Introduction à la plateforme J2EE

Dr. Abdessamad Belangour

Plan du cours

- Partie 1 : Survol de la plateforme J2EE
- Partie 2 : Servlets
- Partie 3 : Java Server Pages (JSP)
- Partie 4 : Accès aux bases de données avec JDBC
- Partie 5 : Entreprise Java Beans (EJB)
- Partie 6 (optionnelle): Exposés
  - Le framework Struts
  - Hibernate
  - JSF
  - …

Partie 1 : Survol de la plateforme J2EE

Pourquoi utiliser une plateforme ?

- Une plateforme est une base générique qui fournit un ensemble de fonctionnalités utiles pour une majorité d'applications.
- Une plateforme se construit sur la base d'un ensemble de besoins génériques partagés entre plusieurs applications (accès aux bases de données, communication réseaux, etc..)
- Avantages :
  - Ne pas re-coder des fonctionnalités communes et récurrentes
  - Se concentrer sur les fonctionnalités réelles de l'application en cours de développement.
  - ➔ Gain de temps et d'argent

Java Framework

- Le Java Framework (Java 2 Platform) est composé de trois éditions, destinées à des usages différents :
  - **J2ME** : *Java 2 Micro Edition* est prévu pour le développement d'applications embarquées, notamment sur des PDA (*Personal Digital Assistant*) et terminaux mobiles (téléphone portables, …) ;
  - **J2SE** : *Java 2 Standard Edition* est destiné au développement d'applications pour ordinateurs personnels ;
  - **J2EE** : *Java 2 Enterprise Edition*, destiné à un usage professionnel avec la mise en oeuvre de serveurs.
- Chaque édition propose un environnement complet pour le développement et l'exécution d'applications basées sur Java et
- Comprend notamment une machine virtuelle Java (Java virtual machine) ainsi qu'une bibliothèque de classes.

La plateforme J2EE

- **J2EE** (*Java 2 Enterprise Edition*) est une norme proposée par la société Sun, portée par un consortium de sociétés internationales, visant à définir un standard de développement d'applications d'entreprises multi-niveaux, basées sur des composants.
- La plateforme J2EE désigne l'ensemble constitué des services (API) offerts et de l'infrastructure d'exécution.
- La norme J2EE comprend :
  - 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.

  Remarque : Sun fournit une implémentation minimale de ces API appelée J2EE SDK (J2EE Software Development Kit).

Les API de J2EE

- Les API de J2EE peuvent se répartir en deux grandes catégories :
  - Les **composants**
    - Les composants web
    - Les composants métier
  - Les **services**
    - Les services d'infrastructures
    - Les services de communication

Les API de J2EE

- Les **composants** :
  - Les composants Web : Servlets et JSP (Java Server Pages). Il s'agit de la partie chargée de l'interface avec l'utilisateur (on parle de *logique de présentation*).
  - Les composants métier : EJB (Enterprise Java Beans). Il s'agit de composants spécifiques chargés des traitements des données propres à un secteur d'activité (on parle de *logique métier* ou de *logique applicative*) et de l'interfaçage avec les bases de données.

Les API de J2EE

- Les **services :**
  - Les **services d'infrastructures** :
    - **JDBC** (*Java DataBase Connectivity*) est une API d'accès aux bases de données relationnelles.
    - **JNDI** (*Java Naming and Directory Interface*) est une API d'accès aux services de nommage et aux annuaires d'entreprises tels que DNS, NIS, LDAP, etc.
    - **JTA/JTS** (*Java Transaction API/ Java Transaction Services*) est une API définissant des interfaces standard avec un gestionnaire de transactions.
    - **JCA** (*J2EE Connector Architecture*) est une API de connexion au système d'information de l'entreprise, notamment aux systèmes dits «Legacy» tels que les ERP.
    - **JMX** (*Java Management Extension*) fournit des extensions permettant de développer des applications web de supervision d'applications.

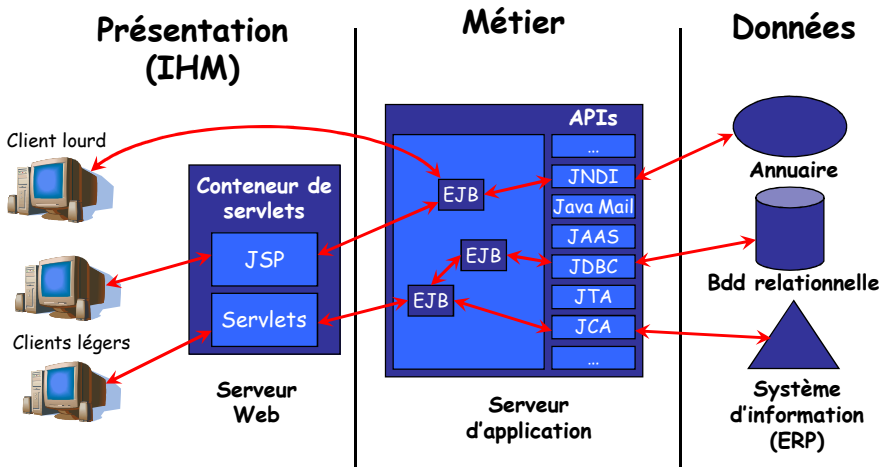
Les API de J2EE

- Les **services de communication** :
  - **JAAS** (*Java Authentication and Authorization Service*) est une API de gestion de l'authentification et des droits d'accès.
  - **JavaMail** est une API permettant l'envoi de courrier électronique.
  - **JMS** (*Java Message Service*) fournit des fonctionnalités de communication asynchrone (appelées *MOM* pour *Middleware Object Message*) entre applications.
  - **RMI-IIOP** (*Remote Method Invocation Over Internet Inter-ORB Protocol*) est une API permettant la communication synchrone entre objets.

L'architecture J2EE

- L'architecture J2EE 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 J2EE



Types de clients

- **Client lourd** (en anglais *fat client* ou *heavy client*) : désigne une application cliente graphique exécutée sur le système d'exploitation de l'utilisateur. Un client lourd possède généralement des capacités de traitement évoluées et peut posséder une interface graphique sophistiquée.
- **Client léger** (en anglais thin client) : désigne une application accessible via une interface web (en HTML) consultable à l'aide d'un navigateur web, où la totalité de la logique métier est traitée du côté du serveur. Pour ces raisons, le navigateur est parfois appelé *client universel*.
- **Client riche** est un compromis entre le client léger et le client lourd. L'objectif du client riche est de proposer une interface graphique, décrite avec une grammaire de description basée sur la syntaxe XML, permettant d'obtenir des fonctionnalités similaires à celles d'un client lourd (glisser déposer, onglets, multi fenêtrage, menus déroulants).

