ENST Bretagne
Département Informatique

Servlets et JSP
TP Installation d'un environnement Serveur HTTP et Servlet
Modification d'une servlet et de son applet

1 Objectifs :

Dans ce premier TP, nous allons installer les applications nécessaires à la mise en œuvre des servlets et de JSP. Pour que ce soit plus facile à gérer, chaque participant doit utiliser les mêmes configurations avec les mêmes noms

2 Installation de l'environnement de travail

Dans cette première partie nous allons installer un serveur http spécialisé pour les servlets et les pages JSP, installer un environnement java, installer des éditeurs spécialisés en java, html, jsp,etc.

L'adresse d'un répertoire "**donneeTPServlet**" vous sera donné en début du TP.

2.1 Installation de l'environnement Java

1) sous le disque C: créer un répertoire "**jsp**"
2) Installation de l'environnement Java, dans le répertoire **donneeTPServlet** , vous avez un répertoire Java, les outils de développement pour Java.
   - Exécutez **"jdk-6u1-windows-i586-p.exe",** mettre le répertoire d'installation sous C:
     Vous devez voir un répertoire par exemple : **C:\ jdk1.6.0_01**
   - Définissez la variable d'environnement
     JAVA_HOME = C:\ jdk1.6.0_01
   - Ajoutez un chemin à la variable d'environnement **PATH**
     C:\ jdk1.6.0_01\bin

2.2 Installation du serveur Tomcat

3) Du répertoire Tomcat du répertoire **donneeTPServlet**, exécutez " **apache-tomcat-6.0.14.exe**" en le mettant sous **C:\jsp**
   - Dans **C:\jsp\ Tomcat 6.0\bin** vous avez la commande de lancement de tomcat « **tomcat6.exe** », exécutez la. Pour arrêter tomcat, fermez la fenêtre « **dos »** associée.
   - A ce niveau vous pouvez ouvrir les pages d'accueil du serveur **tomcat** (http://localhost:8080) et regarder l'exécution de servlets et des pages JSP.

2.3 Installation des éditeurs

Nous allons installer un certain nombre d'éditeurs, des éditeurs simples au départ pour bien comprendre les différentes mises en œuvre, puis des éditeurs de gestion de projets comme **eclipse** qui vous permettra de gérer vos servlets et vos pages jsp sans vous soucier des contextes de mise en œuvre.

4) Sous le répertoire **editeurs** exécutez **pspad453inst_en.exe.** C'est un éditeur syntaxique simple "**PSPAD**" qui nous permettra de lire et modifier les fichiers de type XML, JSP, HTML, Java, etc.

5) Sous le répertoire **editeurs** , exécuter **jgrasp186_05.exe** et préciser la destination sous le répertoire **C:\jsp.** C'est un autre éditeur syntaxique simple qui est plus spécialisé dans les langages particulièrement java, l'exécution des programmes et leur compilation.

Il se lance à partir du menu ***démarrer***.

L'éditeur Eclipse ne sera installé qu'au prochain TP, en effet cet éditeur facilitera le travail, mais ne vous permet pas de bien appréhender les différentes phases de mise en œuvre.

3 Modification, compilation et lancement d'une servlet simple.

Sous votre navigateur demander la page http://localhost:8080 , aller sous la page Servlet Examples et dans cette page lancer la servlet ***Hello World.***

Avec **Jgrasp** ouvrir le fichier java de cette servlet, il se trouve dans le répertoire

C:\jsp\Tomcat 5.5\webapps\servlets-examples\WEB-INF\classes, c'est le fichier ***HelloWorldExample.java,*** changer le nom de la classe par Bonjour et sauvegarder cette classe sous Bonjour.java dans le même répertoire.

Compilez cette classe :

- Votre éditeur Jgrasp peut ne pas connaître la commande javac, dans ce cas mettre l'adresse du répertoire (**C:\ jdk1.6.0_01\bin)** dans le chemin PATH de la commande choisie pour gérer le langage Java avec Jgrasp. (ceci sera précisé pendant le TP).
- Par la suite, le compilateur ne connaîtra pas les servlets, elles ne font pas partie de la distribution jsdk. Rajouter au CLASSPATH de votre éditeur JGRASP le ***jar*** suivant:
  - **C:\jsp\ Tomcat 6.0 \lib\ servlet-api.jar**

Avec votre navigateur demandez son exécution :

http://localhost:8080/servlets-examples/servlet/Bonjour

Que se passe t-il, il y a de grande chance que cette servlet ne soit pas connue, vous ne l'avez pas définie (son alias, c'est-à-dire son nom d'appel par les clients) dans le fichier **web.xml** de votre application WEB.

