

Ecole normale supérieure

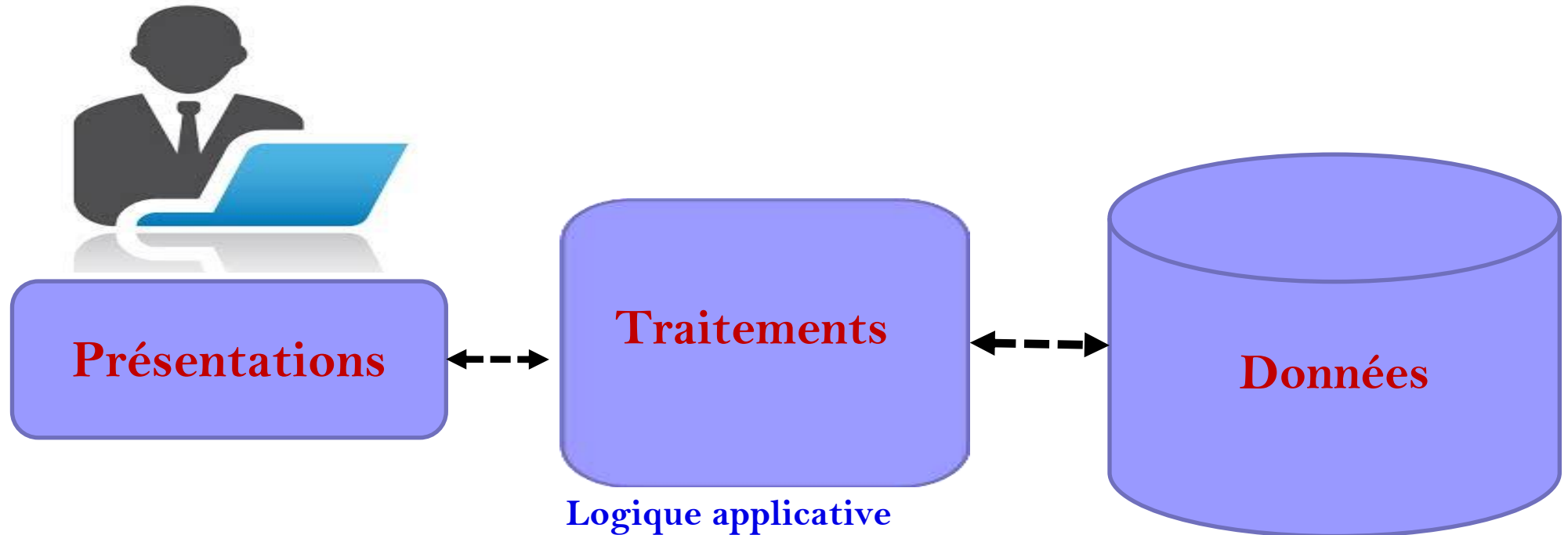


Architecture logicielle multi-niveaux

● Niveaux d'abstraction

□ Une application peut être découpée en trois niveaux d'abstraction

- ❖ Niveau présentation
- ❖ Niveau logique applicative
- ❖ Niveau données



Architecture logicielle multi-niveaux

● Niveaux d'abstraction

- ❑ **La couche de présentation**, ou IHM (Interface Homme Machine), permet l'interaction de l'application avec l'utilisateur. Ce sont : les saisies au clavier, avec la souris et l'affichage des informations à l'écran.
- ❑ **La logique applicative** décrit les traitements à réaliser par l'application pour répondre aux besoins des utilisateurs.
- ❑ **L'accès aux données** permet la gestion des informations stockées par l'application. Fonctions classiques d'un SGBD : Définition de données, Manipulation de données, Sécurité de données et Gestion de transactions.

Architecture logicielle multi-niveaux

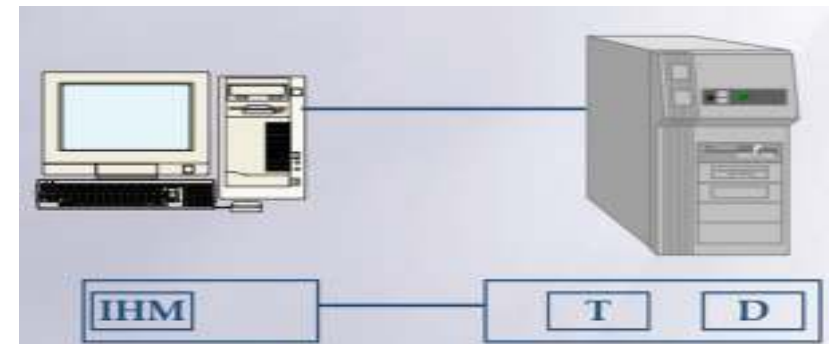
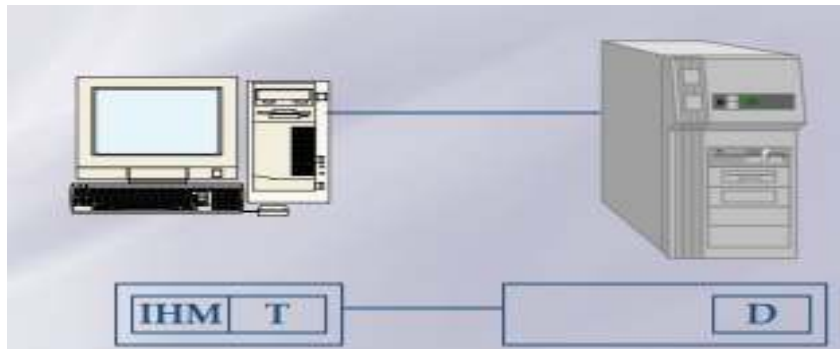
Le découpage et la répartition des 3 niveaux d'abstraction permettent de distinguer les architectures suivantes.

- ☐ Architecture 2-tiers
- ☐ Architecture 3-tiers
- ☐ Architecture n-tiers.

Architecture logicielle multi-niveaux

● Architecture 2-tiers:

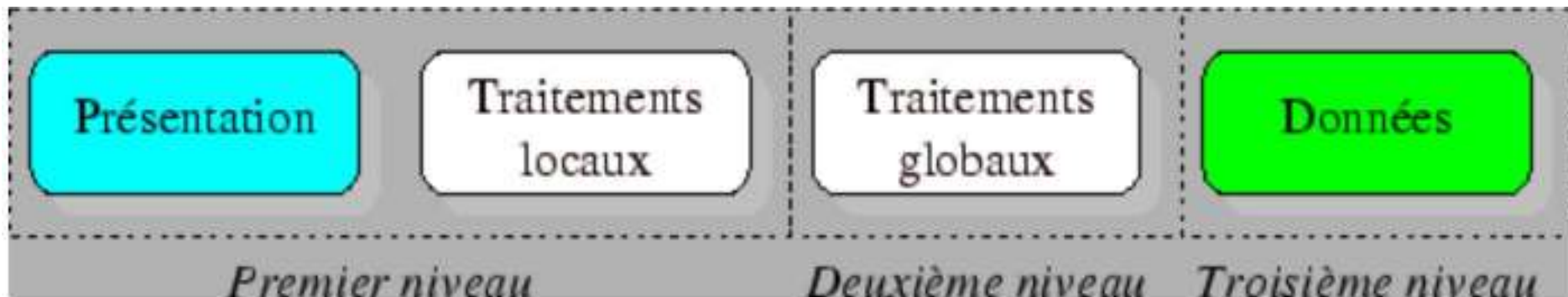
- Le poste de travail héberge l'ensemble de la gestion d'interface homme-machine et le traitement,
 - Le serveur est un serveur de base de données
 - Architecture dénommée « client lourd »
- Le poste de travail n'héberge que l'interface homme-machine
 - Le serveur héberge les données et les traitements
 - Architecture dénommée « client léger »



Architecture logicielle multi-niveaux

● Architecture 3-tiers:

- ❑ L'architecture trois tiers, encore appelée client-serveur de deuxième génération ou client-serveur distribué, sépare l'application en trois niveaux de service distincts :
 - **premier niveau** : l'affichage et les traitements locaux (contrôles de saisie, mise en forme de données...) sont pris en charge par le poste client,
 - **deuxième niveau** : les traitements applicatifs globaux sont pris en charge par le service applicatif,
 - **troisième niveau** : les services de base de données sont pris en charge par un SGBD.

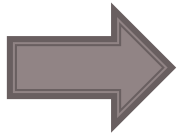


● Architecture n-tiers:

- ❑ N-Tier = Modèle logique d'architecture applicative qui vise à séparer nettement trois couches logicielles au sein d'une même application et à présenter l'application comme un empilement de ces couches :
 - présentation des données
 - traitement métier des données
 - accès aux données persistantes
- ❑ Les couches communiquent entre elles au travers d'un « modèle d'échange », et chacune d'entre elles propose un ensemble de services rendus
- ❑ Les services d'une couche sont mis à disposition de la couche supérieure
- ❑ Avantages :
 - Séparation forte entre les 3 niveaux
 - Chaque niveau peut être managé, dimensionné, distribué
 - Mises à jour et maintenance facilitées en minimisant l'impact sur les autres couches
 - Extensibilité : ajout de nouvelles fonctionnalités simplifié

Internet et le web

- l'internet est le réseau, le support physique de l'information. Pour faire simple, c'est un ensemble de machines, de câbles et d'éléments réseaux en tout genre éparpillés sur la surface du globe ;
- le web constitue une partie seulement du contenu accessible sur l'internet. Comme le courrier électronique ou encore la messagerie instantanée.



W3C, WHATWG

W3C

- Le World Wide Web Consortium, abrégé par le sigle W3C, est un organisme de standardisation à but non lucratif, chargé de promouvoir la compatibilité des technologies du World Wide Web telles que HTML5, HTML, XML, CSS, SVG, MathML, SOAP...

WHATWG

- Le Web Hypertext Application Technology Working Group (ou WHATWG) est une collaboration des différents développeurs de navigateurs web ayant pour but le développement de nouvelles technologies destinées à faciliter l'écriture et le déploiement d'applications à travers le Web.

● Introduction

- JEE=Java Enterprise Edition
- Ensemble de concepts et d'outils
 - basé sur le langage Java
 - Pour le développement des applications Client/Serveur
- JEE permet aussi de développer des applications réparties
- JEE est conçu comme un langage, ou une plate forme de développement des applications web
- JEE peut être comparer avec PHP, Django, ou à ASP.NET

● Fonctionnement du Web

Lorsqu'un utilisateur consulte un site, ce qui se passe derrière les rideaux est un simple échange entre un client et un serveur :

- **le client** : dans la plupart des cas, c'est le navigateur installé sur votre ordinateur. Retenez que ce n'est pas le seul moyen d'accéder au web, mais c'est celui qui nous intéresse dans ce cours.
- **le serveur** : c'est la machine sur laquelle le site est hébergé, où les fichiers sont stockés et les pages web générées.



● Fonctionnement du Web

La communication qui s'effectue entre le client et le serveur est régie par des règles bien définies : **le protocole HTTP**

- Le client envoie une requête HTTP au serveur sous forme d'URL
- Le serveur reçoit la requête et génère un code HTML
- Le serveur renvoie une réponse HTTP sous forme d'un code HTML
- Le navigateur client interprète la réponse (HTML) et affiche l'interface graphique

1 - Le client saisit une URL



Client

2 – Le navigateur envoie une
requête HTTP au serveur



Serveur

3 – Le serveur traite la requête et
génère la page web demandée



4 – Le serveur renvoie une
réponse HTTP au client



Protocole HTTP

<Méthode> <URI> HTTP/<Version>

[<Champ d'entête>: <Valeur>]

[<tab><Suite Valeur si >1024>]

ligne blanche

[corps de la requête pour la méthode POST]

GET /~poulingear/index.htm HTTP/1.1

Accept: image/gif, image/jpeg, */*

Accept-Language: fr

Host: www.msi.unilim.fr

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)

POST /script HTTP/1.0

Accept: www/source

Accept: text/html

Accept: image/gif

User-Agent: Lynx/2.2 libwww/2.14

From: alice@pays.merveilles.net

Content-Length: 24

** une ligne blanche **

name1=value1&

name2=value2

Le protocole HTTP

Réponse envoyé par le serveur au client

HTTP/<Version> <Status> <Commentaire Status>

[< Champ d 'entête >: <Valeur>]

[<tab><Suite Valeur si >1024>]

Ligne blanche

Document

HTTP/1.0 200 OK

Date: Wed, 02Feb97 23:04:12 GMT

Server: NCSA/1.1

MIME-version: 1.0

Last-modified: Mon, 15Nov96 23:33:16 GMT

Content-type: text/html

Content-length: 2345

* une ligne blanche *

<HTML><HEAD><TITLE> ...

</BODY></HTML>

● Méthodes du protocole HTTP

❑ Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:

- **GET** : Pour récupérer le contenu d'un document
- **POST** : Pour soumissionner des formulaires (Envoyer, dans la requête, des données saisies par l'utilisateur)
- **PUT** pour envoyer un fichier du client vers le serveur
- **DELETE** permet de demander au serveur de supprimer un document.
- **HEAD** permet de récupérer les informations sur un document (Type, Capacité, Date de dernière modification etc...)

Le protocole HTTP

- Post /login HTTP/1.1

host: site.com

Accept : application/json

Content-Type : application/x-www-form-urlencoded

cookie : JSESSIONID : C4578512611RT21455212D36144254

Saut de ligne

username=admin&password=123456&action=login

Le protocole HTTP

- Post /login HTTP/1.1

host: site.com

Accept : application/json

Content-Type : application/json

cookie : JSESSIONID : C4578512611RT21455212D36144254

Saut de ligne

```
{"username":"admin","password":"123456","action":"login"}
```

Le protocole HTTP

- GET /login?username=admin&password=123456&action=login HTTP/1.1

host: site.com

Accept : application/json

Content-Type : application/x-www-form-urlencoded

cookie : JSESSIONID : C4578512611RT21455212D36144254

Saut de ligne

Serveur d'application

❑ Les applications d'entreprise ont souvent besoin des mêmes services système :

- Gestion de la concurrence
- Services transactionnels entre composants
- Sécurité
- Gestion de la session utilisateur
- Gestion des montées en charge
- Ouverture sur de multiples sources de données
- *Pools* de connexion
- Système de tolérance aux pannes et reprise sur incident

❑ Le serveur d'application fournira ces services système

Serveur d'application

☐ Les solutions propriétaires et payantes:

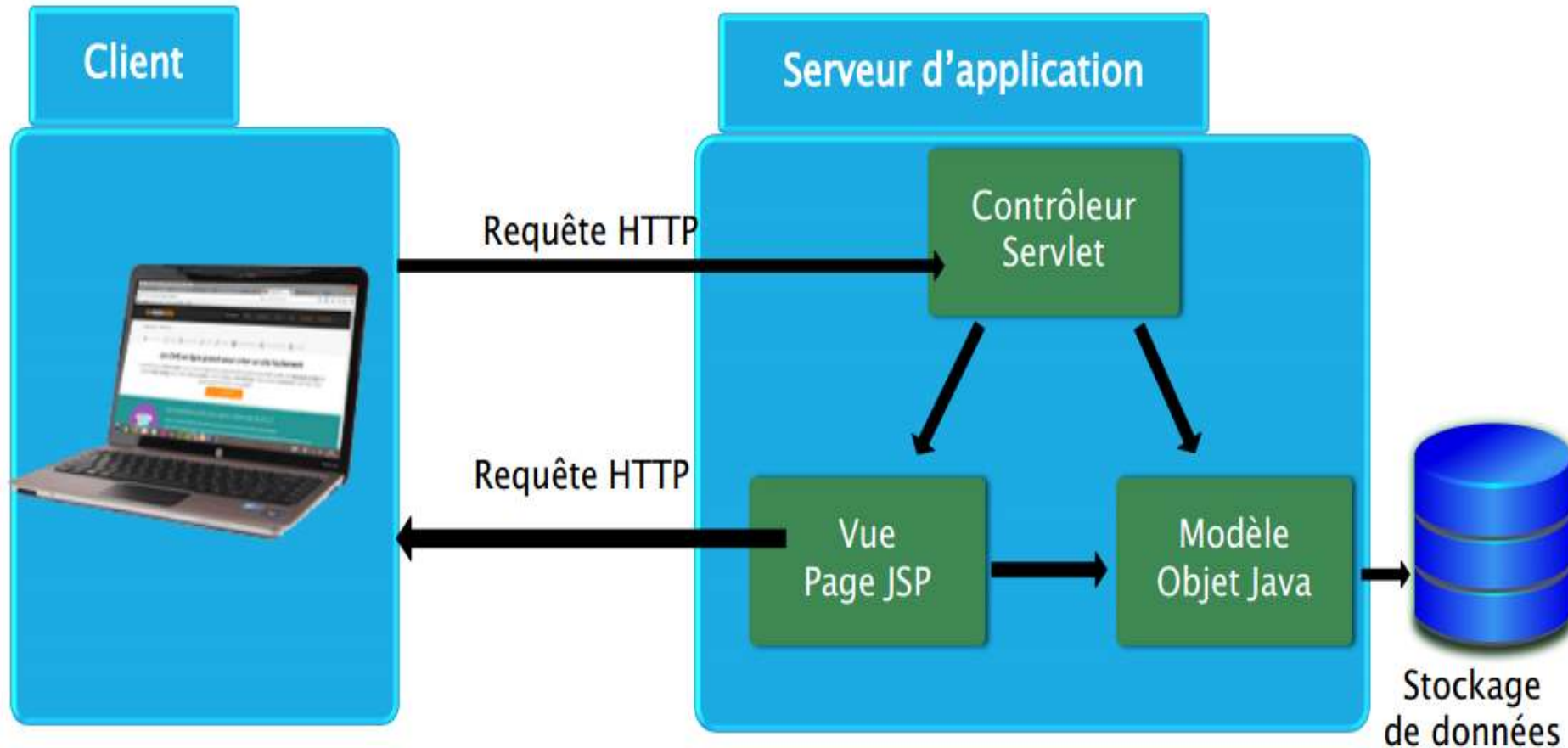
- WebLogic(Oracle)
- WebSphere(IBM)

☐ Les solutions libres et gratuites:

