



# Concurrencia

# Threads

# Índice

---

## Contenido

Introducción.....	2
Creación de <i>threads</i> .....	3
Creación de un hilo a partir de <i>Thread</i> .....	3
Creación de un hilo implementando la interfaz <i>Runnable</i> .....	3
Ejecución de varios hilos de forma simultánea.....	5
Ciclo de vida de un <i>thread</i> .....	8
Ejecución de un nuevo <i>thread</i> .....	8
Detención temporal de un <i>thread</i> .....	9
Finalización de un <i>thread</i> .....	9
Sincronización de hilos.....	12
Regiones críticas.....	13
Productores y consumidores.....	16
Buffer de mensajes.....	20
Semáforos.....	25
Semáforos binarios.....	26
Interbloqueos.....	27
Prioridad de hilos.....	32
Grupos de <i>threads</i> .....	33

## Introducción

Los ordenadores y sistemas operativos modernos permiten la ejecución de varios programas simultáneamente. Es lo que se denomina multitarea. Realmente un ordenador con un solo procesador no puede ejecutar varias tareas a la vez, sin embargo los sistemas operativos actuales son capaces de ejecutar varios programas simultáneamente aunque sólo se disponga de una CPU. Esto lo consiguen repartiendo el tiempo de CPU entre las distintas tareas y aprovechando las pausas que se producen en unas para permitir que prosiga la ejecución de las otras. En ordenadores con dos o más procesadores la multitarea es real ya que cada uno puede ejecutar una tarea distinta aunque está limitada al número de procesadores disponibles.

Un **proceso** es un programa ejecutándose de manera independiente y con un espacio propio de memoria. Un sistema operativo multitarea es capaz de ejecutar varios procesos de forma simultánea. Los recursos que asigna el sistema operativo a un proceso son independientes de los recursos que use cualquier otro proceso.

Un **hilo** o **thread** es un flujo secuencial simple dentro de un proceso. Un proceso puede tener varios hilos ejecutándose también simultáneamente. Esos hilos sí comparten los recursos que proporcionó el sistema operativo al proceso que los ha creado, así, mientras los procesos son independientes unos de otros, los hilos sí pueden interactuar entre ellos.

Gracias a los sistemas multitarea los procesadores pueden aprovecharse mejor y así mejorar el rendimiento del sistema. En un sistema que no fuera multitarea, cuando un programa ejecutara una operación "lenta" como por ejemplo una lectura de disco, el procesador estaría detenido a la espera de la respuesta de esa operación de entrada-salida y por lo tanto se estaría desaprovechando el uso del procesador. En el caso de un sistema multitarea cuando un proceso realiza una operación que requiere la espera del procesador, el sistema operativo se encarga de permitir que otro proceso se ejecute para aprovechar el tiempo que estaría la CPU detenida.

Los hilos pueden ser de dos tipos, normales y *daemon*. Los hilos *daemon* son los que realizan acciones en segundo plano. Un ejemplo de hilo *daemon* sería el *garbage collector*. Un proceso termina su ejecución cuando finalizan todos los hilos que no son *daemon*.

## Creación de *threads*

Los hilos en Java son objetos descendientes de la clase `Thread` (j ava. l ang. `Thread`). Hay dos formas distintas para crear *threads*. La primera es heredar directamente de la clase `Thread` y sobrecargar el método `run()` y la segunda es crear un *thread* a partir de un objeto de una clase que implemente la interfaz `Runnable` (j ava. l ang. `Runnable`).

### Creación de un hilo a partir de *Thread*

La forma más sencilla de crear un hilo es extendiendo de la clase `Thread`.

```
/**
 * Ejemplo para crear un hilo extendiendo de la clase Thread.
 * @author Jesús Frontelo
 */
public class ExtenderThread extends Thread {
    private int repeticiones = 10;

    /**
     * Constructor de la clase.
     * @param nombre nombre del hilo.
     */
    ExtenderThread(String nombre) {
        super(nombre);
    }

    /**
     * Indica las acciones que ejecuta el Thread.
     */
    public void run() {
        for(int i=0; i<repeticiones; i++) {
            System.out.println("El nombre del Thread es " + this.getName());
        }
    }
}
```

En el constructor se utiliza un `String` para darle un nombre al *thread* creado y mediante la llamada a `super()` se llama al constructor de la super-clase `Thread`. También se redefine el método `run()` de la clase `Thread` para que realice alguna acción.

