Web-MI4 et SIMO-MI7

Programmation Web en Java

Francis Brunet-Manquat – SIMO
Hervé Blanchon – AW

Basé sur le cours de Leila Kefi-Khelif

Objectifs

- Développer des applications Web en Java
  - Introduire certaines technologies Java EE / JEE (Entreprise Edition)
  - Version entreprise de la plate-forme Java
  - Utilisation côté serveur

- Utiliser une couche d'accès aux données
  - Mapping Objet-Relationnel

Plan des cours

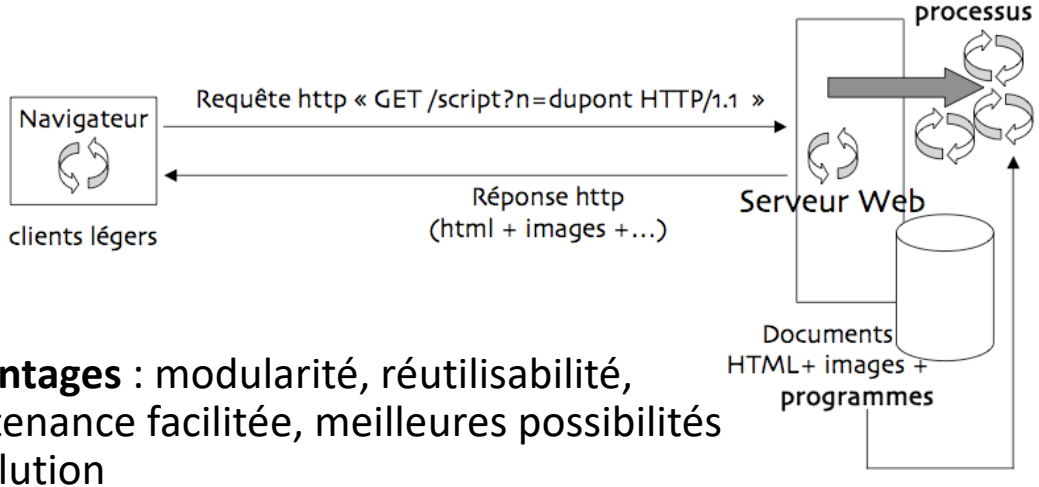
- Première partie: **Application web en Java EE**
  - Cours 1, 2 , 3 : Servlet, JSP, Javabean, MVC, session
  - Projet : mise en place du MVC

- 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 serveur**

PHP, **Servlet, JSP**, ASP, …



- **Avantages** : modularité, réutilisabilité, maintenance facilitée, meilleures possibilités d'évolution
- **Inconvénients** : serveur effectue tout le traitement, plus de trafic sur le réseau

Amélioration possible : coupler avec un client riche, libère le serveur mais sécurité du côté serveur et du côté client !

Java EE : Java Entreprise Edition

- Spécification pour le langage Java destinée aux applications d'entreprise
  - Mots clés : Java EE, JEE (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

- **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**

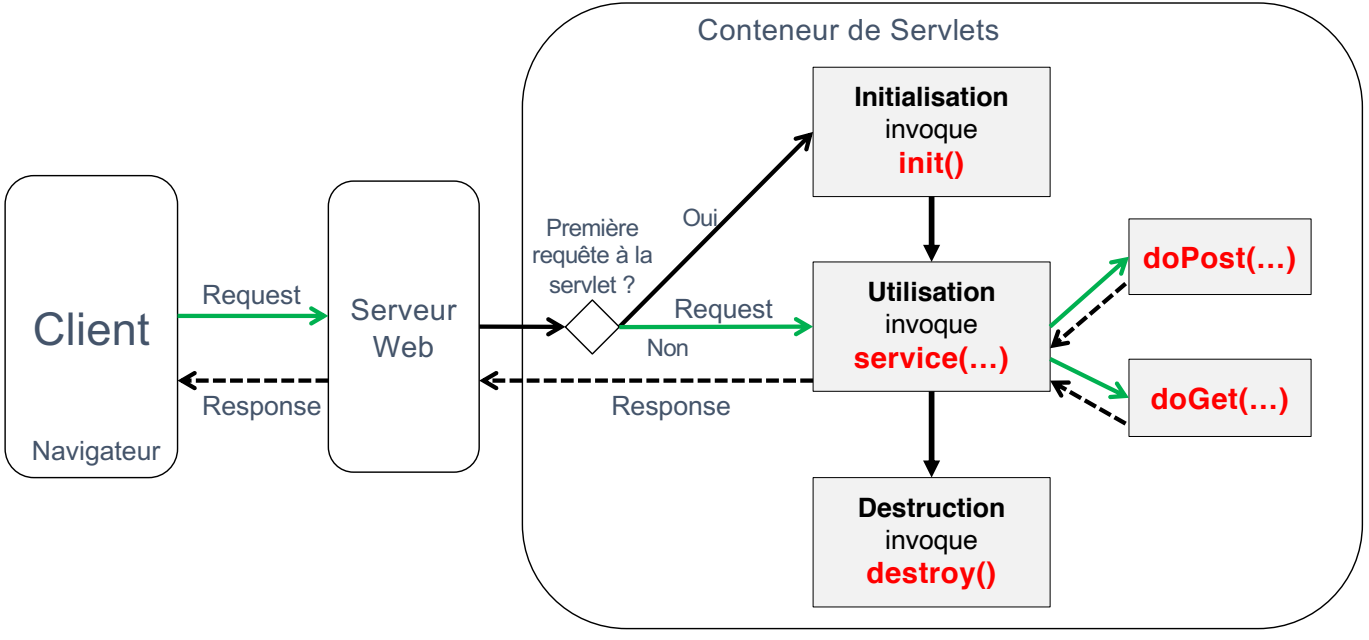
Servlet : cycle de vie (1/3)

