Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по дисциплине «разработка кроссплатформенных приложений»

на тему «Сетевое взаимодействие в Java»

Вариант №2

Выполнили: студенты группы 22ВВВ1

Дунюшкин В.А.

НиколаевА.А.

Приняла:

Доцент Юрова О.В.

Пенза 2025

**Тема работы:**

Научиться создавать клиент-серверные приложения c использованием стандартных классов Java.

**Ход работы:**

**Описаниe:**

1. Создаём проект сервера, где будут обмениваться данные между клиентами.

public static void main(String[] args) {  
 try (ServerSocket serverSocket = new ServerSocket(*PORT*)) {  
 System.*out*.println("Сервер запущен на порту " + *PORT*);  
  
 while (true) {  
 Socket clientSocket = serverSocket.accept();  
 System.*out*.println("Новое подключение: " + clientSocket);  
  
 ClientHandler clientHandler = new ClientHandler(clientSocket);  
  
 *clients*.put(clientSocket.getRemoteSocketAddress().toString(), clientHandler);  
  
 clientHandler.start();  
 }  
 } catch (IOException e) {  
 System.*err*.println("Ошибка при запуске сервера: " + e.getMessage());  
 }  
}

1. В проекте сервера создаем классы processRequest(), HelpRequest() для работы с запросами

private void processRequest(String request) {  
 String[] parts = request.split(":");  
 if (parts.length < 2) {  
 out.println("ERROR:Неправильный формат");  
 return;  
 }  
  
 String command = parts[0];  
 String mid = parts[1];  
 String high = parts[2];  
 String step = parts[3];  
  
 if (command.equalsIgnoreCase("HELP")) {  
 HelpRequest(mid, high, step);  
 }  
}  
  
private void HelpRequest(String mid, String high, String step) {  
 //другой клиент  
 ClientHandler anotherClient = Main.*getAnotherClient*(clientAddress);  
 if (anotherClient != null) {  
 //кто ожидает ответа  
 anotherClient.waitingResponseFor = this.clientAddress;  
  
 anotherClient.out.println("HELP:" + mid + ":" + high + ":" + step);  
 } else {  
 out.println("ERROR:Нет других подключенных клиентов");  
 }  
}

1. На стороне клиента создали класс для подключения Client()

public class Client {  
 private Socket socket;  
 private PrintWriter out;  
 private BufferedReader in;  
 private ExecutorService executor;  
 private volatile boolean running = true;  
  
 private String lastResponse;  
 private final Object lock = new Object();  
  
 public Client(String serverAddress, int serverPort) throws IOException {  
 socket = new Socket(serverAddress, serverPort);  
 out = new PrintWriter(socket.getOutputStream(), true);  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 executor = Executors.*newSingleThreadExecutor*();  
 System.*out*.println("Подключено к серверу: " + serverAddress + ":" + serverPort);  
 }

1. Также добавили классы Help() и receiveMessages() для отправки запроса и приема сообщений

public String Help(double mid, double high, double step) throws IOException {  
 out.println("HELP:" + mid + ":" + high + ":" + step);  
  
 synchronized (lock) {  
 try {  
 // Ожидаем ответ не более 10 секунд  
 lock.wait(10000);  
  
 if (lastResponse != null) {  
 String response = lastResponse;  
 lastResponse = null; // Сбрасываем после получения  
 return response;  
 }  
 return "ERROR:Таймаут ожидания ответа";  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 return "ERROR:Прервано ожидание ответа";  
 }  
 }  
}  
  
private void receiveMessages() {  
 try {  
 while (running) {  
 String message = in.readLine();  
 if (message != null) {  
 System.*out*.println("Сообщение с сервера: " + message);  
  
 if (message.startsWith("HELP:")) {  
  
 String[] parts = message.split(":");  
 double mid = Double.*parseDouble*(parts[1]);  
 double high = Double.*parseDouble*(parts[2]);  
 double step = Double.*parseDouble*(parts[3]);  
  
 double result = Main\_window.*computeIntegral*(mid, high, step);  
 out.println("RESULT:" + mid + ":" + high + ":" + step + ":" + result);  
 }  
 else if (message.startsWith("RESULT:")) {  
 synchronized (lock) {  
 lastResponse = message;  
 lock.notifyAll();  
 }  
 }  
 }  
 }  
 } catch (IOException e) {  
 if (running) e.printStackTrace();  
 } finally {  
 close();  
 }  
}

**Результат работы программы:**

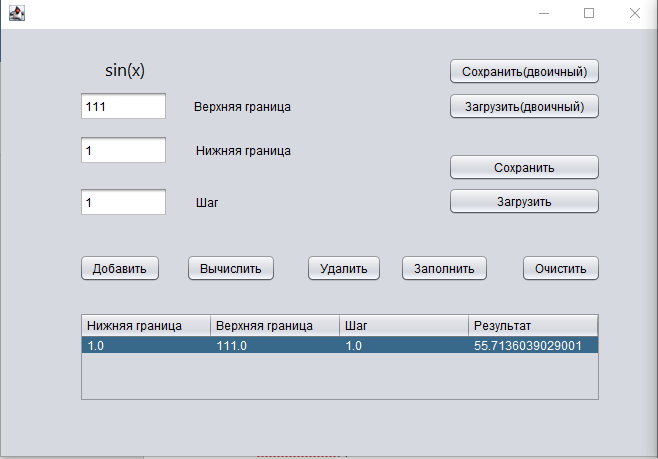
****

Рисунок 1 – Работа клиента номер 1

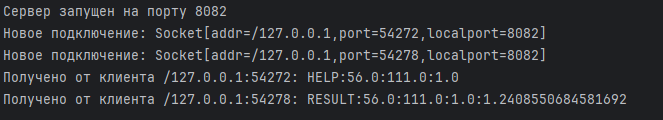


Рисунок 2 – Работа сервера

**Вывод:**

Научились создавать клиент-серверные приложения c использованием стандартных классов Java