Introduction à la plateforme JEE

Dr. Abdessamad Belangour

Plan du cours

- Chapitre 1 : Survol de la plateforme JEE
- Chapitre 2 : Servlets
- Chapitre 3 : JSP
- Chapitre 4 : JDBC
- Chapitre 5 : JPA (exposé)
- Chapitre 6 : JSTL
- Chapitre 7 : EJB 3.2 (exposé)

Chapitre 1 : Survol de la plateforme JEE

Introduction

- JEE est une plateforme de développement informatique pour le développement d'applications Web côté serveur (pour le cloud)
- Elle a été développée initialement par la société SUN comme l'une des trois éditions de Java (Java Standard Edition, Java Entreprise Edition, Java Micro Edition)
- En 2009 la société Oracle a acheté la société SUN
- En 2017 Oracle cède Java EE à la fondation Eclipse :
  - Java EE devient Jakarta EE à partir de la version 9
  - Oracle NetBeans devient Apache NetBeans

Plateforme Jakarta EE

- Jakarta EE est proposée sous forme d'une norme comprenant :
  - Les spécifications du serveur d'application (environnement d'exécution).
  - Des services (un ensemble d'APIs) qui sont des extensions Java indépendantes permettant d'offrir en standard un certain nombre de fonctionnalités.

Les API de JEE

- Les API de Jakarta EE 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

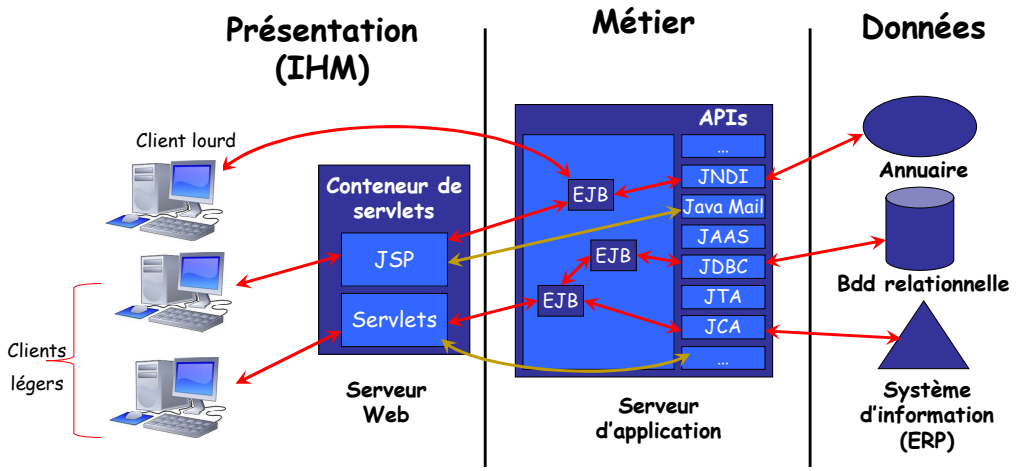
Les API de JEE

- 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.
- **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 SOFTWARE FOUNDATION
WildFly
JOnAS
APACHE GERONIMO
JBoss by Red Hat
IBM WebSphere AS
GlassFish
resin
ORACLE WEBLOGIC SERVER

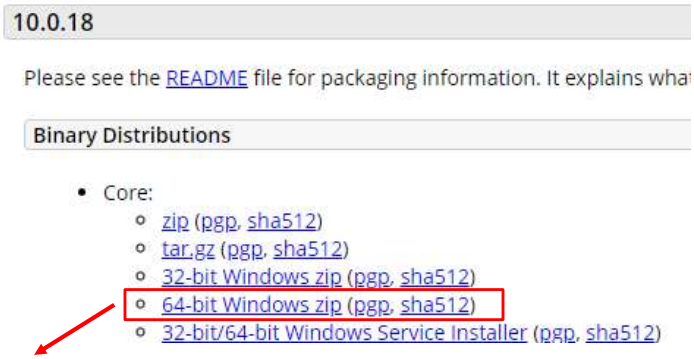
Remarque : Nous allons travailler avec Tomcat dans le reste de ce cours

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

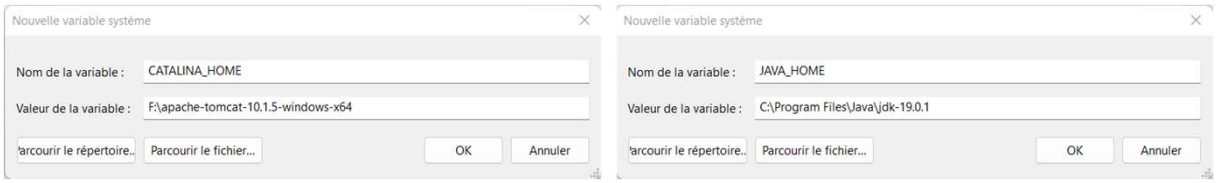
- Disponible pour téléchargement à l'adresse suivante :
  https://tomcat.apache.org/



- Télécharger la version zip
  (non installable)

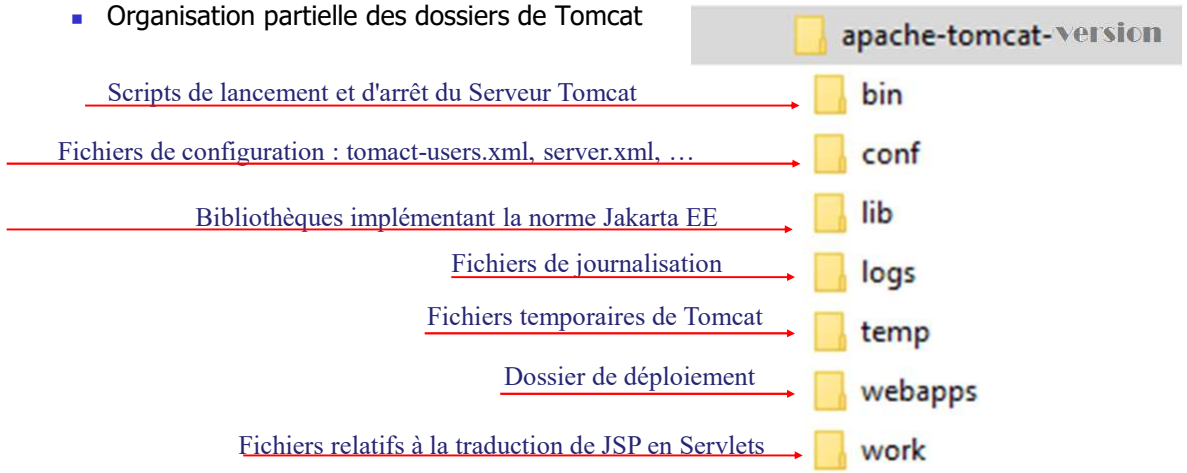
Jakarta Tomcat : version zip

- 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)
- Exemple :



