**LABORATORIO No.1 ARSW**

María Belén Quintero Aldana

Nikolas Martínez Rivera

**Parte I - Introducción a Hilos en 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. Complete el método **main** de la clase CountMainThreads para que:
   1. Cree 3 hilos de tipo CountThread, asignándole al primero el intervalo [0..99], al segundo [99..199], y al tercero [200..299].

Texto

El contenido generado por IA puede ser incorrecto.

* 1. Ejecute y revise la salida por pantalla.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

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

Texto

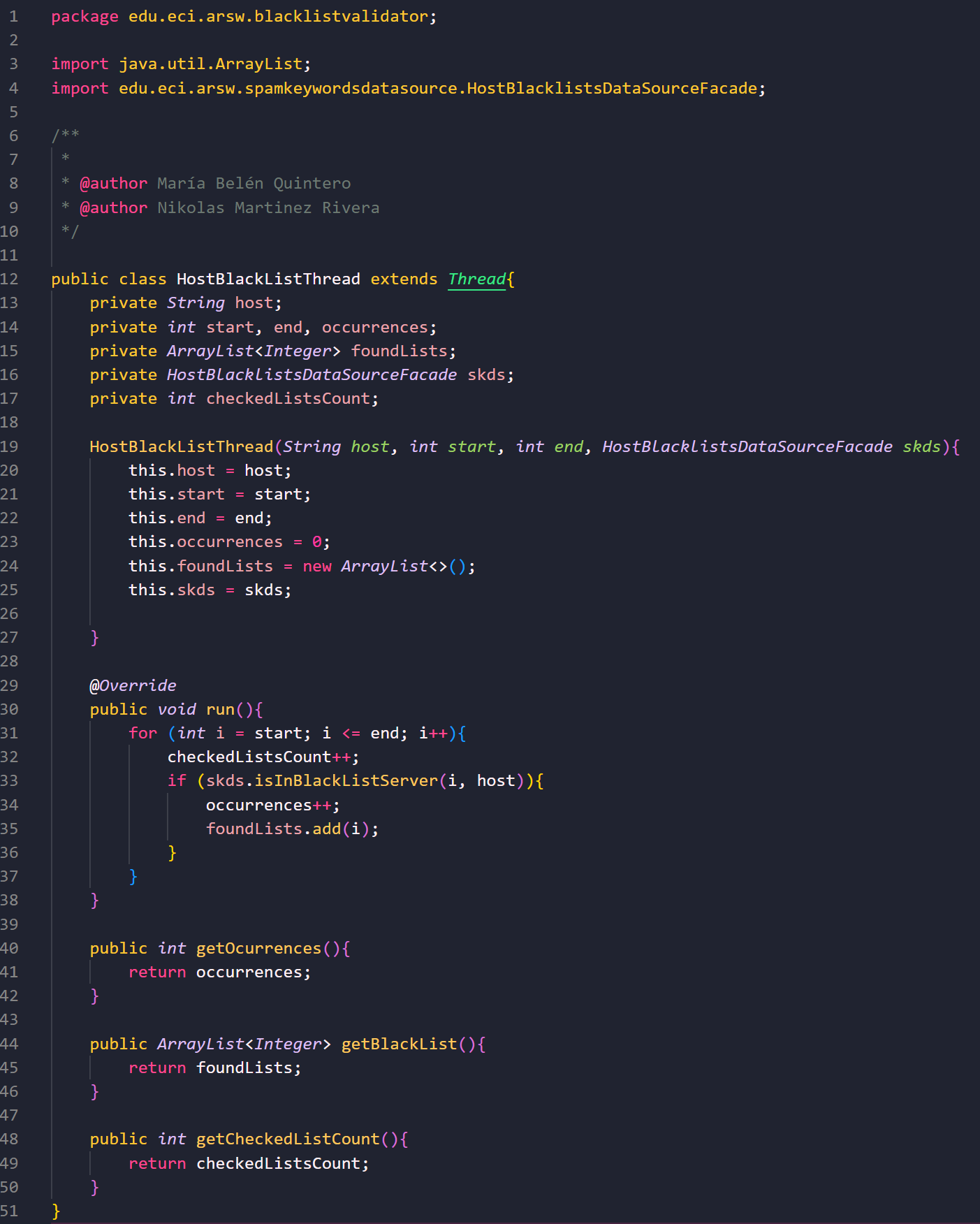
El contenido generado por IA puede ser incorrecto.

La salida cambia pues cuando se utiliza run() se ejecuta el método como cualquiera normal, no crea un hilo o algo parecido, a diferencia de cuando se ejecuta con start(), en aquel caso si se crea el hilo y se ejecuta run, es por eso que la impresión es diferente, por que los hilos están compartiendo recursos del procesador.

**Parte II - Ejercicio Black List Search**

1. Cree una clase de tipo Thread que represente el ciclo de vida de un hilo que haga la búsqueda de un segmento del conjunto de servidores disponibles. Agregue a dicha clase un método que permita 'preguntarle' a las instancias del mismo (los hilos) cuantas ocurrencias de servidores maliciosos ha encontrado o encontró.

**R/** La clase tipo Thread que decidimos implementar recibe el nombre de ‘HostBlackListThread’. He aquí una imagen del contenido de la clase:



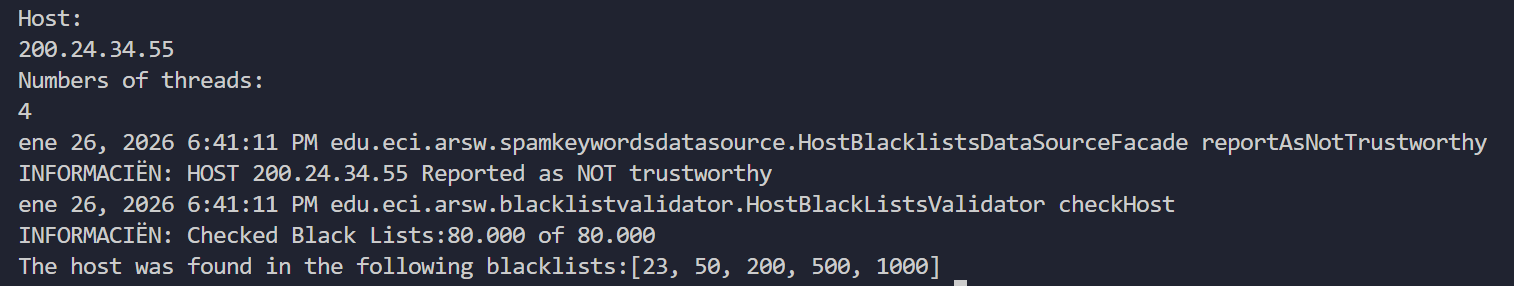
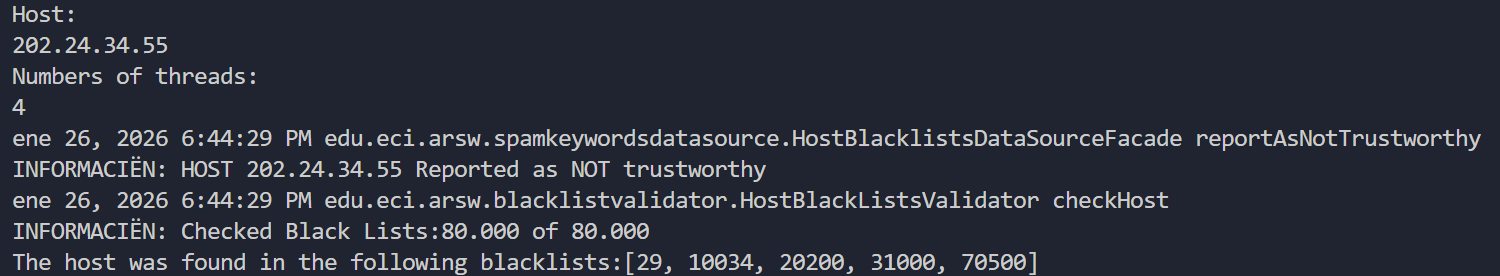
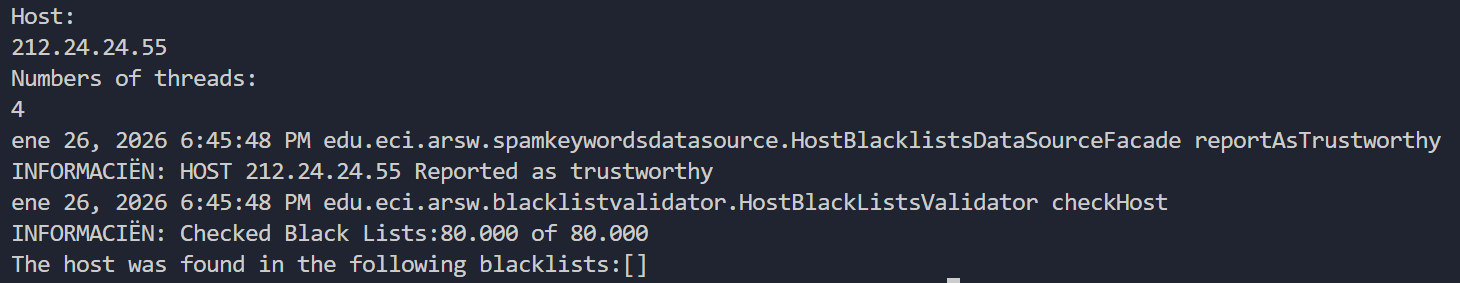
El método que nos permite 'preguntarle' a las instancias cuantas ocurrencias de servidores maliciosos ha encontrado o encontró es el método ‘run’, al cual, se le realizó un override.

