Exercice d'application en Java RMI

Nous disposons de deux services implantés sous forme d'objets `ObjetString` et `ObjetCalcul`: un qui réalise des opérations sur les chaînes de caractère, l'autre qui réalise du calcul numérique. Pour simplifier, l'objet `ObjetString` offre une seule méthode `NbOccurrences` qui calcule le nombre d'occurrences d'un caractère dans une chaîne de caractères (un mot) et l'objet `ObjetCalcul` offre une seule méthode `add` qui calcule la somme de deux nombres. Voici le code de chacune de ces méthodes.

```
public int add (int a, int b) {

    return a+b;
}


public int NbOccurrences(String c, String mot) {
            int longueur=ChaineOrigine.length();
            int Nb=0;
            for (int i=0; i<longueur; i++)
            {
                  if ((ChaineOrigine.substring(i, i+1)).equals(c))
Nb++;
            }
            return Nb;
}
```

1. On souhaite rendre chacune de ces méthodes accessibles à distance. Donnez alors la structure des interfaces qui seront partagées par le serveur et le client sachant que chaque méthode appartient à un objet distinct.

2. Compléter le code de ces méthodes afin qu'elles puissent gérer les erreurs dues à leur appel à distance.

3. Sachant que toute méthode Java appelée par un programme Java distant doit appartenir à un objet accessible à distance.
   Donnez la structure des classes Java qui vont représenter respectivement l'objet `ObjetString` et l'objet `ObjetCalcul`.

4. Si les objets `ObjetString` et `ObjetCalcul` doiventt être installés sur la machine `clementine.cnam.fr`, quelles sont les autres classes Java à implanter sur la machine `clementine`. Donnez leur structure générale.

5. Donnez les commandes à lancer sur la machine `clementine` afin que les méthodes `add` et `NbOccurrences` puissent être appelées par des clients distants si le service de noms est activé sur le port 2001.

6. Ecrire le programme du client qui doit être lancé sur une autre machine.

1. La structure de la classe interface `ObjetStringInterface` est la suivante. Tous les champs en gras sont obligatoires.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ObjetStringInterface extends Remote {
    public int NbOccurrence(String c, String t) throws
RemoteException;
}
```

La structure de la classe interface `ObjetCalculInterface` est la suivante :

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ObjetCalculInterface extends Remote {
    public int add(int a, int b) throws RemoteException;
}
```

2. La méthode `add,` tout comme `NbOccurrences,` doit lever l'exception du type `RemoteException` pour gérer les erreurs générées suite à un appel distant :

```
public int add (int a, int b) throws RemoteException {

    return a+b;
}
```

```
public int NbOccurrences(String c, String mot) throws RemoteException
{
            int longueur=ChaineOrigine.length();
            int Nb=0;
            for (int i=0; i<longueur; i++)
            {
                  if ((ChaineOrigine.substring(i, i+1)).equals(c))
Nb++;
            }
            return Nb;
}
```

3. L'objet `ObjetString` doit contenir la méthode `NbOccurrences`. De même, l'objet `ObjetCalcul` doit définir la méthode `add`. De plus, pour qu'un objet soit accessible à distance, la classe correspondante doit hériter de la classe `UnicastRemoteObject` et implémenter l'interface correspondante définie en 1.

```
import java.rmi.*;
import java.rmi.server.*;
public class ObjetString extends UnicastRemoteObject implements
ObjetStringInterface
{
public ObjetString() throws RemoteException {
    super();
}
```

```
public int NbOccurrence (String c, String ChaineOrigine) throws
RemoteException {

    int longueur=ChaineOrigine.length();
    int Nb=0;
    for (int i=0; i<longueur; i++)
    {
         if ((ChaineOrigine.substring(i, i+1)).equals(c)) Nb++;
    }
    return Nb;
} }
```

```
import java.rmi.*;
import java.rmi.server.*;
public class ObjetCalcul extends UnicastRemoteObject implements
ObjetCalculInterface
{
public ObjetCalcul() throws RemoteException {
    super();
}
public int add (int a, int b) throws RemoteException {

     return (a+b);
  }
}
```

4. Sur la machine `clementine`, on aura à écrire les classes interface `ObjetStringInterface` et `ObjetCalculInterface`, les classes `ObjetString` et `ObjetCalcul` qui représenteront les objets distribués ainsi que le serveur `Serveur`. Comme les autres classes sont déjà données ci-dessus, il reste à écrire la classe du serveur `Serveur`.

```
import java.rmi.*;
import java.rmi.server.*;

public class Serveur {
public static void main(String[] args)
{
      try {
        System.out.println("Serveur : Construction de
l'implémentation");
        ObjetString rev= new ObjetString();
        System.out.println("ObjetString lié dans RMIregistry");
        Naming.rebind("rmi://clementine.cnam.fr:1099/Mot", rev);
          ObjetCalcul r = new ObjetCalcul();
          System.out.println("ObjetCalcul lié dans RMIregistry");
          Naming.rebind("rmi://clementine.cnam.fr:1099/Addition", r);
        System.out.println("Attente des invocations des clients ");
      }
      catch (Exception e) {
        System.out.println("Erreur de liaison de l'objet ObjetCalcul
ou ObjetString");
        System.out.println(e.toString());
      }
} // fin du main
```

```
} // fin de la classe
```

