**מעבדה 2–**

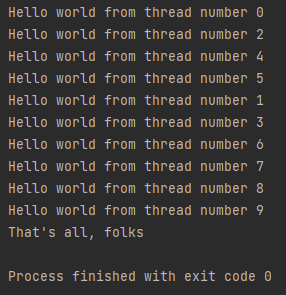
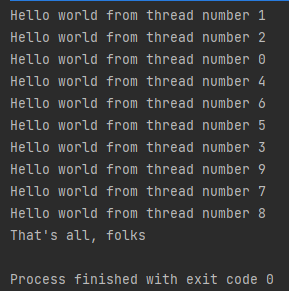
**מגישים:**   
אבישי עוז – 326489069  
ליאון גורין – 214511214

**שאלה 1:**   
ההבדל בין שתי המתודות הוא שבמתודה thread.start() נוצר thread חדש ועליו מורץ המתודה run לאומת thread.run() שפשוט מריץ על החוט הנוכחי כלומר:

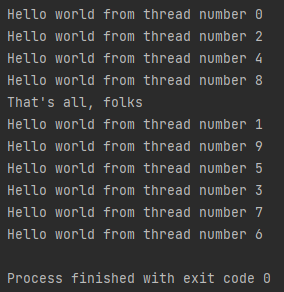
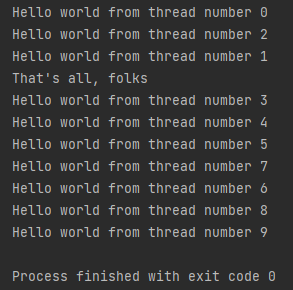
* thread.start() מייצרת חוט חדש ומריצה את המתודה run() (ריצה מקבילית)
* thread.run() לא מייצרת חוט חדש, ומריצה את המתודה run() בחוט הנוכחי (לא ריצה מקבילית)

**שאלה 2:**

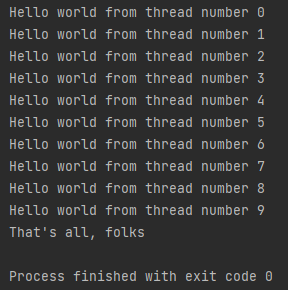
1. פלט התוכנית (2 הרצות שונות)



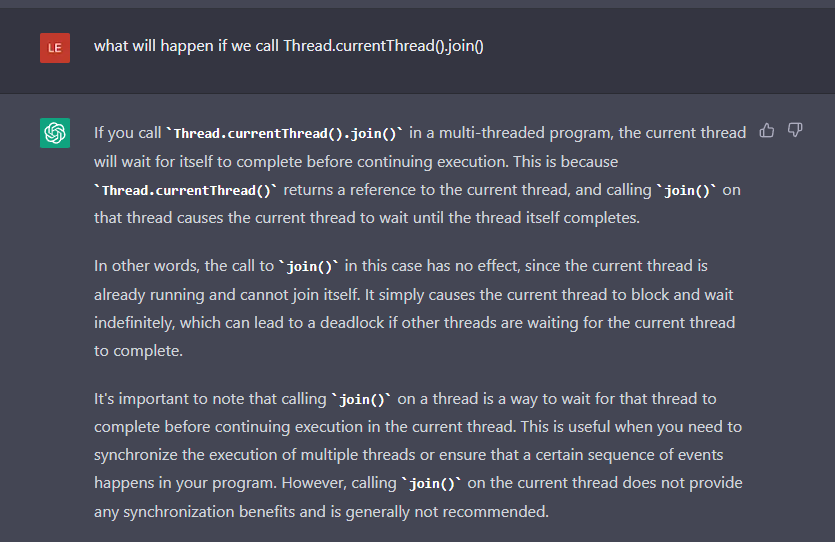
1. אם נעשה "קומנט" ללולאה השלישית נקבל (2 הרצות שונות)   
   נשים לב שכעט לא מחכים שכל החוטים יסיימו על מנת להדפיס את ההודעה " "That's all, folks ולכן ההודעה תודפס כל פעם במקום אחר כאשר החוט הראשי יגיע לשם.



1. אם נוסיף thread.join לאחר thread.start() נקבל:



הסבר: כעט כל פעם שהחוט הראשי (main) יצור חוט חדש במתודה thread.start() הוא יחכה שהחוט שיצר יסיים את תפקידו לפני שיצור את החוט הבא לכן נקבל את אופן ההדפסה המוצג מעלה ובנוסף בעקבות מימוש זה נעבד את תכונת המקביליות שאנחנו מנשים להשיג.



**שאלה 3:**

1. כן, כאשר החישוב מתבצע על מספר חוטים החישוב נעשה באופן "מקבילי" ועבודת החישוב מתחלקת בין החוטים מה שמקטין את זמן הריצה (ואכן ניתן לראות שזמן הריצה מהיר יותר במימוש עם מספר חוטים)
2. לא, בכל פעם שנריץ את התוכנה חלוקת משימות החישוב בין החוטים תשתנה ובעקבות זה גם זמן החישוב ישתנה אך ניתן להניח שזמן החישוב ינוע סביב ערכים דומים.

package org.example;  
  
public class SumThreads extends Thread {  
 public static final long *tenthOfSumNum* = 4294967296L / 10; // 2^32 / 10  
 private int partialSum;  
 public SumThreads(int n){  
 partialSum = n;  
 }  
 public int retSum(){  
 return partialSum;  
 }  
 @Override  
 public void run() {  
 partialSum = 0;  
 while(partialSum < *tenthOfSumNum*){  
 partialSum += 1;  
 }  
 }  
 public static void main(String[] args) {  
 SumThreads[] threads = new SumThreads[10]; // create an array of threads  
 long sum = 0;  
 for (int i = 0; i < 10; i++) {  
 threads[i] = new SumThreads(0);  
 threads[i].start();  
 }  
 for (SumThreads thread : threads) {  
 try {  
 thread.join(); // wait for the threads to terminate  
 sum += thread.retSum();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println("The sum (multithreaded) is: " + (sum+7));  
 }  
}

**מטלה 4:**