Java EE Web

ESGI
Du 23 au 27 avril 2012

Michaël THOMAS

Prérequis

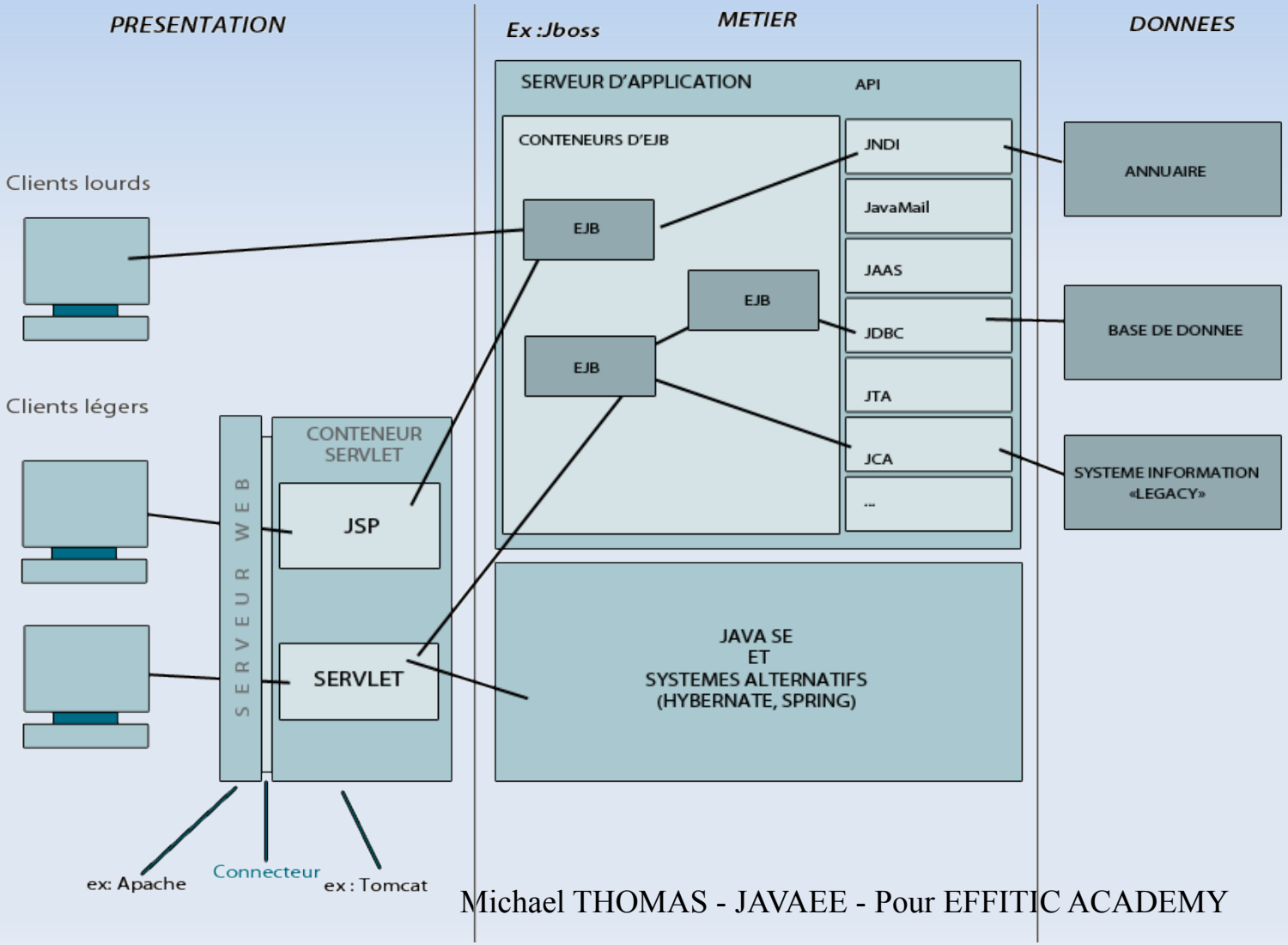
- Paradigme objet
- Base du langage Java
- Approche objet Java : classes, interfaces, packages, droits d'accès, surcharge, redéfinition...
- Gestion des exceptions
- Spécificités et manipulation des collections,
- threads
- Gestion des entrées / sorties (fichier, flux de bytes et de caractères)
- Sérialisation
- Outils de gestion : documentation, packaging

Généralités

- Java EE (Java Enterprise Edition) est une plateforme applicative constituée
  - D'un serveur d'application, c'est à dire un environnement d'éxécution, sur lequel seront exécutées les applications
  - De services au travers d'API thématiques permettant d'offrir en standard un certain nombre de fonctionnalités.

Généralités (2)

- L'architecture de base proposée par Java EE est un modèle de type MVC (Model, Vue, Controlleur) et se décompose comme suit :



Généralirés (3)

- Conteneur de Servlet / Serveurs d'application
  - Toutes les sites web développés en java ne nécessitent pas la mise en place d'un serveur d'application.
  - Le conteneurs, comme Tomcat, permettent de bénéficier de la technologie JSP et SERVLET : idéal pour le développement de petits et de moyens projets nécessitant une mise en oeuvre rapide.
  - Le recours aux serveurs d'application s'avère plutot nécessaire pour de larges projets ou les contraintes non fonctionnelles sont fortes (haute disponibilité, monté en charge, sécurité, hétérogénéité des clients [lourds,légers]...)

Généralités(4)

- **API**, 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).
  - **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.

Généralités (5)

- **API**, Les services d'infrastructure
  - **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 un API définissant des interfaces standard avec un gestionnaire de transactions.
  - **JMX** (Java Management Extension) fournit des extensions permettant de développer des applications web de supervision d'applications.

Généralités (6)

- **API**, 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** est une API permettant la **communication synchrone** entre objets.

Servlet et JSP / Installation de l'environnement

- **Mise en place de l'environnement de développement**
  - Télécharger le binaire tomcat 6.0
  - Télécharger eclipse EE
  - Créer un répertoire local /javaee/votreprenom/tomcat et décompressez y tomcat.
  - Télécharger le plug Eclipse pour tomcat sur :

    http://www.eclipsetotale.com/tomcatPlugin.html
  - Téléchager et décompresser dans le répertoire plugin d'eclipse, fermer eclipse.
  - Créer un répertoire /javaee/votrePrenom/workspace
  - Redémarrer Eclipse et choisissez ce répertoire dans File/SwitchWorspace
  - Configurer (Eclipse/Window/Preferences/Tomcat) le répertoire d'accès à Tomcat , et cocher : dans server.xml, et version 6.x

- **Mise en place de l'environnement de développement (2)**
  - Créer un nouveau projet > Java >Tomcat (HelloWorld)
  - À la racine du projet créer le fichier index.jsp

```
<%= "Hello World, today : " + new java.util.Date() %>
```

  - Redémarrer tomcat à partir d'éclipse
  - Lancer un navigateur :
    - http://localhost:8080/HelloWorld/

Servlet et JSP / Installation de l'environnement

- Mise en place de l'environnement production.
  - Installation VirtualBox Debian
  - Installation de APACHE
  - Installation de TOMCAT
  - Installation de MYSQL Server
  - Installation de ECLIPSE EE
  - Configuration des services et de l'IDE

Servlet et JSP / Installation de l'environnement (2)

- Installation virtualbox debian
  - Télécharger une distribution iso de Debian dans javaee/prenom/virtualbox
  - Installer une Debian virtualisée (nommée Prénom - Java EE)
    - Nouveau, Os : linux, version : debian
    - Taille de base : 512 Mo
    - Nouveau disque, taille dynamique, 8go
      - (stocké dans c:\javaee\prenom\virtualbox)
    - Terminer, Terminer.
  - Modifier l'accès réseau NAT par PONT. Dans les préférences de la VM(Virtual Machine)
  - Lancer la machine virtuelle Java EE

Servlet et JSP / Installation de l'environnement (3)

- Installation virtualbox debian (2)
  - Selectionner le fichier iso précédemment téléchargé.
  - Install (non graphique) (France, France, France)
  - Nom du système : debianjavaee[prenom]
  - Nom du domaine : java.effitic.org
  - Partition : assisté, disque entier, tout dans une seule partition, terminer et appliquer les changements, oui.

Servlet et JSP / Installation de l'environnement (4)

- Installation virtualbox debian (3)
  - Mot de passe root : effitic
  - Nom d'un utilisateur : user, pw : effitic
  - Ne pas analyse d'autre cd ou dvd.
  - Miroir, france, ftp.fr.debian.org
  - Laisser vide le proxy.
  - Ne pas participer à l'étude statistique.
  - Choisir Système standard.
  - Installer grub.
  - Éteindre la machine,

Servlet et JSP / Installation de l'environnement (5)

- Installation apache
  - Lancer le système virtualisé debian (F12 pour la séquence de boot! et/ou supprimer le disque iso dans les params)
  - Se logguer en user/effitic puis **su,** pw effitic
  - ping www.google.fr
  - ping <ip hote> (à partir de la vm)
  - Ping <ip virtual system> à partir de l'hote.
  - Installation de la version multithreadé de apache
    - apt-get install apache2-mpm-worker
    - Valider les demandes apt-get.

