LABORATORIO 1 - MANEJO DE THREADS

Universidad de los Andes, Bogotá, Colombia.
ISIS 2203 - Infraestructura Computacional
Juan D. Lugo Sánchez
29 enero de 2021

Mediante el desarrollo de este laboratorio se pretende comprender la creación de Threads y su implementación en aplicaciones concurrentes tanto en Java como en Python, así como la necesidad de la sincronización y acceso recurrente a variables de acceso común.

1. Taller 1

Se revisaron y ejecutaron los ejemplos 1 y 2 del taller 1 con el fin de establecer las principales semejanzas y diferencias entre *Thread* y *Runnable*.

```java
package ej_Threads_Taller1;

public class T01_Creacion extends Thread{

    public void run() {
        System.out.println("Extendiendo la clase Thread");
    }

    public static void main(String [] args) {
        T01_Creacion thread =new T01_Creacion();
        thread.start();
    }

}
```

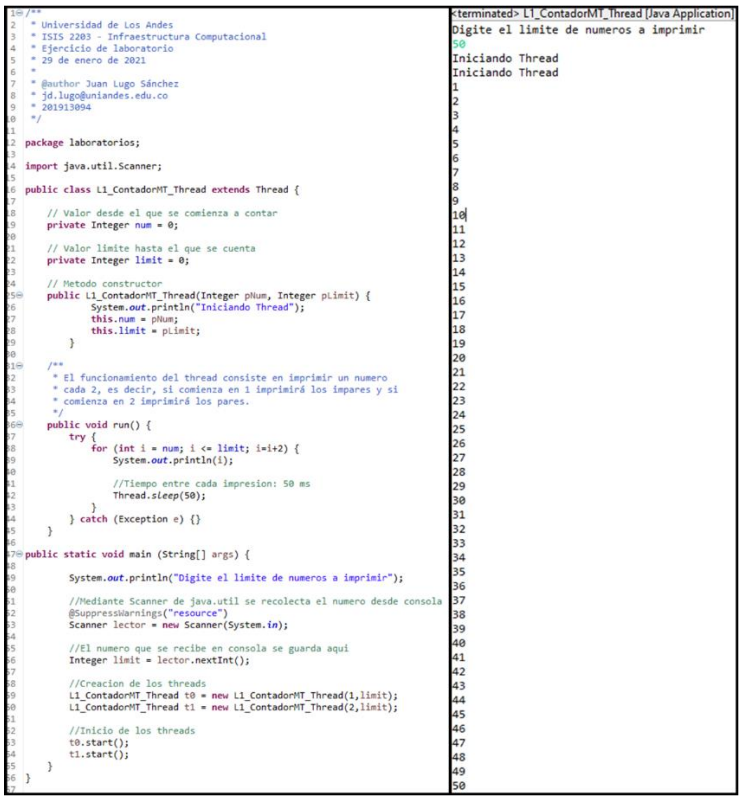
```java
package ej_Threads_Taller1;

public class R01_Creacion implements Runnable{

    @Override
    public void run() {
        System.out.println("Implmentacion de interfaz Runnable");
    }

    public static void main(String[] args) {
        Thread t = new Thread(new R01_Creacion());
        t.start();
    }

}
```

Figura 1. Creación de *threads* mediante la extensión de *Thread* e implementación de *Runnable* (1)

Semejanzas	Diferencias
• Tienen la misma estructura, se crean y después se inician • Los *scrips* que ejecuta el *therad* se establecen en el método `run()`.	• La principal diferencia es que *Thread* y *Runnable* es que los *threads* de *Thread* se pueden iniciar múltiples veces mientras que los de *Runnable* solo se pueden volver a iniciar hasta que terminen su método `run()`. • La creación es diferente, mientras que uno crea un objeto de la sisma clase el otro crea uno de la clase *thread* y le pasa por parámetro la clase sobre la que se está trabajando.

Figura 2. Tabla de diferencias entre extensión de *Thread* e implementación de *Runnable* (1)

