EI1024/MT1024 "Programación Concurrente y Paralela" 2025–26	Entregable para Laboratorio
Nombre y apellidos (1): Joel Gil Juan	
Nombre y apellidos (2): Maria Palloti Romero	
Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio): 2:00	la07_g

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

Tema 08. El Problema de Coordinación en Java

Se dispone de un programa secuencial que calcula el municipio de una provincia que tiene **la mayor diferencia entre la temperatura máxima y la mínima**. En este cálculo, se considera la previsión realizada por la AEMET (Agencia Española de Metereología) en un día determinado de la semana actual. El objetivo de esta práctica es desarrollar una versión paralela que mejore sus prestaciones.

Dado que no se conocen los códigos de municipio válidos de una provincia, se realiza una primera ejecución secuencial en las que se verifican todos los posibles códigos, guardando los códigos de pueblos válidos en el fichero "codPueblos_XX.txt", donde "XX" se corresponde con el código de la provincia. Posteriormente, se realiza una nueva ejecución secuencial en la que únicamente se procesan los municipios que aparecen en el fichero "codPueblos_XX.txt".

La creación del "codPueblos_XX.txt" es la tarea más costosa de la aplicación, razón por la que solo se realiza en la primera ejecución. En el resto de ejecuciones, la aplicación lee los códigos del fichero. **IMPORTANTE:**. Cuando el resultado de una ejecución sea el código de pueblo (-1) , indica que se ha finalizado prematuramente la primera ejecución. Borra el fichero y vuelve a ejecutar el código.

Para facilitar la implementación, el método `obtenMayorDiferenciaDeFichero`, incluye la implementación de todas las versiones, recibiendo como parámetro qué versión se desea ejecutar.

El código secuencial del que se dispone es el siguiente:

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicInteger;

class EjemploTemperaturaProvincia {
  public static void main(String[] args) {
    int                 numHebras, codProvincia, desp;
    String              nombreFichero = "";
    long                t1, t2, tt[];
    double              ts, tp;
    PuebloMaximaMinimaSec MaxMinSec; // Objeto sin sincronizacion
    PuebloMaximaMinimaPar MaxMinPar; // Objeto con sincronizacion

    // Comprobacion y extraccion de los argumentos de entrada.
    if( args.length != 3 ) {
      System.out.println( "ERROR: numero de argumentos incorrecto.");
      System.out.println( "Uso: java programa <numHebras> <provincia> <desplazamiento>" );
      System.exit( -1 );
    }
```

```
try {
  numHebras    = Integer.parseInt( args[ 0 ] );
  codProvincia = Integer.parseInt( args[ 1 ] );
  desp         = Integer.parseInt( args[ 2 ] );
  if( ( numHebras <= 0 ) || (codProvincia < 1) || (codProvincia > 50) ||
      (desp < 0) || (desp >= 7) ) {
      System.out.println( "Uso: java programa <numHebras> <provincia> <desplazamiento>" );
      System.err.print( "  donde ( numHebras > 0 ) , codProvincia in [ 1 , 50 ] ," );
      System.err.println( "( desplazamiento in [ 0 , 6 ]" );
      System.exit( -1 );
  }
} catch( NumberFormatException ex ) {
  numHebras    = -1;
  codProvincia = -1;
  desp         = -1;
  System.out.println( "ERROR: Numero de entrada incorrecto." );
  System.exit( -1 );
}

// Mensaje inicial
System.out.println();
System.out.println( "Obtiene el pueblo de una provincia con mayor diferencia " +
                    "de temperatura." );

// Nombre del fichero de codigos
if (codProvincia < 10) {
  nombreFichero = "codPueblos_0" + codProvincia + ".txt";
} else {
  nombreFichero = "codPueblos_"  + codProvincia + ".txt";
}

