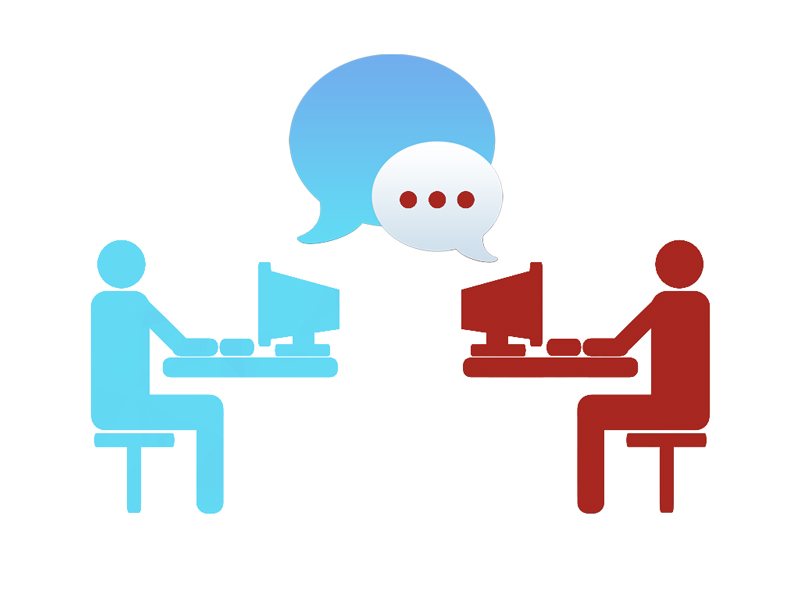


**Rapport de projet POO/COO**

Le projet Clavardage



**Réalisé par :**

LAKHAL Mohammad Amine

JIN Yuheng

(Groupe 4IR B1)

**Sommaire**

**I-Introduction...............................................................3**

**II- Répartition des tâches.............................................5**

**III- L’architecture du système développé ...................6**

**IV- Les fonctionnalités du programme ......................10**

**V- Les tests unitaires…………………..........................11**

**VI- Conclusion............................................................15**

**I- Introduction**

Au cours de ce projet de POO, nous avons dû implémenter un système de communication en temps réel qui permettrait à un groupe d’individus de communiquer entre eux (en mode “chat”, entre 2 personnes) ou à plusieurs (“groupchat”).

La première phase de notre cahier de charge consistait donc à créer une application distribuée incluant un système de clavardage basé sur TCP ainsi qu’un dispositif de mise à jour des listes de users connectés en ligne basé sur UDP.

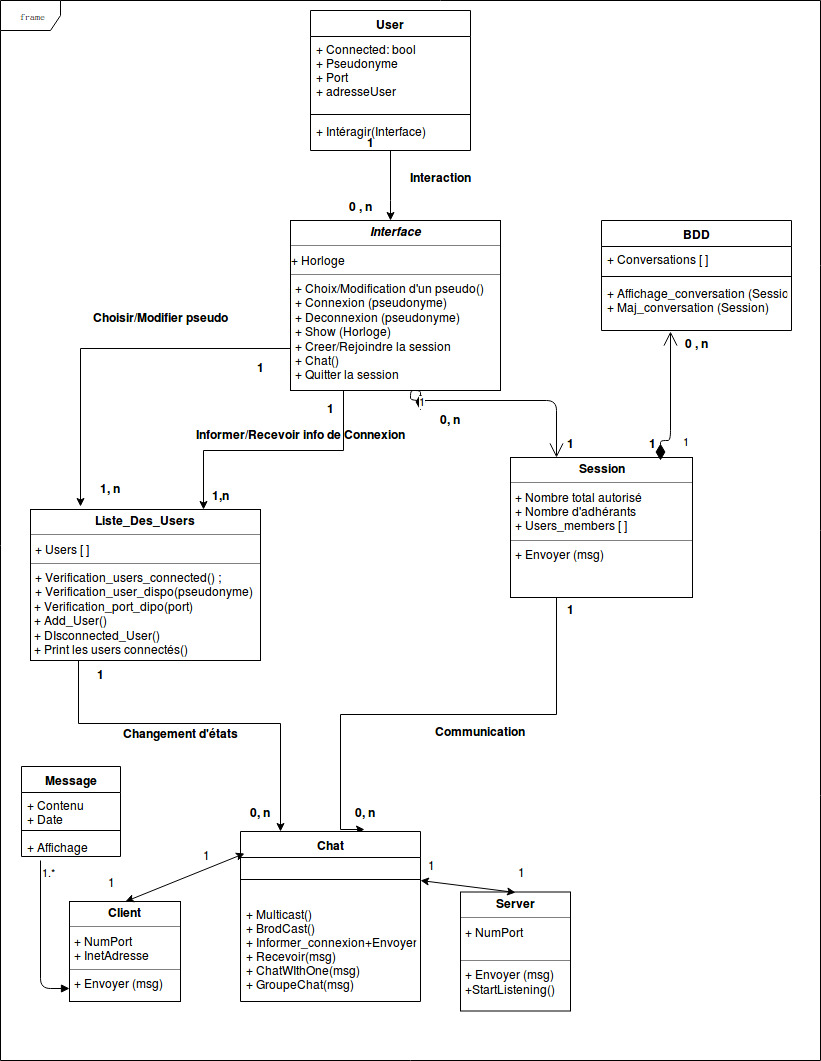
La deuxième phase de notre cahier de charge avait pour but de construire un serveur de présence pour contrôler la présence des users et la validation de leur pseudo, et utiliser les BDD pour enregistrer les historiques en locale. Ceci est basé sur un concept centralisé afin d’alléger et améliorer notre système.

