Kpi-best

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. І.СІКОРСЬКОГО»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №2**

з дисципліни «Паралельні та розподілені обчислення»

на тему: «Потоки в мові Java»

Виконала:

студентка 3-го курсу

факультету ІОТ

групи ІО-43

Даніленко Н.

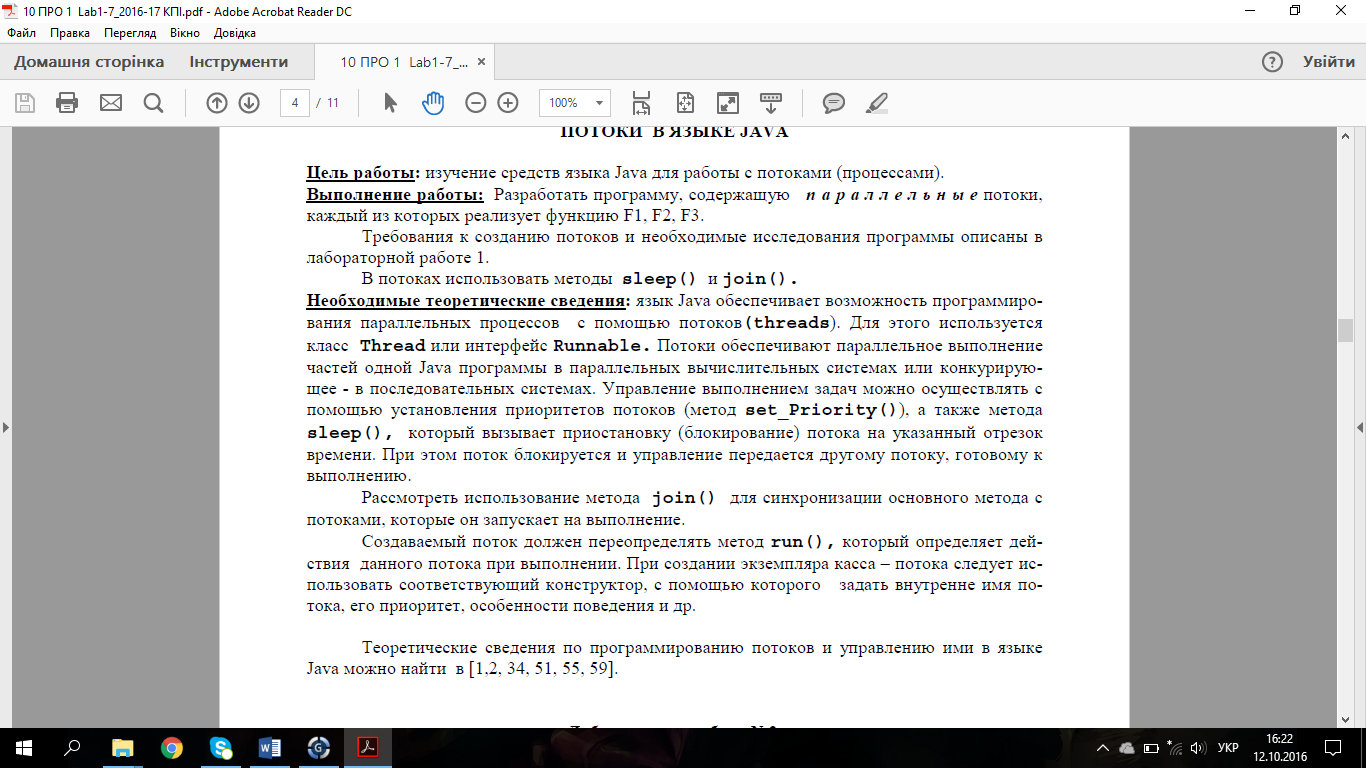
Перевірив:

доц.

Долголенко О. М.

Київ 2016

**Потоки в мові Java**



**Варіант 4**

1.4 C = A + SORT(B) \* (MA \* ME).

2.4 MG = MAX(MH) \* (MK \* ML).

3.4 O = SORT(P) \* SORT (MR \* MS).

**Лістинг:**

package pro\_lab2;

import java.util.Scanner;

/\*\*

\*Parallel and distributed computing.

\*Labwork 2. Threads in Java

\*Danilenko Natalia

\*IO-43

\*1.4 C = A + SORT(B) \* (MA \* ME).

\*2.4 MG = MAX(MH) \* (MK \* ML).

\*3.4 O = SORT(P) \* SORT (MR \* MS).

