INFO0604
PROGRAMMATION MULTI-THREADÉE

COURS 5

PROGRAMMATION MULTI-THREADÉE EN JAVA

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Pierre Delisle
Département de Mathématiques, Mécanique et Informatique
Janvier 2022

Plan de la séance

- Développement de programmes multi-thread en Java
  - Principe de fonctionnement
  - Exemples
- Synchronisations
  - Méthodes et instructions synchronisées
  - Verrous (mutex)
  - Blocs gardés (variables de condition)
- Interruption

Application multi-threadée Java

- Chaque thread est associé à une instance de la classe *Thread*
- 2 façons de procéder
  - Instanciation d'une sous-classe de la classe *Thread*
  - Implémentation de l'interface *Runnable*

- java.lang.Thread
  - Thread fondamental de l'API Java
- 2 constructeurs de base (il y en a d'autres)
  - Thread()
  - Thread (Runnable objet)
- Quelques méthodes
  - run(), start()
  - getId(), getState()
  - join()

PREMIÈRE MÉTHODE

Instanciation d'une sous-classe de *Thread*

Étapes de développement

- Conception d'une classe qui hérite de la classe *Thread*
- Définition de la méthode *run* dans cette classe
- Création d'instances de cette classe
- Démarrage de l'exécution par l'appel de la méthode *start* (qui appelle implicitement la méthode *run*)

Structure d'une classe et de l'exécution

```
public class MonThread extends Thread {
    public MonThread (…) {
        ...
    }

    public void run () {
        /*Cette méthode sera exécutée
        lors de l'appel de la méthode start() /*
        ...
    }
}
```

- Exemple

```
public class programme {
    public static void main (String [ ] args) {
        MonThread unThread;
        MonThread unAutreThread;
        unThread = new MonThread(...);
        unAutreThread = new MonThread(...);
        unThread.start();
        unAutreThread.start();
    }
}
```

DEUXIÈME MÉTHODE

Implémentation de l'interface *Runnable*

L'interface *Runnable*

- Doit être implémentée par la classe que l'on veut exécuter dans un thread
- Définit une seule méthode : *run*
  - Contient le code exécuté dans le thread
  - L'objet Runnable est passé au constructeur de *Thread*
- Étapes de développement
  - Conception d'une classe (n'héritant pas de Thread) qui implémente l'interface *Runnable*
  - Définition de la méthode run dans cette classe
  - Création d'instances de la classe
  - Création d'instances de la classe Thread en passant au constructeur les instances de la classe développée
  - Démarrage de l'exécution par l'appel de la méthode *start* (qui appelle implicitement la méthode *run*)

Structure d'une classe et de l'exécution

```
public class MonThread implements Runnable {
    public MonThread (…) {
        ...
    }

    public void run () {
        /*Cette méthode sera exécutée
        lors de l'appel de la méthode start() /*
        ...
    }
}
```

- Exemple

```
public class programme {
    public static void main (String [ ] args) {
        Thread unThread;
        Thread unAutreThread;
        unThread = new Thread(new MonThread(...));
        unAutreThread = new Thread(new MonThread(...));
        unThread.start();
        unAutreThread.start();
    }
}
```

Quelle méthode utiliser ?

- Instanciation d'une sous-classe de la classe Thread
  - Plus simple à utiliser
  - Problématique lorsque la classe à exécuter est déjà une sous-classe (pas d'héritage multiple en Java)
- Implémentation de l'interface Runnable
  - Plus générale et plus flexible
  - Séparation de la tâche à exécuter et de l'exécution elle-même

SYNCHRONISATIONS

Les verrous en Java

- Les mécanismes de synchronisation sont basés sur une entité de verrou « interne » à Java
  - Verrou intrinsèque (*instrinsic lock* ou *monitor lock)*
  - Chaque objet a un verrou intrinsèque associé
  - Assure la protection et la visibilité mémoire
- 2 techniques de base de synchronisation
  - Méthodes synchronisées (synchronized methods)
  - Instructions de synchronisation (synchronized statements)

Techniques de base de synchronisation

Méthodes synchronisées

- Déclaration de méthodes *synchronized*
  - Quand un thread appelle une méthode synchronisée d'un objet, il acquiert le verrou de cet objet
  - Le verrou est libéré à la fin de la méthode
  - Si un autre thread appelle une méthode synchronisée de cet objet, il sera bloqué jusqu'à la libération du verrou
- Exemple

Instructions de synchronisation

- Permettent une concurrence (parallélisme) plus fine que les méthodes synchronisées
- Instruction *synchronized (Object lock)*
  - Définit un bloc d'instructions exécuté dans une section critique
  - Spécifie un objet qui fournira le verrou intrinsèque (mutex)
  - Si deux blocs spécifient des objets différents, ils pourront être exécutés en parallèle
- Exemple

Blocs gardés (guarded blocks)

- Débutent en vérifiant une condition
  - Le bloc sera exécuté lorsque la condition sera vraie
- À l'intérieur du bloc
  - Méthode *wait* : permet de mettre le thread en attente (nécessite l'acquisition préalable du verrou)
  - Méthode *notify* (similaire au signal) : permet de réveiller un thread en attente en l'informant qu'un événement s'est produit
  - Méthode *notifyall* (similaire au broadcast) : permet de réveiller tous les threads en attente
- Exemple

Thread.sleep()

- Suspend l'exécution du thread pour une durée spécifiée en millisecondes
  - Thread.sleep(4) → 4 millisecondes
- Permet l'exécution d'autres threads
- Lance une exception de type *InterruptedException* si un autre thread interrompt le thread durant le sleep

Interruptions

- Méthode *interrupt()*
  - monThread.interrupt();
- Indication à un thread qu'il devrait arrêter de faire ce qu'il fait
  - Le programmeur décide comment le thread répond à l'interruption (catch/throw)
  - Par exemple le terminer en retournant de la méthode run
  - Le thread doit gérer sa propre interruption
- Exemple

Pour aller plus loin

- Java offre des fonctionnalités avancées de programmation multi-thread
  - Gestion des priorités
  - Groupes de threads
  - Objets *Lock*
  - Interface *Executor*
  - Variables atomiques
  - ...
- http://java.sun.com/docs/books/tutorial/essential/concurrency/index.html

PROCHAIN COURS

PAS DE SEMAINE PROCHAINE, C'EST TERMINÉ ☹

POUR EN SAVOIR PLUS :

CHPS0701 – PROGRAMMATION PARALLÈLE (SUITE À INFO0604)
CHPS0803 – OPTIMISATION ET RECHERCHE OPÉRATIONNELLE (SUITE À INFO0501)