1. Agregue al método 'checkHost' un parámetro entero N, correspondiente al número de hilos entre los que se va a realizar la búsqueda (recuerde tener en cuenta si N es par o impar!). Modifique el código de este método para que divida el espacio de búsqueda entre las N partes indicadas, y paralelice la búsqueda a través de N hilos. Haga que dicha función espere hasta que los N hilos terminen de resolver su respectivo subproblema, agregue las ocurrencias encontradas por cada hilo a la lista que retorna el método, y entonces calcule (sumando el total de ocurrencias encontradas por cada hilo) si el número de ocurrencias es mayor o igual a BLACK\_LIST\_ALARM\_COUNT. Si se da este caso, al final se DEBE reportar el host como confiable o no confiable, y mostrar el listado con los números de las listas negras respectivas. Para lograr este comportamiento de 'espera' revise el método [join](https://docs.oracle.com/javase/tutorial/essential/concurrency/join.html) del API de concurrencia de Java. Tenga también en cuenta:

* Dentro del método checkHost Se debe mantener el LOG que informa, antes de retornar el resultado, el número de listas negras revisadas VS. el número de listas negras total (línea 60). Se debe garantizar que dicha información sea verídica bajo el nuevo esquema de procesamiento en paralelo planteado.
* Se sabe que el HOST 202.24.34.55 está reportado en listas negras de una forma más dispersa, y que el host 212.24.24.55 NO está en ninguna lista negra.

**R/** La versión final del método ‘checkHost’ que se encuentra dentro de la clase ‘HostBlackListValidator’, quedó de la siguiente manera luego de modificarlo con lo pedido por el ejercicio.

  
A continuación, se presentarán los resultados obtenidos al ejecutar el programa con distintos casos de prueba:

* Prueba del host 200.24.34.55  
  
* Prueba del host 202.24.34.55  
  
* Prueba del host 212.24.24.55  
  

**Parte II.I Sincronización**

La estrategia de paralelismo antes implementada es ineficiente en ciertos casos, pues la búsqueda se sigue realizando aún cuando los N hilos (en su conjunto) ya hayan encontrado el número mínimo de ocurrencias requeridas para reportar al servidor como malicioso. Cómo se podría modificar la implementación para minimizar el número de consultas en estos casos?, qué elemento nuevo traería esto al problema?

**R/** Para poder realizar la implementación que minimiza el número de consultas si ya se hallaron en 5 listas, se implementó la clase ‘ShareCounter’ que permite que no se “pisen” entre hilos y que el programa pare al encontrar el host el máximo de veces ya mencionadas.  
La implementación de la clase ‘ShareCounter’ quedó de la siguiente manera:

Texto

El contenido generado por IA puede ser incorrecto.

También fue necesario hacer unos pequeños cambios dentro de las clases ‘HostBlackListThread’ y ‘HostBlackListValidator’, los cuales, fueron los siguientes:

Texto

El contenido generado por IA puede ser incorrecto.  
(Cambios en las líneas: 18, 20, 27, 33-36, 41)

Texto

El contenido generado por IA puede ser incorrecto.  
(Cambios en las líneas 42 y 54)

Un ejemplo del funcionamiento es el siguiente, en el que se usa el host 200.24.34.55 del que se sabe que existen varios reportes. Al empezar la búsqueda, ya no se revisa en las 80000 listas, sino que, después de hacer la revisión en 4014 listas. Por lo tanto, se concluye que la implementación fue exitosa.