Jakarta Tomcat : Hiérarchie des dossiers

- Organisation partielle des dossiers de Tomcat



Jakarta Tomcat : Tomcat Manager

- Tomcat fournit deux modes pour sa gestion:
  - 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 les versions 9 et 10):

```
<user username="admin" password="" roles="manager-gui"/>
<user username= "robot" password="" roles="manager-script"/>
```

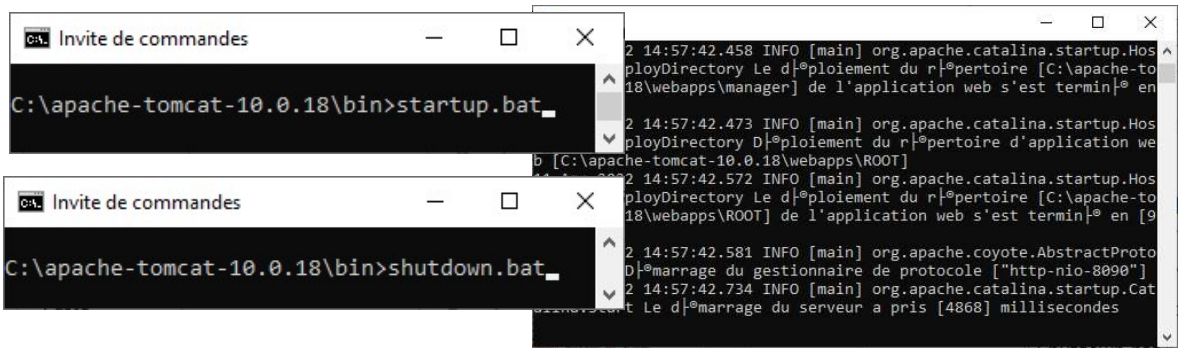

Jakarta Tomcat : Changement de port

- Le port par défaut de Tomcat est 8080
- Si le port est déjà occupé par un autre serveur, vous pouvez changer la port à partir du fichier « server.xml » du dossier « conf »

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

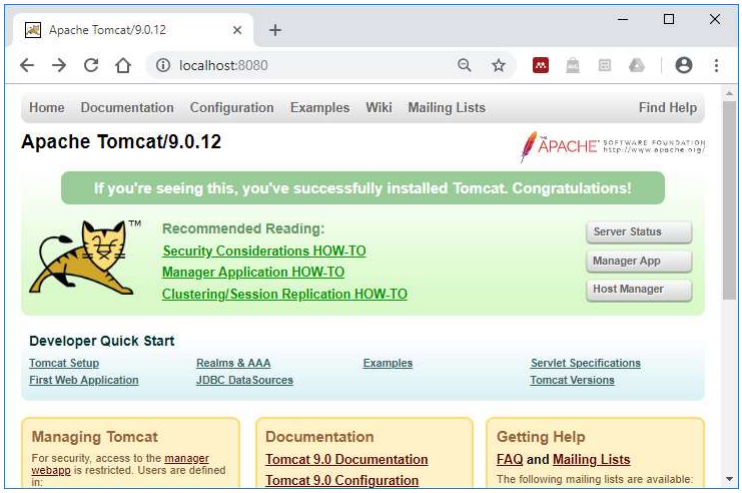
Jakarta Tomcat : démarrage et arrêt manuel

- Les scripts de démarrage de Tomcat sont logés dans le dossier « bin » de Tomcat :
  - « Startup.bat » pour le démarrage
  - « Shutdown.bat » pour l'arrêt



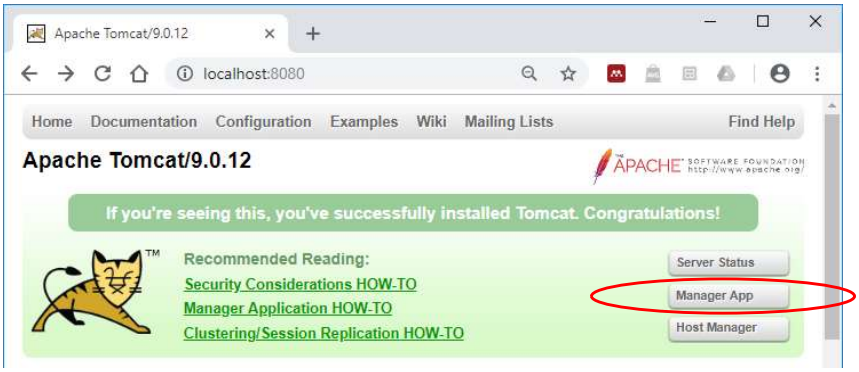
Jakarta Tomcat

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

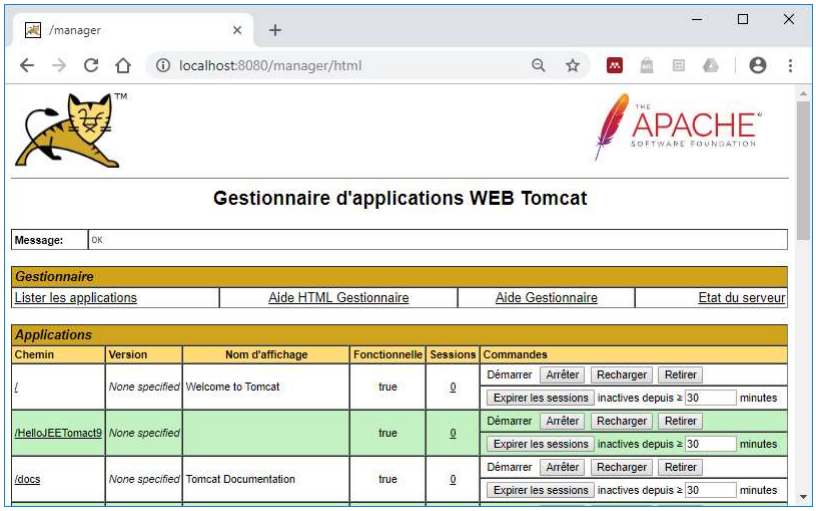


Tomcat Manager

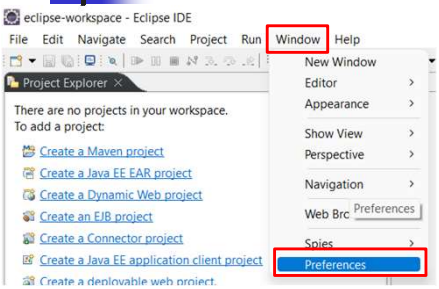
- Les applications hébergées sur Tomcat peuvent être gérées (démarrées, arrêtées, rechargées) grâce au **Tomcat Manager**.