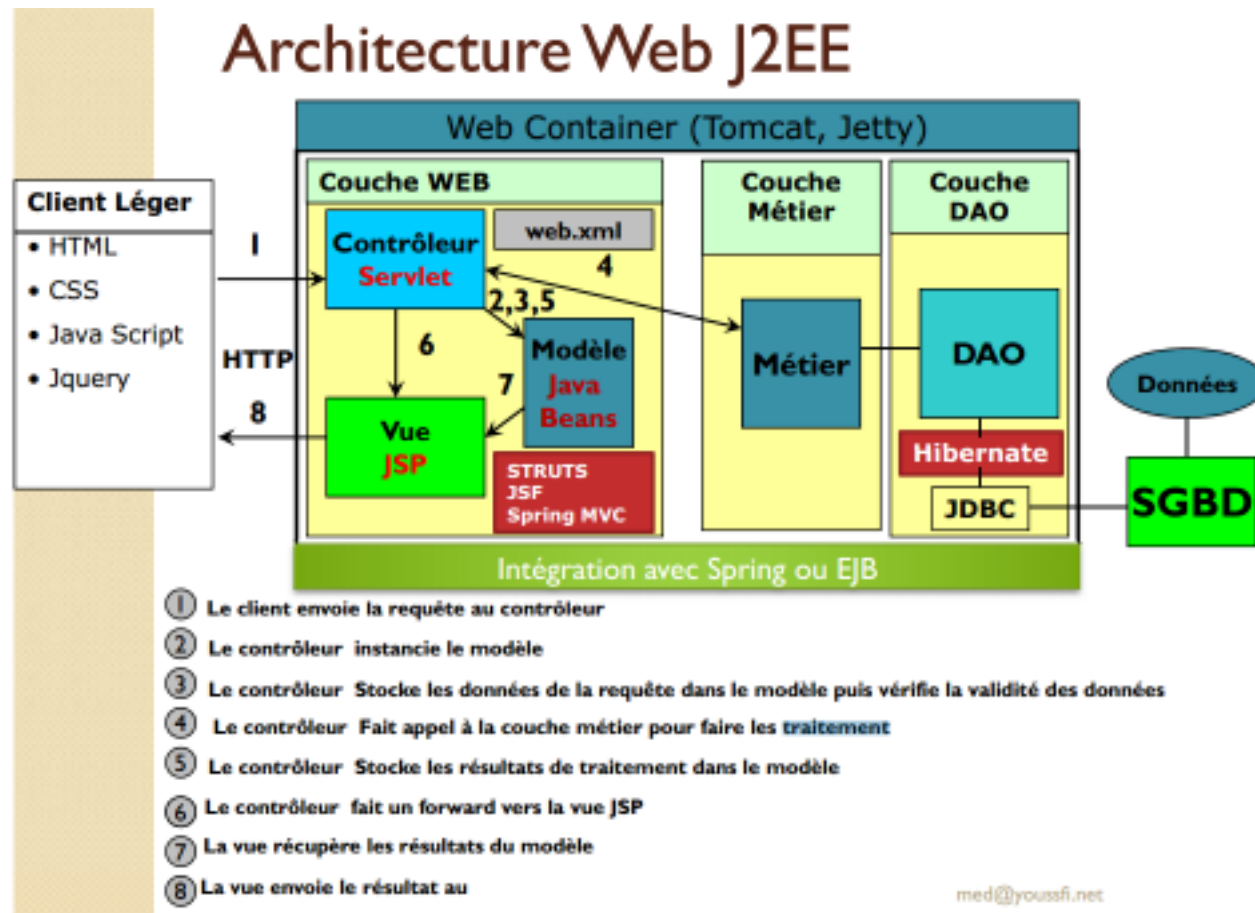
- Apache Tomcat
- Jboss
- GlassFish
- JOnAS

Architecture JEE

Le modèle MVC appliqué au Java EE



Architecture JEE



Préparation de l'environnement

IDE utilisé: Eclipse

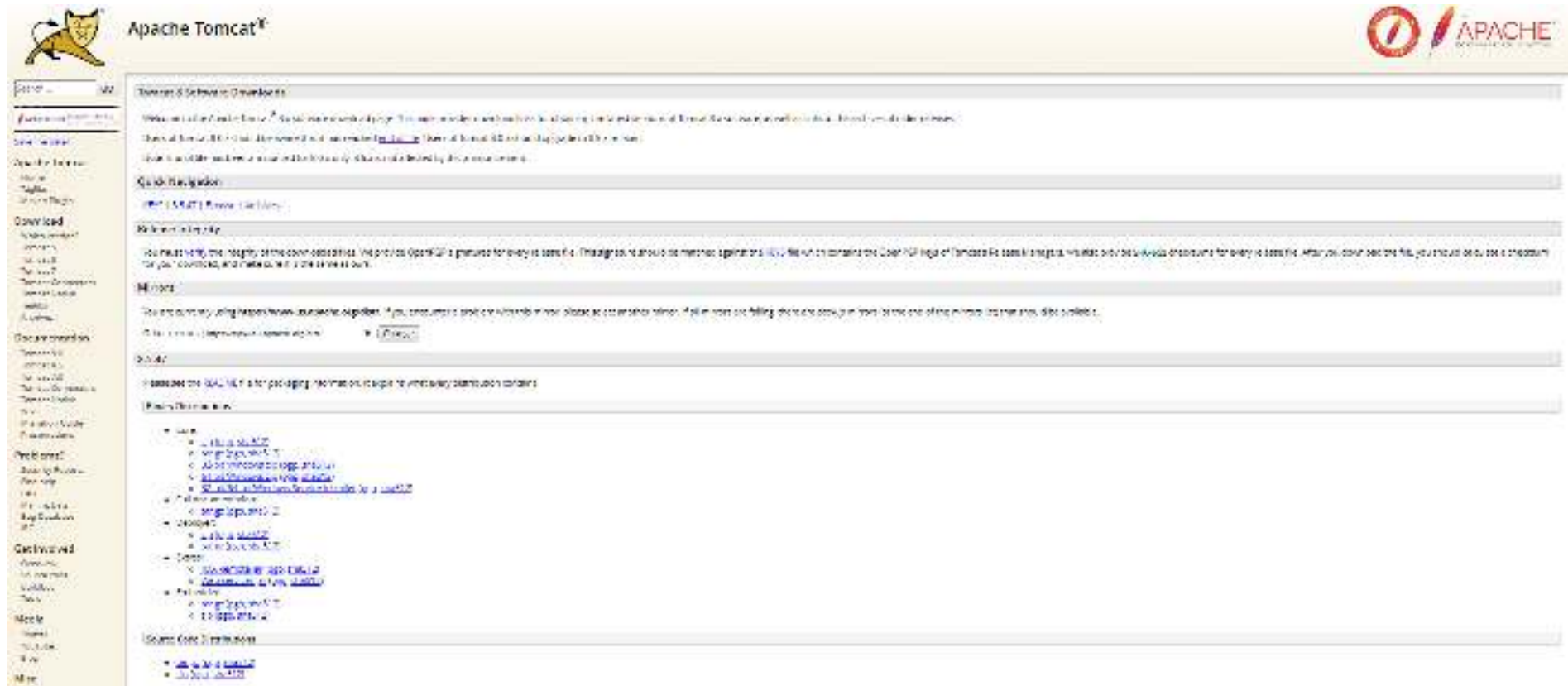
Serveur d'application utilisé: Apache Tomcat

Système d'exploitation: Windows

● Préparation de l'environnement

➔ Apache Tomcat

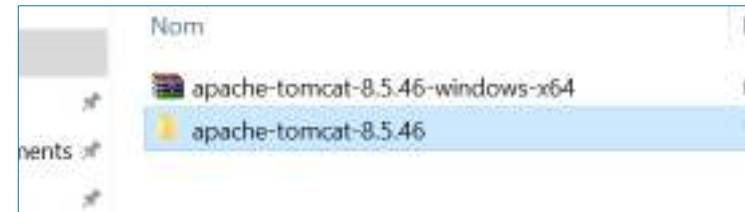
Télécharger Apache Tomcat depuis le site: <https://tomcat.apache.org>



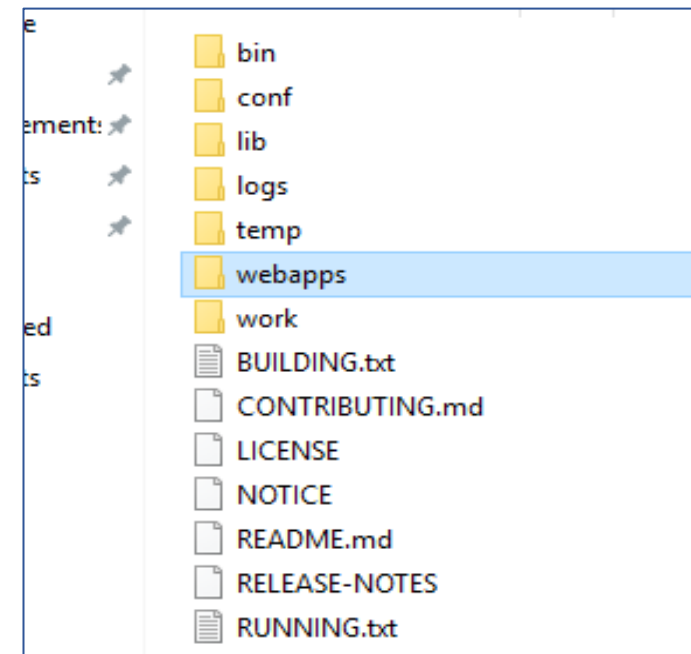
● Préparation de l'environnement

→ Apache Tomcat

Extraire l'archive téléchargé



Dans ce répertoire d'installation de Tomcat, vous trouverez un dossier nommé webapps : c'est ici que seront stockées par défaut vos applications.



● Préparation de l'environnement

➔ Apache Tomcat

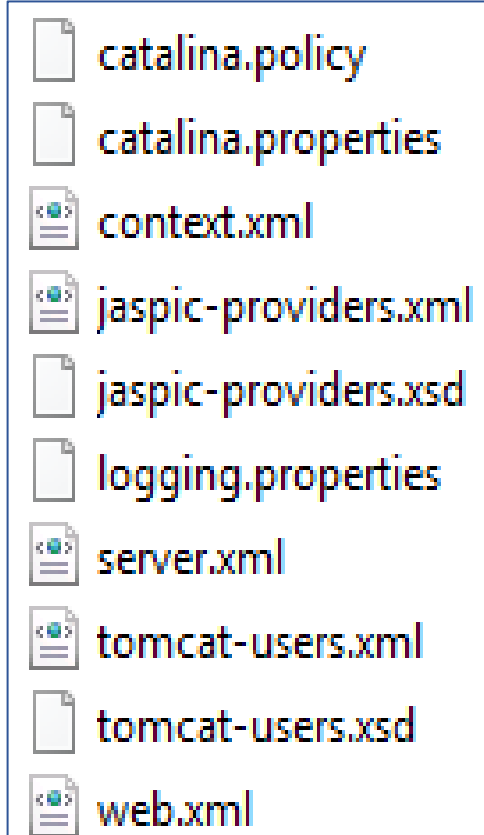
Vous trouverez dans le dossier conf les fichiers suivants :

server.xml : contient les éléments de configuration du serveur ;

context.xml : contient les directives communes à toutes les applications web déployées sur le serveur ;

tomcat-users.xml : contient entre autres l'identifiant et le mot de passe permettant d'accéder à l'interface d'administration de votre serveur Tomcat ;

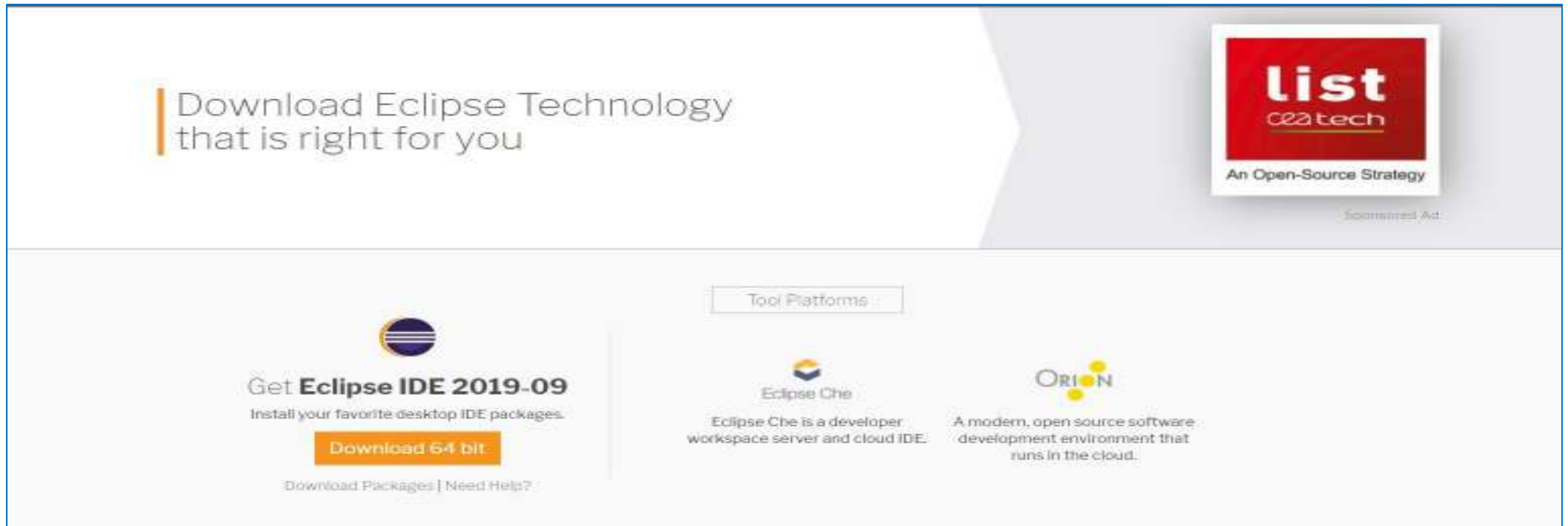
web.xml : contient les paramètres de configuration communs à toutes les applications web déployées sur le serveur.



● Préparation de l'environnement

➔ Eclipse

Télécharger Eclipse depuis le site: <https://www.eclipse.org/downloads/>



The screenshot shows the Eclipse Technology website. At the top, a banner reads "Download Eclipse Technology that is right for you". To the right of the banner is a "list cee tech" logo with the tagline "An Open-Source Strategy" and "Sponsored Ad." below it. Below the banner, there are three main sections. The first section on the left is titled "Get Eclipse IDE 2019-09" and "Install your favorite desktop IDE packages." It features a "Download 64 bit" button and a link "Download Packages | Need Help?". The second section in the middle is titled "Tool Platforms" and features the "Eclipse Che" logo and text: "Eclipse Che is a developer workspace server and cloud IDE." The third section on the right features the "ORION" logo and text: "A modern, open source software development environment that runs in the cloud."

● Préparation de l'environnement

➔ Installer Eclipse

Exécuter Eclipse ensuite choisir Eclipse IDE for Enterprise Java developers

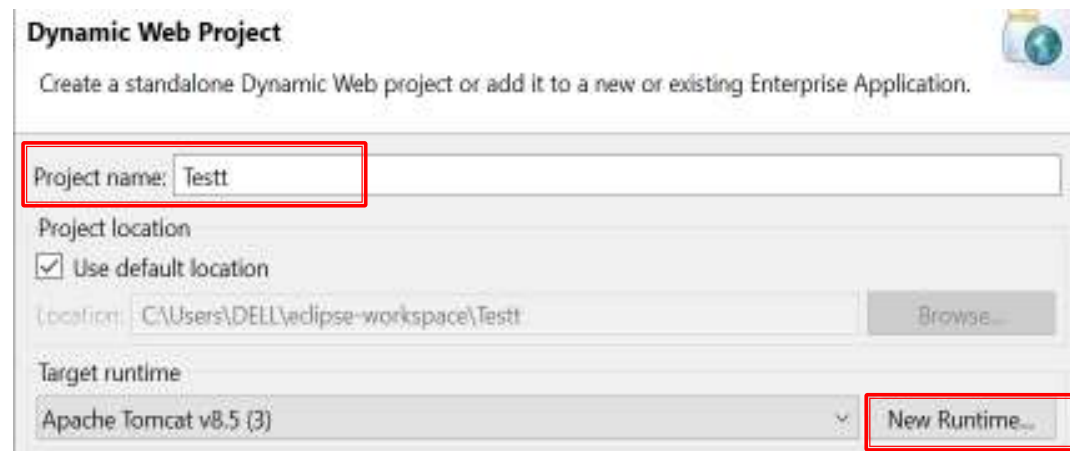
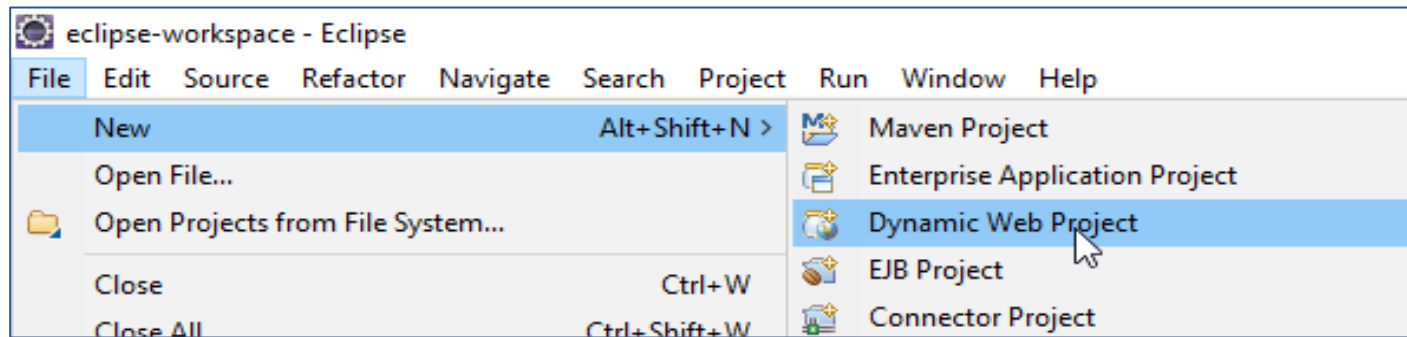


Mise en pratique

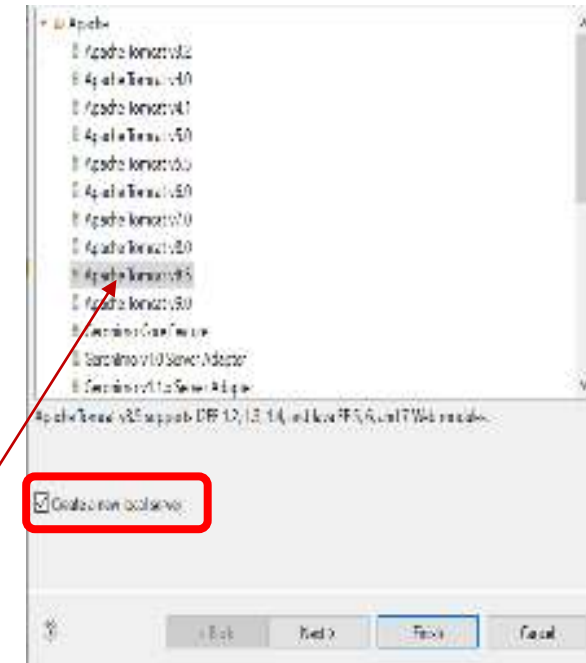
● Préparation de l'environnement

→ Création d'un projet Web -

Depuis Eclipse, suivez le chemin suivant : File > New > Dynamic Web Project



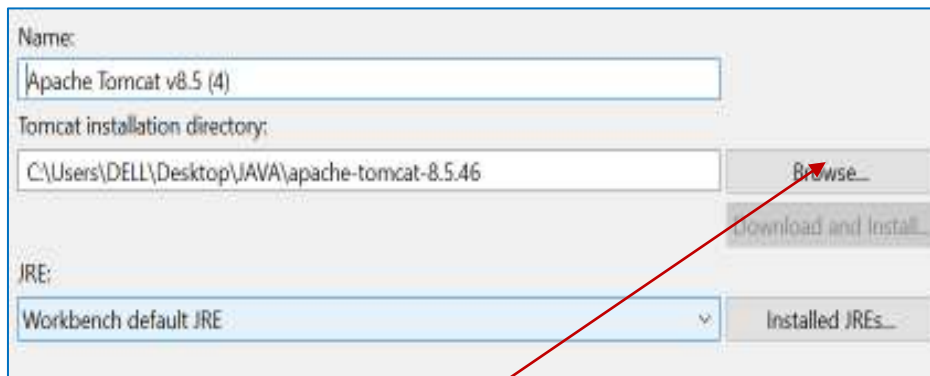
Choisir la version de Tomcat puis Next



Mise en pratique

● Préparation de l'environnement

→ Création d'un projet Web -



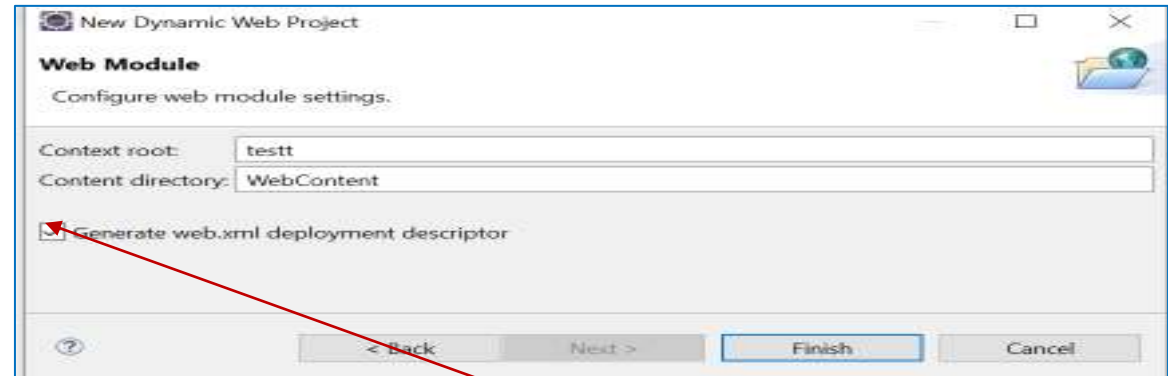
Name: Apache Tomcat v8.5 (4)

