Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по курсу «Программирование на Java»

Выполнили:

студенты группы 20ВВП1

Кирюткин И. А.

Кодиров И.Н.

Приняли:

Юрова О.В.

**Цель работы:** научиться создавать многопоточные приложения c использованием стандартных средств языка Java.

**Задание:** Модифицировать приложение из предыдущей лабораторной работы, реализовав вычисление определенного интеграла в нескольких дополнительных потоках (число потоков определяется номером варианта), снимая нагрузку с основного потока и предотвращая "подвисание" графического интерфейса. Варианты с номерами до 5 включительно реализуют многопоточность путем наследования от класса Thread, остальные реализуют интерфейс Runnable.

**Листинг:**

import java.io.Serializable;  
  
public class DataNumber implements Serializable {  
 private double Min;  
 private double Max;  
 private double Step;  
 private double Result;  
 private String Method;  
 private static String *Simpson*="метод Cимпcона";  
 private static String *Trap*="метод Трапеции";  
 public DataNumber(double min, double max, double step, double result) {  
 Min = min;  
 Max = max;  
 Step = step;  
 Result = result;  
 Method="";  
 }  
 public DataNumber() {  
 Min = 0;  
 Max = 0;  
 Step = 0;  
 Result = 0;  
 Method="";  
 }  
 public DataNumber(DataNumber dataNumber) {  
 this.Min = dataNumber.Min;  
 this.Max = dataNumber.Max;  
 this.Step = dataNumber.Step;  
 this.Result =dataNumber.Result;  
 this.Method= dataNumber.Method;  
 }  
 public DataNumber(double[] temp) throws InterruptedException {  
  
 Min = temp[0];  
 Max = temp[1];  
 Step = temp[2];  
 Result = Trap(temp[0],temp[1],temp[2]);  
 Method=*Trap*;  
  
 }  
 public double getMin() {  
 return Min;  
 }  
  
 public double getMax() {  
 return Max;  
 }  
  
 public double getStep() {  
 return Step;  
 }  
  
 public double getResult() {  
 return Result;  
 }  
  
 public void setMin(double min) {  
 Min = min;  
 }  
  
 public void setMax(double max) {  
 Max = max;  
 }  
  
 public void setStep(double step) {  
 Step = step;  
 }  
 public void setAllFieldTrap(double a, double b,double step) throws InterruptedException {  
 Min = a;  
 Max = b;  
 Step = step;  
 if(Min>Max)  
 {  
 setResultNull();  
 }else {  
 setResultTrap();  
 }  
 }  
 public void setAllField(double a, double b,double step,int method) throws InterruptedException {  
 Min = a;  
 Max = b;  
 Step = step;  
 if(Min>Max)  
 {  
 setResultNull();  
 };  
  
 if(method==0)  
 {  
 setResultTrap();  
 Method=*Trap*;  
 } else{ if(method==1) {  
 setResultSimpson();  
 Method=*Simpson*;  
 }else{  
 setResultNull();  
 Method="";  
 }}  
  
 }  
 public static String getSimpson() {  
 return *Simpson*;  
 }  
  
 public static void setSimpson(String simpson) {  
 *Simpson* = simpson;  
 }  
  
 public static String getTrap() {  
 return *Trap*;  
 }  
  
 public static void setTrap(String trap) {  
 *Trap* = trap;  
 }  
  
 public void setAllFieldSimpson(double a, double b, double step) throws InterruptedException {  
 Min = a;  
 Max = b;  
 Step = step;  
 if(Min>Max)  
 {  
 setResultNull();  
 }else {  
 setResultSimpson();  
 }  
 }  
  
 public void setResultTrap() throws InterruptedException {  
  
 Result = Trap(Min,Max,Step);  
 }  
  
 public void setResultSimpson() throws InterruptedException {  
 Result=Simpson(Min,Max,Step);  
 }  
 public void setResultNull() {  
 Result = 0;  
 }  
  
 public Object[] addModSimpson() {  
 Object[] temp=new Object[5];  
 temp[0]=Min;  
 temp[1]=Max;  
 temp[2]=Step;  
 temp[3]=Result;  
 temp[4]=*Simpson*;  
 return temp;  
 }  
 public Object[] addMod() {  
 Object[] temp=new Object[5];  
 temp[1]=Min;  
 temp[0]=Max;  
 temp[2]=Step;  
 temp[3]=Result;  
 temp[4]=Method;  
 return temp;  
 }  
  
 public double Trap(double a,double b, double h) throws InterruptedException {  
 final double[] result = {0};  
 int n = (int)((a-h- b) / h);  
 result[0] += (*InFunction*(a) + *InFunction*(b)) / 2;  
 int chunkSize = n / 7; // Размер частей  
 Thread[] threads = new Thread[7];  
 for (int i = 0; i < 7; i++) {  
 int startIndex = i \* chunkSize +1;  
 int endIndex = (i +1) \* chunkSize;  
 if (i == 6) {  
 endIndex = n;  
 }  
 int finalEndIndex = endIndex;  
 //////////////////////////////////////////////////////////////////  
 Runnable task = new Runnable() {  
 public void run() {  
 double localResult = 0;  
 for (int j = startIndex; j <= finalEndIndex; j++) {  
 localResult += *InFunction*(a + h \* j);  
 }  
  