\*/

public class Lab2 {

@SuppressWarnings("resource")

public static void main(String[] args) throws InterruptedException{

System.*out*.println("Main thread started");

System.*out*.println("Enter n: ");

Scanner scanner = new Scanner(System.*in*);

int n = scanner.nextInt();

T1 t1 = new T1(n, 100, 2, "T1");

T2 t2 = new T2(n, 1000, 5, "T2");

T3 t3 = new T3(n, 1500, 8, "T3");

t1.start();

t2.start();

t3.start();

t1.join();

t2.join();

t3.join();

System.*out*.println("Main thread finished");

}

}

private double vectA[];

private double vectB[];

private double matrA[][];

private double matrE[][];

private int n;

private int sleep;

/\*\*

\* @param n кількість елементів

\* @param sleep час затримки

\* @param priority пріоритет потоку

\* @param name назва потоку

\*/

public T1(int n, int sleep, int priority, String name){

this.setName(name);

this.setPriority(priority);

this.n = n;

this.sleep = sleep;

}

/\* (non-Javadoc)

\* @see java.lang.Thread#run()

\*/

public void run(){

double vectRes[];

System.*out*.println(this.getName() +" started");

vectA = Tools.*readVect*(n, "Enter vector A for F1");

vectB = Tools.*readVect*(n, "Enter vector B for F1");

matrA = Tools.*readMatr*(n, "Enter matrix A for F1");

matrE = Tools.*readMatr*(n, "Enter matrix Z for F1");

try {

Thread.*sleep*(sleep);

} catch (Exception e) {

}

vectRes = Tools.*addVect*(vectA, (Tools.*multVectMatr*((Tools.*sortVect*(vectB)), (Tools.*multMatr*( matrA, matrE)))));

Tools.*showVect*(vectRes, "Result of F1");

System.*out*.println(this.getName() + " finished");

}

}

package pro\_lab2;

/\*\*

\*Parallel and distributed computing.

\*Labwork 2. Threads in Java

\*Danilenko Natalia

\*IO-43

\*1.4 C = A + SORT(B) \* (MA \* ME).

\*2.4 MG = MAX(MH) \* (MK \* ML).

\*3.4 O = SORT(P) \* SORT (MR \* MS).

\*/

public class T2 extends Thread{

private double matrH[][];

private double matrK[][];

private double matrL[][];

private int n;

private int sleep;

/\*\*

\* @param n кількість елементів

\* @param sleep час затримки

\* @param priority пріоритет потоку

\* @param name назва потоку

\*/

public T2(int n, int sleep, int priority, String name){

this.setName(name);

this.setPriority(priority);

this.n = n;

this.sleep = sleep;

}

/\* (non-Javadoc)

\* @see java.lang.Thread#run()

\*/

public void run(){

double matrRes[][];

System.*out*.println(this.getName() +" started");

matrH = Tools.*readMatr*(n, "Enter matrix A for F2");

matrK = Tools.*readMatr*(n, "Enter matrix B for F2");

matrL = Tools.*readMatr*(n, "Enter matrix M for F2");

try {

Thread.*sleep*(sleep);

} catch (Exception e) {

// TODO: handle exception

}

matrRes = Tools.*multNumbMatr*((Tools.*findMaxMatr*(matrH)), (Tools.*multMatr*(matrK, matrL)));

Tools.*showMatr*(matrRes, "Result of F2");

System.*out*.println(this.getName() + " finished");

}

}

package pro\_lab2;

/\*\*

\*Parallel and distributed computing.

\*Labwork 2. Threads in Java

\*Danilenko Natalia

\*IO-43

\*1.4 C = A + SORT(B) \* (MA \* ME).

\*2.4 MG = MAX(MH) \* (MK \* ML).

\*3.4 O = SORT(P) \* SORT (MR \* MS).

\*/

public class T3 extends Thread{

private double vectP[];

private double matrR[][];

private double matrS[][];

private int n;

private int sleep;

/\*\*

\* @param n кількість елементів

\* @param sleep час затримки

\* @param priority пріоритет потоку

\* @param name назва потоку

\*/

public T3(int n, int sleep, int priority, String name){

this.setName(name);

this.setPriority(priority);

this.n = n;

this.sleep = sleep;

}

/\* (non-Javadoc)

\* @see java.lang.Thread#run()

\*/

public void run(){

double vectRes1[];

System.*out*.println(this.getName() + " started");

vectP = Tools.*readVect*(n, "Enter vector A for F3");

matrR = Tools.*readMatr*(n, "Enter matrix B for F3");

matrS = Tools.*readMatr*(n, "Enter matrix M for F3");

try {

Thread.*sleep*(sleep);

} catch (Exception e) {

// TODO: handle exception

}

vectRes1 = Tools.*multVectMatr*((Tools.*sortVect*(vectP)),(Tools.*sortMatr*((Tools.*multMatr*(matrR, matrS)))));

System.*out*.println("Result of F3: " + vectRes1);

System.*out*.println(this.getName() + " finished");

}

}

package pro\_lab2;

import java.io.InputStreamReader;

import java.util.Scanner;

/\*\*

\*Parallel and distributed computing.

\*Labwork 2. Threads in Java

\*Danilenko Natalia

\*IO-43

\*1.4 C = A + SORT(B) \* (MA \* ME).

\*2.4 MG = MAX(MH) \* (MK \* ML).

\*3.4 O = SORT(P) \* SORT (MR \* MS).

