

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

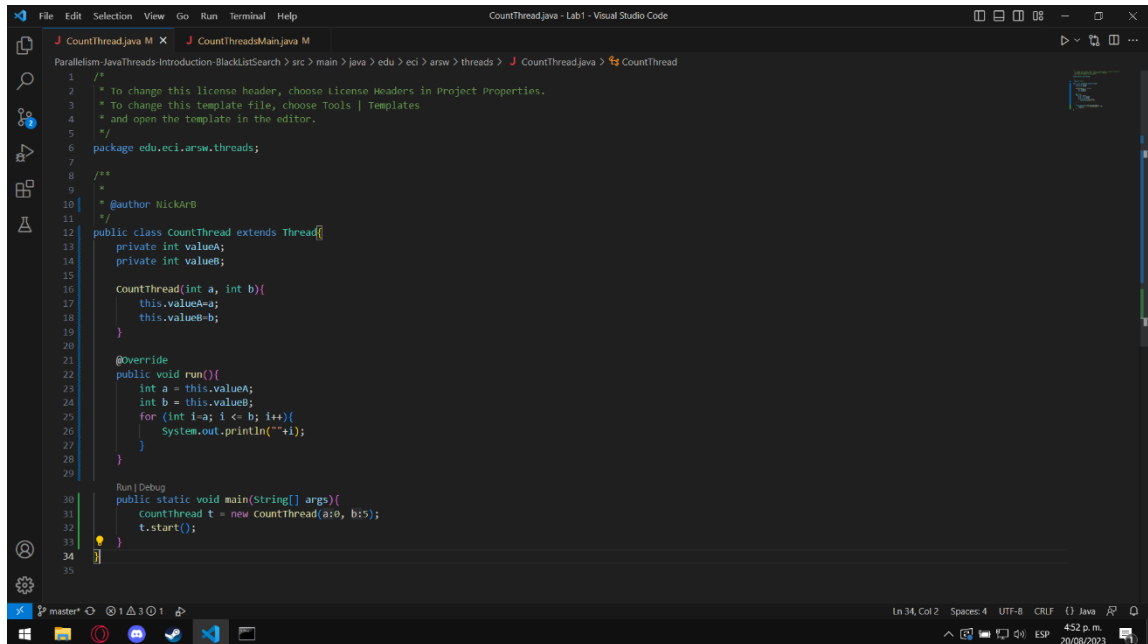
SANTIAGO ANDRÉS ROCHA CRISTANCHO

DAVID EDUARDO VALENCIA CARDONA

BOGOTÁ D.C, AGOSTO 2023

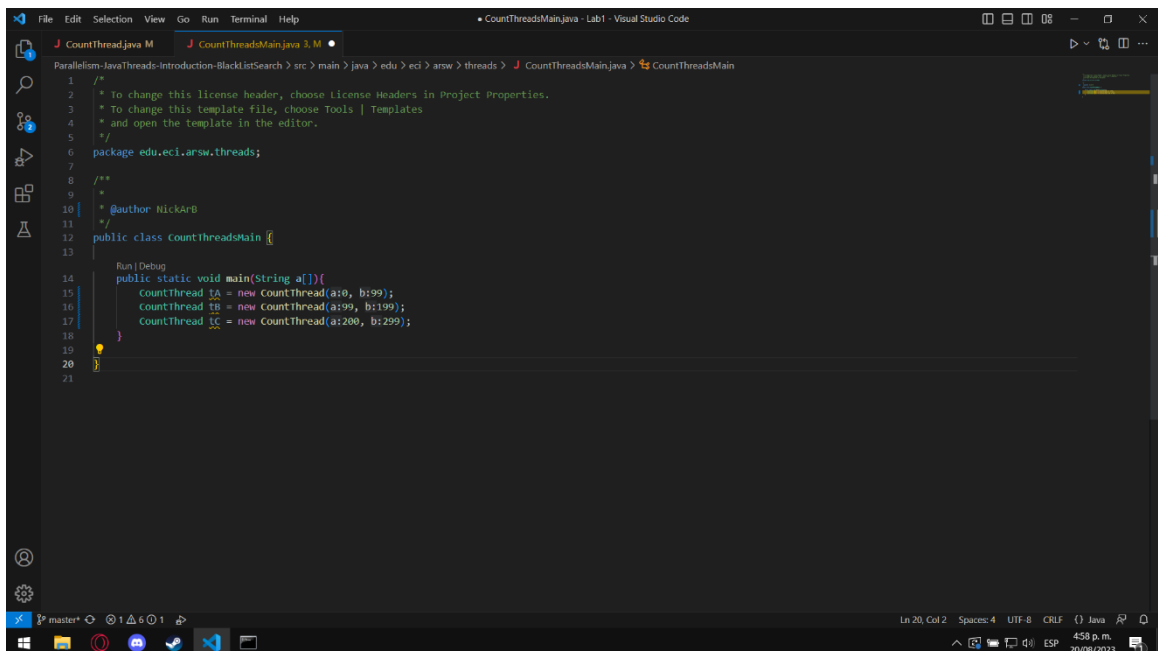
Primera parte

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.



