaire. En fin de compte pour le JPQL on aura : **Select person from Personne person** alors que pour le HQL on aura : **from Personne**.

- **Bean** : le « **Bean** » (ou haricot en français) est une technologie de composants logiciels écrits en langage Java. Les **Beans** sont utilisés pour encapsuler plusieurs objets dans un seul objet. Le « **Bean** » regroupe alors tous les attributs des objets encapsulés. Ainsi, il représente une entité plus globale que les objets encapsulés de manière à répondre à un besoin métier.

- **Pattern IoC** : L'inversion de contrôle (inversion of control, IoC) est un patron d'architecture commun à tous les Frameworks (ou cadre de développement et d'exécution). Il fonctionne selon le principe que le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du Framework ou de la couche logicielle sous-jacente. En effet selon un problème, il existe différentes formes, ou représentation d'IoC, le plus connu étant l'injection de dépendances (dependency injection) qui est un patron de conception permettant, en programmation orientée objet, de découpler les dépendances entre objets.

- **Pattern AOP** : L'AOP (Aspect Oriented Programming) ou POA (Programmation Orientée Aspect) est un paradigme de programmation ayant pour but de compléter la programmation orientée objet et permettre d'implémenter de façon plus propre les problématiques transverses à l'application. En effet, elle permet de factoriser du code dans des greffons et de les injecter en divers endroits sans pour autant modifier le code source des endroits en question.

- **GemFire** : GemFire est une infrastructure de gestion de données distribuée hautes performances qui se situe entre le cluster d'applications et les sources de données back-end. Avec GemFire, les données peuvent être gérées en mémoire, ce qui accélère l'accès.

II) Architecture Java Web et Java EE.

Java EE est la version "entreprise" de Java, elle a pour but de faciliter le développement d'applications distribuées.
Mais en fait, Java EE est avant tout une norme.

C'est un ensemble de standard décrivant des services techniques comme, par exemple, comment accéder à un annuaire, à une base de données, à des documents...

Important : Java EE définit ce qui doit être fournit mais ne dit pas comment cela doit être fournit.

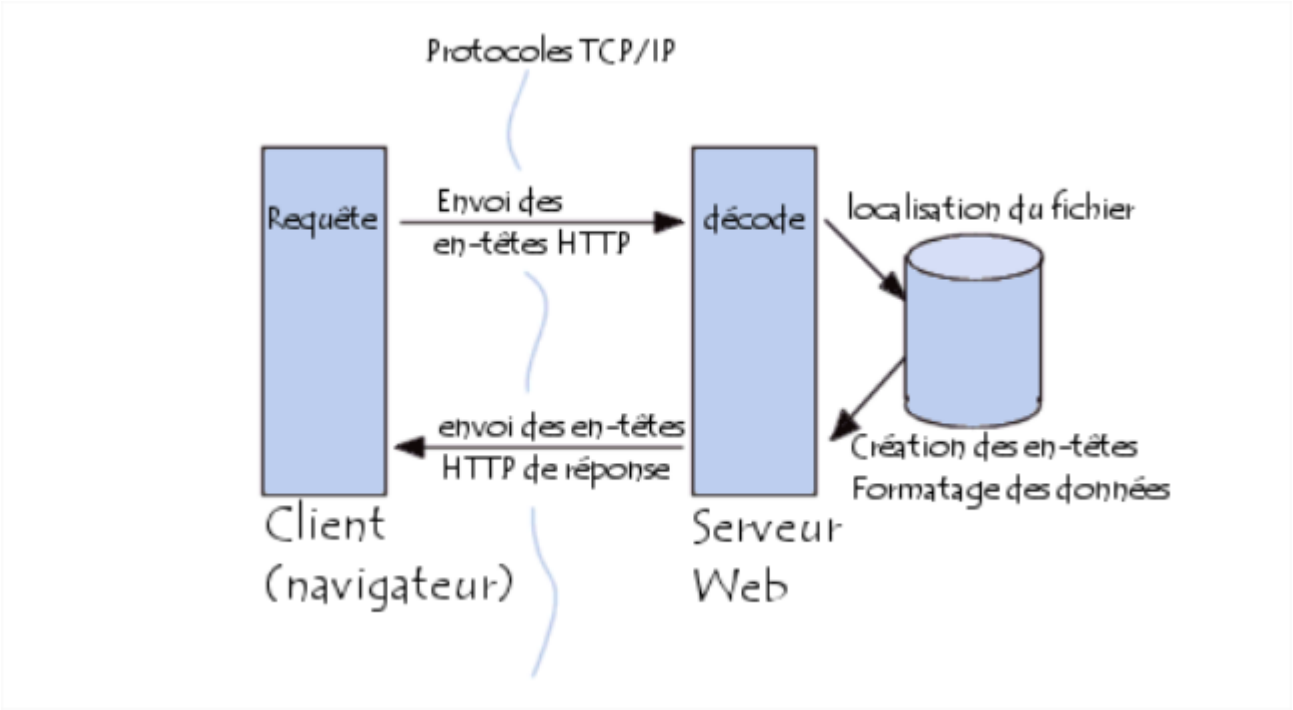
Exemple de services :

- JNDI (Java Naming and Directory Interface) est une API d'accès aux services de nommage et aux annuaires d'entreprises tels que DNS, NIS, LDAP…
- JTA (Java Transaction API) est une API définissant des interfaces standard avec un gestionnaire de transactions.

III) Rappels sur le protocole HTTP.

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet depuis 1990.
Le but ce protocole est de permettre un transfert de fichiers (essentiellement au format HTML) localisés grâce à une chaîne de caractères appelée URL entre un navigateur (le client) et un serveur Web.

La communication entre le navigateur et le serveur se fait en deux temps :



- Le navigateur effectue une **requête HTTP**
- Le serveur traite la requête puis envoie une **réponse HTTP**

Une requête HTTP est traitée à travers plusieurs méthodes :

GET : c'est la méthode la plus courante pour demander une ressource. Une requête GET est sans effet sur la ressource, il est possible de répéter la requête sans effet.

POST : cette méthode est utilisée pour transmettre des données en vue d'un traitement de ressource (le plus souvent depuis un formulaire HTML). L'URI fourni est l'URI d'une ressource à laquelle s'appliqueront les données envoyées. Le résultat peut être la création de nouvelles ressources ou la modification de ressources existantes.

HEAD : cette méthode ne demande que des informations sur la ressource, sans demander la ressource elle-même.

OPTIONS : cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.

CONNECT : cette méthode permet d'utiliser un proxy comme un tunnel de communication.

TRACE : cette méthode demande au serveur de retourner ce qu'il a reçu dans le but de tester et d'effectuer un diagnostic sur la connexion.

PUT: cette méthode permet de remplacer ou d'ajouter une ressource sur le serveur. L'URI fourni est celui de la ressource en question.

