# Programmation événementielle
# TP 1 – Les threads

Dans ce TP, vous travaillerez dans un répertoire source TP1 puis ajouterez un package par exercice. Vous respecterez les principes de la programmation orientée objet vues en première année et deuxième année.

## Exercice 1. Premiers Threads

1. Créez une classe Concours qui possède deux attributs entiers *nb1* et *nb2*.
2. Ajoutez à *Concours* un constructeur sans paramètres qui initialise les 2 entiers à 0.
3. Ajoutez des *getters* permettant de récupérer les valeurs des entiers.
4. Ajoutez des méthodes pub et augmenterNb2 qui augmentent respectivement les valeurs de *nb1* et *nb2* d'un nombre compris entre 1 et 10. On utilisera *(int) (Math.random() * 10) + 1;* pour générer le nombre entre 1 et 10.
5. Créez une classe *Main* qui contient la méthode point d'entrée du programme et instanciez un objet de la classe *Concours.*

Nous allons maintenant créer des Threads qui manipuleront les valeurs de *nb1 et nb2.*

6. Créez une classe *ThreadConcours1* qui hérite de la classe Thread. Cette classe possède en attribut un objet de la classe *Concours c.*
7. Ajoutez le constructeur qui doit prendre en paramètres une chaîne de caractères (le nom du Thread) et un objet de la classe Concours, et initialisez la valeur de l'attribut *c* et du nom du Thread (constructeur classe mère).
8. Redéfinissez la méthode *run()* pour augmenter la valeur de *nb1* de l'attribut *c* en utilisant la méthode a*ugmenterNb1().* On affichera ensuite la valeur de *nb1 (« valeur de nb 1 : valeur »).*
9. Dans la classe *Main,* instanciez un objet de la classe *ThreadConcours* en le nommant « *t1* » et démarrez le Thread (méthode *start()*).
10. Testez le fonctionnement de l'ensemble.

Nous allons maintenant créer un nouveau Thread qui effectue les mêmes traitements mais en utilisant l'interface *Runnable.*

11. Créez une classe *TacheConcours* qui implémente l'interface *Runnable.* Cette classe possède en attribut un objet de la classe *Concours c.*
12. Ajoutez le constructeur qui doit prendre en paramètre un objet de la classe Concours et initialiser la valeur de l'attribut *c*.
13. Redéfinissez la méthode *run()* pour augmenter la valeur de *nb2* de l'attribut *c* en utilisant la méthode a*ugmenterNb2().* On affichera ensuite la valeur de *nb2 (« valeur de nb 2 : valeur »).*
14. Dans la classe *Main,* instanciez un objet de la classe *TacheConcours*.
15. Dans la classe Main, instanciez un objet de la classe *Thread* utilisant la classe *TacheConcours*. Vous nommerez ce *Thread « t2 »*
16. Démarrez le second Thread.

A ce stade, vous avez créé deux threads en exploitant les deux manières possibles.

17. Modifiez les méthodes *run()* pour réaliser 100 fois le traitement augmentant la valeur du nombre.
18. Modifiez la méthode *main* de la classe *Main* pour afficher : « nb1 vainqueur » ou « nb2 vainqueur » si après les 100 augmentations *nb1 (respectivement nb2)* est le plus grand des 2 nombres.
    **Attention,** il faut attendre la fin des threads pour faire la comparaison.

## Exercice 2. Un premier exercice de tri avec join()

Nous souhaitons insérer des éléments dans un tableau puis le trier.

1. Créez une classe Nombres qui contient un ArrayList d'entiers.
2. Dans cette classe, ajoutez une méthode permettant de trier le tableau d'entiers (en utilisant *Collections.sort()*) et une méthode permettant d'ajouter 100 nombres aléatoires entre 0 et 100.
3. Créez 2 threads distincts permettant respectivement d'ajouter et de trier le tableau d'entier. Dans le main, vous appellerez les 2 threads en vous assurant que le thread qui trie le tableau attende la fin du thread d'insertion. Pour cela, il faudra utiliser la méthode *join()*.
4. Affichez le tableau trié à la fin du programme.

## Exercice 3. Même exercice sans join

Reprenez l'exercice 2 pour effectuer le même programme sans utiliser la méthode join(). Pour cela, vous devrez vous appuyer sur les méthodes wait() et notify().

## Exercice 4. La course

C'est la course ! On veut simuler une course entre plusieurs participants. Chaque participant doit additionner tous les chiffres entre 1 et 1000. Le premier à terminer est le vainqueur, les autres ont perdu.

1. Proposez une solution pour représenter le problème avec des Threads. Chaque Thread aura un nom (Thread 0, Thread 1). Le vainqueur devra afficher son nom suivi de c'est moi le vainqueur. Les autres devront afficher « c'est perdu ».
2. Modifiez le code pour que le nombre de participants à la course soit un paramètre et non fixé à 10.
3. Modifiez le code pour faire un tournoi. A chaque tour, le compétiteur le moins bien classé est éliminé. On recommence la course sans lui. Le vainqueur est le dernier compétiteur restant.

## Exercice 5. Tri en parallèle

Nous souhaitons réaliser l'implémentation du Tri Rapide. Le but du tri rapide est le suivant :

- On choisit un pivot (arbitrairement, on choisira ici la dernière valeur du tableau).
- On positionne toutes les valeurs inférieures au pivot en début de tableau.
- On positionne le pivot à sa place.

L'algorithme peut donc se concevoir de cette façon :

```
partitionner(tableau T, entier premier, entier dernier) : entier
        j := premier
        pour i de premier à dernier - 1
                si T[i] <= T[dernier] alors
                        échanger T[i] et T[j]
                        j := j + 1
        échanger T[dernier] et T[j]
        renvoyer j
```

```
tri_rapide(tableau T, entier premier, entier dernier)
    début
        si premier < dernier alors
            pivot := partitionner(T, premier, dernier)
            tri_rapide(T, premier, pivot-1)
            tri_rapide(T, pivot+1, dernier)
        fin si
    fin
```

La fonction du tri rapide peut donc s'effectuer en parallèle pour chaque portion du tableau. A cette fin, nous utiliserons des threads.

Implémentez la solution et testez son fonctionnement sur un tableau de taille raisonnable. On vérifiera le tri par l'affichage des valeurs avant et après tri.