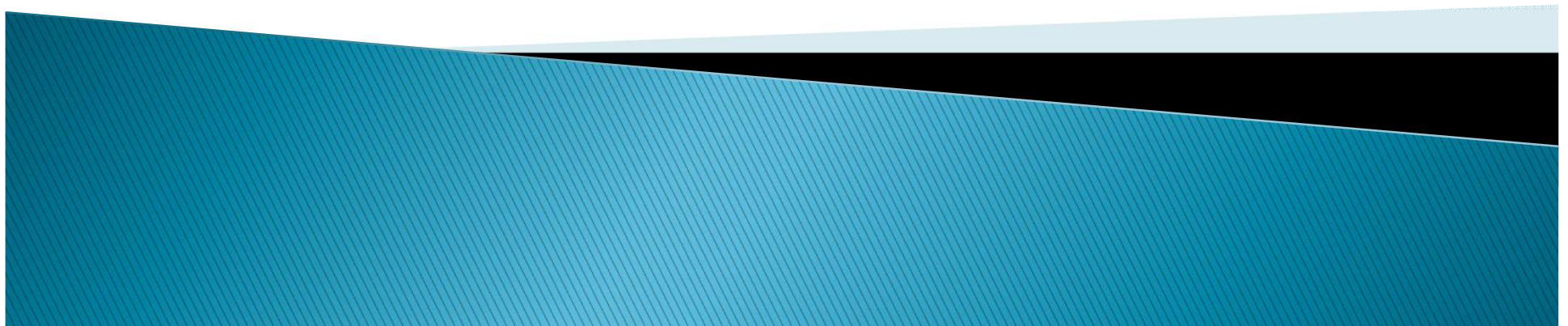


# TEMA 2: PROGRAMACIÓN DE HILOS

Programación de Servicios y Procesos  
José Manuel García



# ÍNDICE

- ▶ Concepto Hilo
- ▶ Multitarea
- ▶ Recursos compartidos por Hilos
- ▶ Estados de un Hilo
- ▶ Gestión de Hilos
- ▶ Planificación de Hilos
- ▶ Sincronización de Hilos
- ▶ Mecanismos de sincronización



# Hilo

## ▶ Hilo (thread):

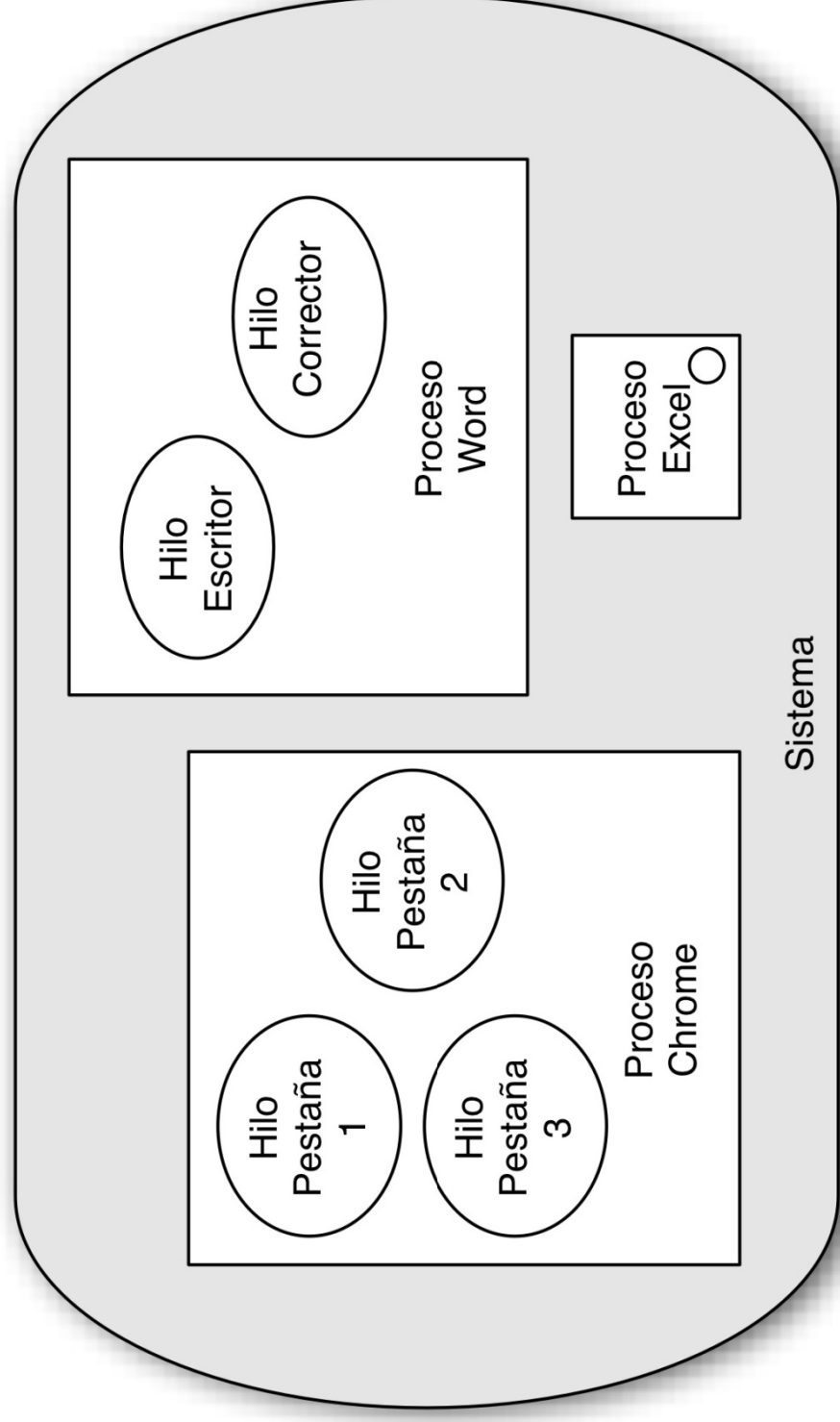
- Unidad básica de utilización de la CPU, y más concretamente de un *core* del procesador.
- Secuencia de código que está en ejecución, pero dentro del contexto de un proceso.