Serveur d'application

- Un serveur d'application 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
- Exemples :
  - JBoss (*www.jboss.org*)
  - WebSphere Application Server d'IBM
  - Weblogic de BEA (*www.bea.com*)
  - ..etc

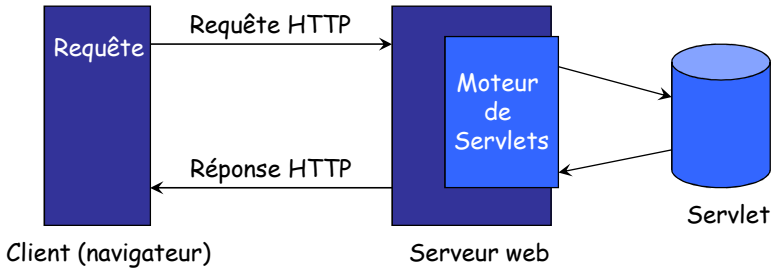
Partie 2 : Les Servlets

Introduction aux servlets

- Ce sont des applications Java fonctionnant du côté serveur (tels que ASP ou bien PHP).
- Les servlets sont au serveur Web ce que les applets sont au navigateur pour le client.
- Servlet:
  - Reçoit des requêtes HTTP
  - Effectue traitement
  - Fournit une réponse HTTP dynamique au client Web (permet donc de créer des pages web dynamiques).

Introduction aux servlets

- les servlets sont indépendantes du serveur web
  - Elles s'exécutent dans un **moteur de servlets** utilisé pour établir le lien entre la servlet et le serveur Web



Avantages des servlets

- Les servlets ont de nombreux avantages par rapport aux autres technologies côté serveur:
  - Peut utiliser toutes les API Java afin de communiquer avec des applications extérieures, se connecter à des bases de données, accéder aux entrée-sorties…
  - Sont indépendantes du serveur Web
  - Se chargent automatiquement lors du démarrage du serveur ou bien lors de la connexion du premier client.
  - La résidence en mémoire leur permettent :
    - De traiter les demandes des clients grâce à des threads.
    - D'occuper moins de mémoire et de charge du processeur.
  - À l'opposé, les langages de script traditionnels créent de nouveaux processus pour chaque requête HTTP.
  - permettant de créer des composants réutilisables.

Moteur de servlets

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

Moteur de servlets

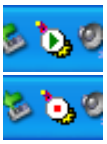
- Nombreux conteneurs de Servlet
  - Jakarta Tomcat (*jakarta.apache.*
  - JBoss (*www.jboss.org*)
  - WebSphere Application Server d'IBM
  - Weblogic de BEA (*www.bea.com*)
- Dans le reste du cours et des TP, nous utiliserons le conteneur Tomcat pour déployer nos servlets.

Jakarta Tomcat

- Tomcat 5.5.4 respecte la spécification Servlet 2.4 et JSP 2.0
- Écrit entièrement en Java, il peut donc être utilisé sur n'importe quel système disposant d'une machine virtuelle
- Disponible gratuitement sous forme d'une licence Open Source
- Nécessite obligatoirement une machine virtuelle respectant la spécification 5.0 (jre 1.5.0)
- Implémentation de référence de la spécification J2EE. Il fournit donc les librairies de façon à concevoir des Servlets (*javax.servlet.http.HttpServlet*)

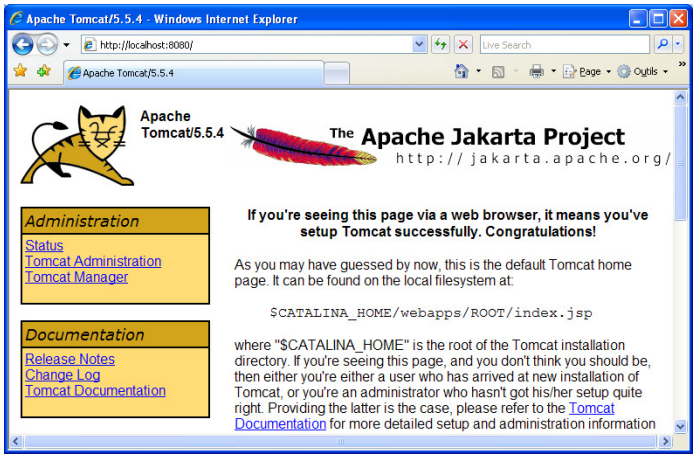
Jakarta Tomcat

- 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



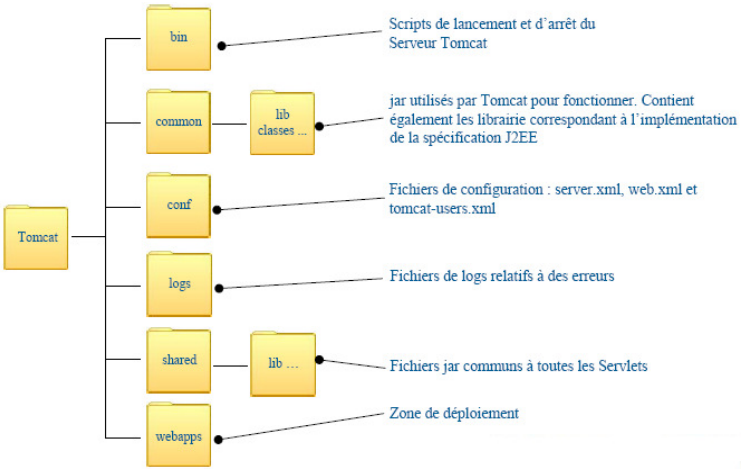
Jakarta Tomcat

- Pour accéder à la page d'accueil de Tomcat il suffit de taper : ***http://localhost:8080***



