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?

EL valor no es el esperado pues si se ejecuta mil veces un método que aumenta 10000 veces un contador el resultado debería sr 10000000. Sin embargo, el resultado del código es 0.

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 resultado de la implementación anterior siempre es un número diferente. En esta implementación se crean 1000 threads y cada uno va aumentando la varible contador que es compartida y la van modificando. La ejecución termina cuando el primer Thread termina el ciclo se retorana el resultado, pero como no todos los threads van al tiempo durante la ejecución por eso el resultado siempre es distinto.

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

Ejecución	Valor obtenido
1	9359193
2	9303646
3	9268976

4	9485208
5	9392409

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

Si hay acceso a la varibble compartida *contador* marcada con static.

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	33787	91007
2	77179	99366
3	101676	101676
4	48531	102839
5	33348	101720

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

Sí, hay acceso concurrente a la variable mayor, pues esta lleva el máximo local en la búsqueda.

3. ¿Puede obtener alguna conclusión?

En conclusión los Threads pueden compartir variables cuando estas sean estáticas. Sin embargo, la variable dejara de cambiar de valor cuando el primer Thread termine su instancia. Es por esto que no siempre se retorna el valor máximo siempre, sino solo cuando este es encontrado por el Thread que primero termina.