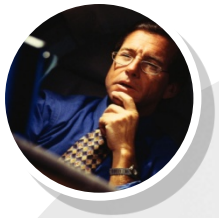


POO – JAVA
Les Threads

ENSA
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES
KHOURIBGA

Université Sultan Moulay Slimane

ENSA KHOURIBGA ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES

Noreddine Gherabi

PROCESSUS ET THREAD

Deux unités d'exécution:

- **Processus :**
  - **Application ou programme**
  - **Environnement d'exécution auto-contenu (mémoire, etc.)**
- **Thread(s) :**
  - **Processus léger**
  - **Rattachée(s) à un processus**
  - **Partage les mêmes ressources (mémoire, accès fichiers, etc.)**
  - **Différents branches d'exécution**

PROCESSUS ET THREAD

→La communication entre threads est plus rapide que celle entre processus, parce que les threads partagent un même espace de mémoire (de travail) entre eux, alors que les processus ont chacun sont espace mémoire personnel.

→Un processus peut lancer plusieurs threads qui se partagent le même espace mémoire et peuvent donc se partager des variables

Exemples d'utilisation:

- ➔ interface graphique (IHM)
- ➔ le serveur réseau
- ➔ produit matricielle

MULTITHREADING

- En Java le multithreading signifie qu'il faut
  1. Créer un thread,
  2. Ecrire la tâche exécutée par le thread,
  3. Lier la tâche au thread.
- Pour réaliser cela, il faut étudier et comprendre la classe Thread.

Thread
void join() void start() static void sleep()

UTILITÉ DU MULTITÂCHE

- Répartition des tâches sur plusieurs processeurs
  - Programmation parallèle…
- Simplification de code (par isolation)
- Interface graphique réactive (sans blocage)

Classe Thread / Interface Runnable

→Il existe deux moyens d'écrire des threads dans un programme. Les deux procédés passent par l'écriture d'une méthode run() décrivant les instructions que doit exécuter un thread.

→La méthode run() est soit :

- ➔déclarée dans une classe dérivée de la classe Thread
- ➔déclarée dans n'importe quelle classe qui doit alors implémenter l'interface.

→ Une thread est lancée par appel d'une méthode start ()

Utilisation de la classe Thread

- Écriture (surchage) de l'invocation de la méthode `run()` :
  - méthode appelée lors de l'invocation de la méthode `start()` sur un objet de la classe `Thread`
  - exemple :

```
class MonThread extends Thread {
    ...
    public void run() {
        ... code de l'activité ...
    }
}
class Principale {
    public static void main(String[] argv) {
        MonThread p = new MonThread(...);
        p.start();
        ...
    }
}
```

Les Constructeurs de Thread

- 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.

Exemple

Exemple : utilisation de la classe Thread

```
public class MonThread extends Thread {
  public MonThread(String name) {
    super(name);
  }

  @Override
  public void run() {
    for (int i = 0; i < 5; i++) {
      System.out.print(this.getName() + " ");
    }
  }
}
```

Exemple

Exemple : utilisation de la classe Thread

Méthode Main :

