Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Программирование на языке Java»

на тему «Сетевое взаимодействие в Java»

Вариант №5

Выполнили студенты группы 20ВВП2:

Горожанин Я. А.

Скирдова В. М.

Приняли:

Юрова О. В.

Карамышева Н. С.

Пенза 2023

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

**Лабораторное 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, как в предыдущей работе. Сервер не занимается вычислениями, а лишь реализует взаимодействие с пользователем и агрегацию результатов вычислений от клиентов. Нечетные варианты используют протокол UDP, а четные TCP. Оформление лабораторной работы должно быть выполнено в соответствии с требованиями, приведенными в Приложении 2.

**Листинг:**

Сервер

package com.company;  
  
import javax.swing.\*;  
import javax.swing.filechooser.FileNameExtensionFilter;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.\*;  
import java.net.\*;  
import java.util.\*;  
import java.util.List;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
public class Form extends JDialog {  
  
 private JPanel rootPanel;  
 private JButton buttonOK;  
 private JButton buttonCancel;  
  
 private JButton addButton;  
 private JButton delButton;  
 private JButton calkButton;  
 private JButton clearButton;  
 private JButton fillButton;  
  
 private JTextField input1;  
 private JTextField input2;  
 private JTextField input3;  
 private JTable table1;  
 private JButton saveBButton;  
 private JButton saveTButton;  
 private JButton loadTButton;  
 private JButton loadBButton;  
  
 class MyThread extends Thread {  
 int size;  
  
 MyThread(String name, int \_size) {  
 super(name);  
 size = \_size;  
 }  
  
 public void run() {  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 DatagramSocket dsocket = null;  
 try {  
 dsocket = new DatagramSocket(26);  
 } catch (SocketException ex) {  
 Logger.*getLogger*(Form.class.getName()).log(Level.*SEVERE*, null, ex);  
 }  
  
 for (int i = 0; i < size; i++) {  
 byte[] buffer = new byte[256];  
 DatagramPacket request = new DatagramPacket(buffer, buffer.length);  
 try {  
 dsocket.receive(request);  
 String Message = new String(request.getData(), 0, request.getLength());  
 String Resoult = "", Num = "";  
 int j = 0;  
 while (Message.charAt(j) != ' ') {  
 Resoult += Message.charAt(j);  
 j++;  
 }  
 j++;  
  
 while (j != Message.length()) {  
 Num += Message.charAt(j);  
 j++;  
 }  
 model.setValueAt(Double.*parseDouble*(Resoult), Integer.*parseInt*(Num), 4);  
 } catch (IOException ex) {  
 Logger.*getLogger*(Form.class.getName()).log(Level.*SEVERE*, null, ex);  
 }  
 }  
 dsocket.close();  
 }  
 }  
  
 class RecIntegral implements Serializable {  
 String Upper, Lower, Step, Result;  
  
 String getUpper() {  
 return Upper;  
 }  
  
 String getLower() {  
 return Lower;  
 }  
  
 String getStep() {  
 return Step;  
 }  
  
 String getResult() {  
 return Result;  
 }  
  
 void setUpper(String Temp) {  
 this.Upper = Temp;  
 }  
  
 void setLower(String Temp) {  
 this.Lower = Temp;  
 }  
  
 void setStep(String Temp) {  
 this.Step = Temp;  
 }  
  
 void setResult(String Temp) {  
 this.Result = Temp;  
 }  
  
 void setAll(String limUp, String limDown, String step, String result) {  
 this.setUpper(limUp);  
 this.setLower(limDown);  
 this.setStep(step);  
 this.setResult(result);  
 }  
 }  
  
 class MyException extends Exception {  
 String msg;  
  
 MyException(String code) {  
 msg = code;  
 }  
 }  
  
 List<RecIntegral> listA = new ArrayList();  
  
 DatagramSocket socket;  
 InetAddress address;  
  
 public Form() throws SocketException, UnknownHostException {  
  
 socket = new DatagramSocket();  
 address = InetAddress.*getByName*("localhost");  
  
 setContentPane(rootPanel);  
 setModal(true);  
 getRootPane().setDefaultButton(buttonOK);  
  
 createTable();  
  
 addButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent ae) {  
 double limUp, limDown, step;  
 String str\_limUp, str\_limDown, str\_step;  
 str\_limUp = input1.getText();  
 str\_limDown = input2.getText();  
 str\_step = input3.getText();  
  
 try {  
 limUp = Double.*parseDouble*(str\_limUp);  
 limDown = Double.*parseDouble*(str\_limDown);  
 step = Double.*parseDouble*(str\_step);  
 if (limUp < 0.000001 || limUp > 100000)  
 throw new MyException("Неверное значение верхнего предела");  
  
 else if (limDown < 0.000001 || limDown > 100000)  
 throw new MyException("Неверное значение нижнего предела");  
 else if (limDown > limUp)  
 throw new MyException("Нижний предел должен быть меньше верхнего");  
 else if ((limUp - limDown) < step)  
 throw new MyException("Шаг должен быть меньше интервала интегрирования");  
  
