**מגישים:**

רון כהן – 209792340

עמית פלד – 314778903

**שאלה 1:**

ההבדל הוא ש thread.start()**יוצרת חוט במערכת ההפעלה** שיריץ את המתודה run(), ואילו thread.run() רק מפעילה את המתודה בחוט הנוכחי, ללא יצירת חוט חדש.

**שאלה 2:**

1.

"

Hello world from thread number 1

Hello world from thread number 4

Hello world from thread number 3

Hello world from thread number 0

Hello world from thread number 2

Hello world from thread number 7

Hello world from thread number 6

Hello world from thread number 5

Hello world from thread number 8

Hello world from thread number 9

That's all, folks

"

2. אם יורדו השורות המדוברות, לא תהיה המתנה לסיום חוטים ולכן הדפסת הפלט מהMain (That's all, folks) לא בהכרח יתרחש לאחר הדפסת/סיום פעולות החוטים. זאת בניגוד להתרחשות בסעיף הקודם כי המשך הרצת החוט הראשי לא יהיה תלוי בסיום החוטים האחרים.  
דוגמא לפלט שיצא:

Hello world from thread number 0

That's all, folks

Hello world from thread number 1

Hello world from thread number 3

Hello world from thread number 2

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

3. בגלל שלאחר כל קריאת start ויצירת החוט והרצתו אנחנו עושים join ומחכים לסיומו הקוד יתבצע בסדר יצירת החוטים. הקוד כבר לא יתבצע בצורה מקבילית למרות שהמשימות יתבצעו בחוטים שונים.  
דוגמא לפלט שיצא:

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

4. במצב זה יתרחש deadlock, כיוון שהחוט מחכה שהוא עצמו יסיים את ביצועו על מנת שיוכל לסיים את ביצועו ולכן הוא יהיה תמיד במצב . במילים אחרות על מנת להתבצע הוא צריך לסיים את ביצועו ולכן יש תלות מעגלית. בגלל שהחוט הראשי גם עושה join לכל החוטים האחרים לפני ההדפסה שלו ושהחוט המדובר אף פעם לא יסיים, יתקע גם החוט הראשי ולא יודפס That's all, folks.

**שאלה 3:**

1. כן, יצא שהקוד המקבילי רץ הרבה יותר מהר מהקוד התורי. בגלל שלמעבד יש מספר cores שעובדים במקביל, יצא שהקוד המקבילי רץ יותר מהר כי הוא פעל על מספר cores בעוד שהקוד התורי פעל רק על אחד.
2. \*\*\*זמן ריצת התוכנית אינו קבוע מהסיבה שהריצה אינה דטרמיניסטית ותלוייה בגורמים נוספים כמו תזמון המעבד (בדומה לכפי שראינו קודם לכן, למרות שהחוטים נקראים על פי הסדר הם לאו דווקא מבוצעים על פיו), תוכנות נוספות שהמעבד מפעיל וכמות הcontex switching.

**שאלה 4:**

הקוד גם מצורף כקובץ java

import java.util.\*;  
  
public class ProducerConsumer2 {  
 Queue<Integer> workingQueue = new LinkedList<Integer>();  
 static int *QUEUE\_LIMIT*=10;  
 public synchronized void produce(int num) throws InterruptedException {  
 while(workingQueue.size()>=*QUEUE\_LIMIT*)  
 {  
 wait();  
 }  
 workingQueue.add(num);  
 notifyAll();  
 }  
  
 public synchronized Integer consume() throws InterruptedException {  
 while (workingQueue.isEmpty()) {  
 wait();  
 }  
 notifyAll(); // it's okay to notify before polling the element because the producers will wait for this::lock to be unlock (what's going to happen after the poll) before checking the queue size  
 return workingQueue.poll();  
 }  
}

**שאלה 5 (תרגיל מעבדה 2):**

4. נוכל לדעת על ידי הפרש הגדול בזמני הפעולה של: הפעלת מספר חוטים (למשל 10 שמיתוכם 9 יצרנים) לעומת הפעלת התוכנית עם 2 חוטים (כלומר בנאי אחד). עבור Max=7^10 ל10 חוטים לקח: 7 שניות, לעומת זאת ל2 לקח: 13 שניות.

5. לא, בגלל שהחוטים פועלים במקביל סדר ההכנסה למערך-סדר ההדפסה, הוא משתנה ולא קבוע. לעומת זאת ההדפסה לאחר המיון כן קבוע.