EI1024/MT1024 "Programación Concurrente y Paralela" 2023–24	Entregable para Laboratorio
Nombre y apellidos (1): Laura Llorens Angulo	
Nombre y apellidos (2): Martin Martinez Ramos	
Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio): 1:30	la01_g

Tema 03. Conceptos Básicos de Concurrencia en Java

1 Se desea crear y arrancar dos hebras que se ejecuten concurrentemente.

Cada hebra debe disponer de un identificador entero único, cuya secuencia comienza por cero. Además, cada hebra debe escribir en pantalla mil veces su identificador.

1.1) Crea las hebras a partir de una subclase de la clase `Thread`, y utiliza el método `start` para arrancar las hebras.
Escribe a continuación el código completo partiendo del siguiente esquema.

```
class MiHebra ... Thread {
  ...
  public MiHebra( int miId ) {
    ...
  }
  public void run() {
    for( int i = 0; i < 1000; i++ ) {
      System.out.println( "Hebra: " + miId );
    }
  }
}
class EjemploCreacionThread {
  public static void main( String args[] ) {
    new ...
    new ...
  }
}
```

```
class MiHebra extends Thread {
 int miId;
 public MiHebra( int miId ) {
  this.miId = miId;
 }
 public void run() {
  for( int i = 0; i < 1000; i++ ) {
   System.out.println( "Hebra: " + miId );
  }
 }
}
class EjemploCreacionThread {
 public static void main( String args[] ) {
  Thread t0 = new Thread(new MiHebra(0));
  Thread t1 = new Thread(new MiHebra(1));
  t0.start();
  t1.start();
 }
}
```

1.2) Crea las hebras a partir de un objeto de la clase `Thread` y un objeto de una clase que implemente la interfaz `Runnable`, utilizando el método `start` para arrancar las hebras.
Escribe a continuación el código completo partiendo del siguiente esquema.

```
class MiRun ... Runnable {
  ...
  public MiRun( int miId ) {
    ...
  }
  public void run() {
```

```
      for( int i = 0; i < 1000; i++ ) {
        System.out.println( "Hebra: " + miId );
      }
    }
  }
  class EjemploCreacionRunnable {
    public static void main( String args[] ) {
      new ...
      new ...
    }
  }
```

```
class MiRun implements Runnable {
 int miId;
 public MiRun( int miId ) {
  this.miId=miId;
 }
 public void run() {
  for( int i = 0; i < 1000; i++ ) {
   System.out.println( "Hebra: " + miId );
  }
 }
}
class EjemploCreacionRunnable {
 public static void main( String args[] ) {
  Thread t0 = new Thread(new MiRun(0));
  Thread t1 = new Thread(new MiRun(1));
  t0.start();
  t1.start();
 }
}
```

1.3) Al probar los códigos, ¿las dos hebras se ejecutan concurrentemente? Razona tu respuesta.

Las hebras se ejecutan de forma concurrente ya que al usar el metodo start es la hebra quien ejecuta el metodo run y no el programa principal, por tanto ambas hebras ejecutan el código de forma concurrente.

1.4) Explica, SIN PROBARLO, qué ocurriría si se sustituyese el método `start` por el método `run` en alguno de los códigos anteriores.

Si usara el método run en lugar del método start se trata como si fuera un método cualquiera en lugar de como un hilo, es el programa principal quien ejecutara el método y por tanto no seria concurrente.

1.5) Realiza el cambio anterior, ejecuta el código y describe qué ocurre.

Ocurre lo descrito en el apartado anterior, primero se ejecuta el hilo t0 imprimiendo 1000 veces seguidas "Hebra: 0", y después la hebra t1 imprimiendo 1000 veces seguidas "Hebra: 1"

2 Se desea crear y arrancar dos hebras que se ejecuten concurrentemente.

Cada hebra debe disponer de un **identificador entero único**, cuya secuencia comienza por cero. Además, cada hebra también debe recibir **dos números en el constructor** y calcular la suma de los números comprendidos entre estos números, ambos inclusive, utilizando un bucle.

