TP RMI

Pour donner un aperçu du TP sur RMI, il est nécessaire d'illustrer ce qu'est RMI en répondant à trois questions : **Quoi ? Pourquoi ? et Comment ?**

1. Quoi ? Qu'est-ce que RMI ?

RMI (Remote Method Invocation) est un mécanisme en Java qui permet à un objet qui s'exécute dans une machine virtuelle Java (JVM) de s'invoquer à distance sur un autre objet se trouvant dans une autre machine. En d'autres termes, RMI permet la communication entre des objets distants en invoquant des méthodes sur ces objets comme.s'ils étaient locaux.

En résumé, RMI est une technologie Java qui permet à un programme (le client) d'invoquer des méthodes d'un autre programme exécuté sur une machine distante (le serveur), comme si elles étaient locales.

2. Pourquoi ? Pourquoi utiliser RMI ?

RMI est utilisé pour simplifier le développement d'applications distribuées en permettant aux développeurs d'appeler des méthodes sur des objets distants de manière transparente, comme s'ils étaient locaux. Cela est particulièrement utile dans des architectures client-serveur, où les systèmes doivent interagir sans que les détails sous-jacents de la communication réseau soient exposés au programmeur.

3. Comment ? Comment fonctionne RMI ?

L'architecture RMI est basée sur trois composants principaux :

- **Stub (client) :** Un stub est un objet client local qui représente l'objet distant (L'ambassadeur). Lorsqu'un client appelle une méthode sur le stub, celui-ci ne réalise pas directement l'action. À la place, il transforme cet appel en une requête réseau contenant toutes les informations nécessaires pour invoquer cette méthode sur l'objet distant situé sur le serveur. Ce mécanisme est transparent pour le client, qui a l'impression d'interagir avec un objet local, alors qu'il communique en réalité avec un objet situé sur un autre machine.

- **Skeleton (serveur) :** Le skeleton est une entité qui réside du côté serveur. Il reçoit les requêtes du stub et les traduit pour qu'elles puissent être exécutées par l'objet distant réel. Le skeleton, en quelque sorte, interprète la requête réseau et la transforme en un appel de méthode sur l'objet réel (qui réside sur le serveur). Ensuite, une fois la méthode exécutée, le skeleton renvoie le résultat au client.

- **Objet distant (serveur)** : L'objet distant est celui qui réside sur le serveur et qui réalise les opérations demandées par le client. Cet objet peut exécuter des méthodes et renvoyer des résultats à un client, même si le client et le serveur ne sont pas sur le même réseau local.

Afin de comprendre ce mécanisme, il est nécessaire de détailler chacune des étapes pour clarifier davantage chaque notion, afin que tout soit bien clair.

1. Création de l'interface distante

L'interface distante définit les méthodes que le client pourra appeler à distance. C'est une sorte de contrat entre le client et le serveur, qui stipule quelles opérations peuvent être effectuées sur l'objet distant. Cette interface doit hériter de **l'interface Remote**, et chaque méthode doit lancer une **RemoteException.**

2. Implémentation de l'objet distant

L'objet distant doit donc implémenter toutes les méthodes de l'interface, et c'est cet objet qui effectuera le véritable travail lorsqu'une requête est reçue du client. Cet objet est instancié sur le serveur et s'occupera d'exécuter la méthode lorsque le client l'appellera à distance. Pour que cet objet soit accessible à distance, il doit hériter de **UnicastRemoteObject.**

3. Enregistrement de l'objet distant dans le registre RMI

Une fois que l'objet distant est créé sur le serveur, il doit être enregistré dans **le registre RMI**. Ce registre agit comme un **annuaire** où les objets distants sont publiés sous un nom unique. Grâce à cet enregistrement, un client pourra retrouver l'objet distant en interrogeant le registre par ce nom.

Le serveur publie l'objet en utilisant le nom par exemple « Animal » et l'enregistre dans le registre RMI. Cela permet au client de le localiser et de l'utiliser comme s'il s'agissait d'un objet local.

4. Le client utilise l'objet distant

Le client va se connecter au registre RMI pour chercher l'objet distant à l'aide de son nom « Animal ». Une fois l'objet localisé, le client reçoit une référence sous forme de stub, et il peut ensuite utiliser cet objet comme s'il était local, en appelant des méthodes dessus.

Le stub va transférer l'appel de méthode à l'objet distant sur le serveur. Le traitement est effectué sur le serveur et le résultat est renvoyé au client.

Maintenant, après avoir donné un survol concis sur RMI et sa vocation, il est primordial de présenter une description concise du parcours effectué afin d'arriver au résultat requis et de lancer le serveur objet de notre TP.

1. Conception de la solution

- **a - Contexte et objectif du TP**

Le projet consiste à mettre en place un système distribué pour un cabinet vétérinaire en utilisant Java RMI (Remote Method Invocation). Le but est de permettre à un client distant d'interagir avec le cabinet, notamment pour :

- Consulter la liste des animaux présents dans le cabinet.
- Ajouter de nouveaux patients (animaux) au cabinet.
- Accéder aux dossiers de suivi des animaux.
- Recevoir des alertes lorsque des seuils spécifiques sont atteints (100, 500 et 1000 animaux).