Tomcat installation directory: C:\Users\DELL\Desktop\JAVA\apache-tomcat-8.5.46

JRE: Workbench default JRE

Generate web.xml deployment descriptor ☒

Cliquer sur Browse et localiser le chemin de Tomcat



Context root: testt

Content directory: WebContent

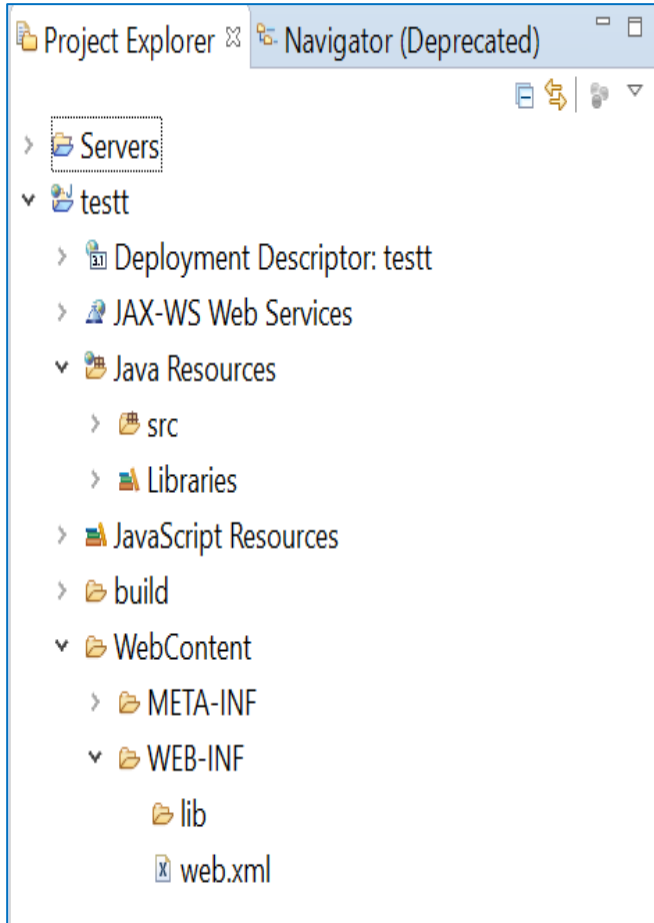
Generate web.xml deployment descriptor ☒

Finish

Cocher cette case pour générer le fichier de configuration Web.xml

● Préparation de l'environnement

→ Création d'un projet Web -



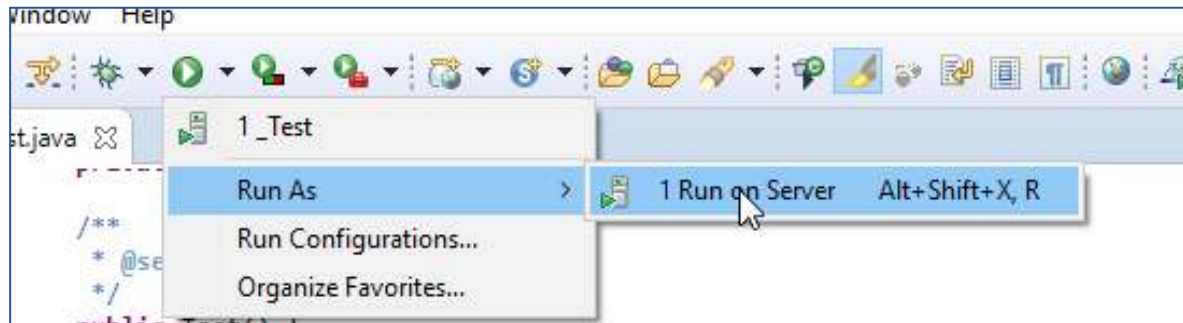
- Le dossier src contient les classes java
- Le byte code est placé dans le dossier build/classes
- Le dossier WebContent contient les documents Web comme les pages HTML, JSP, Images, JavaScript, CSS ...
- Le dossier WEB-INF contient les descripteurs de déploiement comme web.xml
- Le dossier lib permet de stocker les bibliothèques de classes java (Fichiers.jar)

Mise en pratique

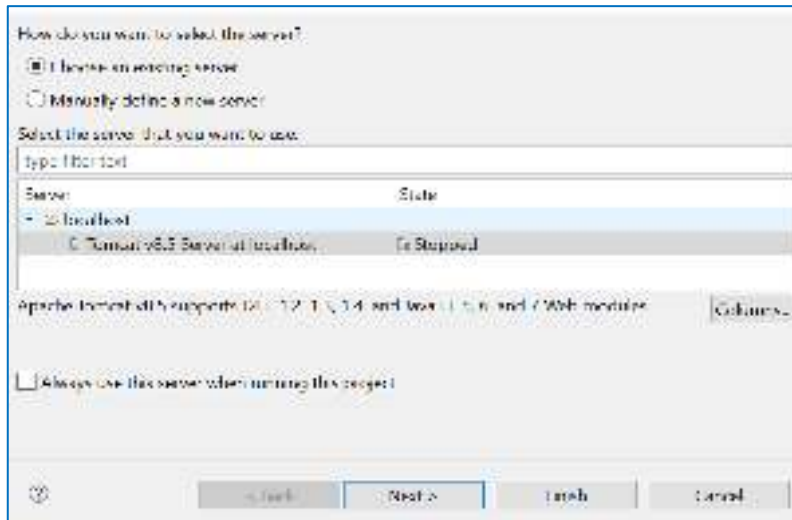
● Préparation de l'environnement

→ Création d'un projet Web -

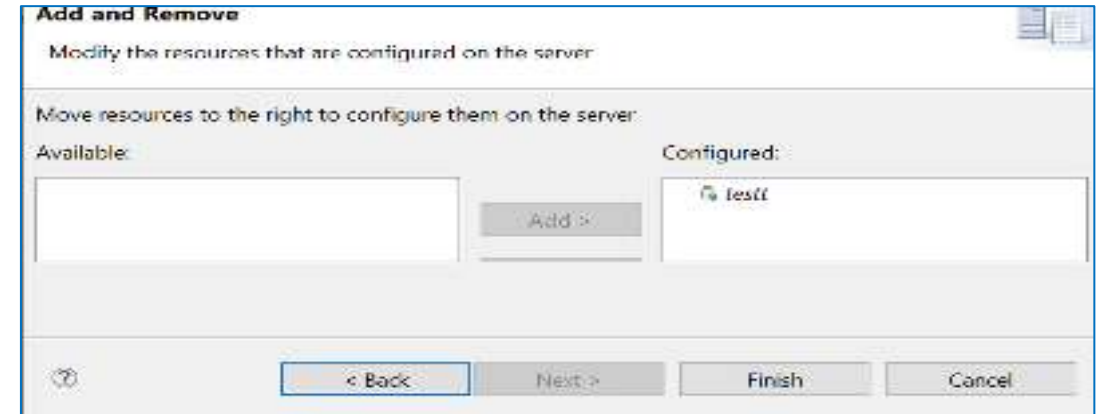
Exécuter le projet



Choisir le serveur puis cliquez sur Next

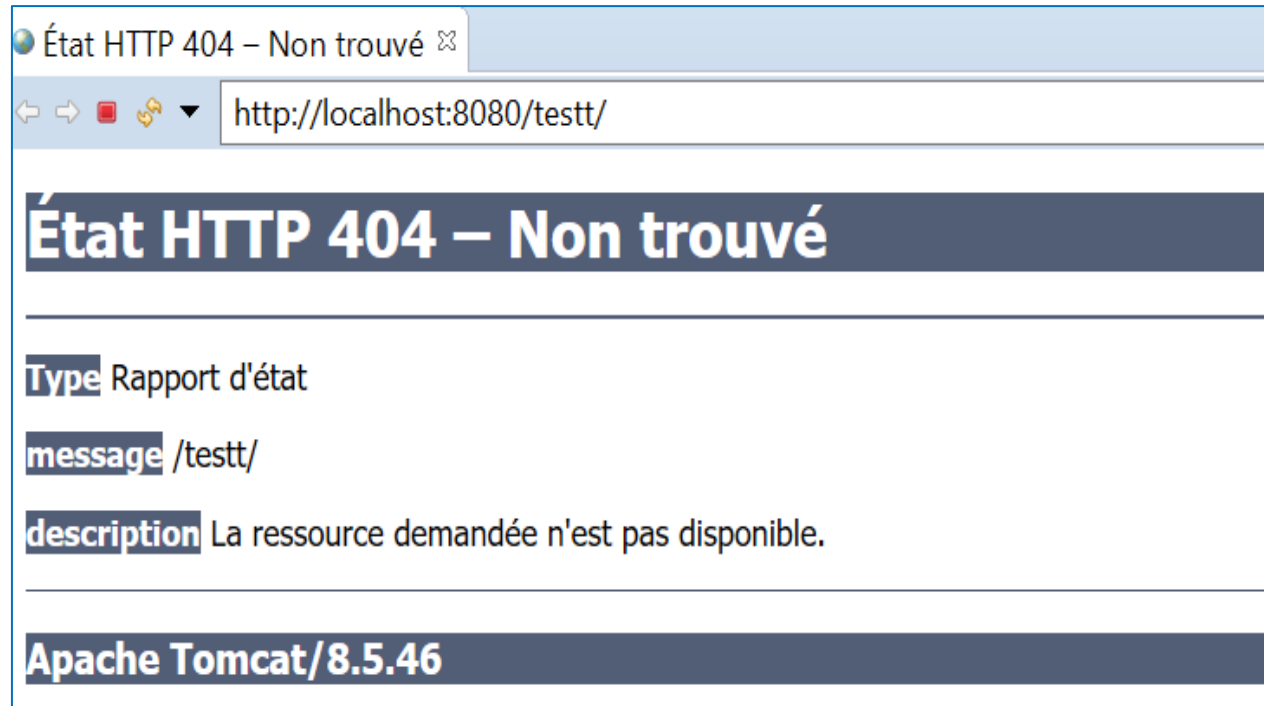


Valider en cliquant sur Finish



● Préparation de l'environnement

→ Création d'un projet Web -



Une erreur
s'affiche, c'est
normal.

Mise en pratique

● Préparation de l'environnement

➔ Création d'un projet Web -

Vous pouvez maintenant placer votre première page HTML dans son dossier public, c'est-à-dire sous le dossier WebContent d'Eclipse



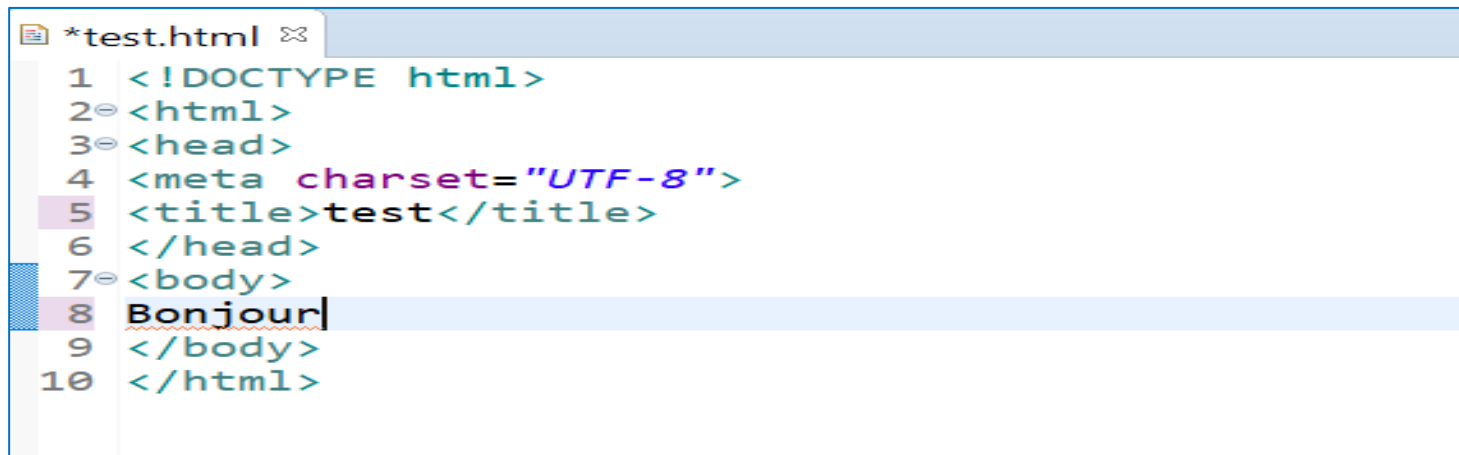
Une page HTML est donc apparue dans votre projet, sous le répertoire WebContent.

Mise en pratique

● Préparation de l'environnement

→ Création d'un projet Web -

Apporter quelques modifications du code HTML comme suit:



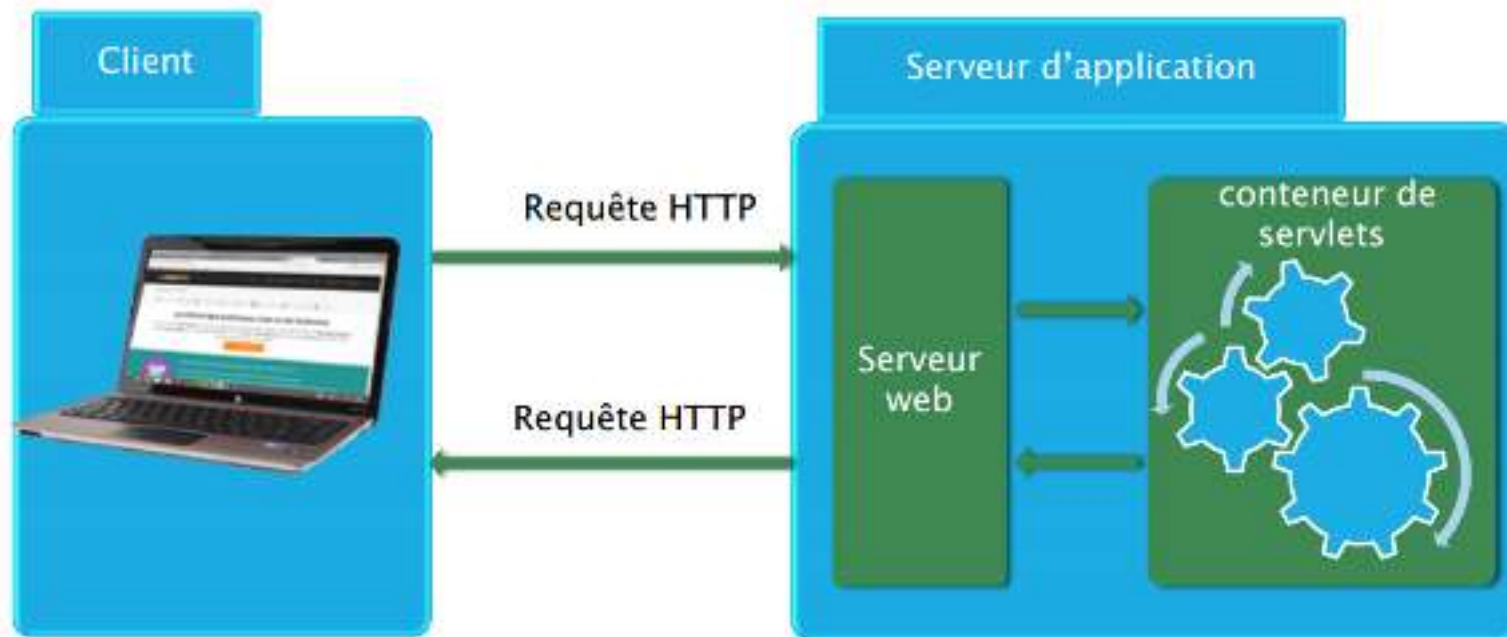
```
*test.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>test</title>
6 </head>
7 <body>
8 Bonjour
9 </body>
10 </html>
```

Exécuter à l'aide de Run As/Run on Server ou bien Entrer le lien <http://localhost:8080/testt/test.html> dans la barre d'adresse de votre navigateur et voir le résultat

Les servlets

C'est quoi une servlet?

- Une servlet est en réalité une simple classe Java, qui a la particularité de permettre le traitement de requêtes et la personnalisation de réponses.
- Une Servlet s'exécute dans un **moteur de Servlet** ou **conteneur de Servlet** permettant d'établir le lien entre la Servlet et le serveur Web



Conteneur de Servlets

- Un conteneur de servlets permet d'établir le lien entre la Servlet et le serveur Web
- Il prend en charge et gère les servlets:
 - chargement de la servlet
 - gestion de son cycle de vie
 - passage des requêtes et des réponses
- Deux types de conteneurs
 - Conteneurs de Servlets autonomes : c'est un serveur Web qui intègre le support des Servlets
 - Conteneurs de Servlets additionnels : fonctionnent comme un plug-in à un serveur Web existant

Conteneur de Servlets

- Un Apache Tomcat est un conteneur web Java EE libre de servlets et JSP. Issu du projet Jakarta, c'est un des nombreux projets de l'Apache Software Foundation.

■ Fonctionnement d'une Servlet

- Une servlet est une classe qui hérite de la classe `HttpServlet`
- `HttpServlet` redéfinit la méthode `service(...)`
 - `service(...)` lit la méthode (GET, POST, ...) à partir de la requête
 - Elle transmet la requête à une méthode appropriée de `HttpServlet` destinée à traiter le type de requête (GET, POST, ...)
- Lorsqu'une servlet est appelée par un client, la méthode `service()` est exécutée. Celle-ci est le principal point d'entrée de toute servlet et accepte deux objets en paramètres:
 - L'objet `HttpServletRequest` encapsulant la requête du client, c'est-à-dire qu'il contient l'ensemble des paramètres passés à la servlet (informations sur l'environnement du client, cookies du client, URL demandée, ...)
 - L'objet `HttpServletResponse` permettant de renvoyer une réponse au client (envoyer des informations au navigateur).
- Une servlet redéfinit les méthodes du protocole HTTP.
 - Si la méthode utilisée est GET, il suffit de redéfinir la méthode :
`public void doGet(HttpServletRequest request, HttpServletResponse response)`
 - Si la méthode utilisée est POST, il suffit de redéfinir la méthode :
`public void doPost(HttpServletRequest request, HttpServletResponse response)`

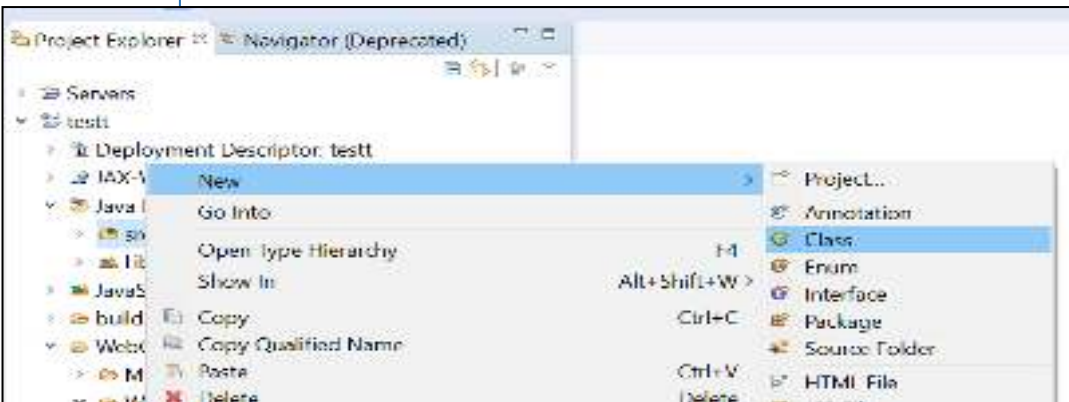
Les servlets

C'est quoi une servlet?