5. Si le service de noms doit être activé sur le port 2001 alors il faudra que les objets `ObjetString` et `ObjetCalcul` soient enregistrés sur un `rmiregistry` de numéro de port 2001. Comme l'enregistrement est fait par le serveur d'objets, on doit modifier les lignes suivantes de la classe `Serveur` :

```
Naming.rebind("rmi://clementine.cnam.fr:1099/Mot", rev);
...
Naming.rebind("rmi://clementine.cnam.fr:1099/Addition", r);
```

Par les lignes suivantes :

```
Naming.rebind("rmi://clementine.cnam.fr:2001/Mot", rev);
...
Naming.rebind("rmi://clementine.cnam.fr:2001/Addition", r);
```

Les commandes suivantes doivent être lancées sur la machine `clementine` afin que les objets `ObjetCalcul` et `ObjetString` soient accessibles par des clients distants :

D'abord compiler les programmes Java (les interfaces, les objets et le serveur).

```
Clementine> javac *.java
```

On obtient des fichiers .class

```
Clementine> ls
ObjetStringInterface.java          ObjetString.java
Serveur.java                       ObjetCalculInterface.java
ObjetCalcul.java

ObjetStringInterface.class         ObjetString.class
Serveur.class                      ObjetCalcul.class
ObjetCalculInterface.class         fichier.policy
```

Générer les stubs associés aux objets `ObjetCalcul` et `ObjetString` :

```
Clementine> rmic –v1.2 ObjetCalcul
Clementine> rmic –v1.2 ObjetString
Clementine> ls
ObjetStringInterface.java          ObjetString.java
Serveur.java                       ObjetCalculInterface.java
ObjetCalcul.java                   fichier.policy

ObjetStringInterface.class         ObjetString.class
Serveur.class                      ObjetCalcul.class
ObjetCalculInterface.class
ObjetString_Stub.class             ObjetCalcul_Stub.class
```

Lancer le service de noms (`rmiregistry`) qui va accueillir les références des objets distribués. Mais pour cela, il doit utiliser une politique de sécurité définie dans un fichier, par exemple `fichier.policy`.

```
Clementine>more fichier.policy
      grant
      {
```

```
      permission java.security.AllPermission;
      };

Clementine>
```

A la lecture de `fichier.policy`, on déduit que `rmiregistry` accepte une requête de n'importe quel programme client, sans restriction sur les numéros de port.

L'activation du service de noms nécessite de spécifier le fichier de sécurité utilisé.
Sous Unix/Linux :

```
Clementine>rmiregistry -J-Djava.security.policy=fichier.policy 2001&
```

Sous Windows :

```
Clementine>start rmiregistry -J-Djava.security.policy=fichier.policy 2001&
```

Une fois le service de noms lancé, on peut lancer le serveur d'objets qui va créer les objets distribués et qui va les enregistrer auprès de `rmiregistry` afin que tout programme client qui demande la référence à un de ces objets puisse l'obtenir en la demandant au service de noms.

```
Clementine>java Serveur &
   ObjetString lié dans RMIregistry
   ObjetCalcul lié dans RMIregistry
   Attente des invocations des clients …
```

6. Voici un exemple de programme client :

```
import java.io.*;
import java.util.*;
import java.rmi.*;

class  Client
{
public static void main (String [] argv) throws IOException
{
      int somme=0,Nb=0;
      String ligne;
      StringTokenizer st;
      String c, mot;

      System.setSecurityManager(new RMISecurityManager());
      BufferedReader entree = new BufferedReader (new
InputStreamReader (System.in));

      ligne= entree.readLine();
      st = new StringTokenizer (ligne);
      c= st.nextToken();
      mot=st.nextToken();
      try {
      ObjetStringInterface s= (ObjetStringInterface) Naming.lookup
("rmi://clementine.cnam.fr:2001/Mot");
      somme=s.NbOccurrence(c, mot);

      while (st.hasMoreTokens()){
          mot=st.nextToken();
          Nb=s.NbOccurrence(c, mot);
```

```
            ObjetCalculInterface a=
(ObjetCalculInterface)Naming.lookup("rmi://clementine.cnam.fr:2001/Ad
dition");
            somme = a.add(Nb, somme);
            }
      System.out.println(" Dans la phrase /"+ligne+"/, il y a
"+somme+" occurrences de " + c);
      }catch (Exception e) {
          System.out.println("Erreur d'acces a un objet distant");
          System.out.println(e.toString());
      }
    }
}
```

