Université Abdelhamid Mehri – Constantine 2

2020/2021. Semestre 1

DAC

– Cours 1 –

Les threads java

Introduction

Staff pédagogique			
Nom	**Grade**	**Faculté/Institut**	**Adresse e-mail**
Meriem Belguidoum	MCB	NTIC	meriem.belguidoum@univ-constantine2.dz

Étudiants concernés			
Faculté/Institut	**Département**	**Niveau**	**spécialité**
NTIC	TLSI	Licence 3	GL

Objectifs du cours 1

- Apprendre un programmer les thread java
- Apprendre le principe de la programmation concurrente
- Apprendre a programmer la synchronisation des threads et gestion des ressourses critiques

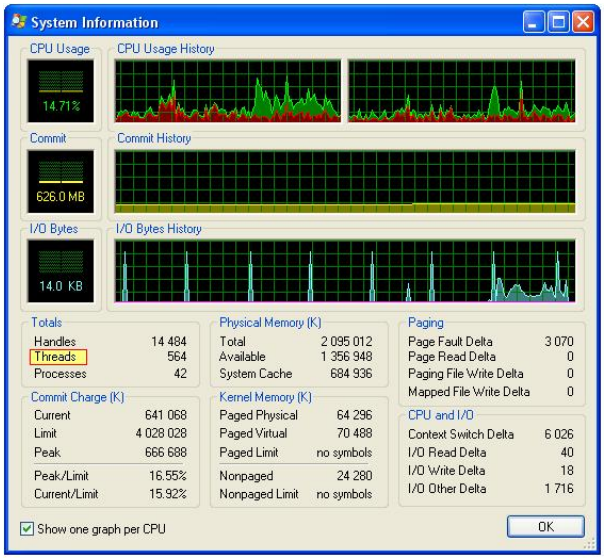

Figure 1: Gestionnaire de processus

1 Introduction

Un système **multi-tâches** est capable d'exécuter plusieurs programmes en parallèle sur une même machine. La plupart du temps, une machine n'a qu'un seul processeur qui ne peux exécuter qu'un seul programme à la fois. Un programme en cours d'exécution est appelé **un processus**. La plupart des systèmes d'exploitation sont équipés d'un ordonnanceur de tâches qui donne à tour de rôle le processeur aux processus. Chaque processus est activé de façon cyclique et pendant une courte durée, ce qui donne à l'utilisateur l'impression que plusieurs processus sont en cours d'exécution. Un processus est donc un ensemble d'instructions avec son état d'exécution (pile, registres, pc, tas, descripteurs d'E/S, gestionnaires de signaux, etc.). Il existe deux classes principales de processus :

- **Processus lourd** (ou tâche ou processus) : qui ne partage pas son état sauf des espaces mémoire partagés déclarés explicitement
- **Processus léger** (ou *thread*) : qui partage son tas, ses descripteurs et ses gestionnaires

2 Notion de thread

2.1 Qu'est ce qu'un thread ?

Un thread (appelée aussi processus léger ou activité) est un fil d'instructions (un chemin d'exécution) à l'intérieur d'un processus. Les threads sont des processus légers qui partagent un ensemble de codes, données, temps processeur, mémoire et ressources au sein d'un processus lourd qui les héberge. Les threads d'un même processus partagent le même espace d'adressage, les mêmes variables d'environnement, les mêmes données, etc. contrairement aux processus lourd.
Les processus légers se caratérisent par rapport aux processus lourd par : la rapidité de lancement et d'exécution, le partage des ressources système du processus englobant et la simplicité d'utilisation.

2.2 Etats des threads

Un processus peut être dans différents états (voir figure 2) :

- **création** : le processus est créé et attend d'être lancé.
- **exécutable** : le processus est prêt à s'exécuter, mais n'a pas le processeur (occupé par un autre processus en exécution) ;
- **en cours** : le processus est en cours d'exécution sur le processeur du système ;

- **bloqué** : le processus est bloqué par une ressource ou autre.
- **détruit** : le processus est détruit ou terminé

