Programmation parallèle en Java

Imene LAHYANI

Rappel

- **Prog Sequentielle**

Tache 1;

Tache 2 ;

Tache 3 ;

- **Prog parallèle**

Processus 1 // processus 2

- **Prog Sequentielle**

X=x+1;

Y=y+6 ;

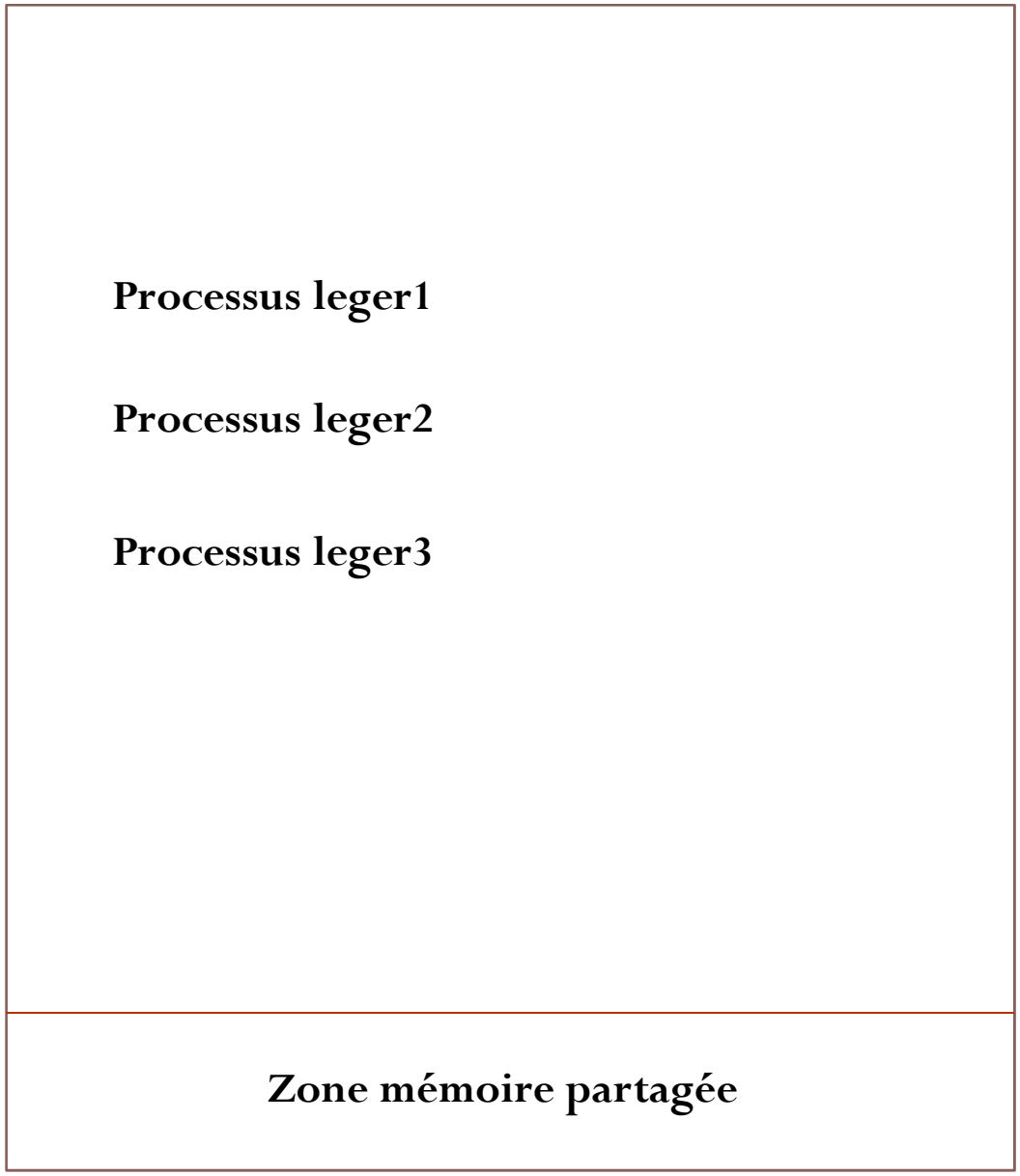
X=x-9;

