Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Программирование на Java»

Выполнили:

студенты группы 20ВВП1

Кирюткин И. А.

Кодиров И.Н.

Приняли:

Юрова О.В.

**Цель работы:** научиться создавать клиент-серверные приложения c использованием стандартных классов Java.

**Задание:** Модифицировать приложение из предыдущей лабораторной работы, реализовав клиент-серверную архитектуру, обеспечивающую распределенное вычисление определенного интеграла на нескольких вычислительных узлах (клиентах) при этом каждый узел использует несколько нитей,

**Результат работы программы:**

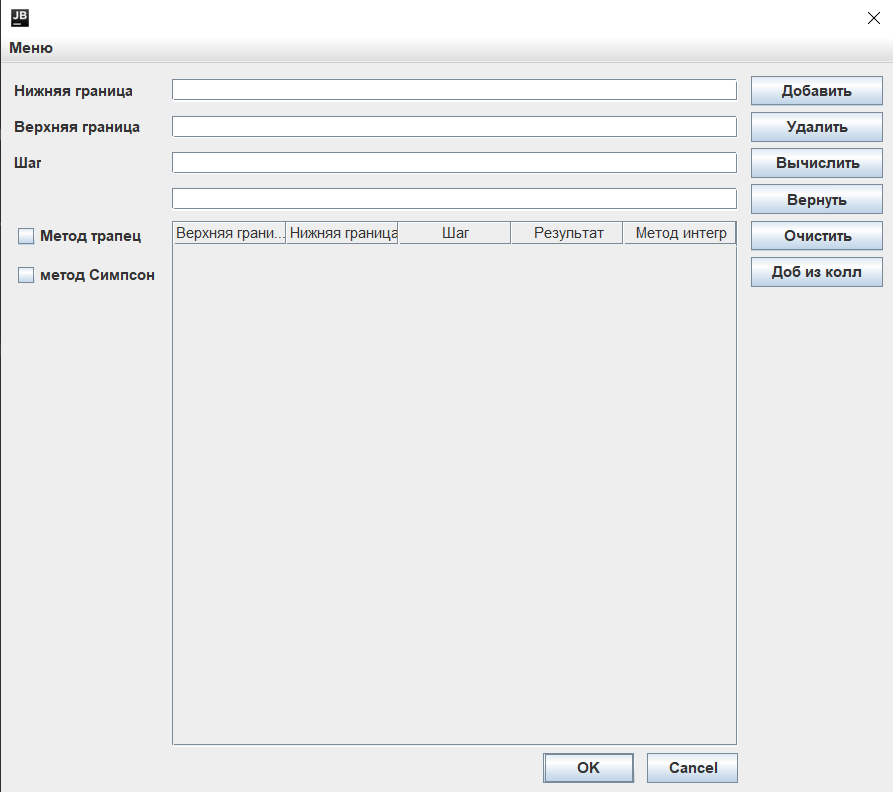


Рисунок 1 Программа

Листинг:

import java.io.Serializable;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

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;

}

public void SendData(int numberMethod) throws IOException {

DatagramSocket socket = new DatagramSocket();

byte[] data = (getMin()+","+getMax()+","+getStep()+","+numberMethod).getBytes();

InetAddress serverAddress = InetAddress.getByName("127.0.0.1");

int serverPort = 12345;

DatagramPacket packet = new DatagramPacket(data, data.length, serverAddress, serverPort);

socket.send(packet);

socket.close();

}

public double ReadData() throws IOException {

DatagramSocket socket = new DatagramSocket(12346);

byte[] buffer = new byte[1024];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

socket.receive(packet);

String message = new String(packet.getData(), 0, packet.getLength());

String response = new String(message);

socket.close();

return Double.valueOf(response);

}

public void runServer(int numberMethod) throws IOException, InterruptedException {

SendData(numberMethod);

setResult(ReadData());

}

}

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

public class Server {

private String[] stlArray;

