

# Projet RMI

---

## Simulation de la bourse

**Jean-Christophe SEPTIER Maxime ESCOURBIAC**

**21/12/2011**

## Table des matières

1. Modélisation du projet.....	2
1.1. RMI .....	2
1.2. Diagramme de cas d'utilisation .....	3
1.3. Diagramme de classe.....	4
2. Serveur .....	6
2.1. Simulation de la bourse .....	6
2.2. Implémentation : .....	6
3. Client et administrateur : .....	7
3.1. Implémentation : .....	7
3.2. Interface graphique du client : .....	8
3.3. Interface graphique de l'administrateur : .....	10
Conclusion .....	11

## Table des illustrations

Figure 1: Fonctionnement du RMI.....	2
Figure 2: Diagramme de cas d'utilisation .....	3
Figure 3: Package ContractProject .....	4
Figure 4: Package Server .....	4
Figure 5: Implémentation du client et de l'administrateur.....	5
Figure 6: Client non connecté .....	8
Figure 7: Client n'ayant pas de stock option suivi .....	8
Figure 8: Clients suivant des stocks options.....	9
Figure 9: Fenêtre d'information d'un stock option .....	9

## Introduction

Lors du cours de programmation web, il nous a été demandé de réaliser une application utilisant l'interface RMI, afin de faire connecter un serveur et des clients.

Cette application doit pouvoir simuler la bourse. On simulera un ensemble de stock option, dont la cote peut monter ou descendre avec la même probabilité. Les stocks options pourront être ajoutés ou supprimés par un administrateur. Une limite minimale pour le stock option est définie, sous laquelle un stock option est supprimé.

Un client se connecte à un serveur avec un surnom. Il doit recevoir la liste des stocks options. Il doit pouvoir se souscrire à un ou plusieurs stocks options. Il reçoit les informations concernant le stock option.

Dans ce rapport, nous allons d'abord vous expliquer l'interface utilisée, la modélisation de notre problème, puis l'implémentation du serveur, du client et de l'administrateur.

# 1. Modélisation du projet

## 1.1. RMI

Pour ce projet, nous devons utiliser le RMI (Remote method interface). C'est une interface qui permet à une application java de faire appel à des méthodes distantes. L'utilisation de cette API nécessite l'emploi d'un registre RMI sur la machine distante hébergeant ces objets que l'on désire appeler au niveau duquel ils ont été enregistrés.

Les connexions et les transferts de données dans RMI sont effectués par Java grâce à un protocole propriétaire (JRMP, *Java Remote Method Protocol*) sur le port 1099.

Une application RMI doit avoir trois éléments :

- une application client utilisant le stub
- une application serveur implémentant le skeleton (*squelette*)
- une application médiatrice (le registre RMI) servie par un processus tiers (*rmiregistry*)

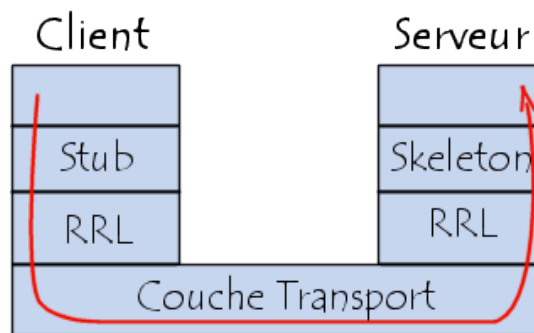


Figure 1: Fonctionnement du RMI

Pour communiquer, le Client utilise un stub pour pouvoir communiquer avec le Skeleton du Serveur. On utilise une couche de référence appelée *RRL* (*remote Reference Layer*) pour fournir un moyen aux objets d'obtenir une référence à l'objet distant. Elle est assurée par le package *java.rmi.Naming*.

La couche de transport permet d'écouter les appels entrants ainsi que d'établir les connexions et le transport des données sur le réseau par l'intermédiaire du protocole TCP.

