Programmation concurrente

Plan

- Programmation concurrente
- Processus
- Processus et Threads
- Threads En Java
- Synchronisation

Programmation concurrente

- La programmation est distribuée lorsque les processus ne partagent pas la mémoire.

  RMI, Corba, EJB …

- Sinon la programmation est dite parallèle.

  Processuss, Thread ….

Processus

- **Qu'est ce qu'un processus**

Chaque application (excel, word, etc.) exécutée sur un ordinateur lui est associée un processus représentant l'activité de cette application.

À ce processus est associé un ensemble de ressources propres à lui comme l'espace mémoire, le temps CPU etc.

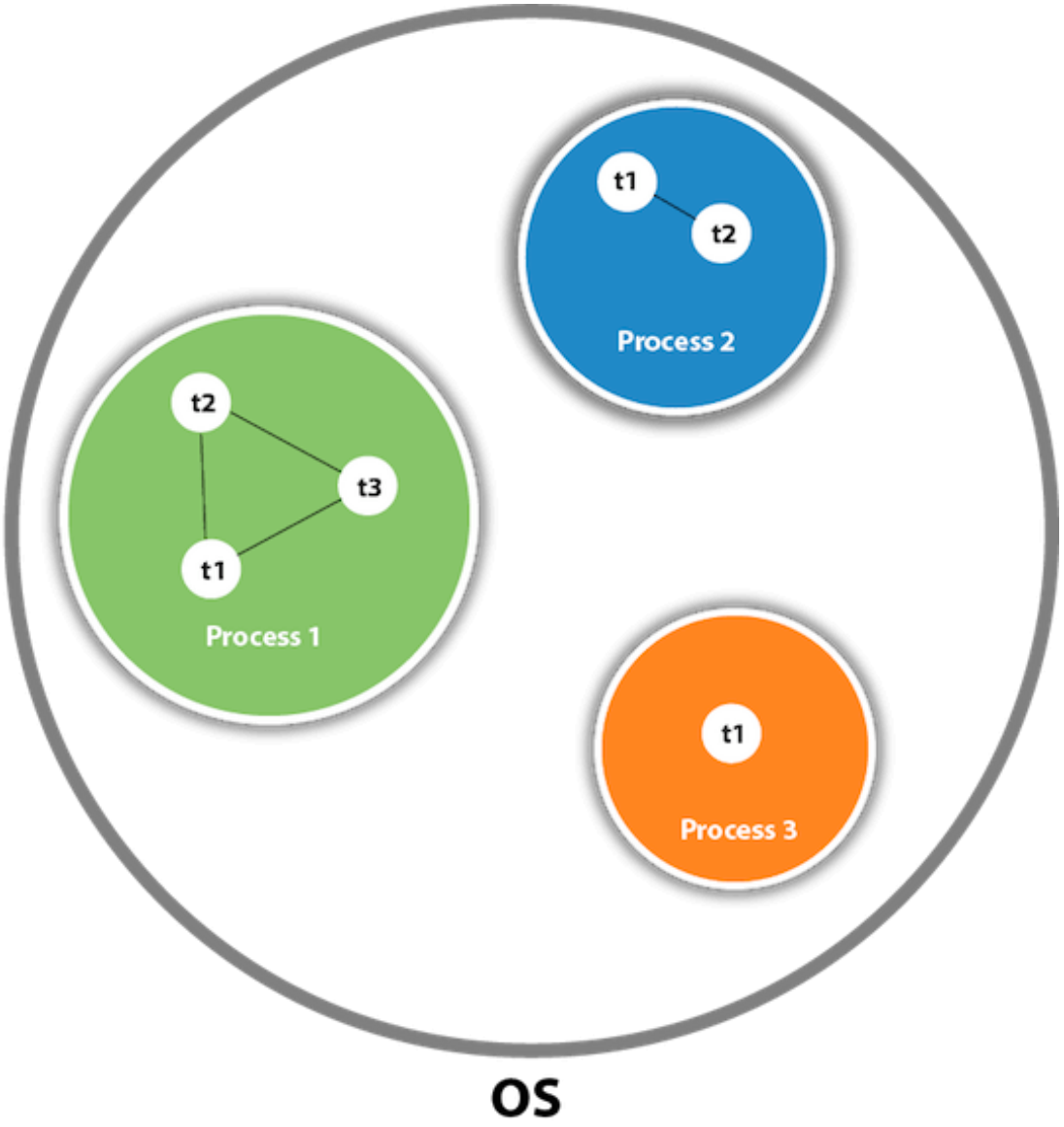
Ces ressources seront dédiées à l'exécution des instructions du programme associé à l'application.

