

# SIL4

## Programmation Web en Java

Francis.Brunet-Manquat@imag.fr - MIAM (bureau 112)

Hervé Blanchon – SIMO (bureau 105)

Basé sur le cours de Leila Kefi-Khelif

# Objectifs

- Développer des applications Web en Java
- Introduction aux technologies Java EE
- Utiliser une couche d'accès aux données
  - Mapping Objet-Relationnel

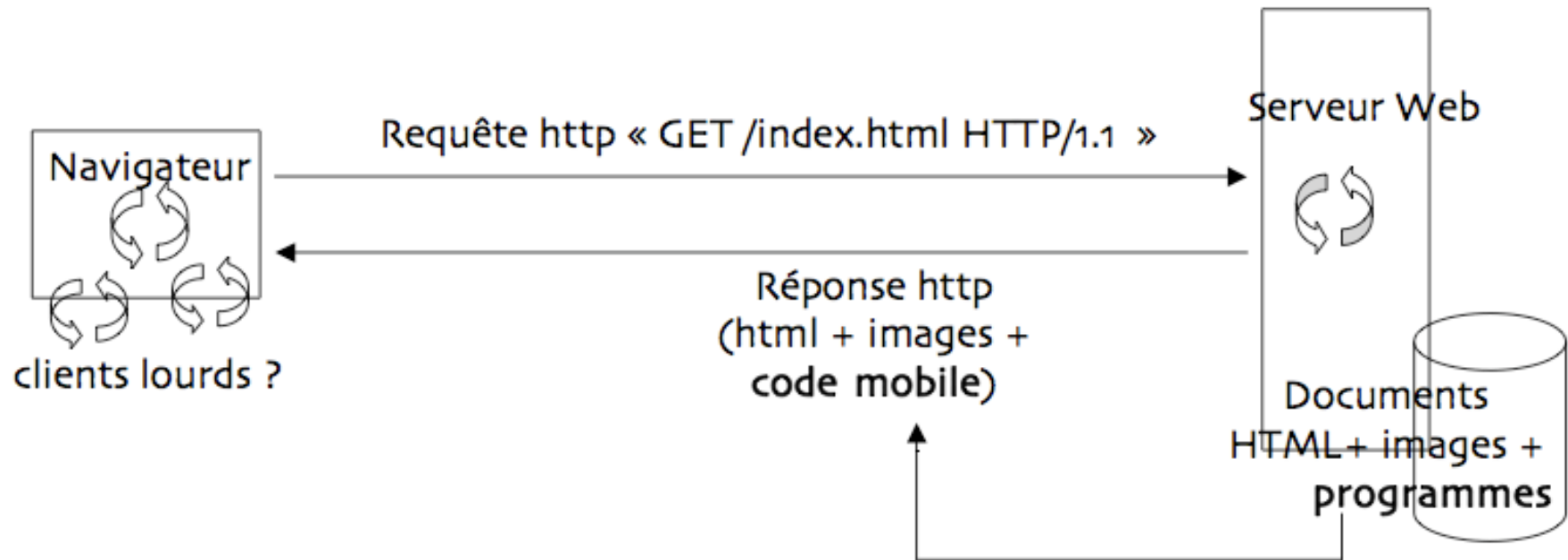
# Modalités

- Planning
  - Semaine 5 Décembre : 4h + 4h (C/TP)
  - Semaine 2 janvier : 4h + 4h (C/TP)
  - Semaine 16 janvier : 4h (C/TP) ou examen ?
  - Semaine 30 janvier : 4h (C/TP) ou examen ?
- Evaluation
  - 1 examen machine
  - 1 projet en monôme

# Plan des cours

- Première partie: **Application web en Java EE**
  - Cours 1 : Servlet, JSP
  - Cours 2 : Javabeau
  - Cours 3 : MVC, session
  - Projet : mise en place du MVC
- Deuxième partie: **Persistance**
  - Cours 4 : Couche d'accès aux données (JPA)
  - Projet : intégration de la persistance
- Troisième et quatrième semaine
  - Projet : ajout de nouvelles fonctionnalités

# Application web : orienté client



- Javascript, Applet, active X, ...
- **Avantages** : libère le serveur, distribution de la charge
- **Inconvénients** : sécurité des 2 cotés

# Application web : orienté serveur



- CGI, **Servlet**, **JSP**, PHP, ASP, ...
- **Avantages** : modularité, réutilisabilité, maintenance facilitée, meilleures possibilités d'évolution
- **Inconvénients** : plus de trafic sur le réseau

# Java Enterprise Edition

- Spécification pour le langage Java destinée aux applications d'entreprise (Java EE, anciennement J2EE)
- Interfaces de programmations (API)
  - **Servlet** : conteneur web
  - Portlet : conteneur web, extension Servlet
  - **JavaServer Pages (JSP)** : framework web
  - JavaServer Faces (JSF) : framework web, extension JSP
  - JDBC : connection à une base de données
  - EJB : composants distribués transactionnels
  - ...

