DAM - Programación de servicios y procesos

Tema 3 - Programación multihilo

Roberto Sanz Requena
rsanz@florida-uni.es

Florida
Grup Educatiu

Índice

1. Introducción
2. Estados de un hilo
3. Clases Java para programación multihilo
4. Gestión de prioridades
5. Sincronización y comunicación entre hilos
6. Problema clásico: productor – consumidor
7. Otras clases de interés

1. Introducción

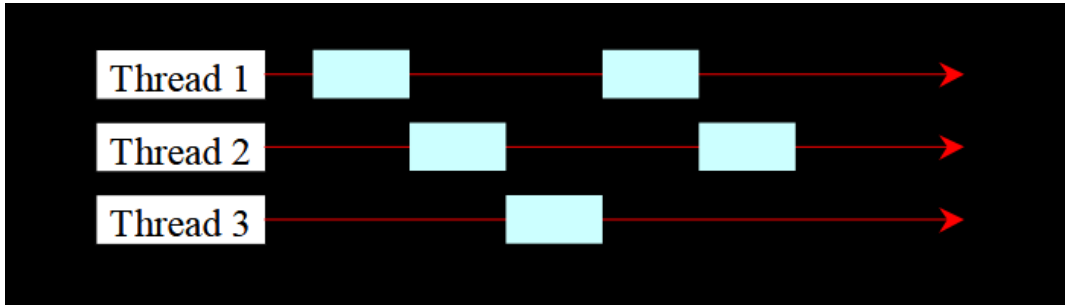
- Un **hilo** es una tarea que puede ser ejecutada al mismo tiempo que otra dentro de un mismo proceso.
- Todos los hilos de un mismo proceso **comparten recursos**: memoria, archivos abiertos, autenticación, …
  - ➢ Esto simplifica y complica las cosas al mismo tiempo.

- Hardware
  - Antiguamente… Sistemas de 1 procesador y 1 núcleo.
  - Sistemas **multinúcleo** → Aplicaciones **multihilo**.
  - Más núcleos + mayores frecuencias de reloj = más potencia.
    - ➢ Hay que saber aprovecharla.
  - Núcleos con HT (HyperThreading) → Simulación de tener más núcleos.