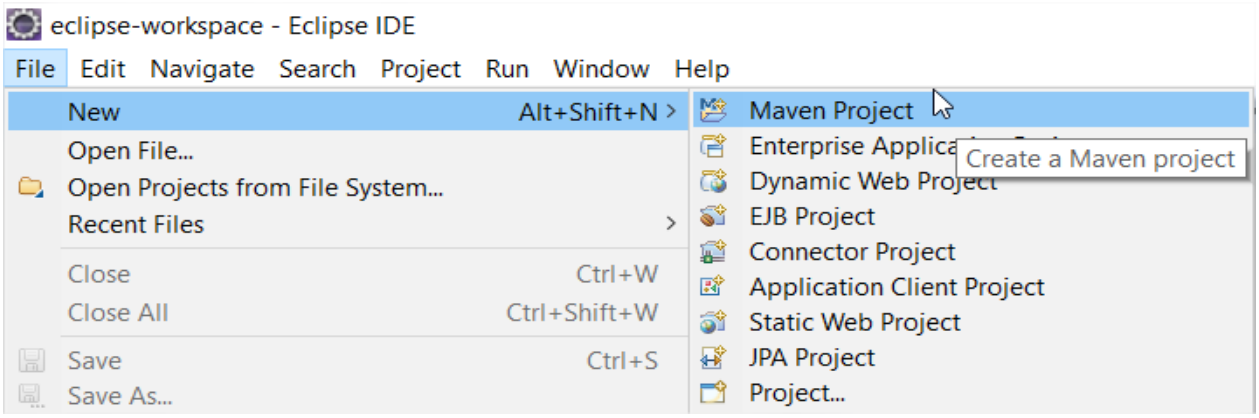
PATCH: cette méthode permet, contrairement à PUT, de faire une modification **partielle** d'une ressource.

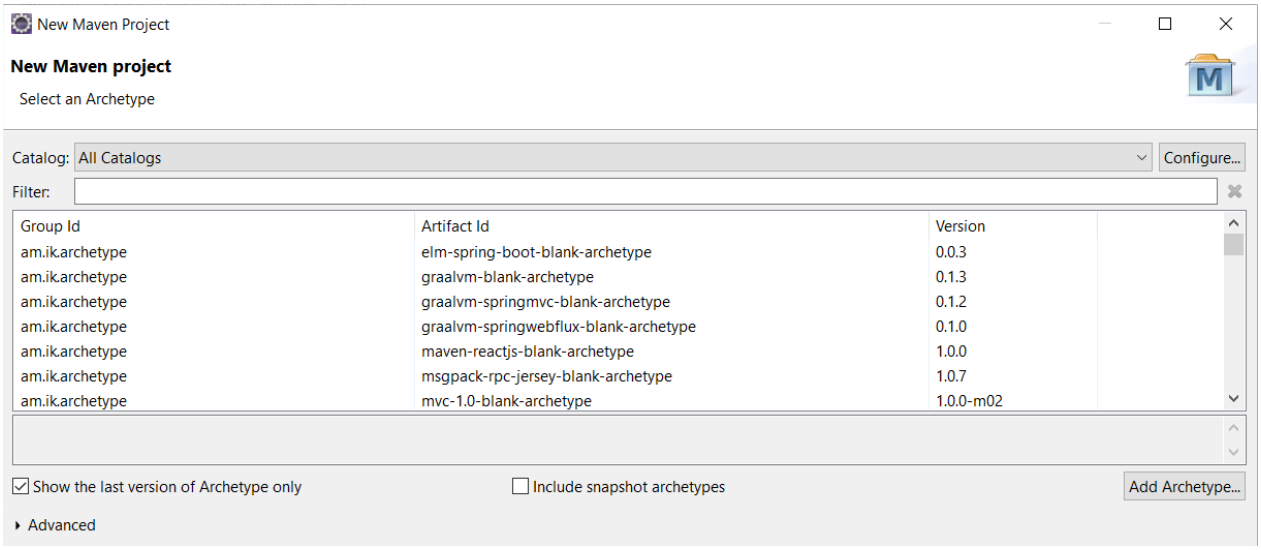
DELETE: cette méthode permet de supprimer une ressource du serveur.

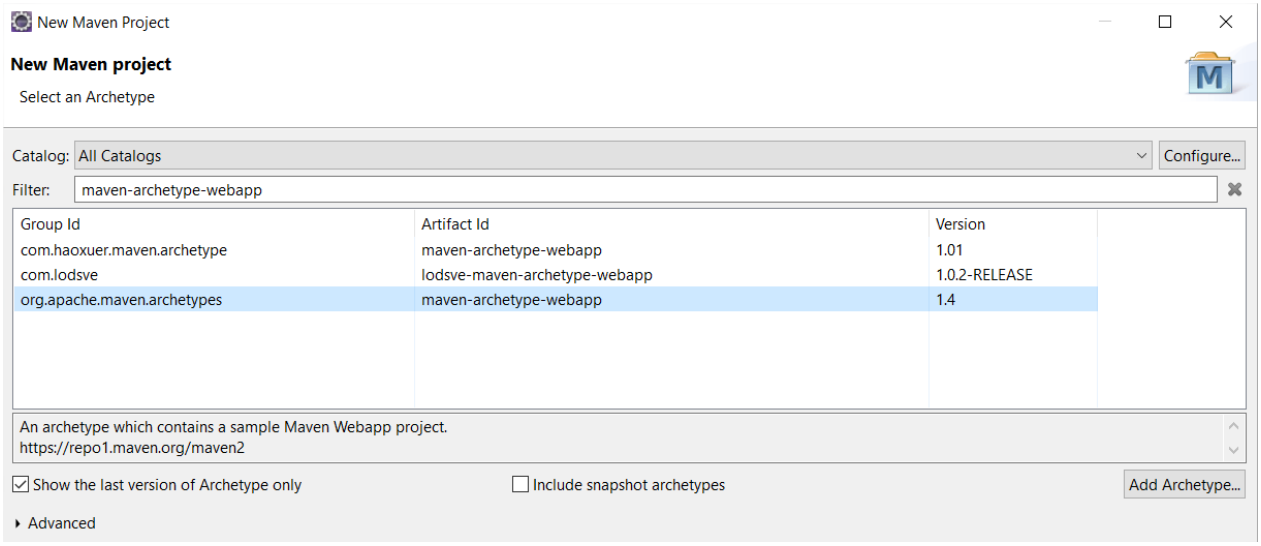
IV) Rappels sur les projets Maven Web HttpServlet

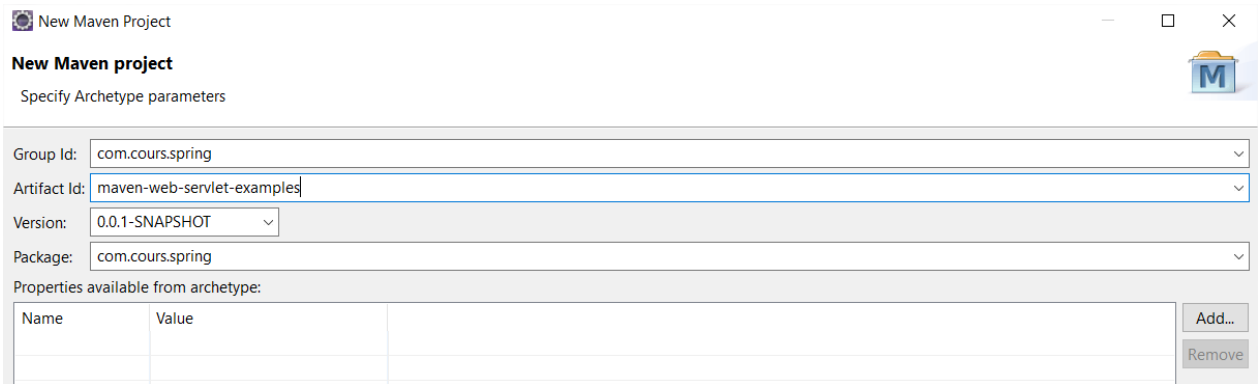
Créer le projet Maven **maven-web-servlet-examples** de type **war** (archetype=**maven-archetype-webapp**).