# Servlet

- Pour la création d' applications **dynamiques** fonctionnant coté serveur
- Classe java: chargée dynamiquement, elle étend les fonctionnalités d' un serveur web et **répond à des requêtes dynamiquement**
  - permet de gérer des requêtes HTTP et de fournir au client une réponse HTTP
- S'exécute par l'intermédiaire d' une JVM
- S'exécute dans un **moteur de Servlet** ou conteneur de Servlet (Tomcat, Jetty, GlassFish, etc.) permettant d' établir le lien entre la Servlet et le serveur Web



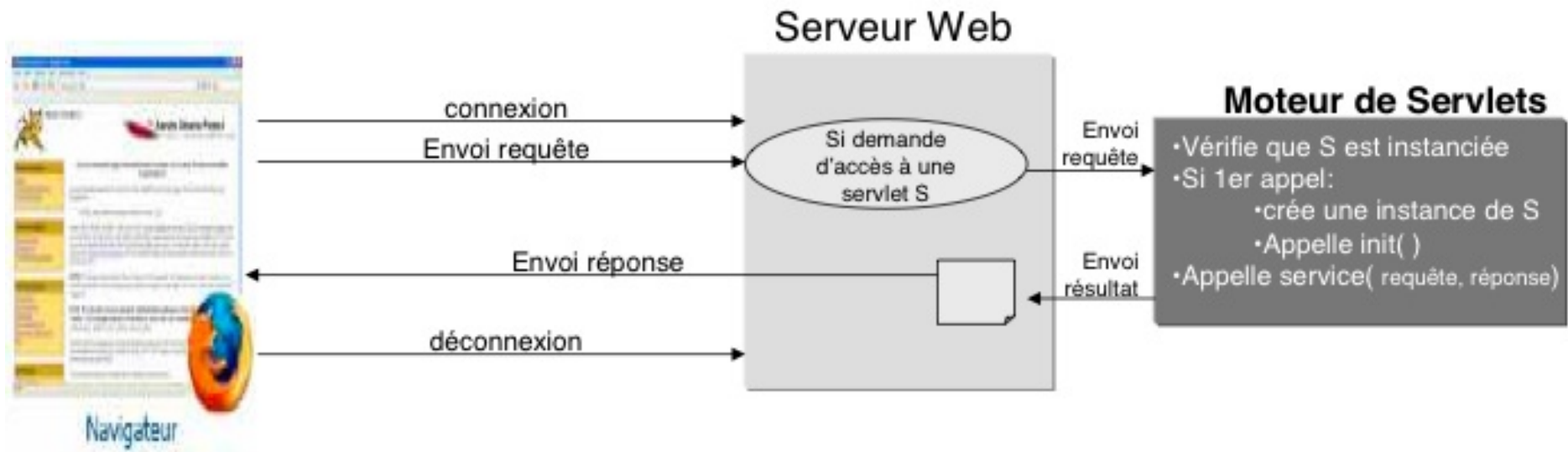
# Servlet : avantages

- **Portabilité** : niveau systèmes et serveurs
- **Efficacité** : semi compilée, multithread, gestion du cache, connexions persistantes
- **Puissance**: partage de données entre servlets, chaînage de servlets
- **Pratique**: gestion des cookies, suivi des sessions, manipulation simple du protocole HTTP
- **Réutilisable, accès aux API Java**

# Servlet : cycle de vie (1/4)

- **Une seule instance** par Servlet est utilisée
- Une requête client a pour résultat un nouveau thread transmis à `service()`
- Une Servlet est **initialisée, utilisée puis détruite**
  - Cycle de vie de la Servlet

# Servlet : cycle de vie (2/4)



Moteurs de servlet : Tomcat, Jetty, GlassFish ...

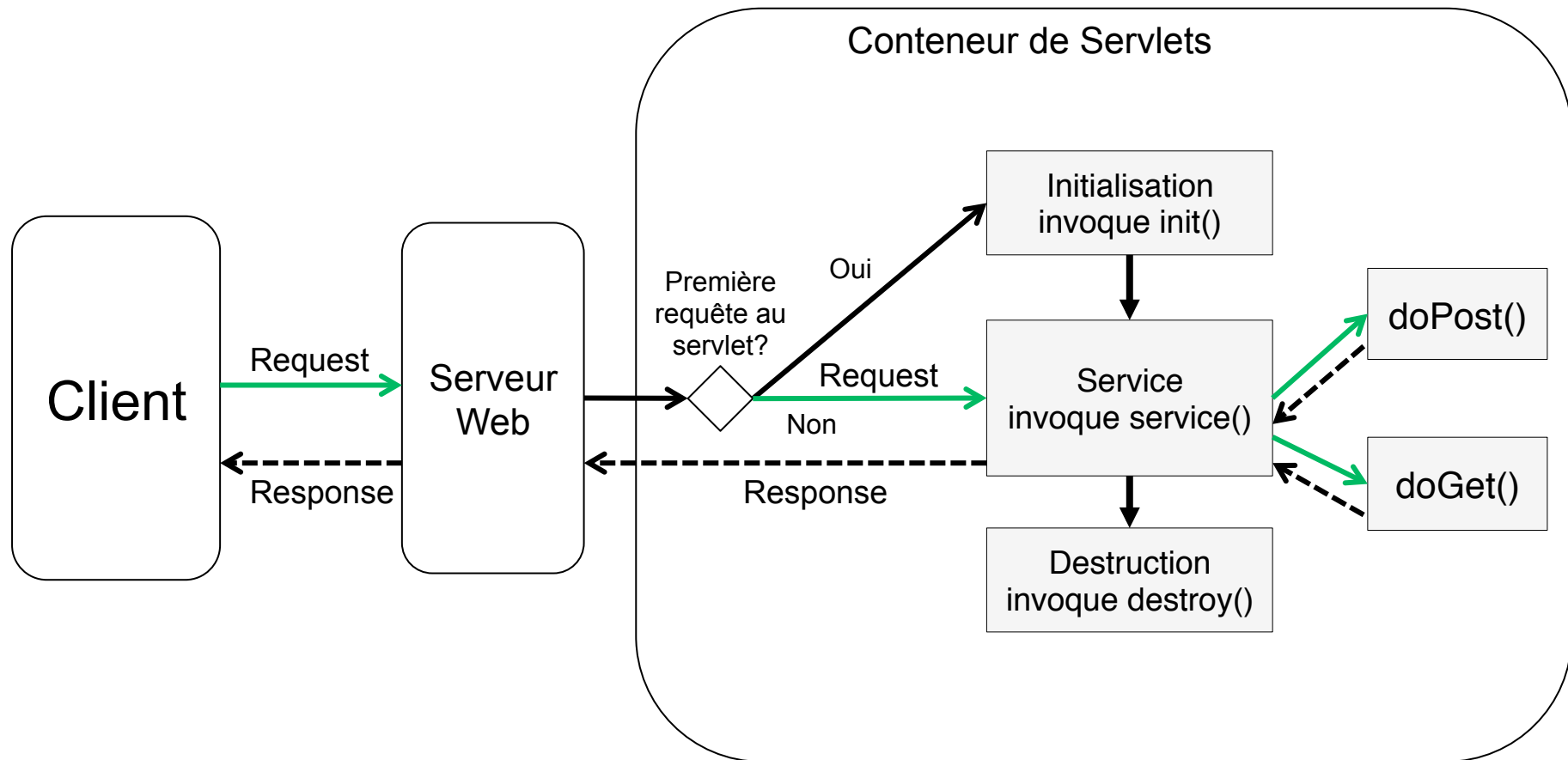
# Servlet : cycle de vie (3/4)

