

Les processus légers en Java : Threads

Stephane Malandain – POO - Java

I. Les threads

- 2 manières de définir un thread en Java :
 - En dérivant la classe `Thread`, en redéfinissant la méthode `run()`, puis en instanciant cette sous-classe de `Thread`

```
Class ThreadDerive extends Thread {  
    private int nFois;  
    ThreadDerive(int n) { nFois = n; }  
    public void run() {  
        for (int j=0;j<nFois;j++)    System.out.println("Exéc. no" + j);  
    }  
}
```

Et le main correspondant :

```
public class Main {  
    public static void main(String[] args) {  
        ThreadDerive td = new ThreadDerive(5);  
        td.start();  
    }  
}
```

I. Les threads

- en implémentant l'interface `Runnable` (ce qui correspond à une implémentation de la méthode `run()`) puis en passant une instance de type `Runnable` à un constructeur de la classe `Thread`.

```
class Boucle implements Runnable {  
    private int nFois;  
    Boucle(int n) { nFois = n; }  
    public void run() { for (int j=0;j<nFois;j++)  
                        System.out.println("Exéc. no" + j);  
    }  
}
```

- Création et démarrage d'un objet `Runnable`

```
public class Main {  
    public static void main(String[] args) {  
        Boucle bcl = new Boucle(4);  
        Thread td = new Thread(bcl);  
        td.start();  
    }  
}
```

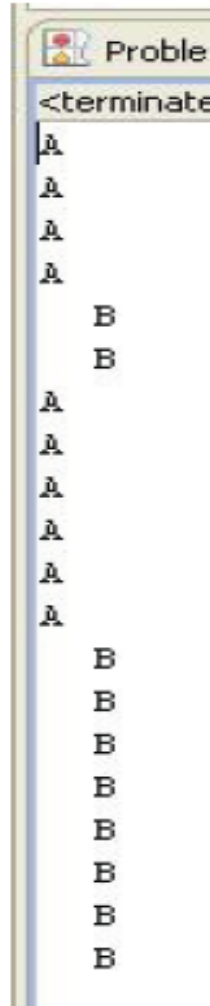
I. Les threads

- La méthode `run()` constitue le corps du thread. L'envoi du message `start()` à un objet `Thread` provoque son démarrage, c'est-à-dire l'exécution de sa méthode `run()`.
- Le thread se termine lorsqu'on quitte la méthode `run()`. La méthode `sleep(delai)` permet de mettre celui-ci en sommeil pendant `delai` millisecondes.

II. Exemple avec héritage

```
9
10
11 public class TestThread extends Thread {
12
13     public TestThread(String name) {
14         super(name);
15     }
16
17     public void run() {
18         for (int i=0;i<10;i++)
19             System.out.println(this.getName());
20     }
21 }
22
23 public class Test {
24     public static void main(String[] args) {
25         TestThread t = new TestThread( name: "A");
26         TestThread t2 = new TestThread( name: "B");
27         t.start();
28         t2.start();
29     }
30 }
31
```