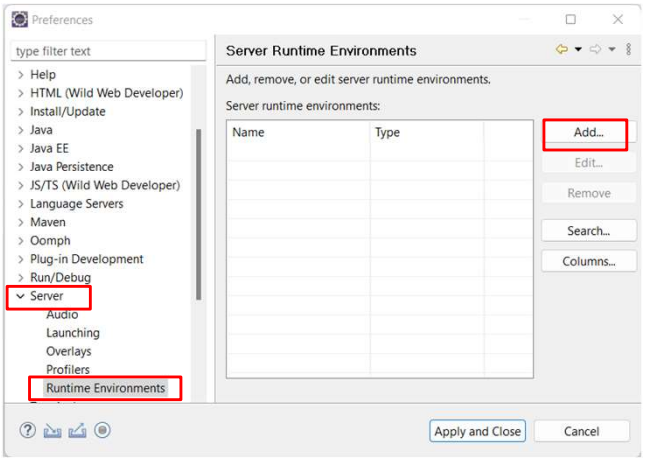


Tomcat Manager

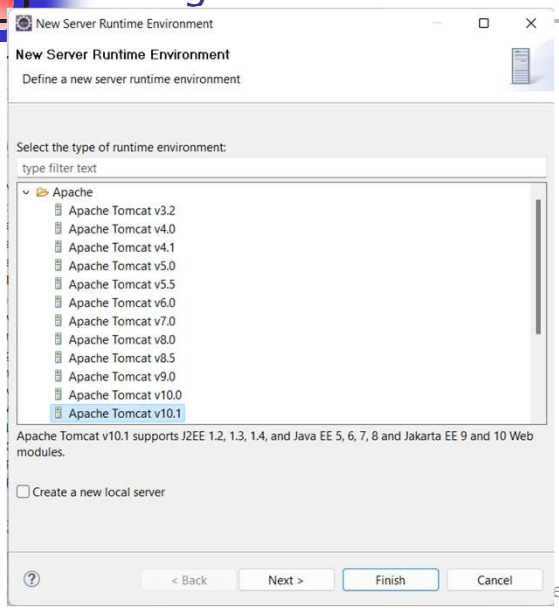


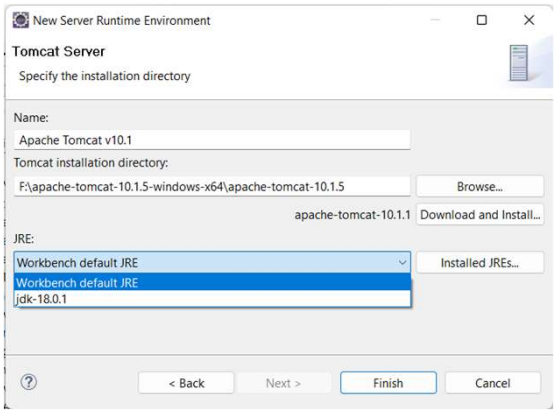
Configuration Tomcat avec Eclipse



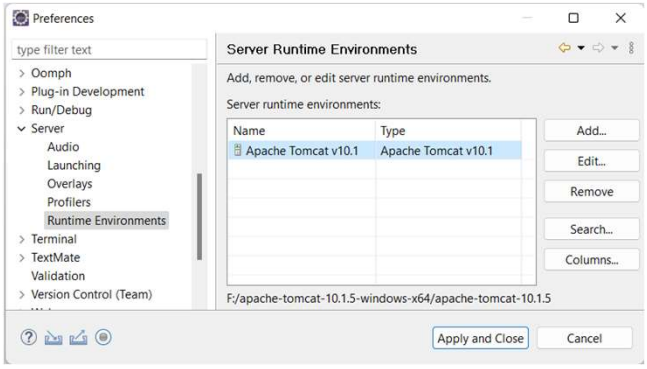


Configuration Tomcat avec Eclipse

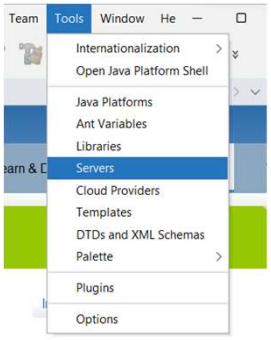


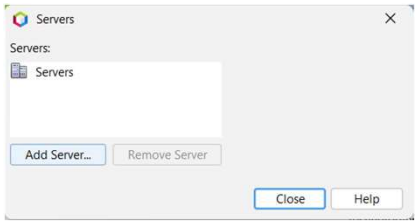


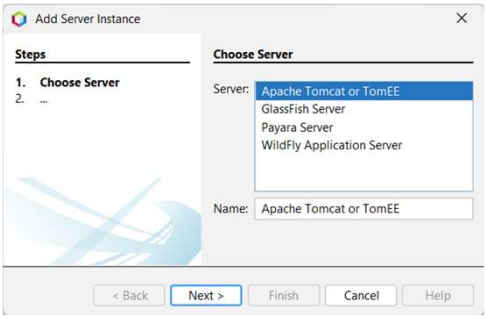
Configuration Tomcat avec Eclipse



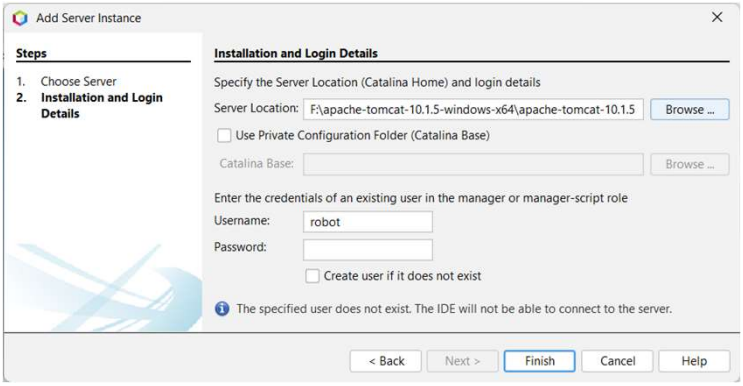
Configuration Tomcat avec NetBeans



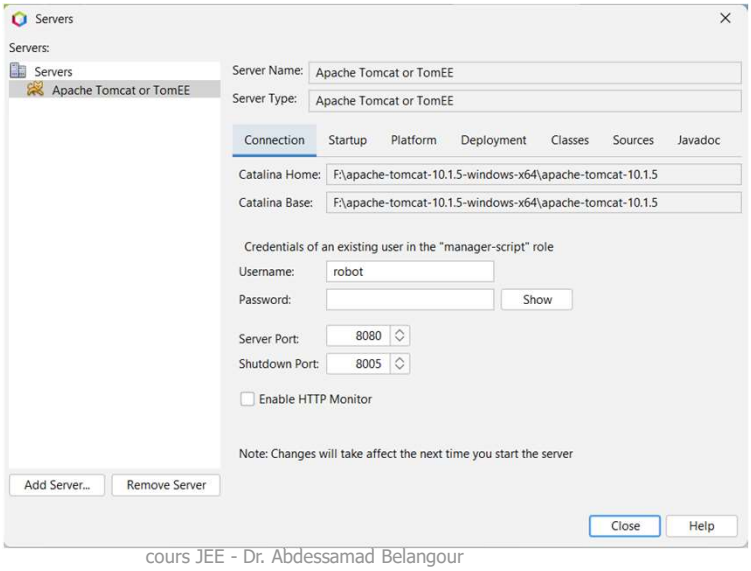




Configuration Tomcat avec NetBeans



Configuration Tomcat avec NetBeans



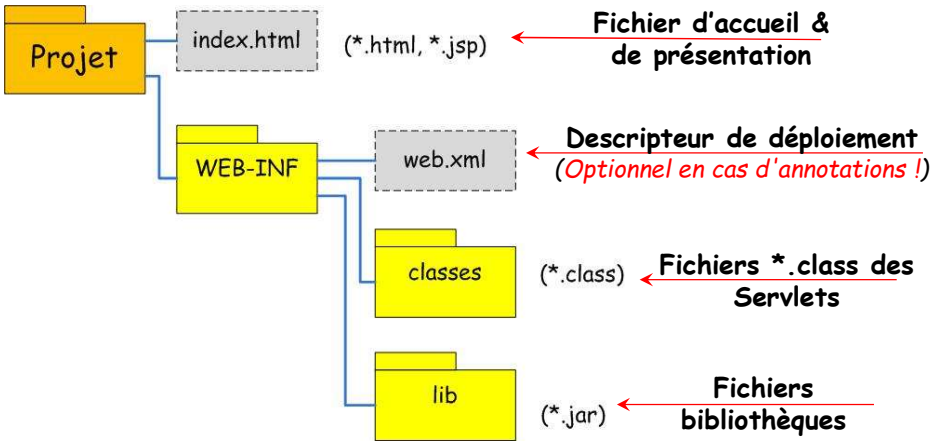
Chapitre 2 : Les Servlets

Introduction

- Une Application Web est composée de plusieurs liens ou chemins (URL)
- Chaque lien mène à une page HTML différente
- En JEE chaque page renvoyée est écrite dynamiquement par une classe Java appelée « servlet »
- L'ensemble des fichiers composant une application Web en JEE est appelée contexte de l'application

Structure d'une application Web sous Tomcat

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



Description du déploiement

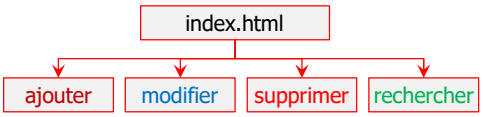
- La définition des différents liens, et des servlets qui en sont responsables peut être :
  - Soit rassemblée dans un fichier **Web.xml** contenant les chemins de toutes les servlets du projet (en plus à d'autres configurations)
  - Soit définies individuellement au niveau des classes de chaque servlet en utilisation l'annotations **@WebServlet**

```
<servlet>
  <servlet-name>nomServlet</servlet-name>
