TP 3 | Applications Clients - Serveurs : RMI

Code 0.1 HelloInterface.java

TP 1

Créer un répertoire RMI. Créer sous RMI le répertoire Hello et tester
Tester l'application helloworld suivante.

```
// Interface de Hello
import java.rmi.*;
/* Remote Interface HelloInterface pour l'application Hello */ .
public interface HelloInterface extends Remote {
public String echo () throws RemoteException;
}
```

Code 0.2 HelloImpl.java

```
// Implémentation de l'interface Hello
import java.rmi.*;
import java.rmi.server.*;

public class HelloImpl extends UnicastRemoteObject implements Hello {
public HelloImpl () throws RemoteException {
    super();  }
public String echo () throws RemoteException {
    return "Hello World ";  }
}
```

Code 0.3 client.java

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

// programme  client pour l'exemple  "Hello, world!"
public class Client{
  public static void main (String[] argv) {
    try {
      int port = 8000;
      HelloInterface obj  =
        (HelloInterface) Naming.lookup ("rmi://localhost:port/hello");
      System.out.println (obj.echo());
    } catch (Exception e) {
      System.out.println ("HelloClient exception: " + e);
    }
  }
}
```

Code 0.4 serveur.java

```
// LE SERVEUR HELLO
import java.rmi.*;
import java.rmi.server.*;
public class serveur{
  public static  void main (String[] argv) {
    try {
      int port = 8000;
     // Naming.rebind ("hello", new HelloImpl());
      LocateRegistry.createRegistry(port);
    Naming.rebind ("rmi://localhost:port/hello", new HelloImpl());
      System.out.println ("Hello Server prêt ! .");
    } catch (Exception e) {
      System.out.println ("Hello Server échec " + e);
    }
  }
}
```

Code 0.5 Compilation et exécution

```
javac HelloInterface.java
javac HelloImpl.java
rmic HelloImpl
javac serveur.java
javac client.java
rmiregistry&
java serveur
java client
```

TP 2 *Il s'agit d'écrire un service de conversion entre degrés Celsius et Fahreinheit.*
La formule de conversion est $C = (5/9)(F - 32)$
Ecrire :

1. *L'interface ConvInterface dans un fichier ConvInterface.java*
2. *La classe Conv implémentant le service dans le fichier Conv.java.*
3. *Le serveur dans un fichier serveur.java*
4. *Le client dans un fichier client.java*

Compiler puis exécuter pour des valeurs allant de -20 degrés à 120 degrés par pas de 10.

TP 3 *Implémenter sous java RMI une application qui résout une équation du second degré*

TP 4 Un serveur de calcul
Ecrire un serveur de calcul dont l'interface est donnée ci-après :

Code 0.6 Interface du serveur de calcul

```
// CalculateurInteface.java
public interface CalculateurInterface extends java.rmi.Remote {

    public double add(double a, double b) throws java.rmi.RemoteException;

    public double sub(double a, double b) throws java.rmi.RemoteException;

    public double mul(double a, double b) throws java.rmi.RemoteException;

    public double div(double a, double b) throws java.rmi.RemoteException;
}
```

1. *Ecrire l'implémentation du calculateur Calculateur.java*

2. *Ecrire Le serveur et le client*

3. *Compiler et exécuter.*

TP 5 Gestion à distance de comptes bancaires

Ecrire une application répartie permettant de gérer des comptes bancaires. Un serveur gérera tous les comptes bancaires et permettra à des clients de se connecter et d'effectuer les opérations suivante :

1. *Créer un compte en banque.*

2. *Consulter la position d'un compte*

3. *Ajouter une somme sur un compte*

4. *Retirer une somme d'un compte.*

Voici la déclaration des méthodes distantes :

Code 0.7 Voici l'interface

```
void creerCompte(String id, double sommeInitiale);
void ajouter(String id, double somme);
void retirer(String id, double somme);
Position position(String id);
```

Où id est une chaîne permettant d'identifier un compte et Position est la classe suivante :

```
public class Position {
public double solde;
public Date derniereOperation;
public Position(double solde) {
    this.solde = solde;
    this.derniereOperation = new Date();
}}
```

1. *Ecrire une interface Banque dérivant de Remote qui déclare les méthodes distantes.*

2. *Ecrire la classe Compte qui permet de consulter la position d'un compte, d'ajouter et de retirer une somme à un compte.*

3. *Ecrire une classe BanqueImpl qui gère la partie serveur de notre application répartie. Les comptes seront stockés dans une Hashtable qui permettra de retrouver un compte à partir de son identification.*

4. *Ecrire une classe BanqueClient qui gère la partie client de notre application répartie.*

TP 6 *On souhaite mettre en place un service d'annuaire permettant d'enregistrer des noms et des numéros de téléphone et permettant de retrouver un numéro à partir d'un un nom. L'annuaire doit être accessible à distance par RMI.*

1. *On considère que le nom et le numéro de téléphone sont des chaînes de caractères.*
   (a) *Définir une interface RMI répondant à ces spécifications.*
   (b) *Développer un objet serveur implémentant cette interface.*
   (c) *Développer un client interrogeant l'objet serveur précédent*
2. *On souhaite, en plus du numéro de téléphone, stocker pour chaque personne son prénom, son adresse et le nombre de fois où cette personne a été recherchée dans l'annuaire.*
   (a) *Définir une classe Personne stockant ces informations.*
   (b) *Modifiez les programmes de la question précédente afin que l'annuaire stocke pour chaque individu, une instance de cette classe.*