SHAKE THE FUTURE

CENTRALE NANTES

Programmation Orientée Objet - Annexe 2 Threads (Processus légers) en Java

Jean-Marie Normand
jean-marie.normand@ec-nantes.fr
Bâtiment E - Bureau 211

Plan du cours I

Plan

Plan

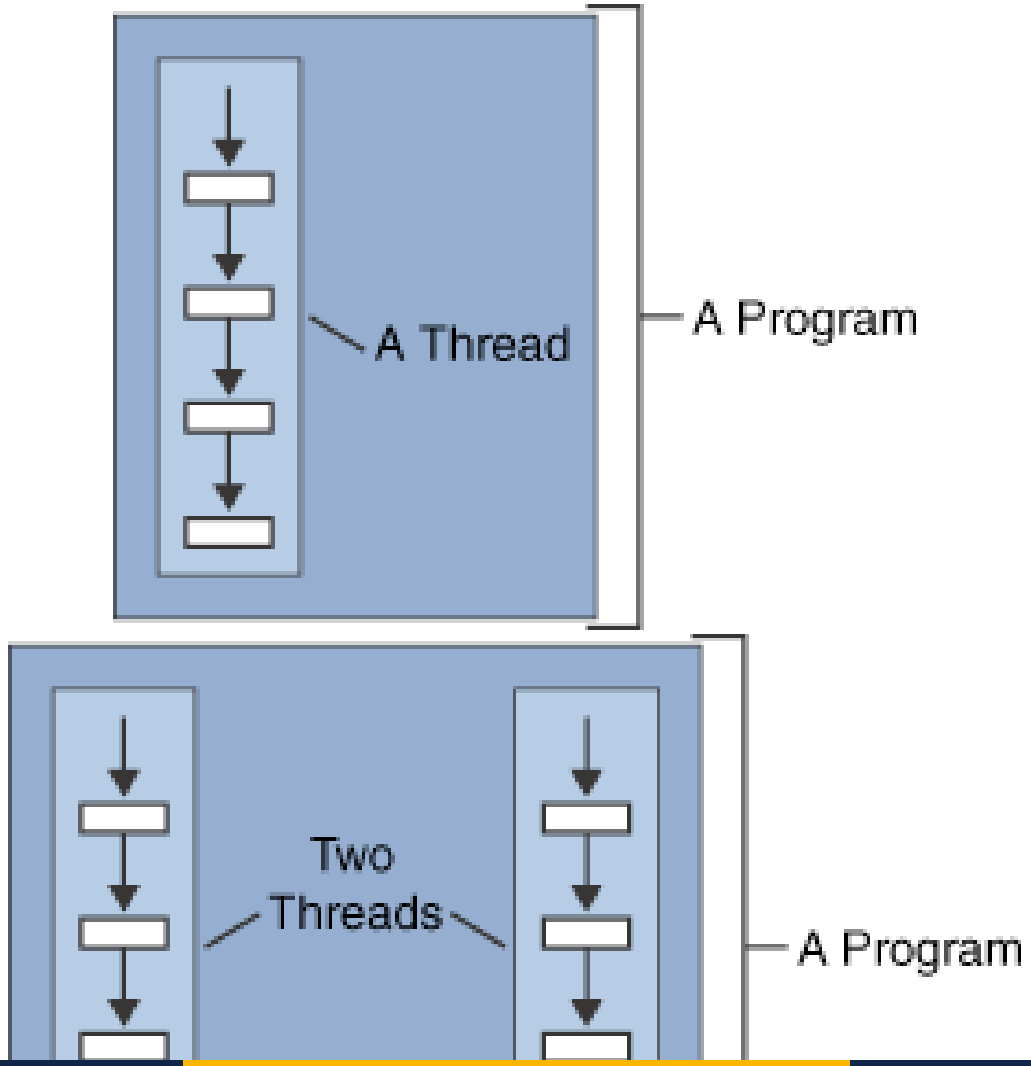
Les threads ?