// Seleccion del dia elegido
String fecha;
Calendar c = Calendar.getInstance();
Integer dia, mes, anyo;

c.add(Calendar.DAY_OF_MONTH, desp);
dia = c.get(Calendar.DATE);
mes = c.get(Calendar.MONTH) + 1;
anyo = c.get(Calendar.YEAR);
fecha = String.format("%02d", anyo) + "-" + String.format("%02d", mes) + "-" +
        String.format("%02d", dia);
System.out.println("Fecha de busqueda: " + fecha);

//
// Implementacion secuencial sin temporizar.
//
MaxMinSec = new PuebloMaximaMinimaSec();
MaxMinPar = new PuebloMaximaMinimaPar();
File f = new File(nombreFichero);
if (f.exists()) {
  obtenMayorDiferenciaDeFichero (nombreFichero, fecha, codProvincia, MaxMinSec, MaxMinPar,
                                 0, numHebras);
} else {
  obtenMayorDiferenciaAFichero_Secuencial (nombreFichero, fecha, codProvincia, MaxMinSec);
}
MaxMinSec = new PuebloMaximaMinimaSec();
obtenMayorDiferenciaDeFichero (nombreFichero, fecha, codProvincia, MaxMinSec, MaxMinPar,
                               0, numHebras);
System.out.println( "  Pueblo: " + MaxMinSec.damePueblo() + " , Maxima = " +
                    MaxMinSec.dameTemperaturaMaxima() + " , Minima = " +
```

```
                        MaxMinSec.dameTemperaturaMinima() );

    //
    // Implementacion secuencial.
    //
    System.out.println();
    t1 = System.nanoTime();
    MaxMinSec = new PuebloMaximaMinimaSec();
    MaxMinPar = new PuebloMaximaMinimaPar();
    obtenMayorDiferenciaDeFichero (nombreFichero, fecha, codProvincia, MaxMinSec, MaxMinPar,
                                   0, numHebras);
    t2 = System.nanoTime();
    ts = ( (double) ( t2 - t1 ) ) / 1.0e9;
    System.out.print( "Implementacion secuencial.                              " );
    System.out.println( " Tiempo(s): " + ts );
    System.out.println( "  Pueblo: " + MaxMinSec.damePueblo() + " , Maxima = " +
                        MaxMinSec.dameTemperaturaMaxima() + " , Minima = " +
                        MaxMinSec.dameTemperaturaMinima() );
/*
    //
    // Implementacion paralela: Gestion Propia.
    //
    System.out.println();
    t1 = System.nanoTime();
    MaxMinSec = new PuebloMaximaMinimaSec();
    MaxMinPar = new PuebloMaximaMinimaPar();
    obtenMayorDiferenciaDeFichero (nombreFichero, fecha, codProvincia, MaxMinSec, MaxMinPar,
                                   1, numHebras);
    t2 = System.nanoTime();
    tp = ( (double) ( t2 - t1 ) ) / 1.0e9;
    System.out.print( "Implementacion paralela: Gestion Propia.      " );
//    System.out.println( " Tiempo(s): " + tp + " , Incremento: " + ... );
//    System.out.println( "  Pueblo: " + ... + " , Maxima = " + ... + " , Minima = " + ... );
*/
    //
    // Implementacion paralela: Thread Pool isTerminated.
    //
    // ...

    //
    // Implementacion paralela: Thread Pool con awaitTermination.
    //
    // ...

    //
    // Implementacion paralela: Thread Pool con Future.
    //
    // ...

  }

  // ------------------------------------------------------------------------
  public static void obtenMayorDiferenciaAFichero_Secuencial (String nombreFichero,
                        String fecha, int codProvincia, PuebloMaximaMinimaSec MaxMin) {
    FileWriter fichero = null;
    PrintWriter pw = null;

    // Verifica todas los codigos de pueblos y escribe el fichero
    try
    {
      // Apertura del fichero y creacion de FileWriter para poder
```

```
      // hacer una lectura comoda (disponer del metodo readLine()).

