МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

КАФЕДРА 44

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

доц.		Н.В.Кучин
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

ПРОГРАММИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

по дисциплине: операционные системы

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	B0441		Г.П. Андриенко
	номер группы	подпись, дата	инициалы, фамилия

Студенческий билет № 20190126

Санкт-Петербург 2023

1. Описание задания

Вариант: 1 – задание 1

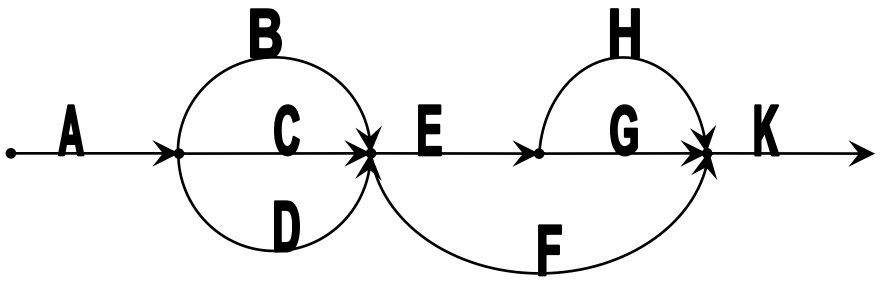

Рисунок 1. Граф задач по 1 варианту

Имя задачи	Длительность	Приоритет	Функция
A	1	0	Генерирует M[1..n, 1..n], R [1..n] - integer
B	1	1	f_1 (M, R)
C	1	1	f_2 (M, R)
D	1	1	f_3 (M, R)
E	1	2	$f_4(f_1, f_2, f_3)$
F	2	3	$f_5(f_1, f_2, f_3)$
G	1	3	$f_6(f_4)$
H	1	3	$f_7(f_4)$
K	1	4	$f_8(f_5, f_6, f_7)$

Таблица 1. Характеристики задач по варианту

Выбранные функции для реализации:

Задача A – генерирует матрицу и массив со случайными числами от 0 до 50

f_1 – добавление ко всем элементам матрицы соответствующий их ряду элемент массива

f_2 – добавление ко всем элементам матрицы соответствующий их колонке элемент массива

f_3 – добавление ко всем элементам матрицы среднего значения массива

f_4 – сложение матриц всех функций

f_5 – сложение матриц функций f1 и f2 и вычитание из них удвоенной матрицы функции f3

f_6 – уменьшает элементы матрицы в 10 раз, отбрасывая остаток

f_7 – записывает в матрицу остаток от деления элементов на 10

f_8 – сложение матриц всех функций

2. Описание используемых методов

Выбранный язык программирования – java

Используемые библиотеки: javax.swing.JFrame, javax.swing.JPanel, javax.swing.border.LineBorder, java.awt.Color, java.awt.Graphics

Задачи представлены как отдельные потоки (наследующие интерфейс Runnable), уже содержащие в себе свои функции. Задачу A вызывает конструктор, остальные вызываются по цепочке друг от друга. Для удобства добавлены функции вывода массивов и матриц в строку. Для создания графического интерфейса используется отдельный поток, ожидающий выполнения всех задач и отображающий процесс их выполнения при помощи инструментов swing и awt библиотек. Отдельно вынесен класс MyPanel наследующий JPanel с переопределенным методом paintComponent, что позволило построить нужный рисунок для интерфейса и дополнять его по мере выполнения задач.

3. Текст программы

Содержимое Main:

```
public class Main {
   public static void main(String[] args) {
   ThreadManager tm = new ThreadManager(10);
   }
}
```