Por su parte, el programa principal debe imprimir un mensaje al inicio del programa, arrancar las hebras para que realicen un millón de sumas, y finalizar con la impresión de otro mensaje.

Escribe a continuación el código completo partiendo del siguiente esquema.

```
class MiHebra extends Thread {
  // ... (A)

  public MiHebra( int miId, int num1, int num2 ) {
    // ... (B)
  }

  public void run() {
    long suma = 0;
```

```
    System.out.println( "Hebra Auxiliar " + miId + " , inicia calculo" );
    for( int i = num1; i <= num2 ; i++ ) {
      suma += (long) i;
    }
    System.out.println( "Hebra Auxiliar " + miId + " , suma: " + suma");
  }

}

class EjemploDaemon {
  public static void main( String args[] ) {
    System.out.println( "Hebra Principal inicia" );
    // Crea y arranca hebra t0 sumando desde 1 hasta 1000000
    // Crea y arranca hebra t1 sumando desde 1 hasta 1000000
    // ... (C)
    System.out.println( "Hebra Principal finaliza" );
  }
}
```

2.1) Escribe la declaración de variables y el constructor de la clase `MiHebra`.
Estas líneas se deben insertar a continuación de las líneas marcadas con "(A)" y "(B)".

```
class MiHebra extends Thread {
  int miId;
  int num1;
  int num2;

  public MiHebra( int miId, int num1, int num2 ) {
    this.miId=miId;
    this.num1=num1;
    this.num2=num2;
  }
```

2.2) Escribe el código del código principal que realiza el arranque de las hebras.
Estas líneas se deben insertar a continuación de la línea marcadas con "(C)" .

```
class EjemploDaemon {
  public static void main( String args[] ) {
    System.out.println( "Hebra Principal inicia" );
    Thread t0 = new Thread(new MiHebra(0,0,1000000));
    Thread t1 = new Thread(new MiHebra(1,0,1000000));
    t0.start();
    t1.start();
    System.out.println( "Hebra Principal finaliza" );
  }
```

2.3) Al probar el código, ¿se observa que las hebras se han ejecutado concurrentemente con el programa principal? ¿Qué hebra ha finalizado antes, el programa principal o las hebras auxiliares? Razona tus respuestas.

Las hebras se han ejecutado concurrentemente ya que hemos usado el método start. Las hebras auxiliares han finalizado antes y así lo harán siempre porque el programa no finaliza hasta que las hebras lo hagan.

2.4) Si antes de arrancar las hebras, éstas se definen como hebras de tipo "Daemon", ¿cómo se altera la ejecución? Prueba el nuevo código y razona tu respuesta.

Si definimos las hebras como Daemon entonces el código principal no espera a que terminen para finalizar, por tanto la mayoria de las veces el programa principal finalizara antes que las hebras.

2.5) Incluye las órdenes necesarias para asegurar que el programa principal espere la finalización de las dos hebras antes de realizar la impresión final. ¿Cómo se altera la ejecución? ¿Cambiaría su comportamiento si las hebras fuesen no "Daemon"? Razona tus respuestas.

```
t0.start();
t1.start();
try{
 t0.join();
 t1.join();
} catch (InterruptedException e) {
  e.printStackTrace();
}
System.out.println("Hebra Principal finaliza");
```

Al añadir los join el programa principal espera a que las hebras terminen para finalizar. En este caso, sin importar si definimos la hebra como "Daemon" o no "Daemon" las hebras finalizarán antes que el programa principal. En el caso de que el programa principal sea interrumpido y finalice antes que las hebras, si son daemon finalizarán, pero si no lo son seguirán ejecutandose en segundo plano.

3 Se desea crear y arrancar un conjunto de hebras.

Cada hebra debe disponer de un identificador entero único, cuya secuencia comienza por cero.

Cada hebra debe realizar un **millón de incrementos sobre un objeto compartido** recibido en el constructor. Además, cada hebra debe imprimir un **mensaje justo antes** de comenzar dicha tarea y también **justo después** de terminar dicha tarea.