## *Les méthodes*

- Initialisation: méthode `init()`
  - Appelée uniquement lors du **1er appel** à la Servlet
  - Paramètres: avec ou sans paramètre de configuration
  - Permet d'effectuer des **opérations d'initialisation** de la Servlet
    - Initialisation des données, ouverture de connexion d'une BD, de fichiers, ...
- Utilisation: méthode `service()`
  - Appelée pour chaque requête reçue
  - Paramètres: **ServletRequest** et **ServletResponse**
  - Permet de traiter la requête et de produire une réponse
- Destruction: méthode `destroy()`
  - Appelée uniquement lors de la **suppression** de l'instance de la servlet
    - Demande administrateur, temps d'inactivité trop grand
  - Permet d'effectuer des **opérations de « nettoyage »**
    - Fermeture de connexion d'une BD, de fichiers, ...

# Servlet : cycle de vie (4/4)

## *Les méthodes*



# Servlet : exemple *réponse XML*

```
@WebServlet("/ExempleGenericServletToXML")
public class ExempleGenericServletToXML extends GenericServlet {

    public void service(ServletRequest request, ServletResponse response)
                        throws ServletException, IOException {

        // Content type
        response.setContentType("application/xml");
        response.setCharacterEncoding("UTF-8");

        // Content
        PrintWriter out = response.getWriter() ;
        out.println("<?xml version='1.0' encoding='UTF-8'?>");
        out.println("<troll>");
        out.println("<name>Profytroll</name>");
        out.println("</troll>");
    }
}
```

## **Réponse produite:**

```
<troll>
    <name>Profytroll</name>
</troll>
```

# Servlet : exemple *réponse HTML* (1/2)

```
@WebServlet("/ExempleGenericServletToHTML")
public class ExempleGenericServletToHTML extends GenericServlet {

    public void service(ServletRequest request, ServletResponse response)
                        throws ServletException, IOException {

        // Content type
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");

        // Content
        PrintWriter out = response.getWriter() ;
        out.println ("<!DOCTYPE html>") ;
        out.println ("...") ;
        out.println ("<p>Hello world!</p>") ;
    }
}
```

## ***Réponse produite:***

```
<!DOCTYPE html>
...
<p>Hello world!</p>
```


# Servlet : exemple *réponse HTML* (2/2)

```
@WebServlet("/ExempleGenericServletToHTML")
public class ExempleGenericServletToHTML extends GenericServlet {

    // DATA
    private Date dateInit;
    private int nombreChargementPage;

    @Override
    public void init() throws ServletException {
        super.init();
        dateInit = new Date();
        nombreChargementPage = 0;
    }

    public void service(ServletRequest request, ServletResponse response)
        throws ServletException, IOException {
        ...
        out.println("<p>Nombre chargement de la page : " + ++nombreChargementPage + "</p>") ;
        out.println("<p>Date de la visite : " + new Date() + "</p>") ;
        out.println("<p>Date d'initialisation de la servlet : " + dateInit + "</p>") ;
    }
}
```



Initialisation



# Démonstration

- Disponible sur chamilo
  - Projet ExemplesServlet

*Intégration lors du TP0*

- A remarquer :
  - Servlets (classes java) dans le dossier src/
  - Route :
    - Annotation @WebServlet
      - Exemple @WebServlet("/ExempleGenericServletToXML")
    - URL d'accès:
      - `<a href="/ExemplesServlet/ExempleGenericServletToXML" >...</a>`

# API servlet

- L'API fournit deux classes pour l'implémentation:
  - **GenericServlet** : pour la conception de Servlets indépendantes du protocole
    - Méthode `service(ServletRequest req, ServletResponse res)`
  - **HttpServlet** : pour la conception de Servlets spécifiques au protocole HTTP
    - Méthode `doPost(HttpServletRequest req, HttpServletResponse res)`
    - Méthode `doGet(HttpServletRequest req, HttpServletResponse res)`

# Lire la requête : HttpServletRequest

- Encapsule la requête HTTP et fournit des méthodes pour :
  - récupérer les paramètres passés au serveur par le client
  - accéder aux informations du client et de l'environnement du serveur

Méthode	Description
String <code>getMethod()</code>	Récupère la méthode HTTP utilisée par le client
String <code>getHeader(String Key)</code>	Récupère la valeur de l'attribut Key de l'en-tête
String <code>getRemoteHost()</code>	Récupère le nom de domaine du client
String <code>getRemoteAddr()</code>	Récupère l'adresse IP du client
String <code>getParameter(String Key)</code>	Récupère la valeur du paramètre Key (clé) d'un formulaire. Lorsque plusieurs valeurs sont présentes, la première est retournée
String[] <code>getParameterValues(String Key)</code>	Récupère les valeurs correspondant au paramètre Key (clé) 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 <code>getParameterNames()</code>	Retourne un objet <i>Enumeration</i> contenant la liste des noms des paramètres passés à la requête
String <code>getServerName()</code>	Récupère le nom du serveur
String <code>getServerPort()</code>	Récupère le numéro de port du serveur