 } catch (MyException e) {  
 ShowMsg(e.msg);  
 return;  
 } catch (Exception e) {  
 ShowMsg("Некорректно введены данные");  
 return;  
 }  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 RecIntegral temp = new RecIntegral();  
 temp.setUpper(str\_limUp);  
 temp.setLower(str\_limDown);  
 temp.setStep(str\_step);  
  
 model.addRow(new Object[]{model.getRowCount() + 1, str\_limUp, str\_limDown, str\_step});  
  
 listA.add(temp);  
 input1.setText("");  
 input2.setText("");  
 input3.setText("");  
 UpdateWindow();  
 }  
 });  
  
 delButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 int SelectedRow;  
 try {  
 SelectedRow = table1.getSelectedRow();  
 if (SelectedRow == -1)  
 throw new Exception();  
 } catch (Exception e1) {  
 ShowMsg("Не выбрана строка в таблице ");  
 return;  
 }  
 int RowCount = table1.getRowCount();  
  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 listA.remove(SelectedRow);  
 model.removeRow(SelectedRow);  
 table1.setModel(model);  
 if (SelectedRow == RowCount - 1) {  
 table1.changeSelection(SelectedRow - 1, 0, false, false);  
 } else {  
 table1.changeSelection(SelectedRow, 0, false, false);  
 }  
 UpdateWindow();  
 }  
 });  
  
 calkButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 Vector data = model.getDataVector();  
 MyThread thread = new MyThread("thread", data.size());  
 for (int i = 0; i < data.size(); i++) {  
 byte[] buf;  
 Vector CurrentData = (Vector) data.get(i);  
 String message = CurrentData.get(1) + " " + CurrentData.get(2) + " " + CurrentData.get(3) + " " + i;  
 buf = message.getBytes();  
 DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 17);  
 try {  
 socket.send(packet);  
 } catch (IOException ex) {  
 Logger.*getLogger*(Form.class.getName()).log(Level.*SEVERE*, null, ex);  
 }  
 }  
 thread.start();  
 UpdateWindow();  
 }  
 });  
  
 fillButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 RecIntegral temp;  
  
 for (RecIntegral recIntegral : listA) {  
 temp = recIntegral;  
 model.addRow(new Object[]{model.getRowCount() + 1, recIntegral.getUpper(), recIntegral.getLower(), recIntegral.getStep(), recIntegral.getResult()});  
 }  
 listA.addAll(listA);  
 UpdateWindow();  
 }  
 });  
  
 clearButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 try {  
 if (table1.getRowCount() == 0)  
 throw new Exception();  
 } catch (Exception e1) {  
 ShowMsg("Таблица пуста");  
 return;  
 }  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 while (model.getRowCount() != 0) {  
 model.removeRow(0);  
 }  
 UpdateWindow();  
 }  
 });  
  
 saveTButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 JFileChooser fc = new JFileChooser();  
 fc.setDialogTitle("Сохранение в текстовом виде");  
 fc.setFileFilter(new FileNameExtensionFilter("Text Files", "txt"));  
 fc.showSaveDialog(null);  
 File f = fc.getSelectedFile();  
  
 try {  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 int row = model.getRowCount();  
 int col = model.getColumnCount();  
  
 FileWriter fw = new FileWriter(f);  
 for (int i = 0; i < row; i++) {  
 for (int j = 0; j < col; j++) {  
 fw.write(model.getValueAt(i, j).toString());  
 fw.write(" ");  
 }  
 fw.write("\n");  
 }  
 fw.close();  
 ShowMsg("Сохранение прошло успешно");  
 } catch (IOException ioException) {  
 ioException.printStackTrace();  
 ShowMsg("Ошибка при сохранении");  
 }  
 }  
 });  
  
 saveBButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 JFileChooser fc = new JFileChooser();  
 fc.setDialogTitle("Сохранение в двоичном виде");  
 fc.setFileFilter(new FileNameExtensionFilter("Binary Files", "bin"));  
 fc.showSaveDialog(null);  
 File f = fc.getSelectedFile();  
 ArrayList<String> arr = new ArrayList<String>();  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 int row = model.getRowCount();  
 int col = model.getColumnCount();  
  