Nous allons définir cette servlet au serveur, en lui définissant un alias ou sans alias en précisant tout simplement le nom de la classe.

3.1 Par définition d'un alias:

Ouvrez le fichier **web.xml de** l'application WEB ***servlets-examples***, il se trouve sous le répertoire ***WEB-INF*** , vous pouvez utilisez l'éditeur **PSPAD**. Vous n'y trouvez pas la définition de la servlet Bonjour.

Définissez sa classe avec la balise <servlet>,

```
<servlet>
  <servlet-name>direBonjour</servlet-name>
  <servlet-class>Bonjour</servlet-class>
</servlet>
```

et le nom par lequel vous voulez l'appeler à partir des navigateur par la balise <servlet-mapping>, par exemple

```
</servlet-mapping>
  <servlet-mapping>
  <servlet-name>direBonjour</servlet-name>
  <url-pattern>/maservlet/salut</url-pattern>
</servlet-mapping>
```

Appeler la par son nom dans le navigateur : ***http://localhost:8080/servlets-examples/maservlet/Salut***

Si elle ne fonctionne pas, attendez, le serveur recharge votre fichier **web.xml,** autrement arrêtez et recharger le serveur tomcat.

3.2 Par son nom de classe :

On va le faire pour toutes les applications WEB, pour cela il suffit de dire que lorsque le serveur s'aperçoit qu'il s'agit d'une servlet, il passe sont nom de classe à une servlet connue qui est **invoker,** elle demande le lancement de votre servlet. Pour permettre au serveur de savoir qu'il s'agit d'une servlet, nous allons utilisé un "mapping" spécial, par exemple, mettre le mot clé servlet devant le nom de la servlet.

Cette méthode n'est pas conseillée sur un serveur en service. Cependant pour la mise au point et pour les tests elle permet de lancer rapidement n'importe quelle servlet, sans modifier le fichier web.xml de l'application web.

Changer encore le nom de la servlet en l'appelant par exemple "**Bonsoir**" et compilez la, vous devez voir un fichier Bonsoir.class

Ouvrez le fichier **web.xml général**, il se trouve sous le répertoire ***conf*** de la racine, vous pouvez utiliser l'éditeur jext.

Et retirez les commentaires sur ces lignes :

```
<servlet>
     <servlet-name>invoker</servlet-name>
     <servlet-class>
       org.apache.catalina.servlets.InvokerServlet
     </servlet-class>
     <init-param>
         <param-name>debug</param-name>
         <param-value>0</param-value>
     </init-param>
     <load-on-startup>2</load-on-startup>
 </servlet>


 <servlet-mapping>
     <servlet-name>invoker</servlet-name>
     <url-pattern>/servlet/*</url-pattern>
 </servlet-mapping>
```

Avec ce mapping, on précise que le texte (*) après le mot clé **servlet** représente le nom d'une servlet.

Avec votre navigateur demander son exécution :

http://localhost:8081/servlets-examples/servlet/Bonsoir

SI elle n'est pas connue, arrêtez votre serveur et relancez le. Ce fichier web.xml, n'est lu qu'au lancement du serveur.
Attention avec la version 6, vous aurez une erreur au lancement du serveur, cette méthode n'étant pas conseillée, vous devez l'autoriser dans le fichier "context.xml' sous le répertoire **conf**., en remplaçant <Context > par <Context privileged="true" >

3.3 Modification de la servlet

Modifiez la génération HTML de la dernière servlet (Bonsoir), en y rajoutant par exemple ;

```
out.println("<h1>" +  "de l'ENST bretagne"  + "</h1>");
```

Compilez la servlet et appelez la par le navigateur, a-elle changé?

Suivant l'état du serveur, il est possible qu'il ignore son changement vous avez deux solutions pour y remédier:

- Rechargez votre application WEB, ici il s'agit de ***servlets-examples.*** Il faut utiliser le manager, en ayant les droits (mot de passe)
- Spécifiez au serveur qu'il doit automatiquement recharger une servlet modifiée.