Para ejecutar un hilo creado de esta forma se debe crear un objeto de esta clase y después realizar una llamada al método `start()` definido en la clase `Thread` que se encarga de llamar al método `run()` y poner en marcha el hilo.

```
ExtenderThread hilo = new ExtenderThread("Hilo de prueba extendiendo de Thread");
hilo.start();
```

### Creación de un hilo implementando la interfaz *Runnable*

La definición de la interfaz `Runnable` es la siguiente:

```
public interface Runnable {
    public void run();
}
```

La interfaz declara un método que se corresponde con el método `run()` de `Thread`.

Esta segunda forma también requiere que se implemente el método `run()`, pero además es necesario crear un objeto de la clase `Thread` para lanzar la ejecución del nuevo hilo. La clase `Thread` tiene un constructor que recibe como parámetro un objeto que implementa la interfaz `Runnable` y permite que ese objeto se convierta en un hilo. Después, cuando invoquemos al método `start()` del *thread*, éste se encargará de realizar la llamada al método `run()` del objeto que implementa `Runnable`.

```
/**
 * Ejemplo para crear un hilo implementando la interfaz Runnable.
 * @author Jesús Frontelo
 */
public class ImplementarRunnable implements Runnable {
    private String nombreThread;
    private int repeticiones = 10;

    /**
     * Constructor de la clase.
     * @param nombre nombre del hilo.
     */
    public ImplementarRunnable(String nombre) {
        nombreThread = nombre;
    }

    /**
     * Indica las acciones que ejecuta el hilo.
     * Método declarado en la interfaz Runnable.
     */
    public void run() {
        for(int i=0; i<repeticiones; i++) {
            System.out.println("El nombre del Thread es " + nombreThread);
        }
    }
}
```

El siguiente código crea un nuevo *thread* y lo ejecuta por este segundo procedimiento:

```
ImplementarRunnable runnable = new ImplementarRunnable("Hilo de prueba Runnable");
Thread hilo = new Thread(runnable);
hilo.start();
```

Este método se utiliza normalmente cuando estamos obligados a extender de una clase y por lo tanto no podemos extender de `Thread`.

## Ejecución de varios hilos de forma simultánea

El siguiente ejemplo muestra la ejecución de varios hilos de forma simultánea:

```
/**
 * Programa que realiza la primera prueba de ejecución de hilos.
 * @author Jesús Frontelo
 */
public class PruebaThreads1 {
    /**
     * Método principal de la aplicación.
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Creación de varios hilos extendiendo Thread o implementando Runnable
        ExtenderThread t1 = new ExtenderThread("Hilo 1");
        ExtenderThread t2 = new ExtenderThread("Hilo 2");

        ImplementarRunnable r1 = new ImplementarRunnable("Hilo 3");
        Thread t3 = new Thread(r1);
        Thread t4 = new Thread(new ImplementarRunnable("Hilo 4"));

        // Se inician todos de forma simultanea
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

La salida del programa será algo parecido a esto:

```
Thread Hilo 1, pasada 0
Thread Hilo 1, pasada 1
Thread Hilo 1, pasada 2
Thread Hilo 1, pasada 3
Thread Hilo 1, pasada 4
Thread Hilo 2, pasada 0
Thread Hilo 2, pasada 1
Thread Hilo 2, pasada 2
Thread Hilo 2, pasada 3
Thread Hilo 2, pasada 4
Thread Hilo 3, pasada 0
Thread Hilo 3, pasada 1
Thread Hilo 3, pasada 2
Thread Hilo 3, pasada 3
Thread Hilo 3, pasada 4
Thread Hilo 4, pasada 0
Thread Hilo 4, pasada 1
Thread Hilo 4, pasada 2
Thread Hilo 4, pasada 3
Thread Hilo 4, pasada 4
```

Viendo el resultado obtenido podemos pensar que un programa secuencial habría hecho lo mismo. En este caso es así, más o menos. Al invocar al método `start()` los hilos intentan ejecutarse a la vez. Como sólo uno puede entrar a ejecutarse, los otros quedan esperando a que les toque su turno. Debemos saber que cuando entra un hilo a ejecutarse, no sale a no ser que se bloquee por alguno de los siguientes motivos:

- Se ejecute una llamada al método `sleep()` de `Thread`.
- Se ejecute una llamada al método `wait()` de `Object`.
- Se realice una operación de entrada-salida.
- Se realice una llamada a un método `synchronized` de un objeto bloqueado.
- Intente ejecutarse un hilo con mayor prioridad.