- Une servlet est **initialisée**, **utilisée** pendant un certain temps puis **détruite**
- **Une seule instance** par servlet est utilisée
- Une requête client a pour résultat un **nouveau thread** qui utilisera la méthode service(…) de l'instance de la servlet

Servlet : cycle de vie (2/3)

- 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(…)** (peut être affinée en **doPost(…)** ou **doGet(…)**)
  - Appelée pour chaque requête du client
  - 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 (3/3)



Conteneurs de servlets : Tomcat, Jetty, GlassFish …

Servlet : réponse XML (1/2)

```
@WebServlet("/ExempleHttpServletToXML")
public class ExempleHttpServletToXML extends HttpServlet {
        private static final long serialVersionUID = 1L;

protected void doGet(HttpServletRequest request, HttpServletResponse 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 sur le navigateur:

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

Servlet : réponse XML (2/2)

```java
@WebServlet("/ExempleHttpServletToXML")
public class ExempleHttpServletToXML extends HttpServlet {
        private static final long serialVersionUID = 1L;

protected void doGet(HttpServletRequest request, HttpServletResponse 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>");
}
```

URL d'accès à la servlet

Description du contenu de la réponse faite au client

Écriture de la réponse

→ *à voir dans le projet ExemplesServlet dans src/sil4/ExempleHttpServletToXML*

Servlet : réponse **HTML** (1/2)

```
@WebServlet("/ExempleHttpServletToHTML")
public class ExempleHttpServletToHTML extends HttpServlet {
        private static final long serialVersionUID = 1L;

protected void doGet(HttpServletRequest request, HttpServletResponse 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 ("<p>Hello world!</p>");
        ...
```

Réponse produite sur le navigateur:

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

Servlet : réponse **HTML** (2/2)

```
@WebServlet("/ExempleHttpServletToHTML")
public class ExempleHttpServletToHTML extends HttpServlet {
        private static final long serialVersionUID = 1L;

        // DATA
        private Date dateInit;
        private int nombreChargementPage;

        // INIT
        public void init(ServletConfig config) throws ServletException {
                dateInit = new Date();
                nombreChargementPage = 0;
        }

protected void doGet(HttpServletRequest request, HttpServletResponse 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>");
```

URL d'accès à la servlet

Attributs de la servlet

Initialisation

Écriture de la réponse en utilisant les attributs

→ *à voir dans le projet ExemplesServlet dans src/sil4/ExempleHttpServletToHTML*

Servlet : 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)`**

*Pour créer nos servlets, nous créerons des classes qui **héritterons** de* `HttpServlet`

La requête : objet de type **`HttpServletRequest`**

- Encapsule la requête HTTP et fournit des méthodes pour :

• Obtenir les paramètres de l'utilisateur	`getParameter(…)`
• Stocker et obtenir les objets au sein de la requête	`getAttribute(…)`
• Lire les en-tête	`getHeader(…)`
• Obtenir les objets Session	`getSession(…)`

https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html

La réponse : objet de type **`HttpServletResponse`**

- Utilisé pour construire un message de réponse HTTP renvoyé au client

• Renvoyer la réponse vers l'utilisateur	`getWriter(…)`
• Changer ou mettre à jour les valeurs des en-tête	`setHeader(…)`
• Rediriger vers un autre URL	`sendRedirect(…)`
• Renvoyer une erreur au client	`sendError(…)`

https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html

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>");
}
```

Description du contenu de la réponse faite au client

Écriture de la réponse

JSP vs Servlet (2/3)