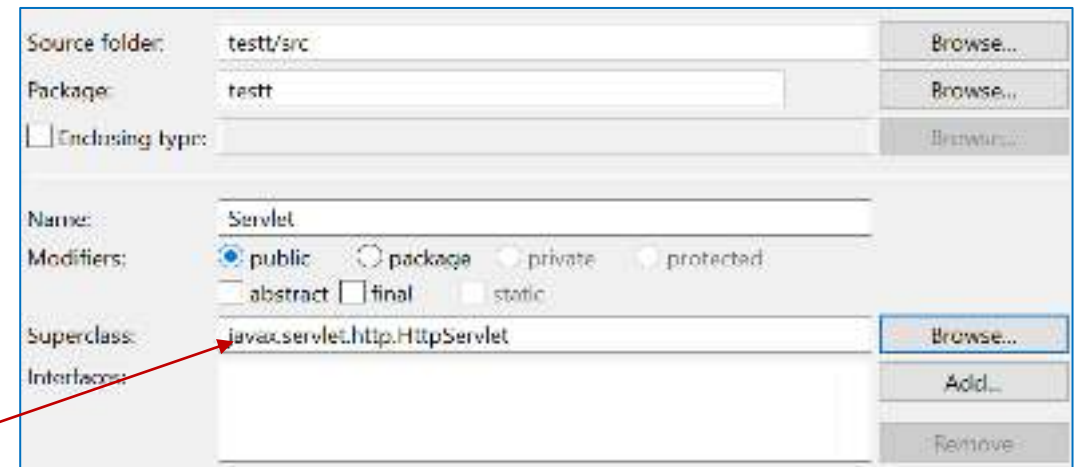
Une servlet est en réalité une simple classe Java, qui a la particularité de permettre le traitement de requêtes et la personnalisation de réponses.

→ Création

Dans le répertoire « Java ressources → src » du projet courant, créer une classe



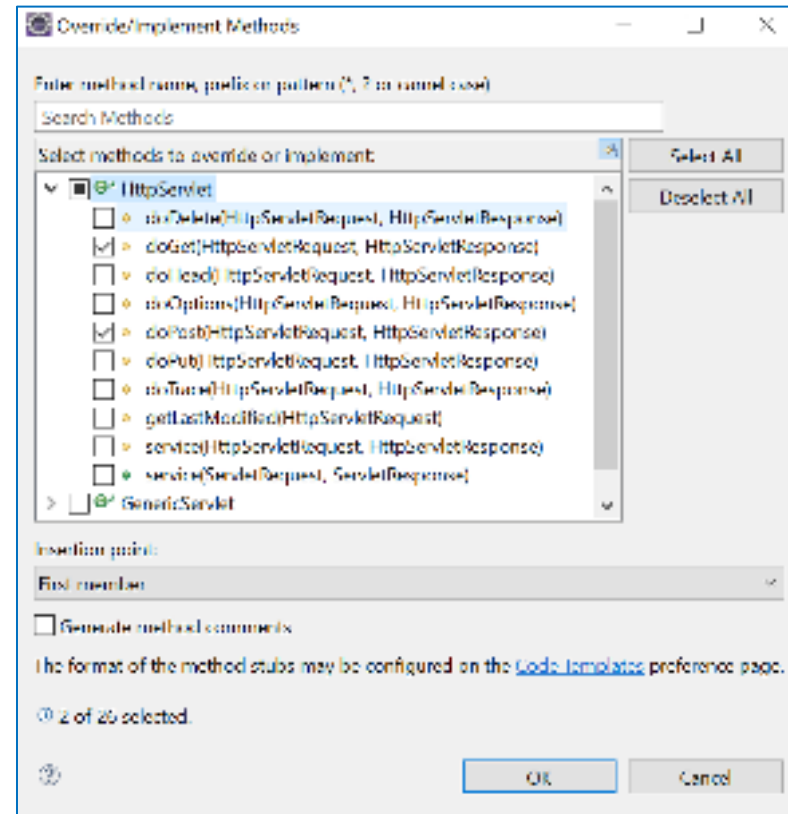
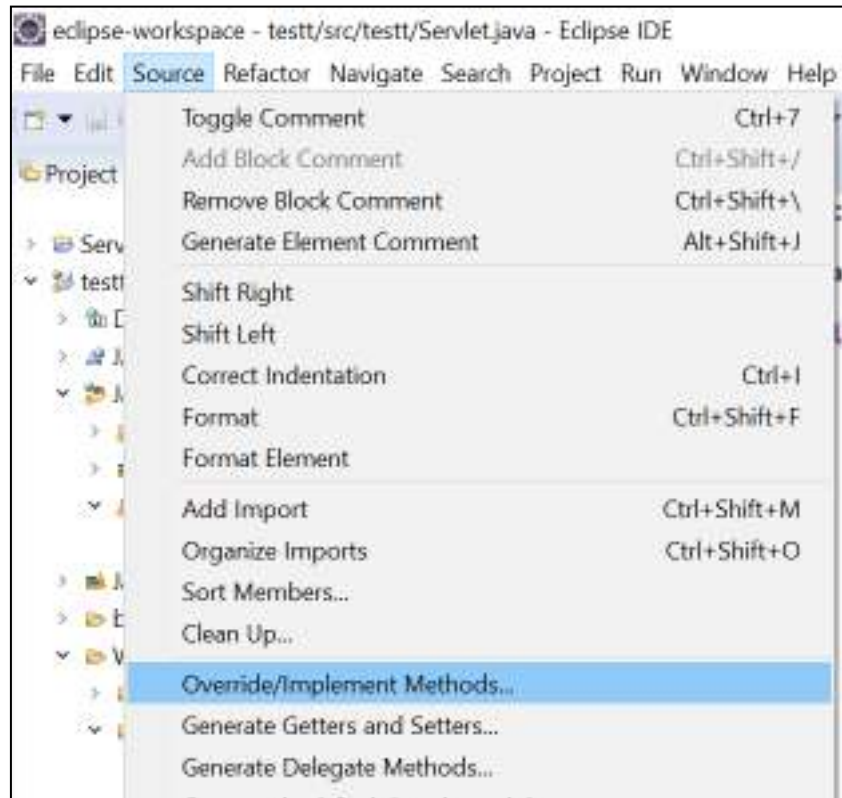
une servlet HTTP doit hériter de la classe abstraite HttpServlet ;



Les servlets

Les méthodes doXXX()

Une servlet doit implémenter au moins une des méthodes doXXX(), afin d'être capable de traiter une requête entrante. Donc, il faut implémenter les méthodes doXXX() à la classe Servlet.

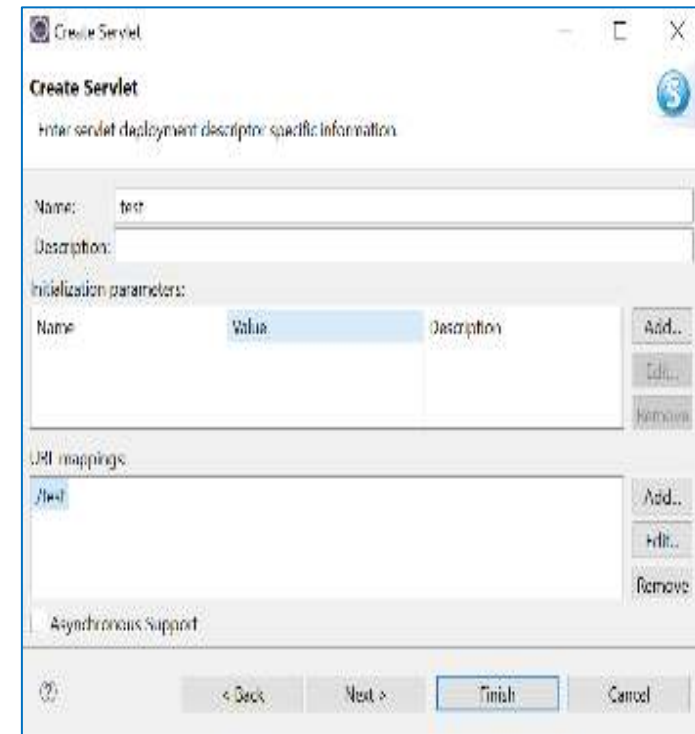
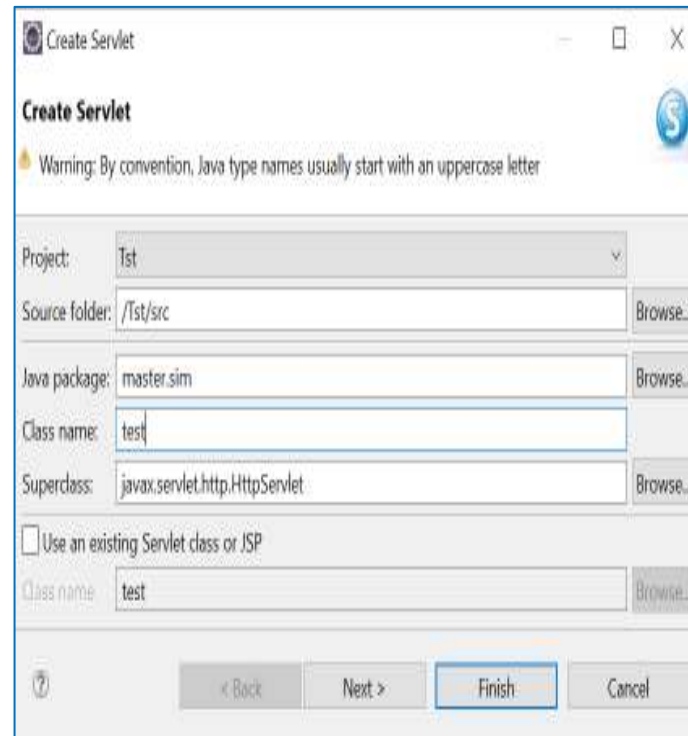
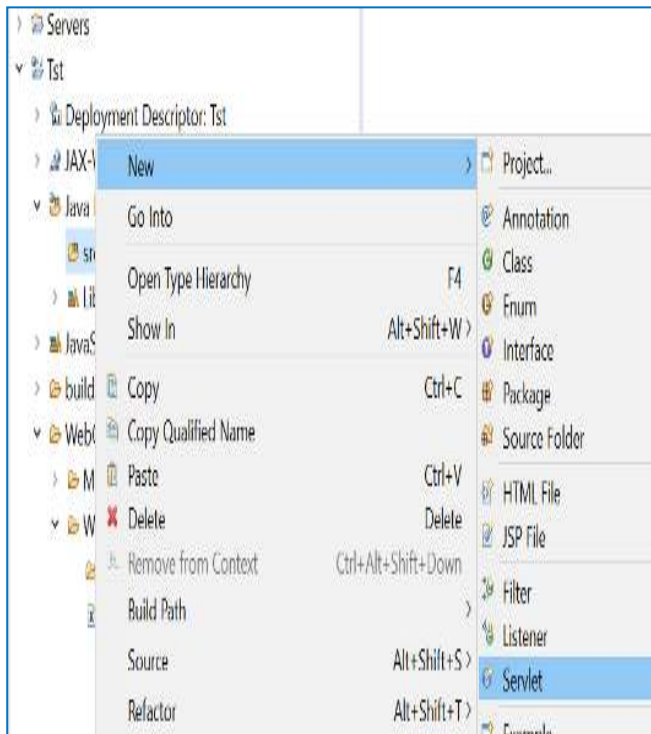


Les méthodes doXXX()

```
*Servlet.java
2
3 import java.io.IOException;
4
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9
10 public class Servlet extends HttpServlet {
11
12     @Override
13     protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
14         // TODO Auto-generated method stub
15         super.doGet(req, resp);
16     }
17
18     @Override
19     protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
20         // TODO Auto-generated method stub
21         super.doPost(req, resp);
22     }
23
24 }
25
```

Les servlets

Autre méthode pour créer une Servlet



Déploiement d'une Servlet

- Pour que le serveur Tomcat reconnaisse une servlet, celle-ci doit être déclarée dans le fichier web.xml qui se trouve dans le dossier WEBINF.
- Le fichier web.xml s'appelle le descripteur de déploiement de Servlet.
- Ce descripteur doit déclarer principalement les éléments suivant :
 - ✓ Le nom attribué à cette servlet
 - ✓ La classe de la servlet
 - ✓ Le nom URL à utiliser pour faire appel à cette servlet via le protocole HTTP.

Les servlets

Configuration (Dans le fichier **web.xml**)

Editer le fichier web.xml

Ajouter les balises `<servlet>` et `<servlet-mapping>`


*test.java ✕ *web.xml ✕

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3   <servlet>
4     <servlet-name>test</servlet-name>
5     <servlet-class>master.sim.test</servlet-class>
6   </servlet>
7   <servlet-mapping>
8     <servlet-name>test</servlet-name>
9     <url-pattern>/test</url-pattern>
10  </servlet-mapping>
11 </web-app>
```

Remarque: Il existe une autre méthode de configuration en utilisant les annotations

Les servlets

Configuration (Avec l'annotation `@WebServlet("/ServletTest")`)



The screenshot shows a code editor with two tabs: `*test.java` and `*web.xml`. The `*test.java` tab is active, displaying the following code:

```
1 package master.sim;
2
3 import java.io.IOException;
4
5
6
7
8
9
10 @WebServlet("/test")
11 public class test extends HttpServlet {
12     private static final long serialVersionUID = 1L;
13
14     public test() {
15         super();
16         // TODO Auto-generated constructor stub
17     }
18
19
20     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
21         // TODO Auto-generated method stub
22         response.getWriter().append("Served at: ").append(request.getContextPath());
23     }
24 }
```

A red box highlights the `@WebServlet("/test")` annotation on line 10. A red arrow points from a yellow callout box to this annotation.

Cette simple annotation suffit

• Test



C'est normal, car le serveur http ne renvoie aucune réponse. Il faut donc, aller vers la méthode doGet (ou doPost) et saisir le code de renvoi au client.

Test

Voilà ce qu'il faut écrire pour un premier test

Définir le format de la réponse

Définir l'encodage de la réponse

PrintWriter pour écrire la réponse

```
1
2 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
3
4     response.setContentType("text/html");
5     response.setCharacterEncoding("UTF-8");
6     PrintWriter out = response.getWriter();
7     out.println("<!DOCTYPE html>");
8     out.println("<html>");
9     out.println("<head>");
10    out.println("<meta charset=\"utf-8\" />");
11    out.println("<title> master SIM</title>");
12    out.println("<head>");
13    out.p
14    out.println("<p>Bonjour </p>");
15    out.println("</body>");
16    out.println("</html>");
17
18 }
19
```

Maintenant, on peut voir la réponse

Critique

```
1 package master.sim;
2
3 import java.io.IOException;
4
5
6
7
8
9
10
11
12 //@WebServlet("/test")
13 public class test extends HttpServlet {
14     private static final long serialVersionUID = 1L;
15
16     public test() {
17         super();
18     }
19     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
20     {
21         response.setContentType("text/html");
22         response.setCharacterEncoding("UTF-8");
23         PrintWriter out = response.getWriter();
24         out.println("<!DOCTYPE html>");
25         out.println("<html>");
26         out.println("<head>");
27         out.println("<meta charset=\"utf-8\" />");
28         out.println("<title> master SIM</title>");
29         out.println("<head>");
30         out.println("<body>");
31         out.println("<p>Bonjour </p>");
32         out.println("</body>");
33         out.println("</html>");
34     }
35     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
36     {
37         // TODO Auto-generated method stub
38         doGet(request, response);
39     }
40 }
```

On remarque que le modèle MVC n'est pas respecté

● Les vues

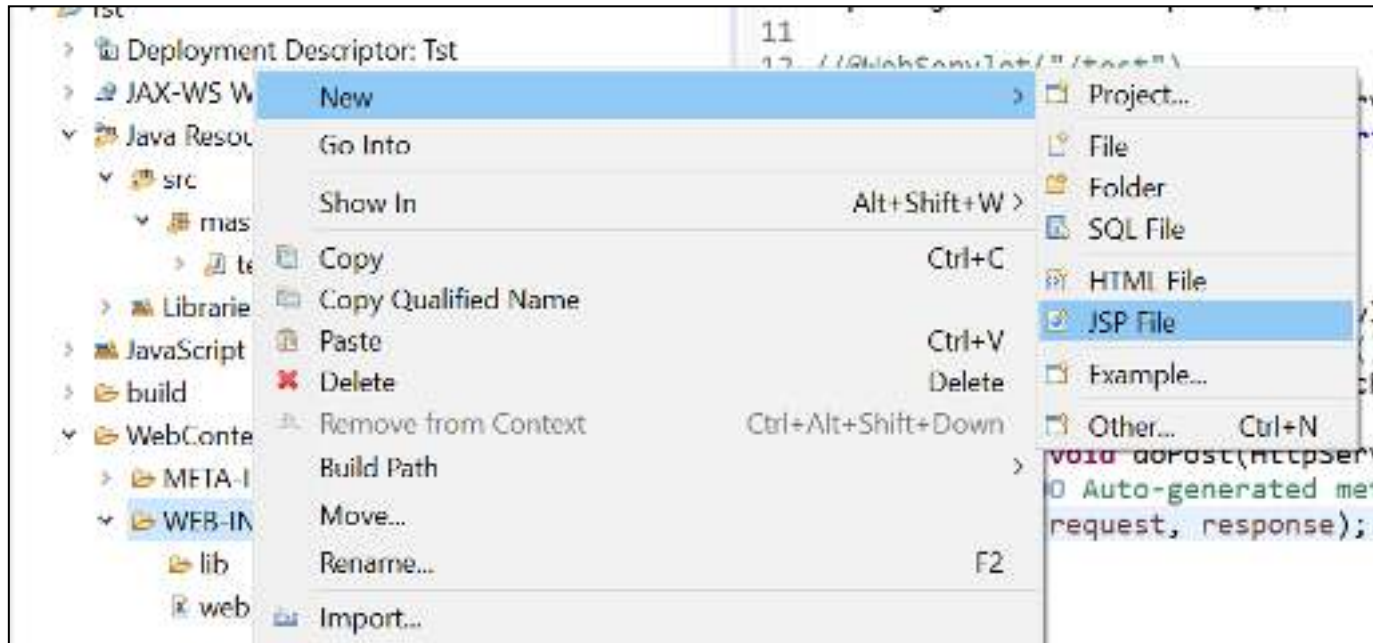
➔ Les pages JSP

- JSP (Java Server Pages) est une technologie Java qui permet la génération de pages web dynamiques
- Une page JSP est un fichier qui porte l'extension *.jsp*
- Une page JSP peut contenir des balises HTML
- Une page JSP contient la balise JSP `<% code Java %>`
- Le code Java inclus est exécuté par le serveur

Les pages JSP

Les vues

→ Les pages JSP



Les pages JSP

Les vues

→ Les pages JSP

Exemple 1

