TD 1 premiers threads

Tests de voitures en parallèle - Interruption de thread

Exercice 1 :

Création, utilisation de processus légers, sans ressource partagée.
Tâches parallèles asynchrones sans concurrence.

L'objectif de cet exercice est de permettre la réalisation de tests de freinage de voitures de modèles différents en parallèle. Ainsi, on pourra reprendre dans un premier temps le corrgé du td 7 du module BPO qui proposait ces tests séquentiellement, le transformer pour paralléliser les tests puis introduire un risque aléatoire d'échec lors du test afin de constater que l'échec du test d'une voiture n'a pas d'incidence sur le bon déroulement des tests des autres voitures.

Le code du main de l'application client.Appli fait apparaître l'enchainement des tests de voitures à l'aide d'une boucle for :

```
for (Voiture v : tab) {
      int nb = testMoteur(v);
      System.out.println(v + " -> " + nb);
    }
```

Cette structure du code implique que les voitures seront testées séquentiellement : le test de la voiture tab[i+1] ne démarre **qu'après la fin** du test de la voiture tab[i].

L'objectif est de créer une tâche (ou activité) pour chaque voiture et de déplacer l'algorithmique de test dans le code du run() de cette activité.

Pour cela, après avoir rapatrié dans votre workspace le code source de l'énoncé et l'avoir testé :

1, Créez une classe TestMoteur qui devra pouvoir être exécutée dans son propre thread (donc être Runnable) et se lancera via la méthode *public void lancer().* Le *run()* de cette tâche devra faire le travail du test d'une voiture.

2, Supprimer de client.Appli le code de test et remplacez dans le main l'appel de la méthode testMoteur (et l'afichage de nb) par le lancement d'une instance de TestMoteur.

3, Testez plusieurs fois votre application. Afin de voir plus précisément ce qui se passe, vous afficherez, à la fin du test moteur, le nombre d'accélérations et le nombre de freinages.
Vous devez constater que, si on aboutit toujours à tester correctement chaque voiture, les tests ne se finissent pas toujours dans le même ordre, et à fortiori pas dans l'ordre où ils ont été lancés.

4, Pour simuler un problème lors du test d'une voiture (le testeur de cette voiture a grillé lors du freinage par exemple...), vous pourrez à l'aide de la fonction Math.random() introduire dans la boucle de freinage un plantage aléatoire de votre activité (vous pouvez générer une RuntimeException par exemple avec un message explicite « Le test de freinage de la voiture xxx a échoué ».
Relancez les tests et vous devez constater que le plantage d'un thread n'affecte pas les autres threads en cours.

Exercice 2 :

On souhaite écrire un programme qui effectue un travail simple et répétitif à l'infini mais qui s'arrête lorsqu'on appuie sur une touche du clavier. Pour cela, vous devrez écrire les classes Compteur et Arreteur du main ci dessous. Le compteur comptera (et affichera) un entier incrémenté et sera interrompue par l'arreteur :

```
public static void main(String[] args) {
      Compteur compte= new Compteur();
      compte.lancer(); // lancement du compteur
      new Arreteur(compte).lancer(); // lancement de l'arreteur du compteur
}
```

2.1. Interruption d'un thread via un booléen

Dans cet exercice, un attribut booléen *fin* (et son setter) dans la classe Compteur sera utilisé par l'arreteur pour dire au compteur de s'arrêter.

2.2. Interruption d'un thread via le mécanisme interrupt()/isInterrupted()/InterruptedException de la classe Thread

On reprend le problème précédent en utilisant cette fois ci le principe d'interruption des threads basé sur le staus d'interruption d'un thread et la méthode *interrupt()* de java.lang.Thread, qui indique à un thread qu'il a reçu une demande d'interruption. Seule la classe Compteur est à modifier pour ce 2.2.

Version thread actif

Le thread qui reçoit interrupt(), s'il est actif, peut tester si on l'a interrompu avec la méthode isInterrupted().

Version thread non actif (en veille)

Si le thread qui reçoit interrupt() est en veille (Thread.sleep(1) par exemple met le thread courant en veille pendant 1ms), une InterruptedException sera levée par la méthode mise en veille[i] et un try/catch permet de prendre en compte la demande d'interruption.

Ecrire les 2 versions au sein de la classe Compteur.

---

[i] Par exemple, le prototype suivant : **public static void sleep (long ms) throws InterruptedException**