Texto

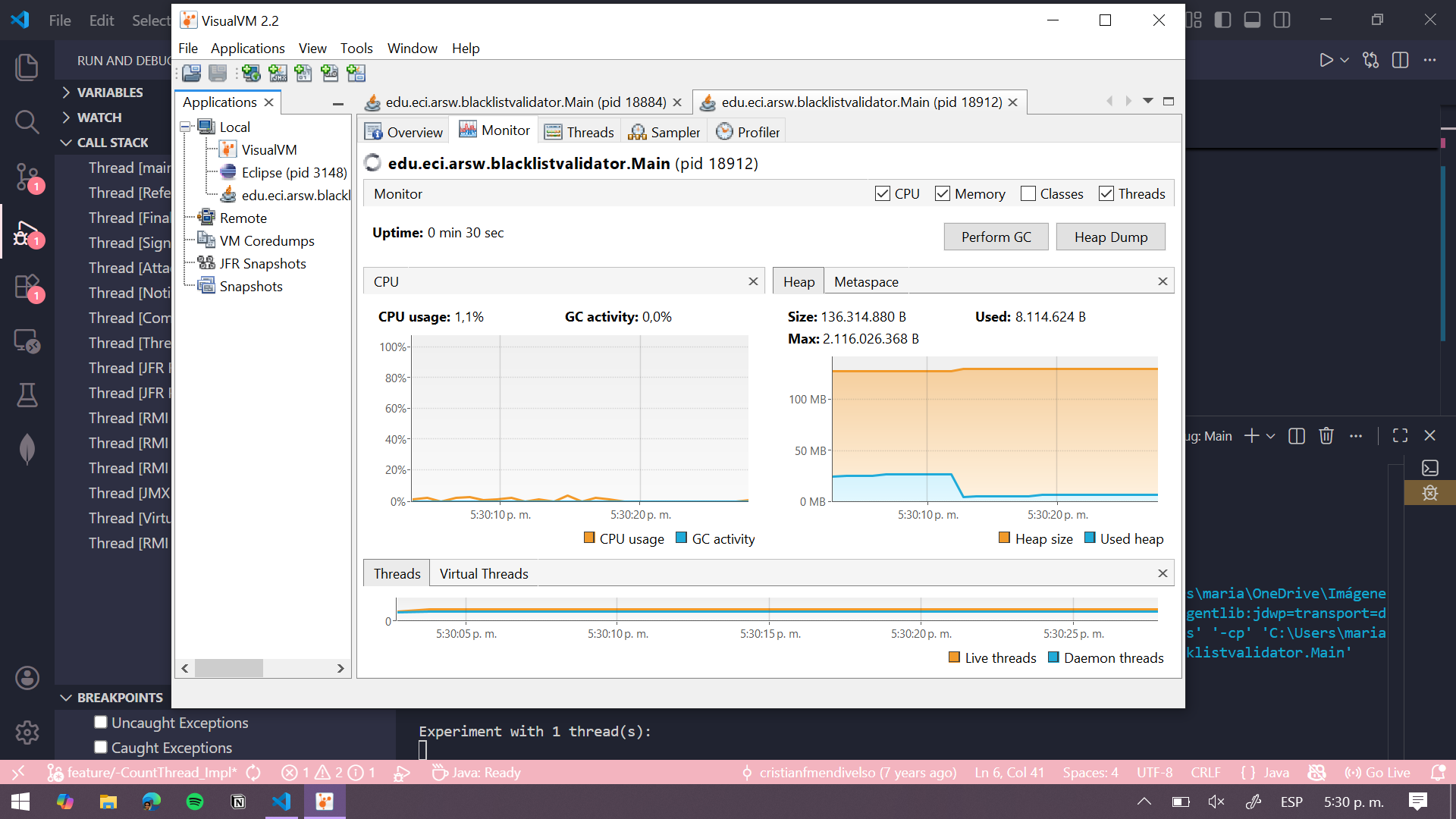
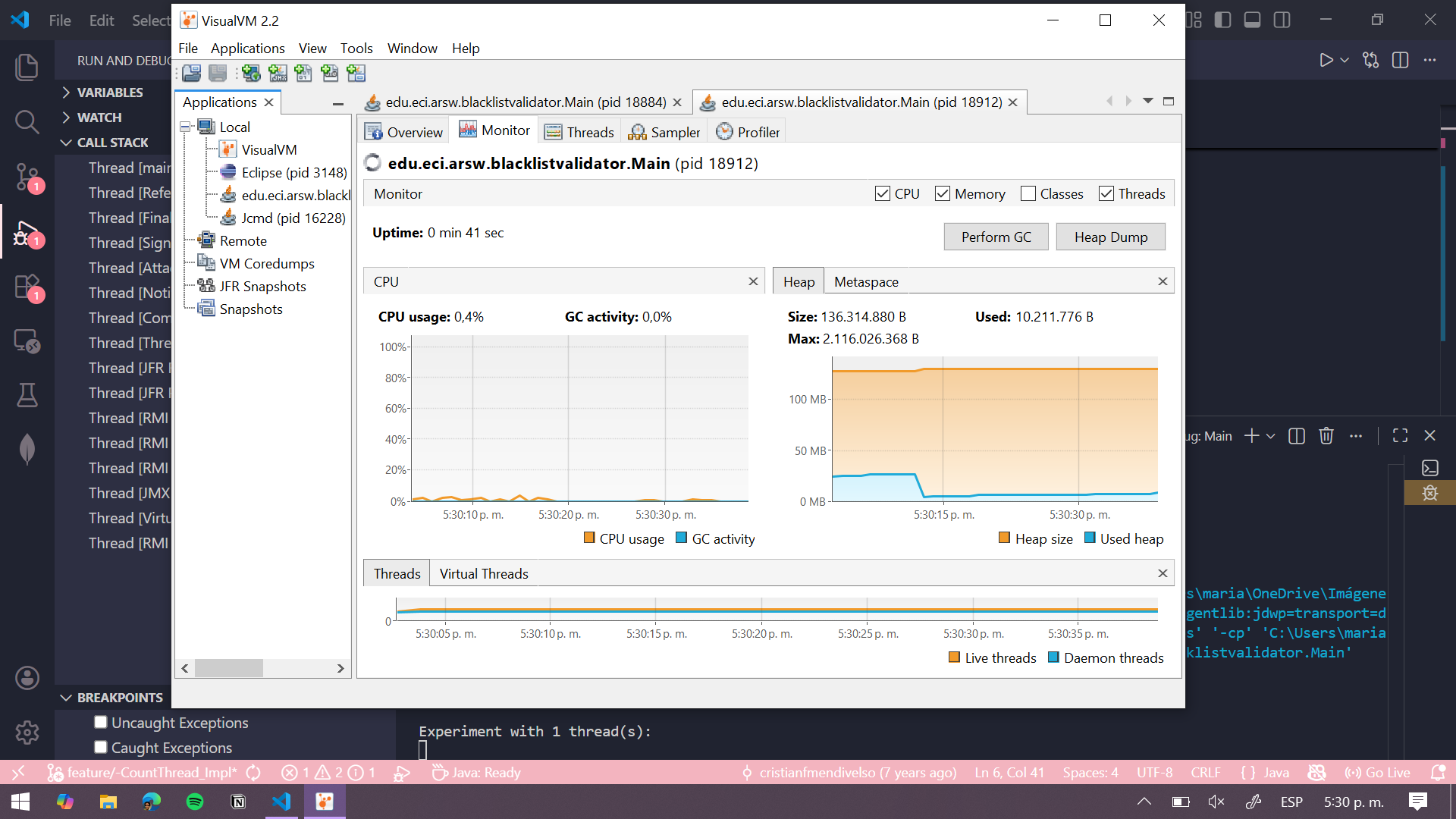
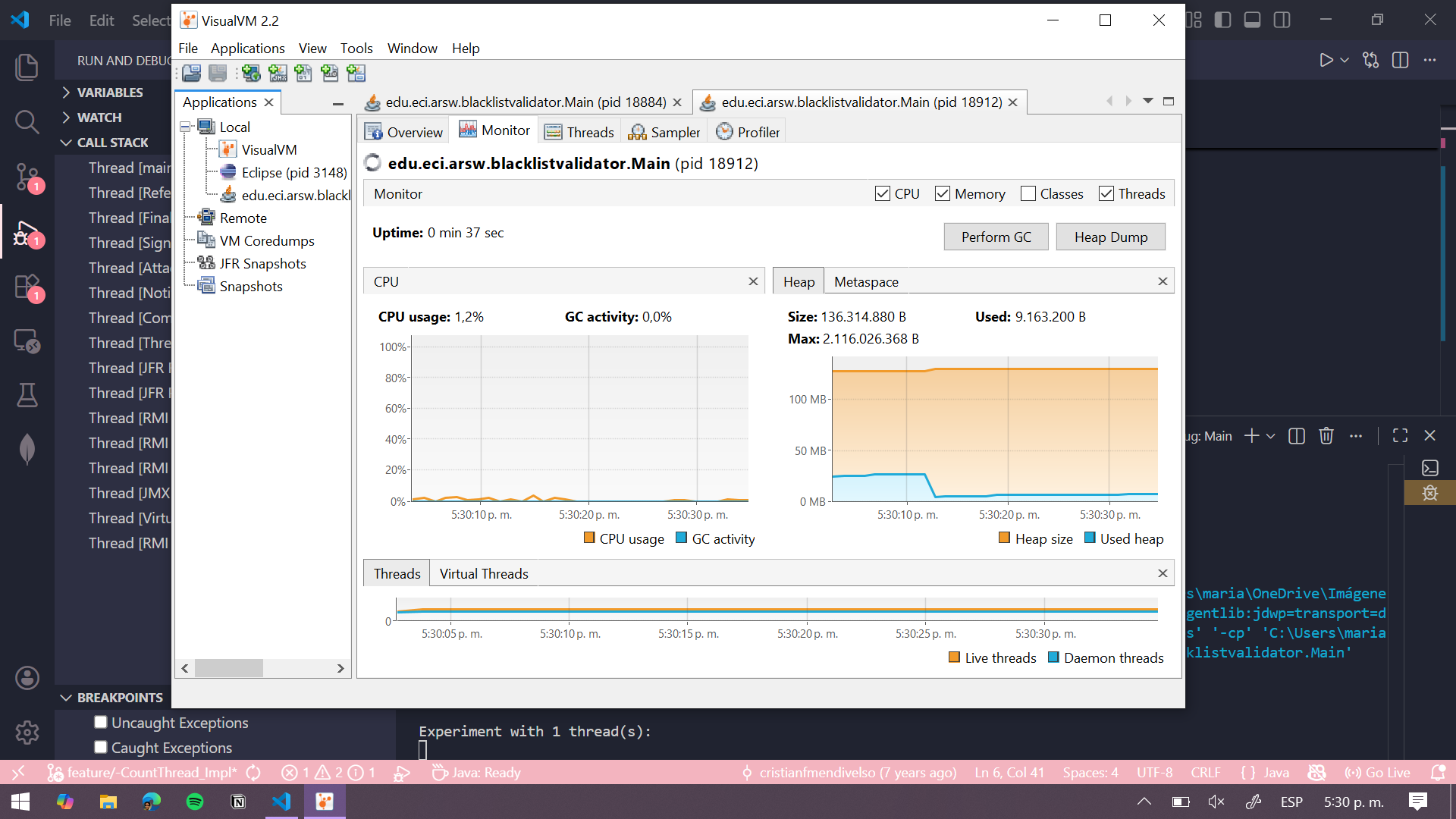
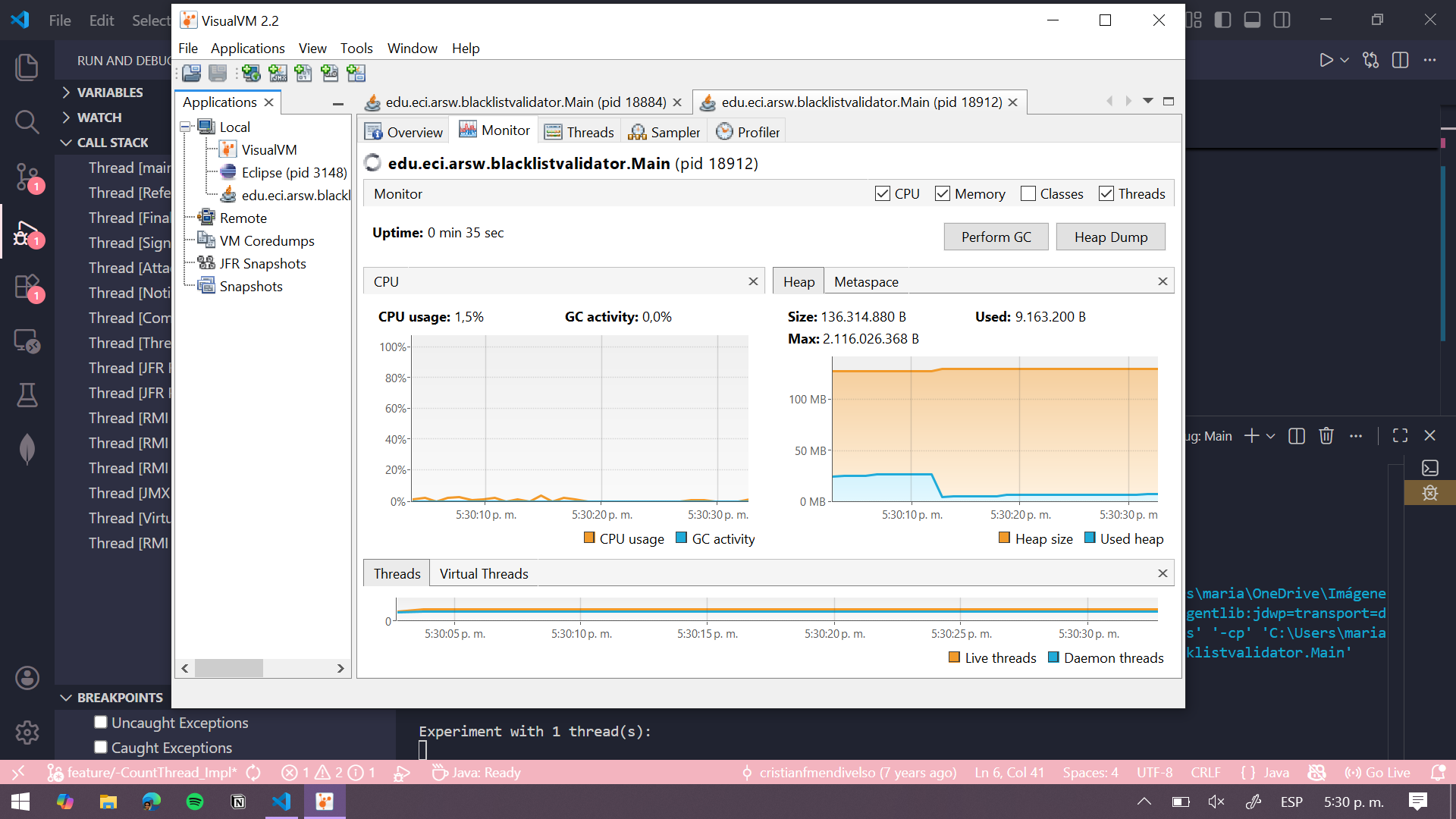
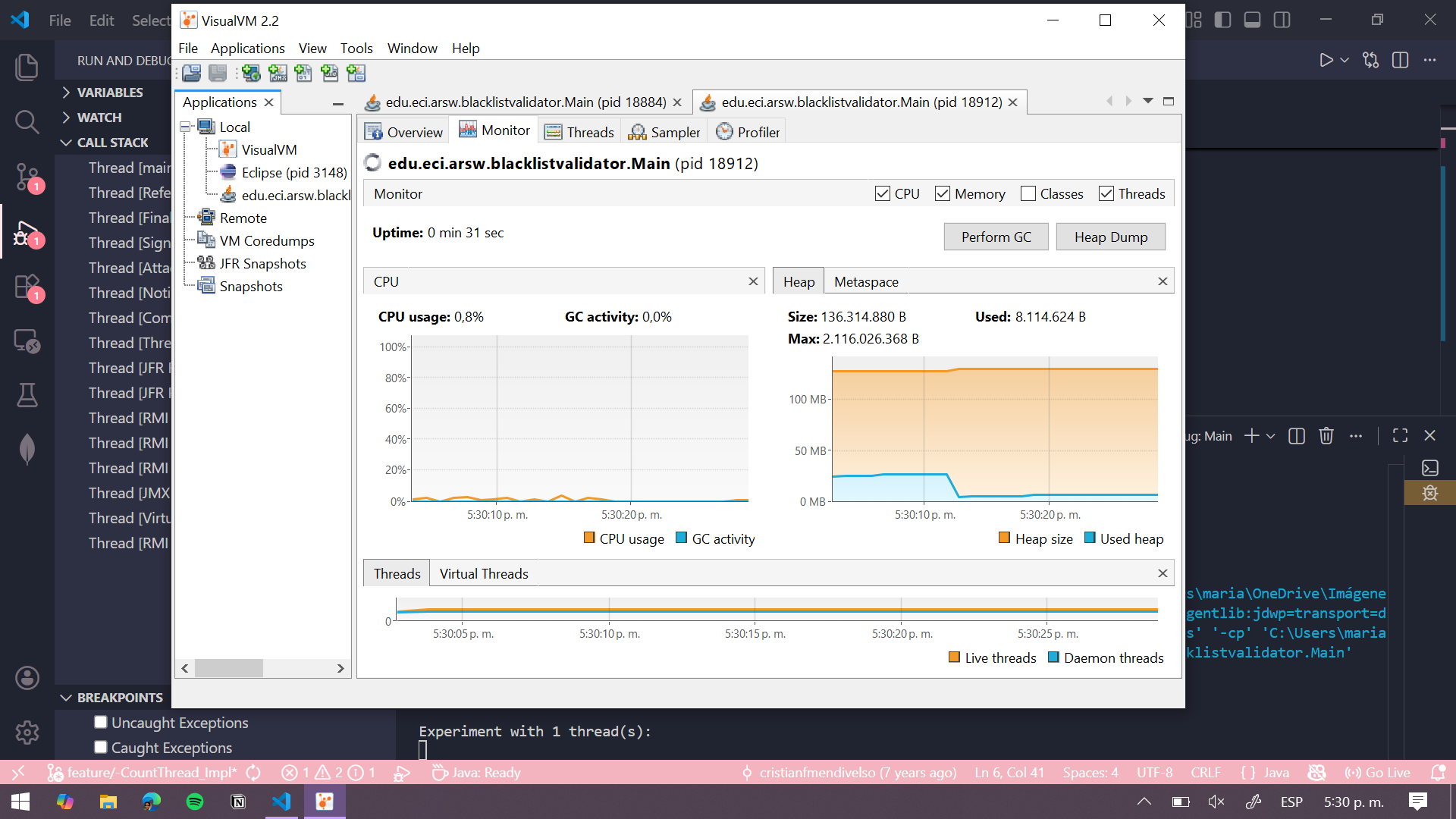
El contenido generado por IA puede ser incorrecto.

