


Java EE

- Introduction au Java EE.
- La servlet.
- Servlet avec vue.
- Transmission de données.
- Le Java Bean.
- La technologie JSP.
- Les sessions.
- Les cookies.
- Communication avec les BDD.

« Notes du cours de OpenClassrooms »

Insaf B.


Introduction au Java EE

Java EE = Java Entreprise Edition
= J2EE

→ La plateforme Java EE est construite sur:

- le langage Java.
- la plateforme Java SE qui est constituée de nombreuses bibliothèques ou API.

et elle y ajoute un grand nombre de bibliothèques.

« Echange dynamique HTTP client ↔ serveur »

② Le navigateur envoie une requête HTTP au serveur.



client



serveur

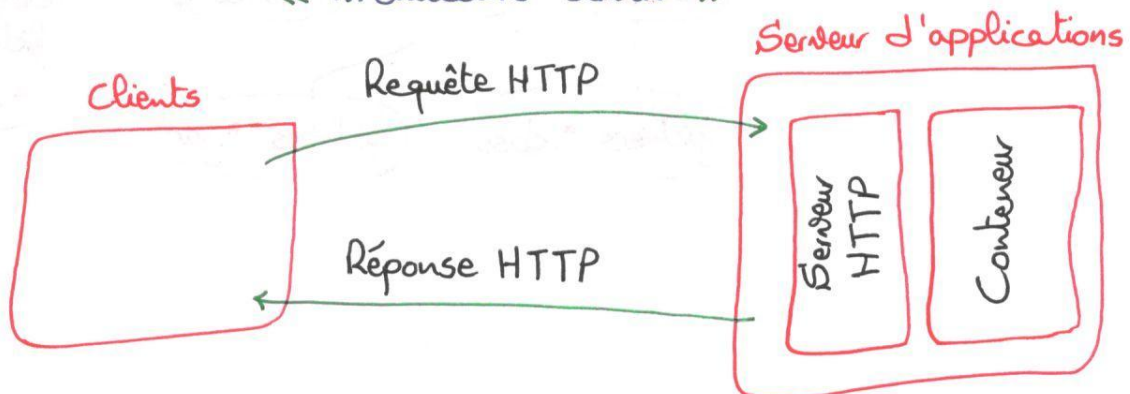
③ Le serveur traite la requête et génère la page web demandée.

① Le client saisit une URL.

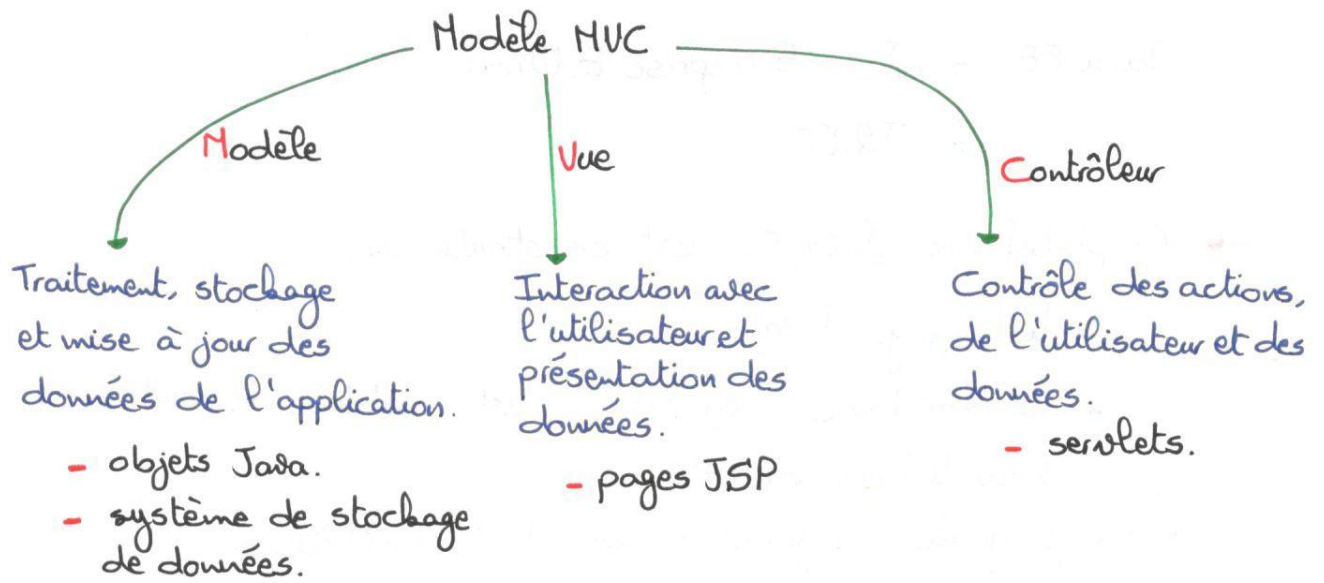
④ Le serveur renvoie une réponse HTTP au client.