 for (int i = 0; i < row; i++) {  
 for (int j = 0; j < col; j++) {  
 arr.add(model.getValueAt(i, j).toString());  
 }  
 }  
  
 try (ObjectOutputStream oos = new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(f)))) {  
 oos.writeObject(arr);  
 oos.close();  
 ShowMsg("Сохранение прошло успешно");  
 } catch (IOException ioException) {  
 ioException.printStackTrace();  
 ShowMsg("Ошибка при сохранении");  
 }  
 }  
 });  
  
 loadTButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 JFileChooser fc = new JFileChooser();  
 fc.setFileFilter(new FileNameExtensionFilter("Text Files", "txt"));  
 fc.showOpenDialog(null);  
 File f = fc.getSelectedFile();  
 try {  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 FileReader fr = new FileReader(f);  
 BufferedReader reader = new BufferedReader(fr);  
 String line;  
 String[] split;  
 RecIntegral temp = new RecIntegral();  
 listA.clear();  
 while (model.getRowCount() != 0)  
 model.removeRow(0);  
 while ((line = reader.readLine()) != null) {  
 split = line.split(" ");  
 model.addRow(new Object[]{model.getRowCount() + 1, split[1], split[2], split[3], split[4]});  
 temp.setAll(split[1], split[2], split[3], split[4]);  
 listA.add(temp);  
 }  
 reader.close();  
 fr.close();  
 ShowMsg("Загрузка прошла успешно");  
 UpdateWindow();  
 } catch (IOException ioException) {  
 ioException.printStackTrace();  
 ShowMsg("Ошибка при загрузке");  
 }  
 }  
 });  
  
 loadBButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 JFileChooser fc = new JFileChooser();  
 fc.setFileFilter(new FileNameExtensionFilter("Binary Files", "bin"));  
 fc.showOpenDialog(null);  
 File f = fc.getSelectedFile();  
 try {  
 DefaultTableModel model = (DefaultTableModel) table1.getModel();  
 ObjectInputStream ois = new ObjectInputStream(new BufferedInputStream(new FileInputStream(f)));  
 ArrayList<String> arr = (ArrayList<String>) ois.readObject();  
 ois.close();  
 listA.clear();  
 while (model.getRowCount() != 0)  
 model.removeRow(0);  
 for (int i = 0; i < arr.size(); i += 5) {  
 RecIntegral recint = new RecIntegral();  
 recint.setAll(arr.get(i + 1), arr.get(i + 2), arr.get(i + 3), arr.get(i + 4));  
 model.addRow(new Object[]{model.getRowCount() + 1, arr.get(i + 1), arr.get(i + 2), arr.get(i + 3), arr.get(i + 4)});  
 listA.add(recint);  
 }  
 ShowMsg("Загрузка прошла успешно");  
 UpdateWindow();  
 } catch (IOException | ClassNotFoundException ioException) {  
 ioException.printStackTrace();  
 ShowMsg("Ошибка при загрузке");  
 }  
 }  
 });  
  
 buttonOK.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 onOK();  
 }  
 });  
  
 buttonCancel.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 onCancel();  
 }  
 });  
  
 setDefaultCloseOperation(*DO\_NOTHING\_ON\_CLOSE*);  
  
 addWindowListener(new WindowAdapter() {  
 public void windowClosing(WindowEvent e) {  
 onCancel();  
 }  
 });  
  
 rootPanel.registerKeyboardAction(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 onCancel();  
 }  
 }, KeyStroke.*getKeyStroke*(KeyEvent.*VK\_ESCAPE*, 0), JComponent.*WHEN\_ANCESTOR\_OF\_FOCUSED\_COMPONENT*);  
 }  
  
 private void onOK() {  
 dispose();  
 }  
  
 private void onCancel() {  
 dispose();  
 }  
  
 private void createTable() {  
  
 table1.setModel(new DefaultTableModel(  
 null,  
 new String[]{  
 "#", "Верхняя граница интегрирования", "Нижняя граница интегрирования",  
 "Шаг интегрирования", "Результат"}  
 ) {  
 @Override  
 public boolean isCellEditable(int row, int column) {  
 return column != 4;  
 }  
 });  
 }  
  
 private void ShowMsg(String s) {  
 this.setVisible(true);  
 JOptionPane.*showMessageDialog*(null, s);  
 this.setVisible(true);  
 }  
  
 private void UpdateWindow() {  
 this.setVisible(true);  
 }  
  
 public static void main(String[] args) throws SocketException, UnknownHostException {  
 Form dialog = new Form();  
 dialog.pack();  
 dialog.setVisible(true);  
 System.*exit*(0);  
 }  
}