```
*test.java  *Bonjour.jsp x
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5   <head>
6     <title> master SIM</title>
7   </head>
8   <body>
9     <p>Bonjour Etudiants du master SIM </p>
10  </body>
11 </html>
```

Exemple 2

```
test.java  Bonjour.jsp x
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Master SIM</title>
8 </head>
9 <body>
10 <% for (int i=1; i<4; i++){
11   out.println("Bonjour à vous");
12 }
13 %>
14 </body>
15 </html>
```

Les pages JSP

Redirection depuis la servlet vers une vue JSP

Servlet.java

```
1 package master.sim;
2
3 import java.io.IOException;
4
11
12 @WebServlet("/test")
13 public class test extends HttpServlet {
14     private static final long serialVersionUID = 1L;
15
16     public test() {
17         super();
18     }
19     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
20         //this.getServletContext().getRequestDispatcher("/WEB-INF/Bonjour.jsp").forward(request, response);
21         request.getRequestDispatcher("/WEB-INF/Bonjour.jsp").forward(request, response);
22     }
23     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
24         // TODO Auto-generated method stub
25         doGet(request, response);
26     }
27 }
```

Bonjour.jsp

```
*test.java *Bonjour.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5     <head>
6         <title> master SIM</title>
7     </head>
8     <body>
9         <p>Bonjour Etudiants du master SIM </p>
10    </body>
11 </html>
```


Transmission de données

Données issues du serveur : les attributs

→ Transmettre des variables de la servlet à la JSP

Servlet.java

```
1 package master.sim;
2
3 import java.io.IOException;
4
11
12 @WebServlet("/test")
13 public class test extends HttpServlet {
14     private static final long serialVersionUID = 1L;
15
16     public test() {
17         super();
18     }
19     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
20     {
21         String nom = "Mohamed";
22         request.setAttribute("St", nom);
23         request.getRequestDispatcher("/WEB-INF/Bonjour.jsp").forward(request, response);
24     }
25     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
26     {
27         // TODO Auto-generated method stub
28         doGet(request, response);
29     }
30 }
```

Bonjour.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Master SIM</title>
8 </head>
9 <body>
10 <p>Bonjour
11 <%
12     String a = (String) request.getAttribute("St");
13     out.println( a );
14 %>
15 </p>
16 </body>
17 </html>
```

Transmission de données

- Données issues du client : les paramètres
 - ➔ Récupération des paramètres par le serveur

Servlet.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
String Nom = request.getParameter("nom");
request.setAttribute("nm", Nom);
request.getRequestDispatcher("/WEB-INF/PageJSP.jsp").forward(request, response);
}
```

PageJSP.jsp

```
<body>
<p> Bonjour
<%
String nom = (String) request.getAttribute("nm");
out.print(nom);
%>
</p>
</body>
```

Pour tester la servlet, taper l'url suivant: *http://localhost:8080/Servlet?nom=Alami*

● Le modèle

➔ Java Beans

Un JavaBean désigne tout simplement un composant réutilisable. C'est un simple objet Java qui suit certaines contraintes, et représente généralement des données du monde réel.

Un bean :

- doit être une **classe publique** ;
- doit avoir au moins un constructeur vide (sans argument);
 - On peut satisfaire cette contrainte soit en définissant explicitement un tel constructeur, soit en ne spécifiant aucun constructeur
- ne doit pas avoir de champs publics ;
- peut définir des propriétés (des champs non publics), qui doivent être accessibles via des méthodes publiques getter et setter.

Java Beans

Exemple de Java Bean

Propriétés privées

Constructeur sans paramètres

Getters et Setters

```
package master.sim;
public class Etudiant {
    private String nom;
    private String prenom;
    public Etudiant() {
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String
    prenom) {
        this.prenom = prenom;
    }
}
```

● Utilisation du Bean (modification de Servlet)

```
@WebServlet("/sam")
public class sam extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        master.sim.Etudiant etudiant = new master.sim.Etudiant();
        etudiant.setNom("Tahiri");
        etudiant.setPrenom("Mohamed");
        request.setAttribute("etudiant", etudiant);
        request.getRequestDispatcher("/WEB-INF/sam1.jsp").forward(request,
response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```



Utilisation du Bean (modification de la page JSP)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>titre1</title>
</head>
<body>
<p> Bonjour
<%
master.sim.Etudiant etudiant = (master.sim.Etudiant)
request.getAttribute("etudiant");
out.print(etudiant.getNom()+" "+etudiant.getPrenom());
%>
</p>
</body>
</html>
```

Exemples

● Exemple1: Utilisation du Bean (Bean)

```
package master.sim;
public class Etudiant {
    private String nom;
    private String prenom;
    public Etudiant() {
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String
        prenom) {
        this.prenom = prenom;
    }
}
```

Exemples

● Exemple1: Utilisation du Bean (Servlet)

```
@WebServlet("/sam")
public class sam extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        request.getRequestDispatcher("/WEB-INF/sam1.jsp").forward(request,
response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Exemples

● Exemple1: Utilisation du Bean (page JSP)

```
<body>
<p> Bonjour
<%
master.sim.Etudiant etudiant = new master.sim.Etudiant();
etudiant.setNom("Tahiri");
etudiant.setPrenom("Mohamed");
request.setAttribute("etudiant", etudiant);
out.print(etudiant.getNom()+" "+etudiant.getPrenom());
%>
</p>
</body>
</html>
```



Exemple2

1. Créer une page jsp (formulaire.jsp) contenant un formulaire avec les champs suivants:
 - Nom (zone de texte)
 - Prénom (zone de texte)
 - Sexe (bouton radio M ou F)
 - Loisirs (cases à cocher)
 - Lecture
 - Sport
 - Voyage
 - Bouton de validation
2. Créer une Servlet (Servlet.java) qui permet de récupérer le contenu du formulaire saisi après sa validation par l'utilisateur et de Rediriger vers une autre page jsp (affichage.jsp), qui affiche la ou les valeurs saisies pour champ de formulaire.

Exemples

● Exemple2

Servlet.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    request.getRequestDispatcher("/WEB-INF/formulaire.jsp").forward(request,
response);
}
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    String Nom = request.getParameter("nom");
    request.setAttribute("Nom", Nom);
    String Prenom = request.getParameter("prenom");
    request.setAttribute("Preom", Prenom);
    String Sexe = request.getParameter("sexe");
    request.setAttribute("Sexe", Sexe);
    String Loisirs = request.getParameter("Loisirs");
    request.setAttribute("Loisirs", Loisirs);
    request.getRequestDispatcher("/WEB-INF/affichage.jsp").forward(request,
response);
}
}
```

Exemples



Exemple2

formulaire.jsp

```
<body>
<form action="/exemple2/Servlet" method="POST">
Nom: <input type="text" name="nom">
<br/>
Prénom: <input type="text" name="prenom" />
<br/>
Sexe:
Homme:<input type = "radio" name = "sexe" value = "M">
Femme:<input type = "radio" name = "sexe" value = "F">
<br/>
Loisirs:
Lecture : <input type = "checkbox" Name = "Loisirs" Value = "Lecture">
Sport : <input type = "checkbox" Name = "Loisirs" Value = "Sport">
Voyage : <input type = "checkbox" Name = "Loisirs" Value = "Voyage">
<br/>
<input type="submit" value="Envoyer" />
</form>
</body>
```

Exemples

● Exemple2

affichage.jsp

```
<body>
<p>
Bonjour
<%
String Nom = (String) request.getAttribute("Nom");
String Prenom = (String) request.getAttribute("Prenom");
String Sexe = (String) request.getAttribute("Sexe");
String Loisirs = (String) request.getAttribute("Loisirs");
if (Sexe.equals("M")){
out.println("Monsieur");
}else{
out.println("Madame");
}
out.println(Nom+" "+Prenom+".");
out.println("votre loisir est:"+Loisirs );
%>
</p>
</body>
```

Exemple3

1. Créer une page jsp (identification.jsp) contenant un formulaire avec :
 - Deux champs de texte nommés login et mot de passe
 - Bouton de validation
2. Créer une Servlet (Identification.java) qui traite le formulaire d'identification et qui implante la logique suivante:
 - Si le login est égale à « admin » et le mot de passe est égale à « admin »
 - Rediriger la requête vers la page UtilisateurValide.jsp
 - Sinon
 - Rediriger la requête vers la page UtilisateurNonValide.jsp
3. La page UtilisateurValide.jsp affiche le texte suivant: « Bonjour ».
4. La page UtilisateurNonValide.jsp affiche le texte suivant:
 - Echec d'identification;
 - Un lien hypertexte vers la page d'identification

Examples

● Example3

Identification.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    request.getRequestDispatcher("/WEB-INF/identification.jsp").forward(request, response);
}
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String login = request.getParameter("login");
    //request.setAttribute("login", login);
    String password = request.getParameter("password");
    //request.setAttribute("password", password);
    if (login.equals("admin") && password.equals("admin")) {
        request.getRequestDispatcher("/WEB-INF/UtilisateurValide.jsp").forward(request, response);
    } else {
        request.getRequestDispatcher("/WEB-INF/UtilisateurNonValide.jsp").forward(request,
        response);
    }
}
}
```

Exemples

● Exemple 3

Identification.jsp

```
<body>
<form action="/exemple2/Identification" method="POST">
login : <input type="text" name="Login"/>
<br>
mot de passe : <input type="text" name="password"/>
<br>
<input type="submit" value="envoyer"/>
</form>
</body>
```

Exemples

Exemple 3

UtilisateurValide.jsp

```
<body>  
<p>Bonjour</p>  
</body>
```

UtilisateurNonValide.jsp

```
<body>  
<p>Echec d'identification</p>  
<a href="/WEB-INF/identification.jsp">  
page d'identification</a>  
</body>
```



Chapitre 4

La technologie JSP



Les Balises de commentaire

- Tout comme dans les langages Java et HTML, il est possible d'écrire des commentaires dans le code de vos pages JSP. Ils doivent être compris entre les balises `<%-- et --%>`.

Exemple:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Exemple</title>
  </head>
  <body>
    <%-- Ceci est un commentaire JSP, non visible dans la page
    HTML finale. --%>
    <!-- Ceci est un simple commentaire HTML. -->
    <p>Ceci est un simple texte.</p>
  </body>
</html>
```

Les Balises de déclaration

- Cette balise vous permet de déclarer une variable à l'intérieur d'une JSP.

Exemple:

```
<%! String chaine = "Salut."; %>
```

Les expressions

- Les expressions JSP sont, des expressions Java qui vont être évaluées à l'intérieur d'un appel de méthode *print*.
- Une expression commence par les caractères `<%=` et se termine par les caractères `%>`.
- Comme l'expression est placée dans un appel de méthode, il est interdit de terminer l'expression via un point-virgule.

Syntaxe: `<%=expression%>`

Equivalent à: `out.println(expression) ;`

Exemple : `<%=new Date()%>`

Equivalent à: `out.println(new Date()) ;`

Les expressions

Exemple

Servlet

```
@WebServlet("/sam")
public class sam extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    request.getRequestDispatcher("/WEB-INF/sam1.jsp").forward(request,
response);
    }
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
    }
}
```

Page JSP

```
<body>
<p> Bonjour
<%
master.sim.Etudiant etudiant = new master.sim.Etudiant();
etudiant.setNom("Tahiri");
etudiant.setPrenom("Mohamed");
request.setAttribute("etudiant", etudiant);
out.print(etudiant.getNom()+" "+etudiant.getPrenom());
%>
</p>
</body>
</html>
```

Java Bean

```
package master.sim;
public class Etudiant {
    private String nom;
    private String prenom;
    public Etudiant() {
    }
    public String getNom() {
    return nom;
    }
    public void setNom(String nom) {
    this.nom = nom;
    }
    public String getPrenom() {
    return prenom;
    }
    public void setPrenom(String prenom) {
    this.prenom = prenom;
    }
}
```

Les expressions

● Exemple

Page JSP

```
<body>
<p> Bonjour
<%
master.sim.Etudiant etudiant = new master.sim.Etudiant();
etudiant.setNom("Tahiri");
etudiant.setPrenom("Mohamed");
request.setAttribute("etudiant", etudiant);
%>
<%=etudiant.getNom()+" "+etudiant.getPrenom() %>
</p>
</body>
</html>
```

Les directives

Les directives JSP permettent :

- d'importer un package ;
- d'inclure d'autres pages JSP ;
- d'inclure des bibliothèques de balises;
- de définir des propriétés et informations relatives à une page JSP.

Elles contrôlent comment le conteneur de servlets va gérer votre JSP. Il en existe trois : **taglib**, **page** et **include**.

Elles sont toujours comprises entre les balises **<%@** et **%>**, et hormis la directive d'inclusion de page qui peut être placée n'importe où, elles sont à placer en tête de page JSP.

Les directives

→ Directive taglib

Le code ci-dessous inclut une bibliothèque personnalisée nommée maTagLib

```
<% @ taglib uri="maTagLib.tld" prefix="tagExemple" %>
```

→ Directive page

La directive page définit des informations relatives à la page JSP. Voici par exemple comment importer des classes Java :

```
<% @ page import="java.util.List, java.util.Date" %>
```

→ Directive include

```
<% @ include file="uneAutreJSP.jsp" %>
```

● Exemple 1 : Date du jour

date.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    request.getRequestDispatcher("/WEB-INF/date.jsp").forward(request,
response);
}

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


Les directives

● Exemple 1 : Date du jour

date.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.Date" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
Date date =new Date();
out.println("la date du jour est :"+date);
%>
</body>
</html>
```

● Exemple 2 : Menu

menu.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    request.getRequestDispatcher("/WEB-INF/accueil.jsp").forward(request,
response);
}

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

Les directives

Exemple 2 : Menu

menu.jsp

```
<ul>
<li><a href= "/menu/accueil">Accueil</a></li>
<li><a href= "/menu/page1">Page1</a></li>
<li><a href= "/menu/page2">Page2</a></li>
</ul>
```

accueil.jsp

```
<body>
<%@ include file="menu.jsp" %>
<p>Bienvenue sur mon site</p>
</body>
```

page1.jsp

```
<body>
<%@ include file="menu.jsp" %>
<p>page1</p>
</body>
```

page2.jsp

```
<body>
<%@ include file="menu.jsp" %>
<p>page2</p>
</body>
```

● Scriptlets

Les scriptlets correspondent aux blocs de code introduit par le caractères `<%` et se terminant par `%>`.

- Ils servent à ajouter du code dans la méthode de service.
- Le code Java du scriptlet est inséré tel quel dans la servlet générée: la vérification, par le compilateur, du code aura lieu au moment de la compilation totale de la servlet équivalent.
- L'exemple complet de JSP présenté précédemment, comportait quelques scriptlets :

`<%`

```
for(int i=1; i<=6; i++) {  
out.println("<h" + i + " align=\"center\">Entête" +i+ "</h" + i + ">");  
}
```

`%>`

La portée des objets

● La portée des objets

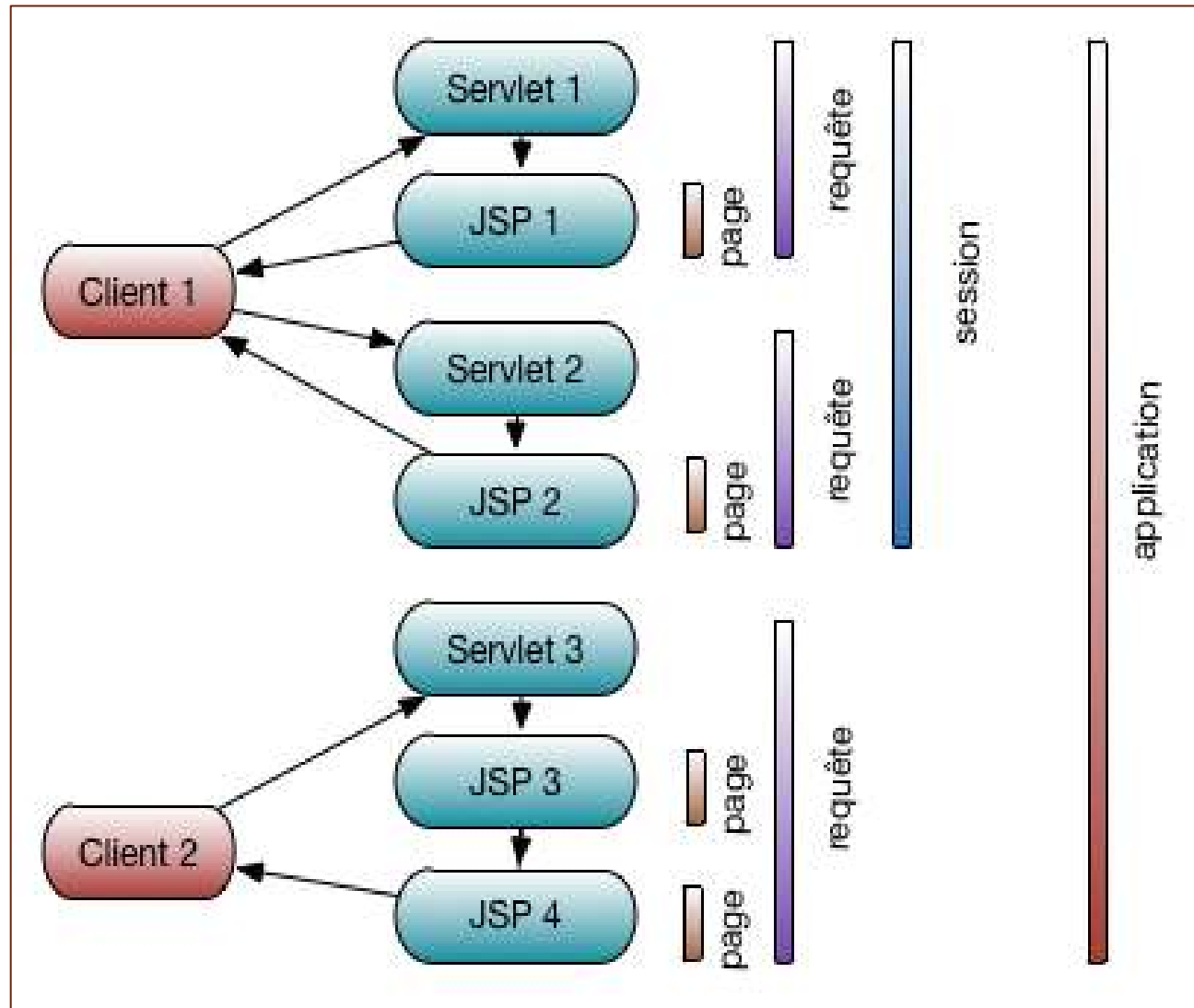
la portée des objets. Souvent appelée visibilité, ou scope en anglais, elle définit tout simplement leur durée de vie.

Il existe au total quatre portées différentes dans une application :

- **Page** (JSP seulement) : les objets dans cette portée sont uniquement accessibles dans la page JSP en question ;
- **Requête** : les objets dans cette portée sont uniquement accessibles durant l'existence de la requête en cours ;
- **Session** : les objets dans cette portée sont accessibles durant l'existence de la session en cours ;
- **Application** : les objets dans cette portée sont accessibles durant toute l'existence de l'application.

