Programmation Répartie
Travaux Pratiques 1
Développement d'une invocation de méthode distante

Dans ce TP, nous allons développer les bases d'un système d'invocation d'objets répartis. Nous allons partir d'une classe classique de calculatrice que nous allons rendre accessible à distance. Votre développement devra être le plus générique possible pour pouvoir être réutilisé pour une autre classe avec le minimum de changements.

La figure 1 présente un cas classique d'invocation d'objet dans un programme (tous les objets sont dans le même espace d'adressage).

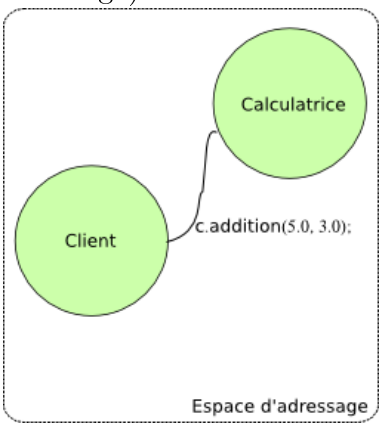

FIGURE 1 – Une invocation de méthode classique

Nous souhaitons rendre notre classe calculatrice accessible à distance tout en conservant un modèle d'appel de méthode classique comme si elle était dans le même espace d'adressage. Pour cela il est nécessaire d'introduire des objets intermédiaires destinés à masquer la répartition et à prendre en charge la communication à distance.

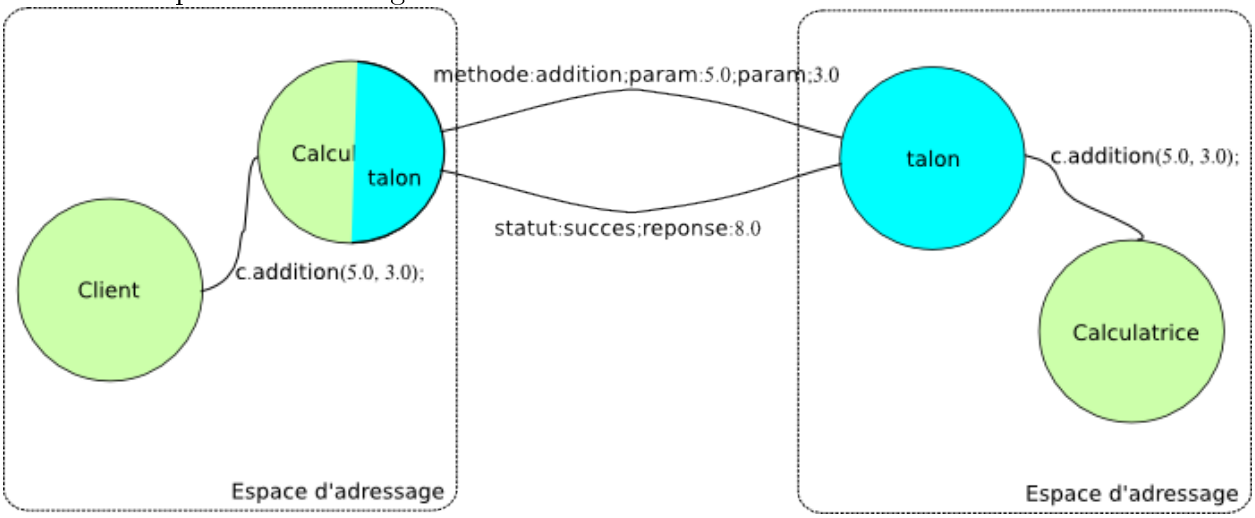

FIGURE 2 – Une invocation de méthode distante

La figure 2 montre les objets nécessaires. Du côté du client, nous avons un objet intermédiaire (*talon*) qui présente les même méthodes que notre calculatrice et qui peut être appelé par le client «comme avant». Ce *talon* va transmettre via une socket les informations sur l'invocation à l'objet intermédiaire situé dans l'espace d'adressage du serveur. Ce *talon* serveur

utilise ces informations pour réaliser l'invocation de méthode locale sur notre calculatrice. Il renverra ensuite le résultat côté client.

Pour ce TP :
— votre code sera développé dans un paquetage `fr.ulille.iut`
— votre code doit pouvoir être exécuté sur n'importe quelle machine. Le client et le serveur ne sont pas forcément sur la même machine.
— vous devez faire des commits **réguliers** au fur et à mesure de votre avancement.