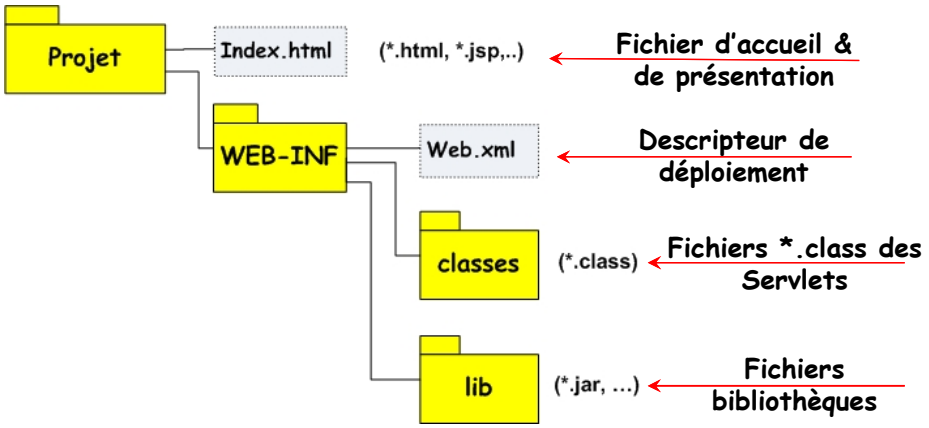
Hiérarchie des dossiers Tomcat

- Organisation partielle des dossiers de Tomcat



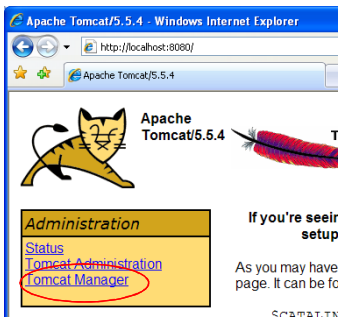
Déploiement d'une Servlet dans Tomcat

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



Tomcat Manager

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

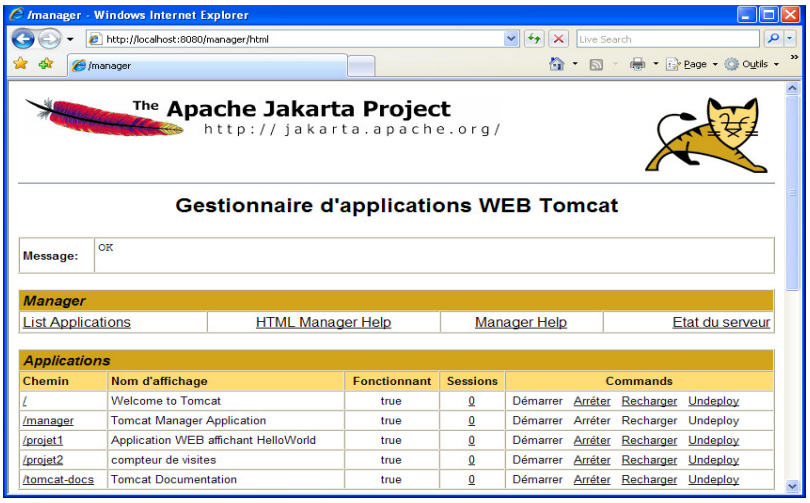


Tomcat Manager

- Le lancement de Tomcat Manager demande un login et un mot de passe.
- Par défaut
  - le login est : admin
  - Le mot de passe est vide
- Si vous voulez changer ces valeurs pour pouvez éditer le fichier « tomcat-users.xml » qui se trouve dans le dossier conf de tomcat.
- Exemple :

```
<user username="admin" password="" roles="admin,manager"/>
```


Tomcat Manager



Le 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..

Le fichier web.xml : Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <display-name>Ma première application Web</display-name>
  <servlet>
      <servlet-name>Hello</servlet-name>
      <servlet-class>HelloWorldServlet</servlet-class>
  </servlet>
  <servlet-mapping>
      <servlet-name>Hello</servlet-name>
      <url-pattern>/salut</url-pattern>
  </servlet-mapping>
</web-app>
```

Le fichier web.xml

- Si le dossier contenant notre Servlet se nomme projet , le chemin d'accès à notre servlet sera : **http://localhost:8080/projet/salut**
- 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 »

Exemple de servlet

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

Analyse de l'exemple

- La première étape consiste à importer les packages nécessaires à la création de la servlet
- il faut donc importer *javax.servlet*, *javax.servlet.http* et *java.io*
  - import javax.servlet.*;
  - import javax.servlet.http.*;
  - import java.io.*

Analyse d'une Servlet

- Si le type de la requête est GET, alors la méthode de la Servlet qui traite la requête est la suivante :

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        //traitement
}
```

- Si le type de la requête est POST alors la méthode doit être

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws IOException, ServletException {
        //traitement
}
```

Analyse d'une servlet

- Remarque :
  - Dans le cas où nous connaissions pas la méthode d'envoi de la requête (GET ou POST) une astuce serait d'écrire les deux méthode comme suit :

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        //traitement
}
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws IOException, ServletException {
        doGet(req, res);
}
```


Analyse d'une servlet

- 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
- Remarque :
  - L'objet *HttpServletRequest* permet d'extraire toutes les informations sur le client (adresse IP, navigateur, Domaine, paramètres d'un formulaire, etc..)

Analyse d'une servlet

- L'objet *HttpServletResponse* doit etre complété d'informations par la servlet avant de le renvoyer au client.
- La première étape consiste à définir le type de données qui vont être envoyées au client (généralement il s'agit d'une page HTML).
- La méthode *setContentType()* de l'objet *HttpServletResponse* prend donc comme paramètre le type MIME associé au format HTML (*text/html*) :
  - res.setContentType("text/html");

Analyse d'une servlet

- La création d'un objet *PrintWriter* grâce à la méthode *getWriter()* de l'objet *HttpServletResponse* permet d'envoyer du texte formatté au navigateur

  PrintWriter out = res.getWriter();
- La méthode *println()* de l'objet *PrintWriter* permet d'envoyer les données textuelles au navigateur
  - out.println("<HTML>");
  - …
  - out.println("</HTML>");
- L'objet *PrintWriter* doit être fermé lorsqu'il n'est plus utile avec sa méthode *close()*
  - out.close();

Notion de Contexte

- Un contexte constitue pour chaque Servlet d'une même application une vue sur le fonctionnement de cette application.
- Une application web peut être composée de :
  - Servlets
  - JSP
  - Classes utilitaires
  - Documents statiques (pages html, images, sons, etc.)
  - Etc..
- Grâce à ce contexte, il est possible d'accéder à chacune des ressources de l'application web correspondant au contexte.
- Dans le code source d'une servlet, un contexte est représenté par un objet de type ServletContext.
- Exemple :
  - *getServletContext().getServerInfo()* retourne le nom du logiciel utilisé pour prendre en charge la requête , par exemple Tomcat/3.2.1.

Notion de Contexte

- Chaque contexte est propre à une application et qu'il n'est pas possible de partager des ressources entre applications différentes.
- A chaque contexte correspond une arborescence dans le système de fichiers qui contient les ressources accédées lors des requêtes vers le moteur de servlets comme nous avons vu pour Tomcat.
- Le fichier web.xml est donc un descripteur de déploiement du contexte. Il peut contenir entre autres:
  - Les paramètres d'initialisation du contexte.
  - Les définitions des servlets et des JSPs.
  - La liste des fichiers de bienvenue.
  - Les pages d'erreur.
  - Etc..

Notion de Contexte

- Exemple de fichiers web.xml:

```
...
<web-app>
   <servlet>
     <servlet-name>maServletToto</servlet-name>
     <servlet-class>Toto</servlet-class>
   </servlet>
   <servlet-mapping>
     <servlet-name>maServletToto</servlet-name>
     <url-pattern>/maServlet</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
     <welcome-file>index.html</welcome-file>
     <welcome-file>accueil.html</welcome-file>
   </welcome-file-list>
   <error-page>
     <error-code>404</error-code>
     <location>/404.html</location>
   </error-page>
</web-app>
```


Atelier 1

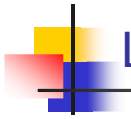
- Reprendre l'exemple de servlet précédent
- Faire en sorte d'écrire un faux nom de servlet
- Remarquer l'appel automatique à la page html indiquant l'erreur comme mentionné par le fichier web.xml

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.

