

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

ARQUITECTURA DE SOFTWARE – ARSW

LABORATORIO 1

INTRODUCCIÓN AL PARALELISMO - HILOS

PROFESOR

JAVIER IVÁN TOQUICA BARRERA

ESTUDIANTES

NICOLÁS ARIZA BARBOSA

MATEO OLAYA GARZÓN

SANTIAGO ANDRÉS ROCHA CRISTANCHO

DAVID EDUARDO VALENCIA CARDONA

BOGOTÁ D.C, AGOSTO 2023

Contenido

Introducción.....	2
Primera parte.....	2
Segunda Parte	6
Tercera Parte.....	8
Conclusiones.....	12

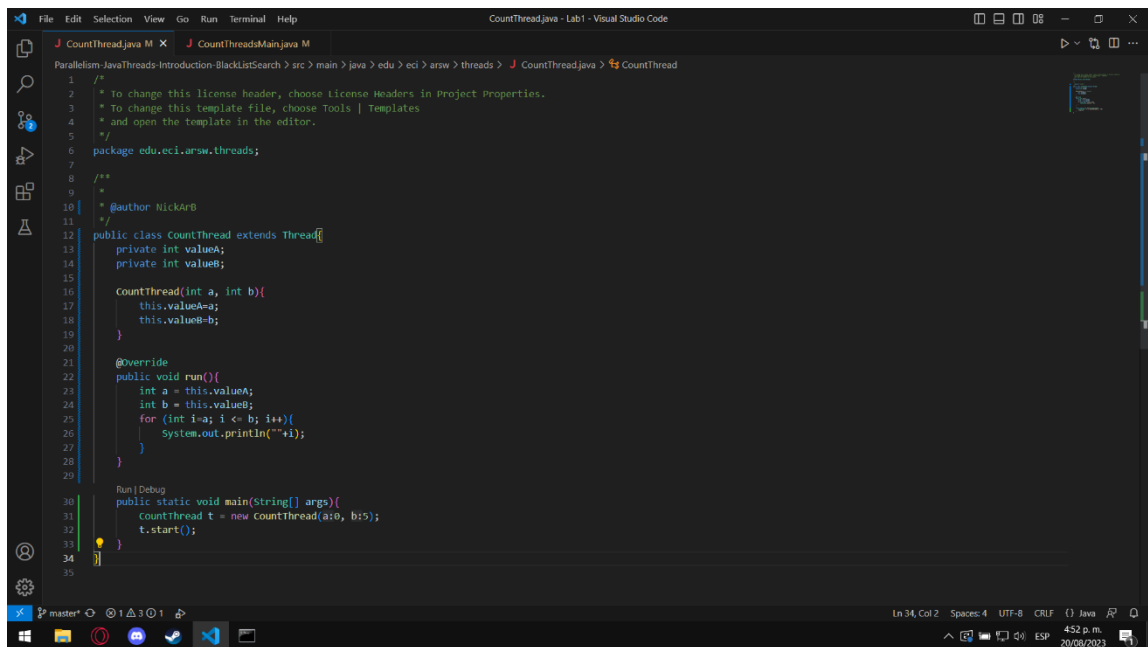
Introducción

Este informe detalla las actividades del laboratorio de arquitectura de software, donde se exploraron conceptos y técnicas relacionadas con el uso de hilos y el paralelismo en la programación. Durante el laboratorio, se abordaron tres partes principales.

- En la Parte I, se trabajó con hilos en Java para entender su ciclo de vida y cómo afecta la ejecución el uso de los métodos `start()` y `run()`.
- En la Parte II, se refactorizó un componente de validación de direcciones IP en listas negras para aprovechar el paralelismo y mejorar su rendimiento.
- Finalmente, en la Parte III se realizaron mediciones de desempeño para tener en cuenta el funcionamiento de los hilos, desde su creación hasta su consumo.