❖ Processucs et les Threads

Dans la programmation concurrente, il existe principalement deux unites d'execution : les processus et les threads.

- L'execution des processus et des threads est gérée par l'OS .
- Un processus possède son propre environnement d'execution (ressources systemes)
- En general on a un processus par application (mais on peut faire cooperer des processus (IPC : Inter Process Communication)
- La plupart des JVM tourne sur un seul processus
- Un thread est souvent appele un processus leger (lightweight process)
- Un thread n'est pas un objet! C'est un sous-processus qui exécute une série d'instructions d'un programme donné.

Processus & Thread



❖ Avantage des processus légers par rapport au processus système

- rapidité de lancement et d'exécution
- partage des ressources système du processus englobant
- simplicité d'utilisation

Thread En Java

❖ Les Threads en Java

La machine virtuelle java (JVM) permet d'exécuter plusieurs traitements en parallèle, pour créer un thread nous avons besoin de décrire le code qu'il doit exécuter et le démarrer par la suite:

- Un thread doit obligatoirement implanter l'interface Runnable

```
public interface Runnable {
        void run();
}
```

- La méthode run doit contenir le code à exécuter par le thread.

Thread En Java

Créer un contrôleur de thread qui permet de démarrer un thread. Ceci est réalisable avec l'une des deux approches suivantes:

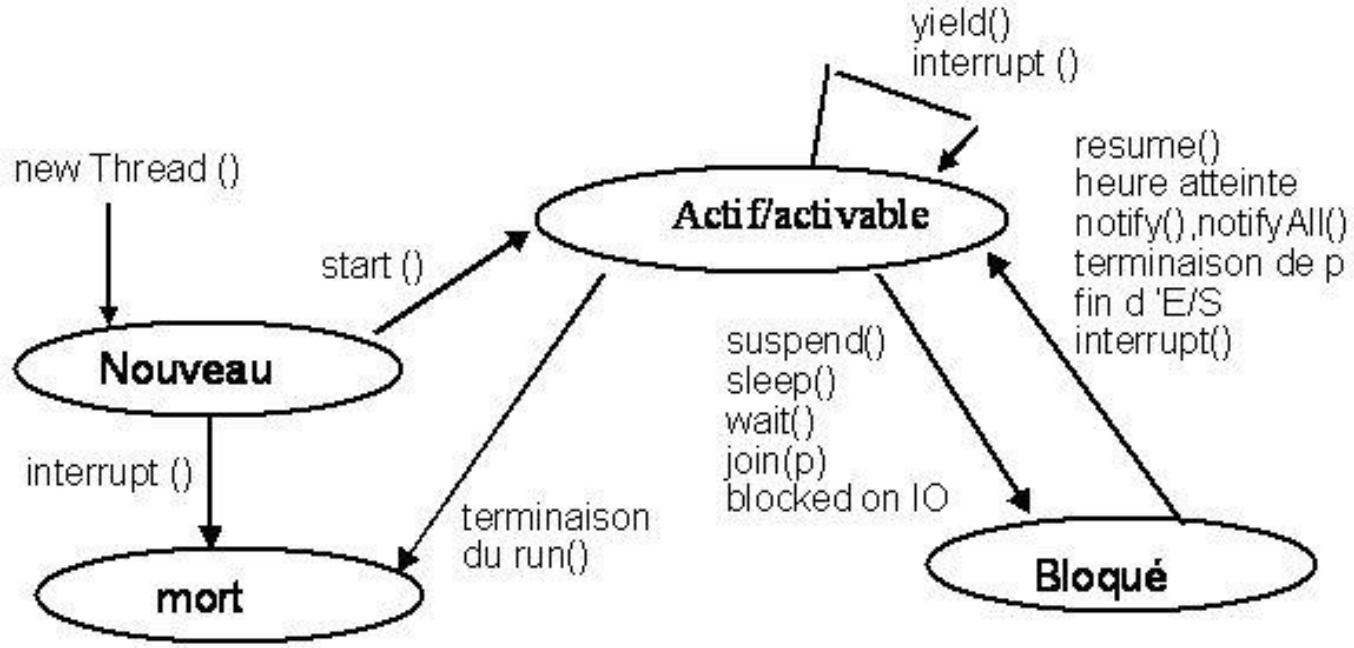
Classe qui dérive de java.lang.thread	Classe qui implemente l'interface Runnable

