ECOLE NATIONALE D'INGÉNIEURS DE CARTHAGE
2 ING INFO

Programmation Java

2020-2021

PROGRAMMATION CONCURRENTE EN JAVA

INTRODUCTION

- Concurrence = habilité d'exécuter des procédures parallèles
- Notion très importante pour le « Big data analysis »
- Équivalent à avoir beaucoup de CPU ou cœurs dans un même CPU ✉ partage de temps et de ressources.
- Son implémentation haut niveau en Java commence en Java 5. Des avancements ont été proposé en Java 9 dans le package: **java.util.concurrent**
- Java propose deux unités d'exécution: process et thread. Usuellement, le process représente la JVM
- Un process peut avoir plusieurs threads en exécution (au moins le main thread)
- Remarque: on peut paralléliser un stream par:

✓ Utiliser la méthode parallelStream pour une collection

✓ Appliquer méthode parallel à un Stream séquentiel.

INTERFACE RUNNABLE ET CLASSE THREAD

- Interface **fonctionnelle** disposant d'une méthode: **void run().**
- Toute classe implémentant cette interface correspond à un thread. Le code à exécuter lors d'un traitement parallèle doit être implémenté dans run()
- **Thread** est une classe **extends Object implements Runnable**. Parmi ces constructeurs:

✓ Thread ()

✓ Thread (Runnable)

❖ Parmi ces méthodes:

✓ void start(): pour déclencher l'exécution

✓ static void sleep(long time): mise en veille « time » millisecondes

Exercice: afficher d'une façon parallèle des messages (chacun un certains nbr de fois). Vous fixez le délai entre deux affichages successifs du même message.

Avec Thread	Avec Runnable

class Message **extends Thread** {
private String texte ;
private int nb ;
private long attente ;
public Message (String texte, int nb, long attente) {
this.texte = texte ; this.nb = nb ;
this.attente = attente;}
public void run(){
try{
for (int i=0;i<nb;i++) {
System.out.print(texte);
sleep(attente);}}
catch (InterruptedException e){}
// imposée par sleep
}}
	class Ecrit **implements Runnable**{
	private String texte ;
	private int nb ;
	private long attente ;
	public Ecrit (String texte, int nb, long attente) {
	this.texte = texte ; this.nb = nb ;
	this.attente = attente;}
	public void run(){
	try{
	for (int i=0;i<nb;i++) {
	System.out.print(texte);
	Thread.sleep(attente);}}
	catch (InterruptedException e){} // imposée par sleep
	}}

Dans une classe principale (aussi un Thread main)	

public static void main (String args[])
{
Message m1 = new
Message("Bonjour",10,5) ;
Message m2 = new
Message("Bonsoir",12,10) ;
Message m3 = new Message("\
n",5,15) ;
m1.start();m2.start(); m3.start();}
	public static void main (String args[])
	{
	Ecrit e1 = new Ecrit(“Hello",10,5) ;
	Ecrit e2 = new Ecrit(“Go!!!",12,10) ;
	Ecrit e3 = new Ecrit("\n",5,15) ;
	Thread t1 = new Thread(e1);
	t1.start();
	Thread t2 = new Thread(e2);
	t2.start();
	Thread t3 = new Thread(e3);
	t3.start();

- **voir exécution des projets « exempleEcritThread » et «exempleEcritRunnable » sous NetBeans IDE**

✓ L’appel direct de « run » oblige de terminer le premier Thread puis le deuxième…

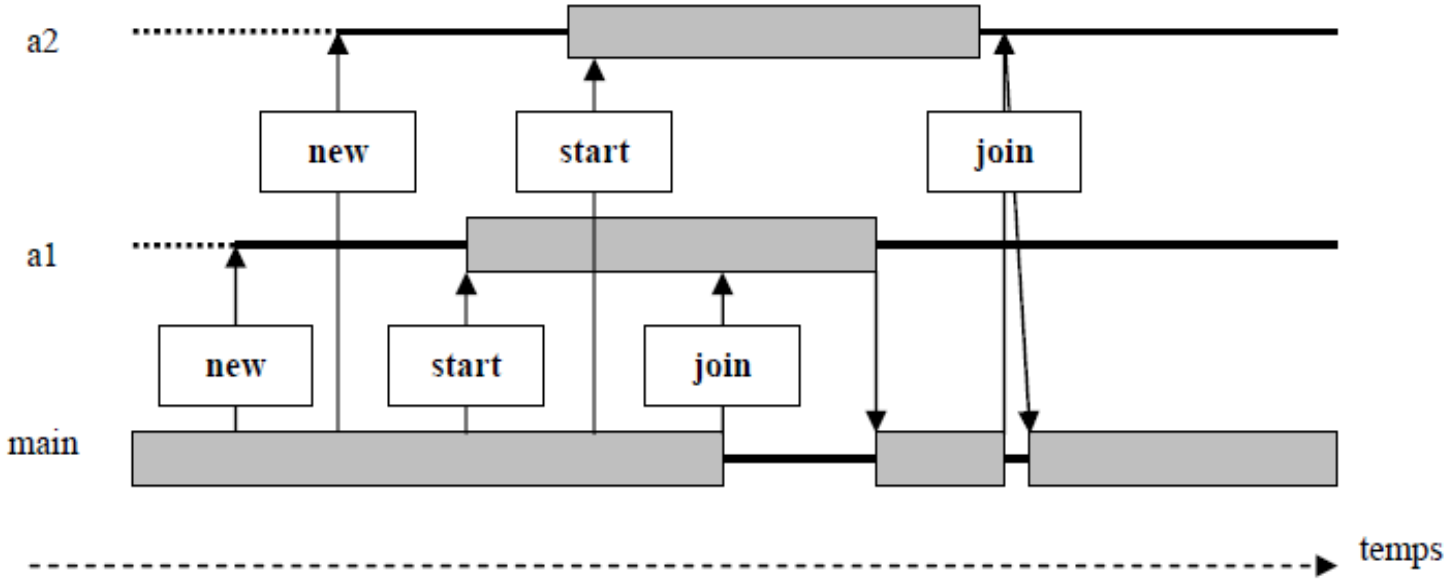
✓L’appel de « start » se fait une seule fois sinon une exception sera générée : IllegalThreadStateException

