HAI704I
Architectures Logicielles Distribuées
TP1 - Compte Rendu

Mohamad Satea ALMALLOUHI - 22011503

Gurgun DAYIOGLU - 22024990

README

Pour lancer le TP, il suffit d'utiliser Eclipse et lancer le fichier "MainServer.java" et puis "MainClient.java".

Video explicative : https://youtu.be/NCvNJSL2X6M?si=CILN_UvYqaYeyPBl

Introduction

RMI (Remote Method Invocation) est un mécanisme de communication entre objets distants en Java. Il permet à un objet Java de faire appel à des méthodes d'un objet situé sur une autre machine virtuelle Java, généralement sur un autre ordinateur, via un réseau. Voici comment cela fonctionne de manière conceptuelle :

1) **Interface Distante :**

Tout commence par la définition d'une interface Java qui déclare les méthodes que le serveur met à disposition pour les clients. Cette interface doit étendre java.rmi.Remote et chaque méthode doit lancer java.rmi.RemoteException en cas d'erreur.

2) **Implémentation de l'Interface Distante :**

Une classe serveur implémente cette interface. Cette classe contient la logique réelle des méthodes distantes.

3) **Enregistrement du Serveur :**

Le serveur crée une instance de la classe d'implémentation, puis l'enregistre dans le registre RMI local en utilisant un nom symbolique. Le registre RMI est essentiellement un service de noms qui permet aux clients de rechercher des objets distants.

4) **Client RMI :**

Le client recherche l'objet distant dans le registre RMI en utilisant le nom symbolique. Une fois qu'il obtient l'objet distant, il peut appeler des méthodes sur cet objet comme s'il s'agissait d'un objet local. Cependant, ces appels de méthode traversent le réseau et sont exécutés sur l'objet distant.

5) **Sérialisation :**

L'objet que vous passez entre les machines doit être sérialisable, ce qui signifie qu'il doit être capable d'être converti en un flux d'octets pour être transmis sur le réseau. Cela signifie généralement que les classes doivent implémenter l'interface java.io.Serializable.

6) **Communication Réseau :**

Lorsque le client invoque une méthode sur l'objet distant, RMI prend en charge tout le travail réseau sous-jacent, y compris la sérialisation des paramètres et le transfert des données sur le réseau. De même, le résultat est transféré du serveur au client.

7) **Gestion des Exceptions :**

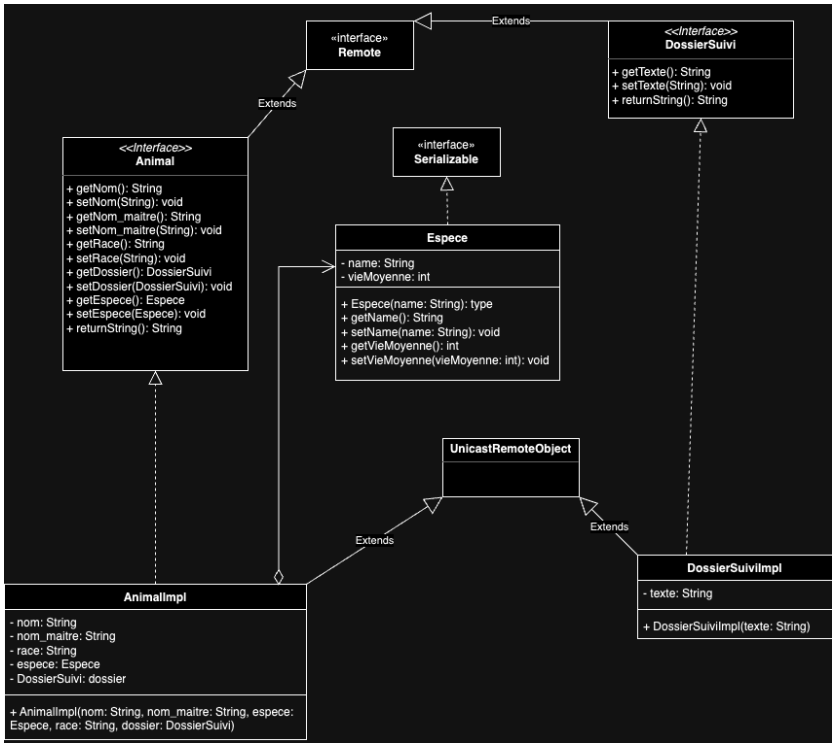
Étant donné que les appels de méthode sont effectués à travers le réseau, des exceptions peuvent se produire. C'est pourquoi toutes les méthodes distantes doivent déclarer java.rmi.RemoteException afin que les erreurs puissent être correctement propagées au client.

