Thread, interruption et Executor

Exercice 1 - Hello Thread

On souhaite créer deux threads exécutant le même code. Pour différencier les deux thread, lors de la construction de celle-ci, un entier unique (`id`) leur sera fourni, 0 pour la première et 1 pour la seconde.
Chaque thread executera le même code qui consiste à afficher `hello` suivi du numéro de la thread ansi que la valeur d'un compteur indiquant le nombre de fois que la thread a affiché ce message.
Exemple :

```
...
hello 0 10714
hello 0 10715
hello 0 10716
hello 0 10717
hello 1 15096
hello 1 15097
hello 1 15098
hello 1 15099
...
```

1 Expliquer sur l'exemple ci-dessus pourquoi le compteur de `hello 0` est beaucoup plus petit que le compteur de `hello 1`.

2 Écrire le programme demandé en héritant de la classe `Thread` et en redéfinissant sa méthode `run()`.

3 A quoi sert l'interface `Runnable` ?
Modifier votre code pour utiliser un `Runnable`.

4 Changer votre code pour que l'on puisse choisir lors de l'exécution du programme le nombre de thread que l'on veut lancer en concurrence.

```
java HelloThread 5
```

Lance 5 threads en concurrence.

Exercice 2 - Coitus interruptus

On souhaite maintenant permettre à l'utilisateur en tapant un nombre d'interrompre le thread ayant cet `id`.

1 Expliquer les différences entre la méthode `interrupted` et `isInterrupted` de la classe `Thread`.

2 Écrire la classe `Interruptus` en utilisant un scanner (`java.util.Scanner`) pour lire l'entrée standard et `interrupt()` pour interrompre un thread.

3 Monitoring de l'activté de la machine virtuelle.
Lancer votre programme avec la commande :

```
java -Dcom.sun.management.jmxremote HelloThread 5
```

Puis dans un autre shell lancer la commande :

```
jconsole
```

et sélectionner `HelloThread` dans les processus locaux. Vérifier par ce biais que les threads sont bien interrompus.

Exercice 3 - Travail en parallèle

On souhaite écrire un programme affichant les valeurs des racines carrées de 0 à 10 000. On souhaite paralléliser le programme et permettre d'executer en même temps plusieurs calculs, chaque thread devra par exemple effectuer le calcul d'une centaine de racine carré.

1. Créer un `ExecutorService` en utilisant la classe `Executors` permettant de distribuer le calcul sur 5 threads différentes.
2. Écrire le programme qui effectue le calcul en parallèle en utilisant des `Runnable` qui effecteut l'affichage.
   On permet ici que l'affichage ne s'effectue pas dans l'ordre des racines carrés croissantes.
   Penser à utiliser la méthode `shutdown` pour indiquer qu'il n'y a plus de calcul à effectuer.
3. Modifier votre programme en utilisant l'interface `Callable` à la place de `Runnable` et le mécansime de `Future` pour effectuer l'affichage dans l'ordre.

Exercice 4 - Find en parallèle [à la maison]

On souhaite permettre à un utilisateur de rechercher un fichier dans une sous-arborescence spécifiée en utilisant une expression régulière.

1. Écrire le programme qui prend en paramètre une expression régulière et qui affiche l'ensemble des fichiers du répertoire courant dont le nom vérifie l'expression régulière.
   On utilisera pour cela les classes `java.util.regex.Pattern` et `java.util.regex.Matcher` ainsi que la méthode `listFiles` de la classe `java.io.File`.
2. On souhaite maintenant étendre le parcours à l'ensemble des sous répertoires en effectuant le parcours de chaque répertoire de façon concurrente.
   Comment doit-on faire ?
3. Discuter des différents arguments qu'il faut fournir au `ThreadPoolExecutor` pour que l'on puisse l'utiliser pour effectuer la recherche.
4. Écrire le programme qui effectueut la recherche parallèle.
   Attention, pour que le programme s'arrête il faut que :
   - Le nombre de `core pool thread` soit zéro lors de la construction.
   - Les `worker threads` meurent après un temps d'inactivité.