class ThreadTest extends Thread {
public void run() {
// le code à exécuté
}
public ThreadTest(…) {
// constructeur . avec/sans args.
}
}
	class ThreadTest implements Runnable
	{
	public void run() {
	// le code à executé.
	}
	public ThreadTest(…) {
	// construcateur. avec/sans args.
	}
	}

ThreadTest MonThread = new ThreadTest(…); MonThread.start();	ThreadTest test = new ThreadTest(…); Thread t1 = new Thread(test); t1.start();

Thread En Java

- **Les différents états d'un Thread :**



Au cours de sa vie, un thread passe par plusieurs états (de sa création à sa mort). La figure donne le graphe des états d'un Thread. Quand une transition se révèle être impossible une exception IllegalThreadStateException est déclenchée.

Thread En Java

- **Etat Nouveau**

  Le thread vient d'être créé et initialisé par exécution de la méthode privée init appelée par le constructeur. Il ne dispose pas encore de ressources systèmes. C'est la méthode native start qui les lui fournira. Le thread existe seulement comme objet passif. Le thread sera ultérieurement activé (thread.start()), auquel cas il se retrouve dans l'état Exécutable qui regroupe 2 sous-états : activable et actif.

- **Etat Exécutable : Actif/Activable**

  Un thread dans l'état actif est en cours d'exécution par un processeur. Un thread à l'état activable attend qu'un processeur se libère pour devenir actif. Le choix du thread à rendre actif est fait par l'ordonnanceur. La machine virtuelle Java de SUN ne met pas en œuvre de mécanismes de temps partagé permettant d'éviter, à priorité égale, qu'un thread monopolise un processeur.

  Le programmeur ne dispose que de deux moyens pour influer sur la politique d'ordonnancement : modifier la priorité des threads ou utiliser la méthode yield (le thread cède la main au profit d'un thread de même priorité).

- **Etat Bloqué**

Un thread bloqué ne peut pas être exécuté par le processeur. Il est en état d'attente. Différents cas d'attente existent : attente qu'un certain laps de temps s'écoule, qu'un signal en provenance d'un autre thread arrive, qu'une condition sur des variables partagées soit satisfaite, qu'une donnée devant être lue soit disponible, …

Exécutable --> Bloqué	Bloqué --> Exécutable
sleep(x) // Suspension pendant x ms	Redevient exécutable au bout de x ms
suspend( )	resume( )
wait( ) // synchronisation	notify( ), notifyAll( )
Blocage en attente de fin d'entrée/sortie	Déblocage à la fin de l'entrée/sortie
synchronized // exclusion mutuelle	Libération de l'exclusion mutuelle.
join //attente qu'un autre thread soit terminé	Le thread attendu est terminé

- **Etat Mort**

  Un thread meurt naturellement lorsque la fin de sa méthode run est atteinte. Attention, l'objet n'est plus actif, mais il existe toujours en tant qu'objet passif, appelable via ses méthodes publiques. Il disparaîtra vraiment lorsque le ramasse-miettes s'apercevra que l'objet n'est plus référencé.

  La méthode join appliquée à un objet Thread permet d'attendre que ce thread passe à l'état mort

- **Connaître l'état d'un thread**

  Il peut être utile de connaître l'état d'un thread (mort ou vivant). La méthode isAlive( ) renvoie true si le thread a été démarré (via start( )), mais n'est pas terminé.

  Il faut en dernier lieu constater que le langage ne fournit pas le moyen de faire la différence entre l'état Nouveau et l'état Mort, pas plus qu'entre l'état Exécutable et l'état Bloqué.

