Universidad El Bosque

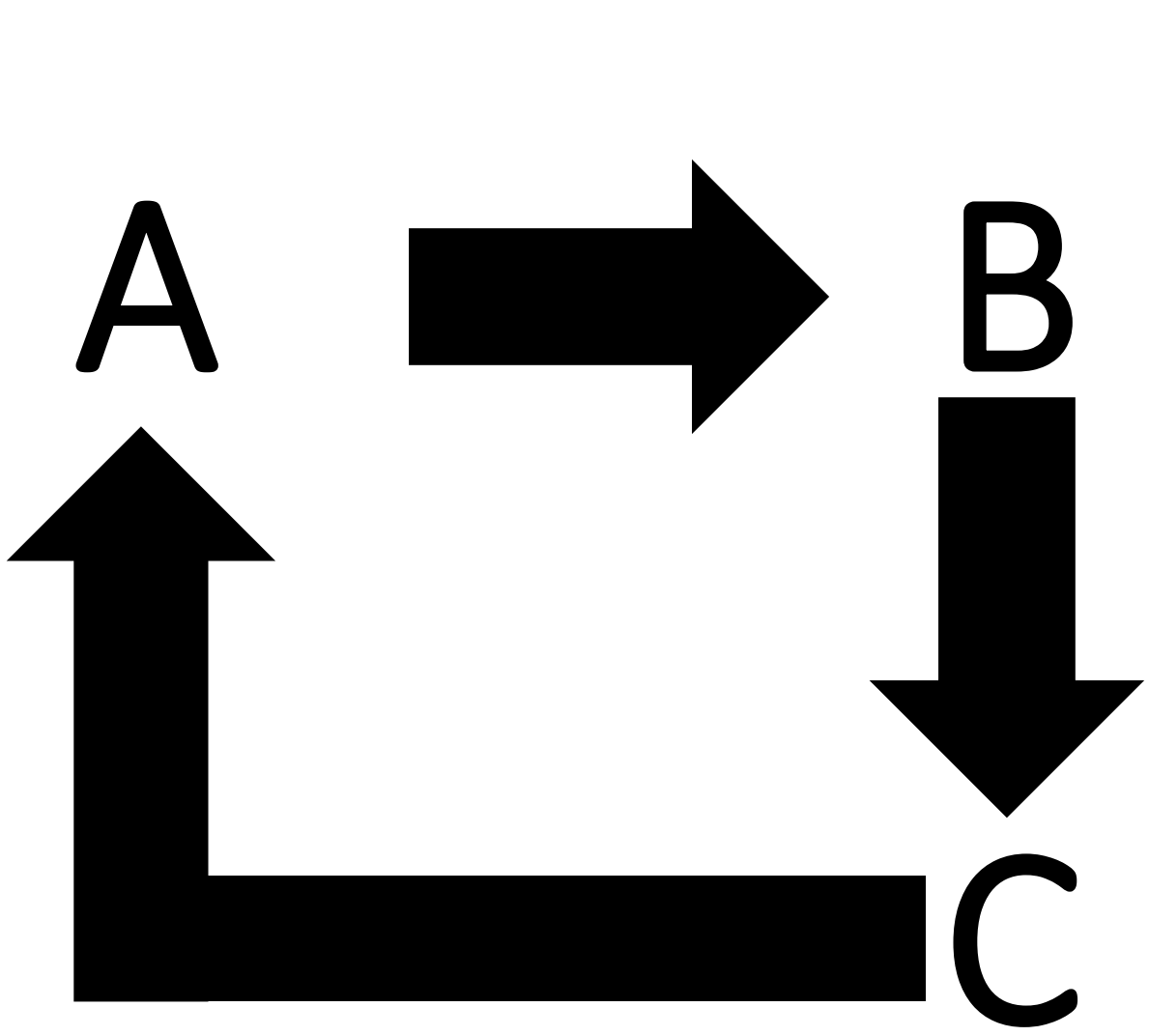
Programa: Ingeniería de Sistemas

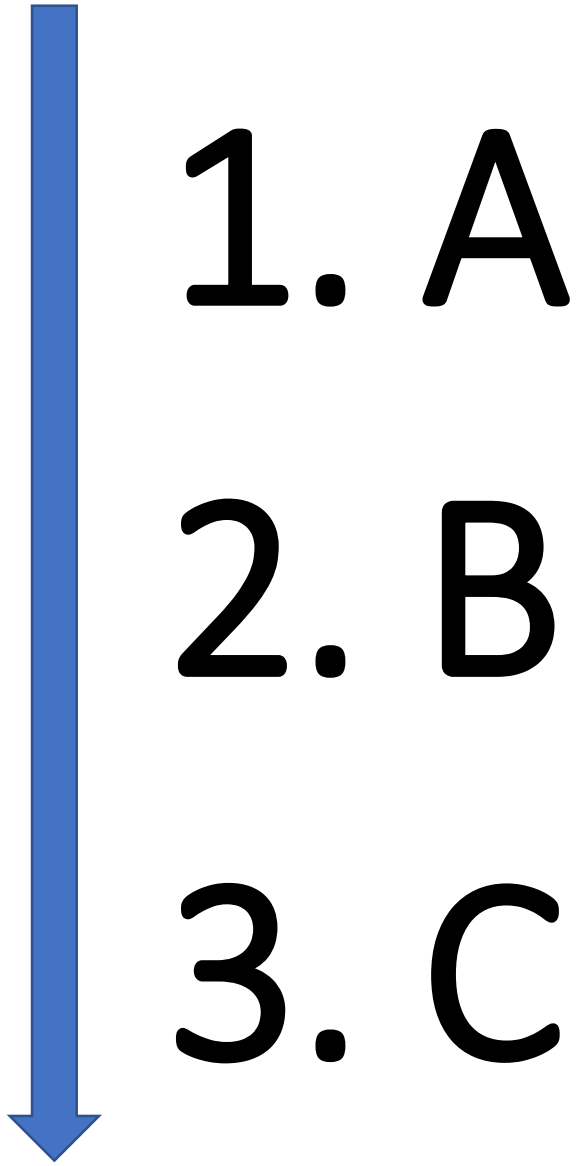
PROGRAMACIÓN CONCURRENTE

PROGRAMACIÓN 2

2020-20

Flujo de control





Flujo de control ⇔ Proceso/Secuencia

¿PROGRAMACIÓN CONCURRENTE?

