

UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

---

# Rapport Devoir AI13

## Application de chat multi\_thread

---

Jana EL RAFEI  
Yousra HASSAN  
Ikram DAMA

21 septembre 2024

# Table des matières

<b>Contexte du projet</b>	<b>1</b>
1 But du projet . . . . .	1
2 Concepts Clés . . . . .	1
3 Objectifs Fixés . . . . .	1
4 Cas d'utilisation . . . . .	2
4.1 Use Case Diagram . . . . .	2
4.2 Description des cas d'utilisation . . . . .	3
<b>Architecture et développement</b>	<b>4</b>
1 Architecture . . . . .	4
2 Développement . . . . .	4
2.1 Serveur . . . . .	4
2.2 Client . . . . .	5
2.3 ClientHandler . . . . .	5
3 Logique d'opérations . . . . .	5
<b>Scénarios d'utilisation de l'application</b>	<b>6</b>
1 Récupération du code source . . . . .	6
2 Lancement de l'application et démonstration des cas d'utilisation . . . . .	6

# Partie 1

## Contexte du projet

### 1 But du projet

Le projet consiste à développer une application de chat en réseau utilisant des sockets en Java. L'objectif principal est de créer un système de communication entre plusieurs clients connectés à un même serveur. Chaque client peut envoyer et recevoir des messages en temps réel, facilitant ainsi les échanges simultanés. Ce type de projet s'inspire des besoins croissants de communication instantanée dans des environnements distribués, tels que les applications de messagerie, les forums de discussion en ligne etc..

### 2 Concepts Clés

- **Architecture Client-Serveur** : Le projet repose sur l'architecture réseau classique client-serveur. Le serveur est un programme central qui écoute les connexions des clients, gère la communication entre eux et assure le bon fonctionnement du système. Les clients sont des programmes qui se connectent au serveur pour échanger des messages.
- **Sockets** : Les sockets sont des mécanismes permettant la communication bidirectionnelle entre deux entités (le client et le serveur, dans ce cas) sur un réseau. Le serveur utilise un socket pour accepter des connexions entrantes, tandis que chaque client en utilise un pour se connecter au serveur.
- **Multi-threading** : Le serveur est capable de gérer plusieurs connexions simultanées en allouant un thread à chaque client. Cela permet à plusieurs clients d'interagir sans attendre que le serveur soit libre pour traiter leurs requêtes.

### 3 Objectifs Fixés

- **Créer un serveur multi-client** : Le serveur doit pouvoir accepter plusieurs connexions simultanées et gérer la communication entre les clients de manière fluide et sans conflit.
- **Assurer l'unicité des noms d'utilisateur** : Chaque client doit entrer un nom d'utilisateur unique. Si un client entre un nom déjà utilisé, le serveur lui demandera de fournir un nouveau nom.
- **Diffuser les messages** : Lorsque l'un des clients envoie un message, ce dernier doit être transmis à tous les autres clients connectés, sauf à celui qui l'a émis. Cela simule un environnement de chat de groupe.
- **Gérer les déconnexions** : Le serveur doit détecter et gérer les déconnexions des clients, libérer les ressources associées, et informer les autres clients de la déconnexion.

