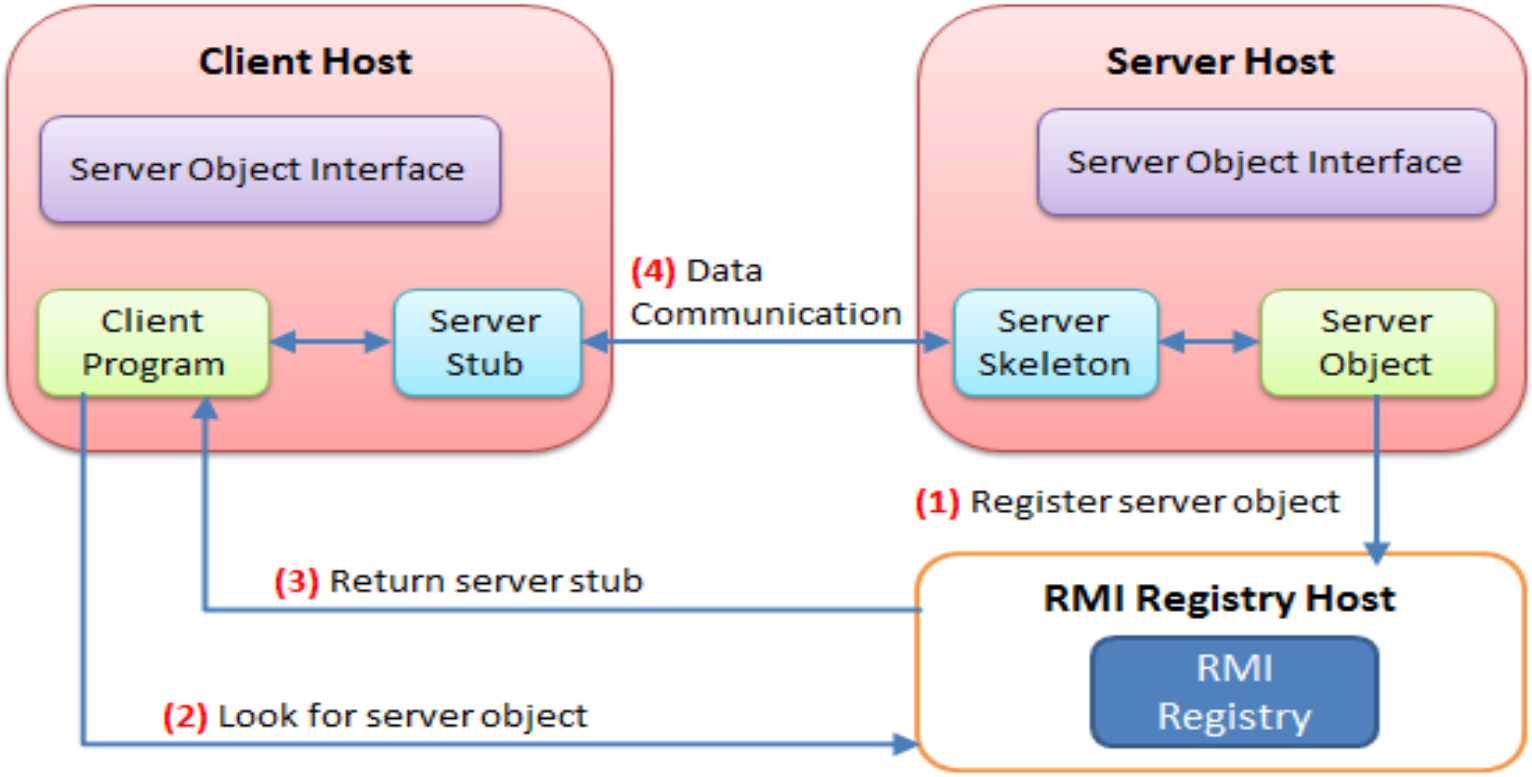
TP4
Appel de méthodes distantes (RMI)

Principe



- ✓Client - Utilise une méthode distante
- ✓Server - Processus qui possède l'objet appelé
- ✓Registry - Serveur de nom qui gère l'association nom-objet

Principe

Etapes nécessaires pour appeler un objet distant par Java RMI:

1. Instancier le serveur et l'enregistrer auprès du RMI Registry avec un nom unique

2. Le client récupère les information du serveur en utilisant le nom unique auprès du RMI Registery

3. Le RMI Registery en utilisant sa base envoie au client le stub qui permettra au client de communiquer avec le server

4. Le stub prends en charge la communication avec le skeleton qui est cote serveur. Les stub et skeleton se chargent de faire le marshalling et l'unmarshalling

Caractéristiques

Principe : Chaque classe d'objet serveur doit être associé à une interface Java

- Seules les méthodes de l'interface peuvent être invoquées

1. Écriture d'une interface
2. Écriture d'une class implantant l'interface
3. Écriture d'un programme serveur
4. Écriture du programme client

⇔

1. Déclaration des services accessibles à distance
2. Définition du code du service
3. Instanciation et enregistrement du serveur
4. Recherche et interaction du serveur

Construction d'une application RMI

1 - Définition du serveur = écriture d'une interface de service

- Interface Java normale
- Doit étendre java.rmi.Remote
- Toutes les méthodes doivent lever java.rmi.RemoteException

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

public interface Compte extends Remote {
  public String getTitulaire() throws RemoteException;
  public float getSolde() throws RemoteException;
}
```

Construction d'une application RMI

2 - Implantation du serveur = écrire une classe implantant l'interface

- Class Java normale
- Constructeurs doivent lever java.rmi.RemoteException;
- Si pas de constructeur, en déclarer un vide qui lève RemoteException

```
public class CompteImpl implements Compte {
  private String proprietaire;
  private double solde;

  public CompteImpl(String proprietaire) throws RemoteException {
    this.proprietaire = proprietaire; this.solde = 0; }

  public String getTitulaire() { return proprietaire; }
  public float getSolde(){ return solde; }
}
```

Construction d'une application RMI

3 - Écriture du serveur = instanciation + enregistrement

```
public static void main(String args[]) {
  Compte compte = new CompteImpl("Bob");  // création de l'objet serveur
  UnicastRemoteObject.exportObject(compte, 0);
  // enregistre compte dans RMI : on peut déjà l'appeler à distance

  // trouve le service de résolution de nom de RMI
  Registry registry = LocateRegistry.getRegistry(hostName );
  // enregistre la référence distante dans le service de résolution
  registry.bind("Bob", compte);
}
```

Le programme ne s'arrête pas tant que compte est enregistré dans RMI
Pour désenregistrer : UnicastRemoteObject.unexportObject(compte, false);
    false : attend la fin du traitement des requête, true : immédiat

Construction d'une application RMI

4 - Écriture du client : trouver compte + interagir avec lui

```
public class Client {
  public static void main(String[] args) {
    // trouve le service de résolution de nom de RMI qui se trouve sur hostName
    Registry registry = LocateRegistry.getRegistry(hostName);
    // demande un mandataire vers le compte de Bob
    Compte compte = (Compte)registry.lookup("Bob");
    // utilise le compte
    System.out.println("Bob possède " + compte.getSolde() + " euros");
  }
}
```

hostName est un nom de machine, celui sur lequel se trouve le serveur de résolution de noms

Services RMI

Service de résolution de noms

- Permet d'enregistrer un Remote sous un nom symbolique
  - Par défaut sur le port 1099
  - Noms "plats" (pas de hiérarchie)

- Deux manières de démarrer le service
  - De façon autonome dans un shell avec l'outil rmiregistry
  - Dans un programme avec static LocateRegistry.createRegistry(int port)

- Trouver le service de résolution de noms
  - static LocateRegistry(String host, int port) : à distance
  - static LocateRegistry(String host) : à distance sur le port 1099
  - static LocateRegistry(int port) : localement
  - static LocateRegistry() : localement sur le port 1099

Exemple

Hello.java

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

public interface Hello extends Remote {
   String sayHello() throws RemoteException;
}
```

Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {

    private Client() {}

    public static void main(String[] args) {

        String host = (args.length < 1) ? null : args[0];
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Hello stub = (Hello) registry.lookup("Hello");
            String response = stub.sayHello();
            System.out.println("response: " + response);
        } catch (Exception e) {
        System.err.println("Client exception: " + e.toString());
        e.printStackTrace();
    }
  }
}
```

Exemple

Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server implements Hello {

    public Server() {}

    public String sayHello() {
      return "Hello, world!";
    }

    public static void main(String args[]) {

        try {
            Server obj = new Server();
            Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);

            // Bind the remote object's stub in the registry
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Hello", stub);

            System.err.println("Server ready");
        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```