

Hilos



Resumen

- Hilos
 - Aspectos generales
 - Estados de los hilos
 - Creación
 - Funcionamiento
 - Prioridades
- Concurrencia
- Secciones críticas
- Notificaciones



Objetivos

- Conocer y comprender que son los hilos y cómo se utilizan en Java
- Conocer y comprender el concepto de concurrencia y cómo aplicarlo al uso de hilos en Java
- Conocer y comprender qué son las secciones críticas y cómo funcionan en Java
- Realizar aplicaciones sencillas en Java que manejen hilos y los conceptos introducidos en este tema



Hilos – Aspectos Generales

- En la programación multihilo se conmuta entre distintas partes del código de un mismo programa durante su ejecución
- Los hilos permiten realizar tareas simultáneamente
- Cada una de estas partes individuales se llama *Thread* (hilos o contextos de ejecución)
- La JVM los ejecuta de forma:
 - Simultánea en una máquina multiprocesador
 - Conmutada en máquina de un solo procesador (algoritmo de planificación para crear sensación de simultaneidad)
- Al ejecutar un programa Java hay varios hilos ejecutándose (manejo de eventos, main, ...)



Estados de un hilo

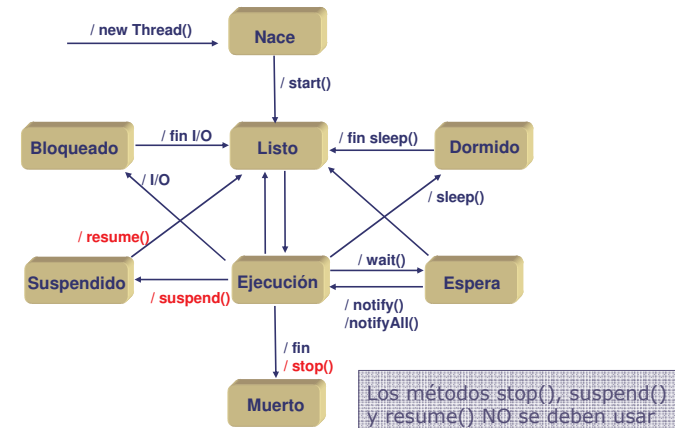
- Nace: Declarado pero todavía no se ha ejecutado start()
- Listo: Preparado para entrar en ejecución pero el planificador aun no ha decidido su puesta en marcha
- Ejecutándose: Está ejecutándose en la CPU
- Dormido: Se ha detenido durante un instante de tiempo definido mediante la utilización de sleep()
- Bloqueado: Pendiente de una operación de I/O y no volverá al estado listo hasta que esta termine
- Suspendido: Detenido temporalmente por suspend(), se reanuda con resume() (Deprecated - NO usar)
- Esperando: Detenido por una condición llamando a wait(), se reanuda con notify() o notifyAll()
- Muerto: Ha terminado su ejecución o porque terminó su trabajo o porque se llamó a stop() (Deprecated - NO usar)



Programación Orientada a Objetos



Estados de un hilo



Programación Orientada a Objetos



Creación de Hilos – Primera posibilidad

- Crear subclase de java.lang.Thread, haciendo overriding del método run de la clase thread

```
public class Hilo extends Thread{
    public void run(){
        while(true){
            S.O.P. ("Hola mundo, soy el hilo!!!");
        }
    }
}
```

- Creación del hilo y puesta en marcha:

```
Thread ht = new Hilo();
ht.start();
```

```
Hola mundo, soy el hilo!!!
Hola mundo, soy el hilo!!!
Hola mundo, soy el hilo!!!
Hola mundo, soy el hilo!!!
Hola mundo, soy el hilo!!!
Hola mundo, soy el hilo!!!
Hola mundo, soy el hilo!!!
Hola mundo, soy el hilo!!!
```



Programación Orientada a Objetos



Creación de Hilos – Primera posibilidad - Ejemplo

```
class TestDosHilos {
    public static void main (String[] args) {
        new SimpleThread("Primero").start();
        new SimpleThread("Segundo").start();
    }
}
class SimpleThread extends Thread {
    public SimpleThread(String str) { super(str); }
    public void run() {
        for (int i = 0; i < 10; i++) {
            S.O.P.(i + " " + getName());
            try { sleep((int)(Math.random() * 1000)); // 1 seg
            } catch (InterruptedException e) {}
        } System.out.println("Hecho! " + getName());
    }
}
```

