Nombre y apellido: Enric Gil Gallen

Nombre y apellido: Victor Granados Segara

Tiempo en casa: 1:00

Tema 06. El Problema de la Atomicidad en Java

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

1.1) ¿Qué realiza el código? ¿Qué debería mostrar en pantalla si se ejecutase con los parámetros hebras 4 y tope 1 000 000?

- El código realiza tantos incrementos como numHebras * tope.
- En este caso debería devolver 4000000 de incrementos

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 tope 1 000 000?

```
numHebras: 4
tope:      1000000
Creando y arrancando 4 hebras.
Total de incrementos: 1052613
Tiempo transcurrido en segs.: 0.0128302
```

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

- No, Aparecen problemas de visibilidad y atomicidad

1.4) Crea una copia del fichero original e inserta en la copia el modificador **volatile** en la clase **CuentaIncrementos1a**. A continuación, compila y prueba el nuevo código.
¿Resuelve el problema el modificador **volatile**? ¿Por qué?

Si que resuelve Visibilidad pero No lo resuelve, porque continuamos teniendo el problema de Atomicidad

1.5) ¿Se puede arreglar con el modificador **synchronized**?
Para ello, crea una copia del código original, aplica el modificador **synchronized** sobre cada una de las rutinas de la clase **CuentaIncrementos1a**.
Después compila y prueba el código contestar la pregunta con el resultado obtenido.
Escribe a continuación los cambios realizados en la clase **CuentaIncrementos1a**.

Si que se puede arreglar

```
package practica_3;

// ============================================================================
class CuentaIncrementos_Synchronized {
// ============================================================================
 int numIncrementos = 0;

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

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


// ============================================================================
class MiHebra_Synchronized extends Thread {
// ============================================================================
 int tope;
 CuentaIncrementos_Synchronized  c;

 // ----------------------------------------------------------------------
 public MiHebra_Synchronized( int tope, CuentaIncrementos_Synchronized c ) {
   this.tope  = tope;
   this.c     = c;
 }

 // ----------------------------------------------------------------------
 synchronized public void run() {
   for( int i = 0; i < tope; i++ ) {
```

```
      c.incrementaNumIncrementos();
    }
  }
}

// ============================================================================
class EjemploCuentaIncrementos_Synchronized {
// ============================================================================

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

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

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

   System.out.println( "Creando y arrancando " + numHebras + " hebras." );
   t1 = System.nanoTime();
   MiHebra_Synchronized v[] = new MiHebra_Synchronized[ numHebras ];
   CuentaIncrementos_Synchronized c = new CuentaIncrementos_Synchronized();
   for( int i = 0; i < numHebras; i++ ) {
     v[ i ] = new MiHebra_Synchronized( tope, 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 );
 }
```

```
}
```

1.6) ¿Se puede arreglar empleando clases y operadores atómicos?
Para ello, crea otra copia del código original, ELIMINA COMPLETAMENTE la clase **`CuentaIncrementos1a`** y utiliza en su lugar una **clase atómica y sus métodos**.
Después compila y prueba el código contestar la pregunta con el resultado obtenido.
Escribe a continuación los cambios realizados en el código.

Si que se puede arreglar

```
// ============================================================================
class MiHebra_ClasesAtomicas extends Thread {
// ============================================================================
  int tope;
  AtomicInteger  c;

  // --------------------------------------------------------------------------
  public MiHebra_ClasesAtomicas( int tope, AtomicInteger c ) {
    this.tope  = tope;
    this.c     = c;
  }

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

// ============================================================================
class EjemploCuentaIncrementos_ClasesAtomicas {
// ============================================================================

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

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

```
    }
    try {
      numHebras = Integer.parseInt( args[ 0 ] );
      tope      = Integer.parseInt( args[ 1 ] );
    } catch( NumberFormatException ex ) {
      numHebras = -1;
      tope      = -1;
      System.out.println( "ERROR: Argumentos numericos incorrectos." );
      System.exit( -1 );
    }

