# Многопоточность в Java 2

# Задача

Код ниже состоит из двух частей:

1- Полный код, который пытается продемонстрировать разницу во времени выполнения при использовании многопоточности и без нее. Код используется для вычисления значений многочлена для большого массива входных данных.

2- Код, который вам нужно заполнить, цель которого - вычислить сумму всех элементов в массиве (pX), который содержит результаты первой части.

Ваша задача — выполнить вторую часть и понять код в целом. Вы должны быть в состоянии ответить на любой вопрос, касающийся кода в целом.

|  |
| --- |
| package org.pool;  import java.util.ArrayList; import java.util.Random; import java.util.concurrent.\*;  */\*\*  \* A class to handle polynomials: pX=a[0]+a[1]\*x+a[2]\*x^2.....+a[n]\*x^n  \*/* class PolynomialCalculator {  */\*\*  \* Caculates the value of a polynomial for a given x  \** ***@param*** *a An array containing polynomial coefficients  \** ***@param*** *x Value to substitute in the polynomial  \** ***@return*** *The value of the polynomial for a given x  \*/* public static double polynomialCalc(double[] a,double x)  {  double pX=0;  for(int j=0;j<a.length;j++)  {  pX+=a[j]\*Math.pow(x,j);  }  return pX;  } }  */\*\*  \* A class for creating threads to calculate the result of substituting  \* a vector of values in a polynomial.  \*/* class VectorPolynomialCalculator implements Runnable {  */\*\*  \* Synchronization mechanism between threads  \*/* private CountDownLatch latch;   */\*\*  \* Polynomial coefficients  \*/* private final double[] a;   */\*\*  \* Values to substitute in the polynomial  \*/* private final double [] x;   */\*\*  \* Polynomial values  \*/* private final double[] pX;   */\*\*  \* first index in array x from which calculations begin.  \*/* private final int start;   */\*\*  \* first index in array x at which calculations end.  \*/* private final int end;   */\*\*  \*  \** ***@param*** *a Constructor  \** ***@param*** *x  \** ***@param*** *pX  \** ***@param*** *start  \** ***@param*** *end  \*/* VectorPolynomialCalculator(double [] a, double [] x, double [] pX, int start, int end,CountDownLatch latch)  {  this.a =a;  this.x = x;  this.pX = pX;  this.start=start;  this.end=end;  this.latch=latch;  }   */\*\*  \* This is where actual work happens  \*/* @Override  public void run() {  for(int i=start;i<=end;i++)  {  pX[i]=PolynomialCalculator.polynomialCalc(a, x[i]);  }  latch.countDown();  } }   public class ThreadPoolExample {   */\*\*  \* size of array x and pX  \*/* public static final int sizeOfArray=10000000;  */\*\*  \* how many elements one thread should handle  \*/* public static final int batch =1000000;   */\*\*  \* input array  \*/* public static double [] x =new double[sizeOfArray];   */\*\*  \* result array  \*/* public static double [] pX =new double[sizeOfArray];   public static void main(String []args) throws ExecutionException, InterruptedException {  Random r=new Random();  //initialization  for(int i=0;i<sizeOfArray;i++)  {  x[i]= r.nextDouble();  }  double [] a=new double[20];  //initialization  for(int i=0;i<a.length;i++)  {  a[i]= r.nextInt();  }  long startTime,stopTime;  /\*Solution without multithreading\*/  startTime= System.nanoTime();  for(int i=0;i<sizeOfArray;i++)  {  pX[i]= PolynomialCalculator.polynomialCalc(a, x[i]);  }  stopTime = System.nanoTime();  System.out.println("elapsed time: "+(stopTime-startTime)/1000+" microseconds");  startTime= System.nanoTime();   /\*Solving using multithreading\*/  CountDownLatch latch1=new CountDownLatch(sizeOfArray/ batch);  ExecutorService executorService = Executors.newFixedThreadPool(sizeOfArray/ batch);  for (int i = 0; i<sizeOfArray/ batch; i++)  {  executorService.submit(new VectorPolynomialCalculator(a, x, pX,i\* batch,(i+1)\* batch -1,latch1));  }  latch1.await();  stopTime = System.nanoTime();  System.out.println("elapsed time: "+(stopTime-startTime)/1000+" microseconds");   /\* YOUR TASK \*/  /\* Please add your code wherever you find a comment with todo comment \*/  /\* The ultimate goal is to find the sum of all elements in array "pX" \*/   double total=0;  startTime= System.nanoTime();   //TODO: calculate the sum of all elements in in array output using "for"  //......................   stopTime = System.nanoTime();  //TODO: calculate execution time and output it in console  //......................   total=0;  startTime= System.nanoTime();  ArrayList<Future<Double>> list = new ArrayList<>();  for (int i = 0; i<sizeOfArray/ batch; i++)  {  //TODO: create threads using interface Callable  //Hint: Callable<Double> callable=........  //TODO: create Future<Double> objects to store results asynchronously  //Hint: Future<Double> future=........  //TODO: store results in the list  //.................  }   //calculate final result  for (Future<Double> future : list) {  //TODO: calculate final result.  //total += ....  }  stopTime = System.nanoTime();  //TODO: calculate execution time and output it in console  //......................   } }  class ArraySum implements Callable<Double> {  private final double[] array;  private final int start,end;   //TODO: add an appropriate constructor  //..............................   //TODO: override function call  //..............................  }  } |