II. Exemple avec héritage : Exécution



Essai de plusieurs Thread

III. Les états d'un thread

- Un thread peut présenter plusieurs états :
 - NEW : lors de sa création.
 - RUNNABLE : lorsqu'on invoque la méthode start() , le thread est prêt à travailler.
 - TERMINATED : lorsque le thread a effectué toutes ses tâches ; on dit aussi qu'il est « mort ». Vous ne pouvez alors plus le relancer par la méthode start() .
 - TIMED_WAITING : lorsque le thread est en pause (quand vous utilisez la méthode sleep() , par exemple).
 - WAITING : lorsque le thread est en attente indéfinie.
 - BLOCKED : lorsque l'ordonnanceur place un thread en sommeil pour en utiliser un autre, il lui impose cet état.

```
public class TestThread extends Thread {
    Thread t;
    public TestThread(String name) {
        super(name);
        System.out.println("statut du thread " + name + " = ")
+this.getState());
        this.start();
        System.out.println("statut du thread " + name + " = ")
+this.getState());
    }

    public TestThread(String name, Thread t) {
        super(name);
        this.t = t;
        System.out.println("statut du thread " + name + " = ")
+this.getState());
        this.start();
        System.out.println("statut du thread " + name + " = ")
+this.getState());
    }

    public void run() {
        for(int i = 0; i < 10; i++) {
            System.out.println("statut " + this.getName() + " = ")
+this.getState());
            if(t != null)
                System.out.println("statut de " + t.getName() + " pendant le
thread " + this.getName() + " = " + t.getState());
        }
    }

    public void setThread(Thread t) {
        this.t = t;
    }
}
```



```
public class Test {  
    public static void main(String[] args) {  
        TestThread t = new TestThread("A");  
        TestThread t2 = new TestThread(" B", t);  
        try {  
            Thread.sleep(1000);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        System.out.println("statut du thread " + t.getName() + " = " +  
t.getState());  
        System.out.println("statut du thread " + t2.getName() + " = "  
+t2.getState());  
    }  
}
```

```
<terminated> Test (4) [Java Application] C:\Program Files\Java\jre1.6.0_03\bin\javaw.exe
statut du thread A = NEW
statut du thread A = RUNNABLE
statut du thread B = NEW
statut du thread B = RUNNABLE
statut A = RUNNABLE
statut A = RUNNABLE
statut A = RUNNABLE
statut A = RUNNABLE
statut A = RUNNABLE
statut B = RUNNABLE
statut de A pendant le thread B = RUNNABLE
statut B = RUNNABLE
statut de A pendant le thread B = BLOCKED
statut B = RUNNABLE
statut de A pendant le thread B = BLOCKED
statut B = RUNNABLE
statut de A pendant le thread B = BLOCKED
statut B = RUNNABLE
statut de A pendant le thread B = BLOCKED
statut B = RUNNABLE
statut de A pendant le thread B = BLOCKED
statut B = RUNNABLE
statut de A pendant le thread B = BLOCKED
statut B = RUNNABLE
statut de A pendant le thread B = BLOCKED
statut A = RUNNABLE
statut A = RUNNABLE
statut A = RUNNABLE
statut A = RUNNABLE
statut de A pendant le thread B = BLOCKED
statut B = RUNNABLE
statut de A pendant le thread B = BLOCKED
statut A = RUNNABLE
statut du thread A = TERMINATED
statut du thread B = TERMINATED
```

Test avec plusieurs threads simultanés

III. Thread – autre exemple

```
final List liste; // liste d'objets non-triés, supposée initialisée

// classe dérivant Thread pour trier la liste en tâche de fond
class TriBackground extends Thread {
    List l;
    public TriBackground(List l) { this.l = l; }
    public void run() { Collections.sort(l); } // corps du thread
}

// création d'un thread de type TriBackground
Thread trieur = new TriBackground(liste);
// démarrage du thread avec exec de la méthode run() pendant que le thread
// original poursuit son travail.

trieur.start();

// création équivalent d'un thread anonyme pour trier en tâche de fond

new Thread(new Runnable() {
    public void run() { Collections.sort(liste); }
}).start();
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class Fenetre extends JFrame{
    private Panneau pan = new Panneau();
    private JButton bouton = new JButton("Go");
    private JButton bouton2 = new JButton("Stop");
    private JPanel container = new JPanel();
    private JLabel label = new JLabel("Le JLabel");
    private int compteur = 0;
    private boolean animated = true;
    private boolean backX, backY;
    private int x, y;
    private Thread t;

    public Fenetre(){
        //Le constructeur n'a pas changé
    }

    private void go(){
        //La méthode n'a pas changé
    }
}
```

Suite

```
public class BoutonListener implements ActionListener{
    public void actionPerformed(ActionEvent arg0) {
        animated = true;
        t = new Thread(new PlayAnimation());
        t.start();
        bouton.setEnabled(false);
        bouton2.setEnabled(true);
    }
}

class Bouton2Listener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        animated = false;
        bouton.setEnabled(true);
        bouton2.setEnabled(false);
    }
}

class PlayAnimation implements Runnable{
    public void run() {
        go();
    }
}
```

