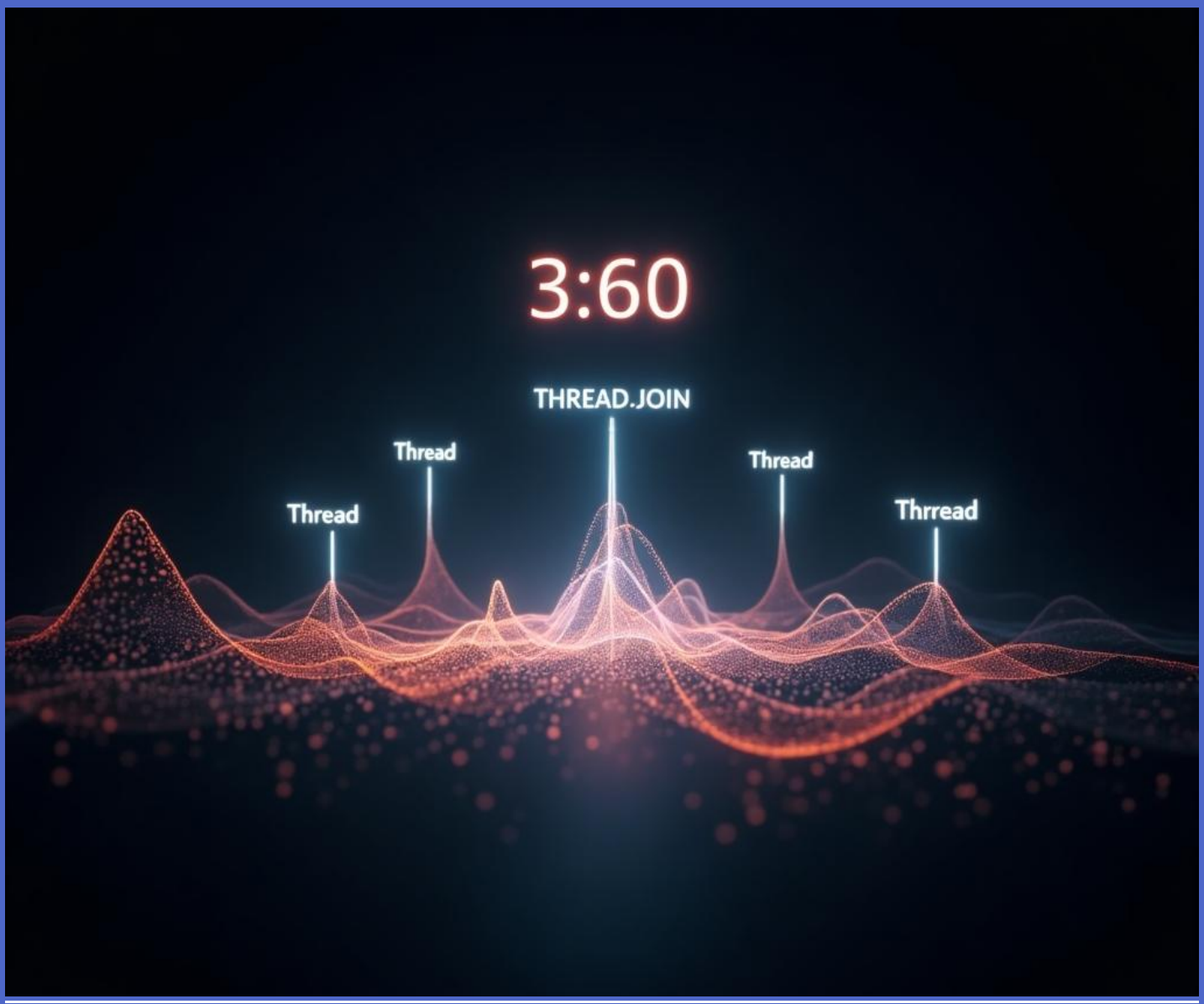
PROGRAMACION DE SERVICIOS

TAREA I UD2



ALUMNO CESUR 25/26

Alejandro Muñoz de la Sierra

PROFESOR

Santiago Martin-palomo Garcia

I N T R O D U C C I O N

Al enfrentarnos a este caso práctico, la sensación inicial fue de incertidumbre. Aunque ya habíamos experimentado con hilos en ejercicios previos, sentíamos que nos adentrábamos en un terreno desconocido. Sin embargo, decidimos avanzar con cautela, procurando comprender cada decisión y analizando las implicaciones de cada línea de código.