- **La priorité d'un thread**

Les threads sont concurrents. Cela signifie que les threads à l'état exécutable sont en compétition les uns avec les autres pour l'obtention du ou des processeurs. C'est le rôle de l'ordonnanceur (scheduler) sous-jacent d'effectuer ce partage.

Lors de sa création, un thread se voit affecter une priorité, celle du thread parent (celui qui l'a créé). Le thread du programme principal possède la priorité 5 (NORM_PRIORITY).

La priorité doit appartenir à l'intervalle défini par 3 attributs constants de la classe Thread : MIN_PRIORITY (qui vaut 1), NORMAL_PRIORITY (qui vaut 5) et MAX_PRIORITY (qui vaut 10). Cette priorité est modifiable par la méthode setPriority de la classe Thread. La priorité d'un thread peut être obtenue par la méthode getPriority.

- **Règle**

Un thread ne doit pas être à l'état actif alors qu'un thread de plus forte priorité est activable.

Thread En Java

❖ La classe Thread

Nous allons présenter quelques méthodes de la classe Thread, l'ensemble de la classe est décrit sur ce lien:

http://java.sun.com/j2se/1.4/docs/api/java/lang/Thread.html

❖ Constructeurs

public Thread();	crée un nouveau Thread dont le nom est généré automatiquement (aléatoirement).
Public Thread (Runnable target);	target est le nom de l'objet dont la méthode run est utilisée pour lancer le Thread.
public Thread(Runnable target, String name);	on précise l'objet et le nom du Thread.
public Thread(String name);	on précise le nom du Thread.

Thread En Java

❖ Méthodes :

Méthodes	Description
void destroy();	Permet de détruire un Thread
String getName();	retourne le nom du Thread.
void interrupt();	interrompt le Thread.
static boolean interrupted();	teste si le Thread courrent a été interrompu.

void join();
void join(long millis);
void join(long millis, int nanos);
	attendre la mort du Thread, ou après un millis de millisecondes, ou millisecondes plus nanosecondes.

void resume();	redémarrer le Thread.
void run();	La méthode contenant le code à exécuter par le Thread.

Thread En Java

Méthodes	Description
void setPriority(int newPriority);	changer la priorité du Thread.
static void sleep(long millis); static void sleep(long millis, int nanos);	mettre en veille le Thread pendant millis millisecondes ou millisecondes plus nanosecondes.
void start();	démarrer un Thread.