**Parte III - Evaluación de Desempeño**

A partir de lo anterior, implemente la siguiente secuencia de experimentos para realizar la validación de direcciones IP dispersas (por ejemplo 202.24.34.55), tomando los tiempos de ejecución de estos (asegúrese de hacerlos en la misma máquina):

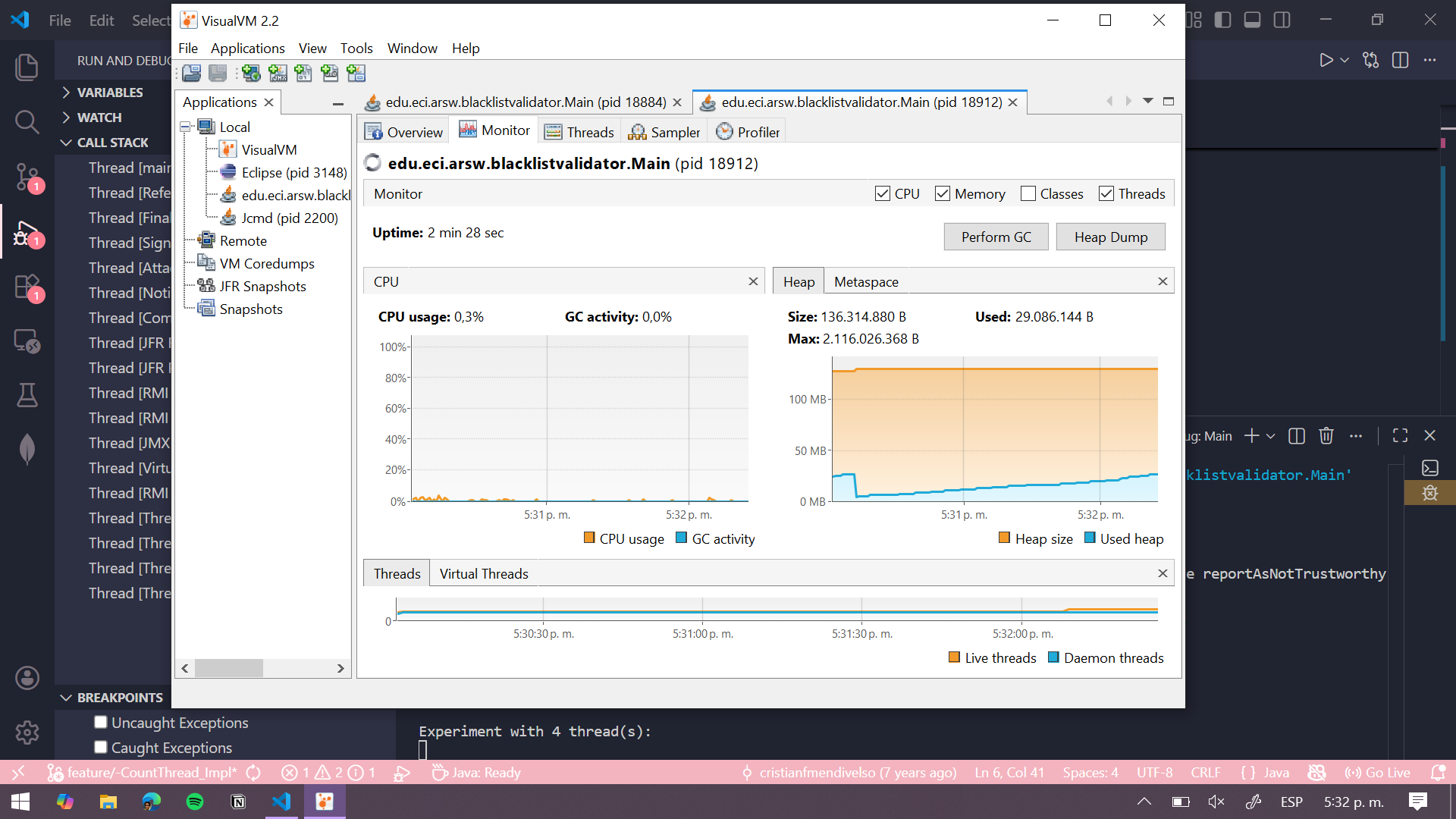
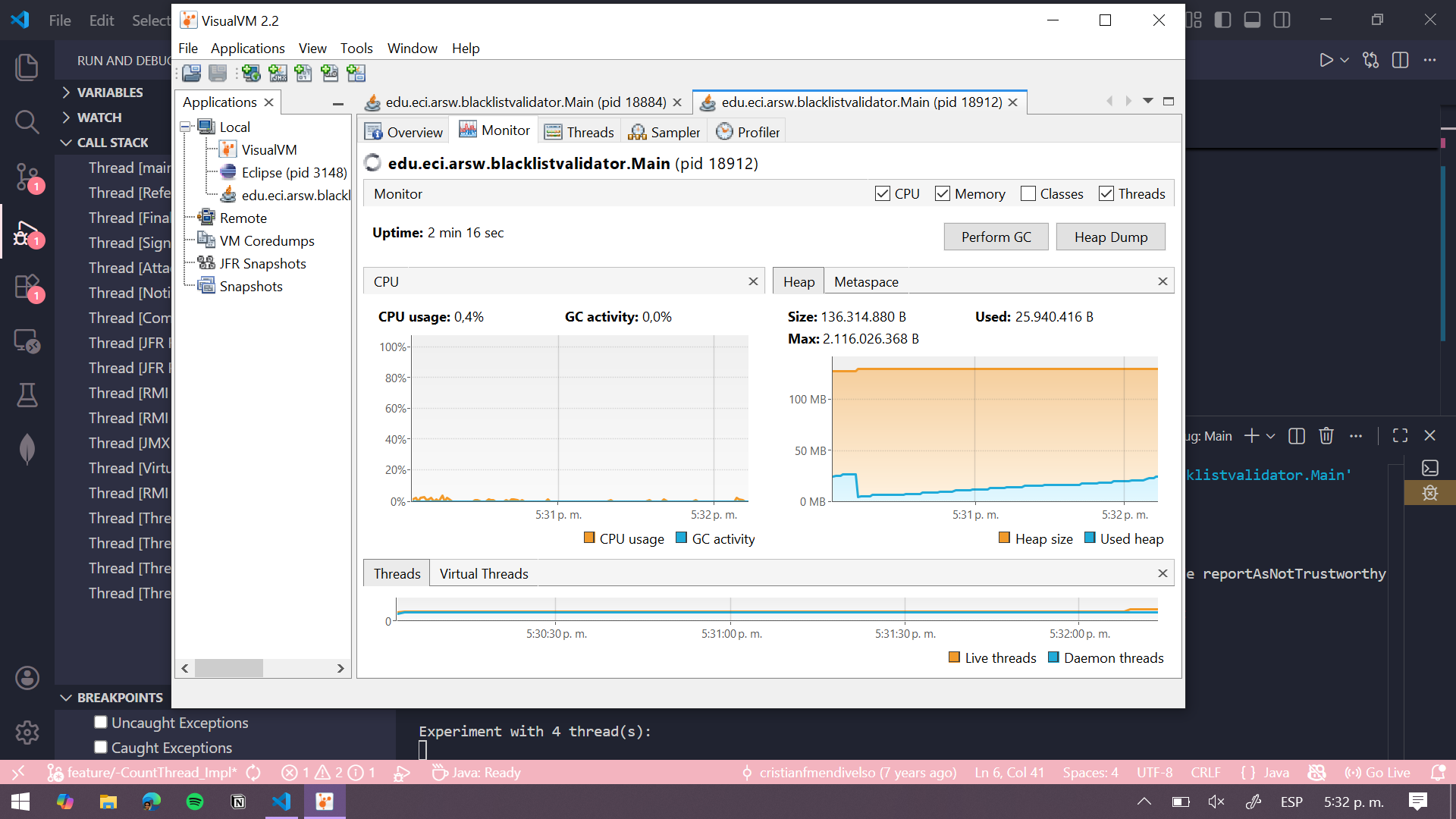
**R/** Para realizar el experimento, se modificó a la clase de Main con tal de facilitar el proceso. Además, es importante mencionar que en el computador donde se llevó a cabo el experimento cuenta con 4 núcleos. A continuación, se presentan los resultados del experimento.

