

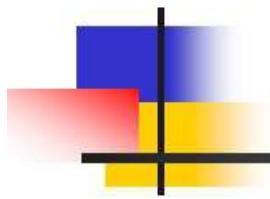
Technologie JEE



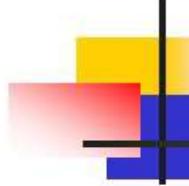
Licence Professionnelle : Informatique et
mathématiques appliquées

Dr. Allae Erraissi
Enseignant-chercheur à FPSB - UCD

2021-2022

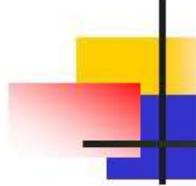


Chapitre 1 : Survol de la plateforme JEE



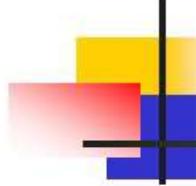
Notion de plateforme

- Une plateforme de développement informatique est une base générique qui fournit une implémentation à un ensemble de besoins récurrents :
 - accès aux bases de données,
 - communication réseaux,
 - etc..
- Avantages :
 - Se concentrer sur les fonctionnalités réelles de l'application en cours de développement.
 - ➔ Gain de temps et d'argent



Plateforme Java EE

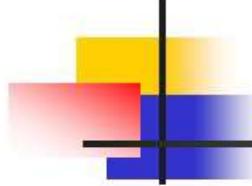
- Est l'Edition du Framework Java destinée au développement Web côté serveur
- Elle désigne l'ensemble constitué des services (API) offerts et de l'infrastructure d'exécution.
- Elle est proposée sous forme d'une norme comprenant :
 - Les spécifications du serveur d'application (environnement d'exécution).
 - Des services (au travers d'API) qui sont des extensions Java indépendantes permettant d'offrir en standard un certain nombre de fonctionnalités.



La Fondation Eclipse

- Est une organisation à but non lucratif fondée en 2004 par IBM autour de l'IDE Eclipse.
- Son objectif s'est élargi depuis à la création et gestions de projets open source
- Elle rallie plus de 200 membres dont : IBM, Oracle, Google, Microsoft, SAP,...
- En 2017, Oracle cède la gestion de sa plateforme JEE (depuis la version 8) et son IDE Netbeans à la fondation Eclipse.
- Ainsi **Java EE** devient **Jakarta EE**
- Et Netbeans devient Apache Netbeans



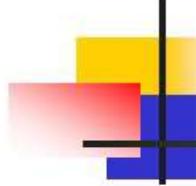


Jakarta EE



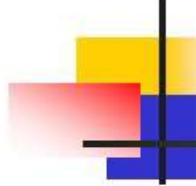
JAKARTA EE

- Est l'édition du Framework Java destinée au développement d'applications entreprise Java natives dans le cloud.
- Elle désigne l'ensemble constitué des services (API) offerts et de l'infrastructure d'exécution.
- Les spécifications Jakarta EE consistent en:
 - Spécification des APIs
 - Kit de compatibilité technologique (Technology Compatibility Kit ou TCK) - utilisé pour tester le code implémenté sur la base des API et du document de spécification
 - Implementation Compatible - implémentation qui passe avec succès le TCK



Les API de JEE

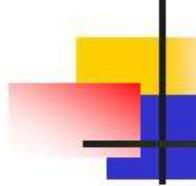
- Les API de JEE sont réparties en 5 grandes catégories :
 - Web Application Technologies
 - Enterprise Application Technologies
 - Web Services Technologies
 - Management and Security Technologies
 - Java EE-related Specs in Java SE



Les API de JEE

- Web Application Technologies

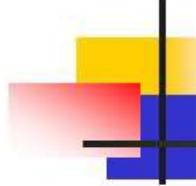
- Java Servlet 4.0
- JavaServer Pages 2.3
- Expression Language 3.0
- Standard Tag Library for JavaServer Pages (JSTL) 1.2
- JavaServer Faces 2.3
- Java API for WebSocket 1.1
- Java API for JSON Binding 1.0
- Java API for JSON Processing 1.1



Les API de JEE

■ Enterprise Application Technologies

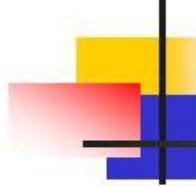
- Batch Applications for the Java Platform 1.0
- Concurrency Utilities for Java EE 1.0
- Contexts and Dependency Injection for Java 2.0
- Dependency Injection for Java 1.0
- Bean Validation 2.0
- Enterprise JavaBeans 3.2
- Interceptors 1.2
- Java EE Connector Architecture 1.7
- Java Persistence 2.2
- Common Annotations for the Java Platform 1.3
- Java Message Service API 2.0
- Java Transaction API (JTA) 1.2
- JavaMail 1.6



Les API de JEE

■ Web Services Technologies

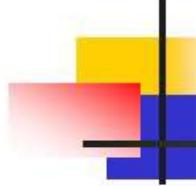
- Java API for RESTful Web Services (JAX-RS) 2.1
- Implementing Enterprise Web Services 1.3
- Web Services Metadata for the Java Platform 2.1
- Java API for XML-Based RPC (JAX-RPC) 1.1 (Optional)
- Java API for XML Registries (JAXR) 1.0 (Optional)



Les API de JEE

■ Management and Security Technologies

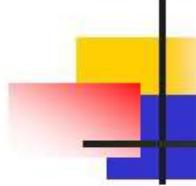
- Java EE Security API 1.0
- Java Authentication Service Provider Interface for Containers 1.1
- Java Authorization Contract for Containers 1.5
- Java EE Application Deployment 1.2 (Optional)
- J2EE Management 1.1
- Debugging Support for Other Languages 1.0



Les API de JEE

■ Java EE-related Specs in Java SE

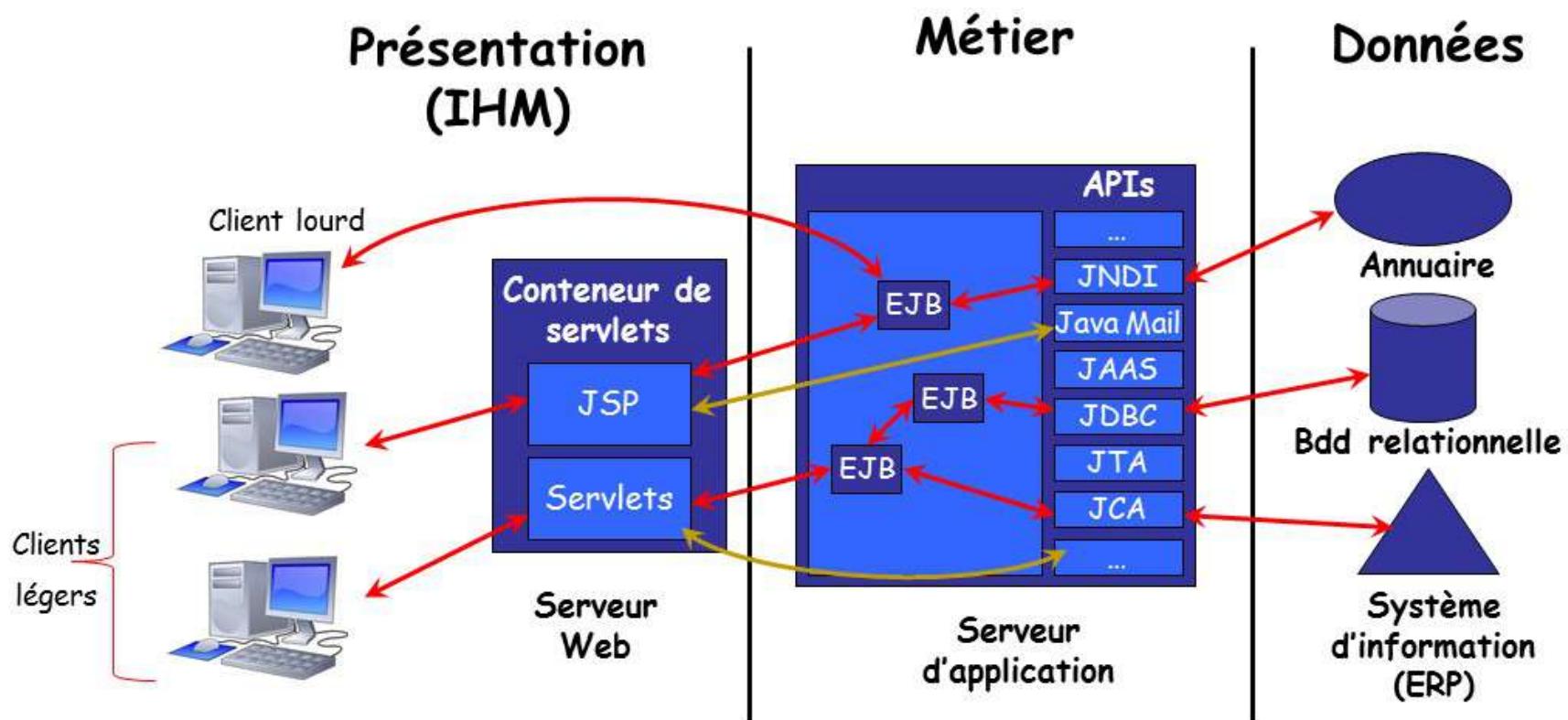
- Java Management Extensions (JMX) 2.0
- SOAP with Attachments API for Java (SAAJ) Specification 1.3
- Streaming API for XML (StAX) 1.0
- Java API for XML Processing (JAXP) 1.6
- Java Database Connectivity 4.0
- Java Architecture for XML Binding (JAXB) 2.2
- Java API for XML-Based Web Services (JAX-WS) 2.2
- JavaBeans Activation Framework (JAF) 1.1

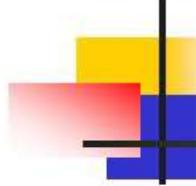


L'architecture JEE

- L'architecture JEE permet ainsi de séparer :
 - La couche présentation, correspondant à l'interface homme-machine (IHM),
 - La couche métier contenant l'essentiel des traitements de données en se basant dans la mesure du possible sur des API existantes,
 - La couche de données correspondant aux informations de l'entreprise stockées dans des :
 - ▶ Fichiers,
 - ▶ Bases de données relationnelles ou XML,
 - ▶ Annuaires d'entreprise
 - ▶ Systèmes d'information complexes (ERP par exemple).

L'architecture JEE





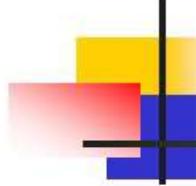
Types de clients

- **Client lourd** (*fat client* ou *heavy client*) :

- Application cliente graphique exécutée sur le système d'exploitation de l'utilisateur.
- Possède généralement des capacités de traitement évoluées
- Peut posséder une interface graphique sophistiquée.

- **Client léger** (*thin client*) :

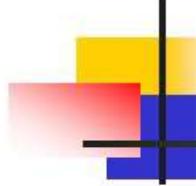
- Application accessible via un navigateur web,
- La totalité de la logique métier est traitée du côté du serveur.



Types de clients

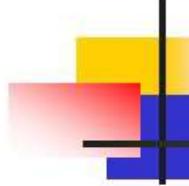
- **Client riche** (rich client) :

- Compromis entre le client léger et le client lourd.
- Propose une interface graphique avec des fonctionnalités avancées (glisser déposer, onglets, multi fenêtrage, menus déroulants) ,
- Décrit avec une grammaire basée sur la syntaxe XML



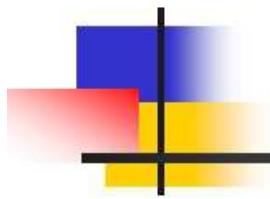
Serveurs d'application

- C'est un environnement d'exécution des applications côté serveur.
- Il prend en charge l'ensemble des fonctionnalités qui permettent à N clients d'utiliser une même application
- Lorsque les serveur d'application n'implémente que la partie Web (Servlets et JSP) de la spécification Java EE, il est appelé conteneur ou moteur de servlets

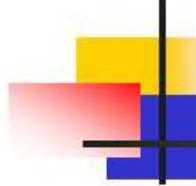


Serveurs d'application : Exemples

- Apache Tomcat
- Apache Geronimo
- GlassFish
- IBM Websphere Application Server
- JOnAS
- Novell exteNd Application Server
- Oracle WebLogic Server
- Resin Server
- WildFly (version communautaire de JBoss)
- JBoss EAP (version Red Hat JBoss)

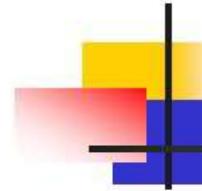


Chapitre 2 : Les Servlets



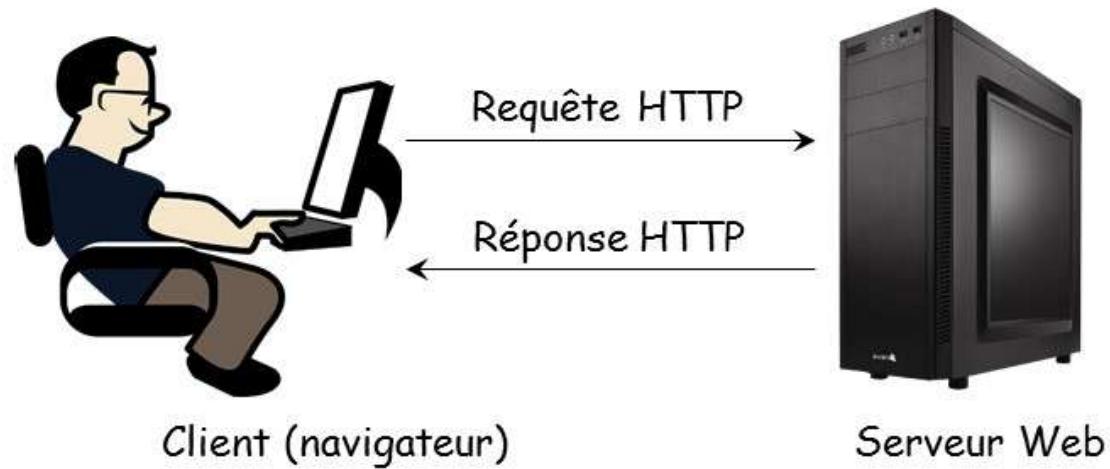
Introduction aux servlets

- Ce sont des classes Java fonctionnant du côté serveur.
- Elles permettent de construire des pages Web dynamiques.
- Une Servlet:
 - Reçoit des requêtes HTTP d'un client Web
 - Effectue traitement
 - Fournit une réponse HTTP dynamique au client Web
- La compréhension du fonctionnement du protocole HTTP est nécessaire



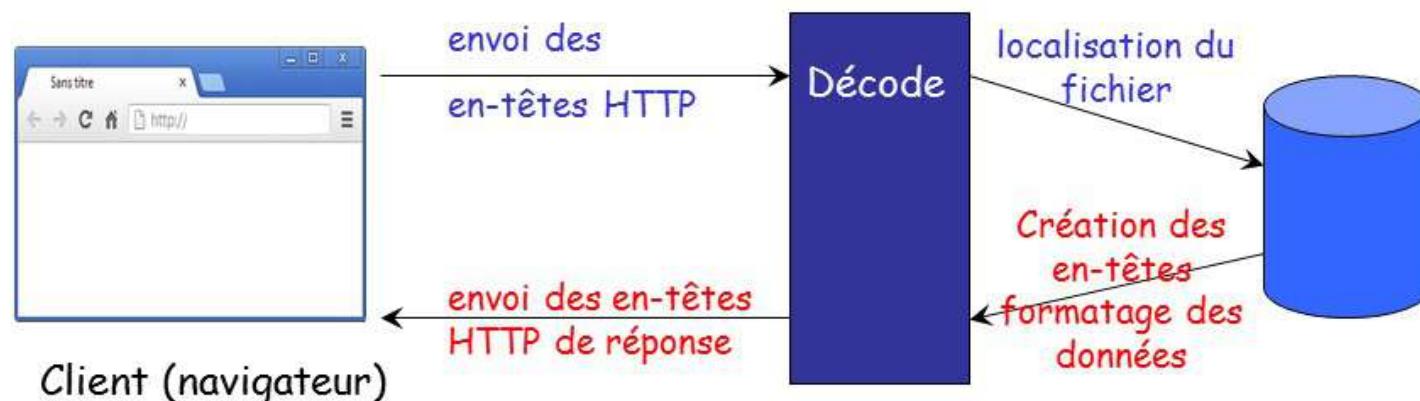
Servlets et protocole HTTP

- Les servlets s'appuient sur le protocole HTTP pour communiquer avec le client.
- HTTP agit comme un protocole de Transport

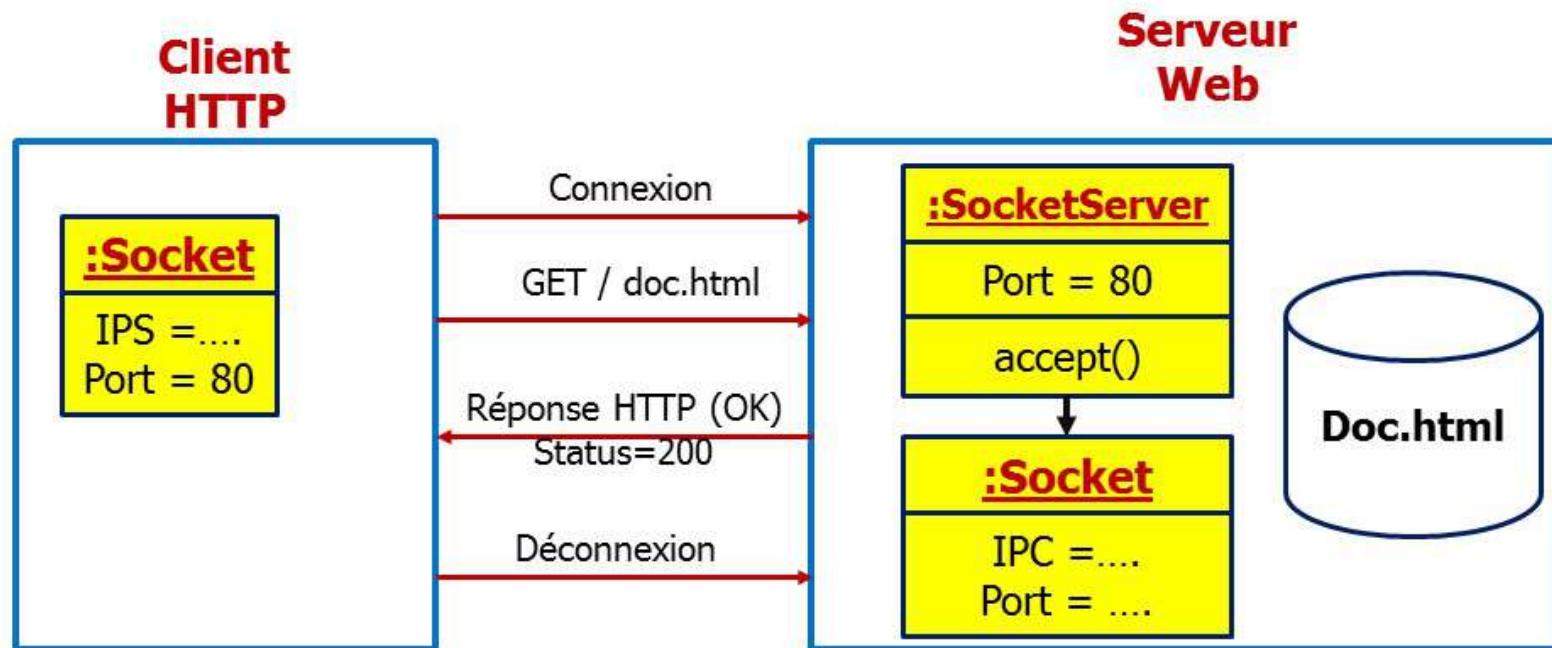


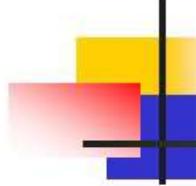
Le protocole HTTP

- HTTP = HyperText Tranfert Protocol
- Protocole qui permet au client de récupérer des documents sur serveur :
 - Documents Statiques (HTML, PDF, Image, etc..) ou
 - Documents Dynamiques (PHP, JSP, ASP...)
- HTTP permet également de soumissionner les formulaires