Tous les fichiers sont disponible sur notre GitHub: <https://github.com/YuhengJin/ChatApplication>

Et tous nos documents de conception UML dans le répertoire suivant:

https://drive.google.com/drive/folders/1P1AZ4hEQHHZy4D-3YqoOl2jS6Ry4ZPBS?usp=sharing

Voici notre diagramme de classe, dans la page suivante:



**II- Répartition des tâches**

Afin de pouvoir commencer à coder les méthodes nécessaires au fonctionnement de nos algorithmes, il a d’abord fallu faire un travail de recherches, nous avons analysé le travail demandé, grâce à des diagrammes de Classes, de séquences et de UML. Durant le travail sur cette partie, nous avons mieux compris les contraintes pour avoir une vue plus globale de notre projet. Après, nous avons réparti les tâches. Il fallait commencer par travailler sur la partie de communication, spécifiant la structure de notre système de multi-thread, lancer des tests unitaires au cours de notre travail.

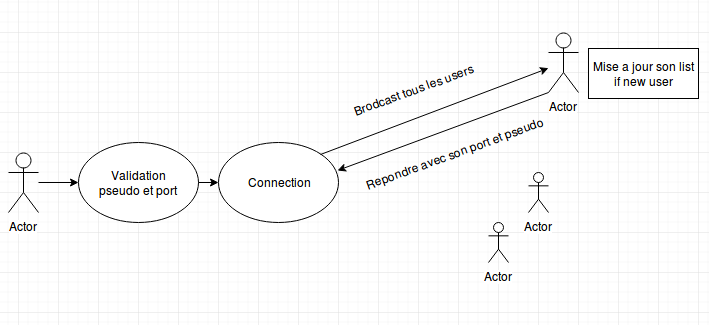
Au départ, nous avons préféré laisser l'implémentation de la BDD pour la fin, et avons commencé à travailler sur le reste.

**III- L’architecture du système développé**

**1- Partie connexion:**

1. Récupérer les informations insérées par l’utilisateur (pseudo et des ports)
2. Brodcast à tout le monde des trames contenants ces informations pour voir si cet utilisateur a le droit de se connecter, puis mise à jour des liste de personnes connectés de tout les utilisateurs (à l’aide de broadcast encore une fois).

(cette vérification sera détaillé dans la partie suivante)



**2- Partie inscription/authentication:**

Afin de pouvoir communiquer avec ses collègues, 2 choix s’offrent à l’utilisateur:

S’inscrire, ou bien se connecter:

* Inscription: L’utilisateur demande aux utilisateurs connectés si le pseudo et le port qu’il a choisi ne sont pas déjà utilisés, à l’aide d’une trame “inscription”. Pour cela, les utilisateurs comparent ces valeurs à celles des utilisateurs déjà présent dans leurs BDD\_users, qui sont simplement des BDD virtuel sous la forme d’une liste qu’ils mettent à jour à chaque fois qu’ils reçoivent un Broadcast d’une personne qu’ils ne connaissent pas déjà.