  <servlet-class>NomClasseServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name> nomServlet </servlet-name>
  <url-pattern>/cheminServlet </url-pattern>
</servlet-mapping>
```

```
@WebServlet("/cheminServlet ")
```

Description du déploiement

- Exemple : Gestion des Etudiants
  - Cas du fichier web.xml

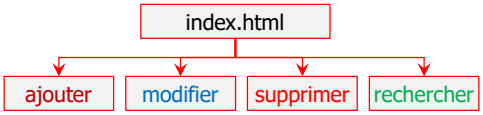



```
<servlet>
  <servlet-name>AS</servlet-name>
  <servlet-class>AjouterServlet</servlet-class>
</servlet>

<servlet>
  <servlet-name>MS</servlet-name>
  <servlet-class>ModifierServlet</servlet-class>
</servlet>

<servlet>
  <servlet-name>SS</servlet-name>
  <servlet-class>SupprimerServlet</servlet-class>
</servlet>

<servlet>
  <servlet-name>RS</servlet-name>
  <servlet-class>RechercherServlet</servlet-
class>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>AS</servlet-name>
  <url-pattern>/ajouter</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>MS</servlet-name>
  <url-pattern>/modifier</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>SS</servlet-name>
  <url-pattern>/supprimer</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>RS</servlet-name>
  <url-pattern>/rechercher</url-pattern>
</servlet-mapping>
```


Description du déploiement

- Cas des annotations




```
@WebServlet("/ajouter")
public class AjouterServlet extends HttpServlet {
...
}
```

```
@WebServlet("/supprimer")
public class SupprimerServlet extends HttpServlet {
...
}
```

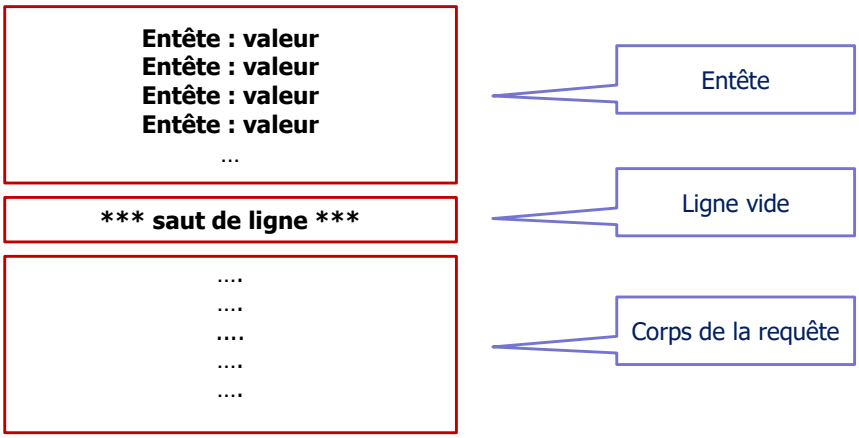
```
@WebServlet("/modifier")
public class ModifierServlet extends HttpServlet {
...
}
```

```
@WebServlet("/rechercher")
public class RechercherServlet extends HttpServlet {
...
}
```

