Etudier le fonctionnement utilisateur de rmi

I Fonctionnement de base

L'objectif de ce premier exercice est de faire un premier service distant réalisant une fonction simple comme le calcul de la suite de fibonacci. Le code d'exemple est le suivant :

```
package service ;
public class Fibo {

  public int calcul(int val){
    int val1=0,val2=1,val3=0;
    for (int i=0; i<val; i++){
     val3=val1+val2;
     val1=val2;
     val2=val3;
     System.out.println("val "+val3);
    }
    return val3;
  }

  public static void main (String [] arg) throws Exception{
    Fibo s=new Fibo();
    System.out.println("Calcul
"+s.calcul(Integer.parseInt(arg[0])));
  }
}
```

I.1 Réaliser cette classe, compiler tester.

*Remarque : fabriquez vous un environnement de travail « propre » contenant les répertoires suivants : src et classes

*Remarque : utilisez l'option –d de javac permettant de déposer le résultat dans un répertoire différent des sources.

*Remarque : la classe Fibo s'appelle en fait service.Fibo et doit donc se trouver dans ~src/service/Fibo.java et le résultat de la compilation doit se trouver dans ~/classes/service/Fibo.class

I.2 Réaliser l'interface, une implementation et un client

Modularité du code

Afin de rendre le code plus « modulaire », il est intéressant de le« refactorer » afin d'avoir une indépendance entre l'interface et l'implémentation d'un service.

Modifier votre code afin d'avoir :

- une interface de service **service.FiboIfc**,
- une classe d'implémentation **service.FiboImpl**,
- un client qui utilise le service **client.Client** (déplacer et modifier la méthode main dans cette classe).

Rappels:

- Une interface est constituée d'un ensemble de déclarations de méthodes sans implantation.

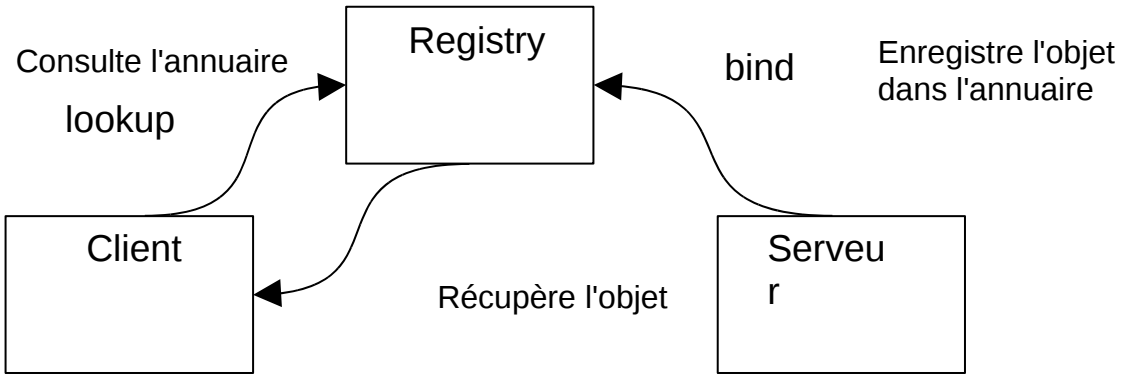
- Une classe peut implanter une ou plusieurs interfaces.
- L'instanciation d' une interface se fait comme suit :

```
interface ifc = new impl();
```

Vous devez alors avoir 3 classes de base : **service.FiboIfc** , **service.FiboImpl**, **client.Client**

Mécanisme RMI

- Un annuaire de nommage qui permet de rendre accessibles les objets.
- Un serveur qui fournit une fonctionalité
- Le serveur est enregistré dans un annuaire
- Un client recherche dans l'annuaire l'objet qui l'intéresse, le récupère et fait des appels sur ses méthodes.



I.3 Fabriquer le service distant

Conformité RMI

- Modifier l'interface de service afin de la rendre *compatible* rmi :
  - l'interface doit étendre l'interface **java.rmi.Remote**
  - chaque méthode doit pouvoir lever une **java.rmi.RemoteException.**

Préparer l'interface, compiler.

Rendre le serveur accesible

Afin de rendre la classe accesible sur un serveur, il faut l'exposer . Cette opération créé et lance le serveur de l'objet exposé sous la forme d'un thread. Cela peut se faire de deux manières :

- Le serveur étend la classe **java.rmi.server.UnicastRemoteObject,** dans ce cas l'exposition est automatique. D'autre part, il est nécessaire de rendre le serveur disponible aux clients (mise à disposition du stub de manipulation). Pour cela, il faut qu'une fonction du système fabrique l'instance du serveur et dépose le stub dans un annuaire.
- Le serveur n'étend pas la classe **java.rmi.server.UnicastRemoteObject** mais expose l'objet explicitement à travers la méthode statique exportObject(obj).

Implanter votre interface de service selon l'une des méthodes. Compiler.