Le répertoire du programme client doit contenir les interfaces distantes utilisées par le client ainsi que les stubs des objets distants, qui sont les représentants locaux de ces objets.

```
AutreMachine>ls
Client.java ObjetCalculInterface.java ObjetStringInterface.java
ObjetCalcul_Stub.java   Objetstring_Stub.java
fichier.policy

AutreMachine>javac Client.java
```

Ensuite lancer ll'exécution du client en spécifiant la même politique de sécurité que celle du registre rmi.

```
AutreMachine>java –Djava.security.policy=fichier.policy Client
e il etait une fois dans l'ouest
Dans la phrase /e il etait une fois dans l'ouest/, il y a 3
occurrences de e
AutreMachine>
```

Exercice d'application en CORBA

Soit la spécification IDL suivante partagée par le serveur et le client et qui offre deux interfaces : une pour traiter une chaîne de caractères, l'autre pour incrémenter un compteur distant :

```
module ServicesApp
{
 exception NbDeMotsIllegal { };

 interface ObjetString
 {
       long NombreDeMots(in string chaine) raises (NbDeMotsIllegal);
       long NbOccurrences(in string c, in string mot);
 };

 interface ObjetCalcul {
       attribute long somme;
       void increment(in long pas);
       };

};
```

1. En supposant qu'on utilise le compilateur `idlj` de la JDK pour faire la projection en Java, donnez la commande nécessaire pour générer le stub d'un client se trouvant sur une machine A. Citez tous les fichiers générés dans ce cas.

2. Utiliser le compilateur `idlj` pour générer le squelette côté serveur. Citez tous les fichiers générés dans ce cas sachant que le serveur se trouve sur une machine B.

3. Ecrire le programme du serveur sachant que ce dernier doit créer deux servants (deux objets) étant donné que la spécification IDL offre deux interfaces.

4. Compiler l'ensemble des classes Java y compris le serveur que l'on vient d'écrire afin de lancer le service de nommage de CORBA sur le port 2000.

5. Ecrire le programme du client sur la machine A. Le client a pour rôle de lire une phrase saisie par l'utilisateur, d'afficher le nombre de mots la composant ainsi que le nombre d'occurrences d'un caractère dans la phrase.

6. La classe `NbDeMotsIllegal.java` générée du côté client et du côté serveur est donnée ci-dessous. Compléter cette classe au niveau du client afin qu'elle puisse générer une erreur si le nombre de mots d'une phrase est plus petit que 2.

```
package ServicesApp;


/**
```

```
* ServicesApp/NbDeMotsIllegal.java
* Generated by the IDL-to-Java compiler (portable), version "3.0"
* from Specification.idl
* mercredi 10 mars 2004 13 h 44 CET
*/

public final class NbDeMotsIllegal extends
org.omg.CORBA.UserException implements
org.omg.CORBA.portable.IDLEntity
{

  public NbDeMotsIllegal ()
  {
  } // ctor

} // class NbDeMotsIllegal
```

Vous pourrez remarquer qu'il est indispensable de compléter la classe `NbDeMotsIllegal.java` afin qu'elle puisse traiter l'erreur côté client. Néanmoins, il n'est pas nécessaire de la compléter au niveau du serveur car c'est la classe locale du client qui est appelée pour traiter l'erreur détectée.

7. Compiler le client en même temps que les autres fichiers générés par projection. Exécuter le client.

1. La commande est la suivante (pour JDK1.3 ou sup) :

```
machineA> ls
machineA> Specification.idl
machineA> idlj -fclient Specification.idl
machineA>ls
Specification.idl       ServicesApp
machineA> ls ServicesApp
_ObjetCalculStub.java              ObjetCalculHolder.java
_ObjetStringStub.java              ObjetCalculOperations.java
NbDeMotsIllegal.java               ObjetString.java
NbDeMotsIllegalHelper.java         ObjetStringHelper.java
NbDeMotsIllegalHolder.java         ObjetStringHolder.java
ObjetCalcul.java                   ObjetStringOperations.java
ObjetCalculHelper.java
machineA>
```

Nous constatons que pour chaque interface IDL, les fichiers suivants sont générés :

1. un fichier `<InterfaceIDL>Operations` qui définit les méthodes distantes à invoquer
2. un fichier `<InterfaceIDL>Helper` qui contient les méthodes de lecture, d'écriture et de conversion
3. un fichier `_<InterfaceIDL>Stub` qui représente le proxy
4. et un fichier `<InterfaceIDL>Holder` qui gère les paramètres out et inout.