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

    System.out.println( "Creando y arrancando " + numHebras + " hebras." );
    t1 = System.nanoTime();
    MiHebra_ClasesAtomicas v[] = new MiHebra_ClasesAtomicas[ numHebras ];
    AtomicInteger c = new AtomicInteger();
    for( int i = 0; i < numHebras; i++ ) {
      v[ i ] = new MiHebra_ClasesAtomicas( tope, 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.get() );
    System.out.println( "Tiempo transcurrido en segs.: " + tt );
  }
}
```

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

Código	Total incrementos
Código original	
Código con `volatile`	
Código con `synchronized`	
Código con clases atómicas	

Código	Total incrementos
Código Original	2522475
Código con volatile	1361911
Código con synchronized	4000000
Código con clases atomicas	4000000

2 Se dispone del siguiente vector:

```
long vectorNumeros[] = {
        200000033L, 200000039L, 200000051L, 200000069L,
        200000161L, 200000183L, 200000201L, 200000209L,
        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
    };
```

Se desea imprimir en pantalla aquellos números primos contenidos en el vector anterior.

El código completo es el siguiente:

2.1) Compila y ejecuta el programa anterior. ¿Cuáles son los números primos contenidos en el vector?

```
Implementacion secuencial.
  Encontrado primo: 200000033
  Encontrado primo: 200000039
  Encontrado primo: 200000051
  Encontrado primo: 200000069
  Encontrado primo: 200000161
  Encontrado primo: 200000183
  Encontrado primo: 200000201
  Encontrado primo: 200000209
Tiempo secuencial (seg.):                    11.8601064

Process finished with exit code 0
```

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, se mostrará su valor SOLO si es primo.

Incluye la gestión de hebras a continuación de la implementación secuencial.

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** y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

```
package practica_3;

import practica_2.Ej1_MyThreadCiclica;

// ============================================================================
public class EjemploMuestraPrimosEnVector2a {
// ============================================================================

 // ----------------------------------------------------------------------
 public static void main( String args[] ) {
   int     numHebras;
   long    t1, t2;
   double  ts, tc, tb, td, sp_ciclical;
   long    vectorNumeros[] = {
               200000033L, 200000039L, 200000051L, 200000069L,
               200000161L, 200000183L, 200000201L, 200000209L,
               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
           };
   //// long    vectorNumeros[] = {
               //// 200000033L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
               //// 200000039L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
               //// 200000051L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
               //// 200000069L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
               //// 200000161L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
               //// 200000183L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
               //// 200000201L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
               //// 200000209L, 4L, 4L, 4L, 4L, 4L, 4L, 4L
           //// };
   // Comprobacion y extraccion de los argumentos de entrada.
   if( args.length != 1 ) {
     System.err.println( "Uso: java programa <numHebras>" );
     System.exit( -1 );
   }
   try {
     numHebras = Integer.parseInt( args[ 0 ] );
```

```
} catch( NumberFormatException ex ) {
  numHebras = -1;
  System.out.println( "ERROR: Argumentos numericos incorrectos." );
  System.exit( -1 );
}
//
// Implementacion secuencial.
//
System.out.println( "" );
System.out.println( "Implementacion secuencial." );
t1 = System.nanoTime();
for( int i = 0; i < vectorNumeros.length; i++ ) {
  if( esPrimo( vectorNumeros[ i ] ) ) {
    System.out.println( "  Encontrado primo: " + vectorNumeros[ 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." );
t1 = System.nanoTime();
// Gestion de hebras para la implementacion paralela ciclica

// Crea y arranca el vector de hebras.
MiHebraPrimoDistCiclica[] vectorHebras = new MiHebraPrimoDistCiclica[numHebras];
for (int i = 0; i < numHebras; i++) {
  vectorHebras[i] = new MiHebraPrimoDistCiclica(i, numHebras, vectorNumeros);
  vectorHebras[i].start();
}

// Espera a que terminen las hebras.