- Habitudes de programmation : programmation séquentielle
  - Chaque programme a un début, une séquence d'exécution et une fin
  - A un instant t, il n'y a qu'un seul point d'exécution
- Thread : notion similaire avec début, point d'exécution et fin
- Mais ce n'est pas un programme, c'est un sous-ensemble d'un programme
- Le vrai intérêt consiste à en mettre plusieurs dans un même programme

Différences entre threads et processus

- À un processus peuvent être associés plusieurs threads (mais la réciproque est fausse)
  - tout processus possède au moins un thread (qui exécute le `main()`)
- Les ressources allouées à un processus sont partagées entre les threads qui le composent
- Contrairement aux processus, les threads partagent le même espace d'adressage
  - communication : plus facile mais plus dangereuse !

Un ou plusieurs threads ?



Les threads en Java : rappels sur la notion d'interface

- Une interface permet d'établir un ensemble de comportements abstraits définissant le contrat que doit respecter une classe :
  - Spécification d'un protocole en termes de signature de méthodes abstraites
  - Une interface ne contient pas de code
  - Une interface ne peut pas avoir de variables membres mais peut avoir des constantes de classes
  - Une classe implémentant l'interface s'engage à remplir le contrat

Les threads en java

- En Java, tout se passe dans `java.util.Thread`
- En gros, il faut instancier une classe `Thread` en lui fournissant une méthode `run()` qui provient de l'interface `Runnable`
- Après cela :
  - Cycle de vie des threads
  - Synchronisation des threads
  - Option Info filière GI ⇒ Cours Parallélisme et Temps réel !

Plan

Un premier exemple : les Timers

```
import java.util.Timer;
import java.util.TimerTask;
public class Reminder {
    Timer timer;
    public Reminder(int seconds) {
        timer = new Timer();
        timer.schedule(new RemindTask(), seconds*1000);
    }
    // class interne !
    class RemindTask extends TimerTask {
        public void run() {
            System.out.println("Time's up!");
            timer.cancel(); //Terminate the timer thread
        }
    }

    public static void main(String args[]) {
        new Reminder(5); // 5 seconds
        System.out.println("Task scheduled.");
    }
}
```

Que dire de l'exemple ?

- À l'exécution, on a bien le message `Task scheduled.`, suivi de quelques (combien d'après vous ?) secondes plus tard de `Time's up!`
- Création d'une sous-classe (interne) spécifique de `TimerTask` avec une méthode `run()` contenant le code à exécuter
- La classe `TimerTask` implémente l'interface `Runnable`
- Création d'un thread par instanciation (appel au constructeur) de la classe `Timer`
- Instanciation de la tâche du `Timer`
- Définition des paramètres du `Timer`

Arrêt d'un thread de timer

- Par défaut, un programme continue son exécution jusqu'à ce que tous ses threads soient terminés
- Mais on peut :
  - Invoquer la méthode `cancel()` sur le `Timer`
  - Le rendre `«daemon»` en créant le `Timer` avec l'argument `True`. S'il ne reste plus que des threads `«daemon»`, le programme s'arrête
  - Suppression de toutes les références au `Timer` dès que les tâches associées ont été exécutées
  - Invoquer `System.exit()`

Exemple : Tâches répétitives I

```
import java.util.Timer;
import java.util.TimerTask;
import java.awt.Toolkit;

public class AnnoyingBeep {
  Toolkit toolkit;
  Timer timer;
  public AnnoyingBeep() {
    toolkit = Toolkit.getDefaultToolkit();
    timer = new Timer();
    timer.schedule(new RemindTask(),0,1000);
  }
}
class RemindTask extends TimerTask {
  int numWarningBeeps = 3;
  public void run() {
    if (numWarningBeeps > 0) {
      toolkit.beep();
```

