

Presentado por: Carlos Alberto Sorza Gómez

ARSW 2024-1

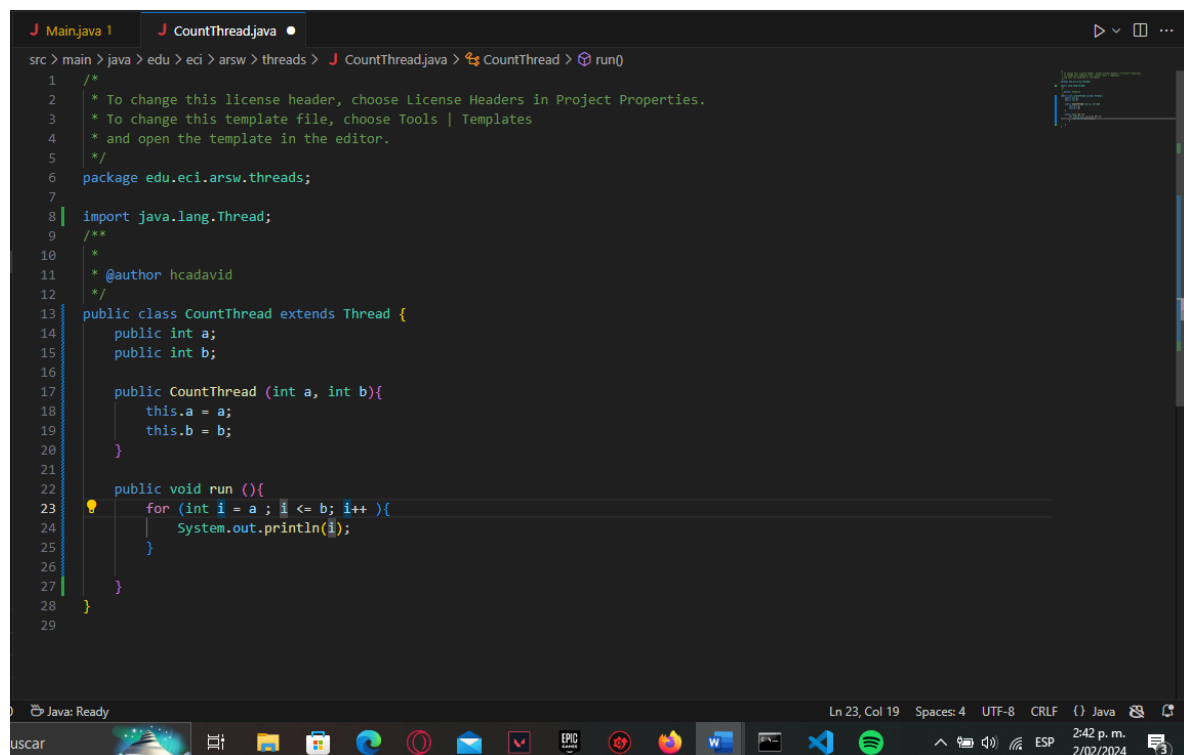
URL Repositorio Git: <https://github.com/CarlosSorza/ARSW-LAB01.git>

## Introducción

Los hilos en Java posibilitan la ejecución simultánea de tareas, mejorando la eficiencia al realizar múltiples operaciones al mismo tiempo. Permiten que un programa maneje varias secuencias de instrucciones independientes, llamadas hilos, que pueden ejecutarse en paralelo. Los hilos pasan por diversos estados, como "nuevo", "ejecutable", "en ejecución", "bloqueado" y "terminado". La sincronización es esencial para evitar problemas cuando varios hilos acceden a recursos compartidos.

