TD/TP 2, THREADS JAVA

L'objectif de ce TD/TP est de vous faire programmer des threads en Java, de vous faire observer le comportement des programmes à activités parallèles (multi-threadés) et de vous montrer le besoin de synchronisation entre threads.

1 IMPLÉMENTATION DES THREADS EN JAVA

Les threads Java peuvent être implémentés de deux manières. La première est d'étendre la classe prédéfinie **`Thread`**:

```
//classes et interfaces prédéfinies Java, JDK
interface Runnable {
   void run();
}
public class Thread     extends Object
                        implements Runnable {
   void run() {...}
   void start() {...}
   ...
}
public class Compteur extends Thread {
   public void run() {...}
   }
   public static main() {
        Compteur c = new Compteur();
        c.start();
   }

 }
```

L'héritage à partir de **`Thread`** est contraignant car il empêche tout autre héritage.

Deuxième manière de faire: implémenter l'iterface **`Runnable`** et de cette manière avoir la possibilité d'héritage et d'implémentation d'autres interfaces.

```
.//classes et interfaces prédéfinies Java, JDK
interface Runnable {
   void run();
}
public class Thread     extends Object
                        implements Runnable {
   void run() {...}
   void start() {...}
   ...
}
public class Compteur implements Runnable{
   public void run() {...}
   }
   public static main() {
        Compteur c = new Compteur();
        new Thread(c).start();}

 }
```

2 EXERCICES

2.1 PROGRAMME EXEMPLETHREAD1.JAVA

Compiler et exécuter le programme exempleThread1.java (pour exécuter plusieurs fois, il est possible d'utiliser **`exec.sh`** qui est fourni)

Le programme crée trois threads qui affichent respectivement "Hello" "World" et "and Everybody". Que pouvez-vous dire à propos de l'ordre de l'affichage? Expliquer.

Le programme utilise le héritage de **`Thread`** pour coder les threads. Changer le code pour utiliser l'interface **`Runnable`**.

2.2 PROGRAMME EXEMPLETHREAD2.JAVA

Compiler et exécuter plusieurs fois le programme exempleThread2.

Voyez-vous tous les affichages? Essayez d'augmenter les valeurs d'attente (le arguments passés lors de la création des threads qui sont utilisés pour les appels à **`sleep`**). Si vous mettez de trop grandes valeurs, vous risquez de ne plus voir aucun affichage. Pourquoi?

2.3 EXEMPLETHREAD3.JAVA

Dans ce programme, nous rajoutons la fonction **`join`** qui force le programme principal d'attendre la terminaison des threads. Cimpiler le programme, exécuter plusieurs fois et s'assurer que tous les affichages sont présents.

2.4 PROGRAMME MULTI-TÂCHE SIMPLE

En s'inspirant des exemples précédents, ecrire une classe **`Compte`** pour la gestion de comptes. Définir des méthodes pour créditer, débiter et consulter le compte. Ecrire également une classe de thread **`CompteModifier`** (en utilisant l'interface **`Runnable`**) qui prend en paramètre un compte et qui modifie son solde. Ecrire un programme principal qui crée un compte, qui crée plusieurs threads **`CompteModifier`** en leur passant en paramètre le compte créé.

Tester, exécuter, observer. Le programme a-t-il le comportement souhaité?

2.5 EXEMPLETHREAD4.JAVA

Dans ce programme, nous manipulons une variable partagée de type **`Compte`**. On peut **`déposer`** de l'argent sur le compte ou **`retirer`** de l'argent. Le programme principal crée 20 threads **`ThreadDeposer`** qui déposent la même somme (10000, puisqu'ils deposent 1000 fois

10), ainsi que 20 threads **`ThreadRetirer`** qui retirent cette même somme. Le résultat à la fin (**`compte.consulter()`**) doit donner 0.

Compiler et exécuter plusieurs fois le programme (en utilisant exec.sh). Y a-t-il des cas où le résultat est différent? Expliquer.

2.6 EXEMPLE THREAD5.JAVA

Pour corriger le problème dans l'exemple précedent, on définit les deux méthodes **`deposer`** et **`retirer`** en tant que **`synchronized`**. Compiler et exécuter le programme. S'assurer que le comportement est correct