Si ce pseudo/port n’est pas déjà utilisé, alors la personne réceptrice envoie une trame contenant ses coordonnés, sinon elle envoie une trame spéciale (“samename” ou “sameport” ou autre). Ainsi la personne voulant s’inscrire peut s’avoir si son pseudo/port est déja utilisé si elle réceptionne une telle trame.

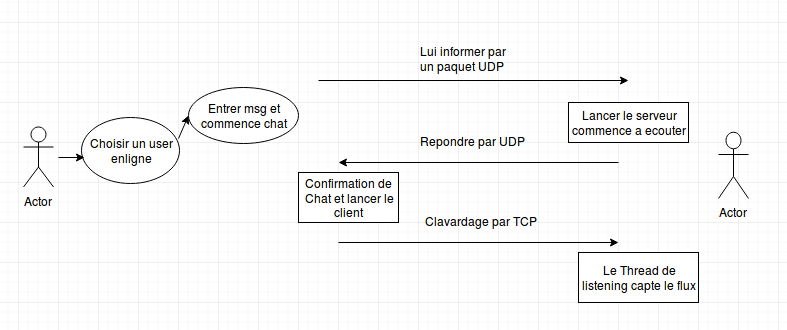
* Authentification: C’est un système assez similaire au système précédent d’inscription. La BDD\_users d’un utilisateur contient la liste des utilisateurs inscrits avec leurs coordonnées, ainsi que leurs mot de passe. Ainsi quand une personne voudra se connecter à l’aide d’une trame “connexion”, la personne réceptrice va chercher dans sa BDD si un utilisateur avec le pseudo donné existe. Si il n’existe pas alors il envoie une trame “utilisateur inexistant” et si il existe alors il vérifie si le mot de passe reçu est même que dans le user qu’il a trouvé. Sinon, il envoie une autre trame d’erreur.

Dans les 2 cas, on a besoin de créer une classe “chat” à la personne connecté à l’interface afin de pouvoir communiquer avec les autres utilisateurs. Si la personne ne reçoit pas de trame d’erreur (que ce soit suite à une tentative de connexion ou inscription), alors elle garde cette classe chat. Si elle reçoit une trame d’erreur, alors on réinitialise cette classe chat (afin de libérer toutes les ressources qu’on lui a mobilisé tel que les sockets et les coordonnées) et on lui recrée une nouvelle classe chat. Cela évite donc de se retrouver avec 2 users “toto”, l’original puis la personne qui a tenté de s’inscrire en tant que “toto”.

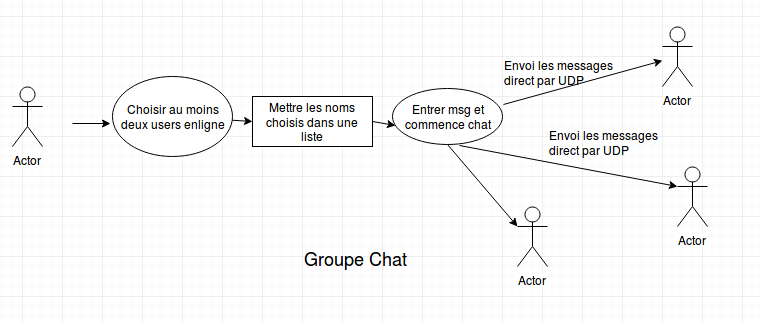
Bien entendu, cela n’est pas visible par les utilisateurs, car si on prend l’exp précédent, “toto” n’ajoute pas le nouvel utilisateur ni dans sa Bdd de users inscrit, ni dans sa Bdd de users connectés. Et la personne qui a tenté de se connecter ne sait même pas qu’on lui crée pendant l’espace de quelques milliseconde une classe chat qui a bien contenu comme identifiant “toto”.

**3- Partie communication:**

1. Le clavardage 1 à 1 est implémenté par TCP pour garantir la fiabilité et alléger la tension de flux de notre système. Le concept de TCP est repris le concept de Client/Server



1. Le clavardage du groupe et l’échange des informations (comme informer le user choisi que je vais dialoguer avec lui) sont développés par UDP, puisque on devrait garantir l’efficacité de tout le système, et les informations transmis est moins important par rapport le dialogue 1 à 1, du coup on a choisi UDP ce genre de protocole de non connection.