En résumé, RMI en Java permet à des objets Java de communiquer à travers un réseau comme s'ils étaient des objets locaux, simplifiant ainsi le développement d'applications distribuées en Java. Le mécanisme sous-jacent gère la transmission des données, la sérialisation des objets et la gestion des exceptions pour vous, ce qui rend la communication entre objets distants transparente pour les développeurs.

Exercice 1 - Une première version simple

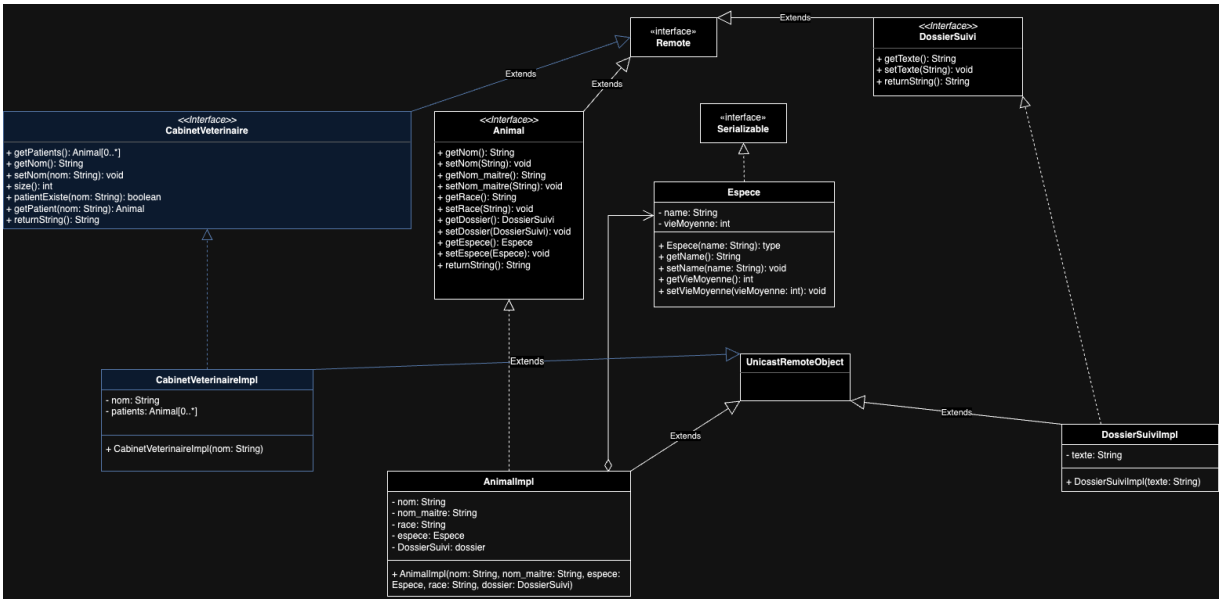
1) Comme la classe "Animal" sera distribuée, on crée une interface commune qui décrit les méthodes auxquelles le client peut faire appel et que le serveur doit implémenter. On ajoute aussi à chaque méthode "throws RemoteException" qui indique qu'on peut avoir des erreurs liées à la connexion. Plus précisément, cette exception permet de capturer les différentes erreurs pouvant survenir lors de l'exécution dans un environnement distribué. Au côté serveur, la classe d'implémentation d'Animal étend "UnicastRemoteObject" et on implémente les méthodes qui ont précédemment été définies.
2) Comme le gestionnaire de sécurité n'est plus utilisé, on a décidé de ne pas l'implémenter.
3) On crée l'interface commune "DossierSuivi" (implémentée toujours au côté serveur) qui va déclarer des méthodes utilisables par le client - "getTexte", "setTexte" et "toString".
4) On crée la classe commune "Espèce" et pour que le client puisse récupérer juste une copie de ses instances mais pas des références distantes, elle implémente "Serializable".