for (int i = 0; i < numHebras; i++) {
  try {
    vectorHebras[i].join();
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}

t2 = System.nanoTime();
tc = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
sp_ciclical = ts / tc;
System.out.println( "Tiempo paralela ciclica (seg.):              " + tc );
System.out.println( "Incremento paralela ciclica:                 " + sp_ciclical );
/*
//
// Implementacion paralela por bloques.
```

```
   //
   // ....
   //
   // Implementacion paralela dinamica.
   //
   // ....
*/
 }

 // ---------------------------------------------------------------------------
 static boolean esPrimo( long num ) {
   boolean primo;
   if( num < 2 ) {
     primo = false;
   } else {
     primo = true;
     long i = 2;
     while( ( i < num )&&( primo ) ) {
       primo = ( num % i != 0 );
       i++;
     }
   }
   return( primo );
 }
}
```

```
import static practica_3.EjemploMuestraPrimosEnVector2a.esPrimo;

public class MiHebraPrimoDistCiclica extends Thread  {
   int miId, numHebdas;
   long[] vectorNumeros;

   public MiHebraPrimoDistCiclica(int miId,  int numHebdas, long[] vectorNumeros) {
       this.miId = miId;this.numHebdas = numHebdas;
       this.vectorNumeros = vectorNumeros;
   }

   // Implementacion paralela con distribucion ciclica
   public void run() {
       int ini = miId;
       int fin = vectorNumeros.length;
       int inc = numHebdas;

       for( int i = ini; i < fin; i += inc ) {
           if( esPrimo( vectorNumeros[ i ]) ) {
               System.out.println( "  Encontrado primo: " + vectorNumeros[ i ] );
           }
       }

   }

}
```

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, se mostrará su valor SOLO si es primo.

Incluye la gestión de hebras a continuación de la implementación cíclica.

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` y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

```
/*
//
// Implementacion paralela por bloques.
//
 */
System.out.println( "" );
System.out.println( "Implementacion paralela bloques." );
t1 = System.nanoTime();
// Gestion de hebras para la implementacion paralela ciclica

// Crea y arranca el vector de hebras.
MiHebraPrimoDistPorBloques[] vectorHebrasBloques = new
MiHebraPrimoDistPorBloques[numHebras];

for (int i = 0; i < numHebras; i++) {
 vectorHebrasBloques[i] = new MiHebraPrimoDistPorBloques(i, numHebras,
vectorNumeros.length, vectorNumeros);
 vectorHebrasBloques[i].start();
}

// Espera a que terminen las hebras.

for (int i = 0; i < numHebras; i++) {
 try {
   vectorHebrasBloques[i].join();
 } catch (InterruptedException e) {
   e.printStackTrace();
 }
}

t2 = System.nanoTime();
tb = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
sp_bloques = ts / tc;
System.out.println( "Tiempo paralela bloques (seg.):              " + tb );
System.out.println( "Incremento paralela or bloques:                 " + sp_bloques );

package practica_3;

import static practica_3.EjemploMuestraPrimosEnVector2a.esPrimo;

public class MiHebraPrimoDistPorBloques extends Thread {
   int miId, numHebdas, nElem;
```

```
   long[] vectorNumeros;

   public MiHebraPrimoDistPorBloques(int miId, int numHebdas, int nElem, long[]
vectorNumeros) {
       this.miId = miId;
       this.numHebdas = numHebdas;
       this.nElem = nElem;
       this.vectorNumeros = vectorNumeros;
   }

   // Implementacion paralela con distribucion ciclica
   public void run() {
       int tamBloque = (nElem + numHebdas - 1) / numHebdas;
       int iniElem = tamBloque * miId;
       int finElem = Math.min(iniElem + tamBloque, nElem);

       for (int i = iniElem; i < finElem; i++)
           if (esPrimo(vectorNumeros[i])) {
               System.out.println("  Encontrado primo: " + vectorNumeros[i]);
           }
   }
}
```

2.4) Realiza una implementación paralela con distribución dinámica, utilizando un número entero atómico (`AtomicInteger`). Las hebras recibe un único objeto de este tipo, que siempre debe almacenar el primer índice 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.

Incluye la gestión de hebras a continuación de la implementación por bloques.

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` y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

```
//
// Implementacion paralela dinamica.
//
// ....

