Programmation et modélisation orientées objet
(USAL5A & USAL2H)

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 de 2 tâches

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

- remplacer l'API client par une méthode :

  ```
  Pair<T, T> executeTasks(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).

3 Exécution concurrente d'une liste de tâches

On souhaite maintenant être capable de demander l'exécution concurrente d'une liste de tâches. Nous utiliserons ici la classe `FutureTask` à la place des esclaves. Il faut pour cela :

- remplacer l'API client par une méthode :

  ```
  List<T> executeTasks(List<Task<T>> t)
  ```

- modifier le serveur pour qu'il lance l'exécution concurrente de la liste de tâches.
- 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).

4 Programme de test

Tester votre implantation de deux manières :

- **Version locale.** Une classe `Main` qui lance à la fois le client et le serveur.
- **Version distante.** Une classe `ClientMain` qui lance le client et une classe `ComputeEngineMain` qui lance le serveur le serveur.
- **Version unifiée.** Les classes `ClientMain`, `ComputeEngineMain` et `Main` adaptées pour utiliser le *subpackage* `channel`. Les classes `ClientMain` et `ComputeEngineMain` utilisent les adaptateurs « distants » et la classes `Main` utilise les adaptateurs « locaux ».

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