Exercice sur les EJB Session avec état

Soit un site de commerce électronique appartenant à un fournisseur de matériel informatique. Une page principale permet à l'utilisateur de s'inscrire, donc de saisir son nom et un mot de passe qu'il aura choisi.
Après inscription, une nouvelle page s'affiche contenant les caractéristiques de machines disponibles. L'utilisateur devra choisir les caractéristiques de la machine qu'il souhaite commander.

1. Ecrire une servlet qui permet d'afficher les pages décrites ici sachant que les informations que devra saisir l'utilisateur sont les suivantes :
   - Taille de la mémoire centrale
   - Fréquence du processeur
   - Avec Graveur de CD et/ou de DVD
   - Type du système d'exploitation.

2. Sachant que la servlet devra transmettre les informations saisies à un composant EJB, écrire la classe correspondant à un EJB Session qui consiste à récupérer les informations saisies. Quel type de EJB Session pensez-vous utiliser ? Pourquoi ?

3. Donnez les différentes étapes nécessaires pour que cette application soit opérationnelle sur un serveur d'application ?

Solution

1. Une solution serait de créer un fichier index.html pour l'inscription de l'utilisateur, qui appelle une servlet qui génère un formulaire pour la saisie de la configuration de la machine à commander.

Le fichier index.html :

```
<html><head><title>Commande de Matériel</title></head><body>
<center> <h2> Vous êtes sur un site de commerce électronique <br>
Prenez le soin de vous inscrire </h2></center><br>
<form method="POST"
action="http://localhost:8080/samia/ConfigurationServlet">
<center>Tapez votre nom : <input name=nom type=text> <br>
<br>
Tapez votre mot de passe : <input name=passwd type=password><br><br>
<br>
<input type=submit value=Validez>
<input type=reset value=Annuler>

</center>
</form>
</body></html>
```

Le fichier *ConfigurationServlet.java*. Cette servlet est composée de deux méthodes *doPost* et *doGet*. La première sert à récupérer le login et et le mot de passe et génère le formulaire alors que la deuxième méthode sert à récupérer les caractéristiques saisies et appelle un composant EJB pour lui transmettre ces informations. La méthode *getConfigRef* sert à associer une session de la servlet (càd une connexion client) à un composant EJB. Il y aura autant de clients connectés que de sessions, par conséquents de composants créés, car à chaque session sont associées des informations saisies différentes.