El Ayuntamiento nos propuso un desafío aparentemente sencillo: determinar la temperatura más alta registrada en la última década. Sin embargo, al constatar que la lista contenía 3.650 valores, comprendimos que el verdadero objetivo era practicar la programación concurrente, permitiendo que varios hilos trabajaran en paralelo sobre diferentes secciones de un mismo conjunto de datos. Desde mi punto de vista, esto era más un ejercicio de optimización que una necesidad real.

Antes de empezar a programar, nos detuvimos a reflexionar:

¿Cómo se divide un trabajo extenso en procesos más pequeños?

¿Cómo se crean y coordinan estos hilos?

¿De qué forma comunican sus resultados?

¿Qué riesgos implica la concurrencia?

Decidimos empezar por lo esencial: simular los datos, dividir la tarea y organizar la cooperación entre hilos, construyendo el programa gradualmente.
Personalmente, creo que esta aproximación fue la más acertada.

0 1
SIMULACIÓN DE LA LISTA DE TEMPERATURAS

Para recrear la situación, generamos un array de 3.650 enteros, uno por cada día de los últimos diez años. Utilizamos la clase Random para crear valores dentro de un rango amplio: de –30°C a 50°C. La verdad es que este rango nos pareció adecuado para simular las temperaturas reales.

Durante esta fase consideramos varios aspectos:

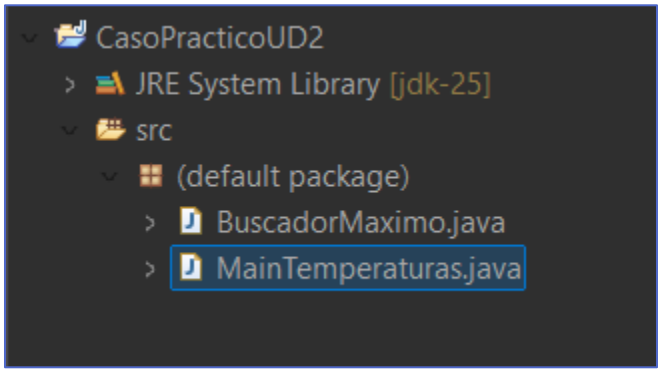
El rango abarca condiciones extremas, lo que nos permite verificar que el código funciona con números muy diferentes, incluso negativos.

La magnitud del array justifica la división en hilos: aunque un solo hilo podría procesarlo rápidamente, el objetivo era practicar la paralelización.

Queríamos que los valores generados fueran verdaderamente aleatorios, por lo que probamos varias ejecuciones y comprobamos la variabilidad. No queríamos patrones predecibles.

El propósito de este paso era familiarizarnos con los datos. Conocer el comportamiento de los números nos ayuda mucho al analizar los resultados finales. Sin embargo, confieso que al principio subestimé la importancia de esta fase.

Estructura y Entorno del ejercicio



02
DIVISIÓN DEL ARRAY: FUTUROS HILOS

Después de generar el array, lo dividimos en partes. Dedicamos un tiempo considerable a determinar la forma más práctica de hacerlo. ¿La mejor manera? Probablemente no existe una única respuesta.

Optamos por una solución sencilla: tres partes aproximadamente iguales. Si bien podríamos haber elegido cuatro o cinco hilos, tres nos pareció más manejable para este ejercicio, aunque reconozco que esto es una cuestión de preferencia personal.

¿Por qué tres? Es un número fácil de manejar.

Permite observar con claridad la cooperación entre hilos.

Simplifica la depuración si surge algún error.

Representa un buen equilibrio entre simplicidad y paralelismo.

La división se realizó mediante índices:

Hilo 1: del 0 al primer tercio.
Hilo 2: del primer tercio al segundo tercio.
Hilo 3: del segundo tercio al final.

Cada hilo solo lee el array, no lo modifica, lo que reduce los riesgos de concurrencia y simplifica la lógica. Esta decisión nos brindó tranquilidad con respecto a posibles errores.

