Java Server Pages
JSP

Mohammadi Zakaria Zakariamhm@gmail.com

Java Server Page (JSP)

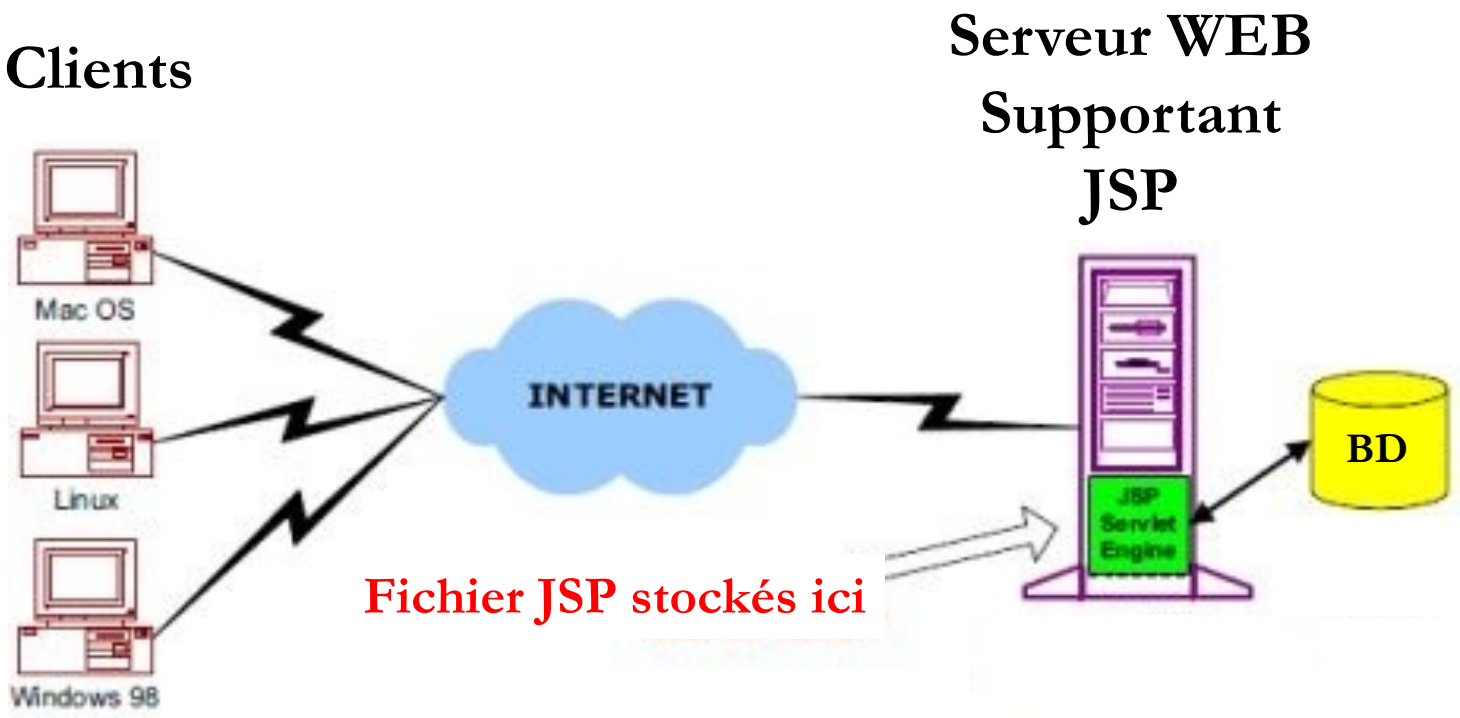
Motivation

- Nécessité d'avoir des pages HTML dynamiques i.e. pages créées lors de la requête (météo, cours de la bourse, vente aux enchères, etc...)
- Technologie des *server side include* (ssi).

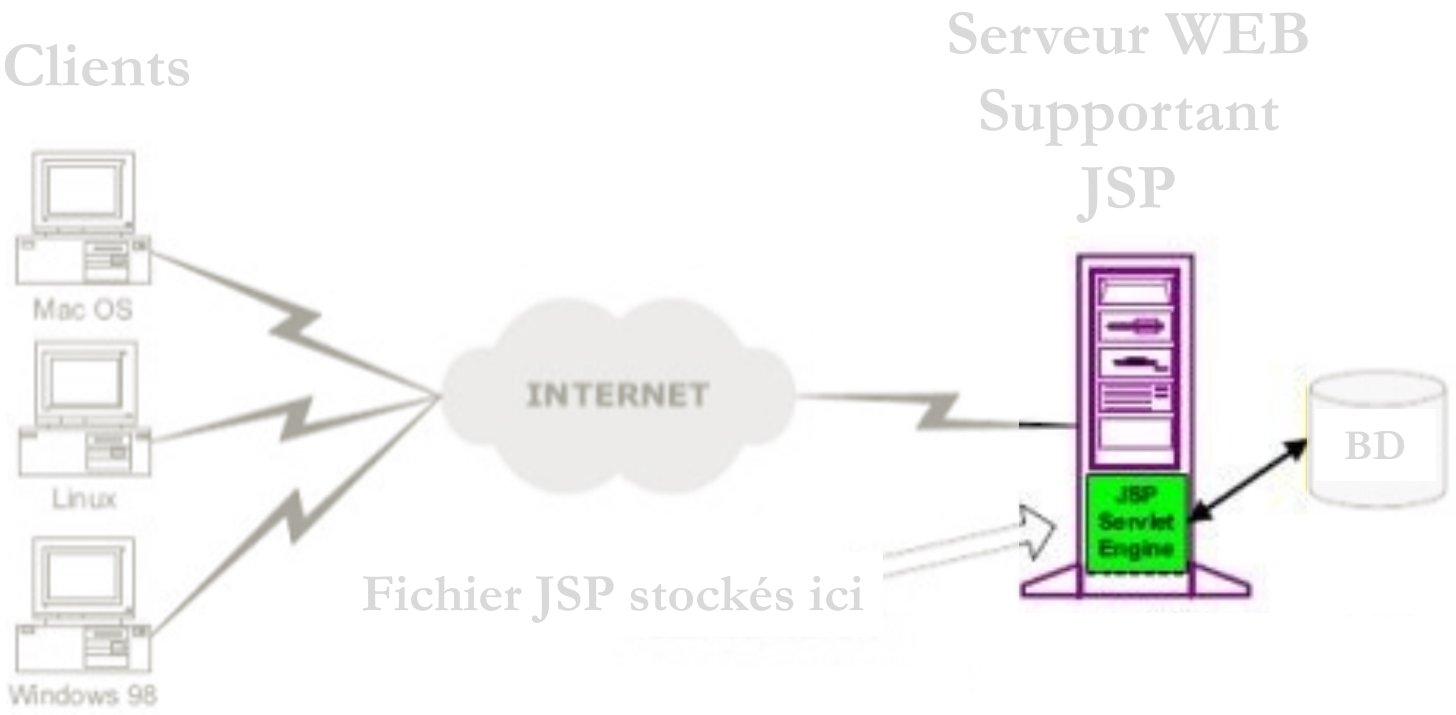
JSP ?

- Standard Java permettant de développer des Applications Web interactives.
- Langage de script qui combine :
  - un langage à balises (html ou xml),
  - des fragments de code java.
- construction coté serveur de pages web dynamiques.
- concurrents de *php, cgi, asp*…
- Langage script exécuté du côté serveur.

JSP ?


Clients
Serveur WEB
Supportant
JSP
Mac OS
Linux
Windows 98
INTERNET
BD
JSP
Servlet
Engine
Fichier JSP stockés ici

JSP ?

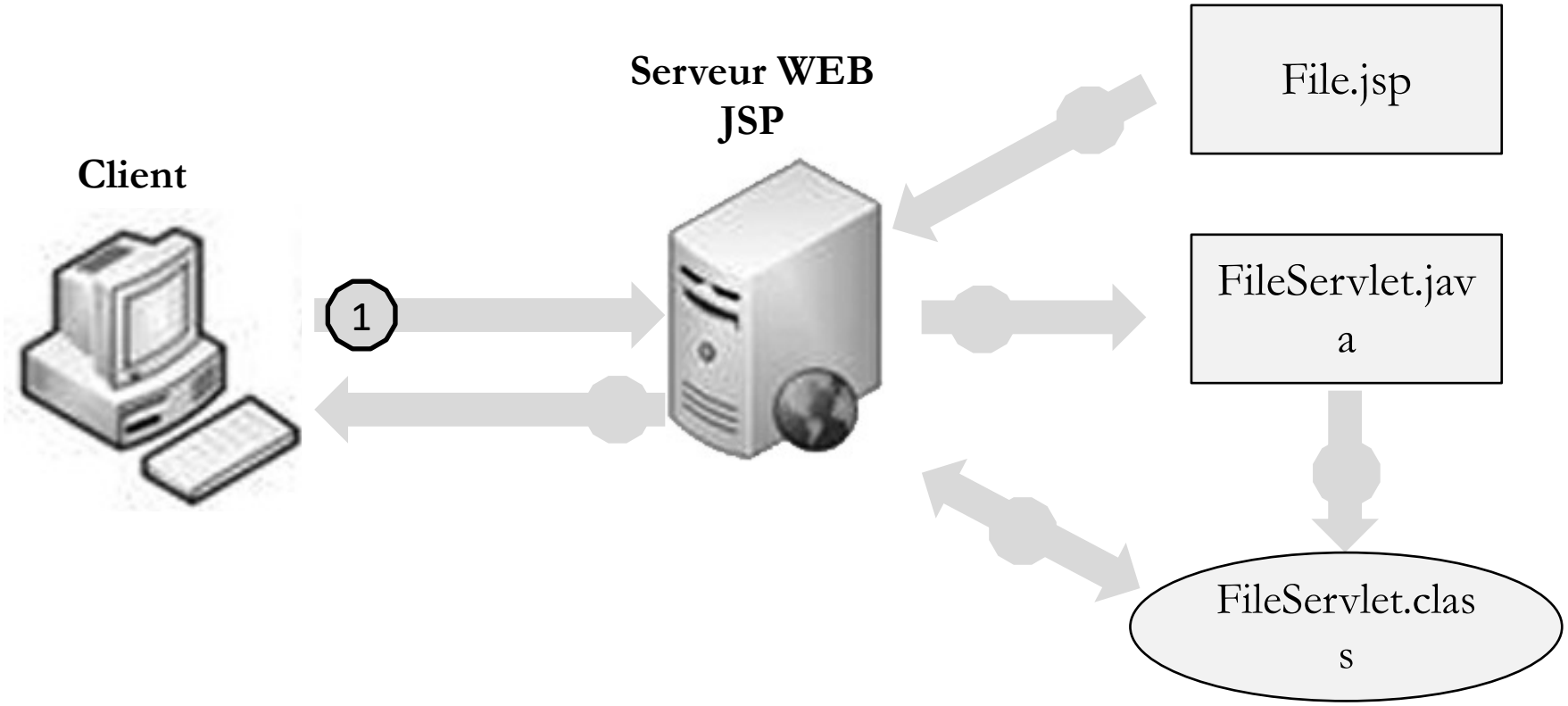


> Le serveur Web, lorsqu'une telle page est demandée, passe la main au programme adéquat qui traite la partie de la page le concernant.