---

1. **Préparation du TP**
   (a) Placer vous dans un répertoire de développement et clonez le point de départ du projet avec la commande suivante :
   `git clone https://git-iut.univ-lille1.fr/peter/da2i_prog_rep`
   (b) Créez un projet **public** sur le gitlab de l'IUT nommé selon le schéma suivant : `da2i_PR_login_groupe` [1].
   (c) Placez vous dans le répertoire `da2i_prog_rep` et effectuez les actions suivantes pour que le code récupéré soit synchronisé avec votre propre projet (en remplaçant bien `login` et `groupe` par les bonnes valeurs) :

```
git remote rename origin old-origin
git remote add origin \
      https://git-iut.univ-lille1.fr/login/da2i_PR_login_groupe.git
git push -u origin --all
```

   (d) Vous disposez maintenant pour point de départ d'une classe calculatrice dans le paquetage `fr.ulille.iut`.
2. **Définition du protocole**

Avant de rentrer dans le code proprement dit, il est nécessaire de réfléchir au protocole de communication qui devra être mis en place entre le client et le serveur. Ce protocole doit définir les messages qui seront envoyés et leur format pour soumettre une opération à la calculatrice et récupérer le résultat. Ce protocole doit être conçu de manière à pouvoir être utilisé avec un objet différent de la calculatrice (i.e. des méthodes différentes).

   (a) Créez un fichier README.md à la racine de votre projet où vous décrirez votre protocole.
   (b) Votre description doit indiquer le ou les messages permettant de demander à la calculatrice d'effectuer une opération. Le ou les messages de réponse avec un résultat ou une erreur.

1. où login est votre nom de login et groupe, la lettre correspondant à votre groupe

(c) Quand vous êtes satisfait de votre protocole, faites un `commit` et un `push` sur gitlab.

3. **Développement du côté serveur**

Nous allons commencer par développer le côté serveur afin de rendre notre calculatrice accessible à distance. Nous allons créer deux classes : une classe `Serveur` et une classe `TalonServeur`.

(a) Le code que vous avez récupéré contient déjà une classe `Serveur`. Modifiez là pour qu'elle attende les connexions des clients et instancie la classe `TalonServeur` pour traiter chaque nouveau client dans un *thread* à part.

(b) Implementez la classe `TalonServeur`. Celle-ci implémentera l'interface `Runnable` afin de pouvoir être exécutée par un *thread*. Cette classe devra lire le message envoyé par le client, y retrouver la méthode invoquée ainsi que les paramètres. Avec ces éléments, elle réalisera l'invocation sur une instance de la classe `Calculatrice`. Ensuite, elle renverra un message au client avec le résultat. Votre implémentation devra vérifier à minima que les messages sont biens formés (i.e., correspondent à votre protocole) et enverra une erreur explicite en cas de problème.

(c) Pour tester le fonctionnement de votre code, vous pouvez utiliser la commande `nc` [2] qui vous permettra de vous connecter et d'envoyer «à la main» vos requêtes au serveur.

(d) Quand vous êtes satisfait de votre implémentation, faites un `commit` et un `push` sur gitlab.

4. **Développement du côté client**

Normalement votre code côté serveur est capable de traiter les messages envoyés avec `nc` et de vous répondre avec le résultat du calcul ou une notification d'erreur. Nous allons maintenant développer et utiliser le talon côté client. Vous allez créer deux classes : une classe `Client` et une classe `TalonClient`.

(a) Le code que vous avez récupéré contient déjà une classe `Client`. Adaptez là afin d'en faire la partie interactive de votre programme permettant à un utilisateur de faire des calculs. Pour chaque calcul, cette classe appellera la méthode correspondante de la classe `TalonClient`.

(b) Implémentez la classe `TalonClient`. Celle-ci sera le représentant de la calculatrice côté client. A ce titre, elle proposera les même méthodes que la calculatrice [3]. Lors d'un appel sur ces méthodes, cette classe enverra au talon côté serveur un message avec les informations concernant l'invocation. Elle renverra ensuite le résultat à la classe client.

2. Pour envoyer votre requête au serveur utilisez CTRL + D plutôt que Entrée qui envoie un retour chariot.

3. La seule différence est qu'il peut maintenant y avoir des problèmes liés à la communication à travers le réseau. Vos méthodes devront donc renvoyer une exception explicite au client dans ce cas.

(c) Quand vous êtes satisfait de votre implémentation, faites un `commit` et un `push` sur gitlab.

5. **Pour aller plus loin**
   L'implémentation standard de méthodes distantes en Java s'appelle Java RMI (*Java Remote Method Invocation*). lisez le tutoriel suivant et comparer la mise en œuvre de RMI avec votre solution.