Servlet et JSP / Installation de l'environnement (6)

- Installation apache (2)
  - Tester l'installation en local
    - telnet localhost 80
    - Puis taper : GET
    - Un contenu html doit apparaître, il contient : It works!
  - Test l'installation à partir de la machine hôte
    - telnet <ip machine virtuelle> 80
    - Puis taper : GET
    - idem.

Servlet et JSP / Installation de l'environnement (7)

- Installation de Tomcat
  - Tomcat est livré en bundle avec une version de Apache. Bien que cela fonctionne, il est préférable de découpler dès le départ ces deux services, surtout si par la suite on veut load-balancer des instances tomcat à partir d'un même apache (en multi-thread).
  - Installer lynx : apt-get install lynx
  - lynx http://tomcat.apache.org
  - Tomcat 6.x, aller sur le lien : core/tar.gz et appuyez sur la touche 'd'

Servlet et JSP / Installation de l'environnement (8)

- Installation Tomcat (2)
  - Puis aller sur « enregistrer sur le disque », cliquer, puis quitter lynx.
  - tar -xvf apache-tomcat.....
  - mv apache-tomcat.... /etc/tomcat
  - Installation le java JDK sur debian (pas packagé)
    - Installer ssh
      - apt-get install ssh
    - Installer un serveur ftp, puis emacs
      - apt-get install vsftpd
      - apt-get install emacs

Servlet et JSP / Installation de l'environnement (9)

- Installation Tomcat(3)
  - Modifier /etc/vsftpd.conf
    - emacs /etc/vsftpd.conf
    - Activer les lignes :
      - local_enable=YES
      - write_enable=YES
      - chroot_local_user=YES
    - Desactiver : #anonymous_enable=TRUE
    - Sauvegarder : Ctrl + x , quitter : Ctrl + c
    - Relancer /etc/init.d/vsftpd restart
  - Installer un client ftp sur la machine hote : fileZilla par exemple.
  - Télécharger à partir de la machine hôte le jdk java sur http://java.sun.com/javase/downloads/index.jsp
  - Connectez vous à l'aide du du FtpClient sur votre serveur virtuel et transférer le jdk java. (dans votre /home/user)

Servlet et JSP / Installation de l'environnement (10)

- Installation de Tomcat (4)
  - Installation du JDK Java SE.
    - Aller dans le repertoire ou vous avez poser le jdk (/home/user)
    - chmod +x jdk-6u18-linux-i586.bin
    - ./jdk-6u18-linux-i586.bin
    - Accepter la licence.
    - Ajouter au path avec la commande
      - echo 'export PATH=$PATH:/home/user/jdk1.6.0_18/bin' >> /home/user/.bashrc
      - echo 'export JAVA_HOME=/home/user/jdk1.6.0_18' >> /home/user/.bashrc
      - echo 'export JRE_HOME=/home/user/jdk1.6.0_18/jre' >> /home/user/.bashrc
      - echo 'export CLASSPATH=/home/user/jdk1.6.0_18/lib:/usr/lib/java' >> /home/user/.bashrc
  - Exit, puis se relogguer.
  - Vérifier l'installation
    - java –version
    - javac, jar, javadoc

Servlet et JSP / Installation de l'environnement (10)

- Installation tomcat (5)
  - Créer le répertoire /home/user/tomcat
  - Tar -xvf apache-tomcat-6.0.26
  - mv apache-tomcat.... /etc/tomcat
  - /etc/tomcat/bin/catalina.sh start &
  - Téléchager le binaire (.so) du connecteur tomcat 1.2.30 pour le serveur apache 2.2.x
  - À partir de la machine virtuelle :
    - mv mod_jk-1.2.30.....so /usr/lib/apache2/modules/mod_jk.so

Servlet et JSP / Installation de l'environnement (11)

- Installation de Tomcat (6)
  - Éditer : emacs /etc/apache2/apache2.conf

```
LoadModule    jk_module  /usr/lib/apache2/modules/mod_jk.so
JkWorkersFile /etc/apache2/workers.properties
JkShmFile     /var/log/httpd/mod_jk.shm
JkLogFile     /var/log/httpd/mod_jk.log
# Set the jk log level [debug/error/info]
JkLogLevel    info
```

mkdir /var/log/httpd

  - Créer : emacs /etc/apache2/workers.properties

```
worker.list=worker1,worker2,loadbalancer
worker.worker1.type=ajp13
worker.worker1.host=127.0.0.1
worker.worker1.port=8009
worker.worker2.type=ajp13
worker.worker2.host=IP WINDOWS HOTE
worker.worker2.port=8009
worker.loadbalancer.type=lb
worker.loadbalancer.balanced_workers=worker1, worker2
```

Servlet et JSP / Installation de l'environnement (12)

- Installation de Tomcat (7)
  - Editer le fichier /etc/apache2/sites-available/default et ajouter les lignes suivantes :

```
DocumentRoot ....
JkMount / worker1
JkMount /HelloWorld* loadbalancer
```

Redémarrer apache : /etc/init.d/apache2 restart

Les Servlets

- Application java exécuté sur un serveur à l'intérieur d'un conteneur (ex. Tomcat)
- Une servlet est chargée lorsque le serveur est mis en route ou lorsque le premier client fait appel aux services de la servlet.
- Une fois chargée, et contrairement à d'autre langages (comme php), la servlet reste active dans l'attente d'une nouvelle requête.
- Une servlet peut utiliser toutes les ressources du serveur et renvoi des données à un navigateur (html ou flux binaire dynamique (image)).

Les Servlets (2)

- Les servlets utilisent des classes et interfaces issues des packages javax.servlet (servlet indépendantes d'un protocole) et javax.servlet.http (spécifique au protocole http)
- Une servlet doit implémenter l'interface javax.servlet.Servlet ou étendre soit javax.servlet.GenericServlet, soit javax.servlet.http.HttpServlet
- **Une servlet n'a ni méthode main(), ni constructeur**.

Les Servlets (3)

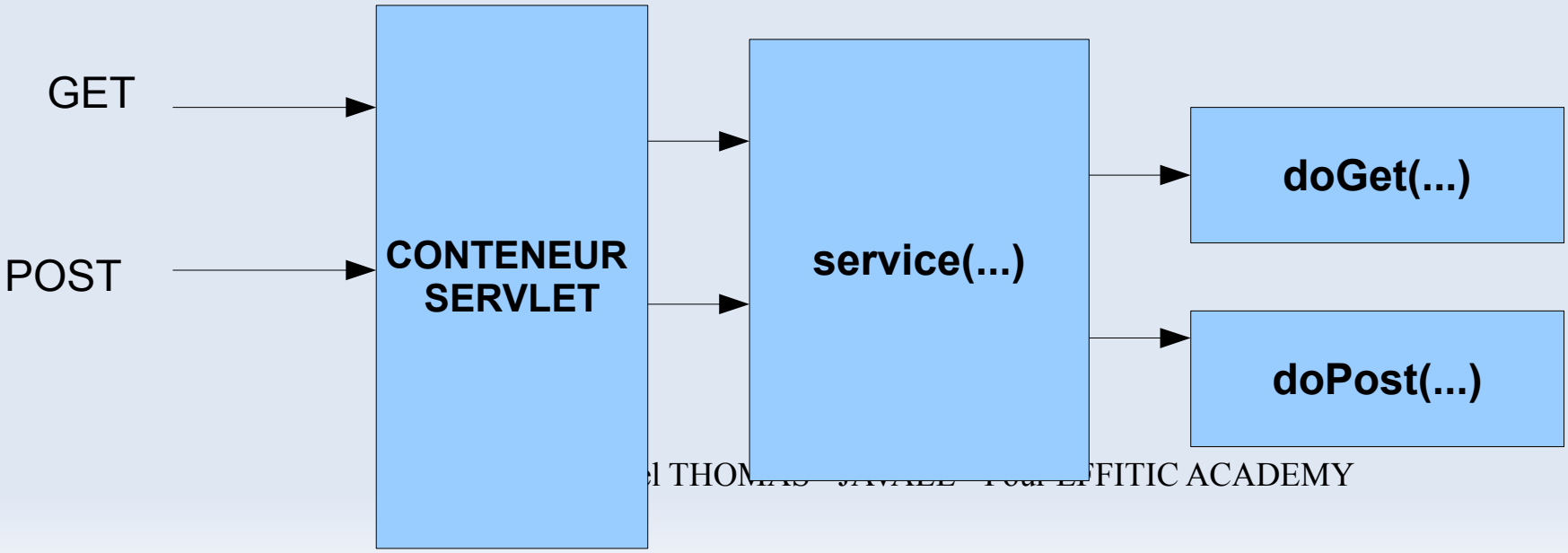
- On initialise une servlet au moyen de 2 méthodes :

```
public void init () throws ServletException
public void init (ServletConfig) throws ServletException
```

- La méthode destroy () permet de libérer les ressources acquises et éventuellement d'écrire des informations persistantes qui pourront être lues au prochain chargement de la servlet par l'une des méthodes init ().

