

Nombre y apellidos (1): **Laura Llorens Angulo**Nombre y apellidos (2): **Martin Martinez Ramos**Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio): **1:15****Tema 01. Introducción a la Programación Concurrente y Paralela****Tema 02. Programación de Procesadores Multinúcleo (PM) y Multiprocesadores con Memoria Compartida (MMC)****Tema 03. Conceptos Básicos de Concurrencia en Java****1** Este ejercicio constituye una primera aproximación al ejercicio siguiente.

Se desea mostrar en pantalla todos los números comprendidos entre 0 y $n - 1$ (incluyendo ambos extremos), donde n es un dato recibido en la línea de argumentos. El orden en el que se muestran los datos no es importante y se desea emplear varias hebras.

En la línea de comandos, el primer argumento será el número de hebras y el número n será el segundo. Si el número de argumentos de la línea de argumentos no es correcto, el programa debe avisar y terminar. Asimismo, si algún argumento no es correcto (por ejemplo, se esperaba un argumento numérico y no lo es), el programa también debe avisar y terminar.

Seguidamente se muestra la versión inicial de los códigos a implementar.

```
// =====
class EjemploMuestraNumeros {
// =====

// =====
public static void main( String args[] ) {
    int n, numHebras;

    // Comprobacion y extraccion de los argumentos de entrada.
    if( args.length != 2 ) {
        System.err.println( "Uso: java programa <numHebras> <n>" );
        System.exit( -1 );
    }
    try {
        numHebras = Integer.parseInt( args[ 0 ] );
        n = Integer.parseInt( args[ 1 ] );
    } catch( NumberFormatException ex ) {
        numHebras = -1;
        n = -1;
        System.out.println( "ERROR: Argumentos numericos incorrectos." );
        System.exit( -1 );
    }
    //
    // Implementacion paralela con distribucion ciclica o por bloques.
    //
    // Crea un vector de hebras. Crea y arranca las hebras
```

```

    // ...
    // Espera a que terminen todas las hebras.
    // ...
}

```

1.1) Implementa una versión paralela mediante el uso de hebras con una *distribución cíclica*.

Realiza varias comprobaciones variando tanto el número de hebras como n , centrándote en los casos en los que n no es un múltiplo del número de hebras, por ejemplo $n = 13$ y $nH = 4$. Comprueba que aparecen todos los números deseados y que no hay repetidos.

Escribe a continuación la definición de la clase y el código a incluir en el programa principal.

```

package Laboratorio02.Ejercicio1.apartado1;

// =====
class EjemploMuestraNumerosDistribuciónCíclica {
// =====

    public static void main( String args[] ) {
        int n, numHebras;

        // Comprobacion y extraccion de los argumentos de entrada.
        if( args.length != 2 ) {
            System.err.println( "Uso: java programa <numHebras> <n>" );
            System.exit( -1 );
        }

        try {
            numHebras = Integer.parseInt( args[ 0 ] );
            n = Integer.parseInt( args[ 1 ] );
        } catch( NumberFormatException ex ) {
            numHebras = -1;
            n = -1;
            System.out.println( "ERROR: Argumentos numericos incorrectos." );
            System.exit( -1 );
        }
        HebraDistCiclica v[] = new HebraDistCiclica[ numHebras ];
        for(int i=0 ; i < numHebras ; i++){
            v[i] = new HebraDistCiclica(i, numHebras, n);
            v[i].start();
        }
        for(int i=0 ; i < numHebras ; i++){
            try{
                v[i].join();
            }catch (InterruptedException ex){
                ex.printStackTrace();
            }
        }
    }
}

class HebraDistCiclica extends Thread{
    int mild, numHebras, n;

    public HebraDistCiclica(int mild, int numHebras, int n){
        this.mild = mild;
        this.numHebras = numHebras;
        this.n = n;
    }

    public void run(){
        for(int i = mild; i < n; i += numHebras){
            System.out.println( " Hebra : " + mild + " Mostrando el número " + i );
        }
    }
}