A continuación se muestra un código que se puede emplear como punto de partida. Este código contiene la definición de la clase del objeto sobre el cual se realizarán los incrementos, y un esquema de la clase `MiHebra`. Además, el programa principal realiza la comprobación y extracción de los argumentos de entrada de la línea de comandos.

En el resto de las prácticas de la asignatura, siempre que se vayan a usar argumentos de la línea de comandos, habrá que comprobarlos (su número y tipo) de forma similar.

```
// ===========================================================
class CuentaIncrementos {
// ===========================================================
  long contador = 0;

  // -----------------------------------------------------------
  void incrementaContador() {
    contador++;
  }

  // -----------------------------------------------------------
  long dameContador() {
    return( contador );
  }
}


// ===========================================================
class MiHebra extends Thread {
// ===========================================================
  // Declaracion de variables
  // ...

  // -----------------------------------------------------------
  // Definicion del constructor, si es necesario
  // ...

  // -----------------------------------------------------------
  public void run() {
    System.out.println( "Hebra: " + miId + " Comenzando incrementos" );
    // Bucle de 1000000 incrementos del objeto compartido
    // ...
    System.out.println( "Hebra: " + miId + " Terminando incrementos" );
  }
```

```
}

// ============================================================================
class EjemploIncrementos {
// ============================================================================

  // ----------------------------------------------------------------------
  public static void main( String args[] ) {
    int  numHebras;

    // 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 );
    }
    System.out.println( "numHebras: " + numHebras );

    // ---------- INCLUIR NUEVO CODIGO A CONTINUACION ----------------------
    // ...
  }
}
```

3.1) Escribe la clase de la hebra.

```
// ============================================================================
class MiHebra extends Thread {
// ============================================================================
 int miId;
 CuentaIncrementos contador;
 // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 public MiHebra( int miId, CuentaIncrementos contador ) {
  this.miId = miId;
  this.contador = contador;
 }
 // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 public void run() {
  System.out.println( "Hebra: " + miId + " Comenzando incrementos" );
  for( int i = 0; i < 1000; i++ ) {
   contador.incrementaContador();
  }
  System.out.println( "Hebra: " + miId + " Terminando incrementos" );
 }
}
```

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

3.2) Escribe un programa principal que realice las siguientes tareas, en el orden especificado:

1. El programa principal debe averiguar el número de hebras que debe crear. Este número debe ser pasado al programa en la línea de argumentos. Así, por ejemplo, el comando `java EjemploIncrementos 4` deberá crear 4 hebras.
   Si el número de argumentos de la línea de argumentos no es correcto, el programa debe avisar y terminar. Asimismo, si algún argumento no es correcto (por ejemplo, se esperaba un argumento numérico y no lo es), el programa también debe avisar y terminar.
   Este apartado aparece resuelto en el código anterior, y servirá de ejemplo para el resto de prácticas.
2. El programa principal debe crear e inicializar el objeto compartido.
   Escribe a continuación la parte de tu código que realiza tal tarea.

```
CuentaIncrementos contador = new CuentaIncrementos();
```

3. El programa principal debe imprimir el valor inicial del contador.
   Escribe a continuación la parte de tu código que realiza tal tarea.

```
System.out.println( "Valor inicial contador: " + contador.dameContador() );
```

4. El programa principal debe crear y arrancar las hebras, utilizando un vector de hebras.
   Escribe a continuación la parte de tu código que realiza tal tarea.

```
MiHebra vh [] = new MiHebra [ numHebras ];
 for ( int i = 0; i < numHebras ; i ++ ) {
  vh [ i ] = new MiHebra ( i, contador );
  vh [ i ]. start () ;
 }
```

5. El programa principal debe esperar a que todas las hebras finalicen su ejecución.
   Escribe a continuación la parte de tu código que realiza tal tarea.

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

6. El programa principal debe imprimir el valor final del contador.
   Escribe a continuación la parte de tu código que realiza tal tarea.

```
System.out.println("Valor final contador: " + contador.dameContador());
```

3.3) Comprueba si hay concurrencia entre las hebras. ¿Cómo puedes demostrarlo? Razona tu respuesta.