Les Servlets (4)

- **javax.servlet.GenericServlet**
  - Une servlet générique doit redéfinir une méthode service (), méthode abstraite de la classe javax.servlet.GenericServlet.
- **javax.servlet.http.http.HttpServlet**
  - Une servlet http doit redéfinir une méthode doGet() ou doPost() en fonction du type de requête qu'elle aura à traiter.



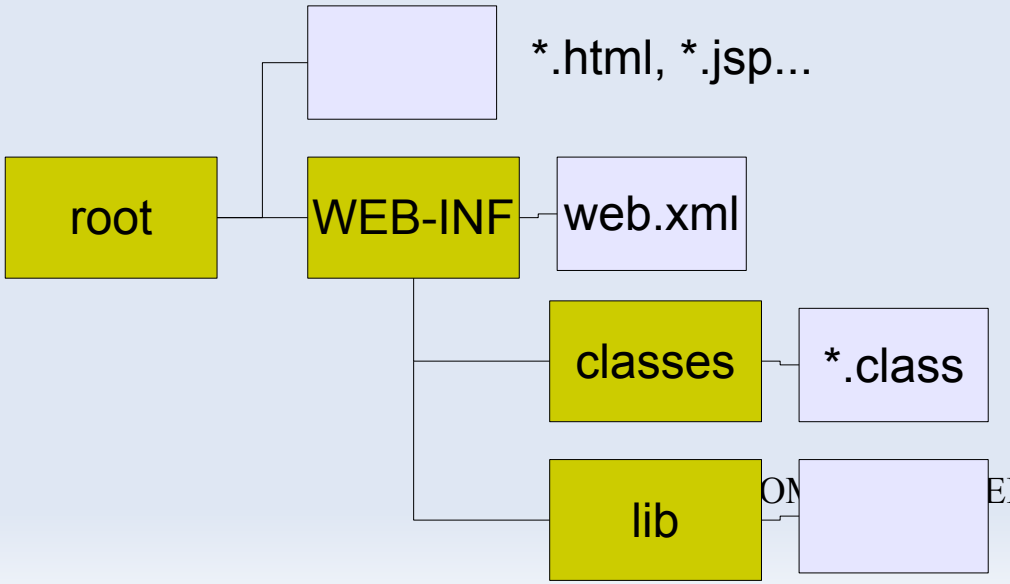
Les Servlets (5)

- HelloWorldServlet.java

```
package org.esgi.servlet.hello;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HelloWorldServlet extends HttpServlet
{
    private static final String CONTENT_TYPE = "text/html";
    public void service(HttpServletRequest request, HttpServletResponse response) throws
     ServletException, IOException
    {
       response.setContentType(CONTENT_TYPE);
       PrintWriter out = response.getWriter();
       out.println("<html><body><h1>Prénom "+System.getProperty("os.name")+"</h1>"
                   +"<p>Hello World Servlet!</p></body></html>");
     }
}
```

Les Servlets (6)

- HelloWorldServlet.java (2) – Déploiement
  - Losqu'une application web est prête à être mise en ligne, on la place dans un fichier d'archive web (extension war)
  - Les fichiers war sont créés avec la commande jar selon une architecture précise.



Les Servlets (7)

- web.xml : descripteur de déploiement.
  - Contient toutes les informations de configuration du fichier d'archive.
  - Dans le cas des servlets, il va permettre de définir la/les classe/s contenant la/les servlet/s, le nom de la servlet, les paramètres éventuels d'initialisation, les chemin virtuel d'accès...

Http://HOTE:PORT/CONTEXT/URL-PATTERN

Les servlets

- WEB-INF/web.xml

```
<web-app>
  <servlet>
    <servlet-name>myFirstServlet</servlet-name>
    <servlet-class>
      org.esgi.servlet.hello.HelloWorldServlet
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>myFirstServlet</servlet-name>
    <url-pattern>/servlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Déploiement de l'application

- En mode invite de commande, allez dans le répertoire du projet HelloWorldServlet

```
jar -cfM michael.war -C . .
```

- Copier le fichier War sur le serveur de production dans le répertoire $TOMCAT/webapps (Windows et dans la virtual machine Linux)

Les Servlets (9)

- web.xml : quelques balises,
  - Paramètres d'initialisation consultable par la méthode getInitParameter(« Nom paramètre »);

```
<init-param>
        <param-name>nom_parametre</param-name>
        <param-value>valeur_parametre</param-value>
        <description> description du parametre </description>
</init-param>
```

  - Gestion des erreurs

```
<error-page>
        <error-code> 404 </error-code>
        <location> /errors/404.html </location>
</error-page>
<error-page>
        <exception-type>javax.servlet.ServletException</exception-type>
        <location> /errors/exception.jsp </location>
</error-page>
```

Les Servlets (10)

- Suivi de session
  - Le protocole TCP/IP est sans état.
  - Le suivi de session peut être simulé grâce à l'utilisation de cookies
    - javax.servlet.http.Cookie
    - Utilisation de la classe javax.servlet.http.HttpSession pour stocker les variables de session.
  - La durée de session peut être définie soit dans le fichier de déploiement (valeur en minute)
  - Soit par HttpSession.setMaxInactiveInterval(int time). Ce délais définit le temps maximum en secondes entre deux requêtes avant que la session n'expire.

Les Servlets (11)

- Suivi de session
  - Quelques méthodes de javax.servlet.http.Cookie
    - Cookie (String name, String Value)
    - String getName ()
    - String getValue ()
    - setValue (String value)
    - setMaxAge (int expiry)
  - Création d'un cookie
    - HttpServletResponse.addCookie(Cookie c)
  - Récupération de cookies
    - Cookie[] HttpServletRequest.getCookies()
  - On préférera utiliser HttpSession plutot que les cookies lorsque c'est possible.

Les Servlets (12)

- javax.servlet.http.HttpSession
  - Portée et durée de vie : session.
  - Création, récupération de la session
    - HttpSession HttpServletRequest.getSession()
    - HttpSession HttpServletRequest.getSession(bool)
  - Gestion de la session
    - Enumeration HttpSession.getAttributeNames();
    - Object getAttribute(String name)
    - setAttribute(String key, Object value)
    - removeAttribute(String name)

Les Servlets (13)

- javax.servlet.http.HttpSession (2) - Exemple

```
public void service(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
	response.setContentType(CONTENT_TYPE);
	HttpSession session = request.getSession() ;
	Integer count = (Integer) session.getAttribute("count") ;
	if (null == count) count = new Integer (1) ;
	else count = new Integer (count.intValue() + 1) ;
	If (5 == count.intValue ()) session.invalidate() ;
	else session.setAttribute("count", count) ;
	PrintWriter out = response.getWriter();
	out.println ("Page vue : " + count) ;
}
```

Les Servlets (14)

- javax.servlet.http.HttpServletRequest
  - Il s'agit de la classe qui transporte l'ensemble des informations la requête (contexte, uri, et données)
  - Principales méthodes
    - getProtocol() // Généralement HTTP/1.1
    - getScheme() // Généralement http
    - getServerName() // en dev, localHost
    - getRemoteAddr() // ip du client
    - GetMethod() // Get ou POST ...
    - getParameter(String param) // Récupérer un paramètre
    - GetParameterNames() // Liste des paramètres
    - getHeaderNames()

Exercice : Ecrire une Servlet RequestInfo qui affiche toutes les données disponibles à l'intérieur de l'objet Request

Les Servlets (15)

- javax.servlet.http.HttpServletResponse
  - Objet utilisé pour construire un message de réponse HTTP renvoyé au client, il contient :
    - Les méthodes nécessaire pour définir le type de contenu, entête et code de retour
    - Un flot de sortie pour envoyer les données (html ou binaire(image dynamique))
  - void setStatus(int) // définit le code retour
  - void setContentType(String) // Définit le MIME
  - ServletOutputStream getOutputStream() // Flux binaire
  - PrintWriter getWriter() // Flux de char
  - Void sendRedirect(String) // redirection du navigateur vers une autre url.

Les Servlets (16)

- Exercice : Téléchargement.
  - Ecrire une servlet qui renvoi un flux binaire (Byte)
  - Vous utiliserez le type mime : application/octet-stream
  - Vous initialiserez la section header avec les paramètres suivants

    "Content-Disposition","attachment;filename=toto.txt"
  - Utilisez un fichier pdf ou une image pour rendre l'exercice visuel.

Les Servlets (17)