Le client représente une application externe qui souhaite accéder aux données sur les animaux, tandis que le serveur héberge le cabinet vétérinaire avec toutes les informations pertinentes sur les animaux.

- **b - Architecture du système**

Le système RMI repose sur quatre composants essentiels :

1. Interface RMI (ICabinetVeterinaire) : Définie pour permettre au client d'accéder à des méthodes distantes comme ajouterAnimal, getListeAnimaux, et rechercherAnimalParNom.

2. **Évolution de l'architecture** :

- Au début du projet, l'accent a été mis sur l'interface **IAnimal** et son implémentation **AnimalImpl**, qui permettent de gérer les informations de base sur chaque animal (nom, espèce, race, et dossier de suivi).

- Avec la progression du projet, nous avons réalisé que l'objet distant principal ne serait pas l'animal en lui-même, mais plutôt une entité plus large : **le cabinet vétérinaire**. Ainsi, l'interface **ICabinetVeterinaire** a été introduite pour encapsuler toute la gestion des animaux, et l'implémentation **CabinetVeterinaireImpl** est devenue l'objet distant central géré par le serveur. Cet objet contient la liste des animaux, leurs espèces, leurs races, ainsi que leurs dossiers de suivi.

3. Implémentation de l'interface (CabinetVeterinaireImpl) : Cet objet distant héberge les informations du cabinet, y compris la liste des animaux et leurs dossiers de suivi. Il est enregistré dans le registre RMI pour que le client puisse y accéder à distance.

4. Client : Le client invoque les méthodes distantes.de l'objet CabinetVeterinaireImpl via l'interface, pour ajouter un animal ou consulter un dossier de suivi.

5. Serveur : Il enregistre l'objet distant dans le registre RMI et exécute les opérations à distance demandées par le client.

Il y a une interaction entre le serveur et le client. Le serveur distribue l'objet ICabinetVeterinaire et permet au client de se connecter et d'interagir avec les animaux (les patients). Le client peut également ajouter de nouveaux animaux dans le cabinet via la méthode Scanner, à condition de ne pas dépasser le seuil.

- **c - Récapitulatif**

1. Création de l'interface :

Initialement, l'interface IAnimal a été créée pour définir les méthodes de base liées à un animal. Cependant, avec l'évolution du projet, l'interface ICabinetVeterinaire est devenue plus pertinente pour permettre la gestion complète des animaux dans le cabinet. Elle définit les méthodes que le client peut appeler à distance.

2. Implémentation de l'interface :

L'implémentation initiale de l'interface IAnimal a permis de gérer les informations liées à chaque animal individuellement via l'objet AnimalImpl. Par la suite, l'interface ICabinetVeterinaire a été mise en œuvre via l'objet distant CabinetVeterinaireImpl. Cet objet contient la logique pour ajouter des patients, gérer les espèces et dossiers de suivi, et consulter la liste des animaux.

3. Enregistrement de l'objet distant dans le registre RMI :

Le serveur enregistre l'objet distant CabinetVeterinaireImpl dans le registre RMI pour permettre au client de le localiser et de l'utiliser. Cet enregistrement se fait via la méthode bind ou rebind, selon les besoins.

4. Utilisation de l'objet distant par le client :

Le client utilise maintenant l'interface ICabinetVeterinaire pour interagir avec l'objet distant. Il peut par exemple :

- Ajouter un animal via la méthode ajouterAnimal
- Consulter la liste des animaux via getListeAnimaux
- Rechercher des informations spécifiques sur un animal via rechercherAnimalParNom

2. Partie ReadMe : Lancer/exécuter votre TP

Pour lancer l'application :

Il faut d'abord démarrer le server. Il s'agit d'ouvrir la classe **Server.java** dans Eslipce IDE, puis de l'exécuter en cliquant sur l'icône Run. Cela enregistra l'objet CabinetVeterinaire dans le registre RMI.

Deuxièmement, il faut ouvrir la classe **Client.java**, et l'exécuter via Run. Cela permettra au client de récupérer des informations sur les animaux et de les manipuler via le serveur distant.

L'Ajout d'un nouvel animal : Apres l'exécution du client, une phrase s'affichera dans la console pour demander l'entrée des informations d'un nouvel patient. Dès la saisie de ces informations, un nouvel animal sera ajouté au cabinet vétérinaire.

En suivant l'exemple du TD Hello.java, on peut déduire que la procédure se résume comme suit :

1. L'Ouverture du registre RMI

Ouverture d'un terminal et positionnement dans le répertoire où se trouve le projet Animal. Lancement du registre RMI sur le port 1099 avec la commande suivante : **rmiregistry 1099 &**

2. Lancement du serveur

Lancement du serveur, qui enregistrera l'objet CabinetVeterinaire dans le registre :**java animal.Server**

3. Lancement du client

Lancement du client pour interagir avec le serveur : **java animal.Client 1099**

Un test réalisé :

Apres avoir testé l'application, nous avons pu :

- Ajouter et modifier plusieurs animaux via le client.
- Tester la recherche d'animaux dans le cabinet par leur nom.
- Vérifier la mise à jour des listes d'animaux après l'ajout de nouveaux patients.