Министерство науки и образования Российской Федерации  
Пензенский государственный университет  
Кафедра “Вычислительная техника”

**ОТЧЁТ**по лабораторной работе №6  
на тему “Сетевое взаимодействие в Java”  
по курсу “Программирование на языке Java”  
Вариант 3

Выполнили:  
ст. гр. 21ВВП1  
Алёшина А.В.  
Нефёдова Е.Д.  
Сорокина Е.А.

Приняли:  
Юрова О.В.  
Карамышева Н.С.

Пенза 2024

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

**Ход работы:**

1. Добавили проект сервера и описали его

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template

\*/

/\*\*

\*

\* @author 40ush

\*/

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()) { // ÐŸÑ€Ð¾Ð²ÐµÑ€ÑÐµÐ¼, Ñ‡Ñ‚Ð¾ Ð¸Ð½Ð´ÐµÐºÑ Ð² Ð¿Ñ€ÐµÐ´ÐµÐ»Ð°Ñ… ÑÐ¿Ð¸ÑÐºÐ°

table.remove(index - 1); // Ð£Ð¼ÐµÐ½ÑŒÑˆÐ°ÐµÐ¼ Ð¸Ð½Ð´ÐµÐºÑ Ð½Ð° 1, Ñ‡Ñ‚Ð¾Ð±Ñ‹ ÑÐ¾Ð¾Ñ‚Ð²ÐµÑ‚ÑÑ‚Ð²Ð¾Ð²Ð°Ñ‚ÑŒ Ð¸Ð½Ð´ÐµÐºÑÐ°Ñ†Ð¸Ð¸ Ð² ÑÐ¿Ð¸ÑÐºÐµ

saveTableToFile(); // Ð¡Ð¾Ñ…Ñ€Ð°Ð½ÐµÐ½Ð¸Ðµ Ð¸Ð·Ð¼ÐµÐ½ÐµÐ½Ð¸Ð¹ Ð² Ñ„Ð°Ð¹Ð»

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();

}

}

}

public static void main(String[] args) {

// TODO code application logic here

Server.startServer(args);

}

}

1. В основном проекте добавили кнопку “Get”, при нажатии которой вызывается функция SendRequest, которая по протоколу UDP отправляет запрос на сервер и получает ответ, после которого обновляет данные в таблице посредством вызова функции UpdateTable

private void jButtonGetActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

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) {

DefaultTableModel Table = (DefaultTableModel)jTable1.getModel();

Table.setRowCount(0);

String[] rows = response.split("\n");

for (String row : rows) {

String[] columns = row.split(",");

if (columns.length == 4) {

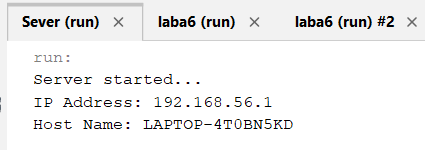
Table.addRow(new Object[]{columns[0].trim(), columns[1].trim(), columns[2].trim(), columns[3].trim()});

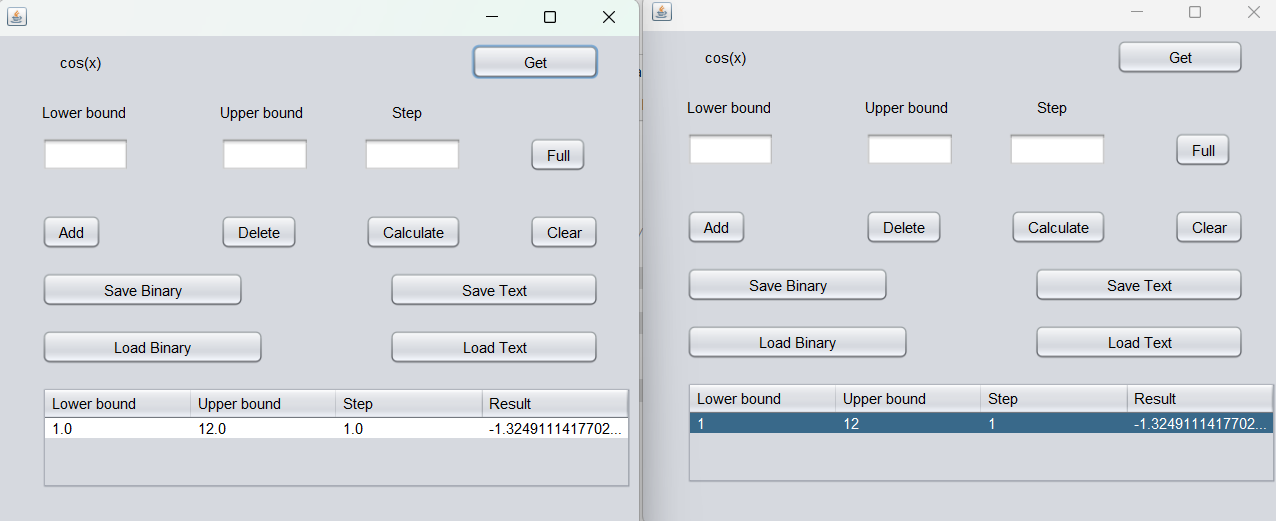
}

}

}

**Результат работы программы:**





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