- **Prog Parallèle**

X=x+1; // Y=y+6

X=x-9;

Processus lourd

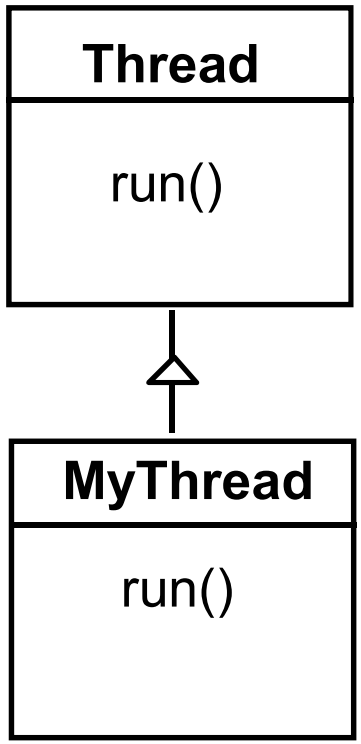


Concepts de base

- Un processus lourd possède une ou plusieurs unités d'exécution appelés Threads
- Les threads s'executent en parallèle
- Un thread est un processus léger
- Un thread est un flux séquentiel de contrôle à l'intérieur d'un processus

Création et lancement (1/2)

- **Java** offre la possibilité de faire des traitement parallèle grâce à la classe **Thread**.



La classe **Thread** exécute les instructions de sa méthode **run()**

```
class MyThread extends Thread {
      public void run() {
            //......
      }
}
```

Création et lancement (2/2)

1 iere méthode de création de thread

1/ Créer une classe héritant de **Thread**

2/ Redéfinir la méthode **Thread.run()**

3/ Créer une instance de la classe

4/ Appeler la méthode **start ()** sur cette instance

N'oubliez pas d'importer **Java.lang.thread**

Exemple

```
public class mythread  extendsThread
{mythread()
{ }
public void run()
{ for(int i=0; i<10;i++)
System.out.println(i);}} / / fin thread

Class test {
public static void main (String args[])
{mythreadT1= new mythread ();
mythreadT2= new mythread ();
T1.start();
T2.start();
}
}
```

```
public class Mythread  extends Thread
{mythread(String nom)
{super(nom);}
public void run()
{ for(int i=0; i<10;i++)
System.out.println(« i »+i+getName());}} // fin thread

Class test {
public static void main (String args[])
{mythread T1= new mythread ("Test1");
mythread T2= new mythread ("Test2");
T1.start();
T2.start();
}
}
```

```
public class Test1  extends Thread
{Test1(String nom)
{super(nom);}
public void run()
{ for(int i=0; i<10;i++)
System.out.println(getName());}} // fin thread


Class test {
public static void main (String args[])
{Test1 T1= new Test1 ("Test1");
Test1 T2= new Test1 ("Test2");
T1.start();
T2.start();
System.out.println(« je suis dans main »);
}
}
```

```
public class Test1  extends Thread
{Test1(String nom)
{super(nom);}
public void run()
{ for(int i=0; i<10;i++)
System.out.println(getName());}} // fin thread


Class test {
public static void main (String args[])
{ Test1 T1= new Test1 ("Test1");
Test1 T2= new Test1 ("Test2");
System.out.println(« je suis au debut de  main  »);
T1.start();
T2.start();
System.out.println(« je suis dans main »);
}
}
```

```
public class Test1  extends Thread
{Test1(String nom)
{super(nom);}
public void run()
{ for(int i=0; i<10;i++)
System.out.println(getName());}} / / fin thread


Class test {
public static void main (String args[])
{ Test1 T1= new Test1 ("Test1");
Test1 T2= new Test1 ("Test2");
System.out.println(« je suis au debut de  main  »);
T1.start();
T2.start();
System.out.println(« je suis à la fin de  main »);
}
}
```

```
Try{
T1.join()
T2.join()
System.out.println(« je suis à la fin de
main »);
}
Catch(Exception e)
{}
```