El método sleep lanza una InterruptedException que debe ser capturada



Programación Orientada a Objetos



Creación de Hilos – Primera Posibilidad – Ejemplo - Ejecución

```

0 Primero      0 Primero      0 Primero
0 Segundo     0 Segundo     0 Segundo
1 Segundo     1 Primero     1 Primero
1 Primero     2 Primero     1 Segundo
2 Segundo     1 Segundo     2 Primero
2 Primero     2 Segundo     2 Segundo
3 Primero     3 Segundo     3 Primero
3 Segundo     3 Primero     3 Segundo
4 Primero     4 Segundo     4 Segundo
4 Segundo     4 Primero     4 Primero
5 Primero     5 Segundo     5 Segundo
5 Segundo     6 Segundo     6 Segundo
6 Primero     7 Segundo     5 Primero
6 Segundo     5 Primero     6 Primero
7 Primero     6 Primero     7 Segundo
7 Segundo     7 Primero     8 Segundo
8 Primero     8 Primero     7 Primero
8 Segundo     8 Segundo     9 Segundo
9 Primero     9 Primero     Acabe! Segundo
9 Segundo     9 Segundo     8 Primero
Acabe! Primero Acabe! Segundo 9 Primero
Acabe! Segundo Acabe! Primero Acabe! Primero
    
```

Programación Orientada a Objetos



Creación de Hilos – Segunda Posibilidad

- Crear una clase que implemente la interfaz *java.lang.Runnable* y crear una instancia usando:
 - Thread(Runnable)
 - Thread(Runnable, String nombre)
- Hay que definir el método run en la clase que implemente la interfaz Runnable
- Útil en los casos que la clase forme parte de una jerarquía de clases

```

public class Hilo2 implements Runnable{
    public void run(){
        while(true){ S.O.P. ("Hola mundo!");
        }
    }
    Thread hr = new Thread(new Hilo2());
    hr.start();
}
    
```

Programación Orientada a Objetos



Hilos – Otros métodos

- `isAlive()`: Devuelve *true* si el hilo al que llama el método está vivo (*false* si se ha terminado de ejecutar)
 - `public final boolean isAlive()`
- `isInterrupted()`: Devuelve *true* si el hilo fue interrumpido o *false* en caso contrario
 - `public boolean isInterrupted()`
- `yield()`: Pausa temporalmente el thread que se está ejecutando para que se ejecute otro/s con la misma prioridad
 - `public static void yield()`
- `join()`: Espera a que muera un determinado hilo. Lanza una excepción si se interrumpe el hilo actual
 - `public final void join() throws InterruptedException{}`

Programación Orientada a Objetos



Ejercicio 1 – Hilos – Clase NuevoHilo

- ¿Qué salida produce el siguiente código?

```

public class NuevoHilo implements Runnable {
    String nombre;
    Thread t;
    public NuevoHilo(String n) {
        nombre = n;
        t = new Thread(this, n);
        S.O.P("Nuevo hilo: " + t);
        t.start();
    }
    public void run() {
        try {
            for (int i = 5; i > 0; i--) {
                S.O.P.(nombre + ":" + i);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
            S.O.P("interrupción en: " + nombre);
        }
    }
}
    
```



Programación Orientada a Objetos



Ejercicio 1 – Hilos - Principal

```
NuevoHilo ob1=new NuevoHilo("Uno");
NuevoHilo ob2=new NuevoHilo("Dos");
NuevoHilo ob3=new NuevoHilo("Tres");
System.out.println("¿El hilo Uno esta vivo?: "+ob1.t.isAlive());
System.out.println("¿El hilo Dos esta vivo?: "+ob2.t.isAlive());
System.out.println("¿El hilo Tres esta vivo?: "+ob3.t.isAlive());
try {
    System.out.println("esperando a que terminen los otros");
    ob1.t.join();    ob2.t.join();    ob3.t.join();
} catch (Exception e) {
    System.out.println("Interrupcion hilo principal");}
System.out.println("¿El hilo Uno esta vivo?: "+ob1.t.isAlive());
System.out.println("¿El hilo Dos esta vivo?: "+ob2.t.isAlive());
System.out.println("¿El hilo Tres esta vivo?: "+ob3.t.isAlive());
System.out.println("Sale del principal");
```