Содержимое ThreadManager:

```
import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.border.LineBorder;

import java.awt.Color;

import java.awt.Graphics;


public class ThreadManager {

   public static final int GlobalTime = 150; //добавочное временя к задачам, необходимое для более наглядной демонстрации поэтапной работы алгоритма

   public static final long TimeStart = System.currentTimeMillis(); //начальное время, от которого берется отсчет

   public static boolean[] flag = {false,false,false}; //аналоги семафор

   public static boolean[] tasks = new boolean[9]; //массив статусов выполнения задач, нужен для их визуализации в реальном времени в интерфейсе

public ThreadManager(int n){

   Thread thr_a = new Thread(new task_a(n)); //вызов задачи A

   thr_a.start();

   Thread thr_draw = new Thread(new task_draw()); //вызов потока, ответственного за создание и обновление интерфейса

   thr_draw.start();
```

```
}

public ThreadManager(){} //пустой конструктор, нужен для доступа к методам класса
    private static class task_draw implements Runnable { //поток вывода интерфейса
        public task_draw(){}
        @Override
        public void run() {
            boolean end = true;
            JFrame a = new JFrame("tasks");
            a.setSize(1000,500);
            a.setVisible(true);
            a.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            while(end){ //будет работать, пока не завершится последний из процессов
                if(tasks[0]||tasks[1]||tasks[2]||tasks[3]||tasks[4]||tasks[5]||tasks[6]||tasks[7]){
                    a.add(new MyPanel());
                    a.repaint();
                }
                if(tasks[8]){
                    a.add(new MyPanel());
                    a.repaint();
                    end = false;
                }
            }
        }
    }

    public static class MyPanel extends JPanel{ //класс для создания графического интерфейса
        public MyPanel(){setBorder(new LineBorder(Color.RED, 5));}
        public void paintComponent(Graphics g){
            super.paintComponent(g);
            g.drawRect(50, 50, 100, 30);
            g.drawLine(150,65,200,65);
            g.drawLine(150,65,200,100);
            g.drawLine(150,65,200,30);
            g.drawRect(200, 15, 100, 30);
            g.drawRect(200, 50, 100, 30);
```

```
g.drawRect(200, 85, 100, 30);
g.drawLine(300,65,350,65);
g.drawLine(300,30,350,65);
g.drawLine(300,100,350,65);
g.drawLine(350,65,400,65);
g.drawLine(350,65,475,100);
g.drawRect(400, 50, 100, 30);
g.drawRect(475, 85, 100, 30);
g.drawLine(500,65,550,65);
g.drawLine(500,65,550,30);
g.drawRect(550, 15, 100, 30);
g.drawRect(550, 50, 100, 30);
g.drawLine(650,65,700,65);
g.drawLine(650,30,700,65);
g.drawLine(575,100,700,65);
g.drawRect(700, 50, 100, 30);
if(tasks[0]) {
   g.drawString("Task A complete",55,70);
}
if(tasks[1]) {
   g.drawString("Task B complete",205,35);
}
if(tasks[2]) {
   g.drawString("Task C complete",205,70);
}
if(tasks[3]) {
   g.drawString("Task D complete",205,105);
}
if(tasks[4]) {
   g.drawString("Task E complete",405,70);
}
if(tasks[5]) {
   g.drawString("Task F complete",480,105);
}
if(tasks[6]) {
   g.drawString("Task G complete",555,70);
```

```
            }
            if(tasks[7]) {
                g.drawString("Task H complete",555,35);
            }
            if(tasks[8]) {
                g.drawString("Task K complete",705,70);
            }
        }
    }
private static class task_a implements Runnable { //задача A

    public int[][] M;
    public int[] R;
    public int n;
    public task_a(int n){
        this.n = n;
    }

    @Override
    public void run(){
        ThreadManager tm = new ThreadManager(); //для обращения к методам этого класса
        long tn = System.currentTimeMillis() - TimeStart;
        System.out.println("время начала выполнения задачи A: " + tn);
        try {
            Thread.sleep(GlobalTime + ((int) (Math.random() * GlobalTime/2))); //имитация долгой работы функции, для наглядности их переключения, с небольшой вариативностью
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        M = new int[n][n];
        R = new int[n];
        for(int i = 0; i < n; ++i) { //заполнение матрицы и массива случайными числами до 50
            for(int j = 0; j < n; ++j) {
                M[i][j] = (int)(Math.random() * 50);
            }
            R[i] = (int)(Math.random() * 50);
        }
```

```
        tn = System.currentTimeMillis() - TimeStart; //определение прошедшего времени с начала работы алгоритма

        System.out.println("результат выполнения задачи A:\n" + "построенная матрица:\n" + tm.toString(M, n) + "построенный массив:\n" + tm.toString(R, n) + "\n" + "время окончания выполнения задачи A: " + tn);