TP N° 1

La Synchronisation

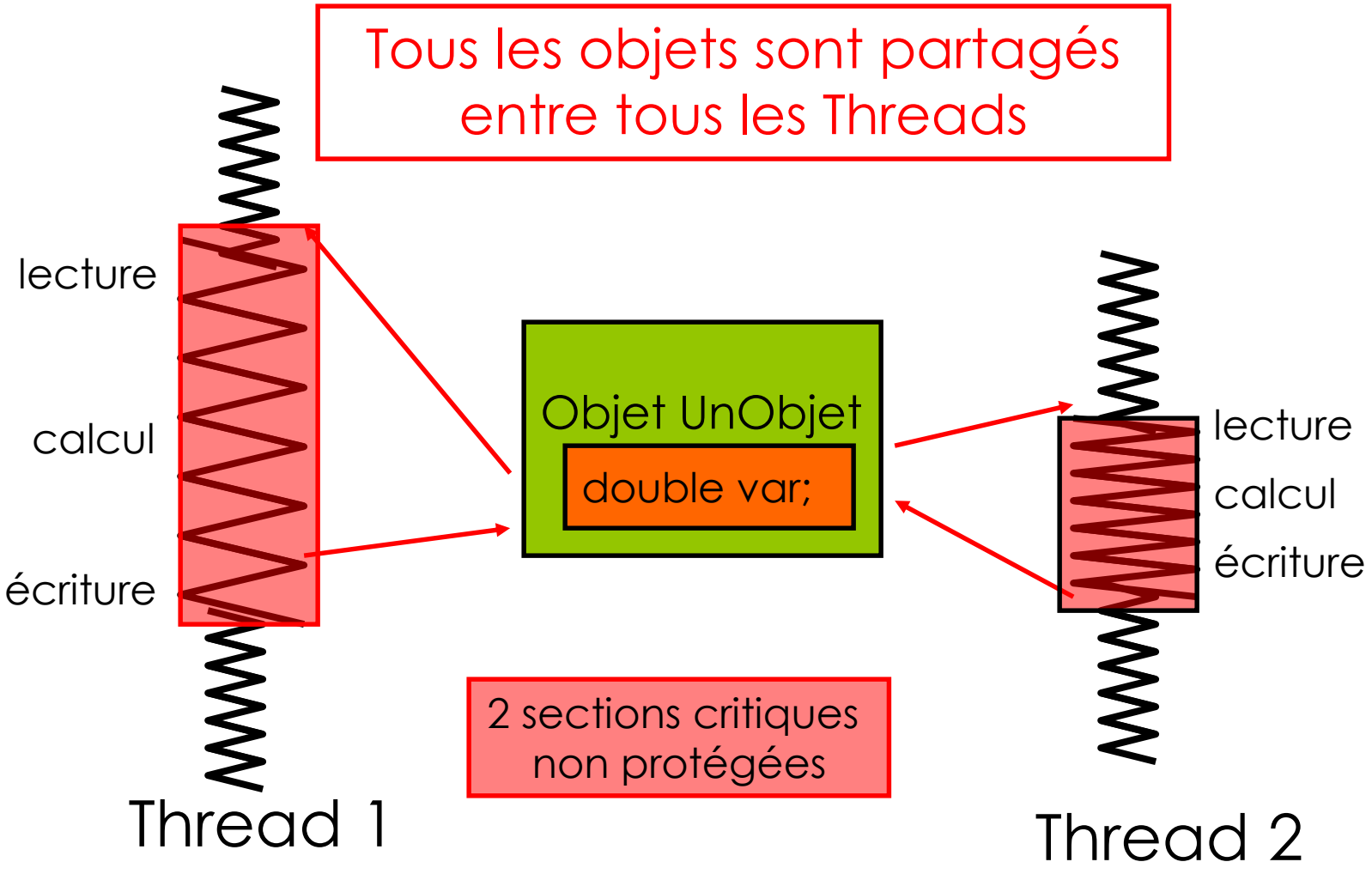
- Soit une classe Compte Bancaire

```
public static void main(String[] args)
 {
      CompteBancaire cp=new CompteBancaire(100);
      Retrait marie = new Retrait(cp);
      Retrait epouse = new Retrait(cp);
      marie.start();
      epouse.start();
 }
```

Solution

- Il faut bloquer le retrait quand un détenteurs du compte commence un retrait (exécution exclusif)
- Solution : placer le bout de code critique dans un bloque garder par le mot synchronized
- Le mot clé synchronized informe la machine que ce bloque ne peut être instancier ou exécuté que par un seul thread à la fois
- Conséquence direct: le Thread qui commence un bloque synchronized à l'exclusivité de l'exécution.

La Synchronisation



Section critique

- Comment protéger une section critique ?
- En java, par un bloc « synchronized » : (bloc, méthode)
- Un seul thread accède au code synchronized à un moment donné.
- Protection efficace mais basique….
  - wait() : met le thread en attente (sans perte de cycles CPU…) et relâche le verrou
  - notify() et notifyAll() : libèrent un ou tous les threads de l'état wait

