מעבדה 2

מגישים :
אור דינר
נדב מלמן

שאלה 1

Thread.start() - המתודה הזו אומרת ל-JVM שצריך להוסיף THREAD לביצוע במקביל.

start

```
public void start()
```

Causes this thread to begin execution; the Java Virtual Machine calls the `run` method of this thread.

The result is that two threads are running concurrently: the current thread (which returns from the call to the `start` method) and the other thread (which executes its `run` method).

It is never legal to start a thread more than once. In particular, a thread may not be restarted once it has completed execution.

Throws:
`IllegalThreadStateException` - if the thread was already started.

מתוך התיעוד של ORACLE.

Thread.run() - כאשר קוראים בתוכנית למתודה הזו היא מריצה את run() בחוט אחד (לא מקביל) וממשיכה בביצוע התכנית המקורית לאחר מכן (ב-THREAD המתודה run() לא עושה כלום).

run

```
public void run()
```

If this thread was constructed using a separate `Runnable` run object, then that `Runnable` object's `run` method is called; otherwise, this method does nothing and returns.

Subclasses of `Thread` should override this method.

Specified by:
`run` in interface `Runnable`

מתוך התיעוד של ORACLE.

שאלה 2

1. הפלט של התכנית הוא :

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

הפלט הזה עלול להשתנות בכל הרצה חדשה.

2. הפלט של התכנית ללא הלולאה השלישית הוא :

```
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 8
```

Hello world from thread number 7
That's all, folks
Hello world from thread number 9

גם כאן הפלט עלול להשתנות בכל הרצה חדשה. הסיבה שהפלט נראה כך היא משום שאנחנו לא מחכים לביצוע כל ה-THREADS שיצרנו במהלך ההרצה, לכן סיימנו את התכנית המקורית לפני שסיימנו להריץ את כל ה-THREADS.

3. הפלט של התכנית כאשר נוסיף JOIN ישירות לאחר START הוא:

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

הסיבה לכך שכל חוט מתבצע לפי הסדר היא שלאחר כל יצירת חוט לביצוע, אנחנו מחכים שהוא יתבצע, ובכך אין מקביליות.

4. קריאה למתודה - Thread.currentThread().join() תחכה לביצוע החוט הנוכחי.

שאלה 3

1.

חוט ראשי בלבד

```
Sum = 4294967296
Total execution time: 0 min, 36 sec
```

10 חוטים

```
Sum = 4294967296
Total execution time: 0 min, 4 sec
```

השימוש ב-10 חוטים מאפשר את מציאת התשובה בפחות זמן, הסיבה לכך היא שכל חוט מחשב מספרים יותר קטנים מאשר סך התשובה ולכן בכל צעד אנחנו מוסיפים 1 (+1) למספרים יותר קטנים מאשר אם היינו מחשבים את כל המספר. (במעבד בעל ליבה אחת מקביליות אמיתית אינה אפשרית)

2. משך זמן החישוב אמור להיות קבוע באופן יחסי, גורמים אפשריים לכך שזמן החישוב אינו קבוע הם פעולות אחרות של המעבד במהלך הריצה של התכנית, ושימוש במחשבים בעל חומרה שונה.

3. קטע הקוד שרץ ובו 10 חוטים: (בעמוד הבא)

```java
package org.example;

import java.util.concurrent.TimeUnit;

public class CalcTime {

    public static void main(String[] args) {
        long startTime = System.nanoTime(); // Computation start time

        long sum = 0;
        int THREADS = 10;

        // since we divide 2^32 by 10 and it is not dividable, we subtract the remainder
from the total sum
        sum -= 10 - (long) Math.pow(2, 32) % THREADS;

        SumThread[] threads = new SumThread[THREADS];
        for (int i = 0; i < THREADS; i++)
            threads[i] = new SumThread(THREADS);

        for (int i = 0; i < THREADS; i++)
            threads[i].start();

        for (int i = 0; i < THREADS; i++) {
            try {
                threads[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        for (int i = 0; i < THREADS; i++)
            sum += threads[i].sum;

        System.out.println("Sum = " + sum);

        // The difference between the start time and the end time
        long difference = System.nanoTime() - startTime;
        // Print it out
        long minutesInDifference = TimeUnit.NANOSECONDS.toMinutes(difference);
        long secondsInDifference =
                TimeUnit.NANOSECONDS.toSeconds(difference) -
                        TimeUnit.MINUTES.toSeconds(minutesInDifference);
        System.out.format(
                "Total execution time: %d min, %d sec\n",
                minutesInDifference,
                secondsInDifference
        );
    }
}

class SumThread extends Thread {
    private final int THREADS;
    public long sum = 0;

    public SumThread(int THREADS) {
        this.THREADS = THREADS;
    }

    @Override
    public void run() {
        for (long i = 0; i < Math.pow(2, 32) / this.THREADS; i++)
            this.sum++;
    }
}
```

שאלה 4

```
public class ProducerConsumer2 {
    Queue<Integer> workingQueue = new LinkedList<Integer>();
    public synchronized void produce(int num) throws InterruptedException {
        if(workingQueue.size() >10)
            wait();
        workingQueue.add(num);
        notifyAll();
    }
    public synchronized Integer consume() throws InterruptedException {
        while (workingQueue.isEmpty()) {
            wait();
        }
        return workingQueue.poll();
    }
}
```

תרגיל מעבדה

ג. זמן הביצוע המקבילי עבור התכנית הסכימה עד 1000 תהיה מהירה יותר, בכל חוט נסכום מספרים יותר קטנים מאשר אם היינו סוכמים הכל בחוט ראשי בלבד.