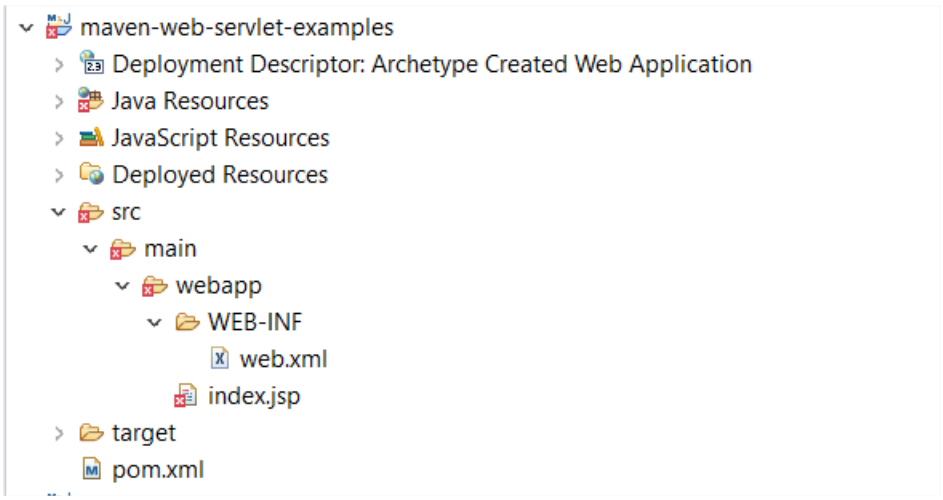
1. *Création du projet Maven*



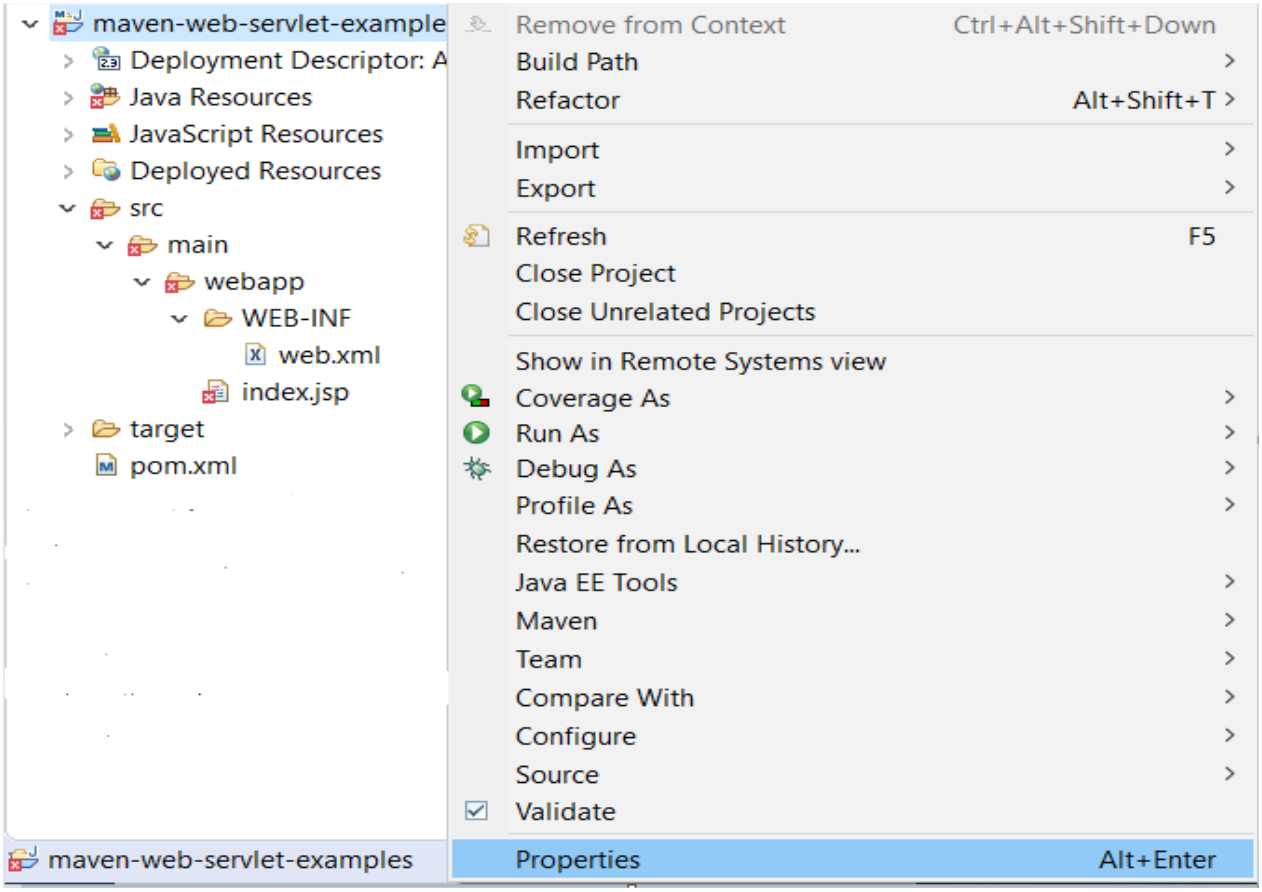


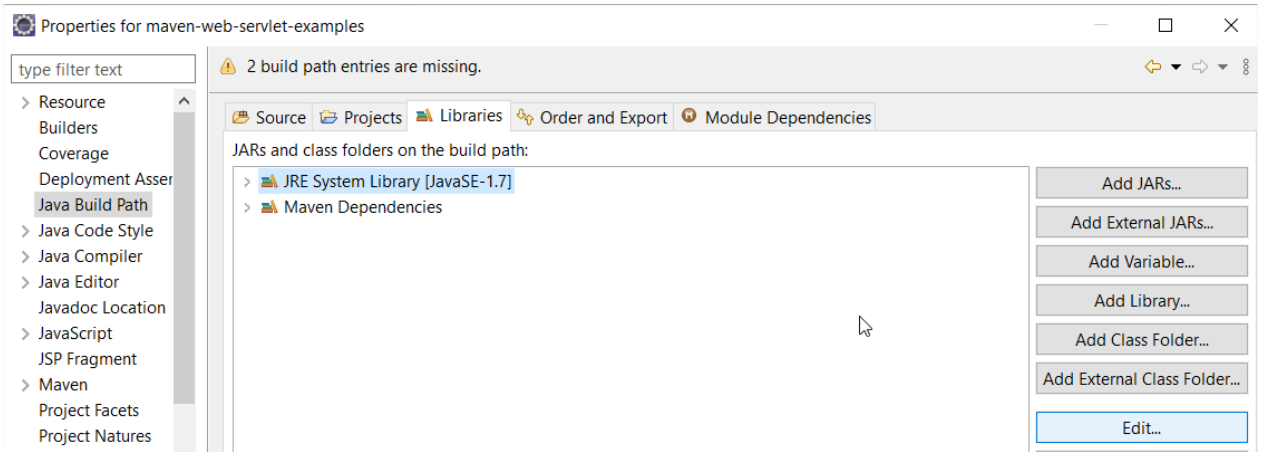

New Maven Project
New Maven project
Select an Archetype
Catalog: All Catalogs
Configure...
Filter: maven-archetype-webapp
Group Id
Artifact Id
Version
com.haoxuer.maven.archetype
maven-archetype-webapp
1.01
com.lodsve
lodsve-maven-archetype-webapp
1.0.2-RELEASE
org.apache.maven.archetypes
maven-archetype-webapp
1.4
An archetype which contains a sample Maven Webapp project.
https://repo1.maven.org/maven2