## Parte I Hilos Java

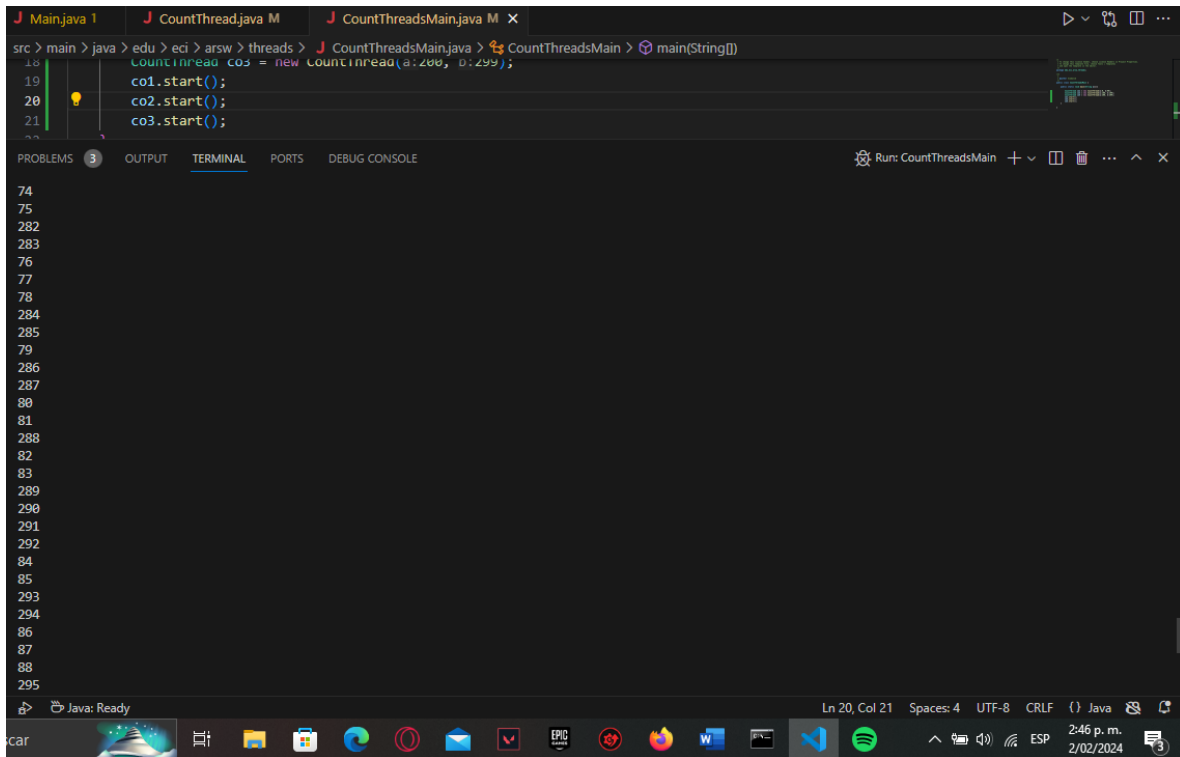
1. De acuerdo con lo revisado en las lecturas, complete las clases CountThread, para que las mismas definan el ciclo de vida de un hilo que imprima por pantalla los números entre A y B.



```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package edu.eci.arsw.threads;
7
8  import java.lang.Thread;
9
10  /**
11   *
12   * @author hcadavid
13   */
14  public class CountThread extends Thread {
15      public int a;
16      public int b;
17
18      public CountThread (int a, int b){
19          this.a = a;
20          this.b = b;
21      }
22
23      public void run (){
24          for (int i = a ; i <= b; i++){
25              System.out.println(i);
26          }
27      }
28  }
29
```



iii. Ejecute y revise la salida por pantalla.



```
src > main > java > edu > eci > arsw > threads > J CountThreadsMain.java > CountThreadsMain > main(String[])
18      CountThread co3 = new CountThread(a:200, b:299);
19      co1.start();
20      co2.start();
21      co3.start();
22  }
```