Solution pour le compte bancaire

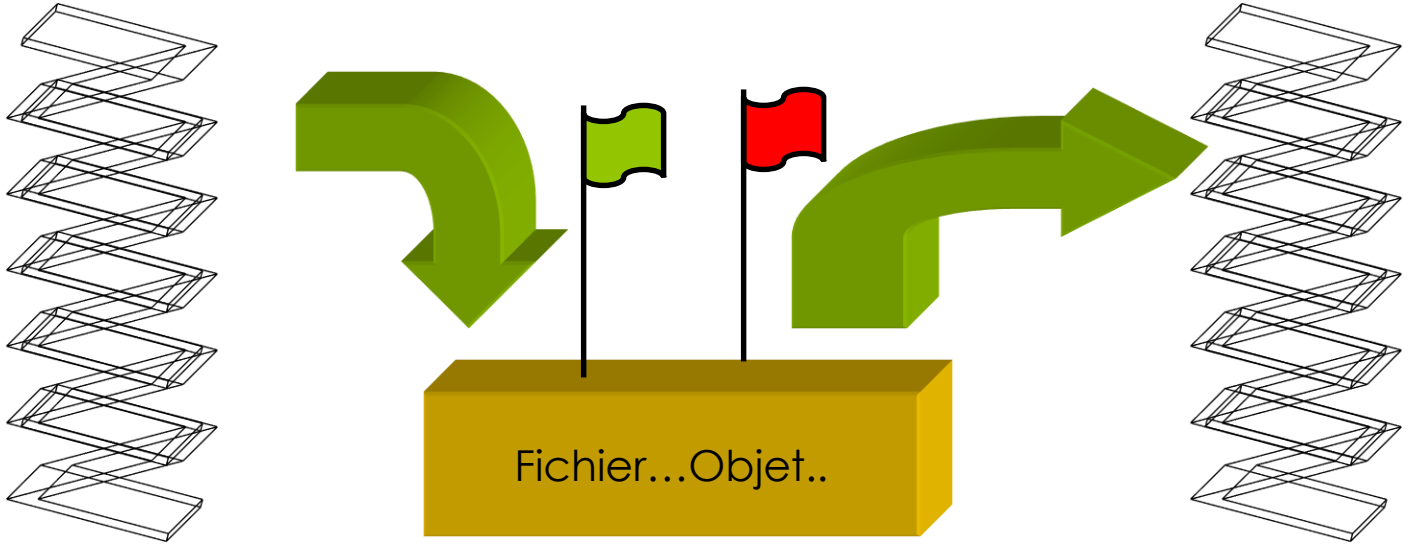
```
public class CompteBancaire {
private float total;
  public CompteBancaire(float init){
          total=init;
   }
  synchronized public boolean retrait(float t){
    if (t <= total) {
          total -=t;
          return true;
    }
    return false;
  }
}
```

Autre type de synchronisation: sémaphore

- Exemple ping pong affiche des suite varié de mot ping et pong : (ping | pong)*
- On veut changer le code de tel façon que le langage généré soit: (ping pong)*
- L'idée est d'utilisé la notion de drapeau ou sémaphore :
- Solution: Chaque thread doit garder son code par une variable qui lui block l'exécution et permet a son concurrent de s'exécuter

Producteur/Consommateur

- Un drapeau indique qu'une valeur a été déposée
- Un drapeau indique qu'une valeur a été consommée



Thread 1
(producteur)

Méthode traditionnelle :
« Attente active »

Thread 2
(consommateur)

Exercice ping pong

- Ecrire une classe Sem qui implémente un sémaphore : elle contient un champ booléen curval et deux méthodes get(boolean val) et set(boolean val).
- La méthode get(boolean val) attend que curval soit égal à val pour continuer
- La méthode set(boolean val) positionne curval à val
- Get et set définissent une section critique

Un problème d'accès concurrent

```
public class ExempleConcurrent extends Thread {
        private static int compte = 0;

        public void run() {
                int tmp = compte;
                try {
                        Thread.sleep(1); // ms
                } catch (InterruptedException e) {
                        System.out.println("ouch!\n");
                        return;
                }
                tmp = tmp + 1;
                compte = tmp;
        }

        public static void main(String args[]) throws InterruptedException {
                        Thread T1 = new ExempleConcurrent();
                        Thread T2 = new ExempleConcurrent();
                        T1.start();
                        T2.start();
                        T1.join();
                        T2.join();
                        System.out.println("compteur=" + compte);

        }
}
```

Les deux threads T1 et T2 accèdent à une même variable partagée compte, travaillent sur une copie locale tmp qui est incrémentée avant d'être réécrite dans compte. Nous avons intercalé un appel à sleep pour simuler un traitement plus long et augmenter la probabilité que le système bascule d'une tâche à l'autre durant l'exécution de run.

Mot clé *synchronized*

Les problèmes d'accès concurrents se règlent en JAVA à l'aide du mot clé **synchronized**, qui permet de déclarer qu'une méthode ou un bloc d'instructions est critique : un seul thread á la fois peut se trouver dans une partie synchronisée sur un objet.