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced


New Maven Project
New Maven project
Specify Archetype parameters
Group Id: com.cours.spring
Artifact Id: maven-web-servlet-examples
Version: 0.0.1-SNAPSHOT
Package: com.cours.spring
Properties available from archetype:
Name
Value
Add...
Remove

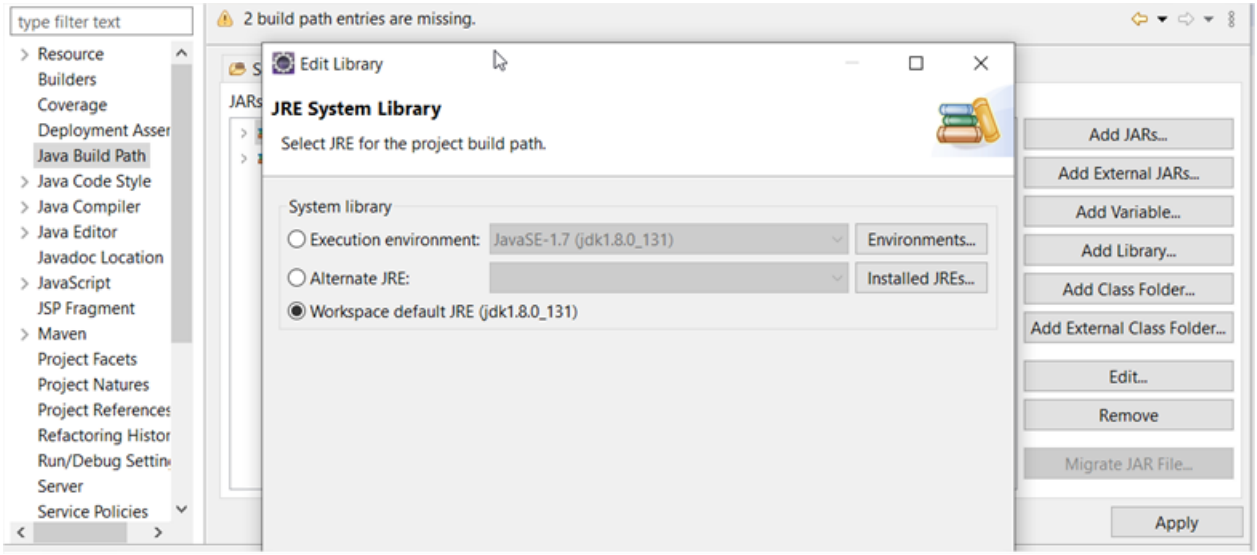

maven-web-servlet-examples
Deployment Descriptor: Archetype Created Web Application
Java Resources
JavaScript Resources
Deployed Resources
src
main
webapp
WEB-INF
web.xml
index.jsp
target
pom.xml

