

Nombre y apellidos (1):

Nombre y apellidos (2):

Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio):Entregable
para
Laboratorio

la03_g

Tema 04. El Problema de la Visibilidad en Java

Tema 05. El Problema de la Atomicidad en Java

1 Estudia el siguiente código y responde a las siguientes preguntas.

```
// =====
class CuentaIncrementos {
    // =
    int numIncrementos = 0;

    // -
    void incrementaNumIncrementos() {
        numIncrementos++;
    }

    // -
    int dameNumIncrementos() {
        return( numIncrementos );
    }
}

// =====
class MiHebra extends Thread {
    // =
    int numIters;
    CuentaIncrementos c;

    // -
    public MiHebra( int numIters, CuentaIncrementos c ) {
        this.numIters = numIters;
        this.c = c;
    }

    // -
    public void run() {
        for( int i = 0; i < numIters; i++ ) {
            c.incrementaNumIncrementos();
        }
    }
}

// =====
class EjemploCuentaIncrementos {
    // =
}
```

```

public static void main( String args[] ) {
    long      t1 , t2;
    double    tt;
    int       numHebras , numIters;

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

    System.out.println( "numHebras: " + numHebras );
    System.out.println( "numIters : " + numIters );

    System.out.println( "Creando y arrancando " + numHebras + " hebras." );
    t1 = System.nanoTime();
    MiHebra v[] = new MiHebra[ numHebras ];
    CuentaIncrementos c = new CuentaIncrementos();
    for( int i = 0; i < numHebras; i++ ) {
        v[ i ] = new MiHebra( numIters , c );
        v[ i ].start();
    }
    for( int i = 0; i < numHebras; i++ ) {
        try {
            v[ i ].join();
        } catch( InterruptedException ex ) {
            ex.printStackTrace();
        }
    }
    t2 = System.nanoTime();
    tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.println( "Total de incrementos: " + c.dameNumIncrementos() );
    System.out.println( "Tiempo transcurrido en segs.: " + tt );
}
}

```

- 1.1) ¿Qué realiza el código? ¿Qué debería mostrar en pantalla si se ejecutase con los parámetros hebras 4 y numIters 1 000 000?
 En principio se trata de un programa que suma sobre el mismo contador, en este caso si todo funcionase correctamente el contador finalmente tendría que ser de 4.000.000 ya que hay 4 hebras y se realizan 1.000.000 de iteraciones.
-

- 1.2) Compila y ejecuta el código con dichos valores en tu ordenador local. ¿Qué muestra realmente en pantalla si se ejecuta con los parámetros hebras 4 y numIters 1 000 000?

```
numHebras: 4
··· numIters : 1000000
··· Creando y arrancando 4.hebras.
··· Total de incrementos: 1062625
··· Tiempo transcurrido en segs.: 0.0210288
```

- 1.3) ¿Es un código *thread-safe*? Justifica tu respuesta.

No se trata de un código *tread-safe* ya que las hebras pueden recoger en cualquier momento valores del contador que no es el contador actual, es decir, están sumando en un contador que no es el valor correcto.

- 1.4) Crea una **copia del código original** (*EjemploCuentaIncrementosVolatile.java*) e inserta el modificador **volatile** en la variable **numIncrementos** de la clase **CuentaIncrementos**. A continuación, compila y prueba el nuevo código.

¿Resuelve el problema el modificador **volatile**? ¿Por qué?

No, ya que con el modificador **volatile** solo estamos resolviendo el problema de visibilidad, es decir, que dos variables no puedan escribir a la vez en la variable **numIncrementos**. Se trata de una operación atomica por lo tanto se tendrá que bloquear.

- 1.5) ¿Se podría resolver con el modificador **synchronized**?

Para comprobarlo, crea una **copia del código original** (*EjemploCuentaIncrementosSynchronized.java*) y aplica el modificador **synchronized** sobre **cada una** de las rutinas de la clase **CuentaIncrementos**.

Después, compila y prueba el código, antes de contestar a la pregunta anterior.

Escribe a continuación los cambios realizados en la clase **CuentaIncrementos**.