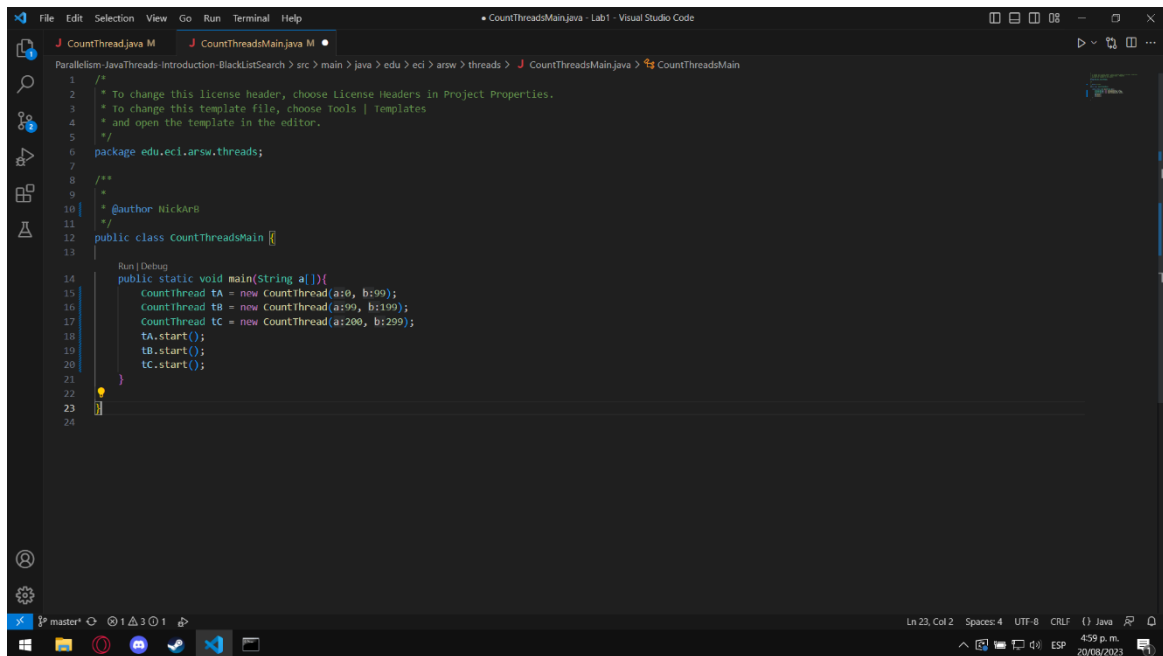
```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package edu.eci.arsw.threads;
7
8  /**
9   *
10   * @author NickArB
11   */
12  public class CountThread 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(a10, b15);
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].



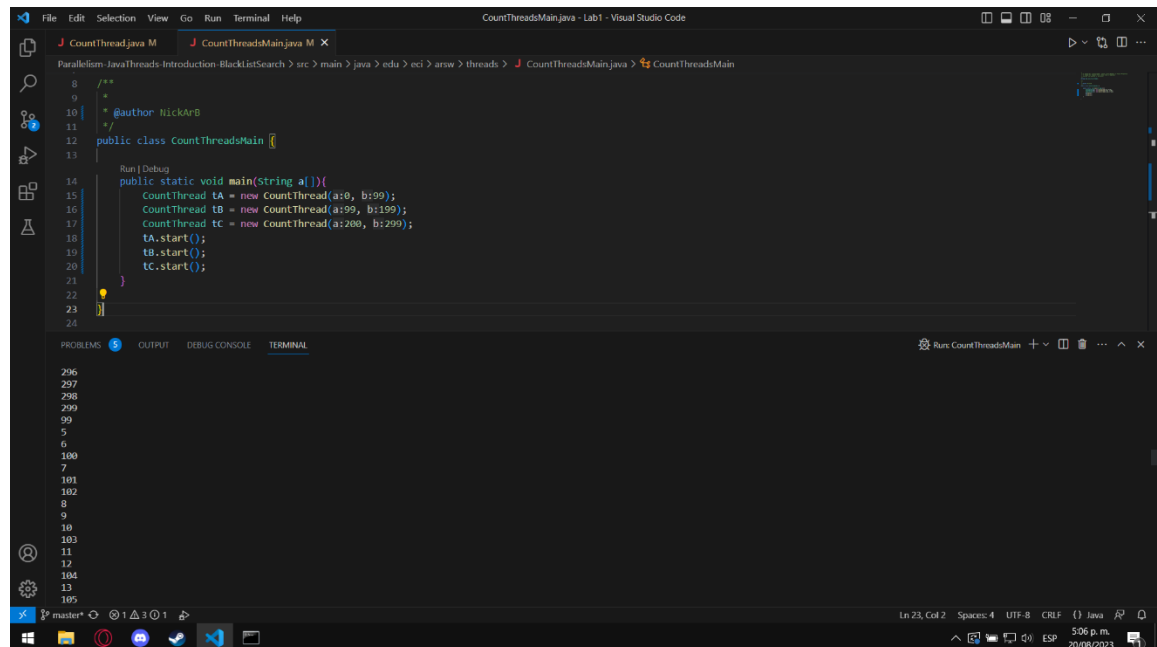
```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package edu.eci.arsw.threads;
7
8  /**
9   *
10   * @author NickArB
11   */
12  public class CountThreadsMain {