0 3

E X P L I C A C I O N D E L A S C L A S E S Y E L C O D I G O

3.1. Clase BuscadorMaximo: Hilos en Acción

```java
public class BuscadorMaximo implements Runnable {

    private int[] datos;      // Array completo de temperaturas
    private int inicio;       // Índice de inicio del segmento
    private int fin;          // Índice de fin del segmento
    private int maxLocal;     // Resultado: máximo local encontrado

    // Constructor que recibe el array y los índices de segmento
    public BuscadorMaximo(int[] datos, int inicio, int fin) {
        this.datos = datos;
        this.inicio = inicio;
        this.fin = fin;
        this.maxLocal = Integer.MIN_VALUE; // Inicializamos con el valor más bajo posible
    }

    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName() +
            " buscando entre índices " + inicio + " y " + fin);

        // Recorremos solo la sección asignada a este hilo
        for (int i = inicio; i < fin; i++) {
            if (datos[i] > maxLocal) {
                maxLocal = datos[i];
            }
        }

        System.out.println(Thread.currentThread().getName() +
            " ha terminado. Máximo local = " + maxLocal);
    }

    // Método para obtener el máximo local desde el hilo principal
    public int getMaxLocal() {
        return maxLocal;
    }
}

```

Para esta tarea, optamos por la clase **BuscadorMaximo**, que implementa la interfaz **Runnable**. ¿Por qué Runnable? Bueno, nos brindaba una flexibilidad genial. Nos permitía separar la tarea en sí, que era buscar el valor máximo, de la ejecución del hilo. Intentar hacer esto extendiendo directamente **Thread**, la verdad, no habría sido tan sencillo.

```java
public class BuscadorMaximo implements Runnable {

    private int[] datos;      // Array completo de temperaturas
    private int inicio;       // Índice de inicio del segmento
    private int fin;          // Índice de fin del segmento
    private int maxLocal;     // Resultado: máximo local encontrado

```

Dentro de esta clase, declaramos algunos atributos que resultaron ser fundamentales:

datos: Aquí almacenamos el array completito de temperaturas que íbamos a analizar.
inicio y **fin**: Estos delimitaban la sección del array que cada hilo debía procesar, asegurando que no se pisasen el trabajo unos a otros. Esto fue crucial para el paralelismo.
maxLocal: En esta variable, cada hilo guardaba el máximo que encontraba dentro de su sección. La inicializamos con **Integer.MIN_VALUE** para asegurarnos de que cualquier temperatura real pudiera sobrescribirla desde el principio.

```java
    // Constructor que recibe el array y los índices de segmento
    public BuscadorMaximo(int[] datos, int inicio, int fin) {
        this.datos = datos;
        this.inicio = inicio;
        this.fin = fin;
        this.maxLocal = Integer.MIN_VALUE; // Inicializamos con el valor más bajo posible
    }
```

El constructor de la clase recibía el array y el rango asignado al hilo, y asignaba esos valores a los atributos. Este simple paso nos hizo reflexionar bastante sobre la importancia de que cada hilo tuviera su propia "zona de trabajo", aislada del resto.

El quid de la cuestión era el método **run()**. Ahí dentro definimos la lógica de cada hilo.

```
16      @Override
17      public void run() {
18          System.out.println(Thread.currentThread().getName() +
19              " buscando entre índices " + inicio + " y " + fin);
20
21          // Recorremos solo la sección asignada a este hilo
22          for (int i = inicio; i < fin; i++) {
23              if (datos[i] > maxLocal) {
24                  maxLocal = datos[i];
25              }
26          }
27
28          System.out.println(Thread.currentThread().getName() +
29              " ha terminado. Máximo local = " + maxLocal);
30      }
31
```

Comenzábamos imprimiendo un mensaje con el nombre del hilo y los índices del array que iba a analizar. Esto nos ayudó un montón a visualizar la ejecución concurrente, a ver cómo los hilos trabajaban al mismo tiempo.
Luego, iterábamos sobre la porción del array asignada al hilo. Cada vez que encontrábamos un valor mayor que **maxLocal**, lo actualizábamos. Era la esencia del algoritmo: súper sencillo, pero perfecto para practicar paralelismo.
Al finalizar, imprimíamos el máximo local que había encontrado el hilo. Esto nos confirmaba que cada hilo estaba trabajando de forma independiente, sin interferencias.

```
32      // Método para obtener el máximo local desde el hilo principal
33      public int getMaxLocal() {
34          return maxLocal;
35      }
36  }
37
```

Finalmente, agregamos un método **getMaxLocal**() para que el hilo principal pudiera acceder a los resultados de forma segura, una vez que los hilos secundarios hubieran terminado.

