Thème 9 - TD

Objectifs

— Pool de threads, interface `Executor`

Les indispensables de l'API (extrait)

Interface *Callable<V>*

Modifier and Type	Method	Description
`V`	`call()`	Computes a result, or throws an exception if unable to do so.

Interface *CompletionService<V>*

Modifier and Type	Method	Description
`Future<V>`	`submit(Callable<V> task)`	Submits a value-returning task for execution and returns a Future representing the pending results of the task.

Interface *Executor*

Modifier and Type	Method	Description
`void`	`execute(Runnable command)`	Executes the given command at some time in the future.

Interface *ExecutorService* (sous-interface d'*Executor*)

Modifier and Type	Method	Description
`boolean`	`isTerminated()`	Returns true if all tasks have completed following shut down.
`void`	`shutdown()`	Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted.
`<T> Future<T>`	`submit(Callable<T> task)`	Submits a value-returning task for execution and returns a Future representing the pending results of the task.

Interface *Future<V>*

Modifier and Type	Method	Description
`V`	`get()`	Waits if necessary for the computation to complete, and then retrieves its result.

Classe *ExecutorCompletionService<V>* (implémente *CompletionService<V>*)

Constructor	Description
`ExecutorCompletionService(Executor executor)`	Creates an ExecutorCompletionService using the supplied executor for base task execution and a LinkedBlockingQueue as a completion queue.

Modifier and Type	Method	Description
`Future<V>`	`take()`	Retrieves and removes the Future representing the next completed task, waiting if none are yet present.

Classe *Executors*

Modifier and Type	Method	Description
`static` `ExecutorService`	`newFixedThreadPool(int nThreads)`	Creates a thread pool that reuses a fixed number of threads operating off a shared unbounded queue.

Exercice 14 – Utilisation d'un *Executor* pour le produit matriciel

Les opérations de création et destruction de threads sont coûteuses en temps. Avoir un thread par tâche n'est donc pas forcément une bonne stratégie, en particulier l'allocation et la désallocation des ressources mémoire affectées à chaque thread peut entraîner un ralentissement de l'exécution. Il est donc souvent intéressant de dissocier la création d'un ensemble de threads des tâches qu'ils vont exécuter.

Dans ce cas, une tâche est "chargée" sur un thread et lorsque cette tâche est terminée, le même thread, s'il n'a pas été détruit, peut être utilisé pour exécuter une autre tâche.

Nous allons appliquer cette approche au calcul du produit matriciel que nous avons réalisé avec un thread par tâche.

Question 1
Donnez l'instruction permettant de créer un ensemble (pool) de threads dont le nombre `NB_THREADS` ne varie pas au cours de l'exécution du programme.

Question 2
Nous avons construit une classe `CalculElem` qui implémente l'interface `Runnable`. Soit :

```
public CalculElem (MatriceEntiere m, MatriceEntiere m1, int i, MatriceEntiere m2, int j)
```

un constructeur de cette classe. La méthode `run` de l'instance `CalculElem(m, m1, i, m2, j)` affecte à l'élément en position `(i, j)` dans la matrice `m` la valeur résultant du produit de la ligne `i` de `m1` par la colonne `j` de `m2`.

Donnez la suite d'instructions permettant de calculer l'ensemble des éléments de la matrice résultat en répartissant les calculs sur les `NB_THREADS` créés.

Question 3
Comment peut-on s'assurer de la terminaison du pool de threads et garantir que l'affichage de la matrice n'a pas lieu avant que l'ensemble des éléments aient été calculés ?

L'une des limites de l'interface `Runnable` est que la méthode `run` ne renvoie pas de résultat et ne peut pas lancer d'exception. Ces restrictions sont levées par la méthode `call` de l'interface `Callable`.

Question 4
Nous avons défini dans la classe `MatriceEntiere` une méthode :

```
public static int produitLigneColonne (MatriceEntiere m1, int i, MatriceEntiere m2, int j)
```

qui peut lancer une exception si les données sont de tailles incompatibles.

Écrivez le code d'une classe implémentant l'interface `Callable` et dont la méthode `call` retourne le résultat calculé par `produitLigneColonne`.

Question 5
Quelle méthode permet de soumettre la tâche pour exécution ? Comment peut-on récupérer le résultat ?

Question 6
Proposez une implémentation du produit matriciel multi-threads en utilisant des objets `Callable`.

Question 7
Quel problème pose l'utilisation d'un *CompletionService* pour récupérer les résultats des différents calculs ? Proposez une solution.