Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

Пенза 2021

**ОТЧЁТ**

по лабораторной работе №6

по курсу «Программирование на языке JAVA»

на тему «Сетевое взаимодействие в Java»

Выполнили:

студенты группы 21ВВП1

Брянцев А. Г.

Сущев М. В.

Тюрин В. И.

Приняли:

Юрова О.В.

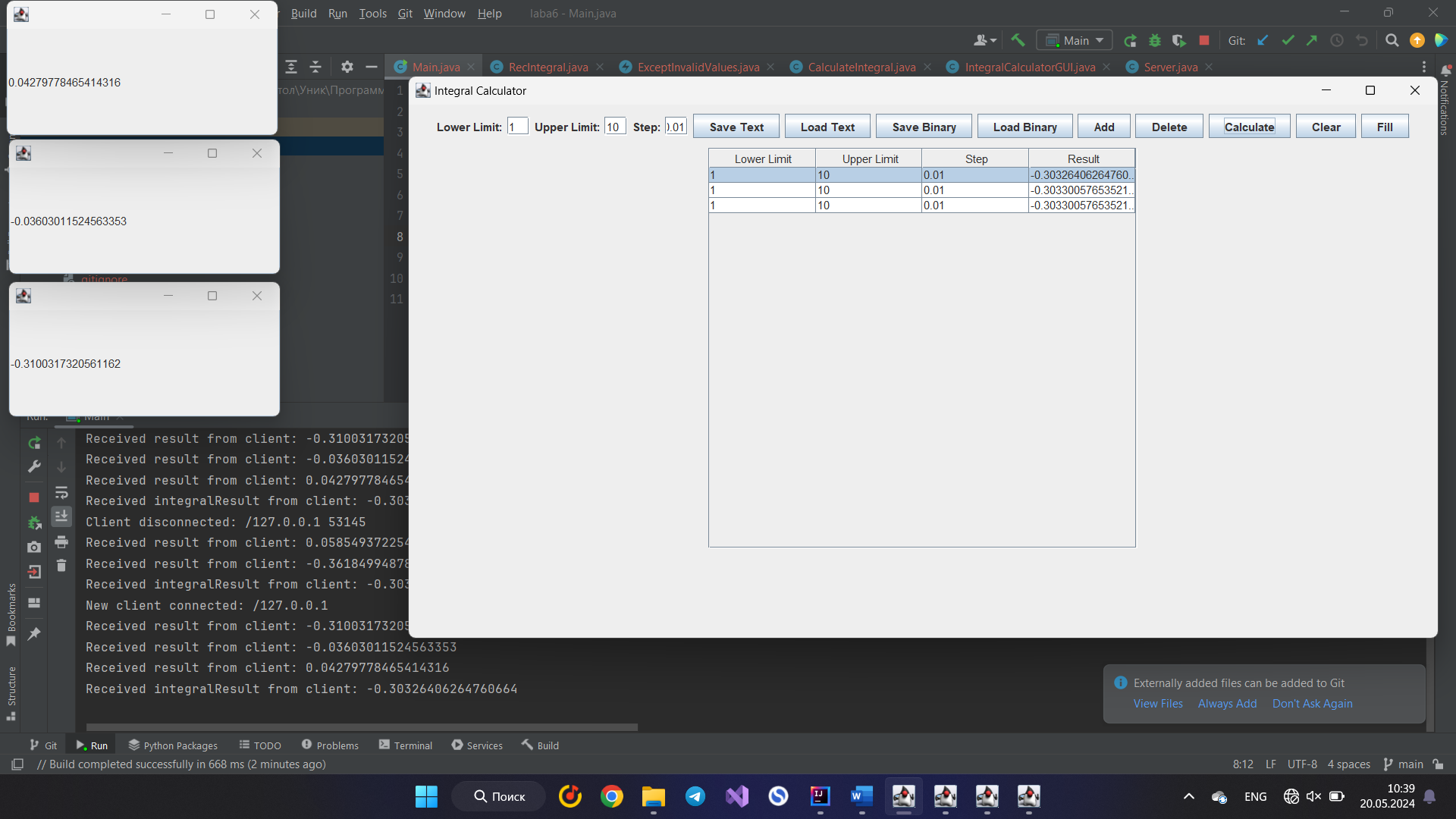
Карамышева Н.С.