∃ Plusieurs technologies pour traiter les informations sur le serveur. Java EE est l'une d'entre elles.

« Architecture serveur »



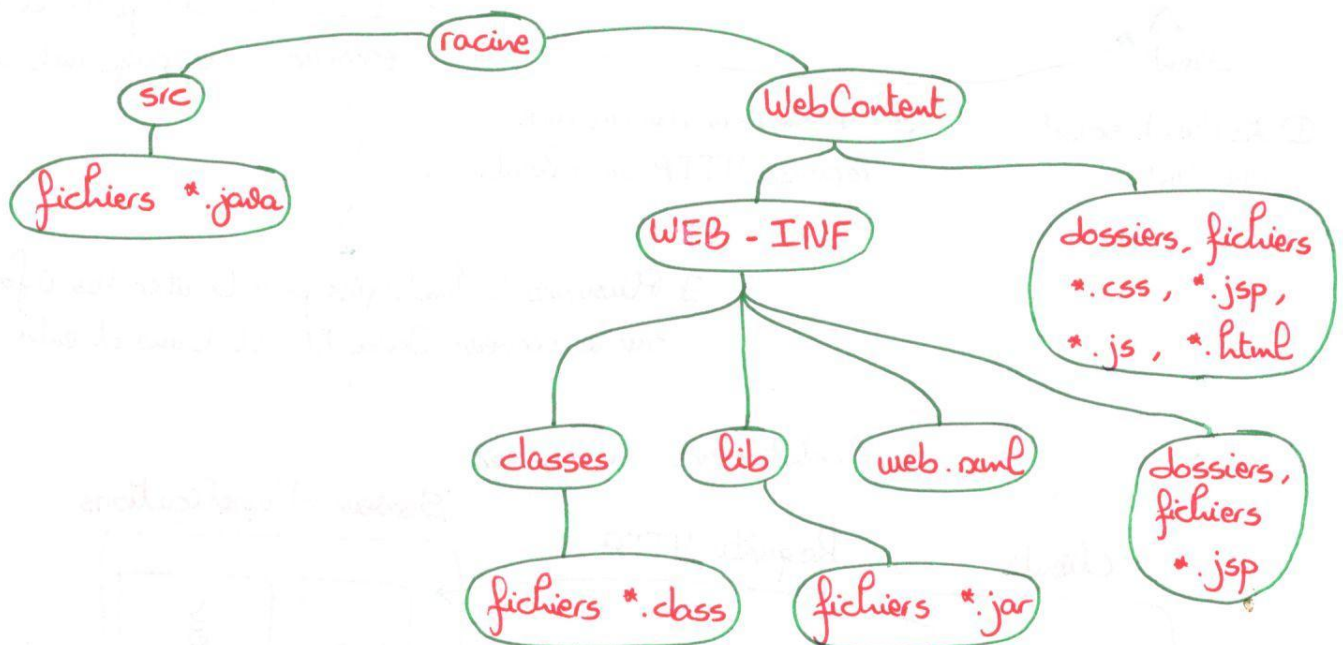
→ Modèle de conception = Design Pattern



→ Outils :

- ⊙ L'IDE Eclipse.
- ⊙ Le serveur Tomcat.

« Structure d'une application web
Java EE sous Eclipse »



