

EI1024/MT1024 “Programación Concurrente y Paralela” 2021–22 Nombre y apellidos (1): Nombre y apellidos (2): Tiempo empleado para tareas en casa en formato <i>hh:mm</i> (obligatorio):	Entregable para Laboratorio la04_g
--	---

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 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 MiHebraMultAcumulacionAtomic1c extends Thread {
// =====
// ...
}

// =====
class MiHebraUnaAcumulacionAtomic1d extends Thread {
// =====
// ...
}

*/

// =====
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;
    // MiHebraMultAcumulaciones1b 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 `MiHebraMultAcumulaciones1` 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 `MiHebraUnaAcumulacion1` 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.

¿Se puede arreglar empleando clases y operadores atómicos, como `DoubleAdder` ?

Arréglalo si es posible. Para ello, crea otra copia del código original y modifícala creando las clases `MiHebraMultAcumulacionesAtomic1` y `MiHebraUnaAcumulacionAtomic1`.

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( ... ) {
// ...
}

// -----
void 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 características 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.