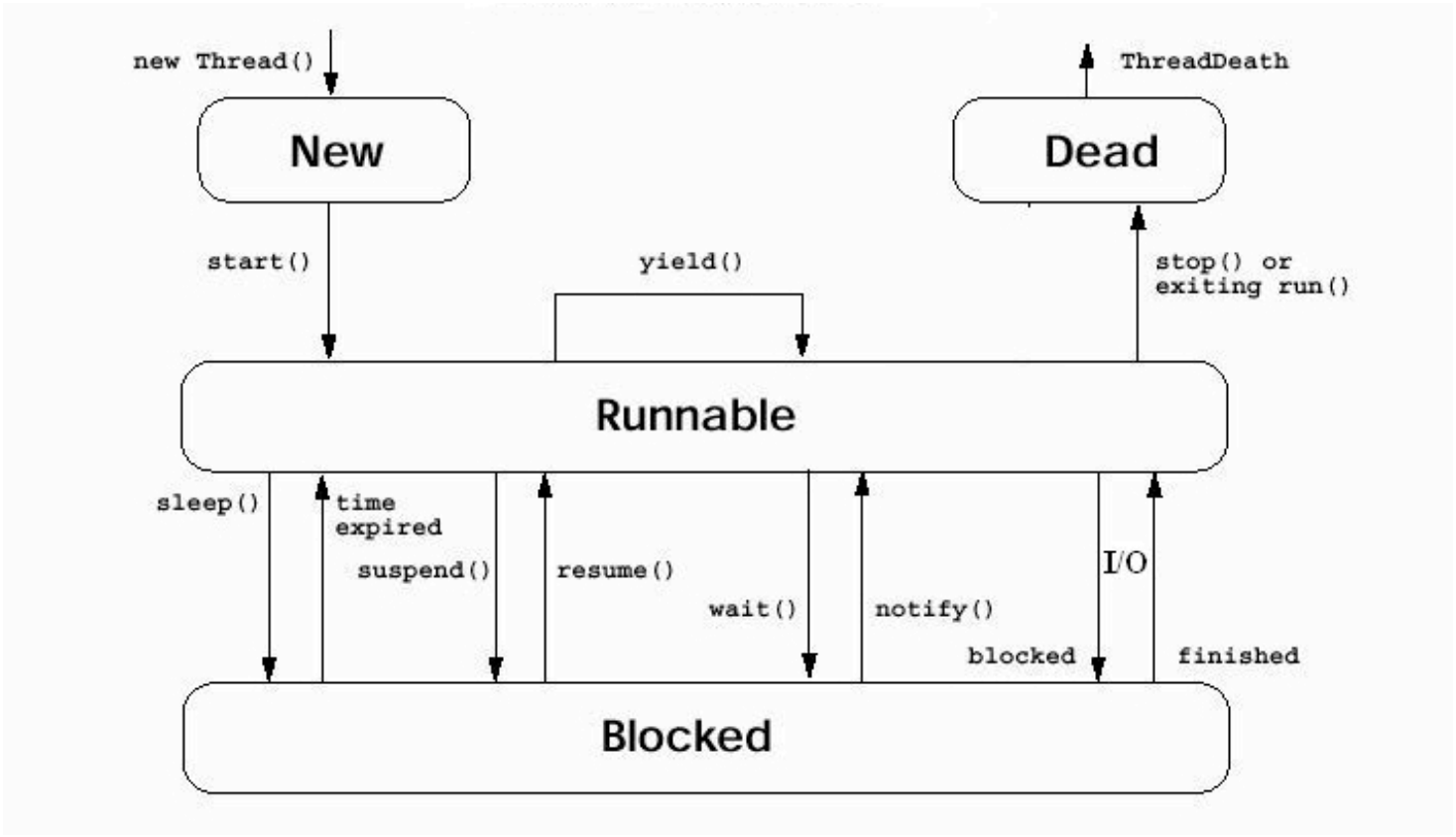
```
public class Test {
  public static void main(String[] args) {
    MonThread A = new MonThread("A");
    MonThread B = new MonThread("B");
    MonThread C = new MonThread("C");
    A.start();
    B.start();
    C.start();
    System.out.println(Thread.currentThread().
       getName() + " : j'ai fini");
  }
}
```

Interface RUNNABLE

- Implémentation de l'interface `Runnable`
  - écriture de la méthode `run()`
  - exemple :

```
class MonThread implements Runnable {
    ...
    public void run() {
        ... code de l'activité ...
    }
}
class Principale {
    public static void main(String[] argv) {
        MonThread p = new MonThread(...);
        Thread t = new Thread(p);
        t.start();
        ...
}
```

CYCLE DE VIE D'UN THREAD



Les Méthodes des Threads

Start

start: exécute donc la méthode run associé au thread. Quand un thread est lancé, vous ne pouvez pas le relancer une 2e fois.

```
class SimpleThread extends Thread {
public void run() {
System.out.println( « je démarre" );
}}
public class Restart {
public static void main( String args[] ) throws Exception {
SimpleThread T1= new SimpleThread();
T1.start();
System.out.println( "en cours d'exécusion");
T1.start();
System.out.println( "Fin");
}}
```

SLEEP

sleep: permet donc d'en dormir un thread un certain temps.

Pour utiliser sleep, il faudra capturer l'exception InterruptedException.

via une implantation de la classe Runnable

- **Thread.sleep((int)(Math.random() * 1000));**

sinon

- **sleep((int)(Math.random() * 1000));**

vu que la classe dérive directement de Thread.

YIELD

yield: met le thread courant dans l'état prêt pour permettre à un autre thread de même priorité ou priorité supérieure de s'exécuter.

```
class YieldThread extends Thread {
public void run() {
for ( int count = 0; count < 4; count++) {
System.out.println( count + "de : " + getName() );
yield();
} } }
class TestYield {
public static void main( String[] args ) {
YieldThread first = new YieldThread();
YieldThread second = new YieldThread();
first.start(); second.start();
System.out.println( "Fin" );
} }
```

Join

join: est utilisé pour mentionner au thread d'attendre la fin d'un autre thread.

```
Coureur j = new Coureur ("Alami");
Coureur p = new Coureur ("Anouari");
// On lance les deux coureurs.
j.start();
p.start();
try { j.join(); p.join(); }
catch (InterruptedException e) {  };
Coureur k = new Coureur ("Hamdi ");
k.start() ;
```

Ici Hamdi va se lancer après que Alami & Anouari terminent leur course.