Respecto al ejercicio planteado de realizar un contador que mediante 2 *threads* imprimieran los números pares e impares en simultaneo. Se logró hacer la implementación del ejercicio en Java mediante las clases *Thread* y *Runnable*, mientras que para Python se realizó el ejercicio usando referencias a objetos invocables.



```
/**
 * Universidad de Los Andes
 * ISIS 2203 - Infraestructura Computacional
 * Ejercicio de laboratorio
 * 29 de enero de 2021
 *
 * @author Juan Lugo Sánchez
 * jd.lugo@uniandes.edu.co
 * 201913094
 */

package laboratorios;

import java.util.Scanner;

public class L1_ContadorMT_Thread extends Thread {

    // Valor desde el que se comienza a contar
    private Integer num = 0;

    // Valor límite hasta el que se cuenta
    private Integer limit = 0;

    // Metodo constructor
    public L1_ContadorMT_Thread(Integer pNum, Integer pLimit) {
            System.out.println("Iniciando Thread");
            this.num = pNum;
            this.limit = pLimit;
        }

    /**
     * El funcionamiento del thread consiste en imprimir un numero
     * cada 2, es decir, si comienza en 1 imprimirá los impares y si
     * comienza en 2 imprimirá los pares.
     */
    public void run() {
        try {
            for (int i = num; i <= limit; i=i+2) {
                System.out.println(i);

                //Tiempo entre cada impresion: 50 ms
                Thread.sleep(50);
            }
        } catch (Exception e) {}
    }

public static void main (String[] args) {

        System.out.println("Digite el límite de numeros a imprimir");

        //Mediante Scanner de java.util se recolecta el numero desde consola
        @SuppressWarnings("resource")
        Scanner lector = new Scanner(System.in);

        //El numero que se recibe en consola se guarda aqui
        Integer limit = lector.nextInt();

        //Creacion de los threads
        L1_ContadorMT_Thread t0 = new L1_ContadorMT_Thread(1,limit);
        L1_ContadorMT_Thread t1 = new L1_ContadorMT_Thread(2,limit);

        //Inicio de los threads
        t0.start();
        t1.start();
    }
}
```

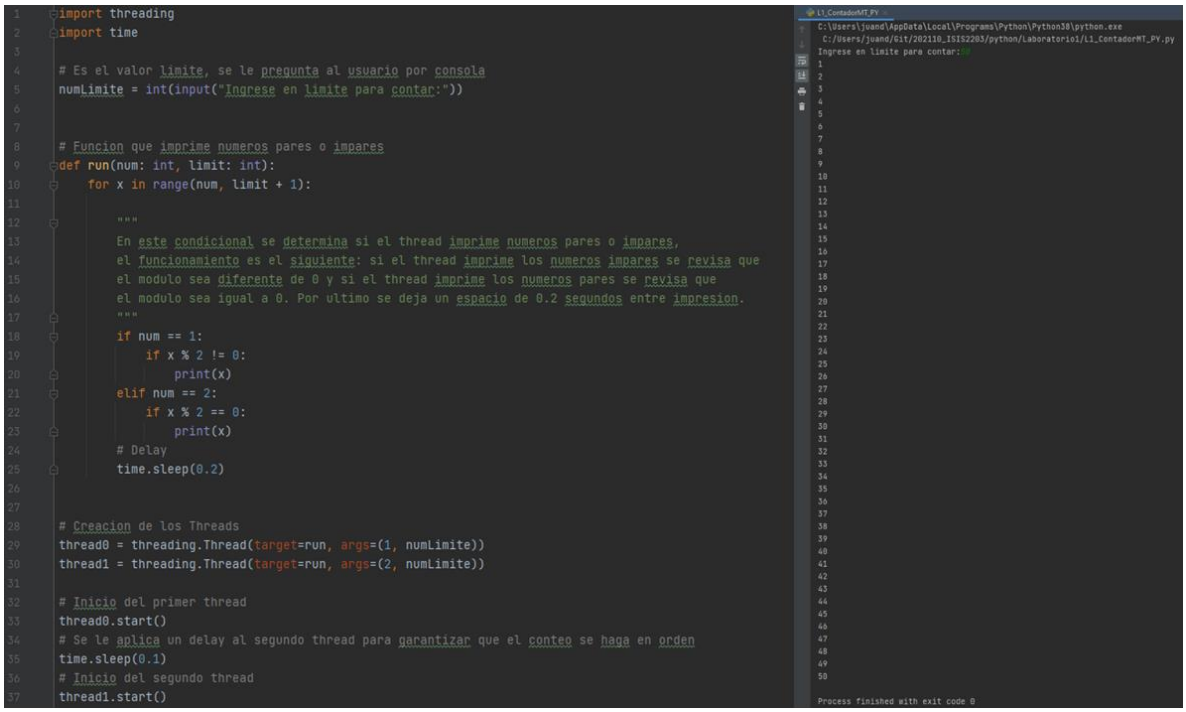
```
<terminated> L1_ContadorMT_Thread [Java Application]
Digite el límite de numeros a imprimir
50
Iniciando Thread
Iniciando Thread
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
```