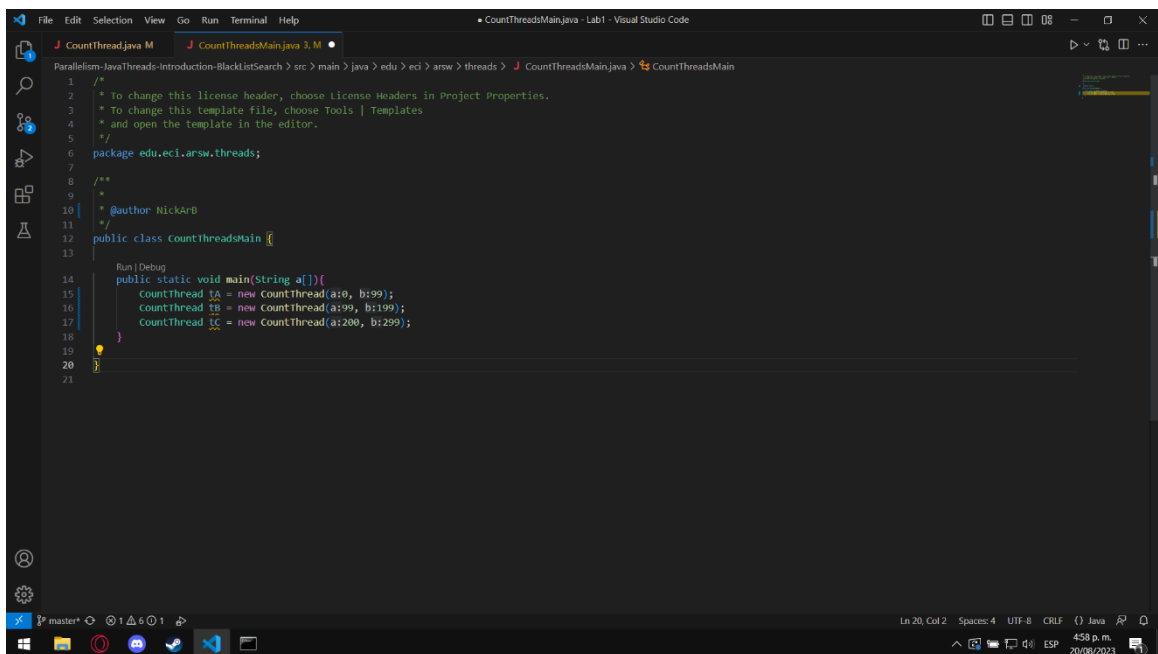
Primera parte

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.



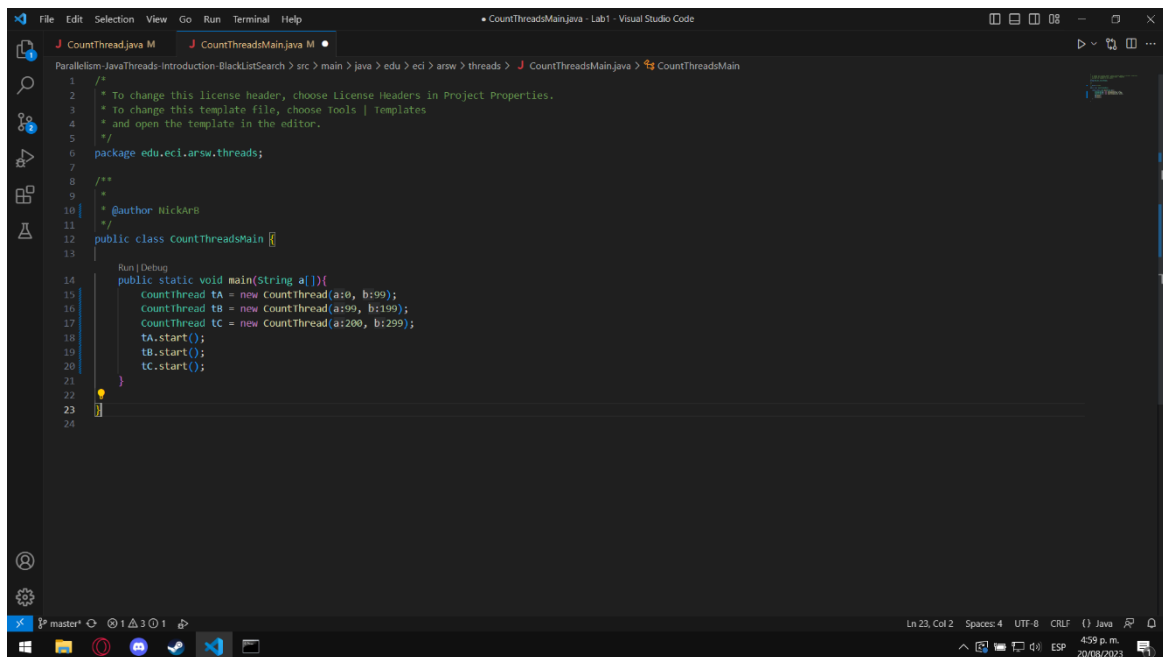
```
1  /**
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package edu.eci.arw.threads;
7
8  /**
9   *
10   * @author NickArB
11   */
12  public class CountThread extends Thread {
13      private int valueA;
14      private int valueB;
15
16      CountThread(int a, int b) {
17          this.valueA = a;
18          this.valueB = b;
19      }
20
21      @Override
22      public void run() {
23          int a = this.valueA;
24          int b = this.valueB;
25          for (int i = a; i <= b; i++) {
26              System.out.println(i);
27          }
28      }
29
30      Run | Debug
31      public static void main(String[] args) {
32          CountThread t = new CountThread(a:10, b:5);
33          t.start();
34      }
35  }
```

2. Complete el método main de la clase CountMainThreads para que:
- Cree 3 hilos de tipo CountThread, asignándole al primero el intervalo [0..99], al segundo [99..199], y al tercero [200..299].



