Seventh Assignment Report

Mohammad Ghaderi

401 222 104

Introduction:

- In this assignment we have Three tasks:
    1. Finding an algorithm to calculating PI using "Multithreading Concept" in Java.
    2. Learning how to scheduling running threads like a Priority Simulator implemented by "Multithreading concept " in Java.
    3. Handling "Race Condition" using "Semaphore" in java.

    That all these tasks should be handled by multithreading concepts.

Design and Implementation:

1. Calculating PI:
   in this task we should calculate PI number amount with 1000 -digits accuracy.
   One of the formulas for calculating the exact amount of PI is this:

$$\pi = \sum_{n=0}^{\infty} \frac{(n!)^2 2^{n+1}}{(2n+1)!}$$

For making sure that it will get the accuracy amount very well I used the formula I calculated up to the five

thousandth term of the sequence and managed it with "thread pool" concept with " 16 threads ".

Because there was a critical section in this algorithm (where the program wants to add the new calculated method to a temporary object) I used the " Synchronized " concept and make that method "synchronized " for preventing race condition.

2. Priority Simulator:
   We should simulate a priority system between black, blue and white threads.
   for implementing this task I used the "Count Down Latch"
   A **Count Down Latch** is a synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes. It has a **counter field** that can be decremented by calling the **countDown()** method. When the counter reaches zero, the thread that called the **await()** method can resume its execution. A Count Down Latch can be used to coordinate parallel processing or to ensure that a dependent thread starts only after other threads have finished their tasks. And as we are expected to implement a method to put our mind to ease that the priority (black threads > blue threads > white threads) will be performed.
   so for implementing this we first make an instance for each kind of thread from the "Count Down Latch" class

In the "Runner" class and also the same thing for the each of the color_thread classes and then initial it in their constructors.
after all of this we should call the "**await()**" method after the each **color_thread** starts to make sure that the priority will followed.

3. For this task we should use the "Semaphore " concept in java but what is semaphore?
   A **semaphore** is a synchronization aid that controls access to a shared resource through the use of a counter. A semaphore has a number of permits that can be acquired and released by threads. A thread that wants to access the resource must acquire a permit from the semaphore by calling the **acquire()** method. If the semaphore has no permits available, the thread will block until one is available. A thread that is done with the resource must release the permit back to the semaphore by calling the **release()** method. This way, a semaphore can limit the number of concurrent threads accessing the resource.
   A semaphore can also be used as a **lock** or a **binary semaphore** if it is initialized with one permit. This means that only one thread can access the resource at a time, and it must release the permit before another thread can acquire it.
   As you already know we should implement this semaphore in a way that at the maximum there are only two threads can get into the critical section. So for this

reason we set the counter with "2" and this approach assure us that the tasks will done correctly.

Done!

I hope this report has helped you.