Dans le premier cas, il faut utiliser le manager

Dans le deuxième cas il faut utiliser créer un fichier de **"contexte"** pour l'application, nous le verrons avec éclipse

3.3.1 Ajout d'utilisateurs et de droit

Ouvrir le fichier **conf\tomcat-users.xml**

Et regardez la ligne suivante

```
<user username="admin" password="admin"roles="admin,manager"/>
```

Elle définit un utilisateur (admin) avec les droits de manager et d'administrateur.

3.3.1.1 Utilisation du manager

Ouvrez le manager : **http://localhost:8080/manager/html**

Vous y voyez les différentes applications WEB qui sont chargées. Faites la commande **Recharger** pour ***servlets-examples*** et appelez la servlet **Bonsoir** par le navigateur, elle a du changer

3.3.2 Ajout d'une application WEB

1) Ajout dans un répertoire quelconque

Sur le répertoire **donneeTPServlet** vous avez une application WEB c'est le répertoire **appliSimple** sous le répertoire "appliWeb" Copiez le sur votre disque ou vous voulez, mais pas dans le répertoire C:\jsp\jakarta-tomcat-5.0.12\webapps.

Avec le manager, ajoutez la en tant qu'application WEB sur le serveur, attention ce sont des « / » et pas des « \ ». Par exemple, vous avez mis **appliSimple** dans le répertoire **c:/jsp**

```
Context Path: /uneApplication
WAR or Directory URL: file:C:/jsp/appliSimple
```

Testez la servlet et la page « **jsp**» de cette application.

Arrêter votre serveur et relancer le, cette application est-elle toujours présente ?

Regardez dans **C:\jsp\jakarta-tomcat-5.0.12\webapps** vous voyez l'application **uneApplication,** avec le manager faites **Undeploy** sur cette application, que remarquez vous ?

2) Ajout dans un répertoire prédéfini

Mettre l'application WEB **appliSimple** du répertoire **donneeTPServlet** dans le répertoire **C:\jsp\jakarta-tomcat-5.0.12\webapps** .

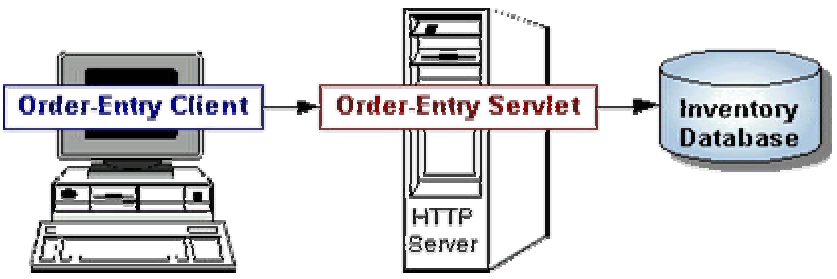
Actualisez votre manager, que remarquez vous?

3) Ajout du contexte de l'application

C'est la méthode choisi par **éclipse** que nous verrons ultérieurement.

4 Le client-serveur.

L'architecture client-serveur que nous allons aborder dans ce TP se situe uniquement dans l'accès par des clients à des pages HTML généré par un serveur. Nous n'abordons pas le coté serveur d'applications qui permet l'accès par différents clients à différentes ressources logicielles