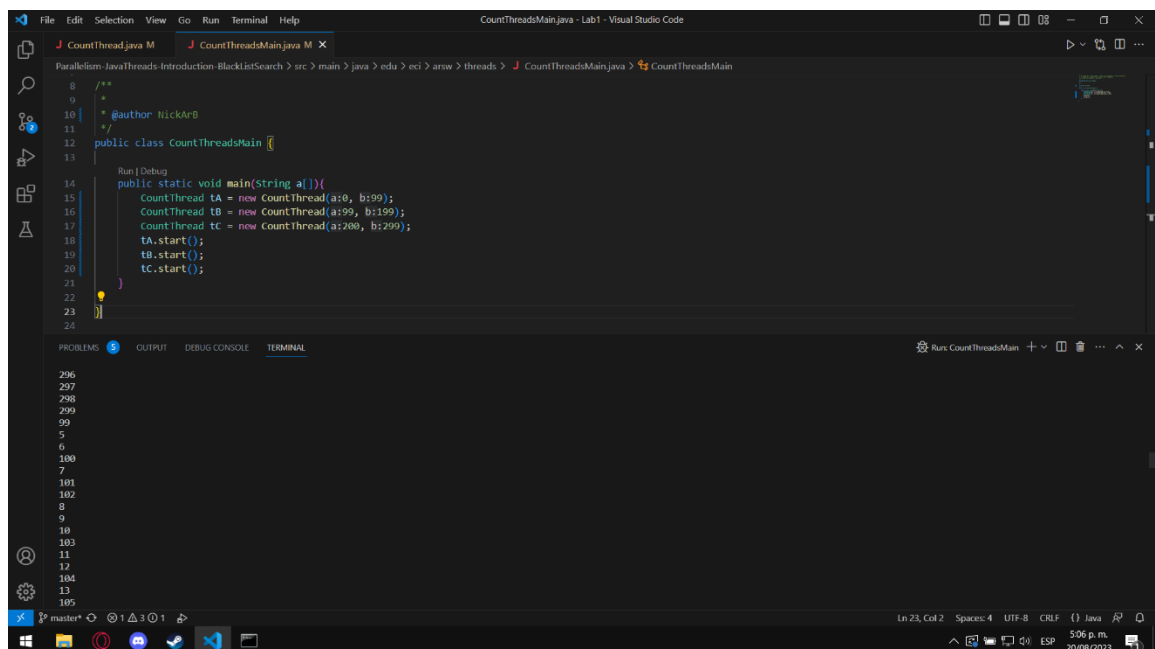
```
1  /**
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package edu.eci.arw.threads;
7
8  /**
9   *
10   * @author NickArB
11   */
12  public class CountThreadsMain {
13
14      Run | Debug
15      public static void main(String a[]) {
16          CountThread tA = new CountThread(a:0, b:99);
17          CountThread tB = new CountThread(a:99, b:199);
18          CountThread tC = new CountThread(a:200, b:299);
19
20
21  }
```

- Inicie los tres hilos con 'start()'.



```
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  /**
9   *
10   * @author NickArB
11   */
12  public class CountThreadsMain {
13
14      Run | Debug
15      public static void main(String a[]){
16          CountThread ta = new CountThread(a:0, b:99);
17          CountThread tb = new CountThread(a:99, b:199);
18          CountThread tc = new CountThread(a:200, b:299);
19          ta.start();
20          tb.start();
21          tc.start();
22      }
23  }
24  }
```

– Ejecute y revise la salida por pantalla.



```
8  /**
9   *
10   * @author NickArB
11   */
12  public class CountThreadsMain {
13
14      Run | Debug
15      public static void main(String a[]){
16          CountThread ta = new CountThread(a:0, b:99);
17          CountThread tb = new CountThread(a:99, b:199);
18          CountThread tc = new CountThread(a:200, b:299);
19          ta.start();
20          tb.start();
21          tc.start();
22      }
23  }
24  }
```

296
297
298
299
99
5
6
100
7
101
102
8
9
10
103
11
12
104
13
105

– Cambie el inicio con 'start()' por 'run()'. ¿Cómo cambia la salida? ¿Por qué?

Con el método start() los threads se ejecutan en paralelo, los números impresos en consola aparecen en desorden. Esta salida cambia si se ejecuta nuevamente.

The screenshot shows the Visual Studio Code editor with a Java file named `CountThreadsMain.java`. The code defines a `CountThread` class and a `main` method. In the `main` method, three `CountThread` objects (TA, TB, and TC) are instantiated with different ranges and then started using the `start()` method. The `main` method is currently selected in the editor. The output window at the bottom shows the execution results, indicating that the threads are running in parallel.

```
8  /**
9  *
10 * @author NickArb
11 */
12 public class CountThreadsMain {
13
14     public static void main(String a[]){
15         CountThread TA = new CountThread(a:0, b:99);
16         CountThread TB = new CountThread(a:99, b:199);
17         CountThread TC = new CountThread(a:200, b:299);
18         TA.start();
19         TB.start();
20         TC.start();
21     }
22 }
23
24
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL

Run | Debug

Run | CountThreadsMain

Ln 23, Col 2 Spaces: 4 UTF-8 CRLF {} Java R

5:12 p.m. 20/06/2023

Mientras que si se cambia la ejecución mediante el método `run()` los threads ya no correrán en paralelo si no que lo harán de forma secuencial como si fuese un llamado de métodos de una clase ordinaria.

The screenshot shows the Visual Studio Code editor with the same `CountThreadsMain.java` file. In this version, the `start()` method has been replaced with the `run()` method for each thread object. The `main` method is still selected. The output window at the bottom shows the execution results, indicating that the threads are running sequentially.

