Introduction à la programmation en JAVA

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Table des matières

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Thread

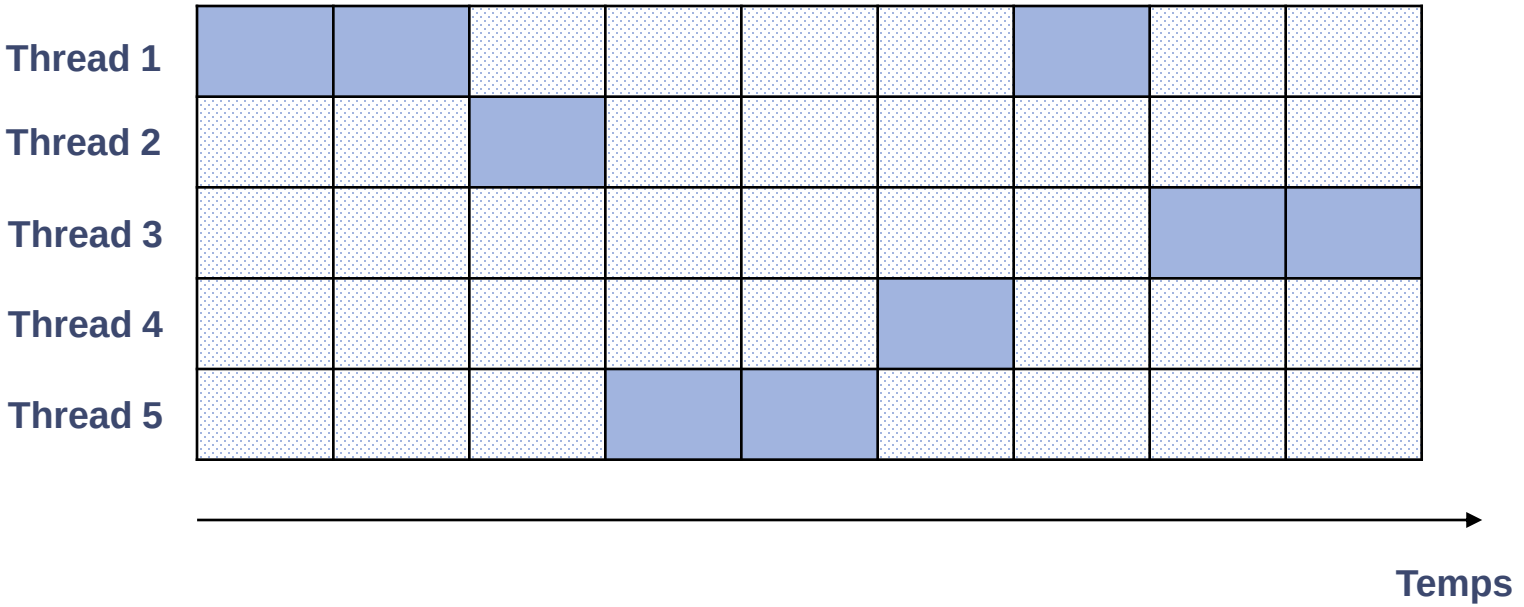
- Un **thread** est un processus léger, c'est-à-dire une tâche indépendante du programme principal

- Chaque thread possède :
  - Un état d'exécution
  - Une pile d'exécution
  - Un espace dédié à ses variables propres

- Java incorpore les primitives permettant l'exécution de threads
  - `java.lang.Thread`
  - `java.lang.ThreadGroup`
  - `java.lang.Runnable`

Multithreading

- Le **multithreading** permet l'exécution simultanée ou pseudo-parallèle de plusieurs threads



technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Créer un Thread

- Un programme Java est constitué d'au minimum un thread, le thread principal

- Il y a deux manières de créer un thread :
  - Créer une classe dérivée de `java.lang.Thread`
  - Implémenter l'interface `java.lang.Runnable`

- Les threads Java sont gérés par la machine virtuelle Java

java.lang.Thread (1/2)

- Mise en œuvre :
  - Créer une classe dérivée de `java.lang.Thread`
  - Surcharger la méthode `run()`
  - Instancier un objet de cette sous-classe
  - Exécuter la méthode `start()` pour démarrer le thread

java.lang.Thread (2/2)

