EI1024/MT1024 "Programación Concurrente y Paralela" 2021–22	Entregable para Laboratorio
Nombre y apellidos (1):	
Nombre y apellidos (2):	la03_g
Tiempo empleado para tareas en casa en formato *hh:mm* (obligatorio):	

Tema 05. El Problema de la Visibilidad en Java

Tema 06. 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                 tope;
  CuentaIncrementos   c;

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

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

// ============================================================
class EjemploCuentaIncrementos {
// ============================================================

  // ------------------------------------------------------------
```

```
  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 v[] = new MiHebra[ numHebras ];
    CuentaIncrementos c = new CuentaIncrementos();
    for( int i = 0; i < numHebras; i++ ) {
      v[ i ] = new MiHebra( 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 );
  }
}
```

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

Crea un vector de hebras y a cada hebra le pasa un objeto para que realice un incremento sobre él
Cada hebra realizaría 1000000 de incrementos por lo que el resultado debería ser 4000000

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?

Muestra un número inferior al que debería de mostrar.

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

No, porque al pasar el mismo objeto y acceder todas las hebras al mismo tiempo cuando realizan el incremento hay mas hebras al tiempo realizando incrementos por lo que tiene sentido que el resultado sea inferior ya que no hay coordinación entre ellos,

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?

No.

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`.

```
class CuentaIncrementos {

  int numIncrementos = 0;

  synchronized void incrementaNumIncrementos() {
    numIncrementos++;
  }

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

}
```

Con synchronized si se soluciona el problema, esto se debe a que cuando una hebra accede cierra el cerrojo de acceso y lo libera cuando ha terminado, entonces cuando accede la hebra que estaba en espera ya tiene el objeto con los incrementos de la hebra anterior por lo que perjudica el tiempo de ejecución pero consigue el resultado esperado.

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.

```
  AtomicInteger numIncrementos = new AtomicInteger(0);

   void incrementaNumIncrementos() {
    numIncrementos.addAndGet(1);
   }

   int dameNumIncrementos() {
    return( numIncrementos.get());
   }

 }
```

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	3906262 en 0,021s
Código con `volatile`	2268566 en 0,034s
Código con `synchronized`	4000000 en 0,133s
Código con clases atómicas	4000000 en 0,096s

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:

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

  // --------------------------------------------------------------------------
  public static void main( String args[] ) {
    int     numHebras;
    long    t1, t2;
    double  ts, tc, tb, td;
    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
    // ....
    t2 = System.nanoTime();
    tc = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.println( "Tiempo paralela ciclica (seg.):              " + tc );
    System.out.println( "Incremento paralela ciclica:                 " + ... );
    //
    // 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 );
  }
}
```

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

200000033, 200000039, 200000051, 200000069, 200000161, 200000183, 200000201 200000209

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.

```
class MiHebraPrimoDistCiclica extends Thread {

  int id;
  int numHebras;
  long[] n;

  public MiHebraPrimoDistCiclica(int id, int numHebras, long vector[]){
    this.id = id;
    this.numHebras = numHebras;
    this.n = vector;
  }
  public void run() {
    for (int i = id; i < n.length; i += numHebras) {
      if( EjemploMuestraPrimosEnVector2a.esPrimo(n[i])) {
        System.out.println( "  Encontrado primo: " + n[i]);
      }
    }
  }
}



MiHebraPrimoDistCiclica[] hebras = new MiHebraPrimoDistCiclica[numHebras];
for (int i = 0; i < hebras.length; i++) {
  hebras[i] = new MiHebraPrimoDistCiclica(i, numHebras, vectorNumeros);
  hebras[i].start();
  try {
    hebras[i].join();
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
}
```

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, 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.

```
class MiHebraPrimoDistPorBloques extends Thread {

  int id;
  int numHebras;
  long[] n;

  public MiHebraPrimoDistPorBloques(int id, int numHebras, long vector[]){
    this.id = id;
    this.numHebras = numHebras;
    this.n = vector;
  }
  public void run() {
    for (int i = n.length * id / numHebras; i < n.length*(id+1)/numHebras; i++) {
      if( EjemploMuestraPrimosEnVector2a.esPrimo(n[i])) {
        System.out.println( "  Encontrado primo: " + n[i]);
      }
    }
  }
}

  MiHebraPrimoDistPorBloques[] misHebras = new MiHebraPrimoDistPorBloques
[numHebras];
  for (int i = 0; i < misHebras.length; i++) {
    misHebras[i] = new MiHebraPrimoDistPorBloques(i, numHebras, vectorNumeros);
    misHebras[i].start();
    try {
      misHebras[i].join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
```

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.

```
class MiHebraPrimoDistDinamica extends Thread {

  AtomicInteger indice;
  long[] n;

  public MiHebraPrimoDistDinamica(AtomicInteger indice, long vector[]){
    this.indice = indice;
    this.n = vector;
  }
  public void run() {
    for (int i = indice.get(); i < n.length; i++){
      if(EjemploMuestraPrimosEnVector2a.esPrimo(n[indice.get()])){
        System.out.println( "  Encontrado primo: " + n[indice.get()]);
      }
      indice.addAndGet(1);
    }
  }
}

MiHebraPrimoDistDinamica[] hebras1 = new MiHebraPrimoDistDinamica[numHebras];
    for(int i = 0; i < hebras1.length; i++){
      hebras1[i] = new MiHebraPrimoDistDinamica(indice, vectorNumeros);
      hebras1[i].start();
      try {
        hebras1[i].join();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
```

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	15,786s	—		—
Paralela con distribución cíclica	14,521s	8,02%		
Paralela con distribución por bloques	14,560s	7,77%		
Paralela con distribución dinámica	14,054s	10,97%		

2.6) 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 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	15,629s	—		—
Paralela con distribución cíclica	14,773s	5,47%		
Paralela con distribución por bloques	14,551s	6,89%		
Paralela con distribución dinámica	13,933s	10,85%		

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

Código	Total incrementos	Tiempo transcurrido (seg.)
Código original	3 989 376	0,019s
Código con `volatile`	2 070 163	0,022s
Código con `synchronized`	4 000 000	0,166s
Código con clases atómicas	4 000 000	0,074s