## 4 Cas d'utilisation

### 4.1 Use Case Diagram

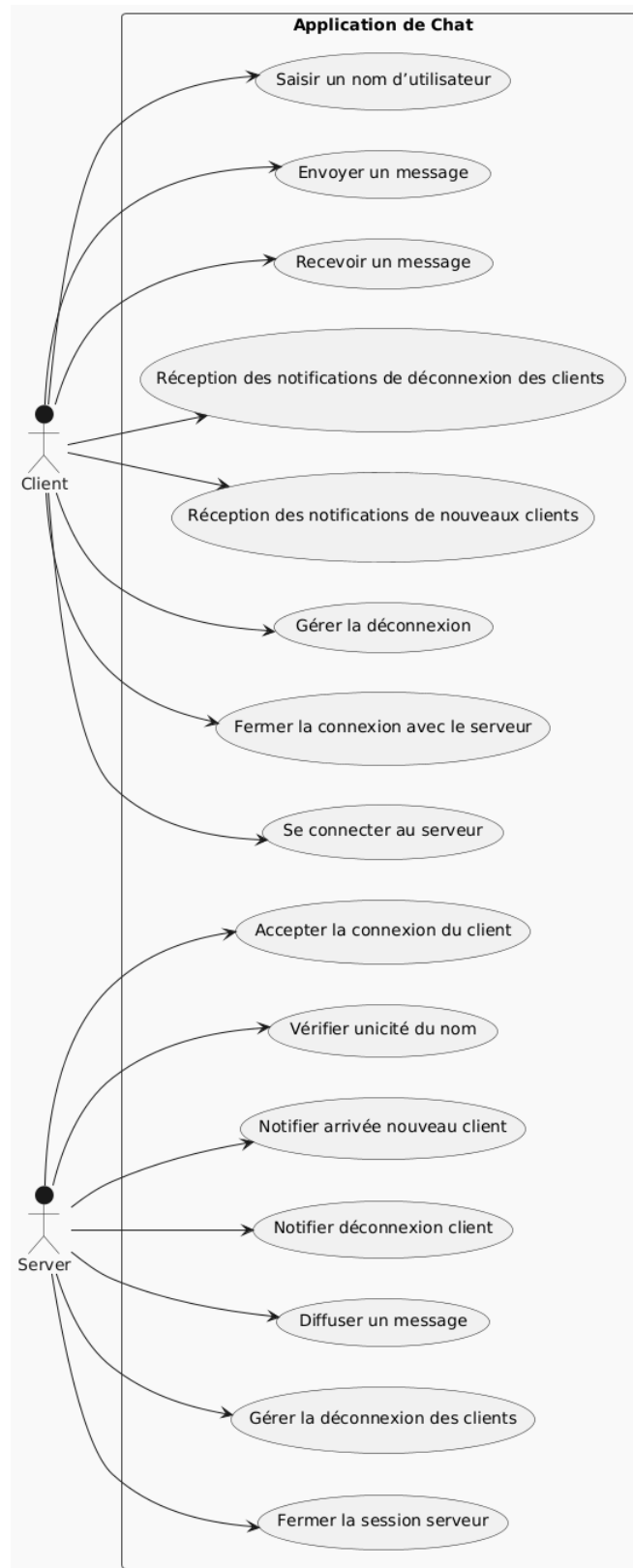


FIGURE 1 – Diagramme de cas d'utilisation

## 4.2 Description des cas d'utilisation

- **Accepter la connexion d'un client** : Le serveur écoute les connexions des clients sur un port spécifique et accepte les nouvelles connexions.
- **Vérifier l'unicité des noms d'utilisateurs** : Lorsque le client fournit un pseudonyme, le serveur vérifie que celui-ci est unique parmi les utilisateurs connectés.
- **Diffuser un message** : Lorsqu'un client envoie un message, le serveur diffuse ce message à tous les autres clients connectés.
- **Notifier l'arrivée d'un nouveau client** : Le serveur diffuse un message aux autres clients pour indiquer qu'un nouveau client s'est connecté.
- **Gérer la déconnexion des clients** : Lorsqu'un client se déconnecte, le serveur libère les ressources associées (fermeture des sockets) et informe les autres clients de la déconnexion.
- **Notifier la déconnexion d'un client** : Le serveur diffuse un message aux autres clients pour indiquer qu'un client s'est déconnecté.
- **Fermer la session serveur** : Le serveur peut éventuellement fermer la session, interrompant ainsi toutes les connexions actives avec les clients.
- **Se connecter au serveur** : Le client initie une connexion avec le serveur en utilisant l'adresse et le port du serveur.
- **Saisir un nom d'utilisateur** : Le client fournit un pseudonyme pour s'identifier auprès du serveur.
- **Envoyer un message** : Le client saisit un message qui est envoyé au serveur pour être diffusé aux autres clients.
- **Recevoir un message** : Le client reçoit des messages provenant du serveur, qui sont envoyés par d'autres clients.
- **Gérer la déconnexion** : Le client peut se déconnecter volontairement, ou la déconnexion peut être causée par une perte de connexion. Il peut également gérer les notifications de déconnexion d'autres utilisateurs.
- **Réception des notifications de nouveaux clients** : Le client reçoit des notifications lorsqu'un nouveau client rejoint la session.
- **Réception des notifications de déconnexion des clients** : Le client reçoit des notifications lorsqu'un client se déconnecte de la session.
- **Fermer la connexion avec le serveur** : Le client ferme sa connexion avec le serveur lorsqu'il souhaite quitter la session de chat.

