TD nº 1 - Tâches

Dans ce TD, nous allons voir comment les processus sont exécutés quand on utilise *Java*. On étudie également, comment implémenter en *Java* une application qui exécute plusieurs tâches en paralléle à l'aide de *threads*.

Remarque : Ce TD servira de base à la plupart des autres TD de programmation paralléle. N'hésitez donc pas à poser des questions pour bien comprendre les différents concepts utilisés et essayez d'écrire un code clair et bien commenté.

Exercice 1. *Processus*

Dans ce premier exercice, on veut examiner l'exécution des programmes écrits en *Java*.

1. Pour cela, créer un programme (une classe) simple qui réalise une tâche (main) simple :
— le programme attend un entier I en argument (4 par exemple)
— l'exécution affiche la valeur de $I + i$ pour $i = 0, .., 99$ dans chaque 2 seconds

R.

```
public class ex1 {
    private String nbr;
    // constructeur
      ex1(String nb) {
        this.nbr=nb;
    }
    public void coco() {
     for (int i=0; i <100;i++) {
            System.out.print("le processus " + nbr + ":");
            System.out.println("le compteur est \'a " + i + ".");
            try {
                Thread.sleep(500); // milliseconds
            } catch (InterruptedException e) {
 //                return;
            }
        }
    }
    public static void main(String args[]) {
           if (args.length != 1) {
System.out.println("Usage :ex1 ");
System.exit(1);
}
        ex1 lec = new ex1(args[0]);
     lec.coco();
    }
} // fin ex1
//  Pour tester en arriere plan :  java ex1 100 &
// java ex1 200 &
// java ex1 300 &
```

2. Compiler la classe et lancer 3 fois avec des valeurs différentes (100, 200 et 300).

3. Comment les programmes `.java` sont exécutés ?

R.

C'est le code dans les fichiers .class qui est execute et c'est la machine virtuelle qui gere l'execution.
Chaque execution commence par le lancement d'une JVM. Une machine virtuelle execute une classe a la fois

4. Comment faire que les trois programmes soient exécutés en parallèle ?

R.

Il faut lancer 3 machines...
Par exemple : lancer les machines Java en arriere plan (pour les lancer plus rapidement, on peut ecrire un script shell...

5. Comment la concurrence entre les processus est gérée ? Comment l'ordonnencement des processus est réalisé ?

R.

Comme c'est la machine virtuelle qui gere l'execution, la concurrence est entre les differentes occurrences de la JVM. C'est le scheduler du syst\'eme qui ordonne...
Dans l'exemple, c'est pas ex1 qui est lance 3 fois, mais java.

Des eventuelles concurrences peuvent se produire, si les programmes utilisent des res

Dans la suite des exercices, on a besoin de simuler les activités des tâches. Pour cela, on va prendre une classe `Activite` qui contient une méthode `faire()`. C'est cette méthode qui peut simuler une activité (malgré le fait qu'ici l'activité corresponde principalement à des `sleep`s... :-))

```
public class Activite {
    private int id;    // un identifiant
    private int delai;
    private int duree;
    private static char[] S1 = {' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
        ' ',' ',' ',' ',' ',' ',' ',' '};
    private  String B; // pour affichage
    // constructeur
    Activite (int nb, int pause, int lng) {
        id=nb;
  delai = pause;
  duree = lng;
  B = new String(S1,0,id); // la longueur de B est proportionnelle a id
    }
    public void faire() {
  double f = 1.3333;  // une valeur arbitraire
        for (int nombre=1; nombre <duree; nombre++) {
    System.out.println("T" + id + " \t|" + B + 'x'); // on laisse une trace
       for (int i=1; i <10000000; i++) f *= 1.000001;  //  pour consommer le temps
            try {
                Thread.sleep(delai); // pause en milliseconds
            } catch (InterruptedException e) {
            }
        }
  System.out.println("Fin activite T" + id);
    }
}
```

Exercice 2. *Threads avec heritage de la classe Thread*

Ici, on veut réaliser un programme multi-tâche organisé en 4 tâches :

Tâche principale (Application)	Tâche T1	Tâche T2	Tâche T3
Création des tâches T1, T2, T3 (Lancer T1, T2, T3)	T1 » activité durant 30 * sleep de 100 msec	T2 » activité durant 30 * 200 msec	T3 » activité durant 10 * 500 msec

1. Créer une classe `TH` qui hérite de la classe `Thread` et qui est paramétrée par son identifiant (un entier initialisé depuis son constructeur). Dans sa méthode `run()`, il lance l'activité d'un objet de type `Activite`. Cet objet affiche NB fois l'identifiant de la tâche et fait une pause supérieure à PAUSE miliseconde entre deux affichages (il doit être construit par le constructeur `Activite(ID, PAUSE, NB)`, où ID est l'identifiant de la tâche).