ArrayList<Double> dbArrayd = new ArrayList<Double>();

int numberMethod;

double dataD;

public Server() {

}

public void runi() throws IOException, InterruptedException {

ReceiveRequest();

Date date = new Date();

SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss:SS");

System.out.println(formatter.format(date)+": Данные приняты");

dbArrayd.clear();

dbArrayd.add(Double.valueOf(stlArray[0]));

dbArrayd.add(Double.valueOf(stlArray[1]));

dbArrayd.add(Double.valueOf(stlArray[2]));

numberMethod=Integer.valueOf(stlArray[3]);

if(numberMethod==0)

{

System.out.println("Метод трапецмии");

Trap();

}else{

if(numberMethod==1)

{

System.out.println("Метод Симпсона");

Simpson();

}else{

System.out.println("Неправильный метод");

}

}

Date date2 = new Date();

SimpleDateFormat formatter2 = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss:SS");

System.out.println(formatter.format(date2)+": Интеграл вычислен");

SendResponse(dataD);

Date date1 = new Date();

SimpleDateFormat formatter1 = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss:SS");

System.out.println(formatter1.format(date1)+": Данные отправлены");

}

public void ReceiveRequest() throws IOException, IOException {

DatagramSocket socket = new DatagramSocket(12345);

byte[] buffer = new byte[1024];

DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

socket.receive(packet);

String message = new String(packet.getData(), 0, packet.getLength());

String requestData = new String(message);

socket.close();

stlArray = requestData.split(",");

}

public void SendResponse(double responseData) throws IOException {

DatagramSocket socket = new DatagramSocket();

byte[] data = String.valueOf(responseData).getBytes();

InetAddress clientAddress = InetAddress.getByName("127.0.0.1");

int clientPort = 12346;

DatagramPacket packet = new DatagramPacket(data, data.length, clientAddress, clientPort);

socket.send(packet);

socket.close();

}

static double InFunction(double x) //Подынтегральная функция

{

return 1/(Math.log(x));

}

public void Trap() throws InterruptedException {

final double[] result = {0};

int n = (int)((dbArrayd.get(0)-dbArrayd.get(2) - dbArrayd.get(1)) / dbArrayd.get(2));

result[0] += (InFunction(dbArrayd.get(0)) + InFunction(dbArrayd.get(1))) / 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(dbArrayd.get(1) + dbArrayd.get(2) \* j);

}

synchronized(this) {

result[0] += localResult;

}

}

};

threads[i] = new Thread(task);

threads[i].start();

// threads[i].join();

}

for (Thread thread : threads) {

try {

thread.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

dataD=dbArrayd.get(2) \* result[0];

}

public void Simpson()throws InterruptedException{

double a=dbArrayd.get(0);

double b=dbArrayd.get(1);

double n=dbArrayd.get(2);

double sum=0,sum2=0;

double[] x=new double[(int) n+1];

double h=(b-a)/n;

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

x[i]=a+i\*h;

}

for(int i=1;i<n;i++){

sum+=InFunction(x[i]);

sum2+=InFunction((x[i-1]+x[i])/2);

}

dataD=h/6\*(InFunction(a)+InFunction(b)+2\*sum+4\*(sum2+b));

/\*

final double[] result = {0};

System.out.println("+");

//int i,

int z;

double h,s;

n=n+n;

s = InFunction(a)\*InFunction(b);

h = (b-a)/n;

z = 4;

int chunkSize = (int)(a-b) / 7; // Размер частей

Thread[] threads = new Thread[7];

System.out.println("++");

for (int k = 0; k < 7; k++) {

System.out.println("++-");

int startIndex = k \* chunkSize +1;

int endIndex = (k +1) \* chunkSize;

if (k == 6) {

endIndex = (int)(b-a);

}

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);

System.out.println("+++");

//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();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

dataD=(result[0] \* h)/3;

\*/

}

}

**Вывод:** научился создавать клиент-серверные приложения c использованием стандартных классов Java