Пенза 2024

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

Вариант – 9 UDP

**Результат работы программы:**

****

**Листинг**

CalculateIntegral.java

import javax.swing.table.DefaultTableModel;

public class CalculateIntegral implements Runnable {

private final double lowerLimit;

private final double upperLimit;

private final double step;

private static double integralResult = 0.0;

public CalculateIntegral(double lowerLimit, double upperLimit, double step){

this.lowerLimit = lowerLimit;

this.upperLimit = upperLimit;

this.step = step;

}

public static double getIntegralResult() {

return integralResult;

}

public static void setIntegralResultNull(){

integralResult = 0.0;

}

@Override

public void run(){

double temp = IntegralCalculatorGUI.calculateIntegral(lowerLimit, upperLimit, step);

integralResult += temp;

System.out.println(temp);

}

}

ExceptInvalidValues.java

public class ExceptInvalidValues extends Exception {

public ExceptInvalidValues(String message) {

super(message);

}

}

RecIntegral.java

class RecIntegral {

private double lowerLimit;

private double upperLimit;

private double step;

public RecIntegral(double lowerLimit, double upperLimit, double step) throws ExceptInvalidValues {

this.lowerLimit = lowerLimit;

this.upperLimit = upperLimit;

this.step = step;

}

public double getLowerLimit() {

return lowerLimit;

}

public double getUpperLimit() {

return upperLimit;

}

public double getStep() {

return step;

}

public void setLowerLimit(String lowerLimit) {

this.lowerLimit = Double.parseDouble(lowerLimit);

}

public void setUpperLimit(String upperLimit) {

this.upperLimit = Double.parseDouble(upperLimit);

}

public void setStep(String step) {

this.step = Double.parseDouble(step);

}

}

IntegralCalculatorGUI.java

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.\*;

import java.util.ArrayList;

import java.util.Scanner;

public class IntegralCalculatorGUI extends JFrame {

public final DefaultTableModel tableModel;

private final JTable table;

private final JTextField lowerLimitField;

private final JTextField upperLimitField;

private final JTextField stepField;

private ArrayList<RecIntegral> integralList;

private Server server;

public IntegralCalculatorGUI() {

server = new Server();

server.start();

setTitle("Integral Calculator");

setSize(800, 600);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

tableModel = new DefaultTableModel();

tableModel.addColumn("Lower Limit");

tableModel.addColumn("Upper Limit");

tableModel.addColumn("Step");

tableModel.addColumn("Result");

table = new JTable(tableModel);

lowerLimitField = new JTextField(2);

upperLimitField = new JTextField(2);

stepField = new JTextField(2);

integralList = new ArrayList<>();

JButton saveTextButton = new JButton("Save Text");

saveTextButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser();

int result = fileChooser.showSaveDialog(null);

if (result == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

// Сохранение данных в текстовом формате в выбранный файл

saveDataAsText(file);

}

}

});

JButton loadTextButton = new JButton("Load Text");

loadTextButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser();

int result = fileChooser.showOpenDialog(null);

if (result == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

// Загрузка данных из выбранного текстового файла

loadDataFromText(file);

}

}

});

JButton saveBinaryButton = new JButton("Save Binary");

saveBinaryButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser();

int result = fileChooser.showSaveDialog(null);

if (result == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

// Сохранение данных в двоичном формате в выбранный файл

saveDataAsBinary(file);

}

}

});

JButton loadBinaryButton = new JButton("Load Binary");

loadBinaryButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser();

int result = fileChooser.showOpenDialog(null);

if (result == JFileChooser.APPROVE\_OPTION) {

File file = fileChooser.getSelectedFile();

// Загрузка данных из выбранного двоичного файла

loadDataFromBinary(file);

}

}

});

JButton addButton = new JButton("Add");

addButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// Создаем объект RecIntegral с заданными значениями и добавляем его в коллекцию

RecIntegral recIntegral = null;

try {

double lowerLimit, upperLimit, step;

lowerLimit = Double.parseDouble(lowerLimitField.getText());

upperLimit = Double.parseDouble(upperLimitField.getText());

step = Double.parseDouble(stepField.getText());

if ((lowerLimit < 0.000001 || lowerLimit > 1000000) ||

(upperLimit < 0.000001 || upperLimit > 1000000) ||

(step < 0.000001 || step > 1000000)) {

throw new ExceptInvalidValues("Values must be in the range from 0.000001 to 1000000");