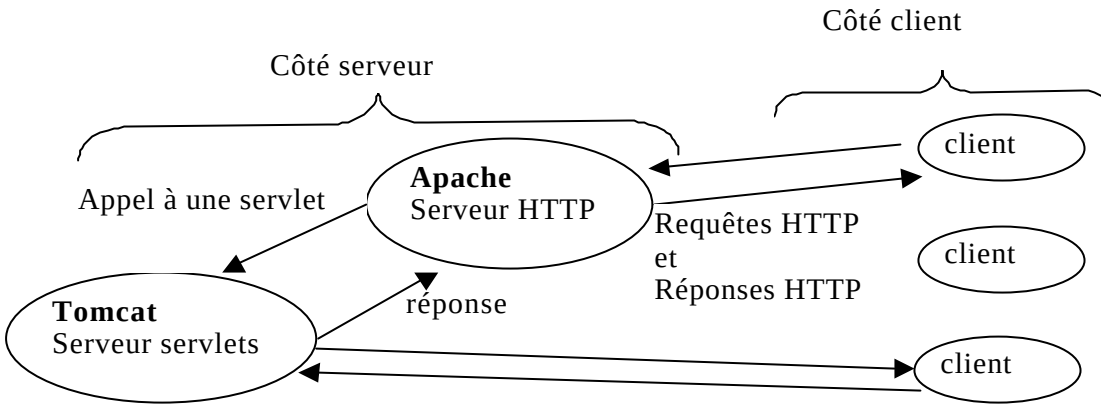
La servlet (un programme java qui s'exécute sur le serveur et qui peut être appelé par le client) dispose du coté serveur de toutes les ressources logicielles pour constituer sa réponse (page HTML par exemple), notamment de l'accès à des bases de données. Dans une servlet le texte HTML est inclus dans le code Java. Dans ce TP, nous chercherons nos informations à l'intérieur de fichiers.

4.1 Coté serveur.

Le rôle d'un programme serveur de pages est de générer des pages en fonction des choix et du contexte de l'utilisateur. Le serveur reçoit par une requête http une demande avec des paramètres de choix, il construit sa page en fonction de cette demande.

Un serveur http attend les requêtes, ici il s'agit d'un serveur ***Apache***, s'il reconnaît l'appel à une servlet ou à une page JSP, il redirige cette requête vers un serveur de servlets et de pages JSP, ici c'est le serveur ***Tomcat***. Ce dernier, lorsqu'il a construit sa réponse la redirige vers le serveur http qui la renvoie au client.

Dans notre TP nous accédons directement au serveur Tomcat en indiquant son port d'accès.



Au départ, le client ne possède aucune page HTML, il a juste un navigateur. C'est le serveur qui lui envoie ces pages, elles sont dites statiques si le serveur se contente de lui envoyer une copie des pages qu'il possède, dynamique si elles sont générées entièrement ou partiellement en fonction de la demande.

Nous trouvons donc chez le serveur, les pages HTML statiques et les programmes permettant de générer d'autres pages (JSP, ASP, CGI, PHP, etc…).

Votre application WEB fait partie du serveur, c'est là qu'il vient chercher vos propres pages ou vos programmes.

4.2 Coté client.

Le client doit formuler ses choix lors de la requête, en général, il le fait de manière interactive par les FORM en HTML ou par l'interface offerte par une applet. Nous allons essayer de réaliser ces deux façons d'accéder à des pages dynamiques. Ces requêtes sont incluses dans les pages HTML venant du serveur.

5 Fournisseur d'accès WEB

Votre serveur Tomcat a été mis en place, il vous permet de préparer les servlets, les pages JSP, les pages HTML, les applets, les beans.