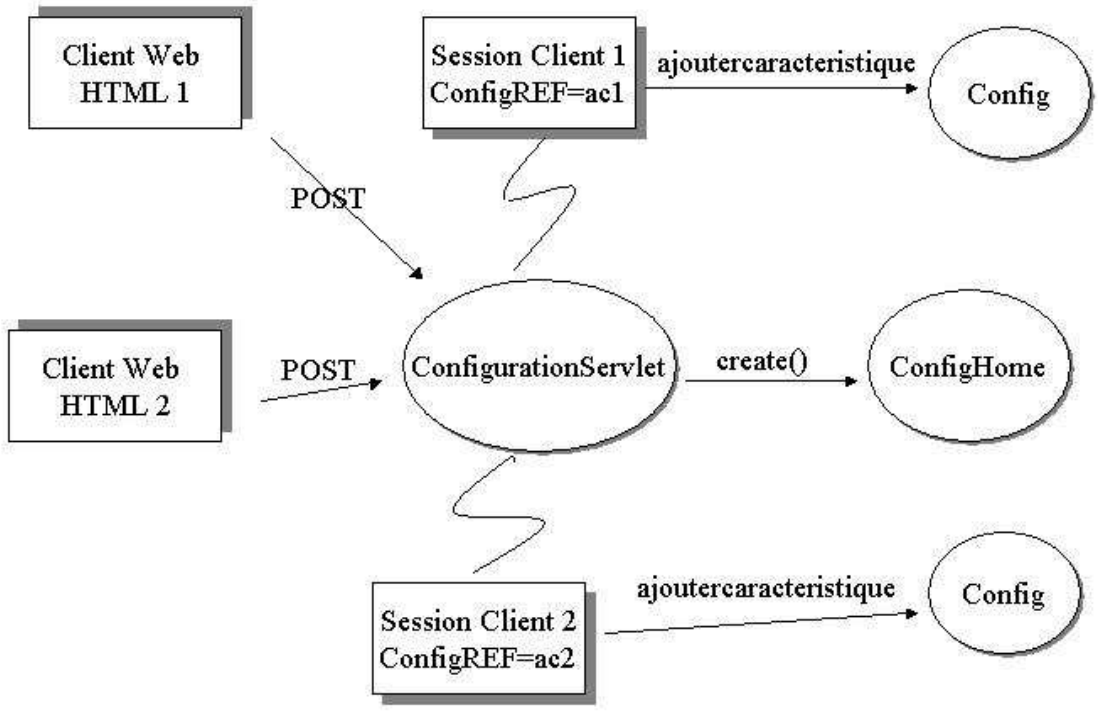
```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;


public class ConfigurationServlet extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html";
      private ConfigHome configHome;


    // à l'initialisation de la servlet, on récupère la référence de
    // l'interface Home de Config
```

```
    public void init() throws ServletException {
        try {
            Context ctx = new InitialContext();

            Object ref = ctx.lookup("Config");
            configHome = (ConfigHome)
                    PortableRemoteObject.narrow(ref,
ConfigHome.class);
            System.out.println("La servlet s'est connecté au serveur
JNDI local a récupéré la référence de l'interface Home distant de
Config");
        } catch (NamingException e) {
            System.out.println("Erreur " + e);
            e.printStackTrace();
        }
    }

    // cette méthode récupère de l'objet session la référence
    // d'un EJB Config; s'il n'en existe pas, l'EJB Config est créé
    // et stocké dans l'objet session pour un usage futur

    private Config getConfigRef(HttpServletRequest req) {
        Object ac = null;

        try {
            HttpSession session = req.getSession(true);
            ac = session.getAttribute("ConfigREF");
            if (ac == null) {
                ac = configHome.create();
                session.setAttribute("ConfigREF", ac);
            }
        } catch (Exception e) {
            System.out.println("Erreur " + e);
            e.printStackTrace();


        }

        return (Config) ac;
    }

   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
            throws ServletException, IOException {
                  System.out.println ("ENTRE DANS GET") ;
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        Config cf = getConfigRef(request);

cf.ajouterCaracteristique("MC",request.getParameter("listMemoire"));

cf.ajouterCaracteristique("PROC",request.getParameter("listFrequence"
));

cf.ajouterCaracteristique("CD_WRITER",request.getParameter("graveur1"
));

cf.ajouterCaracteristique("DVD_WRITER",request.getParameter("graveur2
"));

cf.ajouterCaracteristique("SE",request.getParameter("systeme"));
```

```
out.println("<html><head><title>ConfigServlet</title></head><body>");
        out.println("<center> <h2>Voici la Configuration que vous
avez choisie pour votre Machine </h2></center><br>");
            Hashtable list = cf.listerCaracteristique();
            if (list != null){
                                    out.println("<p> => " +
list.get("MC")+ "</p> <BR>");
                                    out.println("<p> => " +
list.get("PROC")+ "</p><BR>");
                                    out.println("<p> => " +
list.get("CD_WRITER")+ "</p><BR>");
                                    out.println("<p> => " +
list.get("DVD_WRITER")+ "</p><BR>");
                                    out.println("<p> => " +
list.get("SE")+ "</p><BR>");

            }else
            out.println("<p> => AUCUN ELEMENT SELECTIONNE !!</p>");
        out.println("</body></html>");
        out.flush() ;
        out.close() ;

    }


    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException {
                        System.out.println ("ENTRE DANS POST") ;
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();

        Config cf = getConfigRef(request);

        cf.setNomClient(request.getParameter("nom"));
        cf.setPassword(request.getParameter("passwd"));


out.println("<html><head><title>ConfigServlet</title></head><body>");
        out.println("<center> <h2>Veuillez choisir la Configuration
de votre Machine </h2></center><br>");
        afficherFormulaire(out);
        out.println("</body></html>");
            out.flush() ;
            out.close() ;
    }

    // une méthode auxiliaire qui permet de générer le formulaire
    private void afficherFormulaire(PrintWriter out) {
        out.println("<form method=\"GET\"
action=\"http://localhost:8080/samia/ConfigurationServlet\">");

      out.println("Donnez la taille de la Mémoire : <br>");
      out.println("<select name=listMemoire size=2>");
      out.println("<option selected> 512 Mo </option>");
      out.println("<option> 1 Go </option>");
      out.println("</select> <br> <br>");
      out.println("Donnez la fréquence du Processeur : <br>");
      out.println("<select name=listFrequence size=2>");
```

```
      out.println("<option selected> 200 MHz </option>");
      out.println("<option> 400 MHz </option>");
      out.println("</select> <br><br>");
      out.println("Avec un Graveur de :");
      out.println("<input type=checkbox name=graveur1 value=cd
checked> CD/ROM");
      out.println("<input type=checkbox name=graveur2 value=dvd>
DVD");
      out.println("<br><br> Avec un Systeme d'Exploitation :");
      out.println("<input type=radio name=systeme value=Win>
Windows");
      out.println("<input type=radio name=systeme value=Linux> Linux
<br> <br>");
      out.println("<input type=submit value=Validez>");
      out.println("<input type=reset value=Annuler>");
    out.println("</form>");
  // MC, PROC, CD_WRITER,DVD_WRITER , SE

    }
}
```

