* **9. Dans le contexte d'un serveur WEB, qu'est-ce qu'un CGI et qu'est-ce qu'une servlet ? Comparer ces deux technologies par rapport au traitement des formulaires classiques. A quoi ressemble le code d'un CGI basique qui répondrait à une simple sollicitation ? A quoi ressemble le code d'une servlet classique (packages, cycle de vie, principales variables et fonctions membres des classes usuelles, paramètres de ces dernières) ?**

Point 1:

Les CGI:

Les CGI (Common Gateway Interface) sont des programmes qui tourne sur le serveur WEB et qui ont été lancé à partir d’une requête en provenance d’un interface de navigation client. A chaque requête, un nouveau processus est donc crée. Généralement écrit en C/C++ et compilée pour fournir un executable non portable. De nos jours, nous en trouvons de plus en plus écrit en Perl (langage de script assez avancées).

Les Servlets:

Les servlets sont des unités de code (typiquement des classes) Java qui étendent les possibilités des serveurs (essentiellement, des serveurs WEB). Ceux-ci ont donc la même fonctionnalité que les CGIs mais permette une meilleure solution. En effet ceux-ci ont quelques avantages en plus :

- La portabilité : Etant écrite en Java, elle sont exécutées sur une machine virtuelle se trouvant sur le serveur. Son execution est donc indépendante de l’environnement du serveur.

- Instanciation unique : L’objet servlet n’est instancé qu’une seule fois (au démarrage de celle-ci), et non a chaque demande d’un client (contrairement a une CGI). Elle peut donc conserver ses données (compteur, connexion JDBC, ect).

- Principe des Threads : Chaque demande d’un client génère un thread qui s’occupera de cette requête et tous ces thread travaillent sur le même objet (l’objet servlet). Nous nous retrouvons donc dans un degré d’intimité que les CGI, n’ont pas. L’objet servlet devient donc un objet monitor (utilisation de code synchronized pour le code sensible).

- Sécurité : Etant dans un mécanisme Java, ses sécurités peuvent être mise en place.

Attention : lorsque l’on fait une analogie entre servlet et applet, c’est dans le sens qu’elle étende le les possibilités de leurs note (une applet étends les possibilité d’un browser tandis qu’une servlet étends les possibilités d’un serveur). Une servlet != une applet. La servlet est server-side alors que l’applet est client-side.

Point 2:

Une fois l’utilisateur ayant validé les données introduites dans le formulaire, celles-ci sont véhiculées, dans un cas simple, dans l’URL (query-string).

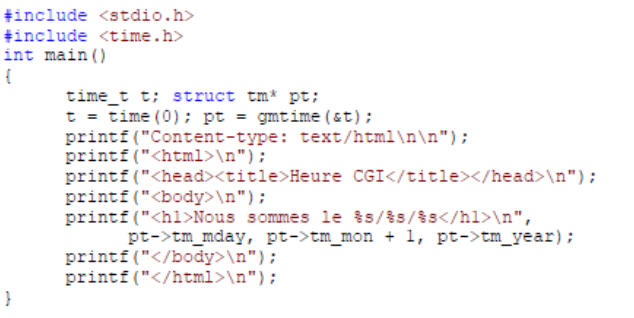
Dans le cas d’une servlet, celles-ci seront récupérées simplement grâce à la méthode :

**public abstract String getParameter(String NomParametre)**

de l’objet **HttpServletRequest** request qui a pour but de nous donner accès au header de la requête client mais surtout aux paramètres. Cette méthode nous renvoi donc le paramètre indiqué en paramètre et ‘null’ si celui-ci n’existe pas. La servlet réalise alors un traitement X et retourne une réponse au client sous forme de page HTML par exemple. Cette page sera donc générée dynamiquement par la servlet.