# Partie 2

## Architecture et développement

### 1 Architecture

Pour répondre aux objectifs fixés, nous avons développé un code qui opère autour de trois classes centrales :

- **Serveur :**
  - Attributs
    - `private static final int PORT = 1234;`
    - `protected static final Set<ClientHandler> clients = Collections.synchronizedSet(new HashSet<>());`
- **ClientHandler :**
  - Attributs
    - `private final Socket clientSocket;`
    - `private PrintWriter out;`
    - `private BufferedReader in;`
    - `private String username;`
  - Méthodes principales (hors *main*)
    - `public void run(); (@override)`
    - `private void verifyUsername(String name);`
    - `private void envoyerMessage(String message, boolean newClient);`
    - `private void disconnectHandler();`
    - `private void diffuserMessage(String message, ClientHandler exclureClient, Boolean newClient);`
- **Client :**
  - Attributs
    - `private static final String SERVER_ADDRESS = "localhost";`
    - `private static final int SERVER_PORT = 1234;`
    - `private Socket socket;`
    - `private BufferedReader input;`
    - `private PrintWriter output;`
  - Méthodes principales (hors *main*)
    - `private void start();`
    - `private class ThreadListener; public void run();`
    - `private class UserInputHandler; public void run();`
    - `private synchronized void closeEverything();`

### 2 Développement

#### 2.1 Serveur

Le serveur est conçu pour accepter plusieurs connexions de clients, en attribuant un thread distinct pour chaque client connecté via la classe `ClientHandler`. Le serveur maintient un ensemble de clients connectés et gère les communications entre eux.

- Le serveur écoute sur un port (ici, le port 1234) pour accepter les connexions entrantes.
- Les clients connectés sont stockés dans un ensemble synchronisé (`synchronizedSet`) afin de garantir la sécurité des threads lors de l'ajout et de la suppression de clients.
- Le serveur reste en attente de connexions en écoutant continuellement sur le port spécifié grâce à une boucle infinie `while(true)`. Chaque nouvelle connexion crée un nouveau `ClientHandler` qui gère les com-

munications avec ce client.

La synchronisation est utilisée lors de l'ajout et du retrait de clients de la collection, afin d'éviter les problèmes de concurrence lorsque plusieurs threads essaient de modifier la collection en même temps.

## 2.2 Client

Le client se connecte au serveur, envoie son pseudonyme (vérifié pour qu'il soit unique) et permet à l'utilisateur de participer à la conversation. Deux threads sont utilisés pour gérer l'écoute des messages envoyés par le serveur (autres clients) et l'envoi des messages saisies par l'utilisateur.

- Le client se connecte au serveur à une adresse donnée (ici, localhost) et au port 1234.
- Les flux d'entrée et de sortie sont utilisés pour recevoir et envoyer des messages au serveur (et par extension aux autres clients).
- Pour gérer le chat, le client dispose de deux threads : l'un pour écouter les messages du serveur (via `ThreadListener`) et l'autre pour capturer l'entrée utilisateur et envoyer des messages (via `UserInputHandler`).

## 2.3 ClientHandler

La classe `ClientHandler` est une classe spécialisée dans la gestion de chaque client connecté. Elle étend *Thread* pour permettre une gestion parallèle des clients (multi-threading).