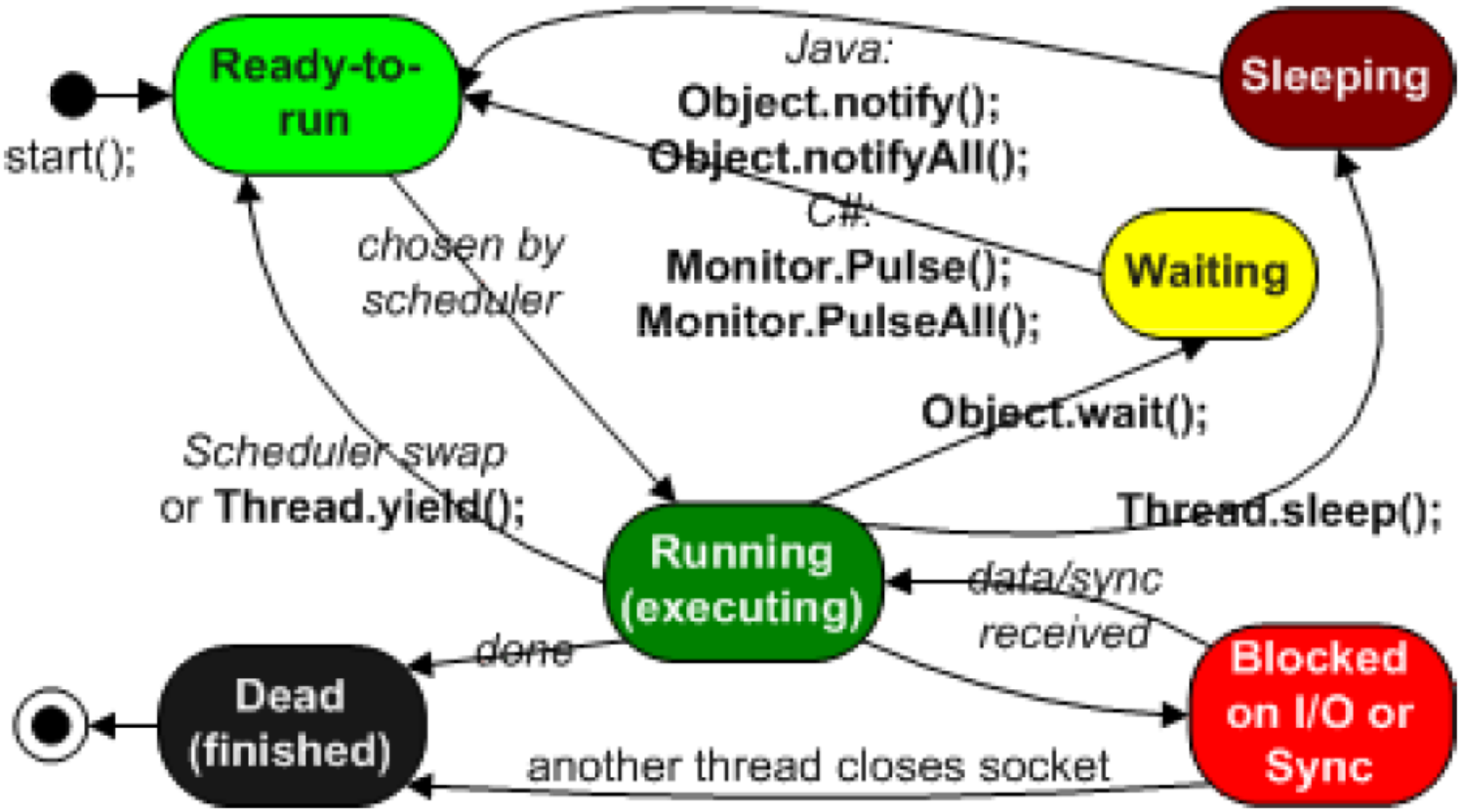
1. Introducción

Threads múltiples en múltiples CPUs

Threads múltiples compartiendo una CPU



2. Estados de un hilo



3. Clases Java para programación multihilo

Clase `Thread`

https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

Método	Descripción
`Thread()`	Constructor.
`currentThread()`	Devuelve el objeto thread que representa al hilo de ejecución actual.
`yield()`	Cambio de contexto entre el hilo actual y el siguiente hilo ejecutable disponible.
`sleep(long)`	Pausa de `<long>` milisegundos.
`start()`	Insta al intérprete Java a crear un contexto del hilo y comenzar a ejecutarlo.
`run()`	Ejecución del hilo, llamado por el método `start()`.
`interrupt()`	Interrumpe el hilo.
`setPriority(int)`	Establece la prioridad de un hilo.
`getPriority()`	Devuelve la prioridad de un hilo.
`setName(String)`	Asigna a un hilo un nombre determinado.
`getName()`	Devuelve el nombre de un hilo.
`join()`	Espera a que el hilo finalice y devuelve el control al hilo principal.

3. Clases Java para programación multihilo

Creación de hilos:

- Heredar la clase `Thread`.
- Implementar la interfaz `Runnable`. ← RECOMENDACIÓN JAVA

```
class EjecutorTareaCompleja implements Runnable{

	private String nombre;
	int numEjecucion;

	public EjecutorTareaCompleja(String nombre){
		this.nombre=nombre;
	}

	@Override
	public void run() {
		String cad;
		while (numEjecucion < 100){
			for (double i=0; i<499.9; i=i+0.02) {
				Math.sqrt(i);
			}
			cad = "Soy el hilo "+this.nombre;
			cad += ", i= "+numEjecucion;
			System.out.println(cad);
			numEjecucion++;
		}
	}
}
```

```
public class LanzaHilos {
	/**
	* @param args
	*/
	public static void main(String[] args) {
		int NUM_HILOS=500;
		EjecutorTareaCompleja op;
		for (int i=0; i<NUM_HILOS; i++) {
			op=new EjecutorTareaCompleja("Op. "+i);
			Thread hilo=new Thread(op);
			hilo.start();
		}
	}
}
```

Ejemplo “PSP_Multihilo_1”

4. Gestión de prioridades

Prioridades:

- Determinar qué hilo recibe el control de la CPU.
- En Java se definen con un entero entre 1 y 10.
- Java define las constantes `MAX_PRIORITY` (10), `NORM_PRIORITY` (5, prioridad por defecto) y `MIN_PRIORITY` (1).
- Para establecer manualmente una prioridad `setPriority(valor)`.
- Práctica habitual: no modificar prioridades y dejar que las gestione el sistema operativo → riesgo de inestabilidad del sistema.