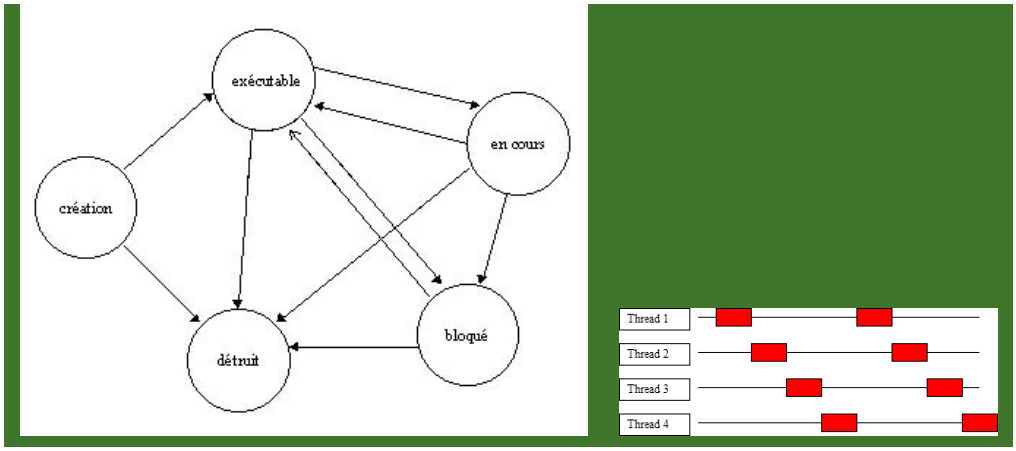

Figure 2: Etat des threads

2.3 Les threads en java

Les threads Java sont des processus légers qui possèdent au moins un thread (qui exécute le programme principal main()) et qui représente une partie intégrante du langage JAVA et géré grâce à la classe Thread. Les programmes qui utilisent plusieurs threads sont dits multithreadés.
La machine virtuelle java (JVM) permet d'exécuter plusieurs traitements en parallèle (en pratique, ils s'exécutent par "tranche" et en alternance sur le processeur).
JVM (Java Virtual Machine) est un processus lourd (voir la figure 3), le thread principal représente le main() et les autres pour : ramasse-miettes, gestion du clavier, etc. Tous les threads Java partagent la mémoire (les objets).

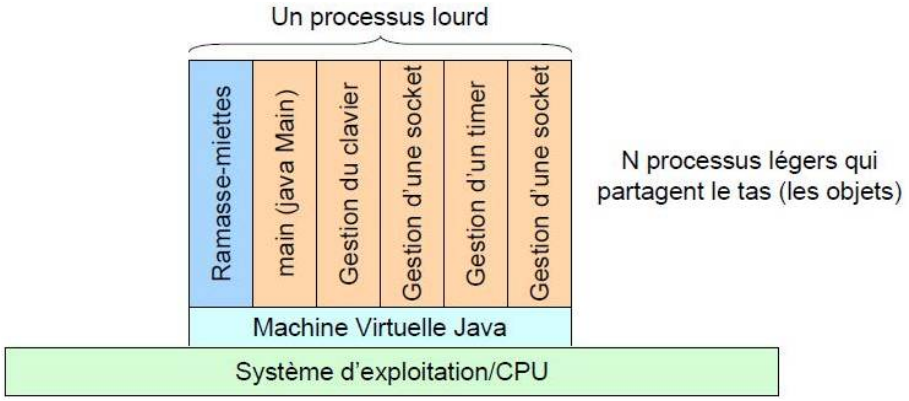

Figure 3: Java Virtual Machine

3 Gestion des threads

3.1 Comment créer un thread ?

pour réaliser plusieurs tâches en parallèle, il est possible de créer des threads au sein des applications Java Nous pouvons définir un thread de deux manières :

- créer un thread par un **héritage** de la classe `java.lang.Thread` (l'héritage multiple n'est pas autorisé en java)

- créer un thread en **implémentant l'interface** `java.lang.Runnable`.

Dans les deux cas, on doit avoir une méthode run() contenant le corps de l'exécution du thread. Le thread est lancé avec l'appel de la méthode start().

3.1.1 Création et démarrage d'un thread avec `extends Thread`

Il faut créer une classe qui hérite de la class Thread et redéfinir la méthode `run()` pour y inclure les traitements à exécuter par le thread.

```
class MonThread extends Thread {
    MonThread() {
            ... code du constructeur ...
    }

    public void run() {
            ... code à exécuter dans le thread ...
    }
}
```

Pour créer et exécuter un tel thread, il faut instancier un objet et appeler sa méthode start() qui va créer le thread et elle-même appeler la méthode run().

```
MonThread p = new MonThread();
p.start();
```

L'appel de la méthode `start()` passe le thread à l'état "prêt", lorsque le thread démarre, JAVA appelle sa méthode `run()`. Un thread se termine lorsque sa méthode run() termine.

```
class MonThread extends Thread {
public void run() {
System.out.println("I'm a thread!");
}
public static void main(String args[]) {
// MonThread sera un Thread nommé automatiquement "Thread-1"
Thread t = new MonThread();
// MonThread.run() est démarré dans un nouveau thread après
//  l'appel de start()
t.start();
}
}
```