Le protocole HTTP

- HTTP (HyperText Tranfert Protocol) agit comme un protocole de Transport qui :
  - Transporte la requête du client vers la servlet
  - Transporte la réponse de la servlet vers le client
- Protocole qui permet au client de :
  - Récupérer des documents statiques (PDF, Image,..) ou dynamiques (JSP, ASP…)
  - Soumissionner les formulaires
- Une requête HTTP peut être envoyée en utilisant les méthodes suivantes:
  - **GET** : Pour récupérer le contenu d'un document depuis le serveur
  - **POST** : Pour soumettre des formulaires au serveur

Le protocole HTTP : Forme d'une requête/réponse HTTP



Le protocole HTTP : Exemple

Requête	Réponse
POST authentication **HTTP/1.0** **Accept** : text/html **Accept-Language** : fr **User-Agent** : Mozilla/4.0	**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-legnth :** 4205
*** saut de ligne ***	*** saut de ligne ***
login=val1&pass=val2&....	<!DOCTYPE html> <HTML> <HEAD>.... </HEAD> <BODY> .... </BODY> </HTML>

Remarque : Dans le cas d'envoi avec la méthode POST le contenu de la requête est envoyé dans son corps

Le protocole HTTP : Types MIME

- Les types des documents transportés par HTTP sont précisés grâce au MIME
- MIME veut dire Multipurpose Internet Mail Extensions
- 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)

Une première Servlet

- Une Servlet:
  - Est une classe Java fonctionnant du côté serveur (Tomcat)
  - Reçoit des requêtes HTTP d'un client Web
  - Effectue traitement
  - Ecrit une réponse HTTP dynamique renvoyée au client Web
- Le serveur d'application :
  - Gère le cycle de vie des servlets
  - Il leur passe les requêtes et reprend les réponses

Une première Servlet

- Généralement, la structure d'une servlet est comme suit :

```
@WebServlet("/cheminServlet")
public class NomClasseServlet extends HttpServlet {

@Override
 public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // traitement
 }
 @Override
 public void doPost(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {
        // traitement
 }

}
```

- Remarque : doGet() et doPost() contiennent le même traitement

Une première Servlet : avec annotation

Chemin virtuel de la servlet (URL-Pattern)
`http://localhost:8080/projet/salut`

```
package com.exemple;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotations.*;
import java.io.*;
@WebServlet("/salut")
public class HelloWorldServlet extends HttpServlet {// la méthode doGet traite les requêtes envoyées avec GET
 @Override
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
          response.setContentType("text/html"); // précision du type MIME de contenu à renvoyer au client
          PrintWriter out = response.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  }
}
```


Une première Servlet

- Remarques :
  - La méthode **doGet** (resp. **doPost**) traite les requêtes envoyées avec **GET** (resp. **POST**)
  - Elle prend deux paramètres :
    - Un paramètre de type ***HttpServletRequest*** représentant la requête client
    - Un paramètre de type ***HttpServletResponse*** représentant la réponse à renvoyer au client
  - Les servlets font partie de la bibliothèque « **servlet-api.jar** » qui se trouve dans le dossier « **lib** » de Tomcat
  - Elle se compose des packages : « jakarta.servlet » et « jakarta.servlet.http »

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>
```

- 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 fichier web.xml

- Le fichier « web.xml » est une alternative plus ancienne aux annotations
- C'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 : avec fichier web.xml

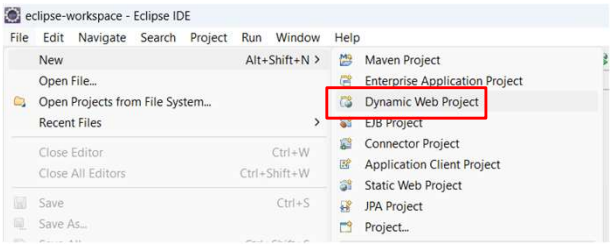
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
   http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
   <display-name>Ma 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>/hello</url-pattern>
   </servlet-mapping>
</web-app>
```

