Conception de logiciels intranet : intergiciels à objets répartis
(NSY102)

Serveur d'exécution concurrent

Le but de ce TP est d'implanter un serveur d'exécution séquentiel et concurrent. Pour être exécutable par le serveur, une tâche devra simplement implanter l'interface suivante :

```
interface Task<T> extends Serializable {
    T execute();
}
```

1 Exécution séquentielle des tâches

Dans le cas séquentiel, l'implantation se divise ainsi :

- une implantation des canaux, pour des communications locales ou distantes, reprise du TP précédent.
- une API client qui implante une méthode `T executeTask(Task<T> t)`.
- un serveur qui attend une tâche, l'exécute, et renvoie le résultat.
- au moins deux tâches différentes pour les tests (qui implantent l'interface `Task<T>` pour divers types `T`).

2 Exécution concurrente des tâches

Dans le cas concurrent, on souhaite de plus être capable de demander l'exécution concurrente de de deux tâches. Il faut pour cela :

- remplacer l'API client par une méthode :

  ```
  Pair<T, T> executeTask(Task<T> t1, Task<T> t2)
  ```

- modifier le serveur pour qu'il crée deux serveurs esclaves à qui il sous-traite l'exécution des deux tâches. Ces serveurs esclaves communiquent avec le serveur maître en utilisant des threads et des canaux locaux.
- plusieurs exemples de tâches différentes pour les tests (qui implantent l'interface `Task<T>` pour divers types `T`). En particulier, un des tests devra exploiter la concurrence pour accélerer l'exécution d'un calcul (en supposant plusieurs CPU).
- (*option*) généraliser le code ci-dessus à une liste de tâches.

1 Programme de test

Tester votre implantation de deux manières :

Version locale. Les canaux utilisent une implantation de l'interface `BlockingQueue<A>`.

Version distante. Les canaux utilisent les sockets et la sérialisation. Le numéro de port sera un paramère optionnel du client et du serveur, on utilisera par default le port 55555.

On utilisera le package `java.util.logging` pour la journalisation côté serveur.