¿CONCURRENCIA?
¿PARALELISMO?

¿Por qué aprender la concurrencia?

* Se consigue un incremento de rendimiento significativo
* Al día de hoy, los procesadores de computadores o dispositivos móviles (máquinas) tienen un mayor número de núcleos que se pueden aprovechar aún más.
* Permite hacer tareas en segundo plano.
* Java está orientado a la concurrencia.

¿Qué es un HILO?

Es la forma de planificar una serie de tareas. Si solamente tenemos 1 único hilo, se ejecutarían las tareas una tras otra. Es decir, de manera **secuencial**.

Con la programación concurrente se pretende que se ejecuten 2 o más tareas al tiempo.

Dependiendo de los **procesadores lógicos** de procesamiento de máquina, los hilos que se lancen se ejecutarán en el orden que los hilos que se lancen lleguen.

Todos los hilos, cuando terminan su ejecución, se destruyen. De esta forma, dejan la **memoria libre**.

CONCURRENCIA

Procesador

Interpretación de información binaria

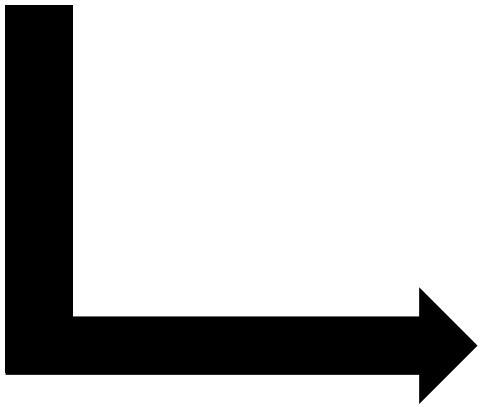
La concurrencia es la capacidad del CPU para procesar más de un proceso al mismo tiempo

La **CPU** se basa en operaciones de programas diseñados especialmente para la **transmisión** y **utilización informática,** cuyos pasos básicos son:

- Recolectar información
- Decodificarla (para interpretación de codificación)
- Ejecutar instrucciones

CONCURRENCIA

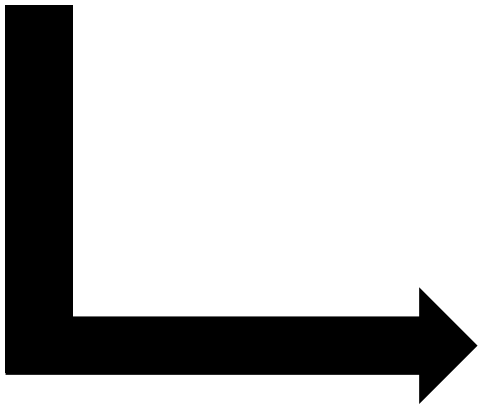
La concurrencia es la capacidad del CPU para procesar más de un proceso al mismo tiempo



Un **procesador** puede procesar al mismo tiempo el mismo número de procesos que el número de CORES que tiene.

CONCURRENCIA

La concurrencia es la capacidad del CPU para procesar más de un proceso al mismo tiempo

