

**Universidad Autonoma de Nuevo León.
Facultad de Ciencias Físico Matemáticas.**

Práctica #10 – Programación concurrente.

Laboratorio de Programación Orientada a Objetos.

Alumno: Jeremy Uriel Rossell Segura

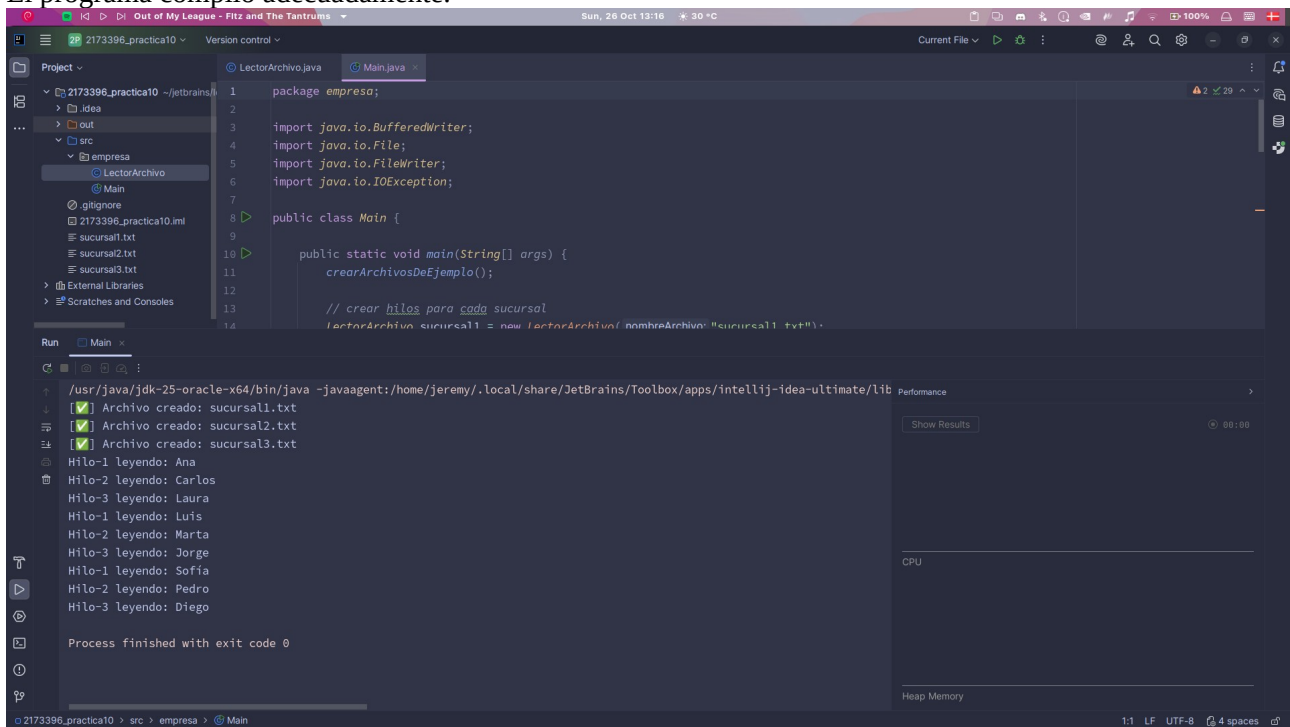
Matrícula: 2173396

Materia: Laboratorio de Programación Orientada a Objetos.