****

**4- Partie Modification de pseudo:**

Avant de pouvoir changer son pseudo, l’utilisateur doit procéder à une petite vérification, il doit entrer son ancien pseudo, le nouveau ainsi que le nouveau. Donc déjà cela fait quelques vérifications en interne avant de pouvoir valider le nouveau pseudonyme auprès des utilisateurs connectés. Ainsi, son ancien pseudo et le mot de passe doivent correspondre à ceux de l’utilisateur connecté et le nouveau pseudo doit être différent de l’ancien. Si ce n’est pas bon, une fenêtre d’erreur personnalisé s’affiche. Sinon, une trame en broadcast “modifpseudo” est envoyé aux personnes connectés afin qu’ils puissent vérifier dans leurs BDD users si il n’y a pas déjà de personne avec un pseudo similaire. C’est le même principe que lors de l’inscription d’une personne. Si la modification est validé, alors on met à jour la classe chat et user de la personne concerné à jour, ainsi que l’interface qu’il est. De plus, on met aussi la liste des users connectés et BDD users des personnes qui reçoivent cette trame (si c’est validé). Etant donné que toutes les personnes déjà connectés ont les memes liste users connectés et Bdd users, car elles sont mis à jour à chaque broadcast, alors une personne ne risque pas d'être validé par un utilisateur et refusé par une autre.

**IV- Les fonctionnalités de l’application**

Cette application offre donc la possibilité à une personne de discuter avec une autre autre personne, ou bien avec plusieurs personnes dans le cadre d’une session, dans le réseau local, comme nous l’impose le cahier des charges.

Ceci est notamment possible à l’aide d’une interface à la fois très simple d’utilisation mais qui contient aussi tous les éléments nécessaires. Il y a 3 fenêtres (dans le sens “panel” de Java), une pour l’inscription/authentification, une la communication, et une dernière pour modifier son pseudo.

Etant donné que la communication est en TCP sans serveur centralisé, alors les utilisateurs se doivent de communiquer entre à chaque connexion ou déconnexion car ils ne peuvent pas s’adresser directement à un serveur. Ainsi, il en découle un premier inconvénient qui est la présence obligatoire systématiquement d’au moins une personne connecté qui contient la BDD des personnes inscrites. Si cette personne se déconnecte aussi, on ne saura pas qui est inscrit car de 1 il n’y a pas de serveur toujours connecté en toute circonstance, et cet éventuel serveur ne pourrait pas disposer d’une réelle BDD physique (type “sql”) car nous somme partit sur une autre méthode de stockage des personnes inscrites.

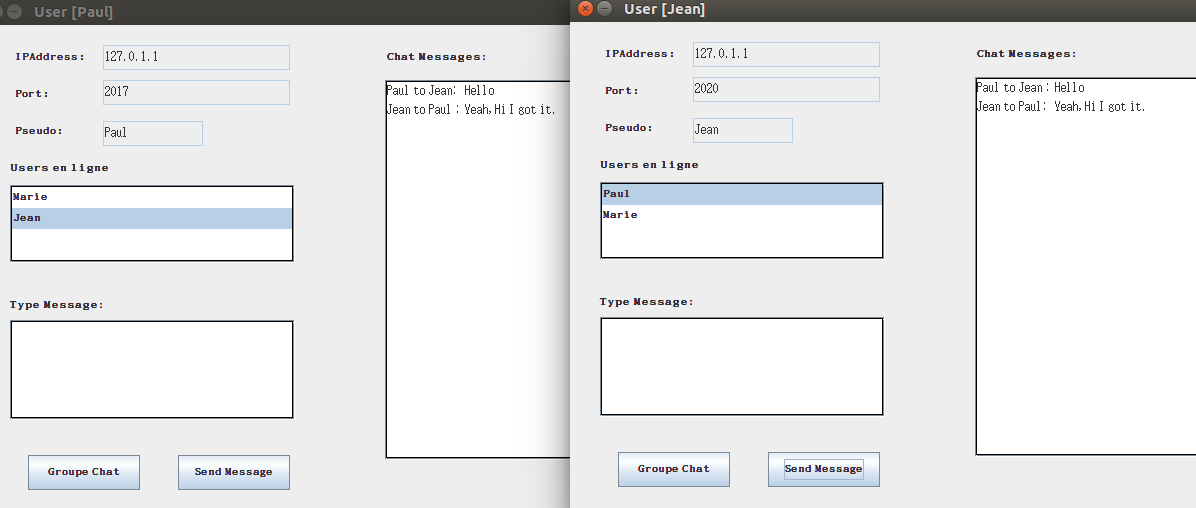
Le 2e inconvénient lié à cette méthode évoquée précédemment est que chaque utilisateur se doit d'être indépendant et de connaître toutes les personnes déjà inscrites. S’il y a un nombre moyen voire grand d’utilisateurs (et dans nos tests aussi), cela ne pose absolument pas de problème. Peut être que si l’on prend un nombre d’utilisateurs infiniment grand, l’espace en mémoire occupé par cette liste deviendrait problématique, surtout que cela concernerait tous les utilisateurs (et non pas un seul serveur qui aurait cette BDD).

