Nombre y apellido: Enric Gil Gallen **Tiempo**: ---

Nombre y apellido: Victor Granados Segara

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.

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

maCuentaPalabras = new HashMap<String, Integer>(1000, 0.75F);
Map<String, Integer> maCuentaPalabrasSynchronized =
Collections.synchronizedMap(maCuentaPalabras);

MiHebra_1[] h1 = new MiHebra_1[numHebras];

for (int i = 0; i < numHebras; i++) {
   h1[i] = new MiHebra_1(i, numHebras, maCuentaPalabrasSynchronized, vectorLineas);
   h1[i].start();
}

try {
   for (int i = 0; i < numHebras; i++) {
       h1[i].join();
   }
} catch (InterruptedException e) {
   e.printStackTrace();
}

t2 = System.nanoTime();
t1SynchronizedMap = ((double) (t2 - t1)) / 1.0e9;
System.out.print("Implemen. paralela 1: ");
imprimePalabraMasUsadaYVeces(maCuentaPalabras);
System.out.println(" Tiempo(s): " + t1SynchronizedMap + " , Incremento " + tSecuencial /
t1SynchronizedMap);
System.out.println("Num. elems. tabla hash: " + maCuentaPalabras.size());
System.out.println();



public static synchronized void contabilizaPalabraSynchronized(Map<String, Integer>
cuentaPalabras, String palabra) {
   Integer numVeces = cuentaPalabras.get( palabra );
   if( numVeces != null ) {
```

```
            cuentaPalabras.put( palabra, numVeces+1 );
        } else {
            cuentaPalabras.put( palabra, 1 );
        }
    }
```

```
package e6;

import java.util.Map;
import java.util.Vector;

import static e6.EjemploPalabraMasUsada1a.contabilizaPalabraSynchronized;

public class MiHebra_1 extends Thread {
    int miId;
    int numHebras;
    Map<String,Integer> maCuentaPalabras;
    Vector<String> vectorLineas;
    String palabraActual;

    public MiHebra_1(int miId, int numHebras, Map<String, Integer> maCuentaPalabras ,
Vector<String> vectorLineas) {
        this.miId = miId;
        this.numHebras = numHebras;
        this.maCuentaPalabras = maCuentaPalabras;
        this.vectorLineas = vectorLineas;
    }

    public void run(){
        for( int i = miId; i < vectorLineas.size(); i+= numHebras ) {
            // 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 blancos.
                palabraActual = palabras[ j ].trim();
                if( palabraActual.length() >= 1 ) {
                    contabilizaPalabraSynchronized( maCuentaPalabras, palabraActual );
                }
            }
        }
    }


}
```

2 Realiza una implementación paralela con la colección **`Hashtable`**.
¿Sería posible reutilizar la clase **`MiHebra_1`** en este ejercicio? Razona tu respuesta.

Si ya que al ser una colección no sincronizada convertida como sincronizada no habría ningún problema.

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.

```
package e6;

import java.util.Hashtable;
import java.util.Vector;

import static e6.EjemploPalabraMasUsada1a.contabilizaPalabraHashtable;

public class MiHebra_2 extends Thread{
   int miId;
   int numHebras;
   Hashtable<String,Integer> htCuentaPalabras;
   Vector<String> vectorLineas;
   String palabraActual;

   public MiHebra_2(int miId, int numHebras, Hashtable<String, Integer>
htCuentaPalabras, Vector<String> vectorLineas) {
       this.miId = miId;
       this.numHebras = numHebras;
       this.htCuentaPalabras = htCuentaPalabras;
       this.vectorLineas = vectorLineas;
   }

   public void run(){
       for( int i = miId; i < vectorLineas.size(); i+= numHebras ) {
           // 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 blancos.
               palabraActual = palabras[ j ].trim();
               if( palabraActual.length() >= 1 ) {
                   contabilizaPalabraHashtable(htCuentaPalabras, palabraActual);
               }
           }
       }
   }


}

//
// Implementacion paralela 2: Uso de Hashtable.
//
t1 = System.nanoTime();

htCuentaPalabras = new Hashtable<>(1000, 0.75F);
MiHebra_2[] h2 = new MiHebra_2[numHebras];

for (int i = 0; i < numHebras; i++) {
   h2[i] = new MiHebra_2(i, numHebras, htCuentaPalabras, vectorLineas);
   h2[i].start();
```

