Servlets

Mohammadi Zakaria Zakaria.mohammadi@iga.ac.ma

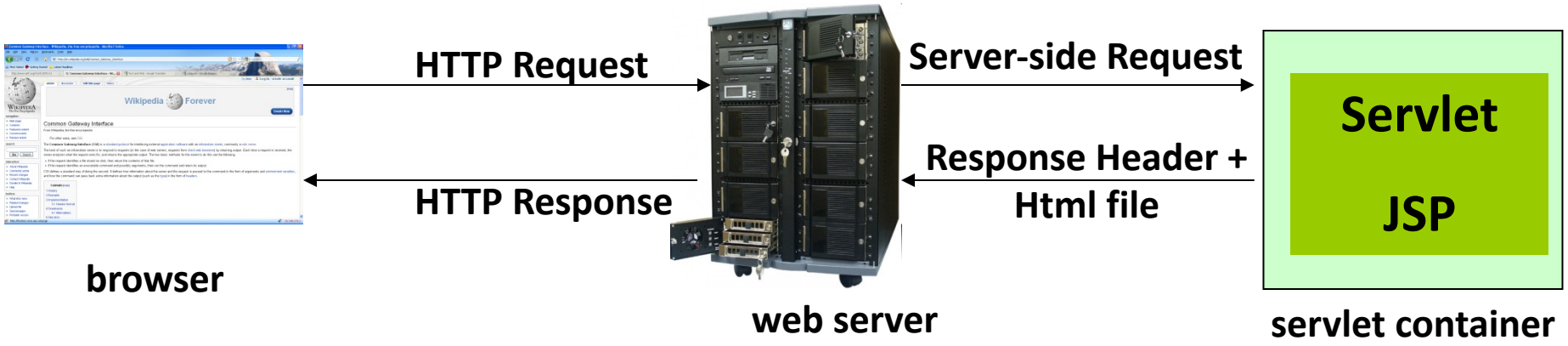
Servlet JAVA

Introduction

- Une Servlet est un programme d'application Web écrit en Java, qui s'exécute sur le serveur en interaction avec le client distant et délivre des données en HTML ou autre format.
- Le client peut accéder au Servlet avec le navigateur par une URL. Il fonctionne dans un conteneur de Servlets.

Introduction

- Une Servlet est un programme JAVA dont la sortie est une page HTML. C'est une *Server-Side* technologie.



Hello World … par une Servlet

```
import javax.servlet.http.*;
import javax.servlet.*;
public class HelloServlet extends HttpServlet {
        public void doGet
                (HttpServletRequest req, HttpServletResponse res)
                                        throws ServletException, IOException {
                    PrintWriter out = res.getWriter();
                    out.println("<html>");
                    // …
                    out.println("Hello, world !");
                    // …
                    out.println("<html>");
                    out.close();
        }
}
```

L'API servlet

- Les Servlets sont conçues pour agir selon un modèle de requête/réponse. Tous les protocoles utilisant ce modèle peuvent être utilisés: http, ftp, etc …
- L'API Servlets est une extension du jdk de base, et en tant que telle elle est regroupée dans des packages javax.*;
- L'API Servlet regroupe des classes dans les 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.

Servlet

- Une Servlet HTTP doit hériter de ***HttpServlet***.
- Elle propose ainsi des méthodes Java nécessaires au traitement des requêtes et réponses HTTP :
  - ***doGet()*** pour gérer la méthode GET ;
  - ***doPost()*** pour gérer la méthode POST ;
  - ***doHead()*** récupérer les informations sur un document (Type, Capacité, Date de dernière modification etc…).
- Une Servlet doit implémenter au moins une des méthodes ***doXXX()***, afin de traiter une requête entrante.