Pour réaliser une application RMI, on doit donc réaliser plusieurs points :

- Définir les spécifications du service RMI sous forme d'une interface que l'on appellera Contract.
- Créer et compiler les implémentations de ces Contract.
- Créer le stub avec la commande *rmic*.
- Lancer le *rmiregister* pour pouvoir faire communiquer le client et le serveur.

Pour notre application, le client doit pouvoir communiquer avec le serveur, mais le serveur doit également pouvoir envoyer des informations au client. On doit donc créer un skeleton et un contract pour le serveur et pour le client, mais aussi pour l'administrateur.

Pour pouvoir récupérer facilement les informations concernant un stock option, on crée également un contract et un skeleton pour un stock option. Ainsi, le client communiquera directement avec le stock option, sans passer par le serveur.

## 1.2. Diagramme de cas d'utilisation

On peut, grâce à ce diagramme, voir l'ensemble de fonctionnalité disponible pour les différents types d'utilisateurs.

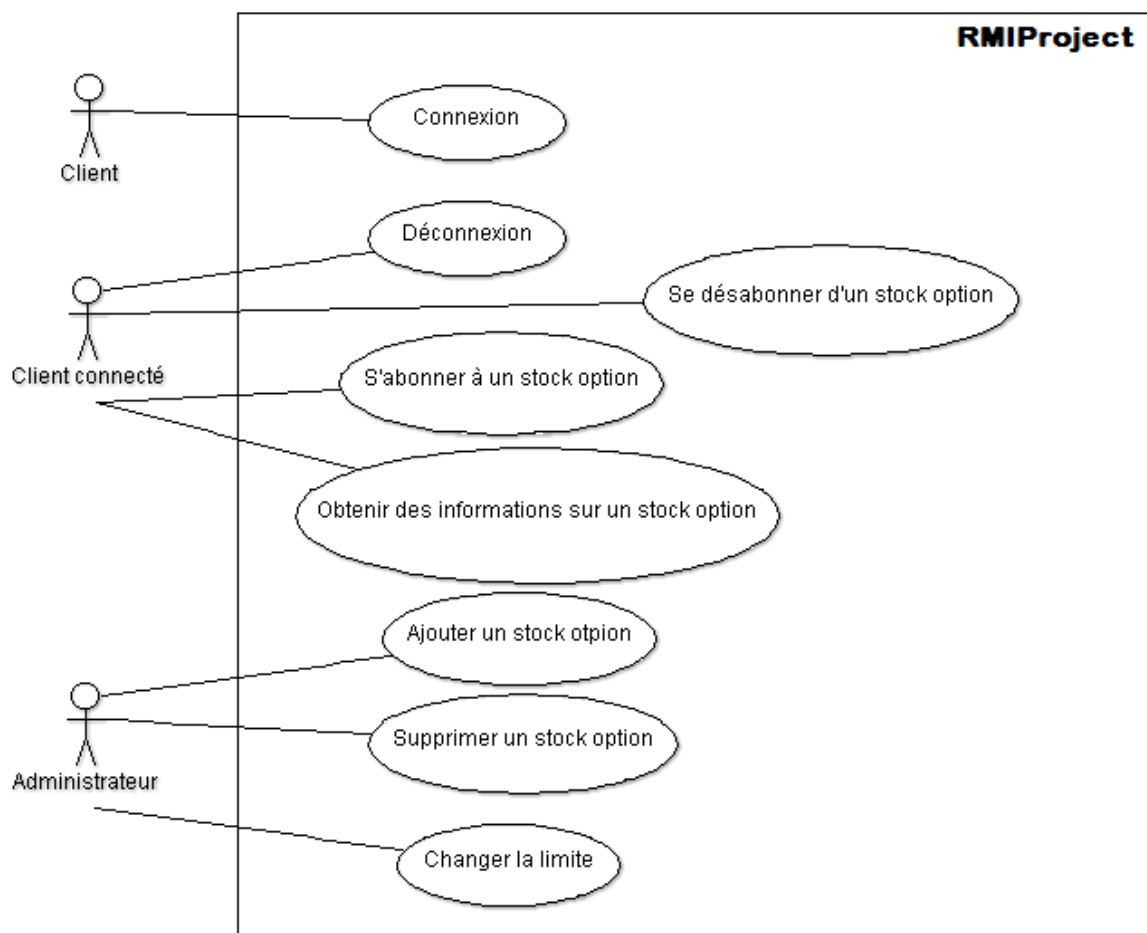


Figure 2: Diagramme de cas d'utilisation

### 1.3. Diagramme de classe

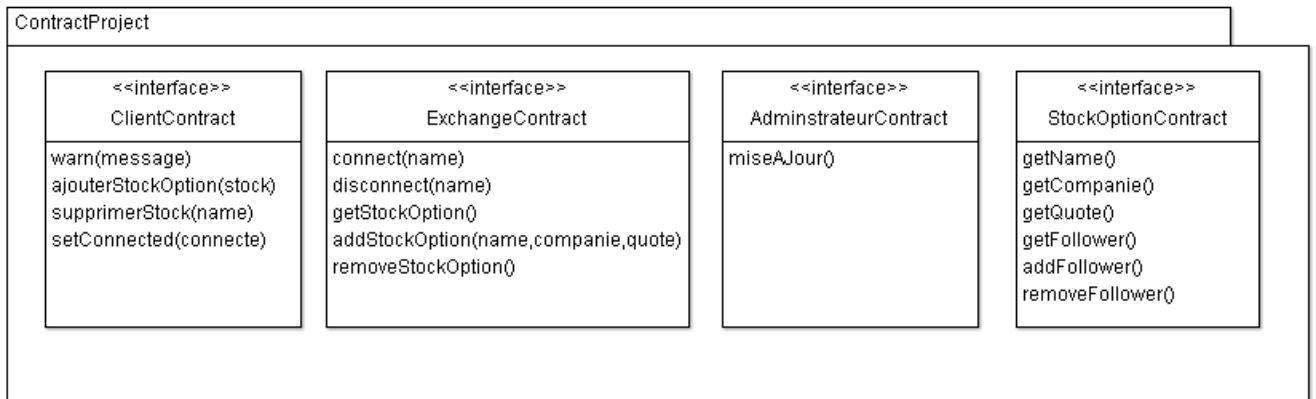


Figure 3: Package ContractProject

Tout d'abord, nous avons un package regroupant l'ensemble des contrats nécessaire :

- **ClientContract** : Permet d'afficher un message à un client (`warn`), ajouter un stock option, supprimer un stock option et recevoir une information pour se connecter ou se déconnecter (`setConnected`).
- **ExchangeContract** : Permet de communiquer avec le server, avec des demandes de connexion, de déconnexion, d'ajout et de suppression de stock option, et permet d'envoyer la liste des stocks options.
- **AdministrateurContract** : Permet de dire à l'administrateur que la liste des stocks options a été modifiée.
- **StockOptionContract** : Donne les informations correspondant à un stock option, et permet d'ajouter ou supprimer des abonnés.