Concernant l'exception `NbDeMotsIllegal` définie dans le module IDL, une classe java de même nom est générée ainsi que des classes de type `Helper` et `Holder`.

2. Pour générer le squelette du serveur, on lance la commande suivante sur la machine B :

```
machineB> ls
machineB> Specification.idl
machineB> idlj -fserver Specification.idl
machineB>ls
Specification.idl       ServicesApp
machineB> ls ServicesApp
_ObjetCalculImplBase.java    ObjetCalcul.java
_ObjetStringImplBase.java    ObjetCalculOperations.java
NbDeMotsIllegal.java         ObjetString.java
NbDeMotsIllegalHelper.java   ObjetStringOperations.java
NbDeMotsIllegalHolder.java
machineB>
```

Notons que deux squelettes `_ObjetCalcul.java` et `_ObjetStringImplBase.java` sont générés : un pour chaque interface IDL.

3. Le programme du serveur est le suivant :

```
// le package qui contient les stubs
import ServicesApp.*;

//traitement des Tokens
import java.util.*;

// ce lanceur de serveur va utiliser le service de nommage
```

```
import org.omg.CosNaming.*;

//inclure le package des exceptions pouvant etre generees par le
//service de nommage
import org.omg.CosNaming.NamingContextPackage.*;

// tous les objets CORBA ont besoin de ces classes
import org.omg.CORBA.*;

public class Serveur {
      public static void main(String args[]) {
      try {
      // creer et initialiser l'ORB
      ORB orb=ORB.init(args, null);

      // creer une instance du servant et l'enregistrer dans l'ORB
      ObjetStringServant revRef1= new ObjetStringServant();
      ObjetCalculServant revRef2= new ObjetCalculServant();
      orb.connect(revRef1);
      orb.connect(revRef2);

      // recuperer le contexte de nommage
      org.omg.CORBA.Object objRef=
      orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(objRef);

      // inscrire la reference de l'objet dans le service de nommage
      // pas d'espace entre " et "
      NameComponent nc1 = new NameComponent("MyStringObject","");
      NameComponent nc2 = new NameComponent("MyComputingObject","");

      NameComponent path1[] = {nc1};
      NameComponent path2[] = {nc2};
      ncRef.rebind(path1, revRef1);
      ncRef.rebind(path2, revRef2);;

      // attendre les requetes des clients
      java.lang.Object sync = new java.lang.Object();
      synchronized (sync) {
            sync.wait();
        }
      } catch(Exception e) {
            System.err.println("Erreur : "+e);
            e.printStackTrace(System.out);
            }
      }
 } // fin du serveur


 class ObjetCalculServant extends _ObjetCalculImplBase {
      private int _somme;

      public int somme () { return _somme; }

      public void somme(int newSomme) { _somme=newSomme; }

      public void increment (int pas) {
            _somme=_somme+pas;
      }
} // fin de ObjetCalculServant
```

```
class ObjetStringServant extends _ObjetStringImplBase {

    public int NombreDeMots (String chaine) throws NbDeMotsIllegal {
      String mot;
      StringTokenizer st= new StringTokenizer(chaine);
      int nb=0;

      while(st.hasMoreTokens()) {
           nb++;
           mot=st.nextToken();
      }
      if (nb<2) throw new NbDeMotsIllegal();
      return nb;
      }

      public  int NbOccurrences(String c, String mot) {
        int longueur=mot.length();
        int Nb=0;

        for (int i=0; i<longueur; i++) {
            if ((mot.substring(i, i+1)).equals(c)) Nb++;
        }
        return Nb;
   }

} // fin de ObjetStringServant
```

4. On procède d'abord à la compilation de tous les programmes Java :

```
machineB>ls
Specification.idl Serveur.idl   ServicesApp
machineB> javac Serveur.java ServicesApp/*.java
machineB>
```

Ensuite on lance le service de nommage CORBA ainsi que le serveur :

```
machineB> tnameserv -ORBInitialPort 2000 &
Initial Naming Context:
IOR:000000000000002849444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e6
7436f6e746578743a312e3000000000010000000000000005c000101000000000f3136332e31
37332e3133362e33320000804700000000019afabcaff000000024ad14e930000000800000
0000000000010000000000000010000000100000014000000000001002000000000000010100
00000000
TransientNameServer: setting port for initial object references to: 2000
Ready.
machineB>
machineB> java Serveur -ORBInitialPort 2000&
machineB>
```

