TP 3
Introduction à RMI

3 heures

1 Objet distant

La première étape va consister à écrire une interface distante et une classe l'implémentant.

1. Écrivez une interface RMI distante nommée *IRemote*, ayant une méthode *echo* sans paramètres et sans valeur de retour ;
2. Implémentez cette interface dans une classe nommée *RemoteObject* et faites afficher un message dans la méthode redéfinie ;
3. Ajoutez une méthode *main* à votre classe afin de créer une instance et l'enregistrer dans un *rmiregistry*.

2 Le client

Pour écrire le client il est nécessaire de connaître que l'interface distante. Il ne devra donc y avoir dans votre code que des références à *IRemote* et jamais à *RemoteObject*.

1. Créez une classe *Client* contenant une méthode *main* qui récupère une représentation distante (stub) de l'objet distant dont la référence est passée en paramètre ;
2. Écrivez le code permettant d'appeler la méthode *echo*.

3 Exécution

Nous allons maintenant tester ce petit programme. Pour simplifier un peu, tout se fait sur une unique machine. Ouvrez trois terminaux différents pour effectuer les manipulations suivantes :

1. Démarrez un *rmiregistry*. Ce dernier peut être démarré en tant que service ou bien programmatiquement. Lancez le une fois en tant que service manuellement depuis un terminal puis une fois que vous avez compris comment faire, faites le de manière programmatique afin de gagner du temps ;
2. Démarrez votre objet distant ;
3. Démarrez votre client ;
4. Dans quel terminal s'affiche le message ? pourquoi ?

4 Étude de la communication

Les communications en RMI sont dites synchrones, c'est ce que nous allons mettre en évidence.

1. Dans la méthode *echo* de l'objet distant, mettez une instruction permettant d'attendre 5 secondes (`Thread.sleep(...)`) ;
2. Du côté du client, faites afficher un message avant et après l'appel de méthode *echo* ;
3. Exécutez, que constatez-vous ? Pourquoi ?

5 Paramètres

RMI permet, de manière transparente pour l'utilisateur, le passage de paramètres ainsi que le retour de résultats.

1. Ajoutez à l'objet distant une méthode distante prenant un entier (type primitif) en paramètre ;
2. Modifiez le client pour faire appel à cette méthode et testez ;
3. Créez une méthode distante qui retourne l'adresse IP de la machine où se trouve l'objet distant (`InetAddress.getLocalHost().getHostAddress()`) ;
4. Modifiez le client et testez.

6 Se connecter à d'autres objets distants

Si les interfaces distantes sont identiques, il est possible de se connecter à l'objet distant d'un autre étudiant(e). Créez un binôme et modifiez vos interfaces pour les avoir identiques. Attention, il est nécessaire d'avoir la même version de Java pour l'exécution.

1. Modifiez votre code pour avoir la même interface distante ;
2. Une des deux personnes doit lancer l'objet distant ;
3. La personne n'ayant pas démarrer l'objet distant doit démarrer son client pour qu'il se connecte à l'objet distant de l'autre personne.

7 Sérialisation Java

L'objectif de cet exercice est de comprendre le mécanisme de sérialisation Java.

7.1 Héritage

1. Créez une classe *Parent* non sérialisable et ne contenant qu'un constructeur sans paramètre ;
2. Créez une classe *Son*, sous-classe de *Parent*, sérialisable ;
3. Écrivez une méthode *main* dans la classe *Son* qui fabrique une instance de *Son*, et la copie en utilisant la sérialisation ;

4. Exécutez. Comprenez vous ce qui s'est passé ?
5. Dans le constructeur de *Parent*, ajoutez l'affichage d'une chaîne de caractère ;
6. Faites de même pour le constructeur de *Son* ;
7. Exécutez à nouveau. Que constatez-vous ? Pourquoi ?
8. Créez une classe *Son2*, non sérialisable, qui hérite de *Parent*. Créez lui un petit-fils *GrandSon*, sérializable, avec une méthode *main* similaire à celle de *Son* ;
9. Ajoutez aux classes qui ne les ont pas encore un constructeur vide affichant un message ;
10. Exécutez. Que constatez-vous ? Pourquoi ?
11. Ajoutez à la classe *Son2* un champs `protected int` ;
12. Dans le main de *GrandSon* et avant la sérialisation, fixez une valeur à ce champs ;
13. Dans le main de *GrandSon* et après la déserialisation, faites affichez la valeur de ce champs pour l'objet copié. Que constatez-vous ? expliquez.

7.2 Sérialisation personnalisée

Nous allons dans cet exercice travailler sur les classes écrites précédemment. Il est cependant conseillé de travailler sur une copie en suffixant le nom des classes copiées par *CustomSerialization*.

1. Redéfinissez dans la classe *SonCustomSerialization* les méthodes *readObject* et *writeObject* afin d'effectuer le traitement défaut et afficher une trace sur la sortie standard.

```
private void writeObject(ObjectOutputStream out)
    throws IOException;
private void readObject(ObjectInputStream in)
    throws IOException, ClassNotFoundException;
```

2. Exécutez et vérifiez le bon fonctionnement ;
3. Ajoutez les même méthodes à la classe *ParentCustomSerialization*. Exécutez. Que constatez-vous ? Pourquoi ?
4. Ajoutez à la classe *GrandSonCustomSerialization* un mécanisme de sérialisation permettant de sauvegarder les champs hérités des parents.