Figura 3. Contador de pares e impares con la implementación de la clase *Thread* junto con su resultado (1)

```
/**
 * Universidad de Los Andes
 * ISIS 2203 - Infraestructura Computacional
 * Ejercicio de laboratorio
 * 29 de enero de 2021
 *
 * @author Juan Lugo Sánchez
 * jd.lugo@uniandes.edu.co
 * 201913094
 */

package laboratorios;

import java.util.Scanner;

// Esta clase implementa la interfaz Runnable
public class L1_ContadorMT_Runnable implements Runnable{

    //Valor desde el que se comienza a contar
    private Integer num = 0;

    //Valor límite hasta el que se cuenta
    private Integer limit = 0;


    //Metodo constructor
    public L1_ContadorMT_Runnable(Integer pNum, Integer pLimit) {
        System.out.println("Iniciando Thread");
        this.num = pNum;
        this.limit = pLimit;
    }

    /**
     * El funcionamiento del thread consiste en imprimir un numero
     * cada 2, es decir, si comienza en 1 imprimirá los impares y si
     * comienza en 2 imprimirá los pares.
     */
    @Override
    public void run() {
        try {
            for (int i = num; i <= limit; i=i+2) {
                System.out.println(i);

                //Tiempo entre cada impresion: 50 ms
                Thread.sleep(50);
            }
        } catch (Exception e) {}
    }

    public static void main (String[] args) {

        System.out.println("Digite el límite de numeros a imprimir");

        //Mediante Scanner de java.util se recolecta el numero desde consola
        @SuppressWarnings("resource")
        Scanner lector = new Scanner(System.in);

        //El numero que se recibe en consola se guarda aqui
        Integer limit = lector.nextInt();

        //Creacion de los threads
        Thread t0 = new Thread(new L1_ContadorMT_Runnable(1,limit));
        Thread t1 = new Thread(new L1_ContadorMT_Runnable(2,limit));

        //Inicio de los threads
        t0.start();
        t1.start();
    }
}
```

```
<terminated> L1_ContadorMT_Runnable [Java Application]
Digite el límite de numeros a imprimir
50
Iniciando Thread
Iniciando Thread
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
```

Figura 4. Contador de pares e impares con la implementación de la clase *Runnable* junto con su resultado (1)



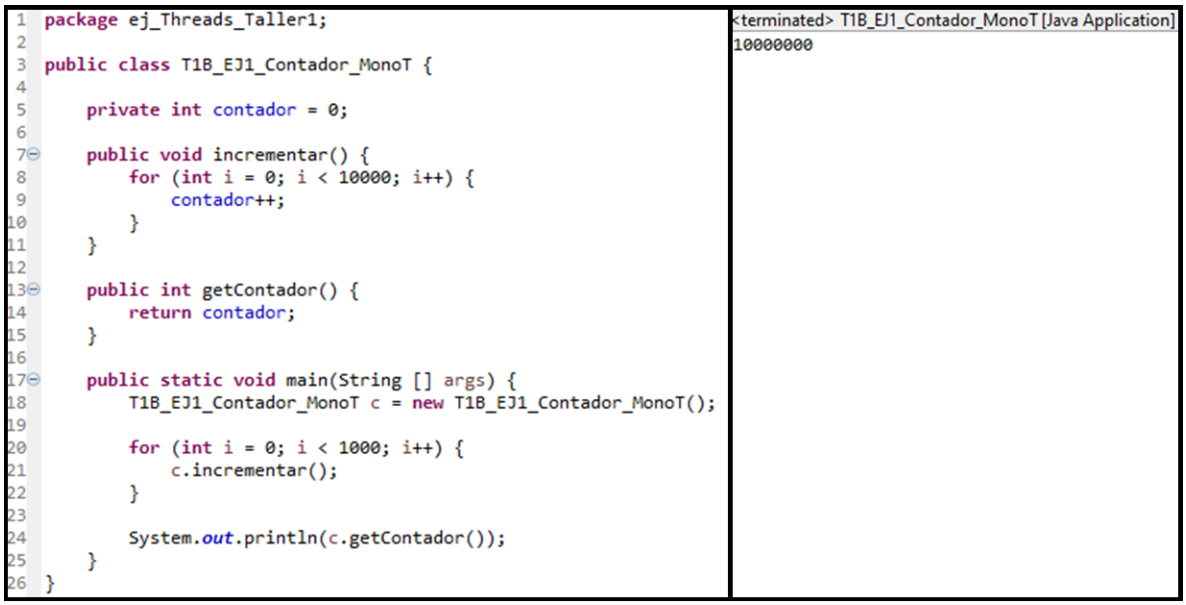
```python
import threading
import time

# Es el valor limite, se le pregunta al usuario por consola
numLimite = int(input("Ingrese en limite para contar:"))


# Funcion que imprime numeros pares o impares
def run(num: int, limit: int):
    for x in range(num, limit + 1):

        """
        En este condicional se determina si el thread imprime numeros pares o impares,
        el funcionamiento es el siguiente: si el thread imprime los numeros impares se revisa que
        el modulo sea diferente de 0 y si el thread imprime los numeros pares se revisa que
        el modulo sea igual a 0. Por ultimo se deja un espacio de 0.2 segundos entre impresion.
        """
        if num == 1:
            if x % 2 != 0:
                print(x)
        elif num == 2:
            if x % 2 == 0:
                print(x)
        # Delay
        time.sleep(0.2)


# Creacion de los Threads
thread0 = threading.Thread(target=run, args=(1, numLimite))
thread1 = threading.Thread(target=run, args=(2, numLimite))

# Inicio del primer thread
thread0.start()
# Se le aplica un delay al segundo thread para garantizar que el conteo se haga en orden
time.sleep(0.1)
# Inicio del segundo thread
thread1.start()
```

```
C:\Users\juand\AppData\Local\Programs\Python\Python38\python.exe
C:/Users/juand/Git/202110_ISIS2203/python/Laboratorio1/L1_ContadorMT_PY.py
Ingrese en limite para contar:
```