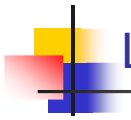
Le package javax.servlet

- Le package **javax.servlet** définit plusieurs interfaces, méthodes et exceptions :
  - Les interfaces :
    - 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.
    - Servlet : interface de base d'une servlet
    - 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
    - ServletRequest : Définit un objet contenant la requête du client.
    - 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



Le package javax.servlet

- 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 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 (classe devenue obsolète)

L'interface d'une servlet

- Pour pouvoir être gérée par le conteneur Web, toute servlet doit implémenter l'interface *Servlet* appartenant au package *javax.servlet*
- Cette interface permet ainsi au conteneur Web de gérer le cycle de vie de la 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()*

servlet
init() *service()* *getServletConfig()* *getServletInfo()* *destroy()*

L'interface d'une servlet

- Afin de mettre en place l'interface *Servlet* nécessaire au conteneur de servlet, il existe plusieurs possibilités :
  1. Définir manuellement chaque méthode
  2. Dériver la classe GenericServlet et redéfinir les méthodes dont on a besoin
  3. Dériver la classe HttpServlet et redéfinir les méthodes dont on a besoin
- C'est la 3eme solution que nous utilisons presque tout le temps

La méthode init()

- Signature :
  - public void init(ServletConfig config) throws ServletException
- 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 *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 Compteur1 extends HttpServlet {
   private int compteur;
   public void init() throws ServletException {
        compteur = 0;
   }
```

La méthode init()

```
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 faire des initialisations au niveau du fichier web.xml et de les utiliser dans la méthode init().
- 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 Compteur2 extends HttpServlet {
 private int compteur;
 public void init() throws ServletException {
   String initial = this.getInitParameter("compteur_initial");
     try {
              compteur = Integer.parseInt(initial);
          }
     catch(NumberFormatException e) {
             compteur= 0;
        }
     }
 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
- 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()
- 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()
- 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 :

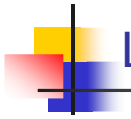
```
public String getServletInfo() {
   return " servlet écrite par x (x@y.com)" ;
}
```

La méthode destroy()

- Signature :
  - void destroy()
- 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.

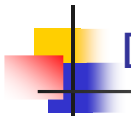
Le cycle de vie d'une servlet

- 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 **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.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;
import java.io.IOException;

public class testServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                 throws ServletException, IOException{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("<html>");
    out.println("  <head>");
    out.println("    <title>entêtes HTTP</title>");
    out.println("  </head>");
    out.println("  <body>");
```

Solution (2/2)

```
out.println("<br><u><font color=red>Method d'envoi du client
    </font></u><br>"); out.println(request.getMethod());
out.println("<br><u><font color=red> adresse IP du client</font></u><br>");
    out.println(request.getRemoteAddr());
out.println("<br><u><font color=red>Nom du serveur  </font></u><br>");
    out.println(request.getServerName());
out.println("<br><u><font color=red>Port du serveur</font></u><br>");
    out.println(request.getServerPort());
out.println("  </body>");
    out.println("</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)

Index.html

```
<HTML>
 <HEAD> <title> formulaires et servlets </title></HEAD>
 <BODY>
    <FORM method="post" action="processform">
     Enregistrement d'un utilisateur
    <TABLE border=0>
      <TR><TD>Nom :</TD> <TD><INPUT type=text name="nom"></TD></TR>
      <TR><TD>Prénom :</TD><TD><INPUT type=text name="prenom"></TD></TR>
      <TR><TD>Sexe :</TD><TD>Homme : <INPUT type=radio name="sexe" value="M" checked><br>
                                    Femme : <INPUT type=radio name="sexe" value="F"></TD></TR>
       <TR><TD>Fonction : </TD>
            <TD><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> </TD></TR>
       <TR><TD>loisirs :</TD>
            <TD><INPUT type=checkbox NAME="loisirs" value="lecture" CHECKED>Lecture
            <INPUT type=checkbox NAME="loisirs" value="sport">Sport
            <INPUT type=checkbox NAME="loisirs" value="voyage">Voyage</TD></TR>
    <TR><TD>Commentaire :</TD><TD><TEXTAREA rows="3" name="commentaire">Tapez ici votre
    commentaire</TEXTAREA></TD></TR>
    <TR><TD COLSPAN=2><INPUT type="submit" value="Envoyer"></TD></TR>
    </TABLE>
    </FORM>
 </BODY>
</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>prénom:"+request.getParameter("prenom"));
out.println("<br>sexe:"+request.getParameter("sexe"));
out.println("<br>fonction:"+request.getParameter("fonction"));
out.println("<br>commentaire:"+request.getParameter("comme
ntaire"));
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 **setContentType**(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...

Atelier 4

- Écrire une servlet qui effectue une re-direction vers un site web donné.

Solution

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

Atelier 5

- Écrire une Servlet qui effectue un téléchargement de fichier sur le client.
  - Le nom réel du fichier sur le serveur se nomme : « c:\monFichier.txt »
  - Le nom que devra voir l'utilisateur est toto.txt
- Rappel:
  - Voir le complément de cours sur les fichiers en Java.

Solution (1/2)

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.IOException;

public class DownloadFileServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
                                        throws ServletException, IOException {
try {
      InputStream is = new FileInputStream("c:\\monFichier.txt");
      OutputStream os = res.getOutputStream();
      res.setContentType("text/plain");
      res.setHeader("Content-Disposition","attachment;filename=toto.txt");
```

Solution (2/2)

```
            int count;
            byte buf[] = new byte[4096]; //buffer de 4 ko
            while ((count = is.read(buf)) > -1)
            os.write(buf, 0, count);
            is.close();
            os.close();
        }
        catch (Exception e) {
            // Y a un problème.
        }
        }
}
```


Envoyer un contenu multimédia

- Pour l'instant nous avons écrit des Servlets qui retournaient des contenus HTML
- Besoin de retourner des contenus différents :
  - Protocole WAP et langage WML utilisés par les téléphones portables
  - Génération de contenus multimédias (création de graphes, manipulation d'images)
- L'API Java facilite la gestion des contenus multimédias en proposant des bibliothèques
  - Encodage d'images sous différents formats (GIF, JPEG)
  - Manipulation et traitement d'images

Envoyer un contenu multimédia

- Exemple : Servlet qui génère et retourne une image JPEG affichant le message « Bonjour monde ! »

```
import com.sun.image.codec.jpeg.*;
import javax.servlet.http.*;
import javax.servlet.*;
import java.awt.*;
import java.awt.image.*;
import java.io.*;
public class ImageServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
    // le contenu produit est une image au format jpeg
            res.setContentType("image/jpeg");
            ServletOutputStream out = res.getOutputStream();
```


Envoyer un contenu multimédia

```
// l'objet enc va encoder les données et les envoyer sur
// le flux de sortie de type ServletOutputStream
    JPEGImageEncoder enc = JPEGCodec.createJPEGEncoder(out);
// création d'une nouvelle image d'une réolution de 1024 par 768
    BufferedImage image = new
    BufferedImage(1024,768,BufferedImage.TYPE_BYTE_INDEXED);
// récupération du contexte graphique lié à l'image
    Graphics2D g = image.createGraphics();
// la prochaine opération s'effectuera avec la couleur rouge
    g.setColor(Color.red);
// affichage de la célèbre phrase
    g.drawString("Bonjour monde !", 400, 500);
// transformation des données au format jpeg et envoi
// de celles-ci sur le flux standard de sortie (le navigateur)
    enc.encode(image);
}
}
```

