Rapport TP1 Programmation orientée objet avancée - 8INF957

Alan Guivarch GUIA24119800

1.Application Serveur

Pour envoyer nos commandes j'utilise RMI, j'ai donc défini une Interface ServeurInterface.java qui extends Remote, la seule fonction dans cette interface est **traiteCommande**. C'est la seule fonction quie **ApplicationClient** aura besoin pour faire traiter ses commandes au serveur.
La Classe **ApplicationServeur** implémente Serveur Interface pour pouvoir implementer cette fonction.
Cette classe possède une **HashMap<String,Object>** pour stocker les objets créés par les commandes "création" envoyées par le client, elle possède également une **HashMap<String,Class<?>>** pour stocker les classes chargées par le client.

repSource, **repClasse** et **sortieServeur** stock respectivement le répertoire des fichiers sources, le répertoire des fichiers classes et le chemin vers le fichier où seront enregistrés les commandes traitées et leurs résultats.
resultObj permet de stocker si il y a besoin le résultat de la commande du client.

La fonction **traiteCommande** peut faire appel à six autres fonctions. Chacune de ces fonctions permet de traiter une type de commande envoyé par le serveur (**traiterLecture**, **traiterEcriture**, **traiterCreation**,
traiterChargement, **traiterCompilation** et **traiterAppel**) .

Les arguments nécessaire pour lancer le serveur sont :
1) numéro de port
2) répertoire source
3) répertoire classes
4) nom du fichier de traces (sortie)

2.Application Client

L'application client possède deux attribut :

- BufferedReader **commandesReader**;
- BufferedWriter **sortieWriter**;

commandesReader permet de lire le fichier de commandes passé en argument, et **sortieWriter** permet d'écrire le résultat des commandes traitées par le serveur dans le fichier de sortie passé en argument.
Pour utiliser RMI depuis le client, on initialise "stub" une **SeveurInterface** pour pouvoir appeler la fonction **traiterCmmande**.
Après l'initialisation des variables, on lance le scénario qui exécute le fichier de commande passé en paramètre.

Les arguments nécessaire pour lancer le client sont :
1) "hostname" du serveur
2) numéro de port,
3) nom fichier commandes
4) nom fichier sortie

3.Commande

La classe commande permet de décomposer les commandes du client pour les envoyer au serveur.

typeMessage type permet de stocker le type de la commande parmis les six types suivants(stocké dans l'enum **typeMessage**): compilation, chargement, création, lecture, écriture, fonction

ArrayList<String> pathSource est une liste contenant les chemins relatifs des fichiers sources pour la compilation.

String pathClasses est le chemin relatif vers les fichiers class.

String className stock le nom qualifié d'une classe (utilisé pour le chargement de classe)

String Id stock l'identifiant d'un objet que le serveur va devoir créer, cette identifiant permettra au serveur de reconnaître l'objet lorsqu'il faudra l'utiliser

String attributeName stock le nom de l'attribut qui doit être lu ou modifié

String value stock la nouvelle valeur de l'attribut qui doit être modifié

String fucntionName stock le nom de la fonction qui doit être exécuté par le serveur

ArrayList<String> parameterType stock les types des paramètres pour exécuter une fonction

ArrayList<String> parameterValue stock les valeurs des paramètres pour exécuter une fonction

Boolean result sa valeur est passée à true si la fonction c'est bien exécutée

Object resultObject prends la valeur du résultat de la commande si il y en a un

3.Exécution des scripts

Pour exécuter le client et le serveur :

Cas Unix:

-Allez dans le fichier TP1_Alan_Guivarch_AOOP/scripts/Unix
-Lancez un terminal et exécutez la commande ./serveur.sh
-Lancer un terminal et éxécutez la commande ./client.sh

Cas Windows:

-Allez dans le fichier TP1_Alan_Guivarch_AOOP/scripts/Windows
-Exécutez serveur.bat
-Exécutez Client.bat

Vous pourrez consulter les résultats des commandes dans les fichier sortieClient.txt et sortieServeur.txt