# Créer la réponse : HttpServletResponse

- Utilisé pour construire un message de réponse HTTP renvoyé au client, il contient :
  - les méthodes nécessaires pour définir le type de contenu, en-tête et code de retour
  - un flot de sortie pour envoyer des données (par exemple HTML) au client

Méthode	Description
String setStatus(int StatusCode)	Définit le code de retour de la réponse
void setHeader(String Nom, String Valeur)	Définit une paire clé/valeur dans les en-têtes
void setContentType(String type)	Définit le <b>type MIME</b> de la réponse HTTP, c'est-à-dire le type de données envoyées au navigateur
void setContentLength(int len)	Définit la taille de la réponse
PrintWriter getWriter()	Retourne un objet <i>PrintWriter</i> 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 <i>ServletOutputStream</i> , dérivé de la classe <i>java.io.OutputStream</i>
void sendredirect(String location)	Permet de rediriger le client vers l'URL <i>location</i>

# JSP - Java Server Pages

- Servlet
  - Accent mis sur le code java
  - Plus « appel de service »
- JSP – « *le php de Java EE* »
  - Code Java embarqué dans une page HTML entre les balises `<%` et `%>`
  - Séparation entre traitement de la requête et génération du flux html

# JSP vs Servlet (1/3)

```
public void service(ServletRequest request, ServletResponse response)
    throws ServletException, IOException {

    // Content type
    response.setContentType("text/html");
    response.setCharacterEncoding("UTF-8");

    // Content
    PrintWriter out = response.getWriter() ;
    out.println ("<!DOCTYPE html>") ;
    out.println ("<html>");
    out.println ("<head>");
    out.println ("<title>Bonjour tout le monde !</title>") ;
    out.println ("</head>");
    out.println ("<body>");
    out.println ("<p>Hello world!</p>") ;

    out.println ("<p>Nombre chargement de la page : " + ++nombreChargementPage + "</p>") ;
    out.println ("<p>Date de la visite : " + new Date() + "</p>") ;
    out.println ("<p>Date d'initialisation de la servlet : " + dateInit + "</p>") ;
    out.println ("</body>");
    out.println ("</html>");
}
```

# JSP vs Servlet (2/3)

```
<!DOCTYPE html>

<html>
  <head>
    <title>Bonjour tout le monde !</title>
  </head>
  <body>

    <!-- Content -->
    <p>Hello world!</p>

    <p>Nombre chargement de la page : <%= ++nombreChargementPage%></p>
    <p>Date de la visite : <%= new Date()%></p>
    <p>Date d'initialisation de la servlet : <%= dateInit%></p>

  </body>
</html>
```

# JSP vs Servlet (3/3)

## Servlet générée

```
public void _jspService(ServletRequest request, ServletResponse response)
    throws ServletException, IOException {

    JspFactory _jspFactory = null;
    PageContext pageContext = null;
    try {
        ...
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        pageContext = _jspxFactory.getPageContext(this, request, response, ...)
        PrintWriter out = pageContext.getOut() ;
        out.println ("<!DOCTYPE html>") ;
        out.println ("<html>");
        out.println ("<head>");
        out.println ("<title>Bonjour tout le monde !</title>") ;
        out.println ("</head>");
        out.println ("<body>");
        out.println ("<p>Hello world!</p>") ;

        out.println ("<p>Nombre chargement de la page : ")
        out.println (++nombreChargementPage);
        out.println ("</p>") ;
        ...
    }
}
```



# JSP - Java Server Pages (2/2)

- Désigné par une URL

**`http://localhost:8080/ExemplesServlet/ExempleJSP.jsp`**

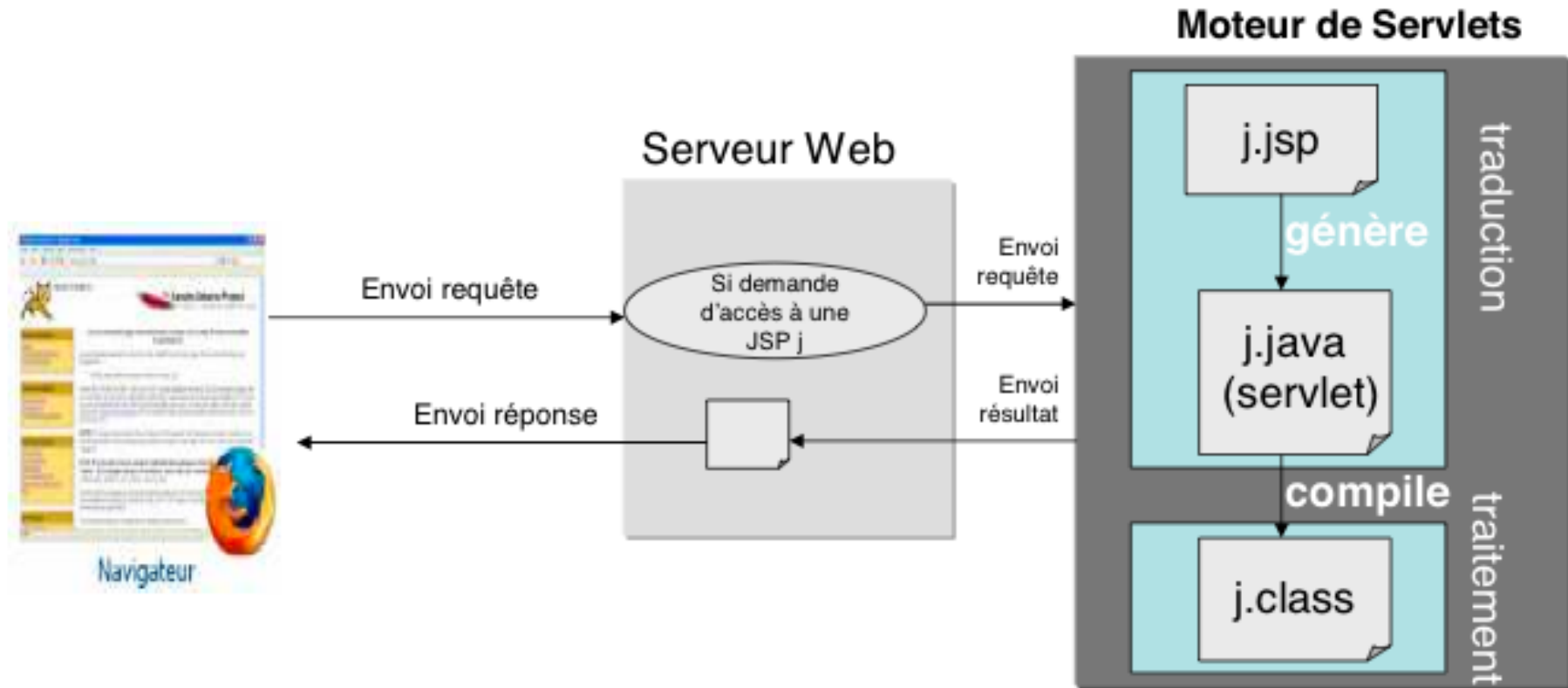
- **Pages JSP converties en Servlet** par le moteur de Servlets lors du premier appel aux JSP

