

# Programmation événementielle

## TP 2 – Les sockets (chaines)

### Objectif

L'objectif de ce TP est de réaliser une application client-serveur où les clients peuvent s'appuyer sur un serveur pour convertir en majuscule une chaîne de caractères. L'ensemble des transmissions concerne des chaînes de caractères.

Plus précisément, l'unique service du serveur sera de renvoyer en majuscule tous les messages qu'un client lui envoie. Le serveur répondra par défaut sur le port 10000, mais il sera possible de le lancer sur un autre port passé en paramètre au lancement du programme. Une fois lancé, le serveur devra attendre la connexion d'un client en affichant « Serveur: à l'écoute sur le port 10000 ». Lorsque la connexion est établie avec un client, le serveur affichera « Serveur: connexion établie avec le client /10.x.y.z ». Un client pourra envoyer une chaîne de caractères sur la socket et il recevra ce même message en majuscule. Un client se déconnectera du serveur par l'envoi du message « exit » (ou tout autre mélange de minuscules et majuscules).

Dans ce TP, vous travaillerez dans un répertoire source TP2. Vous respecterez les principes de la programmation orientée objet vus en première année et deuxième année.

### Serveur Majuscule monothreadé

Dans un premier temps, vous allez mettre en place un serveur pour communiquer avec un seul client. Pour vous aider, la classe tp2.MonoServerMajuscule est fournie sur Celene pour servir de serveur de sockets.




1. Ajoutez la classe tp2.MonoServerMajuscule à votre projet. Pour cela, vous pouvez copier-coller le fichier dans le package approprié de votre projet.
2. Analysez l'ensemble du code en vous appuyant sur les commentaires et répondez aux questions suivantes :
  - a. Quelle méthode reçoit les messages ?
  - b. Quelle méthode envoie les messages ?
  - c. Quelle méthode permet de convertir une chaîne de caractères en majuscules ?
  - d. Quelle méthode permet d'ignorer la casse lors d'une comparaison de chaînes de caractères ?

### Client Majuscule

Vous allez maintenant développer un client pour envoyer les messages sur le serveur et recevoir en retour la chaîne de caractères en majuscules.

3. Créez dans le même package une classe nommée ClientMajuscule.
4. Ajoutez un constructeur contenant la création d'un socket au serveur Majuscule ci-dessus. Le socket créé devra être un attribut de classe défini de la manière suivante : `private Socket socket = null;`. Pour obtenir l'adresse locale, vous pourrez utiliser `InetAddress.getLocalHost()`.
5. Ajoutez l'instruction pour fermer le socket.
6. Ajoutez une fonction principale pour créer un client.

Pour exécuter l'application client-serveur Majuscule, il faut démarrer deux processus : un processus serveur correspondant à l'exécution de la fonction principale de MonoServerMajuscule et un processus client correspondant à l'exécution de la fonction principale de ClientMajuscule. Il faut donc appuyer deux fois sur l'icône

 . Pour afficher dans la console le bon processus, vous pouvez appuyer sur l'icône  . Enfin, il faut penser à arrêter vos processus en appuyant sur  lorsque vous souhaitez relancer votre application.

7. Vérifiez que la connexion s'établit bien avec le serveur. Normalement, vous aurez un message d'erreur (pointeur nul) dans la fonction d'écoute du serveur. Pourquoi ?
8. Recopiez les méthodes `public String readMessage()` et `public void sendMessage(String msg)` de la classe `MonoServerMajuscule` dans votre classe `ClientMajuscule`. Comme les échanges sont symétriques, ces méthodes peuvent être utilisées aussi bien côté serveur que côté client.

Il ne reste plus qu'à ajouter la méthode principale `public void envoi()` qui est la boucle de discussion avec le serveur.

9. Ajoutez la méthode `envoi()` à votre classe en implémentant les points suivants :
  - a. Lire un message au clavier
  - b. Transmettre le message au serveur
  - c. Lire la réponse
  - d. Afficher la réponse
  - e. Mettre l'ensemble dans une boucle dont on sortira lorsque le message sera « exit »
10. Appelez la méthode `envoi()` dans le constructeur.
11. Vérifiez le bon fonctionnement du client et du serveur.

## Serveur Majuscule multithreadé

Enfin, vous allez proposer un serveur pour pouvoir converser avec plusieurs clients. De manière intéressante, le serveur développé dans la section précédente restera le même.

12. Copiez la classe `MonoServerMajuscule` en la renommant en `MultiServerMajuscule`.
13. Créez une classe `ThreadMajuscule` qui héritera de la classe `Thread`. Cette classe contiendra l'ensemble des éléments utiles à la conversation avec un client donné.
14. Copiez les méthodes `private void ecoute()`, `public String readMessage()` et `public void sendMessage(String msg)` de `MultiServerMajuscule` dans `ThreadMajuscule`.
15. Renommez la méthode `ecoute()` en `run()` car il s'agit de l'amorce du `Thread`.
16. Ajoutez un attribut de classe `socket` que vous initialiserez dans le constructeur de la classe. L'idée est que la classe `MultiServerMajuscule` passera en argument du constructeur le `socket` du client.
17. Modifiez la méthode `ecoute()` de la classe `MultiServerMajuscule` pour construire un objet `ThreadMajuscule` en passant le `socket` client à chaque fois qu'une nouvelle connexion arrive sur le serveur. Il faut bien penser à démarrer le `thread`.
18. Vérifiez le bon fonctionnement de l'ensemble en démarrant le serveur et au moins deux clients pour échanger avec le serveur.