✓Avec la deuxième méthode on peut utiliser start() directement avec un objet Ecrit. Seulement, il fallait redéfinir au niveau de la classe Ecrit par :

public void start(){Thread t=new Thread(this) ; t.start() ;}

MÉTHODES JOIN() ET INTERRUPT() DANS THREAD

- La méthode Join() permet le blocage du thread main jusqu'à la fin du thread appelant



- **voir projet « exempleJoin » sous NetBeans IDE**
- **voir projet « exempleInterrupt » sous NetBeans IDE**

PRIORITÉ ET ETAT

- Une valeur de priorité est affectée à chaque thread et détermine sa priorité d'accès au temps CPU.
- Trois champs statiques dans une classe Thread qu'on peut accéder: MIN_PRIORITY (correspond à 1) à MAX_PRIORITY (correspond à 10), NORM_PRIORITY (priorité par défaut qui correspond à 5)
- Méthodes de priorité:

✓ **setPriority(int i)**

✓ **int getPriority()**

✓ **setDaemon(boolean)** appelée avant start() permet de faire du thread un démon, processus de basse priorité qui tourne en tâche de fond.

❖ Méthode getState() retourne l'état (State) du Thread. Exemples: state = RUNNABLE, state = TERMINATED, state =BLOCKED, State = WAITING, state = New...

❖ **voir projet « exempleState » sous NetBeans IDE**

Exercice: L'objectif est de simuler une course des animaux. On fixe pour chaque animal sa vitesse et le temps maximal de la course. A la fin nous aurons la distance parcourue et l'état mort ou vivant. A toute étape, on définit l'état des threads.

- **voir projet « exempleCourse » sous NetBeans IDE**

SYNCHRONISATION ENTRE THREADS

- Problème: gérer l'accès aux ressources communes
- La synchronisation est un élément essentiel dès lors que vous utilisez plusieurs threads.
- Sans synchronisation, il est impossible de développer une application robuste, quel que soit l'entrelacement de l'exécution des threads.

EXEMPLE:

```
private static class Calculator {
private double n;
public double calculer (int i) {
DoubleStream.generate(new Random()::nextDouble).limit(10);
synchronized(this){
this.n=2.0*i;
Return Math.sqrt(this.n); };}
```

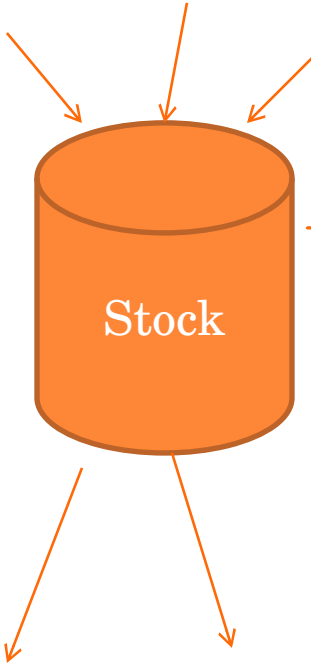
- A un instant donné, une seule méthode synchronisée peut accéder à un objet donné. En effet, l'environnement gère « un verrou » (ou une clé) unique mettant l'accès à un objet qui possède au moins une méthode synchronisée.
- Le verrou est attribuée à la méthode synchronisée appelée pour l'objet et n'est restituée à qu'à sa sortie.
- Lors de la prise de ce verrou, aucune **autre méthode synchronisée** ne peut le prendre
- On peut définir une instruction ou bloc d'instructions comme synchronisée qui prend le verrou sur un objet : **synchronized (objet) { instructions}**

❖ **voir projet « exempleSynchro » sous NetBeans IDE**

Problème de situation d'inter-blocage :
le thread t1 prend le verrou de l'objet o1 et attend le verrou de l'objet o2 et vice versa pour un deuxième thread t2.

Exemple de Réserve

Threads pour le remplissage



- ✓ L'ajout prend un certain temps
- ✓ pas d'ajout au même instant

Threads pour l'épuisement

- ✓ L'épuisement prend un certain temps
- ✓ pas d'épuisement au même instant
- ✓ pas d'épuisement si on dépasse le stock donc il faut attendre dans ce cas

❖ **voir projet « exempleNotif » sous NetBeans IDE**

SÉMAPHORES

- Les sémaphores sont utilisés pour restreindre l'usage des ressources communes partagées en programmation concurrente.
- Un sémaphore permet d'autoriser à un ensemble de threads à accéder à des ressources mutuelles. Le sémaphore gère un ensemble de permis, en nombre fixé, et accorde ces permis aux threads qui en font la demande.

❖ **voir projet « exempleSemaphore » sous NetBeans IDE**