```
public class DownloadServlet extends HttpServlet {
      protected void service(HttpServletRequest request,
                                    HttpServletResponse response) throws IOException {
            try {
                  InputStream is = new  FileInputStream
                    ("/media/datas/FORMATION/java/support_formation_java_se_1.6.pdf");
                  OutputStream os = response.getOutputStream();
                  response.setContentType("application/octet-stream");
                  response.setHeader("Content-Disposition",
                                              "attachment;filename=formation.pdf");
                  int count = 0;
                  byte buf[] = new byte[4096];
                  While (-1 != (count = is.read(buf)))
                        os.write(buf,0,count);
                  is.close();
                  os.close();
            } catch (Exception e) {       }
      }
}
```

Les Servlets (19)

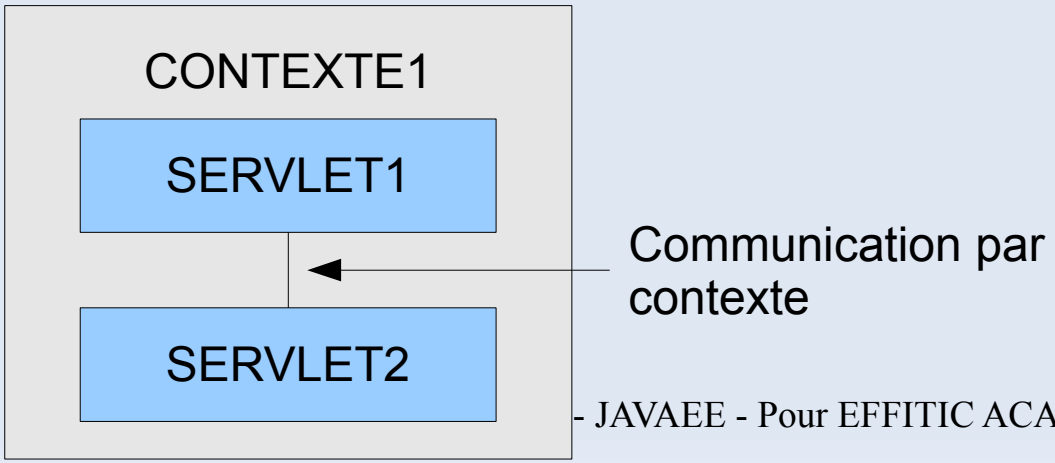
- Technique pour imposer le rafraichissement du client sans utiliser de javascript

```
public class PullClientServlet extends HttpServlet {
        private int count = 10;
        protected void service(HttpServletRequest req, HttpServletResponse resp)
                        throws ServletException, IOException {
            resp.setContentType("text/plain");
            PrintWriter out = resp.getWriter();
            if (count > 0) {
                resp.setHeader("Refresh","1"); // recharge toutes les secondes.
                count--;
                out.println(count + "...");
            } else
                out.println("Fin");
        }
}
```


Exercice : adapter ce code pour qu'il rafraichir 10 fois chaque navigateur client.

Les Servlets (19)

- Collaboration entre Servlets
  - Les Servlets exécutées sur un même serveur peuvent collaborées entre elles :
    - Par partage d'information
      - À travers un conteneur externe (Base de donnée)
      - À travers l'utilisation des contextes :
        ServletContext getServletsContext()



Les Servlets (20)

- Collaboration entre Servlets (2)
  - Par partage d'information (2)

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

public class ContextCommunication1 extends HttpServlet
{

        public void service (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
        {
                PrintWriter out = response.getWriter () ;
                response.setContentType("text/plain") ;
                ServletContext contexte = this.getServletContext() ;
                contexte.setAttribute ("chaine1", "chaine deposee par servlet1") ;
                out.println ("la chaine est deposee") ;
        }
}
```

Les Servlets (21)

- Collaboration entre Servlets (3)
  - Par partage d'information (3)

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

public class ContextCommunication2 extends HttpServlet
{
	public void service(HttpServletRequest request, HttpServletResponse response)
	throws ServletException, IOException
	{
		response.getWriter ().println (this.getServletContext() .getAttribute ("chaine1")) ;
	}
}
```

Les Servlets (22)

- Collaboration entre Servlets (4)
  - Par partage d'information (4)

```
<web-app>
   <display-name> Collaboration servlets </display-name>
   <description> Collaboration de servlets </description>
   <servlet>
          <servlet-name> Servlet1 </servlet-name>
          <servlet-class>ContextCommunication1 </servlet-class>
   </servlet>
   <servlet>
          <servlet-name> Servlet2 </servlet-name>
          <servlet-class> ContextCommunication2 </servlet-class>
   </servlet>
   <servlet-mapping>
          <servlet-name> Servlet1 </servlet-name>
          <url-pattern> /Servlet1 </url-pattern>
   </servlet-mapping>
   <servlet-mapping>
          <servlet-name> Servlet2 </servlet-name>
          <url-pattern> /Servlet2 </url-pattern>
   </servlet-mapping>
</web-app>
```

Les Servlets (23)

- Collaboration entre Servlets (5)
  - Par partage du contrôle
    - Les servlets peuvent partager ou distribuer le contrôle d'une requête grâce à l'interface javax.servlet.RequestDispatcher
      - **Par renvoi** : une servlet peut renvoyer une requête entière à une autre servlet, une page jsp, ou html :
        - Void forward(ServletRequest req, ServletResponse res)
      - **Par inclusion** : une servlets peut inclure du contenu généré.

Les Servlets (24)

- Collaboration entre Servlets (6)
  - Par partage de contrôle (2), **renvoi/dispatch**.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.* ;
import javax.servlet.http.*;
public class ControlCommunication1 extends HttpServlet
{
	public void service (HttpServletRequest request, HttpServletResponse response) throws IOException
	{
		PrintWriter out = response.getWriter();
		out.println("--Servlet1--");
		response.setContentType("text/plain") ;
		request.setAttribute("chaine1","argument1") ; // on transmet un objet
		try {
			RequestDispatcher dispat = // on transmet une chaine
				request.getRequestDispatcher("/process.jsp?chaine2=argument2") ;
			dispat.forward(request, response) ;
		} catch (Exception e) {}
		out.println("--Servlet1--");
		System.out.println("Action executed");
	}
}
```

Les Servlets (25)

- Collaboration entre Serlvets (7)
  - Par partage de contrôle (3), **renvoi/dispatch**

```
process.jsp
<html><body>
<p><%= request.getAttribute ("chaine1") %></p>
<p><%= request.getParameter ("chaine2") %></p>
</body></html>
```

Les Servlets (26)

- Collaboration entre Servlets (8)
  - Par partage de contrôle (4) - **inclusion/include**

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.* ;
import javax.servlet.http.*;
public class ControlCommunication2 extends HttpServlet
{
	public void service(HttpServletRequest request, HttpServletResponse response) throws IOException
	{
		PrintWriter out = response.getWriter();
		out.println("--Servlet1--");
		response.setContentType("text/plain") ;
		request.setAttribute("chaine1","argument1") ; // on transmet un objet
		try {
			RequestDispatcher dispat = // on transmet une chaine
				request.getRequestDispatcher("/process.jsp?chaine2=argument2") ;
			dispat.include(request, response) ;
		} catch (Exception e) {}
		out.println("--Servlet1--");
		System.out.println("Action executed");
	}
}
```

Les Servlets (27)

- Exploitation d'un formulaire html : <form>
  - Une balise form définie la méthode à utiliser pour envoyer dans son attribut METHOD="[GET|POST]"
  - Son attribut ACTION permet de déterminer à quelle url seront envoyée les données.
  -

```
<form method="POST" action="http://localhost:8080/formulaire/action">
<table>
  <tr><td>Login</td><td><input type="text" name="login"/></td></tr>
  <tr><td>Password</td><td><input type="password" name="password"/></td></tr>
  <tr><td>Connection auto</td>
                <td>
                    <input type="radio" name="connect" value="true"/> Oui
                    <input type="radio" name="connect" value="false"/> Non
                </td></tr>
 <tr><td></td><td><input type="submit"/></td></tr>
</table>
</form>
```

Les Servlets (28)

- Exercice :
  - Créer une servlet Action qui récupère et affiche les données du précédent formation.
  - Le formulaire sera mis dans le fichier index.html à la racine du projet.

Les Servlets (29)

- Afficher les données d'un formulaire

```
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws IOException {
      PrintWriter out = response.getWriter();
      out.println("Données reçues du formulaire");
      for (Object s  : request.getParameterMap().keySet() )
            out.println(s.toString()+" --> "+(request.getParameter(s.toString())));

}
```

Les Servlets (30)

- Le cas de l'upload et des formulaires utilisant l'enctype : multipart/form-data
  - Le JDK Java 1.6 n'offre pas de méthodes faciles pour gérer l'accès à ce type d'encodage.
  - Pour cela, il est nécessaire de télécharger un composant supplémentaire fourni par apache :
  - http://commons.apache.org/fileupload/
  - Et sa dépendance : http://commons.apache.org/io/
  - Créer un nouveau projet tomcat, le nommer Upload
  - Copier les précédents jar dans le répertoire lib de votre WEB-INF/lib. **Runtime**
  - Ajouter les également dans le buildpath de votre projet. (Btn droit projet / buildpath / ajout jar) **compile**

Les Servlets (31)

- Le cas de l'upload et des formulaires utilisant l'enctype : multipart/form-data (2)
  - Pour savoir si le formulaire est multipart-form/data