```

- 1.2) Implementa una versión paralela mediante el uso de hebras con una *distribución por bloques*. Realiza varias comprobaciones variando tanto el número de hebras como n , centrándote en los casos en los que n no es un múltiplo del número de hebras, por ejemplo $n = 13$ y $nH = 4$. Comprueba que aparecen todos los números deseados y que no hay repetidos. Escribe a continuación la definición de la clase y el código a incluir en el programa principal.

```
package Laboratorio02.Ejercicio1.apartado2;
// =====
class EjemploMuestraNumerosDistribuciónPorBloques {
// =====

// - - - - -
public static void main( String args[] ) {
    int n, numHebras;

    // Comprobacion y extraccion de los argumentos de entrada.
    if( args.length != 2 ) {
        System.err.println( "Uso: java programa <numHebras> <n>" );
        System.exit( -1 );
    }
    try {
        numHebras = Integer.parseInt( args[ 0 ] );
        n = Integer.parseInt( args[ 1 ] );
    } catch( NumberFormatException ex ) {
        numHebras = -1;
        n = -1;
        System.out.println( "ERROR: Argumentos numericos incorrectos." );
        System.exit( -1 );
    }
    HebraDistPorBloques v[] = new HebraDistPorBloques[ numHebras ];
    for(int i=0 ; i < numHebras ; i++){
        v[i] = new HebraDistPorBloques(i, numHebras, n);
        v[i].start();
    }
    for(int i=0 ; i < numHebras ; i++){
        try{
            v[i].join();
        }catch (InterruptedException ex){
            ex.printStackTrace();
        }
    }
}

class HebraDistPorBloques extends Thread{
    int mild, numHebras, n;

    public HebraDistPorBloques(int mild, int numHebras, int n){
        this.mild = mild;
        this.numHebras = numHebras;
        this.n = n;
    }

    public void run(){
        int tamanyo = (n + numHebras - 1) / numHebras;
        int inicio = tamanyo * mild;
        int fin = Math.min(inicio+tamanyo,n);
        for(int i = inicio; i < fin; i++){
            System.out.println( " Hebra : " + mild + " Mostrando el número " + i );
        }
    }
}
```

2 Evaluación de una función en múltiples puntos.

Se dispone de un programa secuencial que evalúa una función en varios puntos. Para almacenar datos y resultados, el código emplea dos vectores de la misma dimensión: `vectorX` y `vectorY`. El código evalúa la función para cada elemento de `vectorX` dejando el resultado calculado en el correspondiente elemento de `vectorY`. Este tipo de cálculo es muy habitual y se puede realizar por ejemplo para visualizar una función en pantalla.

Tras realizar los cálculos, en lugar de visualizar los resultados en pantalla, el programa realiza y muestra en pantalla la suma de los elementos de ambos vectores para que después se puedan comprobar fácilmente los resultados con las versiones paralelas.

El programa secuencial declara e inicializa una variable con el número de hebras, pero no la emplea para nada. Dicho valor deberá ser aprovechado en las implementaciones paralelas.

El código es el siguiente:

```
// =====
class EjemploFuncionCostosa {
// =====

// =====
public static void main( String args[] ) {
    int      n, numHebras;
    long      t1, t2;
    double    sumaX, sumaY, ts, tc, tb;

    // Comprobacion y extraccion de los argumentos de entrada.
    if( args.length != 2 ) {
        System.err.println( "Uso: java programa <numHebras> <tamanyo>" );
        System.exit( -1 );
    }
    try {
        numHebras = Integer.parseInt( args[ 0 ] );
        n         = Integer.parseInt( args[ 1 ] );
    } catch( NumberFormatException ex ) {
        numHebras = -1;
        n         = -1;
        System.out.println( "ERROR: Argumentos numericos incorrectos." );
        System.exit( -1 );
    }

    // Crea los vectores.
    double vectorX[] = new double[ n ];
    double vectorY[] = new double[ n ];

    //
    // Implementacion secuencial (sin temporizar).
    //
    inicializaVectorX( vectorX );
    inicializaVectorY( vectorY );
    for( int i = 0; i < n; i++ ) {
        vectorY[ i ] = evaluaFuncion( vectorX[ i ] );
    }

    //
    // Implementacion secuencial.
    //
    inicializaVectorX( vectorX );
    inicializaVectorY( vectorY );
    t1 = System.nanoTime();
```

```

    for( int i = 0; i < n; i++ ) {
        vectorY[ i ] = evaluaFuncion( vectorX[ i ] );
    }
    t2 = System.nanoTime();
    ts = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.println( "Tiempo secuencial (seg.):" + ts );
    //// imprimeResultado( vectorX, vectorY );
    // Comprueba el resultado.
    sumaX = sumaVector( vectorX );
    sumaY = sumaVector( vectorY );
    System.out.println( "Suma del vector X:" + sumaX );
    System.out.println( "Suma del vector Y:" + sumaY );
}

/*
//
// Implementacion paralela ciclica.
//
inicializaVectorX( vectorX );
inicializaVectorY( vectorY );
t1 = System.nanoTime();
// Gestion de hebras para la implementacion paralela ciclica
// ....
t2 = System.nanoTime();
tc = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
System.out.println( "Tiempo paralela ciclica (seg.):" + tc );
System.out.println( "Incremento paralela ciclica:" + ... );
//// imprimeResultado( vectorX, vectorY );
// Comprueba el resultado.
sumaX = sumaVector( vectorX );
sumaY = sumaVector( vectorY );
System.out.println( "Suma del vector X:" + sumaX );
System.out.println( "Suma del vector Y:" + sumaY );
//
// Implementacion paralela por bloques.
//
// ....
//
*/

System.out.println( "Fin del programa." );
}

// -----
static void inicializaVectorX( double vectorX[] ) {
    if( vectorX.length == 1 ) {
        vectorX[ 0 ] = 0.0;
    } else {
        for( int i = 0; i < vectorX.length; i++ ) {
            vectorX[ i ] = 10.0 * ( double ) i / ( ( double ) vectorX.length - 1 );
        }
    }
}

// -----
static void inicializaVectorY( double vectorY[] ) {
    for( int i = 0; i < vectorY.length; i++ ) {
        vectorY[ i ] = 0.0;
    }
}

// -----
static double sumaVector( double vector[] ) {