13
14      Run | Debug
15      public static void main(String a[]){
16          CountThread tA = new CountThread(a10, b99);
17          CountThread tB = new CountThread(a99, b199);
18          CountThread tC = new CountThread(a200, b299);
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   *
11   * @author NickAr8
12   */
13  public class CountThreadsMain {
14
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  }
```

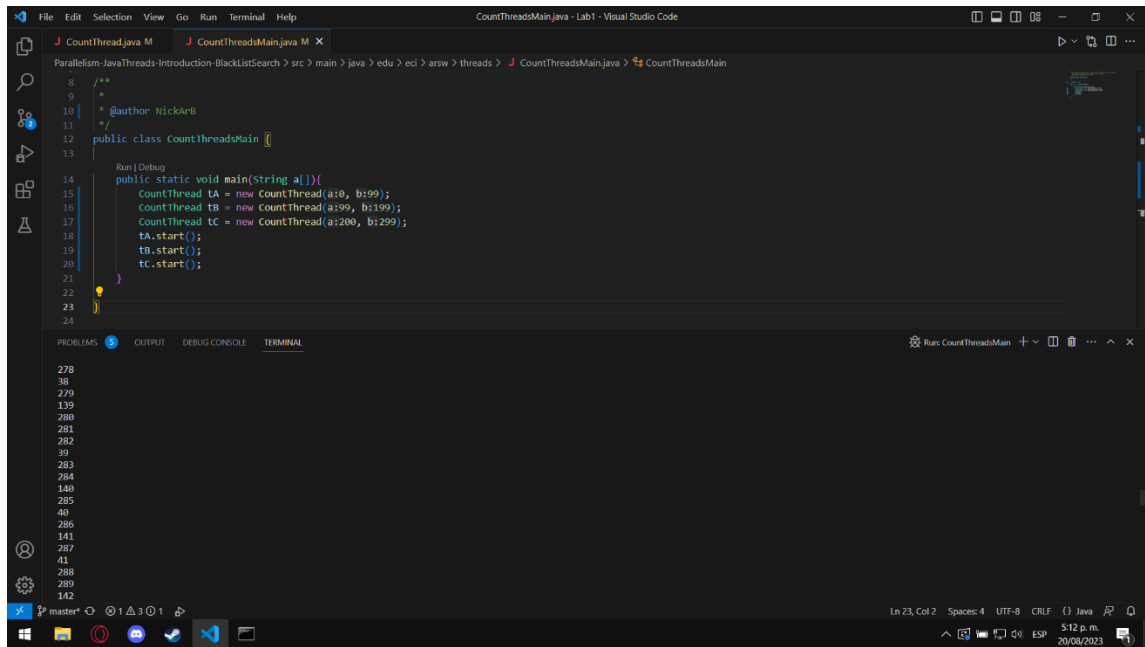
- Ejecute y revise la salida por pantalla.