Déclaration de Servlet

- La mise en place d'une Servlet se déroule en deux étapes : nous devons d'abord déclarer la Servlet, puis lui faire correspondre une URL.

```
<servlet>
    <servlet-name>Test</servlet-name>
    <servlet-class>com.tpServlet.Test</servlet-class>
    …
</servlet>
```

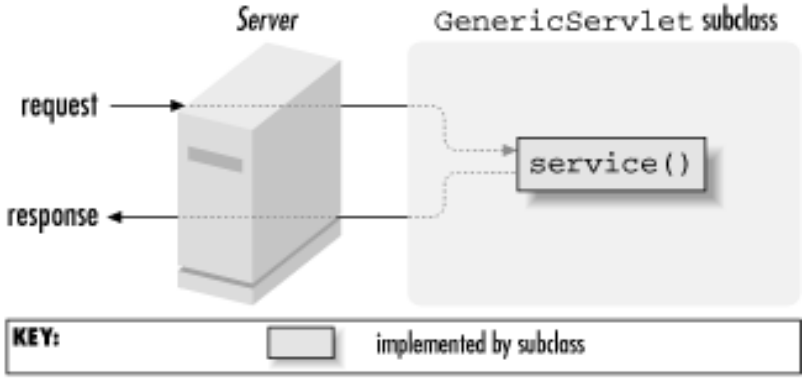
Déclaration de Servlet

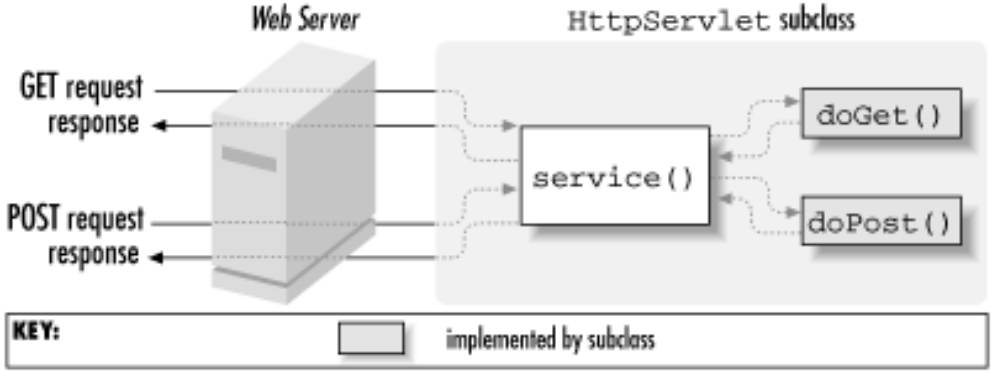
- La mise en place d'une Servlet se déroule en deux étapes : nous devons d'abord déclarer la Servlet, puis lui faire correspondre une URL.

```
<servlet>
    <servlet-name>Test</servlet-name>
    <servlet-class>com.tpServlet.Test</servlet-class>
    …
    <servlet-mapping>
        <servlet-name>Test</servlet-name>
        <url-pattern>/testServlet</url-pattern>
    </servlet-mapping>
</servlet>
```

Comment ca marche ?

- Pas de méthode *main()* mais une méthode *service()*
- Version générique: Version http





HttpServlet

- Package ***javax.servlet.http***
  - ***HttpServlet*** gestion des Servlet pour recevoir des requêtes et envoyer des réponses
  - ***HttpServletRequest*** interface des requêtes
  - ***HttpServletResponse*** interface des réponses
  - ***HttpSession*** gestion de la session
  - ***ServletContext*** gestion du contexte de l'application
  - ***RequestDispatcher*** lancement de Servlet (forward())

HttpServlet

- protected void **doGet**

  (*HttpServletRequest* req, *HttpServletResponse* resp)

  throws *ServletException*, *IOException* {…}

- protected void **doPost**

  (*HttpServletRequest* req, *HttpServletResponse* resp)

  throws *ServletException*, *IOException* {…}

- void **init**() { … }
- void **destroy**() { … }

HttpServlet

- protected void **doGet**

  (*HttpServletRequest* req, *HttpServletResponse* resp)

  throws *ServletException*, *IOException* {…}

- protected void **doPost**

  (*HttpServletRequest* req, *HttpServletResponse* resp)