\*/

public class Tools {

/\*\*

\* Додає два вектори

\* @param vectA вектор

\* @param vectB вектор

\* @return вектор суми

\*/

public static double[] addVect(double vectA[], double vectB[]){

for (int i = 0; i < vectB.length; i++) {

vectA[i] += vectB[i];

}

return vectA;

}

/\*\*

\* Перемножає дві матриці

\* @param matrA матриця

\* @param matrB матриця

\* @return повертає добуток двох матриць

\*/

public static double[][] multMatr(double matrA[][], double matrB[][]){

double temp = 0;

double resultMatr[][] = new double[matrA.length][matrA[1].length];

for (int i = 0; i < matrA.length; i++) {

for (int j = 0; j < matrA[i].length; j++) {

temp = 0;

for (int k = 0; k < matrB.length; k++) {

temp += matrA[i][k] \* matrB[k][j];

}

resultMatr[i][j] = temp;

}

}

return resultMatr;

}

/\*\*

\* Знаходить максимальний елемент в матриці

\* @param matrA матриця для пошуку

\* @return максимальний елемент матриці

\*/

public static double findMaxMatr(double matrA[][]){

double max = matrA[0][0];

for (int i = 0; i < matrA.length; i++) {

for (int j = 0; j < matrA.length; j++) {

if(max < matrA[i][j]){

max = matrA[i][j];

}

}

}

return max;

}

/\*\*

\* Множить матрицю на число

\* @param a число

\* @param matrB матриця

\* @return результат множення

\*/

public static double[][] multNumbMatr(double a, double matrB[][]){

for (int i = 0; i < matrB.length; i++) {

for (int j = 0; j < matrB[0].length; j++) {

matrB[i][j] = matrB[i][j]\*a;

}

}

return matrB;

}

/\*\*

\* Знаходить мінімальний елемент в векторі

\* @param vectA вектор для пошуку

\* @return мінімальний елемент

\*/

public static double findMinVect(double vectA[]){

double min = vectA[0];

for (int i = 0; i < vectA.length; i++) {

if(min > vectA[i]){

min = vectA[i];

}

}

return min;

}

/\*\*

\* Перемножає вектор на матрицю

\* @param vectA вектор

\* @param matrB матрицю

\* @return добуток вектора і матриці

\*/

public static double[] multVectMatr(double vectA[], double matrB[][]){

double temp = 0;

double resultVect[] = new double[vectA.length];

for (int i = 0; i < matrB.length; i++) {

temp = 0;

for (int j = 0; j < resultVect.length; j++) {

temp += vectA[j]\*matrB[j][i];

}

resultVect[i] = temp;

}

return resultVect;

}

/\*\*

\* Сортує вектор за зростанням

\* @param vectA вектор для сортування

\* @return відсортований вектор

\*/

public static double[] sortVect(double vectA[]){

double temp = 0;

for (int i = 0; i < vectA.length; i++) {

for (int j = 0; j < (i-1); j++) {

if(vectA[j] > vectA[j+1]){

temp = vectA[j];

vectA[j] = vectA[j+1];

vectA[j+1] = temp;

}

}

}

return vectA;

}

/\*\*

\* Сортує матрицю за спаданням

\* @param matrA матриця для сортування

\* @return відсортовану матрицю

\*/

public static double[][] sortMatr(double matrA[][]){

double temp = 0;

for (int k = 0; k < matrA.length; k++) {

for (int i = 0; i < matrA.length; i++) {

for (int j = 0; j < (i-1); j++) {

if(matrA[k][j] < matrA[k][j+1]){

temp = matrA[k][j];

matrA[k][j] = matrA[k][j+1];

matrA[k][j+1] = temp;

}

}

}

}

return matrA;

}

/\*\*

\* Зчитує матрицю

\* @param n кількість елементів

\* @param message повідомлення для відображення

\* @return зчитану матрицю

\*/

public static synchronized double[][] readMatr( int n, String message){

double matr[][] = new double[n][n];

System.out.println(message);

Scanner reader = new Scanner(System.in);

for (int i = 0; i < matr.length; i++) {

for (int j = 0; j < matr.length; j++) {

// matr[i][j] = reader.nextDouble();

matr[i][j] = 1;

}

}

return matr;

}

/\*\*

\* Відображає матрицю

\* @param matr матриця для відображення

\* @param message повідомлення для відображення

\*/

public static synchronized void showMatr(double matr[][], String message){

System.out.println(message);

for (int i = 0; i < matr.length; i++) {

for (int j = 0; j < matr[i].length; j++) {

System.out.print(matr[i][j] + " ");

}

System.out.println();

}

System.out.println();

}

/\*\*

\* Зчитує вектор

\* @param n кількість елементів вектора

\* @param message повідомлення для відображення

\* @return зчитаний вектор

\*/

public static synchronized double[] readVect(int n, String message){

double vect[] = new double [n];

Scanner reader = new Scanner(System.in);

System.out.println(message);

for (int i = 0; i < vect.length; i++) {

// vect[i] = reader.nextDouble();

vect[i] = 1;

}

return vect;

}

/\*\*

\* Відображає вектор

\* @param vect вектор для відображення

\* @param message повідомлення для відображення

\*/

public static synchronized void showVect(double vect[], String message){

System.out.println(message);

for (int i = 0; i < vect.length; i++) {

System.out.print(vect[i] + " ");

}

System.out.println();

}

}

**Результати виконання програми:**