      fichero = new FileWriter(nombreFichero);
      pw = new PrintWriter(fichero);

      for (int i=codProvincia*1000; i<(codProvincia+1)*1000; i++){
        if (ProcesaPueblo(fecha, i, MaxMin, false) == true) {
          pw.println(i);
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
    } finally {
      try {
        // Nuevamente aprovechamos el finally para
        // asegurarnos que se cierra el fichero.
        if (null != fichero)
          fichero.close();
      } catch (Exception e2) {
        e2.printStackTrace();
      }
    }
  }

  // ------------------------------------------------------------------------

  public static void obtenMayorDiferenciaDeFichero (String nombreFichero, String fecha,
                      int codProvincia, PuebloMaximaMinimaSec MaxMinSec,
                      PuebloMaximaMinimaPar MaxMinPar, int opcion, int numHebras) {
    File fichero = null;
    FileReader fr = null;
    BufferedReader br = null;

    // Procesa el fichero
    try
    {
      // Apertura del fichero y creacion de BufferedReader para poder
      // hacer una lectura comoda (disponer del metodo readLine()).
      fichero = new File (nombreFichero);
      fr = new FileReader (fichero);
      br = new BufferedReader(fr);

      String           linea;
      ExecutorService  exec;
      switch (opcion) {
        case 0:  // Caso secuencial
          while( ( linea = br.readLine() ) != null ) {
            int codPueblo = Integer.parseInt(linea);
            ProcesaPueblo(fecha, codPueblo, MaxMinSec, false);
          }
          break;
        case 1:  // Gestion Propia
          // ...
          break;
        case 2: // ThreadPools con isTerminated
          // ...
          break;
        case 3: // ThreadPools con awaitTermination
          // ...
          break;
        case 4: // ThreadPools + con Future
```

```
        // ...
        break;
      default:
        break;
    }
  } catch (Exception e) {
    e.printStackTrace();
  }finally{
    // En el finally se cierra el fichero, para asegurar
    // que el cierre se completa tanto si todo va bien
    // como si activa una excepcion.
    try{
      if( null != fr ){
        fr.close();
      }
    }catch (Exception e2){
      e2.printStackTrace();
    }
  }
}

// ------------------------------------------------------------------------------
public static boolean ProcesaPueblo (String fecha, int codPueblo,
                                     PuebloMaximaMinimaSec MaxMin, boolean imprime) {
  URL           url;
  InputStream   is = null;
  BufferedReader br;
  String        line, poblacion = new String (), provincia = new String ();
  int           state, num[]=new int[2];
  boolean       res = false;

  // Procesamiento de la informacion XML asociada a codPueblo
  // Actualizacion de MaxMin de acuerdo a los valores obtenidos
  try {
    String urlStr = "https://www.aemet.es/xml/municipios/localidad_" +
                    String.format("%05d",codPueblo)+ ".xml";
    //url = new URL(urlStr);
    url = URI.create(urlStr).toURL();
    is  = url.openStream();  // throws an IOException
    br  = new BufferedReader(new InputStreamReader(is));
    if (imprime) System.out.println(urlStr);

    state = 0;
    while (((line = br.readLine()) != null) && (state < 6)) {
      //        System.out.println (line);
      if ((state == 0) && (line.contains ("nombre"))) {
        poblacion=line.split(">")[1].split("<")[0].split("/")[0];
        state++;
      } else if ((state == 1) && (line.contains ("provincia"))) {
        provincia=line.split(">")[1].split("<")[0].split("/")[0];
        state++;
      } else if ((state == 2) && (line.contains (fecha))) {
        state++;
      } else if ((state == 3) && (line.contains ("temperatura"))) {
        state++;
      } else if ((state > 3) && ((line.contains ("maxima")) || (line.contains ("minima")))) {
        num[state-4] = Integer.parseInt (line.split(">")[1].split("<")[0]);
        state++;
      }
    }
    // System.out.println("(" + codPueblo + ") " + poblacion + "(" + provincia + ") => " +
```