PROBLEMS 3 OUTPUT TERMINAL PORTS DEBUG CONSOLE

Run: CountThreadsMain

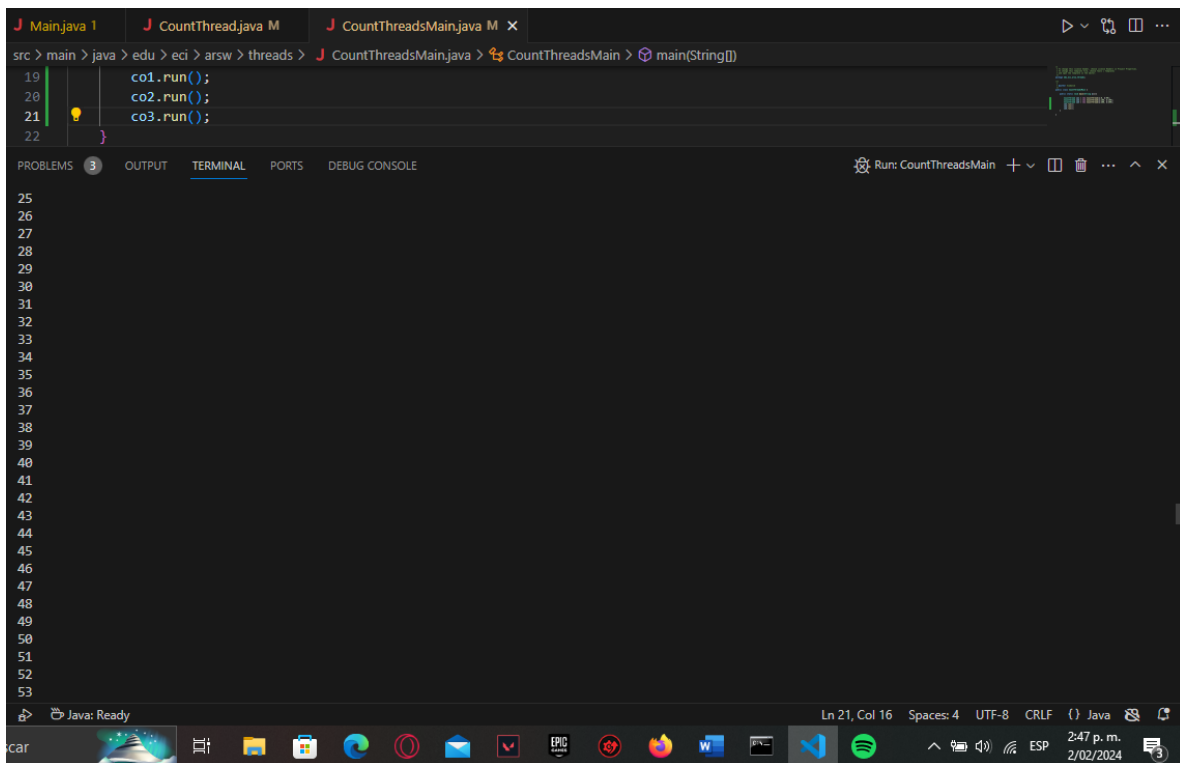
74  
75  
282  
283  
76  
77  
78  
284  
285  
79  
286  
287  
80  
81  
288  
82  
83  
289  
290  
291  
292  
84  
85  
293  
294  
86  
87  
88  
295

Java: Ready

Ln 20, Col 21 Spaces: 4 UTF-8 CRLF () Java

2:46 p. m. 2/02/2024

iv. Cambie el inicio con 'start()' por 'run()'. Cómo cambia la salida?, por qué?.



```
src > main > java > edu > eci > arsw > threads > J CountThreadsMain.java > CountThreadsMain > main(String[])
19      co1.run();
20      co2.run();
21      co3.run();
22  }
```

PROBLEMS 3 OUTPUT TERMINAL PORTS DEBUG CONSOLE

Run: CountThreadsMain

25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53

Java: Ready

Ln 21, Col 16 Spaces: 4 UTF-8 CRLF () Java

2:47 p. m. 2/02/2024

## Parte II Hilos Java

La fórmula [BBP](#) (Bailey–Borwein–Plouffe formula) es un algoritmo que permite calcular el enésimo dígito de PI en base 16, con la particularidad de no necesitar calcular los  $n-1$  dígitos anteriores. Esta característica permite convertir el problema de calcular un número masivo de dígitos de PI (en base 16) a uno [vergonzosamente paralelo](#). En este repositorio encontrará la implementación, junto con un conjunto de pruebas.