  throws *ServletException*, *IOException* {…}

- void **init**(){ … }
- void **destroy**() { … }

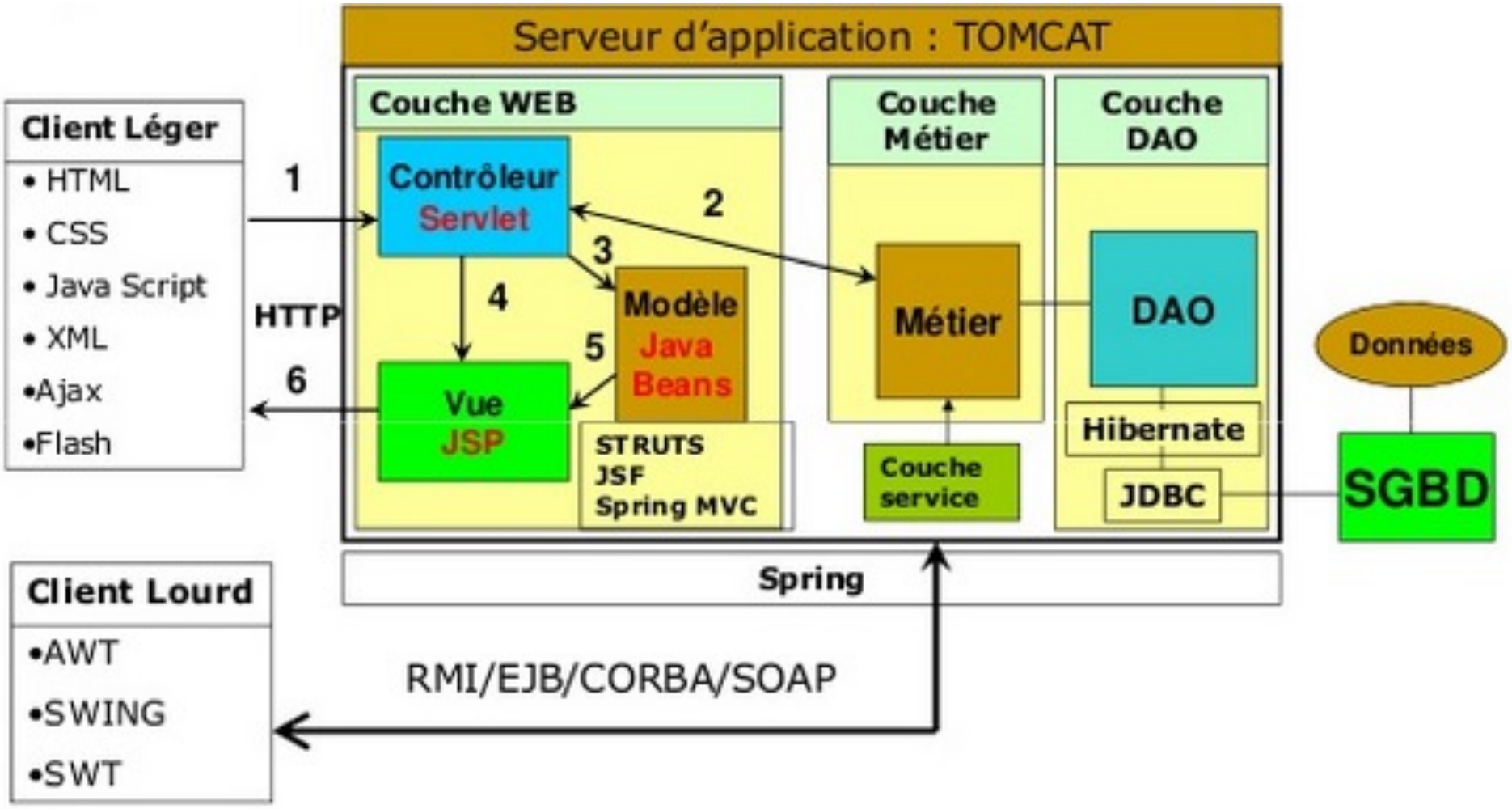
javax.Servlet.http

Javax.servlet	Nom	Rôle
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 reponse de la servlet selon le protocole http
	HttpSession	Définit un objet qui représente une session
Les classes	Cookie	Classe représentant un cookie (ensemble de données sauvegardées par le brower 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