Le protocole HTTP : Fonctionnement

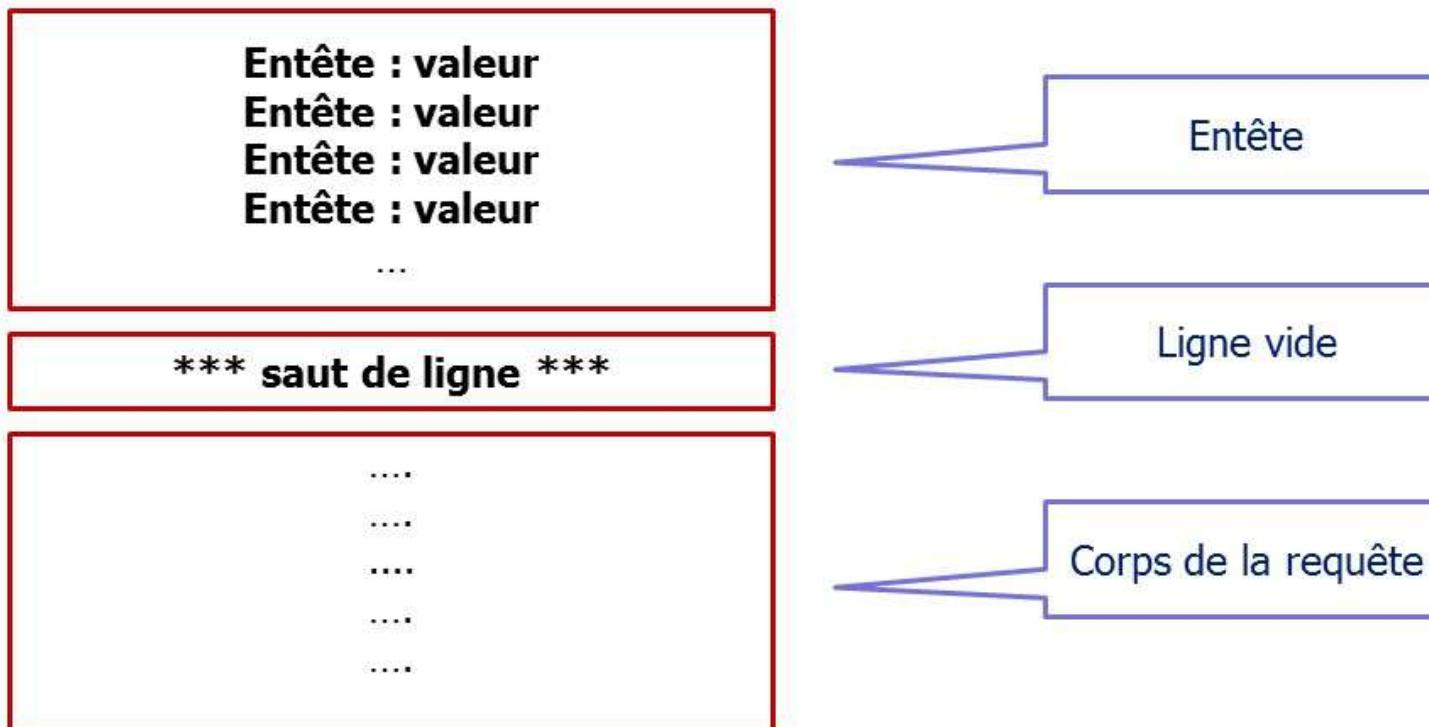




Le protocole HTTP : Méthodes

- 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 : Forme d'une requête HTTP



Le protocole HTTP : Requête de type POST

Post <http://www.site.ma>
HTTP/1.0
Accept : **text/html**
Accept-Language : **fr**
User-Agent : **Mozilla/4.0**

*** saut de ligne ***

param1=Valeur1& param2=Valeur2
& param3=Valeur3

Entête

Ligne vide

Corps de la requête

Le protocole HTTP : Requête de type GET

```
GET http://www.site.ma HTTP/1.0
Accept : text/html
Accept-Language : fr
User-Agent : Mozilla/4.0
```

*** saut de ligne ***

Entête

Ligne vide

Corps de la requête
(VIDE)

Le protocole HTTP : Exemple

Requête

GET Nom_Script?login=val1&pass=val2&.... **HTTP/1.0**

Accept : text/html

Accept-Language : fr

User-Agent : Mozilla/4.0

*** saut de ligne ***

Réponse

HTTP/1.0 200 OK

Date : Wed, 15 Sep 2017 15:02:01 GMT

Server : Apache/1.3.24

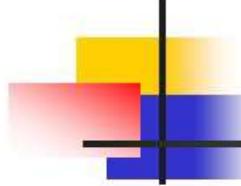
Last-Modified : Wed 02 Oct 2016 22:05:01 GMT

Content-Type : Text/html

Content-length : 4205

*** saut de ligne ***

```
<!DOCTYPE html>
<HTML>
<HEAD>.... </HEAD>
<BODY> .... </BODY>
</HTML>
```

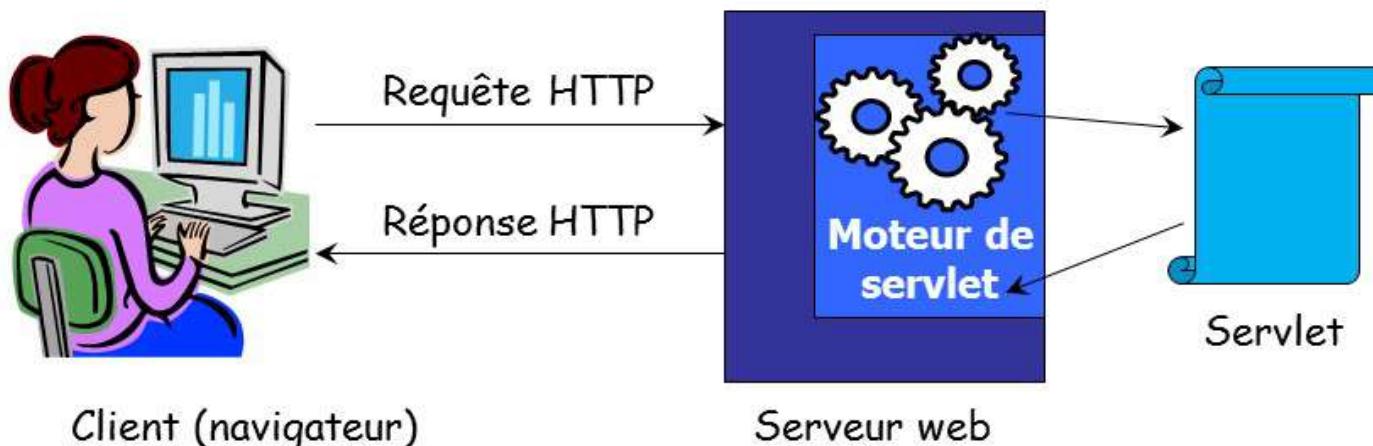


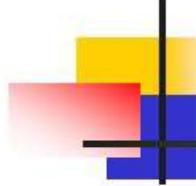
Le protocole HTTP et Types MIME

- MIME veut dire Multipurpose Internet Mail Extensions
- Standard utilisé entre autres pour typer les documents transférés par le protocole HTTP
- Un type MIME est constitué comme suit : **Content-type: TYPE/SOUS-TYPE**
 - Exemples :
 - ▶ **Content-type: image/gif** (Images gif)
 - ▶ **Content-type: image/jpeg** (Images Jpeg)
 - ▶ **Content-type: text/html** (Fichiers HTML)
 - ▶ **Content-type: text/plain** (Fichiers texte sans mise en forme)

Moteur de servlets

- Une Servlet ne tourne pas directement sur un serveur Web : Elle a besoin de tourner sur un Moteur de Servlet
- Est connu aussi par conteneur de servlets (en anglais Servlet Container)
- Il permet d'établir le lien entre la Servlet et le serveur Web



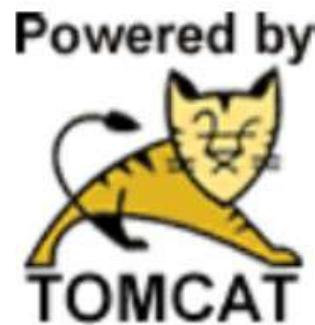


Moteur de servlets

- Un moteur de Servlets prend en charge et gère les servlets:
 - chargement de la servlet
 - gestion de son cycle de vie
 - passage des requêtes et des réponses
- Nombreux conteneurs de Servlet (Tomcat, JBoss, WebSphere Application Server, WebLogic,...)
- Dans le reste du cours et des TP, nous utiliserons le conteneur Tomcat pour déployer nos servlets.

Jakarta Tomcat

- Tomcat est une implémentation de référence de la spécification JEE
- Fournit donc une implémentation de l'API JEE (dossier lib)
- Disponible gratuitement sous forme d'une licence Open Source
- Écrit entièrement en Java et nécessite obligatoirement une machine virtuelle (JRE ou JDK).



Jakarta Tomcat

- Disponible pour téléchargement à l'adresse suivante :

<https://tomcat.apache.org/>

- Existe en

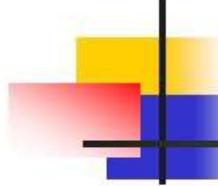
- Version zip (non installable)
 - version installable (.exe)

9.0.41

Please see the [README](#) file for packaging information. It explains wh

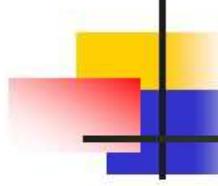
Binary Distributions

- Core:
 - [zip \(pgp, sha512\)](#)
 - [tar.gz \(pgp, sha512\)](#)
 - [32-bit Windows zip \(pgp, sha512\)](#)
 - [64-bit Windows zip \(pgp, sha512\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)



Jakarta Tomcat : version zip

- Téléchargement du fichier compressé
- Décompression dans un endroit spécifique
- Ajout des variables d'environnement suivantes :
 - **CATALINA_HOME** : dossier de décompression de Tomcat
 - **JAVA_HOME** (ou **JRE_HOME**) : dossier d'installation du JDK (ou dossier du JRE)



Jakarta Tomcat : version zip

- Les script de démarrage de Tomcat sont logés dans le dossier « bin » de Tomcat :
 - « Startup.bat » pour le démarrage
 - « Shutdown.bat » pour l'arrêt
- Remarque :
 - L'avantage de la version zip est que vous pouvez déplacer Tomcat facilement
 - Il suffit de modifier le chemin dans la variable d'environnement

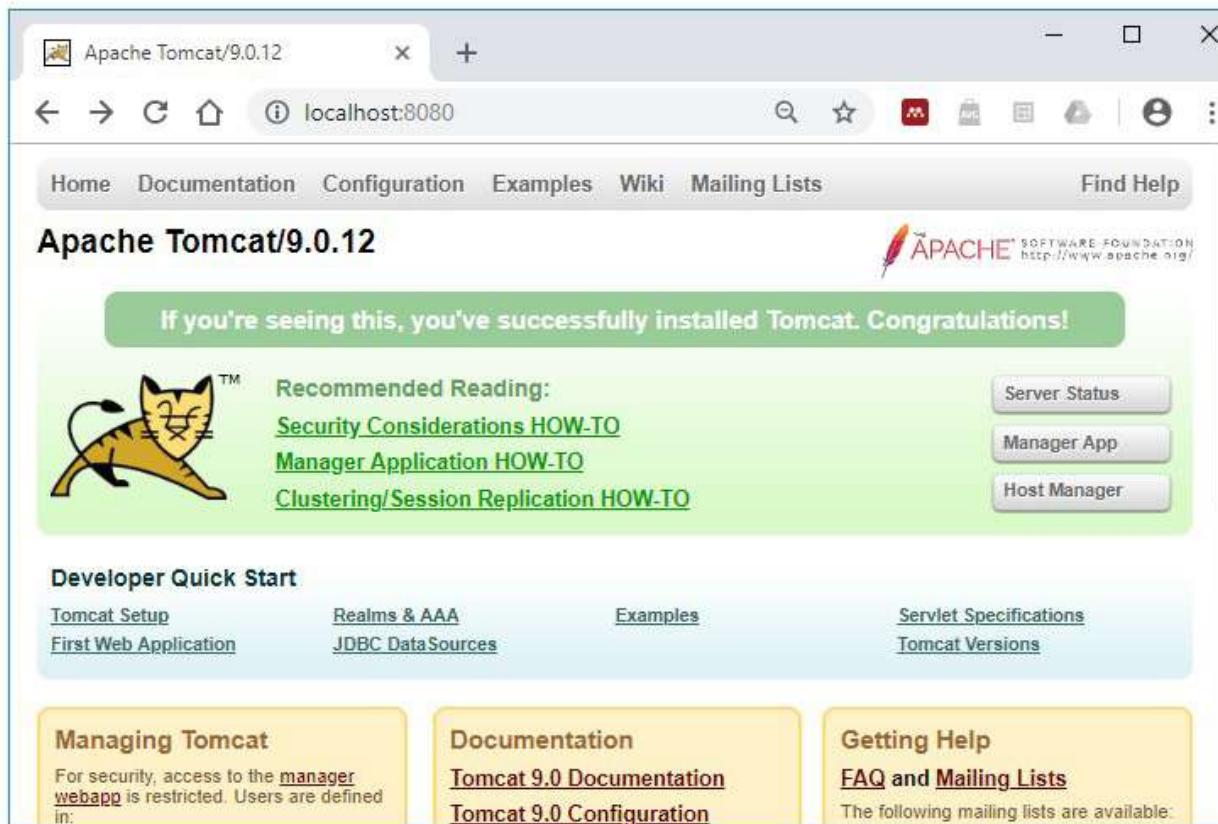
Jakarta Tomcat : version installable

- Après installation de Tomcat :
 - L'icône suivante montre quand il marche
 - L'icône suivante montre quand il est arrêté
 - « Start service » permet de le faire marcher s'il est en arrêt
 - « stop service » permet de l'arrêter s'il est en marche



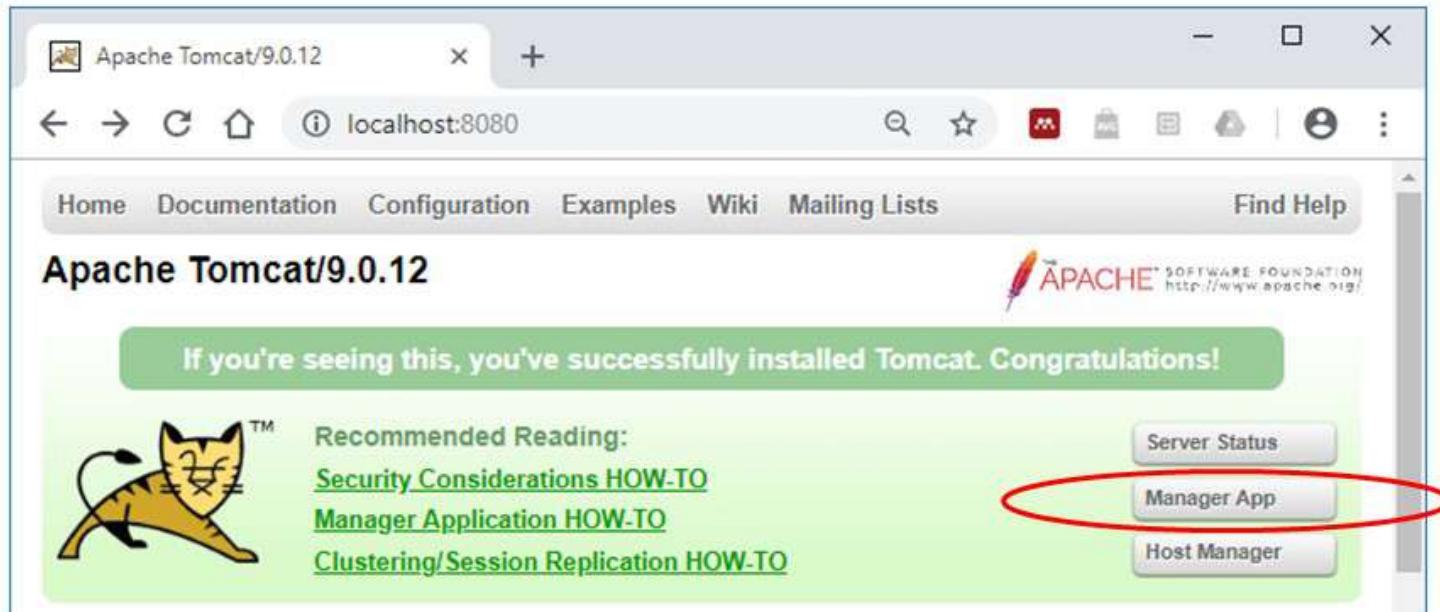
Jakarta Tomcat

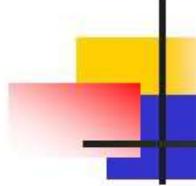
- Page d'accueil de Tomcat : ***http://localhost:8080***



Tomcat Manager

- Les applications hébergées sur Tomcat peuvent être gérées (démarrées, arrêtées, rechargées) grâce au **Tomcat Manager**.





Tomcat Manager

- Tomcat manager requiert un login et un mot de passe.
- Il fournit deux modes de gestion de Tomcat :
 - Une gestion à partir de son interface graphique représentée par le rôle **manager-gui**
 - Une gestion à travers un script (réservées aux IDE comme Eclipse et Netbeans) représentée par le rôle **manager-script**
- Ces deux modes de gestion peuvent être éditées à travers le fichier « tomcat-users.xml » dans le dossier «conf » de Tomcat
- Exemple (pour la version 9):

```
<user username="adminGui" password="adminGui" roles="manager-gui"/>
<user username="adminScript" password="adminScript" roles="manager-script"/>
```

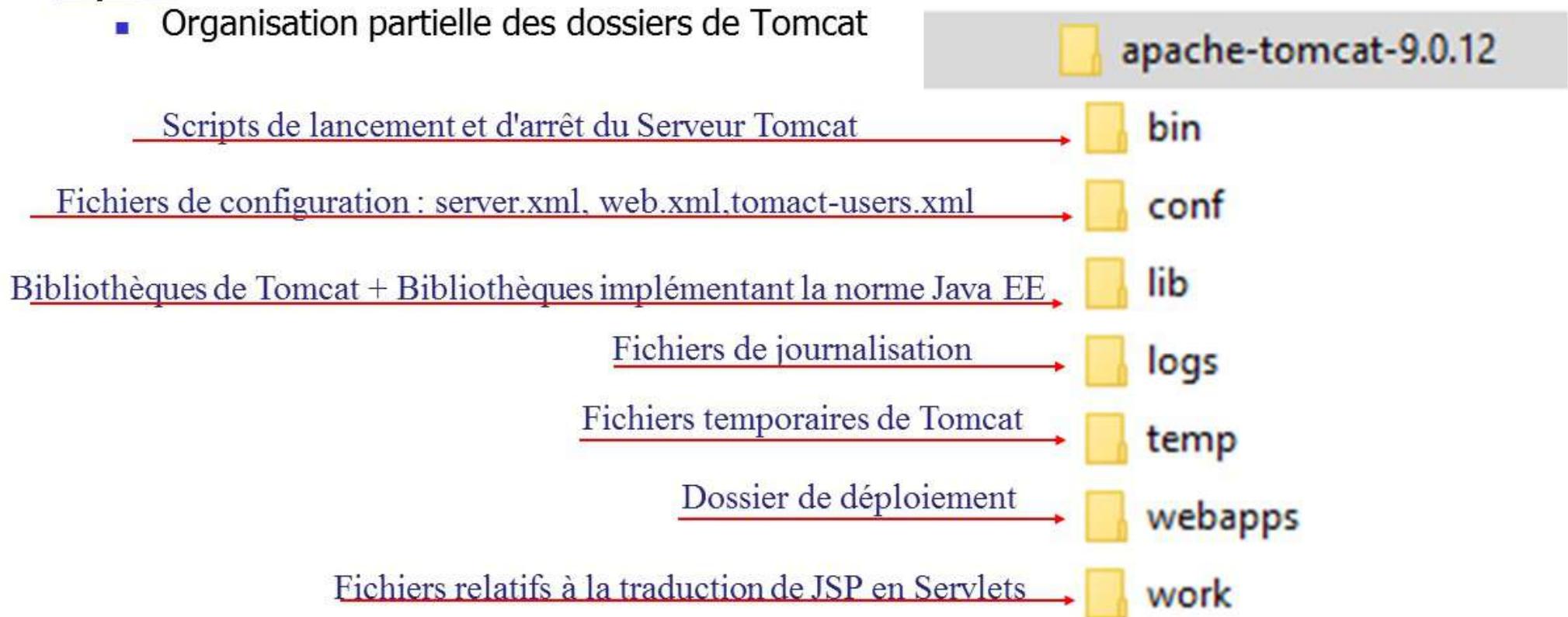
Tomcat Manager

The screenshot shows the Apache Tomcat Manager web interface. At the top, there is a header bar with a logo of a yellow cat, the Apache Software Foundation logo, and navigation links for back, forward, search, and user profile. Below the header, the title "Gestionnaire d'applications WEB Tomcat" is displayed. A message box shows "Message: OK". The main content area has a yellow header titled "Gestionnaire" with four tabs: "Lister les applications", "Aide HTML Gestionnaire", "Aide Gestionnaire", and "Etat du serveur". Below this is a table titled "Applications" with columns: Chemin, Version, Nom d'affichage, Fonctionnelle, Sessions, and Commandes. The table lists three applications:

Chemin	Version	Nom d'affichage	Fonctionnelle	Sessions	Commandes
/	None specified	Welcome to Tomcat	true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/HelloJETEETomact9	None specified		true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Démarrer Arrêter Recharger Retirer Expirer les sessions inactives depuis ≥ 30 minutes

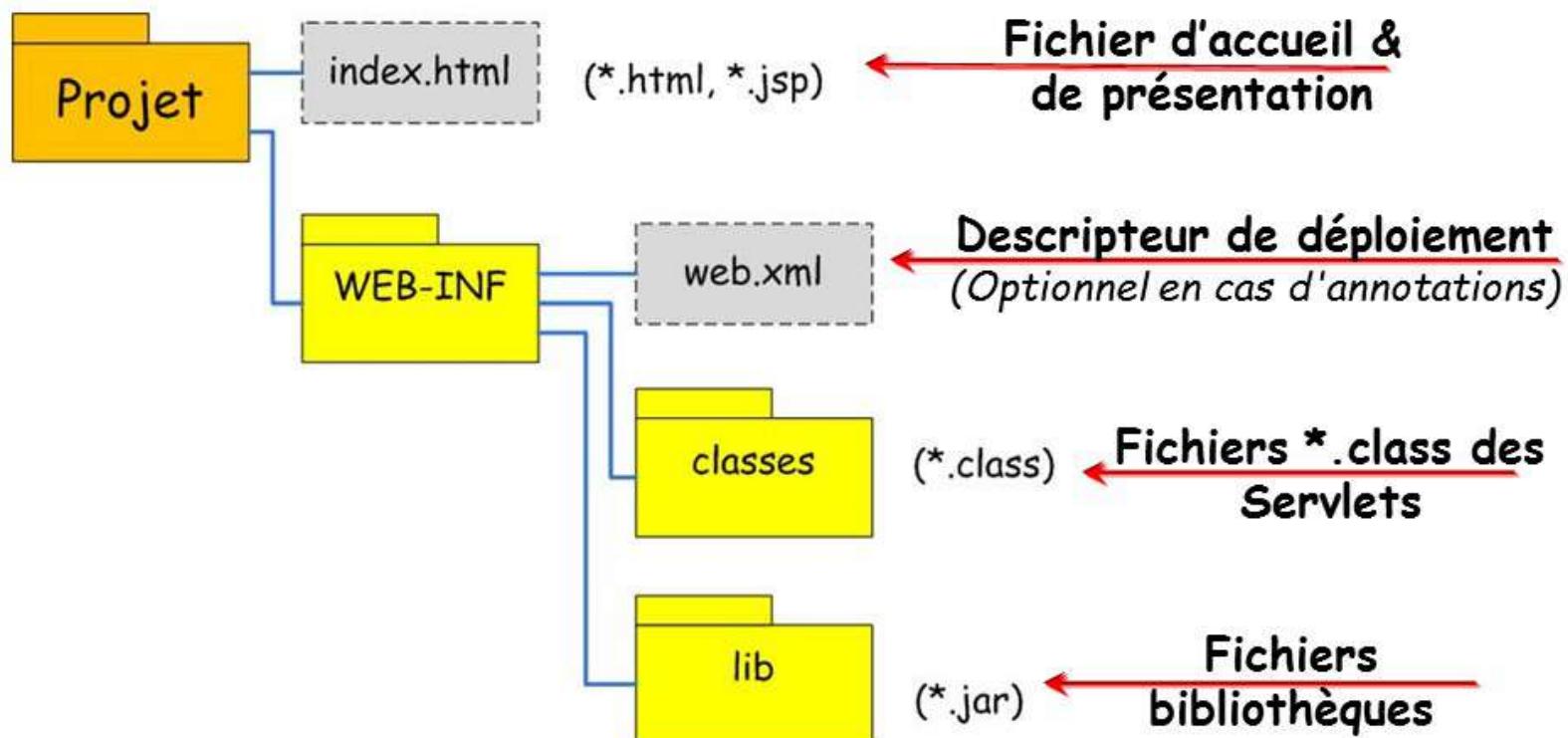
Hiérarchie des dossiers Tomcat

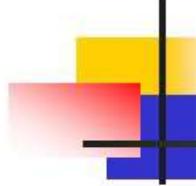
- Organisation partielle des dossiers de Tomcat



Déploiement d'une Servlet dans Tomcat

- Une application Web doit être déployée dans le dossier **webapps** et avoir la structure suivante:





Une première Servlet

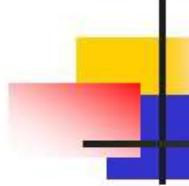
- Une Servlet est :
 - Classe Java
 - Utilise des bibliothèques JEE
 - Qui écrit du HTML
 - Qui a besoin d'une certaine configuration pour tourner sur le Web côté serveur
- Nous allons écrire une Servlet qui écrit Hello World en HTML

Une première Servlet

```
package com.exemple;  
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;  
  
public class HelloWorldServlet extends HttpServlet {  
    // la méthode doGet traite les requêtes envoyées avec GET  
    @Override  
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {  
        res.setContentType("text/html"); // précision du type MIME de contenu à renvoyer au client  
        PrintWriter out = res.getWriter(); // objet responsable d'envoyer du texte au client  
        // rédaction du code HTML à renvoyer au client  
        out.println("<!DOCTYPE html>");  
        out.println("<HTML><HEAD><TITLE> Titre </TITLE></HEAD>");  
        out.println("<BODY> Hello World </BODY></HTML>");  
        out.close(); //fermeture de l'objet Printwriter  
    }  
}
```

Packages pour la création de servlets

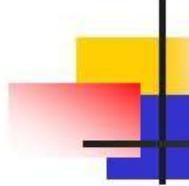
Objet requête Objet réponse



Une première Servlet

- Remarques :

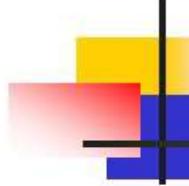
- La méthode **doGet** (resp. **doPost**) traite les requêtes envoyées avec **GET** (resp. **POST**)
- La méthode **doGet** (resp. **doPost**) prend deux paramètres :
 - ▶ Un paramètre de type **HttpServletRequest** représentant la requête client
 - ▶ Un paramètre de type **HttpServletResponse** représentant la réponse à renvoyer au client
- L'objet **HttpServletRequest** permet d'extraire toutes les informations sur le client (adresse IP, navigateur, Domaine, paramètres d'un formulaire, etc..)
- L'objet **HttpServletResponse** doit être complété d'informations par la servlet avant de le renvoyer au client.



Une première Servlet

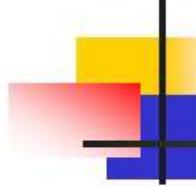
- Remarque :

- Pour compiler une Servlet sous DOS il faudra ajouter la bibliothèque « **servlet-api.jar** » à la variable d'environnement classpath.
- Cette bibliothèque se trouve dans le dossier « **lib** » de Tomcat
- Elle se compose des packages : « javax.servlet » et « javax.servlet.http »



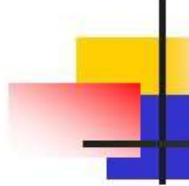
Une première Servlet : fichier web.xml

- Le fichier « web.xml » est le fichier de configuration de l'application Web qu'on est en cours de construire.
- Il permet de donner des informations de l'application à Tomcat comme :
 - Les noms des classes des Servlets
 - Le nom d'affichage de l'application
 - Le chemin virtuel pour accéder aux différents servlets
 - Les fichiers d'accueils
 - Etc..



Une première Servlet : fichier web.xml

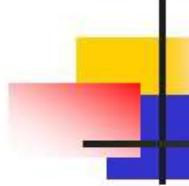
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <display-name>Ma première application Web</display-name>
    <servlet>
        <servlet-name>Hello</servlet-name>
        <servlet-class>com.exemple.HelloWorldServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Hello</servlet-name>
        <url-pattern>/salut</url-pattern>
    </servlet-mapping>
</web-app>
```



Une première Servlet : fichier index.html

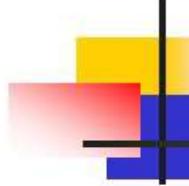
- Le fichier d'accueil par défaut des projets JEE est « index.html » ou « index.jsp »

```
<!DOCTYPE html>
<html>
    <head>
        <title>index</title>
    </head>
    <body>
        <a href=".//salut">Cliquez ici pour lancer la Servlet</a>
    </body>
</html>
```



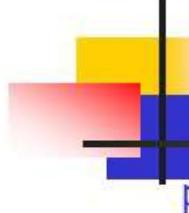
Une première Servlet : chemin d'accès

- Si le dossier contenant notre Servlet se nomme `projet` , le chemin d'accès à notre projet sera :
 - `http://localhost:8080/projet`
- Equivalent à :
 - `http://localhost:8080/projet/index.html`
- Le chemin d'accès à la servlet est :
 - `http://localhost:8080/projet/salut`



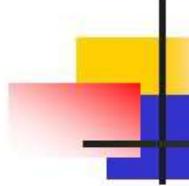
Une première Servlet : avec annotations

- Depuis la norme Servlets 3.0 il est possible de remplacer le fichier « web.xml » par des annotations.
- Ainsi la plupart des informations contenues dans un fichier « web.xml » peut être indiquée par des annotations et leurs attributs.
- Le projet garde exactement la même structure
- Le seul changement est l'absence du fichier web.xml



Une première Servlet : avec annotations

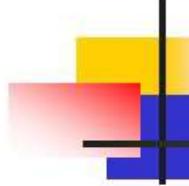
```
package com.exemple;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
@WebServlet("/salut")
public class HelloWorldServlet extends HttpServlet {
@Override
public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<!DOCTYPE html> ");
    out.println("<HTML><HEAD><TITLE> Titre </TITLE></HEAD>");
    out.println("<BODY> Hello World </BODY></HTML>");
    out.close();}
}
```



Une première Servlet : avec annotations

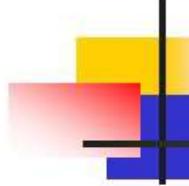
- **Remarque :** Les annotations remplacent tout le paramétrage fait avec le fichier web.xml sauf les fichiers de bienvenue et les pages d'erreurs.

```
<web-app>
    <welcome-file-list>
        <welcome-file>accueil.html</welcome-file>
        <welcome-file>sommaire.html</welcome-file>
    </welcome-file-list>
    <error-page>
        <error-code>404</error-code>
        <location>/erreur404.html</location>
    </error-page>
    <error-page>
        <error-code>500</error-code>
        <location>/erreur500.html</location>
    </error-page>
</web-app>
```



L'API Servlet

- l'API Servlet fournit un ensemble de classes et d'interfaces pour la manipulation des servlets
- Cet API est fourni sous forme d'un kit appelé JSDK (Java Servlet Development Kit)
- L'API servlet regroupe un ensemble de classes dans deux packages :
 - **javax.servlet** : contient les classes pour développer des servlets génériques indépendantes d'un protocole
 - **javax.servlet.http** : contient les classes pour développer des Servlets qui reposent sur le protocole http utilisé par les serveurs web.



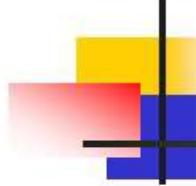
Le package javax.servlet

- Le package javax.servlet définit plusieurs interfaces, méthodes et exceptions :
 - Les interfaces :
 - ▶ **ServletConfig** : Définit d'un objet utilisé par le conteneur de la servlet pour passer de l'information à une servlet pendant son initialisation.
 - ▶ **ServletContext** : Définit un ensemble de méthodes qu'une servlet utilise pour communiquer avec le conteneur de servlets. un objet ServletContext est contenu dans un objet ServletConfig.
 - ▶ **Servlet** : interface de base d'une servlet
 - ▶ **RequestDispatcher** : définit un objet qui reçoit les requêtes du client et les envoie à n'importe quelle ressource (par exemple servlet, fichiers HTML ou JSP) sur le serveur.
 - ▶ **ServletRequest** : Définit un objet contenant la requête du client.



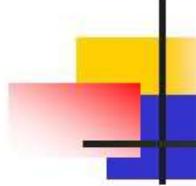
Le package javax.servlet

- ▶ **ServletResponse** : Définit un objet qui contient la réponse renvoyée par la servlet
- ▶ **SingleThreadModel** : Permet de définir une servlet qui ne répondra qu'à une seule requête à la fois
- Les classes :
 - ▶ **GenericServlet** : Classe définissant une servlet indépendante de tout protocoles
 - ▶ **ServletInputStream** : permet la lecture des données de la requête cliente
 - ▶ **ServletOutputStream** : permet l'envoie de la réponse de la servlet
- Les exceptions :
 - ▶ **ServletException** : Exception générale en cas de problème durant l'exécution de la servlet
 - ▶ **UnavailableException** : Exception levée si la servlet n'est pas disponible



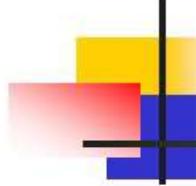
Le package javax.servlet.http

- Le package **javax.servlet.http** définit plusieurs interfaces et méthodes :
 - Les interfaces :
 - ▶ **HttpServletRequest** : Hérite de ServletRequest : définit un objet contenant une requête selon le protocole http
 - ▶ **HttpServletResponse** : Hérite de ServletResponse : définit un objet contenant la réponse de la servlet selon le protocole http
 - ▶ **HttpSession** : Définit un objet qui représente une session



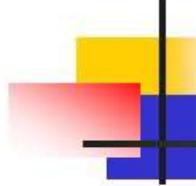
Le package javax.servlet.http

- Les classes :
 - ▶ **Cookie** : Classe représentant un cookie (ensemble de données sauvegardées par le browser sur le poste client)
 - ▶ **HttpServlet** : Hérite de GenericServlet : classe définissant une servlet utilisant le protocole http
 - ▶ **HttpUtils** : Classe proposant des méthodes statiques utiles pour le développement de servlet http (classe devenue obsolète)



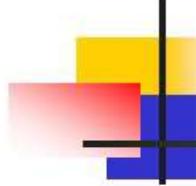
Notion de Contexte

- Une application Web peut être composée de plusieurs types de fichiers (Servlets, JSP, pages html, images, sons, etc.),
- L'ensemble des constituants d'une application est appelé **Contexte** de l'application.
- Les servlets et les JSP d'une application partagent le même contexte.
- Un contexte offre à chaque Servlet (ou JSP) d'une même application une vue sur le fonctionnement de cette application.



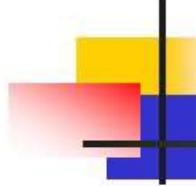
Notion de Contexte

- La description du contexte peut être faite avec :
 - Pour l'ensemble de l'application avec le fichier « web.xml » ou
 - Directement sur chacune des servlets avec les annotations.
- Le contexte d'une application est représenté par un objet appelé ServletContext
- Grâce à cet objet, il est possible d'accéder à chacune des ressources de l'application Web correspondant au contexte.



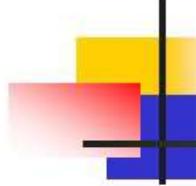
Notion de Contexte

- Quelques méthodes de l'objet ServletContext :
 - **String getInitParameter(String name)** : récupère le paramètre d'initialisation de la servlet, passé en paramètre.
 - **String getServerInfo()** : retourne le nom et la version du conteneur de servlet sur lequel la servlet tourne
 - **int getSessionTimeout()** : retourne le temps d'expiration de la session en secondes.
 - **void log(String msg)** : permet d'écrire dans le fichier journal du conteneur de servlet



L'interface d'une servlet

- Toute Servlet doit implémenter l'interface *Servlet* du package *javax.servlet*
- Le cycle de vie d'une servlet est géré à travers cette interface
- HttpServlet qui implemente l'interface Servlet
- Nos servlets implémentent indirectement l'interface Servlet puisqu'elles héritent de HttpServlet



L'interface d'une servlet

- L'interface d'une servlet se compose des méthodes suivantes :
 - la méthode *init()*
 - la méthode *service()*
 - la méthode *getServletConfig()*
 - la méthode *getServletInfo()*
 - la méthode *destroy()*
- Nous pouvons redéfinir l'une des méthodes au besoin

« interface »

Servlet

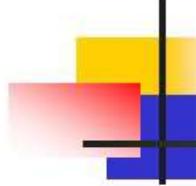
+ **init()**

+ **service()**

+ **getServletConfig()**

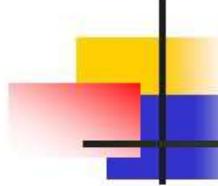
+ **getServletInfo()**

+ **destroy()**



La méthode init()

- Signature :
 - public void init(ServletConfig config) throws ServletException
- Fonctionnement :
 - Est appelée par le conteneur à chaque instantiation de la servlet
 - Lors de l'instanciation, le conteneur de servlet passe en argument à la méthode *init()* un objet *ServletConfig* permettant de charger des paramètres de configuration propres à la servlet.
 - En cas d'anomalie lors de l'appel de la méthode *init()*, celle-ci renvoie une exception de type *ServletException* et la servlet n'est pas initialisée.



La méthode init()

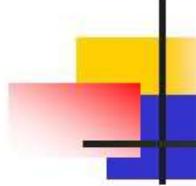
- Exemple :

- Écrire une Servlet qui compte le nombre d'utilisation d'une Servlet depuis son chargement.

- Solution :

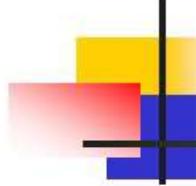
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
```

```
public class ServletCompteur extends HttpServlet {
    private int compteur;
    @Override
    public void init() throws ServletException {
        compteur = 0;
    }
}
```



La méthode init()

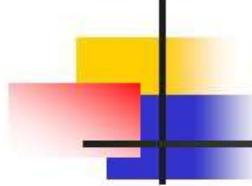
```
@Override  
protected void doGet(HttpServletRequest req, HttpServletResponse res) throws  
ServletException, IOException {  
    res.setContentType("text/plain");  
    PrintWriter out = res.getWriter();  
    compteur++;  
    out.println("Depuis son chargement, on a accédé à cette Servlet " +compteur+"  
fois.");  
}  
}
```



La méthode init()

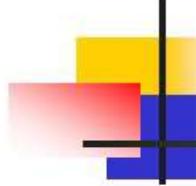
- Il est possible de définir des paramètres d'initialisations pour la servlet (et de les utiliser dans la méthode init) soit avec l'annotation `@WebInitParam` (comme on verra plus loin) soit au niveau du fichier web.xml.
- Exemple :

```
< servlet >
    < servlet-name > Cmp < /servlet-name >
    < servlet-class > Compteur2 < /servlet-class >
    < init-param >
        < param-name > compteur_initial < /param-name >
        < param-value > 50 < /param-value >
        < description > Valeur init du compteur < /description >
    < /init-param >
< /servlet >
```



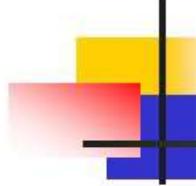
La méthode init()

```
public class ServletCompteur2 extends HttpServlet {  
    private int compteur;  
    @Override  
    public void init() throws ServletException {  
        String initial = this.getInitParameter("compteur_initial");  
        try { compteur = Integer.parseInt(initial);  
        }  
        catch (NumberFormatException e) {  
            compteur= 0;  
        }  
    }  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws  
        ServletException, IOException {  
        //même traitement que l'exemple dernier...  
    }  
}
```



La méthode service()

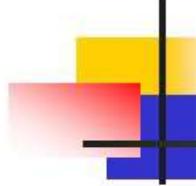
- Signature :
 - public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
- Fonctionnement :
 - Cette méthode est exécutée par le conteneur lorsque la Servlet est sollicitée.
 - Elle détermine le type de requête dont il s'agit, puis transmet la requête et la réponse à la méthode adéquate (*doGet()* ou *doPost*).
 - Chaque requête du client déclenche une seule exécution de cette méthode.



La méthode getServletConfig()

- Signature :
 - public ServletConfig getservletConfig()
- Fonctionnement :
 - Renvoie un objet ServletConfig qui constitue un intermédiaire permettant d'accéder au contexte d'une application.
 - On peut aussi utiliser ServletConfig pour récupérer un paramètre du fichier web.xml :
 - Exemple :

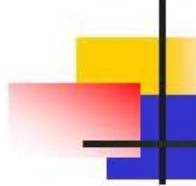
```
String param;  
public void init(ServletConfig config) {  
    param = config.getInitParameter("param");  
}
```



La méthode getServletInfo()

- Signature:
 - public String getServletInfo()
- Fonctionnement :
 - Lorsqu'elle est surchargée permet de retourner des informations sur la servlet comme l'auteur, la version, et le copyright.
 - Ces informations peuvent être exploitées pour affichage par des outils dans les conteneurs Web.
 - Exemple :

```
public String getServletInfo() { return " servlet écrite par x (x@y.com)" ; }
```



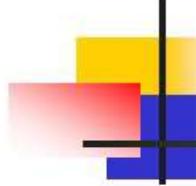
La méthode destroy()

- Signature :

- void destroy()

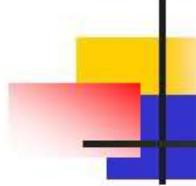
- Fonctionnement :

- La méthode *destroy()* est appelée par le conteneur lors de l'arrêt du serveur Web.
 - Elle permet de libérer proprement certaines ressources (fichiers, bases de données ...).
 - C'est le serveur qui appelle cette méthode.