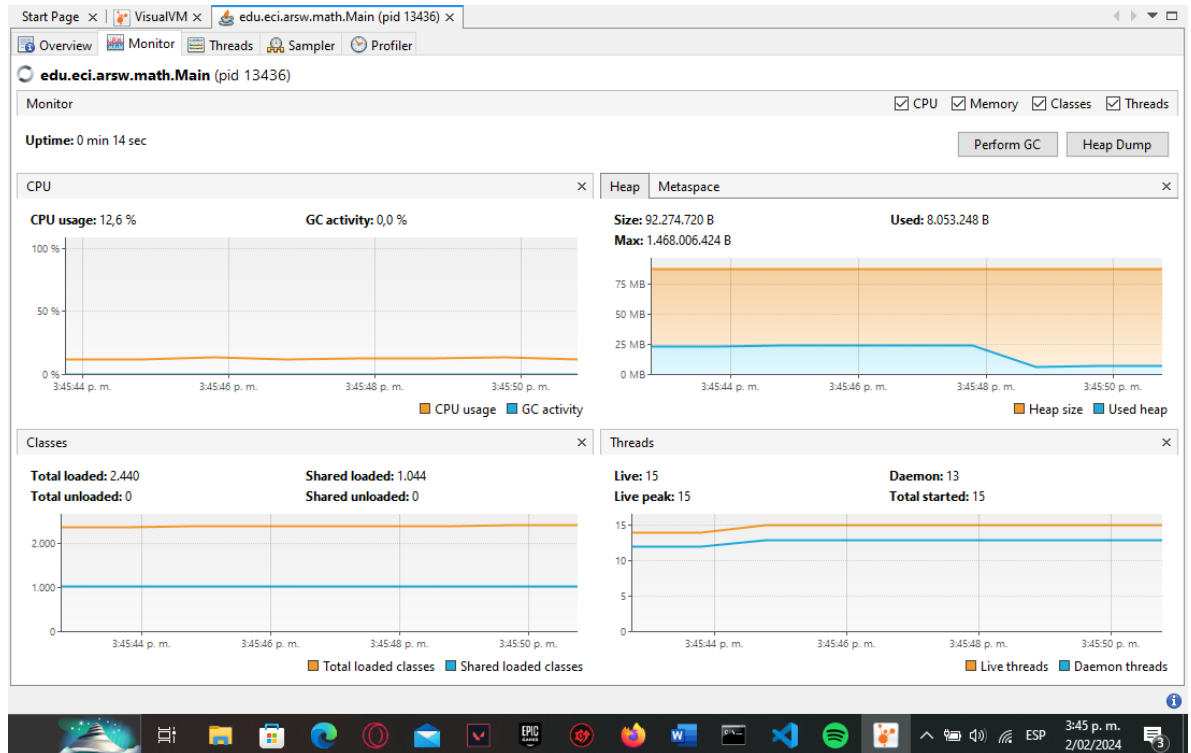
Para este ejercicio se quiere calcular, en el menor tiempo posible, y en una sola máquina (aprovechando las características multi-core de la mismas) al menos el primer millón de dígitos de PI (en base 16). Para esto

1. Cree una clase de tipo Thread que represente el ciclo de vida de un hilo que calcule una parte de los dígitos requeridos.
2. Haga que la función `PiDigits.getDigits()` reciba como parámetro adicional un valor N, correspondiente al número de hilos entre los que se va a paralelizar la solución. Haga que dicha función espere hasta que los N hilos terminen de resolver el problema para combinar las respuestas y entonces retornar el resultado. Para esto, revise el método [join](#) del API de concurrencia de Java.
3. Ajuste las pruebas de JUnit, considerando los casos de usar 1, 2 o 3 hilos (este último para considerar un número impar de hilos!)