> Ce programme génère la partie dynamique en HTML

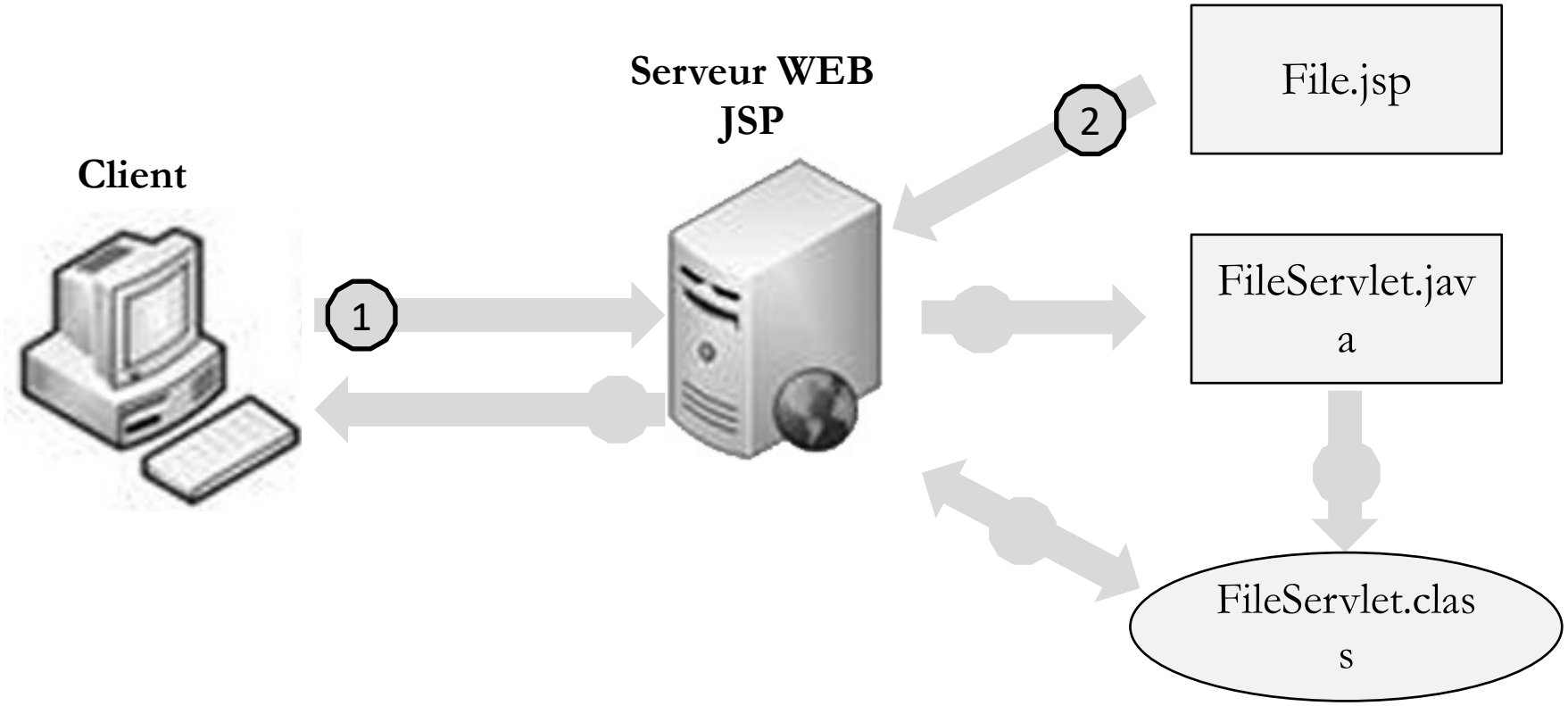
> La page HTML créée dans son ensemble est retournée au client Web.

JSP ?



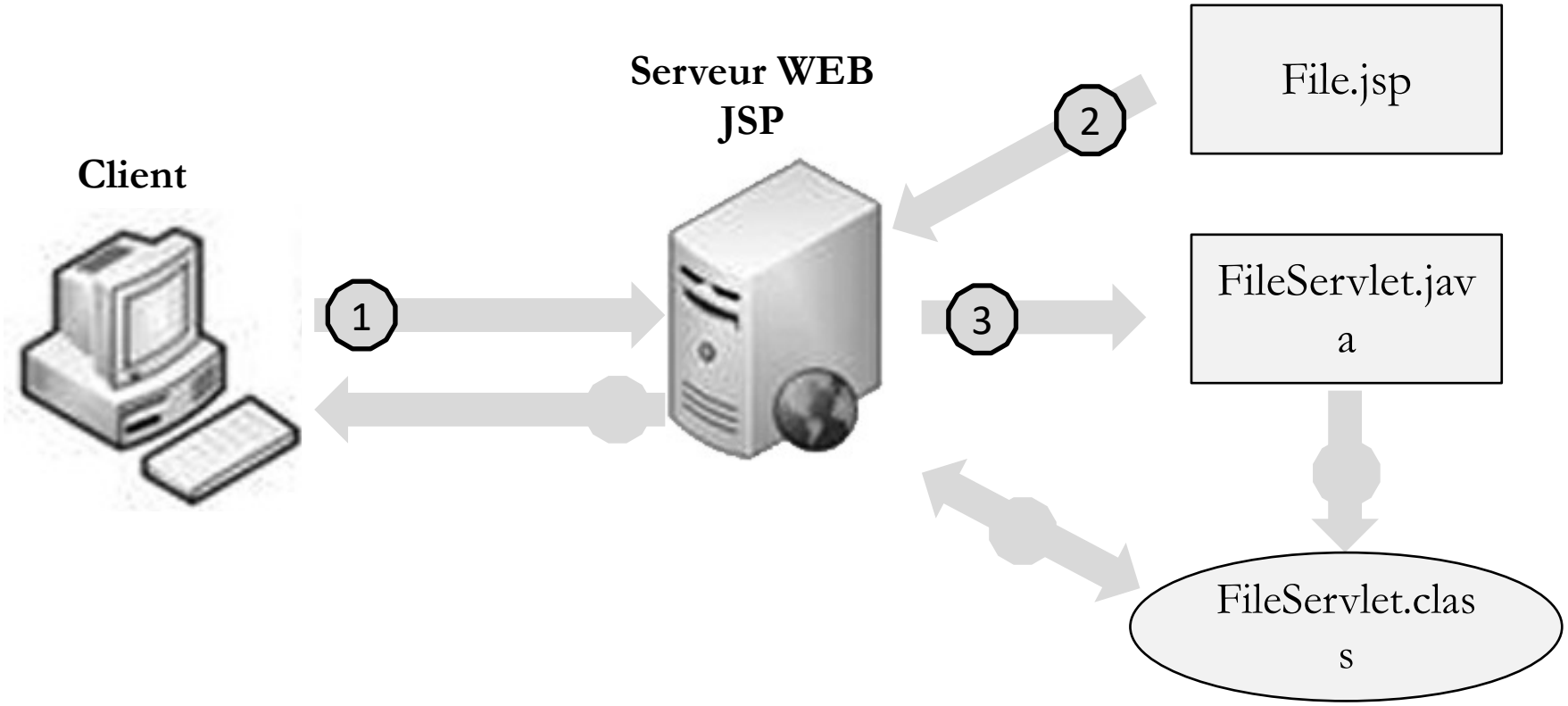
1) Comme pour les pages HTML usuelles, l'explorateur envoi une requête HTTP au serveur web qui possède la page recherchée.

JSP ?



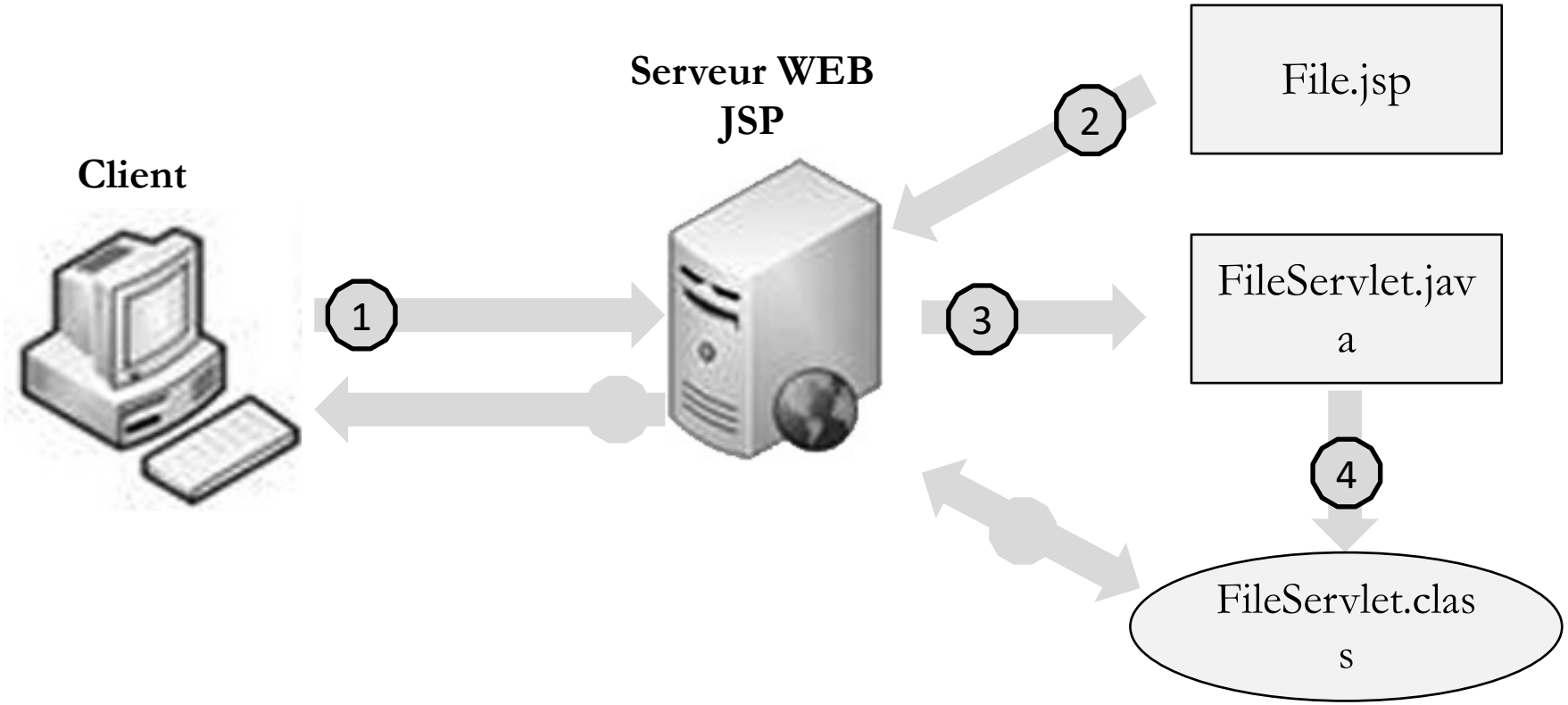
2) Le serveur WEB reconnait que la cible recherchée est la page JSP, et l'envoi au module JSP. Ceci est effectué grâce à l'URL de la page qui se termine par l'extension .jsp au lieu de .html.

