



Module	Assessment Type
Distributed and Cloud Systems Programming	Individual Report

## Workshop 6

Student Id : 2049867 (NP03A190017)  
Student Name : Roshan Parajuli  
Section : L5CG3  
Module Leader : Rupak Koirala  
Lecturer /Tutor : Saroj Sharma  
Submitted on : 2020-04-19

## Introduction

Multithreading is the phenomenon of executing more than a thread in the system, where the execution of these threads can be of two different types, such as Concurrent and Parallel multithread executions.

### 1. Compiling the first Concurrent program

First of all, a module folder was created and within a subfolder, the project was opened through a code editor. The code from the workshop was copied as is and was run to check the output.

```
second Thread
second Thread
first Thread
second Thread
second Thread
first Thread
first Thread
second Thread
second Thread
second Thread
second Thread
first Thread
first Thread
```

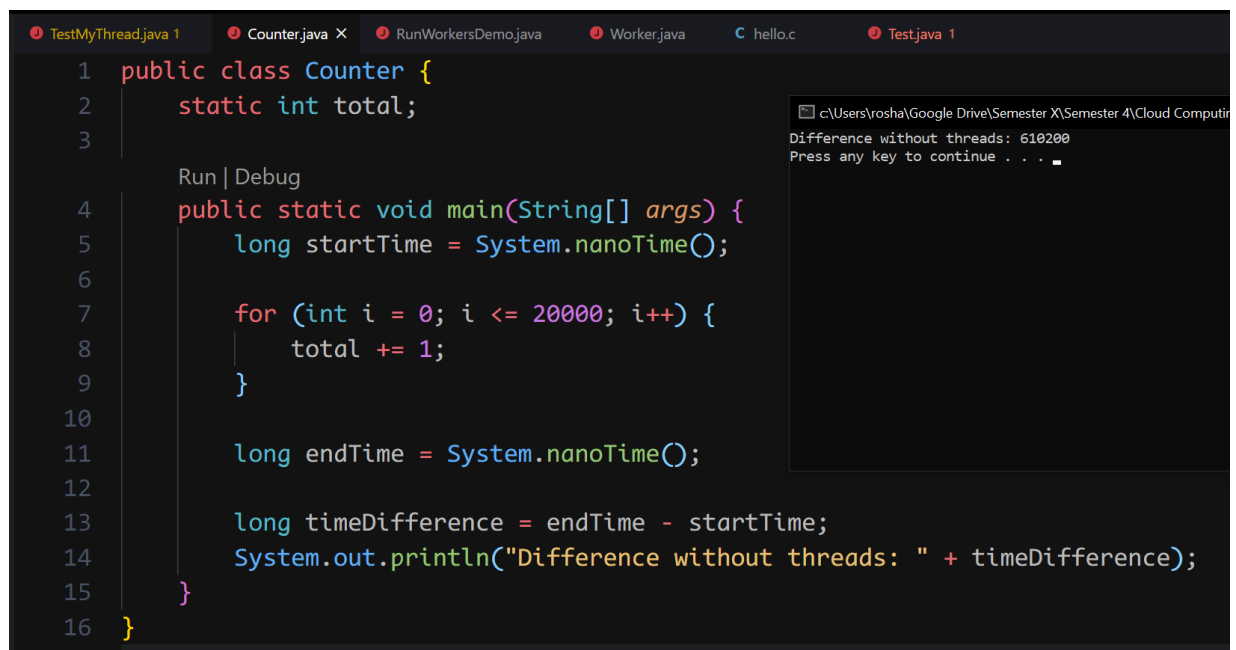
At first, As the output describes, the message passed from TestMyThread is received by the constructor of MyThread class and then printed out by the run function. MyThread class extends Thread because if it didn't the run function would act as a normal function and not as a multithreading function. The run function which is a multithreading function is then called by start method from the TestMyThread class and it is run asynchronously. It is confirmed that the output produced is similar to that described in the lecture notes.

```
first thread is printed 20 times
second thread is printed 20 times
Press any key to continue . . .
```

```
first thread is printed 20 times
second thread is printed 20 times
third thread is printed 20 times
Press any key to continue . . .
```

A new thread is added and joined, and the counter variable is yet again printed on the screen. The exception that might arise during the joining of the threads is handled with a try catch block.