La servlet

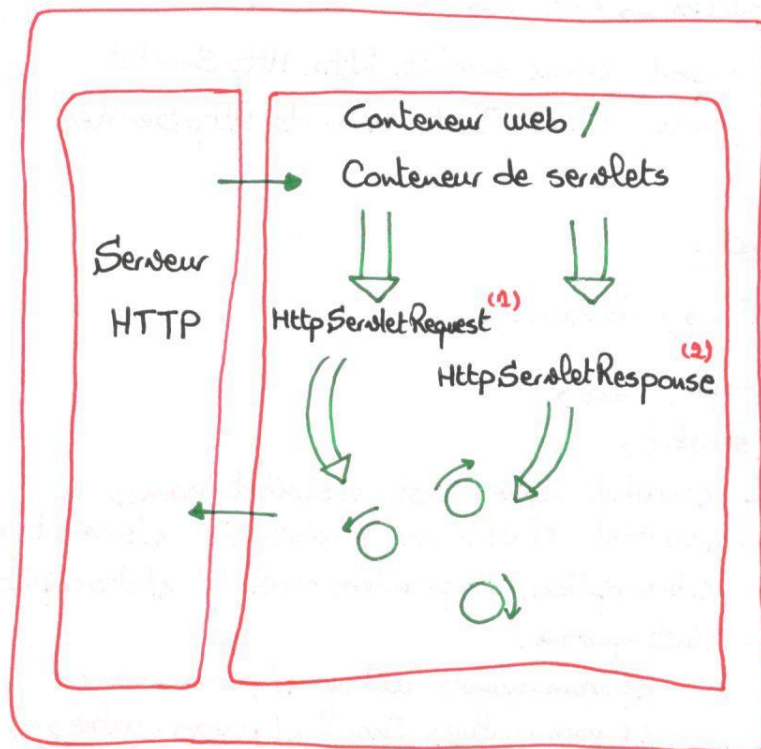
○ Retour sur HTTP:

HTTP = le langage utilisé par le client et le serveur afin de s'échanger des informations.

Il ne comprend que quelques mots appelés méthodes HTTP. On s'intéresse essentiellement à : GET - POST et HEAD.

- **GET** : Récupérer une ressource web du serveur via une URL.
- **POST** : Soumettre au serveur des données de tailles variables ou volumineuses.
- **HEAD** : Identique à GET sauf que le serveur n'y répondra pas en envoyant la ressource accompagnée des informations mais seulement ces informations (les en-têtes HTTP!).

« Conteneur et paire d'objets requête / réponse »



- (1) Cet objet contient la requête HTTP et donne accès à toutes ses informations.
- (2) Cet objet initialise la réponse HTTP qui sera renvoyée au client et permet de la personnaliser.

○ Servlet:

→ Une servlet = Une simple classe Java qui a la particularité de permettre le traitement de requêtes et la personnalisation de réponses.

≈ Une classe capable de recevoir une requête HTTP envoyée depuis le navigateur de l'utilisateur, et de lui renvoyer une réponse HTTP.

→ Une servlet HTTP doit hériter de la classe abstraite `HttpServlet`, qui propose les méthodes Java nécessaires au traitement des requêtes et réponses HTTP. Par exemple :

`doGet()` → gérer la méthode GET.

`doPost()` → gérer la méthode POST.

`doHead()` → gérer la méthode HEAD.

→ Les servlets vont être les points d'entrée de notre application web, c'est par elles que tout va se passer.

① Création:

Java Resources → src → new Class :

```
import javax.servlet.http.HttpServlet;
public class Test extends HttpServlet
{
}
```

② Mise en place:

WEB-INF → web.xml :

`<web-app ...>`

`<servlet>`

Le nom de la servlet

← `<servlet-name> Test </servlet-name>`

Le chemin de sa classe

← `<servlet-class> mes-servlets.Test </servlet-class>`

`<description> Ma première servlet ! </description>`

← `<init-param>`

`<param-name> auteur </param-name>`

`<param-value> Insaf </param-value>`

`</init-param>`

← `<load-on-startup> 1 </load-on-startup>`

`</servlet>`

`<servlet-mapping>`

`<servlet-name> Test </servlet-name>`

← `<url-pattern> /insaf </url-pattern>`

Des paramètres qui seront accessibles à la servlet lors de son chargement.

Forcer le chargement de la servlet dès le démarrage du serveur.

Optionnels.

La ou les URLs relatives au travers desquelles on

③ Mise en service:

Tentons d'accéder à :

`http://localhost:8080/test/insaf`

⇒

Etat HTTP 405 - La méthode HTTP GET n'est pas supportée par cette URL.

Il faut surcharger la méthode `doGet()` de la classe `HttpServlet` dans notre servlet `Test`.

```
public class Test extends HttpServlet
{
    public void doGet (HttpServletRequest request ,
        HttpServletResponse response) throws
        ServletException, IOException
    {
        // Contenu de doGet
    }
}
```

⇒ Page blanche.

⇒ Aal izz well ! :p

○ Cycle de vie d'un servlet :

- L'application web démarre.

- Une servlet peut :

- démarrer automatiquement.