```


2.1) Paraleliza el código anterior mediante el uso de hebras con una *distribución cíclica*.

Descomenta el código situado debajo de “Implementacion secuencial”. Incluye la gestión de hebras que paraleliza el bucle comprendido entre la lectura de `t1` y `t2` en la versión secuencial, y la expresión que permite calcular el incremento de velocidad.

Verifica en varios casos que el nuevo código devuelve el mismo resultado que el original.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de la clase hebra y el código a incluir en el programa principal que permite gestionar los objetos de esta clase, así como la visualización de los resultados y prestaciones (`tc` e incremento).

```
//
// Implementacion paralela ciclica.
//
inicializaVectorX( vectorX );
inicializaVectorY( vectorY );
t1 = System.nanoTime();
// Gestion de hebras para la implementacion paralela ciclica
HebraDistCiclica v[] = new HebraDistCiclica[ numHebras ];
for(int i=0 ; i < numHebras ; i++){
    v[i] = new HebraDistCiclica(i, numHebras, n, vectorX, vectorY);
    v[i].start();
}
for(int i=0 ; i < numHebras ; i++){
    try{
        v[i].join();
    }catch (InterruptedException ex){
        ex.printStackTrace();
    }
}

t2 = System.nanoTime();
tc = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
System.out.println( "Tiempo paralela ciclica (seg.):      " + tc );
System.out.println( "Incremento paralela ciclica:      " + (ts/tc) );
//// imprimeResultado( vectorX, vectorY );
// Comprueba el resultado.
sumaX = sumaVector( vectorX );
sumaY = sumaVector( vectorY );
System.out.println( "Suma del vector X:      " + sumaX );
System.out.println( "Suma del vector Y:      " + sumaY );

System.out.println( "Fin del programa." );
}

class HebraDistCiclica extends Thread{
    int mild, numHebras, n;
    double vectorX[] = new double[ n ];
    double vectorY[] = new double[ n ];

    public HebraDistCiclica(int mild, int numHebras, int n, double vectorX[], double vectorY[]){
        this.mild = mild;
        this.numHebras = numHebras;
        this.n = n;
        this.vectorX = vectorX;
        this.vectorY = vectorY;
    }

    public void run(){
        for(int i = mild; i < n; i += numHebras){
            double x = vectorX[ i ];
            vectorY[ i ] = Math.sin( Math.exp( -x ) + Math.log10( x + 1 ) );
        }
    }
}
```

2.2) Paraleliza el anterior código mediante el uso de hebras con una *distribución por bloques*.

Replica el código de la “Implementación cíclica” y adapta la gestión de hebras. La nueva versión paralela debe ejecutarse tras las anteriores versiones.

Verifica en varios casos que el nuevo código devuelve el mismo resultado que el original.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de la clase hebra y el código a incluir en el programa principal que permite gestionar los objetos de esta clase, así como la visualización de los resultados y prestaciones (**tc** e incremento)..

```
// Implementacion paralela por bloques.
//
inicializaVectorX( vectorX );
inicializaVectorY( vectorY );
t1 = System.nanoTime();
// Gestion de hebras para la implementacion paralela por bloques
HebraDistBloques vB[] = new HebraDistBloques[numHebras];
for(int i=0 ; i < numHebras ; i++){
    vB[i] = new HebraDistBloques(i, numHebras, n, vectorX, vectorY);
    vB[i].start();
}
for(int i=0 ; i < numHebras ; i++){
    try{
        vB[i].join();
    }catch (InterruptedException ex){
        ex.printStackTrace();
    }
}

t2 = System.nanoTime();
tc = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
System.out.println( "Tiempo paralela bloques (seg.):          " + tc );
System.out.println( "Incremento paralela bloques:          " + (ts/tc) );
//// imprimeResultado( vectorX, vectorY );
// Comprueba el resultado.
sumaX = sumaVector( vectorX );
sumaY = sumaVector( vectorY );
System.out.println( "Suma del vector X:          " + sumaX );
System.out.println( "Suma del vector Y:          " + sumaY );
...
class HebraDistBloques extends Thread{
    int mild, numHebras, n;
    double vectorX[] = new double[ n ];
    double vectorY[] = new double[ n ];

    public HebraDistBloques(int mild, int numHebras, int n, double vectorX[], double vectorY[]){
        this.mild = mild;
        this.numHebras = numHebras;
        this.n = n;
        this.vectorX = vectorX;
        this.vectorY = vectorY;
    }

    public void run(){
        int tamanyo = (n + numHebras - 1) / numHebras;
        int inicio = tamanyo * mild;
        int fin = min(inicio+tamanyo,n);
        for(int i = inicio; i < fin; i++){
            double x = vectorX[ i ];
            vectorY[ i ] = Math.sin( Math.exp( -x ) + Math.log10( x + 1 ) );
        }
    }
}
```