Exemple : Tâches répétitives II

```
      System.out.println("Beep!");
      numWarningBeeps--;
    }
    else {
      toolkit.beep();
      System.out.println("Time's up!");
      //timer.cancel(); //Not necessary because we call System.exit
      System.exit(0);   //Stops the AWT thread (and everything else)
    }
  }
  public static void main(String args[]) {
    System.out.println("About to schedule task.");
    new AnnoyingBeep();
    System.out.println("Task scheduled.");
  }
}
```

Différentes manières de spécifier le comportement d'un Timer

- `schedule(TimerTask task, long delay, long period)`
- `schedule(TimerTask task, Date time, long period)`
- `scheduleAtFixedRate(TimerTask task, long delay, long period)`
- `scheduleAtFixedRate(TimerTask task, Date firstTime, long period)`
- Attention, l'implémentation n'est pas *thread-safe* !
- *Thread-safe* : un code *thread-safe* est capable de fonctionner correctement et sans risque lors d'une exécution simultanée par plusieurs *threads*

CENTRALE NANTES

Plan

Création d'un thread en java

Deux manières principales de créer un thread

1. Créer une classe héritée de la classe `Thread` et surcharger la méthode `run()`
   - Appeler ensuite la méthode `start()`
2. Implémenter l'interface `Runnable`
   - Implémenter une méthode `run()`

Sous-classement de la classe Thread (1/2)


```
public class SimpleThread extends Thread {
    public SimpleThread(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
}
```

Sous-classement de la classe Thread (2/2)

```
public class TwoThreadsTest {
    public static void main (String[] args) {
        new SimpleThread("Jamaica").start();
        new SimpleThread("Fiji").start();
    }
}
```

Exemple d'exécution

```
0 Jamaica
0 Fiji
1 Fiji
1 Jamaica
2 Jamaica
3 Jamaica
4 Jamaica
5 Jamaica
2 Fiji
6 Jamaica
7 Jamaica
3 Fiji
4 Fiji
8 Jamaica
5 Fiji
6 Fiji
9 Jamaica
7 Fiji
8 Fiji
DONE! Jamaica
9 Fiji
DONE! Fiji
```

Utilisation de l'interface Runnable I

Voir le fichier `Clock.html` qui lance l'applet Java (attention à vos paramètres de sécurité Java !)

Figure: Applet Clock.

Utilisation de l'interface Runnable II

```
import java.awt.*; import java.util.*;
import java.applet.*; import java.text.*;

public class Clock extends java.applet.Applet implements Runnable {
    private volatile Thread clockThread = null;
    DateFormat formatter;      // Formats the date displayed
    String lastdate;           // String to hold date displayed
    Date currentDate;          // Used to get date to display
    Color numberColor;         // Color of numbers
    Font clockFaceFont;
    Locale locale;

    public void init() {
      setBackground(Color.white);
      numberColor = Color.red;
      locale = Locale.getDefault();
```

Utilisation de l'interface Runnable III

```
    formatter =
   DateFormat.getDateTimeInstance(DateFormat.FULL,DateFormat.MEDIUM, locale);
    currentDate = new Date();
    lastdate = formatter.format(currentDate);
    clockFaceFont = new Font("Sans-Serif",Font.PLAIN, 14);
    resize(275,25);
  }

  public void start() {
    if (clockThread == null) {
       clockThread = new Thread(this, "Clock");
       clockThread.start();
    }
  }


  public void run() {
```

Utilisation de l'interface Runnable IV

```
    Thread myThread = Thread.currentThread();
    while (clockThread == myThread) {
      repaint();
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e){ }
    }
}

public void paint(Graphics g) {
  String today;
  currentDate = new Date();
  formatter =
  DateFormat.getDateTimeInstance(DateFormat.FULL,DateFormat.MEDIUM, locale);
  today = formatter.format(currentDate);
  g.setFont(clockFaceFont);
```