```
      //                    "(" + num[0] + " , " + num[1] + ")");
      if (codPueblo == 24116)
        System.out.println (poblacion + " , " + codPueblo + " , " + num[0] + " , " + num[1]);
      MaxMin.actualizaMaxMin (poblacion, codPueblo, num[0], num[1]);
      res = true;
    } catch (MalformedURLException mue) {
      mue.printStackTrace();
    } catch (IOException ioe) {
      //      ioe.printStackTrace();
    } finally {
      try {
        if (is != null) is.close();
      } catch (IOException ioe) {
        // nothing to see here
      }
    }
    return res;
  }
}

// ============================================================================
class PuebloMaximaMinimaSec {
  // ==========================================================================
  String poblacion;
  int    codigo, max, min;


  // ----------------------------------------------------------------------
  public PuebloMaximaMinimaSec() {
    poblacion = null;
    codigo    = -1;
    max       = -1;
    min       = -1;
  }

  // ----------------------------------------------------------------------
  public void actualizaMaxMin( String poblacion, int codigo, int max, int min ) {
    if ((this.poblacion == null) || ((this.max-this.min) < (max-min)) ||
        (((this.max-this.min) == (max-min)) && (this.min > min)) ||
        (((this.max-this.min) == (max-min)) && (this.min == min) && (this.codigo < codigo))
        ) {
      this.poblacion = poblacion;
      this.codigo = codigo;
      this.max = max;
      this.min = min;
    }
  }

  // ----------------------------------------------------------------------
  public String damePueblo() {
    return this.poblacion + "(" + this.codigo + ")";
  }

  // ----------------------------------------------------------------------
  public int dameCodigo() {
    return this.codigo;
  }

  // ----------------------------------------------------------------------
  public int dameTemperaturaMaxima() {
    return this.max;
```

```
  }

  // ------------------------------------------------------------------------
  public int dameTemperaturaMinima() {
    return this.min;
  }
}

// ============================================================================
class PuebloMaximaMinimaPar {
  // ==========================================================================
  String poblacion;
  int    codigo, max, min;


  // ------------------------------------------------------------------------
  public PuebloMaximaMinimaPar() {
    poblacion = null;
    codigo    = -1;
    max       = -1;
    min       = -1;
  }

  // ------------------------------------------------------------------------
  public void actualizaMaxMin( String poblacion, int codigo, int max, int min ) {
    if ((this.poblacion == null) || ((this.max-this.min) < (max-min)) ||
        (((this.max-this.min) == (max-min)) && (this.min > min)) ||
        (((this.max-this.min) == (max-min)) && (this.min == min) && (this.codigo < codigo))
        ) {
      this.poblacion = poblacion;
      this.codigo = codigo;
      this.max = max;
      this.min = min;
    }
  }

  // ------------------------------------------------------------------------
  public String damePueblo() {
    return this.poblacion + "(" + this.codigo + ")";
  }

  // ------------------------------------------------------------------------
  public int dameCodigo() {
    return this.codigo;
  }

  // ------------------------------------------------------------------------
  public int dameTemperaturaMaxima() {
    return this.max;
  }

  // ------------------------------------------------------------------------
  public int dameTemperaturaMinima() {
    return this.min;
  }
}
```

1 En este apartado, debes realizar una **gestión propia de hebras**, es decir, creando y arrancando las hebras explíctitamente. Su número será igual al parámetro recibido en la línea de comando.

Tanto la lectura del fichero de texto en paralelo como el acceso a la AEMET puede tener un coste muy diverso, por lo que se va a aplicar el **método del Productor-Consumidor**. El programa principal irá leyendo el fichero línea a línea, y tras cada lectura, se generará una nueva tarea que deberá ser procesada. La generación de nuevas tareas se realizará en paralelo al procesamiento de los códigos, y en paralelo a la lectura de las siguientes líneas. Es por ello que se aconseja que la **creación y arranque de las hebras se realice antes** de la lectura del fichero.

En este esquema, la hebra productora (el programa principal) **inserta las tareas en una cola bloqueante** a la que acceden las hebras consumidoras para tomar las tareas. Cuando el fichero se ha leído completamente, la hebra productora **inserta tareas envenenadas** para avisar a las hebras consumidoras.

Por su parte, una hebra consumidora **extrae tareas de la cola bloqueante** hasta que encuentra una **tarea envenenada**. Todas las tareas no envenenadas deben ser procesadas por la hebra consumidora que la haya extraído, actualizando un objeto `PuebloMaximaMinimaXxx`

Se propone que las tareas sean objetos de una clase con dos variables de instancia: `esVeneno` y `codPueblo`. En el caso de una tarea normal, la primera variable valdrá falso y la segunda contendrá el código de pueblo leído. En el caso de una tarea envenenada, la primera variable valdrá cierto y la segunda contendrá el código de pueblo -1. Una posible opción sería la siguiente:

```
// ===========================================================================
class TareaEnColaGestionPropia {
  // =========================================================================
   boolean              esVeneno ;
   int                  codPueblo;

  // -------------------------------------------------------------------------
  public TareaEnColaGestionPropia ( ... ) {
  // ...
  }
  // ...
}
```

Si detectas que debes introducir cambios para que una clase sea *thread-safe*, aplícalos sobre la clase `PuebloMaximaMinimaPar` nunca sobre `PuebloMaximaMinimaSec`. Además, duplica el método `ProcesaPueblo` si fuese necesario.

Comprueba que el nuevo código paralelo funciona correctamente comparando sus resultados con los de la versión secuencial.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de nuevas clases, la modificación de la rutina `obtenMayorDiferenciaDeFichero`, y el código a incluir en el programa principal.

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

2 Realiza una implementación paralela con un *Thread Pool* del tipo `newFixedThreadPool`, en la que **el programa principal incorpore a un *ThreadPool*, al menos, una tarea por cada uno de los códigos de municipio leídos del fichero**. En esta implementación, el programa principal debe esperar a que las hebras terminen **con una espera activa**, es decir, con el método `isTerminated`.

Elige bien la clase `PuebloMaximaMinimaXxx` que debes utilizar, ya que el *Thread Pool* emplea internamente varias hebras. Si éstas acceden a algún objeto compartido, éste debe ser *thread-safe* por lo que debes pensar qué versión de esta clase debes utilizar.

Comprueba que el nuevo código paralelo funciona correctamente comparando sus resultados con los de la versión secuencial inicial.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de nuevas clases, la modificación de la rutina `obtenMayorDiferenciaDeFichero`, y el código a incluir en el programa principal.

3 Realiza una implementación paralela con un *Thread Pool* del tipo `newFixedThreadPool`. Esta implementación es similar al ejercicio anterior, aunque en este caso, el programa principal debe esperar a que las hebras terminen **sin una espera activa**, es decir, con el método `awaitTermination`.

Comprueba que el nuevo código paralelo funciona correctamente comparando sus resultados con los de la versión secuencial inicial.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de nuevas clases, la modificación de la rutina `obtenMayorDiferenciaDeFichero`, y el código a incluir en el programa principal.

4 Realiza una implementación paralela con un *Thread Pool* en el que se maneje la interfaz `Callable`. En esta implementación, **el programa principal incorpora a un *ThreadPool*, al menos, una tarea por cada uno de los códigos de municipio leídos del fichero**, y posteriormente, **debe procesar el resultado obtenido en la ejecución de cada tarea del *Thread Pool***, que debe ser un objeto **PuebloMaximaMinimaXxx**.

Comprueba que el nuevo código paralelo funciona correctamente comparando sus resultados con los de la versión secuencial inicial.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de nuevas clases, la modificación de la rutina `obtenMayorDiferenciaDeFichero`, y el código a incluir en el programa principal.

5 Completa la siguiente tabla, seleccionando el código de provincia de Castellón (12) y eligiendo el desplazamiento para que se analice la previsión del día en el que realizáis la práctica (0).

Municipio	Temp. Máxima	Temp. Mínima
Segorbe	18	4

