EI1024/MT1024 "Programación Concurrente y Paralela" 2022–23	Entregable para Laboratorio
Nombre y apellidos (1):	
Nombre y apellidos (2):	la04_g
Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio):	

Tema 06. El Problema de la Atomicidad en Java

Tema 07. *Thread Pools* e Interfaces Gráficas en Java

1 Se desea calcular el número π mediante integración numérica de la siguiente función:

$$\pi = \int_0^1 \frac{4}{1+x^2} dx.$$

Este método no es el más rápido para calcular el número π, pero sí uno de las más simples. Consiste en calcular la anterior integral mediante una aproximación numérica basada en el cálculo y acumulación del área de numerosos rectángulos pequeños.

Uno de los parámetros más importantes es el número de rectángulos cuya área se va a sumar. En este caso, este parámetro será pasado en la línea de argumentos, después del número de hebras.

El siguiente programa realiza el cálculo de forma secuencial. Con vistas a facilitar el desarrollo posterior de la versión paralela, este código secuencial contiene un fragmento de código comentado, además de la declaración e inicialización de la variables `numHebras` y `numRectangulos`. Ambas partes no son útiles en la versión secuencial, pero, la inclusión de este fragmento de código simplifica el desarrollo de la versión paralela.

```
/*
// ======================================================================
class Acumula {
// ======================================================================
  double suma;

  // ----------------------------------------------------------------------
  Acumula() {
    // ...
  }

  // ----------------------------------------------------------------------
  void acumulaDato( double dato ) {
    // ...
  }

  // ----------------------------------------------------------------------
  double dameDato() {
    // ...
  }
}

// ======================================================================
class MiHebraMultAcumulaciones extends Thread {
```

```
// ============================================================================
  int       miId, numHebras;
  long      numRectangulos;
  Acumula   a;

  // ----------------------------------------------------------------------
  MiHebraMultAcumulaciones( int miId, int numHebras, long numRectangulos,
                            Acumula a ) {
    // ...
  }

  // ----------------------------------------------------------------------
  public void run() {
    // ...
  }
}

// ============================================================================
class MiHebraUnaAcumulacion extends Thread {
// ============================================================================
// ...
}

// ============================================================================
class MiHebraMultAcumulacionAtomic extends Thread {
// ============================================================================
// ...
}

// ============================================================================
class MiHebraUnaAcumulacionAtomic extends Thread {
// ============================================================================
// ...
}

*/

// ============================================================================
class EjemploNumeroPI {
// ============================================================================

  // ----------------------------------------------------------------------
  public static void main( String args[] ) {
    long                    numRectangulos;
    double                  baseRectangulo, x, suma, pi;
    int                     numHebras;
    long                    t1, t2;
    double                  tSec, tPar;
    // Acumula                 a;
    // MiHebraMultAcumulaciones  vt[];

    // Comprobacion de los argumentos de entrada.
    if( args.length != 2 ) {
      System.out.println( "ERROR: numero de argumentos incorrecto." );
      System.out.println( "Uso: java programa <numHebras> <numRectangulos>" );
      System.exit( -1 );
    }
    try {
      numHebras      = Integer.parseInt( args[ 0 ] );
      numRectangulos = Long.parseLong( args[ 1 ] );
    } catch( NumberFormatException ex ) {
```

```
      numHebras       = -1;
      numRectangulos = -1;
      System.out.println( "ERROR: Numeros de entrada incorrectos." );
      System.exit( -1 );
    }
    System.out.println();
    System.out.println( "Calculo del numero PI mediante integracion." );
    //
    // Calculo del numero PI de forma secuencial.
    //
    System.out.println();
    System.out.println( "Comienzo del calculo secuencial." );
    t1 = System.nanoTime();
    baseRectangulo = 1.0 / ( ( double ) numRectangulos );
    suma           = 0.0;
    for( long i = 0; i < numRectangulos; i++ ) {
      x = baseRectangulo * ( ( ( double ) i ) + 0.5 );
      suma += f( x );
    }
    pi = baseRectangulo * suma;
    t2 = System.nanoTime();
    tSec = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.println( "Version secuencial. Numero PI: " + pi );
    System.out.println( "Tiempo secuencial (s.):        " + tSec );
/*
    //
    // Calculo del numero PI de forma paralela:
    // Multiples acumulaciones por hebra.
    //
    System.out.println();
    System.out.print( "Comienzo del calculo paralelo: " );
    System.out.println( "Multiples acumulaciones por hebra." );
    t1 = System.nanoTime();
    // ...
    t2 = System.nanoTime();
    tPar = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.println( "Calculo del numero PI:    " + pi );
    System.out.println( "Tiempo ejecucion (s.):    " + tPar );
    System.out.println( "Incremento velocidad :    " + ... );
    //
    // Calculo del numero PI de forma paralela:
    // Una acumulacion por hebra.
    // ...
    //
    // Calculo del numero PI de forma paralela:
    // Multiples acumulaciones por hebra (Atomica)
    // ...
    //
    // Calculo del numero PI de forma paralela:
    // Una acumulacion por hebra (Atomica).
    // ...
*/
    System.out.println();
    System.out.println( "Fin de programa." );
  }

  // ---------------------------------------------------------------------
  static double f( double x ) {
    return ( 4.0/( 1.0 + x*x ) );
  }
}
```