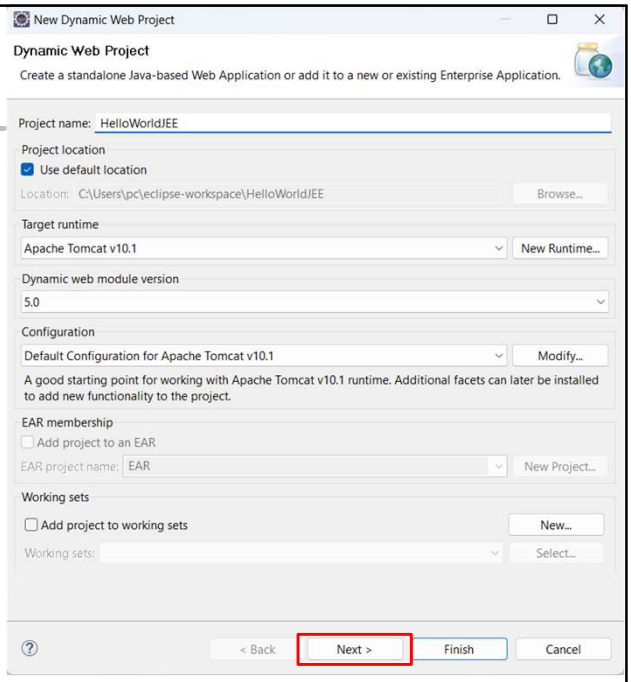

Une première Servlet : avec fichier web.xml

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

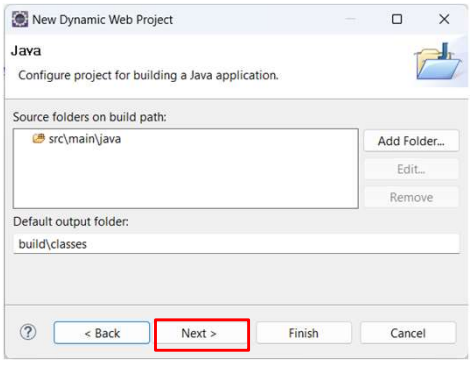
```
<web-app>
   <welcome-file-list>
     <welcome-file>accueil.html</welcome-file>
     <welcome-file>sommaire.html</welcome-file>
   </welcome-file-list>
   <error-page>
     <error-code>404</error-code>
     <location>/erreur404.html</location>
   </error-page>
   <error-page>
     <error-code>500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

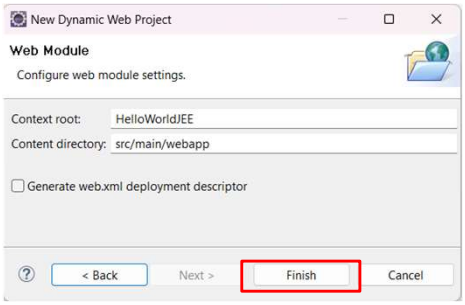
Etapes avec Eclipse





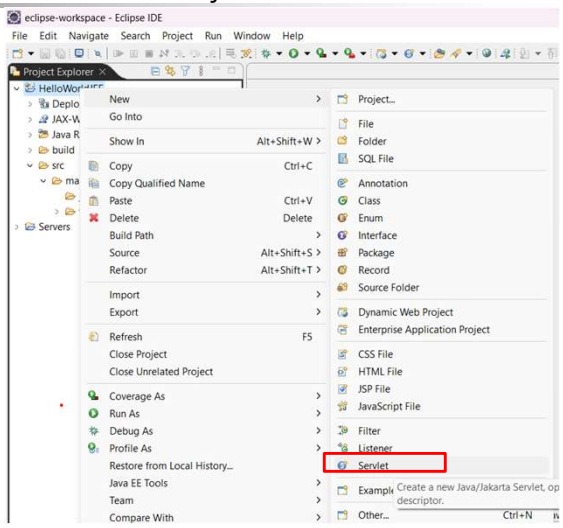
Etapes avec Eclipse



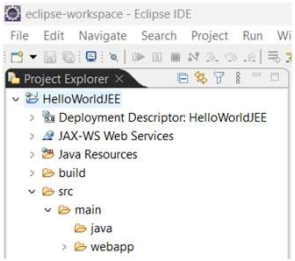


Etapes avec Eclipse

Ajout servlet

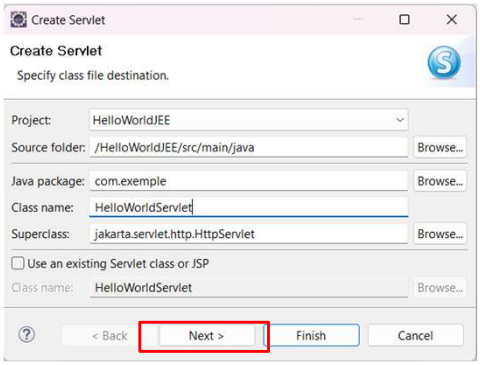


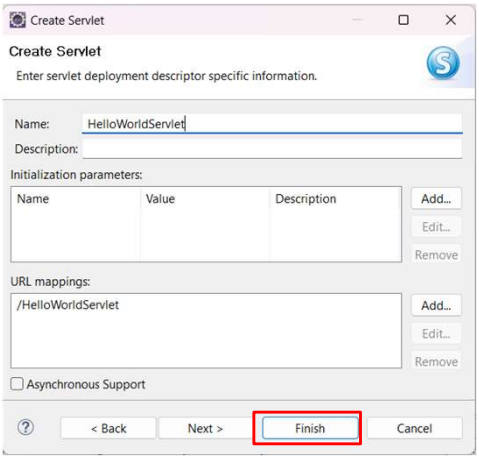
Structure du projet :



Etapes avec Eclipse

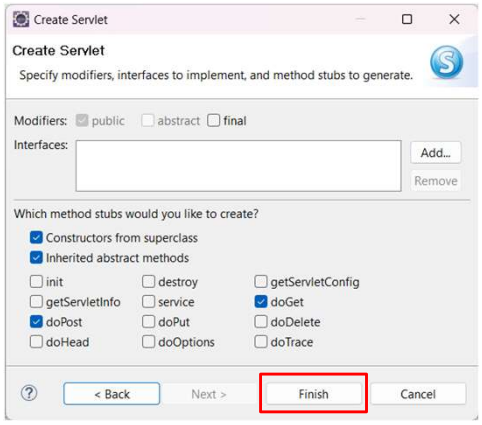
- Détail servlet



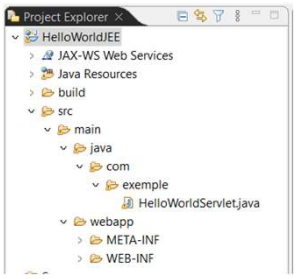


Etapes avec Eclipse

Génération code servlet

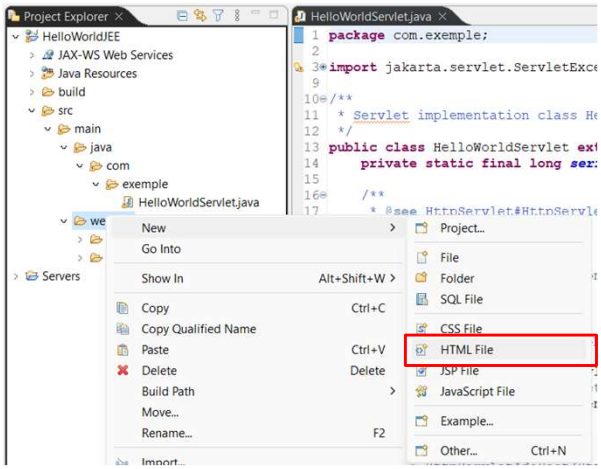


Structure du projet après ajout servlet



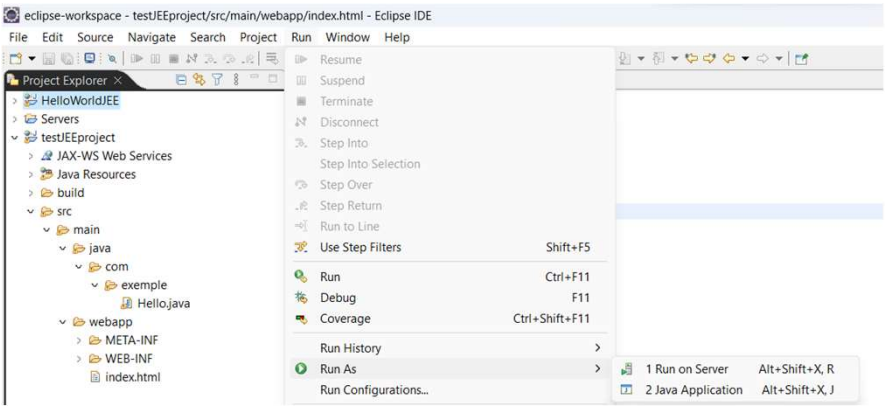
Etapes avec Eclipse

Ajout fichier « index.html »

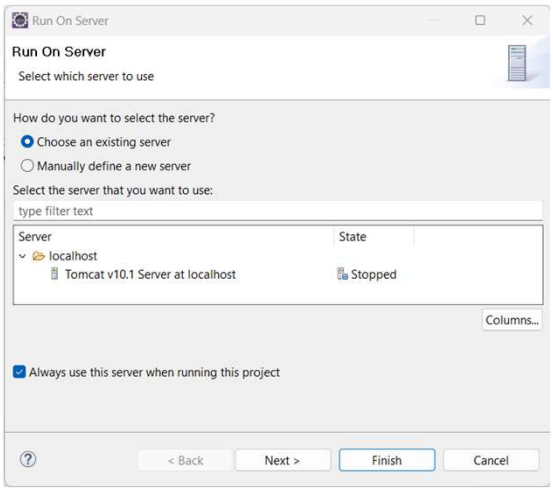


Etapes avec Eclipse

- Exécution : Run → Run As → Run on server



Etapes avec Eclipse



Paramètres d'initialisation d'une servlet

- Une servlet peut admettre un ou plusieurs paramètres d'initialisation
- Un paramètre d'initialisation d'une servlet est :
  - une variable qui a un nom et une valeur et une description
  - Elle peut être stockée dans le fichier web.xml ou avec des annotations
  - Récupérable au démarrage de la Servlet grâce à la méthode ***getInitParameter()***