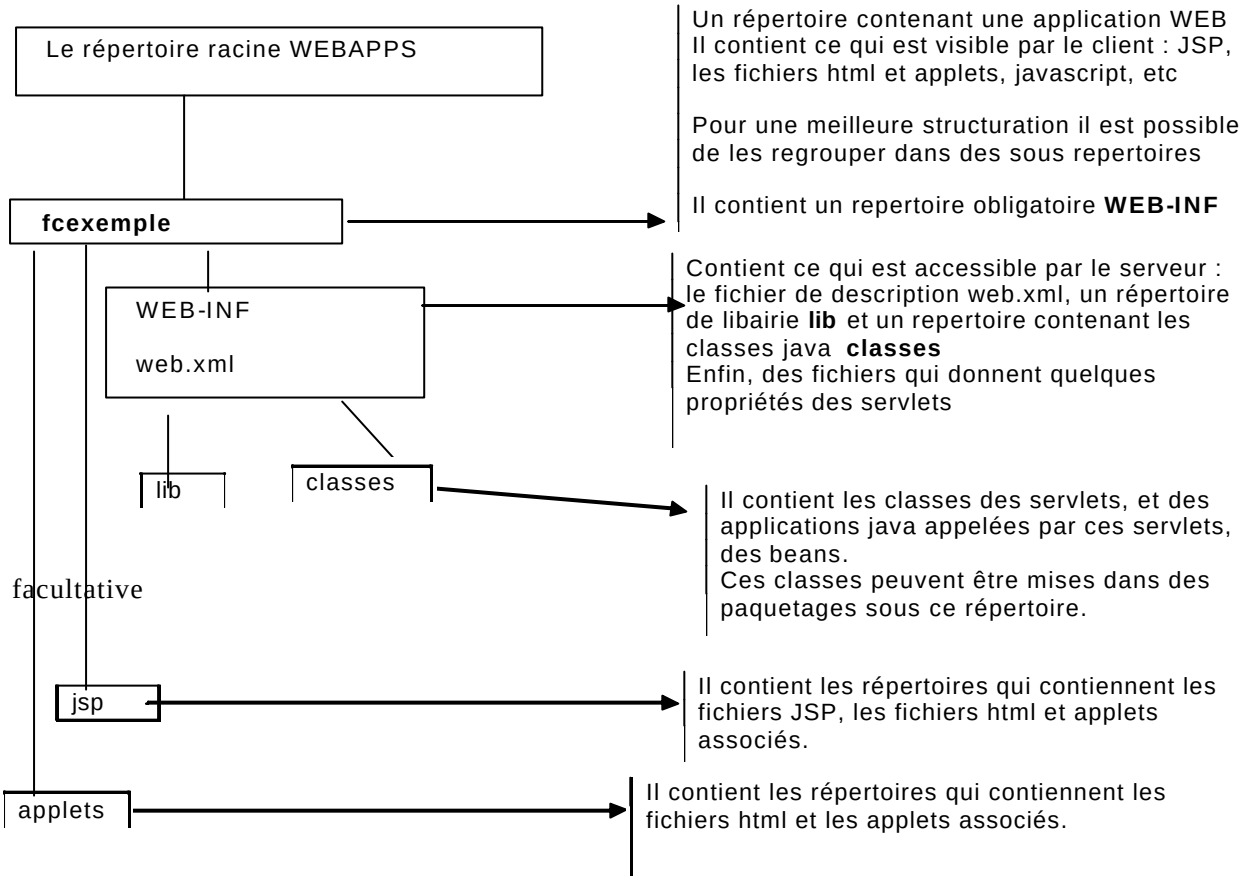
L'adresse de ce serveur est

http://localhost:8080 localhost peut être changé par le nom de la machine ou son adresse IP

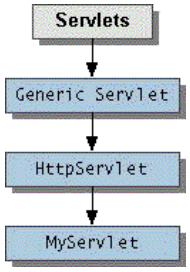
La structuration du répertoire d'une application WEB n'est pas libre, il faut mettre les classes java, les pages HTML, les pages JSP dans des sous répertoires précis.

6 Réalisation coté serveur.

6.1.1 Une organisation de répertoires d'applications web



6.2 Le paquetage javax.servlet.



La classe HttpServlet permet de définir les servlets travaillant avec le protocole http. Cependant, la classe abstraite GenericServlet peut être spécialisée avec d'autres protocoles. Vous trouverez la définition des méthodes de cette classe à l'adresse suivante :

http://java.sun.com/products/servlet/2.1/api/javax.servlet.http.HttpServlet.html

Le début de notre programme s'écrira de la manière suivante :

```
import java.io.* ;
import java.util.* ;
import javax.servlet.* ;
import javax.servelet.http.* ;

public class ServletDrapeau2a extends HttpServlet {
```

Cette classe javax.servlet.http.HttpServlet est un point de départ pour comprendre le fonctionnement d'une servlet. En lisant la documentation de cette classe, on trouve des méthodes qui sont doGet, doPost, doPut etc. qui permettent les échanges entre une servlet et ses clients. Nous allons travailler avec l'appel Get.

6.3 Interaction d'une servlet avec un client.

Quand une servlet accepte un appel d'un client, elle reçoit 2 objets qui sont des paramètres de méthodes citées ci-dessus :