```
8  /**
9  *
10 * @author NickArb
11 */
12 public class CountThreadsMain {
13
14     public static void main(String a[]){
15         CountThread TA = new CountThread(a:0, b:99);
16         CountThread TB = new CountThread(a:99, b:199);
17         CountThread TC = new CountThread(a:200, b:299);
18         TA.run();
19         TB.run();
20         TC.run();
21     }
22 }
23
24
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL

Run | Debug

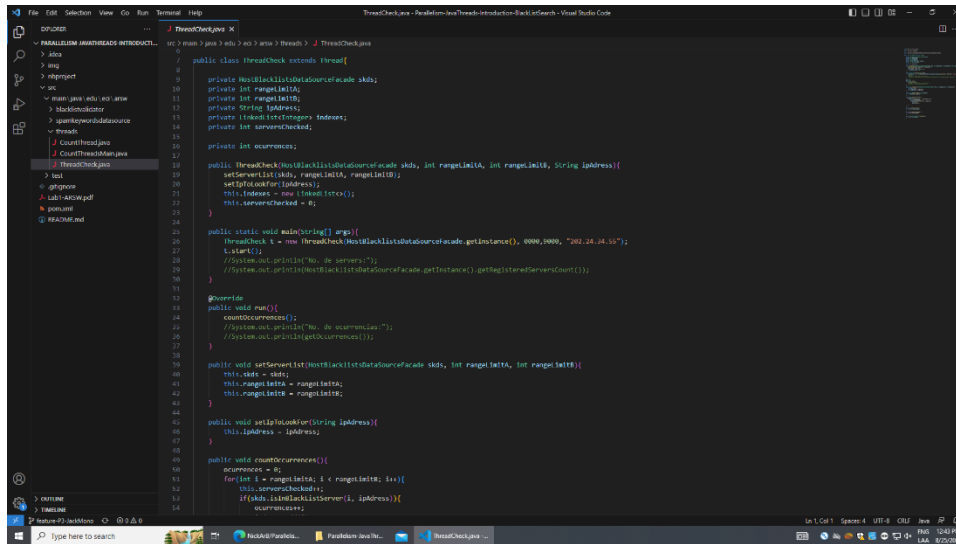
Run | CountThreadsMain

Ln 20, Col 15 Spaces: 4 UTF-8 CRLF {} Java R

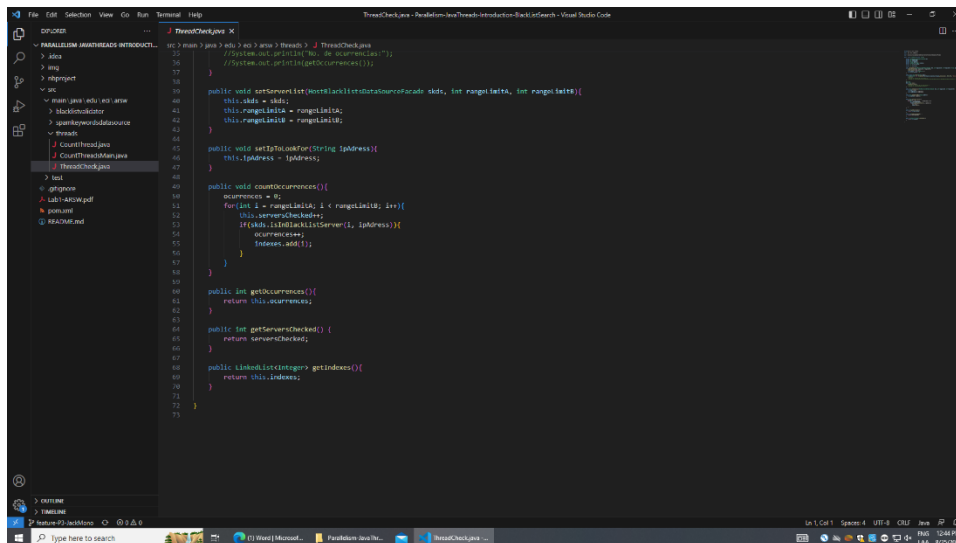
5:15 p.m. 20/06/2023

Segunda Parte

Implementación de la clase Thread:



```
1  public class ThreadCheck extends Thread {
2
3      private HostCheck hostCheck;
4      private int rangeInit;
5      private int rangeEnd;
6      private String ipAddress;
7      private LinkedList<Integer> indexes;
8      private boolean serversChecked;
9
10     private int occurrences;
11
12     public ThreadCheck(HostCheck hostCheck, int rangeInit, int rangeEnd, String ipAddress) {
13         this.hostCheck = hostCheck;
14         this.rangeInit = rangeInit;
15         this.rangeEnd = rangeEnd;
16         this.ipAddress = ipAddress;
17         this.indexes = new LinkedList<>();
18         this.serversChecked = false;
19     }
20
21     public static void main(String[] args) {
22         ThreadCheck t = new ThreadCheck(hostCheck, 0, 100, "192.168.1.1");
23         t.start();
24         //System.out.println("No. de servers:");
25         //System.out.println(hostCheck.getHostCheck().getHostCheck().getRegisteredServerCount());
26     }
27
28     @Override
29     public void run() {
30         countOccurrences();
31         //System.out.println("No. de ocurrencias:");
32         //System.out.println(getOccurrences());
33     }
34
35     public void setServerList(LinkedList<Integer> hostCheck, int rangeInit, int rangeEnd) {
36         this.hostCheck = hostCheck;
37         this.rangeInit = rangeInit;
38         this.rangeEnd = rangeEnd;
39     }
40
41     public void setIpAddress(String ipAddress) {
42         this.ipAddress = ipAddress;
43     }
44
45     public void countOccurrences() {
46         occurrences = 0;
47         for(int i = rangeInit; i < rangeEnd; i++) {
48             this.serversChecked++;
49             if(hostCheck.isHostCheckServer(i, ipAddress)) {
50                 occurrences++;
51             }
52         }
53     }
54 }
```



```
1  public class ThreadCheck extends Thread {
2
3      private HostCheck hostCheck;
4      private int rangeInit;
5      private int rangeEnd;
6      private String ipAddress;