Programación Orientada a Objetos



Hilos "demonio" - *Daemon Threads*

- Normalmente son servicios que se ejecutan normalmente, con prioridad baja
- Ejemplo:
 - Recolector de basura (*garbage collector*) de la máquina virtual de Java que comprueba los recursos que no se utilizan y los libera
- `setDaemon()` → Establece si un hilo es "demonio":
 - `setDaemon(true)` → "*Daemon Thread*"
 - `setDaemon(false)` → Hilo de usuario (hacer antes de `start()`)
- `isDaemon()` → Devuelve *true* si el hilo es un "*daemon threads*"



Programación Orientada a Objetos



Ejercicio 2 – Hilos "demonio"

- Realizar la clase `HiloDemonio` que extiende de la clase `Thread` y debe ser un hilo de tipo "demonio".
- Al ejecutar el método `run` de este hilo, deberá aparecer si el hilo es de tipo demonio o si por el contrario es un hilo de usuario
- El programa principal contendrá el siguiente código:

```
Thread hilo = new MiHiloDemonio();
hilo.start();
```



Programación Orientada a Objetos



Prioridades de los hilos – Aspectos Generales

- Cada hilo tiene asociada una prioridad representada por un número entero (1 - 10, la prioridad por defecto es 5)
- La prioridad determina cuando se debe ejecutar un hilo frente a otro, NO la velocidad de ejecución
- Cuando un hilo deja de ejecutarse, se produce un **cambio de contexto**. Del resto de los hilos se ejecuta:
 - El que tiene prioridad más alta
 - Si todos tienen la misma prioridad se ejecutan alternativamente
- La alternancia entre *threads* se puede producir:
 - Según espacios de tiempo asignados por el SO
 - Alternancia entre bloques de sentencias ("*threads egoístas*" cuando por ejemplo aparecen bucles)



Programación Orientada a Objetos



Prioridades de los hilos - Métodos

- Asignar prioridades a hilos:

```
final void setPriority(int nivel){}
```

- Devolver la prioridad de un hilo:

```
final int getPriority(){}
```

- Ejemplos:

- Hilo.setPriority(2);
- Hilo.setPriority(Thread.MAX_PRIORITY);
- Hilo.setPriority(Thread.MIN_PRIORITY);
- Hilo.setPriority(Thread.NORM_PRIORITY);
- Hilo.getPriority();



Concurrencia

- Situaciones en las que varios hilos se ejecutan simultáneamente: comparten datos y deben considerarse mutuamente
- Estos escenarios se conocen como "productor/consumidor": el productor genera datos que recoge el consumidor
- Ejemplos:
 - Introducir caracteres por teclado. El hilo productor coloca los eventos de teclado en una cola de eventos y el hilo consumidor lee los eventos de la misma cola
 - Productor escribe datos en un fichero y consumidor lee datos del mismo fichero



Concurrencia - Ejemplo

```
public class Almacen {  
    private int contenidos;  
    public int getContenidos() {  
        return contenidos;  
    }  
    public void setContenidos(int valor) {  
        contenidos = valor;  
    }  
}
```



Concurrencia - Ejemplo

```
public class Productor extends Thread {  
    private Almacen almacen;  
    private int num;  
    public Productor (Almacen c, int n) { almacen = c; num = n;}  
    public void run() {  
        for (int i = 0; i < 100; i++) {  
            almacen.setContenidos(i);  
            S.O.P.("Productor #" + this.num + " pone: " + i);  
            try {  
                // Parada de 4 segundos → sleep(4000);  
                sleep((int)(Math.random() * 100));  
            } catch (InterruptedException e) { }  
        }  
    }  
}
```



Concurrencia - Ejemplo

```
public class Consumidor extends Thread {
    private Almacen almacen;
    private int num;
    public Consumidor(Almacen c, int n) { almacen=c; num=n;}
    public void run() {
        int valor = 0;
        for (int i = 0; i < 100; i++) {
            valor = almacen.get();
            S.O.P.("Consumidor #" + this.num + " consume: " + valor);
        }
    }
}
```



Concurrencia - Ejemplo

```
public class ProducerConsumerTest {
    public static void main(String[] args) {
        CubbyHole c = new CubbyHole();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);

        p1.start();
        c1.start();
    }
}
```