5. Sincronización y comunicación entre hilos

Sincronización

- **Condición de carrera** (race condition): varios hilos acceden al mismo objeto (recurso compartido) y lo modifican sin control, produciendo resultados inesperados.

- Cuando un método acceda a una variable miembro que esté compartida deberemos proteger dicha sección crítica, usando `synchronized`.

- Se puede poner todo el método `synchronized` o marcar un trozo de código más pequeño.

- Un hilo se sincroniza cuando se convierte en propietario del monitor del objeto (bloqueo).

- Aplicación **thread-safe**: aplicación programada teniendo en cuenta que si se ejecutan varios hilos se puede estar seguro que se comportará bien.

5. Sincronización y comunicación entre hilos

Sincronización

Ejemplo PSP_Multihilo_Sync

```java
public class PSP_Multihilo_Sync implements Runnable {

    int entradasDisponibles = 100;
    static int entradasVendidas = 0;

    synchronized public void reservaEntrada (String nombre, int entradas) {
    //public void reservaEntrada (String nombre, int entradas) {
        if (entradas <= entradasDisponibles) {
        System.out.println (entradas + " reservadas para " + nombre);
        entradasDisponibles = entradasDisponibles - entradas;
        entradasVendidas = entradasVendidas + entradas;
        } else {
        System.out.println ("No quedan entradas");
        }
    }

    public void run () {
        String nombre = Thread.currentThread().getName ();
        int entradas = (int) (Math.random() * 10 + 1);
        reservaEntrada(nombre,entradas);

    }
```

```java
public static void main (String[]args) {
    PSP_Multihilo_Sync s = new PSP_Multihilo_Sync ();
    Thread t;
    for (int i = 0; i < 100; i++) {
    t = new Thread(s);
    t.setName("Cliente " + (i + 1));
    t.start();
    }
    try {
      Thread.sleep(1000);
      } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      }  //Pausa para dejar que acaben todos los threads
    System.out.println("Total entradas vendidas: " +
entradasVendidas);
    }
}
```

5. Sincronización y comunicación entre hilos

Comunicación entre hilos

- Métodos `wait()`, `notify()` y `notifyAll()` de la clase `Object`.
- Sólo pueden ejecutarse dentro de una sección de código `synchronized`.
- `wait()`: provoca que un hilo libere el bloqueo sobre el objeto, entrando en espera hasta que…
  - otro hilo entre en el mismo monitor y llame a `notify()`,
  - otro hilo entre en el mismo monitor y llame a `notifyAll()`,
  - otro hilo interrumpa este hilo,
  - transcurra el tiempo especificado en la llamada a `wait()`.

  Posteriormente el hilo pasa a estar disponible otra vez para el planificador de la CPU.

https://howtodoinjava.com/java/multi-threading/wait-notify-and-notifyall-methods/