Cette étape est réalisée par le code suivant :

```
public static void main(String [] arg) throws Exception{
```

```
//création du service distant
  service.FiboIfc srv=new service.FiboImpl();

//si le serveur n'étend pas  UnicastRemoteObject
FiboIfc stub = (Fibo) UnicastRemoteObject.exportObject(srv,
0);

}
```

Publier le serveur
La publication des objets mis à disposition aux clients est réalisée à travers un annuaire Registry, *cela consiste à* faire connaitre l'existence des différents objets à l'annuaire de la machine sur laquelle le serveur s'exécute. La classe java.rmi.Naming permet l'enregistrement (bind, rebind), le retrait (unbind), la récupération de l'objet (lookup) et la consultation (list).

```
//enregistrement du service dans l'annuaire
  java.rmi.Naming.bind(FiboIfc.class.getName(), srv);
```

Remarque : `FiboIfc`.class.getName(), permet de renvoyer une chaîne de caractères représentant le nom de la classe. On aurait pu choisir n'importe quelle autre chaîne de caractères.

Compiler – Executer… Que se passe t'il ? identifier clairement l'erreur concernée.

I.4 Fabrication du stub et du skeleton.
Pour fabriquer le stub/skeleton il faut utiliser le compilateur **rmic** qu'il faut appliquer sur la classe du serveur (oui la classe).
Remarque : n'oubliez pas de générer les stubs dans le répertoire classes (option –d)

Cette étape n'est plus nécessaire à partir de java 5.0.
I.5 Lancer l'annuaire
Lancement de l'annuaire
Aller dans le répertoire classes, puis lancer le programme rmiregistry. (C'est le service d'annuaire). Sur quel port tourne le service d'annuaire ?

Lancement du serveur
Lancer le serveur, normalement le stub de manipulation du service est déposé (bind, association nom/objet) sur l'annuaire).

I.6 Lancer le client
Le client s'adresse à l'annuaire dans lequel l'objet est enregistré afin de récupérer un talon de l'objet. Cette opération est réalisée en invoquant la méthode statique *lookup* de la classe *Naming***.** Elle prend en paramètre l'url de la machine distante et le nom avec lequel le serveur a été enregistré dans l'annuaire.

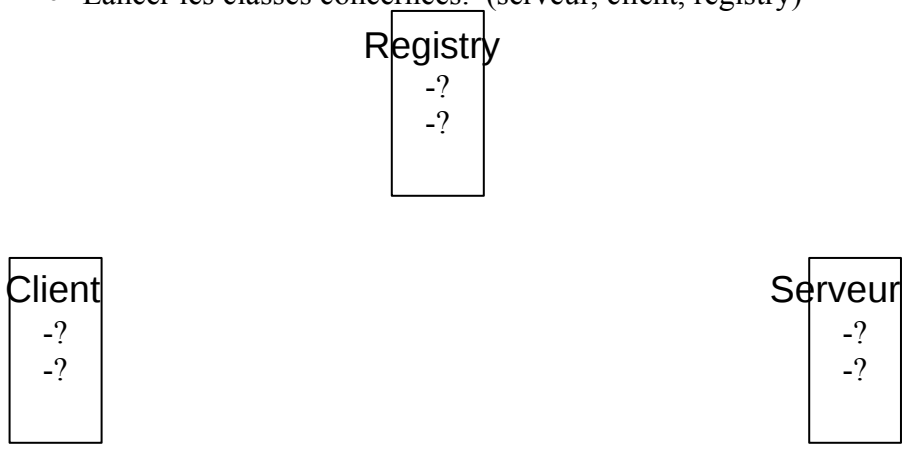
Dans le répertoire client modifier le code afin qu'il récupère le stub sur l'annuaire. L'appel devient :

```
HelloIfc hello=(HelloIfc)Naming.lookup(
     "rmi://ares-frenot-3/"+FiboIfc.class.getName());
```

Compiler exécuter.

II Séparation des classes

- Créer trois répertoires client / serveur / registry, dans lesquelles seules les classes compilées sont installées (classes du client, classes du serveur, l'annuaire ne dépend d'aucune source).
- Lancer les classes concernées. (serveur, client, registry)



Quelles sont les classes nécessaires aux différents éléments?
Valider vos résultats avec l'assistant du TP.

III Chargement dynamique des stubs

Il est possible de ne pas copier les classes des stubs sur le client. Celui-ci récupère les classes à partir d'une url. Pour cela il faut lancer la machine virtuelle du client avec un paramètre indiquant l'url à partir de laquelle il peut charger les classes qui lui manque.

```
java -Djava.rmi.server.codebase=http://serverhost/~username/rmi/ client.Client
ou
java - Djava.rmi.server.codebase=file:///home/sfrenot/tmp/service/classes
clietn.Client
```

IV Système distribué

Utiliser votre client pour utiliser le service provenant d'une machine d'un autre binôme.
Vérifier que votre code fonctionne sur les serveur s des autres groupes.

V Objet local et objet remote

Modifier votre code afin de récupérer un objet contenant toutes les valeurs intermédiaires du calcul de la suite de fibonacci. Tester les deux modes de fonctionnement : par sérialisation de l'objet contenant les valeurs de la suite et par fabrication d'un stub de manipulation distant.

VI Pour les tenaces

Modifiez le code afin que le client fournisse un objet au serveur distant. L'objet fournit par le client peu bien évidemment être local (donc sérialisé) ou distant (le client fabrique un stub de manipulation au serveur).
Et la boucle est bouclée…

V Remarques de fin

- Étudier les sources générées par le compilateur rmid (option –keepgenerated)
- Exploiter l'annuaire afin de déposer des instances qui ne sont pas liées à des Stubs (exemple carnet d'adresse)