Pour faire le lien entre les différentes invocations en provenance d'un même client HTML, on utilise classiquement la notion de session donnée par l'interface `HttpSession` de l'API des servlets. L'objet `HttpSession` se comporte comme une Hashtable. Il permet le stockage de clés avec des valeurs associées. Par conséquent un objet `HttpSession` est créé pour chaque client dès la première interaction avec la servlet. En même temps, une instance de `Config` est créée et est stockée dans l'objet `HttpSession` (voir méthode `getConfigRef`). L'interface `ConfigHome` par contre sera la même pour tous les clients (voir méthode `init`).



2. Un Bean Session avec état sera nécessaire car les informations saisies vont caractériser chaque client connecté. Un EJB Session aura besoin de deux interfaces : une interface métier appelée *Config* et une interface de gestion de cycle de vie appelée *ConfigHome*.

Le fichier Config.java

```
import java.rmi.RemoteException;
import java.util.Hashtable;


public interface Config extends javax.ejb.EJBObject {

      public void ajouterCaracteristique(String key, String value)
throws RemoteException;
      public Hashtable listerCaracteristique() throws
RemoteException;
      public void setNomClient (String nomClient) throws
RemoteException;
      public String getNomClient() throws RemoteException;
      public void setPassword (String passwd) throws RemoteException;
}
```

Le fichier ConfigHome.java

```
import javax.ejb.*;
import java.rmi.*;


public interface ConfigHome extends javax.ejb.EJBHome {

      public Config create() throws CreateException, RemoteException;
      public Config createAvecNom(String nomClient, String password)
throws CreateException, RemoteException;

}
```

Le fichier ConfigBean.java

```
import java.util.Hashtable;
import javax.ejb.*;


public class ConfigBean implements SessionBean {
      SessionContext sc;
      protected static Hashtable caracteristiques = null ;
      String nomClient;
      String passwd;

      public void ejbCreate() throws CreateException {
            caracteristiques = new Hashtable () ;
      }

      public void ejbCreateAvecNom(String nom, String pass) throws
CreateException {
            nomClient=nom;
            passwd = pass;
            Hashtable  caracteristiques  = new Hashtable () ;
      }

      public void ejbRemove() {};
      public void ejbActivate() {};
```

```
	public void ejbPassivate() {};
	public void setSessionContext (SessionContext sc) {
		this.sc =sc;
	}

	public void ajouterCaracteristique(String p, String o) {

		if (o != null)
		caracteristiques.put(p,o) ;

	}

	public Hashtable listerCaracteristique() {
		return caracteristiques;
	}

	public void setNomClient (String nomClient) {
		this.nomClient = nomClient;
	}

	public String getNomClient() {
		return nomClient;
	}

	public void setPassword (String passwd) {
		this.passwd=passwd;
	}

}
```

3. Phases nécessaires au fonctionnement de l'application :

Exemple de l'environnement Windows et du serveur d'application j2ee de java.sun:

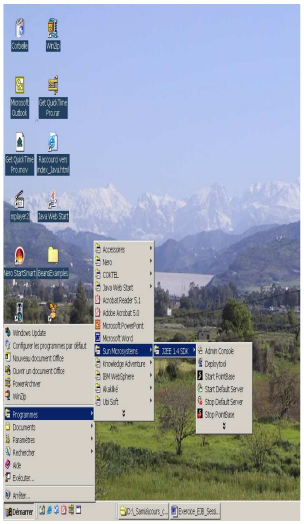
a. Installer un serveur d'application, par exemple j2eesdk-1_4_03-windows.exe (le serveur J2EE de SUN : http://www.java.sun.com) dans le répertoire C:\Sun\AppServer. La variable d'environnement PATH sera automatiquement mise à jour avec C:\Sun\AppServer\bin.
   Si vous avez choisi de télécharger la version complète de J2EE SDK (de 120,75 Mo), vous n'aurez pas besoin d'installer au préalable la machine virtuelle Java qui est directement installée en même temps que l'installation du serveur. Dans ce cas, Java est installé dans le répertoire suivant : C:\Sun\AppServer\jdk. Vous pouvez vérifier que dans le répertoire C:\Sun\AppServer\jdk\bin vous avez les fichiers exécutables (javac, java, javadoc, etc.). La variable PATH doit être alors mise à jour avec C:\Sun\AppServer\jdk\bin.
   Durant l'installation, une boîte de dialogue vous demandera de saisir un mot de passe pour l'utilisateur admin qui sera l'administrateur du serveur d'application.

   Le serveur d'application j2ee 1.4 SDK de Sun requiert un espace de 246,04 Mo sous Windows. Il est basé sur Java 1.5 et Message Queue 3.7.

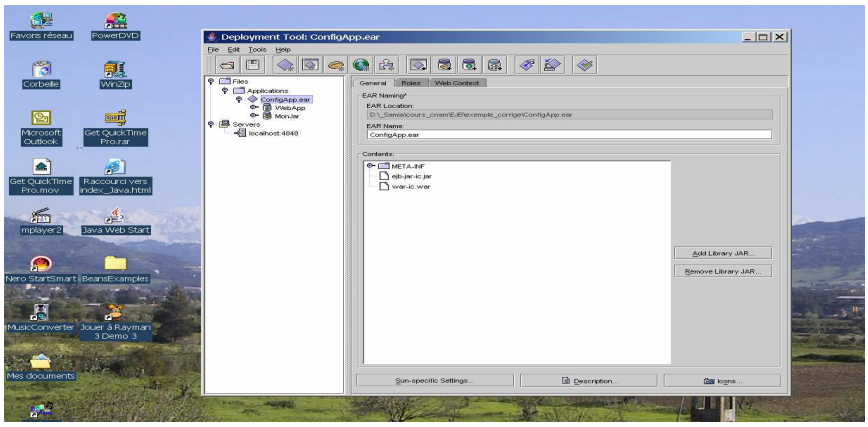
Pour que les programmes Java soient compilables, le CLASSPATH doit contenir un chemin vers le fichier j2ee.jar car celui-ci contient les packages javax.ejb.* (inexistants dans la machine virtuelle Java), dans notre cas ce sera C:\Sun\AppServer\lib\j2ee.jar. Inclure aussi le répertoire courant « . » dans le CLASSPATH.

b. Compiler sur une fenêtre MSDOS les différents programmes Java.
>javac *.java
>