- Exemple en héritant de Thread. Ce thread fait une pause de 1s dès qu'on le démarre.

```
public class HelloThread extends Thread {

    public void run() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Hello");
    }

    public static void main(String[] args) {
        HelloThread h = new HelloThread();
        h.start();
    }

}
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

java.lang.Runnable (1/2)

- Le langage Java n'autorise pas l'héritage multiple …

- Mise en œuvre :

  - Implémenter l'interface `java.lang.Runnable`
  - Implémenter la méthode `run()`
  - Instancier la classe
  - Créer une instance de `Thread`, en passant en argument du constructeur une référence à l'objet implémentant `Runnable`
  - Exécuter la méthode `start()` pour démarrer le thread

java.lang.Runnable (2/2)

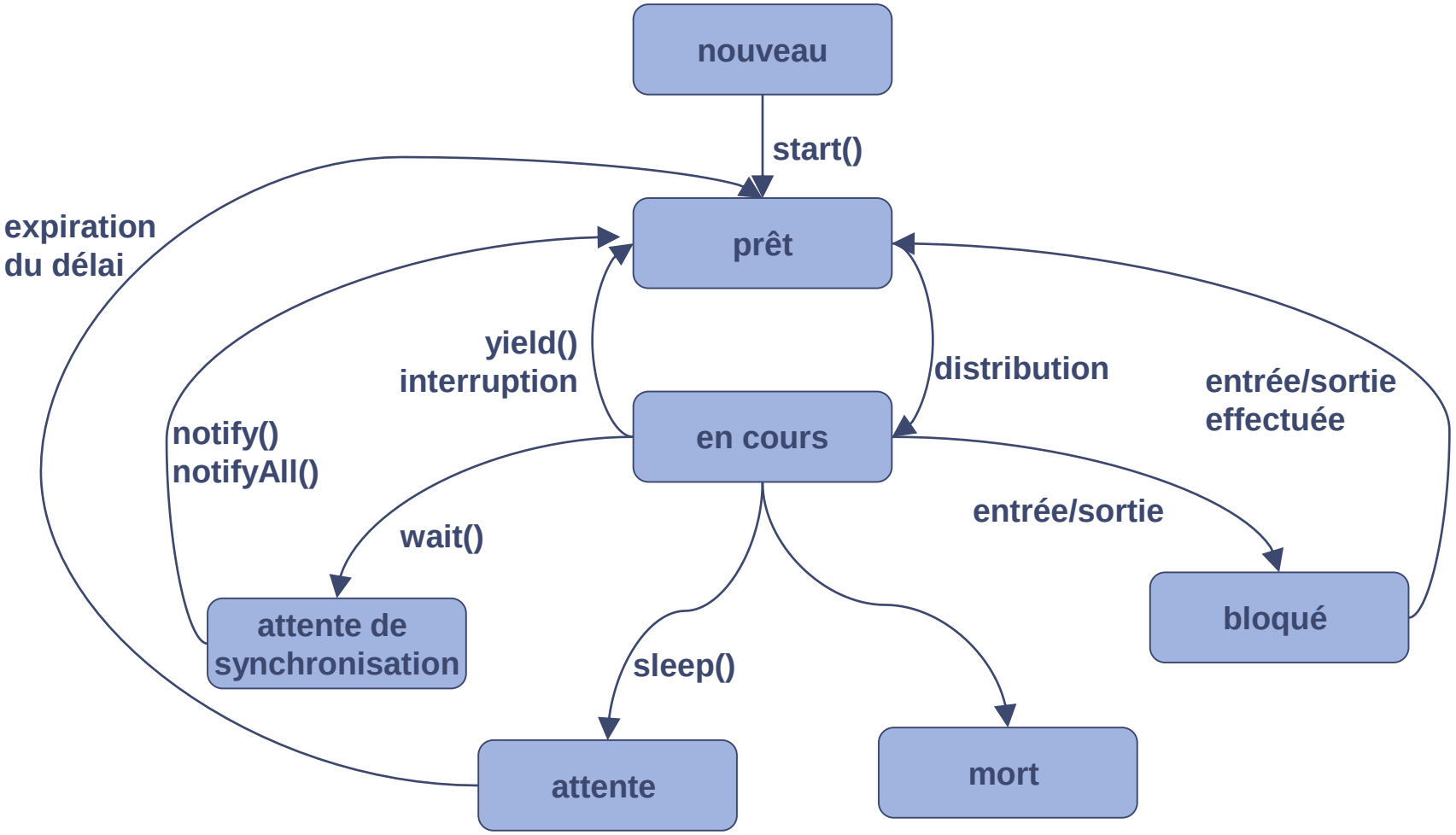
- Exemple d'un thread en implémentant l'interface Runnable. Comparez avec l'exemple précédent.

