



IES AUGUSTO GONZALES DE LINARES
CFGS DAM2º

Gestión de Procesos en Paralelo y Secuencial

Programación De Servicios Y Procesos

Daniel Espinosa García

2023/2024





Índice

Demostración de la funcionalidad de la aplicación con capturas de pantalla.	2
Comparación de tiempos entre la ejecución secuencial y paralela. Para medir tiempos de ejecución	5
Conclusión.....	7
Bibliografía.....	8

Demostración de la funcionalidad de la aplicación con capturas de pantalla.

Para realizar la practica genere varios metodos que ejecutan cada uno de los requerimientos exigidos en la práctica y las llamadas de estos desde el Main.

Método Generar ficheros:

Este método que recibe un entero para saber el número de ficheros a generar la cantidad de números que este generara en el fichero.

```
public class Metodos {

    /**
     * Metodo que recibe un entero para saber el numero de ficheros a generar y
     * la cantidad de numeros que este generara en el fichero.
     * @param numFicheros numero de ficheros que se quieren crear
     * @param cantNumeros cantidad de numeros que se quieren añadir
     */
    public static void generadorDeFicheros(int numFicheros, int cantNumeros) {

        String nombreFichero = null;

        for (int i = 0; i < numFicheros; i++) {

            nombreFichero = "contabilidad" + (i + 1) + ".txt";

            File archivo = new File(pathname: nombreFichero);
            if (archivo.exists()) {
                archivo.delete();
            }

            try (FileWriter fw = new FileWriter(file: archivo, append: true);) {

                for (int j = 0; j < cantNumeros; j++) {

                    //System.out.println((int) (Math.random() * 50));
                    fw.write((int) (Math.random() * 50) + "\n");

                }

            } catch (IOException ex) {
                Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
            }

        }

    }

}
```

Método suma:

Método que suma los números contenidos en un fichero. recibe por argumento el nombre del fichero el cual se quiere leer y genera un nuevo archivo con el nombre del fichero recibido más la extensión .res

```
public class Metodos {

    /** Metodo que recibe un entero para saber el numero de ficheros a generar y ...6 lines */
    public static void generadorDeFicheros(int numFicheros, int cantNumeros) { ...27 lines }

    /**
     * Metodo que suma los numeros contenidos en un fichero. recibe por
     * argumento el nombre del fichero el cual se quiere leer y genera un nuevo
     * archivo con el nombre del fichero recibido mas las extension .res
     * @param nombreFichero nombre del fichero que se quiere leer
     */
    public static void suma(String nombreFichero) {

        int suma = 0;
        String cadena = null;

        File fichero = new File(pathname: nombreFichero);
        File ficheroSuma = new File(nombreFichero + ".res");

        //si el fichero.res ya existe lo borra.
        if (ficheroSuma.exists()) {
            ficheroSuma.delete();
        }

        //Lectura del fichero recibido por argumento, leyendo linea a linea para calcular la suma total de ese fichero y generar el fichero .res
        try (BufferedReader br = new BufferedReader(new FileReader(file: nombreFichero)); FileWriter fw = new FileWriter(file: ficheroSuma, append: true)) {

            while ((cadena = br.readLine()) != null) {
                suma += Integer.parseInt(cadena);
            }

            //Muestra por pantalla la suma.
            System.out.println("La suma es: " + suma);

            //escribe en el fichero la el total de la suma.
            fw.write(suma + "");

        } catch (FileNotFoundException ex) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
        } catch (IOException ex) {
            Logger.getLogger(name: Metodos.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
        }

    }

}
```

Método sumaTotalSecuencial:

Realiza la suma total de manera secuencial de los ficheros que se deseen sumar pasados como argumento, utilizando un bucle for.

```
/**
 * Realiza la suma total de manera secuencial de los ficheros que se deseen
 * sumar pasados como argumento
 * @param argumentos recibe en un Array de Strings el nombre de los ficheros
 * que se quieren leer.
 */
public static void sumaTotalesSecuencial(String[] argumentos) {

    long startTime = System.nanoTime(); // Captura el tiempo al iniciar el proceso para controlar el tiempo de ejecucion
    int sumaTotal = 0;

    // por cada argumento pasado (nombre del fichero) inicializa el proceso y espera a que el proceso termine
    for (String argumento : argumentos) {
        try {
            //asignamos las variables para ejecutar el .Jar
            ProcessBuilder pb = new ProcessBuilder(command:"CMD", command:"/c", "java -jar suma.jar " + argumento);
            Process p = pb.start();
            //espera a que el proceso termine
            p.waitFor();

            String linea = null;

            //Lectura de los ficheros .res, leyendo linea a linea para calcular la suma total de todos los ficheros
            try (BufferedReader br = new BufferedReader(new FileReader(argumento + ".res"))) {
                while ((linea = br.readLine()) != null) {
                    sumaTotal += Integer.parseInt(linea);
                }
            } catch (Exception e) {
            }

        } catch (IOException e) {
            Logger.getLogger(Metodos.class.getName()).log(Level.SEVERE, msg:null, thrown: e);
        } catch (InterruptedException ex) {
            Logger.getLogger(Metodos.class.getName()).log(Level.SEVERE, msg:null, thrown: ex);
        }
    }

    System.out.println("El total de la suma es: " + sumaTotal); //muestra por pantalla la suma total de los ficheros.
    long endTime = System.nanoTime(); //Captura el tiempo al finalizar el proceso para controlar el tiempo de ejecucion
    long timeElapsed = endTime - startTime; //Calcula los nanosegundos transcurridos.
    System.out.println("Suma Secuencial, Tiempo de ejecucion en milisegundos: " + timeElapsed / 1000000); // muestra por pantalla los milisegundos transcurridos
}
```

Método sumaTotalParalela:

Realiza la suma total de manera paralela de los ficheros que se deseen sumar pasados como argumento. Se inicializan todos los procesos hijos dentro de un bucle for se espera a que todos los procesos hijos terminen dentro de otro bucle for se calculan las sumas de los totales dentro de otro bucle for.

```
/**
 * Realiza la suma total de manera paralela de los ficheros que se deseen
 * sumar pasados como argumento.
 * Se inicializan todos los procesos hijos dentro de un bucle for
 * Se espera a que todos los procesos hijos terminen dentro de otro bucle for
 * Se calculan las sumas de los totales dentro de otro bucle for.
 * @param argumentos recibe en un Array de Strings el nombre de los ficheros que se quieren leer.
 */
public static void sumaTotalesParalela(String[] argumentos) {

    long startTime = System.nanoTime(); // Captura el tiempo al iniciar el proceso para controlar el tiempo de ejecucion
    int sumaTotal = 0;

    ProcessBuilder pb = null;
    Process[] p = new Process[argumentos.length];
    //Lanza todos los procesos hijos para que comiencen a ejecutarse.
    for (int i = 0; i < argumentos.length; i++) {
        try {
            pb = new ProcessBuilder(command:"CMD", command:"/c", "java -jar suma.jar " + argumentos[i]);
            p[i] = pb.start();
        } catch (IOException e) {
        }
    }

    //Espera a que todos los hijos terminen. (espera a que el sumar.Jar termine para cada argumento y genere el fichero)
    for (Process i_p : p) {
        try {
            i_p.waitFor();
        } catch (InterruptedException ex) {
            Logger.getLogger(Metodos.class.getName()).log(Level.SEVERE, msg:null, thrown: ex);
        }
    }

    //Realiza la suma de ficheros .res generados para calcular el total
    for (String argumento : argumentos) {
        String linea = null;
        try (BufferedReader br = new BufferedReader(new FileReader(argumento + ".res"))) {
            while ((linea = br.readLine()) != null) {
                sumaTotal += Integer.parseInt(linea);
            }
        } catch (Exception e) {
        }
    }

    System.out.println("El total de la suma es: " + sumaTotal);
    long endTime = System.nanoTime(); //Captura el tiempo al finalizar el proceso para controlar el tiempo de ejecucion
    long timeElapsed = endTime - startTime; //Calcula los nanosegundos transcurridos.
    System.out.println("Suma Paralela, Tiempo de ejecucion en milisegundos: " + timeElapsed / 1000000); // muestra por pantalla los milisegundos transcurridos
}
```

Main:

Para comprobar su funcionamiento dentro del Main se llaman a los metodos que se necesitan usar.

```
public class PSP_Tema1_Tarea2 {

    public static void main(String[] args) {

        //Metodo para Generar los ficheros que se quieren sumar
        Metodos.generadorDeFicheros(numFicheros:3,cantNumeros:1000000);

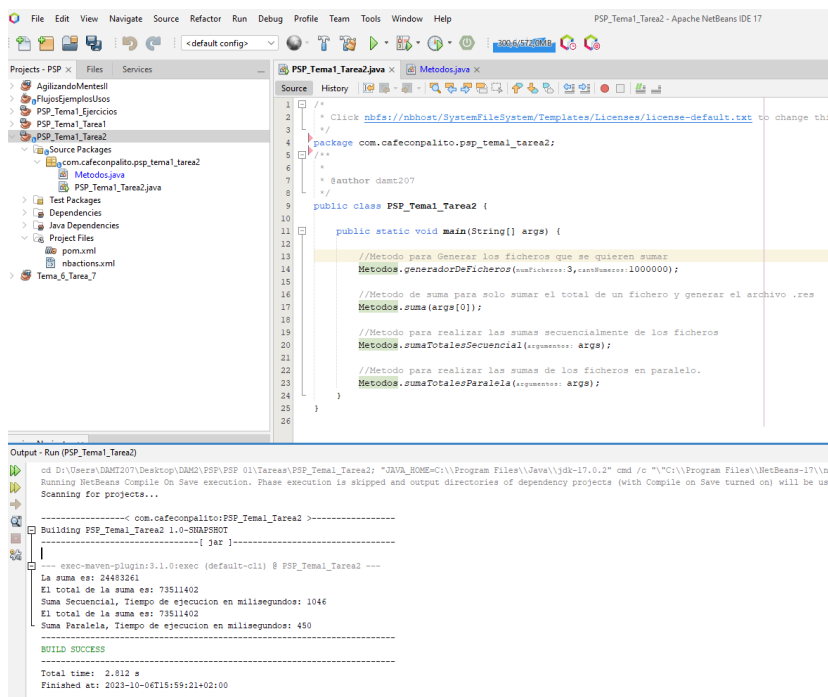
        //Metodo de suma para solo sumar el total de un fichero y generar el archivo .res
        Metodos.suma(args[0]);

        //Metodo para realizar las sumas secuencialmente de los ficheros
        Metodos.sumaTotalesSecuencial(argumentos: args);

        //Metodo para realizar las sumas de los ficheros en paralelo.
        Metodos.sumaTotalesParalela(argumentos: args);




    }
}
```

Ejecución de todos los métodos, para realizar una prueba inicial de que todo funciona correctamente.



Con todos los test pasados desde NetBeans activo solo el método que necesito para crear el .jar del módulo necesario utilizando el clean and build.

Obteniendo 3 .jar uno para cada método.

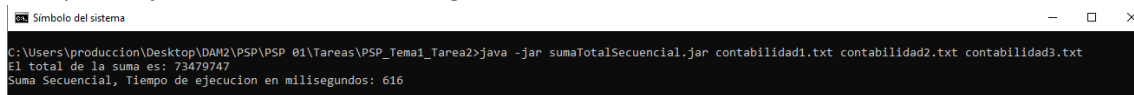
	suma.jar	06/10/2023 15:28	Executable Jar File	6 KB
	sumaTotalParalela.jar	06/10/2023 15:28	Executable Jar File	7 KB
	sumaTotalSecuencial.jar	06/10/2023 15:28	Executable Jar File	7 KB

Comparación de tiempos entre la ejecución secuencial y paralela. Para medir tiempos de ejecución

Con los .jar ya generados realizamos la ejecución de estos mismos para controlar los tiempos de ejecución.

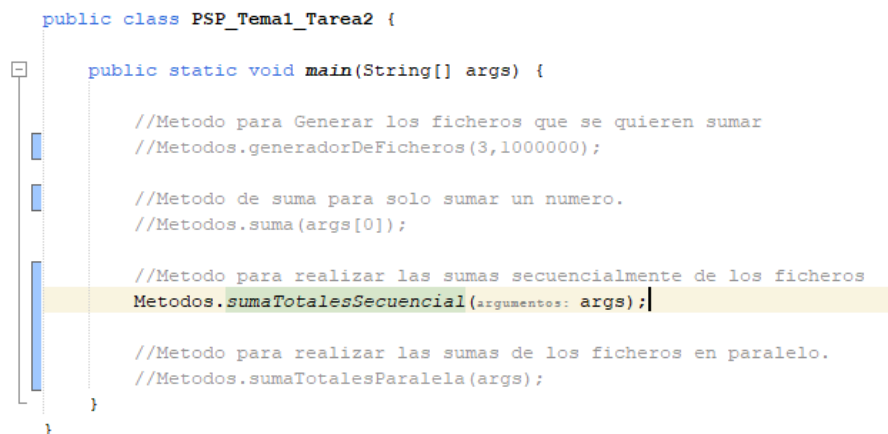
Tiempo de ejecución secuencial:

