ELBEZ Samuel & JACQUETTE Pierrick
samuel.elbe@gmail.com & pierrick.jacquette@gmail.com
21200353 & 21305551

PROJET

Le serveur va lancer un thread pour chaque nouveau client avec toutes ses variables qui seront partagées. Chaque client est un thread, pour la communication direct une nouvelle socket est crée. On prend en compte les Ctrl-C de la part des clients et du serveur.

Architecture :

- Advert : objet représentant une annonce qui contient un id, un contenu et l'identifiant du vendeur.

- Client : lance un thread.

- ClientThread : permet de vérifier les demandes de l'utilisateur faites au clavier, les envoyer au serveur, recevoir ce que nous envoie le serveur et de lancer la communication client-client.
  - bufferedReader pour lire l'input de la socket.
  - bufferedReader pour lire ce qu'écrit l'utilisateur.
  - PrintWriter pour écrire dans l'output de la socket.
  - socket sur laquelle le serveur nous a accepté.

- ClientP2P : gère la communication client-client.
  - bufferedReader pour lire l'input de la socket avec le client.
  - bufferedReader pour lire ce qu'écrit l'utilisateur.
  - PrintWriter pour écrire dans l'output de la socket avec le client.
    socket sur laquelle le serveur nous a accepté.

- Message: énumération des différents messages possibles.

- MessageServer : envoie les messages par le serveurs en fonction du protocole

- MyException : définition des différentes erreurs possibles dans les messages reçus.

- Server : objet représentant le serveur avec une socket, une liste d'annonces, une liste d'utilisateur, un compteur pour les id utilisateurs et un pour les id des annonces.

- ServerThread : objet représentant un client qui est connecté au serveur, reçoit les messages, les vérifie et envoi un message au client une réponse.
  - bufferedReader pour lire l'input de la socket.
  - messageServer pour écrire sur l'ouput de la socket.
  - Socket sur laquelle on accepte le client.
  - boolean pour savoir si l'utilisateur se déconnecte.
  - hashMap d'annonce pour avoir toutes les annonces : variable partagé.
  - hashMap d'utilisateur pour avoir toutes les utilisateurs : variable partagé.

- counter pour savoir l'id d'une nouvelle annonce et nouveau user : variable partagé.
- idUser : id de l'utilisateur de ce thread.

➢ User : objet représentant un utilisateur qui contient un id, une adresse, un port et une liste de ses annonces.

Nous avons choisi d'utiliser des HashMap pour stocker nos éléments car on peux les supprimer en O(1) (éviter de parcourir la structure à chaque fois).

Pour synchroniser nos objets nous utilisons le mot clé « synchronized » sur nos méthodes run.