1.1) Estudia el código anterior y paralelízalo mediante el uso de hebras con una `distribución cíclica`. Utiliza un objeto de la clase `Acumula` para almacenar el resultado.

En esta versión paralela cada vez que las hebras calculan el área de un rectángulo, deben acumularlo sobre el objeto compartido de la clase `Acumula`. Para un correcto manejo del programa, hay que asegurar que el acceso al objeto compartido sea *thread-safe*.

No crees un nuevo programa. Haz que esta implementación paralela se ejecute a continuación de la versión secuencial dentro del mismo programa. Ello permitirá obtener los tiempos y los incrementos de velocidad de forma más rápida y automatizada.

Escribe a continuación la parte de tu código que realiza esta tarea: la definición de la clase `MiHebraMultAcumulaciones` y el código incluido en el programa principal que permite gestionar los objetos de esta clase.

1.2) Modifica el programa anterior, de modo que en la versión paralela las hebras acumulen el área que han calculado en una variable local (`sumaL`), antes de sumarla al objeto compartido.

No crees un nuevo programa. Haz que esta implementación paralela se ejecute a continuación de la versión paralela desarrollada en el apartado anterior. Ello permitirá obtener los tiempos y los incrementos de velocidad de forma más rápida y automatizada.

Escribe a continuación la parte de tu código que realiza esta tarea: la definición de la clase `MiHebraUnaAcumulacion` y el código incluido en el programa principal que permite gestionar los objetos de esta clase.

1.3) En las **DOS** versiones paralelas anteriores se ha utilizado un objeto de la clase `Acumula` que permite acumular números reales con precisión doble de forma atómica, pero también se podría realizar empleando clases y operadores atómicos, como `DoubleAdder`.

Para ello, define las clases `MiHebraMultAcumulacionesAtomic` y `MiHebraUnaAcumulacionAtomic`. Estas clases deben manejar un objeto de la clase `DoubleAdder` y acumular los valores con el método `add`, mientras que el valor final se obtiene con el método `sum`.

Escribe a continuación los cambios realizados en el código.

Ten en cuenta que la mejor opción sería **eliminar completamente** la clase `Acumula` y en su lugar utilizar la clase atómica.

2 Se dispone de una interfaz gráfica con un cuadro de texto y dos botones denominados `Comienza secuencia` y `Cancela secuencia`. Por el momento, la interfaz no hace nada cuando el usuario realiza alguna acción sobre los botones o sobre el cuadro de texto.

La interfaz está definida por el siguiente código:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

/*
// ============================================================================
class ZonaIntercambio1a {
// ============================================================================
  // ...