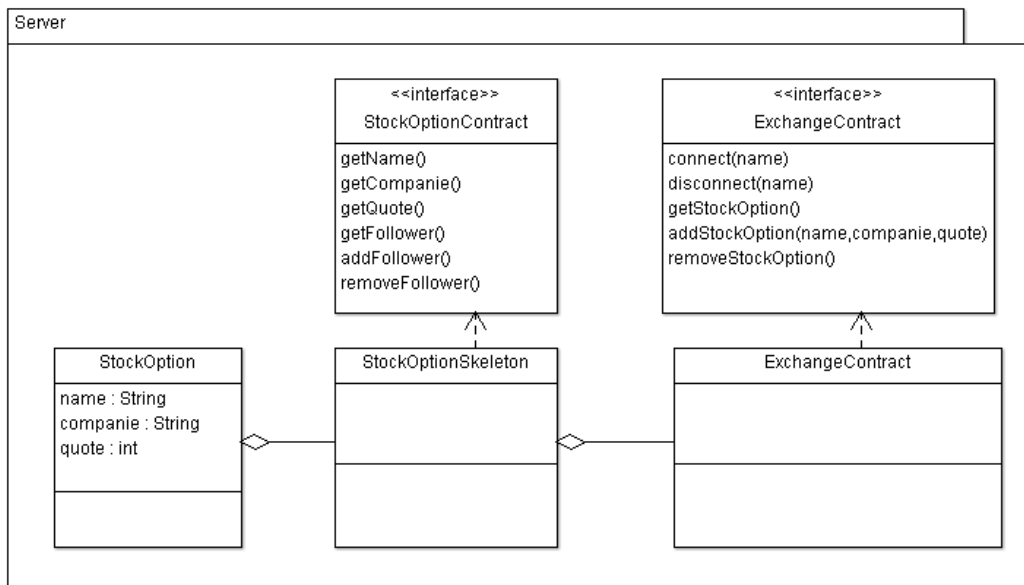


Figure 4: Package Server

Le package server implémente les interfaces StockOptionContract et ExchangeContract. La classe ExchangeContract possède un ensemble des StockOptionContracts. StockOptionSkeleton possède une classe StockOption, afin de séparer les données du contrôleur.

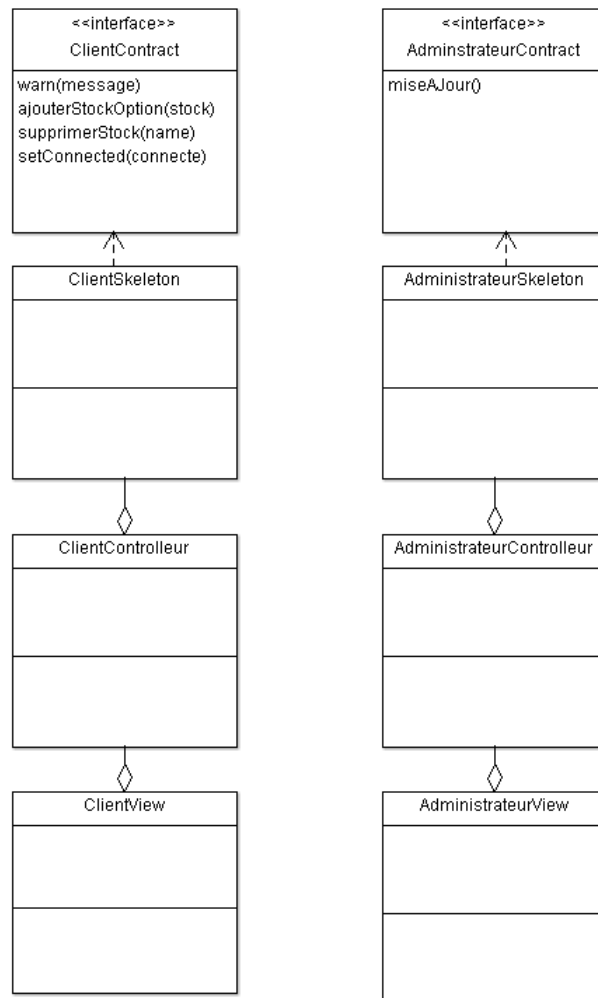


Figure 5: Implémentation du client et de l'administrateur

Le client et l'administrateur respecte le modèle vue contrôleur. Ainsi, on a une vue qui communique avec un contrôleur. Le contrôleur récupère les données grâce au Skeleton du client ou de l'administrateur, qui implémente les méthodes des contrats respectifs.

## 1. Serveur

### 1.1. Simulation de la bourse

Pour ce projet, notre serveur réalisera une simulation de bourse. La classe ServerSkeleton contient certaines informations en static, pour pouvoir les changer plus facilement, comme l'heure d'ouverture de la bourse, l'heure de fermeture ou la variation.

La bourse ouvre à une certaine heure, puis ferme. Pendant l'ouverture, on simule un changement de la cote de chaque stock option. On envoie aux abonnés une notification informant sur le changement. Une fois la bourse fermée, on calcule de combien a augmenté ou diminué la bourse dans la journée, puis on change de jour.