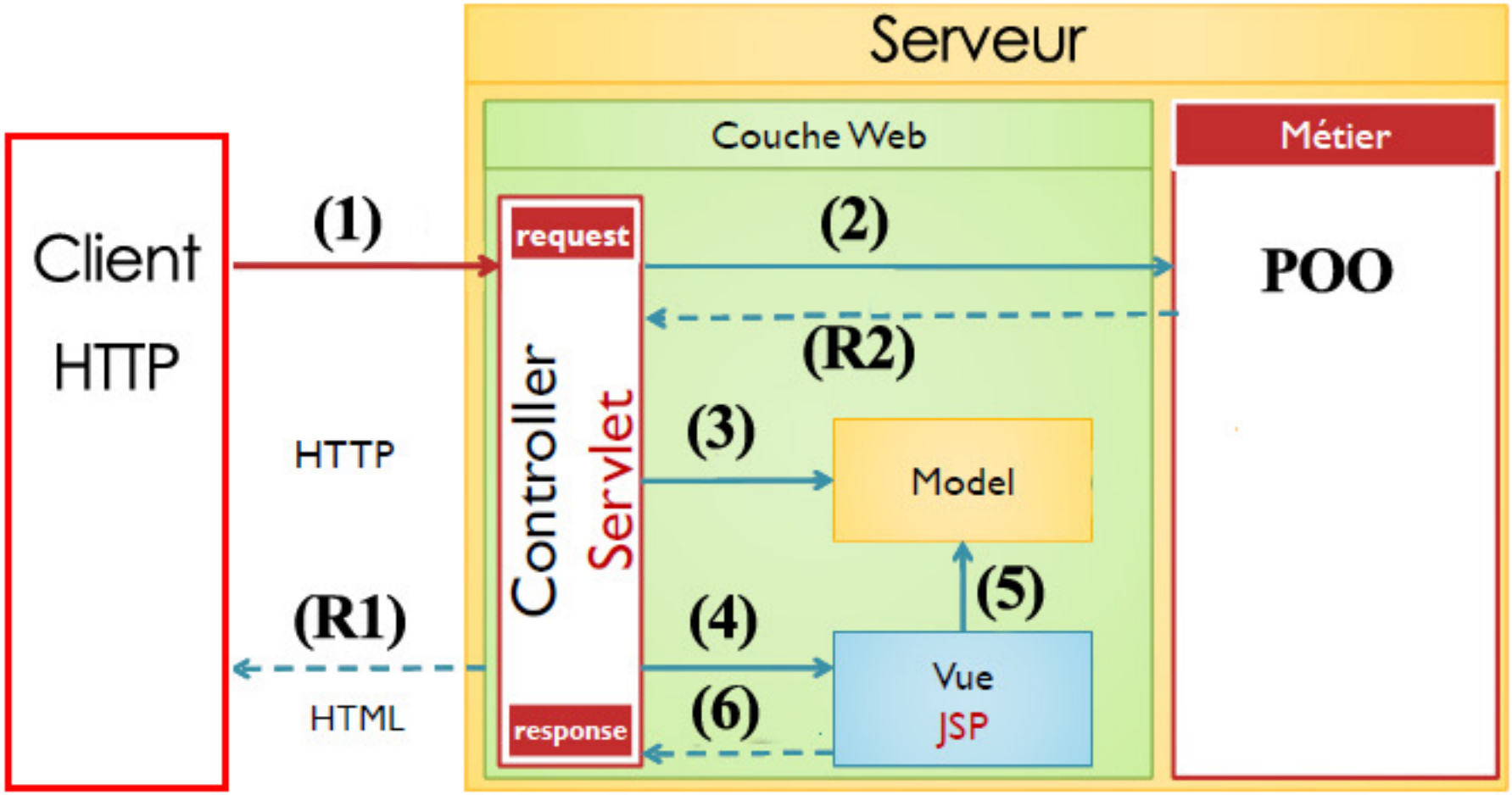
httpServletRequest

- ***String getParameter(nom)*** valeur d'un paramètre de formulaire.
- ***String[] getParamterValues(nom)*** pour plusieurs valeurs.
- ***Enumeration getParameterNames()*** nom de tous les paramètres du formulaire.
- ***void setAttribute(nom, objet)*** pour définir un attribut.
- ***Object getAttribute(nom)*** pour récupérer un attribut.
- ***Enumeration getAttributeNames().***
- ***void removeAttribute().***

