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.

```
package e4;/*
// ============================================================================
class Acumula {
// ============================================================================
  double  suma;

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

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

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

// ============================================================================
class MiHebraMultAcumulaciones1a extends Thread {
// ============================================================================
  int      miId, numHebras;
  long     numRectangulos;
  Acumula  a;

  // --------------------------------------------------------------------------
  MiHebraMultAcumulaciones1a( int miId, int numHebras, long numRectangulos,
                              Acumula a ) {
    // ...
  }
```

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

// ===========================================================================
class MiHebraUnaAcumulacion1b extends Thread {
// ===========================================================================
// ...
}

// ===========================================================================
class MiHebraMultAcumulacionAtomica1c extends Thread {
// ===========================================================================
// ...
}

// ===========================================================================
class MiHebraUnaAcumulacionAtomica1d extends Thread {
// ===========================================================================
// ...
}

*/

import java.util.concurrent.atomic.DoubleAdder;

class Acumula {
  double sum;

  public Acumula(){
    this.sum = 0;
  }

  synchronized void acumulaNum(double num){
    sum += num;
  }

  synchronized double dameNum() {
    return sum;
  }
}

class MiHebraMultAcumulaciones1 extends Thread{
  // ===========================================================================
  int miId, numHebras;
  long numRec;
  Acumula a;
  double baseRec, x;

  // ---------------------------------------------------------------------
  MiHebraMultAcumulaciones1(int miId, int numHebras, long numRec, Acumula a) {
    this.miId = miId;
    this.numHebras = numHebras;
```

```
    this.numRec = numRec;
    this.a = a;
  }

  // ---------------------------------------------------------------------
  public void run() {
    baseRec = 1.0 / ((double) numRec);
    for (long i = miId; i < numRec; i += numHebras) {
      x = baseRec * (((double) i) + 0.5);
      a.acumulaNum(EjemploNumeroPI1a.f(x));
    }
  }
}

class MiHebraUnaAcumulaciones1_2 extends Thread{
  // =====================================================================
  int miId, numHebras;
  long numRec;
  Acumula a;
  double baseRec, x;

  // ---------------------------------------------------------------------
  MiHebraUnaAcumulaciones1_2(int miId, int numHebras, long numRec, Acumula a) {
    this.miId = miId;
    this.numHebras = numHebras;
    this.numRec = numRec;
    this.a = a;
  }

  // ---------------------------------------------------------------------
  public void run() {
    baseRec = 1.0 / ((double) numRec);
    double sumaL = 0.0;
    for (long i = miId; i < numRec; i += numHebras) {
      x = baseRec * (((double) i) + 0.5);
      sumaL += EjemploNumeroPI1a.f(x);
    }
    a.acumulaNum(sumaL);
  }
}

class AcumulaAtomica {
  DoubleAdder sum;

  // ---------------------------------------------------------------------
  public AcumulaAtomica() {
    this.sum = new DoubleAdder();
  }

  // ---------------------------------------------------------------------
  void acumulaDato(double num) {
    this.sum.add(num);
  }

  // ---------------------------------------------------------------------
  double dameDato() {
```

```
    return sum.doubleValue();
  }
}

class MiHebraMultAcumulacionAtomica1_3 extends Thread {
  // ============================================================================
  int miId, numHebras;
  long numRec;
  AcumulaAtomica a;
  double baseRec, x;

  MiHebraMultAcumulacionAtomica1_3(int miId, int numHebras, long numRec, AcumulaAtomica
a) {
    this.miId = miId;
    this.numHebras = numHebras;
    this.numRec = numRec;
    this.a = a;
  }

  // ----------------------------------------------------------------------
  public void run() {
    baseRec = 1.0 / ((double) numRec);
    for (long i = miId; i < numRec; i += numHebras) {
      x = baseRec * (((double) i) + 0.5);
      a.acumulaDato(EjemploNumeroPI1a.f(x));
    }

  }
}

class MiHebraUnaAcumulacionAtomica1_3 extends Thread {
  int miId, numHebras;
  long numRec;
  AcumulaAtomica a;
  double baseRec, x;

  MiHebraUnaAcumulacionAtomica1_3(int miId, int numHebras, long numRec, AcumulaAtomica
a) {
    this.miId = miId;
    this.numHebras = numHebras;
    this.numRec = numRec;
    this.a = a;
  }

  // ----------------------------------------------------------------------
  public void run() {
    baseRec = 1.0 / ((double) numRec);
    double suma = 0.0;
    for (long i = miId; i < numRec; i += numHebras) {
      x = baseRec * (((double) i) + 0.5);
      suma+=EjemploNumeroPI1a.f(x);
    }
    a.acumulaDato(suma);
  }
}
```