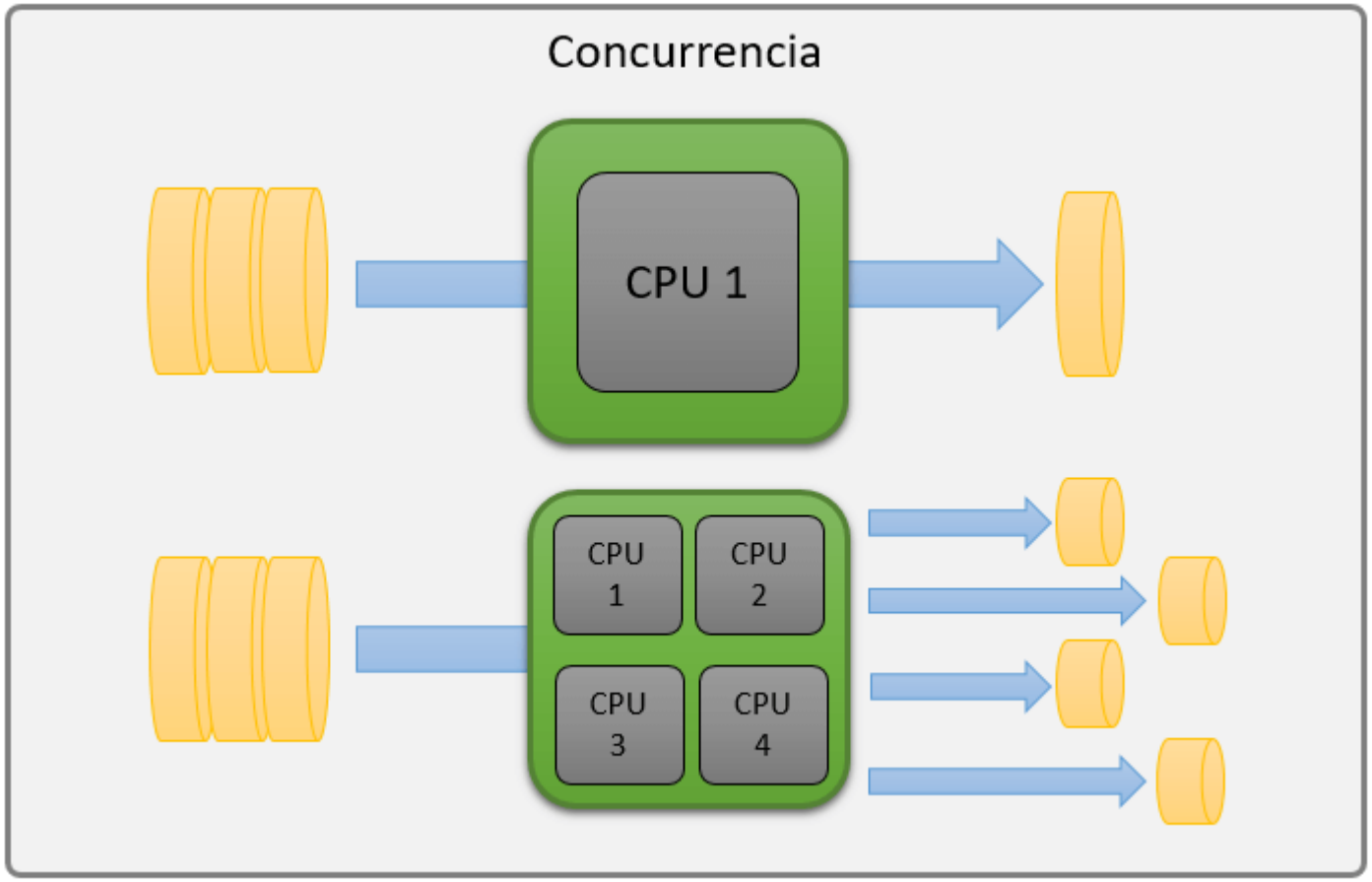


Procesador con 1 CORE ⇔ Sólo podrá ejecutar **un** proceso a la vez

Procesador con 8 CORES ⇔ Podrá ejecutar hasta **8 procesos** a la vez

CONCURRENCIA

La concurrencia es la capacidad del CPU para procesar más de un proceso al mismo tiempo



INDETERMINISMO

Se produce cuando dos o más hilos están escribiendo a la vez en una misma variable compartida. El valor de la variable compartida en este caso es indeterminado. Esto ocurre por cómo el sistema operativo provee responsabilidades/espacio a un hilo o bien puede suspender la ejecución de alguno.

Solución Indeterminismo: **EXCLUSIÓN MUTUA**. Sólo un hilo a la vez pueda acceder a la variable "compartida" (escribir). **¡SINCRONIZACIÓN DE HILOS!**

vector = [1 2 3 4 5 6 7 8]

[…] 2 Hilos […]

vector = [1 2 3 4 5 6 7 8]

vector = [1 2 3 4
4 5 6 7 8]

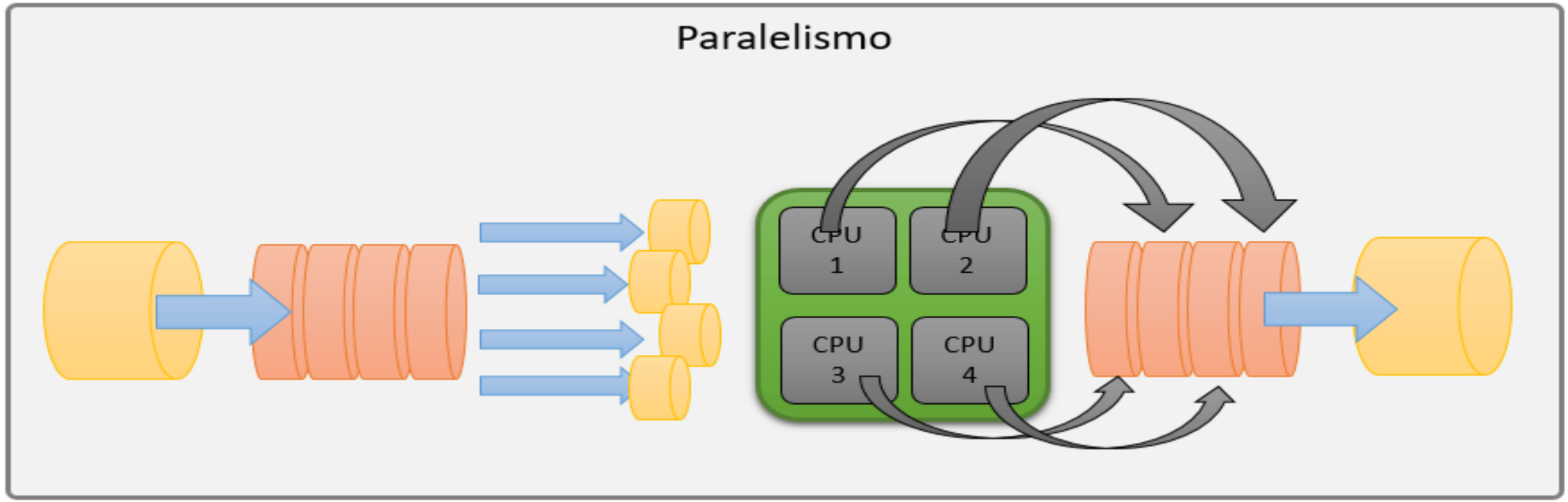
SECCIÓN CRÍTICA → INDETERMINISMO

SECCIÓN CRÍTICA

Es aquella parte del código donde nosotros sabemos que se va producir indeterminismo. Se puede ver que en una variable compartida (**tipo estática**) se está accediendo a ella sin ninguna exclusión mutua. Cada hilo accede a ella y escribe en ella cuando se lo permite el sistema operativo mismo.