La portée des objets

La portée des objets



Les actions standard

● L'actions standard useBean

- Le tag `<jsp:useBean>`
 - Permet de localiser une instance ou bien d'instancier un bean pour l'utiliser dans la JSP

- Syntaxe

`<jsp:useBean id="beanInstanceName" class="package.class" scope="page|request|session|application" />`

Nom utilisé pour accéder au bean dans la page (doit être unique)

Le nom de la classe du bean

la portée durant laquelle le bean est défini et utilisable

Exemple

```
<jsp:useBean id= " etudiant" class=" master.sim.Etudiant " scope="request" />
```

Les actions standard

● L'actions standard useBean

➔setProperty

Permet de modifier une propriété du bean utilisé.

```
<!-- L'action suivante associe une valeur à la propriété 'prenom' du bean  
'etudiant' : --%>
```

```
<jsp:setProperty name="etudiant" property="prenom" value="Mohamed" />
```

```
<!-- Elle a le même effet que le code Java suivant : --%>  
<% etudiant.setPrenom("mohamed"); %>
```


Les actions standard

● L'actions standard useBean

➔ getProperty

Lorsque l'on utilise un bean au sein d'une page, il est possible par le biais de cette action d'obtenir la valeur d'une de ses propriétés :

```
<jsp:useBean id= "etudiant" class="master.sim.Etudiant" />
```

```
<!-- L'action suivante affiche le contenu de la propriété 'prenom'  
du bean 'etudiant' : --%>
```

```
<jsp:getProperty name= "etudiant" property="prenom" />
```

```
<!-- Elle a le même effet que le code Java suivant : --%>
```

```
<%= etudiant.getPrenom() %>
```

Les actions standard

● L'action standard forward

Permet d'effectuer une redirection vers une autre page. Comme toutes les actions standard, elle s'effectue côté serveur et pour cette raison il est impossible via cette balise de rediriger vers une page extérieure à l'application.

```
<!-- Le forwarding vers une page de l'application fonctionne par URL relative : --%>
```

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

```
<!-- Son équivalent en code Java est : --%>
```

```
<% request.getRequestDispatcher("/Servlet").forward( request, response ); %>
```

Notez que lorsque vous utilisez le forwarding, le code présent après cette balise dans la page n'est pas exécuté.

Les actions standard

● Exemple

Servlet

```
@WebServlet("/sam")
public class sam extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    request.getRequestDispatcher("/WEB-INF/sam1.jsp").forward(request,
response);
    }
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
    }
}
```

Page JSP

```
<body>
<p> Bonjour
<%
master.sim.Etudiant etudiant = new master.sim.Etudiant();
etudiant.setNom("Tahiri");
etudiant.setPrenom("Mohamed");
request.setAttribute("etudiant", etudiant);
out.print(etudiant.getNom()+" "+etudiant.getPrenom());
%>
</p>
</body>
</html>
```

Java Bean

```
package master.sim;
public class Etudiant {
    private String nom;
    private String prenom;
    public Etudiant() {
    }
    public String getNom() {
    return nom;
    }
    public void setNom(String nom) {
    this.nom = nom;
    }
    public String getPrenom() {
    return prenom;
    }
    public void setPrenom(String prenom) {
    this.prenom = prenom;
    }
}
```

Les actions standard

Exemple

Page JSP

```
<body>
<p> Bonjour
<jsp:useBean id="etudiant" class="master.sim.Etudiant"/>
<jsp:setProperty name="etudiant" property="nom" value="Tahiri" />
<jsp:setProperty name="etudiant" property="prenom" value="Mohamed" />
<jsp:getProperty name="etudiant" property="nom" />
<jsp:getProperty name="etudiant" property="prenom" />
</p>
</body>
</html>
```

Expression Language (EL)

● Expression Language (EL)

Les expressions EL permettent via une syntaxe très épurée d'effectuer des tests basiques sur des expressions, et de manipuler simplement des objets et attributs dans une page, et cela sans nécessiter l'utilisation de code ni de script Java.

Elles ont la forme suivante: $\$ \{ \textit{expression} \}$

Ce qui est situé entre les accolades va être interprété

Expression Language (EL)

Les opérations

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Test des expressions EL</title>
  </head>
  <body>
    <p>
      <!-- Logiques sur des booléens -->
      ${ true && true } <br /> <!-- Affiche true -->
      ${ true && false } <br /> <!-- Affiche false -->
      ${ !true || false } <br /> <!-- Affiche false -->

      <!-- Calculs arithmétiques -->
      ${ 10 / 4 } <br /> <!-- Affiche 2.5 -->
      ${ 10 mod 4 } <br /> <!-- Affiche le reste de la division entière, soit 2 -->
      ${ 10 % 4 } <br /> <!-- Affiche le reste de la division entière, soit 2 -->
      ${ 6 * 7 } <br /> <!-- Affiche 42 -->
      ${ 63 - 8 } <br /> <!-- Affiche 55 -->
      ${ 12 / -8 } <br /> <!-- Affiche -1.5 -->
      ${ 7 / 0 } <br /> <!-- Affiche Infinity -->
    </p>
  </body>
</html>
```

Expression Language (EL)

Les opérations

```
<!-- Compare les caractères 'a' et 'b'. Le caractère 'a' étant bien situé
avant le caractère 'b' dans l'alphabet ASCII, cette EL affiche true. -->
${ 'a' < 'b' } <br />

<!-- Compare les chaînes 'hip' et 'hit'. Puisque 'p' < 't', cette EL affiche
false. -->
${ 'hip' gt 'hit' } <br />

<!-- Compare les caractères 'a' et 'b', puis les chaînes 'hip' et 'hit'.
Puisque le premier test renvoie true et le second false, le résultat est false. -->
${ 'a' < 'b' && 'hip' gt 'hit' } <br />

<!-- Compare le résultat d'un calcul à une valeur fixe. Ici, 6 x 7 vaut 42 et
non pas 48, le résultat est false. -->
${ 6 * 7 == 48 } <br />
</p>
</body>
</html>
```

Expression Language (EL)

Les tests

```
<!-- Vérifications si vide ou null -->
${ empty 'test' } <!-- La chaîne testée n'est pas vide, le résultat est false -->
${ empty '' } <!-- La chaîne testée est vide, le résultat est true -->
${ !empty '' } <!-- La chaîne testée est vide, le résultat est false -->

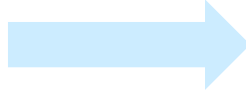
<!-- Conditions ternaires -->
${ true ? 'vrai' : 'faux' } <!-- Le booléen testé vaut true, vrai est affiché -->
${ 'a' > 'b' ? 'oui' : 'non' } <!-- Le résultat de la comparaison vaut false, non
est affiché -->
${ empty 'test' ? 'vide' : 'non vide' } <!-- La chaîne testée n'est pas vide,
non vide est affiché -->
```


La manipulation des beans

- pour accéder à une propriété d'un bean d'une manière simple et efficace:

```
${ bean.propriete }
```

```
<jsp:useBean id="etudiant"  
class="master.sim.Etudiant"/>  
<jsp:getProperty name="  
etudiant" property="nom" />
```



```
${etudiant.nom}
```

où

```
${etudiant.getNom()}
```

Expression Language (EL)

● Exemple

Servlet

```
@WebServlet("/sam")
public class sam extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    request.getRequestDispatcher("/WEB-INF/sam1.jsp").forward(request,
response);
}
    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
}
}
```

Page JSP

```
<body>
<p> Bonjour
<jsp:useBean id="etudiant" class="master.sim.Etudiant"/>
<jsp:setProperty name="etudiant" property="nom" value="Tahiri" />
<jsp:setProperty name="etudiant" property="prenom" value="Mohamed" />
<jsp:getProperty name="etudiant" property="nom" />
<jsp:getProperty name="etudiant" property="prenom" />
</p>
</body>
</html>
```

Java Bean

```
package master.sim;
public class Etudiant {
    private String nom;
    private String prenom;
    public Etudiant() {
    }
    public String getNom() {
    return nom;
    }
    public void setNom(String nom) {
    this.nom = nom;
    }
    public String getPrenom() {
    return prenom;
    }
    public void setPrenom(String prenom) {
    this.prenom = prenom;
    }
}
```

Expression Language (EL)

Exemple

Page JSP

```
<body>
<p> Bonjour
<jsp:useBean id="etudiant" class="master.sim.Etudiant"/>
<jsp:setProperty name="etudiant" property="nom" value="Tahiri" />
<jsp:setProperty name="etudiant" property="prenom" value="Mohamed" />
${etudiant.nom}
${etudiant.prenom}
</p>
</body>
</html>
```

Expression Language (EL)

La manipulation des collections

```
<body>
  <p>
    <%
      java.util.List<String> langages = new java.util.ArrayList<String>();
      langages.add( "Java" );
      langages.add( "Python" );
      request.setAttribute( "langages" , langages );
    %>
    ${ langages.get(0) }<br/>    <!-- Renvoie Java -->
    ${ langages[1] }<br />    <!-- Renvoie Python -->
  </p>
</body>
```

Les objets implicites

Les objets de la technologie JSP

Le conteneur met à notre disposition toute une série d'objets implicites, tous accessibles directement depuis nos pages JSP. En voici la liste :

Identifiant	Type d'objet	Description
request	<i>HttpServletRequest</i>	Permet d'accéder aux paramètres et aux attributs de la requête
response	<i>HttpServletResponse</i>	Permet de définir le Content-Type de la réponse, lui ajouter des en-têtes ou encore pour rediriger le client.
out	<i>JspWriter</i>	Permet d'écrire dans le corps de la réponse.
page	<i>objet this</i>	Il est l'équivalent de la référence this et représente la page JSP courante.
session	<i>HttpSession</i>	Permet de lire ou placer des objets dans la session de l'utilisateur courant.
application	<i>ServletContext</i>	Permet d'obtenir ou de modifier des informations relatives à l'application
config	<i>ServletConfig</i>	Il permet depuis une page JSP d'obtenir les éventuels paramètres d'initialisation
pageContext	<i>PageContext</i>	Permet d'accéder aux attributs des différentes portées de l'application.
exception	<i>Throwable</i>	Il représente l'exception qui a conduit à la page d'erreur en question.

Les objets implicites

Les objets de la technologie JSP

Exemple:

Créer le fichier *exemplejsp.jsp*

```
<body>
  <p>
    <%
      String paramPays = request.getParameter("pays");
      String paramLangue = request.getParameter("langue");

      out.println( "Pays : " + paramPays+"<br>" );
      out.println( "Langue : " + paramLangue );
    %>
  </p>
</body>
</html>
```

Puis, taper l'url suivant:

<http://localhost:8080/WebApp/exemplejsp.jsp?pays=Maroc&langue=Arabe>

Les objets implicites

Les objets de la technologie EL

Catégorie	Identifiant	Description
JSP	<i>PageContext</i>	Objet contenant des informations sur l'environnement du serveur
Portées	<i>pageScope</i>	Une Map qui associe les noms et valeurs des attributs ayant pour portée la page.
	<i>requestScope</i>	Une Map qui associe les noms et valeurs des attributs ayant pour portée la requête.
	<i>sessionScope</i>	Une Map qui associe les noms et valeurs des attributs ayant pour portée la session.
	<i>applicationScope</i>	Une Map qui associe les noms et valeurs des attributs ayant pour portée l'application.
Paramètres de requête	<i>param</i>	Une Map qui associe les noms et valeurs des paramètres de la requête.
	<i>paramValues</i>	Une Map qui associe les noms et multiples valeurs ** des paramètres de la requête sous forme de tableaux de String.
En-têtes de requête	<i>header</i>	Une Map qui associe les noms et valeurs des paramètres des en-têtes HTTP.
	<i>headerValues</i>	Une Map qui associe les noms et multiples valeurs ** des paramètres des en-têtes HTTP sous forme de tableaux de String.
Cookies	<i>cookie</i>	Une Map qui associe les noms et instances des cookies.
Paramètres d'initialisation	<i>initParam</i>	Une Map qui associe les données contenues dans les champs <param-name> et <param-value> de la section <init-param> du fichier web.xml.

Les objets implicites

Les objets de la technologie EL

Exemple:

Créer le fichier *exempleel.jsp*

```
<body>
  <p>
    Pays: ${param.pays}
  <br>
    Langue: ${param.langue}
  </p>
</body>
```

Puis, taper l'url suivant:

<http://localhost:8080/WebApp/exempleel.jsp?pays=Maroc&langue=Arabe>

Les objets implicites

Les objets de la technologie EL

Exemple:

Modifier le fichier *exempleel.jsp*

```
<body>
  <p>
    Pays: ${param.pays}
  <br>
    Langue: ${paramValues.langue[0]}
  <br>
    Langue: ${paramValues.langue[1]}
  </p>
</body>
```

Puis, taper l'url suivant:

<http://localhost:8080/WebApp/exempleel.jsp?pays=Maroc&langue=Arabe&langue=Anglais>

■ El et objets implicites

➔ Exemple 1 :

1. Créer une page jsp (formulaire.jsp) contenant un formulaire avec les champs suivants:
 - Nom (zone de texte)
 - Prénom (zone de texte)
 - Sexe (bouton radio M ou F)
 - Loisirs (cases à cocher)
 - Lecture
 - Sport
 - Voyage
 - Bouton de validation
2. Créer une Servlet (Servlet.java) qui permet de récupérer le contenu du formulaire saisi après sa validation par l'utilisateur et de Rediriger vers une autre page jsp (affichage.jsp), qui affiche la ou les valeurs saisies pour champ de formulaire.

EL et objets implicites

El et objets implicites

➔ Exemple 1 :

formulaire.html

```
<body>
<form action="/exemple2/Servlet"
method="POST">
Nom: <input type="text" name="nom">
<br/>
Prénom: <input type="text"
name="prenom" />
<br/>
Sexe:
Homme:<input type = "radio" name =
"sexe" value = "M">
Femme:<input type = "radio" name =
"sexe" value = "F">
<br/>
Loisirs:
Lecture : <input type = "checkbox"
Name = "Loisirs" Value = "Lecture">
Sport : <input type = "checkbox"
Name = "Loisirs" Value = "Sport">
Voyage : <input type = "checkbox"
Name = "Loisirs" Value = "Voyage">
<br/>
<input type="submit"
value="Envoyer" />
</form>
</body>
```

Servlet.java

```
protected void
doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException,
IOException {
request.getRequestDispatcher("/WEB-
INF/formulaire.jsp").forward(request, response);
}
protected void
doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException,
IOException {
request.getRequestDispatcher("/WEB-
INF/affichage.jsp").forward(request, response);
}
}
```

affichage.jsp

```
<%@ page language="java"
contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>EL</title>
</head>
<body>
    Bonjour <br>
    Nom: ${param.nom} <br>
    Prenom: ${param.prenom}
    Sexe: ${param.sexe} <br>
    Loisirs: ${param.Loisirs}<br>
</body>
</html>
```

● JSTL : Java server page Standard Tag Library

- JSP Standard Tag Library (JSTL) est un ensemble de tags personnalisés développé qui propose des fonctionnalités souvent rencontrées dans les JSP.
- Elle se constitue de cinq sous-bibliothèques:
 - **Core** : Gère les variables, les conditions, les boucles , ...
 - **Format** : Formate les données et faire l'internationalisation de son site
 - **XML** : Manipule des données XML
 - **SQL** : Permet d'écrire des requêtes SQL
 - **Function** : Traite des chaînes de caractères

Java server page Standard Tag Library (JSTL)

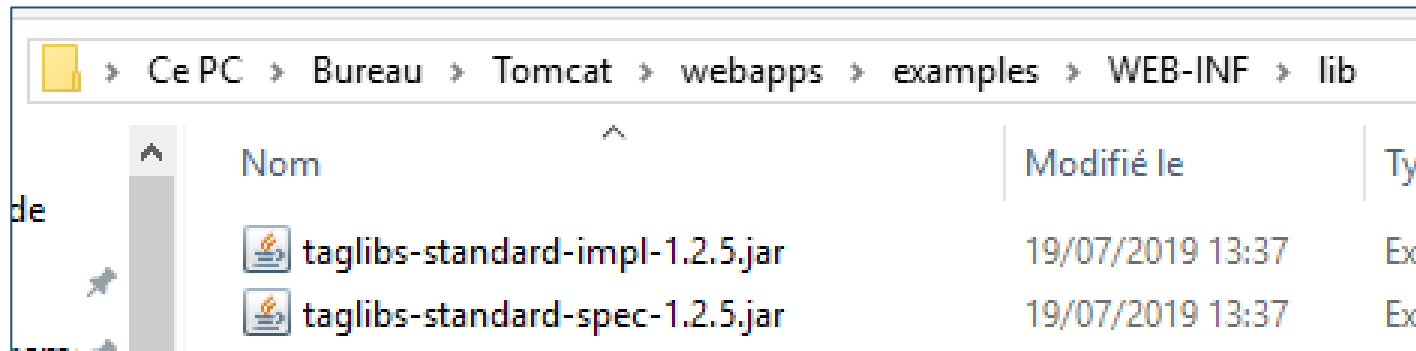
● Téléchargement la bibliothèque JSTL

D'abord, vous avez besoin de télécharger les bibliothèque de standard de JSTL.
Si vous avez téléchargé Tomcat server, ces bibliothèque se trouvent dans le dossier:

<Tomcat>/webapps/examples/WEB-INF/lib

taglibs-standard-impl-*.jar

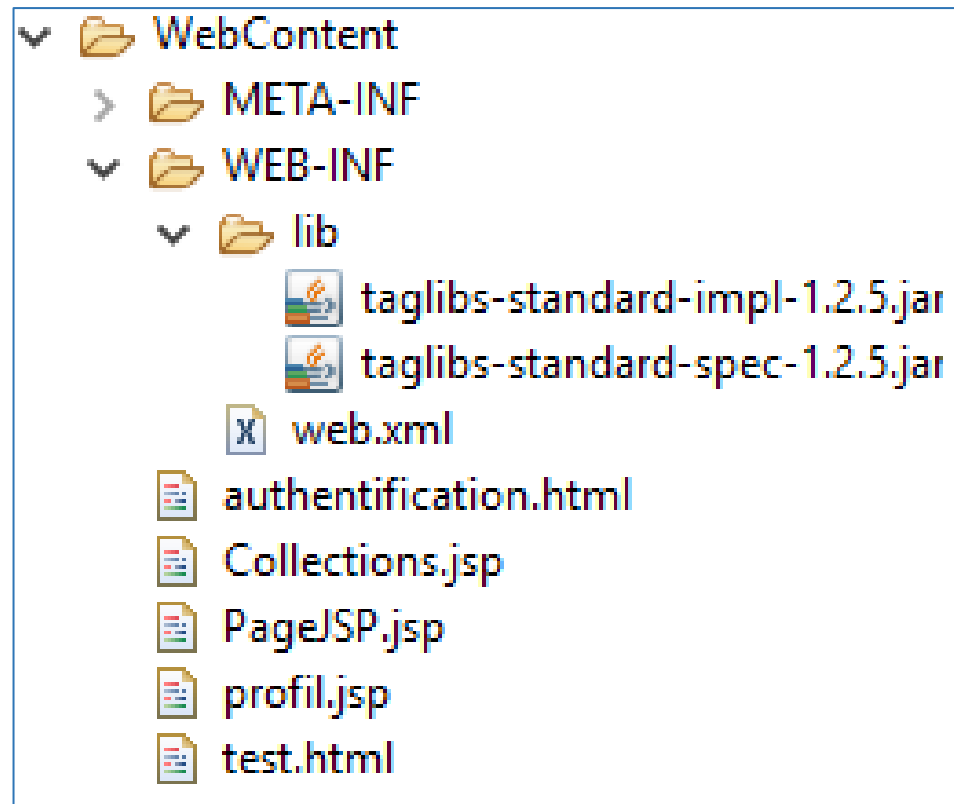
taglibs-standard-spec-*.jar



Java server page Standard Tag Library (JSTL)

■ Déclaration de la bibliothèque JSTL

Copiez les bibliothèques de **JSTL** et les mettez dans le dossier **WEB-INF/lib**



Java server page Standard Tag Library (JSTL)

● Les tags basiques de JSTL

Les fonctions	Description / Déclaration
Core Tags	Itérations, Conditions, Exceptions, URL, les réponses de transfert ou de redirection, etc. Pour utiliser les balises principales JSTL, il faut ajouter la directive suivante
	<code><%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%></code>
Balises de formatage et de localisation	Ces balises permettent de formater la prise en charge de Numbers, Dates ... Vous pouvez inclure ces balises dans JSP avec la syntaxe suivante:
	<code><%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%></code>
Tags SQL :	Les balises SQL JSTL permettent d'interagir avec des bases de données relationnelles pour exécuter des requêtes.
	<code><%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%></code>
XML Tags	Les balises XML permettent de travailler avec des documents XML tels que l'analyse XML, la transformation de données XML et l'évaluation d'expressions XPath.
	<code><%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%></code>
Balises de fonctions JSTL	Pour réaliser les opérations courantes. La plupart d'entre elles concernent la manipulation de chaînes.
	<code><%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%></code>

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

Tag	Description
<c:out>	Ecrire quelque chose sur une page jsp
<c:import>	comme <jsp:include> ou la directive <i>include</i>
<c:redirect>	Redirige la requête vers une autre ressource
<c:set>	Initialise une variable dans une portée donnée
<c:remove>	Supprime une variable dans une portée donnée
<c:catch>	Pour les exceptions
<c:if>	Condition Si
<c:choose>	Conditions à choix multiples, marqués par <c:when> et <c:otherwise>
<c:forEach>	Pour faire une boucle sur une collection
<c:forTokens>	Pour faire une boucle sur des chaines séparés par un délimiteur.
<c:param>	Utilisée avec <c:import> pour passer un paramètre
<c:url>	Pour créer un URL avec des paramètres de type query string

JSTL Core Tags

➡ `<c:out />`

- La balise utilisée pour l'affichage est `<c:out value="" />`.
- Le seul attribut obligatoire est *value*. Cet attribut peut contenir:
 - une chaîne de caractères simple,
 - ou une expression EL.

`<c:out value="nom" />` `<%-- Affiche nom--%>`

`<c:out value="${ param.nom}" />` `<%-- Affiche le paramètre nom de la requête--%>`

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➡ <c:out /> Exemple