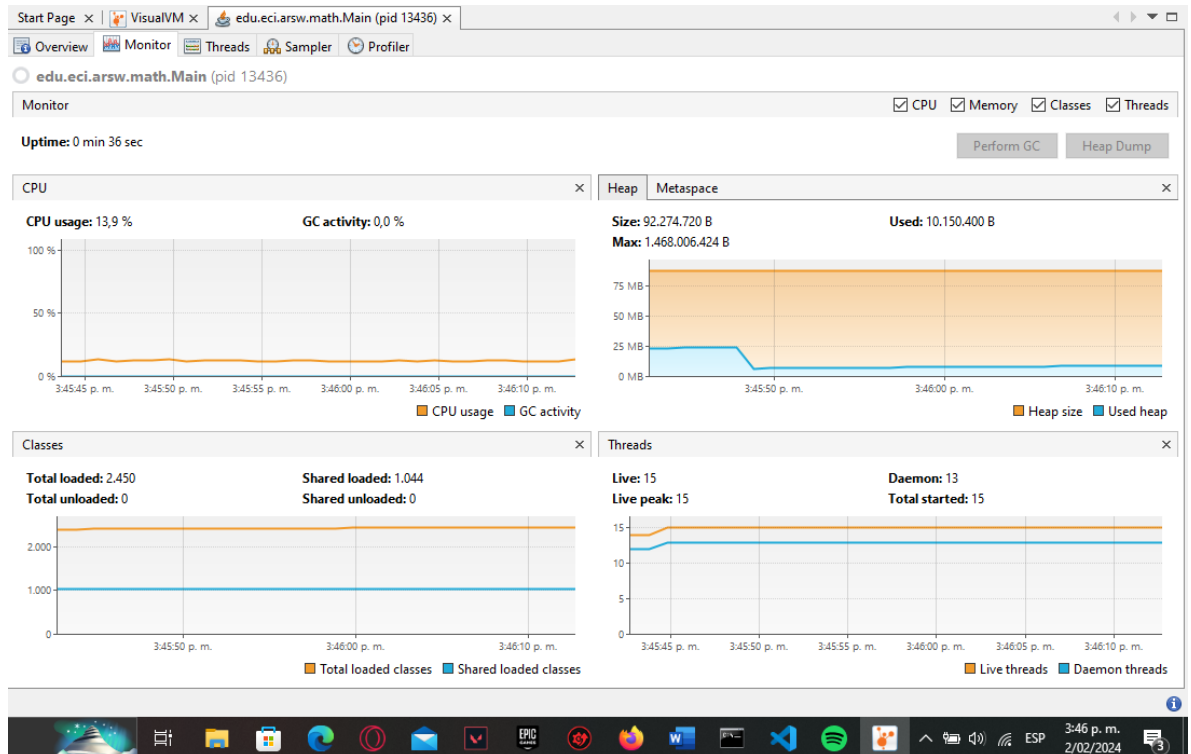
## Parte III Evaluación de Desempeño

A partir de lo anterior, implemente la siguiente secuencia de experimentos para calcular el millón de dígitos (hex) de PI, tomando los tiempos de ejecución de los mismos (asegúrese de hacerlos en la misma máquina):

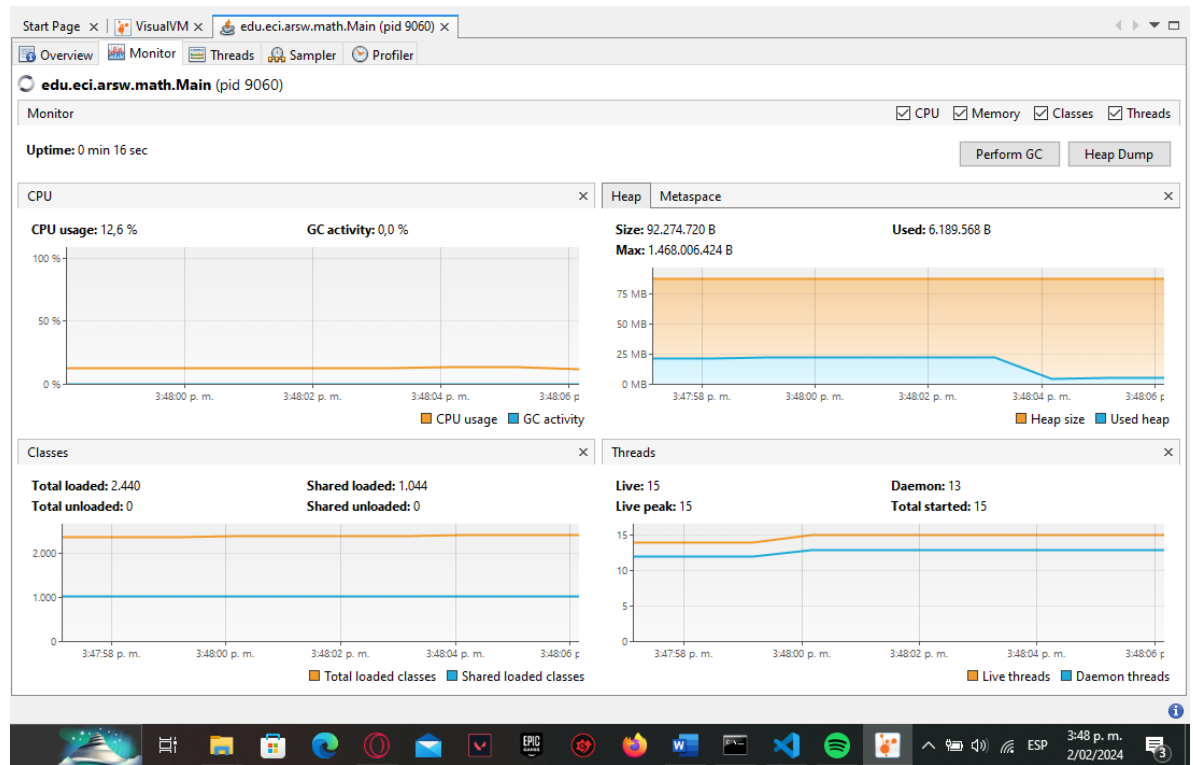
1. Un solo hilo.