2. Dans sa méthode principale (`main()`), créer et exécuter directement 3 occurrences de cette classe. Expliquez les résultats.

R.

```
// Fichier: TH.java ExempleÂă: QUATRE THREADS SIMPLES
public class TH extends Thread {
    private Activite a;
    TH (int id,int pse, int nb) {     // constructeur
        a = new Activite(id, pse, nb);
    }
    public void run() {  //activites
    a.faire();
    }
    public static void main(String args[]) {
        TH T1 = new TH(1, 100, 30);
        TH T2 = new TH(2, 200, 30);
        TH T3 = new TH(3, 500, 10);
       System.out.println("Etats  T1 : " + T1.getState() + " T2 :  " +
                          T2.getState()+ " T3 :  " + T3.getState());
        T1.start();
        T2.start();
        T3.start();
System.out.println("mainÂă: T1, T2, T3 sont lancees" );
       for (int k=0; k <15; k++) {
           System.out.println("Execution  T1 : " + T1.getState() + " T2 :  " +
                              T2.getState()+ " T3 :  " + T3.getState());
            try {
                Thread.sleep(200); // milliseconds
                } catch (InterruptedException e) {   }
         }
  System.out.println("mainÂă: termine" );
        try {
            T1.join();        T2.join();         T3.join();
            } catch (InterruptedException e) {     }
  System.out.println("mainÂă: joints" );
       System.out.println("Etats  T1 : " + T1.getState() + " T2 :  " +
                          T2.getState()+ " T3 :  " + T3.getState());
    }
} // fin TH
```

3. Etudier les méthode `getState()` et . Regarder la documentation. Modifier votre programme pour que le `main()` attend la fin de l'exÃl'cution des threads crÃl'Ãl's. Pour voir l'évolution des threads : afficher les états
— après la création des threads par la méthode `getState()`
— plus tard, après le lancement des threads (par exemple, 15 fois, dans chaque 200 miliseconds)
— après la fin del l'exécution des threads (après le retour des méthodes `join()`)

R.

```
Voir la solution dans le code ci-dessus.

Les affichages : (un exemple)
molnar@HP-8570p:~/Documents/enseignement/archi4_PPN/2019/TD1$ java TH
Etats  T1 : NEW T2 :  NEW T3 :  NEW
mainÂă: T1, T2, T3 sont lancees
Execution  T1 : RUNNABLE T2 :  RUNNABLE T3 :  RUNNABLE
T1  | x
T3  |   x
T2  |  x
T1  | x
T3  |   x
T2  |  x
Execution  T1 : TIMED_WAITING T2 :  TIMED_WAITING T3 :  TIMED_WAITING
T1  | x
T3  |   x
T2  |  x
T1  | x
T3  |   x
T2  |  x
Execution  T1 : RUNNABLE T2 :  RUNNABLE T3 :  RUNNABLE
T1  | x
T3  |   x
T2  |  x
Execution  T1 : TIMED_WAITING T2 :  TIMED_WAITING T3 :  TIMED_WAITING
T1  | x

...
Execution  T1 : RUNNABLE T2 :  RUNNABLE T3 :  RUNNABLE
T3  |   x
T1  | x
T2  |  x
mainÂă: termine
T3  |   x
T1  | x
T2  |  x
T3  |   x
T1  | x
T2  |  x
T3  |   x
T1  | x
T2  |  x
T3  |   x
T1  | x
T2  |  x
Fin activite T3
Fin activite T1
Fin activite T2
```

```
mainÂă: joints
Etats  T1 : TERMINATED T2 :  TERMINATED T3 :  TERMINATED
```

4. Comment l'ordonnancement des tâches se passe-t-il ?

R.

```
Ici, c'est l'ordonnanceur de la JVM qui gere les threads.
Expliquer les resultats de  l'affichage. L'execution des taches commence aleatoiremen
le passage d'une tache a l'autre est aussi aleatoire... mais les sleep() sont prepond
```

Exercice 3. *Threads en implémentant l'interface Runnable*

Même exercice que le précédant, mais ici, les threads sont créés à partir de l'interface `Runnable`.
Indication : pour créer un thread, la classe possède un constructeur :
`public Thread(Runnable rnb);`

1. Ecrivez et testez la classe `TR` similaire à `TH` mais qui implémente `Runnable`.

2. Créez un deuxième constructeur `TR (int ident, int ps, int lg)` pour que l'objet affiche `lg` fois l'identifiant de la tâche et qu'il fasse une pause supérieure à `ps` millisecondes entre deux affichages (il doit utiliser le constructeur `Activite(ID, ps, lg)`, où ID est l'identifiant de la tâche).