3.1.2 Création et démarrage d'un thread avec `implements Runnable`

```
class MonThread implements Runnable {
public void run() {
System.out.println("I'm a thread!");
}
public static void main(String args[]) {
// Un thread possède optionnellement un nom symbolique
Thread t = new Thread(new MonThread(), "My Thread");
// MonThread.run() est démarré dans un nouveau thread après
//l'appel de start()
t.start();
System.out.println("Ce code s'exécute en // de run()");
}
}
```

Déclarer une classe qui implémente l'interface `Runnable` et redéfinir sa seule et unique méthode `run()` pour y inclure les traitements à exécuter dans le thread. La classe Thread a un constructeur new Thread(Runnable). L'argument du constructeur est donc toute instance de classe implémentant cette méthode `run()`. La classe se déclare comme dans l'exemple précédent, mais on implémente `Runnable` au lieu d'hériter de `Thread`.

```
class MonThread2 implements Runnable {
    MonThread2() {
        ... code du constructeur ...
    }
    public void run() {
        ... code à exécuter dans le thread ...
    }
}
```

Pour créer et lancer un thread, on crée d'abord une instance de `MonThread2`, puis une instance de `Thread` sur laquelle on appelle la méthode `start()` :

```
MonThread2 p = new MonThread2();
Thread T = new Thread(p);
T.start();
```

3.1.3 Quelle technique choisir ?

	Avantages	Inconvénients
extends java.lang.Thread	Chaque thread a ses données qui lui sont propres	On ne peut plus hériter d'une autre classe
implements java.lang.Runnable	L'héritage reste possible. En effet, on peut implémenter autant d'interfaces que l'on souhaite.	Les attributs de votre classe sont partagés pour tous les threads qui y sont basés. Dans certains cas, il peut s'avérer que cela soit un atout.

3.2 Thread : Quelques propriétés et méthodes

- `void destroy()` : met fin brutalement au thread.
- `int getPriority()` : renvoie la priorité du thread.
- `void setPriority(int)` : modifie la priorité d'un thread
- `ThreadGroup getThreadGroup()` : renvoie un objet qui encapsule le groupe auquel appartient le thread.
- `boolean isAlive()` : renvoie un booléen qui indique si le thread est actif ou non.
- `boolean isInterrupted()` : renvoie un booléen qui indique si le thread a été interrompu.
- `currentThread()` : donne le thread actuellement en cours d'exécution.
- `setName()` : fixe le nom du thread.
- `getName()` : nom du thread.
- `isAlive()` : indique si le thread est actif(true) ou non (false).
- `void resume()` : reprend l'exécution du thread() préalablement suspendu par `suspend()`.
- `void run()` : méthode déclarée par l'interface Runnable : elle doit contenir le code qui sera exécuté par le thread.
- `void start()` : lance l'exécution d'un thread
- `void suspend()` : suspend le thread jusqu'au moment où il sera relancé par la méthode `resume()`.
- `void yield()` : indique à l'interpréteur que le thread peut être suspendu pour permettre à d'autres threads de s'exécuter.
- `void sleep(long)` : mettre le thread en attente durant le temps exprimé en millisecondes fourni en paramètre. Cette méthode peut lever une exception de type InterruptedException si le thread est réactivé avant la fin du temps.
- `void join()` : opération bloquante - attend la fin du thread pour passer à l'instruction suivante

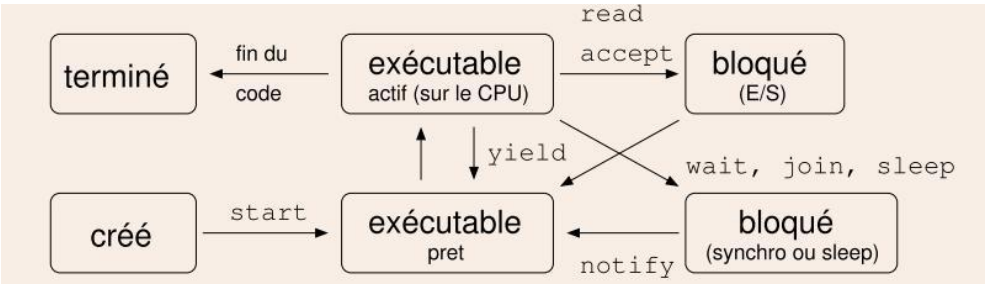

Figure 4: Le cycle de vie d'un threas

3.3 Cycle de vie d'un thread

Le cycle de vie d'un thread Java est similaire au cycle de vie standard d'un processus (voir la figure 4). Le comportement de la méthode `start()` de la classe Thread dépend de la façon dont l'objet est instancié