Cycle de vie d'un thread

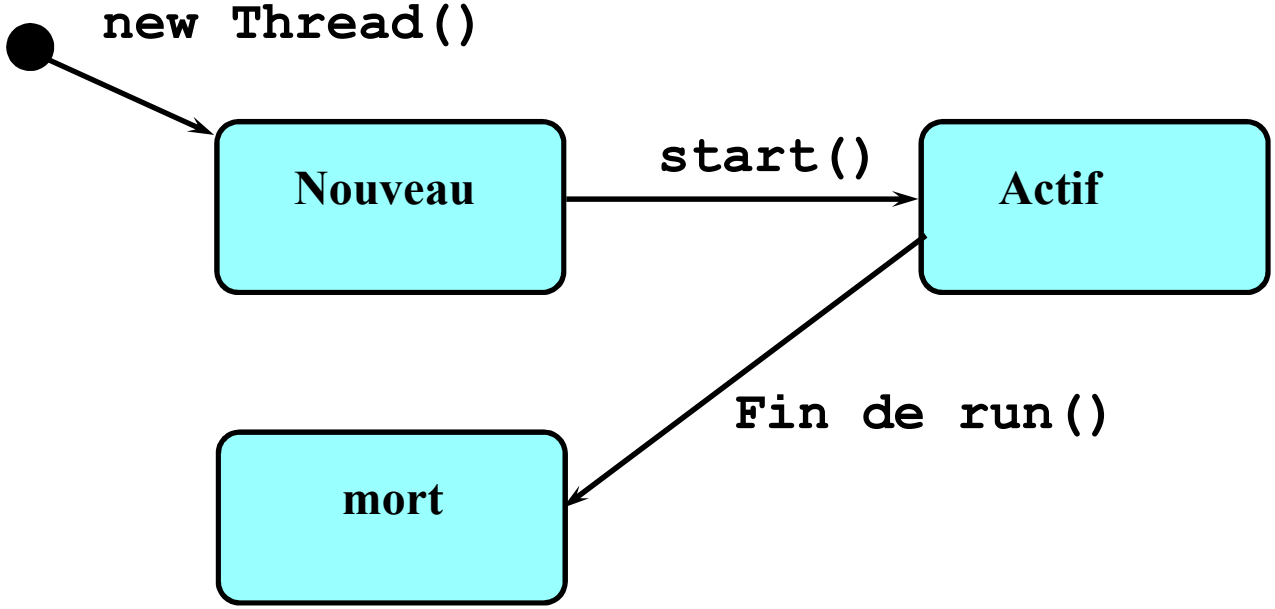
- Le thread peut avoir quatre états différents:

1/ État nouveau

- C'est l'état initial après l'instanciation du thread. À ce stade, le thread est opérationnel, mais celui-ci n'est pas encore actif.
- Un thread prend cet état après son instanciation.

2/ Etat exécutable

- Un thread est dans un état exécutable à partir du moment où il a été lancé par la méthode **start()** et le reste tant qu'il n'est pas sorti de la méthode run().