5. Sincronización y comunicación entre hilos

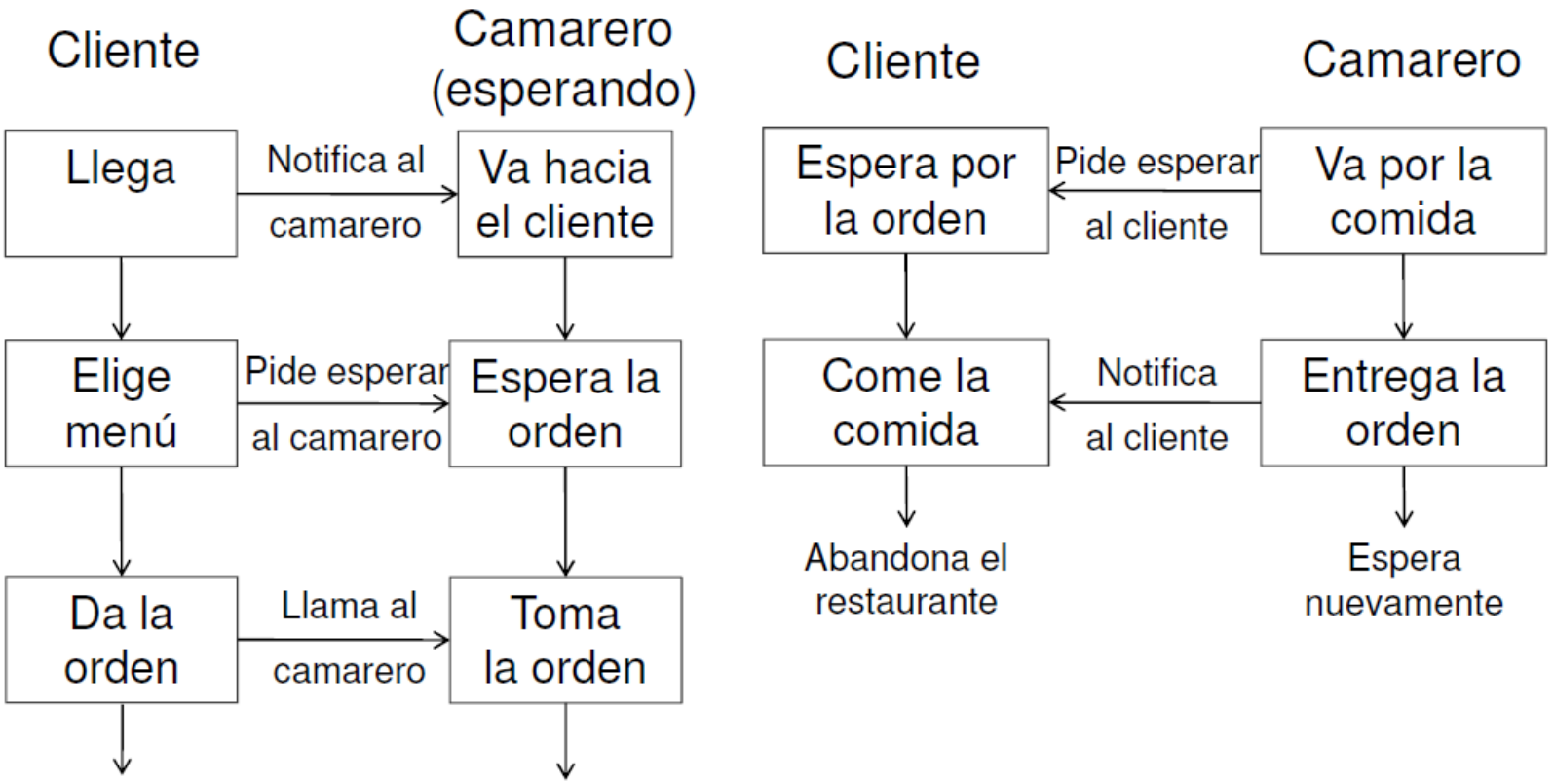
Comunicación entre hilos

- `notify()` : despierta a un hilo que está esperando a este monitor para bloquearlo de nuevo.
  - Si hay varios hilos en espera la selección es arbitraria o se debe implementar por código.
  - El hilo que se ha despertado no podrá continuar la ejecución hasta que el hilo actual abandone el bloqueo.

- `notifyAll()`: despierta a todos los hilos que hayan ejecutado `wait()` sobre este objeto sincronizado.
  - Teóricamente el primer hilo en continuar será el que tenga mayor prioridad asignada (la gestión real la realiza el sistema operativo).

https://howtodoinjava.com/java/multi-threading/wait-notify-and-notifyall-methods/

5. Sincronización y comunicación entre hilos



6. Problema clásico: productor - consumidor

Dos procesos (productor y consumidor) comparten un buffer de tamaño finito:

- Productor: genera un producto, lo almacena en el buffer y comienza de nuevo.
- Consumidor: toma productos de uno en uno.

