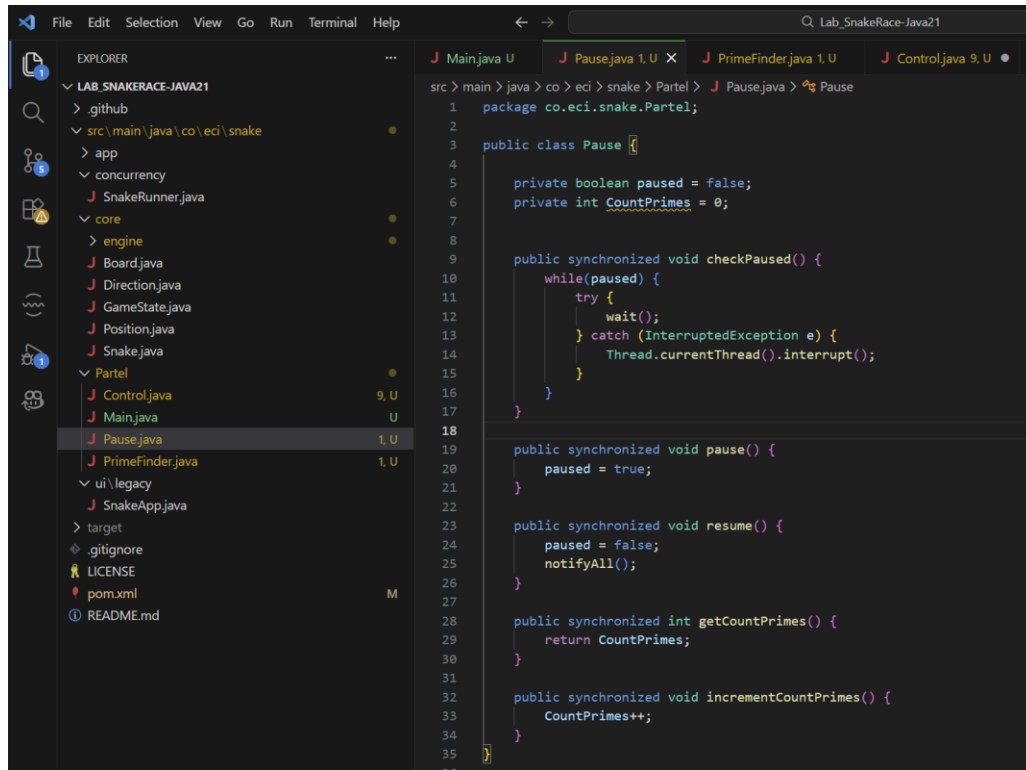


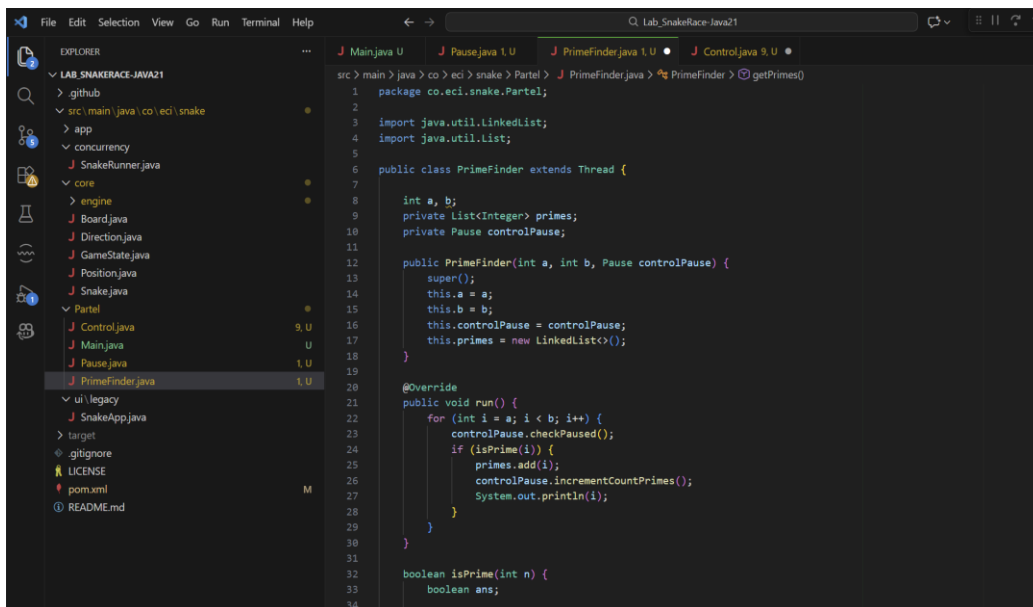
Parte I

Punto 2 y 3



The screenshot shows an IDE with the Explorer view on the left displaying the project structure for 'LAB_SNAKERACE-JAVA21'. The 'Partel' package is expanded, showing files like 'Control.java', 'Main.java', 'Pause.java', 'PrimeFinder.java', and 'SnakeApp.java'. The 'Pause.java' file is selected and its code is visible in the editor. The code defines a 'Pause' class with a 'paused' boolean, a 'CountPrimes' integer, and methods for checking, pausing, resuming, and incrementing the prime count.

```
src > main > java > co > eci > snake > Partel > J Pause.java > Pause
1 package co.eci.snake.Partel;
2
3 public class Pause {
4
5     private boolean paused = false;
6     private int CountPrimes = 0;
7
8
9
10    public synchronized void checkPaused() {
11        while(paused) {
12            try {
13                wait();
14            } catch (InterruptedException e) {
15                Thread.currentThread().interrupt();
16            }
17        }
18    }
19
20    public synchronized void pause() {
21        paused = true;
22    }
23
24    public synchronized void resume() {
25        paused = false;
26        notifyAll();
27    }
28
29    public synchronized int getCountPrimes() {
30        return CountPrimes;
31    }
32
33    public synchronized void incrementCountPrimes() {
34        CountPrimes++;
35    }
36 }
```



The screenshot shows the same IDE with the 'PrimeFinder.java' file selected in the Explorer view. The code defines a 'PrimeFinder' class that extends 'Thread'. It has a 'run()' method that checks for primes within a given range, using the 'Pause' class to control the execution. It also has an 'isPrime()' method.

```
src > main > java > co > eci > snake > Partel > J PrimeFinder.java > PrimeFinder > PrimeFinder()
1 package co.eci.snake.Partel;
2
3 import java.util.LinkedList;
4 import java.util.List;
5
6 public class PrimeFinder extends Thread {
7
8     int a, b;
9     private List<Integer> primes;
10    private Pause controlPause;
11
12    public PrimeFinder(int a, int b, Pause controlPause) {
13        super();
14        this.a = a;
15        this.b = b;
16        this.controlPause = controlPause;
17        this.primes = new LinkedList<>();
18    }
19
20    @Override
21    public void run() {
22        for (int i = a; i < b; i++) {
23            controlPause.checkPaused();
24            if (isPrime(i)) {
25                primes.add(i);
26                controlPause.incrementCountPrimes();
27                System.out.println(i);
28            }
29        }
30    }
31