`<load-on-startup> N </load-on-startup>`

↳ ≥ 0

ordre de priorité du démarrage des servlets.

- démarrer si demandée.

- Le conteneur de servlets va créer une instance de cette servlet et la garder en mémoire pendant toute l'existence de l'application.

- Envoyer des données au client.

« Contenu de doGet »

```
response. setContent ("text / html"); // Préciser ce que nous allons envoyer.
```

```
response. set CharacterEncoding ("UTF-8");
```

```
PrintWriter out = response. getWriter (); // Pour envoyer du texte au client.
```

```
out. println ("< DOCTYPE html >");
```

```
out. println ("<html>");
```

```
:
```

```
out. println ("</html>");
```

○ Récapitulation:

- Le client envoie des requêtes au serveur grâce aux méthodes du protocole HTTP, notamment GET, POST et HEAD.
- Le conteneur web place chaque requête reçue dans un objet `HttpServletRequest`, et place chaque réponse qu'il initialise dans l'objet `HttpServletResponse`.
- Le conteneur transmet chaque couple requête / réponse à une servlet : c'est un objet java assigné à une requête et capable de générer une réponse en conséquence.
- La servlet est donc le point d'entrée d'une application web, et se déclare dans son fichier de configuration web.xml.
- Une servlet peut se charger de répondre à une requête en particulier, ou un groupe entier de requêtes.
- Pour pouvoir traiter une requête HTTP de type GET, une servlet doit implémenter la méthode `doGET()`, ...
- Une servlet n'est pas chargée de l'affichage de données.

Servlet avec vue

⊙ Les pages JSP :

- Les pages JSP se présentent sous la forme d'un simple fichier au format texte, contenant des balises respectant une syntaxe à part entière.
- Le langage JSP combine à la fois les technologies HTML, XML, servlet et JavaBeans en une seule solution permettant aux développeurs de créer des vues dynamiques.
- La technologie JSP est une technologie offrant les capacités dynamiques des servlets tout en permettant une approche naturelle pour la création de contenus statiques.

⊙ Mise en place d'une JSP :

- Création de la vue :

Un fichier **.jsp** sous WebContent.

- Cycle de vie d'une JSP :

- Quand une JSP est demandée pour la première fois, ou quand l'application web démarre, le conteneur de servlets va **vérifier, traduire puis compiler la page JSP en une classe héritant de `HttpServlet`**, et l'utiliser durant l'existence de l'application.

→ La JSP est alors transformée en servlet par le serveur.

→ Les JSP permettent au développeur de faire du Java sans avoir à écrire du code Java.

⊙ Mise en relation avec notre servlet :

Pour respecter l'architecture MVC, nous devons créer une servlet de contrôle pour notre page JSP afin de pouvoir y accéder sans passer par son URL.

- ① Déplaçons notre page `test.jsp` dans le répertoire `/WEB-INF`. (une page présente sous ce répertoire n'est plus accessible directement par une URL côté client).

② Associons notre page JSP à une servlet:

```
public void doGet (HttpServletRequest request,  
    HttpServletResponse response) throws  
    ServletException, IOException  
{  
    this.getServletContext().  
    getRequestDispatcher("/WEB-INF/test.jsp").  
    forward(request, response);  
}
```

↳ rediriger la paire requête/réponse HTTP
vers une autre servlet ou vers une page JSP.

Transmission de Données

○ Données issues du serveur : Les attributs

Transmettre des variables de la servlet à la JSP :

Dans la servlet :

```
String message = "Transmission de variables : OK";  
request.setAttribute("test", message);
```

créer un attribut
dans une requête

le nom que l'on
souhaite donner
à l'attribut

l'objet à transmettre

Dans la page JSP :

```
<%  
    String attribut = (String) request.getAttribute("test");  
    out.println(attribut);  
%>
```

à l'intérieur de
cette balise on
peut écrire du
code Java.

On fait un cast
car la méthode
retourne un
Object.

recupérer un
attribut en
spécifiant
son nom

○ Données issues du client : Les paramètres

Les paramètres sont transmis au serveur directement via l'URL.

exemple : `www.monstersite.com ? ecole = ensias & auteur = insaf`

Récupération des paramètres par le serveur :

(dans la servlet ou dans la page JSP)

```
String parametre = request.getParameter("auteur");  
out.println(parametre);
```

La méthode retourne
un String

Le Java Bean

○ Définition et objectifs:

Un bean est un simple objet Java qui suit certaines contraintes, et représente généralement des données du monde réel.