# JSP : Cycle de vie (1/3)

- Identique au cycle de vie d'une Servlet
  - **MAIS** des noms de méthodes différentes :
    - appel de la méthode **jspInit()** après le chargement de la page
    - appel de la méthode **\_jspService()** à chaque requête
    - appel de la méthode **jspDestroy()** lors du déchargement

Remarque: Il est possible de redéfinir dans la JSP les méthodes `jspInit()` et `jspDestroy()`

# JSP : Cycle de vie (2/3)



# JSP : Cycle de vie (3/3)

```
<!DOCTYPE html>
```

```
<%!
```

```
// DATA
```

```
private Date dateInit;
```

```
private int nombreChargementPage;
```

```
// INITIALISATION
```

```
public void jspInit() {  
    dateInit = new Date();  
    nombreChargementPage = 0;  
}
```

} Modification de jsplnit()

```
%>
```

```
<html>
```

```
<head>
```

```
<title>Bonjour tout le monde !</title>
```

```
</head>
```

```
<body>
```

```
<%-- Content --%>
```

```
<p>Hello world!</p>
```

```
<p>Nombre chargement de la page : <%= ++nombreChargementPage%></p>
```

```
<p>Date de la visite : <%= new Date()%></p>
```

```
<p>Date d'initialisation de la servlet : <%= dateInit%></p>
```

```
</body>
```

```
</html>
```

# Démonstration : jeu1

<b>Pierre-feuille-ciseaux</b> <i>jeu de société</i>	
Ce jeu appartient au <i>domaine public</i> .	
autres noms	papier-caillou-ciseaux roche-papier-ciseaux pierre-papier-ciseaux feuille-caillou-ciseaux chifoumi jankenpon
format	deux mains !
mécanismes	choix simultané intuition
joueur(s)	2
âge	à partir de 6 ans
durée annoncée	5 minutes

## Pierre-Papier-Ciseaux

Choisissez une main



☐ Pierre



☒ Papier



☐ Ciseaux

Valider

Module TC4, 2009-2010

# TP0 (disponible sur Chamilo)

## 1. Installation de Tomcat 8.5

## 2. Installation Eclipse Java EE:

- Si Eclipse installé : installation du plugin Eclipse WTP
- Sinon : installation d'Eclipse IDE for Java EE Developers

## 3. Installation du jeu : Pierre-Papier-Ciseaux

- Télécharger Jeu1.zip sur Chamilo (pareil pour ExemplesServlet.zip)
- Dans Eclipse :
  - Import Existing Projects into Workspace
  - Select archive file : Jeu1.zip (sur l'intratek)
  - Lire le fichier Alire.txt à la racine du projet pour la fin de l'installation

# JSP en détails

- Une page JSP est composée
  - d' une structure statique HTML
  - d' éléments dynamiques de la page
- 3 types d' éléments
  - éléments de script
  - directives
  - éléments d' action

# Éléments de script (1/5)

- 4 types d'éléments de script:
  - les déclarations `<%! ... %>`
  - les expressions `<%=...%>`
  - les scriptlets `<%...%>`
  - les commentaires `<%--...--%>`



# Éléments de script (2/5)

- **Les déclarations : `<%! ... %>`**
  - permettent de déclarer des méthodes et des variables d'instance connus dans toute la page JSP

```
<%!  
private int mon_entier;  
private int somme(int a, int b) {return (a+b);}  
%>
```

# Éléments de script (3/5)

- **Les expressions : `<%=...%>`**
  - permettent d'évaluer une expression et renvoyer sa valeur (string)
  - correspond à `out.println(...);`

```
Nous sommes le : <%=new java.util.Date()%>
```

# Eléments de script (4/5)

- **Les scriptlets : `<%...%>`**
  - permettent d'insérer des blocs de code java  
(*qui seront placés dans `_jspService(...)`*)

```
<% int som=0;  
    for (int i=1; i< 15; i++)  
        {som=somme(som,i);}  
%>
```

