EI1024/MT1024 "Programación Concurrente y Paralela" 2025–26	Entregable para Laboratorio
Nombre y apellidos (1):	
Nombre y apellidos (2):	la02_g
Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio):	

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 introducido 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 ] );
      if( ( numHebras <= 0 ) || ( n <= 0 ) ) {
        System.err.print( "Uso: [ java programa <numHebras> <n> ] " );
        System.err.println( "donde ( numHebras > 0 )  y ( n > 0 )" );
        System.exit( -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
    // (A) ...
    // Espera a que terminen todas las hebras.
    // (B) ...
    //
  }
}

// Crea las clases adicionales que sean necesarias
// (C) ...
//
```

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 el código a incluir en el programa principal (A y B) la definición de la clase (C).

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 el código a incluir en el programa principal (A y B) la definición de la clase (C).

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 utiliza 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 ] );
      if( ( numHebras <= 0 ) || ( n <= 0 ) ) {
        System.err.print( "Uso: [ java programa <numHebras> <n> ] " );
        System.err.println( "donde ( numHebras > 0 )  y ( n > 0 )" );
        System.exit( -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
    // (A) ....
    t2 = System.nanoTime();
    tc = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.println( "Tiempo paralela ciclica (seg.):              " + tc );
    System.out.println( "Incremento paralela ciclica:                 " + ... ); // (B)
    //// 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.
    //
    // (C) ....
    //
*/

    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[] ) {
    double  suma = 0.0;
    for( int i = 0; i < vector.length; i++ ) {
      suma += vector[ i ];
    }
    return suma;
  }

  // ---------------------------------------------------------------------
  static double evaluaFuncion( double x ) {
    return -Math.cos( Math.exp( -x ) + Math.log1p( x ) );
  }

  // ---------------------------------------------------------------------
  static void imprimeVector( double vector[] ) {
    for( int i = 0; i < vector.length; i++ ) {
      System.out.println( " vector[ " + i + " ] = " + vector[ i ] );
    }
  }

  // ---------------------------------------------------------------------
  static void imprimeResultado( double vectorX[], double vectorY[] ) {
    for( int i = 0; i < Math.min( vectorX.length, vectorY.length ); i++ ) {
      System.out.println( "  i: " + i +
                          "  x: " + vectorX[ i ] +
                          "  y: " + vectorY[ i ] );
    }
  }

}

// Crea las clases adicionales que sean necesarias
// (D) ...
//
```

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

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

ATENCIÓN: Los ejercicios anteriores deben realizarse en casa. Los siguientes, en el aula.

2.2) Paraleliza el anterior código mediante el uso de hebras con una *distribución por bloques*.
Crea en (D) una clase nueva hebra para este caso. Replica en (C) el código del programa principal de la "Implementación cíclica", para que se ejecute tras las anteriores versiones.
Verifica en varios casos que el nuevo código devuelve el mismo resultado que el original.
Escribe la parte de tu código que realiza tal tarea: la definición de la clase hebra (D) y el código a incluir en el programa principal que permite gestionar los objetos de esta clase (A), así como la visualización de los resultados y prestaciones (`tb` e incremento) (B).

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 $n = 5\ 000\ 000$			
Número de hebras	1	2	4
Distribución cíclica	0,98	1,80	3,033
Distribución por bloques	1,03	1,79	3,24

Incrementos de velocidad para $n = 50\ 000\ 000$			
Número de hebras	1	2	4
Distribución cíclica	1,04	1,96	3,27
Distribución por bloques	1,01	1,95	3,57

Examina con detalle y justifica los resultados.

Podemos ver que cuanto más datos, más se acerca el incremento de velocidad la máximo teórico del numero de hebras.

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

Está limitada por la CPU porque en cada ejecución tiene que realizar una operación matemática complicada.

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

```
static double evaluaFuncion( double x ) {
  return 5.4 * x;
}
```

Está limitada por la memoria, la operación es sencilla y mucho más rápida que el acceso a memoria.

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 `evaluaFunción` más sencilla. A continuación, calcula con dos decimales los incrementos de velocidad correspondientes para completar las siguientes tablas.

Incrementos de velocidad para $n = 5\ 000\ 000$			
Número de hebras	1	2	4
Distribución cíclica			
Distribución por bloques			

Incrementos de velocidad para $n = 50\ 000\ 000$			
Número de hebras	1	2	4
Distribución cíclica			
Distribución por bloques			

Examina con detalle y justifica los resultados.

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

2.8) Evalúa los códigos en karen.

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 `evaluaFuncionCostosa` Incrementos de velocidad para $n = 100\ 000\ 000$				
Número de hebras	1	4	8	16
Distribución cíclica				
Distribución por bloques				

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