

**IES AUGUSTO GONZALEZ DE LINARES**

**DEPARTAMENTO DE INFORMATICA**

GESTIÓN DE PROCESOS PARA TAREAS EN PARALELO

**PROGRAMACIÓN DE SERVICIOS Y PROCESOS**

**GRADO SUPERIOR DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA**

2023/2024

**Díez de Paulino, Albano**

Índice

[1. Demostración de la funcionalidad de la aplicación 2](#_Toc147518590)

[2. Comparación de tiempos entre la ejecución secuencial y paralela 7](#_Toc147518591)

[3. Conclusiones 10](#_Toc147518592)

[Tabla de Ilustraciones 11](#_Toc147518593)

[Bibliografía 11](#_Toc147518594)

# Demostración de la funcionalidad de la aplicación

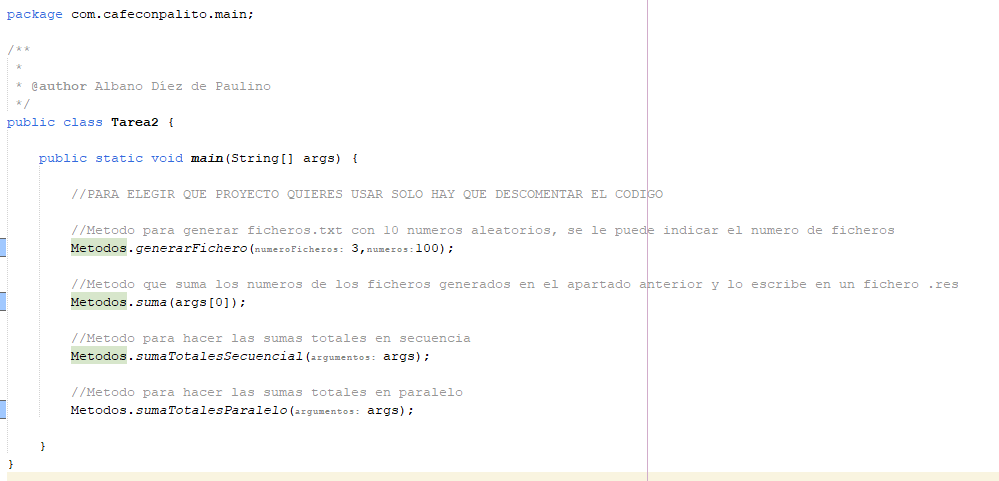
Diagrama

Descripción generada automáticamentePara aprender a manejar la programación de procesos en paralelo se requiere que se realice la siguiente estructura de programas.

Ilustración 1 – Estructura de procesos planteada

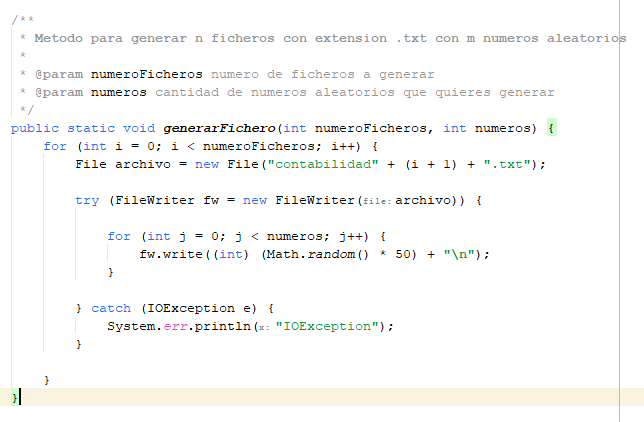
Visto el problema que se plantea, la solución mas optima es realizar una clase que contenga el método de acceso Main desde el cual se llamaran a los 3 programas propuestos, que se encontraran encapsulados en diferentes métodos.

En mi caso he decidió crear una clase extra para que solo tenga los 3 métodos, y los declaro estáticos para no tener que declarar un objeto de la clase extra.



Código 1 - Método Main con llamada a los métodos

* **Método** generarFichero (int numerosFicheros, int números)

Método para generar n ficheros con extensión .txt con m números aleatorios, para ello se le pasa por parámetros el numero de ficheros a generar (primer parámetro) y la cantidad de números aleatorios que se quieren generar.

Código 2 - Método generarFichero

* **Método** suma (String nombre del fichero)

Método que suma los números del fichero que le pases como parámetro y genera un fichero .res

Código 3 - Método suma

Para ello se ha usado la clase FileReader que te permite leer carácter a carácter un fichero de texto, ya que leer de esa forma es poco lógica, se usa la clase BufferedReader, ya que esta clase crea un buffer para poder leer línea a línea, realizada la lectura solo queda escribir en otro fichero la suma parcial usando la clase FileWriter.

