Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу “Программные на языке Java”

на тему: “ **Сетевое взаимодействие в Java**”

Выполнил студент группы 21ВВП2:

Назаров Е.А.  
Макаров И.С.  
Козлова К.С.

Приняли:

Юрова О.В.

Карамышева Н.С.

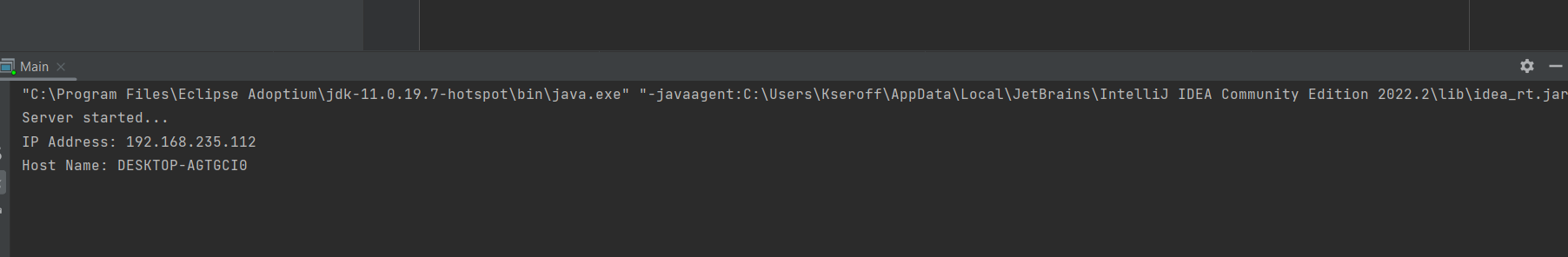
Пенза 2024

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

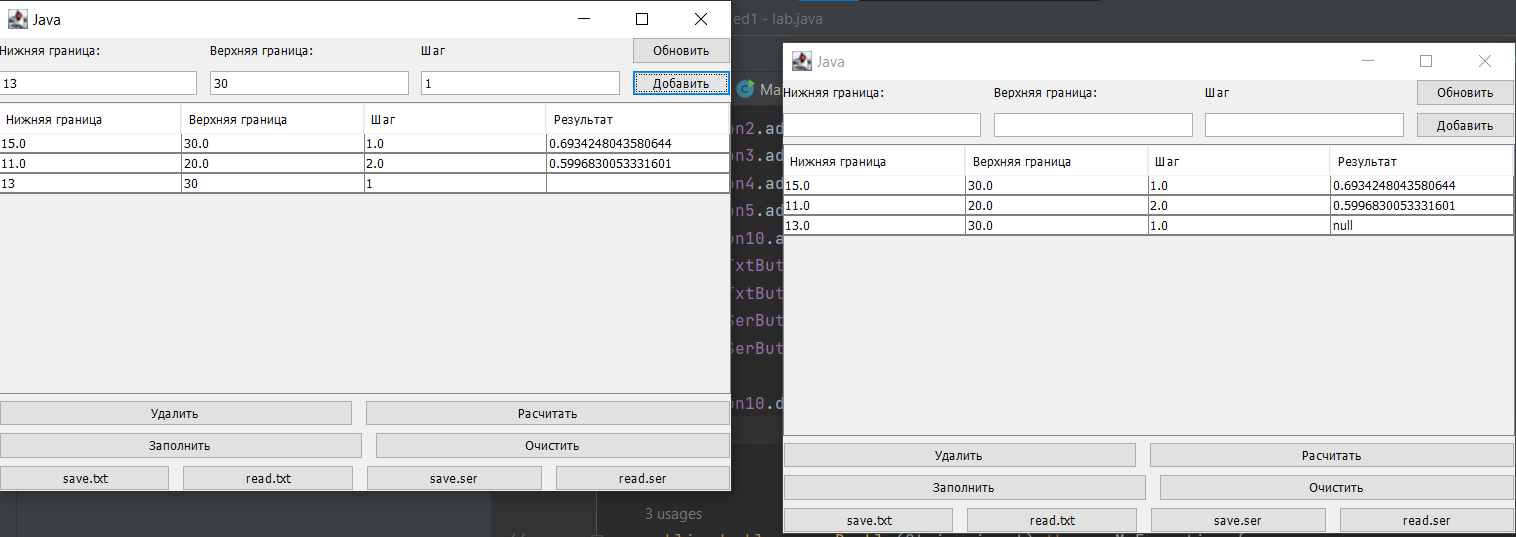
**Вариант: №1**

**Ход работы:**

Запуск сервера:

****

Работа клиента:

****

**Листинг:**

Сервер:

import java.io.\*;  
import java.net.\*;  
import java.util.ArrayList;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class Server {  
 public static class Table {  
 double LowBorder;  
 double HighBorder;  
 double Step;  
 String Result;  
 public Table(double LowBorder, double HighBorder, double Step, String Result) {  
 this.LowBorder = LowBorder;  
 this.HighBorder = HighBorder;  
 this.Step = Step;  
 this.Result = Result;  
 }  
 public Table(double LowBorder, double HighBorder, double Step) {  
 this.LowBorder = LowBorder;  
 this.HighBorder = HighBorder;  
 this.Step = Step;  
 this.Result = null;  
 }  
 }  
 private static final int *PORT* = 12345;  
 private static final int *BUFFER\_SIZE* = 1024;  
 private static ArrayList<Table> *table* = new ArrayList<>();  
  
 public static void startServer(String[] args) {  
 try {  
 *loadTableFromFile*(); // Load table from file  
 DatagramSocket socket = new DatagramSocket(*PORT*);  
 ExecutorService executor = Executors.*newFixedThreadPool*(10);  
  
 System.*out*.println("Server started...");  
  
 try {  
 InetAddress inetAddress = InetAddress.*getLocalHost*();  
 System.*out*.println("IP Address: " + inetAddress.getHostAddress());  
 System.*out*.println("Host Name: " + inetAddress.getHostName());  
 } catch (UnknownHostException e) {  
 e.printStackTrace();  
 }  
  
 while (true) {  
 byte[] buffer = new byte[*BUFFER\_SIZE*];  
 DatagramPacket packet = new DatagramPacket(buffer, buffer.length);  
 socket.receive(packet);  
  
 String request = new String(packet.getData(), 0, packet.getLength());  
 InetAddress clientAddress = packet.getAddress();  
 int clientPort = packet.getPort();  
  
 executor.execute(new RequestHandler(socket, request, clientAddress, clientPort));  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 private static void loadTableFromFile() {  
 File file = new File("table.txt");  
 if (!file.exists()) {  
 try {  
 file.createNewFile();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(file))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 String[] parts = line.split(",");  
 double lowBorder = Double.parseDouble(parts[0]);  
 double highBorder = Double.parseDouble(parts[1]);  
 double step = Double.parseDouble(parts[2]);  
 String result = parts[3];  
 Table tableEntry = new Table(lowBorder, highBorder, step, result);  
 table.add(tableEntry);  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static void saveTableToFile() {  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter("table.txt"))) {  
 for (Table entry : table) {  
 writer.write(entry.LowBorder + "," + entry.HighBorder + "," + entry.Step + "," + entry.Result);  
 writer.newLine();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 static class RequestHandler implements Runnable {  
 private DatagramSocket socket;  
 private String request;  
 private InetAddress clientAddress;  
 private int clientPort;  
  
 public RequestHandler(DatagramSocket socket, String request, InetAddress clientAddress, int clientPort) {  
 this.socket = socket;  
 this.request = request;  
 this.clientAddress = clientAddress;  
 this.clientPort = clientPort;  
 }  
 @Override  
 public void run() {  
 String[] parts = request.split(",");  
 String response = "";  
  
 switch (parts[0]) {  
 case "ADD":  
 double lowBorder = Double.*parseDouble*(parts[1]);  
 double highBorder = Double.*parseDouble*(parts[2]);  
 double step = Double.*parseDouble*(parts[3]);  
 Table newTableEntry = new Table(lowBorder, highBorder, step);  
 *table*.add(newTableEntry);  
 *saveTableToFile*();  
 response = "Record added successfully";  
 break;  
 case "DELETE":  
 int index = Integer.*parseInt*(parts[1]);  
 if (index > 0 && index <= *table*.size()) { // Проверяем, что индекс в пределах списка  
 *table*.remove(index - 1); // Уменьшаем индекс на 1, чтобы соответствовать индексации в списке  
 *saveTableToFile*(); // Сохранение изменений в файл  
 response = "Record deleted successfully";  
 } else {  
 response = "Record not found";  
 }  
 break;  
 case "EDIT":  
 int editIndex = Integer.*parseInt*(parts[1]);  
 if (editIndex >= 0 && editIndex < *table*.size()) {  
 double editLowBorder = Double.*parseDouble*(parts[2]);  
 double editHighBorder = Double.*parseDouble*(parts[3]);  
 double editStep = Double.*parseDouble*(parts[4]);  
 String editResult = parts[5];  
 Table editTableEntry = new Table(editLowBorder, editHighBorder, editStep, editResult);  
 *table*.set(editIndex, editTableEntry);  
 *saveTableToFile*();  
 response = "Record edited successfully";  
 } else {  
 response = "Record not found";  
 }  
 break;  
 case "GET":  
 StringBuilder content = new StringBuilder();  
 for (Table entry : table) {  
 content.append(entry.LowBorder).append(",").append(entry.HighBorder).append(",").append(entry.Step).append(",").append(entry.Result).append("\n");  
 }  
 response = content.toString();  
 break;  
 }  
  
 try {  
 byte[] responseData = response.getBytes();  
 DatagramPacket responsePacket = new DatagramPacket(responseData, responseData.length, clientAddress, clientPort);  
 socket.send(responsePacket);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

