Applications réflexives

TD 1 chargement dynamique et réification des classes

Introduction

Le problème du couplage fort entre le serveur et le service observé dans le td « Service inversion » du module « ApplicationsServeurJava » va ici être résolu en utilisant la possibilité de manipuler, pour chaque classe intervenant dans une exécution de programme Java, une instance de la classe java.lang.Class.

L'architecture de l'application serveur que vous allez mettre en place doit faire apparaitre 3 packages : ***appli***, ***services*** et ***outilserveur***, ce dernier étant un utilitaire (à – très long – terme, une bibliothèque réutilisable)

Le fichier *enoncé.jar* constitue la base de travail pour ce tp. Vous pouvez prendre le code du client sur le corrigé du tp 4, il n'a pas à être modifié.

outilserveur : la relation Serveur-Service-ServiceManager

La répartition des rôles dans *outilserveur* entre ces 3 classes est celle-ci :

- Serveur attend les clients sur un port précisé lors de la création ; losrqu'un client se connecte, il demande une instance de service à ServiceManager ;
- Service est la factorisation des aspects système/réseau d'un service : lancement d'un thread, attribut Socket et déclaration de Runnable ; le run() n'est pas implémenté, cette classe est abstract ;
- ServiceManager est une classe tout static (attributs et méthodes tous static, constructeur bridé) ; elle encapsule la classe de service à instancier.

appli : l'application serveur AppliServeur

Le code du main de l'application *appli.AppliServeur* charge dynamiquement la classe de service et lance le serveur sur un port donné; ces 2 informations sont externalisées du code et placées dans les arguments du main ***args*** : args [0] sera le nom de la classe à charger, args[1] sera le numéro de port.

services : le service d'inversion

Le service est une classe définie à part (package *services*). Le contrat pour cette classe est, d'une part de sous-classer *outilserveur.Service* (et donc d'implémenter le service dans la redéfinition de run()), d'autre part de s'auto-déclarer à ServiceManager via la méthode *init* lors du chargement de cette classe (bloc static).

Travail à réaliser

Vous avez à écrire le code de l'application serveur (dans son package) et à compléter le canevas fourni dans ***enoncé.jar***. En récupérant le code du client, vous testerez l'ensemble.
Après test de ce service, vous pouvez le remplacer par un test de palindrome (ou tout autre travail sur une String saisie coté client) : écrire cette classe dans ***services***, sans rien retoucher au code des 2 autres packages et tester.

En option : le code est stable quand on veut proposer un nouveau service mais il faut arrêter/relancer l'application après avoir modifié les arguments du main. Améliorer cela afin que cette modification soit dynamique.