Sinon ces fonctionnalités d’inscription/authentification sont tout à fait fiables, et notamment grâce aux fait que tous les utilisateurs y contribuent et se mettent à jour très régulièrement.

Ca en est de même pour la modification de pseudo qui suit le même schéma. Un utilisateur peut changer son pseudo à tout moment en demander d’abord aux autres personnes si le pseudo est libre. Puisque chaque personne connaît les personnes inscrites, même si la personne désirant modifier son pseudo est la seule connectée, cela ne pose pas de problème.

**V- Les tests unitaires**

**1- Communication 1 à 1 sur le même ordinateur:**



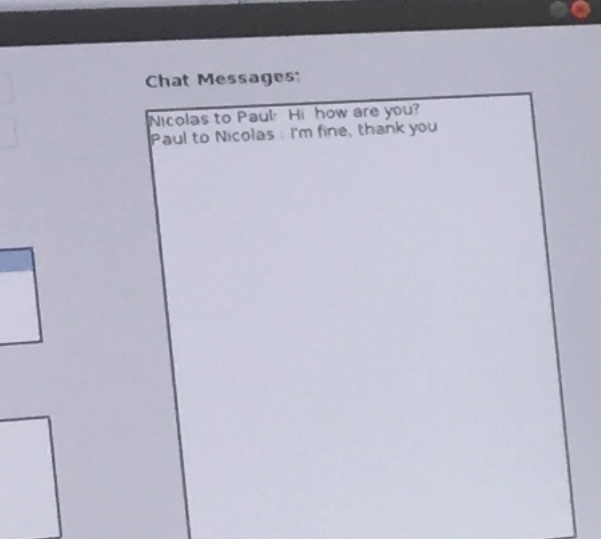
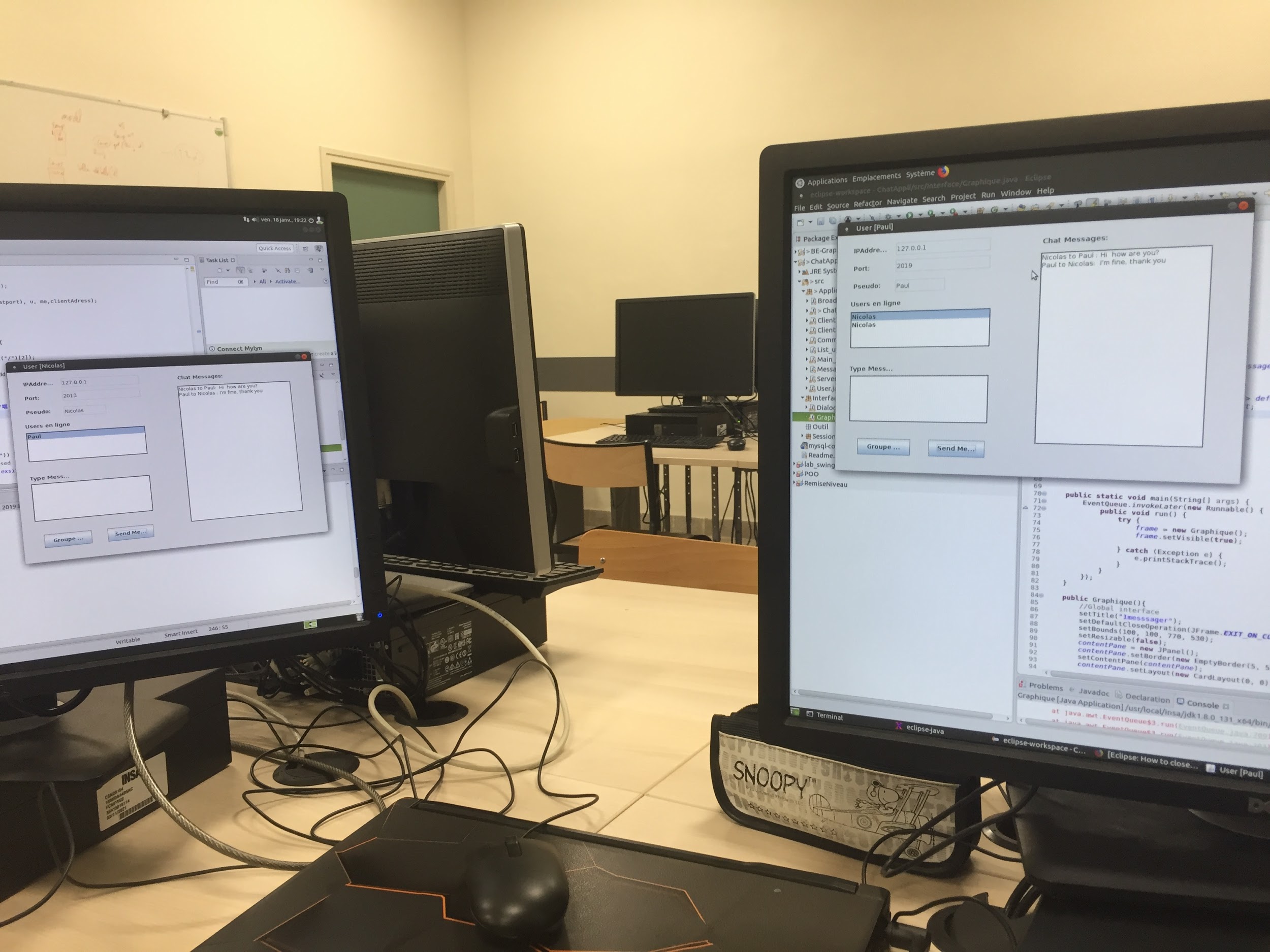
On voudrait simuler le scénario où Paul parle à User Jean et Jean lui répond, c’est une discussion point à point et en temps réel. C’est implémenté par le protocole TCP.

Et vu qu’on a diffusé différents ports pour chaque utilisateur, donc toute la discussion peut marcher avec la même IP adresse mais avec différents ports.

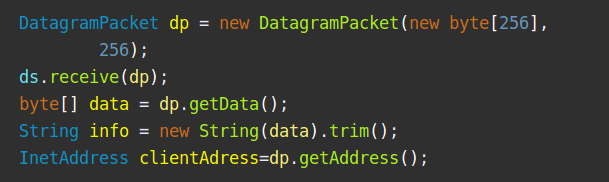
**2- Le cas où on a oublié de choisir un user:**

****

Ici c’est le troisième user Marie qui voudrait se connecter mais elle a oublié de choisir un user dans la liste, du coup il y aura un Warning.

**3- Communication 1 à 1 sur différentes machines dans le même réseau local (au GEI):**

Ici c’est User Nicolas qui essaye de rejoindre User Paul qui est sur une machine distincte et le dialogue marche très bien.

