GUIDE SUR LE FONCTIONNEMENT DU MARCHÉ

Nassym BEN KHALED

Introduction

Toutes les classes dédiées spécifiquement au fonctionnement de notre marche boursier se trouvent dans le package `manageStockExchange`.
Comme nous le verrons par la suite, notre marche boursier suit une architecture (client/serveur) et un protocole (producteur/consommateur), dont les classes se trouvent dans d'autres packages de notre application.
Néanmoins, ces classes, dynamiques, garderont un comportement ajusté en fonction du contexte avec lequel elles sont appelées.
Par exemple, le thread que lance un serveur pour traiter les différentes requêtes reçues des clients, agira différemment si la relation client/serveur correspond à celle d'un courtier avec la bourse ou alors à celle d'un client avec son courtier.

Fonctionnement de notre marché boursier

Notre marché boursier (classe `stockExchange`) implémentera toutes les grandes fonctions imposées par le sujet du projet, à savoir :

- ouverture/fermeture du marché
- enregistrement/déconnexion des courtiers
- enregistrement des ordres des courtiers
- traitement des ordres déposés par les courtiers et exécution des transactions
- mise à jour des prix des actions
- notification de l'état du marche aux courtiers

ouverture du marche

La bourse ouvrira le marche dès lors qu'elle reçoit un message (« **OPEN** ») pour ce faire, de la part du gestionnaire de la bourse.
Pour cette première version, ce message devra être entré manuellement à l'aide du clavier, dans la console de l'application.
Dans une version ultérieure, avec l'implémentation d'une interface graphique usager, un bouton dédié à l'ouverture du marché pourrait simuler cette entrée.
Tant que ce message n'est pas explicitement entré, la bourse n'ouvrira pas.

Par ailleurs, pour cette première version, le marche est réinitialisé à chaque ouverture (liste des entreprises, des courtiers, des ordres, etc. réinitialisés). Dans une version ultérieure (plus réaliste), on pourrait penser à un chargement de l'état du marché de la veille (récupéré via un fichier `config.txt` par exemple).

Fermeture du marché

La fermeture du marché est plus automatique que son ouverture. En effet, c'est la fin d'une journée ouvrable qui décidera de sa fermeture. Or une journée ouvrable se termine si et seulement si tous les courtiers se sont déconnectés. C'est donc seulement lorsque les courtiers se seront tous déconnectés que la bourse fermera.

La fermeture du marché implique les taches suivantes :

- mise à jour des prix des actions pour chaque entreprise sur le marché (cf. section ***Mise à jour des prix des actions***)
- arrêt de tous les threads traitant les requêtes des clients (courtiers)
- arrêt du thread modélisant le marche

Enregistrement/Déconnexion des courtiers

Les courtiers envoient des requêtes au marché. Celui-ci, allouera, à chaque fois, un thread dédié pour les traiter (et y répondre).

L'enregistrement auprès du marche est un des types de requête que le courtier enverra au serveur (détecté par le thread charge de répondre au courtier, grâce au champs `ContentType` du message reçu, et qui vaudra "`registerBrokerRequest`"). Il s'agit des lors de l'enregistrer dans le répertoire des courtiers du marché et de lui acquitter cet enregistrement. Il pourra dès lors envoyer les autres types de requêtes (placement d'un objet `Message` de type `order`, `stateMarket`, ou `deconnexion` par exemple).
Le courtier transmet au marché, via cette requête (dans le champ `content` de l'objet `Message` transmis), son nom, ainsi que le nombre de clients connectés à lui (au moment de l'enregistrement) sous le formalisme :

(nom du courtier, nombre de clients qui lui sont connectés)

Une fois l'enregistrement effectué, le thread va répondre au courtier en lui envoyant un objet `Message` d'acquittement (`ContentType = "registrationBrokerAcknowledgement"`).

Pour se déconnecter, le courtier enverra un message de ce type donc au marché. Celui-ci acquittera sa demande via un thread dédié à son traitement (qui la détectera grâce au champs `ContentType` du message reçu, lequel vaudra "`deconnect`"), mais avant de le retirer du marché, ce dernier lui envoi un état du marche avant sa déconnexion. Ensuite, on le supprime du répertoire où il avait été enregistre lors de son inscription. Si il était le dernier courtier enregistré, on lance automatiquement la procédure de fermeture du marché.

Enregistrement des ordres des courtiers

Les courtiers pourront également demander au marche d'exécuter des ordres (des `SellOrder` ou des `PurchaseOrder` selon la nature de l'ordre), lesquels seront encapsulé sous le format JSON dans le contenu de l'objet `Message` transmis au marché. Le thread alloué au traitement de

la requête du courtier détectera le placement d'un ordre par ce dernier grâce au champs `ContentType` de l'objet `Message` transmis (et qui vaudra alors `"order"`).