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

- Code pour récupérer les cookies

```
Cookie[] cookies = req.getCookies();
if (cookies != null) {
for (int i = 0; i < cookies.length; i++) {
String name = cookies[i].getName();
String value = cookies[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.
  - Aide : vous pouvez obtenir un identifiant unique sur le temps par rapport à l'hôte sur lequel il a été généré grâce à la méthode java.rmi.server.UID().toString().

Solution (1/2)

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

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

Solution (2/2)

```
if (contenu == null) {
      contenu = new java.rmi.server.UID().toString();
      Cookie c = new Cookie("monId", contenu);
      c.setMaxAge(60*60*24*365);
      res.addCookie(c);
      out.println("Bonjour le nouveau");
 }
 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
  - *HttpSession getSession(boolean p)* : création selon la valeur de *p*
- Gestion d'association (*HttpSession*)
  - *Enumeration getAttributeNames()* : retourne les noms de tous les attributs
  - *Object getAttribute(String name)* : retourne l'attribut name
  - *setAttribute(String an, Object av)* : associe l'objet av à la chaîne an
  - *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 la suivi de session pour un compteur

```
public class HttpSessionServlet extends HttpServlet {
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        HttpSession session = req.getSession();
        Integer count = (Integer)session.getAttribute("compteur");
        if (count == null)
            count = new Integer(1);
        else
            count = new Integer(count.intValue() + 1);
        session.setAttribute(" compteur ", count);
        out.println("Vous avez visité cette page " + count + " 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", "Anchois");
        context.setAttribute("Date", new Date());
        out.println("La pizza du jour a été définie.");
}}
```

Création de deux attributs

```
public class PizzasClient extends HttpServlet {
protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        ...
        ServletContext context = this.getServletContext();
        String pizza_spec = (String)context.getAttribute("Specialite");
        Date day = (Date)context.getAttribute("Date");
        DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
        String today = df.format(day);
        out.println("Aujourd'hui (" + today + "), notre specialite est : " + pizza_spec);
}}
```

Lecture des attributs

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 *attr1* 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 {
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {

    req.setAttribute("attr1", "salut");
    RequestDispatcher dispat = req.getRequestDispatcher("/maServlet2Path");
    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 {
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(" attr1
    "));
}}
```


Partage du contrôle (include)

- Soit l'exemple suivant :
  - Une servlet (Servlet1) reçoit une requête
  - Elle y place un attribut *attr1* qu'elle y met la chaîne "bonjour"
  - Elle inclut une autre servlet dans le traitement (Servlet2)
  - Servlet2 récupère cet attribut et l'affiche
  - Servlet1 reprend le contrôle, elle modifie *attr1* en "bonsoir" et l'affiche
  - Servlet2 récupère encore une fois cet attribut et l'affiche
  - Servlet1 reprend le contrôle
- Remarque:
  - Servlet2 ne doit pas fermer la réponse par </body> car c'est Servlet1 qui s'en charge.
  - C'est Servlet1 qui se charge de préciser le type mime de la réponse.

Partage du contrôle (include)

Code pour servlet1

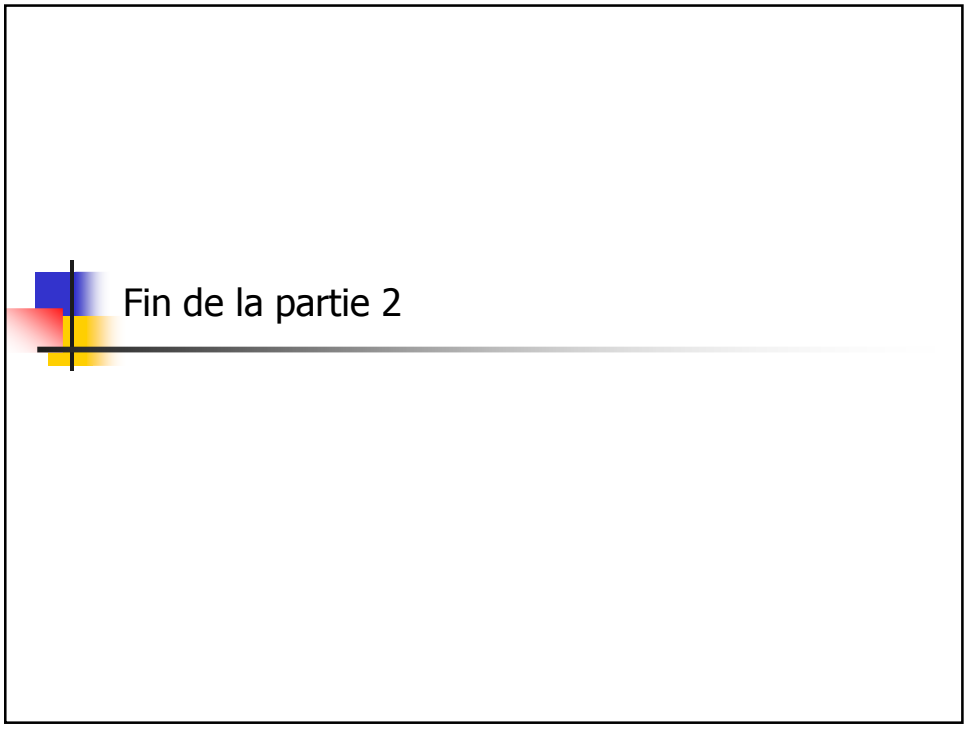
```
public class Servlet1 extends HttpServlet {
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    res.setContentType("text/html");  PrintWriter out = res.getWriter();
    out.println("<HTML><BODY>"); out.println("<br>");
    out.println("je suis servlet1, je place un attribut attr1=bonjour dans la requete");
    out.println("<br>inclusion : "); req.setAttribute("attr1", "bonjour");
    RequestDispatcher dispat = req.getRequestDispatcher("/maServlet2Path");
    dispat.include(req,res); out.println("<br>");
    out.println("je suis la servlet 1, je modifie l'attribut attr1=bonsoir dans la requete");
    req.setAttribute("attr1", "bonsoir"); out.println("<br>inclusion : ");
    dispat.include(req,res); out.println("<br>");
    out.println("</BODY></HTML>");
}}
```

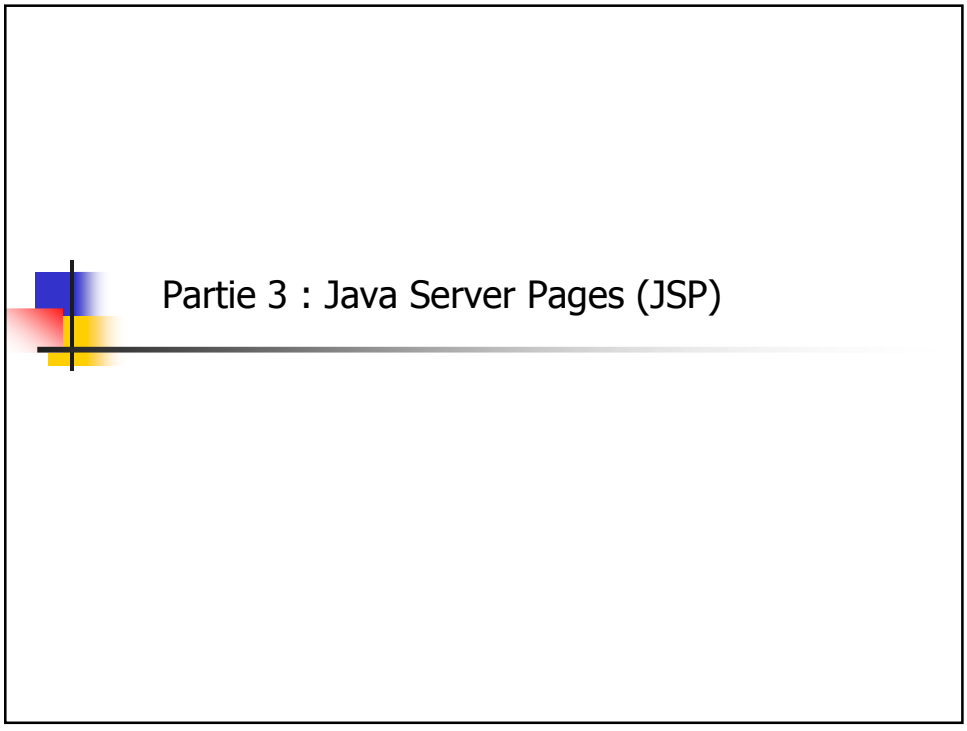

Partage du contrôle (include)

Code pour Servlet2