Клиент

import java.io.IOException;  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
import java.net.SocketException;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
class FunctionIntegral {  
 public double f(double x) {  
 double F = 1 / x;  
 return F;  
 }  
}  
  
class MyThread extends Thread {  
 double limUp;  
 double limDown;  
 double limStep;  
 int num;  
 DatagramSocket socket;  
 InetAddress address;  
  
 MyThread(String name, double \_limUp, double \_limDown, double \_limStep, int \_n, DatagramSocket \_socket, InetAddress \_address) {  
 super(name);  
 limUp = \_limUp;  
 limDown = \_limDown;  
 limStep = \_limStep;  
 num = \_n;  
 socket = \_socket;  
 address = \_address;  
 }  
  
 public void run() {  
 double sum = 0;  
 while (limDown + limStep < limUp) {  
 sum += ((Math.*exp*(-limDown) + Math.*exp*(-(limDown + limStep))) / 2) \* limStep;  
 limDown += limStep;  
 }  
 sum += ((Math.*exp*(-limDown) + Math.*exp*(-limUp)) / 2) \* limStep;  
  
 String message = sum + " " + num;  
 byte[] buf = message.getBytes();  
 DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 26);  
 try {  
 socket.send(packet);  
 System.*out*.print("Ok!\n");  
 } catch (IOException ex) {  
 Logger.*getLogger*(MyThread.class.getName()).log(Level.*SEVERE*, null, ex);  
 }  
 }  
}  
  
public class Client {  
 public static void main(String[] args) throws SocketException, IOException {  
 DatagramSocket socket = new DatagramSocket(17);  
 DatagramSocket socketSend = new DatagramSocket();  
 InetAddress address = InetAddress.*getByName*("localhost");  
 System.*out*.print("waiting\n");  
 while (true) {  
 byte[] buffer = new byte[256];  
 DatagramPacket request = new DatagramPacket(buffer, buffer.length);  
 socket.receive(request);  
 if (request.getLength() != 0) {  
 String Message = new String(request.getData(), 0, request.getLength());  
 String strTop = "",  
 strLower = "",  
 strStep = "",  
 strNum = "";  
  
 int size = Message.length();  
  
 int j = 0;  
 while (Message.charAt(j) != ' ') {  
 strTop += Message.charAt(j);  
 j++;  
 }  
 j++;  
  
 while (Message.charAt(j) != ' ') {  
 strLower += Message.charAt(j);  
 j++;  
 }  
 j++;  
  
 while (Message.charAt(j) != ' ') {  
 strStep += Message.charAt(j);  
 j++;  
 }  
 j++;  
  
 while (j != size) {  
 strNum += Message.charAt(j);  
 j++;  
 }  
  
 MyThread thread = new MyThread("thread", Double.*parseDouble*(strTop), Double.*parseDouble*(strLower), Double.*parseDouble*(strStep), Integer.*parseInt*(strNum), socketSend, address);  
 thread.start();  
 }  
 }  
 }  
}

**Результат выполнения программы:**

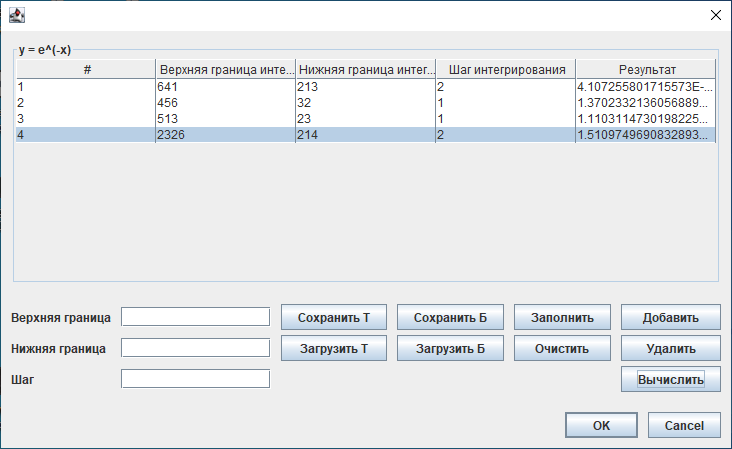


Рисунок 1 — Сервер

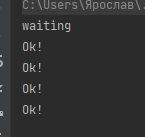


Рисунок 2 — Клиент

**Вывод:** в ходе выполнения данной лабораторной работы были изучены работа с файлами и механизмы сериализации данных.