        System.out.println("начинается инициация задачи B, ее инициировала задача A");

        Thread thr_b = new Thread(new task_b(M, R)); //запуски дальнейших потоков

        thr_b.start();

        System.out.println("начинается инициация задачи C, ее инициировала задача A");

        Thread thr_c = new Thread(new task_c(M, R));

        thr_c.start();

        System.out.println("начинается инициация задачи D, ее инициировала задача A");

        Thread thr_d = new Thread(new task_d(M, R));

        thr_d.start();

        tasks[0] = true; //указание для интерфейса о готовности данного процесса

    }

}

private static class task_b implements Runnable {

    public static int[][] M;

    public int[] R;

    public int n;

    public task_b(int[][] Ma, int[] R){

        n = R.length;

        M = new int[n][n];

        for(int i = 0; i < n; ++i) {

            System.arraycopy(Ma[i], 0, M[i], 0, n); //создаются копии матриц, чтобы результат не зависил от случаной очередности выполнения задач

        }

        this.R = R;

    }

    @Override

    public void run(){

        ThreadManager tm = new ThreadManager();

        long tn = System.currentTimeMillis() - TimeStart;

        System.out.println("время начала выполнения задачи B: " + tn);

        try {

            Thread.sleep(GlobalTime + ((int) (Math.random() * GlobalTime/2)));

        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
        for(int i = 0; i < n; ++i) { //f1:добавление ко всем элементам матрицы соответствующий их ряду элемент массива
            for(int j = 0; j < n; ++j) {
                M[i][j] += R[i];
            }
        }
        tn = System.currentTimeMillis() - TimeStart;
        System.out.println("результат выполнения задачи B:\nновая матрица:\n" + tm.toString(M, n) + "время окончания выполнения задачи B: " + tn);
        flag[0] = true;
        if (flag[1] && flag[2]){ //проверка на то, является ли этот поток последним из выполнившихся
            System.out.println("начинается инициация задачи E, ее инициировала задача B");
            Thread thr_e = new Thread(new task_e(M,task_c.M,task_d.M));
            thr_e.start();
            System.out.println("начинается инициация задачи F, ее инициировала задача B");
            Thread thr_f = new Thread(new task_f(M,task_c.M,task_d.M));
            thr_f.start();
        }
        tasks[1] = true;
    }
}
    private static class task_c implements Runnable { //аналогичен потоку b
        public static int[][] M;
        public int[] R;
        public int n;
        public task_c(int[][] Ma, int[] R){
            n = R.length;
            M = new int[n][n];
            for(int i = 0; i < n; ++i) {
                System.arraycopy(Ma[i], 0, M[i], 0, n);
            }
            this.R = R;
        }
        @Override
        public void run(){
```

```
        ThreadManager tm = new ThreadManager();
        long tn = System.currentTimeMillis() - TimeStart;
        System.out.println("время начала выполнения задачи C: " + tn);
        try {
            Thread.sleep(GlobalTime + ((int) (Math.random() * GlobalTime/2)));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        for(int i = 0; i < n; ++i) { //f2:добавление ко всем элементам матрицы соответствующий их колонке элемент массива
            for(int j = 0; j < n; ++j) {
                M[i][j] += R[j];
            }
        }
        tn = System.currentTimeMillis() - TimeStart;
        System.out.println("результат выполнения задачи C:\nновая матрица:\n" + tm.toString(M, n) + "время окончания выполнения задачи C: " + tn);
        flag[1] = true;
        if (flag[0] && flag[2]){
            System.out.println("начинается инициация задачи E, ее инициировала задача C");
            Thread thr_e = new Thread(new task_e(task_b.M,M,task_d.M));
            thr_e.start();
            System.out.println("начинается инициация задачи F, ее инициировала задача C");
            Thread thr_f = new Thread(new task_f(task_b.M,M,task_d.M));
            thr_f.start();
        }
        tasks[2] = true;
    }
}
private static class task_d implements Runnable { //аналогичен потоку b
    public static int[][] M;
    public int[] R;
    public int n;
    public task_d(int[][] Ma, int[] R){
        n = R.length;
        M = new int[n][n];
        for(int i = 0; i < n; ++i) {
```

```
            System.arraycopy(Ma[i], 0, M[i], 0, n);
        }
        this.R = R;
    }
    @Override
    public void run(){
        ThreadManager tm = new ThreadManager();
        long tn = System.currentTimeMillis() - TimeStart;
        System.out.println("время начала выполнения задачи D: " + tn);
        try {
            Thread.sleep(GlobalTime + ((int) (Math.random() * GlobalTime/2)));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        int average = 0;