PARALELISMO

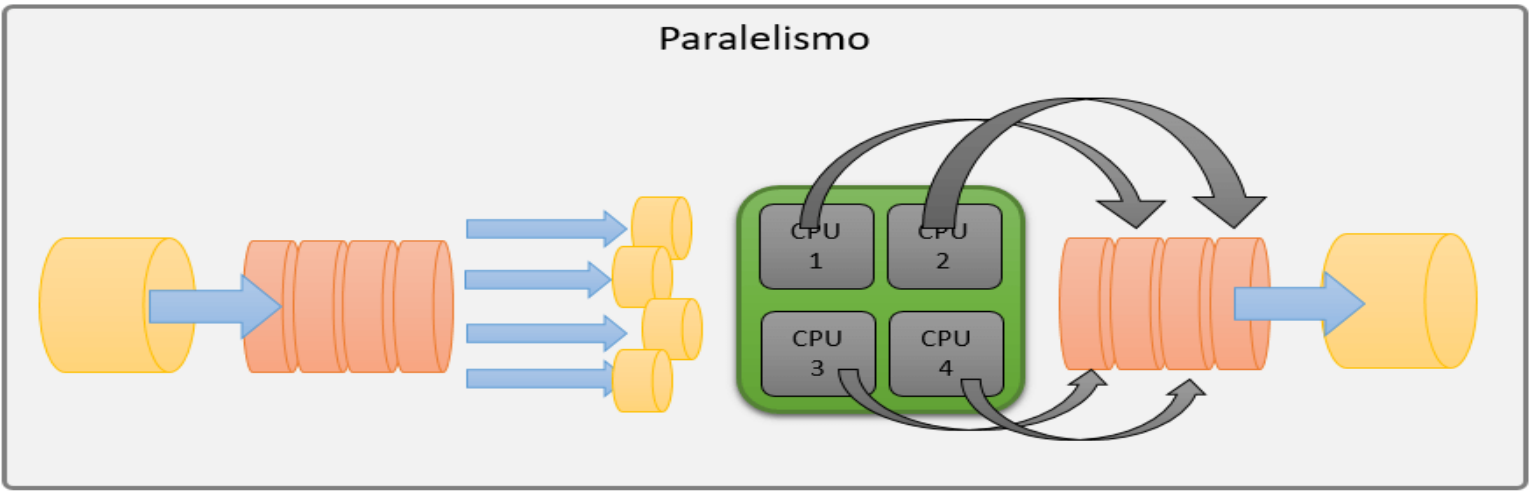
“DIVIDE Y VENCERÁS”



PARALELISMO

Consiste en tomar el problema inicial, **dividirlo** en fracciones más pequeñas y cada una **procesada de forma concurrente**.

Se aprovecha **al máximo** la capacidad del **procesador** para resolver el problema



Debe de haber un paso final que se encargue de unir los resultados de todos los procesos para poder arrojar un resultado final.

CONCURRENCIA vs. PARALELISMO

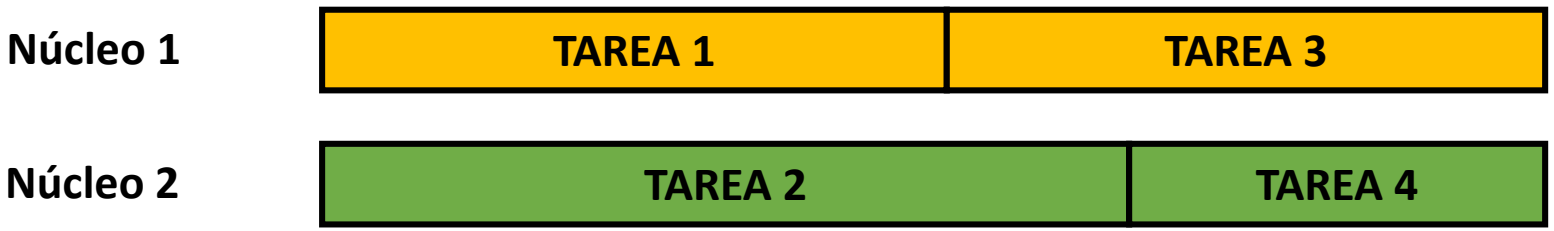
* **PARALELISMO**: Dos o más tareas se están ejecutando al mismo tiempo cada uno en los núcleos disponibles. Se ejecuta una tarea por cada núcleo de la CPU.

* **CONCURRENCIA**: Dos o más tareas NO se están ejecutando al mismo tiempo. Se ejecutan varias tareas por cada núcleo de la CPU. Por cada tarea se asigna un tiempo de CPU. Dos o más tareas comparten el tiempo y tienen como mismo recurso un núcleo para ejecutarse. Se ejecuta un poco de cada tarea, según disponga el sistema operativo para ejecución de tareas.

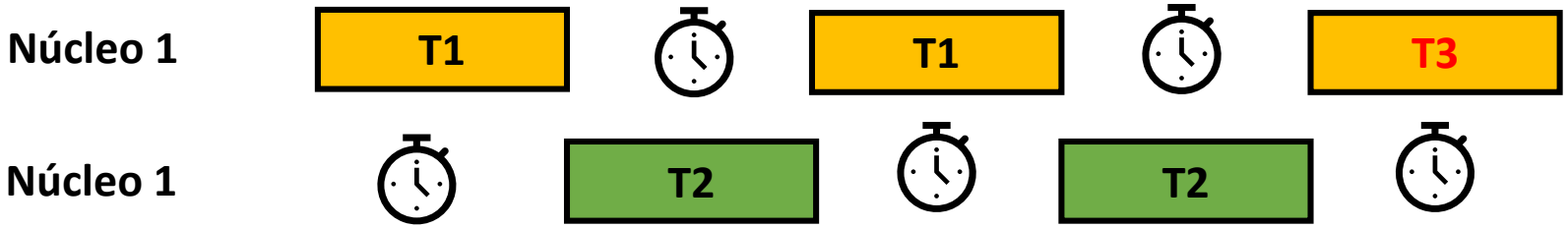
Gráficamente ....

Núcleos = 2

Tareas = 4



Paralelismo



Concurrencia

DIFERENCIAS:

* En el **paralelismo** se ejecutan dos o más tareas al mismo tiempo. En la **concurrencia** no.

* En el **paralelismo** se ejecuta 1 tarea en un núcleo. En la **concurrencia** varias.

* El paralelismo es mucho más rápido que la concurrencia.

* En la concurrencia para cada tarea se asigna un tiempo de CPU

En ocasiones ….

Núcleos = 2

Tareas = 4

Paralelismo + Concurrencia

- Suele ser lo más normal.
- Es la única manera de ejecutar más hilos que núcleos lógicos.
- El Sistema Operativo se organiza para las necesidades con respecto a los núcleos e hilos (procesadores lógicos) disponibles.

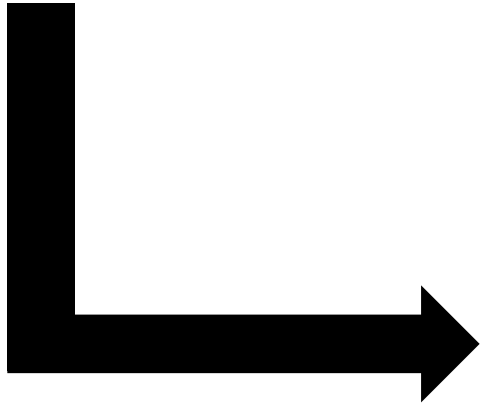
CONCURRENCIA vs. PARALELISMO

Paralelismo → Todos los procesos concurrentes **están íntimamente relacionados a resolver el mismo problema**, de tal forma que el resultado de los demás procesos afecta al resultado final.

