Commençons par examiner le code du client :

Initialisation du client :

- Le code crée une instance de '**SocketChannel**' et se connecte à un serveur distant spécifié par l'adresse IP ('**host**') et le port ('**port'**).
- L'objet '**Scanner**' est utilisé pour lire l'entrée de l'utilisateur à partir de la console. L'utilisateur est invité à entrer son nom d'utilisateur.
- Le nom d'utilisateur est envoyé au serveur en tant que message, indiquant que l'utilisateur s'est connecté.

Threads pour la réception des messages :

- Un nouveau thread est démarré pour gérer la réception des messages du serveur. Ce thread tourne en boucle infinie.
- Dans la boucle, la méthode '**receiveMessage ()**' est appelée pour vérifier s'il y a des messages du serveur. Si oui, le message est affiché à la console. Cette boucle inclut également une pause de 100 millisecondes pour éviter une surcharge du CPU.

Boucle principale pour l'envoi des messages

- Dans la boucle principale, l'utilisateur peut saisir des messages à partir de la console.
- Chaque message est envoyé au serveur en utilisant la méthode '**sendMessage ()**', où le message est encodé en octets et placé dans un '**ByteBuffer**' qui est ensuite écrit dans le '**SocketChannel**'.

Méthode 'sendMessage ()' :

Cette méthode prend un message en tant que chaîne, le convertit en tableau d'octets, puis l'encapsule dans un '**ByteBuffer**'. Enfin, le message est écrit dans le '**SocketChannel**'.

Méthode 'receiveMessage ()' :

Cette méthode alloue un '**ByteBuffer**' pour recevoir les données du '**SocketChannel**'. Elle lit les données du canal dans le tampon et les affiche à la console.

Fonction principale 'main ()' :

Dans la fonction '**main ()**', une instance de '**ChatClient**' est créée en spécifiant l'adresse IP (« localhost ») et le port (8080) du serveur.

Des exceptions liées aux opérations d'E/S sont gérées et affichent les traces de la pile en cas d'erreur. En résumé, ce code représente un client de chat en réseau qui permet à un utilisateur d'envoyer des messages à un serveur distant et de recevoir les messages provenant d'autres utilisateurs connectés.

Passans à l'analyse du code serveur : ce code Java fourni implémente un serveur de chat simple utilisant les sockets non bloquants et le modèle de multiplexage d'E/S avec la classe '**Selector**'. Voici une explication détaillée du code :

Initialisation du serveur :

- Le serveur est créé à l'aide de la classe '**ServerSocketChannel**', qui permet de créer un canal pour écouter les connexions entrantes.
- Le serveur est configuré en mode non bloquant pour permettre la gestion simultanée de plusieurs connexions.
- Un '**Selector**' est créé et enregistré avec le '**ServerSocketChannel**' pour traiter les événements d'acceptation.

Méthode 'start ()' :

- La méthode **'start ()'** est appelée pour démarrer le serveur.
- Une boucle infinie est utilisée pour surveiller les événements d'E/S avec le **'Selector'**.
- Lorsque des événements sont détectés, la méthode **'handleAccept ()'** ou **'handleRead ()'** est appelée en fonction du type d'événement.

Méthode 'handleAccept ()' :

- Lorsqu'une connexion est acceptée, la méthode **'handleAccept ()'** est appelée.
- Un nouveau **'SocketChannel'** est créé pour représenter la connexion avec le client.
- Le nouveau canal est configuré en mode non bloquant et enregistré avec le **'Selector'** pour gérer les événements de lecture.
- Le client est ajouté à la liste des clients connectés avec un nom d'utilisateur généré.
- Un message est affiché sur le serveur indiquant que le client s'est connecté.

Méthode 'handleRead ()' :

- Lorsqu'un canal est prêt à être lu, la méthode **'handleRead ()'** est appelée.
- Les données lues depuis le canal du client sont stockées dans un tampon (**'ByteBuffer'**).
- Si le nombre d'octets lus est -1, cela signifie que le client s'est déconnecté. Dans ce cas, le serveur affiche un message et ferme la connexion.
- Le message est extrait du tampon et affiché sur le serveur avec le nom d'utilisateur du client.
- La méthode '**broadcastMessage ()**' est appelée pour diffuser le message à tous les clients connectés, à l'exception du client émetteur.

Méthode 'broadcastMessage ()' :

- La méthode **'broadcastMessage ()'** envoie le message à tous les clients connectés, à l'exception du client émetteur.
- Elle utilise un **'ByteBuffer'** pour envoyer les données sur tous les canaux des clients connectés.

Fonction principale 'main ()' :

1. La fonction **'main ()'** crée une instance de **'ChatServer'** et démarre le serveur en appelant la méthode **'start ()'**.

2. Les exceptions liées aux opérations d'E/S sont gérées et affichent les traces de la pile en cas d'erreur.

En résumé, ce code implémente un serveur de chat qui gère les connexions simultanées de plusieurs clients en utilisant le modèle de multiplexage d'E/S avec la classe **'Selector'**.
Il prend en charge la réception et l'envoi de messages entre les clients connectés.

Conclusion :

En conclusion, les codes du client et du serveur de chat en Java présentent une mise en œuvre fonctionnelle d'une communication en réseau.