7      private LinkedList<Integer> indexes;
8      private boolean serversChecked;
9
10     private int occurrences;
11
12     public ThreadCheck(HostCheck hostCheck, int rangeInit, int rangeEnd, String ipAddress) {
13         this.hostCheck = hostCheck;
14         this.rangeInit = rangeInit;
15         this.rangeEnd = rangeEnd;
16         this.ipAddress = ipAddress;
17         this.indexes = new LinkedList<>();
18         this.serversChecked = false;
19     }
20
21     public static void main(String[] args) {
22         ThreadCheck t = new ThreadCheck(hostCheck, 0, 100, "192.168.1.1");
23         t.start();
24         //System.out.println("No. de servers:");
25         //System.out.println(hostCheck.getHostCheck().getHostCheck().getRegisteredServerCount());
26     }
27
28     @Override
29     public void run() {
30         countOccurrences();
31         //System.out.println("No. de ocurrencias:");
32         //System.out.println(getOccurrences());
33     }
34
35     public void setServerList(LinkedList<Integer> hostCheck, int rangeInit, int rangeEnd) {
36         this.hostCheck = hostCheck;
37         this.rangeInit = rangeInit;
38         this.rangeEnd = rangeEnd;
39     }
40
41     public void setIpAddress(String ipAddress) {
42         this.ipAddress = ipAddress;
43     }
44
45     public void countOccurrences() {
46         occurrences = 0;
47         for(int i = rangeInit; i < rangeEnd; i++) {
48             this.serversChecked++;
49             if(hostCheck.isHostCheckServer(i, ipAddress)) {
50                 occurrences++;
51                 indexes.add(i);
52             }
53         }
54     }
55
56     public int getOccurrences() {
57         return this.occurrences;
58     }
59
60     public int getServersChecked() {
61         return serversChecked;
62     }
63
64     public LinkedList<Integer> getIndexes() {
65         return this.indexes;
66     }
67
68 }
```

Cambios en el método de checkHost:

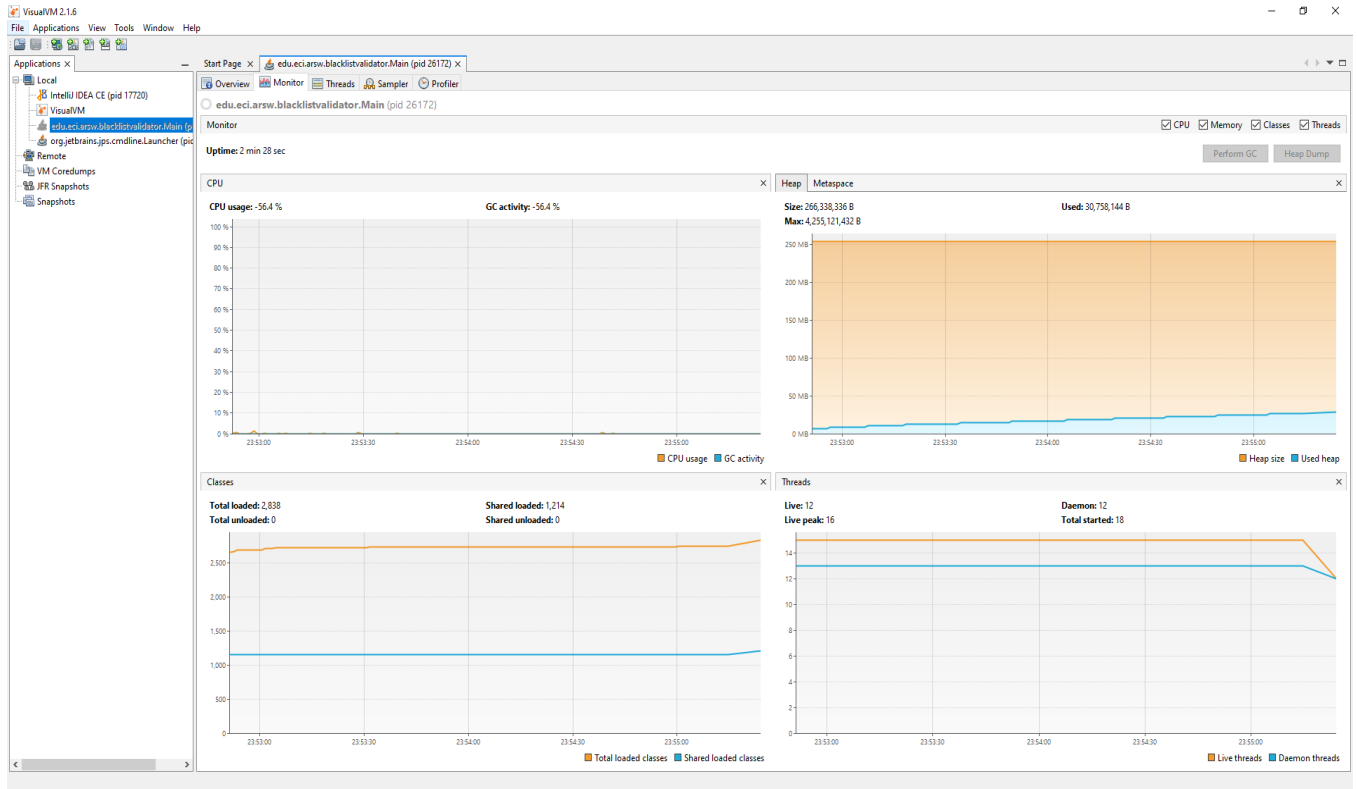
```
1  // HostBlacklistValidator.java
2  ...
3  ...
4  ...
5  ...
6  ...
7  ...
8  ...
9  ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ...
38 ...
39 ...
40 ...
41 ...
42 ...
43 ...
44 ...
45 ...
46 ...
47 ...
48 ...
49 ...
50 ...
51 ...
52 ...
53 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
96 ...
97 ...
98 ...
99 ...
100 ...
```

```
1  // HostBlacklistValidator.java
2  ...
3  ...
4  ...
5  ...
6  ...
7  ...
8  ...
9  ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
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 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
96 ...
97 ...
98 ...
99 ...
100 ...
```