4.2. Clase MainTemperaturas: Simulando y Desatando los Hilos

```java
import java.util.Random;

public class MainTemperaturas {

    public static void main(String[] args) {

        // 1. Simulación de temperaturas
        int tamaño = 3650;
        int[] temperaturas = new int[tamaño];
        Random rnd = new Random();

        for (int i = 0; i < tamaño; i++) {
            temperaturas[i] = rnd.nextInt(81) - 30; // Generamos valores entre -30 y 50
        }

        System.out.println("Array de temperaturas generado correctamente.\n");

        // 2. Configuración de 3 hilos
        int numHilos = 3;
        Thread[] hilos = new Thread[numHilos];
        BuscadorMaximo[] tareas = new BuscadorMaximo[numHilos];

        int tamañoSegmento = tamaño / numHilos;

        // 3. Creación y lanzamiento de hilos
        for (int i = 0; i < numHilos; i++) {
            int inicio = i * tamañoSegmento;
            int fin = (i == numHilos - 1) ? tamaño : inicio + tamañoSegmento;

            tareas[i] = new BuscadorMaximo(temperaturas, inicio, fin);
            hilos[i] = new Thread(tareas[i], "Hilo-" + i);
            hilos[i].start();
        }

        // 4. Espera a que todos los hilos terminen
        for (Thread hilo : hilos) {
            try {
                hilo.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        // 5. Cálculo del máximo global
        int maximoGlobal = Integer.MIN_VALUE;

        for (BuscadorMaximo tarea : tareas) {
            if (tarea.getMaxLocal() > maximoGlobal) {
                maximoGlobal = tarea.getMaxLocal();
            }
        }

        System.out.println("\n==============================");
        System.out.println("Temperatura máxima en 10 años: " + maximoGlobal + "ºC");
        System.out.println("==============================\n");
    }
}
```

Para empezar, simulamos los datos que íbamos a usar:

Creamos un array enorme, de 3.650 posiciones. Un entero por cada día de diez años.
Usamos la clase **Random**, previamente importada, para generar valores aleatorios entre –30 y 50 grados. Para ajustar el rango, hacíamos **rnd.nextInt(81) - 30**.
Imprimimos un mensaje para confirmar que la simulación se había completado.

```java
import java.util.Random;

public class MainTemperaturas {

    public static void main(String[] args) {

        // 1. Simulación de temperaturas
        int tamaño = 3650;
        int[] temperaturas = new int[tamaño];
        Random rnd = new Random();

        for (int i = 0; i < tamaño; i++) {
            temperaturas[i] = rnd.nextInt(81) - 30; // Generamos valores entre -30 y 50
        }

        System.out.println("Array de temperaturas generado correctamente.\n");

```

Queríamos separar claramente la fase de generación de datos de la ejecución de los hilos.

Luego, definimos los tres hilos que se encargarían de dividir el trabajo:

```java
        // 2. Configuración de 3 hilos
        int numHilos = 3;
        Thread[] hilos = new Thread[numHilos];
        BuscadorMaximo[] tareas = new BuscadorMaximo[numHilos];

        int tamañoSegmento = tamaño / numHilos;

```

Calculamos cuántos elementos le correspondían a cada hilo (tamañoSegmento).

```
25          // 3. Creación y lanzamiento de hilos
26          for (int i = 0; i < numHilos; i++) {
27              int inicio = i * tamañoSegmento;
28              int fin = (i == numHilos - 1) ? tamaño : inicio + tamañoSegmento;
29
30              tareas[i] = new BuscadorMaximo(temperaturas, inicio, fin);
31              hilos[i] = new Thread(tareas[i], "Hilo-" + i);
32              hilos[i].start();
33          }
```

Para cada hilo, determinamos el rango de índices. Usamos un operador ternario (i == **numHilos** - 1) ? **tamaño** : **inicio** + **tamañoSegmento** para asegurar que el último hilo llegase hasta el final del array, sin dejar ningún valor sin analizar.

Creamos la instancia **BuscadorMaximo**, podemos hacerlo sin importar la clase, porque está en el mismo paquete default que la clase **main**, después creamos el **Thread** correspondiente, le asignamos un nombre y lo lanzamos con .**start**().

Esta parte fue súper instructiva. Experimentamos con la creación de hilos y entendimos cómo cada uno ejecutaba su tarea en paralelo sin bloquear al hilo principal.

Después, usamos el método **join**() para esperar a que todos los hilos terminaran antes de seguir adelante. Aquí, aprendimos algunas cosas importantes:

```
35          // 4. Espera a que todos los hilos terminen
36          for (Thread hilo : hilos) {
37              try {
38                  hilo.join();
39              } catch (InterruptedException e) {
40                  e.printStackTrace();
41              }
42          }
```

No siempre es necesario usar synchronized. En este caso, los hilos solo estaban leyendo datos, no modificándolos.
join() garantiza que los resultados estén listos y disponibles antes de integrarlos.
Siempre es recomendable manejar **InterruptedException**, aunque sea poco probable que ocurra en este ejercicio. Nunca está de más ser precavidos.