- Chaque instance de `ClientHandler` est responsable de la gestion d'un client particulier, en communiquant avec lui via les flux d'entrée et de sortie.
- Lorsqu'un nouveau client se connecte, le serveur lui demande de saisir un pseudonyme. Le pseudonyme est vérifié pour s'assurer qu'il n'est pas déjà utilisé par un autre client via la méthode `verifyUsername()`.
- Les messages envoyés par un client sont diffusés à tous les autres clients connectés. Cette diffusion se fait en excluant le client qui a envoyé le message initial via la méthode `diffuserMessage()`.
- Lorsque le client envoie "exit" ou se déconnecte de manière inattendue, le `ClientHandler` gère la fermeture des flux et la suppression du client de la liste via la méthode `disconnectHandler()`.

## 3 Logique d'opérations

- On exécute le programme `serveur.java`, ce qui démarre le serveur. Celui-ci se met à écouter les connexions sur le port 1234 ;
- Chaque client se connecte au serveur via une connexion socket (exécute `client.java`). Un `ClientHandler` est alors créé pour ce client ;
- Le serveur demande un pseudonyme au client et vérifie s'il est unique avant de l'accepter. Une fois accepté, le client peut envoyer et réceptionner des messages ;
- Les messages envoyés par un client sont reçus par son `ClientHandler`, puis diffusés à tous les autres clients ; Par extension, ce client réceptionne les messages envoyés par tous les autres.
- Lorsqu'un client se déconnecte ou quitte la conversation, son `ClientHandler` ferme la connexion proprement et l'enlève de la liste des clients actifs (collection synchronisée).

# Partie 3

## Scénarios d'utilisation de l'application

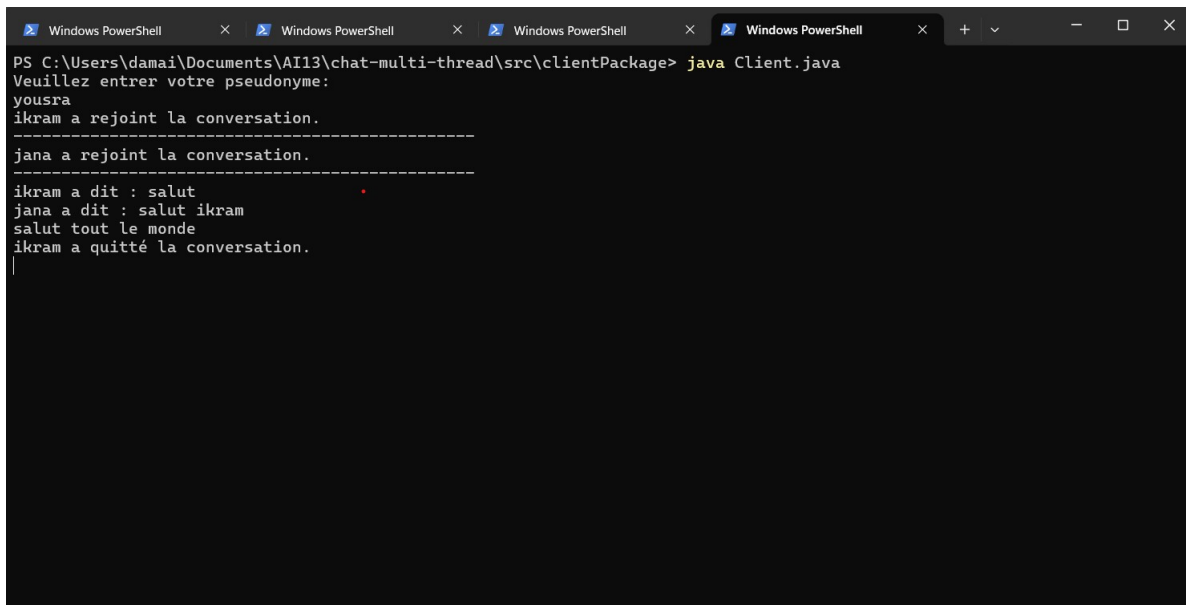
### 1 Récupération du code source

Pour récupérer le code source de l'application, suivez les étapes ci-dessous :

- Ouvrez un terminal sur votre machine.
- Clonez le dépôt Git avec la commande suivante :  

```
git clone -b main_v1 https://gitlab.utc.fr/jeltayeb/chat-multi-thread.git
```
- Accédez au répertoire du projet