- 2.3) Evalúa los códigos en el ordenador del laboratorio.

Copia los resultados en las siguientes tablas redondeándolos con dos decimales.

Ten en cuenta que, para poder calcular los incrementos de velocidad correctamente, se deben realizar las dos versiones paralelas (cíclica y por bloques) en una misma ejecución, comparando sus prestaciones con las de la versión secuencial.

Incrementos de velocidad para $n = 1\,000\,000$			
Número de hebras	1	2	4
Distribución cíclica	0.88	1.55	2.21
Distribución por bloques	0.92	1.47	2.23

Incrementos de velocidad para $n = 10\,000\,000$			
Número de hebras	1	2	4
Distribución cíclica	1.06	2.15	3.65
Distribución por bloques	1.07	2.22	3.66

Examina con detalle y justifica los resultados.

Podemos ver que cuando aumentamos el número de hebras aumenta el incremento porque al repartir la tarea entre varias hebras es mas rápido.

- 2.4) Cuando la función `evaluaFuncion` del apartado anterior se aplica a un vector de números, ¿el código resultante está limitado por la CPU, por la memoria central o por la E/S?

Esta limitado por la CPU porque aunque acceda a memoria central para acceder al dato cuando lo obtiene hace muchas operaciones con él.

- 2.5) Si se utilizase la función `evaluaFuncion` que se muestra y se aplicase a un vector de números, entonces ¿el código estaría limitado por la CPU, por la memoria central o por la E/S?

```
static double evaluaFuncion( double x ) {
    return 4.5 * x;
}
```

Esta limitado por memoria central ya que no podemos almacenar el vector en los registros, por tanto hay que acceder a memoria tantas veces como elementos hay en el vector.

- 2.6) Haz una copia del programa que acabas de completar y dale el nombre de `EjemploFuncion Sencilla.java`. Modifícalo para que emplee la nueva función `evaluaFuncion` más sencilla. A continuación, calcula con dos decimales los incrementos de velocidad correspondientes para completar las siguientes tablas.

Incrementos de velocidad para $n = 1\,000\,000$			
Número de hebras	1	2	4
Distribución cíclica	0.63	0.43	0.33
Distribución por bloques	0.85	0.33	0.24

Incrementos de velocidad para $n = 10\,000\,000$			
Número de hebras	1	2	4
Distribución cíclica	1.17	0.99	0.93
Distribución por bloques	1.15	0.89	0.84

Examina con detalle y justifica los resultados.

Como esta limitado por memoria central a mayor número de hebras menor velocidad por eso vemos que cuantas más hebras menor es el incremento.

- 2.7) ¿Los dos códigos (EjemploFuncionCostosa y EjemploFuncionSencilla) obtienen incrementos aceptables? Comenta los resultados.

.....

En el caso de función constosa vemos un gran incremento, lo que significa que con hebras es mas rápido. Pero en el caso de funcion sencilla el incremento es muy pequeño, no merece la pena hacer hebras, es mejor hacer el programa secuencial.

.....

- 2.8) Evalúa los códigos en patan.

Calcula con dos decimales los incrementos de velocidad necesarios para completar las siguientes tablas.

Recuerda que, para poder calcular los incrementos de velocidad correctamente, se deben ejecutar las dos versiones paralelas (cíclica y por bloques) en una misma ejecución, comparando sus prestaciones con las de la versión secuencial.

Examina con detalle y justifica los resultados.

Resultados para <code>evaluaFuncionCostosa</code> Incrementos de velocidad para $n = 100\ 000\ 000$				
Número de hebras	1	4	8	16
Distribución cíclica				
Distribución por bloques				

Resultados para <code>evaluaFuncionSencilla</code> Incrementos de velocidad para $n = 100\ 000\ 000$				
Número de hebras	1	4	8	16
Distribución cíclica				
Distribución por bloques				

.....

.....

.....

.....

.....

.....

.....

Nota: Esta entrega forma parte de la evaluación de la asignatura. Debe ser guardado por el estudiantado junto con el resto de entregas en una carpeta. El profesorado podrá pedir al estudiantado que le entregue dicha carpeta en cualquier momento.