#### Algorithme :

```
Faire
  Si la bourse ferme
    Calculer la variation de la journée et l'envoyer à tous les clients
    Changer de jour
  Finsi
Envoyer l'heure à tous les clients
Pour tous les stocks options
  Calculer la nouvelle quote.
  Si la quote est inférieur à la limite
    Supprimer le stock option
  Fsi
  Envoyer aux abonnés la nouvelle quote
Finpour
Changer d'heure
Tant que le serveur est ouvert
```

### 1.2. Implémentation :

La classe ExchangeServeur lance un Thread de ExchangeSkeleton. Cette classe va lancer la simulation de la bourse. Ce skeleton va recevoir l'ensemble des informations venant du client ou de l'administrateur.

Il possède une map associant un nom de client avec un contrat de client, ainsi qu'une map associant un nom de stock option à un contrat de stock option. On va ainsi pouvoir retrouver facilement le contrat pour communiquer avec un client ou un stock option. On va pouvoir ainsi communiquer en envoyant simplement la clé correspondante.



Le serveur reçoit du client :

- Les demandes de connexions : il vérifie si le nom de client n'est pas déjà utilisé, bind le client et lui envoie une confirmation de connexion avec la liste des stocks options disponibles.
- Les demandes de déconnexions : il désabonne le client de ses stocks options, et le supprime.
- Les demandes d'abonnements : il abonne le client au stock option demandée.
- Les demandes de désabonnement : il désabonne le client.

Il reçoit également de l'administrateur:

- Les demandes de connexions et déconnexions
- Les demandes d'ajout de stock option : ajoute un stock option, et envoie une notification au client afin de l'ajouter, et aux administrateurs afin qu'ils mettent à jour leur liste.
- Les demandes de suppressions : idem que l'ajout.
- Les demandes de changement de la limite : une notification est envoyée aux administrateurs afin qu'ils mettent à jour la limite.

C'est également le serveur qui implémente le skeleton associé à un stock option. Celui permet de retourner les informations correspondantes, comme son nom ou sa cote.

## **2. Client et administrateur :**

### **2.1. Implémentation :**

Le client respecte le modèle vue controleur. On sépare donc la vue de la communication avec le serveur. On a donc une classe ClientControlleur et ClientView. Le controleur communique avec le server grâce au skeleton du client, et grâce à l'ExchangeContract.

Il possède une map de stockOptionContract pour communiquer avec les stocks options pour récupérer les informations correspondantes.

L'interface graphique a été réalisée grâce à l'interface de NetBeans, permettant de réaliser rapidement une interface avec des éléments appartenant à la bibliothèque AWT.

Il en est de même pour l'administrateur.