Si, ya que con **synchronized** resolvemos los problemas de visibilidad y atomicidad.

- 1.6) ¿También se podría arreglar empleando clases y operadores atómicos?

Para comprobarlo, crea otra **copia del código original** (`EjemploAtomic.java`), **ELIMINA** la clase `CuentaIncrementos` y utiliza en su lugar una **clase atómica y sus métodos**. Despues, compila y prueba el código, antes de contestar la pregunta.

Escribe a continuación los cambios realizados en el código.

Sí, es mejor usar las clases que nos ofrece Java para solucionar los problemas de atomicidad y visibilidad.

- 1.7) Completa la siguiente tabla con datos de todas las versiones anteriores en tu ordenador, utilizando hebras 4 y un numIters de 1 000 000. Comenta los resultados.

Código	Total incrementos
Código original	2469318
Código con <code>volatile</code>	1789023
Código con <code>synchronized</code>	4000000
Código con clases atómicas	4000000

Con synchronized y las clases atomicas resolvemos el problema. Porque se obtiene un número menor en volatile ???

2 Se desea imprimir en pantalla los números primos que aparecen en un vector.

El código completo es el siguiente:

```

// =====
public class EjemploMuestraPrimosEnVector {
// =====

// -----
public static void main( String args[] ) {
    int      numHebras, vectOpt;
    boolean option = true;
    long     t1, t2;
    double   ts, tc, tb, td;

    // Comprobacion y extraccion de los argumentos de entrada.
    if( args.length != 2 ) {
        System.out.println( "Uso: java programa <numHebras> <vectOpt>" );
        System.exit( -1 );
    }
    try {
        numHebras = Integer.parseInt( args[ 0 ] );
        vectOpt   = Integer.parseInt( args[ 1 ] );
        if( ( numHebras <= 0 ) || ( ( vectOpt != 0 ) && ( vectOpt != 1 ) ) ){
            System.out.print( "Uso: [ java programa <numHebras> <vecOpt> ] " );
            System.out.println( "donde ( numHebras > 0 ) y ( vectOpt es 0 o 1 )" );
            System.exit( -1 );
        } else {
            option = ( vectOpt == 0 );
        }
    } catch( NumberFormatException ex ) {
        numHebras = -1;
        System.out.println( "ERROR: Argumentos numericos incorrectos." );
        System.exit( -1 );
    }

    // Eleccion del vector de trabajo
    // -----
    VectorNumeros vn = new VectorNumeros ( option );
    long vectorTrabajo[] = vn.vector;

    // -----
    // Implementacion secuencial.
    // -----
    System.out.println( "" );
    System.out.println( "Implementacion secuencial." );
    t1 = System.nanoTime();
    for( int i = 0; i < vectorTrabajo.length; i++ ) {
        if( esPrimo( vectorTrabajo[ i ] ) ) {
            System.out.println( " Encontrado primo: " + vectorTrabajo[ i ] );
        }
    }
    t2 = System.nanoTime();
    ts = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.println( "Tiempo secuencial (seg.): " + ts );
}

/*
// -----
// Implementacion paralela ciclica.
// -----
System.out.println( "" );
System.out.println( "Implementacion paralela ciclica." );

```


2.1) Compila y ejecuta el programa anterior, utilizando un 0 como segundo parámetro.

En este caso se trabaja con el siguiente vector de números:

Implementacion secuencial.

Encontrado primo: 200000081

Encontrado primo: 200000089

Encontrado primo: 2000000393
Encontrado primo: 2000000107

Encontrado primo: 200000107 Encuentro primo: 200000113

Encontrado primo: 201098166 son los números primos contenidos en el vector?

Encontrado primo: 200000123

Encontrado primo: 200000131

Encontrado primo: 200000161

Encontrado primo: 200000183

Encontrado primo: 200000201

Encontrado primo: 200000209

Encontrado primo: 200000221

Encontrado primo: 200000237

Encontrado primo: 200000239

Encontrado primo: 20000

- 2.2) Realiza una implementación paralela con distribución cíclica, en la que cada hebra procese un conjunto de elementos del vector. Para cada elemento del vector procesado, SOLO se mostrará su valor si el número es primo.

