|  |  |  |
| --- | --- | --- |
| **OSAMA MOHAMMED AFZAL** | **CS-7B** | **237529** |

|  |
| --- |
| **TASK (A)** |
| //package com.mycompany.dc\_lab1;  import java.time.Clock;  /\*\*   \*   \* @author OSAMA   \*/  public class TaskA  {        public static void main(String[] args){            final long startTime = System.currentTimeMillis();          double x,pi, sum = 0.0;          double num\_of\_steps = 100000;            for(int i=0; i< num\_of\_steps;i++){              x = Math.pow(-1,i);              sum = sum + (x/(1+(2\*i)));          }            pi = 4\*sum;          System.out.println("PI "+pi);          final long endTime = System.currentTimeMillis() - startTime;          System.out.println("PI calculation took: "+endTime+" microseconds to run");      }  } |

|  |
| --- |
| **TASK (B)** |
| //package com.mycompany.dc\_lab1;  import java.time.Clock;  /\*\*   \*   \* @author OSAMA   \*/  public class TaskB {      public static void main(String[] args) throws InterruptedException {          for(int i = 2; i<=10;i++){              double num\_of\_threads = i;              calcTimeToExec(num\_of\_threads);          }      }      public static void calcTimeToExec(double num\_of\_threads) throws InterruptedException {          final int iterationsPerThread = (int) (100000 / num\_of\_threads);          final long startTime = System.currentTimeMillis();          MyThread[] sums = new MyThread[(int) num\_of\_threads];          for (int k = 1; k <= num\_of\_threads; k++) {              MyThread t1 = new MyThread(0 + ((k - 1) \* iterationsPerThread), iterationsPerThread + (iterationsPerThread \* (k - 1)));              sums[k - 1] = t1;              sums[k - 1].start();          }          for (int i = 0; i < 2; i++) {              sums[i].join();          }          double total = 0.0;          for (int k = 0; k < num\_of\_threads; k++) {              total = total + sums[k].getSum();          }          double pi = 4 \* total;          System.out.println("NUM\_OF\_THREADS = "+num\_of\_threads);          System.out.println("PI " + pi);          final long endTime = System.currentTimeMillis() - startTime;          System.out.println("PI calculation took: " + endTime + " microseconds to run\n\n");        }  }  class MyThread extends Thread {      private double sum;      private int start;      private int stop;      public MyThread(int start, int stop) {  *this*.start = start;  *this*.stop = stop;      }      @Override      public void run() {          for (int i = *this*.start; i < *this*.stop; i++) {              double x = Math.pow(-1, i);  *this*.sum = *this*.sum + (x / (1 + (2 \* i)));          }      }      public double getSum() {          return *this*.sum;      }  }    There are 2 main trends observable in the above data.   1. Multithreading is way better as compared to single thread. A massive difference of 15 microseconds shows how powerful multithreading is. 2. Increasing the number of threads decreases the time. There is a consistent decrease in time as the number of threads decrease. 2 threads take 11 microseconds to execute whereas 10 threads take 2 microseconds to execute. |