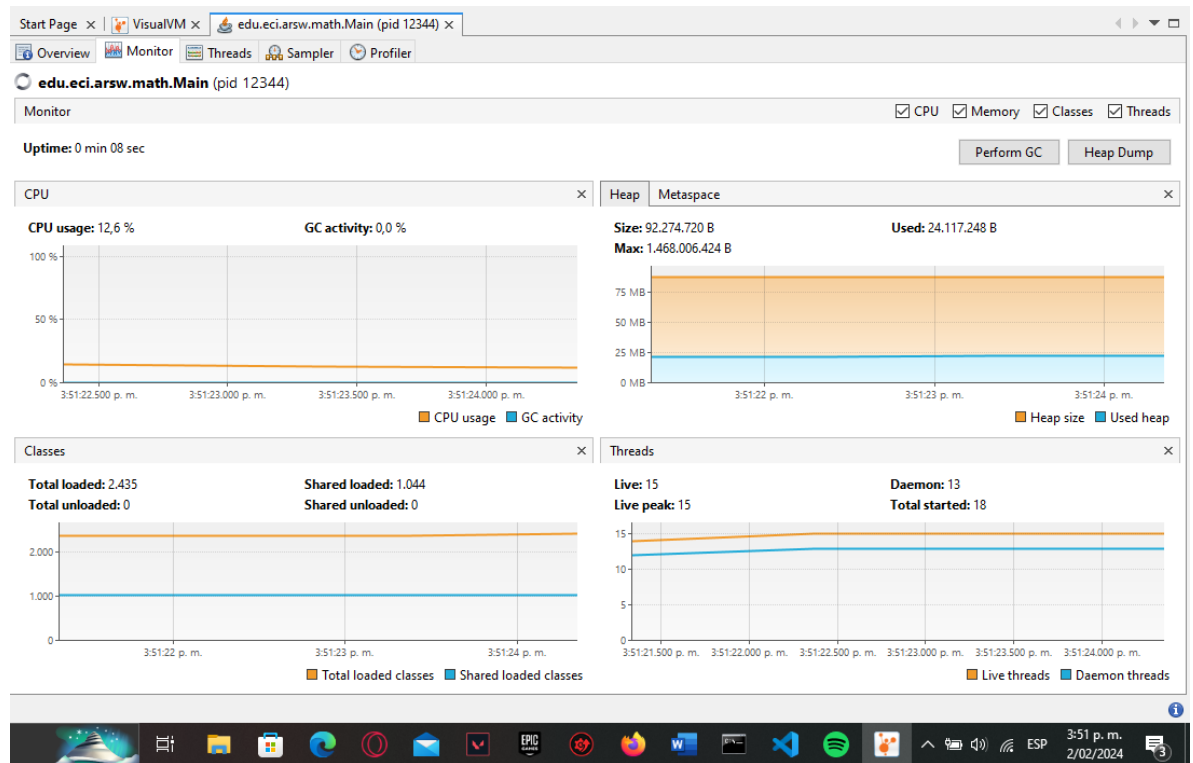
2. Tantos hilos como núcleos de procesamiento (haga que el programa determine esto haciendo uso del [API Runtime](#)). (4 hilos)



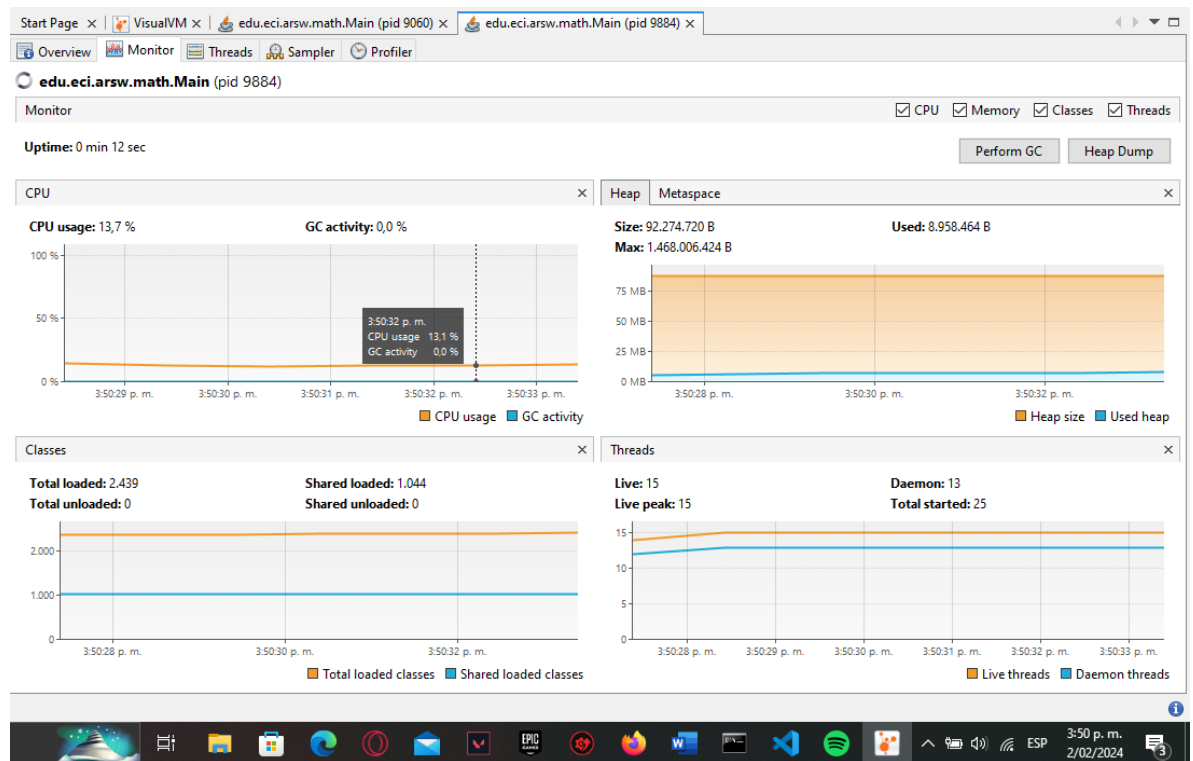
3. Tantos hilos como el doble de núcleos de procesamiento. 8 hilos



