TD sur les RPC

I. Principes de base de java RMI

a) On veut mettre en place avec RMI un objet proposant la méthode distribuée de signature suivante : **int m (P p) {…}**. Cette méthode est implémentée dans une classe **C** et sa signature est définie dans une interface **I**. Indiquer les caractéristiques de la méthode **m**, de l'interface **I**, de la classe **C**, de la classe **P** (le paramètre **p** est transmis par copie du client vers le serveur) en terme d'héritage, d'implémentation d'interface(s) et/ou de levée(s) d'exception(s)). Et si **p** est transmis par référence ?

b) Avant de pouvoir appeler les méthodes d'un objet distant, le client doit disposer de l'objet stub correspondant. Où le trouve t-il et comment le récupère t-il ? Et la classe stub, où le client la trouve t-il et/ou comment la récupère t-il ?

II. Java RMI. Passage d'un paramètre objet à une méthode distante par valeur et par adresse

Voici les classes et interfaces d'un petit programme client/serveur en Java RMI :

1) TraitementsInterface.java

```
import java.rmi.*;

interface TraitementsInterface extends Remote{
   public void vieillirPersonne(Personne p) throws RemoteException;
}
```

2) PetitClient.java (en paramètre : le nom ou l'adresse IP du serveur)

```
import java.rmi.*;

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

         if (args.length<1){
            System.out.println("Nom d'hôte SVP");
            return;
         }
         String nom_hote=args[0];

         Personne p=new Personne ("Lucky", "Luke", 30);

         TraitementsInterface traitementsInterface
=(TraitementsInterface)Naming.lookup("rmi://"+nom_hote+":1099/traitements");
         traitementsInterface.vieillirPersonne(p);

         p.afficherAge();

      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

3) Personne.java

```
import java.io.*;

class Personne implements Serializable{
   private String nom;
```

```
    private String prenom;
    private int age;

    public Personne(String nom, String prenom, int age){
        this.nom=nom;
        this.prenom=prenom;
        this.age=age;
    }

    public void vieillir(){
        age++;
    }

    public void afficherAge(){
        System.out.println(prenom+" "+nom+" a "+age+ "ans");
    }
}
```

4) Traitements.java

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

class Traitements extends UnicastRemoteObject implements TraitementsInterface{

    public Traitements () throws RemoteException{
        super(); // constructeur de la classe mère
    }

    public void vieillirPersonne(Personne p) throws RemoteException {
        p.vieillir();
    }
}
```

5) PetitServeur.java

```
import java.rmi.*;
import java.net.*;
import java.rmi.registry.*;

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

            LocateRegistry.createRegistry(1099);

            Traitements traitements= new Traitements();
            String url="rmi://"+InetAddress.getLocalHost().getHostAddress()+"/traitements";

            Naming.rebind(url, traitements);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

a) De quel côté (client ou serveur) doit-on placer ces différentes interfaces et classes (une fois compilées). A quoi correspond chacune ? Que manque t-il et comment l'obtient-t-on ? Qu'est ce qui est affiché par le client ?
b) Que doit-on modifier dans le code des classes et interfaces précédentes pour que le paramètre objet soit maintenant passé par adresse. Comment doit-on maintenant répartir les différentes classes et interfaces entre le client et le serveur. Qu'est ce qui est maintenant affiché par le client ?

III. Java RMI. Mécanisme du Callback.

Voici un petit programme de tchat en Java RMI. Le programme serveur gère une liste de client enregistrés. Il transfère alors chaque message qu'il reçoit à chacun des clients de sa liste. Le programme client s'enregistre auprès du serveur (pour recevoir des messages), lui envoie les messages saisis par l'utilisateur et se désenregistre quand ce dernier saisit la valeur "-1". Compléter le code suivant aux endroits indiqués **en gras**. De quel côté doit-on placer les différentes classes et interfaces ?

ServeurTchat.java

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

public class ServeurTchat extends UnicastRemoteObject implements ServeurTchatInterface{

   Vector clients=new Vector();

   public ServeurTchat () throws RemoteException {
      super();
   }

    /* en-tête de la méthode enregistrementClient(...) */
   ……..
      clients.add(client);
   }

    /* en-tête de la méthode desenregistrementClient(...) */
   ……..
      for (int i=0; i<clients.size(); i++)
         if (client.equals((ClientTchatInterface)(clients.get(i))))
            clients.removeElementAt(i);
   }

   public int nbClientsEnCours() throws RemoteException {
      return clients.size();
   }

   /* En-tête de la méthode transfertMessage(...) */
   ……..
      for (int i=0; i<clients.size(); i++){
         ClientTchatInterface client= (ClientTchatInterface)(clients.get(i));
         client.recuperationNouveauMessage(msg);
      }
   }

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

         java.rmi.registry.LocateRegistry.createRegistry(1099);

         String url="rmi://"+InetAddress.getLocalHost().getHostAddress()+"/tchat";

         /* On enregistre dans la rmiregistry un objet de la classe courante */
         ……..

      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

ServeurTchatInterface.java