Voici le diagramme UML correspondant à l'exercice 1 :



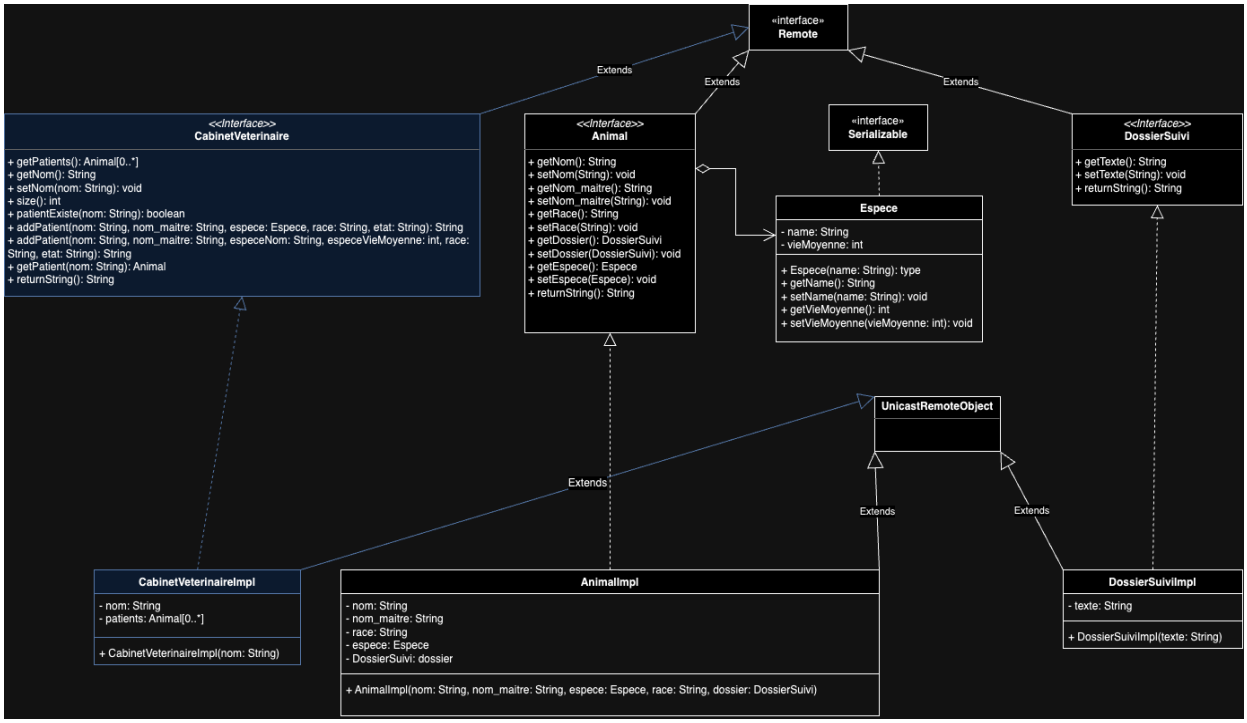
Exercice 2 - Classe cabinet vétérinaire

1) On crée l'interface commune "CabinetVeterinaire" et on l'implémente dans le côté serveur. On ajoute ensuite la méthode "getPatient" qui permet de chercher un animal par son nom.



Exercice 3 - Création de patient

1) On ajoute dans l'interface "CabinetVeterinaire" la méthode "addPatient" qui est surchargée.



Exercice 4 - CodeBase

Dans ce scénario, le client (le vétérinaire) crée une classe “Chien” qui hérite de l’interface “Animal” (dans le package common). Cependant, lors de l'exécution, le serveur rencontre une confusion car il n'a pas d'accès direct à la classe “Chien”. C'est ici que le concept de "codebase" devient crucial. La codebase est un mécanisme qui permet d'accéder à des classes distantes dans un réseau.

Le serveur a besoin d'accéder à la classe Chien pour comprendre l'objet que le client a créé. Pour résoudre ce problème, une propriété est ajoutée au serveur, appelée codeBase, qui lui permet d'accéder à un fichier nommé CodeBase. À l'intérieur de ce fichier, il y a une copie de la classe Chien en bytecode.

Grâce à ce fichier bytecode (représentation binaire du code source), le serveur est en mesure de reconnaître et comprendre l'objet créé par le client (Chien). Cela permet au serveur de travailler avec cet objet sans avoir directement accès à la classe Chien, en utilisant la version stockée dans le fichier CodeBase. Ce mécanisme facilite la communication entre le client et le serveur, en permettant au serveur d'accéder aux classes nécessaires même s'il ne les a pas en interne.

Exercice 5 - Alertes

1) Pour pouvoir implémenter des Alertes, on sépare le fichier "Client.java" en créant une interface commune Client et la classe ClientImpl dans le dossier "client". De plus, dans la classe "CabinetVeterinaireImpl", on ajoute la méthode "alertClients" qui sera appelée par la méthode "addPatient" si le nombre d'animaux dépasse 100, 500 ou 1000.

Voici la version finale du diagramme UML, vous pouvez aussi trouver cette image dans le fichier ZIP :