Llamado del main:

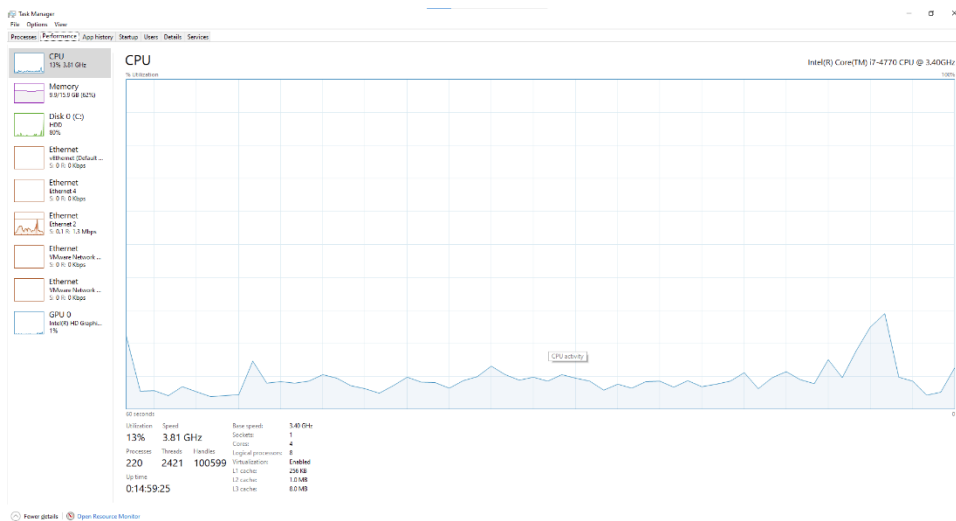
```
1  // Main.java
2  ...
3  ...
4  ...
5  ...
6  ...
7  ...
8  ...
9  ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
```

Con un hilo:

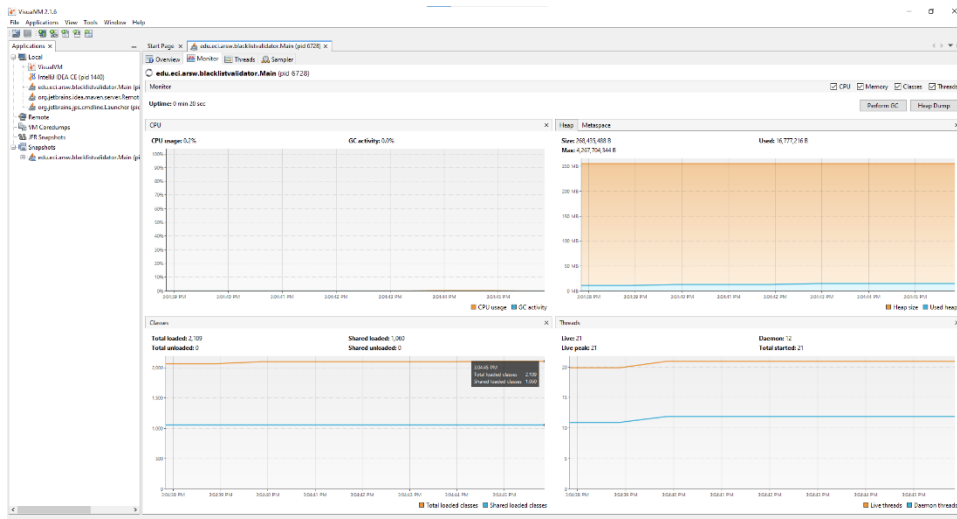


Tantos hilos como núcleos de procesamiento:

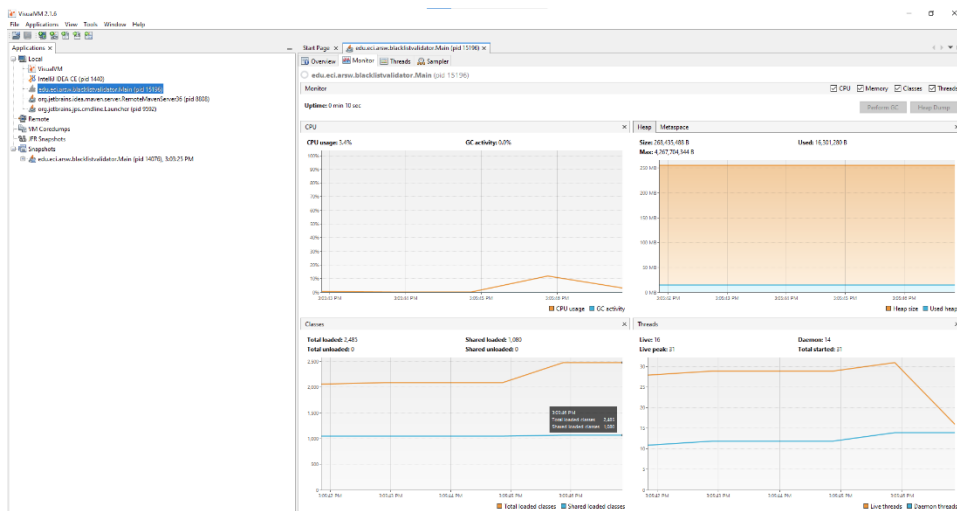
Usando un computador del laboratorio de sistemas con las siguientes características de procesador:



Eso quiere decir que en este caso se hará con 8 hilos



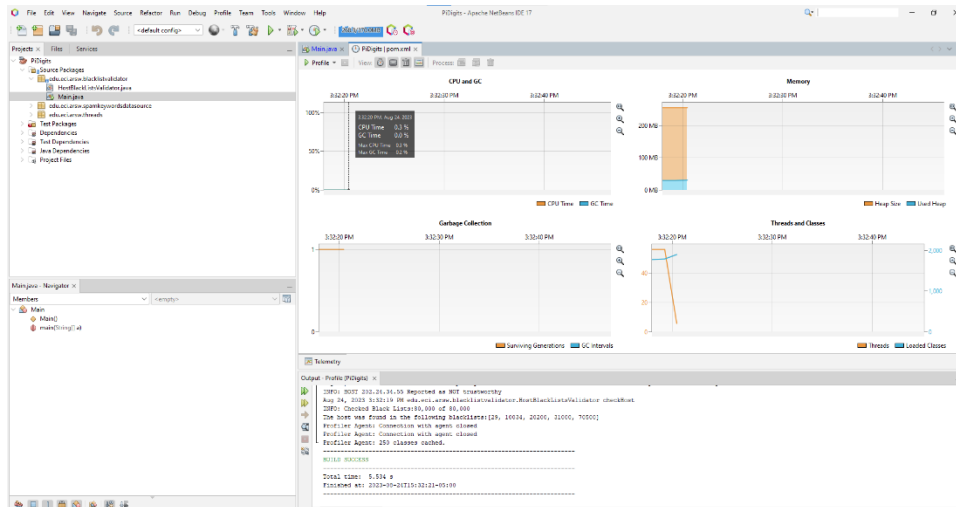
Con el Doble de hilos es decir 16 hilos



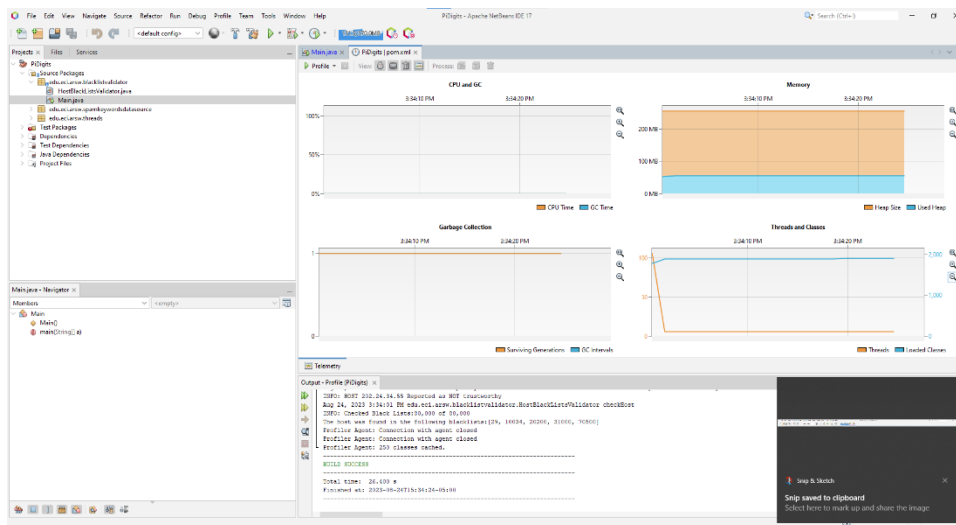
En cuanto a los 50 y 100 hilos habrá un problema ya que al parecer el programa no lo monitoriza como debe dejando vacío el monitor.