Concurrencia → Los procesos en ejecución **NO tienen por qué estar relacionados**. Es decir, cualquiera puede iniciar y terminar en el momento que sea, y el resultado de uno no afecta al otro.

EJEMPLO

Imagina una aplicación de descarga de música, en la cual puedes descargar un número determinado de canciones al mismo tiempo, cada canción es independiente de la otra, por lo que la velocidad y el tiempo que tarde en descargarse cada una no afectara al resto de canciones.



¿De qué tipo de proceso estamos hablando?

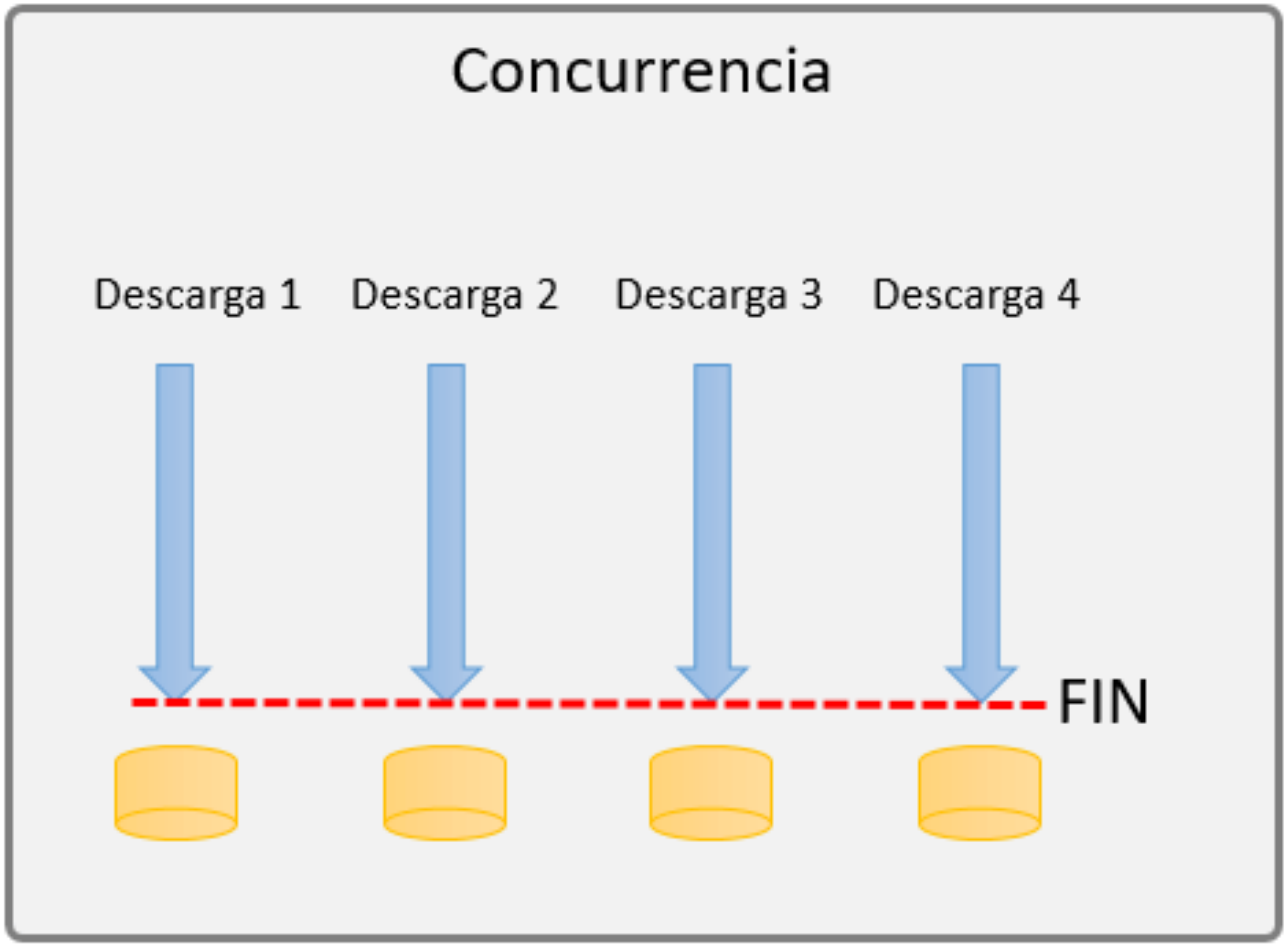
EJEMPLO

Imagina una aplicación de descarga de música, en la cual puedes descargar un número determinado de canciones al mismo tiempo, cada canción es independiente de la otra, por lo que la velocidad y el tiempo que tarde en descargarse cada una no afectara al resto de canciones.