L'ordre sera ensuite analysé pour déterminer s'il s'agit d'un ordre de vente ou d'achat, puis l'objet correspondant sera recréée par parsing du JSON décapsulé, et enfin ajouté à la liste des ordres du marché en attente de traitement.

Le thread alloué à la réponse de la requête du courtier se mettra alors en sommeil, et ne sera réveillé que lorsque le marché aura traité son ordre (partiellement, totalement, ou encore pas du tout si impossible. cf. ***Traitement des ordres***).
Il enverra alors au courtier la chaine de caractères du JSON correspondant à son ordre, mis à jour après le traitement par le marché.

Traitement des ordres

Le marché allouera dès le début un Thread pour "consommer" les ordres places dans la file (respectant l'ordre FIFO). Ce thread respectera les propriétés bloquantes d'une file (attente si vide par exemple).

Il retirera donc l'ordre de tête de file, et lancera le processing du marché pour le consommer, c'est à dire pour le traiter.

Le marché va selon la nature de l'ordre, effectuer des actions bien précises

- si l'ordre est un ordre de vente

le marche cherchera le meilleur ordre d'achat présent dans la file (relatif à la même entreprise), qui permettrait d'aboutir à une transaction équitable, tout en gardant une priorité sur le vendeur (comme c'est son ordre qui est traité). Les considérations algorithmiques et choix effectués pour l'obtention de cette transaction "équitable" sont détaillés dans la partie ***"Considérations algorithmiques"*** ci-dessous.
Si un ordre d'achat correspondant à notre ordre de vente est trouvé, une transaction est possible. Sinon, la transaction n'est pas possible.

Dans le cas d'une transaction, on met alors a jour les données des ordres qui ont eu un "matching" (date de traitement, prix réel de vente/d'achat, quantité effectivement vendue/achetée). On stockera ensuite leur représentation sous format JSON (dans une chaine de caractères), dans le champs prévu pour la réponse du serveur dans chaque thread qui a été alloué pour la réception et le traitement de ces ordres. Nous réveillons alors ces mêmes threads, pour qu'ils envoient aux courtiers associes à cette transaction, les données sur leurs ordres, issues du traitement par le marché. Ces données (JSON mis à jour de l'ordre sera encapsulé dans un `Message` avec pour `ContentType, "agreementAcknowledgment"`).

Dans le cas d'une impossibilité de transaction, nous réalisons les mêmes étapes que pour une transaction, à la différence près que seul le courtier ayant déposé l'ordre (qui n'a pas pu trouver acheteur) est averti (puisqu'un deuxième ordre n'est pas impliqué), via un objet `Message` dont le `ContentType` sera `"orderImpossible"`.

- si l'ordre est un ordre d'achat

Le marché va tout d'abord chercher à acheter des actions flottantes pour l'entreprise considérée par l'ordre (en ne dépendant alors pas d'ordres de ventes).
Le meilleur des cas est donc de pouvoir acheter autant d'actions flottantes que le nombre souhaité par l'ordre d'achat.
Mais le nombre d'actions flottantes disponibles pourra s'avérer être insuffisant pour satisfaire les souhaits de l'ordre. Nous tentons alors d'acheter le maximum d'actions flottantes disponibles, et de tenter d'obtenir le maximum restant (du restant d'actions), par la recherche d'ordres de ventes dans la file, selon le même principe que précédemment, de manière symétrique.

Le marché cherchera donc le meilleur ordre de vente présent dans la file, qui permettrait d'aboutir à une transaction équitable, tout en gardant une priorité sur l'acheteur (comme c'est son ordre qui est traité).
Les considérations algorithmiques et choix effectués pour l'obtention de cette transaction "équitable" sont également détaillés dans la partie ***"Considérations algorithmiques"*** ci-dessous.
Si un ordre de vente correspondant à notre ordre de vente est trouvé, une transaction est possible. Sinon, la transaction n'est que partiellement (si on a pu avoir des actions flottantes) ou pas possible (si il n'y avait plus d'actions flottantes disponibles).

Dans le cas d'une transaction, on met alors a jour les données des ordres qui ont eu un "matching" (date de traitement, prix réel d'achat/de vente, quantité effectivement achetée/vendue). On stockera ensuite leur représentation sous format JSON (dans une chaine de caractères), dans le champs prévu pour la réponse du serveur dans chaque thread qui a été réveillons alors ces mêmes threads, pour qu'ils envoient aux courtiers associés à cette transaction, les données sur leurs ordres, issues du traitement par le marché. Ces données (JSON mis à jour de l'order, sera encapsulé dans un `Message` avec pour `ContentType, "agreementAcknowledgment"`).
Il y a transaction dès lors qu'au moins une action a été achetée grâce à un ordre de vente (même si le nombre effectif d'actions achetées, mis à jour par l'algorithme pour l'ordre d'achat, comprendra également le nombre d'actions flottantes achetées).