```
}

try {
   for (int i = 0; i < numHebras; i++) {
       h2[i].join();
   }
} catch (InterruptedException e) {
   e.printStackTrace();
}

t2 = System.nanoTime();
t2Hashtable = ((double) (t2 - t1)) / 1.0e9;
System.out.print("Implemen. paralela 2: ");
imprimePalabraMasUsadaYVeces(htCuentaPalabras);
System.out.println(" Tiempo(s): " + t2Hashtable + " , Incremento " + tSecuencial /
t2Hashtable);
System.out.println("Num. elems. tabla hash: " + maCuentaPalabras.size());
System.out.println();
```

3 Realiza una implementación paralela con la colección **ConcurrentHashMap** con un cerrojo adicional, es decir, empleando la palabra **synchronized**.

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

En la primera hebra no habría problema pero en la segunda no se podría puesto que al ser de tipo específico HashTable no podemos reutilizarla.

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.

```
//
// Implementacion paralela 3: Uso de ConcurrentHashMap
//
t1 = System.nanoTime();

chCuentaPalabras = new ConcurrentHashMap<>(1000, 0.75F);
MiHebra_3[] h3 = new MiHebra_3[numHebras];

for (int i = 0; i < numHebras; i++) {
   h3[i] = new MiHebra_3(i, numHebras, chCuentaPalabras, vectorLineas);
   h3[i].start();
}

try {
   for (int i = 0; i < numHebras; i++) {
       h3[i].join();
   }
} catch (InterruptedException e) {
```

```
   e.printStackTrace();
}

t2 = System.nanoTime();
t3ConcurrentHashMap = ((double) (t2 - t1)) / 1.0e9;
System.out.print("Implemen. paralela 3: ");
imprimePalabraMasUsadaYVeces(chCuentaPalabras);
System.out.println(" Tiempo(s): " + t3ConcurrentHashMap + " , Incremento " + tSecuencial
/ t3ConcurrentHashMap);
System.out.println("Num. elems. tabla hash: " + chCuentaPalabras.size());
System.out.println();
```

```
package e6;

import java.util.Hashtable;
import java.util.Vector;
import java.util.concurrent.ConcurrentHashMap;

import static e6.EjemploPalabraMasUsada1a.contabilizaPalabraConcurrentHashMap;

public class MiHebra_3 extends Thread{
   int miId;
   int numHebras;
   final ConcurrentHashMap<String,Integer> chCuentaPalabras;
   Vector<String> vectorLineas;
   String palabraActual;

   public MiHebra_3(int miId, int numHebras, ConcurrentHashMap<String, Integer>
chCuentaPalabras, Vector<String> vectorLineas) {
       this.miId = miId;
       this.numHebras = numHebras;
       this.chCuentaPalabras = chCuentaPalabras;
       this.vectorLineas = vectorLineas;
   }

   public void run(){
       for( int i = miId; i < vectorLineas.size(); i+= numHebras ) {
           // 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 blancos.
               palabraActual = palabras[ j ].trim();
               if( palabraActual.length() >= 1 ) {
                   contabilizaPalabraConcurrentHashMap(chCuentaPalabras, palabraActual);
               }
           }
       }
   }


}
```

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.

Se podría reutilizar la 1 y la 3 ya que es ConcurrentHashMap.

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.

```
package e6;

import java.util.Vector;
import java.util.concurrent.ConcurrentHashMap;

import static e6.EjemploPalabraMasUsada1a.contabilizaPalabraConcurrentHashMapEje4;

public class MiHebra_4 extends Thread{
   int miId;
   int numHebras;
   final ConcurrentHashMap<String,Integer> chCuentaPalabras;
   Vector<String> vectorLineas;
   String palabraActual;

   public MiHebra_4(int miId, int numHebras, ConcurrentHashMap<String, Integer>
chCuentaPalabras, Vector<String> vectorLineas) {
       this.miId = miId;
       this.numHebras = numHebras;
       this.chCuentaPalabras = chCuentaPalabras;
       this.vectorLineas = vectorLineas;
   }

   public void run(){
       for( int i = miId; i < vectorLineas.size(); i+= numHebras ) {
           // 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 blancos.
               palabraActual = palabras[ j ].trim();
               if( palabraActual.length() >= 1 ) {
                   contabilizaPalabraConcurrentHashMapEje4(chCuentaPalabras,
palabraActual);
               }
           }
       }
   }
```