JSP ?



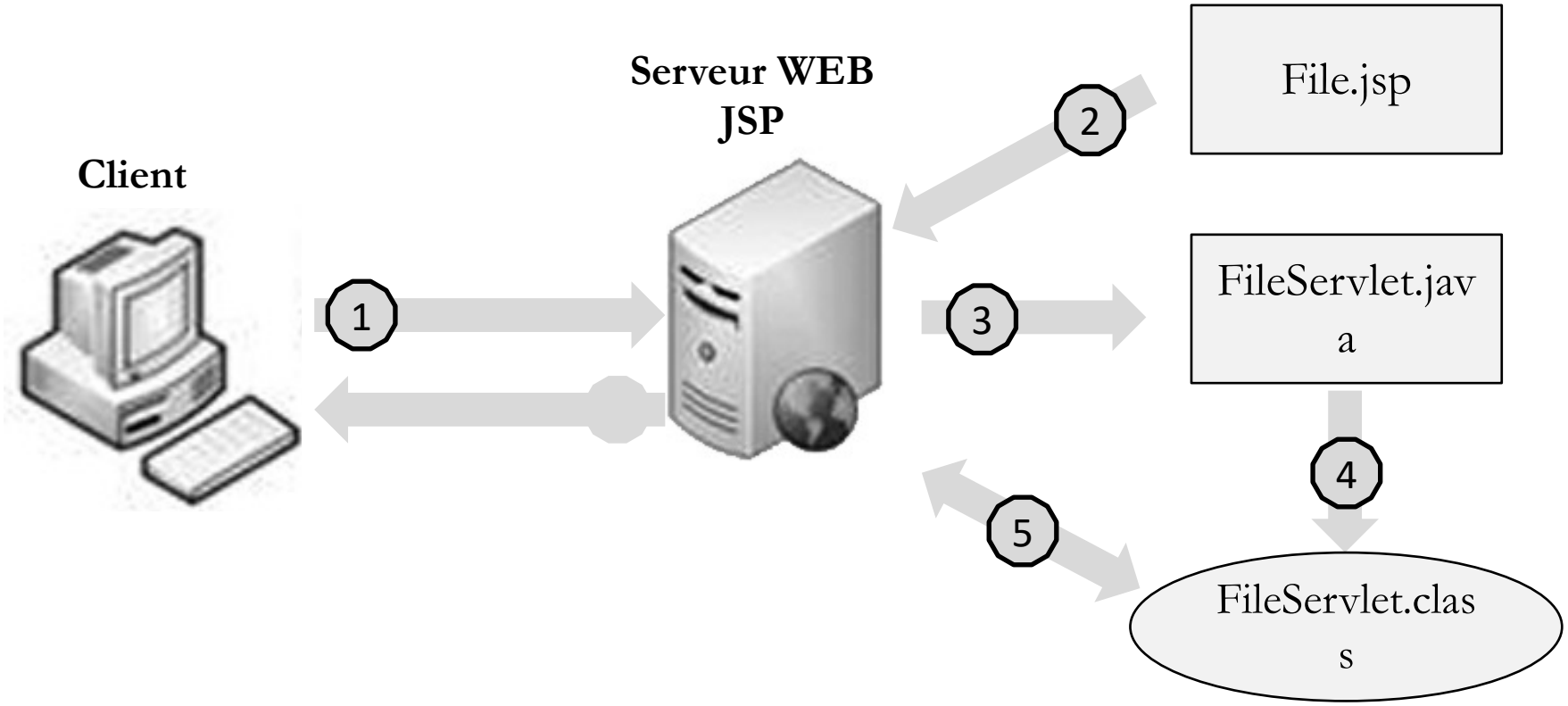
3) Le module JSP charge la page JSP du disque et la convertit en une Servlet. Cette conversion consiste à remplacer le code de la JSP en des instructions JAVA équivalentes.

JSP ?



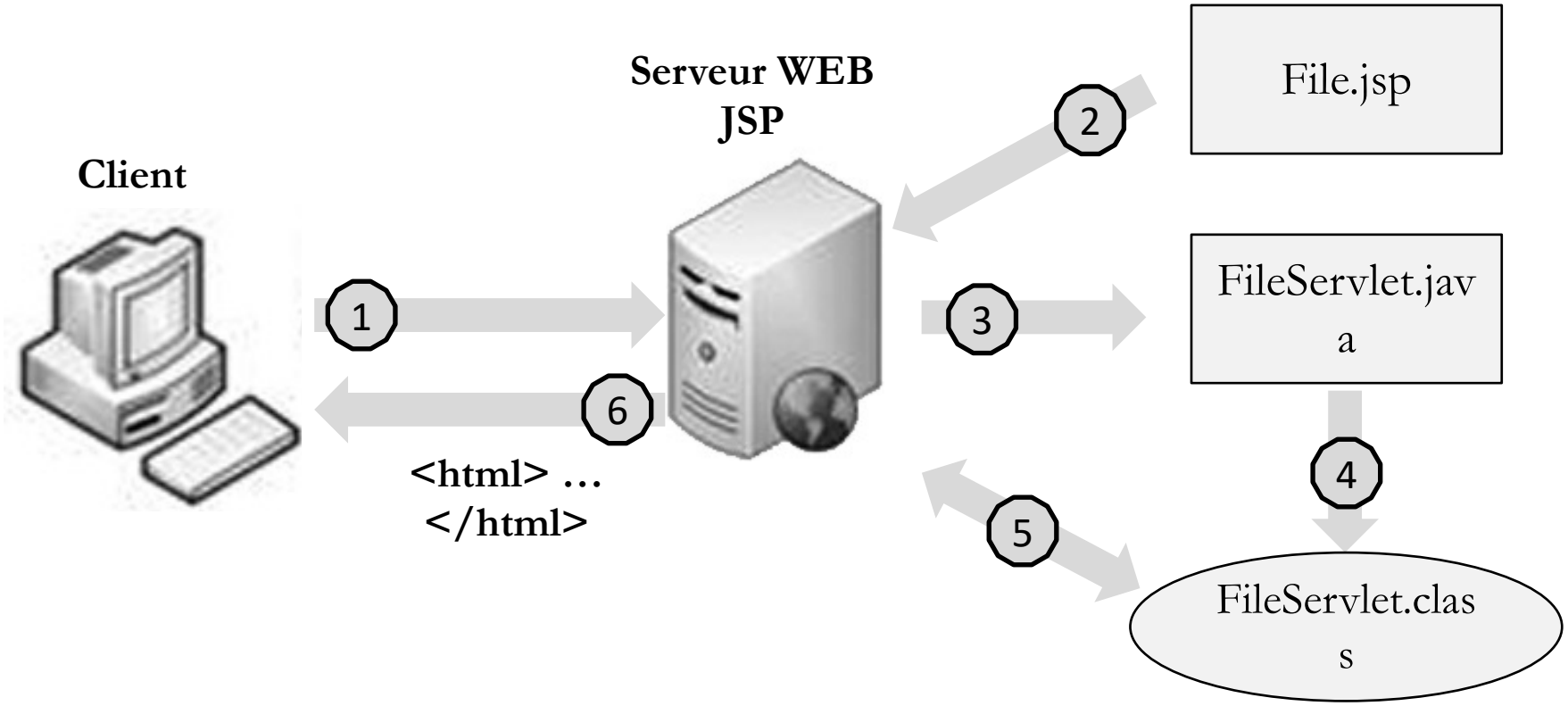
4) Le module JSP compile la Servlet résultante en un ficher .class, et envoi la requête originale vers le module JSP.

JSP ?



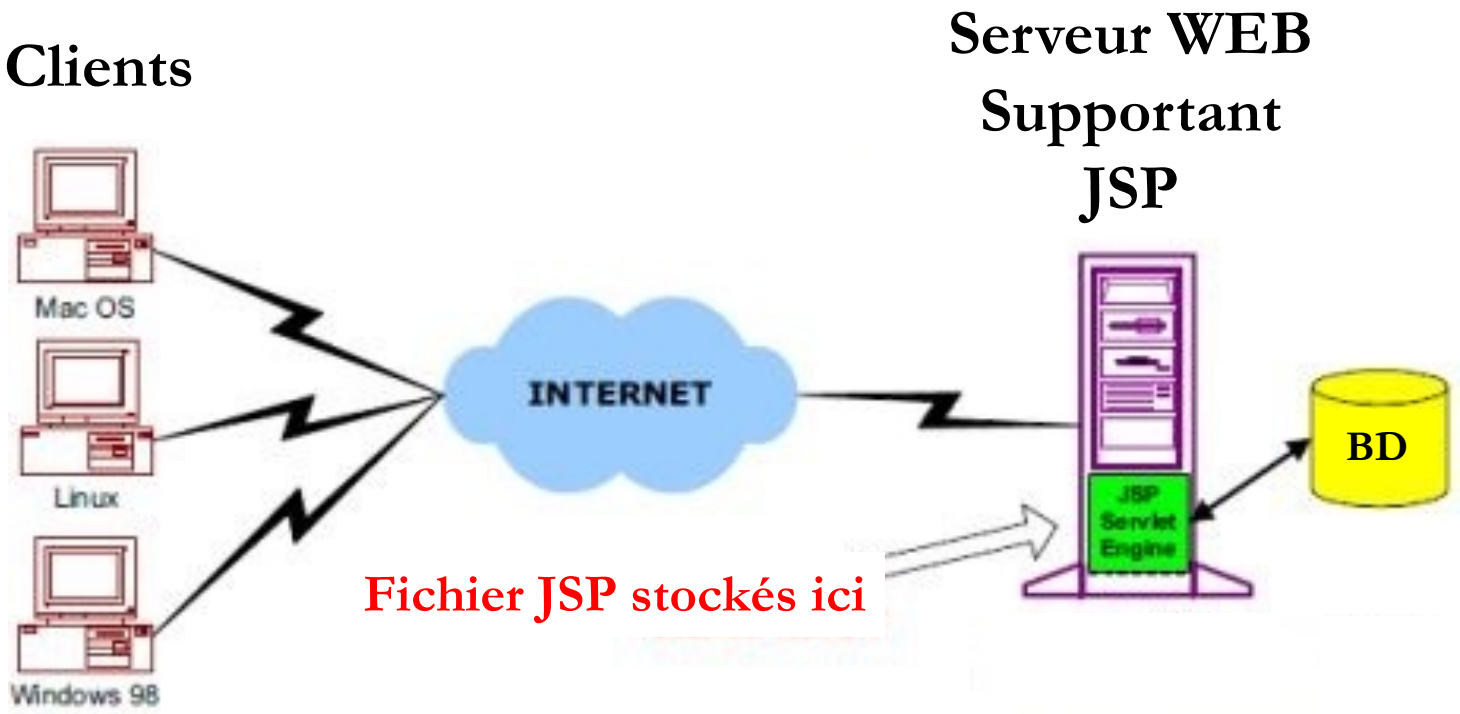
5) Le module des Servlets charge le fichier .class et l'exécute. Pendant l'exécution, la Servlet produit un résultat en format HTML, qui sera envoyé par le module des Servlets dans la réponse HTML.

