

## GESTIÓN DE HILOS

### 1.- Creación de hilos mediante *Thread* o bien interfaz *Runnable*.

Creación de un hilo con paso de parámetros:

```
MiHilo h = new MiHilo("Hilo 1",200);
```

Si va bien tendremos en “h” el objeto hilo y para arrancarlo:

```
h.start(h)
```

Si es con la interfaz *Runnable*

```
new Thread(h).start();
```

Esto lo que hará es llamar al método *run*. Al finalizar el método *run* se finalizarán también los hilos.

### 2.- Suspensión del hilo.

- *sleep()*: detiene el hilo x milisegundos.
- *suspend()*: detiene el hilo hasta que se invoque al método *resume()*. Útil para applets de animaciones. No se usa porque si un hilo está bloqueando un recurso y el hilo se suspende el recurso queda bloqueado.

*Para suspender de forma segura utilizaremos una variable y comprobaremos su valor dentro del método run.*

```
class MyThread extends Thread {  
    private SuspendRequestor suspender = new SuspendRequestor();  
  
    public void requestSuspend() { suspender.set(true); }  
    public void requestResume() { suspender.set(false);}  
    public void run() {  
        try {  
            while (haya trabajo por hacer) {  
                suspender.waitForResume();  
                // realiza trabajo  
            }  
        } catch (InterruptedException exception) {  
            ///  
        }  
    }  
} //
```

Hemos envuelto la variable y mediante el método *set()* le damos valor true o false, y luego llama a *notifyAll()* para avisar a todos los hilo que se pusieron en espera con *wait()*. El método *waitForResume()* hará un *wait()* cuando la variable sea true.

```
class SustendRequistor {  
    private boolean suspendRequested;  
    public synchronized void set (boolean b) {  
        suspendRequest = b;  
        notifyAll();  
    }  
    public synchronized void waitForResume()  
        throws InterruptedException {  
        while (suspendRequested)  
            wait();  
    }  
}
```

## El método wait

### 3.- Parada de un hilo

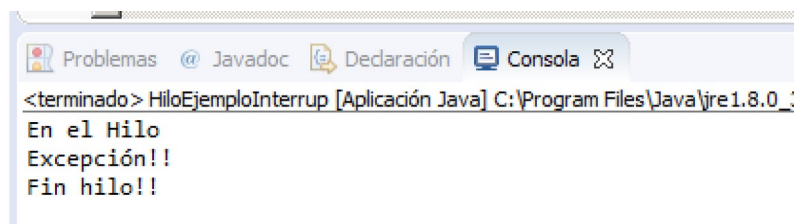
El método **stop()** para el hilo de forma **permanente**. No se puede reanudar con **start()**. NO USAR ya que puede dar problemas de inconsistencia, en su lugar hay que usar una variable, **stop()**, al igual que **suspend()**, **resume()** y **destroy()** hay que evitar utilizarlos.

El método **interrupt()**, envía una petición de interrupción a un hilo, pero si el hilo está bloqueado por un *wait* o *sleep*, se lanza una excepción **InterruptedException**. Si el hilo está interrumpido *isInterrupted()* devolverá *true*.

```
package InterrupcionHilos;

public class HiloEjemploInterrup extends Thread {
    public void run() {
        try {
            while (!isInterrupted()) {
                System.out.println("En el Hilo");
                Thread.sleep(10);
            }
        } catch (InterruptedException e) {
            System.out.println("Excepción!!");
        }
        System.out.println("Fin hilo!!");
    }
    public void interrumpir() {
        interrupt();
    }
    public static void main(String[] args) {
        HiloEjemploInterrup h = new HiloEjemploInterrup();
        h.start();
        for (int i=0; i<10000000; i++); // no hago ná
        h.interrumpir();
    }
}
```

## SALIDA



```
<terminado> HiloEjemploInterrup [Aplicación Java] C:\Program Files\Java\jre1.8.0_...
En el Hilo
Excepción!!
Fin hilo!!
```

Si eliminamos la línea **Thread.sleep(10)** también hay que quitar el **try-catch**, la interrupción es recogida por **isInterrupted()** que valdrá true, el **run** terminará y consecuentemente el **hilo**.

### 3.1.- Join()

*Provoca que el hilo que hace la llamada espere la finalización de otros hilos.*

```
h1.join();
```

quedaríamos a la espera a que finalice el hilo h1.

```
package JOIN;

class HiloJoin extends Thread {
    private int n;
    public HiloJoin(String nom, int n) {
        super(nom);
        this.n=n;
    }
    public void run() {
        for(int i=1; i<=n; i++) {
            System.out.println(getName() +": "+i);
            try {
                sleep(1000);
            } catch (InterruptedException ignore) {}
        }
        System.out.println("Fin de ciclo "+getName());
    }
}
```

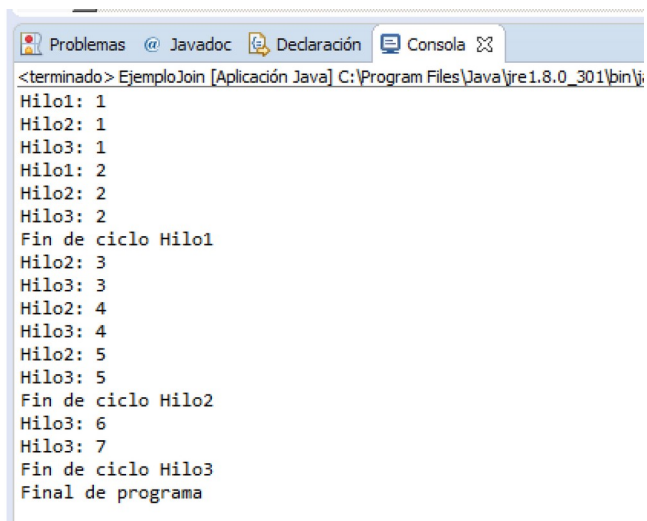
```
package JOIN;

public class EjemploJoin {
    public static void main(String[] args) {
        HiloJoin h1 = new HiloJoin("Hilo1",2);
        HiloJoin h2 = new HiloJoin("Hilo2",5);
        HiloJoin h3 = new HiloJoin("Hilo3",7);
        h1.start();
        h2.start();
        h3.start();
        try {
            h1.join();
            h2.join();
            h3.join();
        } catch (InterruptedException e) { }
        System.out.println("Final de programa");
    }
}
```

### Comentario

En el método **main()** se crean 3 hilos, cada uno da un valor diferente a la **n**, el primero es el valor más pequeño y el último es el valor más grande. A la hora de ir finalizando los hilos el primero debería terminar antes que el último ya que tiene menos carga de instrucciones. Llamando a **join()** podemos hacer que el programa principal se espere a la finalización de los hilos y cada hilo finaliza en el orden marcado según el **join()**.

SALIDA



```
<terminado> EjemploJoin [Aplicación Java] C:\Program Files\Java\jre1.8.0_301\bin\j...
Hilo1: 1
Hilo2: 1
Hilo3: 1
Hilo1: 2
Hilo2: 2
Hilo3: 2
Fin de ciclo Hilo1
Hilo2: 3
Hilo3: 3
Hilo2: 4
Hilo3: 4
Hilo2: 5
Hilo3: 5
Fin de ciclo Hilo2
Hilo3: 6
Hilo3: 7
Fin de ciclo Hilo3
Final de programa
```