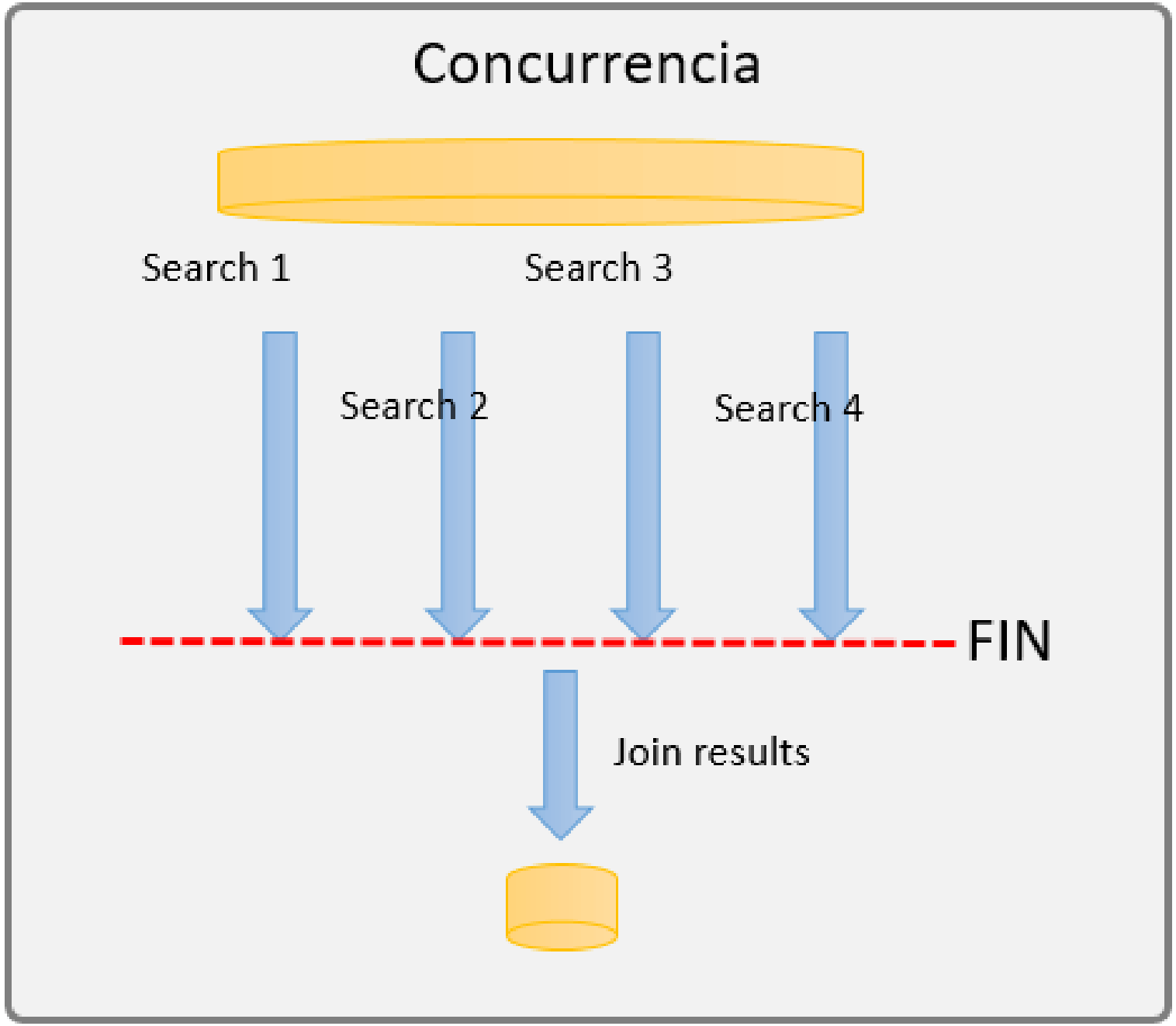
CONCURRENCIA

EJEMPLO

Imagina una aplicación de descarga de música, en la cual puedes descargar un número determinado de canciones al mismo tiempo, cada canción es independiente de la otra, por lo que la velocidad y el tiempo que tarde en descargarse cada una no afectara al resto de canciones.



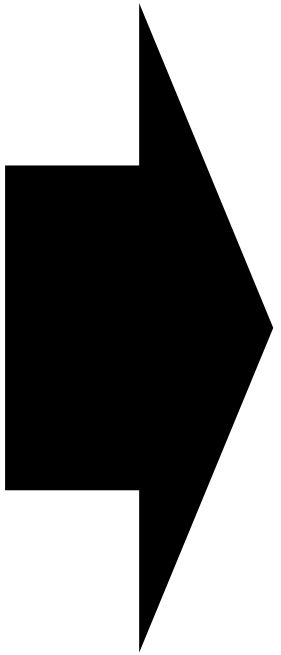
¿EJEMPLO PARALELISMO?



EJEMPLO

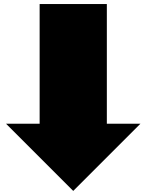
En Java ….. ¿Cómo sería?

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        for(int i=0;i<=5;i++){
            System.out.println("Proceso 1");
        }

        System.out.println("--------");

        for(int i=0;i<=5;i++){
            System.out.println("Proceso 2");
        }
    }
}
```

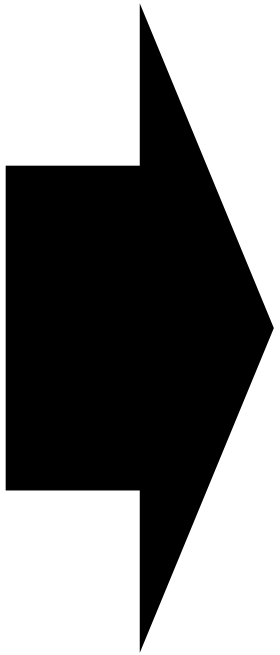


```
Proceso 1
Proceso 1
Proceso 1
Proceso 1
Proceso 1
Proceso 1
--------
Proceso 2
Proceso 2
Proceso 2
Proceso 2
Proceso 2
Proceso 2
```

```java
package clase;

public class Proceso {

}
```



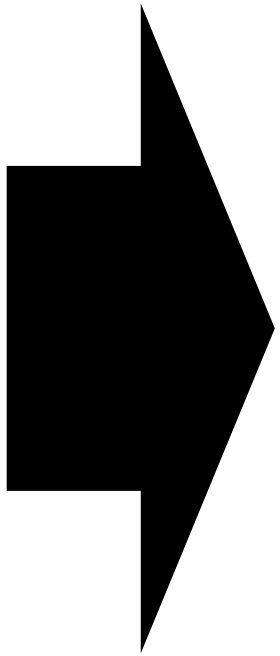
```java
package clase;

public class Proceso extends Thread{

}
```



¿ extends ?

Thread ⇔ “Hilo”

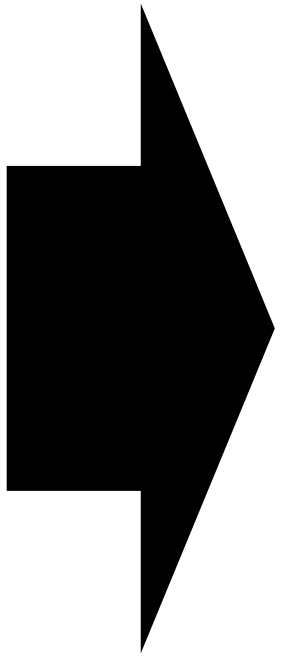
```java
package clase;