• **Método** sumaTotalesSecuencial(String[] argumentos)

Método para sumar las sumas Parciales del método suma, de forma secuencial.

Se le pasa por parámetro un array de String con el nombre de los ficheros .res generados con el método anterior.

Para lanzar los subprocesos desde este método se debe generar un .jar que solo realice el método suma, para ello desde el Main solo se debe llamar al método deseado.



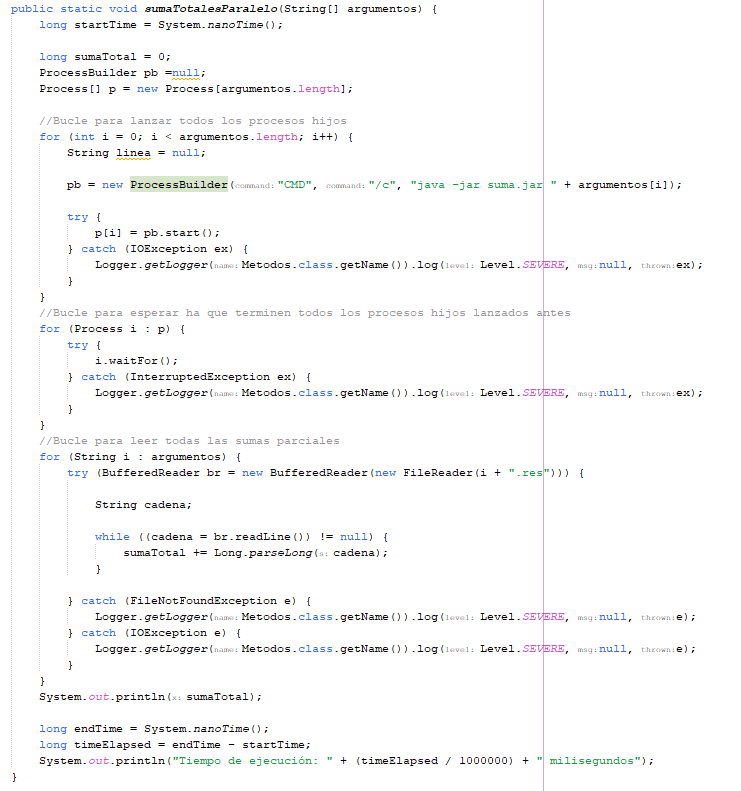
Código 4 - Método sumaTotalesSecuencial

* **Método** sumaTotalesParalelo (String[] argumentos)

Método para sumar las sumas Parciales del método suma de forma paralela.

Se le pasa por parámetro un array de String con el nombre de los ficheros .res generados con el método suma.

Al igual que en el método anterior hay que generar un .jar que solo realice el método suma.



