TP 1
LES THREADS

Consigne :
- La réalisation des exercices se fera avec un éditeur de texte (ni éclipse, ni netbeans, etc)
- La compilation (javac nomfichier.java) et l'exécution (java nomfichier) en ligne de commande
- La consultation des classes Java à l'adresse *http://download.oracle.com/javase/1.4.2/docs/api/...*

EXERCICE 1 :

```
public class DeuxThreads extends Thread {

    public void run(){
        for (int i=0; i<10; i++){
            System.out.println("nouveau thread");
        }
    }

    public static void main (String[] args){
        DeuxThreads th = new DeuxThreads();
        th.start();
        for (int i=0; i<10; i++){
            System.out.println("Main thread");
             System.out.flush();
        }
    }
}
```

1.1 Essayer de prévoir ce qui devrait s'afficher et comparer à l'exécution

1.2 Modifier le programme précédent pour que les threads créés affichent leur nom dans la boucle (faire appel aux méthodes `setName()` et `getName()` de la classe `Thread`, pour accéder au thread de la méthode `main` utiliser la méthode `currentThread()`).

1.3 Modifier le code précédent afin que le thread main passent la main après un affichage (utiliser la méthode `yield()`)

1.4 Remplacer dans le code précédent l'appel à `yield()` par un appel à `sleep(1)`. Quelle est la différence?

EXERCICE 2 :
Créer une classe `Compteur` permettant d'instancier n threads (le nombre n étant passé en paramètre à la méthode `main`). Chaque thread créé possède un nom de la forme « thread x » avec x allant de 1 à n et compte de 1 à 10. Chaque thread marquera une pause d'une durée aléatoire entre 0 et 5000 millisecondes entre chaque comptage.
Chaque thread affichera à chaque comptage son nom et le nombre (par exemple, "Thread 1: 4") et il affichera un message du type "Thread 1 a fini de compter jusqu'à 10" quand il aura fini.

2.1 Ecrire la classe `Compteur` et tester là en lançant plusieurs compteurs.

2.2 Faire en sorte que le programme `main` se bloque jusqu'à ce que les n threads aient terminé leur exécution (utiliser la méthode `join()` de `Thread`).

EXERCICE 3 :

Examiner le code des deux classes suivantes :

- `Compte` qui correspond à un compte bancaire, et
- `Operation` qui effectue des opérations sur un compte bancaire.

```
public class Compte {
    private int solde=0;
    public void ajouter(int somme){
        solde=solde+somme;
        System.out.println(" ajout de " +somme);
    }
    public void retirer (int somme){
        solde=solde-somme;
        System.out.println(" retrait de " +somme);
    }
    public void operationNulle (int somme){
        solde=solde+somme;
        System.out.print(" ajout de " +somme +",");
        solde=solde-somme;
        System.out.print(" et retrait de " +somme +".");
        System.out.println();
    }
    public int getSolde(){
        return solde;
    }
}
```

```
public class Operation extends Thread {
    private Compte compte;
    public Operation(String nom, Compte compte){
        super(nom);
        this.compte=compte;
    }
    public void run(){
        while(true){
            int i= (int) (Math.random() * 10);
            String nom=this.getName();
            System.out.print(nom);
            this.compte.operationNulle(i);
            int montant=this.compte.getSolde();
            if (montant !=0){
                 System.out.println(nom +"solde =" +montant);
                 System.exit(1);
            }
        }
    }
    public static void main(String[] args){
        Compte c = new Compte();
        for(int i=1; i<=2; i++){
            Operation op= new Operation("op"+i,c);
            op.start();
        }
    }
}
```

3.1 Exécuter la classe `Operation`. Constater le problème : `Operation` effectue des opérations qui devraient laisser le solde du compte inchangé, et pourtant, après un moment, le solde ne reste pas à 0. Expliquer.

3.2 Modifier le code de la classe `Compte` pour empêcher ce problème (utilisation de `synchronized`).

3.3 Dans le code de la classe `Operation`, remplacer l'opération `operationNulle` par les opérations `ajouter` et `retirer` qui devraient elles aussi laisser le solde du compte à 0. Lancer l'exécution et constater le problème. Modifier le code pour que ça marche.

EXERCICE 4 :

Examiner l'algorithme de tri en ordre croissant d'une tranche de tableau comprise entre les éléments d'indices `debut` et `fin`. On remarque que les deux tris qui sont effectués avant la fusion sont indépendants l'un de l'autre et il est donc facile des les faire exécuter en parallèle par deux threads. Voici une version mono-tâche de cet algorithme.

```
public class TriParallele{
    private int[] t;

    private TriParallele(int[] t){
        this.t=t;
    }

    private void permuter (int a, int b){
        int temp=t[a];
        t[a]=t[b];
        t[b]=temp;
    }

    private void trier (int debut, int fin){
        if ((fin-debut)<2){
            if (t[debut]>t[fin]){
                permuter(debut,fin);
            }
        }
        else{
            int milieu=debut + (fin-debut)/2;
            trier(debut,milieu);
            trier(milieu+1,fin);
            trifusion(debut,fin);
        }
    }
```

```
        private void trifusion(int debut, int fin){
            int[] tfusion = new int[fin-debut+1];
            int milieu = (debut + fin)/2;
            int i1=debut;
            int i2=milieu+1;
            int ifusion=0;
            while (i1<=milieu && i2<=fin){
                if (t[i1]<t[i2]){
                    tfusion[ifusion++]=t[i1++];
                }
                else{
                    tfusion[ifusion++]=t[i2++];
                }
            }
            if (i1>milieu){
                for (int i=i2; i<=fin; i++){
                    tfusion[ifusion++]=t[i];
                }
           }
           else{
                for (int i=i1; i<=milieu; i++){
                    tfusion[ifusion++]=t[i];
                }
           }
          for (int i=0, j=debut; i<=fin-debut; i++, j++){
              t[j]= tfusion[i];
          }

        }


        public static void main(String[] args){
            int[] tableau={5,8,3,2,7,10,1,12,4};

            TriParallele t = new TriParallele(tableau);
            t.trier(0,tableau.length-1);

            for(int i=0; i<tableau.length; i++){
                System.out.println(tableau[i] + " ; ");
            }
            System.out.println();
        }

    }
```

4.1 En utilisant la méthode `join()`, coder une version multi-tâche de cet algorithme. N'oublier pas d'attendre la fin de l'exécution des deux threads avant d'afficher le résultat final.

4.2 Coder une nouvelle version en utilisant cette fois-ci `wait()` et `notify()` au lieu de `join()`.