```
public class Servlet2 extends HttpServlet {
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {

    PrintWriter out = res.getWriter();
    out.println("je suis servlet2 attr1 vaut "+req.getAttribute("attr1"));
}
}
```


Fin de la partie 2


Partie 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>
```

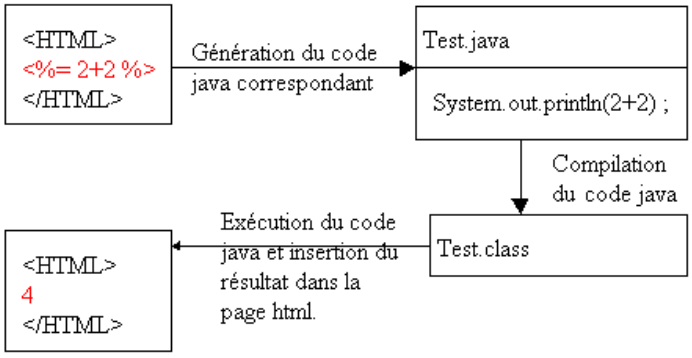
Serveur Web

- 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.
  - **JSWDK** 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 Scriplet
  - 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'entete de la page<br>
```

corps.jsp

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

piedpage.html

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

regroupage.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	Erreurpage.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.
pageEnconding	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

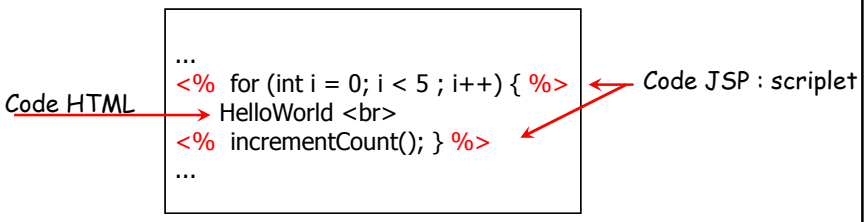
- Une déclaration permet d'insérer du code dans la classe de la Servlet
- Les déclarations sont placés entre les symboles <%! et %>
- Exemple:

```
<%!
   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

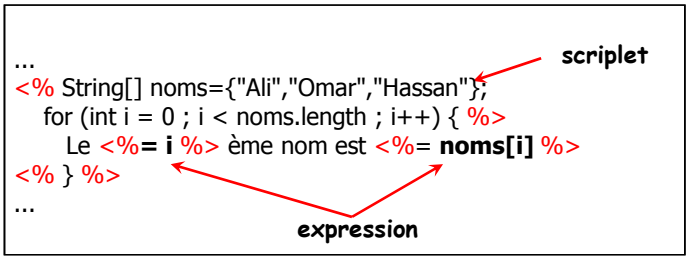
Éléments de scripts JSP : scriplet

- 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 scriplet de la forme *<% out.println(…); %>*
- Se transforme en *out.println("…");* dans la méthode *_jspService(…)* après génération



Eléments de scripts JSP : scriplet 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 : scriplet 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 = 23;} %>
   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 {
    private String nom;
    private String adresse;
    public String getnom () {
       return nom;
    }
    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="Nom" class="Package.class" scope="valeur" />
  - id : nom de l'instance pour identification
  - class : Nom de la classe du bean
  - scope : champ d'existence de l'objet Bean qui peut être :
    - request : Bean valide pour la requête et peut être transmise (forward)
    - page : Bean valide pour la requête sans transmission
    - 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>*
    - *Exemple :* <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 le tag action *<jsp:setProperty>*
  - Exemples :
    - <jsp:setProperty name="référence Bean" property="nom propriété" value="valeur" />
    - <jsp:setProperty name="référence Bean" property="nom propriété" param="nomParametre" />
- Modification de l'ensemble de propriétés suivant les paramètres fournis par la requête
  - 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

- Soit le java bean suivant :

```
package toto;
public class Voiture {
        private int puissance;
        private boolean
est_demarree;
        private double vitesse;


        public void
setDemarree(boolean p)
        {   est_demarree = p;   }
        public boolean
getDemarree()
        {   return est_demarree; }

        public void setVitesse
(double p)
        { vitesse = p; }
        public double getVitesse()
        { return vitesse;   }
```

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:

```
<%@ page language="java" contentType="text/html" %>
<jsp:useBean id="ma_voiture" class=" toto.Voiture">
</jsp:useBean>
<%  ma_voiture.setDemarree(true); ma_voiture.setVitesse(21.2); %>
<html>
<head><title>Page pour lecture d'information</title></head>
<body bgcolor="white">
 La voiture a-t-elle démarré: <%= ma_voiture.getDemarree() %>
<br>La vitesse de la voiture est de :
<jsp:getProperty name="ma_voiture" property="vitesse"/> km/h <br>
 La puissance de la voiture est de :
 <jsp:getProperty name="ma_voiture" property="puissance" /> CV
</body>
</html>
```


Java Beans et JSP : scope

- **Exemple** : affectation et récupération des valeurs d'un Java Bean
- Soit la javabean suivante :

```
public class SimpleName{
    private String name;
    public String getname () {
      return nom;
    }
    public void setName (String nom) {
      name=nom;
    }
}
```

Java Beans et JSP : scope

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

```
<%@ page language="java" contentType="text/html" %>
<jsp:useBean id="mon_bean1" scope="page" class="SimpleName"></jsp:useBean>
<jsp:useBean id="mon_bean2" scope="session" class="SimpleName"></jsp:useBean>
<jsp:useBean id="mon_bean3" scope="application" class="SimpleName"></jsp:useBean>
<jsp:setProperty name="mon_bean1" property="name" value="page"/>
<jsp:setProperty name="mon_bean2" property="name" value="session"/>
<jsp:setProperty name="mon_bean3" property="name" value="application"/>
<html>
<head><title> Utilisation du bean </title></head>
<body bgcolor="white">
Avant<br>
mon_bean1 = <%= mon_bean1.getName() %><br>
mon_bean2 = <%= mon_bean2.getName() %><br>
mon_bean3 = <%= mon_bean3.getName() %><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.

