EI1024/MT1024 "Programación Concurrente y Paralela" 2022–23	Entregable para Laboratorio
Nombre y apellidos (1):	
Nombre y apellidos (2):	la01_g
Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio):	

Tema 04. Conceptos Básicos de Concurrencia en Java

1 El siguiente código crea una **hebra auxiliar** a partir de una subclase de la clase `Thread`. y posteriormente arranca la hebra con el método `start`.

Toda esta labor es realizada por la **hebra principal** (método *main*), la cual realiza otras operaciones.

```
class MiHebra extends Thread {
  public void run() {
    for( int i = 0; i < 100; i++ ) {
      System.out.println( "Ejecutando Hebra Auxiliar" );
    }
  }
}

class EjemploCreacionThreadSimple {
  public static void main( String args[] ) {
    MiHebra t = new MiHebra( );
    t.start( );
    for( int i = 0; i < 100; i++ ) {
      System.out.println( "Ejecutando Hebra Principal" );
    }
  }
}
```

1.1) Compila, ejecuta el código y responde a la siguiente pregunta. ¿La hebra principal y la hebra auxiliar se ejecutan concurrentemente? Razona tu respuesta.

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

1.2) Sustituye el método `start` por el método `run`. Compila, ejecuta el código y responde a la siguiente pregunta. ¿La hebra principal y la hebra auxiliar se ejecutan concurrentemente? Razona tu respuesta.

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

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

2 El siguiente código crea una **hebra auxiliar** a partir de un objeto de la clase `Thread` y un objeto de la una clase que implementa la interfaz `Runnable`, que posteriormente arranca la hebra con el método `start`.

Toda esta labor es realizada por la **hebra principal** que incluye otra serie de operaciones.

```
class MiRun implements Runnable {
  public void run() {
    for( int i = 0; i < 100; i++ ) {
      System.out.println( "Ejecutando Hebra Auxiliar" );
    }
  }
}

class EjemploCreacionRunnableSimple {
  public static void main( String args[] ) {
    MiRun r = new MiRun( );
    Thread t = new Thread( r );
    t.start( );
    for( int i = 0; i < 100; i++ ) {
      System.out.println( "Ejecutando Hebra Principal" );
    }
  }
}
```

Compila, ejecuta el código y responde a las siguientes preguntas.

2.1) ¿La hebra principal y la hebra auxiliar se ejecutan concurrentemente? Razona tu respuesta.

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

2.2) Utiliza el método `run` para arrancar la hebra auxiliar. ¿La hebra principal y la hebra auxiliar se ejecutan concurrentemente? Razona tu respuesta.

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

3 Modifica el código presentado en el ejercicio 1, incluyendo un constructor en la clase `MiHebra` definida, tal y como sigue.

```
class MiHebra extends Thread {
  final int miId;

  public MiHebra( int miId ) {
    this.miId = miId;
  }

  public void run() {
    for( int i = 0; i < 100; i++ ) {
      System.out.println( "Ejecutando Hebra Auxiliar " + miId );
    }
  }
}
```

3.1) ¿De qué tipo es la variable `miId` (variable de clase, de instancia o local)?

3.2) ¿Cómo modificarías el método `main` para que el identificador de la hebra auxiliar fuese 0?

3.3) Modifica el programa principal para que se cree una segunda hebra auxiliar con el identificador 1, sin crear una clase nueva.

3.4) Asegúrate que la hebra principal emplea el método `start` para arrancar a las hebras. ¿Las dos hebras auxiliares y el programa principal se ejecutan simultáneamente? ¿Ambas hebras auxiliares imprimen su identificador? Razona tu respuesta.

3.5) Sustituye el método `start` por el método `run` para ejecutar ambas hebras. ¿Las dos hebras auxiliares y el programa principal se ejecutan simultáneamente? Razona tu respuesta.

4 Modifica el código presentado en el ejercicio 2, incluyendo un constructor en la clase `MiRun`.

```
class MiRun implements Runnable {
  final int miId;
  public MiRun( int miId ) {
    this.miId = miId;
  }
  public void run() {
    for( int i = 0; i < 100; i++ ) {
      System.out.println( "Ejecutando Hebra Auxiliar " + miId );
    }
  }
}
```

4.1) ¿Cómo modificarías el método `main` para que el identificador de la hebra auxiliar fuese 0?

4.2) Si utilizas el siguiente código para añadir una segunda hebra auxiliar.

```
Thread t1 = new Thread( r );
t1.start( );
```

¿Qué identificador se muestra?

.....
.....

4.3) Modifica el código anterior para que el identificador de la segunda hebra auxiliar sea 1.

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

4.4) Asegúrate que ambas hebras utilizan el método `start` para ser ejecutadas, y responde a estas preguntas:

¿Su comportamiento concurrente es similar al del ejercicio 1? ¿Ambas hebras auxiliares imprimen su identificador? Razona tu respuesta.

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

4.5) Utiliza el método `run` para ejecutar ambas hebras. ¿Su comportamiento concurrente es similar al del ejercicio 1? Razona tu respuesta.