```
import java.rmi.*;

interface ServeurTchatInterface extends Remote{
   /* A compléter */
   ……..
   ……..
   ……..
   public int nbClientsEnCours() throws RemoteException;
}
```

ClientTchat.java (en paramètre du programme : le nom ou l'adresse IP du serveur et un pseudo)

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

public class ClientTchat extends UnicastRemoteObject implements ClientTchatInterface  {

   public ClientTchat()throws RemoteException{
      super();
   }

   /* Méthode recuperationNouveauMessage(...)  qui affiche un message reçu depuis le serveur */
   ……..
   ……..
   ……..

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

         if (args.length<2){
            System.out.println("Nom d'hôte puis pseudo en ligne de commande SVP");
            return;
         }
         String nom_hote=args[0];
         String pseudo=args[1];

         ServeurTchatInterface serveurTchatInterface =
(ServeurTchatInterface)Naming.lookup("rmi://"+nom_hote+":1099/tchat");

         /* Création d'un objet de la classe courante et enregistrement de celui-ci auprès du serveur */
         ……..
         ……..

         System.out.println ("Discussion (-1 pour la quitter) : ");
         while(true){
            Scanner keyboard = new Scanner(System.in);
            String s=keyboard.nextLine();
            if(s.equals("-1")){
               /* Désenregistrement de l'objet courant auprès du serveur */
               ……..
               System.exit(0);
            }
            /* Envoi d'un message au serveur */
            ……..
         }
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

ClientTchatInterface.java

```
import java.rmi.*;

interface ClientTchatInterface extends Remote{
   /* A compléter */
```

.....
}

IV. Java RMI. Client multithreadé

Voici un programme client/serveur avec java RMI dans lequel le serveur dispose d'une valeur de taux actualisée. Dont le client peut avoir besoin à certains moments.

1) CalculateurInterface.java

```
import java.rmi.*;

interface CalculateurInterface extends Remote{
   public float donneTauxActuel() throws RemoteException;
}
```

2) Calculateur.java

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

class Calculateur extends UnicastRemoteObject implements CalculateurInterface{
   float taux = 0.0F;
   public Calculateur () throws RemoteException {
      super();
   }

   public void majTauxActuel(float taux) throws RemoteException {
      this.taux=taux;
   }

   public float donneTauxActuel() throws RemoteException {
      return taux;
   }
}
```

3) SaisieEtMajTaux.java

```
import java.util.*;

class SaisieEtMajTaux extends Thread {
   private Calculateur calculateur;
   private String url;

   public SaisieEtMajTaux (String url, Calculateur calculateur){
      this.url=url;
      this.calculateur=calculateur;
   }

   public void run() {

      // Demande à l'utilisateur de saisir son message au clavier
      while(true){
         float taux=0.0F;
         Scanner keyboard = new Scanner(System.in);
         System.out.print ("Nouveau taux : ");
         String s=keyboard.nextLine();

         try{
            taux = Float.parseFloat(s);
         }
         catch (Exception e) {
            System.out.println ("Valeur entiere ou reelle SVP ");
         }
         try{
            calculateur.majTauxActuel(taux);
            // Naming.rebind(url, calculateur);
         }
```

```
            catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

4) ServeurTaux.java

```
import java.rmi.*;
import java.net.*;
import java.rmi.registry.*;

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

            LocateRegistry.createRegistry(1099);

            Calculateur calculateur = new Calculateur();
            String url="rmi://"+InetAddress.getLocalHost().getHostAddress()+"/calculateur";

            SaisieEtMajTaux saisieEtMajTaux=new SaisieEtMajTaux(url, calculateur) ;
            saisieEtMajTaux.start();

