Rapport de conception projet de réalisation d'un serveur de discussion Java M3102

Ce projet a pour but de réaliser un serveur de discussion où plusieurs clients *telnet* viendront s'y connecter pour ainsi communiquer entre eux. Par le fait que ce soit des clients *telne*, cela implique l'utilisation du protocole de couche quatre du modèle *OSI*, le protocole *TCP/IP*. De cette manière, cette production nous amène à utiliser et à implémenter les *Threads*, les sockets *TCP* et les flux d'entrée et de sortie depuis les sockets. Pour permettre la mise en œuvre de ce projet, sous le langage Java, nous avons conçu trois classes qui sont :

- Serveur.java
- ClientThread.java
- Message.java

Les deux premières classes citées vont venir implémenter l'interface *Runnable* et ainsi elles vont permettre l'utilisation des *Threads* et de leur méthode *run().* La classe Serveur va venir jouer le rôle du serveur et va venir instancier les *Threads* clients au fur et à mesure des connections *telnet* acceptées. Ensuite c'est les objets *ClientThread* qui vont faire le lien entre le client et le serveur. Quant à la classe Message, elle va contenir le nom du client et le contenu de son message. C'est cette classe qui après réception d'un message par un *ClientThread*, va demander aux autres *ClientThread* de redistribuer à leurs clients *telnet* respectifs. Cette classe sera instanciée lorsque qu'un client se sera connecté au serveur et aura donné son nom. De cette manière chaque *ClientThread* aura son propre objet Message qu'il manipulera au fur et à mesure que son client envoi des messages. Cependant, *ClientThread* manipulera un objet Message qui n'est pas le sien lorsqu'il devra redistribuer à son client le message envoyé par un autre client.

Focus sur les différentes classes :

Serveur.java :

C'est cette classe qui contient la méthode *main* donc c'est cette classe qui permettre l'exécution du serveur. Par l'implémentation de l'interface *Runnable,* le serveur va pouvoir créer un *Thread* qui dans une boucle infinie, va attendre les demandes de connexion provenant des clients *telnet.* Pour cet exercice, l'adresse est locale à la machine (soit 127.0.0.1) et le port utilisé est 2042. Lorsqu'un client est accepté depuis le port 2042, le serveur ajoute ce client à la liste *clients* qui contient les *ClientThread* actuellement connectés, créé un nouveau *Thread* pour ce nouveau client et démarre son exécution par le biais de la méthode *start*.

Le *Thread* du serveur va passer par plusieurs états. Lors de sa création, il sera à l'état *NEW* puis comme c'est le seul *Thread*, il passera directement à l'état *RUNNING*. Enfin il passera à l'état *WAITING* lorsqu'il attendra l'arrivée d'un nouvelle connexion grâce à la nature de la méthode *accept()* qui est bloquante. C'est de cette manière que l'ordonnanceur *Java* va pouvoir passer la main aux *ClientThread* pour permettre la communication entre les différents clients connectés à ce serveur.

ClientThread.java :

Comme précédemment dit, cette classe sera instanciée lorsqu'une connexion est acceptée. A sa création, elle va initialiser sa variable *Socket* qui va contenir les informations le reliant au client correspondant. Elle va aussi initialiser les *Buffer* de lecture et d'écriture et leurs indiquant d'où transitent les informations qui sont respectivement les flux d'entrée et de sortie du S*ocket.* Après l'initialisation de ce *Thread* (à ce moment précis, il est à l'état *NEW*) durant l'exécution de la classe *Serveur*, c'est dans cette même classe que ce *Thread* va passer à l'état *RUNNING* par le biais de l'appel de la méthode *start()*. Tout comme pour le *Thread* jouant le rôle de serveur, l'objet *ClientThread* va passer à l'état *WAINTING* lorsqu'il va rencontrer la méthode bloquante *readLine()* de l'objet *fluxIn*, pour attendre l'arrivée d'un message de son client, auquel ce *ClientThread* est rattaché.

Lorsque le client aura répondu à la question envoyée par le *ClientThread* qui demande son identifiant, il va enregistrer son nom puis instancie son objet *Message*. Ensuite le *ClientThread* va attendre l'arrivée de nouveaux message provenant de son client pour les enregistrer au sein de son objet *Message* qui se chargera de le redistribuer aux autres clients. Enfin lorsque le client envoi le message « bye », il va se déconnecté du serveur. C'est avant la fin d'exécution du *ClientThread* que sera appellée la méthode *deconnexion()* qui viendra fermer la socket et les flux qui lui sont rattachés. De plus, il va se retirer de la liste *client*s qui appartient au *Serveur.*

Message.java :

C’est cet objet qui va contenir le nom et le contenu du message du *ClientThread* auquel il est rattaché. Il ne sera manipulé qu’uniquement par ce dernier. Lors de sa création et avant la fin du processus *ClientThread* (déconnexion), l’objet *Message* va avoir comme valeur pour son champ *nomFrom* « System ». De cette manière, les autres clients pourront voir qu’une action de connexion ou de déconnexion d’un client est en cours. Entre ces deux phases, l’objet va contenir pour son champ *nomFrom*, le nom du client auquel il est rattaché et pour le contenu des différents messages écrit par ce même client.

Lorsqu’un message est arrivé au niveau de *ClientThread*, il va enregistrer le contenu au sein du champ correspondant de son objet *Message*. Et va ensuite appeler la méthode *sendToAll()* de cet objet pour permettre la redistribution du message aux autres clients, excepté le client auteur du message. Cette redistribution est permise grâce à l’appel de la méthode *send()* de tous les *ClientThread* contenus dans la liste *clients* présente dans la classe *Serveur.*

Réalisé par Valentin Hebert et Jason Liebault