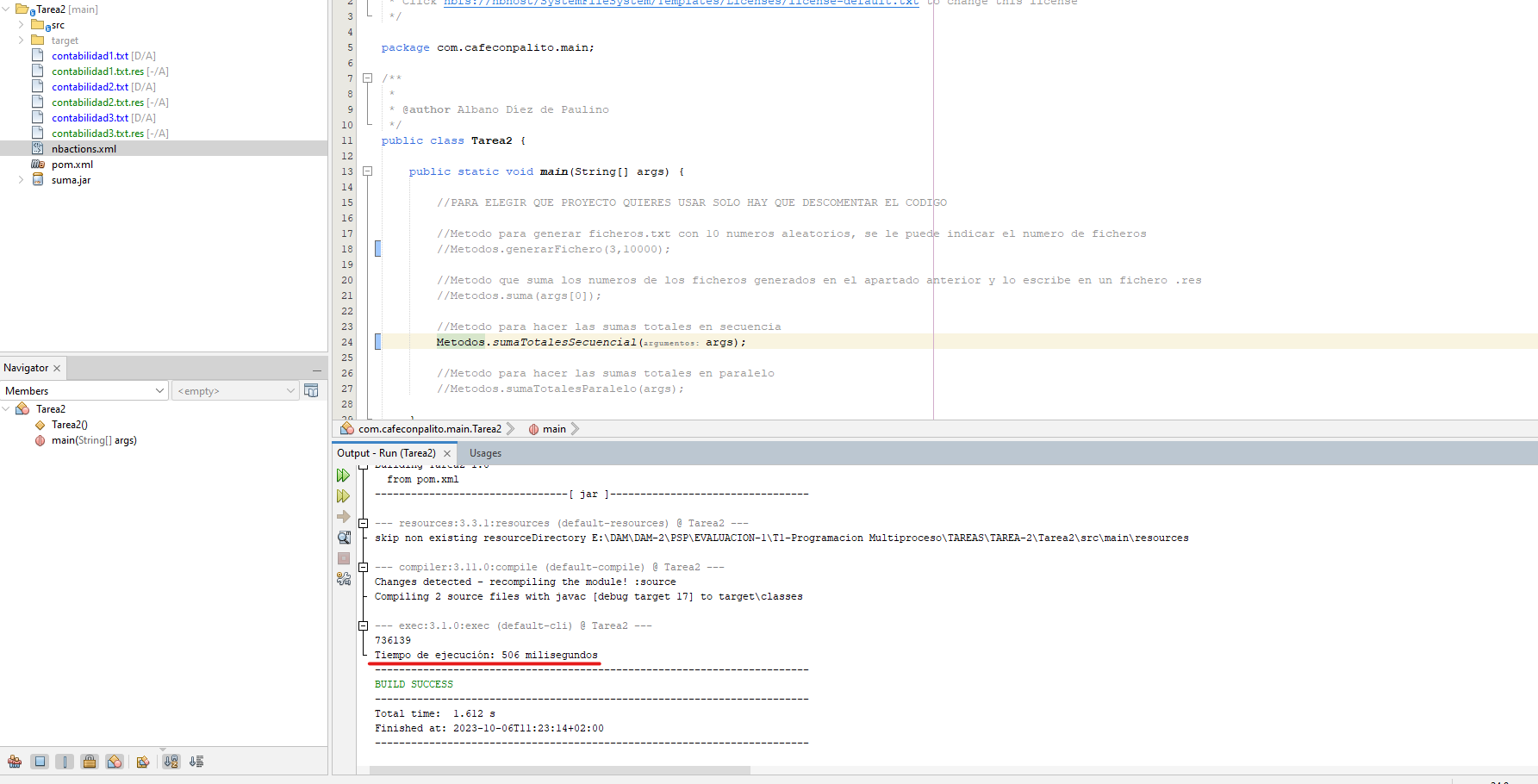
Código 5 – Método sumaTotalesParalelo

# Comparación de tiempos entre la ejecución secuencial y paralela

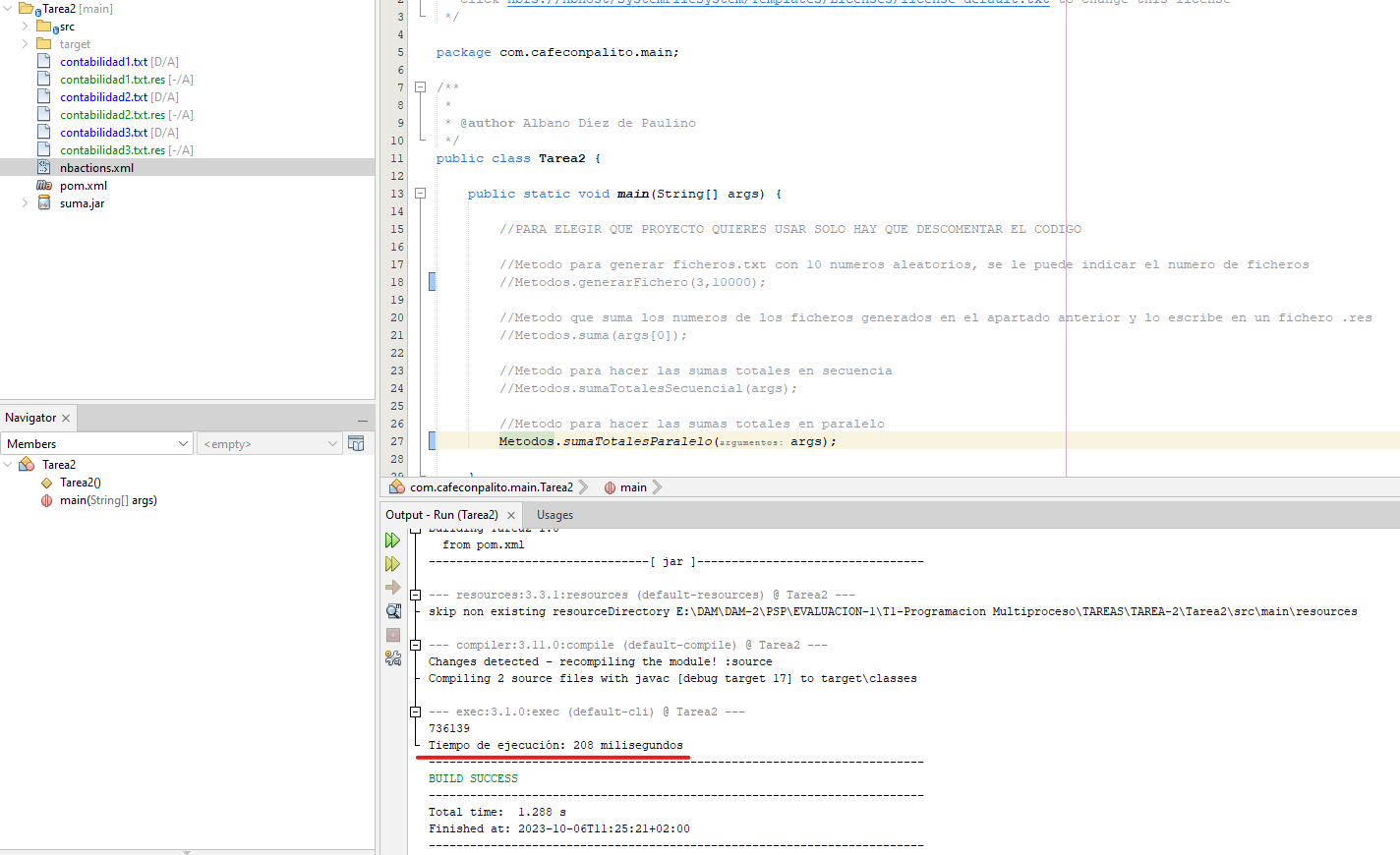
Tras la realización del código se ha procedido a la comprobación del correcto funcionamiento de este, tras corregir pequeños fallos, por ejemplo declarar las sumas parciales y total como tipo int provoco que al tratar de sumar una cifra de números muy elevada, las variables desbordaran y dieran sumas falsas, se ha procedido a comparar los tiempos de ejecución entre lanzar los subprocesos de forma secuencial (*Lanzar uno y esperar a lanzar el siguiente*) y lanzar los subprocesos de forma paralela (*Lanzar todos los a la vez y esperar hasta que terminen todos*).