- Les propriétés: un bean est conçu pour être **paramétrable**.
On appelle « propriétés » les champs non publics présents dans un bean (de type primitif ou objets), elles permettent de paramétrer le bean en y stockant des données.
- La sérialisation: un bean est conçu pour 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.
- La réutilisation: un bean est conçu pour être **réutilisable**.
Ne contenant que des données ou du code métier, un bean est indépendant de la couche de présentation et de la couche d'accès aux données, ce qui lui donne ce caractère réutilisable.
- L'introspection: un bean est conçu pour être **paramétrable de manière dynamique**.
L'introspection est un processus qui permet de connaître le contenu d'un composant de manière dynamique, sans disposer de son code source.

○ Structure:

- Un bean:

- doit être **une classe publique**.
- doit avoir (au moins) un **constructeur par défaut, public et sans paramètres**.
- peut implémenter l'interface « **Serializable** » (⇒ persistance!).
- ne doit **pas avoir de champs publics**.
- les propriétés doivent être accessibles via des méthodes publiques **getter et setter**.

exemple:

```
public class MonBean
{
    /* Java assigne à cet objet un constructeur par défaut, public
    et sans paramètres. */

    private String p1;
    private int p2;

    public String get P1 ()
    { return this.p1; }

    public int get P2 ()
    { return this.p2; }

    public void set P1 (String p1)
    { this.p1 = p1; }

    public void set P2 (int p2)
    { this.p2 = p2; }
}
```

○ Mise en place:

- Les beans doivent être placés dans le répertoire src de notre projet web.
- Par défaut, nos classes compilées sont dans un dossier nommé build. Afin de rendre nos objets accessibles à notre application, on doit les placer dans un dossier nommé classes sous le répertoire /WEB-INF.

Build Path → Configure Build Path →

Source → Default output folder → préciser le chemin
test / WebContent / WEB-INF / classes

- Un bean peut être transmis en tant qu'attribut de requête.

la technologie JSP

○ Les balises:

→ Balises de commentaire:

`<% -- Ceci est un commentaire JSP. -- %>`

→ Balises de déclaration:

`<%! String chaine = "Hey!"; %>`

(on peut faire plusieurs déclarations au sein d'un même bloc)

→ Balises de scriptlet:

`<% (du code Java) %>`

→ Balises d'expression:

`<% = "Bip bip!" %>` (sans le ;)

⇔

`<% out.println("Bip bip!"); %>`

○ Les directives:

○ importer un package.

○ inclure d'autres pages JSP.

○ inclure des bibliothèques de balises.

○ définir des propriétés et informations relatives à une page JSP.

→ Directive taglib:

`<%@ taglib uri = "maTaglib.tld" prefix = "tagExemple" %>`

→ Directive page: définir des informations relatives à la page JSP.

`<%@ page import = "java.net.*", java.util.*"
session = "false"`

`%>`

→ Directive include:

`<%@ include file = "unePage.jsp" %>`

(l'inclusion est réalisée au moment de la compilation)