Клиент:

import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.\*;  
import java.net.\*;  
import java.util.ArrayList;  
  
public class lab extends JFrame {  
 private JTextField input1;  
 private JTextField input2;  
 private JTextField input3;  
 private JTable table;  
 private JButton button1;  
 private JButton button2;  
 private JButton button3;  
 private JButton button4;  
 private JButton button5;  
 private JLabel label1;  
 private JLabel label2;  
 private JLabel label3;  
 private JPanel rootPanel;  
 private JButton saveTxtButton;  
 private JButton readTxtButton;  
 private JButton saveSerButton;  
 private JButton readSerButton;  
 private JButton button10;  
 private static DefaultTableModel *model*;  
 private static ArrayList<Recintegral> *contenttable* = new ArrayList<>();  
 private static final int *SERVER\_PORT* = 12345;  
 private static final String *SERVER\_IP* = "192.168.31.32";  
  
 private DatagramSocket socket;  
 private InetAddress serverAddress;  
  
 public lab() {  
 add(rootPanel);  
 setTitle("Java");  
 setSize(600, 400);  
  
 try {  
 socket = new DatagramSocket();  
 serverAddress = InetAddress.*getByName*(*SERVER\_IP*);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 // Добавляем обработчики событий к кнопкам  
 button1.addActionListener(new Button1EventListener());  
 button2.addActionListener(new Button2EventListener());  
 button3.addActionListener(new Button3EventListener());  
 button4.addActionListener(new Button4EventListener());  
 button5.addActionListener(new Button5EventListener());  
 button10.addActionListener(new Button10EventListener());  
 saveTxtButton.addActionListener(new Button6EventListener());  
 readTxtButton.addActionListener(new Button7EventListener());  
 saveSerButton.addActionListener(new Button8EventListener());  
 readSerButton.addActionListener(new Button9EventListener());  
  
 button10.doClick();  
  
 }  
  
 public double parseDouble(String input) throws MyException {  
 input = input.replace(',', '.');  
 if (input.matches("-?\\d+(\\.\\d+)?")) {  
 double tmp = Double.*parseDouble*(input);  
 if(tmp > 1000000 || tmp< 0.000001)  
 throw new MyException("Выход за диапазон значений");  
 return tmp;  
 } else {  
 throw new MyException("Неверный формат числа");  
 }  
 }  
  
 public boolean IsCorrect(double d1, double d2, double d3) throws MyException {  
 if(d1 >= d2 || d3 >= d2-d1 )  
 throw new MyException("Некорректно заполненные данные");  
 return true;  
 }  
  
 private void createUIComponents() {  
 String[] columns = {"Нижняя граница", "Верхняя граница", "Шаг", "Результат"};  
 String[][] data = {};  
  
 *model* = new DefaultTableModel(data, columns);  
 table = new JTable(*model*);  
 }  
  
 class Button1EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 try {  
 double d1=parseDouble(input1.getText() );  
 double d2=parseDouble(input2.getText() );  
 double d3=parseDouble(input3.getText() );  
 if(IsCorrect(d1,d2,d3)) {  
 Recintegral temp = new Recintegral(d1, d2, d3);  
 *model*.addRow(new Object[]{input1.getText(), input2.getText(), input3.getText()});  
 *contenttable*.add(temp);  
  
 sendRequest("ADD," + input1.getText() + "," + input2.getText() + "," + input3.getText());  
 }  
 }  
 catch (MyException ex) {  
 ex.GetFormMessage();  
 }  
 }  
 }  
  
 class Button2EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 int temp = table.getSelectedRow();  
 if (temp != -1) {  
 *model*.removeRow(temp);  
 *contenttable*.remove(temp);  
  
 // Отправка запроса на сервер для удаления записи  
 sendRequest("DELETE," + (temp + 1)); // Увеличиваем индекс на 1  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Выберите строку для удаления!");  
 }  
 }  
 }  
  
 class Button3EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 // Создаем и запускаем новый поток для выполнения операции по кнопке 3  
 Thread thread = new Thread(new Button3Thread());  
 thread.start();  
 }  
 }  
  
