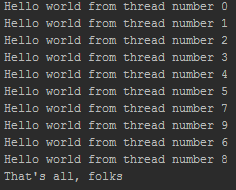
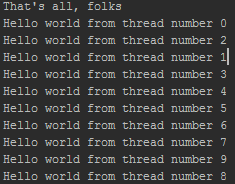
מעבדה 3-

## שאלה 1

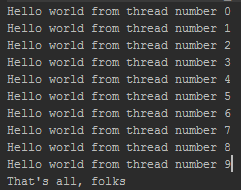
ההבדל הוא ש-thread.start() מתחיל thread חדש ומריץ את run() על הט'רד החדש וthread.run() קורא לפונקציה run() ומריץ אותה מהthread הנוכחי.

## שאלה 2

א)

ב)

זה קורה כיוון שהט'רד הראשי לא מחכה ששאר הט'רדים יסיימו ולפני שהם מספיצים להריץ את הקוד שלהם הוא מסיים להריץ את הלולאה השלישית.

ג)

במקום להריץ את כל הט'רד ולתת למערכת הפעלה להחליט מתי כל ט'רד ירוץ, אנחנו מריצים כל ט'רד ומחכים שיסיים לפני ההרצת הט'רד הבא, התוצאה היא הדפסות לפי הסדר.

ד) לולאה אין סופית, כיוון שאם הט'רד הנוכחי יחכה זמן בלתי מוגבל עד שיסיים לפני שימשיך, הוא לא יוכל להמשיך ולסיים.

## שאלה 3

1. יש יתרון כיוון שבמחשבים המודרניים יש בד"כ יותר מליבה אחת ואפשר לנצל זאת בעיקר לחישובים שלוקחים הרבה זמן מעבד ומעט גישות לזכרון, כמו סכימה של מספרים.
2. משך הזמן עם חוטים הוא לא דטרמיניסטי באופן כללי, מרווחי הזמן שבדקנו בתוכנית הספיציפית שלנו שתמיד יוצא 0 שניות הוא כן קבוע.

import java.util.Calendar;  
  
public class SumThreadsMain {  
 public static void main(String[] args) {  
  
 Calendar cal = Calendar.*getInstance*();  
 int second1, second2;  
 second1 = cal.get(Calendar.*SECOND*);  
 int sumNormal = *sum*();  
 second2 = cal.get(Calendar.*SECOND*);  
 System.*out*.println("normal sum: " + sumNormal + " time it took in sec: " + (second2 - second1));  
  
 second1 = cal.get(Calendar.*SECOND*);  
 Thread[] threads = new Thread[10]; // create an array of threads  
 int[] arr = new int[10];  
 for(int i = 1; i <= 10000; i+=1000) {  
 String threadName = Integer.*toString*(i);  
 threads[i/1000]=new Thread(new SumThreads(i, i+999, arr, i/1000),threadName);  
 // create threads  
 }  
 for (Thread thread : threads) {  
 thread.start(); // start the threads  
 }  
 for (Thread thread : threads) {  
 try {  
 thread.join(); // wait for the threads to terminate  
  
 }  
 catch (InterruptedException e)  
 {  
 e.printStackTrace();  
 }  
 }  
 int sum\_threads = 0;  
 for (int i = 0 ; i < 10; i++)  
 sum\_threads += arr[i];  
 second2 = cal.get(Calendar.*SECOND*);  
 System.*out*.println("threaded sum: " + sum\_threads + " time it took in sec: " + (second2 - second1));  
 }  
  
 static int sum(){  
 int sum = 0;  
 for (int i = 1; i <= 10000; i++) {  
 sum+=i;  
 }  
 return sum;  
 }  
}  
  
class SumThreads implements Runnable {  
 int startIndex, endIndex, index;  
 int[] arr;  
 SumThreads(int startIndex, int endIndex, int[] arr, int index){  
 this.startIndex = startIndex;  
 this.endIndex = endIndex;  
 this.index = index;  
 this.arr = arr;  
 }  
 public void run() {  
 int sum = 0;  
 for (int i = startIndex; i<= endIndex; i++)  
 sum +=i;  
 arr[index] = sum;  
 }  
}



## שאלה 4

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

## תרגיל מעבדה

import java.util.Scanner;  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingQueue;  
  
public class FibThreads {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 int n = scanner.nextInt();  
 int lineLength = scanner.nextInt();  
 int arrLength = scanner.nextInt();  
 BlockingQueue<Integer> workingQueue = new LinkedBlockingQueue<>(arrLength);  
 new Thread(new Producer(workingQueue, n)).start();  
 new Thread(new Consumer(workingQueue, n, lineLength)).start();  
 }  
}  
  
class Producer implements Runnable {  
 private final BlockingQueue<Integer> queue;  
 private final int numElements;  
 Producer(BlockingQueue<Integer> q, int n) { queue = q; numElements = n; }  
 public void run() {  
 try {  
 for(int i = 1; i <= numElements; i ++) {  
 queue.put(fib(i));  
 }  
 } catch (InterruptedException ignored) { }  
 }  
 int fib(int n) {  
 if (n <= 2)  
 return 1;  
 return fib(n-1) + fib(n-2);  
 }  
}  
  
class Consumer implements Runnable {  
 private final BlockingQueue<Integer> queue;  
 private final int lineLength, numElements;  
 Consumer(BlockingQueue<Integer> q, int n, int lineLength) { queue = q; this.lineLength = lineLength; numElements = n; }  
 public void run() {  
 for(int i = 1; i <= numElements; i++) {  
 int val = 0;  
 try {  
 val = queue.take();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.*out*.print(val + ((i % lineLength) !=0 ? " " : "\n" ));  
 }  
 }  
}

תמיד מתקבל אותו פלט כיוון שאנחנו עוברים באופן סדרתי ומכניסים ומוציאים איברים עם ט'רד אחד לכל תפקיד, לא נוצר מצב שבו ט'רד שני יוצר מספר יותר מתקדם, אז הסדר תמיד זהה.