# Application de chat multi-thread en Java

Ahmed Lounis

# Cahiers des charges

Développer une application de chat Client/Serveur en sockets permettant d'organiser une discussion publique entre un ensemble de participants. La discussion dans cette application sera affichée dans la console.

## Côté serveur :

### Classe de serveur :

L'implémentation du serveur principal est simple et similaire à ce que vous avez vu en cours. Les points suivants vous aideront à comprendre l'implémentation du serveur :

1. Le serveur exécute une boucle infinie pour continuer à accepter les demandes entrantes (reste bloqué).
2. Lorsqu'une demande de connexion arrive, le serveur stocke l'objet socket nouvellement connecté dans un tableau "clients" et il lance un thread qui intercepte tout message envoyé dans cet objet socket récemment stocké.
3. Lorsque le serveur reçoit un message dans l'un des sockets de communication, il récupère ce message puis il le diffuse sur l'ensemble des objets socket stockés dans le tableau "clients"
4. Dans le cas où le serveur reçoit un message "exit" dans l'un des objets socket correspondant à l'un des client (Client X par exemple), il diffuse le message suivant "l'utilisateur X a quitté la conversation" sur le reste des clients connectés et libère la socket, les flux d'E/S et le thread associé au client.

## Côté client :

L'implémentation du Client consiste tout d'abord à effectuer une connexion auprès du serveur. Puis, le client envoi son pseudonyme à travers le socket de communication créé. Dans l'implémentation côté client, nous vous conseillons d'implémenter deux threads. Le premier est utilisé pour intercepter les messages venant du serveur et le deuxième est utilisé pour récupérer les messages saisis par l'utilisateur et de les transmettre au serveur. Ainsi, pour développer le côté client il faut :

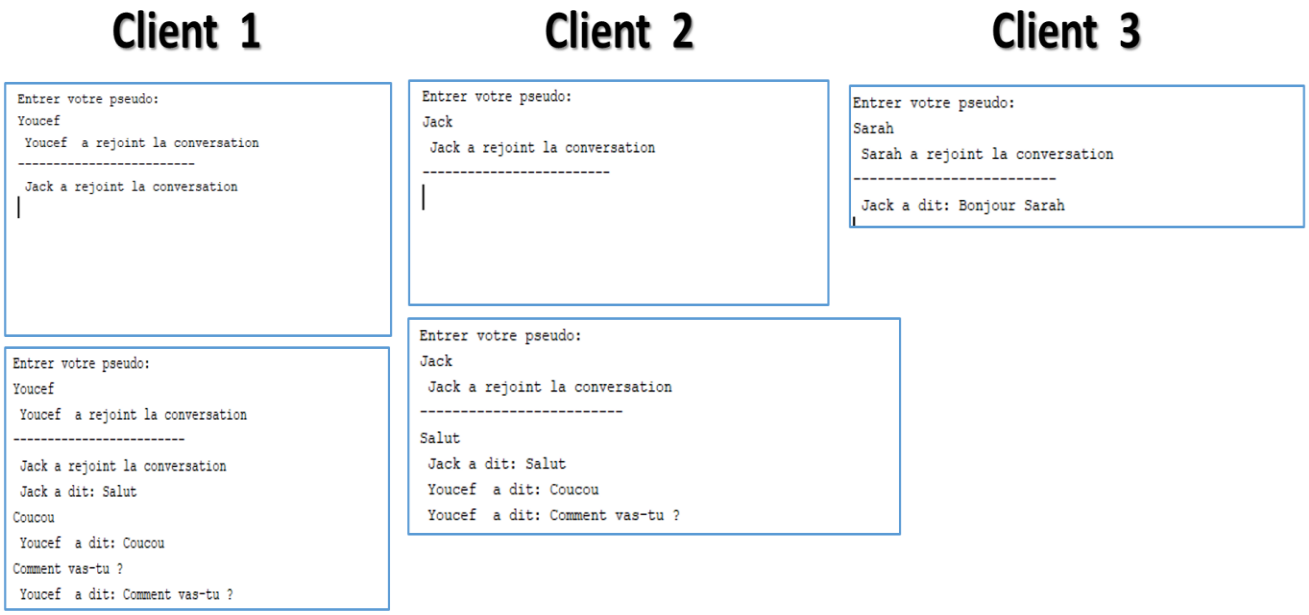
1. Établir une connexion socket,

2. Passer le Socket de communication à un thread qui va intercepter les messages envoyés par le serveur puis les afficher dans la console.
3. Lancer un thread qui intercepte les messages saisis par l'utilisateur et les envoient au serveur à travers la socket de communication.