- ServletRequest englobe la communication du client vers le serveur. Nous disposons de l'interface javax.servlet.ServletRequest. Cette interface va nous permettre d'accéder aux informations suivantes :
  - σ les noms des paramètres transmis par le client,
  - σ le protocole utilisé par le client,
  - σ le nom de la machine qui fait la demande,
  - σ le nom du serveur qui reçoit cette demande,
  - σ le flot d'entrée de données dont les opérations s'implémentent avec l'interface HttpInputStream. Le protocole d'application utilisé par le flot d'entrée est cohérent avec les méthodes HTTP put et HTTP post. Leurs noms précis sont doPost(HttpServletRequest, HttpServletResponse) et doGet(HttpServletRequest, HttpServletResponse).
- ServletResponse englobe la communication du serveur vers le client. Nous disposons de l'interface javax.servlet.ServletReponse. Cette interface nous procure :
  - σ les méthodes pour répondre au client,
  - σ la longueur du message expédié au client,
  - σ le type MIME des données,
  - σ le flot de sortie dont les opérations s'implémentent avec l'interface HttpOutputStream.
  - σ Un Writer à travers lequel la servlet expédie ses informations au client.

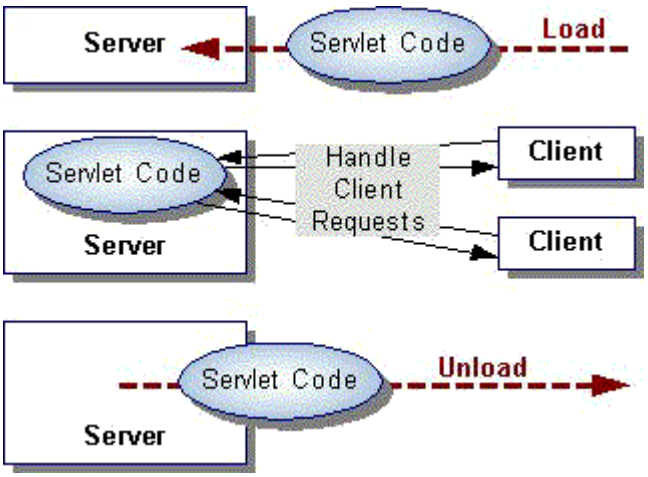
Vous serez à même d'aller chercher des informations plus précises sur ces interfaces et classes lorsque vous écrirez votre servlet , API des servlets :

C:/jsp/Tomcat 205.5/webapps/tomcat-docs/index.html.

Dans notre TP, nous utiliserons uniquement une réponse sous la forme de texte HTML, Vous trouverez dans les exemples des tutoriaux sur le WEB la réalisation de servlets retournant soit des images, du son, de la vidéo, etc.

6.4 Description des méthodes utilisées de la classe Servlet.

Vous trouverez à l'adresse suivante des renseignements sur la vie d'une servlet. Voici son cycle de vie.



La servlet est chargée soit au lancement du serveur, soit à son premier appel, elle reste alors à l'attente de requêtes jusqu'à l'arrêt du serveur ou une demande de son arrêt.

6.4.1 La méthode init() de votre servlet.

Cette méthode est appelée au premier lancement de la servlet.

Le fichier **web.xml**, permet de définir certaines données d'initialisation utilisée dans la méthode *init()*. Nous n'utiliserons pas ce type de fichier dans ce TP.

Comme dans toute application Java, (ce n'est pas une Applet), la servlet a accès aux différents fichiers ou bases de données de votre machine.

6.4.2 La méthode doGet() de votre servlet.

Elle va vous permettre de dialoguer en utilisant les flux d'entrée et de sortie.

Voici par exemple une servlet simple retournant une page HTML

```
public class SimpleServlet extends HttpServlet
    {
       /**
        * Handle the HTTP GET method by building a simple web page.
        */
       public void doGet (HttpServletRequest request,
                          HttpServletResponse response)
       throws ServletException, IOException
       {
          PrintWriter       out;
          String            title = "Simple Servlet Output";

          // set content type and other response header fields first
          response.setContentType("text/html");

          // then write the data of the response
          out = response.getWriter();
          out.println("<HTML><HEAD><TITLE>");
          out.println(title);
          out.println("</TITLE></HEAD><BODY>");
          out.println("<H1>" + title + "</H1>");
          out.println("<P>This is output from SimpleServlet.");
          out.println("</BODY></HTML>");
          out.close();
       }  }
```

Vous voyez que l'objet ***response*** instance de **HttpServletResponse** offre différents services pour préparer la réponse notamment un objet ***PrintWriter*** qui reçoit les chaînes de caractères qui constituent la réponse.

C'est ce type de page que nous allons définir.

