Exercice d'application en Java RMI

Nous disposons d'un service qui offre des opérations de gestion de son compte courant. Voici le code des méthodes offertes par ce service :

```
void debiter(double montant) {

      }

void crediter(double montant) {

      }

 double lire_solde() {

      }
```

1. On souhaite rendre chacune de ces méthodes accessibles à distance de manière à ce qu'elles définissent l'interface entre le client et le serveur. Ecrire cette interface.

2. Déduire la classe qui matérialise le service qui offre les opérations debiter(), crediter() et lire_solde().

3. Compléter le fichier suivant : Serveur.java pour permettre l'enregistrement du service auprès de RMI Registry.

```
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");
        Compte cpt= new Compte(15.50);
        System.out.println("Objet Compte enregistré dans
RMIregistry");
        // a completer

        System.out.println("Attente des invocations des clients ");
      }
      catch (Exception e) {
        System.out.println("Erreur de liaison de l'objet Compte");
        System.out.println(e.toString());
      }
```

```
} // fin du main
} // fin de la classe
```

4. Générer tous les fichiers nécessaires au lancement du Serveur.

5. Lancer la suite de commandes ayant pour finalité le lancement du serveur.

6. Compléter le programme du client Client.java qui doit être lancé à partir d'un autre répertoire ou d'une autre machine.

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

class  Client
{
public static void main (String [] argv) throws IOException
{
      if(argv.length != 2){
            System.out.println("Usage : java Client <nombre>
<operation>");
            System.exit(1);
          }
// operation = 1: credit, 2: dedit

      System.setSecurityManager(new RMISecurityManager());

      double valeur = Double.parseDouble(argv[0]);
      int operation = Integer.parseInt(argv[1]);

      try {
           // a completer : CompteInterface cpt = …

          if (operation==1) cpt.crediter(valeur);
         if (operation ==2) cpt.debiter(valeur);
         System.out.println ("Votre solde courant = " +
cpt.lire_solde() + " euros");


       }catch (Exception e) {
          System.out.println("Erreur d'acces a un objet distant");
          System.out.println(e.toString());
         }
    }
}
```

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

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


public interface CompteInterface extends java.rmi.Remote
{

      void debiter(double montant)
            throws java.rmi.RemoteException;

      void crediter(double montant)
            throws java.rmi.RemoteException;

      double lire_solde()
            throws java.rmi.RemoteException;

};
```

2. La structure de la classe d'implémentation `Compte` est la suivante. Tous les champs en gras sont obligatoires.

```
import java.rmi.*;
import java.rmi.server.*;
public class Compte extends UnicastRemoteObject implements
CompteInterface
{
      private double  solde;

      public Compte(double s) throws java.rmi.RemoteException
      {
            super();
            solde=s;
      }

      public void crediter(double montant)
                              throws java.rmi.RemoteException
      {
            solde=solde+montant;
      }

      public void debiter(double montant)
                               throws java.rmi.RemoteException
      {
            solde=solde-montant;
      }

      public double lire_solde()
            throws java.rmi.RemoteException
      {
            return solde;
      }
}
```

3. La classe Serveur est la suivante :

```
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");
        Compte cpt= new Compte(15.50);
        System.out.println("Objet Compte enregistré dans
RMIregistry");
        Naming.rebind("rmi://machine.cnam.fr:1099/CompteCourant",
cpt);

        System.out.println("Attente des invocations des clients ");
      }
      catch (Exception e) {
        System.out.println("Erreur de liaison de l'objet Compte");
        System.out.println(e.toString());
      }
} // fin du main
} // fin de la classe
```

4. Si le service de noms doit être activé sur un port différent de 1099, par exemple 2001, alors il faudra que l'objet `Compte` soit enregistré sur le serveur `rmiregistry` lancé sur le numéro de port 2001. Comme l'enregistrement est fait par le serveur d'objets, on doit modifier la ligne suivante de la classe `Serveur` :

```
Naming.rebind("rmi://machine.cnam.fr:1099/CompteCourant", cpt);
```

Par la ligne suivante :

```
Naming.rebind("rmi://machine.cnam.fr:2001/CompteCourant", cpt);
```

D'abord compiler les programmes Java (l'interface, l'objet et le serveur).

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

On obtient des fichiers .class

```
Machine_serveur> ls
      CompteInterface.java          Compte.java       Serveur.java

      CompteInterface.class         Compte.class            Serveur.class

      fichier.policy
```

Générer le stub associé à l'objet `Compte`:

```
Machine_serveur> rmic –v1.2 Compte
Machine_serveur> ls
      CompteInterface.java          Compte.java       Serveur.java
      fichier.policy

      CompteInterface.class         Compte.class            Serveur.class
      Compte_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`.

```
Machine_serveur>more fichier.policy
      grant
      {
      permission java.security.AllPermission;
      };

Machine_serveur>
```

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 :

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

Sous Windows :

```
Machine_serveur>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 l'objet distribué et qui va l'enregistrer auprès de `rmiregistry` afin que tout programme client qui demande la référence à cet objet puisse l'obtenir en la demandant au service de noms.

```
Machine_serveur>java Serveur &
   Compte enregistré dans RMIregistry
   Attente des invocations des clients …
```

6. Voici un exemple de programme client :

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

class  Client
{
public static void main (String [] argv) throws IOException
{
      if(argv.length != 2){
            System.out.println("Usage : java Client <nombre>
<operation>");
            System.exit(1);
          }
// operation = 1: credit, 2: dedit

      System.setSecurityManager(new RMISecurityManager());

      double valeur = Double.parseDouble(argv[0]);
      int operation = Integer.parseInt(argv[1]);

      try {
           CompteInterface cpt= (CompteInterface) Naming.lookup
("rmi://machine_serveur.cnam.fr:1099/CompteCourant");
```

```
          if (operation==1) cpt.crediter(valeur);
         if (operation ==2) cpt.debiter(valeur);
         System.out.println ("Votre solde courant = " +
cpt.lire_solde() + " euros");



       }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 le stub de l'objet distant, qui est le représentant local de cet objet.

```
MachineClient>ls
Client.java CompteInterface.class Compte_Stub.class
fichier.policy

MachineClient>javac Client.java
```

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

```
MachineClient>java -Djava.security.policy=fichier.policy Client 11.50
1
MachineClient>
Votre solde courant = 4 euros
```

Remarque : Pour les différents tests en local, remplacer les noms des machines du serveur et du client par « localhost ».