Mediante el CMD ejecuto el archivo sumaTotalSecuencial.jar, como se puede observar el tiempo de ejecución es de 616 milisegundos



```
Símbolo del sistema
C:\Users\produccion\Desktop\DAM2\PSP\PSP_01\Tareas\PSP_Tema1_Tarea2>java -jar sumaTotalSecuencial.jar contabilidad1.txt contabilidad2.txt contabilidad3.txt
El total de la suma es: 73479747
Suma Secuencial, Tiempo de ejecución en milisegundos: 616
```

Lanzando el mismo programa desde el NetBeans observamos que el tiempo de ejecución en secuencia es de 668 milisegundos.



```
public class PSP_Tema1_Tarea2 {

    public static void main(String[] args) {

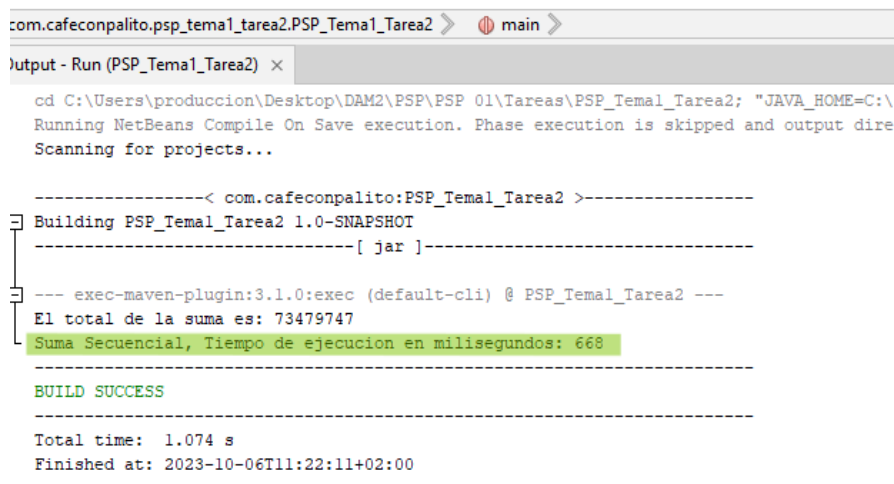
        //Metodo para Generar los ficheros que se quieren sumar
        //Metodos.generadorDeFicheros(3,1000000);

        //Metodo de suma para solo sumar un numero.
        //Metodos.suma(args[0]);

        //Metodo para realizar las sumas secuencialmente de los ficheros
        Metodos.sumaTotalesSecuencial(argumentos: args);

        //Metodo para realizar las sumas de los ficheros en paralelo.
        //Metodos.sumaTotalesParalela(args);

    }
}
```



```
com.cafeconpalito.psp_tema1_tarea2.PSP_Tema1_Tarea2 > main >
Output - Run (PSP_Tema1_Tarea2) x
cd C:\Users\produccion\Desktop\DAM2\PSP\PSP_01\Tareas\PSP_Tema1_Tarea2; "JAVA_HOME=C:\
Running NetBeans Compile On Save execution. Phase execution is skipped and output dire
Scanning for projects...

-----< com.cafeconpalito:PSP_Tema1_Tarea2 >-----
Building PSP_Tema1_Tarea2 1.0-SNAPSHOT
-----[ jar ]-----

--- exec-maven-plugin:3.1.0:exec (default-cli) @ PSP_Tema1_Tarea2 ---
El total de la suma es: 73479747
Suma Secuencial, Tiempo de ejecución en milisegundos: 668

BUILD SUCCESS

Total time: 1.074 s
Finished at: 2023-10-06T11:22:11+02:00
```

Tiempo de ejecución paralela:

Mediante el CMD ejecuto el archivo sumaTotalParalela.jar, como se puede observar el tiempo de ejecución es de 252 milisegundos.

```
C:\Users\produccion\Desktop\DAM2\PSP\PSP_01\Tareas\PSP_Tema1_Tarea2>java -jar sumaTotalParalela.jar contabilidad1.txt contabilidad2.txt contabilidad3.txt
El total de la suma es: 73479747
Suma Paralela, Tiempo de ejecucion en milisegundos: 252
C:\Users\produccion\Desktop\DAM2\PSP\PSP_01\Tareas\PSP_Tema1_Tarea2>_
```

Lanzando el mismo programa desde el NetBeans observamos que el tiempo de ejecución en secuencia es de 281 milisegundos.

```
public class PSP_Tema1_Tarea2 {

    public static void main(String[] args) {

        //Metodo para Generar los ficheros que se quieren sumar
        //Metodos.generadorDeFicheros(3,1000000);

        //Metodo de suma para solo sumar un numero.
        //Metodos.suma(args[0]);

        //Metodo para realizar las sumas secuencialmente de los ficheros
        //Metodos.sumaTotalesSecuencial(args);

        //Metodo para realizar las sumas de los ficheros en paralelo.
        Metodos.sumaTotalesParalela(argumentos: args);

    }

}
```

```
com.cafeconpalito.psp_tema1_tarea2.PSP_Tema1_Tarea2 > main >
Output - Run (PSP_Tema1_Tarea2) x
cd C:\Users\produccion\Desktop\DAM2\PSP\PSP_01\Tareas\PSP_Tema1_Tarea2; "JAVA_HOME=C:\\
Running NetBeans Compile On Save execution. Phase execution is skipped and output direc
Scanning for projects...

-----< com.cafeconpalito:PSP_Tema1_Tarea2 >-----
] Building PSP_Tema1_Tarea2 1.0-SNAPSHOT
-----[ jar ]-----

] --- exec-maven-plugin:3.1.0:exec (default-cli) @ PSP_Tema1_Tarea2 ---
El total de la suma es: 73479747
- Suma Paralela, Tiempo de ejecucion en milisegundos: 281

BUILD SUCCESS

Total time: 0.642 s
Finished at: 2023-10-06T11:23:22+02:00
-----
```

Conclusión

Como se puede observar por los tiempos obtenidos para la ejecución Secuencial de 616 milisegundos y la ejecución Paralela de 252 milisegundos. Para analizar 3 archivos con un millón de números cada uno. Estos tiempos son bastante diferentes, por lo cual trabajar de manera paralela es mucho más rápida para realizar esta tarea, ya que la en Secuencial tiene que esperar a que termine la ejecución de cada Hijo, suma.jar, para pasar al siguiente.

En paralelo se ejecutan todos los Hijos, sumar.jar, a la vez y solo tiene que esperar a que cada hijo termine para empezar a realizar las sumas totales.

```
Símbolo del sistema
C:\Users\produccion\Desktop\DAM2\PSP\PSP_01\Tareas\PSP_Tema1_Tarea2>java -jar sumaTotalSecuencial.jar contabilidad1.txt contabilidad2.txt contabilidad3.txt
El total de la suma es: 73479747
Suma Secuencial, Tiempo de ejecucion en milisegundos: 616
C:\Users\produccion\Desktop\DAM2\PSP\PSP_01\Tareas\PSP_Tema1_Tarea2>java -jar sumaTotalParalela.jar contabilidad1.txt contabilidad2.txt contabilidad3.txt
El total de la suma es: 73479747
Suma Paralela, Tiempo de ejecucion en milisegundos: 252
C:\Users\produccion\Desktop\DAM2\PSP\PSP_01\Tareas\PSP_Tema1_Tarea2>_
```

Se realizaron varias pruebas con la cantidad de números que contenían los archivos para comprobar los tiempos de ejecución, pasando de 10, 100, 1000, 10000 y 1000000. Al final opte por trabajar con un millón números.



Bibliografía

- Teoría del Módulo de Programación De Servicios Y Procesos que se nos facilita por Moodle.
- Método para poder calcular los milisegundos obtenido en el enunciado de la tarea
Link: <https://www.techiedelight.com/es/measure-elapsed-time-execution-time-java/>