## ▶ Diferencia con procesos:

- Los hilos se ejecutan dentro del contexto de un proceso. Dependen de un proceso para ejecutarse.
- Los procesos son independientes y tienen espacios de memoria diferentes.
- Dentro de un mismo proceso pueden coexistir varios hilos ejecutándose que compartirán la memoria de dicho proceso.



# Hilo



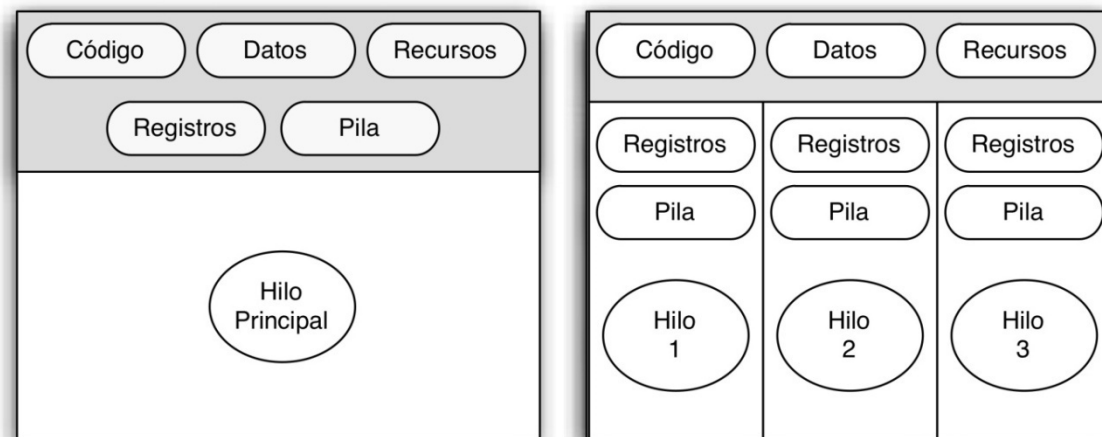
# Multitarea

- ▶ **Multitarea:** ejecución simultanea de varios hilos:
  - **Capacidad de respuesta.** Los hilos permiten a los procesos continuar atendiendo peticiones del usuario aunque alguna de las tareas (hilo) que esté realizando el programa sea muy larga.
  - **Compartición de recursos.** Por defecto, los *threads* comparten la memoria y todos los recursos del proceso al que pertenecen.
  - **La creación de nuevos hilos no supone ninguna reserva adicional de memoria** por parte del sistema operativo.
  - **Paralelismo real.** La utilización de *threads* permite aprovechar la existencia de más de un núcleo en el sistema en arquitecturas *multicore*.