Concurrencia - Ejemplo

Se producen competiciones:

```
Productor #1 pone: 0
Consumidor #1 consume: 0
Consumidor #1 consume: 0
Consumidor #1 consume: 0
Consumidor #1 consume: 0
Productor #1 pone: 1
Consumidor #1 consume: 1
Consumidor #1 consume: 1
Productor #1 pone: 2
Productor #1 pone: 3
...
```



Bloqueos en objetos

- Las partes de un programa que pueden acceder al mismo objeto desde hilos concurrentes se llaman "secciones críticas"
- Las secciones críticas pueden ser:
 - un método (método "synchronized")
 - un bloque de código:

```
synchronized (<objeto>) { <sección-crítica> }
```
- No se pueden ejecutar simultáneamente dos secciones críticas de un mismo objeto
- Si un método *f* es "synchronized" en un objeto, ningún otro método sincronizado en el mismo objeto se puede llamar hasta que *f* termine



Bloqueos en objetos – Ejemplo 1

- Clase Object:
 - Métodos wait(), wait(long), notify(), notifyAll() llamados desde métodos "synchronized"
 - wait() pasa el hilo actual a estado "Not Runnable"
 - notify(), notifyAll() pasan a estado "Runnable" los hilos que pasaron a estado "Not Runnable" al ejecutar "wait()"



Bloqueos en objetos – Ejemplo 2

- Clase java.util.Vector:
 - public final synchronized void addElement(Object o)
 - public final synchronized boolean removeElement(Object o)

```
public final synchronized void addElement(Object obj) {  
    ensureCapacity(elementCount + 1);  
    elementData[elementCount++] = obj;  
}
```



Bloqueos en objetos – Ejemplo 3

- Clase Almacen del ejemplo anterior:

```
public class Almacen {  
    private int contenidos;  
    public synchronized int getContenidos() { return contenidos; }  
    public synchronized void setContenidos(int valor) {  
        contenidos = valor;  
    }  
}
```
- Obs: Se observarían diferencias en la salida del programa si, p.e. el método setContenidos tuviera muchas instrucciones



Notificaciones entre hilos

- Un hilo se convierte en monitor de un objeto cuando está ejecutando una sección crítica del objeto
- Un hilo que es monitor de un objeto se puede detener (estado "not runnable") y continuar más tarde:
 - public final void wait() throws InterruptedException
 - public final void wait(long t) throws InterruptedException → Indicando tiempo máximo de espera
- Un hilo que es monitor de un objeto puede despertar a otro/s hilo/s mediante:
 - public final void notify() → Se elige aleatoriamente uno de los hilos que están esperando
 - public final void notifyAll() → Se despierta a todos, pero en el orden elegido por el S.O.

- Métodos definidos en la clase Object

- Hay que evitar situaciones como *deadlocks*



Notificaciones entre hilos - Ejemplo

```
public class Almacen {
    private int contenidos;
    private boolean disponible = false;

    public synchronized int get() {
        while (disponible == false) {
            try { wait(); } catch (InterruptedException e) { }
        }
        disponible = false;
        notifyAll();
        return contenidos;
    }
}
```



Programación Orientada a Objetos



Notificaciones entre hilos - Ejemplo

```
...
public synchronized void put(int valor) {
    while (disponible == true) {
        try { wait(); }
        catch (InterruptedException e) { }
    }
    contenidos = valor;
    disponible = true;
    notifyAll();
}
}
```

Productor #1 pone: 0
Consumidor #1 consume: 0
Productor #1 pone: 1
Consumidor #1 consume: 1
Productor #1 pone: 2
Consumidor #1 consume: 2
Productor #1 pone: 3
Consumidor #1 consume: 3
Productor #1 pone: 4
Consumidor #1 consume: 4
Productor #1 pone: 5
Consumidor #1 consume: 5
Productor #1 pone: 6
Consumidor #1 consume: 6
Productor #1 pone: 7
Consumidor #1 consume: 7
Productor #1 pone: 8
Consumidor #1 consume: 8



Programación Orientada a Objetos



Ejercicio 3 - Hilos

- Realizar el código de la clase cambiaImagen
 - Esta clase contiene un array de nombres con ficheros de distintas imágenes
 - El método run de esta clase debe ir seleccionando la siguiente imagen cada 5 segundos mostrando por pantalla el nombre de la imagen actual
 - Antes de seleccionar la imagen se deberán cargar los nombres de las imágenes que son imagen1.jpg, ..., imagen10.jpg



Programación Orientada a Objetos