```
// ======================================================================
class EjemploNumeroPI1a {
// ======================================================================

  // ----------------------------------------------------------------------
  public static void main( String args[] ) {
    long                        numRectangulos;
    double                      baseRectangulo, x, suma, pi;
    int                         numHebras;
    long                        t1, t2;
    double                      tSec, tPar;
    Acumula                     a;
    AcumulaAtomica              aAtomica;
    //MiHebraMultAcumulaciones1  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 );
    }
*/
    numHebras      = 4;
    numRectangulos = 500000000;

    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();

a = new Acumula();
MiHebraMultAcumulaciones1[] h1_1 = new MiHebraMultAcumulaciones1[numHebras];

for (int i = 0; i < numHebras; i++) {
  h1_1[i] = new MiHebraMultAcumulaciones1(i, numHebras, numRectangulos, a);
  h1_1[i].start();
}
try{
  for (int i = 0; i < numHebras; i++) {
    h1_1[i].join();
  }
}catch (InterruptedException ex){
  ex.printStackTrace();
}

pi = baseRectangulo * a.dameNum();
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 :   " + tSec/tPar );

//
// Calculo del numero PI de forma paralela:
// Una acumulacion por hebra.
//
System.out.println();
System.out.print( "Comienzo del calculo paralelo: " );
System.out.println( "Una acumulacion por hebra." );
t1 = System.nanoTime();

a = new Acumula();
MiHebraUnaAcumulaciones1_2[] h1_2 = new MiHebraUnaAcumulaciones1_2[numHebras];

for (int i = 0; i < numHebras; i++) {
  h1_2[i] = new MiHebraUnaAcumulaciones1_2(i, numHebras, numRectangulos, a);
  h1_2[i].start();
}
try{
  for (int i = 0; i < numHebras; i++) {
    h1_2[i].join();
  }
}catch (InterruptedException ex){
  ex.printStackTrace();
}

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 :   " + tSec/tPar );

    //
    // Calculo del numero PI de forma paralela:
    // Multiples acumulaciones por hebra (Atomica)
    //
    System.out.println();
    System.out.print( "Comienzo del calculo paralelo: " );
    System.out.println( "Multiples acumulaciones por hebra (At)." );
    t1 = System.nanoTime();

    aAtomica = new AcumulaAtomica();
    MiHebraMultAcumulacionAtomica1_3[] h1_3 = new
MiHebraMultAcumulacionAtomica1_3[numHebras];

    for (int i = 0; i < numHebras; i++) {
      h1_3[i] = new MiHebraMultAcumulacionAtomica1_3(i, numHebras, numRectangulos,
aAtomica);
      h1_3[i].start();
    }
    try{
      for (int i = 0; i < numHebras; i++) {
        h1_3[i].join();
      }
    }catch (InterruptedException ex){
      ex.printStackTrace();
    }

    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 :   " + tSec/tPar );

    //
    // Calculo del numero PI de forma paralela:
    // Una acumulacion por hebra (Atomica).
    //
    System.out.println();
    System.out.print( "Comienzo del calculo paralelo: " );
    System.out.println( "Una acumulacion por hebra (At)." );
    t1 = System.nanoTime();

    aAtomica = new AcumulaAtomica();
    MiHebraUnaAcumulacionAtomica1_3[] h1_4 = new
MiHebraUnaAcumulacionAtomica1_3[numHebras];

    for (int i = 0; i < numHebras; i++) {
      h1_4[i] = new MiHebraUnaAcumulacionAtomica1_3(i, numHebras, numRectangulos,
aAtomica);
      h1_4[i].start();
    }
    try{
```

```
      for (int i = 0; i < numHebras; i++) {
        h1_4[i].join();
      }
    }catch (InterruptedException ex){
      ex.printStackTrace();
    }

    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 :    " + tSec/tPar );

