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?

Al ejecutar el programa, el resultado es 10000000; es decir, sí corresponde al valor esperado. Lo anterior se puede comprobar, ya que se ejecuta 1000 veces el método incrementar de la instancia c y esta instancia a su vez aumenta el contador 10000 cada vez que se ejecuta el método incrementar. Esto hace pensar en el siguiente producto:

$$10000 \times 1000 = 10000000$$

Por lo tanto, se puede afirmar que el resultado es el esperado.

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.

Al ejecutar el programa la primera vez el resultado NO corresponde al esperado. Sin embargo, si se ejecuta una vez más el resultado varía con respecto al anterior. Esto nos hace pensar sobre una falta de control en la concurrencia dentro del código. En otras palabras, lo anterior se presenta por una falta de sincronización y control en relación a la variable static del programa. Además, esto se presenta por una carencia de monitores y, por ende, una falta de manejo apropiado en las secciones críticas. Es importante recordar que los monitores son construccuiones de un lenguaje de programación que ofrecen sincronización a alto nivel.

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

Ejecución	Valor obtenido
1	9852040
2	9790439
3	8076312
4	8076312
5	8076312

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

Sí, existe un acceso concurrente a una variable compartida. Esta variable es "contador". Además, vale recalcar que dicha variable tiene un manejo de concurrencia incorrecto. Lo anterior permite resultados inconsisentes al momento de ejecutar el programa. Esta concurrencia ocurre en el método run.

```
// Este método se ejecuta al llamar el método start().
// Cada thread incrementa 10 mil veces el valor del contador.
public void run() {
    for (int i = 0; i < 10000; i++) {
        contador++;
    }
}
```

Al momento en que el primer hilo entra en el for modifica el i inicial. Por ende, el siguiente hilo o proceso que se ejecute no va a iniciar en un i igual a cero, sino que iniciará justo en el i en donde se encuentra actualmente el hilo anterior dentro del for. Además, así ocurre con todos los hilos siguientes. Es decir, este código no me asegura que cada hilo aumente el contador 10000 veces.

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	82616	99073
2	86780	103177
3	92721	98464
4	78924	78924
5	45137	103986

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

Sí hay acceso concurrente a una variable compartida. Dicha variable es llamada "mayor" y ocurre concurrencia en la siguiente línea de código:

```
@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);
}
```

En el método run ocurre un problema de concurrencia al momento de realizar la condición donde compara el mayor local junto con el mayor global. Lo anterior debido a que una vez entra a la condición ocurre una pausa de 250 ms. Esta pequeña pausa permite que el siguiente hilo no se compare correctamente con el mayor actualizado. Es decir, lo correcto sería que hilo que entra actualiza el mayor global con el mayor local actual. Mientras éste hilo está en dicha pausa los otros deberían estar a la espera para que el programa no colapsé. Sin embargo, en este código puede existir el caso donde pueden entrar varios hilos antes de que el primer hilo actualice la variable mayor.
Lo anterior representa un problema de posibles inconsistencias y un mal manejo de concurrencia en Java, esto es sinónimo de problemas en la ejecución del programa.

3. ¿Puede obtener alguna conclusión?

La conclusión general radica en que toca ser cuidadosos en el manejo de Threads en Java. Lo anterior puede ocasionar problemas de inconsistencia e incluso redundancia al momento de ejecutar un programa. Dichos problemas son ocasionados por un fenomeno que se le denomina concurrencia; es decir, cuando se presenta concurrencia en Java lo ideal es usar monitores si el programa lo requiere, por medio de los monitores se logra un control y una sincronización en la ejecución de múltiples hilos (procesos) en el código. No obstante, en el presente taller ocurrieron problemas en el código, debido a el mal manejo de la concurrencia en la ejecución de múltiples hilos.