7 Travail demandé sur les servlets

Dans l'application WEB fcexemple dans le répertoire **donneeTPServlet** vous trouvez les fichiers suivants

7.1 Coté serveur : la servlet à modifier

Fichier : fcexemple \WEB-INF\classes\ ServletDrapeau2a.java

```
import java.io.*;
import java.util.*;
import javax.servlet.ServletException;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletOutputStream;

public class ServletDrapeau2a extends HttpServlet {

  private String dirData =  "E:/Tomcat5.5/Tomcat 5.5/webapps/fcexemple/data" ;
          // un repertoire quelconque chez le serveur
 private String urlDrapeau = "http://localhost:8080/fcexemple/ data/";

          // une adresse url connue par le serveur
  File repertoireTexte;
  Hashtable drapeaux = new Hashtable();
         // table de hashcode  pour trouver les images indexes par le nom du pays
  Hashtable textes = new Hashtable();
         //table de hashcode  pour trouver les textes indexes par le nom du pays
  private static final int CLE      = 0;
```

Mettre l'adresse du répertoire ou se trouvent les données

Mettre l'adresse web de ce répertoire

```
private static final int NOMFICHIER    = 1;
private static final int PAYS = 2;
private String drapeauParDefaut[] = {"bzh",  "bzh.gif",  "Bretagne"};

private String fichiersDrapeaux[][] = {
          // contenu qui va etre mis dans la Table de Hascode pour les images
  {"uk",  "gb.gif",  "Angleterre"},    // la premiere valeur est la cle   :  uk
  {"fr",  "fr.jpg",  "France"},          // la deuxieme valeur est le nom du fichier image : gb.gif
  {"bzh",  "bzh.gif",  "Bretagne"}  // la troisieme valeur est le nom du pays
};

private String  fichiersTexte[][] = {
         // contenu qui va etre  mis dans la Table de Hascode pour les textes
  {"bzh",  "bretagne.txt",  "Bretagne"},     // la premiere valeur est la cle   :  fr
  {"fr",  "france.txt",  "France"},              // la deuxieme valeur est le nom du fichier
  {"uk",  "angleterre.txt",  "Angleterre"}      // la troisième valeur est nom du pays
};

public void init  (ServletConfig config) throws ServletException {
  super.init(config);
           // recherche du répertoire ou se trouve les fichiers contenant les drapeaux
  repertoireTexte = new File(dirData);
  if (!(repertoireTexte.exists() && repertoireTexte.isDirectory())) {
    log("le mauvais repertoire");
    throw new ServletException("On ne trouve pas votre repertoire" + dirData);
  }
//   Création des table de Hashcode
  for (int i = 0; i < fichiersDrapeaux.length; i++) {   // remplissage  de la table de Hascode
    drapeaux.put(fichiersDrapeaux[i][CLE], fichiersDrapeaux[i]);
  }
  for (int i = 0; i < fichiersTexte.length; i++) {   // remplissage  de la table de Hascode
    textes.put(fichiers Texte[i][CLE], fichiersTexte[i]);
  }
}


public void doGet(HttpServletRequest req, HttpServletResponse resp)
throws IOException, ServletException {
  String country = "";
  try {
        // Recherche du bon drapeau d'après la requete de type country
    if (req.getParameter("country") != null) {
      country = req.getParameter("country");
    }
    else {
      country =fichiersDrapeaux[1][CLE];
    }

    String [] leDrapeau = (String [])(drapeaux.get(country));
    String [] leTexte = (String [])(textes.get(country));
    if (leDrapeau == null) {
      leDrapeau = drapeauParDefaut;
    }
    resp.setContentType("text/html");
    ServletOutputStream out = resp.getOutputStream();
    out.println("<HTML><HEAD><TITLE>");
    out.println("Page générée par une servlet");
    out.println("</TITLE></HEAD><BODY>");

    out.println(" <H1>  drapeau du pays  :  " + leDrapeau[PAYS] + "</H1>");
    out.print(" <IMG SRC=\"" + urlDrapeau + leDrapeau[NOMFICHIER] + "\" > ");
```

```
        out.println(" <H1> Hymne national du pays " + leDrapeau[PAYS] + " </H1>");
        BufferedReader in = new BufferedReader(
                              new FileReader(repertoireTexte + "/" +leTexte[NOMFICHIER]) );
        String laChaineLue;
        while((laChaineLue = in.readLine())!= null) {
          out.print(" <p>" );
          out.print(laChaineLue);
          out.println(" </p>" );
        }
        out.println("</BODY></HTML>");
        in.close();
      }
        catch (IOException e) {
          throw new ServletException("Probleme dans l'écriture de la reponse" );
        }
        finally {
          resp.getOutputStream().close();
        }

    }
  }
```