    System.out.println();
    System.out.println( "Fin de programa." );
  }

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

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.

```
package e4;

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

class ZonaIntercambio1a {
  // ============================================================================
  volatile long tiempo = 250;

  // ----------------------- ---------------------------------------------------
  void setTiempo( long tiempo ) {
    this.tiempo = tiempo;
  }

  // ----------------------------------------------------------------------------
  long getTiempo() {
    return tiempo;
  }

}

class HebraTrabajadora2_2 extends Thread {
  JTextField txfMensajes;
  volatile boolean fin = false;
  public HebraTrabajadora2_2(JTextField txfMensajes){
    this.txfMensajes = txfMensajes;
  }
  public void acabar(boolean fin){
    this.fin = fin;
  }

  @Override
  public void run() {
    long i = 1L;
    while ( !fin ) {
      if ( GUISecuenciaPrimos1a.esPrimo ( i ) ) {
        final long acaba = i;
        SwingUtilities.invokeLater(new Runnable() {
          @Override
          public void run() {
            txfMensajes.setText(String.valueOf(acaba));
          }
        });
      }
      i ++;
    }

  }
}
```

```
class HebraTrabajadora2_3 extends Thread {
  JTextField txfMensaje;
  volatile boolean fin = false;
  ZonaIntercambio1a zonaIntercambio1a;
  public HebraTrabajadora2_3(JTextField txfMensaje, ZonaIntercambio1a zona){
    this.txfMensaje = txfMensaje;
    this.zonaIntercambio1a = zona;

  }
  public void acabar(boolean fin){
    this.fin = fin;
  }

  @Override
  public void run() {
    long i = 1L;
    while ( !fin ) {

      if ( GUISecuenciaPrimos1a.esPrimo ( i ) ) {
        final long acaba = i;
        SwingUtilities.invokeLater(new Runnable() {
          @Override
          public void run() {
            txfMensaje.setText(String.valueOf(acaba));
          }
        });
        try {
          sleep(zonaIntercambio1a.getTiempo());
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
      i ++;
    }

  }
}

// ====================================================================
public class GUISecuenciaPrimos1a {
// ====================================================================
  JFrame      container;
  JPanel      jpanel;
  JTextField  txfMensajes;
  JButton     btnComienzaSecuencia, btnCancelaSecuencia;
  JSlider     sldEspera;
  HebraTrabajadora2_2 t;// Ejercicio 2.2
  HebraTrabajadora2_3 t2; // Ejercicio 2.3
  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( false );
  z = new ZonaIntercambio1a();

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

```
          // t = new HebraTrabajadora2_2(txfMensajes);
          // t.start();
          t2 = new HebraTrabajadora2_3(txfMensajes, z);
          t2.start();
        }
    } );

    // Anyade codigo para procesar el evento del boton de Cancela secuencia.
    btnCancelaSecuencia.addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent e ) {
          btnComienzaSecuencia.setEnabled(true);
          btnCancelaSecuencia.setEnabled(false);
          //t.acabar(true);
          t2.acabar(true);
        }
    } );

    // 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 );
          z.setTiempo(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 );
  }
}
```

2.4) 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.

Al solo tener una hebra que está modificando la variable con volatile sería suficiente.

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 núcleos en el ordenador del aula. Obtén los resultados para 16 núcleos 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 núcleos		16 núcleos	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela: Múltiples acumul.				
Paralela: Una única acumul.				
Paralela: Múltiples acumul. (clase atom.)				
Paralela: Una única acumul. (clase atom.)				

	4 núcleos		**16 nucleos**	
	Tiempo	**Incremento**	**Tiempo**	**Incremento**
Secuencial	12.131	-	2.016	-
Múltiples acumulaciones	22.259	0.545	135.24	0.014
Única acumulación	3.525	3.442	0.272	7.396
Atómica - Múltiples acumulaciones	4.453	2.272	0.701	2.874
Atómica - Única acumulación	3.347	3.624	0.026	7.621

Se observa que cuando las acumulaciones se hacen de forma local en las hebras y después se envía el resultado el tiempo es mucho menor que cuando se envía en cada ejecución.

```
Running on master host ep03.local
Time is Thu May 19 16:26:49 CEST 2022
Directory is /home/al387320/Practica_e4
Procsfile: /var/spool/torque/aux//20864.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
Running on host ep03.local
Time is Thu May 19 16:26:49 CEST 2022
Directory is /home/al387320/Practica_e4

Calculo del numero PI mediante integracion.

Comienzo del calculo secuencial.
Version secuencial. Numero PI: 3.141592653589814
Tiempo secuencial (s.):        2.01678201

Comienzo del calculo paralelo: Multiples acumulaciones por hebra.
Calculo del numero PI:   3.1415926535893064
Tiempo ejecucion (s.):   135.24206603
Incremento velocidad :   0.014912386871941371

Comienzo del calculo paralelo: Una acumulacion por hebra.
Calculo del numero PI:   3.1415926535893064
Tiempo ejecucion (s.):   0.272681651
Incremento velocidad :   7.396104587910097

Comienzo del calculo paralelo: Multiples acumulaciones por hebra (At).
Calculo del numero PI:   3.1415926535893064
Tiempo ejecucion (s.):   0.701558929
Incremento velocidad :   2.8747150476364327

Comienzo del calculo paralelo: Una acumulacion por hebra (At).
Calculo del numero PI:   3.1415926535893064
Tiempo ejecucion (s.):   0.264610831
Incremento velocidad :   7.621691078850812

Fin de programa.
```