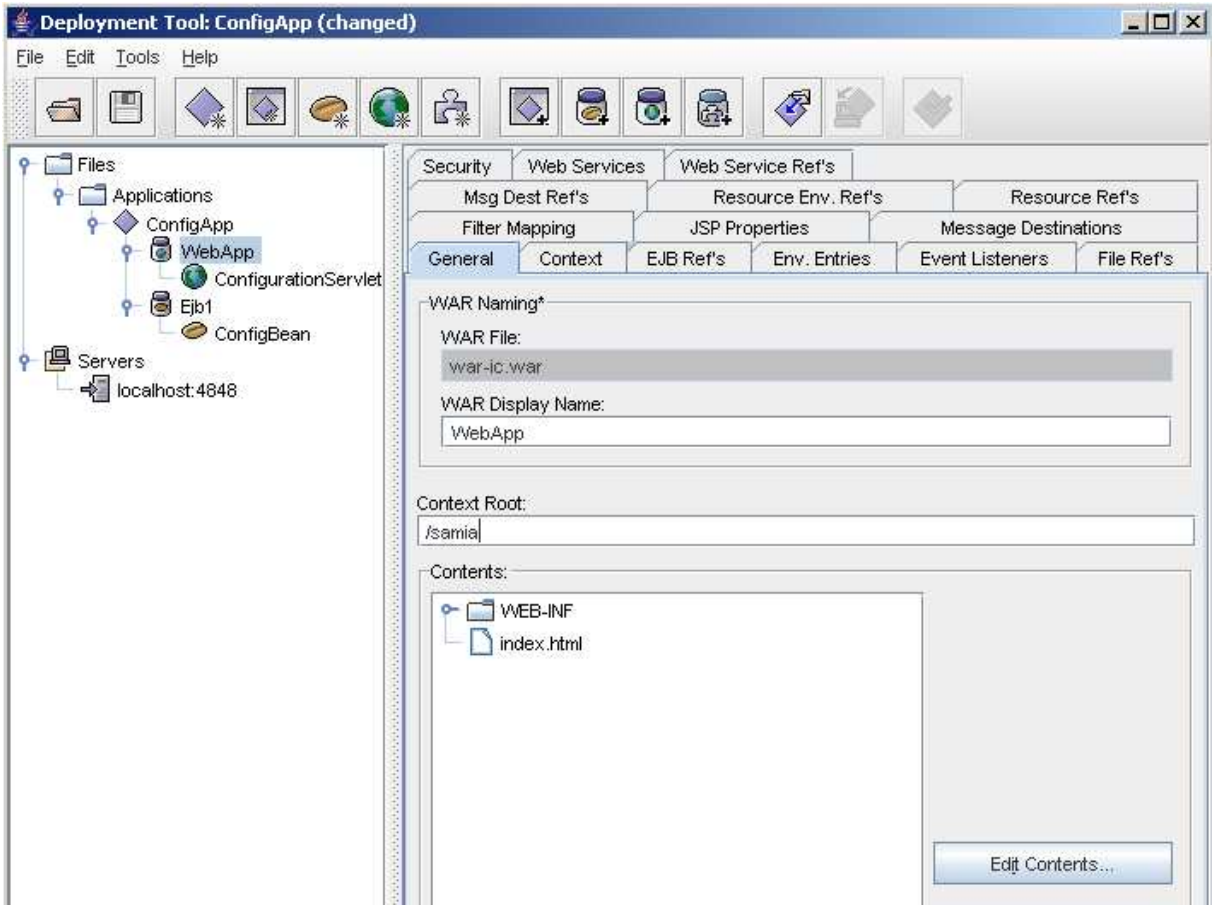
c. Démarrer le serveur :
Programs => Sun MicroSystems => J2EE 1.4 SDK => Start Default Server
Une fenêtre apparaît. Lorsque le message « Press any key to continue … » s'affiche, pressez sur une touche et la fenêtre disparaît (mais le serveur reste actif).



d. Pour vérifier si le serveur est actif, tapez http://localhost:8080. Une page Web contenant le titre « Sun Java System Application Server Platform Edition 8.0 » doit s'afficher.
e. Il existe une console d'administration qui permet de vérifier et de modifier les paramètres du serveur d'application. Cette console est activée en choisissant l'option :
Programmes => Sun MicroSystems => J2EE 1.4 SDK => Admin Console
f. L'étape qui suit est celle qui permet de construire l'application J2EE. L'application est construite grâce à un outil d'assemblage et de déploiement qu'on peut lancer en choisissant l'option :
Programmes=> Sun MicroSystems =>J2EE 1.4 SDK => Deploytool.
Cet outil est une interface graphique qui nous guide pour créer :
   - **une application ConfigApp.ear**. Un descripteur XML décrivant l'application est généré automatiquement.
   - **un composant Web** intégré à l'application qui va contenir la **servlet** et le fichier **index.html**. L'outil nous permet d'associer au composant des fichiers, dans notre cas il s'agit du fichier index.html et du fichier *ConfigurationServlet*.class. Il y a génération automatique d'un fichier WAR qui contient les fichiers sélectionnés ainsi qu'un descripteur XML web.xml. Lorsque le Web component est créé, on le complète en ajoutant un alias à cette servlet. Pour cela, sélectionner la servlet, ouvrer l'onglet *aliases* associé à la servlet et ajouter l'alias */ConfigurationServlet* pour la servlet *ConfigurationServlet*. Terminer la saisie en tapant la touche clavier return.
   - **un composant EJB** intégré à l 'application. L'outil nous permet d'associer au composant des fichiers classes, dans notre cas les fichiers Config.class, ConfigHome.class et ConfigBean.class. Une génération automatique d'un fichier JAR contient les fichiers .class et le descripteur XML sun-ejb-jar.xml.