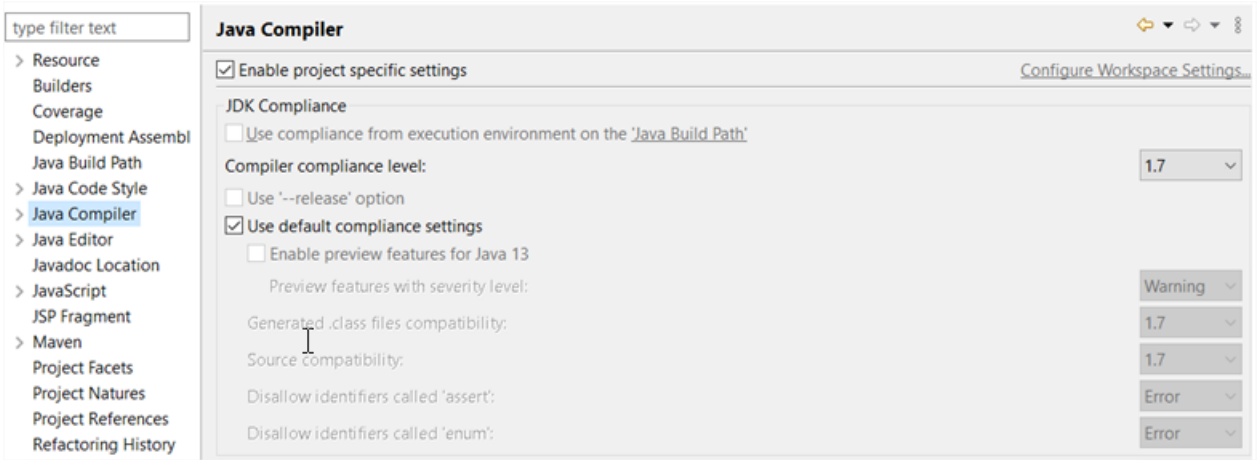
2. *Verification de la version de Java*

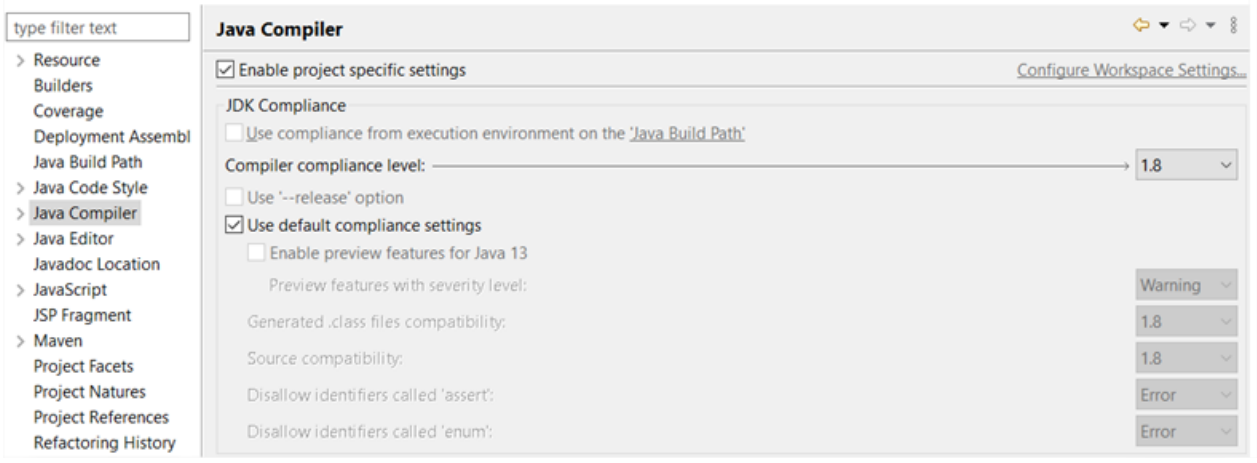




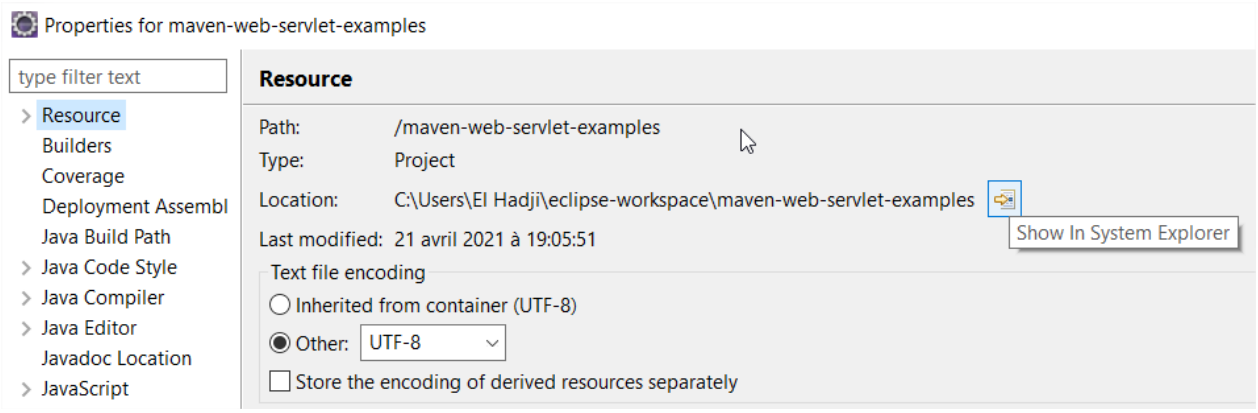
Changer la version du Build Path de **1.7 à 1.8**.

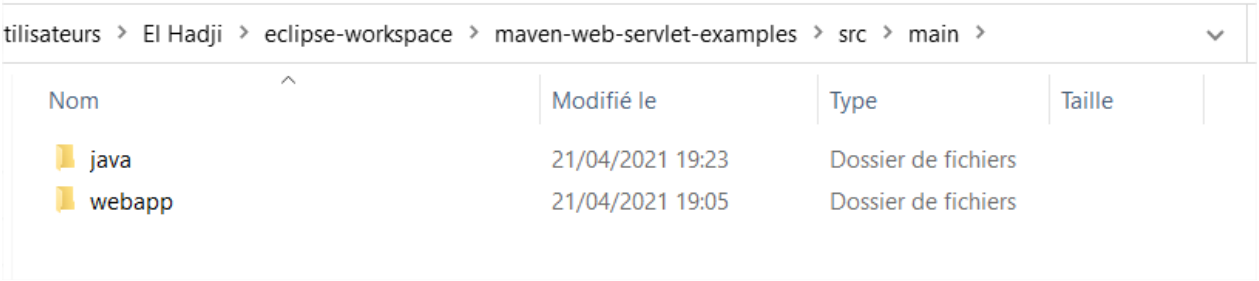




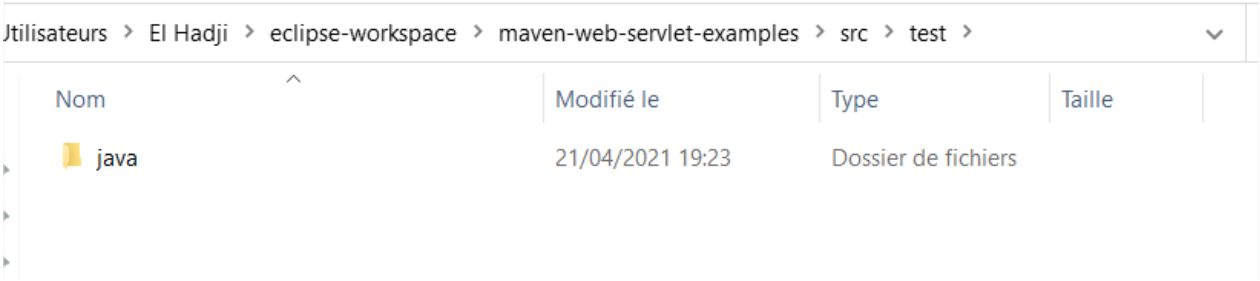


3. *Création des dossiers src/main/resources et src/test/resources*





Créer le dossier **src/main/resources** :



Créer le dossier **src/test/resources** :

Copier le fichier **log4j.properties** dans **/src/main/resources** dont le contenu sera :

```
# Set root logger level to DEBUG and its only appender to CONSOLE.
log4j.rootLogger=DEBUG, CONSOLE

# A1 is set to be a ConsoleAppender.
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.encoding=UTF-8

# A1 uses PatternLayout.
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=[%d{dd/MM/yyyy HH:mm:ss} %p %C{1}.%M] %m%n

# Change the level of messages for various packages.
log4j.logger.com.cours*=DEBUG
```

4. *Le pom.xml*

Le fichier **pom.xml** aura pour contenu initiale :

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

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.cours.spring</groupId>
 <artifactId>maven-web-servlet-examples</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>war</packaging>

 <name>maven-web-servlet-examples Maven Webapp</name>
 <!-- FIXME change it to the project's website -->
 <url>http://www.example.com</url>

 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
 </properties>

 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.11</version>
   <scope>test</scope>
  </dependency>
 </dependencies>

 <build>
  <finalName>maven-web-servlet-examples</finalName>
  <pluginManagement><!--
 lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
   <plugins>
    <plugin>
     <artifactId>maven-clean-plugin</artifactId>
     <version>3.1.0</version>
    </plugin>
    <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_war_packaging -->
    <plugin>
     <artifactId>maven-resources-plugin</artifactId>
     <version>3.0.2</version>
    </plugin>
```

