

Nombre y apellidos (1): .....

Nombre y apellidos (2): .....

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

## 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.err.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.err.print( "Uso: [ java programa <numHebras> <n> ] " );
            System.err.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?

El código crea 4 hilos y hace que incremente 1 000 000 de veces. En teoría debería de incrementarse 4 000 000 (1 000 000 por cada hilo) de veces, pero seguramente como el código no es thread-safe saldrá otra cosa

- 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?

Me da 1937910

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

No lo es porque el valor de numIncrementos es distinto para cada hebra y al actualizarlo cambio no es visible por las demás, además el método incrementaNumIncrementos no es atómico.

- 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 lo resuelve porque el método de incrementaNumIncrementos no es atómico y cuando llama un hilo puede que otro lo este llamando tmb y se de una condición de carrera

- 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`.

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

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

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

- 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**.

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

Escribe a continuación los cambios realizados en el código.

```
class MiHebraAtomic extends Thread {
    //
    =====
    =====
    int numIters;
    AtomicInteger c;

    // -----
    public MiHebraAtomic(int numIters, AtomicInteger c ) {
        this.numIters = numIters;
        this.c = c;
    }

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

- 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	1633019
Código con <code>volatile</code>	2862451
Código con <code>synchronized</code>	4000000
Código con clases atómicas	4000000

**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.err.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.err.print( "Uso: [ java programa <numHebras> <vectOpt> ] " );
            System.err.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." );
```



```

4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
};
} else {
vector = new long [] {
2000000081L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000083L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000089L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000093L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000107L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000117L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000123L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000131L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000161L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000183L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000201L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000209L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000221L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000237L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000239L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
2000000243L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
};
}
}
}

```

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:

[illegible]

¿Cuáles son los números primos contenidos en el vector?

200000081, 200000083, 200000089, 200000093, 200000107, 200000117, 200000123, 200000131,  
200000161, 200000183, 200000201, 200000209, 200000221, 200000237, 200000239, 200000243,

- 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 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.

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).



- 2.5) 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		—	16,023	—
Paralela con distribución cíclica			1,033	15,51
Paralela con distribución por bloques			16,007	1,001
Paralela con distribución dinámica			1,037	15,45

- 2.6) Justifica los resultados de la tabla anterior.

Como todos los números primos están al principio del vector, las distribuciones cíclica y dinámica funcionan mejor porque, en el primer caso casualmente se reparten de tal manera que a cada hebra le tocan la misma distribución de primos y no primos y en el caso de la dinámica por diseño se reparten las tareas más largas. La distribución por bloques no funciona bien aquí porque todos los primos le caen a unas pocas hebras.

- 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, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000083L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000089L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000093L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000107L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000117L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000123L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000131L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000161L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000183L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000201L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000209L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000221L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000237L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000239L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
    200000243L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 4L, 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		—	16,024	—
Paralela con distribución cíclica			16,01	1,001
Paralela con distribución por bloques			1,009	15,89
Paralela con distribución dinámica			1,027	15,61

2.8) Justifica los resultados de la tabla anterior.

En este caso, la cíclica no funciona bien porque todos los primos le tocan a la misma hebra. La distribución por bloques funciona muy bien porque reparte equitativamente la carga. La dinámica funciona bien por diseño sin importar los datos dados.

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

La distribución dinámica es la que mejor funciona porque esta asegura que todas las hebras realicen un trabajo similar. Esta distribución permite que las hebras a las que les toca datos más sencillos y terminan antes, sigan trabajando ayudando a las hebras que aún están calculando. En casos donde los datos pueden ser muy diversos la distribución dinámica se asegura que no pueda caerle todo el trabajo a unas pocas.

**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		