

## Jose Fuentes - 201631094

### Taller 1b – Manejo de Threads

El propósito de este taller es entender la forma como se manejan los threads para implementar aplicaciones concurrentes en Java, e identificar la necesidad sincronización para controlar el acceso concurrente a variables compartidas. El taller tiene dos partes. En la primera parte se va a incrementar un contador un número determinado de veces utilizando dos programas: monothread y multithread. En la segunda parte se seleccionará el mayor de los elementos de una matriz de enteros iniciada al azar.

#### Parte 1: Incremento de un contador

##### Ejemplo 1: Aplicación monothread para el incremento de un contador

El ejemplo a continuación muestra cómo manipular un contador en una aplicación monothread. El ejemplo consiste en llamar 1000 veces un método que incrementa 10000 veces un contador. Este programa es realizado utilizando únicamente el thread principal de la aplicación.

```
1 public class ContadorMonoThread{
2     private int contador = 0;
3
4     public void incrementar() {
5         for (int i = 0; i < 10000; i++) {
6             contador++;
7         }
8     }
9
10    public int getContador () {
11        return contador;
12    }
13
14    public static void main(String[] args) {
15        ContadorMonoThread c = new ContadorMonoThread();
16
17        for (int i = 0; i < 1000; i++) {
18            c.incrementar();
19        }
20
21        System.out.println(c.getContador());
22    }
23 }
```

#### Responda:

1. ¿Al ejecutar el programa, el resultado corresponde al valor esperado?

Si

## Ejemplo 2: Aplicación multithread para el incremento de un contador

El ejemplo a continuación muestra un ejemplo de una aplicación multithread para la manipulación de un contador. El ejemplo consiste en crear 1000 threads que al ejecutarse, incrementen 10000 veces un contador.

```

1 // Esta clase extiende de la clase Thread
2 public class ContadorThreads extends Thread {
3     // Variable de la clase. Todos los objetos de esta clase ven esta variable.
4     private static int contador = 0;
5
6     // Este método se ejecuta al llamar el método start().
7     // Cada thread incrementa 10 mil veces el valor del contador.
8     public void run() {
9         for (int i = 0; i < 10000; i++) {
10             contador++;
11         }
12     }
13
14     public static void main(String[] args) {
15         // Se crea un array mil de threads
16         ContadorThreads[] t = new ContadorThreads[1000];
17
18         // Se crean e inician los mil threads del array.
19         for (int i = 0; i < t.length; i++) {
20             t[i] = new ContadorThreads();
21             t[i].start();
22         }
23
24         System.out.println(contador);
25     }
26 }
  
```

### Responda:

- ¿Al ejecutar el programa, el resultado corresponde al valor esperado? Explique.

No, porque se busca que el contador llegue a 10000 sin embargo el resultado es menor a este ya que se ejecuta en distintos Threads que buscan tener acceso a una misma variable por lo que ocurre este fenómeno

- Ejecute cinco veces el programa y escriba el resultado obtenido en cada ejecución.

Ejecución	Valor obtenido
1	9576095
2	9595644
3	9633015
4	9535583
5	9468104

4. ¿Hay acceso concurrente a alguna variable compartida? Si es así, diga en dónde.  
Si, ya que todos los threads buscan acceso a la variable contador por lo que existe concurrencia en esta variable.

## Parte 2: Elemento mayor en una matriz de enteros

### Ejemplo 3: Aplicación multithread para encontrar el elemento mayor de una matriz de enteros

El ejemplo a continuación muestra cómo utilizar threads para que de manera concurrente se pueda encontrar el mayor de los elementos de una matriz de enteros.



```
import java.util.concurrent.ThreadLocalRandom;

public class MaximoMatriz extends Thread {
    //Vamos a generar los numeros aleatorios en un intervalo amplio
    private final static int INT_MAX = 105345;

    //Dimensiones cuadradas
    private final static int DIM = 3;

    //Matriz
    private static int[][] matriz = new int[DIM][DIM];

    //Mayor global
    private static int mayor = -1;

    //Mayor local
    private int mayorFila = -1;

    //ID Thread
    private int idThread;

    //Fila a registrar
    private int fila;

    //Constructor
    public MaximoMatriz(int pIdThread, int pFila) {
        this.idThread = pIdThread;
        this.fila = pFila;
    }
}
```

```
//Generar la matriz con números aleatorios
public static void crearMatriz() {
    for (int i = 0; i < DIM; i++) {
        for(int j = 0; j < DIM; j++) {
            matriz[i][j] = ThreadLocalRandom.current().nextInt(0, INT_MAX);
        }
    }
    //Imprimir la matriz
    System.out.println("Matriz:");
    System.out.println("=====");
    imprimirMatriz();
}

//Imprimir la matriz en consola
private static void imprimirMatriz() {
    for (int i = 0; i < DIM; i++) {
        for (int j = 0; j < DIM; j++) {
            System.out.print(matriz[i][j] + "\t");
        }
        System.out.println();
    }
}
```

```

@Override
public void run() {
    for (int j = 0; j < DIM; j++) {
        if (this.mayorFila < matriz[this.fila][j]) {
            this.mayorFila = matriz[this.fila][j];
        }
    }
    if (this.mayorFila > mayor) {
        try {
            Thread.sleep(250);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        mayor = this.mayorFila;
        String warn = String.format(
            "===== Nuevo maximo encontrado ===== \n " +
            "ID Thread: %d - Maximo local actual: %d - Maximo global: %d \n" +
            "\n",
            this.idThread,
            mayor,
            this.mayorFila
        );
        System.out.println(warn);
    }
    //Resultados
    String msg = String.format("ID Thread: %d - Maximo Local: %d - Maximo Global: %d",
        this.idThread,
        this.mayorFila,
        mayor);
    System.out.println(msg);
}

//Main
public static void main(String[] args) {
    System.out.println("Busqueda concurrente por una matriz");

    //Iniciar la matriz
    MaximoMatriz.crearMatriz();
    System.out.println();
    System.out.println("Iniciando la busqueda por la matriz \n");

    //Iniciar busqueda
    MaximoMatriz[] bThreads = new MaximoMatriz[DIM];
    for (int i = 0; i < DIM; i++) {
        bThreads[i] = new MaximoMatriz(i, i);
        bThreads[i].start();
    }
}
}
```

### Responda:

1. Ejecute cinco veces el programa y escriba el resultado obtenido en cada ejecución.

Ejecución	Valor obtenido	Valor esperado
1	70358	102398
2	102810	102810
3	71963	86715
4	53329	86404
5	76614	94795

2. ¿Hay acceso concurrente a alguna variable compartida? Si es así, diga en dónde.

Si, tanto la variable mayor como mayor fila son variables en las que hay un acceso concurrente

3. ¿Puede obtener alguna conclusión?

Los monothreads son mas exactos a la hora de calcular un valor puesto que no sufren del acceso concurrente a una variable puesto que solo se ejecutan la instrucción en un thread, sin embargo este carece de rapidez. En Cambio, los multithreads son menos exactos, sin embargo, son más rápidos en la ejecución.