# Recursos compartidos por Hilos

- ▶ Los procesos mantienen su propio espacio de direcciones y recursos de ejecución mientras que los hilos dependen del proceso.
  - Comparten con otros hilos la sección de código, datos y otros recursos.
  - Cada hilo tiene su propio contador de programa, conjunto de registros de la CPU y pila para indicar por dónde se está ejecutando.



Proceso con un único thread

Proceso con varios threads

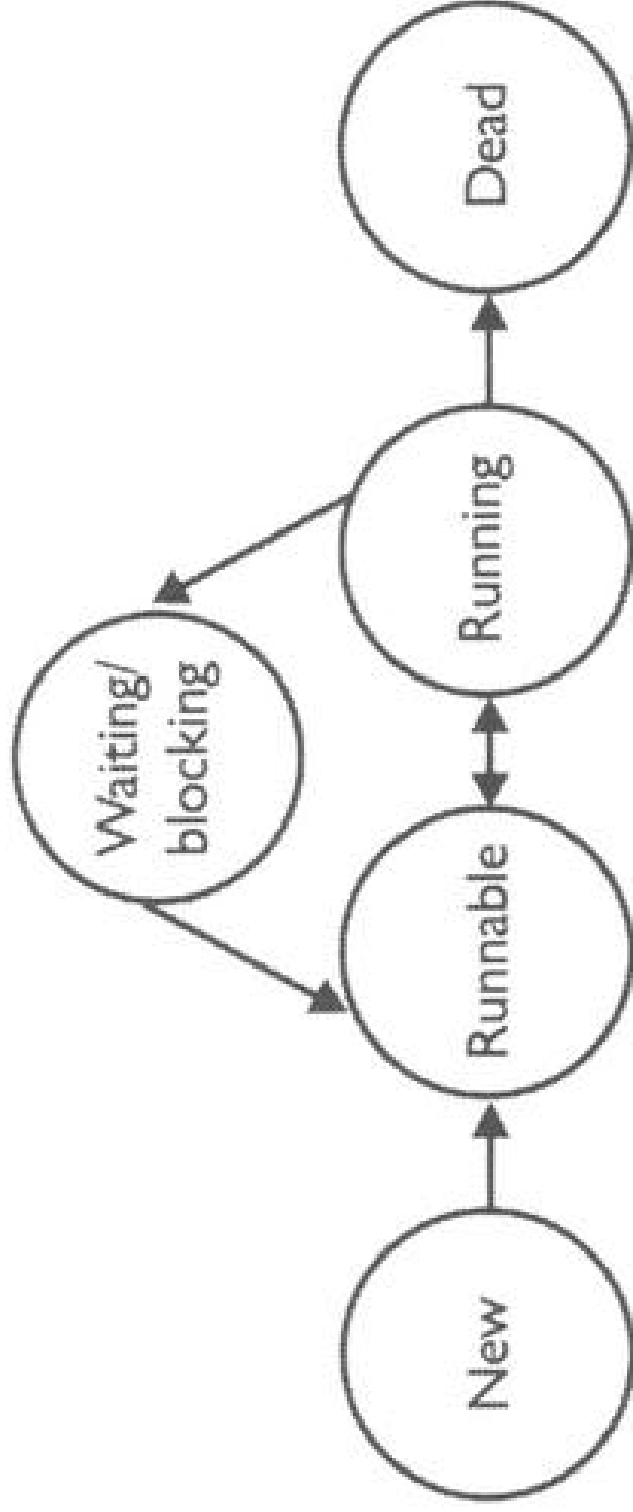
# Estados de un hilo

- ▶ Los hilos pueden cambiar de estado a lo largo de su ejecución.
- ▶ Se definen los siguientes estados:
  - **Nuevo**: el hilo está preparado para su ejecución pero todavía no se ha realizado la llamada correspondiente en la ejecución del código del programa.
  - **Listo**: el proceso no se encuentra en ejecución aunque está preparado para hacerlo. El sistema operativo no le ha asignado todavía un procesador para ejecutarse.
  - **Runnable**: el hilo está preparado para ejecutarse y puede estar ejecutándose.
  - **Bloqueado**: el hilo está bloqueado por diversos motivos esperando que el suceso suceda para volver al estado *Runnable*.
  - **Terminado**: el hilo ha finalizado su ejecución.