Desde un inicio se decidió usar unos ficheros de texto con 10.000 números ya que realizar la suma de menos números, el ordenador la iba a realizar en unidades más pequeñas que milisegundos y no sería útil para comparar tiempos.

Los tiempos con 10.000 números en tres ficheros son los siguientes:

* **Secuencial**: 506 milisegundos
* **Paralelo**: 208 milisegundos

Código 6 – Tiempo de ejecución 10K números de forma secuencial

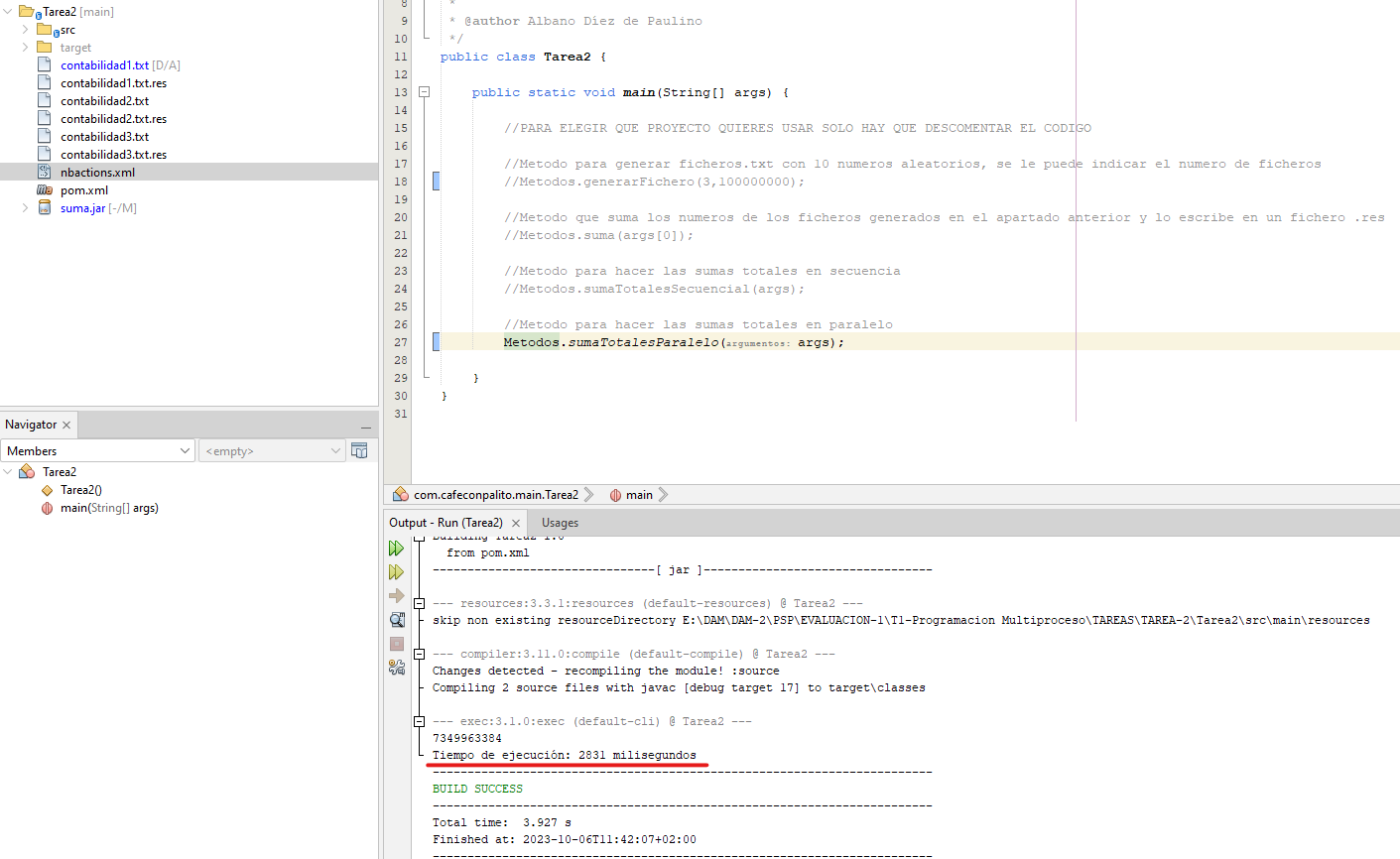


Código 7 – Tiempo de ejecución 10K números de forma paralela

Es evidente con esta prueba que la ejecución paralela es más rápida que secuencial, pero se decidió realizar la misma prueba, pero con 100 Millones de números por fichero, para descubrir si escalando la carga de trabajo se mantiene la premisa.

Los tiempos que salieron de esta prueba fueron los siguientes:

* **Secuencial**: 6326 milisegundos
* **Paralelo**: 2831 milisegundos

Una captura de pantalla de una computadora

Descripción generada automáticamente

Código 8 - Tiempo de ejecución 100 M números de forma secuencial

Código 9 - Tiempo de ejecución 100 M números de forma Paralela

# Conclusiones

Tras las pruebas realizadas se ha llegado a las siguientes conclusiones:

* Si se desea realizar que el programa tarde lo menos posible en ejecutar múltiples tareas, la mejor opción es realizarlas de forma paralela, antes que secuencial.
* Lanzar procesos de forma paralela es más difícil ya que se debe coordinar todos los procesos.
* Lanzar procesos de forma secuencial no requiere mucha coordinación.
* No se debe lanzar procesos con mucha carga de trabajo a la vez en ordenador con procesadores lentos, ya que puedes saturar el sistema.

# Tabla de Ilustraciones

[Ilustración 1 – Estructura de procesos planteada 2](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518150)

[Código 1 - Método Main con llamada a los métodos 3](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518352)

[Código 2 - Método generarFichero 3](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518353)

[Código 3 - Método suma 4](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518354)

[Código 4 - Método sumaTotalesSecuencial 5](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518355)

[Código 5 – Método sumaTotalesParalelo 6](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518356)

[Código 6 – Tiempo de ejecución 10K números de forma secuencial 7](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518357)

[Código 7 – Tiempo de ejecución 10K números de forma paralela 8](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518358)

[Código 8 - Tiempo de ejecución 100 M números de forma secuencial 9](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518359)

[Código 9 - Tiempo de ejecución 100 M números de forma Paralela 9](file:///C:\Users\carra\Documents\DAM\DAM-2\PSP\EVALUACION-1\T1-Programacion%20Multiproceso\TAREAS\TAREA-2\Tarea2.docx#_Toc147518360)

# Bibliografía

<https://www.techiedelight.com/es/measure-elapsed-time-execution-time-java/>

[ProcessBuilder (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html)

[BufferedReader (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html)

[FileReader (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/io/FileReader.html)

[FileWriter (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/io/FileWriter.html)

[Process (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/lang/Process.html#waitFor--)