```
<%@ page language="java" contentType="text/html" %>
<jsp:useBean id="mon_bean1" scope="page" class="SimpleName">
    </jsp:useBean>
<jsp:useBean id="mon_bean2" scope="session" class="SimpleName">
    </jsp:useBean>
<jsp:useBean id="mon_bean3" scope="application" class="SimpleName">
    </jsp:useBean>
<html><head><title> Récupération du bean </title></head>
<body bgcolor="white">
Après<br>
mon_bean1 = <jsp:getProperty name="mon_bean1" property="name"/><br>
mon_bean2 = <jsp:getProperty name="mon_bean2" property="name"/><br>
mon_bean3 = <jsp:getProperty name="mon_bean3" property="name"/><br>
</body>
</html>
```

Java Beans et JSP : scope

- Les résultats de ces deux jsp est comme suit :
  - Avant
    - mon_bean1 = page
    - mon_bean2 = session
    - mon_bean1 = application
  - Après
    - mon_bean1 =null
    - mon_bean2 = session
    - mon_bean1 = 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); %>
```


Fin de la partie 3

Partie 4 : Accès aux bases de données avec JDBC

Introduction

- Dans la plateforme J2EE, l'API chargé de communiquer avec les bases de données est l'API JDBC
- JDBC est une abréviation de Java DataBase Connector
- JDBC présente un modèle fonctionnel riche pour accéder aux bases de données.
- JDBC peut être utilisé pour accéder à une base de données à partir de:
  - Simple application Java
  - Une servlet
  - Page JSP
  - ...

Travail avec une base de données

- Nous avons choisi de travailler avec la base de données MySQL.
- Après le téléchargement de MySQL et son installation, elle sera à l'écoute du port 3306.
- Pour accéder à la base de données à partir du code Java il faut télécharger un pilote JDBC pour cette base de données.
- Ces pilotes sont appelés Connecteurs.
- Pour MySQL ce connecteur se nomme mysql-connector-java-5.0.4-bin.jar.
- Ce connecteur doit être placé dans le sous-répertoire lib de l'application.

Classes de l'API JDBC

- Les classes de l'API JDBC figurent dans le package java.sql : `import java.sql`
- Les classes les plus usuelles sont les suivantes:
  - **DriverManager** : charge et configure le driver de la base de données.
  - **Connection** : réalise la connexion et l'authentification à la base de données.
  - **Statement** (et PreparedStatement) : contient la requête SQL et la transmet à la base de données.
  - **ResultSet** : permet de parcourir les informations retournées par la base de données dans le cas d'une sélection de données

La connexion à une base de données

- Le 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.
  - 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)

La connexion à une base de données

- L'établissement de la connexion
  - Pour se connecter à une base de données, il faut instancier un objet de la classe Connection en lui passant :
    - l'URL de la base à accéder.
    - Le login
    - Le mot de passe
  - Exemple :
    - Dans notre cas (avec MySQL) cette instruction est comme suit :
      - String url="jdbc:mysql://localhost/mydb";
      - Connection con;
      - con=DriverManager.getConnection(url,"root","monPwd");

La connexion à une base de données

- Accéder à la base de données :
  - Après l'établissement de la connexion, il est possible d'exécuter des ordres SQL.
  - Il est possible aussi d'obtenir des informations sur la base de données grâce aux objets suivants :
    - DatabaseMetaData : informations à propos de la base de données : nom des tables, index, version ...
    - ResultSet : résultat d'une requête et information sur une table. L'accès se fait enregistrement par enregistrement.
    - ResultSetMetaData : informations sur les colonnes (nom et type) d'un ResultSet

La connexion à une base de données

- L'exécution de requêtes SQL:
  - Les requêtes d'interrogation SQL sont exécutées avec les méthodes d'un objet Statement que l'on obtient à partir d'un objet Connection
  - Un objet de la classe Statement permet d'envoyer des requêtes SQL à la base.
  - Pour exécuter une requête de :
    - sélection : il faut utiliser la méthode executeQuery()
    - Insertion, suppression, mise à jour : il faut utiliser la méthode executeUpdate() qui, en plus, retourne le nombre d'enregistrements qui ont été mis à jour

Exemple 1 :

```
String requete="SELECT * FROM client"
ResultSet resultats = null;
Statement stmt = con.createStatement();
resultats = stmt.executeQuery(requete);
```

Exemple 2 :

```
String requete="INSERT INTO client VALUES
(4,'client 4','prenom 4')"
Statement stmt = con.createStatement();
int nbMaj = stmt.executeUpdate(requete);
  affiche("nb mise a jour = "+nbMaj););
```

Remarque: Utilisez PreparedStatement à la place de Statement

La connexion à une base de données

- Remarque : Différence entre Statement et PreparedStatement ?
  - L'interface **PreparedStatement étend Statement** et représente une instruction paramétrée.
  - Cette interface diffère de Statement sur deux points principaux :
    - Les instances de PreparedStatement contiennent une **instruction SQL déjà compilée** (d'où le terme *prepared)*. Cela améliore notamment les performances si cette instruction doit être appelée de nombreuses fois.
    - Les instructions SQL des instances de PreparedStatement contiennent **un ou plusieurs paramètres d'entrée**, non spécifiés lors de la création de l'instruction. Ces paramètres sont représentés par des points d'interrogation(?). Ces paramètres doivent être spécifiés avant l'exécution.
  - L'exécution des PreparedStatement est identique à celle des simples Statement, à la différence près qu'il n'y a pas d'argument aux méthodes executeQuery() et executeUpdate().
  - Exemple :

```
PreparedStatement recherchePersonne = con.prepareStatement("SELECT * FROM
    personnes WHERE nom_personne = ?");
recherchePersonne.setString(1, "Soufiane");
resultats = recherchePersonne.executeQuery();
...
```


La connexion à une base de données

- Résultat d'une requête de sélection:
  - Une requête de sélection retourne un objet dont le type est ResultSet
  - ResultSet est une abstraction d'une table qui se compose de plusieurs enregistrements constitués de colonnes qui contiennent les données.
  - Les principales méthodes pour obtenir des données sont :
    - **getInt(int/String)** : retourne le contenu de la colonne dont le numéro (resp. le nom) est passé en paramètre sous forme d'entier.
    - **getFloat(int/String)** : retourne le contenu de la colonne dont le numéro (resp. le nom) est passé en paramètre sous forme de nombre flottant.
    - **getDate(int/String)** : retourne le contenu de la colonne dont le numéro (resp. le nom) est passé en paramètre sous forme de date.
    - **next()** : se déplace sur le prochain enregistrement : retourne false si la fin est atteinte. Le curseur pointe initialement juste avant le premier enregistrement.
    - **Close()** : ferme le ResultSet
    - **getMetaData()** : retourne un objet ResultSetMetaData associé au ResultSet qui permet d'obtenir des informations sur le résultat de la requête.