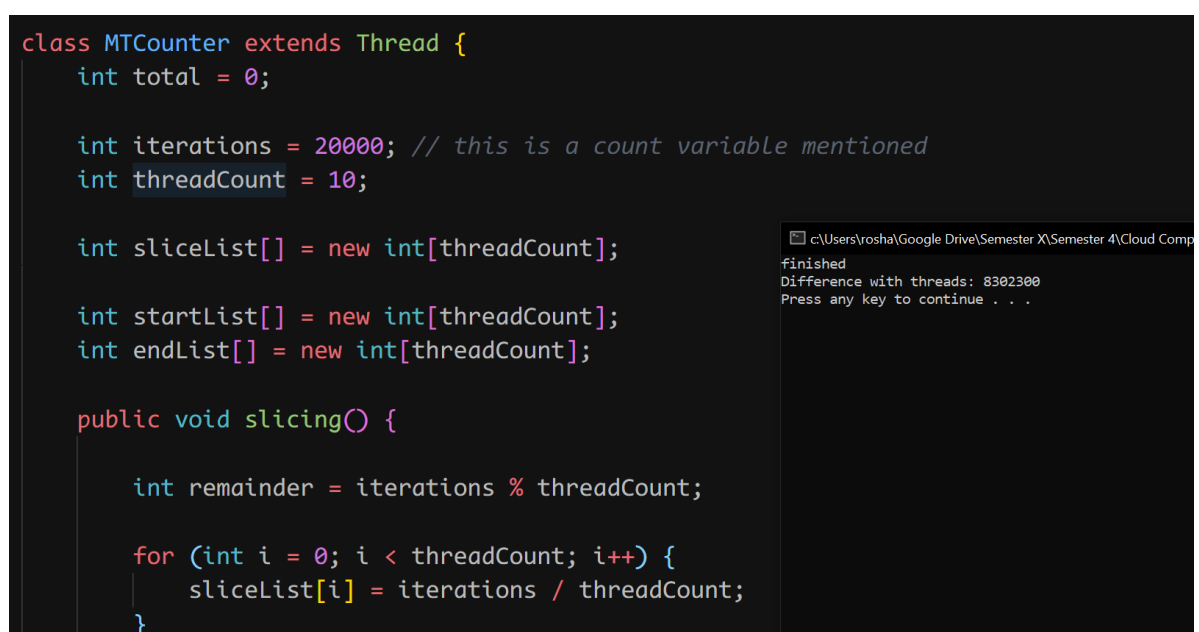
## 2. Multithreading Counter Program



```
1 public class Counter {
2     static int total;
3
4     public static void main(String[] args) {
5         long startTime = System.nanoTime();
6
7         for (int i = 0; i <= 20000; i++) {
8             total += 1;
9         }
10
11         long endTime = System.nanoTime();
12
13         long timeDifference = endTime - startTime;
14         System.out.println("Difference without threads: " + timeDifference);
15     }
16 }
```

Output: Difference without threads: 610200  
Press any key to continue . . .

The Counter class here is a normal (not multithreaded) function which has a loop to count up to 20000 updating a variable total while it is doing so. The timing is noted at the start and the end of the loop with the nanoTime method from the System class. After that, the difference without threads is printed on the screen.



```
class MTCounter extends Thread {
    int total = 0;

    int iterations = 20000; // this is a count variable mentioned
    int threadCount = 10;

    int sliceList[] = new int[threadCount];

    int startList[] = new int[threadCount];
    int endList[] = new int[threadCount];

    public void slicing() {
        int remainder = iterations % threadCount;

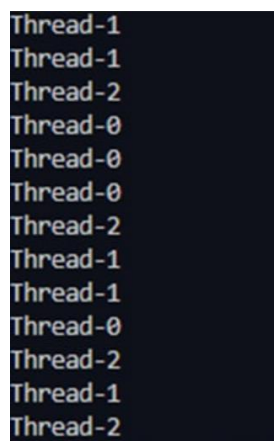
        for (int i = 0; i < threadCount; i++) {
            sliceList[i] = iterations / threadCount;
        }
    }
}
```

Output: finished  
Difference with threads: 8302300  
Press any key to continue . . .

On the MTCounter class, appropriate slicing is done, and the work is divided equally among all the threads. After that, ten threads are created dynamically and are joined. The design issue that is noticed is the complication of the program at first. It can also be noticed that the multithreaded program is slower than the one which was not multithreaded in this case. It is because of the amount of overhead in the multithreaded program. Slicing, loops for dynamic creation of threads were all the overheads that led to the program time getting higher than the other.

### 3. A multithreaded program

In a new project called workersExample, the code was pasted from the lecture notes.



```
Thread-1
Thread-1
Thread-2
Thread-0
Thread-0
Thread-0
Thread-2
Thread-1
Thread-1
Thread-0
Thread-2
Thread-1
Thread-2
```

The output produced was similar to that described in the lecture notes. Instead of extending the Thread class, it is instantiated here and the worker object is passed as an argument.

```
public class RunWorkersDemo {  
  
    Run | Debug  
    public static void main(String[] args) {  
        Worker w1, w2, w3, w4;  
  
        Thread t1, t2, t3, t4;  
        long start = System.nanoTime();  
  
        w1 = new Worker();  
        w2 = new Worker();  
        w3 = new Worker();  
        w4 = new Worker();  
  
        t1 = new Thread(w1);  
        t2 = new Thread(w2);  
        t3 = new Thread(w3);  
        t4 = new Thread(w4);  
  
        t1.start();  
        t2.start();  
        t3.start();  
        t4.start();  
  
        t1.join();  
        t2.join();  
        t3.join();  
        t4.join();  
  
        long end = System.nanoTime();  
        long time = end - start;  
        System.out.println("running time = " + time + " nano seconds or " + (time / 1000000000.0) + "s");  
        System.out.println("Press any key to continue . . .");  
    }  
}
```

c:\Users\rosha\Desktop\workersExample - VS Code Console

Thread-3  
Thread-1  
Thread-3  
Thread-2  
Thread-3  
Thread-0  
Thread-3  
Thread-1  
Thread-2  
Thread-3  
Thread-0  
Thread-0  
Thread-1  
Thread-0  
Thread-1  
Thread-2  
Thread-2  
Thread-1  
Thread-0  
Thread-2  
Thread-2  
running time = 1112210300 nano seconds or 1.112210s  
Press any key to continue . . .

Here a new thread was added and started as well as joined. The terminal which printed the name of the thread has shown the occurrence of “Thread-3” which is actually the fourth thread as the naming started from zero.

## Conclusion

To sum up, the tasks focused on the usage of different threads and their ways of implementation. The design issues regarding the same were also learnt. In such way, the CPU resources were shared and utilized and it was useful for improving application responsiveness.