```
// Check that we have a file upload request
boolean isMultipart = ServletFileUpload.isMultipartContent(request);
```

  - Pour recuperer le contenu du formulaire (champs simple et champs binaires)

```
// Create a factory for disk-based file items
FileItemFactory factory = new DiskFileItemFactory();
// Create a new file upload handler
ServletFileUpload upload = new ServletFileUpload(factory);
// Parse the request
List /* FileItem */ items = upload.parseRequest(request);
```

Les Servlets (32)

- Le cas de l'upload et des formulaires utilisant l'enctype : multipart/form-data (3) – **Plus de contrôle !**

```
// Create a factory for disk-based file items
DiskFileItemFactory factory = new DiskFileItemFactory();
// Set factory constraints
factory.setSizeThreshold(yourMaxMemorySize);
factory.setRepository(yourTempDirectory);
// Create a new file upload handler
ServletFileUpload upload = new ServletFileUpload(factory);
// Parse the request
List /* FileItem */ items = upload.parseRequest(request);
```

Les Servlets (33)

- Le cas de l'upload et des formulaires utilisant l'enctype : multipart/form-data (4) – **Parcours des éléments**

```
Iterator iter = items.iterator();
while (iter.hasNext()) {
    FileItem item = (FileItem) iter.next();

    if (item.isFormField()) {
        out.println(« champs traditionnel - « +item.getFormField()+ » : « +item.getString());
    } else {
        processUploadedFile(item);
    }
}
```

Les Servlets (34)

- Le cas de l'upload et des formulaires utilisant l'enctype : multipart/form-data (5) – **Récupération d'un champs normal**

```
// Process a regular form field
if (item.isFormField()) {
    String name = item.getFieldName();
    String value = item.getString();
    ...
}
```

Les Servlets (35)

- Le cas de l'upload et des formulaires utilisant l'enctype : multipart/form-data (5) – **Récupération de fichiers**

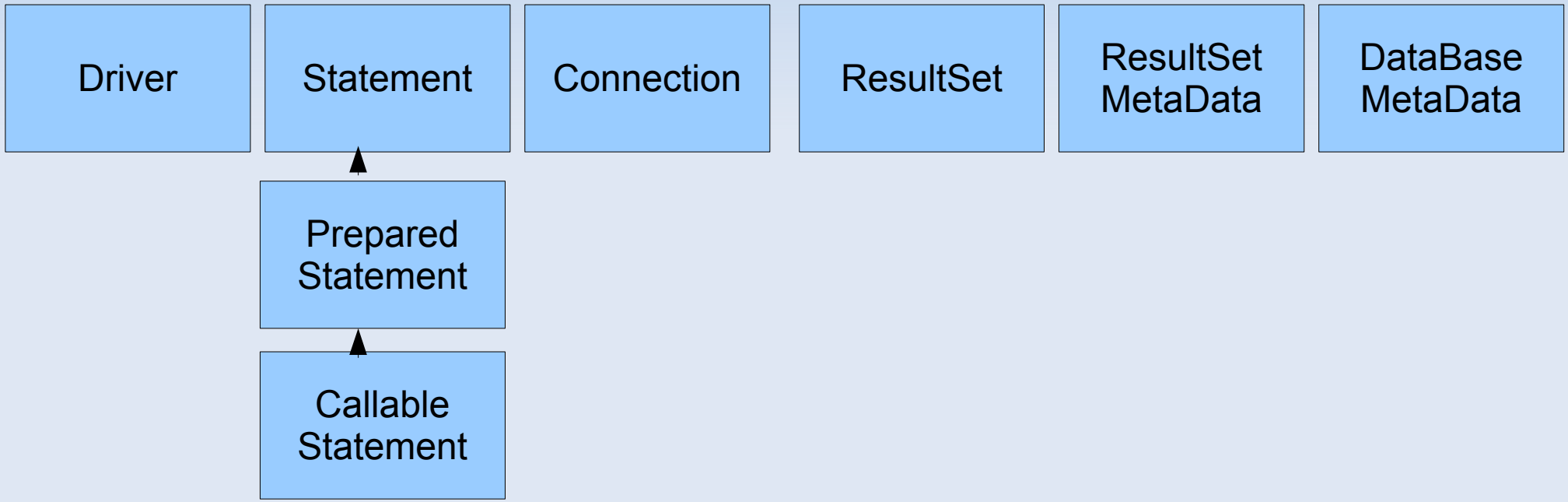
```
if (!item.isFormField()) {
    String fieldName = item.getFieldName();
    String fileName = item.getName();
    String contentType = item.getContentType();
    boolean isInMemory = item.isInMemory();
    long sizeInBytes = item.getSize();
    if (writeToFile) {
    File uploadedFile = new File(...);
    item.write(uploadedFile);
} else {
    InputStream uploadedStream = item.getInputStream();
    ...
    uploadedStream.close();
}
}
```

Java → Base de donnée

- **JDBC : Java DataBase Connectivity**
  - C'est une API java qui permet aux applications de communiquer avec les gestionnaires de base de données.
  - Etant donnée que c'est une interface, les applications sont indépendantes de la base de données (en théorie, en réalité cela dépends de la qualité de l'implémentation de l'interface)
  - Un pilote JDBC permet
    - D'établir une connection avec une base de données.
    - D'envoyer des requêtes sql
    - De traiter les résultats.

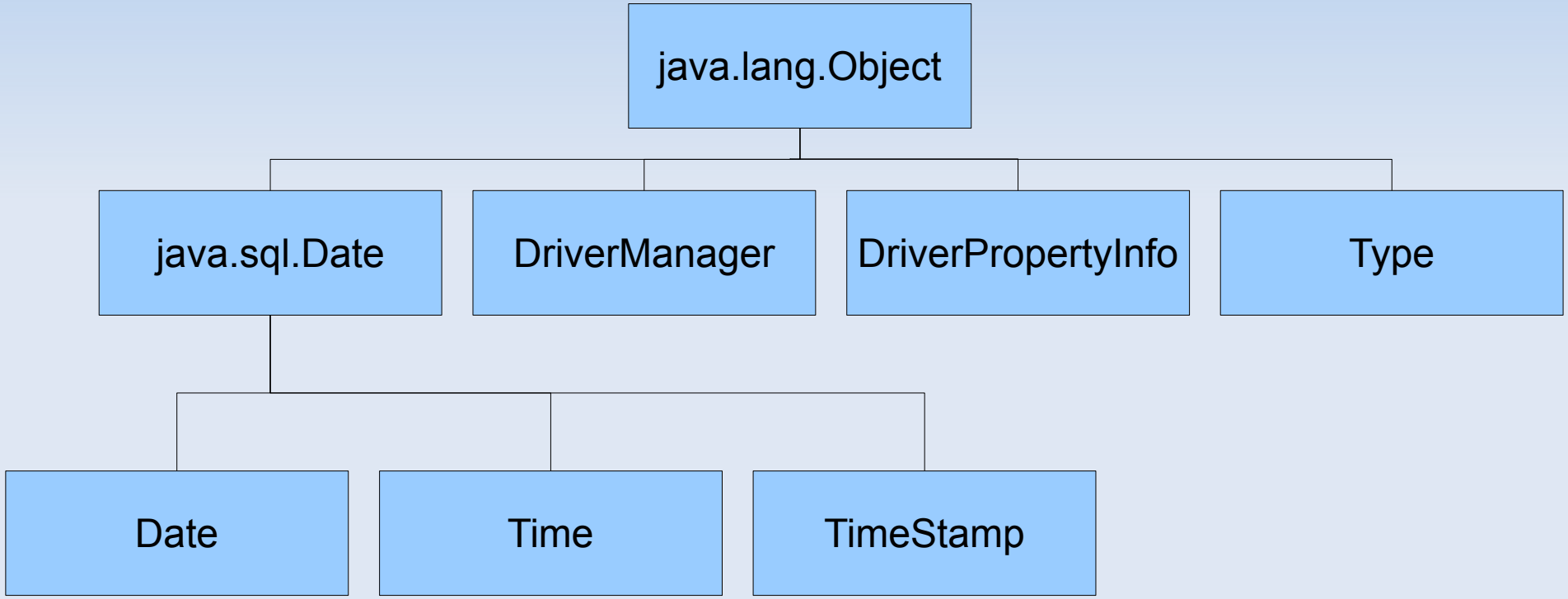
Java → Base de donnée (2)

- Les principales interfaces du package java.sql



Java → Base de donnée (3)

- Organisation des classes liées à JDBC



Java → Base de donnée (3)

- Installation du serveur MySQL
  - Télécharger la version no-install
  - Décompressez la dans un répertoire.
  - Lancer le serveur avec la commande
    - $MYSQL/bin/mysqld
  - Télécharger les outils mysql : Mysql Gui Tools également en version no-install.
  - Lancer MySql Administrator (host : localhost, login : root, sans mot de passe), puis changer le mot de passe root en utilisant l'interface graphique.
  - Attention : il se peut que votre antivirus bloque le port 3306, débloquez le (ou arrêter complètement votre antivirus)

Java → Base de donnée (4)

- Installation du drivers JDBC
  - Pour accéder à une base de donnée il est nécessaire de disposer d'une pilote spécifique à la base de donnée.
  - Un pilote JDBC peut être commercial ou gratuit selon la base de donnée.
  - Pour connaître la liste des pilotes officiels :

    http://industry.java.sun.com/products/jdbc/drivers
  - *Céer un nouveau projet tomcat : Jdbc et ajouter cette nouvelle librairie dans votre buildpath (comme vous l'avez fait précédemment avec Multipart)*

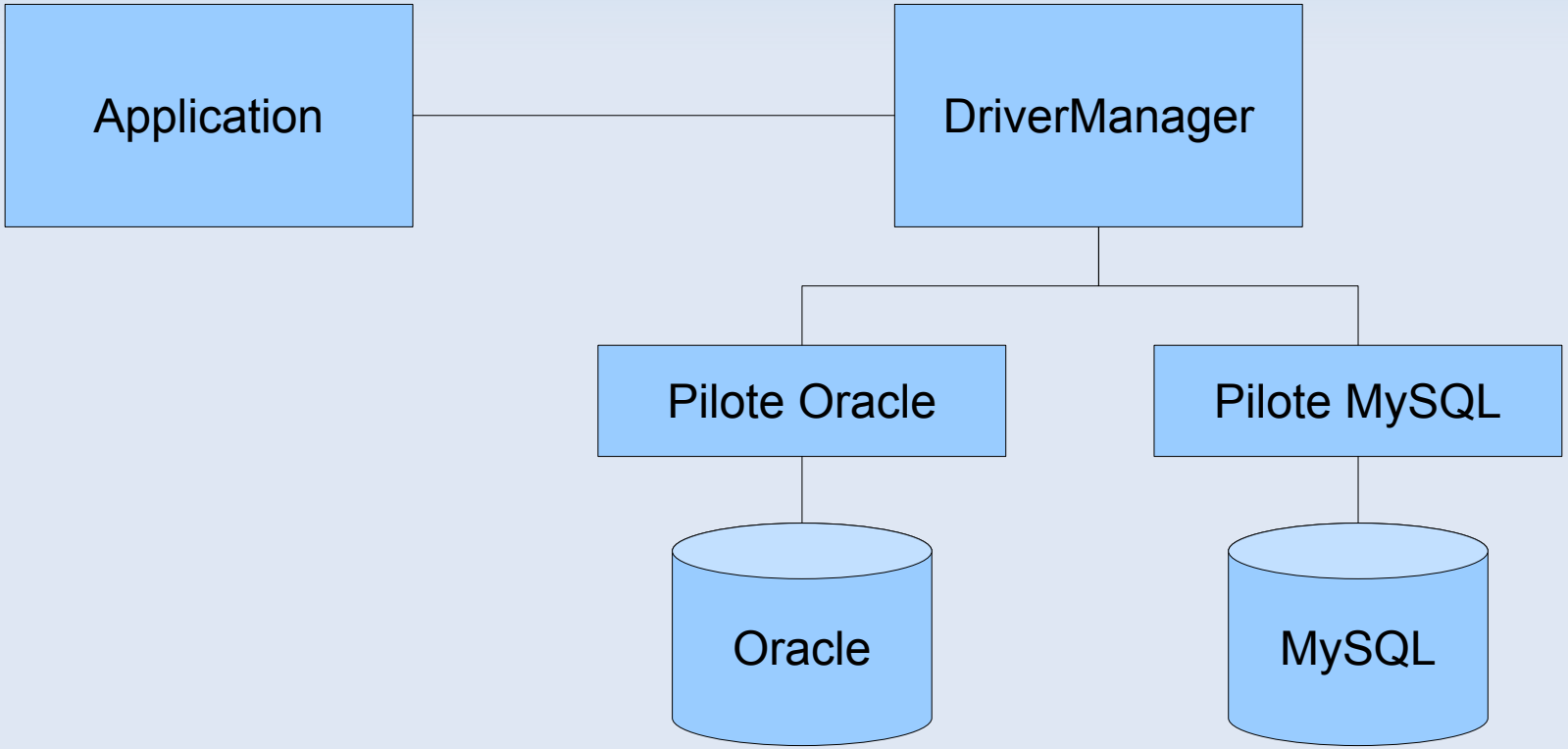
Java → Base de donnée (5)

- 4 Types de pilotes JDBC

TYPE 1	Pilotes accédant aux bases de données grâce à une technologie de ponts. Exemple: le pont ODBC. Cela requiert en général d'installer du code natif sur le poste client.
TYPE2	Le code Java appellent les méthodes C/C++ natives livrées par les éditeurs de base de données. Cela requiert d'installer du code natif sur le poste client.
TYPE 3	Ces pilotes fournissent au client une API générique. Le pilote JDBC sur le client communique au moyen de sockets avec une application intermédiaire sur le serveur qui convertit les requêtes du client en appel API spécifique du pilote souhaité.
TYPE 4	Via des sockets java, ces pilotes interagissent directement avec le gestionnaire de la base de données.

Java → Base de donnée (6)

- Connection à la base de donnée



Java → Base de donnée (7)

- Classes de connection

java.sql.Driver	Interface devant être implémentée par les classes de chargement des pilotes JDBC.
java.sql.DriverManager	Un objet DriverManager va tenter de localiser le pilote JDBC et charger les classes correspondantes.
java.sqlConnection	Un objet Connection représente le lien entre l'application et la base de données. Toutes requêtes SQL transmises et le retour des résultats s'effectueront à travers cet objet.

Java → Base de donnée (8)

- Exemple de connection.
  - Télécharger le pilote Mysql et mettre le .jar dans votre classpath ou le répertoire lib de votre installation java. Ajouter le jar dans le buildpath de votre projet.