# Estados de un hilo





# Gestión de Hilos

- ▶ Operaciones básicas:

- Creación y arranque de hilos. Cualquier programa a ejecutarse es un proceso que tiene un hilo de ejecución principal. Este hilo puede a su vez crear nuevos hilos que ejecutarán código diferente o tareas, es decir el camino de ejecución no tiene por qué ser el mismo.
- Espera de hilos. Como varios hilos comparten el mismo procesador, si alguno no tiene trabajo que hacer, es bueno suspender su ejecución para que haya un mayor tiempo de procesador disponible.



# Creación y Arranque de hilos

- ▶ Creación de hilos en Java:
  - Implementando la interfaz **Runnable**. La interfaz *Runnable* proporciona la capacidad de añadir la funcionalidad de un hilo a una clase simplemente implementando dicha interfaz. La interfaz *Runnable* debería ser utilizada si la clase solamente va a utilizar la funcionalidad *run* de los hilos.
  - Extendiendo de la clase **Thread** mediante la creación de una subclase. La clase *Thread* es responsable de producir hilos funcionales para otras clases e implementa la interfaz *Runnable*
- ▶ El método *run()* implementa la operación *create* conteniendo el código a ejecutar por el hilo. Dicho método contendrá el hilo de ejecución.
- ▶ La clase *Thread* define también el método *start()* para implementar la operación *create*. Este método es que comienza la ejecución del hilo de la clase correspondiente.



# Creación y Arranque de Hilos.

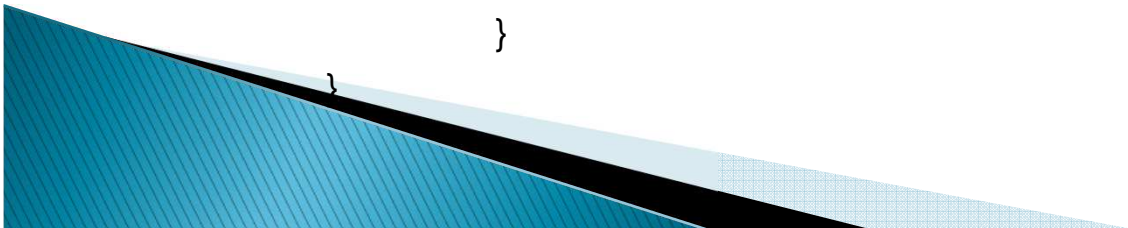
- ▶ Existen en Java, por tanto dos formas:
  - Creando una clase que herede de la clase Thread y sobrecargando el método run().
  - Implementando la interface Runnable, y declarando el método run();



# Creación y Arranque de hilos

- Implementando la interfaz *Runnable*:

```
class HelloThread implements Runnable {  
    Thread t;  
    HelloThread () {  
        t = new Thread(this, "Nuevo Thread");  
        System.out.println("Creado hilo: " + t);  
        t.start(); // Arranca el nuevo hilo de ejecución. Ejecuta run  
    }  
  
    public void run() {  
        System.out.println("Hola desde el hilo creado!");  
        System.out.println("Hilo finalizando.");  
    }  
}  
  
class RunThread {  
    public static void main(String args[]) {  
        new HelloThread(); // Crea un nuevo hilo de ejecución  
        System.out.println("Hola desde el hilo principal!");  
        System.out.println("Proceso acabando.");  
    }  
}
```



# Creación y Arranque de hilos

- ▶ Extendiendo la clase *Thread*

```
class HelloThread extends Thread {  
  
    public void run() {  
        System.out.println("Hola desde el hilo creado!");  
    }  
}  
  
public class RunThread {  
    public static void main(String args[]) {  
  
        new HelloThread().start();// Crea y arranca un nuevo hilo de ejecución  
        System.out.println("Hola desde el hilo principal!");  
        System.out.println("Proceso acabando.");  
    }  
}
```