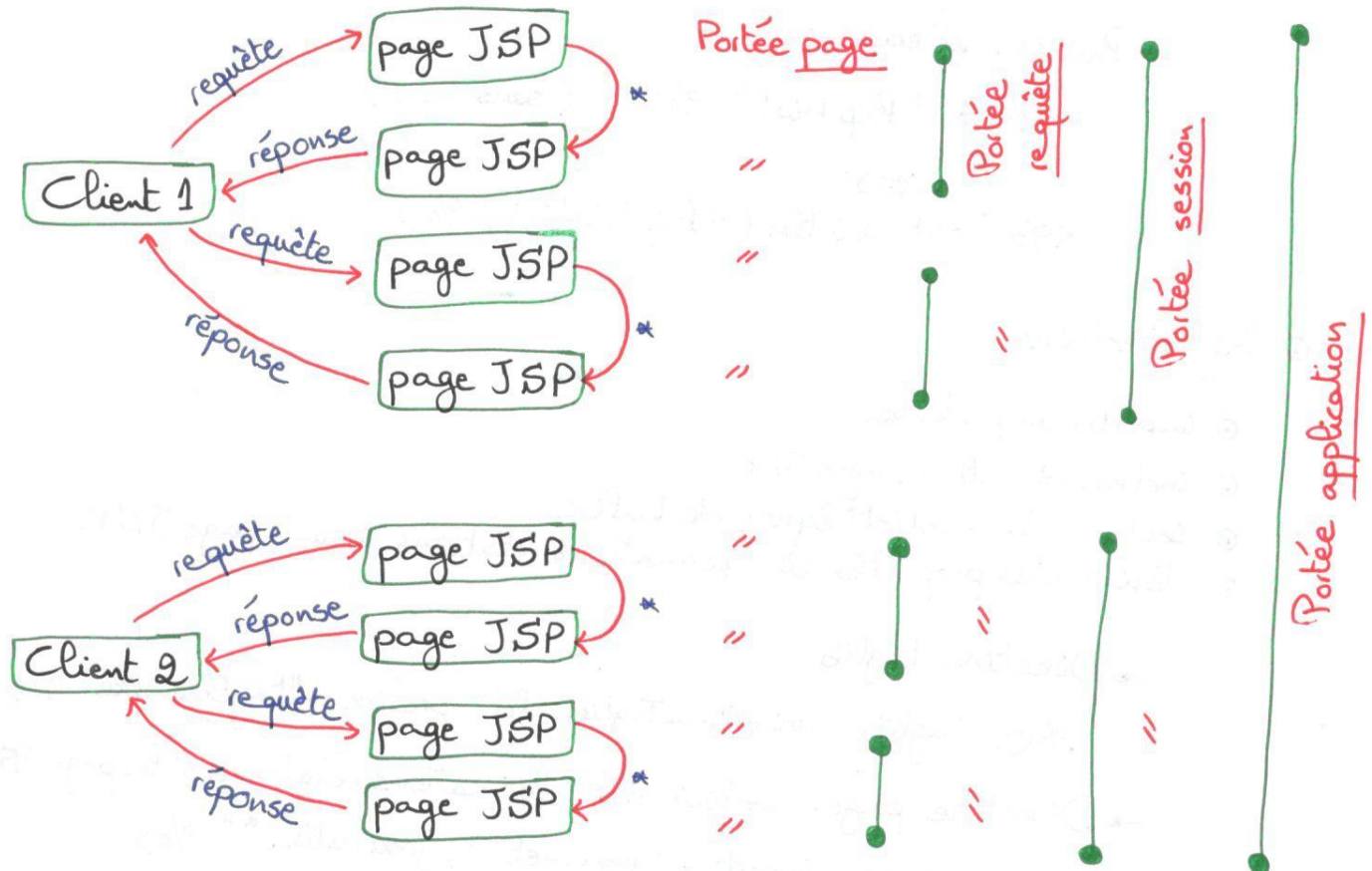
<jsp:include page = "unePage.jsp" />

(les deux pages sont compilées séparément, et l'inclusion est faite au moment de l'exécution.)

○ La portée des objets:

Il existe 4 portées différentes dans une application:

- **page** → dans la page JSP en question seulement.
- **requête** → durant l'existence de la requête en cours.
- **session** → durant l'existence de la session en cours.
- **application** → durant toute l'existence de l'application.



* inclusion ou redirection

Les sessions

- HTTP est un protocole sans état : le serveur, une fois qu'il a envoyé une réponse au client, ne conserve pas les données le concernant (pour lui, chaque nouvelle requête émane d'un nouveau client).

→ nécessité des sessions !

Session = un espace mémoire alloué pour chaque utilisateur, permettant de sauvegarder des informations tout le long de leurs visites.

↳ basée sur l'objet Java : `HttpSession`

/ Création ou récupération de la session */*

```
HttpSession session = request.getSession();
```