            Naming.rebind(url, calculateur);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5) Global.java

```
class Global {

    private float tauxActuel = 0.0F;
    private float tauxPrecedent = -1.0F;

    public float lectureTauxActuel(){
        return tauxActuel;
    }

    public float lectureTauxPrecedent(){
        return tauxPrecedent;
    }

    public float[] lectureTauxActuelEtPrecedent(){
        float tab[]= new float[2];
        tab[0] = tauxActuel;
        tab[1] = tauxPrecedent;
        return tab;
    }

    public void majTauxActuel (float nouveauTaux){
        if (nouveauTaux != tauxActuel){
            tauxPrecedent= tauxActuel;
            tauxActuel = nouveauTaux;

            System.out.println("Nouvelle mise a jour du taux : "+tauxActuel);
        }
    }
}
```

6) TraitementLocal.java

```
class TraitementLocal extends Thread {
```

```
   private Global global;
   private String nom ;
   private AccesDistant accesDistant;

   public TraitementLocal (Global global, String nom, AccesDistant accesDistant){
      this.global=global;
      this.nom=nom ;
      this.accesDistant= accesDistant;
   }

   public void run(){
      try {

            // Première partie du traitement n'ayant pas besoin du taux de reference (celle du site distant)
            System.out.println("Debut d'un traitement, je n'ai pas besoin du taux de reference");

            float taux =global.lectureTauxActuel();    // Récupération du taux

            // Deuxieme partie du traitement utilisant le taux récupéré sur le site distant
            System.out.println("A partir d'ici, je dois poursuivre avec le taux actualise,  taux : "+taux);
            System.out.println("Ca y est, j'ai fini mon traitement");

      }
      catch (Exception e){
         e.printStackTrace();
      }
   }
}
```

7) AccessDistant.java

```
import java.rmi.*;

class AccesDistant extends Thread {

   private Global global;
   private String nom;

   public AccesDistant(Global global, String nom){
      this.global=global;
      this.nom=nom ;
   }

   public void run(){
      try {

         CalculateurInterface calculateur = (CalculateurInterface)Naming.lookup("rmi://localhost:1099/calculateur");

         float tauxActuelDistant=calculateur.donneTauxActuel();

         global.majTauxActuel (tauxActuelDistant);

      }
      catch (Exception e){
         e.printStackTrace() ;
      }
   }
}
```

8) ClientTaux.java

```
public class ClientTaux{
   public static void main(String args[]){
      Global global=new Global();
      AccesDistant accesDistant = new AccesDistant (global, "Distant");
      TraitementLocal traitementLocal = new TraitementLocal (global, "Local", accesDistant);
      accesDistant.start();
      traitementLocal.start();
```

```
   }
}
```

Questions

1) Commenter le rôle des différentes classes.
2) Combien de processus sont lancés dans la partie serveur ? Que font-ils ?
3) Observer la classe Global et ses méthodes. Quel problème pourrait-on avoir en environnement multithreadé. Comment résoudre ce problème ?
4) Dans le thread **traitementLocal**, la deuxième partie du traitement doit utiliser le taux actualisé du serveur, lequel est récupéré par le thread **AccesDistant**. Comment peut-on faire cela simplement (en n'ajoutant qu'une ligne de code) ?
5) On souhaite maintenant que les deux threads client ne s'arrêtent pas : le thread **AccesDistant** doit vérifier toutes les 5 minutes si le taux distant a changé. Si c'est le cas, alors la deuxième partie de **traitementLocal** doit être débloquée (ce dernier recommencera ensuite un nouveau traitement -première partie- puis se remettra de nouveau en attente d'une mise à jour du taux distant pour reprendre sa deuxième partie etc. En d'autres terme **AccesDistant** doit "boucler" toutes les 5 minutes tandis que **traitementLocal** doit "boucler" autant de fois qu'il y a de mises à jour sur le site distant.

V. RPC en C

Programmation RPC couche haute. Compléter le programme client/serveur suivant **aux endroits indiqués en gras** (le serveur propose une fonction appelable à distance qui calcule la somme de deux entiers).

serveur.c

```
#include <rpc/rpc.h>
#include <stdio.h>
#include <stdlib>
#include <string.h>
#include <strings.h>

#define ARITHM_PROG_NUM ((u_long)0x33333333)
#define ARITHM_VERS_NUM ((u_long)1)
#define ADD_FCT_NUM ((u_long)1)

/* Deux entiers pour le calcul */
struct operands{
    int op1;
    int op2;
};

/* Fonction d'/de … */
bool_t encoding_operands(XDR *x, struct operands *ops){
    /* Compléter ici */
    ……..
}

/* Fonction appelable à distance qui calcule la somme de 2 entiers */
int *add (struct operands *ops){
    static int result;
    /* Compléter ici */
    ……..
    ……..
}

main(int argc, char *argv[]){

   pmap_unset(ARITHM_PROG_NUM, ARITHM_VERS_NUM);

    /* Enregistrement de la fonction auprès du portmap */
    ……..

    if (r==-1){
```

```
        perror("Registering service\n");
        exit(1);
    }
    svc_run();
    exit(0);
}
```