```
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   *
11   * @author NickAr8
12   */
13  public class CountThreadsMain {
14
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  }
```

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

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



```
8  /**
9  *
10 * @author NickAR8
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
```

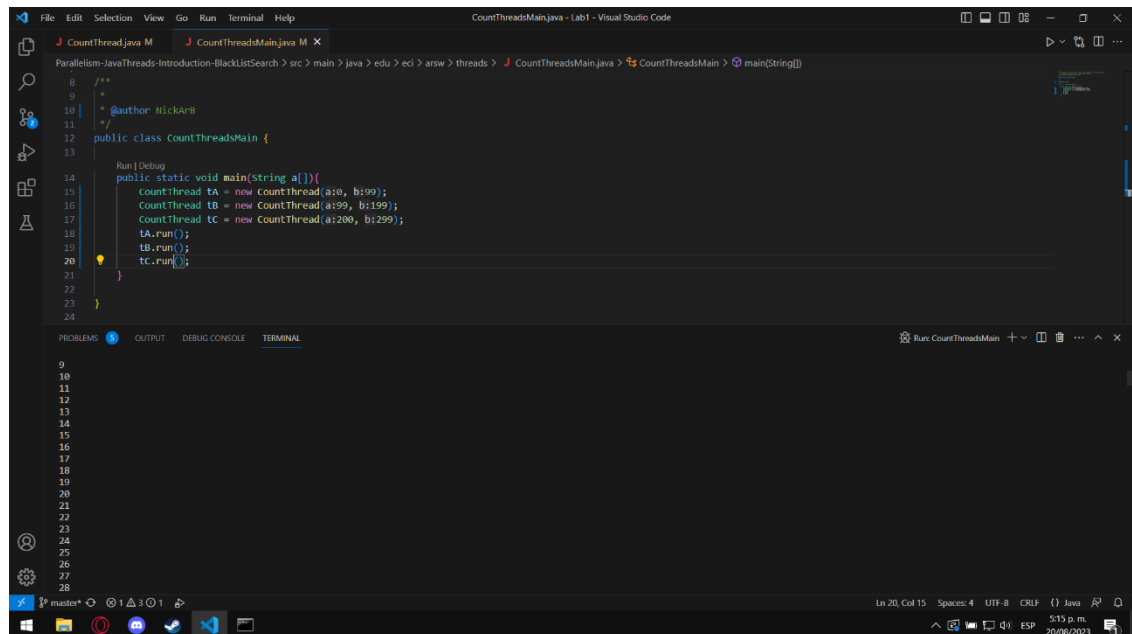
PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL

Run: CountThreadsMain

278
38
279
139
280
281
282
49
283
284
140
285
40
286
141
287
41
288
289
142

Ln 23, Col 2 Spaces: 4 UTF-8 CRLF Java 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.



```
8  /**
9  *
10 * @author NickAR8
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: CountThreadsMain

9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

Ln 20, Col 15 Spaces: 4 UTF-8 CRLF Java 20/06/2023

Segunda Parte:

Implementación de la clase Thread:

```
1 import java.net.*;
2 import java.util.*;
3
4 public class ThreadCheck extends Thread{
5
6     private MulticastSocket sourceSocket;
7     private int rangeInitA;
8     private int rangeInitB;
9     private String ipAddress;
10    private LinkedList<Integer> indexes;
11    private int serverChecked;
12
13    private int occurrences;
14
15    public ThreadCheck(MulticastSocket sourceSocket, int rangeInitA, int rangeInitB, String ipAddress){
16        setServerList(sourceSocket, rangeInitA, rangeInitB);
17        setIpAddress(ipAddress);
18        this.indexes = new LinkedList<>();
19        this.serverChecked = 0;
20    }
21
22    public static void main(String[] args){
23        ThreadCheck t = new ThreadCheck(MulticastSocketFactory.getInstance(), 0, 0, "0.0.0.0", "192.168.1.1");
24        t.start();
25        //System.out.println("No. de servers:");
26        //System.out.println(MulticastSocketFactory.getInstance().getRegisteredServerCount());
27    }
28
29    @Override
30    public void run(){
31        countOccurrences();
32        //System.out.println("No. de occurrences:");
33        //System.out.println(getOccurrences());
34    }
35
36    public void setServerList(MulticastSocket sourceSocket, int rangeInitA, int rangeInitB){
37        this.sourceSocket = sourceSocket;
38        this.rangeInitA = rangeInitA;
39        this.rangeInitB = rangeInitB;
40    }
41
42    public void setIpAddress(String ipAddress){
43        this.ipAddress = ipAddress;
44    }
45
46    public void countOccurrences(){
47        occurrences = 0;
48        for(int i = rangeInitA; i < rangeInitB; i++){
49            this.serverChecked++;
50            if(MulticastSocketFactory.getInstance().getRegisteredServer(i, ipAddress)){
51                occurrences++;
52            }
53        }
54    }
55}
```

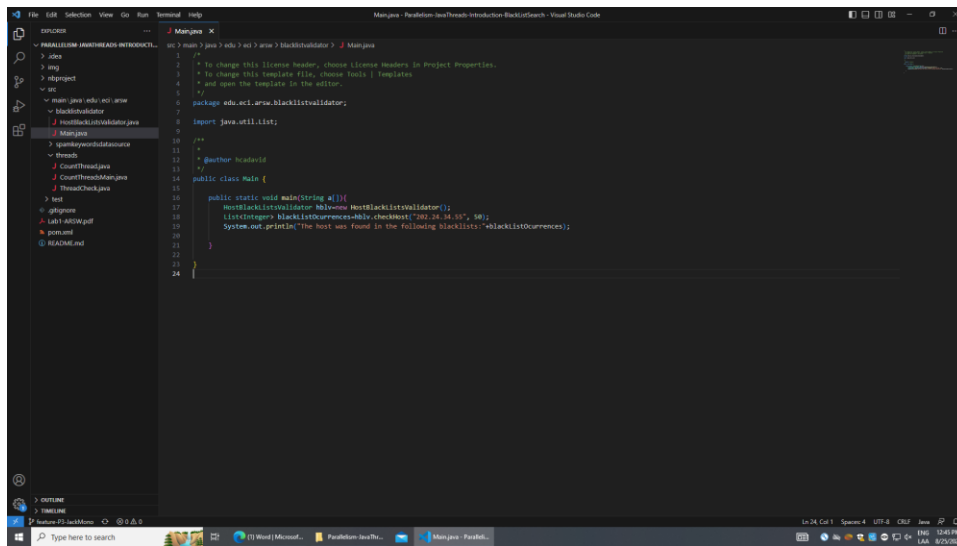
```
1 import java.net.*;
2 import java.util.*;
3
4 public class ThreadCheck extends Thread{
5
6     private MulticastSocket sourceSocket;
7     private int rangeInitA;
8     private int rangeInitB;
9     private String ipAddress;
10    private LinkedList<Integer> indexes;
11    private int serverChecked;
12
13    private int occurrences;
14
15    public ThreadCheck(MulticastSocket sourceSocket, int rangeInitA, int rangeInitB, String ipAddress){
16        setServerList(sourceSocket, rangeInitA, rangeInitB);
17        setIpAddress(ipAddress);
18        this.indexes = new LinkedList<>();
19        this.serverChecked = 0;
20    }
21
22    public static void main(String[] args){
23        ThreadCheck t = new ThreadCheck(MulticastSocketFactory.getInstance(), 0, 0, "0.0.0.0", "192.168.1.1");
24        t.start();
25        //System.out.println("No. de servers:");
26        //System.out.println(MulticastSocketFactory.getInstance().getRegisteredServerCount());
27    }
28
29    @Override
30    public void run(){
31        countOccurrences();
32        //System.out.println("No. de occurrences:");
33        //System.out.println(getOccurrences());
34    }
35
36    public void setServerList(MulticastSocket sourceSocket, int rangeInitA, int rangeInitB){
37        this.sourceSocket = sourceSocket;
38        this.rangeInitA = rangeInitA;
39        this.rangeInitB = rangeInitB;
40    }
41
42    public void setIpAddress(String ipAddress){
43        this.ipAddress = ipAddress;
44    }
45
46    public void countOccurrences(){
47        occurrences = 0;
48        for(int i = rangeInitA; i < rangeInitB; i++){
49            this.serverChecked++;
50            if(MulticastSocketFactory.getInstance().getRegisteredServer(i, ipAddress)){
51                occurrences++;
52                indexes.add(i);
53            }
54        }
55    }
56
57    public int getOccurrences(){
58        return this.occurrences;
59    }
60
61    public int getServerChecked(){
62        return serverChecked;
63    }
64
65    public LinkedList<Integer> getIndexes(){
66        return this.indexes;
67    }
68
69    public void setSourceSocket(MulticastSocket sourceSocket){
70        this.sourceSocket = sourceSocket;
71    }
72}
```