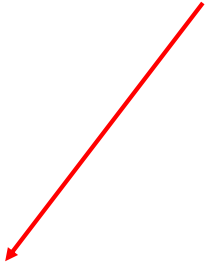
Même réponse au client que la servlet précédente

Désigné par une URL : http://localhost:8080/ExemplesServlet_war_exploded/ExempleJSP.jsp

```
<!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 (2/3)

Code Java embarqué dans une page HTML à l'aide de balise



```
<!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>
```

→ *à voir dans le projet ExemplesServlet dans web/ExempleJSP.jsp*

JSP vs Servlet (3/3)

```
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");
                pageContent = _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>") ;
                …
```

Pour information : La JSP est convertie en servlet par le moteur de servlets lors du premier appel

JSP : Cycle de vie (1/2)

- **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 (2/2)

```jsp
<!DOCTYPE html>
<%!
        // DATA
        private Date dateInit;
        private int nombreChargementPage;

        // INITIALISATION
        public void jspInit() {
                dateInit = new Date();
                nombreChargementPage = 0;
        }
%>

<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>
```

Déclaration des attributs

Exemple de **redéfinition** de ma méthode de jspInit()

Utilisation des attributs

JSP en détails

- Une page JSP est composée
  - d'une structure statique HTML
  - d'éléments dynamiques de la page

- 3 types d'éléments dynamiques
  - **Eléments de script** : déclaration de variables, expressions, scriplet (code), commentaires
  - **Directives** : donne des indications pour la génération de la servlet
  - **Eléments d'action :** simplification d'écriture sous forme de balises

Eléments de script (1/4)

❑ Les déclarations : <%! … %>

- Permettent de déclarer des méthodes et des variables d'instance
- Les méthodes et les variables seront connues dans toute la page JSP

```
<%!
        // Déclaration d'une variable
        private int maVariable;

        // Déclaration d'une méthode
        private int somme(int a, int b) {return a+b;}
%>
```

Eléments de script (2/4)

❑ Les expressions : <%= … %>

- Une expression renvoie la valeur de l'instruction qu'elle contient sous la forme d'une chaîne de caractères (`String`)
  - **Aide**: pensez à redéfinir la méthode `toString()` si vous souhaitez afficher vos objets
- Correspond à `out.println(…)` dans une servlet classique

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

Eléments de script (3/4)

❑ Les scriplets : <% … %>

- Permettent d'insérer des blocs de code java directement dans le HTML
- Ces blocs seront placés dans la méthode `_jspService(…)` lors de la génération de la servlet

```
<%
    int resultat = 0;
    for (int i=1; i<15; i++) {
        resultat = somme(resultat,i);
    }
%>
<div>Le résultat est de <%= resultat %></div>
```

→ *à voir dans le projet ExemplesServlet dans web/ExempleCours.jsp*

Eléments de script (4/4)

- **Les commentaires : <%-- … --%>**
  - permettent d'insérer des commentaires, non visibles pour le client

```
<%-- Exemple d'expression --%>
<p>Nous sommes le : <%= new java.util.Date() %></p>

<%-- Exemple de scriplet --%>
<%
    int resultat = 0;
    for (int i=1; i<15; i++) {
        resultat = somme(resultat,i);
    }
%>

<%-- Exemple de scriplet avec un entier --%>
<div>Le résultat est de <%= resultat %></div>
```

Eléments 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 scriplet et expression

Eléments 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`)

→ *à voir dans le projet ExemplesServlet la page objetsImplicites.jsp*

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
    - Non développé dans ce cours, mot clé : taglib
    - Librairie personnalisée la plus connue JSTL

Les directives de page (1/2)

<%@ 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 (2/2)

- 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

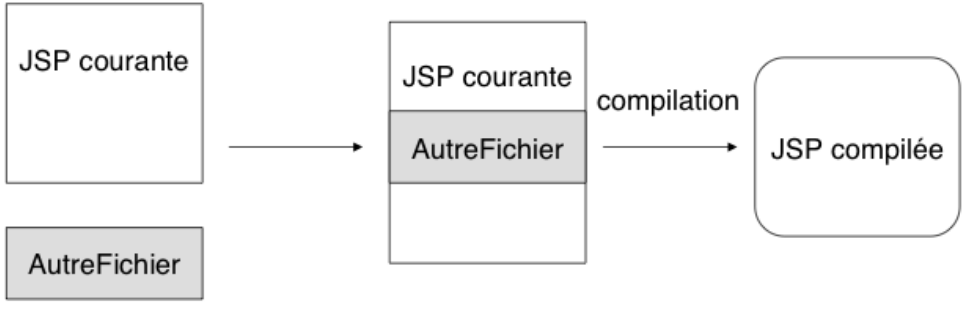
```
<%@ page isThreadSafe="false" %>
```

→ *à voir dans le projet ExemplesServlet dans web/errorPage/*.jsp*

Les directives d'inclusion

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

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



→ *à voir dans le projet ExemplesServlet dans web/directiveInclusion/*.jsp*

Les éléments d'action

- Simplification d'écriture sous forme de balises
  - Récupérer et utiliser des objets (Java Beans)
  - Inclure dynamiquement un fichier
  - Rediriger vers une autre page