```
<%@ 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 charset="UTF-8">
<title>JSTL</title>
</head>
<body>
<c:out value="Samir" />
<c:out value="${param.nom}" />
</body>
</html>
```

JSTL Core Tags

➔ `<c:set />`

- Cette balise permet de gérer les variables et les objets.
- Pour initialiser les variables
 - `<c:set var="prenom" value="Khalid" scope="request" />`
 - `<c:set var="prenom" scope="request">Khalid</c:set>`
- Pour créer un objet du type du bean
 - `<c:set scope="session" var="MonBean" value="${Bean}" />`
- Pour modifier les propriétés du bean créé:
 - `<c:set target="${etudiant}" property="prenom" value="Khalid"/>`

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➔ <c:set /> Exemple1

```
<%@ 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 charset="UTF-8">
<title>JSTL</title>
</head>
<body>
<c:set var="prenom" value="Samir" scope="request" />
<c:out value="${prenom}" />
<c:set var="nom" scope="request">el kaddouhi</c:set>
<c:out value="${nom}" />
</body>
</html>
```

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➔ <c:set /> Exemple2

```
<%@ 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 charset="UTF-8">
<title>JSTL</title>
</head>
<body>
<jsp:useBean id="etud" class="master.sim.Etudiant"/>
<c:set scope="session" var="etudiant" value="${etud}" />
<c:set target="${etudiant}" property="nom" value="Tahiri" />
<c:set target="${etudiant}" property="prenom" value="Mohamed" />
<c:out value="${etudiant.prenom}"/>
<c:out value="${etudiant.nom}"/>
</body>
</html>
```

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➔ `<c:remove />`

- Le tag **remove** permet de supprimer une variable d'une portée particulière.
`<c:remove var=«NomVariable" scope="request" />`

```
<%@ 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 charset="UTF-8">
<title>JSTL</title>
</head>
<body>
<c:set var="prenom" value="Samir" scope="request" />
<c:remove var="prenom" scope="request" />
<c:out value="${prenom}" />
</body>
</html>
```

● JSTL Core Tags

➔ `<c:if /><c:chose />`

- **Condition simple : `<c:if/>`**

`<c:if test="$ { expression}">`

Actions à faire si le test est vrais

`</c:if>`

- **Conditions multiples : `<c:chose/>`**

`<c:choose>`

`<c:when test="$ {expression1}">Action1</c:when>`

`<c:when test="$ {expression2}">Action2</c:when>`

`<c:otherwise>Autre</c:otherwise>`

`</c:choose>`

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➔ <c:if /> Exemple

```
<%@ 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 charset="UTF-8">
<title>JSTL</title>
</head>
<body>
<c:set var="a" value="5" scope="request" />
<c:if test="${a<8}">
<c:out value="La valeur ${a} est inférieure à 8"/>
</c:if>
</body>
</html>
```


Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➡ <c:choose.... /> Exemple

```
<body>
<c:set var="a" value="5" scope="request" />
<c:choose>
    <c:when test="{a}<8}">
        <c:out value="La valeur {a} est inférieure à 8"/>
    </c:when>
    <c:when test="{a}>8}">
        <c:out value="La valeur {a} est supérieure à 8"/>
    </c:when>
    <c:otherwise>
        <c:out value="La valeur {a} est égale à 8"/>
    </c:otherwise>
</c:choose>
</body>
```

JSTL Core Tags

➔ `<c:forEach.... />`

- Permet d'effectuer des itérations sur plusieurs types de collections de données. Il possède plusieurs attributs :
 - Var : nom de la variable qui contient l'élément en cours de traitement.
 - begin : numéro du premier élément à traiter (le premier possède le numéro 0)
 - end : numéro du dernier élément à traiter
 - Step : pas des éléments à traiter (par défaut 1)
 - Items : collection à traiter.
 - varStatus : nom d'une variable qui va contenir des informations sur l'itération en cours de traitement

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➔ <c:forEach.... /> Examples

■ Exemple1:

```
<body>
<c:forEach begin="1" end="4" var="i">
  <c:out value="${i}"/>
</c:forEach>
</body>
```

■ Exemple 2:

```
<body>
<c:forEach begin="1" end="12" var="i" step="3">
  <c:out value="${i}"/><br>
</c:forEach>
</body>
```

■ Exemple 3

```
<body>
<c:forEach var="i" begin="1" end="10">
  <c:choose>
    <c:when test="${i%2 == 0}"> <c:out value="${i} est paire"/><br/> </c:when>
    <c:otherwise> <c:out value="${i} est impaire"/><br/> </c:otherwise>
  </c:choose>
</c:forEach>
</body>
```

JSTL Core Tags

➡ <c:forEach.... /> Exemples

▪ Exemple4:

```
<body>
<table>
  <tr>
    <th>Valeur</th>
    <th>Cube</th>
  </tr>
  <c:forEach var="i" begin="0" end="7" step="1">
    <tr>
      <td><c:out value="${i}"/></td>
      <td><c:out value="${i * i * i}"/></td>
    </tr>
  </c:forEach>
</table>
</body>
```

JSTL Core Tags

➔ <c:forEach.... /> Examples

■ Example5:

```
<%  
String[] noms= {"Mohamed","Rachid","Khalid"};  
request.setAttribute("noms", noms );  
%>  
<c:forEach begin="0" end="2" var="i">  
<c:out value="${noms[i]}" /><br>  
</c:forEach>
```

■ Example6:

```
<%  
String[] noms= {"Mohamed","Rachid","« Khalid"};  
request.setAttribute("noms", noms );  
%>  
<c:forEach items="${noms}" var="nom">  
<c:out value="${nom}" /><br>  
</c:forEach>
```

JSTL Core Tags

➔ <c:forEach.... /> Examples

■ Example7:

```
<%  
String[] noms= {"Mohamed","Rachid","Khalid"};  
request.setAttribute("noms", noms );  
%>  
<c:forEach items="${noms}" var="nom" begin="0" end="1">  
<c:out value="${nom}"/><br>  
</c:forEach>
```

■ Example8:

```
<%  
String[] noms= {"Mohamed","Rachid","Khalid"};  
request.setAttribute("noms", noms );  
%>  
<c:forEach items="${noms}" var="nom" varStatus="status">  
N° <c:out value="${status.count}"/>  
<c:out value="${nom}"/><br></c:forEach>
```

JSTL Core Tags

➔ `<c:forTokens.... />`

- permet de découper une chaîne selon un ou plusieurs séparateurs donnés. Il possède plusieurs attributs :
 - Var : nom de la variable qui contient l'élément en cours de traitement.
 - Items : la chaîne de caractères à traiter (obligatoire).
 - Delims : précise le ou les séparateurs
 - varStatus : nom d'une variable qui va contenir des informations sur l'itération en cours de traitement
 - begin : numéro du premier élément à traiter (le premier possède le numéro 0)
 - end : numéro du dernier élément à traiter
 - Step : pas des éléments à traiter (par défaut 1)

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➔ <c:forTokens.... /> Examples

- **Exemple1:**

```
<c:forTokens var="nom" items="Mohamed;Rachid;Khalid" delims=";">  
<c:out value="{nom}" /><br>  
</c:forTokens>
```

- **Exemple2:**

```
<c:forTokens var="nom" items="Mohamed/Rachid/Khalid" delims="/">  
<c:out value="{nom}" /><br>  
</c:forTokens>
```

- **Exemple3:**

```
<c:forTokens var="nom" items="Mohamed;Rachid;Khalid" delims=";" begin="0"  
end="0">  
<c:out value="{nom}" /><br>  
</c:forTokens>
```


JSTL Core Tags

➡ `<c:url.... />`

- Permet de formater une URL. Il possède plusieurs attributs
- L'attribut *value* contient l'URL, et l'attribut *var* permet de stocker le résultat dans une variable.

- Pour générer une url simple:

```
<a href="<c:url value="test.jsp" />">lien</a>
```

- Pour générer une url et la stocker dans une variable

```
<c:url value="test.jsp" var="lien"/>
```

JSTL Core Tags

➡ `<c:import.... />`

- Permet d'accéder à une ressource grâce à son URL pour l'inclure ou l'utiliser dans les traitements de la JSP. La ressource accédée peut être dans une autre application.
- Son grand intérêt par rapport au tag `<jsp :include>` est de ne pas être limité au contexte de l'application web.

```
<c:import url="menu.jsp"/>
```

JSTL Core Tags

➔ `<c:redirect.... />`

- Permet de faire une redirection vers une nouvelle URL

- Exemple 1:

```
<c:redirect url="http://www.google.com"/>
```

- Exemple 2:

```
<c:redirect url="http://www.google.com">  
  <c:param name="langue" value="fr"/>  
</c:redirect>
```

JSTL Core Tags

→ Exemple 1 :

1. Créer une page jsp (formulaire.jsp) contenant un formulaire avec les champs suivants:
 - Nom (zone de texte)
 - Prénom (zone de texte)
 - Sexe (bouton radio M ou F)
 - Loisirs (cases à cocher)
 - Lecture
 - Sport
 - Voyage
 - Bouton de validation
2. Créer une Servlet (Servlet.java) qui permet de récupérer le contenu du formulaire saisi après sa validation par l'utilisateur et de Rediriger vers une autre page jsp (affichage.jsp), qui affiche la ou les valeurs saisies pour champ de formulaire.

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➔ Exemple 1 :

formulaire.jsp

```
<body>
<form action="/exemple2/Servlet"
method="POST">
Nom: <input type="text" name="nom">
<br/>
Prénom: <input type="text"
name="prenom" />
<br/>
Sexe:
Homme:<input type = "radio" name =
"sexe" value = "M">
Femme:<input type = "radio" name =
"sexe" value = "F">
<br/>
Loisirs:
Lecture : <input type = "checkbox"
Name = "Loisirs" Value = "Lecture">
Sport : <input type = "checkbox"
Name = "Loisirs" Value = "Sport">
Voyage : <input type = "checkbox"
Name = "Loisirs" Value = "Voyage">
<br/>
<input type="submit"
value="Envoyer" />
</form>
</body>
```

Servlet.java

```
protected void
doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException,
IOException {
request.getRequestDispatcher("/WEB-
INF/formulaire.jsp").forward(request, response);
}
protected void
doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException,
IOException {
request.getRequestDispatcher("/WEB-
INF/affichage.jsp").forward(request, response);
}
}
```

affichage.jsp

```
<%@ page language="java"
contentType="text/html;
charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>EL</title>
</head>
<body>
    Bonjour <br>
    Nom: ${param.nom} <br>
    Prenom: ${param.prenom}
    Sexe: ${param.sexe} <br>
    Loisirs: ${param.Loisirs}<br>
</body>
</html>
```

Java server page Standard Tag Library (JSTL)

JSTL Core Tags

➔ Exemple 1 : Solution à l'aide du JSTL

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%> Déclaration
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JSTL</title>
</head>
<body>
Prénom: ${param.prenom}<br>
Nom: ${param.nom}<br>
Sexe: ${param.sexe}<br>
Loisirs:<br>
<c:forEach items="${paramValues.Loisirs}" var="loisir"> Boucle JSTL
    ${loisir}<br>
</c:forEach>
</body>
</html>
```

Java server page Standard Tag Library (JSTL)

■ JSTL formatage et internationalisation

Tags	Descriptions
fmt:formatNumber fmt:parseNumber	Formatage de nombres
fmt:parseDate fmt:timeZone fmt:setTimeZone fmt:formatDate	Formatage de dates
fmt:setLocale	Définition de la langue
fmt:bundle fmt:message fmt:setBundle	Formatage de messages

Java server page Standard Tag Library (JSTL)

JSTL formatage et internationalisation

➔ Exemple 1

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

<html>
  <head>
<title>Exemple 1 fmt</title>
  </head>
  <body>
    <fmt:formatNumber pattern="00.00" value="${10/3}"/><br>
    <fmt:formatNumber pattern="00.00" value="4"/><br>

    <c:set var="nombre" value="6,1250" />
    <fmt:parseNumber var="n" integerOnly="true" type="number" value="${nombre}" />
    Nombre est:<c:out value="${n}" /> <br>
  </body>
</html>
```


Java server page Standard Tag Library (JSTL)

JSTL formatage et internationalisation

➔ Exemple 1

```
<body>
<fmt:formatNumber pattern="00.00" value="${10/3}" /><br>

<fmt:setLocale value="fr_CA" />

<c:set var="Amount" value="9850.14115" />

<fmt:formatNumber pattern="00.00 MAD" value="214.789" /><br>

<fmt:formatNumber value="${Amount}" type="currency" /><br>

<fmt:formatNumber type="number" groupingUsed="true" value="${Amount}" /><br>

<fmt:formatNumber type="number" maxIntegerDigits="3" value="${Amount}" /><br>

<fmt:formatNumber type="number" maxFractionDigits="6" value="${Amount}" /><br>

<fmt:formatNumber type="percent" maxIntegerDigits="8" maxFractionDigits="2" value="${Amount}" /><br>

<fmt:formatNumber type="number" pattern="###.###$" value="${Amount}" />
</body>
```

JSTL formatage et internationalisation

→ Exemple 2

```
<c:set var="Date" value="<%=new java.util.Date()%>" />

<fmt:formatDate type="time" value="${Date}" /><br>

<fmt:formatDate type="date" value="${Date}" /><br>

<fmt:formatDate type="both" value="${Date}" /><br>

<fmt:formatDate type="both" dateStyle="short" timeStyle="short" value="${Date}" /><br>

<fmt:formatDate type="both" dateStyle="medium" timeStyle="medium" value="${Date}" /><br>

<fmt:formatDate type="both" dateStyle="long" timeStyle="long" value="${Date}" /><br>
```

Java server page Standard Tag Library (JSTL)

JSTL formatage et internationalisation

➔ Exemple 3

```
<c:set var="date" value="<%=new java.util.Date()%>" />

<p><b>Date and Time in Morocco (GMT+1): </b>

<fmt:formatDate value="${date}" type="both" timeStyle="long" dateStyle="long" /></p>

<fmt:setTimeZone value="GMT+10" />

<p><b>Date and Time in Melbourne, Australia (GMT+10): </b>

<fmt:formatDate value="${date}" type="both" timeStyle="long" dateStyle="long" /></p>
```

Java server page Standard Tag Library (JSTL)

JSTL Function Tags

JSTL Functions	Description
fn:contains()	Teste si une chaîne d'entrée contenant la sous-chaîne spécifiée dans un programme. (fonction sensible à la casse)
fn:containsIgnoreCase()	Teste si une chaîne d'entrée contenant la sous-chaîne spécifiée dans un programme. (fonction insensible à la casse)
fn:endsWith()	Permet de tester si une chaîne d'entrée se termine par le suffixe spécifié.
fn:indexOf()	Il retourne l'index de la première occurrence d'une sous-chaîne spécifiée dans une chaîne.
fn:trim()	Il supprime les espaces vides des deux extrémités d'une chaîne.
fn:startsWith()	Il est utilisé pour vérifier si la chaîne donnée commence par une valeur de chaîne particulière.
fn:split()	Il divise la chaîne en un tableau de sous-chaînes.
fn:toLowerCase()	Convertir une chaîne de caractères en minuscule.
fn:toUpperCase()	Convertir une chaîne de caractères en majuscule.
fn:substring()	Extrait la sous-chaîne de la chaîne de caractères entre les positions début et fin
fn:substringAfter()	Il retourne le sous-ensemble de chaîne après une sous-chaîne spécifique.
fn:substringBefore()	Il retourne le sous-ensemble de chaîne avant une sous-chaîne spécifique.
fn:length()	Retourne la longueur d'une chaîne
fn:replace()	Il remplace toute l'occurrence d'une chaîne par une autre séquence.

Java server page Standard Tag Library (JSTL)

JSTL Function Tags

→ Exemple 1

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head><title>Exemple 1 fn</title></head>
<body>
<c:set var="texte" value="« Bienvenue à la faculté polydisciplinaire de Taza" />

<c:if test="${fn:contains(texte, 'Taza')}">
  <p>Chaine Taza trouvée</p>
</c:if>

<c:if test="${fn:containsIgnoreCase(texte, 'Taza')}">
  <p>Chaine Taza trouvée</p>
</c:if>

<c:if test="${fn:endsWith(texte, 'Taza') && fn:startsWith(texte, 'Bienvenue')}">
  <p>Texte commence par 'Bienvenue' et se termine par 'Taza'</p>
</c:if>

</body>
</html>
```

Java server page Standard Tag Library (JSTL)

JSTL Function Tags

➔ Exemple 2

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head><title>Exemple 2 fn</title></head>
<body>
<c:set var="str" value="C++;Java;Python;Scala"/>
<c:set var="tableau" value="${fn:split(str, ';')}" />
<c:forEach items="${tableau}" var="element">
${element}<br>
</c:forEach>
<c:set var="str2" value="${fn:join(tableau, '-')}" />
${str2}
${fn:substring(str2, 4, 8)}
${fn:substringAfter(str2, "Java-")}
${fn:substringBefore(str2, "-Scala")}
${fn:substringBefore(fn:substringBefore(str2, "-Scala"), "-Scala")}
${fn:length(str2)}
${fn:replace(str2, "C++", "C#")}

</body>
</html>
```

Java server page Standard Tag Library (JSTL)

JSTL XML Tags

XML Tags	Descriptions
<code>x:out</code>	Similaire à <code><%= ... ></code> , mais pour les expressions XPath.
<code>x:parse</code>	Utilisé pour analyser les données XML et retrouver les valeurs de nœuds ou attributs.
<code>x:set</code>	Utilisée pour affecter à une variable, une valeur d'une expression XPath.
<code>x:choose</code>	Conditions à choix multiples, marqués par <code><c:when></code> et <code><c:otherwise></code>
<code>x:if</code>	Condition SI
<code>x:transform</code>	Pour faire une transformation XSL
<code>x:param</code>	Utilisée avec la balise de transformation pour définir le paramètre dans la feuille de style XSLT.

JSTL XML Tags

→ Exemple 1

Créer le fichier XML suivant:

```
<?xml version="1.0" encoding="UTF-8"?>
<comptes>
  <user login="Samir123">
    <prenom>Samir</prenom>
    <nom>El Kaddouhi</nom>
    <email>samir.kaddouhi@gmail.com</email>
    <password>1234</password>
    <role>admin</role>
  </user>
  <user login="Mohamed321">
    <prenom>Mohamed</prenom>
    <nom>Alami</nom>
    <email>Alami@hotmail.com</email>
    <password>1234</password>
    <role>ordinaire</role>
  </user>
</comptes>
```


Java server page Standard Tag Library (JSTL)

JSTL XML Tags

→ Exemple 1

Récupérer et analyser un document

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html><head> <title>Exemple 1 XML</title> </head> <body>
<c:import url="users.xml" varReader="monReader">
  <x:parse var="doc" doc="{monReader}" ></x:parse>
  <x:out select="$doc/comptes/user[2]/nom" ></x:out><br>
  <x:set var="log" select="$doc/comptes/user[nom='Alami']/@Login"/>
  <x:out select="$log"/>
</c:import>
<x:choose>
  <x:when select="$doc/comptes/user[@Login='Mohamed321']">
    Mohamed321 a déjà un compte, son rôle est :
    <x:out select="$doc/comptes/user[@Login='Mohamed321']/role" ></x:out>
  </x:when>
  <x:otherwise>
    Mohamed321 n'a pas de compte
  </x:otherwise>
</x:choose>
</body> </html>
```

Java server page Standard Tag Library (JSTL)

JSTL SQL Tags

SQL Tags	Descriptions
<code>sql:setDataSource</code>	Il est utilisé pour créer une source de données simple adaptée uniquement au prototypage.
<code>sql:query</code>	Il est utilisé pour exécuter la requête SQL définie dans son attribut SQL ou dans le corps.
<code>sql:update</code>	Il est utilisé pour exécuter la mise à jour SQL définie dans son attribut sql ou dans le corps de la balise.
<code>sql:param</code>	Il est utilisé pour définir le paramètre dans une instruction SQL sur la valeur spécifiée.
<code>sql:dateParam</code>	Il est utilisé pour définir le paramètre dans une instruction SQL sur une valeur <code>java.util.Date</code> spécifiée.
<code>sql:transaction</code>	Il est utilisé pour fournir l'action de base de données imbriquée avec une connexion commune.