## 2.2. Interface graphique du client :

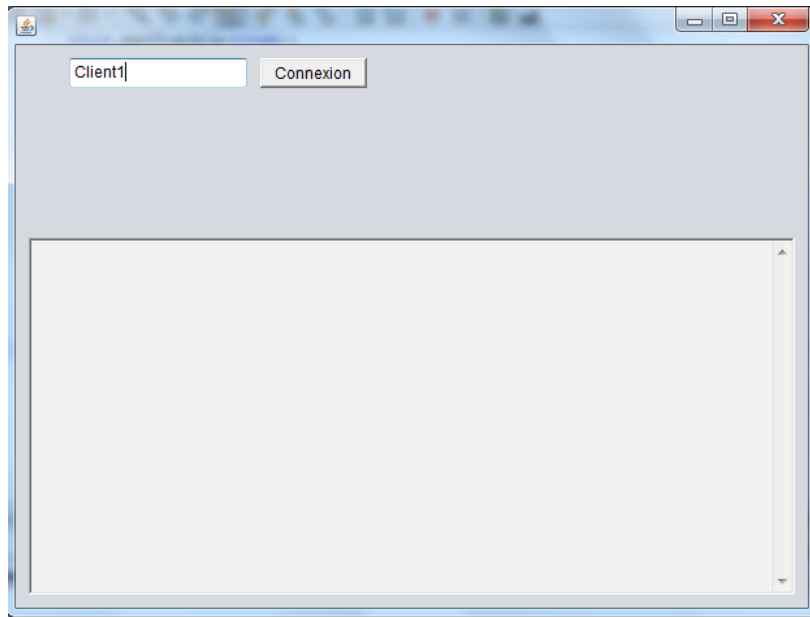


Figure 6: Client non connecté

A l'ouverture du client, une fenêtre apparaît, permettant de se connecter. On entre le nom du client, puis on clique sur le bouton « Connexion ». Le serveur vérifie si le nom de client n'est pas déjà utilisé. Si oui, un message d'information apparaît afin de prévenir l'utilisateur. Sinon, l'interface change.

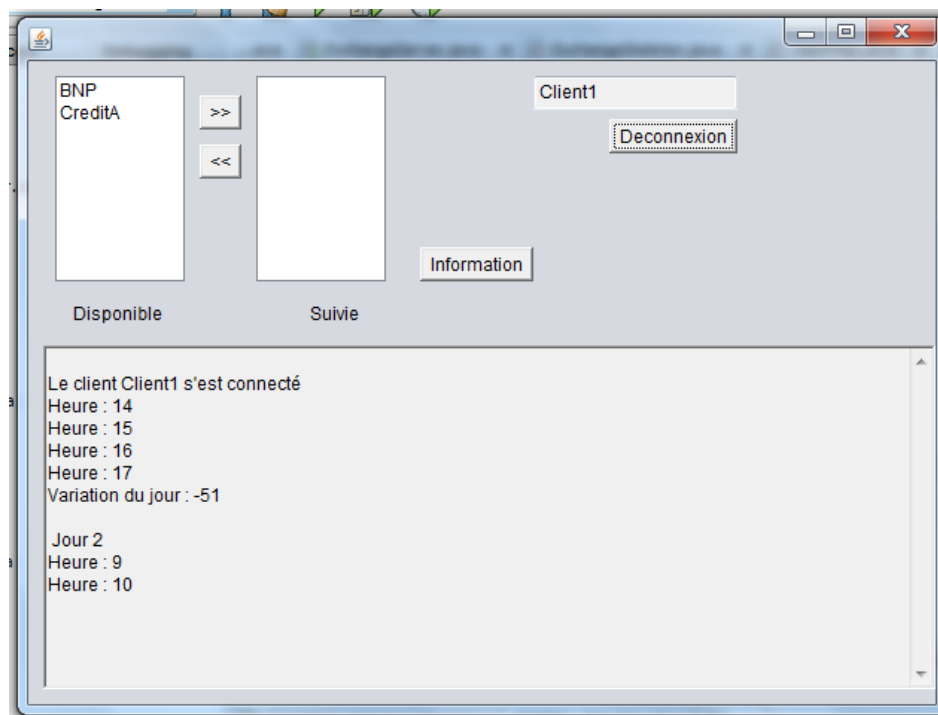


Figure 7: Client n'ayant pas de stock option suivi

Le client reçoit les informations du serveur, avec le changement d'heure, le changement de jour et d'heure. Le client peut choisir un des stocks options disponibles. Il peut faire les stocks options de disponibles à suivie, et inversement. Les informations sont envoyées au serveur. Ainsi, il reçoit maintenant les informations des stocks options suivis, avec leur nom, leur nom de la compagnie, et leur nouvelle cote.