# Éléments de script (5/5)

- **Les commentaires : `<%--...--%>`**
  - permettent d'insérer des commentaires (*qui ont l'avantage de ne pas être visibles pour l'utilisateur*)

```
<%-- ceci est un commentaire --%>
```

# Éléments de script et objets implicites (1/2)

- Objets implicites
  - liste d'objets permettant d'interagir avec l'environnement de la servlet d'exécution (présents dans la méthode *service(...)* )
  - ne sont utilisables que dans les éléments de scripts JSP de type scriptlet et expression

# Éléments de script et objets implicites (2/2)

- **request**: requête courante (HttpServletRequest)
- **response**: réponse courante (HttpServletResponse)
- **out** : flot de sortie permet l'écriture sur la réponse
- **session** : session courante (HttpSession)
- **application** : espace de données partagé entre toutes les JSP (ServletContext)
- **page** : l'instance de servlet associée à la JSP courante (this)

# Les directives `<%@...%>`

- permettent d'indiquer au conteneur de servlet la façon dont il doit générer la servlet
- 3 types de directives:
  - Les directives de pages
  - Les directives d'inclusion
  - Les balises personnalisées

# Les directives de page (1/3)

**<%@ page ...%>**

- permettent de définir les attributs spécifiques à une page
- Par exemple, définir les "import" nécessaires au code Java de la JSP
  - **<%@ page import="java.io.\*"%>**



# Les directives de page (2/3)

- Définir le type MIME du contenu retourné par la JSP
  - `<%@ page contentType="text/html"%>`
- Fournir l'URL de la JSP à charger en cas d'erreur
  - `<%@ page errorPage="err.jsp"%>`
- Définir si la JSP est une page invoquée en cas d'erreur
  - `<%@ page isErrorPage="true" %>`
- Déclarer si la JSP peut être exécutée par plusieurs clients à la fois
  - `<%@ page isThreadSafe="false" %>`

# Les directives de page (3/3)

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ page errorPage="ErrorDiv.jsp" %>
<html>
  <head>
    <title>Exemple @ errorPage</title>
  </head>
  <body>

    <h3>Division de 2 nombres au hasard compris entre 0 et 5</h3>
    <%int denom = (int)(Math.random() * 5 );
    int numer = (int)(Math.random() * 5 );%>
    Le résultat de la division de
    <%=numer%> par <%=denom%> est : <%=numer/denom%>

  </body>
</html>
```

Exp\_errorPage.jsp

```
<%@ page isErrorPage="true" %>
<html>
  <head>
  </head>
  <body>
    <h2>Erreur : Division par zéro </h2>
  </body>
</html>
```

ErrorDiv.jsp

http://localhost:8084/AppliWeb\_JSP/Exp\_errorPage.jsp

**Division de 2 nombres au hasard compris entre 0 et 5**

Le résultat de la division de 3 par 1 est : 3

http://localhost:8084/AppliWeb\_JSP/Exp\_errorPage.jsp

**Erreur : Division par zéro**

Voir projet exemplesServlet

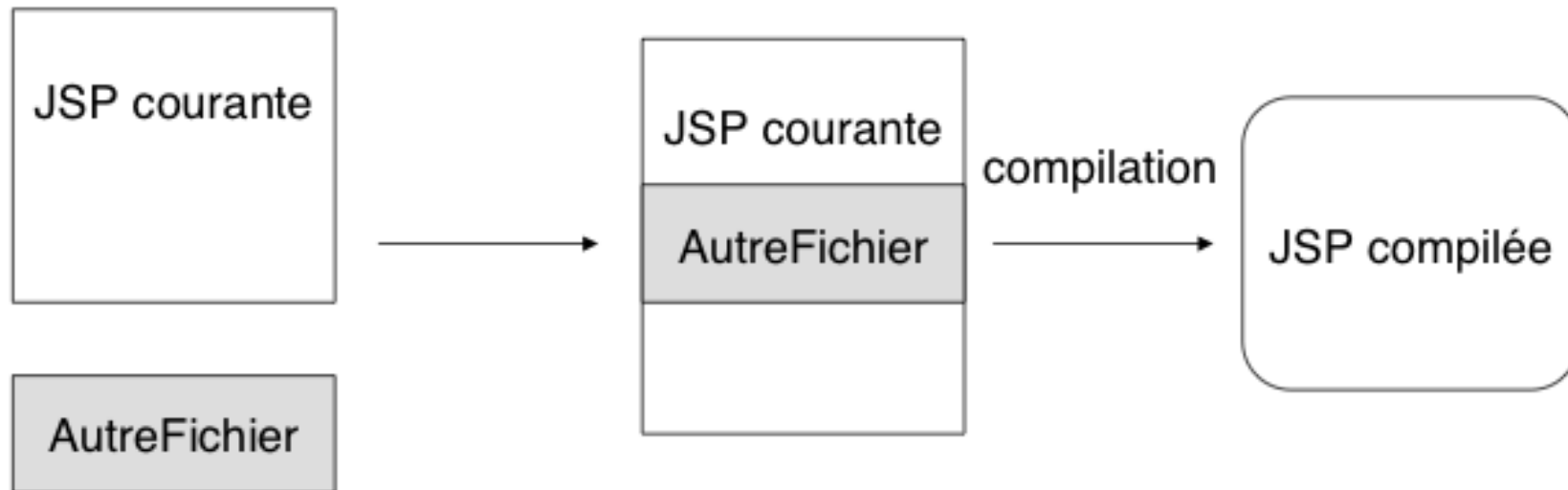
# Les directives d'inclusion (1/3)

**<%@ include ...%>**

- permettent d'inclure le contenu d'un autre fichier dans la page JSP courante
- inclusion effectuée avant la compilation de la jsp

# Les directives d'inclusion (2/3)

`<%@ include file="AutreFichier"%>`



# Les directives d'inclusion (3/3)