Cambios en el método de checkHost:

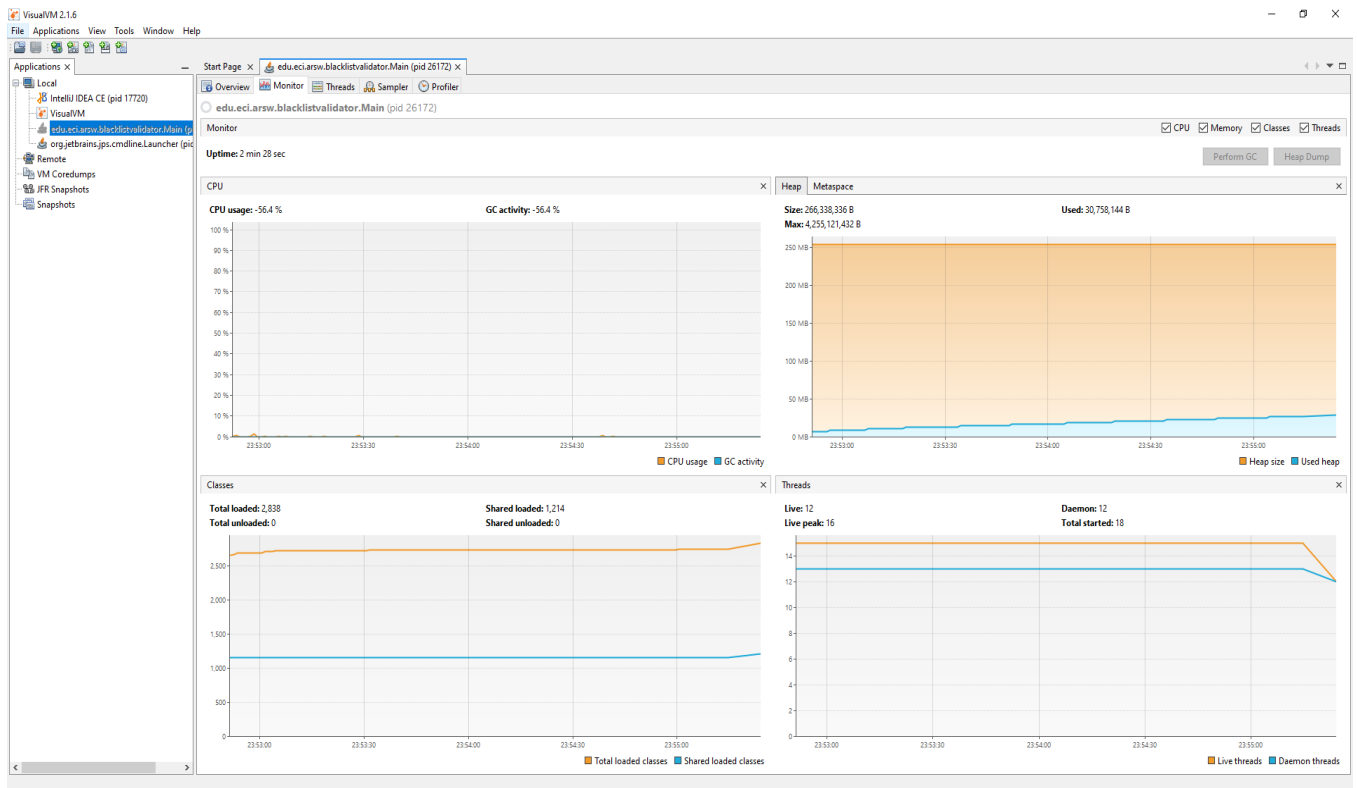
```
1 // HostBlacklistValidator.java
2 //
3 //
4 //
5 //
6 //
7 //
8 //
9 //
10 //
11 //
12 //
13 //
14 //
15 //
16 //
17 //
18 //
19 //
20 //
21 //
22 //
23 //
24 //
25 //
26 //
27 //
28 //
29 //
30 //
31 //
32 //
33 //
34 //
35 //
36 //
37 //
38 //
39 //
40 //
41 //
42 //
43 //
44 //
45 //
46 //
47 //
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 //
101 //
102 //
103 //
104 //
105 //
106 //
107 //
108 //
109 //
110 //
111 //
112 //
113 //
114 //
115 //
116 //
117 //
118 //
119 //
120 //
121 //
122 //
123 //
124 //
125 //
126 //
127 //
128 //
129 //
130 //
131 //
132 //
133 //
134 //
135 //
136 //
137 //
138 //
139 //
140 //
141 //
142 //
143 //
144 //
145 //
146 //
147 //
148 //
149 //
150 //
```