# Hilos: Runnable vs Thread

```
public class Hilosimple extends Thread
{
    public void run{
        for (int i=0; i<5; i++)
            System.out.println("En
el hilo...");
    }
    public class usahilo{
        public static void main (String[] args)
        {
            Hilosimple hs = new
Hilosimple();
            hs.start();
            for (int i=0; i<5;i++)

                System.out.println("Fuera del
hilo..");
        }
    }
}
```

```
public class Hilosimple2 implements
Runnable{
    public void run(){
        for (int i=0; i<5; i++)

            System.out.println("En el
hilo...");
    }
    public class usahilo2{
        public static void main (String[] args)
        {
            Hilosimple2 hs = new
Hilosimple2();
            Thread t=new Thread(hs);
            t.start();
            for (int i=0; i<5;i++)

                System.out.println("Fuera del
hilo..");
        }
    }
}
```

# Creación y Arranque de hilos

- ▶ De las dos alternativas, ¿cuál utilizar? Depende de la necesidad:
  - La utilización de la interfaz *Runnable* es más general, ya que el objeto puede ser una subclase de una clase distinta de *Thread*
  - La utilización de la interfaz *Runnable* no tiene ninguna otra funcionalidad además de *run()* que la incluya por el programador.
  - La extensión de la clase *Thread* es más fácil de utilizar, ya que está definida una serie de métodos útiles para la administración de hilos,
  - La extensión de la clase *Thread* está limitada porque las clases creadas como hilos deben ser descendientes únicamente de dicha clase.





# Multihilos. Runnable

```
class NewThread implements Runnable {
    String name; // nombre del hilo
    Thread t;
    NewThread(String threadname) {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("Nuevo hilo: " + t);
        t.start(); // Comienza el hilo
    }
    // Este es el punto de entrada del hilo.
    public void run() {
        try {
            for(int i = 5; i > 0; i--) {
                System.out.println("Hilo" + name + ": " +
i);
                Thread.sleep(1000);
            }
        }
    }
}
```

```
        catch (InterruptedException e) {
            System.out.println(name + "Interrupci3n del
hilo hijo" + name);
        }
        System.out.println("Sale del hilo hijo" + name);
    }
}
class MultiThreadDemo {
    public static void main(String args[]) {
        new NewThread("Uno"); // comienzan los hilos
        new NewThread("Dos");
        new NewThread("Tres");
        try {
            // espera un tiempo para que terminen los
otros hilos
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            System.out.println("Interrupci3n del hilo
principal");
        }
        System.out.println("Sale del hilo principal.");
    }
}
```



# Multihilos. Thread

Compilación y ejecución  
desde consola

```
C:\>Javac usahilo.java  
C:\>Java usahilo
```

```
public class Hilo1 extends Thread{  
    private int c; //contador de cada hilo  
    private int hilo;  
    public Hilo1(int hilo){ //constructor  
        this.hilo=hilo;  
        System.out.println("Creando Hilo: " + hilo);}  
    public void run() {//metodo run  
        c=0;  
        while (c<=5) {  
            System.out.println("Hilo:" + hilo + " C = " + c);  
            c++;  
        }  
    }  
    public class usahilo {  
        public static void main (String[] args) {  
            Hilo1 h=null;  
            for (int i=0;i<3;i++) { //creando hilos  
                h=new Hilo1(i+1);  
                h.start();  
            }  
            System.out.println("3 Hilos Creados...");  
        }  
    } //clase
```

# Espera de hilos


- ▶ Operaciones:
  - **Join**: La ejecución del hilo puede suspenderse esperando hasta que el hilo correspondiente por el que espera finalice su ejecución.
  - **Sleep**: duerme un hilo por un período especificado
- ▶ Ambas operaciones de espera pueden ser interrumpidas, si otro hilo interrumpe al hilo actual mientras está suspendido por dichas llamadas.
  - Una interrupción es una indicación a un hilo que debería dejar de hacer lo que esté haciendo para hacer otra cosa.
  - Un hilo envía una interrupción mediante la invocación del método ***interrupt()*** en el objeto del hilo que se quiere interrumpir.
- ▶ ***isAlive()***: comprueba si el hilo no ha finalizado su ejecución antes de trabajar con él.