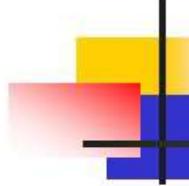
Le cycle de vie d'une servlet

- Le cycle de vie d'une Servlet est assuré par le conteneur de servlet (grâce à l'interface Servlet).
- Cette interface permet à la servlet de suivre le cycle de vie suivant :
 1. le serveur crée un pool de threads auxquels il va pouvoir affecter chaque requête
 2. La Servlet est chargée au démarrage du serveur ou lors de la première requête
 3. La Servlet est instanciée par le serveur
 4. La méthode *init()* est invoquée par le conteneur
 5. Lors de la première requête, le conteneur crée les objets Request et Response spécifiques à la requête



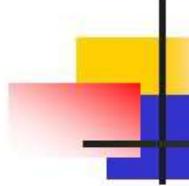
Le cycle de vie d'une servlet

6. La méthode ***service()*** est appelée à chaque requête dans une nouvelle thread. Les objets *Request* et *Response* lui sont passés en paramètre
7. Grâce à l'objet ***Request***, la méthode *service()* va pouvoir analyser les informations en provenance du client
8. Grâce à l'objet ***Response***, la méthode *service()* va fournir une réponse au client
9. La méthode ***destroy()*** est appelée lors du déchargement de la Servlet, c'est-à-dire lorsqu'elle n'est plus requise par le serveur. La Servlet est alors signalée au *garbage collector*



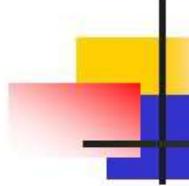
Filtres de Servlets : Définition

- Les filtres de servlet sont des classes Java qui peuvent être utilisées pour :
 - Intercepter les requêtes d'un client avant qu'il n'accède à une ressource en back-end.
 - Manipuler les réponses du serveur avant qu'elles ne soient renvoyées au client.
- La spécification JEE propose plusieurs types de filtres comme :
 - Filtres d'authentification.
 - Filtres de chiffrement.
 - Filtres qui déclenchent des événements d'accès aux ressources.
 - Filtres de journalisation et d'audit...
- Plusieurs filtres peuvent être appliqués de suite : ils sont appelés chaîne de filtres ou Filter Chain.



Filtres de Servlets : Interface Filter

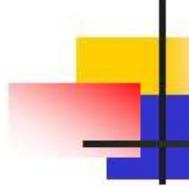
- Les filtres de servlet doivent implémenter l'interface **Filter** qui se compose des trois méthodes suivantes:
 - **public void doFilter (ServletRequest, ServletResponse, FilterChain)** : est appelée par le conteneur chaque fois qu'une paire requête / réponse est transmise à travers la chaîne de filtrage avant qu'une requête client n'accède à une ressource située à la fin de la chaîne.
 - **public void init (FilterConfig filterConfig)** : Cette méthode est appelée par le conteneur Web pour indiquer à un filtre qu'il est mis en service.
 - **public void destroy ()** : Cette méthode est appelée par le conteneur Web pour indiquer à un filtre qu'il est mis hors service.



Filtres de Servlets : Définition dans le web.xml

- Au niveau du fichier web.xml un filtre est défini comme suit :

```
<filter>  
    <filter-name>nomFiltre</filter-name>  
    <filter-class>classeFiltre</filter-class>  
  </filter>  
  
<filter-mapping>  
    <filter-name>nomFiltre</filter-name>  
    <url-pattern>cheminFiltré</url-pattern>  
  </filter-mapping>
```



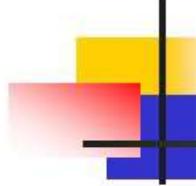
Filtres de Servlets : Exemple

- **Exemple** : Filtre de servlet permettant d'imprimer l'adresse IP du client et l'heure de la date actuelle.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class MonFiltre implements Filter {
    public void init(FilterConfig config) throws ServletException {}

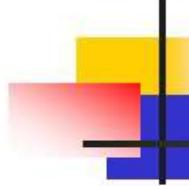
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
        java.io.IOException, ServletException {
        String ipAddress = request.getRemoteAddr();
        System.out.println("IP " + ipAddress + ", Time " + new Date().toString());
        chain.doFilter(request, response); // Repasse la requête à la chaîne de filtrage
    }
    public void destroy() {}
}
```



Filtres de Servlets : Exemple (suite)

- Au niveau du « web.xml »

```
<filter>
    <filter-name>MonFiltre</filter-name>
    <filter-class>MonFiltre</filter-class>
</filter>
<filter-mapping>
    <filter-name>MonFiltre</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```



Filtres de Servlets : Exemple

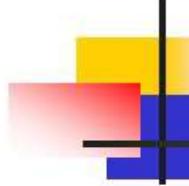
- Remarque :

- le Filtre d'une servlet peut aussi avoir des paramètres d'initialisation.

- Exemple :

```
public void init(FilterConfig config) throws ServletException {  
    String testParam = config.getInitParameter("testParam");  
    System.out.println("Test Param: " + testParam);  
}
```

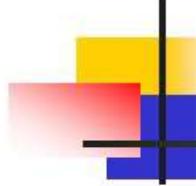
```
<filter>  
    <filter-name>MonFiltre</filter-name>  
    <filter-class>MonFiltre</filter-class>  
    <init-param>  
        <param-name> testParam </param-name>  
        <param-value>Paramètre d'initialisation</param-value>  
    </init-param>  
</filter>
```



Filtres de Servlets : plusieurs filtres

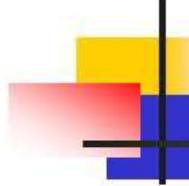
- Remarque : Dans le cas de plusieurs filtres, l'ordre est défini par l'ordre dans le fichier web.xml
 - Exemple : ici **AuthenFilter** précède **LogFilter**

```
<filter-mapping>
    <filter-name>AuthenFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```



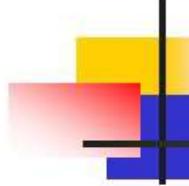
Servlets 3.0

- Depuis la norme Servlets 3.0 il est possible de remplacer le fichier « web.xml » par des annotations.
- Les annotations les plus utilisés (que nous verrons en détail plus loin) sont :
 - **@WebServlet** : permet de déclarer une servlet
 - **@WebInitParam** : permet de déclarer paramètre d'initialisation
 - **@WebFilter** : permet de déclarer un filtre
- Il existe aussi des annotations pour une utilisation plus avancée et sont comme suit :
 - **@WebListener** : permet de déclarer un WebListener
 - **@HandlesTypes** : Pour déclarer les types de classe qu'un ServletContainerInitializer peut gérer.



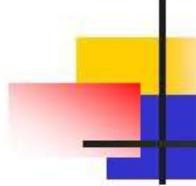
Servlets 3.0

- **@HttpConstraint** : est utilisée dans l'annotation `ServletSecurity` pour représenter les contraintes de sécurité à appliquer à toutes les méthodes du protocole HTTP pour lesquelles un élément `HttpMethodConstraint` correspondant ne se produit pas dans l'annotation `ServletSecurity`.
- **@HttpMethodConstraint** : est utilisée dans l'annotation `ServletSecurity` pour représenter des contraintes de sécurité sur des messages de protocole HTTP spécifiques.
- **@MultipartConfig** : Annotation pouvant être spécifiée sur une classe `Servlet`, indiquant que ses instances attendent des demandes conformes au type MIME multipart / form-data.
- **@ServletSecurity** : Cette annotation est utilisée dans une classe d'implémentation `Servlet` pour spécifier les contraintes de sécurité à appliquer par un conteneur `Servlet` aux messages de protocole HTTP.



L'annotation @WebServlet

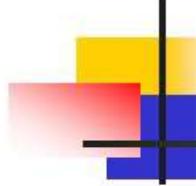
- Sert à déclarer une servlet avec les attributs suivants:
 - String **name** : Nom de la servlet
 - String[] **value** : tableau des URL patterns
 - String[] **urlPatterns** : tableau des URL patterns auxquelles ce filtre s'applique
 - Int **loadOnStartup** : valeur entière de l'indicateur de démarrage
 - WebInitParam[] **initParams** : Tableau de paramètres d'initialisation pour la Servlet
 - Boolean **asyncSupported** : Opération asynchrone prise en charge par la Servlet
 - String **smallIcon** : Petite icône pour cette servlet, si présente
 - String **largeIcon** : Grande icône pour cette servlet, si présente
 - String **description** : Description de cette servlet, si présente
 - String **displayName** : Nom d'affichage de cette servlet, s'il est présent



L'annotation @WebServlet

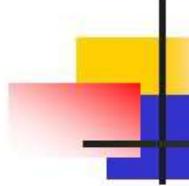
- Exemple :

```
package com.exemple;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
@WebServlet("/salut")
public class HelloWorldServlet extends HttpServlet {
@Override
public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException {
.....
}
}
```



L'annotation @WebInitParam

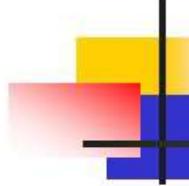
- Sert à définir un paramètre d'initialisation pour une servlet ou un filtre.
- Attributs :
 - String **name** : nom du paramètre d'initialisation
 - String **value** : valeur du paramètre d'initialisation
 - String **description** : description du paramètre d'initialisation



L'annotation @WebInitParam

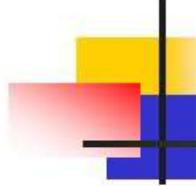
- Exemple :

```
....  
@WebServlet(value = "/salut", initParams = { @WebInitParam(name = "x", value = "Hello "),  
    @WebInitParam(name = "y", value = " World!") })  
public class HelloWorldServlet extends HttpServlet {  
    @Override  
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,  
        IOException {  
        res.setContentType("text/html");  
        PrintWriter out = res.getWriter();  
        out.println("<!DOCTYPE html> ");  
        out.println("<HTML><HEAD><TITLE> Titre </TITLE></HEAD><BODY>");  
        out.println(getInitParameter("x"));  
        out.println(getInitParameter("y"));  
        out.println(" Hello World </BODY></HTML>");  
        out.close();}  
}
```



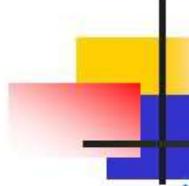
L'annotation @Webfilter

- Sert à déclarer un filtre de servlet qui est appliqué aux URL patterns, servlets et dispatchers.
- Elle admet les attributs suivants :
 - **String filterName** : Nom du filtre
 - **String [] urlPatterns** : Fournit un tableau de valeurs ou urlPatterns auquel le filtre s'applique
 - **DispatcherType [] dispatcherTypes** : Spécifie les types de répartiteur (Request / Response) auxquels le filtre s'applique
 - **String [] servletNames** : Fournit un tableau de noms de servlets
 - **String displayName** : Nom du filtre
 - **String description** : Description du filtre



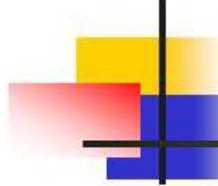
L'annotation @Webfilter

- **WebInitParam [] initParams** : Tableau de paramètres d'initialisation pour ce filtre
 - **Boolean asyncSupported** : Opération asynchrone prise en charge par ce filtre
 - **String smallIcon** : Petite icône pour ce filtre, si présent
 - **String largeIcon** : Grande icône pour ce filtre, si présent
- **Exemple :**
- L'exemple suivant est un simple LogFilter qui affiche la valeur de Init-param test-param et l'horodatage de l'heure actuelle sur la console.
 - Cela signifie que le filtre fonctionne comme une couche entre la requête et la réponse.
 - Ici, nous utilisons "/ *" pour urlPattern (filtre applicable à toutes les servlets)



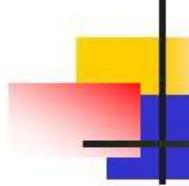
L'annotation @Webfilter

```
import java.io.IOException;
import javax.servlet.annotation.*;
import javax.servlet.*;
import java.util.*;
// classe implementant l'interface Filter
@WebFilter(urlPatterns = {"/*"}, initParams = { @WebInitParam (name = "test-param", value = "paramètre d'initialisation")})
public class LogFilter implements Filter {
    public void init(FilterConfig config) throws ServletException {
        String testParam = config.getInitParameter("test-param"); // récupère le paramètre init
        System.out.println("Test Param: " + testParam); // affiche le paramètre init sur la console
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,
ServletException { System.out.println("Time " + new Date().toString()); // imprime la date courante.
        chain.doFilter(request,response); // repasse la requête à la chaîne de filtres }
    public void destroy( ) {
        /* Appelée avant que l'instance de filtre ne soit supprimée du service par le conteneur Web */
    }
}
```



Développer une servlet http

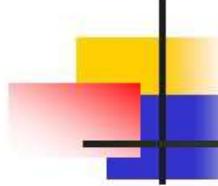
- Les étapes de développement d'une servlet sont les suivantes:
 1. Lecture de la requête (représentée par l'objet *HttpServletRequest*)
 2. Traitement
 3. Crédit de la réponse (représentée par l'objet *HttpServletResponse*)



Développer une servlet http

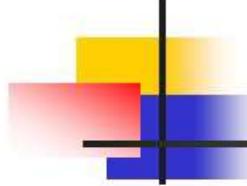
Lecture d'une requête

- L'objet *HttpServletRequest* fournit un ensemble de méthodes pour avoir toutes les informations concernant une requête.
- Ces méthodes sont comme suit :
 - `String getMethod()` : Récupère la méthode HTTP utilisée par le client
 - `String getHeader(String name)`: Récupère la valeur de l'en-tête demandée
 - `String getRemoteHost()` : Récupère le nom de domaine du client
 - `String getRemoteAddr()` : Récupère l'adresse IP du client



Développer une servlet http

- `String getParameter(String name)` : Récupère la valeur du paramètre name d'un formulaire. Lorsque plusieurs valeurs sont présentes, la première est retournée
- `String[] getParameterValues(String name)` : Récupère les valeurs correspondant au paramètre name d'un formulaire, c'est-à-dire dans le cas d'une sélection multiple (cases à cocher, listes à choix multiples) les valeurs de toutes les entités sélectionnées
- `Enumeration getParameterNames()` : Retourne un objet *Enumeration* contenant la liste des noms des paramètres passés à la requête
- `String getServerName()` : Récupère le nom du serveur
- `String getServerPort()` : Récupère le numéro de port du serveur



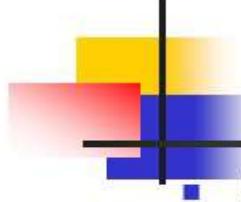
Atelier 2

- Écrire une Servlet qui extrait les informations suivantes de la requête:
 - la méthode d'envoi de la requête HTTP utilisée par le client
 - l'adresse IP du client
 - Le nom du serveur
 - le numéro de port du serveur

Solution (1/2)

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

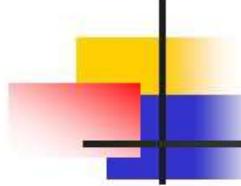
public class TestServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html> <head> <title>entêtes HTTP</title> </head> <body>");
        out.println("Method d'envoi du client :" +request.getMethod());
        out.println("Adresse IP du client : " +request.getRemoteAddr());
        out.println("Nom du serveur : " + request.getServerName());
        out.println("Port du serveur : " + request.getServerPort());
        out.println(" </body></html>");
    }
}
```



Atelier 3

Construire un formulaire HTML comprenant les informations suivantes :

- Nom (zone de texte)
- Prénom (zone de texte)
- Sexe (boutons radio M ou F)
- Fonction (options)
 - ▶ Enseignant
 - ▶ Étudiant (choix initial)
 - ▶ Ingénieur
 - ▶ Retraité
 - ▶ Autre
- Loisirs (cases à cocher)
 - ▶ Lecture
 - ▶ Sport
 - ▶ Voyage
- Commentaire (textarea)



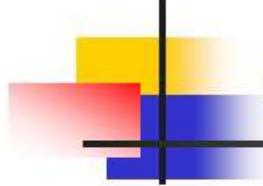
Atelier 3 (suite)

- On demande d'écrire une servlet qui:
 - récupère ces valeurs lorsque l'utilisateur clique sur envoyer.
 - Affiche le nom de chaque champ et la valeur saisie par l'utilisateur

Solution (1/2)

```
<!DOCTYPE html>
<HTML>
    <HEAD> <title> formulaires et servlets </title></HEAD>
    <BODY>
        <FORM method="post" action="traiterFormulaire">
            Enregistrement d'un utilisateur : <br>
            Nom : <INPUT type="text" name="nom"> <br>
            Prénom :<INPUT type="text" name="prenom"><br>
            Sexe : <br> <INPUT type="radio" name="sexe" value="M" checked>Homme<br>
                    <INPUT type="radio" name="sexe" value="F">Femme<br>
            Fonction :<SELECT name="fonction">
                <OPTION VALUE="enseignant">Enseignant</OPTION>
                <OPTION VALUE="etudiant">Etudiant</OPTION>
                <OPTION VALUE="ingenieur" selected>Ingénieur</OPTION>
                <OPTION VALUE="retraite">Retraité</OPTION> <OPTION VALUE="autre">Autre</OPTION>
            </SELECT> <br>
            Loisirs : <br><INPUT type="checkbox" NAME="loisirs" value="lecture" CHECKED>Lecture <br>
                    <INPUT type="checkbox" NAME="loisirs" value="sport">Sport <br>
                    <INPUT type="checkbox" NAME="loisirs" value="voyage">Voyage<br>
            Commentaire :<br><TEXTAREA rows="3" name="commentaire">Votre Commentaire</TEXTAREA><br>
            <INPUT type="submit" value="Envoyer">
        </FORM>
    </BODY>
</HTML>
```

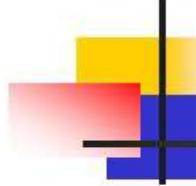
Index.html



Solution (2/2)

Essentiel de la servlet

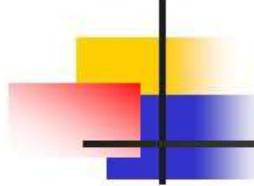
```
...
out.println("<br><H3>Récupération des paramètres utilisateur </H3><br>");
out.println("<br>nom:"+request.getParameter("nom"));
out.println("<br>prenom:"+request.getParameter("prenom"));
out.println("<br>sexe:"+request.getParameter("sexe"));
out.println("<br>fonction:"+request.getParameter("fonction"));
out.println("<br>commentaire:"+request.getParameter("commentaire"));
out.println("essai de getParameterValues<br>");
out.println("<br>loisirs:<br>");
String[] valeursDeLoisirs=request.getParameterValues("loisirs");
for (int i=0 ; i < valeursDeLoisirs.length ; i++) {
    out.println(valeursDeLoisirs[i]+" ");
}
...
```



Développer une servlet http

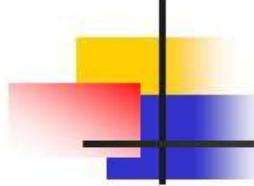
Création de la réponse

- Après lecture et traitement d'une requête, la réponse à fournir à l'utilisateur est représentée sous forme d'objet *HttpServletResponse*.
- Les méthodes de l'objet *HttpServletResponse* sont comme suit :
 - `void setHeader(String Nom, String Valeur)` : Définit une paire clé/valeur dans les en-têtes
 - `void setContent-Type(String type)` : Définit le type MIME de la réponse HTTP, c'est-à-dire le type de données envoyées au navigateur



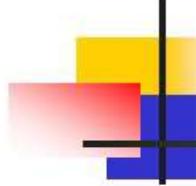
Développer une servlet http

- `void setContentLength(int len)` : Définit la taille de la réponse
- `PrintWriter getWriter()` : Retourne un objet *PrintWriter* permettant d'envoyer du texte au navigateur client. Il se charge de convertir au format approprié les caractères Unicode utilisés par Java
- `ServletOutputStream getOutputStream()` : Définit un flot de données à envoyer au client, par l'intermédiaire d'un objet *ServletOutputStream*, dérivé de la classe *java.io.OutputStream*
- `void sendredirect(String location)` : Permet de rediriger le client vers l'URL *location*



Développer une servlet http

- String **setStatus(int StatusCode)** : Définit le code de retour de la réponse
- Rappelons quelques codes de retour:
 - ▶ Code 202 (**SC_ACCEPTED**) : Requête acceptée.
 - ▶ Code 204 (**SC_NO_CONTENT**) : pas d'information à retourner.
 - ▶ Code 301 (**SC_MOVED_PERMANENTLY**) : la ressource demandée a été déplacée.
 - ▶ Code 400 (**SC_BAD_REQUEST**) : La requête est syntaxiquement incorrecte.
 - ▶ Code 403 (**SC_FORBIDDEN**) : le serveur comprend la requête mais refuse de la servir.
 - ▶ Code 404 (**SC_NOT_FOUND**) : la ressource demandée n'est pas disponible.
 - ▶ etc...