```
<servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>HelloServlet</servlet-class>
    <init-param>
        <param-name>nom</param-name>
        <param-value>Ali</param-value>
        <description>nom responsable</description>
    </init-param>
</servlet>
```

OU

```
@WebServlet(value = "/salutation" , initParams = { @WebInitParam(name = "nom", value = "Ali")})
```


Paramètres d'initialisation d'une servlet

- Exemple :

```
public class SalutationServlet extends HttpServlet {
 private String nom;
 @Override
 public void init() throws ServletException {
    nom = this.getInitParameter("nom");
 }
 @Override
protected void doGet(HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html> ");
        out.println("<HTML><HEAD><TITLE> Titre </TITLE></HEAD>");
        out.println("<BODY> Bonjour"+ nom +"</BODY></HTML>");
        out.close();
    }
}
```

L'annotation @WebServlet

- Sert à déclarer une servlet avec les attributs suivants:
  - String **name** : Nom de la servlet
  - String[] **value** (ou String[] **urlPatterns**) : tableau des URL patterns
  - Int **loadOnStartup** : indique si la servlet est chargée ou moment du déploiement (valeurs positives) ou au moment de ma première requête (valeurs négatives)
  - 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 @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'API Servlet

- l'API Servlet fournit un ensemble de classes et d'interfaces pour la manipulation des servlets
- Elle s'appuie « **servlet-api.jar** » et se trouve dans le dossier « **lib** » de Tomcat
- L'API servlet regroupe un ensemble de classes dans deux packages :
  - **jakarta.servlet** : contient les classes pour développer des servlets génériques indépendantes d'un protocole
  - **jakarta.servlet.http** : contient les classes pour développer des Servlets qui reposent sur le protocole http utilisé par les serveurs web.
- Le package **jakarta.servlet.annotation** est destiné aux annotations

Le package jakarta.servlet

- Le package jakarta.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 jakarta.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 jakarta.servlet.http

- Le package **jakarta.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 reponse de la servlet selon le protocole http
    - **HttpSession** : Définit un objet qui représente une session

Le package jakarta.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

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

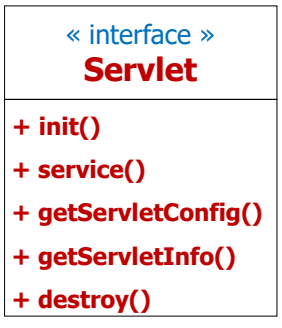
- Quelques méthodes de l'objet ServletContext :
  - **String getInitParameter(String name)** : récupère le paramètre d'initialisation de la servlet, passé en paramètre.
  - **String getServerInfo()** : retourne le nom et la version du conteneur de servlet sur lequel la servlet tourne
  - **int getSessionTimeout()** : 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 *jakarta.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 de ces méthodes au besoin



La méthode init()

- Signature :
  - public void init(ServletConfig config) throws ServletException
- Fonctionnement :
  - Est appelée par le conteneur à chaque instanciation de la servlet
  - Lors de l'instanciation, le conteneur de servlet passe en argument à la méthode *init()* un objet implementant *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 jakarta.servlet.*;
import jakarta.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 request, HttpServletResponse response) throws
  ServletException, IOException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        compteur++;
        out.println("Depuis son chargement, on a accédé à cette Servlet " +compteur+"
  fois.");
 }
}
```


La méthode service()

- Signature :
  - public void service(ServletRequest request, ServletResponse response) 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*

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. Création 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 **getServerName**() : Récupère le nom du serveur
- String **getServerPort**() : Récupère le numéro de port du serveur
- 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

Atelier

- 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

- 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
  - Les valeurs saisie par l'utilisateur dans le formulaire

Solution (1/3)

Index.html

```
<!DOCTYPE html>
<HTML>
 <HEAD> <title> formulaires et servlets </title></HEAD>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<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>
```

url-pattern de la servlet

Solution (2/3)

```
import jakarta.servlet.*;
import jakarta.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("<br><H3>Récupération des paramètres utilisateur </H3><br>") ;
    out.println("<br>nom:"+request.getParameter("nom"));
    out.println("<br>prénom:"+request.getParameter("prenom"));
```

Solution (3/3)

```
        out.println("<br>sexe:"+request.getParameter("sexe"));
        out.println("<br>fonction:"+request.getParameter("fonction"));
        out.println("<br>commentaire:"+request.getParameter("commentaire"));
        out.println("<br>loisirs:<br>");
        String[] valeursDeLoisirs=request.getParameterValues("loisirs");
        for (int i=0 ; i < valeursDeLoisirs.length ; i++) {
                out.println(valeursDeLoisirs[i]+" ");
       }
        out.println(" </body></html>");
        out.close();
    }
}
```


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 **setContentType**(String type) : Définit le type MIME de la réponse HTTP, c'est-à-dire le type de données envoyées au navigateur
  - void **setContentLength**(int len) : Définit la taille de la réponse
  - void **sendredirect**(String location) : Permet de rediriger le client vers l'URL *location*

Développer une servlet http

- PrintWriter **getWriter**() : Retourne un objet *PrintWriter* permettant d'envoyer du texte au navigateur client.
- 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 jakarta.servlet.*;
import jakarta.servlet.http.*;
import java.io.*;
public class RedirectionServlet extends HttpServlet {
  @Override
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws
   ServletException, IOException {
       response.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 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

- La classe **Cookie** du package *jakarta.servlet.http* permet de travailler avec les Cookies.
- Constrcuteur :
  - *Cookie(String name, String value)* : construit un cookie
- Méthodes :
  - *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