Utilisation de l'interface Runnable V

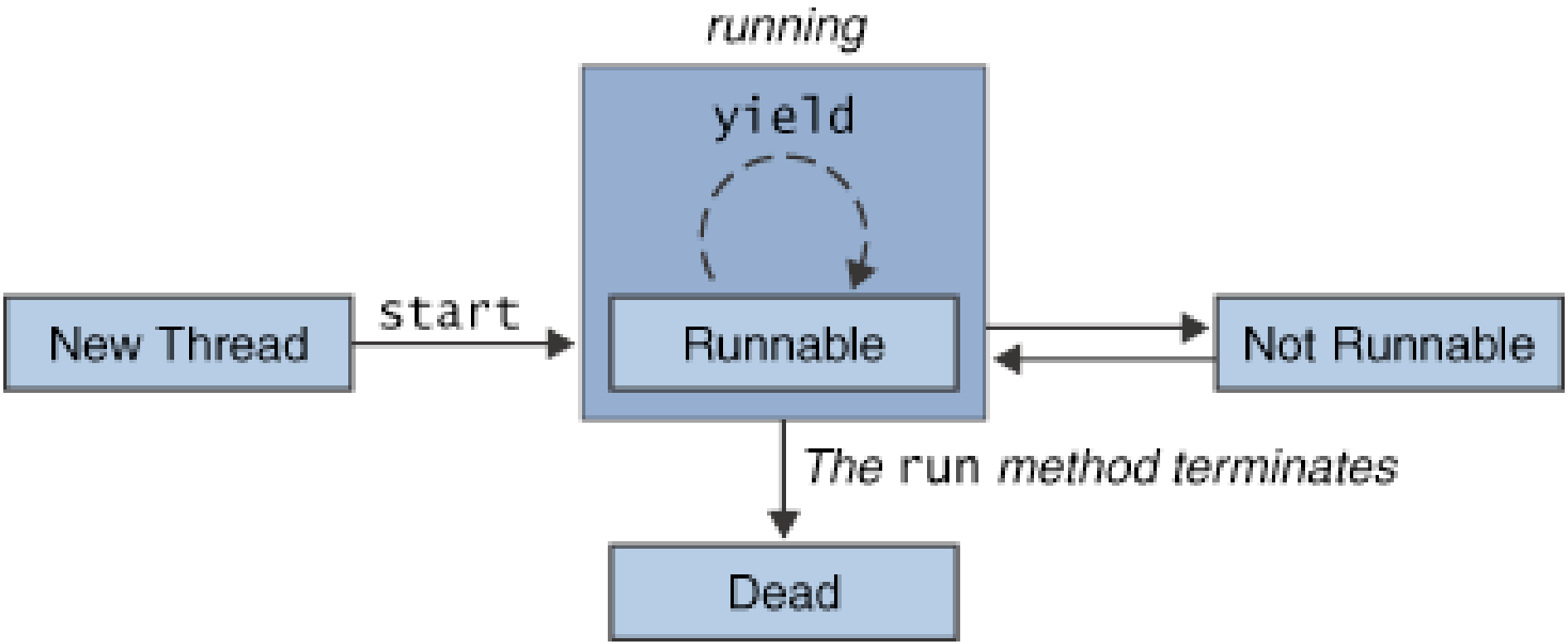
```
    // Erase and redraw
    g.setColor(getBackground());
    g.drawString(lastdate, 0, 12);
    g.setColor(numberColor);
    g.drawString(today, 0, 12);
    lastdate = today;
    currentDate=null;
}

public void stop() {
     clockThread = null;
}
```

Quelle solution choisir ?