5. Voici un exemple de programme client :

```
import ServicesApp.*; // le package qui contient les stubs
import org.omg.CosNaming.*; // inclure le service de nommage
import org.omg.CORBA.*; // pour generer des objets CORBA
import java.util.*; // pour les Tokens
import java.io.*; // pour les entrées/sorties

public class Client {
public static void main (String args[]) {
try {
```

```
      // creer et initialiser l'ORB
      ORB orb = ORB.init(args, null);

      //recuperer le contexte de nommage
      org.omg.CORBA.Object objRef =
      orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(objRef);

      // inscrire la reference de l'objet dans le service de nommage
      // pas d'espace entre " et "
      NameComponent nc1 = new NameComponent("MyStringObject","");
      NameComponent [] chemin1 = {nc1};

      NameComponent nc2 = new NameComponent("MyComputingObject","");
      NameComponent [] chemin2 = {nc2};

      ObjetString stringRef = ObjetStringHelper.narrow
(ncRef.resolve(chemin1));
      ObjetCalcul calculRef = ObjetCalculHelper.narrow
(ncRef.resolve(chemin2));

      // faire appel aux objets servant
      BufferedReader entree = new BufferedReader(new
      InputStreamReader(System.in));
      String ligne= entree.readLine();

      try {
      int nbMots=stringRef.NombreDeMots(ligne);
      System.out.println("La phrase saisie est composee de " + nbMots
+ " mots");
      }
      catch (NbDeMotsIllegal e) {
            System.out.println("Erreur : " + e);
            System.exit(0);
      }

      StringTokenizer st= new StringTokenizer(ligne);
      String c= st.nextToken();
      String mot=st.nextToken();
      calculRef.somme(stringRef.NbOccurrences(c, mot));

      while(st.hasMoreTokens()) {
            mot=st.nextToken();
            calculRef.increment(stringRef.NbOccurrences(c, mot));
      }

      System.out.println("Dans la phrase /" + ligne + "/, il y a " +
      calculRef.somme()+ " occurrences de " + c);

      }
     catch(Exception e) {
            System.out.println("Erreur : " + e);
            e.printStackTrace(System.out);
            }
      } // fin du main

} // fin de la classe
```

6. Toute classe qui hérite de la classe `Exception` doit contenir une méthode `toString()`. Cette méthode est appelée à chaque fois qu'on fait appelle à cette classe suite à un erreur :

```
package ServicesApp;

/**
* ServicesApp/NbDeMotsIllegal.java
* Generated by the IDL-to-Java compiler (portable), version "3.0"
* from Specification.idl
* mercredi 10 mars 2004 14 h 06 CET
*/

public final class NbDeMotsIllegal extends
org.omg.CORBA.UserException implements
org.omg.CORBA.portable.IDLEntity
{
  public NbDeMotsIllegal ()
  {
  } // ctor

  public String toString() {
      return ("La chaine doit etre composee d'au moins deux mots!");
  }

} // class NbDeMotsIllegal
```

7. Le programme du client doit être d'abord compilé :

```
machineB>javac Client.java   ServicesApp/*.java
machineB>ls
Client.class  Client.java  ServicesApp     Specification.idl
machineB>
```

Le Client est lancé en spécifiant le numéro de port du service de nommage CORBA ainsi que le nom de la machine du serveur.

```
machineA>java Client -ORBInitialHost clementine.cnam.fr -ORBInitialPort
2000
e il etait une fois dans l'ouest
La phrase saisie est composee de 7 mots
Dans la phrase /e il etait une fois dans l'ouest/, il y a 3
occurrences de e
machineA>
```

Autre exécution :

```
machineA>java Client -ORBInitialHost clementine.cnam.fr -ORBInitialPort
2000
e
La chaine doit etre composee d'au moins deux mots!
machineA>
```