```
<html>
  <head>
    <title>Exp @ include</title>
  </head>
  <body>
    <%
      String le_nom = "Dupont";
      String le_prenom="toto";
      String l_adresse="Nice";
    %>
    <%% include file="ficheInfo.jsp" %>
  </body>
</html>
```

Exp\_include.jsp

```
<h3> Informations sur la personne </h3>
Nom : <%=le_nom%><br>
Prénom : <%=le_prenom%><br>
Adresse : <%=l_adresse%><br>
```

ficheInfo.jsp



# Les balises personnalisées

**<%@ taglib ...%>**

- permettent d'indiquer une bibliothèque de balises : adresse et préfixe, pouvant être utilisées dans la page

**<%@ taglib prefix="pref" uri="taglib.tld" %>**

<http://adiguba.developpez.com/tutoriels/j2ee/jsp/jstl/>

# Les éléments d'action (1/2)

- 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
- Constitués de balises pouvant être intégrées dans une page jsp (syntaxe XML)

`<jsp: .../>`

# Les éléments d'action (2/2)

- Actions jsp standards :
  - jsp:include et jsp:param
  - jsp:forward
  - jsp:useBean
  - jsp:setProperty et jsp:getProperty

[http://fr.wikipedia.org/wiki/JavaServer\\_Pages#Actions\\_JSP](http://fr.wikipedia.org/wiki/JavaServer_Pages#Actions_JSP)



# Jsp:include et jsp:param

- jsp:include :
  - identique à la directive `<%@ include ...` sauf que l'inclusion est faite au moment de la requête. Donc après compilation ...
- jsp:param :
  - permet de passer des informations à la ressource à inclure

```
<jsp:include page="ficheInfo.jsp" flush="true">  
    <jsp:param name="le_nom" value="Dupont"/>  
    <jsp:param name="le_prenom" value="toto"/>  
    <jsp:param name="l_adresse" value="Nice"/>  
</jsp:include>
```

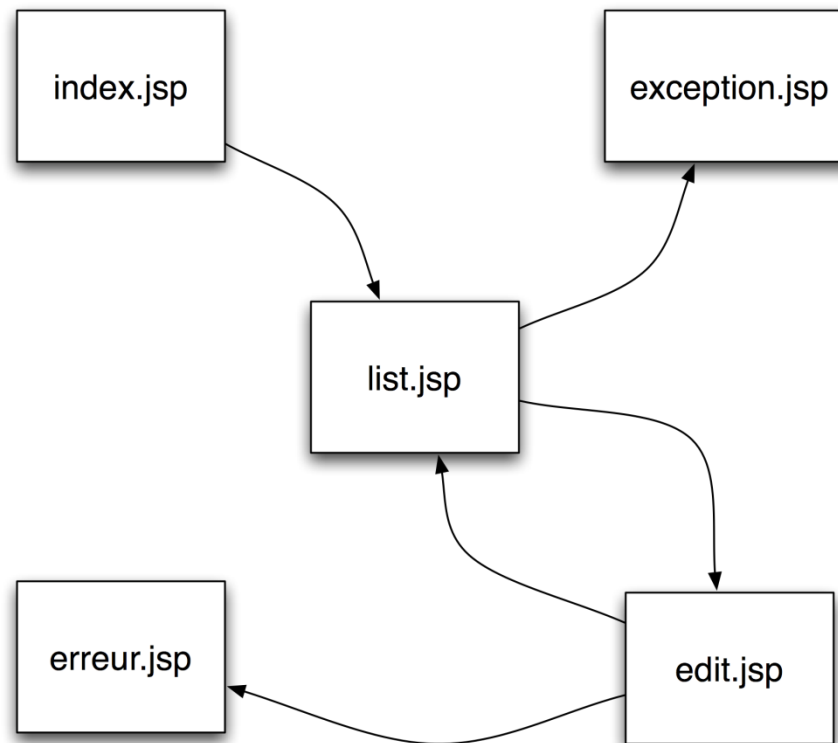
# Jsp:forward

- Permet de passer le contrôle de la requête à une autre ressource
- jsp:param permet ici aussi de passer des informations à la ressource de redirection

```
<jsp:forward page="Redirect.jsp">  
    <jsp:param name="monParam" value="maValeur"/>  
</jsp:forward>
```

# Chaînage de JSP (1/3)

- Enchaînement de JSP pour produire un site web dynamique



# Chaînage de JSP (2/3)

- URL

```
<a href="jeu1/jeu1.jsp">Jeu 1 - directives, elements, chaînage</a>
```

```
<a href="jeu1/resultat1.jsp?mainJoueur=pierre">
```

Passage du paramètre mainJoueur

- Formulaire