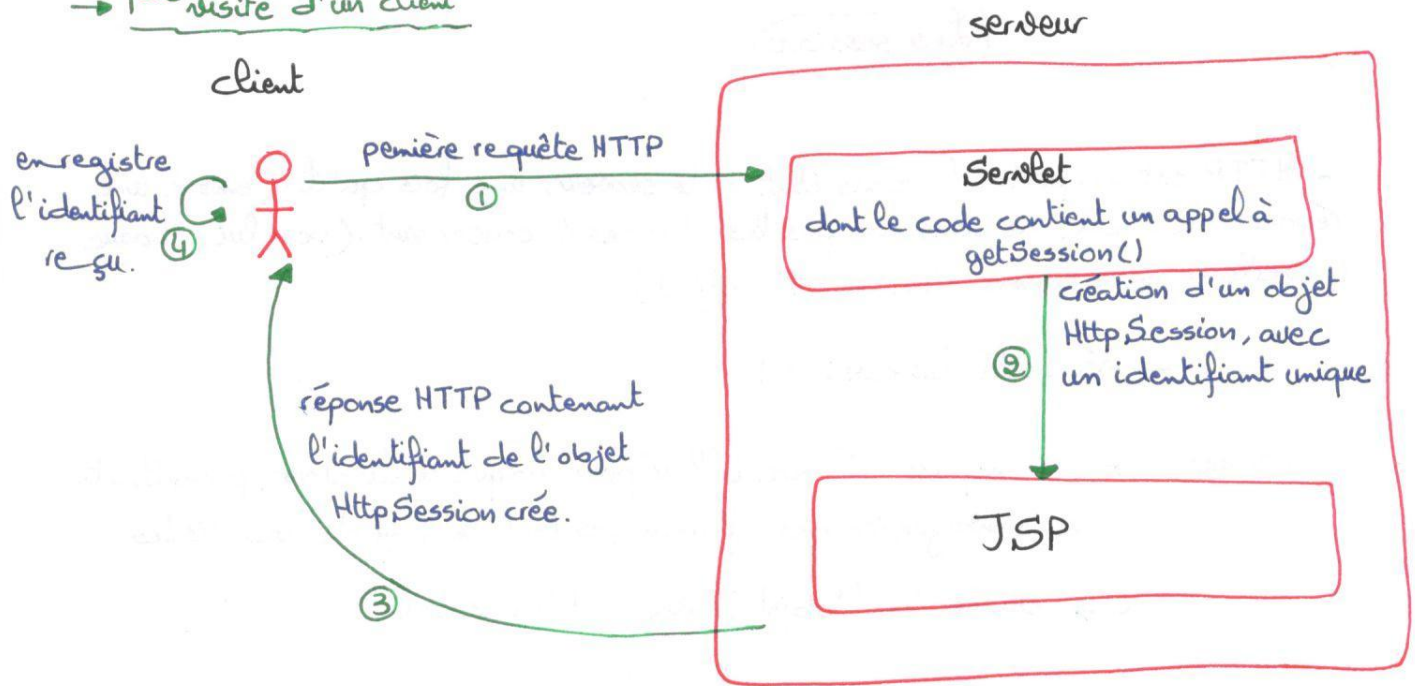
/ Mise en session d'un objet */*

```
session.setAttribute("login", "insaf");
```

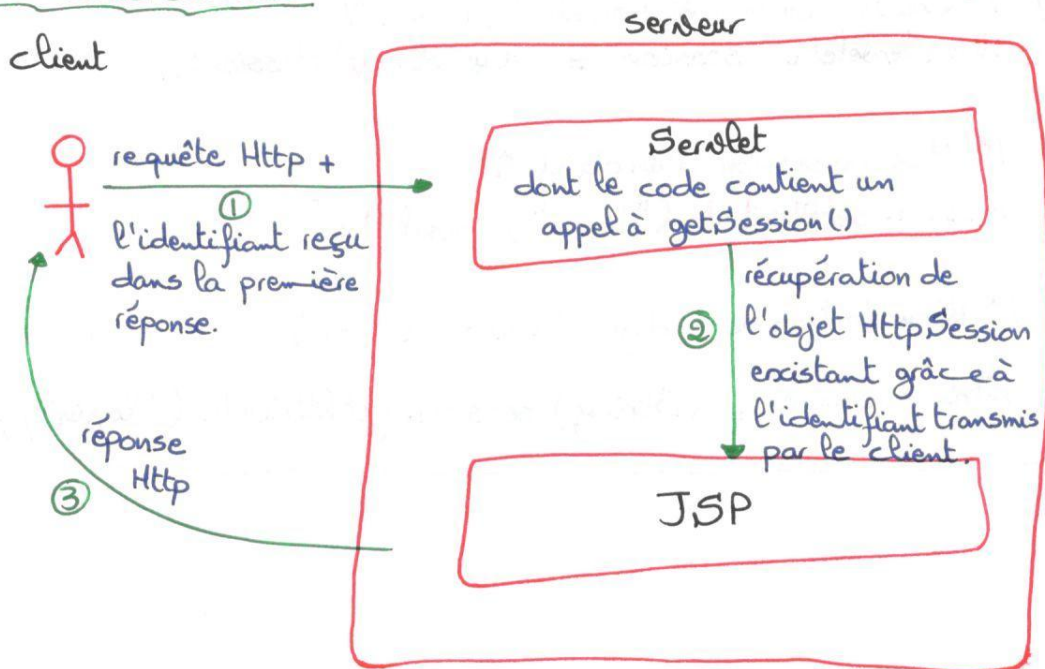
/ Récupération de l'objet depuis la session */*

```
String chaine = (String) session.getAttribute("login");
```