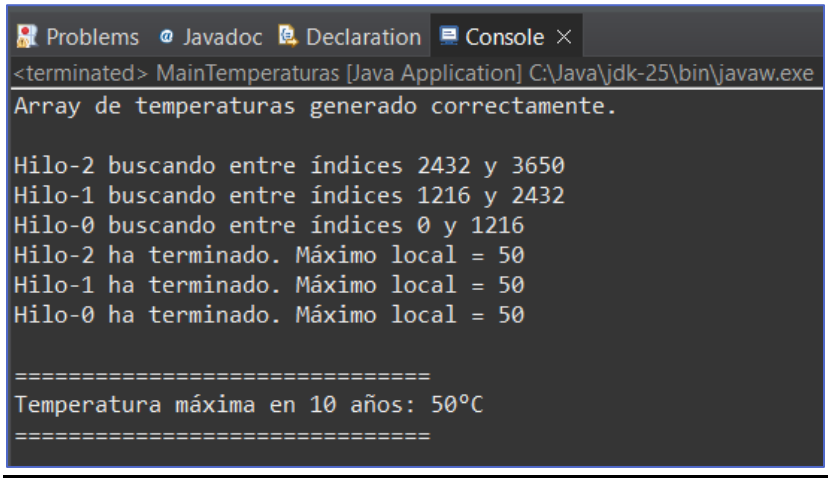
Integrando los Resultados: En Busca del Máximo Global

Una vez que todos los hilos habían finalizado, recopilamos los máximos locales y calculamos el máximo absoluto:

```
44          // 5. Cálculo del máximo global
45          int maximoGlobal = Integer.MIN_VALUE;
46
47          for (BuscadorMaximo tarea : tareas) {
48              if (tarea.getMaxLocal() > maximoGlobal) {
49                  maximoGlobal = tarea.getMaxLocal();
50              }
51          }
52
53          System.out.println("\n==============================");
54          System.out.println("Temperatura máxima en 10 años: " + maximoGlobal + "°C");
55          System.out.println("==============================\n");
56      }
57  }
```

Inicializamos **maximoGlobal** con Integer.MIN_VALUE. Así, nos asegurábamos de que cualquier valor del array fuera mayor que el valor inicial.
Iteramos sobre cada resultado de **getMaxLocal**() y actualizamos **maximoGlobal** si encontrábamos un valor mayor.
Finalmente, imprimimos el resultado de forma clara, utilizando líneas de separación visual (===) para que la salida en consola fuese más legible, sobre todo con múltiples hilos imprimiendo al mismo tiempo.

El resultado final nos demostró que la simulación de datos aleatorios había funcionado correctamente y que cada hilo había realizado su trabajo de forma independiente. Fue muy gratificante comprobar cómo la programación concurrente, que al principio parecía compleja, se vuelve lógica y controlable cuando se organiza paso a paso.



0 4
C O N C L U S I O N E S

Este pequeño ejercicio nos enseñó más de lo que esperábamos inicialmente:

Dividir para optimizar: Aprendimos a separar las tareas de una manera lógica y equilibrada. Algo que creo que es súper útil.
Hilos, la clase Runnable, start() y join() ya no son desconocidas,concurrencia segura.
Entendimos la diferencia entre los hilos que solo leen datos y aquellos escenarios donde podría haber conflictos.
Lecciones en lo simple: Incluso buscar un máximo puede enseñarnos un montón sobre coordinación, planificación y paralelización. En mi opinión, subestimamos los problemas sencillos.

En definitiva, un problema que parecía fácil se transformó en un ejercicio poderoso para asimilar conceptos de concurrencia y ganar confianza al manipular hilos.

0 5
R E F E R E N C I A S

https://docs.oracle.com/javase/tutorial/essential/concurrency/

https://docs.oracle.com/javase/tutorial/essential/concurrency/thread.html

https://jenkov.com/tutorials/java-concurrency/index.html

https://www.geeksforgeeks.org/multithreading-in-java/

https://www.geeksforgeeks.org/java/java-util-concurrent-package

https://www.tatvasoft.com/outsourcing/2025/08/java-concurrency.html

https://www.baeldung.com/java-concurrency

https://www.youtube.com/watch?v=SztE5W41on4

https://www.youtube.com/playlist?list=PLL8woMHwr36EDxjUoCzboZjedsnhLP1j4

https://www.youtube.com/playlist?list=PLmCsXDGbJHdjmXF-w2mWaXa5p0dxHE3nW