```
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.2.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-install-plugin</artifactId>
        <version>2.5.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>2.8.2</version>
      </plugin>
    </plugins>
  </pluginManagement>
 </build>
</project>
```

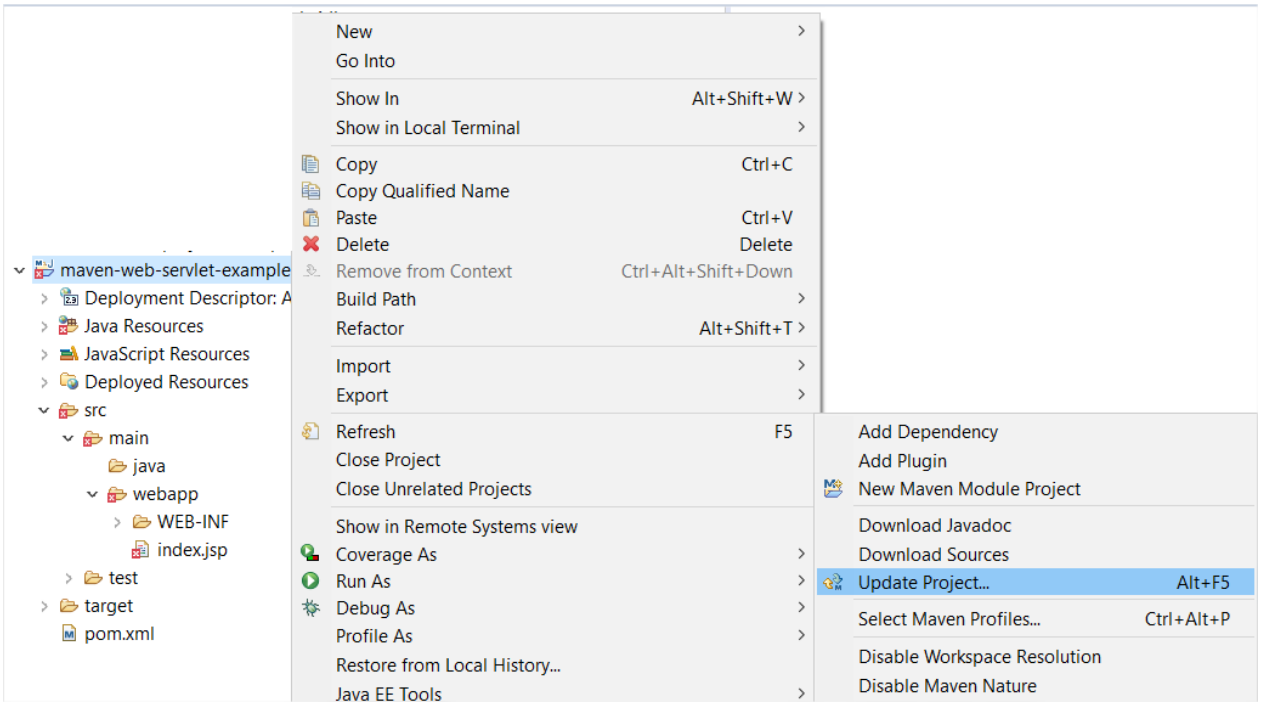
Le fichier **pom.xml** peut devenir :

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

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.cours.spring</groupId>
 <artifactId>maven-web-servlet-examples</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>war</packaging>
 <name>maven-web-servlet-examples</name>
 <properties>
     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
     <maven.compiler.source>1.8</maven.compiler.source>
     <maven.compiler.target>1.8</maven.compiler.target>
     <javax.servlet.version>4.0.0</javax.servlet.version>
     <javax.servlet.jsp.version>2.3.0</javax.servlet.jsp.version>
     <jstl.version>1.2</jstl.version>
     <log4j.version>1.2.17</log4j.version>
     <commons.logging.version>1.2</commons.logging.version>
  </properties>

  <dependencies>
     <!-- API Servlet : implémentation -->
     <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>${javax.servlet.version}</version>
        <scope>provided</scope>
     </dependency>
     <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>javax.servlet.jsp-api</artifactId>
        <version>${javax.servlet.jsp.version}</version>
        <scope>provided</scope>
     </dependency>
     <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>${jstl.version}</version>
     </dependency>
     <!-- Debut log4j dependencies -->
     <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>${log4j.version}</version>
     </dependency>
     <dependency>
```

```
        <groupId>log4j</groupId>
        <artifactId>apache-log4j-extras</artifactId>
        <version>${log4j.version}</version>
      </dependency>
      <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>${commons.logging.version}</version>
      </dependency>
      <!-- Fin log4j dependencies -->
    </dependencies>
    <build>
      <finalName>maven-web-servlet-examples</finalName>
    </build>
</project>
```

5. *Mise à jour du projet*

Faire un **Maven →Update Project** de votre projet.



6. *Le fichier webapp/index.jsp*

Le fichier **index.jsp** aura pour contenu :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
   <head>
      <title>Page d'exemples pour la formation Java Web Servlet </title>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
   </head>
   <body>
      <h1>Page d'exemples pour la formation Java Web Servlet</h1>
   </body>
</html>
```

7. *Le web.xml*

Le fichier **web.xml** aura pour contenu :

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
</web-app>
```

Le fichier **web.xml** peut devenir :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-
app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSch
ema-
instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd">
   <session-config>
      <session-timeout>
            30
      </session-timeout>
   </session-config>
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>
</web-app>
```

8. *Le fichier webapp/pages/myPage.jsp*

Le fichier **webapp/pages/myPage.jsp** aura pour contenu :

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>


<%

String myParam = "";

if (request.getParameter("myParam") != null) {
   myParam = request.getParameter("myParam");
   System.out.println("getParameter myParam : "+myParam);
}

if (request.getAttribute("myParam") != null && request.getAttribute("myParam") instanceof String) {
   myParam = (String) request.getAttribute("myParam");
   System.out.println("getAttribute myParam : "+myParam);
}

%>

<!DOCTYPE html>
<html>
   <head>
      <title>Servlet MyServlet</title>
   </head>
   <body>
      <h1>Servlet MyServlet at ${pageContext.request.contextPath}</h1>
      <div>myParam :  <% out.println(myParam);%></div>
   </body>
</html>
```

9. *Configuration d'une Servlet*

On peut configurer une servlet de deux façons :

- Par configuration XML avec le **web.xml**

```
<servlet>
  <servlet-name>MyServlet</servlet-name>
  <servlet-class>com.cours.servlets.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MyServlet</servlet-name>
  <url-pattern>/MyServlet</url-pattern>
</servlet-mapping>
```

- Par configuration Java avec l'annotation **@WebServlet**

```
package com.cours.servlets;

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

@WebServlet(name = "MyServlet", urlPatterns = { "/MyServlet" })
public class MyServlet extends HttpServlet {

  private static final long serialVersionUID = 1L;

  @Override
  public void init() throws ServletException {

  }

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
  }

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

  @Override
  public void destroy() {
  }
}
```

10.La Servlet MyServlet

La classe **MyServlet** aura pour contenu :

```
package com.cours.servlets;

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

@WebServlet(name = "MyServlet", urlPatterns = { "/MyServlet" })
public class MyServlet extends HttpServlet {

   private static final long serialVersionUID = 1L;

   @Override
   public void init() throws ServletException {
      System.out.println("******************** initialisation de la Servlet " + this.getClass().getSimpleName()
            + " ********************");

   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      System.out.println("******************** doGet de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
      String myParam = request.getParameter("myParam");

      request.setAttribute("myParam", myParam);
      System.out.println("myParam : " + myParam);
      this.getServletContext().getRequestDispatcher("/pages/myPage.jsp").forward(request, response);
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      System.out.println("******************** doPost de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
   }

   @Override
   public void destroy() {
      System.out.println("******************** Destruction de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
   }
}
```