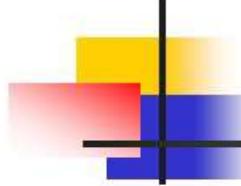


Développer une servlet http

- Exemple :

- Écrire une Servlet qui effectue une redirection vers un site web donné.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class PremiereServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
        IOException {
        res.sendRedirect("http://www.google.co.ma");
    }
}
```

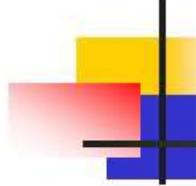


Développer une servlet http

- Remarque :

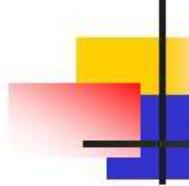
- Pour éviter que la page soit rechargée à partir du cache :

- ▶ `response.setHeader("Cache-Control","no-cache"); //HTTP 1.1`
 - ▶ `response.setHeader("Pragma","no-cache"); //HTTP 1.0`
 - ▶ `response.setDateHeader ("Expires", 0); /*prevents caching at the proxy server*/`



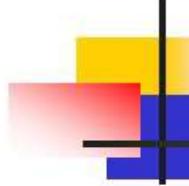
Suivi de session

- Le protocole HTTP est un protocole sans état
- Impossibilité alors de garder des informations d'une requête à l'autre (identifier un client d'un autre)
- Obligation d'utiliser différentes solutions pour remédier au problème d'état dont :
 - L'utilisation de cookies
 - L'utilisation de sessions



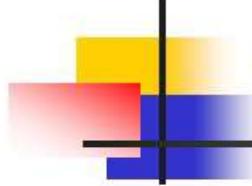
Suivi de session : cookies

- Les cookies représentent un moyen simple de stocker temporairement des informations chez un client, afin de les récupérer ultérieurement.
- Les cookies ont été introduits par la première fois dans Netscape Navigator
- Les cookies font partie des spécifications du protocole HTTP.
- L'en-tête HTTP réservé à l'utilisation des cookies s'appelle Set-Cookie, il s'agit d'une simple ligne de texte de la forme:
 - **Set-Cookie** : **nom=VALEUR**; **domain=NOM_DE_DOMAINE**; **expires=DATE**
- La valeur d'un cookie pouvant identifier de façon unique un client, ils sont souvent utilisés pour le suivi de session



Suivi de session : cookies

- L'API Servlet fournit la classe *javax.servlet.http.Cookie* pour travailler avec les Cookies
 - *Cookie(String name, String value)* : construit un cookie
 - *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 en secondes
- Pour la création d'un nouveau cookie, il faut l'ajouter à la réponse (*HttpServletResponse*)
 - *addCookie(Cookie mon_cook)* : ajoute à la réponse un cookie
- La Servlet récupère les cookies du client en exploitant la requête (*HttpServletRequest*)
 - *Cookie[] getCookies()* : récupère l'ensemble des cookies du site



Suivi de session : cookies

- Code pour créer un cookie et l'ajouter au client :

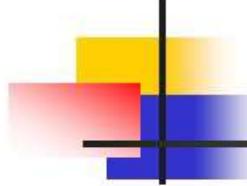
```
Cookie c = new Cookie("Id", "123");  
res.addCookie(c);
```

- Code pour récupérer les cookies

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

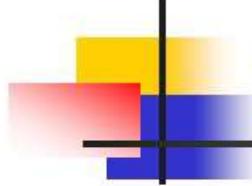
- Remarque :

- Il n'existe pas dans l'API Servlet de méthode permettant de récupérer la valeur d'un cookie par son nom



Atelier 6

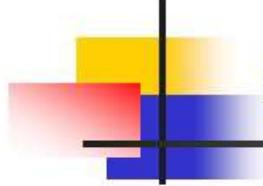
- Écrire une Servlet permettant d'identifier un client par l'intermédiaire des cookies.
 - Pour cela nous allons stocker et chercher un cookie que nous allons appeler monID et dont la valeur est générée grâce à la méthode `java.rmi.server.UID().toString()` qui permet d' obtenir un identifiant unique sur le temps par rapport à l'hôte sur lequel il a été généré.



Solution (1/2)

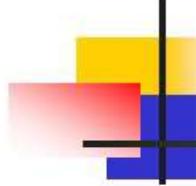
```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class CookiesServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
        IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        String contenu= null;
        Cookie[] tabCookies = req.getCookies();
        if (tabCookies != null)
            for (int i = 0; i < tabCookies .length; i++) {
                if (tabCookies [i].getName().equals("monId"))
                    contenu = tabCookies[i].getValue();
            }
    }
}
```



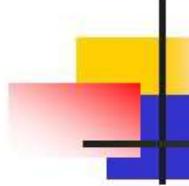
Solution (2/2)

```
if (tabCookies == null || contenu == null) {  
    out.println("Bonjour le nouveau...mais plus maintenant");  
    contenu = new java.rmi.server.UID().toString();  
    Cookie c = new Cookie("monId", contenu);  
    c.setMaxAge(60*60*24*365);  
    res.addCookie(c);  
  
}  
else out.println("Encore vous");  
out.close();  
}  
}
```



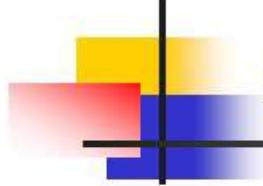
Suivi de session : HttpSession

- 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
- Les navigateurs n'acceptent que 20 cookies par site, 300 par utilisateur et la taille d'un cookie peut être limitée à 4096 octets (4 ko)



Suivi de session : HttpSession

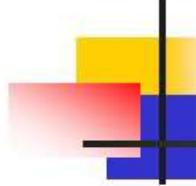
- Solutions : utilisation de l'API de suivi de session *javax.servlet.http.HttpSession*
- Méthodes de création liées à la requête (*HttpServletRequest*)
 - *HttpSession getSession()* : retourne la session associée à l'utilisateur si elle existe sinon lui crée une nouvelle
 - *HttpSession getSession(boolean p)* : création selon la valeur de *p*
- Gestion d'association (*HttpSession*)
 - *setAttribute(String an, Object av)* : associe l'objet *av* à la chaîne *an*
 - *Object getAttribute(String name)* : retourne l'attribut *name*
 - *Enumeration getAttributeNames()* : retourne les noms de tous les attributs
 - *removeAttribute(String na)* : supprime l'attribut associé à *na*
- Destruction (*HttpSession*)
 - *invalidate()* : expire la session



Suivi de session : HttpSession

- Exemple : Servlet qui permet d'utiliser le suivi de session pour un compteur

```
public class HttpSessionServlet extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        HttpSession s = req.getSession();  
        Integer compt= (Integer)s.getAttribute("compteur");  
        if (compt== null)  
            compt = 1;  
        else {  
            compt = compt+ 1;}  
        s.setAttribute("compteur", compt);  
        out.println("Vous avez visité cette page " + compt + " fois.");  
    }  
}
```



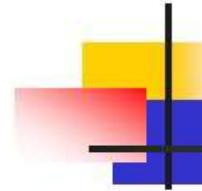
Collaboration de Servlets

- Les Servlets s'exécutant dans le **même** serveur peuvent dialoguer entre elles
- Deux principaux styles de collaboration :
 - **Partage d'information** : un état ou une ressource.

Exemple : un magasin en ligne pourrait partager les informations sur le stock des produits ou une connexion à une base de données

- **Partage du contrôle** : une requête.

Réception d'une requête par une Servlet et laisser l'autre Servlet une partie ou toute la responsabilité du traitement



Collaboration de Servlets : partage d'information

- La collaboration est obtenue par l'interface *ServletContext*
- L'utilisation de *ServletContext* permet aux applications web de disposer de son propre conteneur d'informations unique
- Une Servlet retrouve le *ServletContext* de son application Web par un appel à *getServletContext()*
- Exemples de méthodes :
 - *void setAttribute(String nom, Object o)* : lie un objet sous le nom indiqué
 - *Object getAttribute(String nom)* : retrouve l'objet sous le nom indiqué
 - *Enumeration.getAttributeNames()* : retourne l'ensemble des noms de tous les attributs liés
 - *void removeAttribute(String nom)* : supprime l'objet lié sous le nom indiqué

Partage d'information

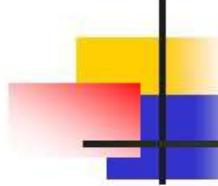
- **Exemple** : Servlets qui vendent des pizzas et partagent une spécialité du jour

```
public class PizzasAdmin extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        ServletContext context = this.getServletContext();  
        context.setAttribute("Specialite", "Quatre saisons");  
        out.println("La pizza du jour a été définie."); }  
}
```

Création d'un attribut

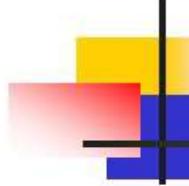
```
public class PizzasClient extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,  
        IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        ServletContext context = this.getServletContext();  
        String pizza_spec = (String)context.getAttribute("Specialite");  
        out.println("Aujourd'hui, notre spécialité est : " + pizza_spec); }  
}
```

Lecture de l'attribut



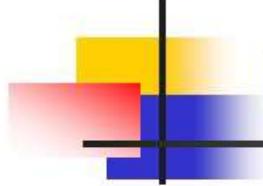
Collaboration de Servlets : partage du contrôle

- Pour une collaboration dynamique entre servlets, deux possibilités existent:
 - Déléguer entièrement la requête à une autre servlet : méthode **forward()**
 - Inclure la réponse d'une autre servlet dans la servlet en cours : méthode **include()**
- Ces deux méthodes appartiennent à l'interface **RequestDispatcher** du package javax.servlet
 - *RequestDispatcher getRequestDispatcher(String path)* : retourne une instance de type *RequestDispatcher* par rapport à un composant
 - Un composant peut-être de tout type : Servlet, JSP, fichier statique, ...
 - *path* est un chemin relatif ou absolu ne pouvant pas sortir du contexte



Partage du contrôle (forward)

- Soit l'exemple suivant :
 - Une servlet (Servlet1) reçoit une requête
 - Elle y place un attribut *X* qu'elle y met la chaîne "salut"
 - Elle renvoie ensuite cette requête à une autre Servlet (Servlet2).
 - Servlet2 récupère cet attribut et se charge de créer la réponse qu'elle renvoie à l'utilisateur.
- Attention:
 - Servlet1 ne doit pas toucher à la réponse car c'est Servlet2 qui s'en charge.
 - Après le renvoi de la requête à Servlet2, Servlet1 ne doit plus toucher à la requête.

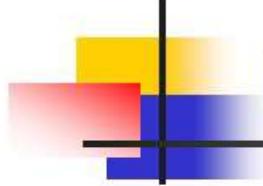


Partage du contrôle (forward)

Code pour servlet1

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Servlet1 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        req.setAttribute("X", "salut");
        RequestDispatcher dispat = req.getRequestDispatcher("/cheminServlet2");
        dispat.forward(req,res);
    }
}
```

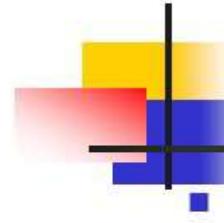


Partage du contrôle (forward)

Code pour servlet2

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

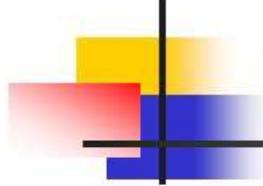
public class Servlet2 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println("l'attribut que j'ai récupéré de servlet 1 est: "+req.getAttribute("X"));
        out.close();
    }
}
```



Partage du contrôle (include)

Soit l'exemple suivant :

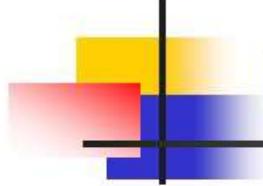
- Une servlet (Principale) reçoit une requête
 - Elle y place un attribut x qu'elle y met la chaîne "3"
 - Elle inclut une autre Servlet dans le traitement (Secondaire)
 - Secondaire récupère cet attribut et affiche son carré
 - Principale reprend le contrôle, elle modifie x en "5"
 - Secondaire récupère encore une fois cet attribut et affiche son carré
 - Principale reprend le contrôle
- Remarque:
- Secondaire ne doit pas fermer la réponse par </body> car c'est Principale qui s'en charge.
 - C'est Principale qui se charge de préciser le type MiMe de la réponse.



Partage du contrôle (include)

Code pour Principale

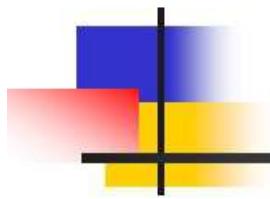
```
public class Principale extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/html");  
        PrintWriter out = res.getWriter();  
        out.println("<HTML><BODY>");  
        req.setAttribute("x", "3");  
        RequestDispatcher dispat = req.getRequestDispatcher("/cheminServlet2");  
        dispat.include(req,res);  
        req.setAttribute("x", "5");  
        dispat.include(req,res);  
        out.println("</BODY></HTML>");  
    }  
}
```



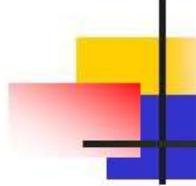
Partage du contrôle (include)

Code pour Secondaire

```
public class Secondaire extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        PrintWriter out = res.getWriter();  
        String ch=(String)req.getAttribute("x");  
        int x=Integer.parseInt(ch);  
        out.println(" Le carré de " + x + " est :" + x*x);  
    } }
```

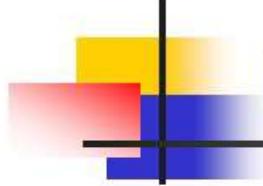


Chapitre 3 : Java Server Pages (JSP)



Introduction

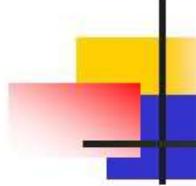
- JSP : Java Server Pages
- Java Server Pages est une technologie qui combine Java et des Tags HTML dans un même document pour produire un fichier JSP.
- But : faciliter la génération dynamique de contenu de sites Web.
- Similitudes : PHP, ASP, etc..



Exemple de fichier JSP

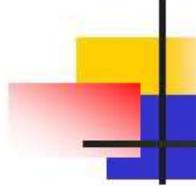
test.jsp

```
<html>
    <head>
        <title>Exemple JSP</title>
    </head>
    <body>
        la somme de 2 et 2 est <%=2+2%>
    </body>
</html>
```



Serveur Web

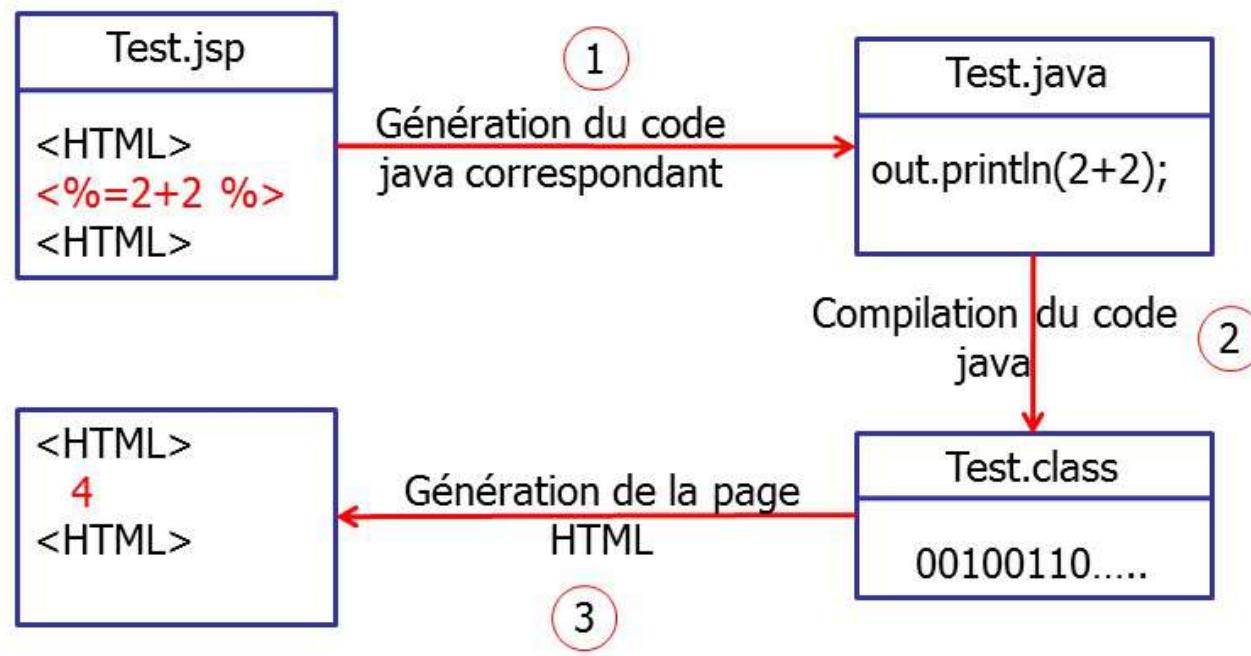
- L'utilisation des JSP implique d'avoir un serveur HTTP (logiciel servant à diffuser les pages Web) disposant d'une extension capable de traiter les JSP.
- Exemple de serveurs HTTP gratuit supportant JSP:
 - **Tomcat** proposé par la fondation Apache.
 - **JSDK** proposé par SUN
- Comme pour les servlets, nous travaillerons avec Tomcat.

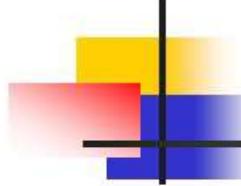


Traitement des JSP

- L'interprétation d'une page contenant des instructions JSP se fait de la manière suivante :
 1. L'utilisateur demande, via son navigateur (client), un document possédant l'extension .jsp
 2. Le serveur HTTP lance une *servlet* (application Java serveur) qui construit le code Java à partir du code contenu dans la page HTML.
 3. Le programme résultant est compilé puis exécuté sur le serveur.
 4. Le résultat est réintroduit dans la page renvoyée au client.

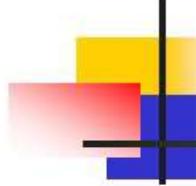
Traitement des JSP





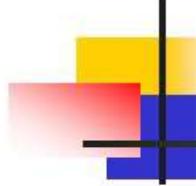
Structure d'un fichier JSP

- Similaire à la structure d'un fichier HTML
- Elle se compose essentiellement de quatre types de tags:
 - Tag de directive
 - Tag de commentaire
 - Tag de Scriptlet
 - Tag d'expression



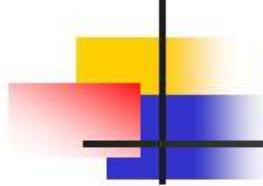
Directives JSP

- Les directives contrôlent comment le serveur WEB doit générer la Servlet
- Elles sont placées entre les symboles `<%@` et `%>`
- Syntaxe : `<%@ directive { attribut="valeur"} %>`
- Les directives les plus importantes sont:
 - **include** : indique au compilateur d'inclure un autre fichier
 - **page** : définit les attributs spécifiques à une page



Directives JSP : include

- Cette inclusion se fait au *moment de la conversion*
- Tout le contenu du fichier externe est inclus comme s'il était saisi directement dans la page JSP
- Pas de séparation de la portée des variables
- Exemple :
 - <%@ include file="unAutreFichier.jsp" %>



Exemple pratique

entete.html

```
<HTML>
<HEAD>
<TITLE>Page de démonstration</TITLE>
</HEAD>
<BODY>
je suis dans l'entête de la page<br>
```

corps.jsp

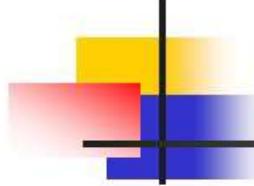
```
<%! String mon_nom; %>
<% mon_nom = "Ali"; %>
```

piedpage.html