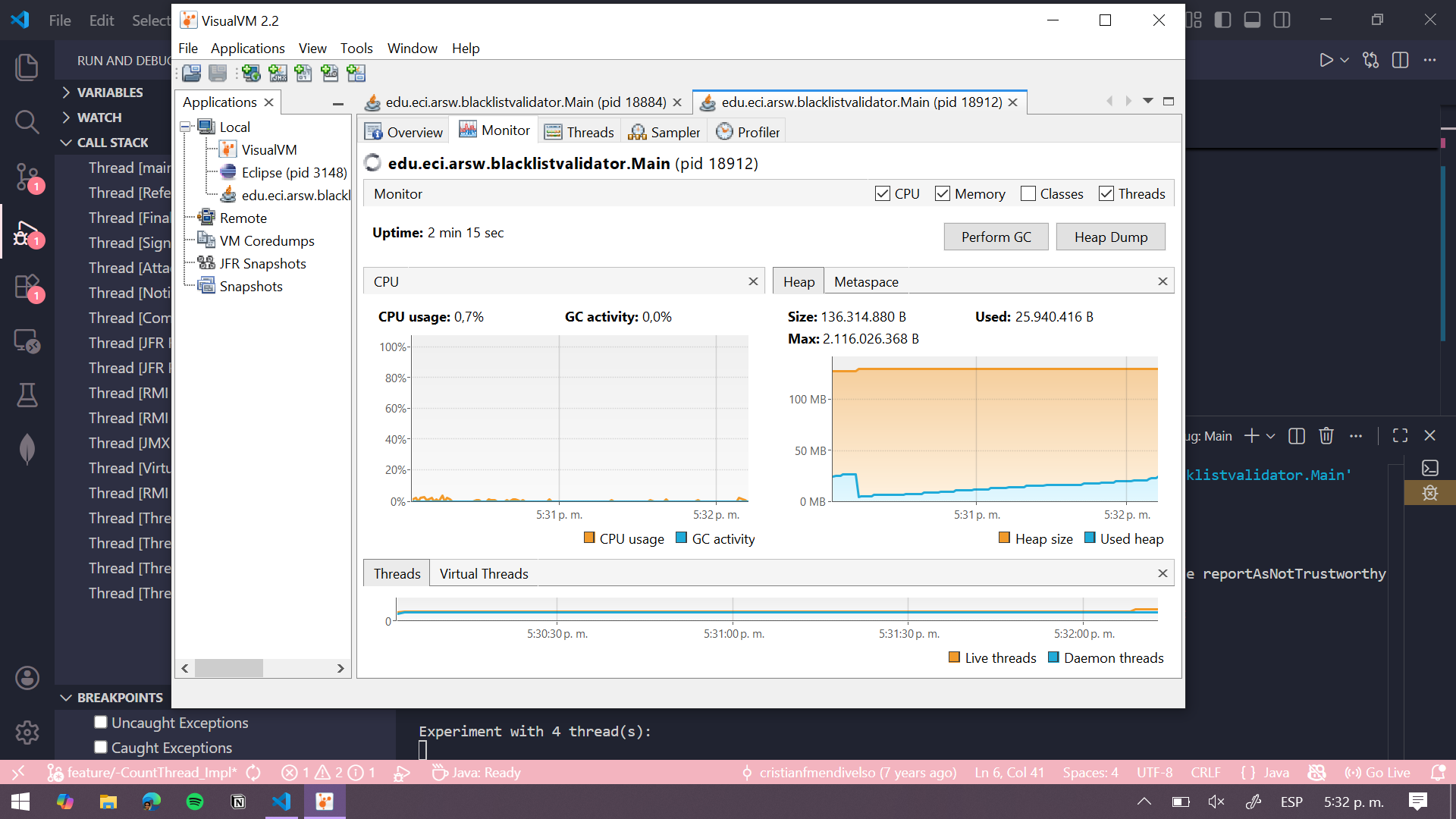
1. Un solo hilo.  
   Este fue el que más tiempo tomó, con un total de 130369ms, lo que es igual a aproximadamente 2 minutos y 10 segundos. En cuanto al porcentaje de uso de la CPU, este permaneció en varias ocasiones en 0% y no superó el 2,4%.

Imágenes del experimento haciendo uso de 1 hilo:

1. Tantos hilos como núcleos de procesamiento (haga que el programa determine esto haciendo uso del [API Runtime](https://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html)).

