TD1 - Java RMI

Listings

Listing 1 - Hello.java

```java
package hai704i.td1;

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

public interface Hello extends Remote {
    /* METHODS */
    String sayHello() throws RemoteException;
    void printHello() throws RemoteException;
}
```

Listing 2 - HelloImpl.java

```java
package hai704i.td1;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject implements Hello {

    /* CONSTRUCTOR */
    public HelloImpl() throws RemoteException {

    }

    /* METHODS */
    @Override
    public String sayHello() throws RemoteException {
        return "Hello world!";
    }

    @Override
    public void printHello() throws RemoteException {
        System.out.println("The server prints: Hello world");
    }

}
```

Listing 3 - Server.java

```java
package hai704i.td1;

import java.rmi.AlreadyBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server {
    /* CONSTRUCTOR */
    public Server() {}

    /* METHODS */
    public static void main(String[] args) {
        try {
            HelloImpl obj = new HelloImpl();
            Registry registry = LocateRegistry.createRegistry(1099);

            if (registry == null)
                System.err.println("Registry not found");
            else {
                registry.bind("Hello", obj);
                System.err.println("Server is ready");
            }
        } catch (RemoteException | AlreadyBoundException e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Listing 4 - Client.java

```java
package hai704i.td1;

import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
    /* CONSTRUCTOR */
    private Client() {}

    /* METHODS */
```

```
    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("Server response: " + response);
            stub.printHello();
        } catch (RemoteException | NotBoundException e) {
            e.printStackTrace();
        }
    }
}
```

Questions

Question 1

Donner la procédure exacte pour lancer le serveur puis le client.

Réponse On se place dans le répertoire où est présent le projet `helloWorld`. On lance le registre (qui tourne donc dans une JVM tierce sur localhost:1099) :

```
rmiregistry &
```

On lance le serveur, qui récupère le registre via la méthode `java.rmi.registry.LocateRegistry#getRegistry host)` :

```
java helloWorld.Server
```

Enfin on lance le client en lui passant en paramètre le port sur lequel le registre écoute :

```
java helloWorld.Client 1099
```

Question 2

Que se passe t-il si on remplace la ligne 15 de `Server.java` par la ligne 14 (commentée) ?

Réponse Si on commente la ligne 15 et décommente la ligne 14, alors le registre sera lancé depuis le serveur et tourne donc dans la même machine virtuelle que lui. Dans ce cas là, on ne lancera plus le registre et il suffit de lancer le serveur et le client comme fait précédemment.

Question 3

Donnez les affichages chez le client et chez le serveur.

Réponse

SERVEUR

```
Server ready
The server prints : Hello, world !
```

CLIENT

```
response : Hello, World !
```

Question 4

Dans quelle JVM seront créés les objets de type `HelloImpl` ?

Réponse Dans la JVM du serveur.

Question 5

A quoi sert l'interface `Hello.java` ?

Réponse

- à typer, côté client, le proxy reçu à la suite du lookup → mise en oeuvre du patron de conception `Proxy`.
- à spécifier les méthodes accessibles à distance pour les classes implémentant cette interface (*ici `HelloImpl.java`*).

Question 6

Donnez des exemples d'exceptions pouvant être attrapées à la ligne 22 de `Server.java`.

Réponse

1. A cause de `java.rmi.registry.LocateRegistry.getRegistry(String host)`:
   - `java.rmi.RemoteException` - if the reference could not be created.
2. A cause de `java.rmi.Naming#bind(String name, Remote od)`:
   - `java.rmi.AlreadyBoundException` - if name is already bound.
   - `java.rmi.RemoteException` - if remote communication with the registry failed; if exception is a `java.rmi.ServerException` containing an `java.rmi.AccessException`, then the registry denies the caller access to perform this operation (*e.g., if originating from a non-local host*).
   - `java.rmi.AccessException` - if this registry is local and it denies the caller access to perform this operation.
   - `java.lang.NullPointerException` - if `name` is `null`, or if `od` is `null`.

Question 7

Quelle est la différence entre `Naming.bind` et `Naming.rebind` ?

Réponse `java.rmi.Naming#bind(String name, Remote od)` lance une exception `java.rmi.AlreadyBoundException` si `name` est déjà utilisé dans le registre, alors que `java.rmi.Naming#rebind(String name, Remote newOd)` remplace dans ce cas l'ancien objet associé à `name` par `newOd`.

Question 8

Que se passe t-il si on ne passe pas d'argument en ligne de commande quand on lance le client ?

Réponse Cela fonctionne quand même, étant donné que la méthode `java.rmi.registry.LocateRegistry.getRegistry(String host)` retourne le registre avec les paramètres par défaut (*localhost, 1099*) quand host est `null`.

Question 9

Faites un diagramme d'instances pour le côté serveur et un pour le côté client illustrant les objets présents au démarrage du serveur et du client. Faites un autre diagramme d'objets illustrant le côté client après la ligne 15 du client.

Réponse