```
Private Connection connection ; ( à mettre en dessous de la déclaration de classe
en tant qu'attribut )
public void init(){
   try { Class.forName("org.gjt.mm.mysql.Driver").newInstance();   }
   catch (Exception e) { e.printStackTrace();     }
  try {   this.connection, = DriverManager.getConnection("jdbc:mysql://localhost/effitic",
                "effitic","effitic") ;    }
 catch (SQLException e)    { e.printStackTrace();}
}
```

Java → Base de donnée (9)

- Classes d'accès à la base de donnée

java.sql.Statement	Classe à utiliser pour les requêtes SQL élémentaires. Quelques méthodes: - public ResultSet executeQuery (String sql) throws SQLException java.sql.Statement - public int executeUpdate (String sql) throws SQLException - public boolean execute(String sql) throws SQLException
java.sql.ResultSet	Une instance de cette classe contient une rangée de données extraite de la base par une requête SQL et offre plusieurs méthodes chargées d'en isoler les colonnes. La notation suivante est utilisée: <type> get<type> (int | String) Exemple: String getString (« title ») A un instant donné, un objet ResultSet ne peut contenir plus d'une rangée mais propose une méthode next() permettant de référencer la rangée suivante.
java.sql.Prepared Statement	Cette classe est utilisée pour pouvoir envoyer au gestionnaire de base de données une requête SQL pour interprétation mais non pour exécution.Cette requête peut contenir des paramètres qui seront renseignés ultérieurement.

Java → Base de donnée (10)

- Exemple d'accès à une base de donnée

```
Statement stmt = conn.createStatement () ;
ResultSet rs = stmt.executeQuery (« SELECT a,b,c FROM Table1 ») ;
while (rs.next ())
{
   int x = rs.getInt ("a") ;
   String s = rs.getString ("b") ;
   float f = rs.getFloat ("c") ;
}
```

Java → Base de donnée (11)

- Exemple de code pour des preparedStatement