- si l'objet qui reçoit le message `start()` est instancié avec un constructeur qui prend en paramètre un objet Runnable, c'est la méthode `run()` de cet objet qui est appelée.
- si l'objet qui reçoit le message `start()` est instancié avec un constructeur qui ne prend pas en paramètre une référence sur un objet Runnable, c'est la méthode `run()` de l'objet qui reçoit le message `start()` qui est appelée.

Un thread en cours de traitement s'exécute jusqu'à ce qu'il soit : achevé, ou stoppé par un appel à la méthode `stop()`, ou en sortant de la méthode `run`, ou interrompu pour passer la main par `yield()`, ou mis en sommeil par `sleep()`, ou désactivé temporairement par suspend() ou wait().

3.4 Gestion de la propriété d'un thread

La priorité du nouveau thread est égale à celle du thread dans lequel il est créé, un thread ayant une priorité plus haute recevra plus fréquemment le processeur qu'un autre thread. La priorité d'un thread varie de 1 à 10. La classe Thread définit trois constantes :

- `MIN_PRIORITY` : priorité inférieure (0)
- `NORM_PRIORITY` : priorité standard (5 : la valeur par défaut)
- `MAX_PRIORITY` : priorité supérieure (10)

Pour déterminer la priorité d'un thread on utilise la méthode `getPriority()` pour la modifier on utilise `setPriority(int)`.

3.5 Gestion d'un groupe de thread

La classe ThreadGroup représente un ensemble de threads, il est ainsi possible de regrouper des threads selon différents critères. Il suffit de créer un objet de la classe ThreadGroup et de lui affecter les différents threads. Un objet ThreadGroup peut contenir des threads mais aussi d'autres objets de type ThreadGroup.
La notion de groupe permet de limiter l'accès aux autres threads : chaque thread ne peut manipuler que les threads de son groupe d'appartenance ou des groupes subordonnés. La classe ThreadGroup possède deux constructeurs :

- `ThreadGroup(String nom)` : création d'un groupe avec attribution d'un nom
- `ThreadGroup(ThreadGoup groupe_parent, String nom)` :création d'un groupe à l'intérieur du groupe spécifié avec l'attribution d'un nom

Pour ajouter un thread à un groupe, il suffit de préciser le groupe en paramètre du constructeur du thread.

4 Synchronisation des threads

Les threads s'exécutant en parallèle sur des données qui peuvent être communes, il faut gérer des conflits d'accès et des problèmes de synchronisation entre threads. Plusieurs threads peuvent accéder à un objet concurrent (problème d'accès concurrent) =¿ Introduction de la notion de section critique. Ainsi, le mot clé `synchronized` dans l'en-tête des méthodes sert à construire ce que l'on appelle des "moniteurs", c'est-à-dire des structures de données qui sont protégées de telle manière que seules des procédures qui travaillent en exclusion mutuelles puissent accéder aux objets.

4.1 Synchronisation des threads : exemple

Plusieurs processus veulent accéder à un compte en banque, pour cela, il faut utiliser la commande `synchronized` qui fait en sorte que les méthodes soient exécutées en exclusion mutuelle.

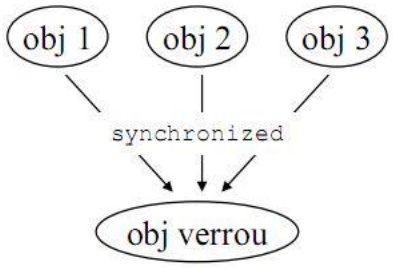

Figure 5: Accès concurrent

```
public class Compte
int solde = 0;
public synchronized void deposer(int s) {
int so = solde + s;
solde = so;
}
public synchronized void retirer(int s) {
int so = solde - s;
solde = so;
}
}
```