```
public class HelloRunnable implements Runnable {

    public void run() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Hello");
    }

    public static void main(String[] args) {
        HelloRunnable h = new HelloRunnable();
        new Thread(h).start();
    }

}
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Contrôler un Thread (1/2)



technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Contrôler un Thread (2/2)

- `Thread.sleep(long millisecond)` : stoppe le thread pour une période déterminée. Si d'autres threads sont prêts, le système autorise l'exécution de l'un d'eux

- `Thread.yield()` : passe la main à un autre thread de priorité égale ou supérieure

- `Thread.interrupt()` : demande l'arrêt prématuré du thread

  Si le thread est bloqué, cet appel provoque le lancement d'une exception `InterruptedException`

  Sinon, le statut du thread est mis à jour pour signaler la demande d'arrêt. `Thread.currentThread().isInterrupted()` permet de contrôler l'état du statut.

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Synchroniser des Threads

- Il est possible de coordonner les traitements effectués par des threads de différentes manières :
  - Synchronisation par **attente** de la fin d'exécution d'un thread
  - Exclusion mutuelle par **synchronisation de méthode**
  - Exclusion mutuelle par **synchronisation de bloc**
  - Barrière de synchronisation par **attente/notification**

Attente de la fin d'un thread (1/2)

- Il est possible d'obliger un thread à attendre la fin d'exécution d'un autre thread par appel de la méthode `join()` sur le thread dont on attend la fin

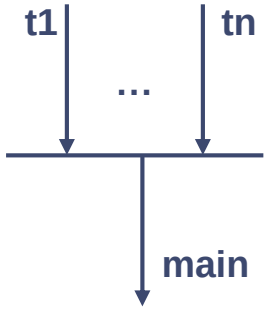
- Syntaxe
  - `void join()`
  - `void join(long timeoutMillisecond)`

Attente de la fin d'un thread (2/2)

- Exemple

Dans Eclipse, ouvrir be.wavenet.course.threads.Sleeper

```
public class Sleeper extends Thread {

    public void run() {
        try {
            Thread.sleep((long) (Math.random() * 1000));
        } catch (InterruptedException e) {}
    }

    public static void main(String[] args) {
        List<Sleeper> sleepers = new ArrayList<Sleeper>();

        for (int i = 0; i < 10; i++) {
            sleepers.add(new Sleeper());
            sleepers.get(i).start();
        }

        System.out.println("Attente …");
        for (Sleeper sleeper : sleepers) {
            try {
                sleeper.join();
            } catch (InterruptedException e) {}
        }
        System.out.println("Threads terminées");
    }

}
```



Synchronisation de méthode (1/3)

- Exemple

Dans Eclipse, ouvrir le wavenet course threads Account

```
public class Account {
    private double amount;

    public Account(double amount) {
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }

    public void setAmount(double amount) {
        this.amount = amount;
    }

    public void deposit(double amount) {
        this.amount += amount;
    }

    public void withdraw(double amount) {
        this.amount -= amount;
    }
}
```

Si deux threads concurrents appellent respectivement les méthodes withdraw() et deposit(), il n'est pas certain que le résultat sera juste …

technocité
Centre de compétence

Synchronisation de méthode (2/3)

- Empêcher l'exécution simultanée de la méthode par plusieurs threads ? Oui, pour empêcher la concurrence entre les threads.