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

    }
}
```

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 la machine virtuelle java, par exemple J2SDK1.4 dans le répertoire c:\java
b. N'oubliez pas de mettre dans la variable d'environnement PATH le chemin des fichiers exécutables (javac, java, rmic, etc.). Dans notre cas, ce sera le chemin C:\java\bin.
c. Installer un serveur d'application, par exemple j2eesdk-1_4-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 J2EESDK (de 100,59 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 boite de dialogue vous demandera de saisir un mot de passe pour l'utilisateur admin qui sera l'administrateur du serveur d'application.

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.

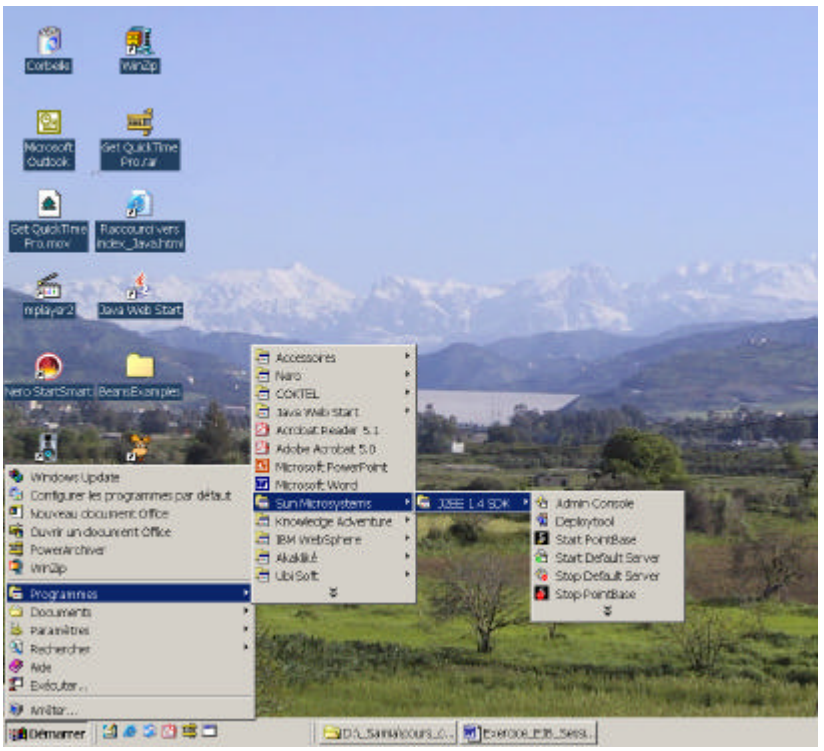
d. Compiler sur une fenêtre MSDOS les différents programmes JAVA.