System.out.println( "" );
System.out.println( "Implementacion paralela dinamica." );
t1 = System.nanoTime();
// Gestion de hebras para la implementacion paralela ciclica

AtomicInteger n = new AtomicInteger(0);
// Crea y arranca el vector de hebras.
MiHebraPrimoDistPorDinamica[] vectorHebrasDinamica = new
```

```
MiHebraPrimoDistPorDinamica[numHebras];

 for (int i = 0; i < numHebras; i++) {
   vectorHebrasDinamica[i] = new MiHebraPrimoDistPorDinamica(n, vectorNumeros, i,
numHebras);
   vectorHebrasDinamica[i].start();
 }

 // Espera a que terminen las hebras.

 for (int i = 0; i < numHebras; i++) {
   try {
     vectorHebrasDinamica[i].join();
   } catch (InterruptedException e) {
     e.printStackTrace();
   }
 }

 t2 = System.nanoTime();
 td = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
 sp_dianmica = ts / td;
 System.out.println( "Tiempo paralela bloques (seg.):              " + td );
 System.out.println( "Incremento paralela or bloques:                 " + sp_dianmica );

}


package practica_3;

import java.util.concurrent.atomic.AtomicInteger;

import static practica_3.EjemploMuestraPrimosEnVector2a.esPrimo;

public class MiHebraPrimoDistPorDinamica  extends Thread {
   AtomicInteger n;
   long[] vectorNumeros;
   int miId, inumHebdas;

   public MiHebraPrimoDistPorDinamica(AtomicInteger n, long[] vectorNumeros, int miId,
int inumHebdas) {
       this.n = n;
       this.vectorNumeros = vectorNumeros;
       this.miId = miId;
       this.inumHebdas = inumHebdas;
   }

   // Implementacion paralela con distribucion ciclica
   public void run() {
       int valor_inicial = n.getAndIncrement();
       int incremento = n.get();

       for (int pos = valor_inicial; pos < vectorNumeros.length; pos = incremento){
```

```
            if (esPrimo(vectorNumeros[pos])) {
                System.out.println("  Encontrado primo: " + vectorNumeros[pos]);
            }

            incremento = n.getAndIncrement();
        }

    }
}
```

2.5) Completa la siguiente tabla, obteniendo los resultados para 4 núcleos en el ordenador del aula y los resultados para 8 núcleos en patan. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

	4 núcleos		8 núcleos	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela con distribución cíclica				
Paralela con distribución por bloques				
Paralela con distribución dinámica				

	4		**8**	
	Tiempo	**Incrementos**	**Tiempo**	**Incremento**
Secuencia	20, 209	-	8,017	-
Cíclica	3.709	5.6301	2.016	9.976
Bloques	12.037	1.73518961	8	1
Dinámica	3.822	5.465	2	3.9

2.6) Justifica los resultados de la tabla anterior.

Se observa que el más lento es el secuencial dado que solo tenemos un hilo. Se observa que la peor distribución es la de bloques ya que encuentra los número primos es el primer hilo, encambio la cíclica cada hilo se encuentra con un primo por hilo. Por ultimo la dimaca reparte “el trabajo” independientemente de las posiciones del vector y el hilo.

```
Running on master host ep03.local
Time is Thu May 5 19:22:27 CEST 2022
Directory is /home/al387320/Practica_e3
Procsfile: /var/spool/torque/aux//20658.patan.act.uji.es
This jobs runs on the following processors:
ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03
Running on host ep03.local
Time is Thu May 5 19:22:27 CEST 2022
Directory is /home/al387320/Practica_e3

Implementacion secuencial.
  Encontrado primo: 200000033
  Encontrado primo: 200000039
  Encontrado primo: 200000051
  Encontrado primo: 200000069
  Encontrado primo: 200000161
  Encontrado primo: 200000183
  Encontrado primo: 200000201
  Encontrado primo: 200000209
Tiempo secuencial (seg.):                 8.016576569