JSTL SQL Tags

➔ Exemple

Avant de commencer, nous devons préparer une table *user* dans le serveur de bases de données. Dans notre cas, nous avons choisit MySQL.

Prenons la table « user » de la base de données « base »:

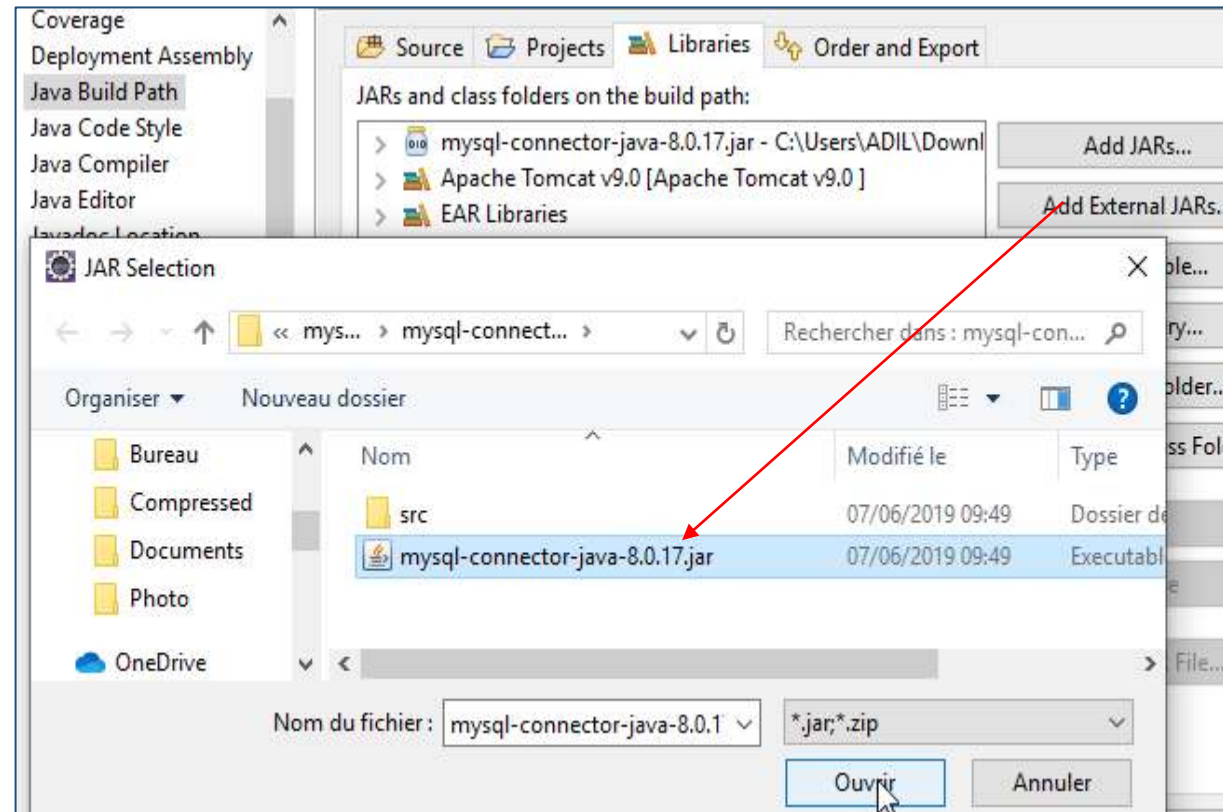
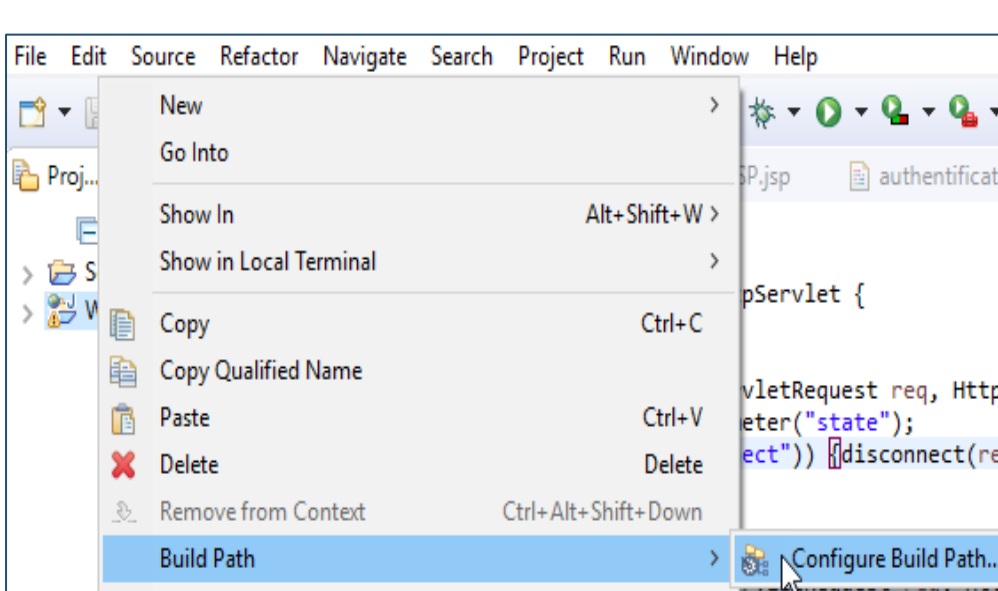
Login	pwd	nom	prenom	email	role
Samir123	123	El Kaddouhi	Samir	Samir.kaddouhi@gmail.com	Admin
Mohamed123	123	Alami	Mohamed	Alami@hotmail.com	user

Java server page Standard Tag Library (JSTL)

JSTL SQL Tags

→ Exemple

Importer le driver JDBC de MySQL



Java server page Standard Tag Library (JSTL)

JSTL SQL Tags

→ Exemple

Sélectionner et afficher les données

```
<sql:setDataSource var="base" driver="com.mysql.cj.jdbc.Driver"
url="jdbc:mysql://localhost/base" user="root" password=""/>
<sql:query dataSource="${base}" var="rs">
SELECT * from user;
</sql:query>
<table border="2" width="100%">
<tr><th>Login</th><th>Prénom</th><th>Nom</th><th>Rôle</th></tr>
<c:forEach var="table" items="${rs.rows}">
<tr><td><c:out value="${table.login}"/></td>
    <td><c:out value="${table.prenom}"/></td>
    <td><c:out value="${table.nom}"/></td>
    <td><c:out value="${table.role}"/></td>
</tr>
</c:forEach>
</table>
```

Java server page Standard Tag Library (JSTL)

JSTL SQL Tags

➔ Exemple

Ajouter, supprimer et modifier

```
<sql:setDataSource var="base" driver="com.mysql.cj.jdbc.Driver"
url="jdbc:mysql://localhost/base" user="root" password=""/>
<sql:query dataSource="${base}" var="rs">
SELECT * from user;
</sql:query>
<sql:update dataSource="${base}" var="ajout">
INSERT INTO user
VALUES ('Adil123','123','Charqui','Adil','Charqui@gmail.com','admin');
</sql:update>  ${'Elément bien inséré' }
<sql:update dataSource="${base}" var="modif">
update user set role='user' where login='Samir123'
</sql:update>  ${'Elément bien modifié' }
<sql:update dataSource="${base}" var="suppr">
delete from user where login='Mohamed123'
</sql:update>  ${'Elément bien supprimé' }
```



Chapitre 5

Cookies, Sessions





Cookies

- **Cookie :** Informations envoyées par le serveur, stockée sur le client et renvoyées par le client quand il revient visiter la même URL.
 - Durée de vie réglable
 - Permet d'avoir des données persistantes côté client
- **Utilisation :**
 - Identification des utilisateurs
 - Éviter la saisie d'informations à répétition (login, password, adresse,...)
 - Gérer des préférences utilisateur, profils



Créer un Cookie

- Pour la création d'un nouveau cookie, il faut l'ajouter à la réponse (HttpServletResponse) `addCookie()`

Exemple

```
Cookie cookie = new Cookie("login", "xxxx");  
cookie.setMaxAge(2*24*60*60); // Durée de vie=2Jours  
response.addCookie(cookie);
```

Récupérer un Cookie

- Pour récupérer les cookies provenant de la requête du client, on utilise la méthode `getCookies()` de l'objet `HttpServletRequest`
- `Cookie[] getCookies()` : retourne un tableau contenant l'ensemble des cookies présentes chez le client.
- `String getName()` : retourne le nom du cookie
- `String getValue()` : retourne la valeur du cookie
- `setValue(String new_value)` : donne une nouvelle valeur au cookie
- `setMaxAge(int expiry)` : spécifie l'âge maximum du cookie

Exemple

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for (int i = 0; i < cookies.length; i++) {
        String name = cookies[i].getName();
        String value = cookies[i].getValue();
    }
}
```

Supprimer un Cookie

- Pour supprimer un cookie présent sur la machine du client, il faut lui envoyer un nouveau cookie avec les paramètres suivants : nom identique, valeur vide, durée de vie égale à 1.

Exemple

```
Cookie cookie = new Cookie("cookie", " ");  
cookie.setMaxAge(-1); // Durée de vie=2Jours  
response.addCookie(cookie);
```



Cookies

- Le plus gros problème des cookies est que les navigateurs ne les acceptent pas toujours
- L'utilisateur peut configurer son navigateur pour qu'il refuse ou pas les cookies



Session

- Une session est un objet associé à un utilisateur en particulier. Elle existe pour la durée pendant laquelle un visiteur va utiliser l'application, cette durée se terminant lorsque l'utilisateur ferme son navigateur, reste inactif trop longtemps, ou encore lorsqu'il se déconnecte du site.
- La session représente un espace mémoire alloué pour chaque utilisateur, permettant de sauvegarder des informations tout le long de leur visite ;
- L'objet Java sur lequel se base une session est l'objet HttpSession ;
- Il existe un objet implicite `sessionScope` permettant d'accéder directement au contenu de la session depuis une expression EL dans une page JSP.



Session

→ HttpSession

- Méthodes de création liées à la requête (HttpServletRequest)
 - L'objet HttpServletRequest propose une méthode getSession(), qui permet de récupérer la session associée à la requête HTTP en cours si elle existe, ou d'en créer une si elle n'existe pas encore :
`HttpSession session = request.getSession();`
- L'objet session propose un ensemble de méthodes
 - Un couple de méthodes setAttribute() / getAttribute(), permettant la mise en place d'objets au sein de la session et leur récupération.
 - Une méthode getId(), retournant un identifiant unique permettant de déterminer à qui appartient telle session.
- Destruction (HttpSession)
 - invalidate() : expire la session

● Exemple Servlet

```
@WebServlet("/ServletS")
public class ServletS extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    request.getRequestDispatcher("/WEB-
INF/authentication.jsp").forward(request, response);
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    String login = request.getParameter("loginuser");
    String mdp = request.getParameter("mdpuser");
    utilisateur utilisateur = new utilisateur(login, mdp);
    HttpSession maSession = request.getSession();
    maSession.setAttribute("utilisateur", utilisateur);
    request.getRequestDispatcher("/WEB-
INF/information.jsp").forward(request, response);
    }
}
```

Java Bean

```
package master.sim;
public class utilisateur {
    private String login;
    private String pwd;
    public utilisateur() {
        super();
    }
    public utilisateur(String login, String
pwd) {
        super();
        this.login = login;
        this.pwd = pwd;
    }
    public String getLogin() {
        return login;
    }
    public void setLogin(String login) {
        this.login = login;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }
}
```

Exemple

authentication.JSP

```
<body>
<div align="center">
<h1>Veuillez vous identifier !!!</h1>
<form method="POST" action="/session/ServletS">
<table>
<tr>
<td>login :</td>
<td><input type="text" name="Loginuser"></td>
</tr>
<tr>
<td>Mot de passe :</td>
<td><input type="password" name="mdpuser" value=""></td>
</tr>
<tr>
<td>></td>
<td><input type="submit" value="Authentification"></td>
</tr>
</table>
</form>
</div>
</body>
```

Informations.JSP

```
<body>
<p>
<% utilisateur u = (utilisateur) session.getAttribute(
"utilisateur" ) ;%>
<h1>Information Utilisateur</h1>
<ul>
<li>Login : <%=u.getLogin() %>
<li>Mot de passe : <%=u.getPwd() %>
</ul>
</p>
</body>
```




Chapitre 6

Accès aux bases de données



Introduction

- JDBC : Java Data Base Connectivity
- Solution pour interagir avec une base de données
- Permet l'accès aux bases de données relationnelles dans un programme Java Indépendamment du type de la base utilisée (mySQL, Oracle, SQL Server ...)
- Permet de faire tout type de requêtes
 - Sélection de données dans des tables
 - Création de tables
 - insertion d'éléments dans les tables
 - Gestion des transactions
- Packages: java.sql et javax.sql

● Introduction

➔ Etapes d'accès à une base de données

- Chargement du driver
- Connexion à la base
- Préparation de l'exécution d'une requête
- Exécution de la requête
- Récupération des données
- fermeture de la connexion

● Etapes d'accès à une base de données

➔ Chargement du driver

➤ Chargement du driver

- Récupéré le driver JDBC correspondant à MySQL

`Class.forName("com.mysql.jdbc.Driver");`

- #### ➤ L'appel de la méthode `Class.forName` peut provoquer une exception de type `ClassNotFoundException` que nous devons attraper avec un bloc d'exception `try catch`..:

```
try{  
    Class.forName("com.mysql.jdbc.Driver");  
} catch(ClassNotFoundException e) {  
    System.out.println("Erreur lors du chargement du pilote" + e);  
}
```

● Etapes d'accès à une base de données

➔ Connexion à la base de données

- La méthode permettant d'obtenir une connexion à une base de données est la méthode **getConnection** de la classe **DriverManager**
- Il existe plusieurs formes de cette méthode, les plus utilisées sont : `getConnection(String url)` et `getConnection(String url, String user, String password)`

- url: identification de la base considérée sur le SGBD

jdbc:mysql://hôte:port/nom_de_la_bdd

- user: nom de l'utilisateur qui se connecte à la base
- password: mot de passe de l'utilisateur

- Exemple:

Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/etudiants","root","")

● Etapes d'accès à une base de données

➔ Connexion à la base de données

- L'appel de la méthode `Class.forName` peut provoquer une exception de type `ClassNotFoundException` que nous devons attraper avec un bloc d'exception `try catch`:

```
try{  
    connection = DriverManager.getConnection  
    ("jdbc:mysql://localhost:3306/etudiant","root","");  
} catch (SQLException e) {  
    System.out.println("Erreur de la connexion");  
}  
}
```

● Etapes d'accès à une base de données

➔ Préparation de l'exécution d'une requête

- Création d'un objet de la classe **Statement**
- l'objet créer permet d'envoyer des requêtes aux bases de données. Il est obtenu en utilisant la méthode **createStatement** de la classe **Connection**

```
Statement statement = connection.createStatement();
```

■ Etapes d'accès à une base de données

➔ Exécution des requêtes

➤ Deux cas sont envisageables. Soit la requête est une sélection de données (sélection), soit une autre action (création, modification ou suppression). Dans le premier cas, la liste des données est renvoyée et la requête exécutée (`executeQuery`), dans le second cas, le nombre d'enregistrements affectés est retourné et la modification de l'état des données (`executeUpdate`) est exécutée.

- **SELECT** : `ResultSet rs = statement.executeQuery(requete);`
- **AUTRE** : `int nblignes = statement.executeUpdate(requete);`

● Etapes d'accès à une base de données

➔ Exécution des requêtes

➤ Consultation (avec un select) des données de la base

- ✓ Réaliser à travers l'objet **ResultSet** obtenu lors de l'appel à `statement.executeQuery(requete)`.

```
ResultSet resultat = statement.executeQuery("SELECT nom, prenom FROM etudiants;");
```

➤ Mise à jour du contenu de la base:

- ✓ `executeUpdate` : cette méthode est réservée à l'exécution de requêtes ayant un effet sur la base de données (écriture ou suppression), typiquement les requêtes de type INSERT, UPDATE, DELETE.

```
int statut = statement.executeUpdate("INSERT INTO etudiants  
(numero, nom, prenom)  
( '4', 'nom4', 'prenom4' );");
```

● Etapes d'accès à une base de données

➔ Récupération des données

- Contient les résultats d'une requête SELECT
- Accès aux colonnes/données dans une ligne
 - **[type] get[Type](int col)**
 - Retourne le contenu de la colonne col dont l'élément est de type [type]
- Ex : String resultat.getString(int col)**
- boolean next()
 - Se place à la ligne suivante s'il y en a une
- boolean previous()
 - Se place à la ligne précédente s'il y en a une
- boolean absolute(int index)
 - Se place à la ligne numérotée index
- Fermeture du ResultSet
 - void close()

● Etapes d'accès à une base de données

➔ Récupération des données

```
while(resultat.next())
{
    System.out.println(resultats.getString("nom")+"-"+resultat.getString("prenom")
+"-"+resultat.getInt("numero"));
}
}
catch (SQLException e)
{
    System.out.println("Erreur lors de l'établissement de la connexion");
}
```

- Etapes d'accès à une base de données
 - ➔ fermeture de la connexion

➤ Pour déconnecté faire:

```
connection.close();
```