```
<form method="post" action="resultat1.jsp">
```

```
  <input type="radio" name="mainJoueur" value="pierre"/>Pierre<br/>
```

```
  ...
```

Passage du paramètre mainJoueur

# Chaînage de JSP (3/3)

- Récupération de paramètre

<%

// Récupération du paramètre mainJoueur

mainJoueur = request.getParameter("mainJoueur");

%>

# Exemple : jeu1 (1/3)

<b>Pierre-feuille-ciseaux</b> jeu de société	
Ce jeu appartient au domaine public.	
autres noms	papier-caillou-ciseaux roche-papier-ciseaux pierre-papier-ciseaux feuille-caillou-ciseaux chifoumi jankenpon
format	deux mains !
mécanismes	choix simultané intuition
joueur(s)	2
âge	à partir de 6 ans
durée annoncée	5 minutes

## Pierre-Papier-Ciseaux

Choisissez une main



☐ Pierre



☒ Papier



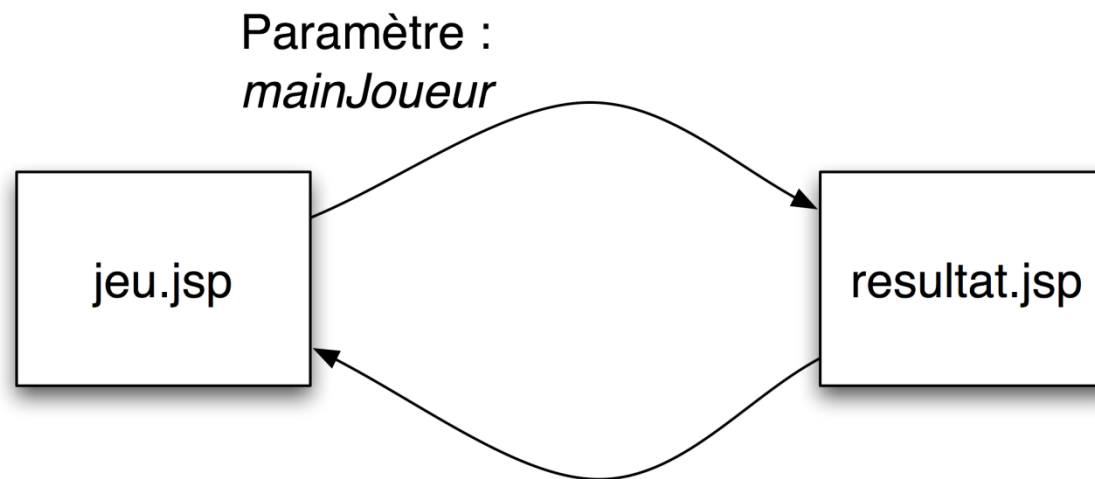
☐ Ciseaux

Valider

Module TC4, 2009-2010

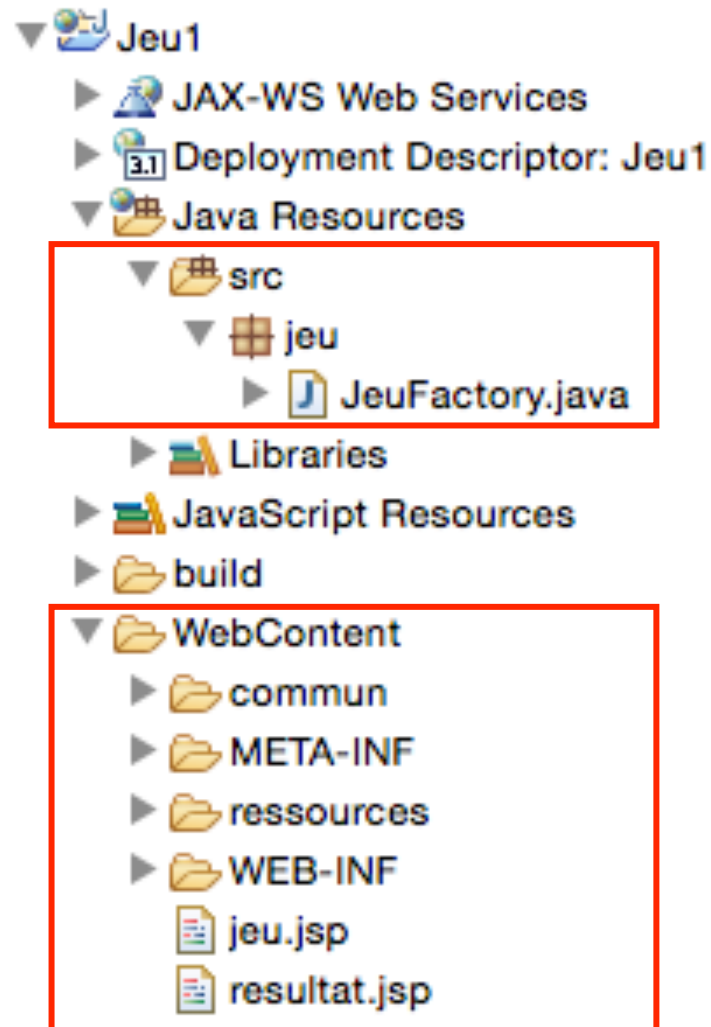
# Exemple : jeu1 (2/3)

- Une solution possible



# Exemple : jeu1 (3/3)

Structure  
de l'application  
web



Sources java

Contenu web : JSP,  
images, HTML, etc.



# Votre projet (1/3)

- Application de gestion des notes et des absences des étudiants

Le département informatique de l'IUT2 propose de vous engager pour réaliser une application client-serveur destinée aux enseignants, aux administratifs et aux étudiants. Cette application permettra de gérer les notes et les absences des étudiants.

Exemple:

<http://projet-tut-info-1.iut2.upmf-grenoble.fr:8080/SIL4-projetNotesEtAbsences/>

# Votre projet (2/3)

- Version minimale
  - Un enseignant doit pouvoir :
    - Consulter les groupes d'étudiants
    - Editer des notes d'un étudiant ou d'un groupe
    - Consulter des notes d'un étudiant ou d'un groupe
    - Editer des absences d'un étudiant ou d'un groupe
    - Consulter les absences d'un étudiant ou d'un groupe

# Votre projet (3/3) *projet\_SIL4.pdf sur Chamilo*

- Etape 1 (basée sur le cours 1)
  - Création de deux pages JSP : index.jsp et details.jsp
  - Index.jsp : affichage de tous les étudiants
  - Envoie d'un paramètre id de index.jsp à details.jsp
  - Details.jsp : affichage des détails d'un étudiant
  - AIDE : Utiliser les classes Etudiant et GestionFactory dans aideProjet.zip
- Etape 2 (basée sur le cours 2)
  - Intégration d'un javabeau
  - Premier pas vers MVC
- Etape 3 : mise en place du MVC (cours 3)
- Etape 4 : mise en place de la persistance (cours 4)