R.

```
// Fichier: TR.java Exemple :  THREADS A PARTIR DE RUNNABLE
public class TR implements Runnable {
    private Activite a;
    TR (int nb) {     // constructeur
        a = new Activite(nb, 100, 30);
    }
    public void run() {  //activites
    a.faire();
    }
    public static void main(String args[]) {
        TR e1 = new TR(1);
        TR e2 = new TR(2);
        TR e3 = new TR(3);
        Thread T1 = new Thread(e1);
        Thread T2 = new Thread(e2);
        Thread T3 = new Thread(e3);
       System.out.println("Etats  T1 : " + T1.getState() + " T2 :  " +
                          T2.getState()+ " T3 :  " + T3.getState());
        T1.start();
        T2.start();
        T3.start();
System.out.println("mainÂă: T1, T2, T3 sont lancees" );
       for (int k=0; k <15; k++) {
           System.out.println("Execution  T1 : " + T1.getState() + " T2 :  " +
                              T2.getState()+ " T3 :  " + T3.getState());
            try {
                Thread.sleep(200); // milliseconds
                } catch (InterruptedException e) {   }
         }
  System.out.println("mainÂă: termine" );
```

```
         try {
             T1.join();        T2.join();         T3.join();
             } catch (InterruptedException e) {     }
  System.out.println("mainÂă: joints" );
       System.out.println("Etats  T1 : " + T1.getState() + " T2 :  " +
                          T2.getState()+ " T3 :  " + T3.getState());
    }
} // fin TR
```

3. Cette question permet l'étude de l'ordonnanceur. Pour creer une classe `Activite2`, dans la méthode `faire()` de votre classe `Activite` remplacez la partie

```
          try {
               Thread.sleep(delai); // milliseconds
           } catch (InterruptedException e) {  }
```

par

```
               Thread.yield();
```

— Créez un programme qui génère 20 tâches basées sur `Activite2` et qui les lance pour l'exécution. Analysez les traces.
— Que fait la méthode `yield()` ?
— Délégez maintenant les activités à `Activite` uniquement pour les tâches de nom inférieur à 10 et regardez les résultats.
— Pour une troisième version, supprimez `yield()` et `sleep(10)` et regardez les résultats.

R.

```
public class Activite2 {   // IDEM que Activite sauf :
...
    public void faire() {
double f = 1.3333;  // une valeur arbitraire
        for (int nombre=1; nombre <duree; nombre++) {
System.out.println("T" + id + " \t|" + B + 'x'); // on laisse une trace
     for (int i=1; i <10000000; i++) f *= 1.000001;  //  pour consommer le temps
                Thread.yield(); // sortie <<<<<=================
        }
System.out.println("Fin activite T" + id);
    }
}
//=========================================
// Fichier: TT.java ExempleÂă:  THREADS TESTS
public class TT implements Runnable {
    private Activite2 a;   // <<< ======
    TT (int nb) {     // consTTucteur
        a = new Activite2(nb, 100, 30);
    }
    public void run() {  //activites
    a.faire();
    }
} // fin TT
//=========================================
class TraceTest {  // creer 20 threads
    public static void main(String args[]) {
     Thread T[] = new Thread[20];
     for (int j=0; j <20; j++) {
        T[j] = new Thread(new TT(j));
```

```
     }
     for (int j=0; j <20; j++) {
       T[j].start();
     }
    System.out.println("mainÂă: les threads sont lancees" );

         try {
             for (int j=0; j <20; j++)
                T[j].join();
           } catch (InterruptedException e) {     }
    System.out.println("mainÂă: termine" );
  }
}
//=========================================
* yield() rend la main a l'ordonnanceur. Les elections sont
  faites apres chaque affichage.
* Difficile a voir une difference significative avec yield() et sans...
```

R.

```
class TraceTest2 {
    public static void main(String args[]) {
     Thread T[] = new Thread[20];
     for (int j=0; j <10; j++) {
        T[j] = new Thread(new TR(j));  // <<== TR avec Activite
     }
     for (int j=10; j <20; j++) {
        T[j] = new Thread(new TT(j));  // <<== TT avec Activite2
     }
     for (int j=0; j <20; j++) {
       T[j].start();
     }
    System.out.println("mainÂă: les threads sont lancees" );

         try {
             for (int j=0; j <20; j++)
                T[j].join();
           } catch (InterruptedException e) {     }
    System.out.println("mainÂă: termine" );
  }
}
//=========================================

* sleep() freine considerablement l'execution
```