HAI704I - TP1 RMI

Compte rendu

BOURRET Maxime

Le code de ce TP est accessible via un répertoire distant hébergé sur GitHub. J'ai fait le choix de tout écrire en anglais pour m'habituer à mon futur métier où la plupart des entreprises adoptent la convention de développer un code universel.

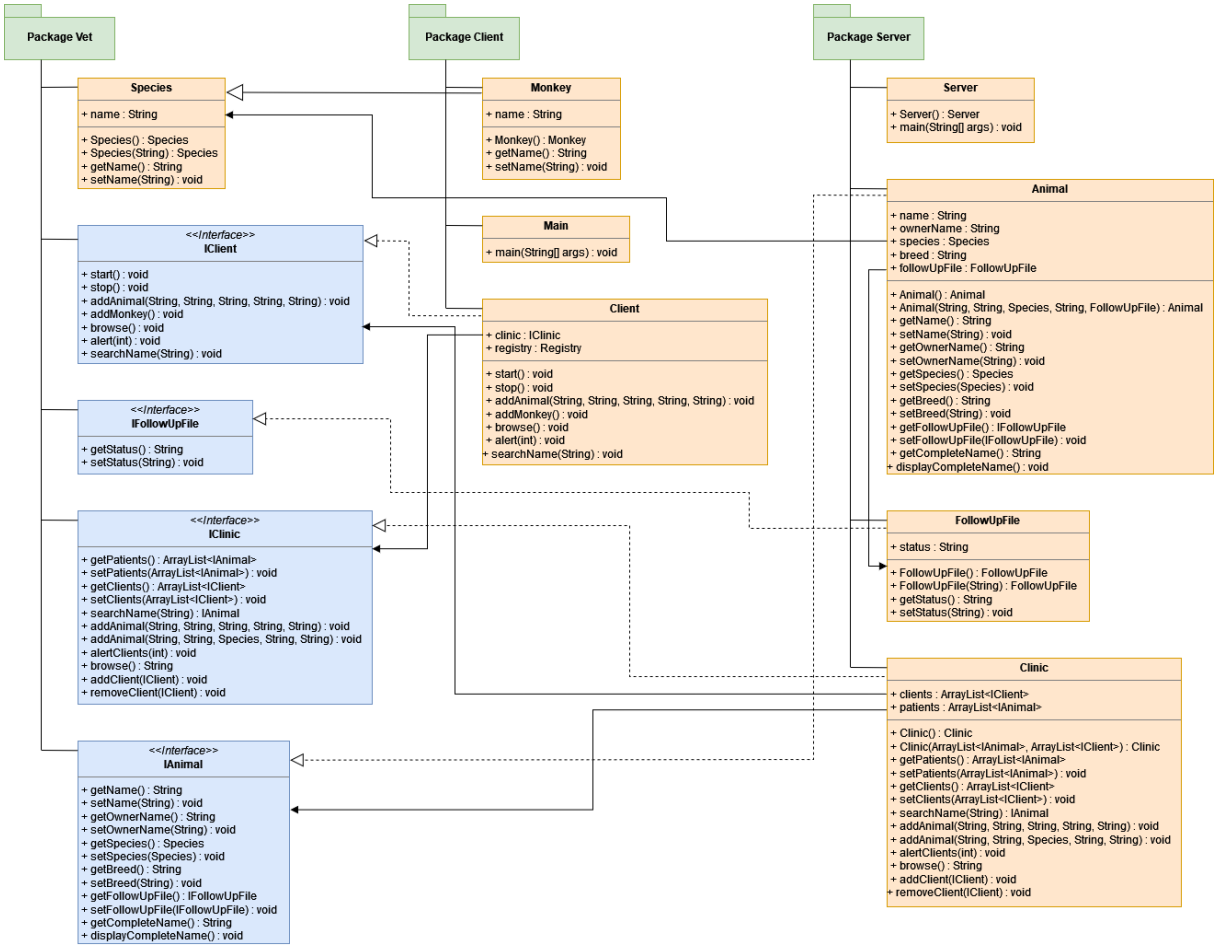

FIGURE 1 – Diagramme UML du projet Java

1 Une première version simple

1.1 Question 1

Tout d'abord, j'ai créé un projet Eclipse comportant 3 dossiers :

- Client, qui comporte tous les éléments nécessaires à l'utilisation d'un client
- Server, qui contient le programme de serveur et les implémentations des classes sauf celle du client
- Vet, qui correspond au package commun au client et serveur avec les interfaces des classes

J'ai ajouté la classe `Animal.java` et l'interface `IAnimal.java`. Il avait des attributs simples comme précisé dans l'énoncé à savoir un nom, un nom de propriétaire, un nom d'espèce et un nom de race ; tous ces attributs étant des `String`. J'ai également ajouté deux méthodes en plus des constructeurs, getters et des setters, `String getCompleteName()` et `void displayCompleteName()`. La première renvoie une `String` contenant le nom

complet de l'animal, la deuxième affichant ce dernier dans la console directement. Par la suite j'ai créé la classe `Server.java` où j'ai déclaré tous les éléments nécessaires à la mise en place du serveur :