Obtén los resultados en el **ordenador del aula**, tanto con 4 como con 8 hebras. Redondea los tiempos, dejando tres decimales, y los incrementos, dejando dos decimales.

	4 hebras		8 hebras	
Implementación	Tiempo	Incremento	Tiempo	Incremento
Secuencial	1.369	—	1.468	—
Paralela con gestión propia de hebras	0.384	3.56	0.216	6.78
Paralela con `newFixedThreadPool` y espera activa con `isTerminated`	0.346	3.95	0.207	7.08
Paralela con `newFixedThreadPool` y espera con `awaitTermination`	0.341	4.01	0.195	7.51
Paralela con `newFixedThreadPool` e interfaz `Callable`	0.346	3.94	0.187	7.84

Justifica los resultados obtenidos de prestaciones.

El uso de ThreadPools casi siempre será mejor que los de gestión propia ya que están optimizados. En los resultados podemos observar que es así. Aunque en nuestro caso con 8 hebras varíe el tiempo de ejecución, esto se debe a las limitaciones de la E/S.

¿Qué versión de todas las paralelas ha sido la más fácil de escribir?

Las más fáciles han sido las que utilizan threadpools con isTerminated y awaitTermination porque no nos tenemos que encargar manualmente de los cerrojos para el acceso a datos compartidos, ya que la propia threadpool se encarga de que el acceso sea thread-safe y a diferencia del que utiliza la interfaz Callable,el resultado lo calculan directamente las hebras.

¿Estos cálculos están limitados por la CPU, la memoria central o la E/S? ¿Por qué?

Los cálculos están limitados por la E/S. Porque para acceder a los datos tenemos que acceder al servidor de la AEMET, si el acceso no es rápido, tendremos mucho sobrecoste por acceso al servidor.

6 Repite los cálculos en **karen**, tanto con 16 como con 32 hebras, seleccionando el código de provincia de Castellón (12) y eligiendo el desplazamiento para que se analice la previsión del día en el que realizáis la práctica (0). Para acortar el tiempo de ejecución, copia también en karen el fichero "codPueblos_XX.txt'.

Municipio	Temp. Máxima	Temp. Mínima
Segorbe	18	4

Redondea los tiempos dejando sólo tres decimales y los incrementos dejando dos decimales.

	16 hebras		32 hebras	
Implementación	Tiempo	Incremento	Tiempo	Incremento
Secuencial	1.491	—	1.492	—
Paralela con gestión propia de hebras	0.253	5.87	0.419	3.55
Paralela con `newFixedThreadPool` y espera activa con `isTerminated`	0.178	8.34	0.315	4.72
Paralela con `newFixedThreadPool` y espera con `awaitTermination`	0.175	8.52	0.491	3.03
Paralela con `newFixedThreadPool` e interfaz `Callable`	0.176	8.46	0.184	8.08

Justifica los resultados obtenidos de prestaciones.

El uso de ThreadPools casi siempre será mejor que los de gestión propia ya que están optimizados. En los resultados podemos observar que es así. Con 32 hebras podemos observar que el tiempo de ejecución en isTerminated y en awaitTermination es más alto, esto es debido a las limitaciones de la E/S. Ya que tenemos que acceder tanto al karen como al servidor de la AEMET, y el canal no puede procesar tanta información a la vez.

¿Qué versión de todas las paralelas ha sido la más fácil de escribir?

Las más fáciles han sido las que utilizan threadpools con isTerminated y awaitTermination porque no nos tenemos que encargar manualmente de los cerrojos para el acceso a datos compartidos, ya que la propia threadpool se encarga de que el acceso sea thread-safe y a diferencia del que utiliza la interfaz Callable,el resultado lo calculan directamente las hebras.

¿Estos cálculos están limitados por la CPU, la memoria central o la E/S? ¿Por qué?

Los cálculos están limitados por la E/S. Porque para acceder a los datos tenemos que acceder al servidor de la AEMET y al karen, si el acceso no es rápido, tendremos mucho sobrecoste por acceso al servidor.