32    boolean isPrime(int n) {
33        boolean ans;
34    }
```

The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes files like Position.java, Snake.java, Partel, Control.java, Main.java, Pause.java, PrimeFinder.java, ui\legacy, SnakeApp.java, target, .gitignore, LICENSE, pom.xml, and README.md. The code editor shows the implementation of the `run()` method in a class, which is a multi-threaded prime number finder. It uses a `for` loop to create `NTHREADS` threads, each of which calls `pft[i].start()`. A `BufferedReader` is used to read input from the user. The code includes a `try` block with a `while` loop that sleeps for 10 milliseconds, pauses the threads, prints the current state, and then resumes them. A `catch` block handles any exceptions that occur.

```
33 public void run() {
34
35     for (int i = 0; i < NTHREADS; i++) {
36         pft[i].start();
37     }
38
39     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
40
41     try {
42         while (true) {
43             Thread.sleep(TMILISECONDS);
44             control.pause();
45
46             System.out.println(x: "\nPaused");
47             System.out.println("Found primes: " + control.getCountPrimes());
48             System.out.println(x: "Press ENTER to continue...");
49             br.readLine();
50             control.resume();
51         }
52     } catch (Exception e) {
53         e.printStackTrace();
54     }
55 }
56
57
58
59
```

```
822007
822011
822013
822037
822049
20049319
822067
822079
822113
822131
822139
822161
822163
822167
822169
822191
822197
10088627

Paused
20049329
Found primes: 74074
Press ENTER to continue...
```

Punto 4

Para coordinar la ejecución de los hilos se utilizó un único objeto compartido que actúa como monitor. Este objeto es usado tanto por los hilos trabajadores como por el hilo controlador, y su candado (lock) se obtiene mediante métodos sincronizados (synchronized).

La pausa de los hilos se controla con una variable booleana que indica si el sistema está en estado de pausa. Cuando esta variable está activa, los hilos trabajadores se detienen de forma cooperativa llamando a `wait()`, lo que les permite quedar bloqueados sin consumir CPU. Cuando el sistema se reanuda, el hilo controlador cambia el estado de la condición y despierta a todos los hilos con `notifyAll()`.

Para evitar problemas de *lost wakeups*, la llamada a `wait()` se realiza dentro de un ciclo `while` que vuelve a verificar la condición cada vez que el hilo se despierta. De esta forma se garantiza que ningún hilo continúe ejecutándose si la pausa aún está activa, incluso en caso de despertares, inesperados.