La connexion à une base de données

- Parcours des données retournées :
  - Les résultat d'une requête de sélection figure dans un objet ResultSet qu'il faudra parcourir pour en extraire les données.
  - Comme le curseur pointe initialement juste avant le premier enregistrement, il est nécessaire de faire un premier appel à la méthode next() pour se placer sur le premier enregistrement.
  - Exemple :

```
try {
ResultSetMetaData rsmd = resultats.getMetaData();
int nbCols = rsmd.getColumnCount();
boolean encore = resultats.next();
while (encore) {
    for (int i = 1; i <= nbCols; i++)
    System.out.println(résultats.getString(i) + " ");
    encore = résultats.next();
    }
résultats.close();
}
catch (SQLException e) { //traitement de l'exception }
```


Exemple TP

1. Création de la base de données :
   - En mode console créer une base de données MySQL qui se nomme maBdd.
     - **mysql**> **CREATE DATABASE** maBdd;
   - Créer ensuite une table qui se nomme personnes et qui est composés des champs id, nom, prenom et age.
     - **mysql**> **USE** maBdd
     - **mysql**>create table personnes(id int not null primary key auto_increment, nom varchar(25), prenom varchar(25), age tinyint);
     - **mysql**>**SHOW TABLES**;

Exemple TP (suite)

- Création de l'application Web :
  - L'application web démarre avec une page d'accueil (HTML) avec deux liens hypertextes:
    - Le premier lien pointe vers un formulaire HTML qui permet de saisir le nom, le prénom et l'age d'une personne. Ce formulaire est traité par une **Servlet** qui se charge de faire l'insertion dans la base de données.
    - 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.
  - 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.
  - La page JSP à son tour affiche un lien pour revenir à la page d'accueil.

Solution : Fichiers de l'exemple

exemple\accueil.html:

```
<html>
 <head><title>page d'accueil de l'exemple JDBC</title></head>
 <body>
  <center>
    <H3> Bienvenu dans l'application qui montre l'utilisation des bases de
    données dans J2EE </H3>
    <UL>
    <LI> <a href="formulaire.html"> Pour vous inscrire cliquez ici</a>.
    <LI> <a href="affichage.jsp"> Pour consulter la liste des personnes inscrites
    cliquez ici</a>.
    </UL>
  </center>
 </body>
</html>
```

Solution : Fichiers de l'exemple

exemple\formulaire.html

```
<html>
 <head><title> formulaires et servlets </title></head>
 <body>
   <form method="post" action="insertionPath">
     Enregistrement d'un utilisateur
    <table border=0 bgcolor="blue">
     <tr> <td>Nom :</td> <td><input type=text name="nom"></ td >          </tr>
     <tr>  <td>Prénom :</td><td><input type=text name="prenom"></td>  </tr>
     <tr>  <td>Age :</td><td><input type=text name="age"></td>             </tr>
     <tr>  <td align="center"><input type="submit" value="Envoyer"></td>
           <td align="center"><input type="reset" value="Effacer"></td>
    </tr>
    </table>
  </form>
</body>
</html>
```


Solution : Fichiers de l'exemple

exemple\WEB-INF\classes\Insertion.java

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

public class Insertion extends HttpServlet {
     protected void doPost(HttpServletRequest req, HttpServletResponse res)throws
     ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
       try {
          String fnom=req.getParameter("nom");
          String fprenom=req.getParameter("prenom");
          String fage=req.getParameter("age");
         //converion de l'age en int : attention cette instruction doit figurer dans un bloc try/catch
          int intAge=Integer.parseInt(fage);
          String url="jdbc:mysql://localhost/maBdd";
          String driver = "com.mysql.jdbc.Driver";
          Class.forName(driver).newInstance();
          Connection con;
```

Solution : Fichiers de l'exemple

```
        con=DriverManager.getConnection(url,"root","toto");
        PreparedStatement  stmt=con.prepareStatement("insert into
            personnes(nom,prenom,age) values (?,?,?)");
        stmt.setString(1,fnom);
        stmt.setString(2,fprenom);
        stmt.setInt(3,intAge);
        stmt.executeUpdate();
        stmt.close();
        con.close();
        out.println("<font color=\"red\"> insertion réussie </font><br>");
        out.println("<a href=\"accueil.html\"> retour à la page d'accueil </a>");
    }
    catch (Exception e){
        out.println("Erreur : "+e.getMessage()+ " source : "+e.getStackTrace());
    }
}
}
```


Solution : Fichiers de l'exemple

exemple\WEB-INF\web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
     app_2_4.xsd"
   version="2.4">
   <servlet>
     <servlet-name>InsertionServlet</servlet-name>
     <servlet-class>Insertion</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>InsertionServlet</servlet-name>
      <url-pattern>/insertionPath</url-pattern>
   </servlet-mapping>
   <welcome-file-list>
     <welcome-file>accueil.html</welcome-file>
   </welcome-file-list>
</web-app>
```

Solution : Fichiers de l'exemple

exemple\affichage.jsp

```
<%@page errorPage="erreur.jsp" import="java.sql.*"%>
<%   String url="jdbc:mysql://localhost/maBdd";
       String driver = "com.mysql.jdbc.Driver";
      Class.forName(driver).newInstance();
       Connection con=DriverManager.getConnection(url,"root","toto");
    PreparedStatement  stmt=con.prepareStatement("select * from personnes;");
    ResultSet resultats = stmt.executeQuery();
%>
<html>
<head><title>liste des personnes inscrites</title></head>
<body>
```


Solution : Fichiers de l'exemple

```
<table border="1">
 <tr><td>Id</td><td>Nom</td><td>Prénom</td><td>Age</td></tr>
    <% boolean encore = resultats.next();
  while (encore) {  %>
    <tr>
       <td><%=resultats.getInt(1)%></td><td><%=resultats.getString(2) %></td>
       <td><%=resultats.getString(3) %></td><td><%=resultats.getInt(4)%></td>
    </tr>
  <% encore = resultats.next(); } %>
</table>
<%
  resultats.close();
  stmt.close();
  con.close();
%>
  <a href="accueil.html"> retour à la page d'accueil </a>
</body>
</html>
```

Solution : Fichiers de l'exemple

exemple\erreur.jsp

```
<%@ page isErrorPage="true" import="java.io.*" %>
<h1>Erreur : </h1>
<% out.println(exception.toString()); %>
```