```
>javac *.java
>
```

e. Démarrer le serveur :

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

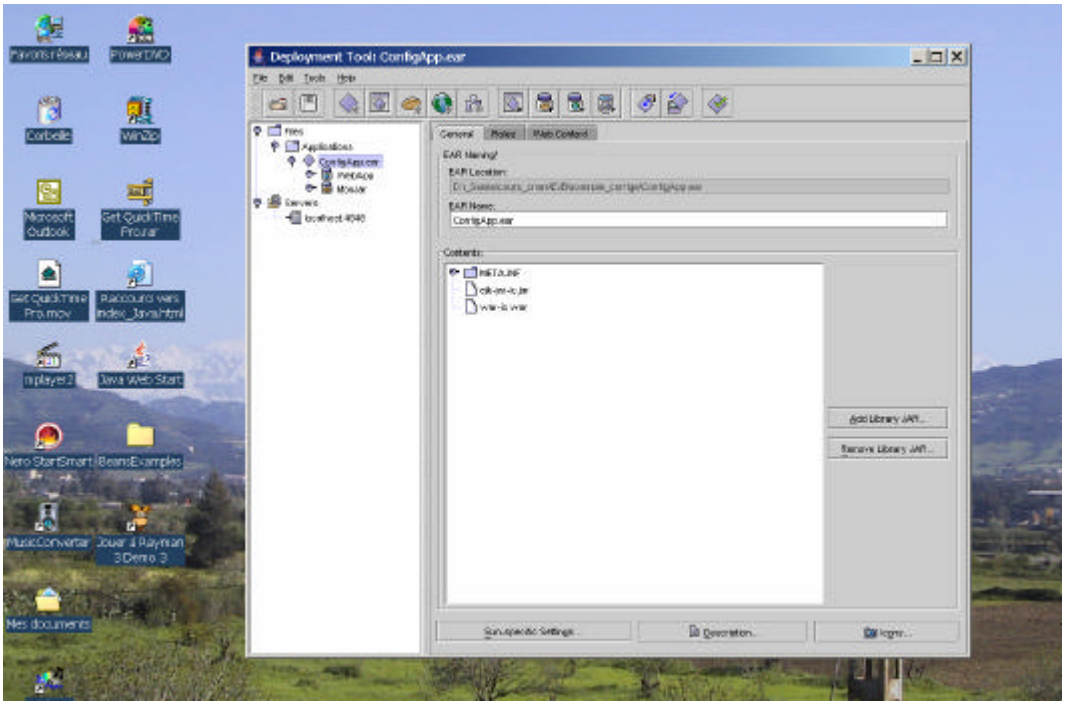
Une fenêtre apparaît. Lorsque le message « Press any key to continue … » s'affiche, pressez sur une touche et la fenêtre disparaît (mais le serveur reste actif).



f. 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.

g. 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

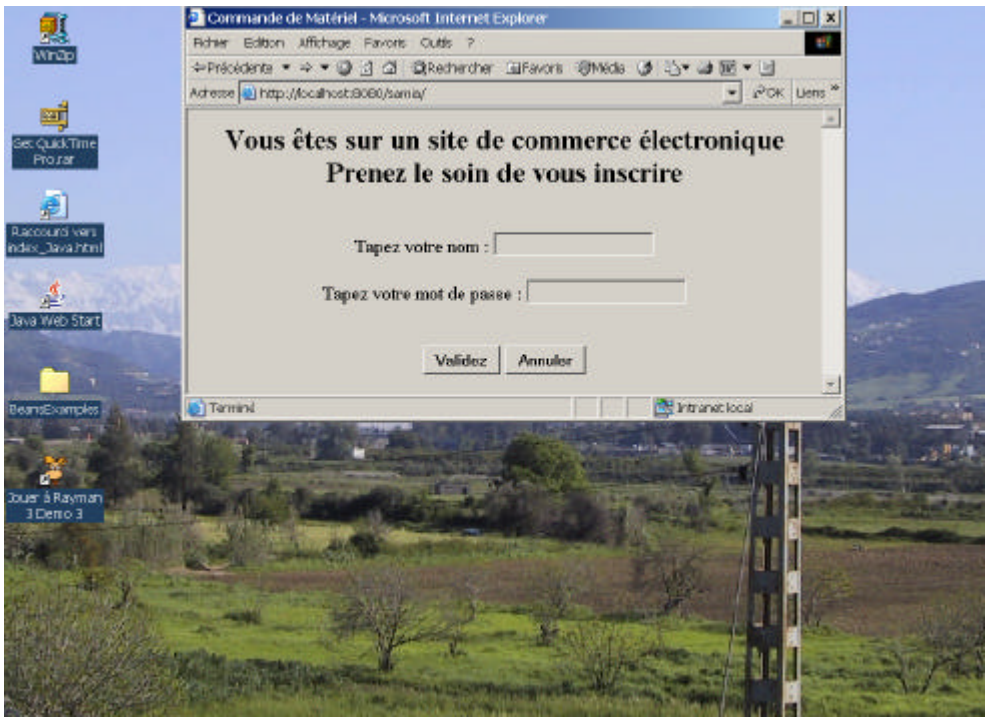
h. L'étape qui suit est celle qui permet de construire l'application J2EE. L'application est construite grâce à un outil d'assemblage et de déploiement qu'on peut lancer en choisissant l'option :
Programmes=> Sun MicroSystems =>J2EE 1.4 SDK => Deploytool.
Cet outil est une interface graphique qui nous guide pour créer
   - une application ConfigApp.ear. Un descripteur XML décrivant l'application est généré automatiquement.
   - Un composant Web intégré à l'application. 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. Il existe un champ qui s'appelle *aliases* associé à la servlet dans lequel il faudra définir un alias vers la servlet. Dans notre exemple on choisit l'alias */ConfigurationServlet* pour la servlet *ConfigurationServlet*.
   - 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.

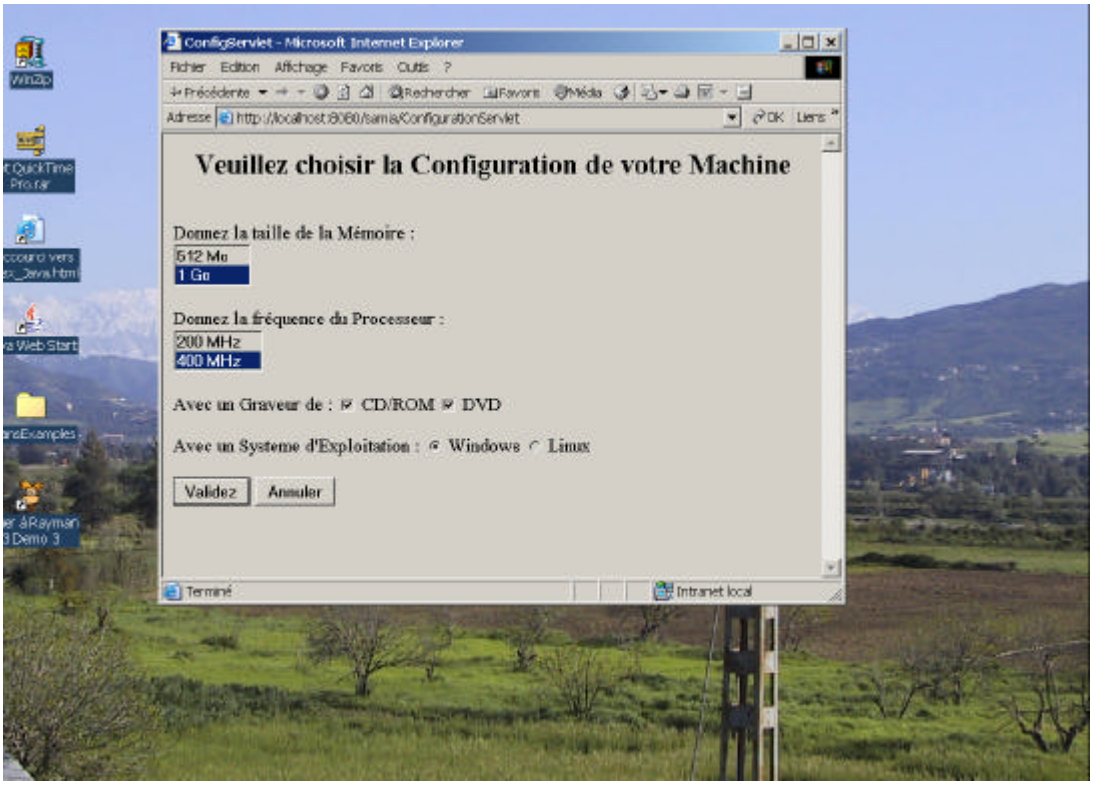


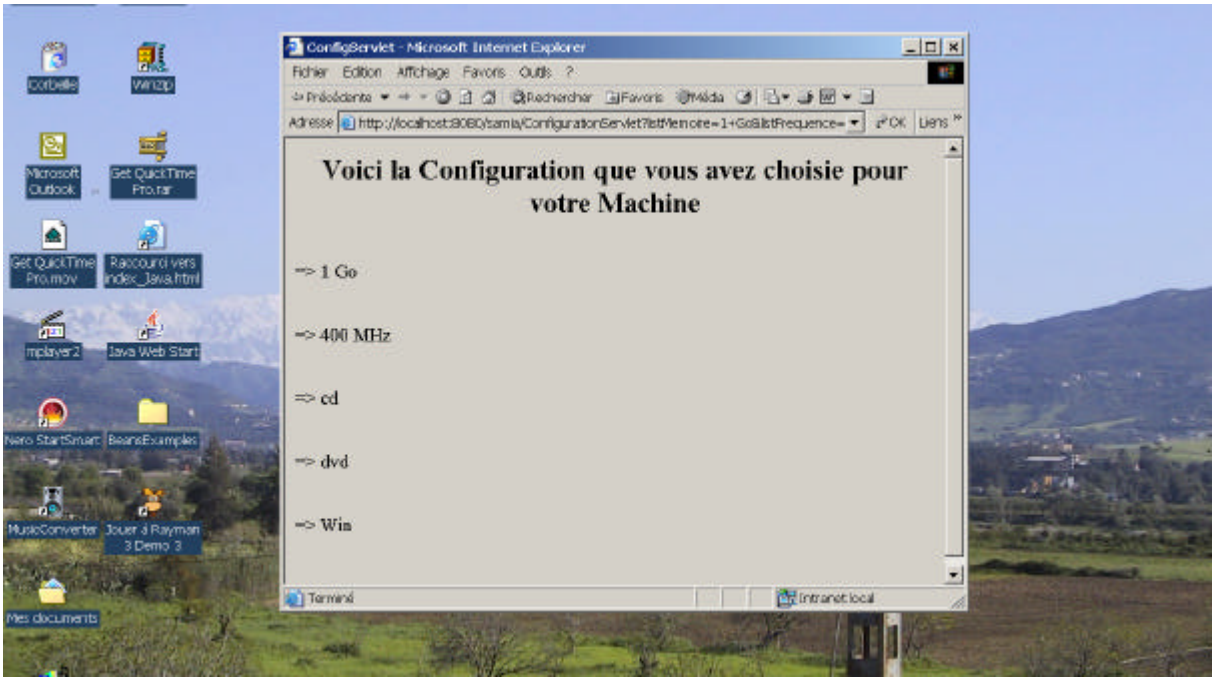