Docente: Jorge Alberto Islas Pineda

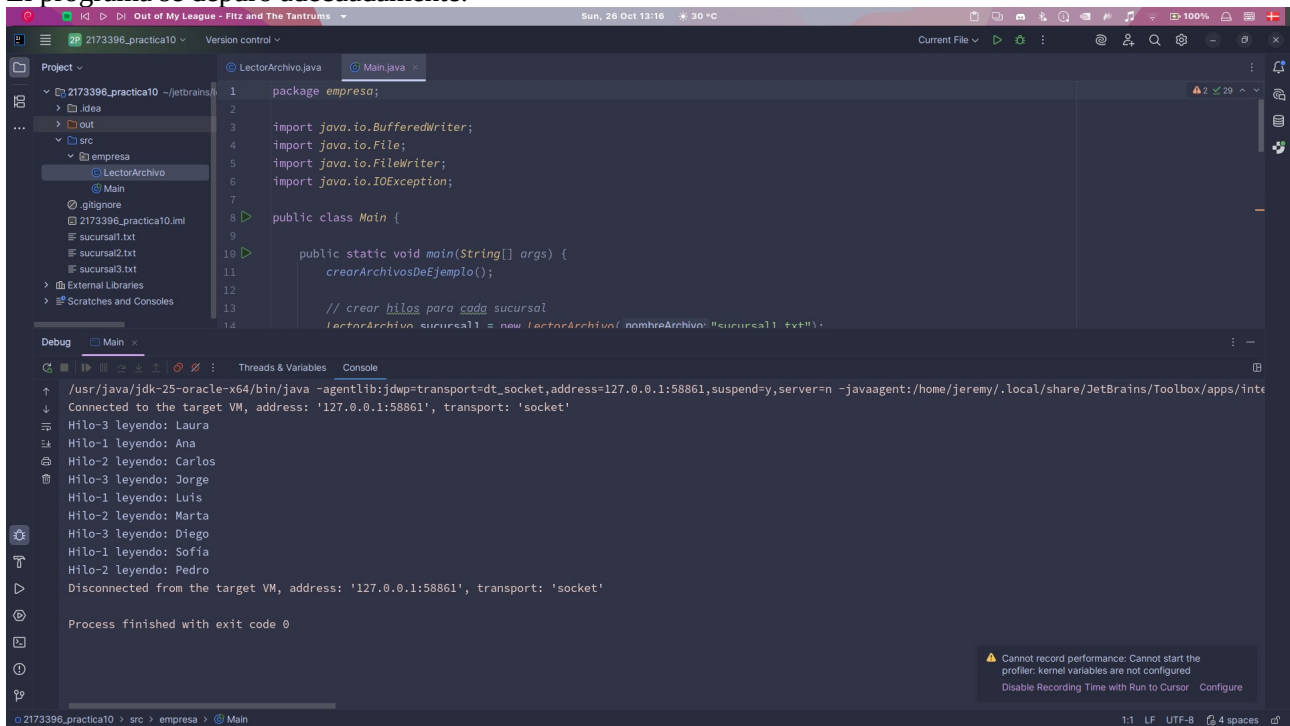
Compilación

El programa compiló adecuadamente.