JSP ?



6)Le serveur WEB envoi la réponse HTTP au explorateur comme si c'était dés le début une page HTML statique.

Outils pour JSP



- Serveur Web HTTP (Apache, Netscape Enterprise Server …)
- Conteneur de JSP (Tomcat …)
- JDK (Java Development Kit) contient un "Java Runtime Environment" (machine virtuelle JRE), un compilateur …

Premier exemple JSP

PremierExample.jsp

```
<%@ page language="java" %>
<html> <head> <title> Sans Titre </title>  </head>
        <body>
            <h1>Résultat</h1>
            <% int x = 40;%>
            <%= x+" Drhms valent "+(float)x/11+" Euros"%>
        </body>
</html>
```

Premier exemple JSP

PremierExample.jsp

```
<%@ page language="java" %>
<html> <head> <title> Sans Titre </title>  </head>
        <body>
            <h1>Résultat</h1>
            <% int x = 40;%>
            <%= x+" Drhms valent "+(float)x/11+" Euros"%>
        </body>
</html>
```

- Traité quand le client demande l'URL de la JSP :

  http://serveurWeb:<port>/.../PremierExample.jsp

Comment ça marche !

- Toute la page HTML est convertie en une servlet.

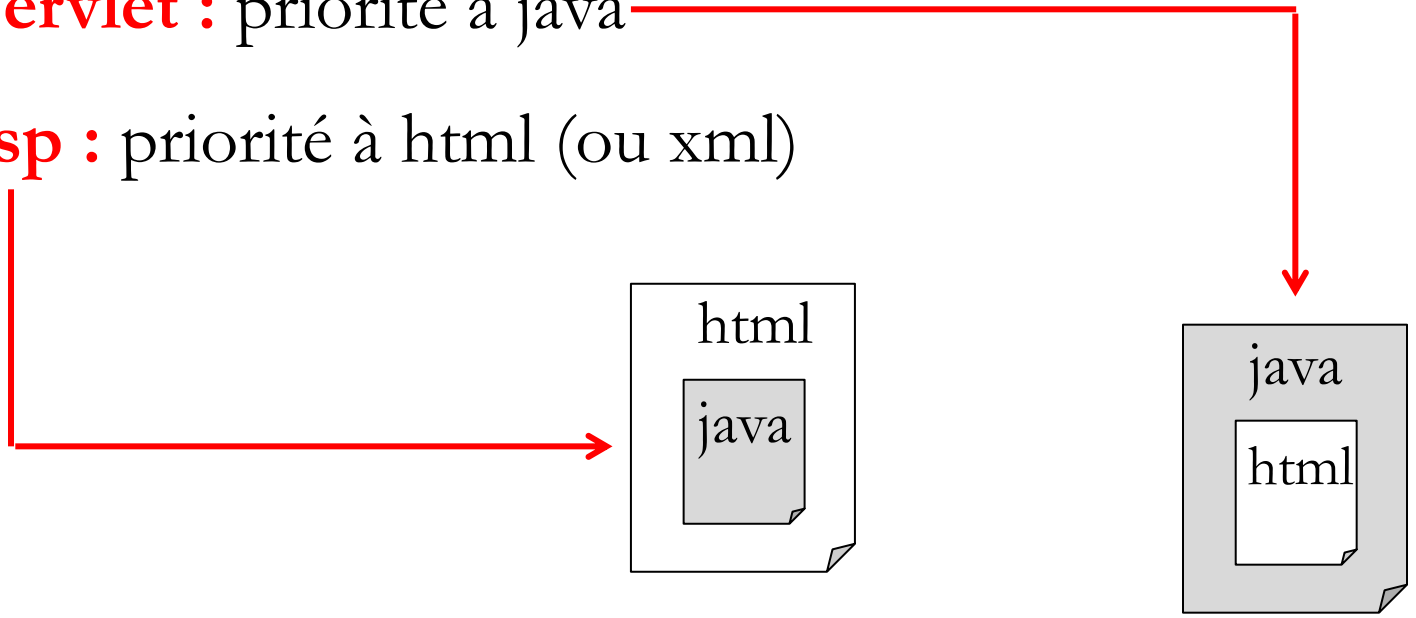
```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
public final class PremierExample_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
            private static final javax.servlet.jsp.JspFactory _jspxFactory =
            javax.servlet.jsp.JspFactory.getDefaultFactory();
            private static java.util.Map<java.lang.String,java.lang.Long>jspx_dependants; }
            private javax.el.ExpressionFactory _el_expressionfactory;
            private org.apache.tomcat.InstanceManager _jsp_instancemanager;
            public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
                        return _jspx_dependants; }
            public void _jspInit() {
                  _el_expressionfactory =
                  _jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).
                  getExpressionFactory();
                     _jsp_instancemanager =
                  org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServ
                  letConfig());
                  …….
}
```

Comment ça marche !

- Toute la page HTML est convertie en une servlet.
- Cette servlet est traitée par le moteur Java intégré au serveur Web (technologie des servlets) et retourne la page HTML construite.
- Pour exécuter des JSP (resp. des servlets), il faut un moteur de JSP (resp . de servlets) dans le serveur Web.
- Ces moteurs sont des plug-in pour des serveurs Web existants.

JSP Vs. Servlet

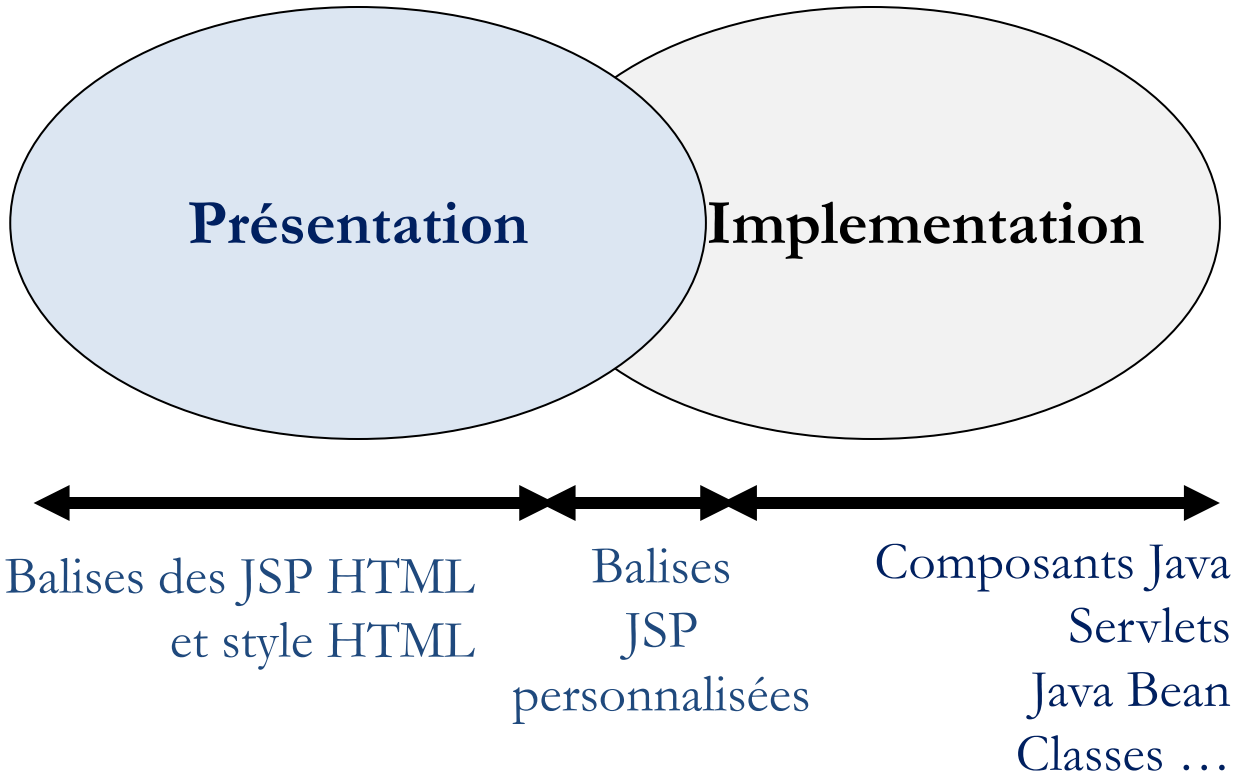
- **servlet :** priorité à java
- **jsp :** priorité à html (ou xml)



- Un même avantage : ouverture sur l'univers Java.
- les parties statiques de la page HTML sont en HTML.
- les parties dynamiques de la page HTML sont en Java.