  // ----------------------------------------------------------------------------
  void setTiempo( ... ) {
    // ...
  }

  // ----------------------------------------------------------------------------
  long getTiempo( ... ) {
    // ...
  }
}
*/

// ============================================================================
public class GUISecuenciaPrimos1a {
// ============================================================================
  JFrame      container;
  JPanel      jpanel;
  JTextField  txfMensajes;
  JButton     btnComienzaSecuencia, btnCancelaSecuencia;
  JSlider     sldEspera;
//  HebraCalculadora1a  t; // Ejercicio 2.2
//  ZonaIntercambio1a   z; // Ejercicio 2.3

  // ----------------------------------------------------------------------------
  public static void main( String args[] ) {
    GUISecuenciaPrimos1a gui = new GUISecuenciaPrimos1a();
    SwingUtilities.invokeLater(new Runnable(){
      public void run(){
        gui.go();
      }
    });
  }

  // ----------------------------------------------------------------------------
  public void go() {
    // Constantes.
    final int valorMaximo = 1000;
    final int valorMedio  = 500;

    // Variables.
    JPanel  tempPanel;

    // Crea el JFrame principal.
    container = new JFrame( "GUI Secuencia de Primos 1a" );
```

```
// Consigue el panel principal del Frame "container".
jpanel = ( JPanel ) container.getContentPane();
jpanel.setLayout( new GridLayout( 3, 1 ) );

// Crea e inserta la etiqueta y el campo de texto para los mensajes.
txfMensajes = new JTextField( 20 );
txfMensajes.setEditable( false );
tempPanel = new JPanel();
tempPanel.setLayout( new FlowLayout() );
tempPanel.add( new JLabel( "Secuencia: " ) );
tempPanel.add( txfMensajes );
jpanel.add( tempPanel );

// Crea e inserta los botones de Comienza secuencia y Cancela secuencia.
btnComienzaSecuencia = new JButton( "Comienza secuencia" );
btnCancelaSecuencia = new JButton( "Cancela secuencia" );
tempPanel = new JPanel();
tempPanel.setLayout( new FlowLayout() );
tempPanel.add( btnComienzaSecuencia );
tempPanel.add( btnCancelaSecuencia );
jpanel.add( tempPanel );

// Crea e inserta el slider para controlar el tiempo de espera.
sldEspera = new JSlider( JSlider.HORIZONTAL, 0, valorMaximo , valorMedio );
tempPanel = new JPanel();
tempPanel.setLayout( new BorderLayout() );
tempPanel.add( new JLabel( "Tiempo de espera: " ) );
tempPanel.add( sldEspera );
jpanel.add( tempPanel );

// Activa inicialmente los 2 botones.
btnComienzaSecuencia.setEnabled( true );
btnCancelaSecuencia.setEnabled( true );

// Anyade codigo para procesar el evento del boton de Comienza secuencia.
btnComienzaSecuencia.addActionListener( new ActionListener() {
    public void actionPerformed( ActionEvent e ) {
      // ...
    }
} );

// Anyade codigo para procesar el evento del boton de Cancela secuencia.
btnCancelaSecuencia.addActionListener( new ActionListener() {
    public void actionPerformed( ActionEvent e ) {
      // ...
    }
} );

// Anyade codigo para procesar el evento del slider " Espera " .
sldEspera.addChangeListener( new ChangeListener() {
  public void stateChanged( ChangeEvent e ) {
    JSlider sl = ( JSlider ) e.getSource();
    if ( ! sl.getValueIsAdjusting() ) {
      long tiempoMilisegundos = ( long ) sl.getValue();
      System.out.println( "JSlider value = " + tiempoMilisegundos );
      // ...
    }
  }
} );
```

```
    // Fija caracteristicas del container.
    container.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    container.pack();
    container.setResizable( false );
    container.setVisible( true );

    System.out.println( "%End of routine: go.\n" );
  }

  // ---------------------------------------------------------------------
  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) Modifica la interfaz gráfica para que los botones `Comienza secuencia` y `Cancela secuencia` se activen y desactiven (`setEnabled`) de acuerdo a la lógica de funcionamiento de la aplicación :

- Inicialmente el botón `Comienza secuencia` debe estar activado y el botón `Cancela secuencia` debe estar desactivado (modificar método `go`).
- Cuando se presione el botón `Comienza secuencia`, éste se desactiva y se activa el botón `Cancela secuencia` (modificar `ActionListener` del primero).
- Cuando se presione el botón `Cancela secuencia`, éste se desactiva y se activa el botón `Comienza secuencia` (modificar `ActionListener` del primero).