g. Selectionner l'application web (WebApp). Dans l'onglet General, completer le champ *Context root* en lui associant un nom qui sera utilisé pour appeler votre page d'accueil. Dans notre exemple, nous avons choisi */samia*. Vous devez obtenir :



h. Associer le nom JNDI Config à l'EJB session. Pour cela sélectionner le bean. Cliquez le bouton « Sun-specific Settings ». Il apparaît un fenêtre. Dans cette fenêtre indique le nom Config comme nom JNDI. Cliquez OK.

i. Une fois l'application sauvegardée, il faut la déployer. Choisir dans le menu de l'outil d'assemblage et de déploiement l'option Tools=>deploy. Pour déployer, une boite de

dialogue vous invite à saisir le mot de passe de admin. Le serveur étant activé sur le port 4848. Une fenêtre s'affiche et comporte le résultat du déploiement.

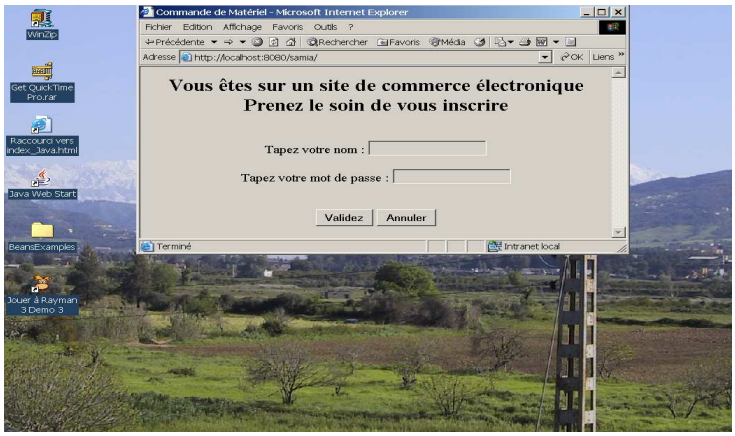
j. Si le déploiement s'est déroulé sans erreur, alors recopier le fichier ConfigApp.ear dans le répertoire C:\Sun\AppServer\domains\domain1\autodeploy\. Dès qu'il est recopié, un second fichier (de taille nulle) est généré dans ce répertoire portant le nom ConfigApp_deployed. Ce répertoire contient toutes les applications déployées et installées.

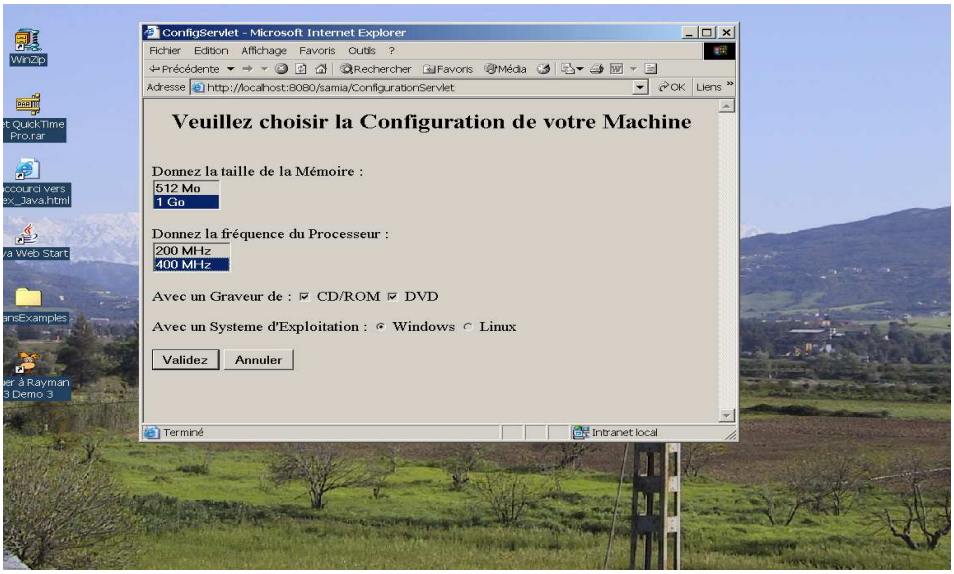
k. Il reste alors à vérifier que l'application marche bien en tapant : lançant le client