public class Proceso2 implements Runnable{

    @Override
    public void run() {

    }

}
```



¿ implements ?

Runnable ⇔ *Interface*

```java
package clase;

public class Proceso2 implements Runnable{

    @Override
    public void run() {

    }

}
```



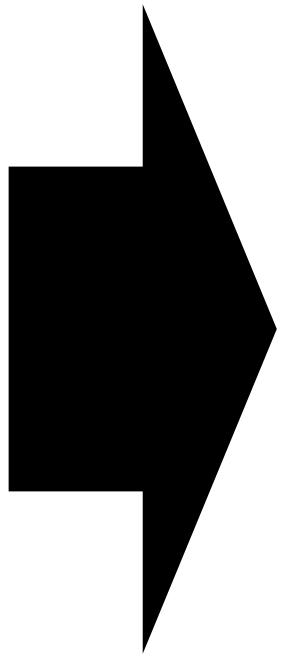
Palabra Reservada de JAVA

¿ implements ?

Runnable ⇔ *Interface*

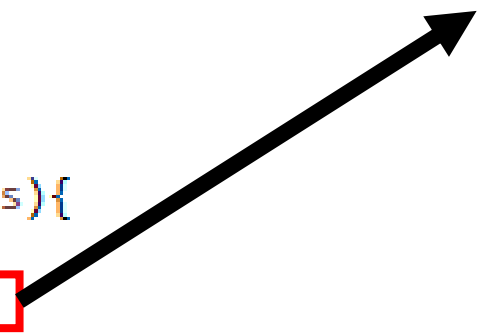
```java
package clase;

public class Proceso1 extends Thread{

    @Override
    public void run(){
        for(int i=0;i<=5;i++){
            System.out.println("Proceso1");
        }

    }
}
```

```java
package clase;

public class Proceso2 implements Runnable{

    @Override
    public void run() {
        for(int i=0;i<=5;i++){
            System.out.println("Proceso2");
        }
    }

}
```



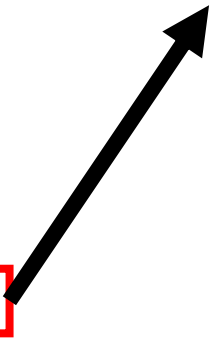
HERENCIA / IMPLEMENTACIÓN MÉTODO run()

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso1 hilo1 = new Proceso1();
        Thread hilo2 = new Thread(new Proceso2());

        hilo1.start();
        hilo2.start();

    }
}
```



... extends Thread

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso1 hilo1 = new Proceso1();
        Thread hilo2 = new Thread(new Proceso2());

        hilo1.start();
        hilo2.start();

    }
}
```



… implements Runnable

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso1 hilo1 = new Proceso1();
        Thread hilo2 = new Thread(new Proceso2());

        hilo1.start();
        hilo2.start();

    }
}
```

Cada "Hilo" debe arrancar según la instrucción

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso1 hilo1 = new Proceso1();
        Thread hilo2 = new Thread(new Proceso2());

        hilo1.start();
        hilo2.start();

    }
}
```

¿Cuál sería el resultado?

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso1 hilo1 = new Proceso1();
        Thread hilo2 = new Thread(new Proceso2());

        hilo1.start();
        hilo2.start();

    }
}
```

```
Proceso1   Proceso2
Proceso1   Proceso2
Proceso1   Proceso1
Proceso1   Proceso1
Proceso1   Proceso1
Proceso1   Proceso1
Proceso2   Proceso1
Proceso2   Proceso1
Proceso2   Proceso2
Proceso2   Proceso2
Proceso2   Proceso2
Proceso2   Proceso2
```

[...]

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso1 hilo1 = new Proceso1();
        Thread hilo2 = new Thread(new Proceso2());

        hilo1.start();
        hilo2.start();

    }
}
```

```
Proceso1    Proceso2
Proceso1    Proceso2
Proceso1    Proceso1
Proceso1    Proceso1
Proceso1    Proceso1
Proceso1    Proceso1
Proceso2    Proceso1
Proceso2    Proceso1
Proceso2    Proceso2
Proceso2    Proceso2
Proceso2    Proceso2
Proceso2    Proceso2
```

[...]

¿Por qué pasa esto?

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso1 hilo1 = new Proceso1();
        Thread hilo2 = new Thread(new Proceso2());

        hilo1.start();
        hilo2.start();

    }
}
```

```
Proceso1    Proceso2
Proceso1    Proceso2
Proceso1    Proceso1
Proceso1    Proceso1
Proceso1    Proceso1
Proceso1    Proceso1
Proceso2    Proceso1
Proceso2    Proceso1
Proceso2    Proceso2
Proceso2    Proceso2
Proceso2    Proceso2
Proceso2    Proceso2
```

[...]

Ambos *for* se están ejecutando de manera simultánea → DEPENDE DE LA VELOCIDAD DEL PROCESADOR

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso1 hilo1 = new Proceso1();
        Thread hilo2 = new Thread(new Proceso2());

        hilo1.start();
        hilo2.start();

    }
}
```

```
Proceso1    Proceso2
Proceso1    Proceso2
Proceso1    Proceso1
Proceso1    Proceso1
Proceso1    Proceso1
Proceso1    Proceso1
Proceso2    Proceso1
Proceso2    Proceso1
Proceso2    Proceso2
Proceso2    Proceso2
Proceso2    Proceso2
Proceso2    Proceso2
```

[...]

¡LA EJECUCIÓN DE LOS HILOS NO DEPENDE DE NOSOTROS!

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso1 hilo1 = new Proceso1();
        hilo1.start();

        Thread hilo2 = new Thread(new Proceso2());
        hilo2.start();

    }
}
```

¿Qué pasaría ante este escenario?