        for(int i = 0; i < n; ++i){average+=R[i];}
        average = average/n;
        for(int i = 0; i < n; ++i) { //f3:добавление ко всем элементам матрицы среднего значения
массива
            for(int j = 0; j < n; ++j) {
                M[i][j] += average;
            }
        }
        tn = System.currentTimeMillis() - TimeStart;
        System.out.println("результат выполнения задачи D:\nновая матрица:\n" + tm.toString(M, n) +
"время окончания выполнения задачи D: " + tn);
        flag[2] = true;
        if (flag[0] && flag[1]){
            System.out.println("начинается инициация задачи E, ее инициировала задача D");
            Thread thr_e = new Thread(new task_e(task_b.M,task_c.M,M));
            thr_e.start();
            System.out.println("начинается инициация задачи F, ее инициировала задача D");
            Thread thr_f = new Thread(new task_f(task_b.M,task_c.M,M));
            thr_f.start();
        }
        tasks[3] = true;
    }
```

```
}
private static class task_e implements Runnable {
    public static int[][] Mb;
    public static int[][] Mc;
    public static int[][] Md;
    public int n;
    public task_e(int[][] Mb_,int[][] Mc_, int[][] Md_){
        n = Mb_.length;
        Mb = new int[n][n];
        Mc = new int[n][n];
        Md = new int[n][n];
        for(int i = 0; i < n; ++i) {
            System.arraycopy(Mb_[i], 0, Mb[i], 0, n);
            System.arraycopy(Mc_[i], 0, Mc[i], 0, n);
            System.arraycopy(Md_[i], 0, Md[i], 0, n);
        }
        flag[0]=false;
        flag[1]=false;
        flag[2]=false;
    }
    @Override
    public void run(){
        ThreadManager tm = new ThreadManager();
        long tn = System.currentTimeMillis() - TimeStart;
        System.out.println("время начала выполнения задачи E: " + tn);
        try {
            Thread.sleep(GlobalTime + ((int) (Math.random() * GlobalTime/2)));
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        for(int i = 0; i < n; ++i) { //f4:сложение матриц
            for(int j = 0; j < n; ++j) {
                Mb[i][j] = Mb[i][j] + Mc[i][j] + Md[i][j];
            }
        }
        tn = System.currentTimeMillis() - TimeStart;
```

```
            System.out.println("результат выполнения задачи E:\nСуммирующая матрица:\n" +
tm.toString(Mb, n) + "время окончания выполнения задачи E: " + tn);
            System.out.println("начинается инициация задачи G, ее инициировала задача E");
            Thread thr_g = new Thread(new task_g(Mb));
            thr_g.start();
            System.out.println("начинается инициация задачи H, ее инициировала задача E");
            Thread thr_h = new Thread(new task_h(Mb));
            thr_h.start();
            tasks[4] = true;
        }
    }
    private static class task_f implements Runnable {
        public static int[][] Mb;
        public static int[][] Mc;
        public static int[][] Md;
        public int n;
        public task_f(int[][] Mb_,int[][] Mc_, int[][] Md_){
            n = Mb_.length;
            Mb = new int[n][n];
            Mc = new int[n][n];
            Md = new int[n][n];
            for(int i = 0; i < n; ++i) {
                System.arraycopy(Mb_[i], 0, Mb[i], 0, n);
                System.arraycopy(Mc_[i], 0, Mc[i], 0, n);
                System.arraycopy(Md_[i], 0, Md[i], 0, n);
            }
        }
        @Override
        public void run(){
            ThreadManager tm = new ThreadManager();
            long tn = System.currentTimeMillis() - TimeStart;
            System.out.println("время начала выполнения задачи F: " + tn);
            try {
                Thread.sleep(GlobalTime*2 + ((int) (Math.random() * GlobalTime))); //удвоено среднее
время выполнения (ожидания) задачи, что требовало условие варианта
            } catch (InterruptedException e) {
                e.printStackTrace();
```

```
            }
            for(int i = 0; i < n; ++i) { //f5:сложение матриц функций f1 и f2 и вычитание из них удвоенной матрицы функции f3
                for(int j = 0; j < n; ++j) {
                    Mb[i][j] = Mb[i][j] + Mc[i][j] - (Md[i][j] * 2);
                }
            }
            tn = System.currentTimeMillis() - TimeStart;
            System.out.println("результат выполнения задачи F:\nСуммирующая матрица:\n" + tm.toString(Mb, n) + "время окончания выполнения задачи F: " + tn);
            flag[2] = true;
            if (flag[0] && flag[1]){
                System.out.println("начинается инициация задачи K, ее инициировала задача F");
                Thread thr_k = new Thread(new task_k(Mb,task_g.M,task_h.M));
                thr_k.start();
            }
            tasks[5] = true;
        }
    }