Sí hay concurrencia, podemos verlo porque las hebras no inician y finalizan en orden y ademas si fueran secuenciales prinmero una conatia un millón, luego la segunda y asi hasta que finalmente el contador muestre 4 millones, pero no imprime eso (Por ejemplo, imprime: 1844011, 2812568).

3.4) ¿Si se crean 4 hebras, qué valor debería imprimir el programa principal? ¿Cuál es el valor escrito por el ordenador?

3.5) ¿Dónde crees que está el error?

Nota: Este apartado no se evaluará, dado que este problema y su solución se estudiarán a fondo en temas siguientes. Simplemente lo hemos añadido para que comencéis a reflexionar sobre este problema.

Como todas las hebras modifican el contador a la vez hay problemas de atomicidad, se arrelgaria haciendo el contador volatile.

4 Se dispone de una interfaz gráfica sencilla, cuyo código se muestra a continuación, que permite examinar si un número es primo o no. Este código también contabiliza el número de veces que se ha pulsado el botón `Pulsa aquí`.

El código es el siguiente:

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


// ============================================================
public class GUIPrimoSencillo {
// ============================================================

  // Declaration of variables.
  JFrame      container;
  JPanel      jpanel;
  JTextField  txfNumero, txfMensajes, txfSugerencias;
  JButton     btnPulsaAqui, btnComienzaCalculo;
  int         numVecesPulsado = 0;

  //-------------------------------------------------------------
  public static void main( String args[] ) {
    GUIPrimoSencillo gui = new GUIPrimoSencillo();
    SwingUtilities.invokeLater(new Runnable(){
```

```
      public void run(){
        gui.go();
      }
    });
  }

  //-----------------------------------------------------------------------
  public void go() {
    // Variables.
    JPanel     tempPanel;

    // Crea el JFrame principal.
    container = new JFrame( "GUI Primo Sencillo" );

    // Consigue el panel principal del Frame "container".
    jpanel = ( JPanel ) container.getContentPane();
    //// jpanel.setPreferredSize( new Dimension( maxWinX, maxWinY ) );
    jpanel.setLayout( new GridLayout( 4, 1 ) );

    // Crea y anyade la zona de entrada de datos.
    tempPanel = new JPanel();
    tempPanel.setLayout( new FlowLayout() );
    tempPanel.add( new JLabel( "Numero a estudiar:" ) );
    txfNumero = new JTextField( "", 20 );
    tempPanel.add( txfNumero );
    jpanel.add( tempPanel );

    // Crea y anyade la zona de control (botones).
    tempPanel = new JPanel();
    tempPanel.setLayout( new FlowLayout() );

    btnPulsaAqui = new JButton( "Pulsa aqui" );
    btnPulsaAqui.addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent e ) {
          numVecesPulsado++;
          txfMensajes.setText( "Has pulsado " + numVecesPulsado +
                               " veces el boton 'Pulsa aqui'" );
        }
      }
    );
    tempPanel.add( btnPulsaAqui );

    btnComienzaCalculo = new JButton( "Comienza calculo" );
    btnComienzaCalculo.addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent e ) {
          if( txfNumero.getText().trim().length() == 0 ) {
            txfMensajes.setText( "Debes escribir un numero." );
          } else {
            try {
              // Validacion del numero
              long numero = Long.parseLong( txfNumero.getText().trim() );
              // Calculo e impresion en el terminal
              System.out.println( "Examinando numero: " + numero );
              boolean primo = esPrimo( numero );
              if( primo ) {
                System.out.println( "El numero " + numero + " SI es primo." );
              } else {
                System.out.println( "El numero " + numero + " NO es primo." );
              }
            } catch( NumberFormatException ex ) {
              txfMensajes.setText( "No es un numero correcto." );
```

```
            }
          }
        }
      }
    );
    tempPanel.add( btnComienzaCalculo );

    jpanel.add( tempPanel );