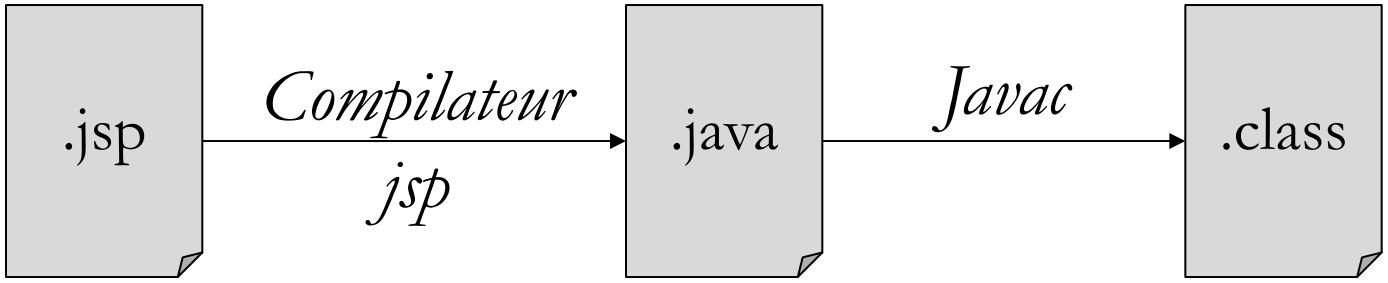
Présentation / Implémentation

- Séparation de la présentation et de l'implémentation :



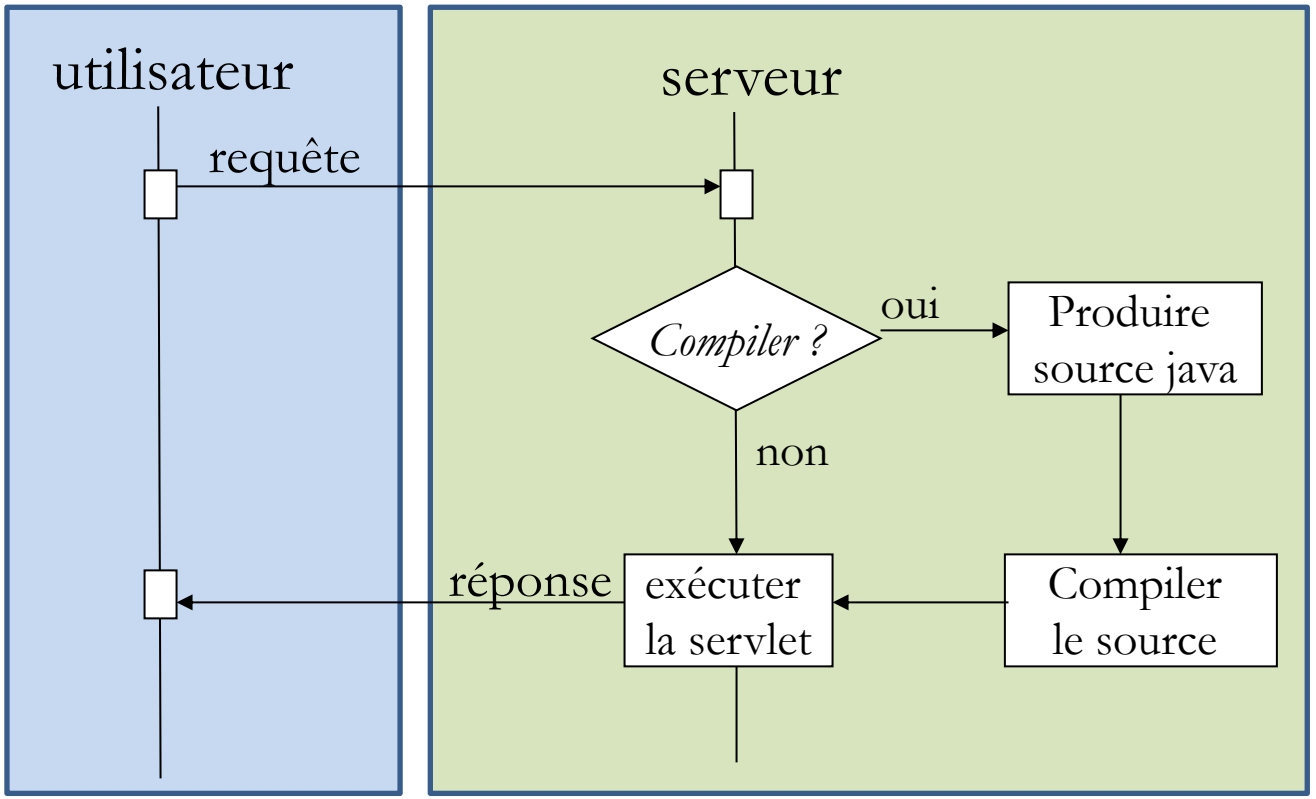
Exécution JSP

- On dépose les pages jsp à coté des pages html.
- Pas de compilation au moment du dépôt.
- Visibilité identique aux pages html.
- url : pas déclaration de service - nomDePage.jsp.
- Compilation automatique en dynamique sur le serveur WEB.



Exécution JSP

- La compilation ne se fait qu'une fois (la première).
- Le premier appel prend plus de temps (normalement) que pour les autres appels qui suivent.



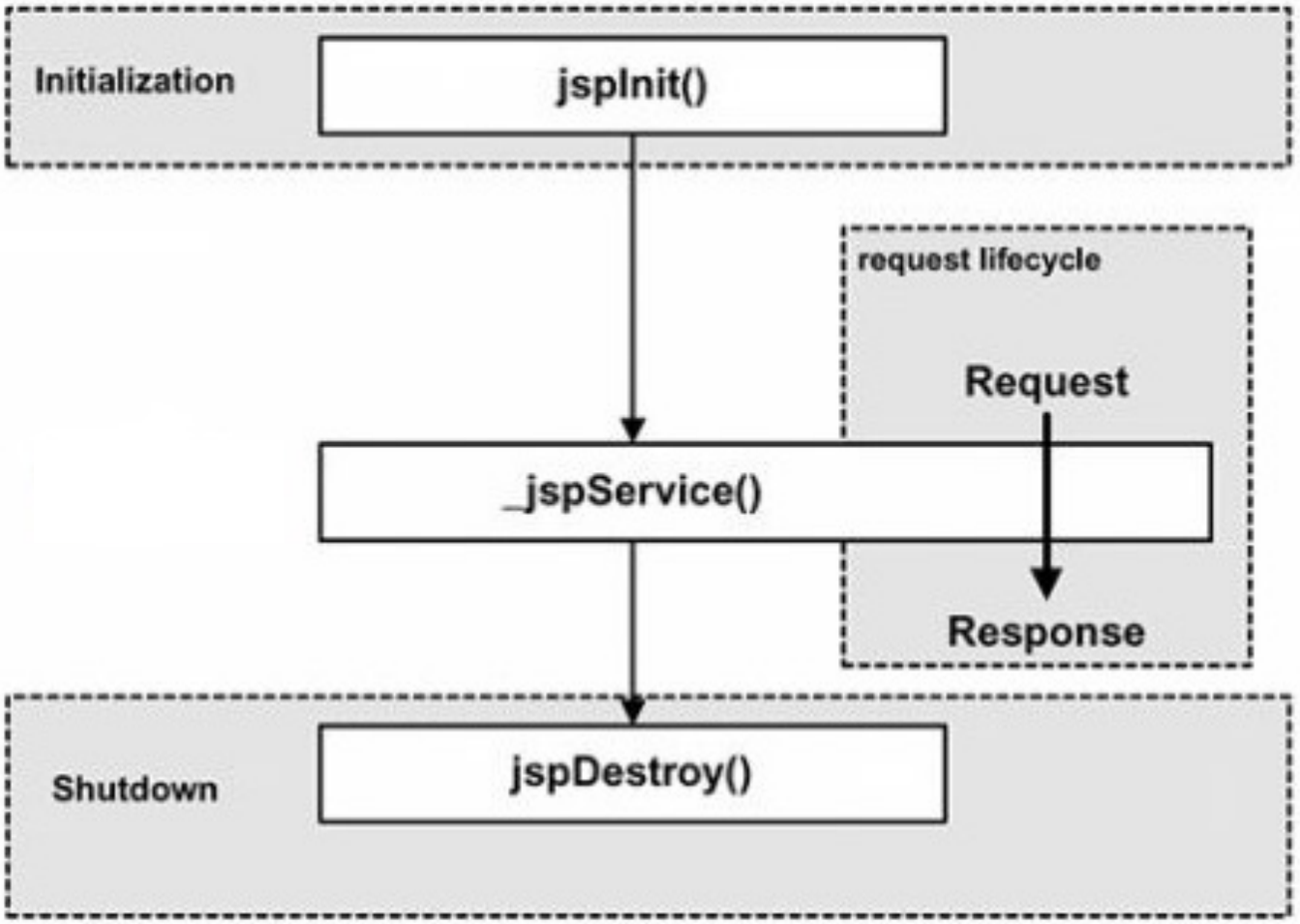
Rôle page JSP

- Une page jsp s'occupe de la sortie :
  - Lire les données expédiées par le demandeur,
  - Lire les méta-infos sur la demande / demandeur,
  - Calculer la réponse (accès aux SGBD …),
  - Formater la réponse (généralement en html),
  - Envelopper la réponse http (type de document en retour, cookies …),
  - Transmettre le document dans un format adapté : texte (eg. html), binaire (eg. gif), compressé (eg. gzip).

Qu'est-ce qui est généré ?

- Un programme java contenant une classe qui implémente l'interface *HttpJspPage*.
- Une interface qui hérite de *JspPage* qui elle même hérite de *Servlet*.
- Son point d'entrée est la méthode *_jspService* .

Comment ça marche !



Eléments Page JSP

- Directives

```
<% @ ... %>
```

- Déclarations (attributs et méthodes)

```
<% ! Déclaration %>
```

- Scriptlets

```
<% Bloc d'instructions %>
```

- Expressions

```
<% = Expression JAVA %>
```

Directives

- Syntaxe

```
<%@ directive attribut="valeur" %>
```

- Trois directives : *page, include, Taglib*
- Exemples :

```
<%@ page language = "java" %>
<%@ page buffer="5" autoFlush="false" %>
<%@ page import=" java.sql.*, cart.* " %>
<%@ include file= " foo.jsp " %>
<%@ taglib
        uri= "http://java.sun.com/jstl/core"
        prefix = "c" %>
```

Directives

- Syntaxe

<%@ page attribut="valeur" %>

buffer	Le mode du buffer du flux de sortie
autoFlush	Controle le comportement du flux de sortie de la Servlet.
contentType	Défini le schéma de codage des caractères.
errorPage	Détermine le chemin d'une page cible en cas d'erreur.
isErrorPage	Indique si cette page JSP est une page d'erreur d'une autre page .
extends	Spécifi une super classe que la Servlet doit généraliser.
import	Liste des packages ou classes à utiliser dans la JSP.
info	Définie une chaine accessible depuis la Servlet par la fonction getServletInfo().
language	Définie le language utilisé dans la page JSP.
session	Détermine si la page JSP participe en une session HTTP.

Directives

- Syntaxe

```
<%@ page attribut="valeur" %>
```

```
<%@ include file = "relative URL"  %>
```

Directives

- Syntaxe

```
<%@ page attribut="valeur" %>
```

```
<%@ include file = "relative URL"  %>
```

```
<%@ taglib uri="uri" prefix="prefixTag" > %>
```

Une JSP Taglib est une collection d'actions prédéfinies destinée à être utilisée dans une page JSP sous forme de tags (balises XML).