- Il y a de bonnes raisons de choisir les deux solutions :
  - Sous-classement de la classe Thread et surcharge de `run()`
  - Classe qui implémente l'interface `Runnable`
- Dans la plupart des cas, si une classe qui crée un thread est une sous-classe d'une autre, il faudra utiliser `Runnable` puisque Java n'autorise pas l'héritage multiple
  - C'est le cas de l'applet horloge

Cycle de vie des threads



Plan

Partage de la mémoire entre threads

- les threads d'un même processus partagent le même espace mémoire
- chaque instance de la classe thread possède ses propres variables
- pour partager une variable entre threads, on a souvent recours à une variable de classe
- Exemple : partage d'une variable
  - 2 threads qui ont chacun un nom
  - ils vont ajouter ce nom à une chaîne de caractères commune

Exemple : Partage de variables

```
public class ExemplePartage extends Thread {
  private static String chaineCommune = "";
  private String nom;
  ExemplePartage ( String s ) {
    nom = s;
  }

  public void run() {
    chaineCommune = chaineCommune + nom;
  }

  public static void main(String args[]) {
    Thread T1 = new ExemplePartage( "T1" );
    Thread T2 = new ExemplePartage( "T2" );
    T1.start();
    T2.start();
    System.out.println( chaineCommune );
  }
}
```

Analyse

- Comportements possibles à l'exécution
  - aucun affichage : le thread principal affiche `chaineCommune` avant d'avoir donné la main à T1 et T2
  - T1 : au moment de l'affichage, T1 a été exécuté mais pas T2
  - T2 : inversement
  - T1T2 ou T2T1 : les deux threads ont été exécutés, dans un ordre arbitraire
- besoin d'un mécanisme de synchronisation !
- La méthode `join()` permet d'attendre qu'un thread soit terminé

Exemple : Problèmes d'accès concurrent I

```
public class ExempleConcurrent extends Thread {
  /// variable partagee par tous les threads
  private static int compte = 0;
  public void run() {
    int tmp = compte;
    try {
      Thread.sleep(1); // ms
    } catch (InterruptedException e) {
      System.out.println("ouch!\n");
      return;
    }
    tmp = tmp + 1;
    compte = tmp;
  }
  public static void main(String args[]) throws InterruptedException {
    Thread T1 = new ExempleConcurrent();
    Thread T2 = new ExempleConcurrent();
    T1.start();
```

Exemple : Problèmes d'accès concurrent II

```
    T2.start();
    T1.join();
    T2.join();
    System.out.println("compteur=" + compte);
  }
}
```

Les deux threads accèdent à une même variable partagée `compte`, travaillent sur une copie locale incrémentée avant d'être réécrite. `sleep()` permet de simuler un traitement plus long.

Le mot clé `synchronized`

- problèmes d'accès concurrent réglés par une directive `synchronized`
  - section critique : un seul thread à la fois
  - implanté par un verrou (lock, en fait un sémaphore)
- si un thread est dans une partie synchronisée, aucun autre thread ne peut y entrer
- attendre la libération du verrou ou l'appel de `wait()`

synchronisation temporelle : `wait()` et `notify()`

- Les méthodes `wait()`, `notify()` et `notifyAll()` permettent de synchroniser différents threads
  - définies dans la classe `Object`
  - à utiliser dans des méthodes `synchronized`
- `wait()` : le thread appelant bloqué jusqu'à un appel à `notify[All]()`
- `notify[All]()` : débloque un thread bloqué dans le même objet (autre tâche)

CENTRALE NANTES

Exemple : classe Evénement I

```
public class ExempleEvenement {
  private Boolean Etat; // etat de l'evenement
  ...

  public synchronized void set() {
    Etat = Boolean.TRUE; // debloque les threads qui attendent cet evenement:
    notifyAll();
  }

  public synchronized void reset() {
    Etat = Boolean.FALSE;
  }

  public synchronized void attente() {
    if(Etat==Boolean.FALSE) {
      try {
        wait(); // bloque jusqu'a un notify()
      }
```

Exemple : classe Evénement II

```
      catch(InterruptedException e) {
        // oups...
      };
    }
  }// fin attente
}// fin classe
```

Internetographie

Références Internet

- http://java.sun.com/docs/books/tutorial/essential/threads/definition.html
- http://perso.wanadoo.fr/jm.doudoux/java/tutorial/
- http://www-gtr.iutv.univ-paris13.fr/Cours/Mat/Systeme/CoursJavaThread.html
- http://alwin.developpez.com/tutorial/JavaThread/