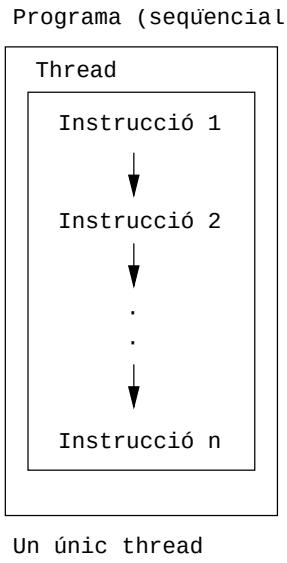
Pràctica. Introducció a la concurrència. Fils d'execució. Sincronització bàsica amb espera activa

Josep Cotrina, Marcel Fernández, Jordi Forga, Juan Luis Gorricho, Francesc Oller

1 Introducció als Threads

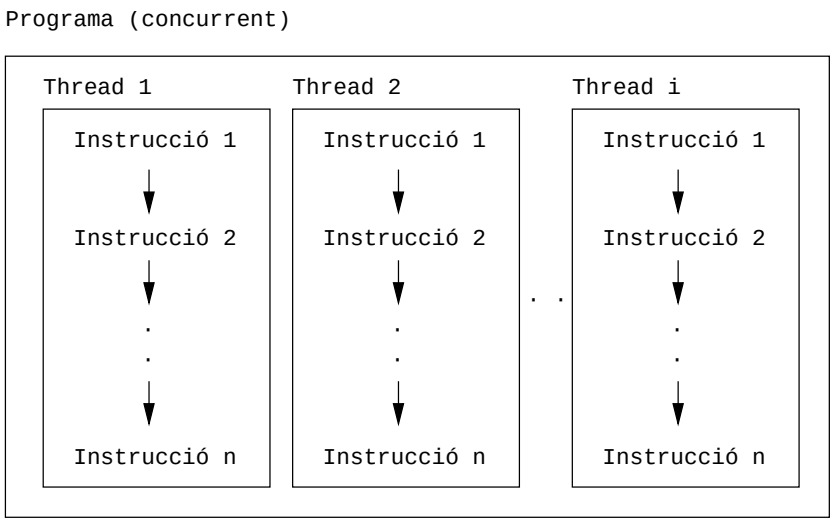
En programació seqüencial, la tasca que realitza un procés es correspon amb una successió d'instruccions que conformen un sol fil d'execució o *thread*. Si el procés ha de realitzar més d'una tasca, llavors les realitza una a continuació de l'altra.



Per altra banda, una aplicació pot tenir més d'un fil d'execució. Cada un d'aquests fils o threads, es correspon amb una tasca i per tant és senzillament una seqüència d'instruccions. Aquest tipus de programació s'anomena programació *multiprocés* o *multithreading*. Quan una aplicació es composa de més d'un thread, llavors l'execució física del programa dependrà de la plataforma que té per sota. Per exemple suposem que disposem de les dues plataformes següents, que a més són dos casos extrems. En una plataforma hi ha tants processadors físics (no necessàriament iguals) com threads, en l'altra plataforma hi ha un sol processador.

En la primera plataforma, es pot assignar un thread a cada processador i per tant l'execució dels threads serà *paral·lela* en el temps. En altres paraules, a ulls d'un observador extern al programa, les instruccions dels diferents threads s'executen entrellaçades en el temps.

En la segona plataforma, en la qual només es disposa d'un processador, en un cert instant de temps s'està executant una instrucció pertanyent a un determinat thread. En un instant de temps posterior (que dependrà del tipus de planificació), s'executaran instruccions pertanyents a un altra thread. Per tant, les instruccions s'executen *concurrentment* i també entrellaçades en el temps.



Seguint el raonament dels dos paràgrafs anteriors, podem dir que desde un punt de vista conceptual, les instruccions dels diferents threads s'executaran de manera entrellaçada independentment de la plataforma de la que disposem. A més, cal tenir en compte que l'ordre de l'entrellaçat es desconegut a priori i que en dues execucions diferents del programa l'entrellaçat serà, en general, diferent.

En aquesta pràctica, es proposa que experimenteu amb threads.

2 JAVA i Multithreading

En aquesta secció, es descriu la sintaxi JAVA per a threads. Hi ha dues formes d'obtenir threads an JAVA.

La primera consisteix en heretar de la classe Thread i redefinir-ne el mètode `run()`, tal com es mostra a continuació.

```
class MeuThread extends Thread {

    public void run() {
        // Mètode que realitza les
        // tasques del thread
    }
}
```

La segona consisteix en implementar l'interfície `Runnable`. Per això cal implementar el mètode `run()` d'aquesta interfície.

```
class MeuRunnable implements Runnable {

    public void run() {
        // Mètode que realitza les
        // tasques del thread
        }
}
```

Un objecte de tipus `Runnable` s'executarà en el seu propi fil d'execució si el passem en el constructor de Thread.

```
...

Thread unThread = new Thread(new MeuRunnable());
```

```
...
```

Un cop definida una classe el que cal és construir objectes d'aquesta classe perquè realitzin les tasques que desitgem. El codi següent mostra com crear i executar un programa que conté dos threads

```
public class ExecucioThreads {
    public static void main (String[] args) {
        // A partir de Thread
        new MeuThread().start();

        // A partir de Runnable
        new Thread(new MeuRunnable()).start();
    }
}
```

La sentència

```
new MeuThread().start();
```

per una banda crea un objecte de tipus `MeuThread`, i per altra banda crida el seu mètode `start()`. La invocació del mètode `start()` implica l'execució de les sentències que componen el mètode `run()` de la classe `MeuThread` en un nou fil d'execució.

3 Exercicis a realitzar

3.1 Primera part

Realitzar un programa amb dos threads, en el qual un dels threads imprimeixi lletres `A` per pantalla i l'altra thread imprimeixi lletres `B`. Què s'observa?

3.2 Segona part

Utilitzar el mètode `sleep()` de la classe `Thread` per modificar el programa de la part anterior de manera que les lletres `A` i `B`, estiguin intercalades una a una, es a dir,

```
A
B
A
B
A
B
.
.
```

Què s'observa? S'aconsegueix un intercalat perfecte? Comproveu-ho guardant la sortida a un fitxer: `java ABsleep >/tmp/ABsleep.log` i executant l'script `sed -n 'H;x;s/\(.\)\n\1/\0\n--/p' /tmp/ABsleep.log` Si surt quelcom per consola és que hi han `A`'s o `B`'s repetides.

3.3 Tercera part

Trobeu un entrellaçat que justifiqui el comportament anòmal de l'apartat anterior.

Cóm aconseguiríeu evitar que surtin `A`'s o `B`'s repetides utilitzant únicament `sleep()`? Per concloure: és `sleep()` un bon mecanisme per aconseguir l'alternança de lletres? perquè?

3.4 Quarta part

Resolgueu l'apartat anterior fent servir una variable simple compartida que en un estat permet escriure A i en l'altre B. Feu servir la plantilla:

```
...
public void run() {
    while (true) {
        flag.waitLletra(lletra);
        System.out.println(lletra);
        flag.nextLletra();
    }
}
...
```

programant la classe auxiliar que fa falta.