new Thread()
Nouveau
start()
Actif
Fin de run()
mort

Cycle de vie d'un thread

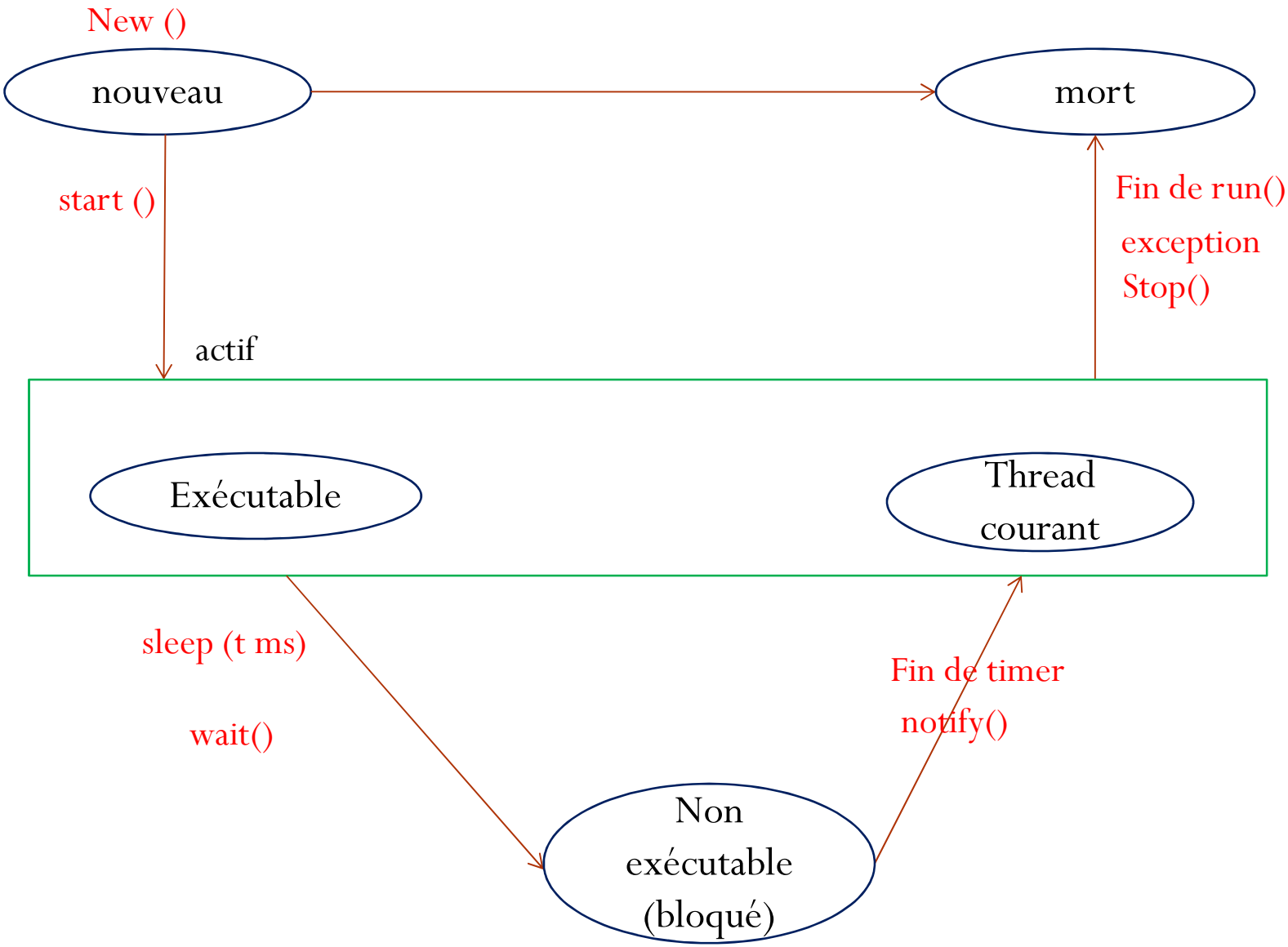
4/ État mort

- Un thread dans un état mort est un thread qui est sorti de sa méthode run() soit de manière naturelle, soit de manière subite (Exception non interceptée).

Cycle de vie d'un thread

3/ État en attente

- Un thread en attente est un thread qui n'exécute aucun traitement et ne consomme aucune ressource CPU.
- Il existe plusieurs manières de mettre un thread en attente.
  - appeler la méthode **thread.sleep** (temps en millisecondes) ;
  - appeler la méthode **wait()** ;
  - accéder à une ressource bloquante (flux, accès en base de données, etc.) ;
  - accéder à une instance sur laquelle un verrou a été posé ;
  - appeler la méthode **suspend()** du thread.