```
<br>Je suis dans le pied de page.
</BODY>
</HTML>
```

index.jsp

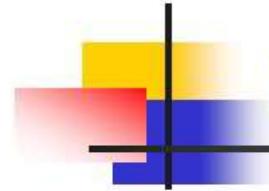
```
<%@ include file = "/entete.html" %>
<%@ include file = "/corps.jsp" %>
Bonjour <%= mon_nom %>
<%@ include file = "/piedpage.html" %>
```



Directives JSP : page

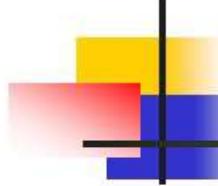
- La directive **page** définit les attributs spécifiques à une page.
- La liste des attributs possibles pour la directive **page** est comme suit :

Attribut	exemple valeurs	Description
language	java	Indique le langage utilisé. Java par défaut
extends	Package.class	Hérite de l'interface du package choisi.
session	false	Si initialisé à false vous ne pouvez pas utiliser les sessions. True par défaut.
import	Java.util.* , *.class , java.*	Importe les classes dont vous avez besoin.
buffer	5 kb	Taille en kb de la mémoire tampon qui contient le flux de données à imprimer sur la JSP. 8 kb par défaut.
autoflush	true	Si à false il ne vide pas automatiquement le buffer une fois rempli. Si vous avez mis le buffer à none vous ne pouvez mettre autoFlush à false. Défaut à true.



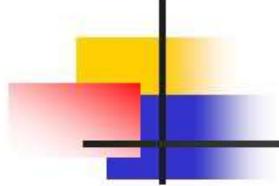
Directives JSP : page

Attribut	Exemple de valeur	Description
isThreadSafe	false	Si à false le serveur d'applications ne permet qu'à un client à la fois d'accéder à la JSP. Défaut à true.
info	Ma première JSP	Information qui apparaît dans le document jsp compilé et utilisé par le serveur.
errorPage	Erreupage.html Erreur.jsp	Adresse de la page d'erreur sur laquelle est renvoyé le visiteur en cas d'erreur (Exception) de la JSP.
contentType	text/html	Le type MIME et le jeu de caractères à utiliser dans cette JSP. Par défaut text/html; charset=ISO-8859-1
isErrorPage	true	Si true la JSP peut afficher l'erreur renvoyée par l'exception. True par défaut.
pageEncoding	ISO-8859-1	ISO-8859-1 par défaut.



Directives JSP : page

- Remarque :
 - Vous n'avez pas besoin d'importer les classes suivantes, qui le sont déjà implicitement:
 - ▶ java.lang.*
 - ▶ javax.servlet.*
 - ▶ javax.servlet.http.*
 - ▶ javax.servlet.jsp.*
- Exemple de directives :
 - <%@ page import="java.util.*" errorPage="erreur.jsp" buffer="5kb" session="false" %>



Éléments de scripts JSP : commentaire

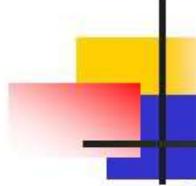
- Cet élément de script est utilisé pour faire un commentaire dans le code JSP
- Le texte dans un commentaire JSP ne sera pas envoyé au client ni compilé dans la Servlet
- Les commentaires sont placés entre les symboles `<%--` et `--%>`

Example.jsp:

```
<html>
    <!-- commentaire HTML -->
    <%-- commentaire JSP --%>
</html>
```



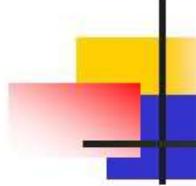
```
<html>
    <!-- commentaire HTML -->
</html>
```



Éléments de scripts JSP : déclaration

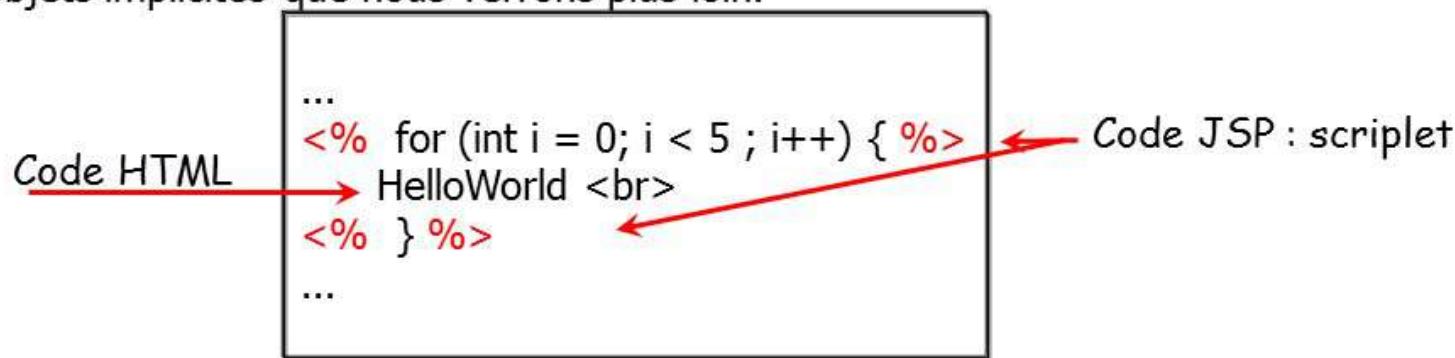
- Une déclaration permet d'insérer du code dans la classe de la Servlet
- Les déclarations sont placées entre les symboles `<%!` et `%>`
- Exemple:

```
<%!
    private int count = 0;
    private int incrementCount() { return count++;}
%>
```
- Elle peut être utilisée pour :
 - Déclarer un attribut de classe
 - Spécifier et implémenter des méthodes
 - Les attributs et les méthodes déclarées dans la page JSP sont utilisables dans toute la page JSP



Éléments de scripts JSP : scriptlet

- C'est un bloc de code Java qui est placé dans `_jspService(...)` de la Servlet générée (équivalent à `service(...)`)
- Les scriplets sont placées entre les symboles `<%>` et `%>`
- Tout code java a accès :
 - aux attributs et méthodes définis par le tag déclaration `<%! ... %>`
 - aux objets implicites que nous verrons plus loin.



Éléments de scripts JSP : expression

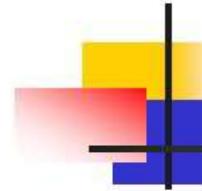
- Sert à évaluer une expression et à renvoyer sa valeur
- Les expressions sont placées entre les symboles `<%=` et `%>`
- Retourne une valeur String de l'expression
- Correspond à une scriptlet de la forme `<% out.println(...); %>`
- Se transforme en `out.println("...");` dans la méthode `_jspService(...)` après génération

```
...
<% String[] noms={"Ali","Omar","Hassan"};
   for (int i = 0 ; i < noms.length ; i++) { %>
      Le <%= i %> ème nom est <%= noms[i] %>
   <% } %>
...

```

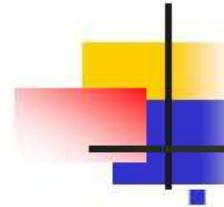
scriptlet

expression



Éléments de scripts JSP : scriptlet et objets implicites

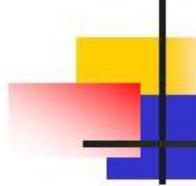
- Les objets implicites sont les objets présents dans la méthode *service(...)* qui ont été employés dans la partie Servlet
- Ils sont identifiés par des noms de variables uniques :
 - **request** : requête courante
 - **response** : réponse courante
 - **session** : session courante
 - **out** : flot de sortie permet l'écriture sur la réponse
 - **application** : contient des méthodes log() permettant d'écrire des messages dans le journal du contenu (*ServletContext*)
 - **pageContext** : utilisé pour partager directement des variables entre des pages JSP et supportant les beans et les balises
 - **exception** : disponible uniquement dans les pages erreurs donnant information sur les erreurs



Éléments de scripts JSP : scriptlet et objets implicites

Exemple : JSP qui récupère des informations du client

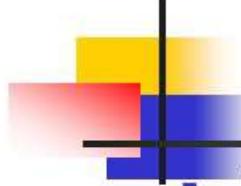
```
<%@ page language="java" contentType="text/html" %>
<html>
<head>
<title>Informations client</title>
</head>
<body bgcolor="white">
    Protocol : <%= request.getProtocol() %><br>
    Scheme : <%= request.getScheme() %><br>
    ServerName : <%= request.getServerName() %><br>
    ServerPort : <% out.println(request.getServerPort()); %><br>
    RemoteAddr : <% out.println(request.getRemoteAddr()); %><br>
    RemoteHost : <% out.println(request.getRemoteHost()); %><br>
    Method : <%= request.getMethod() %><br>
</body>
```



Cycle de vie d'une JSP

- Le cycle de vie d'une Java Server Page est identique à une Servlet :
 - La méthode *jspInit()* est appelée après le chargement de la page
 - La méthode *_jspService()* est appelée à chaque requête
 - La méthode *jspDestroy()* est appelé lors du déchargement (fermeture d'une base de données)
- Possibilité de redéfinir dans le code JSP les méthodes *jspInit()* et *jspDestroy()* en utilisant un élément de scripts déclaration
- Exemple :

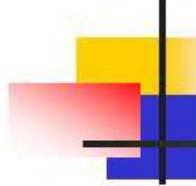
```
<html> <head><title>Bonjour tout </title></head><body>
<%! int compteur; %>
<%! public void jspInit() {
    compteur = 0;} %>
La valeur du compteur est <%= compteur++ %>
</body></html>
```



Cycle de vie d'une JSP

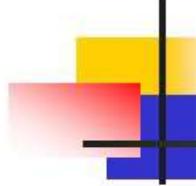
Exemple : un compteur avec une initialisation et une destruction

```
<%@ page language="java" contentType="text/html" %>
<%@ page import="java.util.Date" %>
<%!
int global_counter = 0;
Date start_date;
public void jspInit() {
start_date = new Date();
}
public void jspDestroy() {
ServletContext context = getServletConfig().getServletContext();
context.log("test.jsp a été visitée " + global_counter + "fois entre le
" + start_date + " et le " + (new Date()));
}
%>
<html>
<head><title>Page avec un compteur</title></head>
<body bgcolor="white">
Cette page a été visitée : <%= ++global_counter %> fois depuis le <%
start_date %>.
</body></html>
```



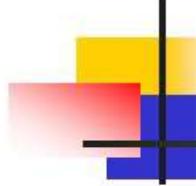
JSP et Actions

- Les actions permettent de faire des traitements au moment où la page est demandée par le client
 - utiliser des Java Beans
 - inclure dynamiquement un fichier
 - rediriger vers une autre page
- Les actions sont ajoutées à la page JSP à l'aide d'une syntaxe d'éléments XML (définis par des balises personnalisées). Exemple :
 - <ma:balise ... />
 - <ma:balise ... > ...</ma:balise>



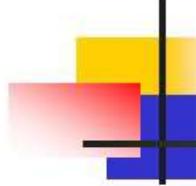
Java Beans

- Permet de coder la logique métier de l'application WEB
- L'état d'un Bean est décrit par des attributs appelés propriétés
- La spécification des Java Beans définit les Beans comme des classes qui supportent
 - Introspection : permet l'analyse d'un Bean (nombre de propriétés)
 - Événements : métaphore de communication
 - Persistance : pour sauvegarder l'état d'un Bean
 - ...



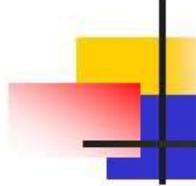
Java Beans

- Les Java Beans sont des classes Java normales respectant un ensemble de directives
 - A un constructeur public sans argument
 - Les propriétés d'un Bean sont accessibles au travers de méthodes *getXXX*(lecture) et *setXXX* (écriture) portant le nom de la propriété
- Lecture et écriture des propriétés
 - *type getNomDeLaPropriété()* : pas de paramètre et son type est celui de la propriété
 - *void setNomDeLaPropriété(type)* : un seul argument du type de la propriété et son type de retour est void
- En option, un Java Beans implémente l'interface *java.io.Serializable* permettant la sauvegarde de l'état du Bean



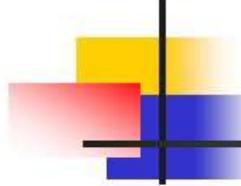
Exemple : classe Client

```
public class Client {  
    // attributs  
    private String nom;  
    private String adresse;  
    // méthodes d'accès et de modification  
    public String getNom () { return nom; }  
    public void setNom (String nm) { nom=nm; }  
    public String getAdresse () { return adresse; }  
    public void setAdresse (String adr) { adresse=adr; }  
}
```



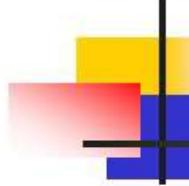
Java Beans et JSP

- Pour déclarer et allouer un Java Beans dans une page JSP il faut employer l'action <jsp:useBean>
- Exemple :
- <jsp:useBean id="nomObjet" class="Package.nomClasse" scope="**page** ou **request** ou **session** ou **application**" />
 - id : nom de l'instance pour identification
 - class : Nom de la classe du bean



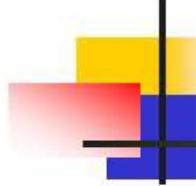
Java Beans et JSP

- scope : portée de la validité de l'objet Bean :
 - ▶ *page* : Bean valide pour la requête sans transmission
 - ▶ *request* : Bean valide pour la requête et peut être transmise (forward)
 - ▶ *session* : Bean ayant la durée de vie de la session
 - ▶ *application* : Bean créée pour l'application WEB courante



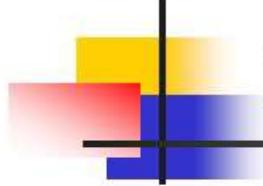
Java Beans et JSP : lecture propriétés

- Pour lire une propriété du Bean deux éléments sont utilisés
 - La référence du Bean définie par l'attribut id
 - Le nom de la propriété
- Deux manières existent pour interroger la valeur d'une propriété et la convertir en String
 - En utilisant un tag action *<jsp:getProperty>*
 - ▶ *Syntaxe:* <jsp:getProperty name="référence Bean" property="nom propriété" />
 - En utilisant l'élément de scripts JSP : expression
 - ▶ <%= nom_instance.getNomPropriete() %>



Java Beans et JSP : écriture propriétés

- Modification de la valeur d'une propriété en utilisant `<jsp:setProperty>`
- Syntaxe:
 - `<jsp:setProperty name="référence Bean" property="nom propriété" value="valeur" />`
 - `<jsp:setProperty name="référence Bean" property="nom propriété" param="nomParametre" />`



Java Beans et JSP : lecture et écriture propriétés

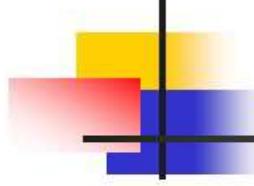
- Exemple : Soit le java bean suivant :

```
package toto;
public class Client {
    private String nom;
    private String adresse;
    public String getNom () { return nom; }
    public void setNom (String nm) { nom=nm; }
    public String getAdresse () { return adresse; }
    public void setAdresse (String adr) { adresse=adr; }
}
```



Attention :

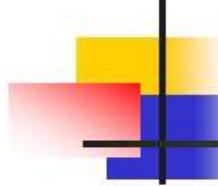
les classes des beans doivent
être mise dans le répertoire
WEB-INF/classes



Java Beans et JSP : lecture et écriture propriétés

- Exemple d'utilisation du bean précédent:

```
<jsp:useBean id="cl" class="toto.Client"/>
<% cl.setNom("Ali");
   cl.setAdresse("31, Bd des FAR, Casablanca");
%>
<html>
  <head>
    <title>Page pour lecture d'information</title>
  </head>
  <body bgcolor="white">
    Nom du Client : <%= cl.getNom() %><br>
    Adresse du Client : <jsp:getProperty name="cl" property="adresse"/>
  </body>
</html>
```



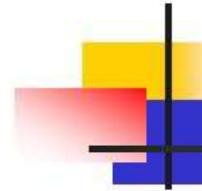
Java Beans et JSP : lecture et écriture propriétés

- Modification de l'ensemble des propriétés :
 - Exemple : <jsp:setProperty name="référence" property="*" />
 - Condition : Les noms des paramètres de requête doivent être identiques aux noms des propriétés

Java Beans et JSP : lecture et écriture propriétés

- Exemple : Soit le fichier « index.html » suivant

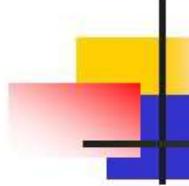
```
<!DOCTYPE html>
<html>
    <head>
        <title>Formulaire</title>
    </head>
    <Form method="GET" action="afficher.jsp">
        CNE:<input type="text" name="cne"/><br>
        Nom:<input type="text" name="nom"/><br>
        Prénom:<input type="text" name="prenom"/><br>
        <input type="submit" value="afficher">
    </Form>
</html>
```



Java Beans et JSP : lecture et écriture propriétés

- Bean Etudiant

```
package toto;
public class Etudiant {
    private String cne;
    private String nom;
    private String prenom;
    public Etudiant() { }
    public String getCne() { return cne; }
    public void setCne(String cne) { this.cne = cne; }
    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; }
}
```



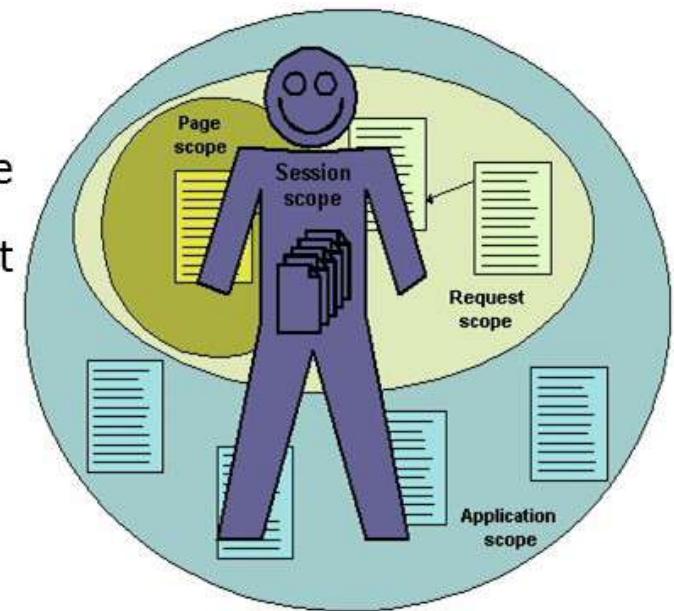
Java Beans et JSP : lecture et écriture propriétés

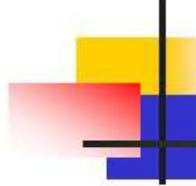
- Fichier « afficher.jsp »

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<jsp:useBean id="etudiant" class="toto.Etudiant"/>
<jsp:setProperty name="etudiant" property="*" />
<!DOCTYPE html>
<html>
    <head>
        <title>afficher.jsp</title>
    </head>
    <body>
        CNE étudiant : <jsp:getProperty name="etudiant" property="cne"/><br>
        Nom étudiant : <jsp:getProperty name="etudiant" property="nom"/><br>
        Prénom étudiant : <jsp:getProperty name="etudiant" property="prenom"/><br>
    </body>
</html>
```

Java Beans et JSP : scope

- Toute variable dans une page JSP a une portée
- Il y a 4 types de portées :
 - Portée **Page** : la variable n'est reconnue qu'au sein de la page
 - Portée **Request** : la variable est reconnue là où la requête est partagée
 - Portée **Session** : la variable est reconnue tant que la session de l'utilisateur est reconnue
 - Portée **Application** : la variable est reconnue dans toute l'application quelque soit la page, quelque soit la requête, quelque soit l'utilisateur.

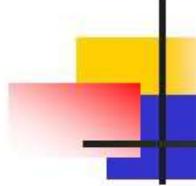




Java Beans et JSP : scope

- **Exemple** : affectation et récupération des valeurs d'un Java Bean
- Soit le java bean suivant :

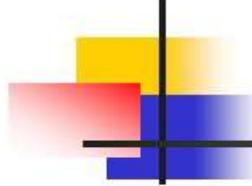
```
Package toto;  
  
public class TestBean{  
  
    private String contenu;  
  
    public String getContenu () { return contenu; }  
  
    public void setContenu (String c) { contenu=c; }  
  
}
```



Java Beans et JSP : scope

- Utilisation du bean avec différentes portées dans une première JSP.