i. Dans un champ *Context root*, associer un nom qui sera utilisé pour appeler votre page d'acceuil. Dans notre exemple, nous avons choisi */samia*.
j. Une fois l'application sauvegardée, il faut la déployer. Choisir dans le menu de l'outil d'assemblage et de déploiement l'option Tools=>deploy. Pour 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.
k. Si le déploiement s'est déroulé sans erreur, alors recopier le fichier ConfigApp.ear (zippé) 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.
l. IL reste alors à vérifier que l'application marche bien en tapant : http://localhost:8080/samia.
m. 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.


Commande de Matériel - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
Précédente Rechercher Favoris Média
Adresse http://localhost:8080/samia/
Vous êtes sur un site de commerce électronique
Prenez le soin de vous inscrire
Tapez votre nom :
Tapez votre mot de passe :
Validez
Annuler
Terminé
Intranet local


ConfigServlet - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
Précédente Rechercher Favoris Média
Adresse http://localhost:8080/samia/ConfigurationServlet
Veuillez choisir la Configuration de votre Machine
Donnez la taille de la Mémoire :
512 Mo
1 Go
Donnez la fréquence du Processeur :
200 MHz
400 MHz
Avec un Graveur de : CD/ROM DVD
Avec un Systeme d'Exploitation : Windows Linux
Validez
Annuler
Terminé
Intranet local



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>
```