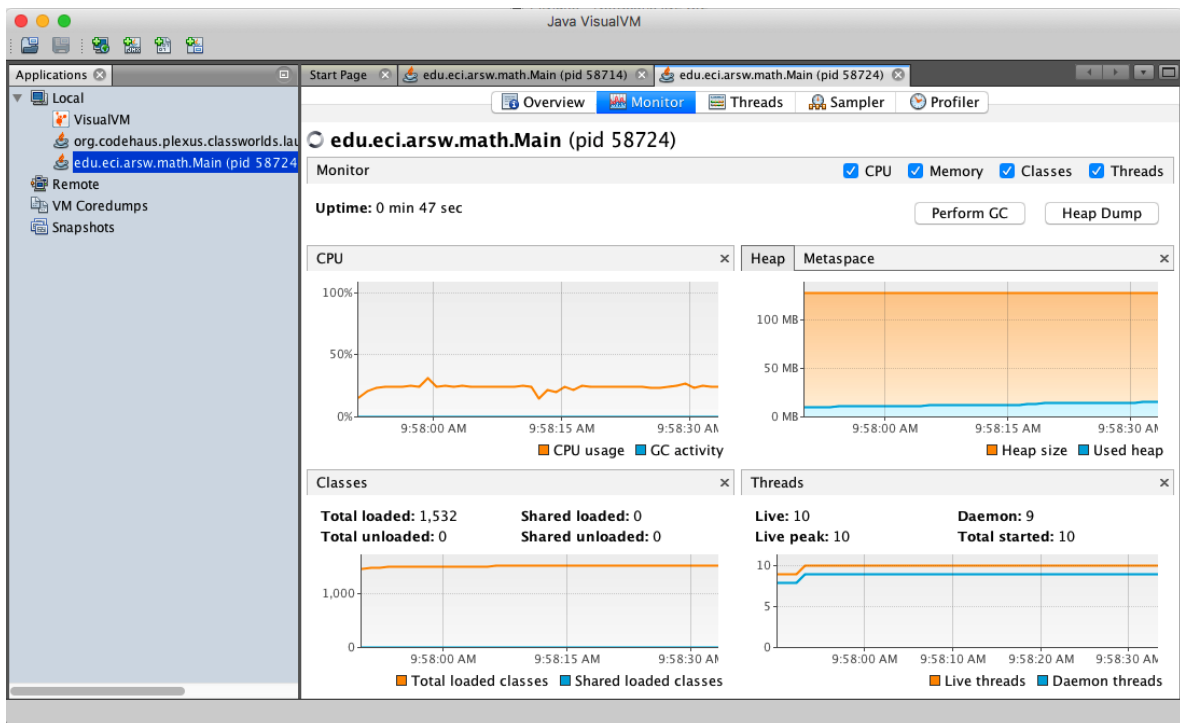
4. 200 hilos.



5. 500 hilos.



Al iniciar el programa ejecute el monitor jVisualVM, y a medida que corran las pruebas, revise y anote el consumo de CPU y de memoria en cada caso.



