EI1024/MT1024 "Programación Concurrente y Paralela" 2022–23	Entregable para Laboratorio
Nombre y apellidos (1):	
Nombre y apellidos (2):	la07_g
Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio):	

Tema 08. Concurrencia en Colecciones de Java

Tema 09. El Problema de la Coordinación en Java

Se dispone de un programa secuencial que calcula y muestra la palabra más usada en un vector de tiras de caracteres, y se desea desarrollar una solución paralela a dicho problema.
En esta práctica, debes emplear las hebras normales (sin *Thread Pools*) y puedes emplear una distribución cíclica del trabajo.
Además, debes ir con cuidado en las versiones concurrentes para evitar que dos hebras intenten añadir la misma palabra al mismo tiempo.
No olvides comprobar que todas las versiones paralelas funcionan correctamente. Para ello debes comparar los resultados (tanto la palabra como su número de veces) de las versiones paralelas con los de la versión secuencial. En las comprobaciones es conveniente que emplees el fichero `f0.txt`, dado que su contenido puede generar más errores.
Para realizar una comparación justa, debes emplear en todas las versiones **los mismos valores de capacidad inicial y factor de carga**. Así, deberías usar 1000 como capacidad inicial y 0.75F como factor de carga.
A continuación se muestra el código secuencial del que se dispone. Como puedes ver, el cálculo secuencial se realiza dos veces, pero sólo se cuenta el tiempo y se muestran resultados para la segunda ejecución. Ello se debe a que la primera ejecución es mucho más costosa (dado que debe cargar un montón de datos en antememoria), pero, obviamente, solo hace falta realizarlo al principio del programa, por lo que **no debes repetirlo** para cada implementación paralela.

```
import java.io.*;
import java.util.*;

// ================================================================
class EjemploPalabraMasUsada {
// ================================================================

  // ----------------------------------------------------------------
  public static void main( String args[] ) {
    long                      t1, t2;
    double                    tt;
    int                       numHebras;
    String                    nombreFichero, palabraActual;
    Vector<String>            vectorLineas;
    HashMap<String,Integer>   hmCuentaPalabras;

    // Comprobacion y extraccion de los argumentos de entrada.
    if( args.length != 2 ) {
      System.err.println( "Uso: java programa <numHebras> <fichero>" );
      System.exit( -1 );
    }
```

```
    try {
      numHebras     = Integer.parseInt( args[ 0 ] );
      nombreFichero = args[ 1 ];
    } catch( NumberFormatException ex ) {
      numHebras = -1;
      nombreFichero = "";
      System.out.println( "ERROR: Argumentos numericos incorrectos." );
      System.exit( -1 );
    }

    // Lectura y carga de lineas en "vectorLineas".
    vectorLineas = readFile( nombreFichero );
    System.out.println( "Numero de lineas leidas: " + vectorLineas.size() );
    System.out.println();

    //
    // Implementacion secuencial sin temporizar.
    //
    hmCuentaPalabras = new HashMap<String,Integer>( 1000, 0.75F );
    for( int i = 0; i < vectorLineas.size(); i++ ) {
      // Procesa la linea "i".
      String[] palabras = vectorLineas.get( i ).split( "\\W+" );
      for( int j = 0; j < palabras.length; j++ ) {
        // Procesa cada palabra de la linea "i", si es distinta de blanco.
        palabraActual = palabras[ j ].trim();
        if( palabraActual.length() > 0 ) {
          contabilizaPalabra( hmCuentaPalabras, palabraActual );
        }
      }
    }

    //
    // Implementacion secuencial.
    //
    t1 = System.nanoTime();
    hmCuentaPalabras = new HashMap<String,Integer>( 1000, 0.75F );
    for( int i = 0; i < vectorLineas.size(); i++ ) {
      // Procesa la linea "i".
      String[] palabras = vectorLineas.get( i ).split( "\\W+" );
      for( int j = 0; j < palabras.length; j++ ) {
        // Procesa cada palabra de la linea "i", si es distinta de blanco.
        palabraActual = palabras[ j ].trim();
        if( palabraActual.length() > 0 ) {
          contabilizaPalabra( hmCuentaPalabras, palabraActual );
        }
      }
    }
    t2 = System.nanoTime();
    tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.print( "Implementacion secuencial: " );
    imprimePalabraMasUsadaYVeces( hmCuentaPalabras );
    System.out.println( " Tiempo(s): " + tt );
    System.out.println( "Num. elems. tabla hash: " + hmCuentaPalabras.size() );
    System.out.println();

/*
    //