Architecture WEB J2EE

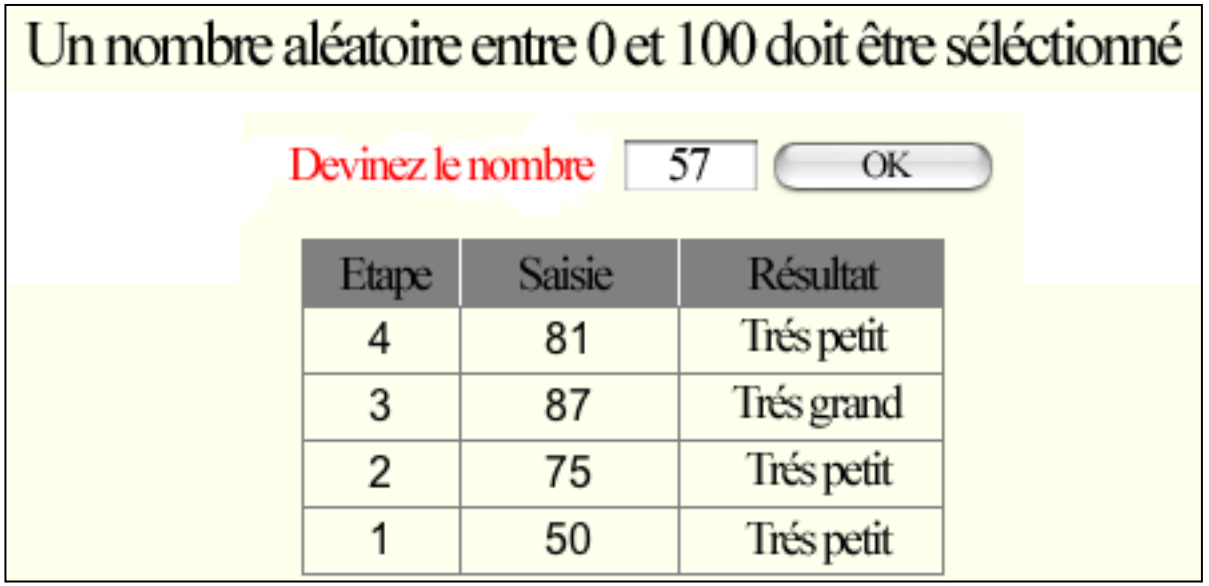


Architecture WEB J2EE



Jsp / Servlet

On désire implémenter une page jsp '***JeuHasard***', et une Servlet '***ControlleurJeu***'. Le principe est de générer aléatoirement un nombre aléatoire (*entre 0 et 100*), et demander à chaque fois à l'utilisateur de saisir un nombre, jusqu'à ce que l'utilisateur tombe sur la valeur exacte. À chaque saisie erronée, on indiquera à l'utilisateur si la valeur saisie est plus grande/petite que celle attendue.



Etape	Saisie	Résultat
4	81	Trés petit
3	87	Trés grand
2	75	Trés petit
1	50	Trés petit

Application Facture

On désire implémenter une application permettant de remplir une facture par une liste d'articles, leurs quantités ainsi que leurs prix. L'utilisateur doit, à travers un formulaire, saisir le nom de l'article, la quantité requise, ainsi que le prix unitaire de l'article concerné. Une fois il valide par le clic sur le bouton ***'Ajouter'***, les différentes informations sur l'article doivent être ajoutées dans le tableau facture, en affichant le prix que doit payer le client. E utilisant le paradigme MVC, essayez d'implémenter cette application web.