} else if (upperLimit <= lowerLimit) {

throw new ExceptInvalidValues("Upper limit must be larger than the lower limit");

} else {

tableModel.addRow(new Object[]{lowerLimitField.getText(), upperLimitField.getText(), stepField.getText(), ""});

recIntegral = new RecIntegral(0, 0, 0);

recIntegral.setLowerLimit(lowerLimitField.getText());

recIntegral.setUpperLimit(upperLimitField.getText());

recIntegral.setStep(stepField.getText());

integralList.add(recIntegral);

}

} catch (ExceptInvalidValues ex) {

JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR\_MESSAGE);

//throw new RuntimeException(ex);

}

}

});

JButton deleteButton = new JButton("Delete");

deleteButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int selectedRow = table.getSelectedRow();

if (selectedRow != -1) {

tableModel.removeRow(selectedRow);

integralList.remove(selectedRow);

}

}

});

JButton calculateButton = new JButton("Calculate");

calculateButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

double lowerLimit, upperLimit, step;

int selectedRow = table.getSelectedRow();

lowerLimit = Double.parseDouble(tableModel.getValueAt(selectedRow, 0).toString());

upperLimit = Double.parseDouble(tableModel.getValueAt(selectedRow, 1).toString());

step = Double.parseDouble(tableModel.getValueAt(selectedRow, 2).toString());

double integralResult = server.sendCalculationDataToAll(lowerLimit, upperLimit, step);

if (selectedRow != -1) {

tableModel.setValueAt(integralResult, selectedRow, 3);

}

/\*try {

int selectedRow = table.getSelectedRow();

lowerLimit = Double.parseDouble(tableModel.getValueAt(selectedRow, 0).toString());

upperLimit = Double.parseDouble(tableModel.getValueAt(selectedRow, 1).toString());

step = Double.parseDouble(tableModel.getValueAt(selectedRow, 2).toString());

int numThread = 9;

double stepRun = (upperLimit - lowerLimit) / numThread;

ArrayList<Thread> threadArrayList = new ArrayList<>();

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

double upperLimitTemp = lowerLimit + stepRun;

Thread thread = new Thread(new CalculateIntegral(lowerLimit, upperLimitTemp, step));

threadArrayList.add(thread);

thread.start();

lowerLimit = upperLimitTemp;

}

for (Thread thread : threadArrayList){

thread.join();

}

//double integralResult = calculateIntegral(lowerLimit, upperLimit, step);

if (selectedRow != -1) {

tableModel.setValueAt(CalculateIntegral.getIntegralResult(), selectedRow, 3);

CalculateIntegral.setIntegralResultNull();

}

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(null, "Please enter valid numerical values.");

} catch (InterruptedException ex) {

throw new RuntimeException(ex);

}\*/

}

});

JButton clearButton = new JButton("Clear");

clearButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

tableModel.setRowCount(0); // Очистка таблицы

// integralList.clear(); // Очистка коллекции

}

});

JButton fillButton = new JButton("Fill");

fillButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// Очищаем таблицу перед заполнением

tableModel.setRowCount(0);

// Заполняем таблицу данными из объектов RecIntegral в коллекции integralList

for (RecIntegral integral : integralList) {

tableModel.addRow(new Object[]{integral.getLowerLimit(), integral.getUpperLimit(), integral.getStep()});

}

}

});

JPanel inputPanel = new JPanel();

inputPanel.add(new JLabel("Lower Limit:"));

inputPanel.add(lowerLimitField);

inputPanel.add(new JLabel("Upper Limit:"));

inputPanel.add(upperLimitField);

inputPanel.add(new JLabel("Step:"));

inputPanel.add(stepField);

inputPanel.add(saveTextButton);

inputPanel.add(loadTextButton);

inputPanel.add(saveBinaryButton);

inputPanel.add(loadBinaryButton);

inputPanel.add(addButton);

inputPanel.add(deleteButton);

inputPanel.add(calculateButton);

inputPanel.add(clearButton);

inputPanel.add(fillButton);