    private static class task_g implements Runnable {
        public static int[][] M;
        public int n;
        public task_g(int[][] Me){
            n = Me.length;
            M = new int[n][n];
            for(int i = 0; i < n; ++i) {
                System.arraycopy(Me[i], 0, M[i], 0, n);
            }
        }
        @Override
        public void run(){
            ThreadManager tm = new ThreadManager();
            long tn = System.currentTimeMillis() - TimeStart;
            System.out.println("время начала выполнения задачи G: " + tn);
            try {
                Thread.sleep(GlobalTime + ((int) (Math.random() * GlobalTime/2)));
            } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
        for(int i = 0; i < n; ++i) { //f6:уменьшает элементы матрицы в 10 раз, отбрасывая остаток
            for(int j = 0; j < n; ++j) {
                M[i][j] = M[i][j]/10;
            }
        }
        tn = System.currentTimeMillis() - TimeStart;
        System.out.println("результат выполнения задачи G:\nСуммирующая матрица:\n" +
tm.toString(M, n) + "время окончания выполнения задачи G: " + tn);
        flag[0] = true;
        if (flag[1] && flag[2]){
            System.out.println("начинается инициация задачи K, ее инициировала задача G");
            Thread thr_k = new Thread(new task_k(task_f.Mb,M,task_h.M));
            thr_k.start();
        }
        tasks[6] = true;
    }
  }
  private static class task_h implements Runnable {
    public static int[][] M;
    public int n;
    public task_h(int[][] Me){
        n = Me.length;
        M = new int[n][n];
        for(int i = 0; i < n; ++i) {
            System.arraycopy(Me[i], 0, M[i], 0, n);
        }
    }
    @Override
    public void run(){
        ThreadManager tm = new ThreadManager();
        long tn = System.currentTimeMillis() - TimeStart;
        System.out.println("время начала выполнения задачи H: " + tn);
        try {
            Thread.sleep(GlobalTime + ((int) (Math.random() * GlobalTime/2)));
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
        for(int i = 0; i < n; ++i) { //f6:записывает в матрицу остаток от деления элементов на 10
            for(int j = 0; j < n; ++j) {
                M[i][j] = M[i][j]%10;
            }
        }
        tn = System.currentTimeMillis() - TimeStart;
        System.out.println("результат выполнения задачи H:\nСуммирующая матрица:\n" +
tm.toString(M, n) + "время окончания выполнения задачи H: " + tn);
        flag[1] = true;
        if (flag[0] && flag[2]){
            System.out.println("начинается инициация задачи K, ее инициировала задача H");
            Thread thr_k = new Thread(new task_k(task_f.Mb,task_g.M,M));
            thr_k.start();
        }
        tasks[7] = true;
    }
  }
  private static class task_k implements Runnable {
    public static int[][] Mf;
    public static int[][] Mg;
    public static int[][] Mh;
    public int n;
    public task_k(int[][] Mf_,int[][] Mg_, int[][] Mh_){
        n = Mf_.length;
        Mf = new int[n][n];
        Mg = new int[n][n];
        Mh = new int[n][n];
        for(int i = 0; i < n; ++i) {
            System.arraycopy(Mf_[i], 0, Mf[i], 0, n);
            System.arraycopy(Mg_[i], 0, Mg[i], 0, n);
            System.arraycopy(Mh_[i], 0, Mh[i], 0, n);
        }
    }
    @Override
    public void run(){
```

```
            ThreadManager tm = new ThreadManager();
            long tn = System.currentTimeMillis() - TimeStart;
            System.out.println("время начала выполнения задачи K: " + tn);
            try {
               Thread.sleep(GlobalTime + ((int) (Math.random() * GlobalTime/2)));
            } catch (InterruptedException e) {
               e.printStackTrace();
            }
            for(int i = 0; i < n; ++i) { //f7:сложение матриц функций
               for(int j = 0; j < n; ++j) {
                  Mf[i][j] = Mf[i][j] + Mg[i][j] +Mh[i][j];
               }
            }
            tn = System.currentTimeMillis() - TimeStart;
            System.out.println("результат выполнения задачи K:\nСуммирующая матрица:\n" +
tm.toString(Mf, n) + "время окончания выполнения задачи K: " + tn);
            tasks[8] = true;
         }
      }
   public final String toString(int[] R, int n) { //создание строк для вывода массивов
      StringBuilder build = new StringBuilder();
      for(int j = 0; j < n; ++j) {build.append(R[j]).append(" ");}
      return build.toString();
   }
   public final String toString(int[][] M, int n) { //создание строк для вывода матриц
      StringBuilder build = new StringBuilder();
      for(int i = 0; i < n; ++i) {
         for(int j = 0; j < n; ++j) {build.append(M[i][j]).append(" ");}
         build.append("\n");
      }
      return build.toString();
   }
   }
```