# Interrupción

- ▶ Indica a un hilo que debe dejar de hacer lo que esté haciendo para hacer otra cosa.
  - Objeto `hilo.interrupt()` y se recibe excepción `InterruptedException`

```
public void run() {  
    for (int i = 0; i < NDatos; i++) {  
        try {  
            System.out.println("Esperando a recibir dato!");  
            Thread.sleep(500);  
        } catch (InterruptedException e) {  
            System.out.println("Hilo interrumpido.");  
            return;  
        }  
        // Gestiona dato i  
    }  
    System.out.println("Hilo finalizando correctamente.");  
}
```



# Clase Thread

Método	Tipo Retorno	Descripción
start()	void	Implementa la operación Create. Comienza la ejecución del hilo de la clase correspondiente. Llama al método run()
run()	void	Si el hilo se construyó implementando la interfaz Runnable, entonces se ejecuta el método run() de ese objeto. En caso contrario no hace nada.
currentThread()	static Thread	Devuelve una referencia al objeto hilo que se está ejecutando actualmente
join()	void	Implementa la operación join para hilos. Suspende la ejecución del hilo hasta que finalice otro
sleep(long milis)	static void	El hilo que se está ejecutando se suspende durante el número de milisegundos especificado
interrupt()	void	Interrumpe el hilo del objeto
interrupted()	static boolean	Comprueba si el hilo ha sido interrumpido
isAlive()	boolean	Devuelve true en caso de que el hilo esté vivo, es decir, no haya terminado el método run() en su ejecución

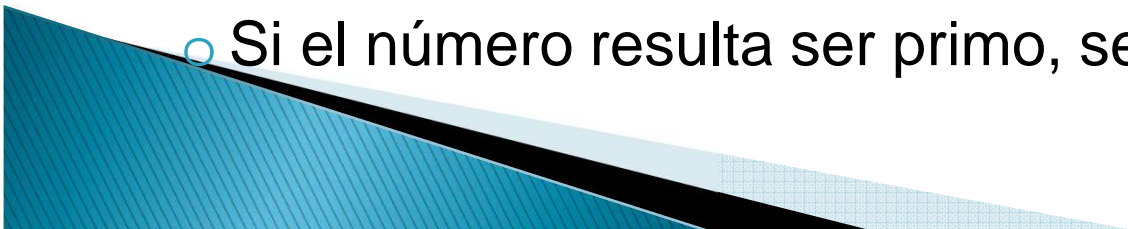
# Práctica. Multihilo

- ▶ Diseña un programa que cree 4 hilos de forma que:
  - uno sume los primeros 10 números naturales
  - otro los multiplique
  - otro devuelva la suma de los 10 primeros números pares
  - otro la suma de los 10 primeros números impares.
- ▶ El programa principal deberá devolver la suma de los cuatro resultados.
- ▶ Implementar una clase con 4 propiedades y escribir en ellas cada resultado.



# Práctica. Multihilo

- ▶ Escribe un programa que calcule la suma desde 1 hasta un número Entero N, dividiéndose en 2 hilos. Ej:  $N = 20$ , Hilo 1 de 1 a 10, Hilo 2 de 11 a 20.
- ▶ Escribe un programa que se le pase por parámetro un número entero N, y calcule la suma desde 1 hasta N, lanzando tantos hilos como sea necesario para que todos sumen la misma cantidad de números. Número máximo de Hilos será 20.
  - Ej:  $10.000 / 500 = 20$  hilos y cada uno suma 500 números.
    - Calculo un divisor y lanzo tantos hilos como me de el cociente siempre que no sea superior a 20.
  - Si el número resulta ser primo, se lanzará solo 1 hilo





# Práctica. Multihilo

- ▶ Escribir un programa que genere un array de  $N \times 4$  números aleatorios.
- ▶ Posteriormente este programa padre creará 4 hilos que recorran cada uno  $\frac{1}{4}$  del array y busque el número más alto.
- ▶ El programa principal pintará los cuatro números obtenidos de cada hijo e indicará cual es el más alto y cual es el más bajo de los cuatro.