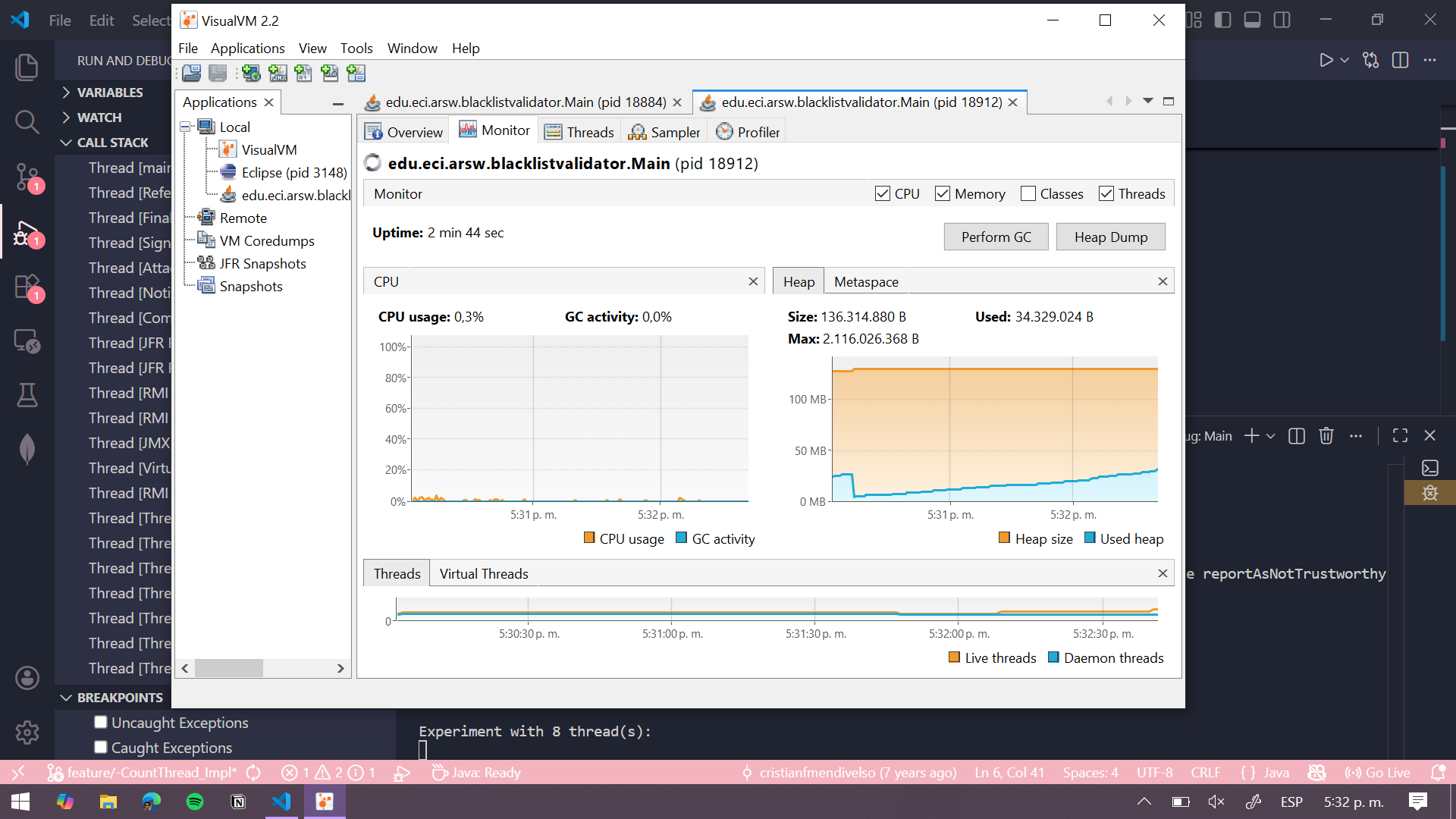
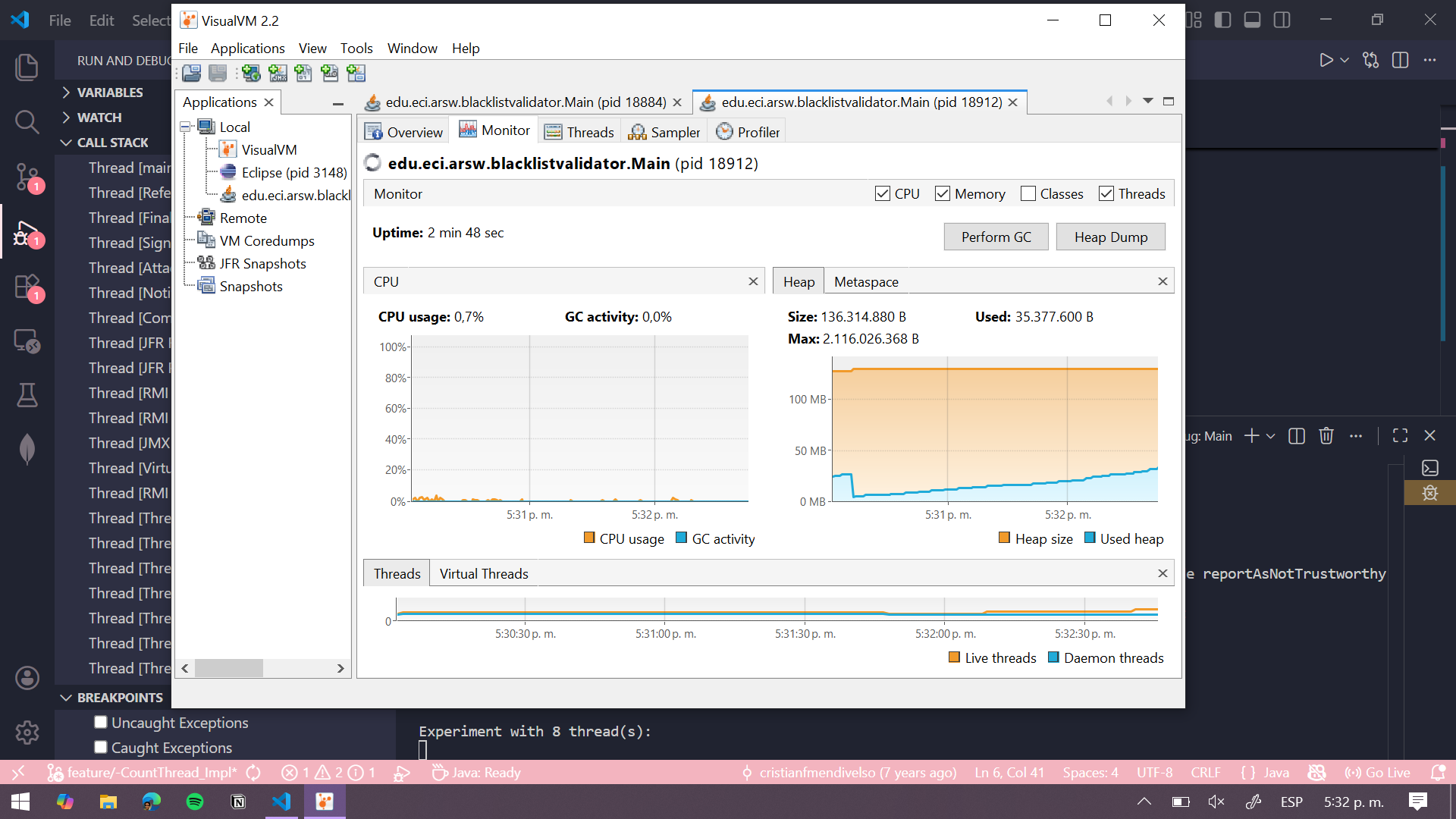
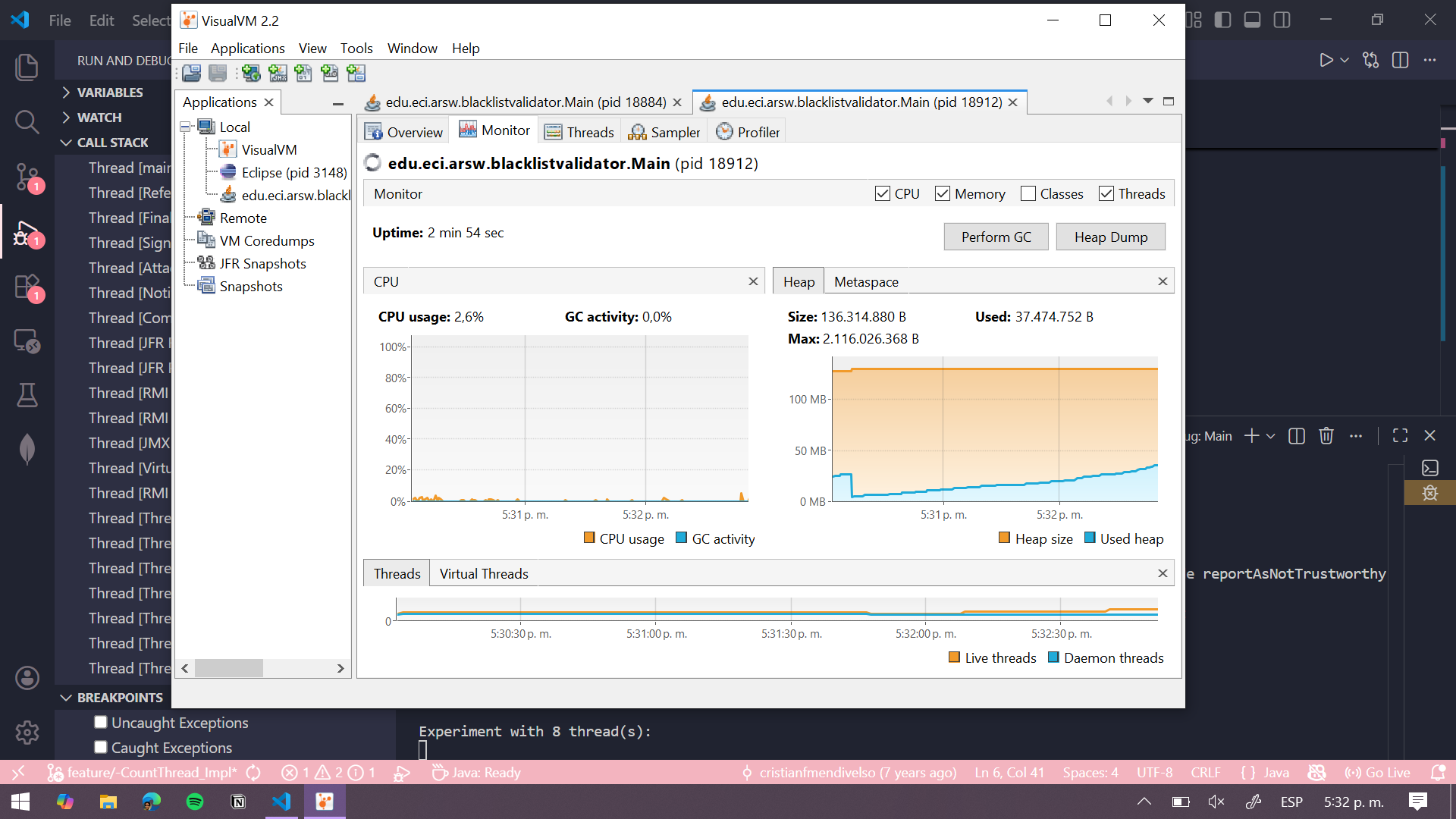
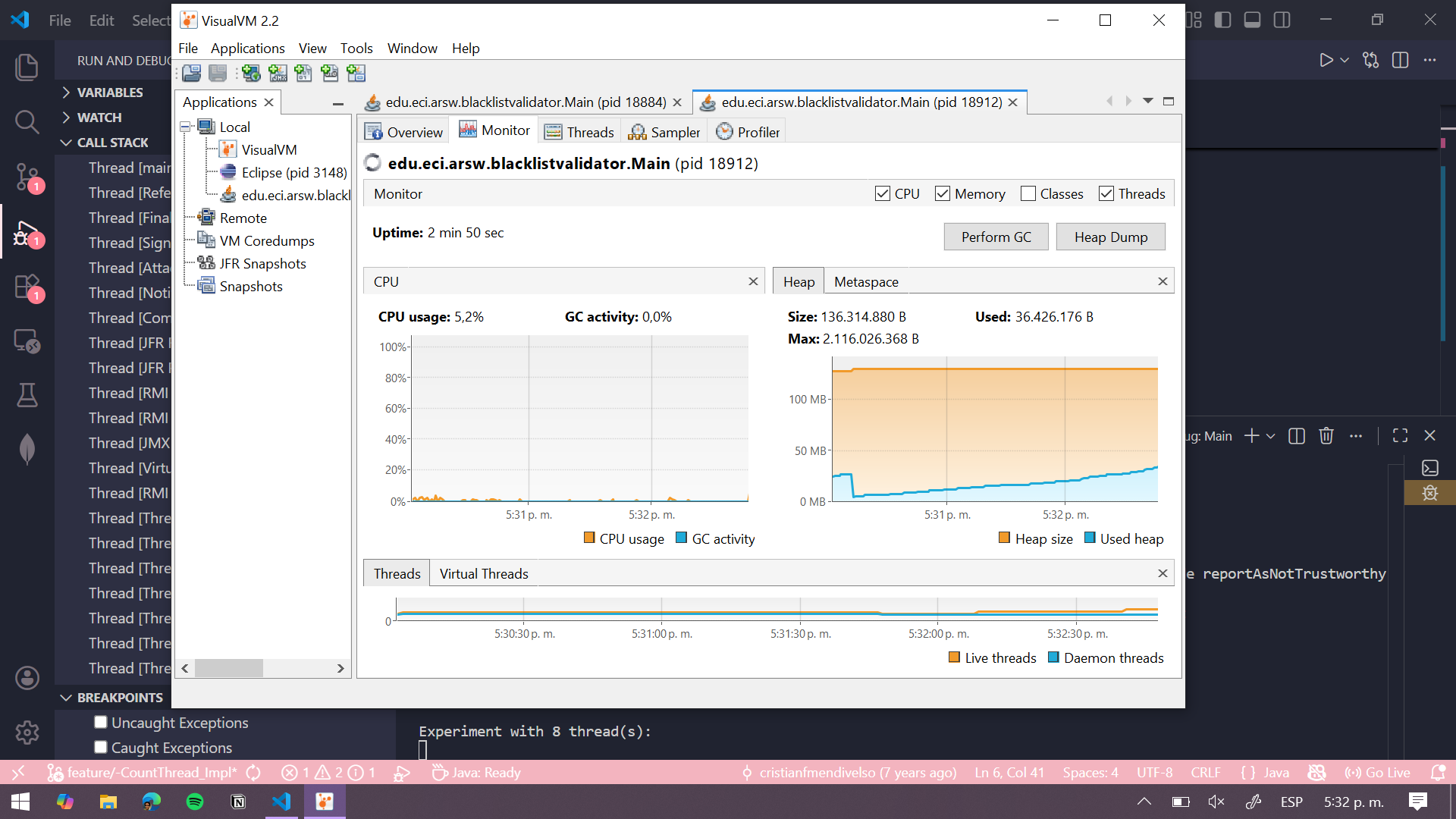
Como se comentó anteriormente, el procesador del computador usado para el experimento cuenta con 4 núcleos. El tiempo que tardaron los 4 hilos en completar la tarea fue de 32155ms, que equivalen a aproximadamente 32,16 segundos. En cuanto al porcentaje de uso de la CPU, este permaneció en varias ocasiones en 0% y no superó el 0,7%.

