

# TD/TP #11 – Exécuteurs Java

## Cours Applications Concurrentes, *Polytech-INFO4*

*F. Boyer, N. de Palma, Université Grenoble Alpes, sept 2020*

### 1. Notion d'exécuteur

La notion d'exécuteur a été introduite afin de faciliter la mise en œuvre de traitements de type producteurs-consommateurs, dans lesquels les producteurs déposent des requêtes dont l'exécution sera prise en charge par les consommateurs. Un exécuteur expose une interface permettant essentiellement de déposer des requêtes à exécuter.

```
public interface Executor {  
    void execute (Runnable request) ;  
}
```

Dans sa mise en œuvre, un exécuteur englobe généralement un buffer de type production-consommation dans lequel sont déposées les requêtes, et un ensemble de threads (généralement appelés *workers*) qui vont consommer ces requêtes et les exécuter.

Différentes politiques de gestion du pool de threads workers peuvent être proposées (Single Thread Executor, Fixed Thread Pool, Cached Thread Pool, etc).

### 2. Première mise en pratique / interface Executor et ExecutorService

**Démarrage.** Décompressez le paquet `executor.zip`. Vous obtenez un répertoire `executor` qui correspond à un package. Vous pouvez soit intégrer ce package dans le projet Eclipse de votre choix (en copiant le répertoire `executor` dans le projet), soit créer un nouveau projet, à votre convenance.

**Exercice.** Dans un premier temps, regardez et comprenez le contenu de la classe `executor.etd.SampleExecutor.java`. Exécutez là tout d'abord avec `NTHREADS=1`, c'est à dire avec un exécuteur composé d'un seul thread *worker*. Ensuite exécutez cette classe avec `NTHREADS=3` par exemple. Expliquez pourquoi les traces qui s'affichent sont entrelacées ou pas.

**Question.** Regardez la console d'exécution d'Eclipse. Est ce que votre programme précédemment lancé est encore en vie ? Pourquoi ? Prenez soin de le terminer en cliquant sur le petit carré rouge dans la console.

**Exercice.** En utilisant l'interface *ExecutorService* permettant de contrôler le cycle de vie d'un exécuteur (notamment les méthodes *shutdown()* et *awaitTerminated(int timeout, TimeUnit unit)*), faites en sorte que votre programme se termine et que la trace « *main terminated* » ne s'affiche qu'après que les tâches de l'exécuteur aient toutes fini leur exécution.

```
public interface ExecutorService {
    void shutdown() ;
    List<Runnable> shutdownNow() ;
    boolean awaitTermination(long timeout, TimeUnit unit) ;
    ...
}
```

### 3. Mise en pratique / interfaces Callable et Futures

Les interfaces *Callable* et *Future* permettent d'exécuter des tâches qui renvoient un résultat de type quelconque. Une tâche qui renvoie un résultat sera définie comme implémentant l'interface *Callable*. Lorsqu'une telle tâche est déposée dans un exécuteur, on récupère un objet de type *Future*, qui permettra de récupérer le résultat de la tâche au travers de la méthode *get()*. Cette méthode est bloquante.

```
public interface Callable<V> {
    V call() throws Exception ;
}

public interface Future<V> {
    V get() throws InterruptedException, ExecutionException ;
    V get(long timeout, TimeUnit unit) throws ..
    boolean cancel(boolean mayInterruptIfRunning) ;
    ..
}
```

**Exercice.** Regardez et comprenez les classes du package `executor.etd.wcount`. Les classes `SerialLauncher` et `ParallelLauncher` lancent un programme de comptage de mots dans le fichier `test-large.utf8`. Pour obtenir des temps significatifs, on réalise le comptage `NTIMES` fois (`NTIMES` étant une constante définie à 5000 par défaut).

Lancez l'exécution de la classe `SerialLauncher`, qui lance cette activité de comptage dans une version totalement séquentielle basée sur un unique thread.

Maintenant, on va comparer cette version séquentielle à une version parallèle basée sur un exécuteur. Le principe est de créer une tâche par fichier à analyser. Cette version parallèle est mise en œuvre dans la classe `ParallelLauncher`, mais elle est incomplète. Vous devez compléter cette classe pour faire en sorte de récupérer les résultats des différentes tâches de comptage de mots, et d'ajouter ces résultats pour obtenir le nombre de mots global.

Une fois cela fait, jouez avec la taille du pool de thread de l'exécuteur, et voyez si cela engendre des différences au niveau des résultats.