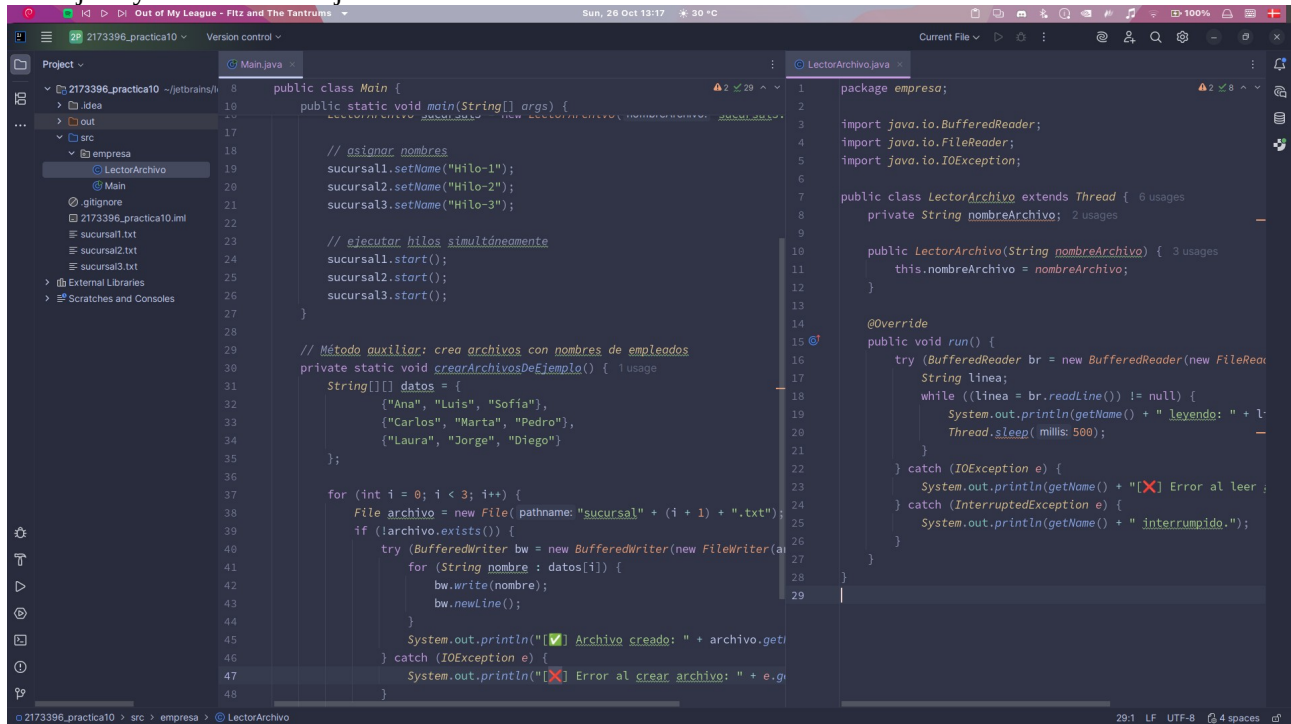
Depuración

El programa se depuró adecuadamente.



Código

Main.java y LectorArchivo.java



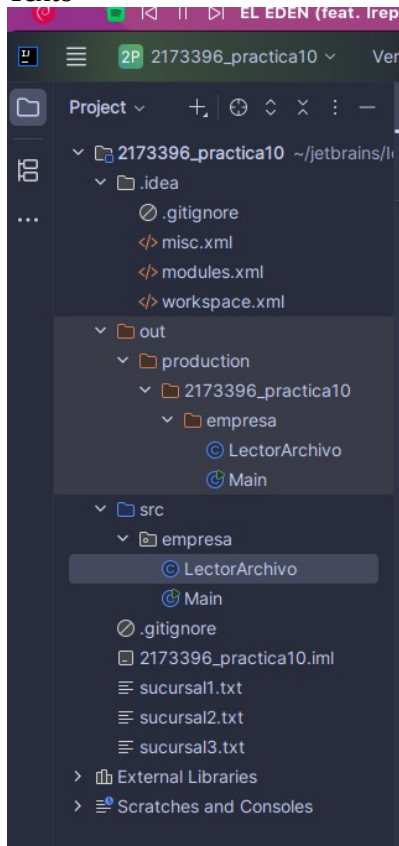
The screenshot shows an IDE with two files open. The left pane shows the project structure with folders like 'src' and 'out', and files like 'Main.java' and 'LectorArchivo.java'. The right pane shows the code for both files.

```
public class Main {  
    public static void main(String[] args) {  
        // asignar nombres  
        sucursal1.setName("Hilo-1");  
        sucursal2.setName("Hilo-2");  
        sucursal3.setName("Hilo-3");  
  
        // ejecutar hilos simultáneamente  
        sucursal1.start();  
        sucursal2.start();  
        sucursal3.start();  
    }  
  
    // Método auxiliar: crea archivos con nombres de empleados  
    private static void crearArchivosDeEjemplo() {  
        String[] datos = {  
            {"Ana", "Luis", "Sofia"},  
            {"Carlos", "Marta", "Pedro"},  
            {"Laura", "Jorge", "Diego"}  
        };  
  
        for (int i = 0; i < 3; i++) {  
            File archivo = new File(pathname: "sucursal" + (i + 1) + ".txt");  
            if (archivo.exists()) {  
                try (BufferedWriter bw = new BufferedWriter(new FileWriter(archivo))) {  
                    for (String nombre : datos[i]) {  
                        bw.write(nombre);  
                        bw.newLine();  
                    }  
                }  
                System.out.println("✅ Archivo creado: " + archivo.getName());  
            } catch (IOException e) {  
                System.out.println("❌ Error al crear archivo: " + e.getMessage());  
            }  
        }  
    }  
}
```

```
package empresa;  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
  
public class LectorArchivo extends Thread {  
    private String nombreArchivo;  
  
    public LectorArchivo(String nombreArchivo) {  
        this.nombreArchivo = nombreArchivo;  
    }  
  
    @Override  
    public void run() {  
        try (BufferedReader br = new BufferedReader(new FileReader(nombreArchivo))) {  
            String linea;  
            while ((linea = br.readLine()) != null) {  
                System.out.println(getName() + " leyendo: " + linea);  
                Thread.sleep(500);  
            }  
        } catch (IOException e) {  
            System.out.println(getName() + "❌ Error al leer: " + e.getMessage());  
        } catch (InterruptedException e) {  
            System.out.println(getName() + "interrumpido.");  
        }  
    }  
}
```