New ()
nouveau
mort
start ()
Fin de run()
exception
Stop()
actif
Exécutable
Thread
courant
sleep (t ms)
wait()
Fin de timer
notify()
Non
exécutable
(bloqué)

Quelques méthodes de thread en Java

- **static int activeCount()** : renvoie le nombre de threads actuellement exécutés
- **static Thread currentThread() :** renvoie le Thread correspondant au thread en train d'être exécuté. (utile notamment avec Runnable)
- **static void sleep(long) :** le thread va etre endormi pour le temps specifié en valeur de l'argument
- **static void join()** : attends la fin de l'execution du thread
- **void setName(String name) :** change le nom du Thread
- **String getName() :** retourne le nom du Thread
- **void setPriority (int)** : fixer la priorité du thread
- **int getPriority ()** : renvoie la priorité du thread
- **static void yield ()** : provoque une "pause" du thread en cours et un réordonnancèrent
- **Boolean isAlive()** : renvoie un boolean qui indique si le thread est actif ou pas

L'interface Runnable (2 ieme methode de creation de thread)

- La méthode de création de thread en héritant de la superclasse thread n'est pas toujours acceptable.
- dans certains cas, on veut étendre une classe existante et en faire une sous-classe de Thread également, pour pouvoir exécuter ses instances en parallèle.
- Pas d'héritage multiple en JAVA !!!
- Solution: utiliser l'**interface Runnable du package Java.lang**

L'interface Runnable

- 2 méthode de création de Thread

1/ Créer une classe implémentant l'interface Runnable

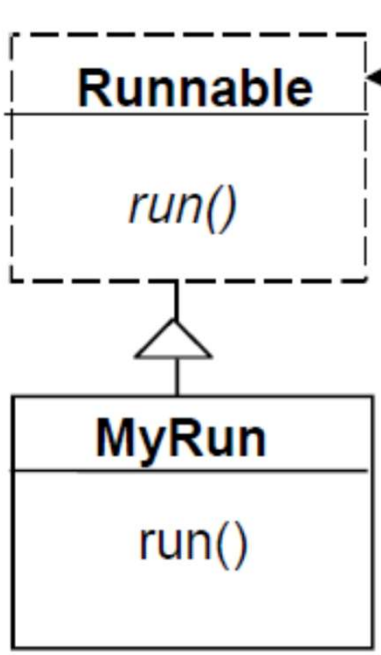
2/ Implémenter la méthode Thread.run()

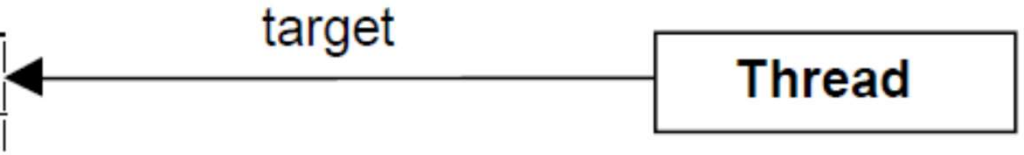
3/ Créer une instance de la classe

4/ Créer un objet Thread à partir de cette instance

5/ Appeler la méthode start() sur cet objet Thread

L'interface Runnable






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

class MyRun implements Runnable{
    public void run() {
        //.....
    }
}
```

Creation et execution du thread b:

```
Thread b = new Thread(new MyRun());
b.start();
```

L'interface Runnable

```
class A extends Mere implements Runnable
{ ..}
public void run()
{System.out.println('....');
}
} // fin MonRunnable
class Test
{
public static void main(String args[])
{
MonRunnable R = new MonRunnable ();
Thread T = new Thread (R);
T.start();
}}
```

Exercice : Tri par fusion

- Principe:
  - On découpe les données à trier en deux parties plus ou moins égales
  - On trie les 2 sous-parties ainsi déterminées
  - On fusionne les deux sous-parties pour retrouver les données de départ
- Solution séquentielle ?
- Solution parallèle ?