 // Дополнительный поток для выполнения операции по кнопке 3  
 class Button3Thread implements Runnable {  
 public void run() {  
 int n = table.getSelectedRow();  
 if (n != -1 && n < *model*.getRowCount()) {  
 double a = Double.*parseDouble*(*model*.getValueAt(n, 0).toString());  
 double b = Double.*parseDouble*(*model*.getValueAt(n, 1).toString());  
 double h = Double.*parseDouble*(*model*.getValueAt(n, 2).toString());  
 String Res = String.*valueOf*(Level1.*Trap*(a, b, h));  
 *model*.setValueAt(Res, table.getSelectedRow(), 3);  
 Recintegral temp = new Recintegral(a, b, h, Res);  
 *contenttable*.set(n, temp);  
 if (*contenttable*.size() > n) {  
 *contenttable*.set(n, temp);  
 } else {  
 *contenttable*.add(temp);  
 }  
  
 // Отправка запроса на сервер для редактирования записи  
 sendRequest("EDIT," + n + "," + a + "," + b + "," + h + "," + Res);  
 } else {  
 JOptionPane.*showMessageDialog*(null, "Выберите строку для редактирования!");  
 }  
 }  
 }  
  
 class Button4EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 *model*.setRowCount(0);  
 for (Recintegral con:*contenttable*){  
 if (con.Result==null)  
 *model*.addRow(new Object[]{con.LowBorder,con.HighBorder,con.Step});  
 else  
 *model*.addRow(new Object[]{con.LowBorder,con.HighBorder,con.Step,con.Result});  
 }  
 }  
 }  
  
 class Button5EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 *model*.setRowCount(0);  
 }  
 }  
  
 class Button6EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 Serializate.*saveToFileTxt*(*contenttable*);  
 }  
 }  
  
 class Button7EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 Serializate.*loadFromFileTxt*(*contenttable*,*model*);  
 }  
 }  
  
 class Button8EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 Serializate.*saveToFileSer*(*contenttable*);  
 }  
 }  
  
 class Button9EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 Serializate.*loadFromFileSer*(*contenttable*,*model*);  
 }  
 }  
  
 class Button10EventListener implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 // Отправка запроса на сервер для получения информации о таблице  
 sendRequest("GET");  
 }  
 }  
  
 private void sendRequest(String request) {  
 try {  
 byte[] requestData = request.getBytes();  
 DatagramPacket packet = new DatagramPacket(requestData, requestData.length, serverAddress, *SERVER\_PORT*);  
 socket.send(packet);  
  
 byte[] responseData = new byte[1024];  
 DatagramPacket responsePacket = new DatagramPacket(responseData, responseData.length);  
 socket.receive(responsePacket);  
  
 String response = new String(responsePacket.getData(), 0, responsePacket.getLength());  
 if(request.equals("GET")) {  
 updateTable(response);  
 }  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
 }  
 private void updateTable(String response) {  
 *model*.setRowCount(0); // Очистка текущей таблицы  
 String[] rows = response.split("\n");  
 for (String row : rows) {  
 String[] columns = row.split(",");  
 if (columns.length == 4) { // Убедитесь, что в строке есть все 4 столбца  
 *model*.addRow(new Object[]{columns[0].trim(), columns[1].trim(), columns[2].trim(), columns[3].trim()});  
 }  
 }  
 }  
  
 public static class Level1 {  
 public static double f(double x) {  
 return 1/x;  
 }  
 public static double Trap(double a, double b, double h){  
 double result = 0;  
 for(double i = a; i < b; i+=h) {  
 if((i+h) < b )  
 result += (*f*(i) + *f*(h+i)) \* h/2;  
 else  
 result += (*f*(i)+*f*(b)) \* (b-i)/2;  
 }  
  
 return result;  
 }  
 };  
  
 static public class Recintegral implements Serializable {  
 double LowBorder;  
 double HighBorder;  
 double Step;  
 String Result;  
  
 public Recintegral(double LowBorder,double HighBorder, double Step, String Result ) {  
 this.LowBorder=LowBorder;  
 this.HighBorder=HighBorder;  
 this.Step=Step;  
 this.Result=Result;  
 }  
  
 public Recintegral(double LowBorder,double HighBorder, double Step) {  
 this.LowBorder=LowBorder;  
 this.HighBorder=HighBorder;  
 this.Step=Step;  
 this.Result=null;  
 }  
 }  
  
 public static void main(String[] args) {  
 lab frame = new lab();  
 frame.setVisible(true);  
 }  
}

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