La réponse contient une page HTML avec un lien sur une image d'un drapeau et le texte de l'hymne national

7.2 Coté clients : Les appels à la servlets

Voici le fichier HTML qui fait ces appels :

Fichier : fcexemple \servlets\index.html

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
  <title>Servlet Drapeaux</title>
</head>
<body bgcolor="#FFFFFF">

<center>
<h1>
  <b><font face="Arial, Helvetica, sans-serif">utilisation des Servlets</font></b>
</h1></center>
<h2 color="red">
<b><font face="Arial, Helvetica, sans-serif"><font color="#CC0000"><font size=+2>
    Appel par une Applet</font></font></font></b></h2>
<b><font face="Arial, Helvetica, sans-serif"><font size=+2>Demander un
drapeau particulier</font></font></b>

<br><applet  code=AppletDrapeau.class width=800 height=50>
</applet>

<h2 color="red">
<b><font face="Arial, Helvetica, sans-serif"><font color="#FF0000"><font size=+2>
Appel par une Form HTML</font> </h2>


<form METHOD=GET
    ACTION="servlet/ServletDrapeau2a?country=select_input1"
    TARGET=_blank>
 <choisissez votre pays
</font></font>
<br>
 <select NAME="country">
   <option VALUE=fr> France 
```

Vérifier si c'est bien le nom de votre servlet

```
    <option VALUE=uk selected>Angleterre 
    <option VALUE=bzh>Bretagne 
 </select><input type="submit" value="Envoyer" >
</form>

</b>
</body>
</html>
```

Voici l'applet référencée dans cette page HTML

Fichier fcexemple \servlets\appletDrapeau.java

```
  import java.awt.*;
  import java.io.*;
  import java.applet.*;
  import java.net.*;
  import java.awt.event.*;
  public class AppletDrapeau extends Applet {
    Panel panelHaut;
 CheckboxGroup choixPays ;
    Button demande = new Button("demander");
    String adressehttp = "http://localhost:8080/fcexemple/servlet/ServletDrapeau2a";
    String [] [] lesPays = {
      { "France" , "fr"},
      { "Angleterre" , "uk"},
      { "Bretagne" , "bzh"}
    };
    public void init() {
      setLayout(new BorderLayout());
      panelHaut = new Panel();
      panelHaut.setLayout(new FlowLayout());
      choixPays = new CheckboxGroup();
      for (int i = 0; i < lesPays.length; i++)
        panelHaut.add(new Checkbox(lesPays[i][0], choixPays, true));
      add(panelHaut,"North");
      add(demande,"South");
      demande.addActionListener(new Choixdemande());
      System.out.println("depart");

    }

    class Choixdemande implements ActionListener {

      public void actionPerformed(ActionEvent  evenement) {
        URL lUrl=null;
        try {
          for (int i = 0; i < lesPays.length; i++) {
            if (choixPays.getSelectedCheckbox().getLabel().equals(lesPays[i][0]))
            {
              lUrl = new URL(adressehttp + "?country=" + lesPays[i][1] );
              getAppletContext().showDocument(lUrl,"_blank" );
            }
          }
          repaint();
        }
l
          catch (Exception erreur ) {
            System.out.println("mauvaise adresse URL : " + erreur.getClass().getName());
          }
      }
    }
  }
```

Mettre l'adresse WEB de votre servlet

7.3 Votre travail

Comprendre la page HTML, l'applet et la servlet.

Modifier l'application précédente pour que l'on puisse avoir le drapeau et l'hymne des usa, le fichier image c'est ***usa.gif*** et le texte c'est ***usa.txt***.

7.4 Adresses WEB de tutoriaux

http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html

http://java.sun.com/j2ee/tutorial/ 1_3-fcs/doc/JSPIntro.html

http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/JSPBeans.html

http://courses .coreservlets.com/Course-Materials/csajsp2.html

http://courses.coreservlets.com/Course-Materials/msajsp.html