Session n° 8

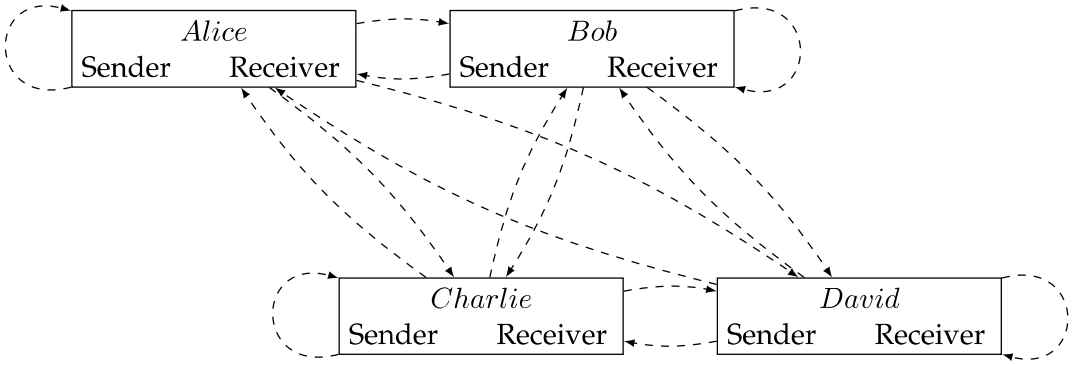
Projet - Group Chat

Le projet peux être préparé en binôme et est à rendre au plus tard le 10 Janvier 2024 à 23h59.

L'objectif de ce projet est la réalisation d'une application de messagerie.

- Il s'agira d'une messagerie de type **"Group chat"**, où chaque utilisateur peut voir les messages envoyés par tout le monde.
- Les messages seront accompagnés d'informations sur le nom de l'utilisateur et la date d'envoi, par exemple : `Alice dit "Bonjour!" le 12/12/2023 à 14h25`
- L'architecture sera **décentralisée**. Chaque machine connectée au chat jouera donc un rôle symétrique, et communiquera avec les autres directement, sans passer par un serveur central.
- **Chaque message** envoyé sur la messagerie sera traité comme une **communication TCP indépendante**. A chaque fois que l'on souhaite envoyer un message entre $Alice$ et Bob, on créera un socket de `java.io`, on enverra le message via les entrées et sorties textuelles du socket, puis on refermera le socket (afin que Bob sois libre d'écouter les messages provenant d'autres machines que $Alice$).
- Le projet sera réalisé en paradigme orienté objet, on évitera donc d'écrire l'ensemble du code dans les fonctions `main`, et on hésitera pas à créer de nouvelles classes.

Pour être plus précis, on écrira deux programmes, le **Sender** (qui permet d'envoyer des messages via l'entrée console standard) et le **Receiver** (qui réceptionne les messages envoyés et les affiche en console).



Exercice 1. *Sender*

1. Le programme `Sender.java` s'exécutera en ligne de commande avec un argument qui indiquera le nom d'utilisateur (exemple une fois la classe compilée : `java Sender Alice`)
2. On obtiendra une liste d'adresses IP, chacune correspondant à un Receiver.
   - Cette liste est lue dans un fichier `IP.txt`, qui contiendra une adresse par ligne.

— Une adresse de bouclage (`127.0.0.1`) figurera dans la liste, afin qu'*Alice* puisse lire ses propres messages dans la messagerie.

3. On lira l'entrée console standard ligne par ligne. Chaque ligne formera un message individuel, à envoyer à chaque Receiver.
4. Pour chaque message, et pour chaque Receiver, envoyer le message.
   — Ouvrir un canal TCP sur le port 2023.
   — Envoyer quatre lignes de texte sur ce canal :
     (a) D'abord, une ligne contenant uniquement le nom d'utilisateur (exemple : `Alice`)
     (b) Puis, la date d'envoi du message (exemple pour un message envoyé le 12 Décembre à 13h33 et 50 secondes : `2023-12-12 13:33:50`). [1]
     (c) Ensuite, le nombre de caractères du message (exemple : `8`)
     (d) Enfin, le message en lui même, c-à-d une ligne de texte sans retour à la ligne de la taille spécifiée précédemment (exemple : `Bonjour!`)
   — Refermer le canal ainsi que les buffers d'entrées/sorties.
5. Revenir à l'étape 3 et attendre l'entrée console suivante.

Exercice 2. *Receiver*

1. On écoutera les connexions TCP entrantes sur le port 2023.
2. À chaque connexion, on ouvrira un canal textuel, sur lequel on lira quatre lignes, comme précisé côté Sender (utilisateur, date, taille, message).
3. La connexion est alors refermée.
4. Le message sera enregistré dans une structure de données interne (exemple : une liste de `Message`, où `Message` est une classe que vous aurez défini qui représentera un message individuel et ses informations annexes).
5. Le message sera affiché en console.
6. Le message sera ajouté en tant que nouvelle ligne dans un fichier nommé : `messages.log`
7. Revenir à l'étape 1 et attendre la communication suivante.

Quelques précisions supplémentaires :

— Vous créerez et utiliserez une classe `Message` permettant de représenter un message envoyé sur le chat. Un objet de cette classe contiendra des attributs, de bon types, représentant le nom d'utilisateur, la date, la taille du message et le message en lui-même.
— Vous créerez une classe `CarnetAdresses` qui s'occupera pour le Sender de la gestion de la liste d'adresses IP. En particulier, un objet de cette classe se chargera de lire le contenu du fichier `IP.txt`, et disposera d'une méthode permettant d'accéder à cette liste sans que le fichier ne soit re-parcouru à chaque fois.
— En particulier, il est possible qu'une adresse dans cette liste ne soit pas en ligne et disponible, ce cas de figure devra être géré de manière satisfaisante (sans stopper l'exécution du Sender).
— Côté Receiver, on vérifiera pour chaque message reçu que la taille du message correspond à ce qui est indiqué. Sinon, le message sera ignoré et un message de warning sera affiché en console.

1. Utilisez java.time pour obtenir la date et l'heure.