Déclaration

- Sont des déclarations Java.
- Seront insérées comme des membres de la servlet.
- Permet de définir des méthodes ou des données membres.

```
<%!
        private int i;
        int  Aleatoire() {
            …
        }
        public void fonction() {
            …
        }
%>
```

Déclaration

- Sont des déclarations Java.
- Seront insérées comme des membres de la servlet.
- Permet de définir des méthodes ou des données membres.

```
<%!     int i = 0;     %>
<%!     int a, b, c;     %>
<%!     Personne P = new Personne();     %>
```

Exemple JSP

```
<%    java.util.Date d = new Date();   %>
on est le
<%   if (d.getHours() < 12) {  %>   matin
<%   } else if (d.getHours()<14) {  %>   midi
<%   } else if (d.getHours()<17) {  %>   après-midi
<%   } else {   %>   soir   <%   }   %>
```

Exemple JSP

```
<%   java.util.Date d = new Date();   %>
on est le
<%   if (d.getHours() < 12) {  %>   matin
<%   } else if (d.getHours()<14) {  %>   midi
<%   } else if (d.getHours()<17) {  %>   après-midi
<%   } else {   %>   soir   <%   }   %>
```

Code généré

```
java.util.Date d= new Date();
out.println("on est le");
if (d.getHours() < 12) {          out.println("matin");          }
} else if (d.getHours()<14) {     out.println("midi");           }
} else if (d.getHours()<17) {     out.println("après-midi");   }
} else {  out.println("soir");      }
```

Exemple 2 JSP

```
Les éléments de t sont:
<ul>  <%     int [] t = {1,5,2,9,0};
             for ( int i=0 ;  i<t.length ; i++) {  %>
      <li>   <% = t[i] %> </li>
             <%     }      %>
</ul>
```

Exemple 2 JSP

```
Les éléments de t sont:
<ul>  <%     int [] t = {1,5,2,9,0};
             for ( int i=0 ;  i<t.length ; i++) {  %>
      <li>   <% = t[i] %> </li>
             <%     }      %>
</ul>
```

Les elements de t sont:

- 1
- 5
- 2
- 9
- 0

Scriptlets

- contient du code Java
- Inséré dans *_jspservice*() de la servlet, donc peut utiliser *out*, *request*, *response*, etc.

```
<%
String[] langages= {"Java", "C++", "Eiffel"};
out.println("<h3>Principaux langages OO : </h3>");
for (int i=0; i <  langages.length; i++) {
      out.println("<p>" +  langages[i] + "</p>");
}
%>
```

Expression JSP

- Expression dont le résultat est sorti sur le sortie courante de la servlet.
- Une expression Java qui renvoie un objet String ou un type primitif.
- attention au ;

```
<% =    (12+1)*2        %>
```

Code généré

```
out.println(""+(12+1)*2);
```

Commentaires

- Un commentaire est un fragment de code non-interprété.
- Commentaires html / xml (générés dans la page de sortie)

<!-- Ceci est un commentaire -->

- Commentaires jsp (Jamais générés dans le page de sortie)

<% -- Ceci est un commentaire -- %>

Code généré JSP

JSP	Code JAVA généré
<html>	out.println("<html>");
<%! int i = 0; %>	int i = 0;
<% int i = 0; %>	int i = 0;
<%= i+2 %>	out.println(""+(i+2));
<!-- coucou -->	out.println("<!-- coucou -->");
<%-- salut --%>	

Interactivité

- L'intérêt des formulaires est de pouvoir introduire des données en vue d'être traitées par une logique métier.
- L'utilisation de formulaire et fait en deux temps:
  - L'utilisateur remplit un formulaire et le soumet au serveur,
  - Celui-ci extrait les informations fournies par l'utilisateur et les utilise pour construire la réponse.

Formulaires JSP

- On distingue deux méthodes pour l'utilisation des formulaires :
  - Méthode ***get*** :
    - ❖ Méthode par défaut pour envoyer des informations du *browser* vers le serveur *WEB*, et provoque l'utilisation d'une longue chaine de caractères apparaissant dans la barre d'adresse de la page.
    - ❖ Faudra faire attention dans ce cas s'il y'a des informations confidentielles (*Password*) qui seront envoyées vers le serveur.
    - ❖ Au max 1024 caractères peuvent être envoyés ainsi.

Formulaires JSP

- On distingue deux méthodes pour l'utilisation des formulaires :
  - Méthode ***post*** :
    - ❖ Méthode plus fiable pour passer des informations.
    - ❖ Cette méthode encapsule les informations de la même manière que pour la méthode ***get***, mais au lieu de transmettre ces informations sous forme de String dans l'URL, elles sont envoyées dans un message séparé.

Formulaires JSP

- La récupération de la valeur d'un paramètre s'effectue à travers l'objet ***request***.
- Cet objet dispose d'une méthode ***getParameter()*** qui cherche la valeur d'un paramètre d'un formulaire.

Formulaires JSP

- La récupération de la valeur d'un paramètre s'effectue à travers l'objet ***request***.
- Cet objet dispose d'une méthode ***getParameter()*** qui cherche la valeur d'un paramètre d'un formulaire.

```
<%
String nomUtilisateur = request.getParameter("nom");
String MotDePasse = request.getParameter("motDePasse");
%>
```

Formulaires JSP

- La récupération de la valeur d'un paramètre s'effectue à travers l'objet ***request***.
- Cet objet dispose d'une méthode ***getParameter()*** qui cherche la valeur d'un paramètre d'un formulaire.
- Un paramètre peut avoir une valeur multiple. Dans ce cas, il faut utiliser la méthode ***getParameterValues()***.
- Cette méthode renvoie un tableau de chaînes de caractères.

Formulaires JSP

- La récupération de la valeur d'un paramètre s'effectue à travers l'objet ***request***.
- Cet objet dispose d'une méthode ***getParameter()*** qui cherche la valeur d'un paramètre d'un formulaire.
- Un paramètre peut avoir une valeur multiple. Dans ce cas, il faut utiliser la méthode ***getParameterValues()***.
- Cette méthode renvoie un tableau de chaînes de caractères.

```
<%
String[] ListeInterets = request.getParameterValues("Interets");
Out.println("<ul>");
For (int i=0; i < ListeInterets.length ; i++) {
      out.println("<li>"+ ListeInterets[i] + ''</li>'');          }
out.println(''</ul>'');                                          %>
```

Formulaires JSP

- La récupération de la valeur d'un paramètre s'effectue à travers l'objet ***request***.
- Cet objet dispose d'une méthode ***getParameter()*** qui cherche la valeur d'un paramètre d'un formulaire.
- Un paramètre peut avoir une valeur multiple. Dans ce cas, il faut utiliser la méthode ***getParameterValues()***.
- Cette méthode renvoie un tableau de chaînes de caractères.
- La méthode ***getParameterNames()*** permet d'obtenir un tableau contenant les noms de tous les paramètres présents dans la requête.

Formulaires JSP

- La méthode ***getParameter()*** renvoie une chaîne de caractères. Si le paramètre est une valeur numérique, il faut convertir la chaîne de caractères en nombre pour pouvoir l'utiliser.
- Cela se fait par la méthode JAVA appropriée :
    - ***Integer.parseInt()***
    - ***Float.parseFloat()***
    - ***Double.parseDouble()***

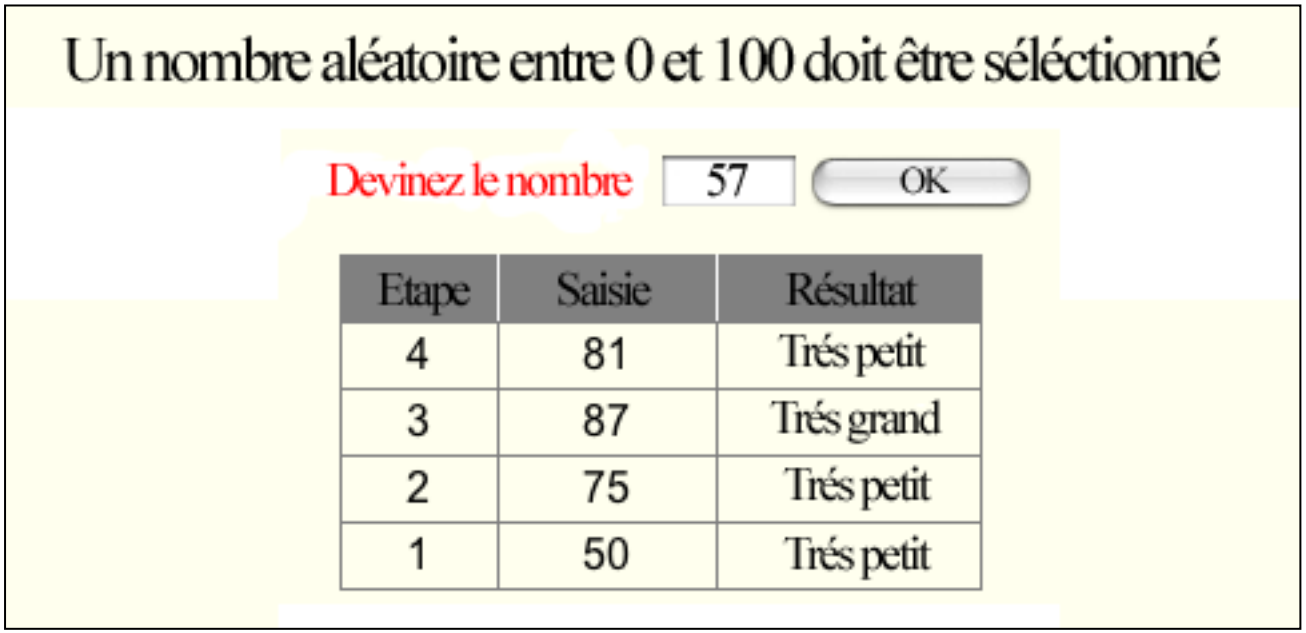
Formulaires JSP

- La méthode ***getParameter()*** renvoie une chaîne de caractères. Si le paramètre est une valeur numérique, il faut convertir la chaîne de caractères en nombre pour pouvoir l'utiliser.
- Cela se fait par la méthode JAVA appropriée :
    - ***Integer.parseInt()***
    - ***Float.parseFloat()***
    - ***Double.parseDouble()***

```
<%
String AgeText = request.getParameter("age");
int Age = Integer.parseInt(AgeText);
%>
```

Application jsp

On désire implémenter une page jsp '***JeuHasard.jsp***'. Le principe est de générer aléatoirement un nombre aléatoire (entre 0 et 100), et demander à chaque fois à l'utilisateur de saisir un nombre, jusqu'à ce que l'utilisateur tombe sur la valeur exacte. À chaque saisie erronée, on indiquera à l'utilisateur si la valeur saisie est plus grande/petite que celle attendue.



Etape	Saisie	Résultat
4	81	Trés petit
3	87	Trés grand
2	75	Trés petit
1	50	Trés petit

Sessions

- Une application web est basée sur le protocole HTTP, qui est un protocole dit "sans état" : une fois que le serveur a envoyé une réponse à la requête d'un client, ne conserve pas les données le concernant.