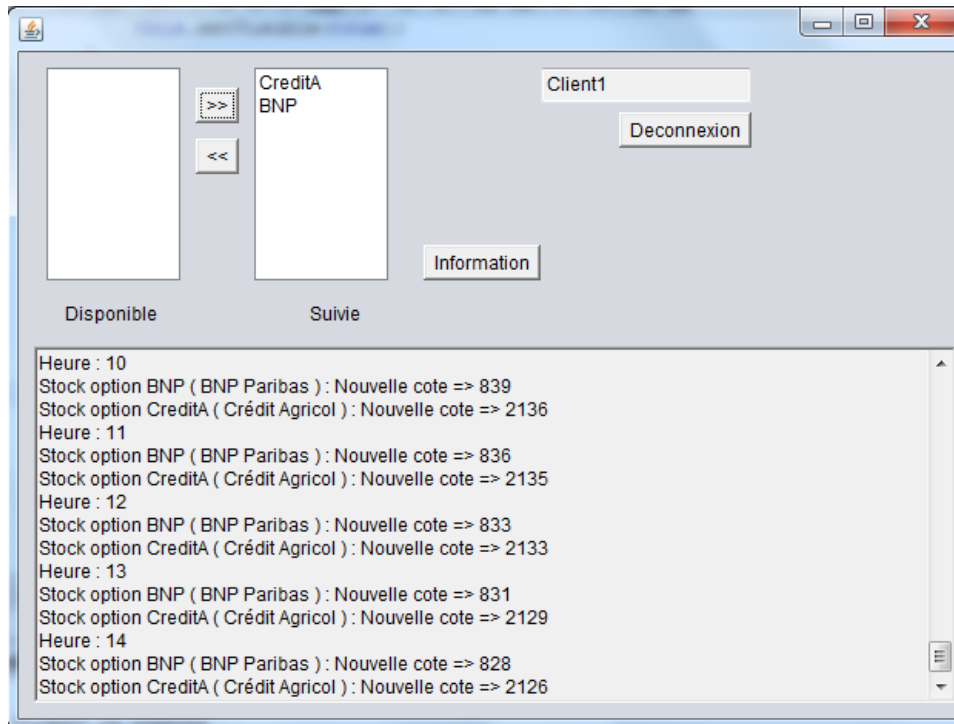


Figure 8: Clients suivant des stocks options

On peut connaître les informations d'un stock option non suivi en cliquant sur le bouton « Information ». Une fenêtre pop-up s'affiche avec les informations du stock option.

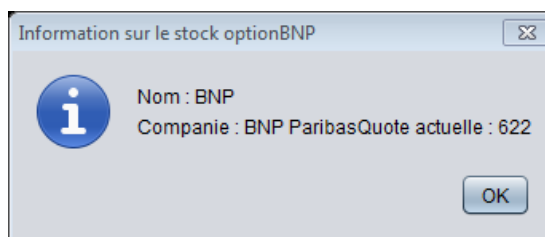
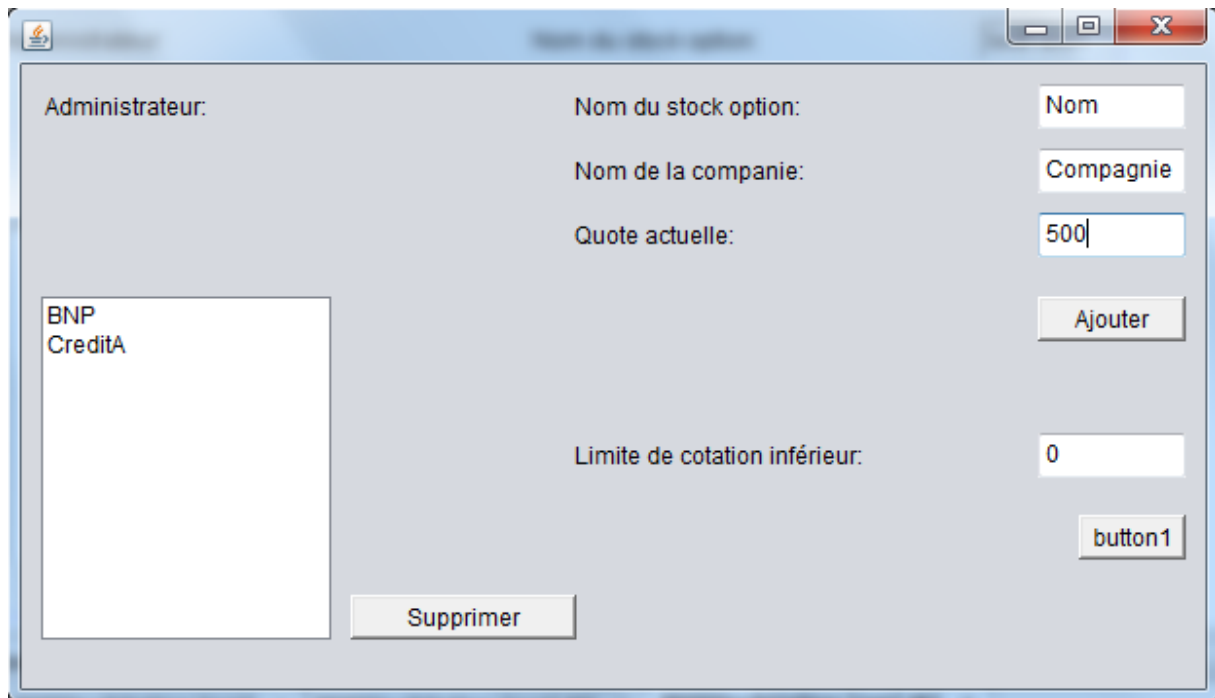