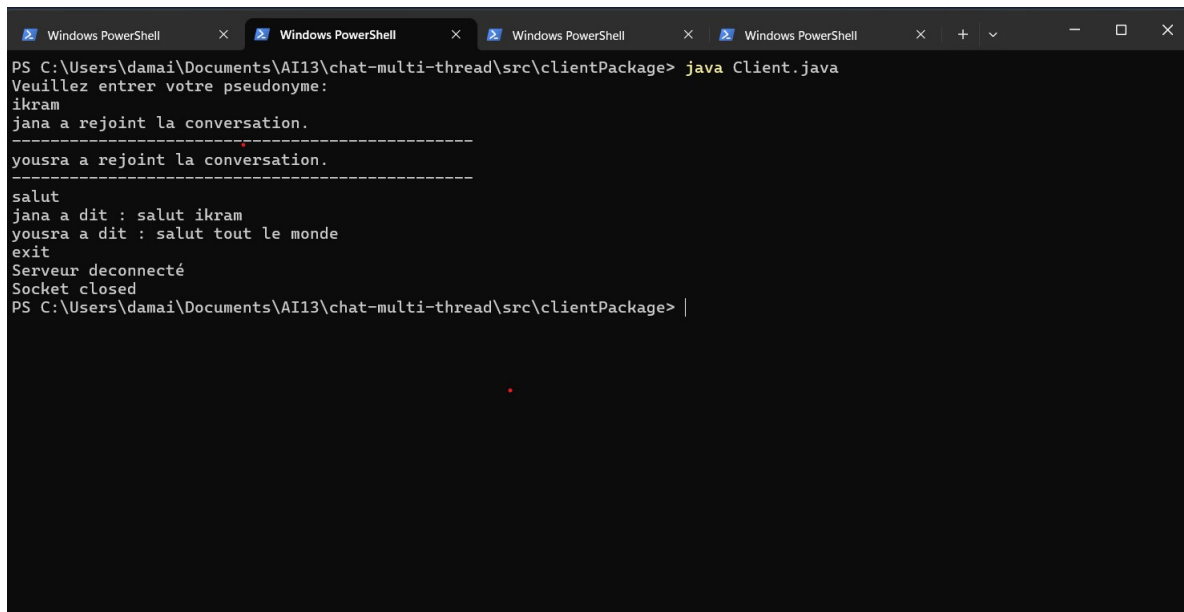
### 2 Lancement de l'application et démonstration des cas d'utilisation



```
PS C:\Users\damai\Documents\AI13\chat-multi-thread\src\clientPackage> java Client.java
Veuillez entrer votre pseudonyme:
yousra
ikram a rejoint la conversation.
-----
jana a rejoint la conversation.
-----
ikram a dit : salut
jana a dit : salut ikram
salut tout le monde
ikram a quitté la conversation.
```