Sessions

- Le concept de session permet au serveur de mémoriser des informations relatives au client, d'une requête à l'autre.
- la session représente un espace mémoire alloué pour chaque utilisateur, permettant de sauvegarder des informations tout le long de leur visite.
- le contenu d'une session est conservé jusqu'à ce que l'utilisateur ferme son navigateur, reste inactif trop longtemps, ou encore lorsqu'il se déconnecte du site.
- l'objet Java sur lequel se base une session est l'objet ***HttpSession***.

Sessions

```
// Création ou récupération de la session
<%
    HttpSession session = request.getSession();
%>
// Mise en session d'une variable de type chaîne de caractères
<%
    String exemple = "azerty";
    session.setAttribute( "chaine", exemple );
%>
// Récupération de la variable de session
<%
    String chaine = (String) session.getAttribute("chaine");
%>
```

Cookies

- Le principe général est simple : il s'agit d'un petit fichier placé directement dans le navigateur du client.
- Il est envoyé par le serveur à travers les en-têtes de la réponse HTTP, et ne contient que du texte.
- Il est propre à un site ou à une partie d'un site en particulier.
- la méthode ***addCookie()*** de l'objet ***HttpServletResponse*** est utilisée pour ajouter un cookie à la réponse qui sera envoyée au client.
- la méthode ***getCookies()*** de l'objet ***HttpServletRequest*** est utilisée pour récupérer les cookies envoyés par le client.

Cookies

```
<%
    // Création d'une cookie avec une clé/valeur
    Cookie c = new Cookie("clé", "valeur");
    // Durée de vie de la cookie (en secondes)
     c.setMaxAge(24*60*60);
    // Enregistrement de la cookie
    response.addCookie(c);
%>
```

Cookies

```
// Récupération des cookies enregistrées
<%
    Cookie[] cookies = request.getCookies();
%>
// Recherche de la cookie, et récupération du contenu
<%
for(int i = 0; i < cookies.length; i++) {
    Cookie c = cookies[i];
    if (c.getName().equals("NomCookieRecherchée")) {
        String valeur = c.getValue();
    }
}
%>
```

Application Panier

On désire implémenter une page jsp '***Panier.jsp***'. Dans cette page, l'utilisateur doit saisir le nom de l'article demandé, la quantité requise, ainsi que le prix unitaire de l'article concerné. Une fois il valide par le clic sur le bouton ***'Ajouter'***, les différentes informations sur l'article doivent être ajoutées dans le tableau facture, en affichant le prix que doit payer le client.

Nom de l'article **Quantité** **Prix unitaire**

Ajouter

Détails de la facture :

Article	*Quantité*	*Prix Unitaire*
Crayon	*20*	*2*
Ramette	*15*	*30*

Total à payer : 490 drhm

Nom & prénom

Date de naissance

1 ▼ 1 ▼ 1985 ▼

Sexe

M O F O

Filière

Compétences

☐JAVA ☐ C ☐C++ ☐PHP

Ajouter

Ahmed ALAOUI	5/5/1995	M	4ILTI	JAVA, C, C++, PHP
Samira KARIMI	9/2/1998	F	1IE	Aucun
Rachid TOUMI	6/11/1995	M	3ISI	JAVA, C

Gestion des erreurs JSP

- Directives **<%@ page errorPage = page.jsp %>**
  <%@ page isErrorPage = "true" %>
- La première directive permet de définir la page erreur destination.
- La deuxième est utilisée dans l'entête de la page erreur.
- Si une exception est levée, le traitement est dérouté vers la page erreur qui connaît la référence *exception* qui repère l'exception.

Gestion des erreurs JSP

```
<%@ page errorPage="PageErreur.jsp" %>
<%
String[] langages= {"Java", "C++", "Eiffel"};
out.println("<h3> Principaux langages OO : </h3>");
for (int i=0; i <  langages.length+1; i++) {
      out.println("<p>" +  langages[i] + "</p>");
}
%>
```

Gestion des erreurs JSP

```
<%@ page errorPage="PageErreur.jsp" %>
<%
String[] langages= {"Java", "C++", "Eiffel"};
out.println("<h3> Principaux langages OO : </h3>");
for (int i=0; i <  langages.length+1; i++) {
      out.println("<p>" +  langages[i] + "</p>");
}
%>
```

PageErreur.jsp

```
<%@ page isErrorPage="true" %>
<p>Désolé, une erreur s'est produite : </p>
<pre>
<%      exception.printStackTrace(response.getWriter());
     %>
</pre>
```

Gestion des erreurs JSP

```
<%
String[] langages= {"Java", "C++", "Eiffel"};
out.println("<h3> Principaux langages OO : </h3>");
try{
     for (int i=0; i <  langages.length+1; i++) {
          out.println("<p>" +  langages[i] + "</p>");
     }
}catch(Exception e){
      out.println("Une exeption levée : " + e.toString());
}
%>
```

Enchainement des pages JSP

- Un page JSP peut en appeler une autre par la directive :

<jsp:forward page="pageDestination" />

Enchainement des pages JSP

- Un page JSP peut en appeler une autre par la directive :

<jsp:forward page="pageDestination" />

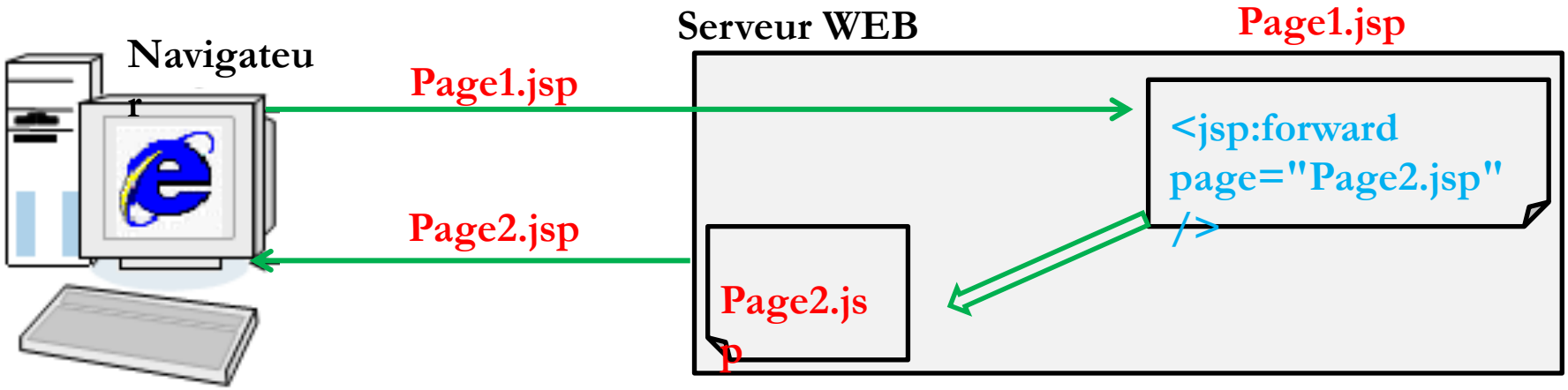
```
<%
String Reponse=request.getParameter("ValeurSaisie");
int r = Integer.parseInt(Reponse);
if ( r > 0) {
%>
<jsp:forwardpage= "positif.jsp"/>
<% }else{ %>
<jsp:forwardpage="negatif.jsp"/>
<% } %>
```

Enchainement des pages JSP

- Un page JSP peut en appeler une autre par la directive :

<jsp:forward page="pageDestination" />

- Après un **<jsp:forward>**, le traitement est entièrement pris en charge par la nouvelle page.



Actions

- Une action JSP permet en utilisant une syntaxe XML de contrôler le comportement du module de Servlet. Vous pouvez ainsi y insérer un fichier, réutiliser les composantes des JAVA Beans, envoyer l'utilisateur vers une autre page, ou générer du code HTML pour le plugin JAVA.

<jsp : NomAction attribute = "value" />

Actions

<jsp : NomAction attribute = "value" />

include	Inclure un fichier lorsque la page est sollicitée.
useBean	Chercher ou instancier une Bean JAVA.
setProperty	Modifier la propriété d'une Bean JAVA.
getProperty	Récupérer une propriété d'une Bean JAVA.
Forward	Rediriger l'utilisateur vers une nouvelle page.

<jsp : include page="relative URL" flush="true" />

Actions

<jsp : NomAction attribute = "value" />

include	Inclure un fichier lorsque la page est sollicitée.
useBean	Chercher ou instancier une Bean JAVA.
setProperty	Modifier la propriété d'une Bean JAVA.
getProperty	Récupérer une propriété d'une Bean JAVA.
Forward	Rediriger l'utilisateur vers une nouvelle page.

<jsp:useBean id="idBean" class= "ClassBean" />

Actions

<jsp : NomAction attribute = "value" />

include	Inclure un fichier lorsque la page est sollicitée.
useBean	Chercher ou instancier une Bean JAVA.
setProperty	Modifier la propriété d'une Bean JAVA.
getProperty	Récupérer une propriété d'une Bean JAVA.
Forward	Rediriger l'utilisateur vers une nouvelle page.

Actions

<jsp : NomAction attribute = "value" />

include	Inclure un fichier lorsque la page est sollicitée.
useBean	Chercher ou instancier une Bean JAVA.
setProperty	Modifier la propriété d'une Bean JAVA.
getProperty	Récupérer une propriété d'une Bean JAVA.
Forward	Rediriger l'utilisateur vers une nouvelle page.

Actions

<jsp : NomAction attribute = "value" />

include	Inclure un fichier lorsque la page est sollicitée.
useBean	Chercher ou instancier une Bean JAVA.
setProperty	Modifier la propriété d'une Bean JAVA.
getProperty	Récupérer une propriété d'une Bean JAVA.
forward	Rediriger l'utilisateur vers une nouvelle page.

Objets JSP

- Objets accessibles automatiquement dans une page jsp:

Variable	Types JAVA
request	***HttpServletRequest***
response	***HttpServletResponse***
session	***HttpSession***
application	***ServletContext***
out	Objet de type ***PrintWriter*** utilisé pour l'envoi de flux de sortie vers le client.
pageContext	***PageContext***
config	Objet ***ServletConfig*** associé à la page.
page	Synonyme de ***this***, utilisé pour l'appel des méthodes dans la classe ***servlet*** traduite.
exception	L'objet de type ***Exception.***

Objets JSP

- 3 objets peuvent être immédiatement utilisés dans une expression ou une scriptlet d'une JSP :
  - out : le canal de sortie.
  - Request (HttpServletRequest) : l'objet requête.
  - response (HttpServletResponse) : l'objet réponse.

Débogage JSP

- La fenêtre de lancement du serveur Web donne des indications. Suivant les serveurs, une page HTML est retournée avec des indications.
- Ces éléments sont très souvent relatifs à la Servlet et pas à la page JSP.

Expression Langage

- Les pages JSP disposent de leur propre langage d'expression (EL).