Imágenes del experimento haciendo uso de 4 hilos:



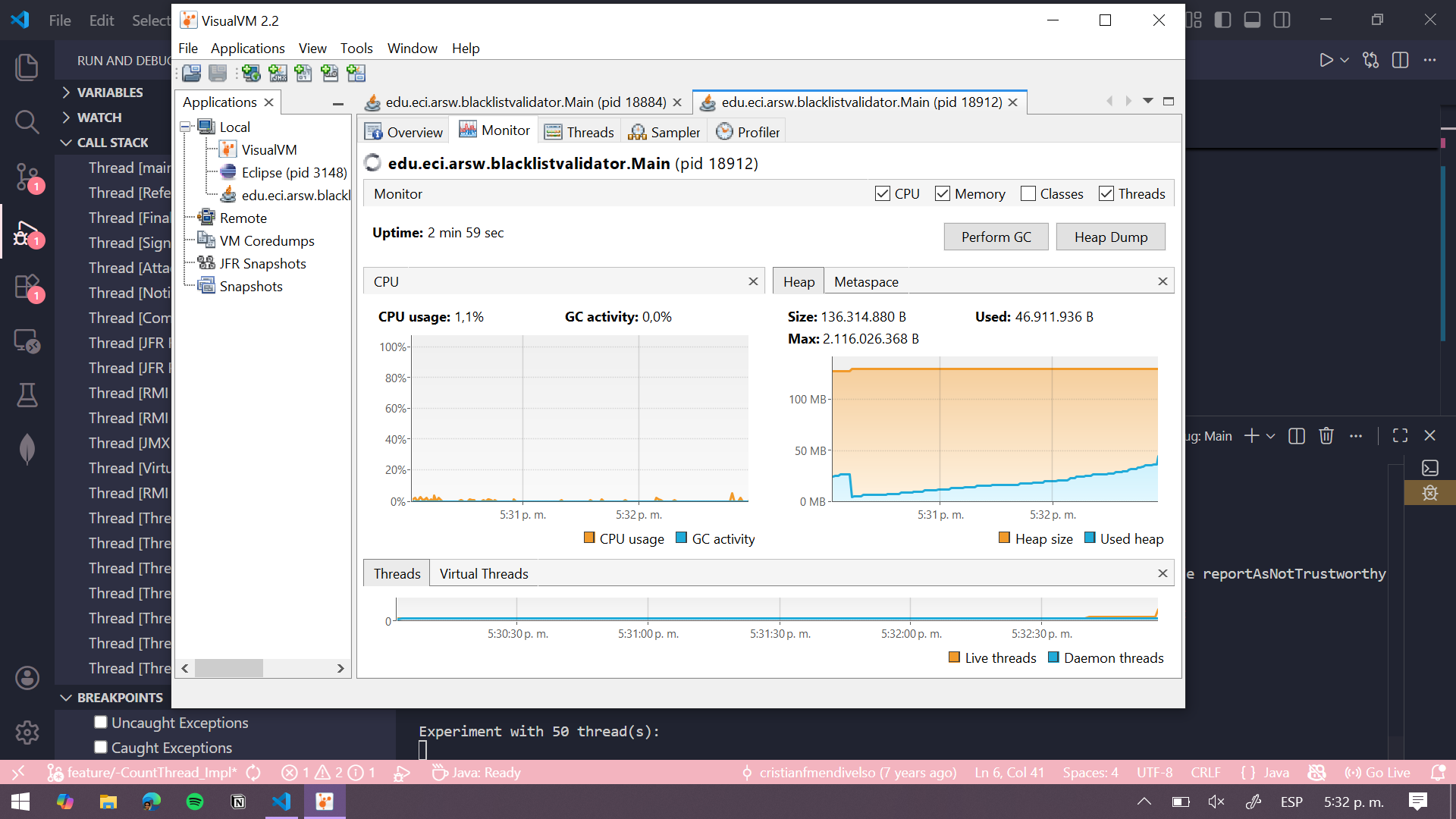
1. Tantos hilos como el doble de núcleos de procesamiento.

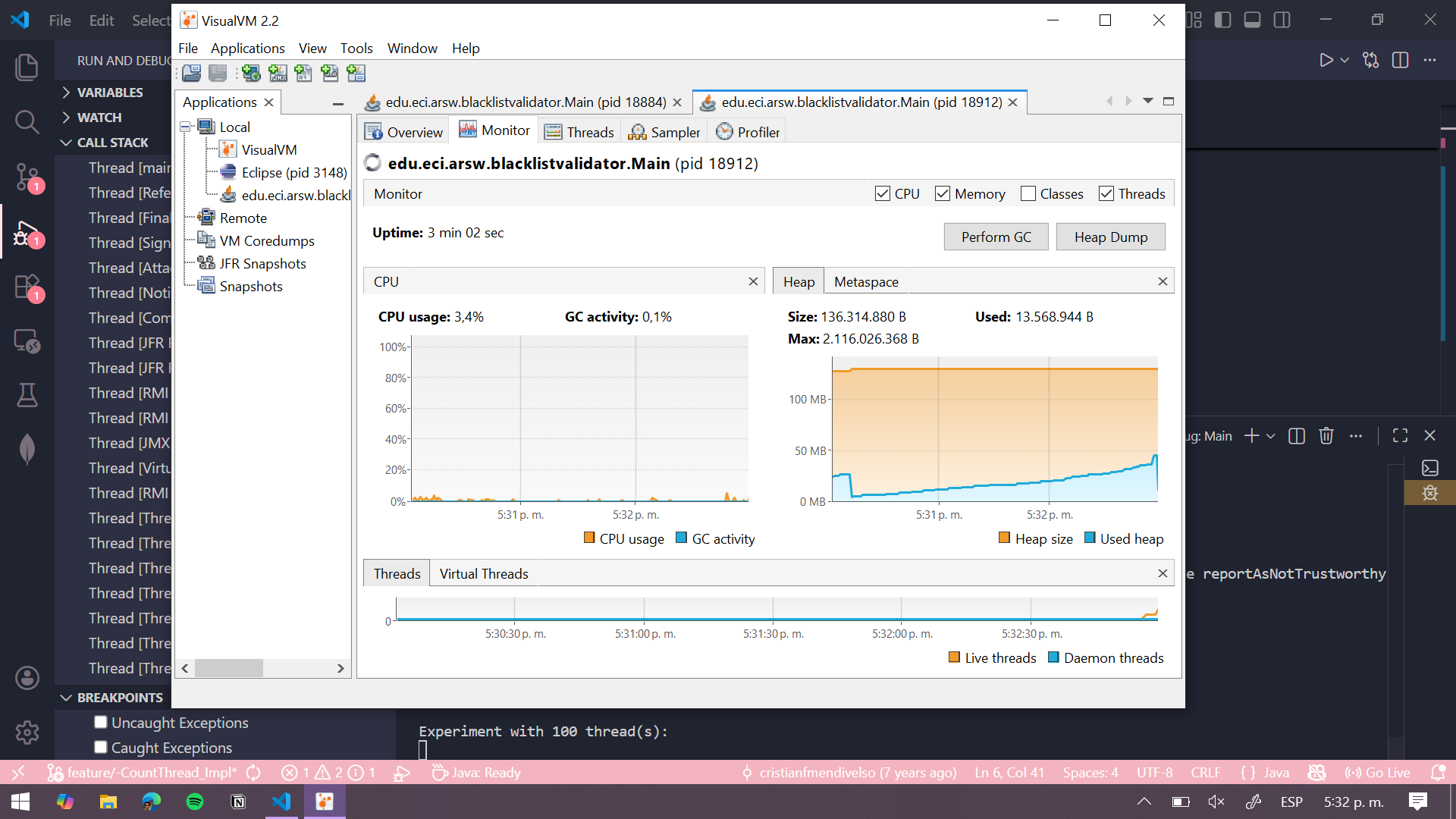
El tiempo que tardaron los 8 hilos en completar la tarea fue de 16251ms, que equivalen a aproximadamente 16,25 segundos. En cuanto al porcentaje de uso de la CPU, este permaneció en varias ocasiones en 0% y no superó el 5,2%.