Figure 9: Fenêtre d'information d'un stock option

Le client reçoit à tout moment des informations sur les stocks options ajoutés ou supprimés, ainsi que la raison, si cela est dû à un passage sous la limite de cote. Il ajoute les stocks options ajoutés en cours dans le liste des stocks options disponibles, et supprime dans l'interface les stocks options qui ne sont plus disponibles. Le client peut à tout moment se déconnecter. Un message est ainsi envoyer au serveur, pour pouvoir supprimer les abonnements du client. L'événement de fermeture de fenêtre est également intercepté afin de quand même déconnecter le client s'il ferme en étant connecté.

### 2.3. Interface graphique de l'administrateur :



The screenshot shows a Java Swing window titled "Interface de l'administrateur". The window has a standard Mac OS X-style title bar with minimize, maximize, and close buttons. The main content area is light gray and contains the following elements:

- Administrateur:** A label on the left side.
- Nom du stock option:** A text label with a text input field containing the text "Nom".
- Nom de la compagnie:** A text label with a text input field containing the text "Compagnie".
- Quote actuelle:** A text label with a text input field containing the text "500".
- Ajouter:** A button located below the "Quote actuelle" field.
- Limite de cotation inférieur:** A text label with a text input field containing the text "0".
- button1:** A button located below the "Limite de cotation inférieur" field.
- Supprimer:** A button located at the bottom center of the window.
- List Box:** A rectangular area on the left side containing the text "BNP" and "CreditA".

L'administrateur n'a pas besoin de se connecter. Pour faciliter l'implémentation, un nom est généré par le serveur pour savoir avec qui il communique.

En ouvrant la fenêtre, on a directement accès à l'ensemble des informations. On a la liste des stocks options. On peut supprimer un stock option sélectionné, ou ajouter un stock option avec les informations entrées en haut à droite. L'information sera envoyée au serveur, qui se chargera de prévenir l'ensemble des clients des modifications.

A chaque modification de la base, l'administrateur reçoit une information, et met à jour les informations concernant les stocks options. La limite peut également être changé. Elle est à 0 par défaut.

## Conclusion

Le projet a été réalisé dans son intégralité. Toutes les fonctionnalités ont été implémentées. Il pourrait être possible dans l'avenir d'améliorer la sécurité de l'application, avec notamment une gestion de mot de passe pour l'administrateur.

Ce projet nous a permis de découvrir la technologie RMI. Celle-ci nous étant complètement inconnue, la mise en place de l'application a été la partie la plus difficile à réaliser.

Il nous a fallu un certain temps pour comprendre le principe de la création des stubs et de la communication par contract. Une fois cette particularité comprise, le reste a été beaucoup plus simple. Nous avons donc remarqué que le RMI est beaucoup plus est beaucoup plus difficile à mettre en place qu'une solution utilisant les web services, qui ont remplacé cette technologie.