```
}
//
// Implementacion paralela 4: Uso de ConcurrentHashMap
//
t1 = System.nanoTime();

chCuentaPalabras = new ConcurrentHashMap<>(1000, 0.75F);
MiHebra_4[] h4 = new MiHebra_4[numHebras];

for (int i = 0; i < numHebras; i++) {
   h4[i] = new MiHebra_4(i, numHebras, chCuentaPalabras, vectorLineas);
   h4[i].start();
}

try {
   for (int i = 0; i < numHebras; i++) {
       h4[i].join();
   }
} catch (InterruptedException e) {
   e.printStackTrace();
}

t2 = System.nanoTime();
t4ConcurrentHashMap = ((double) (t2 - t1)) / 1.0e9;
System.out.print("Implemen. paralela 4: ");
imprimePalabraMasUsadaYVeces(chCuentaPalabras);
System.out.println(" Tiempo(s): " + t4ConcurrentHashMap + " , Incremento " + tSecuencial
/ t4ConcurrentHashMap);
System.out.println("Num. elems. tabla hash: " + chCuentaPalabras.size());
System.out.println();
```

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.

No ya que estamos usando clases Atómicas

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.

```
//
// Implementacion paralela 5: Uso de ConcurrentHashMap
//
t1 = System.nanoTime();

chaCuentaPalabras = new ConcurrentHashMap<>(1000, 0.75F);
MiHebra_5[] h5 = new MiHebra_5[numHebras];
```

```
for (int i = 0; i < numHebras; i++) {
   h5[i] = new MiHebra_5(i, numHebras, chaCuentaPalabras, vectorLineas);
   h5[i].start();
}

try {
   for (int i = 0; i < numHebras; i++) {
       h5[i].join();
   }
} catch (InterruptedException e) {
   e.printStackTrace();
}

t2 = System.nanoTime();
t5ConcurrentHashMapAtomica = ((double) (t2 - t1)) / 1.0e9;
System.out.print("Implemen. paralela 5: ");
imprimePalabraMasUsadaYVecesAtomicas(chaCuentaPalabras);
System.out.println(" Tiempo(s): " + t5ConcurrentHashMapAtomica + " , Incremento " +
tSecuencial / t5ConcurrentHashMapAtomica);
System.out.println("Num. elems. tabla hash: " + chaCuentaPalabras.size());
System.out.println();

package e6;

import java.util.Vector;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;

import static e6.EjemploPalabraMasUsada1a.contabilizaPalabraConcurrentHashMapAtomica;
import static e6.EjemploPalabraMasUsada1a.contabilizaPalabraConcurrentHashMapEje4;

public class MiHebra_5 extends Thread{
   int miId;
   int numHebras;
   final ConcurrentHashMap<String, AtomicInteger> chaCuentaPalabras;
   Vector<String> vectorLineas;
   String palabraActual;

   public MiHebra_5(int miId, int numHebras, ConcurrentHashMap<String, AtomicInteger>
chaCuentaPalabras, Vector<String> vectorLineas) {
       this.miId = miId;
       this.numHebras = numHebras;
       this.chaCuentaPalabras = chaCuentaPalabras;
       this.vectorLineas = vectorLineas;
   }

   public void run(){
       for( int i = miId; i < vectorLineas.size(); i+= numHebras ) {
           // 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 blancos.
               palabraActual = palabras[ j ].trim();
               if( palabraActual.length() >= 1 ) {
                   contabilizaPalabraConcurrentHashMapAtomica(chaCuentaPalabras,
```

```
palabraActual);
                }
            }
        }
    }


}

public static void contabilizaPalabraConcurrentHashMapAtomica(ConcurrentHashMap<String,
AtomicInteger> cuentaPalabras, String palabra) {
    AtomicInteger numVeces;

    numVeces = cuentaPalabras.putIfAbsent(palabra, new AtomicInteger(1));
    if (numVeces != null) {
        numVeces.getAndIncrement();
    }
}
```

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.

Podemos utilizar la anterior puesto que es atómica.

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.