Implementacion paralela ciclica.
  Encontrado primo: 200000051
  Encontrado primo: 200000033
  Encontrado primo: 200000069
  Encontrado primo: 200000039
  Encontrado primo: 200000201
  Encontrado primo: 200000161
  Encontrado primo: 200000209
  Encontrado primo: 200000183
Tiempo paralela ciclica (seg.):           2.015923677
Incremento paralela ciclica:              3.9766270223731293

Implementacion paralela bloques.
  Encontrado primo: 200000033
  Encontrado primo: 200000039
  Encontrado primo: 200000051
  Encontrado primo: 200000069
  Encontrado primo: 200000161
  Encontrado primo: 200000183
  Encontrado primo: 200000201
  Encontrado primo: 200000209
Tiempo paralela bloques (seg.):           8.006789938
Incremento paralela or bloques:              1.001222291464592

Implementacion paralela dinamica.
  Encontrado primo: 200000039
  Encontrado primo: 200000069
  Encontrado primo: 200000051
  Encontrado primo: 200000033
  Encontrado primo: 200000161
  Encontrado primo: 200000183
```

```
  Encontrado primo: 200000201
  Encontrado primo: 200000161
  Encontrado primo: 200000209
  Encontrado primo: 200000183
Tiempo paralela ciclica (seg.):           2.015923677
Incremento paralela ciclica:              3.9766270223731293

Implementacion paralela bloques.
  Encontrado primo: 200000033
  Encontrado primo: 200000039
  Encontrado primo: 200000051
  Encontrado primo: 200000069
  Encontrado primo: 200000161
  Encontrado primo: 200000183
  Encontrado primo: 200000201
  Encontrado primo: 200000209
Tiempo paralela bloques (seg.):           8.006789938
Incremento paralela or bloques:              1.001222291464592

Implementacion paralela dinamica.
  Encontrado primo: 200000039
  Encontrado primo: 200000069
  Encontrado primo: 200000051
  Encontrado primo: 200000033
  Encontrado primo: 200000161
  Encontrado primo: 200000183
  Encontrado primo: 200000201
  Encontrado primo: 200000209
Tiempo paralela bloques (seg.):           2.011531185
Incremento paralela or bloques:              3.9853106075509337
```

2.7) Evalúa y compara las tres versiones (secuencial, paralela cíclica y paralela por bloques), pero en este caso con el vector siguiente:

```
long vectorNumeros[] = {
        200000033L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
        200000039L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
        200000051L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
        200000069L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
        200000161L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
        200000183L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
        200000201L, 4L, 4L, 4L, 4L, 4L, 4L, 4L,
        200000209L, 4L, 4L, 4L, 4L, 4L, 4L, 4L
};
```

Completa la siguiente tabla, obteniendo los resultados para 4 núcleos en el ordenador del aula y los resultados para 8 núcleos en patan. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

	4 núcleos		8 núcleos	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela con distribución cíclica				
Paralela con distribución por bloques				
Paralela con distribución dinámica				

	4		**8**	
	Tiempo	**Incrementos**	**Tiempo**	**Incremento**
Secuencia	12.638	-	8	-
Cíclica	12.127	1.042	8	1
Bloques	3.906	3.234	2	3.994
Dinámica	4.52	2.972	2	4

```
Running on master host ep03.local
Time is Thu May 5 19:29:48 CEST 2022
Directory is /home/al387320/Practica_e3
Procsfile: /var/spool/torque/aux//20660.patan.act.uji.es
This jobs runs on the following processors:
ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03 ep03
Running on host ep03.local
Time is Thu May 5 19:29:48 CEST 2022
Directory is /home/al387320/Practica_e3

Implementacion secuencial.
  Encontrado primo: 200000033
  Encontrado primo: 200000039
  Encontrado primo: 200000051
  Encontrado primo: 200000069
  Encontrado primo: 200000161
  Encontrado primo: 200000183
  Encontrado primo: 200000201
  Encontrado primo: 200000209
Tiempo secuencial (seg.):                   8.016385693

