

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 parallèle à l'aide de *threads*.

Remarque : Ce TD servira de base à la plupart des autres TD de programmation parallèle. 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.

Processus

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 $i = 0, \dots, 99$ dans chaque 2 seconds
2. Compiler la classe et lancer 3 fois avec des valeurs différentes (100, 200 et 300).
3. Comment les programmes `.java` sont exécutés ?
4. Comment faire que les trois programmes soient exécutés en parallèle ?
5. Comment la concurrence entre les processus est gérée ? Comment l'ordonnement des processus est réalisé ?

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 `sleeps...` :-))

1

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` milliseconde 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.
3. Etudier les méthode `getState()` et . Regarder la documentation. Modifier votre programme pour que le `main()` attend la fin de l'exécution des threads créés. 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 millisecondes)
 - après la fin de l'exécution des threads (après le retour des méthodes `join()`)
4. Comment l'ordonnancement des tâches se passe-t-il ?

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

Même exercice que le précédent, 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).
3. Cette question permet l'étude de l'ordonnanceur. Pour créer 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()` ?
- Déléguez 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.