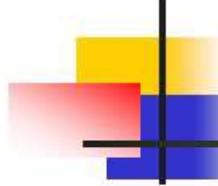
```
<jsp:useBean id="b1" scope="page" class="toto.TestBean"/>
<jsp:useBean id="b2" scope="session" class="toto.TestBean">
<jsp:useBean id="b3" scope="application" class="toto.TestBean"/>
<jsp:setProperty name="b1" property= "contenu" value="page"/>
<jsp:setProperty name="b2" property= "contenu" value="session"/>
<jsp:setProperty name="b3" property= "contenu" value="application"/>
<html>
<head><title> Utilisation du bean </title></head>
<body bgcolor="white">
    Avant<br>
    b1 = <%= b1.getContenu() %><br>
    b2 = <%= b2.getContenu() %><br>
    b3 = <%= b3.getContenu() %><br>
    <form method=GET action="lecture.jsp">
        <p align="center"><input type="submit" name="Submit"></p>
    </form>
</body>
</html>
```



Java Beans et JSP : scope

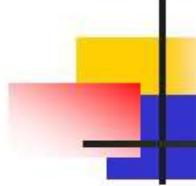
- Récupération du bean avec différentes portées dans une deuxième JSP.

```
<jsp:useBean id="b1" scope="page" class="toto.TestBean"/>
<jsp:useBean id="b2" scope="session" class="toto.TestBean"/>
<jsp:useBean id="b3" scope="application" class="toto.TestBean"/>
<html><head><title> Récupération du bean </title></head>
    <body bgcolor="white">
        Après<br>
        b1 = <jsp:getProperty name="b1" property="contenu"/><br>
        b2 = <jsp:getProperty name="b2" property="contenu"/><br>
        b3 = <jsp:getProperty name="b3" property="contenu"/><br>
    </body>
</html>
```



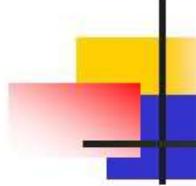
Java Beans et JSP : scope

- Les résultats de ces deux jsp est comme suit :
 - Avant
 - ▶ b1 = page
 - ▶ b2 = session
 - ▶ b3 = application
 - Après
 - ▶ b1 = null
 - ▶ b2 = session
 - ▶ b3 = application



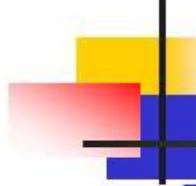
Collaboration de JSP

- Rappel sur la collaboration (voir partie Servlet)
 - partage d'information : un état ou une ressource
 - partage du contrôle : une requête
- Processus identique à la collaboration de Servlet pour le partage d'information et de contrôle
- Partage d'information :
 - Utilisation du contexte pour transmettre des attributs
 - Méthode `getContext(...)`, `setAttribute(...)` et `getAttribute(...)`
- Partage du contrôle : Utilisation des tags action JSP *include* et *forward*



Partage d'information

- Le partage se fait grâce à l'objet implicite `application` qui est de type *ServletContext*
- Exemple : transmettre un simple attribut à tout un contexte
 - Page1.jsp :
Enregistrement dans le contexte d'un attribut
`<% application.setAttribute("attribut1","Bonjour tout le monde"); %>`
 - Page2.jsp :
► `<% out.println(application.getAttribute("attribut1")); %>`



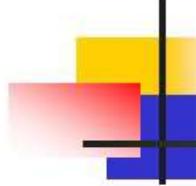
Partage du contrôle

forward:

- Exemple1 : renvoi sans passage de paramètres
`<jsp:forward page="page.html" />`
- Exemple2 : renvoi avec passage de paramètres
`<jsp:forward page="page.html" >`
`<jsp:param name="defaultparam" value="nouvelle" />`
`</jsp:forward>`
- Remarque :
 - ▶ ne pas modifier l'objet response
 - ▶ Ne pas modifier l'objet request après le renvoi

Include :

- Exemple1 : inclusion sans passage de paramètres
`<jsp:include page="page.html" />`
- Exemple2 : inclusion avec passage de paramètres
`<jsp:include page="page.html" >`
`<jsp:param name="defaultparam" value="nouvelle" />`
`</jsp:include>`

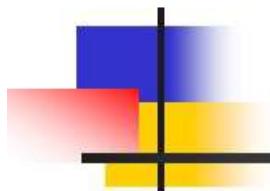


Partage du contrôle

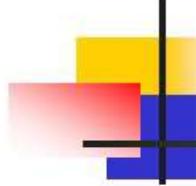
- Remarques :

- Le partage du contrôle et plus précisément l'inclusion et le renvoi par des balises actions ne permettent que le transfert d'attributs de types chaînes de caractères.
- Nécessité d'utiliser *RequestDispatcher* et les objets implicites request et response pour transférer des attributs objets
- Exemple pour une inclusion (même chose pour un renvoi)

```
<% RequestDispatcher dispatch = request.getRequestDispatcher("/fichier.jsp");%>  
<% request.setAttribute("attribut1","bonjour"); %>  
<% dispatch.include(request,response); %>
```

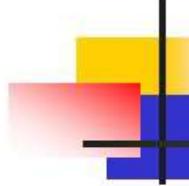


Chapitre 4 : Accès aux bases de données avec JDBC



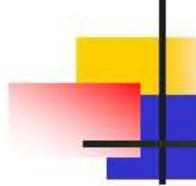
Introduction

- JDBC (Java DataBase Connector) est une API chargée de communiquer avec les bases de données en Java.
- Les classes et interfaces de l'API JDBC figurent dans le package `java.sql` : `import java.sql.*;`
- JDBC peut être utilisé pour accéder à n'importe quelle base de données à partir de:
 - Simple application Java
 - Une servlet
 - Page JSP, ...



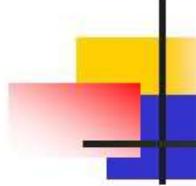
Travail avec une base de données

- JDBC permet de travailler avec les bases de données de la même façon quelque soit leur fournisseur (Oracle, SQL Server, MySQL, PostgreSQL,...).
- Il suffit de télécharger la bibliothèque qui assure la communication entre Java et cette base de donnée.
- Cette bibliothèque s'appelle Driver ou Pilote ou Connecteur.
- Elle figure sur le site du fournisseur du SGBDR utilisé.



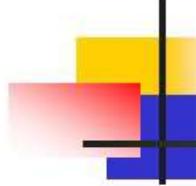
Etapes d'interaction avec une BDD

1. Chargement du pilote
2. Etablissement de la connexion
3. Création des objets encapsulant les requêtes
4. Exécution des requêtes
5. Parcours des résultats dans le cas d'une requête de sélection
6. Fermeture des objets résultats, requêtes et connexion



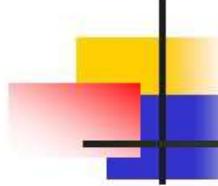
Chargement du pilote

- Pour se connecter à une base de données il faut charger son pilote.
- La documentation de la BDD utilisée fournit le nom de la classe à utiliser.
- Exemple de pilote MYSQL :
 - Le connecteur MySQL pour Java se nomme comme cet exemple : "**mysql-connector-java-5.1.23-bin.jar**"
- Exemple de pilote Derby (intégré à Netbeans pour les tests)
 - Il se nomme Java DB Driver (est composé de 3 jars)
 - A ajouter par le menu (Add Library)



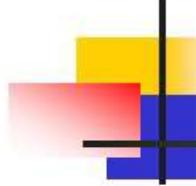
Chargement du pilote

- Le chargement se fait comme suit : `Class.forName("nom_classe_acces_bdd");`
- Exemple :
 - Dans le cas de la Bdd MySQL, ce chargement est comme suit :
`Class.forName(com.mysql.jdbc.Driver)`
 - Dans le cas de la Bdd Derby (notée Java DB), ce chargement est comme suit :
`Class.forName(org.apache.derby.jdbc.ClientDriver)`
- Une fois chargée, la classe JDBC qui se nomme **DriverManager** prend en charge le driver pour communiquer avec la base de donnée.



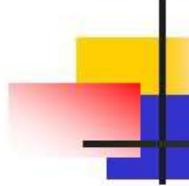
Classes de l'API JDBC

- Les classes et interfaces les plus usuelles sont les suivantes:
 - **DriverManager** (classe): charge et configure le driver de la base de données.
 - **Connection** (interface): réalise la connexion et l'authentification à la base de données.
 - **Statement** (interface): contient la requête SQL et la transmet à la base de données.
 - **PreparedStatement** (interface): représente une requête paramétrée
 - **ResultSet** (interface): représente les résultat d'une requête de sélection.



Etablissement de la connexion

- Pour se connecter à une base de données, il faut disposer d'un objet **Connection** créé grâce au **DriverManager** en lui passant :
 - l'URL de la base à accéder , Le login, Le mot de passe
- Exemples:
 - `String url="jdbc:mysql://localhost/mydb"; // exemple URL BDD MySQL`
 - `String url="jdbc:derby://localhost:1527/EtudiantsDB3"; // exemple URL BDD Derby`
 - `String login="root";`
 - `String password="motdepasse";`
 - **Connection** `con=DriverManager.getConnection(url, login, password);`

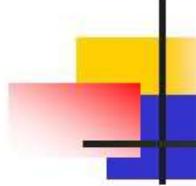


Exécution de requêtes SQL

- L'interface **Statement** permet d'envoyer des requêtes SQL à la base de données.
- Un objet Statement est créé grâce à un objet Connection de la façon suivante :

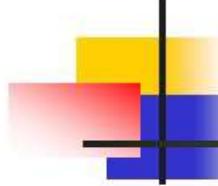
```
Statement st = con.createStatement();
```

- Il possède deux méthodes :
 - **executeUpdate()** : Insertion, suppression, mise à jour.
 - ▶ int n= st.executeUpdate("INSERT INTO Etudiant VALUES (3452,'Taha','Ali')");
 - **executeQuery()** : Selection.
 - ▶ **ResultSet** res= stm.executeQuery("SELECT * FROM Etudiant");



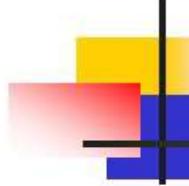
Requêtes avec paramètres

- ▶ L'interface **PreparedStatement** permet d'envoyer des requêtes SQL à la base de données en prenant des paramètres.
- ▶ Ces paramètres sont représentés par des points d'interrogation(?) et doivent être spécifiés avant l'exécution.
- ▶ Exemple :
 - **PreparedStatement** p= con.prepareStatement("select* from Etudiant where cne=? And nom= ?");



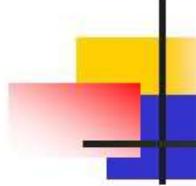
Requêtes avec paramètres

```
p.setInt(1, 3452345);  
p.setString(2, "Alaoui");  
ResultSet resultats = p.executeQuery();
```



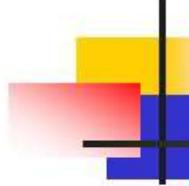
Résultat d'une requête de sélection

- Une requête de sélection retourne un **ResultSet**
- ResultSet est un ensemble d'enregistrements constitués de colonnes qui contiennent les données.
- Les principales méthodes :
 - **next()** : se déplace sur le prochain enregistrement : retourne false si la fin est atteinte. Le curseur pointe initialement juste avant le premier enregistrement.
 - **getInt(int/String)** : retourne le contenu de la colonne dont le numéro (resp. le nom) est passé en paramètre sous forme d'entier.



Résultat d'une requête de sélection

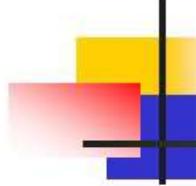
- **getFloat(int/String)** : retourne le contenu de la colonne sous forme de nombre flottant.
- **getDate(int/String)** : retourne le contenu de la colonne sous forme de date.
- **Close()** : ferme le ResultSet



Résultat d'une requête de sélection

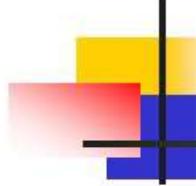
- Exemple :

```
ResultSet res= st.executeQuery("SELECT * FROM Etudiant");  
  
while (res.next()) {  
  
    System.out.println("CNE= "+res.getString(1)+" Nom= " +  
  
        res.getString(2)+" Prénom= "+res.getString(3));  
  
}  
  
res.close();
```



Enoncé TP

- Création de la base de données :
 - Utiliser MYSQL ou DERBY pour créer une base de données ayant une table ETUDIANT composée des champs :
 - CNE
 - NOM
 - PRENOM



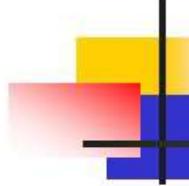
Enoncé TP

- Création de l'application Web :

- L'application web démarre avec une page d'accueil (HTML) avec deux liens hypertextes:
 - ▶ Un lien d'insertion de nouveaux étudiants
 - ▶ Un lien d'affichage des étudiants déjà insérés

Gestion des étudiants :

- [Insérer nouvel Etudiant](#)
- [Afficher liste Etudiants](#)



Enoncé TP

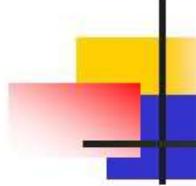
- Le premier lien pointe vers un formulaire HTML qui permet de saisir le CNE, le nom, et le prénom d'un étudiant.
- Ce formulaire est traité par une **Servlet** qui se charge de faire l'insertion dans la base de données.
- Après l'insertion dans la base de données la servlet affiche un message d'insertion réussie et un lien pour revenir à la page d'accueil.

Enregistrement d'un Nouvel Etudiant :

CNE :

Nom :

Prénom :



Enoncé TP

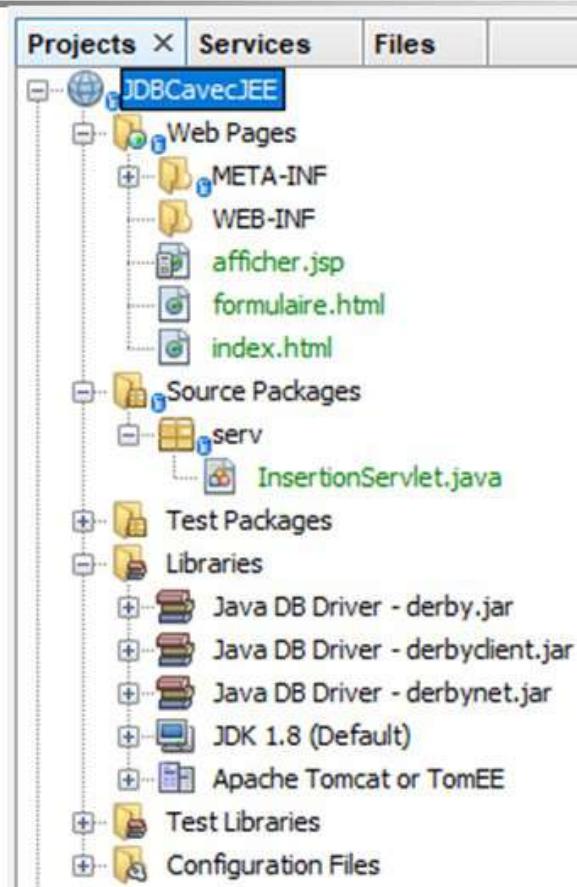
- Le deuxième lien pointe vers une page **JSP** qui se charge d'accéder à la base de données et afficher les données dans un tableau HTML.
- La page JSP à son tour affiche un lien pour revenir à la page d'accueil.

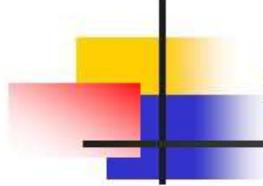
Liste des Etudiants Inscrits :

CNE	NOM	PRENOM
2019a1	Taha	Ali
2019a2	Omari	Omar
2019a5	Taha	Ali
2019a6	Omari	Omar
2019a7	Tahiri	Hassan
2020a	Yousfi	Ali
2020b	Alaoui	Ali
2541s	Aichi	Ayoub
55	Hamdi	Hamid
57	Clayton	Ali
A25689	Borji	Mostafa
A2569548	Juste	Christophe
A25695487	Tahiri	Yassine
B523548	Bachar	Abderrahim

[retour à la page d'accueil](#)

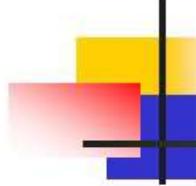
Solution : Structure du projet sous Netbeans





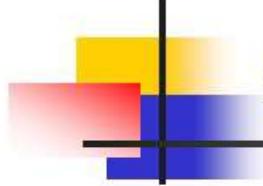
Solution : index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Gestion des étudiants</title>
  </head>
  <body>
    <h2>Gestion des étudiants : </h2>
    <ul type="disc">
      <li><a href="formulaire.html">Insérer nouvel Etudiant</a></li>
      <li><a href="afficher.jsp"> Afficher liste Etudiants</a></li>
    </ul>
  </body>
</html>
```



Solution : formulaire.html

```
<html>
    <head> <title>formulaire</title> </head>
    <body>
        <form method="post" action="traitementFormulaire">
            Enregistrement d'un Nouvel Etudiant : <br>
            CNE : <input type="text" name="cne"><br>
            Nom : <input type="text" name="nom"><br>
            Prénom : <input type="text" name="prenom"><br> <br>
            <input type="submit" value="Envoyer">
            <input type="reset" value="Effacer">
        </form>
    </body>
</html>
```

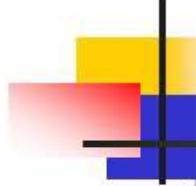


Solution : InsertionServlet.java

```
package serv;
import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
import java.sql.*;
@WebServlet(name = "InsertionServlet", urlPatterns = {"/traitementFormulaire"})
public class InsertionServlet extends HttpServlet {

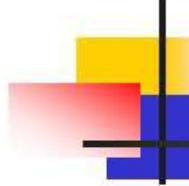
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

```



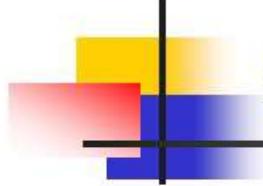
Solution : InsertionServlet.java

```
try { String fcne = request.getParameter("cne");
        String fnom = request.getParameter("nom");
        String fprenom = request.getParameter("prenom");
        String url = "jdbc:derby://localhost:1527/EtudiantsDB3";
        String driver = "org.apache.derby.jdbc.ClientDriver";
        Class.forName(driver);
        Connection con= DriverManager.getConnection(url, "root", "root");
        PreparedStatement stmt = con.prepareStatement("insert into ETUDIANT(CNE ,NOM, PRENOM)
values (?,?,?)");
        stmt.setString(1, fcne);
        stmt.setString(2, fnom);
        stmt.setString(3, fprenom);
        stmt.executeUpdate();
        stmt.close();
        con.close();}
```



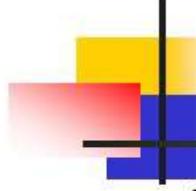
Solution : InsertionServlet.java

```
out.println("insertion nouvel étudiant réussie<br>");  
out.println("<a href=\"index.html\"> retour à la page index </a>");  
}  
catch (ClassNotFoundException | IllegalAccessException | InstantiationException | SQLException e) {  
    out.println("Erreur : " + e.getMessage() + " source : " + e.getStackTrace());  
}  
}  
}  
@Override  
protected void doGet(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException { processRequest(request, response); }  
protected void doPost(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException { processRequest(request, response); }  
}
```



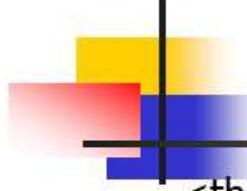
Solution : afficher.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8" import="java.sql.*"%>
<!DOCTYPE html>
<%
    String url = "jdbc:derby://localhost:1527/EtudiantsDB3";
    String driver = "org.apache.derby.jdbc.ClientDriver";
    Class.forName(driver).newInstance();
    Connection con;
    con = DriverManager.getConnection(url, "root", "root");
    PreparedStatement stmt = con.prepareStatement("select * from ETUDIANT");
    ResultSet rs = stmt.executeQuery();
%>
```



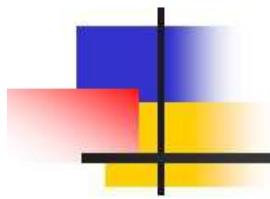
Solution : afficher.jsp

```
<html>
  <head>
    <title>Affichage JSP</title>
  </head>
  <body>
    <h2> Liste des Etudiants Inscrits : </h2>
    <table border="1">
      <thead>
        <tr>
          <th>CNE</th>
          <th>NOM</th>
          <th>PRENOM</th>
        </tr>
      </thead>
```

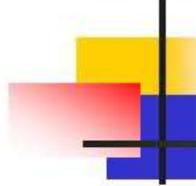


Solution : afficher.jsp

```
<tbody>
    <% while (rs.next()) {%
        <tr> <td><%=rs.getString(1)%></td>
            <td><%=rs.getString(2)%></td>
            <td><%=rs.getString(3)%></td>
        </tr> <%}%
    </tbody>
</table>
<% rs.close();
    stmt.close();
    con.close();
%>
<a href="index.html"> retour à la page d'accueil </a>
</body>
</html>
```



Chapitre 5 : JSTL



Introduction

- JSTL = Java Server Page Standard Tag Library
 - Ensemble de balises prédéfinies organisées en un ensemble de bibliothèques
 - Permet facilement d'accéder et manipuler les données de l'application sans scriptlets.
 - Plus facile à lire car JSTL est basé sur XML, qui est similaire à HTML.
- But : se passer du code Java au sein des pages JSP

Installation

- JSTL fait partie de la spécification JEE
- Il est par conséquent implémenté par tous les conteneurs de servlet
- Tomcat fournit une implémentation téléchargeable à l'adresse :
<http://tomcat.apache.org/download-taglibs.cgi>
- Les bibliothèques téléchargées doivent être placés dans le dossier « WEB-INF/lib »

en sécurisé | tomcat.apache.org/download-taglibs.cgi

Mirrors

You are currently using <https://www-us.apache.org/dist/>. If you encounter problems with this mirror, there are *backup* mirrors (at the end of the mirrors list) that should be available.