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

Comprueba que los números primos mostrados en la versión paralela coinciden con los de la versión secuencial.

Escribe, a continuación, la parte de tu código que realiza tal tarea: la definición de la clase **MiHebraPrimoDistCiclica** (E) y el código a incluir en el programa principal que permite gestionar los objetos de esta clase (A-B).

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

- 2.3) Realiza una implementación paralela con distribución por bloques, en la que cada hebra procese un conjunto de elementos del vector. Para cada elemento del vector procesado, SOLO se mostrará su valor si el número es primo.

Crea en (E) una clase nueva hebra para este caso. Replica en (C) el código del programa principal de la “Implementación paralela cíclica”, para que se ejecute tras las otras versiones. Comprueba que los números primos mostrados en la versión paralela coinciden con los de la versión secuencial.

Escribe, a continuación, la parte de tu código que realiza tal tarea: la definición de la clase **MiHebraPrimoDistPorBloques** (E) y el código a incluir en el programa principal que permite gestionar los objetos de esta clase (A-B).

- 2.4) Realiza una implementación paralela con distribución dinámica, que utilice un número entero atómico (`AtomicInteger`), para apuntar a una posición del vector. Las hebras recibirán un objeto de este tipo, que siempre contendrá la primera posición del vector sin procesar. Para ello, las hebras deben **realizar, de modo atómico, la lectura del valor actual y su incremento**. Las hebras finalizarán cuando el índice sobrepase la dimensión del vector. Crea en (E) una clase nueva hebra para este caso. Replica en (D) el código del programa principal de la “Implementación paralela cíclica”, para que se ejecute tras las otras versiones. Comprueba que los números primos mostrados en la versión paralela coinciden con los de la versión secuencial.

Escribe, a continuación, la parte de tu código que realiza tal tarea: la definición de la clase `MiHebraPrimoDistDinamica` (E) y el código a incluir en el programa principal que permite gestionar los objetos de esta clase (A-B).

- 2.5) Completa la siguiente tabla, obteniendo los resultados para 4 hebras en el ordenador del aula y los resultados para 16 hebras en karen. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

	4 hebras (aula)		16 hebras (karen)	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela con distribución cíclica				
Paralela con distribución por bloques				
Paralela con distribución dinámica				

Justifica los resultados de la tabla anterior.

- 2.7) Evalúa y compara las tres versiones (secuencial, paralela cíclica y paralela por bloques), pero en este caso utilizando 1 como segundo parámetro, es decir, manejando el vector:

```
long vectorTrabajo [] = {  
    200000081L, 4L,  
    200000083L, 4L,  
    200000089L, 4L,  
    200000093L, 4L,  
    200000107L, 4L,  
    200000117L, 4L,  
    200000123L, 4L,  
    200000131L, 4L,  
    200000161L, 4L,  
    200000183L, 4L,  
    200000201L, 4L,  
    200000209L, 4L,  
    200000221L, 4L,  
    200000237L, 4L,  
    200000239L, 4L,  
    200000243L, 4L, 4L}
```

Completa la siguiente tabla, obteniendo los resultados para 4 hebras en el ordenador del aula y los resultados para 16 hebras en karen. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

	4 hebras (aula)		16 hebras (karen)	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela con distribución cíclica				
Paralela con distribución por bloques				
Paralela con distribución dinámica				

2.8) Justifica los resultados de la tabla anterior.

.....

2.9) ¿Cuál es la mejor distribución con ambos vectores? Justifica tu respuesta.

.....

3 Empleando el ordenador del aula, completa la siguiente tabla con datos de todas las versiones desarrolladas en el ejercicio 1, utilizando hebras 4 y un numIters de 10 000 000. Redondea los tiempos dejando sólo tres decimales y comenta los resultados.

Código	Total incrementos	Tiempo transcurrido (seg.)
Código original		
Código con <code>volatile</code>		
Código con <code>synchronized</code>		
Código con clases atómicas		

.....

