TP 0
Threads en Java : premiers pas

A. Threads et exceptions

1. Créer une classe `Activite` qui implémente l'interface `Runnable` et qui a pour comportement (méthode `run`) d'afficher l'identifiant du *thread* sur lequel elle s'exécute.
2. Créer une classe `Application` ayant une méthode `main` qui lance 5 *threads* chacun exécutant une instance de `Activité`, et exécuter le programme.
3. Modifier la méthode `run` de `Activite` pour qu'elle fasse appel à une autre méthode `m()`, qui a son tour appelle une méthode `n()`, qui « simule » une erreur donnant lieu à une exception (division par zéro ou accès à un pointeur null) . Essayez le programme sans traiter l'exception dans la fonction `run`.
4. L'exception sera maintenant traitée par un bloc `try/catch` dans la méthode `run` en affichant l'identifiant du *thread* courant et la pile d'appel au moment de l'erreur. Essayez le programme à nouveau.
5. Créer une classe `MonException` qui dérive de `java.lang.Exception`. Modifier la méthode `run` de `Activite` pour qu'elle fasse appel à une autre méthode `f()` qui soulève une exception de type `MonException` aléatoirement avec une probabilité de 0.2 (voir `java.lang.Math`).
   Comme vous le verrez, le compilateur Java vous obligera alors de déclarer que la fonction `f` peut soulever l'exception (clause `throws`) et de traiter l'exception dans `run`.

B. Propagation des opérations sur la mémoire

Exécuter le programme suivant :

```
public class ThreadTest1 {
  public static boolean running = true;

  public static void main(String[] args) {
   // un premier Thread qui lit la variable running
   Thread t1 = new Thread() {
      public void run() {
        int counter = 0;
        while (running) {
          counter++;
        }
```

```
        System.out.println("Thread 1 fini. counter="+counter);
      }
    };

   // un second Thread qui modifie la variable running
   Thread t2 = new Thread() {
      public void run() {
        try {
          Thread.sleep(100);
        } catch (InterruptedException ignored) { }

        System.out.println("Thread 2 fini");
        running = false;
      }
    };
   t1.start();
   t2.start();
  }
}
```

1. Répéter l'exécution plusieurs fois. Que constatez vous ?
2. Ajouter le mot clé `volatile` à la déclaration de `running`. Que constatez vous ?

C. Exclusion mutuelle

Exécuter le programme suivant :

```
public class ThreadTest2 {
  public static int i = 0;
  public static void main(String[] args) {
   // lancer 5 threads qui incrémentent i

   for(int j=0;j<5;j++) {
     Thread t1 = new Thread() {
       public void run() {
         for(int j=0; j<100000; j++)
           i++;
       }
     };
     t1.start();
   }

   try {
          Thread.sleep(5000); // devrait suffire largement
   } catch (InterruptedException ignored) { }
```

```
    System.out.println("Valeur de i : " +i);
    System.out.println("(devrait etre 500000)");
   }
}
```

1. Répéter l'exécution plusieurs fois. Que constatez vous ?
2. Ajouter `volatile` à la déclaration de `i`. Que constatez vous ?
3. Trouver un moyen de corriger ce programme en utilisant la primitive `synchronized`.