11.La Servlet AdminServlet

La classe **AdminServlet** aura pour contenu :

```
package com.cours.servlets;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "Admin", urlPatterns = { "/Admin" })
public class AdminServlet extends HttpServlet {

   /**
    *
    */
   private static final long serialVersionUID = 1L;

   @Override
   public void init() throws ServletException {
      System.out.println("******************** initialisation de la Servlet " + this.getClass().getSimpleName()
            + " ********************");

   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      System.out.println("******************** doGet de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
      this.getServletContext().getRequestDispatcher("/pages/connectionAdminPage.jsp").forward(request, respon
se);
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      System.out.println("******************** doPost de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
      String name = request.getParameter("name");
      String password = request.getParameter("password");
      request.setAttribute("name", name);
      request.setAttribute("password", password);
      System.out.println("name : " + name + " , password : " + password);
      if ("admin".equals(name) && "admin".equals(password)) {
         this.getServletContext().getRequestDispatcher("/pages/succesAdminPage.jsp").forward(request, response)
;
      } else {
```

```
            request.setAttribute("errorMessage", "Veuillez remplir les informations");
            this.getServletContext().getRequestDispatcher("/pages/connectionAdminPage.jsp").forward(request, resp
onse);
        }
    }

    @Override
    public void destroy() {
        System.out.println("******************** Destruction de la Servlet " + this.getClass().getSimpleName()
                + " ********************");

    }
}
```

La page **/webapp/pages/connectionAdminPage.jsp** aura pour contenu :

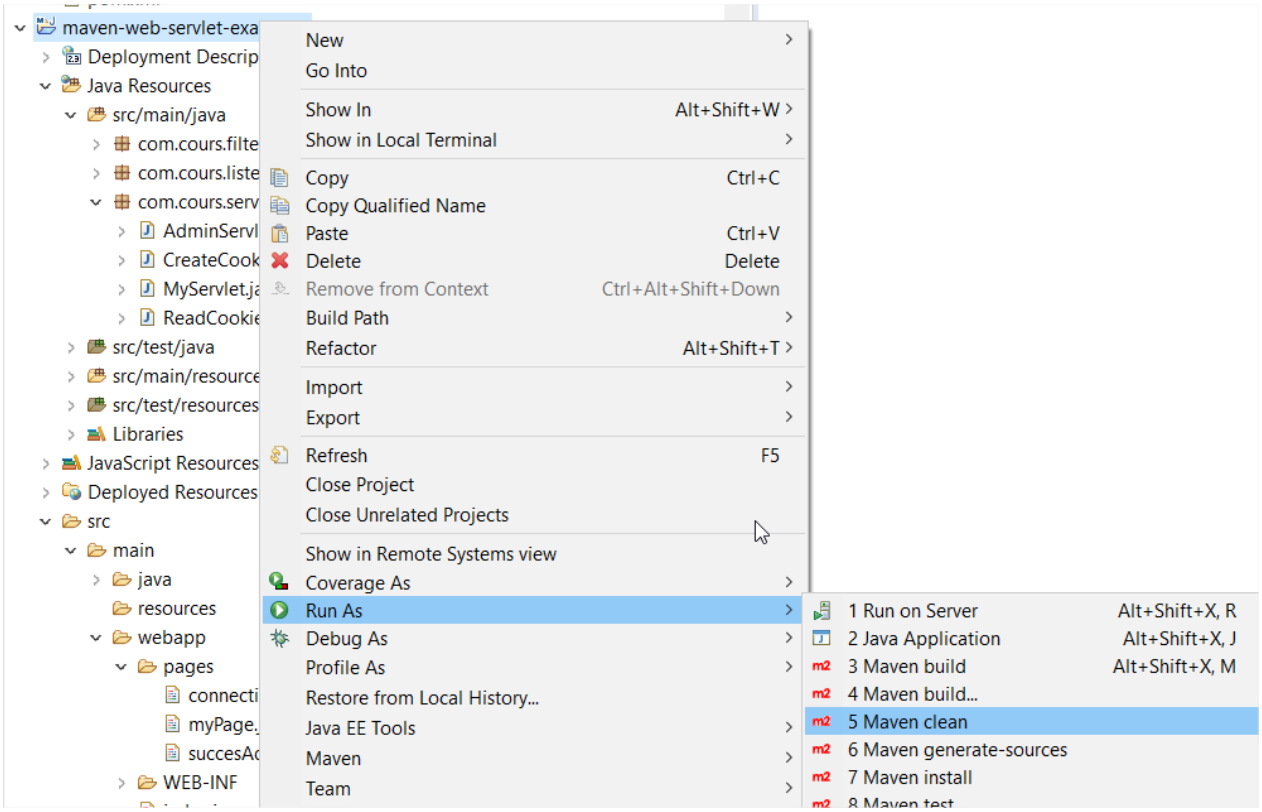
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Formulaire authentification pour la page Admin</title>
    </head>
    <body>
        <h1>Formulaire authentification pour la page Admin</h1>
        <%-- ${pageContext.request.contextPath} --%>
        <c:if test="${not empty errorMessage}">
            <div id="errorMessage" style="color: red">${errorMessage}</div>
             <%-- Equivalent à  <c:out value="${errorMessage}" />--%>
        </c:if>
            <form action="${pageContext.request.contextPath}/Admin" method="POST">
                Name : <input type="text" name="name" id="name"/><br/>
                Password : <input type="password" name="password" id="password"/><br/>
                <input type="submit" value="login">
            </form>
    </body>
</html>
```

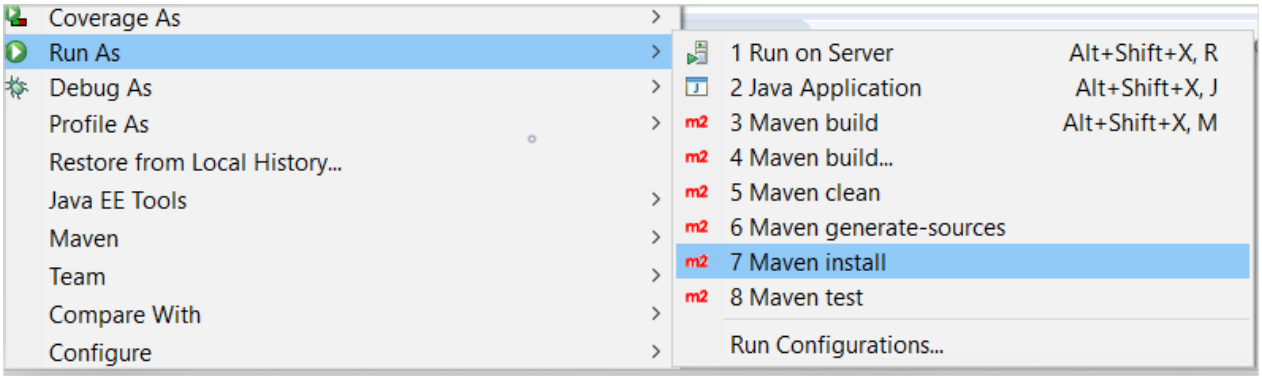
La page **/ webapp/pages/succesAdminPage.jsp** aura pour contenu :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Bravo Mr ${name} pour votre connexion à la page Admin</title>
</head>
<body>
   <h1>Bravo Mr ${name} pour votre connexion à la page Admin</h1>
</body>
</html>
```

