INFO-H-200
Programmation orientée objet

Séance d'exercices 7
Threads

Université libre de Bruxelles
École polytechnique de Bruxelles

Professeur : Hugues Bersini

2017-2018

Thread

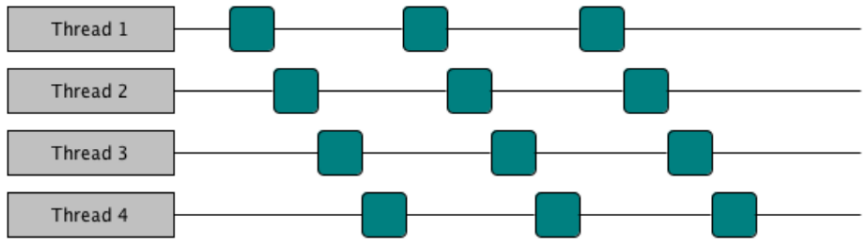
Un Thread est un processus

Le but de l'utilisation des threads est d'exécuter des opérations en parallèle.

Exemple : Un logiciel de traitement de texte:

- Un thread qui gère le clavier
- Un thread qui gère la correction orthographique

Pour donner une illusion de parallélisme sur un processeur, l'OS (ou la JVM) exécute les processus et threads de façon concurrent et synchronisée.



Etats : "Nouveau", "Exécutable", "En Attente", "Mort".

Comment faire ?

Deux manières de faire :

- Hériter de la classe Thread (peu pratique)
- Implémenter l'interface Runnable (préféré)

Une classe *thread* doit contenir une méthode *void run()* qui défini le comportement et le cycle de vie de ce thread.

Une fois la méthode *run()* terminée, le thread s'arrête et est détruit.

Pour démarrer le thread, il faut appeler sa méthode start().

Premier Thread via interface Runnable

```
public class MyTimer implements Runnable{
    String string;
    int waitTime;

    public MyTimer(String string, int waitTime){
        this.string = string;
        this.waitTime = waitTime;
    }

    @Override
    public void run(){
        try{
            while(true){
                System.out.print(string);
                Thread.sleep(waitTime);    //se mets en attente
            }
        }catch(Exception e){};
    }

    public static void main(String[] args){
        Thread t1 = new Thread(new MyTimer("A",100));
        Thread t2 = new Thread(new MyTimer("B",100));
        Thread t3 = new Thread(new MyTimer("C",200));
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Threads sans attendre

```
public class MySync implements Runnable{
    public static int counter;


   @Override
   public void run(){
       int temp = counter;
       temp = temp + 1;
       counter = temp;
   }

   public static void main(String[] args) throws InterruptedException{
      Thread t1 = new Thread(new MySync());
      Thread t2 = new Thread(new MySync());
      t1.start();    // t1 est prêt à être lancé
      t2.start();    // t2 est prêt à être lancé
      System.out.println("Counter = " + String.valueOf(counter));
   }
}
```

- Que pourrait-il donc se passer de mal ?
- C'est l'OS qui gère l'ordonnancement des threads!
- Aucune garantie que t1 et t2 se lanceront avant le print

Tentative de solution, threads avec attente

```
public class MySync implements Runnable{
    public static int counter;


   @Override
   public void run(){
       int temp = counter;
       temp = temp + 1;
       counter = temp;
   }

   public static void main(String[] args) throws InterruptedException{
      Thread t1 = new Thread(new MySync());
      Thread t2 = new Thread(new MySync());
      t1.start();
      t2.start();
      t1.join();        // le thread principal attend ici que t1 finisse
      t2.join();        // le thread principal attend ici que t2 finisse
      System.out.println("Counter = " + String.valueOf(counter));
   }
}
```

- Que pourrait-il donc encore se passer de mal ?
- Le thread principal est en attente, mais quid de t1 et t2 ?
- Rappel: l'OS peut interrompre un thread à tout moment!

Réservation de ressources

Comme ce n'est pas le programmeur qui gère l'ordonnancement des threads, il peut arriver que deux threads accèdent à la même ressource en même temps.

Cela peut être dangereux. Pour cela, un thread qui utilise une ressource partagée doit la réserver pour en avoir un accès exclusif.

Pour avoir un accès exclusif à un objet, ou attendre que celui-ci soit disponible :

```
synchronized(unObjet){
   // Section critique
}
```

Solution : synchronisation

```
public class MySync implements Runnable{
    public static int counter;
    public static my_key = new Object();


   @Override
   public void run(){
       synchronized(my_key){
           int temp = counter;
           temp = temp + 1;
           counter = temp;
        }
   }

   public static void main(String[] args) throws InterruptedException{
      Thread t1 = new Thread(new MySync());
      Thread t2 = new Thread(new MySync());
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println("Counter = " + String.valueOf(counter));
   }
}
```

- Pourquoi *my_key* est elle statique ?

Deadlocks

Situation d'interblocage

Exemple :

- soit deux threads t1 et t2 et deux ressources r1 et r2
- t1 a la main et bloque r1
- t2 prend la main et bloque r2
- t1 prend la main et veut bloquer r2. Il doit attendre que r2 soit libre et donc que t2 la libère
- t2 prend la main et veut bloquer r1. Il doit attendre que r1 soit libre et donc que t1 la libère.
- Le système est bloqué !

Annexe: quelques bonnes et mauvaises synchronisations

Mauvaise synchronisation

```
public class MySync implements Runnable{
    public static int counter;
    public my_key = new Object();


   @Override
   public void run(){
       synchronized(my_key){
           int temp = counter;
           temp = temp + 1;
           counter = temp;
        }
   }

   public static void main(String[] args) throws InterruptedException{
      Thread t1 = new Thread(new MySync());
      Thread t2 = new Thread(new MySync());
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println("Counter = " + String.valueOf(counter));
   }
}
```

- chaque thread a sa propre clé!

Solution

```
public class MySync implements Runnable{
    public static int counter;
    public static my_key = new Object();


   @Override
   public void run(){
       synchronized(my_key){
           int temp = counter;
           temp = temp + 1;
           counter = temp;
        }
   }

   public static void main(String[] args) throws InterruptedException{
      Thread t1 = new Thread(new MySync());
      Thread t2 = new Thread(new MySync());
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println("Counter = " + String.valueOf(counter));
   }
}
```

Mauvaise synchronisation de fonctions

```
public class MySync implements Runnable{
    public static int counter;

    private synchronized void increment(){
       int temp = counter;
       temp = temp + 1;
       counter = temp;


   @Override
   public void run(){
       increment();
   }

   public static void main(String[] args) throws InterruptedException{
      Thread t1 = new Thread(new MySync());
      Thread t2 = new Thread(new MySync());
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println("Counter = " + String.valueOf(counter));
   }
}
```

- chaque thread a sa propre fonction increment!

Solution

```
public class MySync implements Runnable{
    public static int counter;

    private static synchronized void increment(){
       int temp = counter;
       temp = temp + 1;
       counter = temp;


   @Override
   public void run(){
       increment();
   }

   public static void main(String[] args) throws InterruptedException{
      Thread t1 = new Thread(new MySync());
      Thread t2 = new Thread(new MySync());
      t1.start();
      t2.start();
      t1.join();
      t2.join();
      System.out.println("Counter = " + String.valueOf(counter));
   }
}
```