```
PreparedStatement inst = con.prepareStatement ("UPDATE comptes
    SET solde = ? Where id = ?") ;
for (int i = 0 ; i < comptes.length ; i++)
{
  inst.setFloat (1,comptes [i].extraitSolde ()) ;
  inst.setFloat (2,comptes [i].extraitIdf ()) ;
  inst.execute ()
}
```

Java → Base de donnée (12)

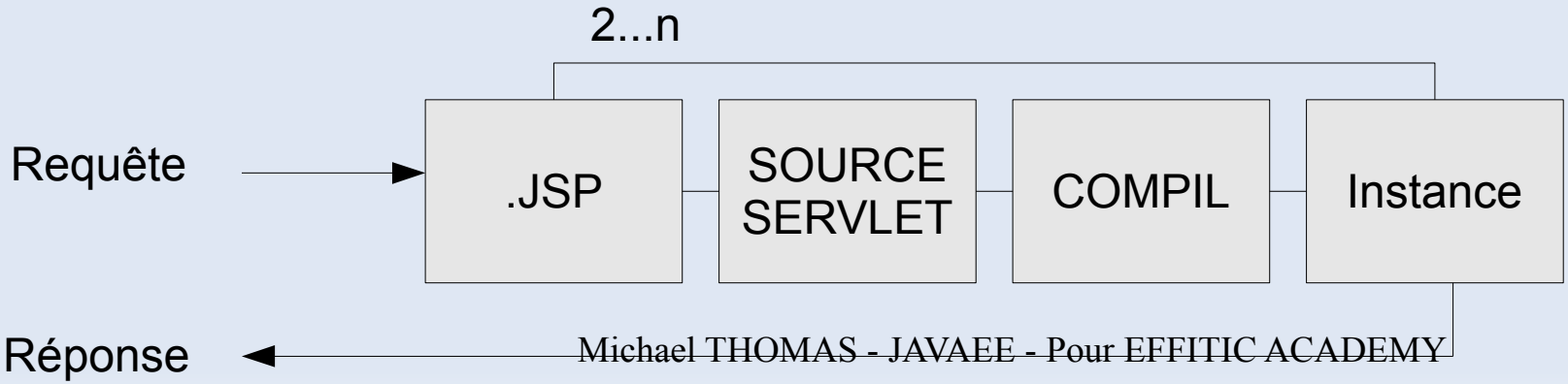
- Aspect transactionnel
  - Par défaut, les opérations sur la base de données sont en mode auto-commit. Dans ce mode, chaque opération est validée unitairement pour former la transaction.
  - Pour rassembler plusieurs opérations en une seule transaction:
    - connection.setAutoCommit(false);
    - connection.commit () ;
  - Retour en arrière
    - connection.rollback ();

Java Server Page (.jsp)

- Les JSP permettent de créer des pages web dynamiques en mélangeant:
  - Du code HTML
  - Des balises jsp spéciales
  - Du code java (scriptlet)
- Les JSP sont multi-plateformes (WriteOnce, Run anywhere)
- L'accès aux entités (données) peuvent être déportés dans des java beans
- Le JSP permettent de se concentrer uniquement sur la couche de présentation et ne nécessitent (masquent) la complexité du langage et du framework
- Comparable aux langages cotés serveur de type PHP, ASP, ...

Java Server Page (2)

- Mécanisme d'une JSP
  - Une jsp mélange du contenu textuel(html) et des directives spécifiques jsp (jsp tag) ainsi que du code java.
  - Avant d'être interprétée pour produire un résultat, une jsp est parsée, puis un code source de HttpServlet est généré puis compilé, enfin une instance de la servlet est créée.
  - Seul le premier appel à une page jsp génère tous ce process, par la suite l'instance reste disponible en mémoire comme un servlet classique.



Java Server Page (3)

- Syntaxe
  - Commentaires
    - Jsp : <%-- Un commentaire --%>
      - Invisible coté client.
    - Html : <!-- Un commentaire visible coté client →
  - Balise de déclaration
    - <%! … %>
    - Permet d'insérer du code dans la classe Servlet (attribut de classe, méthodes)
    - Les attributs et méthodes sont accessibles dans toute la jsp.

```
<%!
   private int counter = 0 ;
   private int getNext (int accountNo) { return ++counter;}
 %>
```

Java Server Page (4)

- Les directives contrôlent comment le serveur WEB doit générer les Servlet.
- Syntaxe
  - <%@ … %>
  - @include : indique au compilateur d'inclure un autre fichier

    <%@ include file="nomDeFichier.ext"%>
  - @taglib : indique une bibliothèque de balises à utiliser

    <%@ taglib prefix="myprefix"
    uri="taglib/mytag.tld" %>
  - @page : définit les attributs spécifiques à une page

Java Server Page (5)

- Directive **include**
  - <%@ include file="unAutreFichier" %>
  - Tout le contenu du fichier externe est inclus comme s'il était saisi directement dans la page JSP
  - Ne concerne que les ressources contenues dans le contexte
  - La racine du chemin du fichier à inclure est la racine du contexte
  - Pas de séparation de la portée des variables
  - Il s'agit d'une substitution de la balise avec le contenu du texte correpondant

Java Server Page (6)

- Directive **include** (exemple)

header.jsp

```
<hml>
  <head>
     <title>Hello World</title>
   </head>
<body>
```

footer.jsp

```
   </body>
</html>
```

unepage.jsp

```
<%@ include file = "/entete.jsp" %>

// Partie spécifique à la page
<%=  "Hello World %>

<%@ include file = "/footer.jsp" %>
```

Inclusion statique.
Comment modifier title ?

Java Server Page (7)

- Directive **include** vs <jsp:include page ../>

  <jsp:include page="header.jsp" >

  <jsp:param name="title" value="Mon titre"/>

  </jsp:include>

- Inclusion dynamique

-

```
<hml>
  <head>
     <title><%= request.getParameter("title")</title>
   </head>
<body>
```

Java Server Page (8)

- Directive **Page**
  - Définit les attributs spécifiques à une page.
    - **Import** : importe un paquetage java. Se réduit à un import dans la servlet.

```
<%@ page import="java.util.*, java.text.*" %>
```

    - **ContentType** : définit le type de contenu généré.

```
<%@ page contentType="text/html" %>
```

    - pendant le traitement de la requête HTTP

```
<%@ page errorPage="errorPage.jsp" %>
```

Java Server Page (9)

- Les Scriptlets
  - Bloc de code java permettant de mixer les directives du langage au contenu brut d'une page (html par exemple)
  - Syntaxe : <% … %>
  - Les scriptels peuvent accéder à tous les éléments définis dans les balises directives.

```
<ul><%  int result = 1;
While (count < 5) { %>
<li><%= result *= getNext(); %></li>
<% } %></ul>
```

Java Server Page (10)

- Les expressions
  - Syntaxe : <%= %>
  - Sert à évaluer une expression et à renvoyer sa valeur. La valeur retournée est converti en String.
  - Correspond à <% out.println(...) ; %>
  -

```
<%=3*5*6 %>
```

Ne nécessite pas de symbole de fin d'instruction ";"

Java Sever Page (11)

- Objets implicites accessible à partir d'un scriptlet :
  - **Request** : requête courante
  - **Response** : réponse courante
  - **Session** : session courante
  - **Out** : flux de sortie permettant l'écriture sur la réponse.
  - **Application** : contient les méthodes log() permettant d'écrire des messages dans le journal du contenu (ServletContent)
  - **PageContext** : utilisé pour partage directement des variables entre des pages jsp collaborant à la réalisation d'une vue
  - **Exception** : disponible uniquement dans les pages erreurs, permet d'obtenir des détails sur le message d'erreur.

Java Server Page (12)

- Cycle de vie d'une JSP
  - Le cycle de vie est identique à celui des servlet.
    - jspInit() est appelée après le chargement de la page
    - _jspService() est appelée à chaque requête
    - jspDestroy() est appelé lors du déchargement (fermeture d'une base de données)
  - Il est possible de redéfinir les méthodes jspInit() et jspDestroy() dans une page jsp en utilisant une directive de déclaration <%! %>

Java Server Page (13)

- Cycle de vie (2), exemple de redéfinition

```
<%!
    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()));
    }
%>
```

Attention : les objets implicites ne sont pas utilisable dans le fragment de déclaration. Porté différent des scriptlets.

Java Server Page (14)

- **Gestion des exceptions**
  - Jsp propose un mécanisme simple permettant de rediriger le contrôle vers une page jsp spécifique lorsqu'une exception se produit.
  - Evite à l'utilisateur d'observer l'instabilité du système et au hacker d'analyser le système pour y trouver des failles

Page pouvant produire une exception

```
<%@ page errorPage="erreur.jsp" %>
```

Page exception

```
<%@ page isErrorPage=true %>
```

Java Server Page (15)

- Gestion des exceptions (2)

Page pouvant produire une exception

```
 %@ page language="java"
                      contentType="text/html" %>
<%@ page errorPage="errorpage.jsp" %>
<html>
<head>
<title>Page avec une erreur</title>
</head>
<body bgcolor="white">
<% int var = 90; %>
Division par <% var = var / 0; %> <%= var %>
</body>
</html>
```

Page exception

```
<%@ page language="java"
contentType="text/html" %>
<%@ page isErrorPage="true" %>
<html>
<head>
  <title>Page gérant une erreur</title>
 </head>
 <body bgcolor="white">
        Attention une exception s'est
       Produite : t <%=
        exception.getMessage() %>
</body>
 </html>
```

Java Server Page (16)

- Collaboration entre jsp
  - Comme vu précédemment, la collaboration concerne le partage d'information et de contrôle.
  - Partage d'information :
    - GetContext(), getAttribute(), setAttribute()
  - Partage du contrôle :
    - inclusion <jsp:include .../>
    - forward <jsp:forward .../>

Java Server Page (17)

- Partage du contrôle
  - **Inclusion**