¿Hilos con parámetros?

```java
package clase;

public class Proceso extends Thread{

    int num;

    public Proceso(String nombreHilo){
        super(nombreHilo);
    }

    @Override
    public void run(){
        for(int i=0;i<=num;i++){
            System.out.println(i + this.getName());
        }


        System.out.println("------");
    }


    public void valorCondicion(int valor){
        this.num=valor;
    }
}
```

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso hilo1 = new Proceso(" Hilo 1");
        Proceso hilo2 = new Proceso(" Hilo 2");

        hilo1.valorCondicion(5);
        hilo2.valorCondicion(10);

        hilo1.start();
        hilo2.start();

    }
}
```

¿Cuál sería el resultado?

Estados de los Hilos

1. *Se crea el Hilo (new Thread)*
2. *Se Ejecuta el Hilo (Ejecutable → start())*
3. *Impedimento de ejecución de un Hilo*
4. *Muerte (Natural o Terminación Forzada de Hilos)*

Estados de los Hilos

```
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso hilo1 = new Proceso(" Hilo 1");
        Proceso hilo2 = new Proceso(" Hilo 2");

        hilo1.valorCondicion(5);
        hilo2.valorCondicion(10);

        hilo1.start();

        try{
            hilo1.sleep(1000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 1 "+ e);
        }

        hilo2.start();

        try{
            hilo2.sleep(2000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 2 "+ e);
        }

    }
}
```

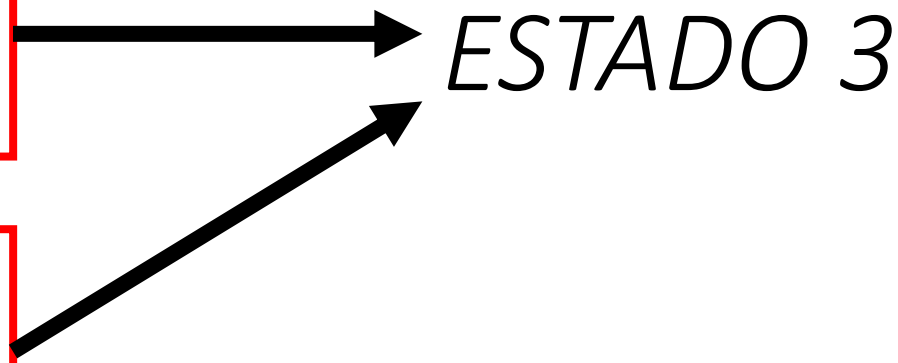
ESTADO 1

Estados de los Hilos

```
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso hilo1 = new Proceso(" Hilo 1");
        Proceso hilo2 = new Proceso(" Hilo 2");

        hilo1.valorCondicion(5);
        hilo2.valorCondicion(10);

        hilo1.start();

        try{
            hilo1.sleep(1000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 1 "+ e);
        }

        hilo2.start();

        try{
            hilo2.sleep(2000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 2 "+ e);
        }

    }
}
```

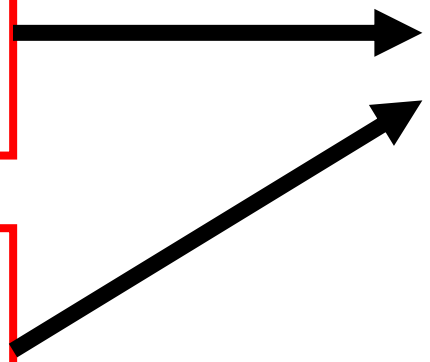
Estados de los Hilos

```
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso hilo1 = new Proceso(" Hilo 1");
        Proceso hilo2 = new Proceso(" Hilo 2");

        hilo1.valorCondicion(5);
        hilo2.valorCondicion(10);

        hilo1.start();

        try{
            hilo1.sleep(1000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 1 "+ e);
        }

        hilo2.start();

        try{
            hilo2.sleep(2000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 2 "+ e);
        }

    }
}
```



Estados de los Hilos

```
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso hilo1 = new Proceso(" Hilo 1");
        Proceso hilo2 = new Proceso(" Hilo 2");

        hilo1.valorCondicion(5);
        hilo2.valorCondicion(10);

        hilo1.start();

        try{
            hilo1.sleep(1000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 1 "+ e);
        }

        hilo2.start();

        try{
            hilo2.sleep(2000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 2 "+ e);
        }

    }
}
```



ESTADO 3: sleep(milisegundos)

Estados de los Hilos

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso hilo1 = new Proceso(" Hilo 1");
        Proceso hilo2 = new Proceso(" Hilo 2");

        hilo1.valorCondicion(5);
        hilo2.valorCondicion(10);

        hilo1.start();

        try{
            hilo1.sleep(1000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 1 "+ e);
        }

        hilo2.start();

        try{
            hilo2.sleep(2000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 2 "+ e);
        }

    }
}
```

Estado 4
Muerte Natural de Hilos

Estados de los Hilos

```
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso hilo1 = new Proceso(" Hilo 1");
        Proceso hilo2 = new Proceso(" Hilo 2");

        hilo1.valorCondicion(5);
        hilo2.valorCondicion(10);

        hilo1.start();


        try{
            hilo1.sleep(1000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 1 "+ e);
        }

        hilo2.start();
        hilo2.stop();

        try{
            hilo2.sleep(2000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 2 "+ e);
        }

    }
}
```

Estado 4
Terminación Forzada de Hilos

Estados de los Hilos

```java
package clase;

public class ClasePrincipal {
    public static void main(String[] args){

        Proceso hilo1 = new Proceso(" Hilo 1");
        Proceso hilo2 = new Proceso(" Hilo 2");

        hilo1.valorCondicion(5);
        hilo2.valorCondicion(10);

        hilo1.start();


        try{
            hilo1.sleep(1000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 1 "+ e);
        }

        hilo2.start();
        hilo2.stop();

        try{
            hilo2.sleep(2000);
        }
        catch(InterruptedException e){
            System.out.println("Error en el hilo 2 "+ e);
        }

    }
}
```

Estado 4
Terminación Forzada de Hilos

Output →

```
0 Hilo 1
1 Hilo 1
2 Hilo 1
3 Hilo 1
4 Hilo 1
5 Hilo 1
------
```

¿Preguntas?