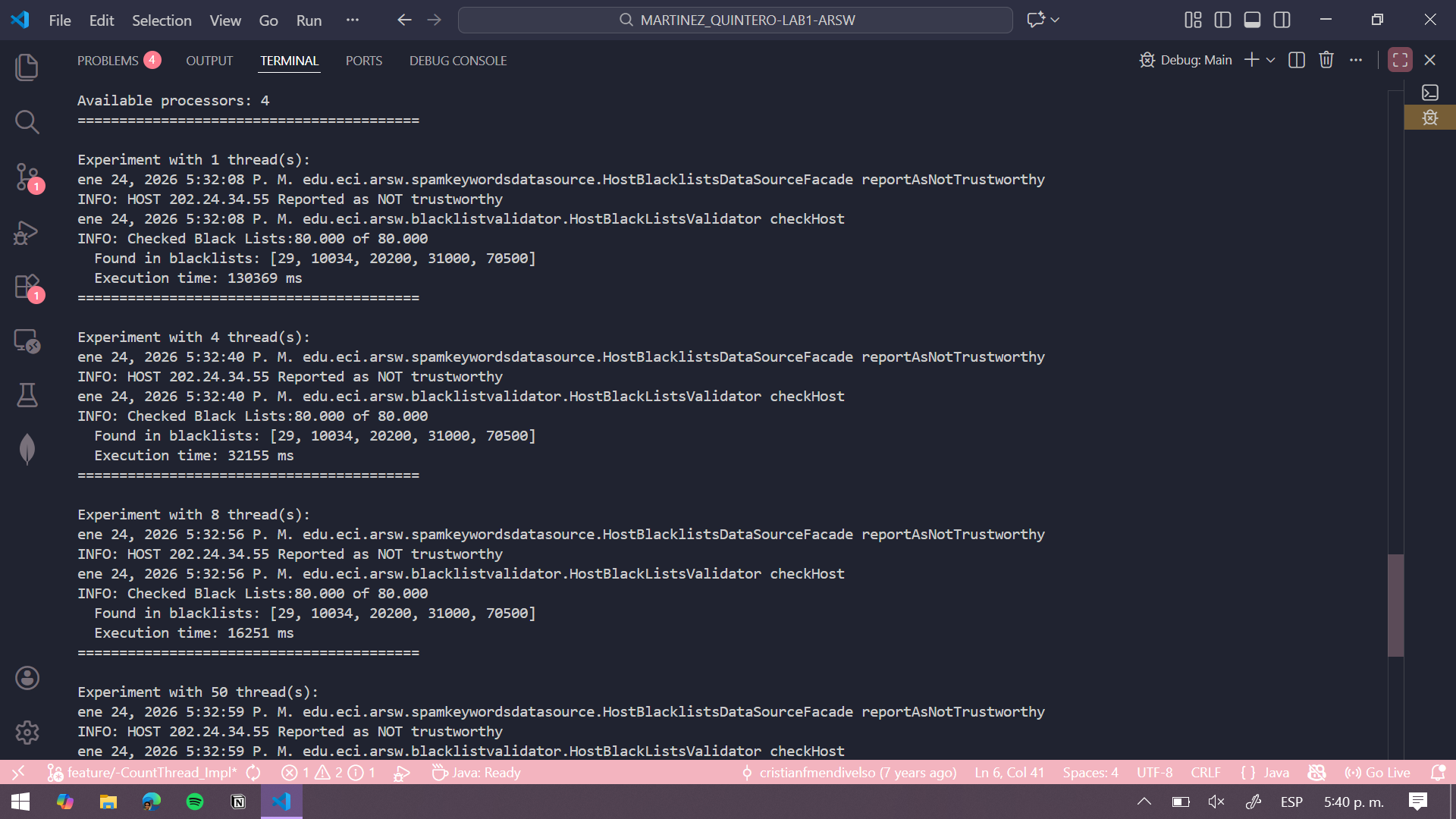
Imágenes del experimento haciendo uso de 8 hilos:

1. 50 hilos.

El tiempo que tardaron los 50 hilos en completar la tarea fue de 2696ms, que equivalen a aproximadamente 2,7 segundos. En cuanto al porcentaje de uso de la CPU, este permaneció en varias ocasiones en 0% y no superó el 1,1%.

Imagen del experimento haciendo uso de 50 hilos:

1. 100 hilos.  
   El tiempo que tardaron los 100 hilos en completar la tarea fue de 1458ms, que equivalen a aproximadamente 1,46 segundos. En cuanto al porcentaje de uso de la CPU, este no superó el 3,4%.  
     
   Imágenes del experimento haciendo uso de 100 hilos:

Las siguientes imágenes tomadas de la terminal luego de finalizar la ejecución del experimento, muestran la información que permitió construir la gráfica de tiempo de solución vs el número de hilos. En estas se puede observar la cantidad de hilos, la respuesta y el tiempo que tardó en ejecutarse.  
  


La tabla y grafica construidas a partir de la información recopilada del experimento se muestran a continuación.

|  |  |
| --- | --- |
| **Tiempo (ms)** | **# Hilos** |
| 130369 | 1 |
| 32155 | 4 |
| 16251 | 8 |
| 2696 | 50 |
| 1458 | 100 |



**Parte IV - Ejercicio Black List Search**

[Diagrama

El contenido generado por IA puede ser incorrecto.](https://github.com/mbquial/MARTINEZ_QUINTERO-LAB1-ARSW/blob/master/img/ahmdahls.png)

1. ¿Por qué el mejor desempeño no se logra con los 500 hilos?, cómo se compara este desempeño cuando se usan 200?

**R/** Precisamente como habla la ley de Amdahls, si un programa no es capaz de ser 100% eficiente con el uso de la CPU, el incremento al desempeño del programa según se agreguen más y más CPU threads será menor. En este caso esta conclusión puede verse de manera clara con los datos graficados en el punto anterior, allí vemos una clara tendencia a que el rendimiento se aproxime a un mínimo (semejante a una asíntota).Es por lo anterior que el mejor desempeño no se logra con 500 hilos y si comparamos con 200 hilos la diferencia no ha de ser representativamente mayor a la que se consigue con 500 hilos, además hay que entender que si estos 500 hilos se ejecutan en una sola máquina, no es que parezcan más recursos, simplemente se dividen más y más en cada hilo.

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

**R/** Se pudo observar que al usar el doble de hilos de procesamiento (8 hilos), el proceso terminó en un tiempo menor al que se evidenció con el uso de 4 hilos. No obstante, el porcentaje de uso de la CPU fue superior en más de un 4%, lo que muestra una mayor demanda de recursos del sistema. Esto sugiere que, aunque el aumento en el número de hilos mejora el tiempo de ejecución gracias a un mayor paralelismo, también aumenta considerablemente el trabajo ejercido por la CPU. En consecuencia, el uso de más hilos que núcleos disponibles puede brindar beneficios en rendimiento hasta cierto punto.

1. De acuerdo con lo anterior, si para este problema en lugar de 100 hilos en una sola CPU se pudiera usar 1 hilo en cada una de 100 máquinas hipotéticas, ¿la ley de Amdahls se aplicaría mejor? Si en lugar de esto se usaran c hilos en 100/c máquinas distribuidas (siendo c es el número de núcleos de dichas máquinas), se mejoraría?

**R/** Si consideramos la ley de Amdahls se aplicaría de una manera similar, esto pues la variable que mide que tan paralelizable es nuestro programa no debería cambiar, eso sí, la razón nos haría pensar que, ya que estos hilos no deben luchar entre sí para tener recurso de los núcleos de una sola máquina, si podría haber una mejora en el desempeño, pero probablemente sería muy poca por que igualmente la ley sigue aplicando. Para responder la pregunta de c hilos en 100/c maquinas, hay que insistir que la fracción paralelizable del programa se mantendrá igual, solo se dividirá entre más o menos procesadores con x hilos, así que teóricamente sucedería lo mismo, el desempeño si podría mejorar un poco pero igualmente tendería a un mínimo teórico y ya.