    // Crea y anyade la zona de mensajes.
    tempPanel = new JPanel();
    tempPanel.setLayout( new FlowLayout() );
    tempPanel.add( new JLabel( "Mensajes: " ) );
    txfMensajes = new JTextField( "", 30 );
    tempPanel.add( txfMensajes );
    jpanel.add( tempPanel );

    // Crea e inserta el cuadro de texto de sugerencias.
    txfSugerencias = new JTextField( 40 );
    txfSugerencias.setEditable( false );
    txfSugerencias.setText( "321534781, 433494437, 780291637, 1405695061, 2971215073" );
    tempPanel = new JPanel();
    tempPanel.setLayout( new FlowLayout() );
    tempPanel.add( new JLabel( "Sugerencias: " ) );
    tempPanel.add( txfSugerencias );
    jpanel.add( tempPanel );

    // 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 );
  }
}
```

4.1) Realiza las siguientes acciones de modo consecutivo: Pulsa varias veces el botón `Pulsa aquí`. A continuación teclea algún número primo grande (321534781, 433494437, 780291637, 1405695061, 2971215073, etc.) y pulsa el botón de `Comienza calculo`. Inmediatamente después de lanzar el cálculo, vuelve a pulsar varias veces el botón `Pulsa aquí`. ¿Qué ocurre? ¿Es interactiva la interfaz? Razona tu respuesta.

La interfaz no es iterativa porque se bloquea durante el cálculo. Mientras calcula si el valor es primo o no, no puedes pulsar el botón.

4.2) Modifica la aplicación para que cada número que se desee examinar sea evaluado por una nueva hebra. La validez del número debe realizarla el programa principal, mientras que las hebras únicamente deben evaluar si un número válido es primo e imprimir el resultado.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de la nueva clase hebra y la modificación del gestor de evento correspondiente.

```
class MiHebra extends Thread {
  long numero;

  public MiHebra(long numero) {
    this.numero = numero;
  }

  public void run() {
    System.out.println( "Examinando numero: " + numero );
    boolean primo = esPrimo( numero );
    if( primo ) {
      System.out.println( "El numero " + numero + " SI es primo." );
    } else {
      System.out.println( "El numero " + numero + " NO es primo." );
    }
  }
}
```

```
....
try {
            // Validacion del numero
            long numero = Long.parseLong( txfNumero.getText().trim() );
            // Calculo e impresion en el terminal
            Thread t = new Thread(new MiHebra(numero));
            t.start();
          } catch( NumberFormatException ex ) {
            txfMensajes.setText( "No es un numero correcto." );
          }
...
```

4.3) Con el nuevo código modificado repite el proceso inicial (pulsa varias veces el botón `Pulsa aquí`, a continuación teclea algún número primo grande, e inmediatamente después vuelve a pulsar varias veces el botón `Pulsa aquí`). ¿Qué ocurre? ¿Es interactiva la interfaz ahora?

Ahora permite pulsar mientras hace varios calculos a la vez, ya que cada calculo de si es primo lo hace una hebra distinta. Ahora la interfaz sí es interactiva.

4.4) ¿Las hebras auxiliares deberían ser definidas del tipo "Daemon"? ¿Cómo varía el comportamiento de la interfaz gráfica si se define o no a las hebras como de tipo "Daemon"? Razona tu respuesta.

Deberian ser de tipo daemon para que el programa no espere a que finalicen las hebras y que cuando cerremos el programa no espere. En este caso no varía, aunque lo definamos como daemon se termina todo al cerrar el programa.

Las hebras indican si un número es primo o no realizando una llamada al método `System.out.println`. Sería más conveniente que las hebras indicarán si un número es o no primo mediante un mensaje en el cuadro de texto de la interfaz gráfica etiquetado con el texto `Mensajes:`.

Desafortunadamente, una hebra auxiliar creada por el programador no puede *modificar* la interfaz gráfica dado que los métodos de los objetos gráficos de AWT y Swing no son *thread-safe*. Por ello, en este caso la hebra auxiliar se limita a escribir el resultado en la salida estándar. Más adelante se estudiará una solución a este problema.