Communiquer avec des capteurs IoT situés "sur le cloud"

EVE - Systèmes d'acquisition - S. REYNAL

Année universitaire 2020-2021

Ce TD Java a pour objectif de vous initier à la communication avec des capteurs distants via l'utilisation de Sockets et du multithreading. Les fonctionnalités que vous allez étudier aujourd'hui sont les suivantes :

- Créer une exécution parallélisée du code avec la classe Thread (java.lang)
- Créer un serveur avec la classe ServerSocket (java.net)
- Créer un client avec la classe Socket (java.net)

L'objectif est de simuler la réception par un serveur (la machine sur laquelle vous développez) de données (température, etc.) émises par plusieurs capteurs intelligents connectés au serveur via le réseau (ethernet ou WiFi, par ex des xbee wifi).

1 Threads

Les threads sont des fils d'exécution, des "processus légers" (lightweight process) qui permettent de simuler du parallélisme à l'intérieur de votre application Java. La classe Thread permet, après surchage de sa méthode run(), de créer un objet de type Thread qui, à l'appel de la méthode start(), executera dans un fil d'exécution distinct le code contenu dans run().

Travail demandé : implémenter la classe "**Sensor**" qui hérite de la classe "Thread". Cette classe se contente pour l'instant, dés que le thread est démarré, d'afficher régulièrement, chaque seconde, une donnée de mesure factice (un nombre aléatoire) à l'écran (via la console). Le code correspondant se trouvera dans la méthode "run()", et repose sur une boucle infinie et l'appel de la méthode "sleep()" (attention à bien gérer les exceptions associées). Les threads seront lancés depuis la méthode "main()". L'affichage souhaité pour chaque Sensor est à titre indicatif du type : *"Capteur numéro 4 : température = 28C"*.

2 Sockets

2.1 Serveur

La classe Socket représente une connection entre deux machines possédant une adresse IP. La classe ServerSocket permet de créer un serveur qui "écoute" et "parle" sur un port et une adresse IP donnés.

Travail demandé : Implémenter la classe Serveur. Cette classe hérite de ServerSocket et est chargé d'attendre les connections émanant de clients (les capteurs). Elle repose sur l'appel de la méthode "accept()" de la classe ServerSocket, qui renvoie un Socket représentant la connection active. Une fois la connection activée, récupérer le InputStream associé à la connection via la méthode ad hoc dans Socket, lire une donnée (un double) depuis la connection, fermer la connection, et passer au client suivant. Pour l'instant, le serveur ne peut gérer qu'une connection active avec un client à la fois. On devra utiliser la classe DataInputStream (paquet java.io).

2.2 Clients

Le code n'étant véritablement fonctionnel que s'il existe des clients, ajouter à la classe Sensor dans la méthode run(), la connection au serveur (via la classe Socket). Une fois la connection acceptée par le serveur, récupérer le OutputStream associé (via la méthode getOutputStream de la classe Socket), et écrire un double correspondant à la température.

Ajouter à la classe Serveur la possibilité de gérer plusieurs connection simultanément, via le multithreading. Pour cela, il faut insérer le code qui attend un client (la méthode accept()), à l'intérieur de la méthode run() d'un thread (ou implémenter la méthode run() de l'interface Runnable, autre approche), puis lancer un nouveau thread à chaque nouveau client. Ensuite, il suffit de laisser les threads tourner en boucle infinie et communiquer avec leur capteur attaché.

Remarque : vous pouvez travailler en équipe, un.e étudiant.e développant la classe Sensor, un.e autre la classe Serveur. Ainsi les "faux" capteurs peuvent être disséminés sur plusieurs machines et émettre leur données pour le reste du groupe ! Attention, en distanciel, assurez vous d'avoir accès à d'autres machines du groupe !