Dans le cas d’un CGI, le serveur, une fois une requête reçue, analysera tout d’abord l’en-tête http pour essayer de lancer le programme appropriée (script, …). Ce programme récupèrera les valeurs du formulaire sous forme de **variable d’environnement**. Il réalisera un traitement et écrira dans un premier temps le type de réponse dans l’entête http tout comme la servlet ainsi que la page HTML (tag) proprement dite.

Ici, la comparaison, outre celles évoquées ci-dessus, est au niveau du passage des informations. Etant donné que le CGI est un programme « externe » le seul moyen qui a été adopté pour lui faire prendre connaissance de ces informations est l’utilisation des variables d’environnements.

Point 3 :

Voici le code simple d’une CGI répondant à une requête GET :

Point 4:

Les classes/interfaces nécessaires se trouve dans les JDK récent de J2SE/J2EE, et essentiellement dans le package **javax.servlet** et ses sous-package :

- Une interface Servlet déclarant les méthodes de gestion des servlets (et communication avec le client).

- Deux interfaces ServletRequest et ServletResponse correspondant respectivement au traitement d’une requête et à l’envoie d’une réponse.

- Une classe abstraite GenericServlet : implémente l’interface Servlet et indépendante du protocole de communication (Nous dériverons donc de cette classe pour les les servlets spécifique tels que HttpServlet).

- Deux classes abstraite ServletInputStream et ServletOutputStream, fournissant les flux de communication avec le client (en in et out respectivement).

Au niveau de HTTP, un sous-package est mis en place : (Mais la question parle d’une servlet classique donc je ne pense pas que ce soit a prendre… Plus a la page 5 de Java III)

Nous trouverons dans **javax.servlet.http** les classes plus spécifique à un traitement avec le protocole de communication HTTP.

- La classe abstraite HttpServlet : dérivé de GenericServlet, dévoue à l’utilisation des méthodes PUT et GET de HTTP.

- Deux interface HttpServletRequest et HttpServletResponse, dérivé de ServletRequest et ServletResponse

…

Le cycle de vie d’une servlet est composé de 3 étapes :

- **L’initialisation** : Le byte code de la servlet doit être chargé en mémoire à l’initiative du serveur, ce dernier exécute alors la méthode :

public abstract void init(ServletConfig config) throws ServletException

ServletException dérive de Exception et est lancé lors d’un problème a l’initialisation de la servlet. L’objet implémentant ServletConfig contient les information de configuration de la servlet mais également la référence vers son container (objet ServletContext). Classiquement, on appel la méthode de la super-classe qui se charge de l’initialisation. C’est également dans cette méthode que les connexions JDBC vers des bases de données seront mis en place. En effet, ce genre de connexion ne se fait qu’une seule fois, et cette méthode ne sera appelé qu’une et une seule fois. Le serveur ne pourra initialisé une servlet que si cette dernière à été au préalable déchargé au moyen de la méthode destroy(). Les servlets ont également leur propre chargeur de classe (Objet ClassLoader), ce qui permet au serveur Web de recharger (en suivant le cycle complet) une servlet dont le fichier class à été modifier (raison pour la quelle, les servlets doivent se trouver dans un dossier particulier du serveur).

- **Le traitement des requêtes client** : pour se faire, la servlet utilise la méthode :

public abstract void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

qui a pour but de matérialiser le mécanisme de requête/réponse en acceptant deux objets représentant la requête entrante du client et la réponse sortante qui sera renvoyé au client. Dans le cadre d’une servlet dédié à un protocole de communication, ce sera cette méthode qui dirigera vers les méthodes approprié (Dans le cas HTTP, vers des méthodes dédicacé au requête POST, ou GET, ect).

- **La terminaison d’une servlet** : une servlet sera déchargé par le serveur par la méthode :

public abstract void destroy()

Cette fonction réalisera le travail inverse de la méthode init(), c’est a dire terminer les connexions JDBC et libérer différente ressource. (A noter qu’il est impératif de s’assurer que la servlet n’est plus en cours d’utilisation, au moyen d’un compteur de requête en cours par exemple).