Archivos

Texto



Preguntas

1. ¿Qué sucede si se ejecutan los tres hilos sin el `Thread.sleep(500)`?

Todos los hilos se podría decir que “competirán” por tiempo de CPU y leerán los archivos demasiado rápido, por lo que la salida aparecerá casi al mismo tiempo, por lo que puede verse desordenada o agrupar varias líneas sin pausas

2. ¿Por qué el orden de lectura varía en cada ejecución?

Porque los hilos son administrados por el planificador del sistema operativo, que decide en qué orden se ejecutan los fragmentos de código, a esto en ciencias computacionales se le conoce como time slicing, que hace que el sistema pueda dar prioridad a distintos hilos en cada ejecución

3. ¿Qué pasaría si todos los hilos intentaran escribir en el mismo archivo?

Se producirían race conditions, el contenido podría mezclarse o peor, sobrescribirse, ya que varios hilos escribirían simultáneamente en la misma posición del archivo

4. ¿Cómo podrías sincronizar el acceso al archivo compartido?

Usando la keyword “synchronized”, o utilizando clases seguras para hilos como `ReentrantLock`, o una `BlockingQueue`

5. ¿Qué clase se usa para crear o verificar un archivo en el sistema?

La clase `java.io.File`, la cual nos deja verificar existencia con `exists()`, crear archivo con `createNewFile()`, y eliminar con `delete()`

6. ¿Qué diferencia hay entre `start()` y `run()` en un hilo?

`start()` crea un nuevo hilo de ejecución independiente y luego llama internamente a `run()`, pero si llamas directamente a `run()`, el código se ejecuta en el mismo hilo actual, sin concurrencia real, esa es su diferencia

7. ¿Qué pasa si no cierras correctamente un `BufferedReader` o `BufferedWriter`?

El archivo puede quedar totalmente bloqueado, los datos no se guardan correctamente, o se pierden recursos del sistema mediante fugas de memoria, por eso usamos `try-with-resources`

8. ¿Cómo podrías modificar el programa para que cada hilo escriba en un nuevo archivo en lugar de leer?

Cambiando `BufferedReader` por un `BufferedWriter` dentro del `run()` y escribir:

```
``java
try (BufferedWriter bw = new BufferedWriter(new FileWriter(nombreDelArchivo))) {
    bw.write("texto generado por " + getName());
}
````
```

### 9. ¿Qué ventajas tiene usar varios hilos en lugar de leer los tres archivos uno por uno?

Mayor eficiencia ya que pueden leerse simultáneamente si hay varios núcleos, y aprovechas el poder del CPU junto a su buffer y clock speed, también no bloquea la ejecución, mientras un hilo espera la entrada/salida otro puede avanzar; y sobre todo escalabilidad, ya que es más fácil procesar grandes volúmenes de archivos y datos

### 10. ¿Podrías agregar un hilo adicional que, al final, cuente cuántas líneas totales se leyeron entre los tres archivos?

Sí, podemos crear un hilo extra al que podemos llamar `ContadorLineas` que lea los tres archivos y sume el total, no es difícil

```
``java
class ContadorLineas extends Thread {
 @Override
 public void run() {
 int total = 0;
```

```

 for (int i = 1; i <= 3; i++) {
 try (BufferedReader br = new BufferedReader(new FileReader("sucursal" + i + ".txt"))) {
 while (br.readLine() != null) total++;
 } catch (IOException e) {
 e.printStackTrace();
 }
 }
 System.out.println("Total de líneas leídas: " + total);
}
}
...

```

y luego en el main():

```

```java
ContadorLineas contador = new ContadorLineas();
contador.start();
...

```