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, el valor de contador va aumentando de 10.000 en cada ciclo del for en el mai, como lo hace 1.000 veces, el valor final debería ser 10.000.000 y corresponde con el resultado obtenido.

Carlos Eduardo Gómez Montoya
Jairo Emilio Bautista Mora
Geovanny Andrés González

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, incremente 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:

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

El valor esperado corresponde a 10.000.000 pero el resultado obtenido no es ese. Luego de realizar un análisis de la situación se llegó a dos conclusiones, primero que el print de contador que se hace luego del for no se hace necesariamente cuando se han terminado de ejecutar todos los threads y por eso no da el resultado esperado. Por otro lado, también se encontró que si se imprime uno por uno los valores de contador en el método run cuando acaba la impresión en consola tampoco se llega al resultado deseado (en algunos casos sí) pero si se acerca mucho más. A partir de esto se puede decir que en algunos momentos varios threads se pueden ejecutar al mismo tiempo y eso hace que contador solo aumente una unidad cuando en realidad debería haber aumentado más.

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

Ejecución	Valor obtenido
1	9660585
2	9539040

3	9586692
4	9538945
5	9467873

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

Sí, la variable compartida por los threads es la variable contador. Se accede a ella de manera concurrente en el método run.

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.

```java
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;
    }
```

```java
//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	43109	104048
2	69573	87841
3	20745	103835
4	98989	98989
5	80924	99919

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

Sí, en este caso la variable compartida es la variable global de mayor; todos los threads la pueden consultar y cambiar en concurrencia.

3. ¿Puede obtener alguna conclusión?

Como se puede ver en los resultados de las ejecuciones solamente en un se logró el resultado deseado. Lo anterior se debe a que si bien en cada fila se calculó correctamente el elemento mayor. Ocurrió que cuando se llegó a la variable **concurrente** lo que sucedió es que el valor mayor quedó con el valor de mayor fila de el último thread que realizó esta operación.

Mas explícitamente:

```
if(this.mayorFila> mayor){
    try{
        Thread.sleep(250);
    }
    catch(InterruptedException e){
        e.printStackTrace();
    }
    mayor = this.mayorFila; // mayor queda con el valor de mayor fila de el ultimo thread que hace esta operacion
```

La variable mayor está inicializada en -1, por lo que cada thread cuando es ejecutado y encuentre su mayorFila este valor será mayor que -1 por lo que los threads pasaran el if sin problema. Luego de la espera cada thread cambiará el valor global. El último thread que lo cambie será aquel que defina al valor de la variable global mayor.
Como conclusión queda claro que no es la mejor idea utilizar una variable compartida cuando estamos utilizando concurrencia mediante threads (a menos que se tenga sincronización).