- En plaçant le mot clé `synchronized` devant l'en-tête de la méthode

```
public synchronized void setAmount(…) { … }
```

- Empêche l'exécution simultanée pour une **même instance**
  - de la méthode
  - de toute autre méthode synchronisée
  - de toute portion de code synchronisée

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Synchronisation de méthode (3/3)

- Exemple, les méthode ne pourront être exécutées que par un thread à la fois.

```
public class ThreadSafeAccount extends Account {
    public ThreadSafeAccount(double amount) {
        super(amount);
    }

    synchronized public double getAmount() {
        return super.getAmount();
    }

    synchronized public void setAmount(double amount) {
        super.setAmount(amount);
    }

    synchronized public void deposit(double amount) {
        super.deposit(amount);
    }

    synchronized public void withdraw(double amount) {
        super.withdraw(amount);
    }
}
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Synchronisation de bloc

- Il est également possible d'empêcher l'exécution simultanée d'un bloc de code à l'aide du mot-clé `synchronized`

- Exemple

```
public void setAmount(double amount) {
    synchronized(this) {
        super.setAmount(amount);
    }
}
```

Attente / Notification (1/5)

- Il est possible de coordonner l'exécution de plusieurs threads utilisant une même resource à l'aide d'un mécanisme de **notification**

- Utilisation des méthodes `wait()` et `notify()` de la classe `java.lang.Object`

  - `wait()` pour suspendre le thread en cours d'exécution en attendant qu'une condition soit réalisée
  - `notify()` pour réactiver un thread mis en attente lorsque la condition est vérifiée
  - `notifyAll()` pour réactiver tous les threads en attente sur la condition

- Ces méthodes nécessitent un accès exclusif à l'objet, elles doivent donc être utilisées au sein d'une méthode `synchronized` ou un bloc `synchronized`

Attente / Notification (2/5)

- Exemple

```
public class Resource {

    private boolean isUsed = false;

    public synchronized void allocate() {
        while (isUsed) {
            try { wait(); }
            catch (InterruptedException ie) { }
        }
        isUsed = true;
    }

    public synchronized void free() {
        isUsed = false;
        notify();
    }

}
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Attente / Notification (3/5)

- Méthode `wait()`
  - Suspend l'exécution du thread courant
  - Libére le verrou assurant l'exclusion mutuelle, d'autres threads peuvent exécuter les méthodes synchronisées du même objet
  - Attend un appel à `notify()` ou `notifyAll()` par un autre thread pour le même objet

- Méthode `notify()`
  - Réactivation d'un thread mise en attente par `wait()` sur le même objet
  - Si plusieurs threads, un thread est choisi au hasard (en pratique, la JVM choisi le premier)

- Méthode `notifyAll()`
  - Réactivation de tous les threads mis en attente par `wait()` sur le même objet
  - *notify()* et *notifyAll()* permettent de débloquer une tâche bloquée par *wait()*

Attente / Notification (4/5)

- Exemple : Producteur / Consommateur

  Dans Eclipse, ouvrir be.wavenet.course.threads.ProducerConsumer

```
public class ProducerConsumer {
     public static void main(String[] args) {
          Queue queue = new Queue();
          new Producer(queue);
          new Consumer(queue);
     }
}

public class Queue {
     private int n;
     private boolean valueSet = false;

     public synchronized int get() {
          while (!valueSet) {
               try { wait(); } catch (InterruptedException e) {}
          }
          System.out.println("Got" + n);
          valueSet = false;
          notify();
          return n;
       }

     public synchronized void put(int n) {
          while (valueSet) {
               try { wait(); } catch (InterruptedException e) {}
          }
          this.n = n;
          valueSet = true;
          System.out.println("Put" + n);
          notify();
     }
}
```

Attente / Notification (5/5)

- … suite

```
public class Producer implements Runnable {
     private Queue queue;

     public Producer(Queue queue) {
          this.queue = queue;
          new Thread(this, "Producer").start();
     }

     public void run() {
          int i = 0;

          while (true)
               queue.put(i++);
     }
}

public class Consumer implements Runnable {
     private Queue queue;

     public Consumer(Queue queue) {
          this.queue = queue;
          new Thread(this, "Consumer").start();
     }

     public void run() {
          while (true)
               queue.get();
     }
}
```

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions

Daemon Thread (1/1)

- Un thread démon est un thread qui tourne en tâche de fond pendant toute la durée d'exécution du programme

- `void setDaemon(boolean isDaemon)` : marquer le thread comme thread démon ou thread utilisateur. La méthode doit être appelée avant que le thread n'ait été démarré !

- `boolean isDaemon()` : tester si un thread est un thread démon

technocité
Centre de compétence

WAVENET
Intelligent IT Solutions