client.c

```
#include <rpc/rpc.h>
#include <stdio.h>
#include <stdlib>
#include <string.h>
#include <strings.h>

#define ARITHM_PROG_NUM ((u_long)0x33333333)
#define ARITHM_VERS_NUM ((u_long)1)
#define ADD_FCT_NUM ((u_long)1)

/* Deux entiers pour le calcul */
struct operands{
    int op1;
    int op2;
};

/* Fonction d'/de… */
bool_t encoding_operands(XDR *x, struct operands *ops){
    /* Même chose que pour serveur.c */
    ……..
}

/* Fonction principale. argv[1] : hôte distant. argv[2] : première opérande. argv[3] : deuxième opérande. */

main(int argc, char *argv[]){

    if (argc<4){
        fprintf(stderr, "missing parameters\n")
        exit(1)
    }

    char *host=argv[1];

    struct operands ops;
    /* Récupération des deux entiers */
    ……..
    ……..

   /* Entier qui contiendra leur somme*/
    int result;

    /* Appel de la fonction distante */
    ……..

    if (r!=0){
        clnt_perrno(r);
        exit(1);
    }

    printf("La somme de %d et %d est : %d\n", ops.op1, ops.op2, result);
    exit(0);

}
```

VI. RPC en C

Programmation RPC couche haute. Compléter **aux endroits indiqués en gras** :
- le code de serveur.c, programme serveur qui propose une fonction appelable à distance qui convertit les caractères d'une chaîne en majuscule

- le code de client.c, programme client qui sollicite cette fonction distante (la chaîne à convertir est fournie en entrée de sa fonction main(...)).

Voici le prototypes d'une fonction utile :
- int toupper(int c); /* Conversion d'un caractère en majuscule. Le caractère est fourni et retourné comme un int */

serveur.c

```
#include <rpc/rpc.h>
#include <stdio.h>
#include <stdlib>
#include <string.h>
#include <strings.h>
#include <ctype.h>

#define STRING_PROG_NUM ((u_long)0x22222222)
#define STRING_VERS_NUM ((u_long)1)
#define STRING_FCT_NUM ((u_long)1)

/* Fonction de décodage du paramètre (d'entrée) et d'encodage du résultat de la fonction upper_string(...) */
bool_t encoding_string(XDR *x, char **m){
   /* Compléter ici */
   ……..
}

/* Fonction appelable à distance qui convertit une chaine en majuscule */
char **upper_string (char **m){
   /* Compléter ici */
   ……..
   ……..
   ……..
}

int main(int argc, char *argv[]){
   pmap_unset(STRING_PROG_NUM, STRING_VERS_NUM);

   /* Enregistrement de la fonction auprès du portmap */
   ……..

   if (r==-1){
      perror("Registering service\n");
      exit(1);
   }

   /* Lancement de l'écoute */
   ……..
   exit(0);
}
```

client.c

```
#include <rpc/rpc.h>
#include <stdio.h>
#include <stdlib>
#include <string.h>
#include <strings.h>

#define STRING_PROG_NUM ((u_long)0x22222222)
#define STRING_VERS_NUM ((u_long)1)
#define STRING_FCT_NUM ((u_long)1)

/* Fonction d'encodage de/du … et de décodage de/du … */
bool_t encoding_string(XDR *x, char **m){
   /* Même chose que pour serveur.c */
   ……..
}
```

```
/* Fonction principale. Premier argument : nom d'hôte distant. Deuxième argument : message à convertir */

int main(int argc, char *argv[]){

    char *host=argv[1]; /* pointeur sur le nom d'hôte */
    char *msg=argv[2]; /* pointeur sur le message à envoyer */
    char *msg_maj=(char *) malloc (100 * sizeof(char)); /* pointeur sur l'emplacement du message de réponse + réservation mémoire

    if (argc<3){
        fprintf(stderr, "missing parameters\n")
        exit(1)
    }

    /* Appel de la fonction distante */
    ……..

    if (r!=0){
        clnt_perrno(r);
        exit(1);
    }
    printf("La chaine en majuscule est : %s\n", msg_maj);
    exit(0);
```