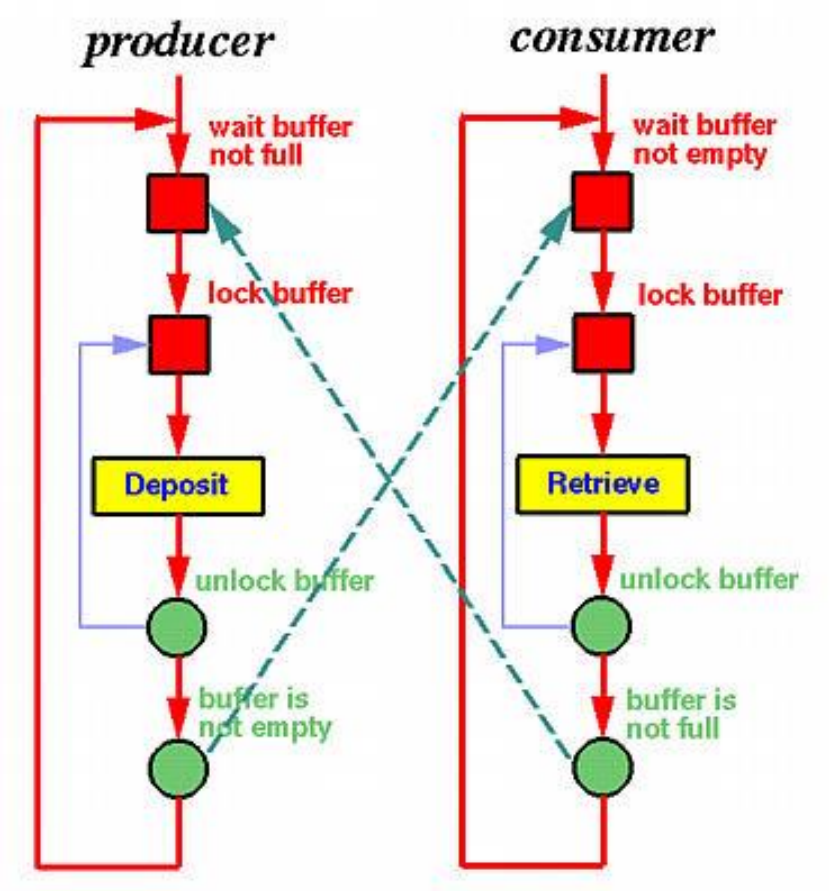
Problema: evitar que el productor añada más productos que la capacidad del buffer y que el consumidor intente tomar un producto si el buffer está vacío.

Solución:

- Idea general: ambos procesos se ejecutan simultáneamente y se “despiertan” o “duermen” según el estado del buffer.
- Productor: agrega productos mientras quede espacio y en el momento en que se llene el buffer se pone a “dormir”, hasta que el consumidor retire un producto y le notifique que puede volver a producir.
- Consumidor: consume productos mientras haya en el buffer y en el momento en que se vacíe se poner a “dormir”, hasta que el productor genere productos de nuevo y le notifique que puede volver a consumir.

https://es.wikipedia.org/wiki/Problema_productor-consumidor

6. Problema clásico: productor - consumidor



https://pages.mtu.edu/~shene/NSF-3/e-Book/SEMA/TM-example-buffer.html

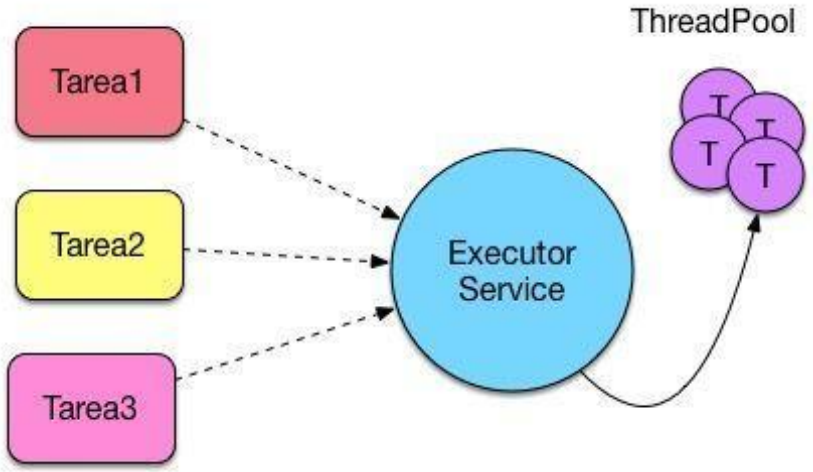
6. Problema clásico: productor - consumidor