```
package e6;

import java.util.Vector;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;

import static e6.EjemploPalabraMasUsada1a.contabilizaPalabraConcurrentHashMapAtomica;

public class MiHebra_6 extends Thread{
    int miId;
    int numHebras;
    final ConcurrentHashMap<String, AtomicInteger> chaCuentaPalabras;
    Vector<String> vectorLineas;
    String palabraActual;

    public MiHebra_6(int miId, int numHebras, ConcurrentHashMap<String, AtomicInteger>
chaCuentaPalabras, Vector<String> vectorLineas) {
        this.miId = miId;
        this.numHebras = numHebras;
        this.chaCuentaPalabras = chaCuentaPalabras;
        this.vectorLineas = vectorLineas;
    }
```

```
   public void run(){
       for( int i = miId; i < vectorLineas.size(); i+= numHebras ) {
           // 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 blancos.
               palabraActual = palabras[ j ].trim();
               if( palabraActual.length() >= 1 ) {
                   contabilizaPalabraConcurrentHashMapAtomica(chaCuentaPalabras,
palabraActual);
               }
           }
       }
   }


}

//
// Implementacion paralela 6: Uso de ConcurrentHashMap
//
t1 = System.nanoTime();

chaCuentaPalabras = new ConcurrentHashMap<>(1000, 0.75F, 256);
MiHebra_6[] h6 = new MiHebra_6[numHebras];

for (int i = 0; i < numHebras; i++) {
   h6[i] = new MiHebra_6(i, numHebras, chaCuentaPalabras, vectorLineas);
   h6[i].start();
}

try {
   for (int i = 0; i < numHebras; i++) {
       h6[i].join();
   }
} catch (InterruptedException e) {
   e.printStackTrace();
}

t2 = System.nanoTime();
t6ConcurrentHashMapAtomica = ((double) (t2 - t1)) / 1.0e9;
System.out.print("Implemen. paralela 6: ");
imprimePalabraMasUsadaYVecesAtomicas(chaCuentaPalabras);
System.out.println(" Tiempo(s): " + t6ConcurrentHashMapAtomica + " , Incremento " +
tSecuencial / t6ConcurrentHashMapAtomica);
System.out.println("Num. elems. tabla hash: " + chaCuentaPalabras.size());
System.out.println();
```

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.

	4 hebras		16 hebras	
	Tiempo	**Incremento**	**Tiempo**	**Incremento**
Secuencial	0.469	-	0.151	-
HashMap	0.520	0.9	0.655	0.231
Hashtable	0.339	1.382	0.659	0.229
ConcurrentHashMap	0.445	1.055	0.667	0.227
ConcurrentHashMap con putIfAbsent y replace	0.216	2.16	0.323	0.468
ConcurrentHashMap con putIfAbsent y AtomicInteger	0.176	2.669	0.286	0.528
ConcurrentHashMap con putIfAbsent y AtomicInteger y con niveles	0.199	2.354	0.278	0.544
Parallel Stream	0.441	1.065	0.395	0.383

En el caso de **HashMap y Hashtable** tenemos un incremento bajo ya que al tratarse de un fichero corto el coste de los accesos a los cerrojos es muy alto. Algo parecido le pasa a **ConcurrentHashMap** ya que al tener el cerrojo adicional también hace que se ralentice. Por lo que respecta al resto se observa una gran mejoría dado a que no tienen cerrojos adicionales. Como ganador tenemos ConcurrentHashMap **con putIfAbsent y AtomicInteger** gracias a la optimización de las clases atómicas.

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	1.454	-	1.036	
HashMap	1.686	0.8662	4.944	0.209
Hashtable	1.894	0.786	4.659	0.222
ConcurrentHashMap	2.619	0.555	4.452	0.232
ConcurrentHashMap con putIfAbsent y replace	0.641	2.269	0.884	1.171
ConcurrentHashMap con putIfAbsent y AtomicInteger	0.495	2.929	0.864	1.199
ConcurrentHashMap con putIfAbsent y AtomicInteger y con niveles	0.495	2.935	0.743	1.395
Parallel Stream	0.818	1.776	0.476	2.177

Pasa lo mismo que en el caso anterior los **ConcurrentHashMap** sin cerrojos adicionales son mucho más eficientes, en este caso al tener mas trabajo el **ConcurrentHashMap con putIfAbsent y AtomicInteger y con niveles** consigue más rendimiento porque divide la faena en tareas más pequeñas.

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	19.516	-	7.843	-
HashMap	22.44	0.869	38.668	0.203
Hashtable	22.812	0.855	36.512	0.218
ConcurrentHashMap	20.068	0.972	34.423	0.228
ConcurrentHashMap con putIfAbsent y replace	7.756	2.516	6.769	1.158
ConcurrentHashMap con putIfAbsent y AtomicInteger	6.296	3.009	5.629	1.393
ConcurrentHashMap con putIfAbsent y AtomicInteger y con niveles	6.484	3.00	6.009	1.305
Parallel Stream	4.375	4.467	1.874	4.246

Ocurre lo mismo que en los casos anteriores lo único que ahora quien despunta un poco más es el **Parallel Stream .**