

EI1024/MT1024 “Programación Concurrente y Paralela” 2025–26 Nombre y apellidos (1): ..... Vicente Ventura Ninot Nombre y apellidos (2): ..... Víctor-Nesta Reig Buendía Tiempo empleado para tareas en casa en formato <i>h:mm</i> (obligatorio): ..... 1:30	Entregable para Laboratorio  la04_g
--	---

## Tema 05. El Problema de la Atomicidad en Java

## Tema 06. *Thread Pools* e Interfaces Gráficas en Java

**1** Se desea calcular el número  $\pi$  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  $\pi$ , 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 ] );
        if( ( numHebras <= 0 ) || ( numRectangulos <= 0 ) ) {

```

```

        System.err.print( "Uso: [ java programa <numHebras> <n> ] " );
        System.err.println( "donde ( numHebras > 0 ) y ( numRectangulos > 0 )" );
        System.exit( -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( "Inicio 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( "Inicio 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." );
}

```



- 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.



- 2** Se dispone de una interfaz gráfica con un cuadro de texto y dos botones denominados **Inicia 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 ZonaIntercambio {
// =====
// ...

    public ZonaIntercambio ( ... ) {
// ...
    }

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

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

// =====
public class GUISecuenciaPrimos {
// =====
    JFrame      container;
    JPanel      jpanel;
    JTextField  txfMensajes;
    JButton     btnIniciaSecuencia , btnCancelaSecuencia;
    JSlider     sldEspera;
// HebraTrabajadora  t; // Ejercicio 2.2
// ZonaIntercambio  z; // Ejercicio 2.3

// =====
    public static void main( String args[] ) {
        GUISecuenciaPrimos gui = new GUISecuenciaPrimos();
        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 " );

// 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 Inicia secuencia y Cancela secuencia.
btnIniciaSecuencia = new JButton( "Inicia secuencia" );
btnCancelaSecuencia = new JButton( "Cancela secuencia" );
tempPanel = new JPanel();
tempPanel.setLayout( new FlowLayout() );
tempPanel.add( btnIniciaSecuencia );
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.
btnIniciaSecuencia.setEnabled( true );
btnCancelaSecuencia.setEnabled( true );

// Anyade codigo para procesar el evento del boton de Inicia secuencia.
btnIniciaSecuencia.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 tiempoEnMilisegundos = ( long ) sl.getValue();

```



```

        System.out.println( "JSlider value = " + tiempoEnMilisegundos );
        // ...
    }
}
} );

// 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 **Inicia secuencia** y **Cancela secuencia** se activen y desactiven (**setEnabled**) de acuerdo a la siguiente lógica de funcionamiento:

- Inicialmente el botón **Inicia secuencia** debe estar activado y el botón **Cancela secuencia** debe estar desactivado (modificar método **go**).
- Cuando se presione el botón **Inicia 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 **Inicia secuencia** (modificar **ActionListener** del primero).

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

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









- 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 le restará 0,1 segundos.

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.

No será necesario utilizar nada si se utiliza la clase `ZonaIntercambio` ya que esta utiliza un `AtomicLong`, solucionando problemas de atomicidad y visibilidad

### 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, y los resultados para 16 hebras en patan. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

Justifica los resultados obtenidos.

Ejecución con 500 000 000 rectángulos				
	4 hebras (aula)		16 hebras (patan)	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial	1,259	—	1,995	—
Paralela: Múltiples acumul.	13,097	0,096	82,967	0,024
Paralela: Una única acumul.	0,308	4,081	0,198	10,1
Paralela: Múltiples acumul. (clase atom.)	1,377	0,91	0,397	5,033
Paralela: Una única acumul. (clase atom.)	0,36	3,5	0,19	10,52

Podemos ver que acceder muchas veces a un recurso compartido que bloquea con `synchronized` produce retrasos muy importantes en la ejecución.

Utilizar un `DoubleAdder` con muchas inserciones es mejor pero sigue creando un cuello de botella que ralentiza el programa.

Las dos opciones que utilizan la ejecución en dos fases son mucho mejores, ya que aprovechan al máximo cada hebra, evitando bloqueos a la espera de guardar valores en el recurso compartido. Parece que la versión con el `DoubleAdder` es algo mejor debido a que es una clase optimizada para esta tarea y para ser accedida por varias hebras a la vez.