הנדסת תוכנה

מעבדה 2

סלאם קייס 327876116

סאמי סרחאן 327876298

שאלה 1 :

Thread.start היא מאתחלת חוט חדש ומפעילה את הפונקציה run שלו, לעומת הפעלת Thread.run() רק תריץ את הקוד של המתודה run .

שאלה 2 :

.1

```
Hello world from thread number 4
Hello world from thread number 9
Hello world from thread number 8
Hello world from thread number 3
Hello world from thread number 0
Hello world from thread number 6
Hello world from thread number 2
Hello world from thread number 5
Hello world from thread number 7
Hello world from thread number 1
That's all, folks
```

איך שאנחנו רואים הקוד מפעיל את המתודה RUN לכל חוט בלי סדר אבל הוא כן מחכה לכל החוטים שיסיימו.

.2

```
That's all, folks
Hello world from thread number 3
Hello world from thread number 5
Hello world from thread number 7
Hello world from thread number 1
Hello world from thread number 0
Hello world from thread number 6
Hello world from thread number 8
Hello world from thread number 9
Hello world from thread number 4
Hello world from thread number 2
```

עכשיו איך שאנחנו רואים שהדפסנו בהתחלה That's all, folks לפני סיום המתודה RUN של כל החוטים וזה בגלל שאנחנו הסרנו את שורת ה JOIN שהיא מתודה המחכה עד שהחוט הנתון יסיים את הריצה שלו

לכן החוט הראשי שלנו עכשיו יכול לסיים את ריצתו לפני שהאחרים יסימו

ואז קיבלנו את התוצאה הזאת

.3

```
Hello world from thread number 0
Hello world from thread number 1
Hello world from thread number 2
Hello world from thread number 3
Hello world from thread number 4
Hello world from thread number 5
Hello world from thread number 6
Hello world from thread number 7
Hello world from thread number 8
Hello world from thread number 9
That's all, folks
```

עכשיו כל ההדפסות יהיו בסדר מי ש עושה START ראשון הוא מדפיס ראשון וזה בגלל שעשינו JOIN אחרי START זה גורם שכל פעם אנחנו עושים START נחכה עד שהחוט מסתיים ואז נמשיך לחוט הבא, לכן אנחנו קיבלנו זה בצורה מסודרת.

.4

CURRENTTHREAD היא מתודה סטטית המחזירה את החוט הנוכחי אז הקריאה ;()currentThread.T.join() היא תעשה המתנה לחוט הנוכחי מהחוט עצמו אז מזה נקבל חוט ימתין לעצמו וכל עוד הוא ממתין לעצמו זה אומר שהוא לא יסתיים ונוצרת לנו דבר שגיו.

שאלה 3

```
Sum (Single-threaded): 4294967297
Total execution time: 1.302 seconds

Sum (muti-threaded): 4294967297
Total execution time: 0.519 seconds
```

1) האם יש יתרון לביצוע החישוב באמצעות מספר חוטים? הסבירו.

כאשר ישנם מספר של מעבדים במחשב , כן, יש יתרון בביצוע החישוב באמצעות מספר חוטים במקרים מסוימים. כאשר מבצעים משימה שבה יכולים לחלק אותה ל תתי משימות בלתי תלויות ,אז החוטים יכולים לחשב את התוצאה במקביליות וזה מפחית זמן החישוב כמו במקרה הזה אנחנו חלקנו את הסכימה לתת קבוצות בלתי תלויות כל חוט מחשב סכום של טווח מסוים

אם יש רק מעבד אחד במחשב, לא יהיה יתרון בביצוע החישוב באמצעות מספר חוטים. במעבד יחיד, כל חוט יתבצע באופן סינכרוני, ולכן לא תהיה שיפור בביצועים. במקרה כזה.

2) האם משך זמן החישוב באמצעות חוטים הוא קבוע? מדוע?

משך זמן החישוב באמצעות חוטים לא תמיד קבוע. זה תלוי בתזמון של החוטים (מי בוחרים בהתחלה?,מתי מחלפים חוט? , עם מי? ...) ועוד. כמו כן, משך זמן החישוב עשוי להשתנות בין הרצות שונות של התוכנית עצמה

3) צרפו את קוד המחלקה SumThreads לדוח.

```java
package org.example;
import java.lang.Thread;
import java.util.concurrent.TimeUnit;
public class SumThreads extends Thread{
    private long start,end,sum;  2 usages
    public SumThreads(long start,long end){  1 usage
        this.start=start;
        this.end=end;
        this.sum=0;
    }
    public long getSum(){  1 usage
        return sum;
    }
    @Override
    public void run() {
        for(long i=start;i<end;i++)
        {
            sum+=1;
        }
    }
```

```java
public static void main(String[] args) {


    long startTime = System.nanoTime(); // Computation start tim
    long sum=0;
    long number_to_colac = ((2L << 31)+1) / 10;
    long devion_offset = (((2L << 31)+1) % 10);
    SumThreads[] threads = new SumThreads[10]; // create an array of threads
    long calc = number_to_colac;
    for (int i = 0; i < 10; i++) {
        if(i==0)
        {
            calc = number_to_colac + devion_offset;
        }
        threads[i] = new SumThreads(0,calc);
        calc=number_to_colac;
    }

    for (SumThreads thread : threads) {
        thread.start();
    }
```

```java
    for (SumThreads thread : threads) {
        try {
            thread.join(); // wait for the threads to terminate
            sum+=thread.getSum();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    System.out.println();

    System.out.println("Sum (muti-threaded): " + sum);
    long  difference = System.nanoTime() - startTime;
    double differenceInSeconds = difference/1000000000.0;
    long millisecondsInDifference = TimeUnit.NANOSECONDS.toMillis(difference);
    System.out.format(
            "Total execution time: %.3f seconds\n",
            differenceInSeconds
    );
}}
```

שאלה 4 :

```java
public class ProducerConsumer2 {  no usages
    Queue<Integer> workingQueue = new LinkedList<Integer>();  4 usages
    public synchronized void produce(int num) throws InterruptedException {  no usages
        while (workingQueue.size() == 10) {
            wait();
        }
        workingQueue.add(num);
        notifyAll();
    }
    public synchronized Integer consume() throws InterruptedException {  no usages
        while (workingQueue.isEmpty()) {
            wait();
        }
        int  value = workingQueue.poll();
        notifyAll();
        return value;
    }
}
```

עכשיו אם הגענו ל 10 איברים ה producer יחכה ל consumer להוציא איבר בשביל שהוא יהיה יכול להוסיף עוד איברים ובזה אנחנו מבדים שהוא לא יעבור את 10 איברים תמיד.

תרגיל מעבדה 2:

```
500500
the running time is 0 seconds and 11 milliseconds
500500
the running time is 0 seconds and 1 milliseconds
```

סכום רגיל :

11ms

עם חוטים:

1ms

ג-הזמן לא קבוע הוא משתנה בכל ריצה בין (1-2ms) אנחנו מקבלים זמן הרבה יותר קצר בחישוב מקבילי באמצעות חוטים כי אנחנו מחלקים את משימת החישוב לעשרה חוטים שנעשים במקביל ו עם מעבד שתומך במקביליות אנחנו מקבלים תוצאות טובות.