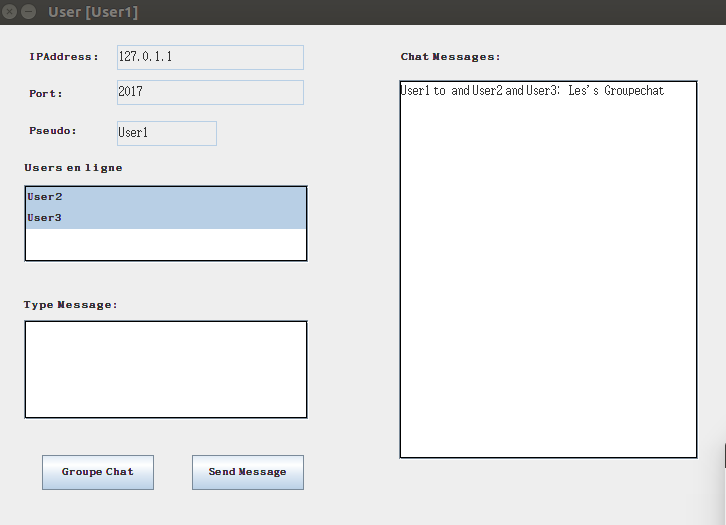
****

Quand on reçoit le broadcast de new user qui vient de connecter, on a enregistré son DatagramPacket, et grâce à la méthode:

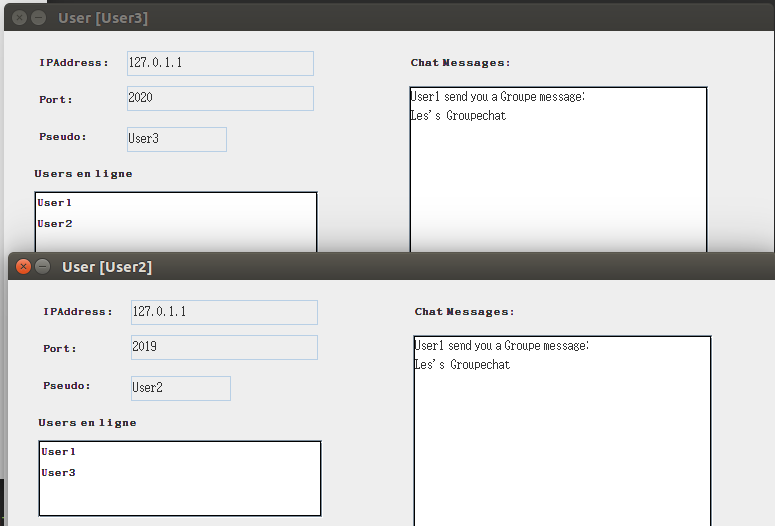
****

On pourrait savoir l’adresse IP de ce new user, et puis on l’enregistre dans notre liste des users. Donc notre communication est non seulement basé sur le numéro de port et aussi de l’adresse IP correspondant.

**4- Groupe chat:**

****

Dans ce cas, on voudrait lancer un GroupeChat, c’est la sénario que User1 qui voudrait parler avec User2 et User3 en même temps.

****

Comme ce qu’on voit, User 2 et User 3 peuvent tous très bien receoir ce groupe message qui vient de User 1.

GroupeChat peut très bien marcher avant ce qu’on a implémenté le BDD, mais dans la version actuelle, il y reste de petits bugs.

**VI- Conclusion**

En conclusion, nous avons pu voir l’utilité de plusieurs fonctionnalités de Java telles que la communications à l’aide de Sockets et Thread. Nous avons autant utilisé l communication en UDP pour le Broadcast en cas de connexion/déconnexion que la communication TCP lors d’échanges entre 2 personnes. Durant ce projet nous d’abord maîtrisé le TCP pour nos 1ers échanges entre utilisateurs puis nous avons surtout utilisé UDP pour créer des groupes en Multicast entre utilisateurs, et aussi pour le Broadcast.

Nous avons aussi maîtrisé la conception d’interface graphique à l’aide de Java Swing, que ce soit boutons, textes, cliques, fenêtres, listes, interactions, etc.

C’est surtout à l’aide de ces 2 principaux outils que nous avons pu mener à bien ce projet. On aurait bien voulu mieux exploiter un 3ème point qui le JDBC pour pouvoir y stocker toutes les personnes inscrites ainsi que tous les messages envoyé afin qu’ils soient accessibles à tout moment pour un utilisateur donné quel que soit le poste où il se trouve. Finalement nous avons opté pour une autre méthode pour gérer les utilisateurs qui fonctionne aussi bien, et nous avons donc laissé tomber le stockage de messages. Peut être qu’avec un peu plus de temps on aurait pu intégrer cette fonctionnalité qu’on avait déjà commencé à exploiter.