- la déclaration du nouveau registre
- la création d'un animal
- l'enregistrement de l'animal dans le registre

Enfin j'ai créé la classe `Client.java` qui se connectait au serveur et récupérait le stub de l'animal enregistré dans le registry. Une fois cela fait, il appelait la méthode `String getCompleteName()`, l'affichait dans le terminal client et appelait la méthode `void displayCompleteName()` qui affichait le nom complet dans le terminal serveur.

1.2 Question 2

J'ai rajouté le fichier `security.policy` à la racine du projet et j'ai complété ma classe serveur pour qu'elle puisse intégrer les paramètres de sécurité passés dans le nouveau fichier (ici toutes les permissions ont été données).

1.3 Question 3

Pour ajouter un dossier de suivi, j'ai créé la classe `FollowUpFile.java` et l'interface `IFollowUpFile.java`. Au début, j'ai implémenté un dossier de suivi avec un attribut de statut qui est une `String` et un attribut de type `Animal`. Par la suite, j'ai modifié mon code pour que le dossier de suivi devienne un attribut d'Animal et non pas l'inverse. Enfin, j'ai édité le client afin qu'il change le statut du dossier.

1.4 Question 4

Afin de changer l'implémentation de l'espèce, j'ai créé la classe `Species.java` et l'interface `ISpecies.java`. Elle contient un attribut de type `String` pour le nom de l'espèce. Ensuite, j'ai modifié l'attribut de la classe animal afin que son type devienne `Species`.

2 Classe Cabinet Véterinaire

Pour créer le cabinet véterinaire, j'ai d'abord créé la classe `Clinic.java` et l'interface `IClinic.java` qui avait pour le moment un seul attribut qui est un tableau de patients. J'avais commencé avec le type `IAnimal[]` mais j'ai au final modifié l'implémentation pour avoir un type `ArrayList<IAnimal>`. Le cabinet était ensuite rempli avec des animaux et distribué par le serveur. La méthode `IAnimal searchName(String name)` lance une recherche séquentielle dans le tableau d'animaux sur l'attribut nom, la première occurence est retournée sinon un animal vide. Le client de son côté avait juste à exécuter la méthode, le reste de l'implémentation ne changeait pas.

3 Création du patient

Afin d'arriver à ajouter un patient depuis le client, j'ai du créer la méthode `public void addAnimal(String newName, String newOwnerName, String newSpecies, String newBreed, String Status)`. Elle prend en paramètres cinq `String` qui vont servir à créer le nouvel animal. Pour les attributs de ce dernier étant déjà du même type, l'affectation est directe mais pour l'espèce et le dossier médical, les chaînes de caractères correspondent au nom de l'espèce et au statut du dossier médical. Le constructeur paramétré des deux classes est donc appelé et l'animal est ajouté au tableau du cabinet. J'ai par la suite rajouté un constructeur similaire qui prend en paramètres un objet de type espèce et un de type dossier de suivi afin de réaliser la question suivante.

4 Téléchargement de code

Dans l'énoncé il nous était maintenant demandé de créer une erreur `ClassNotFoundException` en créant une sous-classe d'espèce côté client que le serveur ne connaissais pas et lors de l'ajout d'un animal de cette espèce, le serveur renverrait la dite erreur. Pour commencer, j'ai créé la classe `Monkey.java` qui extends `Species.java`. Côté client, j'ai rajouté l'option qui, une fois saisie, crée un animal d'espèce singe et l'envoie au serveur. Ce dernier rend alors l'erreur `ClassNotFoundException`. Par la suite, il nous a fallu trouver une solution pour indiquer au serveur où trouver la classe. En rajoutant la ligne `System.setProperty("java.rmi.server.codebase", "file:./Client/obj/");` dans l'initialisation du serveur, je lui indique que le codebase se trouve dans le dossier des `.class` du client. Ainsi, lorsque l'option d'ajout de singe est saisie, le serveur ajoute ce dernier aux animaux sans problème.

5 Alertes

Pour ce qui est des alertes, j'ai commencé à réfléchir à une solution qui signale lorsqu'on franchit un palier dans les deux sens. Mais après réflexion, le projet n'ayant pas de méthode pour supprimer des animaux du cabinet, seul le dépassement du palier était nécessaire. Côté serveur, il y a donc une vérification après chaque ajout qui compare la taille du tableau des animaux avec les paliers. Si l'un d'eux est franchi, on appelle la méthode `void alert(int level)` de chaque client connecté. Cette dernière affiche le palier franchi dans le terminal client.