Other mirrors: <https://www-eu.apache.org/dist/> ▾ [Change](#)

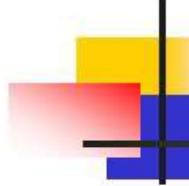
Standard-1.2.5

Source Code Distributions

- [Source README](#)
- [zip \(pgp, sha512\)](#)

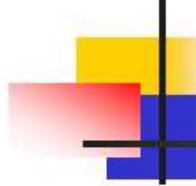
Jar Files

- [Binary README](#)
- Impl:
 - [taglibs-standard-impl-1.2.5.jar \(pgp, sha512\)](#)
- Spec:
 - [taglibs-standard-spec-1.2.5.jar \(pgp, sha512\)](#)
- EL:
 - [taglibs-standard-jstl-1.2.5.jar \(pgp, sha512\)](#)
- Compat:
 - [taglibs-standard-compat-1.2.5.jar \(pgp, sha512\)](#)



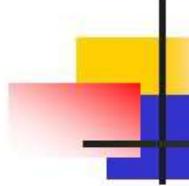
Types de JSTL

- JSTL fournit cinq types de bibliothèques de balises :
 - Core JSTL
 - XML Tag Library
 - Format Tag Library
 - SQL Tag Library
 - Functions Tag Library
- Il propose aussi un langage nommé EL (Expression Language) qui permet de manipuler des objets Java accessibles dans les différentes pages JSP.



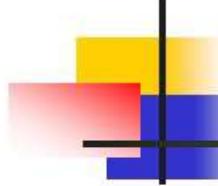
Langage EL : Introduction

- EL= Expression Language
- Est un langage de script qui permet d'accéder d'une façon plus simple aux objets Java accessibles dans les différents contextes de la page JSP.
- La syntaxe de base est \${nomVariable}
- **Exemple :** accès à l'attribut nom d'un objet etudiant situé dans la session
 - Avec Java:<%= session.getAttribute("etudiant").getNom() %>
 - Avec EL: \${sessionScope.etudiant.nom}



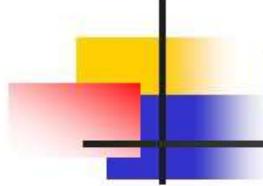
Langage EL : Objets implicites

- **PageScope** : variables couvertes par la portée de la page (correspond à PageContext dans JSP)
- **RequestScope** : variables couvertes par la portée de la requête (HttpServletRequest)
- **SessionScope** : variables couvertes par la portée de la session (HttpSession)
- **ApplicationScope** : variables couvertes par la portée de l'application (ServletContext)
- **Param** : paramètres de la requête HTTP
- **ParamValues** : paramètres de la requête sous forme d'une collection



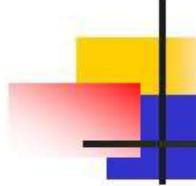
Langage EL : Objets implicites

- **Header** : en-tête de la requête
- **HeaderValues** : en-têtes de la requête sous forme d'une collection
- **InitParam** : Paramètres d'initialisation du contexte
- **Cookie** : valeurs du cookie
- **PageContext** : correspond à l'objet PageContext de la page en cours



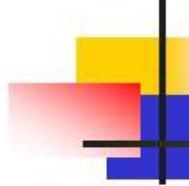
Langage EL : Operateurs de base

- **.** : permet d'obtenir une propriété d'un objet,
 - Exemple \${param.nom}
- **[]** : permet d'obtenir une propriété par son nom ou son indice.
 - Exemple : \${param[" nom "]}, \${row[1]}
- **empty** : Teste si les valeurs de variables sont vides.
 - Exemple : \${empty param.nom}
- **==** ou **eq** : teste l'égalité de deux objets
- **!=** ou **ne** : teste l'inégalité de deux objets



Langage EL : Operateurs de base

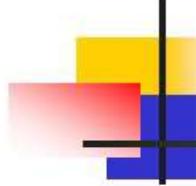
- < ou **lt** : test strictement inférieur
- > ou **gt** : test strictement supérieur
- <= ou **le** : test inférieur ou égal
- >= ou **ge** : test supérieur ou égal
- + : Addition , - : Soustraction, * : Multiplication, / ou **div** : Division
- % ou **mod** : Modulo
- && ou **and** : conjonction
- || ou **or** : disjonction
- ! ou **not** : Négation d'une valeur



Langage EL : variables locales

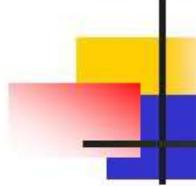
- Le langage EL ne permet pas l'accès aux variables locales.
- Pour pouvoir les utiliser, il faut obligatoirement en créer une copie dans une des portées particulières : page, request, session ou application.
- Exemple :

```
<% int x= 2019;
    int y=2020;
    pageContext.setAttribute("y", new Integer(y));
%>
Valeur de x = <c:out value="${x}" /><BR/>
Valeur de y = <c:out value="${y}" /><BR/>
```
- Exécution :
 - Valeur de x = // x ne sera pas affiché car c'est une variable locale
 - Valeur de y = 2020



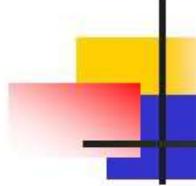
Core JSTL: introduction

- Propose un ensemble de tags pour l'itération, le traitement conditionnel et le langage d'expression.
- Au niveau des pages JSP, la déclaration se fait comme suit:
 - <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
- Ces tags sont répartis en 3 catégories :
 - Pour le langage EL : **set, out, remove, catch**
 - Pour les conditions et itérations : **if, choose, forEach, forTokens**
 - Pour la gestion des URL : **import, url, redirect**



Core JSTL : balise <c:set>

- Permet de stocker une variable dans une portée particulière (page, requête, session ou application).
- Dispose des attributs suivants:
 - **var** : nom de la variable qui va stocker la valeur
 - **value** : valeur à stocker
 - **scope** : portée de la variable qui va stocker la valeur
 - **target** : nom de la variable contenant un bean dont la propriété doit être modifiée
 - **property** : nom de la propriété à modifier



Core JSTL : balise <c:set>

- Exemples :

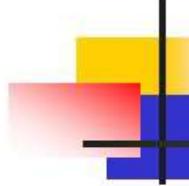
- <c:set var="a" value="valeur1" scope="page" />
- <c:set var="b" value="valeur2" scope="request" />
- <c:set var="c" value="valeur3" scope="session" />
- <c:set var="d" value="valeur4" scope="application" />

- Remarque 1 : la valeur peut être déterminée dynamiquement :

- <c:set var="e" value="\${param.id}" scope="page" />

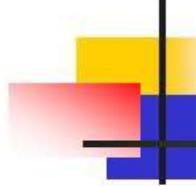
- Remarque 2 : La valeur de la variable peut être précisée dans le corps de la balise

- <c:set var="maVariable" scope="page">Valeur de ma variable</c:set>



Core JSTL : balise <c:out>

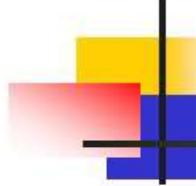
- Est utilisée pour afficher les valeurs contenues dans variables ou le résultat d'une expression implicite de la même façon que <%=...%>
- Attributs :
 - **Value** : valeur à afficher (obligatoire)
 - **Default** : définir une valeur par défaut si la valeur est null
 - **escapeXml** : booléen qui permet de convertir les caractères spéciaux à leurs codes correspondants
- Elle offre en plus la possibilité d'accès aux propriétés grâce au ":"



Core JSTL : balise <c:out>

- Exemples :

- <c:out value='\${pageScope.maVariable1}' />
- <c:out value='\${requestScope.maVariable2}' />
- <c:out value='\${sessionScope.maVariable3}' />
- <c:out value='\${applicationScope.maVariable4}' />



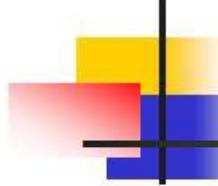
Core JSTL : balise <c:out>

- Remarque :

- Si la protée de la variable n'est pas précisée la variable est recherchée prioritairement dans la page, puis la requête, puis la session et enfin l'application.
- L'attribut default définit une valeur par défaut si le résultat de l'évaluation de la valeur est null.
- Si la valeur est null et que l'attribut default est absent alors c'est une chaîne vide qui est renvoyée

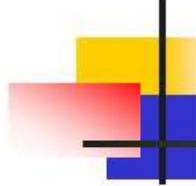
- Exemple :

- `<c:out value="${personne.nom}" default="Inconnu" />`



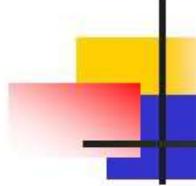
Core JSTL : balise <c:out>

- **Remarque** : exemple de génération de code dans un formulaire sans passer par les scriplets.
 - `<input type="text" name="nom" value=">" />`



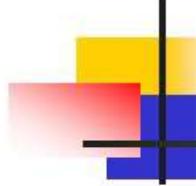
Core JSTL : balise <c:remove>

- Permet de supprimer une variable d'une portée particulière.
- attributs :
 - Var : nom de la variable à supprimer (obligatoire)
 - Scope : portée de la variable
- Exemples :
 - <c:remove var="maVariable1" scope="page" />
 - <c:remove var="maVariable2" scope="request" />
 - <c:remove var="maVariable3" scope="session" />
 - <c:remove var="maVariable4" scope="application" />



Core JSTL : balise <c:catch>

- Permet de capturer des exceptions qui sont levées lors de l'exécution du code inclus dans son corps.
- Attributs :
 - Var : nom d'une variable qui va contenir des informations sur l'anomalie

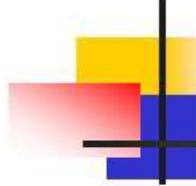


Core JSTL : balise <c:catch>

- Exemple :

```
<c:set var="valeur" value="abc" />
<c:catch var="erreur">
    <fmt:parseNumber var="valeurInt" value="${valeur}" />
</c:catch>
<c:if test="${not empty erreur}">
    la valeur n'est pas numerique
</c:if>
```

- Resultat : la valeur n'est pas numerique



Core JSTL : balise <c:catch>

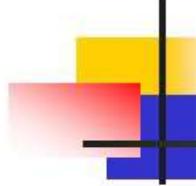
- Remarque :

- L'objet désigné par l'attribut var du tag catch possède une propriété message qui contient le message d'erreur

- Exemple :

```
<c:set var="valeur" value="abc" />
<c:catch var="erreur">
    <fmt:parseNumber var="valeurInt" value="${valeur}" />
</c:catch>
<c:if test="${not empty erreur}">
    <c:out value="${erreur.message}" />
</c:if>
```

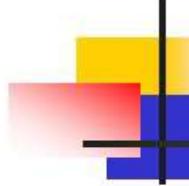
- Résultat : In <parseNumber>, value attribute can not be parsed: "abc"



Core JSTL : balise <c:catch>

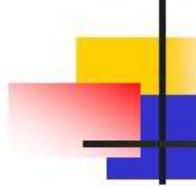
- Remarque :

- Le problème avec cette balise est qu'il n'est pas possible de savoir quelle exception a été levée.



Core JSTL : balise <c:if>

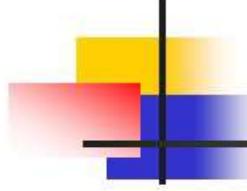
- Évalue une expression et affiche le contenu de son corps uniquement si l'expression est évaluée à Vrai.
- Attributs :
 - test : condition à évaluer
 - var : nom de la variable qui contiendra le résultat de l'évaluation
 - scope : portée de la variable qui contiendra le résultat
- Exemples:
 - <c:if test="\${empty personne.nom}">Inconnu</c:if>
 - <c:if test="\${empty personne.nom}" var="resultat" />
 <%-- le résultat est stocké dans une variable--%>



Core JSTL : balise <c:choose>

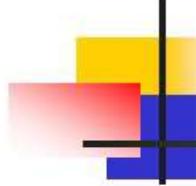
- Permet d'effectuer un choix parmi plusieurs mutuellement exclusifs
- Ne possède pas d'attributs mais deux balises filles <c:when> et <c:otherwise>
- Exemple :

```
<c:choose>
    <c:when test="${personne.civilite == 'Mr'}">
        Bonjour Monsieur
    </c:when>
    <c:when test="${personne.civilite == 'Mme'}">
        Bonjour Madame
    </c:when>
```



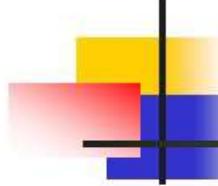
Core JSTL : balise <c:choose>

```
<c:when test="${personne.civilite == 'Mlle'}">  
    Bonjour Mademoiselle  
</c:when>  
<c:otherwise>  
    Bonjour  
</c:otherwise>  
</c:choose>
```



Core JSTL : balises <c:forEach> et <c:forTokens>

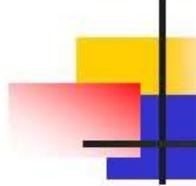
- Constituent une alternative au boucles Java dans les scriptlets.
- La balise <c: forEach> itère sur une collection d'objets.
- La balise <c: forTokens> est utilisée pour diviser une chaîne en jetons et les parcourir.
- Attributs communs :
 - **var** : nom de la variable qui contient l'élément en cours de traitement
 - **items** : collection à traiter
 - **varStatus** : nom d'une variable qui va contenir des informations sur l'itération en cours de traitement



Core JSTL : balises <c:forEach> et <c:forTokens>

- **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)
- Attribut en plus pour <c:forTokens> :
 - **delims** : Caractères à utiliser comme délimiteurs ou séparateurs.
- Exemple pour <c:forEach>

```
<c:forEach var = "i" begin = "1" end = "5">
    élément <c:out value = "${i}" /><p>
</c:forEach>
```



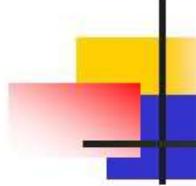
Core JSTL : balises <c:forEach> et <c:forTokens>

- Exemple pour <c: forTokens> :

```
<c:forTokens items = "Ali,Omar,Taha" delims = "," var = "name">
    <c:out value = "${name}"/><p>
</c:forTokens>
```

- **Remarque** : varStatus permet de définir une variable qui va contenir des informations sur l'itération en cours d'exécution. Cette variable possède plusieurs propriétés :

- index indique le numéro de l'occurrence dans l'ensemble de la collection
- count indique le numéro de l'itération en cours (en commençant par 1)
- first booléen qui indique si c'est la première itération
- last booléen qui indique si c'est la dernière itération



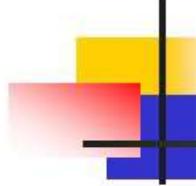
Core JSTL : balises <c:forEach> et <c:forTokens>

- Exemple :

```
<c:forEach begin="1" end="12" var="i" step="3" varStatus="vs">
    index = <c:out value="${vs.index}" /> :
    count = <c:out value="${vs.count}" /> :
    value = <c:out value="${i}" />
    <c:if test="${vs.first}"> : Premier element </c:if>
    <c:if test="${vs.last}"> : Dernier element </c:if> <br>
</c:forEach>
```

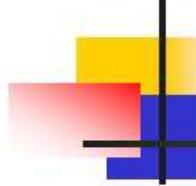
- Résultat :

- index = 1 : count = 1 : value = 1 : Premier element
- index = 4 : count = 2 : value = 4
- index = 7 : count = 3 : value = 7
- index = 10 : count = 4 : value = 10 : Dernier element



Core JSTL : balise <c:import>

- Permet d'inclure une ressource identifiée par une URL tout comme l'action `<jsp :include>` mais a l'avantage de ne pas être limité au contexte de l'application web.
- Attributs :
 - **url** : URL de la ressource (relative à l'application web ou absolue)
 - **var** : nom de la variable qui va stocker le contenu de la ressource sous la forme d'une chaîne de caractères
 - **scope** : portée de la variable qui va stocker le contenu de la ressource
 - **context** : contexte de l'application Web qui contient la ressource (si la ressource n'est pas l'application web courante)

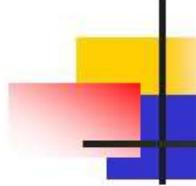


Core JSTL : balise <c:import>

- **charEncoding** : jeu de caractères utilisé par la ressource
 - **varReader** : nom de la variable qui va stocker le contenu de la ressource sous la forme d'un objet de type java.io.Reader
- Exemples :
- Import direct du contenu de la ressource :

```
<c:import url="/message.txt" /><br>
```
 - Import du contenu de la ressource dans une variable :

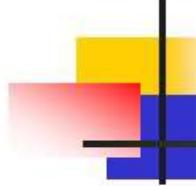
```
<c:import url="/message.txt" var="message" />
<c:out value="${message}" /><BR/>
```



Core JSTL : balise <c:redirect>

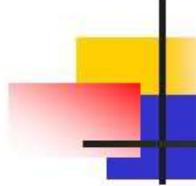
- Permet de faire une redirection vers une nouvelle URL.
- Les paramètres peuvent être fournis grâce à un ou plusieurs tags fils param.
- Exemple :

```
<c:redirect url="liste.jsp">  
    <c:param name="id" value="123"/>  
</c:redirect>
```



Core JSTL : balise <c:url>

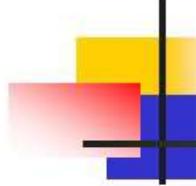
- Formate une URL en chaîne et la stocke dans une variable.
- C'est une alternative à l'appel de la méthode **response.encodeURL()**
- Attributs :
 - value : base de l'URL (obligatoire)
 - var : nom de la variable qui va stocker l'URL
 - scope : portée de la variable qui va stocker l'URL
 - context : contexte.
- Admet **param** comme balise fille, cette balise dispose des attributs :
 - **name** : nom du paramètre
 - **value** : valeur du paramètre



Core JSTL : balise <c:url>

- Exemple :

- `<a href = "<c:url value = "/jsp/index.htm"/>">TEST`

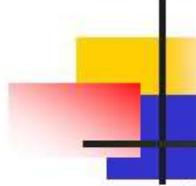


Core JSTL : Exercices

- 1) Créer une page jsp qui affiche dans un tableau HTML la table de multiplication du nombre 3 .
- 2) Créer une page web jsp permettant d'afficher les nombres pairs *compris* entre 1 et 40 en utilisant la JSTL.
- 3) Développer une page jsp qui prend en paramètre un nombre et calcule s'il est premier ou non

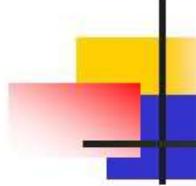
Table de multiplication du nombre 3

3 x 1	=	3
3 x 2	=	6
3 x 3	=	9
3 x 4	=	12
3 x 5	=	15
3 x 6	=	18
3 x 7	=	21
3 x 8	=	24
3 x 9	=	27
3 x 10	=	30



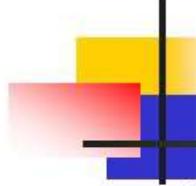
Solution exercice 1

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Table de multiplication du nombre 3</title>
    </head>
    <body>
        <TABLE border="1">
            <CAPTION> Table de multiplication du nombre 3 </CAPTION>
            <TBODY>
```



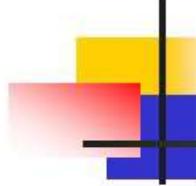
Solution exercice 1 (suite)

```
<c:forEach var = "i" begin = "1" end = "10">
    <TR><TD> 3 x <c:out value = "${i}" /> </TD> <TD> = </TD>
        <TD> <c:out value = "${3*i}" /> </TD>
    </TR>
</c:forEach>
</TBODY>
</TABLE>
</body>
</html>
```



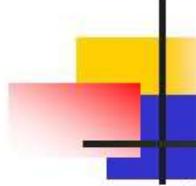
Solution exercice 2

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
    <head>
        <title>Les nombres pairs compris entre 1 et 40</title>
    </head>
    <body>
        <c:forEach var="i" begin="1" end="40" step="1">
            <c:if test="\$\{i%2 == 0\}">
                <c:out value="\$\{i\}" /><br />
            </c:if>
        </c:forEach>
    </body>
</html>
```



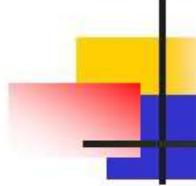
Solution exercice 3

```
<!DOCTYPE html>
<html>
    <head>
        <title>Fichier index.html</title>
    </head>
    <body>
        <form method="post" action="traitement.jsp">
            <label>Entrer un nombre : </label>
            <input type="text" name="nombre" />
            <input type="submit" value="valider" />
        </form>
    </body>
</html>
```



Solution exercice 3 (suite)

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Fichier traitement.jsp</title>
    </head>
    <body>
        <c:set var="nbr" value="${param.nombre}" />
        <c:set var="estPremier" value ="${true}" />
        <c:forEach var="i" begin="${2}" end="${nbr/2}">
            <c:if test="${nbr % i == 0 && nbr != i}">
                <c:set var="estPremier" value ="${false}" />
            </c:if>
        </c:forEach>
```



Solution exercice 3 (suite)

```
<!-- Affichage du résultat-->
<c:choose>
    <c:when test="${estPremier == true}">
        <c:out value="${nbr}"/> est premier <br/>
    </c:when>
    <c:when test="${estPremier == false}">
        <c:out value="${nbr}"/> n'est pas premier <br/>
    </c:when>
</c:choose>
</body>
</html>
```