→ 1^{ère} visite d'un client



→ Prochaine visite du client



Les cookies

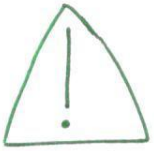
- Cookie** = un petit fichier placé directement dans le navigateur du client.
- Il lui est envoyé par le serveur à travers les en-têtes de la réponse HTTP, et ne contient que du texte.
 - Il est propre à un site ou à une partie d'un site en particulier, et sera renvoyée par le navigateur dans toutes les requêtes HTTP adressées à ce site ou à cette partie du site.

Côté HTTP:

- Un cookie a obligatoirement un nom et une valeur.
- Un cookie peut se voir attribuer certaines options, comme une date d'expiration.
- Le serveur demande la mise en place ou le remplacement d'un cookie par le paramètre **Set-Cookie** dans l'en-tête de la réponse HTTP qu'il envoie au client.
- Le client transmet au serveur un cookie par le paramètre **Cookie** dans l'en-tête de la réponse HTTP qu'il envoie au serveur.

Côté Java EE: on manipule les cookies à travers l'objet Java **Cookie**.

- Un cookie doit obligatoirement avoir un nom et une valeur.
- Il est possible d'attribuer des options à un cookie, telle qu'une date d'expiration ou un numéro de version.
- 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 la liste des cookies envoyés par le client.



Les cookies et les sessions sont deux concepts totalement distincts.

un espace mémoire alloué
dans le navigateur du
client dont lequel on ne
peut placer que du texte.

un espace mémoire alloué
sur le serveur dans lequel
on peut placer n'importe quel
type d'objets.

Communication avec les BDD

- ⊙ Chargement du driver MySQL: une seule fois durant le chargement de l'application

```
try
{
    Class.forName("com.mysql.jdbc.Driver");
}
catch (ClassNotFoundException e) { }
```

- ⊙ Connexion à la base de données:

- ⊙ Identification de l'URL:

jdbc : mysql : // nomhôte : port / nombdd

le nom du hôte
sur lequel le serveur
MySQL est installé.

(localhost si sur la même
machine que l'application
est exécutée).

le nom de la bdd
à laquelle on
souhaite nous
connecter.

le port TCP/IP écouté par
le serveur MySQL.
(3306 par défaut)

- ⊙ Etablissement de la connexion:

```
String url = "jdbc:mysql://localhost:3306/test";
```

```
Connection connexion = null;
```

```
try
{
    connexion = DriverManager.getConnection(url, "user", "mdp");
    /* requêtes */
}
catch (SQLException e) { }

finally
{
    if (connexion != null)
        try { connexion.close(); }
        catch (SQLException ignore) { }
}
```


⊙ Création d'une requête:

```
Statement statement = connexion.createStatement();
```

⊙ Exécution de la requête:

⊙ **Lecture:** SELECT

```
ResultSet resultat = statement.executeQuery  
    ("SELECT id,nom FROM users;");  
while (resultat.next())  
{  
    int id-u = resultat.getInt("id");  
    String nom-u = resultat.getString("nom");  
}
```

⊙ **Ecriture:** INSERT - UPDATE - DELETE - CREATE

```
int statut = statement.executeUpdate("...");
```

⊙ Libération des ressources:

```
finally  
{  
    if (resultat != null)  
        try { resultat.close(); }  
        catch (SQLException ignore) { }  
    if (statement != null)  
        try { statement.close(); }  
        catch (SQLException ignore) { }  
    if (connexion != null)  
        try { connexion.close(); }  
        catch (SQLException ignore) { }  
}
```

○ Les requêtes préparées:

- ↳ pré-compilées.
- ↳ paramétrées.
- ↳ protégées.

```
PreparedStatement pS = connexion.prepareStatement("SELECT  
* FROM users;");
```

```
pS.executeQuery();
```

```
pS = connexion.prepareStatement("INSERT INTO users (id, nom)  
VALUES (?, ?);");
```

```
int id-u = request.getParameter("idU");
```

```
String nom-u = request.getParameter("nomU");
```

```
pS.setInt(1, id-u);
```

```
pS.setString(2, nom-u);
```

```
int statut = pS.executeUpdate();
```