```
<jsp:… />
```

- 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

jsp:include et jsp:param

- jsp:include :
  - Identique à la directive <%@ include … sauf que l'inclusion est faite dynamiquement. Donc après compilation …

- jsp:param :
  - Permet de passer des informations à la ressource à inclure

```
<jsp:include page="ficheInfo2.jsp">
    <jsp:param name="nom" value="Brunet-Manquat"/>
    <jsp:param name="prenom" value="Francis"/>
</jsp:include>
```

→ *à voir dans le projet ExemplesServlet dans web/directiveInclusion/*.jsp*

Jsp:forward

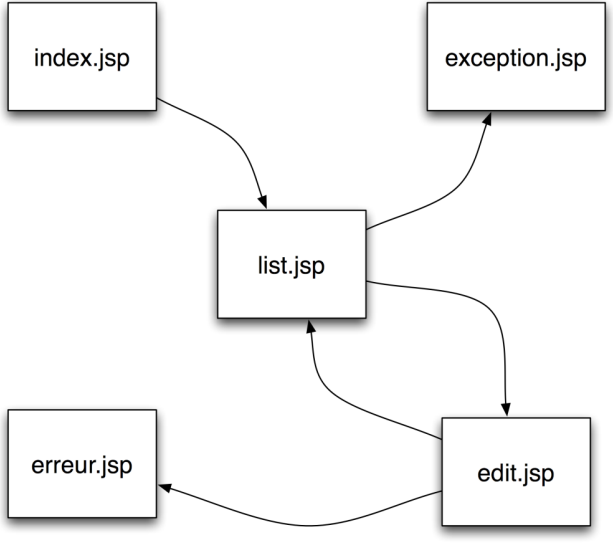
- Permet de passer le contrôler de la requête à une autre ressource

- `jsp:param` permet de passer des informations à la ressource de redirection

```
<jsp:forward page="Redirect.jsp">
    <jsp:param name="nom" value="Brunet-Manquat"/>
    <jsp:param name="prenom" value="Francis"/>
</jsp:forward>
```

Chaînage de JSP (1/2)

- Enchaînement de JSP pour produire un site web dynamique



Chaînage de JSP (2/2)

❑ URL

```
<a href="jeu1.jsp">Jeu 1 – directives, elements, chaînage</a>
<a href="resultat1.jsp?mainJoueur=pierre">
```

Passage du paramètre mainJoueur

❑ Formulaire

```
<form method="post" action="resultat1.jsp">
      <input type="radio" name="mainJoueur" value="pierre"/>Pierre<br/>
      …
```

❑ Récupération de paramètre

```
<% mainJoueur = request.getParameter("mainJoueur");%>
```

Exemple : jeu1 (1/2)

Pierre-feuille-ciseaux 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

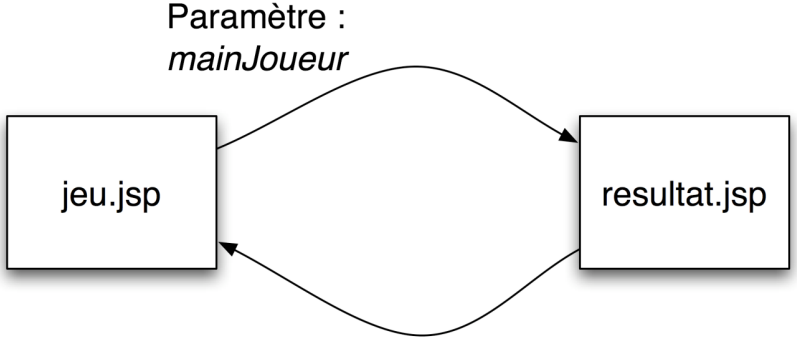
Valider

Module TC4, 2009-2010

Exemple : jeu1 (2/2)

→ *à voir dans le projet Jeu1*

- Une solution possible



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.

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
    details.jsp : affichage des détails d'un étudiant
  - Envoie d'un paramètre id de index.jsp à details.jsp
  - AIDE : Utiliser les classes Etudiant et GestionFactory dans aideProjet.zip

- Etape 2 (basée sur le cours 2)
  - Intégration d'un javabean
  - Premier pas vers MVC

- Etape 3 : mise en place du MVC (cours 3)

- Etape 4 : mise en place de la persistance (cours 4)