**Questions :**

- Comment garantir qu'un pseudonyme est unique ? Implémenter la solution proposée.
- Comment vous faites pour gérer le cas d'une déconnexion de client sans que le serveur soit prévenu et vice-versa ? Implémenter la solution proposée.

Voici un exemple d'exécution de l'application :




# Livrables et date limite

***Date limite : 04/04/2021***

Vous devez déposer sur moodle un seul fichier zip (NomsBinomesDevoir1.zip) avec deux livrables :

**Code sources : un fichier codesource.txt qui contient un lien vers votre dernière commit sur le Gitlab de votre projet devoir**. *N'oubliez pas de donner les droits nécessaires (maintainer) à Lounis Ahmed.*

- Il faut que votre code soit bien organisé et lisible. Voici quelques conseils.
    - Encapsulation et polymorphisme : penser aux constructeurs possibles, utiliser les getters, setters et aux redéfinitions des méthodes (ex.toString).
    - Indentation. Indentation automatique via le raccourci Ctr + Shift +F sous Eclipse et Alt+Shift+F sous NetBeans.
    - Les noms des classes doivent être bien formatés comme MyClassJava.
    - Créer des packages pour organiser vos classes (clientPckage, serverPackage),
    - Les noms des variables commencent par minuscule et ont un sens (ex. nbClientsConnectes)
    - Les noms des fonctions commencent par minuscule et contiennent un verbe (ex. envoyerUnMsg),
- Ajouter des commentaires pour :
    - Décrire une classe.
    - Décrire une fonction (paramètres, return, etc).
    - Décrire un bloc d'instructions.
    - Il est appréciable d'utiliser les annotations de JAVADOC.

**Rapport : uniquement les fichiers PDF sont acceptés.** Dans le rapport, il faut avoir :

- Une partie contexte dans laquelle vous expliquer le projet, les concepts, les objectifs fixés et les cas d'utilisation théorique (schémas)
- Une partie qui explique et justifie votre conception et le développement (ex. Le choix des structures des données JAVA, surcharges des méthodes). Vous pouvez vous appuyer sur des bouts de code et des schémas.
- Une partie scénarios qui explique comment récupérer (ex. code sur git avec un commit), installer et lancer votre application. De plus, vous devez montrer tous les cas d'utilisations pratiques (via votre application sous forme d'un démonstrateur)

# Annexe 1 : les threads en JAVA

L'une des façons pour manipuler les objets Thread dans java, se fait par la déclaration d'une classe qui hérite de la classe Thread, comme suit :

```java
public class MessageReceptor extends Thread{

    private Socket client;
    public MessageReceptor(Socket client) {
        this.client=client;
    }

    @Override
    public void run() {
        try {
            InputStream ins=client.getInputStream();
             while(true){
                // lecture du contenu du InputStream

             }
        } catch (IOException ex) {
            Logger.getLogger(MessageReceptor.class.getName()).log(Level.SEVERE, null, ex);
        }

    }

}
```

**Remarque :**

**La méthode run() définit le traitement que le thread va effectuer et donc il est important de l'implémenter**

Une fois que la classe est définie, il faut tout d'abord instancier des objets de cette classe comme suit :

```java
Socket client=new Socket("localhost",9000);
MessageReceptor msgreceptor=new MessageReceptor(client);
```

L'objet instancié a les mêmes propriétés qu'un Thread, notamment l'exécution en parallèle avec le programme principale ou d'autres thread ainsi que l'accès à tous les objets et variables définis dans le programme appelant. Cependant, une instanciation d'un objet détenant les mêmes propriété qu'un Thread, ne veut pas dire que le traitement défini dans sa méthode "run" est lancé, et donc pour lancer un objet de type Thread, il est obligatoire de faire appel à la méthode start(), comme suit:

```java
msgreceptor.start();
```