Figura 5. Contador de pares e impares con la implementación de objetos invocables en Python (1)

2. Taller 1b parte 1

1. Con el fin de establecer la diferencia entre los contadores monothread y multithread se ejecuto el ejemplo 1, **Esperando que el resultado fuese que el contador final fuera de 10.000.000.** Dicha suposición se pudo confirmar una vez el programa terminó su ejecución.



```java
package ej_Threads_Taller1;

public class T1B_EJ1_Contador_MonoT {

    private int contador = 0;

    public void incrementar() {
        for (int i = 0; i < 10000; i++) {
            contador++;
        }
    }

    public int getContador() {
        return contador;
    }

    public static void main(String [] args) {
        T1B_EJ1_Contador_MonoT c = new T1B_EJ1_Contador_MonoT();

        for (int i = 0; i < 1000; i++) {
            c.incrementar();
        }

        System.out.println(c.getContador());
    }
}
```

```
<terminated> T1B_EJ1_Contador_MonoT [Java Application]
10000000
```

Figura 6. Contador ejemplo contador *monothread* (1b)

2. Para el ejemplo 2 del taller 1b las **hubo un inesperado cambio en el valor que se presuponía sería el mismo que en el ejemplo pasado**, ya que en esencia hacen lo mismo. Sin embargo, al tener 2 *threads* que modifican la misma variable local en algunas ocasiones hubo conflictos sobre la escritura, causando así que no se llegase al número esperado.

3. El programa se ejecutó 5 veces, los siguientes resultados:

Ejecución	Valor obtenido
1	9196620
2	9318423
3	9335645
4	9268568
5	9197301

Figura 7. Tabla de resultado de ejecución ejemplo contador *multithread* (1b)

4. El acceso a la variable compartida de contador tiene genera concurrencia, ya que 1000 *threads* contando a la vez en un espacio reducido de ms genera que en ocasiones se bloquee la variable sobre la que se cuenta, algo que se puede evidenciar ya que la diferencia promedio entre el valor esperado y los valores obtenidos es de 736000.

3. Taller 1b parte 2

1. Para la segunda parte del taller los resultados fueron mucho más volátiles respecto a los ejemplos anteriores, las 5 ejecuciones del programa fueron prueba de ello.

Ejecución	Valor obtenido	Valor esperado
1	80869	94167
2	89650	89650
3	88301	88301
4	100236	101625
5	94807	98256

Figura 7. Tabla de resultado de ejecución ejemplo máximo de una matriz *multithread* (1b)

2. El acceso concurrente de este ejemplo se encuentra en se encuentra sobre la variable mayor, ya que es la variable de máximo global a la que acceden y modifican todos los threads a la vez.

3. Se puede concluir que es importante la sincronización para evitar que se pierdan y/o se sobrescriban variables de uso común en programas multithread.

Enlace al repositorio:
https://github.com/JuanLugoS/202110_ISIS2203
Ejercicios taller 1:

- https://github.com/JuanLugoS/202110_ISIS2203/blob/master/src/laboratorios/L1_ContadorMT_Runnable.java
- https://github.com/JuanLugoS/202110_ISIS2203/blob/master/src/laboratorios/L1_ContadorMT_Thread.java
- https://github.com/JuanLugoS/202110_ISIS2203/blob/master/python/Laboratorio1/L1_ContadorMT_PY.py