Con lo anterior, y con los tiempos de ejecución dados, haga una gráfica de tiempo de solución vs. número de hilos. Analice y plantee hipótesis con su compañero para las siguientes preguntas (puede tener en cuenta lo reportado por jVisualVM):

1. Según la [ley de Amdahls](#):

$$S(n) = \frac{1}{(1 - P) + \frac{P}{n}}$$

, donde  $S(n)$  es el mejoramiento teórico del desempeño,  $P$  la fracción paralelizable del algoritmo, y  $n$  el número de hilos, a mayor  $n$ , mayor debería ser dicha mejora. Por qué el mejor desempeño no se logra con los 500 hilos?, cómo se compara este desempeño cuando se usan 200?.

La ley de Amdahl postula que al mejorar el rendimiento de uno de los procesos de un sistema, se puede optimizar el rendimiento general. Al emplear hilos en el programa BBP, se busca optimizar el tiempo de ejecución de cada subproceso, es decir, el tiempo de los hilos. En consecuencia, un aumento en la cantidad de hilos debería traducirse en un rendimiento mejorado en el programa.

No obstante, al analizar los resultados, se observa que el rendimiento al utilizar 500 hilos es el más lento de todos y, además, es más lento en comparación con el uso de 200 hilos. Este fenómeno se explica por la ley de Amdahl, que indica que existe un punto en el que la mejora de rendimiento en el programa se vuelve casi constante, independientemente de la cantidad de hilos utilizados.

Es importante señalar que la ley de Amdahl es un concepto teórico y no considera factores físicos que impactan en el rendimiento del sistema, como el intercambio de bloqueos o las variables compartidas entre los hilos. Estas operaciones añaden complejidad al tiempo de ejecución del programa, y a medida que se crean más subprocesos, aumenta la ralentización del sistema.

2. Cómo se comporta la solución usando tantos hilos de procesamiento como núcleos comparado con el resultado de usar el doble de éste?.

Al emplear una cantidad de hilos equivalente al número de procesadores (4), experimentamos una mejora de rendimiento significativa. Sin embargo, al utilizar el doble de hilos en comparación con el número de procesadores, se observa una disminución en el rendimiento del programa. Esta desaceleración se puede explicar de manera similar a cuando se utilizan 200 o 500 hilos.

3. De acuerdo con lo anterior, si para este problema en lugar de 500 hilos en una sola CPU se pudiera usar 1 hilo en cada una de 500 máquinas hipotéticas, la ley de Amdahls se aplicaría mejor?. Si en lugar de esto se usaran  $c$  hilos en  $500/c$  máquinas distribuidas (siendo  $c$  es el número de núcleos de dichas máquinas), se mejoraría?. Explique su respuesta.

No se podría lograr una mejora en el rendimiento del sistema, ya que al unir los resultados de cada una de las diferentes máquinas, se estarían consumiendo recursos físicos que



ralentizarían nuestro programa. Además, al analizar los casos previos, se observa que en la gráfica de Número de Threads vs Tiempo después de utilizar 4 o 5 hilos, la mejora en el rendimiento de nuestro programa se mantiene prácticamente constante.

### **Criterios de evaluación.**

#### **1. Funcionalidad:**

- El problema fue paralelizado (el tiempo de ejecución se reduce y el uso de los núcleos aumenta), y permite parametrizar el número de hilos usados simultáneamente.

#### **2. Diseño:**

- La signature del método original sólo fue modificada con el parámetro original, y en el mismo debe quedar encapsulado la paralelización e inicio de la solución, y la sincronización de la finalización de la misma.
- Las nuevas pruebas con sólo UN hilo deben ser exactamente iguales a las originales, variando sólo el parámetro adicional. Se incluyeron pruebas con hilos adicionales, y las mismas pasan.
- Se plantea un método eficiente para combinar los resultados en el orden correcto (iterar sobre cada resultado NO sera eficiente).

#### **3. Análisis.**

- Se deja evidencia de la realización de los experimentos.
- Los análisis realizados son consistentes.

### **Conclusión**

En conclusión, los hilos en Java ofrecen una forma eficiente de lograr la ejecución concurrente en programas, permitiendo que múltiples tareas se realicen simultáneamente. Al dividir la ejecución en unidades más pequeñas y manejables, los hilos mejoran la capacidad de respuesta y la eficiencia de las aplicaciones. Los hilos pasan por distintos estados a lo largo de su ciclo de vida y requieren una correcta sincronización para evitar problemas de concurrencia y garantizar la consistencia de los datos compartidos. La comprensión de los conceptos básicos de los hilos, junto con el uso adecuado de herramientas como el marco de ejecución, es esencial para aprovechar los beneficios de la programación concurrente. En resumen, los hilos en Java son una herramienta poderosa para lograr paralelismo y mejorar el rendimiento de las aplicaciones al permitir que diferentes partes del programa se ejecuten de manera simultánea y coordinada.