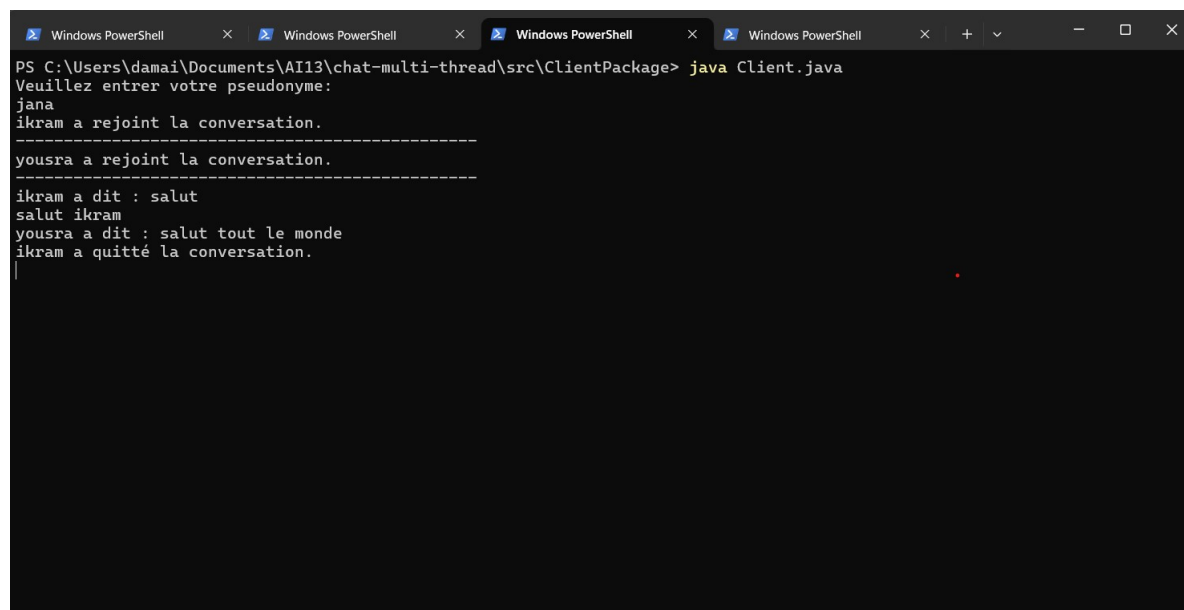
FIGURE 2 – Serveur en cours d'exécution





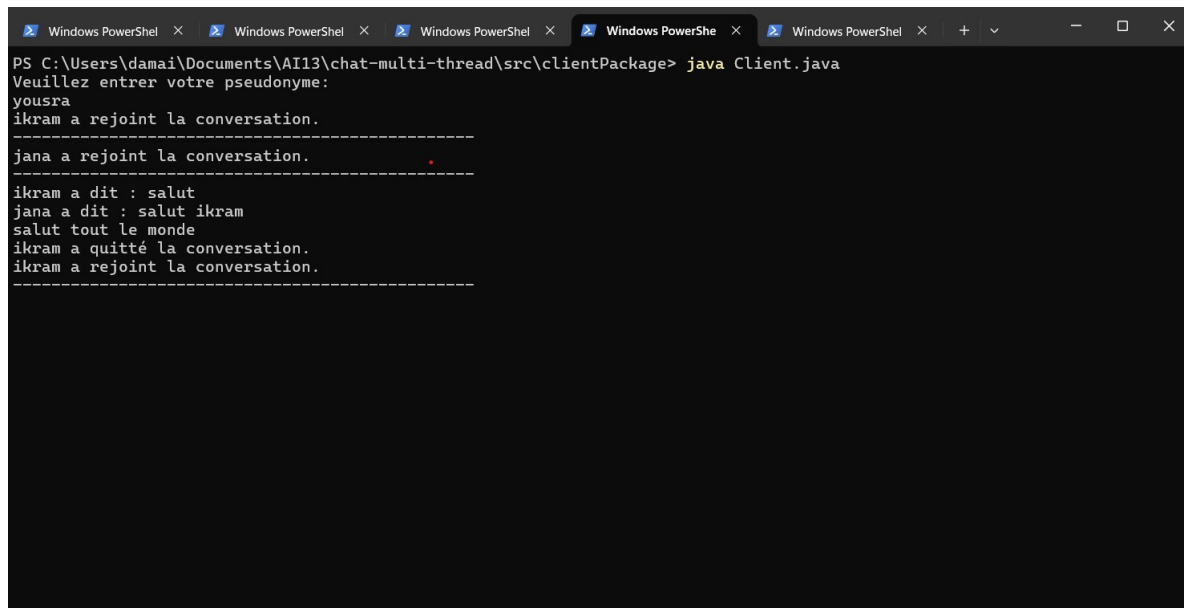
```
PS C:\Users\damai\Documents\AI13\chat-multi-thread\src\clientPackage> java Client.java
Veuillez entrer votre pseudonyme:
ikram
jana a rejoint la conversation.
-----
yousra a rejoint la conversation.
-----
salut
jana a dit : salut ikram
yousra a dit : salut tout le monde
exit
Serveur deconnecté
Socket closed
PS C:\Users\damai\Documents\AI13\chat-multi-thread\src\clientPackage> |
```