JPanel mainPanel = new JPanel();

mainPanel.add(inputPanel);

mainPanel.add(new JScrollPane(table));

getContentPane().add(mainPanel);

}

// Метод сохранения данных в текстовом формате

private void saveDataAsText(File file) {

try (PrintWriter writer = new PrintWriter(file)) {

for (RecIntegral integral : integralList) {

writer.println(integral.getLowerLimit() + "," + integral.getUpperLimit() + "," + integral.getStep());

}

} catch (FileNotFoundException ex) {

ex.printStackTrace();

}

}

// Метод загрузки данных из текстового файла

private void loadDataFromText(File file) {

try (Scanner scanner = new Scanner(file)) {

// Очищаем таблицу перед заполнением

integralList.clear();

tableModel.setRowCount(0);

while (scanner.hasNextLine()) {

String[] data = scanner.nextLine().split(",");

double lowerLimit = Double.parseDouble(data[0].trim());

double upperLimit = Double.parseDouble(data[1].trim());

double step = Double.parseDouble(data[2].trim());

RecIntegral recIntegral = new RecIntegral(lowerLimit, upperLimit, step);

integralList.add(recIntegral);

tableModel.addRow(new Object[]{recIntegral.getLowerLimit(), recIntegral.getUpperLimit(), recIntegral.getStep()});

}

} catch (FileNotFoundException ex) {

ex.printStackTrace();

} catch (ExceptInvalidValues e) {

throw new RuntimeException(e);

}

}

// Метод сохранения данных в двоичном формате

private void saveDataAsBinary(File file) {

try (ObjectOutputStream oos = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(file)))) {

oos.writeObject(integralList);

} catch (IOException ex) {

ex.printStackTrace();

}

}

// Метод загрузки данных из двоичного файла

private void loadDataFromBinary(File file) {

try (ObjectInputStream ois = new ObjectInputStream(new BufferedInputStream(new FileInputStream(file)))) {

//Вариант с очисткой колекции

integralList.clear();

tableModel.setRowCount(0);

integralList = (ArrayList<RecIntegral>) ois.readObject();

for (RecIntegral integral : integralList) {

tableModel.addRow(new Object[]{integral.getLowerLimit(), integral.getUpperLimit(), integral.getStep()});

}

//Вариант без очистки колекции

/\*ArrayList<RecIntegral> temp = (ArrayList<RecIntegral>) ois.readObject();

integralList.addAll(temp);

for (RecIntegral integral : temp) {

tableModel.addRow(new Object[]{integral.getLowerLimit(), integral.getUpperLimit(), integral.getStep()});

}\*/

} catch (IOException | ClassNotFoundException ex) {

ex.printStackTrace();

}

}

public void changeRow(int selectedRow, double integralResult){

if (selectedRow != -1) {

tableModel.setValueAt(integralResult, selectedRow, 3);

}

}

public static double calculateIntegral(double lowerLimit, double upperLimit, double step) {

double x1, x2, sum = 0;

int amountSteps = (int) ((upperLimit - lowerLimit) / step); //округляется в меньшую сторону

x1 = lowerLimit;

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

x2 = x1 + step;

sum += 0.5 \* step \* (Math.cos(x1 \* x1) + Math.cos(x2 \* x2));

x1 = x2;

}

if ((upperLimit - lowerLimit) % step != 0)

sum += 0.5 \* (upperLimit - x1) \* (Math.cos(x1 \* x1) + Math.cos(upperLimit \* upperLimit));

return sum;

}

}

Server.java

import javax.swing.\*;

import java.io.IOException;

import java.net.\*;

import java.util.ArrayList;

import java.util.List;

public class Server extends Thread {

private static final int PORT = 8888;

private List<InetSocketAddress> clients = new ArrayList<>();

private double integralResult = 0.0;

private int counterClients = 0;

@Override

public void run() {

try (DatagramSocket socket = new DatagramSocket(PORT)) {

System.out.println("Server is running...");

byte[] receiveBuffer = new byte[1024];

while (true) {

DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);

socket.receive(receivePacket);

processReceivedData(receivePacket);

}

} catch (IOException e) {

e.printStackTrace();