interrupt

interrompre un thread: 3 méthodes de la classe Thread interviennent à ce niveau:

void interrupt(); pour interrompre un thread

boolean isInterrupted(); true s'il a été interrompu sinon false.

static boolean interrupted();true s'il a été interrompu sinon false

```
class NiceThread extends Thread {
public void run() {
while ( !isInterrupted() ) {System.out.println( "From: " + getName() ); }
System.out.println(  ”Le thread est encore de traitement” );
} }
public class InterruptTest {
public static void main(String args[]) throws InterruptedException{
NiceThread T1= new NiceThread();
      T1.start();
Thread.currentThread().sleep( 200 ); // attend le démarrage d’un autre
thread
T1.interrupt();}}
```

Autres Méthodes

- **void destroy(); // détruit le Thread courant sans faire le ménage.**
- **String getName(); // retourne le nom du Thread.**
- **int getPriority(); // retourne la priorité du Thread.**
- **void setPriority(int newPriority); // changer la priorité du Thread. (*MAX_PRIORITY(10), MIN_PRIORITY(1), 5*)**
- **static boolean interrupted(); // teste si le Thread courant a été interrompu.**
- **void resume(); // redémarrer le Thread.**
- **void stop(); // Arrêter le Thread**
- **Void suspend()(); // Arrêt momentané duThread**
- **String toString(); // nom du thread et sa priorité**

La synchronisation

Objectif : faire en sorte qu'un seul thread exécute une partie du code

- **Blocage possible !!**
- **Un thread n'est pas bloqué par lui-même**
- **Ralentissement du code : attendre son tour**

® Synchroniser juste ce qui est nécessaire

La synchronisation

➢ **Nous avons mentionné que les Threads partagent les mêmes ressources (mémoire, variables du processus etc.). Dans certaines situations, il est nécessaire de synchroniser les threads pour obtenir un résultat cohérent.**

→Si par exemple, un thread threadCalcul a pour charge de modifier une champ variable qu'un autre thread threadAffichage a besoin de la valeur modifier pour l'afficher, ce peut que threadCalcul n'a pas terminé la mise à jour de cette valeur, dans ce cas threadAffichage ne peut pas afficher cette valeur. Ainsi, vous aurez besoin de **synchroniser** les threads threadCalcul et threadAffichage .

Exemple : La synchronisation

```
public class Synchr1
{ public static void main (String args[])
  { Nombres nomb = new Nombres() ;
    Thread calc = new ThrCalc (nomb) ;
    Thread aff  = new ThrAff (nomb) ;
    calc.start() ; aff.start() ;
  }
}
```

```
public class Nombres
{ public synchronized void calcul()
  { n++ ;  facDEn = facDEn*n ;   }
  public synchronized void affiche ()
  { System.out.println ((int)n + "! = " + facDEn) ;   }
  private double n=1, facDEn=1 ;
}
```

Exemple : La synchronisation

```
public class ThrAff extends Thread
{ public ThrAff (Nombres nomb)
  { this.nomb = nomb ; }
   public void run ()
  {int i=0;
  try
    { while (i<14)
      { nomb.affiche() ;
        sleep (75) ; i++;  }
    }
    catch (InterruptedException e) {}
  }
  private Nombres nomb ;
}
```

```
public class ThrCalc extends Thread
{ public ThrCalc (Nombres nomb)
  { this.nomb = nomb ;    }
    public void run ()
  { int i=0;
    try
    { while (i<14)
      { nomb.calcul () ;
        sleep (50) ; i++;        }
    }
    catch (InterruptedException e) {}
  }
  private Nombres nomb ;
}
```

Méthodes de synchronisation

Méthode wait (sur un objet)

⊃objet.wait( );

- **Appel dans une section synchronisée**
- **Possibilité d'attendre que quelque chose réveille le thread**
- **Le thread ne fait plus rien en attendant**
  - **Et il passe aussitôt la main**
  - **Réveil par la méthode notify**

⊃objet.notify( );

- **Appel dans une section synchronisée**
- **Ne débloque qu'un thread, on ne sait pas lequel**

®Réveil par notifyAll( );

- **Possibilité d'utiliser un délai max du wait(délai)**

Dans ce cas, le thread doit gérer lui-même le fait de connaître la cause de son déblocage (notify ou temps écoulé)

Exercice

EN utilisant la synchronisation, créer un code Java qui permet de gérer les opérations de dépôt et de retrait des clients dans un compte
Le programme doit avoir :

- **Une classe Compte contient deux méthodes (Retrait /dépôt)**
- **Un Thread permet de gérer les dépôts (exp : 6 dépôts de 1000 dh)**
- **Un Thread qui gère les retraits (exp : 4 retraits de 1500 dh)**
- **Tester les Threads.**