Exemple avec Runnable

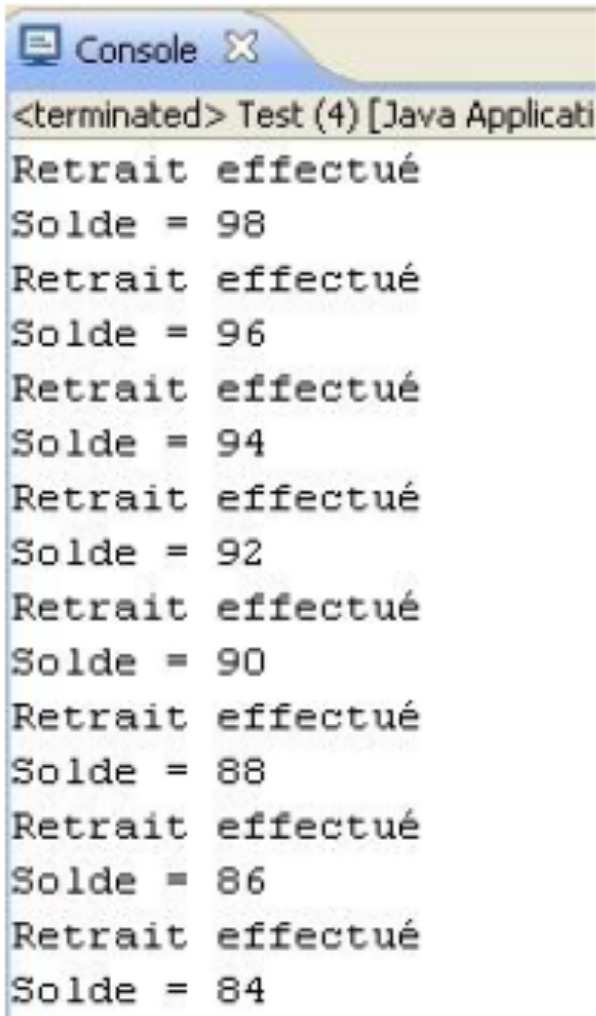
```
public class RunImpl implements Runnable {  
    private CompteEnBanque cb;  
  
    public RunImpl(CompteEnBanque cb) {  
        this.cb = cb;  
    }  
    public void run() {  
        for(int i = 0; i < 25; i++) {  
            if(cb.getSolde() > 0) {  
                cb.retraitArgent(2);  
                System.out.println("Retrait effectué");  
            }  
        }  
    }  
}
```

Exemple avec Runnable

```
public class CompteEnBanque {  
    private int solde = 100;  
  
    public int getSolde(){  
        if(this.solde < 0)  
            System.out.println("Vous êtes à découvert !");  
  
        return this.solde;  
    }  
  
    public void retraitArgent(int retrait){  
        solde = solde - retrait;  
        System.out.println("Solde = " + solde);  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        CompteEnBanque cb = new CompteEnBanque();  
        Thread t = new Thread(new RunImpl(cb));  
        t.start();  
    }  
}
```

Exemple avec Runnable



```
<terminated> Test (4) [Java Applicati  
Retrait effectué  
Solde = 98  
Retrait effectué  
Solde = 96  
Retrait effectué  
Solde = 94  
Retrait effectué  
Solde = 92  
Retrait effectué  
Solde = 90  
Retrait effectué  
Solde = 88  
Retrait effectué  
Solde = 86  
Retrait effectué  
Solde = 84
```

Premier test de retrait d'argent

On ajoute un 2^{ème} thread

```
public class RunImpl implements Runnable {  
    private CompteEnBanque cb;  
    private String name;  
  
    public RunImpl(CompteEnBanque cb, String name){  
        this.cb = cb;  
        this.name = name;  
    }  
  
    public void run() {  
        for(int i = 0; i < 50; i++){  
            if(cb.getSolde() > 0){  
                cb.retraitArgent(2);  
                System.out.println("Retrait effectué par " + this.name);  
            }  
        }  
    }  
}
```

On ajoute un 2^{ème} thread

```
public class Test {  
    public static void main(String[] args) {  
        CompteEnBanque cb = new CompteEnBanque();  
        CompteEnBanque cb2 = new CompteEnBanque();  
  
        Thread t = new Thread(new RunImpl(cb, "Cysboy"));  
        Thread t2 = new Thread(new RunImpl(cb2, "Zéro"));  
        t.start();  
        t2.start();  
    }  
}
```

Même fonctionnement, chaque thread sur son compte

2 threads sur le même objet

```
public class Test {  
    public static void main(String[] args) {  
        CompteEnBanque cb = new CompteEnBanque();  
  
        Thread t = new Thread(new RunImpl(cb, "Cysboy"));  
        Thread t2 = new Thread(new RunImpl(cb, "Zéro"));  
        t.start();  
        t2.start();  
    }  
}
```