}

}

private void processReceivedData(DatagramPacket receivePacket) {

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

InetSocketAddress clientSocketAddress = new InetSocketAddress(clientAddress, clientPort);

if (!clients.contains(clientSocketAddress)) clients.add(clientSocketAddress);

String receivedData = new String(receivePacket.getData(), 0, receivePacket.getLength());

if (receivedData.equals("connect")){

System.out.println("New client connected: " + clientAddress);

} else if (receivedData.equals("disconnect")) {

clients.remove(new InetSocketAddress(clientAddress, clientPort));

System.out.println("Client disconnected: " + clientAddress + " " + clientPort);

} else if (receivedData.startsWith("integralResult")){

String[] parts = receivedData.split(",");

double partOfResult = Double.parseDouble(parts[1]);

integralResult += partOfResult;

System.out.println("Received result from client: " + partOfResult);

counterClients--;

if (counterClients == 0){

System.out.println("Received integralResult from client: " + integralResult);

}

}

}

public double sendCalculationDataToAll(double lowerLimit, double upperLimit, double step) {

try (DatagramSocket socket = new DatagramSocket()) {

if (clients.isEmpty()) {

JOptionPane.showMessageDialog(null, "No connected clients", "Message", JOptionPane.INFORMATION\_MESSAGE);

} else {

integralResult = 0.0;

int numClients = clients.size();

double stepClient = (upperLimit - lowerLimit) / numClients;

for (InetSocketAddress clientAddress : clients) {

counterClients++;

double upperLimitTemp = lowerLimit + stepClient;

String message = "integral," + lowerLimit + "," + upperLimitTemp + "," + step;

byte[] sendData = message.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress.getAddress(), clientAddress.getPort());

socket.send(sendPacket);

lowerLimit = upperLimitTemp;

}

while (counterClients != 0){

sleep(1000);

}

return integralResult;

}

} catch (IOException e) {

e.printStackTrace();

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

return 0.0;

}

}

ClientGUI.java

import javax.swing.\*;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.io.IOException;

import java.net.\*;

import java.util.ArrayList;

public class ClientGUI extends JFrame {

private static final int PORT = 8888;

private JTextField integralResultField;

private DatagramSocket socket;

private InetAddress serverAddress;

public ClientGUI() {

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(300, 150);

try {

socket = new DatagramSocket();

serverAddress = InetAddress.getByName("127.0.0.1");

} catch (SocketException | UnknownHostException e) {

e.printStackTrace();

JOptionPane.showMessageDialog(this, "Failed to connect to server.");

return;

}

sendMessage("connect");

addWindowListener(new WindowAdapter() {

@Override

public void windowClosing(WindowEvent e) {

super.windowClosing(e);

sendMessage("disconnect");

}

});

integralResultField = new JTextField("0.0", 10);

integralResultField.setEditable(false);

add(integralResultField);

new Thread(this::receiveMessages).start();

}

private void sendMessage(String message) {

try {

byte[] data = message.getBytes();

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

socket.send(packet);

} catch (IOException e) {

e.printStackTrace();

}

}

private void receiveMessages() {

try {

while (true) {

byte[] buffer = new byte[1024];

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

socket.receive(packet);

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

if (received.startsWith("integral")) {

String[] parts = received.split(",");

double lowerLimit = Double.parseDouble(parts[1]);

double upperLimit = Double.parseDouble(parts[2]);

double step = Double.parseDouble(parts[3]);

int numThread = 9;

double stepRun = (upperLimit - lowerLimit) / numThread;

ArrayList<Thread> threadArrayList = new ArrayList<>();

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

double upperLimitTemp = lowerLimit + stepRun;

Thread thread = new Thread(new CalculateIntegral(lowerLimit, upperLimitTemp, step));

threadArrayList.add(thread);

thread.start();

lowerLimit = upperLimitTemp;

}

for (Thread thread : threadArrayList){

thread.join();

}

double integralResult = CalculateIntegral.getIntegralResult();

CalculateIntegral.setIntegralResultNull();

integralResultField.setText(String.valueOf(integralResult));

sendMessage("integralResult," + integralResult);

}

}

} catch (IOException e) {

e.printStackTrace();

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

ClientGUI client = new ClientGUI();

client.setVisible(true);

});

}

}

**Вывод:** в ходе данной лабораторной работы научились создавать клиент-серверные приложения c использованием стандартных классов Java.