```
<jsp:include page="/page.html" />
```

  - La racine du chemin de la page est celle du contexte
  - Le serveur exécute la ressource dynamique indiquée et inclut sa sortie au contenu envoyé au client
  - Ne peut définir ni le code d'état ni la valeur des en-têtes
  - Possibilité de transmettre des informations lors de l'inclusion
  -

```
<jsp:include page="/page.jsp" />
   <jsp:param name="unParam" value="uneValeur"/>
</jsp:include>
```

Les Java Beans

- Les Java Beans sont des classes Java respectant un ensemble de directives.
  - 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 getNomPropriété()** : pas de paramètres et son type est celui de la propriété.
  - **void setNomDeLaPropriete(type)** : un seul argument du type de la propriété et son type de valeur est void
- un Java Beans peut également implémenter l'interface java.io.Serializable permettant la sauvegarde de l'état du Bean

Les Java Beans (2)

- Exemple de Java Beans

```
Class Account implements Serializable{
   private static final long serialVersionUID = 6686763335682632873L;

   private String login;
   private String password;

   public String getLogin() {return this.login;}
   public String setLogin(String s) {this.login = s;}
   public String getPassword() {return this.password;}
   public String setPassword(String s) {this.password = s;}
}
```

Les Java Beans (3)

- Déclaration et allocation d'un Java Bean dans une page jsp

```
<jsp:useBean id="Nom" class="Package.class" scope="portée" />
```

- **id** = "nom de l'instance" : nom de l'instance pour identification
- **class** = "Nom de la classe" : package du Bean
- **scope** = "attribut" : portée et accessibilité de l'objet Bean
  - **request** : Bean valide pour la requête et peut être transmise (forward)
  - **page** : idem request sans transmission (le cas de l'objet pageContext)
  - **session** : Bean ayant la durée de vie de la session l'application web
  - **application** : Bean créée pour l'application WEB courante

Les Java Beans (4)

- Lecture d'une propriété d'un bean
  - En utilisant le tag <jsp:getProperty/>

```
<jsp:getProperty name="référence Bean" property="nom propriété" />
```

  - En utilisant un script d'expression:

```
<%= nom_instance.getNomPropriete() %>
```

Les Java Beans (5)

- Modification des propriétés d'un bean
  - En utilisant le tag <jsp:getProperty/>

```
<jsp:setProperty name="référence" property="propriété" param="param" value= "valeur"/>
```

Nom d'un paramètre de requête contenant la valeur pour la propriété indiquée

Valeur explicite à donner à la propriété. Ne peut pas être combiné à l'attribut param

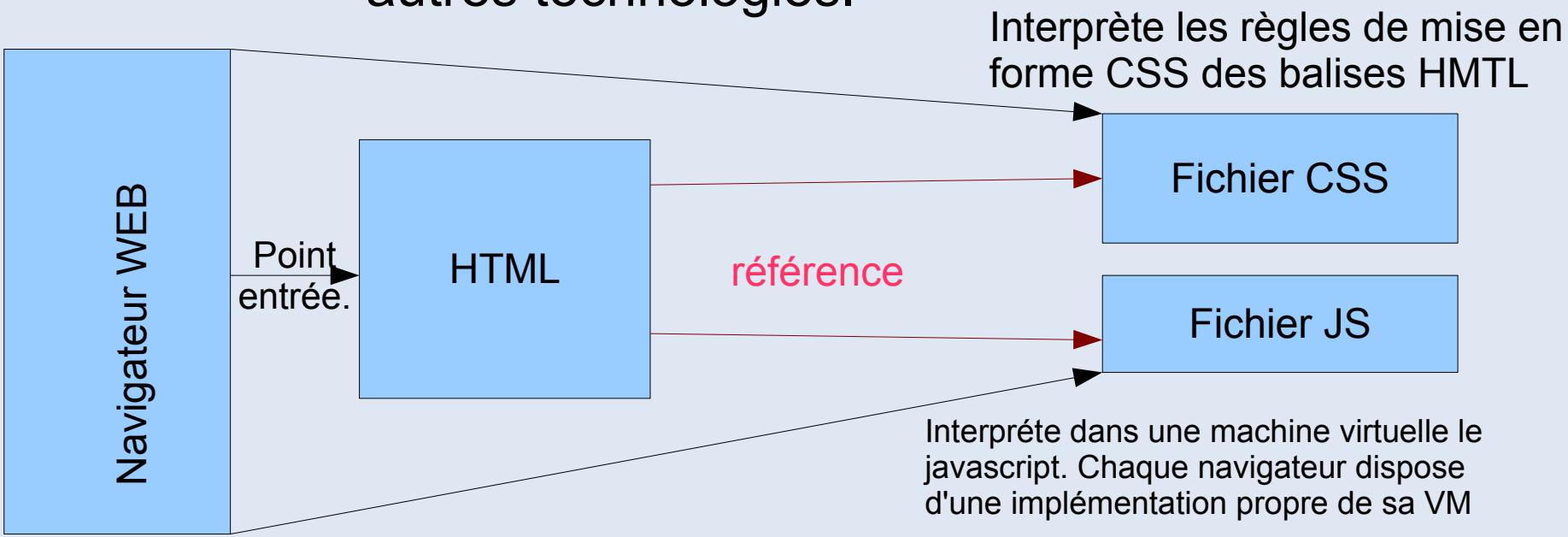
  - En utilisant un script d'expression:

```
<%= nom_instance.setNomPropriete(valeur) %>
```

Interface graphique application web

- Historiquement constituée de 3 technologies
  - Le HTML, le CSS, le Javascript.

La page HTML constitue le noeud principale d'une Vue Web. C'est elle qui fait appel aux autres technologies.



Interface graphique web (2)

- Rôle des différentes technologies.
  - Le **HTML** est un langage dérivé du XML qui a pour rôle de structurer de manière **logique** un contenu. Il ne doit en aucun cas donner des informations de mise en page.
  - Le **CSS**, quand à lui est un langage destiné à la mise en page d'une structure HTML. Il permet d'associer des règles de mise en page à des balises HTML
  - Le **javascript** permet d'apporter un comportement dynamique à des interfaces web html, qui elles sont par définition que des structures et donc statiques.

Interface graphique web (3)

- Squellette d'une page web. Principales balises

```
<html>
	<head></head>
	<body></body>
</html>
```

<**html**> : cette balise représente l'enveloppe d'une page html.
<**head**> : contient toutes les métas informations d'une page html, son contenu n'est pas visuel.
<**body**> : contient la structure visuelle de l'information à présenter aux utilisateurs.

Interface Web (4)

- Principales balises de la section <head>
  - <title></title> : permet de donner un titre à la page. Ce titre apparaitra dans l'onglet et/ou la barre de titre du navigateur.
  - <link> : permet de référencer des fichiers liés à la page web, et particulièrement des fichiers css.
    - Ex : <link rel="stylesheet" type="text/css" href="/Login/style.css"></link>
  - <script> : permet de référencer des fichiers contenant des scripts dynamique, par exemple des javascripts.
    - Ex : <script src="javascript.js"></script>

Interface web (5)

- Création d'un fichier css
  - Créer à la racine de votre projet un fichier style.css
  - Le remplir des règles suivantes

```
body  {background-color:black;}
.MonInput {border:1px solid gray; background-color:#CECECE;}
#monFormulaire {background-color:blue;border:1px solid white;}
```

- Les liens avec les balises se fait tout simplement en nommant la balise.
- Les balises peuvent être regroupées par genre (par classe) grace à l'attribut html class. Ainsi en ajouter l'attribut class="MonInput" toutes vos balises input, celles ci suivront communémement la règle .MonInput. Pour référencer une classe en utilise le symbole . (point)
- Pour référencer une balise particulière, on utilise l'attribut html id, par exemple id="monFormulaire" que l'on ajouter à <form id="monFormulaire"..... et le symbole de liaison en css est le # (diese)

- Pour visualiser sous Tomcat.
  - Ouvrir votre fichier web.xml et rajouter les mappings suivants :

```
<servlet-mapping>
   <servlet-name>default</servlet-name>
   <url-pattern>*.css</url-pattern>
   <url-pattern>*.js</url-pattern>
   <url-pattern>*.jpg</url-pattern>
   <url-pattern>*.png</url-pattern>
   <url-pattern>*.gif</url-pattern>
</servlet-mapping>
```

- Ajouter d'un fichier javascript et de fonctionnalités dynamiques.
  - Ajouter la ligne : <script src="/Login/javascript.js"></script>
  - Créer et éditer le fichier javascript.js à la racine de votre projet.

```
function valid_form(){
 var login = document.getElementsByName("login")[0];
 var password = document.getElementsByName("password")[0];
  if ("" ==login.value ) { alert("votre champs login est vide !"); return false;}
  if ("" ==password.value ) { alert("votre champs password est vide !"); return false;}
   return true;
}
```

Associer cette fonction à l'évènement onSubmit de la balise form

```
<form id="monFormulaire" onSubmit="javascript:return valid_form();" .....
```