Catégorie	Opérateurs
Variable	**.** et **[]**
Arithmétique	**+**,**-**, ***, **/, div**, **%**, **mod** et **-**
Logique	**and**, **&&**, **or**, **||**,**not**,**!**
Relationnel	**==**, **eq**, **!=**, **ne**, **<**, **lt**, **>**, **gt**, **<=**, **le**, **>=**, **ge**
Conditionnel	A **?** B **:** C
Variable vide	**empty**

Expression Langage

- Les pages JSP disposent de leur propre langage d'expression (EL).

Catégorie	Opérateurs
Variable	. et []
Arithmétique	+,-, *, /, div, %, mod et -
Logique	and, &&, or,||,not,!
Relationnel	==, eq, !=, ne, <, lt, >, gt, <=, le, >=, ge
Conditionnel	A ? B : C
Variable vide	empty

```
<% String AgeText = request.getParameter("age");
out.println(AgeText);%>
L'expression EL équivalente est :
${param.age}  ou bien  ${param["age"]}
```

Expression Langage

EL expression	Valeur de retour
${1 > (4/2)}	false
${4.0 >= 3}	true
${100.0 == 100}	true
${(10*10) ne 100}	false
${'a' < 'b'}	true
${'hip' gt 'hit'}	false
${4 > 3}	true
${1.2E4 + 1.4}	12001.4
${3 div 4}	0.75
${10 mod 4}	2
${!empty param.Add}	false si NULL sinon true
${header["host"]}	The host

Java Beans

- But : avoir le moins de code Java possible dans une page JSP (HTML).
- Sous-traiter le code à un Java *bean*.
- balise XML : **< jsp : useBean >**

Java Beans

- But : avoir le moins de code Java possible dans une page JSP (HTML).
- Sous-traiter le code à un Java *bean*.
- balise XML : **< jsp : useBean >**
- Syntaxe générale :

```
<jsp:useBean
        id="nomInstanceJavaBean"
        class="nomClasseDuBean"
        scope="request|session|application|page">
</jsp:useBean>
```

- Le *bean* est alors utilisable par **nomInstanceJavaBean**

Java Beans

- Le tag <jsp:useBean> permet de localiser une instance ou d'instancier un *bean* pour l'utiliser dans la JSP.
- Un bean peut encapsuler des traitements complexes et être réutilisable par d'autre JSP ou composants: par exemple, assurer l'accès à une base de données.
- L'utilisation des *beans* permet de simplifier les traitements inclus dans la JSP.

Particularité

- un *bean* est un simple objet Java qui suit certaines contraintes :
  - Les propriétés - un *bean* est conçu pour être paramétrable. On appelle "propriétés" les champs non-publics présents dans un *bean*. Qu'elles soient de type primitif ou objets, les propriétés permettent de paramétrer le *bean*, en y stockant des données.

Particularité

- un *bean* est un simple objet Java qui suit certaines contraintes :
  - La sérialisation : un *bean* est conçu pour pouvoir être persistant. La sérialisation est un processus qui permet de sauvegarder l'état d'un *bean,* et donne ainsi la possibilité de le restaurer par la suite.

Particularité

- un *bean* est un simple objet Java qui suit certaines contraintes :
  - La réutilisation : un bean est un composant conçu pour être réutilisable. Ne contenant que des données ou du code métier, un tel composant n'a en effet pas de lien direct avec la couche de présentation, et peut également être distant de la couche d'accès aux données. C'est cette indépendance qui lui donne ce caractère réutilisable.

Java Beans

```
<jsp:useBean
        id="nomInstanceJavaBean"
        class="nomClasseDuBean"
        scope="request|session|application|page
            ">
</jsp:useBean>
```

- L'attribut scope permet de déterminer la portée du *bean*.

Java Beans

```
<jsp:useBean
        id="nomInstanceJavaBean"
        class="nomClasseDuBean"
        scope="request|session|application|page
            ">
</jsp:useBean>
```

- L'attribut scope permet de déterminer la portée du bean.

request	Le *bean* est valide pour cette requête. Il est utilisable dans les pages de redirection de la requête (<jsp : forward>), et il est détruit à la fin de la requête. le *bean* est alors accessible durant la durée de vie de la requête.

Java Beans

```
<jsp:useBean
        id="nomInstanceJavaBean"
        class="nomClasseDuBean"
        scope="request|session|application|page
            ">
</jsp:useBean>
```

- L'attribut scope permet de déterminer la portée du bean.

Page	Le *bean* est utilisable dans toute la page JSP ainsi que dans les fichiers statiques inclus. C'est la valeur par défaut.

Java Beans

```
<jsp:useBean
        id="nomInstanceJavaBean"
        class="nomClasseDuBean"
        scope="request|session|application|page
            ">
</jsp:useBean>
```

- L'attribut scope permet de déterminer la portée du bean.

session	Le *bean* est valide pour la session courante. S'il n'existe pas encore dans la session courante, il est créé et placé dans la session du client. le *bean* est utilisable par toutes les JSP qui appartiennent à la même session que la JSP qui a instanciée le *bean*.

Java Beans

```
<jsp:useBean
        id="nomInstanceJavaBean"
        class="nomClasseDuBean"
        scope="request|session|application|page
            ">
</jsp:useBean>
```

- L'attribut scope permet de déterminer la portée du bean.

application	Le *bean* est valide pour l'application courante. Il est donc créé une seule fois et partagé par tous les clients des JSP. le *bean* est utilisable par toutes les JSP qui appartiennent à la même application que la JSP qui a instanciée le *bean*.

Bean : exemple

- Prenons l'exemple de déclaration d'un premier bean:

```
public class ExBean implements java.io.Serializable
{
    private int compt;
    public ExBean() {
            compt=0;    }
    public void setCompt(int Val) {
            compt = Val;          }
    public int getCompt() {
            return compt;         }
    public void increment() {
            compt++;      }
}
```

Bean : exemple

- Prenons l'exemple de déclaration d'un premier bean:

```
public class ExBean implements java.io.Serializable
{
    private int compt;
    public ExBean() {
        compt=0;
    }
    public void setCompt(int Val) {
        compt = Val;
    }
    public int getCompt() {
        return compt;
    }
    public void increment() {
        compt++;
    }
}
```

Sérialiser un objet consiste à le convertir en un tableau d'octets, que l'on peut ensuite écrire dans un fichier, envoyer sur un réseau au travers d'une socket etc...

Java *Bean*

- Un bean est une classe publique.
- doit avoir au moins un constructeur par défaut.
- implémente l'interface *Serializable*, afin de devenir persistant et son état peut être sauvegardé.
- ne doit pas avoir de champs publics.
- peut définir des propriétés (des champs privés), qui doivent être accessibles via des méthodes publiques *getter* et *setter*, suivant des règles de nommage.

Le tag <jsp:setProperty >

- Permet de mettre à jour la valeur d'un ou plusieurs attributs d'un *Bean*.
- Il utilise le modificateur pour mettre à jour la valeur. Le *bean* doit exister grâce à un appel au tag <jsp:useBean>.

Le tag <jsp:setProperty >

- Pour mettre à jour les propriétés :
  - Soit à travers les paramètres correspondants contenus dans la requête.

```
<jsp:setProperty name="monBean" property="*" />
```

Le tag <jsp:setProperty >

- Pour mettre à jour les propriétés :
  - alimenter automatiquement une propriété avec le paramètre correspondant dans la requête.

```
<jsp:setProperty name="monBean" property="nom" />
```

Si le nom de la propriété et du paramètre sont différents, il faut préciser l'attribut *property* et l'attribut *param* qui doit contenir le nom du paramètre qui va alimenter la propriété du *bean*.

Le tag <jsp:setProperty >

- Pour mettre à jour les propriétés :
  - alimenter une propriété directement avec la valeur particulière précisée

```
<jsp:setProperty name="monBean" property="nom"
                                        value="toto" />
```

NB: Il n'est pas possible d'utiliser à la fois *param* et *value* dans le même tag.

Le tag <jsp:getProperty >

- permet d'obtenir la valeur d'un attribut d'un Bean.
- Il utilise l'accesseur (getter) pour obtenir la valeur et l'insérer dans la page HTML générée. Le *bean* doit exister grâce à un appel au tag <jsp:useBean>.

Le tag <jsp:getProperty >

- permet d'obtenir la valeur d'un attribut d'un Bean.
- Il utilise l'accesseur (getter) pour obtenir la valeur et l'insérer dans la page HTML générée. Le *bean* doit exister grâce à un appel au tag <jsp:useBean>.
- La syntaxe est la suivante:

```
<jsp:getProperty name="monBean" property="nom" />
```

- L'attribut name indique le nom du *bean* tel qu'il a été déclaré dans le tag <*jsp:useBean*>, alors que l'attribut *property* indique le nom de la propriété à récupérer.

Bean : exemple

L'utilisation du bean se fait par le nomBean :
<jsp:useBean id="nomBean" class="ExBean"
scope="session">
</jsp:useBean>
On accéde à une propriété avec l'expression :
compteur= <%= nomBean.getCompt() %>
On incrémente le compteur
<% nomBean.increment(); %>
On peut aussi accéder à la propriété par :
<jsp:getProperty name="nomBean" property="compt"/>

Bean : exemple

L'utilisation du bean se fait par le nomBean :
<jsp:useBean id="nomBean" class="ExBean"
scope="session">
</jsp:useBean>
<p> On peut modifier une propriéte avec l'expresion:
<% nomBean.setCompt(6); %>
ou avec la balise
<jsp:setProperty name="nomBean" property="compt"
value="8" />

Bean : exemple

On peut créer et au même temps initialiser la valeur d'une propriété d'un bean :

```
<jsp:useBean id="nomBean" class="ExBean" scope="session">
    <jsp:setProperty name="nomBean" property="compt" value=10 />
</jsp:useBean>
```

Bean : Exercice

- On désire faire une page JSP dans laquelle on procédera à l'instanciation d'un ensemble de trois personne :
  - Ahmed, né en 1990, qui habite à Casablanca.
  - Samir, né en 1988, Habitant à Mohammedia.
  - Hind, née en 1993, Habitant Rabat.
- La création de ces personnes sera faite par un bean JAVA, *PersonneBean*, qui permet de définir les trois attributs des personnes, avec les accesseurs et modificateurs, ainsi que les constructeurs de ce Bean.
- Essayez d'afficher dans la page JSP toute les informations sur ces trois personnes.

Bean : Exercice (2)

- Un premier formulaire (***pageAcceuil.html***) dans lequel on saisira les informations suivantes: Nom, email et âge.
- Créer un java bean pour modéliser l'objet Personne, avec les attributs contenus dans le formulaire.
- Une deuxième page (***Recolte.jsp***) récoltera les informations contenus dans le formulaire sous forme d'un objet, nommé Personne1. Cette page contiendra un lien vers une troisième page (***PageSuivante.jsp***).
- Cette dernière page récupérera les informations contenues dans le bean afin d'afficher les valeurs de ses propriétés.