    // Implementacion paralela 1: Uso de synchronizedMap.
    //
    t1 = System.nanoTime();
    // ...
```

```
    t2 = System.nanoTime();
    tt = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.print( "Implementacion paralela 1: " );
    imprimePalabraMasUsadaYVeces( maCuentaPalabras );
    System.out.println( " Tiempo(s): " + tt  + " , Incremento " + ...);
    System.out.println( "Num. elems. tabla hash: " + ... );
    System.out.println();

    //
    // Implementacion paralela 2: Uso de Hashtable.
    //
    // ...

    //
    // Implementacion paralela 3: Uso de ConcurrentHashMap
    //
    // ...

    //
    // Implementacion paralela 4: Uso de ConcurrentHashMap
    //
    // ...

    //
    // Implementacion paralela 5: Uso de ConcurrentHashMap
    //
    // ...

    //
    // Implementacion paralela 6: Uso de ConcurrentHashMap
    //
    // ...

    //
    // Implementacion paralela 7: Uso de Streams
    // t1 = System.nanoTime();
    // Map<String,Long> stCuentaPalabras = vectorLineas.parallelStream()
    //                                         .filter( s -> s != null )
    //                                         .map( s -> s.split( "\\W+" ) )
    //                                         .flatMap( Arrays::stream )
    //                                         .map( String::trim )
    //                                         .filter( s -> (s.length() > 0) )
    //                                         .collect( groupingBy (s -> s, counting()));
    // t2 = System.nanoTime();
    // ...
*/
    System.out.println( "Fin de programa." );
  }

  // ---------------------------------------------------------------------
  public static Vector<String> readFile( String fileName ) {
    BufferedReader br;
    String         linea;
    Vector<String> data = new Vector<String>();

    try {
      br = new BufferedReader( new FileReader( fileName ) );
      while( ( linea = br.readLine() ) != null ) {
        //// System.out.println( "Leida linea: " + linea );
        data.add( linea );
      }
```

```
      br.close();
    } catch( FileNotFoundException ex ) {
      ex.printStackTrace();
    } catch( IOException ex ) {
      ex.printStackTrace();
    }
    return data;
  }

  // ---------------------------------------------------------------------
  public static void contabilizaPalabra(
                        HashMap<String,Integer> cuentaPalabras,
                        String palabra ) {
    Integer numVeces = cuentaPalabras.get( palabra );
    if( numVeces != null ) {
      cuentaPalabras.put( palabra, numVeces+1 );
    } else {
      cuentaPalabras.put( palabra, 1 );
    }
  }

  // ---------------------------------------------------------------------
  static void imprimePalabraMasUsadaYVeces(
                  Map<String,Integer> cuentaPalabras ) {
    Vector<Map.Entry> lista =
        new Vector<Map.Entry>( cuentaPalabras.entrySet() );

    String palabraMasUsada = "";
    int    numVecesPalabraMasUsada = 0;
    // Calcula la palabra mas usada.
    for( int i = 0; i < lista.size(); i++ ) {
      String palabra = ( String ) lista.get( i ).getKey();
      int numVeces = ( Integer ) lista.get( i ).getValue();
      if( i == 0 ) {
        palabraMasUsada = palabra;
        numVecesPalabraMasUsada = numVeces;
      } else if( numVecesPalabraMasUsada < numVeces ) {
        palabraMasUsada = palabra;
        numVecesPalabraMasUsada = numVeces;
      }
    }
    // Imprime resultado.
    System.out.print( "( Palabra: '" + palabraMasUsada + "' " +
                        "veces: " + numVecesPalabraMasUsada + " )" );
  }

  // ---------------------------------------------------------------------
  static void printCuentaPalabrasOrdenadas(
                  HashMap<String,Integer> cuentaPalabras ) {
    int             i, numVeces;
    List<Map.Entry> list = new Vector<Map.Entry>( cuentaPalabras.entrySet() );

    // Ordena por valor.
    Collections.sort(
        list,
        new Comparator<Map.Entry>() {
          public int compare( Map.Entry e1, Map.Entry e2 ) {
            Integer i1 = ( Integer ) e1.getValue();
            Integer i2 = ( Integer ) e2.getValue();
            return i2.compareTo( i1 );
          }
```

```
        }
    );
    // Muestra contenido.
    i = 1;
    System.out.println( "Veces Palabra" );
    System.out.println( "-------------------" );
    for( Map.Entry e : list ) {
      numVeces = ( ( Integer ) e.getValue () ).intValue();
      System.out.println( i + " " + e.getKey() + " " + numVeces );
      i++;
    }
    System.out.println( "-------------------" );
  }
}
```