Suivi de session : cookies

- La Servlet récupère les cookies du client en exploitant la requête (*HttpServletRequest*)
  - *Cookie[] getCookies()* : récupère l'ensemble des cookies du site
  - Exemple :

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

- 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
  - Exemple :

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


Atelier

- 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 identifiant 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 jakarta.servlet.http.*;
import jakarta.servlet.*;
import java.io.*;
public class CookiesServlet extends HttpServlet {
 @Override
 protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       String valeur;
        Cookie[] tabCookies = request.getCookies();
        boolean existe=false;
        if (tabCookies != null)
            for (int i = 0; i < tabCookies .length; i++) {
              if (tabCookies [i].getName().equals("identifiant"))
                        existe=true;
             }
```


Solution (2/2)

```
  if (tabCookies == null || existe == false) {
     out.println("Bonjour le nouveau…mais plus maintenant");
     valeur = new java.rmi.server.UID().toString(); //génère un identifiant unique
     Cookie monCookie= new Cookie("identifiant", valeur);
     monCookie.setMaxAge(60*60*24*365); //durée de validité d'un an du cookie
     response.addCookie(monCookie);
 }
 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 *jakarta.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
  - **removeAttribute***(String na)* : supprime l'attribut associé à *na*
  - **Enumeration getAttributeNames***()* : retourne les noms de tous les attributs
- 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 request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        HttpSession s = request.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.

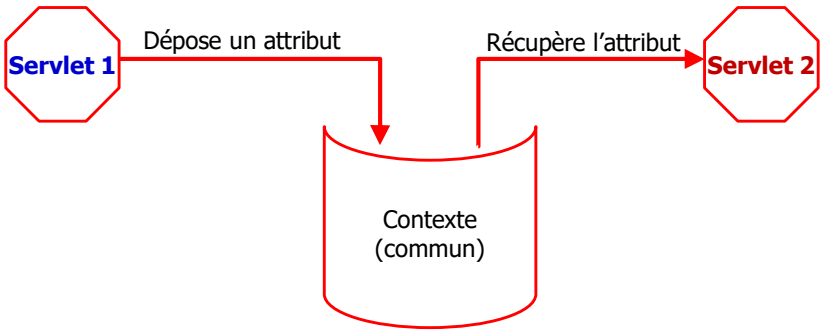
    Exemple : un magasin en ligne pourrait partager les informations sur le stock des produits ou une connexion à une base de données
  - **Partage du contrôle** : une requête.

    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

- Le contexte d'une servlet est commun à toutes les servlets d'un même projet
- Il peut ainsi être utilisé pour stocker ou récupérer des informations en provenance d'autres servlets
- Méthodes :
  - *void setAttribute(String nom, Object valeur)* : lie un objet sous le nom indiqué
  - *Object getAttribute(String nom)* : retrouve l'objet sous le nom indiqué
  - *void removeAttribute(String nom)* : supprime l'objet lié sous le nom indiqué
  - *Enumeration getAttributeNames()* : retourne l'ensemble des noms de tous les attributs liés

Collaboration de Servlets : partage d'information



Partage d'information

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

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

Création d'un attribut

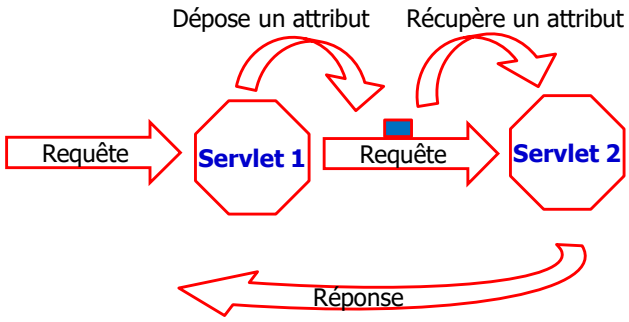
```
public class PizzasClient extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException  {
                        response.setContentType("text/plain");
                        PrintWriter out = response.getWriter();
                        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 jakarta.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)



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)

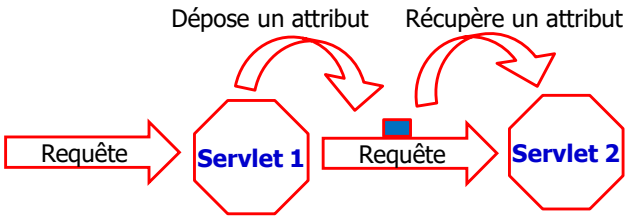
Code pour servlet1

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import java.io.*;
@WebServlet("/servlet1")

public class Servlet1 extends HttpServlet {
 @Override
protected void doGet(HttpServletRequest request,
     HttpServletResponse response)
throws ServletException, IOException {

     request.setAttribute("X", "salut");
     RequestDispatcher dispat =
     request.getRequestDispatcher("/servlet2 ");
     dispat.forward(request, response);
} }
```

Code pour servlet2

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import java.io.*;
@WebServlet("/servlet2")

public class Servlet2 extends HttpServlet {
 @Override
protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

     response.setContentType("text/plain");
     PrintWriter out = response.getWriter();
     String attr=(String) request.getAttribute("X");
     out.println("l'attribut que j'ai récupéré de servlet 1 est: "+ attr);
     out.close();
} }
```


Partage du contrôle (include)



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 et termine le fichier HTML
- 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

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

Code pour Secondaire

```
@WebServlet("/secondaire")
public class Secondaire extends HttpServlet {
 @Override
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
    PrintWriter out = response.getWriter();
    String ch=(String) request.getAttribute("x");
   int x=Integer.parseInt(ch);
    out.println(" <h2> "+  x +" au carré = "+x*x+ "</h2>");
 }
}
```

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 chaine 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 chaine 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 jakarta.servlet.*;
import jakarta.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>
```


Filtres de Servlets : 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

Filtres de Servlets : 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)

Filtres de Servlets : annotation @Webfilter

```
import java.io.IOException;
import jakarta.servlet.annotation.*;
import jakarta.servlet.*;
import java.util.*;
// classe implementant l'interface Filter
@WebFilter(urlPatterns = {"/*"}, initParams = { @WebInitParam (name = "test-param", value = "paramètre d'Initialization")})
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 chaine de filtres  }
  public void destroy( ) {
     /* Appelée avant que l'instance de filtre ne soit supprimée du service par le conteneur Web */ }
}
```

Filtres de Servlets : annotation @Webfilter

- Remarque :
  - L'annotation @Webfilter ne permet pas de définir l'ordre des filtres en cas de plusieurs filtres
  - Solution: mettre le nom du filtre dans l'annotation et le mapping dans le fichier web.xml

```
<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>
```