4.2 Concurrence d'accès

4.2.1 Exercice : crédit sur un compte commun

Soit une classe gérant les sommes créditées sur un compte commun. Écrire le programme formé d'un programme principal et de 2 acteurs apportant respectivement 35000 DA et 25000 DA. À la suite de tout ajout, le solde du compte est affiché par les acteurs. À la fin de l'exécution des acteurs, le solde du compte commun est affiché par le programme principal. Une solution est présentée dans le code ci-dessous.

```
class PartageMemoire extends Thread {
private static int cpteCommun = 0;
private int salaire;
PartageMemoire ( int monSalaire ) {
salaire = monSalaire;
}
public void run() {
cpteCommun = cpteCommun + salaire;
System.out.println( "Cpte Commun "+ cpteCommun + " DA" );}
```

```
public static void main(String args[]) {
Thread t1 = new PartageMemoire ( 35000 );
Thread t2 = new PartageMemoire ( 25000 );
t1.start();
t2.start();
try {
t1.join();
t2.join();}
catch (InterruptedException e){}
System.out.println( "Cpte Commun final "+ cpteCommun + " DA" );
}}
```

4.3 Notion de verrous

Si d'autres threads cherchent à verrouiller le même objet, ils seront endormis jusqu'à ce que l'objet soit déverrouillé =¿ mettre en place la notion de section critique. Pour verrouiller un objet par un thread, il faut utiliser le mot clé `synchronized`. Il existe deux façons de définir une section critique. Soit on synchronise un ensemble d'instructions sur un objet :

```
synchronized(object) {
    // Instructions de manipulation d'une ressource partagée.
}
```

Soit on synchronise directement l'exécution d'une méthode pour une classe donnée.

```
public synchronized void meth(int param) {
    // Le code de la méthode synchronizée.
}
```

4.3.1 L'exclusion mutuelle

Chaque objet Java possède un verrou dont la clé est gérée par la JVM, lorsqu'un thread souhaite accéder à une méthode synchronisée d'un objet, il demande la clé de cet objet à la JVM, entre dans la méthode, puis ferme le verrou à clé. De cette façon, aucun autre thread ne peut accéder aux méthodes synchronisées de cet objet. Lorsque le thread sort de la méthode synchronisée, il ouvre de nouveau le verrou et rend la clé à la JVM. De cette façon, un autre thread peut alors accéder aux méthodes synchronisées de l'objet.

4.4 Synchronisation temporelle : wait() et notify()

Les méthodes `wait()`, `notify()` et `notifyAll()` permettent de synchroniser différents threads. Ces méthodes sont définies dans la classe Object (car elles manipulent le verrou associé à un objet), mais ne doivent s'utiliser que dans des méthodes `synchronized`. Dans le cas de la méthode `wait()`, le thread qui appelle cette méthode est bloqué jusqu'à ce qu'un autre thread appelle `notify()` ou `notifyAll()`. la méthode `wait()` libère le verrou, ce qui permet à d'autres threads d'exécuter des méthodes synchonisées du même objet. les méthodes `notify()` et `notifyAll()` permettent de débloquer une tâche bloqué par `wait()`.
Par exemple, si une tâche T1 appelle wait dans une méthode de l'objet O, seule une autre méthode du même objet pourra la débloquer; cette méthode devra être synchronisée et exécutée par une autre tâche T2 :

- `void unObjet.wait()` : le thread appelant cette méthode rend la clé du verrou et attend l'arrivée d'un signal sur l'objet unObjet. Cette méthode doit être utilisée à l'intérieur d'une méthode ou d'un bloc `synchronized`.
- `void unObjet.notify()` : indique à un thread en attente d'un signal sur unObjet de l'arrivée de celui-ci. A utiliser dans une méthode ou un bloc `synchronized`.
- `void unObjet.notifyAll()` : indique à tous les threads en attente d'un signal sur unObjet de l'arrivée de celui-ci. A utiliser dans une méthode ou un bloc `synchronized`.

- Si plusieurs threads exécutent `unObjet.wait()`, chaque `unObjet.notify()` débloquera un thread bloqué, dans un ordre indéterminé.

5 Références

- Java Threads, 3e édition,Scott Oaks, Henry Wong, O'Reilly, 2004
- Java 7 - Les bases du langage et de la programmation objet, Thierry Groussard, éditeur Eni, 2013.
- Beginning Java EE 7, Antonio Goncalves, Apress, 2013
- La documentation officielle et le tutoriel chez Sun : `http://download.oracle.com/javase/6/docs/index.h`
- Les tutoriels chez développez.com : `http://java.developpez.com/cours/`
- la paquetage Thread de java : `http://doc.java.sun.com/DocWeb/api/java.lang.Thread`
- Apprendre Java - Cours et exercices, Irène Charon
- Apprentissage du langage java, Serge Tahé
- Penser en Java 2nde édition, Bruce Eckel traduit en francais (Thinking in Java) : `http://bruce-eckel.devel`