Nom de l'article **Quantité** **Prix unitaire**

Ajouter

Détails de la facture :

Article	*Quantité*	*Prix Unitaire*
Crayon	*20*	*2*
Ramette	*15*	*30*

Total à payer : 490 drhm

Classes Listener

- Une classe peut implémenter ***ServletContextListener***, dont la méthode ***contextInitialized*** sera appelée au déploiement de l'application (remplir une bd avec des données pour le test, créer un compte utilisateur pour l'admin,…)
- Un ***SessionListener*** peut être utilisé pour associer un écouteur à la session (comme pour récupérer le nombre de session ouvertes sur l'application coté serveur, ou pour effectuer des opérations de nettoyage lorsqu'une session expire).
- L'interface ***ServletRequestListener*** permet de suivre la création et la destruction des requêtes HTTP, et peut être utile pour effectuer des opérations de nettoyage lorsqu'une requête est terminée.

Classes Listener

- Il faut premièrement déclarer ce listener dans le fichier de déploiement ***web.xml***:

```
<listener>
    <description>
        Classe alimentant la BD au déploiement
    </description>
    <listener-class>
        Outils.InitialisationBD
    </listener-class>
<listener>
```

Classes Listener

- Il faut premièrement déclarer ce listener dans le fichier de déploiement ***web.xml***:

```
<listener>
    <description>
        Classe alimentant la BD au déploiement
    </description>
    <listener-class>
        Outils.InitialisationBD
    </listener-class>
<listener>
```

- Une autre approche consiste à utiliser l'annotation ***@WebListener***.

Classes Listener

- Ensuite, il faut implémenter cette classe *listener* déclarée dans le fichier *web.xml*:

```
@WebListener
public class InitialisationBD implements ServletContextListener{
    @Override
    public void contextInitialized( ServletContextEvent e){
        // instructions d'initialisation
        …
    }
    @Override
    public void contextDestroyed( ServletContextEvent e){
        // instructions de destruction
        …
    }
}
```

ServletRequestListener

- Pour un *listener* qui calcule le nombre de requêtes qu'a reçu l'application, on peut utiliser un ***ServletRequestListener*** :

```
@WebListener
public class RequestCounter implements ServletRequestListener{
        private int requestCounter = 0;
        public void requestInitialized(ServletRequestEvent sre) {
            ServletContext context = sre.getServletContext();
            requestCounter++;
            context.setAttribute("requestCounter", requestCounter);
        }
        public void requestDestroyed(ServletRequestEvent sre) {
            ServletContext context = sre.getServletContext();
            requestCounter--;
            context.setAttribute("requestCounter", requestCounter);
        }
}
```

Paramètres d'initialisation

- Le fichier de déploiement peut être utilisé pour instancier et déclarer des paramètres d'initialisation.

```
<init-param>
    <param-name> Taux_conversion </param-name>
    <param-value> 25.58 </param-value>
</init-param>
<init-param>
    <param-name> Coefficient_R </param-name>
    <param-value> 13.2 </param-value>
</init-param>
…
```

Paramètres d'initialisation

- Pour récupérer ces variables, il suffit de les chercher depuis le contexte de l'application

```
public void init(ServletConfig config)
                throws ServletException {
    String taux = config.getInitParameter("Taux_conversion");
    // …
}
```

Servlet 3.0

- Le modèle 3.0 a apporté de nombreuses annotations, évitant de surcharger le contenu du fichier web.xml.

```
@WebServlet(asyncSupported = false,
        name="S1" ,
        urlPattern={"/serv1"},
        initParams ={ @WebInitParam(name="x",value="1") ,
                      @WebInitParam(name="y",value="AB") })
        public class Servlet1 extends HttpServlet{
                ...
        }
```

Jsp / Servlet avec annotations

Généralement, les applications web ne sont accessibles que pour les utilisateurs qui sont connues, et dont les informations sont stockés et accessibles depuis une BD des utilisateurs.

Le but de cet atelier, est d'implémenter les pages nécessaires pour assurer cette activité d'authentification: sur la page ***'acceuil'***, l'utilisateur doit renseigner son login et Password, et cliquer par la suite sur le bouton 'Se connecter'. Si les informations sont correctes, on le redirige vers la page '***Bienvenue***' avec un message de bienvenu suivi de son login. Sinon, on renvoi l'utilisateur vers une page contenant un message d'erreur, avec un lien lui permettant de réessayer encore une fois.

Jsp / Servlet avec annotations