 synchronized(this) {  
 result[0] += localResult;  
 }  
 }  
 };  
 ////////////////////////////////////////////////////////////////  
 threads[i] = new Thread(task);  
 threads[i].start();  
 // threads[i].join();  
 }  
 for (Thread thread : threads) {  
 try {  
 thread.join(); // Ждём завершения всех потоков  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 return h\*result[0];  
 }  
 public double Simpson(double a, double b, double n)throws InterruptedException{  
 final double[] result = {0};  
 //int i,  
 int z;  
 double h,s;  
 n=n+n;  
 s = *InFunction*(a)\**InFunction*(b);  
 h = (b-a)/n;  
 z = 4;  
 int chunkSize = (int)n / 7; // Размер частей  
 Thread[] threads = new Thread[7];  
 for (int k = 0; k < 7; k++) {  
 int startIndex = k \* chunkSize +1;  
 int endIndex = (k +1) \* chunkSize;  
 if (k == 6) {  
 endIndex = (int)n;  
 }  
 int finalEndIndex = endIndex;  
 //////////////////////////////////////////////////////////////////  
 Runnable task = new Runnable() {  
 public void run() {  
 double localResult = 0;  
 for(int i = startIndex; i<finalEndIndex; i++){  
 localResult =localResult + *InFunction*(a+i\*h);  
 //z = 6 - z;  
 }  
 synchronized(this) {  
 result[0] += localResult;  
 }  
 }  
 };  
 ////////////////////////////////////////////////////////////////  
 threads[k] = new Thread(task);  
 threads[k].start();  
 // threads[i].join();  
 }  
 for (Thread thread : threads) {  
 try {  
 thread.join(); // Ждём завершения всех потоков  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 return (result[0] \* h)/3;  
 }  
  
 public void setResult(double result) {  
 Result = result;  
 }  
  
 public String getMethod() {  
 return Method;  
 }  
  
 public void setMethod(String method) {  
 Method = method;  
 }  
  
 public static double InFunction(double x) //Подынтегральная функция  
 {  
 return Math.*sin*(Math.*pow*(x,2));  
 }  
 public void run() {  
 System.*out*.println("Thread LoaderDoc run!");  
 }  
 @Override  
 public String toString() {  
 return "DataNumber{" +  
 "Min=" + Min +  
 ", Max=" + Max +  
 ", Step=" + Step +  
 ", Result=" + Result +  
 ", Method='" + Method + '\'' +  
 '}';  
 }  
 public String toStringApi() {  
 return Min+","+Max+","+Step+","+Result+","+Method;  
 }  
}

**Результат работы программы:**

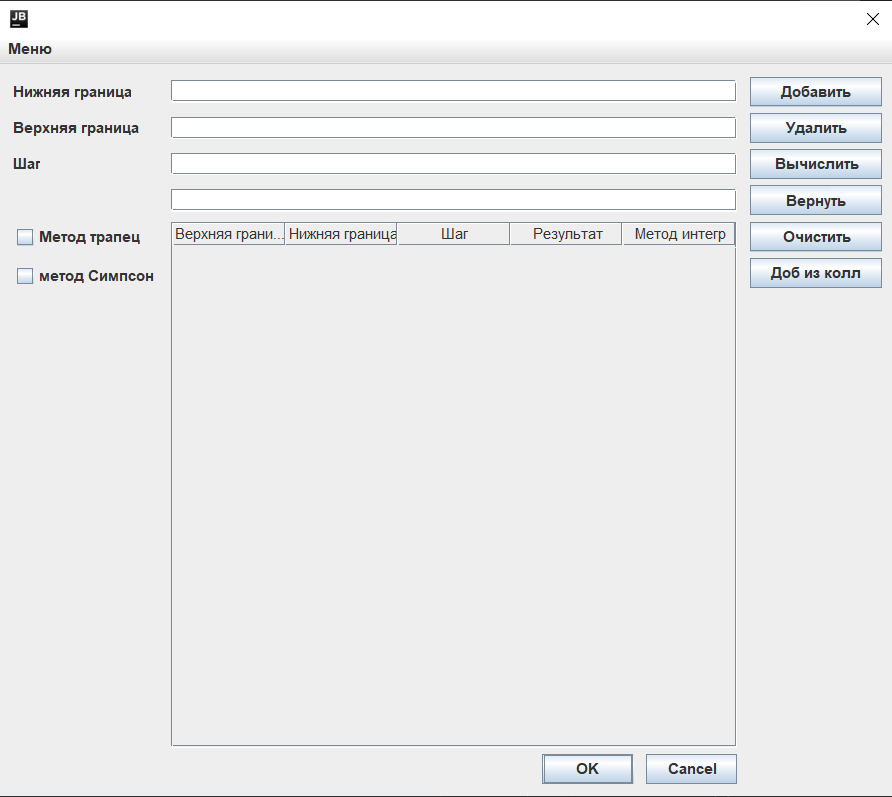


Рисунок 1 Программа

**Вывод:** научился создавать многопоточные приложения c использованием стандартных средств языка Java.