Esto se tuvo que usar usando el perfilador de NETBEANS:

50 Hilos:



100 Hilos



Parte de análisis:

1. Para la ley de amdahl se tiene lo siguiente:

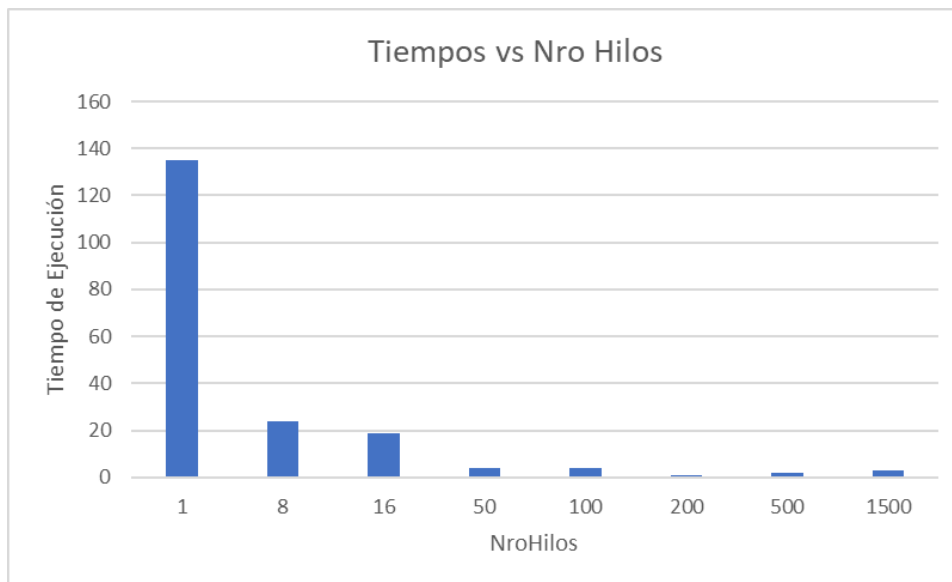
$$S(n) = \frac{1}{(1 - P)\left(\frac{p}{n}\right)}$$

Ahora para hallar ese p se requiere:

$$P = \frac{(T_{\text{secuencial}} - T_{\text{paralelizado}})}{T_{\text{secuencial}}}$$

Analisis experimental:

Tiempo Ejecucion(Segundo)	N(NroHilos)
135	1
24	8
19	16
4	50
4	100
1	200
2	500
3	1500



Y Según la ley:

Para 100 hilos

$$P = \frac{(135 - 4)}{135} = 0.97$$

$$S(100) = \frac{1}{(1 - 0.97) \cdot \left(\frac{0.97}{100}\right)} = 3436.42$$

Es decir la mejora del rendimiento es 3436 veces mejor que al hacerlo paralelo

RTA 1 : Según el análisis experimental ya se puede inferir que en 500 hilos la mejora no aumenta si no que al contrario decae con respecto a 200 hilos, seguramente por los diferentes procesos por los que tiene que pasar ya que se tiene que usar un reuso de threads en el procesador.

La mejor configuración para el procesador al parecer es con 200 hilos para este trabajo, y por lo que se puede ver en experimentación, el monitor de recursos hace un mejor trabajo administrando el reuso de esos threads.

RTA 2 :

En un intel I5-8300 tiene 4 nucleos el resultado en tiempo es : 80 segundos

Con hilos equivalentes al doble de nucleos el resultado es : 24 segundos

La mejora no es la mitad es mas, y para ser exactos esta mejora corresponde al :70% de mejora

RTA 3:

Se cree que la mejor decision es hacer la paralelizacion, ya que si se hace de manera unica o secuencial en 100 computadores diferentes se demorara mas, para esto se hace la siguiente operación:

Hacer las divisiones en diferentes pc nunca sera la manera mas rapidas de hacer la ejecucion ya que todo el tiempo se estaria haciendo las cosas sin una porcion de paralizacion como dice la ley de amdahl, por ende seria mucho mejor ademas de paralelizar los computadores tambien paralelizar los hilos usados por computador ya que así la p seria mas grande(mas cercana a 1) y por ende la mejora seria muchas mas veces más rapida que hacerlo nada más paralelizando computadores

Conclusiones

El laboratorio proporcionó una valiosa oportunidad para explorar y experimentar con conceptos fundamentales relacionados con hilos y paralelismo en la programación. A través de la implementación de casos de estudio prácticos, se logró comprender cómo los hilos pueden utilizarse para ejecutar tareas concurrentemente y cómo el paralelismo puede ser aprovechado para mejorar el rendimiento de aplicaciones.

En la Parte I, se analizó cómo los hilos permiten la ejecución simultánea de tareas y cómo los métodos `start()` y `run()` afectan esta ejecución. Se comprendió que `start()` crea un nuevo hilo para ejecutar el método `run()`, permitiendo una ejecución concurrente, mientras que `run()` ejecuta el método en el hilo actual, lo que resulta en una ejecución secuencial.

La Parte II introdujo el concepto de paralelismo aplicado a la validación de direcciones IP en listas negras. Con la creación de múltiples hilos, se dividió la tarea en subproblemas independientes que pudieron resolverse en paralelo, mejorando significativamente el rendimiento en comparación con un enfoque secuencial. La implementación de una estrategia que permitiera detener la búsqueda una vez que se alcanzara el umbral necesario de ocurrencias en las listas negras, demostró ser una forma eficiente de optimizar el proceso.

Los experimentos de rendimiento realizados en la Parte III brindaron información valiosa sobre cómo la utilización de diferentes cantidades de hilos afecta el tiempo de ejecución y los recursos del sistema. Se observó que, según la Ley de Amdahl, el rendimiento no necesariamente mejora indefinidamente con un mayor número de hilos debido a la fracción no paralelizable de la tarea. Además, se evidenció que la relación entre el número de hilos y el rendimiento puede depender de la arquitectura del sistema y la distribución de recursos.