4. Протокол выполнения задач

Сгенерированный вывод:

время начала выполнения задачи A: 1

результат выполнения задачи A:

построенная матрица:

48 27 20 20 4 3 35 1 22 18

48 14 29 10 30 32 34 26 33 24

32 3 25 12 17 4 40 19 27 6

17 1 35 20 14 14 13 20 4 39

37 11 32 41 17 29 31 20 27 33

3 35 33 13 37 45 44 30 44 44

4 16 4 2 44 29 43 32 15 28

17 49 21 26 28 37 31 15 42 31

11 17 15 40 34 27 26 34 1 22

39 29 42 7 11 14 27 37 29 17

построенный массив:

2 32 29 40 14 49 42 9 31 19

время окончания выполнения задачи A: 218

начинается инициация задачи B, ее инициировала задача A

начинается инициация задачи C, ее инициировала задача A

время начала выполнения задачи B: 228

начинается инициация задачи D, ее инициировала задача A

время начала выполнения задачи C: 229

время начала выполнения задачи D: 230

результат выполнения задачи C:

новая матрица:

50 59 49 60 18 52 77 10 53 37

50 46 58 50 44 81 76 35 64 43

34 35 54 52 31 53 82 28 58 25

19 33 64 60 28 63 55 29 35 58

39 43 61 81 31 78 73 29 58 52

5 67 62 53 51 94 86 39 75 63

6 48 33 42 58 78 85 41 46 47

19 81 50 66 42 86 73 24 73 50

13 49 44 80 48 76 68 43 32 41

41 61 71 47 25 63 69 46 60 36

время окончания выполнения задачи C: 404

результат выполнения задачи D:

новая матрица:

74 53 46 46 30 29 61 27 48 44

74 40 55 36 56 58 60 52 59 50

58 29 51 38 43 30 66 45 53 32

43 27 61 46 40 40 39 46 30 65

63 37 58 67 43 55 57 46 53 59

29 61 59 39 63 71 70 56 70 70

30 42 30 28 70 55 69 58 41 54

43 75 47 52 54 63 57 41 68 57

37 43 41 66 60 53 52 60 27 48

65 55 68 33 37 40 53 63 55 43

время окончания выполнения задачи D: 407

результат выполнения задачи B:

новая матрица:

50 29 22 22 6 5 37 3 24 20

80 46 61 42 62 64 66 58 65 56

61 32 54 41 46 33 69 48 56 35

57 41 75 60 54 54 53 60 44 79

51 25 46 55 31 43 45 34 41 47

52 84 82 62 86 94 93 79 93 93

46 58 46 44 86 71 85 74 57 70

26 58 30 35 37 46 40 24 51 40

42 48 46 71 65 58 57 65 32 53

58 48 61 26 30 33 46 56 48 36

время окончания выполнения задачи B: 446

начинается инициация задачи E, ее инициировала задача B

начинается инициация задачи F, ее инициировала задача B

время начала выполнения задачи E: 447

время начала выполнения задачи F: 448

результат выполнения задачи E:

Суммирующая матрица:

174 141 117 128 54 86 175 40 125 101

204 132 174 128 162 203 202 145 188 149

153 96 159 131 120 116 217 121 167 92

119 101 200 166 122 157 147 135 109 202

153 105 165 203 105 176 175 109 152 158

86 212 203 154 200 259 249 174 238 226

82 148 109 114 214 204 239 173 144 171

88 214 127 153 133 195 170 89 192 147

92 140 131 217 173 187 177 168 91 142

164 164 200 106 92 136 168 165 163 115

время окончания выполнения задачи E: 664

начинается инициация задачи G, ее инициировала задача E

начинается инициация задачи H, ее инициировала задача E

время начала выполнения задачи G: 665

время начала выполнения задачи H: 666

результат выполнения задачи H:

Суммирующая матрица:

4 1 7 8 4 6 5 0 5 1

4 2 4 8 2 3 2 5 8 9

3 6 9 1 0 6 7 1 7 2

9 1 0 6 2 7 7 5 9 2

3 5 5 3 5 6 5 9 2 8

6 2 3 4 0 9 9 4 8 6

2 8 9 4 4 4 9 3 4 1

8 4 7 3 3 5 0 9 2 7

2 0 1 7 3 7 7 8 1 2

4 4 0 6 2 6 8 5 3 5

время окончания выполнения задачи H: 823

результат выполнения задачи F:

Суммирующая матрица:

-48 -18 -21 -10 -36 -1 -8 -41 -19 -31

-18 12 9 20 -6 29 22 -11 11 -1

-21 9 6 17 -9 26 19 -14 8 -4

-10 20 17 28 2 37 30 -3 19 7

-36 -6 -9 2 -24 11 4 -29 -7 -19

-1 29 26 37 11 46 39 6 28 16

-8 22 19 30 4 39 32 -1 21 9

-41 -11 -14 -3 -29 6 -1 -34 -12 -24

-19 11 8 19 -7 28 21 -12 10 -2

-31 -1 -4 7 -19 16 9 -24 -2 -14

время окончания выполнения задачи F: 863

результат выполнения задачи G:

Суммирующая матрица:

17 14 11 12 5 8 17 4 12 10

20 13 17 12 16 20 20 14 18 14

15 9 15 13 12 11 21 12 16 9

11 10 20 16 12 15 14 13 10 20

15 10 16 20 10 17 17 10 15 15

8 21 20 15 20 25 24 17 23 22

8 14 10 11 21 20 23 17 14 17

8 21 12 15 13 19 17 8 19 14

9 14 13 21 17 18 17 16 9 14

16 16 20 10 9 13 16 16 16 11

время окончания выполнения задачи G: 870

начинается инициация задачи K, ее инициировала задача G

время начала выполнения задачи K: 872

результат выполнения задачи K:

Суммирующая матрица:

-27 -3 -3 10 -27 13 14 -37 -2 -20

6 27 30 40 12 52 44 8 37 22

-3 24 30 31 3 43 47 -1 31 7

10 31 37 50 16 59 51 15 38 29

-18 9 12 25 -9 34 26 -10 10 4

13 52 49 56 31 80 72 27 59 44

2 44 38 45 29 63 64 19 39 27

-25 14 5 15 -13 30 16 -17 9 -3

-8 25 22 47 13 53 45 12 20 14

-11 19 16 23 -8 35 33 -3 17 2

время окончания выполнения задачи K: 1025

Скриншоты:

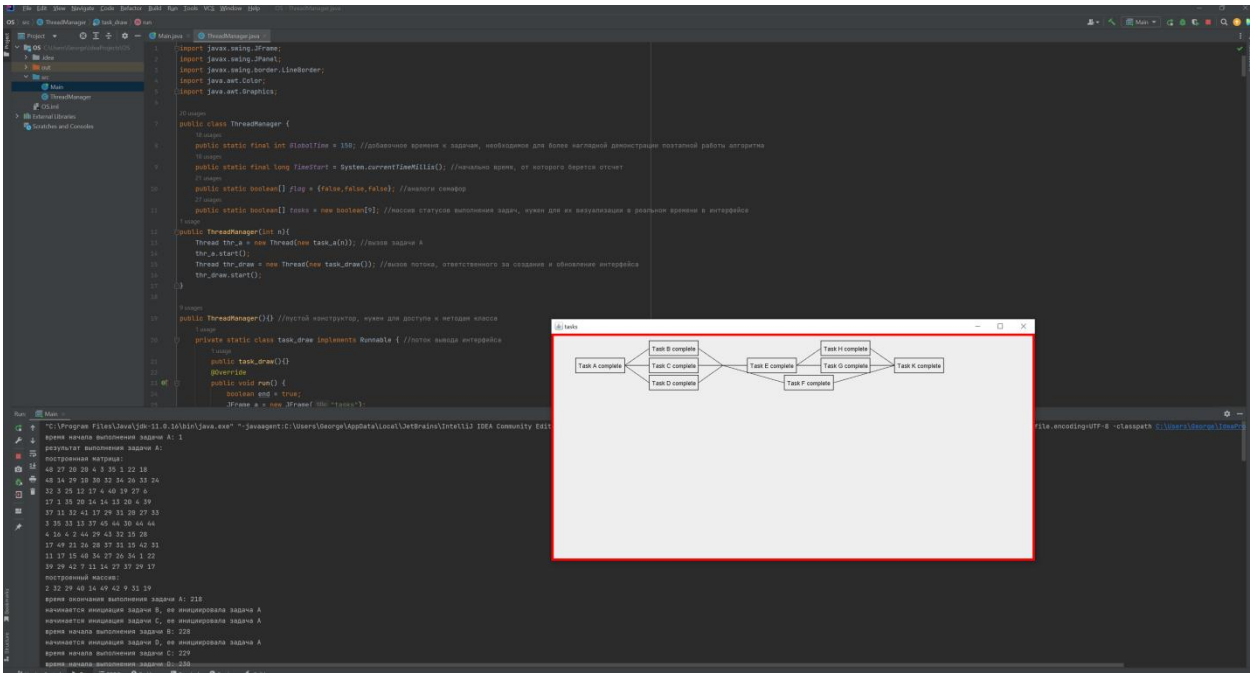

Рисунок 2. Окно запущенной программы

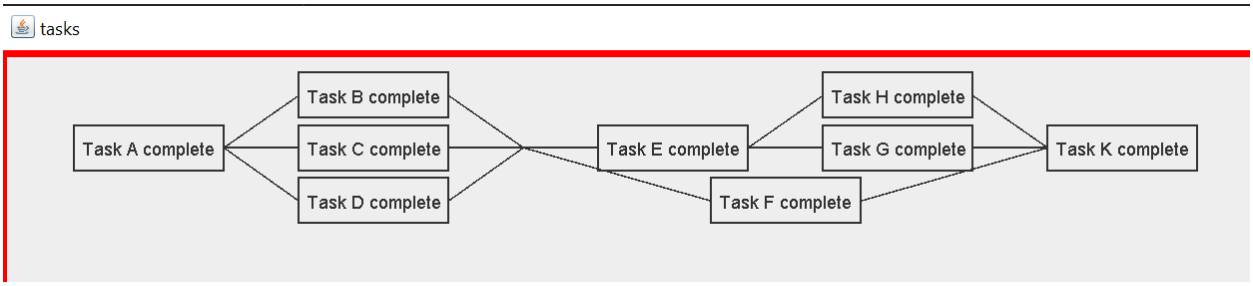

Рисунок 3. Графический интерфейс

5. Вывод

В результате работы получена программа показывающая работу параллельных процессов, выполняющая поставленные вариантом задачи в указанной последовательности.