FIGURE 3 – Client 1 connecté



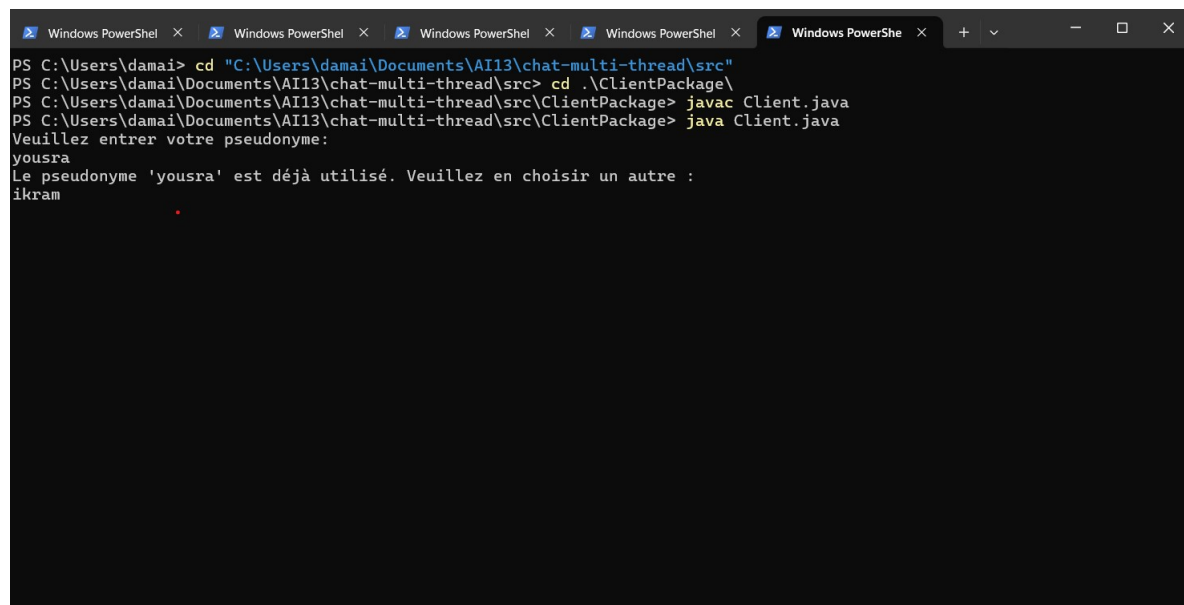
```
PS C:\Users\damai\Documents\AI13\chat-multi-thread\src\clientPackage> java Client.java
Veuillez entrer votre pseudonyme:
jana
ikram a rejoint la conversation.
-----
yousra a rejoint la conversation.
-----
ikram a dit : salut
salut ikram
yousra a dit : salut tout le monde
ikram a quitté la conversation.
|
```

FIGURE 4 – Client 2 connecté



```
PS C:\Users\damai\Documents\AI13\chat-multi-thread\src\clientPackage> java Client.java
Veuillez entrer votre pseudonyme:
yousra
ikram a rejoint la conversation.
-----
jana a rejoint la conversation.
-----
ikram a dit : salut
jana a dit : salut ikram
salut tout le monde
ikram a quitté la conversation.
ikram a rejoint la conversation.
-----
```

FIGURE 5 – Client 3 connecté



```
PS C:\Users\damai> cd "C:\Users\damai\Documents\AI13\chat-multi-thread\src"
PS C:\Users\damai\Documents\AI13\chat-multi-thread\src> cd .\ClientPackage\
PS C:\Users\damai\Documents\AI13\chat-multi-thread\src\ClientPackage> javac Client.java
PS C:\Users\damai\Documents\AI13\chat-multi-thread\src\ClientPackage> java Client.java
Veuillez entrer votre pseudonyme:
yousra
Le pseudonyme 'yousra' est déjà utilisé. Veuillez en choisir un autre :
ikram
.
```

FIGURE 6 – Client 4 connecté