1 Realiza una implementación paralela con la colección `HashMap`. Recuerda que esta colección es `no sincronizada`.

Esta implementación junto a las dos secuenciales se deberá guardar en el fichero llamado `Ejercicio`. Para realizar análisis de costes equilibrados, debes asegurar que las versiones secuenciales no utilicen métodos sincronizados.

En el caso que necesites modificar el método `contabilizaPalabra`, crea una copia con otro nombre, para mantener el método que es utilizado en la versión secuencial.

Escribe a continuación el código que realiza tal tarea: la definición de la clase `MiHebra_1` y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

2 Realiza una implementación paralela con la colección `Hashtable`.

¿Sería posible reutilizar la clase `MiHebra_1` en este ejercicio? Razona tu respuesta.

Esta implementación junto a las dos secuenciales se deberá guardar en el fichero llamado `Ejer_2`. Cuando se valide su ejecución, añade el código correspondiente al fichero `Ejercicio`.

Escribe a continuación lel código que realiza tal tarea: la definición de la clase `MiHebra_2` y y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

3 Realiza una implementación paralela con la colección `ConcurrentHashMap` con un cerrojo adicional, es decir, empleando las cláusulas `synchronized` o `static synchronized`.

¿Sería posible reutilizar las clases `MiHebra_1` o `MiHebra_2` en este ejercicio? Razona tu respuesta.

Esta implementación junto a las dos secuenciales se deberá guardar en el fichero llamado `Ejer_3`. Cuando se valide su ejecución, añade el código correspondiente al fichero `Ejercicio`.

Escribe a continuación el código que realiza tal tarea: la definición de la clase `MiHebra_3` y y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

4 Realiza una implementación paralela con la colección `ConcurrentHashMap` y sin uso de cerrojos adicionales. En este caso, debes emplear los métodos `putIfAbsent`, `get` y `replace` para no tener que usar cerrojos adicionales.

¿Sería posible reutilizar alguna de las clases anteriores en este ejercicio? Razona tu respuesta.

Esta implementación junto a las dos secuenciales se deberá guardar en el fichero llamado `Ejer_4`. Cuando se valide su ejecución, añade el código correspondiente al fichero `Ejercicio`.

Escribe a continuación el código que realiza tal tarea: la definición de la clase `MiHebra_4` y y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

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

5 Realiza una implementación paralela con la colección `ConcurrentHashMap` y sin uso de cerrojos adicionales. En este caso, debes emplear los métodos `putIfAbsent` y `get`, y la clase `AtomicInteger`.

¿Sería posible reutilizar alguna de las clases anteriores en este ejercicio? Razona tu respuesta.

Esta implementación junto a las dos secuenciales se deberá guardar en el fichero llamado `Ejer_5`. Cuando se valide su ejecución, añade el código correspondiente al fichero `Ejercicio`.

Escribe a continuación el código que realiza tal tarea: la definición de la clase `MiHebra_5` y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

6 Realiza una implementación paralela idéntica a la del `Ejer_5` pero con mayor número de niveles de concurrencia. Emplea 256 niveles.

¿Sería posible reutilizar alguna de las clases anteriores en este ejercicio? Razona tu respuesta.

Esta implementación junto a las dos secuenciales se deberá guardar en el fichero llamado `Ejer_6`. Cuando se valide su ejecución, añade el código correspondiente al fichero `Ejercicio`.

Escribe a continuación el código que realiza tal tarea: la definición de la clase `MiHebra_6` y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

7 Realiza una implementación paralela basada en **Streams**.

Esta implementación junto a las dos secuenciales se deberá guardar en el fichero llamado `Ejer_7`. Cuando se valide su ejecución, añade el código correspondiente al fichero `Ejercicio`.