Llamado del main:

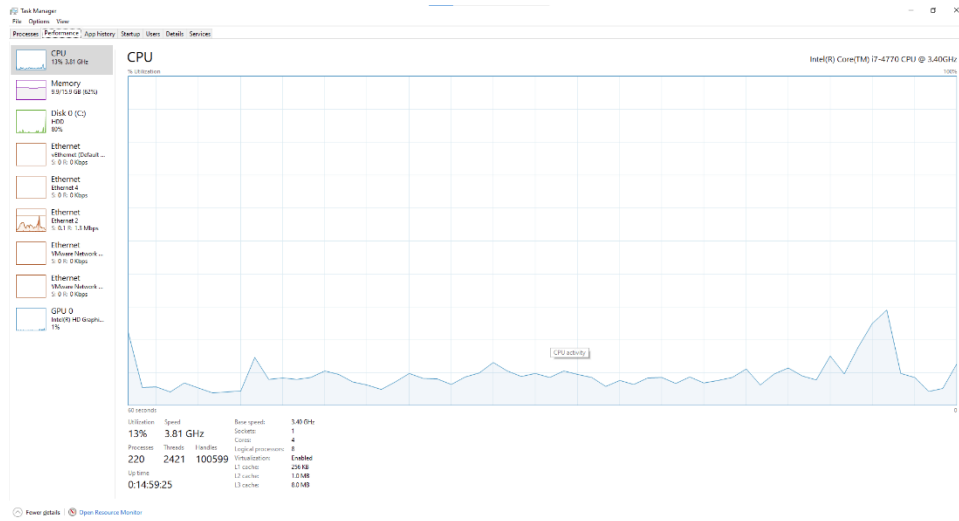


Tercera Parte: Con un hilo:

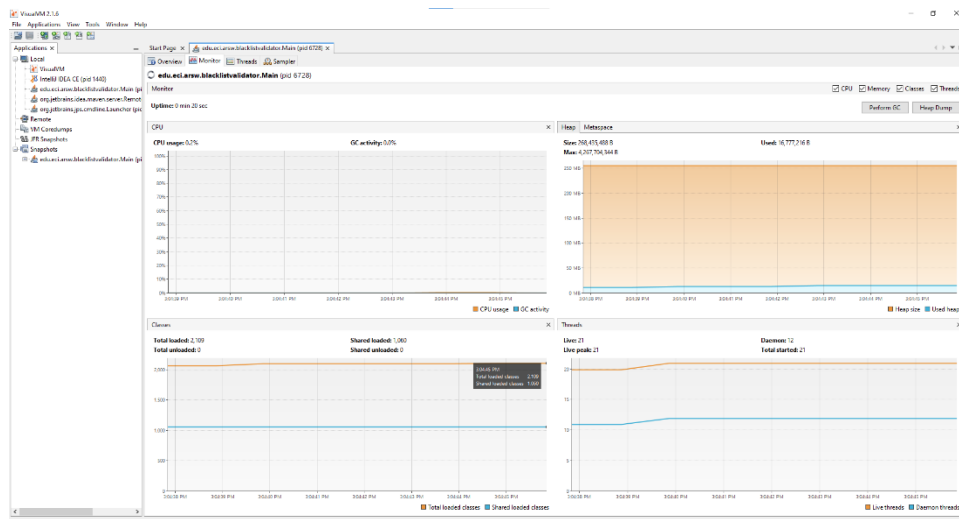


Tantos hilos como núcleos de procesamiento:

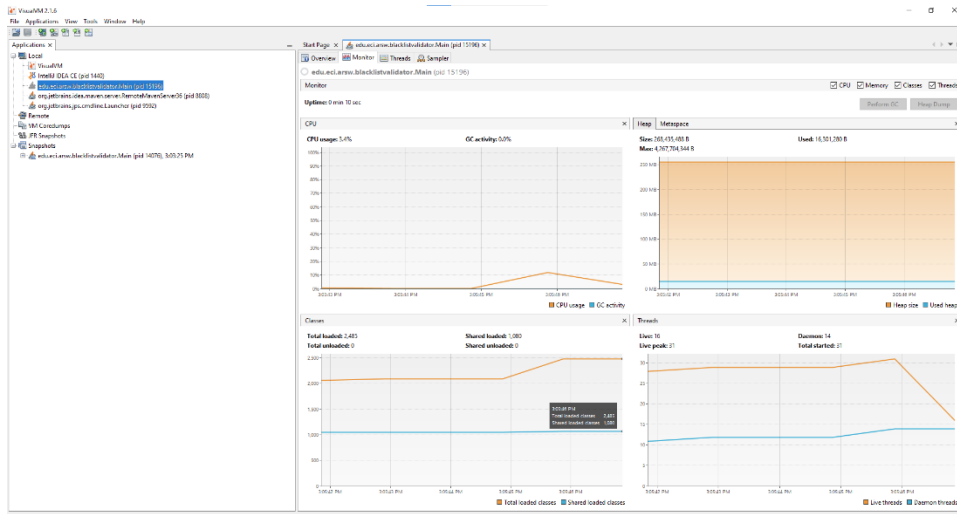
Usando un computador del laboratorio de sistemas con las siguientes características de procesador:



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



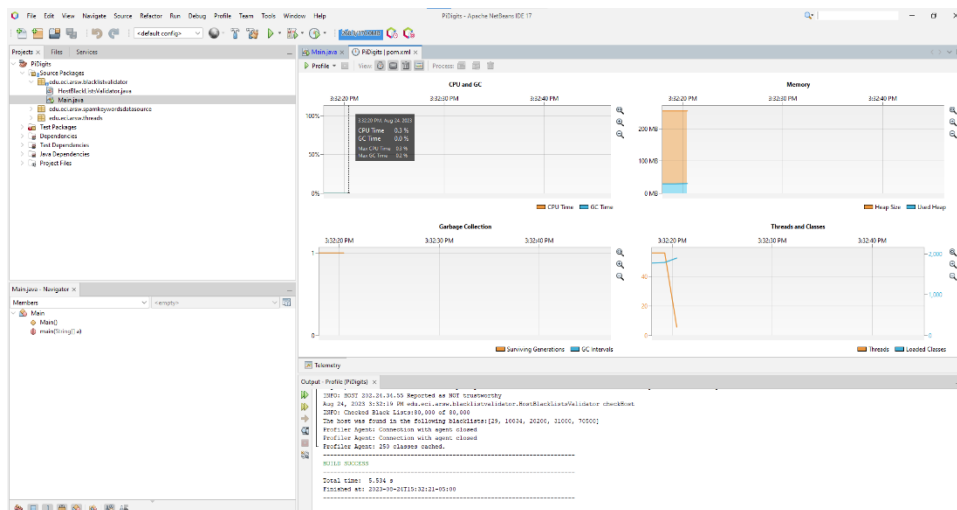
Con el Doble de hilos es decir 16 hilos



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