Dans le cas d'une possibilité de satisfaire entièrement (ou uniquement partiellement) un ordre d'achat exclusivement par des actions flottantes, le principe reste le même mais seul le courtier relatif à l'ordre d'achat sera averti (pas de transaction à proprement dit pour avertir deux courtiers de deux ordres), via un objet `Message` dont le `ContentType` sera `"purchasedWithFloatingStocksAcknowledgment"`

Dans le cas d'une impossibilité de transaction, le principe reste aussi le même mais seul le courtier relatif à l'ordre d'achat sera averti (pas de transaction), via un objet `Message` dont le ContentType sera `"orderImpossible".`

Mise à jour des prix des actions

À la fin de chaque journée (avant de fermer), la bourse appelle une mise à jour des prix des actions.
Cette mise à jour reprend la formule mathématique (détaillée dans le sujet du projet).
Il est donc très important de conserver, au sein du marché, une correspondance "offre/demande" pour chaque action d'entreprise. Cela sera fait au moyen d'un dictionnaire, qui fera correspondre, à chaque objet relatif à une entreprise, une liste de deux entiers : le nombre d'actions offertes et le nombre d'actions demandées (en respectant cet ordre). Ces nombres correspondent aux nombres théoriques totaux idéalement voulus au travers des ordres d'achat et de ventes.

Notification de l'état du marché aux courtiers

Les courtiers pourront demander au marché un état actuel du marché. C'est à dire une chaine de caractères, décrivant le JSONArray, lui-même contenant donc une description JSON pour chaque action en donnant :

- le nom de l'entreprise (champs `"nameCompany"`)
- le prix de son action sur le marché (champs `"priceStock"`)

Ce JSON sera transmis dans la chaine de caractères (du thread traitant la requête du courtier) dédié à la réponse du server, puis ce thread le récupérera, et l'encapsulera dans le champs `"content"` d'un objet de type `Message`.
C'est ensuite au courtier de le parser, afin d'avoir une représentation plus "visuelle" de l'état du marché.

Considérations algorithmiques (exécution des ordres)

Ces considérations s'appliquent surtout pour rechercher et trouver un ordre "optimal" qui "matche" avec l'ordre actuellement traite. En d'autres termes, il s'agira de trouver le meilleur ordre d'achat si on traite un ordre de vente, et de manière symétrique, le meilleur ordre de vente si l'on traite un ordre d'achat.
Pour autant, l'ordre traite aura une plus grande priorité sur celui qui "matchera", ce dernier étant encore en attente dans la file.

Comment trouver ce meilleur ordre ?
Et bien, dans un premier temps, nous chercherons à trouver un ordre symétrique optimal qui satisfait pleinement le nombre d'actions désirées vendues (pour un ordre de vente) ou achetées (pour un ordre d'achat). Parmi les ordres satisfaisant intégralement les quantités voulues pour l'ordre courant, nous recherchons celui, pour lequel la vente le "pénalisera" le moins. C'est à dire, ayant l'écart le plus faible entre ce qu'il cherche lui a acheter (si on traite actuellement un ordre de vente) ou à vendre (si on traite actuellement un ordre d'achat).

Si cela n'est pas possible, nous ne pourront pas pleinement satisfaire nos désirs (en quantité d'actions evidemment) et donc nous revoyons nos ambitions à la baisse. Nous cherchons alors le

meilleur ordre symétrique parmi ceux disponibles dans la file, c'est a dire celui qui va maximiser le nombre d'actions achetées (si on traite un ordre de vente) ou le nombre d'actions vendues (si on traite un ordre d'achat), et donc minimiser l'écart entre les désirs de l'ordre traite actuellement et ceux de l'ordre symétrique trouve.

Nous agissons donc en deux temps : d'abord, nous cherchons a chercher un ordre symétrique qui satisfait pleinement nos désirs d'actions. Si cela n'est pas possible, on prend le meilleur qu'on trouve parmi les ordre symétriques.

Enfin, le calcul du prix "effectif" lors d'une transaction est lui aussi soumis à un choix algorithmique :

Pour rester "justes et équitables" dans le deal, nous avons fait le choix de calculer le prix effectif comme la moyenne entre le prix minimum de vente d'une part de l'ordre de vente considéré dans la transaction, et le prix maximum d'achat d'autre part de l'ordre d'achat considéré dans la transaction.

Précision supplémentaire pour les achats d'actions : notre algorithme a la souplesse de pouvoir permettre d'exécuter un ordre d'achat, même en "mixant" des actions flottantes et des actions issues d'un ordre de vente (via un transaction).

Pour calculer le prix effectif d'achat, nous faisons donc une moyenne pondérée entre

- le prix d'une action flottante (prix du marche) et la quantité d'actions flottantes achetées.
- le prix d'une action issue de transaction (et calculé selon le principe précédant) et la quantité d'actions achetées via un ordre.