2 threads sur le même objet

```
Retrait effectué par Cysboy
Solde = 92
Retrait effectué par Cysboy
Solde = 90
Solde = 86
Retrait effectué par ZérO
Solde = 84
Retrait effectué par ZérO
Retrait effectué par Cysboy
Solde = 88
Solde = 82
Retrait effectué par ZérO
Solde = 80
Retrait effectué par ZérO
Solde = 78
Retrait effectué par ZérO
Solde = 76
Retrait effectué par ZérO
Solde = 74
Retrait effectué par ZérO
Solde = 72
Retrait effectué par ZérO
Solde = 70
Retrait effectué par ZérO
Solde = 68
Retrait effectué par ZérO
Retrait effectué par Cysboy
Solde = 66
Solde = 86
Retrait effectué par ZérO
Retrait effectué par Cysboy
```

Retrait multithreadé

Solution : synchronisation

```
public class CompteEnBanque {  
    //Le début du code ne change pas  
  
    public synchronized void retraitArgent(int retrait) {  
        solde = solde - retrait;  
        System.out.println("Solde = " + solde);  
    }  
}
```

IV. Thread – Synchronisation

- Lorsqu'on utilise plusieurs threads, il faut veiller aux accès concurrents à une ressource.
- Exemple : un thread parcourt une liste pendant que l'autre la trie.
- Un thread peut verrouiller l'objet avant de le modifier, bloquant ainsi l'accès de cette ressource aux autres threads.
- On dit que le thread détient dans ce cas le **verrou**. Tout objet java peut être verrouillé.

IV. Thread – Synchronisation (2)

- On verrouille une méthode d'une classe avec le modificateur **synchronized**.
- Un thread voulant envoyer un message à un objet d'une classe avec une méthode `synchronized` devra d'abord obtenir le verrou sur cet objet, empêchant par-là même tout autre thread d'exécuter pendant ce temps la méthode `synchronized` de cet objet.
- Si la méthode est statique, le thread doit obtenir un verrou sur la classe, le mécanisme étant similaire.
- Il est possible de verrouiller un objet au niveau d'un bloc.

IV. Thread – Coordination

- Les méthodes `wait()` et `notify()` de la classe `Objet` permettent de coordonner l'action des threads, par mise en attente ou activation (réveil)
- Chaque objet Java à un verrou qui lui est associé, et gère une liste de threads en attente.
- Quand un thread appelle la méthode `wait()` d'un objet, les verrous détenus par ce thread sont temporairement restitués. Il est alors bloqué et ajouté à la liste des threads en attente de l'objet.
- Lorsqu'un autre appelle la méthode `notify()` du même objet, celui-ci réveille un des threads en attente qui peut poursuivre son exécution. La méthode `notifyAll()` les réveille tous !

IV. Thread – Coordination - exemple

```
// un thread insère un objet dans la queue via inserer(), un autre
// en retire un avec extraire(); en l'absence de données, extraire()
// met en attente tandis que inserer() avertit de la présence de données

public class Queue {
    LinkedList lst = new LinkedList ();
    public synchronized void inserer (Object obj) {
        lst.add(obj); // ajoute l'objet en queue de liste
        this.notify(); // avertit les threads en attente
                        // que les données sont disponibles
    }

    public synchronized Object extraire() {
        while (lst.empty()) {
            try { lst.wait(); } // mise en attente du thread appelant
            catch( InterruptedException ie ) { }
        }
        return lst.remove(0);
    }
}
```

III. Thread – Timers

- Les classes `Timer` et `TimerTask` facilitent la gestion de tâches répétitives.

```
final DateFormat f = DateFormat.getInstance(DateFormat.MEDIUM);

// définition de la tâche d'affichage du temps
TimerTask displayTemps = new TimerTask() {
    public void run() { System.out.println(f.format(new Date())); }
}

// création d'un objet Timer pour gérer la tâche
Timer timer = new Timer();

// exécution de la tâche toutes les secondes dès maintenant
Timer.schedule(displayTemps, 0, 1000);

// arrêt de la tâche d'affichage du temps
displayTemps.cancel();
```

- La classe `java.util.Timer` permet de lancer un processus une ou plusieurs fois en précisant des délais.
- Un `Timer` gère les exécutions d'une instance de `TimerTask`, classe qui implémente `Runnable`.

```
public class ExempleTimer extends TimerTask{  
  
    public void run(){  
        try{  
            System.out.println("je m'execute");  
            Thread.sleep(500);  
        }  
        catch(InterruptedException e){System.out.println(e.getMessage());}  
    }  
}
```

```
Timer t = new Timer();  
// la tâche se répètera toutes les 2s et démarre dans 1s  
t.schedule(new ExempleTimer(),1000,2000);  
Date d = new Date();  
d.setTime(d.getTime()+10000);  
// la tâche démarre dans 10s  
t.schedule(new ExempleTimer(),d);
```