Implementacion paralela ciclica.
  Encontrado primo: 200000033
  Encontrado primo: 200000039
  Encontrado primo: 200000051
  Encontrado primo: 200000069
  Encontrado primo: 200000161
  Encontrado primo: 200000183
  Encontrado primo: 200000201
  Encontrado primo: 200000209
Tiempo paralela ciclica (seg.):             8.010684985
Incremento paralela ciclica:                1.0007116380198042

Implementacion paralela bloques.
  Encontrado primo: 200000051
  Encontrado primo: 200000033
  Encontrado primo: 200000201
  Encontrado primo: 200000161
  Encontrado primo: 200000069
  Encontrado primo: 200000039
  Encontrado primo: 200000183
  Encontrado primo: 200000209
Tiempo paralela bloques (seg.):             2.006649253
Incremento paralela or bloques:                3.9949112586642963

Implementacion paralela dinamica.
  Encontrado primo: 200000051
  Encontrado primo: 200000033
  Encontrado primo: 200000039
  Encontrado primo: 200000069
  Encontrado primo: 200000201
  Encontrado primo: 200000183
```

```
  Encontrado primo: 200000161
  Encontrado primo: 200000183
  Encontrado primo: 200000201
  Encontrado primo: 200000209
Tiempo paralela ciclica (seg.):             8.010684985
Incremento paralela ciclica:                1.0007116380198042

Implementacion paralela bloques.
  Encontrado primo: 200000051
  Encontrado primo: 200000033
  Encontrado primo: 200000201
  Encontrado primo: 200000161
  Encontrado primo: 200000069
  Encontrado primo: 200000039
  Encontrado primo: 200000183
  Encontrado primo: 200000209
Tiempo paralela bloques (seg.):             2.006649253
Incremento paralela or bloques:                3.9949112586642963

Implementacion paralela dinamica.
  Encontrado primo: 200000051
  Encontrado primo: 200000033
  Encontrado primo: 200000039
  Encontrado primo: 200000069
  Encontrado primo: 200000201
  Encontrado primo: 200000183
  Encontrado primo: 200000161
  Encontrado primo: 200000209
Tiempo paralela bloques (seg.):             2.003891462
Incremento paralela or bloques:                4.000409126449983
```

2.8) Justifica los resultados de la tabla anterior.

La distribución secuencial en este caso no varía sustancialmente del ejercicio anterior. La distribución cíclica, en cambio, una hebra trabajará excesivamente en comparación con las demás. Todo lo contrario en el caso de la distribución por bloques, que se distribuye el trabajo de forma eficiente, por lo que a cada hebra procesa como mínimo un número primo pesado y otros más ligeros. Mismo caso que en el anterior ejercicio, la distribución dinámica se adapta a cada caso.

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

Depende de los núcleos utilizados en el caso de 4

- Vector 1: Cíclica
- Vector 2: Bloques

En cambio con 8, la mejor siempre es la dinámica ya que tenemos más opciones de repartición

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 tope de 1 000 000. Redondea los tiempos dejando sólo tres decimales y comenta los resultados.

Código	Total incrementos	Tiempo transcurrido (seg.)
Código original		
Código con `volatile`		
Código con `synchronized`		
Código con clases atómicas		

Código	**Total incrementos**	**Tiempos transcurrido**
Código original	1725099	0.033
Código volatile	1121190	0.086

Código synchronized	4000000	0.176
Código Atómicas	4000000	0.112

Primero nos centraremos en la velocidad en lo que encontramos que el más rápido es el código original, esto es debido es que a ser operaciones muy sencillo la tarea de trabajar en paralelo es más un coste que un beneficio, entrando más en paralelo se clasifica como Volatile > Atómicas > Synchronized, aunque volátiles es más rápido solo puede resolver problemas de atomicidad en cambio tanto Atomicas como Synchronized si que resuelven ambos problemas, el ganador en Atómico puesto que no es tan versátil para los problemas en general en este caso nos viene perfecto ya que cubrimos todas las necesidades y nos ahorramos una Clase.