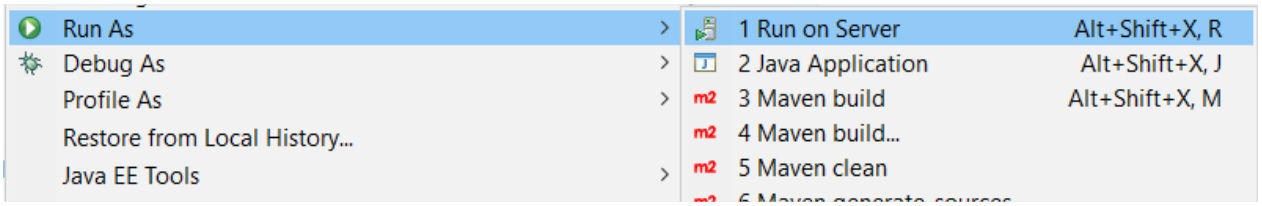
12.Lancement de l'application

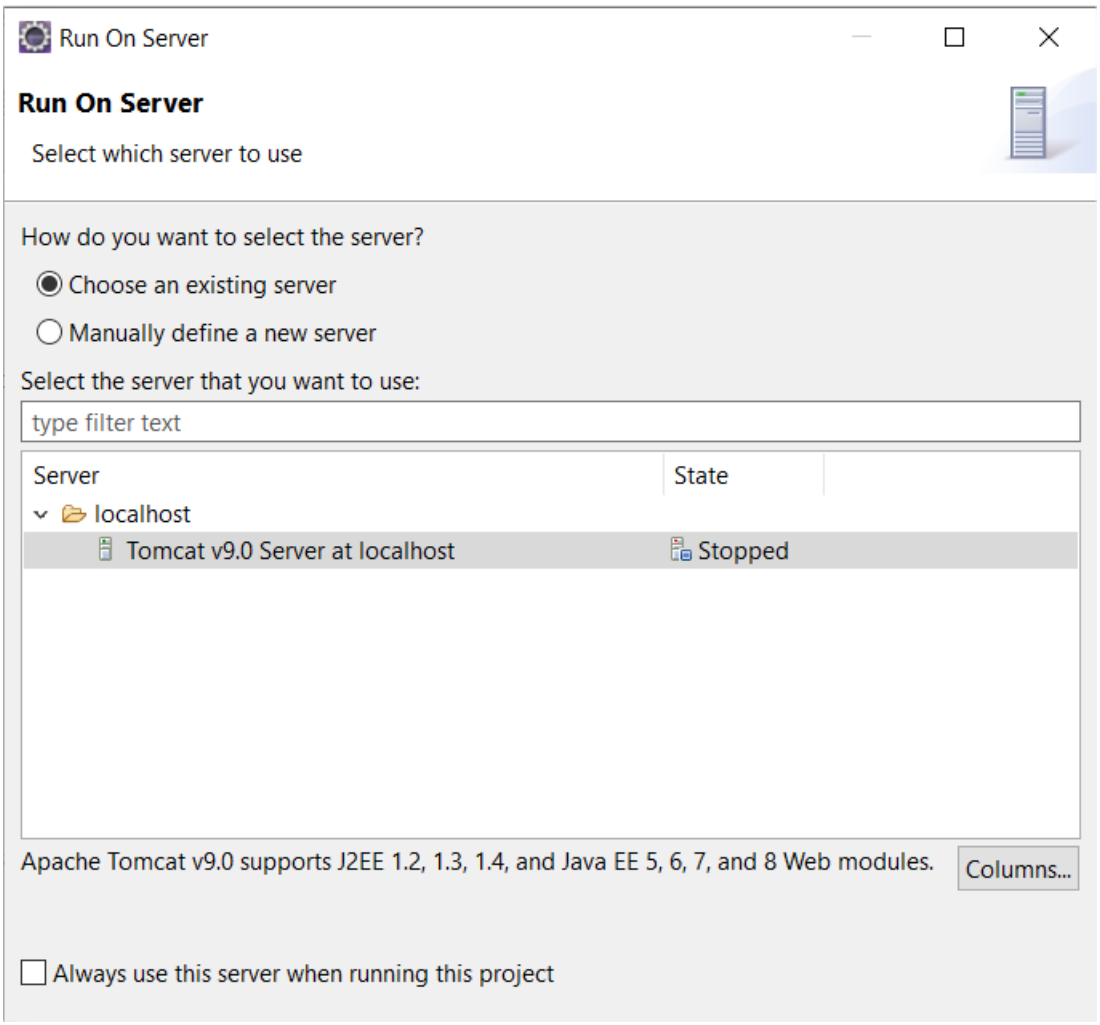
Lancez d'abord un **Maven clean** puis un **Maven install.**





Lancez un **Run on Server.**





On obtient sur http://localhost:8081/maven-web-servlet-examples/ :

On obtient aussi sur l'URL http://localhost:8081/maven-web-servlet-examples/MyServlet?myParam=123456

On obtient sur http://localhost:8081/maven-web-servlet-examples/Admin :

Taper : **admin** puis **admin** puis valider.

13.Le Listener SessionCounterListener

La classe **SessionCounterListener** aura pour contenu :

```
package com.cours.listeners;

import java.util.concurrent.atomic.AtomicInteger;

import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

import javax.servlet.annotation.WebListener;

@WebListener
public class SessionCounterListener implements HttpSessionListener {

   private static AtomicInteger atomicCounter = new AtomicInteger(0);

   public static int getTotalActiveSession() {
      return atomicCounter.get();
   }

   @Override
   public void sessionCreated(HttpSessionEvent arg0) {
      atomicCounter.incrementAndGet();
      System.out.println(
            "sessionCreated - add one session into counter, totalActiveSessions : " + getTotalActiveSession());
   }

   @Override
   public void sessionDestroyed(HttpSessionEvent arg0) {
      atomicCounter.decrementAndGet();
      System.out.println(
            "sessionDestroyed - deduct one session from counter, totalActiveSessions : " + getTotalActiveSession());
   }
}
```

14.Le Filtre à Servlet MyFilter

La classe **MyFilter** aura pour contenu :

```
package com.cours.filters;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

@WebFilter(servletNames = "Admin")
public class MyFilter implements Filter {

   @Override
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
         throws IOException, ServletException {
      String name = request.getParameter("name");
      String password = request.getParameter("password");
      if ("admin".equals(name) && "admin".equals(password)) {
         HttpServletRequest httpRequest = (HttpServletRequest) request;
         HttpSession session = httpRequest.getSession(); //sessionCreated() is executed
         session.setAttribute("name", name);
         session.setAttribute("password", password);
         //session.invalidate();  //sessionDestroyed() is executed
         chain.doFilter(request, response);// sends request to next resource
      } else {
         request.setAttribute("errorMessage", "Votre identifiant ou votre mot de passe est incorrecte");
         RequestDispatcher dispatcher = request.getRequestDispatcher("/pages/connectionAdminPage.jsp");
         dispatcher.include(request, response);
      }
   }
}
```

On obtient sur http://localhost:8081/maven-web-servlet-examples/Admin :