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 valor se obtiene como resultado el número de 10000000.

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.

No , al ejecutar el programa se obtiene valores cercanos a los 10000000, pero no son dicho número, esto se debe a que existen problemas entre los diversos threads que intentan cambiar el valor de la variable del contador al mismo tiempo.

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

Ejecución	Valor obtenido
1	9735258
2	9646984
3	9646861
4	9710199
5	9792846

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

Si, existen muchos threads que son ejecutados al mismo tiempo en la variable contador, lo que implica que su valor no sea modificado correctamente en la pila de registro en memoria.

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. La matriz tiene 4 formas diferentes de generación. Comente y descomente las líneas de código necesarias para probar solo una de esas formas por experimento.

```java
public class MayorMatrizThread extends Thread {
    // Constante de la clase. Todos los objetos de esta clase ven esta
    // constante.
    private final static int SIZE = 20;

    // Variables de la clase. Todos los objetos de esta clase ven estas
    // variables.
    private static int[][] matriz;
    private static int mayor = 0;

    // El id del thread corresponde a la fila de la matriz
    private int id;

    // Variable que almacena el mayor elemento de la fila que le corresponde al
    // thread.
    private int mayorFila;

    // Método constructor. Asigna el id al thread. Inicializa el mayor.
    public MayorMatrizThread(int n) {
        id = n;
        mayor = -1;
    }

    // Este método se ejecuta al llamar el método start().
    // Cada thread incrementa 10 mil veces el valor del contador.
    public void run() {
        // Busca el mayor de los elementos de la fila.
        mayorFila = matriz[id][0];
        for (int j = 0; j < matriz[id].length; j++) {
            if (matriz[id][j] > mayorFila) {
                mayorFila = matriz[id][j];
            }
        }

        // Actualiza el mayor elemento de la matiz global si corresponde.
        if (mayorFila > mayor) {
            mayor = mayorFila;
        }
    }

```

```java
// Crea una matriz cuadrada del tamaño especificado con valores aleatorios entre 0 y 999.
private static void crearMatriz(int n) {
    matriz = new int[n][n];

    int[] numeros=numerosRandomSinRepetir(1, n*n, n*n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            matriz[i][j] = numeros[(n*i) + j ]; //Para números del 1 al size*size aleatorios, sin repetir
            matriz[i][j] = (int) (i*n)+(j+1); //Para numeros aleatorios del 1 al size*size
            matriz[i][j] = (int) (n-i)*(n-j); //para numeros de manera descendente ordenados
            matriz[i][j] = (i*n)+(j+1); //para numeros de manera ascendente ordenados
        }
    }
}

private static void imprimirMatriz() {
    for (int i = 0; i < matriz.length; i++) {
        for (int j = 0; j < matriz[i].length; j++) {
            System.out.print(matriz[i][j] + "\t");
        }
        System.out.println();
    }
}

//Funcion para generar numeroa aleatorios que no se repitan
public static int[] numerosRandomSinRepetir(int minimo, int maximo, int longitud){

    //En caso de que uno sea mayotr que otro
    //Lo intercambiamos
    if(maximo<minimo){
        int aux=maximo;
        maximo=minimo;
        minimo=aux;
    }

    //Si caben los numeros del rango
    //Generamos el array
    if( (maximo-minimo) >= (longitud-1) ){
```

```
        int numero_elementos=0;
        int numeros[]=new int[longitud];
        //RECOMENDADO: rellena el arreglo con un numero que nunca se va a generar
        Arrays.fill(numeros, minimo-1);
        boolean encontrado;
        int aleatorio;

        //Hasta que el numero de elementos no sea como el de la longitud del array no salimos
        while(numero_elementos<longitud){

            aleatorio=generaNumeroAleatorio(minimo, maximo);
            encontrado=false;

            //Buscamos si el numero existe
            for(int i=0;i<numeros.length && !encontrado;i++){
                if(aleatorio==numeros[i]){
                    encontrado=true;
                }
            }

            //Sino lo agregamos
            if(!encontrado){
                numeros[numero_elementos++] = aleatorio;
            }

        }

        return numeros;

    }else{
        System.out.println("No se puede generar el arreglo, revusa los parametros");
        return null;
    }


}

//Genera un numero aleatorio entre el minimo y el maximo, incluido el maximo y el minimo
public static int generaNumeroAleatorio(int minimo,int maximo){

    int num=(int)Math.floor(Math.random()*(maximo-minimo+1)+(minimo));
    return num;
}
```

```
    public static void main(String[] args) {
        int i;
        // Se crea un array de threads del tamaño especificado.
        // Cada thread va a manejar una fila de la matriz.
        MayorMatrizThread[] t = new MayorMatrizThread[SIZE];

        crearMatriz(SIZE);

        // Se crean e inician los threads del array.
        for (i = 0; i < SIZE; i++) {
            t[i] = new MayorMatrizThread(i);
            t[i].start();
        }

        imprimirMatriz();

        System.out.println("\nEl mayor elemento de la matriz es: " + mayor);
    }
}
```

Responda:

1. Ejecute cinco veces el programa (para cada una de las 4 formas de generar la matriz) y escriba el resultado obtenido en cada ejecución.
   Para números del 1 al size*size aleatorios sin repetir.

Ejecución	Valor obtenido	Valor esperado
1	398	400
2	400	400
3	390	400
4	397	400
5	396	400

Para números aleatorios del 1 al size*size.

Ejecución	Valor obtenido	Valor esperado
1	400	400
2	400	400
3	400	400
4	400	400
5	400	400

Para números de manera descendente ordenados.

Ejecución	Valor obtenido	Valor esperado
1	40	400
2	60	400
3	40	400
4	60	400
5	60	400

Para números de manera ascendente ordenados.

Ejecución	Valor obtenido	Valor esperado
1	400	400
2	400	400
3	400	400
4	400	400
5	400	400

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

Si, existe acceso concurrente, puesto que existen conflictos entre los distintos tipos para hallar la variable mayor en el caso 1 y 3. El conflicto se presenta en el método run cuando se sobrescribe la mayor fila con el elemento. Esto provoca inconvenientes entre los distintos threads.

3. ¿Puede obtener alguna conclusión?

Utilizar esta metodología de threads solamente puede ser implementada si se manejan datos de manera ascendente ordenada, o si se manejan datos aleatorios repetidos. En caso contrario, los hilos se cruzarán donde el método run.