Seguidamente se muestra el código implementa esta operación mediante un **Parallel Streams**

```
Map<String,Long> stCuentaPalabras = vectorLineas.parallelStream()
                                    .filter( s -> s != null )
                                    .map( s -> s.split( "\\W+" ) )
                                    .flatMap( Arrays::stream )
                                    .map( String::trim )
                                    .filter( s -> (s.length() > 0) )
                                    .collect( groupingBy (s -> s, counting()));
```

8 Completa la siguiente tabla y justifica los resultados. Obtén los resultados para 4 hebras en tu ordenador local y los resultados para 16 hebras en patan. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

En ambas pruebas debes emplear el fichero `f3.txt`. Este fichero debe generarse en el script de lanzamiento utilizando el siguiente comando:

```
cat f1.txt f2.txt f1.txt f2.txt f1.txt f2.txt f1.txt f2.txt > f3.txt
```

y debe ser borrado al final del script.

	4 hebras		16 hebras	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela con `HashMap`				
Paralela con `Hashtable`				
Paralela con `ConcurrentHashMap` y con cerrojo adicional				
Paralela con `ConcurrentHashMap` y sin cerrojo adicional mediante `putIfAbsent`, `get` y `replace`				
Paralela con `ConcurrentHashMap` y sin cerrojo adiciona, mediante `putIfAbsent`, `get` y `AtomicInteger`				
Paralela con `ConcurrentHashMap` y sin cerrojo adicional mediante `putIfAbsent`, `get` y `AtomicInteger` y con más niveles				
`Parallel Stream`				

Justifica los resultados obtenidos.

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

9 Completa la siguiente tabla y justifica los resultados. Obtén los resultados para 4 hebras en tu ordenador local y los resultados para 16 hebras en patan. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

En ambas pruebas debes emplear el fichero `f4.txt`. Este fichero debe generarse en el script de lanzamiento utilizando el siguiente comando:

```
cat f1.txt f2.txt f1.txt f2.txt f1.txt f2.txt f1.txt f2.txt  > f3.txt
cat f3.txt f3.txt f3.txt f3.txt f3.txt f3.txt f3.txt f3.txt  > f4.txt
```

y debe ser borrado al final del script.

	4 hebras		16 hebras	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela con `HashMap`				
Paralela con `Hashtable`				
Paralela con `ConcurrentHashMap` y con cerrojo adicional				
Paralela con `ConcurrentHashMap` y sin cerrojo adicional mediante `putIfAbsent`, `get` y `replace`				
Paralela con `ConcurrentHashMap` y sin cerrojo adiciona, mediante `putIfAbsent`, `get` y `AtomicInteger`				
Paralela con `ConcurrentHashMap` y sin cerrojo adicional mediante `putIfAbsent`, `get` y `AtomicInteger` y con más niveles				
`Parallel Stream`				

Justifica los resultados obtenidos.

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

10 Completa la siguiente tabla y justifica los resultados. Obtén los resultados para 4 hebras en tu ordenador local y los resultados para 16 hebras en patan. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales.

En ambas pruebas debes emplear el fichero `f5.txt`. Este fichero debe generarse en el script de lanzamiento utilizando el siguiente comando:

```
cat f1.txt f2.txt f1.txt f2.txt f1.txt f2.txt f1.txt f2.txt  > f3.txt
cat f3.txt f3.txt f3.txt f3.txt f3.txt f3.txt f3.txt f3.txt  > f4.txt
cat f4.txt f4.txt f4.txt f4.txt f4.txt f4.txt f4.txt f4.txt  > f5.txt
```

y debe ser borrado al final del script.

	4 hebras		16 hebras	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela con `HashMap`				
Paralela con `Hashtable`				
Paralela con `ConcurrentHashMap` y con cerrojo adicional				
Paralela con `ConcurrentHashMap` y sin cerrojo adicional mediante `putIfAbsent`, `get` y `replace`				
Paralela con `ConcurrentHashMap` y sin cerrojo adiciona, mediante `putIfAbsent`, `get` y `AtomicInteger`				
Paralela con `ConcurrentHashMap` y sin cerrojo adicional mediante `putIfAbsent`, `get` y `AtomicInteger` y con más niveles				
`Parallel Stream`				

Justifica los resultados obtenidos.

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