PR6 – Programmation réseaux
TP n° 4 : Un serveur multifilaire

Comme dans celui de la semaine dernière, dans ce TP toutes les transmissions seront faites en mode connecté TCP.

Vous rappelez-vous du jeu « deviner un nombre aléatoire entre 1 et 100 » de la dernière séance (TP 3, Exercice 3) ? Il s'agit aujourd'hui de le transformer en un serveur multifilaire (*multithreaded*) en bonne et due forme.
On met en place un protocole simple pour encadrer les échanges entre le serveur et le client : Un fois la connexion établie, le client commence par s'identifier par un prénom. Il attends ensuite de recevoir le message `!` du serveur. Ensuite, il envoie des entiers, et le serveur réponds par `+`, `-` ou `=`. Si le serveur ne comprends pas le message du client, il l'ignore et envoie le message `?` au client, pour que celui-ci lui renvoie le message. Si la partie finit pour un autre motif que le gain du client, le serveur envoie le message `.` suivi d'une chaine de caractères contenant le nombre à deviner.
Les fichiers `Server.java` et `Client.java` implémentent ce protocole.

Exercice 1 : Les fils d'execution

Pour commencer, on veut juste permettre à plusieurs joueurs de jouer en même temps, mais séparément. Créez une classe `PlayerService` implémentant l'interface `Runnable` et une classe `ThreadedServer` qui reçoit les connexions et crée un *fil de joueur* (un `Thread`) de `PlayerService` à chaque fois qu'un nouveau client se connecte.
Il s'agit essentiellement de copier-coller, mais il faut faire attention à quelles sont les tâches léguées au `PlayerService` et quelles sont celles réservées au `ThreadedServer`.
Avez-vous besoin de changer vos clients pour cela ?

Exercice 2 : Un jeu multijoueur

Maintenant, on veut ajouter un peu d'esprit de compétition à notre jeu. L'idée est que deux ou plusieurs joueurs qui sont connectés en même temps puissent faire la course à deviner avant les autres le nombre choisi aléatoirement.
Il faut vérifier à chaque fois qu'un joueur se connecte, s'il y en a d'autres connectés. Si ce n'est pas le cas, *lancez une partie*, c'est-à-dire, choisissez un nombre et laissez-le jouer.
S'il y déjà d'autres joueurs en ligne, donc s'il y a déjà une partie en cours, le nouveau joueur *joint la partie*, c'est-à-dire, qu'il se met à deviner le même nombre.
Dès qu'*un joueur gagne*, i.e. trouve le nombre, interrompez le jeu des autres et envoyez leur le bon nombre et l'identifiant du gagnant. N'oubliez pas de féliciter celui-ci. Le jeu est fini et vous pouvez déconnecter tout le monde.
Appelez le serveur de connexions `CompetitiveServer` et les implémentations `Runnable` de fils de joueur `CompetitivePlayerService`. Pour gérer les interruptions, créez un fil intermédiaire, *fil de partie* `GameServer` entre les deux, qui crée les fils de joueur, et qui conserve une liste des fils de joueurs. Quand un joueur gagne, son fil interrompt le `GameServer`, qui à son tour, interrompt les autres fils de joueur. Il faut que ceux-ci gèrent correctement les interruptions.

Exercice 3 : Plus de fils

Dans son état actuel, le jeu n'est ni juste (les joueurs qui se connectent d'abord ont un avantage), ni *scalable* (le jeu perd son intérêt s'il y a trop de joueurs connectés). On veut maintenant modifier le jeu afin d'avoir plusieurs parties en parallèle, chaque partie avec un nombre limité de joueurs (disons deux).
Comme la dernière fois, lancez une partie à la connexion du premier joueur, c'est-à-dire, créez un *fil de partie*. Cependant le joueur ne peut commencer à deviner et doit attendre jusqu'à ce qu'un deuxième joueur se connecte. Alors, vous associez le nouveau joueur à la même partie, et le jeu commence, et se déroule comme dans l'exercice précédent...
... jusqu'à la connexion d'un troisième joueur, qui se trouve dans la situation du premier : vous lui créez un nouveau fil de partie et il doit attendre un opposant.
Et ainsi de suite, pour chaque paire de joueurs qui se connectent.