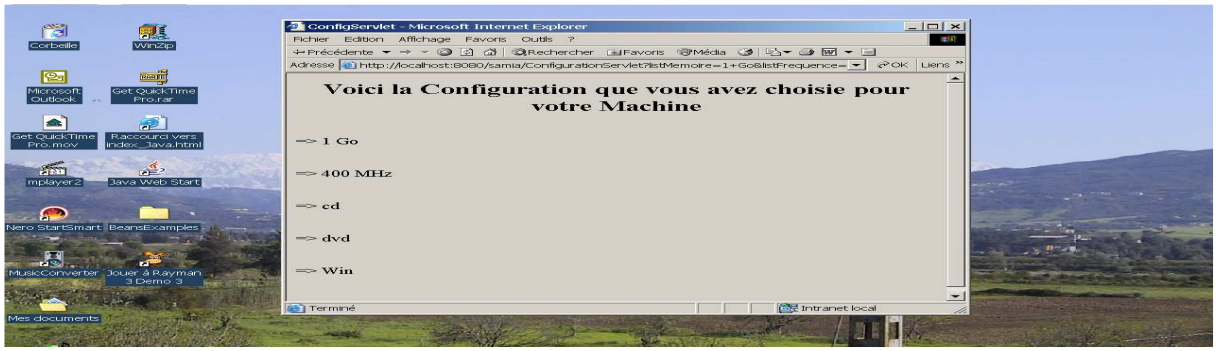
l. Stopper le serveur :

Programs => Sun MicroSystems => J2EE 1.4 SDK => Stop Default Server

Voici quelques fenêtres obtenues lors de l'exécution de l'application présentée ici.







Pour afficher un descripteur XML, on clique sur l'option Tools=>View Descriptor => View Descriptor dans l'outil Deployment Tool.

Le descripteur XML de l'application (application.xml), généré par l'outil d'assemblage :

```
<?xml version='1.0' encoding='UTF-8'?>
<application
      xmlns=" http://java.sun.com/xml/ns/j2ee"
      version=" 1.4"
      xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=" http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/application_1_4.xsd"
      >
   <description
         xml:lang=" fr"
         > Application description</description>
   <display-name
         xml:lang=" fr"
         > ConfigApp.ear</display-name>
   <module>
      <ejb> ejb-jar-ic.jar</ejb>
   </module>
   <module>
      <web>
         <web-uri> war-ic.war</web-uri>
         <context-root> /samia</context-root>
      </web>
   </module>
</application>
```

Le descripteur XML (web.xml) associé au composant Web, généré par l'outil d'assemblage :

```
<?xml version='1.0' encoding='UTF-8'?>
<web-app
      xmlns=" http://java.sun.com/xml/ns/j2ee"
      version=" 2.4"
      xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=" http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd "
      >
   <display-name
         xml:lang=" fr"
         > WebApp</display-name>
   <servlet>
      <display-name
            xml:lang=" fr"
            > ConfigurationServlet</display-name>
      <servlet-name> ConfigurationServlet</servlet-name>
      <servlet-class> ConfigurationServlet</servlet-class>
```

```
   </servlet>
   <servlet-mapping>
      <servlet-name> ConfigurationServlet</servlet-name>
      <url-pattern> /ConfigurationServlet</url-pattern>
   </servlet-mapping>
</web-app>
```

Le descripteur XML (ejb-jar.xml) associé au composant EJB, généré par l'outil d'assemblage :

```
<?xml version='1.0' encoding='UTF-8'?>
<ejb-jar
      xmlns=" http://java.sun.com/xml/ns/j2ee"
      version=" 2.1"
      xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=" http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd "
      >
   <display-name
         xml:lang=" fr"
         > MonJar</display-name>
   <enterprise-beans>
      <session>
         <ejb-name> ConfigBean</ejb-name>
         <home> ConfigHome</home>
         <remote> Config</remote>
         <ejb-class> ConfigBean</ejb-class>
         <session-type> Stateful</session-type>
         <transaction-type> Bean</transaction-type>
         <security-identity>
            <use-caller-identity>
            </use-caller-identity>
         </security-identity>
      </session>
   </enterprise-beans>
</ejb-jar>
```