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

2.2) Modifica la anterior interfaz de tal forma que en cuanto el usuario pulse el botón `Comienza secuencia` el programa muestre la secuencia de números primos (comenzando por el 2, 3, 5, 7, 11, etc.) en el cuadro de texto. Para ello, la hebra *event-dispatching* deberá crear una hebra trabajadora (`t`) en la que delegará dicho trabajo.

En cuanto el usuario pulse el botón `Cancela secuencia`, la generación de la secuencia debe terminar. Para detener la hebra, se fijará un valor especial en un atributo de la hebra (`fin`), cuyo valor será revisado por ésta cada vez que se genere un nuevo número primo.

Además, cada vez que se presione el botón `Comienza secuencia` la secuencia se inicia desde el principio.

Seguidamente se muestra la estructura del cuerpo de la hebra.

```
// Estructura del cuerpo de la hebra
long i = 1L;
while ( ! fin ) {
  if ( esPrimo ( i ) ) {
    // imprime ( i );
  }
  i++;
}
```

Una hebra trabajadora no puede llamar directamente a ningún método de ningún objeto gráfico, ya que éstos sólo pueden ser manejados por la *event-dispatching*. Por tanto, cuando la hebra trabajadora desee realizar alguna escritura sobre el cuadro de texto (`txfMensajes`), debe utilizar los métodos `invokeAndWait` o `invokeLater`, que indican a la *event-dispatching* que labores debe realizar.

Estos métodos ejecutan un objeto Runnable que reciben como parámetro de entrada, el primer método bloquea a la hebras hasta que la *event-dispatching* finaliza, por lo que es necesario gestionar dos excepciones, mientras que el segundo no bloquea a la hebra.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de la clase `HebraTrabajadora` y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

2.3) Modifica el programa anterior de tal forma que se muestre en pantalla una barra de deslizamiento horizontal (`JSlider`) con la que el usuario pueda determinar la velocidad de generación de números primos (ver código inicial).
Si la barra está en un extremo, la hebra deberá generar números primos intercalando una demora (método `sleep`) de un segundo **tras la impresión en el cuadro de texto**. Si la barra está en el otro extremo, la hebra deberá generar números primos sin ninguna demora.
Se recomienda definir y emplear una nueva clase denominada `ZonaIntercambio`, a través de la cual se comunicarán la hebra gráfica y la hebra calculadora. La hebra gráfica escribirá valores en un objeto de dicha clase y la hebra calculadora tomará valores de dicho objeto.
Por último comentar que el tiempo de espera se expresa en milisegundos, y que el valor inicial definido en el código es 500.
Escribe a continuación la parte de tu código que realiza tal tarea: la definición de la clase `ZonaIntercambio`, el código para la gestión de la barra de desplazamiento, y los cambios en la clase `HebraTrabajadora`.

2.4) Se desea sustituir la barra de deslizamiento por dos botones adicionales: Un botón añadirá 0,1 segundos al tiempo de espera, mientras que el otro botón restará 0,1 segundos al tiempo de espera.

No hagas ninguna implementación, pero responde a la siguiente pregunta. ¿Se podría realizar dicha modificación sólo con el operador `volatile` o habría que recurrir al modificador `synchronized`? Justifica la respuesta.

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

3 Este ejercicio es una continuación del ejercicio 1.

3.1) Completa la siguiente tabla para 500 000 000 de rectángulos. Obtén los resultados para 4 hebras en el ordenador del aula. Obtén los resultados para 16 hebras en patan. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

Los resultados utilizando la clase atómica son **optativos**. Para obtenerlos, debes sustituir el objeto de la clase `Acumula` por un objeto de la clase atómica propia de Java 8, y utilizar sus métodos en la actualización.

Justifica los resultados obtenidos.

Ejecución con 500 000 000 rectángulos				
	4 hebras		16 hebras	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela: Múltiples acumul.				
Paralela: Una única acumul.				
Paralela: Múltiples acumul. (clase atom.)				
Paralela: Una única acumul. (clase atom.)				

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