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

5 El siguiente código es una modificación del código del ejercicio 1 para que la hebra calcule el sumatorio de todos los números que se encuentran entre dos números (`n1` y `n2`) que recibe cada hebra en el constructor.

```
class MiHebra extends Thread {
  final int miId;
  final int n1;
  final int n2;
  public MiHebra( int miId, int n1, int n2 ) {
    this.miId = miId;
    this.n1 = n1;
    this.n2 = n2;
  }
  public void run() {
    int suma = 0;
    for( int i = n1; i <= n2; i++ ) {
      suma += i;
    }
    System.out.println( "Hebra Auxiliar " + miId + " , suma: " + suma");
  }
}
```

```
class EjemploCreacionThreadSumas {
  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
    System.out.println( "Hebra Principal finaliza" );
  }
}
```

5.1) Incorpora las instrucciones que permiten arrancar dos hebras que calculen el sumatorio entre 1 y 1000000 de modo concurrente.

5.2) Ejecuta el código varias veces y responde a las siguientes preguntas:

¿Quién finaliza primero: la hebra principal, la hebra auxiliar `t0` o la hebra `t1`? ¿Ocurre en todos los casos?

¿Quién finaliza en último lugar: la hebra principal, la hebra auxiliar `t0` o la hebra `t1`? ¿Ocurre en todos los casos?

Razona tus respuestas.

5.3) En algunos casos, las hebras auxiliares deben finalizar en cuanto lo haga la hebra principal, por ejemplo cuando su función es totalmente auxiliar a la hebra principal. El método `setDaemon` permite activar esta opción, cuando recibe el parámetro `true`.

```
// ...
t0.setDaemon( true ); // t0 se convierte en una "daemon"
t0.start( );
// ...
t1.setDaemon( false ); // t1 no es una "daemon" (valor por defecto)
t1.start( );
```

Razona tus respuestas a las siguientes preguntas antes de ejecutar el código. Después, ejecuta el código y compara lo que ocurre con tu respuesta.

¿Qué ocurre cuando una hebra auxiliar se define como *daemon* (como en el código anterior)?

¿Qué ocurre cuando todas las hebras auxiliares se definen como *daemon*?

5.4) En muchos casos, es conveniente que la hebra principal espere hasta que las hebras auxiliares han finalizado. El método `join` bloquea a la hebra que ejecuta el método hasta que finalice la hebra asociada. Dado que es bloqueante, este método debe gestionar la llegada de excepciones, por lo que su uso siempre debe aparecer en un bloque `try-catch`.

Por ejemplo, el siguiente código hace que la hebra que lo ejecuta se bloquee (espere) hasta que la hebra `t0` ha terminado.

```
try {
  t0.join( );
} catch( InterruptedException ex ) {
  ex.printStackTrace();
}
```

El lugar en el que se sitúa el método `join` es una cuestión crítica que hace que un programa funcione o no correctamente.

En primer lugar, sitúa la llamada al método `join` justo a continuación de ejecutar el arranque de la hebra correspondiente.

```
// ...
t0.start( )
try {
  t0.join( );
} catch( InterruptedException ex ) {
  ex.printStackTrace();
}
//...
```

¿Existe concurrencia en la ejecución? Razona tu respuesta antes de ejecutar el código y después comprueba qué ocurre.

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

Si no hubiese concurrencia, ¿cómo lo resolverías? Razona tu respuesta.

La alternativa propuesta hay que implementarla para asegurarse de que es correcta.

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

Cuando existe concurrencia y la hebra principal espera la finalización de las hebras auxiliares, ¿tiene algún sentido definir a estas últimas como *daemon*?

Razona tu respuesta.

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

5.5) Cuando crece el número de hebras no resulta adecuado crearlas de modo individual, siendo más aconsejable utilizar bucles que simplifiquen los códigos. En estos casos suele ser habitual definir un vector de hebras en el que se almacene los objetos creados, tal y como sigue:

```
MiHebra v[] = new MiHebra[ numHebras ];
```

donde `numHebras` suele ser un parámetro que se recibe en la linea de comando.
Seguidamente se muestra como adaptar el método `main` para leer el valor de `numHebras`.

```
// ------------------------------------------------------------------------
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 );

  // ---------  CODIGO PRINCIPAL TRAS PROCESAR PARAMETROS  ---------------

  System.out.println( "Hebra Principal inicia" );
  // Crear un vector (v) de tipo MiHebra con numHebras elementos
  MiHebra v[] = new MiHebra[ numHebras ];
  // Crear y arrancar las hebras y almacenarlas en v
              // ...
  // Esperar a que terminen todas las hebras almacenadas en v
              // ...
  System.out.println( "Hebra Principal finaliza" );
}
// ------------------------------------------------------------------------
```

Escribe el código que permite crear y arrancar las hebras.

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

Escribe el código que permite a la hebra principal esperar a que finalicen todas las hebras.

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

5.6) En el caso que todas las hebras modificasen un mismo objeto compartido, ¿piensas que la suma final sería la correcta? Razona tu respuesta

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

6 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 GUIPrimoSencillo1a {
// ============================================================================

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

  // ----------------------------------------------------------------------------
  public static void main( String args[] ) {
    GUIPrimoSencillo1a gui = new GUIPrimoSencillo1a();
    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 1a" );

    // 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 {
          long numero = Long.parseLong( txfNumero.getText().trim() );
          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 );
  }
}
```

6.1) 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.

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

6.2) ¿Cómo podrías modificar el programa para resolver el problema?

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

Nota: Esta entrega forma parte de la evaluación de la asignatura. Debe ser guardado por el estudiantado junto con el resto de entregas en una carpeta. El profesorado podrá pedir al estudiantado que le entregue dicha carpeta en cualquier momento.