Ejemplo PSP_Multihilo_PC

```java
import java.util.LinkedList;
public class ProductorConsumidor {
public static class PC {

// Crea una lista compartida de tamanyo <capacity>
LinkedList<Integer> listaProductos = new LinkedList<>();
int capacidad = 2;

// Funcion del productor
public void produce() throws InterruptedException {
int valor = 0;
while (true) {
synchronized (this) {
// El hilo productor espera mientras la lista este llena
while (listaProductos.size() == capacidad) wait();
// Insertar el producto en la lista
listaProductos.add(valor++);
System.out.println("Productor produjo: item nº " + valor + "
(stock " + listaProductos.size() + " productos)");
// Notifica al consumidor que empiece a consumir
notify();
// Pausa para ver el proceso paso a paso
Thread.sleep(1000);
}
}
}
// Funcion del consumidor
public void consume() throws InterruptedException {
while (true) {
synchronized (this) {
// El hilo consumidor espera mientras la lista este vacia
while (listaProductos.size() == 0) wait();
// Retira el primer producto de la lista
int valor = listaProductos.removeFirst();
System.err.println("Consumidor consumio: item nº " + valor + "
(stock " + listaProductos.size() + " productos)");
// Despierta al hilo productor
notify();
// Pausa para ver el proceso paso a paso
Thread.sleep(1000);
}
}
}
}
```

```java
public static void main(String[] args) throws
InterruptedException {

// Crea la cola y los hilos productor
// y consumidor
final PC pc = new PC();

// Hilo productor
Thread t1 = new Thread(new Runnable() {

@Override
public void run() {
try {
pc.produce();
} catch (InterruptedException e) {
e.printStackTrace();
}
}
});

// Hilo consumidor
Thread t2 = new Thread(new Runnable() {
 @Override
public void run() {
try {
pc.consume();
} catch (InterruptedException e) {
e.printStackTrace();
}
}
});

// Iniciar los hilos
t1.start();
t2.start();

// Finalizar hilos, t1 antes que t2
t1.join();
t2.join();
}

}
```

7. Otras clases de interés



API `ExecutorService`

Simplifica la creación de tareas asíncronas proporcionando un **pool** (conjunto) de hilos y una API para asignarles las tareas.

Creación

```
ExecutorService executor = Executors.newFixedThreadPool(<pool>);
ExecutorService executor = Executors.newSingleThreadExecutor();
ExecutorService executor = new ThreadPoolExecutor(<pool>,<pool>,0L,
    TimeUnit.MILLISECONDS,new LinkedBlockingQueue<Runnable>());
```

Ejecución

```
executor.execute(new Runnable());
```

Apagado

```
executor.shutdown();
```

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html
https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ThreadPoolExecutor.html

7. Otras clases de interés

API `ExecutorService`

Ejemplo: PSP_Multihilo_ExecutorService

```
public class Tarea implements Runnable {

    private String nombre;

    public Tarea(String nombre) {
        this.nombre = nombre;
    }

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(nombre + " -> parte " +
                (i + 1));
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

    }
}
```

```
public class Principal {

    public static void main(String[] args) {

        Thread t1= new Thread(new
           Tarea("tarea1"));
        t1.start();
        Thread t2= new Thread(new
           Tarea("tarea2"));
        t2.start();
        Thread t3= new Thread(new
           Tarea("tarea3"));
        t3.start();

    }

}
```

7. Otras clases de interés

API `ExecutorService`

Ejemplo: PSP_Multihilo_ExecutorService

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Principal_ExecutorService {

    public static void main(String[] args) {
        String[] arrayTareas = {"tarea1", "tarea2", "tarea3"};
        ExecutorService executor = Executors.newFixedThreadPool(10);
        for (String tarea : arrayTareas) {
          executor.execute(() -> ejecutarTarea(tarea));
        }
        executor.shutdown();
    }

    private static void ejecutarTarea(String nombre) {
        for (int i = 0; i < 5; i++) {
            System.out.println(nombre + " -> parte " + (i + 1));
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

Actividad Entregable 3 - Multihilo

Presentación de la Actividad Entregable 3 (AE3_T3_Multihilo)